Applications of Deep Learning in Healthcare and the Efficient Adaptation of Deep Neural Networks

Phillip S. Wallis
M. S. Applied Mathematics, University of Colorado at Denver

Presented to the
Computer Science and Engineering Education Program
within the Oregon Health & Science University
School of Medicine
in partial fulfillment of
the requirements for the degree
Doctor of Philosophy
in
Computer Science & Engineering

June 2022

Computer Science and Engineering Education Program
School of Medicine
Oregon Health & Science University

CERTIFICATE OF APPROVAL

This is to certify that the Ph. D. dissertation of

Phillip S. Wallis

has been approved.

Xubo Song, Thesis Advisor
Professor

Peter Heeman
Associate Professor

Meysam Asgari
Assistant Professor

Guillaume Thibault
Assistant Professor

Steven Bedrick
Associate Professor

Dedication

I dedicate this work to my amazing family, Meg, Dennie and Murphy, who have supported me throughout this process. Your endless love and support made this possible.

Acknowledgements

I would first like to give special thanks to my amazing thesis advisor, Xubo Song, for taking me on, believing in me, advocating for my work, and providing endless support and encouragement throughout the years. I couldn't have done this without you and you are most appreciated.

I would also like to give extra thanks to Peter Heeman, who has provided tireless support and council, especially throughout the writing of this dissertation. You really went above and beyond to insure my success, and to bring out the best in my work. Your time and effort is greatly appreciated.

Finally, I would like to thank Weizhu Chen from Microsoft, who has been my greatest mentor in industry, and one of my biggest supporters in general. You have helped me grow my understanding of what's possible in real-world AI, and provided me with amazing opportunities to work on state-of-the-art AI applications. I can't thank you enough for all of your encouragement, trust, and support.

Contents

List of Tables

List of Figures

Abstract

Applications of Deep Learning in Healthcare and the Efficient Adaptation of Deep Neural Networks

Phillip S. Wallis

Doctor of Philosophy
Computer Science and Engineering Education Program
within the Oregon Health & Science University
School of Medicine

June 2022
Thesis Advisor: Xubo Song

Applied deep learning is an area that has been evolving rapidly in recent years and is now ubiquitous within the machine learning and artificial intelligence communities. Deep learning has made its way into every part of modern life, from the pictures we take, to the way our devices operate, to the very information we are exposed to. In this work we aim to contribute meaningful innovations to the field of applied deep learning with an emphasis on healthcare applications. In the following chapters we present original research in clinical representation learning, clinical predictive modelling, and efficient adaptation of large neural networks. First, we explore several methods for learning meaningful representations of clinical events. We show that the resulting learned representations contain semantic relationships, and can be leveraged to effectively perform a range of downstream clinical tasks. Next, we propose a deep learning based approach to sleep behavior disorder event detection, which has been correlated with a later onset of neurodegenerative diseases such as Parkinson's and Alzheimer's. Our approach to sleep behavior disorder event detection is both efficient and accurate, with the potential to save countless human hours, healthcare costs, and ideally contribute to higher quality patient care through early detection, and broader accessibility. Lastly, we propose a novel approach to efficient fine-tuning of deep neural networks under parameter budget constraints. There are many issues associated with fine-tuning deep neural networks,

due in part to the shear scale of modern architectures, and compounded by a lack of adequate task-specific training data which is particularly prevalent in healthcare applications. We build upon the fundamental concepts of parameter efficient fine-tuning to present a novel approach in which the allocation of trainable parameters over a network is learned during training, alongside the network weights. We feel that the original work presented in this thesis contributes to the advancement of healthcare AI, and applied deep learning in the following ways: by learning robust, and flexible representations of clinical events we enable the democratization of clinical data, which can in turn empower a range of downstream applications; by providing a effective and efficient machine learning solution to the problem of sleep behavior disorder event detection we empower clinicians to better serve their patients; and by providing a mechanism for efficient, large model adaptation without loss of performance we are lowering the bar for adoption of state of the art deep neural networks, and empowering more advanced, and flexible applications.

Chapter 1

Introduction

A Deep Neural Network (DNN) is a neural network comprising at least three layers. Deep Learning (DL), a sub-field of machine learning (ML) which employs DNNs, has been highly successful across the pillars of modern ML including Computer Vision (CV), Natural Language Processing (NLP), and Reinforcement Learning (RL). DL is responsible for many of the most useful features of tools we use everyday including our cameras, TVs, security systems, appliances and gaming consoles to name a few. Consider the technological advancement that modern DL has given society. Scientific progress empowering individuals to do previously unimaginable things through simple physical interfaces like the touch of a button on a smart phone. Such capabilities would not be possible without the innovative technologies empowering these devices, of which many are rooted in DL.

A differentiating characteristic of DL compared to traditional ML algorithms, and a common theme throughout this dissertation, is the ability of DNNs to learn effective feature representations that generalize well to new and diverse tasks. In traditional ML, features typically need to be identified and/ or engineered by an domain expert before they are useful for ML models. Feature engineering serves to reduce the complexity of raw data, and makes patterns more identifiable to learning algorithms. DNN, on the other hand, naturally learn feature representations. Feature learning in DNNs can be the primary goal of lower network layers, and more complex feature representations can be learned incrementally through many hidden layers which become increasingly more focused. For example, a DNN tasked with image recognition may have shallow layers which only identify light or dark areas within the input image before passing the output on to intermediate layers which detect edges, or lines, followed by layers tasked with shape detection and finally the recognition of actual faces. The ability to learn effective feature representations is one of the biggest advantages of DL, as it serves to mitigate issues inherent to complex feature engineering processes. For example, while domain expertise is invaluable for formulating solutions within any problem space, domain experts may not be able to construct meaningful, complex feature representations up-front that are most useful for ML models. Moreover, by learning feature representations

incrementally from large amounts of data, DL is able to fuel more robust, flexible solutions.

The ability of DNNs to learn feature representations makes them well suited to healthcare tasks. Healthcare data can be notoriously difficult to understand, and by extension, to leverage for real-world applications. Feature engineering for healthcare applications requires a great deal of domain expertise, and continued understanding of the problem space. In addition to learning feature representations, DL solutions typically out-perform other ML methods by a large margin in areas of healthcare such as medical imaging. A good example of the successful application of DL in medical imaging is semantic segmentation [96, 135, 189]. Semantic segmentation refers to a class of problems in CV where the goal is to predict the label for each pixel in an image. As previously mentioned, a DNN tasked with segmenting a region of interest in a medical image is able to incrementally learn useful features such as neurons capable of identifying edges, which are then used to recognize shapes, and finally, to segment the original input image.

One problem in healthcare that we explore in this dissertation is how to leverage a patient's clinical history. We aim to learn representations of clinical events which can in-turn be used in diverse downstream applications, like clinical predictive modelling. Clinical events are represented by collections of standardized medical codes, which are used throughout the healthcare system. While using standardized codes to represent clinical events serves to insure consistency, accuracy, and efficiency, the codes themselves are not able to represent the complex relationships that clinical events have with one another. Moreover, medical codes representing clinical events cannot be used directly in ML models. Past attempts have tried to create various hand-engineered features to represent a patient's clinical history, but the task is not trivial, and poor quality features can lead to poor performance. Hand-engineering features results in solutions which are restricted by the availability and quality of subject matter expertise, and typically result in features which are less expressive, and less useful than features automatically learned by DNNs. It follow that the ability of DNNs to learn effective feature representations can be very useful in this setting.

Another problem in healthcare that we explore in this dissertation is automatic diagnosis of sleep behavior disorder. Such diagnosis are commonly made through manual inspection of multi-channel sensor data collected during overnight sleep studies. Past attempts have used traditional algorithmic approaches which enforce various rule and standards, or statistical features which are hand-engineered from the raw sensor signals. This task is not only very challenging, but often results in systems which are not able to generalize to a representative patient population. DL offers the potential to learn features directly from multi-channel sensor data which are both performant and generalizable, resulting in superior overall systems.

As with many healthcare problems, we struggled with DL's need for massive amounts of data.

In healthcare, it is not uncommon to have very small datasets for training and evaluation. The current trend in the field is to use transfer learning, where a pre-trained model, trained on massive amounts of data, is then adapted to a new task in part to alleviate much of the original high data requirement. High performing pretrained base models are typically very large, which leads to extremely large task specific models that can easily have over 100 million parameters. Large DNNs have high requirement for GPU memory, storage and compute time, making them very difficult to adapt to new tasks, and to deploy in production systems.

To overcome this issue, the field is currently exploring parameter efficient fine-tuning (PEFT) [93, 89, 136], which aims to supplement a large, pretrained base model (DNN) with supplemental parameters that are learned for task adaptation. The problem with these approaches is they require a pre-determined parameter budget, which is distributed uniformly across the network. A more flexible approach would allow for the distribution of parameters to be determined by the network dynamically during training, thereby allowing the network to allocate "help" to areas that need it the most.

1.1 Dissertation Problem and Statement

This dissertation focuses on two problems. The first problem is that many computational healthcare solutions rely on hand-engineered features and/ or large sets of rules. Relying on hand-engineered features results in solutions which are restricted by the availability, and quality of domain expertise. Moreover, it's simply not feasible to consider all possible feature representations in hopes of finding the most useful one, even for the most competent domain expert. By using DL, these features, along with their interactions and combinations, are learned automatically. It follow that DL is capable of capturing underlying semantic relationships, and rich representations that a domain expert might not be aware of, or able to adequately express. In Chapter 3, we learn representations of clinical events that capture semantic similarities between them. Furthermore, we demonstrate that these representations can be used to improve downstream applications, such as clinical prediction. In chapter 4, features are learned directly from multiple channels of wearable sensor data to detect individual abnormal sleep events automatically.

The second problem addressed in this dissertation is with transfer learning in the context of PEFT. The pretrained base models leveraged by modern applications are very well trained on massive amounts of data, and therefore should not require much adaptation in order to perform well on a new task. Chapter 5 presents a method to learn the distribution of trainable parameters in PEFT for optimal performance with the smallest parameter budget. We provide empirical evidence

that our method outperforms PEFT and full FT on a range of common benchmark tasks in CV. We speculate that preserving the base model as much as possible, by updating the smallest number of parameters needed for task adaptation, and allowing the network to allocate these parameters in the most beneficial way, can help to mitigate sequential training issues such as catastrophic forgetting, and enable the most effective level of knowledge transfer from these powerful, underlying base models.

1.2 Learning Representations of Clinical Events

The level of domain expertise required to hand-engineer meaningful features in healthcare effectively raises the bar for development of high-performing clinical models. Alternatively, one could try a "kitchen sink" approach to creating features where one simply adds a feature for every clinical event that could be in a patient medical history, such as diagnosis, procedures and prescriptions. However, the number of combinations required to represent an inclusive set would be much too large for practical use. For example, creating an indicator feature (0/ 1 representation) for every possible event that could be in a patient's clinical history would result in an exponentially large, redundant, and highly sparse feature vector. A high dimensional feature vector also incurs the curse of dimensionality, requiring a huge number of training examples to fit a model.

In contrast, by representing a patient's medical history as a sequence of embedded clinical events, DL systems can be better trained on many downstream tasks such as the next most likely clinical event, or the likelihood of a specific event happening within a certain amount of time (e.g., an emergency room visit in the next week). Effective clinical predictive models allow for early detection of healthcare related events at the patient level, which in-turn allows for more effective prevention, treatment, and design of interventions. We show that learning clinical event embeddings not only provides an effective representation that can be leveraged by downstream applications, but allows for identification of features and relationships that were previously unknown, even to subject matter experts. Individual clinical events can be inherently related to one other, yet ontologies of such events contain no semantic information. Three methods are for learning embeddings from clinical events are showcased in Chapter 3: Event2Vec, autoregressive embedding, and a hybrid approach which combines the two.

1.3 Automatic Event Detection in Multi-Channel Sensor Data

REM sleep disorder, which will be discussed in detail in Chapter 4, is a sleep disorder which has been correlated with a later onset of neurodegeneartive disease such as Alzheimer's and Parkinson's, which are conditions that come with different treatment options if detected early on. The standard way to diagnose RSWA is by identifying events indicative of the condition, contained within many hours of multi-channel sensor data which is annotated by a trained sleep clinician. This process is costly and time consuming, as well as prone to human error. We propose an effective DL solution capable of automatically detecting RSWA events from wearable sensor data without hand-engineering features. Our solution has the potential to save countless human hours as well as healthcare dollars, and widen the availability of treatment to maximize the potential for positive patient outcomes. By detecting individual events, which can be used for diagnosis, we also allow for the duration, magnitude, and frequency of events to be quickly reviewed by clinicians in order to glean additional insights into patients level of severity, thereby empowering clinicians with a more rounded set of assessment tools. Moreover, by automatically detecting individual events we are better able to gain the trust of the clinicians who would use such a tool, as they are given a much more interpretable, verifiable output compared to a system tasked with automatic binary diagnosis of a condition. We evaluate and analyze several DL architectures for automatic RSWA event detection, and show that our approach can not only perform at a level on par with trained human clinicians, but is efficient enough to be deployed at scale.

1.4 Parameter Budget Allocation in PEFT

The common strategy in modern DL is to leverage a large, pretrained DNN, which is then fine-tuned to customize or "adapt" the network to a new task. Fine-tuning (FT) is an example of "transfer learning," which will be discussed in detail in Section 2.7. The result of FT is usually a model capable of out-performing an identical architecture trained from scratch on only task-specific data (i.e., without pretraining). This differentiation is significantly more pronounced when task-specific training data is scarce, as is common in the healthcare setting, where data can be very expensive to procure, and relevant examples are rare. We aim to introduce algorithmic innovations aimed at addressing these issues by efficiently adapting large models to new tasks by updating only a small number of parameters. Specifically, we introduce a novel, innovative contribution to the paradigm of PEFT, which allows for a fixed parameter budget to be optimally distributed across a network

by learning parameter allocation alongside traditional weight updates. We show that our method is capable of outperforming full FT as well as well known PEFT methods across a range of common benchmark tasks. Our method aims to improve upon efficient large model adaptation methods by preserving as much of the powerful base model as possible, and only adapting the areas of the network that need it the most. While the methods proposed in Chapter 5 are evaluated in the domain of CV, they represent a general algorithmic approach which can be applied to a broad set of diverse problems and DNN architectures.

Chapter 2

Background

This chapter is an overview of the material and methods which will be showcased in this thesis. In order to give context around the methods used in later chapters, the next few sections will cover fundamental neural network concepts at a high level; introduce related topics in deep learning such as attention, transfer learning, and mixture of experts; describe relevant architectures including feed-forward, recurrent, convolutional, and transformer networks; and introduce the paradigm of parameter efficient fine-tuning of large scale, deep neural networks.

2.1 Neural Networks

Neural networks, also referred to as artificial neural networks (ANNs) are a subset of ML and represent the meat of the field of deep learning (DL). The basics of neural networks discussed below can be found in a number of textbooks such as [72]. The name "neural networks", as well as their basic structure are inspired by the way in which biological neurons exchange information in the human brain. A neural network can be represented as a weighted, directed graph, where each node is an artificial neuron. Each neuron can receive information from other neurons, and is equipped with an activation function to regulate the information which is passed on. In the next few sections we will discuss some fundamental concepts of neural networks including the idea of an artificial neuron, the methods by which neural networks are optimized, and common neural network architectures.

2.1.1 Perceptron

One of the earliest, foundational ideas of what has evolved into modern neural networks, is the perceptron [150, 67]. Originally proposed as a simple computational model of a biological neuron, the job of a perceptron, as shown in shown in Figure 2.1, is to take a vector of binary inputs $[x_1, x_2, \ldots, x_n]$, and produce a single binary output y. This process is analogous to the "firing" of a

biological neuron, and is accomplished by creating a weighted sum of the inputs, and comparing that sum to a threshold value. A perceptron is therefore parameterized by a set of weights $[w_1, w_2, ..., w_n]$, and a threshold tr, which it uses to determine the value of its binary output y as follows:

$$output = \begin{cases} 0, & \text{if } \sum_i w_i x_i \leq tr \\ 1, & \text{if } \sum_i w_i x_i > tr \end{cases} \tag{2.1}$$

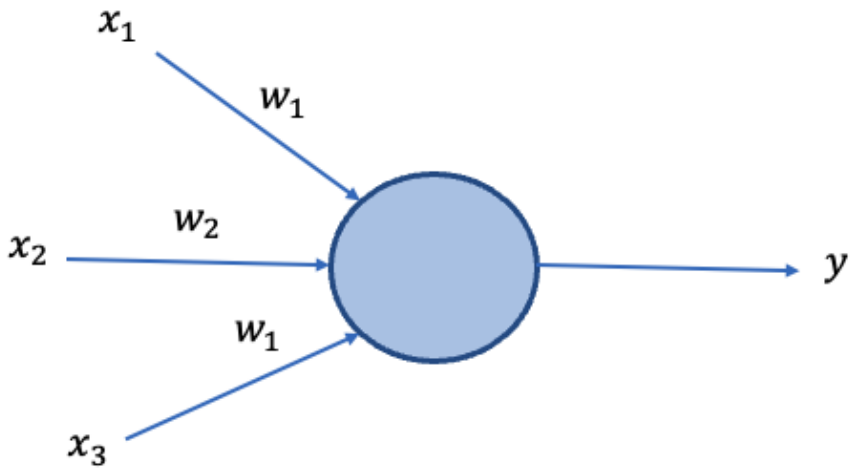


Figure 2.1: The perceptron is a simple artificial neuron parameterized by a set of weights **w** and a threshold th. The output of the perceptron is binary, and "fires" if the weighted combination of inputs $\sum_i w_i x_i$ surpasses the threshold tr.

2.1.2 Multi-Layer Perceptron

A neural network, as the name indicates, is a collection of connected neurons. The neurons in a neural network, as will be seen, are typically organized into groups called layers. It follows that a neural network comprising more than one layer, and utilizing the perceptron definition of a neuron, is referred to as a multi-layer perceptron or MLP. It should be noted that the term MLP is often overloaded within the DL community, and in modern DL, the term is typically associated with a feed forward neural network (FFNN).

FFNNs were first developed in the 1980's and were famously proven to be universal approximators [87]. That is, given some function $f(X)$, there exists a FFNN that can approximately approach the result. This is not to say that a FFNN can be successfully applied to any practical problem. While a FFNN network with a single layer is sufficient to represent any function, the layer may be

infeasibly large and may fail to learn and generalize correctly [72]. A simple FFNN architecture with an input layer, a single hidden layer, and an output layer is shown in Figure 2.2.

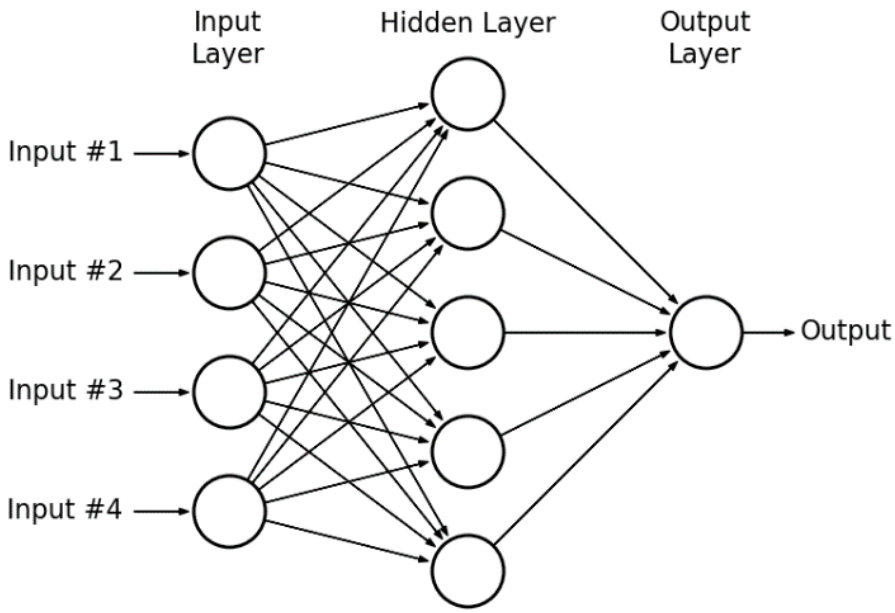


Figure 2.2: A simple FFNN with 1 hidden layer (image from [154]).

Another term which is ubiquitous within the DL community is deep neural networks (DNN), which gives the field of deep learning its "deep," and simply refers to a neural network with more than one hidden layer. In its simplest form, the output from each layer l in a FFNN is a linear transformation of the output from layer $l-1$ passed through an activation function, which is typically, but not always, a non-linear function such as a rectified linear unit (ReLU) [157]. That is:

$$\begin{aligned} \mathbf{z}^l &= \mathbf{w}^l \mathbf{a}^{l-1} + \mathbf{b}^l \quad \text{(linear layer output)} \\ \mathbf{a}^l &= g(\mathbf{z}^l) \quad \text{(“activation” output)} \end{aligned} \tag{2.2}$$

A key design consideration for neural networks is determining the architecture, or overall structure of the network. For example, how many neurons should the network be composed of, how should individual neurons be connected, and how many layers should the network comprise (i.e. network depth). Neural network architectures will commonly organize layers sequentially in a chain structure, where each layer is defined recursively as a function of the layer that preceded it. In this structure, an arbitrary layer l is given by

$$\mathbf{h}^l = g^l(\mathbf{W}^{l\intercal}\mathbf{h}^{\mathbf{l-1}} + \mathbf{b}^l) \tag{2.3}$$

It can be see from Equation 2.3 that the output of layer l, h^l is dependant on the output from layer $l-1$, or h^{l-1}. The input to the first layer (i.e., input layer) are the model inputs $\mathbf{x}$.

The function g is referred to as the "activation" function for the layer, which is typically a nonlinear function such as the Rectified Linear Unit (ReLU), where $ReLU(x) = max(0, x)$ [157], or the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$. An activation function does not have to be nonlinear by definition, but nonlinear activation functions or "nonlinearities" allow for the modeling of non linear problems. There is naturally a trade off between network width (i.e., the number of neurons in a layer), and network depth (i.e., the number of layers in the network). A neural network with a single layer has famously been shown to be a universal approximator [87]. In other words, a single layer neural network, equipped with enough neurons, can be fit perfectly to any data. The caveat is that such a perfect fit equates to extreme overfitting, and therefore very poor generalization to new data. Deeper networks, on the other hand, are often able to employ much narrower layers (i.e., fewer neurons per layer), and by extension far fewer network parameters. In addition, deeper networks frequently show better generalization to new data (i.e., better performance), but also tend to be harder to optimize. Neural networks have many hyperparameters which can be tuned to optimize performance (e.g., number of layers), and each configuration can be computationally expensive, and time consuming, to evaluate via common parameter search methods.

2.1.3 Empirical Risk Minimization

Empirical Risk Minimization (ERM) is a fundamental concept in ML which is well documented in many textbooks [72, 230]. The typical setup of a supervised ML task consists of a domain space $\mathcal{X}$, and a label space $\mathcal{Y}$, along with a model h (hypothesis) mapping the domain space to the label space $h : \mathcal{X} \to \mathcal{Y}$. The model h outputs an object $y \in \mathcal{Y}$ given $x \in \mathcal{X}$, and the goal within the context of ERM is to find h which minimizes error. For example, labeling spam emails can formulated as a classification problem where the objective is to map a vector of features $\mathbf{x}$, extracted from an email, to a value $\hat{y} = p(y|x_1, ..., x_n) \in [0, 1]$, indicating the relative likelihood of being spam. A fundamental assumption in ERM is that there is a join distribution $P(x, y)$ over $\mathcal{X}$ and $\mathcal{Y}$ which is not known, but which can be approximated by an independent and identically distributed ($i.i.d$) sample $\mathbf{S} = [(x_1, y_1), ..., (x_n, y_n)]$ drawn from $P(x, y)$. Let $L(\hat{y}, y)$ be a loss function whose job is to output a measure of the difference between the predicted value $h(x) = \hat{y}$, and the actual value y. The risk R (also referred to as cost) associated with a model h can now be defined as:

$$R(h) = E[L(h(x), y)] = \int L(h(x), y) dP(x, y)$$

As previously mentioned, the true joint distribution $P(x, y)$ is not known, hence the true risk R cannot be computed directly. However, the empirical risk R_{emp}, which is an approximation of the true risk R, can be minimized by taking the average over n training examples. In short, a ML problem can be effectively converted into an optimization problem by minimizing the expected loss on the training set. Essentially, this requires replacing the true distribution $P(x, y)$, which is unknown, with the empirical distribution $\hat{p}(x, y)$, which is defined by the training set. The empirical risk R_{emp} is defined as the average over n training examples.

$$R_{emp}(h) = \frac{1}{n} \sum_{i=1}^{n} L(h(x_i), y_i)$$

A drawback of ERM is that it is prone to overfitting as models with high capacities, such as neural networks, can memorize the training data exactly. In addition, since the data in ERM represents a sample from the true distribution $P(x, y)$, the situation can arise where the empirical error is reduced, but the true error is increased. Furthermore, in many cases ERM is not unfeasible, and most modern optimization algorithms used in DL are based on gradient descent.

2.2 Neural Network Optimization

In DL, as in ML problems in general, the true distribution of the underlying data is not known. As a result the objective, or loss function, to be optimized is represented as an average over examples in the training data, similar to the empirical risk discussed above. There are two key topics which are commonplace in modern neural network optimization: back propagation, and gradient descent.

2.2.1 Back Propagation

The workhorse of DNN optimization is the back propagation algorithm (back-prop). This algorithm was popularized in the 1980s by Rumelhart et al. [191], and is the backbone of all modern iterative methods used in DL, which will be discussed further in Section 2.2.2. The meat of the back-prop algorithm is to calculate $\frac{\partial J}{\partial \theta}$, where J is the overall loss (or "risk" in the context of ERM), and θ are the network parameters (i.e., weights and biases). Back propagation, at a high level, consists of passing data (information) forward through a network, calculating the overall loss J, and then using that value to calculate partial derivatives with respect to each of the weights, all the way back through the network via the chain rule of differentiation.

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x} \tag{2.4}$$

The partial derivative of the loss with respect to a network parameter (i.e., weight or bias) yields the amount by which the overall loss changes when the network parameter changes. Intuitively,

one can think of the partial derivative of the overall loss with respect to the network parameter as the amount of influence that each network parameter has on the overall objective. While back propagation is architecture agnostic within the realm of DL, without loss of generality, and for the purpose of illustration, we will refer to a vanilla FFNN architecture with a single hidden layer for the remainder of this section.

The first step in the back-prop algorithm is to pass information forward through the network and output the associated loss, which is a scalar. Consider the simple, single hidden layer FFNN shown in Figure 2.2. At each layer l, the activation (output) from the previous layer $l-1$ is multiplied by a weight matrix $\mathbf{W}^l$, which is offset by a bias b^l, and in-turn passed through an optional non-linearity. We will use g to denote an arbitrary activation function, which in practice has many choices. The output of the final network layer is then input to a loss function L, which compares the predicted output $\hat{y}$ to the actual output y, and calculates a scalar value. There are many loss functions which are typically suited to certain types of problems (e.g., binary classification vs. regression), but for this example we will use the simple Mean Squared Error $MSE = \frac{1}{2}(y - \hat{y})^2$ which is common in regression problems. The forward pass in this case is derived as follows:

$$\begin{aligned} z^h &= W^h x + b^h \\ a^h &= g(z^h) \\ \hat{y} &= \sigma(W^o a^h + b^o) \\ L(\hat{y}, y) &= \frac{1}{2}(y - \hat{y})^2 \end{aligned} \tag{2.5}$$

We can now update the weights and biases by calculating the gradients with respect to all network parameters (i.e., $\frac{\partial L}{\partial W^l}$ and $\frac{\partial L}{\partial b^l}$ for all layers l) using the chain rule:

$$\begin{aligned} \frac{\partial L}{\partial W^o} &= \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial W^o} = (\hat{y} - y)a^h \\ \frac{\partial L}{\partial b^o} &= \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b^o} = (\hat{y} - y) \\ \frac{\partial L}{\partial b^o} &= \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial a^h} = (\hat{y} - y)W^o \end{aligned} \tag{2.6}$$

$$\begin{aligned} \frac{\partial L}{\partial W^h} &= \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial a^h} \cdot \frac{\partial a^h}{\partial z^h} \cdot \frac{\partial z^h}{\partial W^h} = (\hat{y} - y)W^o g'(z^h)x \\ \frac{\partial L}{\partial b^h} &= \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial a^h} \cdot \frac{\partial a^h}{\partial z^h} \cdot \frac{\partial z^h}{\partial b^h} = (\hat{y} - y)W^o g'(z^h) \end{aligned} \tag{2.7}$$

We have now quantified the degree to which a change in each network parameter will effect the output. We will use these values (gradients) to update the network parameters by iterative taking

a step in the direction of greatest decent, recalculating, and stepping again towards network optimization. This process forms the basis of the iterative optimization algorithm known as gradient descent.

2.2.2 Gradient Descent

DNNs are most commonly optimized via gradient descent based algorithms, as second order algorithms such as Newton's method are not computationally feasible in real-world, high dimensional problems. Gradient descent is guaranteed to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces [190]. There are many flavors of gradient descent, but in this section the focus will be on a select subset including momentum, RMSProp and Adam which are all modifications to mini-batch gradient descent.

Mini-Batch Gradient Descent

Gradient descent uses the partial derivatives calculated by back-prop, along with a fixed learning rate η, to update a network's weights and biases by iteratively moving in the direction of steepest descent. The most common variant used in practice is mini-batch gradient descent, where network parameters are updated for every mini-batch of training examples, each of size n. In mini-batch gradient descent the update rule at each step is given by:

$$\theta = \theta - \eta \nabla_\theta L(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{2.8}$$

Mini-batch updates have been shown to have several advantages over batch gradient descent, where the entire training set is used for each update (i.e. one update per epoch), and stochastic gradient descent where parameters are updated for each training example. One such advantage is more stable convergence due to a reduced variance with respect to parameter updates. Moreover, it is believed that using mini-batches helps to avoid getting stuck in local optima, and makes use of highly optimized matrix optimizations making computation more efficient. However, there are several challenges to this approach as previously mentioned including initial learning rate selection, learning rate scheduling, and difficulty getting out of saddle points (i.e. points where one dimension slopes up and another slopes down). To address these challenges, several modifications to mini-batch gradient descent have been proposed. For the remainder of this section it will be assumed that mini-batches are being used, so the notation $L(\theta; x^{(i:i+n)}; y^{(i:i+n)})$ will be shortened to $L(\theta)$. It should be noted that, in practice, the term stochastic gradient descent (SGD) is commonly used in place of mini-batch gradient descent, even though the formal definition of SGD includes updates

after each example. In the remainder of this section, SGD and mini-batch gradient descent will be used interchangeably.

Momentum

As previously noted, one issue with SGD is navigating portions of the optimization landscape where the surface curves more steeply in one direction than another. These areas, which are local optima, cause SGD to oscillate across the slopes, resulting in a very slow progression towards the local optima. The primary job of momentum [207] is to accelerate SGD in the direction of greatest benefit, and to dampen oscillations. This modification to SGD, which is shown in Figure 2.3, is akin to taking an exponentially weighted moving average and is accomplished as follows: at time step t, momentum takes a portion γ of the previous update at time step $t-1$ and adds it to the current update. In practice, γ is usually set to ≈ 0.9 and η is the learning rate as in SGD.

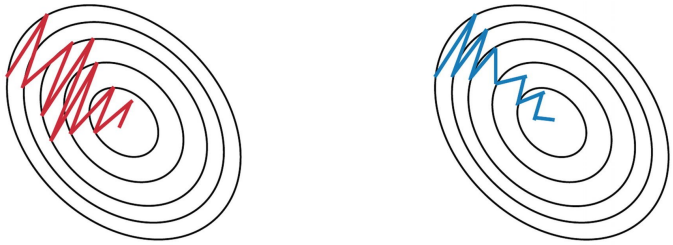


Figure 2.3: SGD without momentum (left), and with momentum (right). The ellipses represent the error surface. Figures adapted from [190].

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta L(\theta) \\ \theta &= \theta - v_t \end{aligned} \tag{2.9}$$

The concept of momentum can be thought of as rolling a ball down a hill. The ball will accelerate in the direction of steepest decline, picking up speed until a terminal velocity is reached. A commonly used modification to SGD with momentum is Nesterov accelerated gradient (NAG) [158]. NAG can be thought of as a way to empower the momentum term with knowledge of what's coming up, in order to better accommodate changes in direction. The term $\theta - \gamma v_{t-1}$ is used to provide an approximation of where the updates are going. This provides a pseudo "look ahead" capability, and calculates the gradient with respect to the approximate future position of the parameters as opposed to the current parameters θ. A NAG update is defined as follows:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta L(\theta - \gamma v_{t-1}) \\ \theta &= \theta - v_t \end{aligned} \tag{2.10}$$

RMSProp

The core contribution of RMSProp is to compute adaptive learning rates for each parameter to allow for variable size updates depending on parameter importance. It was first introduced in a lecture by Geoff Hinton [81], and remains unpublished to this day. RMSProp is a very popular flavor of mini-batch gradient descent due to its simplicity and effectiveness in practice. Moreover, RMSProp solves problems inherent to other adaptive learning rate methods such as Adagrad [53], which will not be discussed in this section, namely rapidly diminishing learning rates. At each iteration of RMSProp, the learning rate is divided by an exponentially decaying average of squared gradients.

$$\begin{aligned} E[g_t^2] &= 0.9E[g^2]_{t-1} + 0.1g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \end{aligned} \tag{2.11}$$

Adam

Like RMSProp, Adaptive Moment Estimation (Adam) proposed by D. Kingma and J. Ba in [116] is also a flavor of mini-batch gradient descent that computes adaptive learning rates for each parameter. Adam stores an exponentially decaying average of past gradients m_t in addition to an exponentially decaying average of past squared gradients v_t which are defined as follows:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \end{aligned} \tag{2.12}$$

As seen from these definitions, the terms m_t and v_t are essentially estimates of the mean and variance of the gradients. In order to correct for some biases related to the initialization of m_t and v_t, as well as choice of β_1 and β_2, Kingma and Ba introduced the following corrections, which are used in place of the original mean and variance estimates:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \end{aligned} \tag{2.13}$$

These modified estimates are then used to construct a final update rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \tag{2.14}$$

The authors propose values of 0.9, 0.999 and 10^{-8} for β_1, β_2 and ϵ respectively.

2.3 ℓ_1 and ℓ_2 Regularization

Regularization is a technique which was introduced in part as a way to obtain results for ill-posed problems, and is commonly used in ML to prevent overfitting [163]. Regularization theory is a broad area of research with an extensive history, but only a small subset of the field will be discussed here, namely ℓ_1 and ℓ_2 [168, 155, 39] regularization, as they are highly relevant to the work which will be presented in Chapter 5. The terms ℓ_1 and ℓ_2 regularization come from that fact that they are based on the ℓ_1 and ℓ_2 norms respectively, which are both specific instances of the more general ℓ_p norm. Let $x = [x_1, ..., x_n]$, then:

$$\ell_p = ||\mathbf{x}||_p = (\sum_{i=1}^{n} |x_i|^p)^{\frac{1}{p}}$$

Therefore

$$\ell_1 = ||\mathbf{x}||_1 = \sum_{i=1}^{n} |x_i| = |x_1| + |x_2| + \cdots + |x_n|$$

and

$$\ell_2 = ||\mathbf{x}||_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2} = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

Both ℓ_1 and ℓ_2 regularization work by adding a penalty term to a model's objective function as follows: Let J be the model's overall objective, L be the loss function, and $\mathbf{w}$ be a weight matrix. For mathematical convenience, when applying ℓ_2 regularization the square root is removed, which does not change the optimization as the result is still monotonic increasing. The new objective function is now:

$$(\ell_1)\ J = \frac{1}{N} \sum_{i=k}^{N} L_i + \lambda ||\mathbf{w}||_1$$

$$(\ell_2)\ J = \frac{1}{N} \sum_{k=1}^{N} L_i + \lambda ||\mathbf{w}||_2^2$$

While ℓ_1 and ℓ_2 regularization both serve to control complexity, ℓ_1 will encourage network parameters (i.e., weights and biases) to be pushed to zero, whereas ℓ_2 will encourage small values, but will rarely push them to 0. This behavior can be inferred from Figure 2.4 which shows a plot of model loss (in red), represented in 2D for the purpose of illustration, along with the ℓ_1 and ℓ_2 constraint functions respectively. The objective of this optimization is to minimize the loss while respecting the regularization constraint. In the case of ℓ_1 the constraint is represented by a diamond. It can be seen from Figure 2.4 that the contours of the loss function will often intersect the the ℓ_1 constraint region at an axis, which results in the corresponding parameter being pushed to 0 (either W_1 or W_2 in this example). In a high dimensional space, as is common in practical

DNNs, many parameters can equal zero simultaneously. It follows that ℓ_1 regularization is better suited to tasks where parameter reduction is desirable since it can remove parameters all together. ℓ_2 on the other hand, is an effective regularization method where model parameter contributions are effectively regulated, but almost never removed completely.

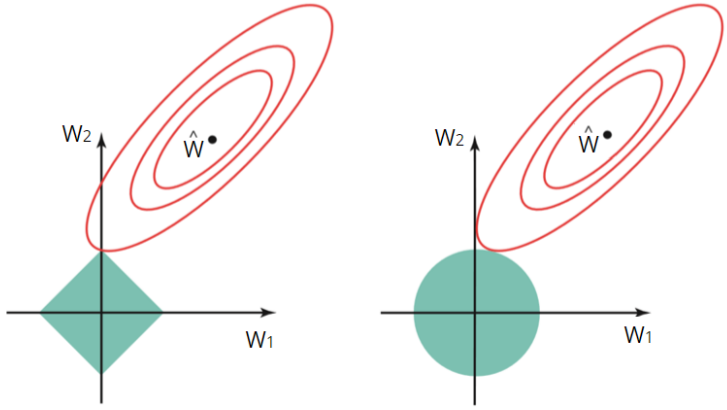


Figure 2.4: ℓ_1 (left) and ℓ_2 (right) regularization. For visualization purposes only two network parameters are assumed, W_1 and W_2. The model loss is represented by the red ellipses, with the ℓ_1 and ℓ_2 constraint functions centered at the origin. Figure taken from [**?**].

