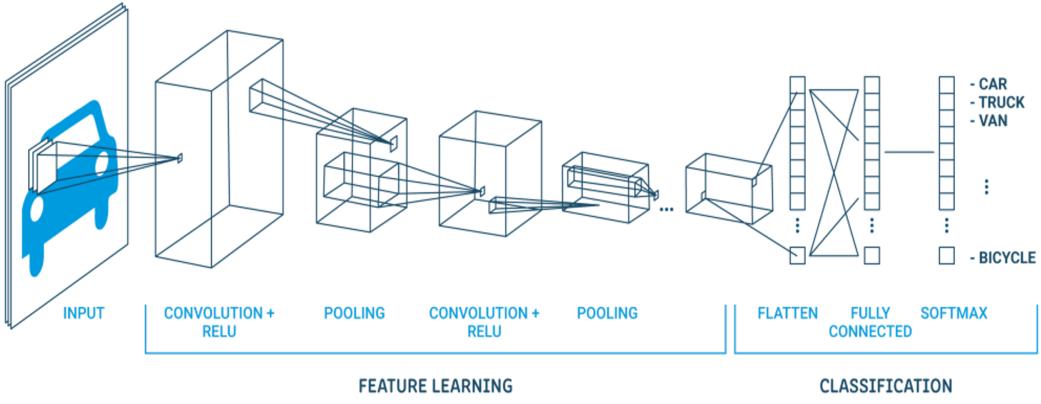
# RNNs and Attention

Arsalan Mosenia (am12546@nyu.edu)

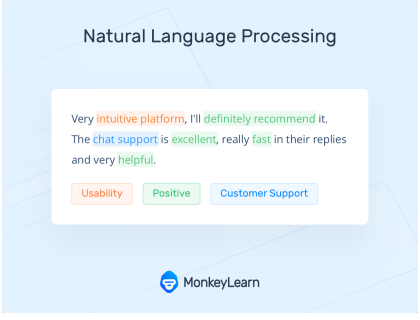
# Limitations of CNNs

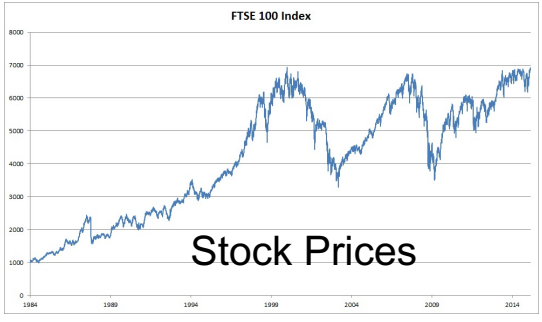



- Fixed size inputs (example: images of a given resolution)
  - Say I design a CNN, can you use it for inputs of a different dimension?
- Inherent spatial structure: spatially localized kernels/filters

固有的空间结构　　空间小范围的

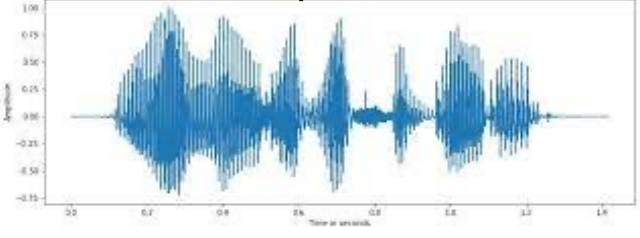
# Sequential Inputs



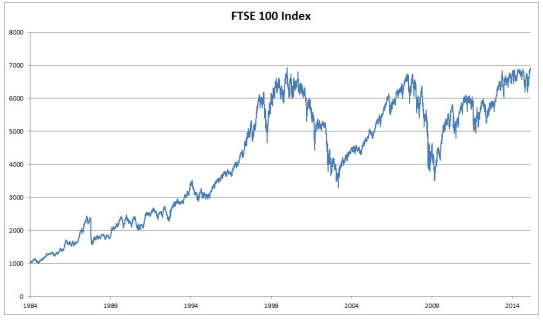




Video Frames

Speech



# Properties of Sequential Inputs




- Can be viewed as an ordered set of many inputs
  - Example: a video is a sequence of *many* images

具有因果模型

- Inherent causal structure (not always)
  - Future depends only on the past -> no time travel!
  - Example: the value of a stock at time *t* depends only what happened in the past

# Stock Prediction

- Say you would like to predict the value of a stock $x_t$ at time $t$
- Start by building a probabilistic model of the form:

$$x_t \sim P(x_t \mid x_{t-1}, \ldots, x_1).$$

Function of ($t$-1) inputs

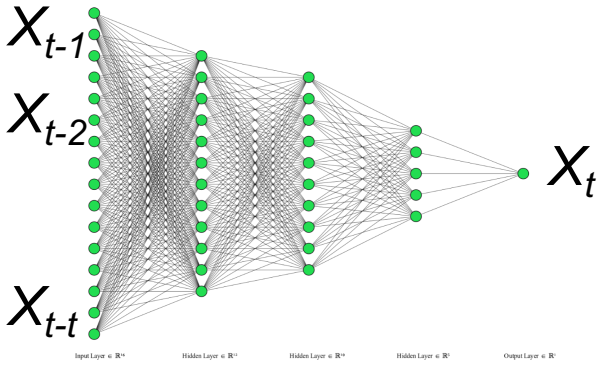
- What's the problem here?
  - Variable length function
  - You need a new function (*and more complex function*) for every time step!

# Autoregressive Models

- Strategy 1: only look at ***t*** time-steps in the past

$$x_t \sim P(x_t \mid x_{t-1}, \ldots, x_1).$$

$$x_t \sim P(x_t \mid x_{t-1}, \ldots, x_{t-\tau}).$$




Drawbacks?

- Models can still be quite large
- Old history forgotten

# *Latent* Autoregressive Models

- Strategy 2: use a "state" variable $h_t$ to store a summary of the history
  - For example, the state could simply count and keep track of time. Of course, in practice, the state will do much more complex things.
  - How do we model state?

summary of data we've seen so far

$$h_t = g(h_{t-1}, x_{t-1})$$

Current state | Past state | Past input

$$\hat{x}_t = P(x_t \mid h_t)$$

Prediction depends on state

# Latent Autoregressive Models

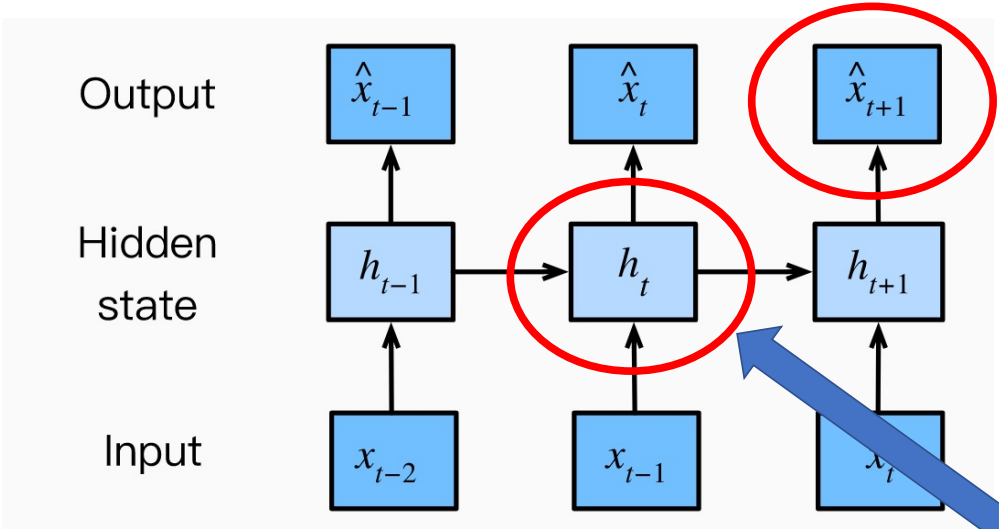



Fig. 8.1.2 A latent autoregressive model.

What does this prediction depend on?

