Artificial Intelligence
EDAP01
Lecture 11.2: Natural Language for Communication

Pierre Nugues

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

February 26, 2025

Neural Networks and NLP

Many NLP tasks involve classifiers, more and more relying on neural networks.
Example with part-of-speech tagging.
The part of speech of 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

Features

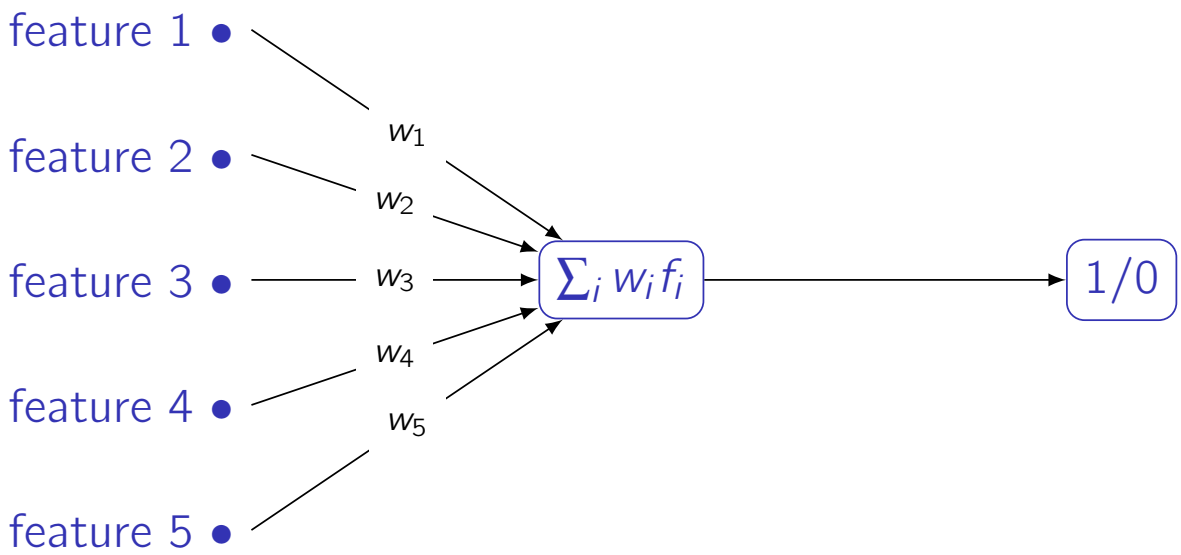
To train and apply the model, the tagger extracts a set of features.
For part-of-speech tagging, we use the surrounding words, for example, a sliding window spanning five words and centered on the current word.
We then associate the feature vector $(w_{i-2}, w_{i-1}, w_i, w_{i+1}, w_{i+2})$ with the part-of-speech tag t_i at index i:

1. (paid, a, visit, to, a) –> not verb
2. (went, to, visit, a, friend) –> verb

The words are encoded using unit vectors: one-hot encoding

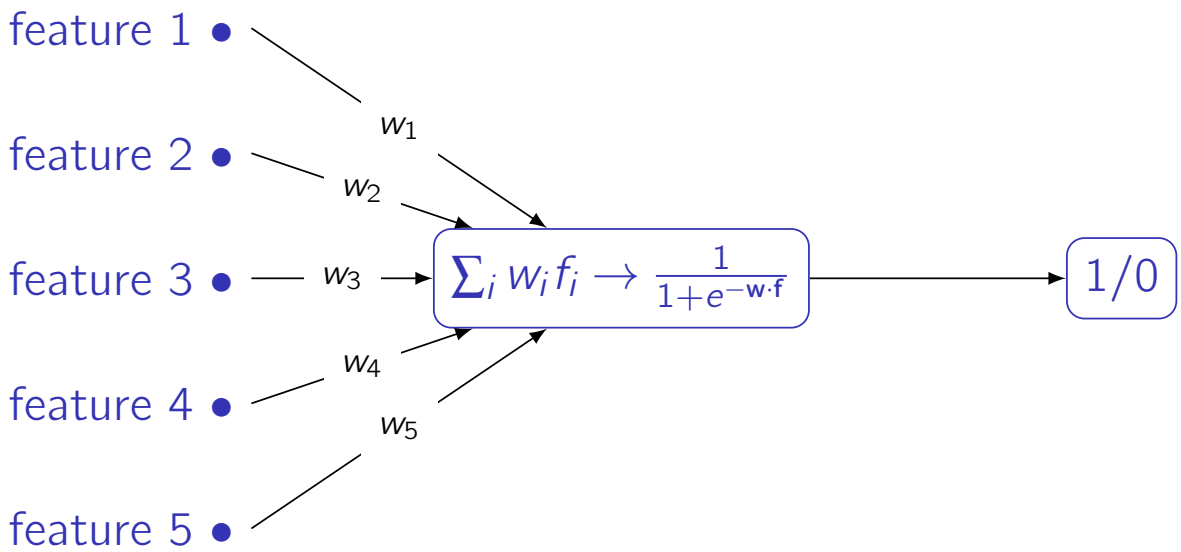
Neural Networks: Representation

The base network: An input layer and an output layer (perceptron):



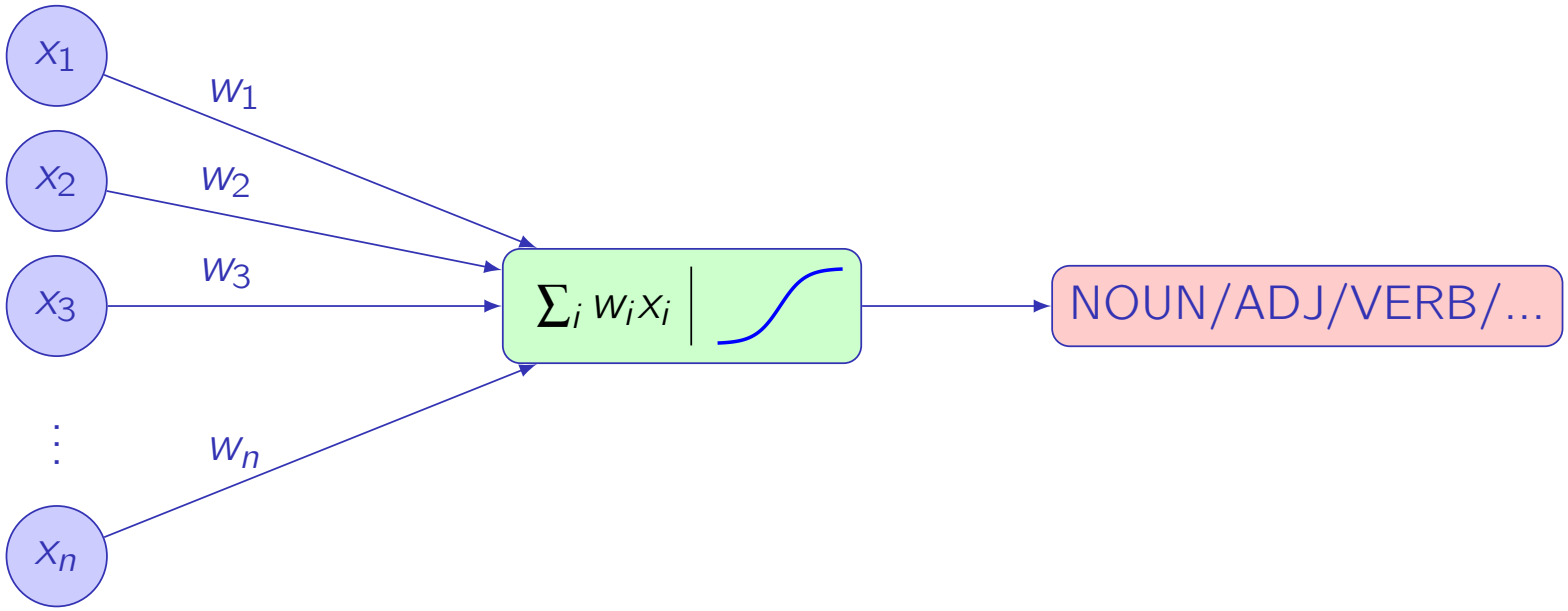
Neural Networks: Activation Function

And logistic regression:



