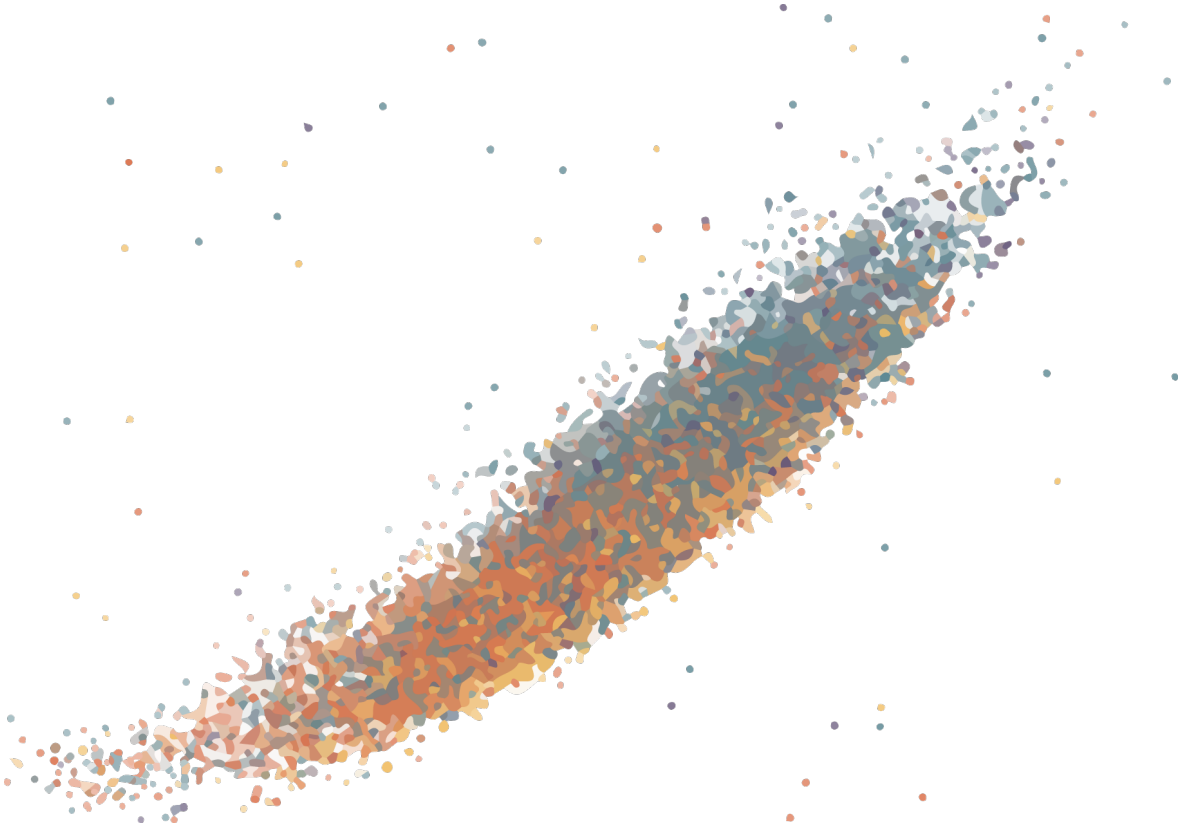


Domain Adapted Language Models

Tailoring BERT for Specialist Language Domains

Master's Thesis in Computer Science, Algorithms, Languages and Logic

Erik Jansson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's Thesis 2019:93

Domain Adapted Language Models

Tailoring BERT for Specialist Language Domains

Erik Jansson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Division of Data Science and AI
Chalmers University of Technology
Gothenburg, Sweden 2019

Domain Adapted Language Models
Tailoring BERT for Specialist Language Domains
Erik Jansson

Academic Supervisor:
Richard Johansson, Department of Computer Science and Engineering
Industry Supervisor:
Aron Lagerberg, Seal Software
Examiner:
Morteza Haghir Chehreghani, Department of Computer Science and Engineering

Master's Thesis 2019:93
Department of Computer Science and Engineering
Division of Data Science and AI
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Four legal provision classes as represented by output corresponding to BERT's `[CLS]` token. Visualization performed with principal component analysis.

Typeset in LaTeX
Gothenburg, Sweden 2019

Domain Adapted Language Models
Tailoring BERT for Specialist Language Domains
Erik Jansson
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

BERT is a recent neural network model that has proven itself a massive leap forward in natural language processing. Due to the tedious training required by this massive model, a pretrained BERT instance has been released as a high-performing starting point for further training on downstream tasks. The pretrained model has been trained on general English text and may not be optimal for applications in specialist language domains. This study examines adapting the pretrained BERT model to the specialist language domain of legal text, with classification as the downstream task of interest. The study finds that domain adaptation is most beneficial if faced with small task-specific datasets, where performance can approach that of a model pretrained from scratch on legal text data. The study further presents practical guidelines for applying BERT in specialist language domains.

Keywords: natural language processing, BERT, transformer, domain adaptation, language model, classification

Acknowledgements

The author would like to thank Seal Sofware for the opportunity to conduct this study, the warm working environment, and for their crazy generosity with computing power. In particular, the author would like to thank Aron Lagerberg for the many enlightening discussions and for his help in setting up the legal pretraining procedure. Furthermore, Richard Johansson is thanked for his encouragement and thorough feedback throughout the study. The author also thanks the Freges study group. Last but not least, the author would like to thank the team at HuggingFace for their implementation of BERT in `pytorch`, which was both easy to work with and simple to extend.

Erik Jansson, Gothenburg, June 2019

Contents

List of Figures

List of Tables

1

Introduction

Constructing machine learning models is a difficult task in the field of natural language processing. These models rely on mathematical data representations which are challenging to construct with regards to human language. Natural language is rife with ambiguity, hidden meanings, long-term dependencies, idioms and other complex constructions that are difficult to capture mathematically. Furthermore, the relationship between these mathematical representations and a given natural language processing task can be even more obscure.

Recent studies have seen great success using so-called *neural language models* for modeling and representing language (Howard and Ruder, 2018; Peters et al., 2018; Radford, Narasimhan, Salimans and Sutskever, 2018). These language models uncover structure in language by training on some natural language task. Once this *pretraining* procedure is completed, these models can then be used as starting points for further training on other language processing tasks. By leveraging the linguistic structures uncovered by the initial language model pretraining, subsequent modeling can focus on learning task-specific attributes, not the language itself. This procedure is similar to work done in computer vision (He, Zhang, Ren and Sun, 2016; Simonyan and Zisserman, 2014) and is an example of *transfer learning* - the act of training a model on a task that provides a beneficial starting point for further training.

The natural language problem space suffers from the usual lack of labeled data that is common in many forms of data analysis. Besides providing a starting point for natural language machine learning models, most language model pretraining procedures have the added benefit of not actually requiring labeled training data. It is therefore possible to reap the benefits of a language model despite having access to very little task-specific labeled data.

There has been a recent surge of research that further builds on creating powerful language models. One of the more prominent studies presented Bidirectional Encoder Representations from Transformers, or *BERT* (Devlin, Chang, Lee and Toutanova, 2019). The authors of the paper have tested the model on industry-standard benchmarking datasets and many different natural language processing tasks. These tests are evidence of state-of-the-art performance improvements and show that BERT is well-adapted to a wide array of tasks (Devlin, Chang, Lee and Toutanova, 2019).

While the performed benchmarking tests offer a well-rounded assessment of model capabilities, this evaluation strategy favors generalist models that capture general language knowledge. If a natural language processing task deals with text in a specialist domain, these models, often pretrained on publicly available text such as Wikipedia articles, may not be the ideal solution for specialist language. In other words, the pretrained models may lack knowledge specific to the specialist domain since it is not found in the pretraining dataset.

In the case of the company Seal Software, the machine learning task of interest is classification in the domain of legal text. Seal Software uses text classification models to identify relevant portions of legal contracts. These portions, called *provisions*, are the building blocks of legal contracts. Finding and identifying relevant provisions is crucial in analyzing large volumes of lengthy contracts. Put simply, the classification task takes a series of contracts as input. It is then the goal of the classifier to analyze these contracts and flag provisions of the sought-after type. Identifying these relevant provisions in a given document is often an extremely imbalanced classification task, with only a few examples of a given provision type. It is thus of utmost importance that the few available annotated examples of data are put to good use in training. By using a pretrained language model, the classification training can be given a linguistic head start.

This study aims to evaluate the state-of-the-art language model, BERT, on the task of classification in the domain of English legal documentation. Furthermore, different ways of tailoring the pretrained model to this specific language domain will be explored, attempting to bridge the disconnect between the specialist language domain of legal text and the language domain on which BERT has been pretrained. The effects of tailoring the pretrained BERT model to the relevant specialist domain, through so-called domain adaptation, will be assessed on the downstream task of legal text classification. The ultimate goal is to describe a procedure in which the pretrained BERT model can be adapted to a specific domain.

1.1 Aim

The purpose of this project is to assess the performance of the pretrained BERT language model when used for legal text classification. Furthermore, the project aims to explore how the pretrained model can be tailored to this specific language domain.

This is interesting for a number of reasons. Firstly, in contrast to the original paper, this study focuses specifically on how BERT deals with classification. Secondly, no other studies have been conducted which specifically focus on the domain adaptability of the pretrained BERT model. Thirdly, the study will measure the pretrained model's performance when dealing specifically with specialist literature - a language category in which many important applications reside. Using a pretrained BERT model in the domain of legal specialist text (often colloquially called "legalese") may result in suboptimal classification performance. This could happen because the

model was trained on non-specialist literature data, and may therefore not capture the relevant legal vernacular. In other words, the linguistic structures inferred by the pretrained model may not effectively translate to this specialist domain.

BERT was originally trained on non-specialist literature. This study aims to answer the following questions,

- How does the pretrained BERT language model perform when applied to specialist language text classification?
- How can the pretrained BERT model be tailored for text classification on specialist literature?
- How does the amount of available task data affect BERT performance?

1.2 Limitations

The study is limited to evaluating the performance of BERT, its pretrained and domain adapted variants, and will therefore not focus on constructing and tuning complex classification networks. Extending the project to include adapting classifiers would both increase the complexity of the project, and make it difficult to isolate whether performance changes stem from the language model, the classifier or an interaction between the two. By keeping classifiers as simple as possible, the downstream performance can be more effectively attributed to the language model. It is also worth mentioning that a good language model should circumvent the need for highly complex classification networks.

2

Related Work

A flurry of further research has been conducted on BERT in the short time since the model was presented. For instance, custom pretrained models have been released for various specialist language domains, and the pretraining procedures have been further explored. The transformer neural network architecture on which BERT is based has also been examined. The new research and its relation to this study are now presented.

There have been previous attempts at leveraging BERT for specialist language domains. For instance, the models SciBERT and BioBERT were recently released and showed significant performance improvements on scientific and biomedical language tasks (Beltagy, Cohan and Lo, 2019; Lee et al., 2019). The performance boosts from these two models came from rerunning pretraining on large domain-specific corpora. This is analogous to the custom legal pretraining procedure presented in Section 4.6. Unlike this study however, the SciBERT and BioBERT papers do not touch on leveraging the preexisting knowledge in the pretrained BERT model. Instead, BioBERT and SciBERT are retrained models, not domain adaptions of the pretrained BERT.

Other studies have showcased the multi-task learning capabilities of BERT. Training BERT (or closely related architectures) on multiple tasks at once can yield performance gains (Stickland and Murray, 2019; Liu, He, Chen and Gao, 2019). These papers describe training BERT on more tasks than those normally used for model pretraining, and find that additional training tasks aid BERT in its natural language processing capabilities. In other words, it has been experimentally verified that adding training tasks to BERT can have performance-boosting effects on downstream tasks. This hints at the promise of adding tasks for domain adaptation such as the domain invariance training procedure introduced in Section 4.4.

BERT is heavily based on a neural network architecture called the transformer (see Section 3.5.1 for a detailed explanation of this architecture). Further studies have empirically investigated training procedures for transformer-based network architectures. One such study presented a number of findings particularly pertinent to this study (Popel and Bojar, 2018). Of the many practical findings presented, one is particularly relevant: transformers (like many other large neural network models) exhibit a reluctance to overfit when training on moderately sized datasets. This means that although the training sets used in this study are not as large as those in the original paper, overfitting is unlikely. In other words, performance may still

increase on these smaller datasets, even after extended training. Furthermore, the study finds that transformers are unlikely to converge on moderately sized datasets. Therefore, performance gains may still be seen when continuing the original pretraining procedure.

In summary, recent research has demonstrated that there is interest in domain-specific BERT models. However, many of the presented domain-specific models eschew the pretrained model's preexisting linguistic knowledge in favor of training from scratch. Recent research has also borne witness to BERT's potential for multi-task learning, suggesting that expanding training procedures can increase model performance. Lastly, transformer-based models have been found to be highly trainable, exhibiting few tendencies to overfit. This attribute may suggest that, above all, BERT will continue to glean knowledge from most textual data.

3

Theory

To begin discussing domain adapting the pretrained BERT model for legal text, basic notions of machine learning, such as neural network models and classification must be explained. This chapter will discuss these topics, and also explain the inner workings of BERT as well as domain adaptation methods that have been previously studied. However, first of all, basic notation conventions must be established.

3.1 General Machine Learning and Notation

A very general definition of machine learning is to learn a task without explicit instruction. In *supervised* machine learning, this typically means defining a model which learns by example. Given enough examples and clever training algorithms, a supervised machine learning model can learn rules which generalize to data it has not yet been exposed to.

The aforementioned examples are sampled from an underlying *feature space*, denoted $\mathcal{X}$, which is the space of all possible input data values. Sets of sampled data points are denoted X. Not all data are equally likely to be sampled, and this is specified by the *underlying data distribution*, denoted $P(X)$. This distribution specifies how likely it is to sample the given data from the feature space, and is typically unknown in practical applications. Together, these two entities define the problem *domain*, $\mathcal{D} = \{\mathcal{X}, P(X)\}$.

In supervised machine learning, the goal is usually to find some rule which maps input to some form of output. This output can take any form in the relevant *output space*, $\mathcal{Y}$. Output exhibits stochasticity similarly to the input feature data, however the output is conditioned on the input, $P(Y|X)$. The output space and the output conditional probability make up the *task*, $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$.

With the task and domain now defined, we can more concretely express supervised machine learning as learning the conditional probability of output given input, based on samples from the output and feature spaces. In other words, the probabilistic relation between input and output, $P(Y|X)$, is to be estimated from a finite sample dataset $D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$, where $\boldsymbol{x}_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. The estimated relationship is to be found among all possible input-to-output mappings in the *hypothesis space* $\mathcal{H}$. This relationship must be found such that it generalizes to data not necessarily

present in the dataset. Put differently, given a previously unseen datapoint $\boldsymbol{x}$, the model should be able to estimate some output $\hat{y}$ which lies close to the true output y according to some output quality measure. Note that these estimated outputs, as well as any other estimated quantities, are decorated with the hat symbol ˆ.

