HW9.

Mengying Lin SID: 3038737132.

1. Meta-learning for Learning 1D functions

Algorithm 1 Model-Agnostic Meta-Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to K examples
6: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
7: **end for**
8: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
9: **end while**

(a) In the original MAML algorithm, the inner loop performs gradient descent to optimize loss with respect to a task distribution. However, here we're going to use the closed form min-norm solution for regression instead of gradient descent.

Let's recall the closed form solution to the min-norm problem. Write the solution to

$$\underset{\boldsymbol{\beta}}{\operatorname{argmin}} \|\boldsymbol{\beta}\| \text{ , such that } \mathbf{y} = A\boldsymbol{\beta}$$

in terms of A and $\mathbf{y}$.

$\beta = A^T(AA^T)^{-1}y.$

in terms of A and $\mathbf{y}$.

(b) For simplicity, suppose that we have exactly one training point (x, y), and one true feature $\phi_t^u(x) = \phi_1^u(x)$. We have a second (alias) feature that is identical to the first true feature, $\phi_a^u(x) = \phi_2^u(x) = \phi_1^u(x)$. This is a caricature of what always happens when we have fewer training points than model parameters.

The function we wish to learn is $y = \phi_t^u(x)$. We learn coefficients $\hat{\boldsymbol{\beta}}$ using the training data. Note, both the coefficients and the feature weights are 2-d vectors.

Show that in this case, the solution to the min-norm problem 1 is $\hat{\boldsymbol{\beta}} = \frac{1}{c_0^2 + c_1^2} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix}$

let $A = \begin{pmatrix} c_0 \phi_a^u(x) \\ c_1 \phi_a^u(x) \end{pmatrix}^T$

$$\beta_{min} = A^T(AA^T)^{-1}y$$

$$= \begin{pmatrix} c_0\phi_a^u(x) \\ c_1\phi_a^u(x) \end{pmatrix} \left(c_0^2\phi_a^{u2}(x) + c_1^2\phi_a^{u2}(x)\right)^{-1} \begin{pmatrix} \phi_a^u(x) \\ \phi_a^u(x) \end{pmatrix}$$

$$= \frac{1}{c_0^2 + c_1^2} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix}$$

(c) Assume for simplicity that we have access to infinite data from the test distribution for the purpose of updating the feature weights $\mathbf{c}$. **Calculate the gradient of the expected test error with respect to the feature weights c_0 and c_1, respectively:**

$$\frac{\mathrm{d}}{\mathrm{d}\mathbf{c}} \left(\mathbb{E}_{x_{test}, y_{test}} \left[\frac{1}{2} \left\| y - \hat{\beta}_0 c_0 \phi_t^u(x) - \hat{\beta}_1 c_1 \phi_a^u(x) \right\|_2^2 \right] \right).$$

Use the values for β from the previous part. *(Hint: the features $\phi_i^u(x)$ are orthonormal under the test distribution. They are not identical here.)*

$$E_{x_{test}, y_{test}} \left[\frac{1}{2} \| y - \hat{\beta}_0 c_0 \phi_t^u(x) - \hat{\beta}_1 c_1 \phi_a^u(x) \|_2^2 \right]$$

$$= \frac{1}{2} E_{x_{test}, y_{test}} \left[\| \phi_t^u(x) - \frac{c_0^2}{c_0^2 + c_1^2} \phi_t^u(x) \|_2^2 + \| \frac{c_1^2}{c_0^2 + c_1^2} \phi_a^u(x) \|_2^2 \right]$$

$$= \frac{1}{2} \left[\left(1 - \frac{c_0^2}{c_0^2 + c_1^2}\right)^2 + \left(\frac{c_1^2}{c_0^2 + c_1^2}\right)^2 \right]$$

