CS330 Autumn 2020 Homework 1
Data Processing and Black-Box Meta-Learning

Due Wednesday September 30, 11:59 PM PST

SUNet ID:

Name:

Collaborators:

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

Overview

Goals: In this assignment, we will look at meta-learing for few shot classification. You will:

1. Learn how to process and partition data for meta learning problems, where training is done over a distribution of training tasks $p(\mathcal{T})$.
2. Implement and train memory augmented neural networks, a black-box meta-learner that uses a recurrent neural network.
3. Analyze the learning performance for different size problems.
4. Experiment with model parameters and explore how they improve performance.

We will be working with Omniglot [1], a dataset with 1623 characters from 50 different languages. Each character has 20 28x28 images. We are interested in training models for K-shot, N-way classification, i.e. training a classifier to distinguish between N previously unseen characters, given only K labeled examples of each character.

Submission: To submit your homework, submit one PDF report to Gradescope containing written answers/plots to the questions below. The PDF should also include your name and any students you talked to or collaborated with.

This year, we are releasing assignments as Google Colab notebooks. The Colab notebook for this assignment can be found here: `https://colab.research.google.com/drive/1slBqgKa20iTatoWThMWZTnFysgAVD1vh?usp=sharing`. As noted in the notebook instructions, you will need to save a copy of the notebook to your Google Drive in order to make edits and complete your submission. **When you submit your PDF responses on Gradescope, please include a clearly-visible link to your Colab notebook so we may examine your code. Any written responses or plots to the questions below must appear in your PDF submission.**

Problem 1: Data Processing for Few-Shot Classification

Before training any models, you must write code to sample batches for training. Fill in the `sample_batch` function in the `DataGenerator` class. The class already has variables defined

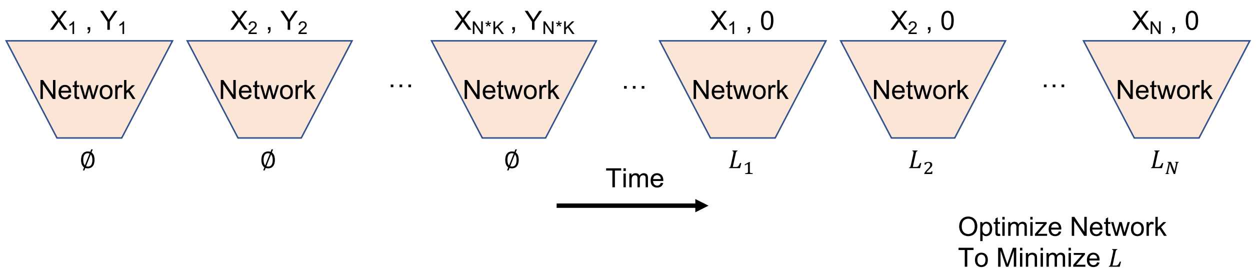

Figure 1: Feed K labeled examples of each of N classes through network with memory. Then feed final set of N examples and optimize to minimize loss.

for batch size `batch_size` (B), number of classes `num_classes` (N), and number of samples per class `num_samples_per_class` (K). Your code should

1. Sample N different classes from either the specified train, test, or validation folders.
2. Load K images per class and collect the associated labels
3. Format the data and return two numpy matrices, one of flattened images with shape $[B, K, N, 784]$ and one of one-hot labels $[B, K, N, N]$

Helper functions are provided to (1) take a list of folders and provide paths to image files/labels, and (2) to take an image file path and return a flattened numpy matrix. **For best performance in problem 2, only shuffle the K training examples within each of the N classes, and do not shuffle the order of examples across classes.**[1]

Problem 2: Memory Augmented Neural Networks [2, 3]

We will be attempting few shot classification using memory augmented neural networks. The idea of memory augmented networks is to use a classifier with recurrent memory, such that information from the K training examples of each class informs classification through the hidden state of the network.

The data processing will be done as in SNAIL [3]. Specifically, during training, you sample batches of N classes with $K+1$ samples per class. Each set of labels and images are concatenated together, and then $N * K$ of these concatenated pairs are sequentially passed through the network (as the task training set). Then the final test example of each class is fed through the network, concatenated with 0 instead of the true label. The loss is computed between these final outputs and the ground truth label, which is then backpropagated through the network. **Note**: The loss is *only* computed on the last set of N images.

[1]Theoretically-speaking, it should be fine to shuffle the order of all of the training examples per task. But, in practice, this makes the optimization much harder.

The idea is that the network will learn how to encode the first K examples of each class into memory such that it can be used to enable accurate classification on the $K + 1$th example. See Figure 1.

In the `hw1.py` file:

1. Fill in the `call` function of the `MANN` class to take in image tensor of shape $[B, K + 1, N, 784]$ and a label tensor of shape $[B, K + 1, N, N]$ and output labels of shape $[B, K + 1, N, N]$. The layers to use have already been defined for you in the `__init__` function. *Hint: Remember to pass zeros, not the ground truth labels for the final N examples.*

2. Fill in the function called `loss_function` in the `MANN` class which takes as input the $[B, K + 1, N, N]$ labels and $[B, K + 1, N, N]$ and computes the cross entropy loss only on the N test images.

Note: Both of the above functions will need to backpropogated through, so they need to be written in TensorFlow in a differentiable way.

Problem 3: Analysis

Once you have completed problems 1 and 2, you can train your few shot classification model. You should observe both the train and testing loss go down, and the test accuracy go up. Now we will examine how the performance varies for different size problems. Train models for the following values of K and N:

- $K = 1$, $N = 2$
- $K = 1$, $N = 3$
- $K = 1$, $N = 4$
- $K = 5$, $N = 4$

For each configuration, submit a plot of the test accuracy over iterations. Write down your observations.

Problem 4: Experimentation

a Experiment with one hyperparameter that affects the performance of the model, such as the type of recurrent layer, size of hidden state, learning rate, number of layers. Show learning curves of how the test accuracy of the model changes on 1-shot, 3-way classification as you change the parameter. Provide a brief rationale for why you chose the parameter and what you observed in the caption for the graph.

b **Extra Credit:** You can now change the MANN architecture however you want (including adding convolutions). Can you achieve over 60% test accuracy on 1-shot, 5-way classification using fewer optimization steps?

References

[1] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[2] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1842–1850, New York, New York, USA, 20–22 Jun 2016. PMLR.

[3] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. Meta-learning with temporal convolutions. *CoRR*, abs/1707.03141, 2017.