Stevens Institute of Technology

Department of Electrical Engineering

EE628: Data Acquisition/Proc II

Homework Assignment 4

Submitted by:
Hadia Hameed
CWID: 10440803

Submitted to:
Prof. Rensheng Wang

February 28, 2019

1 Part 1: Compare the different compiler setups

1.1 Understanding the data:

Dataset of 25,000 movies reviews from IMDB, labeled by sentiment (positive or negative). Reviews have been preprocessed, and each review is encoded as a sequence of word indexes (integers). For convenience, words are indexed by overall frequency in the dataset, so that for instance the integer '3' encodes the 3rd most frequent word in the data. If the *num_words* argument was specific, the maximum possible index value is $num\_words - 1$ [1].

As shown in the figure below, the first review has 218 high frequency words out of the 10,000 words in the bag of words created initially while loading the data. It has the 1st encoded word, the 14th encoded word, the 22nd encoded word and so on.

```
(x_train,y_train),(x_test,y_test) = imdb.load_data(num_words=10000)
```

```
x_train.shape
```

```
(25000,)
```

```
review_1 = x_train[0]
print("Number of high frequency words in the first review = ",len(review_1))
for x in review_1:
    print(x, end=" ")
```

```
Number of high frequency words in the first review =  218
1 14 22 16 43 530 973 1622 1385 65 458 4468 66 3941 4 173 36 256 5 25 100 43 838 112 50 670 2 9 35 480 284 5 150 4 17
2 112 167 2 336 385 39 4 172 4536 1111 17 546 38 13 447 4 192 50 16 6 147 2025 19 14 22 4 1920 4613 469 4 22 71 87 12
16 43 530 38 76 15 13 1247 4 22 17 515 17 12 16 626 18 2 5 62 386 12 8 316 8 106 5 4 2223 5244 16 480 66 3785 33 4 13
0 12 16 38 619 5 25 124 51 36 135 48 25 1415 33 6 22 12 215 28 77 52 5 14 407 16 82 2 8 4 107 117 5952 15 256 4 2 7 3
766 5 723 36 71 43 530 476 26 400 317 46 7 4 2 1029 13 104 88 4 381 15 297 98 32 2071 56 26 141 6 194 7486 18 4 226 2
2 21 134 476 26 480 5 144 30 5535 18 51 36 28 224 92 25 104 4 226 65 16 38 1334 88 12 16 283 5 16 4472 113 103 32 15
16 5345 19 178 32
```

The train_data can also be one-hot encoded, showing the presence of or absence of a certain word from the 10,000 long bag of words.

```
import numpy as np
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results
```

```
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

```
review_1 = x_train[0]
review_1[0:10]
```

```
array([0., 1., 1., 0., 1., 1., 1., 1., 1., 1.])
```

It is binary classification. Train label 1 is for positive review and train label 0 is for negative review. The figure shows train labels for the first 20 reviews.

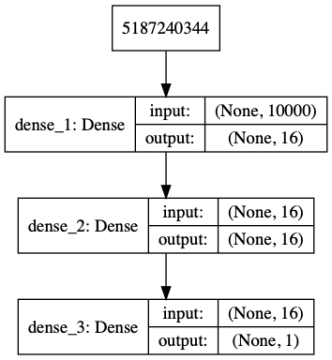
```
train_labels[0:20]
```

```
array([1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1])
```


1.2 Neural Network

```
#Creating Sequential model for a neural network
from keras import models
from keras import layers
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
from keras.utils.vis_utils import plot_model
plot_model(model, to_file = 'model_plot.png', show_shapes=True,show_layer_names=True)
```



1.3 Evaluating performance

```
from keras import optimizers
from keras import losses
from keras import metrics

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss=losses.binary_crossentropy,
              metrics=[metrics.binary_accuracy])
```

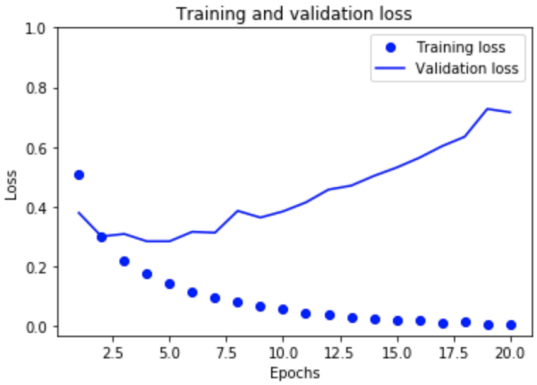
```
#select first 10000 rows
x_val = x_train[:10000]
#leave first 10000 rows
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```



```
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
acc_values = history_dict['acc']
epochs = range(1, len(acc_values) + 1)
plt.plot(epochs, loss_values, 'bo', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.ylim(top=1)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



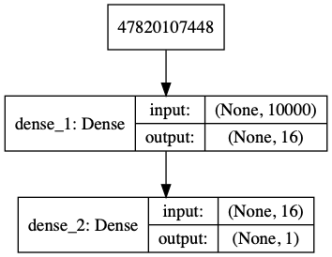
As the number of epochs (iterations) increases, the training loss decreases as the model learns repeatedly from the training data. The model "memorizes" the trends in the training data. However, with more epochs, the variance of the model increases such that it gives poor performance on the validation set and therefore, the validation set loss increases.

1.4 Changing the number of hidden layers:

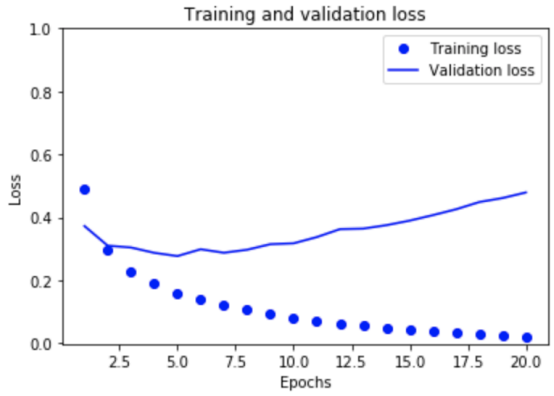
1.4.1 One hidden layer:

With a single hidden layer having 16 neurons and an activation function ''ReLu', the validation set loss has reduced by almost 35% with the same number of epochs.

```
#Creating Sequential model for a neural network
from keras import models
from keras import layers
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(1, activation='sigmoid'))
```

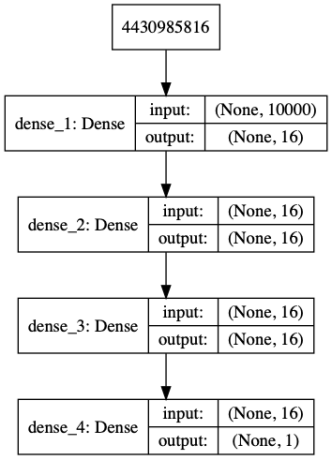



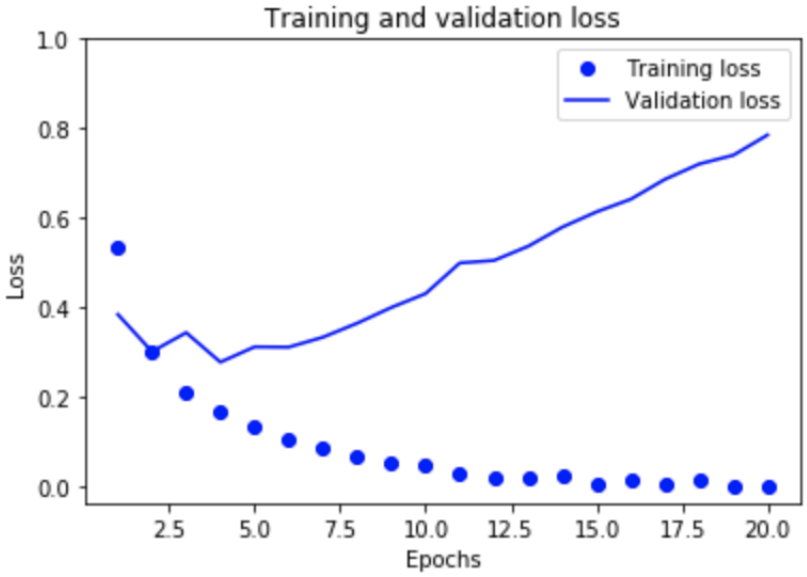

```
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
acc_values = history_dict['val_binary_accuracy']
epochs = range(1, len(acc_values) + 1)
plt.plot(epochs, loss_values, 'bo', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.ylim(top=1)
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



