EDAN96
Applied Machine Learning
Lecture 10: Sequence Prediction

Pierre Nugues

Pierre.Nugues@cs.lth.se
http://cs.lth.se/pierre_nugues/

December 1st, 2021

Outline

In the previous lecture, we used networks to produce one output y per input vector $\mathbf{x}$, for instance one category per sentence.
Given an input sequence $\mathbf{x}$, we will now produce an output sequence: $\mathbf{y}$.
We will experiment four kinds of neural networks:

1. Feed forward
2. Recurrent
3. LSTM
4. Transformers

In the laboratory assignment, you will experiment with items 2. and 3.

Motivation

The analysis of sentences often involves the analysis of words.
We can divide it in three main tasks:

1. Identify the type of word, for instance noun or verb using the classical grammar;
2. Identify a group or segment, for instance are these three words,

   Kjell Olof Andersson,

   the name of a person;
3. Identify the relations between two words: for instance is this group the subject of a verb? This corresponds to parsing, semantic analysis, or information extraction.

We will consider the two first tasks.
This lecture will show you how to solve the first one, part-of-speech tagging, and you will write a program for the second one, named entity recognition (NER), in the next laboratory assignment.

Word Categorization: The Parts of Speech

Sentence:

That round table might collapse

Annotation:

Words	Parts of speech	POS tags
that	Determiner	DET
round	Adjective	ADJ
table	Noun	NOUN
might	Modal verb	AUX
collapse	Verb	VERB

The automatic annotation uses predefined POS tagsets such as the Penn Treebank tagset for English

Ambiguity

Words	Possible tags	Example of use
that	Subordinating conjunction	*That he can swim is good*
	Determiner	*That white table*
	Adverb	*It is not that easy*
	Pronoun	*That is the table*
	Relative pronoun	*The table that collapsed*
round	Verb	*Round up the usual suspects*
	Preposition	*Turn round the corner*
	Noun	*A big round*
	Adjective	*A round box*
	Adverb	*He went round*
table	Noun	*That white table*
	Verb	*I table that*
might	Noun	*The might of the wind*
	Modal verb	*She might come*
collapse	Noun	*The collapse of the empire*
	Verb	*The empire can collapse*

Training Sets: The CoNLL Format

The CoNLL format is a tabular format to distribute annotated texts.
This format was created for evaluations carried out by the Conference in natural language learning
The CoNLL annotation has varied much across the years. We use CoNLL-U, the latest iteration.
Annotation of the Spanish sentence:

> *La reestructuración de los otros bancos checos se está acompañando por la reducción del personal*
> *'The restructuring of Czech banks is accompanied by the reduction of personnel'*

Example of Annotation (CoNLL-U)

La reestructuración de los otros bancos checos se está acompañando por la reducción del personal

ID	FORM	LEMMA	UPOS	FEATS
1	La	el	DET	Definite=Def|Gender=Fem|Number=Sing|PronType=Art
2	reestructuración	reestructuración	NOUN	Gender=Fem|Number=Sing
3	de	de	ADP	AdpType=Prep
4	los	el	DET	Definite=Def|Gender=Masc|Number=Plur|PronType=Art
5	otros	otro	DET	Gender=Masc|Number=Plur|PronType=Ind
6	bancos	banco	NOUN	Gender=Masc|Number=Plur
7	checos	checo	ADJ	Gender=Masc|Number=Plur
8	se	se	PRON	Case=Acc|Person=3|PrepCase=Npr|PronType=Prs|Reflex=Yes
9	está	estar	AUX	Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin
10	acompañando	acompañar	VERB	VerbForm=Ger
11	por	por	ADP	AdpType=Prep
12	la	el	DET	Definite=Def|Gender=Fem|Number=Sing|PronType=Art
13	reducción	reducción	NOUN	Gender=Fem|Number=Sing
14	del	del	ADP	AdpType=Preppron
15	personal	personal	NOUN	Gender=Masc|Number=Sing
16	.	.	PUNCT	PunctType=Peri

Another Example

ID	FORM	LEMMA	PLEMMA	POS	PPOS	FEAT	PFEAT
1	Battle	battle	battle	NN	NN	_	_
2	-	-	-	HYPH	HYPH	_	_
3	tested	tested	tested	NN	NN	_	_
4	Japanese	japanese	japanese	JJ	JJ	_	_
5	industrial	industrial	industrial	JJ	JJ	_	_
6	managers	manager	manager	NNS	NNS	_	_
7	here	here	here	RB	RB	_	_
8	always	always	always	RB	RB	_	_
9	buck	buck	buck	VBP	VB	_	_
10	up	up	up	RP	RP	_	_
11	nervous	nervous	nervous	JJ	JJ	_	_
12	newcomers	newcomer	newcomer	NNS	NNS	_	_
13	with	with	with	IN	IN	_	_
14	the	the	the	DT	DT	_	_
15	tale	tale	tale	NN	NN	_	_
16	of	of	of	IN	IN	_	_
17	the	the	the	DT	DT	_	_
18	first	first	first	JJ	JJ	_	_
19	of	of	of	IN	IN	_	_
20	their	their	their	PRP$	PRP$	_	_
21	countrymen	countryman	countryman	NNS	NNS	_	_
22	to	to	to	TO	TO	_	_
23	visit	visit	visit	VB	VB	_	_
24	Mexico	mexico	mexico	NNP	NNP	_	_
25	,	,	,	,	,	_	_
26	a	a	a	DT	DT	_	_
27	boatload	boatload	boatload	NN	NN	_	_
28	of	of	of	IN	IN	_	_
29	samurai	samurai	samurai	NN	NN	_	_
30	warriors	warrior	warrior	NNS	NNS	_	_
31	blown	blow	blow	VBN	VBN	_	_

Designing a Part-of-Speech Tagger

We will now create part-of-speech taggers, where we will examine three architectures:

1. A feed-forward pipeline with a one-hot encoding of the words;
2. A feed-forward pipeline with word embeddings: We will replace the one-hot vectors with GloVe embeddings;
3. A recurrent neural network, either a simple RNN or a LSTM, with word embeddings.

