**Prompting**

Training large pretrained language models is very time-consuming and compute-intensive. As they continue to grow in size, there is increasing interest in more efficient training methods such as *prompting*. Prompting primes a frozen pretrained model for a specific downstream task by including a text prompt that describes the task or even demonstrates an example of the task. With prompting, you can avoid fully training a separate model for each downstream task, and use the same frozen pretrained model instead. This is a lot easier because you can use the same model for several different tasks, and it is significantly more efficient to train and store a smaller set of prompt parameters than to train all the model’s parameters.

There are two categories of prompting methods:

* hard prompts are manually handcrafted text prompts with discrete input tokens; the downside is that it requires a lot of effort to create a good prompt
* soft prompts are learnable tensors concatenated with the input embeddings that can be optimized to a dataset; the downside is that they aren’t human readable because you aren’t matching these “virtual tokens” to the embeddings of a real word

This conceptual guide provides a brief overview of the soft prompt methods included in 🤗 PEFT: prompt tuning, prefix tuning, and P-tuning.

**Prompt tuning**

Only train and store a significantly smaller set of task-specific prompt parameters [(image source)](https://arxiv.org/abs/2104.08691).

Prompt tuning was developed for text classification tasks on T5 models, and all downstream tasks are cast as a text generation task. For example, sequence classification usually assigns a single class label to a sequence of text. By casting it as a text generation task, the tokens that make up the class label are *generated*. Prompts are added to the input as a series of tokens. Typically, the model parameters are fixed which means the prompt tokens are also fixed by the model parameters.

The key idea behind prompt tuning is that prompt tokens have their own parameters that are updated independently. This means you can keep the pretrained model’s parameters frozen, and only update the gradients of the prompt token embeddings. The results are comparable to the traditional method of training the entire model, and prompt tuning performance scales as model size increases.

Take a look at [Prompt tuning for causal language modeling](https://huggingface.co/docs/peft/task_guides/clm-prompt-tuning) for a step-by-step guide on how to train a model with prompt tuning.

**Prefix tuning**

Optimize the prefix parameters for each task [(image source)](https://arxiv.org/abs/2101.00190).

Prefix tuning was designed for natural language generation (NLG) tasks on GPT models. It is very similar to prompt tuning; prefix tuning also prepends a sequence of task-specific vectors to the input that can be trained and updated while keeping the rest of the pretrained model’s parameters frozen.

The main difference is that the prefix parameters are inserted in **all** of the model layers, whereas prompt tuning only adds the prompt parameters to the model input embeddings. The prefix parameters are also optimized by a separate feed-forward network (FFN) instead of training directly on the soft prompts because it causes instability and hurts performance. The FFN is discarded after updating the soft prompts.

As a result, the authors found that prefix tuning demonstrates comparable performance to fully finetuning a model, despite having 1000x fewer parameters, and it performs even better in low-data settings.

Take a look at [Prefix tuning for conditional generation](https://huggingface.co/docs/peft/task_guides/seq2seq-prefix-tuning) for a step-by-step guide on how to train a model with prefix tuning.

**P-tuning**

Prompt tokens can be inserted anywhere in the input sequence, and they are optimized by a prompt encoder [(image source)](https://arxiv.org/abs/2103.10385).

P-tuning is designed for natural language understanding (NLU) tasks and all language models. It is another variation of a soft prompt method; P-tuning also adds a trainable embedding tensor that can be optimized to find better prompts, and it uses a prompt encoder (a bidirectional long-short term memory network or LSTM) to optimize the prompt parameters. Unlike prefix tuning though:

* the prompt tokens can be inserted anywhere in the input sequence, and it isn’t restricted to only the beginning
* the prompt tokens are only added to the input instead of adding them to every layer of the model
* introducing *anchor* tokens can improve performance because they indicate characteristics of a component in the input sequence

The results suggest that P-tuning is more efficient than manually crafting prompts, and it enables GPT-like models to compete with BERT-like models on NLU tasks.

Take a look at [P-tuning for sequence classification](https://huggingface.co/docs/peft/task_guides/ptuning-seq-classification) for a step-by-step guide on how to train a model with P-tuning.