In [2]:

```
import sys
import os
import json
import pandas
import numpy as np
import optparse
import keras
import tensorflow as tf
import matplotlib.pyplot as plt
plt.style.use('ggplot')
from keras.callbacks import TensorBoard
from keras.models import Sequential, load_model
from keras.layers import LSTM, Dense, Dropout, Flatten, SimpleRNN, SpatialDropout1D, ActivityRegula
rization
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from keras.preprocessing.text import Tokenizer
from collections import OrderedDict
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

from sklearn.model_selection import train_test_split
from random import sample
import re
from keras.optimizers import SGD

import nltk
# nltk.download('stopwords')
from nltk.corpus import stopwords

from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.metrics import fbeta_score, make_scorer
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score
```

In [3]:

```
# Import dataset
dataframe = pandas.read_csv("data_1w (1).csv", engine='python')
```

In [4]:

```
dataframe.head(5)
```

Out[4]:

	Unnamed: 0	0	1	2	3	4	5	6	7	8	...	2040	2041	2042	2043	2044	2045	2046	2047	Score	Label
0	0	0	0	0	0	0	0	0	0	0	...	191	213	64	343	213	64	3026	753	1	0
1	1	0	0	0	0	0	0	0	0	0	...	316	24	702	18	277	149	363	1674	1	0
2	2	0	0	0	0	0	0	0	0	0	...	696	223	106	532	495	16	1062	2018	1	0
3	3	0	0	0	0	0	0	0	0	0	...	18	59	199	287	298	1	726	484	1	0
4	4	0	0	0	0	0	0	0	0	0	...	3507	128	10	73	507	562	277	149	1	0

5 rows × 2051 columns

In [5]:

```
dataframe.iloc[:, 1:-2].values
```

Out[5]:

```
array([[    0,     0,     0, ...,    64,  3026,   753],
       [    0,     0,     0, ...,   149,   363,  1674],
       [    0,     0,     0, ...,    16,  1062,  2018],
       ...,
       [    0,     0,     0, ...,    16,    45,   318],
       [    0,     0,     0, ...,     6,   134, 18865],
       [    0,     0,     0, ...,   782,   418,   514]])
```

In [6]:

```
data = dataframe.iloc[:, 1:-2].values
y_label = dataframe['Label'].values
y_score = dataframe['Score'].values
y_score = y_score - 1
```

In [7]:

```
# split 80% train and 20% test data
X_train, X_test, y_train, y_test = train_test_split(data, y_label, test_size=0.2, random_state=42)
```

Build Model

Random Forest W Grid Search

In [8]:

```
# build a base model using Random Forest with Grid Search
param = {'n_estimators': [50, 100, 200, 500], 'max_features': [2, 4, 'sqrt'], 'max_depth': [8, 10, 12, 16],
         'random_state': [42]}

#scoring = {'accuracy': make_scorer(accuracy_score),
           #'precision': make_scorer(precision_score, average = 'macro'),
           #'recall': make_scorer(recall_score, average = 'macro'),
           #'f1_macro': make_scorer(f1_score, average = 'macro'),
           #'f1_weighted': make_scorer(f1_score, average = 'weighted')}

rf_model = RandomForestClassifier()
rf_grid = GridSearchCV(rf_model, param, cv = 5, refit = True, scoring='accuracy', n_jobs=-1, verbose=5)
```

In [9]:

```
rf_grid.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    4.7s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:   36.0s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:  1.9min
[Parallel(n_jobs=-1)]: Done 240 out of 240 | elapsed:  4.2min finished
```

Out[9]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators='warn', n_jobs=None,
```

```
                                       oob_score=False,
                                       random_state=None, verbose=0,
                                       warm_start=False),
             iid='warn', n_jobs=-1,
             param_grid={'max_depth': [8, 10, 12, 16],
                         'max_features': [2, 4, 'sqrt'],
                         'n_estimators': [50, 100, 200, 500],
                         'random_state': [42]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=5)
```

In [10]:

```
print('Best Parameters: \n', rf_grid.best_params_)
```

```
Best Parameters:
 {'max_depth': 16, 'max_features': 'sqrt', 'n_estimators': 200, 'random_state': 42}
```

In [11]:

```
best_model = rf_grid.best_estimator_
grid_class = best_model.predict(X_test)
#ac = accuracy_score(y_test, grid_class)
#pc = precision_score(y_test, grid_class, average = 'macro', labels=np.unique(grid_class))
#rc = recall_score(y_test, grid_class, average = 'macro')
#f_macro = f1_score(y_test, grid_class, average = 'macro', labels=np.unique(grid_class))
#f_weight = f1_score(y_test, grid_class, average = 'weighted', labels=np.unique(grid_class))
#rf_metric = np.array([ac, pc, rc, f_macro, f_weight])
#print('Accuracy:', rf_metric[0])
#print('Precision:', rf_metric[1])
#print('Recall:', rf_metric[2])
#print('f1_macro:', rf_metric[3])
#print('f1_weighted:', rf_metric[4])
#print('average_metric:', np.mean(rf_metric))
```

In [12]:

```
grid_report = classification_report(y_test, grid_class)
print('test_result: \n',grid_report)
```

```
test_result:
              precision    recall  f1-score   support

           0       0.62      0.98      0.76      1012
           1       0.95      0.38      0.55       988

    accuracy                           0.69      2000
   macro avg       0.79      0.68      0.65      2000
weighted avg       0.79      0.69      0.65      2000
```

In [13]:

```
grid_train = best_model.predict(X_train)
```

In [14]:

```
grid_report_train = classification_report(y_train, grid_train)
print('train_result: \n',grid_report_train)
```

```
train_result:
              precision    recall  f1-score   support

           0       0.64      1.00      0.78      3988
           1       1.00      0.44      0.61      4012

    accuracy                           0.72      8000
   macro avg       0.82      0.72      0.69      8000
weighted avg       0.82      0.72      0.69      8000
```

Overfitting - reduce model complexity

In [36]:

```
rf_model_2 = RandomForestClassifier(max_depth=6, max_features=200, n_estimators=120, random_state=4
2)
rf_model_2.fit(X_train, y_train)
```

Out[36]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=6, max_features=200, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=120,
                       n_jobs=None, oob_score=False, random_state=42, verbose=0,
                       warm_start=False)
```

In [37]:

```
rf_class = rf_model_2.predict(X_test)
```

In [38]:

```
rf_report = classification_report(y_test, rf_class)
print('test_result: \n',rf_report)
```

```
test_result:
               precision    recall  f1-score   support

           0       0.59      1.00      0.74      1012
           1       1.00      0.30      0.46       988

    accuracy                           0.65      2000
   macro avg       0.79      0.65      0.60      2000
weighted avg       0.79      0.65      0.60      2000
```

In [39]:

```
rf_class_train = rf_model_2.predict(X_train)
```

In [40]:

```
rf_report_train = classification_report(y_train, rf_class_train)
print('test_result: \n',rf_report_train)
```

```
test_result:
               precision    recall  f1-score   support

           0       0.60      1.00      0.75      3988
           1       1.00      0.33      0.49      4012

    accuracy                           0.66      8000
   macro avg       0.80      0.66      0.62      8000
weighted avg       0.80      0.66      0.62      8000
```

reduce max_depth, reduce n_estimators, increase max features. Not Overfitting now.

Gradient Boosting Classifier W GridSearch

In [41]:

```
param_gradient = {'n_estimators': [50, 100],
                  'learning_rate': [0.4, 0.8],
```

```
                  'max_depth': [2, 4],
                  'random_state': [42]}

gb_boost = GradientBoostingClassifier()
gb_grid = GridSearchCV(gb_boost, param_gradient, cv = 5, scoring='accuracy', refit='True', n_jobs=-
1, verbose=5)
```

In [42]:

```
gb_grid.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:   45.0s
[Parallel(n_jobs=-1)]: Done  34 out of  40 | elapsed:  5.5min remaining:   58.3s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:  6.4min finished
```

Out[42]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=GradientBoostingClassifier(criterion='friedman_mse',
                                                  init=None, learning_rate=0.1,
                                                  loss='deviance', max_depth=3,
                                                  max_features=None,
                                                  max_leaf_nodes=None,
                                                  min_impurity_decrease=0.0,
                                                  min_impurity_split=None,
                                                  min_samples_leaf=1,
                                                  min_samples_split=2,
                                                  min_weight_fraction_leaf=0.0,
                                                  n_estimators=100,
                                                  n_iter_no_change=None,
                                                  presort='auto',
                                                  random_state=None,
                                                  subsample=1.0, tol=0.0001,
                                                  validation_fraction=0.1,
                                                  verbose=0, warm_start=False),
             iid='warn', n_jobs=-1,
             param_grid={'learning_rate': [0.4, 0.8], 'max_depth': [2, 4],
                         'n_estimators': [50, 100], 'random_state': [42]},
             pre_dispatch='2*n_jobs', refit='True', return_train_score=False,
             scoring='accuracy', verbose=5)
```

In [43]:

```
print('Best Parameters: \n', gb_grid.best_params_)
```

```
Best Parameters:
 {'learning_rate': 0.4, 'max_depth': 4, 'n_estimators': 100, 'random_state': 42}
```

In [44]:

```
best_gb = gb_grid.best_estimator_
gb_class = best_gb.predict(X_test)
#ac_gb = accuracy_score(y_test, gb_class)
#pc_gb = precision_score(y_test, gb_class, average = 'macro')
#rc_gb = recall_score(y_test, gb_class, average = 'macro')
#f_macro_gb = f1_score(y_test, gb_class, average = 'macro', labels=np.unique(gb_class))
#f_weight_gb = f1_score(y_test, gb_class, average = 'weighted', labels=np.unique(gb_class))
#rf_metric_gb = np.array([ac_gb, pc_gb, rc_gb, f_macro_gb, f_weight_gb])
#print('Accuracy:', rf_metric_gb[0])
#print('Precision:', rf_metric_gb[1])
#print('Recall:', rf_metric_gb[2])
#print('f1_macro:', rf_metric_gb[3])
#print('f1_weighted:', rf_metric_gb[4])
#print('average_metric:', np.mean(rf_metric_gb))
```

In [45]:

```
gb_report = classification_report(y_test, gb_class)
```

```
print('test_result: \n', gb_report)
```

```
test_result:
               precision    recall  f1-score   support

           0       0.73      0.80      0.76      1012
           1       0.77      0.69      0.73       988

    accuracy                           0.74      2000
   macro avg       0.75      0.74      0.74      2000
weighted avg       0.75      0.74      0.74      2000
```

In [46]:

```
gb_train = best_gb.predict(X_train)
```

In [47]:

```
gb_report_train = classification_report(y_train, gb_train)
print('test_result: \n', gb_report_train)
```

```
test_result:
               precision    recall  f1-score   support

           0       0.84      0.89      0.86      3988
           1       0.88      0.84      0.86      4012

    accuracy                           0.86      8000
   macro avg       0.86      0.86      0.86      8000
weighted avg       0.86      0.86      0.86      8000
```

In [54]:

```
gb_model = GradientBoostingClassifier(learning_rate=0.2, max_depth=2, n_estimators=100, random_stat
e=42)
gb_model.fit(X_train, y_train)
```

Out[54]:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.2, loss='deviance', max_depth=2,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='auto',
                           random_state=42, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

In [55]:

```
gb_test = gb_model.predict(X_test)
```

In [56]:

```
gb_report_test = classification_report(y_test, gb_test)
print('test_result: \n', gb_report_test)
```

```
test_result:
               precision    recall  f1-score   support

           0       0.67      0.84      0.74      1012
           1       0.78      0.57      0.66       988

    accuracy                           0.71      2000
   macro avg       0.72      0.70      0.70      2000
weighted avg       0.72      0.71      0.70      2000
```

In [57]:

```
gb_train_2 = gb_model.predict(X_train)
```

In [58]:

```
gb_report_train2 = classification_report(y_train, gb_train_2)
print('train_result: \n', gb_report_train2)
```

```
train_result:
               precision    recall  f1-score   support

           0       0.69      0.86      0.76      3988
           1       0.81      0.61      0.70      4012

    accuracy                           0.73      8000
   macro avg       0.75      0.74      0.73      8000
weighted avg       0.75      0.73      0.73      8000
```

LSTM

In [28]:

```
def plot_history(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    x = range(1, len(acc) + 1)

    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(x, acc, 'b', label='Training acc')
    plt.plot(x, val_acc, 'r', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.legend()
    plt.subplot(1, 2, 2)
    plt.plot(x, loss, 'b', label='Training loss')
    plt.plot(x, val_loss, 'r', label='Validation loss')
    plt.title('Training and validation loss')
    plt.legend()
```

In [29]:

```
#### Base Model

m_1 = Sequential()
m_1.add(Embedding(input_dim = 18866, output_dim = 64, input_length = 2048))
m_1.add(LSTM(units = 64, recurrent_dropout = 0.5))
m_1.add(Dense(units = 1, activation = 'sigmoid'))
m_1.compile(loss = "binary_crossentropy", optimizer = "adam", metrics = ['accuracy'])
m_1.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 2048, 64)          1207424
_________________________________________________________________
lstm_1 (LSTM)                (None, 64)                33024
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
Total params: 1,240,513
Trainable params: 1,240,513
Non-trainable params: 0
_________________________________________________________________
```

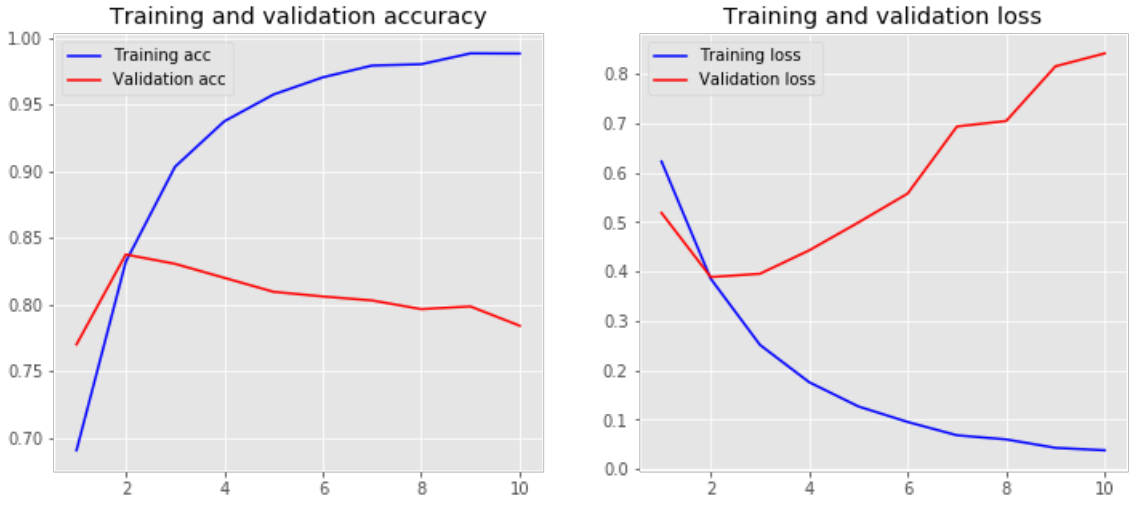
In [30]:

```
h_1 = m_1.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=128)
```

```
Epoch 1/10
63/63 [==============================] - 110s 2s/step - loss: 0.6676 - accuracy: 0.6226 -
val_loss: 0.5184 - val_accuracy: 0.7700
Epoch 2/10
63/63 [==============================] - 108s 2s/step - loss: 0.4212 - accuracy: 0.8172 -
val_loss: 0.3886 - val_accuracy: 0.8375
Epoch 3/10
63/63 [==============================] - 108s 2s/step - loss: 0.2504 - accuracy: 0.9001 -
val_loss: 0.3953 - val_accuracy: 0.8305
Epoch 4/10
63/63 [==============================] - 108s 2s/step - loss: 0.1668 - accuracy: 0.9465 -
val_loss: 0.4427 - val_accuracy: 0.8200
Epoch 5/10
63/63 [==============================] - 109s 2s/step - loss: 0.1179 - accuracy: 0.9617 -
val_loss: 0.4992 - val_accuracy: 0.8095
Epoch 6/10
63/63 [==============================] - 109s 2s/step - loss: 0.0912 - accuracy: 0.9753 -
val_loss: 0.5577 - val_accuracy: 0.8060
Epoch 7/10
63/63 [==============================] - 108s 2s/step - loss: 0.0584 - accuracy: 0.9841 -
val_loss: 0.6933 - val_accuracy: 0.8030
Epoch 8/10
63/63 [==============================] - 112s 2s/step - loss: 0.0559 - accuracy: 0.9821 -
val_loss: 0.7044 - val_accuracy: 0.7965
Epoch 9/10
63/63 [==============================] - 118s 2s/step - loss: 0.0392 - accuracy: 0.9909 -
val_loss: 0.8150 - val_accuracy: 0.7985
Epoch 10/10
63/63 [==============================] - 121s 2s/step - loss: 0.0305 - accuracy: 0.9919 -
val_loss: 0.8411 - val_accuracy: 0.7840
```

In [32]:

```
plot_history(h_1)
```



In [43]:

```
### Second Model with dropout
m_2 = Sequential()
m_2.add(Embedding(input_dim = 18866, output_dim = 64, input_length = 2048))
m_2.add(SpatialDropout1D(0.2))
m_2.add(LSTM(units = 64, recurrent_dropout = 0.5))
m_2.add(Dropout(0.2))
m_2.add(Dense(units = 1, activation = 'sigmoid'))
m_2.compile(loss = "binary_crossentropy", optimizer = "adam", metrics = ['accuracy'])
m_2.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_9 (Embedding)      (None, 2048, 64)          1207424
_________________________________________________________________
spatial_dropout1d_6 (Spatial (None, 2048, 64)          0
_________________________________________________________________
lstm_9 (LSTM)                (None, 64)                33024
_________________________________________________________________
dropout_9 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_13 (Dense)             (None, 1)                 65
=================================================================
Total params: 1,240,513
Trainable params: 1,240,513
Non-trainable params: 0
_________________________________________________________________
```

In [44]:

```
h_2 = m_2.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=128)
```

```
Epoch 1/10
63/63 [==============================] - 108s 2s/step - loss: 0.6709 - accuracy: 0.5719 -
val_loss: 0.4679 - val_accuracy: 0.7900
Epoch 2/10
63/63 [==============================] - 107s 2s/step - loss: 0.4601 - accuracy: 0.8117 -
val_loss: 0.3803 - val_accuracy: 0.8330
Epoch 3/10
63/63 [==============================] - 107s 2s/step - loss: 0.2587 - accuracy: 0.9028 -
val_loss: 0.3957 - val_accuracy: 0.8245
Epoch 4/10
63/63 [==============================] - 107s 2s/step - loss: 0.1856 - accuracy: 0.9378 -
val_loss: 0.4463 - val_accuracy: 0.8200
Epoch 5/10
63/63 [==============================] - 107s 2s/step - loss: 0.1284 - accuracy: 0.9614 -
val_loss: 0.5372 - val_accuracy: 0.8110
Epoch 6/10
63/63 [==============================] - 107s 2s/step - loss: 0.0955 - accuracy: 0.9709 -
val_loss: 0.5614 - val_accuracy: 0.8135
Epoch 7/10
63/63 [==============================] - 108s 2s/step - loss: 0.0801 - accuracy: 0.9770 -
val_loss: 0.6910 - val_accuracy: 0.7965
Epoch 8/10
63/63 [==============================] - 586s 9s/step - loss: 0.0614 - accuracy: 0.9819 -
val_loss: 0.7297 - val_accuracy: 0.7945
Epoch 9/10
63/63 [==============================] - 108s 2s/step - loss: 0.0495 - accuracy: 0.9868 -
val_loss: 0.8668 - val_accuracy: 0.7910
Epoch 10/10
63/63 [==============================] - 107s 2s/step - loss: 0.0383 - accuracy: 0.9886 -
val_loss: 0.8373 - val_accuracy: 0.7860
```

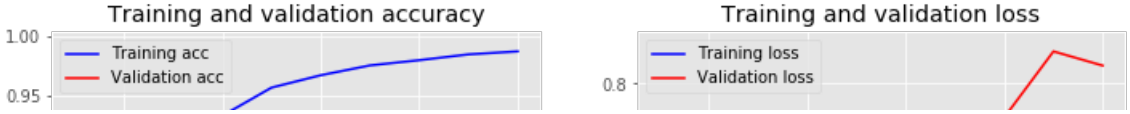
In [46]:

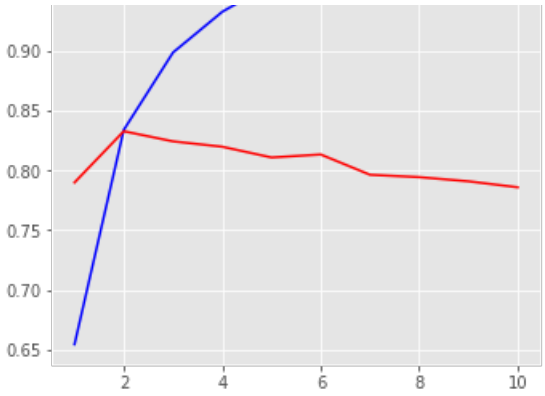
```
loss, accuracy = m_2.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = m_2.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
```

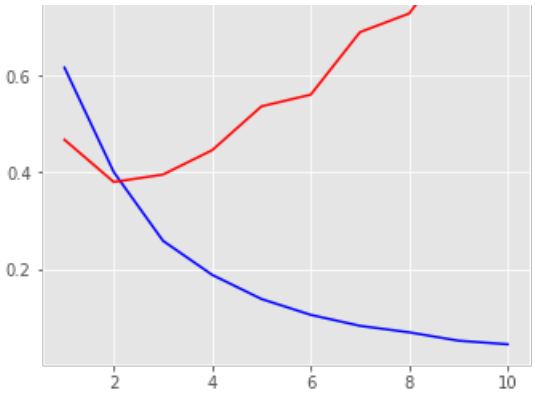
```
Training Accuracy: 0.9931
Testing Accuracy:  0.7860
```

In [45]:

```
plot_history(h_2)
```







In [49]:

```
### Third Model - less complex model higher dropout index

m_3 = Sequential()
m_3.add(Embedding(input_dim = 18866, output_dim = 32, input_length = 2048))
m_3.add(SpatialDropout1D(0.5))
m_3.add(LSTM(units = 32, recurrent_dropout = 0.5))
m_3.add(Dropout(0.5))
m_3.add(Dense(units = 1, activation = 'sigmoid'))
m_3.compile(loss = "binary_crossentropy", optimizer = "adam", metrics = ['accuracy'])
m_3.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_11 (Embedding)     (None, 2048, 32)          603712
_________________________________________________________________
spatial_dropout1d_8 (Spatial (None, 2048, 32)          0
_________________________________________________________________
lstm_11 (LSTM)               (None, 32)                8320
_________________________________________________________________
dropout_11 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 33
=================================================================
Total params: 612,065
Trainable params: 612,065
Non-trainable params: 0
_________________________________________________________________
```

In [50]:

