VIS 2022

NLP4Vis: Natural Language Processing for Information Visualization

Half-day Tutorial

Shafiq Joty

Enamul Hoque

https://nlp4vis.github.io/

About me

Home

Papers

Research

Teaching

Students

CV

Shafiq Joty

Research Director
Salesforce AI Research
Office: 181 Lytton Ave, Palo Alto, CA 94301, United States

Email
Phone
Google Scholar

Associate Professor (On leave since Aug'22)
NTU Natural Language Processing Group
Office: Block N4, 02c-79
School of Computer Science and Engineering, NTU

Email
Phone
Github

Short Biography

- Research Director, Salesforce AI Research, [Sep'22 -]
- Associate Professor (tenured, on leave since Aug 2022), Nanyang Technological University (NTU), Singapore [Mar'22 -]
- Senior Research Manager (part time), Salesforce AI Research, Singapore [Feb'19 - Aug'22]
- Assistant Professor, Nanyang Technological University (NTU), Singapore [Jul'17 - Feb'22]
- Research Scientist, Qatar Computing Research Institute (QCRI), Doha [Jan'14 - Jul'17]
- PhD in Computer Science, University of British Columbia (UBC), Vancouver [Sep'08 - Feb'14]

Research Interests

Natural Language Processing

- NLP tools (Syntax, Semantics and Discourse)
- Multi-lingual NLP (Machine Translation, Cross-lingual tasks)
- NLP Applications (QA, Summarization, Dialogue)
- Multi-modal NLP (Image/Video Captioning)
- Robust/adversarial NLP
- NLP for Programming

Machine Learning

- Deep Learning
- Probabilistic Graphical Models
- Reinforcement Learning

Plan

- NLP & why deep learning for NLP [10 min]
- FFNs & word vectors [15 min]
- RNNs & sequence level NLP tasks [20 min]
- Break [30 min]
- Transformers [13 min]
- Self-supervised learning [12 min]

What is NLP

We study formalisms, models and algorithms to allow computers to perform useful tasks involving knowledge about human languages.

Useful Tasks

- **Conversational agents:**
  - AT&T "How may I help you?" technology
  - Apple SIRI, Amazon's alexa, Microsoft's Cortana.
- **Summarization:** "Please summarize my discussion with Sue about NLP" "What people say about the new Nikon 5000?"
- **Machine Translation:** Google translate (100B USD$ industry)
- **Text Generation:** Data2Text, Table2Text, Video/Image2Text
  - ARRIA world leader in NLG- when it floated on London's Alternative Investment Market (AIM) in 2013, it was valued at over £160 million

Useful Tasks

- **Question answering:** "Was 1991 an El Nino year? ....Was it the first one after 1982?" "Why was it so intense?"
  - IBM Watson Jeopardy (now medicine)
- **Document Classification:** spam detection, news filtering
- **Information Extraction:** Knowledge graphs (in Search)
- **Speech:**
  - speech recognition, text to speech synthesis.
- **Multimodal:** Image2text, text2image (captioning, VQA)

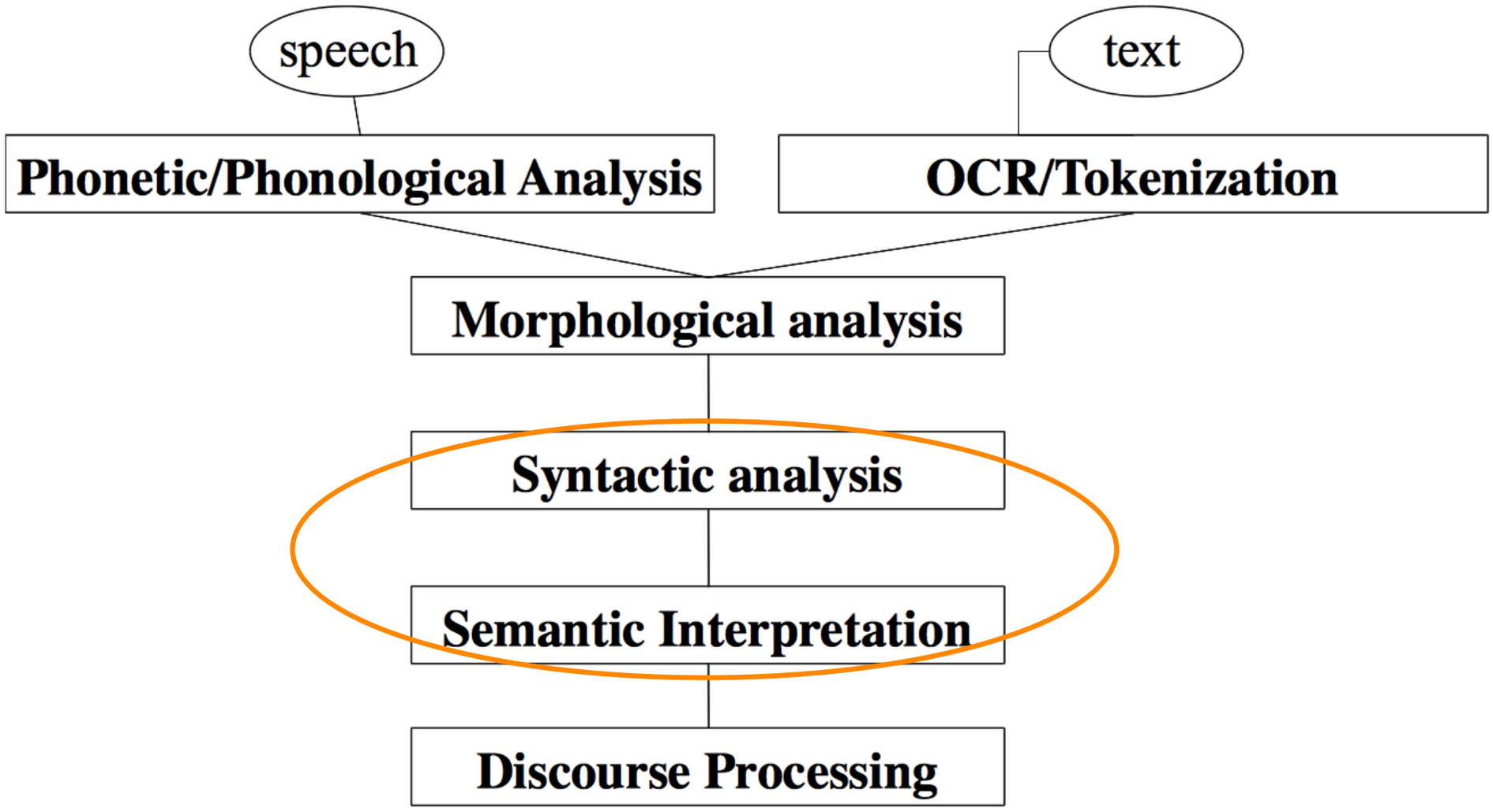
What is NLP

We study formalisms, models and algorithms to allow computers to perform useful tasks involving knowledge about human languages.

Knowledge about Language

- Phonetics and Phonology (sounds)
- Morphology (structure of words)
- Syntax (structure of sentences)
- Semantics (meaning)
- Pragmatics (language use)
- Discourse and Dialogue (units larger than single utterance)

Knowledge about Language

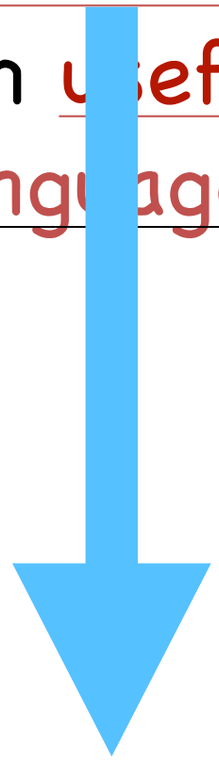


Natural Language as Signal

- Language is mostly a **discrete/symbolic/categorical** signalling system
  - Dog, Cat, Pen
- The categorical symbols of a language can be encoded as a signal for communication in several ways:
  - Sound, Writing, Gesture
- However, human brain's encoding appears to be a **continuous pattern of activation**, and the symbols are transmitted via **continuous signals** of sound/vision

What is NLP

We study formalisms, models and algorithms to allow computers to perform useful tasks involving knowledge about human languages.



Different Neural Models (This tutorial)

Traditionally:

Finite State Automata, Markov Models, Graphical Models

What is Deep Learning?

- **Deep learning** is a subfield of **machine learning.**
- Most machine learning methods work well because of **human-designed representations/input features.**
- Machine learning becomes just optimizing feature weights to make a good prediction.

Feature	NER
Current Word	✓
Previous Word	✓
Next Word	✓
Current Word Character n-gram	all
Current POS Tag	✓
Surrounding POS Tag Sequence	✓
Current Word Shape	✓
Surrounding Word Shape Sequence	✓
Presence of Word in Left Window	size 4
Presence of Word in Right Window	size 4

Features for NER (Finkel et al., 2010)

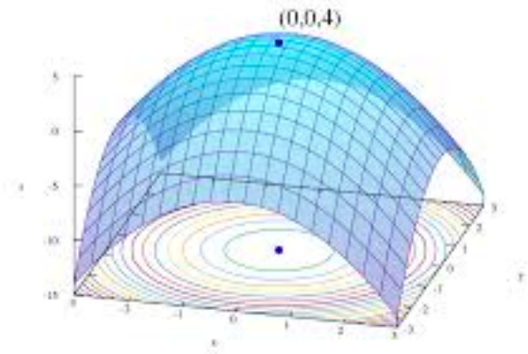
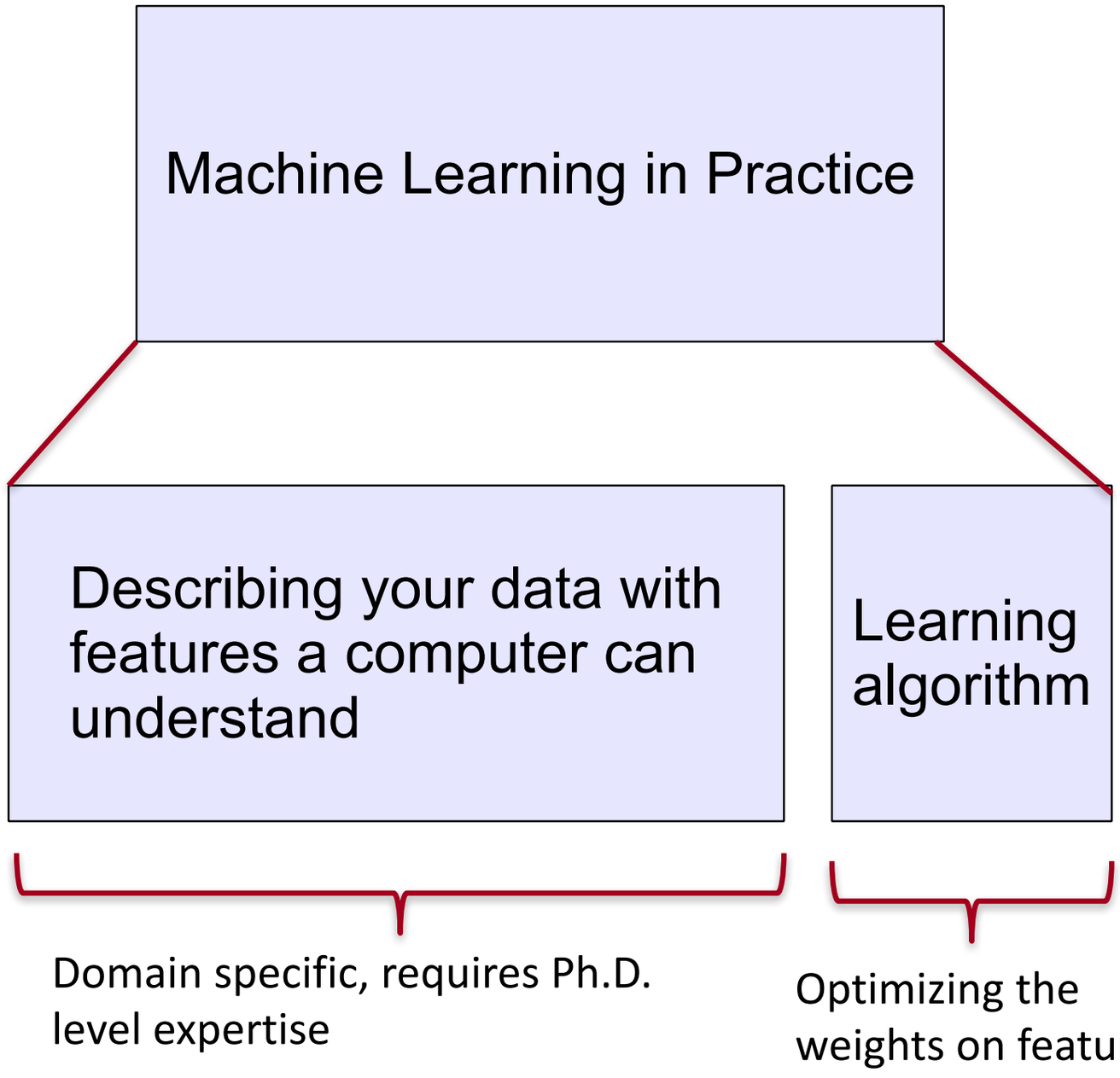


Figure 1: An example of NER application on an example text

Traditional Machine Learning



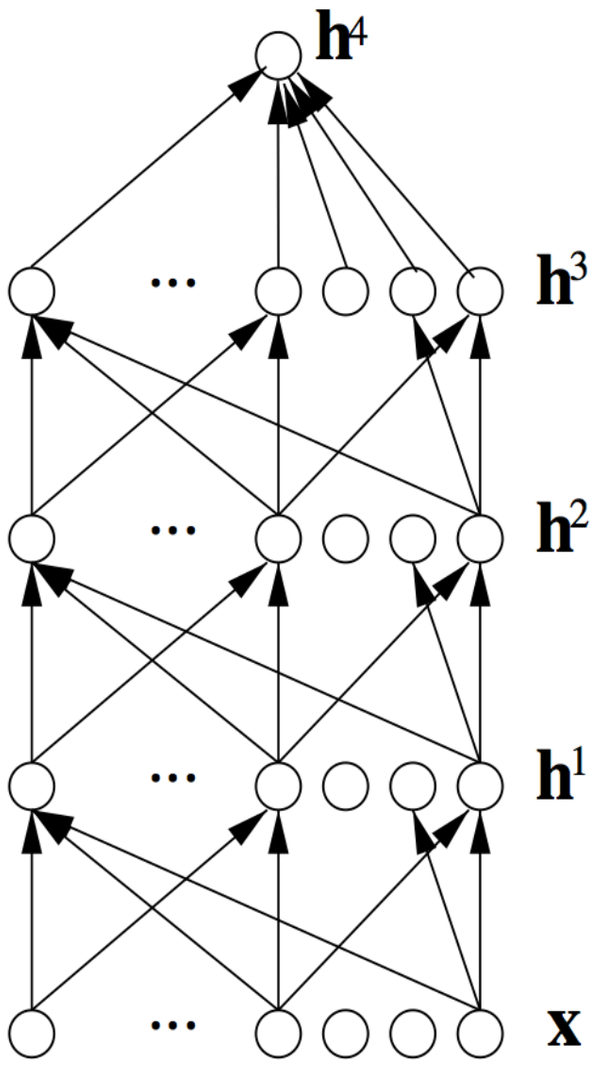
Source: Stanford 224n

Deep Learning

- Representation learning attempts to automatically learn good features or representations

- Deep learning algorithms attempt to learn (multiple levels of) representations (here: h^1,h^2,h^3) and an output (h^4)

- From “raw” inputs **x**
(e.g. sound, pixels, characters, or words)



Source: Stanford 224n

Why Deep Learning?

#1. Flexible R&R System

- Manually designed features are often over-specified, incomplete and take a long time to design and validate
- This has to be done again for each task/domain/…
- **Learned Features** are easy to adapt, fast to learn
- Deep learning provides a very flexible, learnable framework for **representing the** world (e.g., visual and linguistic information), and perform **reasoning** with that representation.
- Deep learning can learn in **unsupervised** (from raw text) and **supervised** (with specific labels like positive/negative) settings

Source: Socher & Manning

Why Deep Learning?

#2. Need for distributed representations

- Symbolic NLP systems are incredibly fragile because of their atomic symbol representation.

- Deep learning uses **distributed representations** of words, phrases, and concepts that generalise better

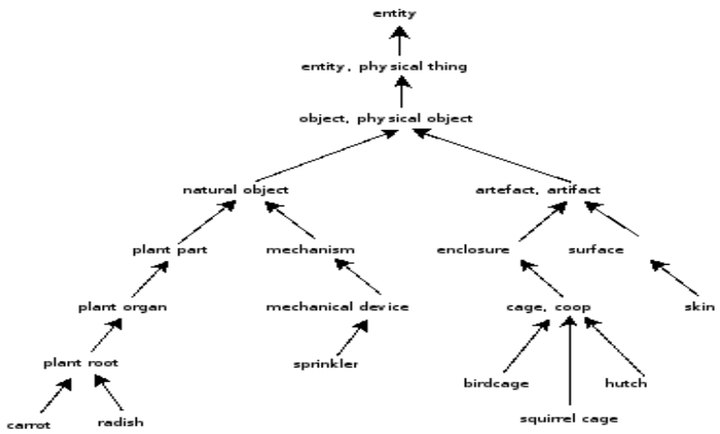
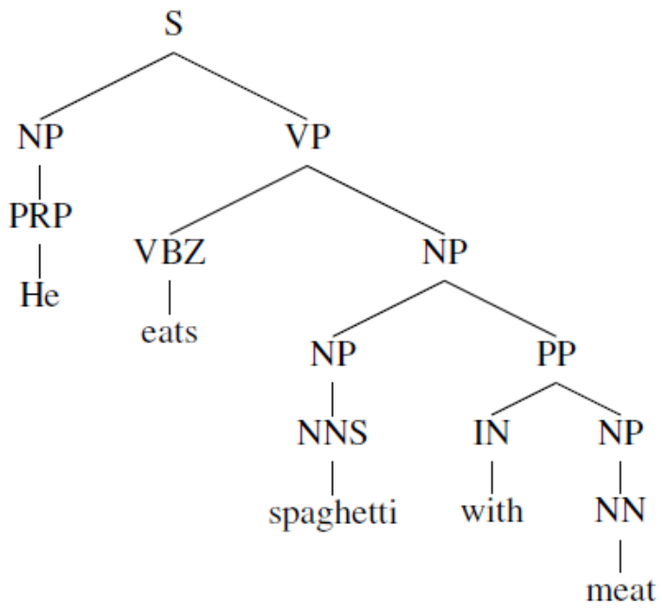
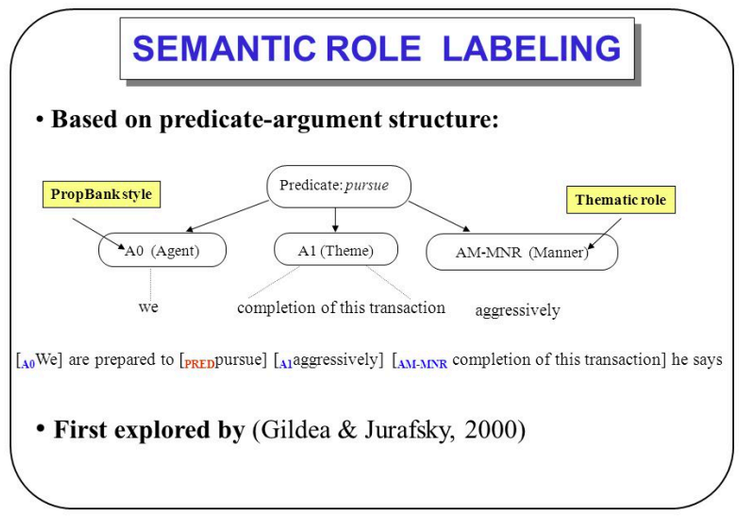


Figure 1. "is-a" relation example

WordNet





Why Deep Learning?

#3. Need for Unsupervised (or self-supervised) learning

- Most practical, good traditional ML methods require labeled training data (i.e., supervised learning).
- However, most information must be acquired **unsupervised**

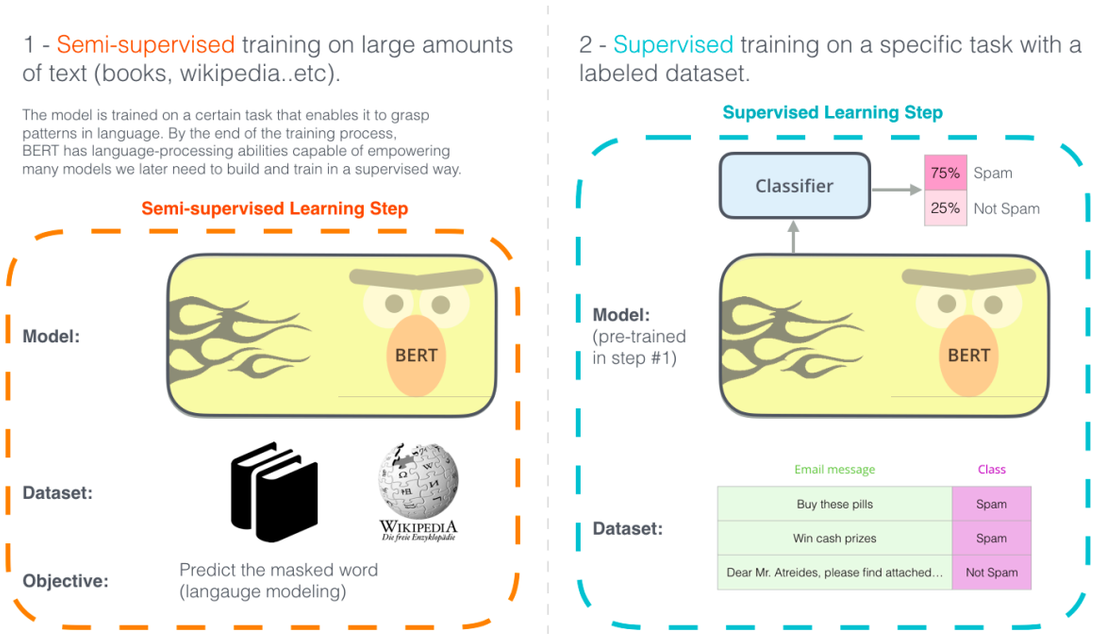
> When we're learning to see, nobody's telling us what the right answers are — we just look. Every so often, your mother says "that's a dog", but that's very little information. You'd be lucky if you got a few bits of information — even one bit per second — that way. The brain's visual system has 10^{14} neural connections. And you only live for 10^9 seconds. So it's no use learning one bit per second. You need more like 10^5 bits per second. And there's only one place you can get that much information: from the input itself. — Geoffrey Hinton, 1996 (quoted in (Gorder 2006)).

