Domain Adversarial Training for Recurrent Language Models

Priyansh Trivedi

Matriculation number: 3005887

July 5, 2019

Master Thesis

Computer Science

Supervisors:

Prof. Dr. Jens Lehmann
Jun.-Prof. Dr. Asja Fischer

Institut für Informatik III

Rheinische Friedrich-Wilhelms-Universität Bonn

Declaration of Authorship

I, Priyansh Trivedi, declare that this thesis, titled "Domain Adversarial Training for Recurrent Language Models", and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. Except for such quotations, this thesis is entirely my own work. I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Acknowledgements

I would like to thank my supervisors, Prof. Dr. Jens Lehmann, and Jun.-Prof. Dr. Asja Fischer for their guidance, support and encouragement. Both Jens, and Asja sat through countless rounds of discussions and advice whenever I needed them. This work would not have been possible had Asja not entertained my barrage of half-baked ideas ever so often, or if Jens would not have proactively suggested tweaks, nudged me throughout the months. Thank you.

I thank my colleagues at the Smart Data Analytics (SDA) lab at the University of Bonn, and at the Enterprise Information Systems (EIS) at Fraunhofer IAIS. Constant interactions with you all helped me traverse the past years with glee and camaraderie. You all shaped me to be researcher I am now. A special shout-out to one of our newer member – sda-srv05, the server which burned the midnight oil as much as I did.

To my friends – Abhishek, Debanjan, Denis, Gaurav, Mayesha, Mohnish, Nilesh, Shantanu, and Yash, for reminding me "How do I eat an elephant? One bite at a time."

Most importantly, to my Mom and Dad, for always being in my corner.

Contents

List of Figures

List of Tables

Abstract

Deep Learning models solving supervised NLP problems benefit from the use of pretrained language models, as shown recently (Howard and Ruder, 2018; Radford et al., 2018). In this work, we investigate whether the domain shift incurred when transferring the language models on to the target task can be actively bridged. To that end, we incorporate domain adversarial training (Ganin et al., 2016) in this context. We, then evaluate this augmented training algorithm in a variety of transfer learning schemes, over the movie review classification task. Through our experiments we find that domain adversarial training can be beneficial in low resource scenarios, but is redundant on tasks with ample training data.

Keywords: Transfer Learning, Domain Adversarial Training, Pretrained Language Models, Natural Language Processing

Chapter 1

Introduction

Understanding and interpreting natural language has been a central tenet of intelligent systems, more so as human interactions are increasingly both conveyed and consumed textually. While the goal of human-esque understanding and interpretation of language has been generally elusive, progress in the field comes in the form of better solutions for natural language processing (NLP) problems which involve various degrees of language understanding, such as machine translation (Bojar et al., 2018), reading comprehension (Choi et al., 2018; Reddy et al., 2018; Nguyen et al., 2016; Rajpurkar et al., 2018), linked data question answering (Trivedi et al., 2017; Berant et al., 2013), text classification (Zhang et al., 2015; Maas et al., 2011), fact verification (Thorne et al., 2018), text simplification (Zhu et al., 2010), summarization (Völske et al., 2017) etc.[1]

Over the years, the performance of deep learning models have been increasingly improving over a wide variety of these problems, and in some cases reaching the inter-annotator agreement[2] levels (Devlin et al., 2018). This increase, while often incrementally brought about by the means of increased training data, deeper models (Schwenk et al., 2017; Cer et al., 2018), better model architectures (Maheshwari et al., 2018; Yu et al., 2018) or efficient hyperparameter search (Eggensperger et al., 2015; Snoek et al., 2012), is sometimes caused by notable innovations[3] such as the use of attention in sequence-to-sequence networks (Bahdanau et al., 2015; Luong et al., 2015), generative adversarial networks for text (Subramanian et al., 2017), cross

[1]For an excellent repository of such tasks, and state of the art approaches to solve them, visit `http://nlpprogress.com/`.

[2]The terminology inspired from this thread of tweets - `https://twitter.com/yoavgo/status/1111179424215502848`

[3]We use the word here very liberally, as many of the following *innovations* heavily draw upon, or simply retry previously proposed techniques with incremental modifications.

attention between input sequences for tasks involving comparison of input sequences (Parikh et al., 2016), and more recently, transformers based encoders (Vaswani et al., 2017; Devlin et al., 2018; Radford et al., 2019).

However, these approaches have a shortcoming, namely, that the model can only extract domain understanding from the given dataset while training to solve a particular problem. For instance, when training to classify movie reviews (in English language) as positive or negative, the model doesn't leverage countless other sources, training on which might imbue the model with better task/domain-invariant understanding of the language potentially improving its performance while classifying reviews. This inability causes noticeable hindrances when a particular dataset has inadequate number of examples, barring the model to ever reach satisfactory accuracy despite solving a simple problem.

The field of transfer learning aims to provide training mechanisms, model architectures, and other techniques to overcome the aforementioned shortcomings. Transfer learning (TL) over a domain D_i and task T_j can be loosely defined as leveraging data, or models trained on data outside the current domain or task, while training over T_j (Pan and Yang, 2010). The use of these methods has been demonstrated to enable faster model convergence (Howard and Ruder, 2018), better task performance in data rich domains (Devlin et al., 2018), and has enabled the use of deep learning techniques for problems in niche tasks and domains, with limited training examples (Nguyen and Chiang, 2017; Cohn et al., 2017; Ruder, 2019, Ch. 5). The impact of these approaches has been seen in the computer vision domain more profoundly where the use of models trained over largescale datasets for downstream tasks such as object detection (Girshick et al., 2014), semantic segmentation (Zhao et al., 2017) and human pose estimation (Cao et al., 2017) is the norm, rather than exception (Mahajan et al., 2018).

The counterpart of these approaches for NLP would be the use of pretrained language models for multiple downstream tasks (Ruder, 2018). Language models estimate the likelihood of a sequence of words, often computed as a product of conditional probabilities characterized by the following equation:

$$P(w_{0:T}) = P(w_0) \prod_{t=1}^{T} P(w_t | w_{0:t-1}) \tag{1.1}$$

where w_t is the t^{th} word token in a sequence of T such tokens. Recurrent neural language models estimate these conditionals often using an encoder-generator architecture where the encoder outputs fixed length vector representing the input words in a high dimensional space which is then used by the generator to output the conditional probabilities.

Using a pretrained encoder (aforementioned) in task specific neural architectures can induce desirable understanding of the language pivotal to boost model performance, and enable low resource training, akin to the advantages mentioned above. Some of the prevalent example of these techniques include the use of pretrained word embeddings (Mikolov et al., 2013; Pennington et al., 2014; Bojanowski et al., 2017), which are found in a majority of neural approaches for NLP (Li and Yang, 2018), albeit only as a part of the encoder. (Dai and Le, 2015) first demonstrated that using the entire encoder of a language model as an initialization for a task specific network can decrease the generalisation gap. However, their language model was pretrained on task-specific text which reduces the appeal of their technique for niche domains and small datasets. Other approaches (Mou et al., 2016) have demonstrated similar results using language models trained on generic (task invariant) text although with limited improvements. Further, (Howard and Ruder, 2018) demonstrate faster model convergence, and overall increase in performance by a combination of both task-invariant and task-dependent language models trained in succession. More recently, (Vaswani et al., 2017) propose a non-recurrent encoder, namely transformer, which uses a complex combination of multi-head self attention for sequence transduction. Transformers, when used by (Radford et al., 2018; Devlin et al., 2018; Radford et al., 2019) in quick succession, in the setting of our interest, namely by training as a general language model whose encoder is used for downstream tasks, significantly outperform the existing state of the art methods.

Motivated by the potential benefits of these family of sequential transfer learning (STL) approaches, we attempt to utilise auxiliary learning methods to further minimise the loss of information when transferring the encoder form the language model to the target task. More specifically, we intend to explore whether enforcing an *invariance* between the source and target domain, by the means of domain adversarial training (Ganin et al., 2016) be beneficial for the process.

Domain adversarial training, as proposed in (Ganin et al., 2016), is an augmentation to a neural network, which adversarially trains it to learn representations of source and target data which are indistinguishable, or are *domain invariant*. In a scenario where the source task is language modeling over general purpose text corpus, and the target task is any supervised NLP task, achieving domain invariance is akin to bridging the *domain shift* between the two. If it can be achieved without losing information needed to make predictions for the target task, we hypothesise that it may enable the model to generalise better over the target task, thereby further improving the aforementioned STL (pretrain-finetune) process.

To that end, we investigate the use of pre-trained recurrent language

model for the purposes of sentiment classification, and the way domain adversarial training affects the process. We pre-train our language models on a task invariant text corpus - Wikitext103 (Merity et al., 2017), and use two commonly used sentiment classification datasets (Maas et al., 2011), (Pang and Lee, 2004). Further, we also perform our experiments in a multi-task setting wherein our target task is solving both of the aforementioned datasets simultaneously, in a comprehensive combination of STL schemes.

Our experiments show that domain adversarial training effectively regularises the model which can decrease the error in some cases, depending upon the specific STL scheme and the amount of data on the target task. However, in some other, specially wherein there is ample labeled examples of the target task, we find that adding domain adversarial aspect to the training negatively affects the model performance. This, alongwith the results of further analysis suggests that while domain adversarial training is an effective form of regularisation, barring low resource tasks, it provides little tangible benefits in the sequential transfer learning scheme of our interest.

1.1 Thesis Structure

The rest of the thesis is structured as follows. In Chapter 2, we introduce numerous concepts pertinent to our work including natural language processing, deep learning, language modeling and transfer learning. Chapter 3 details the premise of our work, namely *the use of pretrained language models for downstream NLP tasks*. Chapter 4 details primarily the work of (Ganin et al., 2016) introducing domain adversarial training, and theoretical results on which it is built. We describe our experiments and their results in Chapter 5, and conclude our work in the last chapter, namely Chapter 6.

Chapter 2

Background

Before discussing our work, and the context surrounding it, we describe some concepts necessary for an in-depth understanding of our work. In the following sections, we shall provide an overview of natural language processing, language modeling and neural approaches to both, before moving on to outline the field of transfer learning in the context of our work.

2.1 Natural Language Processing

The field of natural language processing (NLP) aims to build solutions to problems involving varying degrees of understanding of human language. A *problem* in the context of our work, can be thought of as a task specific labeling scheme applied over samples from a domain, which we expect a particular NLP solution to predict. We begin the discussion of NLP approaches by borrowing the definition of domains and tasks from (Ruder, 2019), which offer a particular perspective suitable to this work.

As mentioned in (Ruder, 2019), a *domain* $\mathcal{D}$ consists of a feature space $\mathcal{X}$, and a marginal probability distribution $p_{\text{data}}(X)$ where $X = \{x_1 \dots x_n\} \in \mathcal{X}$. For instance, in the case of sentiment classification over movie reviews, the feature space $\mathcal{X} \subseteq \mathbb{W}^{|\text{V}|\times L}$ where V would be an arbitrary set of words from the reviews; L would be the maximum length of any review; x_i, the i^{th} review; and X is a set of reviews constituting an unlabeled dataset. Given a domain $\mathcal{D} = \{\mathcal{X}, p_{\text{data}}(X)\}$; a label space $\mathcal{Y}$, a prior distribution $p_{\text{data}}(Y)$ where $Y = \{y_i \dots y_n\}$, and a conditional distribution $p_{\text{data}}(Y|X)$ constitute a *task* $\mathcal{T}$. In our example, $\mathcal{Y}$ is the set of all possible labels i.e. {pos, neg}.

Solutions for such tasks, generally aim to estimate the conditional distribution $p_{\text{data}}(Y|X)$ by learning from labeled datasets consisting of pairs $x_i \in X$ and $y_i \in \mathcal{Y}$ (supervised learning); or the data distribution $p_{\text{data}}(X)$

by learning from unlabeled datasets consisting of samples $x_i \in X$ (unsupervised learning). Thus, in practice, we do not have access to the real distribution p_{data}, but only an approximation $\hat{p}_{data}$ denoting the datasets at hand.

Some of the earliest forays in the field which attempt to solve these tasks were symbolic in nature, premised on the perspective on language understanding as outlined in (Chomsky, 1957). These approaches aimed to create an exhaustive set of rules to controllably interpret the semantic structure of language, represented as expressions of a formal grammar following a finite set of compositional rules, and perform the necessary prediction based on it (Shank and Tesler, 1969; Dasgupta et al., 2018). Such systems are problem specific, their generalization constrained by the exhaustiveness of the rules meticulously set by domain experts.

In the past few decades, statistical approaches for understanding language and solving related tasks emerged, premised on the distributional hypothesis (Harris, 1954), which is eloquently put by (Firth, 1957) as "*You shall know a word by the company it keeps*". Instead of creating rules and decision surfaces manually, domain experts engineer a set of task-specific features and their corresponding extraction mechanisms which are used to represent the inputs. These representations could then be used by mathematical models (Cortes and Vapnik, 1995; Freund and Schapire, 1997) to learn the needed decision surfaces automatically and map the input (word, or word sequences) to output (labels or word sequences) thereby channeling the involvement of domain experts from rule creation to feature engineering. While relatively less brittle in nature, these approaches nevertheless involved significant efforts from domain experts in designing features relevant for the task and domain.

The field of NLP was further streamlined with the rise of deep learning approaches. Deep learning, a particular family of machine learning approaches, offer overparameterised models which are composed of *layers* of nonlinear functions, and are able to fit complex distributions efficiently. This enables the models to learn a latent set of features automatically given inputs and make necessary predictions, further transposing the task of feature engineering to that of model architecture selection, and hyperparameter tweaking.

2.2 Deep Learning

In this section we provide a brief introduction to the foundations of deep learning, skipping over, or summarising a vast amount of literature relevant to this work. Large proportions of this section contain material from

Deep Learning (Goodfellow et al., 2016), which we modify for notation's or brevity's sake. We reserve the focus of this section primarily towards discriminative models i.e. models which learn to predict an instance from the label space, when given instances from the feature space (roughly, supervised learning[1].).

Deep learning models, or neural networks are composite, parameterised functions which can be trained to output $\hat{y}_i \simeq y_i \in \mathcal{Y}$ when given $x_i \in X$; or generate samples $\hat{x}_i \simeq x_i \in X$. From a theoretical perspective, they can be seen as a family of probability distributions $p_{\text{model}}(, \theta)$, indexed by θ, which maps any configurations x to a real number estimating the true probability $p_{\text{data}}(y|x)$ (discriminative models), or in the case of generating samples, $p_{\text{data}}(x)$ (generative models). A deep learning model can then be seen as an arbitrary function f parameterized with θ which works as follows:

$$\hat{y}_i = f(x_i; \theta) \tag{2.1}$$

$$p_{\text{model}}(y_i|x_i; \theta) = \hat{y}_i \tag{2.2}$$

Here, $\hat{y}_i$ is the output of the model given x_i as the input. We would want a value of the index θ which aligns $p_{\text{model}}(, \theta)$ as close as possible to p_{data}. That is, for an input vector x_i, we would want the model's output vector $\hat{y}_i$ be a distribution over all possible labels, which peaks at y_i, where (x_i, y_i) are samples generated from p_{data}. A common way to do so uses the principle of (conditional) *maximum likelihood estimation* or conditional MLE, defined as:

$$\theta^{\text{ML}} = \underset{\theta}{\text{argmax}}\ p_{\text{model}}(Y|X; \theta) \tag{2.3}$$

which can be further resolved to:

$$\theta^{\text{ML}} = \underset{\theta}{\text{argmin}}\ \mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}} - \log\ p_{\text{model}}(\boldsymbol{y}|\boldsymbol{x}; \theta) \tag{2.4}$$

This equation suggests that we can find an optimum set of parameters for our model by treating it as an optimisation problem. The **loss function** (objective) of this optimisation problem would be *negative log likelihood* (NLL) computed over samples (x_i, y_i) sampled i.i.d. from $\hat{p}_{\text{data}}$[2], guided by the MLE principle. Another perspective on (conditional) MLE i.e. maximizing the likelihood is the same as minimizing the KL divergence[3], yields cross entropy

[1]The distinction between discriminative and generative models, or, supervised and unsupervised learning is fuzzy (Goodfellow et al., 2016, p. 106)

[2]Note that in Eqn. 2.4, $\hat{p}_{\text{data}}$ represents the empirical distribution based on sampled data at hand, since we have no direct access to the true data generating distribution p_{data}.

[3]`http://benlansdell.github.io/statistics/likelihood/`

between the training data and the model distribution as the loss function. We can in fact replace NLL with an arbitrary loss function $\mathcal{L}(f(x_i;\theta), y_i)$, which takes the model output over an instance, real label of the instance, and model parameters as the input, and returns a score representative of the prediction errors of the model. Eqn 2.4 can then be generalised as:

$$\theta^* = \underset{\theta}{\text{argmin}}\ \mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}}\mathcal{L}(f(\boldsymbol{x};\theta), \boldsymbol{y}) \tag{2.5}$$

$$= \underset{\theta}{\text{argmin}}\ \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(f(x_i;\theta), y_i) \tag{2.6}$$

where m is the number of labeled samples in a given dataset.

The actual surface of the loss function, in practice is very complex, and lies in a high dimensional space for most deep learning model (number of dimensions corresponding to number of parameters in a model, which can be as high as 350 million (Devlin et al., 2018)). This renders grid search or random search based solutions computationally intractable. The intractability is further compounded upon by the fact that the actual value of the loss surface at any point would require computing the loss function over the entire dataset (which are quite large, themselves). We thus use **gradient descent** to optimize the objective function - $\mathcal{L}$. Recall that the gradient of a function $f(x)$ is the vector containing all the partial derivatives, denoted by $\nabla_x f(x)$, and critical points are points where every element of the gradient is equal to zero. To minimize $\mathcal{L}$, we would like to find the direction in which $\mathcal{L}$ decreases the fastest, at any point in the parameter space θ, which is to say

$$\min_{u,u^Tu=1} \boldsymbol{u}^T\nabla_\theta\mathcal{L}(f(\mathbf{x};\theta), \mathbf{y}) \tag{2.7}$$

$$\min_{u,u^Tu=1} ||\boldsymbol{u}||_2||\nabla_\theta\mathcal{L}(f(\mathbf{x};\theta), \mathbf{y})||_2 cos\phi \tag{2.8}$$

where $\boldsymbol{u}$ is a unit vector representing the direction (in the parameter space) in which we wish to move, and ϕ is the angle between $\boldsymbol{u}$ and the gradient. Since the magnitude of any unit vector ($||\boldsymbol{u}||_2$) is 1, the gradient term isn't influenced by $\boldsymbol{u}$, Eqn. 2.8 can be resolved to $\min_{\boldsymbol{u}} cos\phi$. Thus, moving in opposite direction of $\nabla_\theta\mathcal{L}$ minimizes $\mathcal{L}$ the most. This leads to the following update step:

$$\theta^{'} = \theta - \eta\nabla_\theta\mathcal{L} \tag{2.9}$$

where, η is the **learning rate**, determining the magnitude of the update. This update step, if computed indefinitely (or more practically, *ample* number of times) will converge the model to a local minima (Boyd and Vandenberghe, 2004, Chapter 9.3.1). Note that convergence to a local minima here doesn't

imply that the loss function is at its lowest value (i.e. at a global minima), or even that its reasonably low, but only that the magnitude of the gradients of the loss functions is zero or close to zero.

While theoretically appealing, given the consistent nature of MLE, and the convergence property of gradient descent algorithm, this optimisation scheme has some practical problems. First, the loss surface of any neural network, even a single neuron has very many local minima and saddle point (Auer et al., 1995). Models trained based on (vanilla) gradient descent can get stuck (converge) on either of them. Further, a more practical problem is the fact that computing the loss over the entire dataset (required for every step) is computationally expensive, more so as complex tasks, and larger models require larger datasets to converge.

