Shallow Neural Network for Classification

Reetu Hooda
University of Alabama, Huntsville
rh0059@uah.edu

I. Introduction

This report presents an implementation of an API-like python class for a fully connected neural network for multi-class classification using numpy. We achieve classification of MNIST handwritten dataset and Madison County dataset using back-propagation algorithm. The plot of loss and classification accuracy during training determines the performance of the neural network model. The experimental results show that reasonably high accuracy can be achieved using a fully connected shallow neural network for binary/multi-class classification problems.

II. Implementation

A. Network Architecture

A shallow neural network commonly consist of an input layer, hidden layer and an output layer as shown in Fig. 1.

- No. of input neurons n_x = No. of input features
- No of output neurons n_y = No. of classes

whereas, No of hidden neurons n_h is a variable which can be regulated. Therefore, the training set $X_{n_x \times m}$ is fed to the network to get $Y_{n_y \times m}$ where, m are the no of training examples.

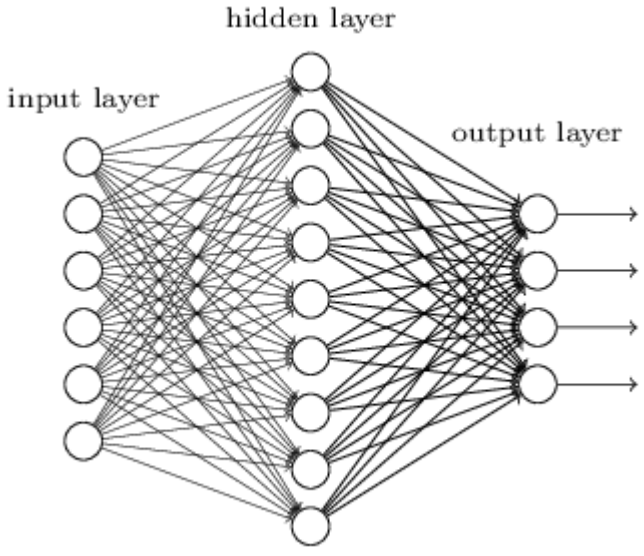

Fig. 1. Shallow neural network

B. Weight Initialization

The algorithm uses random initialization to assign values to the weights.

- $W^{[1]}_{n_h \times (n_x+1)}$: connects input layer and hidden layer.
- $W^{[2]}_{n_y \times (n_h+1)}$: connects hidden layer and output layer.

C. Activation Functions

The class allows users to choose one of the following activation functions for hidden layer:

1) Sigmoid Function

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

$$\sigma'(z) = \sigma(z)(1-\sigma(z))$$

2) ReLU

$$f(z) = max(0, z)$$

$$f'(z) \begin{cases} 1, & \text{if } z > 0. \\ 0, & \text{if } z < 0. \end{cases}$$

3) Swish

$$f(z) = \frac{\mathrm{z}}{1+e^{-z}}$$

$$f'(z) = f(z) + \sigma(z)(1-f(z))$$

We have used softmax activation function for the output layer which is defined as follows:

$$softmax(z_j) = \frac{\mathrm{e}^{\mathrm{z_j}}}{\sum_i e^{z_i}}$$

D. Loss Function

Since we are using softmax activation for output layer while developing the class, we define cross entropy loss as:

$$E = -\sum_{i}^{nclass} y_i \log(\hat{y}_i)$$

where y_i are the target values encoded as one-hot vectors and y_i are the predicitons made by our model.

Dataset	n_h	Hidden layer activation	α	Iterations	Training Accuracy	Testing Accuracy
MNIST (0-9)	100	Swish	0.1	80	99.59 %	97.50 %
Madison County	20	ReLU	0.005	10,000	91.97 %	88.96 %

