Lab Seven: RNNs

Yang Shen

Data overview

This data set is the review given by customer to the hotel. And also have variable Is_response which show whether customer is happy with the hotel services or not. 0 is unhappy and 1 is happy. There are 38932 records and no na value.

In [2]:

```
#https://www.kaggle.com/anu0012/hotel-review
#0 unhappy
#1 happy
```

In [18]:

```
import pandas as pd
import numpy as np
dfraw = pd.read_csv('hotel.csv')
print(dfraw.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38932 entries, 0 to 38931
Data columns (total 5 columns):
User_ID         38932 non-null object
Description     38932 non-null object
Browser_Used    38932 non-null object
Device_Used     38932 non-null object
Is_Response     38932 non-null int64
dtypes: int64(1), object(4)
memory usage: 1.5+ MB
None
```

Id, browser_used, Device_used are not useful, I only want to analysis the text variable and the reponse.

In [19]:

```
df = dfraw[['Description','Is_Response']]
df.dropna(inplace=True)
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 38932 entries, 0 to 38931
Data columns (total 2 columns):
Description    38932 non-null object
Is_Response    38932 non-null int64
dtypes: int64(1), object(1)
memory usage: 912.5+ KB
None
```

```
D:\APP\conda\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/
indexing.html#indexing-view-versus-copy
```

In [20]:

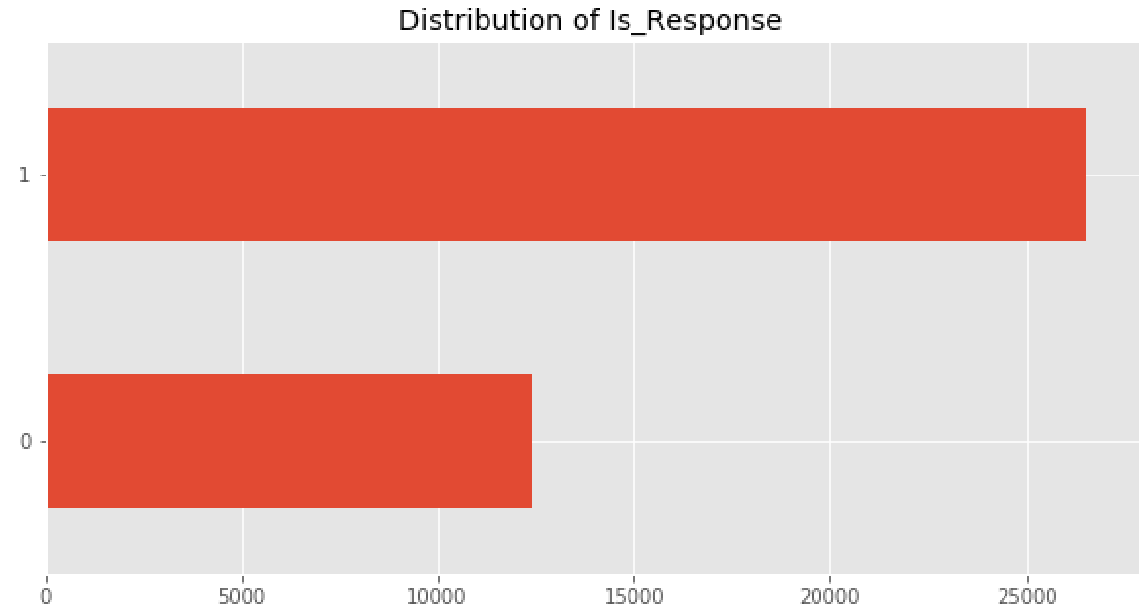
```
X = df['Description']
y = df['Is_Response']
df.head()
```

Out[20]:

	Description	Is_Response
0	The room was kind of clean but had a VERY stro...	0
1	I stayed at the Crown Plaza April -- - April -...	0
2	I booked this hotel through Hotwire at the low...	0
3	Stayed here with husband and sons on the way t...	1
4	My girlfriends and I stayed here to celebrate ...	0

In [25]:

```
import matplotlib
import matplotlib.pyplot as plt
import warnings
warnings.simplefilter('ignore', DeprecationWarning)
%matplotlib inline
plt.style.use('ggplot')

plt.figure(figsize=(10,5))
df['Is_Response'].value_counts(sort=True,ascending=True).plot(kind='barh')
plt.title('Distribution of Is_Response')

plt.show()
```



There are more happy response than unhappy response, about two times than unhappy.

In [21]:

```
#https://github.com/Thakugan/machine-learning-notebooks/blob/master/8-recurrent-neural-networks/
spam.ipynb
#there i want to show the length of the text
length = lambda X: len(X)
df["text_length"] = df["Description"].map(length)  # add a column indicating how long a text
print(df.head(10))
print("Mean length ",df["text_length"].mean())#print the mean length
```

```
                                         Description  Is_Response  text_length
0  The room was kind of clean but had a VERY stro...            0          248
1  I stayed at the Crown Plaza April -- - April -...            0         1077
2  I booked this hotel through Hotwire at the low...            0         1327
3  Stayed here with husband and sons on the way t...            1          502
4  My girlfriends and I stayed here to celebrate ...            0         1613
5  We had - rooms. One was very nice and clearly ...            1          610
6  My husband and I have stayed in this hotel a f...            0          492
7  My wife & I stayed in this glorious city a whi...            1          935
8  My boyfriend and I stayed at the Fairmont on a...            1          641
9  Wonderful staff, great location, but it was de...            0          358
Mean length  866.3808178362273

D:\APP\conda\lib\site-packages\ipykernel_launcher.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/
indexing.html#indexing-view-versus-copy
  after removing the cwd from sys.path.
```

In [22]:

```
import keras
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

NUM_TOP_WORDS = None #every unique word is ohe
MAX_ART_LEN = 1000 # maximum number of words

tokenizer = Tokenizer(num_words=NUM_TOP_WORDS)
tokenizer.fit_on_texts(X.tolist())
sequences = tokenizer.texts_to_sequences(X.tolist())

word_index = tokenizer.word_index
NUM_TOP_WORDS = len(word_index) if NUM_TOP_WORDS==None else NUM_TOP_WORDS
top_words = min((len(word_index),NUM_TOP_WORDS))
print('Found %s unique tokens. Distilled to %d top words.' % (len(word_index),top_words))

X = pad_sequences(sequences, maxlen=MAX_ART_LEN)

y_ohe = keras.utils.to_categorical(y)
print('Shape of data tensor:', X.shape)
print('Shape of label tensor:', y_ohe.shape)
print(np.max(X))
```

```
Found 49048 unique tokens. Distilled to 49048 top words.
Shape of data tensor: (38932, 1000)
Shape of label tensor: (38932, 2)
49048
```

After show the mean length of text is about 860, So I think set the maximum lenght of text to 1000 will be reasonable. And also I want every unique word is ohe. So the tokenizer will fit on every word and return ohe of text.