Source: Socher & Manning

Why Deep Learning?

#3. Need for Unsupervised (or self-supervised) learning

- Models like BERT, ROBERTA, T5, VL-BERT, DALL-E are changing the entire landscape of NLP (and AI in general)



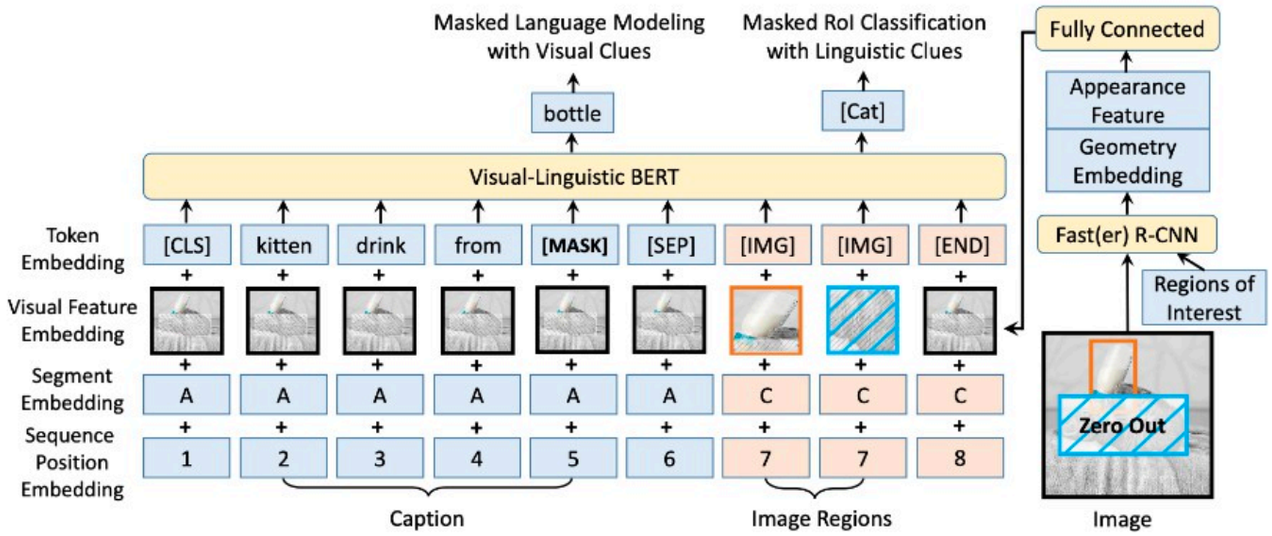
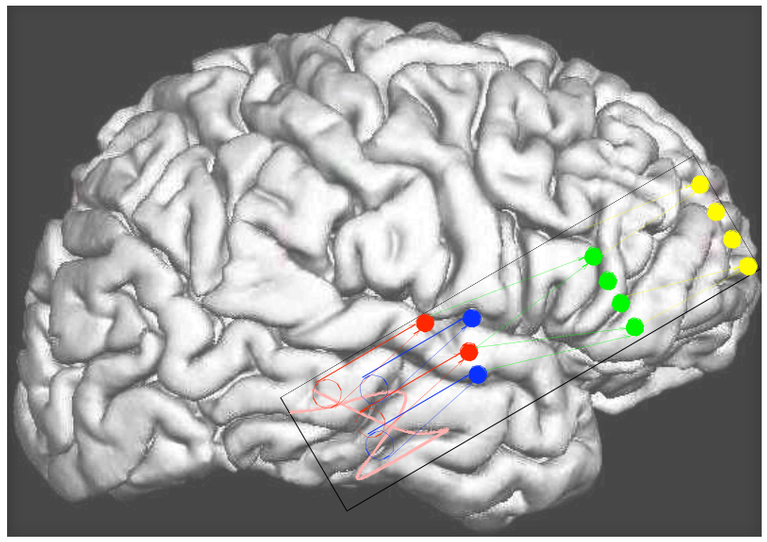


Figure 1. Architecture for Pre-training VL-BERT

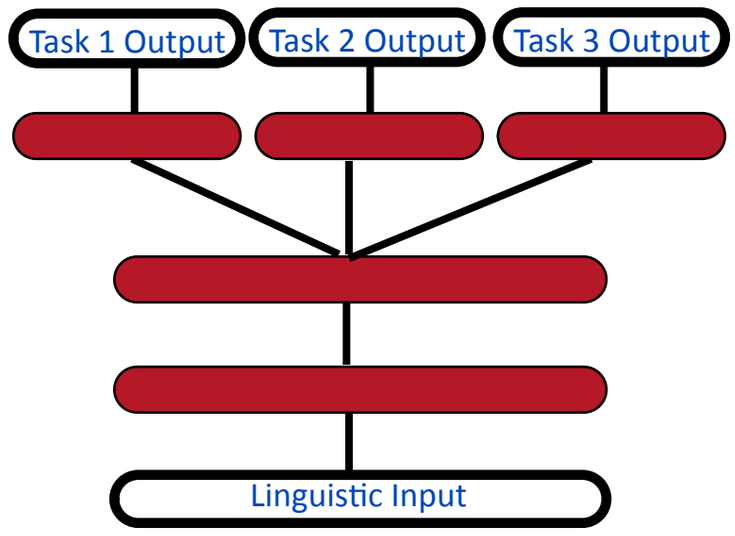
Why Deep Learning?

#4. Learning multiple levels of representation

- The cortex seems to have a generic learning algorithm
- The brain has a deep architecture



- We need good intermediate representations that can be shared across tasks
- Multiple levels of latent variables allow combinatorial sharing of statistical strength
- Insufficient model depth can be exponentially inefficient

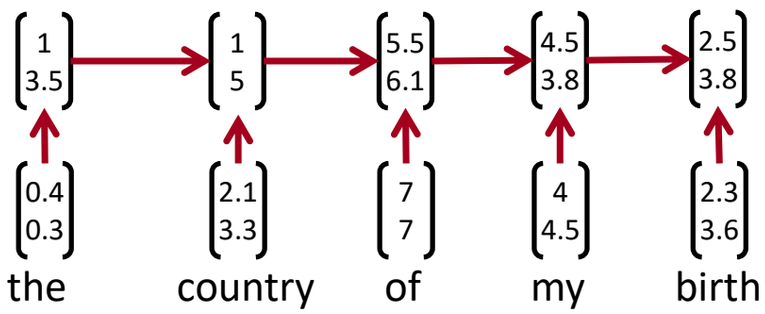


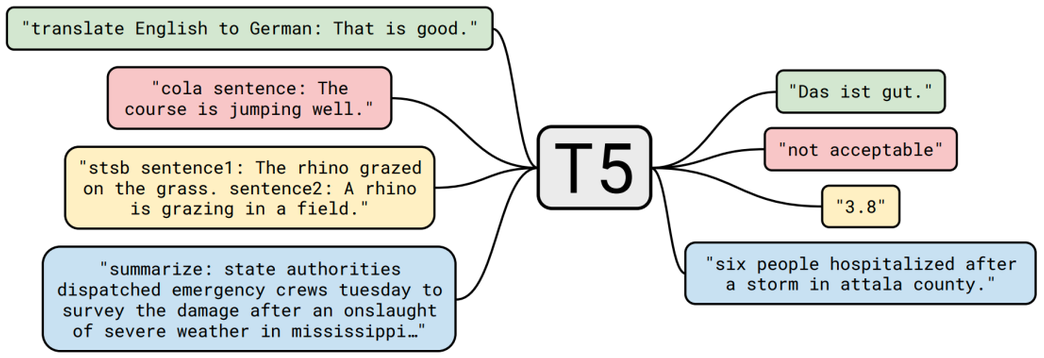
Source: Socher & Manning

Why Deep Learning?

#4. Learning multiple levels of representation

- We need good intermediate representations that can be shared across tasks
- We need compositionality in our ML models





Why Deep Learning?

#5. Why now?

- In ~2010 **deep** learning techniques started outperforming other machine learning techniques. Why this decade?
  - Large amounts of training data favor deep learning
  - Faster machines and multicore CPU/GPU/TPUs favor deep learning
  - New models, algorithms, ideas
    - Better, more flexible learning of intermediate representations
    - Effective end-to-end joint system learning
    - Effective learning methods for using contexts and transferring between tasks
    - Better regularization and optimization methods

- **Improved performance** (first in speech and vision, then NLP)

Source: Stanford 224n

Recent Breakthrough

BLOG POST
RESEARCH

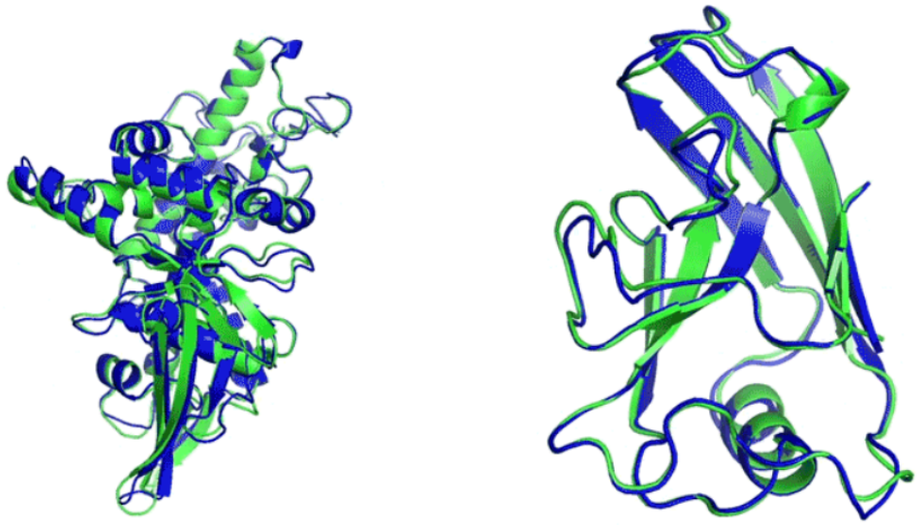
30 NOV 2020

AlphaFold: a solution to a 50-year-old grand challenge in biology

We have been stuck on this one problem – how do proteins fold up – for nearly 50 years. To see DeepMind produce a solution for this, having worked personally on this problem for so long and after so many stops and starts, wondering if we'd ever get there, is a very special moment.

PROFESSOR JOHN MOULT
CO-FOUNDER AND CHAIR OF CASP, UNIVERSITY OF MARYLAND

3D structure



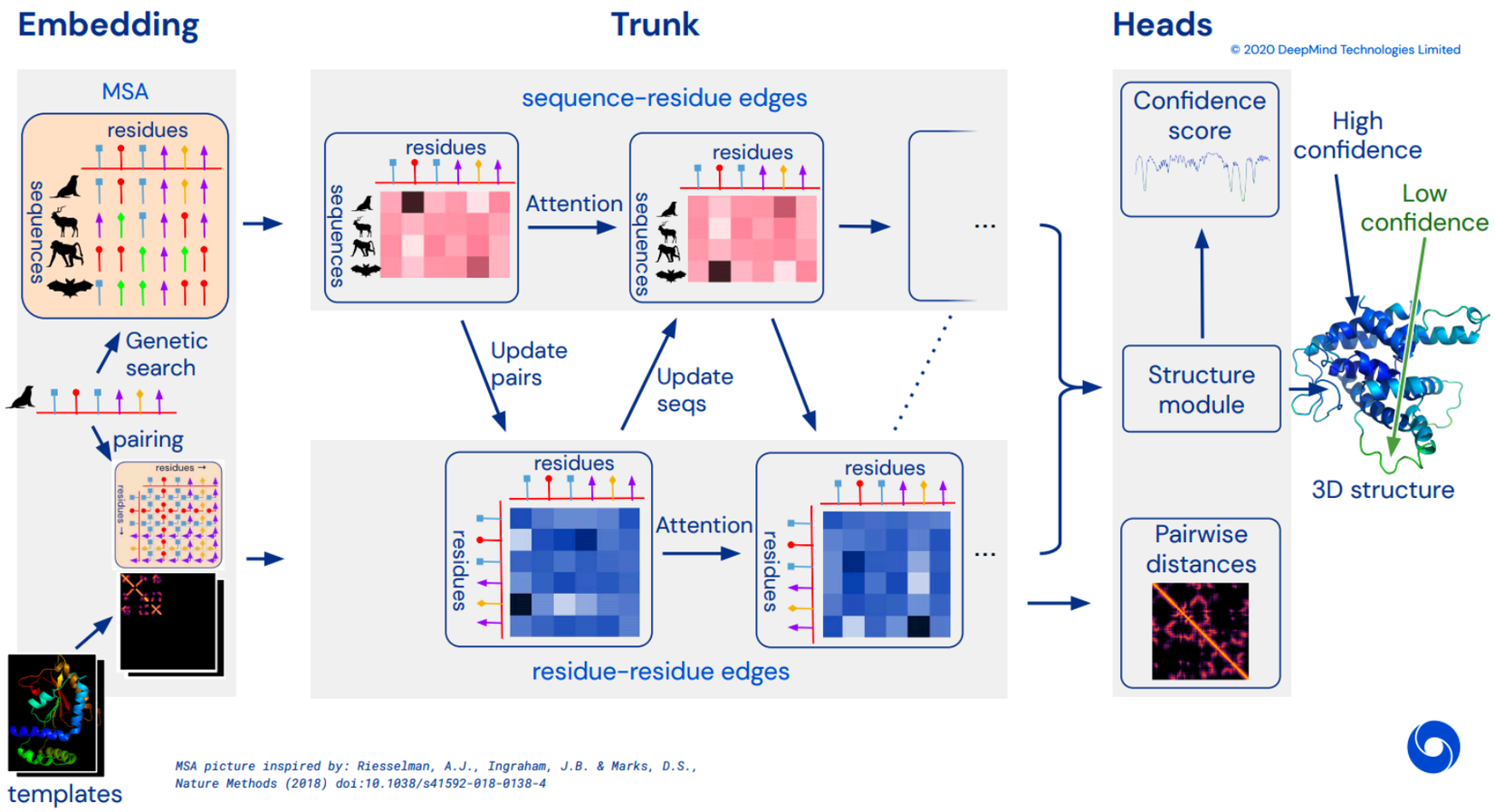
T1037 / 6vr4
90.7 GDT
(RNA polymerase domain)

T1049 / 6y4f
93.3 GDT
(adhesin tip)

Experimental result
Computational prediction

https://deepmind.com/blog/article/alphafold-a-solution-to-a-50-year-old-grand-challenge-in-biology

Recent Breakthrough



*"Improvement over the first version of AlphaFold is mostly usage of **transformer/attention mechanisms** applied to residue space and combining it with the working ideas from the first version"*

Transformer was proposed first for NMT!

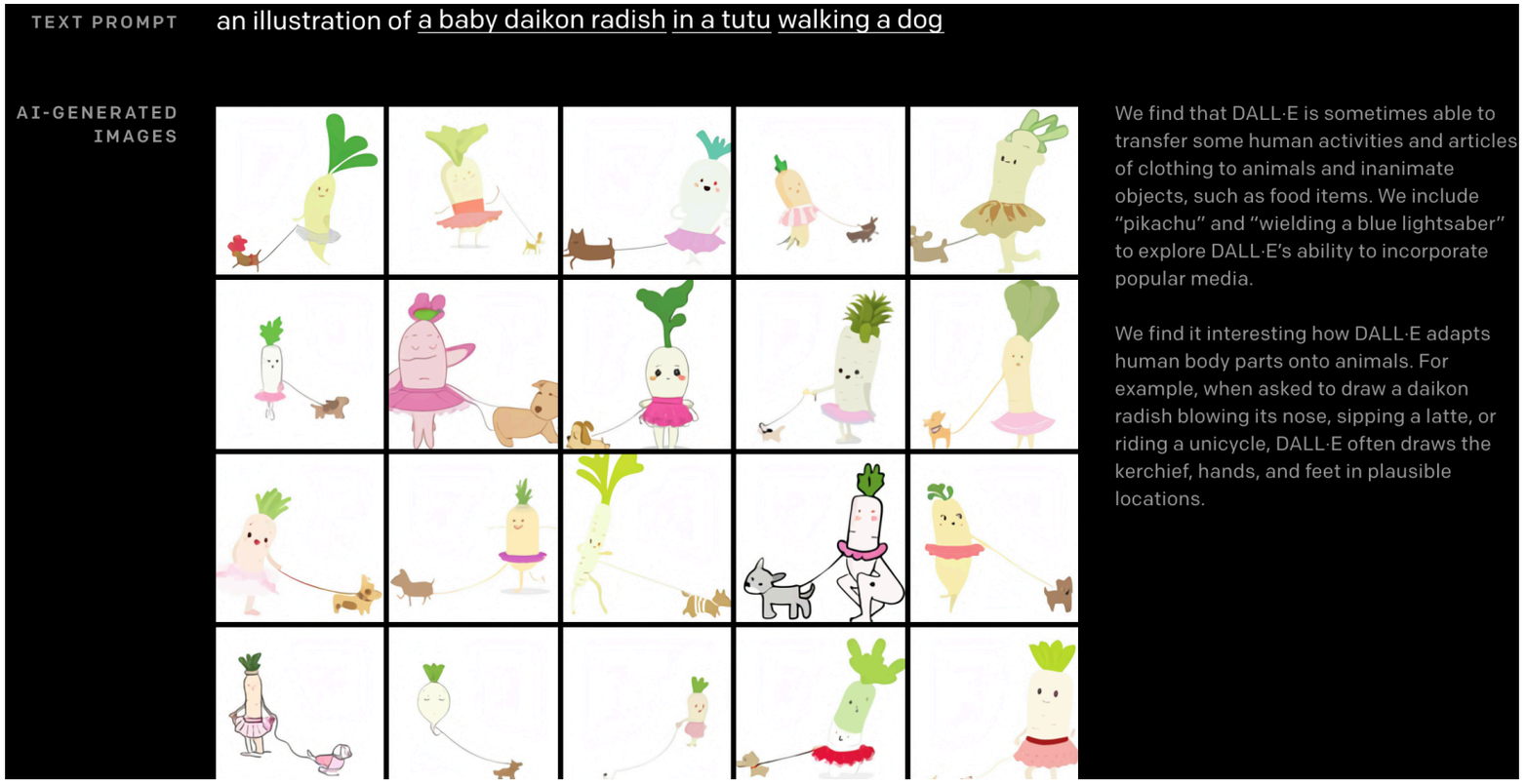
https://deepmind.com/blog/article/alphafold-a-solution-to-a-50-year-old-grand-challenge-in-biology

Recent Breakthrough

DALL·E: Creating Images from Text

We've trained a neural network called DALL·E that creates images from text captions for a wide range of concepts expressible in natural language.

DALL·E[1] is a 12-billion parameter version of GPT-3 trained to generate images from text descriptions, using a dataset of text–image pairs. We've found that it has a diverse set of capabilities, including creating anthropomorphized versions of animals and objects, combining unrelated concepts in plausible ways, rendering text, and applying transformations to existing images.



