CS584: Natural Language Processing **Spring 2024**

Lecture: Pretraining and Finetuning

Lecturer: Yue Ning *Scribe: Akshay Parate*

1.1 Introduction

Are you fascinated by how large language models (LLMs) such as BERT and GPT series models operate? Have you ever pondered over their ability to comprehend human language with such accuracy? What are the pivotal stages that elevate them from mere neural networks to proficient tools capable of tasks like text prediction and sentiment analysis?

The key lies in two essential phases: pre-training and fine-tuning. These stages not only enable language models to be versatile across various tasks but also bring them closer to mimicking human-like language understanding.

Pretraining entails training a model on an extensive dataset to grasp general features and representations. This step aids the model in capturing the underlying structure of the data, laying a robust groundwork for subsequent learning. Fine-tuning, conversely, involves customizing the pretrained model to a specific task using a smaller, task-oriented dataset. Through this process, the model refines its understanding and enhances its performance on the targeted task.

1.1.1 Significance of Understanding LLMs

A large language model is a computer program that learns and generates human-like language using a transformer architecture trained on vast text data.

Large Language Models (LLMs) are foundational machine learning models that use deep learning algorithms to process and understand natural language. These models are trained on massive amounts of text data to learn patterns and entity relationships in the language. LLMs can perform many types of language tasks, such as translating languages, analyzing sentiments, chatbot conversations, and more. They can understand complex textual data, identify entities and relationships between them, and generate new text that is coherent and grammatically accurate.

1.1.2 Aim to elucidate the roles of pre-training and fine-tuning in LM.

Pre-training and fine-tuning are complementary stages in the journey of LLM development, each serving a distinct yet interconnected purpose. Pre-training provides the model with a broad understanding of language, while fine-tuning tailors this understanding to specific tasks, ultimately enabling LLMs to achieve remarkable levels of language comprehension and performance across a wide range of applications. Through this exploration, we aim to unravel the mechanisms underlying these phases and illuminate their crucial roles in advancing the field of natural language processing.

1.2 Background

The evolution and capabilities of large language models (LLMs) have evolved rapidly, emphasizing the diverse landscape of machine learning. These sophisticated models, capable of understanding, generating, and interpreting human language, are at the forefront of AI research and application. The training methodologies are crucial to the effectiveness of these models.This lecture delves into the two predominant approaches: pre-training and fine-tuning.

1.2.1 A brief history of LLMs

The foundation of large language models can be traced back to experiments with neural networks and neural information processing systems that were conducted in the 1950s to allow computers to process natural language. Researchers at IBM and Georgetown University worked together to create a system that would be able to automatically translate phrases from Russian to English. As a notable demonstration of machine translation, research in this field took off from there.

The idea of LLMs was first floated with the creation of Eliza in the 1960s: it was the world's first chatbot, designed by MIT researcher Joseph Weizenbaum. Eliza marked the beginning of research into natural language processing (NLP), providing the foundation for future, more complex LLMs.

Then almost 30+ years later, in 1997, Long Short-Term Memory (LSTM) networks came into existence. Their advent resulted in deeper and more complex neural networks that could handle greater amounts of data. Stanford's CoreNLP suite, introduced in 2010, was the next stage of growth allowing developers to perform sentiment analysis and named entity recognition.

Subsequently, in 2011, a smaller version of Google Brain appeared with advanced features such as word embeddings, which enabled NLP systems to gain a clearer understanding of context. This was a significant turning point, with transformer models bursting onto the scene in 2017. Think GPT, which stands for Generative Pre-trained Transformer, has the ability to generate or "decode" new text. Another example is BERT - Bidirectional Encoder Representations from Transformers. BERT can predict or classify input text based on encoder components.

From 2018 onward, researchers focused on building increasingly larger models. It was in 2019 that researchers from Google introduced BERT, the two-directional, 340-million parameter model (the third largest model of its kind) that could determine context allowing it to adapt to various tasks. By pre-training BERT on a wide variety of unstructured data via self-supervised learning, the model was able to understand the relationships between words. In no time at all, BERT became the go-to tool for natural language processing tasks. In fact, it was BERT that was behind every English-based query administered via Google Search.

1.2.2 Transfer learning and supervised pre-training

The early efforts of pre-training are mainly involved in transfer learning . The study of transfer learning is heavily motivated by the fact that people can rely on previously learned knowledge to solve new problems and even achieve better results. More formally, transfer learning aims to capture important knowledge from multiple source tasks and then apply the knowledge to a target task.

In transfer learning, source tasks and target tasks may have completely different data domains and task settings, yet the knowledge required to handle these tasks is consistent . It is thus important to select a feasible method to transfer knowledge from source tasks to target tasks. To this end, various pre-training methods have been proposed to work as the bridge between source and target tasks. Specifically, these

methods first pre-train models on the data of multiple source tasks to pre-encode knowledge and then transfer the pre-encoded knowledge to train models for target tasks.

Generally, two pre-training approaches are widely explored in transfer learning: feature transfer and parameter transfer. Feature transfer methods pre-train effective feature representations to pre-encode knowledge across domains and tasks . By injecting these pre-trained representations into target tasks, model performance of target tasks can be significantly improved. Parameter transfer methods follow an intuitive assumption that source tasks and target tasks can share model parameters or prior distributions of hyper-parameters. Therefore, these methods pre-encode knowledge into shared model parameters, and then transfer the knowledge by fine-tuning pre-trained parameters with the data of target tasks.

To some extent, both representation transfer and parameter transfer lay the foundation of PTMs. Word embeddings, widely used as the input of NLP tasks, are built on the framework of feature transfer. Inspired by parameter transfer, pre-trained CNNs are applied as the backbone of most state-of-the-art CV models. Some recent well-known PTMs are also based on representation transfer and parameter transfer, e.g., ELMo (Peters et al., 2018) and BERT apply representation transfer and parameter transfer respectively.

Since AlexNet , a series of deep neural networks have been developed for AI tasks. As compared with those conventional machine learning models, deep neural models have more parameters and show better capabilities of fitting complex data. Therefore, from AlexNet to later VGG and GoogleNet, the architecture of these neural networks becomes deeper and deeper, and their performance accordingly becomes better and better. Although the network depth is important, training a deep network is not easy, as stacking more network layers inevitably brings the problem of vanishing or exploding gradients. Besides the gradient issues, model performance may soon meet a ceiling and then degrade rapidly with continually increasing network depths.

By adding normalization to parameter initialization and hidden states, and introducing shortcut connections with residual layers, ResNet effectively tackles these problems. As we mentioned before, deep neural networks require large amounts of data for training. To provide sufficient data to train deep models, some large-scale supervised datasets have also been built, and the most representative one is ImageNet. ImageNet contains millions of images divided into thousands of categories, representing a wide variety of everyday objects. Based on the combination of effective model ResNet, informative dataset ImageNet, as well as mature knowledge transfer methods, a wave of pre-training models on labeled data emerges.

1.2.3 Self-supervised learning and self-supervised pre-training

As shown in Fig. 1, transfer learning can be categorized under four sub-settings, inductive transfer learning, transductive transfer learning, self-taught learning and unsupervised transfer learning.

Among these four settings, the inductive and transductive settings are the core of research, as these two settings aim to transfer knowledge from supervised source tasks to target tasks. Although supervised learning is always one of the core issues of machine learning research, the scale of unlabeled data is much larger than that of manually labeled data. Recently, more and more researchers have noticed the importance of large-scale unlabeled data and are committed to extracting information from unlabeled data. Self-supervised learning has been proposed to extract knowledge from large-scale unlabeled data by leveraging input data itself as supervision.

Self-supervised learning and unsupervised learning have many similarities in their settings. To a certain extent, self-supervised learning can be regarded as a branch of unsupervised learning because they both apply unlabeled data. However, unsupervised learning mainly focuses on detecting data patterns.

