CS 330 Autumn 2021/2022 Homework 1

Data Processing and Black-Box Meta-Learning
Due Wednesday October 6, 11:59 PM PST

SUNetID: tminh
Name: Minh Tran
Collaborators: N/A

October 6, 2021

Abstract

The document contains solutions of classification task for the Omniglot dataset using black-box meta-learning approach for few shot learning. The first part is the explanation of the data generator, the second part is the implementation of a memory augmented neural network, the third and fourth part are the results of the model for different settings and insights from these experiments.

1 Data Processing for Few-Shot Classification

The function sample_batch in class DataGenerator takes input batch type (train, test, val) and batch number B, the function then will sample K+1 images for each character class N, during sampling, the function also flattens images to 1 dimension vectors as well as encodes correspondence labels of these images to a 1 dimension vector of size N. The function returns two tensor of size [B, K+1, N, dim] and [B, K+1, N, N].

Notes for implementation:
+ Need to collect all sample in query set before shuffling.
+ Since output is an immutable object (Torch Tensor), additional processing is needed in the forward function.
Details will be found in "load_data.py".

2 Memory Augmented Neural Networks (MANN)

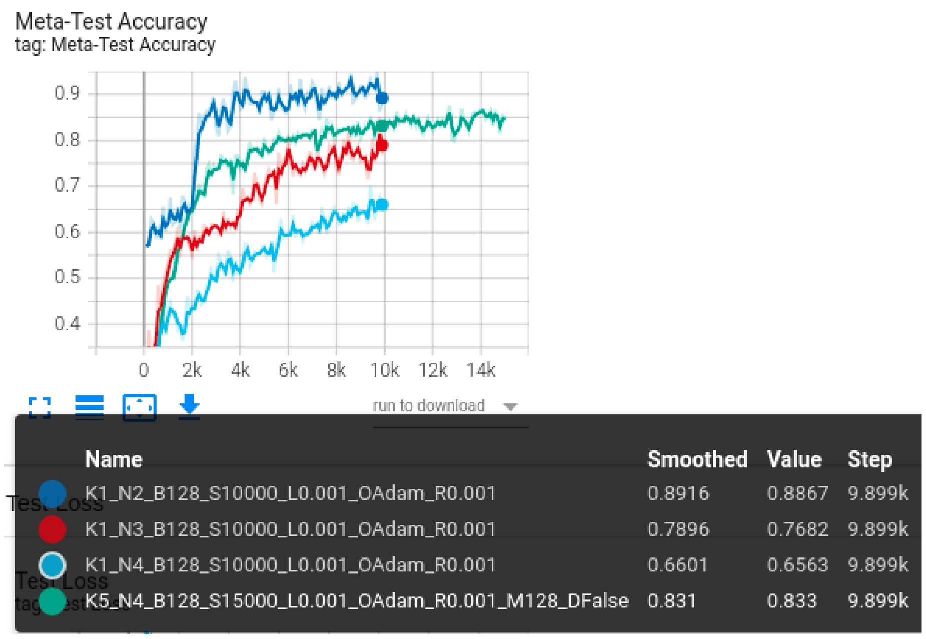
The function forward of class MANN takes two tensor input_images and input_labels which are the outputs of the sample_batch function in the DataGenerator, the batches of image vectors and label vectors then were reshaped and then concatenated to an input to our black-box LSTM model. However for the query set, all the labels were change to zeros vector of length classes number, we will use the query set to evaluate the performance of the model. After feeding the samples through the model, the function return the reshaped Tensor of shape [B, K+1, N, N]. The output tensor from forward function then will be passed to loss function, only the predictions and labels of the query set are chosen to calculate the cross entropy loss which is a scalar value.

Notes for implementation:
+ Since the output of sample_batch is Torch Tensor (immutable object), we need to convert label tensor to numpy to process then convert it back to Torch Tensor, this may cause some bottleneck in some cases when CPU and GPU are both used.
+ Label vector needs to be encoded using torch.argmax to be used in cross entropy loss function.
Details will be found in "multitask.py".

