Classification of Fashion MINST Images Using Neural Networks

Project 2 CSE 574 Intro to Machine Learning

Krishna Naga Karthik BODAPATI
Department of Computer Science
(SUNY) University at Buffalo
Buffalo NY, 14221
kbodapat@buffalo.edu

Abstract

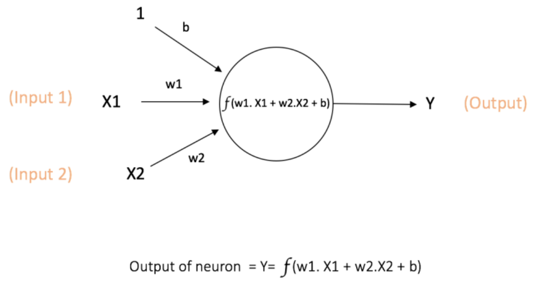
The Goal of this project to classify Fashion MINST clothing images into 10 classes. We achieve this using Artificial Neural networks with one hidden layer, with multiple hidden layers and Convolutional Neural Networks.

1 Introduction

1.1 Neural Networks

An Artificial Neural Network (ANN) is a computational model that is inspired by the way biological neural networks in the human brain process information. Artificial Neural Networks have generated a lot of excitement in Machine Learning research and industry, thanks to many breakthrough results in speech recognition, computer vision and text processing. In this blog post we will try to develop an understanding of a particular type of Artificial Neural Network called the Multi-Layer Perceptron.

The basic unit of computation in a neural network is the **neuron**, often called a **node** or **unit**. It receives input from some other nodes, or from an external source and computes an output. Each input has an associated **weight** (w), which is assigned on the basis of its relative importance to other inputs. The node applies a function ***f*** (defined below) to the weighted sum of its inputs as shown here



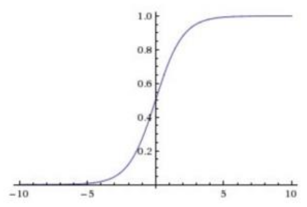
The above network takes numerical inputs **X1** and **X2** and has weights **w1** and **w2** associated with those inputs. Additionally, there is another input **1** with weight **b** (called the **Bias**) associated with it. We will learn more details about role of the bias later.

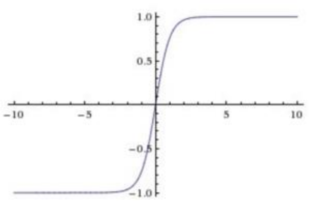
1.2 Activation Function

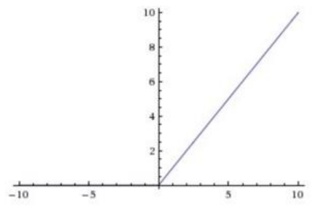
The output **Y** from the neuron is computed as shown in the Figure 1. The function ***f*** is non-linear and is called the **Activation Function**. The purpose of the activation function is to introduce non-linearity into the output of a neuron. This is important because most real-world data is non-linear and we want neurons to *learn* these non-linear representations.
Every activation function (or *non-linearity*) takes a single number and performs a certain fixed mathematical operation on it [2]. There are several activations functions you may encounter in practice:

- **Sigmoid:** takes a real-valued input and squashes it to range between 0 and 1
  $$\sigma(x) = 1 / (1 + \exp(-x))$$
- **tanh:** takes a real-valued input and squashes it to the range [-1, 1]
  $$\tanh(x) = 2\sigma(2x) - 1$$
- **ReLU**: ReLU stands for Rectified Linear Unit. It takes a real-valued input and thresholds it at zero (replaces negative values with zero)
  $$f(x) = \max(0, x)$$

These are graphs for each of the above activation functions.


Sigmoid


tanh


ReLU

2 DATASETS

The Fashion-MNIST dataset used for training, validation and testing of the Neural Networks and Convolutional Neural Networks. 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 (see above), and represents the article of clothing. The rest of the columns contain the pixel-values of the associated image. Each training and test example are assigned to one of the labels as shown in table 1.