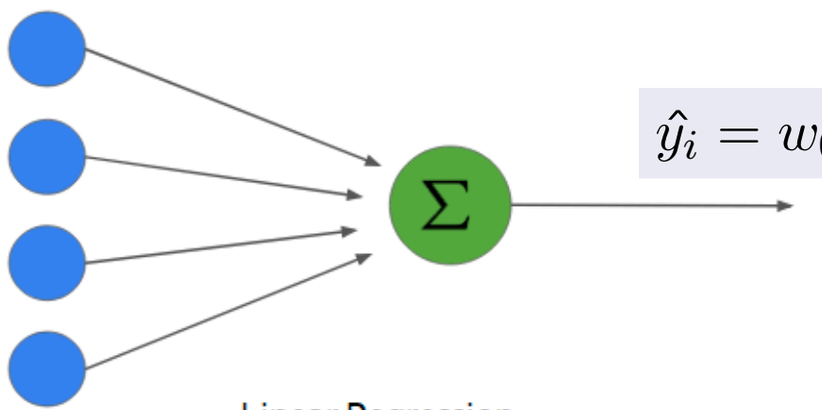
https://openai.com/blog/dall-e/

Plan

- NLP & why deep learning for NLP [10 min]
- FFNs & word vectors [15 min]
- RNNs & sequence level NLP tasks [20 min]
- Break [30 min]
- Transformers [13 min]
- Self-supervised learning [12 min]

Linear Models (Quick Overview)

Linear Regression:



Linear Regression

$$\hat{y}_i = w_0 + w_1 x_{i1} + \ldots + w_d x_{id}$$

MSE loss

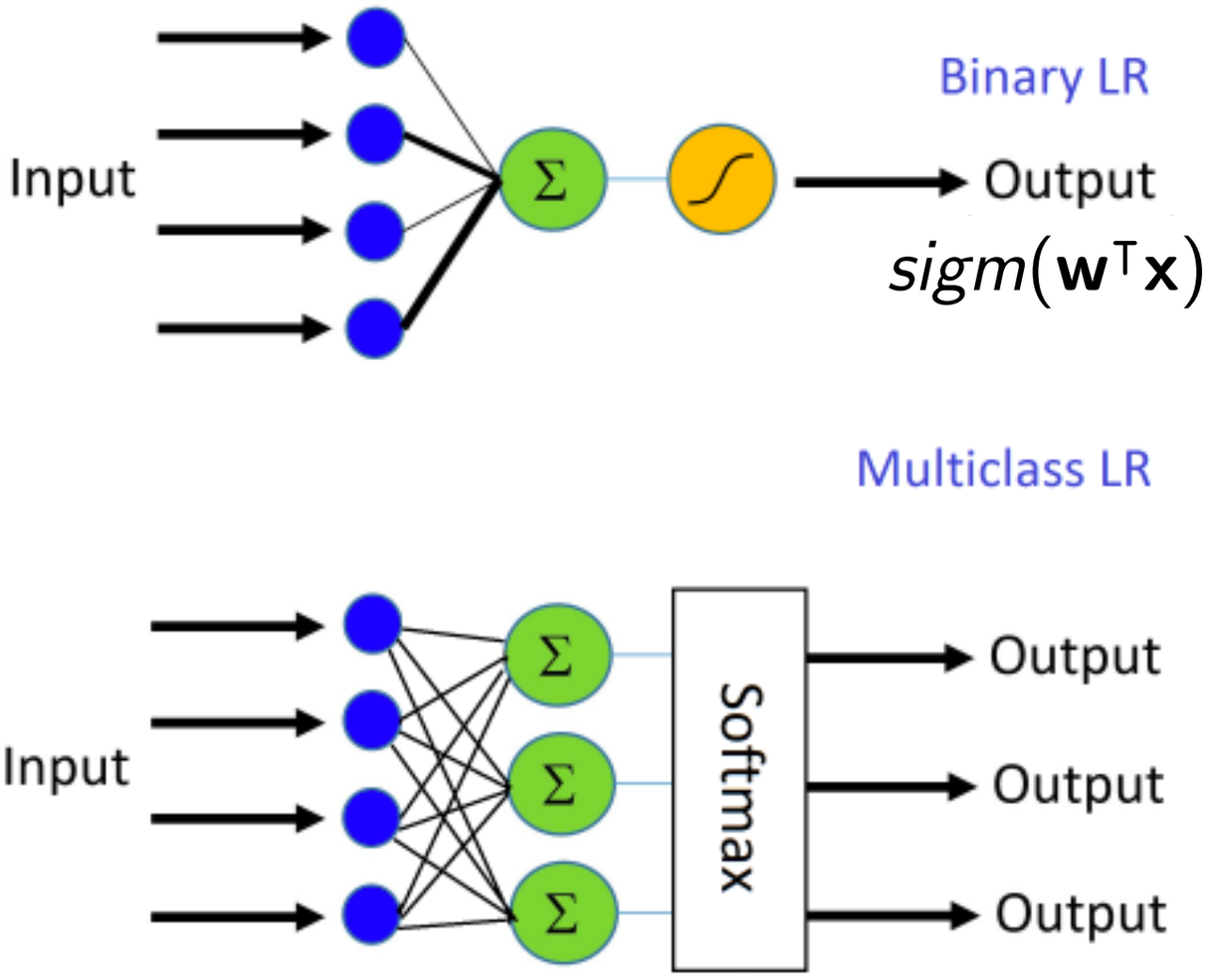
$$RSS(\mathbf{w}) \triangleq \sum_{i=1}^{N} (y_i - \mathbf{w}^\intercal \mathbf{x}_i)^2$$

Closed form solution

$$\mathbf{w} = (\mathbf{X}^\intercal \mathbf{X})^{-1} \mathbf{X}^\intercal y$$

Linear Models (Quick Overview)

Logistic Regression & MaxEnts



Softmax $$\frac{\exp(\mathbf{w}_c^\intercal \mathbf{x})}{\sum_{c'=1}^{C} \exp(\mathbf{w}_{c'}^\intercal \mathbf{x})}$$

SGD

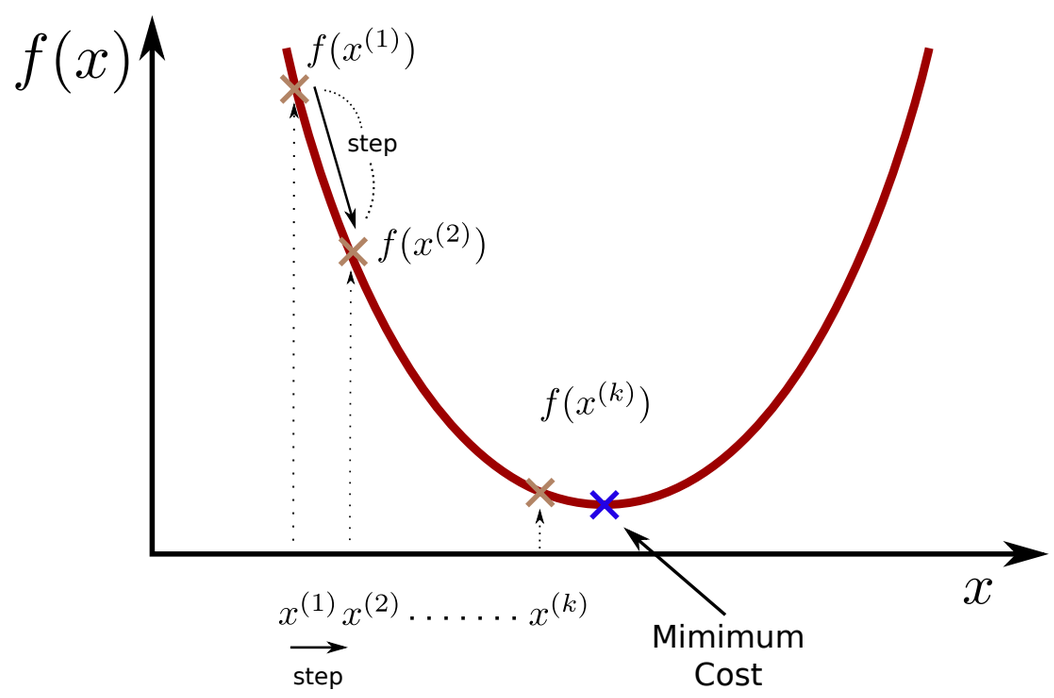


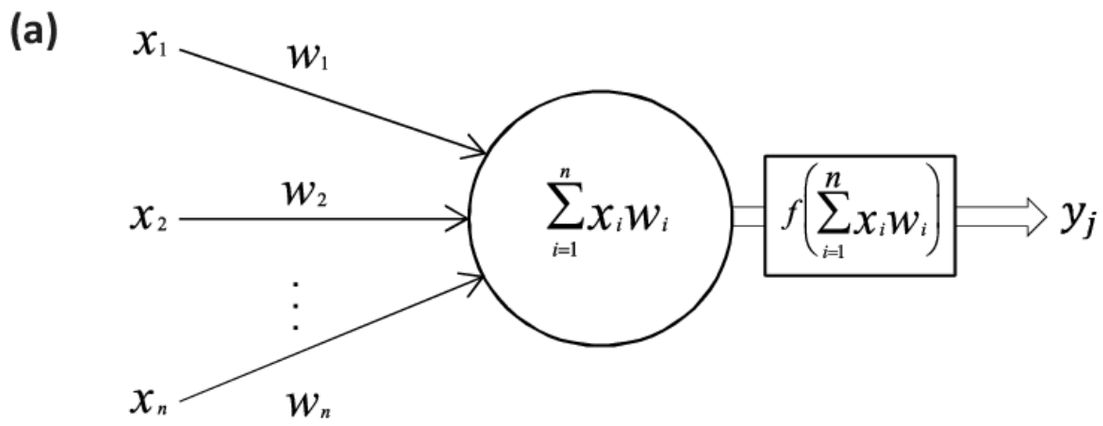
Figure: Illustration of gradient descent

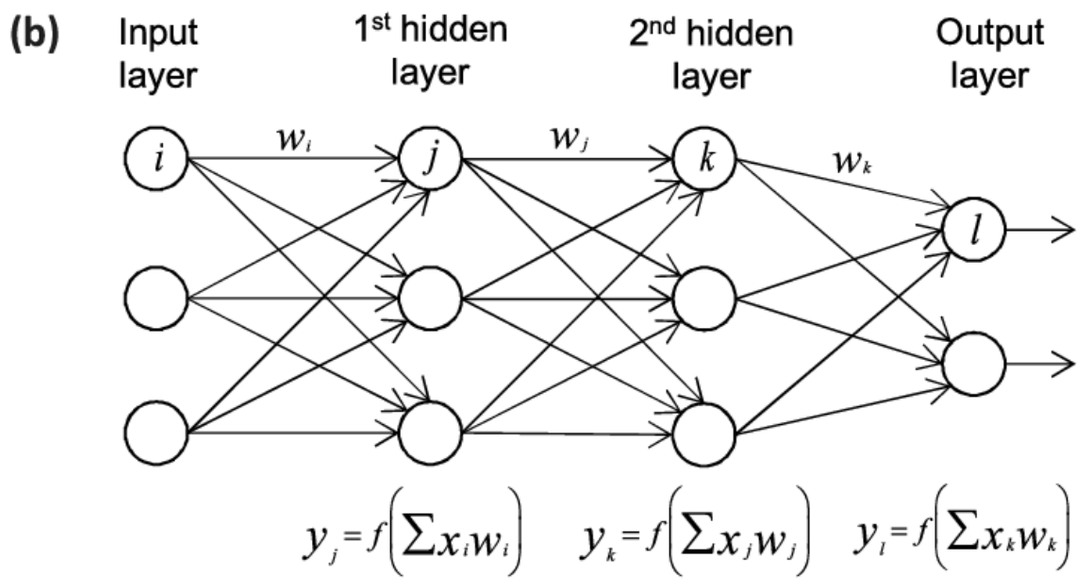
$$\theta_{k+1} = \theta_k - \eta_k \mathbf{g}_k$$

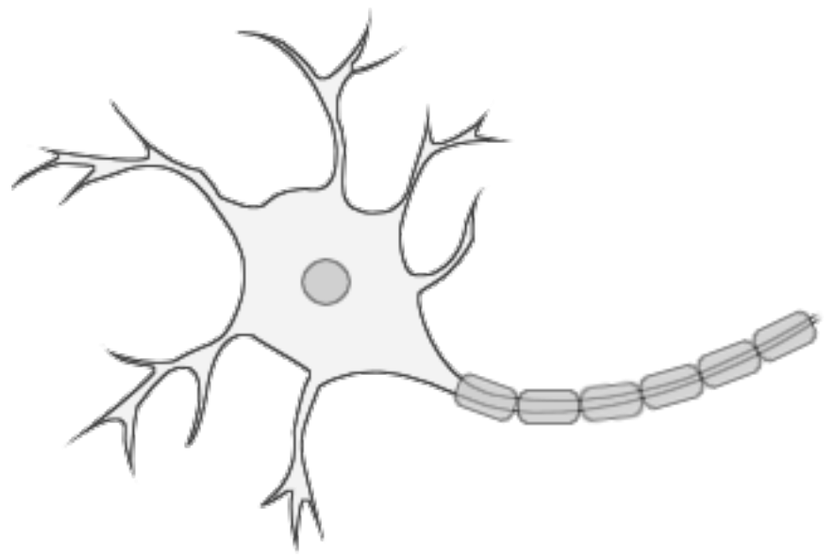
Neurons

if you understand how logistic regression works

Then **you already understand** the operation of a basic neural network neuron!



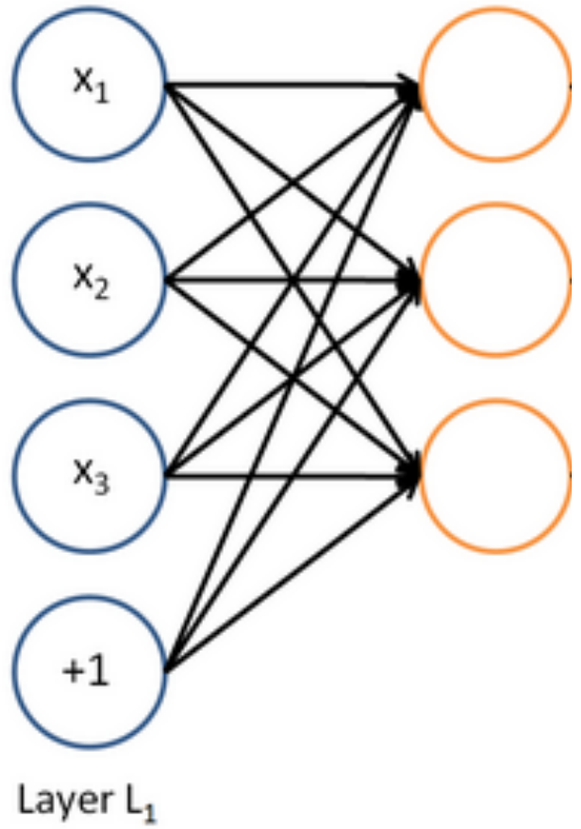




From LR to NN

A neural network = running several logistic regressions at the same time

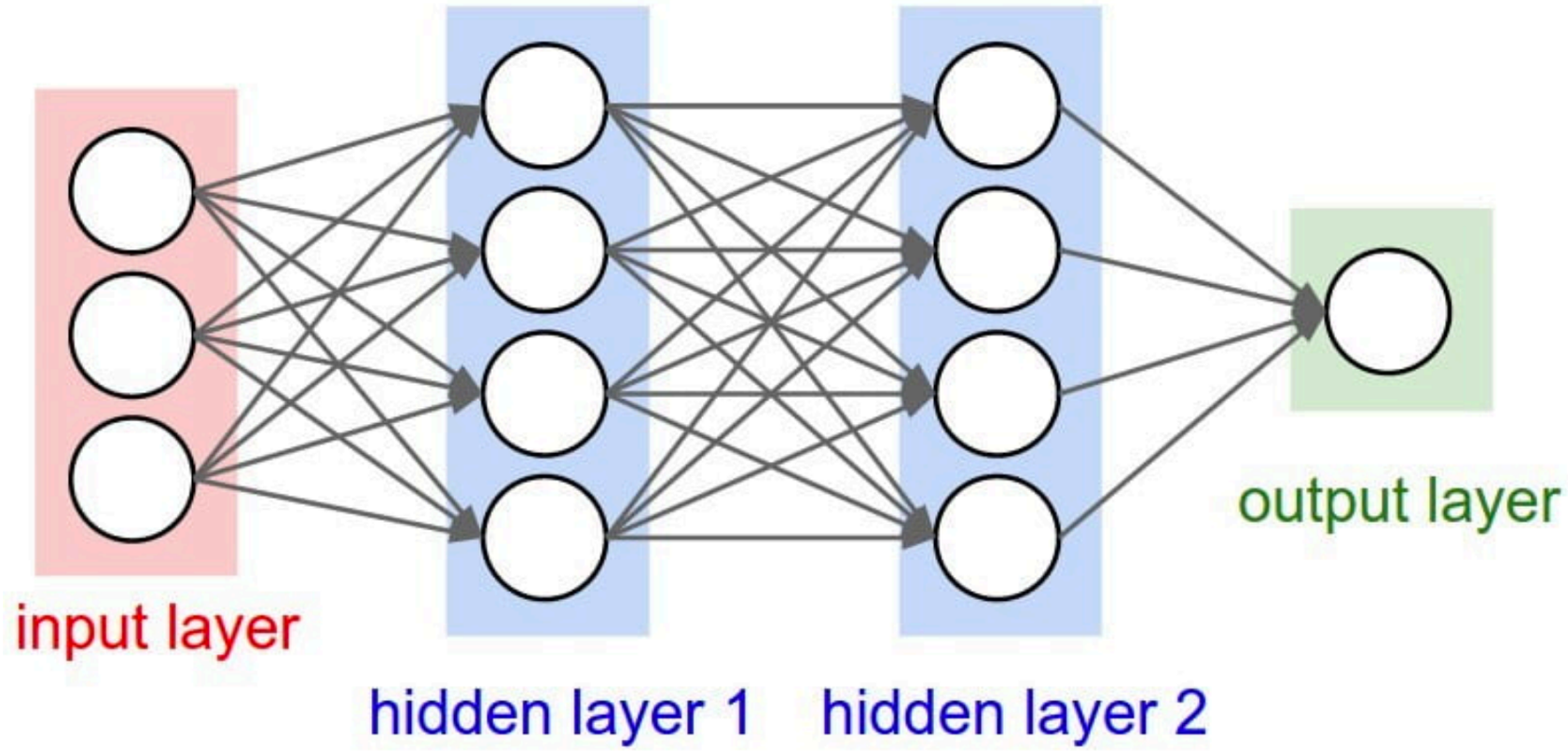
If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs …



But these are (typically) not random variables

From LR to NN

... which we can feed into another logistic regression function

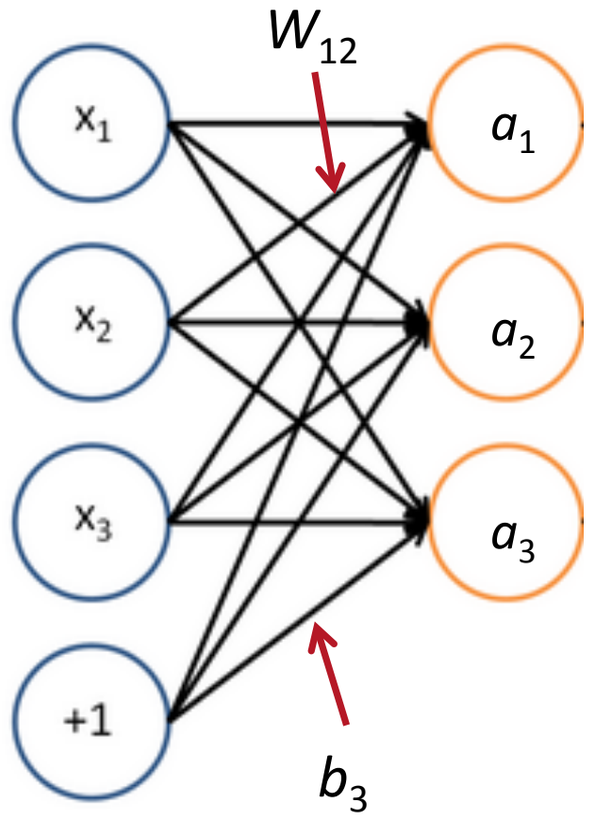


Output layer can be a classification layer or a regression layer

From LR to NN

$$a_1 = f(W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + b_1)$$
$$a_2 = f(W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + b_2)$$

.....



In Matrix Notation

$$z = Wx + b$$
$$a = f(z)$$

Where f() is applied element-wise

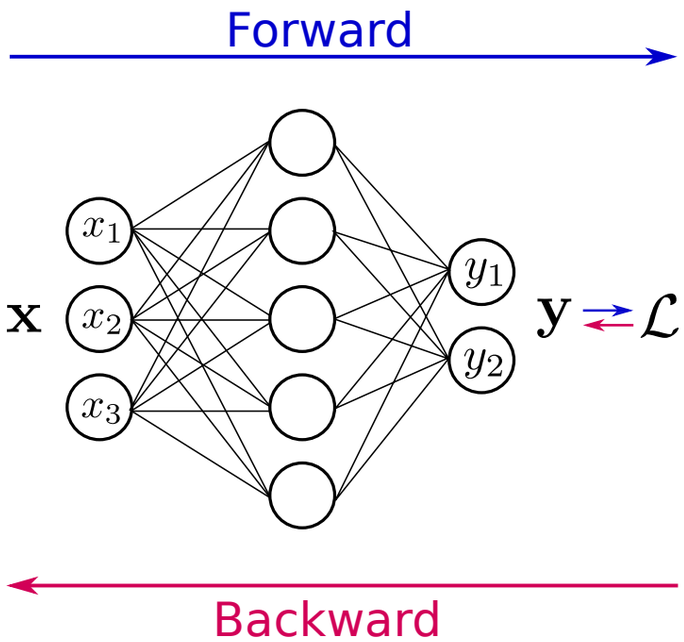
$$f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$$

Forward- & Back-propagation

Two information flow directions:

Forward propagation

- NN accepts an input **x** and produces an output **y**
- During training, it continues onward until it produces a scalar cost **L**



Back-propagation

- Information (**gradients** with respect to the parameters) from the cost flows backward through the network

Word Meaning

- The idea that is represented by a word, phrase, etc.
- How do we represent it in a computer?

Denotational Semantics

Common solution: Use e.g. WordNet, a thesaurus containing lists of **synonym sets** and **hypernyms** ("is a" relationships).

e.g. synonym sets containing "good":

```
from nltk.corpus import wordnet as wn
poses = { 'n':'noun', 'v':'verb', 's':'adj (s)', 'a':'adj', 'r':'adv'}
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[synset.pos()],
            ", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
…
adverb: well, good
adverb: thoroughly, soundly, good
```

e.g. hypernyms of "panda":

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```


Source: Stanford 224N

Problems with WordNet

- Great as a lexical resource but missing nuance
  - e.g. "proficient" is listed as a synonym for "good". This is only correct in some contexts.
- Missing new meanings of words, e.g., wicked, badass, nifty, wizard, genius, ninja, bombest
- Impossible to keep up-to-date!
- Subjective
- Requires human labor to create and adapt

Source: Stanford 224N

Problems with Discrete Representation

- Hard to compute accurate word similarity
- In traditional NLP, we regard words as discrete symbols: hotel, conference, motel

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

These two vectors are orthogonal.

