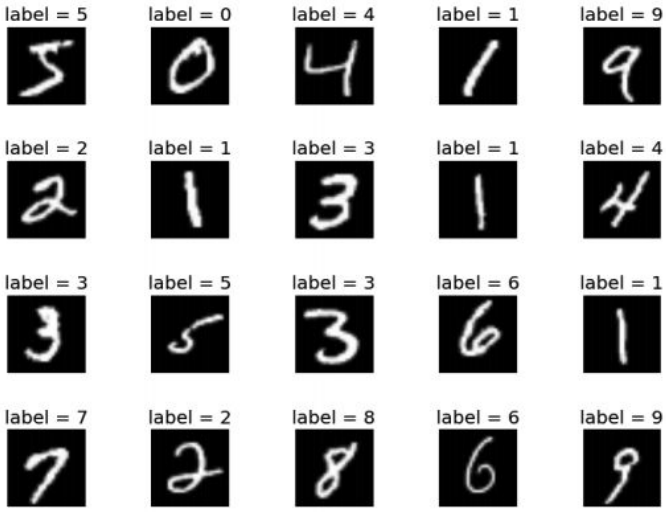
CSCI-561 - Spring 2020 - Foundations of Artificial Intelligence
Homework 3

Due April 15, 2020, 23:59:59



1. Overview

In this programming homework, you will implement a multi layer perceptron(MLP) neural network and use it to classify hand-written digits. You can use numerical libraries such as Numpy/Scipy, but machine learning libraries are **NOT** allowed. You need to implement feedforward/backpropagation as well as training process by yourselves.

2. Data Description

In this assignment you will use MNIST dataset(http://yann.lecun.com/exdb/mnist/), you can read its description and download it by yourselves. This dataset consists of four files:

1. train-images-idx3-ubyte.gz, which contains 60,000 28 × 28 grayscale training images, each representing a single handwritten digit.
2. train-labels-idx1-ubyte.gz, which contains the associated 60,000 labels for the training images.
3. t10k-images-idx3-ubyte.gz, which contains 10,000 28 × 28 grayscale testing images, each representing a single handwritten digit.
4. t10k-labels-idx1-ubyte.gz, which contains the associated 10,000 labels for the testing images.

Files 1 and 2 are the training set. Files 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 (28 − 1, completely white).

The raw MNIST format is described in http://yann.lecun.com/exdb/mnist/, ***but we will use a .csv version for submission and grading for your convenience.*** The format of our csv files will be described in the **Task description** part below, while we will also provide a simple code (under the path $ASNLIB/public on Vocareum) for transfering raw MNIST files to our csv format.

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

3. Task description

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

(1) Construct a neural network classifier from 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 from 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

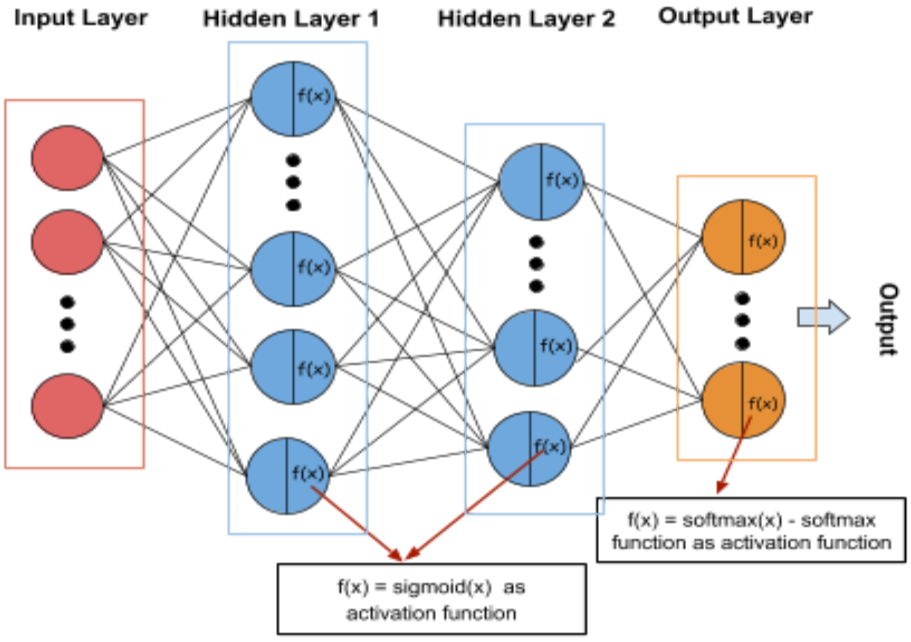
1
0
2
5
… (A single column indicates the predicted class labels for each unlabeled sample in the input test file)

