CS 282 Deep Neural Networks **Fall 2022**

Lecture 23: Nov 10, 2022

Lecturer: Anant Sahai *Scribe: Jing Xu, Kiran Ganeshan*

23.1 More Details on Prompt Tuning

Recall the prompt tuning setup, which is shown in Figure 1.

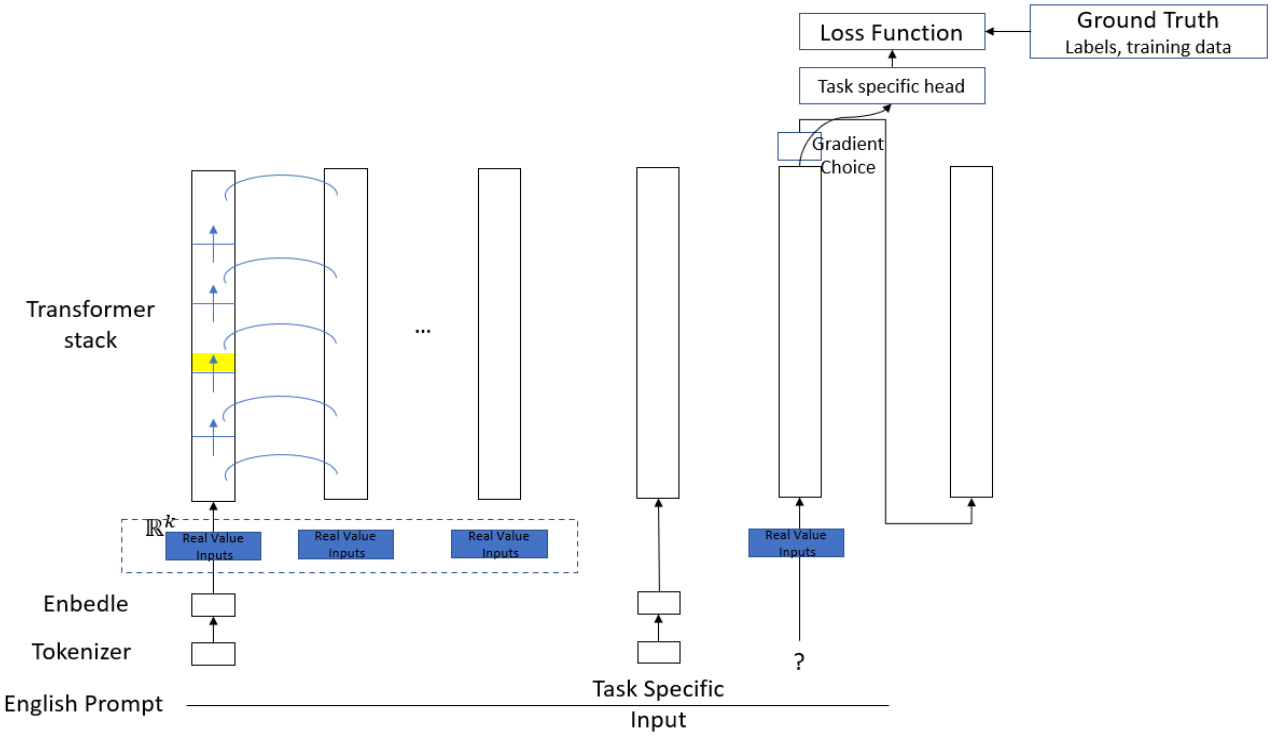

Figure 23.1: Framework of Prompt Tuning

For example, consider the prompt "What is the capital of California". We might consider this prompt to be part of a task that involves recalling capitals. However, the only token which specifies which specific subtask we want to solve is the "California" token. We refer to this as the Task Specific Input token in Figure 1. The remainder of the prompt could be changed without changing the semantics of the question.

With this in mind, note that the outputs of the Transformer depend only on the real vector-valued inputs that are produced by the embedder. This raises the following question: why do these real vector-valued inputs need to correspond to English sentences? (Why would it be that the output of the embedder is the

best choice?) While there are a finite number of choices within our embedding table, there are infinitely many choices for our inputs, so we can utilize gradient descent to find the best one. Fine tuning works with both small and large models, while prompt tuning seems to only work super well with larger models. The only way information is propagated to the next input is through the population of the attention tables. Why do I need consistency across layers? This is the inspiration for soft prompt in between layers. The supervision will be provided by the same loss function used in the feature-extraction view of task-specific finetuning. These are known as **soft prompts**. In Figure 1, there is a soft prompt in each transformer layer.

