# CS 330 Autumn 2019/2020 Homework 1
# Data Processing and Memory Augmented Neural Networks
# Due Wednesday October 9, 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

In this assignment we will be looking 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, which meta-learn through a recurrent 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 the Omniglot dataset [1], a dataset for one-shot learning which contains 1623 different characters from 50 different languages. For each character there are 20 28x28 images. We are interested in training models for $K$-shot, $N$-way classification, that is, we want to train 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 and one zip file to GradeScope, where the report will contain answers to the deliverables listed below and the zip file contains your code (`hw1.py, load_data.py`) with the filled in solutions.

**Code Overview:** The code consists of two files

- `load_data.py`: Contains code to load batches of images and labels
- `hw1.py`: Contains the network architecture/loss functions and training script.

There is also the `omniglot_resized` folder which contains the data. *You should not modify this folder.*

**Dependencies:** We expect code in Python 3.5+ with `Pillow, scipy, numpy, tensorflow` installed.

## 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 in the `load_data.py` file. The class

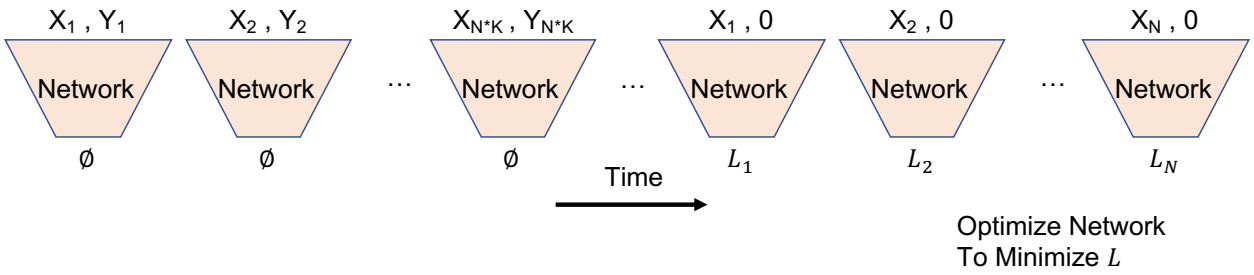


Figure 1: Feed $K$ labeled examples of each of $N$ classes through network with memory. Then feed final set of $N$ examples and optimize to minimize loss.

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

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.

## 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 $K$ examples of unseen classes 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 batch. Each set of labels and images are concatenated together, and then all $K$ of these concatenated pairs are sequentially passed through the network. Then the final 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 $N$ classes.

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` which takes as input the $[B, K+1, N, N]$ labels and $[B, K+1, N, N]$ and computes the cross entropy loss.

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

## Problem 3: Analysis

Once you have completed problems 1 and 2, you can train your few shot classification model.

For example run `python hw1.py --num_classes=2 --num_samples=1 --meta_batch_size=4` to run 1-shot, 2-way classification with a batch size of 4. 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. Note your observations.

## Problem 4: Experimentation

a Experiment with one parameter of the model 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 success rate 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?

## 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.