E0 250 - Deep Learning

Project 2 - FNN and CNN on Fashion-MNIST

Deepesh Virendra Hada
M.Tech, Department of CSA
SR: 17196

21 February, 2020

1 Multilayer Feedforward Neural Network

1.1 Network Architecture

The neural network constructed is a *fully connected neural network* (FCNN) with 5 layers, an *input* layer, 3 *hidden* layers with *LeakyReLU* activation and a Softmax *output* layer. Following are the number of neurons in each layer:

- Hidden Layer 1 - 2000
- Hidden Layer 2 - 2000
- Hidden Layer 3 - 2000
- Output Layer (Softmax) - 10

The inputs, consisting of $60,000$ grayscale images, are first divided into random batches of $1,000$. Each image has a dimension of 28×28, and being grayscale, they have only 1 channel as opposed 3 for color images. Out of the batch selected, each image is first flattened to a 784-dimensional vector as $28 \times 28 = 784$. This flattened vector is then passed on to the first hidden layer with certain *weights*, as can be seen in the code. Also, a *bias* is applied to each neuron in the hidden layer, hence, $2,000$ biases in the first layer.

Each neuron, takes a linear combination of the input variables, and applies a *leaky ReLU* activation function to that combination. The negative slope of the activation function is set as a hyperparameter to 0.01. These outputs are then passed on to the deeper layers where similar processing is done. All the weights and biases of all the layers are initially set randomly. The output of each neuron is given as follows:

$$g(x) = 1(x < 0)(0.01x) + 1(x \geq 0)(x) \tag{1}$$

All these outputs from the last hidden layer are given to the output layer, consisting of 10 neurons, with some weights and bias. Now, the final outputs are probabilities of an input belonging to any one of these 10 classes. Since the leaky ReLU function doesn't satisfy the properties of a probability distribution, the *softmax* function is applied to the *activation outputs* of the hidden layer neurons. The probability of class j for an input x by the *softmax* function is given as:

$$p_j = \frac{e^{-z_j}}{\sum_{i=0}^{3} e^{-z_i}} \tag{2}$$

The 10 softmax activated neurons correspond to the following 10 output classes in which the incoming images have to be classified:

Label	Class description
0	T-shirt/Top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle Boot

1.2 Why this Architecture?

We started off with a reasonably small and sparse network with 2 hidden layers , each consisting of 100 neurons. This network performed poorly with respect to the current state-of-the-art accuracies. Gradually, another hidden layer was added and the number of neurons were increased considerably.

With 2500 neurons in each of the three hidden layers, the network performed at par with only 2000 in each layer. Also, more number of neurons took more training time and resulted in a bulky model with little or no difference in the validation accuracy. Hence, we stuck to this network which gave a reasonable balance between the final validation accuracy and also thinner network.