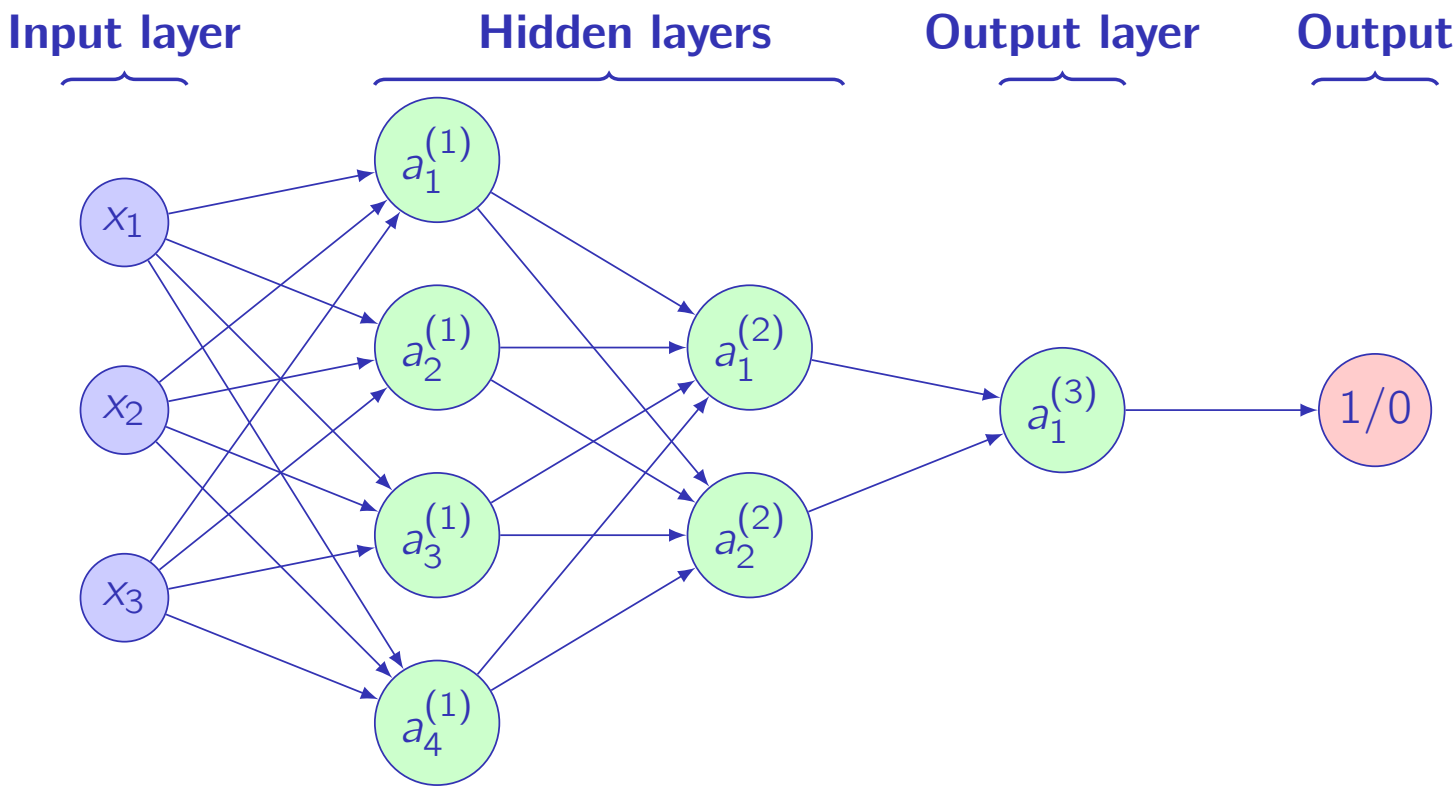
The logistic function is the activation function of the node
Demonstration: `http://playground.tensorflow.org/`

Feed Forward (Multinomial)



Softmax

Feed Forward (Multilayer)



For the first layer, we have:

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

Embeddings

One-hot encoding is impractical for large vocabularies
We need to use dense numerical vectors instead: vector space dimension ranging from 10 to 1000
In most cases, the input consists of dense numerical vectors: the embeddings
Dense vectors are called **embeddings**
Many ways to create such embeddings: for instance word2vec and GloVe

Word2vec Embeddings

word2vec in two forms: CBOW and skipgrams.
CBOW's aim is to predict a word given its surrounding context:

I went to —— a friend

The setup is similar to fill-the-missing-word questionnaires.
The missing word is called the focus word, here *visit*
CBOW's architecture is simply a multinomial logistic regression
The word inputs are trainable dense vectors, where each word is associated with a vector
The word vectors, the embeddings, are trained on a corpus

Word2vec Example

Using contexts of five words and training sentences such as:

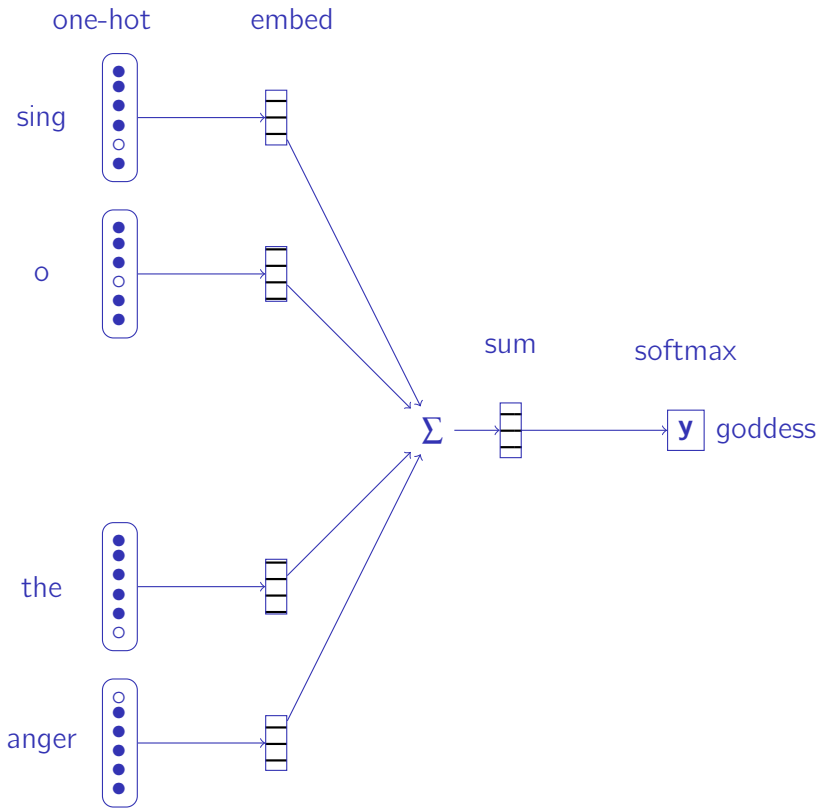
Sing, O goddess, the anger of Achilles son of Peleus,

we generate a training set of contexts deprived of their focus word (X) and the focus word to predict ($\mathbf{y}$):

$$X = \begin{bmatrix} \text{sing} & \text{o} & \text{the} & \text{anger} \\ \text{o} & \text{goddess} & \text{anger} & \text{of} \\ \text{goddess} & \text{the} & \text{of} & \text{achilles} \\ \text{the} & \text{anger} & \text{achilles} & \text{son} \\ \text{anger} & \text{of} & \text{son} & \text{of} \\ \text{of} & \text{achilles} & \text{of} & \text{peleus} \end{bmatrix}; \mathbf{y} = \begin{bmatrix} \text{goddess} \\ \text{the} \\ \text{anger} \\ \text{of} \\ \text{achilles} \\ \text{son} \end{bmatrix}$$

CBOW Architecture

We train a neural network to get the CBOW embeddings: N dimension of the embeddings, V size of the vocabulary. First matrix (V, N), second (N, V).



Sequence Annotation

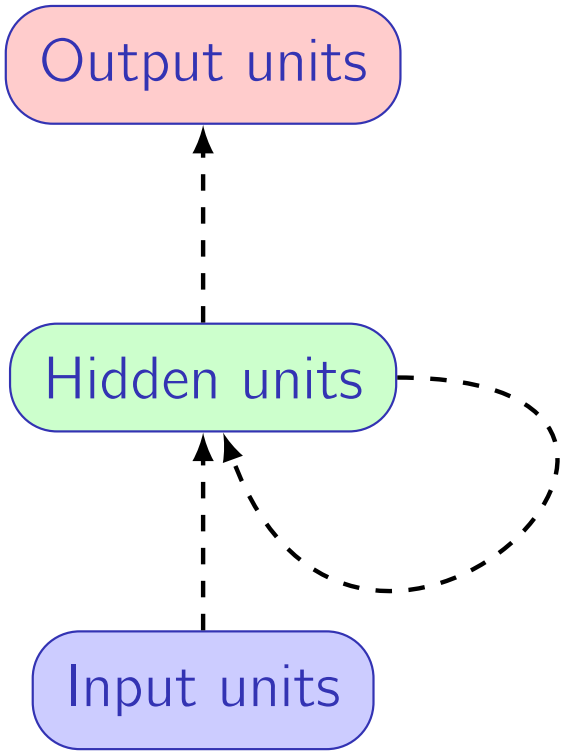
Feed forward networks are not handy when we deal with sequences, such as part-of-speech tagging
Recurrent architectures are better fit for that
We need to build two lists: one for the input and the other for the output

y	DET	NOUN	VERB	DET	NOUN

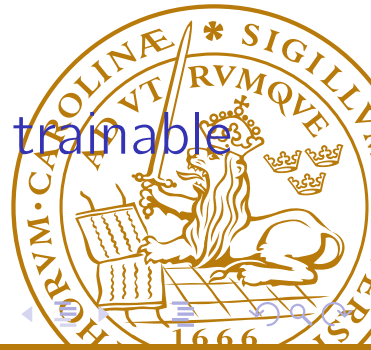
x	The	waiter	brought	the	meal

The input words are represented by their embeddings rather than with a one-hot encoding

