CSE-574: MACHINE LEARNING PROJECT 2

CLASSIFICATION USING NEURAL NETWORKS AND CONVOLUTIONAL NEURAL NETWORKS

Madhumitha Sivasankaran
(50310290)
MS in Computer Science
University at Buffalo, NY

INTRODUCTION:

Artificial neural networks (ANN) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains. The neural network itself isn't an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules.

A **Convolutional neural network** (**CNN**, or **ConvNet**) is a class of deep neural networks, most commonly applied to analyzing visual imagery. CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean 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. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function. However, CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme.

ABSTRACT:

In this project, we demonstrate classification using 3 different classifiers.

1. A Neural Network with one hidden layer built from scratch.
2. A multi-layer Neural Network built using Keras.
3. A Convolutional Neural Network built using Keras.

The dataset that is used for this project is Fashion MNIST, which is originally partitioned into a training set ad testing set. The training set will be trained on each of the above mentioned classifiers. After fine-tuning the necessary hyper parameters while training, the testing set will be tested on each of the models with a set of fixed hyper parameters that were obtained during the training phase.

The Loss and Accuracy score is computed for the training and test sets, with which the efficiency of each of the models will be compared. A confusion matrix is also generated for each of the classifiers to compare the relative strengths and weaknesses.

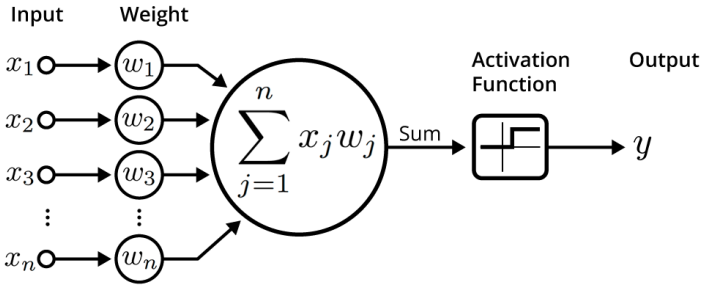
DATASET:

For training and testing of our classifiers, we will use the Fashion-MNIST dataset. The Fashion-MNIST is a dataset of Zalando's article images, consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255. The training and test data sets have 785 columns. The first column consists of the class labels, and represents the type of clothing. The rest of the columns contain the pixel-values of the associated image. Each training and test example is assigned to one of the labels as follows:

1. T-shirt/top
2. Trouser
3. Pullover
4. Dress
5. Coat
6. Sandal
7. Shirt
8. Sneaker
9. Bag
10. Ankle Boot

ARCHITECTURE:

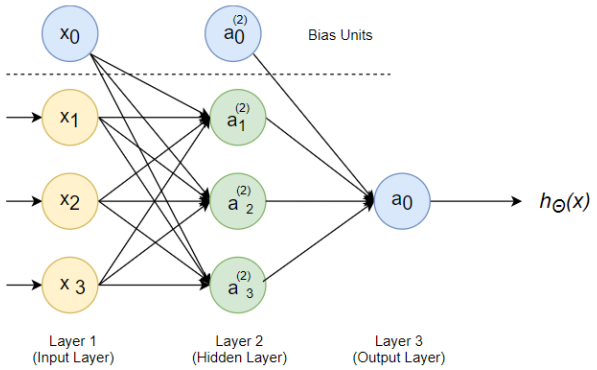
An ANN is based on a collection of connected units or nodes called **artificial neurons**, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal from one artificial neuron to another. An artificial neuron that receives a signal can process it and then signal additional artificial neurons connected to it.



An illustration of an artificial neuron. Source: Becoming Human.

In common ANN implementations, the signal at a connection between artificial neurons is a real number, and the output of each artificial neuron is computed by some non-linear function of the sum of its inputs. The connections between artificial neurons are called **edges**. Artificial neurons and edges typically have a **weight** that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Artificial neurons may have a threshold such that the signal is only sent if the aggregate signal crosses that threshold. Typically, artificial neurons are aggregated into layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first layer (the **input layer**), to the last layer (the **output layer**), possibly after traversing the **inner layers** multiple times.