Dividing training and testing

I will use k-fold becasue I have sbout 30000 rows in my dataset. I think 30000 is not enough to represent the features of dataset. Maybe 50000 is enough. I used 5 fold StratifiedKFold, because I need to use cross validation method and also I want all my response have the same ratio in each fold. becasue happy response are twice more than unhappy response. I used both 80/20 and StratifiedKFold as I need. 80/20 split is just for testing the model if it work.

In [23]:

```
from sklearn.model_selection import train_test_split
# Split it into train / test subsets
X_train, X_test, y_train_ohe, y_test_ohe = train_test_split(X, y_ohe, test_size=0.2,
                                                            stratify=y,
                                                            random_state=42)

X1_train, X1_test, y_train, y_test = train_test_split(X, y, test_size=0.2,stratify=y,
                                                            random_state=42)

NUM_CLASSES = 2
print(X_train.shape,y_train_ohe.shape)
print(np.sum(y_train_ohe,axis=0))
```

```
(31145, 1000) (31145, 2)
[ 9929. 21216.]
```

Metric

The third party who interested in this result is hotels. They have millions of room reservations in a year. The quality of hotel service is important for the them. Customer will leave reviews on the website or with phone apps. Some customer will only leave the review without rating. So that we need a lot of human resources to check their reviews to know whether they are happy or not. My task is to predict the response of customers base on their reviews.

The performance of our model needs to be accurate in order to find out which the target reviews which are those who are not happy with hotel services, so we can send people to look their reviews and make some improvement. My metric is accuracy. Becasue my task is prediciton which class the review is. The accuracy directly reflect what is the ratio of my correction prediction. All other false positive and false negative are meaningless, because if it get wrong, the review need to check by human. The only correct classification can save time from human checking. If my accuracy is high, hotel can only check reviews which is unhappy, and do not need to check all reviews.

Embedding

In [26]:

```
EMBED_SIZE = 100
# the embed size should match the file you load glove from
embeddings_index = {}
f = open('glove.6B.100d.txt', encoding='utf8')
# save key/array pairs of the embeddings
#  the key of the dictionary is the word, the array is the embedding
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Found %s word vectors.' % len(embeddings_index))

# now fill in the matrix, using the ordering from the
#  keras word tokenizer from before
embedding_matrix = np.zeros((len(word_index) + 1, EMBED_SIZE))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector

print(embedding_matrix.shape)
```

```
Found 400000 word vectors.
(49049, 100)
```

In [ ]:

In [27]:

```
from keras.layers import Embedding

embedding_layer = Embedding(len(word_index) + 1,
                            EMBED_SIZE,
                            weights=[embedding_matrix],
                            input_length=MAX_ART_LEN,
                            trainable=False)
```

Model1

In [28]:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM

def rnn1_create():
    rnn1 = Sequential()
    rnn1.add(embedding_layer)
    rnn1.add(LSTM(100,dropout=0.2, recurrent_dropout=0.2))
    rnn1.add(Dense(NUM_CLASSES, activation='sigmoid'))
    rnn1.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

    return rnn1

rnn1 = rnn1_create()

print(rnn1.summary())
```

```
WARNING:tensorflow:From D:\APP\conda\lib\site-packages\tensorflow\python\framework
\op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is de
precated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From D:\APP\conda\lib\site-packages\keras\backend\tensorflow_ba
ckend.py:3445: calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob
is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_p
rob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 1000, 100)         4904900
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               80400
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 202
=================================================================
Total params: 4,985,502
Trainable params: 80,602
Non-trainable params: 4,904,900
_________________________________________________________________
None
```

In [62]:

```
history1 = rnn1.fit(X_train, y_train_ohe, validation_data=(X_test, y_test_ohe), epochs=3, batch_
size=64)
```

```
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 385s 12ms/step - loss: 0.5268 - ac
c: 0.7400 - val_loss: 0.4837 - val_acc: 0.7983
Epoch 2/3
31145/31145 [==============================] - 387s 12ms/step - loss: 0.3964 - ac
c: 0.8260 - val_loss: 0.3217 - val_acc: 0.8684
Epoch 3/3
31145/31145 [==============================] - 383s 12ms/step - loss: 0.3455 - ac
c: 0.8515 - val_loss: 0.3068 - val_acc: 0.8734
```

In [72]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html#
sklearn.model_selection.StratifiedKFold.split
#https://github.com/Thakugan/machine-learning-notebooks/blob/master/6-wide-and-deep-networks/mus
hroom-hunting.ipynb
#I used this in last lab, just modified to new version
from sklearn.model_selection import StratifiedKFold
num_folds = 5

acc_scores1 =[]
skf1 = StratifiedKFold(n_splits=num_folds, shuffle=True)
for i, (train, test) in enumerate(skf1.split(X, y)):
    rnn1 = rnn1_create()
    #doing modeling same as above, without for loop

    rnn1.fit(X_train, y_train_ohe, validation_data=(X_test, y_test_ohe), epochs=3, batch_size=64
)
    #this is just what i do without cross validation
    yhat = np.argmax(rnn1.predict(X_test), axis=1)

    acc_score1 = mt.accuracy_score(y_test,yhat)
    acc_scores1.append(acc_score1)
    print("Accuracy: ", acc_score1)
print(acc_scores1)
```

```
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 387s 12ms/step - loss: 0.5312 - ac
c: 0.7376 - val_loss: 0.4173 - val_acc: 0.8146
Epoch 2/3
31145/31145 [==============================] - 386s 12ms/step - loss: 0.4009 - ac
c: 0.8234 - val_loss: 0.3296 - val_acc: 0.8609
Epoch 3/3
31145/31145 [==============================] - 384s 12ms/step - loss: 0.3500 - ac
c: 0.8502 - val_loss: 0.3104 - val_acc: 0.8683
Accuracy:  0.8686271991781174
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 380s 12ms/step - loss: 0.5327 - ac
c: 0.7338 - val_loss: 0.4007 - val_acc: 0.8323
Epoch 2/3
31145/31145 [==============================] - 380s 12ms/step - loss: 0.4015 - ac
c: 0.8238 - val_loss: 0.3324 - val_acc: 0.8604
Epoch 3/3
31145/31145 [==============================] - 377s 12ms/step - loss: 0.3525 - ac
c: 0.8493 - val_loss: 0.3061 - val_acc: 0.8688
Accuracy:  0.869654552459227
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 384s 12ms/step - loss: 0.5361 - ac
c: 0.7322 - val_loss: 0.3898 - val_acc: 0.8399
Epoch 2/3
31145/31145 [==============================] - 377s 12ms/step - loss: 0.4037 - ac
c: 0.8227 - val_loss: 0.3601 - val_acc: 0.8504
Epoch 3/3
31145/31145 [==============================] - 380s 12ms/step - loss: 0.3521 - ac
c: 0.8491 - val_loss: 0.3089 - val_acc: 0.8688
Accuracy:  0.8692692949788109
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 387s 12ms/step - loss: 0.5392 - ac
c: 0.7301 - val_loss: 0.4110 - val_acc: 0.8267
Epoch 2/3
31145/31145 [==============================] - 386s 12ms/step - loss: 0.4258 - ac
c: 0.8096 - val_loss: 0.3677 - val_acc: 0.8411
Epoch 3/3
31145/31145 [==============================] - 386s 12ms/step - loss: 0.3599 - ac
c: 0.8454 - val_loss: 0.3435 - val_acc: 0.8492
Accuracy:  0.8491074868370361
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 381s 12ms/step - loss: 0.5423 - ac
c: 0.7266 - val_loss: 0.4887 - val_acc: 0.7701
Epoch 2/3
31145/31145 [==============================] - 381s 12ms/step - loss: 0.4114 - ac
c: 0.8157 - val_loss: 0.3287 - val_acc: 0.8607
Epoch 3/3
31145/31145 [==============================] - 377s 12ms/step - loss: 0.3536 - ac
c: 0.8480 - val_loss: 0.3113 - val_acc: 0.8666
Accuracy:  0.8684987800179786
[0.8686271991781174, 0.869654552459227, 0.8692692949788109, 0.8491074868370361, 0.
8684987800179786]
```