How is this model different from a model that looks at all past history?

"next state" function is the same in every time step

RNN

# Recurrent Neural Networks

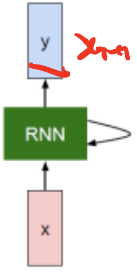
Small mod: we'll use past state and current input to output current state

We can process a sequence of vectors **x** by applying a recurrence formula at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

new state — some function with parameters W — old state — input vector at some time step

not $x_{t-1}$



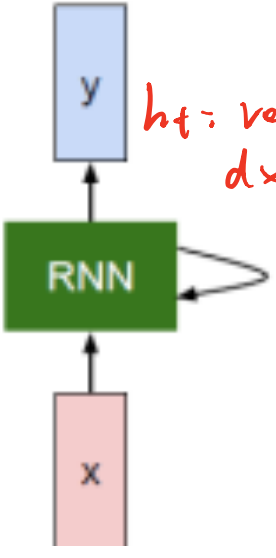

$g = f_W$ = Neural network

$x_t \rightarrow x_{t+1}$

What does $f_W$ look like?

A neural network with N≥1 layers!

# [Vanilla] Recurrent Neural Networks




$h_t$ : vector of size $d \times 1$

3 matrices: $W_{hh}$ $W_{xh}$ $W_{hy}$

$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

FC layer FC layer

$d \times d$ $d \times 1$ $d \times n$ $n \times 1$

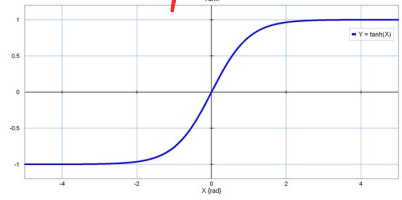
none linearity

linear form

$$y_t = W_{hy} h_t$$

$n \times 1$ $n \times d$ $d \times 1$

Output FC layer



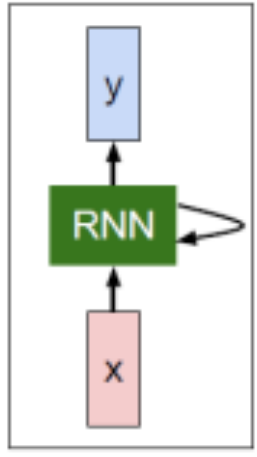
**Character-level language model example**

Vocabulary:
[h,e,l,o]

Example training sequence:
**“hello”**

**Goal:** Train a model to auto-complete a word, assuming a vocabulary of 4 letters [h,e,l,o]

**One-hot coding:**
h = [1,0,0,0]
e = [0,1,0,0]
l = [0,0,1,0]
o = [0,0,0,1]



# [Vanilla] Recurrent Neural Networks

Assume a model has been trained! (we'll see later how to train the model)

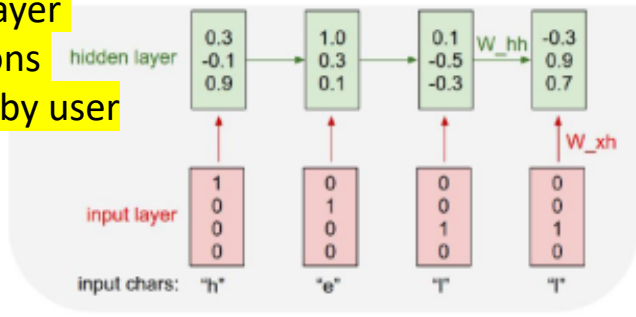
**Character-level language model example**

Vocabulary:
[h,e,l,o]

Example training sequence:
**"hello"**

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Hidden layer dimensions selected by user

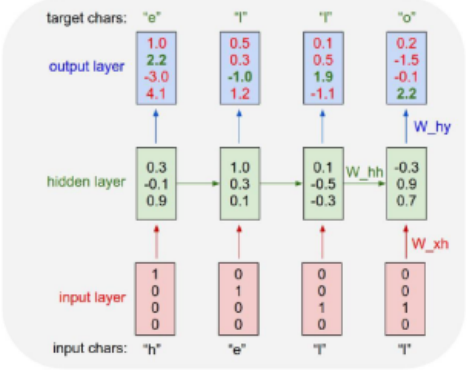


# [Vanilla] Recurrent Neural Networks

**Character-level language model example**

Vocabulary:
[h,e,l,o]

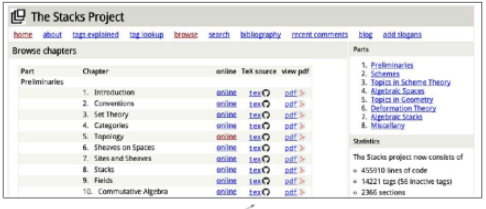
Example training sequence:
**“hello”**



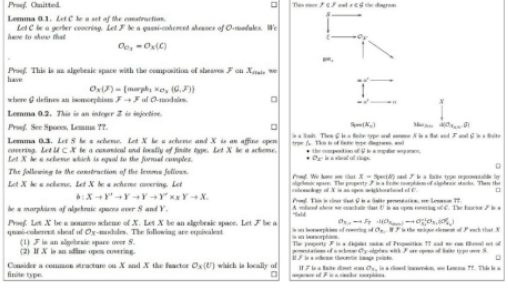

What happens if we input “h”, “e”, “l”, “l”?

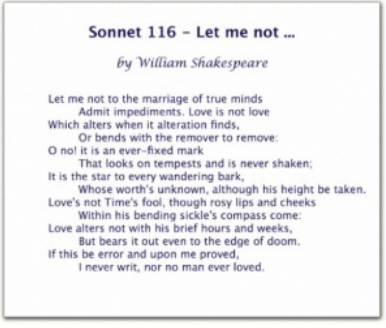
# Examples

open source textbook on algebraic geometry




Latex source






at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hreqtrs nigtike,aoaenns lng
```

train more

```
"Tnant thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lisn thond hon at. MeiDimorotion in ther thize."
```

train more

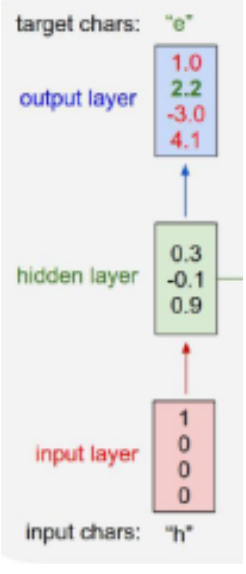
```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in hed oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

# Generating New Words

- Once we've trained the model, how do we use it?

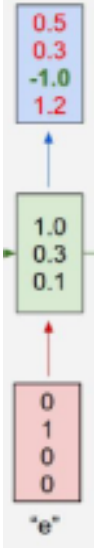



$$p_i = \frac{e^{y_i}}{\sum e^{y_i}}$$

**RNN output defines a probability distribution over characters**

**Sample from distribution**

**'e'**

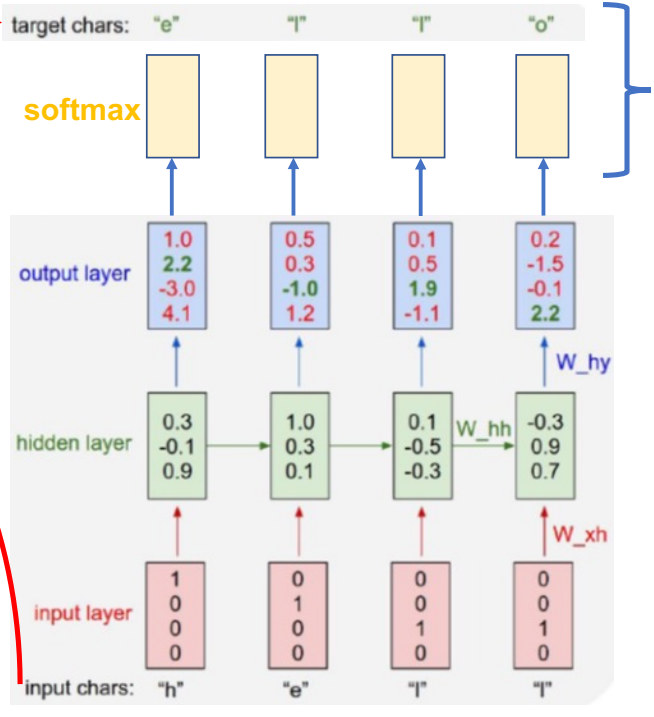



**Sample from distribution**

**'h'**

Note: with some probability you might pick every other letter also

# RNN Training




Input sequence shifted right by 1

Cross-entropy loss between target chars and predictions, averaged over the T predictions.