A **multilayer perceptron (MLP)** is a class of feedforward artificial neural network. An MLP consists of, at least, three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.



$a_i^{(j)}$ - "activation" of unit *i* in layer *j*.

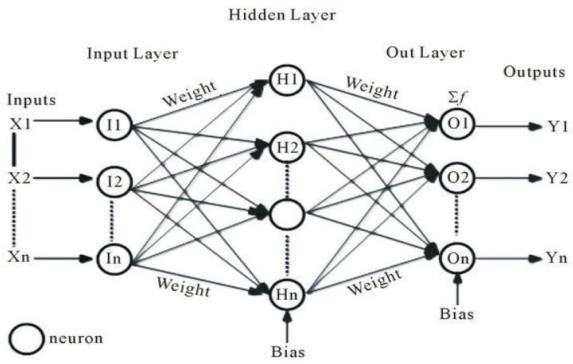
$\Theta^{(j)}$ - matrix of weights controlling function mapping from layer *j* to layer *j + 1*. For example for the first layer: $\Theta^{(1)} \in R^{3x4}$.

L - total number of layers in network.

s_l - number of units (not counting bias unit) in layer *l*.

K - number of output units (1 in our example but could be any real number for multi-class classification).

Multi-class Classification: In order to make neural neutral network to work with multi-class notification we will use One-vs- All approach. In our case, we have to distinguish amongst ten classes of clothing varieties. For this, the output layer of our network will have 10 units.



Forward propagation: Forward propagation is an interactive process of calculating activations for each layer starting from the input layer and going to the output layer. For the simple network mentioned in a previous section above we're able to calculate activations for second layer based on the input layer and our network parameters:

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right)$$

$$a_2^{(2)} = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right)$$

$$a_3^{(2)} = g\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right)$$

The output layer activation will be calculated based on the hidden layer activations:

$$h_\Theta(x) = a_1^{(3)} = g\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right)$$

Where g() function is a sigmoid:

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

Vectorized implementation of Forward Propagation:

$$x = \begin{bmatrix} x_o \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

To simplify previous activation equation, we introduce a ‘z’ variable:

$$z_1^{(2)} = \Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3$$

$$z_2^{(2)} = \Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3$$

$$z_3^{(2)} = \Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3$$

$$z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} = \Theta^{(1)}x = \Theta^{(1)}a^{(1)}$$

It is necessary to add the bias units (activations) before propagating to the next layer.

$a_0^{(2)} = 1,\ so\ that\ a^{(2)} \in R^4$

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$

$$h_\Theta(x) = a^{(3)} = g\left(z^{(3)}\right)$$

Cost Function: The cost function for the neuron network is quite similar to the logistic regression cost function.

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} log\left(h_\Theta\left(x^{(i)}\right)\right)_k + \left(1 - y_k^{(i)}\right) log\left(1 - \left(h_\Theta\left(x^{(i)}\right)\right)_k\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\Theta_{ji}^{(l)}\right)^2$$

$$h_\Theta(x) \in R^K$$

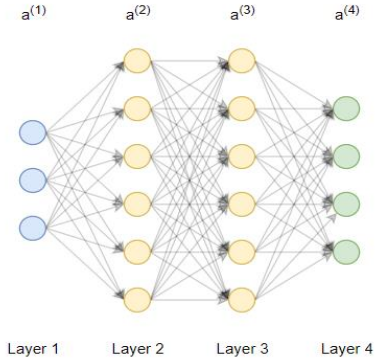
$$\left(h_\Theta(x)\right)_i = i^{th}\ output$$

Back Propagation:

Gradient Computation: The Back Propagation Algorithm has the same purpose as gradient descent for logistic regression – it corrects the values of thetas to minimize the cost function. So, we have to calculate partial derivative of cost function for each theta.

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$$

Consider the below MLP, let us assume that:



$\delta_j^{(l)}$ - "error" of node *j* in layer *l*.