$$= \frac{c_1^4}{(c_0^2 + c_1^2)^2}$$

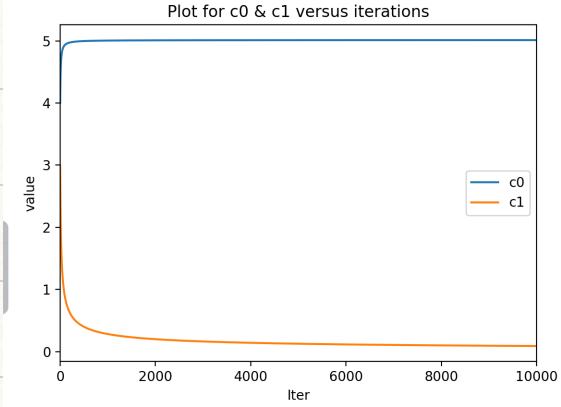
$$\frac{dE}{dc_0} = -\frac{4c_1^4 c_0}{(c_0^2 + c_1^2)^3}$$

$$\frac{dE}{dc_1} = \frac{4c_1^3(c_0^2 + c_1^2) - 4c_1^5}{(c_0^2 + c_1^2)^3}$$

$$= \frac{4c_1^3 c_0^2}{(c_0^2 + c_1^2)^3}.$$

(d) **Generate a plot** showing that, for some initialization $\mathbf{c}^{(0)}$, as the number of iterations $i \to \infty$ the weights empirically converge to $c_0 = \|\mathbf{c}^{(0)}\|$, $c_1 = 0$ using gradient descent with a sufficiently small step size. Include the initialization and its norm and the final weights. What will β go to?

Run the code in the Jupyter Notebook and then answer these questions:



Init: $c_0 = 4$, $c_1 = 3$. $\|c^{(0)}\| = 5$.

Final: $c_0 = 5.0126$ $c_1 = 0.0889$

(e) (In MAML for regression using closed-form solutions) Considering the plot of regression test loss versus `n_train_post`, **how does the performance of the meta-learned feature weights compare to the case where all feature weights are set to 1?** Additionally, **how does their performance compare to the oracle**, which performs regression using only the features present in the data? Can you **explain the reason for the downward spike observed at `n_train_post = 32`**?

The former performs way more better.

Oracle performs better than the two cuz it has the knowledge of the true features.

when n_train_post = 32 there is a spike because the inner tasks are trained with a batchsize of 32. It means in meta-training, the inner update loop will only use a subset of the available training examples

(f) By examining the changes in feature weights over time during meta-learning, **can you justify the observed improvement in performance?** Specifically, can you **explain why certain feature weights are driven towards zero**?

At first all of the weights are initialized to 0. Afterwards most of them are driven to 0 while the others are gradually driven to large values.

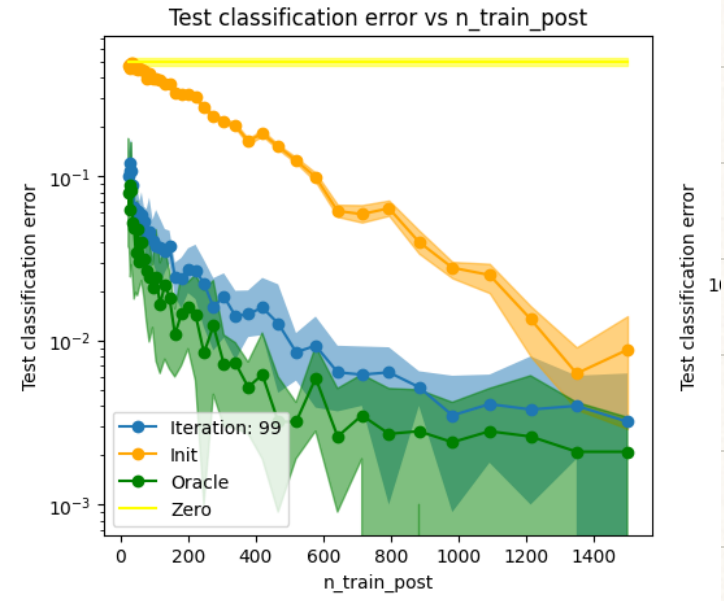
For those driven to 0, it could be because they are less significant

(g) (In MAML for regression using gradient descent) **With `num_gd_steps = 5`, does meta-learning contribute to improved performance during test time? Furthermore, if we change `num_gd_steps` to 1, does meta-learning continue to function effectively?**

Yes. No, its performance degraded because it couldn't be finely adapted to the data within just 1 step.