```
h_3 = m_3.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=128)
```

```
Epoch 1/10
63/63 [==============================] - 87s 1s/step - loss: 0.6883 - accuracy: 0.5662 - val_loss:
0.6163 - val_accuracy: 0.7060
Epoch 2/10
63/63 [==============================] - 86s 1s/step - loss: 0.5132 - accuracy: 0.7911 - val_loss:
0.4112 - val_accuracy: 0.8210
Epoch 3/10
63/63 [==============================] - 85s 1s/step - loss: 0.3521 - accuracy: 0.8640 - val_loss:
0.3846 - val_accuracy: 0.8310
Epoch 4/10
63/63 [==============================] - 85s 1s/step - loss: 0.2786 - accuracy: 0.8968 - val_loss:
0.3919 - val_accuracy: 0.8315
Epoch 5/10
63/63 [==============================] - 84s 1s/step - loss: 0.2322 - accuracy: 0.9152 - val_loss:
0.4045 - val_accuracy: 0.8315
Epoch 6/10
63/63 [==============================] - 86s 1s/step - loss: 0.1933 - accuracy: 0.9318 - val_loss:
0.4533 - val_accuracy: 0.8320
Epoch 7/10
63/63 [==============================] - 85s 1s/step - loss: 0.1630 - accuracy: 0.9443 - val_loss:
0.5715 - val_accuracy: 0.8155
Epoch 8/10
```

```
63/63 [==============================] - 84s 1s/step - loss: 0.1403 - accuracy: 0.9545 - val_loss:
0.5350 - val_accuracy: 0.8205
Epoch 9/10
63/63 [==============================] - 84s 1s/step - loss: 0.1251 - accuracy: 0.9579 - val_loss:
0.5528 - val_accuracy: 0.8130
Epoch 10/10
63/63 [==============================] - 85s 1s/step - loss: 0.1049 - accuracy: 0.9669 - val_loss:
0.5828 - val_accuracy: 0.8080
```

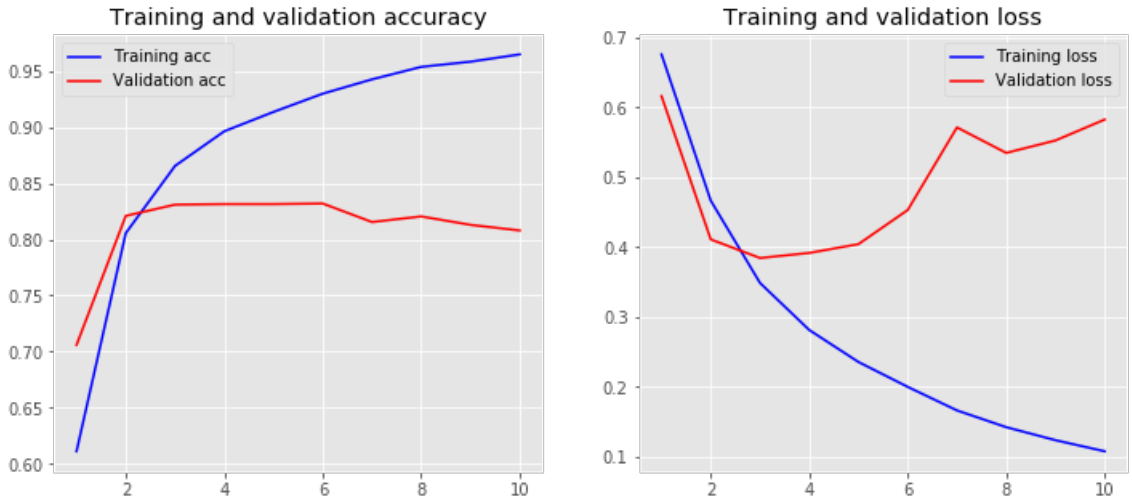
In [51]:

```
loss, accuracy = m_3.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = m_3.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
```

```
Training Accuracy: 0.9834
Testing Accuracy:  0.8080
```

In [52]:

```
plot_history(h_3)
```



In [53]:

```
### fourth Model

m_4 = Sequential()
m_4.add(Embedding(input_dim = 18866, output_dim = 16, input_length = 2048))
m_4.add(SpatialDropout1D(0.5))
m_4.add(LSTM(units = 16, recurrent_dropout = 0.5))
m_4.add(Dropout(0.5))
m_4.add(Dense(units = 1, activation = 'sigmoid'))
m_4.compile(loss = "binary_crossentropy", optimizer = "adam", metrics = ['accuracy'])
m_4.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_12 (Embedding)     (None, 2048, 16)          301856
_________________________________________________________________
spatial_dropout1d_9 (Spatial (None, 2048, 16)          0
_________________________________________________________________
lstm_12 (LSTM)               (None, 16)                2112
_________________________________________________________________
dropout_12 (Dropout)         (None, 16)                0
_________________________________________________________________
dense_16 (Dense)             (None, 1)                 17
=================================================================
Total params: 303,985
Trainable params: 303,985
Non-trainable params: 0
```

Non-trainable params: 0
_____

In [54]:

```
h_4 = m_4.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=128)
```

```
Epoch 1/10
63/63 [==============================] - 77s 1s/step - loss: 0.6906 - accuracy: 0.5644 - val_loss:
0.6657 - val_accuracy: 0.6695
Epoch 2/10
63/63 [==============================] - 75s 1s/step - loss: 0.6236 - accuracy: 0.7170 - val_loss:
0.4800 - val_accuracy: 0.8005
Epoch 3/10
63/63 [==============================] - 74s 1s/step - loss: 0.4437 - accuracy: 0.8364 - val_loss:
0.4051 - val_accuracy: 0.8175
Epoch 4/10
63/63 [==============================] - 74s 1s/step - loss: 0.3750 - accuracy: 0.8595 - val_loss:
0.4071 - val_accuracy: 0.8170
Epoch 5/10
63/63 [==============================] - 74s 1s/step - loss: 0.3238 - accuracy: 0.8790 - val_loss:
0.3921 - val_accuracy: 0.8295
Epoch 6/10
63/63 [==============================] - 74s 1s/step - loss: 0.2785 - accuracy: 0.9021 - val_loss:
0.3995 - val_accuracy: 0.8270
Epoch 7/10
63/63 [==============================] - 74s 1s/step - loss: 0.2350 - accuracy: 0.9265 - val_loss:
0.4181 - val_accuracy: 0.8305
Epoch 8/10
63/63 [==============================] - 75s 1s/step - loss: 0.2303 - accuracy: 0.9294 - val_loss:
0.4428 - val_accuracy: 0.8230
Epoch 9/10
63/63 [==============================] - 74s 1s/step - loss: 0.1887 - accuracy: 0.9387 - val_loss:
0.4497 - val_accuracy: 0.8200
Epoch 10/10
63/63 [==============================] - 73s 1s/step - loss: 0.1744 - accuracy: 0.9460 - val_loss:
0.4711 - val_accuracy: 0.8175
```

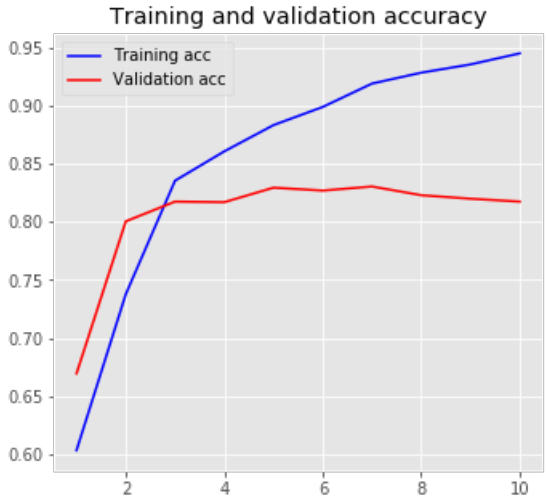
In [55]:

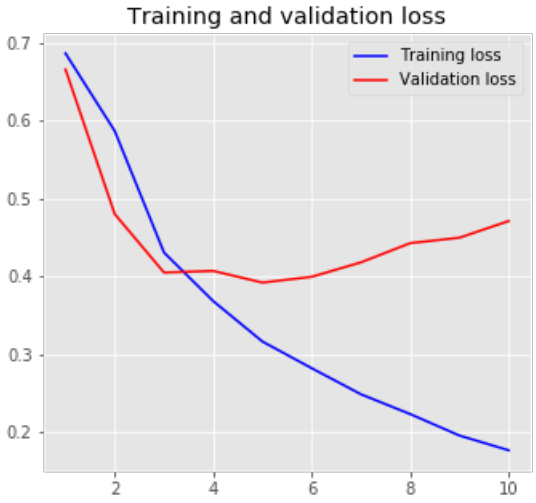
```
loss, accuracy = m_4.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = m_4.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
```

```
Training Accuracy: 0.9685
Testing Accuracy:  0.8175
```

In [56]:

```
plot_history(h_4)
```





In [57]:

```
# fifth model - simple RNN

m_5 = Sequential()
m_5.add(Embedding(input_dim = 18866, output_dim = 32, input_length = 2048))
m_5.add(SpatialDropout1D(0.5))
m_5.add(SimpleRNN(units = 32, activation = 'relu'))
m_5.add(Dropout(0.5))
m_5.add(Dense(units = 1, activation = 'sigmoid'))
m_5.compile(loss = "binary_crossentropy", optimizer = "adam", metrics = ['accuracy'])
m_5.summary()
```

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_13 (Embedding)     (None, 2048, 32)          603712
_________________________________________________________________
spatial_dropout1d_10 (Spatia (None, 2048, 32)          0
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, 32)                2080
_________________________________________________________________
dropout_13 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_17 (Dense)             (None, 1)                 33
=================================================================
Total params: 605,825
Trainable params: 605,825
Non-trainable params: 0
_________________________________________________________________
```

In [58]:

```
h_5 = m_5.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=128)
```

```
Epoch 1/10
63/63 [==============================] - 31s 479ms/step - loss: 0.6930 - accuracy: 0.5109 - val_lo
ss: 0.6882 - val_accuracy: 0.5880
Epoch 2/10
63/63 [==============================] - 30s 477ms/step - loss: 0.6839 - accuracy: 0.5813 - val_lo
ss: 0.6583 - val_accuracy: 0.7045
Epoch 3/10
63/63 [==============================] - 30s 477ms/step - loss: 0.5997 - accuracy: 0.7121 - val_lo
ss: 0.4522 - val_accuracy: 0.8005
Epoch 4/10
63/63 [==============================] - 30s 480ms/step - loss: 0.4300 - accuracy: 0.8250 - val_lo
ss: 0.4019 - val_accuracy: 0.8190
Epoch 5/10
63/63 [==============================] - 30s 478ms/step - loss: 0.3199 - accuracy: 0.8766 - val_lo
ss: 0.3980 - val_accuracy: 0.8175
Epoch 6/10
63/63 [==============================] - 30s 480ms/step - loss: 0.2701 - accuracy: 0.9048 - val_lo
ss: 0.4260 - val_accuracy: 0.8280
Epoch 7/10
63/63 [==============================] - 33s 523ms/step - loss: 0.2311 - accuracy: 0.9192 - val_lo
ss: 0.4228 - val_accuracy: 0.8240
Epoch 8/10
63/63 [==============================] - 33s 519ms/step - loss: 0.1975 - accuracy: 0.9317 - val_lo
ss: 0.4418 - val_accuracy: 0.8155
Epoch 9/10
63/63 [==============================] - 31s 488ms/step - loss: 0.1691 - accuracy: 0.9422 - val_lo
ss: 0.4363 - val_accuracy: 0.8095
Epoch 10/10
63/63 [==============================] - 32s 511ms/step - loss: 0.1592 - accuracy: 0.9511 - val_lo
ss: 0.4832 - val_accuracy: 0.8140
```