- Could try to rely on WordNet's list of synonyms to get similarity?
  - But it is well-known to fail badly: incompleteness, etc.
  - How to represent polysemous words?

Source: Stanford 224N

Representing words by their context

- Instead: learn to encode similarity in the vectors themselves

- Distributional semantics: **A word's meaning is given by the words that frequently appear close-by**

    - "You shall know a word by the company it keeps"

(J. R. Firth 1957: 11)

- When a word w appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).

- Use the many contexts of w to build up a representation of w

Source: Stanford 224N

Representing words by their context

- When a word w appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).

- Use the many contexts of w to build up a representation of w

…government debt problems turning into	***banking***	*crises as happened in 2009…*
…saying that Europe needs unified	***banking***	*regulation to replace the hodgepodge…*
…India has just given its	***banking***	*system a shot in the arm…*

These context words will represent ***banking***

Source: Stanford 224N

Word Vectors

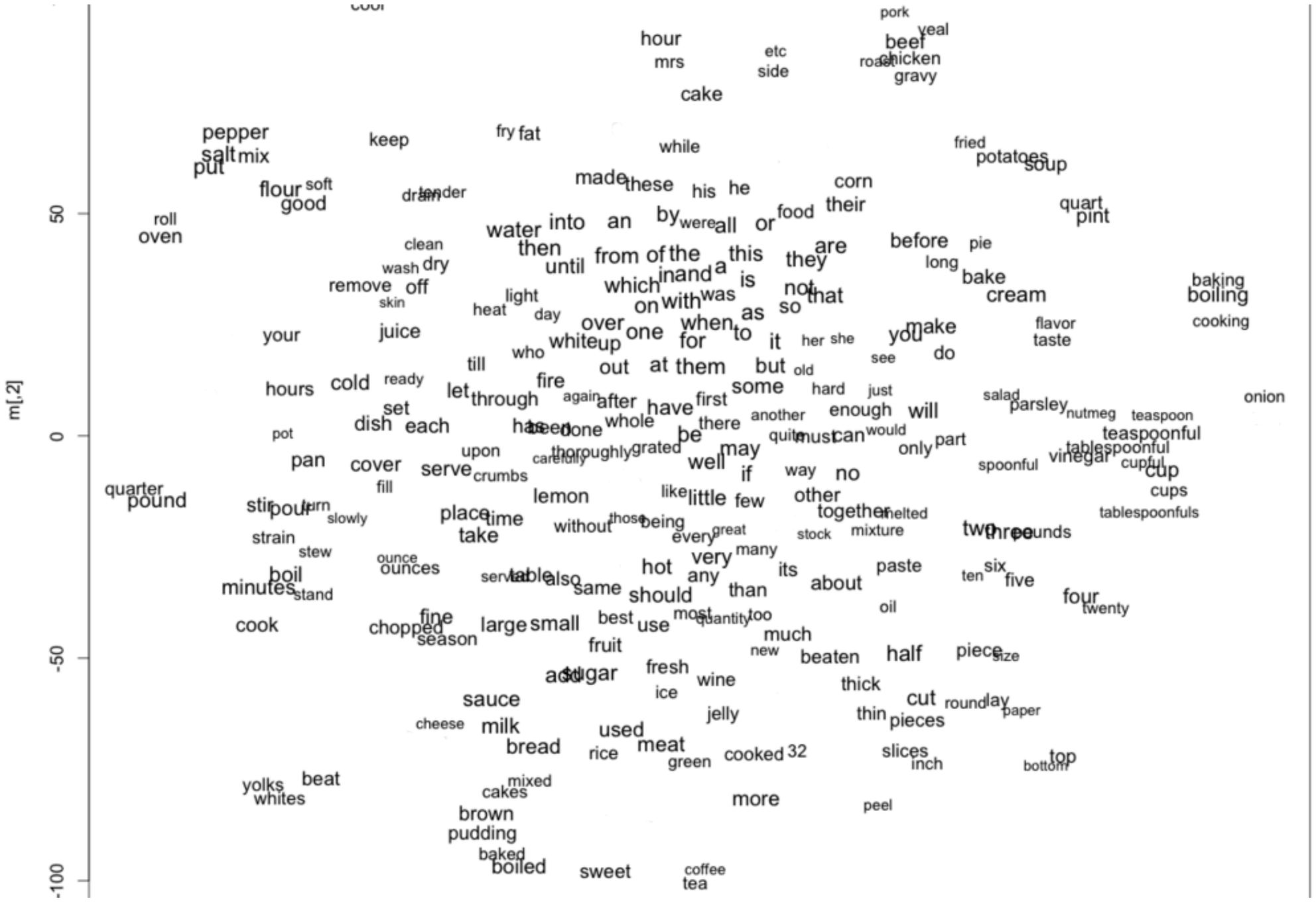
We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts

$$banking = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

word vectors are sometimes called word embeddings or word representations. They are a distributed representation.

Source: Stanford 224N

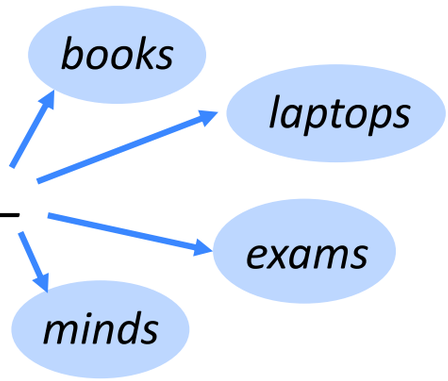
Word Vectors



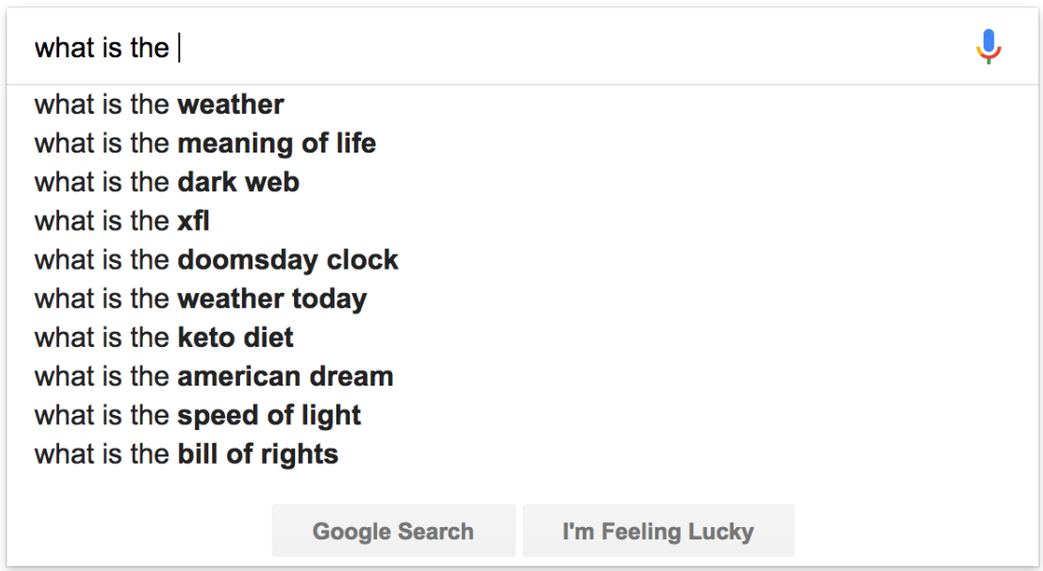
The Earliest Neural Model to Learn Word Embeddings

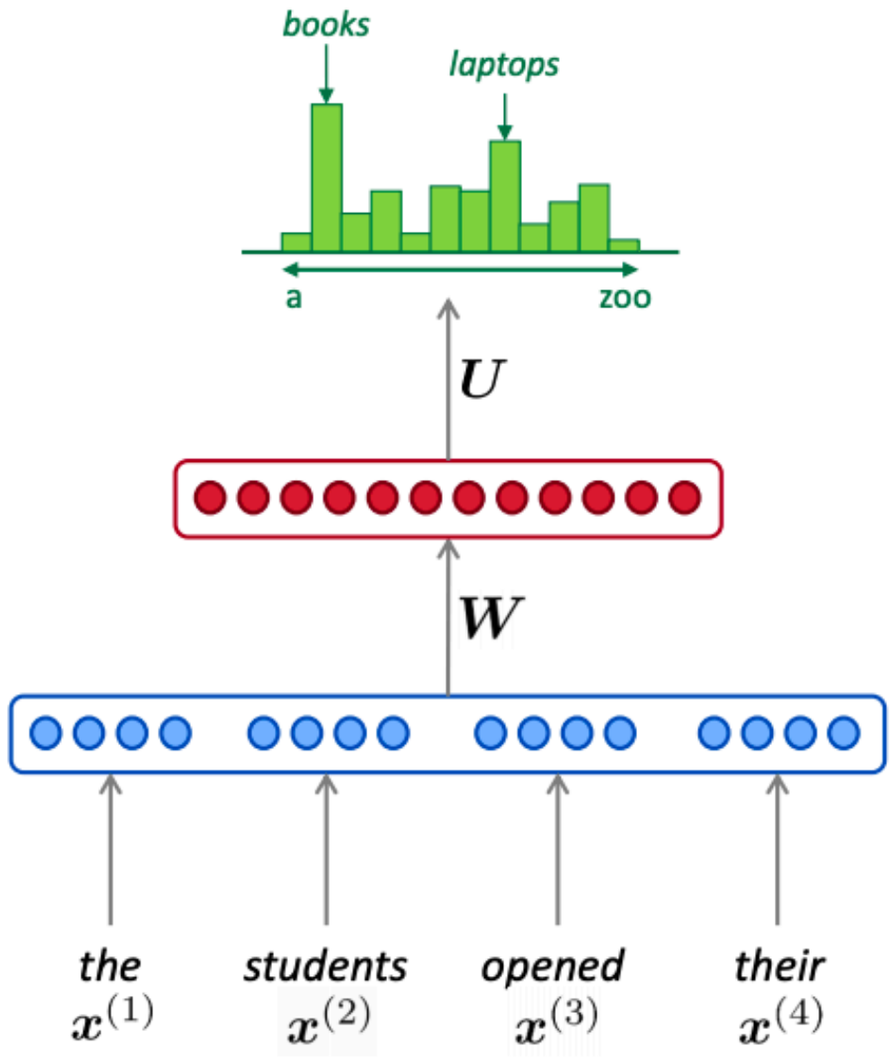
A **language model** takes a list of words (history), and attempts to predict the word that follows them

the students opened their _____



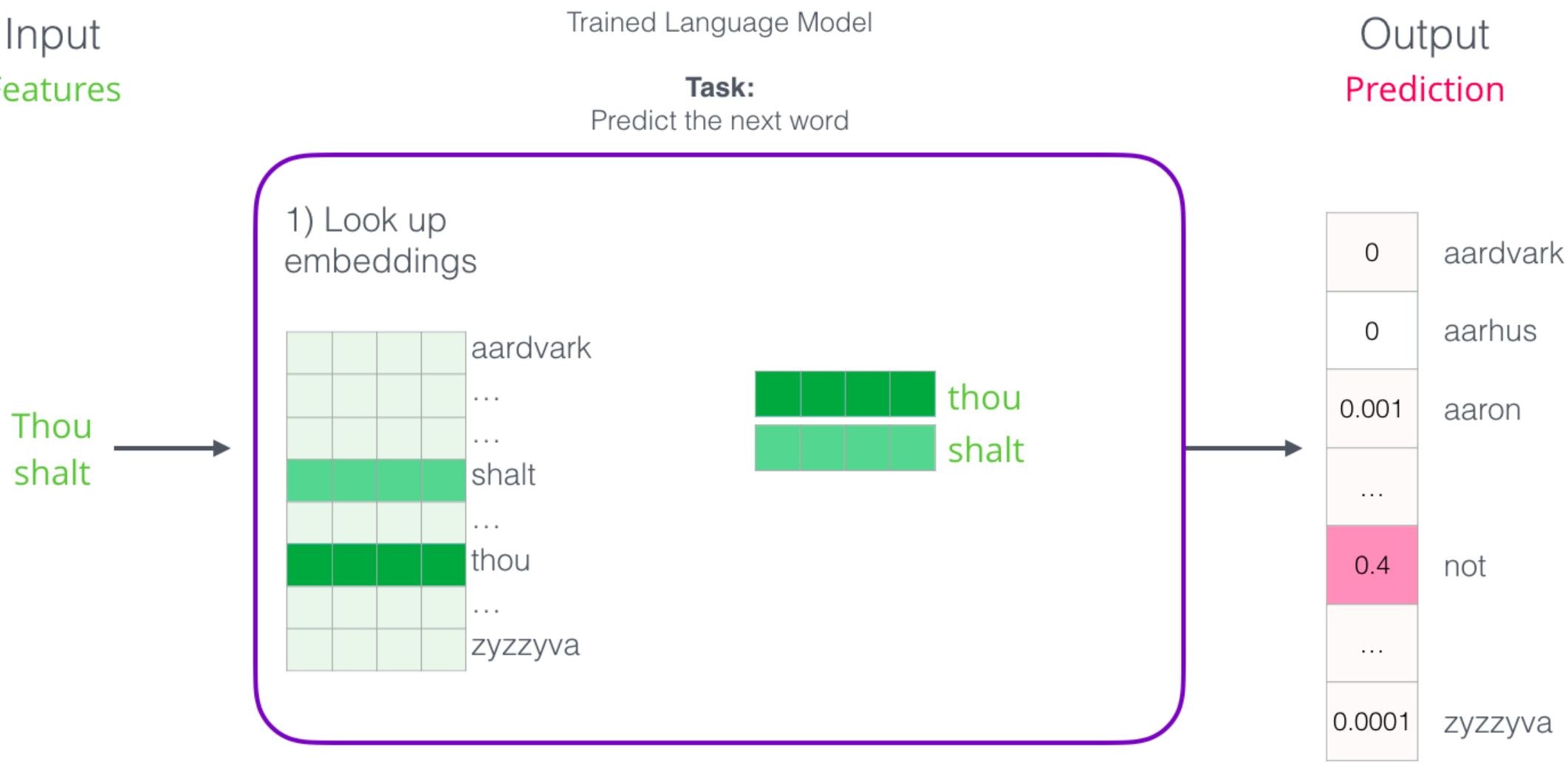
Google





Bengio, 2003

Language Model



Source:http://jalammar.github.io/

Word2Vec

Methods to efficiently create word embeddings

Idea:

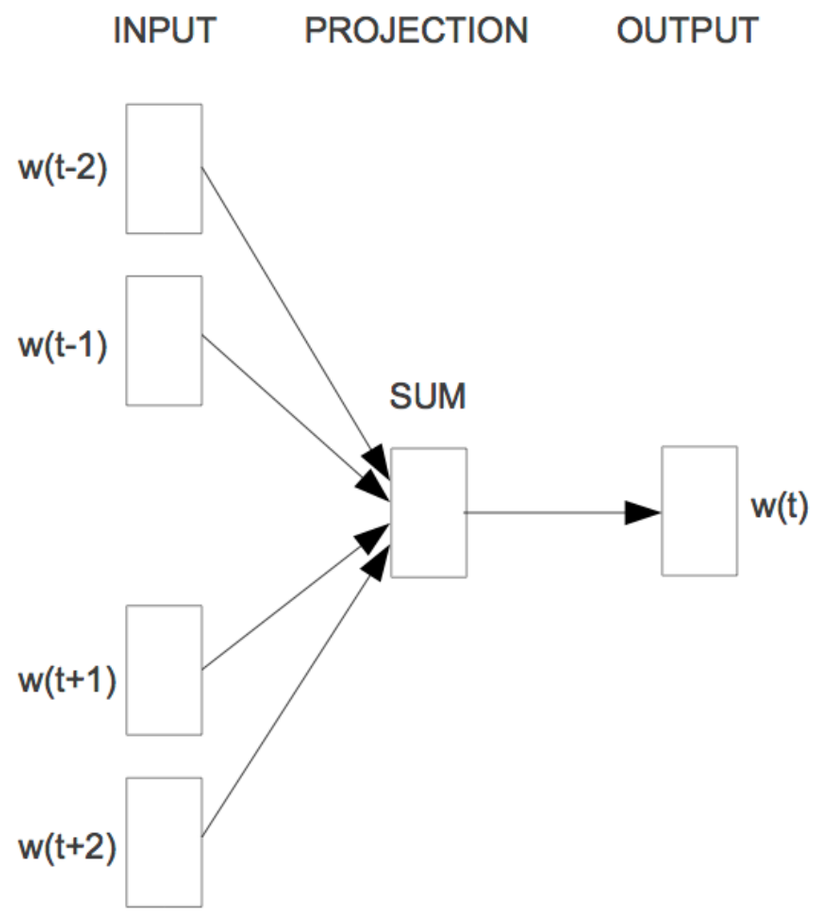
- We have a large corpus of text
- Every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word c and context ("outside") words o
- Use the similarity of the word vectors for c and o to calculate the probability of o given c (or vice versa)
- Keep adjusting the word vectors to maximize this probability

Source: Stanford 224N

Word2Vec - CBOW and Skipgram

Methods to efficiently create word embeddings

If our goal is just to learn word embeddings, instead of only looking words before the target word, we can also look at words after it.



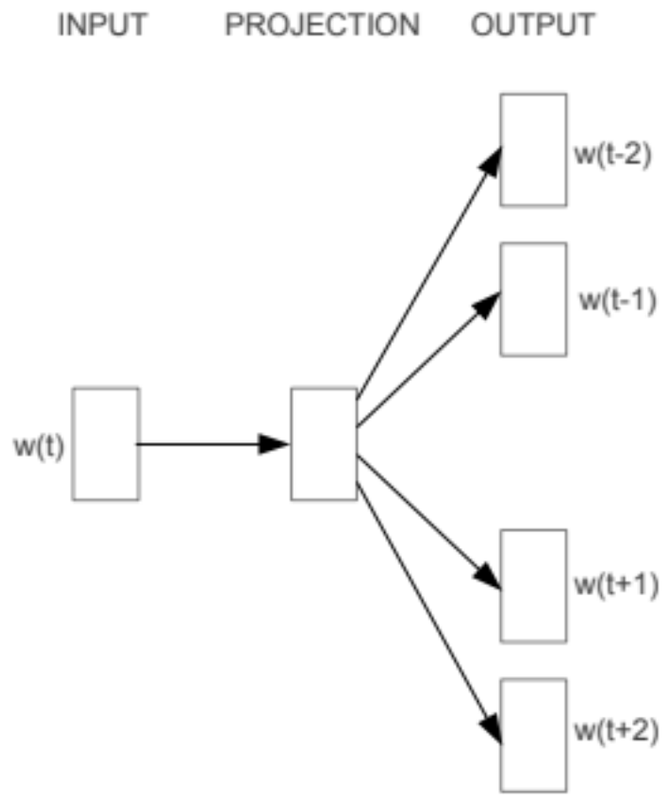
This is called a **Continuous Bag of Words (CBOW)** architecture

Source: Stanford 224N

Word2Vec - CBOW and Skipgram

Methods to efficiently create word embeddings

Instead of guessing a word based on its context (the words before and after it), the other architecture tries to guess neighbouring words using the current word.