Features for Part-of-Speech Tagging

The word *visit* is ambiguous in English:

I paid a **visit** *to a friend –> noun*

I went to **visit** *a friend –> verb*

The context of the word enables us to tell, here an article or the infinitive marker

To train and apply the model, the tagger extracts a set of features from the surrounding words, for example, a sliding window spanning five words and centered on the current word.

We then associate the feature vector $(w_{i-2}, w_{i-1}, w_i, w_{i+1}, w_{i+2})$ with the part-of-speech tag t_i at index i.

Part-of-Speech Tagging

ID	FORM	PPOS	
	BOS	BOS	**Padding**
	BOS	BOS	
1	Battle	NN	
2	-	HYPH	
3	tested	NN	
...	...	...	
17	the	DT	
18	first	JJ	
19	of	IN	
20	their	PRP$	
21	countrymen	NNS	**Input features**
22	to	TO	
23	**visit**	VB	**Predicted tag**
24	Mexico		↓
25	,		
26	a		
27	boatload		
...	...	...	
34	years		
35	ago		
36	.		
	EOS		**Padding**
	EOS		

Feature Vectors

ID	Feature vectors							PPOS
	w_{i-2}	w_{i-1}	w_i	w_{i+1}	w_{i+2}	t_{i-2}	t_{i-1}	
1	BOS	BOS	Battle	-	tested	BOS	BOS	NN
2	BOS	Battle	-	tested	Japanese	BOS	NN	HYPH
3	Battle	-	tested	Japanese	industrial	NN	HYPH	JJ
...	...	...	...	...	...	...	...	...
19	the	first	of	their	countrymen	DT	JJ	IN
20	first	of	their	countrymen	to	JJ	IN	PRP$
21	of	their	countrymen	to	visit	IN	PRP$	NNS
22	their	countrymen	to	visit	Mexico	PRP$	NNS	TO
23	countrymen	to	visit	Mexico	,	NNS	TO	VB
24	to	visit	Mexico	,	a	TO	VB	NNP
25	visit	Mexico	,	a	boatload	VB	NNP	,
...	...	...	...	...	...	...	...	...
34	ashore	375	years	ago	.	RB	CD	NNS
35	375	years	ago	.	EOS	CD	NNS	RB
36	years	ago	.	EOS	EOS	NNS	RB	.

Architecture 1: A Feed-Forward Neural Network

We first use a feed-forward architecture corresponding to a logistic regression:

```
np.random.seed(0)

model = models.Sequential([Dense(NB_CLASSES,
                                 input_dim=X.shape[1],
                                 activation='softmax')])
model.compile(loss='sparse_categorical_crossentropy',
              optimizer=OPTIMIZER,
              metrics=['accuracy'])
model.summary()
model.fit(X, y, epochs=EPOCHS, batch_size=BATCH_SIZE)
model.save('out.model')
```

Encoding the **y** Vector

In the previous examples, we used `categorical_crossentropy`
This requires that all the targets are encoded with one-hot vectors
For instance:

- determiner: [1, 0, 0, 0]
- noun: [0, 1, 0, 0]
- verb: [0, 0, 1, 0]
- adjective: [0, 0, 0, 1]

With `sparse_categorical_crossentropy`, we can use numerical indices:

- determiner: 1
- noun: 2
- verb: 3
- adjective: 4

We do not need to use the `to_categorical` function.

Preprocessing

Preprocessing is more complex though: Four steps:

1. Read the corpus

```
train_sentences, dev_sentences, test_sentences, \
  column_names = load_ud_en_ewt()
```

2. Store the rows of the CoNLL corpus in dictionaries

```
conll_dict = CoNLLDictorizer(column_names, col_sep='\t')
train_dict = conll_dict.transform(train_sentences)
test_dict = conll_dict.transform(test_sentences)
```

3. Extract the features and store them in dictionaries

```
context_dictorizer = ContextDictorizer()
context_dictorizer.fit(train_dict)
X_dict, y_cat = context_dictorizer.transform(train_dict)
```

4. Vectorize the symbols

```
# We transform the X symbols into numbers
dict_vectorizer = DictVectorizer()
X_num = dict_vectorizer.fit_transform(X_dict)
```

Code Example

Jupyter Notebook: `4.1-nn-pos-tagger.ipynb`

Architecture 2: Using Embeddings

We replace the one-hot vectors with embeddings, the rest being the same

Word embeddings are dense vectors obtained by a principal component analysis or another method.

They can be trained by the neural network or pretained

In this implementation:

1. We use pretrained embeddings from the GloVe project;
2. Our version of GloVe is lowercased, so we set all the characters in lowercase;
3. We add the embeddings as an `Embedding` layer at the start of the network;
4. We initialize the embedding layer with GloVe and make it trainable or not.

