CS182 / CS282 - UC Berkeley

Lecture 21: Meta-learning, Fine-Tuning, Transfer Learning

3rd November, 2022

Lecturer: Prof. Anant Sahai *Scribes: Aman Saraf, Tianlun Zhang*

Reviewers: Kiran Eiden, Nikhil Prakash, Jiayang Nie

1 Lecture Overview

The context for this lecture is based on a exploration of transformer based language models which typically have very large parameter sets. This lecture will seek to explore strategies for fine-tuning and then come back to look at pre-training.

2 Fine Tuning

The overall approach to fine-tuning is as follows:

- **Step 1:** Pretrain a large model with self-supervision and lots of data. (***Classical ML Analog***: PCA on unsupervised data)
- **Step 2:** Fine-tune the model on task specific data set or objective. (***Classical ML Analog***: Train a classifier or regression model on data with reduced dimensionality)

Next we look at some strategies to fine-tune effectively, considering the benefits and limitations of each of them.

2.1 Strategies to Fine Tune

Let's start by considering the BERT-style model [1], which is pre-trained with two tasks:

- Sentence Order
- Fill in the blank (Masked Denoising Autoencoder)

2.1.1 Decapitate training "head" and *only* train a new "head"

Note: Here "head" refers to the part doing the surrogate task, and not a transformer "head".

In this strategy, we remove just the surrogate task layers from the end of the model and replace them with another newly initialized head. This head should be well suited to our task and usually comes from a similar domain to the surrogate tasks that the original model was trained on.
This new head now operates on features extracted from our task's dataset by the backbone for our

model. This is like lifting the feature space and treating the output before the surrogate head as features we want to run our task on.
For example we could:

1. Replace the final linear layer and softmax for the final sentence order task with a new linear + softmax layer for your own specific sentence-level task. We keep every other layer intact.
2. Train the new final layer, using Stochastic Gradient Descent (or any optimizer of your choice) for your specific task.

Note: In this approach we treat BERT as a feature map, where the last layers provide features.

Here we prompt the model using the token for sentence order classification. Remember that for BERT the input is as follows:

<classification token> sentence 1 <separation token> sentence 2

where sentence 1 and sentence 2 are masked. Masking is done by replacing a word by the special mask token, and therefore removal of prompt words requires no scaling like in dropout. This is also possible since tokens are discrete whereas in real number spaces having mask tokens is more difficult.

2.1.2 Variation: Use other BERT Embeddings

The choice of using features/embeddings from the last layer is not fixed. We can use other embeddings that BERT gives us. For example:

- Average together all the layers from the transformer.
- Concatenate the last four layers (arbitrary)
- Average together the last four layers (arbitrary)
- Use the last $N - 1$ layers
- and so on...

Similarly, if the task is "word-level" then we can use BERT as a word-level feature extractor as well.

Notice that here the language model almost behaves like a feature extractor, and exploring these strategies to combine features or choose the extraction backbone, is almost like a hyper-parameter search problem.

2.1.3 *Major* Variation: Train the entirety of the model after replacing head

Previously, we were only training the final layer and everything else was frozen. In this strategy, we will let **all** the weights adapt to our new task by training all layers at once.

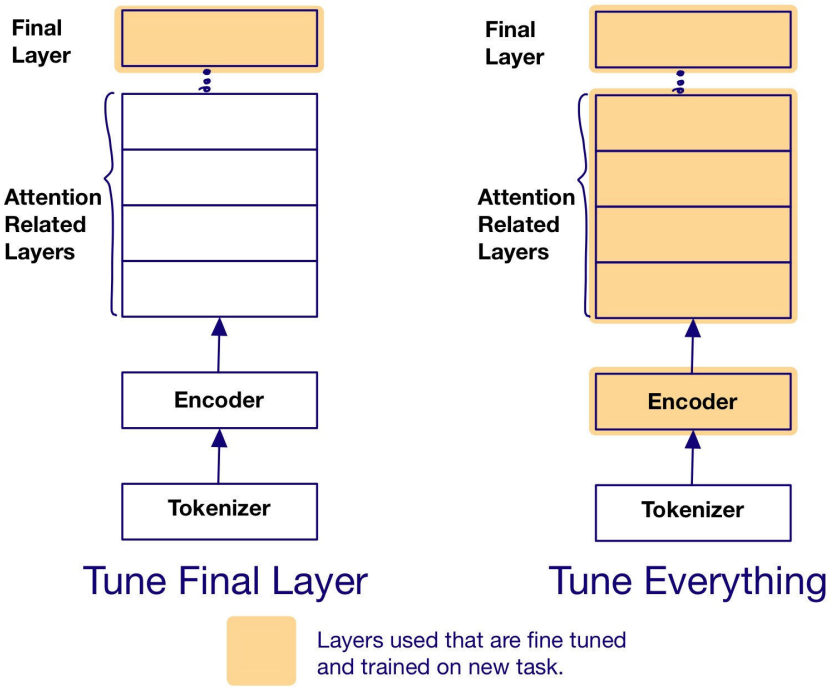

Figure 1: Illustration of different strategies of for fine tuning.

