PROJECT 2 MULTI-CLASS CLASSIFICATION USING NEURAL NETWORK

Prakhathi Murugesan
University of Buffalo
prakhath@buffalo.edu

Abstract

Neural Network was modeled after human brain, which are designed to recognize patterns and cluster the data. The goal of this project is to implement neural network and convolutional neural network for classification. we use Fashion-MNIST clothing images as a dataset. We build models which train on the data and will be able to identify the image into one of the ten classes. Evaluation of each model is done by its accuracy, confusion matrix and observations are made on the result. The models developed are single layer neural network, multilayer neural network, and convolution neural network. The multilayer neural network and convolution neural network is done using keras, open-source neural-network library. The result obtained from each model for the same dataset tells us which model's performance is better and efficient. We found out that the Convolutional Neural Network gives the better performance with 88% accuracy and loss of about 0.34.

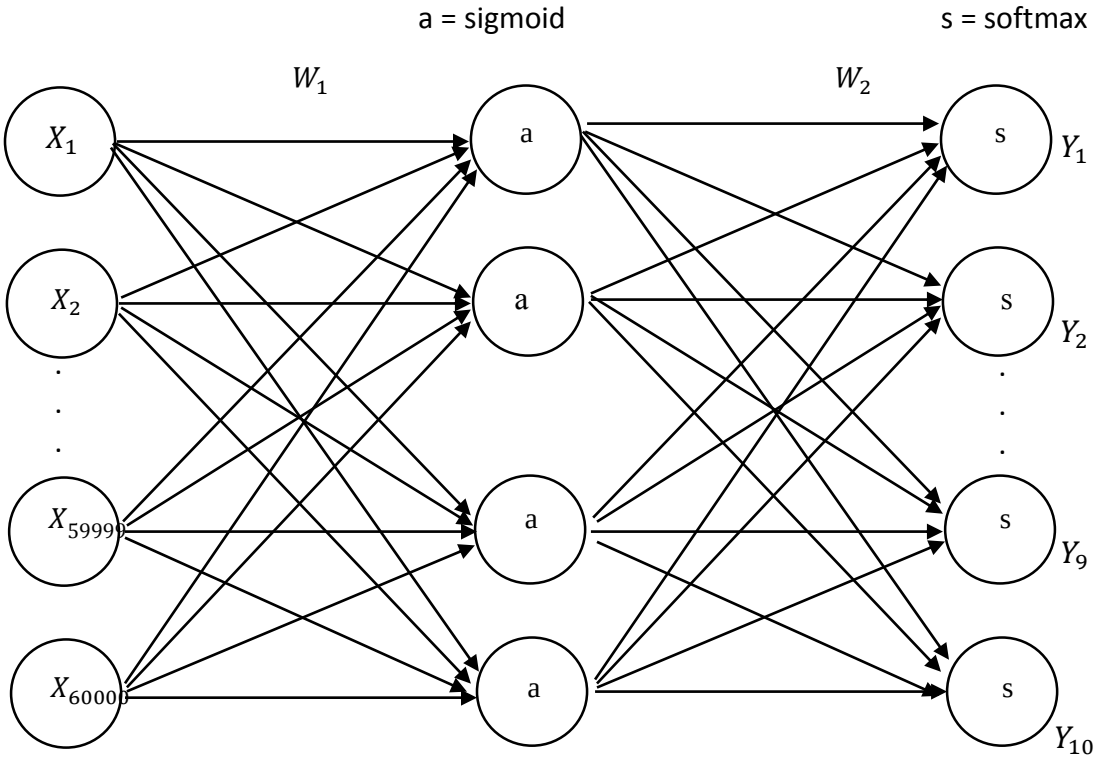
1.INTRODUCTION

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. Neural networks can adapt to changing input, so the network generates the best possible result without needing to redesign the output criteria. A neural network contains layers of interconnected nodes. Each node is a perceptron and is similar to multiple linear regression. The perceptron feeds the signal produced by a multiple linear regression into an activation function that may be non-linear. Models are developed to group the data and classify them into different classes, when they are trained with training dataset which has feature sets. The Model trains on the feature dataset. It classifies each data into one of the classes and predicts the output. The dataset used for this project is fashion-MNIST which has 60,000 instances of training data and 10,000 test data. The dataset consists of 28*28 grayscale image, associated with a label from 10 classes. Each image is of 28 pixels in height and 28 pixels in width, and a total of 784 pixels. This pixel value ranges from 0 and 255. The dataset consists of 784 columns denoting feature or attribute. Each image in dataset is assigned to one of the 10 classes or labels. The labels are T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag and Ankle Boot which are taken as class 0 to 9. The goal is to build the various model and evaluate which gives better performance.