It would be possible to use a randomly initialized matrix as embeddings instead

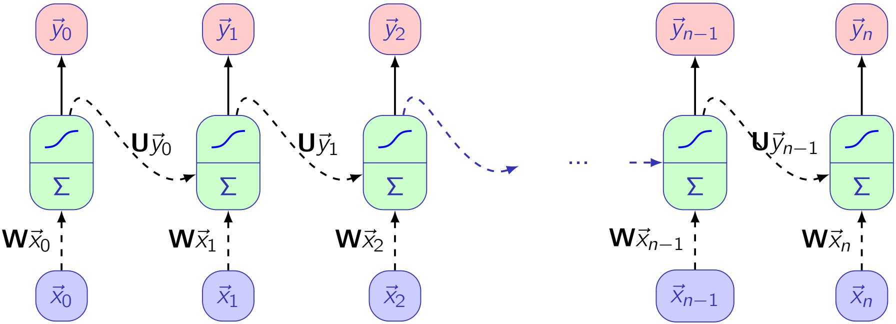
The Embedding Layer

```
embedding_matrix = np.random.random(
                            (len(vocabulary_words) + 1,
                            EMBEDDING_DIM))

for word in vocabulary_words:
    if word in embeddings_dict:
        embedding_matrix[word_idx[word]] = embeddings_dict[word]

model = models.Sequential([
    Embedding(cnt_uniq,
              EMBEDDING_DIM,
              trainable=True,
              embeddings_initializer=
                      initializers.Constant(embedding_matrix),
              input_length=2 * W_SIZE + 1),
    Flatten(),
    Dense(NB_CLASSES, activation='softmax')
])
```

Code Example

Jupyter Notebook: `4.2-nn-pos-tagger-embeddings.ipynb`

The RNN Architecture



Input Format for RNNs

The input format is different from feed forward networks.
We need to build two lists: one for the input and the other for the output

y	DET	NOUN	VERB	DET	NOUN

x	The	waiter	brought	the	meal

All the vectors in a same batch must have the same length. We pad them:

y	PAD	PAD	PAD	DET	NOUN	VERB	DET	NOUN

x	PAD	PAD	PAD	The	waiter	brought	the	meal

We could apply the padding after

Building the Sequences

```
def build_sequences(corpus_dict, key_x='form', key_y='pos',
                    tolower=True):
    X, Y = [], []
    for sentence in corpus_dict:
        x, y = [], []
        for word in sentence:
            x += [word[key_x]]
            y += [word[key_y]]
        if tolower:
            x = list(map(str.lower, x))
        X += [x]
        Y += [y]
    return X, Y
```

At this point, we have **x** and **y** vectors of symbols

Building Index Sequences

0 is for the padding symbol and 1 for the unknown words

```
idx_word = dict(enumerate(vocabulary_words, start=2))
idx_pos = dict(enumerate(pos, start=2))
word_idx = {v: k for k, v in idx_word.items()}
pos_idx = {v: k for k, v in idx_pos.items()}
```

At this point, we have **x** and **y** vectors of numbers

Padding the Index Sequences

We build the complete **X_idx** and **Y_idx** matrices for the whole corpus
And we pad the matrices:

```
X = pad_sequences(X_idx)
Y = pad_sequences(Y_idx)

# The number of POS classes and 0 (padding symbol)
Y_train = to_categorical(Y, num_classes=len(pos) + 2)
```

`pad_sequences` can have an argument that specifies the maximal length `maxlen` (`MAX_SEQUENCE_LENGTH`).
The padded sentences must have the same length in a batch. This is automatically computed by Keras

Recurrent Neural Networks (RNN)

```
model = models.Sequential([
    Embedding(len(vocabulary_words) + 2,
              EMBEDDING_DIM,
              mask_zero=True,
              embeddings_initializer=
                  initializers.Constant(embedding_matrix),
              trainable=True,
              input_length=None),
    SimpleRNN(100, return_sequences=True),
    # Bidirectional(SimpleRNN(100, return_sequences=True)),
    Dense(NB_CLASSES + 2, activation='softmax')])
```

Parameters

Keras functions have many parameters.
In case of doubt, read the documentation
A few useful parameters:

1. `mask_zero=True` is to tell whether or not the input value 0 is a special "padding" value;
2. `return_sequences=True` tells whether to return the last output in the output sequence, or the full sequence. In sequences, it is essential;
3. `recurrent_dropout=0.3` tells how much to drop for the linear transformation of the recurrent state.

Code Example

Jupyter Notebook: `4.3-rnn-pos-tagger.ipynb`

Long Short-Term Memory (LSTM)

```
model = models.Sequential([
    Embedding(len(vocabulary_words) + 2,
              EMBEDDING_DIM,
              mask_zero=True,
              embeddings_initializer=
                      initializers.Constant(embedding_matrix),
              trainable=True,
              input_length=None),
    Bidirectional(LSTM(100, return_sequences=True)),
    Dense(NB_CLASSES + 2, activation='softmax')])
```

Segment Recognition

Group detection – chunking –:

Brackets: [$_{NG}$ The government $_{NG}$] has [$_{NG}$ other agencies and instruments $_{NG}$] for pursuing [$_{NG}$ these other objectives $_{NG}$] .

Tags: *The*/I *government*/I *has*/O *other*/I *agencies*/I *and*/I *instruments*/I *for*/O *pursuing*/O *these*/I *other*/I *objectives*/I *.*/O

Brackets: Even [$_{NG}$ Mao Tse-tung $_{NG}$] [$_{NG}$'s China $_{NG}$] began in [$_{NG}$ 1949 $_{NG}$] with [$_{NG}$ a partnership $_{NG}$] between [$_{NG}$ the communists $_{NG}$] and [$_{NG}$ a number $_{NG}$] of [$_{NG}$ smaller, non-communists parties $_{NG}$] .

