3.Model for Prediction

Import all necessary packages for execution

In [0]:

```
FN = 'predict'
```

In [0]:

```
import os
os.environ['THEANO_FLAGS'] = 'device=cpu,floatX=float32'
```

Importing Keras with Theano backend.

In [10]:

```
from keras import backend as K
import os

def set_keras_backend(backend):

    if K.backend() != backend:
        os.environ['KERAS_BACKEND'] = backend
        reload(K)
        assert K.backend() == backend

set_keras_backend("theano")
```

```
Using Theano backend.
```

In [11]:

```
import keras
keras.__version__
```

Out[11]:

```
'2.2.4'
```

We use the index of token generated in the previous process which lies in vocab_size limit. The words whose index lies outside vocab_size is replaced with special words which appears in same desciption.

In [0]:

```
FN0 = 'vocabulary-embedding'
```

In [0]:

```
FN1 = 'train'
```

Input data (X) is made from description with a length on maxlend. If the description is shorter than we description is padded with empty space in left side to make maxlend string. If the length is larger than maxlend then the string is clipped. The similiar process is done for labels(Y) with respect to maxlenh.

the model parameters should be identical with what used in training but notice that `maxlend` is flexible

In [0]:

```
maxlend=50 # 0 - if we dont want to use description at all
maxlenh=25
maxlen = maxlend + maxlenh
rnn_size = 512
rnn_layers = 3  # match FN1
batch_norm=False
```

the out of the first `activation_rnn_size` nodes from the top layer will be used for activation and the rest will be used to select predicted word

In [0]:

```
activation_rnn_size = 40 if maxlend else 0
```

In [0]:

```
seed=42
p_W, p_U, p_dense, p_emb, weight_decay = 0, 0, 0, 0, 0
optimizer = 'adam'
batch_size=64
```

In [0]:

```
nb_train_samples = 30000
nb_val_samples = 3000
```

In [18]:

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/au
th?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.goog
leusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&s
cope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20h
ttps%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.g
oogleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.goo
gleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

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

Read Word Embedding

The vocabulary embedding file produced in previous process is read.

In [0]:

```
import cPickle as pickle

with open('/content/drive/My Drive/Text_Summarizer/vocabulary-embedding.pkl', 'r
b') as fp:
    embedding, idx2word, word2idx, glove_idx2idx = pickle.load(fp)
vocab_size, embedding_size = embedding.shape
```

In [0]:

```
nb_unknown_words = 10
```

In [21]:

```
print 'dimension of embedding space for words',embedding_size
print 'vocabulary size', vocab_size, 'the last %d words can be used as place hol
ders for unknown/oov words'%nb_unknown_words
print 'total number of different words',len(idx2word), len(word2idx)
print 'number of words outside vocabulary which we can substitue using glove sim
ilarity', len(glove_idx2idx)
print 'number of words that will be regarded as unknonw(unk)/out-of-vocabulary(o
ov)',len(idx2word)-vocab_size-len(glove_idx2idx)
```

```
dimension of embedding space for words 100
vocabulary size 40000 the last 10 words can be used as place holders
for unknown/oov words
total number of different words 289382 289382
number of words outside vocabulary which we can substitue using glov
e similarity 48240
number of words that will be regarded as unknonw(unk)/out-of-vocabul
ary(oov) 201142
```

The unknown word are replaced with similiar word in vocab_size.

In [0]:

```
for i in range(nb_unknown_words):
    idx2word[vocab_size-1-i] = '<%d>'%i
```

The unknown words are suffixed with '^'.

In [0]:

```
for i in range(vocab_size-nb_unknown_words, len(idx2word)):
    idx2word[i] = idx2word[i]+'^'
```

Empty is represented using '_' and eos with '~'.

