IMDB Sentiment Prediction with Recurrent Neural Network

In [1]:
```
import os
import pandas as pd
import numpy as np

from keras.models import Sequential, load_model
from keras.layers import Dense, Embedding,Bidirectional,GRU

from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
from sklearn.model_selection import KFold

import re
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
import nltk
import pickle

nltk.download('stopwords')
nltk.download('wordnet')
```

```
Using TensorFlow backend.

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[1]:
```
True
```

Pre-processing functions

In [0]:
```
def preprocess_reviews(reviews):
  # delete or replace special chars with whith spaces or End Of Senten
  # remove stopwords and lemmanize
  # turn review strings into word list


  reviews = [REPLACE_NO_SPACE.sub("", line.lower()) for line in review
  reviews = [REPLACE_WITH_EOS.sub(" EOS ", line) for line in reviews]
  reviews = [REPLACE_WITH_SPACE.sub(" ", line) for line in reviews]

  for i in range(len(reviews)):
    review = reviews[i]

    #lemmanize
    review = [lemmatizer.lemmatize(token) for token in review.split("
    review = [lemmatizer.lemmatize(token, "v") for token in review]

    #remove stopwords
    review = [word for word in review if not word in stop_words]
    review = " ".join(review)

    reviews[i] = review

  return reviews

def split_seq(s, y, SEQ_LEN):
    #splits seuence s into subsequences of SEQ_LEN in list s_extend
    #creates vector y_extend of the same length
  s_extend = sequence.pad_sequences([s[i:i+SEQ_LEN]  for i in range(0,
  y_extend = np.repeat(y,s_extend.shape[0]).reshape(-1,1)
  l = s_extend.shape[0]

  return(s_extend, y_extend, l)
```

In [0]:
```
stop_words = set(stopwords.words("english"))
lemmatizer = WordNetLemmatizer()

# creating regular expression objects
REPLACE_NO_SPACE = re.compile("(\;)|(\:)|(\')|(\,)|(\")|(\()|(\))|(\[)
REPLACE_WITH_EOS = re.compile("(\.)|(\!)|(\?)")
REPLACE_WITH_SPACE = re.compile("(<br\s*/><br\s*/>)|(\-)|(\/)")
```

In [4]:
```
#PATH = 'drive/My Drive/IMDB/'
FILENAME = "reviews_train.tsv"

