amazon-reviews-sensitivity-study

March 17, 2021

1 Amazon Reviews Sensitivity Study

Georg Zhelev

1.0.1 Notebook Metaparameters

[1]:
```
voc_size = 2000             #vocabulary size (50,4000, 6000, 12000)
epo=4                       #epochs
train_size = 50000         #size of train sample
test_size = 10000          #size of test sample
pre_process = "yes"         #"yes" or "no"

threshold = 300000           #threshold above which not to one-hot-encode
batches = 512
```

[2]:
```
#load data
import os # accessing directory structure on kaggle
import bz2 #unzip and load

#view/work with data
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import numpy as np

#split
from sklearn.model_selection import train_test_split

#pre-process
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer

#create dictionary
from tensorflow.keras.preprocessing.text import Tokenizer, text_to_word_sequence
from tensorflow.keras.preprocessing.sequence import pad_sequences

#fit DL
import tensorflow
```

```
from tensorflow.python.keras import models, layers, optimizers
from keras.models import Sequential
from keras import layers
from keras.layers import Dropout


#fit ML
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix

#evaluate
from sklearn.metrics import accuracy_score

#to reset the trained model
from keras.backend import clear_session

import time
tic = time.time()
```

```
Using TensorFlow backend.
```

1.0.2 Show file names in directory

[3]:
```
print(os.listdir('../input'))
```

```
['test.ft.txt.bz2', 'train.ft.txt.bz2']
```

1.0.3 1. Data Description

- Amazon customer reviews (input) and star ratings (output)
- industrial level dataset (3,6 mil. train)

The data format is the following: - label, Text (all in one line) - label 1 corresponds to 1- and 2-star reviews - label 2 corresponds to 4- and 5-star reviews - Most of the reviews are in English

1.0.4 1.1 Load and decompress Files

Decompress files and return a list containing each line as a list item.

[4]:
```
#%%cache

train_file = bz2.BZ2File('../input/train.ft.txt.bz2')
test_file = bz2.BZ2File('../input/test.ft.txt.bz2')

train_file_lines = train_file.readlines(int(448.374641923*train_size))
test_file_lines = test_file.readlines(int(452.488687783*test_size))
```

[5]:
```
print(int(448.374641923*500000))  #bytes to read
print(int(452.488687783*100000)) #bytes to read
```

```
224187320
45248868
```

The readlines() method returns a list containing each line in the file as a list item.

[6]:
```
print(type(train_file_lines[0]))
train_file_lines[0]
```

```
<class 'bytes'>
```

[6]:
```
b'__label__2 Stuning even for the non-gamer: This sound track was beautiful! It
paints the senery in your mind so well I would recomend it even to people who
hate vid. game music! I have played the game Chrono Cross but out of all of the
games I have ever played it has the best music! It backs away from crude
keyboarding and takes a fresher step with grate guitars and soulful orchestras.
It would impress anyone who cares to listen! ^_^\n'
```

1.0.5 1.2 Decode from raw binary strings to strings that can be parsed. Extract labels and extract texts.

[7]:
```
dec=train_file_lines[0].decode("utf-8")
dec
```

[7]:
```
'__label__2 Stuning even for the non-gamer: This sound track was beautiful! It
paints the senery in your mind so well I would recomend it even to people who
hate vid. game music! I have played the game Chrono Cross but out of all of the
games I have ever played it has the best music! It backs away from crude
keyboarding and takes a fresher step with grate guitars and soulful orchestras.
It would impress anyone who cares to listen! ^_^\n'
```

1.0.6 Extract labels and extract texts.

[8]:
```
print(dec[0:10])
dec[10:]
```

```
__label__2
```

[8]:
```
' Stuning even for the non-gamer: This sound track was beautiful! It paints the
senery in your mind so well I would recomend it even to people who hate vid.
game music! I have played the game Chrono Cross but out of all of the games I
have ever played it has the best music! It backs away from crude keyboarding and
takes a fresher step with grate guitars and soulful orchestras. It would impress
anyone who cares to listen! ^_^\n'
```

[9]:
```
def get_labels_and_texts(file):
    labels = []
    texts = []
    for review in file:
```

```
        x = review.decode("utf-8")
        labels.append(int(x[9]) - 1)
        texts.append(x[10:].strip())
    return np.array(labels), texts
```

strip() remove spaces at the beginning and at the end of the string: 1 is subtracted from labeles 1 and 2 to result in labels 0 and 1

[10]:
```
train_labels_l, train_texts_l = get_labels_and_texts(train_file_lines)
test_labels, test_texts_l = get_labels_and_texts(test_file_lines)

del train_file, test_file
```

