Summarization of Medical Abstracts

Ricardo Martinez
Dejan Dukic

Contents

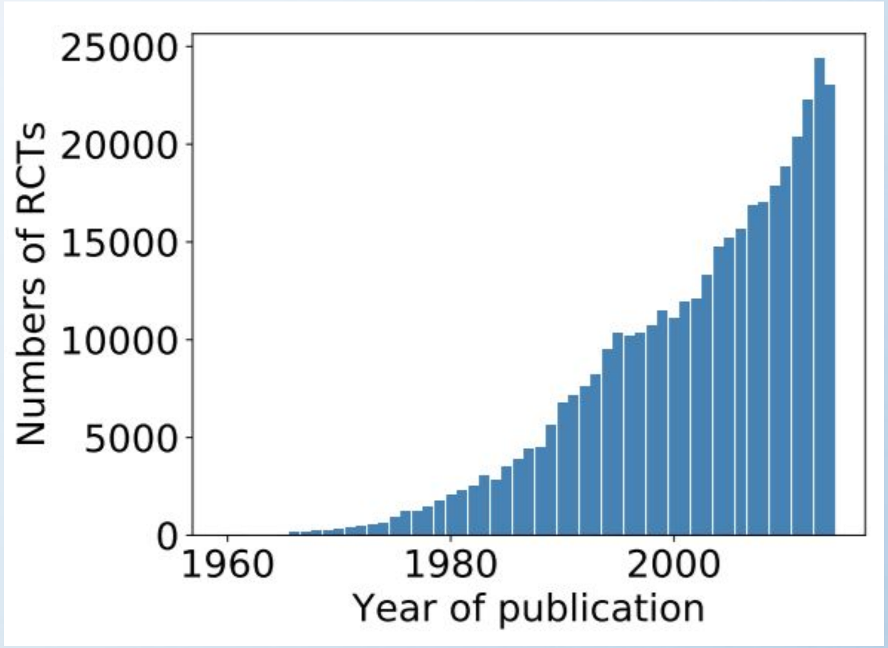
Introduction

Impetus

Staying up to date with literature is a daunting task;

The exponential availability of knowledge in the biomedical field →

PubMed currently containing over 26 million articles, July 2018



Introduction

Data

PubMed 200k Randomized Control Trials

Tools

Word2Vec, Recurrent Neural Networks

Approaches

Extractive: Naive frequentist approach, relying on a metric (i.e. inverse-document frequency) to extract interesting parts of the source and join them back together:

Abstractive: Human-like, non-verbatim rephrasing of the key points from the source; represented by sequence2sequence learning models:

Important to note that a purely abstractive approach does not yet exist (reliance on the extractive pre-processing steps)

Word embeddings

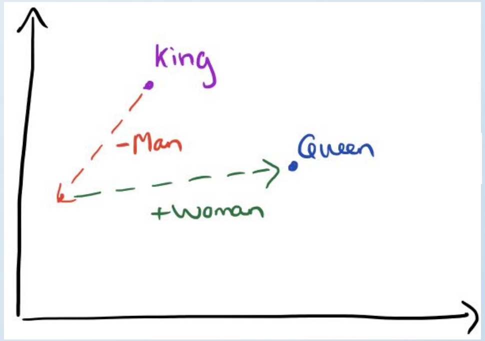
J.R. Firth (1957)

"You shall know a word by the company it keeps"

One-hot encoding vs continuous representation

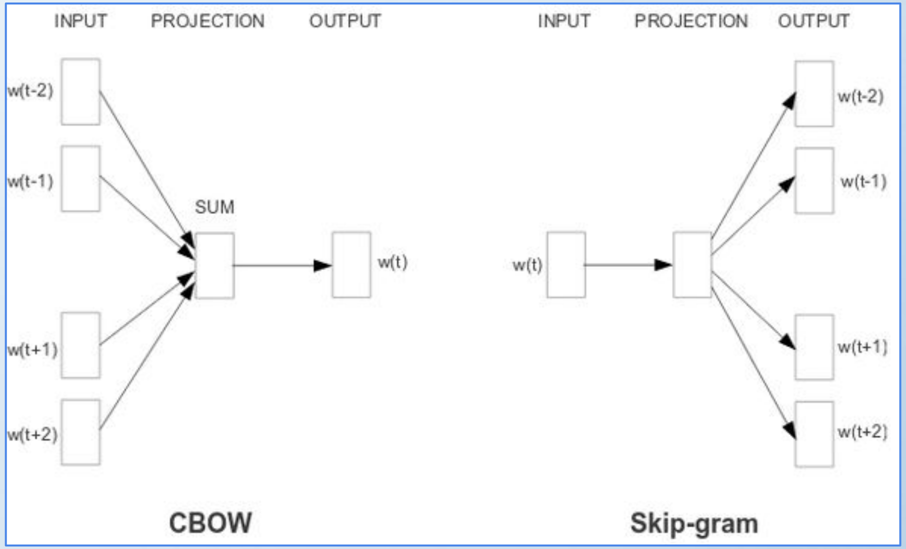
One-hot: no relations between the words preserved; computationally expensive for large corpora

Continuous representations:



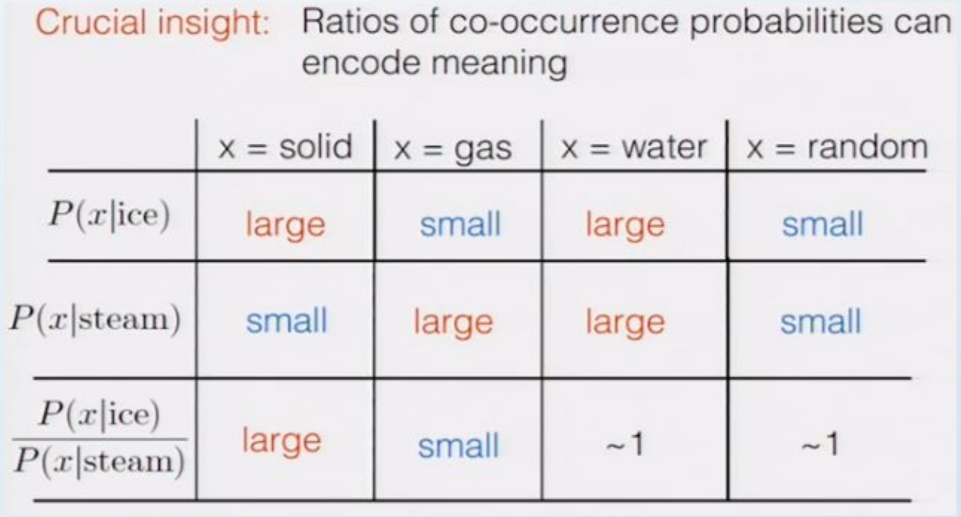
Word2Vec

- Word2Vec
  - Aims to create Vectors by using context
    - “I ***laughed*** at his **joke.”** | **”**His **jokes** weren’t **funny.**”
  - Continuous Bag-Of-Words
  - Skip-Gram



GloVe:
Global Vectors for Word Representation

- GloVe
  - Aims to create Vectors by using word-word co-occurrence statistics from a corpus