FILEPATH =  FILENAME
FILEPATH
```

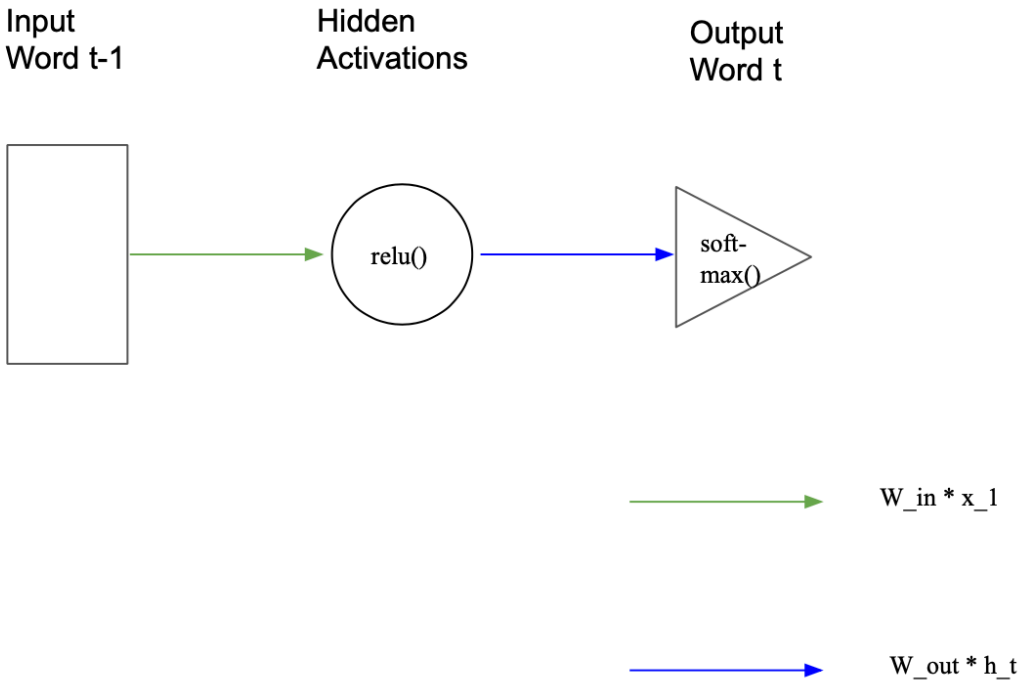
Out[4]:
```
'reviews_train.tsv'
```

Load and pre-process data

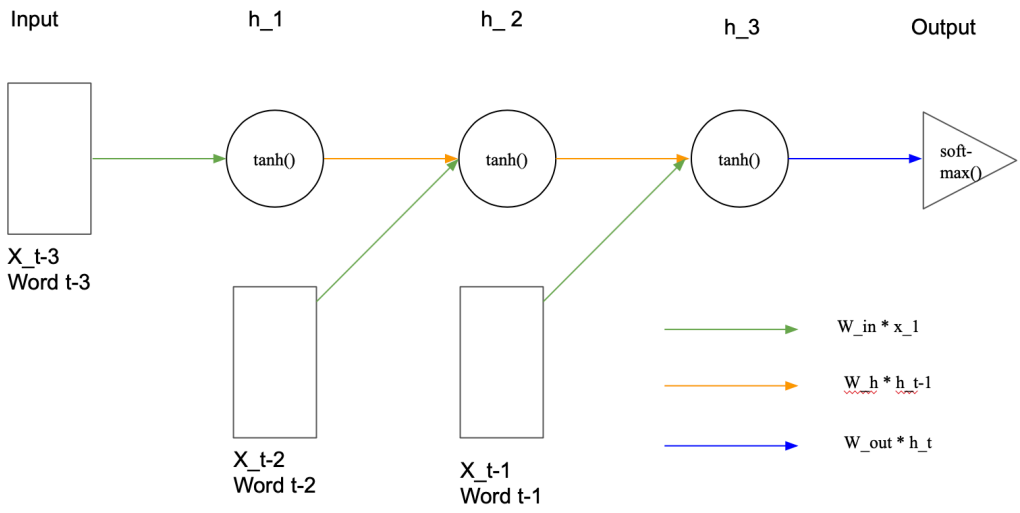
In [ ]:

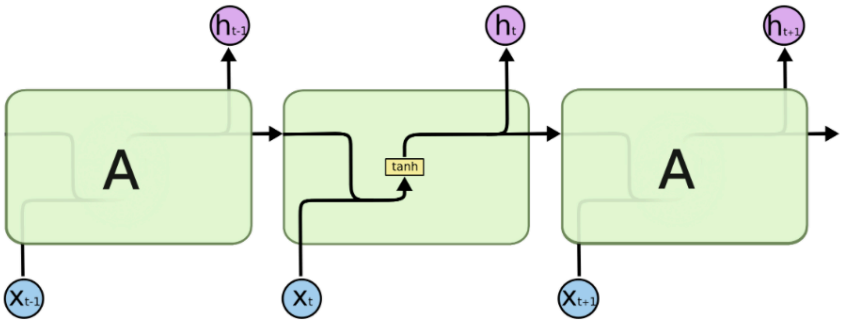
```
## Train

#PATH = 'drive/My Drive/data/'
TRAIN_FILE = 'reviews_train.tsv'
TRAIN_PATH =  TRAIN_FILE

MODEL_NAME = 'GRU_BIDIREC.h5'
TOKENIZER_NAME = 'tokenizer.pickle'

#input params
MAX_VOCAB = 10000
SEQ_LEN = 70

#read data
df = pd.read_table(TRAIN_PATH, header = None,names = ['id', 'y','text'