The development of self-supervised learning makes it possible to perform pre-training on large-scale unsupervised data. Compared to supervised pre-training working as the cornerstone of CV in the deep learning era, self-supervised pre-training allows for huge advances in the field of NLP. Although some supervised

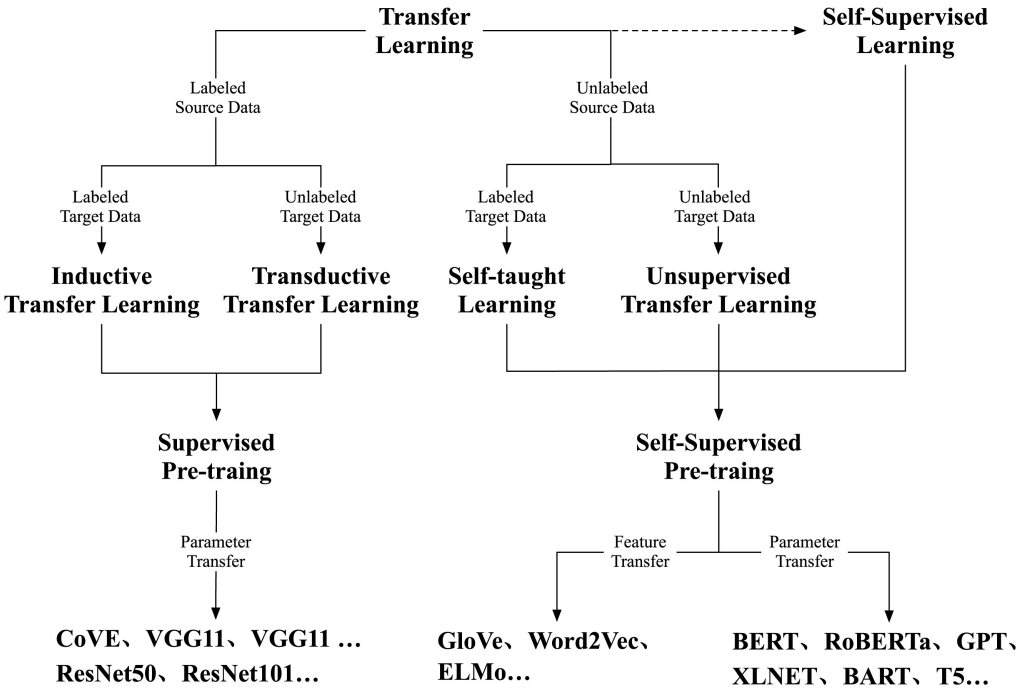

Figure 1.1: The spectrum of pre-training methods from transfer learning, self-supervised learning to the latest pre-training neural models..

pre-training methods like CoVE have achieved promising results on NLP tasks, it is nearly impossible to annotate a textual dataset as large as ImageNet, considering annotating textual data is far more complex than annotating images. Hence, applying self-supervised learning to utilize unlabeled data becomes the best choice to pre-train models for NLP tasks. The recent stunning breakthroughs in PTMs are mainly towards NLP tasks, more specifically pre-trained language models.

1.3 Model : Transformer and representative PTMs

As we mentioned before, the key to the success of recent PTMs is an integration of self-supervised learning and Transformer. Hence, this section begins with the dominant basic neural architecture, Transformer. Then, we will introduce two landmark Transformer-based PTMs, GPT and BERT. These two PTMs respectively use autoregressive language modeling and autoencoding language modeling as pre-training objectives. All subsequent PTMs are variants of these two models. The final part of this section gives a brief review of typical variants after GPT and BERT to reveal the recent development of PTMs.

Before Transformer, RNNs have long been a typical tool for processing sequential data, especially for processing natural languages. As RNNs are equipped with sequential nature, they read a word at each time step in order. For each word, RNNs refer to all hidden states of its previous words to process it. Such a mechanism is considered to be difficult to take advantage of the parallel capabilities of high-performance computing devices such as GPUs and TPUs.