[11]:
```
print("Labels:", test_labels)
print("Labels Type:", type(test_labels))

print("Text example 1st: \n",  train_texts_l[0])
print("Text type \n",  type(train_texts_l))
```

```
Labels: [1 1 0 … 1 0 1]
Labels Type: <class 'numpy.ndarray'>
Text example 1st:
 Stuning even for the non-gamer: This sound track was beautiful! It paints the
senery in your mind so well I would recomend it even to people who hate vid.
game music! I have played the game Chrono Cross but out of all of the games I
have ever played it has the best music! It backs away from crude keyboarding and
takes a fresher step with grate guitars and soulful orchestras. It would impress
anyone who cares to listen! ^_^
Text type
 <class 'list'>
```

1.0.7 2. View Prepared Data

Text is saved into a list, while labels saved into an array.

[12]:
```
#print(train_labels_l.shape)
#print(type(train_labels_l))
#print(type(train_texts_l))

print("Length train:", len(train_texts_l))
print("Length test:", len(test_texts_l))
```

```
Length train: 49473
Length test: 10007
```

Text is saved into a list, while labels saved into an array. The sample is abou balanced.

[13]:
```
unique, counts = np.unique(train_labels_l, return_counts=True)
```

```
print(np.asarray((unique, counts)).T)
```

```
[[    0 24256]
 [    1 25217]]
```

About equal distribution of classes. (1 is positive, while 0 a negative review).

1.0.8 2.2 Show examples of the Data

Positive and negative reviews (y-variable)

[14]:
```
train_labels_l[:10]
```

[14]:
```
array([1, 1, 1, 1, 1, 1, 0, 1, 1, 1])
```

First two text (x-variable)

[15]:
```
train_texts_l[0:2]
```

[15]:
```
['Stuning even for the non-gamer: This sound track was beautiful! It paints the
senery in your mind so well I would recomend it even to people who hate vid.
game music! I have played the game Chrono Cross but out of all of the games I
have ever played it has the best music! It backs away from crude keyboarding and
takes a fresher step with grate guitars and soulful orchestras. It would impress
anyone who cares to listen! ^_^',
 "The best soundtrack ever to anything.: I'm reading a lot of reviews saying
that this is the best 'game soundtrack' and I figured that I'd write a review to
disagree a bit. This in my opinino is Yasunori Mitsuda's ultimate masterpiece.
The music is timeless and I'm been listening to it for years now and its beauty
simply refuses to fade.The price tag on this is pretty staggering I must say,
but if you are going to buy any cd for this much money, this is the only one
that I feel would be worth every penny."]
```

1.0.9 3. Pre-Process

- bild a small and efficient vocabulary
- Stopwords only blow up the vocabulary
- non-numerical values
- Using the regular expressions module
- Match characters and subsititute them with spaces

1. Lowercase text
2. Remove non-word characters:
   - numbers and punctuation
3. Removes non-english language characters

[16]:
```
stop_words = stopwords.words('english')
#print(stop_words)
```

[17]:
```
import string
table = str.maketrans('', '', string.punctuation)
#stripped = [w.translate(table) for w in words]
#print(stripped[:100])

#words = [word for word in tokens if word.isalpha()]
```

The levels of pre-processing can be adjusted by commenting in and out different lines of the script below.

[18]:
```
%%time

NON_ALPHANUM = re.compile(r'[\W]')
NON_ASCII = re.compile(r'[^a-z0-1\s]')
stop_words = stopwords.words('english')
porter = PorterStemmer()

def normalize_texts(texts):
    normalized_texts = []
    for text in texts:
        lower = text.lower()
        no_punctuation = NON_ALPHANUM.sub(r' ', lower)
        no_non_ascii = NON_ASCII.sub(r'', no_punctuation)
        rev = no_non_ascii
        #rev = lower
        #rev = lower
        #rev = str(rev)
        #rev = word_tokenize(rev)
        #rev = [word.translate(table) for word in rev] #remove punktuation
        #rev = [word for word in rev if word.isalpha()] #remove other␣
↪non-alphanumerik tokens
        #rev = [word for word in rev if not word in stop_words]
        #rev = [porter.stem(word) for word in rev]
        normalized_texts.append(rev)
    return normalized_texts

if pre_process == "yes":
    train_texts_p = normalize_texts(train_texts_l)
    test_texts_p = normalize_texts(test_texts_l)
else:
    train_texts_p = train_texts_l
    test_texts_p = test_texts_l
```

```
CPU times: user 2.92 s, sys: 4.81 ms, total: 2.92 s
Wall time: 2.92 s
```

[19]:
```
print(len(train_texts_p))
print(len(test_texts_p))
```

```
49473
10007
```

