Pattern Recognition
Department of Electrical and Information Engineering
University of Cassino and Southern Latium, Second Semester 2018
Homework Assignment 2
Assigned 23 May 2018; due 11:59 PM, 1 June 2018

In preparing my solutions, I did not look at any old homework's, copy anybody's answers or let them copy mine.

NAME: FAKRUL ISLAM TUSHAR. Signature:

Problem 2.1 [60%] Consider the dataset contained in the hw2data.csv available on the course site. It contains 8,000 samples coming from a two {class problem, each made of 10 numerical features and a binary label (±1). Split the data into training and test sets by randomly selecting 25% of the examples from each class for the test set. For the classification of the test data, use TFLearn to train a Multi-Layer Perceptron with 10 input nodes, 2 hidden layers and 2 output nodes. Consider the following options:

(a) Weight initialization:

(a.1) all the weights set to zero

(a.2) random values from a uniform distribution

(a.3) uniform Xavier distribution

(b) Batch normalization

(b.1) yes

(b.2) no

(c) Activation function for the hidden layers

(c.1) ReLu

(c.2) tanh

Evaluate the accuracy obtained on the test set with the various options and discuss the results obtained. Choose reasonable values for the remaining parameters (e.g. learning rate, batch size, number of epochs,) and keep them fixed during the experiments.

Answer: According to the instructions given considering different options such as (a) weight initialization (b) Batch normalization and (c) Activation functions need to be used to train a Multi-Layer Perceptron of the given configuration. First challenge of the assignment is to find out the optimal hyperparameters such as learning rate, batch size number of epoch.

Optimal hyperparameters were determined by using number of experiments on the architecture Multi-Layer Perceptron (MLP) model asked in question 1. Figure.1, shown the configured architecture of the MLP.

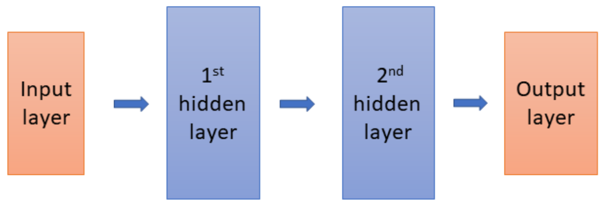

Figure.1: Multi-Layer Perceptron architecture with two hidden layers.

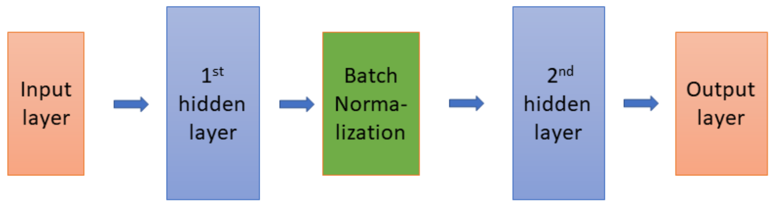

Figure.2: Multi-Layer Perceptron architecture with two hidden layers and batch normalization.

Table.1 and Table.2 shown the experimental configuration for finding out the optimal hyperparameters for the experiments. In these experiments the learning rate is set fixed to 0.003 and the batch size and epoch number is varied with the number of nodes in hidden layers to find out the best hyper parameters.

able 1. Experimental results for 64 nodes.

Number of nodes in hidden layers= 64								
Batch size	64	64	64	64	100	100	100	100
Epoch	100	200	300	500	100	200	300	500
Accuracy	77.28	77.14	80.74	78.45	76.35	77.08	77.35	81.12

Table 2. Experimental results for 256 nodes.

Number of nodes in hidden layers= 256								
Batch size	64	64	64	64	100	100	100	100
Epoch	100	200	300	500	100	200	300	500
Accuracy	74.00	79.14	80.10	80.62	73.97	78.04	79.60	81.00

Taking into consideration computation cost and timing, the optimal hyperparameters are selected as follow.

