EECS 182 Deep Neural Networks
Spring 2023 Anant Sahai

Review: Fine-tuning

1. Finetuning

(a) If you pretrain using a masked autoencoder, when you finetune the autoencoder encoder for downstream tasks, **do you still mask the inputs?**

Solution: Typically you don't do this, though it's possible as a form of data augmentation in the downstream task.

(b) Let's say you want to train an LSTM encoder/decoder model on translating from English to Spanish using a paired English/Spanish training set. You also have a much larger corpus of unpaired English sentences. **Describe one way to pretrain the LSTM encoder using the unpaired data.**

Solution: One option is to train an LSTM encoder/decoder model which acts as an autoencoder. An English sentence is fed into the encoder (possibly with noise or masking), and the decoder resconstructs the original sentence. Hopefully this will train the encoder to learn good representations of English sentences, so this encoder can be finetuned on the paired data.

(c) For each of the following finetuning problems, **describe whether you should prefer to use (a) feature extraction (also called linear probing), (b) full finetuning, (c) hard prompting, or (d) soft prompting.**

(i) You are using a 175B parameter language model for a question-answering task. You have a dataset of 100k examples.

Solution: Soft prompting is preferred here, since when you have lots of data points (like here), soft prompting outperforms hard prompting.

(ii) You are using a 90B parameter language model for a spam classification task. You have a task description but no training data.

Solution: If you have no training data, hard-promting is the only valid option.

(iii) You are using a 1B parameter conv net pretrained on ImageNet for wildlife classification task. You have 100 training examples.

Solution: Feature extraction is probably the best. We can't (yet) use prompting with conv nets. With so few training examples, full-finetuning isn't a great option either since finetuning may distort the pretrained features and result in overfitting.

(iv) You are using a 1B parameter vision transformer pretrained on ImageNet for an X-Ray fracture localization task. You have 100M training examples.

Solution: You should use full finetuning. Since the downstream task is fairly different from the pretraining task, and the dataset is pretty large, full finetuning is likely to result in better performance.

2. Prompting

(a) Typically, when you create a soft prompt for use with a GPT model, you prepend the prompt to the left of the input. **Could you also get good performance by appending it to the right**?

Solution: This could still work. Prompts are often appended at the beginning so that the input tokens can attend to the prompt, but if the prompt is on the right, the sequence items corresponding to the prompt can still attend to the other input tokens.

(b) Let's say you would like to use hard prompting with a large GPT model. You have a dataset of 10 thousand training examples for your downstream task. Would it be a good idea to include all of these examples (except for a held-out validation set) in your prompt?

Solution: Probably not, for a few reasons:

- The dataset is likely longer than the sequences the transformer was trained on. If you are using learned positional encodings, you cannot run the model on longer sequences than were seen before. If you use fixed (e.g. sinusoidal) positional encodings, you can use longer sequences, but performance might degrade if the sequence length is very out of distribution from what is seen before.
- Past some point, performance is unlikely to keep getting better as you add more examples.
- The longer your prompt is, the slower and more computationally expensive it is to run the model. You may also eventually run into memory errors.

3. Finetuning Pretrained NLP Models (From Discussion 11)

In this problem, we will compare finetuning strategies for three popular architectures for NLP.