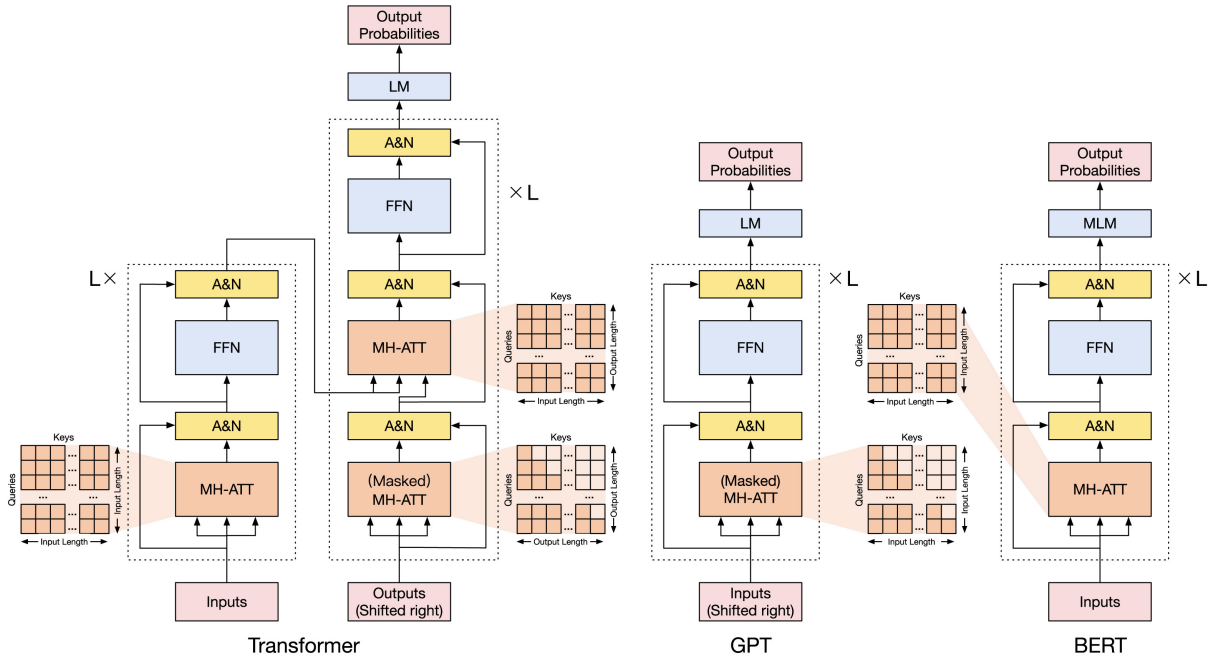

Figure 1.2: The architecture of Transformer, GPT, and BERT.

1.3.1 Transformer

As shown in Fig. 5, Transformer is a non-recurrent sequence-to-sequence (seq2seq) architecture consisting of an encoder and a decoder. The encoder and decoder of a Transformer are both stacked by several identical blocks. Each encoder block is composed of a multi-head self-attention layer and a position-wise feed-forward layer. Compared with the encoder block, each decoder block has an additional cross-attention layer since the decoder requires to consider the output of the encoder as a context for generation. Between neural layers, residual connection and layer normalization are employed, making it possible to train a deep Transformer.

As shown in Fig. 5, there are three variants of the multi-head attention in Transformer:

(1) Self-attention is used in the encoder, which uses the output of the previous layer as Q, K, V. In the encoding phase, given a word, the self-attention computes its attention scores by comparing it with all words in the input sequence. And such attention scores indicate how much each of the other words should contribute to the next representation of the given word. We give an example in Fig 1.3, where the self-attention accurately captures the referential relationships between “Jack” and “he”, generating the highest attention score.

(2) Masked self-attention is used in the decoder, whose attention matrix satisfies Aij=0, i¿j. This attention is beneficial to autoregressive language modeling. In the decoding phase, the self-attention is similar to the encoding, except that it only decodes one representation from left to right at one time. Since each step of the decoding phase only consults the previously decoded results, we thus require to add the masking function into the self-attention.

(3) Cross-attention is also used in the decoder, which uses the output of the previous decoder block as well as the output of the encoder. Such a procedure is essentially an aggregation of the information of the whole input sequence, and it will be applied to all the words to generate in the decoding phase.

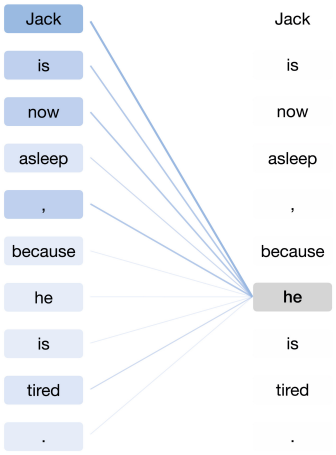

Figure 1.3: An illustration of the self-attention mechanism of Transformer. The figure shows the self-attention results when encoding the word "he", where the darker the color of the square is, the larger the corresponding attention score is.