(h) (In MAML for classification) Based on the plot of classification error versus `n_train_post`, **how does the performance of the meta-learned feature weights compare to the case where all feature weights are 1? How does the performance of the meta-learned feature weights compare to the oracle** (which performs logistic regression using only the features present in the data)?

Iteration: 100



The same answer in the regression part.

(i) By observing the evolution of the feature weights over time as we perform meta-learning, **can you justify the improvement in performance? Specifically, can you explain why some feature weights are being driven towards zero?**

The same answer in the regression part.

2. Vision Transformer

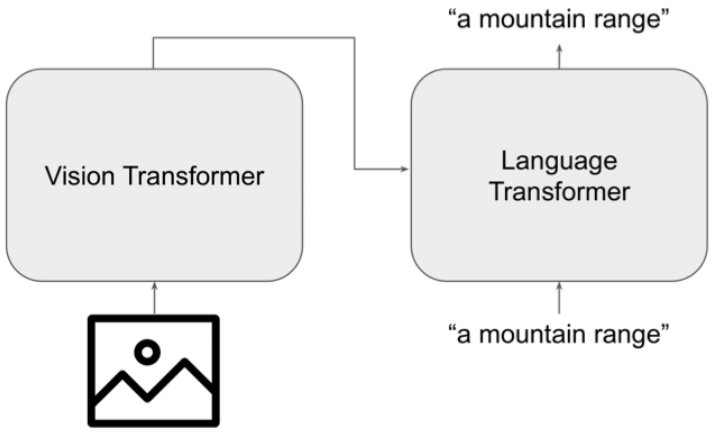


Figure 3: Image captioning model

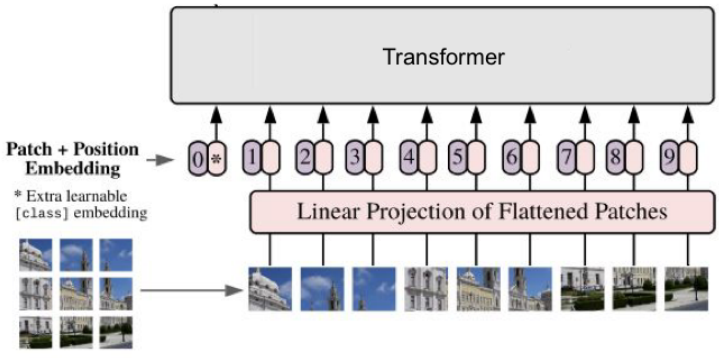


Figure 4: Vision Transformer

(a) For each transformer, state whether it is more appropriate to use a transformer encoder (a transformer with no masking except to handle padding) or decoder (a transformer with autoregressive self-attention masking) and why.

Vision transformer?

- ○ Encoder-style transformer
- ○ Decoder-style transformer

Reason: It ~~feeds in already encoded vectors.~~ Produce an image representation

Language transformer?

- ○ Encoder-style transformer
- ○ Decoder-style transformer

Reason: Need to encode the texts into vectors.

↳ Need to generate tokens autoregressively.

(b) A standard language transformer for captioning problems alternates between layers with cross-attention between visual and language features and layers with self-attention among language features. Let's say we modify the language transformer to have a single layer which performs both attention operations at once. The grid below shows the attention mask for this operation. (For now, assume the vision transformer only outputs 3 image tokens called <ENC1>, <ENC2>, and <ENC3>. <SOS> is the start token, and <PAD> is a padding token.)

