Deep Learning Assignment 1

Wednesday, 04.03.2020

Name

Aya Ashraf Saber, 02

Outline

Part 1 Linear Regression

1. Model
2. Hyper-parameters and effect on performance.
3. Adding artificial features
4. Model performance using different loss functions.
5. the effect of using different optimizers and which one produced better Performance.
6. Show if your model is overfitting or under-fitting and explain how can you solve these problems.

Part 2 Logistic Regression

1. Model
2. Code updated
3. Hyper-parameters and effect on performance.
4. Adding artificial features
5. Model performance using different loss functions.
6. the effect of using different optimizers and which one produced better Performance.
7. Show if your model is overfitting or under-fitting and explain how can you solve these problems.

Part 1 Linear Regression :

Problem :

The problem we are trying to solve here is finding a new house which is suitable to our needs and the budget we assigned. The client who wants to buy the new house did her research and found some houses. She wrote the details of each house she visited including location, sale condition, sale type, house price, among others. She needs some help to know how much she is expected to pay to get a house that conforms with her specific needs.

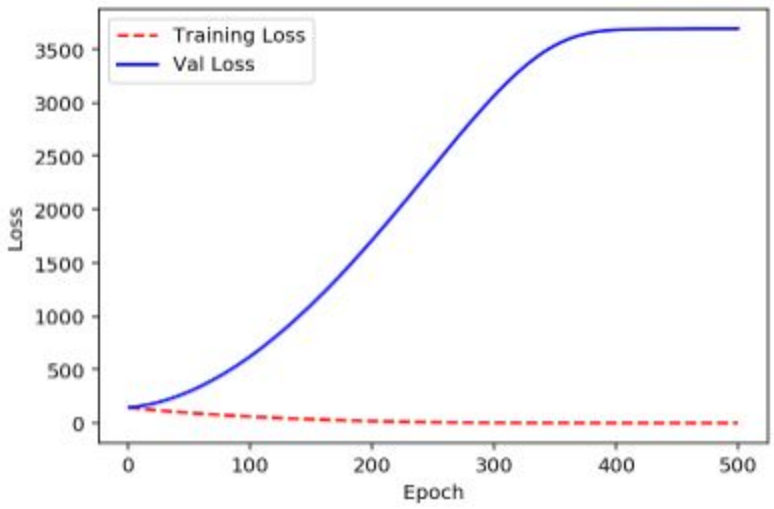
Code:

```
## TODO START CODE HERE, Writing the Model using Tensorflow.Keras is a must
################################################################
# define our MLP network
model = Sequential([Dense(1, input_dim= 288, activation='linear')])

################################################################
```

Evaluation :

Hyper-parameters :

1. Learning Rate
2. Number of Epoc
3. optimizer
4. Activation
5. Loss Function



Model is overfitting

Part 2 Logistic Regression

Problem

We are trying to increase peoples' attention regarding heart diseases. Like any

disease, it is always better to know if you are sick early so you can get the treatment you need before it is too late. Therefore, we use a dataset that gathered some information about two

groups: a group with a heart disease and the other group has no disease.

The gathered information includes age, chest pain type, fasting blood sugar, etc. Your goal is to train a logistic regression model to predict if a person has a heart disease or not depending on the given information.

Code

Normalize X :

```
## TODO START:: Data Pre-Processing

X = preprocessing.normalize(X)

## TODO End:: Data Pre-Processing
```

Build Model :

```
# create different Models
model2 = Sequential()
model2.add(Dense(1, input_dim= 13, kernel_initializer= 'random_normal',activation='linear'));
model2.add(Dense(1, activation= 'relu'));
models.append(model2)
## TODO END:: Model Definition
```

Code For Parameter Tuning :

```
## TODO START:: Model Definition, Writing the Model using Tensorflow.Keras is a must
models = []
activations = ['relu', 'tanh', 'sigmoid', 'hard_sigmoid']
intializers = ['random_uniform', 'random_normal']

# create different Models
for i in activations:
  for j in intializers:
    model2 = Sequential()
    model2.add(Dense(1, input_dim= 13, kernel_initializer= j,activation='linear'));
    model2.add(Dense(1, activation= i));
    models.append(model2)
## TODO END:: Model Definition
```

