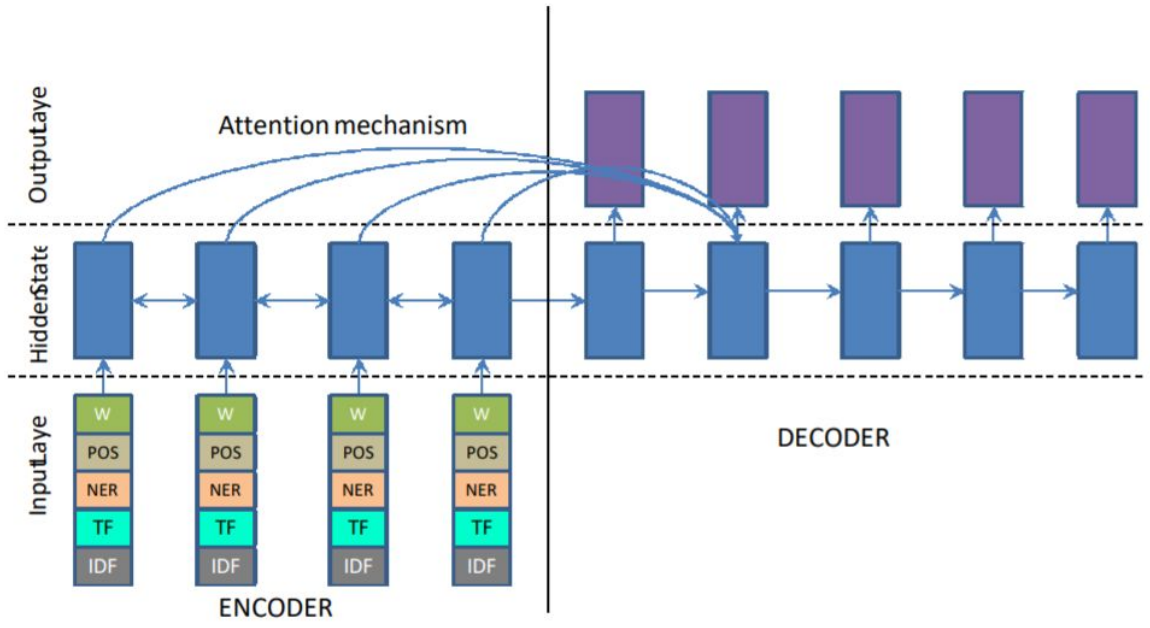
Abstractive Text Summarization of Legal Judgements: Seq2Seq Model

Using Bahdanau Attention Model and SpaCy



Courtesy: https://arxiv.org/pdf/1602.06023.pdf

A. Introduction

This section presents in a suited detail, my project for abstractive text summarization which I have modelled using a, encoder-decoder Seq2Seq framework of Sequential Models, integrating an Attention Model to account for the extended length of an average legal judgement.

B. Preface

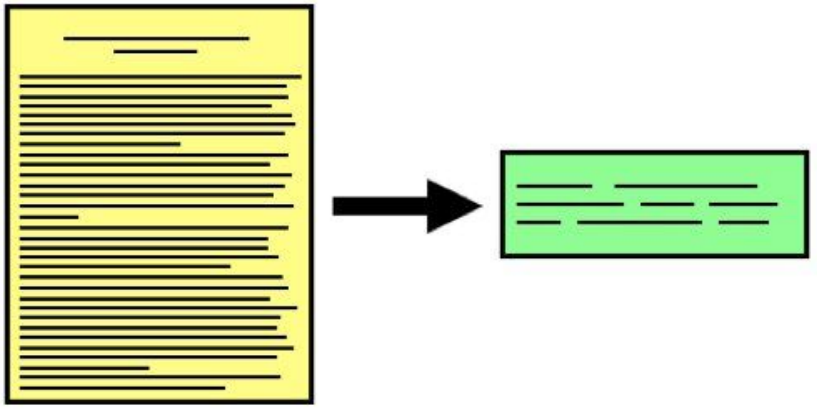
My first assignment was to implement two models, first machine learning and then using a deep-learning approach for a problem statement to categorize e-commerce products into categories provided by Google for marketing campaigns, taking an input of the product description. I accomplished the task with 89% accuracy for the Machine Learning approach using the SVM Classifier, and a 93% accuracy for the Deep Learning approach which made use of dense neural networks using Tensorflow.

I was then handed a project to summarize legal judgements using the technique of abstraction.

C. Outline

Section 1 covers the preprocessing of legal text which involved feature creation which contained information about the corpus which helped the training model recognize the nuances of the legal domain better. This information included Part of Speech Tagging, Named-Entity Recognition and the TF-IDF scores of the tokens in the corpus.