In [0]:

```
empty = 0
eos = 1
idx2word[empty] = '_'
idx2word[eos] = '~'
```

In [0]:

```
import numpy as np
from keras.preprocessing import sequence
from keras.utils import np_utils
import random, sys
```

In [0]:

```
def prt(label, x):
    print label+':',
    for w in x:
        print idx2word[w],
    print
```

Model

In [0]:

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation, Dropout, RepeatVector
from keras.layers.recurrent import LSTM
from keras.layers.embeddings import Embedding
from keras.regularizers import l2
from keras.layers.core import Lambda
import keras.backend as K
```

Seed weight initialization

In [0]:

```
random.seed(seed)
np.random.seed(seed)
```

In [0]:

```
regularizer = l2(weight_decay) if weight_decay else None
```

RNN Model

We use a stacked LSTM model similiar to our Training model.

In [30]:

```
rnn_model = Sequential()
rnn_model.add(Embedding(vocab_size, embedding_size,
                        input_length=maxlen,
                        W_regularizer=regularizer, dropout=p_emb, weights=[embed
ding], mask_zero=True,
                        name='embedding_1'))
for i in range(rnn_layers):
    lstm = LSTM(rnn_size, return_sequences=True, ,
                W_regularizer=regularizer, U_regularizer=regularizer,
                b_regularizer=regularizer, dropout_W=p_W, dropout_U=p_U,
                name='lstm_%d'%(i+1)
                  )
    rnn_model.add(lstm)
    rnn_model.add(Dropout(p_dense, name='dropout_%d'%(i+1)))
```

```
/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:5: User
Warning: The `dropout` argument is no longer support in `Embedding`.
You can apply a `keras.layers.SpatialDropout1D` layer right after th
e `Embedding` layer to get the same behavior.
  """
/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:5: User
Warning: Update your `Embedding` call to the Keras 2 API: `Embedding
(40000, 100, name="embedding_1", weights=[array([[-..., embeddings_r
egularizer=None, mask_zero=True, input_length=75)`
  """
/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:10: Use
rWarning: Update your `LSTM` call to the Keras 2 API: `LSTM(512, rec
urrent_regularizer=None, recurrent_dropout=0, name="lstm_1", bias_re
gularizer=None, dropout=0, kernel_regularizer=None, return_sequences
=True)`
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:10: Use
rWarning: Update your `LSTM` call to the Keras 2 API: `LSTM(512, rec
urrent_regularizer=None, recurrent_dropout=0, name="lstm_2", bias_re
gularizer=None, dropout=0, kernel_regularizer=None, return_sequences
=True)`
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:10: Use
rWarning: Update your `LSTM` call to the Keras 2 API: `LSTM(512, rec
urrent_regularizer=None, recurrent_dropout=0, name="lstm_3", bias_re
gularizer=None, dropout=0, kernel_regularizer=None, return_sequences
=True)`
  # Remove the CWD from sys.path while we load stuff.
```

Load

We use the bottom weight from trained model and use th top weigth for later purposes.

In [0]:

```
import h5py
def str_shape(x):
    return 'x'.join(map(str,x.shape))

def inspect_model(model):
    print model.name
    for i,l in enumerate(model.layers):
        print i, 'cls=%s name=%s'%(type(l).__name__, l.name)
        weights = l.get_weights()
        for weight in weights:
            print str_shape(weight),
        print

def load_weights(model, filepath):
    """Modified version of keras load_weights that loads as much as it can
    if there is a mismatch between file and model. It returns the weights
    of the first layer in which the mismatch has happened
    """
    print 'Loading', filepath, 'to', model.name
    flattened_layers = model.layers
    with h5py.File(filepath, mode='r') as f:
        # new file format
        layer_names = [n.decode('utf8') for n in f.attrs['layer_names']]

        # we batch weight value assignments in a single backend call
        # which provides a speedup in TensorFlow.
        weight_value_tuples = []
        for name in layer_names:
            print name
            g = f[name]
            weight_names = [n.decode('utf8') for n in g.attrs['weight_names']]
            if len(weight_names):
                weight_values = [g[weight_name] for weight_name in weight_names]
                try:
                    layer = model.get_layer(name=name)
                except:
                    layer = None
                if not layer:
                    print 'failed to find layer', name, 'in model'
                    print 'weights', ' '.join(str_shape(w) for w in weight_value
s)
                    print 'stopping to load all other layers'
                    weight_values = [np.array(w) for w in weight_values]
                    break
                symbolic_weights = layer.trainable_weights + layer.non_trainable
_weights
                weight_value_tuples += zip(symbolic_weights, weight_values)
                weight_values = None
        K.batch_set_value(weight_value_tuples)
    return weight_values
```

In [32]:

```
weights = load_weights(rnn_model, '/content/drive/My Drive/Text_Summarizer/%s.hdf5'%FN1)
```

```
Loading /content/drive/My Drive/Text_Summarizer/train.hdf5 to sequen
tial_1
embedding_1
lstm_1
dropout_1
lstm_2
dropout_2
lstm_3
dropout_3
simplecontext_1
time_distributed_1
failed to find layer time_distributed_1 in model
weights 944x40000 40000
stopping to load all other layers
```

In [33]:

```
[w.shape for w in weights]
```

Out[33]:

```
[(944, 40000), (40000,)]
```

Headline Model

A special layer that reduces the input just to its headline part (second half). For each word in this part it concatenate the output of the previous layer (RNN) with a weighted average of the outputs of the description part. In this only the last `rnn_size - activation_rnn_size` are used from each output. The first `activation_rnn_size` output is used to computer the weights for the averaging.

In [0]:

```
context_weight = K.variable(1.)
head_weight = K.variable(1.)
cross_weight = K.variable(0.)
import tensorflow as tf

def simple_context(X, mask, n=activation_rnn_size, maxlend=maxlend, maxlenh=maxl
enh):
    desc, head = X[:,:maxlend], X[:,maxlend:]
    head_activations, head_words = head[:,:,:n], head[:,:,n:]
    desc_activations, desc_words = desc[:,:,:n], desc[:,:,n:]
    activation_energies = K.batch_dot(head_activations, desc_activations, axes=(
2,2))
    assert mask.ndim == 2
    activation_energies = K.switch(mask[:, None, :maxlend], activation_energies,
 -1e20)
    activation_energies = K.reshape(activation_energies,(-1,maxlend))
    activation_weights = K.softmax(activation_energies)
    activation_weights = K.reshape(activation_weights,(-1,maxlenh,maxlend))
    desc_avg_word = K.batch_dot(activation_weights, desc_words, axes=(2,1))
    return K.concatenate((context_weight*desc_avg_word, head_weight*head_words))


class SimpleContext(Lambda):
    def __init__(self,**kwargs):
        super(SimpleContext, self).__init__(simple_context,**kwargs)
        self.supports_masking = True

    def compute_mask(self, input, input_mask=None):
        return input_mask[:, maxlend:]

    def get_output_shape_for(self, input_shape):
        nb_samples = input_shape[0]
        n = 2*(rnn_size - activation_rnn_size)
        return (nb_samples, maxlenh, n)
```

In [35]:

```
model = Sequential()
model.add(rnn_model)

if activation_rnn_size:
    model.add(SimpleContext(name='simplecontext_1'))
```

```
/usr/local/lib/python2.7/dist-packages/keras/layers/core.py:665: Use
rWarning: `output_shape` argument not specified for layer simplecont
ext_1 and cannot be automatically inferred with the Theano backend.
Defaulting to output shape `(None, 75, 512)` (same as input shape).
If the expected output shape is different, specify it via the `outpu
t_shape` argument.
  .format(self.name, input_shape))
```

In [0]:

```
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

In [37]:

```
n = 2*(rnn_size - activation_rnn_size)
n
```

Out[37]:

944

perform the top dense of the trained model in numpy using softmax

In [0]:

```
def output2probs(output):
    output = np.dot(output, weights[0]) + weights[1]
    output -= output.max()
    output = np.exp(output)
    output /= output.sum()
    return output
```

In [0]:

```
def output2probs1(output):
    output0 = np.dot(output[:n//2], weights[0][:n//2,:])
    output1 = np.dot(output[n//2:], weights[0][n//2:,:])
    output = output0 + output1 # + output0 * output1
    output += weights[1]
    output -= output.max()
    output = np.exp(output)
    output /= output.sum()
    return output
```

Test

```
Left pad a description to maxlend and then add eos.
The eos is the input to predicting the first word in the headline
```

In [0]:

```
def lpadd(x, maxlend=maxlend, eos=eos):
    assert maxlend >= 0
    if maxlend == 0:
        return [eos]
    n = len(x)
    if n > maxlend:
        x = x[-maxlend:]
        n = maxlend
    return [empty]*(maxlend-n) + x + [eos]
```

pad from right (post) so the first maxlend will be description followed by headline