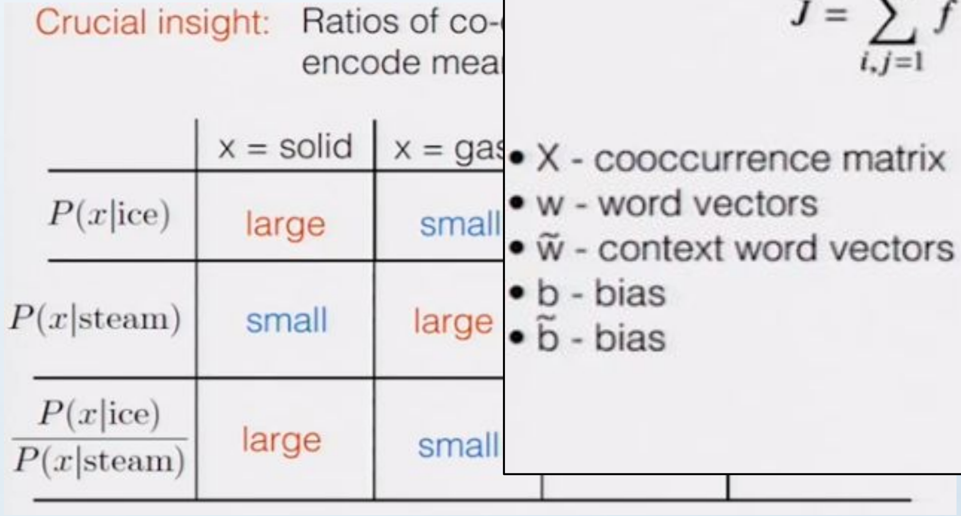


Crucial insight: Ratios of co-occurrence probabilities can encode meaning

	x = solid	x = gas	x = water	x = random
$P(x\|\text{ice})$	large	small	large	small
$P(x\|\text{steam})$	small	large	large	small
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	large	small	~1	~1

GloVe:
Global Vectors for Word Representation

- GloVe
  - Aims to create Vectors by using word-word co-occurrence statistics from a corpus

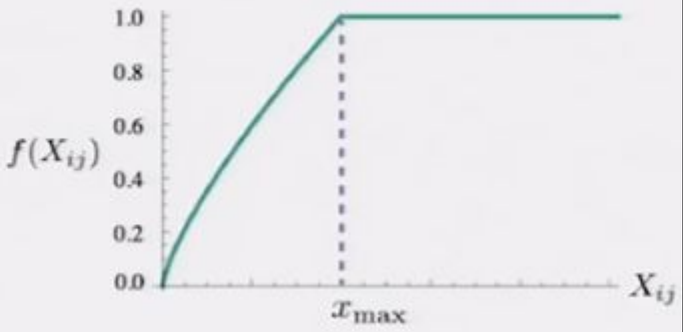


Crucial insight: Ratios of co-
encode mea

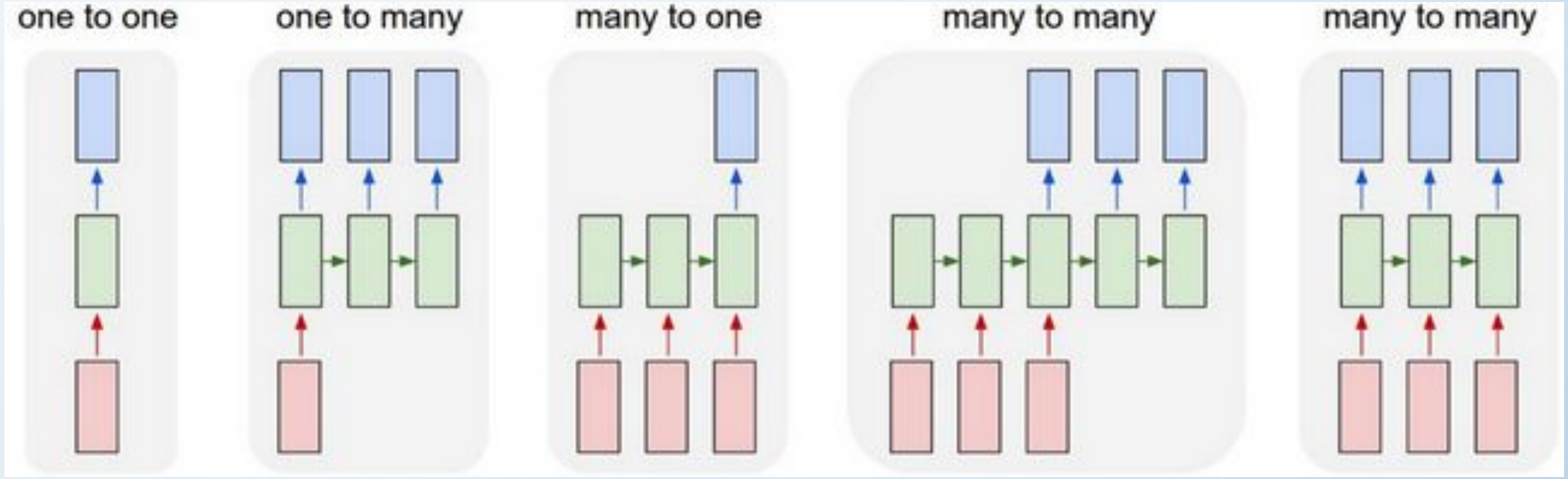
	x = solid	x = gas
$P(x\|\text{ice})$	large	small
$P(x\|\text{steam})$	small	large
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	large	small

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

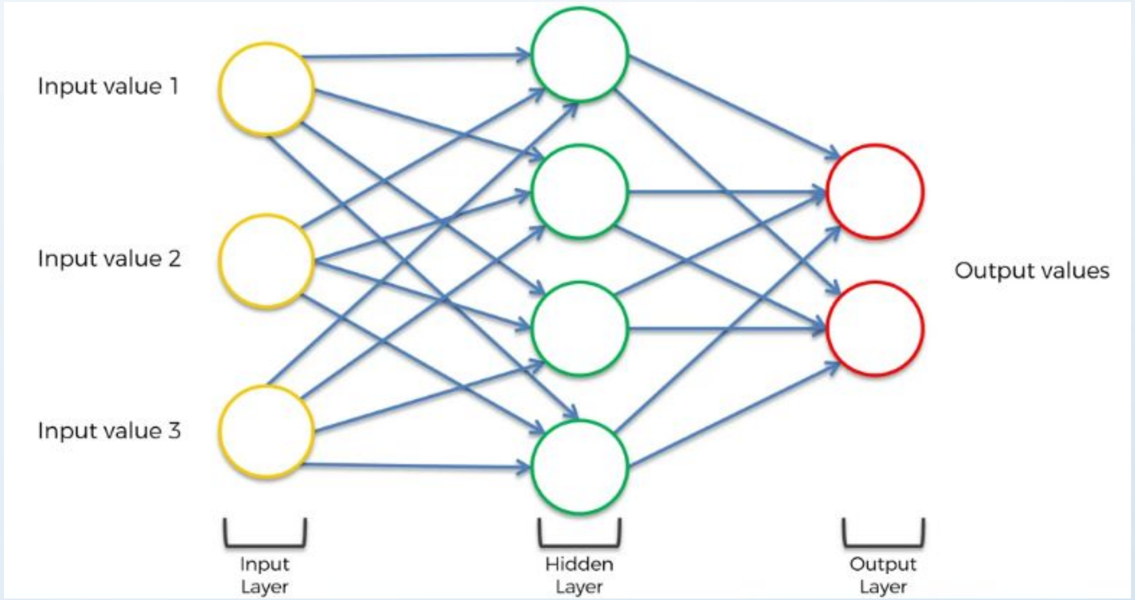
- X - cooccurrence matrix
- w - word vectors
- $\tilde{\text{w}}$ - context word vectors
- b - bias
- $\tilde{\text{b}}$ - bias