2.4 Additional Neural Network Architectures

In this section, the primary architectures used throughout this work will be discussed. While low level details are not elaborated on, the base architectures discussed herein are foundational within DL and are discussed in great detail in many DL texts, such as [72].

2.4.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a family of neural networks designed for processing sequential data (i.e., sequences of values $x^1, x^2, ..., x^n$, where each x^t is an input vector for time step t). A key concept which make RNNs a powerful architecture for sequence problems is internal memory, which allows RNNs to "remember" what it deems important about its input which can then be used to predict what's coming next. The input to a RNN at time step t includes both the input x^t, and the previous state of the network h^t, referred to as the "hidden state", therefore $h^{t+1} = f(\theta^{hh}h^t, \theta^{xh}x^{t+1})$. The previous equation, representing a single time step t in a vanilla RNN, illustrates the recurrence relation responsible for giving the RNN its "R.". Consider the

classical form of a dynamical system, which has the same form as that of a single RNN step:

$$s^t = f(s^{t-1}; \theta) \tag{2.15}$$

This definition is recurrent because the definition of the state of the network s at time t includes the previous state s^{t-1}, and a RNN can be "unfolded" by repeatedly applying this definition. For example, with $t = 3$ Equation 2.15 becomes $s^3 = f(s^2; \theta) = f(f(s^1; \theta); \theta)$. Unfolding the equation like this yields an expression that does not involve recurrence, and can be represented by a traditional acyclic computational graph as can be seen in the unrolled RNN cell shown in Figure 2.5. Unrolling a RNN is meant to provide a better view into how the architecture works, and can be used under the hood by DL frameworks such as pytorch [169] when constructing computation graph representations of a network. The unrolled view of the network also serves to show how RNNs handle sequential inputs, and inputs of different lengths, by taking advantage of parameter sharing over different parts of the network. That is, an unrolled RNN has a step t for each element i in the input sequence, and each step t shares the same weights (**U**, **V** and **W** in Figure 2.5). RNNs are

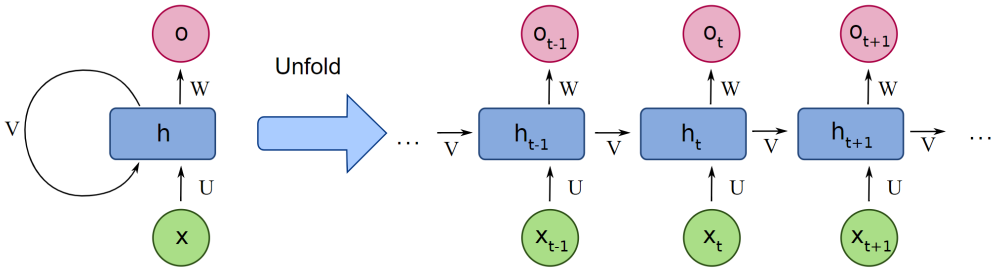


Figure 2.5: Unrolling a RNN cell (image borrowed from [159]).

very versatile and can be successfully applied to many diverse problems involving sequential inputs and / or outputs as eluded to by Figure 2.6. The first sub-figure shows the high level flow of a vanilla FFNN (i.e., one-to-one). The remaining figures show common RNN setups including one-to-many, which is used for problems like image captioning; many-to-one, such as in text classification; and many-to-many as is standard in machine translation and natural language generation.

There are several key issues associated with vanilla RNNs. First, they do not deal well with long term dependencies, i.e., cases where the input is long, and the correct output prediction is dependant on information contained early on in the sequence. Second, RNNs are known to have issues with "exploding" and "vanishing" gradients, which can result from performing a large number of multiplication operations with very small values when back propagating through the unrolled RNN. In practice, the vanilla RNN is typically replaced by a variant of Long Short-Term Memory

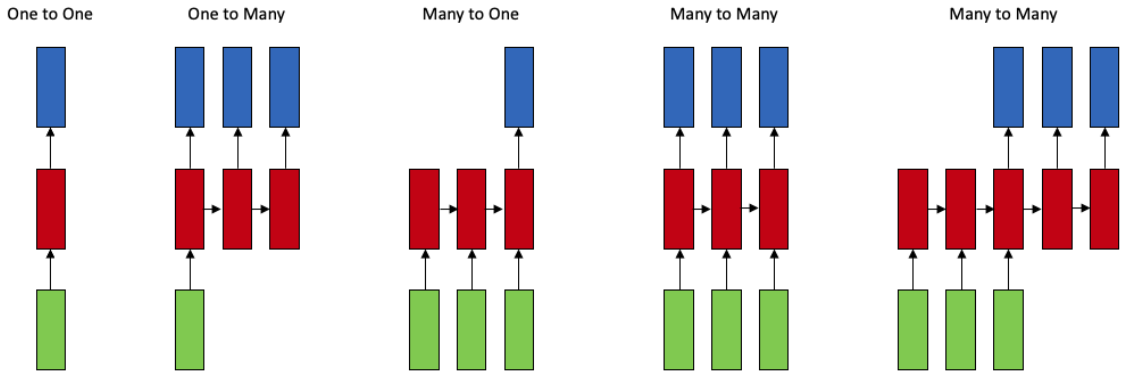


Figure 2.6: FFNN flow (one-to-one) compared to RNN variations: one-to-many, where a single input is used to generate a sequence output (i.e., decoder only); many-to-one, where the input is a sequence and the output is a single prediction (e.g., binary text classification); many-to-many, where the input and output are both sequences.

Networks (LSTM), which were introduced in part to deal with the before mentioned issues.

Long Sort-Term Memory

LSTMs [86, 192, 229] were introduced in the 1990s and are arguably the most popular, and versatile flavor of RNN. In a LSTM, the single RNN recurrent layer is replaced with four layers that interact with each other. The key to LSTMs is cell state and gating mechanisms, which serve to regulate the amount of old information left in, and new information let in. To manage the cell state contribution from past steps, the LSTM employ a forget gate f_t, whose job is to regulate the amount of information from step $t-1$ which is retained in step t. Equipped with the before mentioned gating mechanisms, the cell state can be updated in the following sequential steps: first, the input gate i_t decides how much information should be updated which is subsequently passed though a hyperbolic tangent activation (tanh) that outputs new candidate values $\tilde{C}_t$ which can be added to the cell state. Next, the new cell state C_t is defined as the combination of the old cell state C_{t-1}, regulated by the forget gate f_t, and the new candidate state values $\tilde{C}_t$. Finally, the new cell state is put through another tanh activation to scale the values to $[-1, 1]$, and the output gate o_t is used to regulate what information is passed on (h_t). Under the hood, each "gate" is a separate layer with its own set of parameters W and b, and activation, as can be seen from Equation 2.16. A gate activation is typically a sigmoid function, which "squashes" its input to a value $\in [0, 1]$. This

has the intuitive interpretation of a gate which allows a proportion of the input through.

$$
\begin{aligned}
f_t &= g(W_f[h_{t-1}, x_t] + b_f) \\
i_t &= g(W_i[h_{t-1}, x_t] + b_i) \\
\tilde{C} &= tanh(W_C[h_{t-1}, x_t] + b_C) \\
C_t &= f_t C_{t-1} + i_t \tilde{C}_t \\
o_t &= g(W_o[h_{t-1}, x_t] + b_o) \\
h_t &= o_t tanh(C_t)
\end{aligned}
\tag{2.16}
$$

An example of a generic LSTM cell is shown in figure 2.7. LSTMs are the primary RNN architecture utilized in Chapter 3, and are leveraged for the task of creating embedded representations of clinical events, as well as for clinical predictive modeling. LSTMs are also employed as an event detection method in Chapter 4, where the input consists of multiple channels of time series sensor data collected from overnight sleep studies, and the output is the location of events, and corresponding event type.

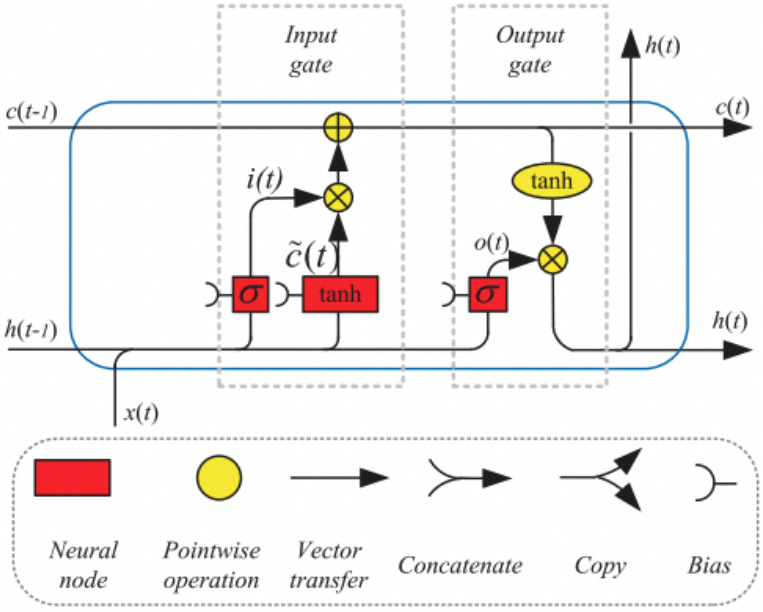


Figure 2.7: Anatomy of an LSTM cell. In this figure the arrow passing through the top portion of the cell represents the cell state from step $t-1$ to step t where $i(t)$, $f(t)$ and $o(t)$ are the input, forget and output gates respectively. Figure borrowed from [229]

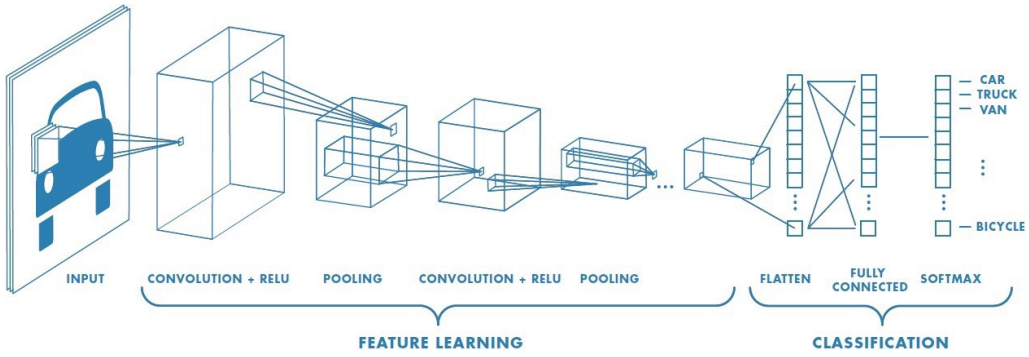


Figure 2.8: 2D CNN for image classification (figure borrowed from [142]).

2.4.2 Convolutional Neural Networks

The first work on modern convolutional neural networks (CNNs) was arguably contributed by Lecun et al in the 1990s [131]. CNNs were introduced in the context of image recognition, and served to improve upon well known shortcomings of densely connected, FFNN applied to images. FFNNs can be an appropriate choice for tasks where the input data consists of examples which all have the same set of predefined features such as with tabular data. While it's quite possible that the patterns hidden within data of this type contain feature interactions, there is no inherent feature interaction structure which is assumed a priori. In contrast, images do have structure which is known a priori, and which can be exploited. Furthermore, consider a typical CV task where the inputs consist of $n \times n$ images where n can realistically be 256, 512, or much larger in the case of high resolution. The resulting input dimensionality of a traditional FFNN can easily surpass 1M in this case, making training infeasible, if not intractable.

CNNs are essentially a specialized neural network variant for processing data that has a known grid-like topology [72]. CNNs, and 1D CNNs in particular, are the primary architecture employed in Chapter 4 for the task of event prediction from multi-channel sensor data. The most common use of CNNs is within the domain of computer vision where inputs are commonly images represented by a 2D grid of pixel intensities, as illustrated in Figure 2.8. CNNs can also accommodate 3D inputs, which are employed for tasks such as semantic segmentation [35], and image classification in medical imaging domains such as radiology. Figure 2.9 shows an example of a 3D CNN used for the medical imaging task of CT-based parametric response mapping for classifying COPD. CNNs have also been successfully applied to time-series problems, where data can be thought of as a 1D grid where samples are taken at evenly spaced time intervals as shown in Figure 2.10.

The architecture of a CNN was meant to mimic connectivity patterns found in the brain, and was inspired by the primate visual cortex [97, 66]. Individual neurons in a CNN respond to stimuli

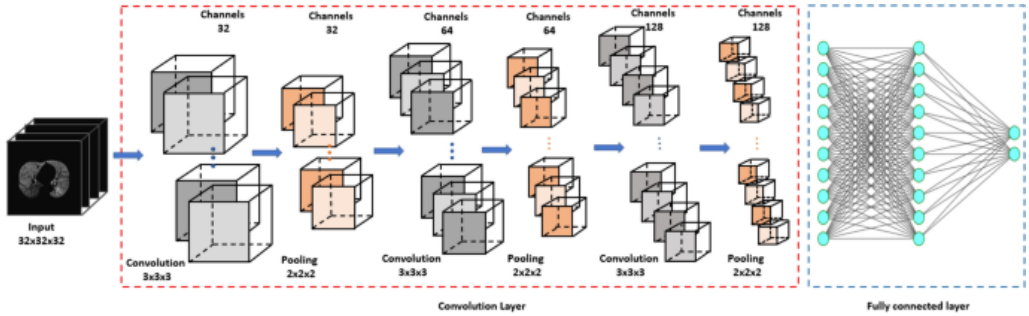


Figure 2.9: A 3D-CNN model with CT-based parametric response mapping for classifying COPD. Figure taken from [84]

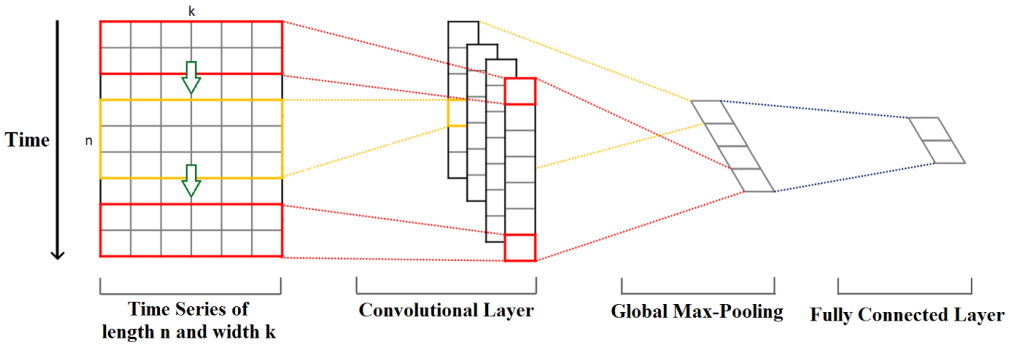


Figure 2.10: A 1D CNN for time series classification. Image taken from [145]

only in a restricted region of the visual field known as the "receptive field." A CNN differs from a FFNN in that it can capture spatial dependencies in an input. This is accomplished via the primary building blocks of CNNs: convolutional layers and pooling layers. CNNs, and CNNs augmented with residual connections, a concept which will be introduced in Section 2.4.3, will be leveraged extensively in Chapter 4.

Convolutional Layers

The convolutional layer [4, 166] is arguably the core building block of a CNN, which comprises a set of relatively small, learnable filters (or kernels, or neurons). For example, a common 2D convolutional filter size is $3 \times 3 \times d$, where d is the depth of the corresponding input. In the 2D case, during the forward pass, each filter is applied across (i.e., slid over) the width and height of the input volume, computing the dot product at each position, thereby producing a 2D activation map with respect to each filter. In other words, each filter is slid over the input from left-to-right,

top-to-bottom, performing a computation between the filter and the input at each position. The output is a 3-dimentional "convolutional volume" of activation maps with a depth $d = n$, where n is the number of filters in the corresponding layer. During training, a CNN learns features that respond to a particular type of stimuli at some spatial position in the input (e.g., edge detection). Convolution in CNNs constrains the network connections such that they innately capture a property known as translational invariance [132], which is essentially the idea that an object can slide around an image while maintaining its identity.

Pooling

Pooling is essentially a kind of non-linear down-sampling that comes in many flavors [228, 162]. While there are many choices of pooling function which can be used in a pooling layer, the commonly used max-pool function will be used in this section for the purpose of illustration. Max-pooling function is one of the most popular pooling choices, and works by simply partitioning an input volume into a set of continuous rectangular regions, and calculating the maximum value with respect to each. The key idea in pooling is that the approximate location of a feature with respect to other features is more important that the exact location of the feature itself. Another job of pooling layers is to progressively reduce the spatial size of the overall representation, which serves to reduce parameters. A max-pool operation only passes on one value for each $n \times m$ region, which translates to a large parameter reduction. Moreover, pooling regions do not commonly overlap, which also contributes to the parameter reduction. That is, if each pooling filter is $n \times n$, then the stride will commonly be n as well. For example, a common choice for filter size in a max-pool is 2×2, which is illustrated in Figure 2.11. It can be seen from Figure 2.11 that a max-pool layer with a 2×2 filter and a stride of 2 will effectively discard 75% of the original input. Parameter reduction also translates to a lower computational overhead, as well as less model complexity, thereby helping to control overfitting. Pooling layers are commonly inserted between successive convolutional layers. A typical "convolutional block" often includes stacks of $a^{(l)} = g(pool(conv(a^{(l-1)}))$ operations, where g is a nonlinearity such as ReLU, and a is a convolutional volume. Pooling contributes to local translation invariance, and commonly operates independently on every slice of the input.

Key Characteristics of a CNN

The following key characteristic of CNNs differentiate them from traditional FFNNs, and make them better suited to inputs such as images:

- **Feature Learning**: The input to a CNN is a grid of values which rarely represent hand engineered features. This is in contrast to FFNNs which commonly use the same set of

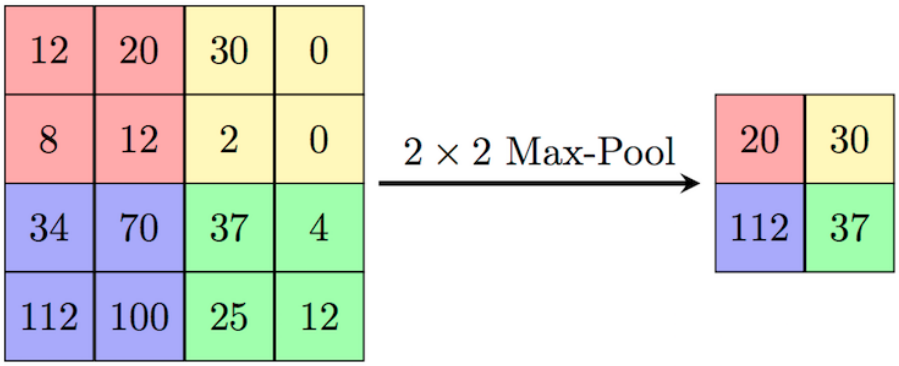


Figure 2.11: A max-pooling layer with 2×2 filter and stride of 2 (image taken from [42]). Each 2×2 colored section on the left corresponds to a portion of the input which is being max-pooled. The right-hand-side shows the output from max-pooling, where each position represents the maximum value of its corresponding section from the left-land-side.

features for each input example, such as with tabular data. Each convolutional filter is a learnable feature with weights that are updated during network optimization.

- **Local Connectivity and Parameter Sharing**: High-dimensional data, such as images, are ill-suited to traditional FFNNs which employ dense connections, making them very computationally expensive given such inputs. Moreover, an architecture which connects each neuron (i.e., filter) in layer l to all neurons in layer $l+1$ is impractical because it does not take into account the spatial structure of the input data. CNNs exploit spatially local correlation by enforcing a sparse local connectivity pattern between neurons of adjacent layers. That is, each neuron, or filter, is connected to only a small region of the input volume, and the same filter weights are shared by all local regions of the input as the filter is convolved, or slid over the entire input to generate an output activation map.

2.4.3 Residual Networks

It seems reasonable to assume that increasing the depth of a DNN will yield better performance. In theory, one can construct a very deep network without loss of performance by simple adding more complexity control via mechanisms such as regularization. In practice however, one of the key issues with increasing the depth of DNNs is that at some point the performance starts degrade, regardless of regularization or others means of controlling model complexity. Residual connections were first introduced in the domain of CV to address this issue, and have been shown empirically to help increase performance of very deep networks [78, 193, 137]. A DNN with residual connections is

known as a residual network or "ResNet." The residual, or "skip" connection allow for information from a previous layer to be combined with the output of a subsequent layer, which results in a more resilient, and robust network. It can be seen from the FFNN description given in Section 2.1.2, that for information to flow from layer $l-2$ to layer l it must pass through layer $l-1$. That is:

$$a^{(l)} = g(z^{(l)}) = g(\theta^{(l)}a^{(l-1)} + b^{(l)}) = g(\theta^{(l)}(g(z^{(l-1)}) + b^{(l)}) = g(\theta^{(l)}(g(\theta^{(l-1)}a^{(l-2)} + b^{(l-1)}) + b^{(l)})$$

Residual connections are sometimes referred to as "short-cut" connections since they allow an output activation $a^{(l-2)}$ from layer $l-2$, to be combined with the output activation $a^{(l)}$ from layer l as shown in Figure 2.12. That is, in a residual layer l, the output is a linear combination of the

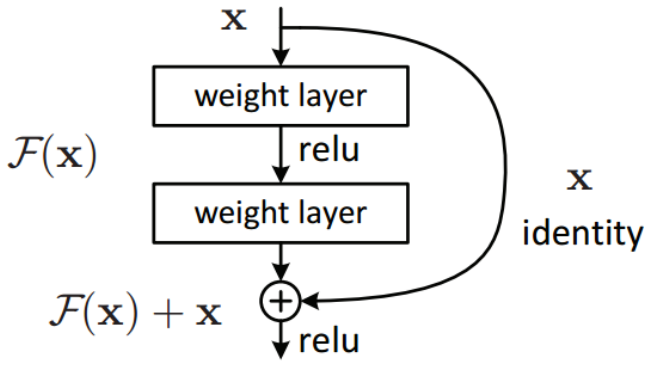


Figure 2.12: Example of the flow of information in a residual connection. Image taken from [78]

output activation from layer l, and the output activation from layer $l-2$.

$$a^{l} = g(z^{(l)} + a^{(l-2)}) = g(\theta^{(l)}a^{(l-1)} + b^{(l)} + a^{(l-2)})$$

An intuitive explanation for why residual connections allow for increased depth without decreased performance is as follows: A DNN with residual connections can essentially learn the identity function (i.e., $g(x) = x$) for all layers beyond a certain depth if they do not benefit to the overall network objective. That is, past a certain depth the additional layers may not help the objective, so the network can simply pass the information forward from that point unchanged without effecting performance. This is a powerful addition to a DNN as it eliminates some of the need to search for an optimal depth up front, which can be costly and time consuming. ResNets in practice have been scaled to extreme depths without the potential for degraded performance, which can be beneficial in many cases [220]. An example of ResNet50, a moderately deep ResNet variant, is shown in Figure 2.13.

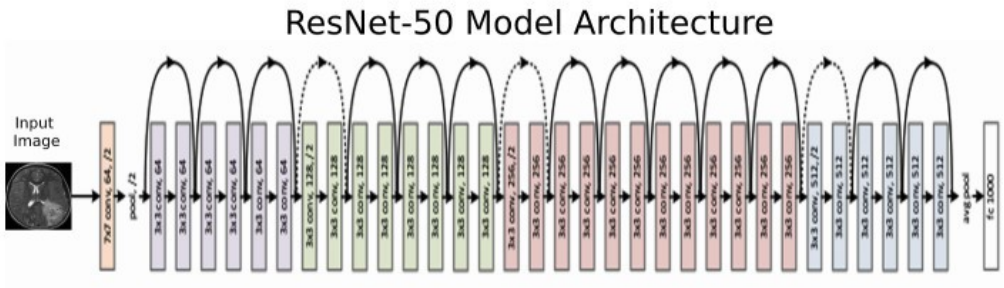


Figure 2.13: ResNet50 architecture applied to brain tumor classification (image taken from [112])

2.5 Attention in Neural Networks

Attention in DNNs [27] is a very important concept, and has been integrated into many modern DNN architectures including attention augmented RNNs [7, 165], and Transformers networks [211] which will be introduced in Section 2.6. Attention in DNNs is a general concept meant to mimic cognitive attention by quantifying the idea of "importance" of an input. Attention in DNNs provides the network with a way to focus on certain subsections of the input, while diminishing others. Attention mechanisms can be added to a range of DNN architectures, and vary in complexity from a simple dot product, to a composition of simple functions, to a full DNN tasked with outputting attention scores. While attention mechanisms vary in complexity, their objective is to quantify the similarity between a set of entities (typically two), which is used as a proxy for the idea of importance. While attention has been integrated into most modern DNN architecture, we will focus the remainder of this section on seq2seq attention and self-attention, which are the flavors of attention most relevant to Chapter 3, and Chapter 5 respectively.

2.5.1 Seq2Seq Attention

Attention was first introduced within the domain of machine translation (MT). In Bahdanau et al., MT is posed as a sequence-to-sequence (seq2seq) task where the base architecture used for modelling is a RNN augmented with attention (seq2seq attention) [7, 174, 37]. Seq2seq attention is also one of the primary methods by which semantic relationships are encoded into vectorized representations of medical events in Chapter 3. While MT is not the only use case for seq2seq attention, it is an effective way to illustrate the concept, and will be used for the remainder of this section. However, in order to introduce seq2seq attention, we must first introduce seq2seq models.

Seq2seq models, in the space of DL, simply refer to DNNs which receive sequential inputs, and produce sequential outputs. For example, in the MT setting, the input and output represent

the same chunk of text in different languages. A key insight from Bahdanau et al. is that seq2seq modelling becomes more difficult as inputs get longer. One intuitive reason for this is that traditionally, the decoder side of the network, which is the side responsible for generating the output, only had access to a single, globally encoded representation of the input at the start of decoding. That is, the network would encode an input sequence into a fixed length vectorized representation, which is in-turn used to start the decoding process. After the first step, the decoder only has access to the previous hidden state at each subsequent step with no memory of the original inputs. Seq2seq attention in RNNs was introduced to address this issue by providing the decoder with a way utilize the inputs during each decoder step, via a weighted combination of the input states. In other words, the decoder has access to the entire input sequence at each decoding step, and can use the seq2seq attention mechanism to focus more attention on elements from the input which are most relevant with respect to the current decoder step.

It is our opinion, although shared by many in the ML community, that seq2seq attention more closely resembles the way that a human would perform a seq2seq task. Consider the MT case, a human translating a long piece of text would likely reference back to the input many times while performing the translation as opposed to simply reading the input, and then translating it accurately without a second glance. The basic idea is as follows: at each decoder step t, we calculate the "relevance" (also referred to as importance, or compatibility) between the decoder hidden state (discussed in Section 2.4.1), and each of the encoder states (i.e., encoded representations of each position). That is, for encoder states $s_1, ..., s_n$, and decoder (hidden) state $h_t = f(x_t, h_{t-1})$, we calculate a $score(h_t, s_k)$ which represents how relevant token k is at decoder step t. Note, as mentioned previously, *score* can be a range of functions that infer similarity or "importance." These scores are then normalized by passing them though a softmax activation:

$$a_k^{(t)} = \frac{e^{score(h_t, s_k)}}{\sum_{i=1}^{n} e^{score(h_t, s_i)}} \quad \forall k \in [1, ..., n]$$

The attention weighted input context for decoder step t can now be given by:

$$c^{(t)} = a_1^{(t)} s_1^{(t)} + \cdots + a_n^{(t)} s_n^{(t)} = \sum_{k=1}^{n} a_k^{(t) s_k}$$

Seq2seq attention in a MT system is shown in Figure 2.14.

A good way to visualize the concept of attention in a seq2seq model is via the problem of text alignment. Alignment, which is generally defined as the task of finding lexicon translation equivalents from a parallel corpus, is a fundamental problem in NLP [19]. While alignment is useful in many scenarios, we will stick with the ongoing example of MT to illustrate this concept. Consider a single sentence, or section of text in two languages A and B. Depending on the languages,

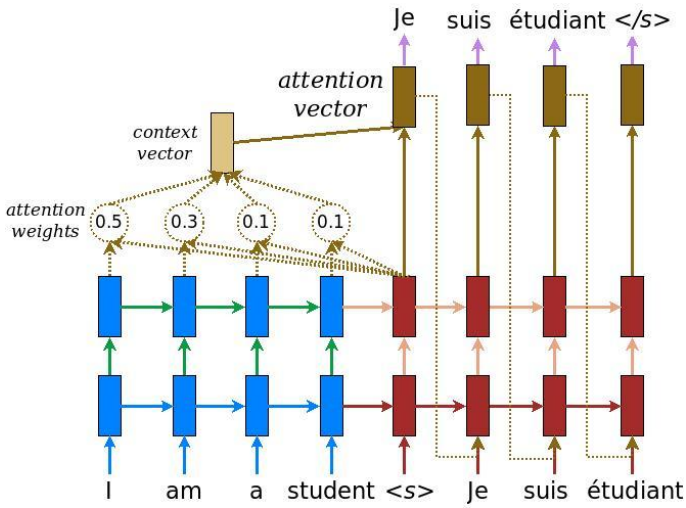


Figure 2.14: An attention mechanism in a seq2seq style RNN used for machine translation. At each decoder step t, the network calculates an attention score for each input with respect to the current output, and creates an attention weighted combination of the inputs in order to make a better prediction (figure taken from [7]).

there are several common scenarios that can arise: first, the individual tokens that make up the input statement in language A do not line up exactly (i.e., word for word) with their counterpart tokens in language B; second, there are multiple tokens in one of the languages which are needed to represent a single token in the other, therefore a single token in language A could be highly important to several tokens in language B. Since attention allows for inputs to be re-weighted by importance with respect to each time step, it can be very effective in reducing poor alignment by putting more weight on a position, or positions in the input which correspond to the current output position. The problem of alignment in MT is visualized in Figure 2.15.

2.5.2 Self-Attention

Self-attention is one of the primary building blocks of the Transformer architecture, which will be introduced in Section 2.6 and is arguably the most important architecture in modern NLP. A primary focus of Chapter 5 is to take Transformers, which have been so successful in NLP, and apply them within the domain of CV.

Consider the following example: "The duck did not swim across the pond because it was too tired". In this example, how does one know that "it" is referring to the duck and not the pond? While this is a trivial task for a human, to a DNN the differentiation is not so straight forward. Self-attention is a flavor of general attention which aims to address this problem by creating a representation of each element of an input sequence as a weighted combination of all the other

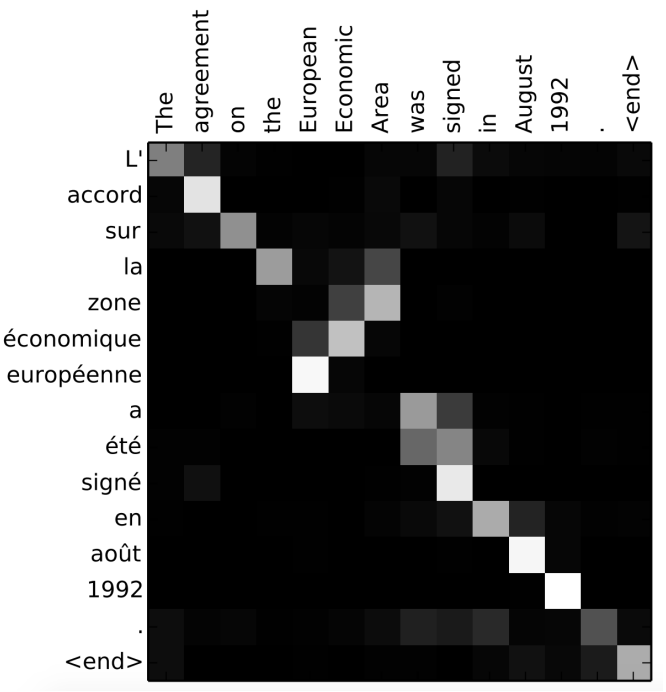


Figure 2.15: The attention map shown text alignment between the same input sentence in two languages (taken from [7]).

elements. That is, as each position in a sequence is processed by the network, self-attention allows the network to look at all other positions in the input sequence for clues that can help lead to a better encoding for this word. The way that this is accomplished in practice is by projecting each element of the input sequence to three vectors denoted as q (query), k (key), and v (value). These vectors are created by multiplying the embedded representation of each input position by three projection matrices that are learned as part of the overall network optimization. The next step is to calculate a score which determines how much focus to place on other parts of the input sequence in order to encode a given position. The score can be calculated by any appropriate compatibility function, as previously discussed in Section 2.5.1, but is traditionally calculated by a simple dot product of the query vector with the key vector of the respective input being scored. That is, for processing self-attention scores for the input at position i in a sequence of length n where $i < n$, the scores would be $< q_i, k_1 >, < q_i, k_2 >, ..., < q_i, k_n >$. These scores are then normalized by passing them through a softmax function and used to weigh the value vectors, which are then aggregated to create a final representation of the input at position i. The flow of an input $\mathbf{X}$ through a self-attention layer is shown in Figure 2.16 and the output from a self-attention layer with respect to a single input from our original example is shown in Figure 2.17..

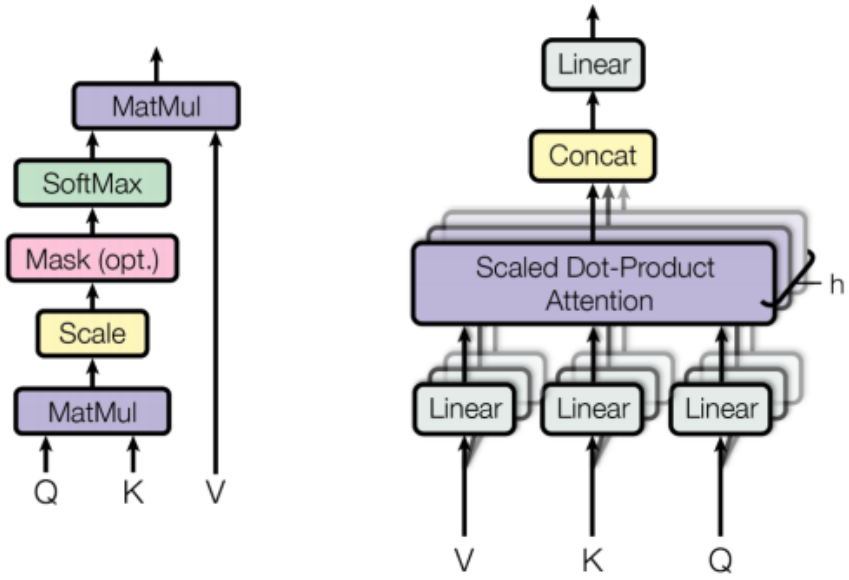


Figure 2.16: A transformer self-attention layer (left) employs three projection matrices which all act on the same input X. The resulting projections Q, K and V are then used to calculate attention weights, which serve to create a representation of each input position based on it's relevance with respect to every input position in the sequence. The right shows the structure of a multi-headed attention layer which applies h separate attention projections to the same input in parallel, and aggregates the outputs (images from [211]).

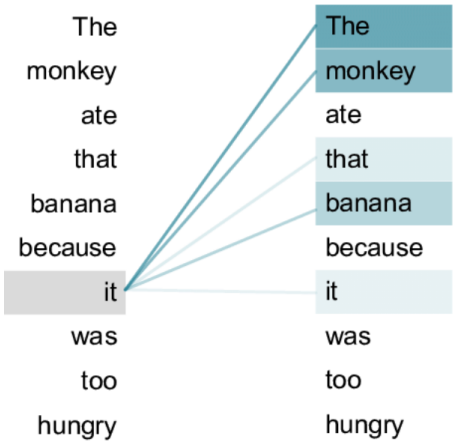


Figure 2.17: The idea of self-attention is to represent each input word ("it" in this example) as a weighted combination of all input words. It can be seen from this example that "the," "monkey" and "banana" are given more importance with respect to the word "it." Higher weighted tokens are deemed more relevant to the token in question and appear darker in this figure, taken from [223]

2.6 Transformer Networks

At its core, a Transformer is a DNN used for processing sequential data. Transformer networks [211], which are the core architecture used in Chapter 5, have seen a great deal of success in recent years across a range of tasks in NLP, as well as other domains with tasks having sequential inputs [138, 114]. Transformers have been noted to have several advantages over RNN based models including their ability to better handle long-term dependencies, thereby allowing for longer input sequences without degraded performance. Moreover, the Transformer does not rely on a recurrence relationship, so the architecture is easily parallelalizable which allows for distributed computation, in particular, GPU accelerated training. One key innovation of Transformer networks which contribute to their ability to better handle long-term dependencies is that they take in an entire sequence simultaneously, as opposed to the step-by-step processing done by RNNs. This is aided by self-attention, which was introduced in Section 2.5.2, and positional encoding, which supplements the embedded representation of each input position with information related to its relative position within the sequence.

Originally introduced in the domain of seq2seq, in particular neural machine translation (NMT), transformers have produced SOTA results across a range of tasks including question answering, summarization, natural language understanding and natural language generation. Since transformer networks were originally introduced to solve problems in the space of seq2seq they were proposed with the encoder-decoder style architecture shown in Figure 2.18. However, the Transformer encoder or decoder can be used as a stand alone architecture, and commonly is, in many applications which will be further discussed below.

While the original Transformer paper showcased an encoder-decoder style architecture, the encoder or decoder network can be used as a stand-alone architecture to learn general, as well as task specific representations of sequential inputs which can then be applied to a range of downstream tasks. A key difference between a Transformer encoder and decoder is in the self-attention layers. In a Transformer encoder, the network is provided access to the entire sequence up front, and can therefore attend to elements on the left or the right of a given element at position i. Self-attention which can look to the left and the right is referred to a "bi-directional" attention. On the other hand, a Transformer decoder is only given the partial sequence up to the element at the current position i, and can therefore only attend to elements to the left. For example, in NMT the encoder is given the entire sequence of text in language A and creates an encoded representation, but the decoder will generate the output in language B one token at a time.