Select T (# of time-steps) for training. Let's select T=4. The RNN is "unrolled" T times….

# Training Data Creation

- In practice, you have a loooo…ooong sequence of text. For example, an entire book. We will use *HG Wells*' book, *The Time Machine* as an example and assume T=5.

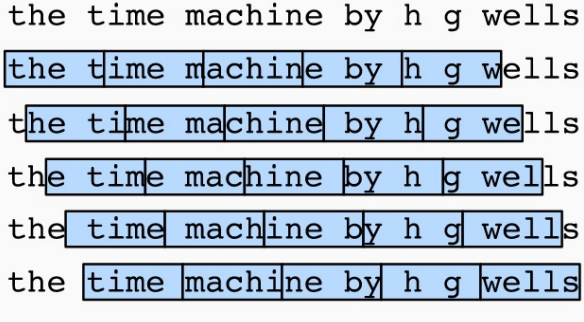



Fig. 8.3.1 Different offsets lead to different subsequences when splitting up text.

The entire book can be partitioned into T=5 length character strings starting with an offset O.

Different offsets give different sequences. How should we create our training data and batches?

# Sequential Sampling

- In each epoch

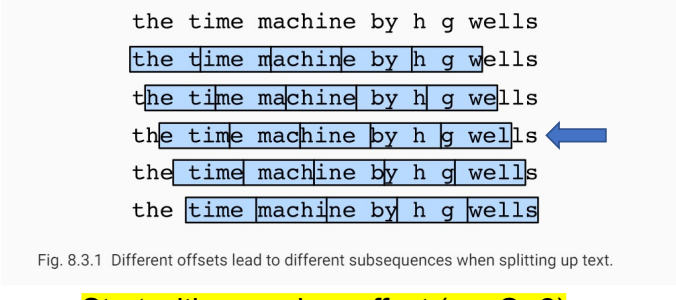



Fig. 8.3.1 Different offsets lead to different subsequences when splitting up text.

Start with a random offset (ex: O=2)

e tme

e mac

hine

by h

We set T=5.
Mini-batch size B=2

Samples from adjacent mini-batches are adjacent in the original sequence

# Random Sampling

Recall, we set T=5.
Assume mini-batch size B=2

- In each epoch

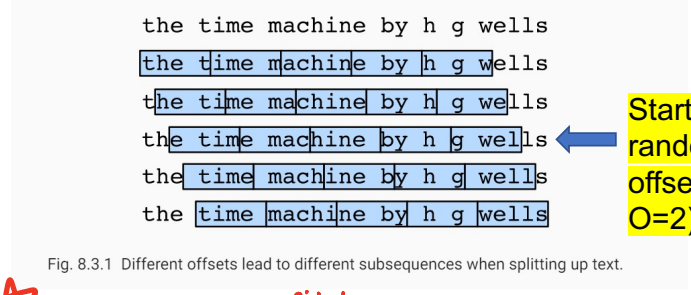



Fig. 8.3.1 Different offsets lead to different subsequences when splitting up text.

Start with a random offset (ex: O=2)

e tim
e mac
hine
by h
g wel

Randomly shuffle

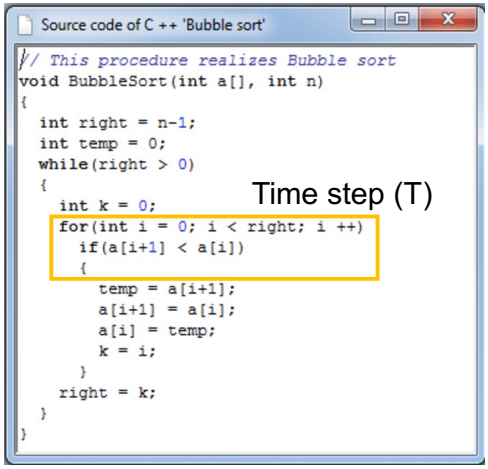
hine
e tim
by h
g wel
e mac

1. avoid overfitting

2. have a slightly different distribution in your dataset, more 泛化

Samples from adjacent mini-batches are NOT adjacent in the original sequence

# Real-world Problems are Complex




What does this code do?

Sometimes the earliest tokens might be the most informative!

I am writing about topic X…

……..

On a different note…

Back to topic X…..

What is the topic of this email?

Sometimes intermediate tokens might be irrelevant!

How are you doing? Hope you've been keeping well…

I am writing because…

What is the topic of this email?

Sometimes you might want to "reset" your RNN

GRU
LSTM
} extensions of RNN

# Going Beyond Character Level Modeling

- In practice, character level models are not very successful. We would ideally like to encode more semantically meaningful inputs
- *Tokenization*: partitions inputs into meaningful tokens (example: characters, words, combinations of words, or sub-words)

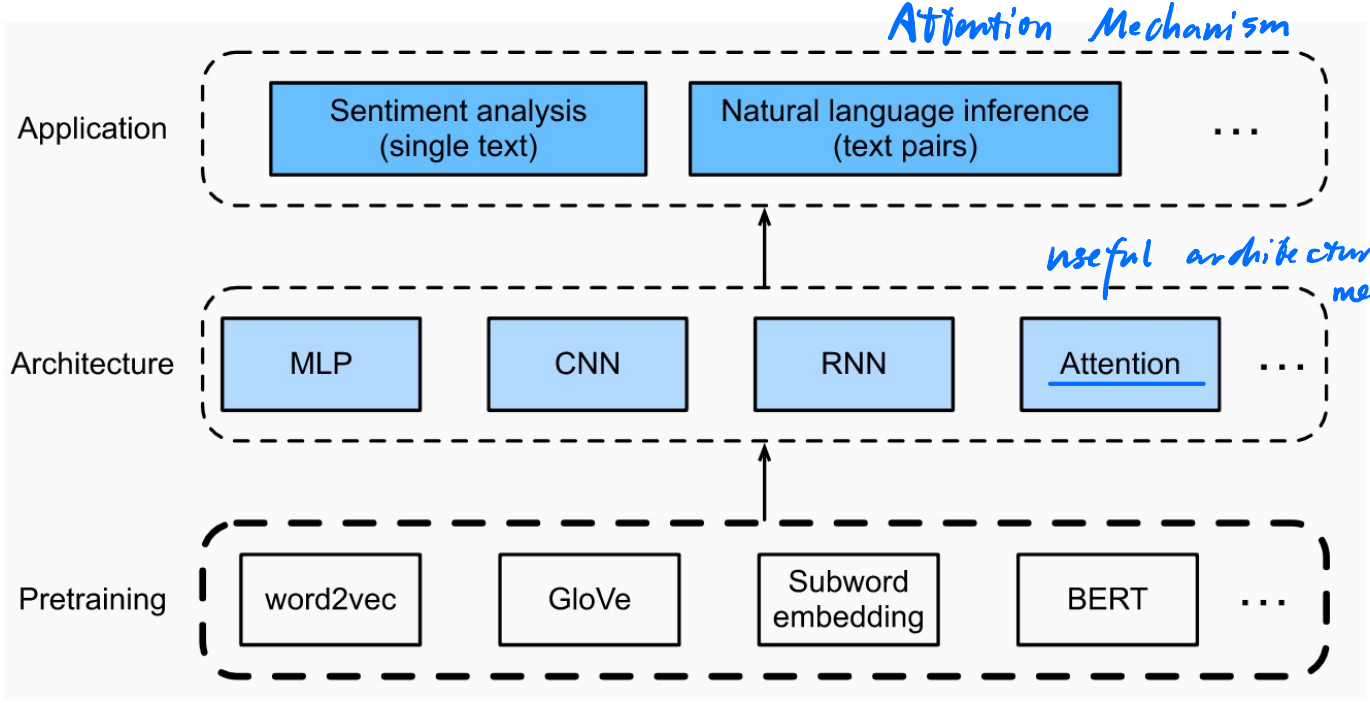
MXNET PYTORCH TENSORFLOW

```
def tokenize(lines, token='word'):  #@save
    """Split text lines into word or character tokens."""
    if token == 'word':
        return [line.split() for line in lines]
    elif token == 'char':
        return [list(line) for line in lines]
    else:
        print('ERROR: unknown token type: ' + token)

tokens = tokenize(lines)
for i in range(11):
    print(tokens[i])
```

```
['the', 'time', 'machine', 'by', 'h', 'g', 'wells']
[]
[]
[]
[]
['i']
[]
[]
['the', 'time', 'traveller', 'for', 'so', 'it', 'will', 'be', 'convenient', 'to', 'spe
['was', 'expounding', 'a', 'recondite', 'matter', 'to', 'us', 'his', 'grey', 'eyes',
['twinkled', 'and', 'his', 'usually', 'pale', 'face', 'was', 'flushed', 'and', 'animat
```