1.4.2 Three hidden layers:

```
#Creating Sequential model for a neural network
from keras import models
from keras import layers
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```



As the neural network becomes deeper with three hidden layers, there are more number of parameters and the model becomes overfit. It performs well on the training data but losses on the validation set increase and the model becomes less generalized.

Number of hidden layers	Crossentropy Loss (epoch_size = 20)
1	0.49
2	0.70
3	0.78

Figure 1: Change in validation set loss with increase in the number of hidden layers. Each layer had 16 hidden units.

1.5 Changing the number of hidden units:

For a single hidden layer, as the number of neurons is increased from 16 to 64, the cross entropy loss increases for the validation set (the model becomes less generalized). This is because as the number of neurons increases, the number of parameters also increases and the model becomes more over fit and performs poorly on the validation set.

For two hidden layers, the loss is the least when 16 neurons are used per layer. It increases when the first layer has 32 neurons and the second layer has 16 neurons. It slightly increases further when both layers have 32 neurons. The loss shows similar increasing trend when the number of neurons are increased to 64. This testifies that increasing the number of parameters leads to overfitting.

Number of hidden layers	layer1_neurons	layer2_neurons	layer3_neurons	Crossentropy Loss (epoch_size = 20)
1	16	-	-	0.4584
	32	-	-	0.5705
	64	-	-	0.6401
2	16	16	-	0.6190
	32	16	-	0.7452
	32	32	-	0.7486
	64	16	-	0.7914
	64	32	-	0.7753
3	16	16	16	0.7196
	32	16	16	0.7513
	32	32	16	0.8070
	32	32	32	0.8621
	64	32	16	0.9724

1.6 Changing loss function and activation function:

Number of hidden layers	No. of neurons in each layer	Activation Function	Crossentropy Loss (epoch_size = 20)	MSE (epoch_size. =20)
1	16	ReLu	0.4584	0.1020
		Tanh	0.5069	0.1080
2	16	ReLu	0.7452	0.1102
		Tanh	0.8320	0.1269
3	16	ReLu	0.7196	0.1105
		Tanh	0.7664	0.1217