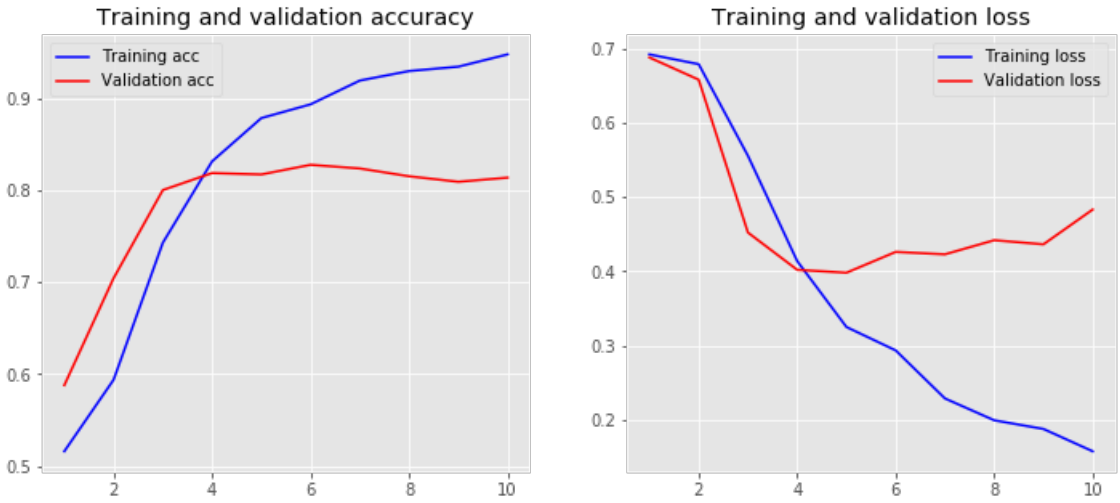
In [59]:

```
loss, accuracy = m_5.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = m_5.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
```

```
Training Accuracy: 0.9769
Testing Accuracy:  0.8140
```

In [60]:

```
plot_history(h_5)
```



Best LSTM

In [61]:

```
## best lstm model with epoch 5
m_ls = Sequential()
m_ls.add(Embedding(input_dim = 18866, output_dim = 16, input_length = 2048))
m_ls.add(SpatialDropout1D(0.5))
m_ls.add(LSTM(units = 16, recurrent_dropout = 0.5))
m_ls.add(Dropout(0.5))
m_ls.add(Dense(units = 1, activation = 'sigmoid'))
m_ls.compile(loss = "binary_crossentropy", optimizer = "adam", metrics = ['accuracy'])
m_ls.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_14 (Embedding)     (None, 2048, 16)          301856
_________________________________________________________________
spatial_dropout1d_11 (Spatia (None, 2048, 16)          0
_________________________________________________________________
lstm_13 (LSTM)               (None, 16)                2112
_________________________________________________________________
dropout_14 (Dropout)         (None, 16)                0
_________________________________________________________________
dense_18 (Dense)             (None, 1)                 17
=================================================================
Total params: 303,985
Trainable params: 303,985
Non-trainable params: 0
_________________________________________________________________
```

In [62]:

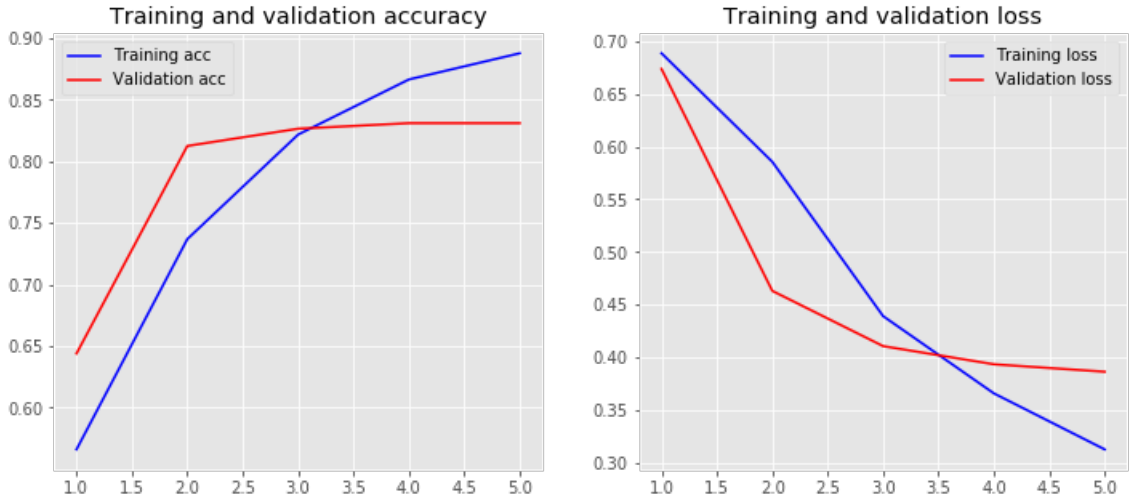
```
h_ls = m_ls.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=5, batch_size=128)
```

```
Epoch 1/5
63/63 [==============================] - 76s 1s/step - loss: 0.6913 - accuracy: 0.5320 - val_loss:
0.6738 - val_accuracy: 0.6440
Epoch 2/5
63/63 [==============================] - 75s 1s/step - loss: 0.6278 - accuracy: 0.7089 - val_loss:
```

```
0.4631 - val_accuracy: 0.8125
Epoch 3/5
63/63 [==============================] - 73s 1s/step - loss: 0.4521 - accuracy: 0.8142 - val_loss:
0.4107 - val_accuracy: 0.8265
Epoch 4/5
63/63 [==============================] - 72s 1s/step - loss: 0.3709 - accuracy: 0.8653 - val_loss:
0.3934 - val_accuracy: 0.8310
Epoch 5/5
63/63 [==============================] - 72s 1s/step - loss: 0.3106 - accuracy: 0.8917 - val_loss:
0.3864 - val_accuracy: 0.8310
```

In [63]:

```
plot_history(h_ls)
```



In [65]:

```
loss, accuracy = m_ls.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = m_ls.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
```

```
Training Accuracy: 0.9276
Testing Accuracy:  0.8310
```