A slightly changed learning scheme - **stochastic gradient descent** (SGD) is thus more commonly used to train neural models. Instead of computing the loss over the entire dataset, training under SGD involves sampling a *mini-batch* of examples $B = x_1, \ldots x_{m'}$ drawn uniformly from the training set. This is done treating the gradient as an expectation, which can be approximately estimated using a small set of samples. Sampling mini batches from the dataset for each update step ensures a much reduced training time. Further, this stochastic nature introduces variations to the loss function and consequentially the gradients, reducing convergence time and possibly converging to a better minima (i.e. with lower loss).

This concludes, in the simplest form, the fundamental theory involving the training of deep learning models. While there's a wide variety of existing literature on the topic (Choromanska et al., 2015; Ioffe and Szegedy, 2015; Kingma and Ba, 2015; Loshchilov and Hutter, 2017; Bengio et al., 2009), here we outlined the simplest method in which this is accomplished, and refer interested readers to (Goodfellow et al., 2016, Chapter 8). We can now deconstruct a deep learning algorithm into (i) a labeled dataset denoted by $\hat{p}_{\text{data}}$, (ii) a loss function $\mathcal{L}(f(x_i; \theta), y_i, \theta)$, (iii) an optimisation procedure (e.g. stochastic gradient descent), and (iv) a model. In the rest of this section, we discuss some common model components and architectures relevant to this work.

2.2.1 Deep Learning Models

In this section we describe different deep learning models or neural networks that are used, and can be trained in the aforementioned manner. We begin the discussion by giving an intuitive description of what properties would we want these models to have, and then continue with its simplest variant.

Given our objective, that of modeling $\hat{p}_{\text{data}}$, and the aforementioned

method of training our models, namely, by gradient descent based optimisation, we would want to use functions which (i) are parameterised, and receptive to change in parameters, (ii) are able to learn complex representations of input data by learning simpler representations and *stacking* (or combining) them, and (iii) are non linear, since multiple linear transformations when combined can be represented by one linear transformation. Finally, being able to train them using gradient descent requires that these functions must be (iv) differentiable.

Consider the following linear transformation:

$$\hat{\mathbf{y}} = \mathrm{W}\mathbf{x} + \mathrm{b} \tag{2.10}$$

where $\mathbf{x} \in \mathbb{R}^N, \mathrm{W} \in \mathbb{R}^{N \times M}$ and $\hat{\mathbf{y}}, \mathrm{b} \in \mathbb{R}^M$. Treating W and b as parameters, we can represent it with the following function:

$$f_{\text{linear}}(\mathbf{x}; \theta) = \mathrm{W}_\theta \mathbf{x} + \mathrm{b}_\theta \tag{2.11}$$

where $f_{\text{linear}} : \mathbb{R}^N \mapsto \mathbb{R}^M$ is a parameterised function which can compute an M-dimensional vector given N-dimensional vectors as inputs. Known in existing literature as a **linear regressor**, this function can be trained to weigh the N different features of input differently and produce an M dimensional vector. For classification over an M dimensional label space $\mathcal{Y}$, we can interpret f_{linear}'s output as a score over the M classes, and can choose the class with the highest score as the model's prediction. We can alternatively normalise, and interpret its output ($\hat{\mathbf{y}} = f_{\text{linear}}(\mathbf{x}; \theta)$) as a distribution $p_{\text{model}}(\mathbf{y}|\mathbf{x})$.

A *linear regressor* thus satisfies most of our aforementioned requirements except for non-linearity. To imbue non-linearity, we can simply augment $\hat{\mathbf{y}}$ s.t. $\hat{\mathbf{y}} = g \circ f$, where g is a non linear function, known as the **activation function**. The most common activation function, namely the *sigmoid function* ($\sigma(\mathrm{z})$), defined as

$$\sigma(\mathrm{z}) = \frac{1}{1 + e^{-\mathrm{z}}} \tag{2.12}$$

is a bounded, differentiable and monotonic function $\sigma : \mathbb{R} \mapsto (0, 1)$. Its generalisation to multi-dimensional domain and range is known as the **softmax function**.

$$\text{softmax}(\mathbf{z})_i = \frac{e^{\mathbf{z}_i}}{\sum_{j=1}^{M} e^{\mathbf{z}_j}} \tag{2.13}$$

Softmax is most often used as the activation function over the final outputs of a model (and rarely anywhere else), given that it can interpret an n-dimensional vector as a probability distribution over n classes.

Other common choices of activation functions include the hyperbolic tangent function $tanh : \mathbb{R} \mapsto (-1, 1)$, and the more commonly used Rectified Linear Unit $g(\mathrm{z}) = \max\{0, \mathrm{z}\}$ or (ReLU)[4].

This constitutes the simplest building block of neural networks, namely - a **feed forward layer**, composed of a linear function $f_{\text{linear}}(\mathbf{x}; \theta)$, and an activation function. We can trivially *stack* these layers to constitute a complex non-linear parameterised function, which can be trained to fit a distribution. From a bottom-up perspective, all but the final layers are called *hidden layers*, and layer whose outputs constitute the model's output is called *output layer*. The output of one layer acts as the input to the next, and the process in which these inputs are passed across layers is called *forward propagation*. Correspondingly, during learning, we compute the gradients of the model one layer at a time. Using the chain rule of calculus, only the gradients of the layer directly above the current one affects its gradients, making the process of computing the gradients computationally efficient. The process in which these gradients are computed layer by layer (from top to bottom) is called *backpropagation*. For example, a two layered feed forward network[5] f_{mlp} with two hidden layers can be characterised as follows:

$$\begin{aligned} \mathbf{h}_1 &= g_1(\mathrm{W}_1\ \mathbf{x} + \mathrm{b}_{a1}) \\ \mathbf{h}_2 &= g_2(\mathrm{W}_2\ \mathbf{h}_1 + \mathrm{b}_2) \\ \hat{\mathbf{y}} &= \mathrm{W}_{out}\ \mathbf{h}_2 + \mathrm{b}_2) \\ p_{\text{model}}(y|\mathbf{x}) &= \text{softmax}(\hat{\mathbf{y}}) \end{aligned} \tag{2.14}$$

We discuss some commonly used layers in the next subsection.

2.2.2 Layers in a Deep Learning Model

While generic in nature, feedforward layers are inefficient in terms of parameters used. For instance, when making predictions over sequential inputs (of l length) belonging to an n dimensional feature space, and making predictions over an m dimensional label space, a single layered feed forward network would contain $l \times n \times m + m$ parameters. In practice, we rarely use single layered networks and input spaces contain hundreds of dimensions resulting in tens of thousands of parameters in the simplest cases.

[4]While ReLU is non-differentiable at $\mathrm{z} = 0$, in practice the gradients of the models never reach 0, and thus "it is acceptable for the minima of the cost function to correspond to points with undefined gradient" (Goodfellow et al., 2016, Chapter 6.3).

[5]A feedforward network is a neural network composed of feed forward layers, and is also commonly referred to as a *multi-layer perceptron* or MLP

Further, these models are agnostic to structure of the inputs. That is to say that they would treat a sequence of 5 inputs having 125 dimensions each, same as they would consider a single unstructured input of 625 dimensions, or an image (matrix) of 25×25 greyscale pixels. This is disadvantegous because structural information can be pivotal when making predictions. For instance, while segmenting images into binary classes, it is more likely that a pixels adjacent to an already labeled pixel share the same class, than not.

Owing to these reasons, common neural models contain layers which can actively take advantage of input structures.

Recurrent Layers

Recurrent neural networks or RNNs (Rumelhart et al., 1988) are a family of neural networks which can harness sequential nature of data $\langle \boldsymbol{x}^{(1)}, \dots \boldsymbol{x}^{(T)} \rangle \in \mathcal{X}$. In these cases, inputs at different time steps ($\boldsymbol{x}^{(t)}$) belong to same latent space. In the case of text classification, for instance, we expect them to each belong to a shared space of possible words.

Premised on this, recurrent models share parameters when processing inputs at different time steps. Seen from another perspective, they process the inputs repeatedly - applying the same parameterised transformation over each vector ($\boldsymbol{x}^{(t)}$), one time step at a time. This enables the model to generalise over sequences of multiple different lengths.

Further, to be able to make inferences across different positions in time, recurrent models pass a fixed length vector across time steps, often referred to as the *hidden state*, or the *memory* which gets updated at each time step. We can define them using the following equation:

$$\mathbf{h}^{(t)} = g_{\text{rnn}}(\mathrm{W}\mathbf{x}^{(t)} + \mathrm{U}\mathbf{h}^{(t-1)} + \mathrm{b}) \tag{2.15}$$

where g is an activation function (e.g. tanh), $\mathbf{x}^{(t)}$ is the t^{th} input in the sequence, $\mathbf{h}^{(t-1)}$ is the hidden state corresponding to the $t - 1^{th}$ input. The weight matrices W, U and the bias b parameterise this layer, and are shared across time steps.

One straight-forward limitation of these models is directionality. The flow of information in Eqn. 2.15 is left-to-right which might be sub-optimal when making decisions involving elements across the sequences, or right-to-left scripts (e.g. Hebrew sentences). A **bidirectional RNN** (or BiRNN) aims to partially rectify these limitations by using an RTL (right to left) RNN alongwith the conventional LTR (left to right) one, and combining their hidden states at each time to produce the final summary vector, as

follows:

$$\begin{aligned}
\mathbf{h}_l^{(t)} &= g_{\text{rnn}}(\mathrm{W}_l\mathbf{x}^{(t)} + \mathrm{U}_l\mathbf{h}_l^{(t-1)} + \mathrm{b}_l) \\
\mathbf{h}_r^{(t)} &= g_{\text{rnn}}^{'}(\mathrm{W}_r\mathbf{x}^{(t)} + \mathrm{U}_r\mathbf{h}_r^{(t+1)} + \mathrm{b}_r) \\
\mathbf{h}^{(t)} &= [\mathbf{h}_l^{(t)}, \mathbf{h}_r^{(t)}]
\end{aligned} \tag{2.16}$$

where $\mathrm{W}_l, \mathrm{W}_r, \mathrm{U}_l, \mathrm{U}_r, \mathrm{b}_l, \mathrm{b}_r$ are parameters of the layer and shared across time steps.

Multiple augmentations have been proposed to rectify other shortcomings of these models such as vanishing or exploding gradients, information loss across long time steps (Hochreiter, 1991; Bengio et al., 1993, 1994a) etc. We discuss them briefly, below.

Long Short-Term Memory (LSTM) Layers

Hochreiter and Schmidhuber (1997) proposed a Long Short-Term Memory (LSTM) cell, an augmented recurrent cell, which can explicitly retain information over longer periods of time. An LSTM cell transfers not just the last layer's hidden output $\mathrm{h}^{(t)}$ across time steps, but an internal cell state $\mathrm{c}^{(t)}$ on which the cell's hidden outputs depend. Further, using, removing and adding information to the cells is managed by the means of *output*, *forget*, and *input* gates. For brevity's sake, we omit an in depth explanation of each gate, and refer interested readers to an excellent explanation of the topic - (Olah, 2015), and directly mention the equations which characterise an LSTM cell:

$$\begin{aligned}
\mathbf{f}^{(t)} &= g_g(\mathrm{W}_f\mathbf{x}^{(t)} + \mathrm{U}_f\mathbf{h}^{(t-1)} + \mathrm{b}_f) \\
\mathbf{i}^{(t)} &= g_g(\mathrm{W}_i\mathbf{x}^{(t)} + \mathrm{U}_i\mathbf{h}^{(t-1)} + \mathrm{b}_i) \\
\mathbf{o}^{(t)} &= g_g(\mathrm{W}_o\mathbf{x}^{(t)} + \mathrm{U}_o\mathbf{h}^{(t)} + \mathrm{b}_o) \\
\mathbf{c}^{(t)} &= \mathbf{f}^{(t)} \circ \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \circ g_c(\mathrm{W}_c\mathbf{x}^{(t)} + \mathrm{U}_c\mathbf{h}^{(t-1)} + \mathrm{b}_c) \qquad (2.17) \\
\mathbf{h}^{(t)} &= \mathbf{o}^{(t)} \circ g_h(\mathbf{c}^{(t)}) \qquad (2.18)
\end{aligned}$$

where $\mathbf{f}^{(t)}, \mathbf{i}^{(t)}$ and $\mathbf{o}^{(t)}$ represent the different gates which dictate the edit operations on the cell state $\mathbf{c}^{(t)}$ as depicted in Eqn. 2.17. Further, g_g are sigmoid activations, g_c, g_h are *hyperbolic tangent* activations, $\circ$ represents the Hadamard product, and $\mathrm{W}_*, \mathrm{U}_*$, and b_* are parameters, invariant to time steps. Akin to the RNN layer, a layer of LSTM can be trivially modified for bidirectionality.

Different variations of these recurrent units have been proposed, including Gated Recurrent Unit (GRU) (Cho et al., 2014), Minimal Gated Unit (MGU) (Zhou et al., 2016).

Convolutional Layers

Convolutional Neural Networks (CNNs) proposed in (LeCun et al., 1989) are neural layers specializing in processing data with grid like topology such as images (2D grid), or even sequential data which can be thought of as a 1D grid. As the name suggests, these layers involve the convolutional operation using parameterised multi-dimensional kernels. These kernels are used to compute a weighted average over subset of input values to produce one value of the output representation: $h_j^{\text{conv}} = \mathrm{W}_k \mathbf{x}_{j-m/2:j+m/2}$ where W_k is a kernel of m dimensions, and $\mathbf{x}$ is a 1D input of n. By sliding this kernel over the length of $\mathbf{x}$, we can compute an output representations of $n - m + 1$ dimensions. This operation of *sliding* the kernel over an input is called the *convolution operation*, and can be generalised for multidimensional inputs and kernels.

When an input passes through a feed-forward layer, each output value m is influenced by each input value n correspondingly weighted by $W_{m,n}$, a scalar parameter. When processing structured inputs, this is disadvantegous because of the increased number of parameters, as well as tasking the model to be spatially aware. Convolutional layers instead have *sparse interactions* as the inputs are convolved with kernels that are significantly smaller in size. Further, given that we slide the same kernel across the inputs, as opposed to having different parameters corresponding to each term in the output vector, we in effect *share parameters*, similar to Sec. 2.2.2. This invariance helps to train the kernels to "find" a specific feature over the inputs, for instance an edge in an image, or in less abstract layers a geometric shape, or a written character. In practice, we have multiple kernels (with independent parameters in each) produce multiple output representations of the input corresponding to the presence of feature each kernel is trying to find, giving the kernel their other commonly used name - a feature map. Note that we use the term feature not to denote human-interpretable features but rather a latent high dimensional representation in a space with little co-relation to interpretable concepts.

These multiple outputs are then passed through activation functions, and finally through a **pooling layer** which "replace the output of the net at a certain location with a summary statistic of the nearby outputs" (Goodfellow et al., 2016, Chapter 9.3), making the outputs become more invariant to slight variations in the input. A convolution layer, thus, is composed of (i) a convolution operation, (ii) an activation, and (iii) a pooling function. Models composed primarily of convolution layes have shown tremendous performance over multiple tasks, primarily, but not restricted to the computer vision domain (Zhao et al., 2017; Cao et al., 2017; Girshick et al., 2014). More recently, their use for NLP tasks has been explored, and has shown

promising results (Kim, 2014; Mou et al., 2015).

Time Distributed Feed Forward Layers

A time distributed feed forward layer is feed forward layer applied on one element of the input sequence at a time. Another perspective on the matter would be to think of them as a recurrent layer without a hidden state. We can describe them simply by the following equations:

$$\hat{\mathbf{y}}^{(t)} = g(\mathrm{W}\mathbf{x}^{(t)} + \mathrm{t}) \tag{2.19}$$

which we can alternatively represent as:

$$\hat{\mathbf{y}}^{(t)} = g(f_{\text{mlp}}(\mathbf{x}^{(t)}; \theta)) \tag{2.20}$$

In practice, we use this to classify tokens (as explained below), and more commonly to embed input word tokens into a low dimensional vector space, typically as the first layer in models working with natural language. The latter, known as the **embedding layer** uses no activation function. We discuss the use and the process of training embedding layer at length in the next section - Sec. 2.3.2.

This concludes a quick summary of neural layers which hold relevance to our work, enabling us to finally discuss the ways in which they can be combined to form models specific to a given task. While extremely broad in nature, we focus the discussion to the use of recurrent layers, and solving supervised NLP tasks.

2.2.3 Deep Learning Architectures for NLP Tasks

Based on the domain $\mathcal{D}(\mathcal{X}, p_{\text{data}}(X))$, and the task $\mathcal{T}(\mathcal{Y}, p_{\text{data}}(Y), p_{\text{data}}(Y|X))$, we can broadly classify tasks which involve some degree of textual understanding into four categories mentioned in this subsection.

Remark. We shall use f_{rnn} as an abstract function which may be composed of multiple recurrent layers, with or without bidirectionality, have any activation function, and may use any recurrent cell including RNNs, LSTMs etc. Similarly, f_{mlp} may be used to denote one feedforward layer, or a combination of multiple such layers, with any activation function on each.

Sequence Level Classification

Given a sequence of word tokens $x_i = x_i^{(1)} \dots x_i^{(T)}$, we wish to classify the sequence x_i into one of n classes. Common examples of such tasks include

sentiment classification, topic classification etc. In these cases, the feature space $\mathcal{X}$ (from which input sequences are sampled) can be represented by V^L where V is the set of all unique tokens, also known as the *vocabulary*, and L represents the maximum length of sequences. The label space $\mathcal{Y}$ is a much smaller space of possible classes, depending on the task.

When making predictions about sequences, we would want to produce a comprehensive summary of factors contributing to the summary, and learn to make predictions based on the summary. This is accomplished using a two step network wherein the first sub-network (or layers) encodes input text as a fixed-length vector in a latent representation space. Thereafter, a classifier is tasked with using the vector (representing the input sequence) and create a distribution over a label space. We cumulatively refer to the layers which create an encoded representation of the text as *encoder*. In practice the encoder consists of the aforementioned embedding layer (or a time distributed feed forward layer), and a recurrent layer. The following equations provide an abstract summary of the same:

$$\mathbf{x}_{\text{emb}}^{(t)} = f_{\text{emb}}(\mathbf{x}^{(t)}; \theta_{\text{emb}}) \tag{2.21}$$

$$\mathbf{h}_{\text{enc}}^{(t)} = f_{\text{rnn}}(\mathbf{x}_{\text{emb}}^{(t)}; \theta_{\text{emb}}) \tag{2.22}$$

$$\hat{\mathbf{y}} = f_{\text{mlp}}(\mathbf{h}_{\text{enc}}^{(T)}; \theta_{\text{mlp}}) \tag{2.23}$$

where f_{emb} and f_{rnn} collectively constitute the *encoder*, and f_{mlp} is called the classifier which takes the *final* hidden state of the RNN as the sequence's summary. A common variation of this network uses a more comprehensive summary of the encoded representations by concatenating the mean and max pooled (across time) representations of the hidden states $\mathbf{h}_{\text{enc}}^{(1)} \dots \mathbf{h}_{\text{enc}}^{(T)}$ along with the final state.

Token Level Classification

Token level classification is a variation of the aforementioned task wherein instead of making one prediction for the entire sequence, we wish to classify each element of a sequence in one of n classes. Common examples of which include Part of Speech tagging, named entity recognition etc.

To solve this task, we modify the network above by using a time distributed feed forward layer (See Sec. 2.2.2), as opposed to a regular MLP. We can thus modify Eqn. 2.23 as follows:

$$\hat{\mathbf{y}}^{(t)} = f_{\text{mlp}}(\mathbf{h}_{\text{enc}}^{(t)}; \theta_{\text{mlp}}) \tag{2.24}$$

to solve the task. This modification allows us to make prediction corresponding to each hidden state (representing a summary of the sequence up-to that point).

Interestingly, by keeping the label space same as the feature space, we can train the model to predict the next word given the sequence so far. This task is referred to as *language modeling*, and as we discuss in the next section (Sec. 2.3) can impart a deeper understanding of the given language.