[20]:
```
print(train_texts_l[0:1])
print("\n")
print(train_texts_p[0:1])
```

```
['Stuning even for the non-gamer: This sound track was beautiful! It paints the
senery in your mind so well I would recomend it even to people who hate vid.
game music! I have played the game Chrono Cross but out of all of the games I
have ever played it has the best music! It backs away from crude keyboarding and
takes a fresher step with grate guitars and soulful orchestras. It would impress
anyone who cares to listen! ^_^']


['stuning even for the non gamer  this sound track was beautiful  it paints the
senery in your mind so well i would recomend it even to people who hate vid
game music  i have played the game chrono cross but out of all of the games i
have ever played it has the best music  it backs away from crude keyboarding and
takes a fresher step with grate guitars and soulful orchestras  it would impress
anyone who cares to listen    ']
```

1.0.10 4. Split Data

[21]:
```
train_texts_s, val_texts_s, train_labels, val_labels =␣
 ↪train_test_split(train_texts_p, train_labels_l, random_state=57643892,␣
 ↪test_size=0.2)

print("Length train texts:", len(train_texts_s))
#print(len(train_labels))

print("Length validation texts:", len(val_texts_s))
#print(len(val_labels))

print("Length test texts", len(test_texts_p))
#print(len(test_labels))
```

```
Length train texts: 39578
Length validation texts: 9895
Length test texts 10007
```

1.1 5 .Tokenize Text

- split texts into lists of tokens.
- assign max features
- creates the vocabulary based on train data
- resulting vectors equal the length of each text

[22]:
```
%%time

MAX_FEATURES = voc_size

tokenizer = Tokenizer(num_words=MAX_FEATURES)
tokenizer.fit_on_texts(train_texts_s)
```

```
CPU times: user 2.68 s, sys: 16.4 ms, total: 2.7 s
Wall time: 2.7 s
```

1.1.1 5.1 Encode training data sentences into sequences

- Transforms each text into a sequence of integers.
- Assigns an integer to each word
- Can access the word index (a dictionary) to verify assigned integer to the word

1.2 Vector

[23]:
```
%%time

train_texts_b = tokenizer.texts_to_sequences(train_texts_s)
val_texts_b = tokenizer.texts_to_sequences(val_texts_s)
test_texts_b = tokenizer.texts_to_sequences(test_texts_p)
```

```
CPU times: user 4.06 s, sys: 22.3 ms, total: 4.08 s
Wall time: 4.11 s
```

1.3 One-Hot

[24]:
```
%%time

if train_size > threshold:
    pass
else:
    train_texts_bm = tokenizer.texts_to_matrix(train_texts_s, mode="binary")
    val_texts_bm = tokenizer.texts_to_matrix(val_texts_s, mode="binary")
    test_texts_bm = tokenizer.texts_to_matrix(test_texts_p, mode="binary")
```

```
CPU times: user 6.99 s, sys: 415 ms, total: 7.41 s
Wall time: 7.42 s
```

[25]:
```
print(train_texts_b[0])
print(len(train_texts_b[0]))
print(" ")

if train_size > threshold:
    pass
else:
```

```
    print(train_texts_bm.shape)
    print(len(train_texts_bm))
```

```
[235, 1992, 123, 29, 106, 1404, 5, 162, 240, 154, 20, 43, 104, 7, 91, 290, 374,
96, 27, 1598, 719, 5, 1275, 77, 12, 8, 104, 3, 52, 17, 16, 4, 10, 1, 1098, 54,
154, 25, 982, 53, 7, 123, 568, 79, 185, 447, 18, 54, 677, 1528, 272, 26, 19,
519, 185, 1222, 10, 4, 34, 99, 1844, 7, 78, 466, 3, 5, 78, 25, 1, 113, 1201, 18,
36, 118, 61, 771, 5, 1, 7, 1, 571, 12, 171, 65, 69, 78, 8, 104, 24, 73, 24, 2,
69, 589, 47, 24, 70, 54, 104, 9, 1, 82, 7, 310]
104

(39578, 2000)
39578
```

