CS330 Autumn 2022 Homework 1
Data Processing and Black-Box Meta-Learning

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-learning 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 [1].
3. Analyze the learning performance for different size problems.
4. Experiment with model parameters and explore how they improve performance.

We have provided you with the starter code, which can be downloaded from the course website. We will be working with Omniglot [2], 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.

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` function in the `DataGenerator` class. The class already has variables defined for batch size `batch_size` (B), number of classes `num_classes` (N), and number of samples per class `num_samples_per_class` ($K+1$). Your code should:

1. Sample N different characters from either the specified train, test, or validation folder.
2. Load $K+1$ images per character and collect the associated labels, using K images per class for the support set and 1 image per class for the query set.
3. Format the data and return two tensors, one of flattened images with shape $[K+1, N, 784]$ and one of one-hot labels $[K+1, N, N]$.

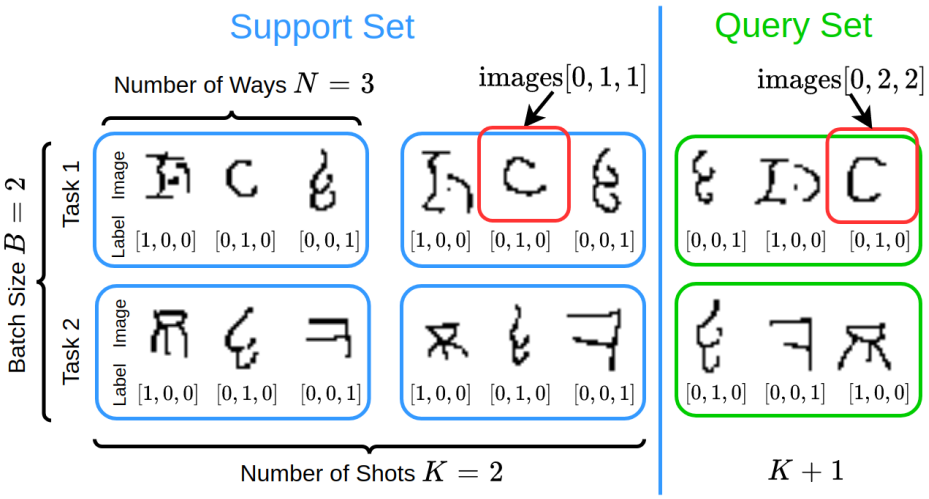

Figure 1: Example data batch from the Data Generator. The first K sets of images form the support set and are passed in the *same order*. The final set of images forms the query set and must be shuffled.

Note that your code only needs to return one single (image, label) tuple. We batch the inputs using an instance of torch.utils.data.DataLoader, and the final shape input images is $[B, K + 1, N, 784]$, and that of the input labels is $[B, K + 1, N, N]$, where B is the batch size.

Figure 1 illustrates the data organization. In this example, we have: (1) images from $N = 3$ different classes; (2) we are provided $K = 2$ sets of labeled images in the support set and (3) our batch consists of only two tasks, i.e. $B = 2$.

1. We will sample both the support and query sets as a single batch, hence one batch element should obtain image and label tensors of shapes $[K+1, N, 784]$ and $[K+1, N, N]$ respectively. In the example of Fig. 1, `images[0, 1, 1]` would be the image of the letter "C" in the support set with corresponding class label $[0, 1, 0]$ and `images[0, 2, 2]` would be the the letter "C" in the query set (with the same label).

2. We must shuffle the order of examples in the **query set**, as otherwise the network can learn to output the same sequence of classes and achieve 100% accuracy, without actually learning to recognize the images. If you get 100% accuracy, you likely did not shuffle the query data correctly. In principle, you should be able to shuffle the order of data in the support set as well; however, this makes the model optimization much harder. **You should feed the support set examples in the same, fixed order**. In the example above, the support set examples are always in the same order.

We provide helper functions 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. The functions `np.random.shuffle` and `np.eye` will also be helpful. **Be careful about output shapes and data types!**