Transformers, like many DNN architectures, are very versatile and model adaptation can be

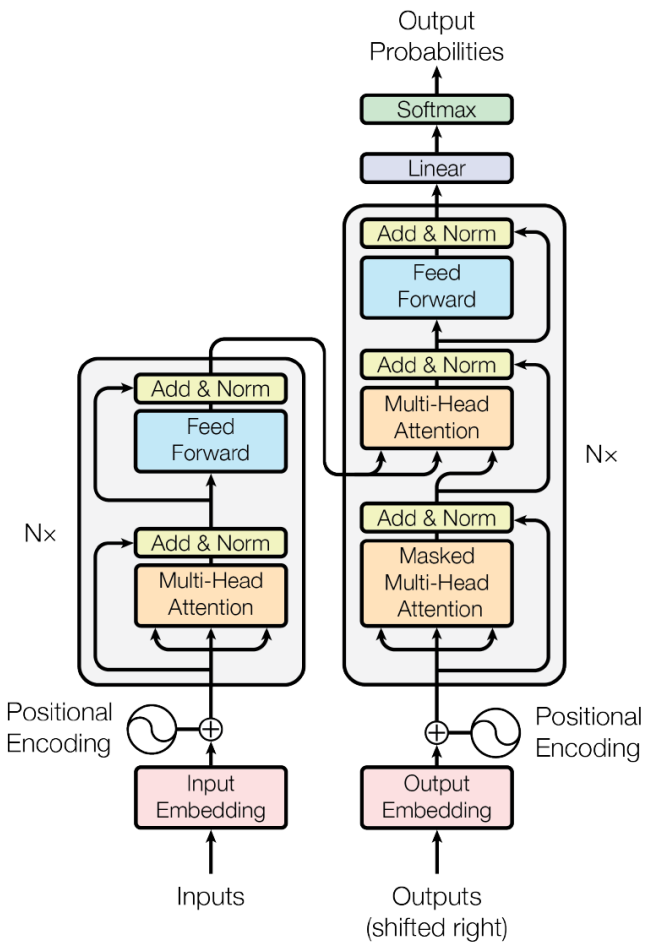


Figure 2.18: Transformer encoder-decoder architecture. This encoder-decoder architecture was proposed in the original Transformer work within the context of machine translation. Image is taken from [211].

done by simply adding task specific "heads" (i.e., sets of supplemental, task-specific layers) to the end of the base Transformer. For example, to construct a document classifier, we could leverage a generally pretrained Transformer encoder and feed the output representation through a set of m additional, task-specific layers, the last of which would output a vector of C class probabilities, where C represents the number of possible document classes. The approach of adapting a pretrained Transformer to a new task became widely popular within the NLP community after the introduction of Bidirectional Encoder Representations from Transformers (BERT) [48]. The original BERT model was pretrained on the multi-task objective of masked language modelling (MLM), and next-sentence prediction. The "bi-directional" part of BERT refers to the fact that the architecture is a Transformer encoder. Therefore each self-attention layer can attend to context to the left and the

right as shown in Figure 2.19. As previously mentioned, the decoder portion of an encoder-decoder

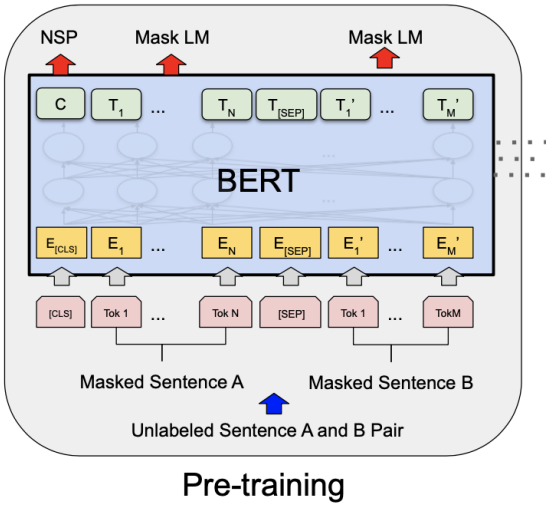


Figure 2.19: BERT encoder only style architecture (image taken from [48]). Attention in this case is bi-directional so can attend to position to the left and the right.

style Transformer can also be leveraged as a stand-alone architecture. The pretraining objective in this case is typically the autoregressive task of next token prediction. Arguably the most well known example of autoregressive transformers is the Generative Pretrained Transformer (GPT) family of models, which includes GPT-2 [179] and GPT-3 [20]. Autoregressive Transformers have been leveraged with a great degree of success within NLG (e.g., chatbots, dialog systems and digital assistants). An example of a Transformer decoder-only block is shown in Figure 2.20.

As previously mentioned, the key difference between a Transformer encoder and decoder based architecture is their respective self-attention layers. The encoder self-attention is bi-directional, and can attend to the left and right since the entire input is accessible by the network. The decoder, on the other hand, is responsible for generating new tokens, and can therefore only see tokens to the left of the current position as tokens to the right have not yet been generated.

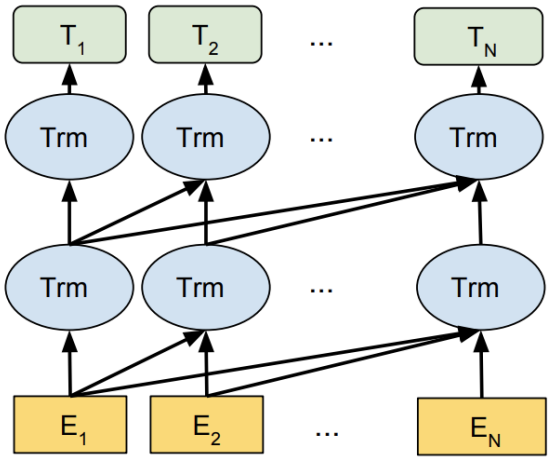


Figure 2.20: GPT2 style decoder only architecture (image taken from [48]). The GPT family of models are autoregressive with a next token prediction objective. Attention is not bi-directional in this case as only inputs to the left can be attended.

2.7 Transfer Learning and Robust Pretraining

A big problem in training DNNs is that they are complex models with a large number of parameters, and can therefore take a huge amount of data to train. Since large amounts of task-specific training data is not always available, we need ways to achieve good performance with less. In recent years it has become increasingly popular to leverage large, pretrained models in order to achieve better performance across many diverse tasks, and often with a shortage of task-specific examples [20, 134, 6]. This practice, commonly referred to as fine-tuning, works because of concepts from transfer learning. At a high level, transfer learning is the idea that the knowledge gained by learning an initial task A can aid in the more effective, and efficient learning of a new task B. This makes sense intuitively as it more closely resembles the way that humans learn. For example, think about learning to stand, and then learning to hop on one leg. Through the process of learning to stand, which is a very general task, we gain knowledge of necessary and fundamental skills such as balance, which making the process of learning to hop on one leg, a task which also requires balance, that much easier. The concept of leveraging a pretrained model to better perform a new task will be a primary focus of Chapter 5, in which large, pretrained DNNs are adapted to new tasks in an efficient manner, where very few parameters are updated during training. Transfer learning is also showcased in Chapter 3, where embeddings are learned from clinical event codes, and the resulting embeddings are used to train a clinical predictive model.

An ideal scenario in ML is that an abundance of labeled training instances not only exists, but

is accessible and has the same distribution as the test data. In practice however, collecting sufficient training data can be expensive, time-consuming, or even unrealistic in many scenarios [234]. The lack of adequate training data is particularly prevalent in healthcare, where data is notoriously scarce due to the rarity of relevant examples, and the cost associated with data collection. The basic flow of transfer learning versus traditional supervised ML is show in Figure 2.21.

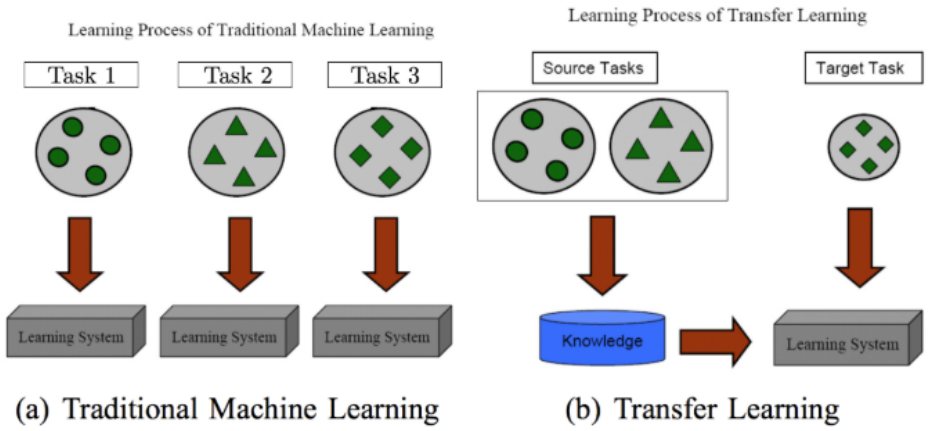


Figure 2.21: A comparison of traditional ML and transfer learning [167]

.

2.7.1 Formal Definition

A formal definition of transfer learning was provided in [167], which involves the concepts of domains and tasks as follows:

> A domain $\mathcal{D}$ consists of a feature space $\mathcal{X}$ and a marginal probability distribution $P(\mathbf{X})$ over the feature space, where $\mathbf{X} = x_1, ..., x_n \in \mathcal{X}$. Given a domain $\mathcal{D} = \{\mathcal{X}, P(\mathbf{X})\}$, a task $\mathcal{T}$ consists of a label space $\mathcal{Y}$ and a conditional probability distribution $P(\mathbf{Y}|\mathbf{X})$ that is typically learned from training data consisting of (x, y) pairs, where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. Given a source domain $\mathcal{D}_S$ and source task $\mathcal{T}_S$, as well as a target domain $\mathcal{D}_T$ and a target task $\mathcal{T}_T$, the objective of transfer learning is to enable the learning of the target conditional distribution $P_T(\mathbf{Y_T}|\mathbf{X_T})$ in $\mathcal{D}_T$ with the information gained from $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$ or $\mathcal{T}_S \neq \mathcal{T}_T$.

Given the preceding definition of transfer learning there are four key scenarios that arise:

1. $P(\mathbf{Y}_S|\mathbf{X}_S) \neq P(\mathbf{Y}_T|\mathbf{X}_T)$. The conditional probability distributions of the source and target

tasks are different. This is a very common scenario in practice and is often accommodated via artificial balancing techniques such as over sampling, under sampling, or SMOTE [28]. Consider the task of classifying malignant tumors from MRIs. Positive examples in this case, as in many healthcare problems, can be very scarce.

2. $\mathcal{Y}_S \neq \mathcal{Y}_T$. The label spaces between the two tasks are different. Consider again the binary classification problem of identifying malignant tumors from MRIs, and then a multi-class classification problem where a severity $1-5$ is assigned.

3. $\mathcal{X}_S \neq \mathcal{X}_T$. The feature spaces of the source and target domain are different. An example of this case is document classification where the documents are written in different languages.

4. $P(\mathbf{X_S}) \neq P(\mathbf{X_T})$. The marginal probability distributions of source and target domain are different (domain adaptation). For example, consider a collection of electronic health records discussing different topics (e.g., diabetes and weight management, COPD and smoking, or Alzheimer's and Dementia).

2.7.2 Pretrained Language Models

Since the introduction of the Transformer [211], introduced in Section 2.6, we have witnessed huge advances in the field of NLP. Such breakthroughs have been attributed in large part to leveraging huge, pretrained, Transformer-based language models. It can be concluded that Transformers lend themselves well to transfer learning strategies such as fine-tuning, which was not as common or successful in NLP when the field was dominated by RNNs. In order for pretrained base models to be robust and generalizable with respect to a range of diverse downstream applications, they are typically trained in a self-supervised fashion using a large amount of general purpose data. For example, a common pretraining task in NLP is masked language modelling (MLM) using an immense, diverse corpus such as Wikipedia. In MLM a proportion of each input sequence is hidden from the model (i.e., n input tokens are "masked"), and the objective is to correctly predict the missing tokens. Another common pretraining task in NLP is predicting the next token in a sequence in an autoregressive fashion. That is, the pretraining task in an autoregressive language model is to simply predict the token at position i given the preceding n tokens (i.e., $P(x_i|x_{i-1}, ..., x_{i-n})$). Fine-tuning is not only effective in boosting model performance with respect to task-specific applications, but also in better handling tasks that fall into the low-data regime, where training data is scarce, which is a common scenario in healthcare.

2.7.3 Robust Pretraining Strategies

While fully supervised pretraining is a popular strategy when labeled training data is available, the base task used for knowledge transfer is not dependant on supervision (i.e., labels). That is, with the abundance of general purpose, open source data available online it is common to train highly effective base models in an self-supervised fashion when large scale labeled data is not available. A well known example of self-supervised pretraining is contrastive representation learning [216, 104, 76], where the goal of the resulting network is to effectively differentiate between positive and negative examples. That is, the network is trained via an optimization objective where the goal is to minimize the distance between embedded positive pairs of examples (x_i, x_j), and maximize the distance otherwise. Contrastive loss is commonly defined as:

$$l_{i,j} = -log\frac{e^{sim(z_i,z_j)/\tau}}{\sum_{i=1}^{2N} \mathbf{1}_{k\neq i} e^{sim(z_i,z_k)/\tau}}$$

where temperature (τ) is a scalar, and sim is a similarity metric, such as cosine similarity given by:

$$sim(A, B) = cos(\theta) = \frac{A \cdot B}{||A||||B||}$$

Consider the domain of CV, where pretrained models have long been used as starting points for task specific training. In this case, the input is an image, positive pairs are augmentations of the original input with various augmentation strategies (e.g., rotation, cropping, distortion), and negative examples are selected from a pool of candidate images which are not inherently related to the original image. Another example of contrastive learning is in NLP, where examples are pairs of input queries and their user selected retrieval results (i.e., click through pairs), obtained through web logs. In this case positive pairs are given by the user, and negative pairs can be sampled from a pool of seemingly unrelated candidates, as in the CV example.

It makes intuitive sense that negative examples that are "too different" from positive examples would not help the model very much, since they are easy for the model to differentiate. As a result, it's wise to employ smart sampling techniques to extract negative examples that are more useful to the models learning task. So called "hard" negative examples can be mined in an online fashion (i.e., while the model is training) by using a similarity metric to calculate a score between the current positive example and all of the examples in the negative pool, and then using this score as a sampling probability, or by simply selecting from the top-k most similar negative examples. By hard-negative mining, the model is able to learn to differentiate between true positive and negative pairs much more effectively, and to generalize better to new data.

A popular contrastive pretraining method is momentum contrast [76]. The basic idea of momentum contrast is to use contrastive learning, as described above, but to employ a memory

bank in order to accumulate a larger pool of negative candidates to choose from, thereby allowing the model to ideally select better negative examples at each iteration. The basic flow of momentum contrast is shown in figure Figure 2.22.

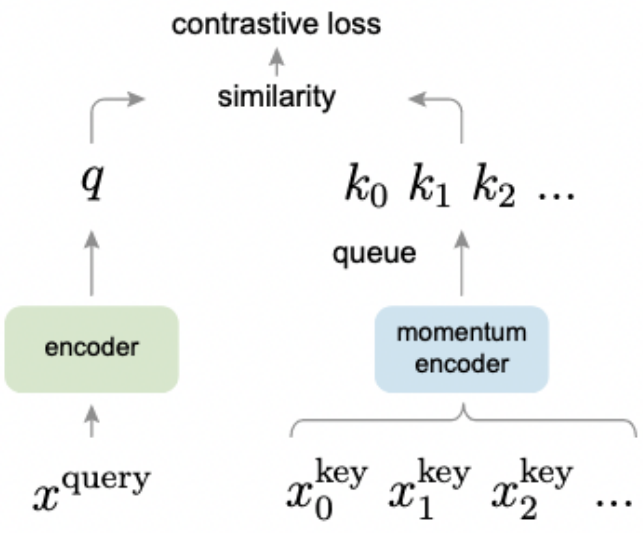


Figure 2.22: Momentum contrast, as in contrastive learning in general, aims to maximize similarity between positive pairs and minimize similarity between negative pairs (image from [76]).

To summarize, transfer learning is powerful and versatile in that it not only allows for knowledge to be transferred across tasks, but across diverse domains. The idea of using knowledge gained during the learning of one task, and applying that knowledge to another task makes intuitive sense as it's aligned with the way humans learn. Most tasks are related to other tasks in some way, and exploiting such complimentary, overlapping knowledge seems naturally beneficial to efficient learning.

2.7.4 Semi-Supervised and Multitask Learning

Several important areas of ML, including semi-supervised learning (SSL) and multi-task learning (MTL), are highly related to transfer learning. Semi-supervised learning serves to bridge the gap between supervised learning, where all instances are labeled, and self-supervised learning where no labels are available. It's common in practical SSL to have access to only a small number of labeled examples, which are supplemented with a large set of unlabeled examples to use for model training. SSL relaxes the dependence on labeled instances, and thus reduces the considerable expense of labeling data. It should be noted that, in SSL, both labeled and unlabeled instances are drawn from the same distribution. In contrast, in transfer learning, the data distributions of the source

and the target domains are usually different [234].

MTL can be differentiated from transfer learning in that tasks are learned in parallel in the multi-task setting as opposed to sequentially. That is, a set of tasks $T_1, ..., T_n$ are learned simultaneously whereas in transfer learning, a model is first trained on task T_1, and then fine-tuned on task T_2. MTL reinforces each task by taking advantage of the interconnections between task, and averaging the noise across tasks which serves to aid in generalization. That is, MTL tends to add a regularization effect by forcing the model to consider multiple tasks at the same time, instead of focusing on a single task, which is related to the concept of implicit data augmentation. To elaborate, implicit data augmentation refers to the fact that all tasks contain some degree of noise. As tasks $T_1, ..., T_n$ will have different noise patterns, a model that learns these tasks simultaneously should learn a more general representation. That is, learning a single task T_i in isolation runs the risk of overfitting to task i, while learning tasks i and j together allows the model to average the different noise patterns, yielding a more generalizable representation, and effectively increasing the sample size used for model training. Other benefits of MTL include concepts such as eavesdropping, wherein there exists a learned feature F, which is mutually beneficial to multiple tasks, say T_i and T_j, but one of the tasks is harder to learn, say T_j. It follows that the model may not be able to learn feature F if trained on T_j in isolation, but learning both tasks simultaneously allows for feature F to be learned more easily, which in turn benefits both tasks. Another beneficial quality of MTL is known as attention focusing [23], which can be summarized as follows: In cases where training data is high-dimensional and scarce, it can be difficult for a model to differentiate between relevant and irrelevant features. Attention focusing refers to the ability of MTL to prioritize relevant features by collectively taking into account all tasks, each of which provides its own evidence for the relevance of individual features.

We note that while the definition of MTL and transfer learning are not interchangeable, models trained in a multi-task fashion are commonly used as base models for subsequent fine-tuning. One of the most widely used examples of a base model trained with a multi-task objective is BERT, which was pretrained using a dual-objective and was introduced in Section 2.6. The main difference between transfer learning and MTL is that the former transfers knowledge contained in the related domains, while the latter transfers knowledge by simultaneously learning multiple tasks. At the end of the day, both transfer learning and MTL aim to improve the overall performance of a model via knowledge transfer.

2.8 Mixture of Experts

We introduce Mixture of Experts in this section as it motivated methods which will be showcased in Chapter 5, although it was not leverage directly. Mixture of Experts (MoE) is a paradigm, motivated by ensembling, which has existed in the ML community since the 1990s [101, 149, 95]. The general idea behind MoE is fairly straightforward: instead of using a single global model (e.g., DNN), or many local models (e.g., KNN), one can use several models of intermediate complexity which are referred to as "experts." This is advantageous if the input data comprises multiple regimes, where each regime has a different relationship between input and output. Under such conditions, each "expert" could learn to specialize in one regime, thereby achieving better performance across the dataset versus a single global model tasked with learning a representation that can adequately accommodate the full dataset.

One of the primary goals in MoE is encouraging the experts to specialize instead of cooperate (i.e., focus on a regime instead of averaging across predictors as in a traditional ensemble)? If the predictors are simply averaged, each expert is trying to compensate for the combined error made by the rest of the group. The goal is to make each expert focus on predicting the right answer with respect to cases where it is already doing better than the other experts.

A MoE layer, which can be a single block within a larger network, is composed of a set of n "experts" $E_1, ..., E_n$ (e.g., n separate networks), and a gating network $G \in \mathbf{R}^n$. The job of G is to regulate the contribution of each of the experts with respect to the current input. Let $E_i(x)$ and $G(x)$ be the output of the i^{th} expert network and the gating network respectively. The output y of the MoE layer can now be written as:

$$y = \sum_{i=1}^{N} G(x)_i E_i(x)$$

A simple gating function is softmax gating $G_\sigma(x) = Softmax(x \cdot W_g)$, where the softmax function is applied to the product of the input x and a trainable weight matrix W_g. Differentiating with respect to the outputs of the experts yields a signal for training each expert, and differentiating with respect to the outputs from the gating network provides a signal for training the gates. The basic structure of an MoE layer is shown in Figure 2.23.

One of the first practical, and wide reaching examples of MoE in modern DL was through the introduction of sparsely gated MoE by Shazeer et al. in 2017 [202]. A key contribution of this work was in the sparsification of the gating network G, which allowed for the use of a very large number of experts without the high computational overhead. The idea of sparsely gated MoE is as follows: the output of G will be a sparse n-dimensional vector to save computation by only

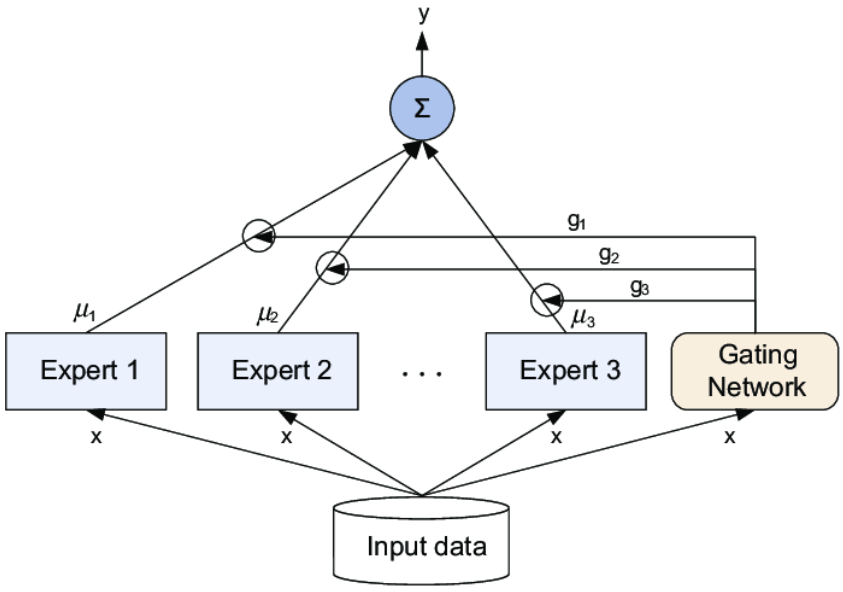


Figure 2.23: Mixture of Experts layer.

retaining k of the n experts, where $k << n$, at each iteration. In other words, $\forall G(x)_i = 0$ the corresponding expert E_i does not need to be computed. This is accomplished via a mechanism that the authors refer to as "noisy top-k gating." Noisy top-k gating adds two additional components to basic softmax gating function defined above. First, before the softmax is applied, tunable Gaussian noise is added. Second, only the top-k values are kept. All other values are set to $-\infty$ which causes the gate to zero them out. The gating function is now:

$$\begin{aligned} G(x) &= Softmax(KeepTopK(H(x), k) \quad where \\ H(x)_i &= (x \cdot W_g)_i + StandardNormal() \cdot Softplus((x \cdot W_{noise})_i) \quad and \\ KeepTopK(v, k)_i &= \begin{cases} v_i & \text{if } v_i \text{ is in the top-k elements of v} \\ -\infty & \text{otherwise} \end{cases} \end{aligned} \tag{2.17}$$

The sparsity condition is primarily a means to save computation, while the noise aids in load-balancing. The latter will not be discussed here, but is elaborated on in detail in [202]. In addition to introducing this modified gating network Shazeer et al. also identify the following phenomenon: the gating network tends to converge to a state where it always produces large weights for the same small subset of experts. This imbalance is self-reinforcing, as the favored experts are trained more rapidly and thus are selected even more by the gating network. To accommodate this behavior a penalty term $L_{importance}$ is added to the overall objective, and serves to encourage importance

equality across experts. $L_{importance}$ is defines as:

$$\begin{aligned} L_{importance} &= w_{importance} \cdot CV(Importance(X))^2 \quad where \\ Importance(X) &= \sum_{x \in X} G(x) \end{aligned} \tag{2.18}$$

MoE is an effective way to scale models and achieve better performance. However, increasing model scale comes with its own set of challenges which need to be taken into account alongside performance gains. Effective ways to make practical use of massive, modern DNNs is a theme which is central to Chapter 5, for which Sparsely gated MoE served as motivation. More specifically, the idea of over-parameterizing a network, and then reducing network parameters during training, which is a central concept in Chapter 5, is related to reducing an initially huge number of candidate "experts" to a more manageable, and practical subset.

2.9 Parameter Efficient Fine-Tuning

Transfer learning via fine-tuning a large, pretrained DNNs is a common, and highly effective technique for adapting a model to a new task (i.e., model customization). Modern transfer learning strategies have yielded SOTA results across a range of domains and tasks. As the best performing pretrained models increase in size, as is the trend in DL, new issues become increasingly evident. Some key issues related to model size include, but are not limited to, high storage overhead, high GPU memory requirements, long training times, and the potential for catastrophic forgetting, where beneficial knowledge can be lost as a result of sequentially training the same model on different tasks [64].

One of the most useful lessons from transfer learning is that a single, generally pretrained DNN can be adapted to perform many diverse, downstream tasks extremely well. However, adapting and deploying a model of immense size in a real-world production system is not a trivial task. With traditional fine-tuning, every parameter in the base model is updated during adaptation. It follows that a full copy of the base model is required for each task adaptation, resulting in a fine-tuned model variant for each task which is the same size as its base model counterpart. In short, adapting a pretrained base model for n different tasks via traditional fine-tuning quickly becomes prohibitively expensive as n gets large.

Parameter efficient fine-tuning (PEFT) is a methodology which has evolved alongside massive pretrained DNNs in order to enable practical adaptation to downstream tasks. Some popular methods of PEFT are Adapters [89], PrefixTuning [136], and Low-Rank Adaptation [93]. A key concept in PEFT is that a large, pretrained model can be adapted to a new task by updating a

small number of parameters compared to the overall network size. This idea of PEFT is in contrast to full fine-tuning, wherein all network parameters are updated during adaptation. PEFT will be elaborated on in Chapter 5, where the primary theme is parameter budget allocation during PEFT or large vision models.

Adapters

Adapters, which were introduced by Houlsby et al. in 2019 [89], are one of the earliest practical implementations of PEFT. In Adapters the base model is frozen, and only a small set of supplemental parameters are updated to help the model adapt to a new task. Freezing, or "preserving" base model parameters is a tactic which is commonly employed in PEFT. Supplemental parameters are added to the base model by inserting trainable layers (adapter layers) between consecutive base model layers at strategic points in a network. Although Adapters are a highly effective PEFT method, their sequential application effectively increases the depth of the resulting network, which can incur unwanted penalties such as increased latency. The basic anatomy of an Adapter layer is shown in Figure 2.24.

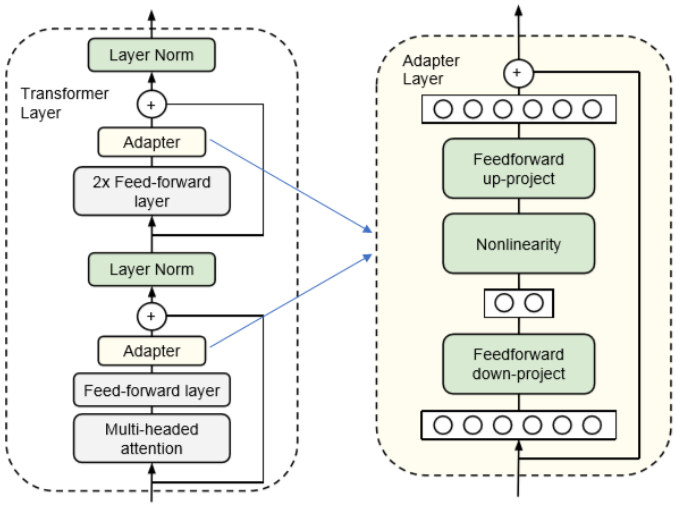


Figure 2.24: Design of the adapter module (right) and placement within a transformer block (left). Each adapter is inserted between two subsequent layers in the base model.

Prefix Tuning

Prefix tuning was introduced withing the domain of NLP by Li et al. [136], and has become a popular PEFT method in the last few years. In Prefix Tuning [136], the basic idea is that a language model can better perform the task of generating an appropriate completion if it's provided with a context (prompt) which contains enough information to steer the model in the right direction. Consider an example where the prompt is "the sky is" and the desired completion is "dark." If the model is fed this prompt as is, it will most likely yield fairly high likelihoods for multiple candidate tokens including "blue," "clear," "cloudy" and others. Now, if the prompt was supplemented with the prefix "At night" to make it "At night the sky is," the likelihood of returning "dark" as the next token becomes much higher. The concept of providing helpful hints to a language model to help guide its behavior is the intuition behind prefix tuning. This concept can be made even more efficient by adding trainable embeddings to only the first layer in the network, which was demonstrated in [133], and can be leveraged to further reduce the number of trainable parameters. The general idea of prefix tuning is illustrated in Figure 2.25.

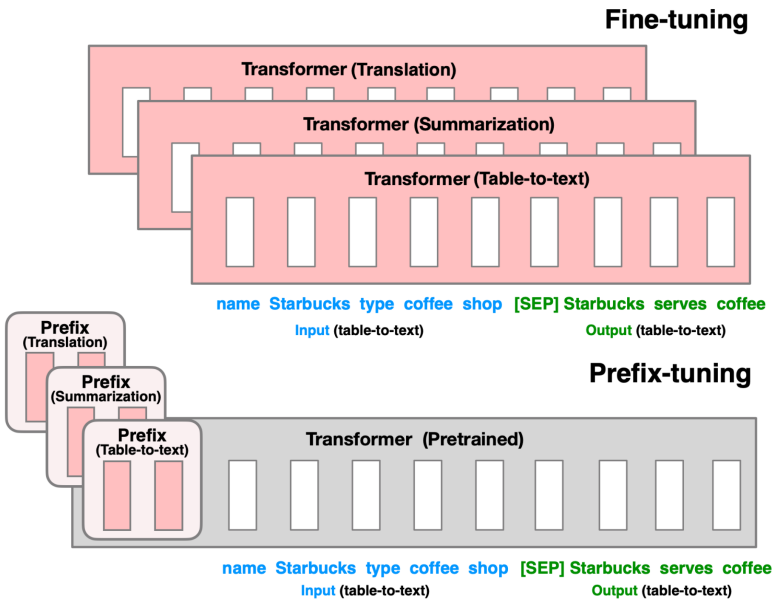


Figure 2.25: In prefix tuning, a small number of learnable parameters are prepended to each prompt. For the example of table-to-text, each example is fed into the model as prompt, which is the linearlized representation of the table, followed by a special token to separate the prompt and completion (i.e., [SEP]) in this example, followed by the completion, which is a natural language description of the table. A similar input representation can be followed for other common tasks such as translation, where the prompt is the statement in language A and the completion is the statement in language B, or summarization, where the prompt and completion are the full text, and the summarized version respectively.

Low-Rank Adaptations

Low-Rank Adaptations (LoRA) [93] borrow from the layer structure introduced in Adapters, but add several notable innovations.[1] It can be seen from Figure 2.26 that LoRA layers are applied in parallel, which allows for weights to be merged during inference thereby bypassing the increased latency associated with adding depth to the network, which is a drawback of Adapters. Moreover, LoRA layers can be thought of as factorizations of their base model counterparts, therefore LoRA can be merged and unmerged with the base model via simple linear operations, which makes task switching highly efficient in practice. That is, LoRA weights can be "folded" back into the base model during inference, yielding a forward pass which is computationally identical to that of the base model. As such, a single base model with many LoRA customization's (i.e., n different task specific applications), can efficiently switch between tasks with very low computational overhead.

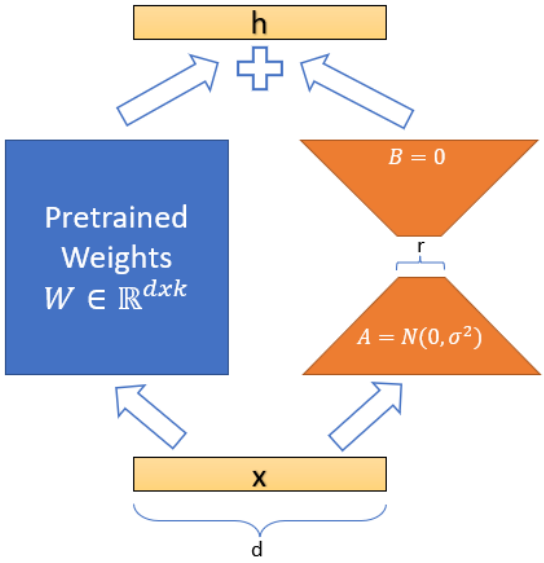


Figure 2.26: In a LoRA layer, the input x is passed simultaneously through the LoRA layer and the corresponding base model layer. The output signals are then combined into a single output activation via component-wise addition.

LoRA is a simple, elegant, and highly effective approach to PEFT which was shown in [93] to outperform other methods including Adapters, PrefixTuning, and full fine-tuning across several common benchmark tasks in NLP. In short, LoRA is highly scalable, versatile, and can be employed as a drop-in replacement for any dense layer in a DNN. It's also worth noting that while this section

[1] The author of this thesis was also a co-author of the original LoRA paper, but the work was done as part of the author's work at Microsoft, and was not done in association with OHSU as part of this thesis.

discusses LoRA in the context of dense DNN layers, this method can easily be extended to other commonly used layers, such as $1D$ and $2D$ convolutions.

Chapter 3

Learning Representations of Clinical Events

3.1 Motivation

In many clinical modeling efforts, researchers are tasked with hand engineering features from the medical histories of patients. Examples of such features can include type and/ or frequency of visits, quantity and characteristics of prescriptions, proximity to care, and patient demographics. While hand-engineered features can be used to create highly effective clinical models, the manual process of creating meaningful representations of a patient's medical history in this way is not trivial, and requires a great deal of domain expertise. Moreover, creating features by aggregating events will squander most of, if not all of the temporal information contained in the underlying data. Hand engineering sequential features to preserve some temporal information, such as $surgery_A \rightarrow prescription_B$, would require an exponentially large number of features including many which could be irrelevant. The ability to automatically learn meaningful representations from clinical data allows for complex relationships to be identified and leveraged, even those not immediately apparent to a subject matter expert (SME).

In this chapter we propose methods to learn clinical event embeddings, which we define as dense, vectorized representations of the standardized medical codes discussed in Section 3.2.1. In order to learn such meaningful, and robust embeddings in an automated fashion we leverage techniques from modern NLP, in particular neural language modeling[1]. We will show that embedded representations from different clinical event domains including diagnosis, procedures and prescriptions, can be learned jointly in order to infer semantic relationships within, as well as between domains. We

[1]The work in this chapter was previously published in [213]: P. Wallis, P. Danaee, Learning Semantic Relationships from Medical Codes in AAAI Publications, as part of the Thirty-Second International FLAIRS Conference, Special Track on Artificial Intelligence in Healthcare Informatics 2019

present three methods for learning clinical event embeddings from sequences of medical codes which we refer to as event2vec embeddings, autoregressive embeddings, and a combination of the two which we refer to as "hybrid" embeddings. Each embedding method is evaluated, with the help of a SME, to determine if the resulting representations are meaningful and informative from a clinical perspective. We will also demonstrate how meaningful embeddings of clinical events can be used directly as inputs to clinical predictive models, thereby eliminating the need to hand-engineer features based on intuition, heuristics, and/ or domain expertise. The ability to represent medical codes, which are ubiquitous in healthcare, in a way that embeds semantic meaning is crucial to effectively leverage the rich, meaningful, and complex information they represent. We aim to provide evidence that clinical event embeddings are useful, robust, and interpretable representations of clinical events, which can be leveraged by many diverse applications. The work presented in this chapter exemplifies the broad power of DNNs in healthcare as a tool for advancing healthcare AI towards the common good.

In Section 3.2.1 we discuss medical coding, in part to illustrate some of the challenges associated with using such representations in practical machine learning systems. We then introduce several established methods for creating embeddings from sequences of including word2vec in Section 3.2.2, and RNNs in Section 3.2.3, both of which we later extend to learning meaningful embeddings of clinical events. Finally, we evaluate the effectiveness of such learned representations on the practical application of clinical event prediction in Section 3.7.

3.2 Background

Much of a patient's medical history is documented using coded representations of clinical events (i.e., medical codes), treatment locations, and provider notes along with corresponding timestamps. Codes for diagnosis, procedures and prescriptions represent conditions, as well as interventions. There are several well-known and widely used methods for learning embeddings from words including Word2Vec [148] and Global Vectors (GloVe) [170]. Due in part to the prevalence of electronic health records (EHRs), researchers have begun to widely explore the use of established NLP methods, such as embeddings and neural language models, in healthcare. The sequential nature of clinical events lends itself naturally to established natural language embedding methods such as word2vec, which is discussed in Section 3.2.2 and which forms the basis of our first approach, showcased in Section 3.5.1. Moreover, sequences of clinical events can be used as inputs to seq2seq models including RNNs, which were discussed in Section 2.4.1, and which are the DNN architecture leveraged for our second embedding method in Section 3.5.2, and which are further showcased in the context of

clinical predictive modelling in Section 3.7.[2].

3.2.1 Medical Coding

Ontologies of medical concepts such as the Unified Medical Language System (UMLS), the International Classification of Diseases (ICD-10-CM and ICD-10-PCS), and the National Drug Code (NDC) are used throughout the healthcare system by insurance providers and medical providers alike. On the provider side, medical codes are used for billing, and by doctors, nurses and clinicians to document clinical actions. On the insurance side, these same medical coding systems are used to document and summarize clinical events for insurance claims.