The neural architecture defined here, and above (Equations (2.21) to (2.23)) are collectively referred to as an **encoder-classifier** network.

Sequence Generation

Instead of classifying a given sequence into one of n classes, some task might involve generating text given a particular input. Some examples include generating captions given an image, or generating sentences given a particular keyword.

Neural approaches to solve this task include the concept of a **conditional recurrent layer**. So far we've seen a recurrent layer as a function which encodes an input sequence's elements by the means of the cell state. Instead, we can think of it as a generative model which generates a hidden state vector *conditioned* on the input. Recall that by the means of a time distributed feed forward layer, we can further transform the hidden state vector to a distribution over the vocabulary. We can then extend this notion of conditional generation, and train the RNN to generate a sequence of words conditioned not only on a given input sequence (language modeling), but an arbitrary *context* vector.

The task of sequence generation then boils down to creating a fixed length *context* vector given an input from an arbitrary feature space, and then using it to condition a recurrent layer to generate the desired sequence. The latter (combination of an RNN and a time distributed feed forward layer) is referred to as the *decoder* of the model. As above, we refer to the part of the model which encodes the given input to a fixed length context vector as the encoder. The resulting model can be described by the following equations:

$$\mathbf{h} = f_{\text{enc}}(\mathbf{x}; \theta_{\text{enc}}) \tag{2.25}$$

$$\mathbf{s}^{(t)} = f_{\text{rnn}}(\mathbf{h}, \mathbf{s}^{(t-1)}; \theta_{\text{rnn}}) \tag{2.26}$$

$$\hat{\mathbf{y}} = f_{\text{mlp}}(\mathbf{s}^{(t)}; \theta_{\text{mlp}}) \tag{2.27}$$

where f_{enc} is an arbitrary encoder, and f_{rnn} is conditioned on the encoder's output $\mathbf{h}$.

Sequence Transduction

Tasks which fall under this classification involve trying to predict a sequence given another. Typical examples include machine translation, wherein we

wish for the model to generate a sequence having the same semantic meaning, but in a different language, or paraphrasing wherein we wish for the model to rephrase one or many sentence keeping their semantic meaning intact.

To solve this, we can modify the network above to create a summary of a sequence of text. That is to say, that we can replace f_{enc} in Eqn. 2.25 with a combination of an embedding layer and a recurrent layer (Equations (2.21) to (2.22)). Here, we can use both, the last hidden state of the encoding RNN, or its pooled summary (across time) as the context vector. These models are called encoder-decoder, or sequence-to-sequence networks.

Remark. This classification excludes tasks like textual entailment, reading comprehension, knowledge graph question answering, semantic parsing etc. However, some of these tasks can be treated as a special case of the aforementioned. For instance, textual entailment, where we wish to predict whether a sentence is logically followed by another can be modeled as a case of sequence level classification, where the two sentences are concatenated and passed to the model which is tasked to make a binary prediction regarding the entailment.

2.3 Language Modeling

While briefly mentioned in the previous section, we shall now discuss language modeling and its various aspects, including the pertinence of the topic to our work.

Language modeling is one of the fundamental task in the field of statistical NLP. In essence, it requires an understanding of a natural language based on which, a sequence of words can be assigned a probability, representing the model's confidence in the plausibility of the statement. In other words, given a sequence of words $w_i = w_i^{(1)}, \dots w_i^{(T)}$ where each word $w_i^{(t)}$ belongs to a fixed vocabulary V, we require the language model to learn to predict $p_{\text{data}}(w_i^{(1)}, \dots w_i^{(T)})$. This can be calculated by a product of conditionals as follows:

$$p(\mathrm{w}^{(1)}, \dots, \mathrm{w}^{(T)}) = \prod_{t=1}^{T} p(\mathrm{w}^{(t)} \mid \mathrm{w}^{(1)}, \dots, \mathrm{w}^{(t-1)}) \tag{2.28}$$

Based on this, we can task the models with predicting the next word given a sub-sequence of input sequence, iterating one word at a time, and train them so.

2.3.1 n-gram Models

One of the simplest ways to model the aforementioned conditionals is by the use n-grams. An n-gram is a sequence of n word tokens. For instance, a bigram (or 2 gram) model consists of pairs of 2 words like "hello world", "domain invariance" etc, along with a score $p_{\text{2-gram}}('world'|'hello')$. Thus, in an n-gram model, the probability of m^{th} word of a sequence depends upon the past $n-1$ words preceding it. Eqn. 2.28 can thus be appropriated as follows:

$$p(\mathrm{w}^{(1)}, \dots \mathrm{w}^{(T)}) = p(\mathrm{w}^{(1)}, \dots, \mathrm{w}^{(m-1)}) \prod_{t=m}^{T} p(\mathrm{w}^{(t)} \ |\mathrm{w}^{(t-n+1)}, \dots \mathrm{w}^{(t-1)}) \quad (2.29)$$

These models can be trained with the MLE principle (Sec. 2.2) "simply by counting how many times each possible n gram occurs in the training set." (Goodfellow et al., 2016, Ch. 12.4.1). By nature, n-grams with short values of n cannot capture long range dependencies of the language, and increasing n leads to an exponential increase in the possible word combinations, plaguing the model with data sparsity

2.3.2 Word Embeddings

Word embeddings, a neural approach to language modeling overcome the aforementioned shortcomings by producing a distributed representation of words. Instead of assigning a unique representation of each word, and explicitly modeling the relations between them, in a distributed representation scheme, each word is assigned a vector of value ("a pattern of activations") in a shared space. The meaning of the words and their relations to others are captured by the activations in the vector and the similarity between them (Goldberg, 2017, Ch. 10.4). This results in words with similar meaning being neighbours in the shared space.

Numerous approaches exist for designing and training word embeddings model. Notably, Mikolov et al. (2013) propose one of the most popular approach for training word embeddings namely, skip-gram models. Skip-gram models are trained to predict the context given a target word, i.e. words surrounding the target words. This is done by optimising the following loss function:

$$\mathcal{L}_{\text{SGNS}} = -\frac{1}{|\mathcal{C}|} \sum_{t=1}^{|\mathcal{C}|} \sum_{-C \le j \le C, j \neq 0} \log p_{\text{model}} \left(x^{(t+j)} | x^{(t)} \right) \quad (2.30)$$

where $x^{(1:T)}$ is a sequence sampled from the training data. In practice $x^{(t)}$ represents the t^{th} word occurring in the sequence, as an index of the vocabulary V (the feature space for the task). Using this, we can then define a skip-gram model as:

$$\mathbf{x}_{\text{emb}}^{(t)} = \mathrm{W}_{\text{emb}}\mathbf{x}^{(t)} \tag{2.31}$$

$$\hat{\mathbf{y}} = \text{softmax}(\mathrm{W}_{\text{context}}\mathbf{x}_{\text{emb}}^{(t)}) \tag{2.32}$$

$$p_{\text{model}}(\mathbf{x}^{(t+j)}|\mathbf{x}^{(t)}) = \hat{\mathbf{y}} \tag{2.33}$$

where W_{emb} is a linear layer (without a bias or an activation) with |V| input dimensions and an arbitrary hidden dimension (typically 300), also known as the **embedding layer**, and correspondingly $\mathrm{W}_{\text{context}}$ is another linear layer known as the context layer which maps the transforms the hidden space to the output space.

Here the embedding layer (or the embedding matrix) is of interest to us, since it transforms a word index to a low dimensional representation space. This is used in downstream models by using the weights of a pretrained embedding layer to initialise f_{emb}, the bottom layer of most encoder based architectures discussed in Sec. 2.2.3. We discuss their use in more detail in the next chapter.

2.3.3 Recurrent Language Models

The embedding models while beneficial across a wide variety of tasks, are shallow networks which take a limited context of words into accounts. Thus in a conditional word prediction task involving long term dependencies, we don't expect the embedding models to excel. Further, these models treat language not as a sequence but as a bag of words. To be able to make inferences based on the sequential nature of words, we can use recurrent networks trained on the language modeling task.

As described in Sec. 2.2.3, we can treat language modeling as a token based classification task wherein we train the model to predict the next word given the sequence so far. Given an arbitrary corpus of unlabeled text, which we have practically endless sources of, we can use the following labeling function: $f(\mathbf{x}^{(t)}) = \mathbf{x}^{t+1}$ to create a supervised dataset for language modeling whose feature and label spaces are both V (the vocabulary).

We can define our recurrent language model using Eqn. 2.21, Eqn. 2.22, Eqn. 2.24 based on the encoder-classifier model. In the context of language models, we refer to this architecture as the *encoder-generator* architecture given that f_{mlp} (Eqn. 2.24) is used to generate the next word. Using the

output of the classifier as a distribution over the label space gives us a p_{model} which we can use to train our model using the MLE principle:

$$p_{\text{model}}(\mathbf{y}^{(t)}|\mathbf{x}^{(1:t)}) = \hat{\mathbf{y}}^{(t)} \tag{2.34}$$

The recurrent encoder, in this model enables it to capture sequential information, and therefore model more complex phenomenon like long term dependency (Linzen et al., 2016), and hierarchical relations (Gulordava et al., 2018). Notably, given that the same encoder architecture is used in multiple forms of NLP models (including sequence classification, transduction models), we can potentially use the encoder of an already trained language model to initialise other task specific models, imparting them with a better understanding of the language. In the next section, we shall outline the field of transfer learning, the use of pretrained models falls under whose purview.

2.4 Transfer Learning

In so far, we discussed the manners in which neural models can solve a wide variety of NLP tasks. This isn't to suggest that these models *understand* the language in a sense that we're familiar with. Only, that these models can fit very complex distributions given ample labeled data to train on. However, labeled data is not always easy to come by. When working with low resource languages or on complex tasks with no publicly available datasets, creating labeled data can be difficult, hindering the use of neural approaches. The field of transfer learning allows us to deal with these scenarios by leveraging the data of some related task or domain, and increasing the generalisation capabilities of models to perform better on out of domain data. In this section, we provide a short overview of the field, focusing primarily on aspects which are of relevance to our work, namely *inductive transfer learning*. For a comprehensive summary, we refer interested readers to Chapter 3 of Ruder (2019).

We begin by providing a formalism which can ground the discussion below to consistent notations and expressions. Recall, that we can describe any domain $\mathcal{D}$ by a feature space $\mathcal{X}$, and a marginal distribution $p_{\text{data}}(X)$; and any task $\mathcal{T}$ by a label space $\mathcal{T}$, a marginal distribution of labels $p_{\text{data}}(Y)$, and a conditional distribution $p_{\text{data}}(Y|X)$. Given a source domain $\mathcal{D}_a$ and task $\mathcal{T}_a$, and a target domain $\mathcal{D}_b$ and task $\mathcal{T}_b$, we wish to be able to leverage the source $\mathcal{D}_a$ and $\mathcal{T}_a$ while learning to model $p_{\text{data}}(Y_b|X_b)$.

Whether information from the source is transferable to the target depends on the similarity between the two. It would be generally easier to achieve a good performance on the target task if for instance the they both involve

inputs sampled from articles within the same genre, as opposed to articles from different genres and languages. This notion of difference between the domains of source and target task is called the **domain shift**, and is discussed at length in Sec. 4.3.2. A similar notion, that of difference between the labeling distribution between the two is called the **task shift**.

2.4.1 Multi-task Learning

Under multi-task learning or MTL, we train the model to solve both the tasks at the same time. This is usually accomplished by the means of partly sharing the model between the tasks, as proposed in (Caruana, 1993). Specifically, this involves sharing the hidden layers between all tasks, while keeping the output layer (or few layers preceding it) specific for each task. While training the model, we use a different loss function for each task - $\mathcal{L}_a\big(f_a(g(\mathbf{x};\theta_\mathrm{g});\theta_\mathrm{fa}),\mathbf{y}\big)$, and $\mathcal{L}_b\big(f_b(g(\mathbf{x};\theta_\mathrm{g});\theta_\mathrm{fb}),\mathbf{y}\big)$ where f_a, f_b are task specific layers, and g is the shared layer of the model. We can then modify the objective function (Eqn. 2.6) as follows:

$$\begin{aligned}\theta^* = \underset{\theta}{\operatorname{argmin}}\ & \lambda_a \mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_\mathrm{a}} \mathcal{L}\big(f_a(g(\boldsymbol{x};\theta_\mathrm{g});\theta_\mathrm{fa}),\boldsymbol{y}\big) \\ & + \lambda_b \mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_\mathrm{b}} \mathcal{L}\big(f_b(g(\boldsymbol{x};\theta_\mathrm{g});\theta_\mathrm{fb}),\boldsymbol{y}\big) \end{aligned} \tag{2.35}$$

where λ_a, λ_b are hyperparameters.

Training the models to perform more than one tasks simultaneously incentivises (biases) it to prefer representations which captures knowledge useful for more than one task, imparting it with a so called *representation bias*, and correspondingly reducing its tendency to overfit on one distribution. This strategy thus can be better for the performance over both the source and target tasks, as opposed to training independent models to perform them separately. Further, MTL increases the amount of samples the model can train of, which in cases of slight domain and task shift, can be beneficial in and of itself. Moreover, given that all tasks (or rather, samples from all tasks) contain some degree of noise, generated by different noise patterns, MTL can bias the model to learn better representations of the input by abstracting out the noise. Thus, through averaging noise patterns, increased training data, and representation bias MTL can increase performance across multiple tasks.

Interestingly, we can leverage the aforementioned benefits even in scenarios when we don't have more than one tasks we wish the model to excel on (non-transfer learning scenarios). Treating the main task as our source task, we can choose a related **auxiliary task** as a target task in a manner which increases the model performance on source. Choosing an appropriate

auxiliary task is of paramount importance, and has invited numerous investigations over the years (Caruana, 1997; Baxter, 2000; Ben-David and Schuller, 2003; Plank and Alonso, 2017). Generally, auxiliary tasks include predicting underlying statistical features of inputs, unsupervised outputs (such as reconstructing the input sequence (Rei, 2017)), and more commonly performing a related supervised task such as predicting low level characteristics of the language (parts of speech tagging, named entity recognition) (Niehues and Cho, 2017) etc.

For the purposes of our work, we are interested in using supervised *adversarial tasks* as our auxiliary task. An adversarial task is any auxiliary task which accomplishes the opposite of what we want as our main task. That is to say that minimising the loss for adversarial tasks would result (directly or otherwise) in the maximisation of the loss on main one. These tasks can be used to reduce certain biases from the model, explicitly ignore certain features of the data (Gong et al., 2018), or generally regularise the model. In Chapter 4, we discuss one such adversarial task which we hypothesise can be beneficial for our purposes.

2.4.2 Sequential Transfer Learning

A much more common scheme of trying to solve source and target tasks is to train the model one after the other, or sequentially over the different tasks. The goal in these cases is to effectively transfer the (task specific) understanding of the domain gained while training on the source task, to improve performance on the target. Typically, we refer to the first phase, or when a model is trained from scratch to perform a task as the **pretraining** phase, and when using it subsequently to solve another task as the **finetuning** (or adaption) phase. Thus, given a task from a low resource domain, or with inadequate number of labeled samples, we can choose a task pretraining on which can impart an understanding of the target domain beneficial for the latter. Based on the sources of supervision, a pretraining task can be categorised as (i) supervised pretraining, (ii) unsupervised pretraining or (iii) pretraining based on distant supervision.

In this work, we are interested in unsupervised pretraining to improve target task performance. While pretraining a model to perform a specific task can be helpful in cases with minimal task shift, these models learn to only focus on aspects of the data which are pivotal for solving the task at hand. On the contrary, unsupervised pretraining can be more general, increasing the number of potential downstream tasks which can benefit from it. Further, unsupervised data is much more easily available even in low resource languages. Given this, unsupervised pretraining is an appealing

method of increasing model performance, and has been extensively studied both in the context of neural and non-neural machine learning approaches.

In NLP, primarily two forms of pretraining are predominant namely, matrix factorisation to factorise a word-word co-occurrence matrix (Søgaard et al., 2017), and language modeling. As discussed in Sec. 2.3, using neural language models can capture complex linguistic phenomenons which can be beneficial for a wide variety of NLP tasks. We discuss this in greater detail in the next chapter, Chapter 3.

2.5 Conclusion

In this chapter, we outlined numerous concepts needed for a comprehensive understanding of our work. We began the discussion with introducing the field of Natural Language Processing (NLP) (Sec. 2.1), a subset of deep learning foundations including the concepts of MLE, loss, and gradient descent (Sec. 2.2). We also elaborated some of the pertinent layers constituting deep learning models, and the architectures in which they are frequently combined to solve NLP problems (Sec. 2.2.2, 2.2.3). Further, we introduced the task of language modeling, and provided an overview of approaches which can solve the task (Sec. 2.3), and finally provided a brief outline of inductive transfer learning approaches including sequential transfer learning, and multi-task learning (Sec. 2.4).

Chapter 3

Using Language Models for NLP Tasks

After discussing the concepts of NLP tasks (Sec. 2.1), language models (Sec. 2.3) and transfer learning (Sec. 2.4), we now discuss the context surrounding our work, namely *"the use of pretrained (recurrent) language models for downstream tasks"*.

3.1 Motivation

Recurrent Neural Language models (word embeddings, or entire encoders), as discussed previously are neural networks which are capable of generating text akin to the one they're trained on. Their flexibility in considering k-gram or skip-gram words when needed, along with their generalisation capabilities on unseen combinations of words in the context lead us to conclude that they're reasonably capable of learning the idiosyncrasies of the domain (and given language), and modeling $p_{\text{data}}(X)$ for a given domain $\mathcal{D}$. Given that they can be trained in an unsupervised fashion, i.e. on unlabeled datasets, they can capture more general aspects of the structure and semantics of the language, as opposed to training language models on labeled data as an auxiliary task wherein they're incentivized to capture the meaning necessary for a given task and discard everything else (Ruder, 2019, Ch. 3.3.2.3).

Further, recall (Sec. 2.3.3) that neural language models are typically actualized as an *encoder-generator* network where the *encoder* learns a mapping from the input feature space $\mathcal{X} \subseteq \mathbb{W}^{|V| \times L}$ to a latent feature space $\mathcal{X}_{\text{enc}}$, ideally comprised of explicit information about the input word sequence, which can be readily used by the *generator* to model $p_{\text{data}}(X)$. The role of encoder thus is similar in neural architectures commonly employed to solve NLP tasks

(See Sec. 2.2.3), and it would be desirable to incorporate the richer language model's encoder for these tasks, as a pretrained sub-network.

Pretrained encoders in these task-specific models could potentially impart the model with a better understanding of the underlying text positively affecting its performance. Also, providing the model with coherent language understanding from the very start of the training process could help the model in faster convergence, consequentially enabling the use of neural approaches in low resource scenarios (with limited availability of labeled data).

3.2 Using Word Embeddings

The aforementioned form of using pretrained models, has long been the norm in neural approaches for NLP, in the form of pretrained word embeddings. In this section, we outline their use and the effects they've been shown to have on model performance.

Word embeddings[1] (Mikolov et al., 2013; Pennington et al., 2014; Levy and Goldberg, 2014; Bojanowski et al., 2017) as defined in Sec. 2.3.2 are word level affine transformations, typically constituting the lowest layer in a neural network for NLP tasks. They embed input words in a low-dimensional distributed (often inter-related) feature space representative of the semantic structure of the word (and its context), typically, such that similar words have similar vectors (Goldberg, 2016).

Using pretrained word embeddings has shown to be beneficial for numerous tasks. Some of these include part-of-speech tagging (Collobert et al., 2011), dependency parsing (Chen and Manning, 2014; Kong et al., 2014), sentence classification (Kim, 2014), and even machine translation (Liu et al., 2014; Kalchbrenner and Blunsom, 2013; Devlin et al., 2014). Owing to their ease of use, and tangible performance gains, the use of word embeddings "has had a large impact in practice and is used in most state-of-the-art models." (Howard and Ruder, 2018)

The way to use these embeddings in a task specific model is rather straight-forward. Weight matrices from the pretrained embedding model (e.g. word2vec[2], GloVe[3], fastText[4] etc) are used to initialize the first (lowest) layer of the network, the rest of which is unchanged and trained on the labeled dataset at hand. In doing so, the embedding layer might be kept fixed or trainable. We illustrate this in Fig. 3.1.

