CSCI-561 - Fall 2021 - Foundations of Artificial Intelligence
Homework 3

Due April 13, 2022, 23:59:59 PDT

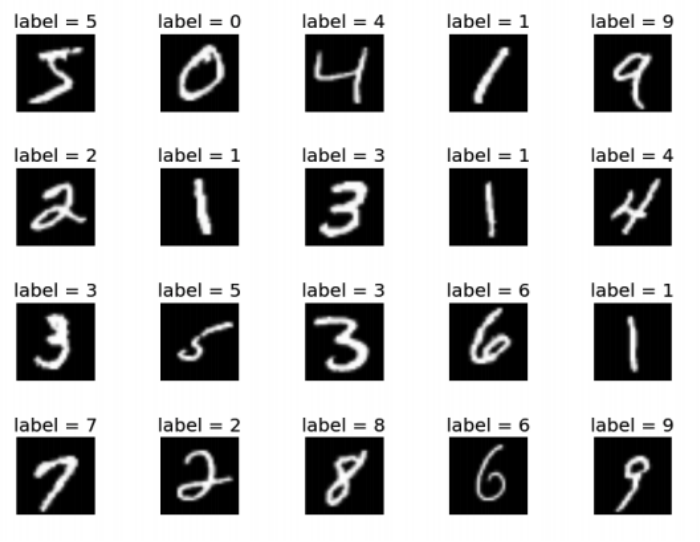


Figure 1: Hand-written digits images.

1. Overview

In this programming homework, you will implement a multi-layer perceptron (MLP) neural network and use it to classify hand-written digits shown in Figure 1. You can use numerical libraries such as Numpy/Scipy, but machine learning libraries are **NOT** allowed (including TensorFlow (v1&v2), Caffe, PyTorch, Torch, mxnet, etc.). You need to implement the feed-forward & back-propagation algorithms as well as training processes by yourselves.

2. Data Description

In this assignment, you will use the MNIST dataset (http://yann.lecun.com/exdb/mnist/). You can read its description from the URL above. This dataset consists of four files:

1. The images of the training set: 60k 28 × 28 grayscale training images, each representing a single handwritten digit.
2. The labels of the training set: the associated 60k labels for the training images.
3. The images of the test set: 10k 28 × 28 grayscale testing images, each representing a single handwritten digit.
4. The labels of the test set: the associated 10k labels for the testing images.

File 1 and 2 are the training set. File 3 and 4 are the test set. Each training and test instance in the MNIST database consists of a 28 × 28 grayscale image of a handwritten digit and an associated integer label indicating the digit that this image represents (0-9). Each of the 28 × 28 = 784 pixels of each of these images is represented by a single 8-bit color channel. Thus, the values each pixel can take on range from 0 (completely black) to 255 (2^8 − 1, completely white). The raw MNIST format is described in http://yann.lecun.com/exdb/mnist/.

For your convenience, we will use the .csv version of the dataset for submission and grading. To access it, please download `mnist.pkl.gz` and `mnist_csv3.py` from `HW3->resource->asnlib->public` to your local machine and run the following command:

```
python3 mnist_csv3.py
```

After that, you should be able to see File 1, 2, 3, 4 listed below:

- 1) `train_image.csv`
- 2) `train_label.csv`
- 3) `test_image.csv`
- 4) `test_label.csv`

The format of our CSV files will be described in **Section 3 Task description** below.

You can train and test your networks locally with the whole or partial dataset. When you submit, we provide a subset of MNIST for your training/testing (not for grading). We reserve the grading training/testing set (but it must be a subset of MNIST).

As an option, note that File 1 and 3 could be combined into File1+3, and File 2 and 4 can be combined into File2+4 (with the same index as File1+3). Viewed this way, the whole data will be contained in these two files: File1+3 contains all the images, and File2+4 contains all the labels of the images. One advantage of this is that one could partition the whole data into training and testing sets any way that is desired. You may easily modify `mnist_csv3.py` or simply merge the .csv files to achieve this option.