Section 2 covers building a Seq2Seq Model which uses LSTM units (Long Short Term Memory) in an encoder-decoder fashion, integrated with the Attention Model. I have trained the model to generate customer-review summaries from an Amazon Reviews dataset from Kaggle.



Section I

Preprocessing of Data

I was given data collected from the source website along the metadata of each judgement. Initial steps pertain to achieving a clean corpus and a structured metadata. Each document has a unique id denoted by '_id'

```
# importing judgements from json
with open('response.json',encoding="utf8") as f:
    mydata = json.load(f)

# dictionary to store documents
document = {}

for obj in mydata['hits']:
    _id = obj['_id']
    doc = _id
    content = obj['_source']['content']
    html_soup = BeautifulSoup(content,'html.parser')
    flag=1
    for para in html_soup.find_all('p'):
        if(para.text!="@JUDGMENTTAG-JUDGMENT"):
            if(flag):
                flag=0
                continue
            doc+= "<para>\n" + para.get_text(strip=True)
    document[_id] = doc

# List of IDs for Lookup

ids = []
rev_ids = {}
for obj in mydata['hits']:
    ids.append(obj['_id'])
for i in range(len(ids)):
    rev_ids[ids[i]] = i
print(ids)
print(rev_ids)
```

```
# Extracting Corpus

corpus = {}

for i in document:
    text = document[i]
    text=re.sub("\xa0",' ',text)
    text=re.sub("\xad",' ',text)
    text=re.sub('\r\n',' ',text)
    text = re.sub("<para>\n\d*\.*",'\n',text)
    text = re.sub("\n",' ',text)
    corpus[i] = text

# Extracting Metadata
metadata = {}

for obj in mydata['hits']:
    _id = obj['_id']
    src = obj['_source']
    court = src['court']
    case_no = src['case_no']
    coram = []
    counsel = []
    for crm in src['coram_nested']:
        coram.append(crm['value'])
    for cns in src['counsel_nested']:
        counsel.append(cns['value'])
    ck_citation = src['ckcitation']
    date = src['date']
    acts = src['acts']
    abbr = src['abbr']
    try:
        result = src['result']
    except:
        result = ""

    appellant = src['appellant'].split('@')
    respondent = src['respondent'].split('@')
    curdata = {
        "court": court,
        "case_no": case_no,
        "coram": coram,
        "counsel": counsel,
        "ck_citation": ck_citation,
        "date": date,
        "acts": acts,
        "abbr": abbr,
        "result": result,
        "appellant": appellant,
        "respondent": respondent
    }
    metadata[_id] = curdata
```

```
# example metadata
# 'court' : Supreme Court Of India
# 'case_no' : Civil Appeal No. 219, 222, 223 Of 2019
# 'coram' : ['Arun Mishra, J', 'Vineet Saran, J']
# 'counsel' : ['nishe rajen shonker', 'jogy scaria']
# 'ck_citation' : (2019) 01 SC CK 0031
# 'date' : 09-01-2019
# 'acts' : [{'rules': 'Rule 47, 47(1)(a), 47(1)(d), 47(1)(g), 47(a), 47(d), 48, 92,
92(1), 100, 104A, 106, 126', 'title': 'Central Motor Vehicles Rules, 1989'}, {'rules':
'Rule 93, 93(1), 93(2), 93(4), 93(6), 96, 103, 261', 'title': 'Kerala Motor Vehicle
Rules, 1989'}, {'section': 'Section 27, 28, 52', 'title': 'Motor Vehicles Act, 1988'},
{'section': 'Section 32', 'title': 'Motor Vehicles Act, 1939'}, {'section': 'Section
52, 52(1), 52(1)(a), 52(1)(b), 52(2), 52(3), 52(4), 52(5), 52(6)', 'title': 'Motor
Vehicles (Amendment) Act, 2000'}]
# 'abbr' : SC
# 'result' : Allowed
# 'appellant' : ['Regional Transport Officer & Ors. Etc']
# 'respondent' : ['K. Jayachandra & Anr. Etc']

# Example Corpus
"""
2363439  Leave granted.  This appeal is filed against the final judgment and order
dated 25.10.2016 passed by the High Court of Judicature for Rajasthan, Bench at Jaipur
in D.B. Income Tax Appeal No. 43 of 2002 whereby the Division Bench of the High Court
dismissed the appeal filed by the appellant herein and affirmed the order dated
24.05.2001 passed by the Income Tax Appellate Tribunal (ITAT), Jaipur Bench, Jaipur in
I.T.S.S.A. No.29/JP/2000.  A few facts need mention infra for the disposal of the
appeal.  This appeal filed by the Revenue arises out of the income tax proceedings
initiated against the respondent(assessee) on the basis of a search operation which
was carried out by the Income Tax Department in assessee's premises on 04.09.1997.
This gave rise to initiation of assessment proceedings for the block period from
01.04.1987 to 04.09.1997 (Assessment Years 1987 88 to 1996 97 and 1997 98 up to
04.09.1997) against the assessee to determine their tax liability as a result of
search operations carried in their premises. The matter, out of the block assessment
proceedings, reached to the Income Tax Appellate Tribunal at the instance of the
respondent against the order of the assessing authorities.  The Tribunal (ITAT),
however, decided the various issues arising in the case in favour of the
respondent(assessee) by allowing the respondent's appeal, which gave rise to filing of
the appeal by the Revenue before the High Court under Section 260A of the Income Tax
Act, 1961 (hereinafter referred to as "the Act").  The High Court by impugned judgment
dismissed the Revenue's appeal, which gave rise to filing of this appeal by way of
special leave by the Revenue in this Court.  Having heard the learned counsel for the
parties and on perusal of the record of the case, we are constrained to allow the
appeal and remand the case to the High Court for deciding the appeal afresh on merits
in accordance with law.... with law.  Before parting, we may observe that we have not
expressed any opinion on the merits of the case having formed an opinion to remand the
case to the High Court in the light of our foregoing discussion. The High Court will,
therefore, decide the appeal in accordance with law uninfluenced by any observations
made by this Court.
"""
```