[1] Commonly referred to as feature embeddings, word vectors or embedding matrices.

[2] `https://code.google.com/archive/p/word2vec/`

[3] `https://nlp.stanford.edu/projects/glove/`

[4] `https://fasttext.cc/`

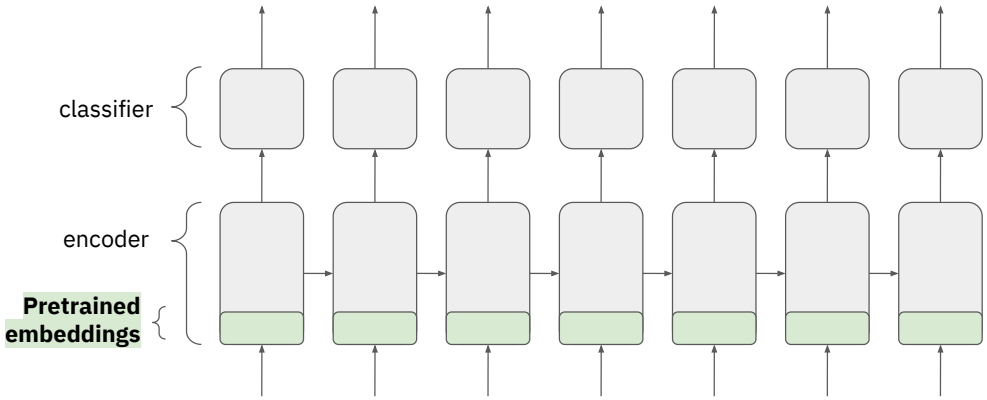


Figure 3.1: A typical neural model (for NLP tasks), with the green highlights indicating the pretrained embeddings.

While beneficial, this approach of leveraging unlabeled data by pretraining is insufficient, since the rest of the encoder (recurrent, convolutional layers etc) remains untrained. As a result, while the model gains a better semantic and lexical understanding of the input word tokens, it needs to learn to derive meaning from a sequence from scratch, thereby serving little assistance in modeling compositionality, anaphora, negation etc. (Ruder, 2018) makes the followings statement on the matter:

> Using word embeddings is like initializing a computer vision model with pretrained representations that only encode edges: they will be helpful for many tasks, but they fail to capture higher-level information that might be even more useful.

3.3 Using Recurrent Language Models

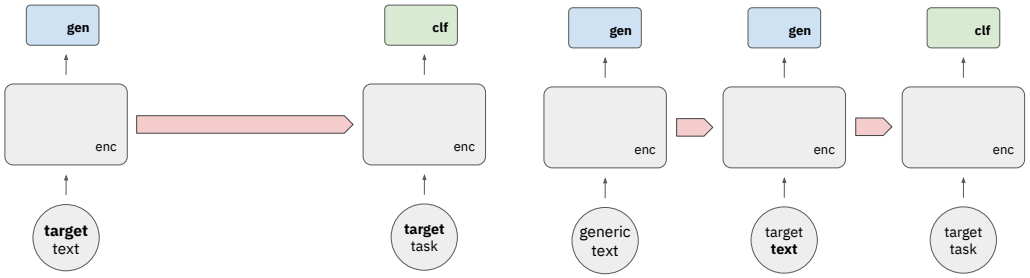
Given the benefits of pretraining (Sec. 3.1), and the limitations of using pretrained embeddings (Sec. 3.2), it is desirable to be able to replace the entire encoder of a task specific network, with a language model's (LM). In the simplest setting, this would be akin to using pretrained embeddings wherein, granted that the task specific model and the language model have similar encoders, we could simply initialize the former with the already trained parameters of the latter. Doing so would induce the yet untrained (task specific) model with the ability to not only get a rudimentary lexical and semantic understanding of the input word tokens, but an entire set of hierarchical representations (features) that the LM has learned.

This line of reasoning is confirmed by (Dai and Le, 2015) who pretrain a language model on the data from the task's domain, and use its encoder as an initialization for the task specific model. We visualize their transfer mechanism in Fig 3.2a. They find that the process decreases the error rate by $\sim$ 43 % (13.50% to 7.64%) for IMDB Review sentiment classification (Maas et al., 2011) (which we experiment with, in Sec. 5.4), which is significantly more when compared to only using pretrained embeddings: $\sim$ 25% (13.50% to 10.00%). Their findings confirm that the *task shift* (Sec. 2.4) between language modeling ($\mathcal{T}_a$) and sentiment classification ($\mathcal{T}_b$) can be reasonably bridged, in this setting.

However, in their experiments, their language model is trained on text from within the same domain. We refer to this form of pretraining as **task-specific** pretraining. This negatively affects the appeal of their work since this approach might not retain the performance improvement in low resource scenarios where there is a limited availability of labeled data. Ideally, we would want to leverage text from general domains by the means of pretraining in a manner which enables faster convergence and/or overall improvement. We refer to this form of pretraining as **task-invariant** pretraining.

Therein lies a problem, namely that if the *domain shift* (Sec. 2.4) between the domains $\mathcal{D}_b$ of the task at hand, and $\mathcal{D}_a$ on which the LM was trained, is substantial, some features would be not representative, when computed over the task's data. The feature spaces (outputs of different layers of a network) have shown to be increasingly *less general* or *more specific* from bottom (input) to top (output) layers (Yosinski et al., 2014), wherein this generality is representative of that layer's ability to perform satisfactorily across data from different domains. This is why pretrained embeddings are easier to use than entire encoders (Mou et al., 2016). The work of (Dai and Le, 2015) did not suffer from the problem as their pretrained language model was trained on the same data, thereby incurring no domain shift.

In this light, it is imperative to actively bridge the domain shift to retain pretrained benefits while fine tuning the model for the target task. Fine tuning, here refers to training the model with pretrained encoder on task data $\{x_1, y_1; x_n, y_n | x_i \in X_b, y_i \in \mathcal{Y}_b\}$. Mou et al. (2016) demonstrate that recurrent networks are ineffective in bridging both, the domain shift and the task shift simultaneously. Note however that in their experiments, they didn't attempt to transfer between language modeling and sentiment classification tasks, as is our case. Nevertheless, their findings hint at a need for complex, multi-part approaches for satisfactory performance improvement while pretraining with *task-invariant* language models. The difficulty is further compounded upon by the fact that during fine-tuning, the model might forgo language understanding in favor of overfitting on the given training



(a) Two Phased Transfer Scheme (b) Three Phased Transfer Scheme

Figure 3.2: An illustration of pretraining schemes of (a) Dai and Le (2015) and (b) Howard and Ruder (2018). In (a), the language model is pretrained on text from the target domain. Its encoder is then used to initialize the parameters of the task solver network. In (b), the language model is first pretrained on text from general domain, and then fine tuned again with text from the target task before being used in the task solver network. In both cases, we use (■) blue color to denote generators (for language modeling), and (■) green for classifiers (for the target task).

data, effectively negating the benefits of pretraining, and in some cases being adversely affected by it. This phenomenon is referred in literature as *catastrophic forgetting* (McCloskey and Cohen, 1989; Kemker et al., 2018), and has been actively studied, with no single solution [5] which works across tasks, and domains. Further, in our setting, the classifier (top-most layers of the task specific model) would be completely untrained, requiring it to be trained more than the encoder. However, given higher loss at the start of the training (fine-tuning) process and no means to disincentivise the encoder from changing its parameters, the model stands an increased chance of catastrophic forgetting. This particular effect is partly countered by freezing schedules which inhibit certain parts of the model from training (hence the word "freezing") during certain parts of the training process (Felbo et al., 2017).

To tackle the aforementioned challenges, Howard and Ruder (2018) propose a hybrid technique involving both *task-invariant* and *task-dependent* pretraining which shows noticeable improvements for a variety of NLP tasks

[5] Elastic Weight Consolidation (Kirkpatrick et al., 2017) was shown to solve catastrophic forgetting in supervised learning (over MNIST (LeCun et al., 2010)) and reinforcement learning settings (over Atari games (Bellemare et al., 2013)). (Kemker et al., 2018) instead "demonstrate that despite popular claims [Kirkpatrick et al. (2017)], catastrophic forgetting is not solved."

including sentiment (Maas et al., 2011), question (Voorhees and Tice, 1999), and topic (Zhang et al., 2015) classification. We outline their three phased learning scheme following the use-case for sentiment classification[6]. Here we use $\mathcal{T}_a$ to denote the language modeling task over task invariant text - Wikitext103 (Merity et al., 2017) ($\mathcal{D}_a$), and $\mathcal{T}_b$ to denote the sentiment classification task over IMDB Review Dataset (Maas et al., 2011) ($\mathcal{D}_b$):

1. *Phase 1:* Pretrain an three layered AWD-LSTM (Merity et al., 2018) based language model ($\mathcal{T}_a$) over text from $\mathcal{D}_a$.

2. *Phase 2:* Train another similar language model ($\mathcal{T}_{a^*}$), with using the encoder from Phase 1 over text from $\mathcal{D}_b$, i.e. the IMDB Review dataset. In this manner, the encoder could familiarize itself with target task's text, bridging the domain shift.

3. *Phase 3:* Use the encoder from Phase 2 to initialize a task specific text classification model following an encoder-classifier architecture (Sec. 2.2.3), and train it over $\mathcal{T}_b$.

We illustrate this using Fig. 3.2b. Apart from improved performance over tasks with ample training data, their approach benefits from comparable performance in very low resource domains (100 labeled examples) w.r.t. training from scratch with 1000 labeled examples (randomly sampled subsets of IMDB Review Classification task). That said, their approach relies on a slew of regularization and training optimization tricks including learning rate schedules, layer freezing schedules, batch normalization (Ioffe and Szegedy, 2015), embedding and variational dropout , variable length backpropagation sequences (Merity et al., 2018), some of which are novel, developed alongside their three phased scheme, and have substantial contributions in their approach's improvements.

3.4 Using Transformers based Language Models

Transformers (Vaswani et al., 2017), are non-recurrent (state-less) sequence transduction models which use multi-head self-attention to condition its outputs across time steps. Very recently (Radford et al., 2018; Devlin et al., 2018; Radford et al., 2019), it has been shown that transformers can be pretrained and used in a setting similar to the one of our interest, namely - unsupervised, task-invariant pretraining, and supervised fine tuning to achieve

[6]Although the approach is unchanged for any given task.

state-of-the-art results over a variety of NLP tasks. While not the focus of our work, the use of transformers in this setting is nonetheless very interesting. We thus discuss the aforementioned approaches and outline their findings in this section.

OpenAI GPT (Radford et al., 2018) use the transformer-decoder (Liu et al., 2018) model which uses position-wise feed-forward layers over encoded input tokens to compute the familiar marginal likelihood for language modeling (Eqn. 2.28), and train it over the BooksCorpus dataset (Zhu et al., 2015), consisting of "long stretches of contiguous text which allows the generative model to learn to condition on long range information" (Radford et al., 2018). Post pretraining, they use a new feed-forward layer to compute the conditional distribution over the target task's label space the transformer's final activation, along-with recreating the input sequence, by the means of the aforementioned language modeling setup. This technique, has also been explored in different contexts and is shown to lead to general performance improvements (Peters et al., 2017; Rei, 2017). Through their experiments, performed over multiple NLP tasks including natural language inferencing (Bowman et al., 2015), semantic similarity (Dolan and Brockett, 2005) and text classification (Warstadt et al., 2018), they find that their approach outperforms existing ones on 9 out of 12 tasks. They attribute this, primarily to the ability of their model to generalize over long sequences, capturing long range dependency effectively, something which recurrent models have shown to have problems with (Bengio et al., 1994b).

Devlin et al. (2018) improve upon their work, primarily by introducing a bidirectionality in their language encoding scheme. Previously, transformers encoded information in a classical left-to-right fashion enforced by positional embeddings, and attention masks over decoder's self attention layers (since transformers, by design are sequence agnostic). To do so, they re-purpose masked language modeling technique (Taylor, 1953) to pretrain their language model wherein random words from the input sequence as replaced with `[MASK]` tokens, and training the model to predict the masked words. They also include a textual entailment aspect to their pretraining, occasionally replacing consecutive sentences in the input corpus with a random sentence, and training another classifier alongside which is tasked with identifying this swap. Through their experiments, over a larger variety of NLP tasks, they find that their model converges relatively slower when compared to Radford et al. (2018), after which it outperforms them more often than not. This is attributed by the authors primarily to the bidirectionality of the model, rather than the auxiliary entailment based pretraining. These advances have significantly impacted the NLP research area, inviting numerous further applications and investigations (Radford et al., 2019; Alberti et al.,

2019; Stickland and Murray, 2019).

3.5 Conclusion

In this chapter we outline the existing literature regarding the use of language models, trained in a *task-invariant* fashion, for the benefits of downstream NLP tasks. We discuss potential pitfalls caused by *domain shift* and *task shift* between the language modeling task and the target task. We further discussed the ways in which these pitfalls have been (or can be circumvented), by primarily focusing on the works of Dai and Le (2015); Howard and Ruder (2018), whose approaches we summarise in Fig. 3.2.

In the next chapter, we discuss domain adversarial training - a technique which we hypothesize can help overcoming the domain shift, and further improving the use of pretrained models in our setting.

Chapter 4

Domain Adversarial Training of Neural Networks

As mentioned in Sec. 1, we aim to explore whether domain adversarial training can benefit the use of pretrained language models for downstream task (Chapter 3). In this chapter, we introduce the concept of domain adversarial training, discuss the theoretical foundations behind the concept, and illustrate the manners in which we can use it for our potential benefit.

4.1 Motivation

In Chapter 3, we discussed the potential pitfalls that a measurable domain shift can bring during fine-tuning a pretrained model. Here, domain shift can be thought of as a notion of difference in marginal distributions between the source and target domains ($p_{\text{data}}(X_b)$ and $p_{\text{data}}(X_a)$). A related concept is that of **domain invariance**, which refers to a model's ability to encode inputs from different domains, in such a manner that given representations corresponding to the input, it is impossible (or *more* difficult) to deduce its corresponding domain. In this work, the idea is of interest to us, due to the following reasoning - "if we can induce some degree of domain invariance to the encoder (during pretraining or finetuning), we effectively reduce the domain shift between the two phases. If accomplished without losing information needed to make predictions for the target task, this may lead to a potential performance improvement." We thus focus on means of inducing domain invariance in our models, in the rest of this chapter.

4.2 Overview

Domain adversarial training, proposed in (Ganin et al., 2016) is an auxiliary task, intended to induce *domain invariance* in a model during the training process. Broadly put, the idea is to augment a given neural network with a domain classifier in a multi-task fashion (see Sec. 2.4.1), tasked with performing the following auxiliary task: predicting a given input's domain using its encoded representations. During training, the domain classifier learns to correctly predict the domain, while the encoder, adversarially, learns to obscure that information (simultaneously, while training to perform well on the task). This is accomplished by the means of flipping the polarity of the gradients during backpropagation between the domain classifier and the encoder.

Formally, let $\mathcal{T}_a$ be a task over domain $\mathcal{D}_a$, and $\mathcal{T}_b$ another task over domain $\mathcal{D}_b$. For simplicity's sake, we assume that $\mathcal{T}_a$ and $\mathcal{T}_b$ have the same label space[1] $\mathcal{Y}$ (e.g. sentiment classification over movie and restaurant reviews). In this scenario, also referred to as supervised domain adaption, we have samples (x_i^a, y_i^a) drawn i.i.d. from task $\mathcal{T}_a$ i.e. $x_i^a \sim X_a,\ y_i^a \sim \mathcal{Y}$, and similarly (x_i^b, y_i^b) drawn from task $\mathcal{T}_b$. Further, given an encoder-classifier network (Sec. 2.2.3) characterised by the following equations:

$$h = f_{enc}(x; \theta_{enc}) \tag{4.1}$$

$$\hat{y} = \text{softmax}(f_{clf}(h; \theta_{clf})) \tag{4.2}$$

which is trained using an arbitrary mini-batch based algorithm (for eg. stochastic gradient descent), with mini-batches sampled from both (x_i^a, y_i^a) and (x_i^b, y_i^b). Using an arbitrary loss function $\mathcal{L}_{task}$, we can use the following objective function to train the model:

$$\theta^* = \underset{\theta}{\text{argmin}}\ \mathbb{E}_{\mathbf{x},\mathbf{y}\sim\{\hat{p}^a_{\text{data}},\hat{p}^b_{\text{data}}\}}\mathcal{L}_{task}(\hat{\boldsymbol{y}}, \boldsymbol{y}) \tag{4.3}$$

Domain adversarial training in this generic scenario involves solving an auxiliary task $\mathcal{T}_{a,b}$ whose domain consists of samples from $\mathcal{T}_a$ and $\mathcal{T}_b$, which is to say $X_{a,b} = X_a \cup X_b$. The label space $\mathcal{Y}_{a,b}$ is a set $\{0, 1\}$ where 1 indicates that a given input is sampled from X_a, and 0 from X_b. We can then use the following labeling function to create a label set $Y_{a,b}$: I_{X_a}, where $I : X_{a,b} \mapsto \{0, 1\}$ is an indicator function[2]. During the (domain adversarial) training of our model, we would want the model to predict labels for this task,

[1] We will demonstrate, towards the end of Sec. 4.3.4, the trivial modifications we make to the setup below, to enable domain adversarial training in a multitask setting (e.g. sentiment classification of movie reviews, and topic classification of factual questions).

[2] `https://en.wikipedia.org/wiki/Indicator_function`

along with $\mathcal{T}_a$ and $\mathcal{T}_b$. To do so, we augment our encoder-classifier network with an auxiliary classifier f_{dom} called the **domain regressor**, parameterised by θ_{dom}. This augmented network is represented by the following equations:

$$h = f_{enc}(x; \theta_{enc}) \tag{4.4}$$

$$\hat{y}^{task} = \text{softmax}(f_{clf}(h; \theta_{clf})) \tag{4.5}$$

$$\hat{y}^{dom} = \text{softmax}(f_{dom}(h; \theta_{dom})) \tag{4.6}$$

Let $\mathcal{L}_{dom}$ be another loss function which computes the model's error for the domain classification task. Now, if we add this loss term to our objective function akin to Eqn. 2.35, during training, the model would strive to improve at both - the main and the auxiliary task. That is, it would learn to predict the label for the target task, as well as find which domain does the input sample belong to. We, however aim to do quite the opposite - train the model to predict the target task's label while not being able to predict the domain from which the input was sampled. Thus, our modified objective function is:

$$\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\{\hat{p}^a_{\text{data}},\hat{p}^a_{\text{data}}\}} \left[\mathcal{L}_{task}(\hat{\boldsymbol{y}}^{task}, \boldsymbol{y}) - \lambda \mathcal{L}_{dom}(\hat{\boldsymbol{y}}^{dom}, I_{X_a}(\boldsymbol{x}))\right] \tag{4.7}$$

where λ is a hyper-parameter which is used to tune the trade-off between these two quantities during the training process.

Correspondingly, the parameter update step under this objective function would involve the following changes:

$$\theta_{enc} \leftarrow \theta_{enc} - \eta(\nabla_{\theta_{enc}} \mathcal{L}_{task} - \lambda \nabla_{\theta_{enc}} \mathcal{L}_{dom}) \tag{4.8}$$

$$\theta_{clf} \leftarrow \theta_{clf} - \eta \nabla_{\theta_{clf}} \mathcal{L}_{task} \tag{4.9}$$

$$\theta_{dom} \leftarrow \theta_{dom} - \eta \lambda \nabla_{\theta_{dom}} \mathcal{L}_{dom} \tag{4.10}$$

We can then use the augmented (i) model (Eqn. 4.4, 4.5, 4.6), (ii) tasks $(\mathcal{T}_a, \mathcal{T}_b, \mathcal{T}_{a,b})$, (iii) objective function (Eqn: 4.7) and (iv) the aforementioned update steps together to create a training schedule which collectively constitute the **domain adversarial training** of neural networks. An illustration of the training schedule is provided in Fig. 4.1.