(i) One axis on this grid represents sequence embeddings used to make the queries, and the other axis represents sequence embeddings used to make the keys. **Which is which?**

- ◯ Each column creates a query, each row creates a key and a value
- ◯ Each column creates a key and a value, each row creates a query
- ◯ Each column creates a query and a value, each row creates a key
- ◯ Each column creates a key, each row creates a query and a value

(ii) **Mark X in some of the blank cells in the grid to illustrate the attention masks.** (A X marked cell is masked out, a blank cell is not.)

	<SOS>	a	mountain	range	<PAD>
<SOS>		X	X	X	X
a	X		X	X	X
mountain	X	X		X	X
range	X	X	X		X
<PAD>	X	X	X	X	X
<ENC1>					X
<ENC2>					X
<ENC3>					X

The obj of a decoder-side transformer is next token prediction.

(c) In discussion, we showed that the runtime complexity of vision transformer attention is $O(D(H^4/P^4))$, where H is the image height and width, P is the patch size, and D is the feature dimension of the queries, keys, and values. Some recent papers have reduced the complexity of vision transformer attention by segmenting an image into windows, as shown in Figure 5.

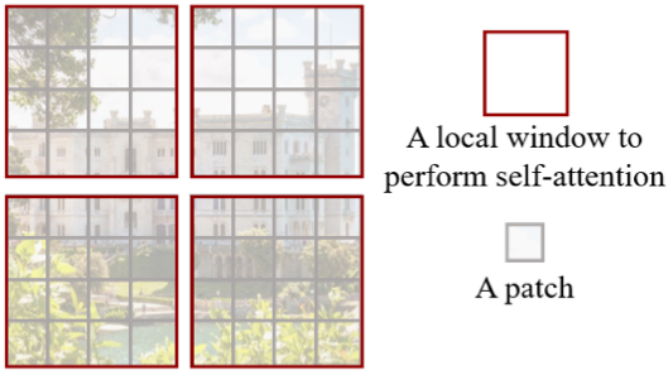


Figure 5: Vision transformer attention with windows

Patches only attend to other patches within the same window. **What is the Big-O runtime complexity of the attention operation after this modification?** Assume each window consists of K by K patches.

There are K^2 patches in a window.

$(H/kp)^2$ windows in total.

Each vec is size D

~~Computing attention between each pair of patches: $O(p^2)$~~

The time complexity is $O(k^4 \cdot H^2/k^2p^2 \cdot p^2) = O(H^2k^2D/p^2)$.

Another: $(H/p)^2$ items in total. each will attend to k^2 other patches. Vector is of size D. $\Rightarrow O(\frac{H^2}{p^2} \cdot k^2 D)$.

3. Pretraining and Finetuning

When we use a pretrained model without fine-tuning, we typically just train a new task-specific head. With standard fine-tuning, we also allow the model weights to be adapted.

However, it has recently been found that we can selectively fine-tune a subset of layers to get better performance especially under certain kinds of distribution shifts on the inputs. Suppose that we have a ResNet-26 model pretrained with CIFAR-10. Our target task is CIFAR-10-C, which adds pixel-level corruptions (like adding noise, different kinds of blurring, pixelation, changing brightness and contrast, etc) to CIFAR-10. If we could only afford to fine-tune one layer, **which layer (i.e. 1,2,3,4,5) in Figure 6 should we choose to finetune to get the best performance on CIFAR-10-C? Give brief intuition as to why.**

