Allison Roeser

IMDB Movie Review: Language Modeling with an RNN

Data Preparation, Exploration, Visualization:

The IMDB movie review dataset contains 1,000 reviews (500 positive and 500 negative) formatted as text files. The median length per review is 924 characters (with an interquartile range of 682 to 1504 characters) as shown in Figure 1. On average, positive reviews are longer with a median length of 937 characters and wider variance in length (IQR of 669 to 1561), while negative reviews have a median length of 912 characters (IQR of 694 to 1442) as shown in Figure 2. Figure 3 illustrates that the distributions for review length are right skewed for both positive and negative reviews. The mean is greater than the median.

Figure 4 shows a sample negative review. The negative words are colored red and the positive words are colored green. Although this review is negative, it includes many positive words such as “good”, “funny”, “glowing”, “inventive”, and “touching”. These positive words must be taken in context, which is difficult for a machine learning program to do. Figure 5 shows a positive review with the words colored red and green. There are negative words such as “dreadful”, “disappointed”, “convoluted”, and “nonsense” in this positive review.

Glove word vectors are used for text processing and sentiment analysis. Two glove vectors (glove.6B.50d and glove.6B.300d) were used in this analysis. Figure 6 shows the first 200 words of the glove.6B.50d vectors after they have been reduced from fifty dimensions to two dimensions using t-SNE. Words with closely related meanings have similar vectors and are shown together. For instance, “war”, “military”, “security”, and “police” are shown together. Figure 7 displays the glove.6B.300d vectors reduced to two dimensions.

Implementation and Programming:

Full code is attached in the Appendix to show the specific implementation. The TensorFlow Keras package was leveraged to create four different RNN models for text sentiment analysis and classification. The goal was to determine if a movie review text string represented a positive review or a negative review by using pretrained word vectors (glove.6B.50d and glove.6B.300d) for the analysis.

Confusion matrixes to were used to evaluate the classification accuracy of potential models. Learning curve plots were used to illustrate the improvement in training and validation set accuracy as the number of epochs used in modeling increased. Pandas and Seaborn were leveraged for exploratory data analysis and visualization.

Review Research Design and Modeling Methods:

Four Neural Networks were created using the TensorFlow Keras API. The design of the experiment was to determine how differing the vocabulary size and the word vector impacted processing time and training / validation / testing accuracy. The model structures were: 10,000-word vocabulary with glove.6B.50d. vector, 20,000-word vocabulary with glove.6B.50d. vector, 10,000-word vocabulary with glove.6B.300d. vector, and 20,000-word vocabulary with glove.6B.300d. vector. Time to fit each model was also measured in order to weight the tradeoff between improved accuracy from more complex models with the longer processing times generally required. Training, validation, and test sets were utilized to verify that the models generalized well.

Each model was run for 10 epochs. Learning curve plots were created to illustrate how the training and validation accuracy improved with each epoch. Confusion matrixes were created for the training and validation sets to understand how well each model classified the positive and negative movie review text strings.

Review Results, Evaluate Models:

Model Number	Vocab Size	Word Vector	Processing Time (Sec)	Training Accuracy	Validation Accuracy	Testing Accuracy
1	10,000	glove.6B.50d	50.47	0.847	0.744	0.695
2	20,000	glove.6B.50d	56.24	0.853	0.750	0.695
3	10,000	glove.6B.300d	91.26	0.991	0.819	0.755
4	20,000	glove.6B.300d	118.54	0.991	0.781	0.705

The chart above displays the results of the four tests. All four neural networks were highly successful in correctly identifying if the sentiment of the review was positive or negative. Validation and test set accuracy scores were lower than training score accuracy, showing that the model was overfit to the training data. The most accurate model on test and validation data was Model 3, which used a 10,000-word vocabulary and the glove.6B.300d word vector. Specificity, Positive Predictive Value, and Negative Predictive Value were highest for Model 3. Sensitivity was highest in Model 4;however, Model 4 did not perform as well on the other measures.

Model Number	Vocab Size	Word Vector	Sensitivity	Specificity	Positive Predictive Value	Negative Predictive Value
1	10,000	glove.6B.50d	0.763	0.631	0.661	0.739
2	20,000	glove.6B.50d	0.773	0.621	0.658	0.744
3	10,000	glove.6B.300d	0.784	0.728	0.731	0.781
4	20,000	glove.6B.300d	0.794	0.621	0.664	0.762

Figures 8 displays detailed information for Model 3, including the number of layers and the various hyperparameters used. Figure 9 includes the learning curve plots for Model 3 by epoch. The accuracy scores improved significantly thru the sixth epoch and then began to plateau. The confusion matrices for training and testing data for Model 3 are shown in Figures 10 & 11.

Exposition, Problem Description and Management Recommendations:

RNNs are a fantastic, quick way to determine the sentiment of a text string with reasonable certainty. Model 3 had a testing accuracy score of 75.5%. The customer service team should go about implementing a model similar to Model 3. Although it will not catch every case of poor customer sentiment, the quick model will be much better than doing nothing.

If higher accuracy is required a custom word vector could even be used to code industry specific words as having negative sentiment in this scenario. These words could include “manager”, “refund”, “delay”, “broken”, and “return”. Real time modeling of any conversation over a specified time (such as three minutes) could be immediately run through a sentiment analysis model and flag a manager if negative sentiment is detected. Sentiment analysis could also be used to see which words and phrases an agent or manger has used to successfully turn a negative call into a positive call. The sentiment of a call could be determined each minute (rather than one sentiment for the entire call) to see how customer sentiment changes over time.

References

Ahmed, S. (Jan 13, 2018). Text Classification Using CNN, LSTM and Pre-trained Glove Word Embeddings: Part-3 https://medium.com/@sabber/classifying-yelp-review-comments-using-cnn-lstm-and-pre-trained-glove-word-embeddings-part-3-53fcea9a17fa

Theiler, S. (Sep 7, 2019). Basics of Using Pre-trained GloVe Vectors in Python. Medium. Retrieved from: https://medium.com/analytics-vidhya/basics-of-using-pre-trained-glove-vectors-in-python-d38905f356db

Appendix

Figure 1: Length of movie review descriptive statistics

```
count    1000.00
mean     1248.54
std       947.74
min       123.00
25%       682.00
50%       924.50
75%      1504.25
max      6309.00
Name: length, dtype: float64
```

Figure 2: Length of movie review descriptive statistics by label (0 = negative / 1 = positive)

	length							
	count	mean	std	min	25%	50%	75%	max
Label								
0	500.0	1231.712	898.084830	123.0	693.75	911.5	1442.25	6037.0
1	500.0	1265.360	995.530808	136.0	668.50	936.5	1560.50	6309.0

Figure 3: Distribution of move review length by label (0 = negative / 1 = positive)

```
Label
0    [[AxesSubplot(0.125,0.125;0.775x0.755)]]
1    [[AxesSubplot(0.125,0.125;0.775x0.755)]]
dtype: object
```

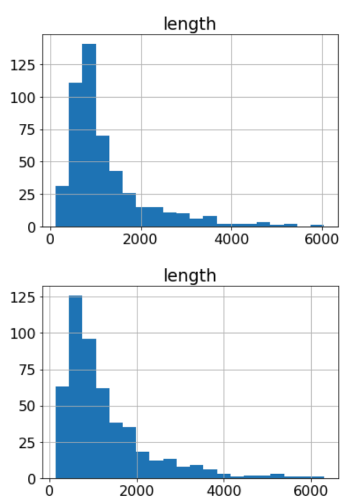


Figure 4: Negative review with negative words in red and positive words in green

While some performances were **good**-Victoria Rowell, Adrienne Barbeau, and the two Italian girlfriends come to mind-the story was **lame** and **derivative**, the emphasis on the girlfriend's racial background was handled **clumsily**, at best, and the relatives were mostly portrayed as **stereotypes**, not as real people. I found myself **wincing uncomfortably** at many moments that were supposed to be **funny**. I can hardly comprehend why the local paper here in SF said this was a **good** movie, and wonder WHO posted the **glowing** review here on IMDB. Very **disappointed** in this movie, and **mad** I actually went to a theatre to see it, based on the **faulty** connection to Garden State, which is a far **funnier**, more **inventive**, and **touching** movie than this one. I must especially mention the emotional climax in the church, which was so **wooden** and by-the-numbers that I nearly **left**, and some in the audience actually DID. That was followed by a **silly** climax at the graveyard, which I saw coming 10 minutes before it happened. I really don't like being **misled** to spend my money so **uselessly.**

Figure 5: Positive review with negative words in red and positive words in green

'When I saw the **elaborate** DVD box for this and the **dreadful** Red Queen figurine, I felt certain I was in for a big **disappointment**, but **surprise, surprise**, I **loved** it. **Convoluted nonsense** of course and **unforgivable** that such a **complicated** denouement should be **rushed** to the point of **barely** being able to read the subtitles, let alone take in the **ridiculous** explanation. These quibbles apart, however, the film is a **dream. Fabulous** ladies in **fabulous** outfits in **wonderful** settings and the whole thing constantly on the move and accompanied by a **wonderful** Bruno Nicolai score. He may not be Morricone but in these lighter pieces he might as well be so. Really **enjoyable** with lots of **color**, **plenty** of **sexiness**, some gory kills and minimal police interference. **Super**.'

Figure 6: Glove.6B.50d. words projected in 2 dimensions (Theiler 2019)