3.2 Classification

A classification problem involves grouping a number of data points into a predetermined set of categories or *classes*. The aim is to construct a classification rule or model that can correctly assign each data point $\boldsymbol{x}$ a label $\hat{y}$ from the set of available classes $\mathcal{Y}$. Classification is a supervised machine learning task, meaning that labeled data is used to construct the classification rule. Labeled data is a dataset $D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$ in which each data point has been pre-assigned a true label $y \in \mathcal{Y}$. This labeled dataset functions as an answer key when constructing the classification model and (hopefully) allows a classification rule to be constructed by example. The correctness of the final classifier is determined by how close it is to labeling data points with their true labels.

When constructing a classification model, care must be taken to ensure that the model is not simply memorizing the given labeled dataset. One such example of a memorizing classifier would be to take the entire dataset and simply map each data point to its given true label. Since the given dataset is finite in practice, such a mapping-based classification rule would fail to predict the labels of new data points not present in the initial dataset. This is an extreme example of adhering too closely to the given dataset and is known as *overtraining* or *overfitting* (see Figure 3.1). Overtraining causes models to be ill-prepared for unseen future data, thus exhibiting poor *generalization*.

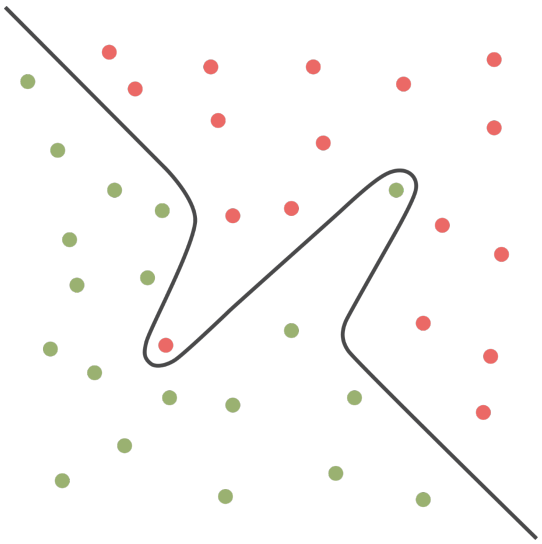


Figure 3.1: An example of an overtrained classifier. This classifier appears to adhere too strictly to the two noisy data points in the two classes.

Overtraining can also result in poor generalization in the presence of noisy data. A data point's true value may have been perturbed by some form of random noise $\hat{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{\epsilon}$ (where $\boldsymbol{\epsilon} \sim \boldsymbol{P}$, where $\boldsymbol{P}$ is some statistical distribution). This noise can be the result of a non-perfect data measurement, a human error made during the dataset construction or any number of random phenomena. Noise in the labeling of the given dataset can also be the result of a manual labeling procedure. In this noisy scenario, the true label is not a class assignment on the data point, but instead *a class assignment on the data as combined with noise*. Blindly assuming that all data points and labels are noiseless may also result in an overtrained classifier. This overtrained classifier will have attempted to learn noise as part of the data, ultimately leading to poor generalization on future data.

In order to diagnose a poorly generalizing model, the labeled dataset is commonly divided into two non-overlapping partitions, $D = \{D_{train}, D_{test}\}$. The training data D_{train} is used to learn some classification rule. The knowledge that the classifier has gained during this training process is then evaluated on the test data, D_{test} by attempting to predict a label $\hat{y}$ for each data point $\boldsymbol{x} \in X_{test}$. The classifier's performance is judged by its tendency stray from the true labels in $\boldsymbol{y}$. In other words, the performance is a measurement of the degree in which the predicted labels $\hat{\boldsymbol{y}}$ differ from the true labels $\boldsymbol{y}$.

There are countless metrics that can be used to quantify classifier performance. The most straightforward metric is simply calculated as the fraction of data points that a classifier correctly labels,[1]

$$\frac{\sum_i^n \mathbb{I}_{\hat{y}_i = y_i}}{N}$$

where N is the number of data points in the evaluation dataset.

This metric, known as *accuracy*, can be misleading in classification problems that exhibit classes of vastly different sizes. For instance, given a dataset consisting of two classes of disparate sizes, a naive classifier that always labels points as belonging to the larger class will only misclassify a small fraction of the data points. This classifier will garner a favorable accuracy metric while being utterly useless. Other metrics are therefore required when dealing with these *imbalanced* classification problems.

In this case, *precision* and *recall* are two metrics that are more appropriate for classifier evaluation in an imbalanced setting (see Figure 3.2 for visualization). Consider a binary classification problem where the goal is to assign data points to two classes, denoted as positive and negative. In this binary scenario, precision and recall are defined as,

$$P = \frac{\text{true positives}}{\text{true positives + false positives}} \qquad R = \frac{\text{true positives}}{\text{true positives + false negatives}}$$

where *true positives*, *false positives* and *false negatives* are defined as,

$$tp = \sum_i^n \mathbb{I}_{\hat{y}_i = +} \cdot \mathbb{I}_{y_i = +} \qquad fp = \sum_i^n \mathbb{I}_{\hat{y}_i = +} \cdot \mathbb{I}_{y_i = -} \qquad fn = \sum_i^n \mathbb{I}_{\hat{y}_i = -} \cdot \mathbb{I}_{y_i = +}$$

[1] Let $\mathbb{I}$ be the indicator function defined as $\mathbb{I}_{condition} = \begin{cases} 1, & \text{if } condition \text{ true} \\ 0, & \text{if } condition \text{ false} \end{cases}$

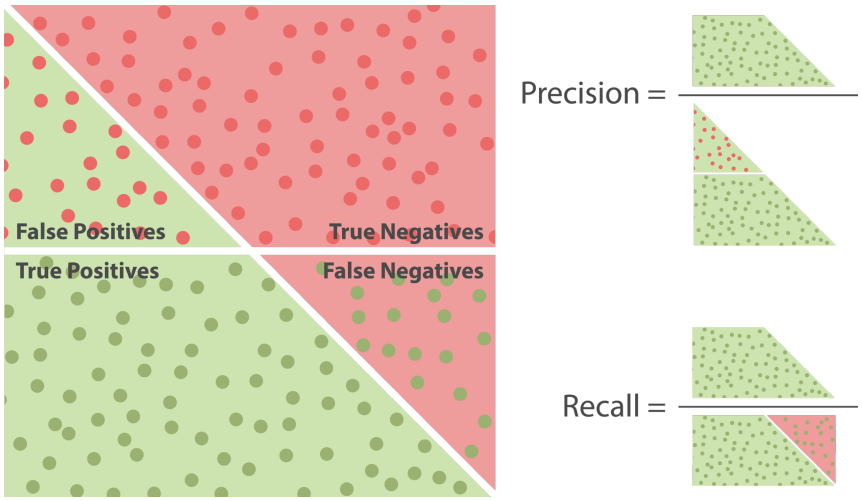


Figure 3.2: *Left:* A binary classification setting with a non-perfect classifier. The red and green points are sample data, while the background color shows the model's classification rule. *Right:* Visual definitions of precision and recall

The above metrics give a more nuanced picture of classifier performance. However, it would be easier to compare classifiers on the basis of only one single metric. This leads to the aggregation of precision and recall known as the F_1 *score*. This is defined as,

$$F_1 = \frac{2PR}{P+R}$$

Precision, recall and F_1 scores are not limited to binary classification problems. In a multiclass setting, precision, recall and F_1 scores are calculated once for each class. Consider a multiclass calculation of precision, recall and F_1 score for one of the classes, class C. To do this, the true positives, false positives and false negatives must first be calculated. In the case of class C, this becomes,

$$tp_C = \sum_i^n \mathbb{I}_{\hat{y}_i=C} \cdot \mathbb{I}_{y_i=C} \qquad fp_C = \sum_i^n \mathbb{I}_{\hat{y}_i=C} \cdot \mathbb{I}_{y_i\neq C} \qquad fn_C = \sum_i^n \mathbb{I}_{\hat{y}_i\neq C} \cdot \mathbb{I}_{y_i=C}$$

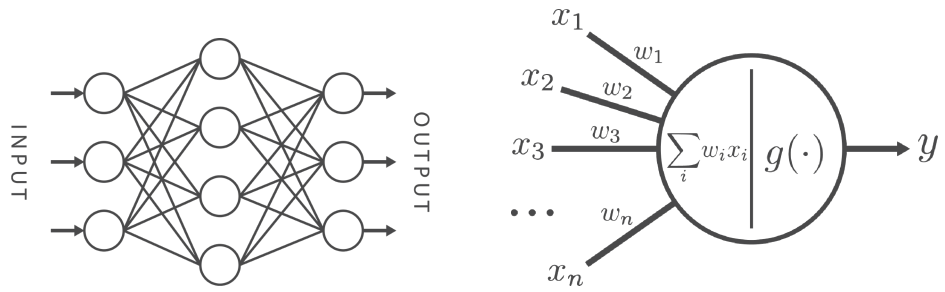
From these values, the calculations for the class-specific precision, recall and F_1 score values follow analogously to the binary classification case.

By using the above metrics, a classification model's ability to generalize on future data can be empirically estimated with the aforementioned partitioning of data into training and testing subsets. The training and evaluation procedures as well as the above metrics will be used throughout the study.

3.3 Neural Networks for Classification

Artificial neural networks are a family of machine learning models that have seen a rush of research in recent years. Neural networks are graph-like structures consisting

of nodes, or *neurons*, and connections between neurons, or *edges*. These neurons are commonly organized into layers, with edges running between layers (see Figure 3.3a). Numerically represented input data is fed into the network at the input layer, propagates through the network via the edges and results in some output signal at an output layer. The process of input propagating through the network is called a *forward pass*. During a forward pass, each neuron typically applies some function to a weighted sum of its input (see Figure 3.3b). The power of this seemingly simple concept lies in its adaptability. There are countless ways to connect and construct neurons and any number of functions can be applied at each neuron. There are also many different training strategies.



(a) A schematic of a basic neural network with one hidden layer between its input and output layers

(b) A close-up view of a single neuron, which outputs a non-linear transformation of a weighted sum of its inputs

Figure 3.3: Schematic overviews of neural networks

The final neural network layer often performs some kind of normalization. For instance, in a classification setting with K different classes, the output layer is typically a vector in K dimensions where element i corresponds to the probability of the input belonging to class i. To construct valid probabilities the final layer applies the *softmax* function to each element. This function transforms the final layer output, $\boldsymbol{z}$ into a set of valid probabilities which sum to 1. The softmax function is defined as,

$$\text{softmax}(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$

3.3.1 Loss

As previously mentioned, neurons perform a weighted summation of incoming signals. A neural network learns by adjusting these edge weights (and sometimes other parameters) to better approximate the desired output, thereby adapting to the task at hand. In classification learning, the output of the network, $\hat{y}$ is compared to the desired output y. The difference between these two outputs is called the *loss*. It is possible to calculate the manner in which model parameters should be changed

in order to decrease this loss. The process of calculating loss and adjusting model parameters will now be more formally explained.

In order to adapt the model to minimize loss, the loss must first be quantitatively calculated. This is done using a so-called *loss function*. Many different varieties of loss functions exist, where one of the more commonly used in classification is *cross-entropy*. Cross-entropy is defined as,

$$H(p, q) = -\sum_{x \in X} p(x) \log q(x)$$

where p is the true distribution, q is the model distribution and x is some input. In a multiclass classification setting with K classes, cross-entropy becomes,[2]

$$\begin{aligned} H(p, q) &= -\sum_{x_i \in X} \sum_{k=1}^{K} p(y_i = k) \log q(\hat{y}_i = k) \\ &= -\sum_{x_i \in X} \sum_{k=1}^{K} p(y_i = k) \log \hat{y}_{ik} \end{aligned}$$

Unfortunately, the true underlying distribution of text classes, p, is not typically known in language classification. For this reason, the cross-entropy loss must be empirically estimated from the dataset. The obvious way to do this is to set the probability of a point belonging to the true label to $1/|X|$, and the probability of all other classes to 0. This can be formulated with an indicator function,

$$H(X, q) = -\sum_{x_i \in X} \sum_{k=1}^{K} \frac{\mathbb{I}_{y_i = k}}{|X|} \log \hat{y}_{ik}$$

The use of this loss and its relation to classification may not be immediately clear. To motivate the use of cross-entropy, consider the joint likelihood of the training data,

$$\mathcal{L} = \prod_{i=1}^{N} \prod_{k=1}^{K} \hat{y}_{ik}^{\mathbb{I}_{y_i = k}}$$

Taking the logarithm of this likelihood yields,

$$\log \mathcal{L} = \sum_{i=1}^{N} \sum_{k=1}^{K} \mathbb{I}_{y_i = k} \log \hat{y}_{ik} = -H(X, q) \cdot |X|$$

Therefore, it is apparent that minimizing the cross entropy is equivalent to maximizing the log likelihood of the model distribution in a classification setting (Bishop, 2006). The cross-entropy loss is then used to determine how the model must be updated to decrease the loss.

[2]For a model which applies the softmax function to the classification output, the model probability of the datapoint x_i belonging to class k, or $q(y_i = k)$, is simply $\hat{y}_{ik}$ - the model output at position k.

3.3.2 Gradient Descent and Backpropagation

Once the loss has been calculated, the model parameters must be adapted to decrease it. There are many optimization strategies that strive to decrease a chosen loss function. One such strategy is *gradient descent*. This optimization strategy aims to jointly find each model parameter's effect on the final loss through a process called *backpropagation* (Hinton, Rumelhart and Williams, 1986). Once each parameter's influence on the loss has been determined, the next step in gradient descent is to adjust each parameter value by a small amount to (hopefully) decrease the loss. This is an iterative procedure that continues until some stopping criteria is reached. This optimization process will now be explained in more detail.

The initial step of gradient descent is to calculate the rate of change of the loss with respect to the model parameters. The resulting vector, known as the *gradient*, expresses how the loss function changes as a result of changes in model parameter values. In other words, the gradient is a multivariate generalization of the derivative. This gradient can then be used to determine how to descend the loss function surface to reach a model with smaller loss. Given model parameters $\boldsymbol{\theta}$, the gradient of the loss $\mathcal{L}$ is defined as,

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial \theta_1}, \frac{\partial \mathcal{L}}{\partial \theta_2}, \dots \right]$$

The loss function increases fastest in the direction of the gradient. It therefore decreases fastest in the direction of the negative gradient (Goodfellow, Bengio and Courville, 2016). The gradient descent update of model parameters is therefore expressed as,

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \, \nabla_{\boldsymbol{\theta}} \mathcal{L}$$

where η is the step size, or *learning rate*, which dictates the magnitude of the parameter update.

Thanks to the mathematically simple building blocks of neural networks, the gradient can be calculated with the backpropagation procedure (Hinton, Rumelhart and Williams, 1986). Backpropagation relies on the calculus chain rule,

$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \cdot \frac{\partial g(x)}{\partial x}$$

to recursively calculate how model parameters deep within the network affect the final loss. This is easiest explained with an example.