# NLP Overview



# NLP Applications

- Machine Translation
- Sentiment Analysis (e.g., Social Media Monitoring)
- Question-answering (e.g., Chatbots)
- Hiring and Recruitment
- Automatic Summarization
- Spam Filtering
- Voice Assistants

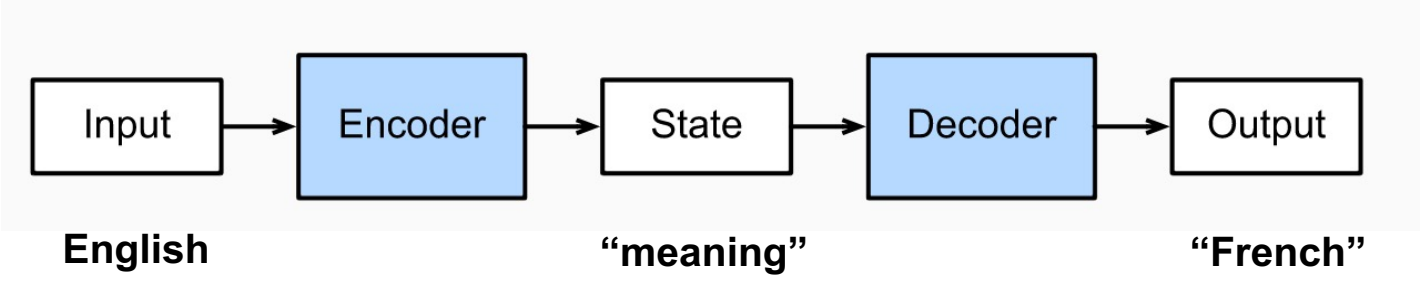
# Machine Translation

- Statistical Machine Translation (SMT)
- Rule-based Machine Translation (RBMT)
- Hybrid Machine Translation (HMT)
- Neural Machine Translation (NMT)

Classic Sequence-to-Sequence Problem

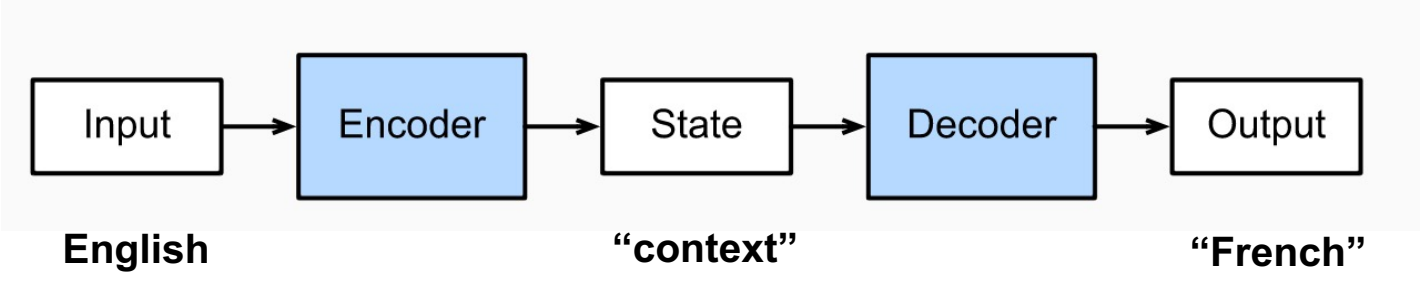
# Neural Machine Translation (NMT)

- Language translation



# Neural Machine Translation (NMT)

- Language translation



# Implementation

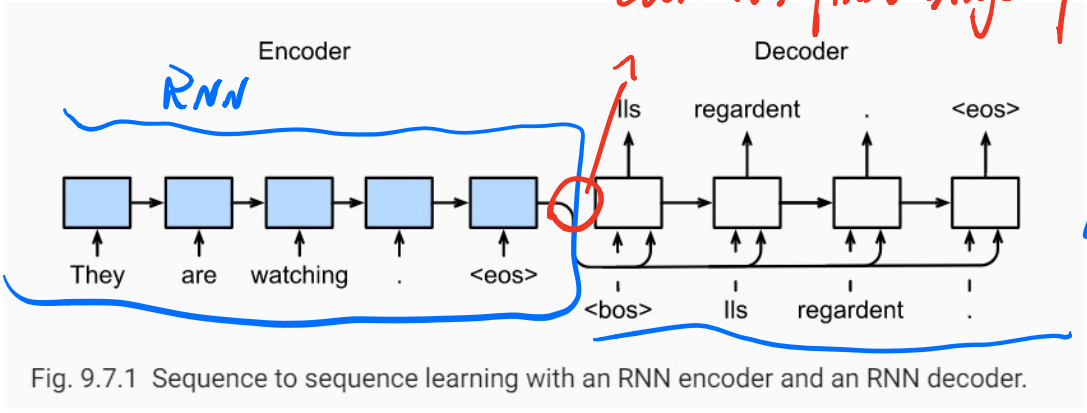
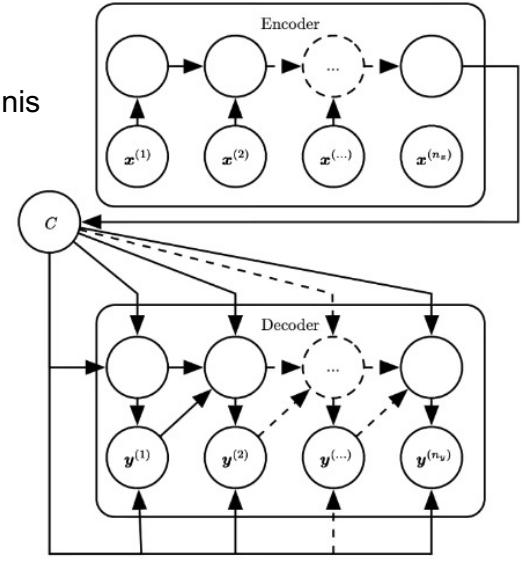



Fig. 9.7.1 Sequence to sequence learning with an RNN encoder and an RNN decoder.