4.3 Theoretical Basis for Domain Adversarial Training

The aforementioned scheme of training justified so far by an intuition (Sec. 4.1) has roots in a seminal work on domain adaption (Ben-David et al., 2006,

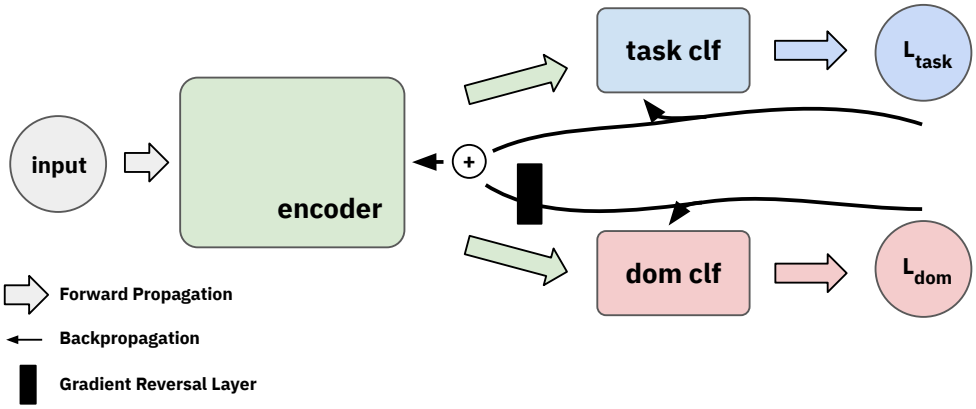


Figure 4.1: An abstract representation of the domain adversarial augmentations to generic neural architectures . Here the (■) green colored encoder, and the (■) blue colored task classifier together comprise a regular deep learning model. The (■) red colored domain classifier is augmented to the model. During the parameter update step, the (■) black colored gradient reversal layers flips the polarity of the gradients, modeling the update step as depicted by Equations (4.8) to (4.10).

2010), the findings of which were adapted to constitute a neural training algorithm by (Ganin et al., 2016), which we slightly modify for our purposes. incorporate in our context (Chapter 3). In this section we primarily discuss their work, and outline the manner in which their findings give a theoretical basis to the objective function (Eqn. 4.7)

4.3.1 Preliminaries

Consider a unsupervised domain adaption scenario wherein the source task $\mathcal{T}_a$ has labeled samples but the target task $\mathcal{T}_b$ does not. That is to say that we have been provided with n samples $\{(\boldsymbol{x}_i^a, y_i^a)\}_{i=1}^n$ where $\boldsymbol{x}_i^a \sim p_{\text{data}}(X_a)$,and $y_i^a \sim p_{\text{data}}(Y|X = \boldsymbol{x}_i^a)$; and $n^{'}$ samples $\{(\boldsymbol{x}_i^b)\}_{i=1}^{n^{'}}$ where $\boldsymbol{x}_i^b \sim p_{\text{data}}(X_b)$. Since we work in domain adaption scenario, the feature space $\mathcal{X}_{a,b}$ and label space of the two tasks $\mathcal{Y}_{a,b}$ should be the same. This is not to suggest that the marginal distribution $p_{\text{data}}(X)$ or the conditional distribution $p_{\text{data}}(Y|X)$ should be identical. We follow this scenario in the rest of this section unless specified otherwise.

Additionally, we shall require the following definitions for our purposes:

Definition 4.1. Hypothesis: A hypothesis h is a function which maps elements from the feature space to the label space i.e. $h : \mathcal{X} \mapsto \mathcal{Y}$.

This generic definition includes a wide variety of machine learning models including a linear classifier, logistic regressor and even complex transformers based classifiers. In the context of neural networks, we can see them simply as an alternate representation of a model $f(;\theta_i)$ with a fixed set of parameters θ_i.

Definition 4.2. Hypothesis Class: A hypothesis class $\mathcal{H}$ is any set of hypothesis h. We constrain $\mathcal{H}$ to only include hypotheses having the same Vapnik–Chervonenkis (VC) dimension (Vapnik and Chervonenkis, 1971).

In the context of neural networks, $\mathcal{H}$ can be seen as a model architecture $f(;\theta)$ where θ may take any value in the parameter space.

Definition 4.3. Symmetric Hypothesis Class: As mentioned in Ganin et al. (2016), a given hypothesis class $\mathcal{H}$ for an arbitrarily sized discrete label space $\mathcal{Y}$ is symmetric if for all $h \in \mathcal{H}$, and for any permutation of labels $c : \mathcal{Y} \mapsto \mathcal{Y}$, we have $c(h) \in \mathcal{H}$.

Under this definition, most neural architectures are a symmetric hypothesis class. "[Symmetric hypothesis classes] are useful, since they are a natural choice when there is no prior knowledge about the relations between the possible labels." (Daniely et al., 2015). While not pivotal to the discussion below, this concept is used in Lemma 4.3.1 and the generalisation of Definition 4.7.

Definition 4.4. Error of a hypothesis: We define a notion of *error* of a hypothesis as a probability that the label assigned to an input by a hypothesis h disagrees with the label assigned to it within the dataset.

$$\epsilon(h, p_x, p_{y/x}) = \mathbb{E}_{\mathbf{x}\sim p_x, y\sim p_{y/x}(X=\mathbf{x})} I[h(\boldsymbol{x}) \neq y] \tag{4.11}$$

Definition 4.4.1 Empirical Error: We can easily extend this notion to compute error based on labeled samples simply as:

$$\hat{\epsilon}(h, (x_i)_{i=1}^n, (y_i)_{i=1}^n) = \frac{1}{n}\sum_{i=1}^{n} I[h(x_i) \neq y_i] \tag{4.12}$$

We can thus represent the error of a given hypothesis on task $\mathcal{T}_a$ with $\epsilon(h, p_{\text{data}}(X_a), p_{\text{data}}(Y_a|X_a))$, or simply as $\epsilon_{\mathcal{T}_a}(h)$ for notational simplicity. Likewise, we can represent the empirical error by $\hat{\epsilon}(h, X_a, Y_a)$ or $\hat{\epsilon}_{\mathcal{T}_a}(h)$

Definition 4.5. Ideal Joint Hypothesis: (As mentioned in (Ben-David et al., 2010, Def. 2),) Given a hypothesis class $\mathcal{H}$, and tasks $\mathcal{T}_a$, and $\mathcal{T}_b$, an

ideal joint hypothesis h is the hypothesis from the class which minimises the combined error:

$$h^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \ \epsilon_{\mathcal{T}_a}(h) + \epsilon_{\mathcal{T}_b}(h) \tag{4.13}$$

We denote the combined error of the ideal hypothesis by

$$\gamma = \epsilon_{\mathcal{T}_a}(h^*) + \epsilon_{\mathcal{T}_b}(h^*) \tag{4.14}$$

In practice, we expect γ to be rather small for most domain adaption scenario, signifying that it is *possible* to find a hypothesis that works well across the two tasks. An example where this assumption wouldn't hold would be the case when $\mathcal{T}_b$ is adversarial w.r.t. to $\mathcal{T}_a$, which is valid to some extent in Generative Adversarial Networks (Goodfellow et al., 2014).

4.3.2 Domain Divergence

Central to the work of Ben-David et al. (2006, 2010) is a notion of *domain divergence* in the context of a given hypothesis or hypothesis class. In essence, domain divergence quantifies our intuitive idea of *domain shift* (See Sec. 2.4). There are a multitude of ways to estimate the difference between two domains including L^1 divergence (variational divergence) or Kullback-Leibler divergence (Kullback and Leibler, 1951).

Definition 4.6. L^1 **Divergence**: L^1 divergence or variational divergence between two domains $\mathcal{D}_a, \mathcal{D}_b$ is a notion of distance between their constituent marginal distributions $p_{\text{data}}(X)$ defined as:

$$d_1(\mathcal{D}_a, \mathcal{D}_b) = 2 \sup_{B \in \mathcal{B}} |p_{\text{data}}(X_a = B) - p_{\text{data}}(X_b = B)| \tag{4.15}$$

where $\mathcal{B}$ is the set of measurable subsets under the feature spaces $\mathcal{X}_a, \mathcal{X}_b$

The use of L^1 estimate, in our context is discouraged primarily owing to the following two reasons:

- Given that we work with a finite samples from an unknown distribution p_{data}, finding $\mathcal{B}$ – all measurable subsets from the feature space hinders our ability to accurately estimate the L^1 divergence between $\mathcal{D}_a$ and $\mathcal{D}_b$ (Batu et al., 2000; Kifer et al., 2004).
- For our purposes, which we discuss in the next section, it is disadvantageous to use a metric as strict as L^1 since it may include subsets $B \in \mathcal{B}$ (while computing the supremum) which our hypothesis class might not be able to solve anyway.

Ben-David et al. (2006, 2010) thus use new divergence measure, based on the earlier work of (Kifer et al., 2004). We now mention their definition, for the purposes of binary classification tasks i.e. $\mathcal{Y} = \{0, 1\}$[3].

Definition 4.7. $\mathcal{H}$-Divergence: (As mentioned in Ganin et al. (2016))[4], Given domains $\mathcal{D}_a$, $\mathcal{D}_b$ over a feature space $\mathcal{X}_{a,b}$ characterized by marginal distribution $P(X_a)$ and $P(X_b)$ respectively, and $\mathcal{H}$ be a hypothesis class, the $\mathcal{H}$-Divergence between $\mathcal{D}_a, \mathcal{D}_b$ can be defined as:

$$d_{\mathcal{H}}(\mathcal{D}_a, \mathcal{D}_b) = 2 \sup_{h \in \mathcal{H}} \Big| \Pr_{\mathbf{x} \sim p_{\text{data}}(X_a)} [h(\boldsymbol{x}) = 1] - \Pr_{\mathbf{x} \sim p_{\text{data}}(X_b)} [h(\boldsymbol{x}) = 1] \Big| \quad (4.16)$$

This is to say that $\mathcal{H}$-divergence between two domains, for a hypothesis class $\mathcal{H}$ depends on ability of hypotheses $h \in \mathcal{H}$ to infer whether an example was sampled from the first or the second distribution. This measure thus overcomes the two limitations mentioned above. Recall however that we do not have access to the real p_{data}, but only to a set of samples generated from it, characterized by $\hat{p}_{\text{data}}$. We thus need an empirically measurable variant of $\mathcal{H}$-Divergence.

Lemma 4.3.1. (Ben-David et al., 2010, Lemma 2) Consider a symmetric hypothesis class $\mathcal{H}$ (i.e. for every $h \in \mathcal{H}$, the inverse hypothesis $1 - h$ is also in $\mathcal{H}$). Further, consider a task $\mathcal{T}_{a,b}$ (as defined in Sec. 4.2) whose domain consists of m samples X_a, X_b from $\mathcal{D}_a$ and $\mathcal{D}_b$ respectively; and the labels corresponding to each sample are characterized by $I[\mathbf{x} \in X_a] \; \forall \; \mathbf{x} \; \in X_a \cup X_b$. The **empirical $\mathcal{H}$-divergence** can then be defined as follows:

$$\hat{d}_{\mathcal{H}}(X_a, X_b) = 2\Big(1 - \min_{h \in \mathcal{H}} \Big[\frac{1}{m} \sum_{\mathbf{x}: h(\mathbf{x})=0} I[\mathbf{x} \in X_a] + \frac{1}{m} \sum_{\mathbf{x}: h(\mathbf{x})=1} I[\mathbf{x} \in X_b] \Big] \Big) \quad (4.17)$$

where $I[\mathbf{x} \in X_a]$ is the binary indicator variable which is 1 when $x \in X_a$ and 0 otherwise.

We now have a divergence measure that we can compute given finite samples and a hypothesis class. The following lemma can then be used alongside Lemma 4.3.1 as a substitute to $\mathcal{H}$-Divergence (which we can not compute).

Lemma 4.3.2. (Ben-David et al., 2010, Lemma 1) Consider a hypothesis class $\mathcal{H}$ with VC Dimension (Vapnik and Chervonenkis, 1971) d, and m

[3]Note that as mentioned by Ben-David et al. (2006), and further repeated in (Ganin et al., 2016), the following can be easily generalised to a multi-class setting, given that $\mathcal{H}$, the hypothesis class is *symmetrical* (Def. 4.3).

[4]We appropriate their notations to maintain consistency with ours.

samples X_a, X_b from $\mathcal{D}_a$ and $\mathcal{D}_b$ respectively. Let $\hat{d}_{\mathcal{H}}(X_a, X_b)$ be the empirical $\mathcal{H}$-Divergence between the samples. Then, for any $\delta \in (0, 1)$ with probability at least $1 - \delta$,

$$d_{\mathcal{H}}(\mathcal{D}_a, \mathcal{D}_b) \leqslant \hat{d}_{\mathcal{H}}(X_a, X_b) + 4\sqrt{\frac{d\ \log(2m) + \log(\frac{2}{\delta})}{m}} \tag{4.18}$$

This lemma is based on (Kifer et al., 2004, Theorem 3.4), upon which Ben-David et al. (2010) made slight modifications.

Note that in Eqn. 4.17, $h(x_i) = 0$ implies that according to the hypothesis h, $x_i \in X_b$, and that $I[x_i \in X_a] = 1$ if $x_i \in X_a$. Thus, the term being minimised is simply error of a given hypothesis while classifying the domains from which a given input was sampled - $e_{\mathcal{T}_{a,b}}(h)$ where $\mathcal{T}_{a,b}$ is as defined in Sec. 4.2. Note also that we intend to find the hypothesis with minimum $e_{\mathcal{T}_{a,b}}(h)$ amongst all possible hypothesis in $\mathcal{H}$. For any slightly complex feature space $\mathcal{X}_{a,b}$, and slightly complex model, this becomes infeasible due to an exponentially increasing parameter (hypothesis) space. Ben-David et al. (2006) thus define *Proxy* $\mathcal{A}$ *− distance* (PAD), which can approximate Empirical $\mathcal{H}$-Divergence (defined above). Here, $\mathcal{A}$-distance refers to the metric introduced in Kifer et al. (2004), which can be considered a generalised version of $\mathcal{H}$-divergence. Put simply, the idea is to run a learning algorithm training a classification model (a hypothesis class $\mathcal{H}$) over task $\mathcal{T}_{a,b}$ and use the classification error of the final hypothesis h^* as the minimum error (which the $\underset{h \in \mathcal{H}}{min}[.]$ term tries to find in Eqn. 4.17)

Definition 4.8. Proxy $\mathcal{A}$-distance (PAD): Let $\epsilon_{\mathcal{T}_{a,b}}(h)$ be the generalisation error of an arbitrary classifier ($h \in \mathcal{H}$) trained to perform task $\mathcal{T}_{a,b}$. Proxy $\mathcal{A}$-distance ($\hat{d}_{\mathcal{A}}$) can then be defined as:

$$\hat{d}_{\mathcal{A}} = 2(1 - 2\epsilon_{\mathcal{T}_{a,b}}(h)) \tag{4.19}$$

In practice, PAD is computed by an linear SVM (Glorot et al., 2011), or an MLP (Chen et al., 2012) trained on a split of labeled examples available under $\mathcal{T}_{a,b}$; whose classification error is computed over another subset of $\mathcal{T}_{a,b}$. In our setting, outlined by Equations (4.4) to (4.6), calculating PAD would be akin to only training the model to predict $\hat{y}^{dom}$.

In the following section, we discuss precisely how is the notion of domain divergence used to justify the domain adversarial training procedure (Sec. 4.2).

4.3.3 A bound relating the source and target error

The domain adversarial training procedure is primarily premised on the following theorem which bounds the error of a classifier on the unlabeled target domains based on (i) error on the source domain, (ii) domain shift i.e. the domain divergence between the two domains, and (iii) task shift i.e. the difference in labeling functions (or the conditional distribution $p_{\text{data}}(Y|X)$) across the domains.

Theorem 4.3.3. (Ben-David et al., 2010, Theorem 1) Let $\epsilon_{\mathcal{T}_a}(h)$ be the generalisation error of a classifier h on task $\mathcal{T}_a$, and l_a be the labeling function $l_a : \mathcal{X}_a \mapsto \mathcal{Y}_a$ for the task, corresponding to $p_{\text{data}}(Y_a|X_a)$. Similarly, let $\epsilon_{\mathcal{T}_b}(h)$, l_b correspond to $\mathcal{T}_b$ whose error we wish to bound. We can then define the following inequality:

$$\begin{aligned}\epsilon_{\mathcal{T}_b}(h) \leq \epsilon_{\mathcal{T}_a}(h) + d_1(\mathcal{D}_a, \mathcal{D}_b) \\ + \min\left\{\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(X_a)}\left[|l_a(\boldsymbol{x}) - l_b(\boldsymbol{x})|\right], \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(X_b)}\left[|l_a(\boldsymbol{x}) - l_b(\boldsymbol{x})|\right]\right\}\end{aligned} \tag{4.20}$$

We are familiar with the first (Def. 4.4) and second (Def. 4.6) terms on the right side of the inequality. The third term represents the difference between the labeling functions across the two tasks, quantifying our notion of task shift (Sec. 2.4). However, in our scenario we cannot estimate this term, since apart from access to the real distributions, we do not even have access to labeled samples from $\mathcal{T}_b$. Further, akin to L^1 divergence, this term is a conservative bound since it includes instances which a given hypothesis class might not be able to solve. However, as mentioned in Sec. 4.3.1, we assume that joint error across the tasks to be low, and thus approximate it with γ for our purposes.

Further, for reasons mentioned in Sec. 4.3.2, we would like to replace L^1 divergence with a generous measure of domain divergence which can be empirically measured given samples from the domains - PAD (Def. 4.8). To do so, we first replace L^1-divergence term in Eqn 4.20 with $\mathcal{H}$-divergence. Then, with the help of Lemma 4.3.2, we replace it with its empirical variant (Lemma 4.3.1).

Through these modifications, we finally arrive at the following inequality:

Theorem 4.3.4. (Ben-David et al., 2006, Theorem 1) Let $\mathcal{H}$ be a hypothesis class of VC dimension d. With probability $1 - \delta$ over the choice of samples $X_a \sim (\mathcal{D}_a)^n$ and $X_b \sim (\mathcal{D}_b)^n$, for every $h \in \mathcal{H}$:

$$\epsilon_{\mathcal{T}_b}(h) \leq \hat{\epsilon}_{\mathcal{T}_a}(h) + \hat{d}_{\mathcal{H}}(X_a, X_b) + \gamma + c \tag{4.21}$$

where c is a term invariant to data samples or hypothesis h[5].

Through their experiments, (Ben-David et al., 2006) confirm that algorithms which not only have low error on $\mathcal{T}_a$, but also have a low $\mathcal{H}$-divergence achieves a lower error on $\mathcal{T}_b$. They conclude their discussion section with the following statement - "Rather than heuristically choosing a representation, as previous research has done [(Blitzer et al., 2006)], we can try to learn a representation which directly minimizes a combination of the terms in [T]heorem [4.3.4]".

4.3.4 From Theory to Domain Adversarial Training

Based on the findings above, it is desirable to find a hypothesis h' which is unable to distinguish between domains, i.e. is *domain invariant* (Sec. 4.1). However, h' must also exhibit low empirical error on $\mathcal{T}_a$ for it to perform well on the unlabeled task $\mathcal{T}_b$. This can be problematic if making predictions for $\mathcal{T}_a$ involves features specific to its domain. Thus, generally, it would be easier for a model to either have low source error, or to have low $\mathcal{H}$-divergence, but not both.