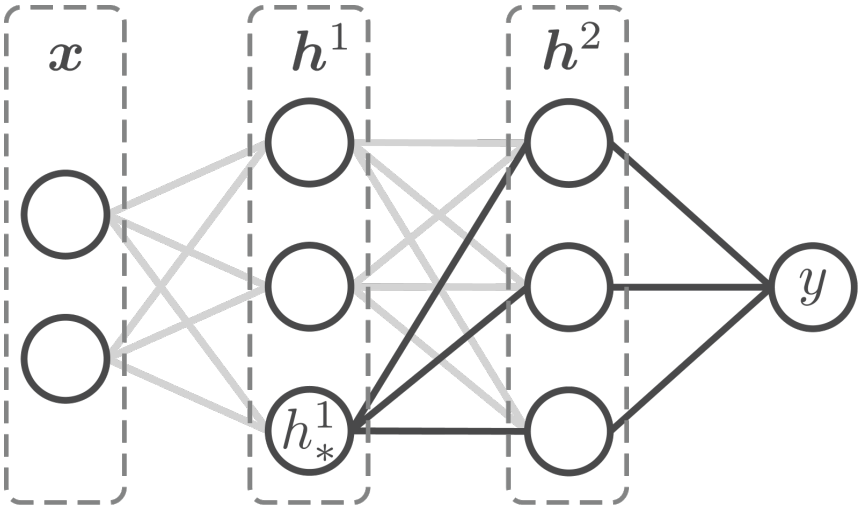


Figure 3.4: A simple neural network. Inputs and output are denoted by $\boldsymbol{x}$ and y respectively. Hidden layer parameters are denoted with $\boldsymbol{h}^L$ where L denotes the model layer. An arbitrary model parameter h^1_* is highlighted in the figure. Note that the connection weights normally found in neural networks have been omitted for simplicity.

Calculating the gradient means deriving the loss function with respect to all model parameters. In Figure 3.4, one of these parameters is h^1_*. Calculating the loss gradient therefore involves calculating $\frac{\partial \mathcal{L}}{\partial h^1_*}$. In backpropagation, the calculus chain rule is used to recursively define $\frac{\partial \mathcal{L}}{\partial h^1_*}$ in terms of other partial derivatives calculated earlier in the backpropagation procedure. For model parameters h^1_*, this becomes,

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial h^1_*} &= \frac{\partial \mathcal{L}}{\partial y} \cdot \frac{\partial y}{\partial h^1_*} \\ &= \frac{\partial \mathcal{L}}{\partial y} \cdot \sum \frac{\partial y}{\partial h^2_i} \frac{\partial h^2_i}{h^1_*}\end{aligned}$$

This recursive definition allows efficient calculation of all parameters' influence on the final loss. [3]

To form a complete picture of the model's performance on the training dataset, the loss would ideally be calculated over the entire dataset. The resulting loss would be,

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \mathcal{L}(\boldsymbol{x}_i, y_i, \boldsymbol{\theta})$$

Backpropagating on this loss would require backpropagating separately over each of the N individual loss terms. This process would both be computationally tedious

[3] In practice, backpropagation is not done analytically as described above. Modern neural network libraries such as pytorch save each node's numerically calculated gradient during the forward pass. During back propagation, the final loss vector is calculated from these node-specific gradient vectors. For a model with many nodes, this saving procedure can use immense amounts of memory.

and place heavy requirements on memory. Therefore, calculating the loss over an entire training dataset is usually not a feasible option.

It is important to realize that the training data are random variables sampled from some underlying data distribution. Calculating the loss gradient over this finite training set is therefore an attempt at estimating the expected loss gradient via a large sample set. While a large sample set will usually yield a better estimation of this expected gradient, there are no hard restrictions on using smaller sample sets to perform this estimation. This is the rationale behind using *batches* in neural network training. Batches are small sample sets drawn from the full training data. The loss is calculated on one batch at a time and backpropagated to facilitate model learning. This batch-wise procedure strikes a balance between a representative estimation of the loss gradient, while keeping model learning computationally feasible.

Since the advent of gradient descent, many different adaptations of the base algorithm have been proposed. One of the more recent and successful innovations is that of *Adam*. Adam is an optimization algorithm in the gradient descent family which uses an adaptive learning rate. Adam's learning rate adapts to each parameter individually. Furthermore, the learning rate is also affected by the its magnitude during previous iterations (Kingma and Ba, 2015).

3.4 Transfer Learning and Language Models

In typical machine learning scenarios, a model is trained on the same task it is meant to perform. In other words, a model with no prior exposure to any training data learns from data that is directly related to the task of interest. In transfer learning, a model is instead trained on a *source task* in order to improve performance on a *target task*.

Transfer learning is more concretely defined as follows. In transfer learning, a target domain $\mathcal{D}_T$ and target task $\mathcal{T}_T$, and a source domain $\mathcal{D}_S$ and source task $\mathcal{T}_S$ are given. The aim is to learn $\mathcal{T}_S$ such that performance on $\mathcal{T}_T$ is improved. There is also the condition that either $\mathcal{D}_T \neq \mathcal{D}_S$ or $\mathcal{T}_T \neq \mathcal{T}_S$ (or else no transfer would be necessary[4]) (Pan and Yang, 2010). Recalling the definitions of domain and task[5], in transfer learning, a model first estimates $P_S(Y|X)$ in the hope that this makes it easier to estimate $P_T(Y|X)$ (Torrey and Shavlik, 2010). Put slightly more formally, learning the source task can bias the target task learning or change the target task hypothesis space (Baxter, 2000) (see Figure 3.5).

[4] In a "normal" machine learning scenario $\mathcal{D}_T = \mathcal{D}_S$ and $\mathcal{T}_T = \mathcal{T}_S$

[5] $\mathcal{D} = \{\mathcal{X}, P(X)\}$; $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$ (see Section 3.1)

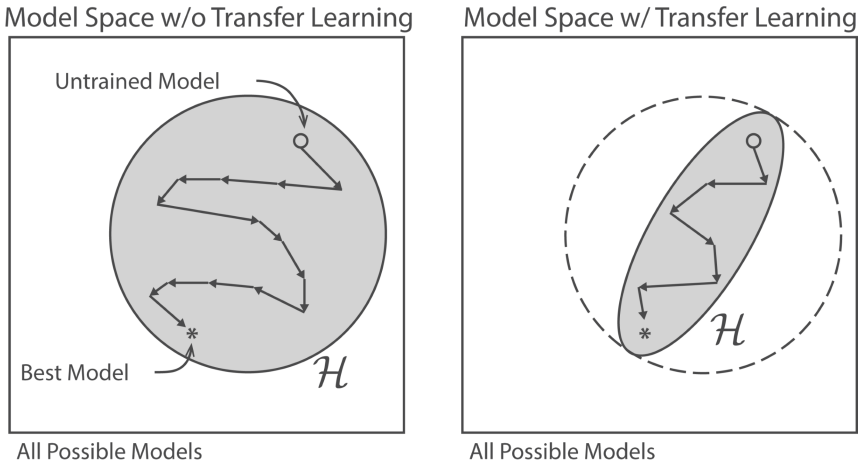


Figure 3.5: The search for the best possible model in the hypothesis space can be shortened through transfer learning.

In the best of cases, this leads to faster target task training and increased performance. For instance, model performance on a target task may be limited by the size of the available labeled dataset. In this situation, transfer learning can lend the model a helping hand by first pretraining on some source task for which there is plenty of data. This *pretraining* induces a hypothesis space wherein the target task is easier to learn, thus resulting in improved performance as compared to non-pretrained models. The pretraining also allows models to reach higher performance with less data and fewer training steps, thereby alleviating the data scarcity problem (Howard and Ruder, 2018). Some pretrained models have the added benefit of being well-suited as starting points for a number of different target tasks, thus acting as general off-the-shelf transfer learning solutions (Howard and Ruder, 2018; Devlin, Chang, Lee and Toutanova, 2019). A single pretraining procedure can therefore potentially yield rewards in numerous downstream tasks.

Since the aim of transfer learning is to facilitate knowledge transfer between source and target tasks, the tasks are usually somewhat related. Unfortunately, no rigorous theories exist that can point out the exact source task that should be learned to improve model performance on a given target task. However, beneficial source-target relationships do exhibit certain tendencies that can be exploited for effective transfer learning.

First, recall that a prerequisite for transfer learning is that either $\mathcal{D}_T \neq \mathcal{D}_S$ or $\mathcal{T}_T \neq \mathcal{T}_S$ hold. This implies that at least one of the following is true [6],

1 $\mathcal{X}_T \neq \mathcal{X}_S$
The feature spaces are not equal, meaning that data can take different shapes in the two domains

2 $P_T(X) \neq P_S(X)$

[6]Note that if inequality 1 holds, then inequality 2 is ill-posed. Likewise, if inequalities 1 or 3 hold, inequality 4 is ill-posed.

The underlying marginal distributions of features are not equal

3 $\mathcal{Y}_T \neq \mathcal{Y}_S$
The output spaces are not equal

4 $P_T(Y|X) \neq P_S(Y|X)$
The conditional distributions of the output given data are not equal

Put simply, beneficial transfer learning tends to occur when there is some form of similarity between target and source settings. In other words, transfer between source and target tasks is *positive* when the left and right hand sides of the above four points are at least somewhat related (Torrey and Shavlik, 2010).

Modern transfer learning for natural language processing uses so-called *language models*. Successful language models are commonly trained using language modeling tasks like *next word prediction* (Howard and Ruder, 2018; Devlin, Chang, Lee and Toutanova, 2019; Tenney, Das and Pavlick, 2019). In next word prediction, the model is fed a truncated sentence with the goal of predicting the next word. Studies show that learning this task captures many aspects of language and can induce a hypothesis space that is well-suited for many linguistic target tasks (Baxter, 2000; Howard and Ruder, 2018). Once the language modelling task has been adequately trained, the language model architecture is typically modified in order to train on the downstream target task.

It should be noted that the size of corpus used to train the source task affects the performance of the language model on downstream tasks. Pretraining on a larger corpus exposes the language model to more data, thus imbuing the model with increased linguistic knowledge (Moore and Lewis, 2010).

3.5 BERT

Bidirectional Encoder Representations from Transformers or *BERT* is an example of how transfer learning and language models can be highly effective in natural language processing. BERT is a recent neural network model whose efficacy has been showcased on a number of industry-standard natural language benchmarking tests (Devlin, Chang, Lee and Toutanova, 2019). The BERT language model is pretrained on a large unlabeled corpus. Once this initial language model pretraining has been performed, BERT can be further trained for a number of downstream natural language tasks with minimal architectural changes. This task-specific training, known as *finetuning*, typically requires less data and less training than the pretraining phase (Devlin, Chang, Lee and Toutanova, 2019). In other words, BERT is not only powerful, but also easily adaptable to new tasks once the initial language modeling has been performed.

BERT has achieved its landmark results by utilizing the encoder from the *transformer* architecture and by pretraining on two novel language modeling tasks - *masked language modeling* and *next sentence prediction*. These language modeling tasks and the transformer architecture will now be explained.

3.5.1 The Transformer Architecture

Language is contextually dependent. This means certain words are more likely than others to appear in a given context. In the past, this contextual dependency has been modelled with the help of recurrent neural network architectures. The recurrent architectures are associated with a number of disadvantages compared to non-recurrent models. For instance, recurrent networks are difficult to parallellize, leading to lengthy training times (Vaswani et al., 2017) and also have difficulty tracking dependencies between words that are spread far apart in the input (Lin et al., 2017). In summary, although recurrent models have previously been the foundation of progress in natural language processing, they are not without flaws.

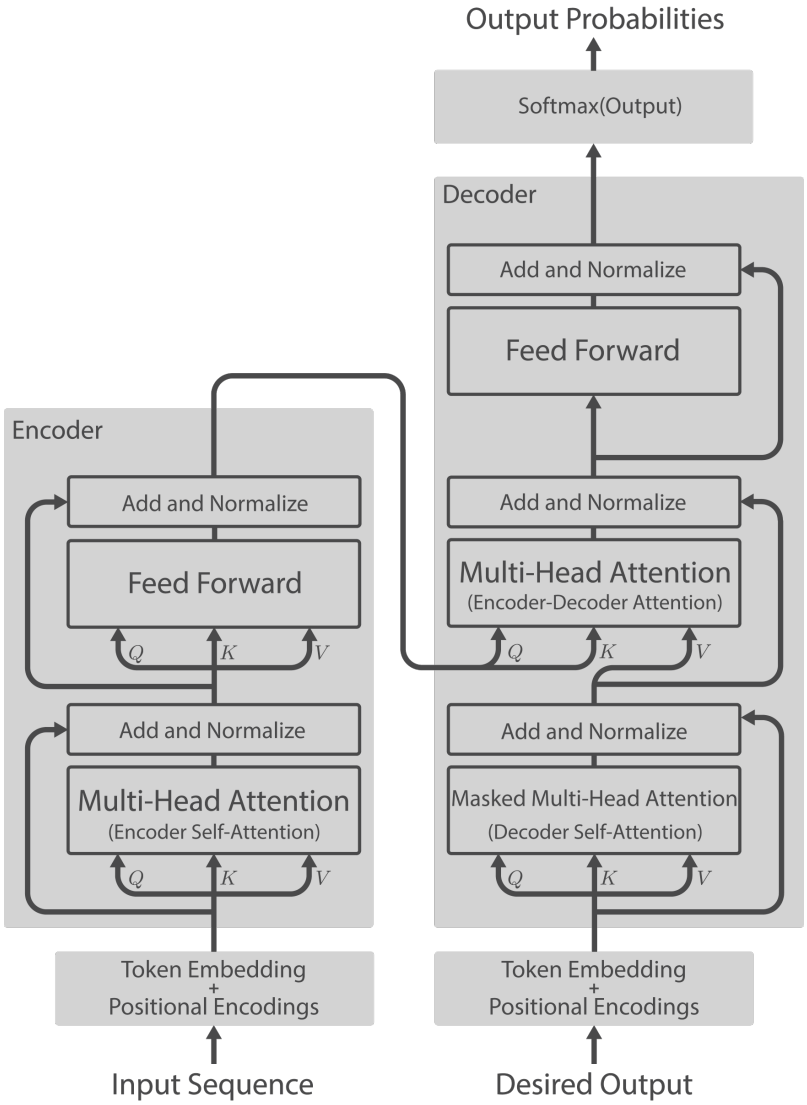


Figure 3.6: A schematic overview of the transformer neural network architecture, where the encoder and decoder submodules are visible.

The transformer architecture in Figure 3.6, which greatly inspired the BERT language model, is one attempt at modeling these contextual word dependencies without using recurrent models. The initial application of the transformer was machine translation, where input sequences in one language are trained to match output sequences in another language. The model consists of two similarly constructed modules, an *encoder* and a *decoder*. The encoder module is fed an input sequence, while the decoder module is fed a target sequence, shifted one step to the right. This shifting is done to ensure that the model does not simply copy the encoder input, but rather learns to predict subsequent tokens. The encoder influences the decoder output by way of inter-module connections that will be explained shortly. The final output of the decoder are the predicted probabilities of the encoder input corresponding to the decoder output.

The novel transformer architecture circumvents the need for recurrence by making heavy use of the *self-attention mechanism* to capture sequential dependencies between words (Vaswani et al., 2017). The self-attention mechanism builds on the fact that word dependencies are not necessarily sequentially ordered. In other words, the context most relevant to a given word is not necessarily found in the word's immediately preceding neighbors (see Figure 3.7). The self-attention mechanism allows the model to pay attention to, or *attend* to, any part of the input which it deems most relevant. This will now be explained more formally.

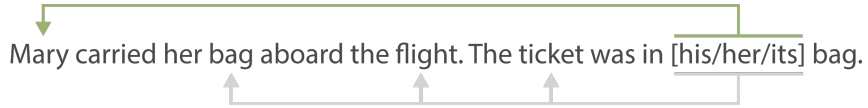


Figure 3.7: A given word can be dependent on distant words. In this example sequence, the chosen pronoun (his/her/its) will not be in reference to the closest nouns. Instead, the pronoun refers to Mary, mentioned in the very beginning of the sequence.

3.5.1.1 Attention

Attention and its variations are used across many disciplines within machine learning (Xu et al., 2015; Ambartsoumian and Popowich, 2018) and can be thought of as a mapping of a scalar query q, and vectors key $\boldsymbol{k}$ and value $\boldsymbol{v}$ to some attention value a (Vaswani et al., 2017). What the quantities q, k and v actually represent depends on the application. A general expression for attention is,

$$a(q, \boldsymbol{k}, \boldsymbol{v}) = \boldsymbol{\alpha}(q, \boldsymbol{k}) \cdot \boldsymbol{v}$$

where $\boldsymbol{\alpha}$ is the vector with elements,

$$\alpha_j = \text{softmax}(f(q, k_j)) = \frac{\exp(f(q, k_j))}{\sum_{j=1}^{n} \exp(f(q, k_j))}$$

as stated by Ambartsoumian and Popowich (2018) and Vaswani et al. (2017). The function f is an alignment function that calculates some measure of compatibility

or similarity between inputs q and k_j. The transformer uses the scaled dot-product as this similarity measure,

$$f(q, k_j) = \frac{kq}{\sqrt{d_k}}$$

where d_k is the dimensionality of k. Intuitively, attention is a weighted sum of the elements of $\boldsymbol{v}$ as according to the similarities between q and the elements of $\boldsymbol{k}$.

The transformer relies on a variant of attention known as self-attention. Given an input sequence, self-attention represents each position's attention value in relation to all other positions in the same sequence. Calculating self-attention is equivalent to calculating how every input in a sequence relates to every other input in that sequence. To calculate all attention values for all positions in an input sequence, the self-attention calculation can be written in matrix notation as,

$$\boldsymbol{a}(Q, K, V) = \text{softmax}\Big(\frac{QK^T}{\sqrt{d_k}}\Big)V$$

The transformer model further tweaks attention into *multi-head attention*. In the calculation of multi-head attention, the matrices Q, K and V are linearly projected with the learned parameter matrices $W_{Q,h}$, $W_{K,h}$ and $W_{V,h}$ for a number of different *attention heads*. These heads are each an instance of this linear projection. The attention vectors from each head are then concatenated and multiplied with one last learned matrix W_0 to produce the final multi-head attention matrix. The calculations for the multi-head attention matrix M are,

$$M = (\boldsymbol{h}_1, \boldsymbol{h}_2, ..., \boldsymbol{h}_h)W_0$$

where,

$$\boldsymbol{h}_i = \boldsymbol{a}(QW_{Q,i}, KW_{K,i}, VW_{V,i})$$

As seen in Figure 3.6, the matrices Q, V and K are retrieved in three slightly different ways depending on where in the transformer the multi-head attention is being calculated,

Encoder-decoder attention
This variant is calculated with values from both encoder and decoder modules. The encoder and decoder attention sets Q to the output of the decoder, K and V as the ouput from the encoder.

Encoder self-attention
This variant is used in the encoder module. Here, Q, K and V are all set to the same value, namely the output of the previous encoder layer.

Decoder self-attention
This variant is used in the decoder module. Similarly to the above Encoder self-attention, Q, K and V all take the value of the output of the preceding decoder layer. However, these values are tweaked so that when querying value x_i, all succeeding positions $> i$ are set to $-\infty$. In other words, for position i, set $Q = K = V = (x_1, x_2, ..., x_i, -\infty, ..., -\infty)$

The self-attention mechanism does not take the ordering of the input sequence into account. To create an implicit ordering of the input, the transformer first applies a positional encoding,

$$PE_{pos,2i} = \sin(pos/10000^{2i/d_{model}})$$
$$PE_{pos,2i+1} = \cos(pos/10000^{2i/d_{model}})$$

where pos is the position of the token and i is the dimension in question. These positional encoding values are simply added to the input vectors.

In summary, the transformer architecture is an encoder-decoder model that uses multi-head attention to map dependencies between input tokens in long sequences. This mechanism allows for parallelized training and learning long-term dependencies. Augmenting the multi-head attention with positional encodings allows the transformer to capture the ordering of input sequences. The resulting model has proved adept at a number of natural language tasks (Vaswani et al., 2017). While the transformer architecture model lends itself well to language processing, the true novelty of BERT is the training used to instill this model with linguistic knowledge.

3.5.2 Training BERT

The original BERT paper presented two architectures, $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$. The smaller of the two, $\text{BERT}_{\text{BASE}}$, is the focus of this study and consists of 12 sequentially ordered transformer encoders (see Figure 3.8).

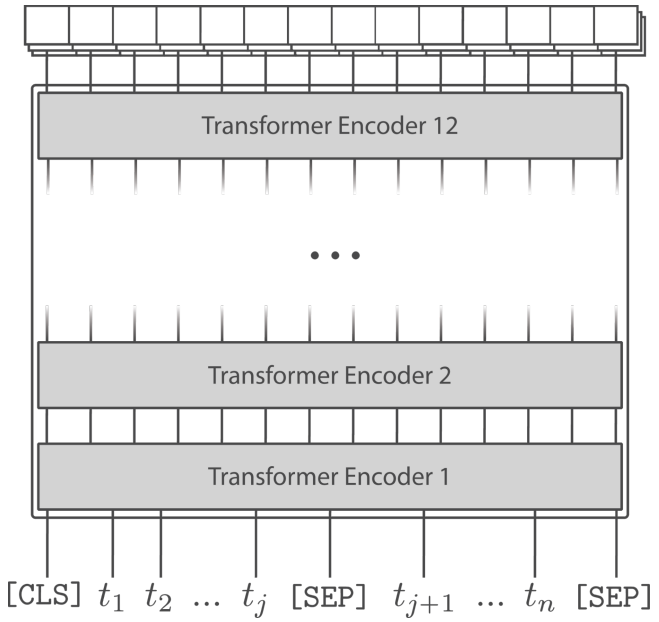


Figure 3.8: A high-level schematic of the BERT model. $\text{BERT}_{\text{BASE}}$ consists of 12 sequentially stacked transformer encoders. Note the concatenation scheme of input tokens and the `[CLS]` and `[SEP]` tokens. The purpose of these special tokens will be more closely examined in the next sections.

Each of these encoders represent input in 768 dimensions, where each multi-head attention layer contains 12 heads. The resulting language model contains around 110 million learnable parameters. Due to the complexity of the model, pretraining requires a formidable corpus. The corpus used for pretraining BERT was a combination of the Bookcorpus and an English corpus scraped from Wikipedia totalling 3.2 billion words (Devlin, Chang, Lee and Toutanova, 2019).

3.5.2.1 Input Representation

Machine learning models are typically not fed strings of raw text as input data, and BERT is no different. A ubiquitous initial step in language processing is to divide input sequences into their constituent parts in a process known as *tokenization*. Tokenizing input data preprocesses input text by removing whitespace and sometimes punctuation. Tokenization can also split words into constituent subwords. Through tokenization, models can be exposed to relationships between words that may appear superficially different. For instance, 'gone' and 'going' are two distinct strings when viewed at word-level. By first tokenizing the words into 'go'+'ne' and 'go'+'ing', it becomes apparent that these two words show some commonalities on the subword level.

Word Sequence: [I, saw, a, hippopotamus, in, my, garden]

Tokenized Sequence: [I, saw, a, hip, pop, ot, a, mus, in, my, gar, den]

Figure 3.9: The WordPiece tokenizer will never concede that a word is outside its vocabulary. It instead performs a recursive splitting of an unknown word. In this example sequence, the words 'hippopotamus' and 'garden' are not present in the tokenizer's vocabulary. These words are instead split into subwords that do exist in the vocabulary. Note that this can substantially increase the sequence length if the sequence contains many out-of-vocabulary words.

There are many tokenization strategies, each producing different divisions of words into tokens. Studies show that these different strategies can drastically affect performance on downstream natural language tasks (Jiang and Zhai, 2007; Kudo and Richardson, 2018). For BERT, tokenization is done with the WordPiece tokenizer. The WordPiece tokenizer strives to express the input corpus with a fixed-size vocabulary. If a word is not found in this vocabulary, it is split into subwords in a way that minimizes the number of tokens needed to express the original word. If some subword still cannot be found in the vocabulary, it is split into sub-subwords. In a worst-case scenario, this recursive procedure continues splitting until the word has been divided into individual letters (which are guaranteed to exist in the vocabulary). This tokenization procedure ensures that there are no out-of-vocabulary tokens (Wu et al., 2016). One side-effect of the continued splitting is that text which is dense in out-of-vocabulary words and subwords can result in excessive splitting, thereby significantly lengthening the input text (see Figure 3.9).

The aforementioned vocabulary is constructed through an optimization problem where a source corpus must be represented by a vocabulary of fixed size. The tokens to be included in the vocabulary are chosen such that the source corpus can be fully represented with as few splits as possible. In other words, it should be possible to reconstruct the source corpus with the chosen tokens such that word-splitting is kept to a minimum (Wu et al., 2016).

Once tokenization has been performed, pairs of input sequences are concatenated as, `[CLS]` + sequence 1 + `[SEP]` + sequence 2 + `[SEP]`. The reason for this particular concatenation scheme will become clear in Section 3.5.2.2. In the next preprocessing step, input tokens are represented as high-dimensional representations. These token representations, known as *embeddings*, represent each token in the vocabulary as a vector in 768-dimensional space. Each token's unique vector representation is randomly initialized at the start of training and subsequently trained in the same manner as any other learnable model parameter.

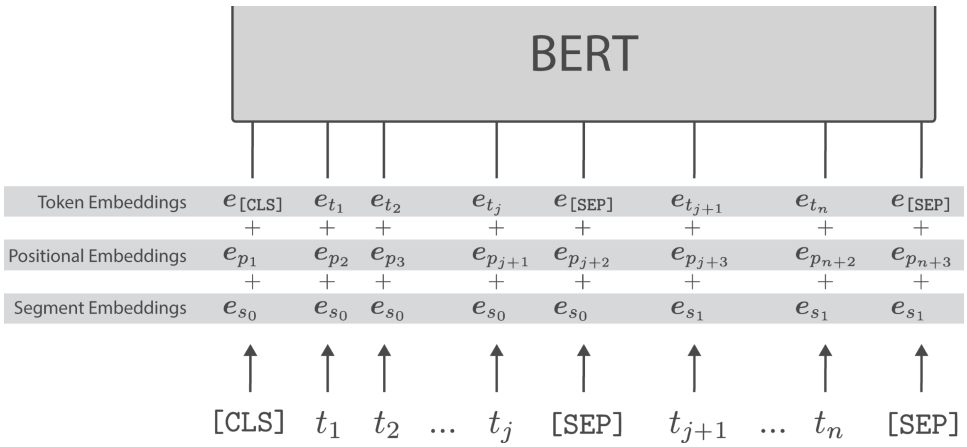


Figure 3.10: A schematic of the ingredients comprising the complete BERT input. Token, positional and segment embeddings are added prior to being fed to the model.

The final input representation consists of more than just the token embeddings however (see Figure 3.10). The input representation is a sum of the token embeddings, and position and segment embeddings, $\boldsymbol{e}_{token} + \boldsymbol{e}_{pos} + \boldsymbol{e}_{seg}$. While the token embeddings consist of a unique 768-dimensional vector for each token in the vocabulary, the position embeddings are 768-dimensional vectors unique to each of the 512 input positions (the maximum possible input sequence length for BERT). Adding these positional embeddings allows a single token's vector representation to differ depending on its position in the input sequence. For instance, the token 'walk' is represented by a single token embedding, $\boldsymbol{e}_{walk}$. However, the input representation of the token will differ in the sequences 'walk away' and 'I walk to work' thanks to the addition of position embeddings $\boldsymbol{e}_{p_1}$ to the first sequence and $\boldsymbol{e}_{p_2}$ to the second.[7] The last ingredient to the input representation is the addition of segment embeddings. For

[7]Note that this addition of position embeddings replaces the positional encodings in the transformer architecture as described in Section 3.5.1.1. Furthermore, the positional embeddings are learned parameters in contrast to the deterministically calculated positional encodings.

reasons that will become clear, BERT can accept inputs that are concatenations of two sequences. The segment embeddings allow BERT to differentiate between the two sequences. The segment embeddings can take the value of only two unique 768-dimensional vectors. The values of the token, position and segment embeddings are randomly initialized. These vectors are trained in the same manner as all other learnable parameters in the model.

3.5.2.2 Pretraining Tasks

The pretraining tasks are arguably the most significant finding in the BERT paper. The BERT language model is pretrained simultaneously on two tasks: masked language modeling and next sentence prediction; both of which contribute to BERT's performance on downstream natural language tasks (Devlin, Chang, Lee and Toutanova, 2019). Both tasks are unsupervised[8], thus enabling BERT to be trained on text data without costly and time-consuming annotation. The training procedure uses two so-called pretraining *heads* (see Figure 3.11). These are small, task-specific networks that use the output of BERT's transformer encoder model as input. Losses are calculated on the outputs of the pretraining heads and gradients are propagated down through both heads, and into BERT itself. The two pretraining tasks will now be explained in further detail.

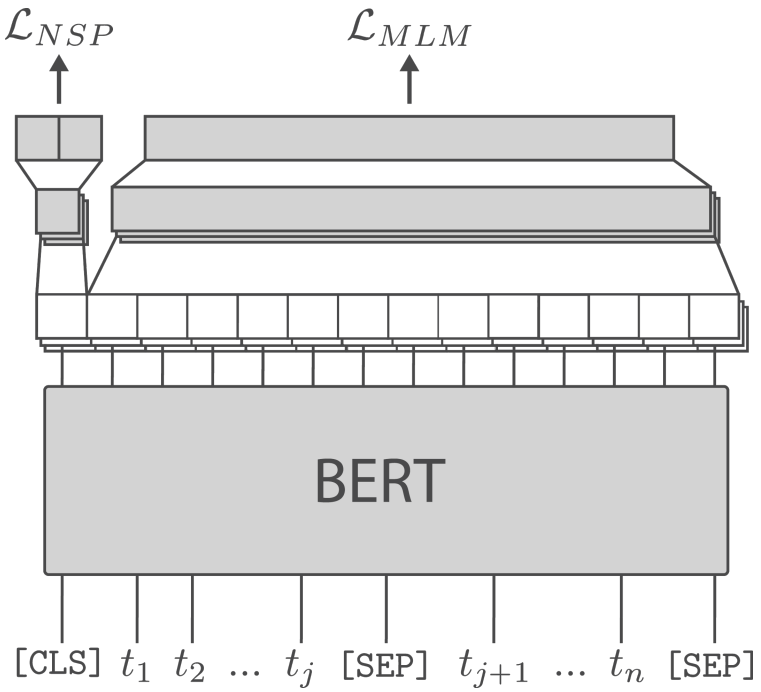


Figure 3.11: The BERT model and its pretraining heads. Note that the next sentence prediction head connects to the output corresponding to the `[CLS]` token.

[8]Note that *unsupervised* here refers to the absence of human annotation. Training is supervised in the sense that a label is still required for calculating losses. However, this labeling is synthetically generated from the textual data.