1	T-Shirt/Top
2	Trousers
3	Pullover
4	Dress
5	Coat
6	Sandal
7	Shirt
8	Sneaker
9	Bag
10	Ankle Boot

3 Pre-Processing

3.1 Normalization

It is not a great idea to use un normalized data for training, Normalizing data gives equal importance all features and stops the domination of features with larger values. In this project since all input variables are pixels Normalization is dividing the Input by 255

3.2 Partitioning of Data:

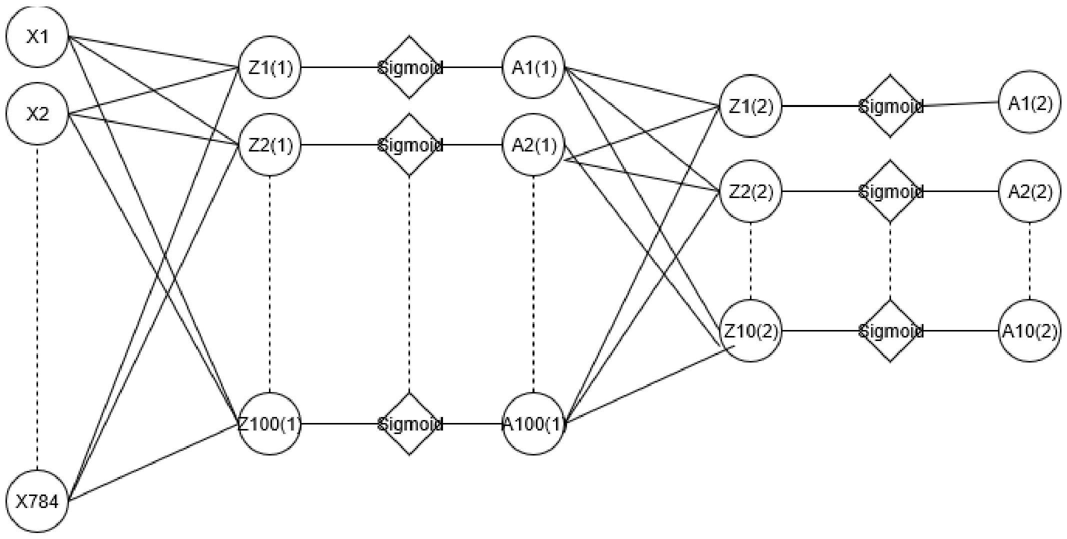
It is bad idea to use all examples to train the model, because it would be impossible to test accuracy of the model as It remembers all training examples and also, we cannot distinguish whether the fit is good fit or overfit. So, it is a good idea to partition dataset before Training

In this problem we split the data set into 2 parts: Training set (60000 examples) which is used to train the model and Testing Set (10000 examples) to test and determine the accuracy of the model.

4 Model Architecture

Task 1

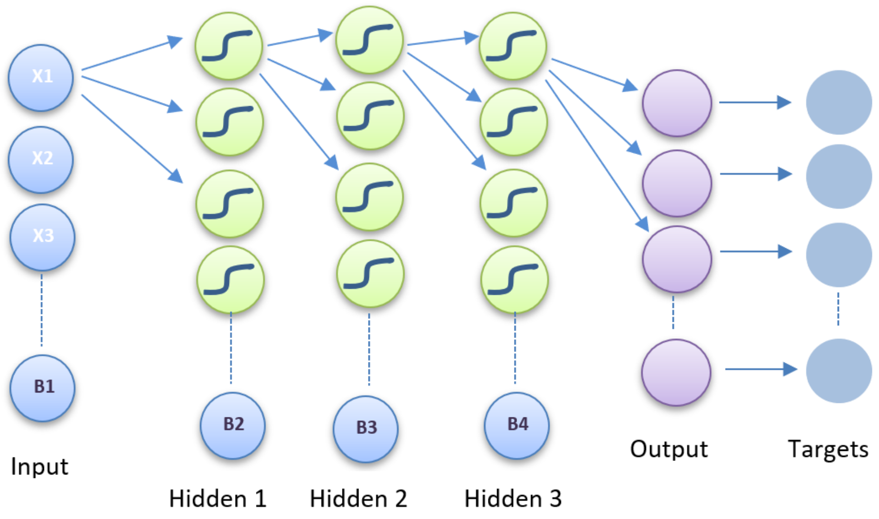
There are 784 input features(28x28 images) and 10 output features. In task1, neural network with only one hidden layer is implemented with hidden nodes 100. Since they are 3 layers in total (input, hidden, output) there are 2 weight matrices: W1 weights from input to hidden layer and W2 weights from hidden layer to output layer. The input to hidden layer is linear combination of weights * input vectors and this input is passed through activation function (sigmoid in our case) and Prediction is linear combination of W2 * hidden layer values and is passed through SoftMax function.



