EDAN95
Applied Machine Learning
Lecture 7: Sequence Prediction

Pierre Nugues

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

November 23, 2020

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 three kinds of neural networks:

1. Feed forward
2. Recurrent
3. LSTM

In the laboratory assignment, you will use the two last ones.

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

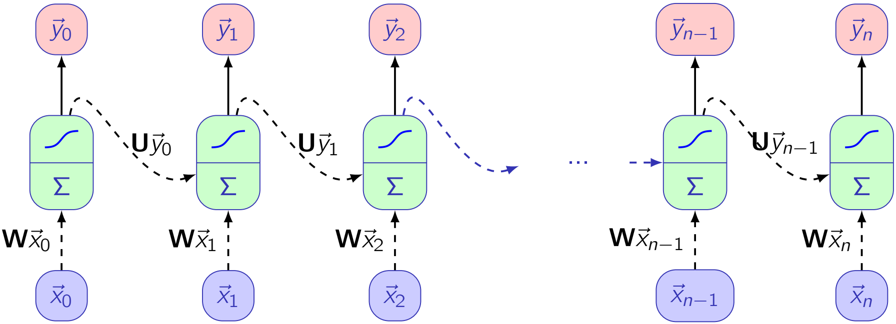
The Embedding Layer

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

if embedding_matrix is not None:
    model.layers[0].set_weights([embedding_matrix])
model.layers[0].trainable = True

model.compile(loss='sparse_categorical_crossentropy',
              optimizer=OPTIMIZER,
              metrics=['accuracy'])
model.summary()
model.fit(X, y, epochs=EPOCHS, batch_size=BATCH_SIZE)
```

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,
              input_length=None),
    SimpleRNN(100, return_sequences=True),
    # Bidirectional(SimpleRNN(100, return_sequences=True)),
    Dense(NB_CLASSES + 2, activation='softmax')])

model.layers[0].set_weights([embedding_matrix])
# The default is True
model.layers[0].trainable = True
```

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,
              input_length=None),
    Bidirectional(LSTM(100, return_sequences=True)),
    Dense(NB_CLASSES + 2, activation='softmax')])

model.layers[0].set_weights([embedding_matrix])
# The default is True
model.layers[0].trainable = True
```

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

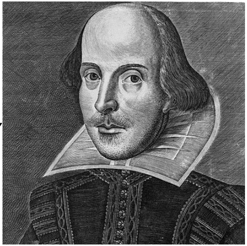
Word	POS	Group	Word	POS	Group
He	PRP	B-NP	*to*	TO	B-PP
reckons	VBZ	B-VP	*only*	RB	B-NP
the	DT	B-NP	*£*	#	I-NP
current	JJ	I-NP	*1.8*	CD	I-NP
account	NN	I-NP	*billion*	CD	I-NP
deficit	NN	I-NP	*in*	IN	B-PP
will	MD	B-VP	*September*	NNP	B-NP
narrow	VB	I-VP	*.*	.	O

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

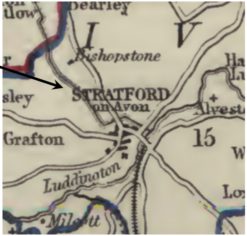
IOB Annotation for Named Entities

CoNLL 2002		CoNLL 2003			
Words	**Named entities**	**Words**	**POS**	**Groups**	**Named entities**
Wolff	B-PER	U.N.	NNP	I-NP	I-ORG
,	O	official	NN	I-NP	O
currently	O	Ekeus	NNP	I-NP	I-PER
a	O	heads	VBZ	I-VP	O
journalist	O	for	IN	I-PP	O
in	O	Baghdad	NNP	I-NP	I-LOC
Argentina	B-LOC	.	.	O	O
,	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				

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:

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

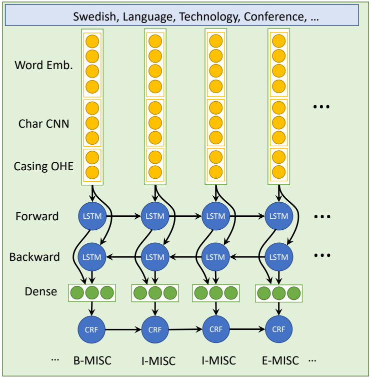
Evaluation

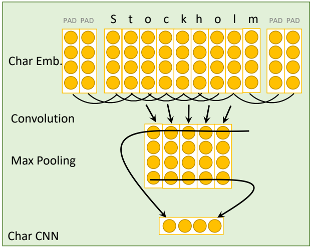
Accuracy, precision, and recall.
For noun groups with the predicted output:

Word	POS	Group	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	**B-NP**	*1.8*	CD	**B-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

Accuracy $= \frac{14}{16}$, recall $= \frac{2}{4} = 0.5$, precision $= \frac{2}{6} = 0.33$
harmonic mean $= 2 \times \frac{0.33 \times 0.5}{0.33 + 0.5} = 0.4$

The Architecture of a Full-Fledged Network





Courtesy: Marcus Klang. See also:

- Xuezhe Ma, Eduard Hovy, End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF, 2016, `https://arxiv.org/abs/1603.01354`
- Jason P.C. Chiu, Eric Nichols, Named Entity Recognition with Bidirectional LSTM-CNNs, 2016, `https://arxiv.org/abs/1511.08308`
- Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, Chris Dyer, Neural Architectures for Named Entity Recognition, 2016, `https://arxiv.org/abs/1603.01360`