In the masked language modeling task[9], 15% of the input tokens are randomly selected and replaced by the `[MASK]` token or some other randomly selected token from the vocabulary. It is then the model's objective to predict the missing token, based solely on its surrounding context. The masked language modeling head takes the outputs corresponding to the masked tokens and feeds these through a feed-forward network into a softmax function, calculated over the entire vocabulary.

It is worth noting that the masked language modeling task allows for a bidirectional language model. This means that word prediction is based on both preceding and succeeding context. This is different from previous transformer- and recurrent-based language models where next word prediction was commonly used as the language modeling task. Next word prediction is performed by feeding the model with a snippet of text and having it predict the next token. This imbues the model with an understanding of how words are conditioned on their preceding context. However, only focusing on preceding context can be limiting when there is potential information in the succeeding context as well (see Figure 3.12). The masked language modeling task makes it possible to form an understanding of the interplay between left-to-right and right-to-left contexts, leading to a richer model.

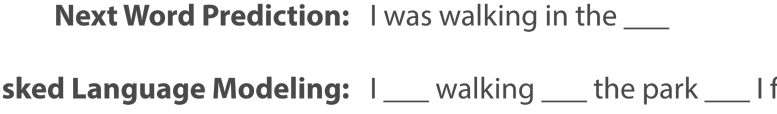


Figure 3.12: A comparison between next word prediction and masked language modeling tasks. Note that next word prediction only takes the preceding context into account, while masked language modeling uses both preceding and succeeding contexts.

The second pretraining task, next sentence prediction, aims to train BERT to recognize long-range dependencies (Devlin, Chang, Lee and Toutanova, 2019). Recall that BERT is fed input in which two sequences have been concatenated with the scheme `[CLS]` + sequence 1 + `[SEP]` + sequence 2 + `[SEP]`. The next sentence prediction task is a binary classification problem, with the goal of identifying whether these two input sequences are adjacent in the pretraining corpus. During pretraining, 50% of these input sequences are neighbors in the pretraining corpus, while the other half are randomly concatenated sequences. This task is a so-called *sequence classification* task because predictions are made on a sequence level (as opposed to token-level predictions in the masked language task). In this task, as in all sequence classification tasks, output corresponding to the `[CLS]` token is routed to a fully connected feed-forward network. This classification head is then trained to recognize whether the two input sequences are adjacent.

Pretraining BERT on these two tasks is a computationally intense endeavor. The original model was pretrained on four TPUs, with each TPU housing four TPU chips, for a total of four days. Training time would increase drastically if the same pretraining procedure was performed on a GPU-based setup (Devlin, Chang, Lee

[9]This task has been previously mentioned in linguistics literature as the Cloze task.

and Toutanova, 2019). Pretraining as well as task-specific finetuning are performed with the Adam gradient descent optimization algorithm. Once the strenous pretraining procedure is complete, BERT is well-prepared to tackle further training for downstream tasks.

3.5.3 Task-Specific Finetuning of BERT

When BERT has been sufficiently pretrained it contains a linguistic understanding that can be effectively transferred to a variety of natural language tasks (Devlin, Chang, Lee and Toutanova, 2019). For the purposes of this study, that task is sequence classification. This final training is known as finetuning, and requires task-specific labeled data. Thanks to the transfer learning of the language model, the amount of this labeled data is typically lower than if the language modeling step had been omitted (Howard and Ruder, 2018; Peters et al., 2018). As previously mentioned, sequence classification tasks route the output corresponding to the `[CLS]` token into a task-specific head. This head is a feedforward classification network where output is run through a softmax calculation. In the finetuning process, the entirety of the model (both BERT and the task-specific head) is trained.

3.6 Domain Adaptation

As previously mentioned, language models like BERT are pretrained on a large unlabeled corpus. BERT uses a pretraining corpus scraped from Wikipedia and Bookcorpus. Training on this source domain has a beneficial impact on the model's performance on downstream language processing tasks in a separate target domain (Devlin, Chang, Lee and Toutanova, 2019). However, when dealing with downstream tasks in specialist language domains, there may be a linguistic disparity between source and target domains. For instance, the vocabulary of the specialist domain may differ from the source domain vocabulary, words may differ in their usage, or sentences may have a very different structure. Therefore, the pretraining procedure may not have captured the language structures most relevant to the downstream task domain. *Domain adaptation* seeks to smooth the transition from the source domain in pretraining to the target domain of the downstream task; striving for maximum knowledge transfer between the two.

Given a target domain $\mathcal{D}_T = \{\mathcal{X}_T, P_T(X)\}$, target task $\mathcal{T}_T = \{\mathcal{Y}_T, P_T(Y|X)\}$, source domain $\mathcal{D}_S = \{\mathcal{X}_S, P_S(X)\}$ and source task $\mathcal{T}_S = \{\mathcal{Y}_S, P_S(Y|X)\}$, recall the four inequalities (of which at least one holds) in transfer learning scenarios (Pan and Yang, 2010),

$$\mathcal{X}_T \neq \mathcal{X}_S \qquad P_T(X) \neq P_S(X) \qquad \mathcal{Y}_T \neq \mathcal{Y}_S \qquad P_T(Y|X) \neq P_S(Y|X)$$

Because BERT's training procedures commence by recreating data in terms of tokens from a vocabulary of fixed size, the inequality $\mathcal{X}_T \neq \mathcal{X}_S$ does not hold. However, it can safely be assumed that the underlying probability distributions of the tokens are

not the same in the two domains. In other words, the inequality $P_T(X) \neq P_S(X)$ holds. Likewise, since the output spaces of the source tasks and target task are different, the inequality $\mathcal{Y}_T \neq \mathcal{Y}_S$ holds. These two valid inequalities render the inequality $P_T(Y|X) \neq P_S(Y|X)$ nonsensical.

Since estimating $P(Y|X)$ is the aim of machine learning, it would seem plausible that an operation attempting to equate learning $P_T(Y|X)$ and learning $P_S(Y|X)$ would be beneficial to the transfer learning procedure of language models. There exist many such procedures that attempt to relate learning in the source domain to learning in the target domain. Two methods which prove relevant to this study are presented in the following sections.

3.6.1 Domain Finetuning

A simple method of domain adaptation is to attempt to imbue the language model with target domain specific language knowledge. This is done by continuing the pretraining procedure on an unlabeled corpus from the target domain after the initial source domain pretraining. When continuing the pretraining on more relevant language, the goal is to adjust the linguistic knowledge already present in the language model to the target domain. This procedure is mentioned (though not formally attempted) in the original BERT paper (Devlin, Chang, Lee and Toutanova, 2019).

3.6.2 Domain Invariance Training

When training in the source domain, it would be counterproductive to learn language features which are unapplicable to the target domain. If these unnecessary source-specific features are present, the model could identify their absence, thereby differentiating between source and target domains. Removing the model's ability to differentiate between domains would lead the model to ignore source-specific features and thereby focus on learning more general features. This is exactly the intuition behind *domain invariance* adaptation. In this domain adaptation method, the model is made unable to discriminate between source and target domains. By hampering the model's ability to identify the domain, no features are learned that are specific to any one domain. This shifts the learning focus to features that are present in both domains, ensuring that knowledge learned from the source domain is applicable to the target domain as well (Ben-David et al., 2010; Ganin et al., 2016).

To make a neural network domain invariant, training is expanded to include a domain identification task in addition to the preexisting modeling task. For domain identification, the model is fed data from both source and target domains. The model then attempts to identify whether input originates from the source or target domain. Since the model should not be able to discriminate between domains, the model is trained to perform as poorly as possible on the domain identification task while performing as well as possible on the original training task. This method will now be formally explained.

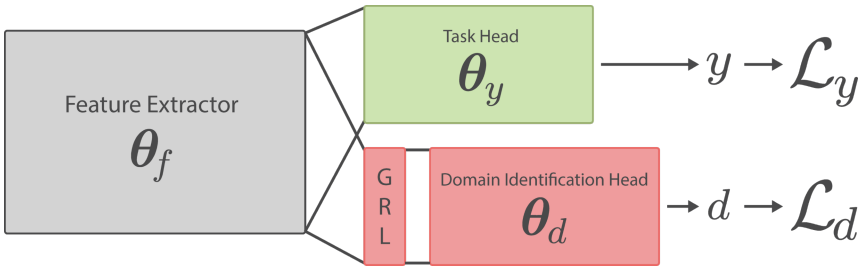


Figure 3.13: Domain invariance training partitions the original network into a feature extractor and a task-specific head. These two subnetworks have the parameters $\boldsymbol{\theta}_f$ and $\boldsymbol{\theta}_y$ respectively. An additional subnetwork is appended to the feature extractor. This subnetwork, with parameters $\boldsymbol{\theta}_d$, attempts to identify the domain of the input, given the output from the feature extractor.

The original model is partitioned into two subnetworks: the initial subnetwork functions as a feature extractor with model parameters $\boldsymbol{\theta}_f$ and the second subnetwork is a task-specific head with parameters $\boldsymbol{\theta}_y$ (see Figure 3.13). By appending a domain identification head (with parameters $\boldsymbol{\theta}_d$) to the feature extractor, the original task can be trained while the domain identification task is simultaneously un-trained. The goal is now to update each head's parameters $\boldsymbol{\theta}_y$ and $\boldsymbol{\theta}_d$ such that performance on their respective tasks is maximized. Furthermore, the feature extractor's parameters $\boldsymbol{\theta}_f$ should be updated such that performance on the original task is maximized, but domain identification performs as poorly as possible. This yields the following interlocked optimization problems,

$$\arg\min_{\hat{\boldsymbol{\theta}}_f, \hat{\boldsymbol{\theta}}_y} \mathcal{L}_y(\hat{\boldsymbol{\theta}}_f, \hat{\boldsymbol{\theta}}_y) - \mathcal{L}_d(\hat{\boldsymbol{\theta}}_f, \boldsymbol{\theta}_d) \tag{3.1}$$

$$\arg\max_{\hat{\boldsymbol{\theta}}_d} \mathcal{L}_y(\boldsymbol{\theta}_f, \boldsymbol{\theta}_y) - \mathcal{L}_d(\boldsymbol{\theta}_f, \hat{\boldsymbol{\theta}}_d) \tag{3.2}$$

where $\mathcal{L}_y$ is the loss associated with the original task and $\mathcal{L}_d$ is the domain identification loss. Note the conflict between the two optimizations, where we simultaneously wish to maximize the domain identification loss $\mathcal{L}_d$ with respect to $\boldsymbol{\theta}_f$ but minimize the same quantity with respect to $\boldsymbol{\theta}_d$. These conflicting goals make it impossible to optimize model parameters through stochastic gradient descent, as gradient descent cannot maximize on certain parameters while minimizing on others. Using stochastic gradient descent on the total loss of $\mathcal{L}_y - \mathcal{L}_d$ would yield the following erroneous update rules,

$$\begin{aligned}
\boldsymbol{\theta}_f &\leftarrow \boldsymbol{\theta}_f - \eta\nabla_{\boldsymbol{\theta}_f}(\mathcal{L}_y - \mathcal{L}_d) &&= \boldsymbol{\theta}_f - \eta\nabla_{\boldsymbol{\theta}_f}\mathcal{L}_y + \eta\nabla_{\boldsymbol{\theta}_f}\mathcal{L}_d \\
\boldsymbol{\theta}_y &\leftarrow \boldsymbol{\theta}_y - \eta\nabla_{\boldsymbol{\theta}_y}(\mathcal{L}_y - \mathcal{L}_d) &&= \boldsymbol{\theta}_y - \eta\nabla_{\boldsymbol{\theta}_y}\mathcal{L}_y \\
\boldsymbol{\theta}_d &\leftarrow \boldsymbol{\theta}_d - \eta\nabla_{\boldsymbol{\theta}_d}(\mathcal{L}_y - \mathcal{L}_d) &&= \boldsymbol{\theta}_d + \eta\nabla_{\boldsymbol{\theta}_d}\mathcal{L}_d
\end{aligned} \tag{3.3}$$

In the above update rules, it is apparent that optimizing on $\mathcal{L}_y - \mathcal{L}_d$ with gradient descent leads to incorrect updates on $\boldsymbol{\theta}_d$ in equation (3.3). The update rule in (3.3) adjusts $\boldsymbol{\theta}_d$ in the direction of the gradient, thereby increasing the domain

identification loss $\mathcal{L}_d$ with respect to $\boldsymbol{\theta}_d$. This is the direct opposite of the sought-after optimization in equation (3.2).

Utilizing a clever trick makes it possible to simultaneously optimize both minimization and maximization problems while still using stochastic gradient descent. To enable simultaneous minimization and maximization, a *gradient reversal layer* can be inserted between the feature extractor and the domain identification head. This neural network layer multiplies the gradient by -1 during backpropagation, but leaves values unchanged during forward passes.

Performing stochastic gradient descent on $\mathcal{L}_y + \mathcal{L}_d$ (note that the losses are now added instead of subtracted), yields the following update rules,

$$\begin{aligned} & & \boldsymbol{\theta}_f &\leftarrow \boldsymbol{\theta}_f - \eta\nabla_{\boldsymbol{\theta}_f}\mathcal{L}_y - \eta\nabla_{\boldsymbol{\theta}_f}\mathcal{L}_d \\ \text{gradient descent on } \mathcal{L}_y + \mathcal{L}_d & \Rightarrow & \boldsymbol{\theta}_y &\leftarrow \boldsymbol{\theta}_y - \eta\nabla_{\boldsymbol{\theta}_y}\mathcal{L}_y \\ & & \boldsymbol{\theta}_d &\leftarrow \boldsymbol{\theta}_d - \eta\nabla_{\boldsymbol{\theta}_d}\mathcal{L}_d \end{aligned} \tag{3.4}$$

The above update rules are still incorrect. However, by including the aforementioned gradient reversal layer between the feature extractor and the domain identification head, the subtraction of $\nabla_{\boldsymbol{\theta}_f}\mathcal{L}_d$ in equation (3.4) is changed to addition. This results in the correct update rules for simultaneous optimization on (3.1) and (3.2).

$$\begin{aligned} & & \boldsymbol{\theta}_f &\leftarrow \boldsymbol{\theta}_f - \eta\nabla_{\boldsymbol{\theta}_f}\mathcal{L}_y + \eta\nabla_{\boldsymbol{\theta}_f}\mathcal{L}_d \\ \text{gradient descent on } \mathcal{L}_y + \mathcal{L}_d & & & \\ \textit{with} & \Rightarrow & \boldsymbol{\theta}_y &\leftarrow \boldsymbol{\theta}_y - \eta\nabla_{\boldsymbol{\theta}_y}\mathcal{L}_y \\ \text{Gradient Reversal Layer} & & \boldsymbol{\theta}_d &\leftarrow \boldsymbol{\theta}_d - \eta\nabla_{\boldsymbol{\theta}_d}\mathcal{L}_d \end{aligned}$$

4

Method

This study aims to benchmark the performance of a pretrained BERT model on legal text classification. This benchmarking serves as an indication of how effectively BERT's pretrained knowledge transfers to a different language domain. Once this initial benchmarking has been performed, three methods of adapting the pretrained BERT model to the legal language domain will also be studied. Finally, a BERT instance which is pretrained completely from scratch will be benchmarked and compared to the three domain adaptation techniques.