3. Task description

Your task is to implement a multi-hidden-layer neural network learner (see model description part for details of the neural network you need to implement), that will

(1) Construct a neural network classifier via training on the given labeled training data,
(2) Use the learned classifier to classify the unlabeled test data, and
(3) Output the predictions of your classifier on the test data into a file in the ***same*** directory**,**
(4) ***Finish in 30 minutes (for both training your model and making predictions)***.

Your program will take three input files and produce one output file as follows:

```
run your_program train_image.csv train_label.csv test_image.csv
⇒ test_predictions.csv
```

For example,

```
python3 NeuralNetwork.py train_image.csv train_label.csv test_image.csv
⇒ test_predictions.csv
```

In other words, your algorithm file `NeuralNetwork.***` will take training data, training labels, and testing data as inputs, and output your classification predictions on the testing data as output. In your implementation, ***please do not use any existing machine learning library call***. You must implement the algorithm yourself. Please develop your code yourself and do not copy from other students or the Internet.

The format of `***_image.csv` looks like:

a1, a2, a3, …… a784

b1, b2, b3, …… b784
……

Where x1, x2, and x3 are the pixels, so each row is an image. Each file contains at least one image and at most 60000 images.

The `train_label.csv` and your output `test_predictions.csv` will look like

1
0
2
5
…

(A single column indicates the predicted class labels for each unlabeled sample in the input test file.)

The format of your `test_predictions.csv` file is crucial. It has to be in the ***same name and format*** so that it can be parsed correctly to compare with true labels by the AI auto-grading scripts automatically.

When we grade your algorithm, we will use ***hidden*** training data and ***hidden*** testing data (randomly picked from MNIST) instead of the public data that was given to you for development. Your code will be auto-graded for correctness. Please name your file correctly, or you will wreak havoc on the auto-grader.

The maximum running time to train and test a model is 30 minutes *(for both training and testing)*, so please make sure your program finishes in 30 minutes.

As listed in Section 6, we have two sets to train and evaluate your model. Your model would be trained and evaluated on those two sets independently. The running time limitation is applied for training and evaluating ***one*** of the sets (i.e., the training time for each should be shorter than 30 mins).

4. Model description

The basic structure model of a neural network in this homework assignment is as Figure 2 below. The figure shows a 2-hidden-layer neural network. The input layer is one-dimensional, you need to reshape input to 1-d by yourself. At each hidden layer, you need to use a **sigmoid activation function** (see references below). Since it is a multi-class classification problem, you need to use **the softmax function** (see references below) as activation at the final output layer to generate the probability distribution of each class. For computing loss, you need to use the **cross-entropy loss function**. (see references below) ***There is no specific requirement on the number of nodes in each layer***, you need to choose them to make your neural network reach the best performance. Also, the number of nodes in the input layer should be the number of features, and the number of nodes in the output layer should be the number of classes.