Learning rate=0.003
Batch Size=64
Epoch=300

After selecting the hypergamies now the experiment was performed based on the options provided in (a),(b) and (c). the pseudocode below shown the general approach followed in the experiments.

General pseudocode for the experiments.

```
   I.  Import necessary libraries
  II.  Load the dataset
 III.  Preprocessing dataset
  IV.      Getting the label -1 to 0
   V.      if the label is -1
  VI.          Set the label=0
 VII.      else
VIII.          Set the label=1
  IX.  Shuffle the data
   X.  Split the data into training and testing set.
  XI.  Initialization of the model
 XII.     Set the input layer
XIII.     Set 1st hidden layer
 XIV.     Set 2nd hidden layer
  XV.     Set SoftMax layer
 XVI.     Set regression
 (VII.     Compile the model
VIII.     Train the model.
```

For the batch normalization asked in the section (b) A batch normalization function is used from TFLearn.

Now all the options are performed and their outcomes are shown below in Table.3 , Table.4 and Table.5

Table.3: (a.1) Weight initialized to zero experiments results.

Case	(b.1) Batch normalization=Yes (c.1) Activation = ReLu		(b.2) Batch normalization=No (c.1) Activation = ReLu		(b.1) Batch normalization=Yes (c.2) Activation = tanh		(b.2) Batch normalization=No (c.2) Activation = tanh	
Hyper parameters	Batch size	64	Batch size	64	Batch size	64	Batch size	64
	Epoch	300	Epoch	300	Epoch	300	Epoch	300
	LR	0.003	LR	0.003	LR	0.003	LR	0.003
	Optimizer	Adam	Optimizer	Adam	Optimizer	Adam	Optimizer	Adam
Accuracy	0.5098		0.4931		0.5042		0.5170	

Table.3 shown the accuracy for the options when the weights were initialized to zero for both the hidden layers. In the all 4 possible cases we can see the accuracy was ±50%. It shows that layers were not learning anything as the weights set zero. As we know learning performed through gradient decent by updating the weights to reduce the cost function.

Table.4: (a.1) Weight initialized to random values from a uniform distribution.

Case	(b.1) Batch normalization=Yes (c.1) Activation = ReLu		(b.2) Batch normalization=No (c.1) Activation = ReLu		(b.1) Batch normalization=Yes (c.2) Activation = tanh		(b.2) Batch normalization=No (c.2) Activation = tanh	
Hyper parameters	Batch size	64	Batch size	64	Batch size	64	Batch size	64
	Epoch	300	Epoch	300	Epoch	300	Epoch	300
	LR	0.003	LR	0.003	LR	0.003	LR	0.003
	Optimizer	Adam	Optimizer	Adam	Optimizer	Adam	Optimizer	Adam
Accuracy	0.8462		0.8288		0.8104		0.8223	

Table 4 shown the accuracy for the options when the Weight initialized to random values from a uniform distribution. Here a big deference between the accuracies can be observed compared to the accuracies that obtained in the Table.2 for weights initialized to zero. Here the accuracy is above 80 because the weights are not randomly initialized by uniform distribution so the nodes in the hidden layers are performing better than weights zero. Another important observation is that ReLu activation of the hidden layers with batch normalization performing fairly better than other cases. This was because compared to tanh, Relu prevent the gradient from saturation by providing non-linear function.

Table.5: (a.1) Weight initialized to uniform Xavier distribution

Case	(b.1) Batch normalization=Yes (c.1) Activation = ReLu		(b.2) Batch normalization=No (c.1) Activation = ReLu		(b.1) Batch normalization=Yes (c.2) Activation = tanh		(b.2) Batch normalization=No (c.2) Activation = tanh	
Hyper parameters	Batch size	64	Batch size	64	Batch size	64	Batch size	64
	Epoch	300	Epoch	300	Epoch	300	Epoch	300
	LR	0.003	LR	0.003	LR	0.003	LR	0.003
	Optimizer	Adam	Optimizer	Adam	Optimizer	Adam	Optimizer	Adam
Accuracy	0.8542		0.8261		0.8069		0.7874	

Table 5 shown the accuracy for the options when the Weight initialized to random values from a uniform Xavier distribution. When we initialized weights, we want to keep zero means and some finite variance. Xavier distribution initialize waits in such a way that the variance remains the same for all the layers and keep the signal from exploding to a high value or vanishing to zero.

