Lab Assignment – 2

Subject: Deep Learning

Improving the Neural Network with Fine-Tuning

Abstract

A neural network is a network or circuit of neurons, or in a modern sense, an artificial neural network, composed of artificial neurons or nodes. Thus a neural network is either a biological neural network, made up of biological neurons, or an artificial neural network, for solving artificial intelligence problems. For our assignment I used Fashion MINST Dataset with 10 classes. We are fine tuning the model by varying Number of epochs, Number of neurons in the hidden layer, batch size, dropout and optimizers to increase the accuracy for the given dataset.

1. Introduction

A neural network is a network or circuit of neurons, or in a modern sense, an artificial neural network, composed of artificial neurons or nodes. The name has been taken from biology term neurons. A biological neural network is composed of a groups of chemically connected or functionally associated neurons. A single neuron may be connected to many other neurons and the total number of neurons and connections in a network may be extensive. Neural networks are a class of machine learning algorithms used to model complex patterns in datasets using multiple hidden layers and non-linear activation functions. A neural network takes an input, passes it through multiple layers of hidden neurons (mini-functions with unique coefficients that must be learned), and outputs a prediction representing the combined input of all the neurons.

We have used Keras library the assignment. Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.

2. Methodology

First we loaded the required python libraries required for our assignment. Keras, pandas and PIL are the required ones. We imported the dataset of Fashion MINST which have 60000 training data and 10000 testing data. We have done total 5 experiments for the assignment. We used sequential model as base model with number of epochs 20. We added two DENSE hidden layers and then we added output layer.

3. Experiments

Experiment 1: Normal Model with 2 Hidden Layers and epoch = 20

We used the base model defined above. Here we have used Activation function as “relu” for hidden layers. In relu Activation function we take value as 0 of the output is less than 0, else we take the output value directly. The output layer activation function is “softmax”. Softmax activation function is used to predict a multinomial probability distribution. As we are having 10 output classes we used Softmax as our output activation function. The size of Hidden layers is 128 and output layer is 10. For loss function we used “sparse categorical crossentropy” and optimizer as RMSprop. RMSprop is a gradient based optimization technique used in training neural networks.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               100480

 activation (Activation)     (None, 128)               0

 dense_1 (Dense)             (None, 128)               16512

 activation_1 (Activation)   (None, 128)               0

 dense_2 (Dense)             (None, 10)                1290

 activation_2 (Activation)   (None, 10)                0

=================================================================
Total params: 118,282
Trainable params: 118,282
Non-trainable params: 0
```

Experiment 2: Variation in Epochs (30, 50) With 2 Hidden Layers

Using the same model defined above we changed the number of epochs to 30 and 50.

No of Epochs = 30

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 128)               100480

 activation_3 (Activation)   (None, 128)               0

 dense_4 (Dense)             (None, 128)               16512

 activation_4 (Activation)   (None, 128)               0

 dense_5 (Dense)             (None, 10)                1290

 activation_5 (Activation)   (None, 10)                0

=================================================================
Total params: 118,282
Trainable params: 118,282
Non-trainable params: 0
_________________________________________________________________
```

No of Epochs = 50

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 128)               100480

 activation_6 (Activation)   (None, 128)               0

 dense_7 (Dense)             (None, 128)               16512

 activation_7 (Activation)   (None, 128)               0

 dense_8 (Dense)             (None, 10)                1290

 activation_8 (Activation)   (None, 10)                0

=================================================================
Total params: 118,282
Trainable params: 118,282
Non-trainable params: 0
```

Experiment 3: Changing No of Neurons (64,256) in Each Layer

Using the best model from experiment 2 we changed the Number of neurons in each layer from initial 128 to 64 and 256.

N_Hidden = 64

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 64)                50240

 activation_9 (Activation)   (None, 64)                0

 dense_10 (Dense)            (None, 64)                4160

 activation_10 (Activation)  (None, 64)                0

 dense_11 (Dense)            (None, 10)                650

 activation_11 (Activation)  (None, 10)                0

=================================================================
Total params: 55,050
Trainable params: 55,050
Non-trainable params: 0
```

N_Hidden = 256

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 256)               200960

 activation_12 (Activation)  (None, 256)               0

 dense_13 (Dense)            (None, 256)               65792

 activation_13 (Activation)  (None, 256)               0

 dense_14 (Dense)            (None, 10)                2570

 activation_14 (Activation)  (None, 10)                0

=================================================================
Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0
_________________________________________________________________
```

Experiment 4: Changing BatchSize (64,256)

Using the best model from experiment 3 we changed the batch size from initial 128 to 64 and 256.

Batch Size = 64

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_15 (Dense)            (None, 256)               200960

 activation_15 (Activation)  (None, 256)               0

 dense_16 (Dense)            (None, 256)               65792

 activation_16 (Activation)  (None, 256)               0

 dense_17 (Dense)            (None, 10)                2570

 activation_17 (Activation)  (None, 10)                0

=================================================================
Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0
```

Batch Size = 256

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_18 (Dense)            (None, 256)               200960

 activation_18 (Activation)  (None, 256)               0

 dense_19 (Dense)            (None, 256)               65792

 activation_19 (Activation)  (None, 256)               0

 dense_20 (Dense)            (None, 10)                2570

 activation_20 (Activation)  (None, 10)                0

=================================================================
Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0
```

Experiment 5: Improving the net with dropout (0.1, 0.2, 0.3)

Using the best model from experiment 4 we added dropout layer by dropping 10 %, 20 % and 30 % units. Dropout means to dropping out units (both hidden and visible) in a neural network. It refers to ignoring units (i.e. neurons) during the training phase of certain set of neurons which is chosen at random. Dropout is used to prevent a model from overfitting.

Dropout = 0.1

Model: "sequential_7"

Layer (type)	Output Shape	Param #
dense_21 (Dense)	(None, 256)	200960
activation_21 (Activation)	(None, 256)	0
dropout (Dropout)	(None, 256)	0
dense_22 (Dense)	(None, 256)	65792
activation_22 (Activation)	(None, 256)	0
dropout_1 (Dropout)	(None, 256)	0
dense_23 (Dense)	(None, 10)	2570
activation_23 (Activation)	(None, 10)	0

Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0

Dropout = 0.2

Model: "sequential_8"

Layer (type)	Output Shape	Param #
dense_24 (Dense)	(None, 256)	200960
activation_24 (Activation)	(None, 256)	0
dropout_2 (Dropout)	(None, 256)	0
dense_25 (Dense)	(None, 256)	65792
activation_25 (Activation)	(None, 256)	0
dropout_3 (Dropout)	(None, 256)	0
dense_26 (Dense)	(None, 10)	2570
activation_26 (Activation)	(None, 10)	0

Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0

Dropout = 0.3

Model: "sequential_9"

Layer (type)	Output Shape	Param #
dense_27 (Dense)	(None, 256)	200960
activation_27 (Activation)	(None, 256)	0
dropout_4 (Dropout)	(None, 256)	0
dense_28 (Dense)	(None, 256)	65792
activation_28 (Activation)	(None, 256)	0
dropout_5 (Dropout)	(None, 256)	0
dense_29 (Dense)	(None, 10)	2570
activation_29 (Activation)	(None, 10)	0

Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0

Experiment 5: Testing with Different Optimizers

Here we have used three optimizers RMSprop, Adam, SGD for fitting our model. RMSprop is a gradient based optimization technique used in training neural networks. Adam is a replacement optimization algorithm for stochastic gradient descent for training deep learning models. Stochastic Gradient Descent (SGD) addresses both of these issues by following the negative gradient of the objective after seeing only a single or a few training examples.

Optimizer = RMSprop

Model: "sequential_10"

Layer (type)	Output Shape	Param #
dense_30 (Dense)	(None, 256)	200960
activation_30 (Activation)	(None, 256)	0
dropout_6 (Dropout)	(None, 256)	0
dense_31 (Dense)	(None, 256)	65792
activation_31 (Activation)	(None, 256)	0
dropout_7 (Dropout)	(None, 256)	0
dense_32 (Dense)	(None, 10)	2570
activation_32 (Activation)	(None, 10)	0

Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0

Optimizer = Adam

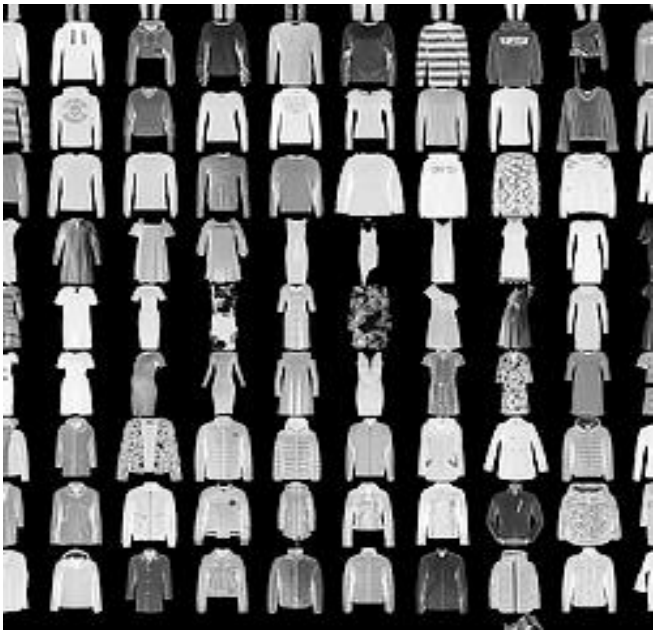
```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_33 (Dense)            (None, 256)               200960