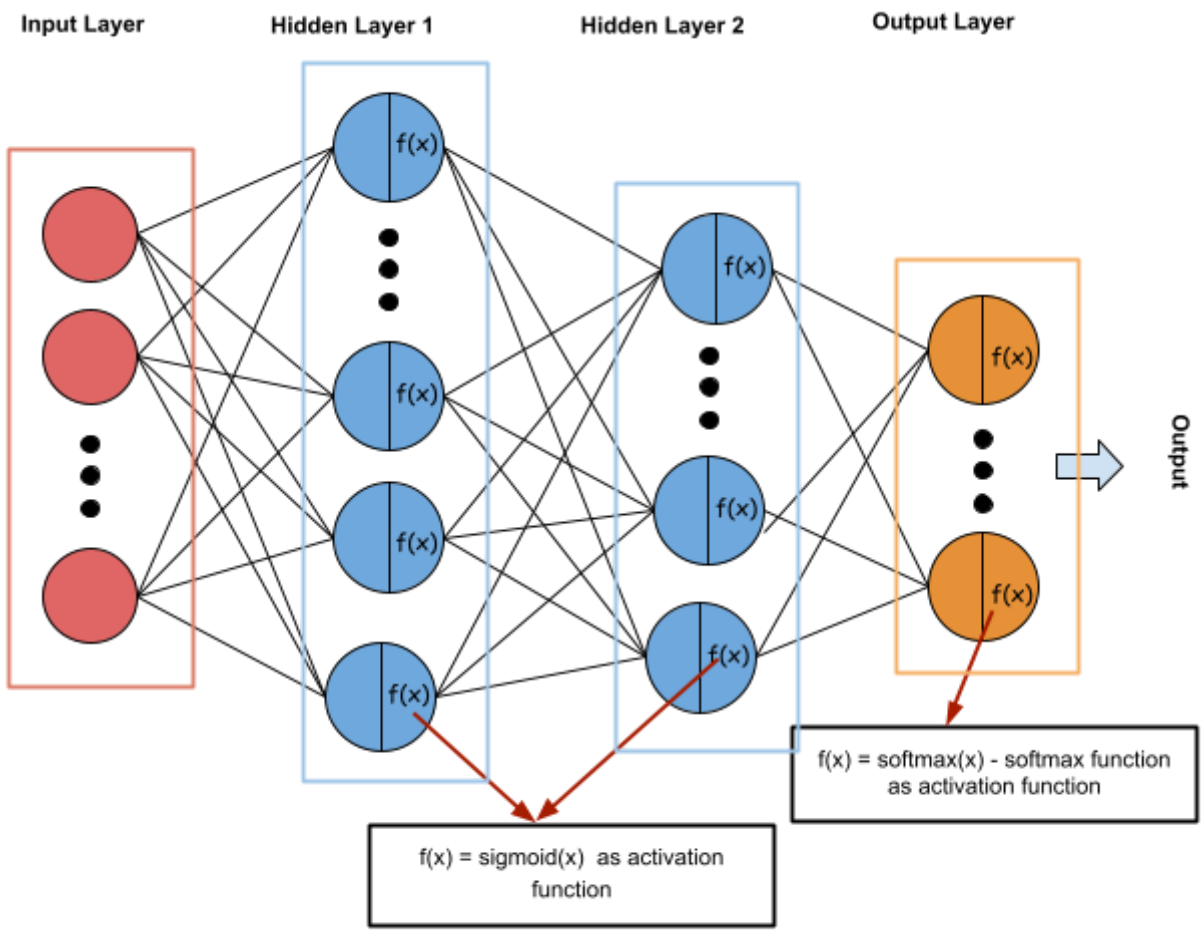


Figure 2: Example Network Configurations.

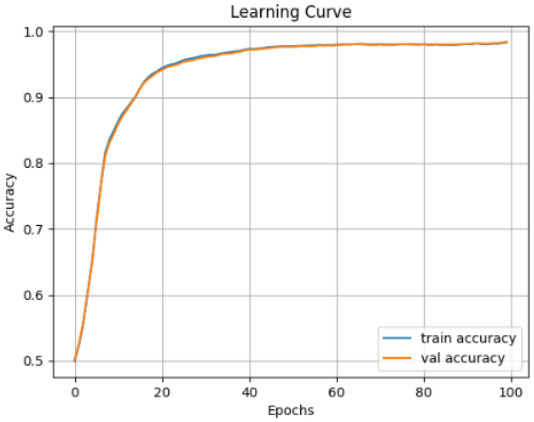
There are some hyper-parameters you need to tune to get better performance. You need to find the best hyper-parameters offline so that your neural network can get good performance on the given test data as well as on the hidden grading data.

- **Learning rate**: step size for update weights (*e.g. weights = weights - learning * grads*), different optimizers have different ways to use learning rate. (see reference in 2.1)
- **Batch size**: number of samples processed each time before the model is updated. The size of a batch must be more than or equal to one, and less than or equal to the number of samples in the training dataset. (*e.g suppose your dataset size is 1000, and your batch size is 100, then you have 10 batches, each time you train one batch (100 samples) and after 10 batches, it trains all samples in your dataset.*)
- **Number of the epochs**: the number of complete passes through the training dataset (*e.g. you have 1000 samples, 20 epochs mean you loop this 1000 samples 20 times, suppose your batch size is 100, so in each epoch, you train 1000/100 = 10 batches to loop the entire dataset and then you repeat this process 20 times*)
- **Number of units in each hidden layer**

Remember that the program has to **finish in 30 minutes**, so choose your hyper-parameters wisely.