TABLE I
SUMMARY OF RESULTS

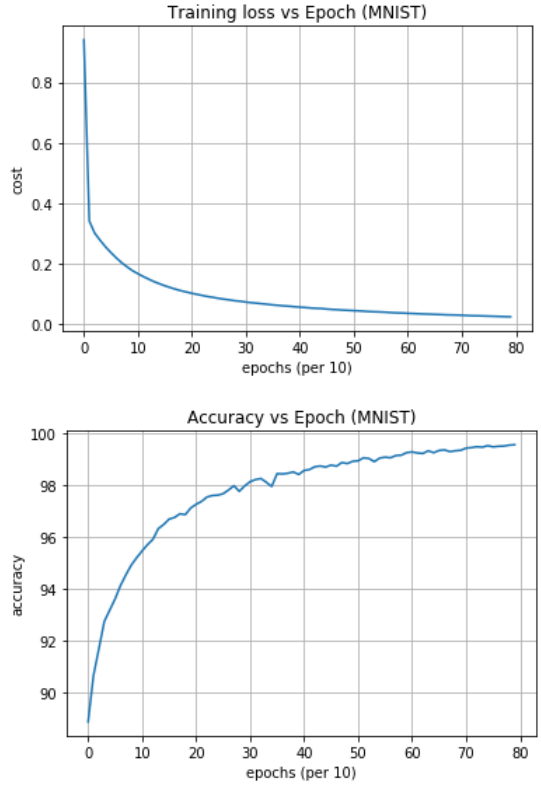

Fig. 2. Loss and Accuracy Vs Epoch

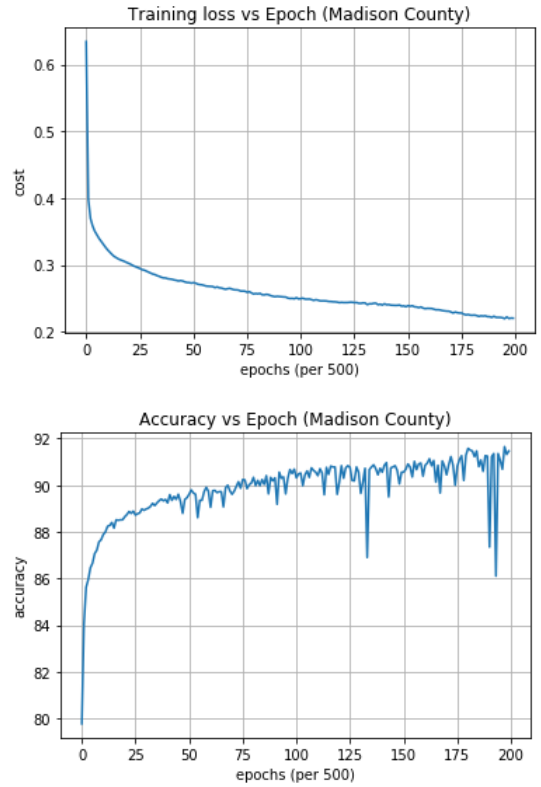

Fig. 3. Loss and Accuracy Vs Epoch

E. Forward Propagation

The forward propagation used can be summarized as follows:

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$A^{[1]} = f(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = \hat{y} = softmax(Z^{[2]})$$

F. Backward Propagation using Gradient Descent

We calculate error at each layer starting from output layer and back-propagate using gradient descent algorithm. The backward propagation for 2-layer NN can be summarized as follows:

$$dZ^{[2]} = A^{[2]} - y$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} (A^{[1]})^T$$

$$dZ^{[1]} = (W^{[2]})^T dZ^{[2]} * f'(Z^{[1]})$$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} (X)^T$$

G. Parameter Update

The parameters are updated as follows:

$$W^{[1]} = W^{[1]} - \alpha \times dW^{[1]}$$

$$W^{[2]} = W^{[2]} - \alpha \times dW^{[2]}$$

where α is the learning rate.

III. Results

A. Datsets

1) MNIST (2-d) : It is a database of handwritten digits that has 60,000 training examples and 10,000 testing examples. Each image is 28×28 $(n_x = 784 \times 1)$ pixels image classified into 10 $(n_y = 10 \times 1)$ classes (0-9). It is a multi-class classification problem.

   Normalization: We have used min-max normalization for MNIST dataset i.e.,

$$I(x, y) = \frac{I(x, y) - min(I(x, y))}{max(I(x, y)) - min(I(x, y))}$$

2) Madison County(1-d): It is binary classification problem that has negative and positive data points and each data point has $n_x = 24$ features. The missing values in the dataset are taken as 0 while training.

   Normalization: We have used Z-normalization for Madison county dataset i.e.,

$$X = \frac{X - \mu(X)}{\sigma(X)}$$

   And the labels $y's$ are one-hot encoded (Y) of both datasets.

B. Training Loss and Accuracy

Training and accuracy plots are shown in Fig. 2 and Fig. 3

IV. Conclusion

An API-like python class was designed using numpy for a 2-layer neural network. It is shown to provide good accuracy for two datasets.