5.2 Show an encoded Sequence

[26]:
```
print("First review:", train_texts_s[0], end=" ")
print("\n")
print("First encoded review:", train_texts_b[0], end=" ")
print("\n")

print("Lenth before encoding", len(train_texts_s[0]))
print("Lenth after encoding", len(train_texts_b[0]))

#print("wonderful", tokenizer.word_index["wonderful"])
#print("inspiring", tokenizer.word_index["inspiring"])
```

```
First review: wonderful  inspiring music   so many artists struggle to put 10
songs on an album  of which maybe half could be considered decent  joseph arthur
manages to create 1 for this album and there s not a loser in the bunch his
songs are pure poetry surrounded by swirling layers of gorgeous music
sometimes simplistic folk  other times upbeat rock  but his lyrics carry each
one with often times devastating results  in a good way   tales of love lost and
struggles to love are the most common  but they never get tiring due to the
diversity of the tracks for those who do love this album as much as i do  check
out gavin degraw as well  his album chariot is arguably the best of 00  ebhp

First encoded review: [235, 1992, 123, 29, 106, 1404, 5, 162, 240, 154, 20, 43,
104, 7, 91, 290, 374, 96, 27, 1598, 719, 5, 1275, 77, 12, 8, 104, 3, 52, 17, 16,
4, 10, 1, 1098, 54, 154, 25, 982, 53, 7, 123, 568, 79, 185, 447, 18, 54, 677,
1528, 272, 26, 19, 519, 185, 1222, 10, 4, 34, 99, 1844, 7, 78, 466, 3, 5, 78,
25, 1, 113, 1201, 18, 36, 118, 61, 771, 5, 1, 7, 1, 571, 12, 171, 65, 69, 78, 8,
104, 24, 73, 24, 2, 69, 589, 47, 24, 70, 54, 104, 9, 1, 82, 7, 310]

Lenth before encoding 684
Lenth after encoding 104
```

5.3. Unique tokens and Document Count

[27]:
```
print('Found %d unique words.' % len(tokenizer.word_index))
print("Documents",tokenizer.document_count)
```

```
Found 64191 unique words.
Documents 39578
```

5.4 Word Index (according to its frequency)

[28]:
```
print("First five:", list(tokenizer.word_index.items())[0:5])
print("Last five:", list(tokenizer.word_index.items())[-5:])
print("500th to 505th:", list(tokenizer.word_index.items())[500:505])
```

```
First five: [('the', 1), ('i', 2), ('and', 3), ('a', 4), ('to', 5)]
Last five: [('revengeful', 64187), ('dices', 64188), ('laryngitis', 64189),
('guitarrist', 64190), ('punchless', 64191)]
500th to 505th: [('mr', 501), ('working', 502), ('entire', 503), ('name', 504),
('totally', 505)]
```

[29]:
```
#tokenizer.word_index["the"]
```

5.5 Word Counts

[30]:
```
print("First five:", list(tokenizer.word_counts.items())[:5])
print("Last five:", list(tokenizer.word_counts.items())[-5:])
print("100th to 105th:", list(tokenizer.word_counts.items())[100:105])

#print(tokenizer.word_counts["the"])
```

```
First five: [('wonderful', 1724), ('inspiring', 132), ('music', 3601), ('so',
13166), ('many', 3935)]
Last five: [('revengeful', 1), ('dices', 1), ('laryngitis', 1), ('guitarrist',
1), ('punchless', 1)]
100th to 105th: [('0', 3342), ('just', 10498), ('over', 3880), ('month', 620),
('now', 3644)]
```

5.6 Count of Words in Documents

[31]:
```
print("Count of words in Documents:")

print(list(tokenizer.word_docs.items())[0:5])
print(list(tokenizer.word_docs.items())[-5:])
```

```
Count of words in Documents:
[('many', 3367), ('is', 26455), ('an', 7772), ('often', 631), ('are', 11720)]
[('revengeful', 1), ('dices', 1), ('laryngitis', 1), ('guitarrist', 1),
('punchless', 1)]
```

[32]:
```
#tokenizer.word_docs["the"]
```

1.4 6. Padding with Keras

- text lengths are not be uniform
- a neural network requeres it
- select a maximum length
- pad shorter sentences with 0
- needed, to use batches effectively
- equal the length of the longest sentance

[33]:
```
%%time

MAX_LENGTH = max(len(train_ex) for train_ex in train_texts_b)

train_texts = pad_sequences(train_texts_b, maxlen=MAX_LENGTH)
val_texts = pad_sequences(val_texts_b, maxlen=MAX_LENGTH)
test_texts = pad_sequences(test_texts_b, maxlen=MAX_LENGTH)

#train_texts = train_texts_b
#val_texts = val_texts_b
#test_texts = test_texts_b
```

```
CPU times: user 1.46 s, sys: 52.9 ms, total: 1.51 s
Wall time: 1.49 s
```

[34]:
```
print("Max length of a sequence:", max(len(train_ex) for train_ex in␣
↪train_texts_b))
print("Min length of a sequence:", min(len(train_ex) for train_ex in␣
↪train_texts_b))
```

```
Max length of a sequence: 233
Min length of a sequence: 1
```

[35]:
```
#print("Max length of a sequence:", max(len(train_ex) for train_ex in␣
↪train_texts_bm))
#print("Min length of a sequence:", min(len(train_ex) for train_ex in␣
↪train_texts_bm))
```