For each output unit (layer *L = 4*):

$\delta_j^{(4)} = a_j^{(4)} - y_j$, or in vectorized form:

$$\delta^{(4)} = a^{(4)} - y$$

$$\delta^{(3)} = \left(\Theta^{(3)}\right)^T \delta^{(4)} .* g'\left(z^{(3)}\right)$$
$$\delta^{(2)} = \left(\Theta^{(2)}\right)^T \delta^{(3)} .* g'\left(z^{(2)}\right)$$

$\delta^{(1)}$ - is for input layer, we cannot change it

g' - sigmoid gradient

$$g'(z) = \frac{\partial}{\partial z} g(z) = g(z)(1 - g(z)),\ where\ g(z) = \frac{1}{1 + e^{-z}}$$

Now, calculate the gradient step:

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$

Backpropagation algorithm:

For the training set : $\left\{\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right)\right\}$, we have to set : $\Delta_{ij}^{(l)} = 0\ (for\ all\ l,\ i,\ j)$

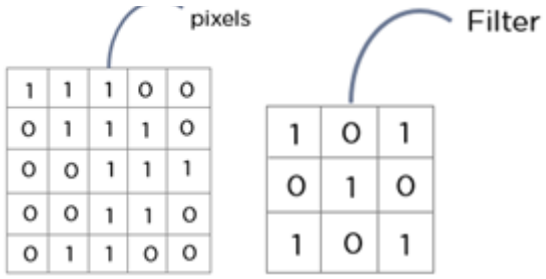
$for\ i = 1\ to\ m$
$\quad Set\ a^{(1)} = x^{(i)}$
$\quad Perform\ forward\ propagation\ to\ compute\ a^{(l)}\ for\ l = 2, 3, \ldots, L$
$\quad Using\ y^{(i)},\ compute\ \delta^{(L)} = a^{(L)} - y^{(i)}$
$\quad Compute\ \delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^2$
$\quad \Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}\ \left(or\ in\ vectorized\ form\ \Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} \left(a^{(l)}\right)^T\right)$

$$D_{ij}^{(l)} := \begin{cases} \frac{1}{m} \Delta_{ij}^{(l)} + \frac{\lambda}{m} \Theta_{ij}^{(l)} & if\ j \geq 1 \\ \frac{1}{m} \Delta_{ij}^{(l)} & if\ j = 0 \end{cases}$$

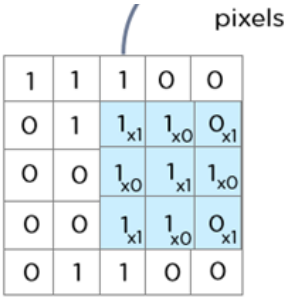
$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

A CNN accepts arrays of pixel values as input to the network. The hidden layer consists of several different layers which carry out feature extraction. There is a fully connected layer that recognizes the objects in the image. Convolution operation forms the core of every convolution neural network. There are 4 layers in a CNN. These are **Convolution layer, ReLU layer, Pooling layer and Fully Connected Layer**.

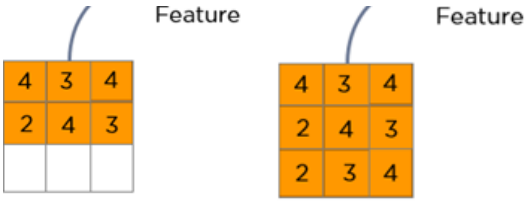
The **Convolution layer** uses a filter matrix over the array of image pixels and performs convolution operation to obtain a **convolved feature map**. Below is an example which represents the convolution operation over the input array.