Next, for POS Tagging and Named-Entitiy Recognition, I took use of the SpaCy framework available for Python to extract general named entities. To refine these entities to be domain specific, I created a custom pipeline to add to the existing SpaCy framework for better annotation of the judgments.

Classes				#	%
f	1	PER	Person	1,747	3.26
f	2	RR	Judge	1,519	2.83
f	3	AN	Lawyer	111	0.21
c	**1**	**PER**	**Person**	**3,377**	**6.30**
f	4	LD	Country	1,429	2.66
f	5	ST	City	705	1.31
f	6	STR	Street	136	0.25
f	7	LDS	Landscape	198	0.37
c	**2**	**LOC**	**Location**	**2,468**	**4.60**
f	8	ORG	Organization	1,166	2.17
f	9	UN	Company	1,058	1.97
f	10	INN	Institution	2,196	4.09
f	11	GRT	Court	3,212	5.99
f	12	MRK	Brand	283	0.53
c	**3**	**ORG**	**Organization**	**7,915**	**14.76**
f	13	GS	Law	18,520	34.53
f	14	VO	Ordinance	797	1.49
f	15	EUN	EU legal norm	1,499	2.79
c	**4**	**NRM**	**Legal norm**	**20,816**	**38.81**
f	16	VS	Regulation	607	1.13
f	17	VT	Contract	2,863	5.34
c	**5**	**REG**	**Case-by-c. regul.**	**3,470**	**6.47**
f	18				
c	**6**	**RS**	**Court decision**	**12,580**	**23.46**
f	19				
c	**7**	**LIT**	**Legal literature**	**3,006**	**5.60**
Total				**53,632**	**100**