6.2 Length Before Padding

[36]:
```
print("First sequence before padding: \n", val_texts_b[0], end=" ")
print("\n")
print("Length before padding:", len(val_texts_b[0]))
```

```
First sequence before padding:
 [4, 260, 1143, 159, 63, 7, 1, 1842, 3, 25, 295, 106, 25, 85, 1143, 52, 25, 4,
177, 7, 60, 831, 621, 91, 9, 356, 18, 1, 108, 66, 21, 247, 111, 2, 102, 1, 465,
7, 8, 15, 9, 35, 1728, 89, 66, 21, 196, 117, 38, 28, 163, 30, 94, 25, 536, 265,
18, 12, 1]
```

```
Length before padding: 59
```

6.3 Length after Padding

[37]:
```python
print("First sequence after padding: \n", val_texts[0], end=" ")
print("\n")
print("Length of this observation:", len(val_texts[0]))
print("Length of 501st observation:", len(val_texts[501]))
```

```
First sequence after padding:
 [    0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    4  260 1143  159   63    7    1 1842
     3   25  295  106   25   85 1143   52   25    4  177    7   60  831
   621   91    9  356   18    1  108   66   21  247  111    2  102    1
   465    7    8   15    9   35 1728   89   66   21  196  117   38   28
   163   30   94   25  536  265   18   12    1]

Length of this observation: 233
Length of 501st observation: 233
```

1.5 7. Neural Network

Input Parameters into Neural Network

- Embedding Layer
- input dimention: size of the vocabulary
- output dimention: embedding size
- learned embedding
- for dense layer include length of input sequences

[38]:
```python
vocab_size = len(tokenizer.word_index) + 1
length = min(len(train_ex) for train_ex in train_texts)
embedding_dim = 100

print("Total Vocab size:", vocab_size)
print("Features to use:", MAX_FEATURES)
```

```
print("Length:", length)
print("Train shape:", train_texts.shape)
```

```
Total Vocab size: 64192
Features to use: 2000
Length: 233
Train shape: (39578, 233)
```

1.5.1 7.1 Define Function to Plot Epochs

[39]:
```
import matplotlib.pyplot as plt
plt.style.use('ggplot')

def plot_history(history):
    acc = history.history['acc']
    val_acc = history.history['val_acc']
    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()
```

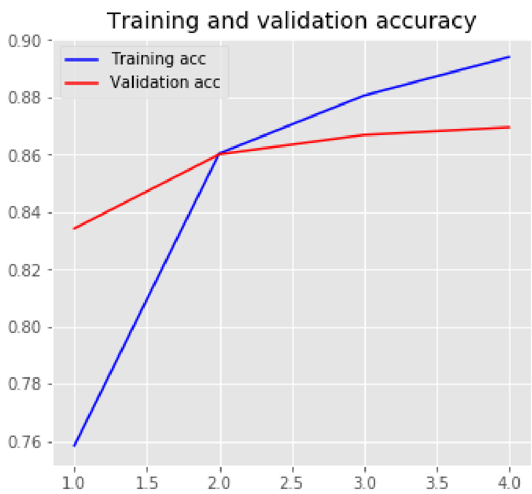
1.5.2 7.2 With Embedding Layer

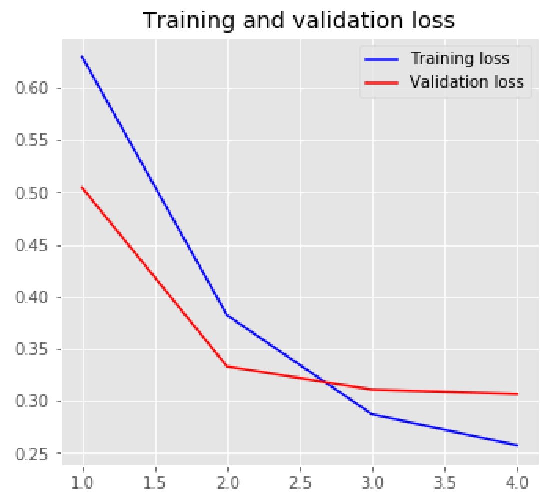
[40]:
```
%%time

model = Sequential()
model.add(layers.Embedding(MAX_FEATURES, embedding_dim, input_length=length))

model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

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

```
history = model.fit(train_texts, train_labels,
                    epochs=epo,
                    verbose=True,
                    validation_data=(val_texts, val_labels),
                    batch_size=batches)

loss, accuracy = model.evaluate(val_texts, val_labels, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))

loss, accuracy_emb = model.evaluate(test_texts, test_labels, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy_emb))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 233, 100)          200000
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1010
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 11
=================================================================
Total params: 201,021
Trainable params: 201,021
Non-trainable params: 0
_________________________________________________________________
Train on 39578 samples, validate on 9895 samples
Epoch 1/4
39578/39578 [==============================] - 4s 93us/step - loss: 0.6294 -
acc: 0.7585 - val_loss: 0.5042 - val_acc: 0.8343
Epoch 2/4
39578/39578 [==============================] - 1s 20us/step - loss: 0.3821 -
acc: 0.8604 - val_loss: 0.3330 - val_acc: 0.8601
Epoch 3/4
39578/39578 [==============================] - 1s 20us/step - loss: 0.2872 -
acc: 0.8806 - val_loss: 0.3107 - val_acc: 0.8669
Epoch 4/4
39578/39578 [==============================] - 1s 20us/step - loss: 0.2574 -
acc: 0.8940 - val_loss: 0.3066 - val_acc: 0.8695
Training Accuracy: 0.8695
Testing Accuracy:  0.8741
CPU times: user 5.45 s, sys: 987 ms, total: 6.44 s
Wall time: 7.09 s
```

[41]:
```
plot_history(history)
```





The loss and the accuracy of the model are reported at the end of each training epoch, and finally, the accuracy of the model on the test dataset is reported.

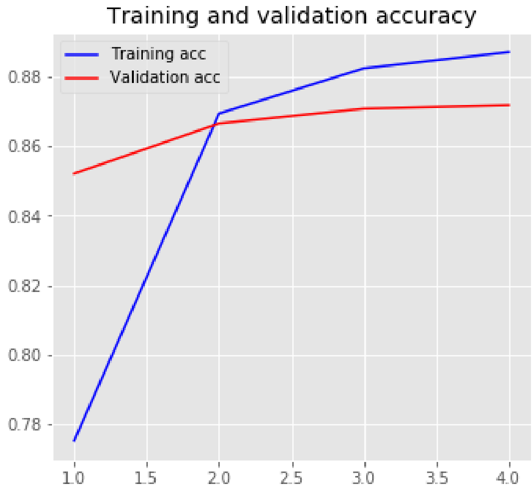
1.5.3 7.2 With One-Hot-Encoding

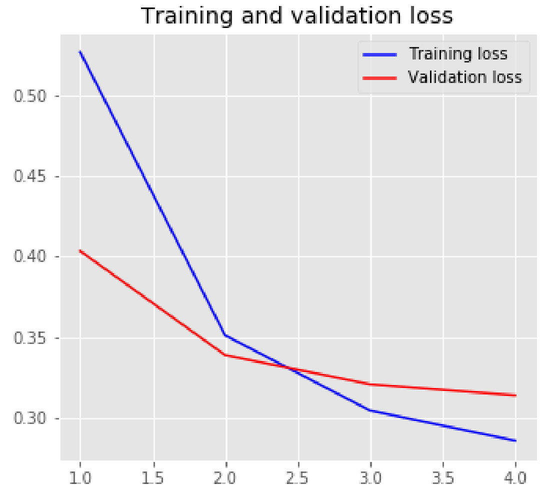
[42]:
```
%%time

if train_size > threshold:
    pass
else:

    print(train_texts_bm.shape)

    model = Sequential()
    model.add(layers.InputLayer(input_shape=(train_texts_bm.shape[1],)))

    model.add(layers.Dense(10, activation='relu'))
    model.add(layers.Dense(1, activation='sigmoid'))

    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    model.summary()

    history = model.fit(train_texts_bm, train_labels,
                        epochs=epo,
                        verbose=True,
                        validation_data=(val_texts_bm, val_labels),
                        batch_size=batches)
```

```
loss, accuracy = model.evaluate(val_texts_bm, val_labels, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))

loss, accuracy_oh = model.evaluate(test_texts_bm, test_labels,␣
↪verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy_oh))
```

```
(39578, 2000)
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 10)                20010
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 11
=================================================================
Total params: 20,021
Trainable params: 20,021
Non-trainable params: 0
_________________________________________________________________
Train on 39578 samples, validate on 9895 samples
Epoch 1/4
39578/39578 [==============================] - 1s 19us/step - loss: 0.5265 -
acc: 0.7752 - val_loss: 0.4033 - val_acc: 0.8521
Epoch 2/4
39578/39578 [==============================] - 1s 13us/step - loss: 0.3511 -
acc: 0.8694 - val_loss: 0.3389 - val_acc: 0.8666
Epoch 3/4
39578/39578 [==============================] - 1s 13us/step - loss: 0.3046 -
acc: 0.8825 - val_loss: 0.3208 - val_acc: 0.8708
Epoch 4/4
39578/39578 [==============================] - 1s 16us/step - loss: 0.2860 -
acc: 0.8872 - val_loss: 0.3140 - val_acc: 0.8719
Training Accuracy: 0.8719
Testing Accuracy:  0.8741
CPU times: user 4.04 s, sys: 317 ms, total: 4.36 s
Wall time: 3.53 s
```

[43]:
```
plot_history(history)
```





1.6 7.3 Neural Network on Integer Data

[44]:
```
%%time

print(train_texts.shape)

if train_size > threshold:
    pass
else:

    print(train_texts.shape)

    model = Sequential()
    model.add(layers.InputLayer(input_shape=(train_texts.shape[1],)))

    model.add(layers.Dense(10, activation='relu'))
    model.add(layers.Dense(1, activation='sigmoid'))