Context: final encoder state, or a combination of all intermediate states
Decoder inputs: target sequences and in some cases, context
Decoder outputs: shifted target sequence

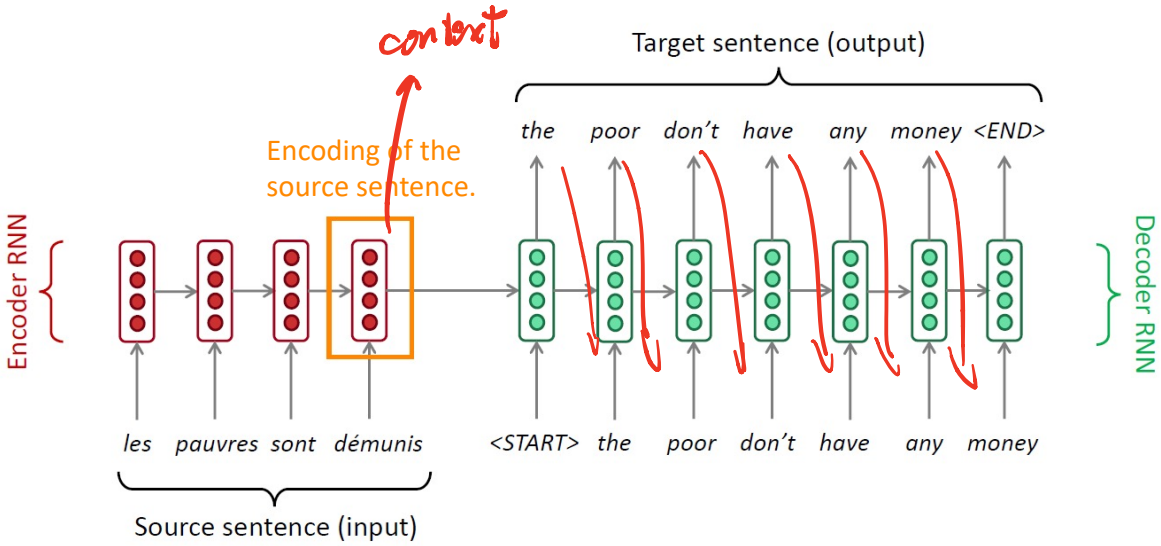
# Decoder-Encoder Model

Les pauvres sont démunis




The poor do not have any money

# Information Bottleneck




**Information Bottleneck:** *All information* about the source sentence captured as a single context

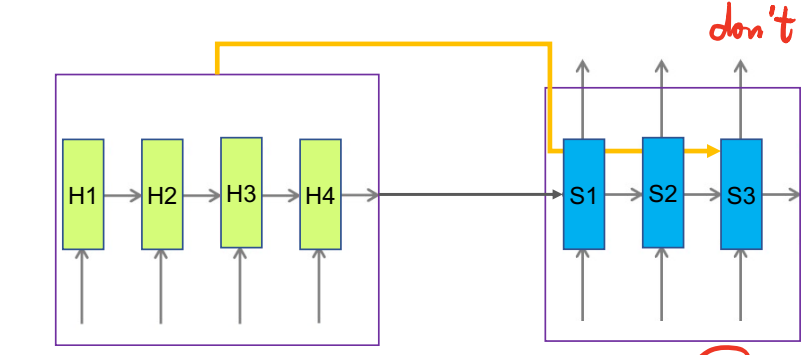
only pass h4 h4 has less information about previous

# Classic Attention Mechanism

Solution to the bottleneck problem: *Attention*

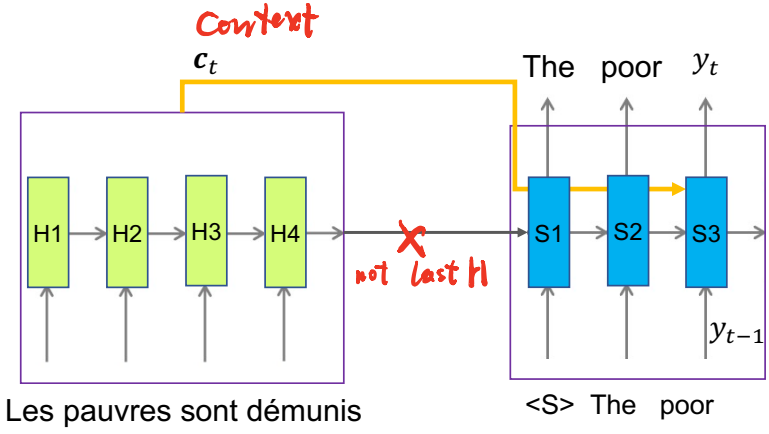
**Key idea:** On each step of the decoder, *focus on a particular part* of the source sequence

# Classic Attention Mechanism




Random Access Memory: Retrieve as Needed

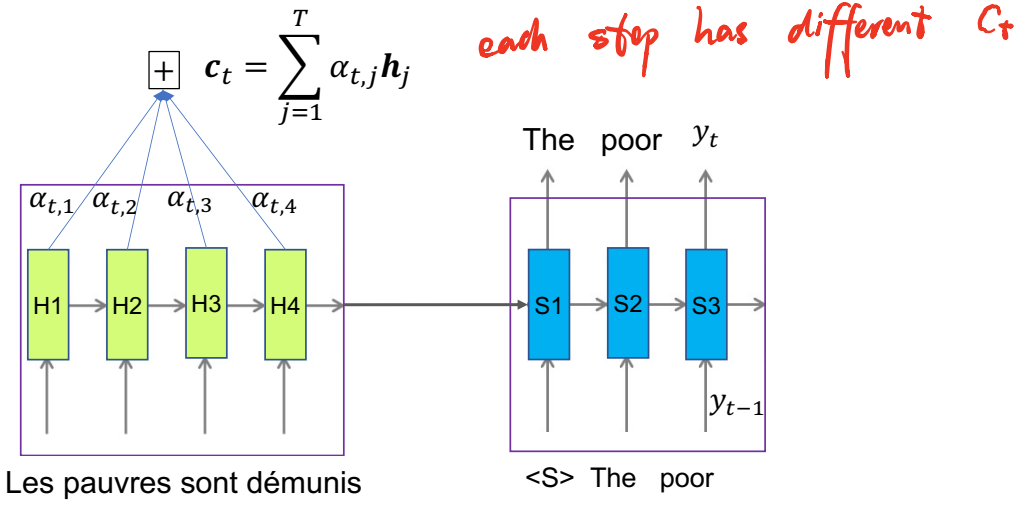
# Classic Attention Mechanism




$$p(y_t|y_{t-1}, \dots, y_1, \boldsymbol{x}) = f(y_{t-1}, \boldsymbol{s}_t, \boldsymbol{c}_t)$$

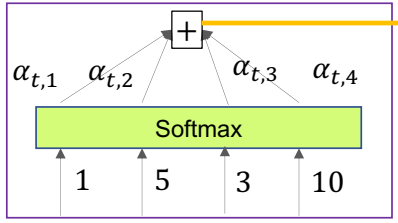
Hidden state

# Classic Attention Mechanism



# Classic Attention Mechanism

$$\alpha_{t,j} = \frac{\exp(Score_{t,j})}{\sum_{k=1}^{T} \exp(Score_{t,k})} = \text{Softmax (scores)}$$