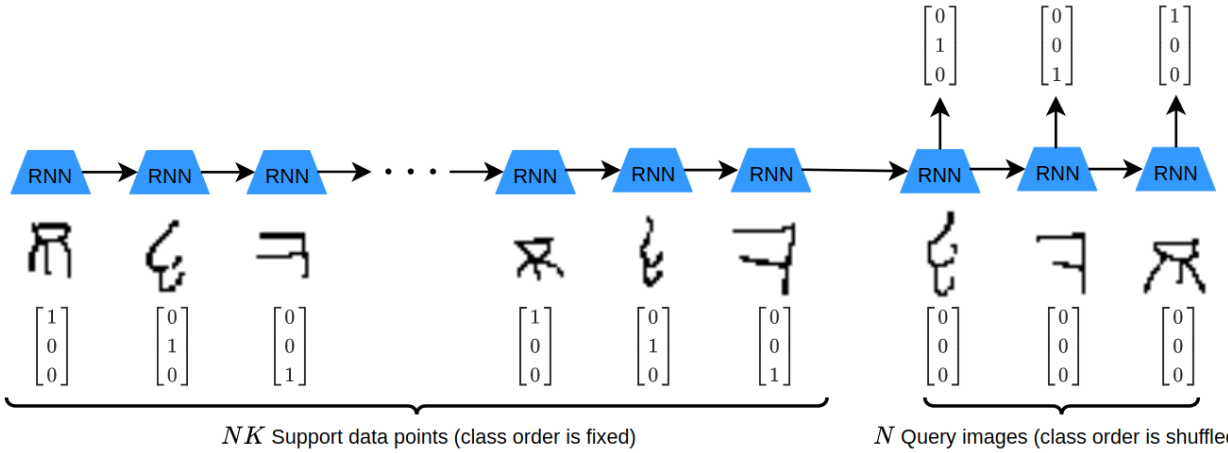

Figure 2: Feed K labeled examples of each of N classes through the memory-augmented network. Then feed final set of N examples and optimize to minimize loss.

Problem 2: Memory Augmented Neural Networks (MANN) [1, 3]

We will now be implementing few-shot classification using memory augmented neural networks (MANNs). **The main idea of MANN is that the network should learn how to encode the first K examples of each class into memory such that it can be used to accurately classify the $K+1$th example.** See Figure 2 for a graphical representation of this process.

Data processing will be done as in SNAIL [3]. Each set of labels and images are concatenated together, and the $N * K$ support set examples are sequentially passed through the network as shown in Fig. 2. Then the query example of each class is fed through the network, **concatenated with 0 instead of the true label**. The loss is computed between the query set predictions and the ground truth labels, which is then backpropagated through the network. **Note**: The loss is *only* computed on the set of N query images, which comprise of the last examples from each character class.
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 $[B, K+1, N, N]$ predicted labels and computes the cross entropy loss only on the N test images.

Note: Both of the above functions will need to be backpropogated through, so they need to be written in PyTorch 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 support and query losses go down, and the query 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:

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

Example code:
`python hw1.py --num_shot K --num_classes N`
For checking training results and/or taking a screenshot for the writeup, use:
`tensorboard --logdir runs/`

You should start with the case $K = 1, N = 2$ as it can aid you in the implementation and debugging process. Your model should be able to achieve a query set accuracy of above 90% in this first two scenarios scenario on held-out test tasks, around 80% in the second scenario, and around 70% in the final scenario.

For each configuration, submit a plot of the meta-test query set classification accuracy over training iterations (A TensorBoard screenshot is fine). Answer the following questions:

Your plot goes here.

1. How does increasing the number of classes affect learning and performance?

   Your answer goes here.

   Figure 3, 5 and Figure 6 shows that increasing the number of classes will make the model learning process slower and less effective.

2. How does increasing the number of examples in the support set affect performance?

   Your answer goes here.

   Figure 3 and Figure 4 shows that increasing the number of examples will improve the performance.