The format of your `test_predictions.csv` file is crucial. It has to be in the ***exact 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 testing data that was given to you for submission. Your code will be autograded for technical correctness. Please name your file correctly, or you will wreak havoc on the autograder. **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.**

4. Model description

The basic structure model of a neural network in this homework assignment is as below.



The above 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 **softmax function** (see references below) as activation at the final output layer to generate probability distribution of each class. For computing loss, you need to use **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 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.

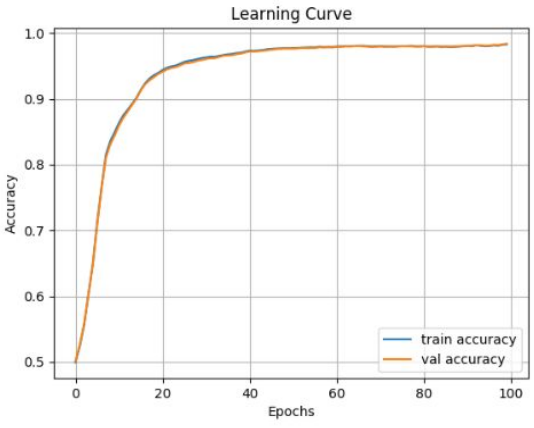
There are some hyper-parameters you need to tune to get better performance. You need to find best hyper-parameters so that your neural network can get good performance on the given test data as well as on the hidden grading data.

- **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 is of 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 epoch**: the number of complete passes through the training dataset (*e.g. you have 1000 samples, 20 epochs means 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)