1.3.2 GPT

PTMs typically consist of two phases, the pre-training phase and the fine-tuning phase. Equipped by the Transformer decoder as the backbone,5 GPT applies a generative pre-training and a discriminative finetuning. Theoretically, compared to precedents of PTMs, GPT is the first model that combines the modern Transformer architecture and the self-supervised pre-training objective. Empirically, GPT achieves significant success on almost all NLP tasks, including natural language inference, question answering, commonsense reasoning, semantic similarity and classification.

Given large-scale corpora without labels, GPT optimizes a standard autoregressive language modeling, that is, maximizing the conditional probabilities of all the words by taking their previous words as contexts. In the pre-training phase of GPT, the conditional probability of each word is modeled by Transformer

The adaptation procedure of GPT to specific tasks is fine-tuning, by using the pre-trained parameters of GPT as a start point of downstream tasks. In the fine-tuning phase, passing the input sequence through GPT, we can obtain the representations of the final layer of the GPT Transformer. By using the representations of the final layer and task-specific labels, GPT optimizes standard objectives of downstream tasks with simple extra output layers. As GPT has hundreds of millions of parameters, it is trained for 1 month on 8 GPUs, which is fairly the first "large-scale" PTM in the history of NLP. And undoubtedly, the success of GPT pave the way for the subsequent rise of a series of large-scale PTMs. In the next part, we will introduce another most representative model BERT.

1.3.3 BERT

The emergence of BERT has also greatly promoted the development of the PTM field. Theoretically, compared with GPT, BERT uses a bidirectional deep Transformer as the main structure. There are also two separate stages to adapt BERT for specific tasks, pre-training and fine-tuning. In the pre-training phase, BERT applies autoencoding language modeling rather than autoregressive language modeling used in GPT. More specifically, inspired by cloze (Taylor, 1953), the objective masked language modeling (MLM) is designed. As shown in Fig. 7, in the procedure of MLM, tokens are randomly masked with a special token , the objective is to predict words at the masked positions with contexts. Compared with standard

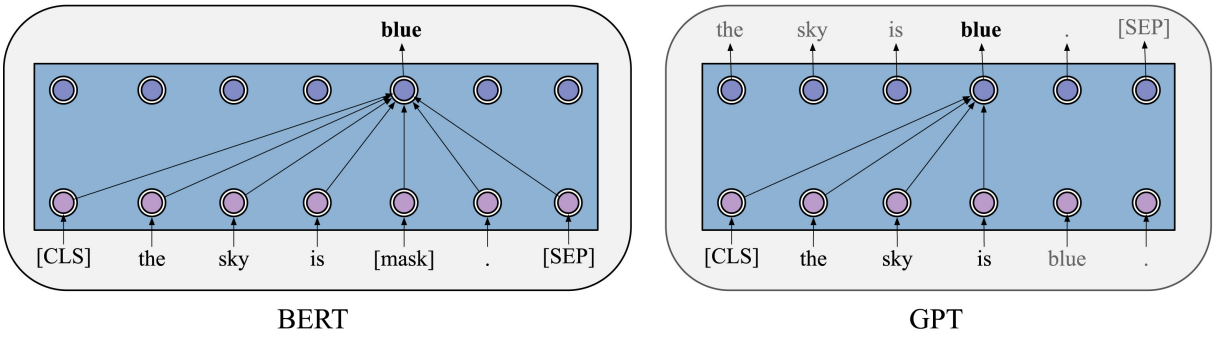

Figure 1.4: The difference between GPT and BERT in their self-attention mechanisms and pre-training objectives.

unidirectional autoregressive language modeling, MLM can lead to a deep bidirectional representation of all tokens.

Besides MLM, the objective of next sentence prediction (NSP) is also adopted to capture discourse relationships between sentences for some downstream tasks with multiple sentences, such as natural language inference and question answering. For this task, a binary classifier is used to predict whether two sentences are coherent. In the pre-training phase, MLM and NSP work together to optimize the parameters of BERT.