In [71]:

```
ls_pred = (m_ls.predict(X_test) > 0.5).astype("int32")
```

In [72]:

```
ls_report = classification_report(y_test, ls_pred)
print('ls_result: \n', ls_report)
```

```
ls_result:
              precision    recall  f1-score   support

           0       0.83      0.84      0.83      1012
           1       0.83      0.82      0.83       988

    accuracy                           0.83      2000
   macro avg       0.83      0.83      0.83      2000
weighted avg       0.83      0.83      0.83      2000
```

In [73]:

```
ls_train = (m_ls.predict(X_train) > 0.5).astype("int32")
```

In [74]:

```
ls_report_train = classification_report(y_train, ls_train)
print('ls_result: \n', ls_report_train)
```

```
ls_result:
               precision    recall  f1-score   support

           0       0.93      0.93      0.93      3988
           1       0.93      0.93      0.93      4012

    accuracy                           0.93      8000
   macro avg       0.93      0.93      0.93      8000
weighted avg       0.93      0.93      0.93      8000
```

Best simpleRNN

In [75]:

```
# fifth model - simple RNN

m_sr = Sequential()
m_sr.add(Embedding(input_dim = 18866, output_dim = 32, input_length = 2048))
m_sr.add(SpatialDropout1D(0.5))
m_sr.add(SimpleRNN(units = 32, activation = 'relu'))
m_sr.add(Dropout(0.5))
m_sr.add(Dense(units = 1, activation = 'sigmoid'))
m_sr.compile(loss = "binary_crossentropy", optimizer = "adam", metrics = ['accuracy'])
m_sr.summary()
```

```
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_15 (Embedding)     (None, 2048, 32)          603712
_________________________________________________________________
spatial_dropout1d_12 (Spatia (None, 2048, 32)          0
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 32)                2080
_________________________________________________________________
dropout_15 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_19 (Dense)             (None, 1)                 33
=================================================================
Total params: 605,825
Trainable params: 605,825
Non-trainable params: 0
_________________________________________________________________
```

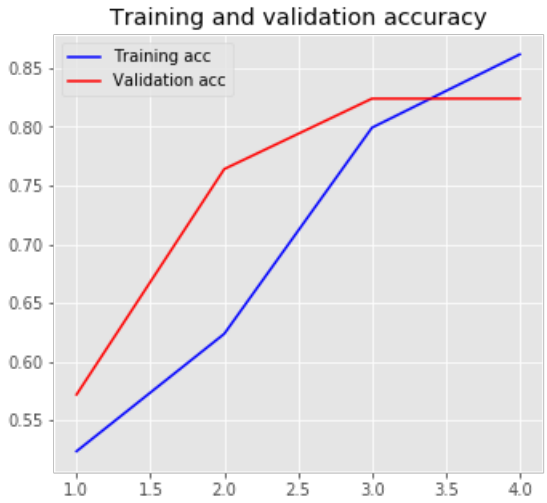
In [76]:

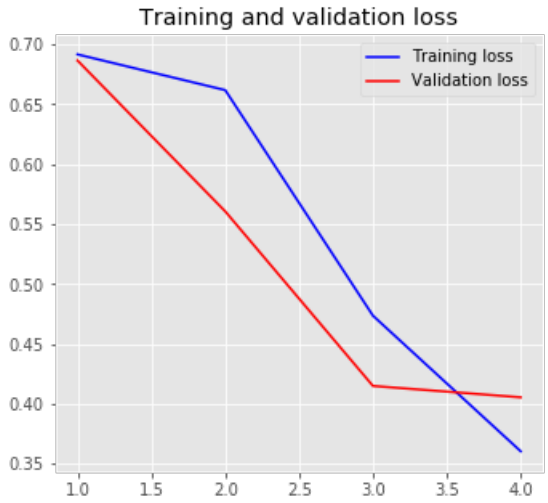
```
h_sr = m_sr.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=4, batch_size=128)
```

```
Epoch 1/4
63/63 [==============================] - 31s 482ms/step - loss: 0.6933 - accuracy: 0.5083 - val_lo
ss: 0.6866 - val_accuracy: 0.5720
Epoch 2/4
63/63 [==============================] - 30s 480ms/step - loss: 0.6780 - accuracy: 0.5892 - val_lo
ss: 0.5607 - val_accuracy: 0.7640
Epoch 3/4
63/63 [==============================] - 30s 482ms/step - loss: 0.5086 - accuracy: 0.7788 - val_lo
ss: 0.4150 - val_accuracy: 0.8240
Epoch 4/4
63/63 [==============================] - 30s 482ms/step - loss: 0.3560 - accuracy: 0.8660 - val_lo
ss: 0.4055 - val_accuracy: 0.8240
```

In [77]:

```
plot_history(h_sr)
```





In [78]:

```
loss, accuracy = m_sr.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = m_sr.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
```

```
Training Accuracy: 0.9078
Testing Accuracy:  0.8240
```

In [79]:

```
sr_pred = (m_sr.predict(X_test) > 0.5).astype("int32")
```

In [81]:

```
sr_report = classification_report(y_test, sr_pred)
print('ls_result: \n', sr_report)
```

```
ls_result:
              precision    recall  f1-score   support

           0       0.87      0.77      0.82      1012
           1       0.79      0.88      0.83       988

    accuracy                           0.82      2000
   macro avg       0.83      0.82      0.82      2000
weighted avg       0.83      0.82      0.82      2000
```

In [82]:

```
sr_train = (m_sr.predict(X_train) > 0.5).astype("int32")
```

In [83]:

```
sr_report_train = classification_report(y_train, sr_train)
print('ls_result_train: \n', sr_report_train)
```

```
ls_result_train:
              precision    recall  f1-score   support

           0       0.93      0.88      0.90      3988
           1       0.88      0.94      0.91      4012

    accuracy                           0.91      8000
   macro avg       0.91      0.91      0.91      8000
weighted avg       0.91      0.91      0.91      8000
```