1.3 Hyperparameters

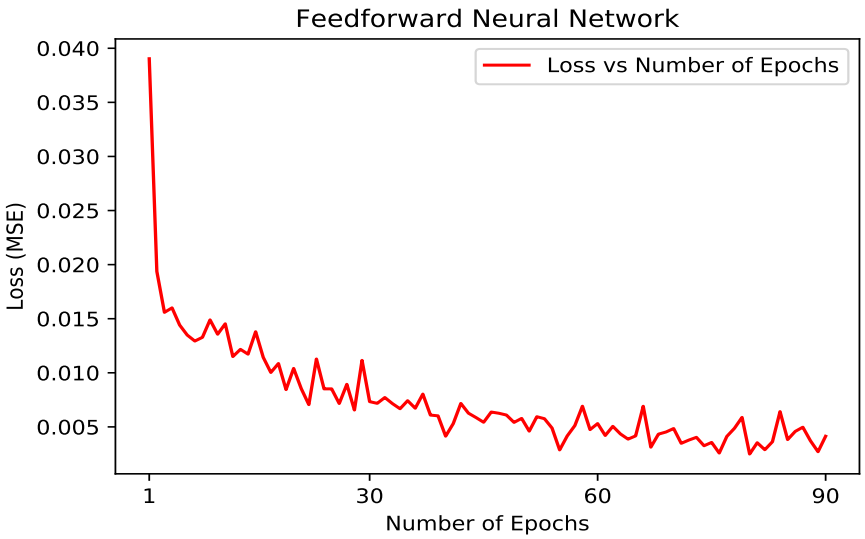
1. **Loss function:** The loss function used is the *Mean Sqaured Error* loss. For this particular dataset, the *MSE* loss function gave a slightly better performance than the *Cross Entropy* loss function for the same number of parameters.
2. **Regularizer:** To prevent overfitting, *Dropout* was used as a regularizer with $p = 0.2$ in the second hidden layer. *Dropout* randomly sets some of the weights to 0 with a parameter, p, determining the probability. Also, *L2* norm regularizer with a *weight decay* of 0.00001 was used to prevent overtraining. The values of these regularizing hyperparameters were decided based on there reactions towards loss value and training accuracy.
3. **Epochs:** The number of max epochs for which the training was done was experimentally set to various values, and 90 seemed to be a reasonable fit.
4. **Optimizer:** We have used the *Adam* optimizer as it seemed to perform reasonably faster and better than *SGD* and *AdaGrad*.
5. **Learning Rate, η:** The initial learning rate for training the model was set to 0.0008, after observing the performance of *Adam* with respect to the loss value.

1.4 Metrics

- **Accuracy:** The max output probabilities of the 10 classes were compared with the true labels of the test dataset. The number of inversions were measured through these comparisons and the accuracy of the model was calculated using:

$$Accuracy = \frac{N_{correct}}{N} \tag{3}$$

- **Loss vs Epochs**: The following plot shows the change in loss as a function of the number of epochs.



- **Confusion Matrix:**

  The confusion matrix for the **FNN** is as follows:

$$C_{FNN} = \begin{bmatrix} 882 & 0 & 18 & 18 & 1 & 2 & 71 & 0 & 8 & 0 \\ 2 & 986 & 2 & 9 & 0 & 0 & 0 & 0 & 1 & 0 \\ 16 & 1 & 886 & 9 & 45 & 0 & 41 & 0 & 2 & 0 \\ 19 & 3 & 12 & 918 & 26 & 0 & 18 & 0 & 4 & 0 \\ 0 & 0 & 72 & 29 & 856 & 0 & 43 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 955 & 0 & 25 & 3 & 17 \\ 126 & 0 & 69 & 28 & 70 & 1 & 696 & 0 & 9 & 1 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 985 & 2 & 11 \\ 5 & 1 & 4 & 4 & 2 & 1 & 3 & 2 & 976 & 2 \\ 0 & 0 & 0 & 0 & 0 & 2 & 1 & 33 & 0 & 964 \end{bmatrix}$$

  In **Section** 2.4, we compare the confusion matrices obtained for the two networks.

1.5 Conclusion

The number of hidden layers used is 3 with 2000 neurons each and $LeakyReLU$ activation with $negative\_slope = 0.01$. The output layer consists of 10 neurons and $softmax$ activation.

The accuracy and final training loss obtained with $\eta = 0.0008$, $Adam$ optimizer and 90 epochs are:

- Accuracy on test data = 90.33%
- Final Training Loss = 0.002703

2 Convolutional Neural Network

2.1 Network Architecture

This network consists of 2 convolutional layers, each followed by a *max pooling* layer. The output of the final max pool layer is given to a fully connected network with 3 layers. The following are the specifications of the network in the order:

1. Convolutional Network
   (i) Conv2d layer 1: 12 5×5 filters with a padding of 2 and $ReLU$ activation.
   (ii) Max Pool layer 1: One 2×2 filter with a stride of 2 on all the 12 filtered images.
   (iii) Conv2d layer 2: 205×5 filters with a no padding and $ReLU$ activation.
   (iv) Max Pool layer 2: One 2×2 filter with a stride of 2 on all the 20 filtered images obtained from the previous convolutional layer.
2. Fully Connected Network