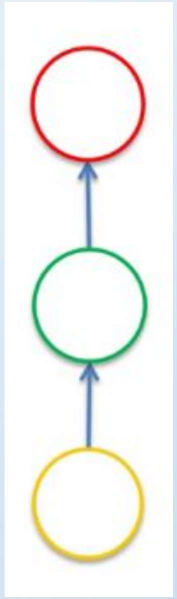


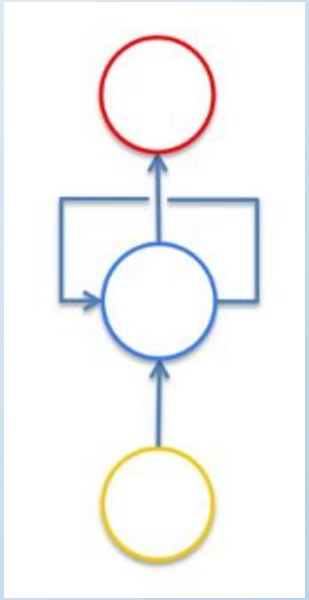
Sequence-to-Sequence Problems



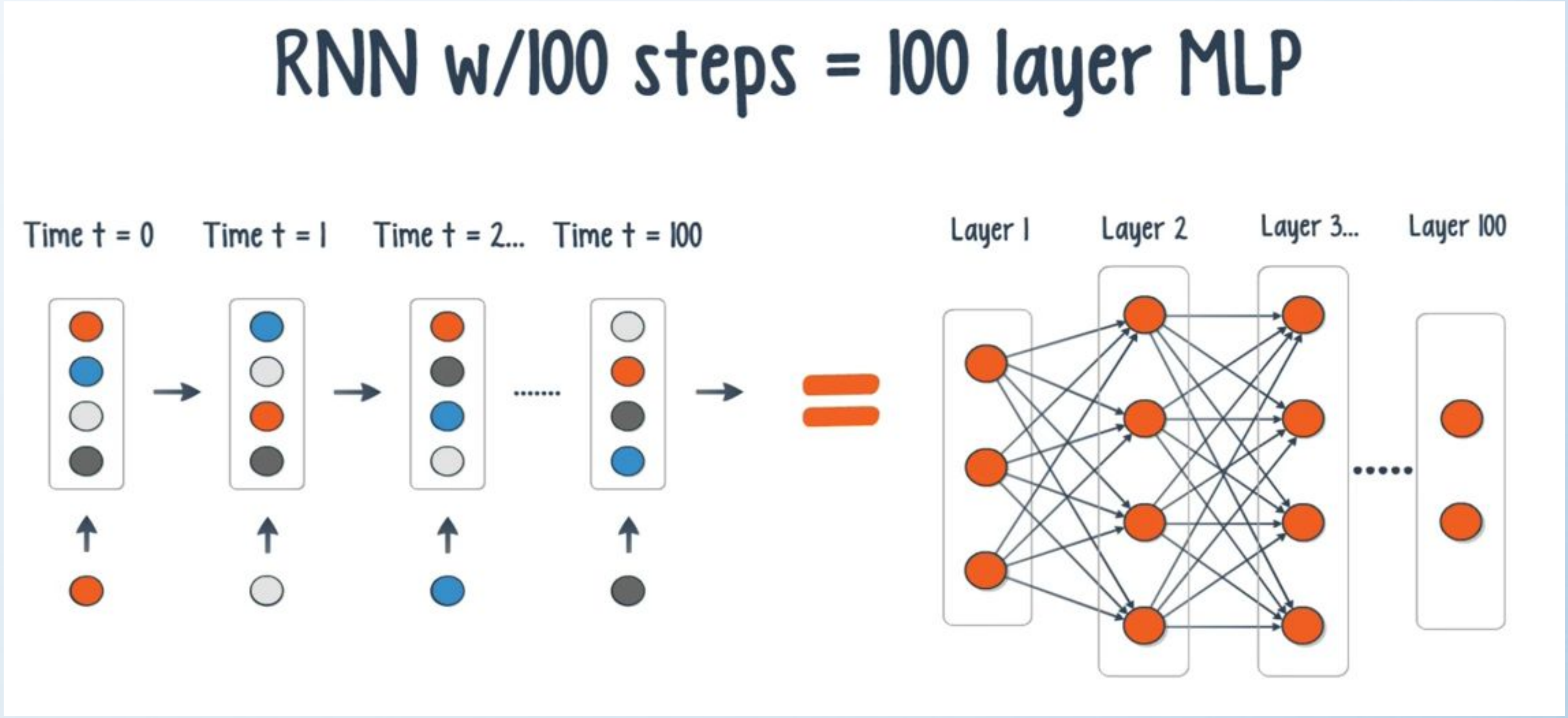
Recurrent Neural Networks



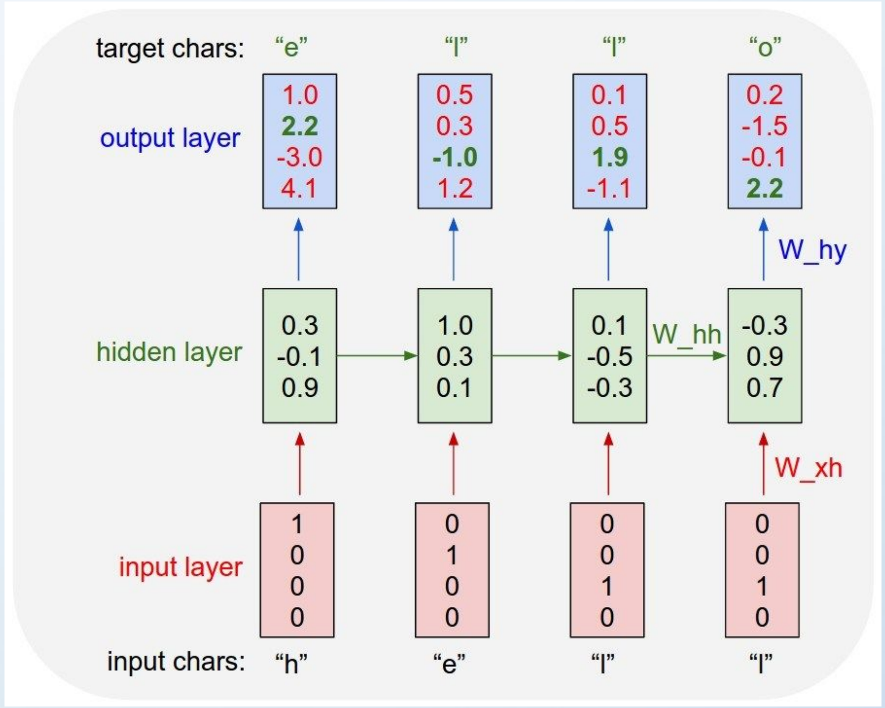




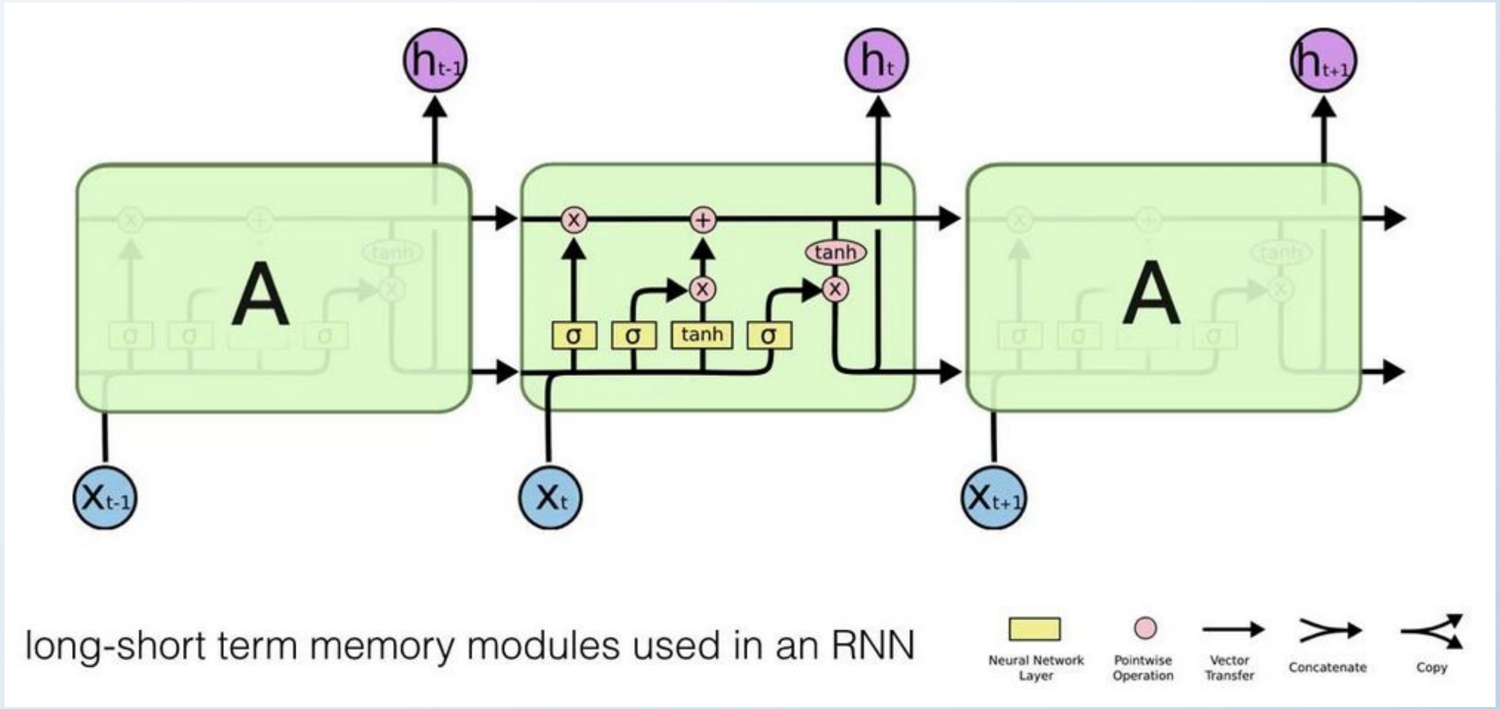
Recurrent Neural Networks



Recurrent Neural Networks