With these standardized coding systems, a patient's medical history can be represented by a sequence of diagnosis, procedure, and prescription codes. Medical ontologies however, are large, which can make them challenging to use in machine leaning systems. There are roughly 70K diagnosis codes alone, with an equally large number of procedure codes, and over 360K NDC codes making for a very large vocabulary for practical purposes (i.e., $|V| \approx 500K$). To provide some perspective, modern tokenization strategies such as the word-piece tokenizer used in [48], or byte-pair tokenizer introduced in [179] will require $\approx 30K - 50K$ tokens to represent the English language. A multilingual tokenizer, capable of handling dozens of different languages, and utilizing a similar tokenization strategy may have $|V| \approx 250K$.

Medical codes are the standard for representing clinical information across the healthcare system, and while these schemes are created to encode relevant information, they do not contain clear semantic relationships. The hierarchical structure of standard clinical ontologies allows for the identification of certain types of relationships, but they are limited by their top-down structure. It follows that individual codes have meaningful relationships within, and across clinical domains, which are not inherent in the codes themselves.

3.2.2 Word2Vec

Word2vec, which was introduced by Mikolov et al. in [146], was a seminal work within the field of NLP, and described two methods for learning dense representations of words: skip-gram, and continuous bag-of-words. Both methods are unsupervised neural language embedding techniques which are themselves very closely related. We will focus on the skip-gram method here as it is more relevant to this work.

[2]It should be noted that modern language models, and models that take sequential data in general are typically transformer based. At the time of this research the transformer, while having recently been proposed in [211], was still a new DNN architecture and was not used during this research.

The objective of the skip-gram model, as illustrated in Figure 3.1, is to output higher probabilities for vocabulary words which are more likely to be seen in close proximity to a given target word. In other words, we would ideally like the model to output a high value for $P(c_j|w_i)$ for all j such

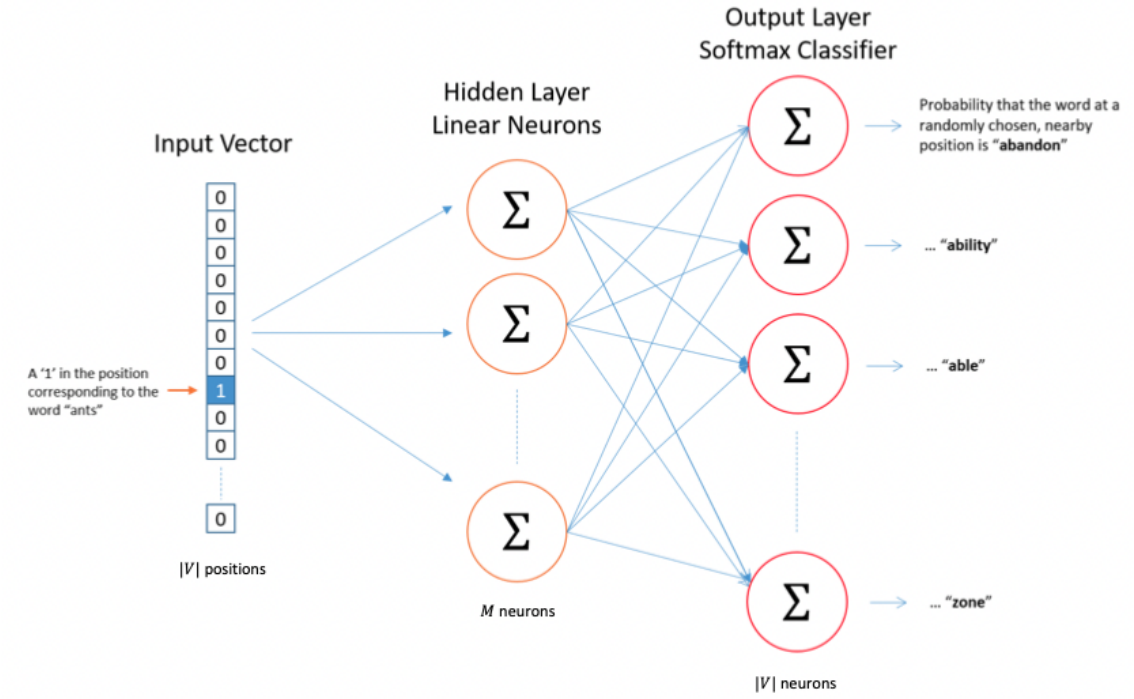


Figure 3.1: Skip-gram architecture taken from [144]. The input is a one-hot vector for the word "ants" in this example. The ouput of the network is the probability that a randomly selected vocabulary word is "close" to the given target word. In this figure each input is $|V|$ dimentional, where V is the vocabulary. M is the number of neurons in the hidden layer, which equates to the embedding size.

that c_j appears within a reasonably sized context window around w_i, which is the center word or "target." For example, if the target word is "pool" it should be more likely that the word "swimming" appears within a small context window (say within two words), compared to the word "fire." A common window size in practice is 2 to 5. Larger context windows typically increase performance, up to a point, but increasing window size adds significantly to the training set size. During training, individual tokens are input as one-hot vectors of size $[1, |V|]$, where V is the vocabulary. Each one-hot vector is passed through a hidden layer (i.e., embedding layer) $W^h \in \mathbb{R}^{|V| \times M}$, where M is the embedding size, which is then fed to an output layer $W^o \in \mathbb{R}^{M \times |V|}$ with softmax activation. Given an input target word, the network will output a vector where each position i represents the probability that a randomly selected word close to the given target is the vocabulary word w_i. Once the network is trained, the output layer is discarded so only the hidden layer is retrained as it

represents a lookup table of embeddings for each vocabulary word.

To train a skip-gram model, a large training corpus (i.e., collection of documents) is converted into a set of $(target, context)$ pairs, which are used as training examples. A $|V| \times M$ matrix, even with a moderate vocabulary size (say $10K$) and reasonable embedding dimension (e.g., 200) is still fairly large, and would require a huge number of samples to train without overfitting to the training data. To address this large data requirement, two methods were introduced by Mikolov et al.: subsampling frequent words to decrease the number of training examples, and negative sampling, which is essentially a modification to the models optimization objective, wherein each training sample updates only a small percentage of the model's total weights.

Subsampling

With any training corpus there will inevitably be tokens that appear much more frequently than others (i.e., common tokens). For example, in a typical English language corpus the token "the" will appear in the context with respect to the majority or target tokens. It follows that the network will naturally see many more examples containing common tokens during training. The idea of subsampling was proposed as a way to address this issue. At a high level, each token is assigned a probability that it will be effectively deleted from the text. This probability, denoted $P(w_i)$, is directly proportional to the token's frequency $f(w_i)$, as defined by:

$$P(w_i) = (\sqrt{\frac{z(w_i)}{s}}) \cdot \frac{s}{z(w_i)}$$

In this equation s is a sampling rate, originally set to 0.001 by default, and $z(w_i) = \frac{1}{f(w_i)}$ is the inverse frequency of occurrence for token w_i in the corpus.

Negative Sampling

During skip-gram model training, all model parameters will be updated for each example by default. Updating a large number of parameters slightly for each example is clearly not efficient. The idea of negative sampling is to reduce this overhead by only updating a small percentage of the total network parameters at each step. The correct label for each example pair is a one-hot vector, hence we want the neuron in the output layer corresponding to the correct vocabulary word to be 1, and all others to be 0. When we employ negative sampling, we randomly select a small number of"negative" words (e.g., 5) to update the weights for. By "negative" words we are referring to words for which we want the network to output a 0. Of course we will still need to update the weights for "positive" word, which is the correct context word. In the hidden layer, only the weights for the input word are updated (this is true whether you're using Negative Sampling or not). The

"negative samples" (that is, the 5 output words that we'll train to output 0) are chosen using a "unigram distribution". Essentially, the probability for selecting a word as a negative sample is related to its frequency, with more frequent words being more likely to be selected as negative samples. The probability of selecting a negative example w_i in a corpus is given by:

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n} f(w_i)^{3/4}}$$

The power $\frac{3}{4}$ was arrived at empirically by the authors of the original paper.

3.2.3 Recurrent Neural Networks

Another method for learning embeddings in an unsupervised fashion is by training an autoregressive language model. At the time of this research RNNs were the dominant architecture for problems involving sequential inputs, and LSTMs in particular, which were introduced in Section 2.4.1, will be the RNN variant used in this chapter. Although LSTMs can be used for many common NLP problems including machine translation, question answering, text classification and others, the applications in this chapter will be focused on next token prediction, and sequence classification (i.e., the task of assigning a class label to a sequence) which is illustrated in Figure 3.2.

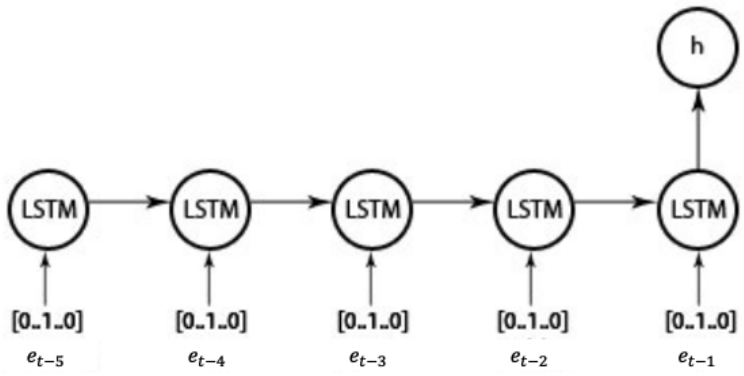


Figure 3.2: An example of an LSTM used for sequence classification. Tokens are input as one-hot encodings.

To train a RNN for the task of predicting the next most likely token, it suffices to break up sequences of inputs into input-target pairs where $input = [x_{n-k}, x_{n-k+1}, ..., x_{n-1}]$ and $target = x_n$. That is, the network will be tasked with generating $P(x_n|x_{n-1}, ..., x_{n-k})$. Cross-entropy loss is commonly used with this setup, where at each step the network produces a probability distribution over each token in the input vocabulary, which is penalized based on how different it is from the

actual distribution. Since the true distribution would have a 1 in the position corresponding to the correct token and 0's everywhere else, this comparison only needs to the output probability of the correct token.

$$L(\theta) = -\sum_i y_i \log \hat{y}_i = -\log \hat{y}_i$$

With m training examples this becomes the average negative log probability of correct prediction.

$$L(\theta) = -\frac{1}{m}\sum_{i=1}^{m} \log \hat{y}_i$$

Once the RNN is trained, the frozen embedding layer can be extracted and used to map new sequences of tokens (clinical event codes in our case) to embedded representations for use in downstream applications.

3.3 Related Work

Deep Patient

Deep Patient by Miotto et al. [151] proposes a general-purpose patient embedding approach using EHR data. The resulting learned patient level representations were empirically shown be useful in downstream clinical predictive modeling tasks. Specifically, patient representations were used to infer the probability of a patient to develop various diseases. Miotto et al. collected EHR data from around 700K patients which was aggregated, and fed to a stacked denoising autoencoder tasking with capturing hierarchical regularities, and dependencies contained within the aggregated patient records. EHR data was aggregated by extracting general demographic information such as age, gender and race, as well as clinical descriptors for diagnosis, medications and procedures, along with lab tests and clinical notes. To reduce the vocabulary size needed for EHR aggregation, ICD-9 codes were mapped to a disease categorization structure used at Mount Sinai comprising 231 general disease definitions, which was further filtered to 78. The final vocabulary size was approximately 41K. Even so, the final patient dataset used for representation learning was still $\approx$ 99% sparse (i.e., 1% of the data in the patient-descriptor matrix was populated with values ¿ 0).

Evaluation was performed using roughly 76K test patients, spanning 78 disease classes from a range of diverse clinical domains including diabetes, schizophrenia, and various cancers. The likelihood of developing a certain disease was performed in a one-vs-all fashion using a random forest classifier which was fed the patient level representations described above. The proposed representation learning approach was shown empirically to outperform other feature learning

methods such as PCA and GMM with respect to the clinical predictive modeling task. While the results of this study showed promise for patient level representation leaning, the authors do not attempt to lean representations while performing a clinical predictive modeling task in an end-to-end fashion, and do not compare their results to those obtained by other well known neural language modelling approaches such as Word2Vec. Moreover, aggregating EHRs to the patient level before learning embeddings results in a loss of valuable longitudinal information contained in the time-coded sequences of clinical events that comprise a patient's clinical history.

Patient Similarity via Medical Concept Embedding

Zhu et al. present a DL approach to learn contextual embeddings of medical concepts. The leaned embeddings are then used to assess clinical similarities between pairs of patients, which is a fundamental problem in healthcare informatics [233]. Like Deep Patient, Zhu et al. leverage EHRs to extract relevant training data. However, their data processing involves extracting sequences of events from EHRs by timestamp, thereby preserving longitudinal information rather than aggregating the full EHR to the patient level. Unlike traditional word2vec variants, which uses a fixed sized context window and consider successive sequence elements to be evenly spaced (i.e., no notion of time between events), the authors propose a variation of word2vec, in particular skip-gram, which incorporates an adaptive method for determining context window length with respect to an event in the EHR of a specific patient. This methods relies on various heuristics. For example, chronic conditions are more likely to occur repeatedly in an EHR as compared to acute conditions. The window length L with respect to an event i and a patient p is then represented as $L(i,p) = f(i,p) \times a + \theta$, where a and θ are constants and $f(i,p)$ is the frequency of event i in the EHR for patient p. Once medical concept embeddings have been trained, each patient is represented by a sequence of "visit" embeddings. That is, for each visit in a patient EHR, the embedded visit is calculated as the sum of the embedded representations of all medical concepts that occurred during that visit. It's unclear if the authors evaluated other embedding pooling methods such as mean, and max-pooling to compare to summing of embeddings. Zhu et al. then propose a dual-encoder style CNN architecture which encodes each patient embedding matrix and then uses the encoded representation of each patient to measuring distance between patient vectors. The network is trained in a supervised fashion, hence requires a label for each patient-pair. Second, the authors proposes unsupervised methods for assessing patient similarity which including the use of RV coefficient, defined as $RV(\mathbf{X},\mathbf{Y}) = \frac{tr(\mathbf{X}\mathbf{X}'\mathbf{Y}\mathbf{Y}')}{\sqrt{tr(\mathbf{X}\mathbf{X}')^2 tr(\mathbf{Y}\mathbf{Y}')^2}}$ where $\mathbf{X} \in \mathcal{R}^{n \times k}$ and $\mathbf{Y} \in \mathcal{R}^{m \times k}$ are matrix representations for a pair of patients.

Zhu et al. use around 200K EHRs in total for model training and evaluation, with patients

belonging to four cohorts: Chronic Obstructive Pulmonary Disease (COPD), Diabetes, Heart Failure, and Obesity. Evaluation of the proposed embedding method was performed on the basis of patient cluster quality as measure by three popular cluster evaluation criteria: Rand Index , Purity and normalized mutual information(NMI). While the proposed method showed more desirable cluster characteristics compared to baseline methods such as KMeans, the learned embeddings are not applied to downstream applications to assess their effectiveness with respect to broader clinical tasks such as diagnosis, or more general clinical event prediction.

Doctor AI

In Doctor AI [31], Choi et al. present an effective RNN based approach to clinical predictive modeling which leverages longitudinal, time-stamped EHR data from 260K patients spanning an eight year period. Choi et al. extracted diagnosis, medication and procedure data, which was then fed to a RNN based model tasked with multi-label, next event prediction of all medications and diagnosis. That is, the model outputs a prediction for each event that could come next (i.e., can predict more than one next event) as opposed to a probability distribution over a vocabulary of clinical events which would be output in the multi-class setting. In order to mitigate issues related to large vocabulary size, Choi et al. reduce the dimentionality by using 3-digit ICD-9 codes resulting in 1183 unique codes, versus around 11K. For medication codes, they used the coarser Generic Product Identifier Drug Class as opposed to the more granular GPI, which reduced the total number of medication codes from roughly 19K to around 600. The authors reported a 79% top-30 recall which outperformed baseline approaches, where top-k recall=$\frac{\text{TP in top-k predictions}}{TP}$ on a held out test set. While this study highlights the predictive power of DNN based sequence modeling approaches applied to clinical events, the embeddings themselves are not extracted, evaluated, and applied to downstream applications to assess the generalizability of the learned representations.

Low Dimensional Representations of Medical Concepts

In Choi et al. the authors aim to learn effective embeddings from collections of medical journals, clinical notes, and medical codes. The authors speculate that such learned embeddings could be highly useful in medical informatics for tasks such as cohort selection and patient summarization. Another contribution of this work is a simple method for learning medical concept embeddings from co-occurrence counts extracted from clinical narratives with the goal of preserving privacy. Choi et al. learn medical concept embeddings via a modified skip-gram style approach which uses partitioning and random-shuffling. Specifically, given a time interval T used for partitioning, they partition the data into intervals of size T, and then remove duplicate mentions of a concept within

each partition s_i (sub-sequence). The resulting concepts within each partition are then randomly shuffled yielding a new sequence s'_i, which is subsequently treated as a single sentence to be used for model training. Choi et al. assess the embedding method by quantifying "medical relatedness" between clinical events, but do not apply the leaned representations to downstream tasks such as clinical predictive modeling. Moreover, while the proposed method serves to add an element of privacy by partitioning and randomizing sub-sequences, it also results in a loss of some temporal information contained in the specific event orderings, as well as a loss of event magnitude which could be gleaned from the frequency by which events occur.

Medical Concept Embedding with Time-Aware Attention

Cai et al. propose a variation of word2vec, specifically CBOW [148], which aims to learn embeddings of medical concepts which take into account the non-uniform time between medical concepts in EHRs. Like the related studies outlined above, Cai et al. extract sequences of medical concepts from EHRs, and use said sequences to train a word2vec style model. However, they replace the fixed context window employed by traditional CBOW by what the authors define as a time-aware attention mechanism tasked with learning attention weights with respect to sequences of medical concepts, and their associated timestamp. The authors define a "temporal context scope" as the largest number of time units between a target medical concept, and its corresponding set of context medical concepts. Essentially, each EMR is converted to a sequence of medical concepts, ordered by timestamp, where sub-sequences of medical concepts that fall within a time interval t are represented by the sequence $E_t = [c_{t,1}, c_{t,2}, ..., c_{t,K_t}]$ where K_t is the total number of medical concepts in E_t. The authors note that relations between medical concepts and time periods can be learned from the correlations between medical concepts (e.g., a common cold may be correlated to context concepts that happen in the following week). They use this observation to build what they refer to as a time-aware attention model to learn non-uniform attention weights within a temporal context scope. The goal of attention in this setup is to allow the model to pay more attention to contexts within the deemed "relevant" this time period with respect to each target. In order to avoid issues like a very large number of contexts for a given target, Cai et al. define a predefined threshold to cap the maximum number of context medical concepts. The embedding layer in the resulting modified CBOW formulation is composed of non-uniform weighted context vectors, incorporating the time-aware attention model, which is parameterized by the target medical concept and the time gap between the target concept and each context concept. The result is essentially an attention weighted sequence which is used to train a CBOW style embedding network.

The learned embeddings are evaluated on clustering and nearest neighbour search tasks. While

the proposed method does outperform comparable embedding methods which do not utilize time-aware attention, such as word2vec and Glove [170], the evaluation is limited to vector search and clustering, and the leaned embeddings are not evaluated on a broader, more diverse set of downstream tasks.

3.4 Data

3.4.1 Description of the Data

Healthcare data can be notoriously high dimensional, sparse, and heterogeneous, which requires thoughtful preprocessing. All data used in this chapter was extracted from anonymized, historical medical claims which were used to extract sequences of clinical events, represented by diagnosis, procedures, and pharmacy codes at the patient level. All data used in this research was accessed with permission from Regence BCBS while the author was employed at Cambia Health Solutions in Portland, OR. All data extraction and processing procedures were created by the author as part of this research.

A typical medical claim contains collections of medical codes representing low-level clinical events, such as diagnosis, which are grouped together in time by higher-level events, such as an office visit or hospital stay. By extension, a patient's claims history comprises a sequence of clinical events, along with their respective timestamps, over a certain time period. In this work, only the clinical events themselves were used to create sequences, not the timestamps. That is, high level groupings as well as time deltas between subsequent events were discarded during training.

The data described in this section was used for all three embedding methods described in Section 3.5, and each of the predictive modelling experiments showcased in Section 3.7. Since all data used in this chapter originated from insurance claims, clinical events that were grouped within the same high level event, such as a diagnosis and prescription occurring during the same office visit, were associated with the same timestamp by convention. Although this issue affected $< 1\%$ of the events in our filtered training set, we attempted to deal with the ambiguity in several ways: First, we simply filtered out sequences that contained multiple events under the same timestamp. Second, when a group of clinical events associated with a common timestamp were encountered (for a single patient) we either (1) randomized the events, or (2) randomly selected a single event for each group. Last, we used a set of simple rules to prioritize the order of events by domain. For example, diagnosis $<$ procedure $<$ prescription. If more than one event from the same domain occurred in the same visit, they would be randomly assigned order. Since the number of sequences effected was small, we decided to determine the ordering of simultaneous events using the rule-based system.

We collected up to one years worth of data from $\approx 1M$ patients having at least three months of eligibility (potential claims) between Jan 2016 to Jan 2018. For each patient, we selected a random date from the patient's set of eligible claims, and pulled all claims going back up to one year from that date. By doing so, we allowed for each patient to have a different start and end date with respect to their individual clinical snapshots. The rational for distributing patient claims over time in this way was to mitigate the effects of seasonality that could have been an influencing factor if all patient data was restricted to the same date range. Figure 3.3 shows how the historical data can be distributed in time. Figure 3.4 provides an example of a timeline of clinical events for a hypothetical patient. The patient represented in Figure 3.4 corresponds to $patient_1$ in Figure 3.3. We can see that this patient's clinical snapshot is taken going back up to one year from 07/07/2016, and that the patient only had around 6 months of prior clinical history. Hence a full year snapshot is not given for this patient.

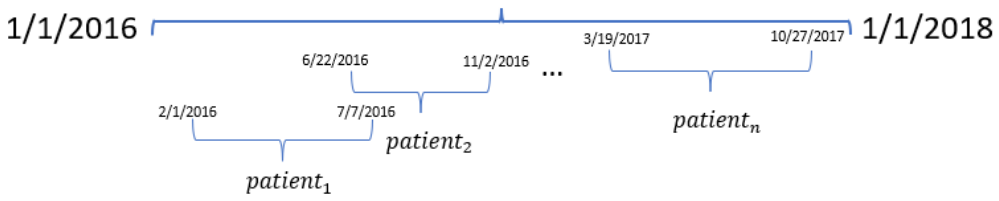


Figure 3.3: A random sample of 1M patient with eligibility between Jan 2016 and Jan 2018 was used for all experiments. For each patient, a random date was sampled from the patients eligibility range, and all claims going back up to one year were used for subsequent analysis and modelling. The above figure shows how each patient can have a different date range. This was done to avoid issues related to seasonality.

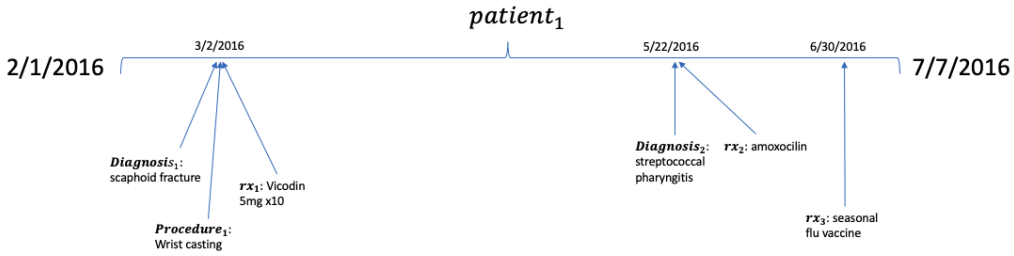


Figure 3.4: An example of an individual patient timeline (patient 1 from Figure 3.3). This patient has 3 clusters of clinical events: first, a wrist fracture diagnosis followed by casting and a prescription for pain medication (all happen during the same visit); second, a diagnosis of strep throat and a prescription for antibiotics (both events occur during the same visit); third, a seasonal flu shot

.

3.4.2 Data Preprocessing

Within the full patient sample there were $\approx 20K$ unique procedure codes, $\approx 40K$ unique diagnosis codes, and over $300K$ unique pharmaceutical codes, yielding a large total vocabulary size of $|procedures| + |diagnosis| + |pharmacy| > 360k$. In order to get the raw data into a format which could be processed by the models, several preprocessing steps were performed: First, to make the large initial vocabulary more manageable, highly sparse, fine grained codes were clustered into coarser groupings. The overall vocabulary size was reduced by grouping diagnosis and procedure codes using the Clinical Classifications Software (CCS) for ICD-9-CM, developed and maintained by the Healthcare Cost and Utilization Project (HCUP), which is sponsored by the Agency for Healthcare Research and Quality (AHRQ) [164]. CCS categories for diagnosis are generated based on the International Classification of Diseases (ICD-9 and ICD-10), and Current Procedural Terminology codes (CPT) for procedures,[3] which are widely used standard grouping techniques in clinical applications. Pharmaceutical codes were also grouped, in part to eliminate duplication (i.e., there can be many NDC codes used for the same medication), and to further reduce the vocabulary to a manageable size. This step helped decrease complexity and improve generalization of the models. Next, we estimated the mean and standard deviation of the number of events in a patients clinical history (up to one year) via bootstrapping. We then filtered out examples with clinical event sequences that were not within three standard deviation from the mean by number of events, or that had less than 15 events in total. That is, if a patient had a very high, or very low number of events in their claims history, with respect to the estimated population statistics, they were excluded from the training set.

3.5 Embedding Methods

General purpose embeddings of clinical events have many practical, and beneficial uses in healthcare AI. In this section we present three methods for learning embeddings of medical codes (i.e., clinical events) in a manner which takes clinical context into account. Different embedding methods were considered as we did not know which would be the most effective a priori. The three embedding methods employed are Event2Vec, autoregressive, and hybrid embeddings which will be described in Section 3.5.1, Section 3.5.2 and Section 3.5.3 respectively.

[3] CPT is a registered trademark of the American Medical Association, All Rights Reserved

3.5.1 Event2Vec Embedding

Out first embedding approach, which we will refer to as event2vec, is based on word2vec and utilizes a skip-gram style architecture with negative sampling. As discussed in Section 3.2.2, the skip-gram approach is an efficient and effective method for learning dense representations from sequences of tokens where embeddings are learned by considering the context in which each code appears over time. In other words, if the model sees a particular clinical event, say event A, what's the likelihood of seeing clinical event B within a window of size w. We can think of such embeddings as "general purpose" as they are trained in an unsupervised fashion, with no specific task being optimized for in the models objective.

Data for Event2Vec Embeddings

Data used for event2vec embedding training was created in a similar fashion to sequences of words used in the traditional skip-gram method. Specifically, given a sample of a patient's clinical history as described in Section 3.4 (up to one year), which is a sequence of clinical events, and a context window size w, we slide over the sequence in chunks of $2 \times w + 1$ generating $(target, context)$ groups. For each chunk there is at most $2 \times w + 1$ events corresponding to upwards of $2 \times w$ context events (w to the left and w to the right of the target) and a single target event. The number of context events can be less than $2 \times w$ if the target is at a low or high position in the sequence. For example, given a sequence length $l \geq 2 \times w + 1$ and a target w_i, if $i < w$ or $i > l - w$, there will be less than $2 \times w$ context events. Next, each $(target, context)$ group is broken out into (t, c_i) pairs, where t is the target event in each case, and c_i is the i^{th} context event. We chose a window size of 3 for event2vec, resulting in a maximum of 6 context events for each target.

3.5.2 Autoregressive Embedding

In recent years, sequence to sequence (seq2seq) models have gained popularity in language modeling, image captioning, and speech recognition [33, 225, 26]. The nature of the seq2seq framework involves addressing the problem of variable length inputs and outputs [206]. Our second approach explored learning embeddings as a byproduct of training a LSTM, as described in Section 3.2.3, tasked with next event prediction. The training of our LSTM variant is similar to autoregressive language model pretraining, wherein the pretraining objective is to correctly predict the next token in a given sequence from left to right. The key difference is that the vocabulary in our case is a collection of clinical events represented by medical codes, as opposed to a language such as English as is commonly the case in language model pretraining. As such, we refer to the learned

representations as "autoregressive" embeddings.

The seq2seq learning strategy includes an encoder (e.g., embedding layer or full encoder network), where raw inputs are converted to fixed-length, dense, vectorized representations. To further improve the model, we incorporate seq2seq attention, as discussed in Section 2.5.1, which serve to assist the predictive task [7]. We utilize the LSTM architecture with attention for the purpose of next-clinical-event prediction. First, the sparse data representation of a medical code (one-hot encoding) is passed to an embedding layer. Next, the resulting dense representation is fed into a bidirectional LSTM layer which further encodes the data. Then, the attention mechanism calculates scores (coefficients) which are used to construct a linear combination of the output vectors. Finally, the raw output scores are passed through a softmax activation to normalize them into probabilities, which are used to make a final prediction.

Data for Autoregressive Embeddings

For the purpose of autoregressive embeddings, it's acceptable to use multiple examples from a single patient, given that the patient has an acceptably large number of events in their clinical history. In other words, in order to inflate the number of examples we have for training, we can specify a fixed sequence length, and cut each patient's full clinical event sequence, descbed in Section 3.4, into many sub-sequences. To accomplish this, we specify a fixed window size w, corresponding to the desired sub-sequence length, and slide the window over a patients year long clinical history, generating $(\mathbf{x}, y)$ pairs of training samples in the process where $\mathbf{x} = [x_i, x_{i+1}, ..., x_{i+w}]$ and $y = x_{i+w+1}$. That is, each $\mathbf{x}$ is a sequence of w clinical events and y is the next event in the sequence. We used a value of $w = 25$ in this experiment, but in general the choice of w can be thought of as a hyperparameter, and determined empirically through experimentation. Patients with year long clinical histories consisting of less than $w + 1$ events were left-padded with zeros, where the last event in the sequence would be used as the target. The preprocessed data was split randomly by patients into a training set (80%), a validation set (10%) used for hyperparameter tuning, and a test set (10%) used for final model evaluation. This dataset was also used for the experiments discussed in Section 3.7.1.

3.5.3 Hybrid Embedding

Lastly, we explored combining event2vec and the autogressive embeddings, which we refer to as hybrid-embeddings. This was done in part to evaluate the potential to improve a subsequent classification task via transfer learning, discussed in Section 2.7, as well as to asses the overall effectiveness of combining different representations. We combined the event2vec embeddings learned from the initial skip-gram style model with a new, untrained embedding layer as part of a

sequence model for next event prediction. For simplicity, we used the same architecture and data as the autoregressive model described in Section 3.5.2, but replaced the embedding layer with two concatenated embeddings. The first was the frozen embeddings learned by the event2vec model, and the second was left free to be learned as part of the overall network optimization. The network architecture with a hybrid embedding layer is shown in Figure 3.5.

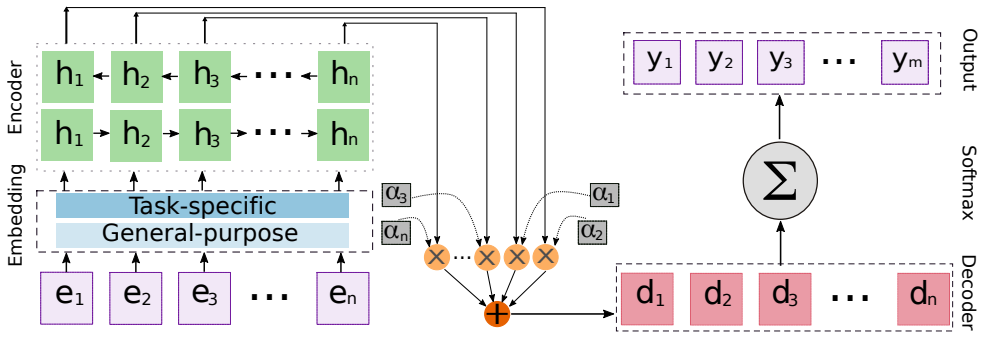


Figure 3.5: The network architecture takes sparse medical codes as input, followed by a hybrid embedding layer: pre-trained event2vec embeddings and autoregressive embeddings. The encoder is a bidirectional LSTM with attention. The output layer is passed through a softmax function to normalize the scores into probabilities.

3.6 Analysis of Embedding Methods

3.6.1 Analysis

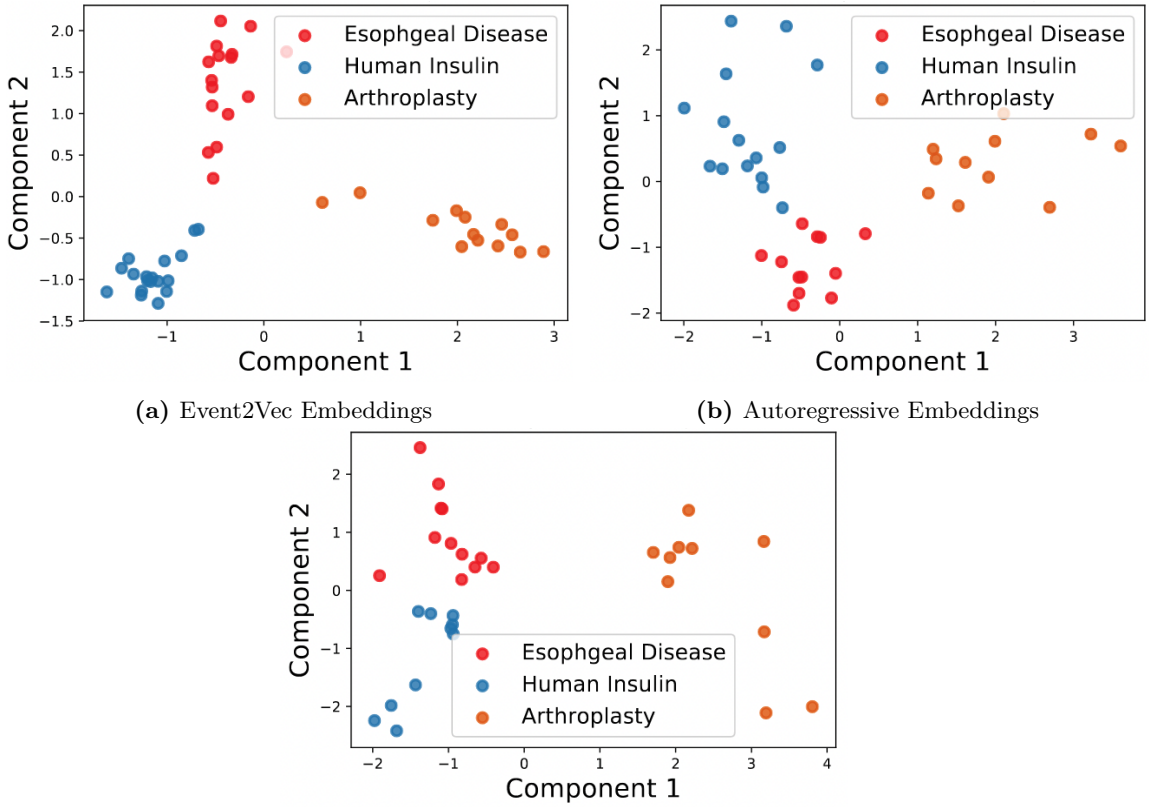
Empirical evaluation of each embedding method was performed with the help of a clinical SME[4]. To allow for a focused assessment of the individual embedding methods, one example from each clinical event category was selected for closer analysis: Esophgeal Disease was chosen as an example diagnosis, Human Insulin for prescription, and Anthroplasty for procedure. In order to obtain different views of the learned embeddings we sanctioned two analysis from our clinical SME. Both SME aided analysis were manual in nature, and designed to determine if the representations made semantic sense to a human with expert knowledge of the domain.

Identification of Clusters

The first clinical analysis required the SME to provide a list of 10-15 events, not necessarily from the same category and preferably non-obvious, that an expert would deem "related" to each example

[4]Analysis of clinical event embeddings was done with the help of by Dr. Malhar Jhavei, Internal Medicine Resident at University of Michigan Health

(anchor) event. To visualize the results, we applied Principle Component Analysis (PCA) [106] to the embedded representations of each example in order to project the examples, and each of the "related" event embeddings to $2D$ for visualization. The resulting plots with respect to each model, and each example can be seen in Figure 3.6a (Event2Vec), Figure 3.6b (autoregressive), and Figure 3.6c (hybrid). We see that the embeddings for each example group form obvious clusters, implying that each of the methods was able to learn strong relationships within, and across domains.



(a) Event2Vec Embeddings

(b) Autoregressive Embeddings

(c) Hybrid Embeddings

Figure 3.6: PCA plots of "most similar" events by embedding method.

Similarity Analysis

For the second embedding analysis we leveraged the same SME, and tasked them with determining the relevance of clinical events identified by the model. That is, we first selected the top-5 "most similar" events to each example by cosine distance between embeddings, and then asked the SME to determine the relevance of each candidate event to its corresponding anchor event. One notable finding is that "similar" events to Esophgeal disease and Human Insulin are closer to each other as compared to Arthroplasty procedures. This result would make sense to a SME familiar with

studies such as [224], which have shown that diabetes, in many cases, carries the risk of Esophgeal disease. In contrast, the relevance of this analysis would not be at all clear to someone tasked with analysis of the data without such expertise. It is notable that the distinction between the clusters for Esophgeal and Human insulin with Arthroplasty in the hybrid embedding model is even more pronounced, suggesting that this model may have learned a slightly more effective, or at least nuanced representation of the data.

Esophgeal Disease		
Event2Vec	**Autoregressive**	**Hybrid**
Gasduo Ulcer	Other stomach disease	Gasduo Ulcer
Gastritis	Gastritis	Anti-ulcer
Proton Pump Inhibitor	Gasduo ulcer	Gastritis
Other stomach disease	Pancreas disease	Other stomach disease
Esophageal dilatation	Anti-ulcer	Nausea/vomiting

Table 3.1: "Top-5 Most Similar" clinical events with respect to a diagnosis of Esophgeal Disease.

To supplement the before mentioned figures, the top 5 most similar code descriptions by example for the event2vec, autoregressive, and hybrid embeddings models are shown in Table 3.1, Table 3.2, and Table 3.3 respectively. The tables serve to reinforce the conclusion that all three methods learned representations that contain valid semantic relationships.