Table 2: Distribution of fine-grained (f) and coarse-grained (c) classes in the dataset

```
import spacy
import en_core_web_lg
from spacy import displacy
from spacy.tokens import Span, Doc
from spacy.lang.en import English
tokenizer = English()

# instantiating the spacy instance
nlp = spacy.load('en_core_web_lg')

# adding the custom function to the pipeline
nlp.add_pipe(legal_entity_recog,name="legal_ent",last=True)
```

```
def legal_entity_recog(tags):
    new_ents = []
    data = tags._.data(tags[0].text)
    meta= data['meta']

    counsel = []
    coram = []
    appset = []
    resset = []


    for name in meta['counsel']:
        counsel.append(set([word.lower() for word in name.split()]))
    for name in meta['coram']:
        name = name.split(',')[0]
        coram.append(set([word.lower() for word in name.split()]))
    for name in meta['appellant']:
        appset.append(set([word.lower() for word in name.split()]))
    for name in meta['respondent']:
        resset.append(set([word.lower() for word in name.split()]))

    meta_names = {'CNS': counsel,'COR': coram,'APP': appset,'RES': resset}

    ######


    for ent in tags.ents:
        # ent is a span which also is an entity identified by spacy

        ########################################

        # identifying if span contains PW or FIR
        if("PW" in set(ent.text.split()) or "FIR" in set(ent.text.split())):
            new_ent = Span(tags,ent.start,ent.end,label="LAW")
            new_ents.append(new_ent)
            continue
        ########################################
        if("judge" in set(ent.lower_.split())):
            if("court" not in set(ent.lower_.split())):
                new_ent = Span(tags,ent.start,ent.end,label="JUDGE")
                new_ents.append(new_ent)
                continue
        ########################################
        # turning all courts into court
        if(ent.label_!="LAW"):
            if("court" in set([txt.lower() for txt in ent.text.split()])):
                new_ent = Span(tags,ent.start,ent.end,label="COURT")
                new_ents.append(new_ent)
                continue
        ########################################
        #identifying acts is not needed as acts is identified by spacy as LAW
```

```
        # if date/number is near a LAW token, then that Date is Also LAW
        # unless there is Rs. Beside Dates
        #take numbers from acts metadata and if such a number/expression present, then
LAW tag
        if(ent.label_ == "DATE" or ent.label_=="CARDINAL"):

            # make a set for act_nums
            act_nums = []
            for tls in meta['acts']:
                for obj in tls:
                    try:
                        for word in tls[obj].split(' '):
                            word = word.strip(', ')
                            if(not word.isalpha()):
                                act_nums.append(word)
                    except:
                        None
            act_nums = set(act_nums)
            flag=0
            for wrds in ent.text.split():
                if(wrds.strip('() ,.;') in act_nums):
                    new_ent = Span(tags,ent.start,ent.end,label="LAW")
                    new_ents.append(new_ent)
                    flag=1
                    break
            if(flag):
                continue
            #band for scanning nbd of token
            band = 3
            flag = 0
            for i in range(2*band+1):
                if(ent.start-band+i>=0 and ent.end-band+i<len(tags)-1):
                    tkns = tags[ent.start-band+i]
                    tkn = tkns.text.lower()
                    tkn = tkn.strip(',.')
                    if(tkn=='rs' or tkn=='sum' or tkn=='money'):
                        new_ent = Span(tags,ent.start,ent.end,label="MONEY")
                        new_ents.append(new_ent)
                        flag=1
                        break
                    if(tkns.ent_type_=="LAW" or tkns.text.lower()=="citation" or
tkns.text.lower()=="section"):
                        new_ent = Span(tags,ent.start,ent.end,label="LAW")
                        new_ents.append(new_ent)
                        flag=1
                        break
            if(flag):
                continue
        ########################################
        # Handling GPE to be related with a court
        if(ent.label_ == "GPE"):
            band = 2
            flag=0
```

```
            for i in range(2*band+1):
                nbr_token = tags[ent.start-band+i]
                if(ent.start-band+i>=0 and ent.end-band+i<len(tags)-1):
                    if(nbr_token.ent_type_=="COURT"):
                        new_ent = Span(tags,ent.start,ent.end,label="COURT")
                        new_ents.append(new_ent)
                        flag=1
                        break
                    elif(nbr_token.ent_type_=="ORG"):
                        if("court" in set([txt.lower() for txt in
nbr_token.text.split()])):
                            new_ent = Span(tags,ent.start,ent.end,label="COURT")
                            new_ents.append(new_ent)
                            flag=1
                            break
            if(flag):
                continue
        ########################################
        if(ent.label_ == "PERSON" or ent.label_ == "ORG"):
            if("court" in set(ent.text.split())):
                new_ent = Span(tags,ent.start,ent.end,label="COURT")
                new_ents.append(new_ent)
                continue

            if(ent.start-1>=0 and ent.start+1<len(doc)):
                if(tags[ent.start-1].text.lower() == "judge" or
tags[ent.start+1].text.lower()=="judge"):
                    new_ent = Span(tags,ent.start,ent.end,label="JUDGE")
                    new_ents.append(new_ent)
                    continue
            # preprocessing for all proper nouns from metadata
            words = [word.lower() for word in ent.orth_.split()]
            flag=0
            #find if this word has a strong match with any coresponding words in the
four lists
            for grp in meta_names:
                names = meta_names[grp]
                cnt = 0
                for name in names:
                    cnt = 0
                    for word in words:
                        if(word in name):
                            cnt+=1
                    if(cnt>=1):
                        if(len(words)==1):
                            new_ent = Span(tags,ent.start,ent.end,label=grp)
                            new_ents.append(new_ent)
                            flag=1
                            break
                        if(cnt>1):
                            new_ent = Span(tags,ent.start,ent.end,label=grp)
                            new_ents.append(new_ent)
                            flag=1
```

```
                            break
                if(flag):
                    break
            if(flag):
                Continue
        new_ents.append(ent)
    print('Done', len(new_ents))
    tags.ents = new_ents
    return tags
```

```
# final run with each corpus, getting POS and Named-Entities

for i in corpus:
    tags = nlp(corpus[i])
    pos = [token.tag_ for token in tags]
    ner = [ent.label for ent in tags.ents]
```

```
# making TF_IDF buckets

corp = list(corpus.values())
tfidf = TfidfVectorizer(ngram_range=(1,1))
tfidf.fit(corp)
finaltf = tfidf.transform(corp)
df = pd.DataFrame(finaltf.toarray())

bins = list(np.linspace(0,0.25,64,endpoint=False)) +
list(np.linspace(0.25,max_val,36))

# assigning into bins
df_bins = np.digitize(df,bins,right=True)

# the tfidf(bins) dataframe
x = pd.DataFrame(df_bins)
x = x.values

# the vocab for identifying word
vocab  = tfidf.vocabulary_

# for generating the corresponding word tfidf vector
def get_tfidf_vec(word,corp_id,data=x,vocab = vocab, rev_ids = rev_ids):
    col = vocab[word]
    row = rev_ids[corp_id]
    cat = x[row][col]
    vec = list(np.zeros(100).astype(int))
    vec[cat] = 1
    return vec
```