    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    model.summary()

    history = model.fit(train_texts, train_labels,
                        epochs=20,
                        verbose=True,
                        validation_data=(val_texts, val_labels),
                        batch_size=batches)
```

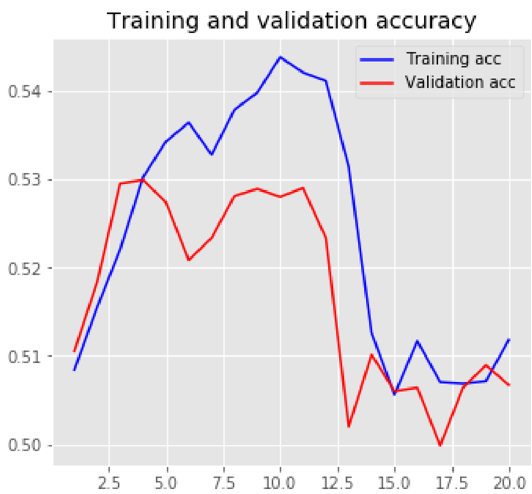
```python
loss, accuracy = model.evaluate(val_texts, val_labels, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))

loss, accuracy_int = model.evaluate(test_texts, test_labels, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy_oh))
```

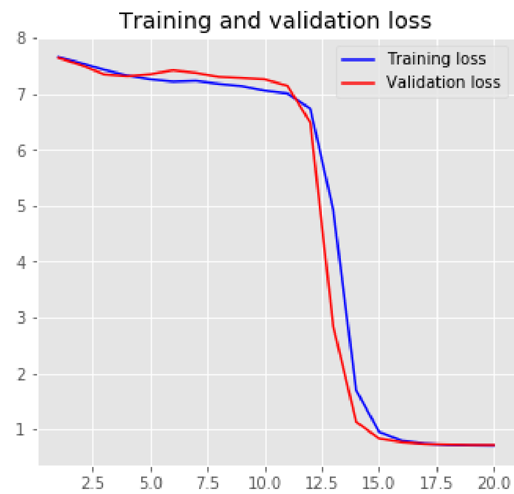
```
(39578, 233)
(39578, 233)
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 10)                2340
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 11
=================================================================
Total params: 2,351
Trainable params: 2,351
Non-trainable params: 0
_________________________________________________________________
Train on 39578 samples, validate on 9895 samples
Epoch 1/20
39578/39578 [==============================] - 1s 13us/step - loss: 7.6583 -
acc: 0.5084 - val_loss: 7.6438 - val_acc: 0.5106
Epoch 2/20
39578/39578 [==============================] - 0s 7us/step - loss: 7.5506 - acc:
0.5156 - val_loss: 7.5147 - val_acc: 0.5184
Epoch 3/20
39578/39578 [==============================] - 0s 7us/step - loss: 7.4319 - acc:
0.5221 - val_loss: 7.3494 - val_acc: 0.5296
Epoch 4/20
39578/39578 [==============================] - 0s 7us/step - loss: 7.3301 - acc:
0.5303 - val_loss: 7.3173 - val_acc: 0.5300
Epoch 5/20
39578/39578 [==============================] - 0s 7us/step - loss: 7.2608 - acc:
0.5343 - val_loss: 7.3524 - val_acc: 0.5274
Epoch 6/20
39578/39578 [==============================] - 0s 7us/step - loss: 7.2202 - acc:
0.5365 - val_loss: 7.4233 - val_acc: 0.5209
Epoch 7/20
39578/39578 [==============================] - 0s 7us/step - loss: 7.2339 - acc:
0.5328 - val_loss: 7.3768 - val_acc: 0.5234
Epoch 8/20
39578/39578 [==============================] - 0s 7us/step - loss: 7.1792 - acc:
0.5379 - val_loss: 7.3053 - val_acc: 0.5281
Epoch 9/20
39578/39578 [==============================] - 0s 7us/step - loss: 7.1395 - acc:
```