Tags: *Even*/O *Mao*/I *Tse-tung*/I *'s*/B *China*/I *began*/O *in*/O *1949*/I *with*/O *a*/I *partnership*/I *between*/O *the*/I *communists*/I *and*/O *a*/I *number*/I *of*/O *smaller*/I *,*/I *non-communists*/I *parties*/I *.*/O

Segment Categorization

Tages extendible to any type of chunks: nominal, verbal, etc.
For the IOB scheme, this means tags such as I.Type, O.Type, and B.Type, Types being NG, VG, PG, etc.
In CoNLL 2000, ten types of chunks

Word	POS	Group	Word	POS	Group
He	PRP	B-NP	*to*	TO	B-PP
reckons	VBZ	B-VP	*only*	RB	B-NP
the	DT	B-NP	*£*	#	I-NP
current	JJ	I-NP	*1.8*	CD	I-NP
account	NN	I-NP	*billion*	CD	I-NP
deficit	NN	I-NP	*in*	IN	B-PP
will	MD	B-VP	*September*	NNP	B-NP
narrow	VB	I-VP	*.*	.	O

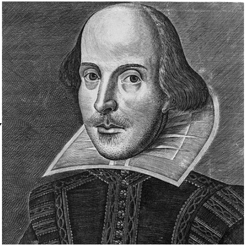
Noun groups (NP) are in red and verb groups (VP) are in blue.

IOB Annotation for Named Entities

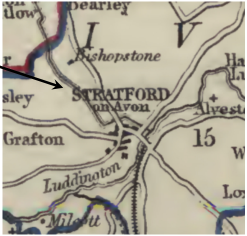
CoNLL 2002	
Words	**Named entities**
Wolff	B-PER
,	O
currently	O
a	O
journalist	O
in	O
Argentina	B-LOC
,	O
played	O
with	O
Del	B-PER
Bosque	I-PER
in	O
the	O
final	O
years	O
of	O
the	O
seventies	O
in	O
Real	B-ORG
Madrid	I-ORG
.	O

CoNLL 2003			
Words	**POS**	**Groups**	**Named entities**
U.N.	NNP	I-NP	I-ORG
official	NN	I-NP	O
Ekeus	NNP	I-NP	I-PER
heads	VBZ	I-VP	O
for	IN	I-PP	O
Baghdad	NNP	I-NP	I-LOC
.	.	O	O

Named Entities: Proper Nouns



William Shakespeare was born and brought up in Stratford-upon-Avon



Others Entities: Common Nouns

Meeting with our guest on the landing at lunchtime

Evaluation

There are different kinds of measures to evaluate the performance of machine learning techniques, for instance:

- Precision and recall in information retrieval and natural language processing;
- The *receiver operating characteristic* (ROC) in medicine.

	Positive examples: P	**Negative examples:** N
Classified as P	True positives: A	False positives: B
Classified as N	False negatives: C	True negatives: D

More on the receiver operating characteristic here: `http://en.wikipedia.org/wiki/Receiver_operating_characteristic`

Recall, Precision, and the F-Measure

The **accuracy** is $\dfrac{|A \cup D|}{|P \cup N|}$.

Recall measures how much relevant examples the system has classified correctly, for P:

$$\text{Recall} = \frac{|A|}{|A \cup C|}.$$

Precision is the accuracy of what has been returned, for P:

$$\text{Precision} = \frac{|A|}{|A \cup B|}.$$

Recall and precision are combined into the **F-measure**, which is defined as the harmonic mean of both numbers:

$$F = \frac{2 \cdot \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$

Evaluation: Accuracy, precision, and recall

For noun groups with the predicted output:

Word	POS	Group	Predicted		Word	POS	Group	Predicted	
He	PRP	B-NP	B-NP		*to*	TO	B-PP	B-PP	
reckons	VBZ	B-VP	B-VP		*only*	RB	B-NP	B-NP	X
the	DT	B-NP	B-NP	X	*£*	#	I-NP	I-NP	X
current	JJ	I-NP	**B-NP**	X	*1.8*	CD	I-NP	**B-NP**	X
account	NN	I-NP	I-NP	X	*billion*	CD	I-NP	I-NP	X
deficit	NN	I-NP	I-NP	X	*in*	IN	B-PP	B-PP	
will	MD	B-VP	B-VP		*September*	NNP	B-NP	B-NP	
narrow	VB	I-VP	I-VP		*.*	.	O	O	

There are 16 chunk tags, 14 are correct: Accuracy $= \frac{14}{16} = 0.875$
There are 4 noun groups, the system retrieved 2 of them: Recall $= \frac{2}{4} = 0.5$
The system identified 6 noun groups, two are correct: Precision $= \frac{2}{6} = 0.33$
Harmonic mean $= 2 \times \frac{0.33 \times 0.5}{0.33 + 0.5} = 0.4$

Transformers

An architecture proposed in 2018 based on the concept of **attention**

Sometimes marketed as the ImageNet moment (See https://ruder.io/nlp-imagenet/)

The similarity lies in the possibility to train semantic relations on very large corpora and memorize them in matrices

We obtain these matrices through a **masked language model**

Contextual Embeddings

Embeddings we have seen so far do not take the context into account
Attention is a way to make them aware of the context.
Consider the sentence:

> *I must go back to my ship and to my crew*
> *Odyssey, book I*

The word *ship* can be a verb or a noun with different meanings, but has only one GloVe embedding vector
Self-attention will enable us to compute contextual word embeddings.

Self-Attention

In the paper *Attention is all you need*, Vaswani et al. (2017) use three kinds of vectors, queries, keys, and values. Here we will use one type corresponding to GloVe embeddings.
We compute the attention this way:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{QK}^\intercal}{\sqrt{d_k}})\mathbf{V},$$

where d_k is the dimension of the input. The softmax function is defined as:

$$\text{softmax}(x_1, x_2, \ldots, x_j, \ldots, x_n) = (\frac{e^{x_1}}{\sum_{i=1}^n e^{x_i}}, \frac{e^{x_2}}{\sum_{i=1}^n e^{x_i}}, \ldots, \frac{e^{x_j}}{\sum_{i=1}^n e^{x_i}}, \ldots, \frac{e^{x_n}}{\sum_{i=1}^n e^{x_i}})$$

also defined as

$$\text{softmax}(x_1, x_2, \ldots, x_j, \ldots, x_n) = (\frac{e^{-x_1}}{\sum_{i=1}^n e^{-x_i}}, \frac{e^{-x_2}}{\sum_{i=1}^n e^{-x_i}}, \ldots, \frac{e^{-x_j}}{\sum_{i=1}^n e^{-x_i}}, \ldots, \frac{e^{-x_n}}{\sum_{i=1}^n e^{-x_i}})$$

in physics.

The meaning of **QK**

QK is the dot product of the GloVe vectors. It will tell us the similarity between the words

This is analogous to cosine similarity:

	i	must	go	back	to	my	ship	and	to	my	crew
i	1.00	0.75	0.86	0.76	0.73	0.90	0.35	0.65	0.73	0.90	0.42
must	0.75	1.00	0.85	0.68	0.87	0.69	0.42	0.69	0.87	0.69	0.45
go	0.86	0.85	1.00	0.84	0.84	0.81	0.41	0.68	0.84	0.81	0.49
back	0.76	0.68	0.84	1.00	0.83	0.76	0.49	0.77	0.83	0.76	0.51
to	0.73	0.87	0.84	0.83	1.00	0.68	0.54	0.86	1.00	0.68	0.51
my	0.90	0.69	0.81	0.76	0.68	1.00	0.38	0.63	0.68	1.00	0.44
ship	0.35	0.42	0.41	0.49	0.54	0.38	1.00	0.46	0.54	0.38	0.78
and	0.65	0.69	0.68	0.77	0.86	0.63	0.46	1.00	0.86	0.63	0.49
to	0.73	0.87	0.84	0.83	1.00	0.68	0.54	0.86	1.00	0.68	0.51
my	0.90	0.69	0.81	0.76	0.68	1.00	0.38	0.63	0.68	1.00	0.44
crew	0.42	0.45	0.49	0.51	0.51	0.44	0.78	0.49	0.51	0.44	1.00

Vaswani's attention score

The attention scores are scaled and normalized by the softmax function.

$$\mathrm{softmax}(\frac{\mathbf{Q}\mathbf{K}^\intercal}{\sqrt{d_k}}),$$

	i	must	go	back	to	my	ship	and	to	my	crew
i	0.36	0.05	0.07	0.05	0.04	0.19	0.01	0.02	0.04	0.19	0.01
must	0.14	0.20	0.10	0.06	0.11	0.10	0.03	0.05	0.11	0.10	0.02
go	0.18	0.09	0.14	0.09	0.08	0.13	0.02	0.04	0.08	0.13	0.02
back	0.14	0.05	0.09	0.19	0.08	0.12	0.03	0.06	0.08	0.12	0.03
to	0.11	0.11	0.09	0.09	0.15	0.08	0.04	0.07	0.15	0.08	0.03
my	0.19	0.03	0.05	0.04	0.03	0.29	0.01	0.02	0.03	0.29	0.01
ship	0.03	0.03	0.03	0.04	0.05	0.03	0.55	0.03	0.05	0.03	0.13
and	0.10	0.08	0.07	0.10	0.12	0.09	0.04	0.15	0.12	0.09	0.04
to	0.11	0.11	0.09	0.09	0.15	0.08	0.04	0.07	0.15	0.08	0.03
my	0.19	0.03	0.05	0.04	0.03	0.29	0.01	0.02	0.03	0.29	0.01
crew	0.06	0.05	0.05	0.06	0.05	0.06	0.21	0.04	0.05	0.06	0.31

Attention

We use these scores to compute the attention.

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{Q}) = \text{softmax}(\frac{\mathbf{QK}^\intercal}{\sqrt{d_k}})\mathbf{V},$$

For *ship:*

```
attention_ship = (0.03 * embeddings_dict['i'] +
                  0.03 * embeddings_dict['must'] +
                  0.03 * embeddings_dict['go'] +
                  0.03 * embeddings_dict['back'] +
                  0.04 * embeddings_dict['to'] +
                  0.05 * embeddings_dict['my'] +
                  0.55 * embeddings_dict['ship'] +
                  0.03 * embeddings_dict['and'] +
                  0.05 * embeddings_dict['to'] +
                  0.03 * embeddings_dict['my'] +
                  0.13 * embeddings_dict['crew'])
```

where the *ship* vector received 13% of its value from *crew*

Code Example

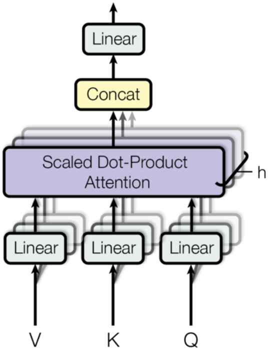
Experiment: Jupyter Notebook: `4.4-attention.ipynb`
(First part of the notebook)

Multihead Attention

This attention is preceded by dense layers:
If $\mathbf{X}$ represents complete input sequence (all the tokens), we have:

$$\begin{aligned} \mathbf{Q} &= \mathbf{X}\mathbf{W}_Q, \\ \mathbf{K} &= \mathbf{X}\mathbf{W}_K, \\ \mathbf{V} &= \mathbf{X}\mathbf{W}_V. \end{aligned}$$

And followed by another dense layer.
In addition, most architectures have parallel attentions, where the outputs (called heads) are concatenated (multihead)



From *Attention is all you need*, Vaswani et al. (2017)

Code Example

Keras has an implementation of this architecture with the `MultiHeadAttention()` layer.

