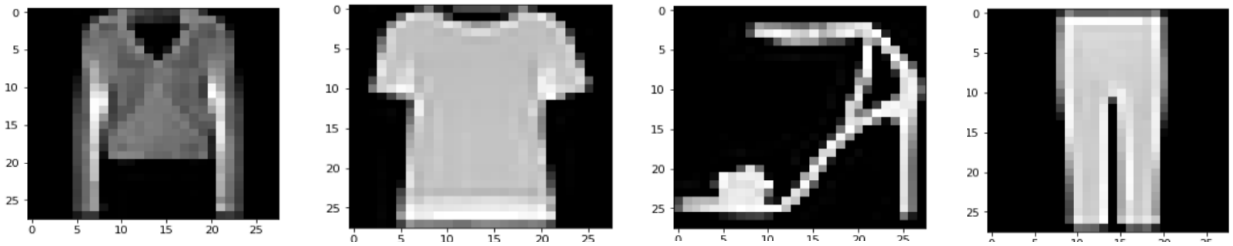
MULTI LAYER NEURAL NETWORK FROM SCRATCH

Submitted by : M P Fousiya

MULTILAYER PERCEPTRON

Many problems that comes across machine learning will have features that have non-linear relationship with the target column. The neural networks are good at understanding this non-linearity of features. Here I am having an image recognition dataset with 784 features(28x28 pixels) and one 'label' column with values from 0-9 representing different classes of images. The training dataset 'fashion-mnist_train.csv' have 60,000 rows and the testing dataset 'fashion-mnist_test.csv' have 10,000 rows. Following are some of the examples of images in our dataset.

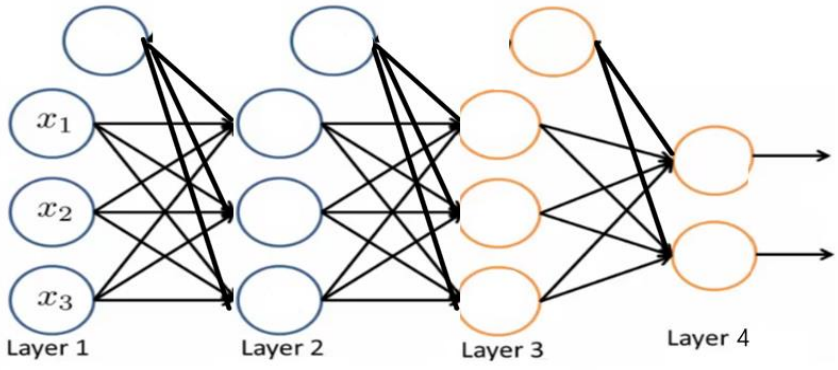


The images in the dataset belongs to one of the following ten categories:

0	1	2	3	4	5	6	7	8	9
Tshirts/Top	Trousers	PullOver	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle Boot

I am using python programming language to build a multi layer neural network with four layers : input layer,output layer and two hidden layer.The input layer have 784 (+ one bias) units representing 784 pixel values and output layer have 10 units for representing 10 different classes.Here I had chosen 20 (+ one bias) units in the hidden layer1 and 5(+one bias) in hidden layer2.

Following is an example of a four layer neural network with 3(+one bias) units at input and two hidden layers and two units at the output layer.



1. Normalizing the dataset

Currently the pixel values are ranging between 0 – 255 .For the model to work well with the activation functions we are using ,these features need to be normalised so that we can avoid

computations with higher numbers.The feature values are normalized so that their mean is 0 and variance 1.

2. Initialising weight matrix

The weight matrix at each layer is randomly initialised with a normal distribution at first.Initialising the weights to a zero matrix will not work well in complex neural networks.The weight matrix at the input layer,hidden layer 1 and hidden layer 2 is having dimension 785x20, 21x5 and 6x10 respectively.The values inside the weight matrix needed to be small for the overall operation to be done without any overflow.

3. Forward Propagation

Forward propagation is the method by which we obtain the output.The input units with bias is multiplied with the weight matrix with dimension 785x20 to obtain the corresponding value.The activation function **tanh** is applied on this value to obtain the hidden layer 1. This value at the hidden layer 1 and an added bias is multiplied with the matrix of dimension 21x5 and the resulting values are then passed to the **tanh** activation function to obtain the hidden layer 2.This value at the hidden layer 2 and an added bias is multiplied with the matrix of dimension 6x10 to produce the corresponding value.This value is then passed to an activaion function called **softmax** to produce the output layer.If a_j is the activation units in layer j and the weight matrix mapping the layer j to j+1 is Θ^j, then $a_{(j+1)}$ is obtained using the following:

$z_{(j+1)} = (\Theta^j)^T a_j$ and $a_{(j+1)}=g(z_{(j+1)})$ where g is the activation function used and T is for transpose.In our model **tanh** is the activation function used at the hidden layers . The tanh of any value ranges between -1 to +1 and is centered around 0.So that the average of those values also comes closer to 0.It is easy to find the derivation of tanh ie $\mathbf{1-tanh^2x}$.