In [0]:

```
samples = [lpadd([3]*26)]
data = sequence.pad_sequences(samples, maxlen=maxlen, value=empty, padding='pos
t', truncating='post')
```

In [42]:

```
np.all(data[:,maxlend] == eos)
```

Out[42]:

```
True
```

In [43]:

```
data.shape,map(len, samples)
```

Out[43]:

```
((1, 75), [51])
```

In [44]:

```
probs = model.predict(data, verbose=0, batch_size=1)
probs.shape
```

```
WARNING (theano.tensor.blas): We did not find a dynamic library in t
he library_dir of the library we use for blas. If you use ATLAS, mak
e sure to compile it with dynamics library.
```

Out[44]:

```
(1, 25, 944)
```

Sample generation

Return k samples (beams) and their NLL scores, each sample is a sequence of labels, all samples starts with an `empty` label and end with `eos` or truncated to length of `maxsample`. You need to supply `predict` which returns the label probability of each sample. `use_unk` allow usage of out-of-vocabulary label in samples The function samples at most n different elements according to their energy each element only once. For every possible live sample calculate probability for every possible label. The total score for every sample is sum of -log of word probability. Find the best (lowest) scores we have from all possible dead samples andall live samples and all possible new words added. The words avoided is added to a list.

In [0]:

```
def beamsearch(predict, start=[empty]*maxlend + [eos], avoid=None, avoid_score=1
,
               k=1, maxsample=maxlen, use_unk=True, oov=vocab_size-1, empty=empt
y, eos=eos, temperature=1.0):

    def sample(energy, n, temperature=temperature):
        n = min(n,len(energy))
        prb = np.exp(-np.array(energy) / temperature )
        res = []
        for i in xrange(n):
            z = np.sum(prb)
            r = np.argmax(np.random.multinomial(1, prb/z, 1))
            res.append(r)
            prb[r] = 0.
        return res

    dead_samples = []
    dead_scores = []
    live_samples = [list(start)]
    live_scores = [0]

    while live_samples:
        probs = predict(live_samples, empty=empty)
        assert vocab_size == probs.shape[1]
        cand_scores = np.array(live_scores)[:,None] - np.log(probs)
        cand_scores[:,empty] = 1e20
        if not use_unk and oov is not None:
            cand_scores[:,oov] = 1e20
        if avoid:
            for a in avoid:
                for i, s in enumerate(live_samples):
                    n = len(s) - len(start)
                    if n < len(a):
                        cand_scores[i,a[n]] += avoid_score
        live_scores = list(cand_scores.flatten())
        scores = dead_scores + live_scores
        ranks = sample(scores, k)
        n = len(dead_scores)
        dead_scores = [dead_scores[r] for r in ranks if r < n]
        dead_samples = [dead_samples[r] for r in ranks if r < n]

        live_scores = [live_scores[r-n] for r in ranks if r >= n]
        live_samples = [live_samples[(r-n)//vocab_size]+[(r-n)%vocab_size] for r
 in ranks if r >= n]
        def is_zombie(s):
            return s[-1] == eos or len(s) > maxsample
        dead_scores += [c for s, c in zip(live_samples, live_scores) if is_zombi
e(s)]
        dead_samples += [s for s in live_samples if is_zombie(s)]
        live_scores = [c for s, c in zip(live_samples, live_scores) if not is_zo
mbie(s)]
        live_samples = [s for s in live_samples if not is_zombie(s)]

    return dead_samples, dead_scores
```

In [0]:

For every sample, calculate probability for every possible label by supplying RNN model and maxlen - the length of sequences.

In [0]:

```
def keras_rnn_predict(samples, empty=empty, model=model, maxlen=maxlen):
    sample_lengths = map(len, samples)
    assert all(l > maxlend for l in sample_lengths)
    assert all(l[maxlend] == eos for l in samples)
    data = sequence.pad_sequences(samples, maxlen=maxlen, value=empty, padding=
'post', truncating='post')
    probs = model.predict(data, verbose=0, batch_size=batch_size)
    return np.array([output2probs(prob[sample_length-maxlend-1]) for prob, sampl
e_length in zip(probs, sample_lengths)])
```

Convert list of word indexes that may contain words outside vocab_size to words inside. If a word is outside, try first to use glove_idx2idx to find a similar word inside. If none exist then replace all accurancies of the same unknown word with <0>, <1>.