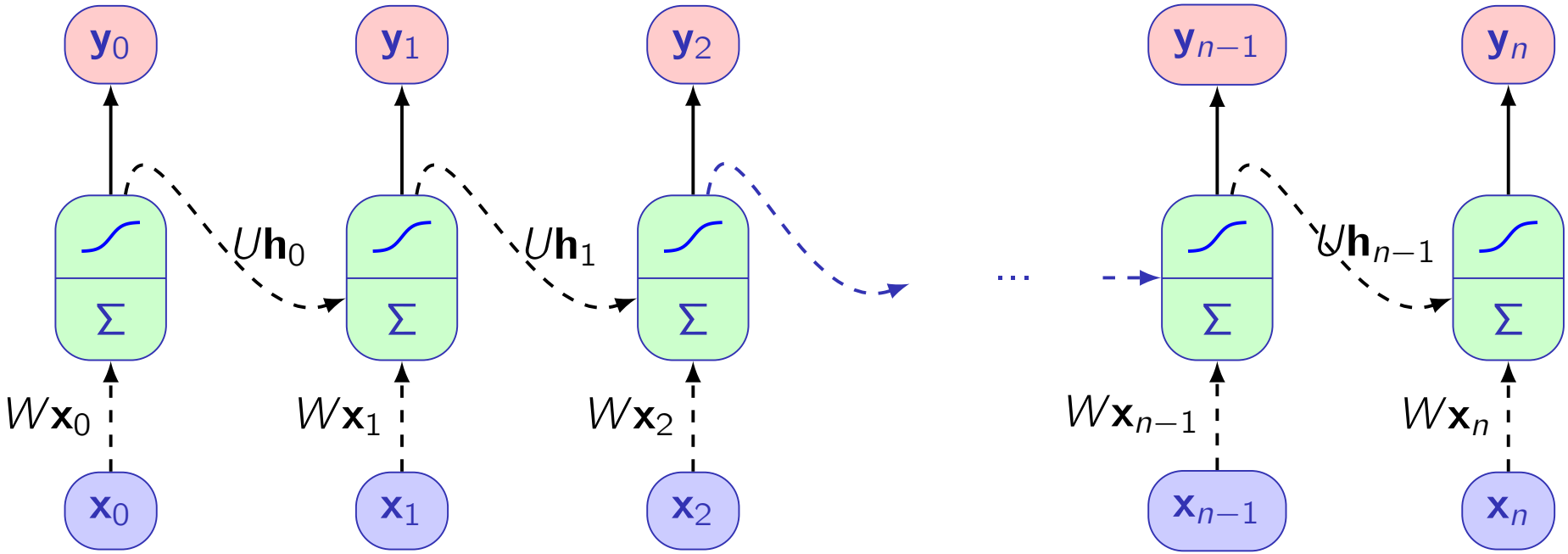
Recurrent Neural Networks



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].

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

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

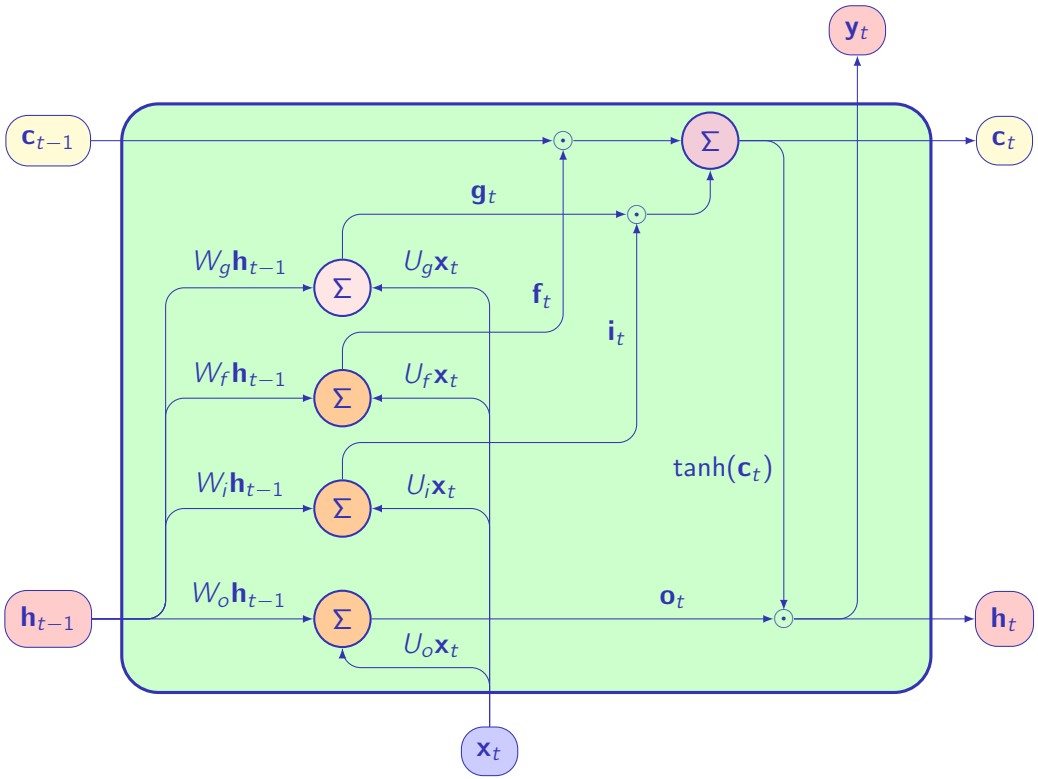
LSTMs

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.

The short-term state, $\mathbf{h}_t$, is the unit output, i.e. $\mathbf{y}_t$; but both the long-term and short-term states are reused as inputs to the next unit.

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

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.

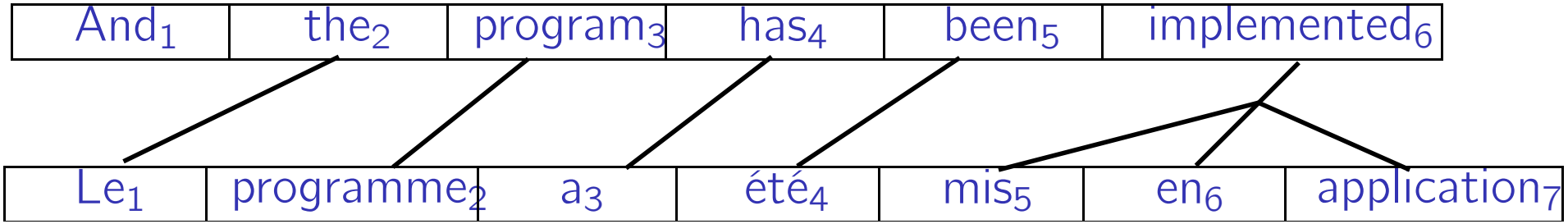
Pioneered at IBM on French and English with Bayesian statistics.
Neural nets are now dominant

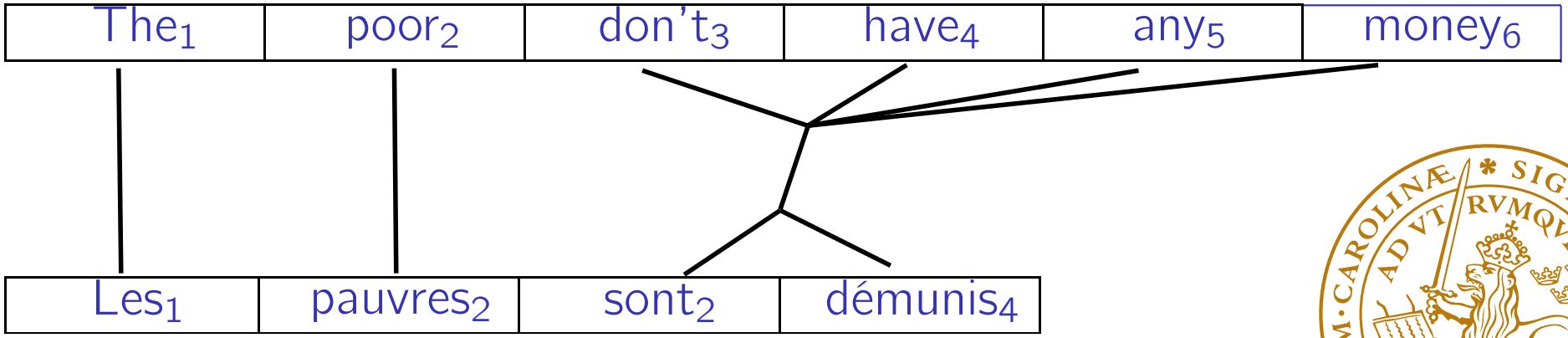
Parallel Corpora (Swiss Federal Law)

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.
We need a more flexible way to generate the target sentence.