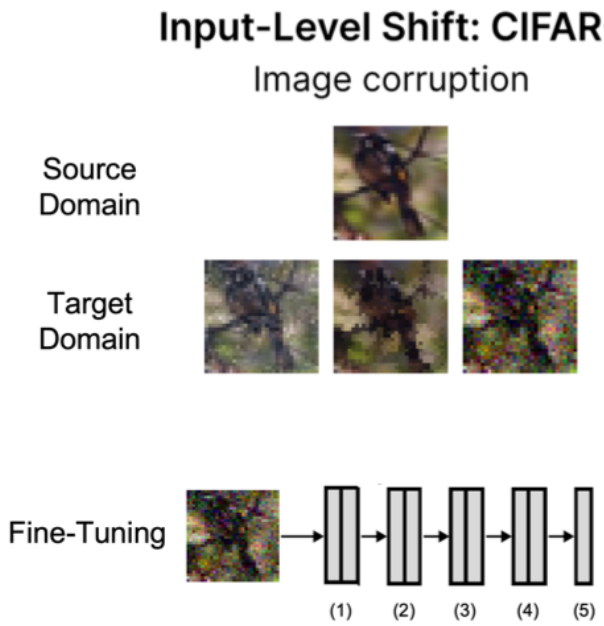


Figure 6: Fine-tuning the model pretrained with CIFAR-10 on CIFAR-10-C dataset

The ~~5th~~ 1st layer, because it is considered that the lower layers usually extract common features while the deep layers will be responsible for extracting more complex and high-level features which are more targeted at task. CIFAR-10-C is low level feature shifts.

4. Prompting Language Models

(a) **Exploring Pretrained LMs**

Play around with the web interface at https://dashboard.cohere.ai/playground/generate. This plaground provides you an interface to interact with a large language model from Cohere and tweak various parameters. You will need to sign up for a free account.

Once you're logged in, you can choose a model in the parameters pane on the right. "command-xlarge-nightly" is a generative model that responds well with instruction-like prompts. "xlarge" and "medium" are generative models focusing on sentence completion. Spend a while exploring prompting these models for different tasks. Here are some suggestions:

- Look through the 'Examples ...' button at the top of the page for example prompts.
- Ask the model to answer factual questions.
- Prompt the model to generate a list of 100 numbers sampled uniformly between 0 and 9. Are the numbers actually randomly distributed?
- Insert a poorly written sentence, and have the model correct the errors.
- Have the model brainstorm creative ideas (names for a storybook character, recipes, solutions to solve a problem, etc.)
- Chat with the model like a chatbot.

Answer the questions below:

i. **Describe one new thing you learned by playing with these models.**

ii. **How does the temperature parameter affect the outputs?** Justify your answer with a few examples.

iii. **Describe a task where the larger models (e.g., "xlarge" or "command-xlarge-nightly") significantly outperform the smaller ones (e.g., "medium").** Paste in examples from the biggest and smallest model to show this.

iv. **Describe a task where even the largest model performs badly**. Paste in an example to show this.

v. **Describe a task where the model's outputs improve significantly with few-shot prompting compared to zero-shot prompting.**

i. Not every GPT can be called chatGPT !!.

ii. The higher the temperature, the more random the result.

e.g. when input "How to fit in a country with a foreign language."

when temperature set to 0.1, the output is:

How to fit in a new environment with a foreign language

How to fit in a new environment with a foreign language

How to fit in a new environment with a foreign language

when temperature set to 0.4:

How to learn a new language
How to learn a new language in a short time
How to learn a new language in a short time
How to learn a new language in a short time

When temperature set to 1.2:

This week's question was sent in by a member of the Art of Likability community.
Question: How to fit in a new environment with a foreign language
This question came in from Michael, a young guy who is originally from New York City.
In college, he studied in Germany for a semester.
He made some friends, had a great time, and fell in love with a local German girl.
After graduation, he decided to stay in Germany and get a job.
But now he's realizing how much of a struggle it's going to be to make new friends.
Here's what Michael wrote to me:
...

iii. e.g. Prove 0=1.

Command - medium - nightly:

To prove 1=0, you need to find a counterexample. A counterexample is a number that disproves a statement. In this case, you are trying to prove that 1=0, so you need to find a number that is equal to 1 but does not equal 0. One such number is 5.