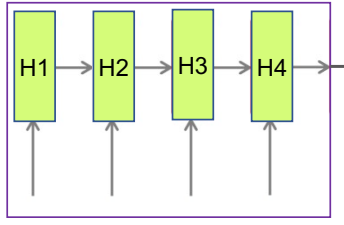

$C_t = \sum_i h_i \rightarrow \text{softmax (scores)}$

$C_t = [0\ 0\ 0\ 1]$ for all cases

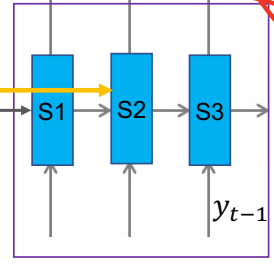
$c_t$ is different for different steps

$c_t [0.1, 0.5, 0.2, 0.2]$

$c_t [0.5, 0.5, 0, 0]$




Les pauvres sont démunis

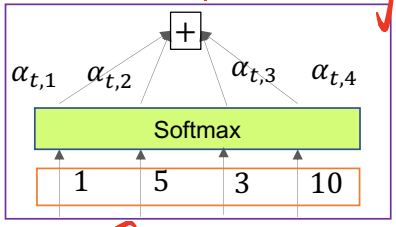



The poor $y_t$

$y_{t-1}$

<S> The poor

# Classic Attention Mechanism



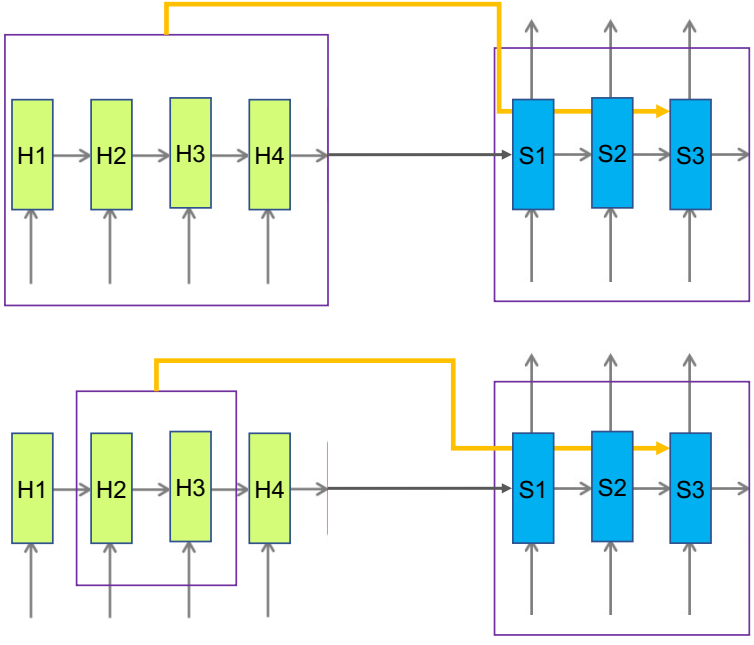

$$Score_{t,j} = Score(\boldsymbol{s}_t, \boldsymbol{h}_j)$$

Common choices

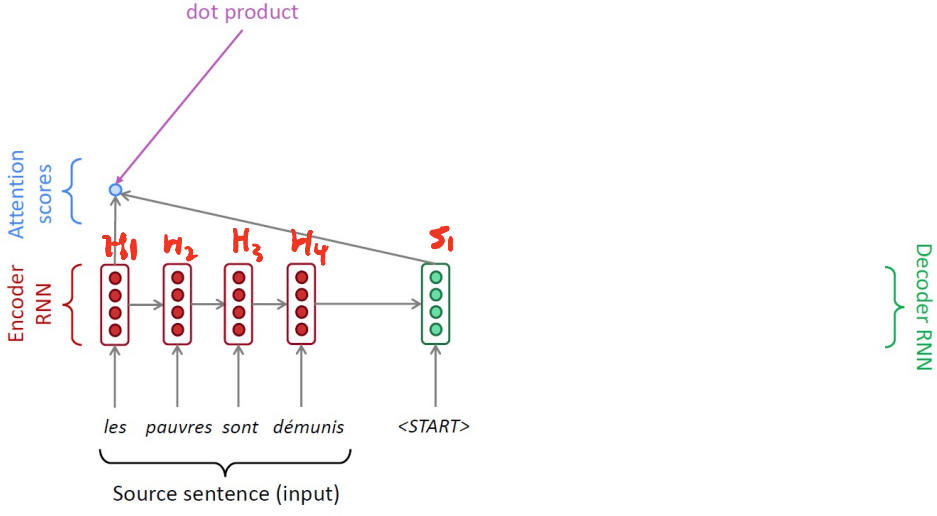
$$Dot(\boldsymbol{s}_t, \boldsymbol{h}_j) = h_t^T s_t$$