In [71]:

```
num_folds = 5

acc_scores2 =[]
skf1 = StratifiedKFold(n_splits=num_folds, shuffle=True)
for i, (train, test) in enumerate(skf1.split(X, y)):

    rnn1 = rnn1_create()
    #doing modeling same as above, without for loop

    rnn1.fit(X_train, y_train_ohe, validation_data=(X_test, y_test_ohe), epochs=3, batch_size=12
8)
    #this is just what i do without cross validation
    yhat = np.argmax(rnn1.predict(X_test), axis=1)

    acc_score2 = mt.accuracy_score(y_test,yhat)
    acc_scores2.append(acc_score2)
    print("Accuracy: ", acc_score2)
print(acc_scores2)
```

```
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 194s 6ms/step - loss: 0.5530 - acc:
0.7230 - val_loss: 0.5312 - val_acc: 0.7892
Epoch 2/3
31145/31145 [==============================] - 192s 6ms/step - loss: 0.4604 - acc:
0.7885 - val_loss: 0.4241 - val_acc: 0.8166
Epoch 3/3
31145/31145 [==============================] - 194s 6ms/step - loss: 0.3943 - acc:
0.8282 - val_loss: 0.3279 - val_acc: 0.8653
Accuracy:  0.865031462694234
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 191s 6ms/step - loss: 0.5564 - acc:
0.7127 - val_loss: 0.4972 - val_acc: 0.7791
Epoch 2/3
31145/31145 [==============================] - 189s 6ms/step - loss: 0.4671 - acc:
0.7848 - val_loss: 0.4353 - val_acc: 0.8350
Epoch 3/3
31145/31145 [==============================] - 189s 6ms/step - loss: 0.3960 - acc:
0.8286 - val_loss: 0.3314 - val_acc: 0.8619
Accuracy:  0.8622062411711827
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 191s 6ms/step - loss: 0.5581 - acc:
0.7166 - val_loss: 0.4435 - val_acc: 0.8079
Epoch 2/3
31145/31145 [==============================] - 190s 6ms/step - loss: 0.4603 - acc:
0.7885 - val_loss: 0.3683 - val_acc: 0.8474
Epoch 3/3
31145/31145 [==============================] - 191s 6ms/step - loss: 0.3911 - acc:
0.8284 - val_loss: 0.3585 - val_acc: 0.8508
Accuracy:  0.8506485167587003
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 191s 6ms/step - loss: 0.5458 - acc:
0.7326 - val_loss: 0.3995 - val_acc: 0.8332
Epoch 2/3
31145/31145 [==============================] - 191s 6ms/step - loss: 0.4255 - acc:
0.8099 - val_loss: 0.4669 - val_acc: 0.7917
Epoch 3/3
31145/31145 [==============================] - 190s 6ms/step - loss: 0.3715 - acc:
0.8388 - val_loss: 0.3231 - val_acc: 0.8625
Accuracy:  0.8632335944522923
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 192s 6ms/step - loss: 0.5573 - acc:
0.7195 - val_loss: 0.4251 - val_acc: 0.8101
Epoch 2/3
31145/31145 [==============================] - 192s 6ms/step - loss: 0.4499 - acc:
0.7950 - val_loss: 0.3555 - val_acc: 0.8509
Epoch 3/3
31145/31145 [==============================] - 191s 6ms/step - loss: 0.3797 - acc:
0.8344 - val_loss: 0.3288 - val_acc: 0.8599
Accuracy:  0.8600231154488249
[0.865031462694234, 0.8622062411711827, 0.8506485167587003, 0.8632335944522923, 0.
8600231154488249]
```

In [ ]:

In [29]:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM

def rnn2_create():
    rnn2 = Sequential()
    rnn2.add(embedding_layer)
    rnn2.add(LSTM(100,dropout=0.2, recurrent_dropout=0.2))
    rnn2.add(Dense(NUM_CLASSES, activation='softmax'))
    rnn2.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
    return rnn2

rnn2 = rnn1_create()
print(rnn2.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 1000, 100)         4904900
_________________________________________________________________
lstm_2 (LSTM)                (None, 100)               80400
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 202
=================================================================
Total params: 4,985,502
Trainable params: 80,602
Non-trainable params: 4,904,900
_________________________________________________________________
None
```

In [ ]:

```
history2 = rnn2.fit(X_train, y_train_ohe, validation_data=(X_test, y_test_ohe), epochs=3, batch_
size=64)
```

In [74]:

```
num_folds = 5

acc_scores3 =[]
skf2 = StratifiedKFold(n_splits=num_folds, shuffle=True)
for i, (train, test) in enumerate(skf1.split(X, y)):

    rnn3 = rnn1_create()
    #doing modeling same as above, without for loop

    rnn3.fit(X_train, y_train_ohe, validation_data=(X_test, y_test_ohe), epochs=3, batch_size=12
8)
    #this is just what i do without cross validation
    yhat = np.argmax(rnn3.predict(X_test), axis=1)

    acc_score3 = mt.accuracy_score(y_test,yhat)
    acc_scores3.append(acc_score3)
    print("Accuracy: ", acc_score3)
print(acc_scores3)
```