Human Insulin medication		
Event2Vec	**Autoregressive**	**Hybrid**
Needles/Syringes	Needles/Syringes	Sodium-Glucose Co-Transporter
Diagnistic Tests	Diabetic Other	Needles/Syringes
Diabetic Other	Diagnostic Tests	Biguanides
Incretin Mimetic Agents	Diabetes Mellitus with Complications	Sulfonylureas
Diabetes Mellitus with Complications	Sodium-Glucose Co-Transporter	Antidiabetic

Table 3.2: "Top 5 Most Similar" clinical events with respect to a prescription for human insulin

3.6.2 Summary

In this section we investigated the effectiveness of various techniques for learning embedded representations from medical codes. Validation of the results with respect to each embedding method was done by a clinical SME, confirming that all of the embedding methods showcased in this section captured meaningful, yet slightly different relationships within, and across domains.

Arthroplasty Procedure		
Event2Vec	**Autoregressive**	**Hybrid**
Hip Replacement	Hip Replacement	Hip Replacement
Arthroplasty other than hip or knee	Arthroplasty other than hip or knee	Arthroplasty other than hip or knee
Other therapeutic procedures on musculoskeletal system	Treatment, fracture or dislocation of hip and femur	Other therapeutic procedures on joints
Other therapeutic procedures on joints	Arthroscopy Knee	Arthroscopy Knee
Arthroscopy Knee	Other therapeutic procedures on joints	Excision of semilunar cartilage of knee

Table 3.3: "Top 5 most similar" clinical events with respect to an Arthroplasty procedure (i.e., the surgical reconstruction or replacement of a joint).

Moreover, since each embedded representation is slightly different, combining these approaches yielded a unique, relevant view into the underlying connections between clinical events.

3.7 Clinical Predictive Models

In this section we will evaluate the effectiveness of our clinical event representations on two tasks. The first task we evaluate is next clinical event prediction, where a clinical event is defined as a diagnosis, procedure, or prescription, represented by a standard medical code (as in Section 3.5). Second, we use our general purpose embeddings as input to a model tasked with predicting non-urgent Emergency Room (ER) visits. While the task of next event prediction requires outputting a distribution over the vocabulary of clinical events, here we are concerned with predicting the likelihood of a single event (which is not part of the input vocabulary), hence the output is a single score.

3.7.1 Next Clinical Event Prediction

In NLP, the performance of a next-token prediction model is routinely evaluated based on whether the true target exists in the $top-k$ most probable outputs. This is done in part due to the high dimensionality of the vocabulary being used, as well as the fact that there can be multiple plausible answers. That is, if we have a vocabulary consisting of $10K$ tokens, the output of the model will be a probability distribution over $10K$ positions, where each position corresponds to a token in the vocabulary. The model could then output the token corresponding to the highest probability. The issue is that with a large vocabulary, the distribution is spread very thin over the many potential tokens. Assigning a winner by $top-k$ allows for a range of tokens to be considered, as opposed to

a single token. This is certainly the case with medical data, which is sparse, high dimension, and therefore challenging to predict precisely down to the individual code. As a result, we report on top-5, top-10, and top-20 accuracy for each model.

All evaluation metrics shown below are with respect to a held-out test set, which was not used in training or hyperparameter tuning. Hyperparameters, such as initial learning rate and regularization coefficients, as well as network architecture were tuned via random search, using the the validation set for intermediate evaluation. Table 3.4 shows evaluation of the autoregressive model, which is a "next clinical event" prediction model using Bidirectional LSTM with attention.

Top N Accuracy (Autoregressive)	
N	**Accuracy(%)**
5	61.31
10	72.12
20	81.09

Table 3.4: Top N accuracy for autoregressive model

Table 3.5 illustrates the performance of the "next clinical event" prediction model using hybrid-embeddings. These preliminary results indicate that combining the event2vec and autoregressive embeddings helped the overall performance compared to using the autoregressive embeddings alone, although the performance is fairly close, and further investigation would be needed to be fully confident in this conclusion.

Top N Accuracy (hybrid embedding)	
N	**Accuracy(%)**
5	62.33
10	73.22
20	82.20

Table 3.5: Top N accuracy for hybrid embedding model.

3.7.2 Predicting Non-Urgent ER Visits

Non-urgent ER visits are typically defined as visits for conditions for which a delay of several hours would not increase the likelihood of an adverse outcome [209]. Since non-urgent ER visits are defined by a collection of possible medical codes along with a temporal factor, they are not represented in the data used to train the embedding models. For each patient in the train, validation

and test set we pulled three months of medical history (via claims data as described in Section 3.4), as well as an indicator of whether a non-urgent ER visit occurred in the subsequent two week period.

For the task of predicting non-urgent er visits, we compared several models: First, we trained a traditional clinical predictive model using a standard FFNN architecture with hand-engineered features. Input features were created with the help of our clinical SME, and comprised demographic information, indicators of prior and current clinical events, time-since-last features and more. In total around 500 features were engineered to represent a three month period in a patient's clinical history. It's notable that the final ≈ 500 hand engineered features required a good deal of feature selection and pooling. That is, creating indicator features representing the presence of all diagnosis, procedures and prescriptions would require hundreds or thousands or features as described in Section 3.2.1. To reduce this number we pooled all clinical events that were deemed to be "rare" into a single "uncommon" indicator feature. This was done by pulling a random sample of patients, calculating a percentage, or pseudo-likelihood of each event being in a three month history with respect to the sample, and then bucketing the collection of clinical events with a pseudo-likelihood $\leq 1\%$. The presence of rare events alone can be a powerful predictor, but not all rare events are meaningful. We speculate that information is inevitably lost by bucketing so many events, which is not the case when using clinical event embeddings. Moreover, even with the extreme reduction in hand-engineered features described above, the input layer must still comprise > 500 neurons, which is over twice the size of our clinical event embeddings. It follows that the use of clinical event embeddings not only allows for the nuances of rare events to be retained, but that a comparable network architecture employing clinical event embeddings requires considerably less parameters compared to its hand-engineered couterpart.

Next, we constructed patient level embeddings by aggregating the individual event level embeddings described in Section 3.5. As in the case of hand-engineered features, we aggregated clinical event embeddings over a three month period in a patient's clinical history, which was then used as input to a standard FFNN architecture with three hidden layers of size 128, 128 and 64 respectively. Both FFNN architectures, with the exception of the input layer size, were identical to allow for a fair comparison. Several pooling strategies were attempted including average-pooling, where embeddings are simply averaged, and max-pooling, in which we take the maximum value across embedding dimensions. Max-pooling was selected as the final pooling method.

Lastly, we used our pretrained embeddings as input to a LSTM based sequence classification model tasked with predicting non-urgent ER visits. That is, a LSTM was trained on sequences of clinical events, represented by our pretrained embeddings, to produce a score representing the

"likelyhood" of a non-urgent ER visit. This is in contrast to the patient level embeddings described in the second method above, in which the embeddings were aggregated. For each patient we used a sequence of clinical events representing up to three months of claims data. Each input sequence was left padded (if the history was too short) as needed to achieve a consistent input sequence length of 128. Initial exploratory analysis was done to upper bound the number of events in a three month history at 128. We then used the pretrained embeddings as initial inputs to a LSTM based model, which was trained to output a single score indicating the "likelyhood" of a non-urgent ER visit. The performance of the resulting clinical predictive models, trained with each of the strategies described above, is shown in Table 3.6. We see that performance is similar using a comparable

Model Comparison	
Features	**ROC (AUC)**
FFNN with hand-engineered	0.7427
FFNN with patient embeddings	0.7434
LSTM with event embeddings	0.7531

Table 3.6: Comparison of clinical predictive models trained on the task of predicting non-urgent er visits using (a) FFNN with hand-engineered features, (2) FFNN with aggregated embeddings (i.e., patient-level embeddings) (3) LSTM sequence classification model with event-level embeddings

FFNN architecture with hand-engineered features versus patient level embeddings, although the model with patient level embeddings slightly outperforms its hand-engineered counterpart. What's more notable is that the performance gain was more noticeable when using a sequence model fueled by pretrained, event-level embeddings.

A key takeaway here is that hand engineering features from clinical data is difficult, and time consuming. Hand engineering clinical features for a task like clinical predictive modeling requires a great deal of subject matter expertise to ensure that the resulting features are meaningful, and appropriate for the given task. Moreover, hand-engineered features require significantly higher dimentionality compared to event embeddings, and still likely lose information related to rare, but meaningful events. In contrast, learning useful embeddings requires considerably less subject matter expertise (with the exception of interpretation), and therefore effectively lowers the bar for development of high quality clinical predictive models. In addition, clinical event embeddings are capable of learning semantic relationships that may be previously unknown, even to a clinical SMEs, and can result in a more efficient, powerful network requiring fewer parameters.

Leveraging clinical event embeddings can improve the learning process for a predictive model, since the input features already encompass information about the relationships between data points. Pretrained embeddings can prove especially useful in cases where a SME is not available to aid in

the features engineering process. This is not to say that leveraging effective embeddings will always result in a high quality ML model, and there are many factors that contribute to the formulation of a clinical problem as a ML system, aside from the feature representations alone. It is our opinion that neither deep knowledge and understanding of the underlying clinical problems, or the methods being applied to solve them should be discounted when approaching problems in healthcare AI.

3.8 Conclusion

Embedded representations learned from sparse, high dimensional medical data can be effectively leveraged by a broad range of downstream tasks such as cohort selection, patient similarity mapping, and clinical predictive modeling. In this chapter we took the idea of learning general representations from medical codes a step further as part of a task specific modeling effort. Moreover, we showed that informative general representations can be achieved by combining event2vec and autoregressive embeddings into a hybrid representation of clinical events. Finally, we gave empirical evidence that knowledge learned during general pretraining can be successfully transferred to new tasks, such as clinical predictive modeling, via embeddings which are capable of out-performing their traditional hand-engineered feature counterparts.

3.9 Future Work

The bulk of the work reflected in this chapter was done prior to 2018, and leaves much room for further exploration. One area which was investigated during the time of this research, but that was not fully explored, was how to effectively deal with the problem of unevenly spaced events. Some sequences, such as text, have no inherent concept of time associated with the individual elements, and therefore no "time between" elements. Sequences of clinical events on the other hand are not evenly spaced, which introduces its own set of challenges. For example, how does one interpret the time delta between subsequent events? At the time of this research, we investigated two simple ways to deal with the problem of unevenly spaced events. First, for the purpose of learning clinical event embeddings, we ignored the time component and treated the events as evenly spaced. Second, we modified the approach outlined in Section 3.7.1 by supplementing each input and output event with a corresponding time delta. That is, the model is fed sequences of (e_t, d_t) where e_t is the clinical event at time t, and d_t is the time since e_{t-1} with $d_0 = 0$. The goal of this modified formulation was to predict not only the next clinical event, but the amount of time from the last event as well. While this simple approach was fairly effective at predicting both next clinical event and time delta

simultaneously, we felt that the degradation in next clinical event prediction was significant enough to abandon the join-prediction model in favor of the better performing, single-prediction model described in Section 3.7.1, along with the clinical event prediction model from Section 3.7, which predicts a specific clinical event within a predefined time window.

Chapter 4

Deep Learning Approaches to RSWA Event Detection

4.1 Motivation

Rapid eye movement (REM) sleep is a stage of sleep that features random and rapid movement of the eyes, low muscle tone or atonia (extremely relaxed muscles), and, commonly, vivid dreaming. All other stages of sleep are collectively referred to as "non-REM" (NREM) sleep. REM and NREM sleep alternate within ultradian sleep rhythms, each cycle of which lasts about 90 minutes in an adult human. REM sleep behavior disorder (RBD) is a parasomnia (disruptive sleep-related disorder) characterized by repeated episodes of dream enactment behavior and REM sleep without atonia (RSWA). RSWA is detectable during polysomnography (PSG) recording [143], and is typically diagnosed through the detection of individual events indicative of the disorder from PSGs.

The standard way to identify RSWA events from PSGs is through manual inspection by a certified sleep technician, or otherwise qualified clinician. The process of manually annotating PSGs is costly, time consuming, and prone to human error. Sleep technicians identify individual RSWA events by adhering to a set of standardized rules governed by the American Association for Sleep Medicine (AASM). However, multiple technicians analyzing the same PSG may not yield a consistent labelling. That is, a certain amount of inter-labeler disagreement is common when manually inspecting many channels of time-series data in an overnight study, even among highly trained technicians. Such inter-labeler disagreement could be indicative of the inconsistencies that human labelers display when applying a predefined set of rules to multidimensional data. Moreover, there may be more complexity to the task of identifying RSWA events that cannot be captured by applying a simple set of rules. Inconsistencies among human labelers is a serious issue as accurate identification of RSWA, and diagnosis of RBD by extension, is crucial for proper treatment planning.

The design and implementation of effective, efficient, and accurate machine learning solutions for the problem of RSWA event detection has the potential to significantly reduce human hours, human error, and ultimately, overall healthcare costs. A direct result of more efficient, less costly RSWA assessment and diagnosis is increased identification of effected patients across a broader patient population. Furthermore, a diagnosis of RSWA is thought, within the sleep medicine community, to be correlated with an increased risk of neurodegeneartive disease such as Alzheimer's and Parkinson's later in life. More treatment options are available to patients who suffer from such diseases if they can be anticipated, or diagnosed early on [182, 44]. It follows that the efficient and accurate detection of RSWA related events is critical for providing the highest level of care to the affected patient population.

The goal of this chapter is to provide automated and reproducible quantification of RSWA to establish more objective criteria for analysis. Specifically, this work proposes a novel method for RSWA event detection from PSGs using DNN based approaches[1]. Previous supervised learning approaches related to RSWA, or by extension RBD, attempted to predict only binary, patient level diagnosis. In contrast, this work presents methods to predict the individual locations within EMG sequences (signals) where phasic (P) or tonic (T) activity is present. We refer to such events as "signal-level" RSWA events. We speculate that RSWA event detection is preferable to binary diagnosis prediction for several reasons: first, RSWA event detection is more robust, flexible, and generally useful as compared to binary diagnosis prediction since RSWA events, once detected, can be converted into patient level diagnoses by subsequent application of the AASM guidelines [13]; next, RSWA event detection allows for the quantification of event duration and frequency. By extension, the quantification of event duration and frequency can in-turn help to quantify disease load and severity, as well as provide useful statistics for downstream tasks such as population studies. Finally, a model capable of RSWA event detection that aligns with expert annotations may be more trusted by clinicians, and therefore more widely used compared to models that use obtuse, or hidden features to directly predict binary diagnosis.

[1]The work in this chapter was previously published in [215]: P. Wallis, D. Yaeger, A. Kain, X. Song, and M. Lim, Miranda Automatic Event Detection of REM Sleep Without Atonia From Polysomnography Signals Using Deep Neural Networks in IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) 2020

4.2 REM Sleep Behavior Disorder

4.2.1 RBD as an Early Warning

RBD is a serious clinical diagnosis with significant implications. It can lead to dream enactment, from simple limb twitching to violent behaviors, sometimes resulting in injuries [194]. RBD is widely considered a prodromal synucleinopathy, one of the earliest symptoms of the family of neurodegenerative disorders which include Parkinson's Disease, Lewy Body Dementia and multiple system atrophy [195, 196, 99, 143]. To put this in perspective, in some studies nearly 50% of patients with Parkinson's Disease have RBD [71]. In Postuma et. al [173], patients showed an overall conversion rate of 6.3% from RBD to an overt neurodegenerative syndrome, with 73.5% of individuals converting after a 12 year follow up. It follows that symptoms of RBD may begin years or decades before the onset of a neurodegenerative disease. Moreover, RBD diagnosis may provide insights into the development of neurodegenerative diseases, as is invaluable with respect to early detection.

RBD affects 1–3% of the population [74, 197, 199], and is particularly prevalent in patients with trauma, e.g., traumatic brain injury and post-traumatic stress disorder, both of which are particularly common in military veterans [226, 55].

4.2.2 RBD Diagnosis

Currently, a diagnosis of RBD is characterized by both having a report, or video observation of dream enactment, and observing events related to RSWA, such as excessive chin and limb muscle activity, in a PSG. Muscle activity is divided into two primary classes: *phasic* (P), which is associated with short bursts of activity, and *tonic* (T), which means longer, sustained segments of activity [143]. Within the context of PSGs, and sleep studies in general, an "epoch" is defined as a short interval of arbitrarily defined length (usually 20-60 seconds), but is commonly associated with a 30 second interval as is the convention in sleep staging. With respect to RSWA event detection, the American Association for Sleep Medicine (AASM) standard requires that, within a single 30-second epoch of REM sleep, at least 5 of 10 mini-epochs of 3-second duration contain phasic muscle activity (P events), or at least 15 seconds contain sustained tonic muscle activity (T event) for the epoch to be classified as abnormal [13]. By extension, a single abnormal REM epoch in combination with dream enactment garners the diagnosis of RBD.

As previously noted, the current gold standard for diagnosis of RSWA is through manual, visual scoring of PSGs by a trained clinician which is labor-intensive, costly, time-consuming, and error prone. For example, it can take many hours for a qualified sleep lab technician to annotate a single

PSG, and agreement amongst scorers in labeling signal-level T and P events is low [15]. It follows that the development of an accurate, effective, and automated method for detecting events related to RSWA would allow for faster, less costly, reproducible and more objective diagnosis of RBD.

4.3 Previous Approaches and Related Work

Several semi, or fully-automated methods have been previously proposed for RBD detection. These approaches fall into two general categories: rule-based, such as a traditional algorithmic approach which implements the existing AASM definition, and machine learning based, which includes flexible supervised methods such as support vector machines (SVM), random forest (RF), and DNNs. It's worth emphasizing that many methods for automated RBD detection predict final diagnoses, not signal-level events. While the task of predicting a final diagnosis is a more straightforward approach, it's also less robust, and less flexible compared with predicting individual signal-level events which can in-turn be used to make clinical diagnoses.

4.3.1 Rule-Based Methods

There have been numerous attempts over the years to develop a rule based system for diagnosing RSWA, and more generally RBD. We will discuss several such methods in more detail.

STREAM

In 2007, Burns et al. [21] proposed a rule-based method for identifying PSG evidence of RBD. For the purpose of algorithmic development and assessment, they collected data from 23 subjects: 17 with neurodegenerative disorders including 9 with probable or possible RBD, and 6 controls. Qualified medical professionals visually scored two PSGs from each subject for the purpose of comparing the effectiveness of the rule-based approach versus expert human labeling. The algorithm consisted of three primary steps: first, each 30 second epoch was divided into 10 three-second mini-epochs; next, variance of the chin EMG was calculated with respect to each mini-epoch; finally, variances during REM sleep were compared to a threshold defined by variances during quiet NREM sleep. The job of the algorithm was to calculate a metric which the authors refer to as Supra-Threshold REM EMG Activity Metric (STREAM), which was calculated for each subject. STREAM is defined as the percentage of all REM mini-epochs with variance above the pre-defined threshold, and was found to be highly correlated with the manual, visually-derived score for RBD severity (Spearman $\rho = 0.87$, $P < 0.0001$). With an optimal cutoff, STREAM was able to identify probable or possible RBD with 100% sensitivity and 71% specificity. While the authors looked at

several metrics for evaluating the effectiveness of STREAM, with respect to the task of diagnosing RBD the authors used area under the receiver operating curve (AUROC), for which STREAM achieved a score of 0.84.

Although the evaluation results reported in Burns et al. seem impressive, the study was conducted with a very small sample (23 subjects in total). Cesari et al. [24] did a comparison of computerized RSWA detection methods, including STREAM, using a sample of 146 subjects including 27 healthy control patients and reported average values for sensitivity, specificity and accuracy of $\approx 70\%$ when employing STREAM, which is significantly less than the 100% sensitivity reported in the original paper.

Sleep Atonia Index

Sleep Atonia Index (SAI) is a metric which was proposed by Ferri et al. [58, 59], who conducted a study to quantitatively evaluate the submentalis muscle (chin) EMG activity during sleep in patients with RBD, or with RBD and multiple system atrophy (MSA). Ferri et al. suggest SAI should be used in conjunction with the other criteria for clinical evaluation, and for patient diagnosis with respect to RBD. The study included 21 patients with RBD, 10 patients with MSA, as well as 34 controls. The over arching goal of Ferri et al. was to provide practical indices for EMG activations, and for the objective evaluation of EMG atonia during REM sleep. The authors calculated the percentage of 1-second segments of REM sleep with rectified amplitude greater than 1 mV. The average amplitude of the rectified chin EMG was used to develop a SAI, which was able to clearly distinguish REM from NREM sleep in normal controls, with values very close to 1 in young normal subjects and only slightly (but significantly) lower in age-matched controls. RBD patients showed an additional, significant decrease in SAI; MSA patients showed the lowest values of REM SAI, which were distinguishable from those of normal controls and of RBD patients. The distribution of the duration of chin activations was mono-modal in all groups, with RBD patients showing the highest levels.

Frauscher Scoring Algorithm

Frauscher et al. [62, 63] developed a rule-based algorithm to score polysomnographic EMG signals according to the Sleep Innsbruck Barcelona (SINBAR) group criteria with the goal of detecting RSWA. Researchers collected and analyzed the PSGs of 20 patients with RBD and 60 healthy volunteers. Motor activity during REM sleep was quantified manually, and algorithmically according to the SINBAR criteria for the mentalis ("any", phasic, tonic EMG activity) and the flexor digitorum superficialis (FDS) muscle (phasic EMG activity). Computer-derived indices for "any," phasic, tonic

mentalis EMG activity, phasic FDS EMG activity, and the SINBAR index, which is "any" mentalis + phasic FDS, correlated well with manually derived indices yielding a Spearman $\rho \in [0.66, 0.98]$. In contrast with computerized scoring alone, computerized scoring plus manual artifact correction (median duration 5.4 min) led to a significant reduction of false positives for "any" mentalis (40%), phasic mentalis (40.6%), and the SINBAR index (41.2%).

Conclusion

While some of the existing automated methods for RBD detection discussed in the preceding sections showed impressive evaluation metrics for their respective test populations, all were developed using very small datasets with fewer than 40 participants. When applied to larger datasets, the same rule-based methods yielded less impressive results, with none of the before mentioned methods achieving specificities and sensitivities above 80% in diagnosing RBD [24]. We speculate that the task of detecting RBD is more complicated and diverse across larger populations, hence a simple rule-based approach is not robust enough to accommodate such a range of cases. While the performance of rule-based methods developed on large, diverse data is not known, it seems reasonable to assume that scaling up the data would also require scaling the number of rules, making the algorithm difficult to maintain. It follows that, given a population of adequate size, ML approaches are better suited to the task of RBD detection in complex, real-world populations.

4.3.2 Machine Learning Methods

In this section we look at previous applications of ML in sleep medicine. Up to this point such applications had been focused on RBD diagnosis and sleep stage classification, not RSWA event detection.

One-Class SVM

Kempfner et al. [108, 109] conducted a study on a sample comprising 16 subjects with idiopathic REM sleep behavior disorder, 16 subjects with periodic limb movement disorder, and 16 healthy control subjects. While the study proposed a semi-automatic algorithm for the early detection of Parkinson's disease, it was not explicitly designed for RSWA event detection. Kempfner et al. claimed that RSWA could be detected by distinguishing between normal REM sleep and RSWA by considering muscle activity as an outlier detection problem. Different combinations of five surface electromyographic channels, including the EOG, were tested. A muscle activity score was automatically computed using manually scored REM sleep, by way of a one-class SVM trained with subject-specific features. Kempfner et al. then computed the mean of each three-second mini-epoch

during REM sleep relative to the minimum of a surrounding 30 second window and using a one-class SVM, and the percentage of min-epochs classified as outliers were used to differentiate patients with an AUROC of 0.99. Similar to the examples of rule-based systems described in Section 4.6.1, this study was conducted using a very small sample, and results reported therein may not generalize well to larger populations.

Random Forest

Cooray et al. [38] developed a RF based approach for predicting RBD diagnosis. For model training, a total of 156 features were hand engineered from EEG, EOG, and EMG signals. Analysis was performed using PSGs from 53 participants with RBD, and 53 age-matched healthy controls. RBD diagnosis requires analysis of a patient's PSGs during REM sleep. In order to remove the need for manual sleep stage annotation, Cooray et al. trained an RF classifier on the task of sleep stage classification which achieved a Cohen's Kappa score of 0.62. The sleep stage and RBD classifiers used a similar set of input features, and differed primarily by target. While manual annotations can be more time consuming and expensive to obtain compared to a classification model, Cooray et al. found that RBD diagnosis accuracy improved from 86% to 96% when using manually annotated sleep staging. It follows that automatic sleep stage classification by way of this particular RF classifier significantly hindered performance with respect to the primary task of RBD diagnosis.

The classification accuracy reported by the authors sounds very good, but there are several factors to be considered: First, it is important that the test set class distribution is representative of the actual population when assessing the effectiveness of a ML model. An even binary class split is not common in the real world where the proportion of positive examples may be very small. Second, when a target distribution is highly skewed, as is the case for RBD diagnosis, accuracy can be a very deceptive metric. There are many metrics more appropriate for evaluating a classifier with skewed target data, such as precision-recall, and $F - measure = \frac{2*Precision*Recall}{Precision+Recall}$. Lastly, this study employed a very small sample where the number of subjects was less than the number of input features to the model (i.e., 106 subjects with 156 features for each). In other words, the dimentionality is higher than the sample size so the training data cannot provide an adequately representation of the domain, and the model would be very prone to overfitting. In such a situation it would be advisable to either increase sample size (e.g., resampling), reduce the dimentionality (e.g., PCA), or performing a preliminary feature selection. In either case, 106 examples is a very small sample for ML model training, validation and testing. A follow up analysis would be required to assess the degree to which the model overfit to the data, as well as how generalizable the trained model would be given a larger, more diverse population.

Deep Neural Networks

Up to this point, the sleep medicine and machine learning research community has not focused on designing systems to automatically detect individual RSWA events. With that said, there have been several studies which used DNNs for automatic sleep stage classification.

Yildirim et al. [227] successfully trained a DNN, composed primarily of 1D CNN blocks, to classify sleep stages with a high degree of accuracy. Data used for this study was taken from two open source sleep datasets: sleep-edf and sleep-edfx [172]. The first dataset consists of PSG signals of eight healthy males and females, while the second dataset (extended version) contains PSG records for 61 subjects in total. This study considered sleep stage classification with five classes, W (wake), S1, S2, SWS and REM, as well as six classes, where the single class SWS was replaced with S3 and S4. The network architecture employed in this study comprised several 1D CNN blocks, each of which contained two 1D conv layers, followed by a max-polling layer. The first block in the network used dropout following the max pooling layer, and the last block was flattened and fed into two additional dense layers. The researchers in the study used the following combinations of PSG signals to train the network: single-channel EOG, single-channel EEG, and single-channel EOG + single-channel EEG. Hyperparameter tuning was performed via brute force. The network performance showed high sensitivity for the W class, which was by far the dominant class in both datasets. Reported sensitivities for the remaining classes were 0.48, 0.84, 0.77 and 0.81 for the five class, and 0.41, 0.89, 0.35, 0.78 and 0.84 for the six class network respectively.

Malafeev et al. [141] outfitted a DNN composed of 1D CNN blocks, similar to that of Yildirim et al., but with a RNN layer to classify sleep stages. In this study Malafeev et al. used two datasets: the first contained 54 whole night sleep recordings of healthy participants, and the second consisted of 22 whole night sleep recordings, as well as 21 recordings of a multiple sleep latency test (MSLT) for patients with sleep disorders. Malafeev et al. tried two CNN - LSTM architectures: the first took as input a single-channel EEG which was passed through a series of 1D CNN blocks, the last of which was fed into an LSTM which output a probability distribution over five target classes (W, S1, S2, S3 and REM). The second architecture used single-channel EEG, EOG and EMG signals as input. The EEG and EOG were passed through separate 1D CNN encoder branches with residual connections, which were discussed in Section 2.4.3. The encoded EEG and EOG signals were then concatenated together along with the original EMG signal, and passed to a final LSTM which output probabilities over the same five classes. For the purposes of this study each epoch was considered to be independent, not taking the temporal structure of sleep into account. The two DNN architectures were evaluated for each sleep stage via Kohens Cappa. Evaluation

results varied considerably by population, showing best results for the healthy population, and lowest results for the patient group. In addition, the models also showed variable performance with respect to sleep stage, with stages W, S2, S3 and REM all receiving relatively high Kohens Cappa scores, while stage S1 consistently scored significantly lower, indicating that some sleep stages may be considerably harder for a DNN of this type to classify. Malafeev et al. noted that training on a mixture of the two datasets resulted in an increased performance, which suggests that the underlying population is diverse, and therefore better generalization and overall performance is achieved when the model is trained on a more representative, diverse dataset.

We took inspiration from Malafeev et al. when constructing several of the architectures used in our work. Specifically, the second architecture discussed above employed a separate encoder for the EEG and EOG signals. Several of the model we present in Section 4.6.2 utilize a similar technique, which we refer to as "multi-branch" encoding.

4.4 Data Description

The overnight PSG data used in this chapter was collected from an approved protocol performed according to the Declaration of Helsinki with approval of the Veterans Affairs (VA) Portland Health Care System Institutional Review Board (#3641). All participants provided verbal and written informed consent prior to participation. Participants in this study were US Veterans and enrolled prospectively in a cross-sectional manner through the VA Portland Health Care System Sleep Clinic [153]. Data collection was not done as part of this body of work, but was made available through a partnership with the Portland VA hospital to be used for scientific research and development related to automatic detection of RDB related events.

The final dataset used in this work consisted of 843 anonymized PSG studies. Nineteen channels were recorded in all PSG studies including standard electroencephalography (EEG) and electrooculography (EOG) channels, as well as EMG recordings of the submentalis (chin), and the left and right anterior tibialis (L leg and R leg). Single-channel EMG, EOG, and EEG signals were all sampled at 200 Hz. Sleep staging and annotation of respiratory events, mainly consisting of apneas and hypopneas, were performed by a certified sleep technologist according to AASM scoring guidelines [13]. Annotation of EMG recordings for P and T events were performed by a trained research assistant under the supervision of a board certified sleep physician,[2] according to AASM scoring guidelines [13]. P events were defined as short bursts of elevation in EMG tone

[2]Dr. Miranda Lim was the board certified sleep physician that oversaw this project. Dr. Lim was the head of the sleep lab at Portland VA Hospital at the time of this research, and oversaw the work conducted during this time.

lasting 0.1–5 seconds with an amplitude of at least four times that of the baseline during REM sleep. T events were defined as tonic elevation in EMG tone higher than the NREM baseline lasting longer than 5 seconds. Baselines were gathered via AASM guidelines, and are discussed further in Section 4.6.1. While REM and NREM segments of sleep are needed to calculate baselines for P and T events respectively, actual P and T events can only occur during REM sleep. The beginning and end of T events were always labeled. All P events had a labeled beginning, but $\approx 60\%$ of P events lacked a labeled ending due to a problem with the software used by the sleep technician to manually annotate RSWA events. More specifically, for P events, the quick burst of activity could appear to be almost instantaneous to a human labeler, therefore a P event annotation could be logged as a single location pointer within a PSG, without a corresponding end time. In the case of seemingly instantaneous P events, the event would be logged with a start time and no end time by the software, as opposed to simply setting the end time to be the same as the start time to indicate that the associated duration was zero. A strategy for accommodating a slight uncertainty in human labeled events will be discussed in Section 4.5.

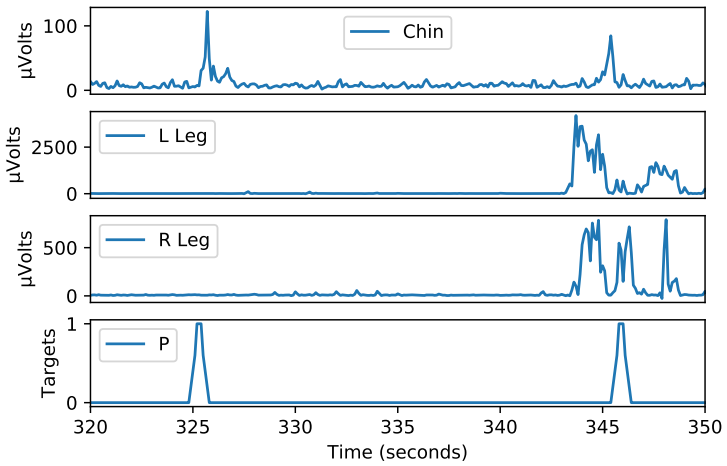
Of the 843 total PSGs that were collected, 802 met the inclusion criteria of being at least 4 hours long, and containing at least 5 minutes of REM sleep. Of these 802 PSGs, 110 were subsequently excluded due to data corruption issues. The remaining 692 studies were partitioned into train, validation and test sets with the following split: 554 patients were used for training ($\approx 80\%$), 78 patients were designated for validation ($\approx 11\%$), and the remaining 60 patients were held out for testing ($\approx 9\%$). Each PSG study contained a average of 8.3 ± 5 (mean$\pm$std) unique REM periods, each of which lasted an average of 420 ± 495 seconds. Across the dataset, there were 3,915 labeled T events that lasted 5.6 ± 11.1 seconds on average, and 13,600 labeled P events with a mean duration of 0.6 ± 0.7 seconds. P events where the duration was not entered were assigned a duration of 0 seconds, and $\approx 5\%$ of PSG studies contained no labeled P or T events. Only REM regions, and up to two minutes of NREM sleep immediately preceding a REM region were used as input to the models discussed in Section 4.6. We discarded the first REM regions from a PSG study if they were not preceded by NREM sleep of any duration, which occurred in 3 of the 692 PSGs.

4.5 Data Preprocessing

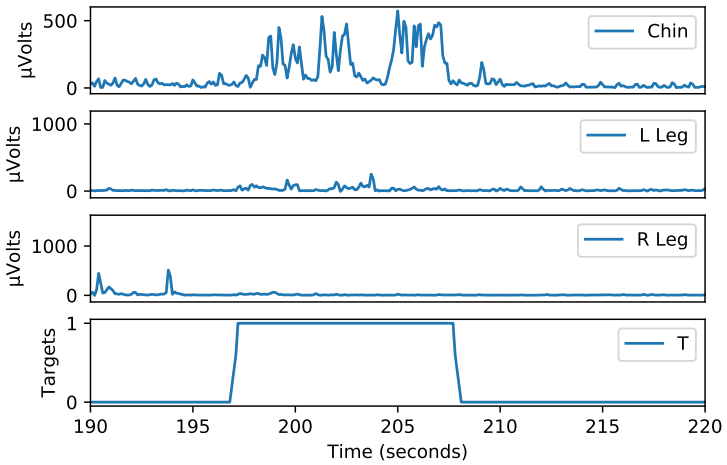
EMG signals were filtered using a zero-phase FIR high-pass filter with a cut-off frequency of 20 Hz. Signals were further processed by rectifying the chin, L leg, and R leg EMG signals [58, 108, 109], and down-sampling to 10 Hz for all channels. Overlapping labeled P and T events were rare ($< 0.01\%$), and P events that overlapped with T events were dropped. Outputs were 3-dimensional

vectors representing probabilities of each event class: "no event," "P," and "T."

One-hot encoding were not used in order to accommodate a software issue associated with P events having duration between 0 and 1 second, and also to reflect the timing imprecision of human labelers. The before mentioned issues were addressed by assigning a probability to each event, which was linearly ramping up or down within an ϵ-neighborhood of the labeled event. The value of ϵ was determined empirically, and set to $\epsilon = 0.1$ seconds across experiments. The ramp-up / ramp-down around labeled events can be seen in Figure 4.1).



(a) Example-P



(b) Example-T

Figure 4.1: EMG signals and targets. For both sub-figures, Chin, L leg, and R leg rectified EMG signals are shown in the top three panels. Target probabilities are shown in the bottom panels.

4.5.1 Artifact Reduction

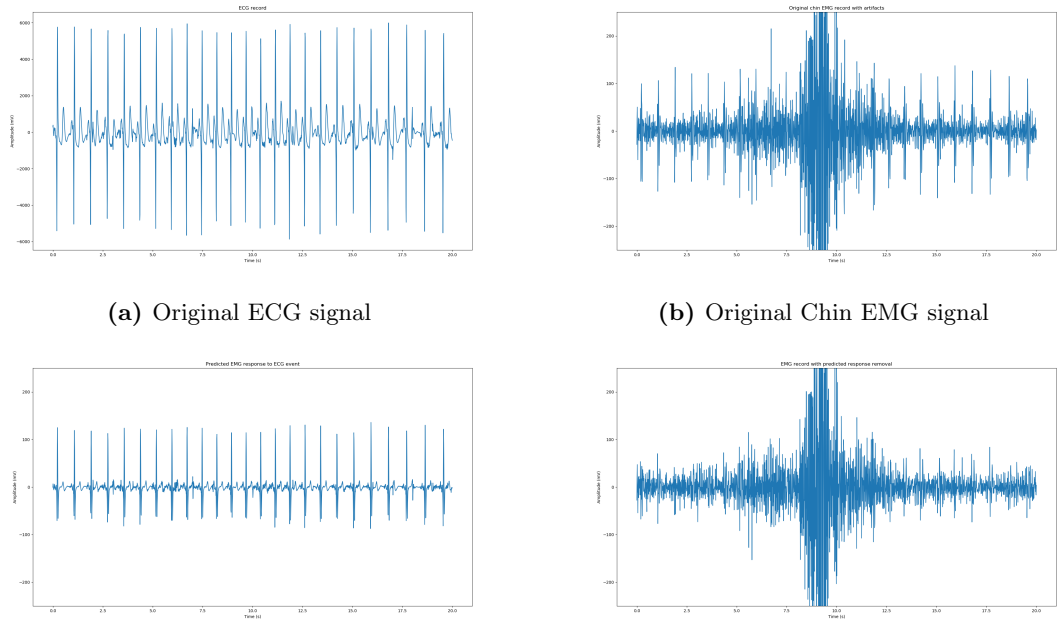
During preliminary data analysis we observed the presence of artifacts in EMG signals that appeared to be rooted in the ECG signal. That is, the heart signal was noticeably present in EMGs. In order to mitigate this effect, we developed a method to reduce these artifacts as follows: Let x be the source signal that drives the appearance of artifacts in a given channel z. In our case, x was the ECG signal, and z was any one of the EMGs. We assumed that there existed an artifact signal a that was added to the artifact-free signal $\tilde{z}$ such that the observed signal $z = \tilde{z} + a$. Furthermore, we assumed that the artifact in question was the result of a linear convolution of the source signal x with impulse response h, or $a = x * h$. In regions where z shows no muscle activity, we can assume that z contains only noise, and thus we can solve for h via a finite impulse-response system identification approach using least-squares in the time domain [177]. We adaptively calculated h on NREM regions immediately preceding REM regions. The length of time used to calculate h was treated as a hyper-parameter, and chosen as the minimum value that achieved maximum performance on validation data using a metric that measured the decrease in overall root-mean-squared (RMS) energy. The average "best" duration was found to be approximately 80 ms. Finally, the artifact-reduced signal in a subsequent REM region was estimated as $\tilde{z} = z - x * h$ (see Figure 4.2).