$$h_t^T W_a s_t$$

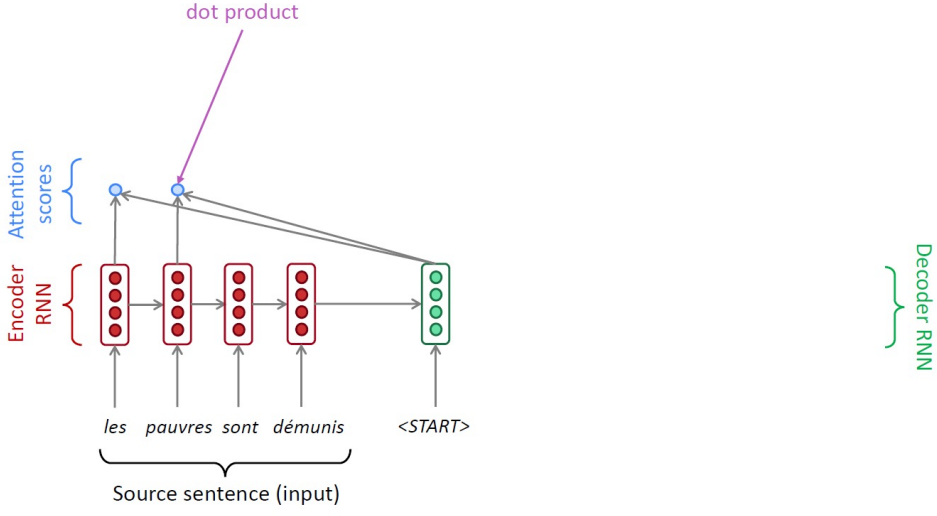
# Global vs Local Attention



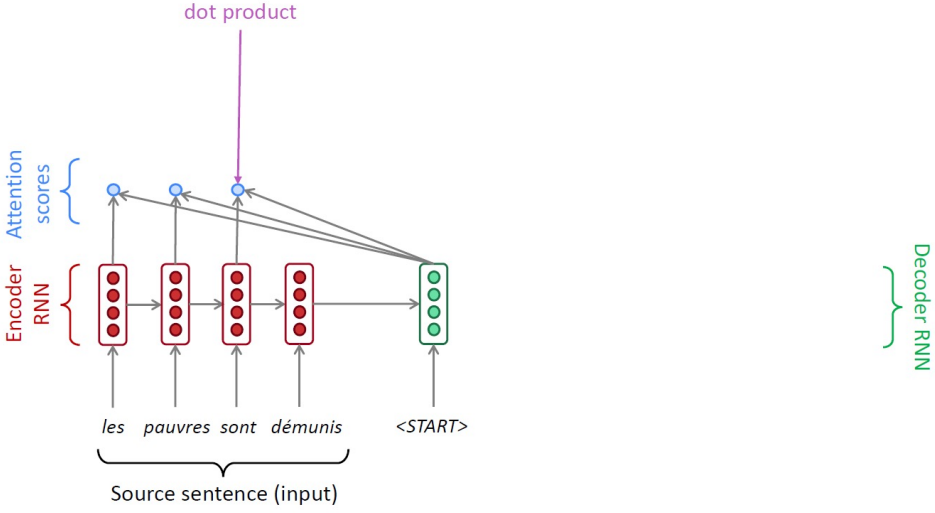
# Running Example



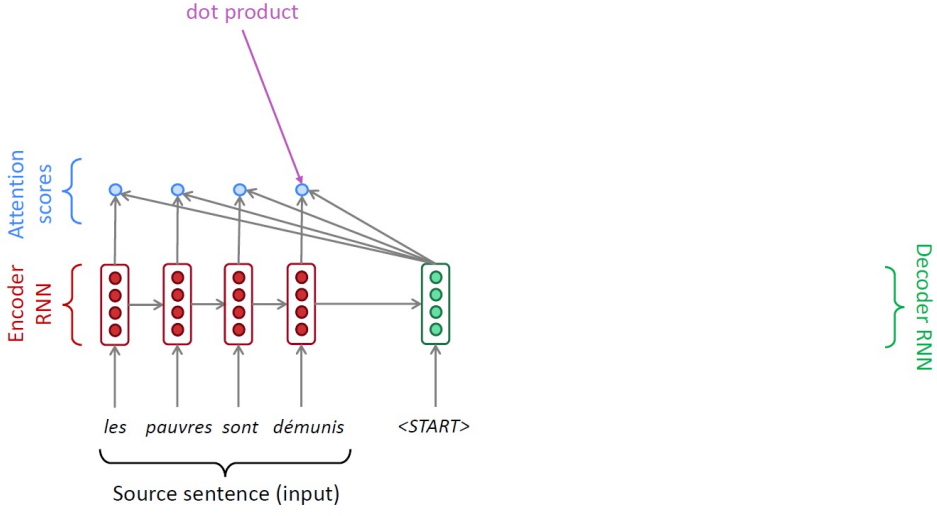
# Running Example



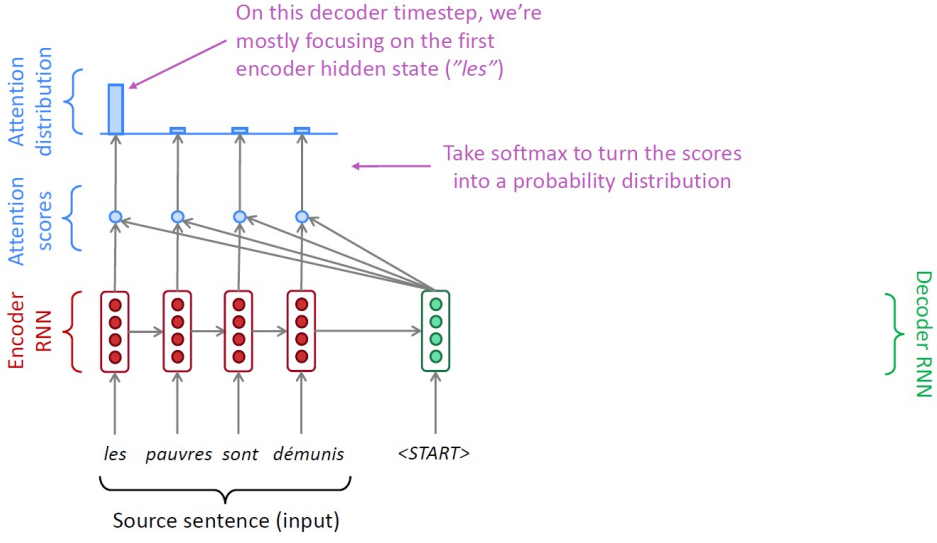
# Running Example



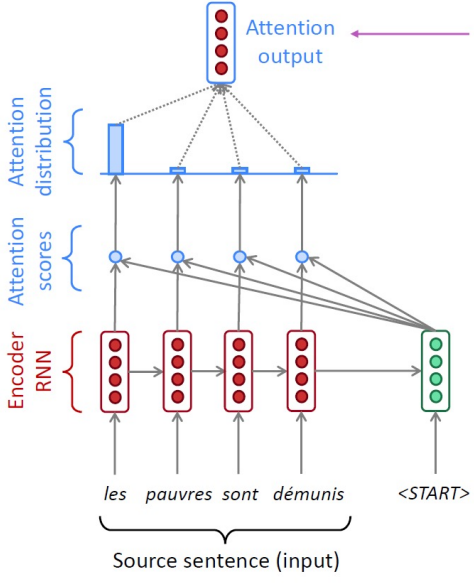
# Running Example



# Running Example



# Running Example

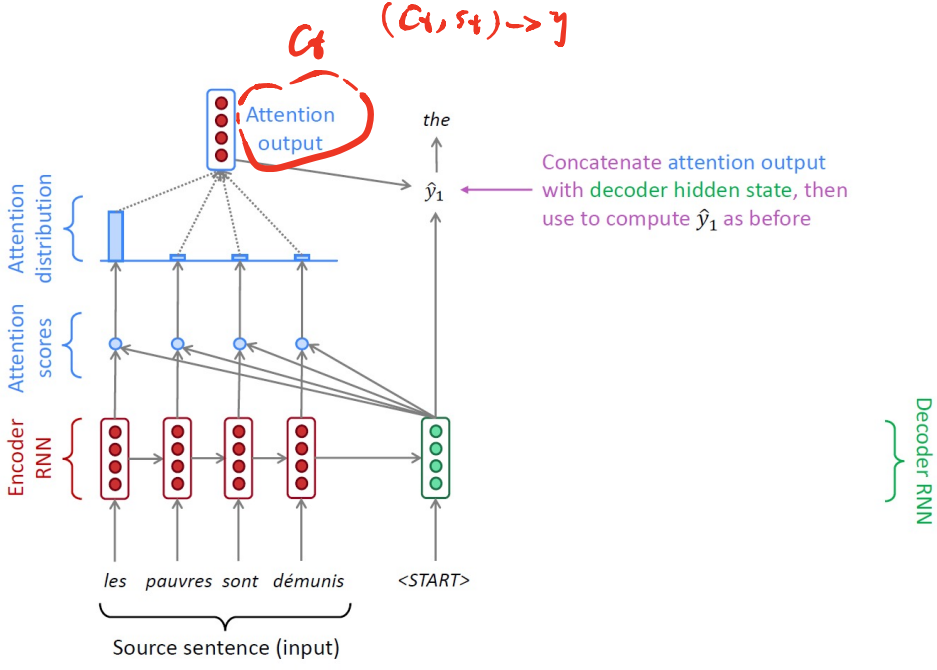



Use the attention distribution to take a **weighted sum** of the encoder hidden states.

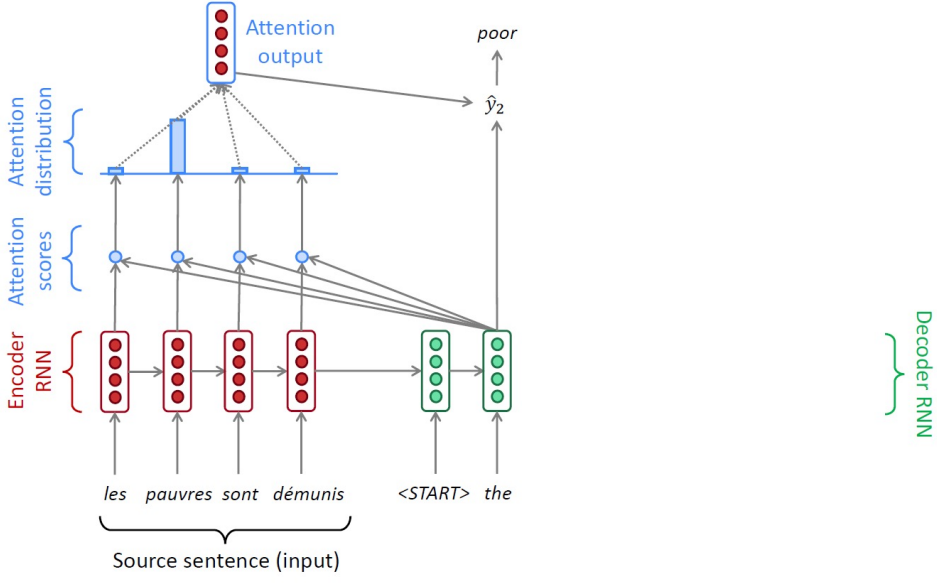
The attention output mostly contains information the hidden states that received high attention.