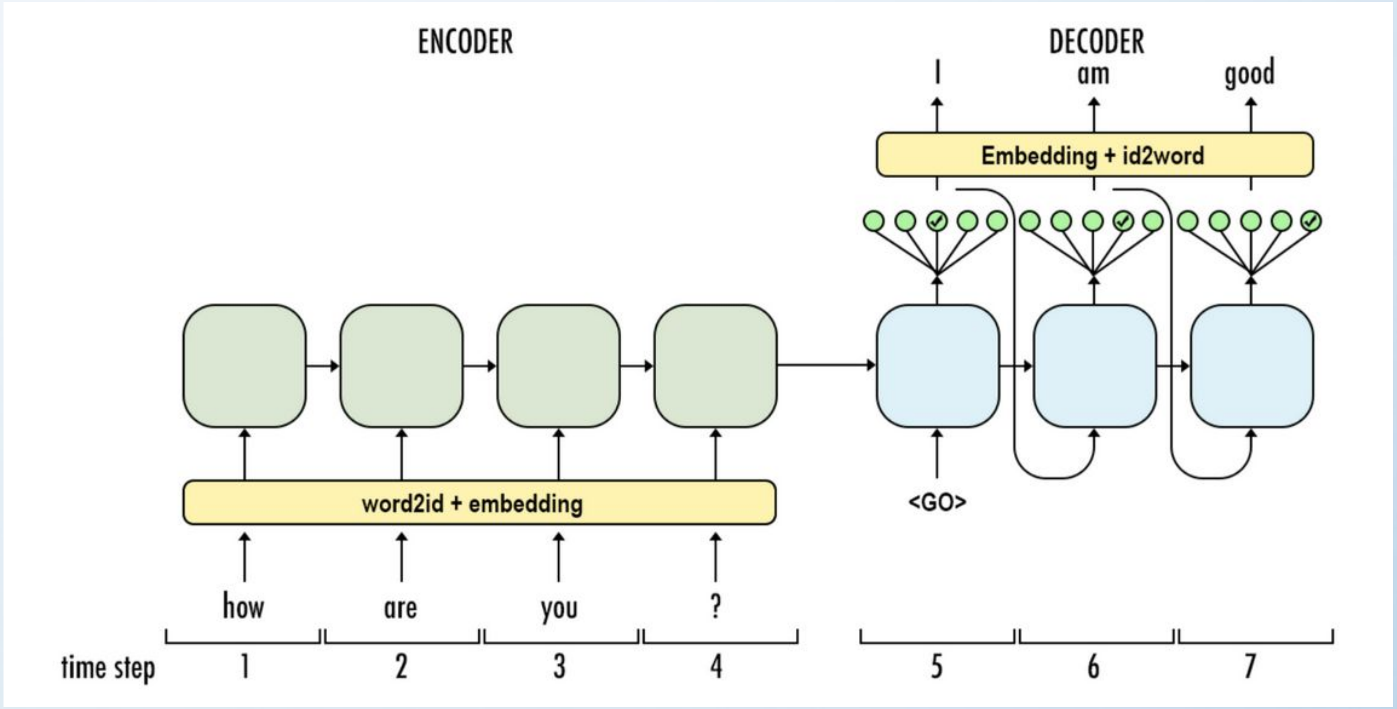


Long Short-Term Memory



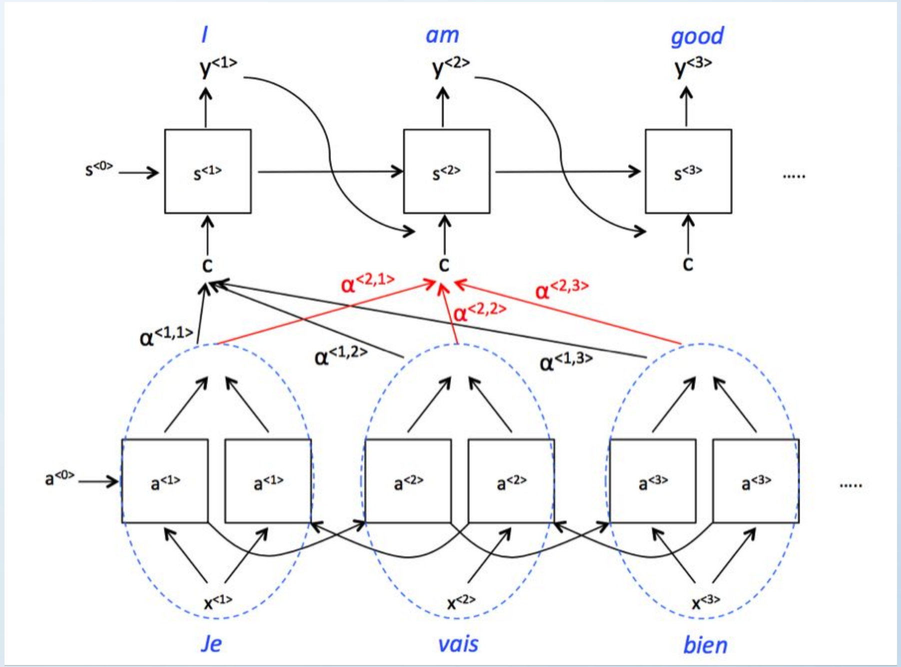
long-short term memory modules used in an RNN

Encoder-Decoder



Attention Model

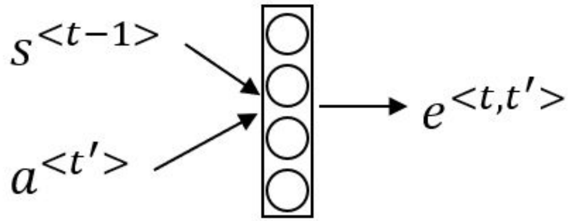
A way of making the network focus on (memorize) particular parts of the sentence