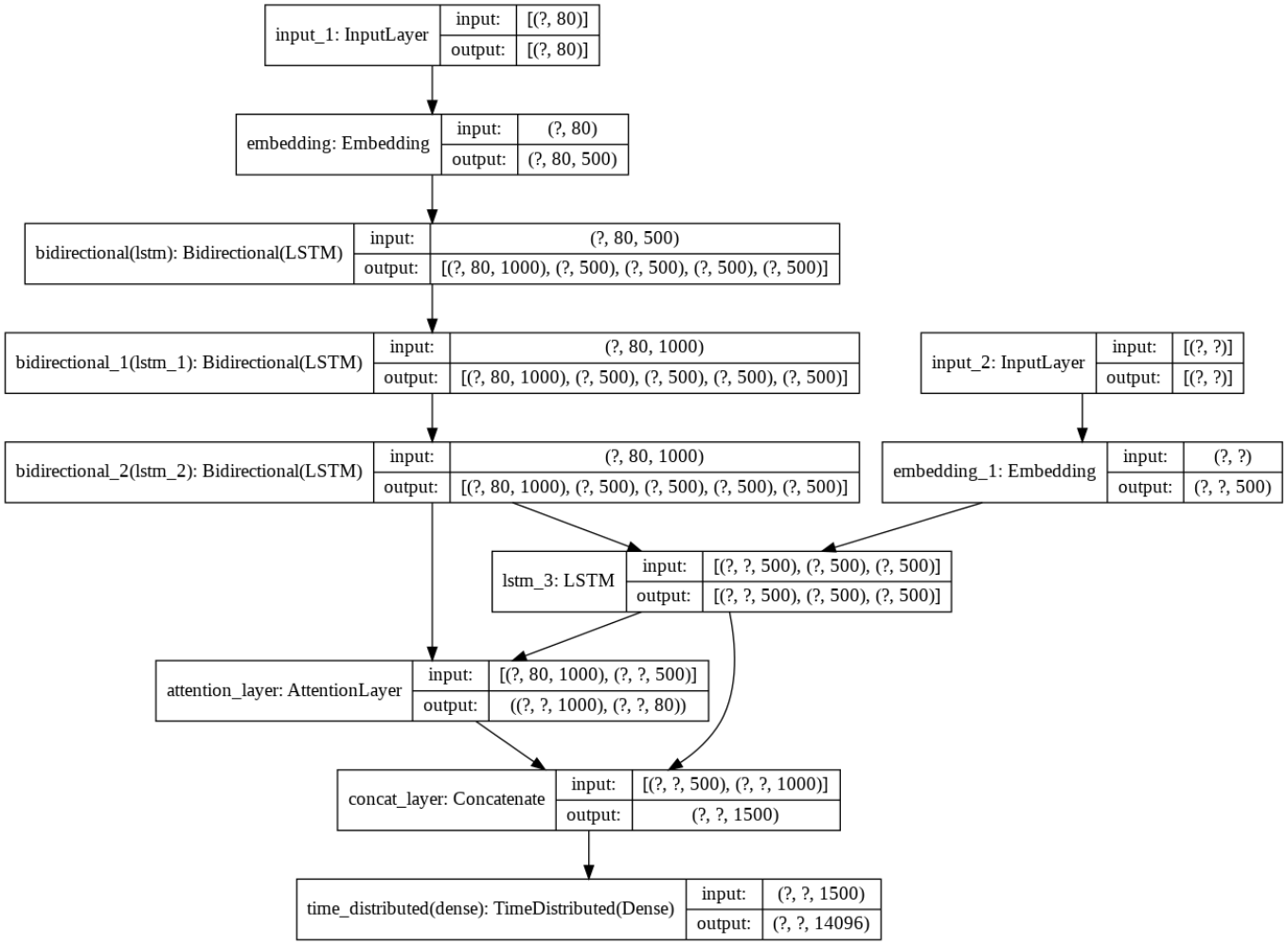
This brings me to the end of Section I

Section II

Creating Model for Abstractive Text Summarization

I have created a Seq2Seq model for Abstractive text summarization on an Amazon Review Dataset from Kaggle, of which I aim to generate reviews.