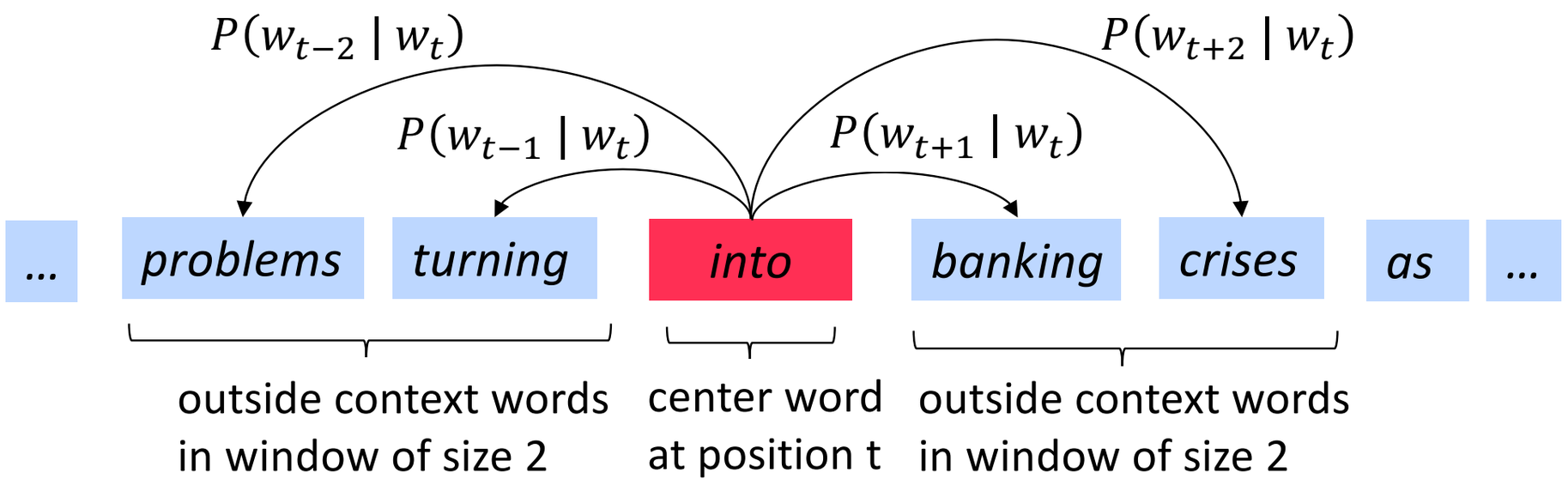


Skip-gram

This is called a Skipgram architecture

Source: Stanford 224N

Word2Vec - skipgram



Source: Stanford 224N

Word2Vec - skipgram

Input-Output training pairs

Thou shalt not make a machine in the likeness of a human mind

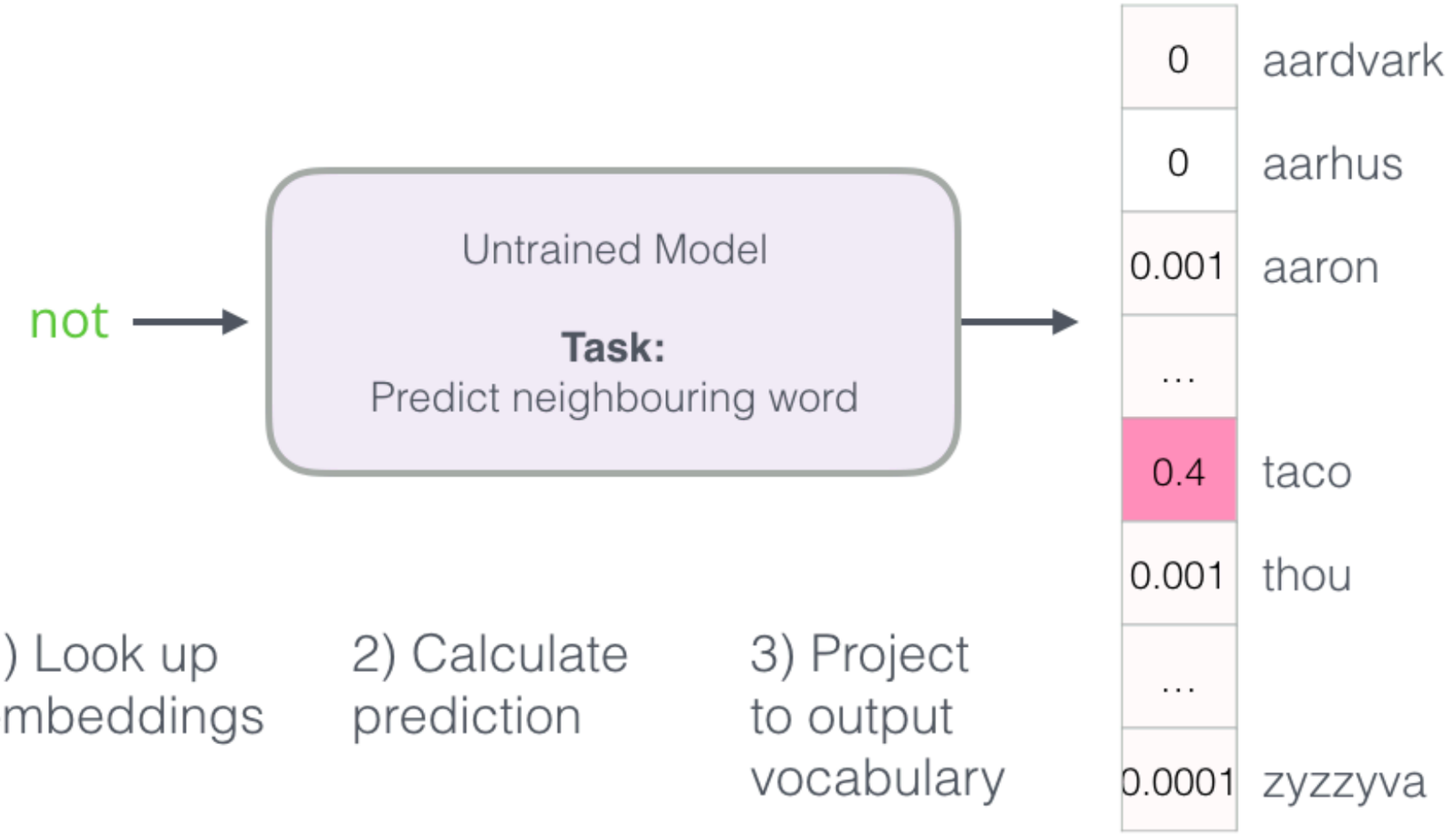
thou	shalt	not	make	a	machine	in	the	…

input word	target word
not	thou
not	shalt
not	make
not	a

Source: http://jalammar.github.io/

Word2Vec - prediction

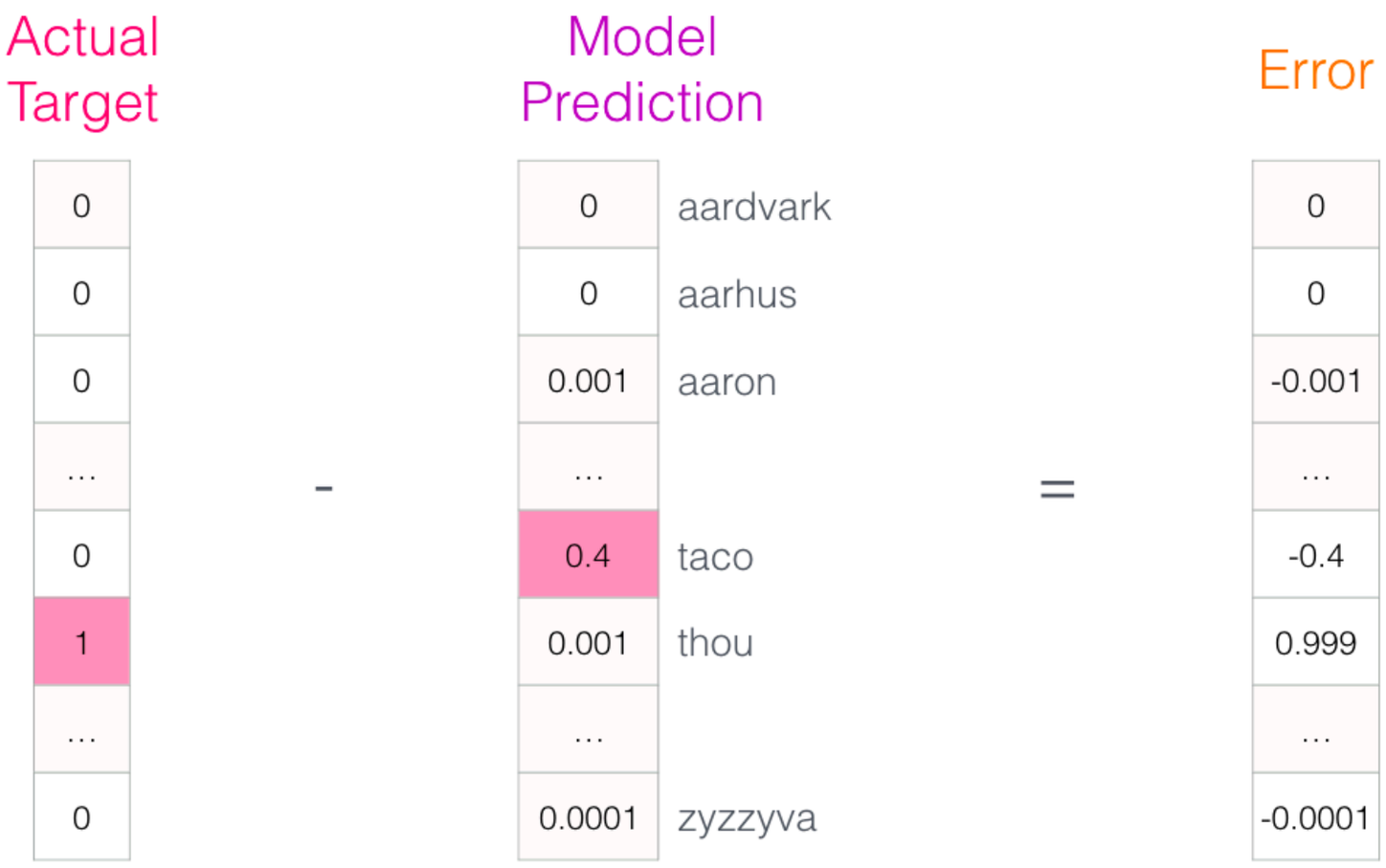
Thou shalt not make a machine in the likeness of a human mind



Word2Vec - Loss/Gradient computation

Thou shalt not make a machine in the likeness of a human mind

$$\log p(o|c) = \log \frac{\exp(u_o{}^T v_c)}{\sum_{w=1}^{V} \exp(u_w{}^T v_c)}$$

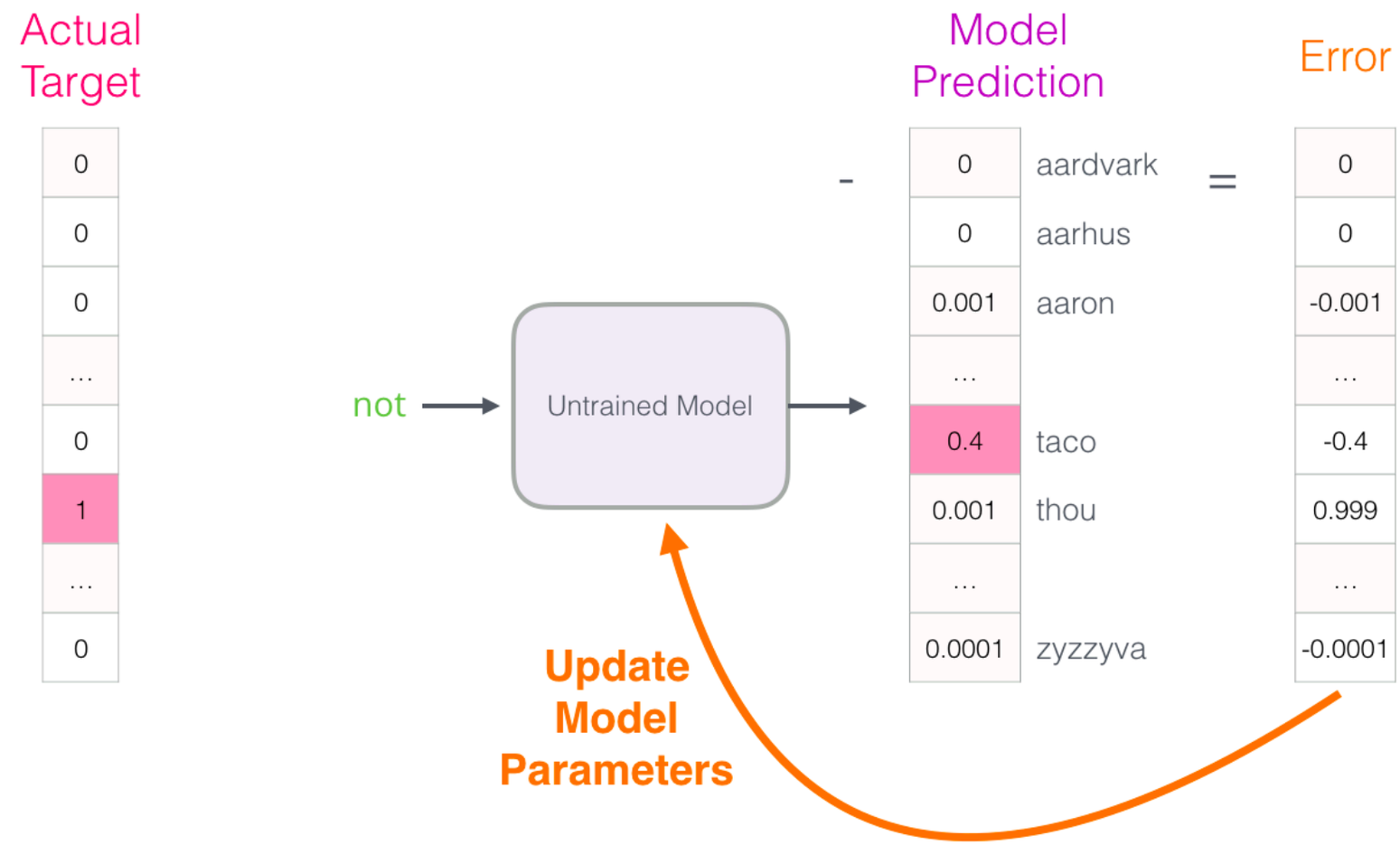


How far off was the model? We subtract the two vectors resulting in an error vector

Source: http://jalammar.github.io/

Word2Vec - training

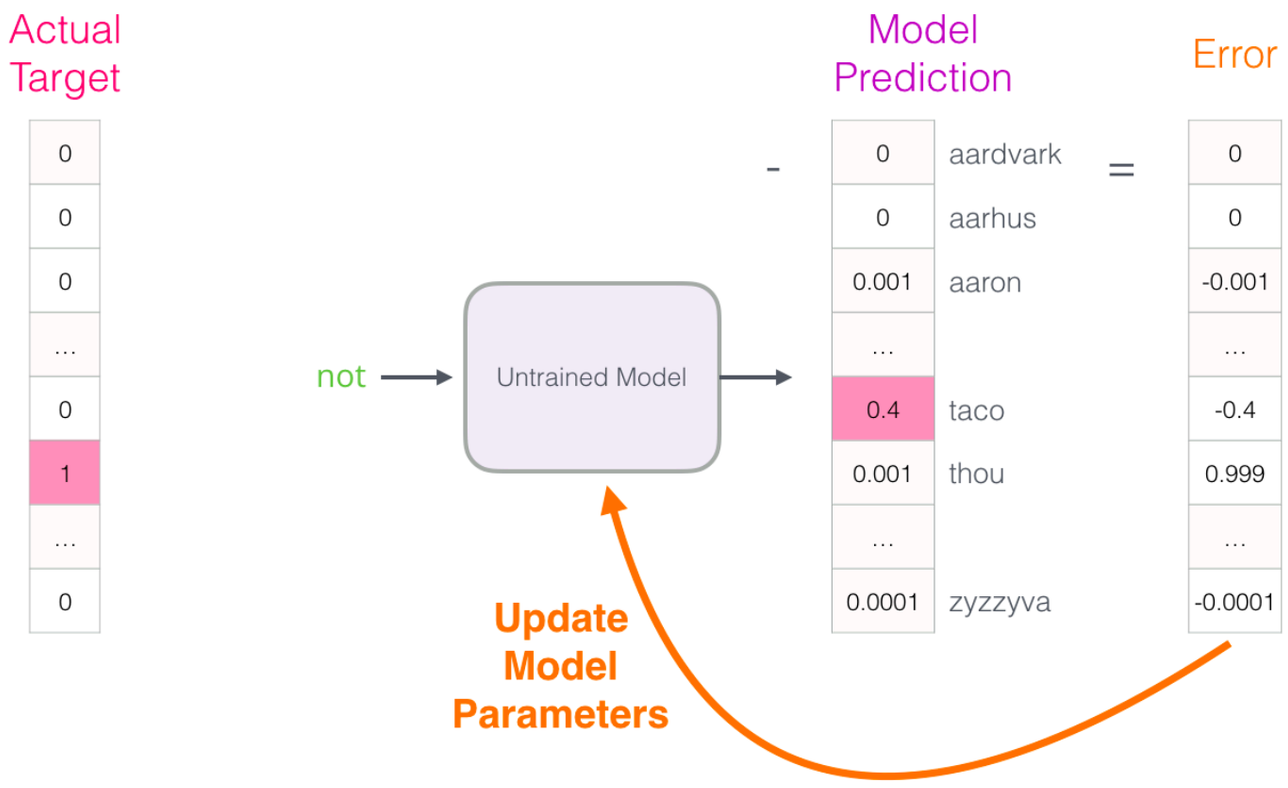
Thou shalt not make a machine in the likeness of a human mind



This error vector can now be used to update the model so the next time, it's a little more likely to guess thou when it gets not as input.

Source: http://jalammar.github.io/

Word2Vec - training



- We proceed to do the same process with the next sample in our dataset, and then the next, until we've covered all the samples in the dataset. That concludes one epoch of training. We do it over again for a number of epochs.
- Then we'd have our trained model and we can extract the embedding matrix from it and use it for any other application.

Source: http://jalammar.github.io/

Where we are

Models/Algorithms

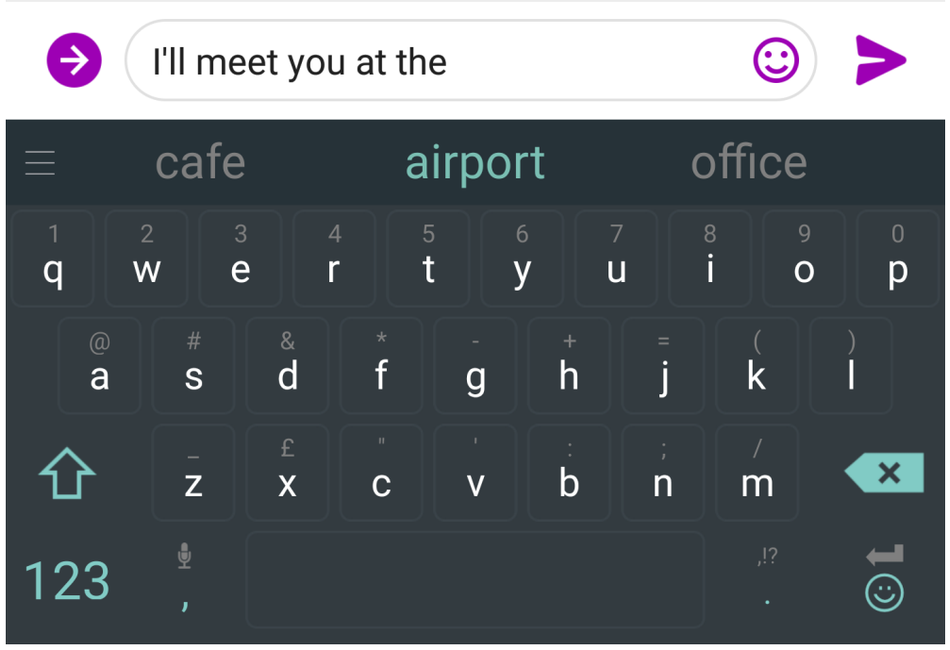
- Linear models
- Feed-forward Neural Nets (FNN)
- **Recurrent Neural Nets**

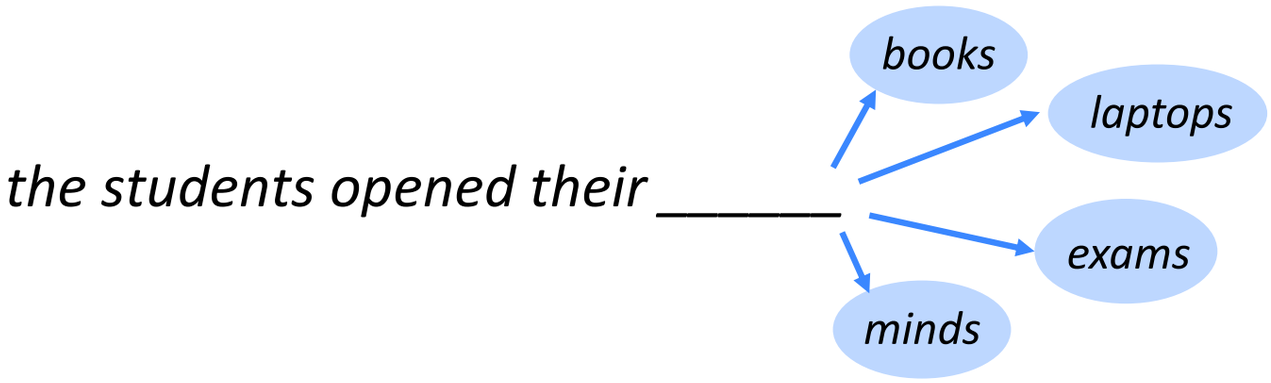
NLP tasks/applications

- Word meaning
- **Language modelling**
- **Sequence tagging**
- **Sequence encoding**

Language Modelling

A language model takes a list of words (history/context), and attempts to predict the word that follows them





Google

what is the

what is the **weather**
what is the **meaning of life**
what is the **dark web**
what is the **xfl**
what is the **doomsday clock**
what is the **weather today**
what is the **keto diet**
what is the **american dream**
what is the **speed of light**
what is the **bill of rights**

Google Search I'm Feeling Lucky

Source: Stanford 224n

Language Modelling

A language model takes a list of words (history/context), and attempts to predict the word that follows them

More formally: given a sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$, compute the probability distribution of the next word $\boldsymbol{x}^{(t+1)}$:

$$P(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

where $\boldsymbol{x}^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|V|}\}$

Source: Stanford 224n

Language Modeling

Language Models (LM) also assigns a probability to a piece of text.