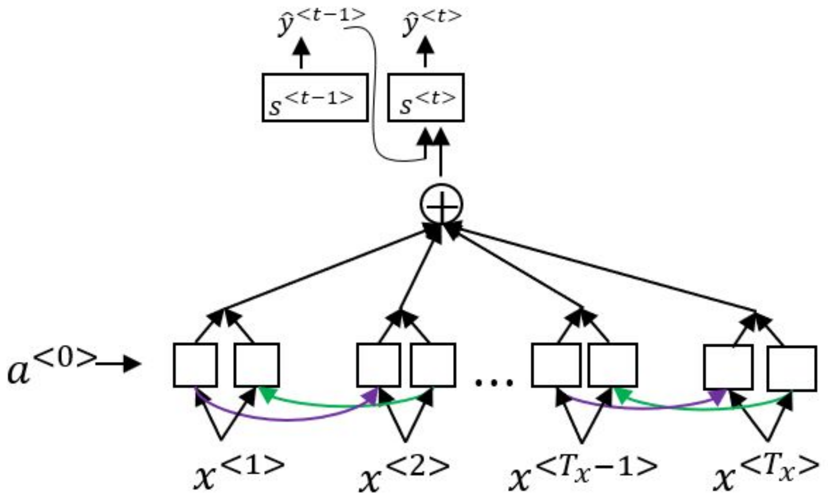


Attention Model & Implementation

$\alpha^{<t,t'>}$ = amount of attention $y^{<t>}$ should pay to $a^{<t'>}$

$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t'=1}^{T_x} \exp(e^{<t,t'>})}$$



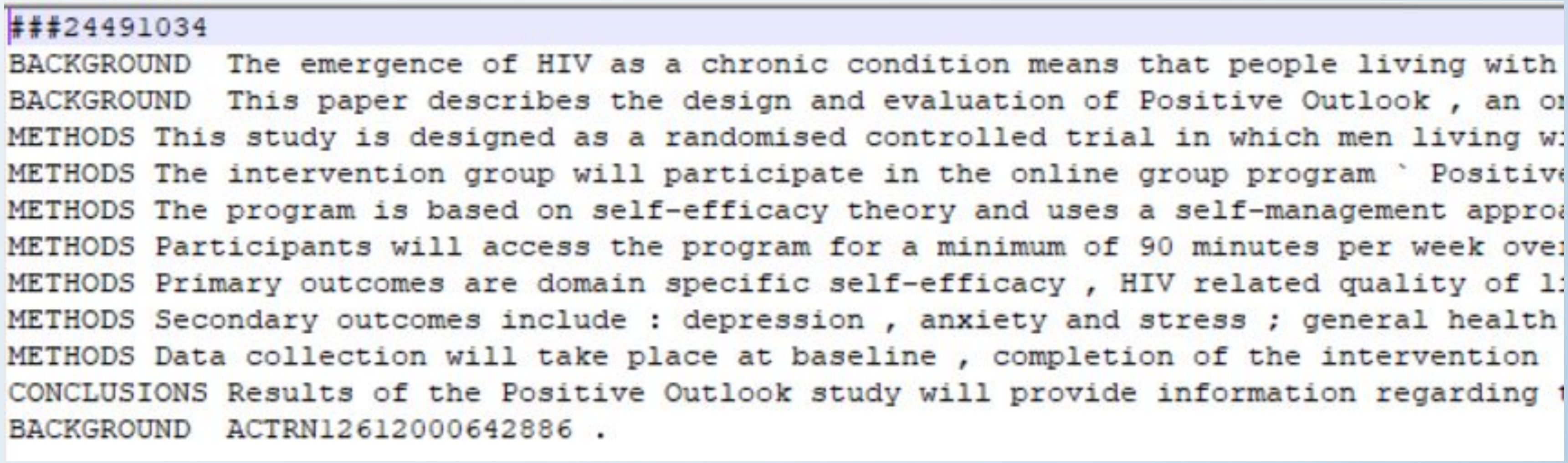


Quadratic run time; len(input) x len(output) makes it not feasible for non specialized hardware

Corpus

200k publications; 500k different tokens

Example:



```
###24491034
BACKGROUND  The emergence of HIV as a chronic condition means that people living with
BACKGROUND  This paper describes the design and evaluation of Positive Outlook , an o
METHODS This study is designed as a randomised controlled trial in which men living w
METHODS The intervention group will participate in the online group program ` Positive
METHODS The program is based on self-efficacy theory and uses a self-management appro
METHODS Participants will access the program for a minimum of 90 minutes per week ove
METHODS Primary outcomes are domain specific self-efficacy , HIV related quality of l
METHODS Secondary outcomes include : depression , anxiety and stress ; general health
METHODS Data collection will take place at baseline , completion of the intervention
CONCLUSIONS Results of the Positive Outlook study will provide information regarding t
BACKGROUND  ACTRN12612000642886 .
```

Results

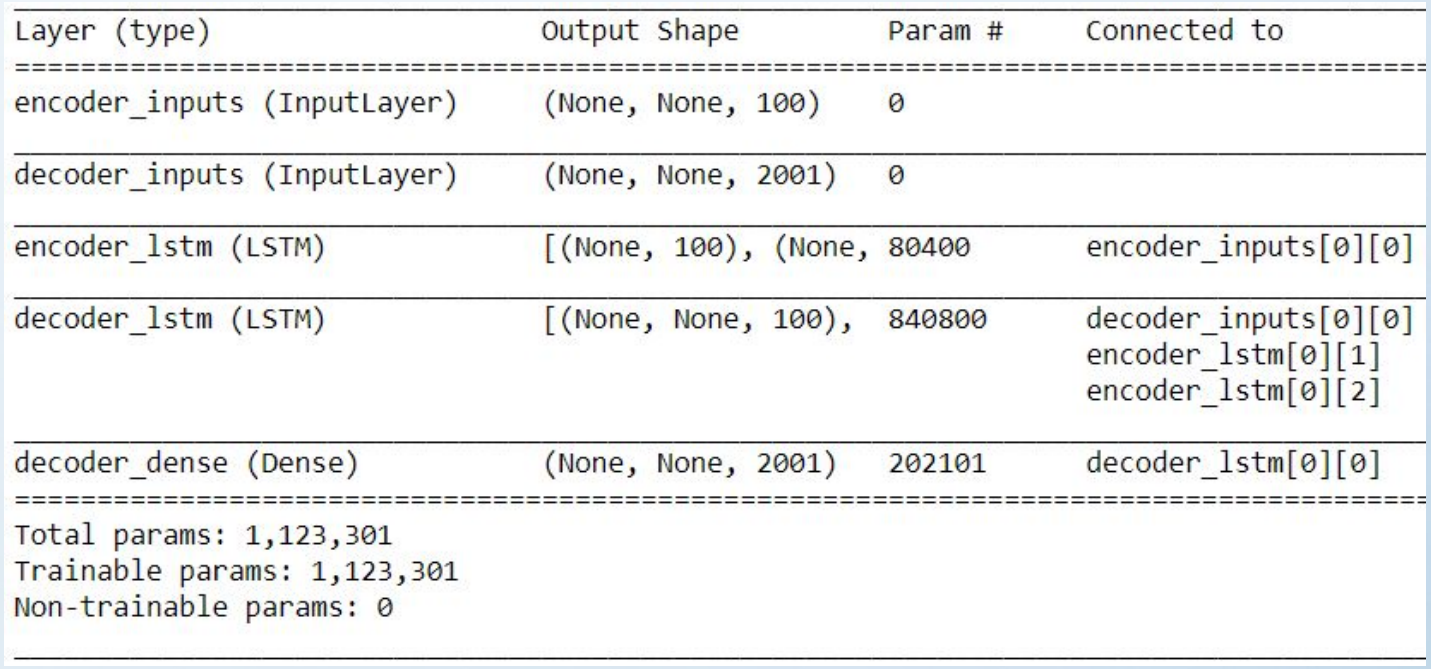
4 different models to display (2 base models, some adjustments)