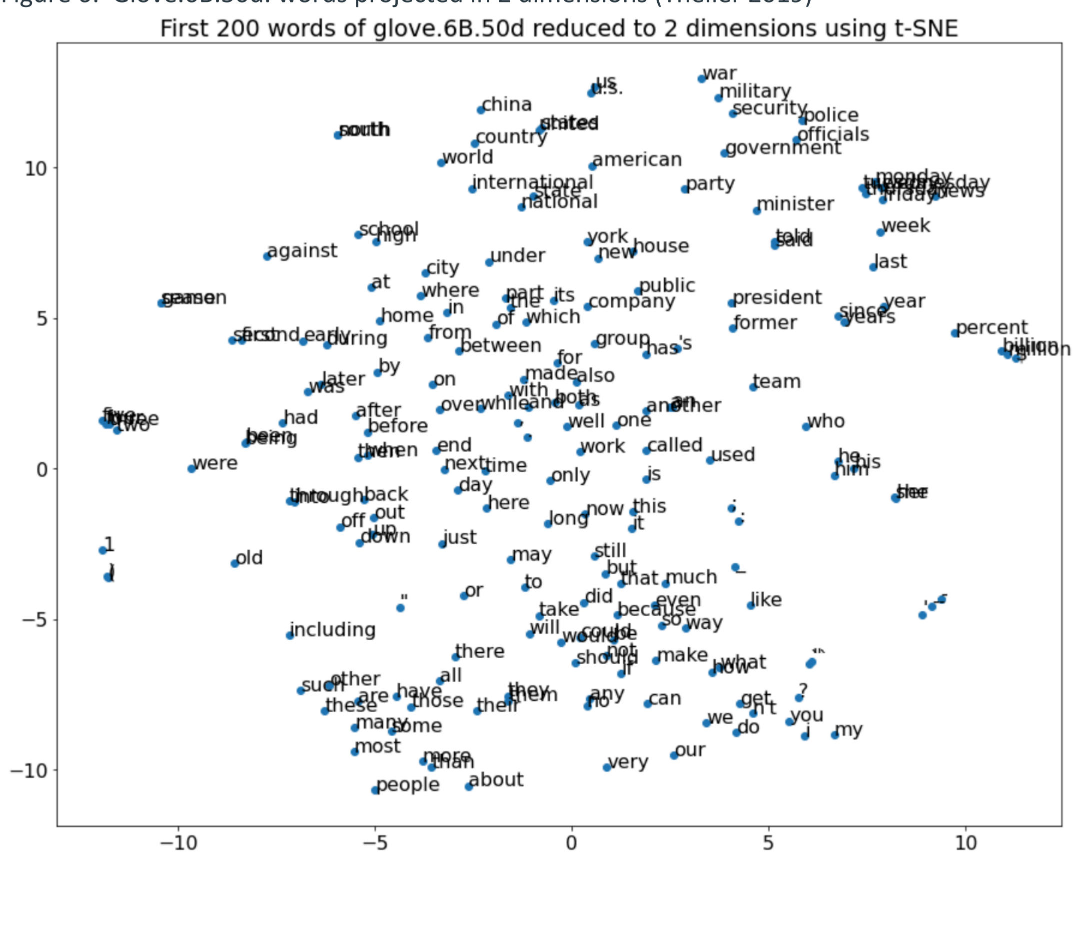


Figure 7: Glove.6B.300d. words projected in 2 dimensions (Theiler 2019)