Text Generation with Language Models

Using a n-gram language model, we can generate a sequence of words.
Starting from a first word, w_1, we extract the conditional probabilities: $P(w_2|w_1)$.
We could take the highest value, but it would always generate the same sequence.
Instead, we will draw our words from a multinomial distribution using `np.random.multinomial()`.
Given a probability distribution, this function draws a sample that complies the distribution.
Having, $P(want|I) = 0.5$, $P(wish|I) = 0.3$, $P(will|I) = 0.2$, the function will draw wish 30% of the time.

Code Example

Generating sequences with Bayesian probabilities
Jupyter Notebooks: `https://github.com/pnugues/edap01/blob/master/programs/5.7-generation.ipynb`

Generating Character Sequences with LSTMs

In the previous example, we used words. We can use characters instead.
We also used Bayesian probabilities. We can use LSTMs instead.
This is the idea of Chollet's program, pages 272-278.
X consists of sequences of 60 characters with a step of 3 characters
y is the character following the sequence
Let us use this excerpt:

is there not ground for suspecting that all philosophers

and 10 characters, where ␣ marks a space:

$$X = \begin{bmatrix} \text{i} & \text{s} & \text{␣} & \text{t} & \text{h} & \text{e} & \text{r} & \text{e} & \text{␣} & \text{n} \\ \text{t} & \text{h} & \text{e} & \text{r} & \text{e} & \text{␣} & \text{n} & \text{o} & \text{t} & \text{␣} \\ \text{r} & \text{e} & \text{␣} & \text{n} & \text{o} & \text{t} & \text{␣} & \text{g} & \text{r} & \text{o} \\ \text{n} & \text{o} & \text{t} & \text{␣} & \text{g} & \text{r} & \text{o} & \text{u} & \text{n} & \text{d} \\ \text{␣} & \text{g} & \text{r} & \text{o} & \text{u} & \text{n} & \text{d} & \text{␣} & \text{f} & \text{o} \end{bmatrix}; \mathbf{y} = \begin{bmatrix} \text{o} \\ \text{g} \\ \text{u} \\ \text{␣} \\ \text{r} \end{bmatrix}$$

Generating Character Sequences with LSTMs

In addition, Chollet uses a "temperature" function to transform the probability distribution: sharpen or damp it: $\exp(\frac{\log(x)}{temp}) = x^{\frac{1}{temp}}$

```
def sample(preds, temperature=1.0):
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds) / temperature
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    probas = np.random.multinomial(1, preds, 1)
    return np.argmax(probas)
```

with the input `[0.2, 0.5, 0.3]`, we obtain:

- Temperature = 2, `[0.26275107 0.41544591 0.32180302]`
- Temperature = 1, `[0.2 0.5 0.3]`
- Temperature = 0.5 `[0.10526316 0.65789474 0.23684211]`
- Temperature = 0.2 `[0.00941176 0.91911765 0.07147059]`

Code Example

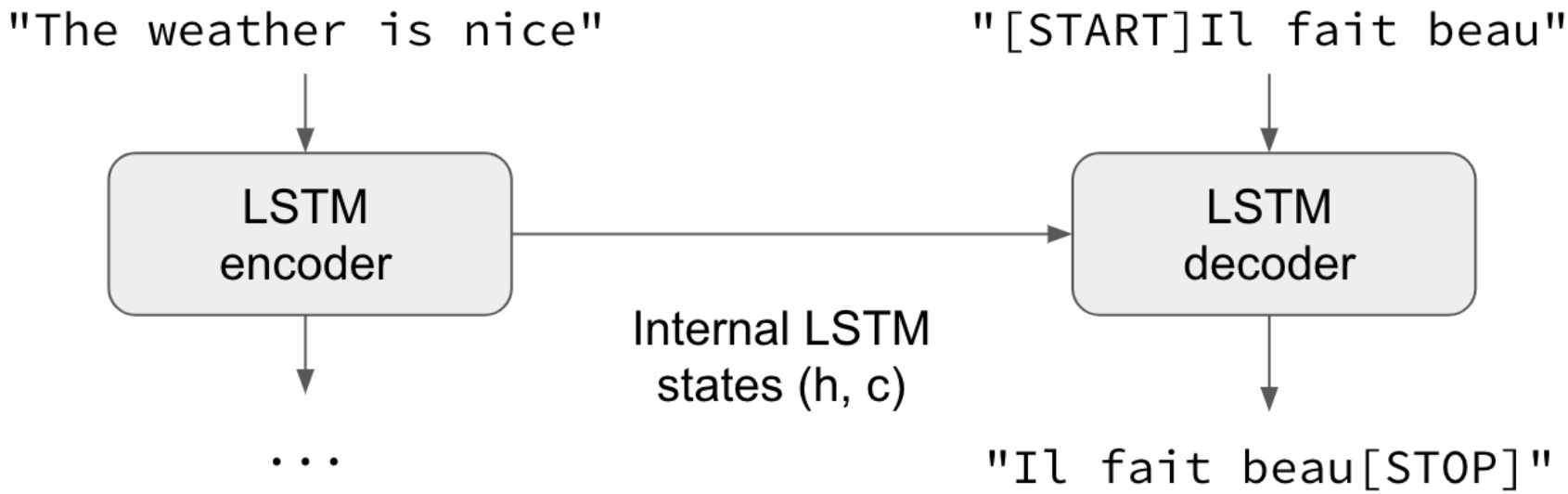
From Chollet's github repository:
Jupyter Notebooks: `https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/first_edition/8.1-text-generation-with-lstm.ipynb`

Translation with LSTMs: 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 $p(y_1, ..., y_{T'}|x_1, ..., x_T)$, where $(x_1, ..., x_T)$ is an input sequence and $y_1, ..., y_{T'}$ 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 $(x1, ..., xT)$ given by the last hidden state of the LSTM, (**encoder**) and then
   2. computing the probability of $y_1, ..., y_{T'}$ with a standard LSTM-LM formulation whose initial hidden state is set to the representation v of $x_1, ..., x_T$ (**decoder**)

Sequence-to-Sequence Translation



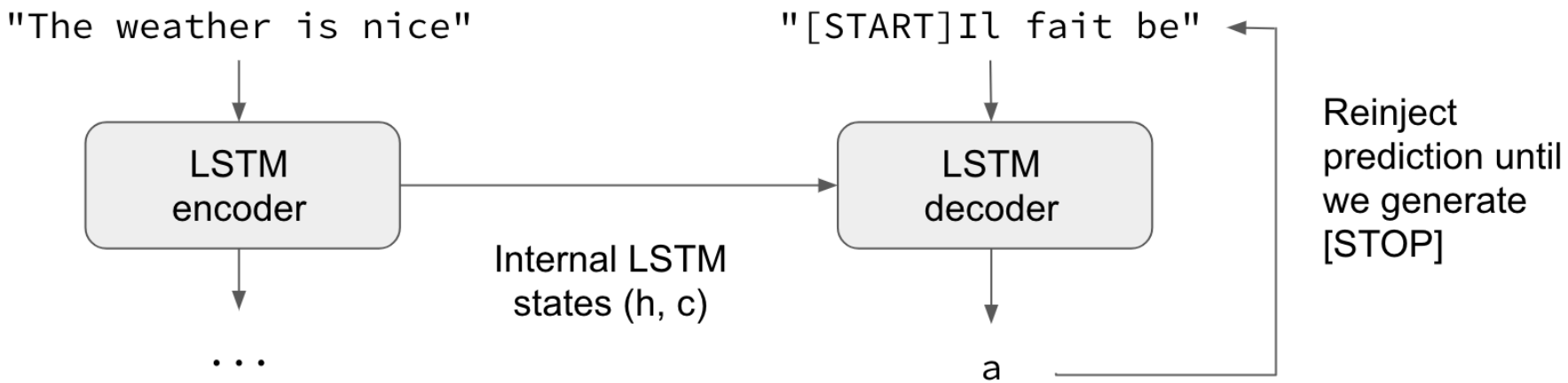
From https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-html