Softmax is the activation function used at the output layer.This function produces probability values between 0 and 1 and sum of these values will be 1. If z is the K dimensional output vector obtained at the output layer from the hidden layer 2.Then softmax(z_j) is,

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

This function is used in the output layer when the classes are mutually exclusive.The output layer now have the probability values of corresponding target class.The class having higher probability is the corresponding prediction value

4. Backward Propagation

Backward propagation is used to identify the error at the output layer and propagate that error back to the input layer to adjust the weight matrix there such that in next iteration we should have less error than before.As soon as we done the forward propagation the error at the output layer is found and according to that the weight matrix at the two hidden layers and the input layer are adjusted There is no error for the input unit values as they are the features provided to us. The steps 3 and 4 is done iteratively on the training dataset until we reach to a minimum cost and optimized weights.If error at layer j is δ_j **,** then the error at the previous layer is as follows:

$\delta_{j-1} = (\Theta^{j-1})^T \delta_j * g'(z_{j-1})$ where $g'(z_{j-1})$ is the derivation of the activation function used.After finding δ at each layer ,the weight matrices Θ^j is updated as follows.

$\Theta^j = \Theta^j$ – alpha*(δ_{j+1})(a_j)^T where alpha is the learning rate

The cost function used when we use softmax at the output layer is $\mathcal{L}(y, \hat{y}) = -\sum_{i=1}^{N} y^{(i)} \log \hat{y}^{(i)}$

5. Regularisation

If there are large number of features it may be difficult to run our neural network with all of those features and also may result in overfitting for complex neural networks(with more than one hidden layer) when we are having small training datasets. But reducing the number of features is not a good idea as it is like throwing away some information. Regularisation is a method that helps us to deal with this problem.This method keeps all the features,but reduce the magnitude of weights corresponding to each features during each iteration .Regularisation works well when we have lot of features and each of which contributes a bit for predicting corresponding class by multiplying it with a regularisation parameter **λ** ,say 0.04.If **λ is very large** all Θ values are heavily penalized and most of their values will become closer to 0 and our prediction also becomes closer to 0. This may result in high bias or underfitting. If **λ is very small ie λ approximately equals 0 ,**so no regularization occurs and this may result in high variance or overfitting

During updation of the weight matrix Θ^j this regularisation term **λ*** Θ^j is also included as:
$\Theta^j = \Theta^j$ – alpha*[((δ_{j+1}) a_j^T) + (**λ*** Θ^j)] where alpha is the learning rate

6. Momentum

We use gradient descent algorithm in neural networks to minimize the error to reach a global minima.In real the error surface is very complex comprising of local minimas and our algorithm may get stuck at the local minima.To avoid this we use a term of momentum which has a value between 0 and 1 that enables to increase the size of the steps taken to reach the minimum and try to jump from the local minima.Using the momentum term make the covergence process faster.Keeping very small value for the momentum may not help in avoiding local minima.Including the momentum term the equation for weight updation is as follows:

$\Theta^j = \Theta^j$ – alpha*[((δ_{j+1}) a_j^T) + (**λ*** Θ^j)]+(dw_j * momentum) where dw_j is the change in weight in layer j in previous iteration without including the regularization term.

7. Learning rate decay

As we had used a term for momentum keeping the learning rate high may cause overshooting while performing gradient descent algorithm.So in each iteration the learning rate is decayed to avoid this problem.

8. Testing the model on the test set

After training the model using above methods,it is now tested against the testing dataset and obtained the prediction for the ‘label’ column.The prediction value is obtained as a list of probabilities out of which the index of the list with high probability is taken as the class value for our prediction.The accuracy of the model is tested.

COURSEWORK 1b - RESULTS

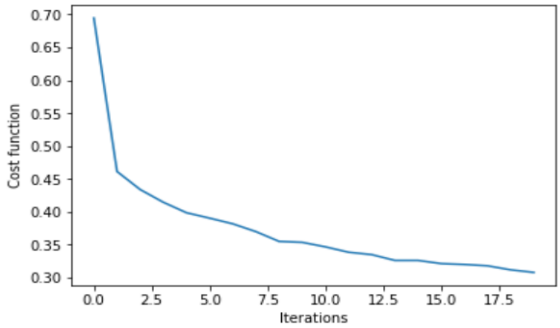
Topology: Four layered Neural Network with number of units in each layer as follows:

- Input layer units=784
- 1st Hidden Layer units=20
- 2nd Hidden Layer units=5
- Output Unit=10

1. Model is trained on the entire training dataset of 60,000 rows

- Learning rate =0.001 with the application of learning rate decay
- Regularisation term, λ =0.01 for all layers
- Momentum=0.01
- Number of iterations=200

Following is the graph of the cost function:



Accuracy on the training set=70.728%

Accuracy on the testing set=69.71%

The training took about `6918.298996686935 seconds`

Both training set and testing set is having approximately same accuracy without overfitting.