Training time is about 3 hours per 5 epochs

Predictions are still mostly incoherent

Results; **Model 1**

3.5 different models to display; **0.0763 acc**



```
____________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
============================================================================================
encoder_inputs (InputLayer)     (None, None, 100)    0

____________________________________________________________________________________________
decoder_inputs (InputLayer)     (None, None, 2001)   0

____________________________________________________________________________________________
encoder_lstm (LSTM)             [(None, 100), (None, 80400       encoder_inputs[0][0]

____________________________________________________________________________________________
decoder_lstm (LSTM)             [(None, None, 100),  840800      decoder_inputs[0][0]
                                                                 encoder_lstm[0][1]
                                                                 encoder_lstm[0][2]

____________________________________________________________________________________________
decoder_dense (Dense)           (None, None, 2001)   202101      decoder_lstm[0][0]
============================================================================================
Total params: 1,123,301
Trainable params: 1,123,301
Non-trainable params: 0

____________________________________________________________________________________________
```

Results; **Model 1**

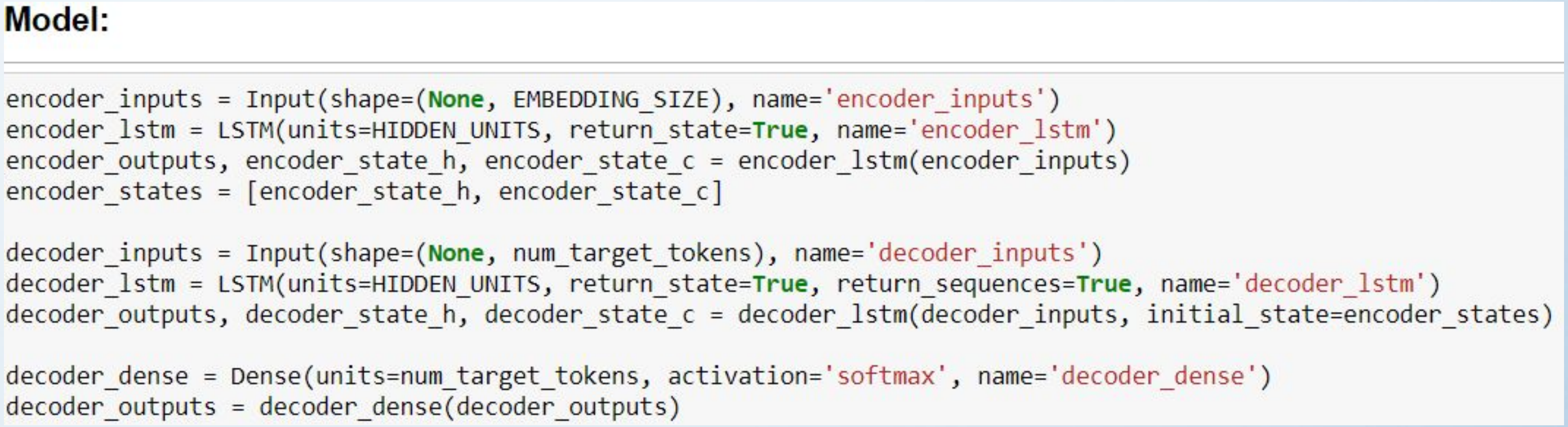
3.5 different models to display; **Model 1: 0.0763 acc**

A simple shallow encoder-decoder architecture, quite slow to train

Issues:

our dictionary size is quite large at **67641** unique tokens (from 10k articles) → OOM issues quite prominent; had to use batch size of 2

Results; **Model 1**



Model:

```
encoder_inputs = Input(shape=(None, EMBEDDING_SIZE), name='encoder_inputs')
encoder_lstm = LSTM(units=HIDDEN_UNITS, return_state=True, name='encoder_lstm')
encoder_outputs, encoder_state_h, encoder_state_c = encoder_lstm(encoder_inputs)
encoder_states = [encoder_state_h, encoder_state_c]

decoder_inputs = Input(shape=(None, num_target_tokens), name='decoder_inputs')
decoder_lstm = LSTM(units=HIDDEN_UNITS, return_state=True, return_sequences=True, name='decoder_lstm')
decoder_outputs, decoder_state_h, decoder_state_c = decoder_lstm(decoder_inputs, initial_state=encoder_states)

decoder_dense = Dense(units=num_target_tokens, activation='softmax', name='decoder_dense')
decoder_outputs = decoder_dense(decoder_outputs)
```

Results; **Model 1**

Why such low performance?

Input example:

‘We conducted randomized , crossover studies that compared the pharmacokinetics (PK) , bioequivalence and safety of topical diclofenac sodium **2 %** twice daily (BID) , diclofenac sodium **1.5 %** four times daily (QID) and oral diclofenac sodium in healthy subjects . The results of three bioequivalence studies are reviewed . Healthy adult subjects (**n = 76**) applied topical diclofenac sodium **2 %** solution (**40.4 mg/2 mL**) BID ; or **1.5 %** solution (**19.3 mg/40 drops**) QID to each knee for **7.5** consecutive days separated by a washout period . Subjects (**n = 22**) in one study also received oral diclofenac sodium **75 mg** BID for **7.5** days .’

Results; **Model 2**

Identical corpus, numbers switched for ‘@’ symbols;

Accuracy after the same number of epochs: **0.0823** (compared to **0.0763**)

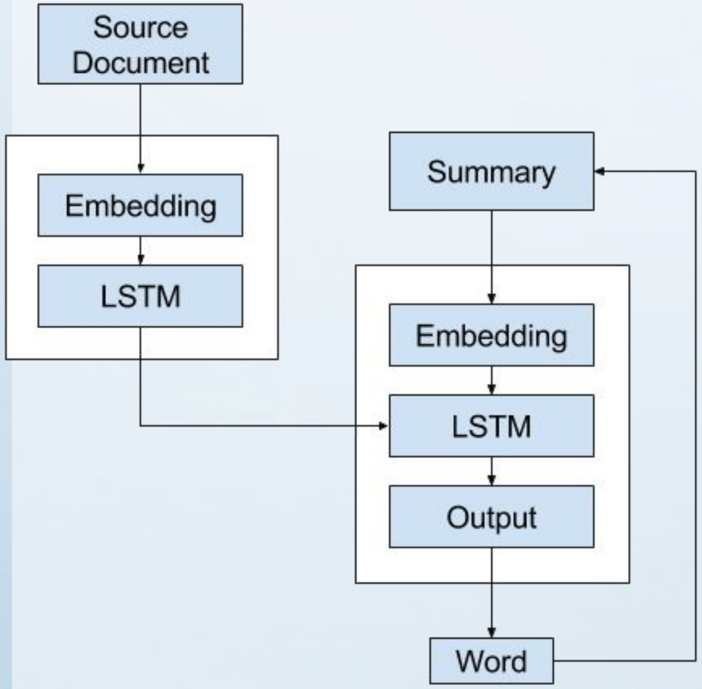
Predictions in the form of:

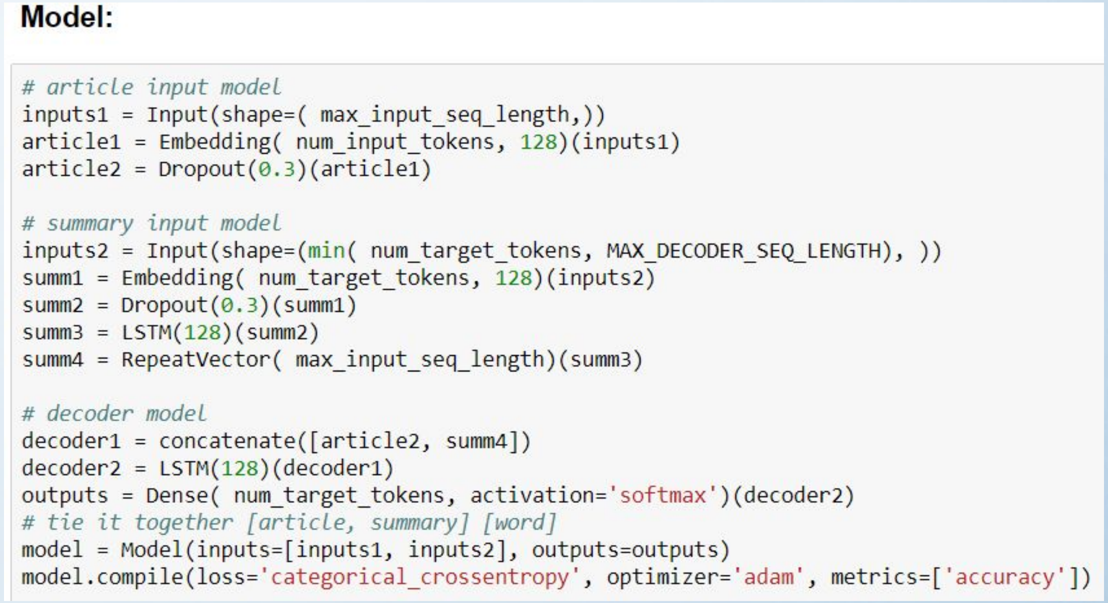
‘experience atorvastatin atorvastatin mexican hepatic atorvastatin mr radiation perceptions colony-stimulating psychological intake extension oncology orthodontic needle illness.’