Experiment: Jupyter Notebook: `4.4-attention.ipynb` (Second part of the notebook)

Transformers

Transformers are architectures, where:

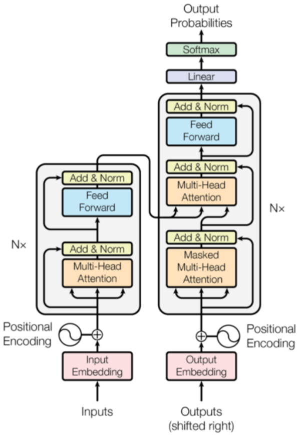
1. The first part of the layer is a multihead attention;
2. We reinject the input to the attention output in the form of an addition:

$$\mathbf{X} + \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{Q}).$$

   This operation is called a skip or residual connection, which improves stability.
3. The result in then normalized per instance, i.e. a unique sequence, defined as:

$$x_{i,j_{norm}} = \frac{x_{i,j} - \bar{x}_{i,.}}{\sigma_{x_{i,.}}}.$$

4. It is followed by dense layers.



Left part, from *Attention is all you need*, Vaswani et al. (2017)

Code Example

Experiment: Jupyter Notebook:
`chapter11_part03_transformer.ipynb` from Chollet's book, first part up to positional embeddings

Positional Embeddings

BERT (the first transformer, Devlin et al. (2019)) maps each token to three embedding vectors:

- the token embedding,
- the position of the token in the sentence (positional embeddings), and
- the segment embeddings (we will skip this part).

The three kinds of embeddings are learnable vectors.

In the BERT base version, each embedding vector has 768 dimensions.

Let us consider two sentences simplified from the *Odyssey:*

Tell me of that hero. Many cities did he visit.

Input:	[CLS]	Tell	me	of	that	hero	[SEP]	Many	cities	did	he	visit	[SEP]
Token	$\mathbf{E}_{\texttt{[CLS]}}$	$\mathbf{E}_{\text{tell}}$	$\mathbf{E}_{\text{me}}$	$\mathbf{E}_{\text{of}}$	$\mathbf{E}_{\text{that}}$	$\mathbf{E}_{\text{hero}}$	$\mathbf{E}_{\texttt{[SEP]}}$	$\mathbf{E}_{\text{many}}$	$\mathbf{E}_{\text{cities}}$	$\mathbf{E}_{\text{did}}$	$\mathbf{E}_{\text{he}}$	$\mathbf{E}_{\text{visit}}$	$\mathbf{E}_{\texttt{[SEP]}}$
Segment	$\mathbf{E}_{\text{A}}$	$\mathbf{E}_{\text{A}}$	$\mathbf{E}_{\text{A}}$	$\mathbf{E}_{\text{A}}$	$\mathbf{E}_{\text{A}}$	$\mathbf{E}_{\text{A}}$	$\mathbf{E}_{\text{A}}$	$\mathbf{E}_{\text{B}}$	$\mathbf{E}_{\text{B}}$	$\mathbf{E}_{\text{B}}$	$\mathbf{E}_{\text{B}}$	$\mathbf{E}_{\text{B}}$	$\mathbf{E}_{\text{B}}$
Position	$\mathbf{E}_{0}$	$\mathbf{E}_{1}$	$\mathbf{E}_{2}$	$\mathbf{E}_{3}$	$\mathbf{E}_{4}$	$\mathbf{E}_{5}$	$\mathbf{E}_{6}$	$\mathbf{E}_{7}$	$\mathbf{E}_{8}$	$\mathbf{E}_{9}$	$\mathbf{E}_{10}$	$\mathbf{E}_{11}$	$\mathbf{E}_{12}$

Code Example

Experiment: Jupyter Notebook:
`chapter11_part03_transformer.ipynb` from Chollet's book, second part from positional embeddings

As a personal work and to gain a deeper understanding, you can read a tutorial in PyTorch:
`https://nlp.seas.harvard.edu/2018/04/03/attention.html`

Training Transformers

Transformers, such as BERT, are often trained on masked language models with two tasks:

1. For a sentence, predict masked words: We replace 15% of the tokens with a specific mask token and we train the model to predict them. This is just a cloze test;
2. For a pair of sentences, predict if the second one is the successor of the first one;

Taking the two first sentences from the *Odyssey*:

> *Tell me, O Muse, of that ingenious hero who travelled far and wide after he had sacked the famous town of Troy.*
> *Many cities did he visit, and many were the nations with whose manners and customs he was acquainted;*

Masked language models

We add two special tokens: `[CLS]` at the start of the first sentence and `[SEP]` at the end of both sentences, and the token `[MASK]` to denote the words to predict.

We would have for the first task:

> *[CLS] Tell me, O Muse, of that [MASK] hero who travelled far and wide [MASK] he had sacked the [MASK] town of Troy. [SEP]*

For the second task, we would have as input:

> *[CLS] Tell me, O Muse, of that [MASK] hero who travelled far and wide [MASK] he had sacked the [MASK] town of Troy. [SEP] Many cities did he [MASK visit], and many were the [MASK nations] with whose manners [MASK and] customs he was acquainted; [SEP]*

where the prediction would return that the second sentence is the next one (as opposed to random sequences)

Model size

Transformers are trained on large corpora like the colossal clean crawled corpus (https://arxiv.org/abs/2104.08758) and encapsulate semantics found in text in the form of numerical matrices.

This results in large models (Devlin et al., 2019):

> *In this work, we denote the number of layers (i.e., Transformer blocks) as L, the hidden size as H, and the number of self-attention heads as A. We primarily report results on two model sizes: BERT$_{BASE}$ (L=12, H=768, A=12, Total Parameters=110M) and BERT$_{LARGE}$ (L=24, H=1024, A=16, Total Parameters=340M).*

Transformers can then act as pre-trained models for a variety of tasks.

See the list from Huggingface

Finally, an interesting reading: https://sayakpaul.medium.com/an-interview-with-colin-raffel-research-scientist-at-google-5

Optional Part

The rest of the slides is not part of the course

Machine Translation

Process of translating automatically a text from a source language into a target language

Started after the 2nd world war to translate documents from Russian to English

Early working systems from French to English in Canada

Renewed huge interest with the advent of the web

Google claims it has more than 500m users daily worldwide, with 103 languages.

Massive progress permitted by the neural networks

Corpora for Machine Translation

Initial ideas in machine translation: use bilingual dictionaries and formalize grammatical rules to transfer them from a source language to a target language.

Statistical machine translation:

1. Use very large bilingual corpora;
2. Align the sentences or phrases, and
3. Given a sentence in the source language, find the matching sentence in the target language.

Pioneered at IBM on French and English with Bayesian statistics.

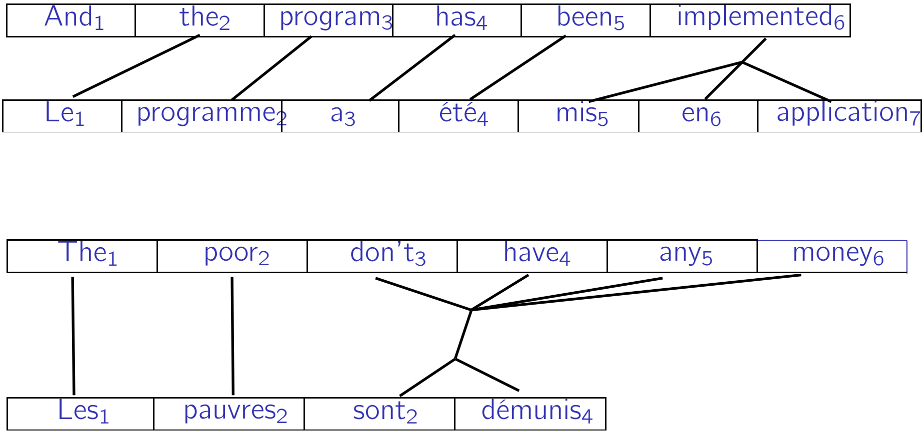
Neural nets are now dominant

Parallel Corpora (Swiss Federal Law)

German	French	Italian
Art. 35 Milchtransport	**Art. 35 Transport du lait**	**Art. 35 Trasporto del latte**
1 Die Milch ist schonend und hygienisch in den Verarbeitungsbetrieb zu transportieren. Das Transportfahrzeug ist stets sauber zu halten. Zusammen mit der Milch dürfen keine Tiere und milchfremde Gegenstände transportiert werden, welche die Qualität der Milch beeinträchtigen können.	1 Le lait doit être transporté jusqu'à l'entreprise de transformation avec ménagement et conformément aux normes d'hygiène. Le véhicule de transport doit être toujours propre. Il ne doit transporter avec le lait aucun animal ou objet susceptible d'en altérer la qualité.	1 Il latte va trasportato verso l'azienda di trasformazione in modo accurato e igienico. Il veicolo adibito al trasporto va mantenuto pulito. Con il latte non possono essere trasportati animali e oggetti estranei, che potrebbero pregiudicarne la qualità.

Alignment (Brown et al. 1993)

Canadian Hansard



Translations with RNNs

RNN can easily map sequences to sequences, where we have two lists: one for the source and the other for the target

y	Le	serveur	apporta	le	plat

x	The	waiter	brought	the	meal

The **x** and **y** vectors must have the same length.
In our case, *a apporté* is more frequent than *apporta*, but it breaks the alignment, as well as in many other examples

Translation with RNN

To solve the alignment problem, Sutskever al al. (2014) proposed (quoted from their paper, `https://arxiv.org/abs/1409.3215`):

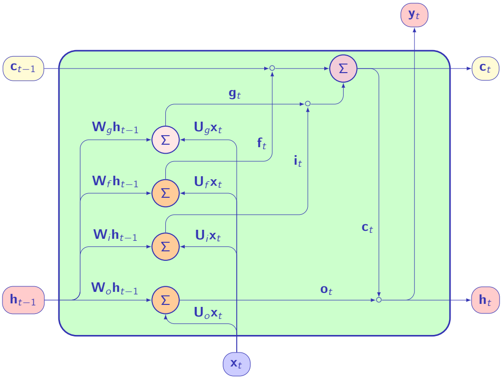
1. The simplest strategy for general sequence learning is to map the input sequence to a fixed-sized vector using one RNN, and then to map the vector to the target sequence with another RNN [...]
2. it would be difficult to train the RNNs due to the resulting long term dependencies [...]. However, the Long Short-Term Memory (LSTM) is known to learn problems with long range temporal dependencies.

Using the Hidden States

To solve the alignment problem, Sutskever al al. (2014) proposed (quoted from their paper, `https://arxiv.org/abs/1409.3215`):

1. LSTM estimate[s] the conditional probability $p(y_1, \dots, y_{T'} | x_1, \dots, x_T)$, where $(x_1, \dots, x_T)$ is an input sequence and $y_1, \dots, y_{T'}$ is its corresponding output sequence whose length T' may differ from T.
2. The LSTM computes this conditional probability by:
   1. First obtaining the fixed-dimensional representation v of the input sequence $(x1, \dots, xT)$ given by the last hidden state of the LSTM, (**encoder**) and then
   2. computing the probability of $y_1, \dots, y_{T'}$ with a standard LSTM-LM formulation whose initial hidden state is set to the representation v of $x_1, \dots, x_T$ (**decoder**)

The LSTM Architecture



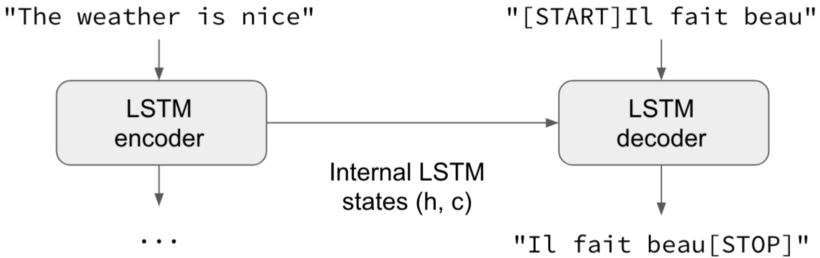
An LSTM unit showing the data flow, where $\mathbf{g}_t$ is the unit input, $\mathbf{i}_t$, the input gate, $\mathbf{f}_t$, the forget gate, and $\mathbf{o}_t$, the output gate. The activation functions have been omitted

Sequence-to-Sequence Translation

We follow and reuse: `https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in`
`html` and `https://keras.io/examples/nlp/lstm_seq2seq/` from Chollet.

1. We start with input sequences from a language (e.g. English sentences) and corresponding target sequences from another language (e.g. French sentences).
2. An encoder LSTM turns input sequences to 2 state vectors (we keep the last LSTM state and discard the outputs).
3. A decoder LSTM is trained to turn the target sequences into the same sequence but offset by one timestep in the future. This training process is called "teacher forcing" in this context.
4. It uses the state vectors from the encoder as initial state. Effectively, the decoder learns to generate `targets[t+1...]` given `targets[...t]`, conditioned on the input sequence.

Sequence-to-Sequence Translation



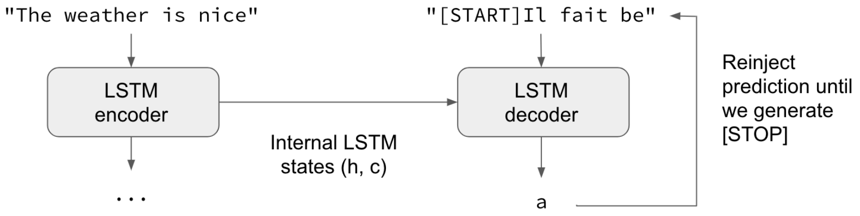
From https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in html

Inference

Following Chollet, in inference mode, to decode unknown input sequences, we:

- Encode the input sequence into state vectors
- Start with a target sequence of size 1 (just the start-of-sequence character)
- Feed the state vectors and 1-char target sequence to the decoder to produce predictions for the next character
- Sample the next character using these predictions (we simply use argmax).
- Append the sampled character to the target sequence
- Repeat until we generate the end-of-sequence character or we hit the character limit.

Sequence-to-Sequence Translation



From https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in
html

Improving the Architecture: Encoder-Decoder

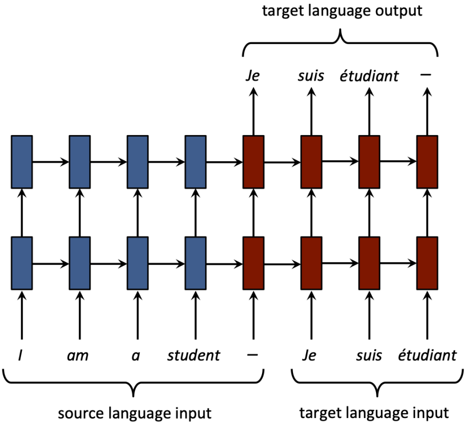


Figure 1: A simplified diagram of NMT.

From: Compression of Neural Machine Translation Models via Pruning by Abigail See, Minh-Thang Luong, and Christopher D. Manning

Improving the Architecture: Adding Attention

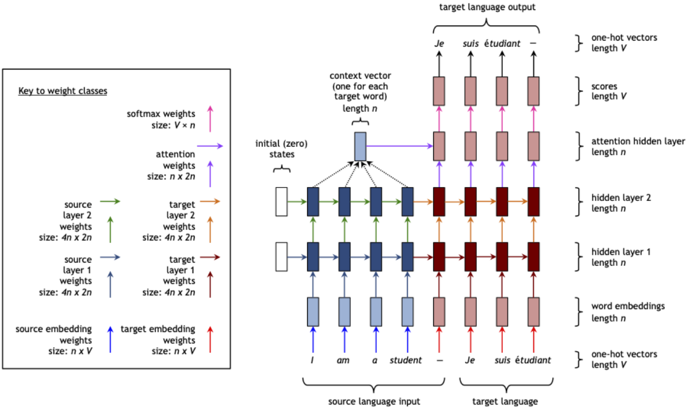


Figure 2: NMT architecture. This example has two layers, but our system has four. The different weight classes are indicated by arrows of different color (the black arrows in the top right represent simply choosing the highest-scoring word, and thus require no parameters). Best viewed in color.

From: Compression of Neural Machine Translation Models via Pruning by Abigail See, Minh-Thang Luong, and Christopher D. Manning