The results show that ReLu performs better than Tanh as an activation function as it reduces the loss.

2 Examine the impact of regularization and dropout

2.1 Adding regularization

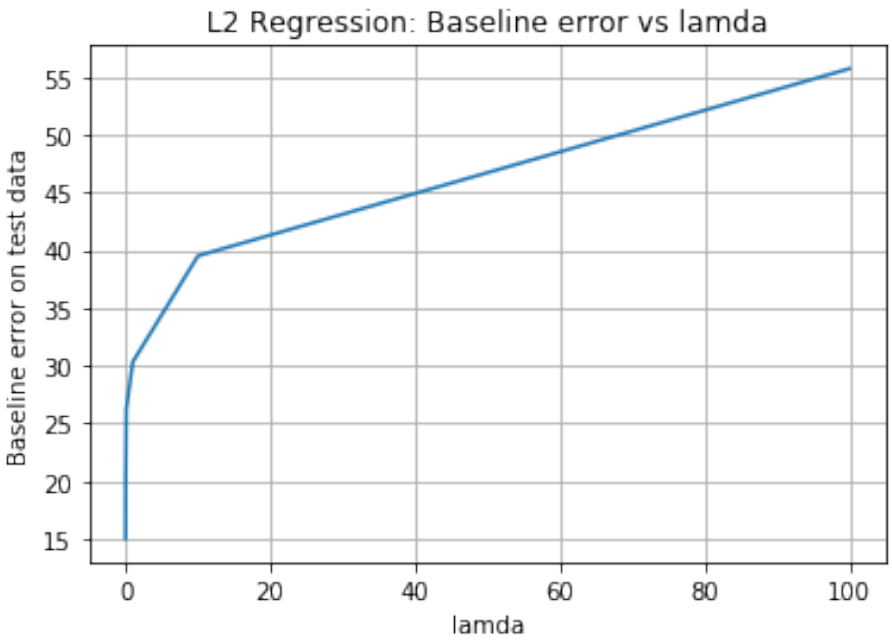
Regularizers allow to apply penalties on layer parameters or layer activity during optimization. These penalties are incorporated in the loss function that the network optimizes. The penalties are applied on a per-layer basis [2].

2.1.1 L2 weight regularization penalty/ weight decay/Ridge

Ridge regression adds "squared magnitude" of coefficient as penalty term to the loss function. Here the highlighted part represents L2 regularization element [3].

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Cost function



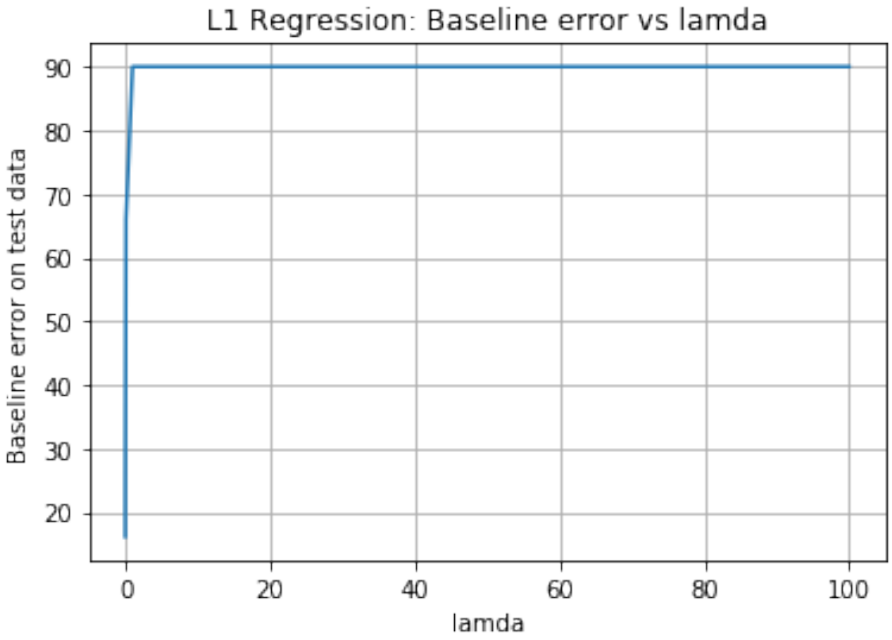
L2 (Ridge) Regression	
Lambda	**Baseline Error**
0.0001	15.03%
0.001	15.69%
0.01	19.89%
0.1	26.25%
1	30.34%
10	39.52%
100	55.73%

2.1.2 L1 weight regularization penalty/ Lasso

Lasso Regression (Least Absolute Shrinkage and Selection Operator) adds "absolute value of magnitude" of coefficient as penalty term to the loss function [3].

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

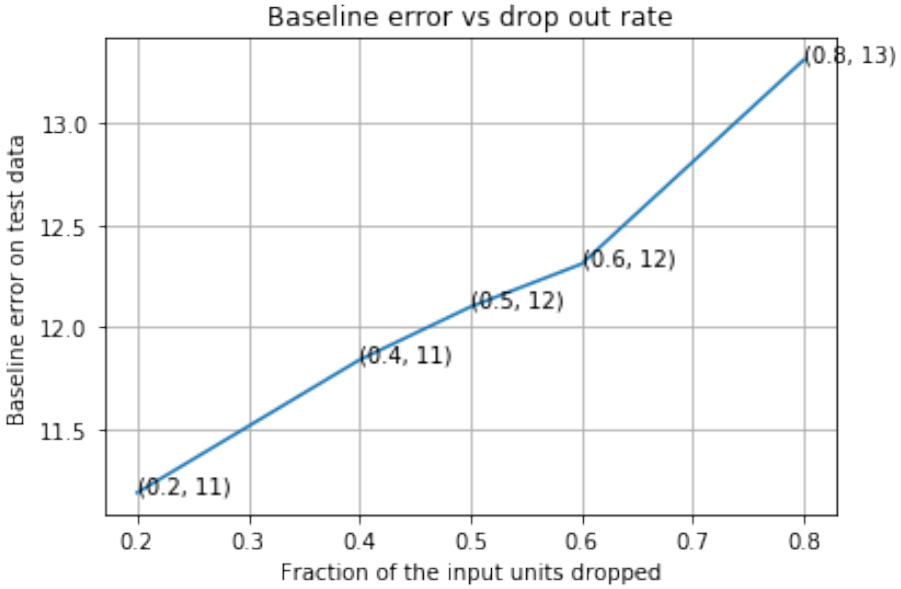
Cost function



L1 (Lasso) Regression	
Lambda	**Baseline Error**
0.0001	16.19%
0.001	20.67%
0.01	27.07%
0.1	63.57%
1	90.00%
10	90.00%
100	90.00%

2.2 Performing drop out

As more input units are dropped, the model get less information for each instance of data and hence the baseline error increases. Drop out technique works better than regularization as the mean baseline error for different drop-out rates is lower than that for different values of regularization parameter λ.



References

[1] K. Documentation. Datasets. https://keras.io/datasets//, 2019. [Online; accessed 22-February-2019].

[2] K. Documentation. Datasets. https://keras.io/regularizers/, 2019. [Online; accessed 22-February-2019].

[3] A. Nagpal. L1 and L2 Regularization Methods, Towards Data Science. https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c, 2017. [Online; accessed 27-February-2019].