 activation_33 (Activation)  (None, 256)               0

 dropout_8 (Dropout)         (None, 256)               0

 dense_34 (Dense)            (None, 256)               65792

 activation_34 (Activation)  (None, 256)               0

 dropout_9 (Dropout)         (None, 256)               0

 dense_35 (Dense)            (None, 10)                2570

 activation_35 (Activation)  (None, 10)                0

=================================================================
Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0
```

Optimizer = SGD

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_36 (Dense)            (None, 256)               200960

 activation_36 (Activation)  (None, 256)               0

 dropout_10 (Dropout)        (None, 256)               0

 dense_37 (Dense)            (None, 256)               65792

 activation_37 (Activation)  (None, 256)               0

 dropout_11 (Dropout)        (None, 256)               0

 dense_38 (Dense)            (None, 10)                2570

 activation_38 (Activation)  (None, 10)                0

=================================================================
Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0
_________________________________________________________________
```

4. Dataset

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. This image is converted to numpy array of 784 columns.



5. Experiments

Different 6 experiments are done to create best accuracy model and the procedure for all six experiments is defined in the methodology section above. The model is trained on the training dataset and then accuracy is checked using testing dataset. Accuracy is checked after every experiment. The results are described in the results section.

Sr, No.	Model Name	Train Accuracy	Validation Accuracy	Test Accuracy
1.	Simple Model	0.8984	0.8791	0.8881
2.	Epoch = 30	0.9120	0.8811	0.8963
3.	Epoch = 50	0.9296	0.8857	0.8906
4.	N_Hidden = 64	0.8906	0.8822	0.8824
5.	N_Hidden = 256	0.9301	0.9301	0.8913
6.	Batch Size = 64	0.9189	0.8802	0.8776
7.	Batch Size = 256	0.9342	0.8820	0.8899
8.	Dropout = 0.1	0.9155	0.8865	0.8909

9.	Dropout = 0.2	0.9043	0.8865	0.8946
10.	Dropout = 0.3	0.8934	0.8854	0.8888
11.	Optimizer = RMSprop	0.9155	0.8867	0.8945
12.	Optimizer = Adam	0.9269	0.8923	0.9031
13.	Optimizer = SGD	0.8609	0.8545	0.8810

6. Conclusion

When we increased the no of epochs accuracy increased. Also when we increased Number of Neurons and Batch Size accuracy increased. But when we increased Dropout accuracy Decreased. For RMSprop Optimizer we got 0.8945 accuracy, for Adam Optimizer 0.9031 and for SGD Optimizer we got 0.8810 Accuracy.

7. References

[1] https://en.wikipedia.org/wiki/Neural_network

[2] https://www.tensorflow.org/datasets/catalog/fashion_mnist

[3] https://www.geeksforgeeks.org/convert-a-numpy-array-to-an-image/

[4] https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/#:~:text=Adam%20is%20a%20replacement%20optimization,sparse%20gradients%20on%20noisy%20problems.

[5] https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

[6] https://ml-cheatsheet.readthedocs.io/en/latest/nn_concepts.html#:~:text=Neural%20networks%20are%20a%20class,and%20non%2Dlinear%20activation%20functions.&text=Each%20neuron's%20coefficients%20(weights)%20are,contributed%20to%20the%20total%20error.

[7] https://machinelearningmastery.com/gradient-descent-with-rmsprop-from-scratch/#:~:text=Root%20Mean%20Squared%20Propagation%2C%20or,step%20size%20for%20each%20parameter.

[8] https://machinelearningmastery.com/softmax-activation-function-with-python/#:~:text=The%20softmax%20function%20is%20used%20as%20the%20activation%20function%20in,more%20than%20two%20class%20labe