More concretely, each of the three domain adapted models and the continued pretraining model are evaluated as follows. The starting point for all models is the freely available pretrained version of BERT. This pretrained BERT is then put through three epochs of domain adaptation or continued pretraining (see Figure 4.1). At each epoch, the model state is saved for evaluation on the legal classification task (see Figure 4.2).

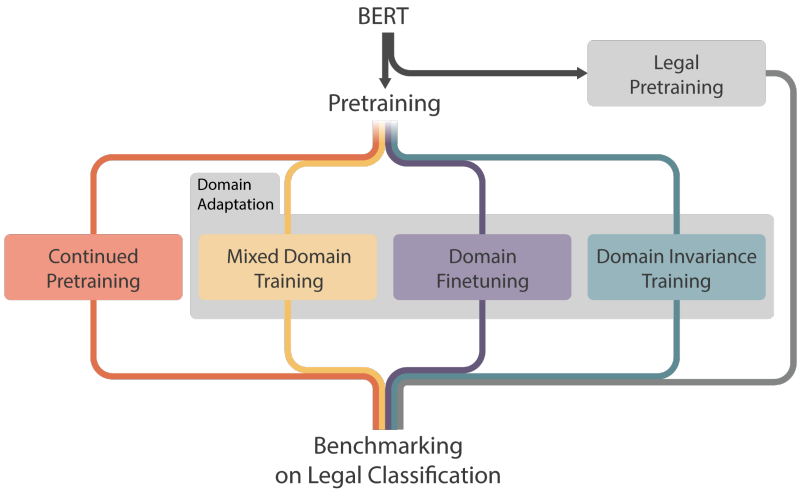


Figure 4.1: An overview of the experiment pipeline. The pretrained BERT model is used a starting point for all domain adaptation methods and continued pretraining. A BERT instance is also pretrained on legal data and benchmarked.

By evaluating models after each epoch of domain adaptation, it is possible to track the rate at which domain adaptation occurs. A fast domain adaptation method requires few domain adaptation epochs before improvement is seen on target domain tasks. Since domain adaptation is computationally expensive, a fast domain adaptation method may be particularly interesting in practical use cases.

In this study, the domain adaptation is run on two NVIDIA GTX Titan X GPUs, each with 12 gigabytes of graphical memory. The training is run with half-precision operations (enabling larger batch sizes) with a loss scaling of 128 through NVIDIA's `apex` library. BERT is set to accept sequences of 256 tokens, and training is performed with a learning rate of $3 \cdot 10^{-5}$ with a batch size of 64.

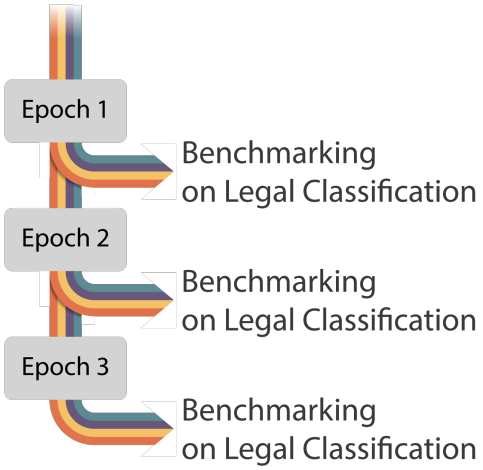


Figure 4.2: Each of the domain adaptation methods as well as continued pretraining are benchmarked after every epoch.

This chapter describes the legal classification task used for benchmarking, the benchmarking of pretrained and continued pretraining models, the benchmarking of domain adaptation methods tailored for BERT, as well as the benchmarking of a BERT model pretrained on legal text.

4.1 Benchmarking on Legal Text Classification

In this study, legal text classification is used to evaluate the degree with which the pretrained and domain adapted BERT model has captured target domain language. Each of the domain adapted BERT models, as well as the BERT models which have undergone continued and legal pretraining, are evaluated on this task. The classification task is performed by connecting the output vector corresponding to the sequence-level `[CLS]` tag to a feed-forward classification network. Evaluation is performed by training the complete model (BERT and its task-specific classification head) on legal classification for five epochs and subsequently testing on a hold-out set.

The legal text classification dataset is an example of specialist literature from a language domain different to the source language domain used for the original pretraining. Beyond exhibiting linguistic differences, the dataset also contains noise in the form of optical character recognition errors. These errors arose during dataset construction when the legal text was electronically transferred from paper docu-

ments to the digital medium. The errors starkly contrast the heavily proofread text which BERT was pretrained on, and further differentiate the source and target language domains. The differences between the source domain and the language domain of the legal text make classification in this setting apt for measuring domain adaptation.

The classification task itself consists of classifying sentences as one of seven legal text classes, called provisions, or an additional non-provision class reserved for text lacking any specific legal role. There are significant class imbalances in the 136 megabyte text dataset, with the largest class containing 615 900 examples and the smallest class containing only 1388 examples (see Table 4.1).

Each model is benchmarked by training on subsamplings of the total classification dataset. These subsampled training sets come in four different sizes and serve to illustrate the effects of domain adaptation in scenarios ranging from very little task-specific data, to plentiful task-specific data (see Figure 4.3). These subsamplings were stratified, meaning that the distributions of classes were equal regardless of subsample size. After training for five epochs with a batch size of 32 and learning rate of $2 \cdot 10^{-5}$, the model has undergone task-specific fine-tuning and is evaluated on a classification test set. The size of the test set is kept constant, regardless of the size of the training set. It is worth noting that the test set does not overlap with the training set. Since both test and training sets are randomly sampled from the total classification dataset, the evaluation experiment is repeated seven times to yield statistically informative results (see Figure 4.4).

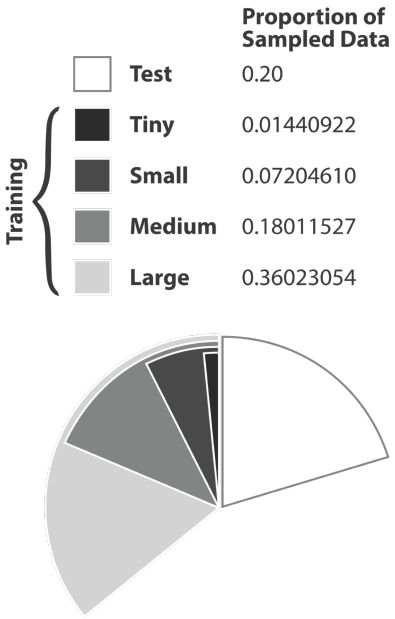


Figure 4.3: An illustration of the four different training set subsample sizes.

BERT builds on the transformer architecture, which means that the model is fed entire sequences instead of the batchwise input schemes commonly used for recurrent language models. While this allows the bidirectionality of BERT, it also means that BERT places a fixed constraint on the maximum length of input sequences. In the legal text classification task, this maximum sequence length is set to 256 tokens. Using longer sequences leads to higher memory requirements thus limiting training batch size. In turn, this increases the number of batches per epoch ultimately leading to longer training. The 256 token limit allows large enough batch sizes that training is completed within reasonable timeframes. However, the restriction forces a few lengthy training examples to be truncated, thus destroying information (see Table 4.1).

Provision	No. of Sentences	Example	Truncated
Auto Renewal	1647	termination this agreement shall continue until terminated in accordance with this clause 20 or clause 23 .	0.8 %
Change of Control	1388	neither party may assign this agreement without the prior written consent of the other party .	6.4 %
Force Majeure	3199	neither party will be liable for performance delays or for non-observance or non-performance due to causes beyond its reasonable control .	1.4 %
Indemnity	6380	13 indemnity 13.1 the supplier shall indemnify and hold the customer harmless from all claims in connection with : 13.1.1	3.2 %
Limitation of Liability	6118	the broker shall not be responsible for any investment risk or any loss or profit realized from the operation of the assets .	1.0 %
Termination Cause	6762	by reason of bankruptcy or insolvency of either party as described below .	2.9 %
Termination Convenience	1815	if client wishes to terminate ihis agreement without cause prior to the end of the initial term .	2.5 %
NONE	615900	upon signature by the parties , each order form and provisioning form will be incorporated into and governed by the terms of this agreement .	0.5 %

Table 4.1: The distributions of classes in the legal classification dataset. The last column specifies the percentage of examples which have been truncated upon restricting input to 256 tokens. It should be noted that the dataset used for benchmarking has been specially constructed for this study.

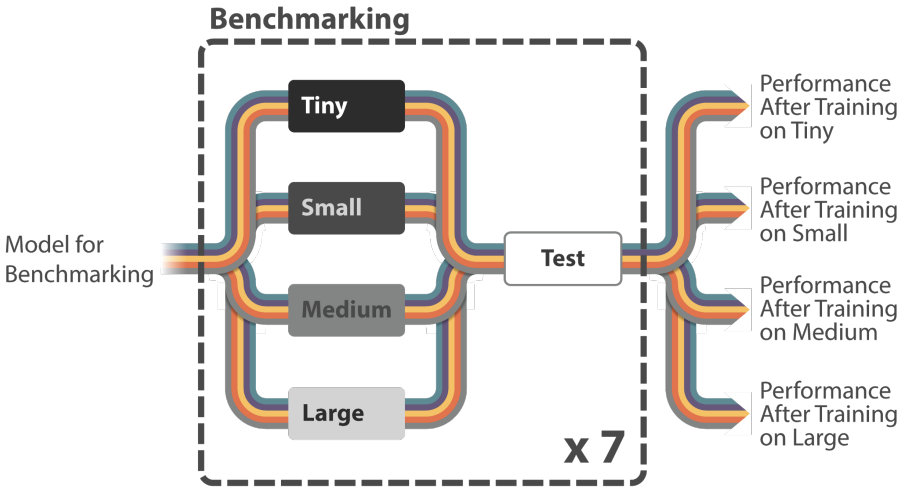


Figure 4.4: An overview of the benchmarking pipeline. Each model is trained on four subsamplings of the labeled training data. These subsamplings vary in size in order to investigate the effect of training set size on performance. Due to the stochastic nature of this sampling, the benchmarking is repeated seven times.

Now that the evaluation task has been more thoroughly explained, the model that has undergone continued pretraining, the legally pretrained model, as well as the domain adapted model versions are presented.

4.2 Pretrained BERT and Continued Pretraining

There is a possibility that the pretrained BERT model may benefit from any training, regardless if it strives for domain adaptation or not. To control for this possibility, an additional pretrained BERT model continues pretraining on the original BERT language modeling task (instead of undergoing domain adaptation) for three epochs and is then evaluated on the legal classification task. The result is a model that has been trained for the same number of total epochs as the domain adapted variants, thus enabling a fair comparison between domain adapted and non-domain adapted models.

This continued pretraining procedure requires reconstructing the original pretraining datasets. As previously mentioned, BERT has been pretrained on English language data retrieved from Wikipedia and Bookcorpus. The pretraining dataset combined a 2.5 billion word corpus scraped from Wikipedia and an 800 million word corpus from Bookcorpus. Unfortunately, the original paper does not publish the datasets, nor detail the exact steps taken during data preprocessing. The original pretraining dataset has therefore been approximated as closely as possible to allow for continued pretraining.

The approximated pretraining data is created by first downloading raw Wikipedia articles and Bookcorpus texts. This text data is cleaned by removing HTML artifacts, tabular data and the incomplete sentences found in headings, titles, chapter headers and other formatting-related text. This data does not typically follow traditional language rules and is therefore deemed irrelevant to language modeling.

Furthermore, the approximated dataset is constructed so as to retain the ratio of Wikipedia to Bookcorpus data present in the original pretraining dataset. Maintaining the same proportion of Wikipedia to Bookcorpus data is an attempt at allowing continued pretraining to occur in a language domain with underlying data distributions similar to those of the original pretraining procedure. In total, the reconstructed pretraining dataset contains approximately 940 megabytes of text from Bookcorpus and approximately 2.99 gigabytes of textual data scraped from Wikipedia.

The model is benchmarked on the legal text classification task after each of the three epochs of continued pretraining on the approximated Wikipedia and Bookcorpus datasets.

4.3 Domain Finetuning

Domain finetuning is the domain adaptation method suggested, but not formally attempted, in the original BERT paper. Domain finetuning consists of training the pretrained model on the original language modeling tasks (masked language modeling and next sentence prediction) with data directly related to the language domain of the downstream task.

The unlabeled dataset used for legal domain finetuning consists of 3.9 gigabytes of data retrieved from legal contracts. The contracts have been converted from paper to digital form with optical character recognition. The resulting dataset therefore contains the same type of character recognition errors present in the text classification dataset. Note that the size of the domain finetuning dataset is similar to the size of the dataset used for continued pretraining. This is no accident. By using domain adaptation datasets and continued pretraining datasets of similar sizes, each of the compared models have been exposed to similar numbers of training examples. Conclusions can then be based on the domain adaptation techniques (or lack thereof) and not on the number of training examples used for domain adaptation.

The model is benchmarked on the legal text classification task after each of the three epochs of domain finetuning.

4.4 Domain Invariance Training

The aim of domain invariance training is to force the model to attend only to features which are present in both source and target domains (as explained in section 3.6.2). For this study, domain invariance training requires constructing an appropriate dataset as well as describing a BERT-specific domain invariance training procedure.

The domain adaptation dataset used for domain invariance training contains equal parts domain-related and non-domain-related data. Half of the dataset consists of data from the domain finetuning dataset, and the other half stems from the continued pretraining procedure. The concatenation of these two training subsets is a 3.9 gigabyte domain invariance training set.

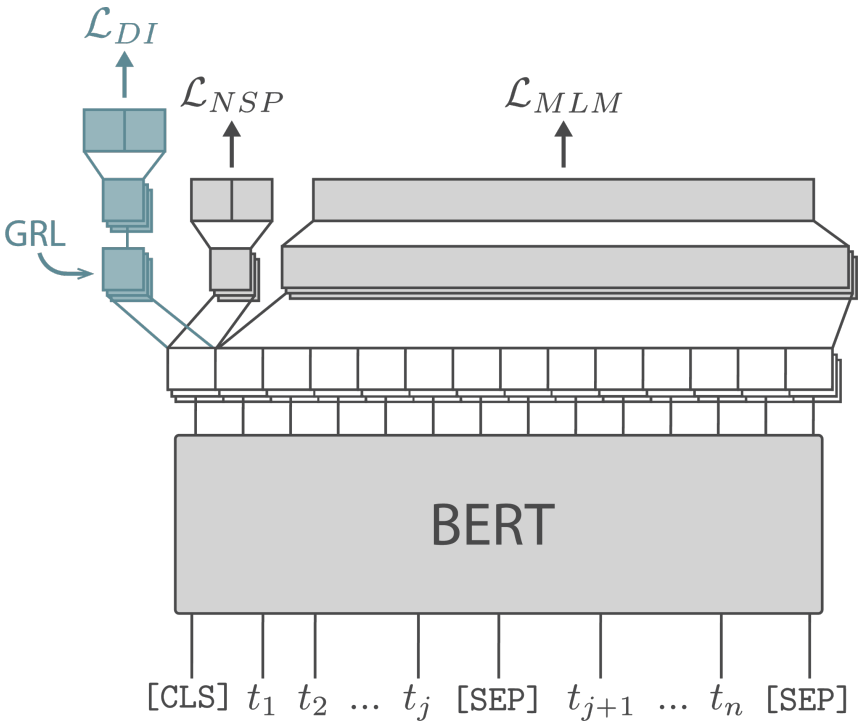


Figure 4.5: The domain invariance adaptation uses a domain identification head in addition to the original pretraining heads. The gradient reversal layer is visible as the first layer of the domain identification head.

