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

Due Wednesday October 12, 11:59 PM PST

SUNet ID:	Jinang
Name:	Jinang Shah
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-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.

Submission: To submit your homework, submit one PDF report to Gradescope containing written answers and Tensorboard graphs (screenshots are fine) to the questions below, as well as the `hw1.py` and `load_data.py` scripts in a single zip file. The PDF should also include your name and any students you talked to or collaborated with. **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` 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:

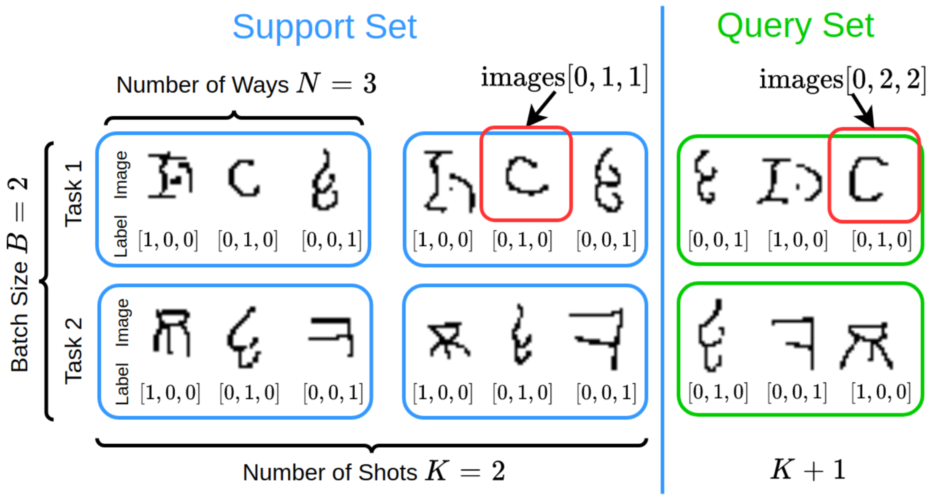


Figure 1: Example data batch from the Data Generator. The first K sets of images form the support set and are passed in the *same order*. The final set of images forms the query set and must be shuffled.

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 $[K + 1, N, N]$.

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

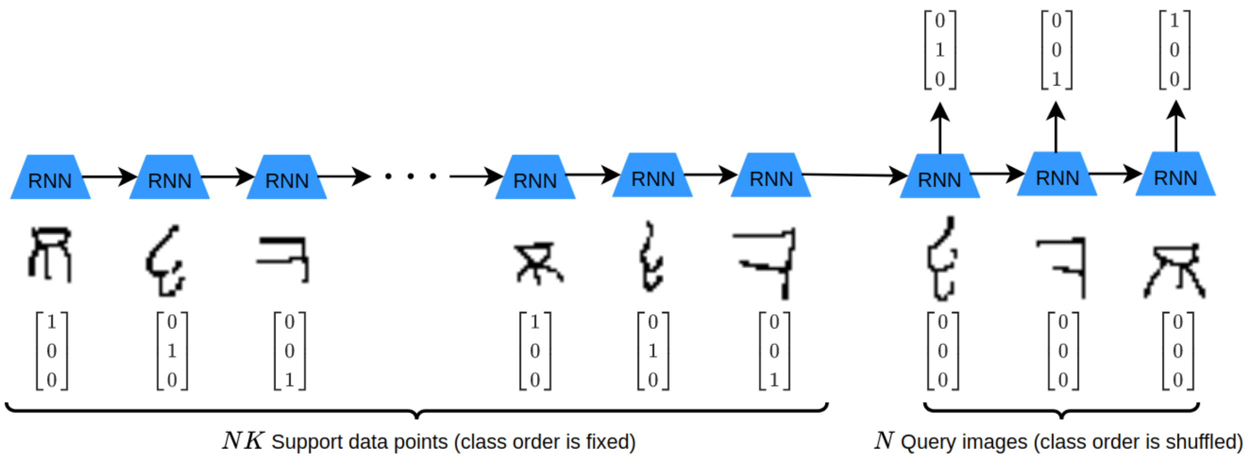

RNN
NK Support data points (class order is fixed)
N Query images (class order is shuffled)

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.