Learning Curve Graph (we will not grade it but it may help)

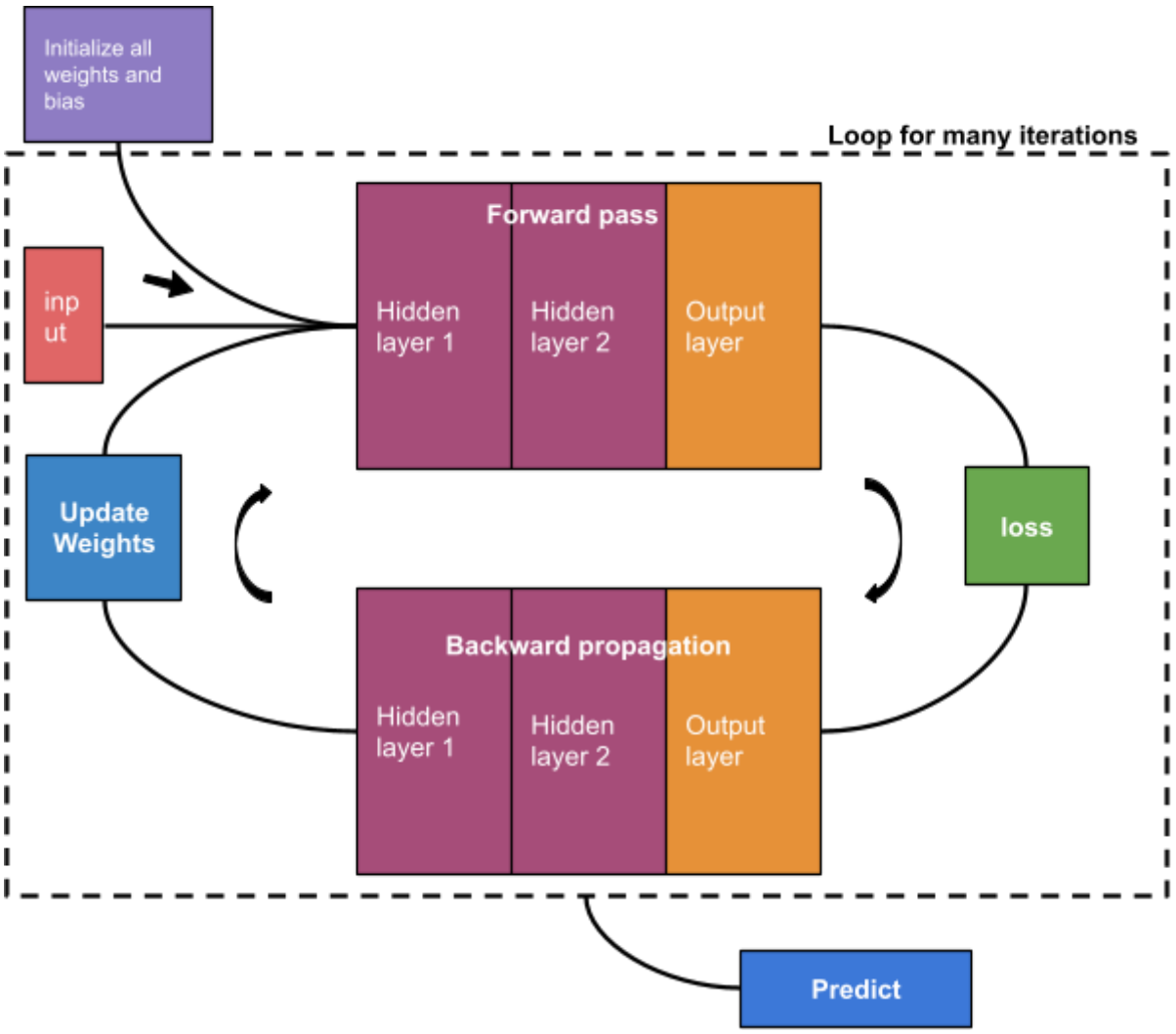
To make sure your neural network learns something, You may need to make a plot to show the learning process of your neural networks. After every epoch (one epoch means going through all the samples in your training data once), it may be a good idea to record your accuracy on the training set and the validation set (it is just the test set we give you) and make a plot of the accuracy as shown in the figure on the right.