If you input 5 into the equation 1=0, you get the equation 5=0, which is false. This counterexample disproves the statement 1=0.

Command - xlarge - nightly:

No, it is not possible to prove 1 = 0, because 1 is a positive number and 0 is a negative number, and they are not equal.

iv) Q: Introduce the basic idea of your training process:
A: This is an example description. To edit, click on the pencil icon.
This is an example of a description. To edit, click on the pencil icon.

(v.) when you want your answer to have a specific format. it's likely the case

(b) **Using LMs for classification**

Run `lm_prompting.ipynb`, then answer the following questions. If you did not do part (a), you will still need to get a Cohere account to complete this part.

i. Analyze the command-xlarge-nightly model's failures. **What kinds of failures do you see with different prompting strategies?**

ii. **Does providing correct labels in few-shot prompting have a significant impact on accuracy?**

iii. **Observe the model's log probabilities. Does it seem more confident when it is correct than when it is incorrect?**

iv. **Why do you think the GPT2 model performed so much worse than the command-xlarge-nightly model on the question answering task?**

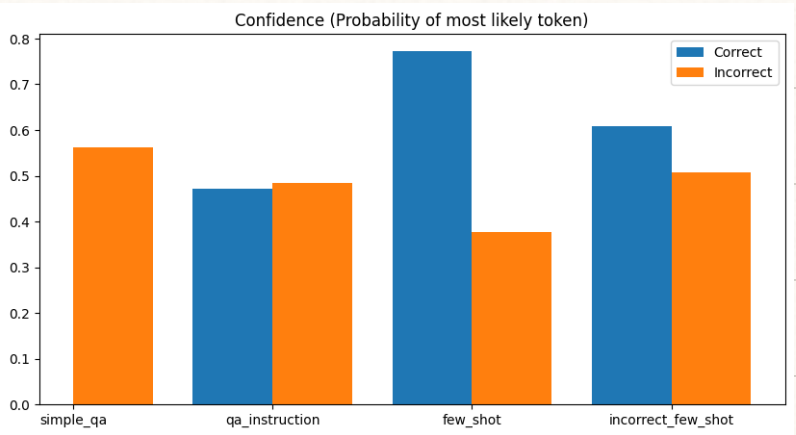
v. **How did soft prompting compare to hard prompting on the pluralize task?**

vi. You should see that when the model fails (especially early in training of a soft prompt or with a bad hard prompt) it often outputs common but uninformative tokens such as the, ", or \n. **Why does this occur?**

i. when strategies is simple/simple-qa/qa-instruction, the failures include invalid answers and WA.

ii. Yes.

iii.



when using few_shot/incorrect_few_shot, the model is more confident when it is correct. Otherwise it's the opposite.

iv. GPT2 is trained to generate texts that are related and coherent to its input. It's smaller and trained on less data.

v. Soft-embedding performs way more better than hard one in pluralizing a word because the former is more effective at capturing semantic relationships of words while the latter only considers word frequencies.

vi. When the model is uncertain about the

next token, it may choose to
output one that is frequently used in daily life.

5. Soft-Prompting Language Models

You are using a pretrained language model with prompting to answer math word problems. You are using chain-of-thought reasoning, a technique that induces the model to "show its work" before outputting a final answer.

Here is an example of how this works:

```
[prompt] Question: If you split a dozen apples evenly among yourself
and three friends, how many apples do you get? Answer: There are 12
apples, and the number of people is 3 + 1 = 4. Therefore, 12 / 4 = 3.
Final answer: 3\n
```

If we were doing hard prompting with a frozen language model, we would use a hand-designed [prompt] that is a set of tokens prepended to each question (for instance, the prompt might contain instructions for the task). At test time, you would pass the model the sequence and end after "Answer:" The language model will continue the sequence. You extract answers from the output sequence by parsing any tokens between the phrase "Final answer: " and the newline character "\n".