Inference

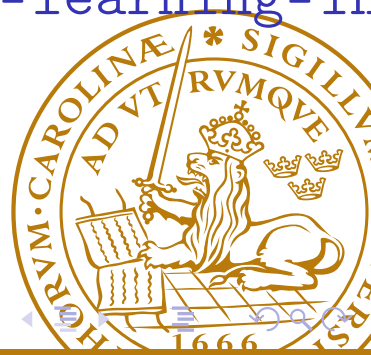
Following Chollet, in inference mode, to decode unknown input sequences, we:

- 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

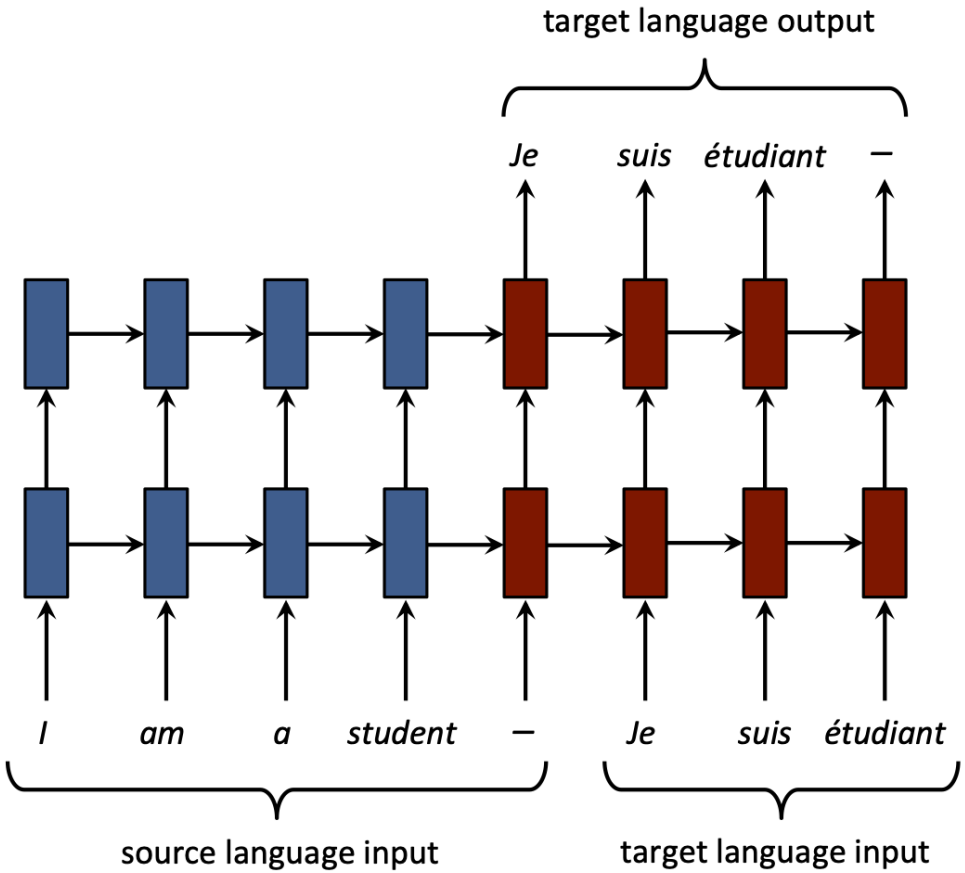


Figure 1: A simplified diagram of NMT.

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

Improving the Architecture: Adding Attention

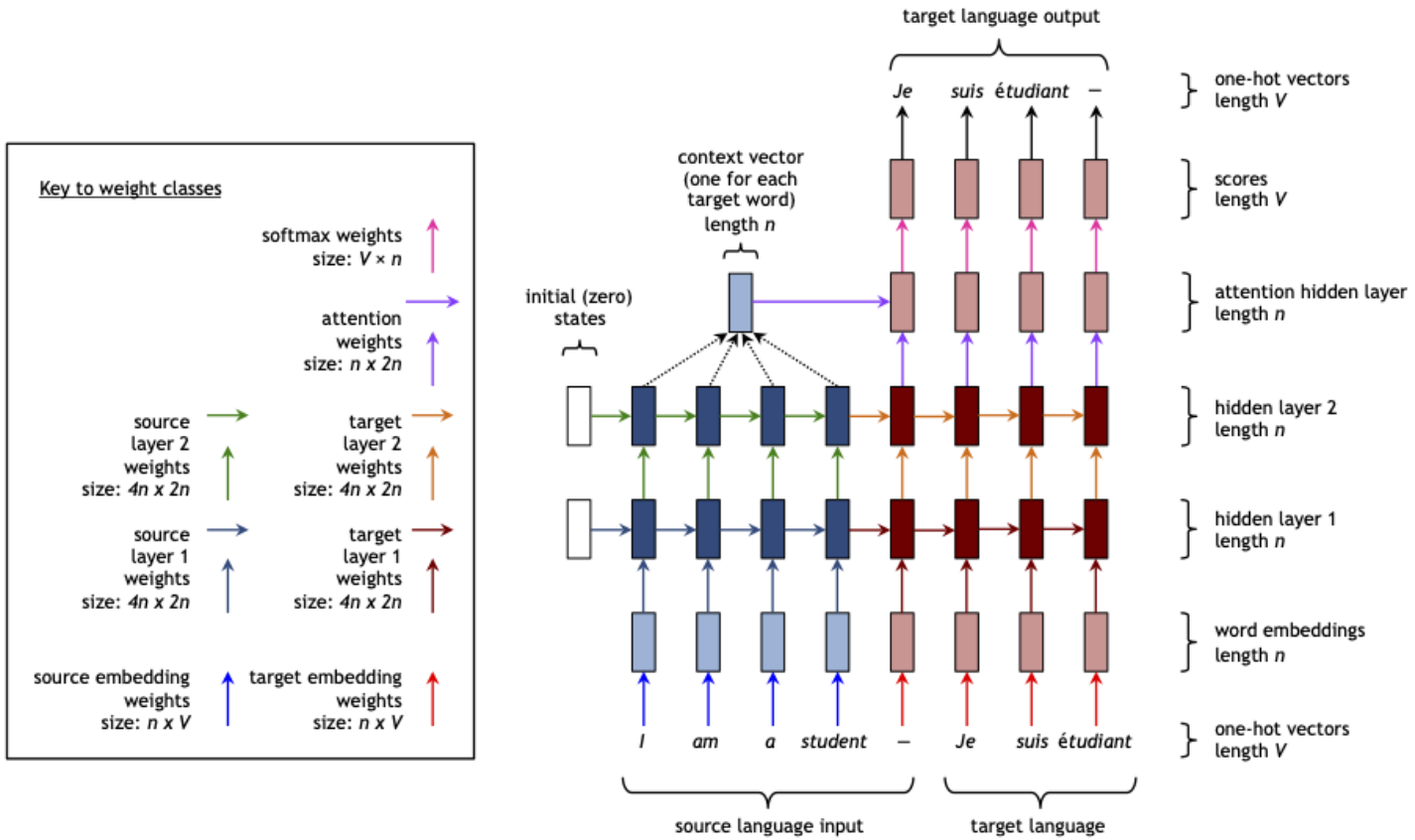


Figure 2: NMT architecture. This example has two layers, but our system has four. The different weight classes are indicated by arrows of different color (the black arrows in the top right represent simply choosing the highest-scoring word, and thus require no parameters). Best viewed in color.

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

Transformers

An architecture proposed in 2018
Original transformers consist of two parts: An encoder and a decoder
Both are feed-forward networks and contain an **attention** mechanism
Attention is a simple matrix product that can store semantic relations
We train the matrices on very large corpora
Sometimes marketed as the ImageNet moment (See `https://ruder.io/nlp-imagenet/`)

Self-Attention

In the paper *Attention is all you need*, Vaswani et al. (2017) the input words are represented as dense vectors
We compute the attention this way:

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

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

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

also defined as

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

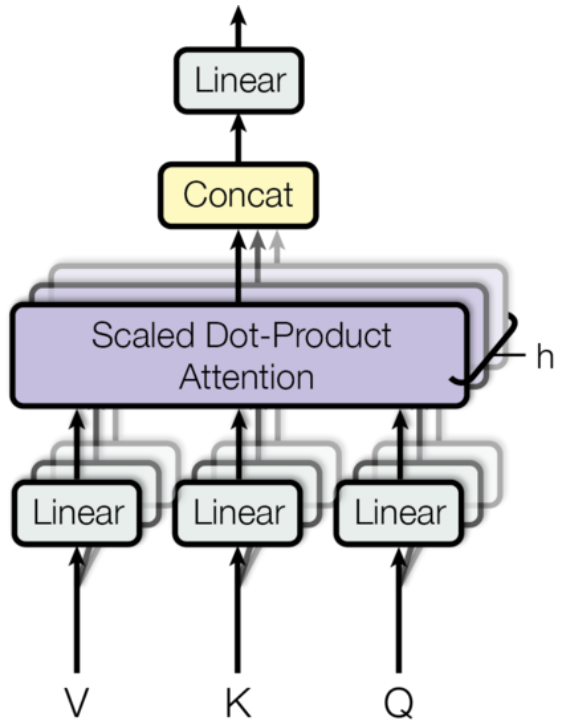
in physics.