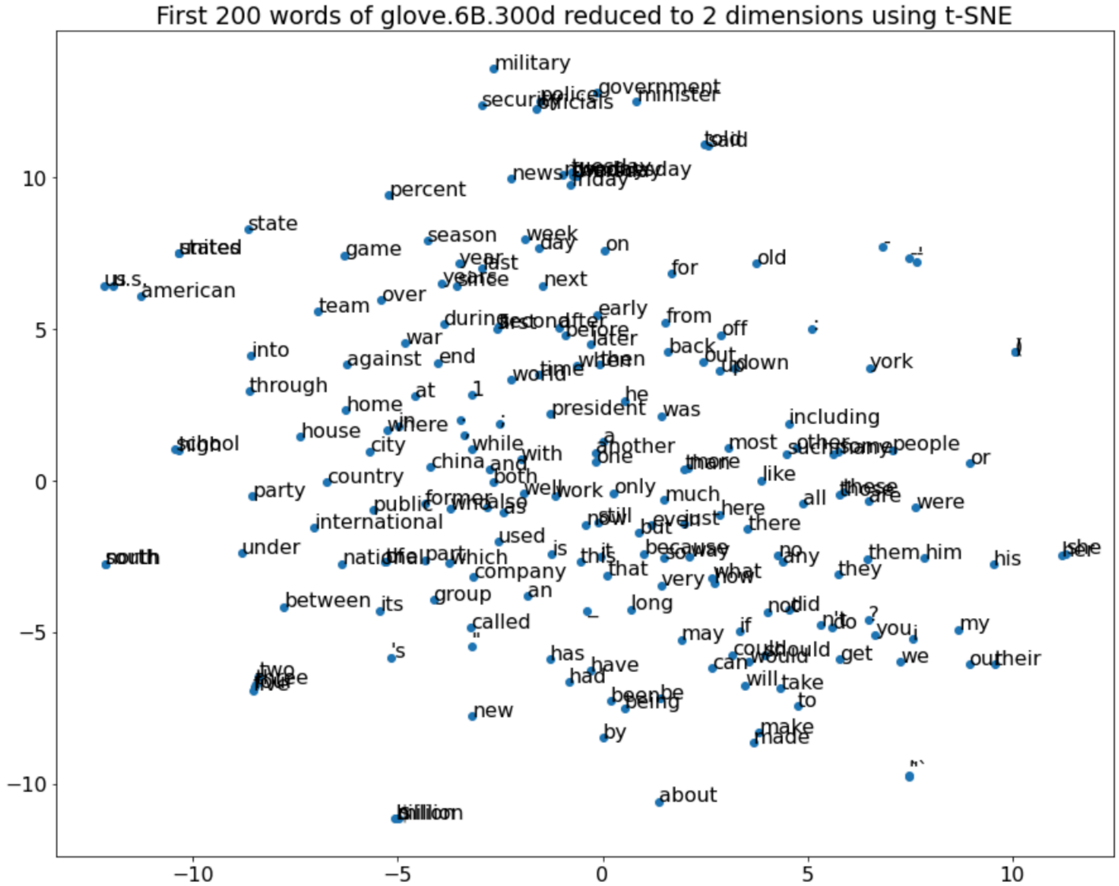


Figure 8: Model 3 summary

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_13 (Embedding)     (None, None, 300)         3000000
_________________________________________________________________
lstm_13 (LSTM)               (None, 128)               219648
_________________________________________________________________
dense_13 (Dense)             (None, 1)                 129
=================================================================
Total params: 3,219,777
Trainable params: 3,219,777
Non-trainable params: 0
```

Figure 9: Learning curve plots (Model 3)

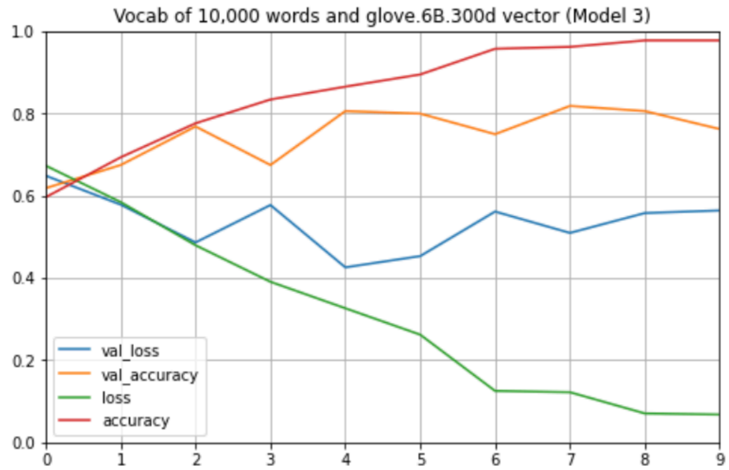


Figure 10: Confusion matrix for training data (Model 3)

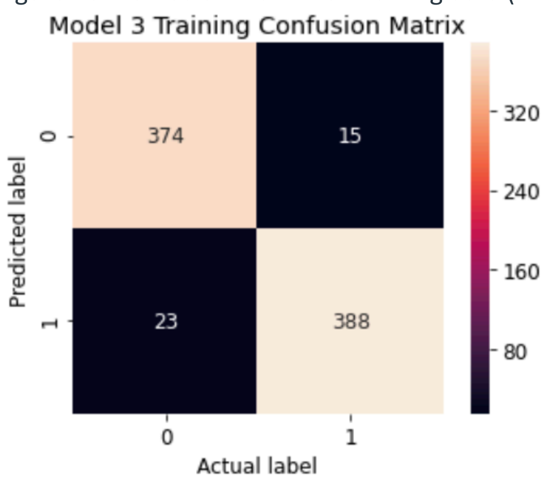
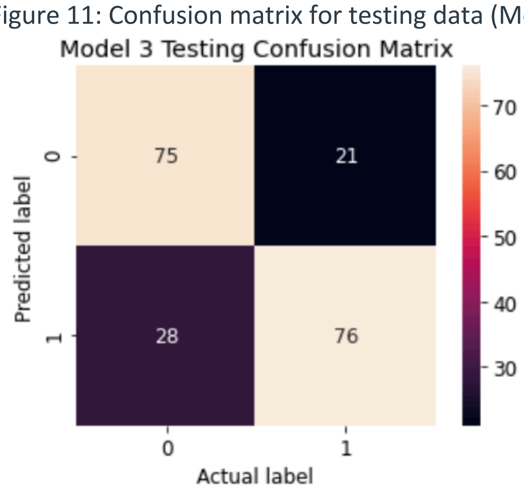


Figure 11: Confusion matrix for testing data (Model 3)



https://medium.com/analytics-vidhya/basics-of-using-pre-trained-glove-vectors-in-python-d38905f356db (https://medium.com/analytics-vidhya/basics-of-using-pre-trained-glove-vectors-in-python-d38905f356db)

In [1]:
```
import numpy as np
from scipy import spatial
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
import chakin
import os
import pandas as pd
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Embedding, SimpleRNN, Dense, Flatten, LSTM, Conv1D
from keras.preprocessing.text import Tokenizer
from sklearn.metrics import roc_auc_score, confusion_matrix
import seaborn as sns
import time
```

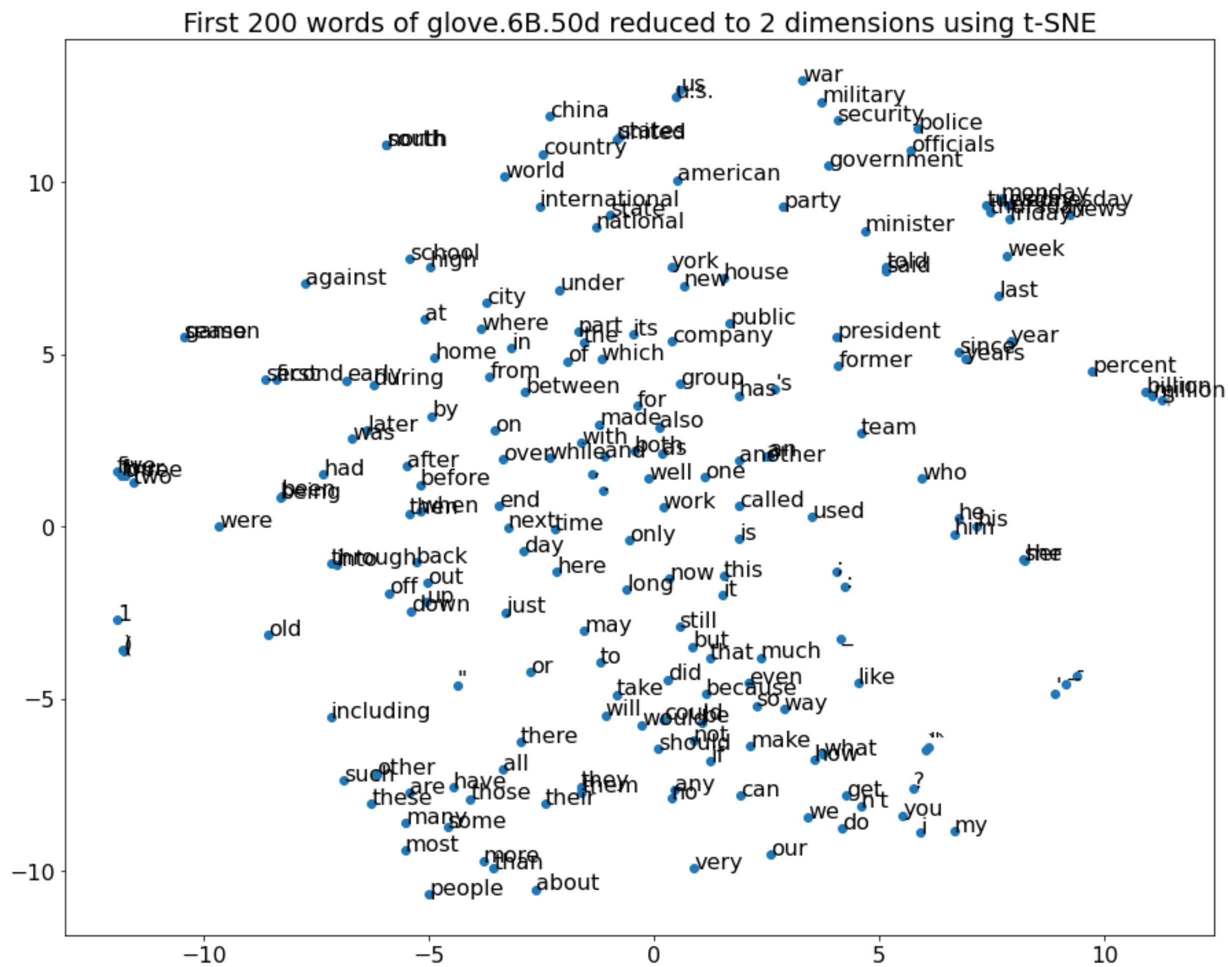
```
Using TensorFlow backend.
```

In [2]:
```
embeddings_index_50 = dict()
f = open("glove.6B.50d.txt")
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index_50[word] = coefs
f.close()
```

In [3]:
```
embeddings_index_300 = dict()
f = open("glove.6B.300d.txt")
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index_300[word] = coefs
f.close()
```

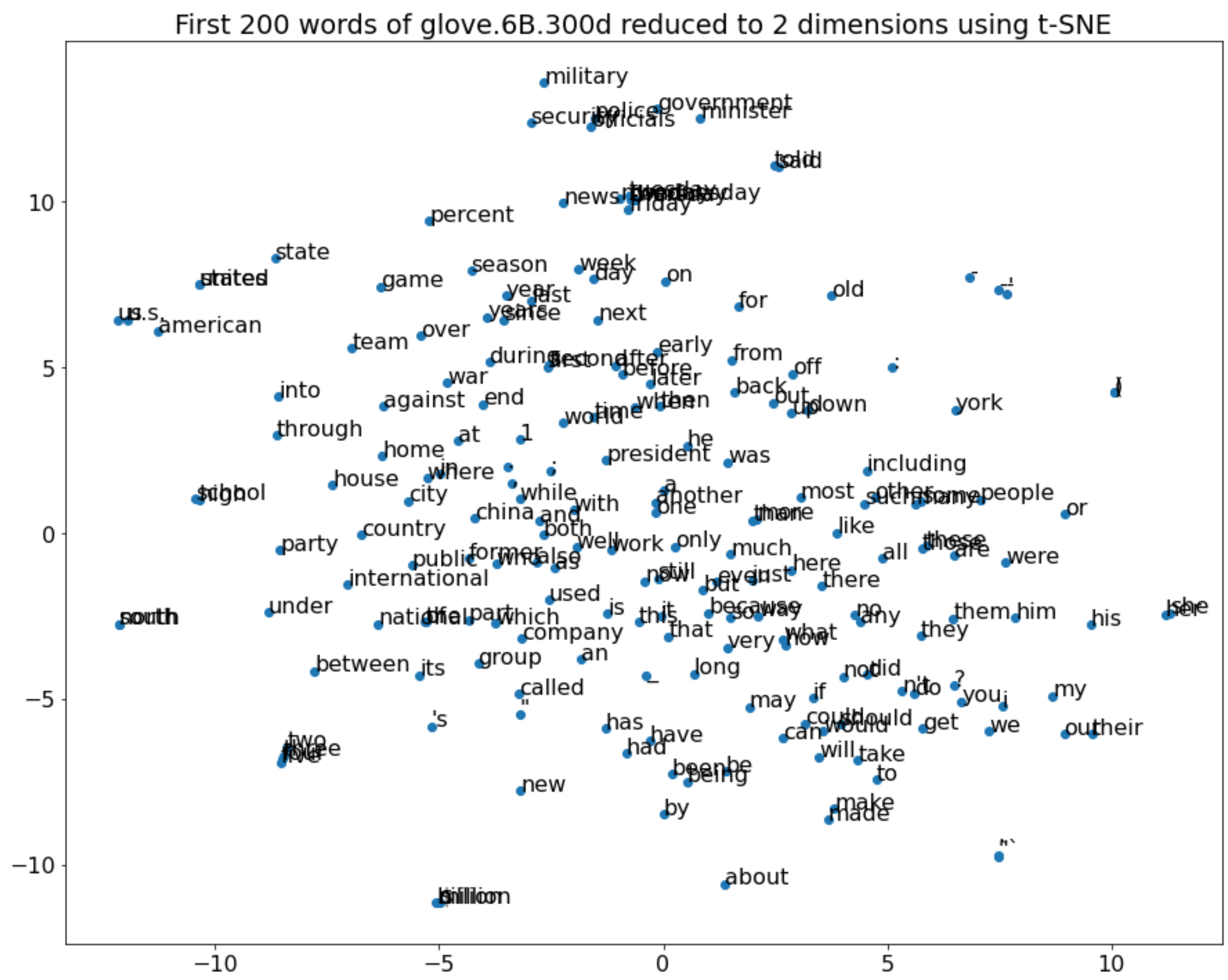
In [4]:
```
embeddings_dict_glove50 = {}
with open("glove.6B.50d.txt", 'r') as f:
    for line in f:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], "float32")
        embeddings_dict_glove50[word] = vector
```

In [5]:
```
plt.rcParams.update({'font.size': 16})
tsne = TSNE(n_components=2, random_state=0)
words =  list(embeddings_dict_glove50.keys())
vectors = [embeddings_dict_glove50[word] for word in words]
Y = tsne.fit_transform(vectors[:200])
plt.figure(1, figsize = (15,12))
plt.scatter(Y[:, 0], Y[:, 1])
#plt.ylim(0, 10)
for label, x, y in zip(words, Y[:, 0], Y[:, 1]):
    plt.annotate(label, xy=(x, y), xytext=(0, 0), textcoords="offset points
plt.title("First 200 words of glove.6B.50d reduced to 2 dimensions using t-
plt.show()
```



In [6]:
```python
embeddings_dict_glove300 = {}
with open("glove.6B.300d.txt", 'r') as f:
    for line in f:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], "float32")
        embeddings_dict_glove300[word] = vector
```

In [7]:
```python
plt.rcParams.update({'font.size': 16})
tsne = TSNE(n_components=2, random_state=0)
words =  list(embeddings_dict_glove300.keys())
vectors = [embeddings_dict_glove300[word] for word in words]
Y = tsne.fit_transform(vectors[:200])
plt.figure(1, figsize = (15,12))
plt.scatter(Y[:, 0], Y[:, 1])
plt.title("First 200 words of glove.6B.300d reduced to 2 dimensions using t
for label, x, y in zip(words, Y[:, 0], Y[:, 1]):
    plt.annotate(label, xy=(x, y), xytext=(0, 0), textcoords="offset points
plt.show()
```



In [8]:
```
loc = '/Users/allisonroeser/Desktop/movie-reviews-negative'
os.chdir(loc)
filelist = os.listdir()
#print (len((pd.concat([pd.read_csv(item, names=[item[:-4]]) for item in fi

data = []
path = loc
files = [f for f in os.listdir(path) if os.path.isfile(f)]
for f in files:
    with open(f,'r') as myfile:
        data.append(myfile.read())

negative_df = pd.DataFrame(data)
print (negative_df.shape)
```

```
(500, 1)
```

In [9]:
```
negative_df.reset_index(drop=True)
```

Out[9]:

	0
0	While some performances were good-Victoria Row...
1	There are many different versions of this one ...
2	0.5/10. This movie has absolutely nothing good...
3	I don't recall walking out of a movie theater ...
4	Home Alone 3 is one of my least favourite movi...
...	...
495	If I had not read Pat Barker's 'Union Street' ...
496	Mark Pirro's "Deathrow Gameshow" of 1987 is a ...
497	I know I've already added a comment but I just...
498	DEATHSTALKER is perfect for B-fantasy movie fa...
499	This was awful. Andie Macdowell is a terrible ...

500 rows × 1 columns

In [10]:
```
negative_df["Label"] = 0
negative_df
```

Out[10]:

	0	Label
0	While some performances were good-Victoria Row...	0
1	There are many different versions of this one ...	0
2	0.5/10. This movie has absolutely nothing good...	0
3	I don't recall walking out of a movie theater ...	0
4	Home Alone 3 is one of my least favourite movi...	0
...	...	...
495	If I had not read Pat Barker's 'Union Street' ...	0
496	Mark Pirro's "Deathrow Gameshow" of 1987 is a ...	0
497	I know I've already added a comment but I just...	0
498	DEATHSTALKER is perfect for B-fantasy movie fa...	0
499	This was awful. Andie Macdowell is a terrible ...	0

500 rows × 2 columns

In [11]:
```
negative_df.columns = ("Message", "Label")
negative_df
```

Out[11]:

	Message	Label
0	While some performances were good-Victoria Row...	0
1	There are many different versions of this one ...	0
2	0.5/10. This movie has absolutely nothing good...	0
3	I don't recall walking out of a movie theater ...	0
4	Home Alone 3 is one of my least favourite movi...	0
...	...	...
495	If I had not read Pat Barker's 'Union Street' ...	0
496	Mark Pirro's "Deathrow Gameshow" of 1987 is a ...	0
497	I know I've already added a comment but I just...	0
498	DEATHSTALKER is perfect for B-fantasy movie fa...	0
499	This was awful. Andie Macdowell is a terrible ...	0

500 rows × 2 columns

In [12]:

```
loc = '/Users/allisonroeser/Desktop/movie-reviews-positive'
os.chdir(loc)
filelist = os.listdir()
#print (len((pd.concat([pd.read_csv(item, names=[item[:-4]]) for item in fi

data = []
path = loc
files = [f for f in os.listdir(path) if os.path.isfile(f)]
for f in files:
    with open(f,'r') as myfile:
        data.append(myfile.read())

positive_df = pd.DataFrame(data)
print (positive_df.shape)
positive_df.reset_index(drop=True)
positive_df["Label"] = 1
positive_df.columns = ("Message", "Label")
positive_df
```

```
(500, 1)
```

Out[12]:

	Message	Label
0	Bizarre horror movie filled with famous faces ...	1
1	Caught the tail end of this movie channel surf...	1
2	this movie has a great message,a impressive ca...	1
3	I just recently watched this 1954 movie starri...	1
4	One of my favorite scenes is at the beginning ...	1
...	...	...
495	This is an excellent film, and is the sort of ...	1
496	Liked Stanley & Iris very much. Acting was ver...	1
497	Emilio Miraglio's "The Red Queen Kills Seven T...	1
498	Rumour has it that around the time that ABBA ...	1
499	When I saw the elaborate DVD box for this and ...	1

500 rows × 2 columns

In [13]:
```
data = negative_df.append(positive_df)
data
```

Out[13]:

	Message	Label
0	While some performances were good-Victoria Row...	0
1	There are many different versions of this one ...	0
2	0.5/10. This movie has absolutely nothing good...	0
3	I don't recall walking out of a movie theater ...	0
4	Home Alone 3 is one of my least favourite movi...	0
...	...	...
495	This is an excellent film, and is the sort of ...	1
496	Liked Stanley & Iris very much. Acting was ver...	1
497	Emilio Miraglio's "The Red Queen Kills Seven T...	1
498	Rumour has it that around the time that ABBA ...	1
499	When I saw the elaborate DVD box for this and ...	1

1000 rows × 2 columns

In [14]:
```
data = data.sample(frac=1).reset_index(drop=True)
data
```

Out[14]:

	Message	Label
0	What was always missing with the Matrix story ...	1
1	An old vaudeville team of Willy Clark (Walter ...	1
2	This only gets bashed because it stars David H...	0
3	Normally, I don't watch action movies because ...	1
4	I've really enjoyed this adaptation of "Emma"....	1
...	...	...
995	I saw this movie originally in the theater, wh...	0
996	Walter Matthau and George Burns were a famous ...	1
997	"Home Room" like "Zero Day" and "Elephant", wa...	1
998	This movie has beautiful scenery. Unfortunatel...	0
999	I think it's one of the greatest movies which ...	1

1000 rows × 2 columns

In [15]:
```
messages = []
labels = []
for index, row in data.iterrows():
    messages.append(row['Message'])
    labels.append(row['Label'])
messages = np.asarray(messages)
labels = np.asarray(labels)
```

In [16]:
```
len(messages)
```

Out[16]:
```
1000
```

In [17]:
```
length = []
for i in range (0, 1000):
    length.append(len(messages[i]))
```

In [18]:
```
length
```

Out[18]:
```
[784,
 1285,
 407,
 472,
 1559,
 680,
 1823,
 1739,
 712,
 1014,
 579,
 861,
 2365,
 381,
 1158,
 3002,
 231,
 601,
 2093,
```

In [19]:
```
messages[0]
```

Out[19]:
```
"What was always missing with the Matrix story was how things came to be
in the real world. Say no more, because this part of the story covered mo
st of the bases. What was truly interesting was how political it was, may
be even a cheap shot at the current presidential administration. Fascism
and violence were the only things man could think of in regards to fighti
ng the robotic horde, who were meant as nothing more than servants to hum
anity. What I also found interesting was the use of fear and how it was p
erpetuated by the idea of the unknown. We as humans tend to fall into tha
t trap quite often, letting the lack of logic and thought overtake us bec
ause people can't believe the contrary. Well represented and put togethe
r, this a true testament to how illogical humans can be."
```

In [20]:
```
data["length"] = length
```

In [21]:
```
data
```

Out[21]:

	Message	Label	length
0	What was always missing with the Matrix story ...	1	784
1	An old vaudeville team of Willy Clark (Walter ...	1	1285
2	This only gets bashed because it stars David H...	0	407
3	Normally, I don't watch action movies because ...	1	472
4	I've really enjoyed this adaptation of "Emma"....	1	1559
...	...	...	...
995	I saw this movie originally in the theater, wh...	0	950
996	Walter Matthau and George Burns were a famous ...	1	1532
997	"Home Room" like "Zero Day" and "Elephant", wa...	1	3315
998	This movie has beautiful scenery. Unfortunatel...	0	461
999	I think it's one of the greatest movies which ...	1	136

1000 rows × 3 columns

In [22]:
```
round(data.length.describe(),2)
```

Out[22]:
```
count     1000.00
mean      1248.54
std        947.74
min        123.00
25%        682.00
50%        924.50
75%       1504.25
max       6309.00
Name: length, dtype: float64
```

In [23]:
```
len(messages[0])
```

Out[23]:
```
784
```

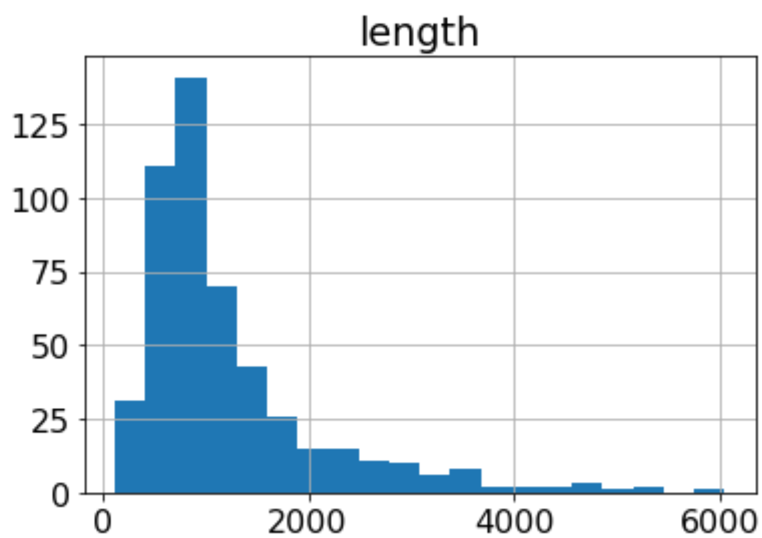
In [24]:
```
data.groupby(['Label']).describe()
```

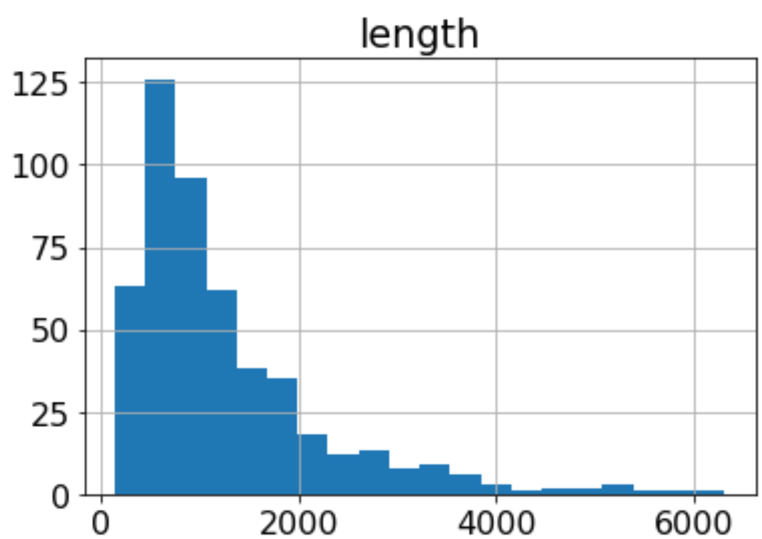
Out[24]:

	length							
	count	mean	std	min	25%	50%	75%	max
Label								
0	500.0	1231.712	898.084830	123.0	693.75	911.5	1442.25	6037.0
1	500.0	1265.360	995.530808	136.0	668.50	936.5	1560.50	6309.0

In [25]:
```
data.groupby(['Label']).hist(bins = 20)
```

Out[25]:
```
Label
0    [[AxesSubplot(0.125,0.125;0.775x0.755)]]
1    [[AxesSubplot(0.125,0.125;0.775x0.755)]]
dtype: object
```





In [26]:
```
messages[999]
```

Out[26]:
```
"I think it's one of the greatest movies which are ever made, and I've se
en many... The book is better, but it's still a very good movie!"
```

In [27]:
```
len(messages[999])
```
Out[27]: 136

In [28]:
```
labels[999]
```
Out[28]: 1

Create vocabulary size variables

In [29]:
```
vocabulary_size_20000 = 20000
tokenizer_20000 = Tokenizer(num_words= vocabulary_size_20000)
vocabulary_size_10000 = 10000
tokenizer_10000 = Tokenizer(num_words= vocabulary_size_10000)
```

In [30]:
```
from keras.preprocessing.text import Tokenizer
max_vocab = 10000
max_len = 50
tokenizer_10000 = Tokenizer(num_words=max_vocab)
tokenizer_10000.fit_on_texts(messages)
sequences_10000 = tokenizer_10000.texts_to_sequences(messages)
```

In [31]:
```
from keras.preprocessing.text import Tokenizer
max_vocab = 20000
max_len = 50
tokenizer_20000 = Tokenizer(num_words=max_vocab)
tokenizer_20000.fit_on_texts(messages)
sequences_20000 = tokenizer_20000.texts_to_sequences(messages)
```

In [32]:
```
from keras.preprocessing.sequence import pad_sequences
word_index = tokenizer_10000.word_index
data= pad_sequences(sequences_10000, maxlen=max_len)
```

Create embedding matrix

Modified code from:

https://medium.com/@sabber/classifying-yelp-review-comments-using-cnn-lstm-and-pre-trained-glove-word-embeddings-part-3-53fcea9a17fa

In [34]:
```
embedding_matrix_50d_10000v = np.zeros((vocabulary_size_10000, 50))
for word, index in tokenizer_10000.word_index.items():
    if index > vocabulary_size_10000 - 1:
        break
    else:
        embedding_vector_50 = embeddings_index_50.get(word)
        if embedding_vector_50 is not None:
            embedding_matrix_50d_10000v[index] = embedding_vector_50
```

In [35]:
```
embedding_matrix_50d_20000v = np.zeros((vocabulary_size_20000, 50))
for word, index in tokenizer_20000.word_index.items():
    if index > vocabulary_size_20000 - 1:
        break
    else:
        embedding_vector_50 = embeddings_index_50.get(word)
        if embedding_vector_50 is not None:
            embedding_matrix_50d_20000v[index] = embedding_vector_50
```

In [36]:
```
embedding_matrix_300d_10000v = np.zeros((vocabulary_size_10000, 300))
for word, index in tokenizer_10000.word_index.items():
    if index > vocabulary_size_10000 - 1:
        break
    else:
        embedding_vector_300 = embeddings_index_300.get(word)
        if embedding_vector_300 is not None:
            embedding_matrix_300d_10000v[index] = embedding_vector_300
```

In [37]:
```
embedding_matrix_300d_20000v = np.zeros((vocabulary_size_20000, 300))
for word, index in tokenizer_20000.word_index.items():
    if index > vocabulary_size_20000 - 1:
        break
    else:
        embedding_vector_300 = embeddings_index_300.get(word)
        if embedding_vector_300 is not None:
            embedding_matrix_300d_20000v[index] = embedding_vector_300
```

Model 1 / RNN model with vocab of 10,000 words and glove.6B.50d vector

In [167]:
```
word_index = tokenizer_10000.word_index
data= pad_sequences(sequences_10000, maxlen=max_len)
```

In [168]:
```
train_samples = int(len(messages)*0.8)
messages_train = data[:train_samples]
labels_train = labels[:train_samples]
messages_test = data[train_samples:len(messages)]
labels_test = labels[train_samples:len(messages)]
```

In [169]:
```
print('---Review---')
print(messages_train[0])
print('---Label---')
print(labels_train[0])
```