To evaluate the proposed method, we processed 164 regions of REM sleep with 0.5–2 minutes of preceding NREM sleep, and found that, on average, RMS energy was reduced by 35.4±13.9, 2.0±2.4, and 3.6±2.6 percent for the chin, L leg, and R leg channels respectively. We speculate that the greater distance between the heart and the legs relative to the distance between the heart and the chin contributed to the difference in severity of artifact influence, and thus subsequent artifact reduction performance.

4.5.2 P and T Event Baselines

AASM rules for detecting P and T events require constructing an appropriate baseline with respect to each event type. In this section we provide specifics around the process that was used to construct the baselines used in subsequent experiments. While there are many differences between baselines used to detect P versus T events by AASM rules, one differentiating characteristic is that baselines used for detecting P events are based on REM sleep segments, while T event baselines use NREM sleep.

AASM guidelines were used to calculate appropriate baselines. To calculate a baseline for T events, the tenth percentile of the chin EMG signal from up to 120 seconds of NREM sleep



(a) Original ECG signal

(b) Original Chin EMG signal

(c) Estimated artifact signal

(d) Processed Chin EMG signal

Figure 4.2: Artifact reduction example.

immediately preceding a REM region is used. If no NREM sleep immediately precedes a REM region, then the baseline from a previous NREM region is used. In contrast, P event baselines are initially the initial 120 seconds from the first REM region. After 120 seconds of REM sleep is been analyzed, the P baseline is taken as the preceding 120 seconds of REM sleep, which may include REM sleep from one or more previous REM regions. The REM baseline for each channel is determined separately as the q^{th} percentile of the EMG signal, where q is a configurable parameter. We interpreted the AASM rules to imply a q value of 50 for the median. Events identified as P were excluded from the REM baseline.

4.6 RSWA Event Detection Methods

In this section we will use the term "RSWA event" or "signal-level RSWA event" interchangeably with a single P or T event. Recall from Section 4.2.2 that a diagnosis of RBD is largely based on the presence of RSWA events in a PSG. It follow that the accurate detection of RSWA events can be directly used for downstream RDB diagnosis. The goal of this section is to present DL based approaches to the task of RSWA event detection by leveraging several DNN architectures capable of handling sequential inputs, which in our case are processed signals from PSGs.

In order to evaluate the effectiveness of automatic, signal-level RSWA event detection, we investigated several methods. We first established a simple baseline for the task of RSWA event detection by implementing a rule-based system designed to mimic the AASM standards. Next, we constructed a fairly straight forward FFNN to gauge the effectiveness of a basic DNN architecture on the task of detecting RSWA events. Finally, we implemented and trained several DNN variants, leveraging more advanced architectures that we speculated would be better suited to the non-trivial task of RSWA event detection. In all cases, the DNNs we leveraged were capable of learning effective feature representations, hence no hand-engineered features were required.

4.6.1 Rule-Based

To establish a performance baseline, we first implemented a rule-based algorithm to detect RSWA signal-level events using the rules outlined in the AASM manual [13]. The algorithm used a sliding window approach to compare the amplitude of the rectified EMG signal to baselines calculated separately for T and P events. Section 4.5.2 provided details as to how baselines with respect to P and T events were calculated.

Using the before mentioned baselines, chin signals were considered to contain a T event if d_T consecutive seconds of the signal exceed the product of a threshold parameter θ_T and the NREM baseline. AASM rules specify d_T and θ_T to be 1 second and 1.0 respectively, although they can be considered configurable parameters. Signals from the chin or legs were considered to be signal-level P events if d_P consecutive seconds of signal were greater than the product of a threshold parameter θ_P and the REM baseline. AASM rules specify a value of 0.1 second for d_P and 4.0 for θ_P. Detected P and T event labels were combined into a sequence of predicted events according to the following rules: (1) A P event is recorded if a P event is detected in the Chin, L leg, or R leg channel and no overlapping T event occurs. (2) A T event is recorded if a T event is detected in the Chin channel, regardless of whether a P event was detected in any other channel. P and T events were ignored if they occurred within 15 seconds of an annotated apnea or hypopnea event.

Optimal values for q, θ_T, d_T, d_P, and θ_P were selected via grid search over the parameter space using balanced accuracy score, which was calculated over the combined training and validation set. The optimal parameters found were $q = 0.9$, $\theta_T = 4.0$, $d_T = 25$, $d_P = 10$, and $\theta_P = 6$. Optimized there parameters effectively increased evaluation on the combined training and validation from 0.42 to 0.49.

4.6.2 Deep Neural Networks

We evaluated several DNN architectures on the task of predicting signal-level RSWA events from a common set of input channels C. Each architecture used a consistent data representation $(\mathbf{X}, \mathbf{y})$, where $\mathbf{X} \in \mathbb{R}^{C \times 10n}$ was a n-second window for each channel down-samples to 10hz, and $\mathbf{y} \in \mathbb{R}^{1 \times 3}$ represented the three target class probabilities (no event, P, and T). Each input channel was processed as described in Section 4.5 with the exception of the baseline calculation discussed in Section 4.5.2, which was only used in the rule-based methods. With three input channels down-sampled to 10hz, and three output classes, we have $\mathbf{X} \in \mathbb{R}$ The job of each DNN architecture described in the following sections was to take in a $n-$second context window of processed signal from each channel, and correctly classify the event at the center of the window. Window size was treated as a hyperparameter, with a final window size of 20 seconds used in all reported experiments. Samples were taken via a sliding window approach with overlap. We compared several DNN architectures in the following experiments, each of which used the same train, validation and test data. The following is a brief discussion of our approaches:

Feed Forward Neural Network (DNN Baseline)

For a baseline DL approach we employed a traditional FFNN. The FFNN first encodes the three input channels, each consisting of $10n$ values, separately using different sets of weights. Each encoder branch consisted of two dense blocks, where a dense block was constructed from a dense layer with 128 neurons, ReLU activation and dropout. The output from each encoder branch was concatenated, and the resulting intermediate representation was then passed through two additional dense blocks with 128 and 64 neurons respectively, followed by an output dense layer with softmax activation to normalize the output values.

Recurrent Neural Networks

RNNs, which were introduced in Section 2.4.1, have long been used with great success for tasks involving sequential input data such as language modeling [147] and speech recognition [73]. We chose to leverage the LSTM flavor of RNN and constructed a DNN composed of two LSTM layers, each with 128 units, followed by two dense blocks with 128 and 64 neurons respectively and an output layer with softmax activation. While we chose the dense layer sizes to be consistent with the baseline FFNN. However, unlike the baseline, which encoded each input separately and then combined the outputs, all channels were input to the LSTM simultaneously.

Convolutional Neural Networks

CNNs, which were introduced in Section 2.4.2, were first popularized for computer vision applications [124], and are now commonplace in modern technology from consumer smart phone cameras, to industrial video monitoring systems. In addition to the 2D CNNs which have dominated the field of CV for many years, 1D CNNs have also seen success in sequence modeling tasks such as time series modeling [115, 175]. We experimented with three 1D CNN variants in this section:

1D CNN (multi-channel): We constructed a DNN composed of 1D Conv blocks, each of which comprised a 1D convolutional layer, followed by a batch normalization layer, ReLU activation, and a pooling layer. This variant is referred to as "mutli-channel" as processed signals from all channels were simultaneously input to the network, and passed through a single encoder branch. The encoder branch was composed of a 1D Conv input block with d_{in} convolutional filters of size $h \times w \times 3$ (for the Chin, L Leg and R Leg channels), followed by an additional 1D Conv blocks consisting of 64 $5 \times 5 \times d_{in}$ filters. The output activation was then propagated forward through two additional 1D Conv blocks with 128 $3 \times 3 \times 64$ and 128 $3 \times 3 \times 128$ filters respectively. This was followed by a max-pooling layer, and then two final 1D Conv blocks with 256 $3 \times 3 \times 128$ and 256 $3 \times 3 \times 256$ filters with subsequent pooling. The output from the final 1D Conv block was then flattened, and passed through several dense blocks, the last of which output a 1×3 vector of normalized scores for each of the 3 independent event classes. The basic architecture of the multi-channel 1D CNN is shown in Figure 4.3.

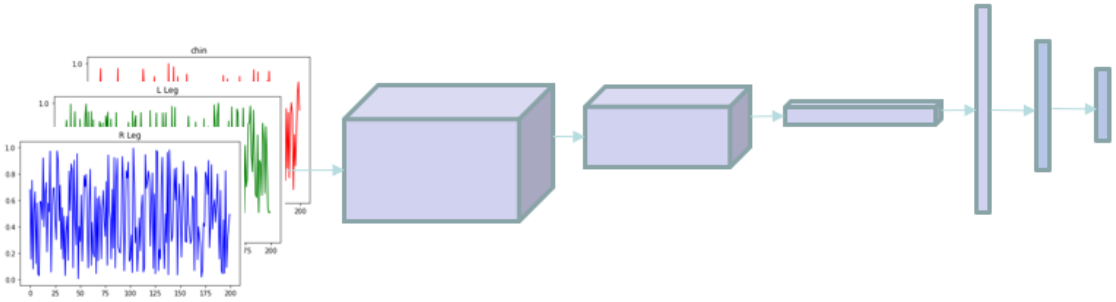


Figure 4.3: In the mutli-channel 1D CNN variant all channels are input to a single encoder branch. Each filter in the first Conv layer of the joint encoder is $h \times w \times 3$.

1D CNN (multi-branch): For this next 1D CNN variant, we kept the encoder architecture consistent with that of the 1D CNN multi-channel variant, but encoded each input channel separately using a different set of weights, in the same fashion as the baseline FFNN [231].

That is, instead of 1 encoder for N signals, we employed N separate encoders, each of which was tasked with encoding 1 signal. The outputs from the encoder branches were then individually flattened, concatenated, and passed through the same sequence of dense blocks as the multi-channel network. The architectural choices for this variant were made in order to allow for fair comparison with the multi-channel variant. A multi-branch 1D CNN is shown in Figure 4.4.

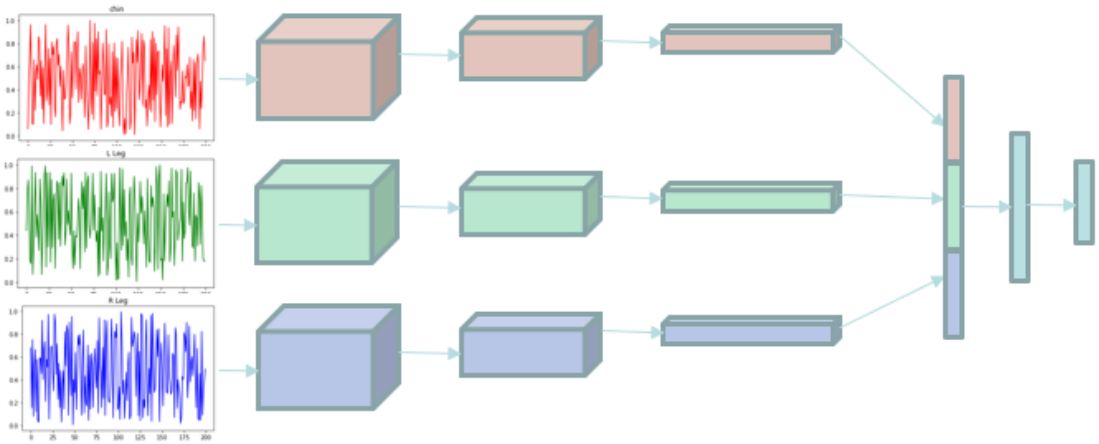


Figure 4.4: In the mutli-branch 1D CNN variant each channel is encoded by a different encoder branch. Each filter in the first Conv layer of each encoder is $h \times w \times 1$.

1D CNN (multi-channel with residual connections): As discussed in Section 2.4.3, the residual, or "skip" connection, allows for information from a previous layer to be combined with the output of a subsequent layer, which in-turn allows for a more resilient network that is robust to increased depth. This approach utilized three times as many convolutional blocks as the 1D CNN multi-channel approach, but with residual connections added. While residual connections are an effective way of learning much deeper networks, which in many cases outperform their shallower counterparts, they are not necessarily appropriate for relatively small networks, hence the increased depth of this variant.

Each network described above was trained with the Adam optimizer [116] for up to 200 epochs, using a mini-batch size of 128, and an initial learning rate of $1e^{-3}$. Early stopping was employed with cut-off of 20 epochs, as well as learning rate decay with a decay factor of 0.1. A test set comprising ≈ 600 REM regions from ≈ 60 patients was used for evaluation, with each REM region representing $\approx 4,000$ samples. The test set therefore consisted of $\approx 240k$ examples in total.

4.7 Experiments

4.7.1 Metrics

Two metrics were used to quantify the performances of the proposed event detection models, using signal-level, human labels as ground truth:

1. **Balanced Accuracy**: Balanced Accuracy (BAC) was selected as a primary evaluation metric due to the highly skewed nature of the data. That is, the likelihood of observing "no event" is far higher than that of a P or T event. BAC avoids inflated performance estimates on imbalanced datasets by taking into account the frequency of occurrence with respect to each class. In the case of multiclass classification, BAC equates to the average of recall obtained on each class, which is essentially accuracy if the dataset is balanced, but provides a much more fair assessment of performance when a dataset is skewed. BAC in our case is defined as follows:
$$BAC = \frac{(recall_{none} + recall_P + recall_T)}{3}$$
where
$$recall = \frac{TP}{TP + FN}$$

2. **Inter-Labeler Agreement**: For Inter-Labeler Agreement we used an approach similar to the human inter-labeler metric used to certify new technicians for scoring apneas, and assessed agreement between each model and the human labeler using Cohen's Kappa coefficient [100], which is defined as follows:
$$\mathbb{K} = \frac{p_o - p_e}{1 - p_e} = 1 - \frac{1 - p_o}{1 - p_e}$$
In the above equation p_o is the relative observed agreement among raters, and p_e is the hypothetical probability of chance agreement, using the observed data to calculate the probabilities of each observer randomly seeing each category. Epochs are classified as T, P, or None, according to the following rules: (a) T epoch: At least one T event in the epoch, and the total duration of the T events is longer than the total duration of the P events in the epoch. (b) P epoch: At least one P event in the epoch, and the total duration of the P events is longer than or equal to the total duration of the T events in the epoch. (c) None: No T or P events in the epoch.

4.7.2 Results

Results of each metric are shown in Table 4.1. The deep learning approaches achieved better BAC scores than the rule-based approach. Regardless of variant, the 1D CNNs were better able

to classify events at the sample level than were the LTSM and FFNN variants. Despite similar BAC scores, the 1D CNN variants were differentiated by inter-labeler agreement, where the best performance was observed for the multi-channel 1D CNN with residual connections. Although the Cohen's Kappa values reported here are modest, this measure can yield small values (≈ 0.6) even when agreement is excellent ($\approx 90\%$) [152]. To our knowledge, inter-labeler agreements has not been quantitatively measured for human labelers on the task of RSWA event detection, but the agreement would likely be low [15].

Method / Metric	Signal-level BAC	Inter-Labeler Agreement
Rule-based	0.73 (0.27)	0.52 (0.47)
FFNN	0.85 (0.19)	0.60 (0.40)
LSTM	0.89 (0.16)	0.52 (0.41)
1D CNN (MC)	**0.91 (0.14)**	0.47 (0.40)
1D CNN (MB)	**0.91 (0.15)**	0.61 (0.40)
1D CNN (MC + res)	**0.91 (0.15)**	**0.68 (0.39)**

Table 4.1: Performance comparison of RSWA event detection methods. Methods are evaluated by mean balanced accuracy (BAC) and inter-labeler agreement (Cohen's Kappa) on the test set (standard deviation in parentheses). MC, MB, and res stand for multi-branch, multi-channel, and residual connections, respectively.

4.7.3 Discussion of Results

The rule-based system performed relatively poorly in signal-level event prediction, similar to previously reported results, suggesting that rule-based systems may be too simplistic, and lack the flexibility to achieve human-level performance in detecting RSWA events. Human labelers may use other implicit criteria or contextual cues, in addition to clinical rules, to detect RSWA events. For instance, PSG recordings are often contaminated by cardiac noise. While human labelers may perform robustly when assessing noisy PSG recordings by mentally filtering out the noise, the rule-based systems may fail unless it is explicitly programmed to do so. DNNs on the other hand, and CNNs in particular are capable of sophisticated feature learning, and may be able to effectively learn the relevant features and contextual cues used by human labelers in distinguishing between true RSWA events, and contaminating noise.

We speculate that the similarity in performance of the multi-channel and multi-branch models without residual connections was primarily the result of their similar encoder architectures. We note that there is little difference between the encoding achieved by passing all channels through a layer consisting of N $M \times 1 \times C$ sized filters simultaneously, and that achieved by passing each of the C channels though a separate layer comprising N $M \times 1 \times 1$ filters individually. After the first

layer, the comparison between subsequent encoding layers diverged between the two approaches, which may have helped the multi-branch network achieve a better inter-labeler agreement than its multi-channel counterpart. Increasing the network depth and adding residual connections resulted in significantly better inter-labeler agreement, but comparable signal-level accuracy. This may be in part because the signal-level accuracy of the networks without residual connections was already quite high, so the depth of these networks may have already been adequate to optimize signal-level balanced accuracy for the detection problem with the given inputs. Although we did not attempt to classify patients based on signal-level events, we speculate that the relatively high inter-labeler agreement of the residual network, in conjunction with the strong signal-level classification performance, may enable more accurate diagnosis.

4.8 Summary

Existing automated methods are largely focused on RDB diagnosis as a binary classification problem. Instead of patient-level binary diagnosis, the methods proposed in this chapter are capable of creating a much finer-grained analysis by detecting the location and class of all RSWA related events in a sleep study. In addition, the more granulated detection allows for the quantification of event duration, and frequency, which in-turn can help quantify disease load and severity, and can provide characterizing statistics for tasks such as population studies. Signal-level event detection also allows clinicians to compare their own scoring of events with the events identified by the model. It follows that utilization of our method by clinicians may enable better explainability of RSWA diagnoses by combining the results of our model with patient-level, binary classification systems using hidden, or obtuse features.

In this chapter we demonstrated that modern DNNs can perform very well on the task of automatic RSWA event detection at the signal level (i.e., an event of type x at time t). In addition, or method takes sensor data as input and automatically learn feature representations. The end result is a capable system which is efficient and objective. While a skilled sleep lab technician can spend hours to manually annotate a single study, our DL based approach, by comparison, is capable of doing the job, well, in a matter of seconds on a single commodity machine. By significantly reducing the overall cost of PSG annotation, both in human hours and in healthcare costs, we allow for RSWA event detection, and by extension RBD diagnosis, to be applied to a much broader patient population. By empowering clinicians with a tool for fast, accurate RSWA event detection we aim to lower the bar for early diagnosis, and better overall patient care.

Chapter 5

Parameter Efficient Fine-Tuning of Deep Neural Networks with Budget Allocation

5.1 Motivation

Since the resurgence and subsequent mainstream popularization of neural networks, DL has become the dominant paradigm in ML. DL has been extremely successful across the primary domains of ML including natural language processing (NLP), automatic speech recognition (ASR), computer vision (CV), reinforcement learning (RL) and more. While DNNs are becoming more powerful, robust, and generalizable, they are also getting significantly larger. In recent years we have seen DNNs with hundreds of millions of parameters, such as GPT-2 [179], evolving into networks with hundreds of billions of parameters, like GPT-3 which was proposed by Brown et al. in [20]. One of the most notable accomplishments of DNNs like GPT-3 is that they have been shown to perform reasonably well in few-shot settings where the pretrained "base" network is evaluated on a new task (not seen in training), with no additional task-specific adaptation, by providing only a few examples. One few-shot method proposed by Brown et al. is known as prompt engineering. Essentially, the goal of prompt engineering is to illicit a desired output by constructing a small set of "good" examples (i.e., prompts in the case of language modelling), consisting of formatted input/ output pairs concatenated together, which are then given to the model with the last output omitted for the model to generate. While the performance of GPT-3 in the few-shot setting is impressive, such methods are rarely on par with smaller, task-specific models, and show inconsistent performance from task to task.

On the other end of the spectrum we have full fine-tuning (FT), where all network parameters are updated to adapt the network to a new task. Full FT is capable of out-performing few-shot, and smaller task-specific models, but becomes a challenge in itself when the model being fine-tuned is huge. Models of large scale bring their own set of challenges through massive storage overhead,

and the shear amount of GPU memory required for training.

As mega-scale DNNs achieve increasingly impressive performance, researchers and product groups alike are posing the questions: (1) how can such models be adapted to new tasks, and (2) what would it take to deploy such a model a production system? Consider a DNN like GPT-3 Divinci, which is a 175 billion parameter transformer-based language model requiring around 350G to store a single copy, and close to 1TB of GPU memory to train. Working with models of this size requires access to huge amounts of resources, making them prohibitively expensive for many within the ML and AI community to leverage. In other words, such models may be out of reach for all but the biggest tech companies who have the resources to utilize them. A high bar for adoption of top-performing technologies equates to an uneven playing field, and the slowing of scientific progress. Moreover, in addition to the shear amount of resources required to leverage a modern, mega-scale DNN, there are other problems that can arise out of sequentially training a model on different tasks. Catastrophic forgetting [64], for example, is a problem which can arise where knowledge of a previously learned task, or tasks, is lost as information relevant to a new task is incorporated.

The work in this chapter was proposed in part as a solution to the problems discussed above, as well as a way to adapt pretrained DNNs to new tasks in a more efficient way, without loss of performance as compared to full FT[1]. We take the basic ideas and designs of Parameter Efficient Fine-Tuning (PEFT) of transformers, which has been successfully demonstrated in NLP, and apply them to the domain of CV. PEFT was outlined in Section 2.9, and transformers were discussed in Section 2.6. We then propose a novel approach to learning a parameter budget allocation, which is done as part of the overall network optimization, in order to make PEFT even more efficient and powerful. We show empirically that our method does not under-perform, and often outperforms base PEFT methods, as well as full fine-tuning. Moreover, we show that our method is particularly effective in closing the performance gap between full FT and PEFT in cases where the deficiency between the two is more pronounced.

The methods we propose in this chapter are relevant to the domain of healthcare AI for several reasons. While healthcare AI, much like AI in general, can benefit from efficient model adaptation, healthcare problems are much more commonly associated with other challenges that make them well suited to PEFT. First, healthcare problems often fall into the low data regime, as data can be expensive to collect, and relevant examples can be extremely rare. Second, task-specific data

[1]The work in this chapter was previously published in [214]: P. Wallis and X. Song Efficient Fine-Tuning of Deep Neural Networks with Effective Parameter Allocation in IEEE International Conference on Image Processing (ICIP) 2022

in healthcare applications can differ greatly from the general purpose data used in base model pretraining, making effective model adaptation critical.

5.2 Background and Related Work

To learn complex tasks well in an ML setting, it's common to employ highly complex models such as DNNs. While DNNs have been shown to be high performing, they are also known to be notoriously data hungry, and often require a huge number of training examples to converge. As discussed in Section 2.9, there are many problems that arise out of a large data requirement including availability of labeled data, and computational resources needed for training. In this section we highlight several key concepts and DNN architectures which are crucial to understanding the primary contribution of this chapter, which is the PEFT method we propose in Section 5.3.2. To that end, we will summarize some of the key concepts of transfer learning and PEFT, which were first introduced in Sections 2.7 and 2.9 respectively. We will also introduce the concept of "parameter budget allocation" with respect to PEFT, as well as Vision Transformers (ViT), the primary DNN architecture used in Section 5.4.

5.2.1 Transfer Learning

A key lesson from transfer learning is that knowledge gained while solving one problem can later be applied to a different, but potentially related problem. The concept of knowledge transfer in ML is not new. The basic ideas that are commonly associated with transfer learning have been around since the 1970s [16], popularized in the 1990s in part through the paradigm of multi-task learning [23], and are now ubiquitous within the ML and AI communities. In modern DL, fine-tuning a pretrained base model, trained on a general task with a large amount of diverse data, has become a highly successful, and widely used strategy.

Pretrained Vision Models

Transfer learning has been applied in CV with great success for many years, and has traditionally employed the CNN style architectures discussed in Section 2.4.2. Transformers, on the other hand, were initially proposed for seq2seq tasks such as machine translation (MT) or time series modeling, and were overlooked by CV researchers. More recently, ViTs, which will be discussed more in Section 5.2.6, have been shown to outperform CNNs across a range of common CV tasks [113]. Much like the various objectives used to pretrain transformer-based language models (Section 2.7.2), ViTs can be pretrained in a number of ways including: fully supervised (when labels are available),

such as in the widely used ImageNet dataset [45]; unsupervised, which has been shown to be highly effective through approaches such as Momentum Contrast [76] and Masked Autoencoders [75]; and even in multi-modal setting as demonstrated in CLIP [178], where inputs are pairs of text and images. In short, modern transformer fueled applications have reinforced the principals of transfer learning. In particular, models trained on large amounts of diverse, general purpose data can be leveraged to learn new tasks more effectively, and efficiently.

5.2.2 Parameter Efficient Fine-Tuning

PEFT, which was discussed in Section 2.9, is a methodology where a pretrained base model is adapted to a new task by updating a small number of parameters as compared to the full model size. PEFT can dramatically reduce storage and memory requirements, significantly lower training time, and has been shown, in many cases, to result in a better performing model when compared with full fine-tuning [89, 136, 93]. Hu et al. [93] showed that the reduction in storage required for top-performing PEFT models can be upwards of $10,000x$ (for GPT-3 Divinci) versus traditional fine-tuning, with an accompanying drop in GPU memory of $\approx \frac{1}{3}$. The large drop in storage associated with PEFT is attributed to the fact that a PEFT checkpoint, or single model adaptation, requires a very small number of parameters to be stored compared to the base model size. The drop in GPU memory, on the other hand, is the result of the optimizer not needing to store gradients with respect to the base model's parameters, as they are frozen, or "preserved" during training and are therefore not updated. As a result, PEFT serves to significantly lower the bar for adoption of massive, highly performant DNNs, thereby providing a more inclusive research landscape, and allowing for such models to be practically deployed in production systems. Furthermore, since PEFT preserves a base model's weights, we speculate that models adapted via PEFT should be less prone to catastrophic forgetting compared to their fully fine-tuning counterparts.

While the benefits of low GPU memory overhead and reduced training time are obvious, one could argue that storage savings are of less importance, as the costs associated with storage are very low compared to the cost of memory. For example, the cost of 1G of storage from a modern cloud provider can be as little as $0.02 per month at the time of this writing, while the cost of a single compute node outfitted with 4 to 8 NVidia A100 GPUs can cost upwards of $40 per hour. While this argument is certainly valid, consider the case where a single pretrained DNN will be used as a base model with respect to a large number of task-specific applications. One such example is the case of personalization in digital assistants. In an extreme example such as personalization, a collection of user level, or user-segment level models may be required, resulting in a huge number of model variants. In such a case, a traditional fine-tuning approach would require a full sized copy of the

model for each adaptation. Therefore, with n model variants (i.e., customizations) and a base model size of S_{model}, the application would require $S_{model}+n\times S_{model} = (n+1)\times S_{model}$ storage. The same application would only need $S_{model} + n \times S_{peft}$ where $S_{peft} << S_{model}$. That is, $m = \frac{S_{model}}{Speft}$, where $m >> 1$ (e.g., 1K-10K). This represents a significant savings as m becomes large. For example, with $1K$ tasks (e.g., users in the previous example), and a base model on the scale of GPT-3 Divinci (i.e., 350G), one would need $350G + 1,000 \times 350G \approx 350T$ of persistent storage. With PEFT, which has been shown to reduce the checkpoint size of a GPT-3 Divinci adaptation by $10,000x$ [93], the same model requirement would carry a storage overhead of $350G + 1,000 \times \frac{350G}{10,000} = 350G + 35G \approx 385G$. Hence, if we assume $\frac{1TB}{Month} \approx \$5$, our example application would cost roughly \$21K/year for storage of the models alone, whereas with PEFT the annual model storage costs would be < 100. Moreover, a large model checkpoint can be very slow to load, and expensive to keep in memory, especially when switching between many tasks since each task requires a different full sized model. In contrast, switching between tasks via merging and un-merging a base model's weights in memory with a small supplemental set of task-specific parameters, is significantly more efficient.

Parameter Budget Allocation

While prior PEFT methods have provided a solid foundation, they leave room for improvement in a few key areas. One such area is the distribution of trainable parameters over a network, which we will refer to as "parameter budget allocation." There are many practical reasons, such as storage availability (e.g., on-device) or latency (e.g., near real-time applications) to impose a hard constraint on the parameter budget. Popular PEFT approaches such as Adapters [89] and Low-Rank Adaptation (LoRA) [93] allocate parameters based on the size of the layer being adapted and the rank of the adaptation layer, which is fixed regardless of the type of layer being adapted, or its depth in the network.

One of the early lessons from transfer learning in CV is that we can often freeze entire layers during fine-tuning, updating only a select subset. For example, in a pretrained CNN with $n + m$ layers, we may be able to freeze the first n layers, responsible for general sub-tasks such as edge detection, and tune only the last m. By extension, it seems reasonable to assume that not all layers in a complex DNN will require the same degree of adaptation with respect to a given task. Moreover, it's possible that different tasks may require different parameter budget allocations, even when leveraging the same pretrained base model.

5.2.3 Adapters

One of the most popular PEFT methods is known as Adapters tuning, or simply Adapters [89]. In this approach, a pretrained base model is equipped with "Adapter" modules which are inserted between consecutive layers at strategic points in a network. In the original work, Adapter modules were inserted after each MLP layer, within each transformer block as shown in Figure 5.1 (repeated from Section 2.9). Each Adapter module is composed of a bottleneck projection (i.e., one down projection and one up), a non-linearity, and a residual connection. The original work also employed layer normalization at the output. In an Adapter module, the bottleneck dimension is treated as a hyperparameter which can be tuned for performance, and / or chosen to regulate the number of trainable parameters being allotted for model adaptation. Adapters are a simple and highly effective PEFT method, capable of achieving results on par with, or even exceeding full fine-tuning. However, the sequential application of supplemental Adapter layers effectively adds depth to the network, which can result in increased latency.

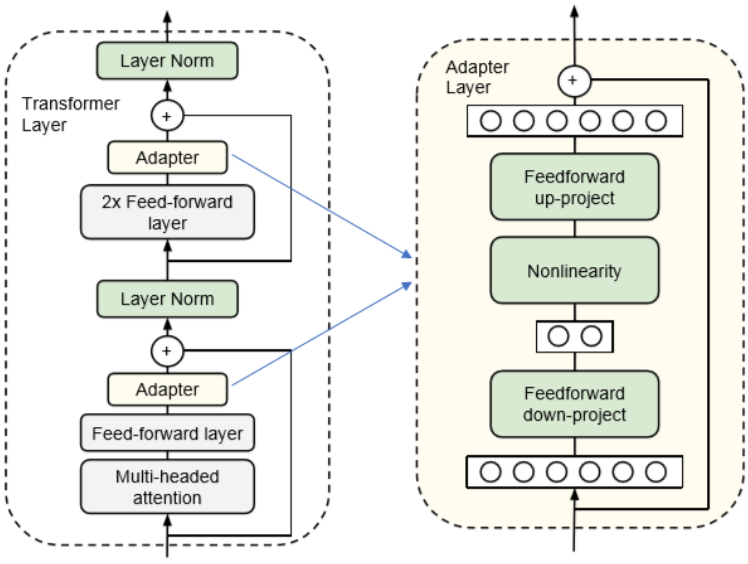


Figure 5.1: Design of the Adapter module (right) and placement within a transformer block (left). Each Adapter is inserted between two subsequent layers in the base model. Figure taken from [89].

5.2.4 Low-Rank Adaptations

Low-Rank Adaptations (LoRA) [93] is a PEFT method introduced by Hu et al. in 2021 [2]. LoRA was inspired by Aghajanyan et al [2], who showed that pre-trained language models have a low "intrinsic dimension" and can still learn efficiently despite a random projection to a smaller subspace. Hu et al. hypothesize that the weight updates for a pretrained weight matrix W_0 could have low "intrinsic rank" during adaptation. In LoRA, the weight updates ΔW with respect to W_0 are constrained by representing them as a low-rank decomposition $W_0 + \Delta W = W_0 + BA$, where $W_0 \in \mathbb{R}^{d \times k}$ (where d and k are the layer input and output sizes respectively), $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times k}$, and $r << min(d, k)$.

The structure of a LoRA layer is shown in Figure 5.2 (repeated from Section 2.9), which depicts an arbitrary base model layer with weights $W \in d \times k$. A LoRA layer has a high level structure similar to the Adapter layer discussed in Section 5.2.3, but the method itself has several notable differences. One such difference is that LoRA layers are applied in parallel, while Adapter layers are applied sequentially. It follows that each Adapter layer effectively adds to the overall depth of the resulting network, which can increase latency as well storage requirements. The parallel application of LoRA allows for such issues to be bypassed completely during inference. More specifically, for a base model layer W_0 and a corresponding LoRA layer $\Delta W = BA$, the forward pass becomes $h = W_0x + \Delta Wx = W_0x + BAx$. That is, during training, both W_0 and ΔW are fed the same input, and their outputs are summed component-wise. Each LoRA A matrix is initialized with a random Gaussian distribution, while B is initialized to 0 so that only the base model will be contributing to the loss at the start of training for stability.

Another key difference between LoRA and Adapters is that a LoRA layer can be viewed as a factorization of its base model counterpart layer, which is not the case with Adapters. It follows that LoRA weights can be easily merged (and un-merged) with those of its base model counterpart via simple linear operations (i.e., $W_0 \pm \Delta W$). Hence, during inference, the forward pass through a task-specific adaptation is computationally identical to a forward pass through the base model, and a base model with many LoRA variants (i.e., task specific adaptations) can be efficiently switched between tasks with low computational overhead. In addition, since a LoRA layer is essentially a factorization of its base model counterpart, the effect of full fine-tuning should be theoretically recoverable as r (i.e., LoRA rank) approaches D_{out}. That is, if LoRA was applied to each base model layer, and if the shape of each LoRA layer was equal to the shape of its base

[2]The author of this thesis was also a co-author of the original LoRA paper, but the work was done as part of the author's work at Microsoft, and was not done in association with OHSU as part of this thesis.

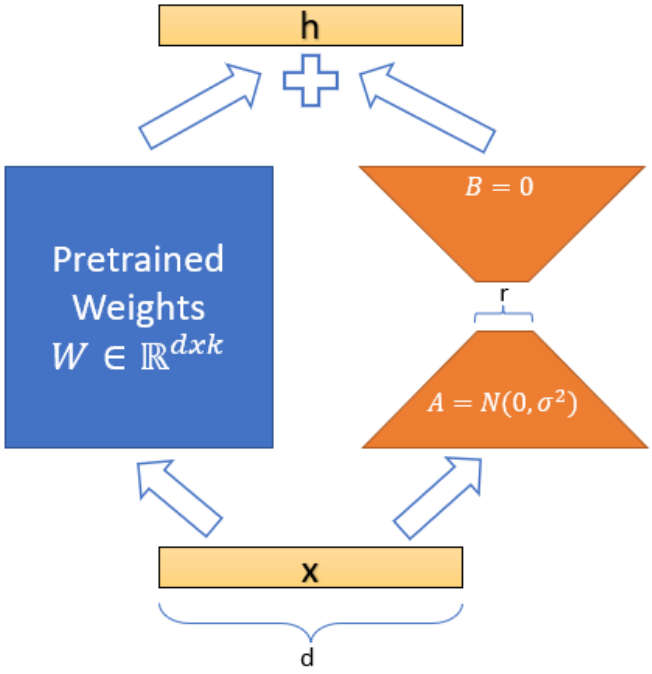


Figure 5.2: This figure depicts a single LoRA layer on the right, which is essentially a factorization of the corresponding single base model layer on the left. In a LoRA layer, the input x to layer l, which represents the output from layer $l-1$, is passed forward through the LoRA layer $\Delta W^{(l)} = BA$ and corresponding base model layer $W^{(l)}$ simultaneously. The output signals are then combined into a single output activation $h^{(l)}$, which is propagated forward. That is, $h^{(l)} = W^{(l)}x + \Delta W^{(l)}x = W^{(l)}x + B^{(l)}A^{(l)}x$.

model counterpart, the result of training should be equivalent to that of fully fine-tuning the base model. Adapter layers, on the other hand, come with no such assurances. While Hu et al. proposed a solution in LoRA which improved upon some of the shortcomings inherent in Adapters, one issue that remains in both is the use of a fixed rank across all adapted layers. That is, both LoRA and Adapter layers are instantiated with the same rank r across all layers being adapted regardless of the base model layer type, or depth in the network. It may be more efficient and effective to let the network decide the rank it needs at each layer, and therefore the amount of help a given layer requires to be adapted to a new task.

5.2.5 Sparsely Gated Mixture of Experts

Sparsely Gated Mixture of Experts (MoE) was introduced by Shazeer et al [202] in 2017, and was discussed in more detail in Section 2.8. The key idea behind Sparsely Gated MoE is to add one or more MoE layers to a network, where each MoE layer comprises many smaller "expert" networks

that learn to specialize in different data regimes in order to perform the overall network objective better. In each MoE layer, the output from the previous layer is fed into n "expert" layers, denoted as $E_i, ..., E_n$, simultaneously. The corresponding outputs $y_1, ..., y_n$ are then aggregated by taking a weighted combination. The weights used for aggregation are output from a gating network G, with learnable parameters W_g, whose job is to regulate the contribution of each "expert." That is, each MoE layer l outputs a $y^{(l)}$ where $y^{(l)} = \sum_{i=1}^{N} G(x)_i E_i(x)$. To achieve SOTA performance in practice, MoE networks often contain many individual MoE layers, each of which consists of thousands of individual "experts," therefore requiring a huge amount of computation to optimize. For example, a single DNN could utilize 5 MoE layers, where each MoE layer employs 1K individual expert networks. The solution to this computation bottleneck proposed by Shazeer et al, which also encourages specialization, is to replace the standard gating mechanism with a sparse gating mechanism, wherein only the top-k experts are used for each iteration, hence the computation of $n - k$ of the original experts is eliminated making the computational overhead much more manageable. A single sparsely gated MoE layer embedded within a recurrent language model is shown in Figure 5.3.

