LAHORE UNIVERSITY OF MANAGEMENT SCIENCES
Syed Babar Ali School of Science and Engineering

EE514/CS535 Machine Learning
Spring Semester 2021

Programming Assignment 5 – Neural Network

Issued: Sunday, 18 April, 2021

Total Marks: 100

Submission: 11:55 pm, Sunday, 25 April, 2021.

Goal

The aim of this assignment is to give you an initial hands-on experience regarding neural networks and how they can be used to make a handwritten digits detection system.

Instructions

- Submit your code BOTH as notebook file (.ipynb) and python script (.py) on LMS. The name of both files should be your roll number. Failing to submit any one of them will result in the deduction of marks.
- You have been provided with the starter code. All tasks should be implemented in that file, in the order provided.
- The code MUST be implemented independently. Any plagiarism or cheating of work from others or the internet will be immediately referred to the DC.
- Late submissions: 10% penalty per day for 3 days after due date.
- **You are strongly advised to use Google Colab for this assignment.**

Loading Dataset

You will be using MNIST handwritten digits to create a digit detection system. The dataset contain binary images of handwritten digits, and has has a training set of 60,000 examples, and a test set of 10,000 examples. The dataset can be loaded using the following commands:

```
from keras.datasets import mnist
(trainX, trainY), (testX, testY) = mnist.load_data()
```

You are required to:

- Normalize all pixel values between 0 and 1.
- One hot encode true labels using the to_categorical() function.

Part 1 (70 Marks)

In the first part, you will be creating a neural network from scratch. Often the code is hidden behind libraries. The purpose of this task is to give you hand-on experience with the mathematical foundations of neural network architectures. After implementing this task, you will know exactly how forward and backward pass functions are implemented mathematically and in code. To help you get started, we have provided you with a skeleton code containing the Neural Network class. You are allowed to change the skeleton code. However, it is necessary to perform the following tasks:

- Reshape the dataset to a 2D array using NumPy reshape function.
- Initialize your class. The class can have arbitrary number of hidden layers and nodes depending on the parameters passed. For example, the class instance declared as:

  $$\text{nn} = \text{NeuralNetwork}([784, 20, 20, 10])$$

  contains an input layer with 784 nodes, 2 hidden layer with 20 nodes each and an output layer with 10 nodes.
- *Cross-entropy loss function.* Suppose you have your predictions as $\hat{\boldsymbol{y}} = [\hat{y}_0, \hat{y}_1, \ldots, \hat{y}_{M-1}]^T$, where $\hat{y}_m$ is the probability of that data point belonging to class m. To access the cross entropy loss of a single training instance, we can implement the cross entropy function as shown.

  $$L(\hat{\boldsymbol{y}}, y) = -\log(\hat{y}_m), \text{ (where } m \text{ is the index of the label } y \text{ of the correct class)}$$

  To write this more compactly, we use the discrete delta function $\delta(t)$ defined as:

  $$\delta(t) = \begin{cases} 1 & t = 0 \\ 0 & \text{otherwise} \end{cases}$$

  We will use our $\delta(t)$ function to define: $\delta(y - m)$, which will be one in the case that $y = m$, i.e., y belongs to class m, and zero otherwise. Then, we can rewrite our loss function as:

  $$L(\hat{\boldsymbol{y}}, y) = -\sum_{m=0}^{M-1} \delta(y - m)\log(\hat{y}_m),$$

  We can extend this function to the case with n instances, and we can replace our prediction vector $\hat{\boldsymbol{y}} = h_{\boldsymbol{\theta}_m}(\boldsymbol{x}_i)$, where $\boldsymbol{x}_i$ is the feature vector of the i-th index, the resulting final expression is given below:

  $$L(\boldsymbol{\theta}) = -\sum_{i=1}^{n}\sum_{m=0}^{M-1} \delta(y_i - m)\log(h_{\boldsymbol{\theta}_m}(\boldsymbol{x}_i)),$$

  where y_i is the label of the i-th instance.
- *Softmax function* is to be used as the activation function in the output layer. The formula of the softmax function is as follows:

  $$\text{softmax}(z_m) = \frac{e^{z_m}}{\sum_{k=0}^{M-1} e^{z_k}},$$

If we take z_m as $\boldsymbol{\theta}_m^T \boldsymbol{x}$, where $\boldsymbol{\theta}_m$ is the weight vector. Then the softmax function gives us the probability of the feature vector $\boldsymbol{x}$ belonging to that class:

$$P(y = m|\boldsymbol{x}) = h_{\boldsymbol{\theta}_m}(\boldsymbol{x}) = \frac{e^{\boldsymbol{\theta}_m^T \boldsymbol{x}}}{\sum_{k=0}^{M-1} e^{\boldsymbol{\theta}_k^T \boldsymbol{x}}},$$