For example, if we have some text $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$, then the probability of this text (according to the Language Model) is:

$$P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}) = P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} | \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} | \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)})$$

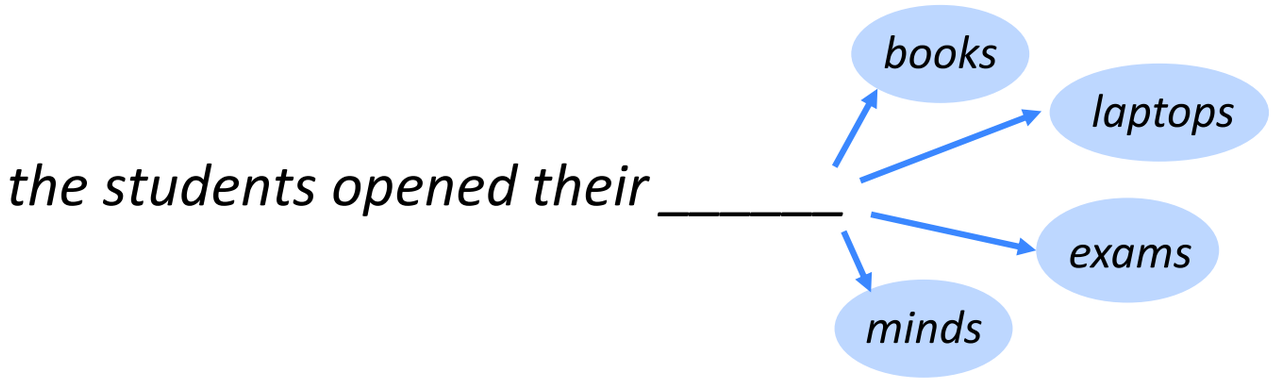
$$= \prod_{t=1}^{T} P(\boldsymbol{x}^{(t)} | \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)})$$

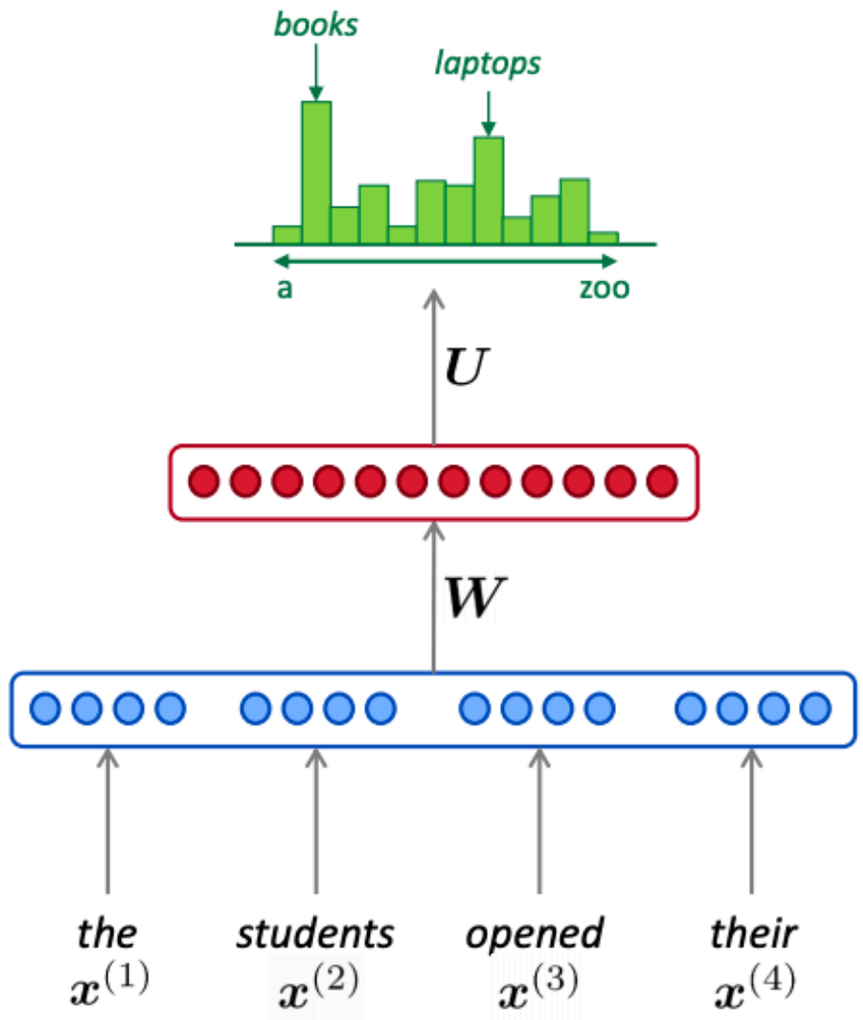
This is what our LM provides

Source: Stanford 224n

Language Modeling with Window Based Approach (Revisit)

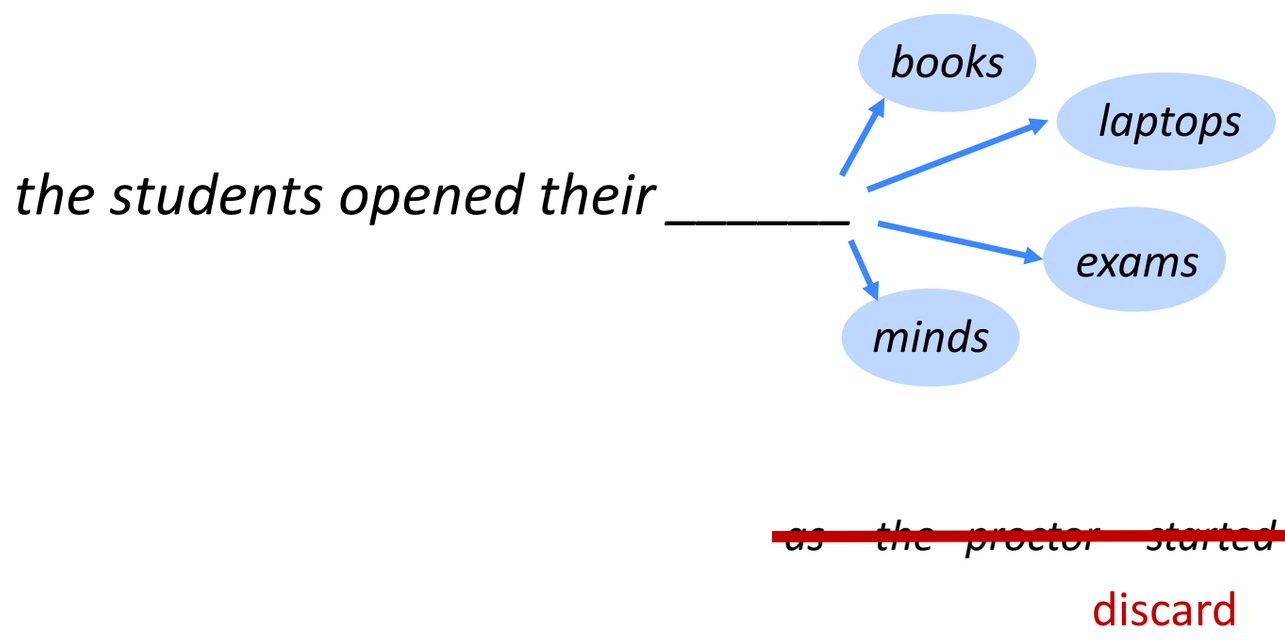
A language model takes a list of words (history/context), and attempts to predict the word that follows them

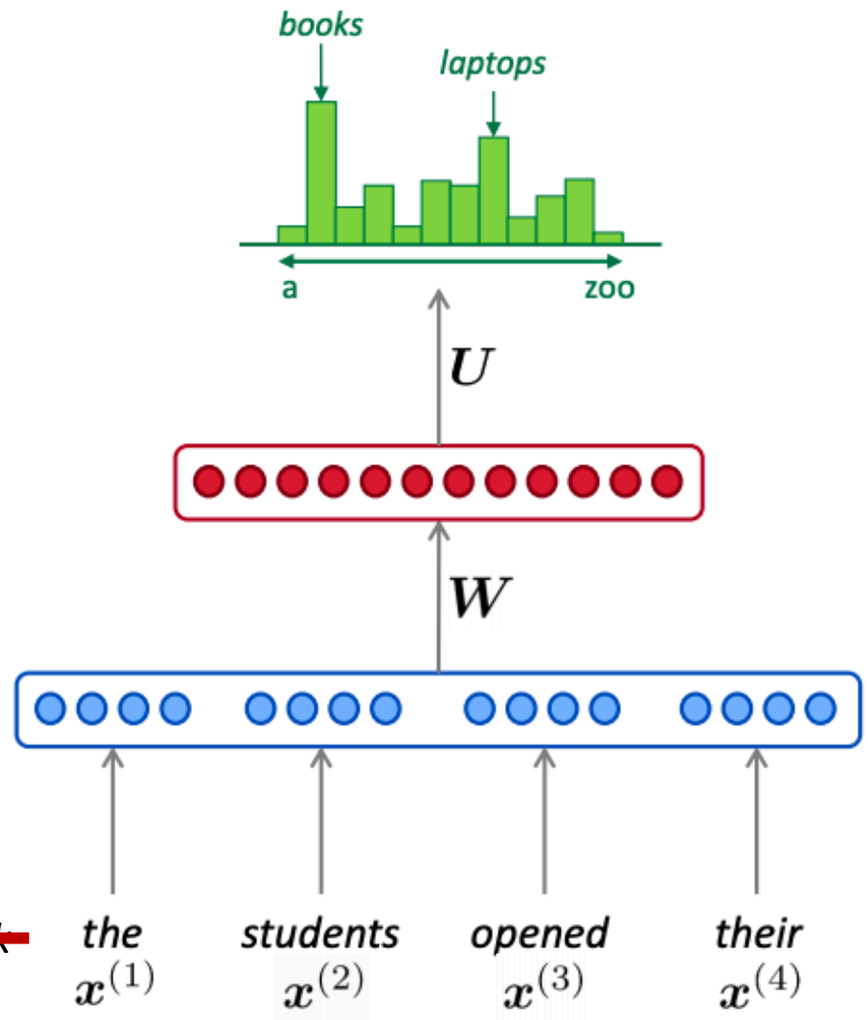




Language Modeling with Window Based Approach (Revisit)

A language model takes a list of words (history/context), and attempts to predict the word that follows them





Source: Stanford 224n

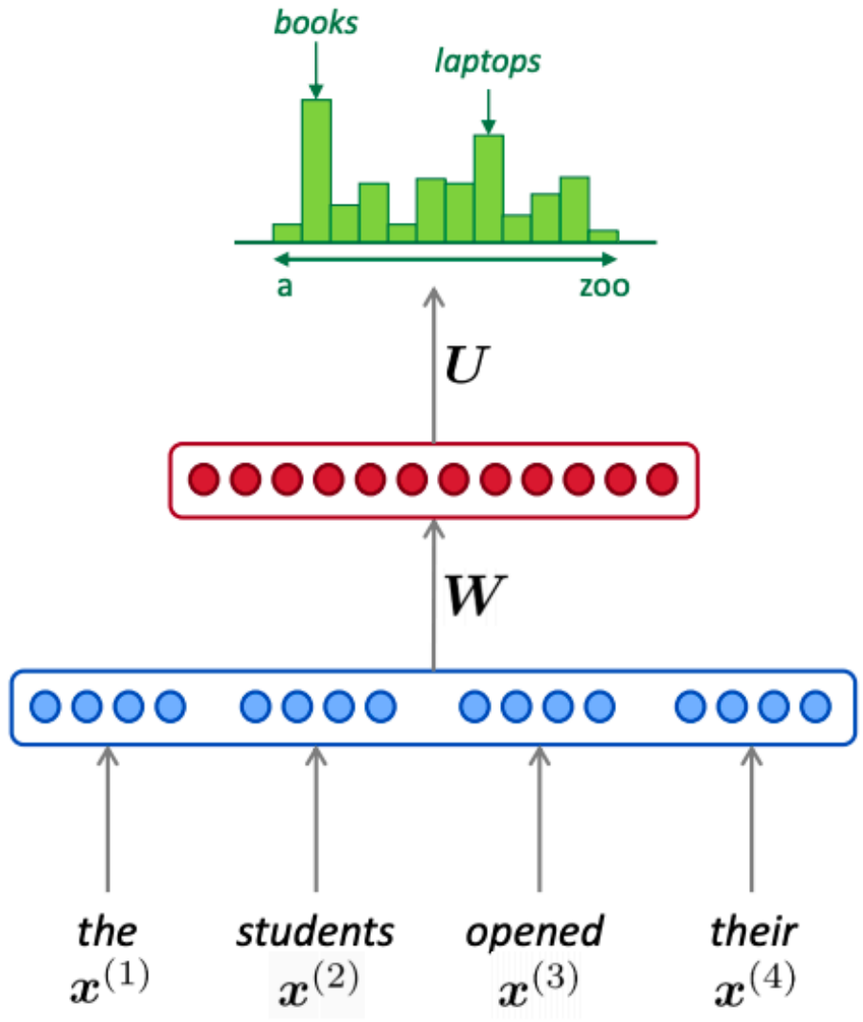
Language Modeling with Window Based Approach (Revisit)

Improvements over n-gram LM:

- No sparsity problem
- Don't need to store the n-grams

Remaining **problems**:

- Fixed window is too small
- Window can never be large enough! Enlarging window enlarges W
- Each input vector is multiplied by completely different weights. No symmetry in how the inputs are processed.
- Convolution? Not suitable for variable length sequences

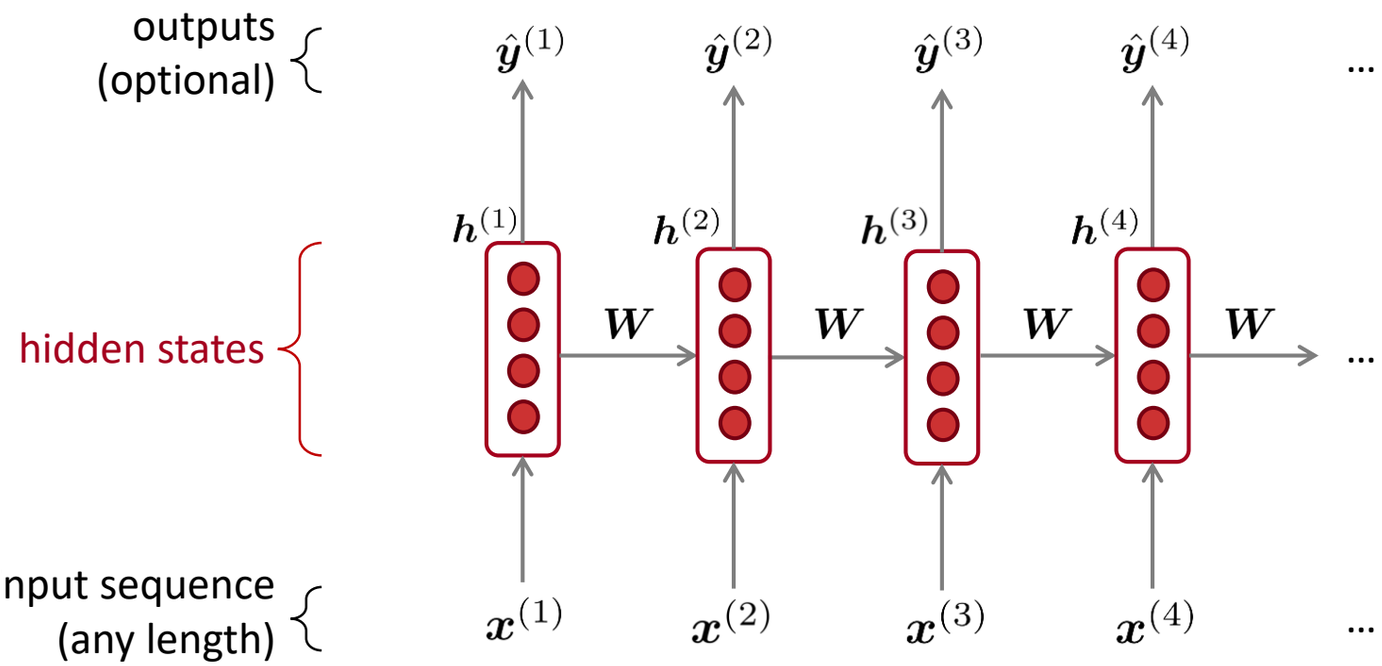


We need a neural architecture that can process any length input

Source: Stanford 224n

Recurrent Neural Networks

Main idea: Apply the same weights repeatedly (recurrently)



Notice:

- Output at each step is optional
- Recurrence is done with hidden states

Source: Stanford 224n

A RNN Language Model

output distribution

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

$\boldsymbol{h}^{(0)}$ is the initial hidden state

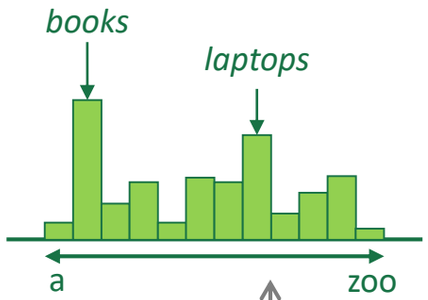
word embeddings

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

words / one-hot vectors

$$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$$

$\hat{\boldsymbol{y}}^{(4)} = P(\boldsymbol{x}^{(5)}|\text{the students opened their})$



the input sequence can be of arbitrary length

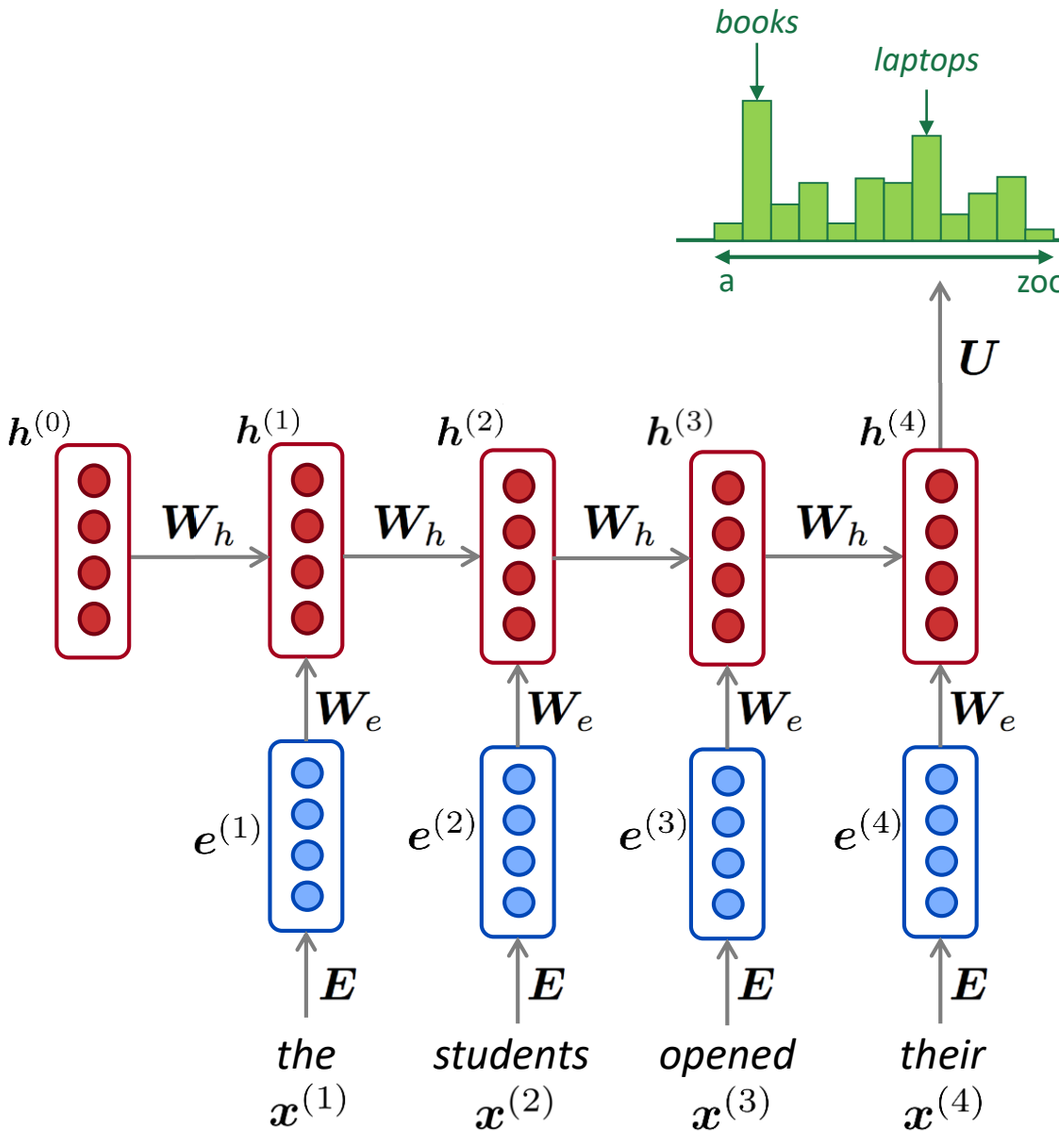
Language Modeling with RNNs

Advantages:

- Can process any length input
- Computation at step t can (in theory) use information from many steps back
- Model size doesn't increase for longer input
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

Problems:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back (remembers only recent states)



Source: Stanford 224n

Training a RNN-LM

- Get a big corpus of text which is a sequence of words $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$
- Feed into RNN-LM; compute output distribution $\hat{\boldsymbol{y}}^{(t)}$ for *every step t*.
  - i.e. predict probability dist of *every word*, given words so far

- Loss function on step *t* is cross-entropy between predicted probability distribution $\hat{\boldsymbol{y}}^{(t)}$, and the true next word $\boldsymbol{y}^{(t)}$ (one-hot for $\boldsymbol{x}^{(t+1)}$):