   This part of the network has 5 layers, an *input* receiving layer which receives output from the last layer of the convolutional network, 3 *hidden* layers with $LeakyReLU$ activation and a *Softmax output* layer. Following are the number of neurons in each layer:
   (i) Hidden Layer 1 - 250
   (ii) Hidden Layer 2 - 200
   (iii) Hidden Layer 3 - 84
   (iv) Output Layer (Softmax) - 10

The inputs, consisting of $60,000$ grayscale images, are first segregated into random batches of $1,000$. Each image has a dimension of 28×28, and being grayscale, they have only 1 channel as opposed 3 for color images.

The convolutional network receives these images from a batch as is, applies filters and $ReLU$ activation and passes it on to the fully connected part of the network. The fully connected part first flattens the input to a vector instead of a matrix. This flattened vector is then passed on to the first hidden layer with certain *weights*, as can be seen in the code. Also, a *bias* is applied to each neuron in the hidden layer.

Each neuron, takes a linear combination of the input variables, and applies a *leaky* $ReLU$ activation function to that combination. The negative slope of the activation function is set as a hyperparameter to 0.01. These outputs are then passed on to the deeper layers where similar processing is done. All the weights and biases of all the layers are initially set randomly. The output of each neuron is given as follows:

$$ReLU(x) = max(0, x) \tag{4}$$

$$LeakyReLU(x) = 1_{(x<0)}(0.01x) + 1_{(x\geq 0)}(x) \tag{5}$$

Note that $1_{(x>0)} = 1$ when $x > 0$ and is 0 otherwise.

All these outputs from the last hidden layer are given to the output layer, consisting of 10 neurons, with some weights and bias. Now, the final outputs are probabilities of an input belonging to any one of these 10 classes. Since the leaky ReLU function doesn't satisfy the properties of a probability distribution, the *softmax* function is applied to the *activation outputs* of the hidden layer neurons. The probability of class j for an input x by the *softmax* function is given as:

$$p_j = \frac{e^{-z_j}}{\sum_{i=0}^{3} e^{-z_i}} \tag{6}$$

The 10 softmax activated neurons correspond to the following 10 output classes in which the incoming images have to be classified.

2.2 Why this Architecture?

The popular **LeNet** architecture was initially sought for the network. The network performed reasonably good while also being quite thin and the accuracy achieved on the test network using this network was around 85%.

However, the network was too thin, and with the computing power at hand, we decided to increase the number of parameters of the network. First, the number of filters in the *Conv2d* layers were increased and experimented with. The hyperparameters considered here were the number of filters, size of each filter, stride, padding, activation function and the pooling layer (MaxPool vs AvgPool) followed. The architecture above gave a better performance than **LeNet**, which was expected; and validation accuracy touched 88.5%.

There was still a scope of improvement as we had not touched the fully connected part of the network. Increasing the number of layers in this part and also the number of neurons in each layer resulted in a validation accuracy of around 92%.