```
---Review---
[  11   14 8998   30    1  317    4    1 1151   81   13 1337 2067    5
  724   86   12 3073  202  413 2724    1  638    4 2257    2  203 8999
  184   82   75  188  303    1 3542   71 3074    2  328  294    9    3
  234 6426    5   90 9000 1337   67   26]
---Label---
1
```

In [170]:
```
len(messages_train)
```
Out[170]:
```
800
```

In [171]:
```
len(messages_test)
```
Out[171]:
```
200
```

In [172]:
```
print('---Review---')
print(messages_train[0])
print('---Label---')
print(labels_train[0])
```
```
---Review---
[  11   14 8998   30    1  317    4    1 1151   81   13 1337 2067    5
  724   86   12 3073  202  413 2724    1  638    4 2257    2  203 8999
  184   82   75  188  303    1 3542   71 3074    2  328  294    9    3
  234 6426    5   90 9000 1337   67   26]
---Label---
1
```

In [173]:
```
max_features = 10000
maxlen = 80
batch_size = 32
```

In [174]:
```
model = Sequential()
model.add(Embedding(max_features, 50, weights=[embedding_matrix_50d_10000v]
model.add(LSTM(128, dropout = 0.2, recurrent_dropout = 0.2))
model.add(Dense(1, activation = "sigmoid"))
```

In [175]:
```
model.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['accuracy'])
```

In [176]:
```
start_time = time.process_time()
history= model.fit(messages_train, labels_train, batch_size = batch_size, v
end_time = time.process_time()
model_1_time = end_time - start_time
```

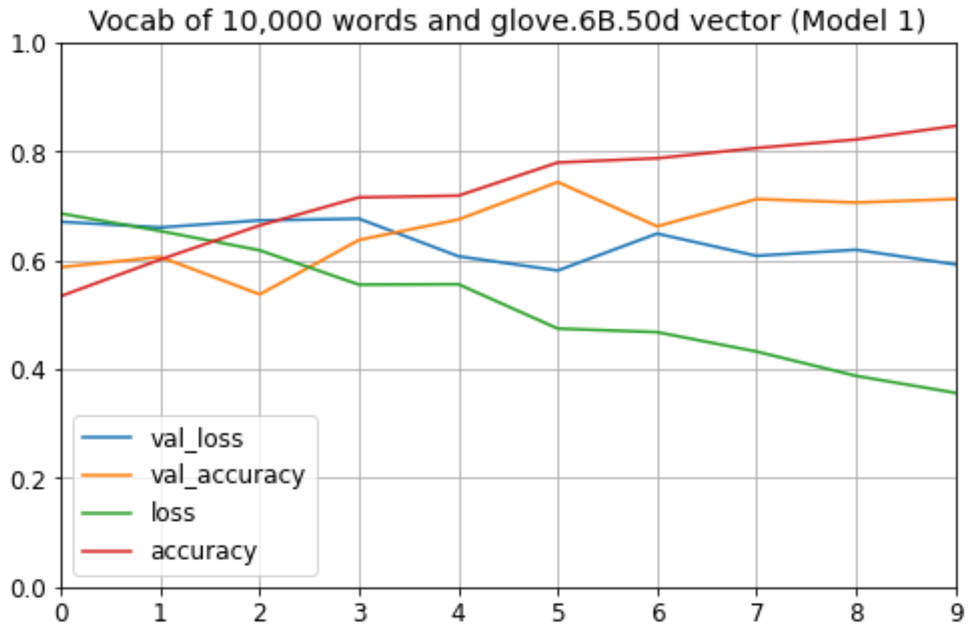
```
/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/tensorflo
w/python/framework/indexed_slices.py:434: UserWarning: Converting sparse
IndexedSlices to a dense Tensor of unknown shape. This may consume a larg
e amount of memory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "

Train on 640 samples, validate on 160 samples
Epoch 1/10
640/640 [==============================] - 1s 2ms/step - loss: 0.6860 - a
ccuracy: 0.5344 - val_loss: 0.6708 - val_accuracy: 0.5875
Epoch 2/10
640/640 [==============================] - 1s 1ms/step - loss: 0.6538 - a
ccuracy: 0.6016 - val_loss: 0.6604 - val_accuracy: 0.6062
Epoch 3/10
640/640 [==============================] - 1s 2ms/step - loss: 0.6184 - a
ccuracy: 0.6641 - val_loss: 0.6734 - val_accuracy: 0.5375
Epoch 4/10
640/640 [==============================] - 1s 2ms/step - loss: 0.5554 - a
ccuracy: 0.7156 - val_loss: 0.6765 - val_accuracy: 0.6375
Epoch 5/10
640/640 [==============================] - 1s 2ms/step - loss: 0.5561 - a
ccuracy: 0.7188 - val_loss: 0.6074 - val_accuracy: 0.6750
Epoch 6/10
640/640 [==============================] - 1s 2ms/step - loss: 0.4746 - a
ccuracy: 0.7797 - val_loss: 0.5814 - val_accuracy: 0.7437
Epoch 7/10
640/640 [==============================] - 1s 2ms/step - loss: 0.4681 - a
ccuracy: 0.7875 - val_loss: 0.6491 - val_accuracy: 0.6625
Epoch 8/10
640/640 [==============================] - 1s 1ms/step - loss: 0.4325 - a
ccuracy: 0.8062 - val_loss: 0.6083 - val_accuracy: 0.7125
Epoch 9/10
640/640 [==============================] - 1s 1ms/step - loss: 0.3879 - a
ccuracy: 0.8219 - val_loss: 0.6194 - val_accuracy: 0.7063
Epoch 10/10
640/640 [==============================] - 1s 2ms/step - loss: 0.3567 - a
ccuracy: 0.8469 - val_loss: 0.5925 - val_accuracy: 0.7125
```

In [177]:
```
acc = model.evaluate(messages_test, labels_test)
print("Test loss is {0:.2f} accuracy is {1:.2f}  ".format(acc[0],acc[1]))
```

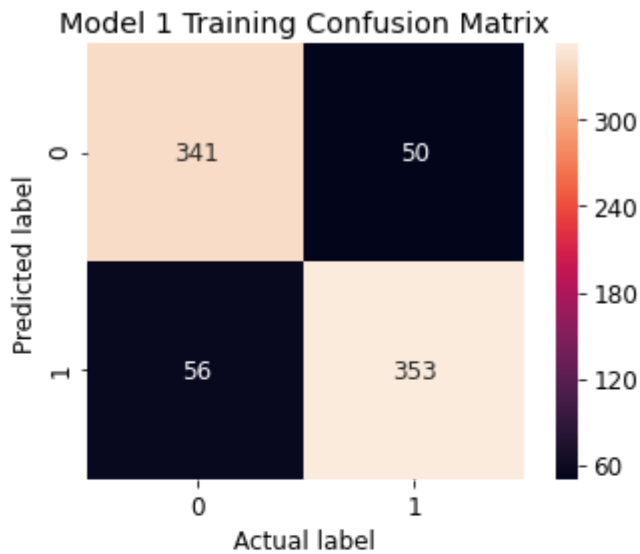
```
200/200 [==============================] - 0s 416us/step
Test loss is 0.63 accuracy is 0.69
```

In [178]:
```
plt.rcParams.update({'font.size': 12})
history.params
pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1) #save_fig("keras_learning_curves_plot") plt.show()
plt.title("Vocab of 10,000 words and glove.6B.50d vector (Model 1)")
```

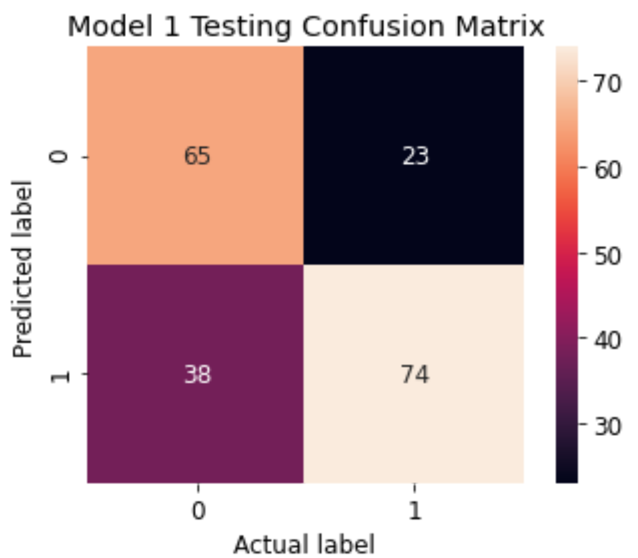
Out[178]:
```
Text(0.5, 1.0, 'Vocab of 10,000 words and glove.6B.50d vector (Model 1)')
```



In [179]:
```
#Plot Confusion Matrix DNN
cm_tst = confusion_matrix(labels_train, model.predict_classes(messages_trai
cm_tst_plt=sns.heatmap(cm_tst.T, square=True, annot=True, fmt='d')
plt.xlabel('Actual label')
plt.ylabel('Predicted label')
plt.title("Model 1 Training Confusion Matrix");
fig3 = cm_tst_plt.get_figure()
```



In [180]:
```
#Plot Confusion Matrix DNN
cm_tst_1 = confusion_matrix(labels_test, model.predict_classes(messages_tes
cm_tst_plt=sns.heatmap(cm_tst_1.T, square=True, annot=True, fmt='d')
plt.xlabel('Actual label')
plt.ylabel('Predicted label')
plt.title("Model 1 Testing Confusion Matrix");
fig3 = cm_tst_plt.get_figure()
```



In [181]:
```
model_1_train_acc = max(history.history["accuracy"])
model_1_train_acc
```

Out[181]: 0.846875

In [182]:
```
model_1_val_acc = max(history.history["val_accuracy"])
model_1_val_acc
```

Out[182]: 0.7437499761581421

In [183]:
```
model_1_test_loss, model_1_test_acc = model.evaluate(messages_test, labels_
model_1_test_acc
```

200/200 [==============================] - 0s 392us/step

Out[183]: 0.6949999928474426

In [ ]:

Model 2 / RNN model with vocab of 20,000 words and glove.6B.50d vector

In [184]:
```
word_index = tokenizer_20000.word_index
data= pad_sequences(sequences_20000, maxlen=max_len)
```

In [185]:
```
train_samples = int(len(messages)*0.8)
messages_train = data[:train_samples]
labels_train = labels[:train_samples]
messages_test = data[train_samples:len(messages)]
labels_test = labels[train_samples:len(messages)]
```

In [186]:
```
max_features = 20000
maxlen = 80
batch_size = 32
```

In [187]:
```
model = Sequential()
model.add(Embedding(max_features, 50, weights=[embedding_matrix_50d_20000v]
model.add(LSTM(128, dropout = 0.2, recurrent_dropout = 0.2))
model.add(Dense(1, activation = "sigmoid"))
```

In [188]:
```
model.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['accuracy'])
```

In [189]:
```
start_time = time.process_time()
history= model.fit(messages_train, labels_train, batch_size = batch_size, v
end_time = time.process_time()
model_2_time = end_time - start_time
```

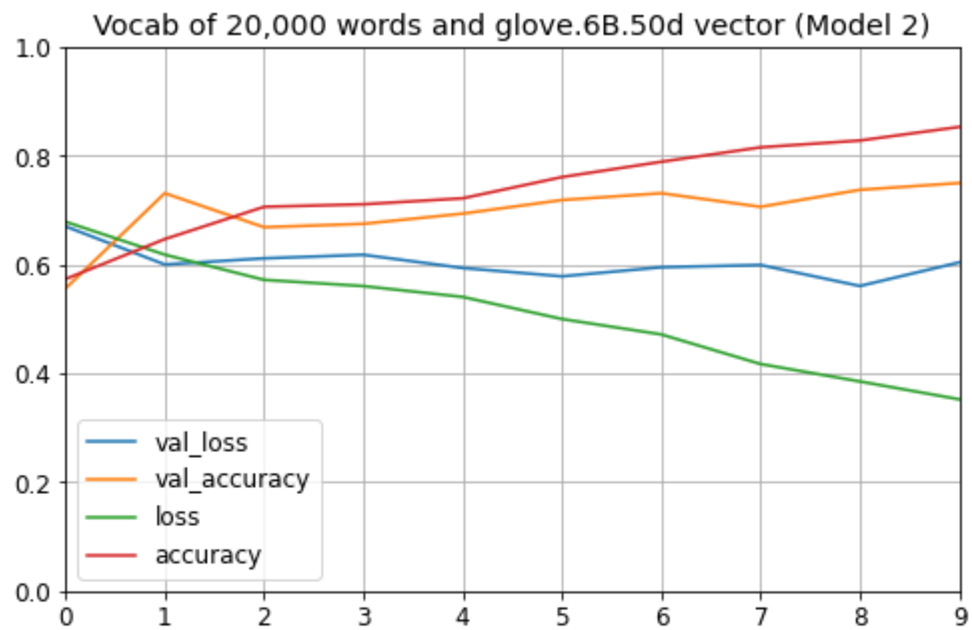
```
/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/tensorflo
w/python/framework/indexed_slices.py:434: UserWarning: Converting sparse
IndexedSlices to a dense Tensor of unknown shape. This may consume a larg
e amount of memory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "

Train on 640 samples, validate on 160 samples
Epoch 1/10
640/640 [==============================] - 2s 3ms/step - loss: 0.6788 - a
ccuracy: 0.5734 - val_loss: 0.6704 - val_accuracy: 0.5562
Epoch 2/10
640/640 [==============================] - 1s 2ms/step - loss: 0.6184 - a
ccuracy: 0.6469 - val_loss: 0.6003 - val_accuracy: 0.7312
Epoch 3/10
640/640 [==============================] - 1s 2ms/step - loss: 0.5720 - a
ccuracy: 0.7063 - val_loss: 0.6115 - val_accuracy: 0.6687
Epoch 4/10
640/640 [==============================] - 1s 2ms/step - loss: 0.5606 - a
ccuracy: 0.7109 - val_loss: 0.6181 - val_accuracy: 0.6750
Epoch 5/10
640/640 [==============================] - 1s 2ms/step - loss: 0.5406 - a
ccuracy: 0.7219 - val_loss: 0.5938 - val_accuracy: 0.6938
Epoch 6/10
640/640 [==============================] - 1s 2ms/step - loss: 0.5000 - a
ccuracy: 0.7609 - val_loss: 0.5785 - val_accuracy: 0.7188
Epoch 7/10
640/640 [==============================] - 1s 2ms/step - loss: 0.4717 - a
ccuracy: 0.7891 - val_loss: 0.5952 - val_accuracy: 0.7312
Epoch 8/10
640/640 [==============================] - 1s 2ms/step - loss: 0.4173 - a
ccuracy: 0.8156 - val_loss: 0.5993 - val_accuracy: 0.7063
Epoch 9/10
640/640 [==============================] - 1s 2ms/step - loss: 0.3853 - a
ccuracy: 0.8281 - val_loss: 0.5607 - val_accuracy: 0.7375
Epoch 10/10
640/640 [==============================] - 1s 2ms/step - loss: 0.3524 - a
ccuracy: 0.8531 - val_loss: 0.6041 - val_accuracy: 0.7500
```

In [190]:
```
acc = model.evaluate(messages_test, labels_test)
print("Test loss is {0:.2f} accuracy is {1:.2f}  ".format(acc[0],acc[1]))
```

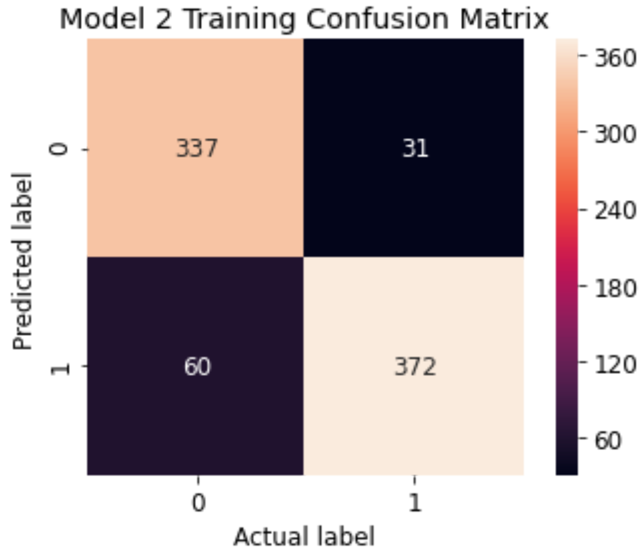
```
200/200 [==============================] - 0s 440us/step
Test loss is 0.63 accuracy is 0.69
```

In [191]:
```
history.params
pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1) #save_fig("keras_learning_curves_plot") plt.show()
plt.title("Vocab of 20,000 words and glove.6B.50d vector (Model 2)")
```

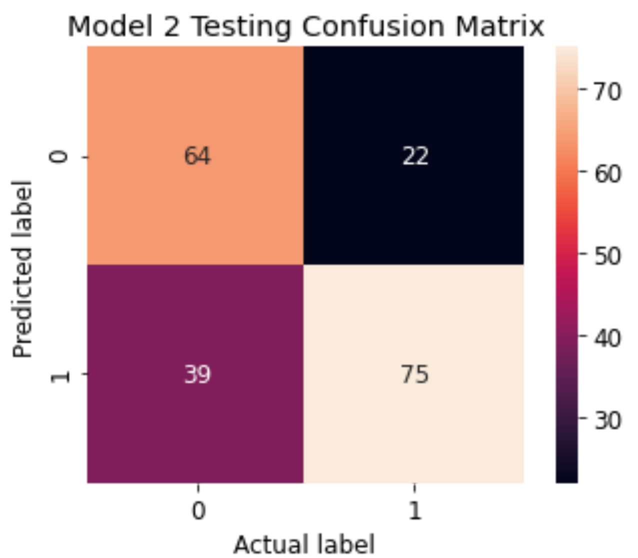
Out[191]:
```
Text(0.5, 1.0, 'Vocab of 20,000 words and glove.6B.50d vector (Model 2)')
```



In [192]:
```
#Plot Confusion Matrix DNN
cm_tst = confusion_matrix(labels_train, model.predict_classes(messages_trai
cm_tst_plt=sns.heatmap(cm_tst.T, square=True, annot=True, fmt='d')
plt.xlabel('Actual label')
plt.ylabel('Predicted label')
plt.title("Model 2 Training Confusion Matrix");
fig3 = cm_tst_plt.get_figure()
```



In [193]:
```
#Plot Confusion Matrix DNN
cm_tst_2 = confusion_matrix(labels_test, model.predict_classes(messages_tes
cm_tst_plt=sns.heatmap(cm_tst_2.T, square=True, annot=True, fmt='d')
plt.xlabel('Actual label')
plt.ylabel('Predicted label')
plt.title("Model 2 Testing Confusion Matrix");
fig3 = cm_tst_plt.get_figure()
```



In [194]:
```
model_2_train_acc = max(history.history["accuracy"])
model_2_train_acc
```

Out[194]: 0.853125

In [195]:
```
model_2_val_acc = max(history.history["val_accuracy"])
model_2_val_acc
```

Out[195]: 0.75

In [196]:
```
model_2_test_loss, model_2_test_acc = model.evaluate(messages_test, labels_
model_2_test_acc
```