(a) **BERT** - encoder-only model

(b) **T5** - encoder-decoder model

(c) **GPT** - decoder-only model

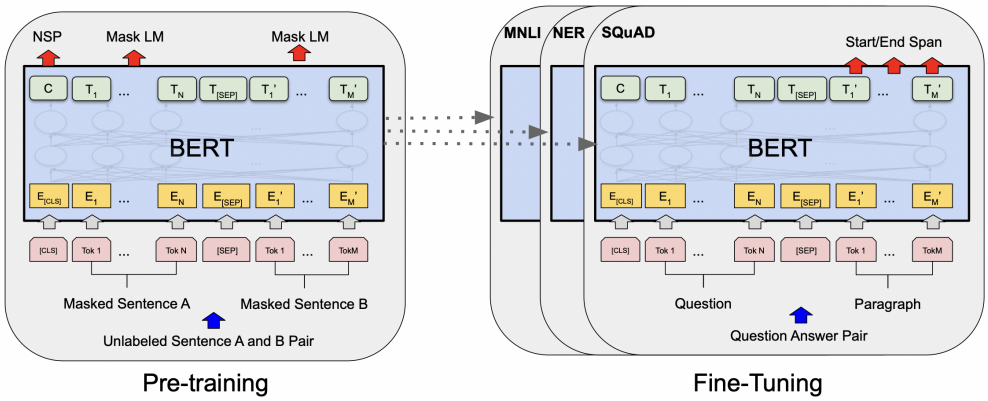

Figure 1: Overall pre-training and fine-tuning procedures for BERT.

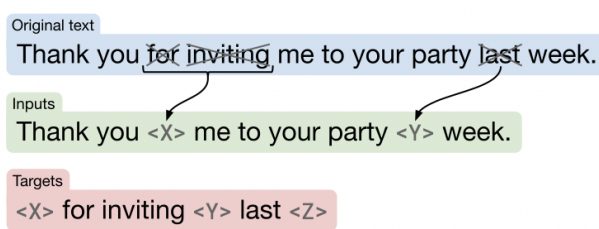

Figure 2: T5 Training procedure

(a) For each of the three models, **state the objective used for pretraining.**

Solution: BERT uses two training objectives: a masked token prediction objective (i.e. a cross-entropy loss for predicting input tokens which were masked out), and a next sentence prediction objective which predicts whether the two sentences passed in are sequential or not. This is shown in Figure 1 (Followup works such as RoBERTa only use the masked prediction objective.)

T5 uses a slightly different masking objective. Spans of multiple words are masked out from the encoder. The decoder must predict the tokens for each span, as shown in Figure 2.

GPT is trained using using a next-token classification loss.

(b) Consider the MNLI (Multi-Genre Natural Language Inference Corpus) task. It provides a passage and a hypothesis, and you must state whether the hypothesis is an entailment, contradition, or neutral.

> **EXAMPLE:**
> **Passage**: At the other end of Pennsylvania Avenue, people began to line up for a White House tour.
> **Hypothesis**: People formed a line at the end of Pennsylvania Avenue.
> **Classification**: entailment

(i) **With each of the 3 models, state whether it is possible to use the model for this task with no finetuning or additional parameters.** If so, state how.

Solution: With GPT and T5, you could accomplish this by **few-shot prompting**. Input a set of examples (passage, hypothesis, and answer), followed by the passage and hypothesis for the current question. The decoder would then predict the answer.

It is technically possible to use BERT for this task by inputting a prompt, masking a token for the answer, and predicting it, but BERT is not good at learning through prompting.[1].

(ii) With each of the 3 models, **state how you would use the model for this task if you were able to add additional parameters and/or finetune existing parameters.**

Solution: With GPT and T5, you could input the passage and hypothesis, then finetune the model to predict the answer as the next token. Note that with large models it is often common to only finetune a subset of the model parameters.[2]

With BERT, you input a sequence to the model: [<CLS>, passage, <SEP>, hypothesis]. Remove the output layer of the network and replace it with a classification head on the CLS output token. (Some approaches also make use of the other tokens, such as by taking a mean across the other output tokens and concatenating them with the CLS token.) You can either finetune all parameters or only the parameters of the classification head.

(Students may suggest other answers which are valid too - e.g. you could extract features from GPT by concatenating token features from the last couple of layers and training a classification head on top of them, but these may not perform as well).

(c) **Compare and contrast the ways we use pretrained representations in BERT to the way we use pretrained autoencoder representations.**

Solution: When you use a single layer of BERT as your representation, the bottom of the network serves the role of the AE encoder (the piece you use to encode inputs for the downstream task), and the final layer(s) serve the role of the AE decoder (the piece which is used in training then thrown away). However, NLP applications sometimes use features from multiple layers in the model, whereas AEs typically use a single layer representation.

[1] For new work modifying BERT to benefit from prompting, see https://arxiv.org/pdf/2201.04337.pdf

[2] For a comparison between of several parameter-efficient finetuning methods, check out the experiments in this paper: https://arxiv.org/pdf/2205.05638.pdf