Also note that while the theorem was discussed in the context of unlabeled domain adaption, the resultant trade-off situation holds even while working with labeled target task, as is our case.

The domain adversarial training mechanism (outlined in Sec. 4.2), as proposed by Ganin et al. (2016) was created to minimise this trade-off between $\hat{\epsilon}_{\mathcal{T}_a}(h)$ and $\hat{d}_{\mathcal{H}}(X_a, X_b)$. Their first modification i.e. to approximate $\hat{d}_{\mathcal{H}}(X_a, X_b)$ with a logistic classifier f_{dom} stems from Proxy $\mathcal{A}$-distance (Def. 4.8). The domain divergence thus is inversely proportional to the empirical error of the logistic classifier over $\mathcal{T}_{a,b}$ i.e. the task of predicting the domain from which a given input was sampled.

Further, consider that we often minimise trade-offs by the means of augmentations to the loss (objective) function (e.g. L1 and L2 regularisation). Treating f_{dom} as a form of paramterised regularisation, we would thus want to add an additional term to our loss function ($\mathcal{L}_{task}$ in Eqn. 4.3) which calculates the loss between $\hat{y}^{dom}$ and $I_{X_a}(\boldsymbol{x})$ (the output of labeling function for task $\mathcal{T}_{a,b}$), referred to as $\mathcal{L}_{dom}$ hereon. However, if we simply *add* it to the Eqn. 4.3, the training algorithm would try to make features dissimilar across domains in order to minimise the domain classification loss $\mathcal{L}_{dom}$. We instead desire for the algorithm to minimise $\mathcal{L}_{task}$ (and correspondingly minimise $\hat{\epsilon}_{\mathcal{T}_{a,b}}(h)$) while maximising $\mathcal{L}_{dom}$, thereby characterising the aforementioned

[5]The actual value of c depends upon δ, number of samples n, and the VC dimension d of $\mathcal{H}$.

trade-off. We thus *subtract* $\mathcal{L}_{task}$ from the existing objective, along with a hyperparameter λ used to tune to trade-off between the two quantities. The resultant objective function is as defined in Eqn. 4.7.

Correspondingly we modify the parameter update equation for the encoder (Eqn. 4.8) (used while backpropagating) by accumulating the gradients from two classifiers ($\nabla_{\theta_{enc}}\mathcal{L}_{task}$, and $\nabla_{\theta_{enc}}\mathcal{L}_{dom}$) after flipping the polarity of the latter.

This concludes the reasoning behind the domain adversarial training mechanism. Training a model using these augmentations can impart a degree of domain invariance to the model. The exact extent of which indeed depends upon the domains at hand; whether the labeling functions across the domains is similar or not; and the hyperparameter λ. Note that this doesn't always translate to a positive performance improvement over tasks. Like other regularisations, domain adversarial training biases the learning procedure to favor a particular subset of the hypothesis class over others, which may be detrimental to performance over the source task. Thus, we perform a variety of experiments which we describe in the Chapter 5 geared towards a better understanding of the potential benefits and pitfalls of this technique.

Remark. So far, we have assumed that that $\mathcal{T}_a$, and $\mathcal{T}_b$ share the same label space $\mathcal{Y}$, and thus the same loss function $\mathcal{L}_{task}$. However the generic nature of domain adversarial training algorithm allows us to simply generalise for tasks with different label spaces and loss functions. In this case, our base optimisation equation (prior to domain adversarial augmentations) will resemble that of Eqn. 2.35.

4.4 Conclusion

In this chapter, we describe the domain adversarial training mechansim which induces a certain degree of domain invariance to the models. We discussed the works of (Kifer et al., 2004; Ben-David et al., 2006, 2010) which propose novel measures of divergence between domains, quantifying the domain shift. Building upon it, they theorise (Theorem 4.3.4) a bound on the generalisation error of a hypothesis class over an unlabeled domain. Through the theorem, they deduce that domain invariance (degree of inability of a class of hypothesis to predict the domains from which a given input was sampled) can be beneficial for the given scenario. We then discussed the work of (Ganin et al., 2016) which proposes the learning algorithm which can accomplish this. The resultant mechanism is described in Sec. 4.2, and illustrated by Fig. 4.1.

Chapter 5

Experiments

In this chapter, we describe two major experiments where we evaluate whether domain adversarial training (**DATr**) can be beneficial while using pretrained language models for a particular NLP task. We begin the discussion by outlining the intuition behind the experiments, and the evaluation metrics used in this empirical investigation.

5.1 Motivation

As discussed in Sec. 3.3, the transfer of pretrained language models can be tricky owing to the encoder's tendency to overfit on the current task, when faced with a substantial task or domain shift. We thus investigate whether the bias induced by DATr, namely – to make the encoder *domain invariant*, sufficiently regularise the model to prevent the aforementioned overfitting (Sec. 5.4).

Further, we intend to find whether this mechanism can help in multi-task scenarios. In these cases, when using a pretrained encoder to solve multiple NLP tasks at a time, the encoder can either (a) maintain domain invariance in its output representations, tasking the classifiers to make predictions based on general features of the text, or (b) learn to map inputs from different tasks to the shared latent feature space, enabling the use of domain specific features while making predictions. DATr in this scenario is expected to bias the model to do the former. Through our experiments, we attempt to quantify whether we can train an encoder to produce domain invariant representations in multi-task settings, and whether making predictions with only general features of the input is instead detrimental to model performance. We discuss this experimental setup, and its results in Sec. 5.5.

5.2 Tasks and Datasets

We evaluate our models over, arguably the simplest NLP task namely, **text classification**. Given a sequence of words, we expect our model to classify it in one of the n (predefined) classes. Given their use in recent text classification approaches, we choose to primarily evaluate the approach on the task of binary sentiment classification of movie reviews. We use the encoder-classifier architecture, as defined in Sec. 2.2.3 to solve the task. Also, in line with the premise of our work, we pretrain the encoder through the **language modeling** task over a domain-agnostic (or general purpose) corpus. For this, we simply use the encoder-generator architecture as discussed in Sec. 2.2.3.

The datasets used for our experiments include:

- **Wikitext103**[1]: Merity et al. (2017) released this dataset using text extracted from Wikipedia. Made for the purposes of language modeling, the dataset is an unlabeled collection of text across 28,475 articles containing 103,227,021 word tokens. We use this dataset to pretrain our language models for the purposes of downstream fine-tuning. We choose this dataset primarily for two reasons. Firstly, it is aptly sized to rigorously train our language model. PennTreeBank (Marcus et al., 1993), a much more popular dataset for this task, contains roughly 1M tokens consisting of words from a very limited vocabulary of 10k words. Whereas, (Chelba et al., 2014) consisting of 1B word tokens, is a collection of sentences in random order, preventing the learning of long term dependencies as pointed out by (Merity et al., 2017; Devlin et al., 2018). Secondly, an important work whose technique for using pretrained language models (Howard and Ruder, 2018) (also discussed in Sec. 3.3) we incorporate as our baseline, pretrain their models with this dataset.

- **PL04 (Pang and Lee, 2004)**[2]: A collection of 2000 movie reviews was released by (Pang and Lee, 2004). They collected movie reviews written by 312 authors before 2002, taking a maximum of 20 reviews per author. The reviews are labeled as positive or negative, depending on the ratings assigned by the reviewer along with their review. The data points (review, labels) are balanced across the labeling space, that is to say that there are 1000 positive, and 1000 negative reviews

[1]This dataset is available for download at `https://blog.einstein.ai/the-wikitext-long-term-dependency-language-modeling-dataset/` under the Creative Commons Attribution-ShareAlike 3.0 Unported License.

[2]Available at `https://www.cs.cornell.edu/people/pabo/movie-review-data/` under the name - **polarity dataset v2.0**.

in the dataset. On an average, each review has approximately 813 word tokens, across a vocabulary of 36,658 words. Note, that in practice, we don't use all of the vocabulary (See Sec. 5.4.1). Further, the authors do not specify the train-test splits of the data, thus we generally perform an 80-20 split (train on 1600 reviews, and evaluate the model on 400).

- **MD11 (Maas et al., 2011)**[3]: Similar to (Pang and Lee, 2004), this dataset is a collection of labeled movie reviews, however larger in size, collected from IMDb[4]. Specifically, it consists of 50,000 labeled, and 50,000 unlabeled reviews. In our experiments, we use the unlabeled reviews while pretraining the language model, but not while performing the actual classification task. The labeled reviews are balanced across the labeling space, same as above. Interestingly, the train-test splits of the dataset consist of 25,000 reviews each. This 50-50 split enables the low variance in the performance estimate over the dataset (Maas et al., 2011, Sec. 4.3.2). While consisting of movie reviews from the same source, the reviews in this dataset are considerably shorter - having approx. 278 words per review.

Thus MD11(Maas et al., 2011) and PL04(Pang and Lee, 2004) constitute the two datasets consisting of *domain specific* text for our purposes, while Wikitext103 (Merity et al., 2017) is used as an *domain agnostic* corpus.

5.3 Models

The models that we use across all our experiments are composed of the following sub-networks.

- **Encoder**: The most common sub-network which we use across all our settings, the encoder (Sec. 2.2.3) is primarily tasked with mapping the input word sequences to a latent space. In practice, we use a three layered, bi-directional LSTM model along with an embedding layer. Our encoder is loosely based on AWD-LSTM (Merity et al., 2018) which is a culmination of a variety of regularisation techniques including DropConnect (Wan et al., 2013), embedding dropout (Gal and Ghahramani, 2016), and, independent embedding and hidden size. The bottom most layer of the encoder, namely the embedding layer is randomly initialised with a 400 dimensional vector corresponding

[3]Publicly available at `http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz`

[4]Internet Movie Database - `https://www.imdb.com/`

to each word in the given vocabulary. The first and second recurrent layers are of 1150 dimensions each, and the final one has the same output dimension as that of the embedding – 400. We represent the encoder simply with the following equation:

$$h_t^{enc} = f_{enc}(\mathbf{x}_t; \theta_{enc}) \tag{5.1}$$

where f_{enc} collectively represents the embedding and the encoding layers.

- **Generator**: The generator is a simple single layered feedforward (Sec. 2.2.2) sub-network with softmax activations, which we use in our language model to output a distribution over the entire vocabulary (the label space for the language modeling task). The generator takes vectors of 400 dimensions as inputs, and linearly transforms them to a $|V|$ dimensional vector before passing it through a softmax layer. Here $|V|$ represents the number of words in our vocabulary. Similar to (Howard and Ruder, 2018), and as suggested in (Inan et al., 2017; Merity et al., 2018) the actual weights of the generator are tied to that of encoder's embedding layer, which has shown to be empirically beneficial for the language modeling task. We use the following two equations (in conjunction with Eqn. 5.1) to represent our model:

$$\hat{\mathbf{y}}_t^{task} = f_{gen}(h_t^{enc}; \theta_{gen}) \tag{5.2}$$

$$p_{\text{model}}(\mathbf{y}|\mathbf{x}; \theta) = \text{softmax}(\hat{\mathbf{y}}_t^{task}) \tag{5.3}$$

- **Classifier**: The classifier, similar to the generator above is a (set of) layers which produce a distribution over the label space. Since here we're making sequence level predictions as opposed to word level predictions (above), we require a fixed length vector representing the input sequence. We thus concatenate the encoder's output corresponding to the last token in the input sequence, along with a mean pooled and a max pooled representation of the entire encoded sequence to represent it. We pass this vector through two feed forward layers with ReLU () activations on the first, and a softmax on the latter to generate the final distribution.

$$\mathbf{h} = \left[h_T^{enc}, pool_{avg}(h_{1:T}^{enc}), pool_{max}(h_{1:T}^{enc})\right] \tag{5.4}$$

$$\hat{\mathbf{y}}^{task} = f_{clf}(\mathbf{h}; \theta_{clf}) \tag{5.5}$$

$$p_{\text{model}}(\mathbf{y}|\mathbf{x}; \theta) = \text{softmax}(\hat{\mathbf{y}}^{task}) \tag{5.6}$$

In practice, we use a two layered classifier with 1200×50, and $50 \times n_{classes}$ dimensional weight matrices respectively. We use this module as a task classifier, as well as a domain regressor in our experiments, and represent the latter with f_{dom} instead.

- **Gradient Reversal Layer**: As described in Sec. 4.3.4, DATr requires the polarity of the gradients to be flipped while backpropagating from the domain regressor to the encoder. We use a simple parameter-less identity function for these purposes. During forward propagation, the gradient reversal layer acts as an identity function. However, during back-propagation, it multiplies the gradients with -1.

All the models in our experiments are composed of a combination of these four functions (interchangeably referred to as layer, modules or sub-networks). Among these, the encoder is of paramount importance, since throughout our experiments we strive to provide it with an optimum understanding of the text, and transfer its parameters across different models. We shall briefly discuss the exact models we use, in the Experimental Setup subsections of the following experiments.

5.4 Empirical Evaluation of Domain Adversarial Training

As suggested in Sec. 5.1, through this experiment, we wish to find if domain adversarial setting can benefit the pretrain-finetune procedure. The source task (on which we pretrain our models) remains the same – language modeling over Wikitext103 in all the variations mentioned below. The target task, is that of binary classification of movie reviews as positive or negative. We perform the experiment separately for both PL04 and MD11 datasets.

5.4.1 Experimental Setup

Our setup can be characterized by (i) the schedule in which we train and transfer our models; (ii) the exact architecture and layers of the model; and (iii) hyperparameters. We begin with a description of the transfer schedules which we augment and compare within this experiment.

(a) $\mathbf{P}_1 \rightarrow \mathbf{P}_2 \rightarrow \mathbf{P}_3$: Recall the three phased transfer schedule (discussed in Sec. 3.3). Briefly, it involves the following steps: (i) $\mathbf{P}_1$: Pretrain a encoder-classifier based language model on a domain agnostic corpus;

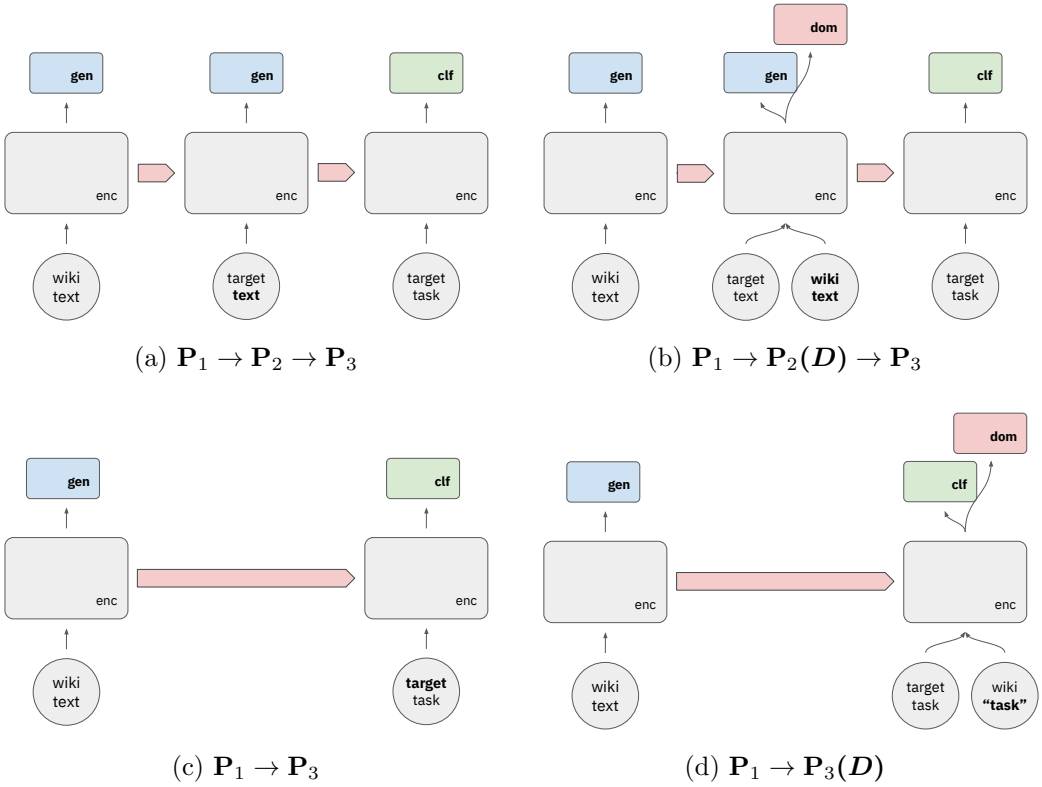


Figure 5.1: An illustration of the four learning schemes we use in the first experiment. Here, the blue colored rectangles (■) denote the generator f_{gen}; the green ones (■) denote the classifier f_{clf}; the red ones (■) denote the domain regressor f_{dom}. The description of each of the four configurations can be found in Sec. 5.4.1.

(ii) $\mathbf{P}_2$: Finetune the pretrained language model on domain *specific* corpus – specifically the unlabeled text of the target task; and (iii) $\mathbf{P}_3$: Use the encoder from P_2 to initialise a text classification model, and finetune it on the actual labeled samples from the task.

(b) $\mathbf{P}_1 \rightarrow \mathbf{P}_2(\boldsymbol{D}) \rightarrow \mathbf{P}_3$: The second phase of the aforementioned schedule aims to partially bridge the domain shift by training on the unlabeled samples from the target task. Through DATr, we intend to make this process smoother. We thus, not only sample word sequences from the target task (PL04 or MD11), but also from the source task (Wikitext103). Further, we adapt the DATr algorithm (Sec. 4.2) to train the encoder to be invariant to samples from both the source and target domains. In this setting, the P_1 and P_3 step remains unchanged.

(c) $\mathbf{P}_1 \rightarrow \mathbf{P}_3$: We simplify the first schedule to transfer the language model trained over domain agnostic text (P_1) *directly* to the target task (P_3).

(d) $\mathbf{P}_1 \rightarrow \mathbf{P}_3(\boldsymbol{D})$: In this variant, we modify the final phase of the $P_1 \rightarrow P_3$ schedule. We add a domain regressor (f_{dom}) along with the task specific classifier (f_{clf}) to the model, and feed it inputs from the target dataset as well as Wikitext103 (the source dataset). We train f_{dom} on samples from both domains, but train f_{clf} only on samples from the target task.

(e) $\mathbf{P}_3$: For the purpose of illustrating the effect of pretraining, we also incorporate a non-pretrained model in our experiment. In this variant, we simply initialise the model parameters randomly and train on the given task.

Figure 5.1 illustrates these transfer schedules, although in an abstract manner. Given its straightforward nature, we omit visualising the P_3 learning scheme. Further, note that since our encoder architecture is consistent with that of Howard and Ruder (2018), we use the language model provided by them for P_1[5].

Hyperparameters and other Configurations

The work of (Howard and Ruder, 2018) which proposes the three phased transfer mechanism also claims that the wide variety of regularisation techniques used by them are beneficial (and in some cases, pivotal) for the performance improvement shown by their tasks. We thus implement them in our experiments as well, as described below. Note that along with the description of the technique, we mention the models and the phases (See 5.4.1) during which they're used.

Optimiser: We use SGD to optimise our language models (P_2), given that SGD without momentum has been found to outperform most other optimisation algorithms for the neural language modeling task (Merity et al., 2018). For P_3, we use an Adam (Kingma and Ba, 2015) optimiser, with β_1, β_2 set as 0.7 and 0.99 respectively as suggested in (Dozat and Manning, 2017).

Variable Length Backpropagation: As pointed out in (Merity et al., 2018), using truncated BPTT () to train our language models with static sequence length t_{tbptt} results in $1/{t_{tbptt}}^{th}$ of the training data to never contribute to the training. We thus randomly alter sequence length for each mini-batch, as suggested. This is used only while training the language models, i.e. P_2.