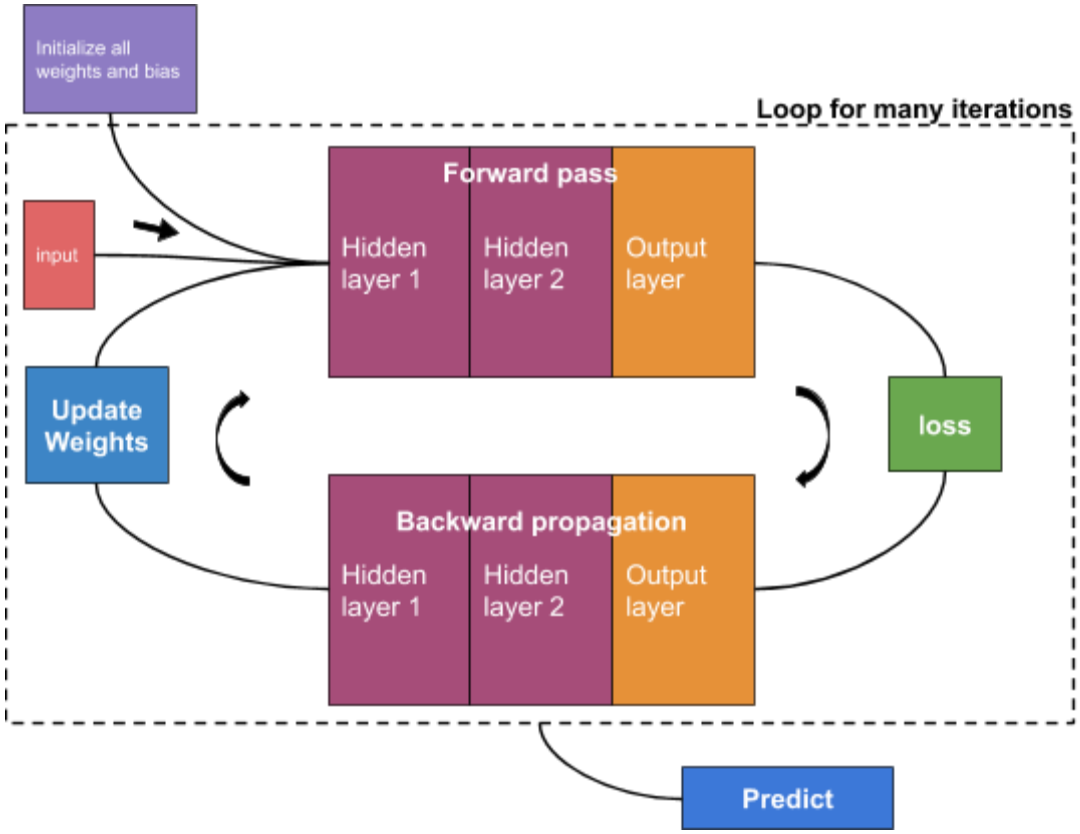
In order to make sure your neural network actually learns something, You may need to make a plot to show the learning process of your neural networks. After every epoch (one epoch means going through all the samples in your training data once), it may be a good idea to record your accuracy on the training set and the validation set (it is just the test set we give you) and make a plot of those 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 neural network to compute output for each layer, you might need to cache 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 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

There are many techniques that 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 optimizer such as SGD with momentum or Adam.
2. Try other weights initialization methods such as Xavier initialization.
3. Try dropout or batchnorm.

And so on, but you **DO NOT** really need them to achieve our accuracy goal. A “vanilla” or naive implementation with 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 C11 name it as NeuralNetwork3.py/NeuralNetwork11.cpp)` and run to output a file `test_predictions.csv`. You need to take care of the file names so that they are correct.

Training/test dataset will be different in submission and grading, but they are subsets of MNIST.

Grading is based on your prediction accuracy. We hope you can get at least 90% accuracy, any result better than 90% will get all credit. Results between 50% and 90% will get 50% credit, but if your accuracy is less than 50% you will get nothing.
Notice: 90% is not a hard goal, if your implementation is correct, you will find little extra work is needed to achieve the accuracy. In other words, if you cannot get close to the goal, there is a high possibility that your code has some problems.

7. Academic Honesty and Integrity

All homework material is checked vigorously for dishonesty using several methods. All detected violations of academic honesty are forwarded to the Office of Student Judicial Affairs. To be safe, you are urged to err on the side of caution. Do not copy work from another student or off the web. Keep in mind that sanctions for dishonesty are reflected in *your permanent record* and can negatively impact your future success. As a general guide:

Do not copy code or written material from another student. Even single lines of code should not be copied.
Do not collaborate on this assignment. The assignment is to be solved individually.
Do not copy code off the web. This is easier to detect than you may think.
Do not share any custom test cases you may create to check your program's behavior in more complex scenarios than the simplistic ones that are given.
Do not copy code from past students. We keep copies of past work to check for this. Even though this project differs from those of previous years, do not try to copy from homeworks of previous years.
Do not ask on Piazza how to implement some function for this homework, or how to calculate something needed for this homework.
Do not post code on Piazza asking whether or not it is correct. This is a violation of academic integrity because it biases other students who may read your post.
Do not post test cases on Piazza asking for what the correct solution should be.
Do ask the professor or TAs if you are unsure about whether certain actions constitute dishonesty. It is better to be safe than sorry.

DO NOT USE ANY existing machine learning library such as Tensorflow and Pytorch. Violation may cause a penalty to your credit.