5. Implementation Guidance

Suggested Steps

1. **Split the dataset into batches**
2. **Initialize weights and bias**
3. **Select one batch of data and calculate forward pass** - follow the basic structure of the neural network to compute the output for each layer, you might need to cache the output of each layer for the convenience of backward propagation.
4. **Compute loss function** - you need to use cross-entropy (logistic loss - see references above) as loss function
5. **Backward propagation** - use backward propagation (your implementation) to update hidden weights
6. **Updates weights using optimization algorithms** - there are many ways to update weights you can use plain SGD or advanced methods such as Momentum and Adam. (but you can get full credit easily without any advanced methods)
7. **Repeat 2,3,4,5,6 for all batches** - after finishing this process for all batches (it just iterates all data points of the dataset), it is called ‘one epoch’.
8. **Repeat 2,3,4,5,6,7 number of epochs times**- You might need to train many epochs to get a good result. As an option, you may want to print out the accuracy of your network at the end of each epoch.



Tips

Many techniques can speed up the training process of your neural networks. Feel free to use them. For example, we suggest using vectorization such as Numpy instead of for loop in python. You can also

1. Try advanced optimizers such as SGD with momentum or Adam.
2. Try other weights initialization methods such as Xavier initialization.
3. Try dropout or batch norm.

And so on, but you **DO NOT** need them to achieve our accuracy goal. A “vanilla” or naive implementation with a proper learning rate can work very well by itself.

DO NOT USE ANY existing machine learning library such as Tensorflow and Pytorch.

6. Submission and Grading

As described previously, we will provide 3 input files (`train_image.csv train_label.csv test_image.csv`) in your working path. Your program file should be named as `NeuralNetwork.***`. (if you are using python3 or C++11, name it as `NeuralNetwork3.py/NeuralNetwork11.cpp`) and output a file `test_predictions.csv`. You need to make sure the output file name is the same.

Grading would be conducted on two individual sets:

1. MNIST Set
    a. Training: ***10,000*** images that are randomly sampled from the MNIST dataset.
    b. Testing: 10,000 images that are randomly selected from the MNIST dataset. *There is no overlap between the training and testing images.*
2. TA Set:
    a. Training: ***10,000*** images that are randomly sampled from the MNIST dataset.
    b. Testing: some (around 100) hand-written images created by the 561 TA team.

Grading is based on your prediction accuracy ***on both testing sets***. We would evaluate your model on those two sets separately. The final grade would be a weighted average of the credits for both testing sets:

Final Grade = 0.6 * Credit(MNIST set) + 0.4 * Credit(TA set)

MNIST acc-to-score mapping:

[90, 100] → 100
[80, 90) → 75
[70, 80) → 60
[50, 70) → 50
[0, 50) → 0

TA-set acc-to-score mapping:

[55, 100] → 100
[50, 55) → 80
[40, 50) → 70
[35, 40) → 60
[30, 35) → 50
[0, 30) → 0

[A, B) means A<= x < B.

Notice: 90% and 55% are not a very difficult goal -- if your implementation is mostly correct, you will find little extra work is needed to achieve such an accuracy. In other words, if you cannot get close to the goal, there is a high possibility that your code has some problems.

Directly loading pre-trained weights of the Neural Network is prohibited.

As the TA set is newly created, we provide 10 examples below.
Note those samples are just for illustrative purposes.