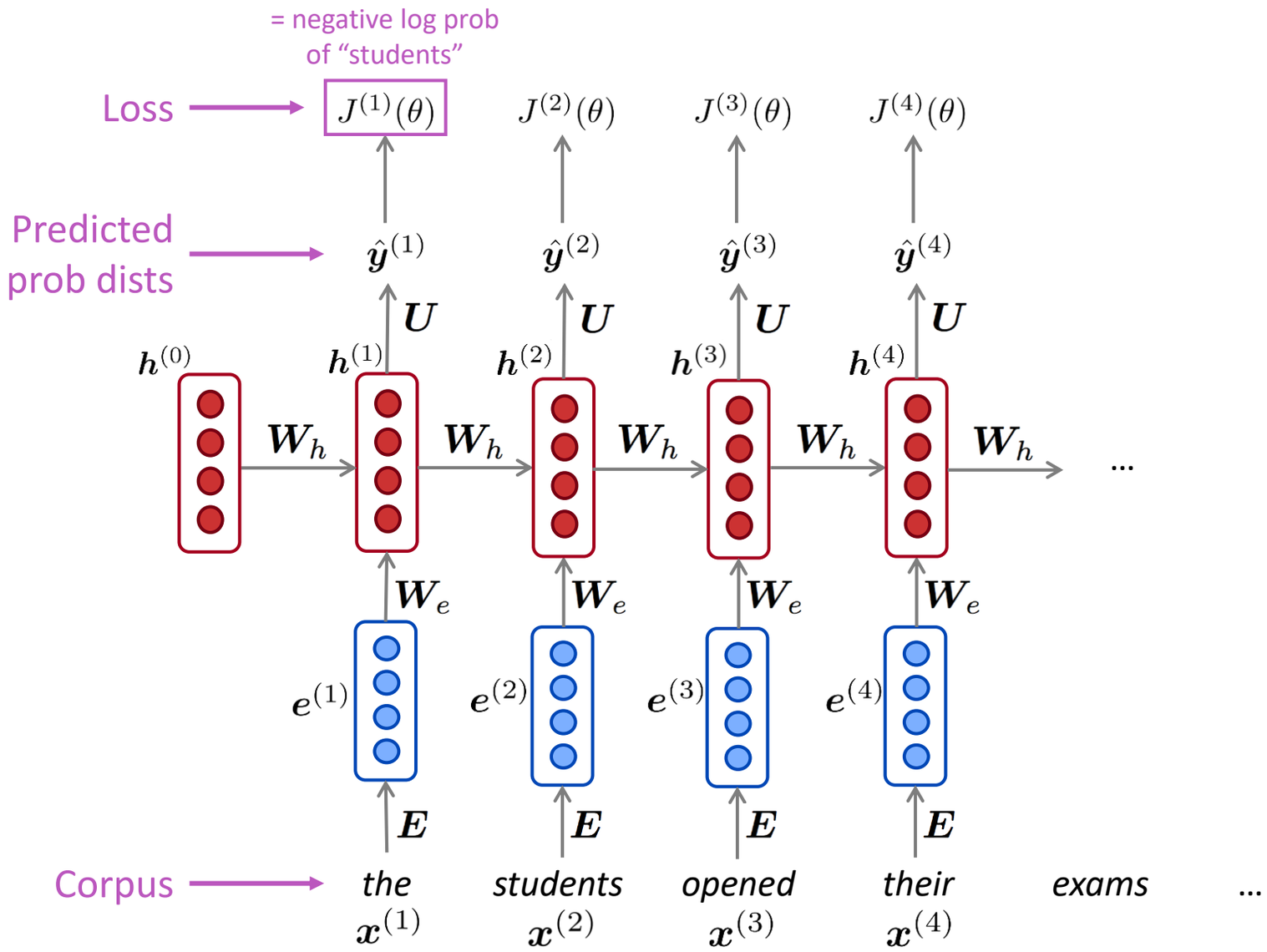
$$J^{(t)}(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{w \in V} \boldsymbol{y}_w^{(t)} \log \hat{\boldsymbol{y}}_w^{(t)} = -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

- Average this to get overall loss for entire training set:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

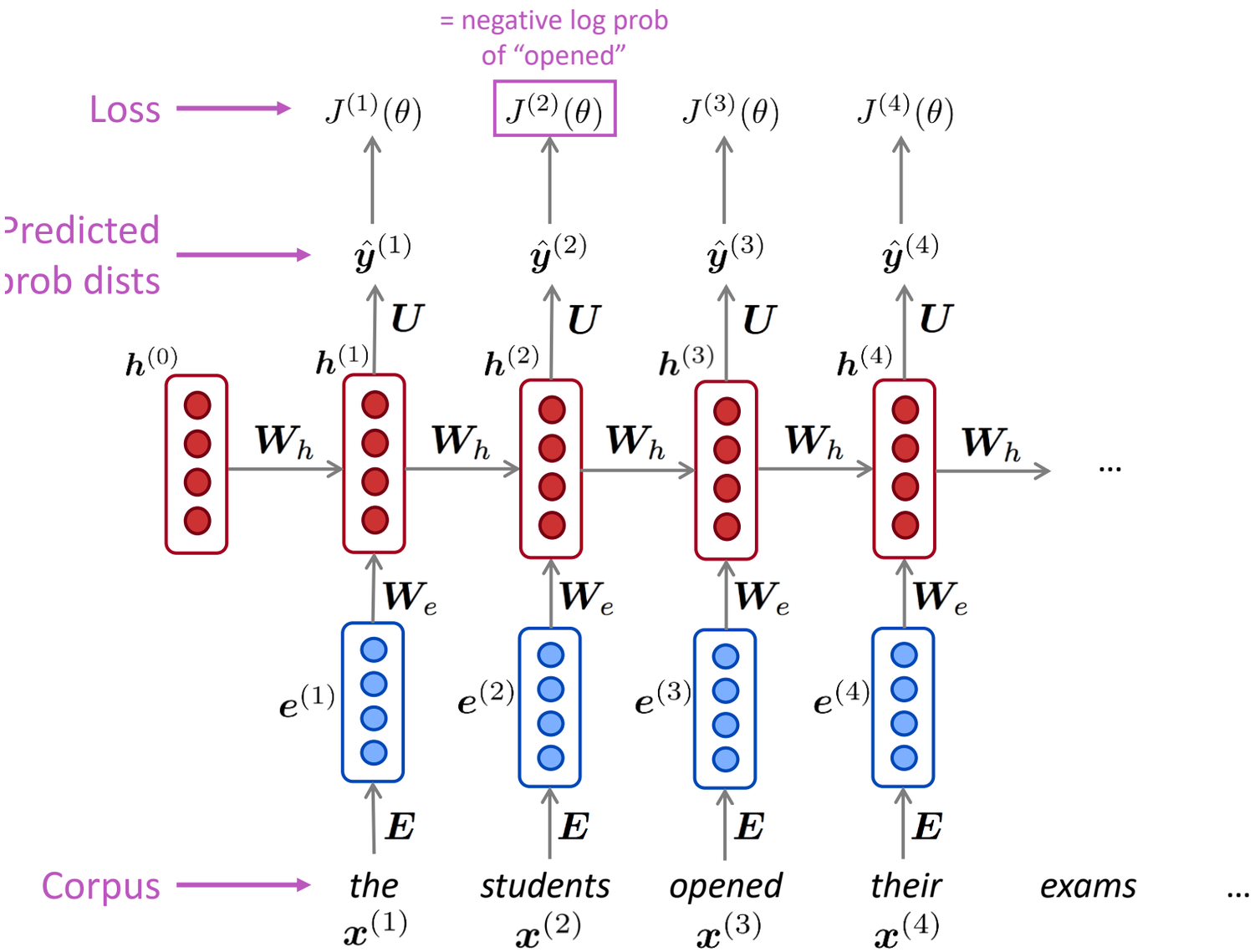
Source: Stanford 224n

Training a RNN-LM



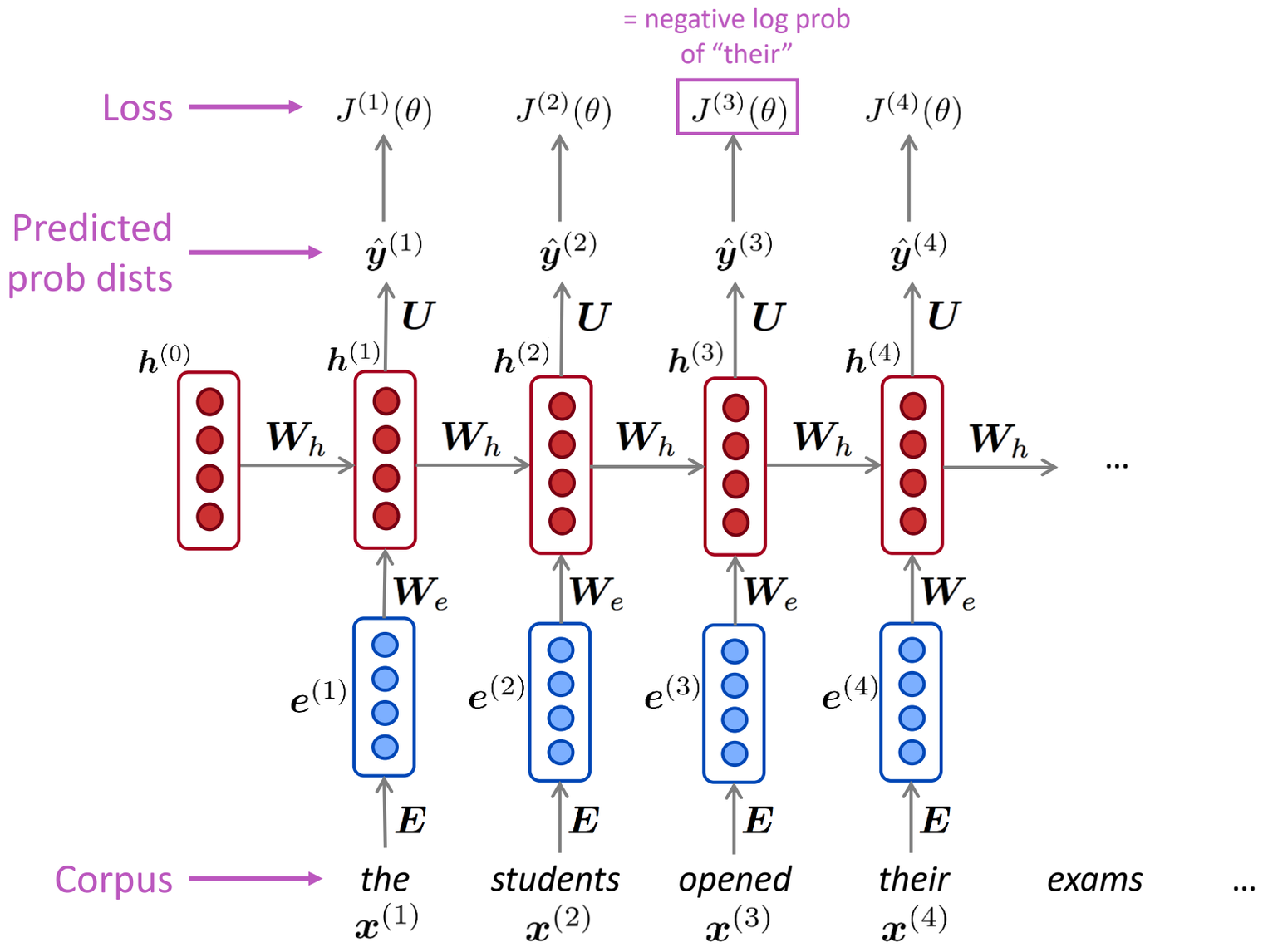
Source: Stanford 224n

Training a RNN-LM



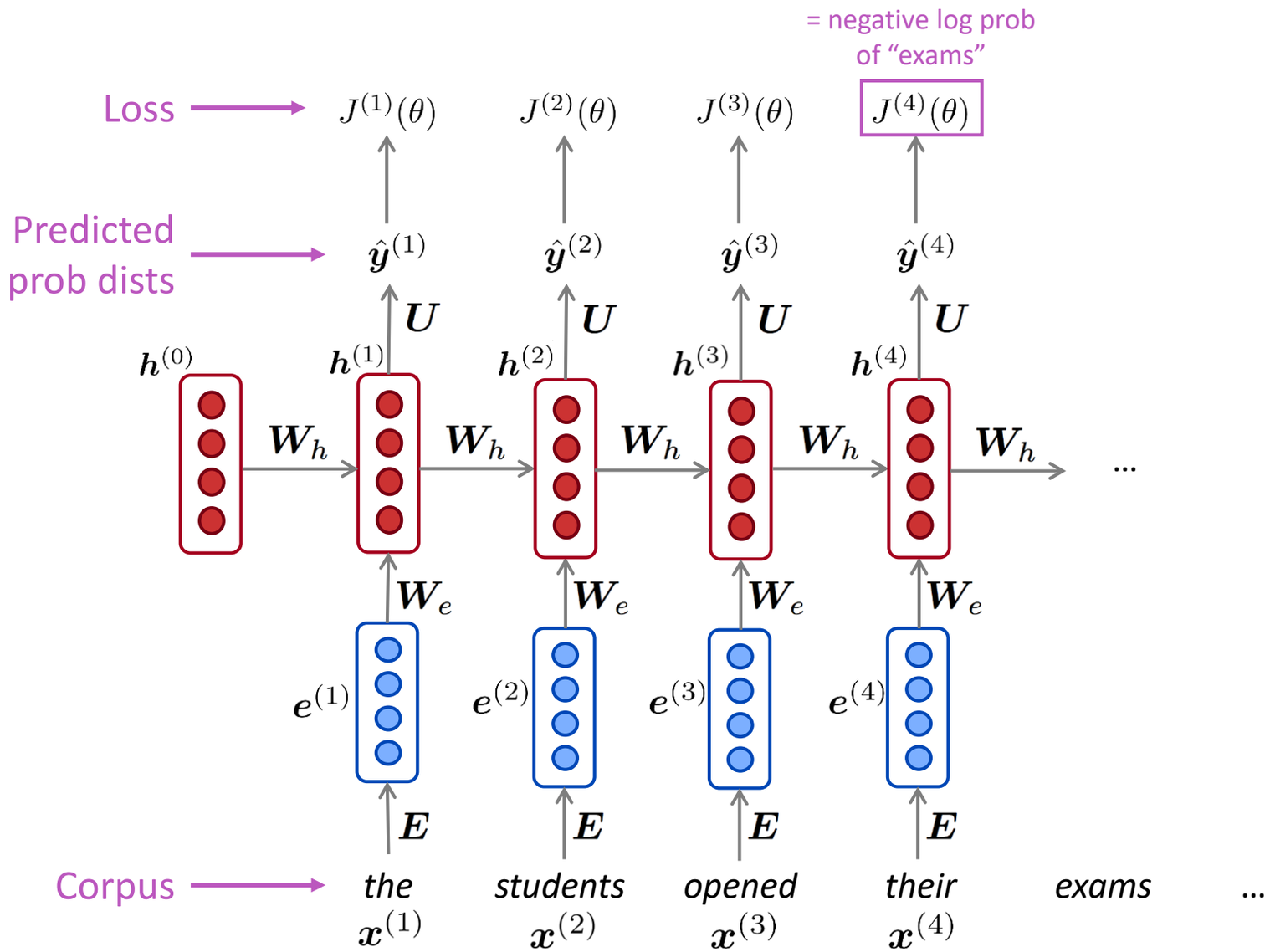
Source: Stanford 224n

Training a RNN-LM



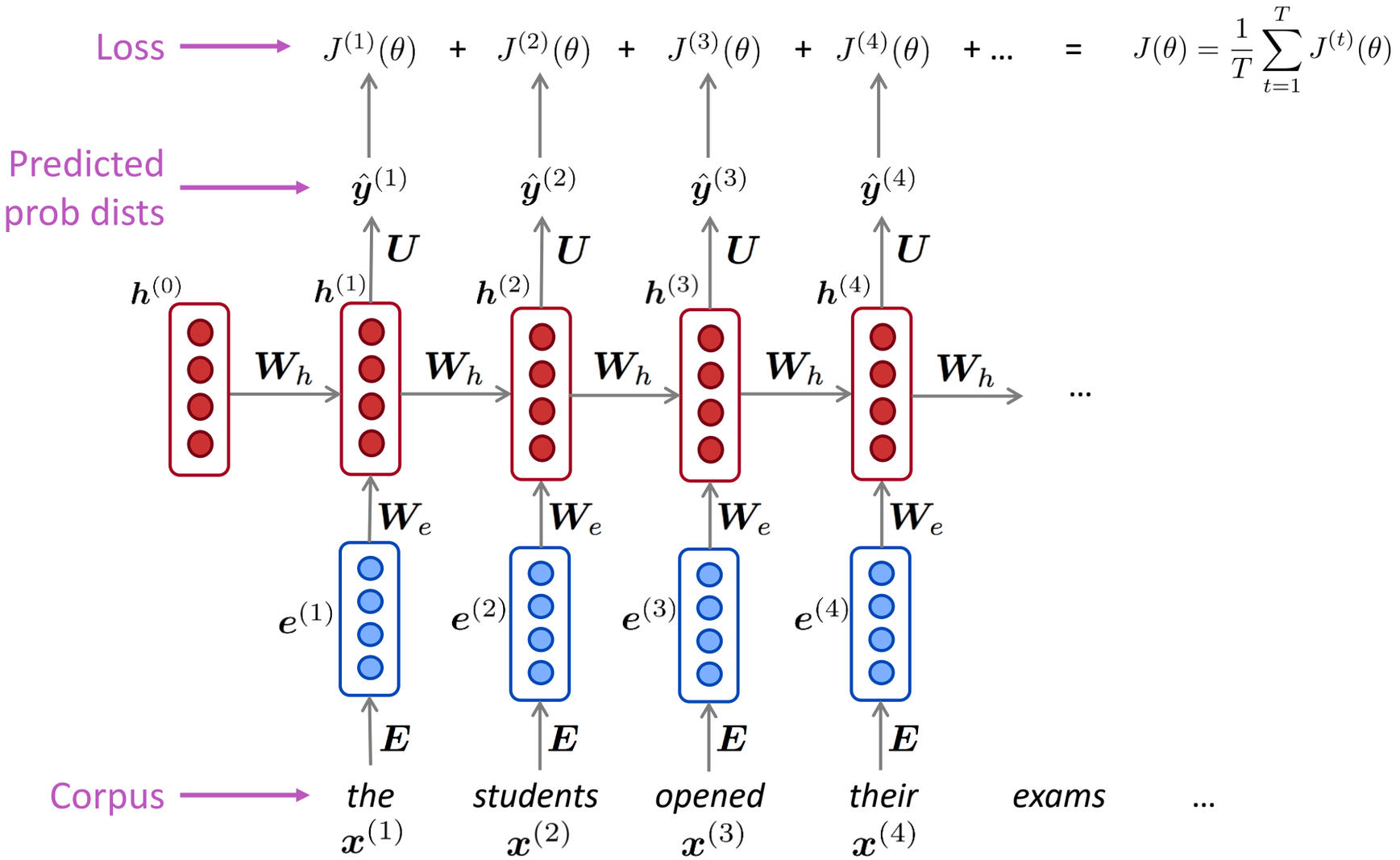
Source: Stanford 224n

Training a RNN-LM



Source: Stanford 224n

Training a RNN-LM



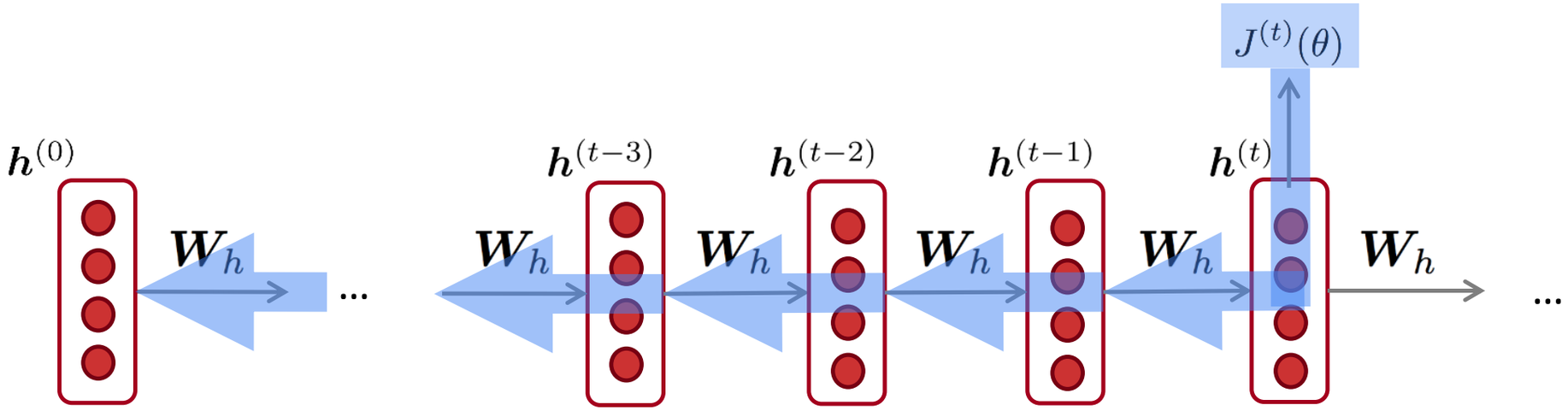
Source: Stanford 224n

Training a RNN-LM

- However: Computing loss and gradients across the entire corpus is way too expensive (no parallelisation)
- In practice, consider as a sentence (or a fixed length sequence)
- In SGD, we compute loss and gradients for batches and update, and then repeat