The domain identification task is appended to the preexisting language modeling tasks, next sentence prediction and masked language modeling, for a total of three tasks (see Figure 4.5). The training head used for domain identification is connected to the sequence-level `[CLS]`-tag output in much the same way as the next sentence prediction task. Similarly to the next sentence prediction head, the domain identification training head is a fully connected feedforward network which acts as a binary classifier. However, the domain identification head is preceded by a gradient

reversal layer that multiplies values by -1 during backward passes. The result is a training head which minimizes the language model's ability to identify the domain of the training data thus making the model domain invariant. The complete loss function becomes,

$$\mathcal{L} = \mathcal{L}_{NSP} + \mathcal{L}_{MLM} + \mathcal{L}_{DI}$$

where $\mathcal{L}_{NSP}$ is the loss from the next sentence prediction task, $\mathcal{L}_{MLM}$ is the loss associated with masked language modeling and $\mathcal{L}_{DI}$ is the domain identification loss. The gradient reversal layer which precedes the domain identification head causes the language model to unlearn domain-specific features. The domain identification head must resort to identifying the domain of the input data as well as possible given the domain invariant features that are propagated from the BERT language model. Updates to parameters of BERT, $\boldsymbol{\theta}$, are made with gradient descent according to,

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}_{NSP} - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}_{MLM} + \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}_{DI}$$

Note that gradients are subtracted along the next sentence prediction and masked language modeling losses but added along the domain identification loss. This sign change is the result of the aforementioned gradient reversal layer. While parameter updates descend the loss surface of the next sentence prediction and masked language modeling tasks, parameter updates instead ascend the loss surface of the domain identification task. This increases future domain identification losses, thereby promoting domain invariance in the language model.

It should be noted that each training example comes exclusively from either the domain-related or non-domain-related texts. In other words, the random pairing used for the next sentence prediction task never retrieves one sentence from the domain-related training subset and the other sentence from the opposite training subset. If the sentence pairing had been permitted to cross training subset boundaries in this way, the next sentence prediction task would have benefited from learning domain-specific features, thus opposing domain invariance training.

After each of the three epochs of domain invariance training, the model is trained and benchmarked on the legal text classification task.

4.5 Mixed Domain Finetuning

In order to isolate the performance effects of domain invariance training, an additional model is trained on the same mixed dataset but without the domain invariance task. This *mixed domain finetuning* functions as a control for studying the impact of domain invariance training.

After each of the three epochs of mixed domain finetuning, the model is trained and benchmarked on the legal text classification task.

4.6 Legal Pretraining

BERT's state-of-the-art performance comes at a cost. Its unprecedented results are in part due to its massive number of parameters. Because of these 110 million learnable parameters, pretraining from scratch is a costly and tedious endeavor. While avoiding a complete pretraining procedure is the point of domain adaptation, pretraining a BERT instance on legal data acts as an empirically derived upper bound on performance. This custom pretrained BERT model serves to approximate a perfect domain adaptation, where the model has captured all linguistic knowledge available in the domain-specific data. By comparing domain adapted BERT models to the custom pretrained model that constitutes an upper bound, it is possible to measure how much of BERT's maximum potential is unlocked through domain adaptation techniques.

The legal training procedure closely mimics the pretraining presented in the original BERT paper, save for four important differences that will now be discussed. Firstly, the original pretraining data from Wikipedia and Bookcorpus is replaced by a legal text corpus. This legal corpus is the same corpus that was used for domain finetuning. The second difference is the number of pretraining steps in the legal pretraining procedure. The original pretrained BERT model was pretrained on 10^6 batches each consisting of 256 sequences. Since the often formulaic language found in legal documents is assumed to be less varying than the text from Bookcorpus and Wikipedia, the legal pretraining procedure was decreased to $5 \cdot 10^5$ steps with a batch size of 256 sequences. The third difference is the pretraining hardware setup. BERT was originally pretrained on four cloud TPUs, while the custom pretrained BERT used in this study is pretrained on a single cloud TPU. The fourth and final difference between the original BERT pretraining and the legal pretraining model lies in the vocabulary, and warrants a detailed description.

As mentioned in section 3.5.2.1, BERT relies on the WordPiece tokenizer to split input sequences into tokens available in a fixed-size vocabulary. When some input word is not found in the vocabulary, the word is split with the hope that its constituent subwords exist in the vocabulary instead. The vocabulary is created by finding the list of words which can recreate some corpus with as few splits as possible. Since legal pretraining should exhibit the absolute best performance BERT can muster, a custom vocabulary is used. This vocabulary consists of 29 927 tokens that were selected to represent the legal corpus with as few token splits as possible.

Aside from the aforementioned four differences, legal pretraining is carried out identically to the original pretraining procedure. After the $5 \cdot 10^5$ legal pretraining steps, the model is benchmarked on the legal text classification task.

4.7 Run Configuration Summary

This section contains a succinct overview of model hyperparameters and run configurations (Table 4.2) and training data used for legal pretraining, domain adaptation and benchmarking runs (Figure 4.6).

	Learning Rate	**Loss Scale**	**Max Sequence Length**	**Batch Size**	**Dataset**
Legal Pretraining	$2 \cdot 10^{-4}$	128	256	256	Target Domain
Continued Pretraining	$3 \cdot 10^{-5}$			64	Source Domain
Domain Finetuning					Target Domain
Domain Invariance Training					Mixed Domain
Mixed Domain Finetuning					Mixed Domain
Legal Classification	$2 \cdot 10^{-5}$			32	Legal Classification

Table 4.2: A tabular summary of run configurations, hyperparameters and the datasets used for legal pretraining, the four domain adaptation methods and for the final benchmarking runs.

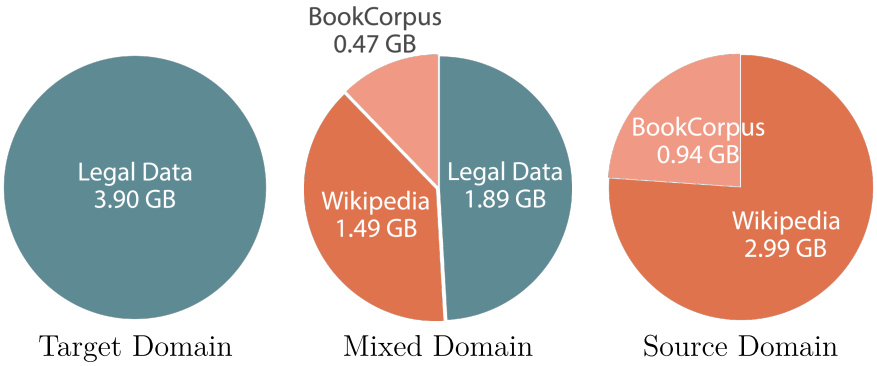


Figure 4.6: The three different datasets used in the legal pretraining, domain adaptation and continued pretraining methods.

5

Results

The purpose of this study is to benchmark various domain adapted instances of BERT on a classification task in the specialist domain of legal text. During benchmarking, each model is finetuned to legal classification on four differently sized subsamplings of the legal classification dataset, and subsequently evaluated on a test set of fixed size. The resulting performance measurements of these models are presented and discussed in this chapter.

5.1 General Findings

Figure 5.1 shows the different models' F_1 score on the test set after finetuning on training sets of different sizes. Note that the shown F_1 score is an average of the seven classwise F_1 scores (one for each provision).

A few general conclusions can be drawn from these results. Firstly, model performance is highly variant. So much so, that any effects of domain adaptation may be overtaken simply by variance in the training set, particularly if this training set is small. It is therefore difficult to guarantee any performance benefits from any of the domain adaptation methods. It is however possible to make statements concerning on-average model performance.

From these averages, it is visible that methods of domain adaptation or continued pretraining are most helpful when task-specific finetuning is performed on a smaller training dataset, while they are negligible with larger task-specific training sets. A larger performance increase in these small-data scenarios is to be expected, because a large task-specific training dataset allows the model to compensate for poor pretraining or poor domain adaptation. In other words, a large task-specific training dataset lets the model recoup task-relevant knowledge missed during the preceding training phases. Additionally, it is clearly visible that the largest performance gains occur after only one epoch of domain adaptation or continued pretraining. Since an epoch of domain adaptation drags on for approximately 45 hours (and upwards of 55 hours for domain invariance training), the fact that the first epoch results in the largest performance increase is relevant from a time-efficiency standpoint. In other words, if computational power or time are scarce commodities, a single domain adaptation epoch will yield a majority of the available performance gains.

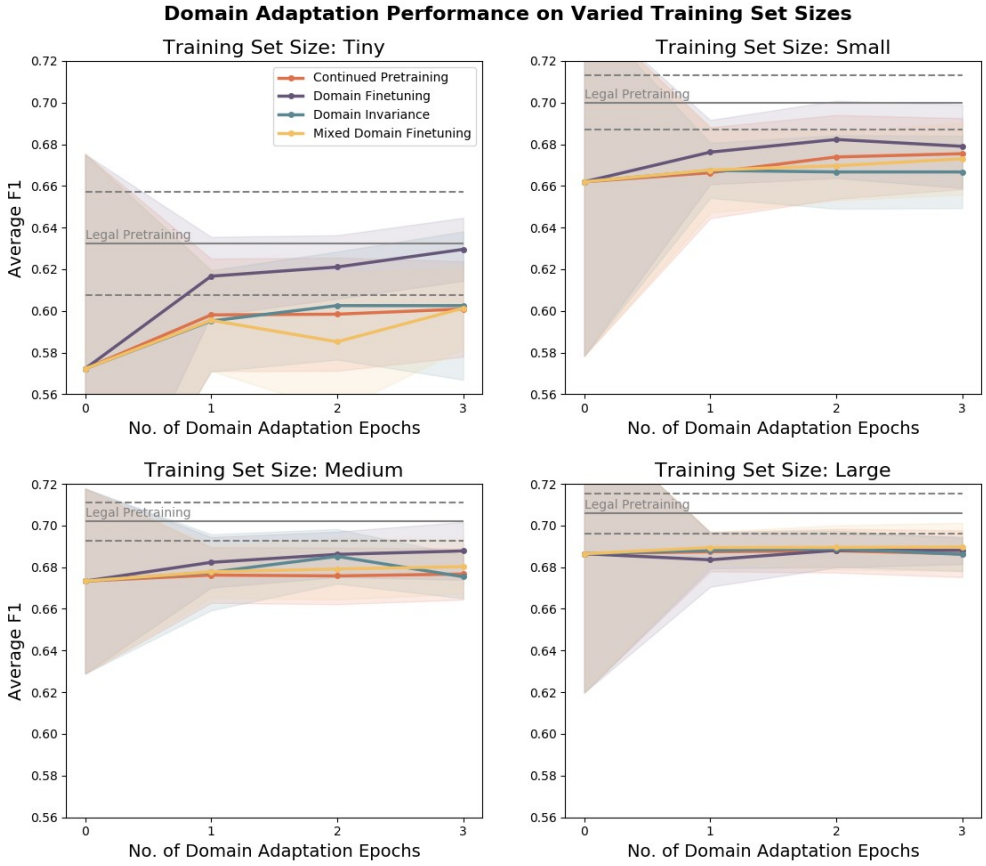


Figure 5.1: Average F_1 score on the legal classification task after training on four differently sized task-specific training sets. Each graph shows the legal classification performance after each epoch of domain adaptation or continued pretraining and subsequent task finetuning. The standard deviation is indicated by the shaded areas. The F_1 score for the legally pretrained model is visible as the grey line.

The are other less clear-cut findings in Figure 5.1. For instance, it would seem improbable that a heavily trained model such as the pretrained BERT instance would benefit from additional training on Wikipedia and Bookcorpus text. Contrary to this intuition, Figure 5.1 shows surprisingly substantial on-average performance gains from continued pretraining. Two factors could cause this to happen. Firstly, the reconstructed Wikipedia and Bookcorpus dataset could have differed from the original pretraining dataset. This is highly possible given that the original paper does not explain the steps taken to construct the original pretraining dataset. If substantial differences exist between the reconstructed and original datasets, the reconstructed dataset would have exposed the pretrained BERT model to previously unseen linguistic structures, allowing the model to further its linguistic knowledge. Another possible cause of the benefits of continued pretraining is that BERT simply had not converged during pretraining. This is corroborated by empirical studies on transformers, stating that transformers rarely converge or overfit on large datasets (see Chapter 2). In this case, it may be possible that continued pretraining (even

on the original pretraining datasets) allows BERT to uncover yet more knowledge.

There are further surprises evident in Figure 5.1. For example, continued pretraining, mixed domain training and domain invariance training have almost identical impacts on classification performance. This is unexpected because the mixed domain training and domain invariance training procedures, at the very least, expose BERT to some target domain-related legal text. Allowing BERT to train on this legal data should better prime the model for downstream tasks in the relevant legal domain, yet no improvements over continued pretraining are seen. This is doubly surprising given that domain finetuning (which only trains on data from the legal domain) does yield performance improvements. Thus, it can be concluded that the mere presence of source domain data in the domain adaptation phase can negate desired model adjustments made for the target domain.

5.2 Performance Per Provision

Examining the model on a per-provision basis shows that performance on individual provisions can in some cases decrease, while class average performance increases. For example, Figure 5.1 shows that the on-average performance of domain finetuning increases. However, as seen in Figure 5.2, domain finetuning does not necessarily increase performance for all provisions. In this scenario, average performance on the Indemnity provision increases by almost 0.1, while performance on Termination Convenience decreases by almost 0.05.

This disparity in performance could be due to the different levels of complexity and variation between the involved classes. For instance, the Indemnity provision contains an average of 1.71 unique words per example, while the Termination Convenience class contains 2.47 unique words per example. This hints at a more varied language usage within the Termination Convenience provision class. It can also be noted that 75% of Indemnity examples contain the token ‘indem’ (as found in words like ‘indemnity’, ‘indemnify’, ‘indemnification’ or ‘indemnitees’). This means that even a rudimentary string search for ‘indem’ will manage to correctly flag 75% of Indemnity examples. No such “tell-tale” string is found in the Termination Convenience class.

It is also worth noting that the Termination Cause and Termination Convenience classes were difficult even for the human annotators to correctly. As a result, the classification dataset has examples of these classes that have not been annotated. A model which has been specialized could possibly be more sensitive to these erroneous or missed labelings.

In summary, the hypothesis is that classification on highly variable textual data does not necessarily benefit from domain-specific training. Training procedures geared to the target domain could possibly specialize the model in a way that is detrimental to classifying the more varied classes. Noisy labeling in the downstream task dataset could further exacerbate this problem. While the above discussion is by no means a definitive proof, it does suggest that highly variable classes may be more difficult

to handle after a specializing domain adaptation. However, this hypothesis requires further investigation.