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!**

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:

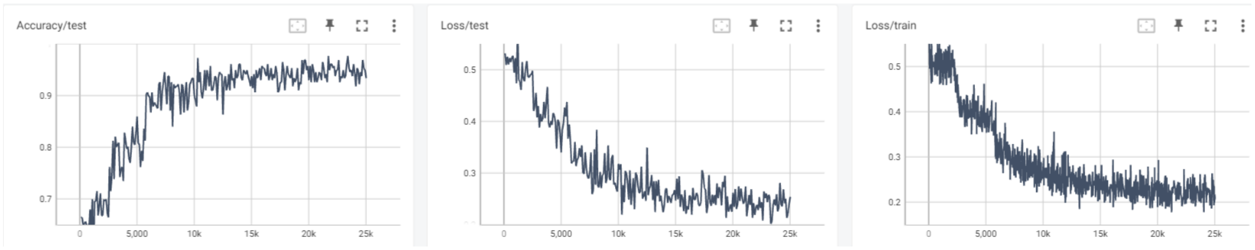


Figure 3: k=1 N=2

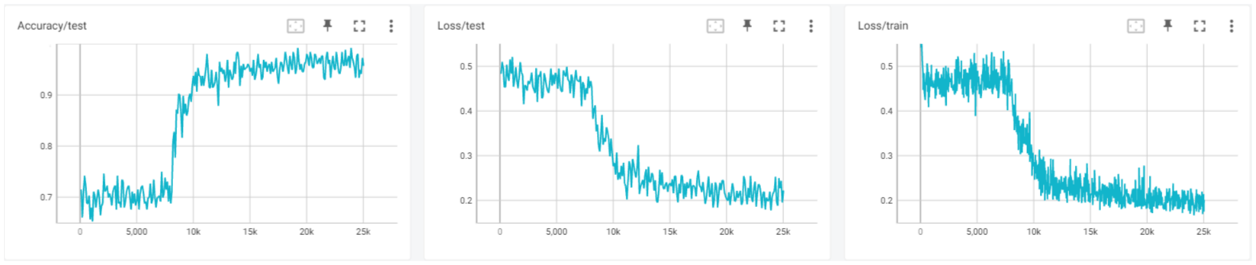


Figure 4: k=2 N=2

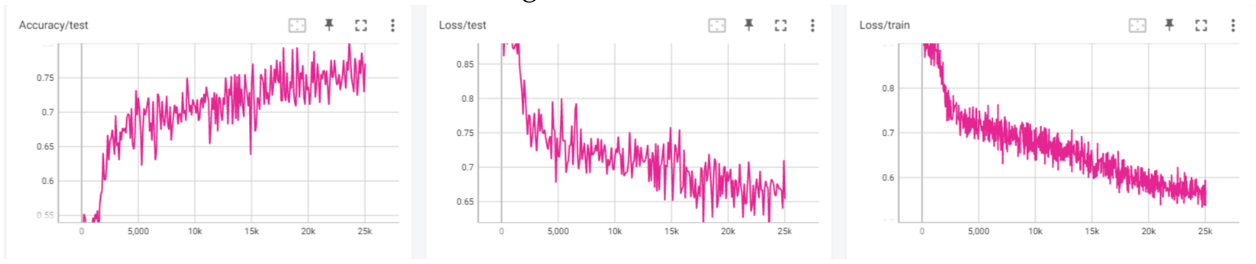


Figure 5: k=1 N=3

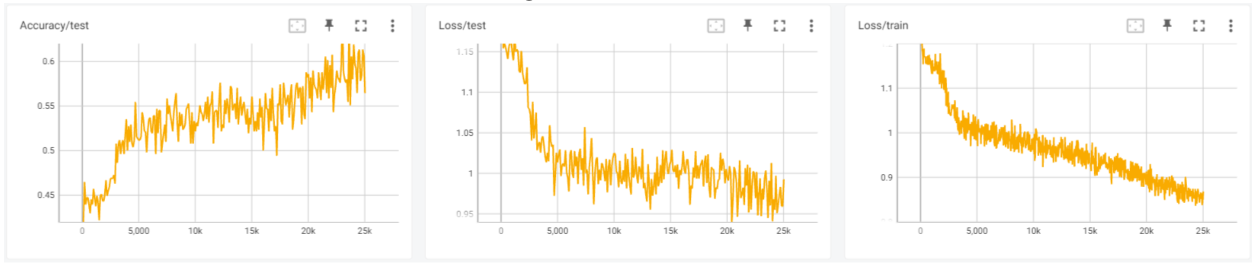


Figure 6: k=1 N=4

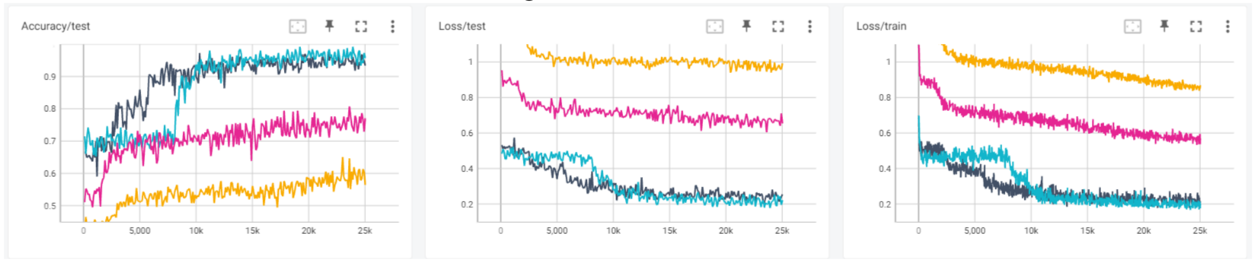


Figure 7: Combined

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

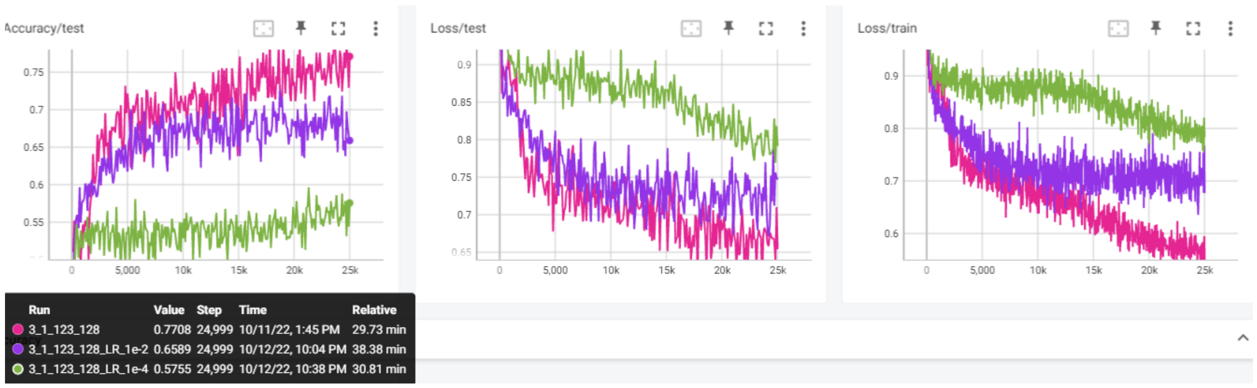
   Ans. Increasing number of classes **adversely** affects learning and performance. As clearly evident from figure 7, Performance decreases significantly during both test and train time as number of classes goes up. It is also expected as with just one sample available per class, model is finding it harder to estimate and generalise differentiators between classes.

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

   Ans. As expected, increasing number of samples for 2 classes from one-shot to two-shot, increases performance in test as well as train time, but **marginally.** Ideally, more number of support samples, better equips model to learn generalised differences between classes, but since to differentiate two classes, one-shot training already achieves good results, increasing support sample, although increases the performance but not significantly.

Problem 4: Experimentation

a Experiment with one hyper parameter that affects the performance of the model, such as the type of recurrent layer, size of hidden state, learning rate, or a number of layers. Submit a plot that shows how the meta-test query set classification 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 plot.