3 Analysis

Increasing number of classes and examples affects learning and performance

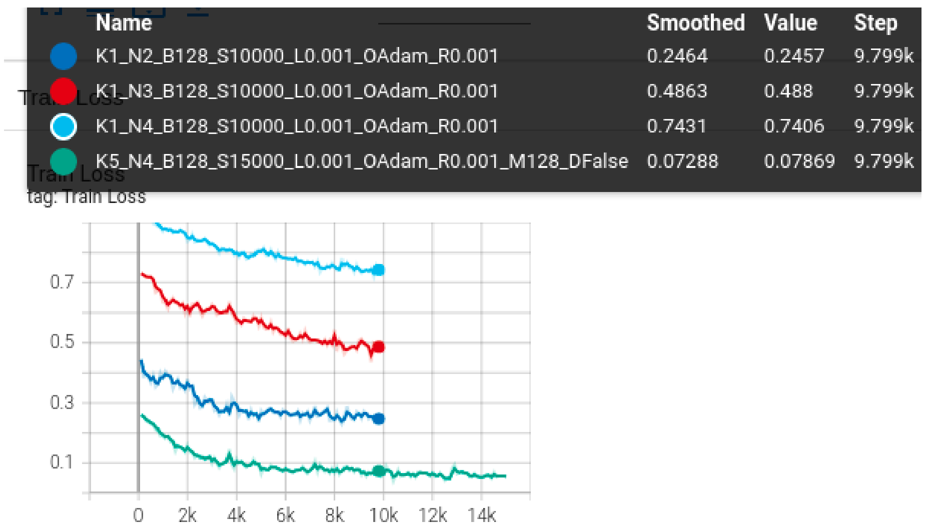
From the meta-test accuracy chart we can see that increasing number of class will make the learning process become slower, for just 1 class, the model reach more than 90% accuracy in around 10000 steps, while at the same spot, the settings with higher number of class will reach lower meta-test accuracy, around 70% to 80%, extra iterations could improve the performance since looking at the test loss chart, it is decreasing slowly but there is not sign of overfitting yet. Adding number of samples in support set improved performance of model compare to the setting which there is only one sample for each class in the support set, the difference is around 15% to 20% at step 10000, it makes sense since more samples provided more information about the data and the model will also have better generalization.



	Name	Smoothed	Value	Step
●	K1_N2_B128_S10000_L0.001_OAdam_R0.001	0.8916	0.8867	9.899k
●	K1_N3_B128_S10000_L0.001_OAdam_R0.001	0.7896	0.7682	9.899k
●	K1_N4_B128_S10000_L0.001_OAdam_R0.001	0.6601	0.6563	9.899k
●	K5_N4_B128_S15000_L0.001_OAdam_R0.001_M128_DFalse	0.831	0.833	9.899k

Figure 1: Meta-Test Accuracy

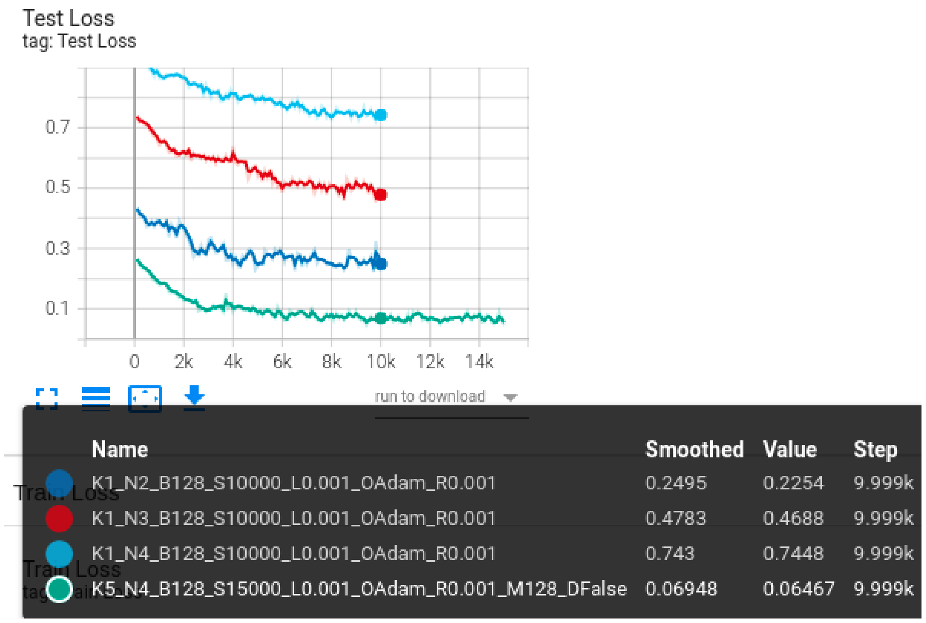
From the train loss chart we can see that adding more classes will make the initial loss higher as expected, but higher number classes will make the training loss less fluctuate which makes sense, since more classes help to improve the generalization of model. Increasing the number of examples in the support set in the examples where N = 4, in fact makes the model initialized with lower loss and also the loss is more stable. In conclusion, the more number of classes added, the more complex the model is, therefore the loss will be higher, however the high loss could be compensated by providing additional examples in the support set.