Multihead Attention

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

$$\begin{aligned} Q &= XW_Q, \\ K &= XW_K, \\ V &= XW_V. \end{aligned}$$

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)

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:

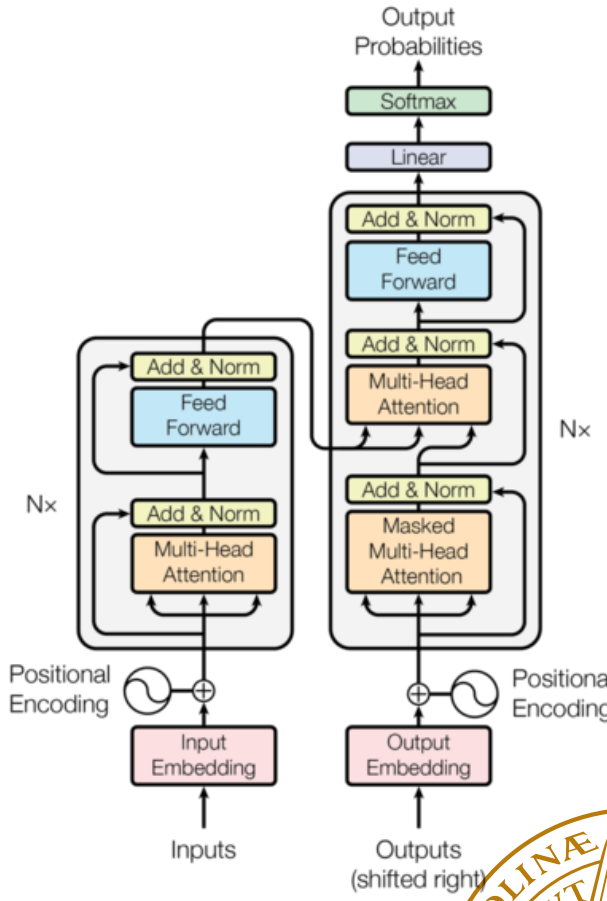
$$X + \text{Attention}(Q, K, Q).$$

   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)

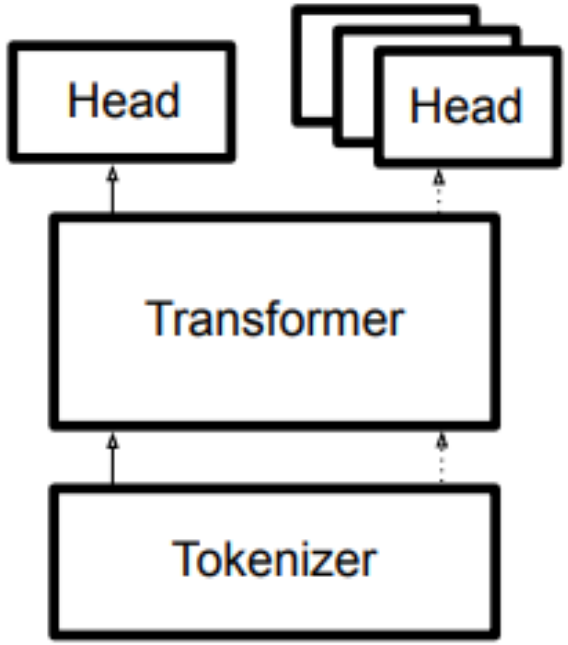
Training Transformers

Encoders, such as BERT, are often trained on masked language models:

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. Taking the first sentence 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.
3. We would have to find:
   [CLS] Tell me, O Muse, of that [MASK] hero who travelled far and wide [MASK] he had sacked the [MASK] town of Troy. [SEP]

Applying Transformers

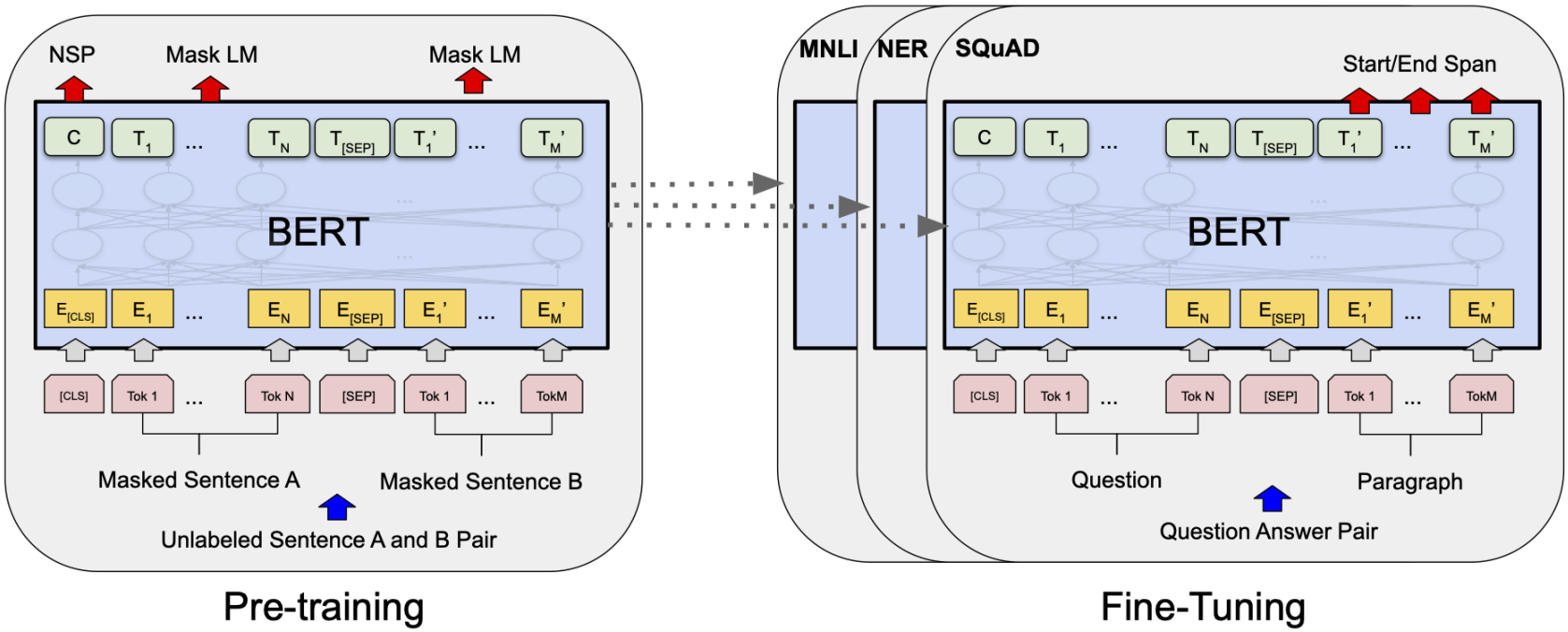
- Matrices in the transformer architecture encapsulate massive knowledge from text.
- We can apply them to tasks beyond what they have been trained for (masked language model)
- We use them as pretrained models and fine-tune a so-called dedicated *head*.
- The simplest head is a logistic regression



Picture from Wolf et al., Transformers: State-of-the-art Natural Language Processing, EMNLP Demos 2020.

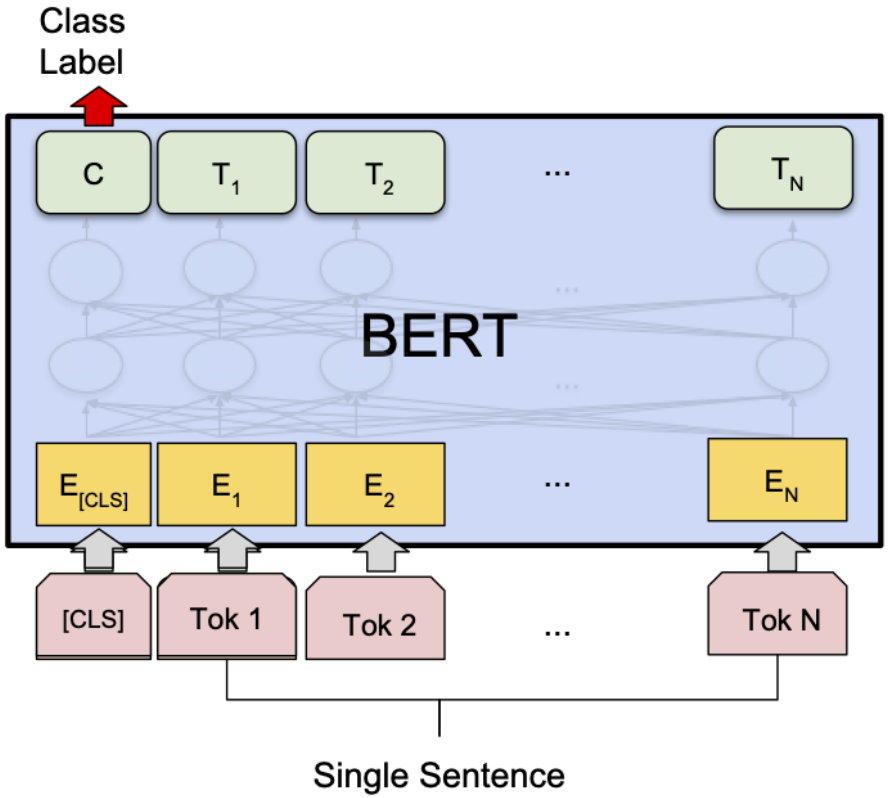
Transfer Learning

Transfer learning consists of a costly **pretraining** step and an adaptation to applications called **fine-tuning**.