In [0]:

```
def vocab_fold(xs):
    xs = [x if x < vocab_size-nb_unknown_words else glove_idx2idx.get(x,x) for x
 in xs]
    outside = sorted([x for x in xs if x >= vocab_size-nb_unknown_words])
    outside = dict((x,vocab_size-1-min(i, nb_unknown_words-1)) for i, x in enume
rate(outside))
    xs = [outside.get(x,x) for x in xs]
    return xs
```

In [0]:

```
def vocab_unfold(desc,xs):
    unfold = {}
    for i, unfold_idx in enumerate(desc):
        fold_idx = xs[i]
        if fold_idx >= vocab_size-nb_unknown_words:
            unfold[fold_idx] = unfold_idx
    return [unfold.get(x,x) for x in xs]
```

In [50]:

```
!pip install python-levenshtein
```

```
Collecting python-levenshtein
  Downloading https://files.pythonhosted.org/packages/42/a9/d1785c85
ebf9b7dfacd08938dd028209c34a0ea3b1bcdb895208bd40a67d/python-Levensht
ein-0.12.0.tar.gz (48kB)
    100% |████████████████████████████████| 51kB 3.5MB/s
Requirement already satisfied: setuptools in /usr/local/lib/python2.
7/dist-packages (from python-levenshtein) (40.9.0)
Building wheels for collected packages: python-levenshtein
  Building wheel for python-levenshtein (setup.py) ... done
  Stored in directory: /root/.cache/pip/wheels/de/c2/93/660fd5f75590
49268ad2dc6d81c4e39e9e36518766eaf7e342
Successfully built python-levenshtein
Installing collected packages: python-levenshtein
```

In [0]:

```
import sys
import Levenshtein

def gensamples(X=None, X_test=None, Y_test=None, avoid=None, avoid_score=1, skip
s=2, k=10, batch_size=batch_size, short=True, temperature=1., use_unk=True):
    if X is None or isinstance(X,int):
        if X is None:
            i = random.randint(0,len(X_test)-1)
        else:
            i = X
        print 'HEAD %d:'%i,' '.join(idx2word[w] for w in Y_test[i])
        print 'DESC:',' '.join(idx2word[w] for w in X_test[i])
        sys.stdout.flush()
        x = X_test[i]
    else:
        x = [word2idx[w.rstrip('^')] for w in X.split()]

    if avoid:
        if isinstance(avoid,str) or isinstance(avoid[0], int):
            avoid = [avoid]
        avoid = [a.split() if isinstance(a,str) else a for a in avoid]
        avoid = [vocab_fold([w if isinstance(w,int) else word2idx[w] for w in a
])
                 for a in avoid]

    print 'HEADS:'
    samples = []
    if maxlend == 0:
        skips = [0]
    else:
        skips = range(min(maxlend,len(x)), max(maxlend,len(x)), abs(maxlend - le
n(x)) // skips + 1)
    for s in skips:
        start = lpadd(x[:s])
        fold_start = vocab_fold(start)
        sample, score = beamsearch(predict=keras_rnn_predict, start=fold_start,
avoid=avoid, avoid_score=avoid_score,
                                   k=k, temperature=temperature, use_unk=use_unk
)
        assert all(s[maxlend] == eos for s in sample)
        samples += [(s,start,scr) for s,scr in zip(sample,score)]

    samples.sort(key=lambda x: x[-1])
    codes = []
    for sample, start, score in samples:
        code = ''
        words = []
        sample = vocab_unfold(start, sample)[len(start):]
        for w in sample:
            if w == eos:
                break
            words.append(idx2word[w])
            code += chr(w//(256*256)) + chr((w//256)%256) + chr(w%256)
        if short:
            distance = min([100] + [-Levenshtein.jaro(code,c) for c in codes])
            if distance > -0.6:
                print score, ' '.join(words)
```

```
        codes.append(code)
    return samples
```