2.3 Hyperparameters

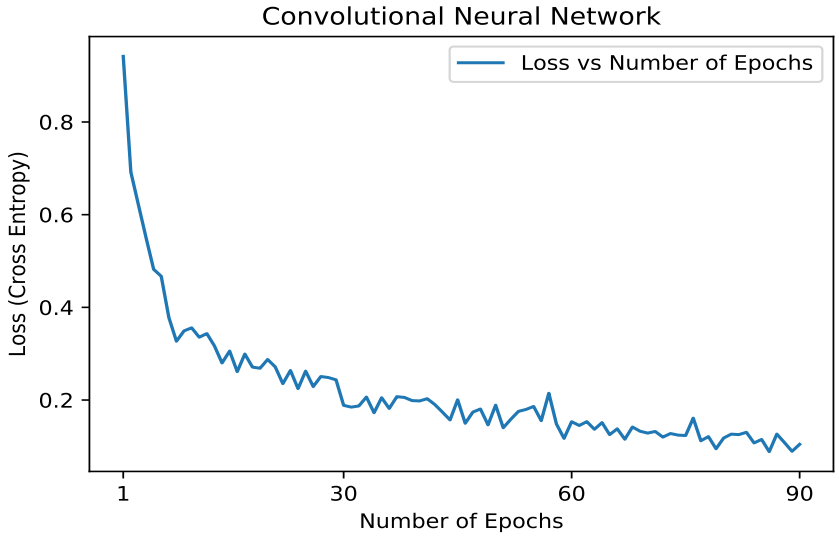
1. **Loss function:** The loss function used is *Cross Entropy*. For this particular dataset and tested with a variety of CNN architectures, the *Cross Entropy* loss function gave a slightly better performance than the *MSE* loss function for the same number of parameters.
2. **Regularizer:** To prevent overfitting, *Dropout* was used as a regularizer with $p = 0.2$ after the first hidden layer of the fully connected part. Also, an *L2* norm regularizer with a *weight decay* of 0.00001 was used to prevent overtraining. The values of these regularizing hyperparameters were decided based on there reactions towards loss value and training accuracy.
3. **Epochs:** The number of max epochs for which the training was done was experimentally set to various values, and 90 seemed to be a reasonable fit.
4. **Optimizer:** We have used the *Adam* optimizer as it seemed to perform reasonably faster and better than *SGD* and *AdaGrad*.
5. **Learning Rate, η:** The initial learning rate for training the model was set to 0.0008, after observing the performance of *Adam* with respect to the loss value.

2.4 Metrics

- **Accuracy:** The max output probabilities of the 10 classes were compared with the true labels of the test dataset. The number of inversions were measured through these comparisons and the accuracy of the model was calculated using:

$$Accuracy = \frac{N_{correct}}{N} \tag{7}$$

- **Loss vs Epochs**: The following plot shows the change in loss as a function of the number of epochs.



- **Confusion Matrix:**

The confusion matrix for the **CNN** is as follows:

$$C_{CNN} = \begin{bmatrix} 873 & 1 & 12 & 14 & 3 & 4 & 89 & 0 & 4 & 0 \\ 1 & 986 & 0 & 6 & 3 & 1 & 2 & 0 & 1 & 0 \\ 28 & 1 & 837 & 9 & 68 & 1 & 53 & 0 & 3 & 0 \\ 17 & 17 & 7 & 902 & 27 & 1 & 24 & 0 & 5 & 0 \\ 2 & 0 & 81 & 22 & 841 & 0 & 52 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 979 & 0 & 17 & 1 & 3 \\ 128 & 2 & 68 & 19 & 67 & 0 & 710 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 & 0 & 963 & 0 & 30 \\ 4 & 0 & 3 & 2 & 0 & 2 & 3 & 3 & 983 & 0 \\ 0 & 0 & 0 & 0 & 0 & 6 & 1 & 23 & 0 & 970 \end{bmatrix}$$

By analyzing C_{CNN} and C_{FNN}, a clear inference one can draw is higher accuracy for the *CNN* model, which is what governs this dataset. The other metrics do not form a big part of this role and hence, can be ignored. Had the the incorrect classifications been penalized differently, the other metrics like *Precision*, *Recall* and *F1 Score* could've been considered.

2.5 Conclusion

The number of hidden layers used is 3 with 2000 neurons each and $LeakyReLU$ activation with $negative\_slope = 0.01$. The output layer consists of 10 neurons and $softmax$ activation.

The accuracy and final training loss obtained with $\eta = 0.0008$, $Adam$ optimizer and 90 epochs are:

- Accuracy on test data = 91.07%
- Final Training Loss = 0.104466