Conclusion: **the architecture needs to change**

(Nevertheless, we kept working with the modified corpus)

Results; **Model 3**





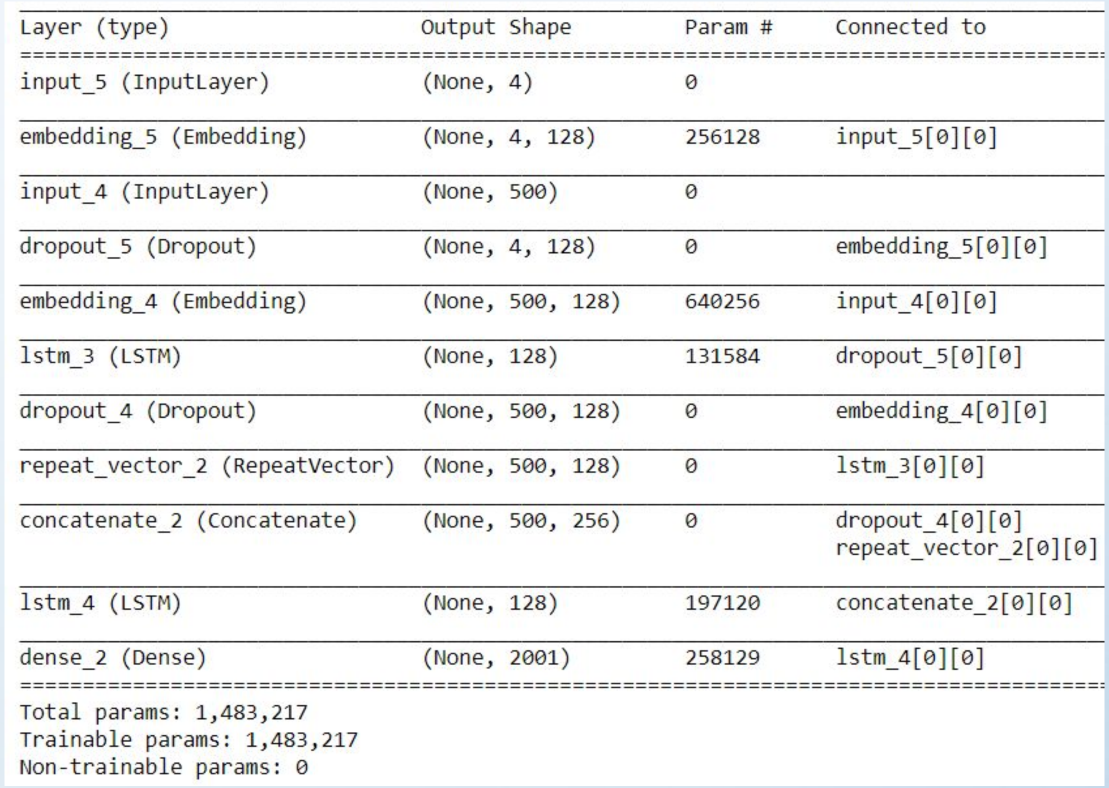
Model:

```
# article input model
inputs1 = Input(shape=( max_input_seq_length,))
article1 = Embedding( num_input_tokens, 128)(inputs1)
article2 = Dropout(0.3)(article1)

# summary input model
inputs2 = Input(shape=(min( num_target_tokens, MAX_DECODER_SEQ_LENGTH), ))
summ1 = Embedding( num_target_tokens, 128)(inputs2)
summ2 = Dropout(0.3)(summ1)
summ3 = LSTM(128)(summ2)
summ4 = RepeatVector( max_input_seq_length)(summ3)

# decoder model
decoder1 = concatenate([article2, summ4])
decoder2 = LSTM(128)(decoder1)
outputs = Dense( num_target_tokens, activation='softmax')(decoder2)
# tie it together [article, summary] [word]
model = Model(inputs=[inputs1, inputs2], outputs=outputs)
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Results; **Model 3**



```
____________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
============================================================================================
input_5 (InputLayer)            (None, 4)            0
____________________________________________________________________________________________
embedding_5 (Embedding)         (None, 4, 128)       256128      input_5[0][0]
____________________________________________________________________________________________
input_4 (InputLayer)            (None, 500)          0
____________________________________________________________________________________________
dropout_5 (Dropout)             (None, 4, 128)       0           embedding_5[0][0]
____________________________________________________________________________________________
embedding_4 (Embedding)         (None, 500, 128)     640256      input_4[0][0]
____________________________________________________________________________________________
lstm_3 (LSTM)                   (None, 128)          131584      dropout_5[0][0]
____________________________________________________________________________________________
dropout_4 (Dropout)             (None, 500, 128)     0           embedding_4[0][0]
____________________________________________________________________________________________
repeat_vector_2 (RepeatVector)  (None, 500, 128)     0           lstm_3[0][0]
____________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 500, 256)     0           dropout_4[0][0]
                                                                 repeat_vector_2[0][0]
____________________________________________________________________________________________
lstm_4 (LSTM)                   (None, 128)          197120      concatenate_2[0][0]
____________________________________________________________________________________________
dense_2 (Dense)                 (None, 2001)         258129      lstm_4[0][0]
============================================================================================
Total params: 1,483,217
Trainable params: 1,483,217
Non-trainable params: 0
```

Results; **Model 3**

Model 3 accuracy: **0.2422**

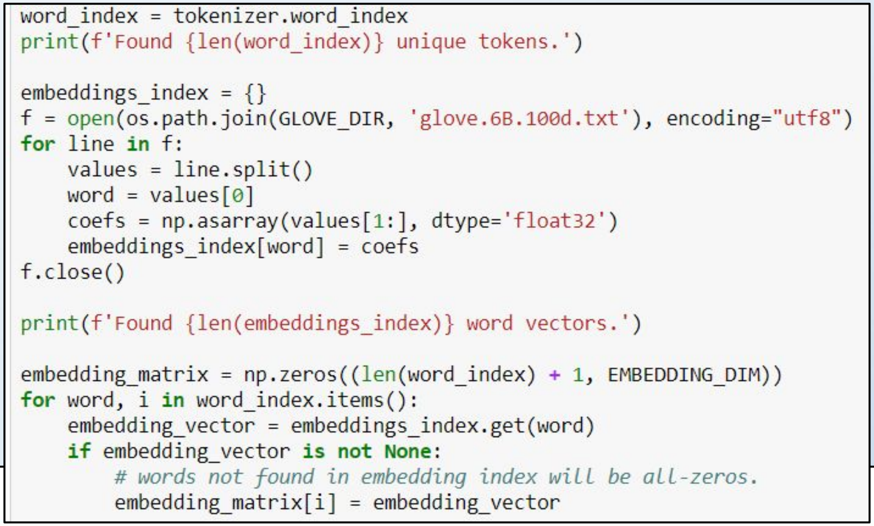
We can finally make (vaguely coherent) predictions:

Generated headline: 'a randomized controlled trial of a brief intervention on the quality of life in patients with type 2 diabetes mellitus: a randomized controlled trial.'

Original headline: 'Continuous subcutaneous delivery of exenatide via ITCA 650 leads to sustained glycemic control and weight loss for 48 weeks in metformin-treated subjects with type 2 diabetes.'

Results; **Model 4**

Same architecture as model 3, using precomputed GloVe based word embeddings:

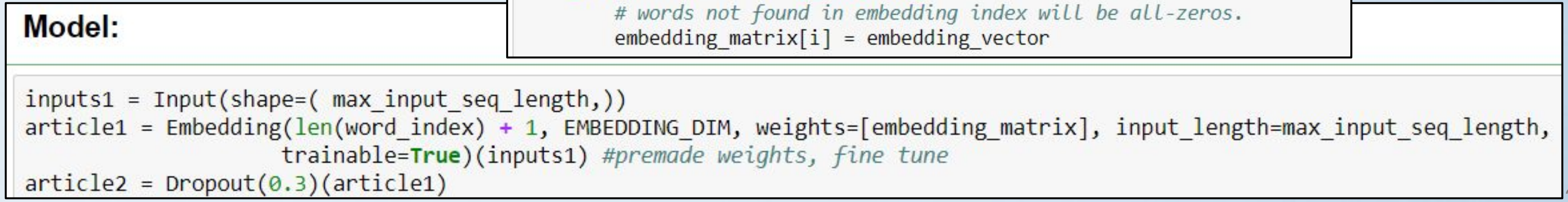