In [0]:

```
seed = 8
random.seed(seed)
np.random.seed(seed)
```

In [0]:

```
X = "starting from march 1, the island province will implement strict market acc
ess control on all incoming livestock and animal products to prevent the possibl
e spread of epidemic diseases "
Y = "island to curb spread of diseases"
```

In [54]:

```
samples = gensamples(X=X, skips=2, batch_size=batch_size, k=10, temperature=1.)
```

```
HEADS:
7.193180128931999 $35,000 of of supplies of common Guam asthma
```

In [0]:

```
headline = samples[0][0][len(samples[0][1]):]
```

In [0]:

```
' '.join(idx2word[w] for w in headline)
```

Weights

Activation for every head word and every desc word. Don't use description for masked out words. Compute weight for every description word.

In [0]:

```
def wsimple_context(X, mask, n=activation_rnn_size, maxlend=maxlend, maxlenh=max
lenh):
    desc, head = X[:,:maxlend], X[:,maxlend:]
    head_activations, head_words = head[:,:,:n], head[:,:,n:]
    desc_activations, desc_words = desc[:,:,:n], desc[:,:,n:]
    activation_energies = K.batch_dot(head_activations, desc_activations, axes=(
2,2))
    assert mask.ndim == 2
    activation_energies = K.switch(mask[:, None, :maxlend], activation_energies,
 -1e20)
    activation_energies = K.reshape(activation_energies,(-1,maxlend))
    activation_weights = K.softmax(activation_energies)
    activation_weights = K.reshape(activation_weights,(-1,maxlenh,maxlend))

    return activation_weights


class WSimpleContext(Lambda):
    def __init__(self):
        super(WSimpleContext, self).__init__(wsimple_context)
        self.supports_masking = True

    def compute_mask(self, input, input_mask=None):
        return input_mask[:, maxlend:]

    def get_output_shape_for(self, input_shape):
        nb_samples = input_shape[0]
        n = 2*(rnn_size - activation_rnn_size)
        return (nb_samples, maxlenh, n)
```

In [0]:

```
wmodel = Sequential()
wmodel.add(rnn_model)
```

In [86]:

```
wmodel.add(WSimpleContext())
```

```
/usr/local/lib/python2.7/dist-packages/keras/layers/core.py:665: Use
rWarning: `output_shape` argument not specified for layer w_simple_c
ontext_2 and cannot be automatically inferred with the Theano backen
d. Defaulting to output shape `(None, 75, 512)` (same as input shap
e). If the expected output shape is different, specify it via the `o
utput_shape` argument.
  .format(self.name, input_shape))
```

In [0]:

```
wmodel.compile(loss='categorical_crossentropy', optimizer=optimizer)
```

Test

In [0]:

```
seed = 8
random.seed(seed)
np.random.seed(seed)
```

In [0]:

```
context_weight.set_value(np.float32(1.))
head_weight.set_value(np.float32(1.))
```

In [0]:

```
X = "VETERANS saluted first ever breakfast club for ex-soldiers which won over h
earts minds and   The Breakfast Club for HM Forces Veterans met at the Postal Or
der in Foregate Street at 10am on Saturday  The club is designed to allow vetera
ns a place to meet  eat and drink giving hunger and loneliness their marching or
ders  Father-of-two Dave Carney aged 43 of Merrimans Hill Worcester set up the c
lub after being inspired by other similar clubs across the country  He said: As
 you can see from the picture we had a good response Five out of the 10 that att
ended said they saw the article in the newspaper and turned up  We even had an o
ld chap travel from Droitwich and he was late on parade by three hours  Its gene
rated a lot of interest and I estimate from other veterans who saw the article t
hat next months meeting will attract about 20 people Onwards and upwards  He sai
d the management at the pub had been extremely hospitable to them  Mr Carney sai
d: They bent over backwards for us They really looked after us well That is the
 best choice of venue I could have made They even put reserved for the armed for
ces Promoted stories The reserve veteran with the Royal Engineers wanted to go t
o a breakfast club but found the nearest ones were in Bromsgrove and Gloucester
 so he decided to set up his own closer to home  He was influenced by Derek Hard
man who set up a breakfast club for veterans in Hull and Andy Wilson who set one
 up in Newcastle He said the idea has snowballed and there were now 70 similar c
lubs across the country and even some in Germany  Mr Carney said with many Royal
 British Legion clubs closing he wanted veterans and serving personnel to feel t
hey had somewhere they could go for good grub beer and banter to recapture the c
omradery of being in the forces  The Postal Order was chosen because of its cent
ral location and its proximity to the railway station and hotels and reasonably
 priced food and drink  The management of the pub have even given the veterans a
 designated area within the pub  Share article  The next meeting is at the Posta
l Order on Saturday October 3 at 10am  The breakfast club meets on the first Sat
urday of each month for those who want to attend in future"
```