(a) Let's say you want to improve a frozen GPT model's performance on this task through soft prompting and training the soft prompt using a gradient-based method. This soft prompt consists of 5 vectors prepended to the sequence at the input — these bypass the standard layer of embedding tokens into vectors. (Note: we do not apply a soft prompt at other layers.) Imagine an input training sequence which looks like this:

```
["Tokens" 1-5: soft prompt] [Tokens 6-50: question]
[Tokens 51-70: chain of thought reasoning]
[Token 71: answer] [Token 72: newline]
[Tokens 73-100: padding].
```

want to let model better at reasoning & answer.

We compute the loss by passing this sequence through a transformer model and computing the cross-entropy loss on the output predictions. If we want to train the soft-prompt to output correct reasoning and produce the correct answer, **which output tokens will be used to compute the loss?** (Remember that the target sequence is shifted over by 1 compared to the input sequence. So, for example, the answer token is position 71 in the input and position 70 in the target).

Output tokens 1-5. ⇒ × No ground truth.

~~50~~ to 71.

(b) Continuing the setup above, **how many parameters are being trained in this model?** You may write this in terms of the max sequence length S, the token embedding dimension E, the vocab size V, the hidden state size H, the number of layers L, and the attention query/key feature dimension D.

Embedding layer: VE. (a) will be applied to 5 vectors that constitute the soft-prompt.

For decoder:

- Self attention: $3H^2$ (The matrices used for generating keys, values and queries are all $H \times H$).
- Cross attention: $3H^2$.

Everything else in the model will be

frozen

Final linear layer: HV

The overall parameters are $VE + 6H^2L + HV$.

(c) **Mark each of the following statements as True or False and give a brief explanation.**

(i) If you are using an autoregressive GPT model as described in part (a), it's possible to precompute the representations at each layer for the indices corresponding to prompt tokens (i.e. compute them once for use in all different training points within a batch).

(ii) If you compare the validation-set performance of the *best possible* K-token hard prompt to the *best possible* K-vector soft prompt, the soft-prompt performance will always be equal or better.

(iii) If you are not constrained by computational cost, then fully finetuning the language model is always guaranteed to be a better choice than soft prompt tuning.

(iv) If you use a dataset of samples from Task A to do prompt tuning to generate a soft prompt which is only prepended to inputs of Task A, then performance on some other Task B with its own soft prompt might decrease due to catastrophic forgetting.

i) ~~False~~ True. The auto-regressive process is strictly causal.

ii) ~~False~~ True. It depends on what sort of task we are performing.

iii) False. If we don't have sufficient data, fine-tune the whole network might lead to overfitting.

iv) ~~True~~ False.

(d) Suppose that you had a family of related tasks for which you want use a frozen GPT-style language model together with learned soft-prompting to give solutions for the task. Suppose that you have substantial training data for many examples of tasks from this family. **Describe how you would adapt a meta-learning approach like MAML for this situation?**

(HINT: This is a relatively open-ended question, but you need to think about what it is that you want to learn during meta-learning, how you will learn it, and how you will use what you have learned when faced with a previously unseen task from this family.)

- In each iteration, sample data pts from different tasks and then use the model to generate soft-prompting.
- Feed the embeddings to frozen pretrained GPT and get the task-specific loss.
- Do backprop and update the model.

6. TinyML - Quantization and Pruning.

(This question has been adapted with permission from MIT 6.S965 Fall 2022)

TinyML aims at addressing the need for efficient, low-latency, and localized machine learning solutions in the age of IoT and edge computing. It enables real-time decision-making and analytics on the device itself, ensuring faster response times, lower energy consumption, and improved data privacy.

To achieve these efficiency gains, techniques like quantization and pruning become critical. Quantization reduces the size of the model and the memory footprint by representing weights and activations with fewer bits, while pruning eliminates unimportant weights or neurons, further compressing the model.

(a) Please complete `pruning.ipynb`, then answer the following questions.

i. In part 1 the histogram of weights is plotted. **What are the common characteristics of the weight distribution in the different layers?**

ii. **How do these characteristics help pruning?**

iii. After viewing the sensitivity curves, please answer the following questions. **What's the relationship between pruning sparsity and model accuracy? (i.e., does accuracy increase or decrease when sparsity becomes higher?)**

iv. **Do all the layers have the same sensitivity?**

v. **Which layer is the most sensitive to the pruning sparsity?**

vi. (Optional) After completing part 7 in the notebook, please answer the following questions. **Explain why removing 30 percent of channels roughly leads to 50 percent computation reduction.**

vii. (Optional) **Explain why the latency reduction ratio is slightly smaller than computation reduction.**

viii. (Optional) **What are the advantages and disadvantages of fine-grained pruning and channel pruning? You can discuss from the perspective of compression ratio, accuracy, latency, hardware support (*i.e.*, requiring specialized hardware accelerator), etc.**

ix. (Optional) **If you want to make your model run faster on a smartphone, which pruning method will you use? Why?**

What are the common characteristics of the weight distribution in the different layers?

```
**Your Answer:** They all follow a guassian distribution.
```
Markdown

→ The distribution of weights is centered on 0, with tails dropping off easily.

```
### Question 1.2
💡
How do these characteristics help pruning?
```
Markdown

Question 4.1

What's the relationship between pruning sparsity and model accuracy? (*i.e.*, does accuracy increase or decrease when sparsity becomes higher?)

```
**Your Answer:** Generally, the acc drops when sparsity becomes higher.
```
Markdown

Question 4.2

Do all the layers have the same sensitivity?

```
**Your Answer:** No.For example, conv0 sees a significant acc drop when sparsity is 0.5, while conv7 does not
until sparsity increases to 0.8.
```
Markdown

Question 4.3

Which layer is the most sensitive to the pruning sparsity?

```
**Your Answer:** Conv0.
```
Markdown

method will you use? Why?

(b) Please complete `quantization.ipynb`, then answer the following questions.

i. After completing K-means Quantization, please answer the following questions. **If 4-bit k-means quantization is performed, how many unique colors will be rendered in the quantized tensor?**

ii. **If n-bit k-means quantization is performed, how many unique colors will be rendered in the quantized tensor?**

iii. After quantization aware training we see that even models that use 4 bit, or even 2 bit precision can still perform well. **Why do you think low precision quantization works at all?**

iv. (Optional) Please read through and complete up to question 4 in the notebook, then answer this question.

Recall that linear quantization can be represented as $r = S(q - Z)$. Linear quantization projects the floating point range $[fp_{min}, fp_{max}]$ to the quantized range $[quantized_{min}, quantized_{max}]$.

That is to say,

$$r_{\max} = S(q_{\max} - Z)$$
$$r_{\min} = S(q_{\min} - Z)$$

Substracting these two equations, we have,

$$S = r_{\max}/q_{\max}$$
$$S = (r_{\max} + r_{\min})/(q_{\max} + q_{\min})$$
$$S = (r_{\max} - r_{\min})/(q_{\max} - q_{\min})$$
$$S = r_{\max}/q_{\max} - r_{\min}/q_{\min}$$

4.2.1 Question 2.1

If 4-bit k-means quantization is performed, how many unique colors will be rendered in the quantized tensor?

Your Answer: $2^4 = 16$

4.2.2 Question 2.2

If n-bit k-means quantization is performed, how many unique colors will be rendered in the quantized tensor?

Your Answer: 2^n

4.4.2 Question 3.1

After quantization aware training we see that even models that use 4 bit, or even 2 bit precision can still perform well. Why do you think low precision quantization works at all?

Your Answer: Sometimes the ~~higher~~ lower bits are not that significant.

① Pruning as noise

② General structure is still maintained after quantization.