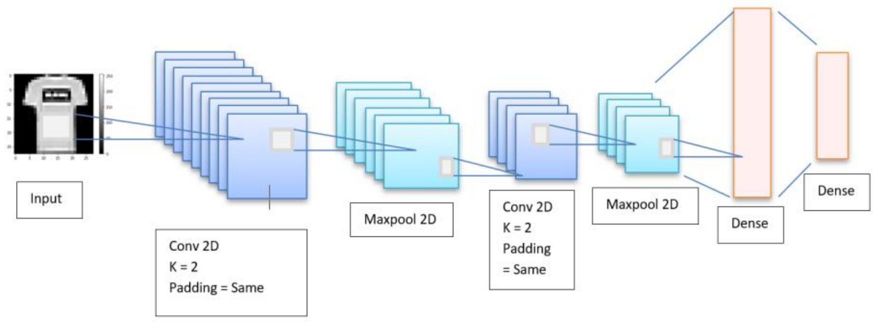
Task 2:

Created Multi-layered Neural Network using keras and the Model summary is

Model Summary:



Task 3:



4.1 Back Propagation:

We need to determine the weights for our model. First, we initialize all weights to 0s and update these weights for each iteration. Here our goal is to minimize cost function which, Cost function is minimum when slope of cost function = 0. So we We have two inputs: x1 and x2. There is a single hidden layer with 3 units (nodes): y1, y2, and y3. Finally, there are two outputs: y1 and y2. The arrows that connect them are the weights. There are two weights matrices: w, and u. The w weights connect the input layer and the hidden layer. The u weights connect the hidden layer and the output layer. We have employed the letters w, and u, so it is easier to follow the computation to follow. You can also see that we compare the outputs y1 and y2 with the targets t1 and t2.

There is one last letter we need to introduce before we can get to the computations. Let a be the linear combination prior to activation. Thus, we have:
a(1) = xw + b(1) and a(2) = hu + b(2).
Since we cannot exhaust all activation functions and all loss functions, we will focus on two of the most common. A sigmoid activation and an L2-norm loss. With this new information and the new notation, the output y is equal to the activated linear combination. Therefore, for the output layer, we have y = σ(a(2)), while for the hidden layer: h = σ(a(1)).

$$\text{L2-norm loss: } L = \frac{1}{2}\sum_i (y_i - t_i)^2 \qquad \sigma(x) = \frac{1}{1+e^{-x}}$$

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

$$\mathbf{u} \leftarrow \mathbf{u} - \eta \nabla_{\mathbf{u}} L(\mathbf{u})$$

$$\frac{\partial L}{\partial u_{ij}} = \frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial a_j^{(2)}}\frac{\partial a_j^{(2)}}{\partial u_{ij}}$$

$$\frac{\partial L}{\partial y_j} = (y_j - t_j)$$

$$\frac{\partial y_j}{\partial a_j^{(2)}} = \sigma(a_j^{(2)})(1 - \sigma(a_j^{(2)})) = y_j(1 - y_j)$$

$$\frac{\partial a_j^{(2)}}{\partial u_{ij}} = h_i \qquad \frac{\partial L}{\partial u_{ij}} = \frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial a_j^{(2)}}\frac{\partial a_j^{(2)}}{\partial u_{ij}} = (y_j - t_j)y_j(1 - y_j)h_i = \delta_j h_i$$

