EECS 182 Deep Neural Networks
Fall 2022 Anant Sahai

Homework 9

This homework is due on Tue, Nov 22, 2022, at 10:59PM.

Deliverables: Please submit the code/notebooks/saved model as a zip file to the code gradescope assignment. Submit your written answers in the written gradescope assignment, and attach a pdf printout of the notebooks.

1. Masked Auto-Encoding

Here we will be implementing MAE in the MAE_student.ipynb notebook. Please upload a copy of the associated MAE notebook to colab where you will be able to run this on a GPU. Make sure to deactivate the notebook when you're not using to to prevent colab from locking you out of using a GPU due to runtime limits exceeded.

When you complete this notebook, it will generate a hw4_submission.zip file. Please upload this to the coding assignment, and upload a PDF of the notebook to the written assignment.

2. Meta-learning for Learning 1D functions

A common toy example with Neural Networks is to learn a 1D function. Suppose now that our task is not to learn not just one 1D function, but any of a class of 1D functions drawn from a *task distribution* $\mathcal{D}_T$.

In this problem we consider all signals of the form

$$y = \sum_{s \in \mathcal{S}} \alpha_s \phi_s^u(x)$$

The task distribution produces individual tasks which have true features with random coefficients in some *a priori* unknown set of indices $\mathcal{S}$. We do not yet know the contents of $\mathcal{S}$, but we can sample tasks from $\mathcal{D}_T$.

The important question is thus, **how do we use sampled tasks in training to improve our performance on an unseen task drawn from $\mathcal{D}_T$ at test time?**

One solution is to use our training tasks to learn a set of weights to apply to the features before performing regression through meta-learning. That is, we choose feature weights c_k to apply to the features $\phi_k^u(x)$ before learning coefficients $\hat{\beta}_k$ such that

$$\hat{y} = \sum_{k=0}^{d-1} \hat{\beta}_k c_k \phi_k^u(x).$$

These feature weights c_k are a toy model for the deep network that precedes the task-specific final layer in meta-learning.

We can then perform the min-norm optimization

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \, \|\boldsymbol{\beta}\|_2^2 \tag{1}$$

$$\text{s.t. } \mathbf{y} = \sum_{k=0}^{d-1} \beta_k c_k \mathbf{\Phi}_k^u \tag{2}$$

where $\mathbf{\Phi}^u$ is the column vector of features $[\phi_0^u(x), \phi_1^u(x), \ldots, \phi_{d-1}^u(x)]^\top$ which are orthonormal with respect to the test distribution.

Now, we want to **learn c** which minimizes the expectation for $\hat{\boldsymbol{\beta}}$ over all tasks,

$$\underset{\mathbf{c}}{\operatorname{argmin}} \, \mathbb{E}_{\mathcal{D}_T} \left[\mathcal{L}_T \left(\hat{\boldsymbol{\beta}}_T, \mathbf{c} \right) \right]$$

where $\mathcal{L}_T \left(\hat{\boldsymbol{\beta}}_T, \mathbf{c} \right)$ is the loss from learning $\hat{\boldsymbol{\beta}}$ for a specific task with the original formulation and a given $\mathbf{c}$ vector. $\mathbf{c}$ is shared across all tasks and is what we will optimize with meta-learning.

There are many machine learning techniques which can fall under the nebulous heading of meta-learning, but we will focus on one with Berkeley roots called Model Agnostic Meta-Learning (MAML)[1] which optimizes the initial weights of a network to rapidly converge to low loss within the task distribution. The MAML algorithm as described by the original paper is shown in Fig. 1.

```
Algorithm 1 Model-Agnostic Meta-Learning
Require: $p(\mathcal{T})$: distribution over tasks
Require: $\alpha$, $\beta$: step size hyperparameters
1: randomly initialize $\theta$
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
```

Figure 1: MAML algorithm. We will refer to the training steps on line 6 as the *inner update*, and the training step on line 8 as the *meta update*.

At a high level, MAML works by sampling a "mini-batch" of tasks $\{T_i\}$ and using regular gradient descent updates to find a new set of parameters θ_i for each task starting from the same initialization θ. Then the gradient w.r.t. the original θ each calculated for each task using the task-specific updated weights θ_i, and θ is updated with these 'meta' gradients. Fig. 2 illustrates the path the weights take with these updates.

[1] C. Finn, P. Abbeel, S. Levine, "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks," in *Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017*

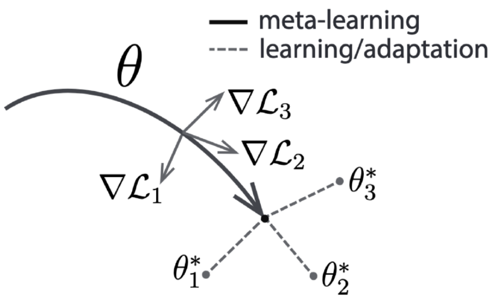

Figure 2: MAML gradient trajectory illustration

The end goal is to produce weights θ^* which can reach a state useful for a particular task from $\mathcal{D}_T$ after a few steps — needing to use less data to learn. If you want to understand the fine details of the algorithm and implementation, we recommend reading the original paper and diving into the code provided with this problem.

(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 $\mathbf{y}$.

Solution:

$$\hat{\boldsymbol{\beta}} = A^\top (AA^\top)^{-1}\mathbf{y}$$

is the min-norm solution.

(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 $\hat{\boldsymbol{\beta}} = \frac{1}{c_0^2+c_1^2}\begin{bmatrix} c_0 \\ c_1 \end{bmatrix}$

Solution: Since $\phi_1^u(x) = \phi_2^u(x) \equiv \phi^u(x)$, we plug into the min-norm solution which is

$$\begin{aligned}
\hat{\boldsymbol{\beta}} &= \begin{bmatrix} c_0\phi^u(x) \\ c_1\phi^u(x) \end{bmatrix} \left(\begin{bmatrix} c_0\phi^u(x) & c_1\phi^u(x) \end{bmatrix} \begin{bmatrix} c_0\phi^u(x) \\ c_1\phi^u(x) \end{bmatrix} \right)^{-1} y \\
&= \phi^u(x)^2 \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} \left(\phi^u(x)^2 \left(c_0^2 + c_1^2 \right) \right)^{-1} \\
&= \frac{1}{c_0^2 + c_1^2} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix}
\end{aligned}$$

The least-squares $\hat{\boldsymbol{\beta}}$ subject to the constraint $\phi^u(x) = \beta_0 c_0 \phi^u(x) + \beta_1 c_1 \phi^u(x)$ (which you could also find by using the Lagrangian and KKT conditions) is

$$\hat{\beta}_0 = \frac{c_0}{c_0^2 + c_1^2}$$

$$\hat{\beta}_1 = \frac{c_1}{c_0^2 + c_1^2}$$

(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 $\boldsymbol{\beta}$ from the previous part. *(Hint: the features $\phi_i^u(x)$ are orthonormal under the test distribution. They are not identical here.)*

Solution: Plugging $\hat{\boldsymbol{\beta}}$ from part (b) into the expected test error, we have

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

Now we use the orthogonality of the features under the test distribution to evaluate the expected norm.

$$\begin{aligned}
\mathcal{E} &= \mathbb{E}_{x_{test},y_{test}}\left[\frac{1}{2}\left\|\phi_t^u(x) - \frac{c_0^2}{c_0^2+c_1^2}\phi_t^u(x) - \frac{c_1^2}{c_0^2+c_1^2}\phi_a^u(x)\right\|_2^2\right] \\
&= \frac{1}{2}\left(1 - \frac{c_0^2}{c_0^2+c_1^2}\right)^2 + \frac{1}{2}\left(\frac{c_1^2}{c_0^2+c_1^2}\right)^2 \\
&= \left(\frac{c_1^2}{c_0^2+c_1^2}\right)^2 \\
&= \frac{c_1^4}{(c_0^2+c_1^2)^2}
\end{aligned}$$

Calculating the gradient, we have

$$\frac{\mathrm{d}\mathcal{E}}{\mathrm{d}c_0} = \frac{-4c_0 c_1^4}{\left(c_0^2+c_1^2\right)^3}$$

$$\frac{\mathrm{d}\mathcal{E}}{\mathrm{d}c_1} = \frac{4c_0^2 c_1^3}{\left(c_0^2+c_1^2\right)^3}$$

(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 $\boldsymbol{\beta}$ go to?

Solution:

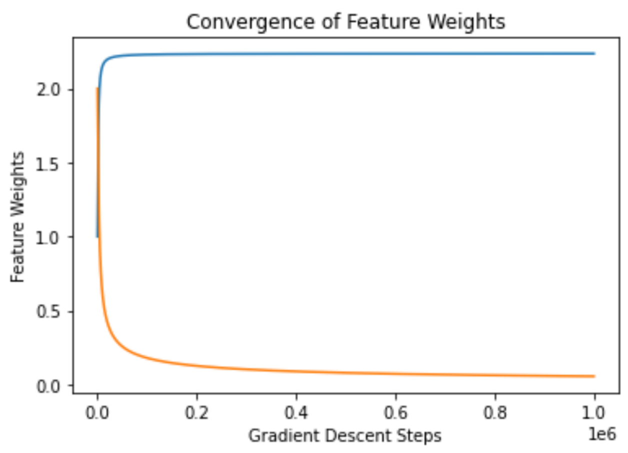

Figure 3: Asymptotic behavior of the feature weights with learning rate 0.001. $c_1 \to \|\mathbf{c}^{(0)}\|$ and $c_1 \to 0$.

As $c_1 \to 0$, $\hat{\beta}_0 \to \frac{1}{c_0}$ and $\hat{\beta}_1 \to 0$.

The rest of the parts (e)-(h) are in the accompanying jupyter notebooks 'metalearning-part1.ipynb' and 'metalearning-part2.ipynb'. Follow the instructions in the notebook and write your answers down in your written submission. In part 1, you do not have to write any code, just run the cells and write your observations. In part 2, you will modify provided regression code to perform classification.

(e) **Meta-learning with closed-from min-norm least squares regression, grid-spaced training data and uniform random meta update and test data.**

1. Based on the plot of regression test loss vs n_train_post, how do the meta-learned feature weights perform compared to the case where all feature weights are 1? How do the meta-learned feature weights perform compared to the oracle (that performs regression only using the features present in the data)? Why is there a downward spike at n_train_post = 32?

2. By looking at the evolution of the feature-weights with time as we perform meta-learning can you justify the improvement in performance? In particular, can you explain why some feature weights are being sent towards zero?

Solution: 1. By looking at the plot of test loss vs n_train_post we see that using the meta-learned feature weights the test performance is better than when we use the all 1s feature weights but worse than the oracle. As we increase n the effect of feature weights is less prominent and in all cases our test error goes down. There is a prominent downward spike at n_train_post = 32 since we do the training on inner tasks with n_train_inner = 32. The feature weights evolve during the meta learning process to downweight the aliases of the true feature towards 0 (when n = 32) which results in the performance being close to that of the oracle for this particular value of n.

2. We see that the weight on the favored features grows throughout the meta learning process but what is unique to this setting is the weights on the aliases of the true feature (corresponding to n=32) are getting downweighted towards 0. The weights on the other features are largely unchanged.

(f) **Replacing the closed-form solution with gradient descent and investigating the impact of the number of gradient descent steps on meta-learning success.**

With num_gd_steps = 5 does meta-learning help improve performance during test time? What happens if instead we use num_gd_steps = 1. Does meta-learning still work?

Solution: We observe that for both values of num_gd_steps (1 and 5) meta learning helps us improve performance. From the previous cell observe that with num_gd_steps=1, the solution using gradient descent is not the same as the closed form solution but this does not deter the meta learning procedure. For a sanity check you can try running the cell with num_gd_steps = 0 and observe that in this case we don't learn anything.

(g) **Meta-learning for classification with copy-pasted regression code.**
Follow instructions in the jupyter notebook.

Solution: See released code.

(h) **Meta-learning for classification with logistic regression.**
1. Based on the plot of classification error vs n_train_post, how do the meta-learned feature weights perform compared to the case where all feature weights are 1? How do the meta-learned feature weights perform compared to the oracle (that performs logistic regression only using the features present in the data)?
2. By looking at the evolution of the feature-weights with time as we perform meta-learning can you justify the improvement in performance? In particular, can you explain why some feature weights are being sent towards zero?

Solution: 1. By looking at the plot of classification error vs n_train_post we see that using the meta-learned feature weights the test performance is better than when we use the all 1s feature weights but worse than the oracle. As we increase n the effect of feature weights is less prominent and in all cases our classification error goes down.
2. We see that the weight on the favored features grows throughout the meta learning process and the weights on the other features are slightly decreased. The feature weight vector after meta learning is moving towards the oracle feature weights (1 on favored features and 0 on other featuers) and this explains the improvement in performance.

3. Homework Process and Study Group

Citing sources and collaborators are an important part of life, including being a student!
We also want to understand what resources you find helpful and how much time homework is taking, so we can change things in the future if possible.

(a) **What sources (if any) did you use as you worked through the homework?**

(b) **If you worked with someone on this homework, who did you work with?**
List names and student ID's. (In case of homework party, you can also just describe the group.)

(c) **Roughly how many total hours did you work on this homework? Write it down here where you'll need to remember it for the self-grade form.**

Contributors:

- CS182 Staff from past semesters.
- Jake Austin.
- Olivia Watkins.
- Anant Sahai.
- Saagar Sanghavi.

- Vignesh Subramanian.
- Josh Sanz.
- Ana Tudor.
- Mandi Zhao.