Note: this idea requires that we sample outputs in a differentiable way (with respect to scores) at the orange boxes to enable gradient flow to the prompt inputs. For example, argmax may not work.

Questions:

- How do we choose the length of our prompts?
  → Start with an English prompt that works OK, and stick with that length.
  → Search over prompt lengths for maximum accuracy, starting with the shorted prompt lengths. Which of these performs better is very sensitive to the underlying task.

Advantages:
1) Improves performance on underlying task
2) Maintains scalability by keeping the model weights the same across tasks
3) Allows us to use large training datasets
4) Lets us leverage ensembles, etc... (Standard Gradient Descent Approach)
5) Avoids the large performance differences between semantically similar prompts suffered by manual prompt engineering
6) Lets us use feature extraction as well, unlike manual prompt engineering

Disadvantages:
In order to reach full fine tuning level performance, we normally must use very large models. Example: If the transformer is a very large model (10billion+ params), then prompt will approach performance of fine tuning. If the transformer has 100 million params, for example, then prompt tuning will not get you there. This is not fully understood.

How can we improve this further?
A) Put soft prompts around the input, rather than having the input follow the soft prompt
B) Apply a learnable mapping to the input itself
A') Let the soft prompt depend on the input, rather than learning it independently
C) Add soft prompts to intermediate layers of the Transformer. Language models observe the initial condition and generate and continue the flow of the "differential equation". The prompt shapes the evolution of the system. We get a richer shape of how we set the initial conditions.

Questions:

- Would it help to add a trainable layer between the embedder and the soft prompts?
  → No, since this would just be another way to learn soft prompts. This time, soft prompts would be a linear transformation of embedder outputs, rather than parameter vectors. If we use a more complex model like an LSTM, this becomes identical to A'.
- If we add soft prompts to an intermediate layer in the Transformer at a token T, throwing away the output from previous layers at T, do we stop gradient flow to the soft prompts at the input?
  → No, because there is layer-wise self attention. Specifically, because the outputs of the first layer at later tokens depends on the keys and values of the first layer at T, there is "sideways" information flow along the sequence dimension at each layer, so the input soft prompt still receives gradient flow. The advantage of using soft prompts in intermediate layers is that performance benefits extend to much smaller models.
- What is the relationship between approach C and skip connections?
  → This approach is uniquely opposite to skip connections in that rather than encouraging gradient flow by adding inputs, we completely overwrite the intermediate values with a parameter vector, stopping gradient flow.

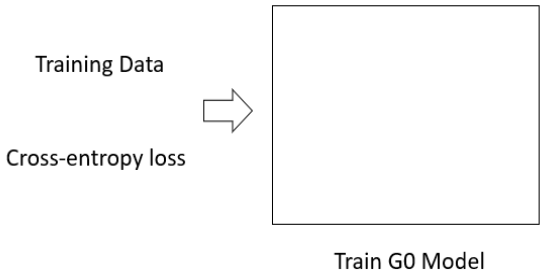

Figure 23.2: G0 Model

Aside

- Knowledge Distillation: use what's learned before as a source of labels
  → Take my training data and cross-entropy loss and train a model called generation zero model shown in Figure 2.

  → Consider Figure 3. Which is more true? The data points or the line? On one hand, the data points come from the actual world like experiments and have a truth value in that. However, the points are all noisy, and what you care about is the underlying pattern, and if you have successfully captured that, then the line is more true. The points are merely in the world and have all the hindrances of the real world. But, what you are truly looking for is the platonic reality of the model of the real world and the line is closer to that than the points. You ask the question, "maybe the outputs of my model are more true?". A good model is smoothing noise and other artifacts out of your data. Classification data is always corrupted. You know a cat is more dog than it is ice cream. But the labels did not reflect that at all. The labels only assign one class to each image. The scores of a classification model will presumably tell you that it is mostly a cat, but if I had to pick another class a good second choice would be a dog. What you learned from the G0 model is richer that your original training data.

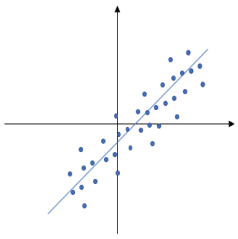

Figure 23.3: Which is more true?

→ Therefore, we can update the model with knowledge distillation loss like Figure 4.