```
0.5399 - val_loss: 7.2851 - val_acc: 0.5290
Epoch 10/20
39578/39578 [==============================] - 0s 7us/step - loss: 7.0584 - acc:
0.5439 - val_loss: 7.2609 - val_acc: 0.5280
Epoch 11/20
39578/39578 [==============================] - 0s 7us/step - loss: 7.0072 - acc:
0.5421 - val_loss: 7.1454 - val_acc: 0.5291
Epoch 12/20
39578/39578 [==============================] - 0s 7us/step - loss: 6.7349 - acc:
0.5412 - val_loss: 6.4856 - val_acc: 0.5234
Epoch 13/20
39578/39578 [==============================] - 0s 7us/step - loss: 4.9227 - acc:
0.5315 - val_loss: 2.8432 - val_acc: 0.5020
Epoch 14/20
39578/39578 [==============================] - 0s 7us/step - loss: 1.7005 - acc:
0.5126 - val_loss: 1.1352 - val_acc: 0.5102
Epoch 15/20
39578/39578 [==============================] - 0s 7us/step - loss: 0.9543 - acc:
0.5056 - val_loss: 0.8347 - val_acc: 0.5060
Epoch 16/20
39578/39578 [==============================] - 0s 7us/step - loss: 0.7955 - acc:
0.5117 - val_loss: 0.7690 - val_acc: 0.5064
Epoch 17/20
39578/39578 [==============================] - 0s 7us/step - loss: 0.7435 - acc:
0.5070 - val_loss: 0.7370 - val_acc: 0.4998
Epoch 18/20
39578/39578 [==============================] - 0s 7us/step - loss: 0.7215 - acc:
0.5069 - val_loss: 0.7297 - val_acc: 0.5064
Epoch 19/20
39578/39578 [==============================] - 0s 7us/step - loss: 0.7138 - acc:
0.5072 - val_loss: 0.7214 - val_acc: 0.5089
Epoch 20/20
39578/39578 [==============================] - 0s 7us/step - loss: 0.7091 - acc:
0.5118 - val_loss: 0.7210 - val_acc: 0.5067
Training Accuracy: 0.5067
Testing Accuracy:  0.5121
CPU times: user 8.76 s, sys: 662 ms, total: 9.42 s
Wall time: 6.7 s
```

[45]:

```
plot_history(history)
```





1.7 8. Baseline Model: Logistic Regression

[46]:

```
%%time

if train_size > threshold:
    pass
else:
    print(train_texts_bm.shape)
    print(train_texts.shape)
    print(length)

    classifier = LogisticRegression(penalty='l2',C=1)
    classifier.fit(train_texts_bm, train_labels)

    score = classifier.score(val_texts_bm, val_labels)
    print("Accuracy Val:", score)

    preds = classifier.predict(test_texts_bm)
    cm = confusion_matrix(test_labels, preds)
    accuracy_log = accuracy_score(test_labels , preds)
    print("Accuracy Test:", accuracy_log)
    print("Size of Test sample:", len(test_texts_bm))
```

```
(39578, 2000)
(39578, 233)
233
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:433:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

```
Accuracy Val: 0.8722587165234967
Accuracy Test: 0.8706905166383532
Size of Test sample: 10007
CPU times: user 1.26 s, sys: 54.5 ms, total: 1.31 s
Wall time: 1.23 s
```

2 9. Results Summary

[47]:
```
print("Results Summary:" "\n")

print("Testing Accuracy of Embedding:  {:.4f}".format(round(accuracy_emb,3)))

if train_size > threshold:
    pass
else:
    print("Testing Accuracy of Integer-Encoding:  {:.4f}".
 ↪format(round(accuracy_int,3)))
    print("Testing Accuracy of One-Hot-Encoding:  {:.4f}".
 ↪format(round(accuracy_oh,3)))
    print("Testing Accuracy of Logistic Regression:", round(accuracy_log,3))


toc = time.time()
print("\n" "Parameters:", "\n" "vocab size:", voc_size, "\n" "train size:",␣
 ↪train_size, "test size:", test_size, "\n" "pre-process:", pre_process, "\n"␣
 ↪"epochs:", epo, "\n" "batch size:", batches, "\n" "time:", (toc-tic)/60)
#print("NLTK punktuation and non-ascii")
```

```
Results Summary:

Testing Accuracy of Embedding:  0.8730
Testing Accuracy of Integer-Encoding:  0.5130
Testing Accuracy of One-Hot-Encoding:  0.8720
Testing Accuracy of Logistic Regression: 0.871

Parameters:
vocab size: 2000
train size: 50000 test size: 10000
pre-process: yes
epochs: 4
batch size: 512
time: 0.7004092772801717
```

[48]:
```
#print("All Pre-Processing steps possible with NLTK only batch size: 64")
```