```
word_index = tokenizer.word_index
print(f'Found {len(word_index)} unique tokens.')

embeddings_index = {}
f = open(os.path.join(GLOVE_DIR, 'glove.6B.100d.txt'), encoding="utf8")
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print(f'Found {len(embeddings_index)} word vectors.')

embedding_matrix = np.zeros((len(word_index) + 1, EMBEDDING_DIM))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector
```



Model:

```
inputs1 = Input(shape=( max_input_seq_length,))
article1 = Embedding(len(word_index) + 1, EMBEDDING_DIM, weights=[embedding_matrix], input_length=max_input_seq_length,
                     trainable=True)(inputs1) #premade weights, fine tune
article2 = Dropout(0.3)(article1)
```

Results; **Model 4**

Accuracy: **0.2430**

Marginally better than model 3 after the same number of epochs

Prediction quality indistinguishable:

Generated headline: ‘a randomized controlled trial of a web-based intervention for the treatment of patients with type 2 diabetes mellitus and improve the risk of depression in a randomized controlled trial’

Original headline: ‘Six-month outcomes of a Web-based intervention for users of amphetamine-type stimulants: randomized controlled trial’

Results; **Model 4**

Evaluation based on the ROUGE-1 score:

Average over 50 random predictions: **14.509**

(state of the art is **28.97** (on the DUC corpus))

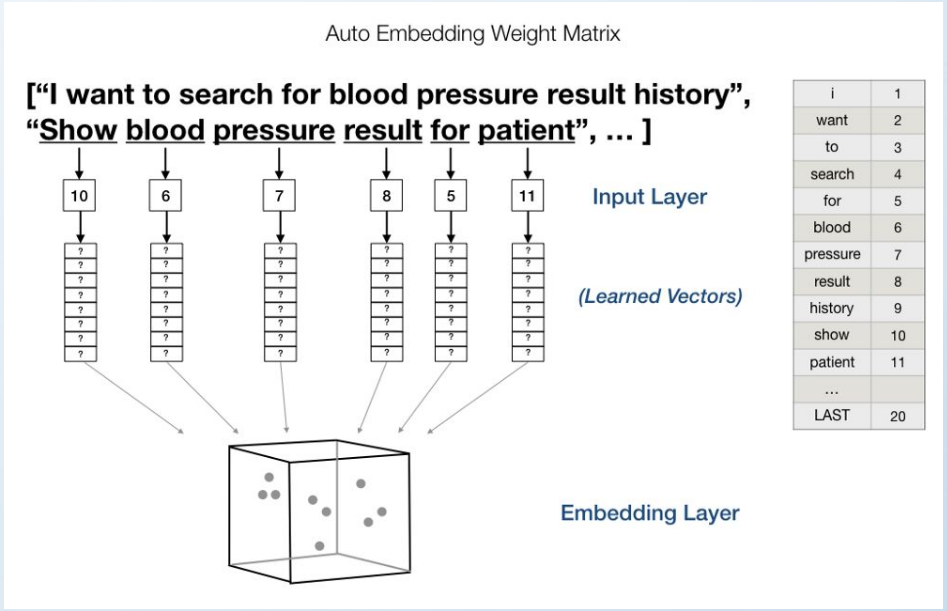
Example of a 50 score:

summary='I would like to eat an apple.'

references='I feel like having an apple.'

Note

Keras Embedding layer API learns word vectors automatically, which was why the gains between model 4 to model 3 were marginal


Auto Embedding Weight Matrix
[“I want to search for blood pressure result history”,
“Show blood pressure result for patient”, …]
10
6
7
8
5
11
Input Layer
(Learned Vectors)
Embedding Layer

i	1
want	2
to	3
search	4
for	5
blood	6
pressure	7
result	8
history	9
show	10
patient	11
…	
LAST	20

Outlook

Using the classification scheme along with summarization, in order to pull the information we most want.... I.E. Segregation by two dimensions.

Doc2Vec can perhaps be used to vectorize an entire corpus allowing one to quickly discriminate against research papers that are only slightly related to one’s interest.

Stacking many LSTMs (+ Attention models), this would require excellent hardware or alternatively payed cloud services

Conclusion

Text summarization is difficult; both in terms of computational complexity and proper implementation

Quotes:

"For lack of a better word, abstractive summarization is a messy task" (Jobson and Gutiérrez, Stanford)

"Since we spent most of time on implementing Model 1, which turns out to be non-trivial.." (Yang , Stanford)

“We trained all our models on a single Tesla K40 GPU. Most models took about 10 hours per epoch on an average except the hierarchical attention model, which took 12 hours per epoch.” - IBM Watson team

thank you
danke
рахмат
謝謝
ngiyabonga
شكراً جزيلاً
teşekkür ederim
spas
Баярлалаа
mersi
kia ora
barka
welalin
tack
mahalo
tapadh leat
спасибо
faafetai lava
vinaka
спасибі
blagodaram
dank je
misaotra
matondo
paldies
grazzi
gracias
хвала
kiitos
dankie
nanni
nandri
dhanyavad
mauruuru
köszönöm
asante
manana
bedankt
bayarlalaa
gràcie
hvala
obrigada
tenki
dankon
aciu
akun
enkosi
djiere dieuf
tau
mochchakkeram
murakoze
дякую
mamnun
dziękuję
chnorakaloutioun
gratias ago
gràcies
sulpáy
go raibh maith agat
chokrane
sobodi
dėkuji
taiku
obrigado
sagolun
sukriya
kop khun krap
arigatô
takk
dakujem
trugarez
mèsi
didi madloba
ありがとう
tanemirt
grazie
najis tuke
terima kasih
rahmet
kam sah hamnida
rahmat
diolch
dhanyavadagalu
shukriya
mercè
мерси
তোমাকে ধন্যবাদ
감사합니다
xièxie
ευχαριστώ
merci

References

Dernoncourt, Franck, and Ji Young Lee. "PubMed 200k RCT: a Dataset for Sequential Sentence Classification in Medical Abstracts." *arXiv preprint arXiv:1710.06071* (2017).

Wikipedia contributors, "Word2vec," *Wikipedia, The Free Encyclopedia,* https://en.wikipedia.org/w/index.php?title=Word2vec&oldid=851879199 (accessed July 27, 2018)

https://machinelearningmastery.com/encoder-decoder-models-text-summarization-keras/

https://machinelearningmastery.com/define-encoder-decoder-sequence-sequence-model-neural-machine-translation-keras

https://machinelearningmastery.com/sequence-prediction-problems-learning-lstm-recurrent-neural-networks/