```
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 193s 6ms/step - loss: 0.5571 - acc:
0.7150 - val_loss: 0.4289 - val_acc: 0.8126
Epoch 2/3
31145/31145 [==============================] - 192s 6ms/step - loss: 0.4491 - acc:
0.7937 - val_loss: 0.3716 - val_acc: 0.8428
Epoch 3/3
31145/31145 [==============================] - 190s 6ms/step - loss: 0.3800 - acc:
0.8344 - val_loss: 0.3237 - val_acc: 0.8641
Accuracy:  0.8642609477334018
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 193s 6ms/step - loss: 0.5489 - acc:
0.7273 - val_loss: 0.4168 - val_acc: 0.8252
Epoch 2/3
31145/31145 [==============================] - 191s 6ms/step - loss: 0.4401 - acc:
0.8041 - val_loss: 0.3466 - val_acc: 0.8529
Epoch 3/3
31145/31145 [==============================] - 192s 6ms/step - loss: 0.3815 - acc:
0.8355 - val_loss: 0.3281 - val_acc: 0.8628
Accuracy:  0.8633620136124309
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 192s 6ms/step - loss: 0.5571 - acc:
0.7202 - val_loss: 0.4777 - val_acc: 0.7627
Epoch 2/3
31145/31145 [==============================] - 191s 6ms/step - loss: 0.4608 - acc:
0.7906 - val_loss: 0.4178 - val_acc: 0.8179
Epoch 3/3
31145/31145 [==============================] - 191s 6ms/step - loss: 0.3912 - acc:
0.8282 - val_loss: 0.3382 - val_acc: 0.8537
Accuracy:  0.8528316424810581
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 196s 6ms/step - loss: 0.5657 - acc:
0.7119 - val_loss: 0.7003 - val_acc: 0.7178
Epoch 2/3
31145/31145 [==============================] - 192s 6ms/step - loss: 0.4846 - acc:
0.7762 - val_loss: 0.3841 - val_acc: 0.8354
Epoch 3/3
31145/31145 [==============================] - 193s 6ms/step - loss: 0.4076 - acc:
0.8184 - val_loss: 0.3733 - val_acc: 0.8299
Accuracy:  0.8297161936560935
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 193s 6ms/step - loss: 0.5578 - acc:
0.7168 - val_loss: 0.4446 - val_acc: 0.8084
Epoch 2/3
31145/31145 [==============================] - 192s 6ms/step - loss: 0.4760 - acc:
0.7799 - val_loss: 0.4140 - val_acc: 0.8279
Epoch 3/3
31145/31145 [==============================] - 190s 6ms/step - loss: 0.4089 - acc:
0.8195 - val_loss: 0.3385 - val_acc: 0.8569
Accuracy:  0.8573263130859125
[0.8642609477334018, 0.8633620136124309, 0.8528316424810581, 0.8297161936560935,
0.8573263130859125]
```

In [84]:

```
t = 2.26 / np.sqrt(10)
e = (1-np.array(acc_scores1))-(1-np.array(acc_scores2))
stdtot =np.std(e)

dbar = np.mean(e)
print('model1 64 vs model1 128 acc range :',dbar-t*stdtot,dbar+t*stdtot)
```

```
model1 64 vs model1 128 acc range : -0.012441447456650008 0.002835694278275837
```

Becasue the range is include 0, so we can not say that with 95% confident level, model1 64 and model1 128 are statisitcally different base on accuracy.

In [87]:

```
from statistics import mean
print('Average accuracy for model1 64 ',mean(acc_scores1))
print('Average accuracy for model1 128 ',mean(acc_scores2))
```

```
Average accuracy for model1 64  0.8650314626942339
Average accuracy for model1 128  0.8602285861050468
```

Base on average accuracy, model 64 batch size is litte bit better.

In [85]:

```
t = 2.26 / np.sqrt(10)
e = (1-np.array(acc_scores1))-(1-np.array(acc_scores3))
stdtot =np.std(e)

dbar = np.mean(e)
print('model1 64 vs model1 softmax acc range :',dbar-t*stdtot,dbar+t*stdtot)
```

```
model1 64 vs model1 softmax acc range : -0.015634585971797028 -0.007429495189112228
8
```

Becasue the range is not include 0, so we can say that with 95% confident level, model1 64 and model1 softmax are statisitcally different base on accuracy.

In [89]:

```
print('Average accuracy for model1 64 ',mean(acc_scores1))
print('Average accuracy for model1 softmax ',mean(acc_scores3))
```

```
Average accuracy for model1 64  0.8650314626942339
Average accuracy for model1 softmax  0.8534994221137794
```

Base on average accuracy, model 64 batch size is better.

For model1, model1 with sigmoid and 64 batch size is better.

Model2 GRU

In [30]:

```
from keras.models import Sequential, Input, Model
from keras.layers import Dense
from keras.layers import LSTM, GRU, SimpleRNN
from keras.layers.embeddings import Embedding

def rnn3_create():
    rnn_gru3 = Sequential()
    rnn_gru3.add(embedding_layer)
    rnn_gru3.add(GRU(100,dropout=0.2, recurrent_dropout=0.2))
    rnn_gru3.add(Dense(NUM_CLASSES, activation='sigmoid'))
    rnn_gru3.compile(loss='binary_crossentropy',
                  optimizer='rmsprop',
                  metrics=['accuracy'])
    return rnn_gru3

rnn_gru3 = rnn3_create()

print(rnn_gru3.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 1000, 100)         4904900
_________________________________________________________________
gru_1 (GRU)                  (None, 100)               60300
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 202
=================================================================
Total params: 4,965,402
Trainable params: 60,502
Non-trainable params: 4,904,900
_________________________________________________________________
None
```

In [21]:

```
history4 = rnn_gru3.fit(X_train, y_train_ohe, validation_data=(X_test, y_test_ohe), epochs=3, ba
tch_size=64)
```

```
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 160s 5ms/step - loss: 0.4725 - acc:
0.7773 - val_loss: 0.3325 - val_acc: 0.8618
Epoch 2/3
31145/31145 [==============================] - 160s 5ms/step - loss: 0.3380 - acc:
0.8567 - val_loss: 0.2999 - val_acc: 0.8757
Epoch 3/3
31145/31145 [==============================] - 160s 5ms/step - loss: 0.3106 - acc:
0.8696 - val_loss: 0.2968 - val_acc: 0.8774
```

In [76]:

```
num_folds = 5

acc_scores4 =[]
skf3 = StratifiedKFold(n_splits=num_folds, shuffle=True)
for i, (train, test) in enumerate(skf1.split(X, y)):

    rnn_gru3 = rnn3_create()
    #doing modeling same as above, without for loop

    rnn_gru3.fit(X_train, y_train_ohe, validation_data=(X_test, y_test_ohe), epochs=3, batch_siz
e=64)
    #this is just what i do without cross validation
    yhat = np.argmax(rnn_gru3.predict(X_test), axis=1)

    acc_score4 = mt.accuracy_score(y_test,yhat)
    acc_scores4.append(acc_score4)
    print("Accuracy: ", acc_score4)
print(acc_scores4)
```

```
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 294s 9ms/step - loss: 0.4758 - acc:
0.7733 - val_loss: 0.3374 - val_acc: 0.8604
Epoch 2/3
31145/31145 [==============================] - 292s 9ms/step - loss: 0.3403 - acc:
0.8566 - val_loss: 0.2996 - val_acc: 0.8743
Epoch 3/3
31145/31145 [==============================] - 291s 9ms/step - loss: 0.3119 - acc:
0.8680 - val_loss: 0.2975 - val_acc: 0.8813
Accuracy:  0.8814691151919867
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 294s 9ms/step - loss: 0.4899 - acc:
0.7641 - val_loss: 0.3361 - val_acc: 0.8607
Epoch 2/3
31145/31145 [==============================] - 293s 9ms/step - loss: 0.3440 - acc:
0.8540 - val_loss: 0.2954 - val_acc: 0.8758
Epoch 3/3
31145/31145 [==============================] - 290s 9ms/step - loss: 0.3116 - acc:
0.8678 - val_loss: 0.2788 - val_acc: 0.8831
Accuracy:  0.8831385642737897
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 294s 9ms/step - loss: 0.4806 - acc:
0.7737 - val_loss: 0.3578 - val_acc: 0.8490
Epoch 2/3
31145/31145 [==============================] - 292s 9ms/step - loss: 0.3441 - acc:
0.8557 - val_loss: 0.2893 - val_acc: 0.8811
Epoch 3/3
31145/31145 [==============================] - 290s 9ms/step - loss: 0.3097 - acc:
0.8709 - val_loss: 0.2836 - val_acc: 0.8813
Accuracy:  0.8815975343521253
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 296s 10ms/step - loss: 0.4895 - ac
c: 0.7638 - val_loss: 0.3718 - val_acc: 0.8423
Epoch 2/3
31145/31145 [==============================] - 293s 9ms/step - loss: 0.3477 - acc:
0.8528 - val_loss: 0.2977 - val_acc: 0.8776
Epoch 3/3
31145/31145 [==============================] - 290s 9ms/step - loss: 0.3129 - acc:
0.8686 - val_loss: 0.2871 - val_acc: 0.8838
Accuracy:  0.8846795941954539
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 296s 9ms/step - loss: 0.4784 - acc:
0.7721 - val_loss: 0.3461 - val_acc: 0.8525
Epoch 2/3
31145/31145 [==============================] - 293s 9ms/step - loss: 0.3519 - acc:
0.8515 - val_loss: 0.3468 - val_acc: 0.8474
Epoch 3/3
31145/31145 [==============================] - 290s 9ms/step - loss: 0.3162 - acc:
0.8661 - val_loss: 0.2963 - val_acc: 0.8744
Accuracy:  0.8741492230640812
[0.8814691151919867, 0.8831385642737897, 0.8815975343521253, 0.8846795941954539,
0.8741492230640812]
```

In [ ]:

```

```

In [77]:

```
num_folds = 5

acc_scores5 =[]
skf3 = StratifiedKFold(n_splits=num_folds, shuffle=True)
for i, (train, test) in enumerate(skf1.split(X, y)):

    rnn_gru3 = rnn3_create()
    #doing modeling same as above, without for loop

    rnn_gru3.fit(X_train, y_train_ohe, validation_data=(X_test, y_test_ohe), epochs=3, batch_siz
e=128)
    #this is just what i do without cross validation
    yhat = np.argmax(rnn_gru3.predict(X_test), axis=1)

    acc_score5 = mt.accuracy_score(y_test,yhat)
    acc_scores5.append(acc_score5)
    print("Accuracy: ", acc_score5)
print(acc_scores5)
```

```
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 150s 5ms/step - loss: 0.5148 - acc:
0.7477 - val_loss: 0.3669 - val_acc: 0.8480
Epoch 2/3
31145/31145 [==============================] - 147s 5ms/step - loss: 0.3720 - acc:
0.8405 - val_loss: 0.3206 - val_acc: 0.8638
Epoch 3/3
31145/31145 [==============================] - 147s 5ms/step - loss: 0.3305 - acc:
0.8599 - val_loss: 0.2920 - val_acc: 0.8777
Accuracy:  0.8782586361885193
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 150s 5ms/step - loss: 0.5257 - acc:
0.7414 - val_loss: 0.4399 - val_acc: 0.8166
Epoch 2/3
31145/31145 [==============================] - 147s 5ms/step - loss: 0.3805 - acc:
0.8352 - val_loss: 0.3168 - val_acc: 0.8670
Epoch 3/3
31145/31145 [==============================] - 147s 5ms/step - loss: 0.3344 - acc:
0.8564 - val_loss: 0.3006 - val_acc: 0.8740
Accuracy:  0.8745344805444972
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 150s 5ms/step - loss: 0.5218 - acc:
0.7424 - val_loss: 0.3884 - val_acc: 0.8323
Epoch 2/3
31145/31145 [==============================] - 147s 5ms/step - loss: 0.3735 - acc:
0.8389 - val_loss: 0.3259 - val_acc: 0.8692
Epoch 3/3
31145/31145 [==============================] - 148s 5ms/step - loss: 0.3292 - acc:
0.8608 - val_loss: 0.3101 - val_acc: 0.8743
Accuracy:  0.8756902529857454
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 150s 5ms/step - loss: 0.5330 - acc:
0.7368 - val_loss: 0.5049 - val_acc: 0.7793
Epoch 2/3
31145/31145 [==============================] - 147s 5ms/step - loss: 0.3787 - acc:
0.8378 - val_loss: 0.3795 - val_acc: 0.8456
Epoch 3/3
31145/31145 [==============================] - 147s 5ms/step - loss: 0.3313 - acc:
0.8578 - val_loss: 0.2952 - val_acc: 0.8779
Accuracy:  0.8777449595479646
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 151s 5ms/step - loss: 0.5209 - acc:
0.7453 - val_loss: 0.3582 - val_acc: 0.8474
Epoch 2/3
31145/31145 [==============================] - 148s 5ms/step - loss: 0.3716 - acc:
0.8410 - val_loss: 0.3146 - val_acc: 0.8689
Epoch 3/3
31145/31145 [==============================] - 146s 5ms/step - loss: 0.3303 - acc:
0.8593 - val_loss: 0.3013 - val_acc: 0.8722
Accuracy:  0.8727366123025555
[0.8782586361885193, 0.8745344805444972, 0.8756902529857454, 0.8777449595479646,
0.8727366123025555]
```

In [31]:

```
def rnn4_create():
    rnn_gru4 = Sequential()
    rnn_gru4.add(embedding_layer)
    rnn_gru4.add(GRU(100,dropout=0.2, recurrent_dropout=0.2))
    rnn_gru4.add(Dense(NUM_CLASSES, activation='softmax'))
    rnn_gru4.compile(loss='binary_crossentropy',
                  optimizer='rmsprop',
                  metrics=['accuracy'])
    return rnn_gru4

rnn_gru4 = rnn4_create()

print(rnn_gru4.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 1000, 100)         4904900   
_________________________________________________________________
gru_2 (GRU)                  (None, 100)               60300     
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 202       
=================================================================
Total params: 4,965,402
Trainable params: 60,502
Non-trainable params: 4,904,900
_________________________________________________________________
None
```

In [79]:

```
num_folds = 5

acc_scores6 =[]
skf3 = StratifiedKFold(n_splits=num_folds, shuffle=True)
for i, (train, test) in enumerate(skf1.split(X, y)):

    rnn_gru4 = rnn4_create()
    #doing modeling same as above, without for loop

    rnn_gru4.fit(X_train, y_train_ohe, validation_data=(X_test, y_test_ohe), epochs=3, batch_siz
e=64)
    #this is just what i do without cross validation
    yhat = np.argmax(rnn_gru4.predict(X_test), axis=1)

    acc_score6 = mt.accuracy_score(y_test,yhat)
    acc_scores6.append(acc_score6)
    print("Accuracy: ", acc_score6)
print(acc_scores6)
```

```
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 297s 10ms/step - loss: 0.4922 - ac
c: 0.7644 - val_loss: 0.3335 - val_acc: 0.8627
Epoch 2/3
31145/31145 [==============================] - 293s 9ms/step - loss: 0.3456 - acc:
0.8522 - val_loss: 0.2950 - val_acc: 0.8790
Epoch 3/3
31145/31145 [==============================] - 292s 9ms/step - loss: 0.3128 - acc:
0.8700 - val_loss: 0.2815 - val_acc: 0.8844
Accuracy:  0.8844227558751766
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 298s 10ms/step - loss: 0.4881 - ac
c: 0.7655 - val_loss: 0.3263 - val_acc: 0.8625
Epoch 2/3
31145/31145 [==============================] - 296s 9ms/step - loss: 0.3429 - acc:
0.8546 - val_loss: 0.2918 - val_acc: 0.8781
Epoch 3/3
31145/31145 [==============================] - 293s 9ms/step - loss: 0.3100 - acc:
0.8687 - val_loss: 0.2833 - val_acc: 0.8842
Accuracy:  0.8841659175548992
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 299s 10ms/step - loss: 0.4885 - ac
c: 0.7674 - val_loss: 0.3487 - val_acc: 0.8533
Epoch 2/3
31145/31145 [==============================] - 296s 9ms/step - loss: 0.3461 - acc:
0.8507 - val_loss: 0.2989 - val_acc: 0.8747
Epoch 3/3
31145/31145 [==============================] - 292s 9ms/step - loss: 0.3126 - acc:
0.8682 - val_loss: 0.2870 - val_acc: 0.8804
Accuracy:  0.8804417619108771
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 298s 10ms/step - loss: 0.4967 - ac
c: 0.7591 - val_loss: 0.4437 - val_acc: 0.7929
Epoch 2/3
31145/31145 [==============================] - 295s 9ms/step - loss: 0.3505 - acc:
0.8502 - val_loss: 0.3073 - val_acc: 0.8731
Epoch 3/3
31145/31145 [==============================] - 291s 9ms/step - loss: 0.3133 - acc:
0.8671 - val_loss: 0.3488 - val_acc: 0.8551
Accuracy:  0.8551431873635547
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 300s 10ms/step - loss: 0.4936 - ac
c: 0.7621 - val_loss: 0.3354 - val_acc: 0.8613
Epoch 2/3
31145/31145 [==============================] - 295s 9ms/step - loss: 0.3483 - acc:
0.8516 - val_loss: 0.3052 - val_acc: 0.8715
Epoch 3/3
31145/31145 [==============================] - 293s 9ms/step - loss: 0.3144 - acc:
0.8669 - val_loss: 0.3041 - val_acc: 0.8716
Accuracy:  0.8715808398613073
[0.8844227558751766, 0.8841659175548992, 0.8804417619108771, 0.8551431873635547,
0.8715808398613073]
```

In [90]:

```
t = 2.26 / np.sqrt(10)
e = (1-np.array(acc_scores4))-(1-np.array(acc_scores5))
stdtot =np.std(e)

dbar = np.mean(e)
print('model2 64 vs model2 128 acc range :',dbar-t*stdtot,dbar+t*stdtot)
```

```
model2 64 vs model2 128 acc range : -0.007060904348436647 -0.0033667314548252217
```

Becasue the range is not include 0, so we can say that with 95% confident level, model2 64 and model2 128 are statisitcally different base on accuracy.

In [91]:

```
print('Average accuracy for model2 64 ',mean(acc_scores4))
print('Average accuracy for model2 128 ',mean(acc_scores5))
```

```
Average accuracy for model2 64  0.8810068062154872
Average accuracy for model2 128  0.8757929883138564
```

Base on average accuracy, model2 64 is better.

In [92]:

```
t = 2.26 / np.sqrt(10)
e = (1-np.array(acc_scores4))-(1-np.array(acc_scores6))
stdtot =np.std(e)

dbar = np.mean(e)
print('model2 64 vs model2 softmax acc range :',dbar-t*stdtot,dbar+t*stdtot)
```

```
model2 64 vs model2 softmax acc range : -0.007060904348436647 -0.00336673145482522
17
```

Becasue the range is not include 0, so we can say that with 95% confident level, model2 64 and model2 softmax are statisitcally different base on accuracy.

In [94]:

```
print('Average accuracy for model2 64 ',mean(acc_scores4))
print('Average accuracy for model2 softmax ',mean(acc_scores6))
```