After cleaning the data and a bit of preprocessing, I input it to the following model:

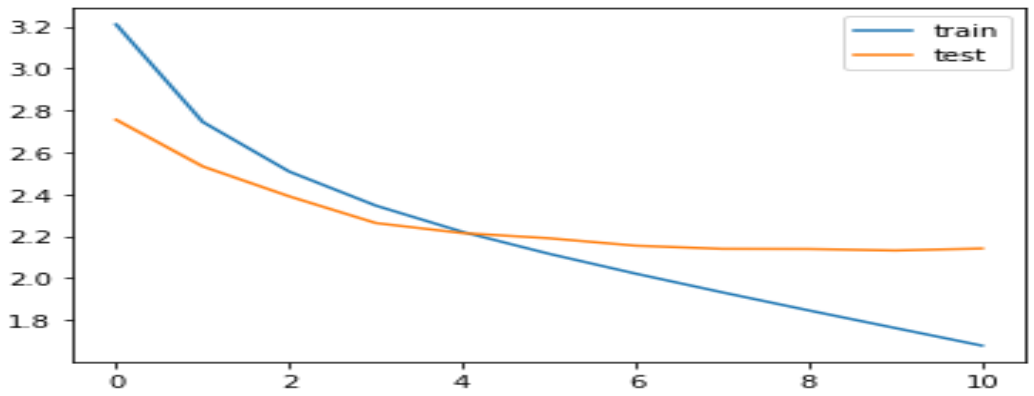


```
from keras import backend as K
K.clear_session()
latent_dim = 500

# Encoder
encoder_inputs = Input(shape=(max_len_text,))
enc_emb = Embedding(x_voc_size, latent_dim,trainable=True)(encoder_inputs)

#LSTM 1
encoder_lstm1 =
Bidirectional(LSTM(latent_dim,return_sequences=True,return_state=True))
encoder_output1, state_h1, state_c1, *_ = encoder_lstm1(enc_emb)

#LSTM 2
encoder_lstm2 =
Bidirectional(LSTM(latent_dim,return_sequences=True,return_state=True))
encoder_output2, state_h2, state_c2, *_ = encoder_lstm2(encoder_output1)

#LSTM 3
encoder_lstm3=Bidirectional(LSTM(latent_dim, return_state=True,
return_sequences=True))
encoder_outputs, state_h, state_c, *_ = encoder_lstm3(encoder_output2)

# Set up the decoder.
decoder_inputs = Input(shape=(None,))
dec_emb_layer = Embedding(y_voc_size, latent_dim,trainable=True)
dec_emb = dec_emb_layer(decoder_inputs)

#LSTM using encoder_states as initial state
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs,decoder_fwd_state, decoder_back_state =
decoder_lstm(dec_emb,initial_state=[state_h, state_c])

#Attention Layer
attn_layer = attention.AttentionLayer(name='attention_layer')
attn_out, attn_states = attn_layer([encoder_outputs, decoder_outputs])

# Concat attention output and decoder LSTM output
decoder_concat_input = Concatenate(axis=-1, name='concat_layer')([decoder_outputs,
attn_out])

#Dense layer
decoder_dense = TimeDistributed(Dense(y_voc_size, activation='softmax'))
decoder_outputs = decoder_dense(decoder_concat_input)

# Define the model
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

Then compiling and training the model:

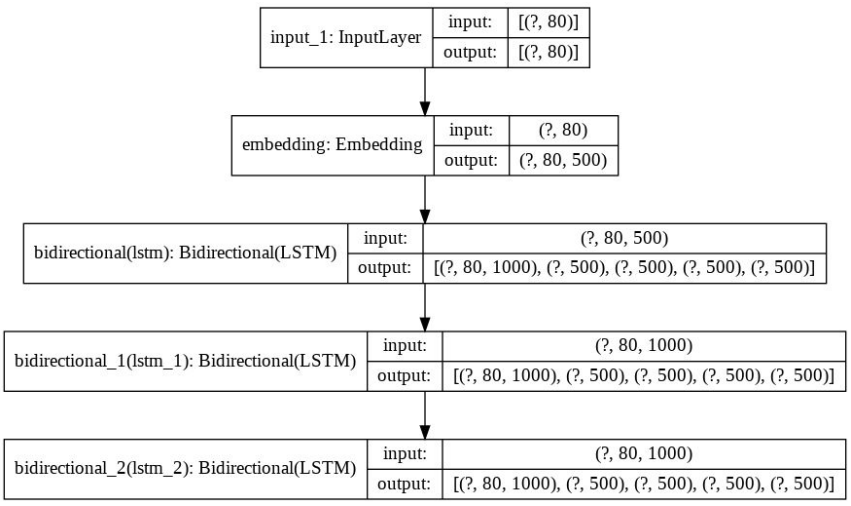
```
model.compile(optimizer='rmsprop', loss='sparse_categorical_crossentropy')

