Deep Learning

Yannet Interian
yinterian@usfca.edu

Agenda

- Machine translation task
- Seq2seq models with attention
- Contextual word representations
- Transformer architecture for NLP

Sequence to sequence models with attention

Machine Translation as supervised ML

Machine Translation (MT) is the task of translating a sentence *x* from one language (the source language) to a sentence *y* in another language (the target language).

x: *L'homme est né libre, et partout il est dans les fers*

$\downarrow$

y: *Man is born free, but everywhere he is in chains*

Pre-deep learning Machine Translation (90’s-2000’s)

- Huge research field
- The best systems are extremely complex
  - Systems have many separately-designed subcomponents
  - Lots of feature engineering
    - Need to design features to capture particular language phenomena
  - Require compiling and maintaining extra resources
    - Like tables of equivalent phrases
  - Lots of human effort to maintain
    - Repeated effort for each language pair!

In 2014 Machine Translation meets Deep learning

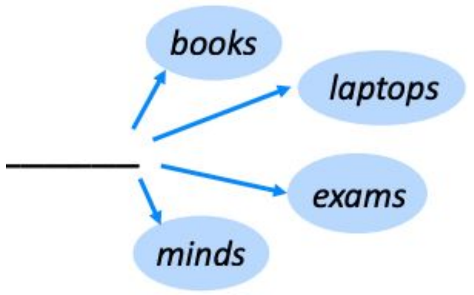
What is Neural Machine Translation?

- Neural Machine Translation (NMT) is a way to do Machine Translation with a single neural network.
- The neural network architecture is called sequence-to-sequence (aka seq2seq) and it involves two RNNs.

Language Model (LM)

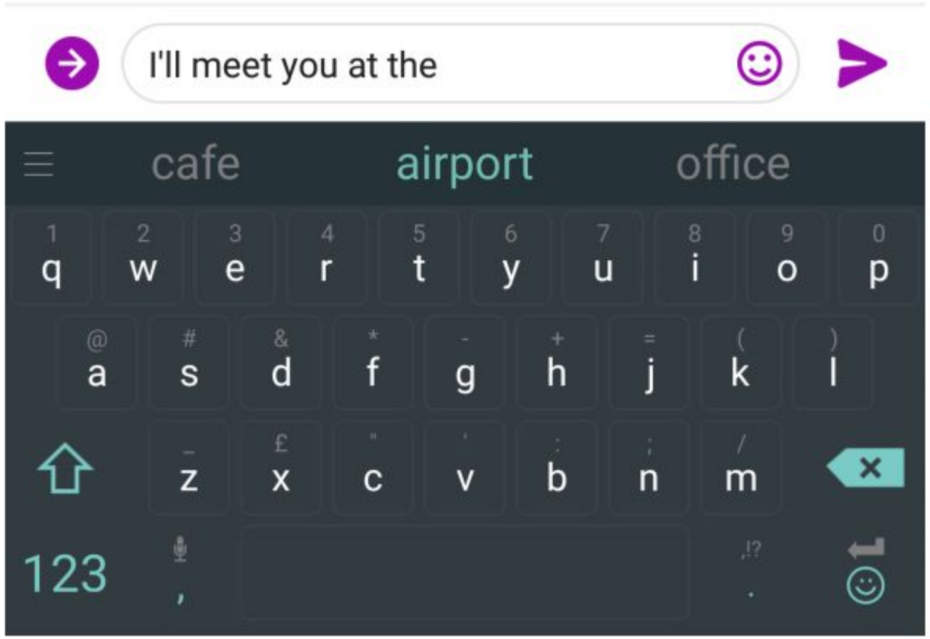
- Language Modeling is the task of predicting what word comes next.

  The students opened their ______



- More formally: given a sequence of words, compute the probability distribution of the next word

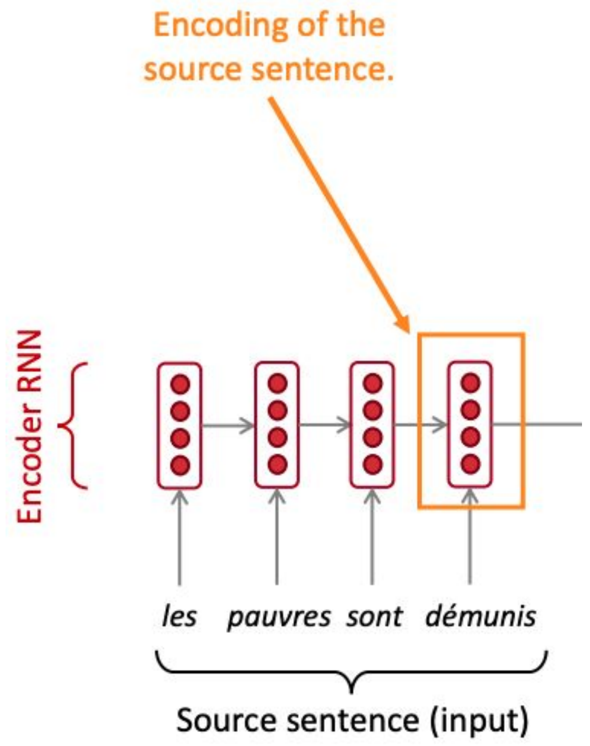
You use Language Models every day!



I'll meet you at the

Neural Machine Translation (NMT)

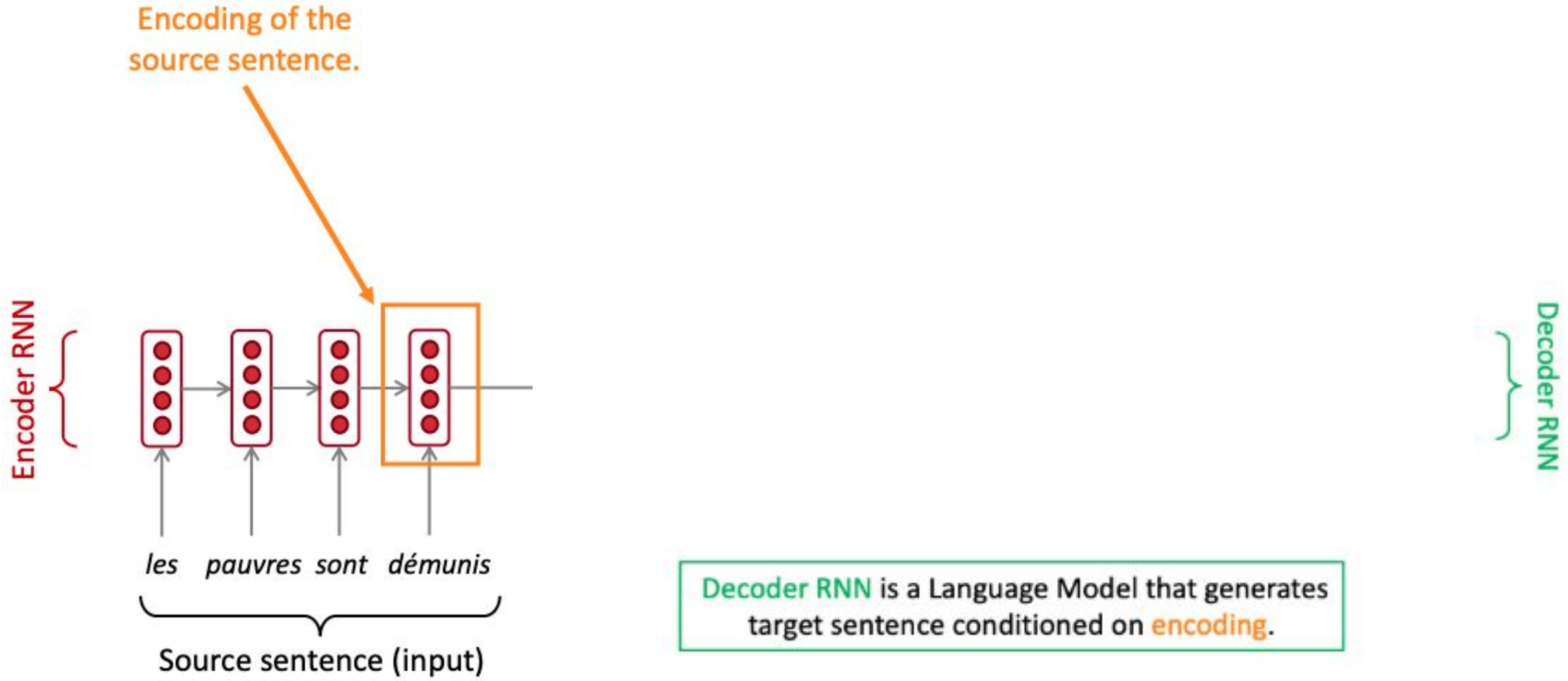
Sequence to sequence model



- Encoder RNN produces an encoding of the source sentence
  - Final hidden state of the RNN
- This encoding provides initial hidden state for the Decoder RNN

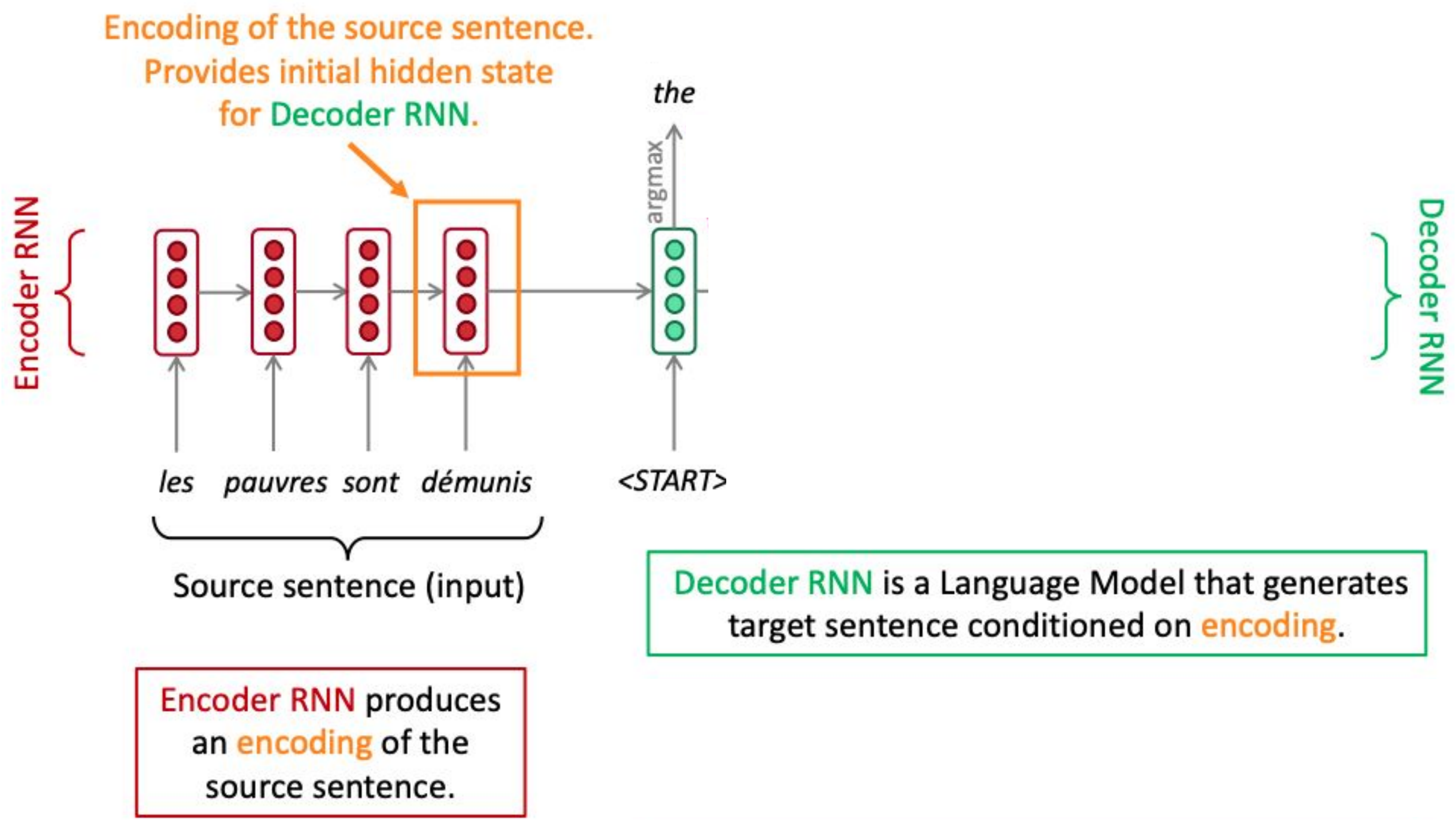
Neural Machine Translation (NMT)

Sequence to sequence model



Decoder RNN is a Language Model that generates target sentence conditioned on encoding.

Neural Machine Translation (NMT)
Sequence to sequence model

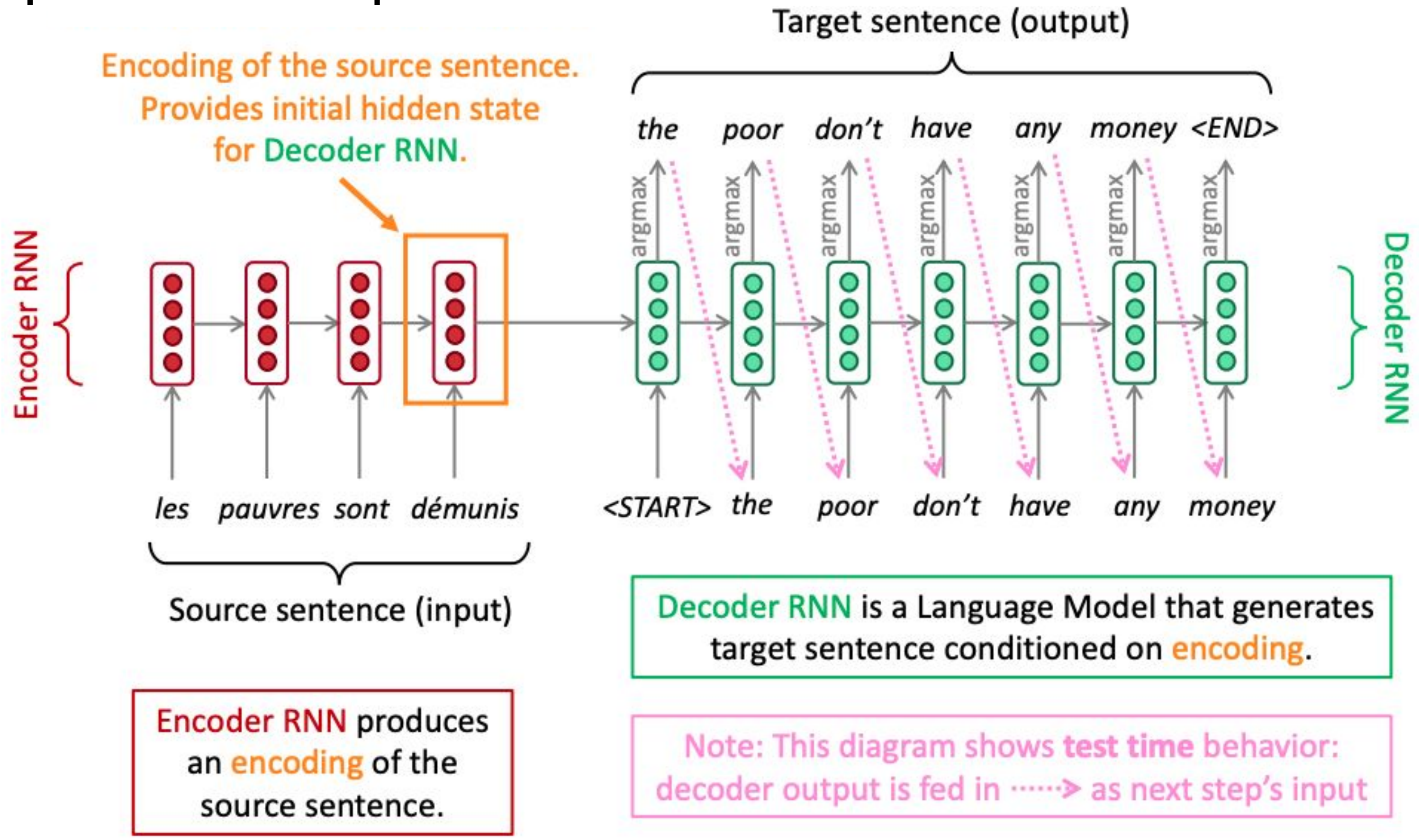


Decoder RNN is a Language Model that generates target sentence conditioned on encoding.

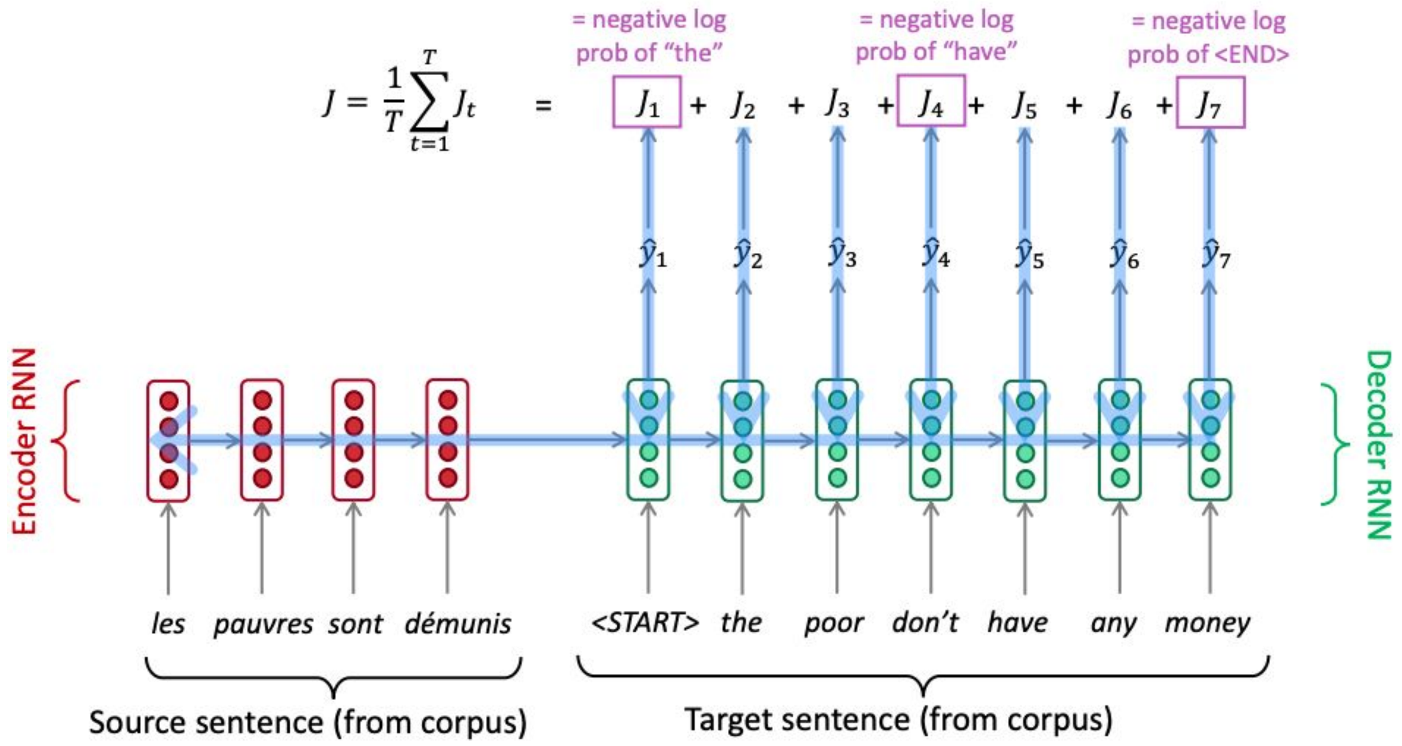
Encoder RNN produces an encoding of the source sentence.

Neural Machine Translation (NMT)

Sequence to sequence model


Target sentence (output)
Encoding of the source sentence. Provides initial hidden state for Decoder RNN.
the
poor
don't
have
any
money
<END>
argmax
Encoder RNN
Decoder RNN
les
pauvres
sont
démunis
<START>
the
poor
don't
have
any
money
Source sentence (input)
Decoder RNN is a Language Model that generates target sentence conditioned on encoding.
Encoder RNN produces an encoding of the source sentence.
Note: This diagram shows test time behavior: decoder output is fed in ·····> as next step's input

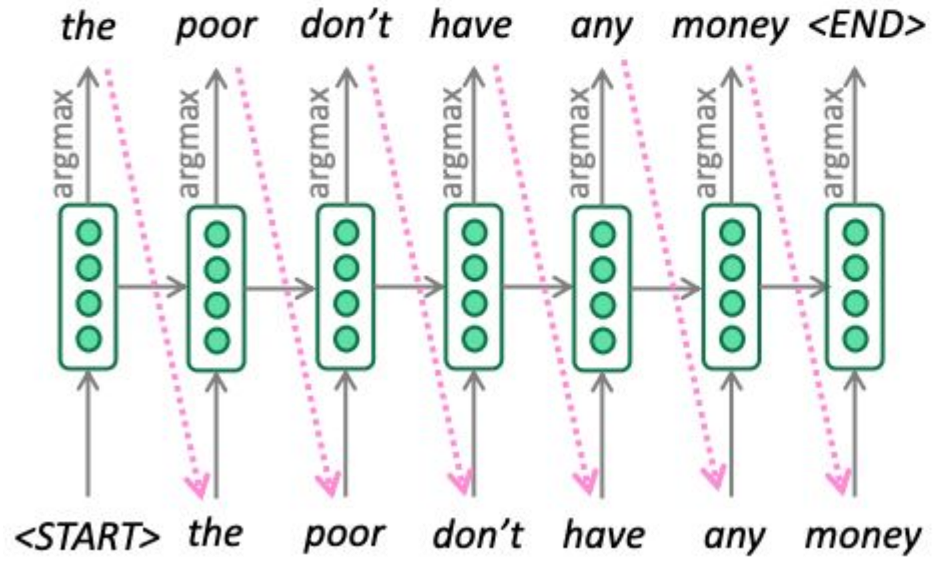
Training a Neural Machine Translation system


= negative log prob of "the"
= negative log prob of "have"
= negative log prob of <END>
$J = \frac{1}{T}\sum_{t=1}^{T} J_t = \boxed{J_1} + J_2 + J_3 + \boxed{J_4} + J_5 + J_6 + \boxed{J_7}$
$\hat{y}_1$ $\hat{y}_2$ $\hat{y}_3$ $\hat{y}_4$ $\hat{y}_5$ $\hat{y}_6$ $\hat{y}_7$
Encoder RNN
Decoder RNN
les pauvres sont démunis
<START> the poor don't have any money
Source sentence (from corpus)
Target sentence (from corpus)

Seq2seq is optimized as a **single system.**
Backpropagation operates *"end to end"*.

Greedy decoding

- Generate (or "decode") the target sentence by taking argmax on each step of the decoder
- That is, take most probable word on each step
- Use that as the input to the next step
- Keep going until you find <END> token or reach max_length



Better than greedy decoder

- Greedy decoding has no way to undo decisions!
- Better option: use beam search (a search algorithm) to explore several hypotheses and select the best one
- Beam search expands all possible next steps and keeps the k most likely, where k is a user-specified parameter.

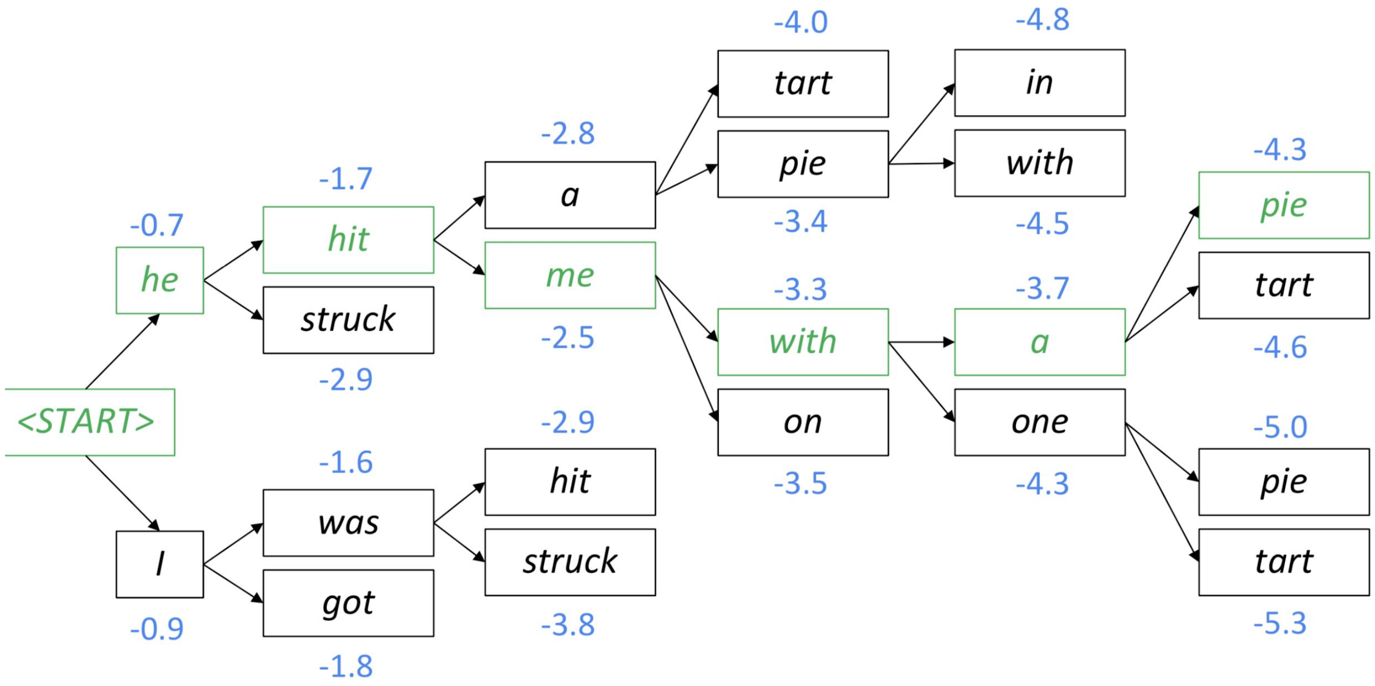
Beam search decoding

Search algorithm which aims to find **a high-probability** sequence by tracking multiple sequence at once.

Idea: On each step of the decoder, keep track of the **k** most probable partial sequences

Beam decoder example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Sampling based decoding

- Pure sample
  - On each step t, randomly sample from the probability distribution to obtain your next word
  - Like greedy decoding, but sample instead of taking the argmax
- Top-k sampling
  - On each step t, randomly sample from the probability distribution but restricted to just the top-k most probable words
  - We are truncating the probability distribution at every step t
  - k=1 is greedy search, K=V is pure sampling
- More efficient than beam search
- Good for creative generation (poetry, stories)

Advantages of Neural Machine Translation (NMT)

- Better performance
  - More fluent
  - Better use of context
  - Better use of phrase similarities
- A single neural network to be optimized end-to-end
  - No subcomponents to be individually optimized
- Requires much less human engineering effort
  - No feature engineering
  - Same method for all language pairs

Disadvantages of NMT?

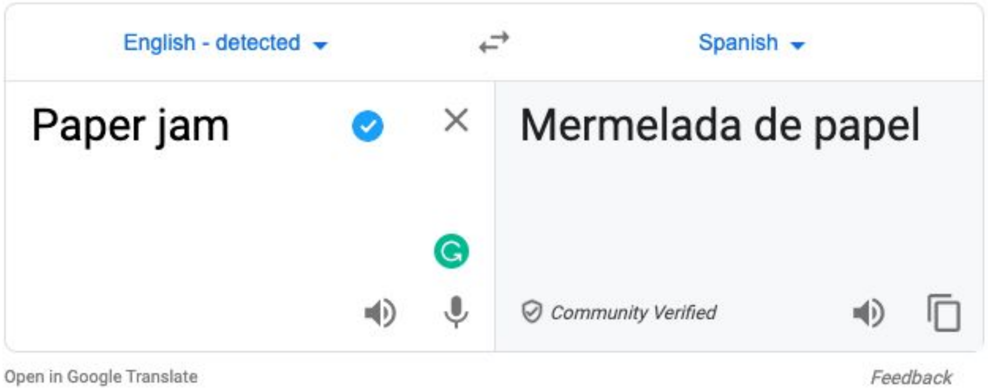
- NMT is less interpretable
  - Hard to debug
- NMT is difficult to control
  - For example, can’t easily specify rules or guidelines for translation
  - Safety concerns!

NMT: the biggest success story of NLP Deep Learning (before bert)

Neural Machine Translation went from a research activity in 2014 to the leading standard method in 2016

- 2014: First seq2seq paper published
- 2016: Google Translate switches to NMT
- Amazing!
    - Old system, built by hundreds of engineers over many years, outperformed by NMT systems trained by a handful of engineers in a few months

Is machine translation solved? nope!


English - detected

Spanish

Paper jam

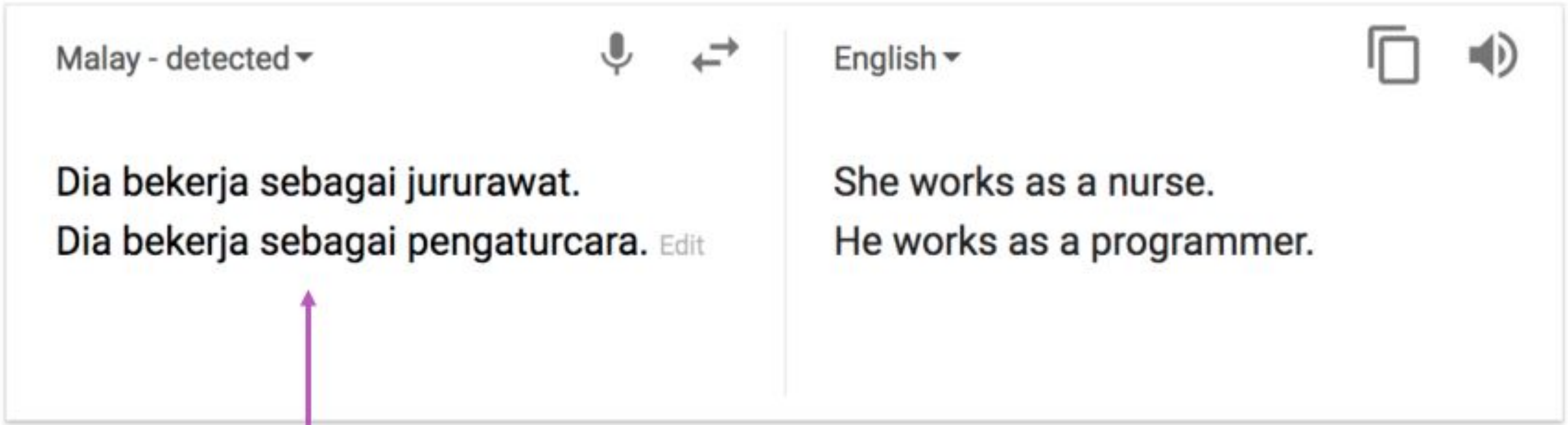
Mermelada de papel

Community Verified

Open in Google Translate

Feedback

Is machine translation solved? nope!



Dia bekerja sebagai jururawat.
Dia bekerja sebagai pengaturcara.

She works as a nurse.
He works as a programmer.

Didn’t specify gender

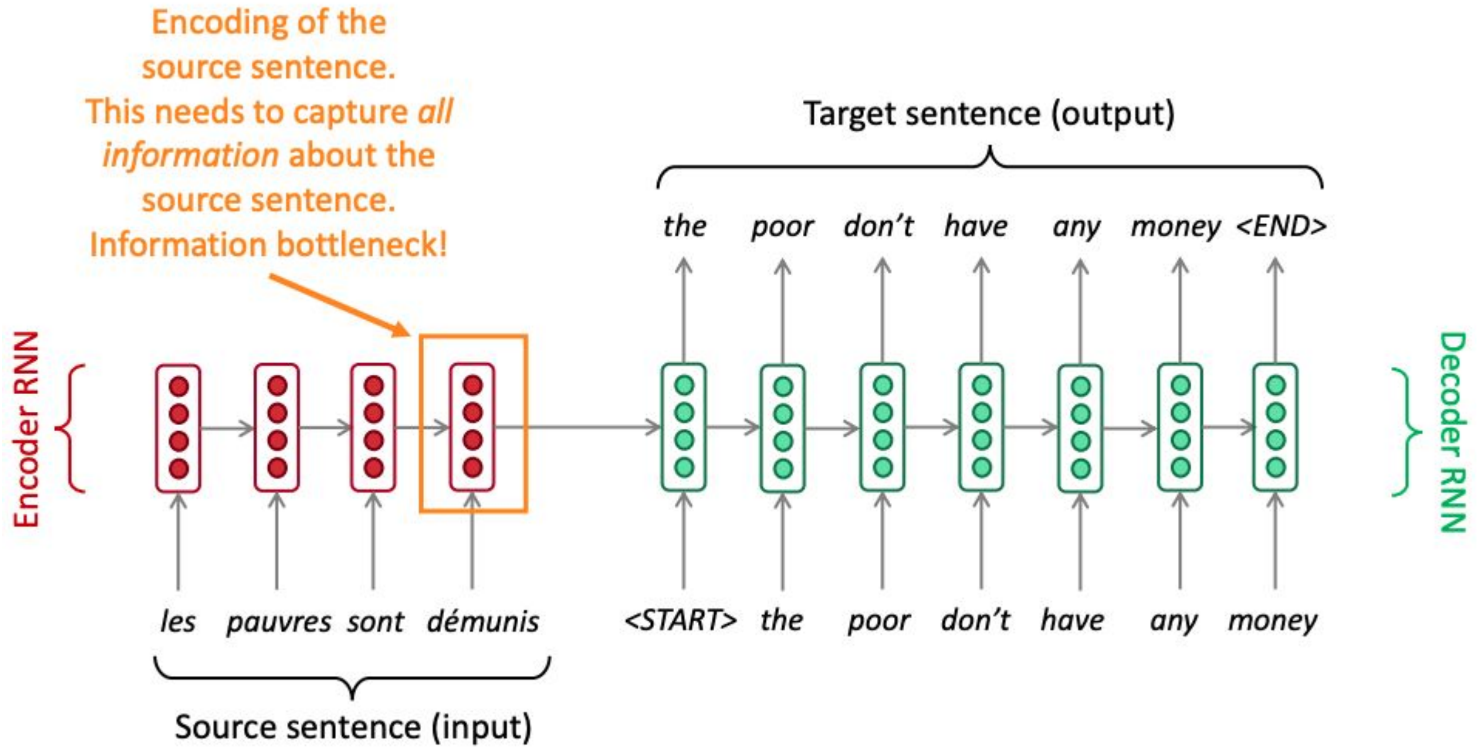
Attention

NMT with Attention

- In 2018: NMT research gets better
- Researchers have found many, many improvements to the “vanilla” seq2seq NMT
- But one improvement is so integral that it is the new vanilla…

ATTENTION

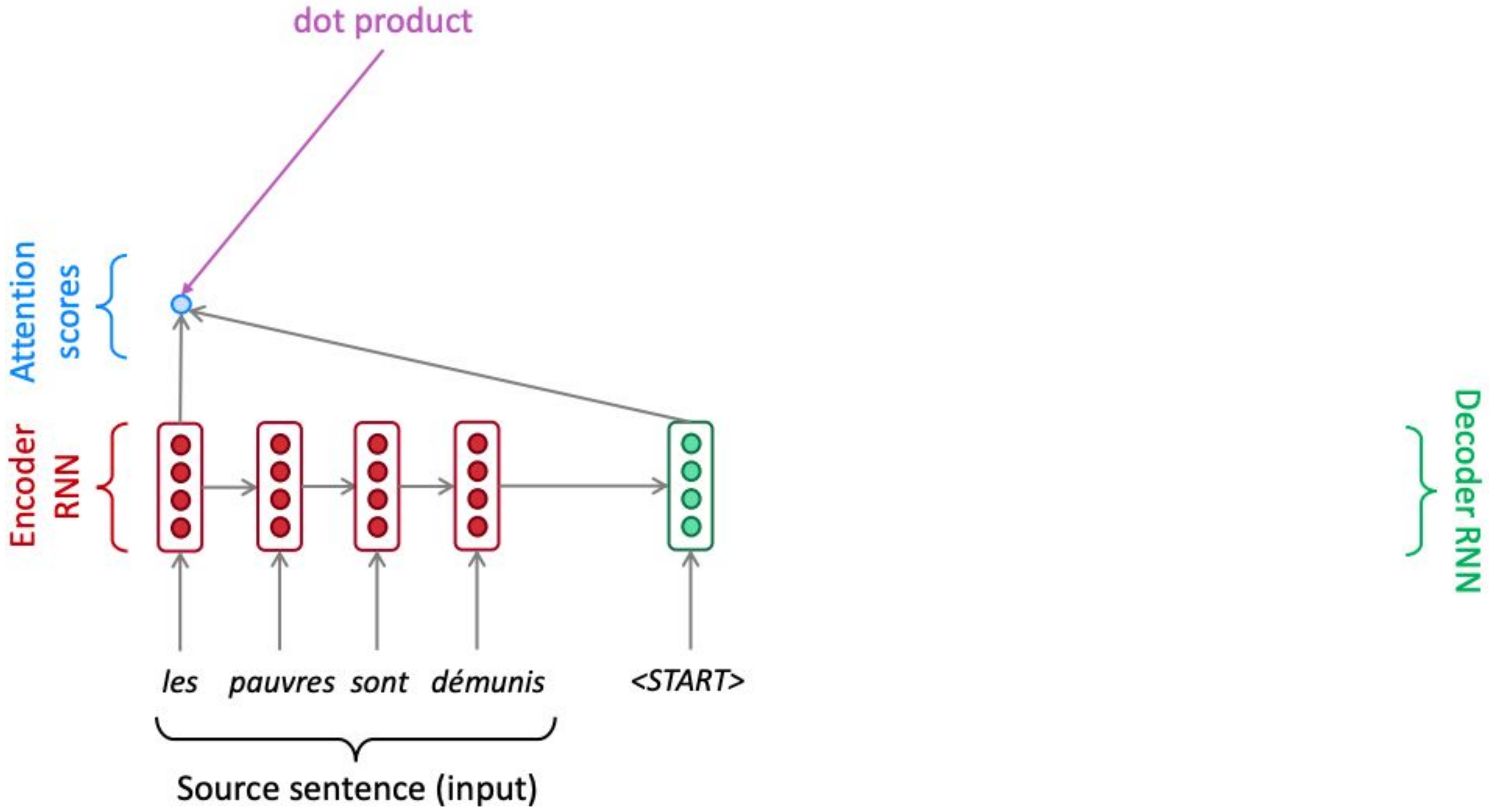
Sequence-to-sequence: the bottleneck problem


Encoding of the source sentence.
This needs to capture *all information* about the source sentence.
Information bottleneck!
Target sentence (output)
the poor don't have any money <END>
Encoder RNN
Decoder RNN
les pauvres sont démunis
<START> the poor don't have any money
Source sentence (input)

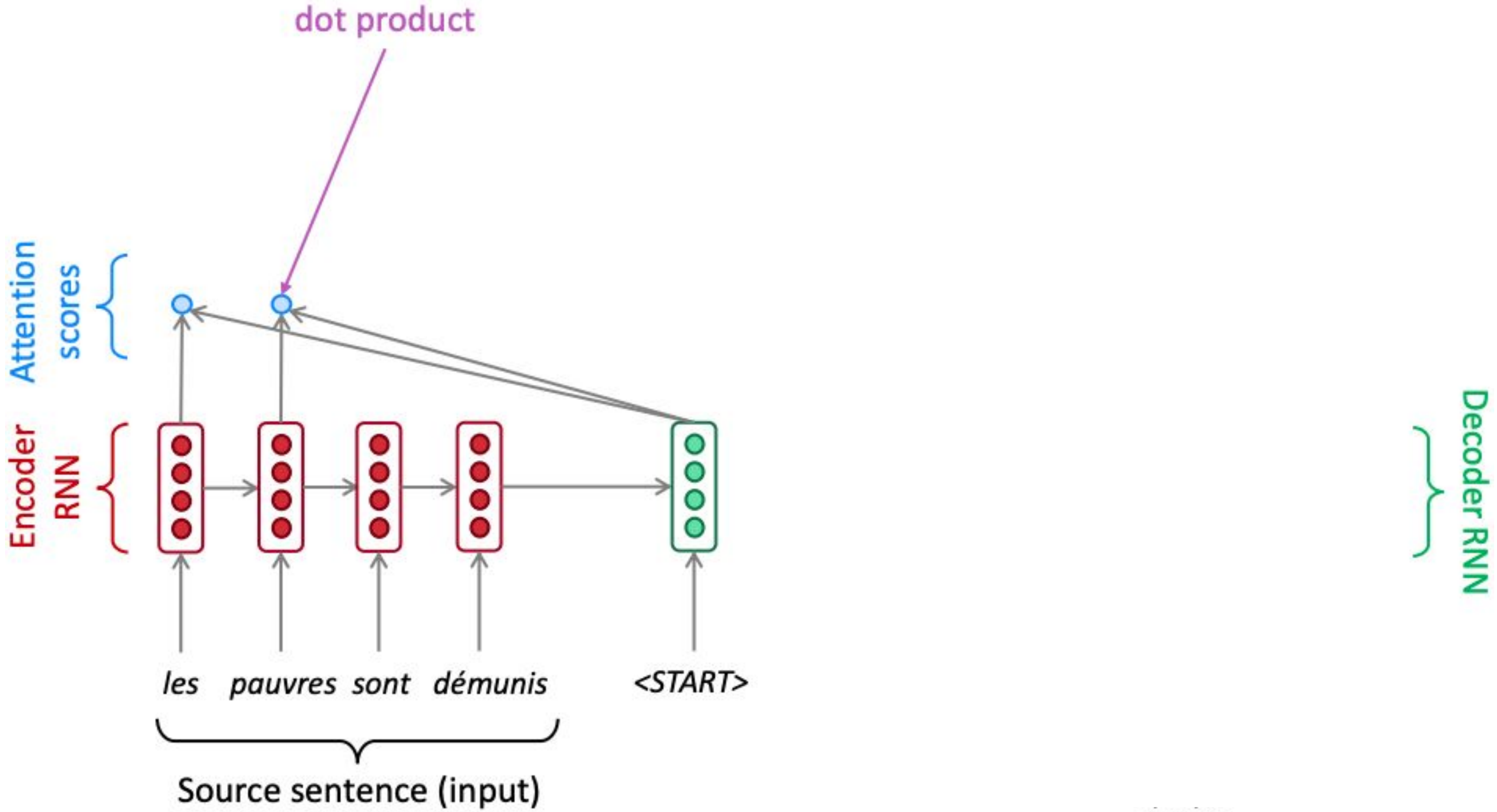
Attention

- Attention provides a solution to the bottleneck problem.
- Core idea: on each step of the decoder, focus on a particular part of the source sequence

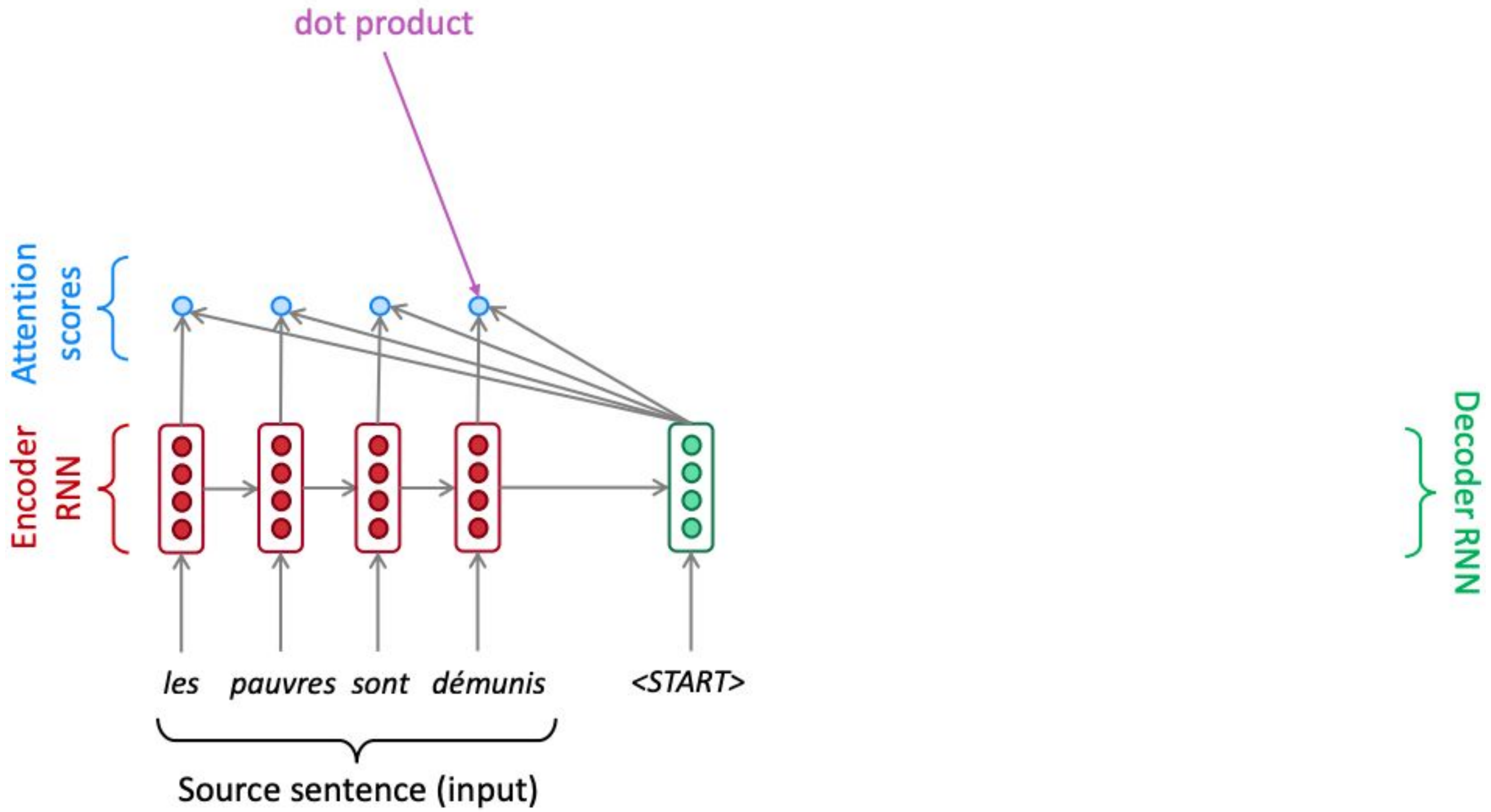
Sequence-to-sequence with attention



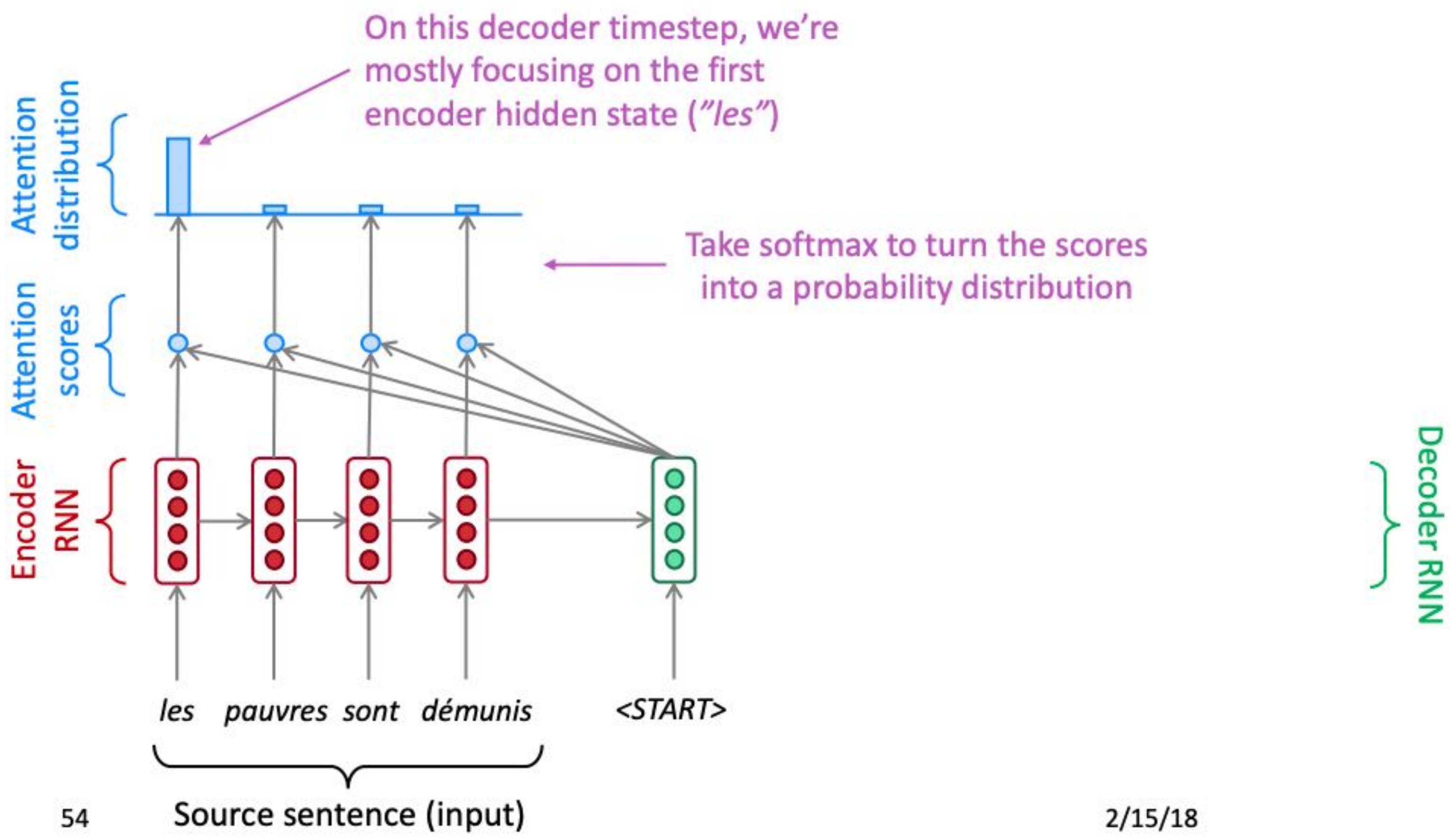
Sequence-to-sequence with attention



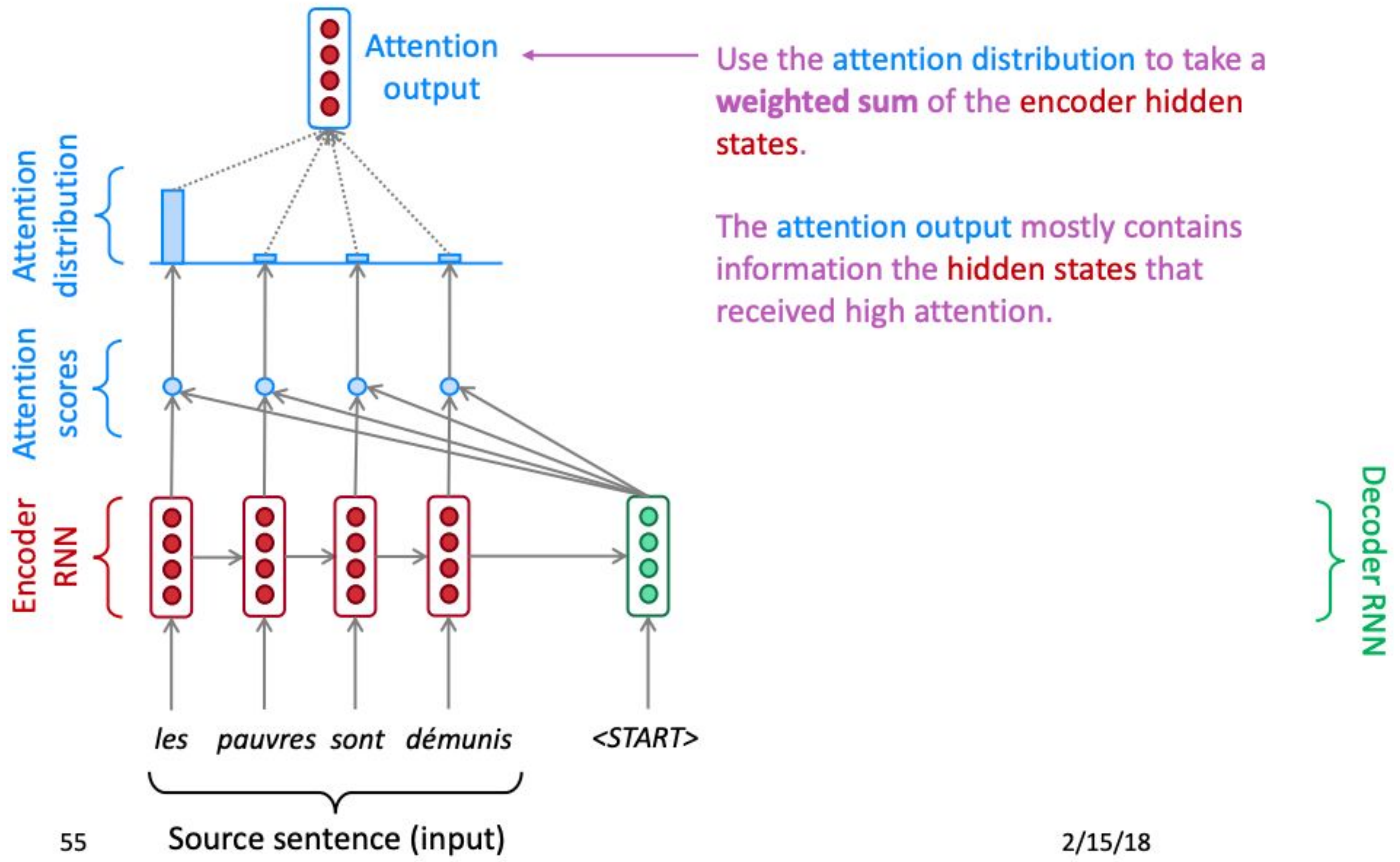
Sequence-to-sequence with attention



Sequence-to-sequence with attention


On this decoder timestep, we're mostly focusing on the first encoder hidden state ("les")
Take softmax to turn the scores into a probability distribution
Attention distribution
Attention scores
Encoder RNN
Decoder RNN
les
pauvres
sont
démunis
<START>
Source sentence (input)

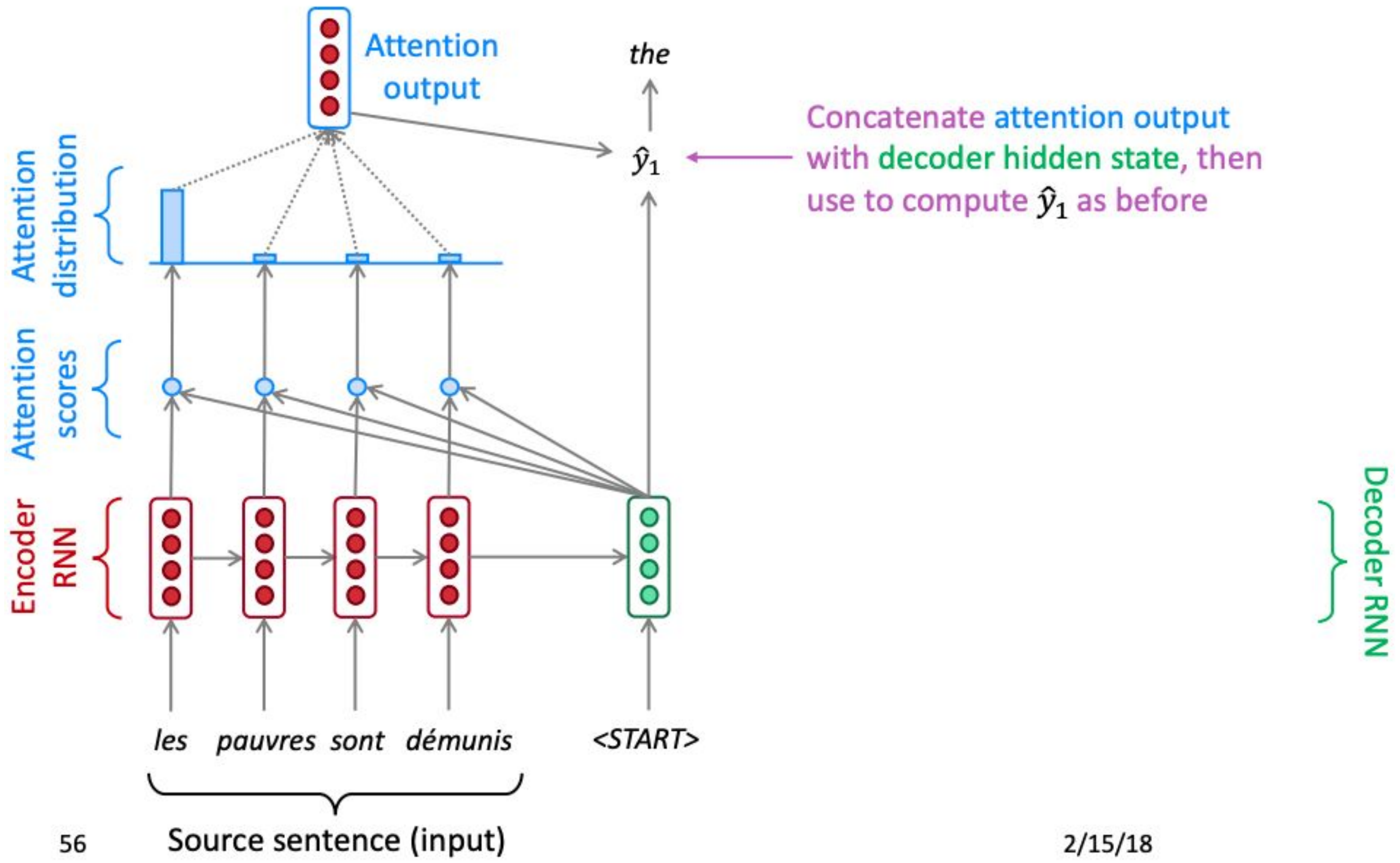
Sequence-to-sequence with attention



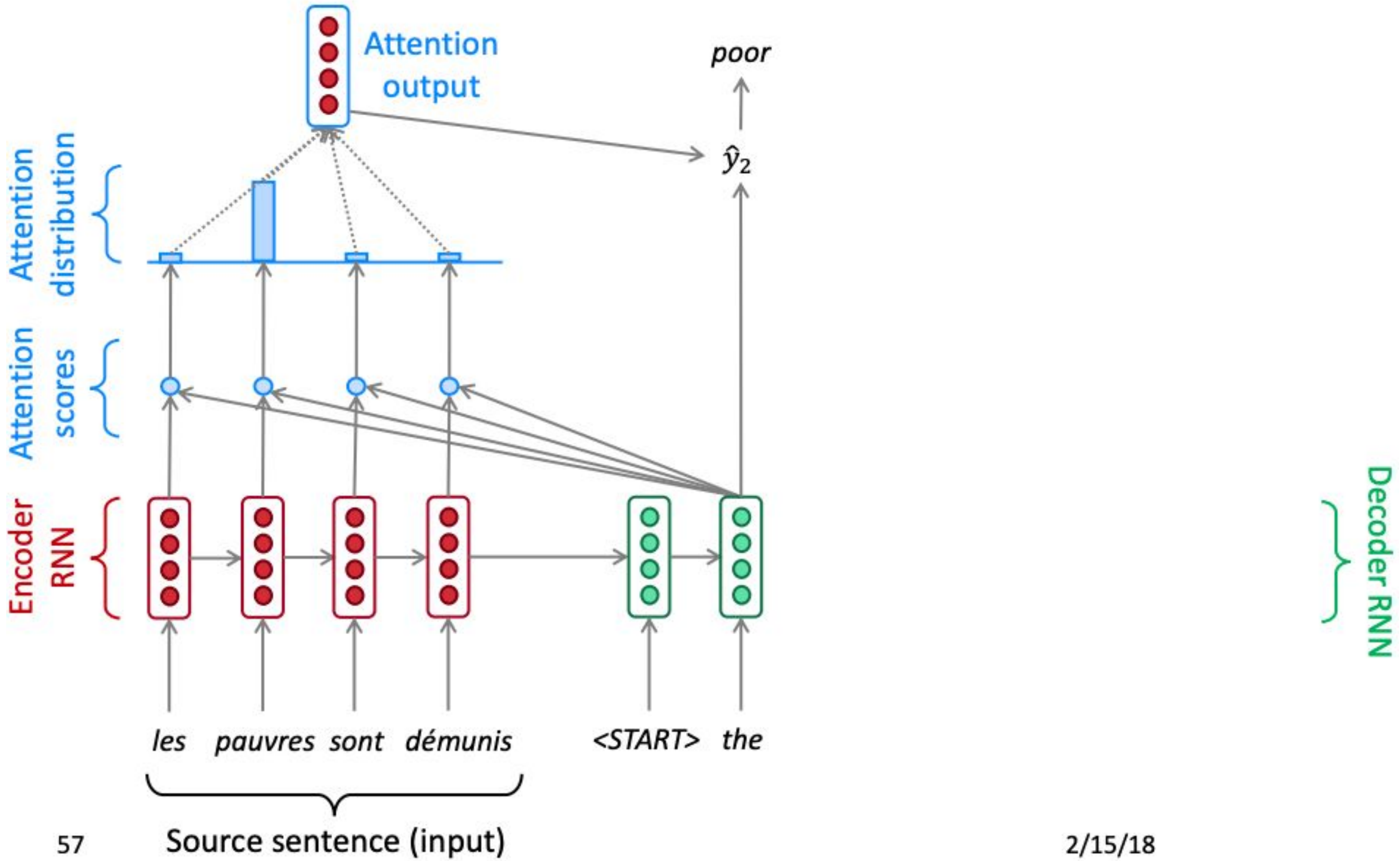
Use the attention distribution to take a **weighted sum** of the encoder hidden states.

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

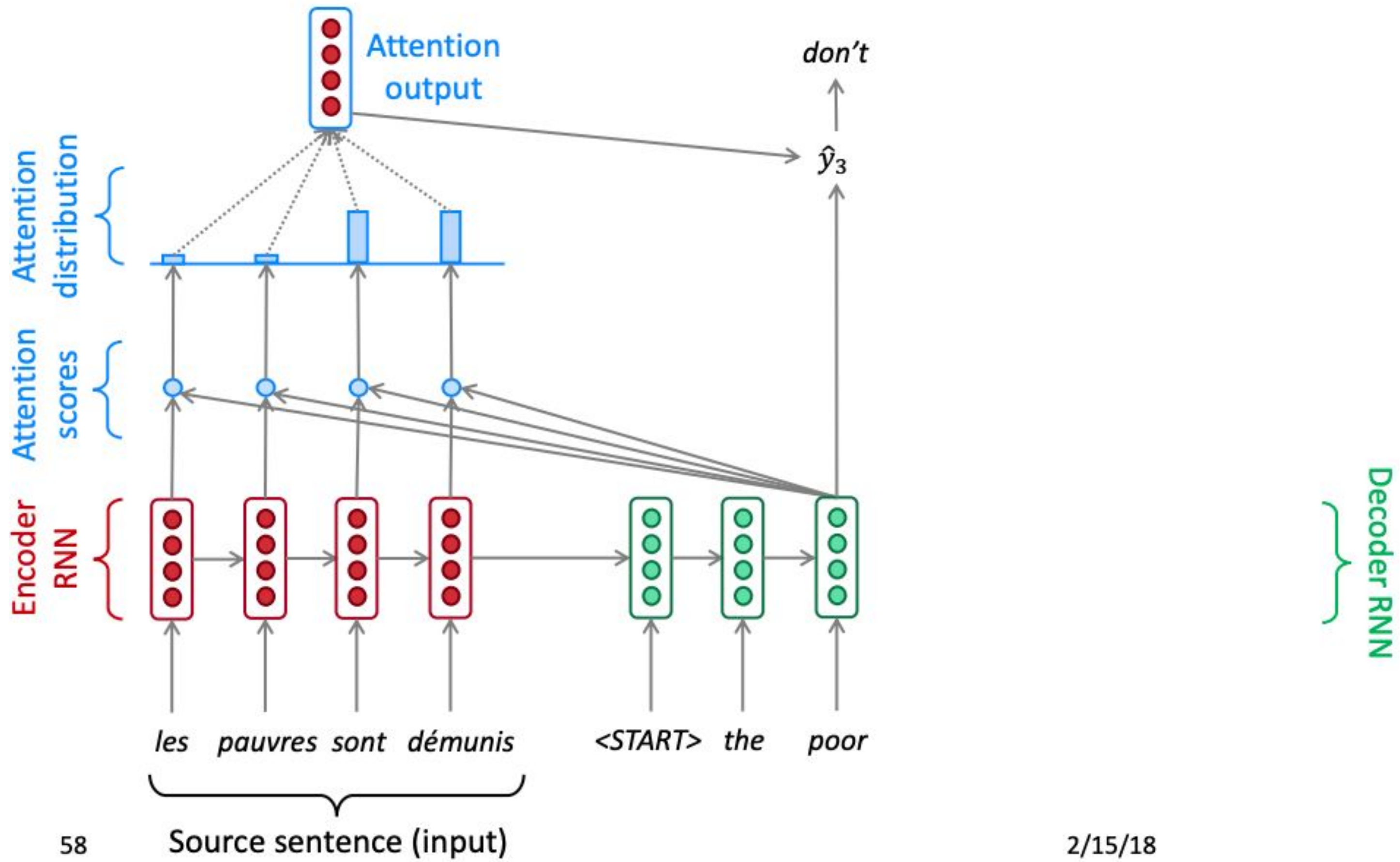
Sequence-to-sequence with attention



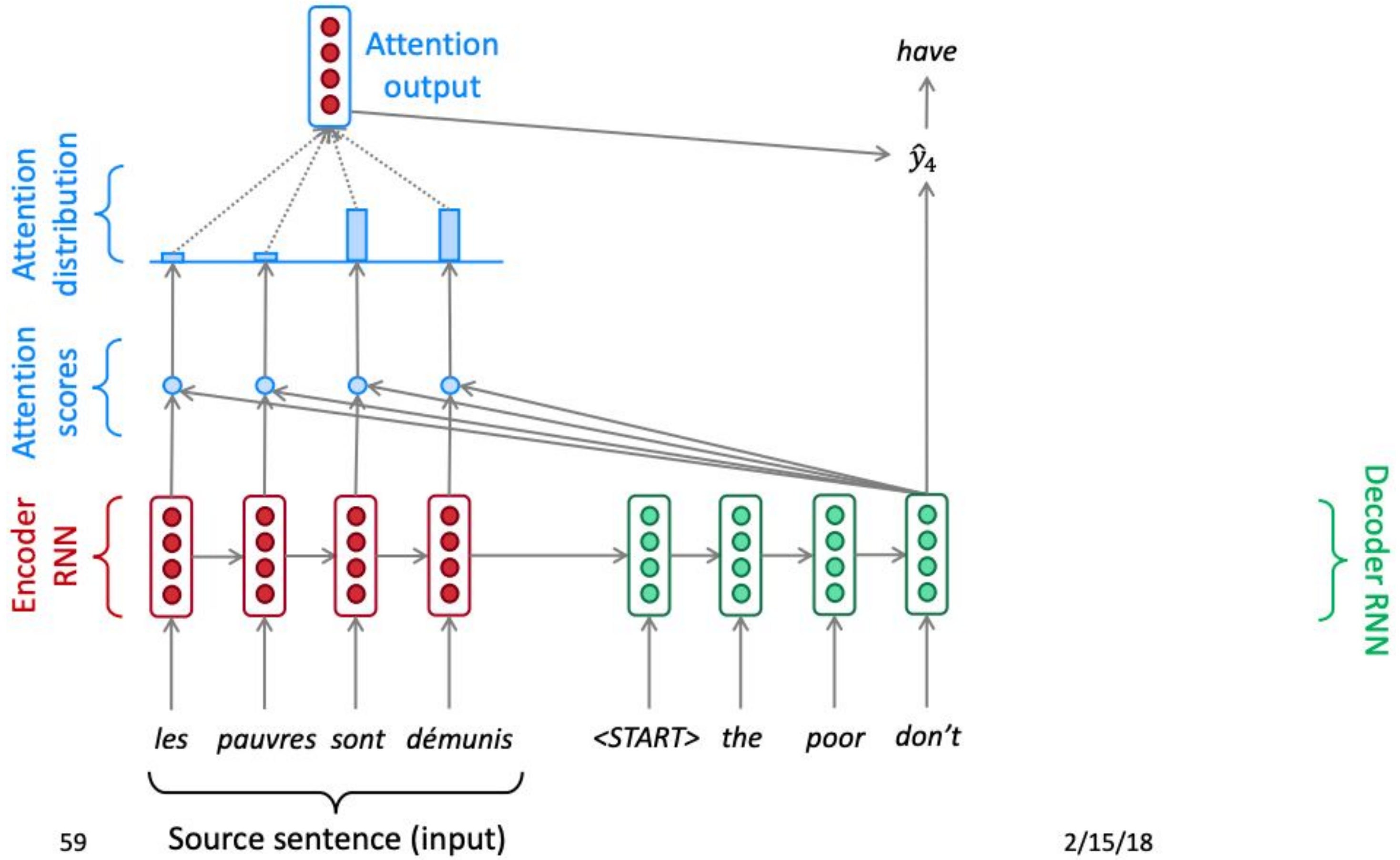
Sequence-to-sequence with attention



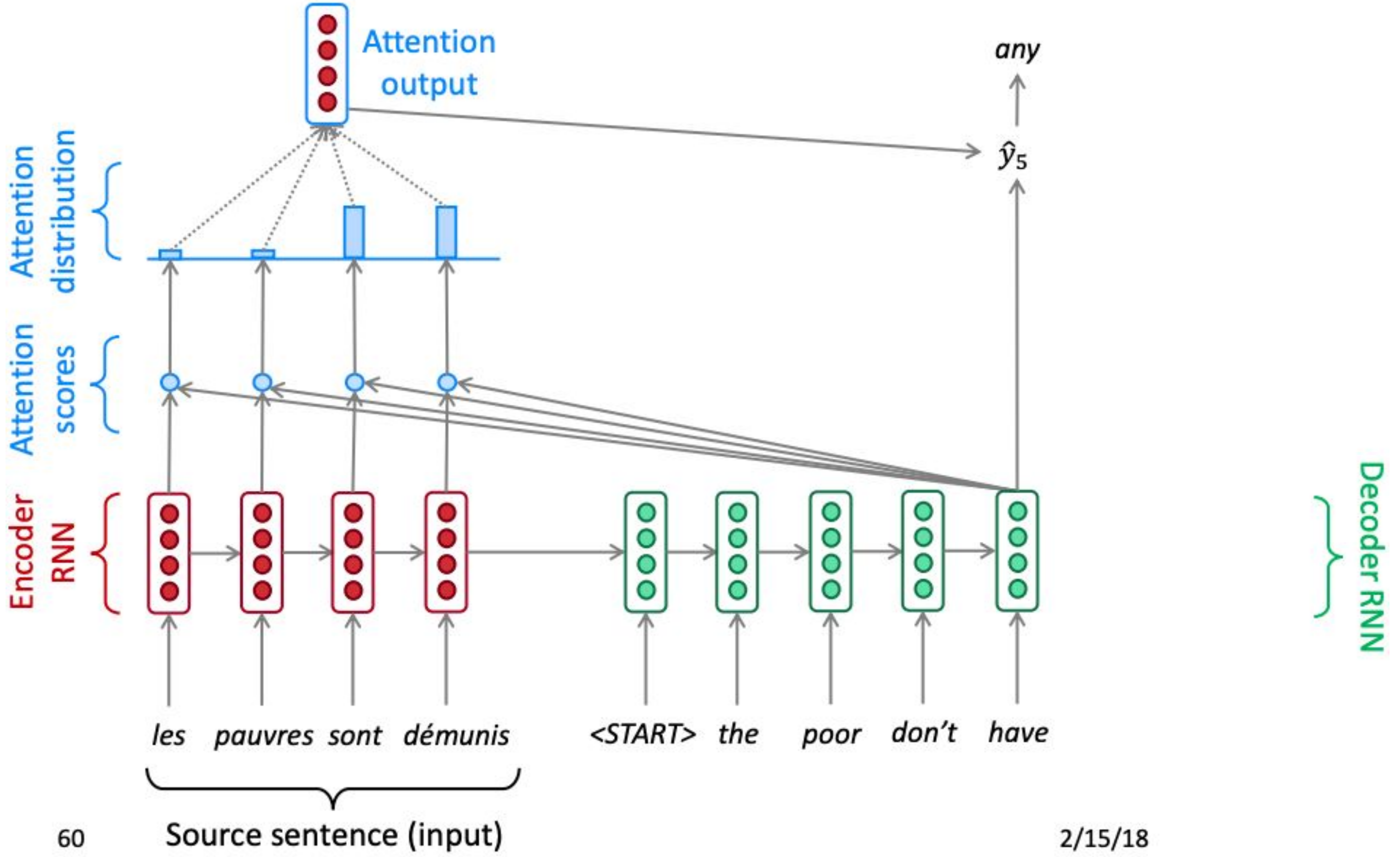
Sequence-to-sequence with attention



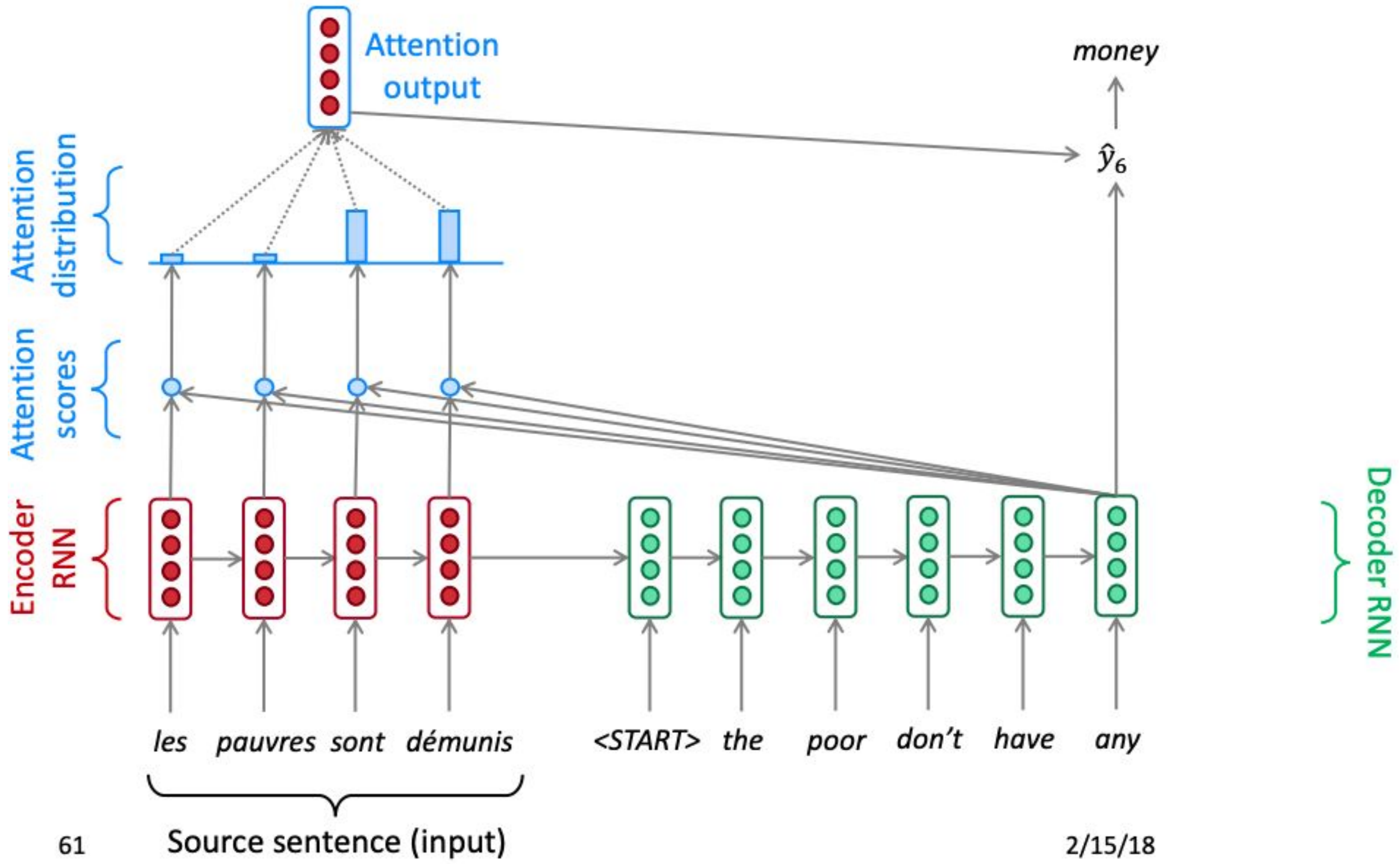
Sequence-to-sequence with attention



Sequence-to-sequence with attention



Sequence-to-sequence with attention



Attention in equations

- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep *t*, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores $\boldsymbol{e}^t$ for this step:

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \mathrm{softmax}(\boldsymbol{e}^t) \in \mathbb{R}^N$$

- We use α^t to take a weighted sum of the encoder hidden states to get the attention output $\boldsymbol{a}_t$

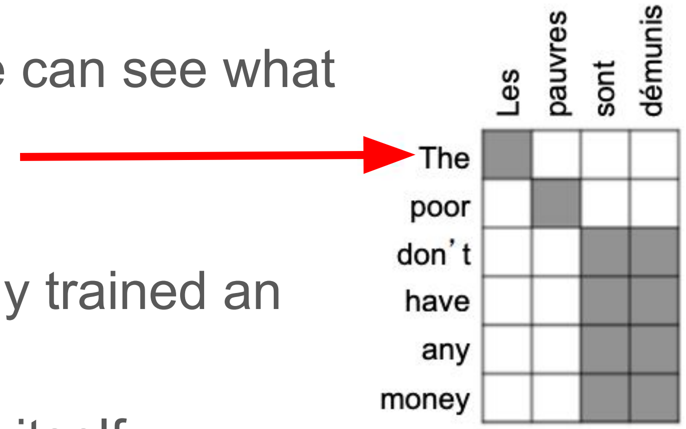
$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

- Finally we concatenate the attention output $\boldsymbol{a}_t$ with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

$$[\boldsymbol{a}_t; \boldsymbol{s}_t] \in \mathbb{R}^{2h}$$

Attention is great

- Attention significantly improves NMT performance
  - It's very useful to allow decoder to focus on certain parts of the source
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention provides some interpretability
  - By inspecting attention distribution, we can see what the decoder was focusing on
  - We get alignment for free!
  - This is cool because we never explicitly trained an alignment system
  - The network just learned alignment by itself



Bert / Transformers

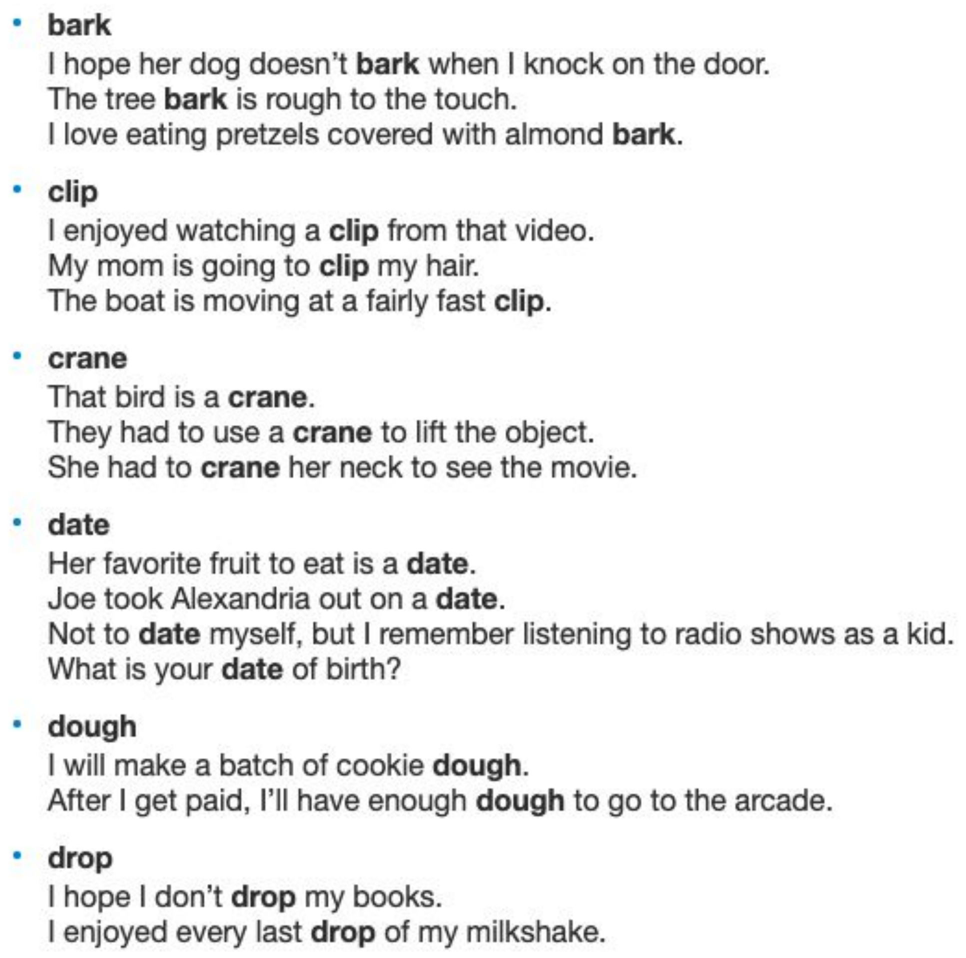
Pre-trained word vectors (~2014)

- Start with random word vectors and train them on our task of interest
- Or, use **pre-trained word vectors**
  - Trained on much more data (e.g. Wikipedia)
  - To initialize your LSTM or GRU
- Example of pre-trained embeddings:
  - Word2vec, GloVe, fastText

Representations of a word

- Up until now we have used one representation for each word
  - **One** word vector per word
- Problems with this approach:
  - Use the same representation for a **word** regardless of the context in which it occurs
  - Words have different aspects, senses, connotations
  - E.g. In WordNet `get` is mapped to over thirty different meanings (or senses)

Words with different meanings



- **bark**
  I hope her dog doesn't **bark** when I knock on the door.
  The tree **bark** is rough to the touch.
  I love eating pretzels covered with almond **bark**.
- **clip**
  I enjoyed watching a **clip** from that video.
  My mom is going to **clip** my hair.
  The boat is moving at a fairly fast **clip**.
- **crane**
  That bird is a **crane**.
  They had to use a **crane** to lift the object.
  She had to **crane** her neck to see the movie.
- **date**
  Her favorite fruit to eat is a **date**.
  Joe took Alexandria out on a **date**.
  Not to **date** myself, but I remember listening to radio shows as a kid.
  What is your **date** of birth?
- **dough**
  I will make a batch of cookie **dough**.
  After I get paid, I'll have enough **dough** to go to the arcade.
- **drop**
  I hope I don't **drop** my books.
  I enjoyed every last **drop** of my milkshake.

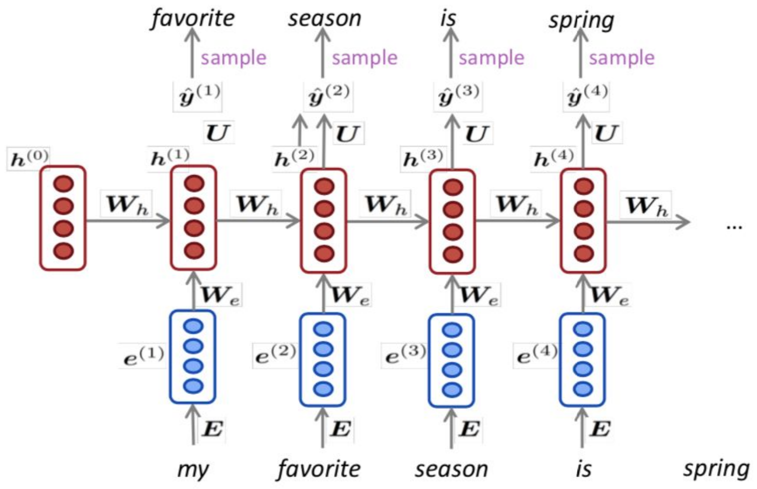
https://grammar.yourdictionary.com/for-students-and-parents/words-with-multiple-meanings.html

Can we learn **contextual word representations**?

What is an LSTM/ GRU doing?

When we train a language model with an LSTM we are producing **context-specific** representation at each position

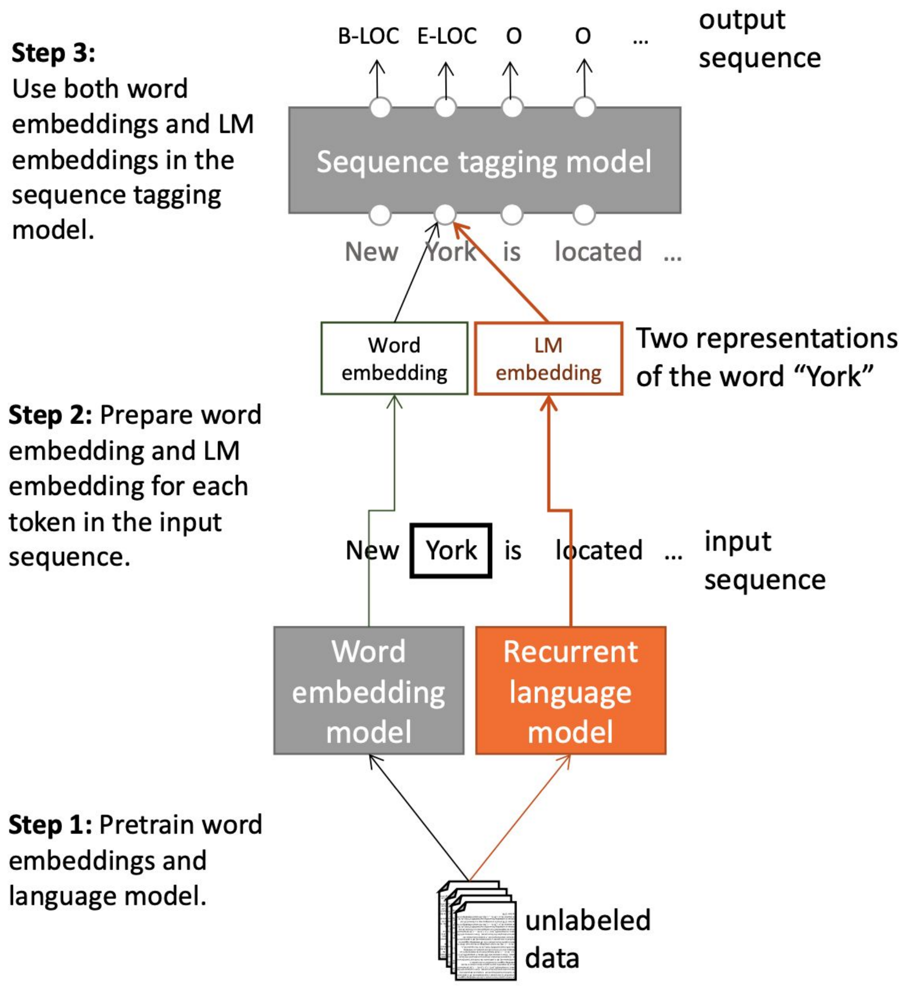
- We can have a "right" and a "left" representation per word



Tag ML

Task: train a named entity recognition classifier

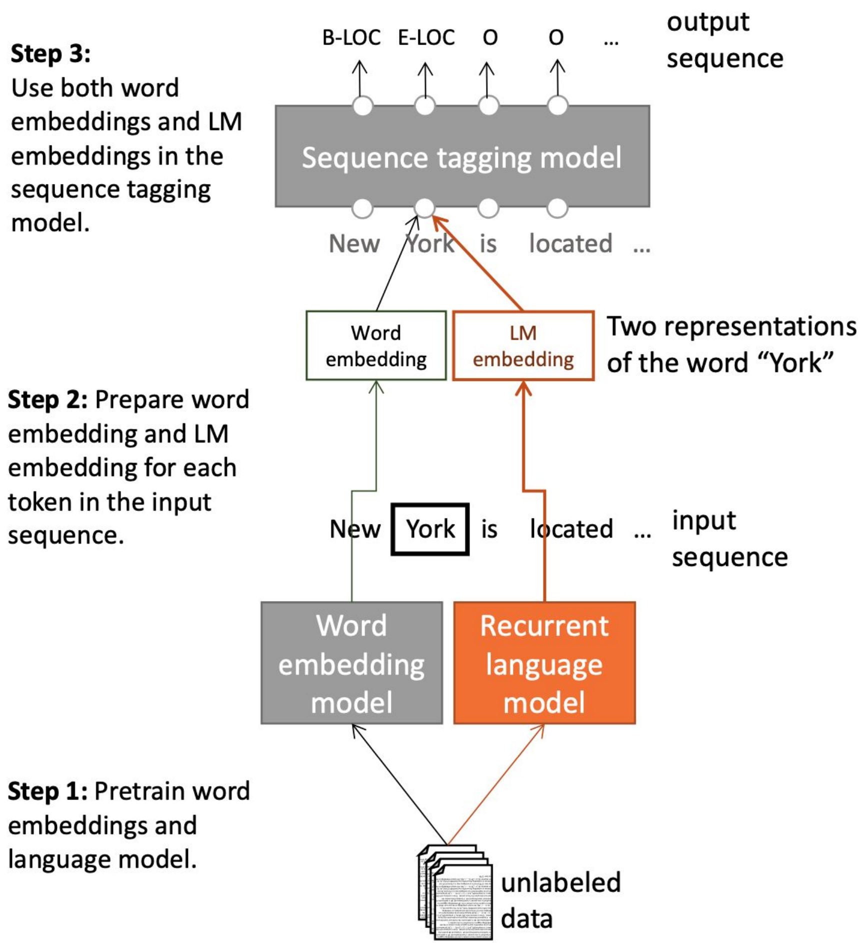
- Train word vector and language model from Wikipedia
- Meaning of word in context


Step 3:
Use both word embeddings and LM embeddings in the sequence tagging model.
B-LOC
E-LOC
O
O
...
output sequence
Sequence tagging model
New
York
is
located
...
Word embedding
LM embedding
Two representations of the word "York"
Step 2: Prepare word embedding and LM embedding for each token in the input sequence.
New
York
is
located
...
input sequence
Word embedding model
Recurrent language model
Step 1: Pretrain word embeddings and language model.
unlabeled data

https://arxiv.org/abs/1705.00108
Peters et all 2017

Tag ML

- Used two representations each word
  - From a word embedding
  - From a bidirectional LSTM (contextual)
- Train a sequence tagging model



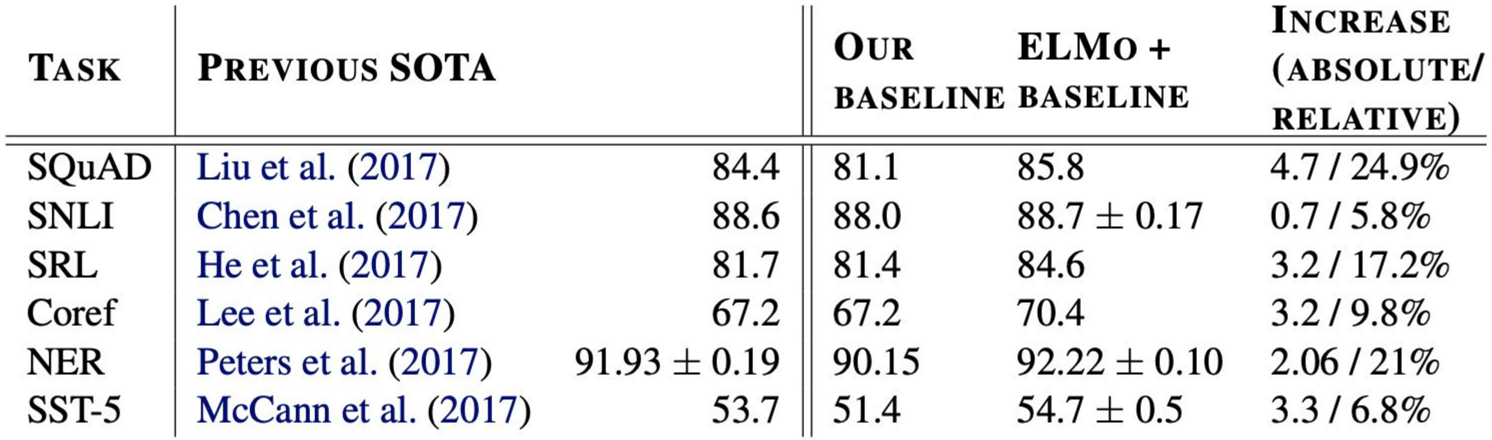
Step 3: Use both word embeddings and LM embeddings in the sequence tagging model.
B-LOC E-LOC O O ...
output sequence
Sequence tagging model
New York is located ...
Word embedding
LM embedding
Two representations of the word "York"
Step 2: Prepare word embedding and LM embedding for each token in the input sequence.
New York is located ...
input sequence
Word embedding model
Recurrent language model
Step 1: Pretrain word embeddings and language model.
unlabeled data

https://arxiv.org/abs/1705.00108
Peters et al. 2017

Elmo: Embeddings from Language Models

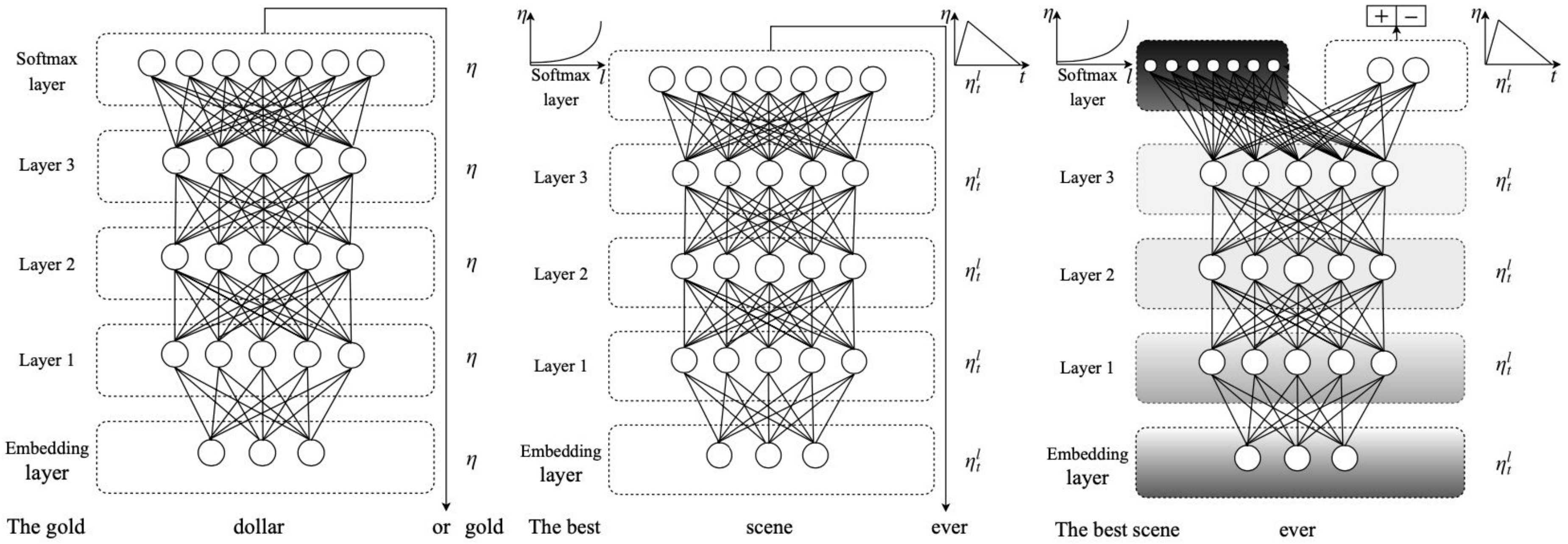
Peters et al. (2018) https://arxiv.org/abs/1802.05365

- A much more sophisticated version of TagML
- Contextual world embedding that are tasks specific
- They showed that you can use Elmo's representation for any other NLP tasks and get good gains



TASK	PREVIOUS SOTA		OUR BASELINE	ELMO + BASELINE	INCREASE (ABSOLUTE/ RELATIVE)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

ULMfit



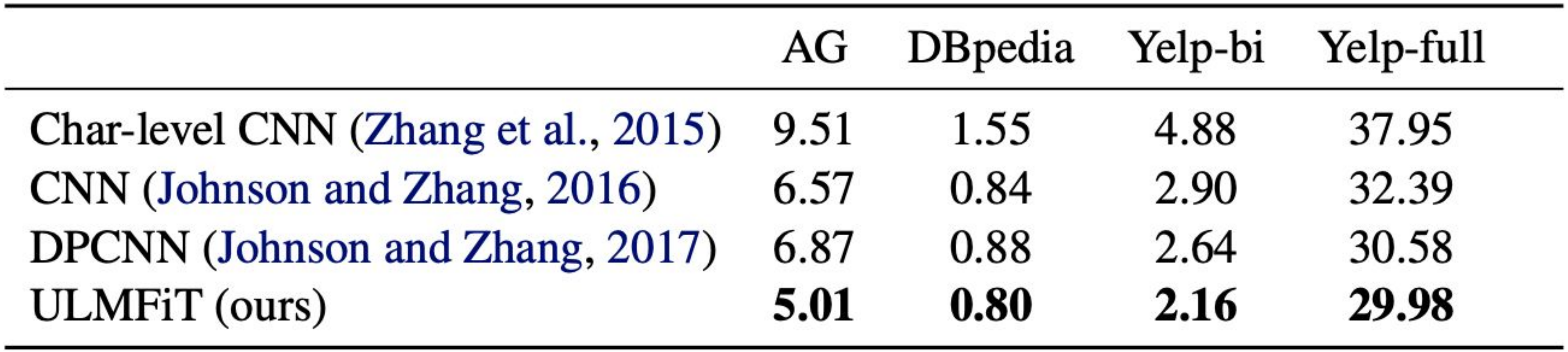
(a) LM pre-training (b) LM fine-tuning (c) Classifier fine-tuning

Howard and Ruder (2018) https://arxiv.org/abs/1801.06146

ULMfit: three stages

1. The LM is trained on a general-domain corpus to capture general features of the language in different layers.
2. The full LM is fine-tuned on target task
3. The classifier is fine-tuned on the target task
4. Steps 2,3 use gradual unfreezing, discriminative fine-tuning and slanted triangular learning

ULMfit results



	AG	DBpedia	Yelp-bi	Yelp-full
Char-level CNN (Zhang et al., 2015)	9.51	1.55	4.88	37.95
CNN (Johnson and Zhang, 2016)	6.57	0.84	2.90	32.39
DPCNN (Johnson and Zhang, 2017)	6.87	0.88	2.64	30.58
ULMFiT (ours)	**5.01**	**0.80**	**2.16**	**29.98**

Recent models

ULMfit

Jan 2018

Training:

1 GPU day

fast.ai

GPT

June 2018

Training

240 GPU days

OpenAI

BERT

Oct 2018

Training

256 TPU days

~320–560 GPU days

Google AI

GPT-2

Feb 2019

Training

~2048 TPU v3 days according to a reddit thread

OpenAI

Transformer models

ULMfit	GPT	BERT	GPT-2
Jan 2018	June 2018	Oct 2018	Feb 2019
Training:	Training	Training	Training
1 GPU day	240 GPU days	256 TPU days	~2048 TPU v3 days according to a reddit thread
		~320–560 GPU days	
fast.ai	OpenAI	Google AI	OpenAI

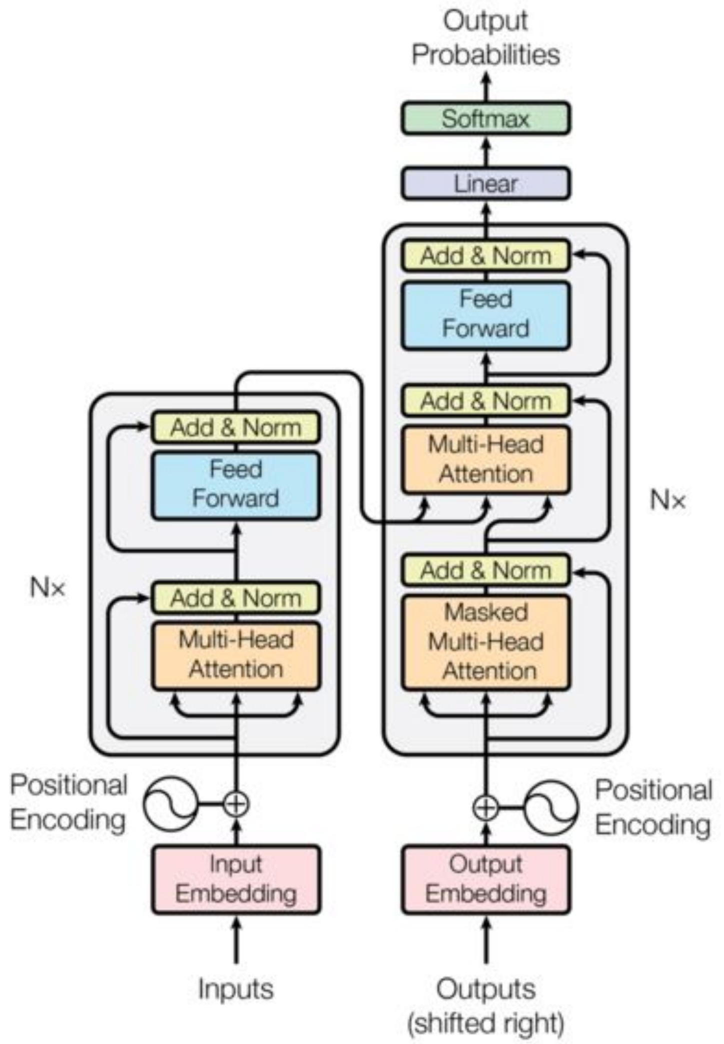
Motivation for Transformers

- Nice to have parallelization by RNNs are sequential
- RNNs still need attention mechanism to deal with long range dependencies
- Maybe we just need **Attention**

Transformer overview

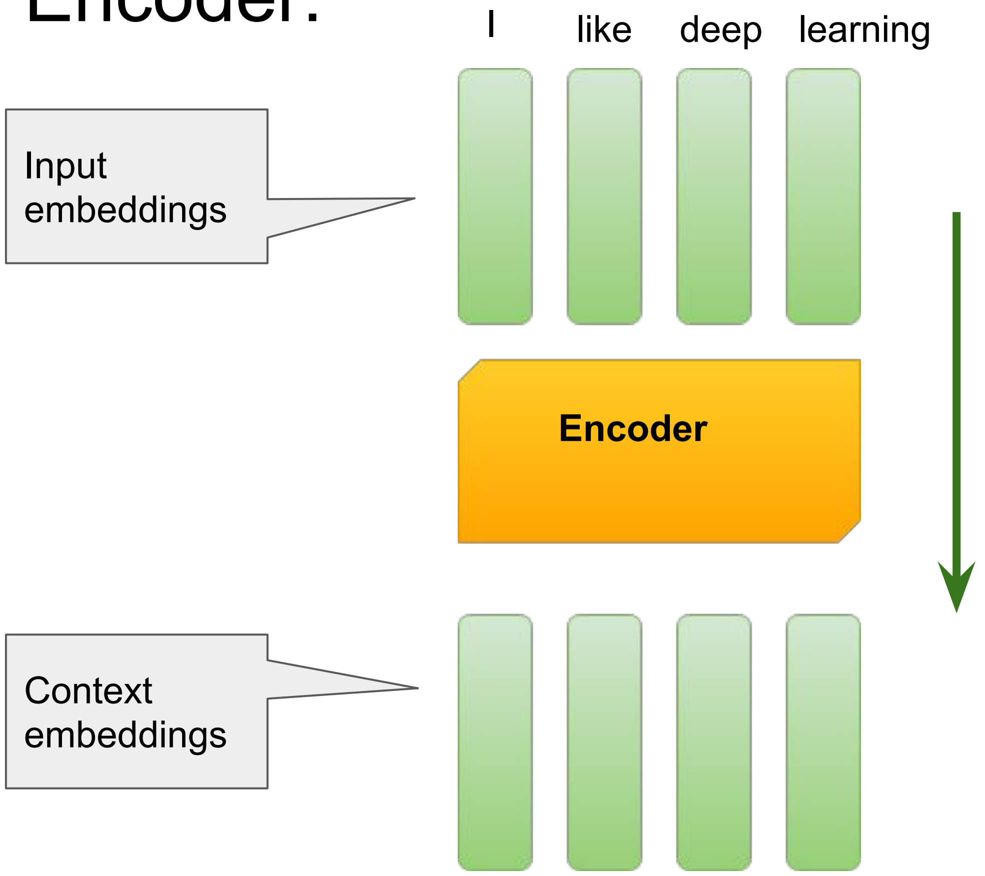
- Encoder-Decoder architecture
- Non-recurrent sequence to sequence model
- Task: machine translation
- Cross-entropy loss on top of softmax classifier

https://arxiv.org/abs/1706.03762 2017

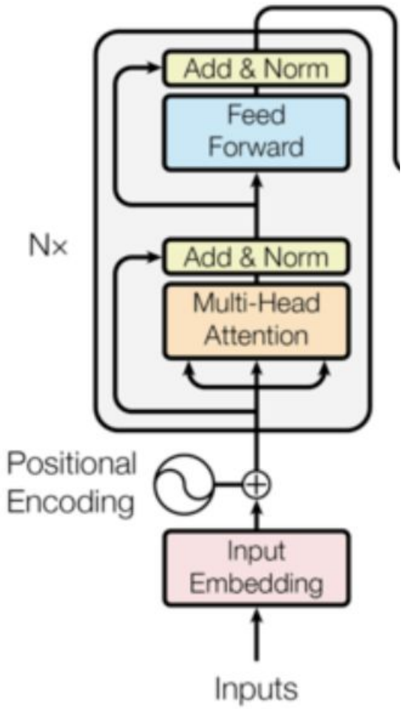


Encoder

Encoder:



I
like
deep
learning
Input embeddings
Encoder
Context embeddings



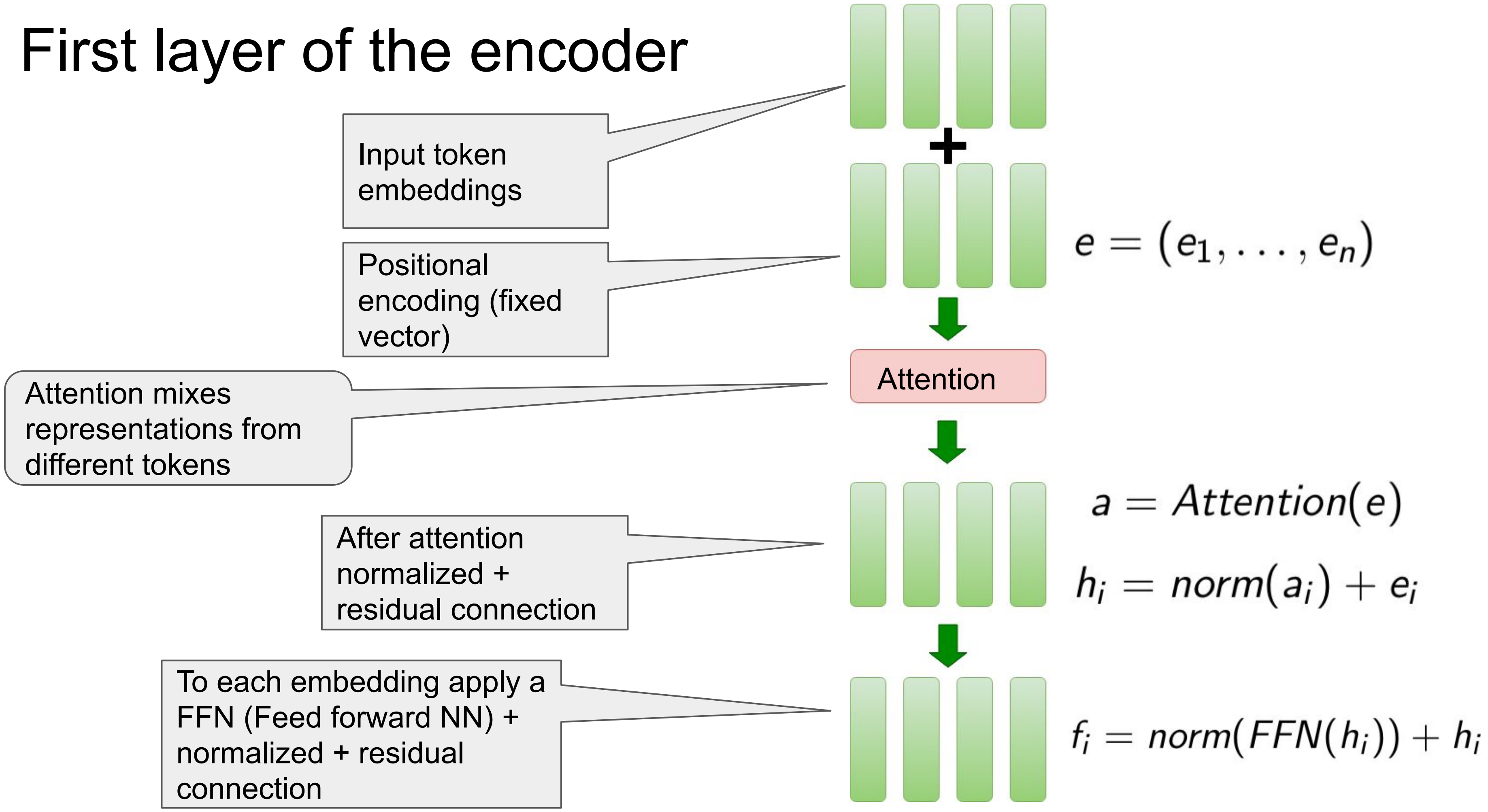
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Input Embedding
Inputs

Transformer pointers

Learning about transformers on your own?

- http://nlp.seas.harvard.edu/2018/04/03/attention.html
- The Annotated Transformer by Sasha Rush
- A Jupyter Notebook using PyTorch that explains everything!

First layer of the encoder



Input token embeddings

Positional encoding (fixed vector)

Attention mixes representations from different tokens

After attention normalized + residual connection

To each embedding apply a FFN (Feed forward NN) + normalized + residual connection

$$e = (e_1, \ldots, e_n)$$

$$a = Attention(e)$$

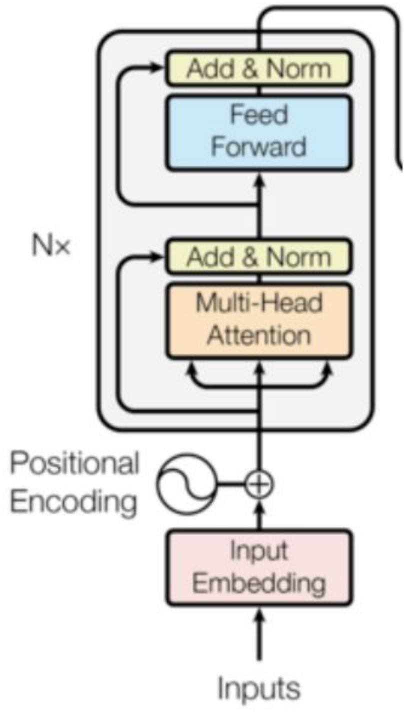
$$h_i = norm(a_i) + e_i$$

$$f_i = norm(FFN(h_i)) + h_i$$

N=6 encoder layers?

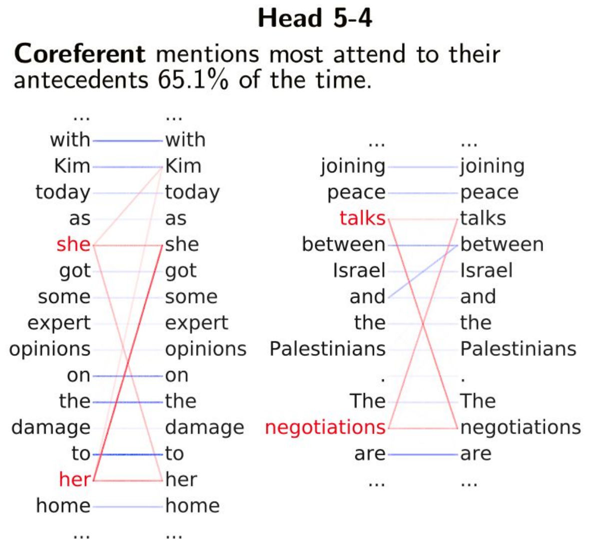
```
def encoder_layer($e = (e_1, \dots, e_n)$):
    $a = Attention(e)$
    $h_i = norm(a_i) + e_i$
    $f_i = norm(FFN(h_i)) + h_i$
    return $f = (f_1, \dots, f_n)$

def encoder($e = (e_1, \dots, e_n)$):
    for $k$ in range(N):
        $e$ = encoder_layer($e$)
    return $e$
```



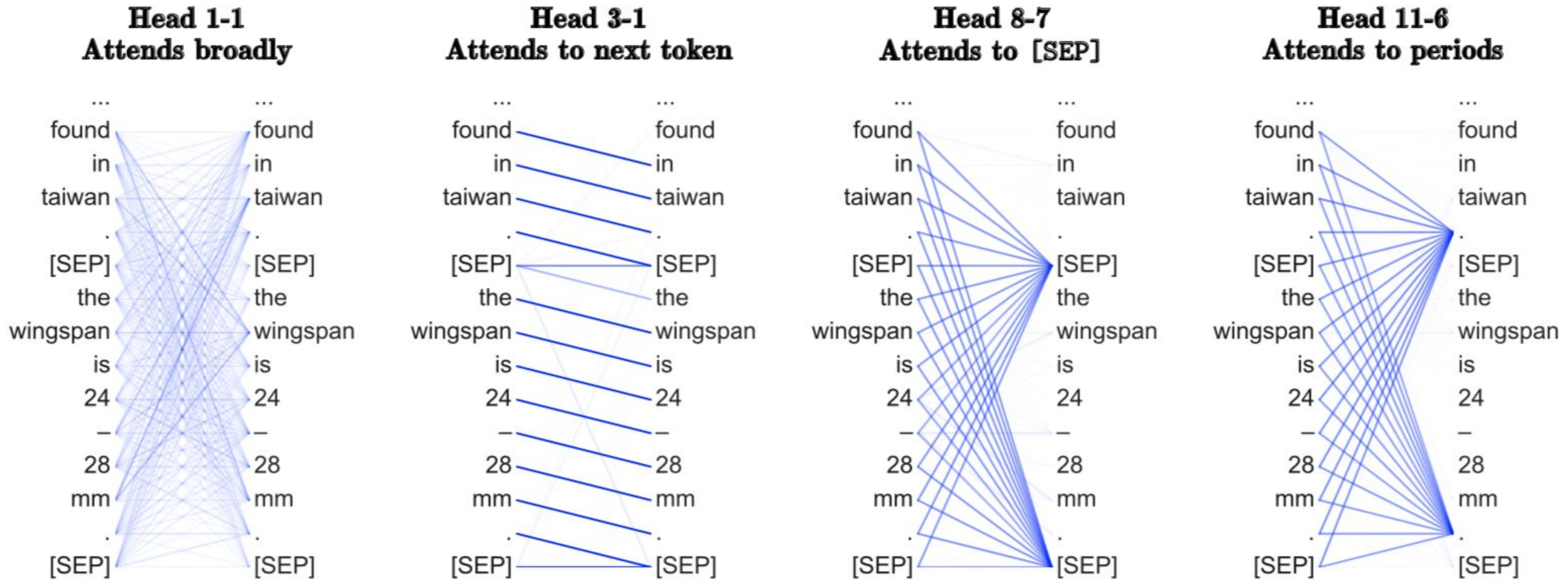
(Self) Attention

- How much the model should “pay attention to” the word X when computing the representation of word XX?
- Transformers use (train) multiple attention “heads”. Each attention head can capture a different relationship between words
    - Coreference: which mentions of an entity refer to the same entity



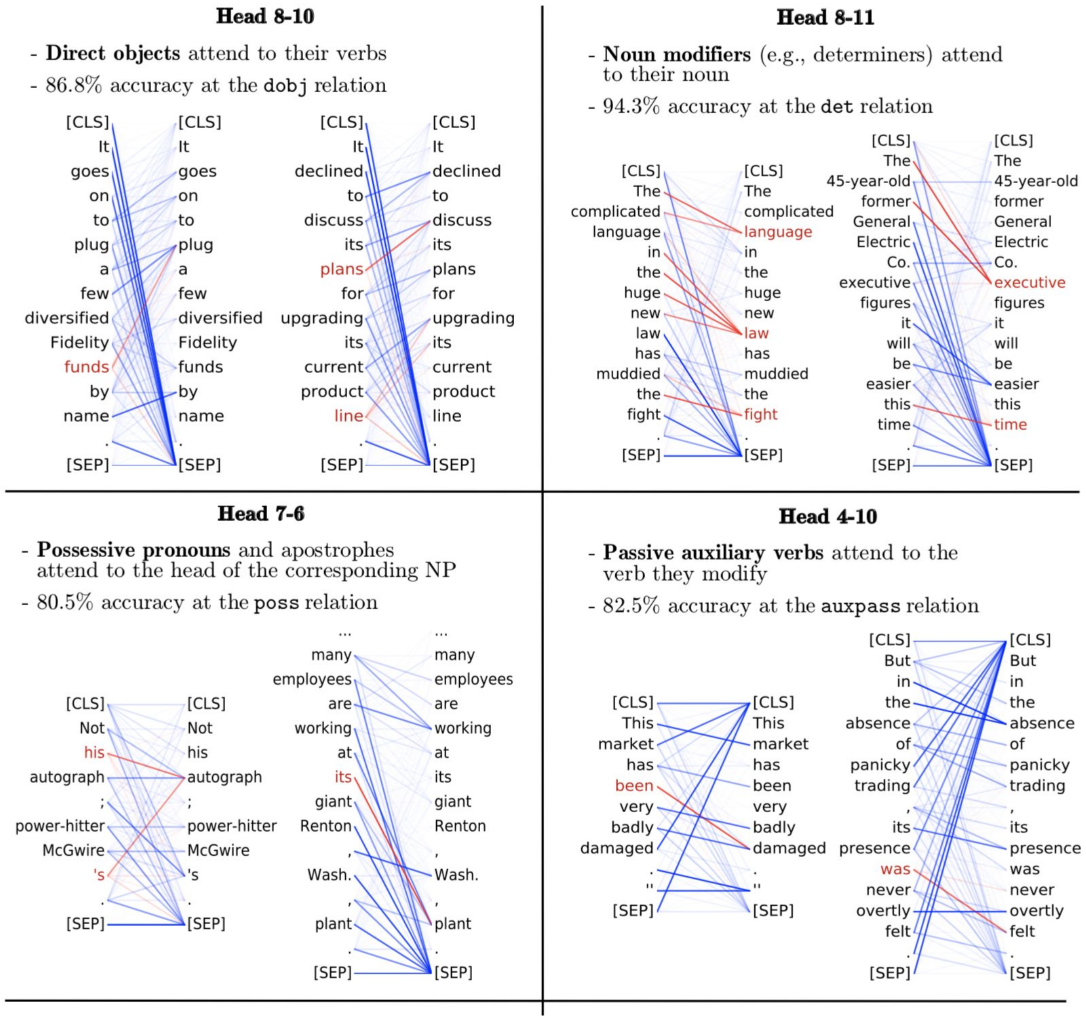
Head 5-4
Coreferent mentions most attend to their antecedents 65.1% of the time.
... with Kim today as she got some expert opinions on the damage to her home ...
... with Kim today as she got some expert opinions on the damage to her home ...
... joining peace talks between Israel and the Palestinians . The negotiations are ...
... joining peace talks between Israel and the Palestinians . The negotiations are ...

Self attention:



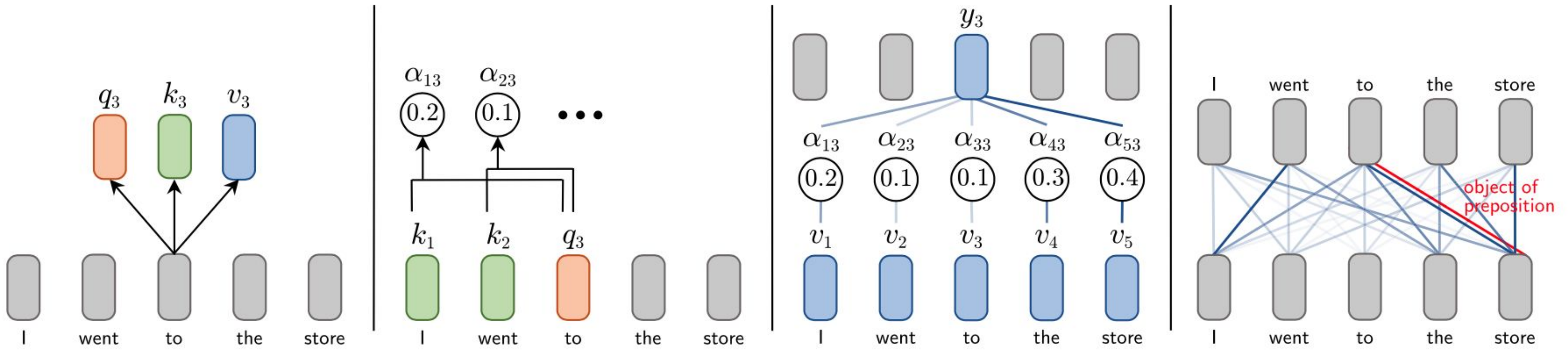
https://arxiv.org/abs/1906.04341

BERT attention heads that correspond to linguistic phenomena


Head 8-10
- Direct objects attend to their verbs
- 86.8% accuracy at the dobj relation
[CLS] It goes on to plug a few diversified Fidelity funds by name . [SEP]
[CLS] It declined to discuss its plans for upgrading its current product line . [SEP]
Head 8-11
- Noun modifiers (e.g., determiners) attend to their noun
- 94.3% accuracy at the det relation
[CLS] The complicated language in the huge new law has muddied the fight . [SEP]
[CLS] The 45-year-old former General Electric Co. executive figures it will be easier this time . [SEP]
Head 7-6
- Possessive pronouns and apostrophes attend to the head of the corresponding NP
- 80.5% accuracy at the poss relation
[CLS] Not his autograph ; power-hitter McGwire 's . [SEP]
... many employees are working at its giant Renton , Wash. , plant . [SEP]
Head 4-10
- Passive auxiliary verbs attend to the verb they modify
- 82.5% accuracy at the auxpass relation
[CLS] This market has been very badly damaged . '' [SEP]
[CLS] But in the absence of panicky trading , its presence was never overtly felt . [SEP]

https://arxiv.org/abs/1906.04341

Attention mechanism in transformers



Each input vector is linearly transformed into query, key, and value vectors

Attention weights are normalized inner products of query and key vectors

Outputs are weighted sums of value vectors

After training, the attention weights can be compared with linguistic annotations

https://www.pnas.org/content/117/48/30046/tab-article-info

Self Attention: computing **query, key, value**

- An attention head takes as input a sequence of vectors $e = (e_1, \ldots, e_n)$ corresponding to the n tokens of the input sentence.
- Each vector e_i is transformed into query, key and value vectors q_i, k_i, v_i through separate linear transformations:

$$q_i = W^q \cdot e_i + b^q$$
$$k_i = W^k \cdot e_i + b^k$$
$$v_i = W^v \cdot e_i + b^v$$

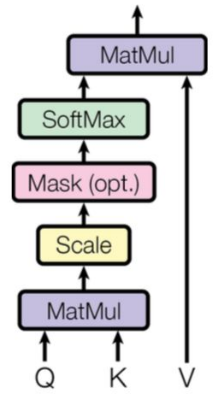
Self Attention: one head

- An attention head takes as input a sequence of vectors $e = (e_1, \dots, e_n)$ corresponding to the n tokens of the input sentence.
- Each vector e_i is transformed into query, key and value vectors q_i, k_i, v_i through separate linear transformations.
- Attention weights α are computed between all pairs of words as softmax-normalized dot products between the query and key vectors:

$$\alpha_{ij} = \frac{\exp(q_i \cdot k_j)}{\sum_{l=1}^{n} \exp(q_i \cdot k_l)}$$

- The output o of the attention head is a weighted sum of the value vectors: $o_i = \sum_{j=1}^{n} \alpha_{ij} v_j$

Scaled Dot-Product Attention



Scaling Attention weights

In the Transformer paper Attention weights α are computed with the following formula

$$\alpha_{ij} = \frac{\exp(q_i \cdot k_j / \sqrt{d_k})}{\sum_{l=1}^{n} \exp(q_i \cdot k_l / \sqrt{d_k})}$$

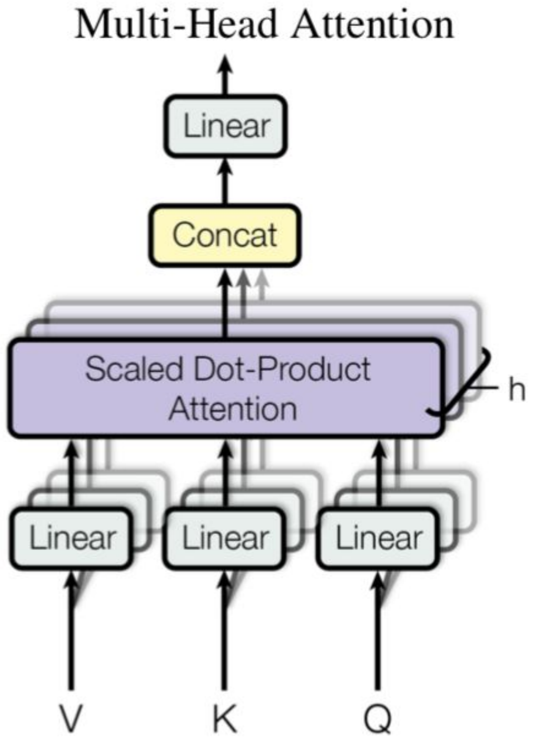
where d_k is the dimension of the query and key vectors. Note that we can rewrite the vector $\alpha_i = (\alpha_{i1}, \ldots, \alpha_{in})$ as

$$\alpha_i = softmax(\frac{K^T \cdot q_i}{\sqrt{d_k}})$$

where K is a matrix in which each column is the vector k_j. Note also that $\sum_j \alpha_{ij} = 1$

Self Attention multi-heads

- Apply attention `h` times
- Concatenate the outputs per token
- Apply a linear layer to each vector



Positional encoding

Positional encoding so same words at different locations have different overall representations:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

BERT

BERT (Bidirectional Encoder Representations from **Transformers**): Pre-training of Deep Bidirectional Transformers for Language Understanding

https://arxiv.org/abs/1810.04805

BERT

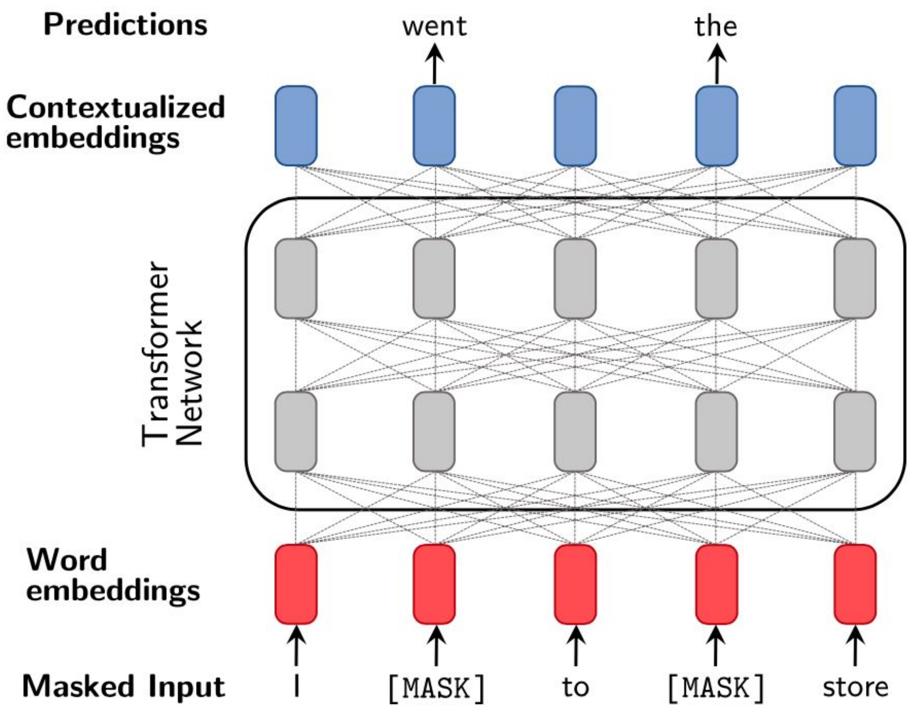
- **Problem**: Language models only use left context or right context, but language understanding is bidirectional.
- Why are LMs unidirectional?
- Reason 1: Directionality is needed to generate a well-formed probability distribution.
  - We don’t care about this.
- Reason 2: Words can “see themselves” in a bidirectional encoder.

Self supervised training

- Self-supervision: supervised task is created out of unlabelled data (language models)
- State-of-the-art NLP systems are trained on a large corpus of text using self-supervision

BERT task 1

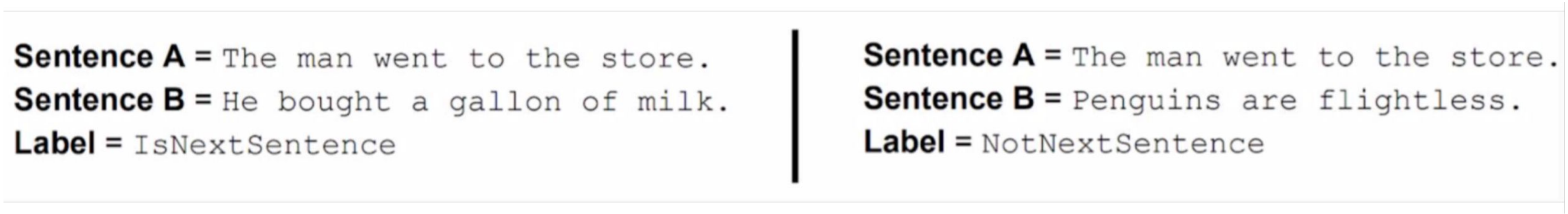
Mask 15% of the input words, and then predict masked words.



https://www.pnas.org/content/117/48/30046/tab-article-info

BERT task 2: next sentence prediction

To learn relationships between sentences, predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence.



Sentence A = `The man went to the store.`
Sentence B = `He bought a gallon of milk.`
Label = `IsNextSentence`

Sentence A = `The man went to the store.`
Sentence B = `Penguins are flightless.`
Label = `NotNextSentence`

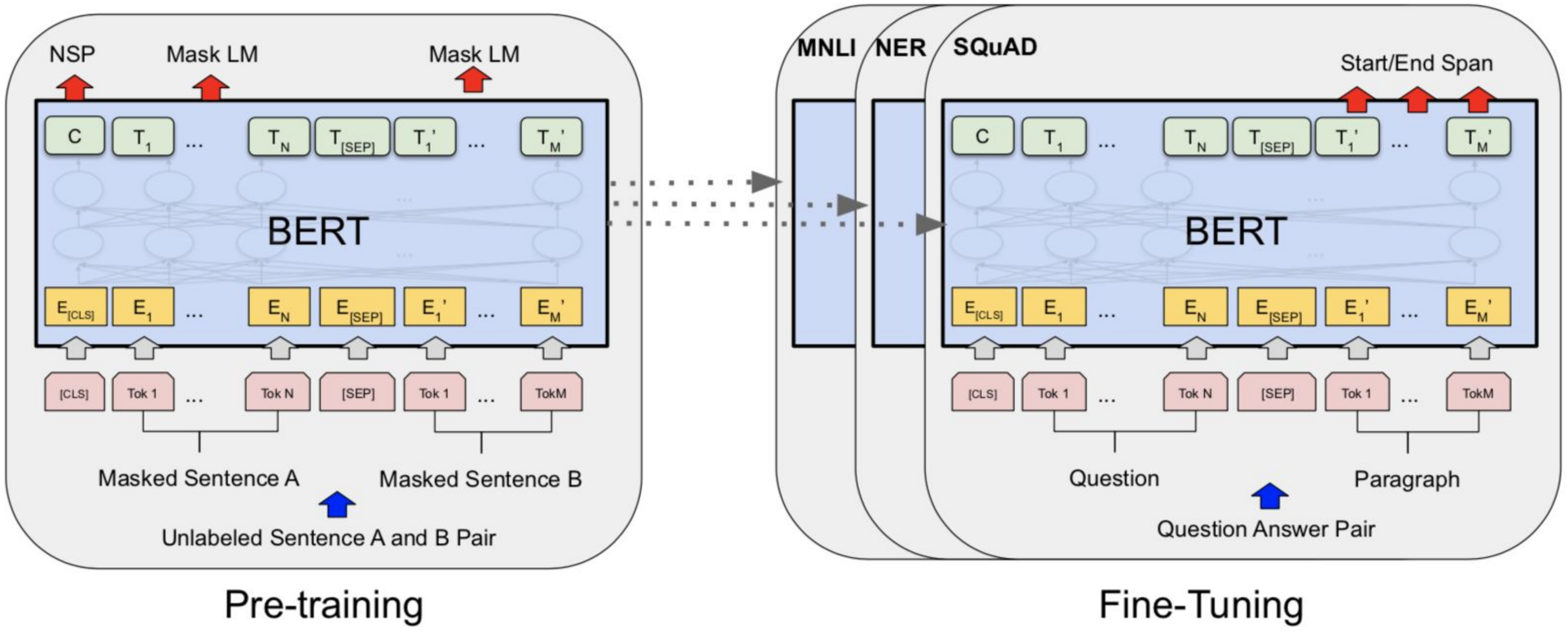
BERT (unsupervised) training

- During pre-training, the model is trained on unlabeled data over different pre-training tasks.
    - Train on Wikipedia + BookCorpus

- Train 2 model sizes:
    - BERT-Base: 12-layer, 768-hidden, 12-head
    - BERT-Large: 24-layer, 1024-hidden, 16-head

- Trained on 4x4 or 8x8 TPU slice for 4 days

Pre-training (unsupervised) then fine-tuning



References

- http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture13-contextual-representations.pdf
- https://arxiv.org/abs/1902.06006
- http://nlp.seas.harvard.edu/2018/04/03/attention.html
- https://cs.stanford.edu/people/karpathy/sfmltalk.pdf
- http://cs231n.stanford.edu/slides/2016/winter1516_lecture13.pdf
- https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture8.pdf
- http://cs231n.stanford.edu/slides/2016/winter1516_lecture10.pdf
- https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture10.pdf