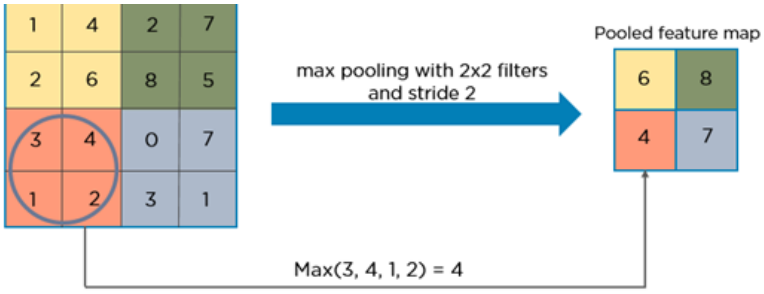


We shall slide this filter matrix over the input image and compute the convolution operation.



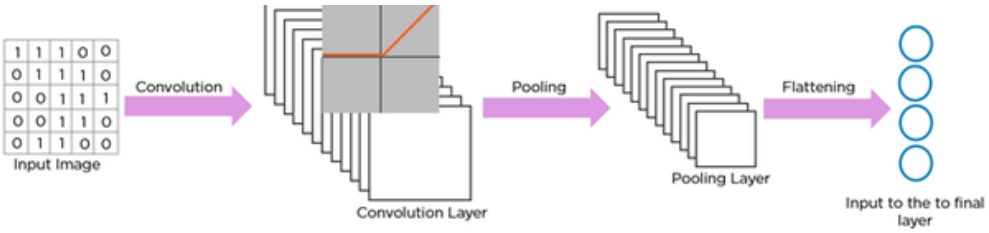
The result is a matrix called the the Convolved feature map.





Pooling layers uses different filters to identify different parts of the image like edges, corners, body, etc.

The pooled feature map is then converted into a **long continuous linear vector**. This process is called **Flattening**. This flattened matrix goes through a Fully Connected Layer to classify the images.



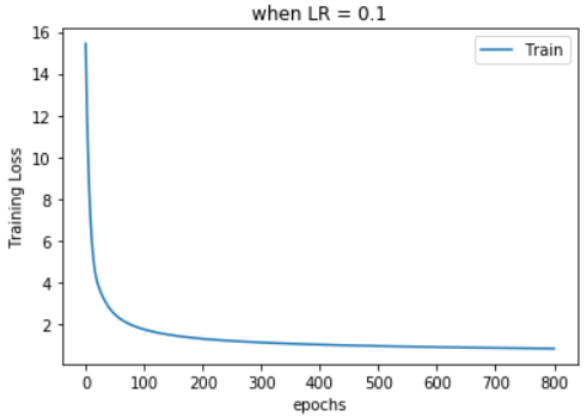
RESULT AND OBSERVATIONS:

Based on the given dataset, the samples have been trained, tested and validated in each of the three classifiers. The training set was trained and the optimal hyper parameters that seemed the most efficient were found out. The detailed performance based analysis on the basis of Loss and Accuracy is as follows:

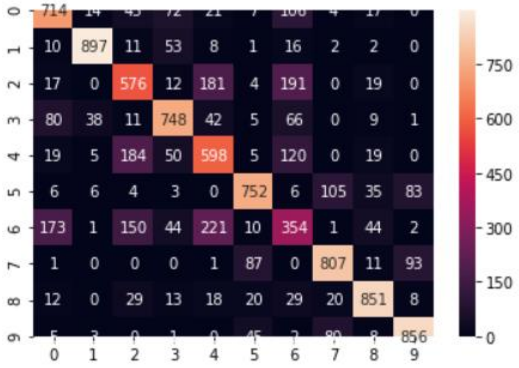
1. SINGLE HIDDEN LAYER NEURAL NETWORK

Epochs: 100
Learning Rate: 0.1
Training Loss: 0.811630
Training Accuracy: 74.628 %
Test Accuracy: 73.009 %

Training Loss vs Epochs plot:



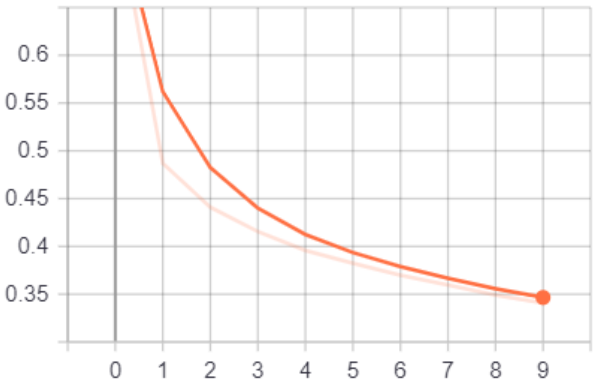
Confusion Matrix:



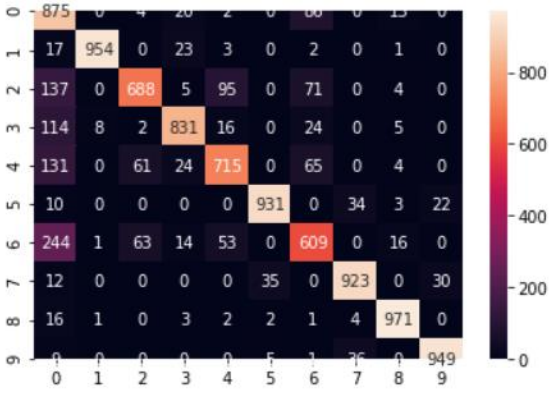
2. MULTILAYER NEURAL NETWORK

Epochs: 10
Batch Size: 128
Number of Hidden Nodes: 500
Test Accuracy: 88.34%
Test loss : 0.33166

Training loss vs Epochs plot:



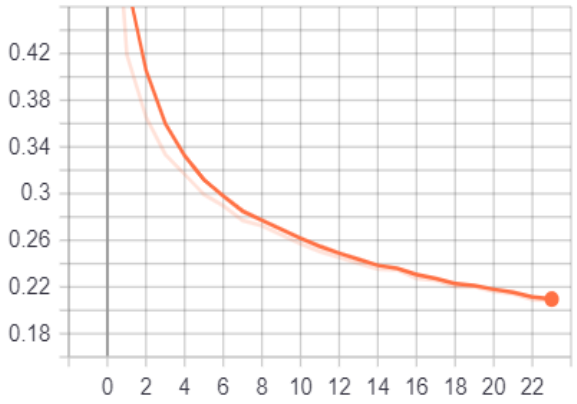
Confusion Matrix:



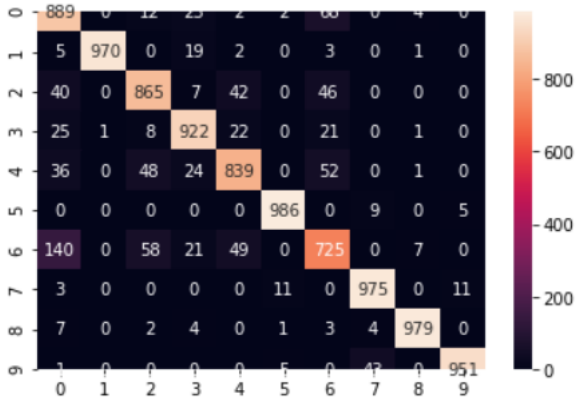
3. CONVOLUTIONAL NEURAL NETWORK

Epochs: 24
Batch Size: 128
Test Accuracy: 91.37%

Training loss vs Epochs plot:



Confusion Matrix:



CONCLUSION:

Thus from the above results and observations, it can be concluded that the Convolutional Neural Networks is the most efficient out of the three classifiers that has been used in this project. Comparing the cost and accuracy values, the CNN provided the highest Test Accuracy of 91.37%

REFERENCES:

- https://keras.io/models/model/
- https://en.wikipedia.org/wiki/Convolutional_neural_network
- https://towardsdatascience.com/simple-introduction-to-neural-networks-ac1d7c3d7a2c
- https://towardsdatascience.com/a-gentle-introduction-to-neural-networks-series-part-1-2b90b87795bc
- http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html
- https://www.geeksforgeeks.org/applying-convolutional-neural-network-on-mnist-dataset/