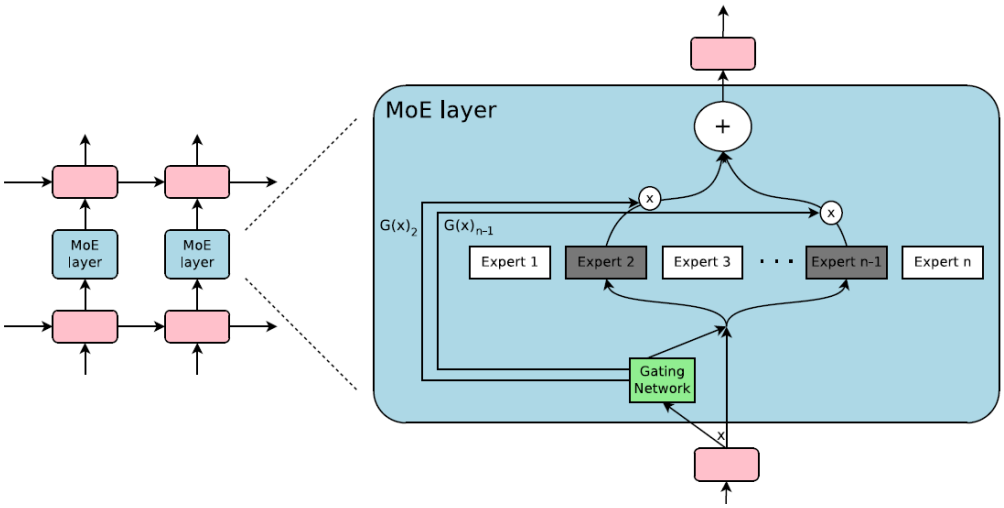


Figure 5.3: A Sparsely Gated MoE layer embedded within a recurrent language model.

Sparsely gated MoE was inspirational to our work in learning to allocate parameters in PEFT, which will be discussed in Section 5.3.2. At a high level, like sparsely gated MoE, we also employ gates to regulate the contribution of individual parameters, but the parameters in our case are individual LoRA vectors, not entire networks, and sparsity is obtained through very different means. Specifically, we use regularization to encourage gate sparsity, as opposed to sparsely gated MoE where an explicit gating function is employed whose job is to weigh the "relevance" of each expert

with respect to the current example, which is then used to select the top-k. Therefore the "gates" in sparsely gated MoE do not actually remove experts from the network as much as allow for candidate selection by assigning a weight to each expert. Moreover, in sparsely gated MoE the number of experts is a hyperparameter, which does not allow for the number of experts to vary by layer.

5.2.6 Vision Transformers

Much like the Transformer itself, PEFT has commonly been associated with NLP, in part due to the massive scale of modern language models. In this work we aim to extend the success of PEFT in NLP to the domain of CV. While PEFT is an architecture agnostic methodology, we will use Vision Transformers (ViTs) [51] as the base architecture to showcase the effectiveness of PEFT in CV for the remainder of this chapter. ViTs were chosen in part due to their recent success across many common CV problems, coupled with the fact that PEFT of Transformers has been successfully demonstrated within NLP.

Since the introduction of ViTs in 2020, we have seen variants of the Transformer architecture applied to a range of common CV tasks including image classification [51, 218], semantic segmentation [181, 205], and image retrieval [54]. A Transformer encoder is composed of one or more Transformer blocks, and utilizes the following key concepts and components of the Transformer architecture which were introduced in Sections 2.5.2 and 2.6: self-attention, multi-head attention, and positional encoding.

In a transformer-based language model, the initial inputs are sequences of tokens. ViTs on the other hand start with an input image, which is not naturally interpreted as a sequence. While various encoding methods have been proposed (e.g., [218]), the simplest mechanism for converting an image into a sequence in ViTs consists of the following steps: break the input image into an $n \times n$ grid of patches (where each patch is contiguous region of pixel intensities); convert each patch into an embedding via linear transformation; add a positional encoding to each patch representing the position of the patch in the original input image. This process produces an input which is analogous to the sequence of token embeddings which are fed into a transformer-based language model. One advantage of ViTs over CNNs is that the former uses a global representation of the input image throughout the network. That is, CNNs have a relatively small receptive field, proportional to kernel size, and a global representation of the input is only obtained late in the network. ViTs on the other hand use self-attention to represent each patch in the input image as a weighted combination of all input patches, hence the entire image is taken into account at each stage of the network. The basic ViT architecture is shown in Figure 5.4.

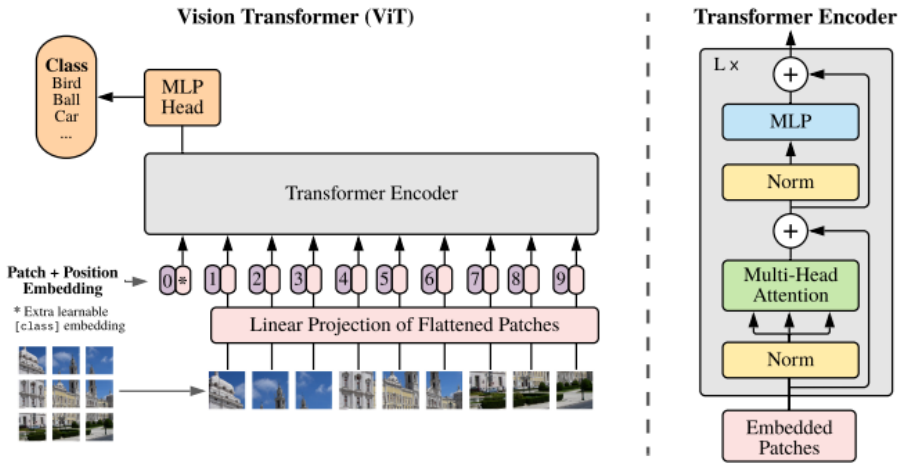


Figure 5.4: Basic flow of a Vision Transformer (ViT) for image classification. Left hand side shows the flow of information though the network: The input image is divided into an $n \times n$ grid of patches, which are then flattened out into patch embeddings via linear transformation (i.e., projection of flattened images), and combined with positional encodings. The resulting input sequence of patch + positional embeddings are then fed to a task-specific Transformer, which is a transformer encoder for image classification in this case. The output from the transformer encoder in this case is passed though a classification head (MLP with softmax), which outputs a distribution over the image classes. The right hand side shows the high level architecture of the transformer encoder, which takes embedded image patches as inputs and outputs embeddings.

5.3 Methods

In order to maximize performance with a fixed parameter budget, we propose a simple yet effective solution. Inspired by regularization based feature selection methods such as Lasso [208], and sparsely gated MoE [202] which was discussed in Section 5.2.5, we extend upon the matrix factorization design of LoRA [93] by inserting a gating mechanism between each bottleneck projection, and encouraging sparsity over the gates via ℓ_1 regularization. This differs significantly from sparsely gated MOE, which learns a gating function which is used to select a subset of "experts" via ranking and top-k selection. While our approach can be applied to Adapters and LoRA alike, we choose to leverage LoRA as the basic building block for this work as it has other desirable characteristics as noted in Section 5.2.4. The goal of the proposed method is to maximize performance while adhering to a fixed parameter budget by gating parameter vectors in a LoRA influenced layer. We therefore refer to this new method as Gated Low-Rank Economical Adaptations, or GLoREA.

5.3.1 Encouraging Sparsity through Regularization

ℓ_1 regularization, which was discussed in Section 2.3, is not only an effective approach for preventing overfitting, but can also be leveraged as a flavor of feature selection due to its sparsity encouraging behavior. Feature reduction in a model can be accomplished by adding a ℓ_1 penalty term to the model's overall objective, which effectively pushes less influential parameter weights to 0 during optimization. Given a loss function L, the ℓ_1 penalized objective J is defined as follows:

$$J = \frac{1}{N}\sum_{i=1}^{N} L_i + \lambda||w||_1 = \frac{1}{N}\sum_{i=1}^{N} L_i + \lambda \sum_j \sum_k |w_{jk}|$$

For the purpose of illustration, a 2D example of ℓ_1 regularization is shown in Figure 5.5, where the ellipses represent the model loss to be minimized, and the diamond centered at the origin is the ℓ_1 constraint. It follows that, in order to minimize the loss while respecting this constraint, the optimal solution must be at an intersection of the two. Since this intersection will often be at an axis, the corresponding parameter will be set to 0. When data is high dimensional, as is commonly the case in DNNs, many parameters can be set to 0 simultaneously.

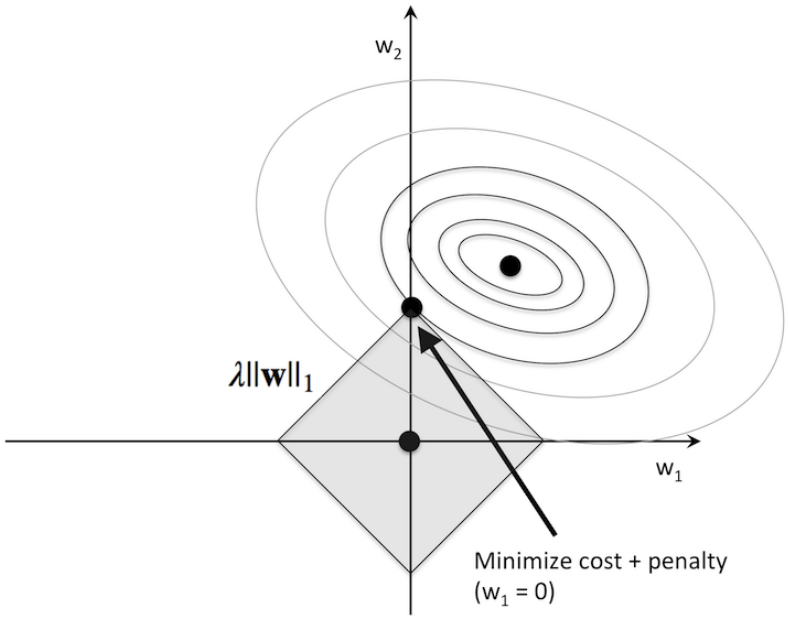


Figure 5.5: Overall model loss with L1 constraint

5.3.2 GLoREA

The goal of LoRA, as discussed in Section 5.2.4, is to efficiently adapt a pretrained base model to a new task by outfitting a subset of the base model's layers with low-rank decompositions. That is,

for each base model layer being adapted, a small set of additional parameters are allocated and updated during training, while base model weights are preserved. While LoRA has several key advantages over related methods such as Adapters and PrefixTuning, the base approach leaves room for improvement. Notably, each LoRA layer is parameterized by a rank r, which is directly proportional to the number of parameters the layer is allotted. Moreover, in LoRA this value is fixed for all layers being adapted. To the best of our knowledge, a fixed rank allows for a simple implementation in practice, but carries with it no concrete theoretical justification. It seems reasonable to assume, in the presence of a hard parameter budget constraint, that a variable r would be better equipped to maximize model performance versus a fixed r. That is, it may be more beneficial to allow the value of r to be different from layer to layer.

Recall, each LoRA layer has fixed rank r, corresponding to r independent weight vectors. We extended upon the idea of LoRA by equipping each low-rank adaptation with a vector of gates g, whose job is to regulate the contribution of each corresponding weight vector. By empowering a LoRA-style layer with the ability to regulate its component vectors via gating, the base LoRA forward pass $h = W_0x + BAx$ now becomes $h = W_0x + BGAx$, where $G = ReLU(gI)$ (i.e., the matrix G is diagonal with weights g). A ReLU non-linearity is applied to the gates to enforce a rule that a negative gate will effectively remove the contribution of its associated weight vector. In a GLoREA layer, each weight vector is meant to help the base model adapt to a new task. Intuitively, for each layer l being supplemented with a GLoREA module, each gate g_i^l can be thought of as a gauge of the relative importance of its associated weight vector. It follows that a weight vector corresponding to a higher gate value can be considered more influential to the network, and vectors corresponding to non-positive gate weights (i.e. where $g_i <= 0$) should be removable without hindering overall network performance. During training it is sufficient to simply zero out GLoREA vectors with non-positive gate values. Once training is complete, we can simply remove the set of weight vectors for each GLoREA layer corresponding to zero valued gates, thereby reducing the corresponding adaptation layer's rank, and by extension, the overall GLoREA model size in proportion to the number of "zero-gates". For example, consider a case where GLoREA is used to supplement all MLP layers in a ViT where all MLP layers have the same size, say N parameters. With an initial GLoREA rank r, if k gates are zeroed out on average from each adapted layer, the overall GLoREA model size is reduced by $\frac{k}{r}$. By allowing the network to decide where to allocate trainable parameters, and discouraging the retention of non-essential parameters, the network can learn more task-specialized features. In other words, in a GLoREA layer with an appropriate gate regularization coefficient, parameters that are deemed to be non-beneficial to the overall network objective are removed. We speculate that removing "non-essential" parameters should equate

to the remaining parameters being more specialized to the current objective, with little to no parameters being squandered. We note that only weight vectors in GLoREA layers are removed by the zero-gates. Each corresponding base model layer remains untouched. The Anatomy of a GLoREA layer is shown in Figure 5.6.

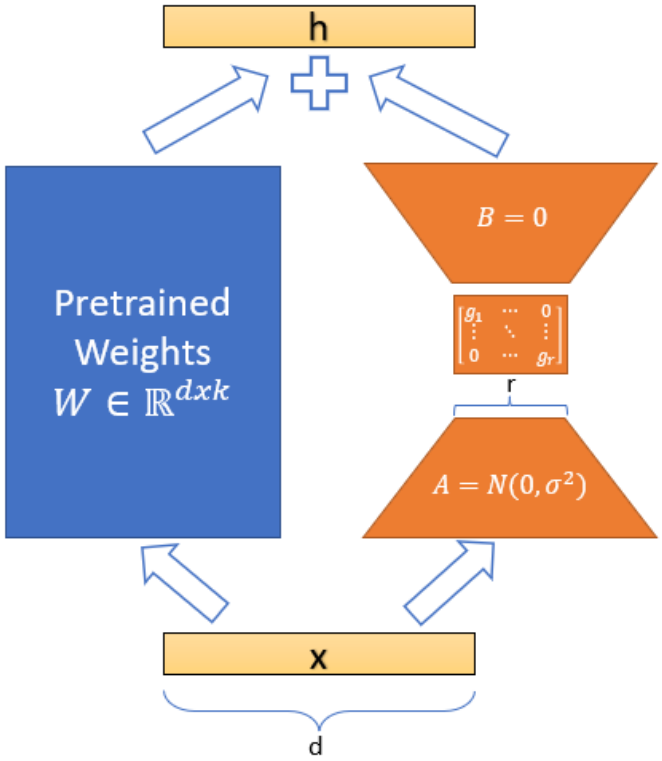


Figure 5.6: A GLoREA layer adds a set of r learnable gates between A and B to regulate the contribution of each individual weight vector. In GLoREA, A and B are encouraged to be orthogonal to promote diversity and reduce redundancy across dimensions.

Encouraging Orthogonality Through Regularization

Encouraging orthogonality in DNNs has been shown to encourage more effective feature learning, and boost overall network performance [9, 222]. We adapt this idea to PEFT by encouraging the vectors in each GLoREA projection matrix to be orthogonal. That is, we aim to enforce $A^\intercal A \approx I$ and $BB^\intercal \approx I$ by adding two additional penalty terms to the network objective function:

$$\lambda_{A_{ortho}} ||A^\intercal A - I||_F^2 \quad \text{and} \quad \lambda_{B_{ortho}} ||BB^\intercal - I||_F^2$$

$||\cdot||_F$ is the Frobenius norm, which is an extension of the Euclidean norm to $K^{n\times n}$. That is:

$$||A||_F = \sqrt{\sum_{i,j=1}^{n} |a_{i,j}|^2} = \sqrt{trace(A^*A)} = \sqrt{\sum_{i=1}^{min(m,n)} \sigma_i^2(A)}$$

where $\sigma_i(A)$ are the singular values of A. This is referred to as "soft orthogonality regularization" in Bansal et al. [9], which was inspired by Xie et al. [222], who proposed that the Gram matrix of a DNN weight matrix should be close to I. In the basic LoRA setup, allowing for some redundancy across weight vectors may be acceptable since budget allocation is not a factor. With GLoREA however, we want to remove unneeded parameters in order to stretch our budget. Hence, the orthogonality condition is imposed to empower the network to make more informed decisions around which dimensions to remove during optimization. In other words, encouraging GLoREA vectors to be independent serves to remove some of the ambiguity associated with rank reduction.

5.3.3 Training Procedure

Since GLoREA reduces the initial parameter budget by removing entire weight vectors during training, we can over-parameterize our network at first by initializing each GLoREA layer with a fixed r_{init}, so that $r_i = r_{init}\ \forall i$, where r_{init} is the initial bottleneck dimension. Once training begins, the optimizer is free to update individual gates as it deems beneficial to the overall objective, which in-turn can reduce individual r_i values. It follows that when training is done, $r_i <= r_{init}\ \forall i$, and $r_i = r_{init}$ only when no gates are set to 0. In addition to regularizing the weights and gates, we add additional regularization terms to encourage A and B to be orthogonal. The reason for encouraging orthogonality is to keep the vectors of A and B independent so that their respective contributions are more diverse, and less redundant. We speculate that the set of weight vectors that are retained in an orthogonal factorization are more meaningful, and specialized. We will show that this method can, in many cases, achieve better performance compared to baseline methods with the same number of parameters, and often times fewer.

The objective function for GLoREA training takes the following form:

$$J = \frac{1}{N}\sum_{i=1}^{N} L_i + \lambda_2||\Delta W||_2^2 + \lambda_1||g||_1 + \lambda_{orth}(||A^\intercal A - I||_F^2 + ||BB^\intercal - I||_F^2) \tag{5.1}$$

The first term in Equation 5.1 is the empirical loss, or mean loss over all input samples. The second term is a regularization penalty which acts only on the trainable parameters ΔW (which does not include gates). This penalty is imposed to keep the ΔW weights from getting too big, and to reduce the potential for overfitting, but not to remove weights completely. The third term represents the ℓ_1 penalty imposed on the gates g. The primary role of the gate penalty term is to

to drop GLoREA weight vectors which are determined to be non-essential for network adaptation with respect to the current task, as well as to regulate the contribution of the GLoREA vectors which are retained. The last term is meant to encourage orthogonality of GLoREA vectors, as discussed in Section 5.3.2. The hyperparameters λ_1, λ_2, and λ_{orth} serve to weigh the contributions of each penalty term on the overall objective.

5.4 Experiments

We evaluate our approach across a range of common benchmarks tasks in image classification and semantic segmentation. As a side note, image classification and semantic segmentation are two tasks which are highly relevant in medicine. For each benchmark, we select an architecture and train three variants using different adaptation methods: first, a fully fine-tuned (FT) model where all parameters are updated during training; second, a LoRA variant which has fixed, uniform parameter budget allocation; lastly, a GLoREA variant (our method) which allows for parameter budget allocation to be learned. We allow for an equal number of trainable parameters to be used for LoRA and GLoREA in each experiment in order to make a comparison which is not biased by parameter budget. GLoREA does not have a mechanism to enforce a pre-defined parameter budget. The gate regularization coefficient is tuned during training optimize performance, and to remove more or less parameters depending on the use case, but the network optimization determines the final budget. Since we cannot specify a specific the exact, final GLoREA parameter budget up-front, we optimize the hyperparameters in each GLoREA experiment to maximize performance, and then set the number of parameters in each corresponding LoRA experiment to approximately the final parameter budget retained by GLoREA. Adding a mechanism by which a ending parameter budget can be specified up-front will be left to future work.

5.4.1 Base Model

A ViT-small variant, pretrained on ImageNet 1K [46], was used as the base model architecture for each adaption method. ViT-small is a 12 layer ViT, with $\approx 23M$ parameters. ImageNet was first introduced in 2009 and is one of the most widely used datasets in CV. While there is more than one version of ImageNet (e.g., ImageNet 21K), ImageNet 1K is arguably the most popular. More specifically, ImageNet 1K is a high quality, supervised image dataset which spans 1K object classes and contains over 1M training examples, 50K validation images and 100K test images.

5.4.2 Image Classification

We first evaluate each of the adaptation methods on eight common benchmark tasks in image classification. The specific benchmark image classification tasks used for evaluation are as follows:

- **CIFAR10**: CIFAR10 is a subset of the "tiny images" dataset introduced in [123], and is one of the most commonly used benchmarks in image classification. The dataset consists of $60K$ 32×32 colour images in 10 general classes which include animals, vehicles, ships and aircraft. The dataset comprises $50K$ training images and $10K$ test images.

- **MIT Indoor**: The objective of the MIT Indoor task is scene recognition [176]. The dataset is made up of $\approx 16K$ example scenes in total across 67 indoor seen categories. Scene categories include office, bedroom and kitchen, and span 5 domains: store, home, public spaces, leisure and work.

- **MINC**: The Materials in Context Database (MINC) was introduced in [12] and is used to evaluate a models ability to recognize materials in real-world images. The dataset consists of $\approx 7K$ samples in total spanning 23 material classes.

- **Oxford Pets**: The pets dataset is another common image classification dataset comprising $\approx 7K$ samples across 37 categories of cats and dogs.

- **Inaturalist-2019-insects and plants (two separate tasks)**: Inaturalist is a large repository of natural images which span many high level classes as described in [210]. We focus on two species classification sub-tasks from the larger 2019 Inaturalist repository in this evaluation: insects and plants. Plants is one of the largest datasets in the Inaturalist repository with $\approx 160K$ training, and $\approx 40K$ validation images from around 2K categories. Insects, another of the larger Inaturalist datasets consists of $\approx 100K$ training and $\approx 20K$ validation images across 1K species.

- **Stanford Cars**: The Cars dataset is another widely used benchmark in image classification and was first introduced in [122]. There are $16K$ samples in this dataset which span 196 categories in total.

- **Colorectal Histology**: The task of classifying textures in RGB colorectal cancer histology images is discussed in detail in [107]. The dataset is small, comprising only $5K$ images in total, spanning 8 distinct classes. Furthermore, this dataset represents medical images, and is dissimilar to the data used in ViT pretraining, making it a good test for the robustness of the PEFT methods being evaluated.

Each of the experiments shown in Table 5.1 was ran using a gate regularization coefficient $\lambda_1 = 1e^{-3}$ (from Equation 5.1), which was shown empirically to reduce the overproduced budget by $\approx 50\%$ with respect to the tasks being evaluated. This will not always be the case, and should be evaluated by task, and by architecture. Imposing larger or smaller amounts of gate regularization could yield varying amounts of parameter reduction, as could a change in base model. The amount of weight regularization imposed varied by task, but was always relatively small with $\lambda_2 \in [1e^{-5}, 2e^{-4}]$. Each job was ran for upwards of 100 epochs, where the final model selected was based on best validation set metrics, as evaluated via standard cross-validation.

Dataset	Full FT	LoRA	GLoREA
trainable params	23M	0.2M	0.2M
cifar-10	98.4	98.5	**98.8**
mit-indoor	86.7	87.0	**87.5**
minc	85.5	**87.0**	**87.0**
pets	92.6	91.4	**93.5**
inaturalist-2019-insects	**60.5**	56.7	57.6
inaturalist-2019-plants	65.0	64.4	**65.5**
colorectal-histology	98.4	98.7	**99.2**
stanford-cars	**90.1**	87.9	89.2
mean	84.7	84.0	**84.8**

Table 5.1: Comparison of fine-tuning method on ViT-small by classification accuracy.

Analysis of Image Classification

There are several key takeaways from our experiments in comparing full FT, LoRA and GLoREA on the task of image classification. First, we see that PEFT is a viable option for model adaptation with respect to this high-level task. LoRA is on par with full FT in most cases, with an average classification accuracy delta of $< 1\%$ across all benchmark tasks. Moreover, LoRA wins over full FT on 4 out of 8 tasks including cifar-10 5.4.2, mit-indoor 5.4.2, minc 5.4.2 (by a fairly large margin) and colorecral-histology 5.4.2. Of the four remaining tasks where LoRA did not out-perform full FT, two showed a deficiency which was more pronounced than the rest. Specifically, inaturalist-insects and stanford-cars. We speculate that this deficiency could be due to a disconnect between the pretraining and FT tasks. In other words, a base model pretrained on a task which is not complementary to the FT task could result in a worse performing LoRA based adaptation. In general, PEFT is highly dependant on transfer learning. It follows that if new task cannot gain any benefit from the pretraining task, then PEFT will likely fall short of full FT. This seems intuitive

since full FT, despite its shortcomings (discussed in Section 2.9), can adjust all of the base model weights to adapt to a new task, and can therefore mimic training from scratch in the case where the base model is ill-suited to the FT task. LoRA, and other PEFT methods, can only update the small number of parameters allotted for adaptation, and will therefore not be as flexible as full FT in such a setting.

Next, we can see from Table 5.1 that GLoREA does not exclusively outperform all other adaptation approaches. GLoREA does however outperforms LoRA across all but one task (which was tied), and performs best of the three methods on average. GLoREA also stands out in tasks where base LoRA under performs full FT by a larger margin, and therefore serves to close the gap with respect to such tasks, while still respecting the parameter budget. For example, we see LoRA under-performing full FT on the pets dataset. GLoREA not only closes the gap between PEFT and full FT, but actually out performs full FT on this task by a respectable margin of around 1%. Another such example is stanford-cars. With a large number of classes (≈ 200) and a relatively small number of total examples ($\approx 16K$), the cars dataset can be challenging for PEFT, and may benefit from GLoREA's ability to create more focused, task-specialized features through learned parameter allocation.

Finally, the results indicate that GLoREA, which is capable of allocating resources to parts of the network that need the most help, is well suited to tasks which fall into the low-data regime. We speculate that this is due in part to GLoREAs ability to use trainable parameters sparingly, retaining only the most useful subset, which is particularly important when training examples are scarce. If a DNN has too many parameters it's hard for the model to optimize adequately with few training examples. Moreover, if parameters are allocated to the parts of the network that are most helpful for the task at hand, the network can optimize a much smaller number of "specialized" parameters instead of a larger number of general parameters. To emphasize this point further we look at the colorectal-histology task, which is a medical imaging task with only 5K examples. We see that while LoRA slightly outperforms full FT on this task, GLoREA shows a much more impressive performance margin with the same parameter budget. Again, this result is likely the result of the learned parameter allocation, which results in a more effective utilization of the same parameter budget as compared to LoRA. We noted a clear pattern with respect to the proportion of parameters which are retained, or allocated, to different layers in the network based on layer type, and layer depth in the network. First, we noted that MLP layers retained a much larger proportion of their initial parameter budget as compared to self-attention layers, which was consistent across transformer blocks at different network depths. Second, early layers in the network seemed to retain less parameters as compared to mid and final layers in the network, which makes intuitive sense as

early layers are typically responsible for much more general tasks, such as edge detection, and may therefore require less help in adapting to new network-level tasks efficiently.

5.4.3 Semantic Segmentation

The second set of experiments focused on dense prediction, specifically semantic segmentation, where the goal is to correctly predict the correct class label for each pixel in the input image. Semantic segmentation typically employs an encoder-decoder style architecture [139, 187, 35]. While it's common to leverage a pretrained backbone for encoding, the decoder is usually trained from scratch which we also do for this set of experiments. As a result, parameter reduction of an encoder-decoder segmentation network of this type is bounded below by the size of the decoder.

We evaluate each adaptation method on two commonly used datasets for semantic segmentation: Pascal Context [156], which consists of roughly 10K train and 10K test images with around 450 classes; and ade20k [232], comprising 20K train, 2K validation, and 3K test images with approximately 150 pixel classes. The architectures trained for this set of experiments are based on Segmenter [205], which is a Transformer-based encoder-decoder style architecture for dense prediction. Segmenter leverages a ViT pretrained on ImageNet [45] as the encoder backbone, and initializes a smaller decoder consisting of two Transformer blocks. The role of the decoder is to map patch-level class scores to patch-level encodings, which are in turn up-sampled via bi-linear interpolation to represent pixel-level scores. We chose to leverage the Segmenter architecture to compare approaches, and to evaluate the general effectiveness of PEFT in dense prediction, but note that several other Transformer-based architectures for dense prediction could have also been leveraged in a similar way. For example, Dense Prediction Transformers (DPT) [181], BERT Pretraining of Image Transformers (BEiT) [10] and Masked Auto Encoder pretraining (MAE) [75] with a segmentation head would all have been viable base architectures.

For this evaluation we used an ImageNet pretrained variant of ViT-Base as the encoder backbone with an image size of 384×384, and a patch size of 16×16. ViT-Base is a 12 layer, 12 head ViT with $\approx 87M$ parameters. The Segmenter architecture with ViT-Base comprises $103M$ parameters. The encoder, with $\approx 87M$ parameters, represents $\approx 84.5\%$ of the overall architecture, with an additional $\approx 15.5\%(16M)$ parameters in the decoder. We allow for a PEFT parameter budget of $500K$ and $300K$ for the encoder, resulting in two reported models for both LoRA and GLoREA in Table 5.2, and reducing the encoder's trainable parameters by 99.5% and 99.7% respectively. The decoder, however, is not pretrained, so must be trained from scratch which adds $\approx 16M$ parameters to the budget. The final parameter budget reduction was $\approx 84\%$ for each PEFT semantic segmentation experiment.

We use a GLoREA gate regularization coefficient of $\lambda_1 = 1e^{-3}$, resulting in a 20% reduction in trainable parameters across the encoder network, which is a smaller reduction compared to image classification. Performance on the task of semantic segmentation is compared via mean intersection over union (mIoU), which is reported in Table 5.2.

Method	Trainable	ade20k	PascalContext
Full FT	103M	**49.1**	53.9
LoRA	16.3M	46.6	53.3
LoRA	16.5M	46.7	53.2
GLoREA	16.3M	47.5	53.8
GLoREA	16.5M	48.0	**54.1**

Table 5.2: Comparison of fine-tuning method on ViT-base by mean Intersection over Union (mIoU).

Analysis of Semantic Segmentation

The results of our semantic segmentation experiments echo several of the key points noted in Section 5.4.2. First, we can see from Table 5.2 that LoRA achieves results which approach that of full FT on the pascal context dataset. This is an impressive validation of PEFT as semantic segmentation, where a label is output for each pixel, is typically a much more challenging task compared to image classification where the objective is to correctly predict a single class label. Second, when we look at the results from ade20k we see a much more pronounced deficiency between LoRA and full FT. As in the case of image classification, this could be the result of a pretrained base model which is ill-suited to the FT task. In such as a setting we would expect the base model to require fairly dramatic changes to adapt to the new task, and would therefore benefit less from PEFT where the new task is assumed to benefit from knowledge transferred from the pretraining task. It is worth noting that in such a case the base model would likely suffer from catastrophic forgetting, since the base model weights would be changed significantly during FT to accommodate the new task.

It is notable that GLoREA not only performs well on the pascal context dataset, but actually out-performs full FT. This is an impressive result on a task with hundreds of pixel level classes, and a relatively small number of training examples. As in the case of image classification, we speculate that GLoREA is able to achieve such an impressive result by effectively leveraging the base model, and allocating trainable parameters only to the parts of the network that need them the most with respect to the specific task adaptation. GLoREA thereby allows for the number of parameters to be kept as low as possible, which helps to encourage task-specialized parameters while simultaniously helping to prevent overfitting. Lastly, we see that GLoREA closes the gap

considerably on ade20k, although still falls short of full FT. As in the case of LoRA discussed in Section 5.4.2, it's possible that the base model being adapted is simply not well suited to the ade20k task, and therefore PEFT, even with budget allocation, is not flexible enough to effectively adapt this imagenet pretrained variant of ViT to the new task.

5.5 Conclusion

We have shown that a general pretrained ViT, equipped with a very small number of supplemental, trainable parameters, can perform on par with or even outperform its full FT counterpart. That is, the accomplishments of PEFT that have been proven to be highly successful in NLP can be applied to vision tasks with a similar degree of success. Moreover, we have introduced a simple yet powerful method, which we refer to as GLoREA, that learns an effective parameter budget allocation alongside weight updates during overall network optimization. The success of GLoREA over LoRA and full FT in our experiments shows the benefit of stretching a parameter budget by allowing the network to decide where it needs to the most help. This is in contrast to simply allocating parameters uniformly as in LoRA, or updating all network weights as in full FT. It is our opinion that using a smaller number of parameters, and learning how they should be distributed contributes to a more focused adaptation, which better preserves the most effective contributions of these very powerful base models. A lot of thought and computational resources went into training these widely used base models, and it may be the case that we not only don't need to change these well-trained DNNs much during adaptation, but we shouldn't. We further speculate that since preservation of the base model weights is more effective during GLoREA training, it should also reduced the likelihood of catastrophic forgetting.

Much like the work showcased in Chapter 3 and Chapter 4, PEFT, and GLoREA in particular, can be effectively leveraged towards advancing healthcare specific applications. Healthcare problems often fall into the low-data regime, and are commonly associated with data which can be significantly different from the general purpose data used in most pretraining tasks. It follows that healthcare applications can benefit greatly from leveraging modern, pretrained DNNs adapted via effective PEFT methods. We note that while GLoREA does not always out-perform base LoRA, it performs at least as well, and often times better with the same, or even smaller parameter budget. Since there does not seem to be a downside to GLoREA over base LoRA, we would suggest considering it as a drop-in replacement when choosing a PEFT method.

Chapter 6

Conclusions

6.1 Summary

This thesis has showcased original contributions with respect to three individual, but complimentary layers of applied DL: First, in Chapter 3, we detailed an effective approach for learning representations of clinical events which can be leveraged by a host of downstream tasks such as recommendation, personalization, and clinical predictive modeling within the domain of healthcare. Next, we showed in Chapter 4 how DNNs could be used to automatically, and accurately identify events related to RSWA from multi-channel sensor data (i.e., PSGs). By doing so we empower clinicians by providing an efficient, reliable, and high performing method for automatically annotating PSGs. The approach we developed was able to annotate a full overnight PSG in a matter of seconds, and performed on par with expert human sleep clinicians who typically take hours to perform a similar task. Our approach to RSWA event detection has the potential to save countless human hours, as well as healthcare costs, thereby significantly broadening accessibility, and availability of effective diagnosis of RBD. By extension, our method enables early detection and possibly treatment of neurodegenerative diseases such as Parkinson's and Alzheimer's. Lastly, in Chapter 5, we proposed and evaluated a novel PEFT method capable of learning how to allocate a fixed budget of trainable parameters, alongside network weights, to best serve the overall network objective. By applying our approach, a DNN composed of a large, pretrained base model and a small set of supplementary trainable parameters is able to stretch a fixed parameter budget, in a task driven way, in order to more effectively and efficiently adapt to new tasks. By introducing this approach we aim to lower the bar to adoption of top-performing modern DNNs, and aid in the development of SOTA systems which are less bound by the ever-increasing trends associated with data, storage, and distributed memory requirements. Through the methods and concepts proposed in this thesis we hope to inspire and empower researchers, and industry professionals alike to continue pushing applied DL and healthcare AI forward. The future of heathcare, and AI, can

benefit greatly from a continued partnership resulting in high quality, personalized care, and wide reaching clinical solutions. Advancing medicine and AI together has the potential to change lives, to save lives, and to revolutionize an industry that we all rely on.

6.2 Future Direction

This section in no way claims to provide an exhaustive list of extensions to the work proposed in this thesis, but simply aims to outline a few immediate directions which could be explored further with respect to each area of research.

6.2.1 Learning Semantic Relationships from Medical Codes

There are several logical updates, and extensions to the work presented in Chapter 3. One such update would be to replace the LSTM used in the autoregressive embedding approach with a Transformer based architecture. When this research was conducted, LSTMs were still the dominant architecture in NLP. Since then, Transformers have all but replaced RNNs in research and industry. It makes sense that Transformer based modeling approaches could yield better, more robust, and more generalizable representations. Since the time of the work outlined in Chapter 3 Transformers have been integrated into biomedical NLP. For example, in Med-BERT [183], the authors leverage a BERT-like Transformer to learn embeddings from EHRs, which are subsequently used for disease prediction.

Another natural extension to this chapter would be to create a combined representation by leveraging various patient level features, such as demographics and geographic data, in addition to the sequences of clinical events used in our work. Moreover, it seems reasonable to assume that there could be benefits to learning embeddings of such features by way of reconstruction style embedding methods like autoencoders. Such an approach could allow for encoding of richer relationships between patients, and allow for modeling of clinical scenarios through a combined embedded representation which reflects clinical events as well as non-clinical, patient specific features.

Lastly, a well known issue with representing medical history via sequences of clinical events is that the events are not evenly spaced. That is, unlike many common language applications where time between sequential elements (e.g., tokens) is not taken into consideration, time between successive clinical events can be highly informative, and should therefore be accounted for in clinical models. Factoring in the time between events would be a natural extension to the work presented in Chapter 3, and has the potential to greatly improve upon our methods.

6.2.2 Deep Learning Approaches to RSWA Event Detection

The AASM standards requires that an RSWA event cannot be classified outside of REM sleep, or in the presence of apnia / hypopnia events. In Chapter 4, we leverage expert clinical annotations to identify epochs which can, or cannot be used in model training as per the before mentioned criteria. One extension to this work would be to build on the event detection capability, which utilizes human in the loop components, towards an end-to-end system where no human is needed. It follow that an end-to-end system for RSWA event detection would need to be able to accurately determine sleep stage, and identify apnia / hypopnia events.