- *Sigmoid function* is to be used as the activation function in every hidden layer. The formula of the sigmoid function is as follows:

$$\sigma(z) = \frac{1}{1 + e^{-z}},$$

- *_Init_* is the constructor of our neural network class which is called automatically when our class is initialized. You are required to store important information of your network architecture such as number of nodes in every layer.
- *Initialize weights and biases function* is called by the constructor. You are required to initialize all weights and biases of your neural network in this function. Initialize all weights based on normal standard distribution and all biases to 0.
- *Forward Pass function* traverses the neural network and calculate the output of every layer. The output of a layer is calculated using the following equations:

$$\boldsymbol{a}^{[l]} = g(\boldsymbol{z}^{[l]}),$$
$$\boldsymbol{z}^{[l]} = \boldsymbol{W}^{[l]} \boldsymbol{a}^{[l-1]} + \boldsymbol{b}^{[l]}, \quad \boldsymbol{a}^{[0]} = \boldsymbol{x},$$

  where $g(\boldsymbol{z}^{[l]})$ is the activation function (operating on each entry of the vector) and the output of the l-th layer. You are required to use sigmoid as an activation function for every hidden layer and softmax for the output layer. The function should return a list containing outputs of every layer. You may refer to the lecture notes for the notation adopted here. These equations provide output for a single input.

- *Backward Pass function* will traverse the neural network starting from the output layer. You are required to to compute partial derivatives of the cost function with respect to the weights and biases of every layer. The partial derivatives are calculated using the following equations:

$$\frac{\partial L}{\partial \mathbf{z}^{[l]}} = (\mathbf{W}^{[l+1]^T} \frac{\partial L}{\partial \mathbf{z}^{[l+1]}}) \odot g'(\mathbf{z}^{[l]})$$
$$\frac{\partial L}{\partial \boldsymbol{b}^{[l]}} = \frac{\partial L}{\partial \boldsymbol{z}^{[l]}},$$
$$\frac{\partial L}{\partial \boldsymbol{W}^{[l]}} = \frac{\partial L}{\partial \boldsymbol{z}^{[l]}} \boldsymbol{a}^{[l-1]^T},$$

  The function should return a list containing partial derivatives of every layer.

- *Update* function will traverse all layers and update their weights and biases using the following equation:

$$\mathbf{W}^{[l]} = \mathbf{W}^{[l]} - \alpha \frac{\partial L}{\partial \mathbf{W}^{[l]}},$$
$$\mathbf{b}^{[l]} = \mathbf{b}^{[l]} - \alpha \frac{\partial L}{\partial \mathbf{b}^{[l]}},$$

  where α is the learning rate, $\boldsymbol{W}^{[l]}$ and $\boldsymbol{b}^{[l]}$ are weights and biases of the l-th layer.

- *Fit function* skeleton code is already provided to you. Parameters passed in this function are training data, learning rate and number of iterations (epochs). The function uses stochastic gradient descent to train the neural network . Call $Forward\_Pass()$, $Backward\_Pass()$, and the $Update()$ function for every training sample. The fit function should return lists of training loss, validation loss, training accuracy, and validation accuracy for every iteration. After you initialize an instance of your class, call the fit function.
- Plot the following graphs:
  - Training loss (y-axis) vs no. of epochs (x-axis).
  - Validation loss (y-axis) vs no. of epochs (x-axis).
  - Training accuracy (y-axis) vs no. of epochs (x-axis).
  - Validation accuracy (y-axis) vs no. of epochs (x-axis).
- Predict the labels of test data using the *predict function*. Skeleton code for this function is already given to you. You may find np.argmax useful to extract labels.
- *Evaluate function* that calculates classification accuracy, F1 score and confusion matrix. Pass the labels of test data in this function and report your results.

Part 2 (30 Marks)

In this part, you will be creating a convolution neural network for image classification on the same dataset. However in this part, you will be using the TensorFlow and Keras API to create model for classification. The purpose of this task is to give you hands-on experience in using TensorFlow API to create deep learning models. To help you get started, skeleton code has been provided. However you may change the skeleton code if necessary. Below mentioned are the tasks which have to be done.

- Reshape the images so that each image is of the size (28,28,1) to have a single color channel.
- Update the *hyperparameters* which in this case are *batch size*, *learning rate* and *epochs*.
- *Define model*: define the model using tensorflow API. Read the documentation here. A sample model has been attached in the notebook, you may choose to implement the same model architecture or develop your own. Create the model and print its summary.
- *Compile your model*. Read the documentation here to understand how to compile models. To understand how to use optimizers, refer to this link.
- *Fit your model*. Read the documentation here to understand how to use model.fit(). Use optimizer = Adam, loss = categorical_crossentropy and metrics = accuracy Hint: While defining steps_per_epoch, define it as total_images/batch_size. Also divide your train data into train and validation set.
- Plot the following graphs:
  - Training loss (y-axis) vs no. of epochs (x-axis).
  - Validation loss (y-axis) vs no. of epochs (x-axis).
  - Training accuracy (y-axis) vs no. of epochs (x-axis).
  - Validation accuracy (y-axis) vs no. of epochs (x-axis).

  Use hist.history['loss'], hist.history['val_loss'], hist.history['accuracy'], hist.history['val_accuracy'] where $hist = model.fit()$.
- Make predictions on test data using model.predict(). Read the documentation here to understand how to use model.predict(). You may find np.argmax useful to extract labels.
- *Evaluate function* that calculates classification accuracy, F1 score and confusion matrix. Pass the labels of test data in this function and report your results.