2. Model is trained on the entire dataset of 60,000 rows of the training set ,without regularisation

- Learning rate =0.001 with the application of learning rate decay
- Regularisation term, λ =0 for all layers
- Momentum=0.01
- Number of iterations=200

Accuracy on the training set=71.54%

Accuracy on the testing set=71.7%

Since we are having large dataset no overfitting occurred even though we have not applied regularisation.

3. Model is trained on the subset of 10,000 rows of the training set ,without regularisation

- Learning rate =0.001 with the application of learning rate decay

- Regularisation term, λ =0 for all layers
- Momentum=0.01
- Number of iterations=200

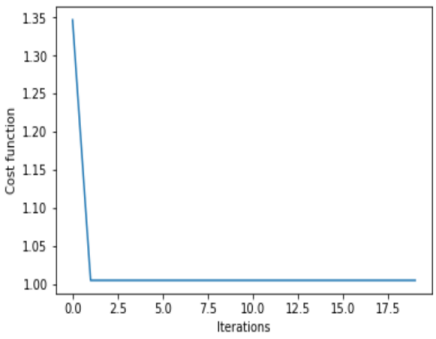
Accuracy on the training set=70.36%

Accuracy on the testing set=67.98%

Since we are having small dataset overfitting occurred when λ =0

3. **Model trained on the subset of 10,000 rows of training set and with no momentum.**

- Regularisation term, λ =0.01
- Momentum=0
- Number of iterations=200




```
Training error 1.34656
Training error 1.00510
Training error 1.00510
Training error 1.00510
Training error 1.00510
Training error 1.00510
Training error 1.00510
Training error 1.00510
Training error 1.00510
Training error 1.00510
Training error 1.00510
Training error 1.00510
Training error 1.00510
Training error 1.00510
Training error 1.00510
Training error 1.00510
Training error 1.00510
Training error 1.00510
Training error 1.00510
Training error 1.00510
```

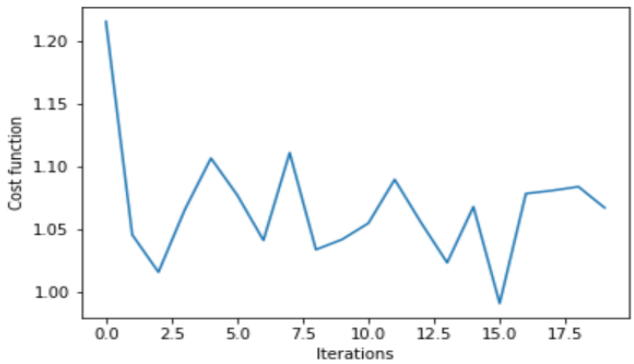
Accuracy on the training set =59.77%

Accuracy on the testing set=58.18%

The model is now not able to reach to the global minima.It has got stuck at a local minima and not able to reduce the cost further.The accuracy is less compared to the previous model.

4. **Model trained on the subset of 10,000 rows of training set with static learning rate=0.001 by avoiding learning rate decaying process and with high momentum.**

- Regularisation term, λ =0.01
- Momentum=0.1
- Number of iterations=200



Accuracy on the training set =24.73% . Accuracy on the testing set=23%

The error value is overshooting between high and low values,but not converging to a minimum value.

6. **Model trained on the subset of 10,000 rows of training set with high value for regularisation λ**

- Regularisation term, λ =0.5
- Momentum=0.05
- Number of iterations=200

Accuracy for training set:57.23%

Accuracy for the testing set:56.79%

Now the parameters in the weight matrix is heavily penalised and it results in underfitting.

Conclusions:

a. Smaller neural network is prone to underfitting and computationally cheaper.Larger neural network is computationally expensive and more prone to high variance when we are doing training on smaller datasets .
b. Following can avoid overfitting:
    - Having large dataset.
    - Using regularisation.
c. Increasing λ fixes high variance or overfitting and decreasing λ fixes high bias or underfitting.
d. The momentum term is also highly important with learning rate decay to avoid our model to get stuck in a local minima while optimizing the weight matrices.
e. Tweaking the number of units in the hidden layers can also help in increasing the accuracy level.