A simple and intuitive way to design an end-to-end system for RSWA event detection is by combining several component, each of which is responsible for an individual task, contributing to the primary goal. We will refer to this design as "train and chain." For example, one could train a stand alone model to perform sleep stage classification, another for apnia / hypopnia identification, another yet for RSWA event detection, and then chain the individual components together in an end-to-end fashion. Figure 6.1 depicts a pipeline where each example (i.e., preprocessed PSG) is routed first to a component tasked with sleep stage classification, then to a component for apnia / hypopnia identification, through an artifact reducer as described in Section 4.5, and finally into a RSWA event detection component.

A primary drawback to this "train and chain" style approach is that a chain only as strong as its weakest link. Since each component in a system such as this is trained with its own independent objective function, the final system can be very hard to debug and improve as the individual components cannot take into account the end-to-end system performance. Moreover, a system composed of several components trained in isolation, each with its own performance issues, will inevitably compound individual component level shortcomings into poor system level performance.

There are several ways to potentially improve upon the pipeline discussed above. First, one could take advantage of transfer learning 2.7 by adapting a pretrained backbone to each of the individual tasks. This could improve the performance of the individual component networks, and by extension, the end to end system. However, the previously discussed problems associated with training components in isolation will remain. Second, one could reformulate the pipeline shown in Figure 6.1, which consists of several, single task models, into a multi-task (MT) problem (discussed in Section 2.7.4) where the individual tasks of sleep stage classification, apnia / hypopnia identification, and RSWA event detection are learned simultaneously. Converting a sequence of single task DNNs into a single MT DNN would allow for information to be shared across tasks, and to more easily assess the individual contributions of each task through a shared objective.

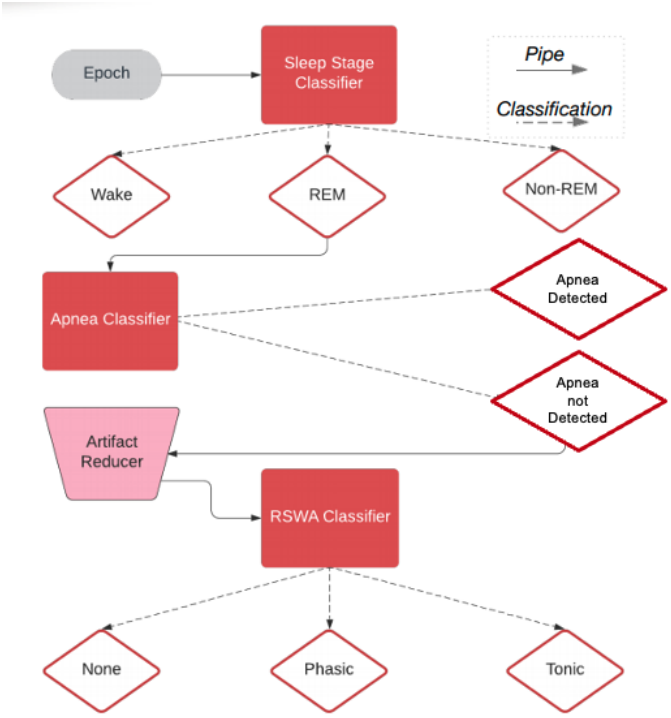


Figure 6.1: A example pipeline for end to end RSWA event detection. In this system, each component is trained in isolation to perform a specific task, and then chained together.

Moreover, in the MT setting one could affect the contributions of individual tasks on the overall model objective by assigning different weights to each. Weighing individual tasks in MTL can be done in many ways including manual tuning, or through task uncertainty [110].

6.2.3 Parameter Efficient Fine Tuning of Deep Neural Networks with Budget Allocation

A simple, but interesting extension of the methods proposed in Chapter 5 would be to remove the requirement of setting an initial rank r (i.e., how much to overparameterize), and simply initialize each GLoREA layer with the rank of its base model counterpart. That is, let $r^l = d_{out}$ for a base model layer $W^{(l)} \in \mathbf{R}^{d_{in} \times d_{out}}$. While this would require more memory to train at first, it could be implemented in a way that removes memory requirements along with feature dimensions, which could result in a better performing final (reduced) model by allowing more freedom for parameter budget allocation up front. For example, a task could benefit most if a small subset of pretrained layers are fully retrained, while all others are kept frozen. In such a case the model adapted with GLoREA could learn to retain the full rank for the before mentioned subset of layers, while dropping all GLoREA weight vectors in the other layers.

The methods proposed in Chapter 5 were tested on CV applications with well known benchmark datasets. An area that could be explored in order to "stress test" our approach is real world 3D medical imaging applications, such as multimodal brain tumor segmentation [**?**]. One of the key blockers of applying DNNs within domains such as medical imaging is the availability of data. We have emphasized throughout this thesis that DNNs are notoriously data hungry, and that high quality healthcare data, such as medical images, are expensive to collect and typically contains very few examples of important classes. We showed empirically in Section 5.4.2 that GLoREA is effective in adapting large DNNs to low-data tasks using fairly small, task specific datasets. However, we did not apply our method to dense prediction tasks involving medical images, nor tasks involving 3D images. We note that while future medical applications are used as examples in this section, GLoREA itself is a general purpose PEFT method which could be leveraged for large model adaptation across many diverse problems in AI.

Lastly, GLoREA is most certainly not the only way to distribute parameter budget while adapting a DNN in a parameter efficient way. For example, one could consider taking an approach more directly related to sparsely gated MoE, wherein an explicit, but very small gating network could assign weights to each low-rank vector for each layer being adapted. The output from such a gating network, along with a layer-specific threshold (possibly learned as part of the network), could be used to reduce an over-parameterized budget.

Furthermore, GLoREA does not include a way to determine the proportion by which the initial parameter budget will be reduced by over the course of the model training a priori. We would highly recommend integrating a mechanism by which the final parameter budget can be specified, such as an adaptable regularization coefficient.

Regardless of the method employed, parameter budget allocation is an effective way to get the most out of PEFT with a small budget constraint, which is an area worthy of further research. By allowing the distribution of parameters to be learned, we are making already powerful DNNs even more robust and adaptable. Coupling PEFT with parameter budget allocation allows DNNs to strengthen themselves on the fly, and to learn specialized features which are better equipped to help specific network layers towards the overall adaptation task.

Bibliography

[1] Y. S. Abu-Mostafa. Learning from hints in neural networks. *Journal of Complexity*, 6:192–198, 1989.

[2] A. Aghajanyan, S. Gupta, and L. Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7319–7328, 2021.

[3] F. Agostinelli, M. Hoffman, P. Sadowski, and P. Baldi. Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*, 2014.

[4] S. Albawi, T. A. Mohammed, and S. Al-Zawi. Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)*, pages 1–6. Ieee, 2017.

[5] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, M. Hasan, B. C. Van Essen, A. A. Awwal, and V. K. Asari. A state-of-the-art survey on deep learning theory and architectures. *Electronics*, 8(3):292, 2019.

[6] C. Alt, M. Hübner, and L. Hennig. Fine-tuning pre-trained transformer language models to distantly supervised relation extraction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1388–1398, 2019.

[7] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[8] H. Bahuleyan, L. Mou, O. Vechtomova, and P. Poupart. Variational attention for sequence-to-sequence models. *arXiv preprint arXiv:1712.08207*, 2017.

[9] N. Bansal, X. Chen, and Z. Wang. Can we gain more from orthogonality regularizations in training deep cnns? In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 4266–4276, 2018.

[10] H. Bao, L. Dong, and F. Wei. Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254*, 2021.

[11] A. L. Beam, B. Kompa, A. Schmaltz, I. Fried, G. Weber, N. Palmer, X. Shi, T. Cai, and I. S. Kohane. Clinical concept embeddings learned from massive sources of multimodal medical data. In *PACIFIC SYMPOSIUM ON BIOCOMPUTING 2020*, pages 295–306. World Scientific, 2019.

[12] S. Bell, P. Upchurch, N. Snavely, and K. Bala. Material recognition in the wild with the materials in context database. *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[13] R. B. Berry, R. Brooks, C. E. Gamaldo, S. M. Harding, C. Marcus, and B. V. Vaughn. *The AASM manual for the scoring of sleep and associated events*. American Academy of Sleep Medicine, Darien, Illinois, 2.0 edition.

[14] Y. Biberman. A context similarity measure. In *Proceedings of the Ninth European Conference on Machine Learning*, pages 49–63, Catania, Italy, 1994. Springer.

[15] D. Bliwise, J. Fairley, S. Hoff, R. Rosenberg, D. Rye, D. Schulman, and L. Trotti. Inter-rater agreement for visual discrimination of phasic and tonic electromyographic activity in sleep. *Sleep*, 41(7):1–6, 2018.

[16] S. Bozinovski and A. Fulgosi. The influence of pattern similarity and transfer learning upon training of a base perceptron b2. In *Proceedings of Symposium Informatica*, pages 3–121, 1976.

[17] L. Breiman. Out-of-bag estimation. Technical report, Statistics Department, University of California at Berkeley, Berkeley, CA, 1998.

[18] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, CA, 1984.

[19] P. F. Brown, S. A. Della Pietra, V. J. Della Pietra, and R. L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311, 1993.

[20] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[21] J. Burns, F. Consens, R. Little, K. Angell, S. Gilman, and R. Chervin. EMG variance during polysomnography as an assessment for REM sleep behavior disorder. *Sleep*, 30(12):1771–8, 2007.

[22] X. Cai, J. Gao, K. Y. Ngiam, B. C. Ooi, Y. Zhang, and X. Yuan. Medical concept embedding with time-aware attention. *arXiv preprint arXiv:1806.02873*, 2018.

[23] R. Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[24] M. Cesari, J. Christensen, L. Kempfner, A. Olesen, G. Mayer, K. Kesper, W. Oertel, F. Sixel-Döring, C. Trenkwalder, H. Sorensen, and P. Jennum. Comparison of computerized methods for rapid eye movement sleep without atonia detection. *Sleep*, 41(10):1 – 11, 2018.

[25] M. Cesari, J. Christensen, F. Sixel-Döring, C. Trenkwalder, G. Mayer, W. Oertel, P. Jennum, and H. Sorensen. Validation of a new data-driven automated algorithm for muscular activity detection in REM sleep behavior disorder. *J Neurosci Methods*, 312:53–64, 2019.

[26] W. Chan, N. Jaitly, Q. Le, and O. Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 4960–4964. IEEE, 2016.

[27] S. Chaudhari, V. Mithal, G. Polatkan, and R. Ramanath. An attentive survey of attention models. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 12(5):1–32, 2021.

[28] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[29] Z. Che, Y. Cheng, Z. Sun, and Y. Liu. Exploiting convolutional neural network for risk prediction with medical feature embedding. *arXiv preprint arXiv:1701.07474*, 2017.

[30] Z. Chi. Statistical properties of probabilistic context-free grammars. *Computational Linguistics*, 25:131–160, 1999.

[31] E. Choi, M. T. Bahadori, A. Schuetz, W. F. Stewart, and J. Sun. Doctor ai: Predicting clinical events via recurrent neural networks. In *Machine Learning for Healthcare Conference*, pages 301–318, 2016.

[32] Y. Choi, C. Y.-I. Chiu, and D. Sontag. Learning low-dimensional representations of medical concepts. *AMIA Summits on Translational Science Proceedings*, 2016:41, 2016.

[33] S. Chopra, M. Auli, and A. M. Rush. Abstractive sentence summarization with attentive recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 93–98, 2016.

[34] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[35] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger. 3d u-net: learning dense volumetric segmentation from sparse annotation. In *International conference on medical image computing and computer-assisted intervention*, pages 424–432. Springer, 2016.

[36] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[37] P. Colombo, E. Chapuis, M. Manica, E. Vignon, G. Varni, and C. Clavel. Guiding attention in sequence-to-sequence models for dialogue act prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7594–7601, 2020.

[38] N. Cooray, F. Andreotti, C. Lo, M. Symmonds, M. T. Hu, and M. De Vos. Detection of rem sleep behaviour disorder by automated polysomnography analysis. *Clinical Neurophysiology*, 130(4):505–514, 2019.

[39] C. Cortes, M. Mohri, and A. Rostamizadeh. L2 regularization for learning kernels. *UAI, 2009. Montreal, Canada*, 2009.

[40] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, New York, NY, 1991.

[41] M. W. Craven. *Extracting Comprehensible Models from Trained Neural Networks*. PhD thesis, Department of Computer Sciences, University of Wisconsin – Madison, Madison, WI, 1996.

[42] cs wiki. Max-pooling, 2018.

[43] L. Dehaspe. Maximum entropy modeling with clausal constraints. In *Proceedings of the Seventh International Workshop on Inductive Logic Programming*, pages 109–125, Prague, Czech Republic, 1997. Springer.

[44] G. DeMaagd and A. Philip. Parkinson's disease and its management: part 1: disease entity, risk factors, pathophysiology, clinical presentation, and diagnosis. *Pharmacy and therapeutics*, 40(8):504, 2015.

[45] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[46] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[47] P. Denis and J. Baldridge. Joint determination of anaphoricity and coreference resolution using integer programming. In *Proceedings of NAACL HLT 2007*, pages 236–243, Rochester, New York, 2007. ACL.

[48] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[49] X. Ding, Y. Zhang, T. Liu, and J. Duan. Deep learning for event-driven stock prediction. In *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[50] L. Dong, N. Yang, W. Wang, F. Wei, X. Liu, Y. Wang, J. Gao, M. Zhou, and H.-W. Hon. Unified language model pre-training for natural language understanding and generation. In *Advances in Neural Information Processing Systems*, pages 13063–13075, 2019.

[51] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[52] S. Dubois and N. Romano. Learning effective embeddings from medical notes. *arXiv preprint arXiv:1705.07025*, 2017.

[53] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[54] A. El-Nouby, N. Neverova, I. Laptev, and H. Jégou. Training vision transformers for image retrieval. *arXiv preprint arXiv:2102.05644*, 2021.

[55] J. Elliott, R. Opel, D. Pleshakov, A. Chau, T. Rachakonda, K. Weymann, and M. Lim. Post-traumatic stress disorder, with and without comorbid traumatic brain injury, increases the prevalence of rapid eye movement sleep behavior disorder in veterans. *Sleep*, October, 2019.

[56] W. Farhan, Z. Wang, Y. Huang, S. Wang, F. Wang, X. Jiang, et al. A predictive model for medical events based on contextual embedding of temporal sequences. *JMIR medical informatics*, 4(4):e5977, 2016.

[57] W. Fedus, B. Zoph, and N. Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

[58] R. Ferri, M. Manconi, G. Plazzi, O. Bruni, S. Vandi, P. Montagna, L. Ferini-Strambi, and M. Zucconi. A quantitative statistical analysis of the submentalis muscle EMG amplitude during sleep in normal controls and patients with rem sleep behavior disorder. *J Sleep Res*, 17(1):89–100, 2008.

[59] R. Ferri, F. Rundo, M. Manconi, G. Plazzi, O. Bruni, A. Oldani, L. Ferini-Strambi, and M. Zucconi. Improved computation of the atonia index in normal controls and patients with REM sleep behavior disorder. *Sleep Med*, 11(9):947–949, 2010.

[60] T. Fischer and C. Krauss. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2):654–669, 2018.

[61] R. Frandsen, M. Nikolic, M. Zoetmulder, L. Kempfner, and P. Jennum. Analysis of automated quantification of motor activity in REM sleep behaviour disorder. *J Sleep Res*, 24(5):583–590, 2015.

[62] B. Frauscher, D. Gabelia, M. Biermayr, A. Stefani, H. Hackner, T. Mitterling, W. Poewe, and B. Högl. Validation of an integrated software for the detection of rapid eye movement sleep behavior disorder. *Sleep*, 37(10):1663–71, 2014.

[63] B. Frauscher, A. Iranzo, C. Gaig, V. Gschliesser, M. Guaita, V. Raffelseder, L. Ehrmann, N. Sola, M. Salamero, E. Tolosa, W. Poewe, J. Santamaria, and B. Högl. Normative EMG values during REM sleep for the diagnosis of REM sleep behavior disorder. *Sleep*, 35(6):835–847, 2012.

[64] R. M. French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[65] J. H. Friedman. On bias, variance, 0/1 - loss, and the curse-of-dimensionality. Technical report, Department of Statistics and Stanford Linear Accelerator Center, Stanford University, Stanford, CA, 1996.

[66] K. Fukushima and S. Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[67] S. I. Gallant et al. Perceptron-based learning algorithms. *IEEE Transactions on neural networks*, 1(2):179–191, 1990.

[68] H. Gholamalinezhad and H. Khosravi. Pooling methods in deep neural networks, a review. 2020.

[69] B. S. Glicksberg, R. Miotto, K. W. Johnson, K. Shameer, L. Li, R. Chen, and J. T. Dudley. Automated disease cohort selection using word embeddings from electronic health records. In *Pac Symp Biocomput*, volume 23, pages 145–56. World Scientific, 2018.

[70] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

[71] P. Gomutbutra, K. Kanjanaratanakorn, and N. Tiyapun. Prevalence and clinical characteristics of probable rem behavior disorder in thai parkinson's disease patients. *Parkinsons Disease*, 2018.

[72] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[73] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[74] J. Haba-Rubio, B. Frauscher, P. Marques-Vidal, J. Toriel, N. Tobback, D. Andries, M. Preisig, P. Vollenweider, R. Postuma, and R. Heinzer. Prevalence and determinants of rapid eye movement sleep behavior disorder in the general population. *Sleep*, 41(2):zsx197, 2017.

[75] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick. Masked autoencoders are scalable vision learners. *arXiv preprint arXiv:2111.06377*, 2021.

[76] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020.

[77] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[78] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[79] P. He, X. Liu, J. Gao, and W. Chen. Deberta: Decoding-enhanced bert with disentangled attention. In *International Conference on Learning Representations*, 2020.

[80] J. Herlocker, J. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 1999 Conference on Research and Development in Information Retrieval*, Berkeley, CA, 1999.

[81] G. Hinton, N. Srivastava, and K. Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8):2, 2012.

[82] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313:504–507, 2006.

[83] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[84] T. T. Ho, T. Kim, W. J. Kim, C. H. Lee, K. J. Chae, S. H. Bak, S. O. Kwon, G. Y. Jin, E.-K. Park, and S. Choi. A 3d-cnn model with ct-based parametric response mapping for classifying copd subjects. *Scientific Reports*, 11(1):1–12, 2021.

[85] S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[86] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[87] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[88] E. Horvitz. *Computation and Action Under Bounded Resources*. PhD thesis, Department of Computer Science, Stanford University, Stanford, CA, 1990.

[89] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

[90] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[91] J. Howard and S. Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

[92] G. Hripcsak, D. J. Albers, and A. Perotte. Parameterizing time in electronic health record studies. *Journal of the American Medical Informatics Association*, 22(4):794–804, 2015.

[93] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, and W. Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[94] J. Hu, S. Li, Y. Yao, L. Yu, Y. Guanci, and J. Hu. Patent keyword extraction algorithm based on distributed representation for patent classification. *Entropy*, 20:104, 02 2018.

[95] Y. H. Hu, S. Palreddy, and W. J. Tompkins. A patient-adaptable ecg beat classifier using a mixture of experts approach. *IEEE transactions on biomedical engineering*, 44(9):891–900, 1997.

[96] H. Huang, L. Lin, R. Tong, H. Hu, Q. Zhang, Y. Iwamoto, X. Han, Y.-W. Chen, and J. Wu. Unet 3+: A full-scale connected unet for medical image segmentation. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1055–1059. IEEE, 2020.

[97] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.

[98] V. Ibáñez, J. Silva, and O. Cauli. A survey on sleep assessment methods. *PeerJ*, 6:e4849, 2018.

[99] A. Iranzo, J. L. Molinuevo, J. Santamaría, M. Serradell, M. J. Martí, F. Valldeoriola, and E. Tolosa. Rapid-eye-movement sleep behaviour disorder as an early marker for a neurodegenerative disorder: a descriptive study. *The Lancet Neurology*, 5(7):572–577, 2006.

[100] C. J. A coefficient of agreement for nominal scales. *Educ Psychol Meas*, 20(1):37–46, 1960.

[101] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[102] A. K. Jain. *Fundamentals of Digital Image Processing*. Prentice-Hall, Englewood Cliffs, NJ, 1988.

[103] G. James and T. Hastie. Generalizations of the bias/variance decomposition for prediction error. Technical report, Department of Statistics, Stanford University, Stanford, CA, 1997.

[104] Z. Jiang, T. Chen, T. Chen, and Z. Wang. Robust pre-training by adversarial contrastive learning. *Advances in Neural Information Processing Systems*, 33:16199–16210, 2020.

[105] A. E. Johnson, T. J. Pollard, L. Shen, L.-w. H. Lehman, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. Anthony Celi, and R. G. Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016.

[106] I. Jolliffe. Principal component analysis. In *International encyclopedia of statistical science*, pages 1094–1096. Springer, 2011.

[107] J. N. Kather, C.-A. Weis, F. Bianconi, S. M. Melchers, L. R. Schad, T. Gaiser, A. Marx, and F. G. Z"ollner. Multi-class texture analysis in colorectal cancer histology. *Scientific reports*, 6:27988, 2016.

[108] J. Kempfner, G. Sorensen, M. Nikolic, R. Frandsen, H. Sorensen, and P. Jennum. Rapid eye movement sleep behavior disorder as an outlier detection problem. *Journal of Clinical Neurophysiology*, 31(1):86–93, 2014.

[109] J. Kempfner, H. B. Sorensen, M. Nikolic, and P. Jennum. Early automatic detection of parkinson's disease based on sleep recordings. *Journal of Clinical Neurophysiology*, 31(5):409–415, 2014.

[110] A. Kendall, Y. Gal, and R. Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7482–7491, 2018.

[111] K. Kersting, T. Raiko, S. Kramer, and L. De Raedt. Towards discovering structural signatures of protein folds based on logical hidden Markov models. In *Proc. 8th Pacific Symposium on Biocomputing*, Kauai, HI, 2003.

[112] H. A. Khan, W. Jue, M. Mushtaq, and M. U. Mushtaq. Brain tumor classification in mri image using convolutional neural network. *Math. Biosci. Eng*, 17(5):6203–6216, 2020.

[113] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah. Transformers in vision: A survey. *ACM Computing Surveys (CSUR)*, 2021.

[114] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah. Transformers in vision: A survey. *ACM computing surveys (CSUR)*, 54(10s):1–41, 2022.

[115] Y. Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[116] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[117] J. Kittler. Feature selection and extraction. In T. Y. Young and K. S. Fu, editors, *Handbook of Pattern Recognition and Image Processing*. Academic Press, New York, NY, 1986.

[118] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-normalizing neural networks. *arXiv preprint arXiv:1706.02515*, 2017.

[119] S. Kok and P. Domingos. Extracting semantic networks from text via relational clustering. In *Proceedings of the Nineteenth European Conference on Machine Learning*, pages 624–639, Antwerp, Belgium, 2008. Springer.

[120] S. Kok, M. Sumner, M. Richardson, P. Singla, H. Poon, D. Lowd, and P. Domingos. The Alchemy system for statistical relational AI. Technical report, Department of Computer Science and Engineering, University of Washington, Seattle, WA, 2007. http://alchemy.cs.washington.edu.

[121] I. Kononenko. Combining decisions of multiple rules. In B. du Boulay and V. Sgurev, editors, *Artificial Intelligence V: Methodology, Systems, Applications*, pages 87–96. Elsevier, Amsterdam, Netherlands, 1992.

[122] J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[123] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[124] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[125] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47:498–519, 2001.

[126] M. Kubat, D. Flotzinger, and G. Pfurtscheller. Discovering patterns in EEG-Signals: Comparative study of a few methods. In *Proceedings of the Eighth European Conference on Machine Learning*, pages 366–371, Vienna, Austria, 1993. Springer.

[127] M. Kubat and G. Widmer, editors. *Proceedings of the ICML-96 Workshop on Learning in Context-Sensitive Domains*. Bari, Italy, 1996.

[128] J. Kukačka, V. Golkov, and D. Cremers. Regularization for deep learning: A taxonomy. 2017.

[129] G. Lai, W.-C. Chang, Y. Yang, and H. Liu. Modeling long-and short-term temporal patterns with deep neural networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 95–104. ACM, 2018.

[130] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[131] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[132] Y. LeCun et al. Generalization and network design strategies. *Connectionism in perspective*, 19(143-155):18, 1989.

[133] B. Lester, R. Al-Rfou, and N. Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, 2021.

[134] B. Li, H. Zhou, J. He, M. Wang, Y. Yang, and L. Li. On the sentence embeddings from pre-trained language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9119–9130, 2020.

[135] X. Li, H. Chen, X. Qi, Q. Dou, C.-W. Fu, and P.-A. Heng. H-denseunet: hybrid densely connected unet for liver and tumor segmentation from ct volumes. *IEEE transactions on medical imaging*, 37(12):2663–2674, 2018.

[136] X. L. Li and P. Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

[137] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee. Enhanced deep residual networks for single image super-resolution. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1132–1140. IEEE Computer Society, 2017.

[138] T. Lin, Y. Wang, X. Liu, and X. Qiu. A survey of transformers. *AI Open*, 2022.

[139] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[140] J. Ma, Z. Zhao, X. Yi, J. Chen, L. Hong, and E. H. Chi. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1930–1939, 2018.

[141] A. Malafeev, D. Laptev, S. Bauer, X. Omlin, A. Wierzbicka, A. Wichniak, W. Jernajczyk, R. Riener, J. Buhmann, and P. Achermann. Automatic human sleep stage scoring using deep neural networks. *Frontiers in neuroscience*, 12:781, 2018.

[142] mathworks. What is a convolutional neural network?

[143] S. J. McCarter, E. K. St Louis, and B. F. Boeve. REM sleep behavior disorder and REM sleep without atonia as an early manifestation of degenerative neurological disease. *Current neurology and neuroscience reports*, 12(2):182–192, 2012.

[144] C. McCormick. The skip-gram model, 2016.

[145] medium. 1d cnn for time series, 2019.

[146] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[147] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.

[148] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[149] D. J. Miller and H. Uyar. A mixture of experts classifier with learning based on both labelled and unlabelled data. *Advances in neural information processing systems*, 9, 1996.

[150] M. Minsky and S. Papert. Perceptrons. 1969.

[151] R. Miotto, L. Li, B. A. Kidd, and J. T. Dudley. Deep patient: an unsupervised representation to predict the future of patients from the electronic health records. *Scientific reports*, 6(1):1–10, 2016.

[152] M. ML. Interrater reliability: the kappa statistic. *Biochem Med (Zagreb)*, 22(3):276–82, 2012.

[153] M. H. Modarres, J. E. Elliott, K. B. Weymann, D. Pleshakov, D. L. Bliwise, and M. M. Lim. Validation of visually identified muscle potentials during human sleep using high frequency/low frequency spectral power ratios. *Sensors*, 22(1):55, 2022.

[154] H. Mohamed, A. Negm, M. Zahran, and O. C. Saavedra. Assessment of artificial neural network for bathymetry estimation using high resolution satellite imagery in shallow lakes: case study el burullus lake. In *International water technology conference*, pages 12–14, 2015.

[155] R. C. Moore and J. DeNero. L1 and l2 regularization for multiclass hinge loss models. 2011.

[156] R. Mottaghi, X. Chen, X. Liu, N.-G. Cho, S.-W. Lee, S. Fidler, R. Urtasun, and A. Yuille. The role of context for object detection and semantic segmentation in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 891–898, 2014.

[157] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

[158] Y. Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2). In *Doklady an ussr*, volume 269, pages 543–547, 1983.

[159] R. neural network. Rnn.

[160] A. Y. Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78, 2004.

[161] D. Nguyen, W. Luo, S. Venkatesh, and D. Phung. Effective identification of similar patients through sequential matching over icd code embedding. *Journal of medical systems*, 42(5):94, 2018.

[162] R. Nirthika, S. Manivannan, A. Ramanan, and R. Wang. Pooling in convolutional neural networks for medical image analysis: a survey and an empirical study. *Neural Computing and Applications*, pages 1–27, 2022.

[163] I. Nusrat and S.-B. Jang. A comparison of regularization techniques in deep neural networks. *Symmetry*, 10(11):648, 2018.

[164] D. of Health Human Services. Healthcare cost utilization project user support, 2012.

[165] C. Olah and S. Carter. Attention and augmented recurrent neural networks. *Distill*, 1(9):e1, 2016.

[166] K. O'Shea and R. Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[167] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[168] M. Y. Park and T. Hastie. L1-regularization path algorithm for generalized linear models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(4):659–677, 2007.

[169] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[170] J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[171] T. Pham, T. Tran, D. Phung, and S. Venkatesh. Deepcare: A deep dynamic memory model for predictive medicine. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 30–41. Springer, 2016.

[172] PhysioNet. Sleep-edf, 2013.

[173] R. B. Postuma, A. Iranzo, M. Hu, B. Högl, B. F. Boeve, R. Manni, W. H. Oertel, I. Arnulf, L. Ferini-Strambi, M. Puligheddu, et al. Risk and predictors of dementia and parkinsonism in idiopathic rem sleep behaviour disorder: a multicentre study. *Brain*, 142(3):744–759, 2019.

[174] R. Prabhavalkar, T. N. Sainath, B. Li, K. Rao, and N. Jaitly. An analysis of" attention" in sequence-to-sequence models. In *Interspeech*, pages 3702–3706, 2017.

[175] B. Pyakillya, N. Kazachenko, and N. Mikhailovsky. Deep learning for ecg classification. In *Journal of physics: conference series*, volume 913, page 012004. IOP Publishing, 2017.

[176] A. Quattoni and A. Torralba. Recognizing indoor scenes. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 413–420. IEEE, 2009.

[177] L. Rabiner, R. Crochiere, and J. Allen. Fir system modeling and identification in the presence of noise and with band-limited inputs. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(4):319–333, 1978.

[178] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. *arXiv preprint arXiv:2103.00020*, 2021.

[179] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[180] A. Rajkomar, E. Oren, K. Chen, A. M. Dai, N. Hajaj, M. Hardt, P. J. Liu, X. Liu, J. Marcus, M. Sun, et al. Scalable and accurate deep learning with electronic health records. *NPJ Digital Medicine*, 1(1):18, 2018.

[181] R. Ranftl, A. Bochkovskiy, and V. Koltun. Vision transformers for dense prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12179–12188, 2021.

[182] J. Rasmussen and H. Langerman. Alzheimer's disease–why we need early diagnosis. *Degenerative neurological and neuromuscular disease*, 9:123, 2019.

[183] L. Rasmy, Y. Xiang, Z. Xie, C. Tao, and D. Zhi. Med-bert: pretrained contextualized embeddings on large-scale structured electronic health records for disease prediction. *NPJ digital medicine*, 4(1):1–13, 2021.

[184] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[185] R. Rehurek and P. Sojka. Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, 3(2), 2011.

[186] C. Riquelme, J. Puigcerver, B. Mustafa, M. Neumann, R. Jenatton, A. Susano Pinto, D. Keysers, and N. Houlsby. Scaling vision with sparse mixture of experts. *Advances in Neural Information Processing Systems*, 34, 2021.

[187] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[188] F. Rosenblatt. *Perceptions and the theory of brain mechanisms*. Spartan books, 1962.

[189] H. R. Roth, L. Lu, A. Farag, H.-C. Shin, J. Liu, E. B. Turkbey, and R. M. Summers. Deeporgan: Multi-level deep convolutional networks for automated pancreas segmentation. In *International conference on medical image computing and computer-assisted intervention*, pages 556–564. Springer, 2015.

[190] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[191] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[192] H. Sak, A. Senior, and F. Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. 2014.

[193] D. Sarwinda, R. H. Paradisa, A. Bustamam, and P. Anggia. Deep learning in image classification using residual network (resnet) variants for detection of colorectal cancer. *Procedia Computer Science*, 179:423–431, 2021.

[194] C. H. Schenck, S. R. Bundlie, M. G. Ettinger, and M. W. Mahowald. Chronic behavioral disorders of human rem sleep: a new category of parasomnia. *Sleep*, 9(2):293–308, 1986.

[195] C. H. Schenck, S. R. Bundlie, and M. W. Mahowald. Delayed emergence of a parkinsonian disorder in 38% of 29 older men initially diagnosed with idiopathic rapid eye movement sleep behavior disorder. *Neurology*, 46(2):388–393, 1996.

[196] C. Schenk, S. Bundlie, and M. Mahowald. Rem behavior disorder: delayed emergence of parkinsonism and/or dementia in 65% of older men initially diagnosed with idiopathic rbd, and an analysis of the minimum and maximum tonic and/or phasic electromyographic abnormalities found during rem sleep. *Sleep*, 26:A316, 2003.

[197] C. Schenk, T. Hurwitz, and M. Mahowald. REM sleep behaviour disorder: an update on a series of 96 patients and a review of the world literature. *J Sleep Res*, 2:224–231, 1993.

[198] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.

[199] E. Sforza, J. Krieger, and C. Petiau. REM sleep behavior disorder: clinical and physiopathological findings. *Sleep medicine reviews*, 1(1):57–69, 1997.

[200] R. R. Shah and R. L. Smith. Addressing phenoconversion: the achilles' heel of personalized medicine. *British journal of clinical pharmacology*, 79(2):222–240, 2015.

[201] S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[202] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. 2017.

[203] B. Shickel, P. J. Tighe, A. Bihorac, and P. Rashidi. Deep ehr: a survey of recent advances in deep learning techniques for electronic health record (ehr) analysis. *IEEE journal of biomedical and health informatics*, 22(5):1589–1604, 2017.

[204] S. P. Singh, L. Wang, S. Gupta, H. Goli, P. Padmanabhan, and B. Gulyás. 3d deep learning on medical images: a review. *Sensors*, 20(18):5097, 2020.

[205] R. Strudel, R. Garcia, I. Laptev, and C. Schmid. Segmenter: Transformer for semantic segmentation. *arXiv preprint arXiv:2105.05633*, 2021.

[206] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[207] R. Sutton. Two problems with back propagation and other steepest descent learning procedures for networks. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 823–832, 1986.

[208] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[209] L. Uscher-Pines, J. Pines, A. Kellermann, E. Gillen, and A. Mehrotra. Deciding to visit the emergency department for non-urgent conditions: a systematic review of the literature. *The American journal of managed care*, 19(1):47, 2013.

[210] G. Van Horn, O. Mac Aodha, Y. Song, Y. Cui, C. Sun, A. Shepard, H. Adam, P. Perona, and S. Belongie. The inaturalist species classification and detection dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8769–8778, 2018.

[211] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[212] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec):3371–3408, 2010.

[213] P. Wallis and P. Danaee. Learning semantic relationships from medical codes. *AAAI Publications, The Thirty-Second International Flairs Conference*, 2019.

[214] P. Wallis and X. Song. Efficient fine-tuning of deep neural networks with effective parameter allocation. In *2022 IEEE International Conference on Image Processing (ICIP)*, pages 3510–3514, 2022.

[215] P. Wallis, D. Yaeger, A. Kain, X. Song, and M. Lim. Automatic event detection of rem sleep without atonia from polysomnography signals using deep neural networks. In *45th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4112–4116, 2020.

[216] X. Wang, R. Zhang, C. Shen, T. Kong, and L. Li. Dense contrastive learning for self-supervised visual pre-training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3024–3033, 2021.

[217] K. Weiss, T. M. Khoshgoftaar, and D. Wang. A survey of transfer learning. *Journal of Big data*, 3(1):1–40, 2016.

[218] H. Wu, B. Xiao, N. Codella, M. Liu, X. Dai, L. Yuan, and L. Zhang. Cvt: Introducing convolutions to vision transformers. *arXiv preprint arXiv:2103.15808*, 2021.

[219] S. Wu, S. Liu, S. Sohn, S. Moon, C.-i. Wi, Y. Juhn, and H. Liu. Modeling asynchronous event sequences with rnns. *Journal of biomedical informatics*, 83:167–177, 2018.

[220] Z. Wu, C. Shen, and A. Van Den Hengel. Wider or deeper: Revisiting the resnet model for visual recognition. *Pattern Recognition*, 90:119–133, 2019.

[221] Y. Xiang, J. Xu, Y. Si, Z. Li, L. Rasmy, Y. Zhou, F. Tiryaki, F. Li, Y. Zhang, Y. Wu, et al. Time-sensitive clinical concept embeddings learned from large electronic health records. *BMC medical informatics and decision making*, 19(2):139–148, 2019.

[222] D. Xie, J. Xiong, and S. Pu. All you need is beyond a good init: Exploring better solution for training extremely deep convolutional neural networks with orthonormality and modulation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5075–5084. IEEE Computer Society, 2017.

[223] H. Xie, Z. Qin, G. Y. Li, and B.-H. Juang. Deep learning enabled semantic communication systems. *IEEE Transactions on Signal Processing*, 69:2663–2675, 2021.

[224] B. Xu, X. Zhou, X. Li, C. Liu, and C. Yang. Diabetes mellitus carries a risk of esophageal cancer: a meta-analysis. *Medicine*, 96(35), 2017.

[225] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

[226] C. Yao, S.-M. Fereshtehnejad, M. R. Keezer, C. Wolfson, A. Pelletier, and R. B. Postuma. Risk factors for possible REM sleep behavior disorder: A clsa population-based cohort study. *Neurology*, 92(5):e475–e485, 2019.

[227] O. Yildirim, U. B. Baloglu, and U. R. Acharya. A deep learning model for automated sleep stages classification using psg signals. *International journal of environmental research and public health*, 16(4):599, 2019.

[228] D. Yu, H. Wang, P. Chen, and Z. Wei. Mixed pooling for convolutional neural networks. In *International conference on rough sets and knowledge technology*, pages 364–375. Springer, 2014.

[229] Y. Yu, X. Si, C. Hu, and J. Zhang. A review of recurrent neural networks: Lstm cells and network architectures. *Neural Computation*, 31(7):1235–1270, 2019.

[230] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. Dive into deep learning. *arXiv e-prints*, pages arXiv–2106, 2021.

[231] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao. Time series classification using multi-channels deep convolutional neural networks. In *International Conference on Web-Age Information Management*, pages 298–310. Springer, 2014.

[232] B. Zhou, H. Zhao, X. Puig, T. Xiao, S. Fidler, A. Barriuso, and A. Torralba. Semantic understanding of scenes through the ade20k dataset. *International Journal of Computer Vision*, 127(3):302–321, 2019.

[233] Z. Zhu, C. Yin, B. Qian, Y. Cheng, J. Wei, and F. Wang. Measuring patient similarities via a deep architecture with medical concept embedding. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 749–758. IEEE, 2016.

[234] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.