4.2 Hyperparameters

Here we need to decide what my hyper parameters which are: 1. No. of iterations 2. Learning rate,3. No. hidden layers, 4:No nodes in each layer tuning theses parameters is crucial because we don't want to overfit or under fit the data and also, we need our model to converge fast. If the learning rate is too high the model converges too fast and cost function may increase and If the learning rate is too small, it takes longer to converge.

4.3 Convolutional Neural Networks

Convolution Operation:

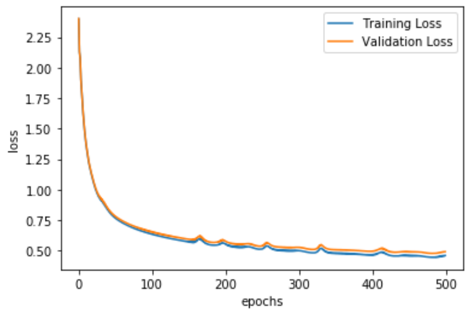
In mathematics (in particular, functional analysis) convolution is a mathematical operation on two functions (f and g) that produces a third function expressing how the shape of one is modified by the other. The term convolution refers to both the result function and to the process of computing it. It is defined as the integral of the product of the two functions after one is reversed and shifted.

Pooling

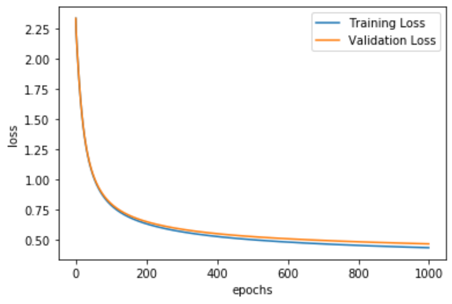
It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 down samples every depth slice in the input by 2 along both width and height, discarding 75% of the activations. Every MAX operation would in this case be taking a max over 4 numbers (little 2x2 region in some depth slice). The depth dimension remains unchanged

Plots (loss vs epocs)

Task 1 :

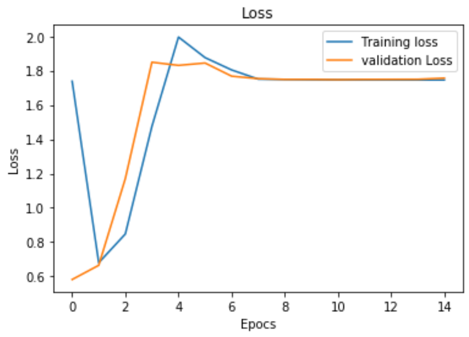


High Alpha

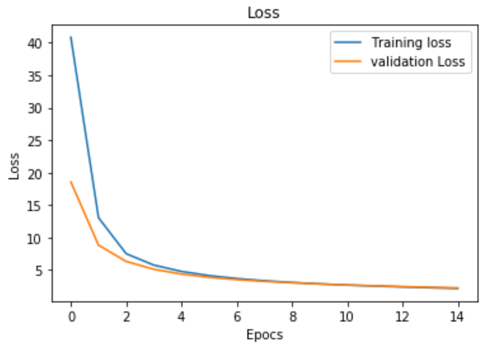


Good Alpha

Task 2 :