Dropouts: We extensively use dropout in our networks, starting with the embedding layer. The embedding dropout works in two phases wherein

[5]Available for download at `http://files.fast.ai/models/wt103/`.

first we drop entire words from the embedded output with $p = 0.014$ during P_2, and $p = 0.025$ during P_3. Further, we randomly drop individual weights from the embedded representations with $p = 0.175$ during P_2, and $p = 0.20$ during P_3.

Dropouts on the recurrent layer include a weight dropout of $p = 0.14$ during P_2, and $p = 0.25$ during P_3; standard dropout with $p = 0.10$, $p = 0.15$ on activations going from one LSTM layer to another during P_2, P_3 respectively. Finally, within the generator (f_{gen}), we drop elements from the input with $p = 0.07$; in classifiers (f_{clf}, f_{dom}) with $p = 0.2, 0.1$ (corresponding to each layer).

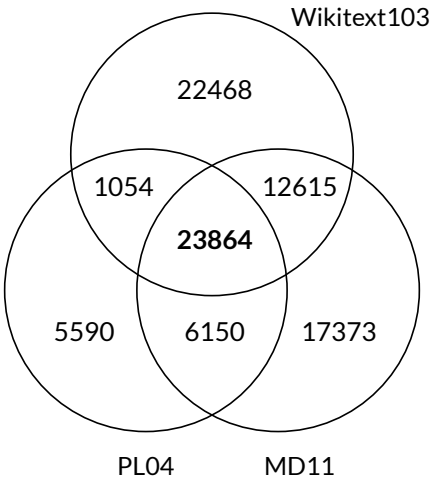


Figure 5.2: A Venn diagram depicting the intersection of vocabulary of the three datasets we use in our experiments, as discussed in Sec. 5.2

.

Vocabulary: If the words constituting the vocabulary of a given task do not appear (frequently) during pretraining (i.e. $V_{unk} = \{w_i | w_i \in V_{task} \cap \neg V_{wikitext}\}$), the encoder would have little to no understanding of them. In these cases (visualised in Fig. 5.2), we initialise the embedding vectors corresponding to these *unknown* words with a random vector. In practice, the vocabulary of a given dataset is restricted to contain a maximum of 60,000 words, where each word must have appeared at least twice in the dataset.

Learning Rates: Our learning rate schedules are similar to that proposed in (Howard and Ruder, 2018), includes the use of slanted triangular learning rate (SLTR) (which changes the learning rate across iterations[6]), discriminative finetuning (DSCR) (decrease the learning rate by a certain factor per layer during fine-tuning) during P_2. For brevity's sake, we omit the descriptions of these schedules, and only mention the hyperparameters we changed. We set the initial learning rate for both phases at 0.003. We find that combining domain regressor in the model, along with a 2.6 factor of decay per layer (as a part of DSCR) slows the model convergence substantially. We thus set it to be 1.3. Further, we find that decaying the learning rate within each epoch based on the cosine annealing schedule (Loshchilov and

[6]Here, one iterations refers to one gradient update step. Thus #iterations = #epochs × #mini-batches per epoch

Hutter, 2017), with one cycle per epoch is beneficial across multiple model and transfer schedules, and is thus incorporated.

Freezing Schedule: Freezing schedules aim to make certain parts of a model non-trainable for a particular time during the training, typically to avoid overfitting or catastrophic forgetting. At the start of P_2, we freeze the encoder and train only the generator for one epoch, after which we proceed as normal (with DSCR scaled learning rate per layer). During P_3, we instead freeze the entire model, and unfreeze a layer per epoch starting from the top (final layer of the classifier).

Loss Functions: We use weighted cross entropy for both $\mathcal{L}_{task}$, and $\mathcal{L}_{dom}$. While the classes of the target task are balanced in our experiments, the samples used for calculating $\mathcal{L}_{dom}$ vary, and thus are appropriately weighted within the loss function.

Domain Regressor Ratio (λ): The scaling factor used in Eqn. 4.7 is used to tune the trade-off between $\mathcal{L}_{task}$ and $\mathcal{L}_{dom}$, as discussed in Sec. 4.3.4. We find that setting $\lambda = 6$ leads to a consistent performance improvement on the target task across multiple settings, and thus keep it unchanged through all our experiments. Further, we discuss the impact of different values of λ in Sec. 5.4.3.

Remark. The hyperparameter values were not decided through an exhaustive grid search, due to practical limitations. However, we find that the models were very sensitive towards the initial learning rate, which when set at the initial value of 0.003 demonstrated the most consistent performance across multiple phases, and multiple models.

The dataset splits and model configurations mentioned in Sec. 5.2, 5.3 remain unchanged and hence aren't repeated here.

5.4.2 Results

Through our experiments, we find that pretraining our models as a language model over Wikitext is beneficial, across both the datasets. Further, despite different learning schemes leading to different results, the accuracies over the final task lie in the narrow margin between 92.53% to 93.44% over MD11, and 87.50% to 90.76% over PL04 (excluding the non-pretrained variant). The results on the latter vary more, which is understandable given the difference in the sizes of training data between the two. We report the results of this experiment in Table 5.1.

Surprisingly, we find that the simplest $P_1 \rightarrow P_3$ transfer technique outperforms the three phased setting, both with and without domain adversarial training. We attribute this discrepancy with results from (Howard and

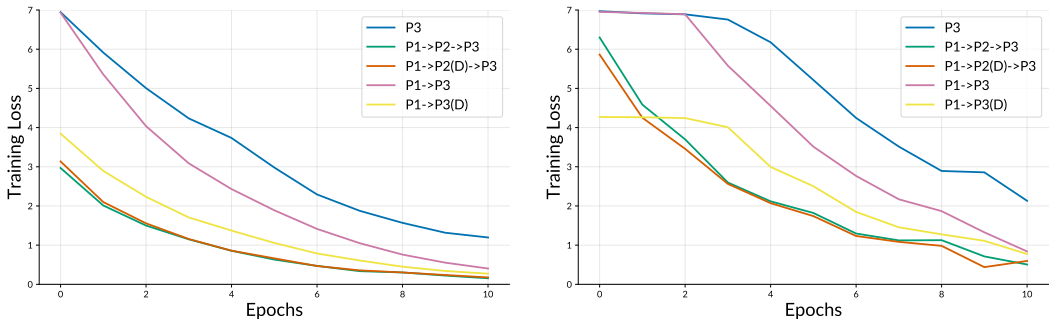
Scheme	MD11			PL04		
	TASK	DOM_{P2}	DOM_{P3}	TASK	DOM_{P2}	DOM_{P3}
$P_1 \rightarrow P_2 \longrightarrow P_3$	92.65	NA	NA	87.50	NA	NA
$P_1 \rightarrow P_2(\boldsymbol{D}) \rightarrow P_3$	92.53	51.90	NA	88.75	86.74	NA
$P_1 \longrightarrow P_3$	**93.44**	NA	NA	89.00	NA	NA
$P_1 \longrightarrow P_3(\boldsymbol{D})$	93.31	NA	55.42	**90.76**	NA	62.69
P_3	89.39	NA	NA	81.60	NA	NA

Table 5.1: Performance of different transfer learning schedules over the **MD11** (Maas et al., 2011), and **PL04** (Pang and Lee, 2004) sentiment classification tasks. Here, TASK refer to accuracy (denoted in percentage points) of the model when predicting the target label. DOM_{P2} and DOM_{P3} refer the model's accuracy when predicting the source of the input (domain label) during second and third phase of training respectively.

Ruder, 2018) to the change in hyperparameters including initial learning rate and a slightly changed freezing schedule. As mentioned above, we changed these hyperparameters when doing so was beneficial across multiple models and transfer settings, thereby deviating from those used by (Howard and Ruder, 2018) which were decided through a much more focused search for their transfer learning scheme.

Further, when focusing on the results of one dataset (MD11, PL04) at a time, we find that the DATr affects the performance differently in both cases. On PL04, it has tangible benefits (+1.25%, +1.76%) when compared to models trained without it. Consider that PL04 has only 1600 samples to train on. Given this, and the fact that pretrained models tend to converge very quickly (discussed in Sec. 5.4.3), the results suggest that DATr can offset the tendency to overfit on small datasets, effectively regularising the model. On MD11, DATr does not improve the task accuracy, and is in fact, slightly detrimental (-0.12%, -0.13% respectively). Given that MD11 has 25 times more data to train on, the above result suggests that the increased training data itself regularises the model, rendering the effect of DATr redundant (and hence, detrimental). The redundancy is also evident given the minute difference in accuracies between (a), (b), and (c), (d) pairs.

To conclude, we find that it is easier to classify representations which contain features specific to the target task. However, regulating this specificity (by learning domain invariant representations) prevents the model from overfitting, in low resource scenarios (See discussion regarding generalisation gap below). Further, that domain adversarial training is an effective means



(a) MD11 (b) PL04

Figure 5.3: Plots demonstrating the decrease in model loss w.r.t. epochs.

of regulating this specificity, but this augmentation makes the model susceptible to overpowering the domain invariance, and thus can be detrimental to model performance.

5.4.3 Further Analysis

In this section, we take a more focused look on the results above, aiming for a better understanding of the effect of DATr.

Model Convergence: We begin the discussion by talking about the convergence of our models under different training schemes. Fig 5.3a, and Fig. 5.3b visualise $\mathcal{L}_{task}$ (loss on the target task) w.r.t. epochs. Here, by convergence, we mean that the parameters reach a local minima in the loss surface, where their training accuracy reaches near 100, and loss decreases substantially, as is further hinted by small gradient steps around these iterations. We make the following observations through the loss plots, across both the datasets: (i) pretraining the models lead to a significantly faster convergence, and that (ii) the three-phased transfer learning further decreases the convergence time.

Amongst the pretrained models, while the three-phased schedules converge faster, their performance lags slightly behind the two-phased schedules. This suggests that the former converges to a suboptimal minima w.r.t the latter. We call this convergence sub-optimal since the models exhibit a higher generalisation error (or error computed over test data), as well as a higher generalisation gap (the difference between accuracy over the training and test set) (See Table 5.2).

Effect of Domain Regressor Ratio (λ): In our main experiment, we arrived at a value of 6 for λ based on careful considerations of the trade-off

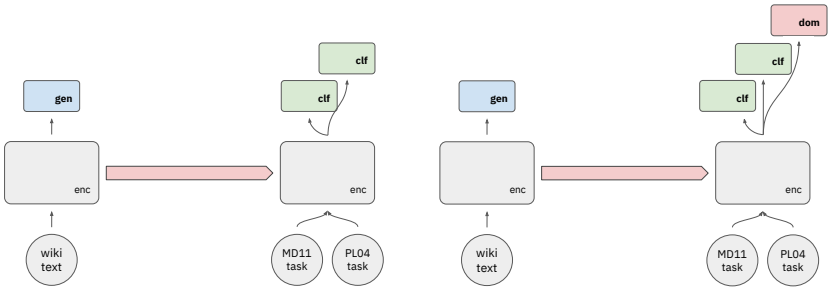
Scheme	MD11			PL04		
	Train	Test	DOM_{P3}	Train	Test	DOM_{P3}
$\text{P}_1 \rightarrow \text{P}_3$	96.27	93.44	NA	97.13	89.00	NA
$\text{P}_1 \rightarrow \text{P}_3(\boldsymbol{D_3})$	97.94	92.59	55.92	92.76	89.66	92.35
$\text{P}_1 \rightarrow \text{P}_3(\boldsymbol{D_6})$	95.90	93.31	55.42	92.49	90.76	62.69
$\text{P}_1 \rightarrow \text{P}_3(\boldsymbol{D_{12}})$	95.65	93.32	55.41	89.63	88.04	62.23

Table 5.2: Effect of different values of λ for the two-phased transfer scheme. For each row, the value of lambda is denoted by the suffix of $(\boldsymbol{D})$. Here, *Train*, *Test*, and DOM_{P3} refer to the task accuracy over train set, test set, and the domain classification accuracy over the test set respectively. Finally, note that the first and third row of the table correspond to experiments already mentioned in Table 5.1. We repeat them here for completion's sake.

between domain classification accuracy and task accuracy. In this experiment, we intend to illustrate the effect of this choice. Thus, we repeat the aforementioned experiment for different values of λ including $3, 6$ and 12. To retain the focus of this experiment, we only perform it for the two phased learning schemes (scheme (c) & (d)). We report its results in Table 5.2.

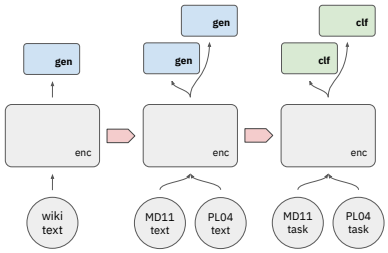
Through the results, it is evident that DATr has little to no effect on the performance over MD11, regardless of the value of λ, but does affect the performance over PL04 (the smaller dataset). This strengthens the evidence that ample training data can itself regularise the model and prevents the model from overfitting. Focusing on the performance over PL04, we find that $\lambda = 6$ provides substantial performance improvements, $\lambda = 12$ overpowers the regularisation whereas a value of 3 has less effects on the domain invariance, as evident by $\text{DOM}_{\text{P3}} = 92.35\%$. Thus, given its consistent performance across the task we choose $\lambda = 6$ for the main experiment. However, we recommend interested readers to experiment with different values when using DATr in a different setting.

Further, a higher value of λ consistently decreases the generalisation gap (difference between accuracy on training data and accuracy on test data), some cases at the expense of increasing the empirical error.

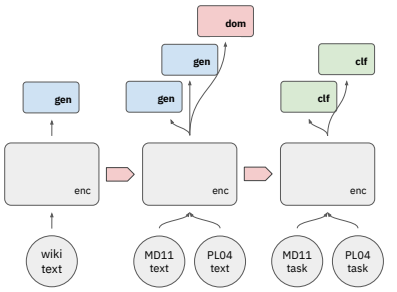


(a) $\mathbf{P}_1 \rightarrow \mathbf{P}_3(\boldsymbol{M})$

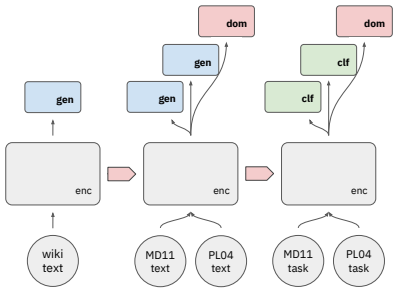
(b) $\mathbf{P}_1 \rightarrow \mathbf{P}_3(\boldsymbol{M})\ (\boldsymbol{D})$



(c) $\mathbf{P}_1 \rightarrow \mathbf{P}_2(\boldsymbol{M}) \rightarrow \mathbf{P}_3(\boldsymbol{M})$



(d) $\mathbf{P}_1 \rightarrow \mathbf{P}_2(\boldsymbol{M})\ (\boldsymbol{D}) \rightarrow \mathbf{P}_3(\boldsymbol{M})$



(e) $\mathbf{P}_1 \rightarrow \mathbf{P}_2(\boldsymbol{M})\ (\boldsymbol{D}) \rightarrow \mathbf{P}_3(\boldsymbol{M})\ (\boldsymbol{D})$

Figure 5.4: An illustration of the five learning schemes we use in the second experiment. Similar to Fig. 5.1, the blue colored rectangles (■) denote the generator f_{gen}; the green ones (■) denote the classifier f_{clf}; the red ones (■) denote the domain regressor f_{dom}. The description of each of the four configurations can be found in Sec. 5.5.1.

5.5 Experiment: Domain Adversarial Training in a Multi-task Setting

As discussed in Sec. 5.1, domain adversarial training in a multitask scenario tilts the models towards not overloading a shared latent space with features corresponding to multiple domains/tasks. Instead, it incentivizes the use of domain invariant features for the purposes of solving the target tasks.

In this experiment, we attempt to find the extent to which DATr enforces this, and its corresponding effect on the model performance.

5.5.1 Experimental Setup

While we do not *include* the task index in the input representations, we provide the model with it for the following purposes: Given a multi-task scenario with N tasks, we train our models with N different classifiers f^n_{clf} and pass the encoded input representation to the classifier corresponding to its class. Doing so enables us to discount confusions caused by the task shift as a possible reason for generalisation errors.

The variations in transfer schemes (two and three phased), along with possible variations of multi-task classification increases the number of schedules we evaluate in this experiment. In all cases, however we use MD11 and PL04 as our target task, and define a hybrid task $\mathcal{T}_{a,b}$ representing the two together. We illustrate these models in Fig. 5.4, and provide their description below:

(a) $\mathbf{P}_1 \rightarrow \mathbf{P}_3(\boldsymbol{M})$: In the simplest setting, we take the pretrained language model's encoder to initialise the task specific model. Further, as denoted by the $(\boldsymbol{M})$ suffix, we solve $\boldsymbol{T}^{clf}_{a,b}$ here, which is to say that we sample inputs $x \sim X_{MD11} \cup X_{PL04}$, and ask the model to predict a label from $\mathcal{Y}_{MD11}$ or $\mathcal{Y}_{PL04}$ depending upon the task index. Correspondingly, we initialise two task-specific classifiers f^{MD11}_{enc}, and f^{PL04}_{enc} both of which use the shared encoder's outputs as their input to make the prediction.

(b) $\mathbf{P}_1 \rightarrow \mathbf{P}_3(\boldsymbol{M})$ $(\boldsymbol{D})$: We augment the setting above by adding a domain regressor f_{dom} along with the two existing classifiers and train the entire model based on the domain adversarial training Algorithm.

(c) $\mathbf{P}_1 \rightarrow \mathbf{P}_2(\boldsymbol{M}) \rightarrow \mathbf{P}_3(\boldsymbol{M})$: In this baseline three phased scheme, we augment the phase two to solve $\mathcal{T}^{lm}_{a,b}$. Akin to the first configuration, we use two different generators f^{MD11}_{gen} and f^{PL04}_{gen} which are used to predict the next word indicated by the task vector sampled along with inputs

Scheme	MD11	PL04	DOM_{P2}	DOM_{P3}
$\mathbf{P}_1 \rightarrow \mathbf{P}_3(\boldsymbol{M})$	**93.04**	93.66	NA	NA
$\mathbf{P}_1 \rightarrow \mathbf{P}_3(\boldsymbol{M})\ (\boldsymbol{D})$	92.46	91.05	NA	93.98
$\mathbf{P}_1 \rightarrow \mathbf{P}_2(\boldsymbol{M}) \rightarrow \mathbf{P}_3(\boldsymbol{M})$	91.83	93.56	NA	NA
$\mathbf{P}_1 \rightarrow \mathbf{P}_2(\boldsymbol{M})\ (\boldsymbol{D}) \rightarrow \mathbf{P}_3(\boldsymbol{M})$	92.85	**94.25**	54.46	NA
$\mathbf{P}_1 \rightarrow \mathbf{P}_2(\boldsymbol{M})\ (\boldsymbol{D}) \rightarrow \mathbf{P}_3(\boldsymbol{M})\ (\boldsymbol{D})$	92.64	92.87	54.46	93.89

Table 5.3: Performance of different transfer learning schedules when solving MD11 and PL04 together in a multi-task setting.

$x \sim X_{MD11} \cup X_{PL04}$. After fine-tuning the language model encoder on target text in this manner, we transfer it to the task specific model which follows the same setup as (a).

(d) $\mathbf{P}_1 \rightarrow \mathbf{P}_2(\boldsymbol{M})\ (\boldsymbol{D}) \rightarrow \mathbf{P}_3(\boldsymbol{M})$: Same as above but with a domain regressor trained along with the language model using DATr. Under this scenario, we use DATr to impart a degree of domain invariance to the language model, but do not reinforce it during the actual task.

(e) $\mathbf{P}_1 \rightarrow \mathbf{P}_2(\boldsymbol{M})\ (\boldsymbol{D}) \rightarrow \mathbf{P}_3(\boldsymbol{M})\ (\boldsymbol{D})$: Finally, we augment the third phase with domain adversarial training effectively trying to maintain the invariance across both fine-tuning phases.