If we observed the accuracies in this case with ReLu activation again performing better than tanh. The maximum accuracy obtained is 85.61% which is also maximum for all the cases shown above.

Problem 2.2 [40%] Use the dataset of Problem 2.1 and perform several splits into a training set and a test set (also with di_erent sizes) to determine which combination of options among the ones you considered in Problem 2.1 ensures the best accuracy. Once you picked out the best model, save it by using the method tflearn.models.dnn.save. The saved model must be submitted together with your report and will be run on a separate matrix containing new test data. Your grade will be based on the performance of your classi_er on the new test data, which will contain a very large number of examples generated from the same distribution.

Answer: From the above observation the **Weight initialized to** uniform Xavier distribution configuration giving the best results. So this configuration is picked and Two different splits of training and testing data was performed to see which one perform better.

Tabel.6: 80% for training and 20% for testing data split.

Case	(b.1) Batch normalization=Yes (c.1) Activation = ReLu		(b.2) Batch normalization=No (c.1) Activation = ReLu		(b.1) Batch normalization=Yes (c.2) Activation = tanh		(b.2) Batch normalization=No (c.2) Activation = tanh	
Hyper parameters	Batch size	64	Batch size	64	Batch size	64	Batch size	64
	Epoch	300	Epoch	300	Epoch	300	Epoch	300
	LR	0.003	LR	0.003	LR	0.003	LR	0.003
	Optimizer	Adam	Optimizer	Adam	Optimizer	Adam	Optimizer	Adam
Accuracy	0.8339		0.8362		0.8245		0.8091	

Tabel.7: 70% for training and 30% for testing data split.

Case	(b.1) Batch normalization=Yes (c.1) Activation = ReLu		(b.2) Batch normalization=No (c.1) Activation = ReLu		(b.1) Batch normalization=Yes (c.2) Activation = tanh		(b.2) Batch normalization=No (c.2) Activation = tanh	
Hyper parameters	Batch size	64	Batch size	64	Batch size	64	Batch size	64
	Epoch	300	Epoch	300	Epoch	300	Epoch	300
	LR	0.003	LR	0.003	LR	0.003	LR	0.003
	Optimizer	Adam	Optimizer	Adam	Optimizer	Adam	Optimizer	Adam
Accuracy	0.8476		0.8425		0.8195		0.8287	

From the above experiment it can be seen that 70 training and 30% testing data split providing the best results. This split has been used for the best model. The architecture of the saved model showing bellow here.

Architecture of the saved model.

```
I.     with tf.Graph().as_default():

II.        input_layer = tflearn.input_data(shape=[None, 10], name='input')
           dense1 = tflearn.fully_connected(input_layer, 64,
       activation='relu',weights_init=tflearn.initializations.xavier(),bias_init=tflearn.initializati
       ons.xavier(),regularizer='L2')
III.       network = tflearn.layers.normalization.batch_normalization(dense1)
IV.        dense2 = tflearn.fully_connected(network, 64,
       activation='relu',weights_init=tflearn.initializations.xavier(),bias_init=tflearn.initializati
       ons.xavier(),regularizer='L2')
V.         softmax = tflearn.fully_connected(dense2, 2, activation='softmax')
VI.        regression = tflearn.regression(softmax, optimizer='adam',
                                learning_rate=0.003,
                                loss='categorical_crossentropy')

VII.       model = tflearn.DNN(regression,tensorboard_verbose=0)

VIII.      model.fit(x_train,y_train, show_metric=True, batch_size=64, n_epoch=300,
       snapshot_epoch=False,validation_set=(x_test,y_test))
```

****All the codes attached in the folder uploaded.