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1)

history=model.fit([x_tr,y_tr[:,:-1]], y_tr.reshape(y_tr.shape[0],y_tr.shape[1],
1)[:,1:] ,epochs=50,callbacks=[es],batch_size=512,
validation_data=([x_val,y_val[:,:-1]], y_val.reshape(y_val.shape[0],y_val.shape[1],
1)[:,1:]))
```

```
Epoch 1/50
156/156 [==============================] - 397s 3s/step - loss: 3.1942 - val_loss: 2.7440
Epoch 2/50
156/156 [==============================] - 410s 3s/step - loss: 2.7440 - val_loss: 2.5695
Epoch 3/50
156/156 [==============================] - 411s 3s/step - loss: 2.5637 - val_loss: 2.4239
Epoch 4/50
156/156 [==============================] - 413s 3s/step - loss: 2.4272 - val_loss: 2.3560
Epoch 5/50
156/156 [==============================] - 413s 3s/step - loss: 2.3289 - val_loss: 2.3018
Epoch 6/50
156/156 [==============================] - 413s 3s/step - loss: 2.2365 - val_loss: 2.2335
Epoch 7/50
156/156 [==============================] - 412s 3s/step - loss: 2.1442 - val_loss: 2.2110
Epoch 8/50
156/156 [==============================] - 411s 3s/step - loss: 2.0579 - val_loss: 2.1686
Epoch 9/50
156/156 [==============================] - 410s 3s/step - loss: 1.9728 - val_loss: 2.1785
Epoch 00009: early stopping
```

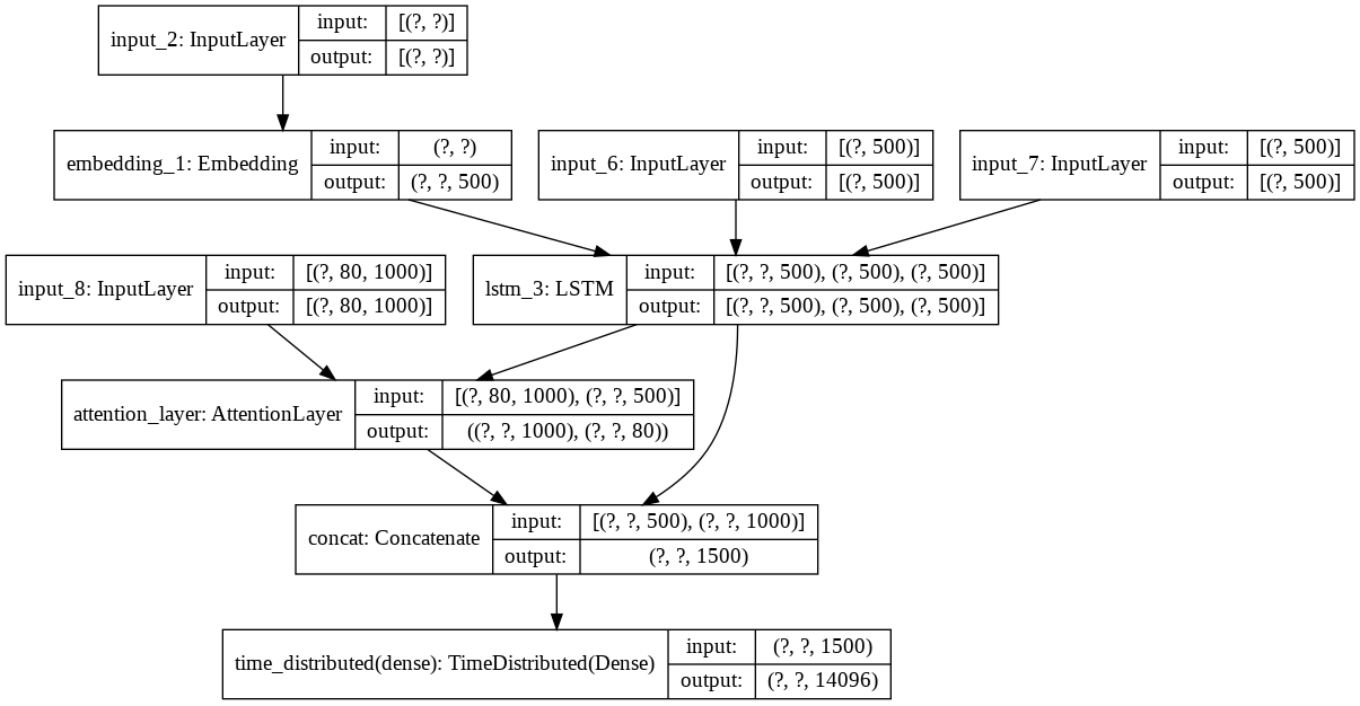


The next step in inference, where we predict the summaries of the reviews.