In [100]:

```
samples = gensamples(X, skips=2, batch_size=batch_size, k=10, temperature=1.)
```

```
HEADS:


ValueErrorTraceback (most recent call last)
<ipython-input-100-29be5167ca40> in <module>()
----> 1 samples = gensamples(X, skips=2, batch_size=batch_size, k=10
, temperature=1.)

<ipython-input-51-5d59500008dd> in gensamples(X, X_test, Y_test, avo
id, avoid_score, skips, k, batch_size, short, temperature, use_unk)
     33         fold_start = vocab_fold(start)
     34         sample, score = beamsearch(predict=keras_rnn_predic
t, start=fold_start, avoid=avoid, avoid_score=avoid_score,
---> 35                                    k=k, temperature=temperat
ure, use_unk=use_unk)
     36         assert all(s[maxlend] == eos for s in sample)
     37         samples += [(s,start,scr) for s,scr in zip(sample,sc
ore)]

<ipython-input-45-f6b5d5ab7654> in beamsearch(predict, start, avoid,
 avoid_score, k, maxsample, use_unk, oov, empty, eos, temperature)
     25     while live_samples:
     26         # for every possible live sample calc prob for every
 possible label
---> 27         probs = predict(live_samples, empty=empty)
     28         assert vocab_size == probs.shape[1]
     29

<ipython-input-47-eec4bb675fdc> in keras_rnn_predict(samples, empty,
 model, maxlen)
      9     data = sequence.pad_sequences(samples, maxlen=maxlen, va
lue=empty, padding='post', truncating='post')
     10     probs = model.predict(data, verbose=0, batch_size=batch_
size)
---> 11     return np.array([output2probs(prob[sample_length-maxlend
-1]) for prob, sample_length in zip(probs, sample_lengths)])

<ipython-input-38-b6f20dccbce5> in output2probs(output)
      1 def output2probs(output):
----> 2     output = np.dot(output, weights[0]) + weights[1]
      3     output -= output.max()
      4     output = np.exp(output)
      5     output /= output.sum()

ValueError: shapes (944,) and (25,50) not aligned: 944 (dim 0) != 25
 (dim 0)
```

In [0]:

```
sample = samples[0][0]
```

In [70]:

```
' '.join([idx2word[w] for w in sample])
```

Out[70]:

```
u'breakfast club for <0>^ which won over hearts, minds and <1>^ The
Worcester Breakfast Club for HM Forces Veterans met at the Postal Or
der in Brattle Street at 10am on Saturday. The club is designed to a
llow veterans a place to meet, <2>^ eat and drink, giving hunger and
loneliness ~ <0>^ ~'
```

X = "What have you been listening to this year ? If you want to find out using cold , hard evidence , then Spotify 's new Year in Music tool will tell you ."

In [71]:

```
data = sequence.pad_sequences([sample], maxlen=maxlen, value=empty, padding='pos
t', truncating='post')
data.shape
```

Out[71]:

```
(1, 75)
```

In [72]:

```
weights = wmodel.predict(data, verbose=0, batch_size=1)
weights.shape
```

Out[72]:

```
(1, 25, 50)
```

In [73]:

```
startd = np.where(data[0,:] != empty)[0][0]
lenh = np.where(data[0,maxlend+1:] == eos)[0][0]
startd, lenh
```

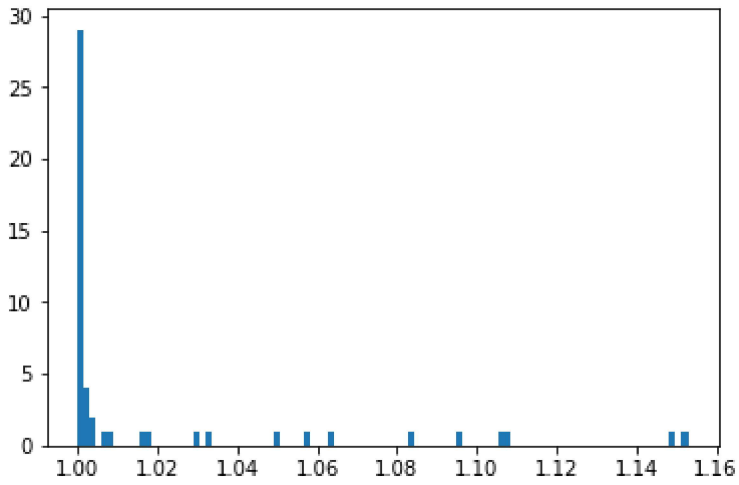
Out[73]:

```
(0, 1)
```

Plotting histogram of weight

In [74]:

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.hist(np.array(weights[0,:lenh,startd:].flatten()+1), bins=100);
```



In [0]:

```
import numpy as np
from IPython.core.display import display, HTML

def heat(sample,weights,dark=0.3):
    weights = (weights - weights.min())/(weights.max() - weights.min() + 1e-4)
    html = ''
    fmt = ' <span style="background-color: #{0:x}{0:x}ff">{1}</span>'
    for t,w in zip(sample,weights):
        c = int(256*((1.-dark)*(1.-w)+dark))
        html += fmt.format(c,idx2word[t])
    display(HTML(html))
```

In [76]:

```
heat(sample, weights[0,-1])
```

breakfast club for <0>^ which won over hearts, minds and <1>^ The Worcester Breakfast Club for HM Forces Veterans met at the Postal Order in Brattle Street at 10am on Saturday. The club is designed to allow veterans a place to meet, <2>^ eat and drink, giving hunger and loneliness

In [0]:

```
import pandas as pd
import seaborn as sns
```

In [0]:

```
columns = [idx2word[data[0,i]] for i in range(startd,maxlend)]
rows = [idx2word[data[0,i]] for i in range(maxlend+1,maxlend+lenh+1)]
```

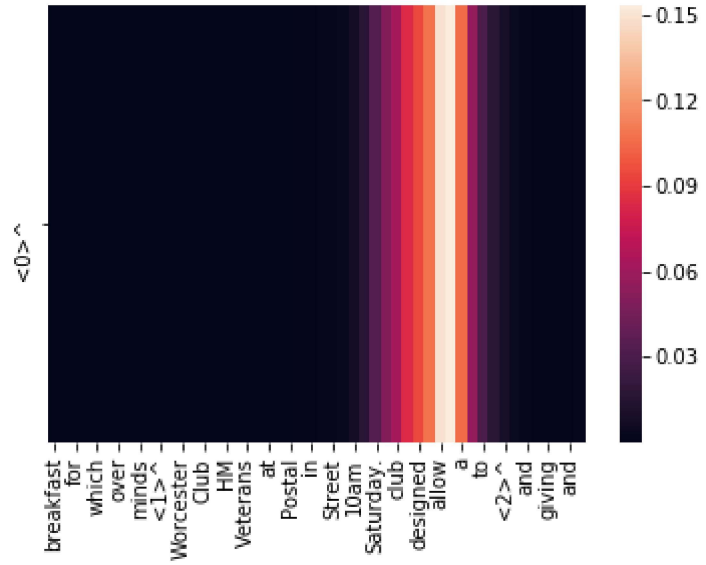
In [0]:

```
df = pd.DataFrame(weights[0,:lenh,startd:],columns=columns,index=rows)
```

Plotting heatmap of weights

In [80]:

```
sns.heatmap(df);
```



In [0]:

```
from keras.utils.vis_utils import plot_model
```

In [0]:

```
plot_model(wmodel, to_file='/content/drive/My Drive/Text_Summarizer/decode.png',
 show_shapes=True, show_layer_names=True)
```

In [0]:

```
plot_model(rnn_model, to_file='/content/drive/My Drive/Text_Summarizer/decode1.p
ng', show_shapes=True, show_layer_names=True)
```