2. PRE-PROCESSING OF DATASET:

The first step of Pre-processing is to convert the feature variable to desired format. Given dataset of images, each of depth 1. But the dataset has shape of (n, width, height), we want to transform the shape of our dataset to (n, width, height, depth). The next preprocessing step for the input data is to convert our data type to float32 and normalize our data values to the range [0, 1]. Given feature dataset consists of images which are 28 * 28 pixels. Pixels are represented in the range [0-255], but the neural network converges faster with smaller values, in the range [0-1]. Therefore, the given dataset is normalized to this range. Many Machine learning algorithms cannot work with categorical data directly. Given target variable, y_train, y_test, we need to convert it to 'one-hot-encode', which means a column will be created for each output category and a binary variable is inputted for each category. This converts 1-dimensional class arrays to 10-dimensional class matrices. The final step of pre-processing is to partition the dataset into validation sets.

3. ARCHITECTURE:



The model is developed with input layer, hidden layers and output layer. Input layer buffers the input feature and multiplies it with weight. Hidden layer gets the weighted input and activation function is invoked. The output of this layer is then added with weights and passed to output layer. Output layer applies softmax function to give the predicted output. The output of the final layer should be evaluated with target set and loss/cost is calculated. To reduce the loss value, the weights and bias has to be updated.

3.1. NEURAL NETWORK WITH ONE HIDDEN LAYER:

The entire operation is splitted into two steps:

- Forward Propagation
- Backward Propagation

Forward Propagation:

Forward Propagation is the process of feeding the input data in the forward direction through the network. Each hidden layer accepts the input data, processes it as per the activation function, and passes to the successive layer. The activation function used here is sigmoid. Sigmoid function is represented by,

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

The sum of weighted output from sigmoid is then added with bias and then passed to softmax. The softmax function is often used in the final layer of a neural network-based classifier. It normalizes an input value into a vector of values that follows a probability distribution whose sum equal to 1.

Softmax function is represented by,

$$y_i(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{k} e^{z_k}}$$

Steps involved in forward propagation,

Z1 = W1X + B1

A1 = σ (Z1)

Z2 = W2 (A1) + B2

A2 = softmax(Z2)

Backward Propagation:

Backward Propagation is the essence of neural net training. It is the practice of fine-tuning the weights of a neural net based on the error rate (i.e., loss) obtained in the previous epoch (i.e., iteration). Proper tuning of the weights ensures lower error rates, making the model reliable by increasing its generalization. To update the weights, we find the partial derivative of the cost function with respect to each weight and bias and subtract the result from the existing weight values to get the new weight values.

dw1 = (a2 - y) * W2 * a1 * (1 - a1) x

dw2 = (a2 – y) * a1

db1 = a1*(1-a1) * dot(w2, a2-y)

db2 = a2-y

Hyperparameters:

Hyper parameters are factors that eventually affect the performance of the model. The hyper parameters that are of significant importance are number of iterations(epoch), learning rate, number of hidden nodes, bias, weight. By tuning the number of iteration and learning rate, the weight and bias get updated. And the best fitting model is obtained for the optimal values.

w1 = w1 - learning rate * dw1

b1 = b1 - learning rate * db1

w2 = w2 - learning rate * dw2

b2 = b2 – learning rate * dw2

Cost Function:

The Cost function is expressed as a difference between the actual value and the predicted value. Cost function helps the model to correct its behavior to minimize mistakes. The cost function is given by:

$$H(y, \hat{y}) = -\sum_{i} y_i \log \hat{y}_i$$

3.2 MULTI-LAYER NEURAL NETWORK:

The network model is developed with multiple layer such as input layer, hidden layers and output layer. The more hidden layer the deeper the network is. The Input layer takes training dataset, x_train and passes it to hidden layer. Hidden layer multiplies data with weight, w1 and adds it with bias, b1. The output is then passed to activation function such as ReLU, tanh, sigmoid, etc. The output of the activation function is then passed to second hidden layer where it

is then multiplied with weight w2 and added with bias b2. Similarly, this process may go on depends on the hidden layers specified. The final output of the last hidden layer is passed to softmax function and the ouput is predicted. This predicted output is then tested against output dataset, y_train. Each hidden layer has n nodes which is interconnected with each input nodes. The number of nodes in each hidden layer is a hyperparameter. The larger models (with more hidden units) are able to fit the training set better, until eventually the largest models over-fit the data. Tuning the hyperparameters like epoch, batch size, hidden nodes, learning rate, etc, the accuracy of the model is increased.