##training
reviews_trn = df['text']
y_trn = (df.y.values == 'pos').astype('int')

#preprocess and tokenize training data
X_trn = preprocess_reviews(reviews_trn)

tokenizer = Tokenizer(num_words = MAX_VOCAB)
tokenizer.fit_on_texts(X_trn)
# save Tokenizer
with open(TOKENIZER_NAME, 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)


X_trn = tokenizer.texts_to_sequences(X_trn)


#add observations by dividing reviews at SEQ_LEN in sub sequences
y_long = np.empty([0,1], int)
long_sequences  = np.empty([0,SEQ_LEN], int)


for s, y in zip(X_trn, y_trn):
  s, y, _ = split_seq(s, y, SEQ_LEN)
  y_long = np.vstack([y_long, y])
  long_sequences = np.vstack([long_sequences, s])


long_sequences.shape, y_long.shape
X_trn = long_sequences
y_trn = y_long
```

Simple Feedforward Network

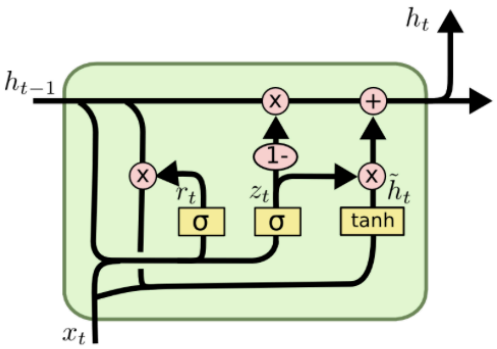


Simple Recurrent Neural Network





Extension: Gated Recurrent Unit



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Construct and Train Model

In [ ]:
```
#model params
embedding_size= 100
h_size = 32

#train params
batch_size = 128
epochs = 1

#construct model
model=Sequential()
model.add(Embedding(MAX_VOCAB, embedding_size, input_length=SEQ_LEN))
model.add(Bidirectional(GRU(h_size, input_shape = (SEQ_LEN,embedding_s
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['

#train model for one epoch
model.fit(X_trn,y_trn, batch_size=batch_size, epochs=epochs)

model.save(MODEL_NAME)  # creates a HDF5 File MODELNAME
```

Prediction

In [ ]:
```
##Make Predictions

PREDICT_FILE = "reviews_train.tsv"
PREDICT_PATH =  PREDICT_FILE
OUTPUT_FILE =  "result_file.csv"

MODEL_NAME = "GRU_BIDIREC.h5"
TOKENIZER_NAME = 'tokenizer.pickle'

df = pd.read_table(PREDICT_PATH, header = None,names = ['id', 'y','tex

#load tokenizer
with open(TOKENIZER_NAME, 'rb') as handle:
    tokenizer = pickle.load(handle)
#load model
model = load_model(MODEL_NAME)

reviews_val = df['text']

X_val = preprocess_reviews(reviews_val)
X_val = tokenizer.texts_to_sequences(X_val)

#prepare validation data
len_ar = []
X_long  = np.empty([0,SEQ_LEN], int)

for s in X_val:
    s, _, l = split_seq(s, 0, SEQ_LEN)
    X_long = np.vstack([X_long, s])
    len_ar.append(l)

len_ar = np.array(len_ar)


# make predictions
pred = model.predict(X_long)
predictions = []
idx = 0
for i in range(len_ar.shape[0]):
  y_hat = pred[idx: (idx+len_ar[i])].mean().round()
  predictions.append(y_hat)
  idx = idx + len_ar[i]

predictions = np.array(predictions)
output = pd.DataFrame(predictions, index = df['id'], columns = ['y_hat
output.replace([1.0,0.0], ['pos', 'neg'], inplace = True)
output.to_csv(OUTPUT_FILE, sep = '\t')
```

CV Results

In [15]:
```
cv_res = pd.read_csv('Cross_Validation_results.tsv', sep='\t')
cv_res.tail(3)
```

Out[15]:

	Unnamed: 0	Accuracy
10	Mean	0.889280
11	Std	0.006352
12	Median	0.890440