	Name	Smoothed	Value	Step
●	K1_N2_B128_S10000_L0.001_OAdam_R0.001	0.2464	0.2457	9.799k
●	K1_N3_B128_S10000_L0.001_OAdam_R0.001	0.4863	0.488	9.799k
○	K1_N4_B128_S10000_L0.001_OAdam_R0.001	0.7431	0.7406	9.799k
●	K5_N4_B128_S15000_L0.001_OAdam_R0.001_M128_DFalse	0.07288	0.07869	9.799k

Figure 2: Train Loss

From the test loss chart we can see the similar pattern as the train loss since there is no sign of over fitting yet. Adding more class will cause a higher loss in general while adding more support samples will cause lower and less fluctuate loss.

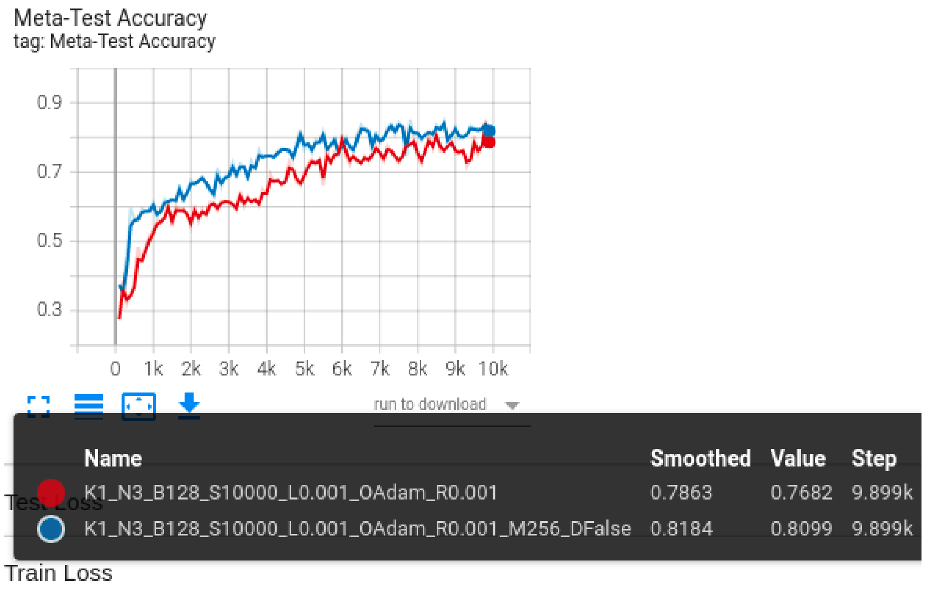


	Name	Smoothed	Value	Step
●	K1_N2_B128_S10000_L0.001_OAdam_R0.001	0.2495	0.2254	9.999k
●	K1_N3_B128_S10000_L0.001_OAdam_R0.001	0.4783	0.4688	9.999k
●	K1_N4_B128_S10000_L0.001_OAdam_R0.001	0.743	0.7448	9.999k
●	K5_N4_B128_S15000_L0.001_OAdam_R0.001_M128_DFalse	0.06948	0.06467	9.999k

Figure 3: Test Loss

4 Experimentation

Question 1:
From the chart we can see that increasing the memory capacity of model does improve model's performance slightly. Though the train/test loss look similar but the accuracy of bigger model is higher since the beginning of the training, the range of difference is about 5-10% and the gap decreased at the final steps.

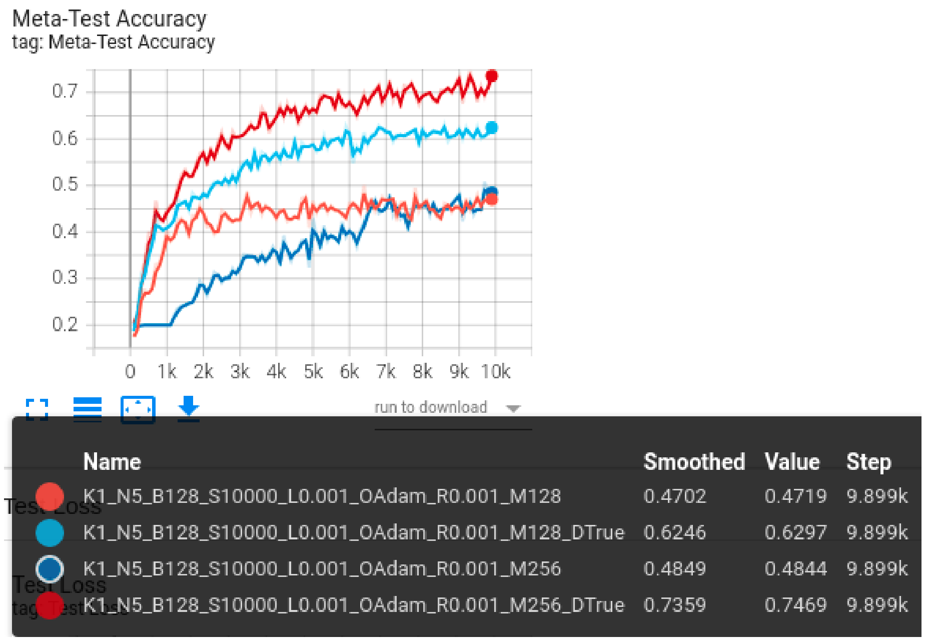


	Name	Smoothed	Value	Step
●	K1_N3_B128_S10000_L0.001_OAdam_R0.001	0.7863	0.7682	9.899k
●	K1_N3_B128_S10000_L0.001_OAdam_R0.001_M256_DFalse	0.8184	0.8099	9.899k

Figure 4: Meta-Test Accuracy of different hidden units

Question 2:
b1.