Architecture used as follows:

Input → Dense(64, tanh) → dropout(0.5) → dense(32, tanh) → dropout(0.5) → dense(10, softmax) → output

3.3. CONVOLUTION NEURAL NETWORK:

A convolutional neural network is a class of deep neural networks, which can take in an input image and assign weights and bias to various objects in the image and will be able to differentiate it from one other. They are regularized versions of multilayer perceptron. Multilayer perceptron usually means fully connected networks; that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfitting data. The role of the convolution network is to reduce the images into a form which is easier to process, without losing any significant feature which are critical for getting a better prediction.

3.3.1. Architecture:

A Convolution Network architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better. The first part of the convolution layer is Kernel/Filter which carries out the convolution operation. Kernel performs a matrix multiplication operation between itself and the portion of image that the kernel is hovering. Then the kernel shifts to the right with certain stride value till it parses the complete width. Then it moves down to the beginning of the image and repeats the process. The convolution operation captures the high-level features. First layer extracts the low-level features like edges, color. With adding more layers, model tends to extract high-level features and predict better value. The convolved feature has either reduced dimensionality or increased/ same dimensionality compared to input. It can be done by applying valid padding in the case of former or same padding in the case of latter. Model Architecture as follows:

Input → conv2D(filters = 64, kernel_size=(3,3), relu) → conv2D(32, (3,3), relu) → maxPooling2D(pool_size=(2,2)) → Dropout(0.25) → flatten() → dense(128, relu) → dropout(0.5) → dense(10, softmax) → Output

3.3.2. Dense Layer:

A dense layer is a linear operation in which every input is connected to every output by a weight (n_inputs * n_outputs weights). A non-linear activation function generally follows dense layers. A densely connected layer provides a learning feature from all the combinations of the features of the previous layer.

3.3.3 Filters:

In convolutional (filtering and encoding by transformation) neural networks (CNN), every network layer acts as a detection filter for the presence of specific features or patterns present in the original data. The first layers in a CNN detect (large) features that can be recognized and interpreted relatively easy. Later layers detect increasingly (smaller) features that are more abstract (and are usually present in many of the more significant features detected by earlier layers). The last layer of the CNN can make an ultra-specific classification by combining all the specific features detected by the previous layers in the input data.

3.3.4 Kernel:

Kernels are used in convolutional layers to extract features. They are filters that you apply to a small region of the image. They are implemented as matrices, the kernel "moves" above the input data, and at each step, the dot product between the kernel and the sub-region of the input matrix below it is calculated, the result of the whole process is a matrix of the dot products. The kernels are learned during the training, just like weights in a conventional layer.

3.3.5 Max pooling 2D:

Max Pooling is a downsampling strategy in Convolutional Neural Networks. It is a sample-based discretization process. The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.), reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned.

3.3.6. Dropout:

Dropout is a regularization technique patented by Google for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks. The term "dropout" refers to dropping out units (both hidden and visible) in a neural network.

3.3.7 Flatten:

Once the pooled featured map is obtained, the next step is to flatten it. Flattening involves transforming the entire pooled feature map matrix into a single column, which is then fed to the neural network for processing. After flattening, the flattened feature map is passed through a neural network. This step is made up of the input layer, the fully connected layer, and the output layer.

3.3.8. Compilation:

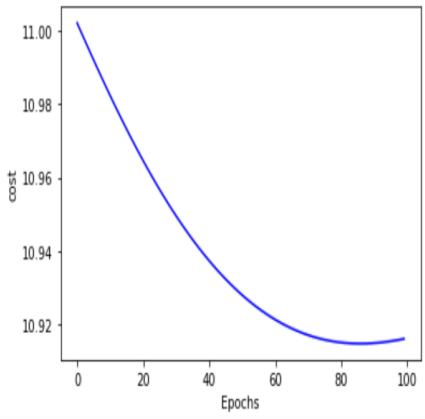
Before training a model, the learning process needs to be configured, which is done via the 'compile' method. It receives three arguments:
· An optimizer: This could be the string identifier of an existing optimizer (such as 'rmsprop' or 'adagrad') or an instance of the Optimizer class.
· A loss function: This is the objective that the model will try to minimize. It can be the string identifier of an existing loss function (such as 'categorical_crossentropy' or 'mse'), or it can be an objective function.
· A list of metrics: For any classification problem, you will want to set this to 'metrics=['accuracy']. A metric could be the string identifier of an existing metric or a custom metric function.

4. MODEL EVALUATION:

The model is then trained with training dataset and the history is used for plotting the graph. Testing dataset is used to evaluate the trained model and accuracy and confusion matrix is calculated for the model. The hyperparameters are tuned so that the accuracy is increased. Predicted output of the model is evaluated with the target variable dataset. Confusion matrix is created for evaluation. It is of 10*10 matrix with the count of correctly predicted value in diagonals.

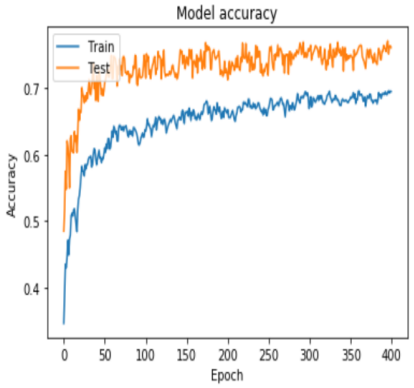
Loss and Accuracy for Single layer Neural Network

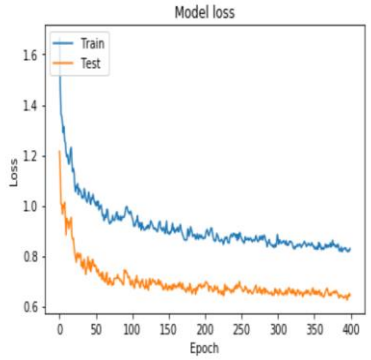
```
Accuracy Score: 61.7 %
Confusion Matrix: [[676  93  71  86   1  56   7   1   8   1]
 [ 36 910  18  27   3   3   2   0   1   0]
 [ 14   9 677  10  78  87 106   0  18   1]
 [137 104  35 676   6  32   8   0   1   1]
 [ 58  20 684  49 103  51  31   0   4   0]
 [  0   1   0   2   0 588   0 234   0 175]
 [229  27 381  55  72 129  89   2  16   0]
 [  0   0   0   0   0  62   0 835   0 103]
 [  4   4  95  47   2  59   0  91 685  13]
 [  1   0   3   1   0  12   0  51   1 931]]
```



hidden_nodes = 512, learning_rate = 0.001, epochs = 100

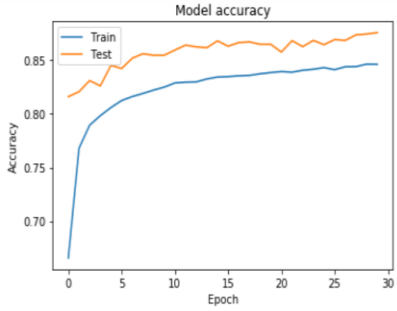
Loss and Accuracy for Multilayer Neural Network

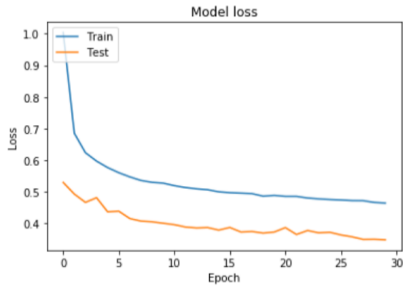




Accuracy 74.2900013923645

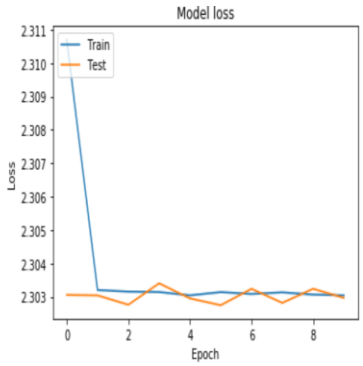
Epoch = 400, optimizer = ‘adam’, loss = categorical_crossentropy, activation function = ‘tanh’, hidden layer = ‘2’