from Devlin et al., *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019. Note the picture comes from the second version of the paper from 2019

Application: Sentence Classification



(b) Single Sentence Classification Tasks: SST-2, CoLA

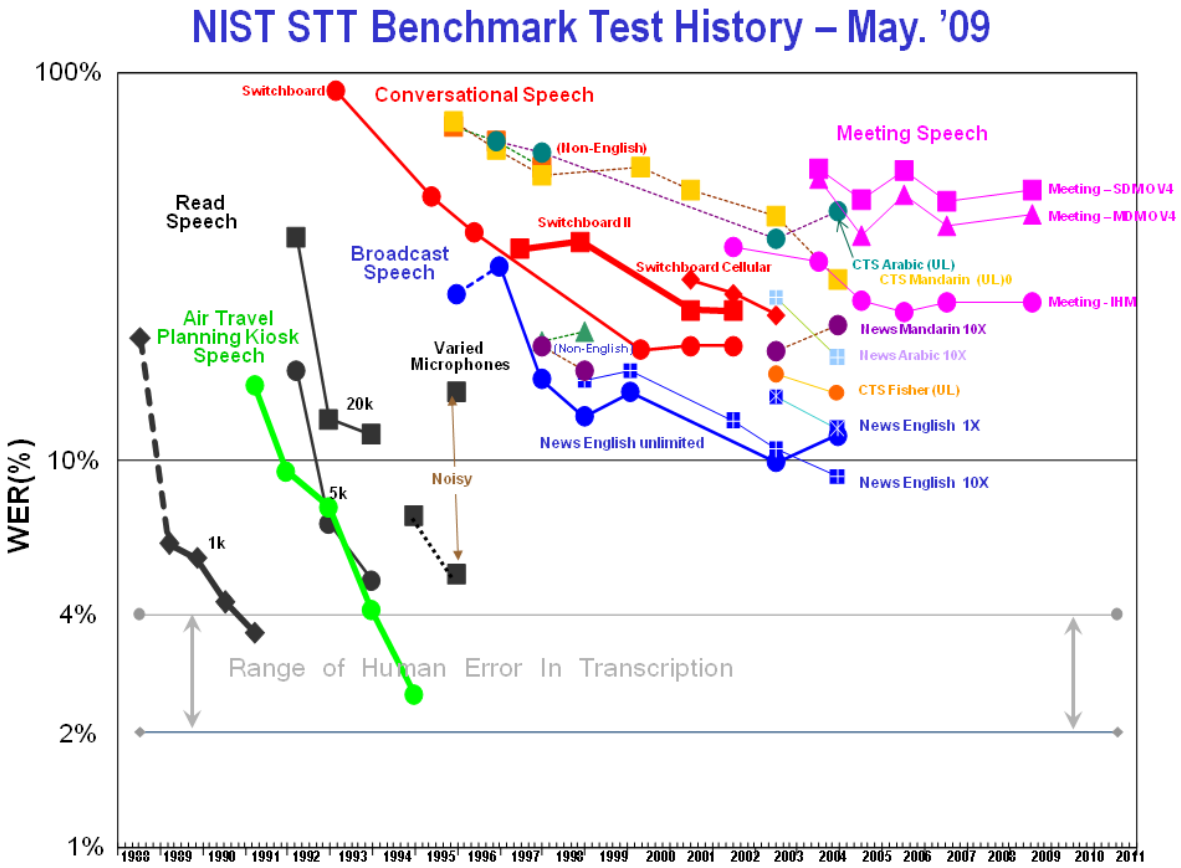
from Devlin et al., *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019

Speech Recognition

Conditions to take into account:

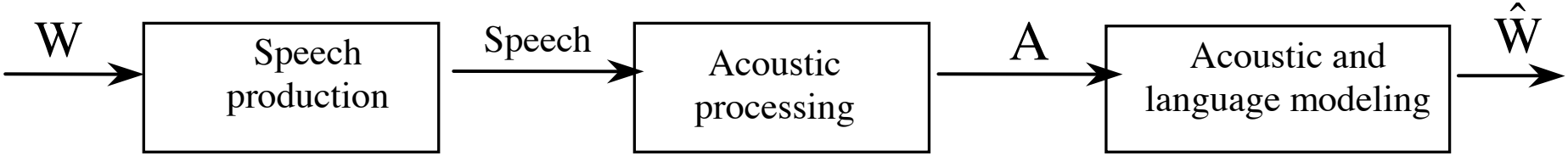
- Number of speakers
- Fluency of speech.
- Size of vocabulary
- Syntax
- Environment

Speech Recognition Progress

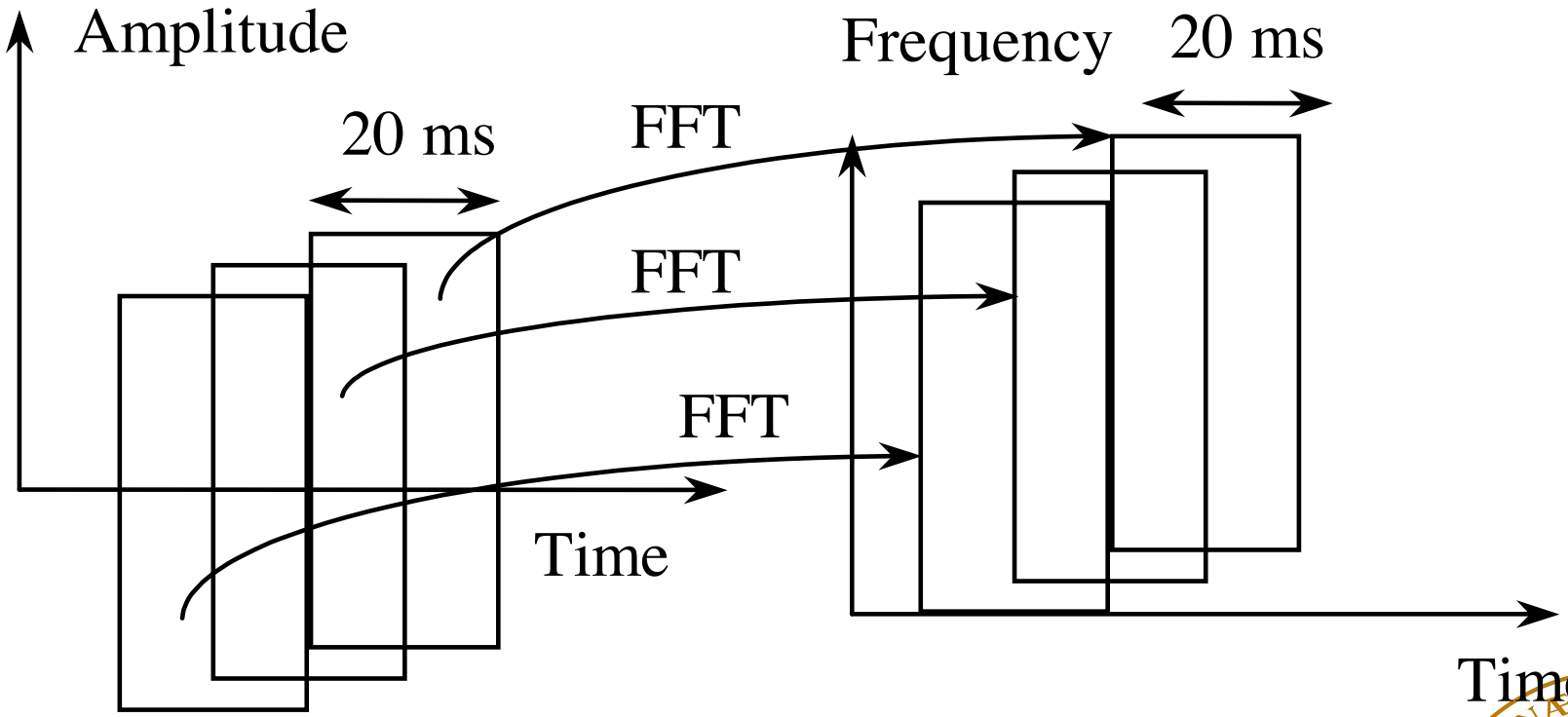


https://www.nist.gov/itl/iad/mig/rich-transcription-evaluation

Two-Step Recognition

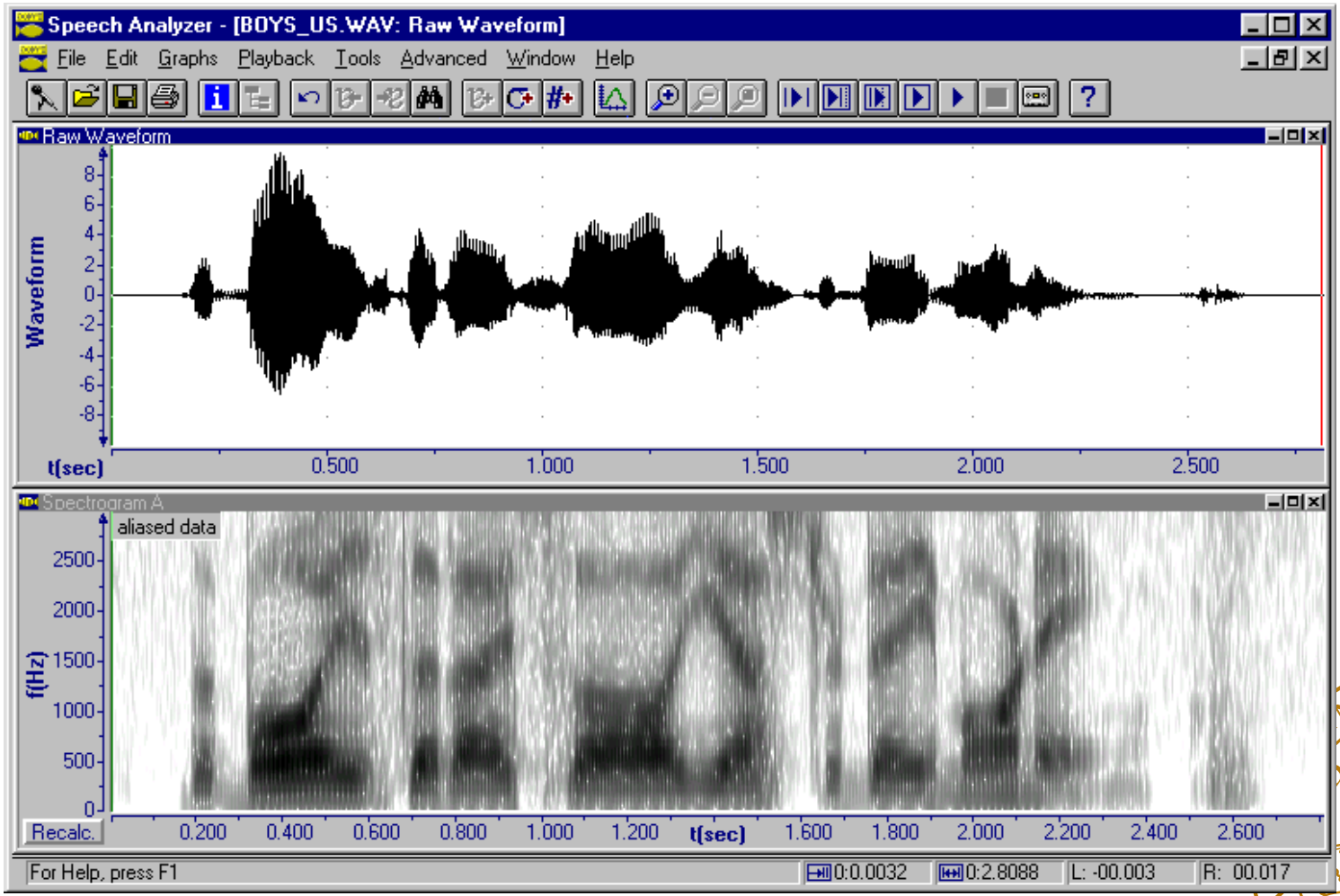


Speech Spectrograms



Speech Signals

The boys I saw yesterday morning



Neural Networks for Speech Recognition (I)

From 2015-2016, neural network architectures started to overtake HMM.
Most current systems use variants of recurrent neural networks.
A historical model from Waibel et al., Phoneme recognition using time-delay neural networks, 1989.

- Three phonemes B, D, and G
- An input vector consists of 16 melscale coefficients from a Fourier transform of a speech window of 10 ms: Energy at certain frequencies
- The context is modeled as a sequence of three such input vectors.
- Two hidden layers

Neural Networks for Speech Recognition (II)

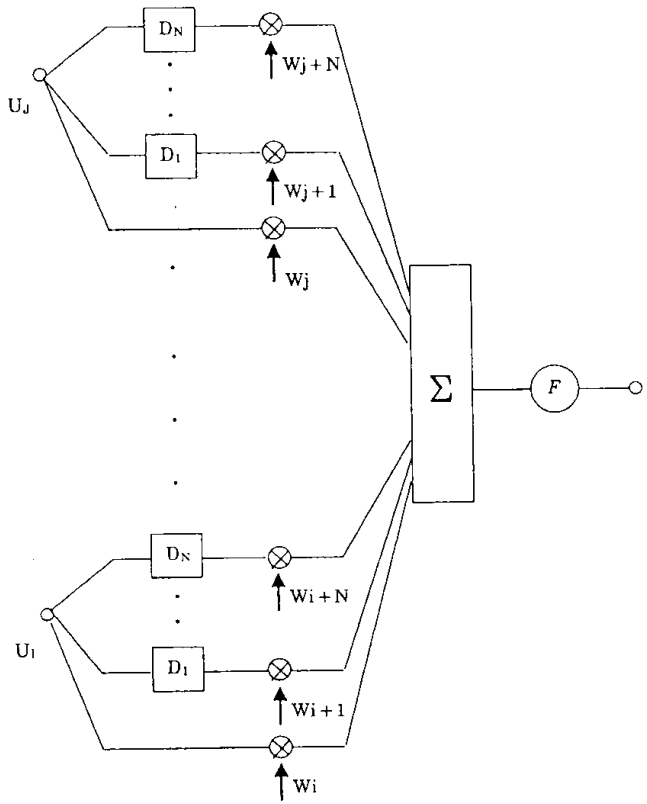


Fig. 1. A Time-Delay Neural Network (TDNN) unit.

From Waibel et al., Phoneme recognition using time-delay neural networks, IEEE Transactions of Acoustics, Speech, and Signal Processing, 37(3),1989

Neural Networks for Speech Recognition (III)

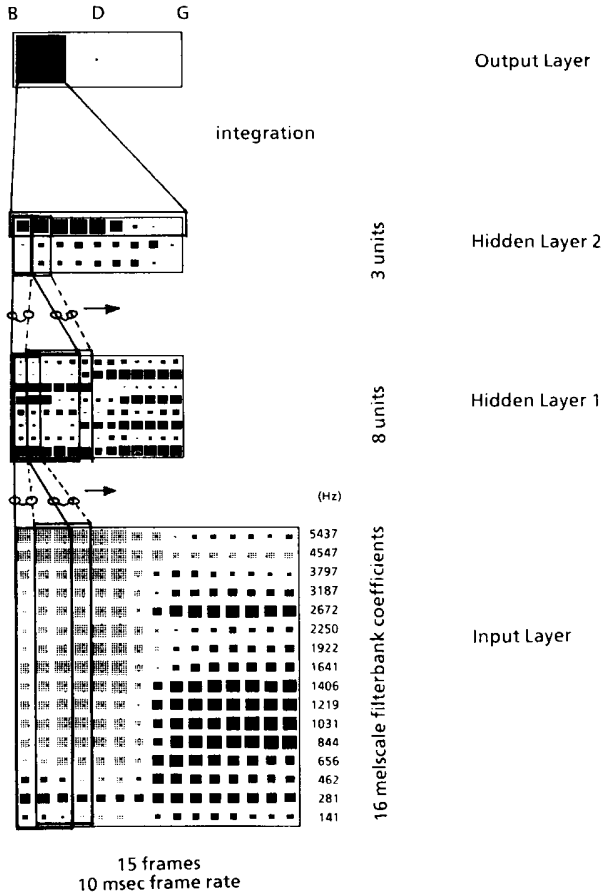


Fig. 2. The architecture of the TDNN.

From Waibel et al., Phoneme recognition using time-delay neural networks, IEEE Transactions of Acoustics, Speech, and Signal Processing, 37(3),1989