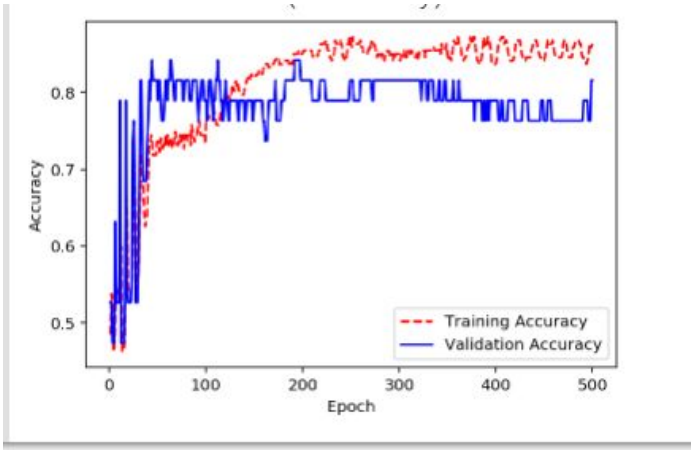
```
## TODO Try Different losses & optimizers here
learning_rates= [0.5,0.1,.01,.001]
beta = [0.9,0.999,0.855]
momentums = [0.9,0.95,0.7]
epoch = [50,100,200]
loss_functions = [ 'binary_crossentropy','mse','mean_absolute_error', 'categorical_hinge']

optimizers = []
scores = []
accuracies = []
# generate optimizer type Adam with different alpha
for i in learning_rates:
  for j in beta:
    for k in beta:
      optimizers.append(Adam(lr = i,  beta_1=j , beta_2 = k))

# generate optimizer type Streaming GD  with differebt alpha and momentum
for i in learning_rates:
  for j in momentums:
       optimizers.append(SGD(lr = i, momentum =j))

# generate optimizer type RMSprop with different alphav
for i in learning_rates:
       optimizers.append(RMSprop(lr = i))
```

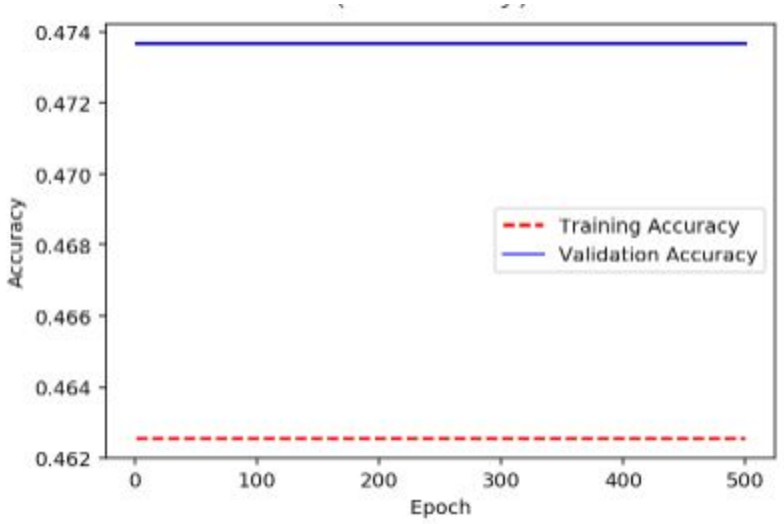
```
#genretate optimizer type Adagrad with different alphav
for i in learning_rates:
      optimizers.append(Adagrad(lr = i))

for model2 in models:
  for i in loss_functions:
   for j in optimizers:
      model2.compile(loss=i, metrics=['accuracy'], optimizer=j)
      hist2 = model2.fit(train_X, train_y, verbose=1, validation_data=(val_X, val_y), batch_size=16, epochs=250)
      score2, accuracy2 = model2.evaluate(test_X, test_y, batch_size=16, verbose=0)
      scores.append(score2)
      accuracies.append(accuracy2)

      print("Test fraction correct (NN-Score) = {:.2f}".format(score2))
      print("Test fraction correct (NN-Accuracy) = {:.2f}".format(accuracy2))
      # Get training and test loss histories
      training_loss2 = hist2.history['accuracy']
      val_loss2 = hist2.history['val_accuracy']

      # Create count of the number of epochs
      epoch_count2 = range(1, len(training_loss2) + 1)
      # Visualize loss history
      plt.figure()
      plt.plot(epoch_count2, training_loss2, 'r--')
      plt.plot(epoch_count2, val_loss2, 'b-')
      plt.legend(['Training Accuracy', 'Validation Accuracy'])
      plt.xlabel('Epoch')
      plt.ylabel('Accuracy')
      plt.show()
```

Hyper-parameters :

6. Learning Rate
7. Number of Epoc
8. optimizer
9. Activation
10. Loss Function

#	LR	# Epoc	Activates	Optimizer	Intializers	Loss functions	Accuracy	Score
1	0.01	500	Relu	Adam	random_normal	mse	0.87	0.12
2	0.001	500	Relu	Adam	random_normal	mse	0.61	0.39
3	0.1	500	Relu	Adam	random_normal	mse	0.61	0.39




0.474
0.472
0.470
0.468
0.466
0.464
0.462
Accuracy
0
100
200
300
400
500
Epoch
Training Accuracy
Validation Accuracy

Resources