After pre-training, BERT can obtain robust parameters for downstream tasks. By modifying inputs and outputs with the data of downstream tasks, BERT could be fine-tuned for any NLP tasks. As shown in Fig. 8, BERT could effectively handle those applications with the input of a single sentence or sentence pairs. For the input, its schema is two sentences concatenated with the special token , which could represent: (1) sentence pairs in paraphrase, (2) hypothesis-premise pairs in entailment, (3) question-passage pairs in question answering, and (4) a single sentence for text classification or sequence tagging. For the output, BERT will produce a token-level representation for each token, which can be used to handle sequence tagging or question answering, and the special token [CLS] can be fed into an extra layer for classification. After GPT, BERT has further achieved significant improvements on 17 different NLP tasks, including SQuAD (better than human performance), GLUE (7.7

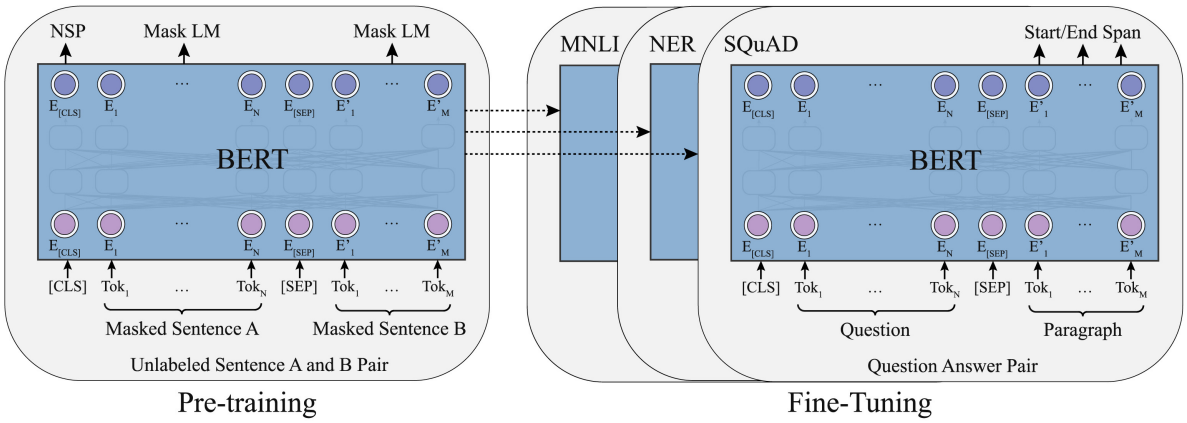

Figure 1.5: The pre-training and fine-tuning phases for BERT.

1.4 Fine-tuning Strategies

We'll talk about two different fine-tuning strategies. Usually, we combine them in practice.

1.4.1 Freezing Layers

The most common approach is to freeze the layers. What does that mean?

Naturally, we can initialize a neural network with pre-trained weights and let all weights be adjusted with the new training. One way to address the issue is to freeze only the first few layers and conduct the training