```
# encoder inference
encoder_model = Model(inputs=encoder_inputs,outputs=[encoder_outputs, state_h,
state_c])

# decoder inference
# Below tensors will hold the states of the previous time step
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_hidden_state_input = Input(shape=(max_len_text,2*latent_dim))

# Get the embeddings of the decoder sequence
dec_emb2= dec_emb_layer(decoder_inputs)

# To predict the next word in the sequence, set the initial states to the states from
the previous time step
decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2,
initial_state=[decoder_state_input_h, decoder_state_input_c])

#attention inference
attn_out_inf, attn_states_inf = attn_layer([decoder_hidden_state_input,
decoder_outputs2])
decoder_inf_concat = Concatenate(axis=-1, name='concat')([decoder_outputs2,
attn_out_inf])

# A dense softmax layer to generate prob dist. over the target vocabulary
decoder_outputs2 = decoder_dense(decoder_inf_concat)

# Final decoder model
decoder_model = Model(
[decoder_inputs] + [decoder_hidden_state_input,decoder_state_input_h,
decoder_state_input_c],
[decoder_outputs2] + [state_h2, state_c2])
```



Encoder Inference Model



Decoder Inference Model

```
# supplementary functions to generate text results from vectors and perform inference

def decode_sequence(input_seq):
    # Encode the input as state vectors.
    e_out, e_h, e_c = encoder_model.predict(input_seq)

    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1,1))

    # Chose the 'start' word as the first word of the target sequence
    target_seq[0, 0] = target_word_index['start']

    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:
        output_tokens, h, c = decoder_model.predict([target_seq] + [e_out, e_h, e_c])

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_token = reverse_target_word_index[sampled_token_index]

        if(sampled_token!='end'):
            decoded_sentence += ' '+sampled_token

            # Exit condition: either hit max length or find stop word.
            if (sampled_token == 'end' or len(decoded_sentence.split()) >=
(max_len_summary-1)):
                stop_condition = True

        # Update the target sequence (of length 1).
        target_seq = np.zeros((1,1))
        target_seq[0, 0] = sampled_token_index
```

```
        # Update internal states
        e_h, e_c = h, c

    return decoded_sentence

def seq2summary(input_seq):
    newString=''
    for i in input_seq:
      if((i!=0 and i!=target_word_index['start']) and i!=target_word_index['end']):
        newString=newString+reverse_target_word_index[i]+' '
    return newString

def seq2text(input_seq):
    newString=''
    for i in input_seq:
      if(i!=0):
        newString=newString+reverse_source_word_index[i]+' '
    return newString
```

Finally, for the output:

```
for i in range(len(x_val)):
    print("Original summary:",seq2summary(y_val[i]))
    print("Predicted summary:",decode_sequence(x_val[i].reshape(1,max_len_text)))
    print("\n")
```

Original summary: alaska smokehouse smoked salmon
Predicted summary: great product

Original summary: it was perfect little sweet without being too sweet
Predicted summary: great tasting coffee

Original summary: great taste
Predicted summary: yummy

Original summary: yum
Predicted summary: delicious

Original summary: delicious brownie
Predicted summary: delicious brownies

Further Work

- The annotated keywords generated from spaCy need to be used for training the NER on legal documents better.
- There needs to be a suitable number of legal judgements along with corresponding summaries
- The legal judgements needs to be analysed properly with regards to the given summaries, and correspondingly more generic features can be identified
- Section I and Section II have to be linked to form one continuous pipeline for summarization

References

- https://arxiv.org/pdf/1807.05511.pdf
- https://arxiv.org/pdf/1602.06023.pdf,
- https://arxiv.org/pdf/1710.09306.pdf
- https://arxiv.org/pdf/1812.02303v3.pdf
- https://link.springer.com/chapter/10.1007/978-3-030-33220-4_20
- https://spacy.io/usage