This strategy has a few advantages and many disadvantages, as follows:

- Pro:
  - **Increased Capacity:** We provide more capacity in the model, to fit our new tasks. This is obvious as we are training many more weights now in an over-parameterized model.
- Cons:
  - **Fear of Overfitting:** Excessive parameters leave us open to this risk.
  - **Harder to Scale:** to lots of different tasks - for example, if we only train the individual heads for each of our tasks, we only store each individual head, whereas if we allow the entire model to train we need to store a version of the entire model for each task.
  - **Divergent Backbone Gradients for different tasks:** If trained end to end on different tasks, the various gradients can start pushing the backbone in all sorts of directions, often divergent from one another.

2.2 Interesting Questions:

2.2.1 How is it possible, that "pre-training + fine-tuning" works, when just fine-tuning from random initialization does not work? Is not the model equally large in both cases and heavily over-parameterized?

There are two explanations for this behavior, although some questions still remain:

- **Explanation 1**: Somehow pre-training gets us to a "fortunate" initial condition. This initial condition is particularly suited for fine-tuning.
- **Explanation 2**: Because we have so many parameters, it's as though we are training a generalized linear model using "derivative" features.This works because the principal features of the generalized linear model are aligned to this family of tasks.

2.2.2 Is it necessary to replace the head with a linear + softmax layer or can we use other layers like Random Forests?

Intuitively should work similarly to how random forests work for other problems. Need to try it out.

2.2.3 Are there certain architectures for backbones and head combinations that when pre-trained adapt well to head replacement for other tasks, or that allow for training with multiple different heads without degrading performance for each other?

Seems like an open question.

3 Emergent Useful Behaviour in Large Language Models

Interesting new behaviors were observed in large language models like word2vec and Generative Pre-trained Transformer(GPT) [2]:

3.1 Word2Vec:

It turns out the geometry of learning word embeddings can capture regularities of the meanings. For example, using the difference between two words' embeddings learned by word2vec in solving the "analogies" problem, can get better results than guessing without any additional training:

Solving "analogies" example:
Input: ("Man", "Woman") ("King", "?")
Predict(output) : "?" — the right answer should be "Queen"

Here the strategy for extracting the analogy answer without training on this task was to use the difference vector between embeddings.

$$emb(\text{"}King\text{"}) + (emb(\text{"}Woman\text{"}) - emb(\text{"}Man\text{"})) = emb(\text{"?"})$$

By calculating the distance vector between the embeddings of the prompt analogy (in this case, "Man" and "Woman"), and then adding that to the incomplete analogy (here, "King"), we can predict the analogy output should be "Queen".

A more dramatic example of such learning, GPT-like models demonstrated an ability to answer many questions and complete sentences from just pretraining on predicting the next token tasks. **Complete the sentence:** Seed GPT with some text and see what it says next

Example:
Input : "Bob went to the zoo when he was frightened upon seeing a ferocious ....."
GPT continues: "...tiger. This tiger was huge. . . ."

People observed even more surprising behavior in that GPT could be used to answer questions. For example, when seeded with a prompt that provides analogies of states and their capitals, GPT was typically able to produce the capital for any state when asked to continue:

"Let's think of capitals of states. For example, the capital of California is Sacramento; the capital of Massachusetts is Boston; the capital of <QUERY STATE> is _____"

Here <QUERY STATE> could be any state and GPT would answer with the capital of the state correctly. This led to the area of **prompt engineering** - figuring out prompts that make the models do what we want. It turns out that we can get a language model to perform new tasks simply by exploring the generation capability of the model and picking appropriate texts, without any further training. We can simply treat the generative language model as a black box and encode our tasks as a call to that black box. There are no gradients and weights updating in this process. This is often called Zero-Shot or Few-Shot learning.

References

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

[2] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. Improving language understanding by generative pre-training. 2018.