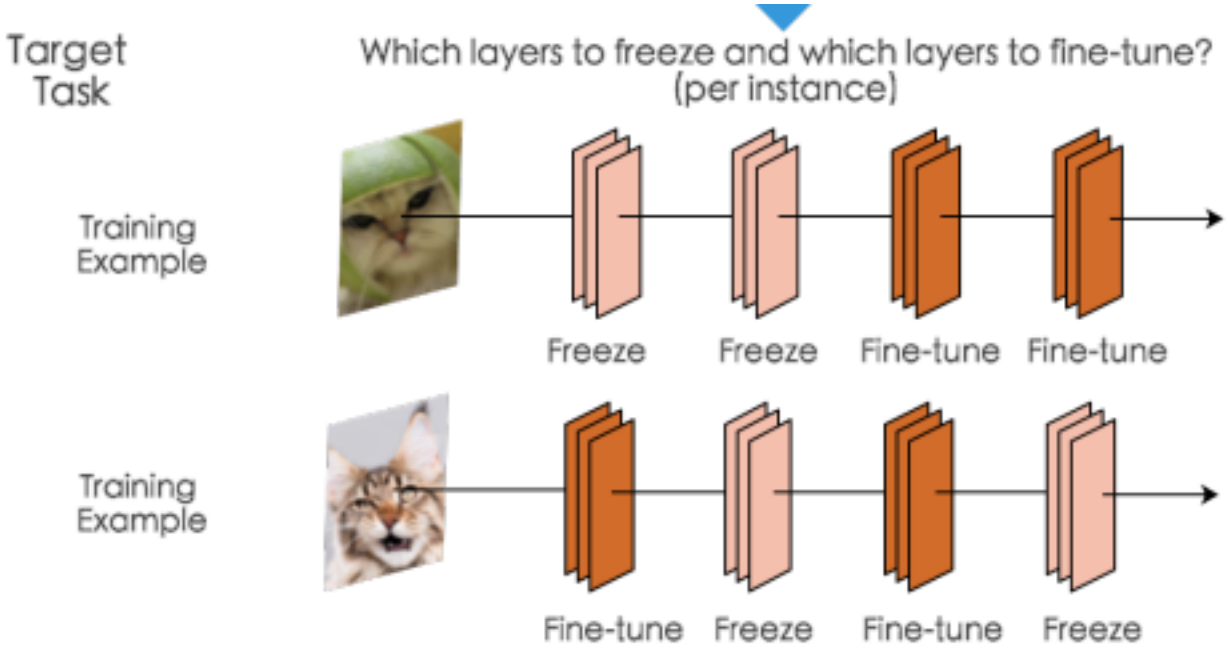

Figure 1.6: The pre-training and fine-tuning phases for BERT.

for the others. Freezing the initial layer relies on the fact that in the initial stages, the network is learning basic features. This is exactly what we want to extract when we implement the fine-tuning. Alternatively, we can freeze all layers except the last one, whose weights we adjust during training.

Of course, we may need to change the output layer if, e.g., the old network was discriminating between two classes, but we have more classes in the current problem. The same goes for the first layer if the numbers of input features differ.

Finally, we can add several layers on top of the old architecture. That way, we keep all the old weights and change only the weights of new layers.

1.4.2 Gradual Unfreezing

In this strategy, instead of keeping all layers frozen except the last one (as in freezing layers), you gradually unfreeze layers starting from the top and moving downwards. The idea behind this approach is to allow the

model to fine-tune its parameters progressively, enabling it to adapt to the specific task while retaining the general knowledge learned during pre-training.

Here's how gradual unfreezing typically works:

Initially, all layers except the final layer are frozen, allowing only the final layer to be fine-tuned on the task-specific data. After a certain number of epochs or when the performance plateaus, the topmost frozen layer is unfrozen, and both the unfrozen layer and the final layer are fine-tuned together while keeping the rest of the layers frozen. This process continues iteratively, gradually unfreezing lower layers and fine-tuning them along with the previously unfrozen layers, until all layers have been fine-tuned. Gradual unfreezing enables the model to leverage the pre-trained weights across multiple layers while also adapting the model to the specific nuances of the target task. This approach often leads to better performance compared to freezing only the lower layers or fine-tuning all layers simultaneously, especially when dealing with limited task-specific data.

1.5 Conclusion

In this lecture, we take a look into the history of pre-training to indicate the core issue of PTMs, and meanwhile reveal the crucial position of PTMs in the AI development spectrum. Furthermore, we comprehensively review the latest efforts towards better PTMs, including designing effective architectures, utilizing rich contexts, improving computational efficiency, and conducting interpretation and theoretical analysis. All these works contribute to the recent wave of developing PTMs. Although existing PTMs have achieved promising results, especially those large-scale PTMs showing amazing abilities in zero/few-shot learning scenarios, how to develop PTMs next is still an open question. The knowledge stored in PTMs is represented as real-valued vectors, which is quite different from the discrete symbolic knowledge formalized by human beings. We name this continuous and machine-friendly knowledge "modeledge" and believe that it is promising to capture the modeledge in a more effective and efficient way and stimulate the modeledge for specific tasks. We hope our view could inspire more efforts in this field and advance the development of PTMs.

[5] [1] [4] [3] [2]

References

[1] S. Hore. What are large language models (llms)? *https://www.analyticsvidhya.com/blog/2023/03/an-introduction-to-large-language-models-llms/: :text=Analysis*

[2] A. Kumar. Pre-training vs fine-tuning in llm. https://vitalflux.com/pre-training-vs-fine-tuning-in-llm-examples/.

[3] A. Patel. Pre-training vs. fine-tuning large language models. https://www.ankursnewsletter.com/p/pre-training-vs-fine-tuning-large.

[4] T. ValizadehAslani. Pretraining and fine-tuning. https://www.activeloop.ai/resources/glossary/pretraining-and-fine-tuning/.

[5] Z. Z. Xu Han. Pre-trained models: Past, present and future. https://www.sciencedirect.com/science/article/pii/S2666651021000231bib188.