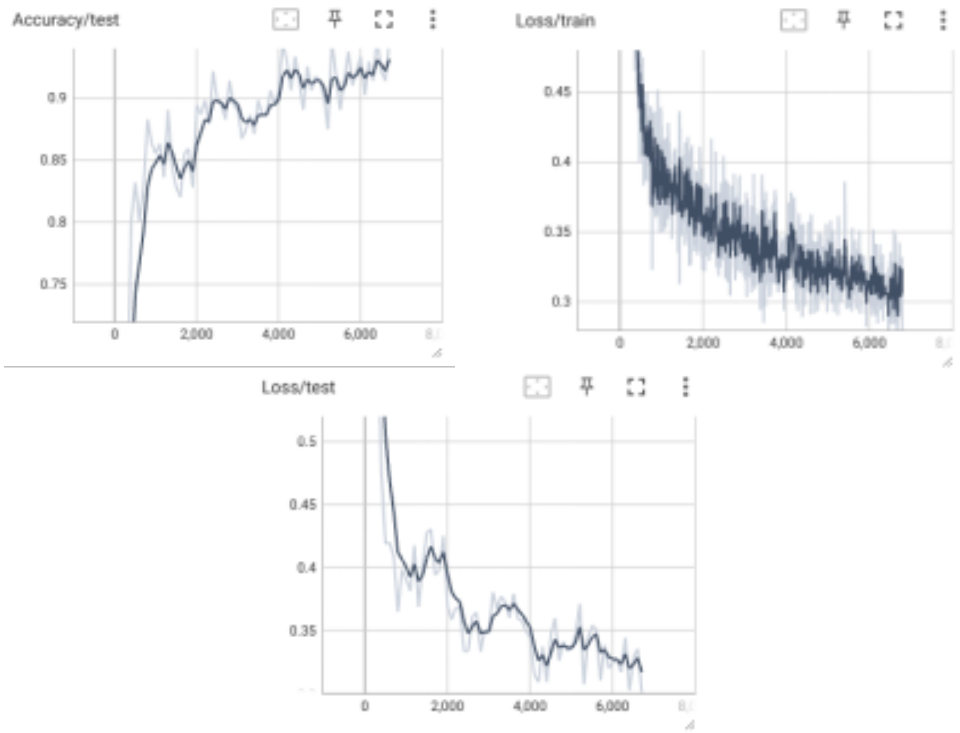

Figure 3: 2-Way 1-Shot

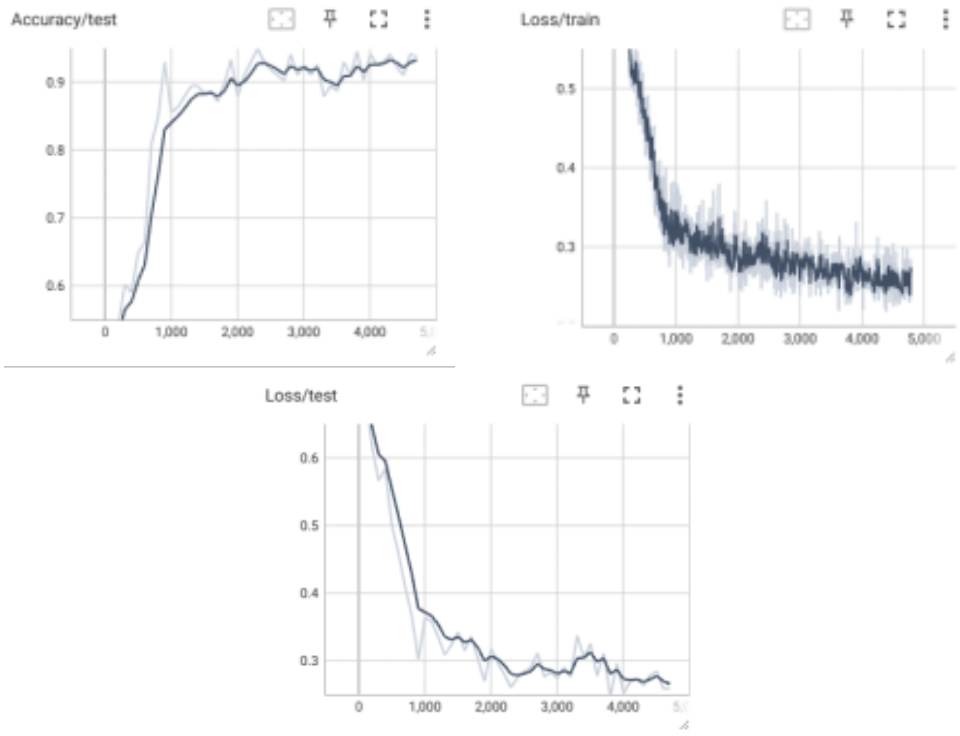

Figure 4: 2-Way 2-Shot

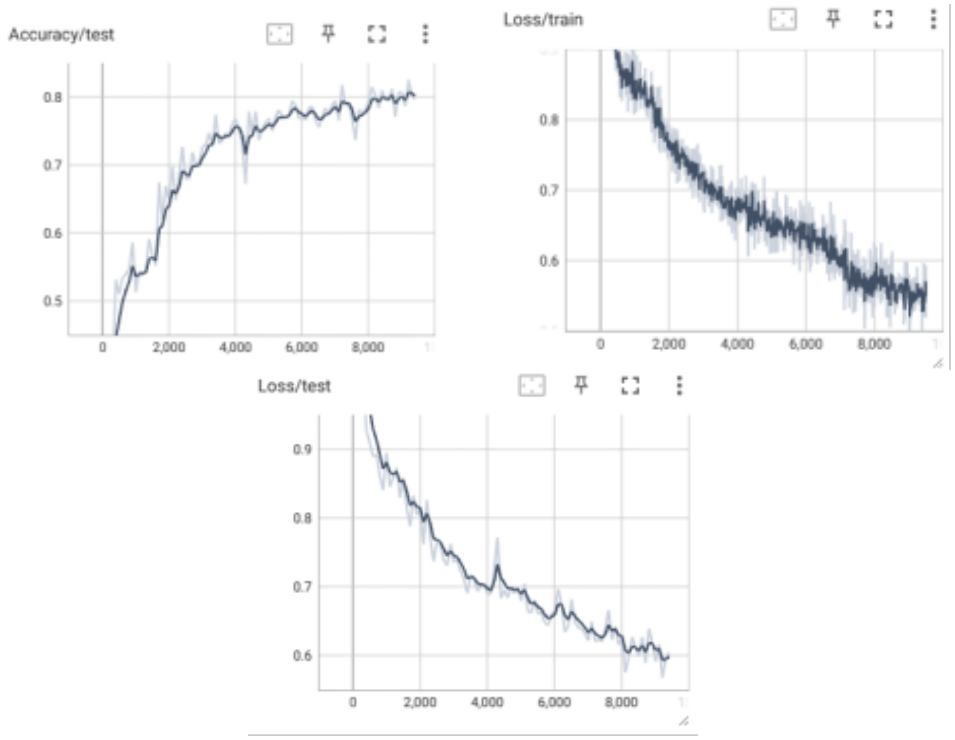

Figure 5: 3-Way 1-Shot, 128 hidden state

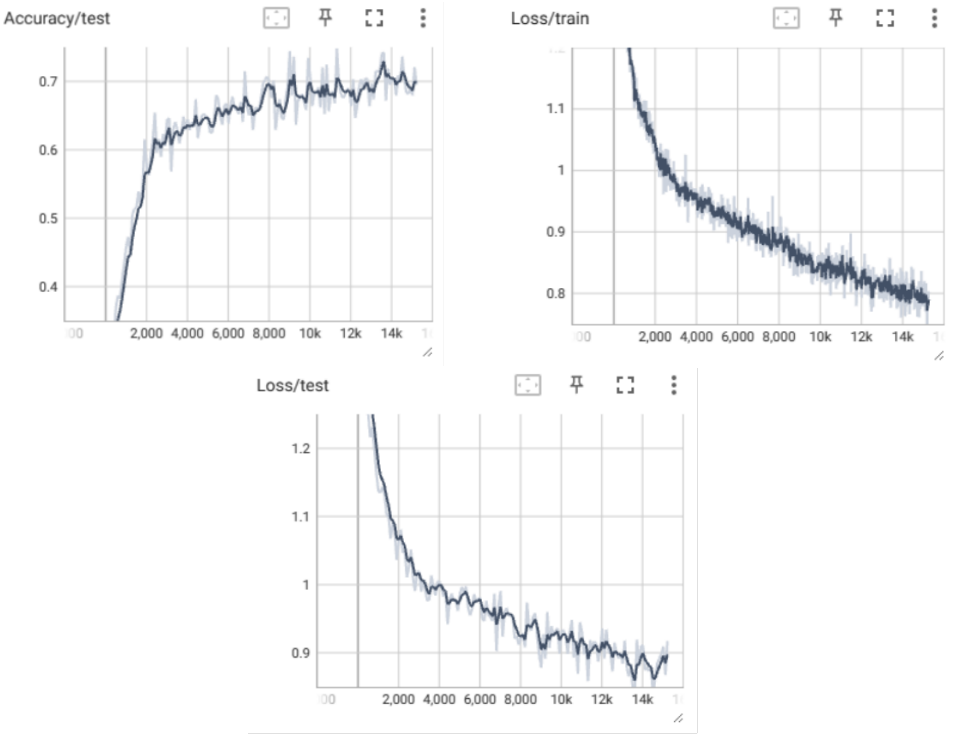

Figure 6: 4-Way 1-Shot

Problem 4: Experimentation

a In this question we'll explore the effect of memory representation on model performance. We will focus on the $K = 1$, $N = 3$ case.

In the previous experiments we used an LSTM model with 128 units. Consider additional memory sizes of 256, and 8. How does increasing and decreasing the memory capacity influence performance?

Your plot and answer goes here.

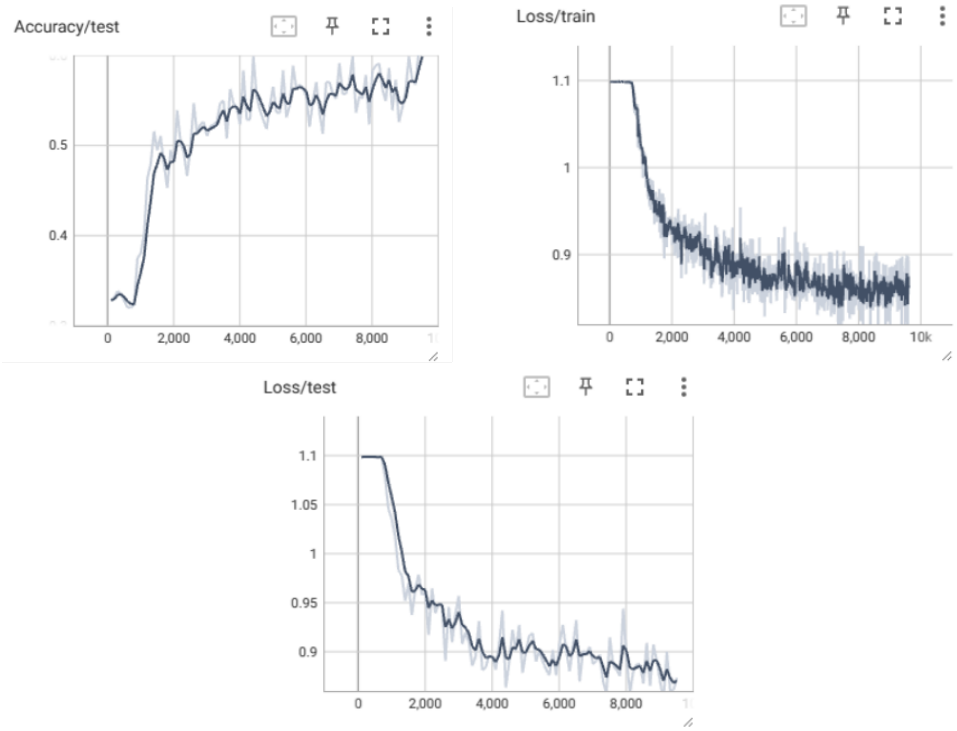

Figure 7: 3-Way 1-Shot, 8 hidden state

Figure 7 shows that decreasing the hidden state size to 8 causes lower performance while Figure 8 shows that increasing the hidden state size to 256 causes higher performance. It's because hidden state size determines the amount of information the model can remember in the support set.

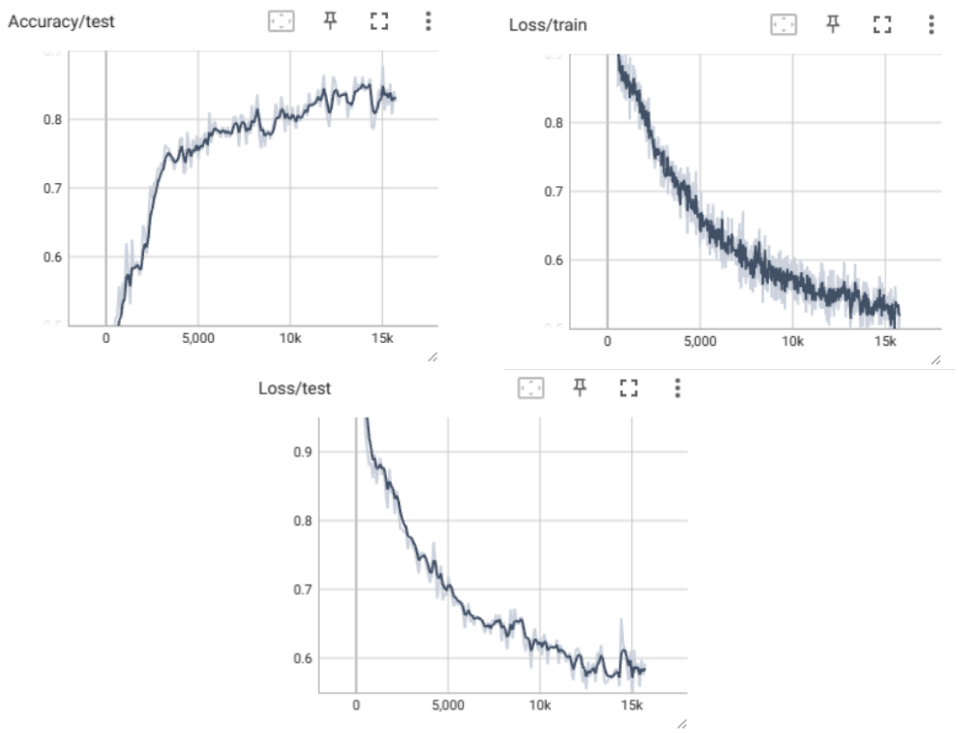

Figure 8: 3-Way 1-Shot, 256 hidden state

References

[1] 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.

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

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