Hyper-parameters and other configurations

We keep this unchanged from the first experiment (See Sec. 5.4.1) except for the follows:

Vocabulary: When solving two target tasks (MD11, PL04) at a time, it is pivotal to have the maximum vocabulary overlap between the two. However if we let our vocabulary constitute of all the words across both the datasets, we risk suffering from data sparsity and a corresponding performance loss. Thus, we take the most frequent $60,000$ words from MD11, and add another $2,000$ most frequent words from PL04 which aren't already included, and therefore have a maximum of $62,000$ words in our vocabulary.

5.5.2 Results

Table 5.3 lists the results of this experiment. We see that the simplest scheme, namely (a) has the best performance over MD11, whereas (d) has the highest accuracy over PL04. Overall, the performance of all the variations lie within

a narrow range of 91.83% and 93.04% on MD11, and 91.05% and 94.25% over PL04.

Upon comparing these results to that of the first experiment (see Sec. 5.4.2), we notice that the performance of these models over PL04 increases substantially, even in the absence of domain adversarial training (93.66%, 93.56% for (a) and (c) respectively; whereas the best performing approach in the first experiment had an accuracy of 90.76%). This suggests that training the encoder on MD11 (along with PL04) imparts an understanding of the inputs which is beneficial for the latter.

Further, we notice that adding domain regressor as an auxiliary task during P_3 negatively affects the performance over PL04 (-2.61% between (b) & (a), -1.38% between (e) & (d)). Taking this into account, along with the above suggests that the models derive value from domain specificity and trying to curb it results in a performance loss. This is further reinforced upon observing the actual value of the domain regressor in the third phase. Comparing the DOM_{P3} values between Table. 5.3 and Table. 5.1, we find that enforcing domain invariance is more ineffective in the multi-task setting.

Thus, to conclude, this experiment suggests that domain invariance in a multi-task setting provides no tangible benefits[7]. We restrict our findings to when trying to perform binary classification of movie reviews across two different datasets, and leave the investigation of DATr on multi class classification, and more complex tasks for future investigations.

[7]While scheme (d) outperforms the others over PL04, it doesn't over MD11. Considering that MD11's test set has 25,000 samples whereas PL04's has only 400, we assign more prominence to models which exhibit better performance over MD11.

Chapter 6

Conclusions and Future Work

In this thesis, we investigated whether enforcing domain invariance can improve the performance of recurrent neural models on common NLP tasks. Specifically, we ask if it can bridge the domain gap incurred when transferring pretrained language models for the purposes of finetuning over a target task. To that end, we use the domain adversarial training algorithm, proposed in (Ganin et al., 2016) while finetuning the models, and experiment with this augmentation in a variety of schemes. For our experiments, we pretrain a language model on Wikitext103 (Merity et al., 2017), and finetune its encoder, for the purposes of classifying movie reviews. We use two datasets for the latter namely, MD11 (Maas et al., 2011), and PL04 (Pang and Lee, 2004).

Through our experiments, we find that enforcing domain invariance regularises the model, and decreases the generalisation gap. Thus, in low resource settings, i.e. target tasks with insufficient labeled examples, domain adversarial training can benefit model performance. However, this form of regularisation is detrimental in cases where ample training data is available. Further, we find that in a multi-task scenario, i.e. when we train our model to perform classification on both the target tasks simultaneously, an increased number of labeled examples renders the use of domain adversarial training ineffective.

6.1 Future Work

In the future, we are interested in repeating our experiments with a wider set of NLP tasks for a more comprehensive understanding of the effects of domain invariant representations. We intend to investigate the effects of domain adversarial training in a multi-task setting involving pairs of tasks

with different complexity and abstraction. Further, we want to explore the use of other auxiliary tasks such as sequence recreation (Rei, 2017), and low level NLP tasks (Niehues and Cho, 2017) along with our augmentations.

Given that the generalisation gap of deep learning models changes during the training process, we would like to experiment with changing the value of λ mid-training, preferably in a dynamic manner. Along similar lines, we would want to explore the use of changes in the training schedule by the means of interleaving mini-batches from across the domains, as opposed to interleaving individual samples within the mini-batches.

Recently, several works (Glockner et al., 2018; Carmona et al., 2018) focus on an active investigation of robustness of NLP models to out-of-distribution data, and noisy data. We intend to undertake similar investigations in the context of our domain adversarially trained models, premised on the fact that domain invariance prevents models from picking up idiosyncrasies of a particular dataset, enabling generalisation to a wider extent. To that end, we will incorporate experiments suggested in the aforementioned works, as well as adversarially generated examples, following the techniques proposed in (Jia and Liang, 2017; Lei et al., 2018).

Bibliography

Alberti, C., Lee, K., and Collins, M. (2019). A BERT baseline for the natural questions. *CoRR*, abs/1901.08634.

Auer, P., Herbster, M., and Warmuth, M. K. (1995). Exponentially many local minima for single neurons. In *Advances in Neural Information Processing Systems 8, NIPS, Denver, CO, USA, November 27-30, 1995*, pages 316–322.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Batu, T., Fortnow, L., Rubinfeld, R., Smith, W. D., and White, P. (2000). Testing that distributions are close. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 259–269.

Baxter, J. (2000). A model of inductive bias learning. *J. Artif. Intell. Res.*, 12:149–198.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

Ben-David, S., Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., and Vaughan, J. W. (2010). A theory of learning from different domains. *Machine Learning*, 79(1-2):151–175.

Ben-David, S., Blitzer, J., Crammer, K., and Pereira, F. (2006). Analysis of representations for domain adaptation. In *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, pages 137–144.

Ben-David, S. and Schuller, R. (2003). Exploiting task relatedness for mulitple task learning. In *Computational Learning Theory and Kernel Machines, 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003, Proceedings*, pages 567–580.

Bengio, Y., Frasconi, P., and Simard, P. (1993). The problem of learning long-term dependencies in recurrent networks. In *IEEE international conference on neural networks*, pages 1183–1188. IEEE.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, pages 41–48.

Bengio, Y., Simard, P., Frasconi, P., et al. (1994a). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

Bengio, Y., Simard, P. Y., and Frasconi, P. (1994b). Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks*, 5(2):157–166.

Berant, J., Chou, A., Frostig, R., and Liang, P. (2013). Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1533–1544.

Blitzer, J., McDonald, R. T., and Pereira, F. (2006). Domain adaptation with structural correspondence learning. In *EMNLP 2006, Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing, 22-23 July 2006, Sydney, Australia*, pages 120–128.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *TACL*, 5:135–146.

Bojar, O., Federmann, C., Fishel, M., Graham, Y., Haddow, B., Koehn, P., and Monz, C. (2018). Findings of the 2018 conference on machine translation (WMT18). In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers, WMT 2018, Belgium, Brussels, October 31 - November 1, 2018*, pages 272–303.

Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. (2015). A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 632–642.

Boyd, S. and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.

Cao, Z., Simon, T., Wei, S., and Sheikh, Y. (2017). Realtime multi-person 2d pose estimation using part affinity fields. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 1302–1310.

Carmona, V. I. S., Mitchell, J., and Riedel, S. (2018). Behavior analysis of NLI models: Uncovering the influence of three factors on robustness. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 1975–1985.

Caruana, R. (1993). Multitask learning: A knowledge-based source of inductive bias. In *Machine Learning, Proceedings of the Tenth International Conference, University of Massachusetts, Amherst, MA, USA, June 27-29, 1993*, pages 41–48.

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

Cer, D., Yang, Y., Kong, S., Hua, N., Limtiaco, N., John, R. S., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., Strope, B., and Kurzweil, R. (2018). Universal sentence encoder for english. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018: System Demonstrations, Brussels, Belgium, October 31 - November 4, 2018*, pages 169–174.

Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., Koehn, P., and Robinson, T. (2014). One billion word benchmark for measuring progress in statistical language modeling. In *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*, pages 2635–2639.

Chen, D. and Manning, C. D. (2014). A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical*

Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL, pages 740–750.

Chen, M., Xu, Z. E., Weinberger, K. Q., and Sha, F. (2012). Marginalized denoising autoencoders for domain adaptation. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*.

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734.

Choi, E., He, H., Iyyer, M., Yatskar, M., Yih, W., Choi, Y., Liang, P., and Zettlemoyer, L. (2018). Quac: Question answering in context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 2174–2184.

Chomsky, N. (1957). The structure of language.

Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., and LeCun, Y. (2015). The loss surfaces of multilayer networks. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2015, San Diego, California, USA, May 9-12, 2015*.

Cohn, T., Bird, S., Neubig, G., Adams, O., and Makarucha, A. J. (2017). Cross-lingual word embeddings for low-resource language modeling. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 1: Long Papers*, pages 937–947.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

Dai, A. M. and Le, Q. V. (2015). Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems 28: Annual Conference*

on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada, pages 3079–3087.

Daniely, A., Sabato, S., Ben-David, S., and Shalev-Shwartz, S. (2015). Multiclass learnability and the ERM principle. *Journal of Machine Learning Research*, 16:2377–2404.

Dasgupta, S., Padia, A., Maheshwari, G., Trivedi, P., and Lehmann, J. (2018). Formal ontology learning from english IS-A sentences. *CoRR*, abs/1802.03701.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Devlin, J., Zbib, R., Huang, Z., Lamar, T., Schwartz, R. M., and Makhoul, J. (2014). Fast and robust neural network joint models for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pages 1370–1380.

Dolan, W. B. and Brockett, C. (2005). Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing, IWP@IJCNLP 2005, Jeju Island, Korea, October 2005, 2005*.

Dozat, T. and Manning, C. D. (2017). Deep biaffine attention for neural dependency parsing. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Eggensperger, K., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2015). Efficient benchmarking of hyperparameter optimizers via surrogates. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 1114–1120.

Felbo, B., Mislove, A., Søgaard, A., Rahwan, I., and Lehmann, S. (2017). Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 1615–1625.

Firth, J. R. (1957). A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis*.

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139.

Gal, Y. and Ghahramani, Z. (2016). A theoretically grounded application of dropout in recurrent neural networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 1019–1027.

Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., and Lempitsky, V. S. (2016). Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 17:59:1–59:35.

Girshick, R. B., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pages 580–587.

Glockner, M., Shwartz, V., and Goldberg, Y. (2018). Breaking NLI systems with sentences that require simple lexical inferences. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers*, pages 650–655.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 513–520.

Goldberg, Y. (2016). A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420.

Goldberg, Y. (2017). *Neural Network Methods for Natural Language Processing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

Gong, C., He, D., Tan, X., Qin, T., Wang, L., and Liu, T. (2018). FRAGE: frequency-agnostic word representation. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 1341–1352.

Goodfellow, I. J., Bengio, Y., and Courville, A. C. (2016). *Deep Learning*. Adaptive computation and machine learning. MIT Press.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2672–2680.

Gulordava, K., Bojanowski, P., Grave, E., Linzen, T., and Baroni, M. (2018). Colorless green recurrent networks dream hierarchically. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 1195–1205.

Harris, Z. S. (1954). Distributional structure. *Word*, 10(2-3):146–162.

Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1).

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

Howard, J. and Ruder, S. (2018). Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 328–339.

Inan, H., Khosravi, K., and Socher, R. (2017). Tying word vectors and word classifiers: A loss framework for language modeling. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 448–456.

Jia, R. and Liang, P. (2017). Adversarial examples for evaluating reading comprehension systems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 2021–2031.

Kalchbrenner, N. and Blunsom, P. (2013). Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1700–1709.

Kemker, R., McClure, M., Abitino, A., Hayes, T. L., and Kanan, C. (2018). Measuring catastrophic forgetting in neural networks. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3390–3398.

Kifer, D., Ben-David, S., and Gehrke, J. (2004). Detecting change in data streams. In *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*, pages 180–191.

Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1746–1751.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526.

Kong, L., Schneider, N., Swayamdipta, S., Bhatia, A., Dyer, C., and Smith, N. A. (2014). A dependency parser for tweets. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1001–1012.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551.

LeCun, Y., Cortes, C., and Burges, C. (2010). Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2:18.

Lei, Q., Wu, L., Chen, P., Dimakis, A. G., Dhillon, I. S., and Witbrock, M. (2018). Discrete attacks and submodular optimization with applications to text classification. *CoRR*, abs/1812.00151.

Levy, O. and Goldberg, Y. (2014). Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 2: Short Papers*, pages 302–308.

Li, Y. and Yang, T. (2018). Word embedding for understanding natural language: A survey. In *Guide to Big Data Applications*, pages 83–104. Springer.

Linzen, T., Dupoux, E., and Goldberg, Y. (2016). Assessing the ability of lstms to learn syntax-sensitive dependencies. *TACL*, 4:521–535.

Liu, P. J., Saleh, M., Pot, E., Goodrich, B., Sepassi, R., Kaiser, L., and Shazeer, N. (2018). Generating wikipedia by summarizing long sequences. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

Liu, S., Yang, N., Li, M., and Zhou, M. (2014). A recursive recurrent neural network for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pages 1491–1500.

Loshchilov, I. and Hutter, F. (2017). SGDR: stochastic gradient descent with warm restarts. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Luong, T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In *Proceedings of the*

2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015, pages 1412–1421.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. (2011). Learning word vectors for sentiment analysis. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*, pages 142–150.

Mahajan, D., Girshick, R. B., Ramanathan, V., He, K., Paluri, M., Li, Y., Bharambe, A., and van der Maaten, L. (2018). Exploring the limits of weakly supervised pretraining. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part II*, pages 185–201.

Maheshwari, G., Trivedi, P., Lukovnikov, D., Chakraborty, N., Fischer, A., and Lehmann, J. (2018). Learning to rank query graphs for complex question answering over knowledge graphs. *CoRR*, abs/1811.01118.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330.

McCloskey, M. and Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier.

Merity, S., Keskar, N. S., and Socher, R. (2018). Regularizing and optimizing LSTM language models. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. (2017). Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 3111–3119.

Mou, L., Meng, Z., Yan, R., Li, G., Xu, Y., Zhang, L., and Jin, Z. (2016). How transferable are neural networks in NLP applications? In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 479–489.

Mou, L., Peng, H., Li, G., Xu, Y., Zhang, L., and Jin, Z. (2015). Discriminative neural sentence modeling by tree-based convolution. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 2315–2325.

Nguyen, T., Rosenberg, M., Song, X., Gao, J., Tiwary, S., Majumder, R., and Deng, L. (2016). MS MARCO: A human generated machine reading comprehension dataset. In *Proceedings of the Workshop on Cognitive Computation: Integrating neural and symbolic approaches 2016 co-located with the 30th Annual Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, December 9, 2016.*

Nguyen, T. Q. and Chiang, D. (2017). Transfer learning across low-resource, related languages for neural machine translation. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing, IJCNLP 2017, Taipei, Taiwan, November 27 - December 1, 2017, Volume 2: Short Papers*, pages 296–301.

Niehues, J. and Cho, E. (2017). Exploiting linguistic resources for neural machine translation using multi-task learning. In *Proceedings of the Second Conference on Machine Translation, WMT 2017, Copenhagen, Denmark, September 7-8, 2017*, pages 80–89.

Olah, C. (2015). Understanding lstm networks. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`. Last checked on May 04, 2019.

Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Trans. Knowl. Data Eng.*, 22(10):1345–1359.

Pang, B. and Lee, L. (2004). A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics, 21-26 July, 2004, Barcelona, Spain.*, pages 271–278.

Parikh, A. P., Täckström, O., Das, D., and Uszkoreit, J. (2016). A decomposable attention model for natural language inference. In *Proceedings*

of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016, pages 2249–2255.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543.

Peters, M. E., Ammar, W., Bhagavatula, C., and Power, R. (2017). Semi-supervised sequence tagging with bidirectional language models. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1756–1765.

Plank, B. and Alonso, H. M. (2017). When is multitask learning effective? semantic sequence prediction under varying data conditions. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 1: Long Papers*, pages 44–53.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners.

Rajpurkar, P., Jia, R., and Liang, P. (2018). Know what you don't know: Unanswerable questions for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers*, pages 784–789.

Reddy, S., Chen, D., and Manning, C. D. (2018). Coqa: A conversational question answering challenge. *CoRR*, abs/1808.07042.

Rei, M. (2017). Semi-supervised multitask learning for sequence labeling. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 2121–2130.

Ruder, S. (2018). NLP's Imagenet moment has arrived. `https://thegradient.pub/nlp-imagenet/`.

Ruder, S. (2019). *Neural Transfer Learning for Natural Language Processing*. PhD thesis, National University of Ireland, Galway.

Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

Schwenk, H., Barrault, L., Conneau, A., and LeCun, Y. (2017). Very deep convolutional networks for text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 1: Long Papers*, pages 1107–1116.

Shank, R. C. and Tesler, L. (1969). A conceptual dependency parser for natural language. In *Third International Conference on Computational Linguistics, COLING 1969, Stockholm, Sweden, September 1-4, 1969*.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 2960–2968.

Søgaard, A., Goldberg, Y., and Levy, O. (2017). A strong baseline for learning cross-lingual word embeddings from sentence alignments. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 1: Long Papers*, pages 765–774.

Stickland, A. C. and Murray, I. (2019). BERT and pals: Projected attention layers for efficient adaptation in multi-task learning. *CoRR*, abs/1902.02671.

Subramanian, S., Rajeswar, S., Dutil, F., Pal, C., and Courville, A. C. (2017). Adversarial generation of natural language. In *Proceedings of the 2nd Workshop on Representation Learning for NLP, Rep4NLP@ACL 2017, Vancouver, Canada, August 3, 2017*, pages 241–251.

Taylor, W. L. (1953). "cloze procedure": A new tool for measuring readability. *Journalism Bulletin*, 30(4):415–433.

Thorne, J., Vlachos, A., Christodoulopoulos, C., and Mittal, A. (2018). FEVER: a large-scale dataset for fact extraction and verification. In *Proceedings of the 2018 Conference of the North American Chapter of the*

Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers), pages 809–819.

Trivedi, P., Maheshwari, G., Dubey, M., and Lehmann, J. (2017). Lc-quad: A corpus for complex question answering over knowledge graphs. In *The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part II*, pages 210–218.

Vapnik, V. N. and Chervonenkis, A. Y. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2):264–280.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6000–6010.

Völske, M., Potthast, M., Syed, S., and Stein, B. (2017). Tl;dr: Mining reddit to learn automatic summarization. In *Proceedings of the Workshop on New Frontiers in Summarization, NFiS@EMNLP 2017, Copenhagen, Denmark, September 7, 2017*, pages 59–63.

Voorhees, E. M. and Tice, D. M. (1999). The TREC-8 question answering track evaluation. In *Proceedings of The Eighth Text REtrieval Conference, TREC 1999, Gaithersburg, Maryland, USA, November 17-19, 1999*.

Wan, L., Zeiler, M. D., Zhang, S., LeCun, Y., and Fergus, R. (2013). Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 1058–1066.

Warstadt, A., Singh, A., and Bowman, S. R. (2018). Neural network acceptability judgments. *CoRR*, abs/1805.12471.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3320–3328.

Yu, A. W., Dohan, D., Luong, M., Zhao, R., Chen, K., Norouzi, M., and Le, Q. V. (2018). Qanet: Combining local convolution with global self-attention for reading comprehension. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

Zhang, X., Zhao, J. J., and LeCun, Y. (2015). Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 649–657.

Zhao, H., Shi, J., Qi, X., Wang, X., and Jia, J. (2017). Pyramid scene parsing network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 6230–6239.

Zhou, G., Wu, J., Zhang, C., and Zhou, Z. (2016). Minimal gated unit for recurrent neural networks. *CoRR*, abs/1603.09420.

Zhu, Y., Kiros, R., Zemel, R. S., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 19–27.

Zhu, Z., Bernhard, D., and Gurevych, I. (2010). A monolingual tree-based translation model for sentence simplification. In *COLING 2010, 23rd International Conference on Computational Linguistics, Proceedings of the Conference, 23-27 August 2010, Beijing, China*, pages 1353–1361.