From the chart we can see that with our base model, adding hidden unit did not make a big difference in accuracy at step 10000, however the model is still converging so I expect the accuracy of the bigger model could be slightly better with additional iterations. In the other hand, for DNC model, adding additional hidden unit improved the accuracy significantly from early and at step 10000, the bigger model has 10% higher accuracy around 75% and the difference keeps increasing for later iterations.

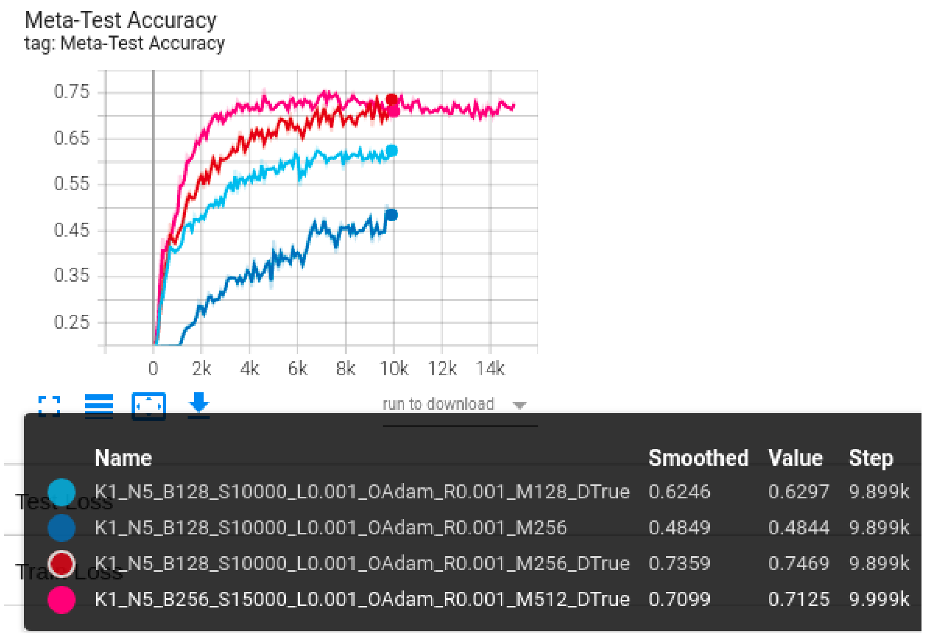


	Name	Smoothed	Value	Step
●	K1_N5_B128_S10000_L0.001_OAdam_R0.001_M128	0.4702	0.4719	9.899k
●	K1_N5_B128_S10000_L0.001_OAdam_R0.001_M128_DTrue	0.6246	0.6297	9.899k
●	K1_N5_B128_S10000_L0.001_OAdam_R0.001_M256	0.4849	0.4844	9.899k
●	K1_N5_B128_S10000_L0.001_OAdam_R0.001_M256_DTrue	0.7359	0.7469	9.899k

Figure 5: Meta-Test Accuracy of Normal and DNC models

b2.

I was able to reach only to 75% using the DNC model with 256 units while the model with 512 units reached the same but had sign of over fitting in the later iterations, I think the smaller DNC model of 128 units could reach better accuracy with many more iterations. However, even the smallest DNC model still bested the LSTM with 256 units, I think the memory structure and retrieval mechanism of DNC is much more better than LSTM, especially it can store rich information and longer sequence of complex data structure, make the model retrieves information and learn in a more efficient way than LSTM.



	Name	Smoothed	Value	Step
●	K1_N5_B128_S10000_L0.001_OAdam_R0.001_M128_DTrue	0.6246	0.6297	9.899k
●	K1_N5_B128_S10000_L0.001_OAdam_R0.001_M256	0.4849	0.4844	9.899k
●	K1_N5_B128_S10000_L0.001_OAdam_R0.001_M256_DTrue	0.7359	0.7469	9.899k
●	K1_N5_B256_S15000_L0.001_OAdam_R0.001_M512_DTrue	0.7099	0.7125	9.999k

Figure 6: Meta-Test Accuracy of Normal and DNC models