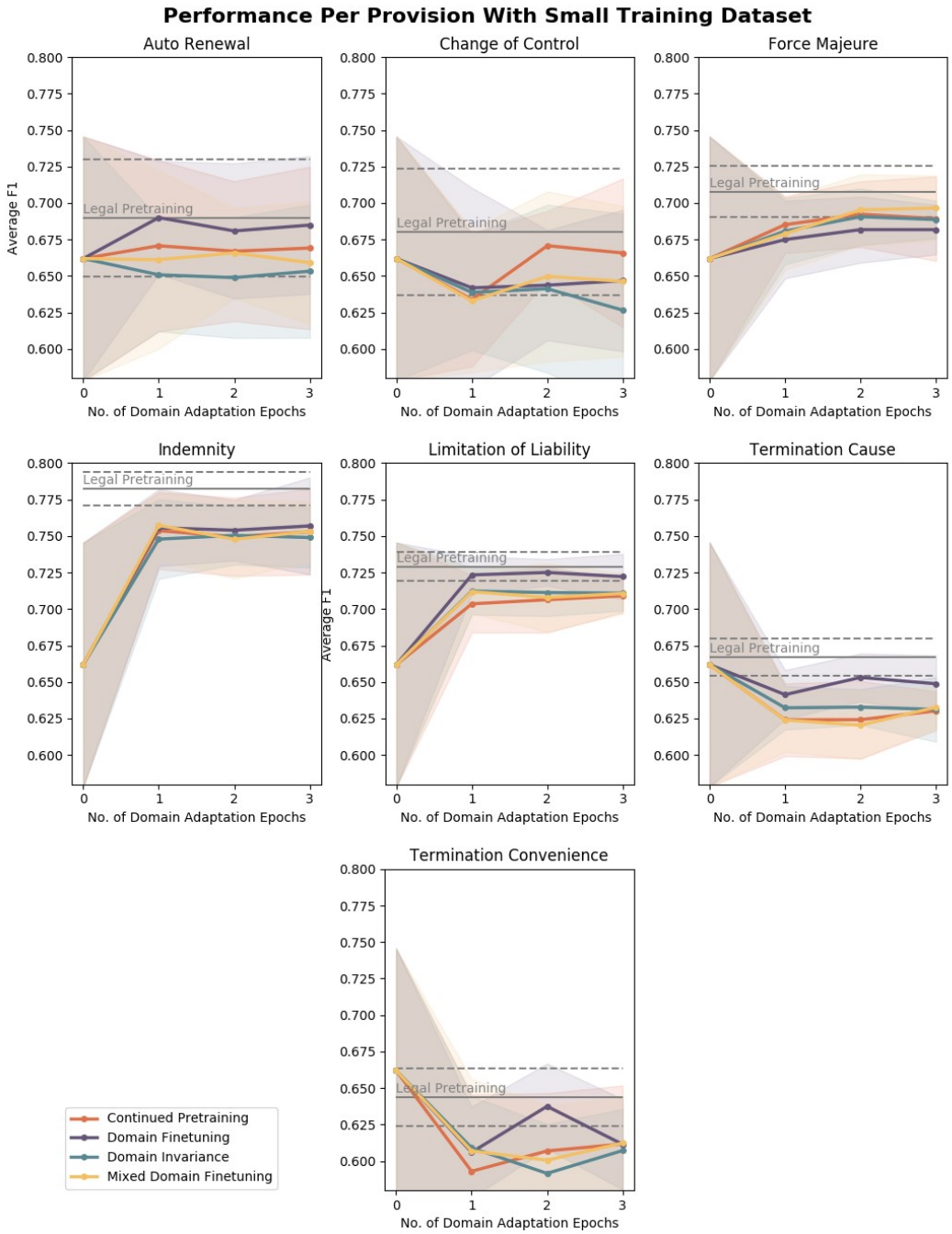


Figure 5.2: Performance per provision after performing task-specific training on the Small dataset.

5.3 Domain Invariance Training

Figure 5.1 shows that domain invariance training yielded no performance benefits over other methods, despite the promising results presented in the original paper. This section will examine possible causes of domain invariance training's lackluster results.

The most glaring difference between domain invariance training as presented in the original paper, and domain invariance training in this study, is that this study uses domain invariance training alongside pretraining tasks. The original paper presents domain invariance training as a supplementary procedure meant to be performed alongside the training of the task of interest. This enables models to gain knowledge from another domain thus learning a domain invariant classification rule. In contrast, training for domain invariance during language model pretraining may cause the model to ignore linguistic structures that may not be present in the domain-related data, but nevertheless could have been useful in downstream tasks. In short, ignoring any language features could be a destructive action. This is further corroborated by results indicating that even continued pretraining leads to performance gains on downstream tasks. It can thus be concluded that BERT is not picky with data used during pretraining.

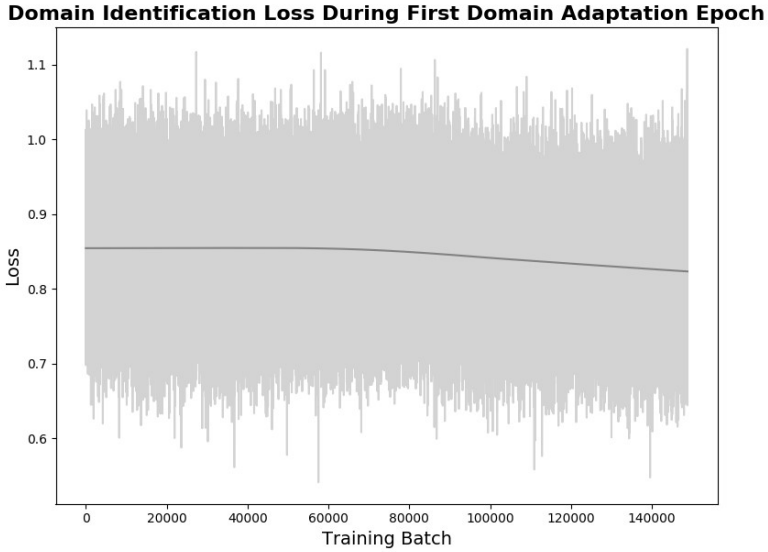


Figure 5.3: Batchwise domain identification loss during the first epoch of domain invariance training. The raw, per-batch loss values are shown in light grey. The LOWESS smoothing method has been applied to these raw values and visualized in dark grey in order to illustrate overall loss trends. Since training for domain invariance is synonymous to adversarially training for domain identification, the domain identification loss should grow during training. No such trend is discernible in this plot.

Figure 5.3 shows how the domain identification loss remains near-constant during training, instead of showing the increase that adversarial training aims for. This could further confirm that domain invariance and language modeling are two opposing goals. In other words, the two language modeling tasks could cancel out the domain invariance training in a sort of training tug-of-war. However, the non-moving losses may also be the result of the loss formulation used during domain invariance training. Recall, as presented in Section 4.4, that domain invariance training combines next sentence prediction, masked language modeling and domain identification into the loss function,

$$\mathcal{L} = \mathcal{L}_{NSP} + \mathcal{L}_{MLM} + \mathcal{L}_{DI}$$

which yields the parameter update rule,

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}_{NSP} - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}_{MLM} + \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}_{DI}$$

Note that the same learning rate, η, is used for all three training tasks. Using identical learning rates for all tasks may be problematic. Unlike next sentence prediction and masked language modeling, domain identification has not been trained for one million steps during pretraining. Therefore, domain invariance losses might remain virtually unchanged and show almost no effect on legal classification performance simply because domain invariance training is using a learning rate that is adapted for tasks that have undergone pretraining.

In summary, training for domain invariance shows no advantage over other domain adaptation techniques. This could be due to domain invariance training being incompatible with language modeling, or because domain invariance requires a separately tuned learning rate. In either case, this study finds no benefits to domain invariance training for BERT in the aforementioned experiment setup.

5.4 Practical Guidelines

To make the above results practically applicable, a general set of domain adaptation recommendations will now be presented. If seeking to adapt the pretrained BERT model to some specialist domain, it must first be established that the source domain (Wikipedia and Bookcorpus) and target domain are at least somewhat related. This means that source and target domains should show some overlap and, for instance, not stem from two completely different languages. If no relation exists between the domains, very little linguistic knowledge from the source domain will be applicable in the target domain. However, if this relation can be established, three factors dictate how BERT should be used in the relevant domain.

- Access to a large unlabelled corpus from the target domain
- Access to computing power
- Size of the labeled task-specific dataset

These three factors and their practical ramifications are presented in Figure 5.4.

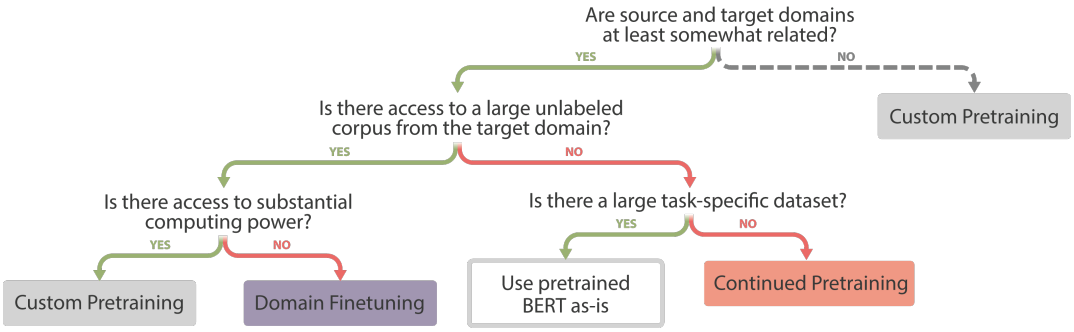


Figure 5.4: This figure shows how best to apply BERT in specialist language domains with respect to the available data and computing power. Note that the dashed line representing the case of unrelated source and target domains has not been formally investigated in this study. However, it would seem likely that a target domain completely unrelated to the language found in the Wikipedia and Bookcorpus pretraining corpora would require a custom pretraining procedure.

6

Conclusion

This study has focused on the performance of BERT on downstream classification tasks in the specialist language domain of legal text. A number of methods for adapting the pretrained BERT model to this language domain have been explored. The impact of domain adaptation techniques or custom pretraining are more pronounced when task-specific datasets are small. The study further finds that pretraining BERT from scratch will result in the best on-average performance on downstream tasks. However, finetuning for only a few epochs on relevant domain data will bring on-average downstream performance close to that of a custom model. Additionally, the study finds that continuing pretraining on (an approximation of) the original pretraining data can increase model performance in the specialist language domain. This is thought to be the result of transformer-based models' reluctance to converge on moderately large training datasets. Furthermore, the study examines the use of domain invariance training as a domain adaptation technique. This method shows no advantage over other, simpler domain adaptation techniques, like pretraining on domain-related data. This nonbeneficial effect is presumed to either be the result of training a language modeling task adversarially, or the result of an ill-tuned learning rate. This destructive influence obstructs the model's ability to glean as much knowledge from the corpora as possible, leading to lackluster performance on downstream tasks. Finally, the study suggests practical guidelines for applying BERT in a specialist language domain.

Bibliography

Ambartsoumian, A. and Popowich, F. (2018), Self-attention: A better building block for sentiment analysis neural network classifiers, *in* 'Proceedings of the 9th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis', pp. 130–139.

Baxter, J. (2000), 'A model of inductive bias learning', *Journal of Artificial Intelligence Research* **12**, 149–198.

Beltagy, I., Cohan, A. and Lo, K. (2019), 'Scibert: Pretrained contextualized embeddings for scientific text', *arXiv preprint arXiv:1903.10676* .

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.

Bishop, C. M. (2006), *Pattern Recognition and Machine Learning*, Springer, pp. 209–210.

Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K. (2019), BERT: Pre-training of deep bidirectional transformers for language understanding, *in* 'Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)', Association for Computational Linguistics, Minneapolis, Minnesota, pp. 4171–4186.
URL: *https://www.aclweb.org/anthology/N19-1423*

Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M. and Lempitsky, V. (2016), 'Domain-adversarial training of neural networks', *The Journal of Machine Learning Research* **17**(1), 2096–2030.

Goodfellow, I., Bengio, Y. and Courville, A. (2016), *Deep Learning*, MIT press, p. 83.

He, K., Zhang, X., Ren, S. and Sun, J. (2016), Deep residual learning for image recognition, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition', pp. 770–778.

Hinton, G. E., Rumelhart, D. and Williams, R. J. (1986), 'Learning representations by back-propagating errors', *Nature* **323**(9), 533–536.

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 (Volume 1: Long Papers)', pp. 328–339.

Jiang, J. and Zhai, C. (2007), 'An empirical study of tokenization strategies for biomedical information retrieval', *Information Retrieval* **10**(4-5), 341–363.

Kingma, D. P. and Ba, J. (2015), 'Adam: A method for stochastic optimization', *CoRR* **abs/1412.6980**.

Kudo, T. and Richardson, J. (2018), Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing, *in* 'Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations', pp. 66–71.

Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H. and Kang, J. (2019), 'Biobert: a pre-trained biomedical language representation model for biomedical text mining', *arXiv preprint arXiv:1901.08746* .

Lin, Z., Feng, M., Santos, C. N. d., Yu, M., Xiang, B., Zhou, B. and Bengio, Y. (2017), 'A structured self-attentive sentence embedding', *arXiv preprint arXiv:1703.03130* .

Liu, X., He, P., Chen, W. and Gao, J. (2019), 'Multi-task deep neural networks for natural language understanding', *CoRR* **abs/1901.11504**.

Moore, R. C. and Lewis, W. (2010), Intelligent selection of language model training data, *in* 'Proceedings of the ACL 2010 Conference Short Papers', Association for Computational Linguistics, pp. 220–224.

Pan, S. J. and Yang, Q. (2010), 'A survey on transfer learning', *IEEE Transactions on Knowledge and Data Engineering* **22**(10), 1345–1359.

Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K. and Zettlemoyer, L. (2018), Deep contextualized word representations, *in* 'Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)', pp. 2227–2237.

Popel, M. and Bojar, O. (2018), 'Training tips for the transformer model', *The Prague Bulletin of Mathematical Linguistics* **110**(1), 43–70.

Radford, A., Narasimhan, K., Salimans, T. and Sutskever, I. (2018), 'Improving language understanding by generative pre-training'.
URL: *https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/languageunsupervised/language_understanding_paper.pdf*

Simonyan, K. and Zisserman, A. (2014), 'Very deep convolutional networks for large-scale image recognition', *arXiv preprint arXiv:1409.1556* .

Stickland, A. C. and Murray, I. (2019), BERT and PALs: Projected attention layers for efficient adaptation in multi-task learning, *in* K. Chaudhuri and R. Salakhutdinov, eds, 'Proceedings of the 36th International Conference on Machine Learning',

Vol. 97 of *Proceedings of Machine Learning Research*, PMLR, Long Beach, California, USA, pp. 5986–5995.
URL: *http://proceedings.mlr.press/v97/stickland19a.html*

Tenney, I., Das, D. and Pavlick, E. (2019), 'Bert rediscovers the classical nlp pipeline', *arXiv preprint arXiv:1905.05950* .

Torrey, L. and Shavlik, J. (2010), Transfer learning, *in* 'Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques', IGI Global, pp. 242–264.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. and Polosukhin, I. (2017), Attention is all you need, *in* 'Advances in Neural Information Processing Systems', pp. 5998–6008.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K. et al. (2016), 'Google's neural machine translation system: Bridging the gap between human and machine translation', *arXiv preprint arXiv:1609.08144* .

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R. and Bengio, Y. (2015), Show, attend and tell: Neural image caption generation with visual attention, *in* 'International Conference on Machine Learning', pp. 2048–2057.