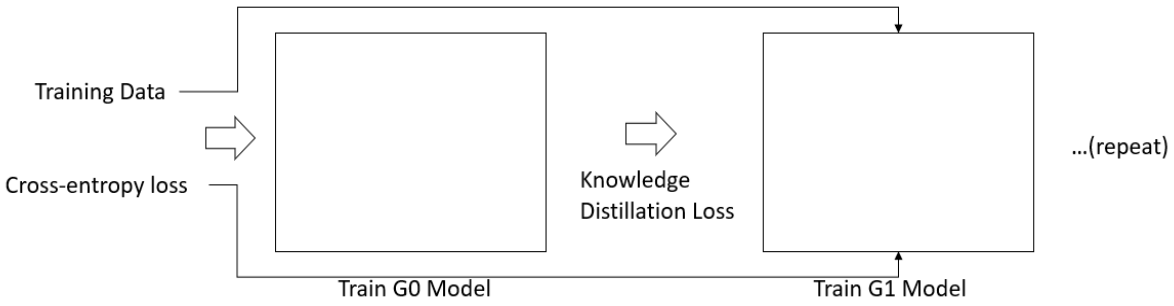

Figure 23.4: Iterations based on Knowledge Distillation

23.2 Meta-Learning

23.2.1 Multi-task Learning

Suppose we have a whole family of tasks, each with its own training data, and we want to find a system that can quickly/reliably learn new tasks. The new task is unseen.

Approaches to learn the new task:
1) Follow "feature extract" paradigm
2) Follow "fine tuning" paradigm

More details for "fine tuning" paradigm, what do we do on a new task?
a) Initialize our network with some parameters.
b) Do K-steps of SGD using our training data.
c) Evaluate on our hold-out set.
3) Follow nearest-neighbour paradigm
...

Consider the "fine tuning" paradigm, where we use MAML(Model-Agnostic Meta-Learning).

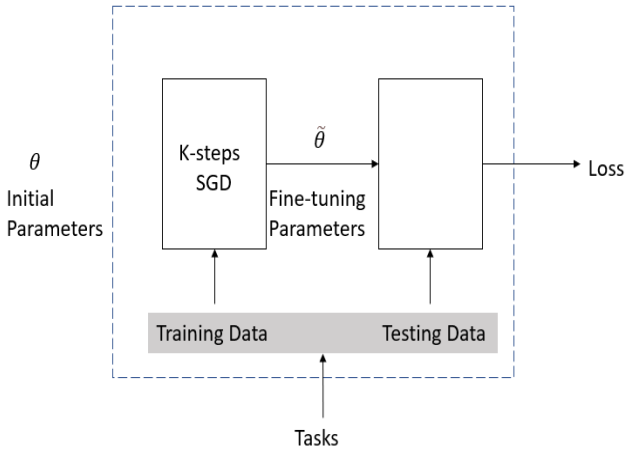

Figure 23.5: Framework of MAML for Learning the System

Let's just do SGD on the above problem in "fine tuning" paradigm shown in Figure 5. We split the tasks into training data and testing data. For this framework, the input is the initial parameters and the output is the loss. Therefore, we can do SGD for the system.

Questions:

- If we approach this according to Figure 5 with large K, could this lead to exploding or vanishing gradients?
  → Yes, it is a real concern. Therefore, we should keep K reasonable. We should sample each task many times and take different subsets of training data, since we can only afford to do so many steps of SGD.
- What is $\tilde{\theta}$?
  → These are the parameters fine tuned via SGD, which we can use on on testing data.

K-steps SGD itself in Equation 1 is differentiable, and we can view the equation like an RNN.

$$\theta_l[i] = \theta_{l-1}[i] + \eta * (-Gradient) \tag{23.1}$$

Questions:

- Comments
  → The goal of this algorithm is to find meta-parameter values such that optimal parameter values for each task are within 1 gradient step from the meta-parameter values. It seems unreasonable that we could find such meta-parameter values, as individual tasks likely have optimal parameter values that are far away from each other in parameter space. However, if we were to try taking multiple gradient steps and differentiating through them, this would create a more complex version of the algorithm that takes 1 gradient step. Therefore, we try taking 1 task-specific gradient step before exploring fancier algorithms. More generally, to create fancier algorithms, we should try what basic things first and get insights.
- What is the relationship between this approach and the second derivative, since second derivatives don't work on ReLU nonlinearities?
  → Yes, it looks like we do second derivative here. It turns out, however, since we use an approximation for the Hessian that only relies on taking multiple first derivatives, we never need to consider the second derivative of the ReLU function.