```
200/200 [==============================] - 0s 371us/step
```

Out[196]: 0.6949999928474426

In [ ]:

Model 3 / RNN model with vocab of 10,000 words and glove.6B.300d vector

In [197]:
```
from keras.preprocessing.sequence import pad_sequences
word_index = tokenizer_10000.word_index
data= pad_sequences(sequences_10000, maxlen=max_len)
```

In [198]:
```
train_samples = int(len(messages)*0.8)
messages_train = data[:train_samples]
labels_train = labels[:train_samples]
messages_test = data[train_samples:len(messages)]
labels_test = labels[train_samples:len(messages)]
```

In [199]:
```
max_features = 10000
maxlen = 80
batch_size = 32
```

In [200]:
```
model = Sequential()
model.add(Embedding(max_features, 300, weights=[embedding_matrix_300d_10000
model.add(LSTM(128, dropout = 0.2, recurrent_dropout = 0.2))
model.add(Dense(1, activation = "sigmoid"))
```

In [201]:
```
model.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['accuracy'])
```

In [202]:
```
start_time = time.process_time()
history= model.fit(messages_train, labels_train, batch_size = batch_size, v
end_time = time.process_time()
model_3_time = end_time - start_time
```

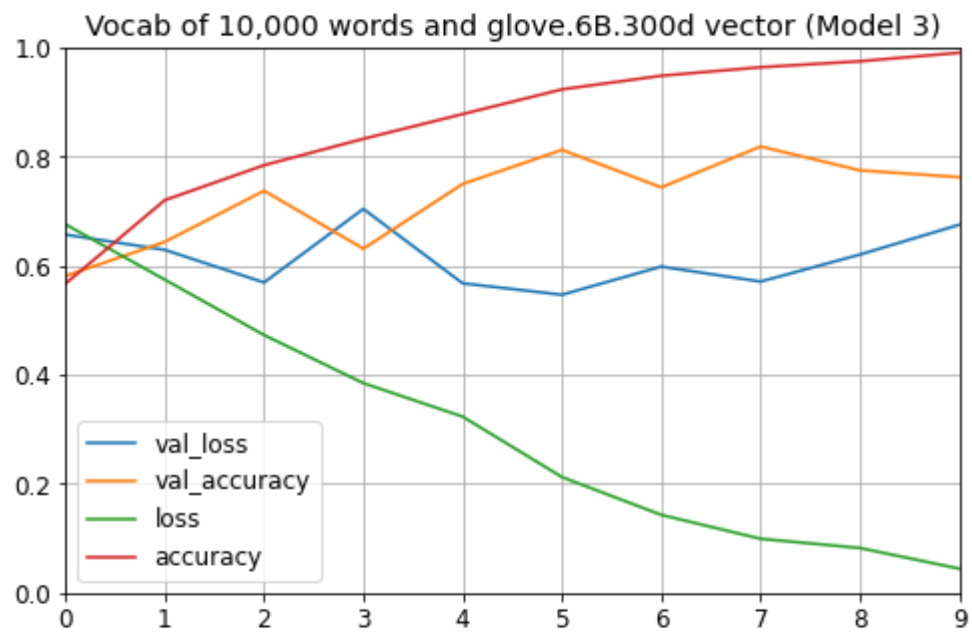
```
/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/tensorflo
w/python/framework/indexed_slices.py:434: UserWarning: Converting sparse
IndexedSlices to a dense Tensor of unknown shape. This may consume a larg
e amount of memory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "

Train on 640 samples, validate on 160 samples
Epoch 1/10
640/640 [==============================] - 3s 4ms/step - loss: 0.6760 - a
ccuracy: 0.5672 - val_loss: 0.6571 - val_accuracy: 0.5813
Epoch 2/10
640/640 [==============================] - 2s 3ms/step - loss: 0.5746 - a
ccuracy: 0.7203 - val_loss: 0.6292 - val_accuracy: 0.6438
Epoch 3/10
640/640 [==============================] - 2s 3ms/step - loss: 0.4729 - a
ccuracy: 0.7844 - val_loss: 0.5693 - val_accuracy: 0.7375
Epoch 4/10
640/640 [==============================] - 2s 3ms/step - loss: 0.3847 - a
ccuracy: 0.8328 - val_loss: 0.7043 - val_accuracy: 0.6313
Epoch 5/10
640/640 [==============================] - 2s 3ms/step - loss: 0.3233 - a
ccuracy: 0.8781 - val_loss: 0.5678 - val_accuracy: 0.7500
Epoch 6/10
640/640 [==============================] - 2s 3ms/step - loss: 0.2124 - a
ccuracy: 0.9234 - val_loss: 0.5469 - val_accuracy: 0.8125
Epoch 7/10
640/640 [==============================] - 2s 3ms/step - loss: 0.1432 - a
ccuracy: 0.9484 - val_loss: 0.5985 - val_accuracy: 0.7437
Epoch 8/10
640/640 [==============================] - 2s 3ms/step - loss: 0.0993 - a
ccuracy: 0.9641 - val_loss: 0.5710 - val_accuracy: 0.8188
Epoch 9/10
640/640 [==============================] - 2s 3ms/step - loss: 0.0824 - a
ccuracy: 0.9750 - val_loss: 0.6205 - val_accuracy: 0.7750
Epoch 10/10
640/640 [==============================] - 2s 3ms/step - loss: 0.0446 - a
ccuracy: 0.9906 - val_loss: 0.6752 - val_accuracy: 0.7625
```

In [203]:
```
acc = model.evaluate(messages_test, labels_test)
print("Test loss is {0:.2f} accuracy is {1:.2f}  ".format(acc[0],acc[1]))
```

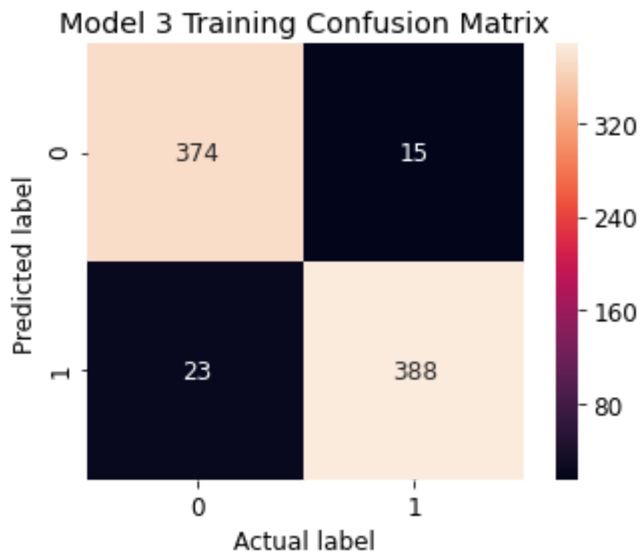
```
200/200 [==============================] - 0s 811us/step
Test loss is 0.69 accuracy is 0.75
```

In [204]:
```
history.params
pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1) #save_fig("keras_learning_curves_plot") plt.show()
plt.title("Vocab of 10,000 words and glove.6B.300d vector (Model 3)")
```

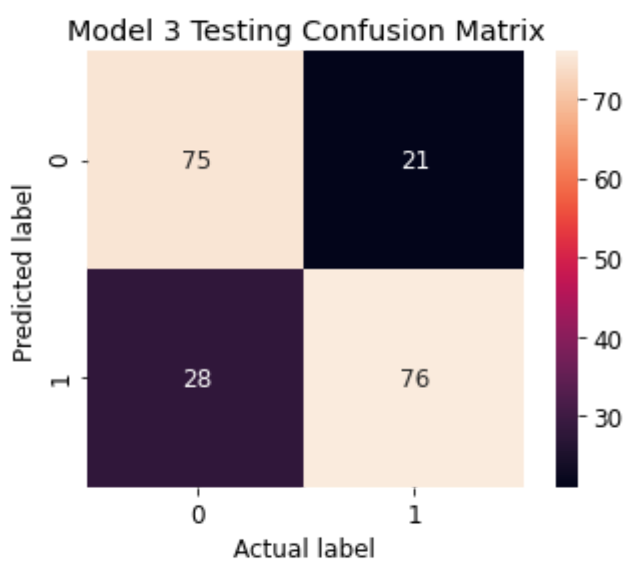
Out[204]:
```
Text(0.5, 1.0, 'Vocab of 10,000 words and glove.6B.300d vector (Model
3)')
```



In [205]:
```
#Plot Confusion Matrix DNN
cm_tst = confusion_matrix(labels_train, model.predict_classes(messages_trai
cm_tst_plt=sns.heatmap(cm_tst.T, square=True, annot=True, fmt='d')
plt.xlabel('Actual label')
plt.ylabel('Predicted label')
plt.title("Model 3 Training Confusion Matrix");
fig3 = cm_tst_plt.get_figure()
```



In [206]:
```
#Plot Confusion Matrix DNN
cm_tst_3 = confusion_matrix(labels_test, model.predict_classes(messages_tes
cm_tst_plt=sns.heatmap(cm_tst_3.T, square=True, annot=True, fmt='d')
plt.xlabel('Actual label')
plt.ylabel('Predicted label')
plt.title("Model 3 Testing Confusion Matrix");
fig3 = cm_tst_plt.get_figure()
```



In [207]:
```
model_3_train_acc = max(history.history["accuracy"])
model_3_train_acc
```

Out[207]: 0.990625

In [208]:
```
model_3_val_acc = max(history.history["val_accuracy"])
model_3_val_acc
```

Out[208]: 0.8187500238418579

In [209]:
```
model_3_test_loss, model_3_test_acc = model.evaluate(messages_test, labels_
model_3_test_acc
```

```
200/200 [==============================] - 0s 577us/step
```

Out[209]: 0.7549999952316284

In [ ]:

In [ ]:

In [ ]:

Model 4 / RNN model with vocab of 20,000 words and glove.6B.300d vector

In [210]:
```
word_index = tokenizer_20000.word_index
data= pad_sequences(sequences_20000, maxlen=max_len)
```

In [211]:
```
train_samples = int(len(messages)*0.8)
messages_train = data[:train_samples]
labels_train = labels[:train_samples]
messages_test = data[train_samples:len(messages)]
labels_test = labels[train_samples:len(messages)]
```

In [212]:
```
max_features = 20000
maxlen = 80
batch_size = 32
```

In [213]:
```
model = Sequential()
model.add(Embedding(max_features, 300, weights=[embedding_matrix_300d_20000
model.add(LSTM(128, dropout = 0.2, recurrent_dropout = 0.2))
model.add(Dense(1, activation = "sigmoid"))
```

In [214]:
```
model.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['accuracy'])
```

In [215]:
```
start_time = time.process_time()
history= model.fit(messages_train, labels_train, batch_size = batch_size, v
end_time = time.process_time()
model_4_time = end_time - start_time
```

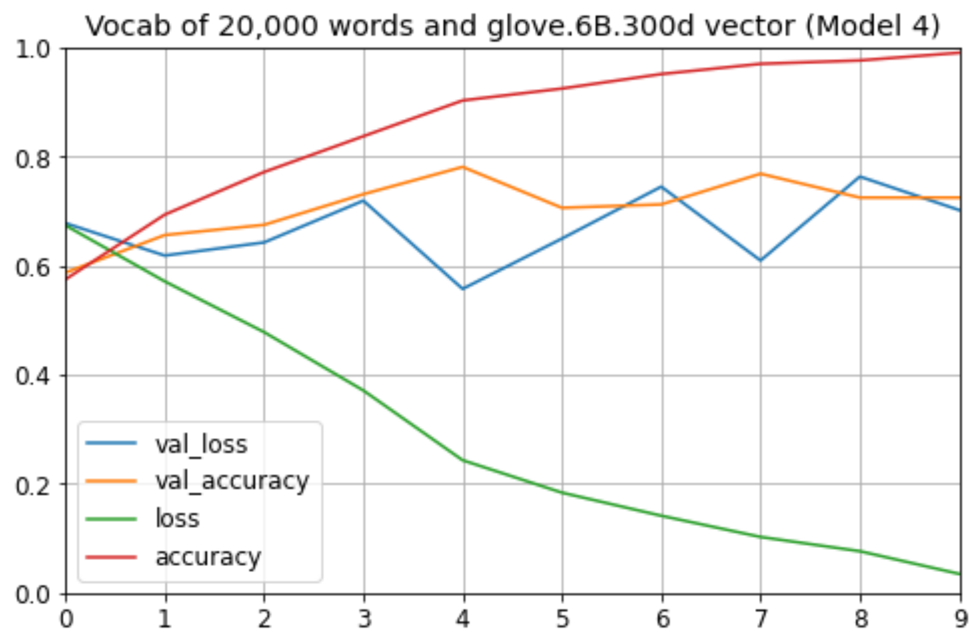
```
/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/tensorflo
w/python/framework/indexed_slices.py:434: UserWarning: Converting sparse
IndexedSlices to a dense Tensor of unknown shape. This may consume a larg
e amount of memory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "

Train on 640 samples, validate on 160 samples
Epoch 1/10
640/640 [==============================] - 3s 5ms/step - loss: 0.6741 - a
ccuracy: 0.5750 - val_loss: 0.6781 - val_accuracy: 0.5875
Epoch 2/10
640/640 [==============================] - 3s 4ms/step - loss: 0.5714 - a
ccuracy: 0.6938 - val_loss: 0.6187 - val_accuracy: 0.6562
Epoch 3/10
640/640 [==============================] - 2s 4ms/step - loss: 0.4783 - a
ccuracy: 0.7719 - val_loss: 0.6428 - val_accuracy: 0.6750
Epoch 4/10
640/640 [==============================] - 2s 4ms/step - loss: 0.3713 - a
ccuracy: 0.8375 - val_loss: 0.7194 - val_accuracy: 0.7312
Epoch 5/10
640/640 [==============================] - 2s 4ms/step - loss: 0.2433 - a
ccuracy: 0.9031 - val_loss: 0.5576 - val_accuracy: 0.7812
Epoch 6/10
640/640 [==============================] - 2s 4ms/step - loss: 0.1839 - a
ccuracy: 0.9250 - val_loss: 0.6499 - val_accuracy: 0.7063
Epoch 7/10
640/640 [==============================] - 2s 4ms/step - loss: 0.1415 - a
ccuracy: 0.9516 - val_loss: 0.7450 - val_accuracy: 0.7125
Epoch 8/10
640/640 [==============================] - 3s 4ms/step - loss: 0.1026 - a
ccuracy: 0.9703 - val_loss: 0.6097 - val_accuracy: 0.7688
Epoch 9/10
640/640 [==============================] - 2s 4ms/step - loss: 0.0765 - a
ccuracy: 0.9766 - val_loss: 0.7635 - val_accuracy: 0.7250
Epoch 10/10
640/640 [==============================] - 2s 4ms/step - loss: 0.0349 - a
ccuracy: 0.9906 - val_loss: 0.7022 - val_accuracy: 0.7250
```

In [216]:
```
acc = model.evaluate(messages_test, labels_test)
print("Test loss is {0:.2f} accuracy is {1:.2f}  ".format(acc[0],acc[1]))
```

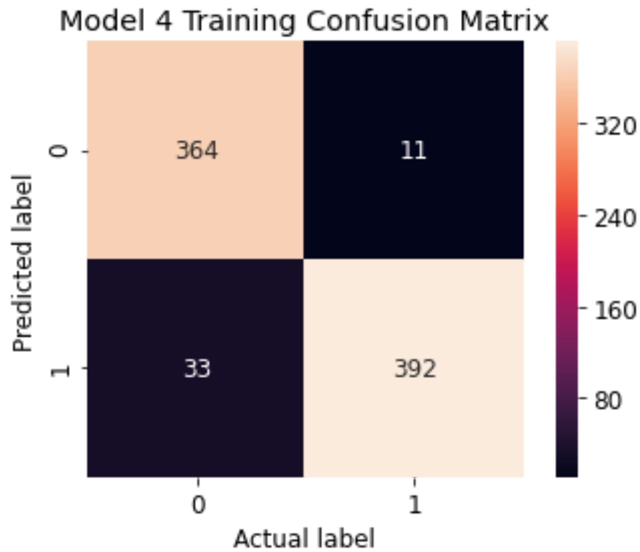
```
200/200 [==============================] - 0s 617us/step
Test loss is 0.81 accuracy is 0.70
```

In [217]:
```
history.params
pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1) #save_fig("keras_learning_curves_plot") plt.show()
plt.title("Vocab of 20,000 words and glove.6B.300d vector (Model 4)")
```

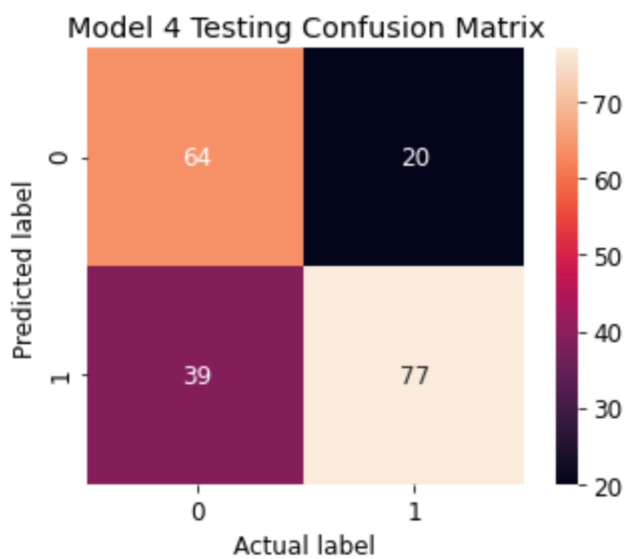
Out[217]:
```
Text(0.5, 1.0, 'Vocab of 20,000 words and glove.6B.300d vector (Model
4)')
```



In [218]:
```
#Plot Confusion Matrix DNN
cm_tst = confusion_matrix(labels_train, model.predict_classes(messages_trai
cm_tst_plt=sns.heatmap(cm_tst.T, square=True, annot=True, fmt='d')
plt.xlabel('Actual label')
plt.ylabel('Predicted label')
plt.title("Model 4 Training Confusion Matrix");
fig3 = cm_tst_plt.get_figure()
```



In [219]:
```
#Plot Confusion Matrix DNN
cm_tst_4 = confusion_matrix(labels_test, model.predict_classes(messages_tes
cm_tst_plt=sns.heatmap(cm_tst_4.T, square=True, annot=True, fmt='d')
plt.xlabel('Actual label')
plt.ylabel('Predicted label')
plt.title("Model 4 Testing Confusion Matrix");
fig3 = cm_tst_plt.get_figure()
```



In [220]:
```
model_4_train_acc = max(history.history["accuracy"])
model_4_train_acc
```

Out[220]: 0.990625

In [221]:
```
model_4_val_acc = max(history.history["val_accuracy"])
model_4_val_acc
```

Out[221]: 0.78125

In [222]:
```
model_4_test_loss, model_4_test_acc = model.evaluate(messages_test, labels_
model_4_test_acc
```

```
200/200 [==============================] - 0s 618us/step
```

Out[222]: 0.7049999833106995

In [223]:
```
model_1_sen = cm_tst_1[1,1]/(cm_tst_1[1,1]+cm_tst_1[1,0])
model_2_sen = cm_tst_2[1,1]/(cm_tst_2[1,1]+cm_tst_2[1,0])
model_3_sen = cm_tst_3[1,1]/(cm_tst_3[1,1]+cm_tst_3[1,0])
model_4_sen = cm_tst_4[1,1]/(cm_tst_4[1,1]+cm_tst_4[1,0])
```

In [224]:
```
model_1_spec = cm_tst_1[0,0]/(cm_tst_1[0,0]+cm_tst_1[0,1])
model_2_spec = cm_tst_2[0,0]/(cm_tst_2[0,0]+cm_tst_2[0,1])
model_3_spec = cm_tst_3[0,0]/(cm_tst_3[0,0]+cm_tst_3[0,1])
model_4_spec = cm_tst_4[0,0]/(cm_tst_4[0,0]+cm_tst_4[0,1])
```

In [225]:
```
model_1_sen
```

Out[225]: 0.7628865979381443

In [226]:
```
model_1_spec
```

Out[226]: 0.6310679611650486

In [239]:
```
model_1_tppv = cm_tst_1[1,1]/(cm_tst_1[1,1]+cm_tst_1[0,1])
model_2_tppv = cm_tst_2[1,1]/(cm_tst_2[1,1]+cm_tst_2[0,1])
model_3_tppv = cm_tst_3[1,1]/(cm_tst_3[1,1]+cm_tst_3[0,1])
model_4_tppv = cm_tst_4[1,1]/(cm_tst_4[1,1]+cm_tst_4[0,1])
```

In [243]:
```
model_1_tnpv = cm_tst_1[0,0]/(cm_tst_1[0,0]+cm_tst_1[1,0])
model_2_tnpv = cm_tst_2[0,0]/(cm_tst_2[0,0]+cm_tst_2[1,0])
model_3_tnpv = cm_tst_3[0,0]/(cm_tst_3[0,0]+cm_tst_3[1,0])
model_4_tnpv = cm_tst_4[0,0]/(cm_tst_4[0,0]+cm_tst_4[1,0])
```

Summary

In [241]:
```
summary = [[1,"10,000", "glove.6B.50d", round(model_1_time,2), round(model_
           [2,"20,000", "glove.6B.50d", round(model_2_time,2), round(model_2
           [3,"10,000", "glove.6B.300d", round(model_3_time,2), round(model_
           [4,"20,000", "glove.6B.300d",  round(model_4_time,2), round(model

summary_df = pd.DataFrame(summary, columns =['Model Number' ,'Vocab Size',

summary_df
```

Out[241]:

	Model Number	Vocab Size	Word Vector	Processing Time (Sec)	Training Accuracy	Validation Accuracy	Testing Accuracy
0	1	10,000	glove.6B.50d	50.47	0.847	0.744	0.695
1	2	20,000	glove.6B.50d	56.24	0.853	0.750	0.695
2	3	10,000	glove.6B.300d	91.26	0.991	0.819	0.755
3	4	20,000	glove.6B.300d	118.54	0.991	0.781	0.705

In [245]:
```
summary = [[1,"10,000", "glove.6B.50d", round(model_1_sen,3), round(model_1
           [2,"20,000", "glove.6B.50d", round(model_2_sen,3), round(model_2_
           [3,"10,000", "glove.6B.300d", round(model_3_sen,3), round(model_3
           [4,"20,000", "glove.6B.300d",  round(model_4_sen,3), round(model_

summary_df = pd.DataFrame(summary, columns =['Model Number' ,'Vocab Size',

summary_df
```

Out[245]:

	Model Number	Vocab Size	Word Vector	Sensitivity	Specificity	Positive Predictive Value	Negative Predictive Value
0	1	10,000	glove.6B.50d	0.763	0.631	0.661	0.739
1	2	20,000	glove.6B.50d	0.773	0.621	0.658	0.744
2	3	10,000	glove.6B.300d	0.784	0.728	0.731	0.781
3	4	20,000	glove.6B.300d	0.794	0.621	0.664	0.762

In [ ]:

In [ ]:

In [ ]: