EDAP30
Advanced Applied Machine Learning
Lecture 6: Recurrent Networks

Pierre Nugues

Pierre.Nugues@cs.lth.se

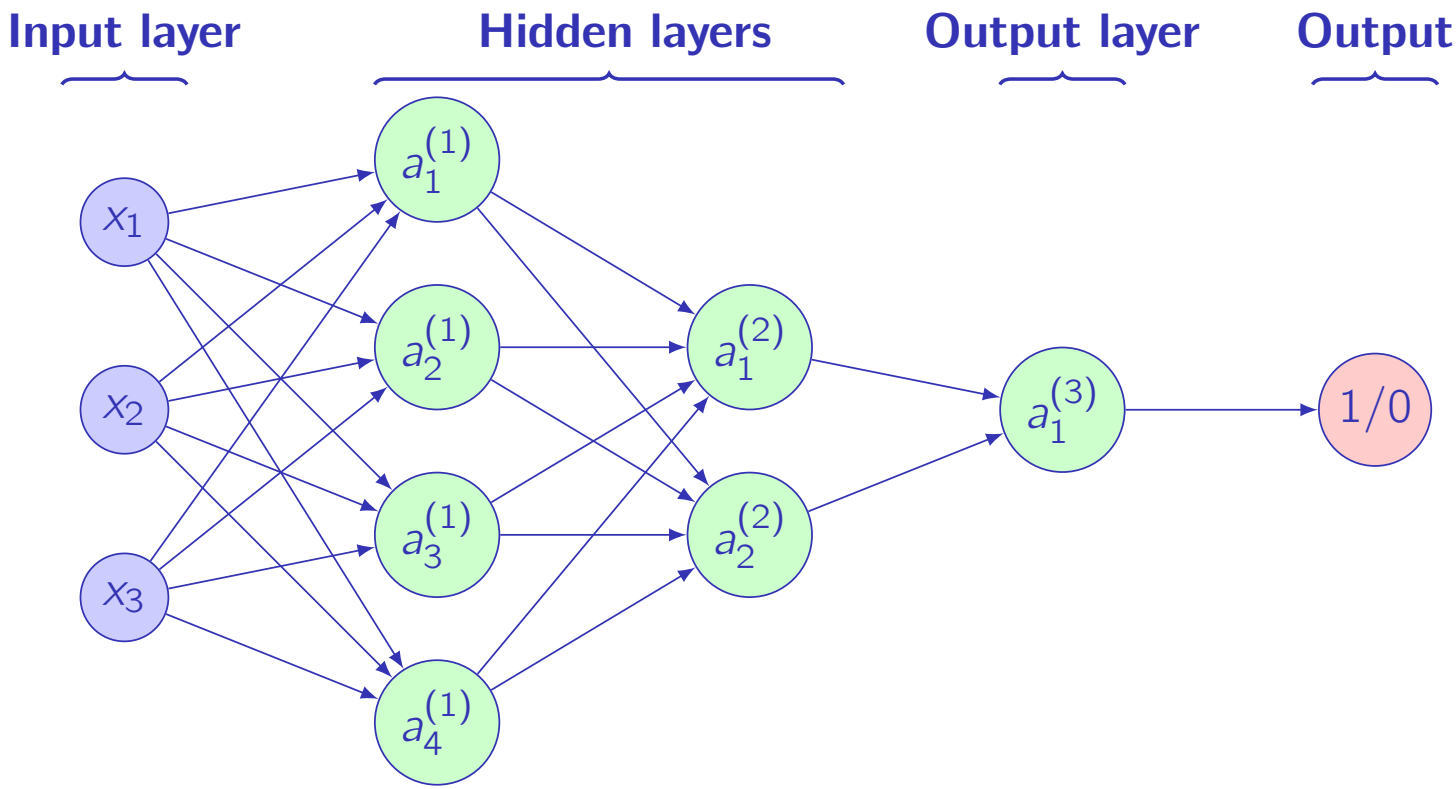
April 17, 2024

Recurrent Neural Networks

In feed-forward networks, predictions in a sequence of classifications are independent.
In many cases, given an input, the prediction also depends on the previous decision.
For instance, in weather forecast, if the input is the temperature and the output is rain/not rain, for a same temperature, it the previous output was rain, the next one is likely to be rain.
Recurrent neural networks (RNN) try to model these dependencies
In this lecture, we will examine recurrent neural networks for:

1. Categorization, i.e. given a sentence, predict its category, one output
2. Sequence annotation, i.e. given a sentence, predict a sequence of symbols, a sequence of outputs

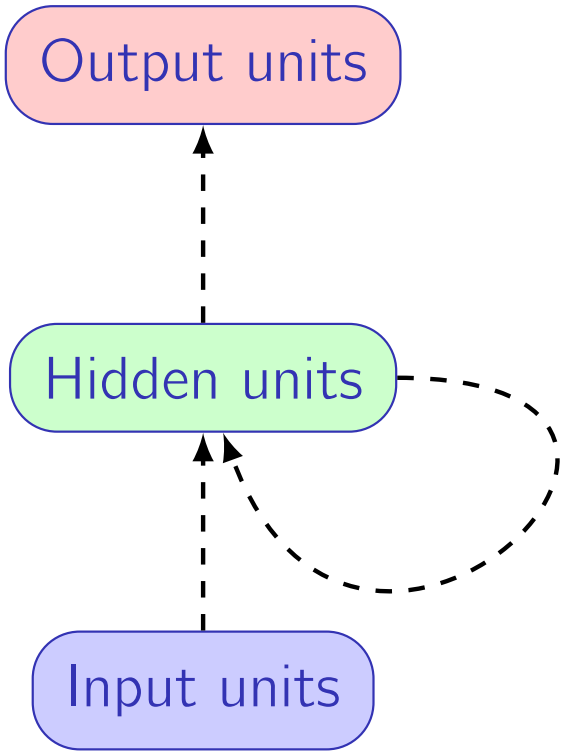
Feed Forward (Reminder)



For the first layer, we have:

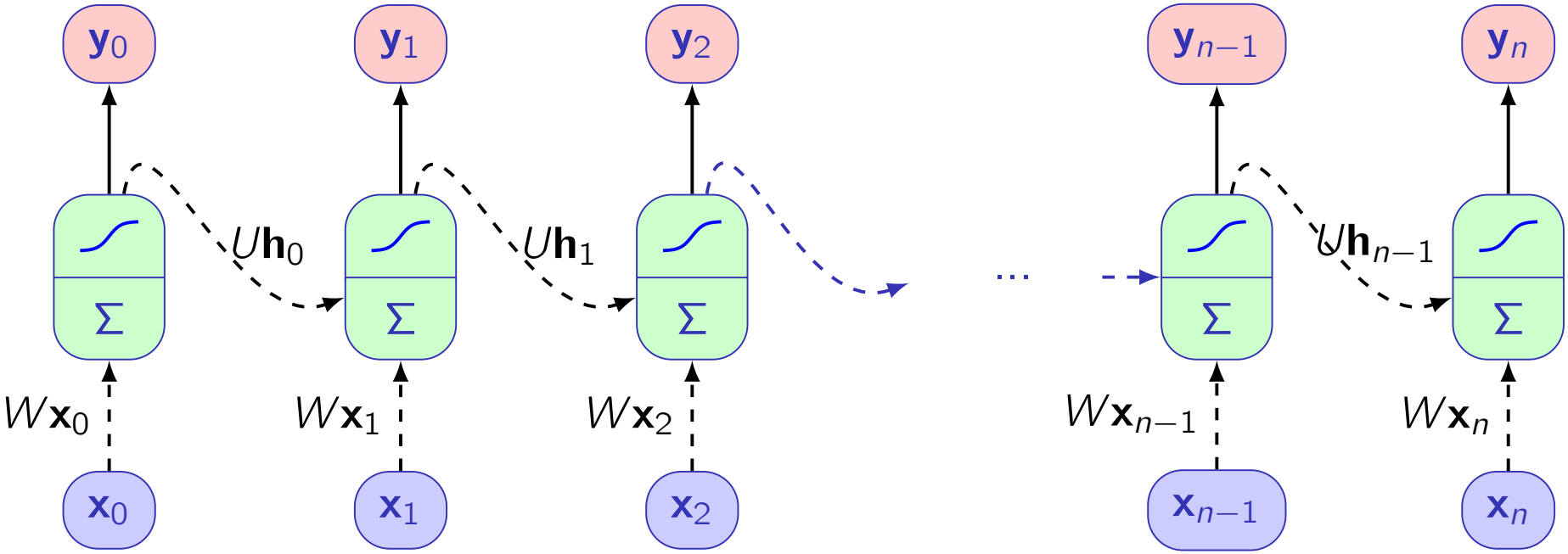
$$\text{activation}(W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}).$$

The RNN Architecture



A simple recurrent neural network; the dashed lines represent trainable connections.

The Unfolded RNN Architecture



The network unfolded in time. Equation used by implementations[1].

$$\mathbf{h}_{(t)} = \tanh(W\mathbf{x}_{(t)} + U\mathbf{h}_{(t-1)} + \mathbf{b})$$

[1] https://pytorch.org/docs/stable/generated/torch.nn.RNN.html

LSTMs

Simple RNNs use the previous output as input. They have then a very limited feature context.
Long short-term memory units (LSTM) are an extension to RNNs that can remember, possibly forget, information from longer or more distant sequences.
Given an input at index t, $\mathbf{x}_t$, a LSTM unit produces:

- A short term state, called $\mathbf{h}_t$ and
- A long-term state, called $\mathbf{c}_t$ or memory cell.

We use the short-term state, $\mathbf{h}_t$, to produce the output, i.e. $\mathbf{y}_t$ with a linear layer and a softmax activation; but both the long-term and short-term states are reused as inputs to the next unit.

LSTM Equations

A LSTM unit starts from a core equation that is identical to that of a RNN:

$$\mathbf{g}_t = \tanh(W_g \mathbf{x}_t + U_g \mathbf{h}_{t-1} + \mathbf{b}_g).$$

From the previous output and current input, we compute three kinds of filters, or gates, that will control how much information is passed through the LSTM cell
The two first gates, $\mathbf{i}$ and $\mathbf{f}$, defined as:

$$\begin{aligned} \mathbf{i}_t &= \text{activation}(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + \mathbf{b}_i), \\ \mathbf{f}_t &= \text{activation}(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + \mathbf{b}_f), \end{aligned}$$

model respectively how much we will keep from the base equation and how much we will forget from the long-term state.

LSTM Equations (II)

To implement this selective memory, we apply the two gates to the base equation and to the previous long-term state with the element-wise product (Hadamard product), denoted $\circ$, and we sum the resulting terms to get the current long-term state:

$$\mathbf{c}_t = \mathbf{i}_t \circ \mathbf{g}_t + \mathbf{f}_t \circ \mathbf{c}_{t-1}.$$

The third gate:

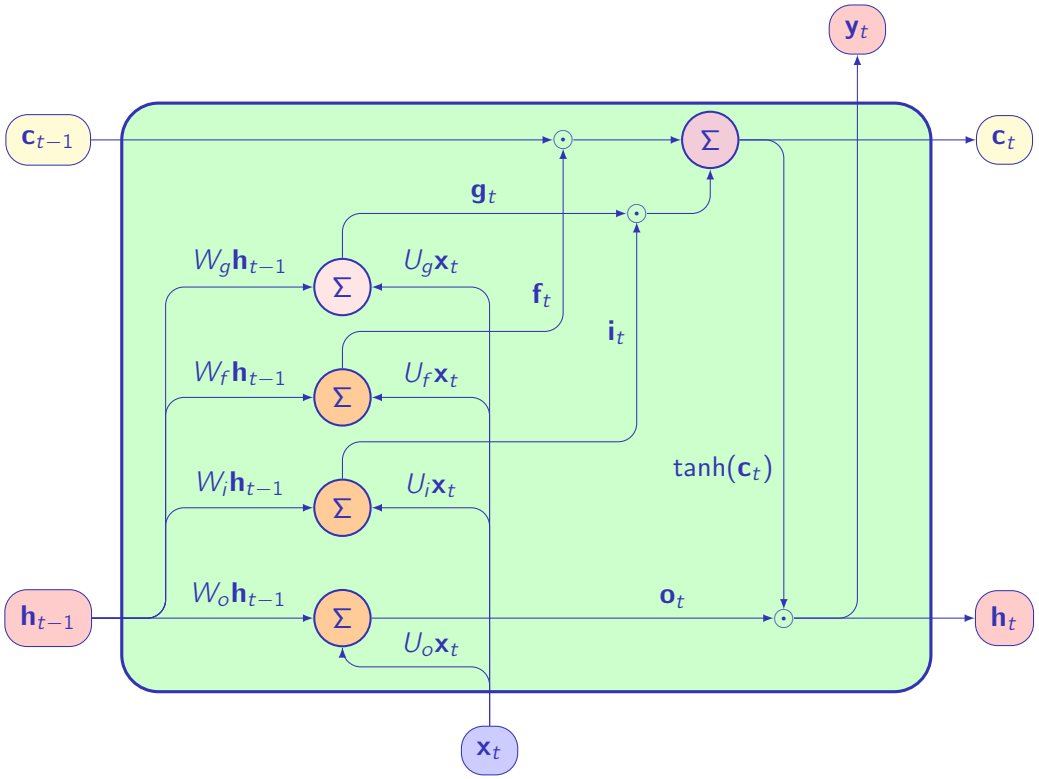
$$\mathbf{o}_t = \text{activation}(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + \mathbf{b}_o)$$

modulates the current long-term state to produce the output:

$$\mathbf{h}_t = \mathbf{o}_t \circ \tanh(\mathbf{c}_t).$$

The LSTM parameters are determined by a gradient descent.
See also:
`https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html`

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

Recurrent Networks for Classification

- We will now use a recurrent network to classify texts
- We will use the IMDB dataset of movie reviews annotated as positive or negative
- In a feed-forward network, we build a representation of the document using the vector space concepts (with tfidf parameters for instance)
- In a recurrent architecture, the words will go through the network and we will use the last output to classify a text
- As vectorization, we will first use a one-hot encoding and then GloVe

Building a LSTM with PyTorch

```
def __init__(self, lstm_units, nbr_classes, num_layers=1,
             bidi_lstm=False):
    super().__init__()
    self.dropout = nn.Dropout(DROPOUT)
    self.lstm = nn.LSTM(MAX_TOKENS + 2, lstm_units,
          num_layers=num_layers,
          dropout=DROPOUT, batch_first=True,
          bidirectional=bidi_lstm)
    if not bidi_lstm:
      self.fc = nn.Linear(lstm_units, nbr_classes)
    else:
      # twice the units if bidirectional
      self.fc = nn.Linear(2*lstm_units, nbr_classes)
```

Building a LSTM with PyTorch

```
def forward(self, sentence):
        embeds = F.one_hot(sentence,
                num_classes=MAX_TOKENS + 2).float()
        lstm_out, (h_n, c_n) = self.lstm(embeds)
        lstm_last = F.relu(h_n[-1])
        lstm_last = self.dropout(lstm_last)
        logits = self.fc(lstm_last)
        return logits
```

Code Example

Experiment: Jupyter Notebook:
https://github.com/pnugues/edap30/blob/main/programs/1-text-classification_lstm_one_hot.ipynb
The IMDB corpus with a LSTM and unit vectors