```
Average accuracy for model2 64  0.8810068062154872
Average accuracy for model2 softmax  0.8751508925131629
```

Base on average accuracy, model2 64 is better.

For model2, model2 with sigmoid and 64 batch size is better.

In [95]:

```
t = 2.26 / np.sqrt(10)
e = (1-np.array(acc_scores1))-(1-np.array(acc_scores4))
stdtot =np.std(e)

dbar = np.mean(e)
print('model1 64 vs model2 64 acc range :',dbar-t*stdtot,dbar+t*stdtot)
```

```
model1 64 vs model2 64 acc range : 0.008687320463473508 0.023263366579O3321
```

Becasue the range is not include 0, so we can say that with 95% confident level, model1 64 and model2 64 are statisitcally different base on accuracy.

In [97]:

```
print('Average accuracy for model1 64 ',mean(acc_scores1))
print('Average accuracy for model2 64 ',mean(acc_scores4))
```

```
Average accuracy for model1 64  0.8650314626942339
Average accuracy for model2 64  0.8810068062154872
```

Base on average accuracy, model2 64 is better.

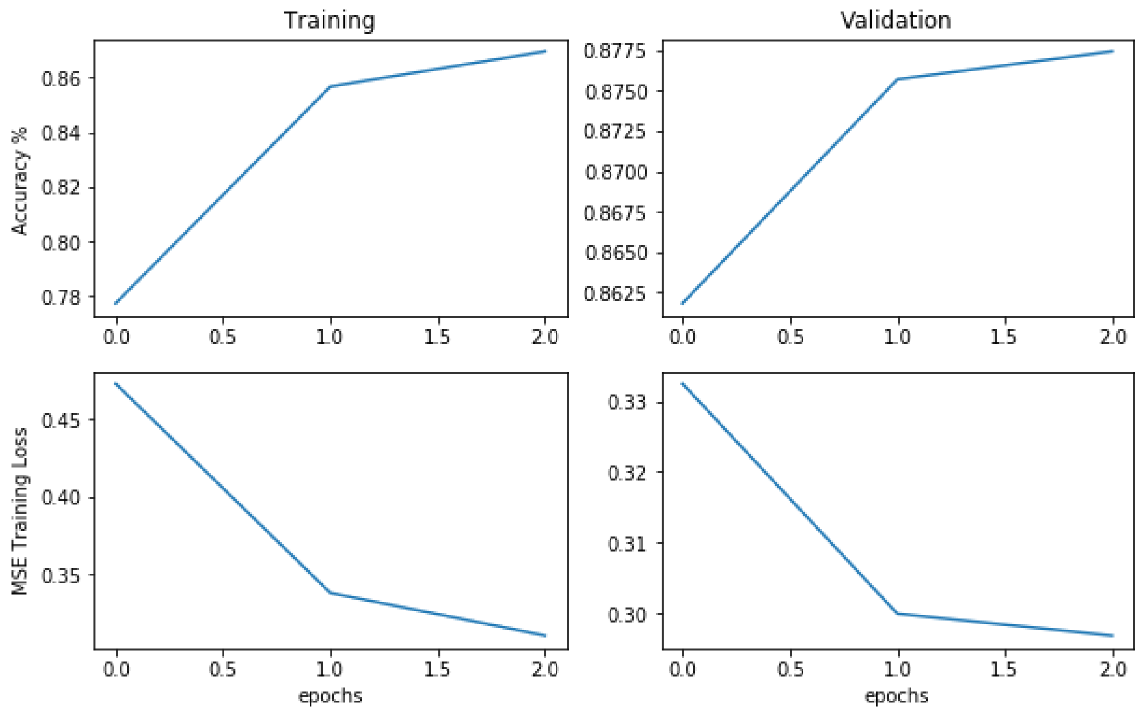
Model2 with 64 batch size and sigmoid is better.

In [22]:

```
from matplotlib import pyplot as plt

%matplotlib inline

plt.figure(figsize=(10,6))
plt.subplot(2,2,1)
plt.plot(history4.history['acc'])

plt.ylabel('Accuracy %')
plt.title('Training')
plt.subplot(2,2,2)
plt.plot(history4.history['val_acc'])
plt.title('Validation')

plt.subplot(2,2,3)
plt.plot(history4.history['loss'])
plt.ylabel('MSE Training Loss')
plt.xlabel('epochs')

plt.subplot(2,2,4)
plt.plot(history4.history['val_loss'])
plt.xlabel('epochs')
```

Out[22]:

Text(0.5, 0, 'epochs')



For train vs loss, I think it converge, the line reach the bottom. For validation, the line is also converge. Loss decreasing and accuracy increasing.

Second recurrent chain

In [26]:

```
from keras.models import Sequential, Input, Model
from keras.layers import Dense
from keras.layers import LSTM, GRU, SimpleRNN
from keras.layers.embeddings import Embedding

def rnn7_create():
    rnn_gru7 = Sequential()
    rnn_gru7.add(embedding_layer)

    rnn_gru7.add(GRU(100,dropout=0.2, recurrent_dropout=0.2,return_sequences=True))#add a secon
d recurrent chain to your RNN

    rnn_gru7.add(GRU(100,dropout=0.2, recurrent_dropout=0.2))
    rnn_gru7.add(Dense(NUM_CLASSES, activation='sigmoid'))
    rnn_gru7.compile(loss='binary_crossentropy',
                  optimizer='rmsprop',
                  metrics=['accuracy'])
    return rnn_gru7

rnn_gru7 = rnn7_create()

print(rnn_gru7.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 500, 100)          4904900   
_________________________________________________________________
gru_16 (GRU)                 (None, 500, 100)          60300     
_________________________________________________________________
gru_17 (GRU)                 (None, 100)               60300     
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 202       
=================================================================
Total params: 5,025,702
Trainable params: 120,802
Non-trainable params: 4,904,900
_________________________________________________________________
None
```

In [30]:

```
history7 = rnn_gru7.fit(X_train, y_train_ohe, validation_data=(X_test, y_test_ohe), epochs=3, ba
tch_size=64)
```

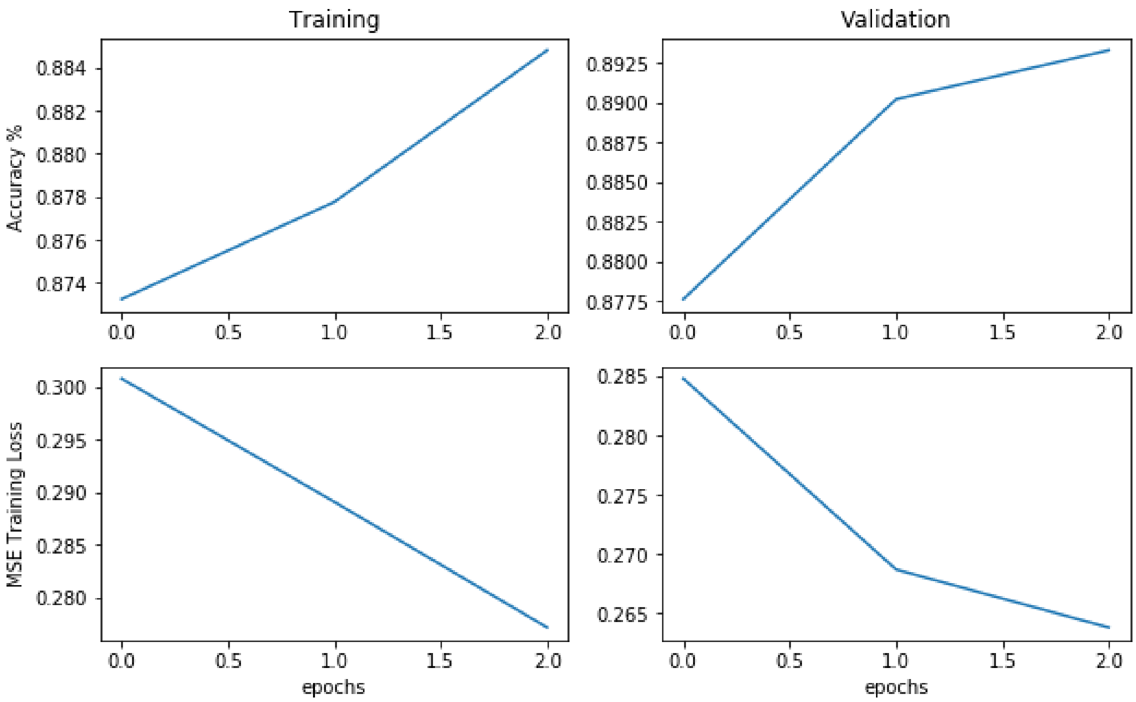
```
Train on 31145 samples, validate on 7787 samples
Epoch 1/3
31145/31145 [==============================] - 547s 18ms/step - loss: 0.3008 - ac
c: 0.8732 - val_loss: 0.2848 - val_acc: 0.8776
Epoch 2/3
31145/31145 [==============================] - 560s 18ms/step - loss: 0.2891 - ac
c: 0.8777 - val_loss: 0.2687 - val_acc: 0.8902
Epoch 3/3
31145/31145 [==============================] - 557s 18ms/step - loss: 0.2771 - ac
c: 0.8848 - val_loss: 0.2638 - val_acc: 0.8933
```

In [31]:

```
from matplotlib import pyplot as plt

%matplotlib inline

plt.figure(figsize=(10,6))
plt.subplot(2,2,1)
plt.plot(history7.history['acc'])

plt.ylabel('Accuracy %')
plt.title('Training')
plt.subplot(2,2,2)
plt.plot(history7.history['val_acc'])
plt.title('Validation')

plt.subplot(2,2,3)
plt.plot(history7.history['loss'])
plt.ylabel('MSE Training Loss')
plt.xlabel('epochs')

plt.subplot(2,2,4)
plt.plot(history7.history['val_loss'])
plt.xlabel('epochs')
```

Out[31]:

```
Text(0.5, 0, 'epochs')
```



For train vs loss, I think it converge, the line reach the bottom. For validation, the line is also converge. Loss decreasing and accuracy increasing.

Exceptional Work

I will paticipate the Seminar Research Study on 17th of may 1pm.

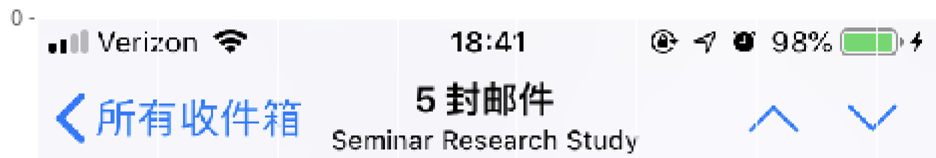
In [33]:

```
from PIL import Image

plt.figure(figsize=(100,50))

img = Image.open('d10b59955278539f3dc78a818e0307a.png')
plt.subplot(3,1,1)
plt.imshow(img)
img1 = Image.open('33b07f786e91f2048d04da7034cc0ad.png')
plt.subplot(3,1,2)
plt.imshow(img1)
plt.subplot(3,1,3)
img2 = Image.open('f1bc574092a564dd5937407fc512ac5.png')
plt.imshow(img2)
```

Out[33]:

```
<matplotlib.image.AxesImage at 0x18cdeffca90>
```


Verizon
18:40
98%
所有收件箱
5 封邮件
Seminar Research Study
Yang Shen
昨天
收件人： Elena Sharp
详细资料
YS
Hi
I am a student from Dr. Larson Machine learning class. I want to participate in the research.
Best wishes,
Yang Shen
查看更多
eesharp@mail.smu.edu
昨天
收件人： Yang Shen
详细资料
ES
Hello,
It's awesome that you're interested! To tell you a little bit more, basically you are going to be asked to complete several everyday tasks while we collect information using everyday sensors that can be found on things such as an iPhone or a laptop. It will
0
200
400
600
800
1000
1200
0
100
200
300
400
500
600
700


0
Verizon
18:41
98%
所有收件箱
5 封邮件
Seminar Research Study


Yang Shen
昨天
收件人： Elena Sharp
详细资料
YS
I have time on next friday.
Best wishes,
Yang Shen
查看更多来自“Elena Sharp”的内容
Siri 找到了 1 个日程
5月17日 周五 13:00 查看...
eesharp@mail.smu.edu
10:05
收件人： Yang Shen
详细资料
ES
Hello,
Would next Friday, May 17th, at 1 work for you?
Thanks,
Elena Sharp
查看更多来自“Yang Shen”的内容
200
400
600
800
1000
1200
0
100
200
300
400
500
600
700


Verizon
18:41
98%
所有收件箱
5 封邮件
Seminar Research Study
eesharp@mail.smu.edu
10:05
收件人： Yang Shen
详细资料
ES
0
200


Hello,
Would next Friday, May 17th, at 1 work for you?
Thanks,
Elena Sharp
查看更多来自“Yang Shen”的内容
在smu“已发送”邮箱中找到
Yang Shen
16:10
收件人： Elena Sharp
详细资料
YS
Yes!
Best wishes,
Yang Shen
查看更多来自“Elena Sharp”的内容
400
600
800
1000
1200
0
100
200
300
400
500
600
700

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
```

In [ ]:

```
```