	Run	Value	Step	Time	Relative
●	3_1_123_128	0.7708	24,999	10/11/22, 1:45 PM	29.73 min
●	3_1_123_128_LR_1e-2	0.6589	24,999	10/12/22, 10:04 PM	38.38 min
●	3_1_123_128_LR_1e-4	0.5755	24,999	10/12/22, 10:38 PM	30.81 min

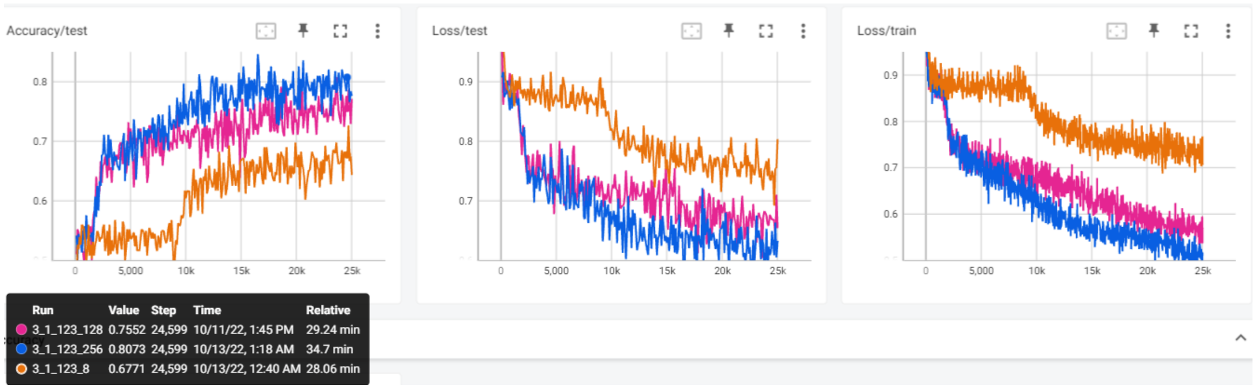
Figure 8: Varying Learning Rate 1e-2 (Purple),1e-3 (Pink), 1e-4 (Green)

Ans. Even with best models, data or optimization algorithm, present learning rate is one of the parameter that significantly affects model convergence and performance (from my past experience). I wanted to show even when not that far apart, different learning rates can lead to very different outcomes. Therefore, I examined learning rate vs Performance for 1-shot, 3-way classification.

Increasing or decreasing learning rate from 1e-3, both adversely affect performance. Some rational would be, higher learning rate (1e-2), although converges fast at a very early stage, due to large gradient updates, it could never reach to optimal minima, whereas lower learning rate, dramatically reduce convergence speed, and might threaten model to get stuck in a local minima. Specifically for figure 8, lower learning rate seems to be initially get stuck, and then starting to improve at near end, whereas higher learning rate seems to converge very fast but at a suboptimal solution.

b **Extra Credit:** 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?



	Run	Value	Step	Time	Relative
●	3_1_123_128	0.7552	24,599	10/11/22, 1:45 PM	29.24 min
●	3_1_123_256	0.8073	24,599	10/13/22, 1:18 AM	34.7 min
●	3_1_123_8	0.6771	24,599	10/13/22, 12:40 AM	28.06 min

Figure 9: Varying Hidden Size 8 (Orange),128 (Pink), 256 (Blue)

Ans. As evident from figure 9, increasing memory size to 256 increases and decreasing it to 8 drastically reduce the performance and learning of LSTM model both during train and test times. Memory Size in general is used for model to remember information from it's past predictions. For meta-learning, it essentially encompasses knowledge, model has learned from it's past samples. Depending upon application, optimal memory size might vary based on how much past information is needed to effectively predict query samples. From figure, we can say that since using 256 nodes don't produce too better a result, 128 nodes is also able to store relatively good amount of past information; Whereas 8 nodes are essentially not even near enough to store past information from class samples to produce good performance over query samples.

Note. A windows-specific error "BrokenPipe2" was observed during training for parallel processing of dataloader. When ran on 0 workers, unexpectedly, training not just took more time but took more steps to converge. An unusual training graph was observed especially for case k=2 N=2.

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.