Using Embeddings

- Embeddings encapsulate knowledge: If the training procedure has seen *Denmark*, but not *Sweden*, the embeddings will be close and the prediction will be possible
- With one-hot vectors, *Denmark* and *Sweden* are orthogonal
- In addition, replacing one-hot vectors with embeddings will save space
- Training embeddings is unsupervised

Using GloVe Embeddings

We create a dictionary, where the keys are the words and the value, the embedding vector

```
file = 'glove.6B.50d.txt'

embeddings = {}
glove = open(file, encoding='utf8')
for line in glove:
        values = line.strip().split()
        word = values[0]
        vector = torch.FloatTensor(list(map(float, values[1:])
        embeddings[word] = vector
glove.close()
```

Initializing the Matrix

We create the embeddings matrix by using the GloVe embedding or a random vector, if not in GloVe

```
vocabulary_words = sorted(list(set(embedded_words + voc_train)))

embedding_matrix =\
   torch.rand((len(vocabulary_words) + 2, EMBEDDING_DIM))/10 - 0.05

out_of_embeddings = []
for word in vocabulary_words:
    if word in embeddings_dict:
        # If the words are in the embeddings, we fill them with a va
        embedding_matrix[word2idx[word]] = embeddings_dict[word]
    else:
        # Otherwise, it keeps a random value in the matrix
        # We store the out of vocabulary words
        out_of_embeddings += [word]
```

Building the Network

```
def __init__(self, embedding_matrix, lstm_units, nbr_classes,
    freeze_embs=True, num_layers=1, bidi_lstm=False):
        super().__init__()
        embedding_dim = embedding_matrix.size()[-1]
        self.embeddings = nn.Embedding.from_pretrained(
          embedding_matrix, freeze=freeze_embs, padding_idx=0)
        self.dropout = nn.Dropout(DROPOUT)
        self.lstm = nn.LSTM(embedding_dim, lstm_units,
                            num_layers=num_layers,
                            dropout=DROPOUT, batch_first=True,
                            bidirectional=bidi_lstm)
        if not bidi_lstm:
            self.fc = nn.Linear(lstm_units, nbr_classes)
        else:
            # twice the units if bidirectional
            self.fc = nn.Linear(2*lstm_units, nbr_classes)
```

Structure of a Network

```
def forward(self, sentence):
        embeds = self.embeddings(sentence)
        embeds = self.dropout(embeds)
        lstm_out, (h_n, c_n) = self.lstm(embeds)
        lstm_last = F.relu(h_n[-1])
        lstm_last = self.dropout(lstm_last)
        logits = self.fc(lstm_last)
        return logits
```

Code Example

Experiment: Jupyter Notebook,
`https://github.com/pnugues/edap30/blob/main/programs/2-text-classification_lstm_embs.ipynb`
The IMDB corpus with a LSTM and embeddings

Outline

So far, 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 see 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:

```
if SIMPLE_MODEL:
    model = nn.Sequential(nn.Linear(X_train.size()[1],
                                    NB_CLASSES))
else:
    model = nn.Sequential(
        nn.Linear(X_train.size()[1],
                  NB_CLASSES * 2),
        nn.ReLU(),
        nn.Dropout(0.2),
        nn.Linear(NB_CLASSES * 2, NB_CLASSES))
```

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: https://github.com/pnugues/edap30/blob/main/programs/3-pos-tagger_ff.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

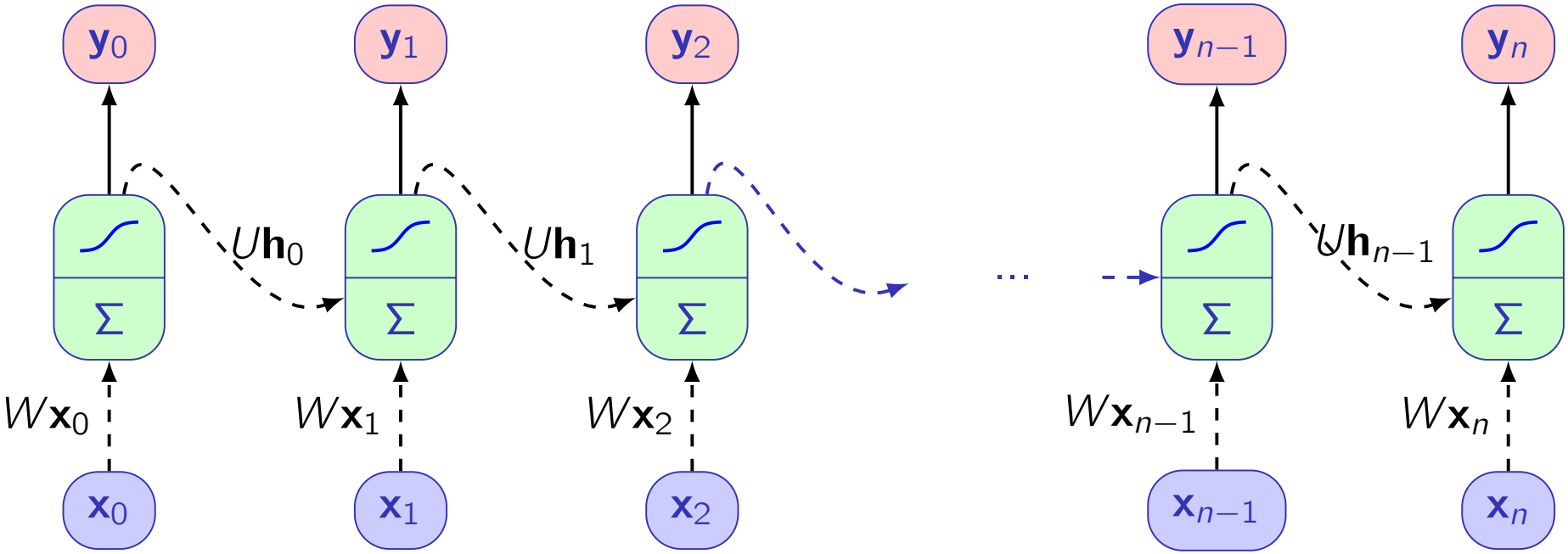
The Model with Embeddings

```
model = nn.Sequential(
    nn.Embedding.from_pretrained(embedding_matrix, freeze=False),
    nn.Flatten(),
    nn.Linear(5 * embedding_matrix.size()[1], NB_CLASSES)
)
```

Code Example

Jupyter Notebook: https://github.com/pnugues/edap30/blob/main/programs/4-pos-tagger_ff_embs.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_train_padded = pad_sequence(X_train_idx, batch_first=True)
Y_train_padded = pad_sequence(Y_train_idx, batch_first=True)

X_val_padded = pad_sequence(X_val_idx, batch_first=True)
Y_val_padded = pad_sequence(Y_val_idx, batch_first=True)
```

See: `https://pytorch.org/docs/stable/generated/torch.nn.utils.rnn.pad_sequence.html`
`pad_sequences` can have an argument that specifies the padding value `padding_value`
The padded sentences must have the same length in a batch. This is automatically computed by PyTorch

Recurrent Neural Networks (RNN)

```
def __init__(self, embedding_matrix, lstm_units, nbr_classes,
    freeze_embs=True, num_layers=1, bidi_lstm=False):
    super().__init__()
    embedding_dim = embedding_matrix.size()[-1]
    self.embeddings = nn.Embedding.from_pretrained(
      embedding_matrix, freeze=freeze_embs, padding_idx=0)
    self.dropout = nn.Dropout(DROPOUT)
    self.lstm = nn.LSTM(embedding_dim, lstm_units,
                num_layers=num_layers, dropout=DROPOUT,
                batch_first=True, bidirectional=bidi_lstm)
    if not bidi_lstm:
        self.fc = nn.Linear(lstm_units, nbr_classes)
    else:
        # twice the units if bidirectional
        self.fc = nn.Linear(2*lstm_units, nbr_classes)
```

Recurrent Neural Networks (RNN)

```
def forward(self, sentence):
    embeds = self.embeddings(sentence)
    embeds = self.dropout(embeds)
    lstm_out, _ = self.lstm(embeds)
    lstm_out = F.relu(lstm_out)
    lstm_out = self.dropout(lstm_out)
    logits = self.fc(lstm_out)
    return logits
```

Code Example

Jupyter Notebook: https://github.com/pnugues/edap30/blob/main/programs/5-pos-tagger_lstm.ipynb

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

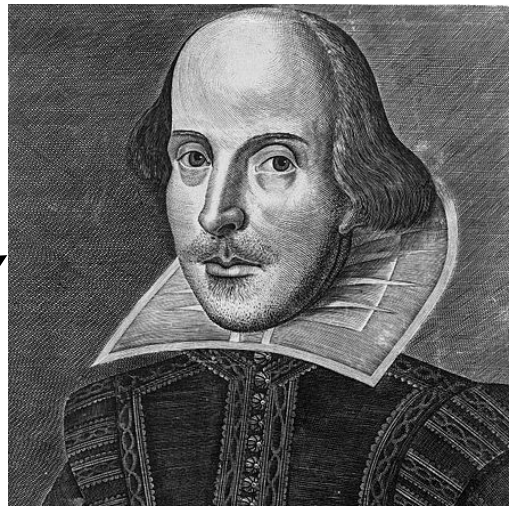
Noun groups (NP) are in red and verb groups (VP) are in blue.

IOB Annotation for Named Entities

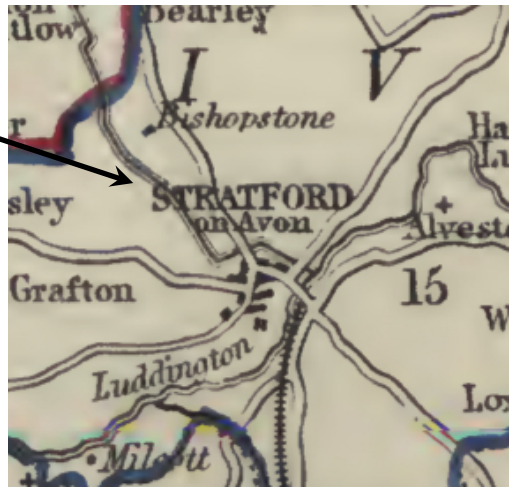
CoNLL 2002	
Words	**Named entities**
Wolff	B-PER
,	O
currently	O
a	O
journalist	O
in	O
Argentina	B-LOC
,	O
played	O
with	O
Del	B-PER
Bosque	I-PER
in	O
the	O
final	O
years	O
of	O
the	O
seventies	O
in	O
Real	B-ORG
Madrid	I-ORG
.	O

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

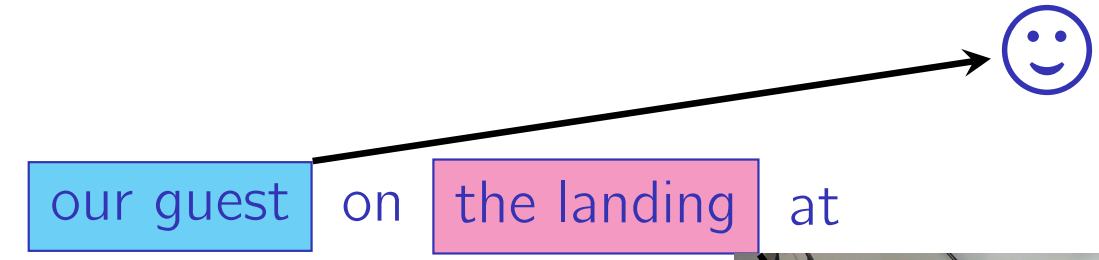
Named Entities: Proper Nouns



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



Others Entities: Common Nouns



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 $\frac{|A \cup D|}{|P \cup N|}$.

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 = $2 \times \frac{0.33 \times 0.5}{0.33 + 0.5} = 0.4$