CS 639 Assignment 1: Neural Network Fundamentals

Due 11:59PM CT on February 24th, 2026 on Gradescope
In this assignment, you will be implementing a classic feed-forward neural network from scratch. You will then perform a variety of experiments by varying the training dataset and hyperparameters, creating plots and writing analyses of what you observe.

You are expected to submit the following:

- Your codebase as a .zip directory. Your codebase can be structured however you choose, however please include a script called "hw1.py" in the base level of this directory that runs all of the assignment's experiments and produces all requested output files, such as plots.
- A pdf document containing your answers to each question in the written portion.

Implementation (40%)

Implement a feed-forward neural network from scratch. Your model should have one hidden layer and be capable of being initialized with any number of input, hidden and output units/vertices. The input and hidden layers should always both include a bias unit (how you implement the bias is your choice). Prior to training, set `np.random.seed(0)` and then initialize the model weights from a uniform random distribution U(-0.01, 0.01). Use ReLU activation in the hidden layer and use a linear output layer (no activation function). When implementing softmax, ensure numerical stability by subtracting the maximum logit before exponentiation. Links to the training datasets you are expected to use are in the Written Portion.

Permitted libraries include:

- Any built-in Python library (e.g., os, datetime, json)
- pandas
- numpy
- matplotlib
- struct (for reading MNIST, see below)

Note that you are *not* allowed to use any premade deep learning frameworks including (but not limited to) pytorch or tensorflow. Use of these libraries will not be accepted as a completed assignment; please ask course staff if you are in doubt as to whether a particular library is acceptable.

For the training algorithm, use mini-batch SGD with batch size of 32, and re-shuffle training data at the start of each epoch. Make sure you have set `np.random.seed(0)` to obtain consistent results. Do not use regularization, momentum, or adaptive optimizers: use plain mini-batch SGD only.

Your implementation is worth 40% of the assignment and will be manually reviewed for completeness, as well as good programming practices. One crucial practice will be vectorized matrix operations, with others including choosing meaningful variable names, structuring reused code into functions or classes, and commenting your code to explain any tricky or important logic.

Written Portion (60%)

1. Download the Iris dataset:
   https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data
   Randomly place 80% of the datapoints into a training dataset and the remaining 20% into a test dataset (80/20 split), representing labels as three-element one-hot vectors. Set `np.random.seed(0)` prior to performing the random split.
   a. Using a network with 5 hidden units, create a plot of average loss across all training samples (e.g., cross-entropy after you perform softmax) per-epoch, over 10 training epochs. Include four lines on your plot, one for each learning rate of 1, 1e-2, 1e-3 and 1e-8. Do not vary any other hyperparameters. Make sure your plot includes a title, x/y axis labels, and a legend.
   b. According to this plot, which LR seems to produce the best model? Explain your reasoning.
   c. Give the average loss across all test set datapoints for each of these four models (make sure not to perform any further model parameter updates!). According to test set loss, which LR seems to produce the best model? Explain your reasoning and compare these findings to the findings produced by the training set losses.
   d. Using an LR of 1e-2, create a plot of average loss by training epoch over 10 epochs. Include four lines on your plot, one for each hidden layer size of 2, 8, 16 and 32 units. Do not vary any other hyperparameters. Make sure your plot includes a title, x/y axis labels, and a legend.
   e. Give the average test loss for these four models, as well as the models' accuracy on the test set in predicting the species of Iris (accuracy is defined as the fraction of predictions where argmax(output) equals the true label). Describe what patterns you notice in performance across these model sizes in both the training and the test losses.
2. Repeat subparts a-e again for an 80/20 split of the California housing dataset:
   raw.githubusercontent.com/ageron/handson-ml2/master/datasets/housing/housing.csv
   Use linear output with mean squared error (MSE) loss and "median_house_value" as the target variable. In part e, provide MSE instead of accuracy. Standardize input features to zero mean and variance of 1 before training to ensure stable gradients.
3. Repeat subparts a-e again for the MNIST dataset. Download the train/test images and labels from: https://github.com/cvdfoundation/mnist
   Because MNIST datapoints are images (represented by 28×28 matrices), we will need to convert each datapoint into a $28 \times 28 = 784$-element vector. To do so, use the MNIST starter code provided at the end of this assignment. This is a classification

dataset, so use cross-entropy and represent labels as 10-element one-hot vectors. Note: we will use the official 60,000 / 10,000 train/test split. Do not create a custom 80/20 split for MNIST. You may train on a subset of 20,000 MNIST training examples if runtime becomes an issue–just make sure to note it in your Written submission.

4. Compare the performance of the network across datasets. Does any of the datasets seem easier or harder for the model to learn than the others? Use your plots and other results as evidence. Why do you think this might be, and what steps would you consider taking to help the model perform better?
5. During implementation of the neural network, what are three bugs/errors you encountered? For each bug, paste a few lines of output/error printout showcasing the issue, describe the cause and how you went about debugging the issue.

Code for reading in MNIST

```
import numpy as np
import struct

def load_images(filename):
    with open(filename, 'rb') as f:
        magic, num_images, rows, cols = struct.unpack(">IIII", f.read(16))
        data = np.frombuffer(f.read(), dtype=np.uint8)
        images = data.reshape(num_images, rows, cols)
    return images

def load_labels(filename):
    with open(filename, 'rb') as f:
        magic, num_labels = struct.unpack(">II", f.read(8))
        labels = np.frombuffer(f.read(), dtype=np.uint8)
    return labels

X_train = load_images("train-images.idx3-ubyte")
y_train = load_labels("train-labels.idx1-ubyte")
X_test  = load_images("t10k-images.idx3-ubyte")
y_test  = load_labels("t10k-labels.idx1-ubyte")

# Flatten
X_train = X_train.reshape(X_train.shape[0], -1)
X_test  = X_test.reshape(X_test.shape[0], -1)

# Normalize to [0,1]
X_train = X_train.astype(float) / 255.0
X_test  = X_test.astype(float) / 255.0
```