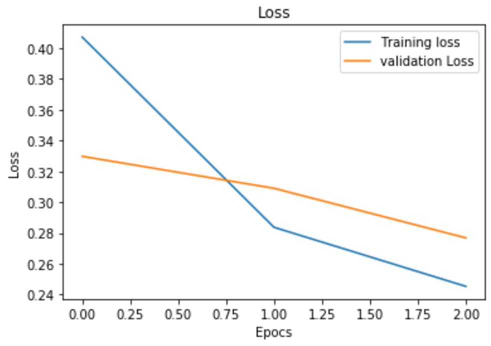


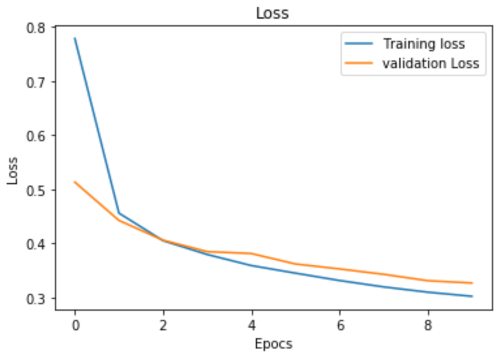
High Alpha



Good Alpha

Task 3





Less Alpha

Good Alpha

5 Results

Task 1:

Accuracy:

```
0.8287
```

Confusion Matrix for Task 1

```
 [[807,   3,  13,  55,   6,   4,  95,   0,  16,   1],
  [  4, 944,  11,  32,   6,   0,   1,   0,   2,   0],
  [ 17,   4, 724,  11, 140,   2,  91,   0,  11,   0],
  [ 34,  13,  12, 858,  37,   0,  42,   0,   4,   0],
  [  0,   2, 107,  35, 761,   1,  88,   0,   6,   0],
  [  0,   0,   0,   2,   0, 890,   0,  60,   8,  40],
  [160,   2, 124,  41, 106,   2, 533,   0,  32,   0],
  [  0,   0,   0,   0,   0,  48,   0, 891,   0,  61],
  [  2,   1,  11,  11,   2,   5,  20,   5, 942,   1],
  [  0,   0,   0,   0,   0,  17,   0,  45,   1, 937]]
```

Task 2:

Confusion Matrix for Task 2

```
[[900,   0,  17,  45,   1,   0,  35,   0,   2,   0],
 [  4, 958,   2,  29,   4,   0,   2,   0,   1,   0],
 [ 25,   1, 852,  18,  67,   1,  36,   0,   0,   0],
 [ 37,   1,   9, 927,   8,   0,  17,   0,   1,   0],
 [  1,   0, 144,  74, 738,   0,  41,   0,   2,   0],
 [  0,   0,   0,   1,   0, 917,   0,  22,   4,  56],
 [252,   0, 136,  34,  76,   0, 498,   0,   4,   0],
 [  0,   0,   0,   0,   0,  16,   0, 944,   0,  40],
 [ 19,   0,   4,   7,   3,   3,  11,   4, 949,   0],
 [  0,   0,   0,   0,   0,   2,   1,  30,   0, 967]]
```

Accuracy

```
0.8650000095367432
```

Task 3

Confusion Matrix for Task 3

```
[[855,   1,  23,  16,   3,   1,  93,   1,   7,   0],
 [  1, 977,   0,  15,   2,   0,   4,   0,   1,   0],
 [ 18,   0, 894,   7,  39,   0,  39,   0,   3,   0],
 [ 13,   4,   7, 919,  22,   0,  31,   0,   4,   0],
 [  4,   2,  80,  28, 832,   1,  51,   0,   2,   0],
 [  0,   0,   0,   0,   0, 971,   0,  15,   1,  13],
 [ 95,    0,   72,   16,   56,    0, 753,    0,    8,    0],
 [  0,    0,    0,    0,    0,    2,    0, 981,    0,   17],
 [  7,    0,    5,    2,    1,    2,    1,    3, 979,    0],
 [  1,    0,    0,    0,    0,    3,    0,   28,    0, 968]]
```

Accuracy:

```
0.9128999710083008
```

6 References

1. Introduction - https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/
2. Pooling layer theory - http://cs231n.github.io/convolutional-networks/#pool
3. Wikipedia – CNN Intro
4. Backpropagation - https://365datascience.com/backpropagation/