Welcome to Session 03 [Long Short-Term Memory (LSTM)]

This is a Google Colaboratory (https://colab.research.google.com/notebooks/welcome.ipynb) notebook file. Python programs are run directly in the browser—a great way to learn and use TensorFlow. To follow this tutorial, run the notebook in Google Colab by clicking the button at the top of this page.

1. In Colab, connect to a Python runtime: At the top-right of the menu bar, select *CONNECT*.
2. Run all the notebook code cells: Select *Runtime* > *Run all*.

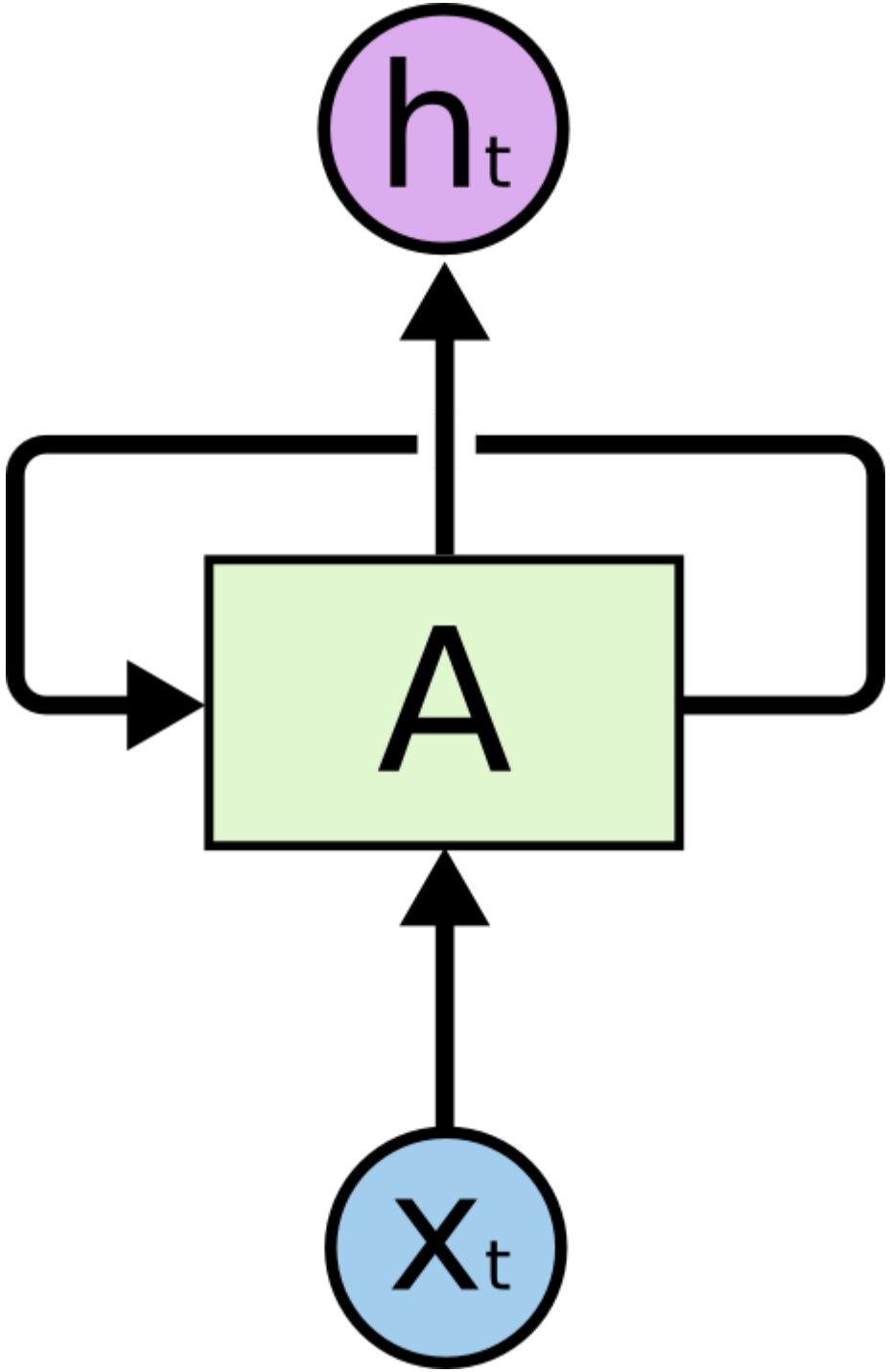
RNN Model

Recurrent Neural Networks

Humans don’t start their thinking from scratch every second. As you read this essay, you understand each word based on your understanding of previous words. You don’t throw everything away and start thinking from scratch again. Your thoughts have persistence.

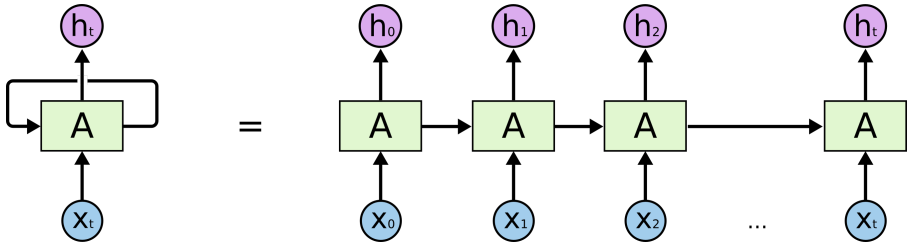
Traditional neural networks can’t do this, and it seems like a major shortcoming. For example, imagine you want to classify what kind of event is happening at every point in a movie. It’s unclear how a traditional neural network could use its reasoning about previous events in the film to inform later ones.

Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.


h_t
A
X_t

In the above diagram, a chunk of neural network, A , looks at some input x_t and outputs a value h_t . A loop allows information to be passed from one step of the network to the next.

These loops make recurrent neural networks seem kind of mysterious. However, if you think a bit more, it turns out that they aren't all that different than a normal neural network. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop:



This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They're the natural architecture of neural network to use for such data.

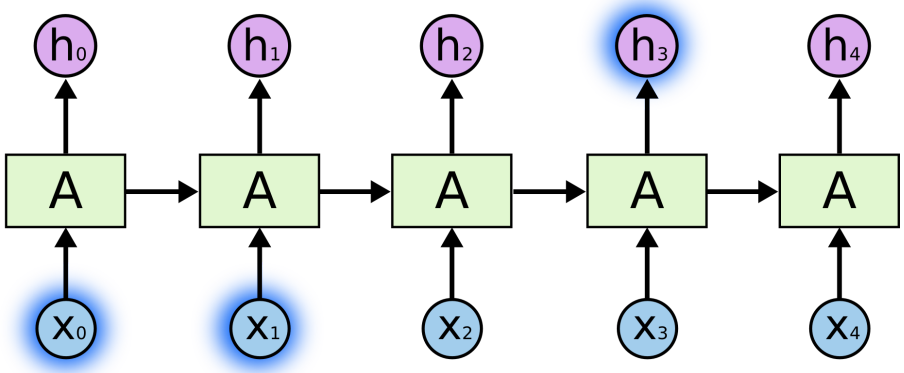
And they certainly are used! In the last few years, there have been incredible success applying RNNs to a variety of problems: speech recognition, language modeling, translation, image captioning… The list goes on. I'll leave discussion of the amazing feats one can achieve with RNNs to Andrej Karpathy's excellent blog post, The Unreasonable Effectiveness of Recurrent Neural Networks. But they really are pretty amazing.

Essential to these successes is the use of "LSTMs," a very special kind of recurrent neural network which works, for many tasks, much much better than the standard version. Almost all exciting results based on recurrent neural networks are achieved with them. It's these LSTMs that this essay will explore.

The Problem of Long-Term Dependencies

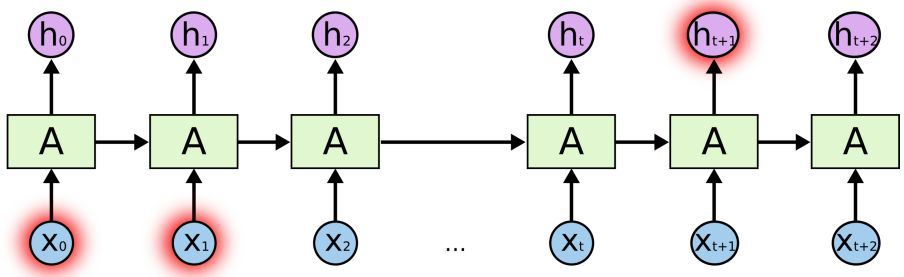
One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task, such as using previous video frames might inform the understanding of the present frame. If RNNs could do this, they'd be extremely useful. But can they? It depends.

Sometimes, we only need to look at recent information to perform the present task. For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in "the clouds are in the sky," we don't need any further context – it's pretty obvious the next word is going to be sky. In such cases, where the gap between the relevant information and the place that it's needed is small, RNNs can learn to use the past information.



But there are also cases where we need more context. Consider trying to predict the last word in the text “I grew up in France… I speak fluent French.” Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It’s entirely possible for the gap between the relevant information and the point where it is needed to become very large.

Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.

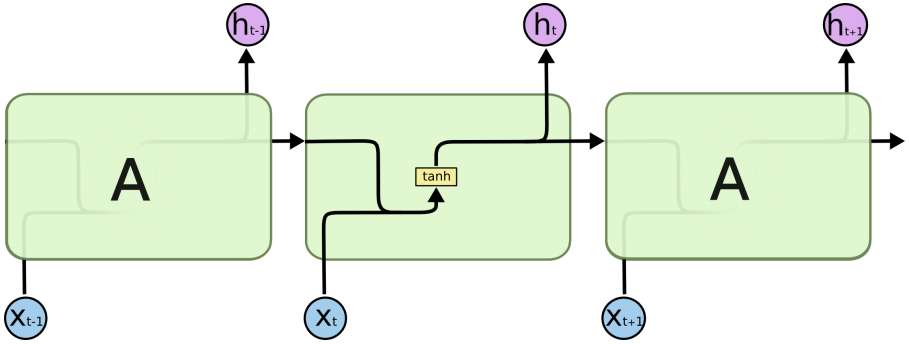


In theory, RNNs are absolutely capable of handling such “long-term dependencies.” A human could carefully pick parameters for them to solve toy problems of this form. Sadly, in practice, RNNs don’t seem to be able to learn them. Thankfully, LSTMs don’t have this problem!

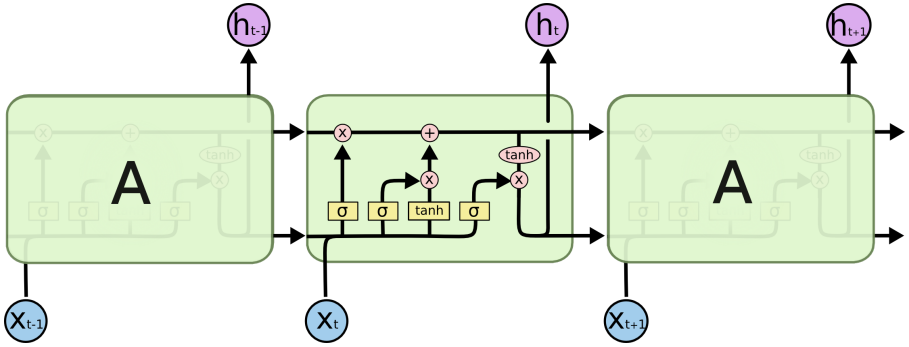
LSTM Networks

Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies. LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

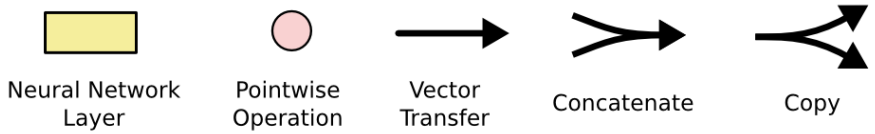
All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.



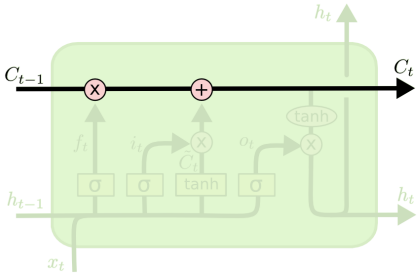
LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



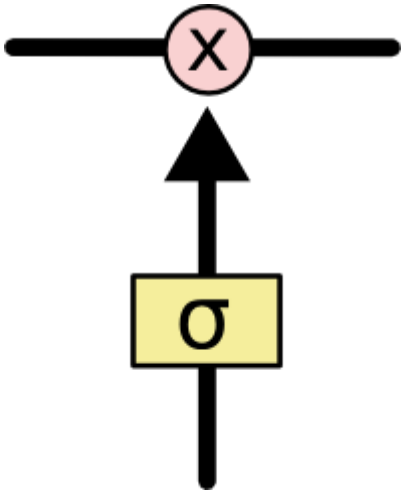
Don’t worry about the details of what’s going on. We’ll walk through the LSTM diagram step by step later. For now, let’s just try to get comfortable with the notation we’ll be using.



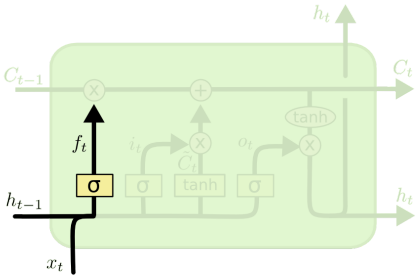
In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations. ## The Core Idea Behind LSTMs The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It’s very easy for information to just flow along it unchanged.



The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

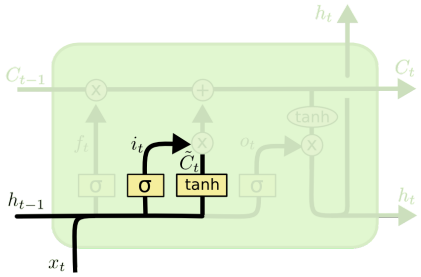


The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means “let nothing through,” while a value of one means “let everything through!” An LSTM has three of these gates, to protect and control the cell state. ## Step-by-Step LSTM Walk Through The first step in our LSTM is to decide what information we’re going to throw away from the cell state. This decision is made by a sigmoid layer called the “forget gate layer.” It looks at h_{t-1} and x_t , and outputs a number between 0 and 1 for each number in the cell state C_{t-1} . A 1 represents “completely keep this” while a 0 represents “completely get rid of this.” Let’s go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.



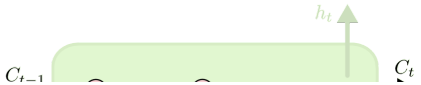
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state. In the next step, we'll combine these two to create an update to the state. In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

It's now time to update the old cell state, C_{t-1} , into the new cell state C_t . The previous steps already decided what to do, we just need to actually do it. We multiply the old state by f_t , forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$ This is the new candidate values, scaled by how much we decided to update each state value. In the case of the language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in the previous steps.



Implementing The Model

Initialization

Mounting

In [0]:
```
# Mounting Gdrive

USE_G_COLAB = True

if USE_G_COLAB:
    from google.colab import drive


    drive.mount('/content/drive', force_remount=True)
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk
8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Ao
ob&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.
com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2
Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:
```
# Project Root

root_dir = ''

if USE_G_COLAB:
    root_dir = '/content/drive/My Drive/workshops/2019_07_21/sessions_01/'
```

Install Requirement

In [0]:
```
!pip install -q tensorflow-gpu==2.0.0-beta1
!pip install -qU watermark
!pip install -qU persiantools
!pip install -qU faker
```

```
     |████████████████████████████████| 880kB 2.8MB/s
```

Custom Matplotlib Style

In [0]:
```
mpl_style = "https://gist.githubusercontent.com/m3hrdadfi/af8aca01094afb7d3e5b46de9ad8d509/raw/871ec5d721a3b438c3c896718ea4aafc91ea9744/gadfly.mplstyle"

!wget -q $mpl_style -O /root/.config/matplotlib/matplotlibrc
```

General Paramas

A random seed is a number used to initialize a pseudorandom number generator. For a seed to be used in a pseudorandom number generator, it does not need to be random

In [0]:
```
RANDOM_SEED = 141
```

Import requried packages

In [0]:
```
import tensorflow as tf

import requests
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.utils import shuffle

import matplotlib as mpl
import matplotlib.pyplot as plt

import cv2

import random
import sys
import io
import re
import time
from datetime import date, datetime, time
import os
import struct
import itertools
from tqdm import tqdm

from pprint import pprint


%matplotlib inline
mpl.rc_file(mpl.matplotlib_fname())
```

Load the data

In [0]:
```
!wget https://www.dropbox.com/s/o8qcunpgw88of4r/Sentiment.zip
!unzip Sentiment.zip
!ls
```

```
--2019-07-28 05:28:44--  https://www.dropbox.com/s/o8qcunpgw88of4r/Sentiment.zip
Resolving www.dropbox.com (www.dropbox.com)... 162.125.82.1, 2620:100:6032:1::a27d:5201
Connecting to www.dropbox.com (www.dropbox.com)|162.125.82.1|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/o8qcunpgw88of4r/Sentiment.zip [following]
--2019-07-28 05:28:45--  https://www.dropbox.com/s/raw/o8qcunpgw88of4r/Sentiment.zip
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uc4549fe2d649d3d15a7daea9812.dl.dropboxusercontent.com/cd/0/inline/AliKrJi2ynPJvYs
YRpWN0LwHpid0d6NC-CcJoUMI81OVdMVEEYW-adGXhuq5EDVBqKbTLDo63G5-CQswlUcgnsdtYLxSW-jCw5WAFf05b2sKuw/file
# [following]
--2019-07-28 05:28:45--  https://uc4549fe2d649d3d15a7daea9812.dl.dropboxusercontent.com/cd/0/inline/
AliKrJi2ynPJvYsYRpWN0LwHpid0d6NC-CcJoUMI81OVdMVEEYW-adGXhuq5EDVBqKbTLDo63G5-CQswlUcgnsdtYLxSW-jCw5WA
Ff05b2sKuw/file
Resolving uc4549fe2d649d3d15a7daea9812.dl.dropboxusercontent.com (uc4549fe2d649d3d15a7daea9812.dl.dr
opboxusercontent.com)... 162.125.82.6, 2620:100:6032:6::a27d:5206
Connecting to uc4549fe2d649d3d15a7daea9812.dl.dropboxusercontent.com (uc4549fe2d649d3d15a7daea9812.d
l.dropboxusercontent.com)|162.125.82.6|:443... connected.
HTTP request sent, awaiting response... 302 FOUND
Location: /cd/0/inline2/AlhnXMq9Ti1dmugudYnLg2i6BcVJxXJFBJfkn2vhmGc2YeZ67r3TdNRQOaS_29I2WYDgwGkLJ6FB
Ao78rJoGMyFmeNfrC7az46sop3TbwhGrQYL7fI7CokhxRSF671XbnToMQ_sFTRv02ViKP4ISFmbnIbGdES3ypsPovoZvD5jXycnY
A_qU16O0s7Pi9FILvrW2-Y6en4R9y112W7wX5siJki2Rd9y3qLRo-u6NYMifeGzkiPbWpKu-4gs-jDOceX_S82g-AYIMCe3KtS9d
0d2b-K6yqknRGIPJI60v-dYONLuwG2dvzwQjCOmPDF3ckT3gyfT1iyjLtnCsgavIb9IS/file [following]
--2019-07-28 05:28:46--  https://uc4549fe2d649d3d15a7daea9812.dl.dropboxusercontent.com/cd/0/inline
2/AlhnXMq9Ti1dmugudYnLg2i6BcVJxXJFBJfkn2vhmGc2YeZ67r3TdNRQOaS_29I2WYDgwGkLJ6FBAo78rJoGMyFmeNfrC7az46
sop3TbwhGrQYL7fI7CokhxRSF671XbnToMQ_sFTRv02ViKP4ISFmbnIbGdES3ypsPovoZvD5jXycnYA_qU16O0s7Pi9FILvrW2-Y
6en4R9y112W7wX5siJki2Rd9y3qLRo-u6NYMifeGzkiPbWpKu-4gs-jDOceX_S82g-AYIMCe3KtS9d0d2b-K6yqknRGIPJI60v-d
YONLuwG2dvzwQjCOmPDF3ckT3gyfT1iyjLtnCsgavIb9IS/file
Reusing existing connection to uc4549fe2d649d3d15a7daea9812.dl.dropboxusercontent.com:443.
HTTP request sent, awaiting response... 200 OK
Length: 1136822 (1.1M) [application/zip]
Saving to: ‘Sentiment.zip’

Sentiment.zip       100%[===================>]   1.08M  --.-KB/s    in 0.08s

2019-07-28 05:28:46 (13.7 MB/s) - ‘Sentiment.zip’ saved [1136822/1136822]

Archive:  Sentiment.zip
  inflating: sentiment.csv
   creating: __MACOSX/
  inflating: __MACOSX/._sentiment.csv
drive  __MACOSX  sample_data  sentiment.csv  Sentiment.zip
```

In [0]:
```
data = pd.read_csv('./sentiment.csv')
data = data[['text', 'sentiment']]
data = data[data.sentiment != "Neutral"]
data.head()
```

Out[0]:

	text	sentiment
1	RT @ScottWalker: Didn't catch the full #GOPdeb...	Positive
3	RT @RobGeorge: That Carly Fiorina is trending ...	Positive
4	RT @DanScavino: #GOPDebate w/ @realDonaldTrump...	Positive
5	RT @GregAbbott_TX: @TedCruz: "On my first day ...	Positive
6	RT @warriorwoman91: I liked her and was happy ...	Negative

Preprocessing

In [0]:
```
def cleanize(text):
    text = text.lower()
    text = re.sub('[^a-zA-z0-9\s]', '', text)
    text = text.replace('rt', ' ')
    return text
```

In [0]:
```
data['text'] = data['text'].apply(lambda t: cleanize(t))

print(data[ data['sentiment'] == 'Positive'].size)
print(data[ data['sentiment'] == 'Negative'].size)
```

```
4472
16986
```

In [0]:
```
data.head()
```

Out[0]:

	text	sentiment
1	scottwalker didnt catch the full gopdebate l...	Positive
3	robgeorge that carly fiorina is trending ho...	Positive
4	danscavino gopdebate w realdonaldtrump deliv...	Positive
5	gregabbott_tx tedcruz on my first day i will...	Positive
6	warriorwoman91 i liked her and was happy whe...	Negative

Preparing

In [0]:
```
max_fatures = 2000
tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=max_fatures, split=' ')
tokenizer.fit_on_texts(data['text'].values)
x_texts = tokenizer.texts_to_sequences(data['text'].values)
x_texts = tf.keras.preprocessing.sequence.pad_sequences(x_texts)

print(x_texts.shape)
```

```
(10729, 28)
```

Model

In [0]:
```
def build_model(input_dim, output_dim, max_features, embedding_dim, rnn_units, dnn_units):
    inputs = tf.keras.layers.Input(shape=[input_dim])
    embedding = tf.keras.layers.Embedding(max_features, embedding_dim)

    x = embedding(inputs)
    x = tf.keras.layers.LSTM(rnn_units)(x)
    x = tf.keras.layers.Dense(dnn_units)(x)

    outputs = tf.keras.layers.Dense(output_dim, activation='softmax')(x)

    model = tf.keras.Model(inputs=inputs, outputs=outputs)
    return model
```

In [0]:
```
model = build_model(28, 2, 2000, 100, 128, 256)
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 28)]              0
_________________________________________________________________
embedding_2 (Embedding)      (None, 28, 100)           200000
_________________________________________________________________
lstm (LSTM)                  (None, 128)               117248
_________________________________________________________________
dense (Dense)                (None, 256)               33024
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 514
=================================================================
Total params: 350,786
Trainable params: 350,786
Non-trainable params: 0
_________________________________________________________________
```

In [0]:
```
y = pd.get_dummies(data['sentiment']).values

x_train, x_test, y_train, y_test = train_test_split(x_texts, y, test_size=0.3, random_state=RANDOM_SEED)

print(x_train.shape, y_train.shape)
print(x_test.shape, y_test.shape)
```

```
(7510, 28) (7510, 2)
(3219, 28) (3219, 2)
```

In [0]:
```
x_train[0], y_train[0]
```

Out[0]:
```
(array([   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        1178,    7,  185,   26,    5,    1,   81, 1709,    4,   34,    7,
        1367,    6, 1099,   19,    7,    2], dtype=int32),
 array([1, 0], dtype=uint8))
```

In [0]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

In [0]:
```
r = model.fit(x_train, y_train,
              validation_split=0.1,
              epochs=10,
              batch_size=64,
              verbose=1)

print(model.evaluate(x_test, y_test, verbose=0))
```

```
Train on 6759 samples, validate on 751 samples
Epoch 1/10
6759/6759 [==============================] - 2s 258us/sample - loss: 0.0833 - accuracy: 0.9661 - val
_loss: 1.1235 - val_accuracy: 0.8123
Epoch 2/10
6759/6759 [==============================] - 2s 251us/sample - loss: 0.0749 - accuracy: 0.9680 - val
_loss: 1.2041 - val_accuracy: 0.8149
Epoch 3/10
6759/6759 [==============================] - 2s 252us/sample - loss: 0.0700 - accuracy: 0.9685 - val
_loss: 1.1297 - val_accuracy: 0.8109
Epoch 4/10
6759/6759 [==============================] - 2s 249us/sample - loss: 0.0759 - accuracy: 0.9685 - val
_loss: 1.1098 - val_accuracy: 0.7963
Epoch 5/10
6759/6759 [==============================] - 2s 249us/sample - loss: 0.0748 - accuracy: 0.9682 - val
_loss: 1.1258 - val_accuracy: 0.8029
Epoch 6/10
6759/6759 [==============================] - 2s 248us/sample - loss: 0.0747 - accuracy: 0.9675 - val
_loss: 1.2071 - val_accuracy: 0.8136
Epoch 7/10
6759/6759 [==============================] - 2s 249us/sample - loss: 0.0668 - accuracy: 0.9706 - val
_loss: 1.2679 - val_accuracy: 0.8149
Epoch 8/10
6759/6759 [==============================] - 2s 249us/sample - loss: 0.0649 - accuracy: 0.9720 - val
_loss: 1.3205 - val_accuracy: 0.8096
Epoch 9/10
6759/6759 [==============================] - 2s 250us/sample - loss: 0.0623 - accuracy: 0.9722 - val
_loss: 1.3719 - val_accuracy: 0.8096
Epoch 10/10
6759/6759 [==============================] - 2s 249us/sample - loss: 0.0603 - accuracy: 0.9735 - val
_loss: 1.4836 - val_accuracy: 0.8096
[1.3641104665962425, 0.8313141]
```

In [0]:

In [0]:
```
twt = ['Meetings: Because none of us is as dumb as all of us.']
print(twt)

twt = tokenizer.texts_to_sequences(twt)
twt = tf.keras.preprocessing.sequence.pad_sequences(twt, maxlen=28, dtype='int32', value=0)
print(twt)

sentiment = model.predict(twt, batch_size=1, verbose=0)[0]

if(np.argmax(sentiment) == 0):
    print("negative")
elif(np.argmax(sentiment) == 1):
    print("positive")
```

```
['Meetings: Because none of us is as dumb as all of us.']
[[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0  206  633    6  150    5   55 1055   55   46    6  150]]
negative
```

Resources

- LSTM Official Paper (https://www.bioinf.jku.at/publications/older/2604.pdf)
- Understanding LSTMs (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)
- Sequence Modeling (https://www.deeplearningbook.org/slides/10_rnn.pdf)
- RNN Effectiveness (http://karpathy.github.io/2015/05/21/rnn-effectiveness/)

In [0]: