CS330 Autumn 2021 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 [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_batch` 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). Your code should:

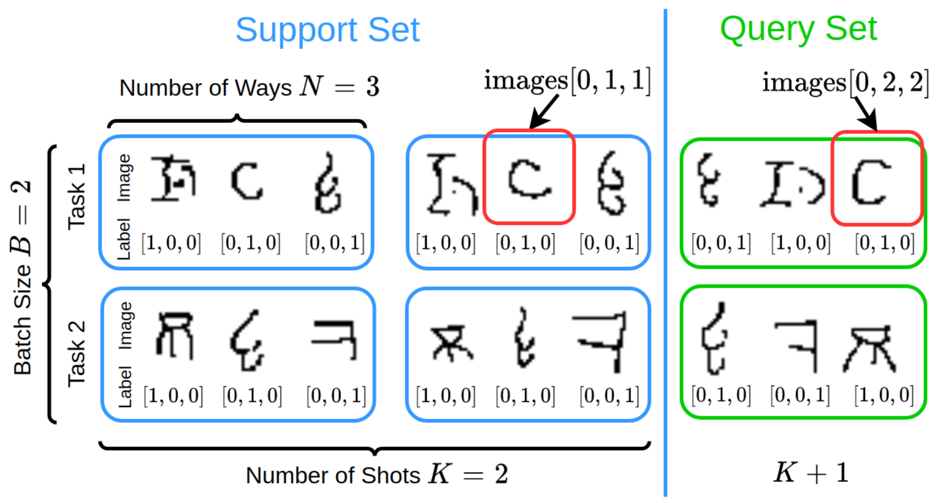

Figure 1: Example data batch from the Data Generator. The first K sets of images form the support set and are passed in the *same order*. The final set of images forms the query set and must be shuffled.

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 one image per class for the query set.
3. Format the data and return two numpy matrices, one of flattened images with shape $[B, K + 1, N, 784]$ and one of one-hot labels $[B, K + 1, N, N]$.

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 we should obtain image and label tensors of shapes $[B, K + 1, N, 784]$ and $[B, K + 1, N, N]$ 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 function `np.random.shuffle` will also be helpful. **Be careful about output shapes and data types!**

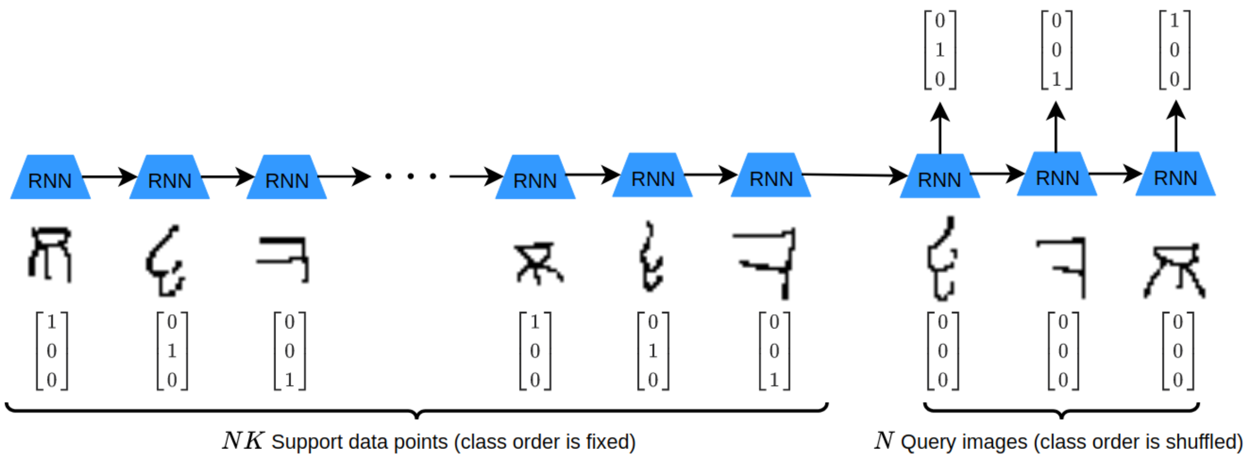
RNN
NK Support data points (class order is fixed)
N Query images (class order is shuffled)

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.

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 last set of N query images.
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 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:

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

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 about 90% in this first scenario on held-out test tasks, and accuracies of about 70% or higher in the other three cases.

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:

1. How does increasing the number of classes affect learning and performance?
2. How does increasing the number of examples in the support set affect performance?

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, or 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.

b **Extra Credit:** In this question we'll explore the effect of memory representation on model performance. We will focus on the $K = 1$, $N = 5$ case.

  b.1 In the previous experiments we used an LSTM model with 128 units. Consider a model with 256 hidden units. Does the increased memory capacity improve model performance?

b.2 A related family of MANNs utilize a recurrent controller network, which has read-write access to an external memory buffer, allowing them to better retain and retrieve information over longer horizons. We have provided you with an implementation of a Differentiable Neural Computer model [4]. The layer is already defined in the `MANN` class and you can use it as a regular recurrent layer. Can you use this model to achieve more than 80% accuracy? Compare your results with the LSTM model with 256 units. What conclusion can you draw about the different memory structures and retrieval mechanisms?

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.

[4] Robert Csordas and Jurgen Schmidhuber. Improving differentiable neural computers through memory masking, de-allocation, and link distribution sharpness control. *ICLR*, abs/1707.03141, 2019.