Decoder RNN

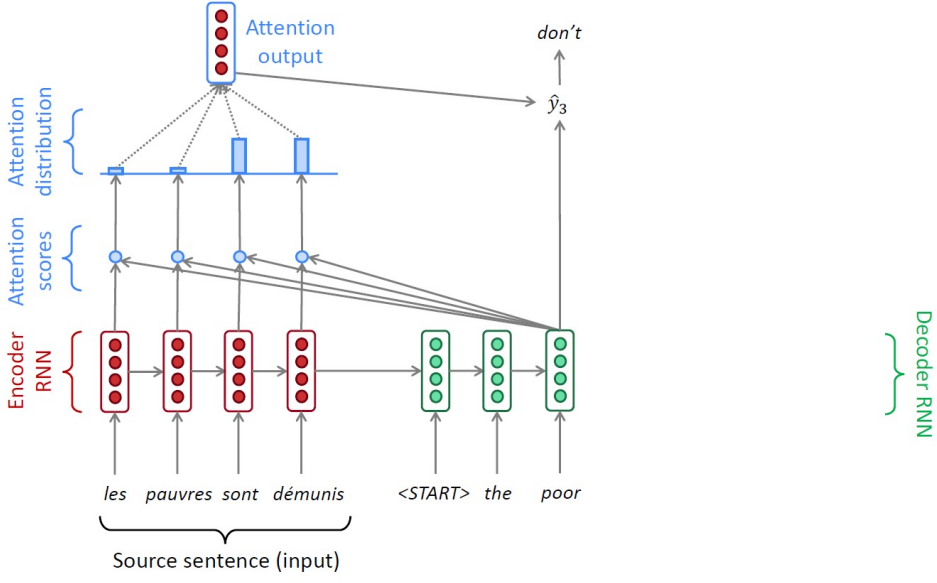
# Running Example



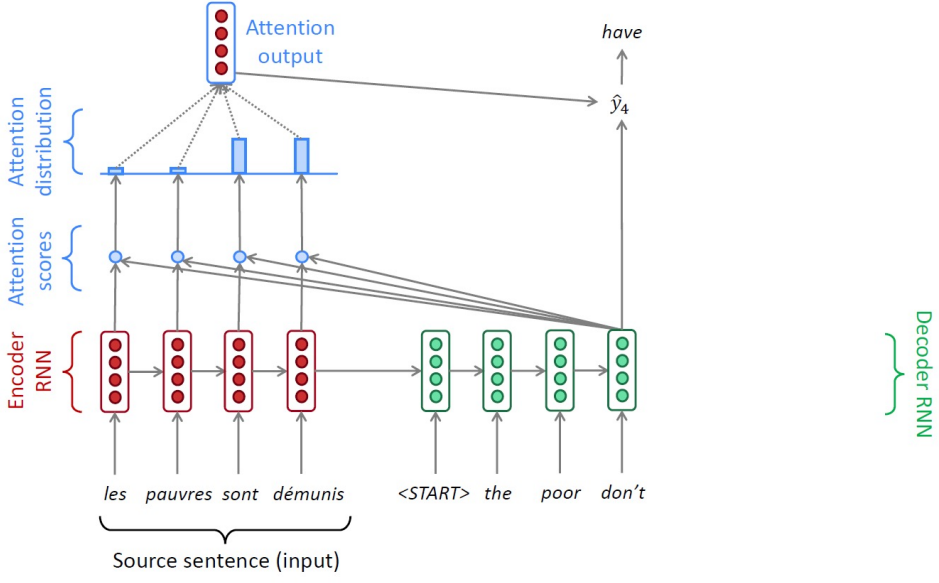
# Running Example



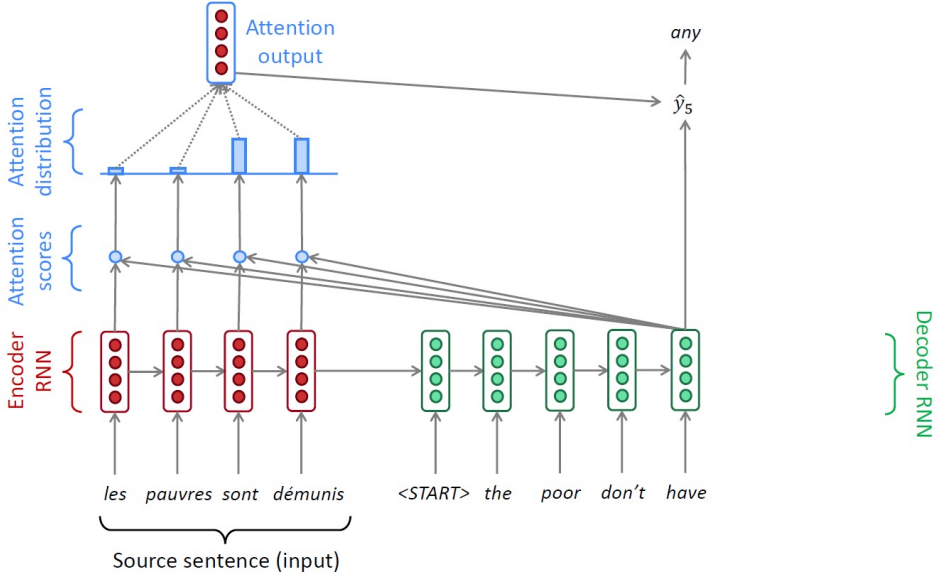
# Running Example



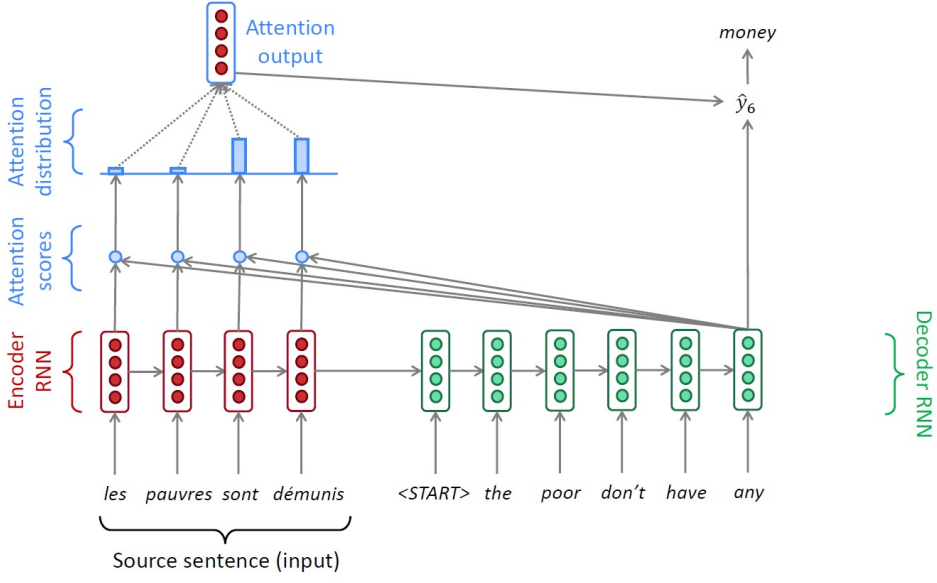
# Running Example



# Running Example



# Running Example

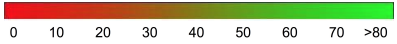


# Translation with Attention Mechanism

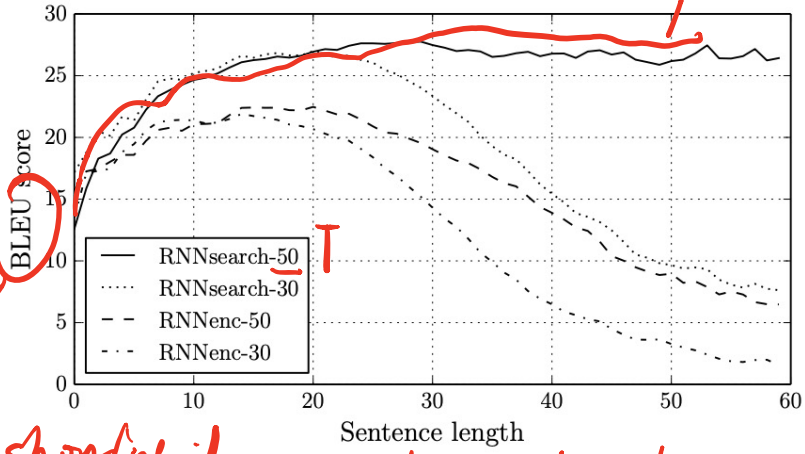
Attention results

RNN

RNN · Attention




| BLEU Score | Interpretation |
|---|---|
| < 10 | Almost useless |
| 10 - 19 | Hard to get the gist |
| 20 - 29 | The gist is clear, but has significant grammatical errors |
| 30 - 40 | Understandable to good translations |
| 40 - 50 | High quality translations |
| 50 - 60 | Very high quality, adequate, and fluent translations |
| > 60 | Quality often better than human |




metric for showing how well a translation is

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." *ICLR 2015*