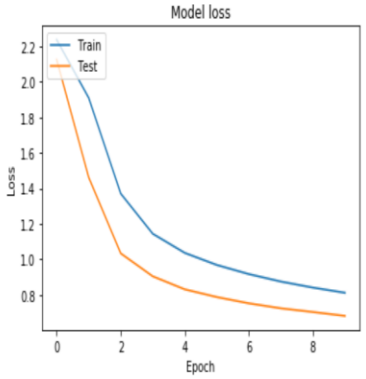


Accuracy for Multilayer Network Model 85.75999736785889

Epoch = 30, optimizer = ‘adam’, loss = categorical_crossentropy, activation function = ‘tanh’, hidden layer = ‘2’



Accuracy for Convolutional Neural Network Model 10.000000149011612

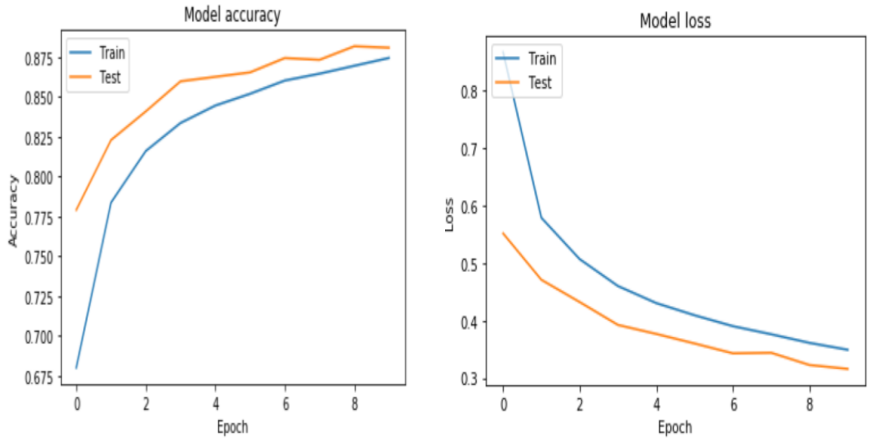


Accuracy for Convolutional Neural Network Model 74.52999949455261

Figure 1 Figure 2

Figure 1 : Graph obtained with values: Epoch = 10, Activation function = ‘relu’, Learning rate = 0.5, batch_size=256, optimizer = ‘stochastic gradient descent’, loss=categorical_crossentropy,

Figure 2: Graph obtained with values: Epoch = 10, Activation function = ‘relu’, Learning rate = 0.5, batch_size=256, optimizer = ‘stochastic gradient descent’, loss=categorical_crossentropy,




```
Accuracy for Convolutional Neural Network Model 88.09000253677368
```

Epoch = 10, Activation function = ‘relu’, Learning rate = 0.05, optimizer = ‘stochastic gradient descent’, loss=categorical_crossentropy,

```
Confusion Matrix [[812   0  16  20   4   2 136   0  10   0]
 [  2 964   0  20   5   0   7   0   2   0]
 [  9   0 799  10 118   1  61   0   2   0]
 [ 17   3   8 892  40   0  37   0   3   0]
 [  0   1  57  39 872   0  31   0   0   0]
 [  0   0   0   1   0 972   0  19   1   7]
 [113   0  89  23 154   0 610   0  11   0]
 [  0   0   0   0   0  14   0 962   0  24]
 [  3   1   4   6   3   2   5   5 971   0]
 [  0   0   0   0   0   6   0  38   1 955]]
```

5.CONCLUSION:

This project helps get a clear idea of the working of Neural Network. It has also helped understand the sigmoid function and the stochastic gradient descent. We calculated the updated weights by computing the error function and minimizing it by each of the approaches. Then finally computed the target values for the testing dataset. We also calculated the confusion matrix and the accuracy for the datasets on each classifier and the results are observed. This project helped us to get a deeper considering of the impact of the hyper parameters on the performance of the system. Thus, we performed a classification predictive model to classify the dataset.

REFERENCES

[1] Comprehensive Guide to Convolutional Neural Network- https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[2] Fundamentals of Keras library in-built function – https://keras.io/getting-started/sequential-model-guide/

[3] Learning Confusion Matrix - https://www.geeksforgeeks.org/confusion-matrix-machine-learning/