Source: Stanford 224n

Backpropagation for RNNs

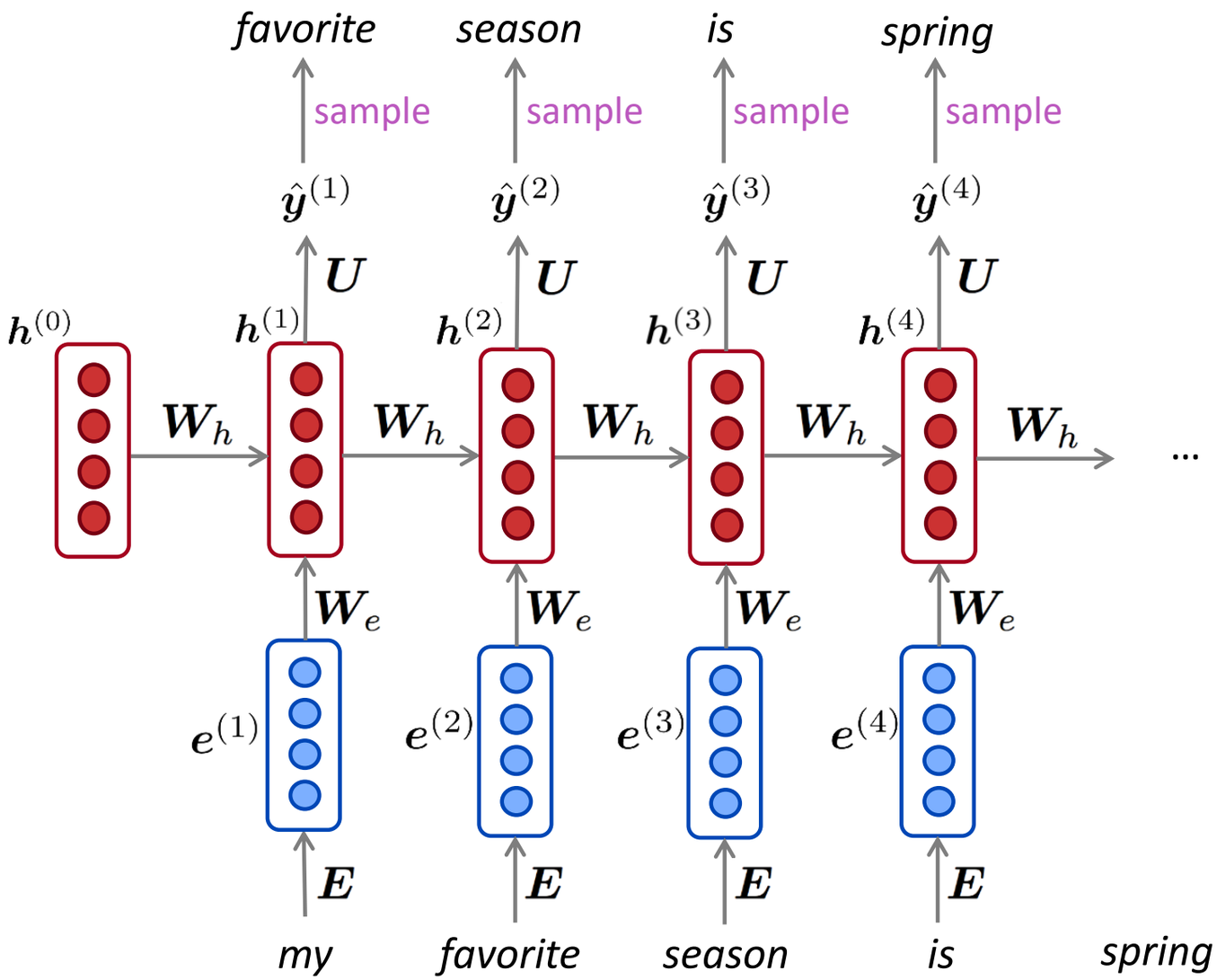


Backpropagate over timesteps i=t,...,0, summing gradients as you go.

This algorithm is called **"backpropagation through time"**

Source: Stanford 224n

Generating Text with RNN-LM



Just like a n-gram LM, you can use an RNN-LM to generate text by repeated sampling. Sampled output is next step's input.

Source: Stanford 224n

Why Language Modeling is Important

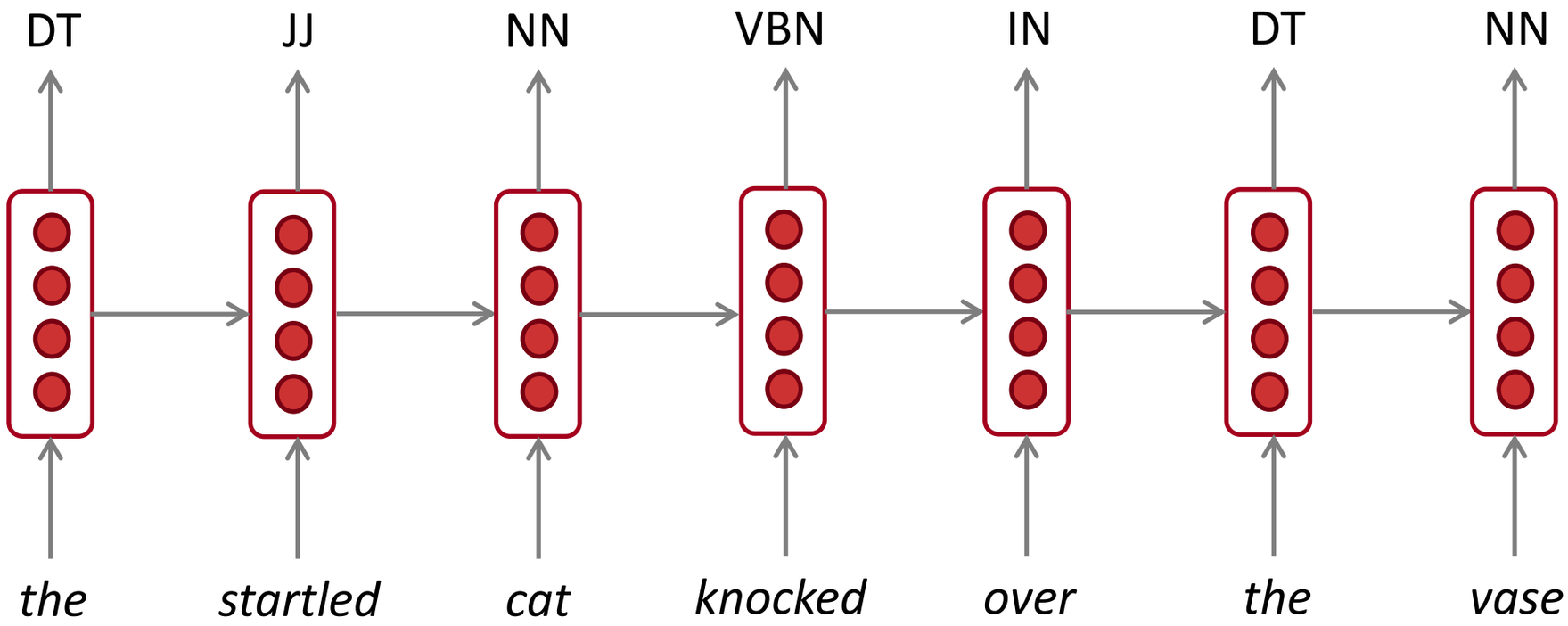
- A benchmark task to track our progress on understanding language
- An important component of many NLP tasks, especially those involving generating text or estimating the probability of a text

  - Speech recognition
  - Spelling/grammar correction
  - Authorship identification
  - Machine translation
  - Summarization
  - Dialogue
  - etc.

Source: Stanford 224n

Sequence Tagging with RNNs

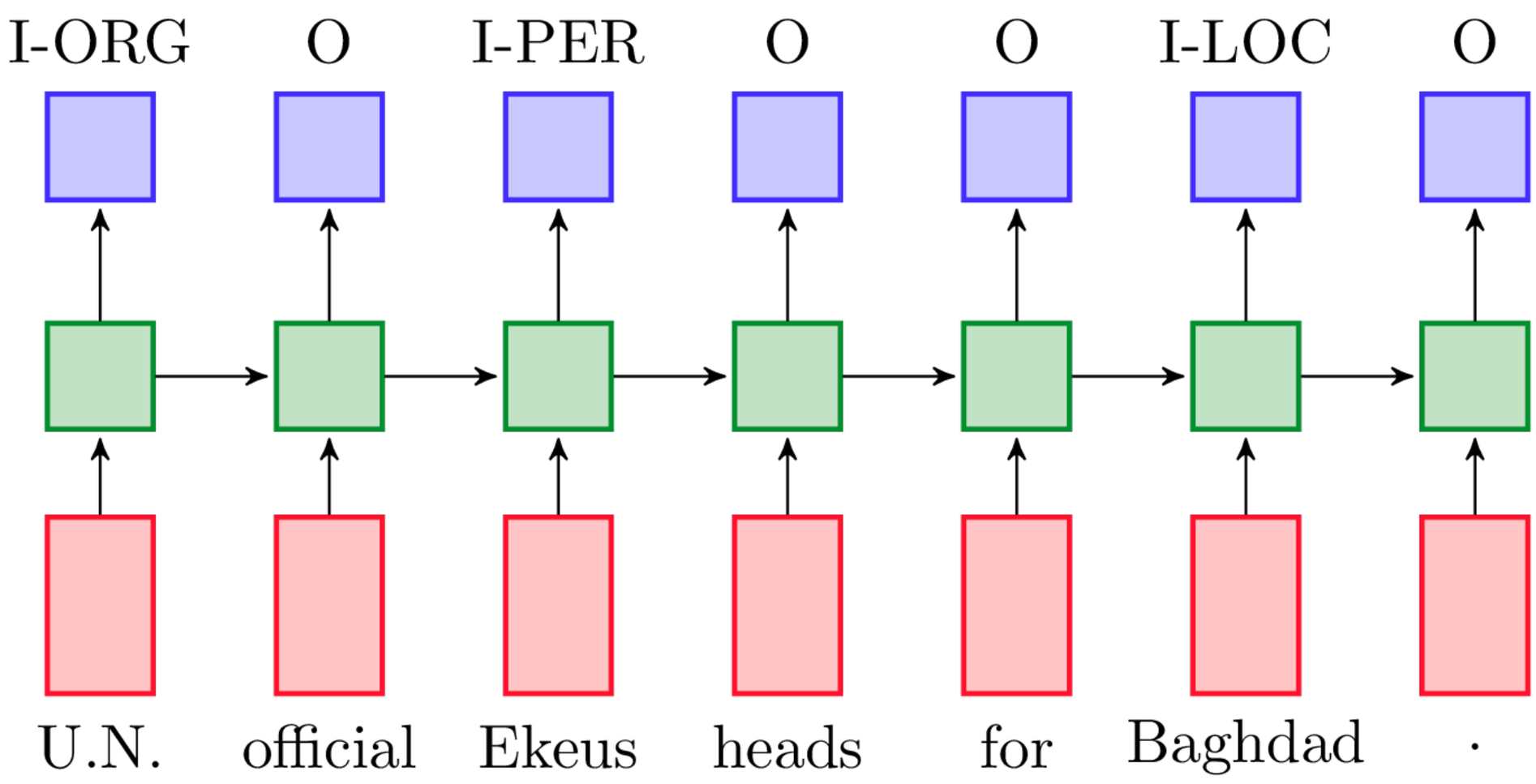
- POS Tagging:



Source: Stanford 224n

Sequence Tagging with RNNs

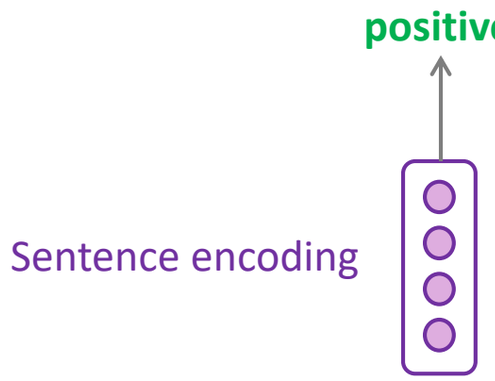
- NER Tagging:



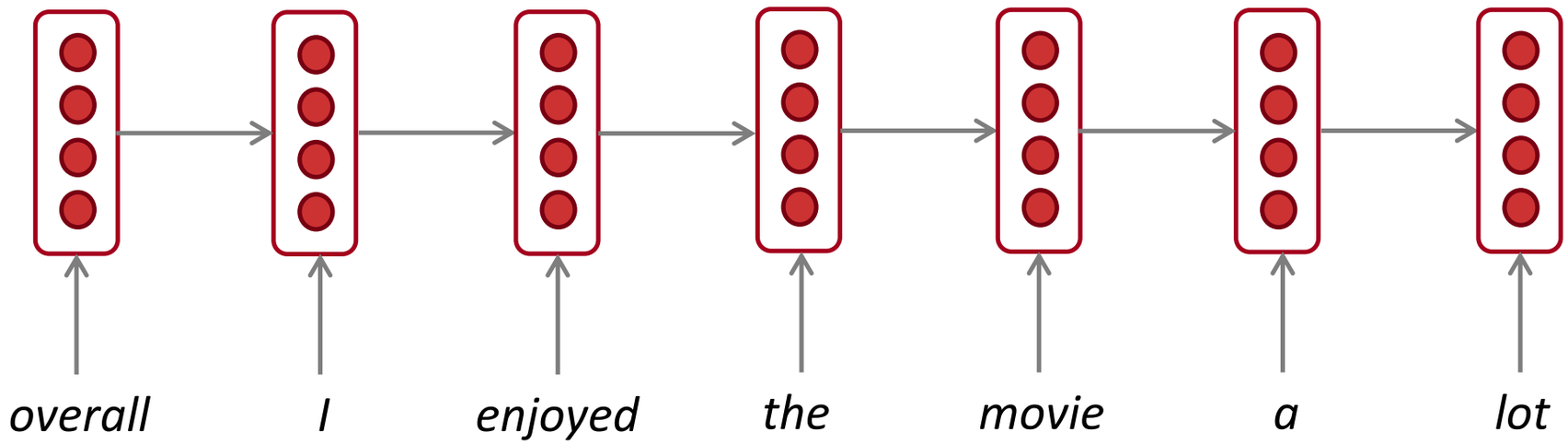
Source: https://www.depends-on-the-definition.com/guide-sequence-tagging-neural-networks-python/

Sequence Classification with RNNs

- Sentiment Classification:



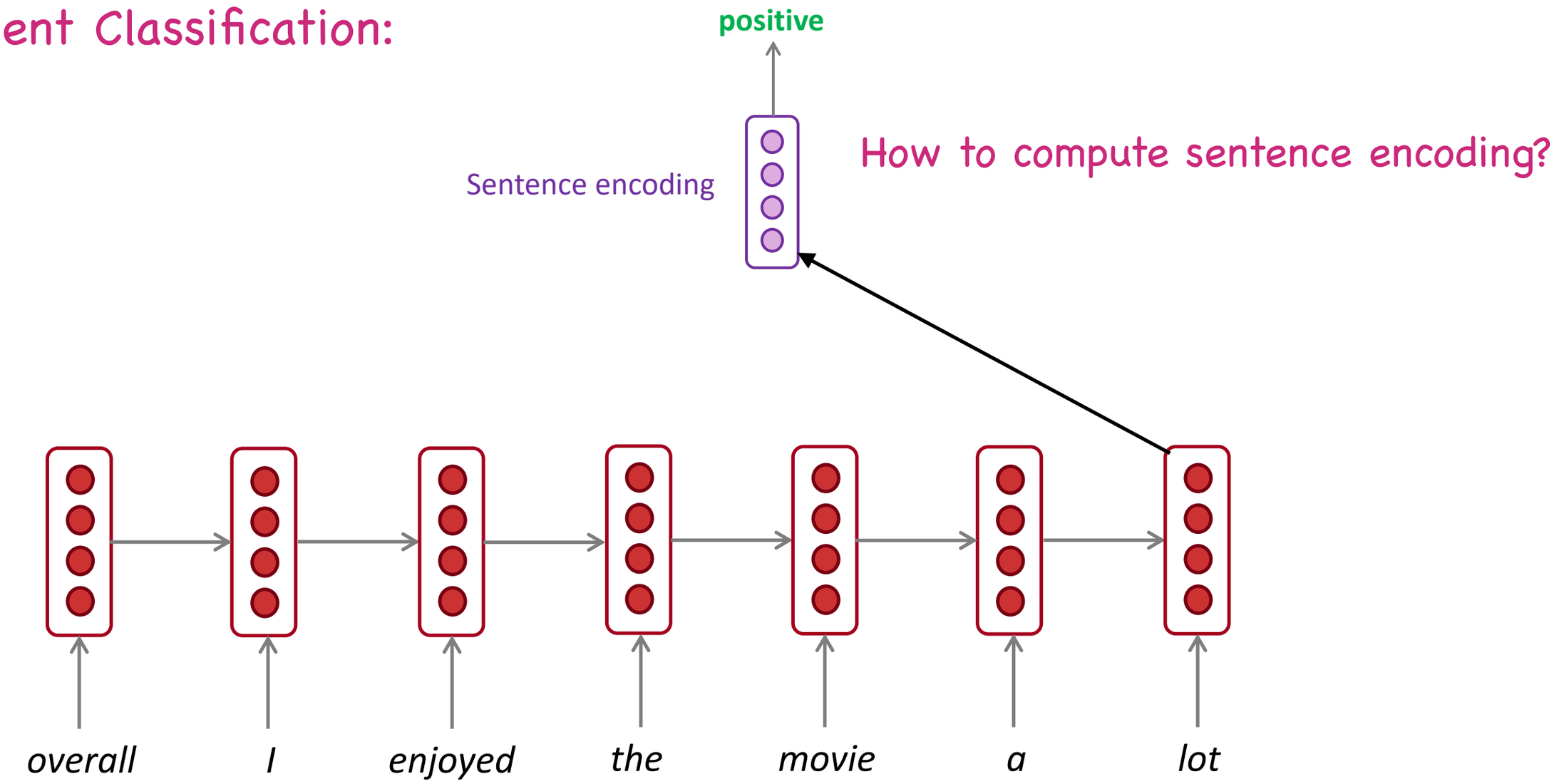
How to compute sentence encoding?



Source: Stanford 224n

Sequence Classification with RNNs

- Sentiment Classification:

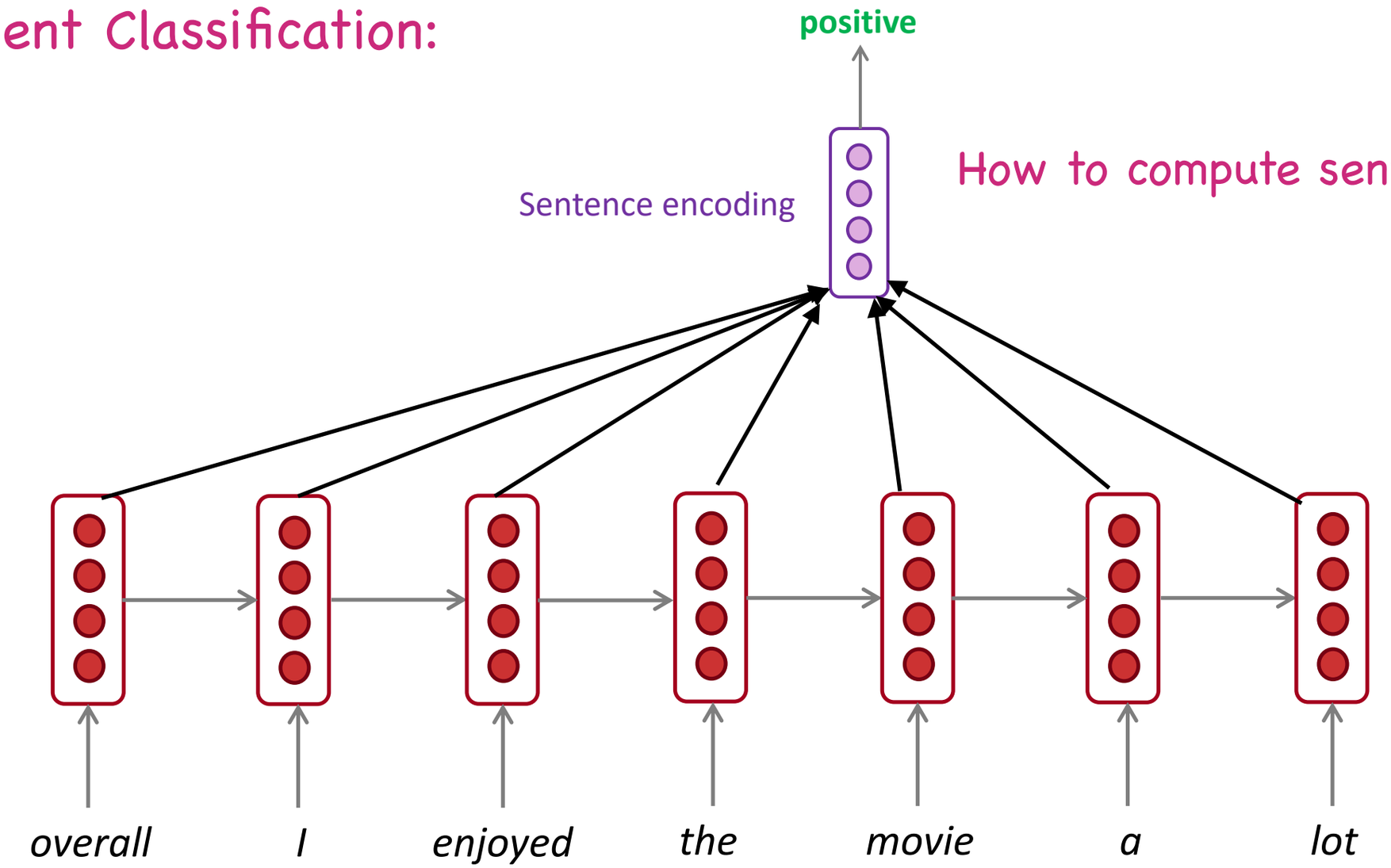


Use the final hidden state as the summary of the sentence

Source: Stanford 224n

Sequence Classification with RNNs

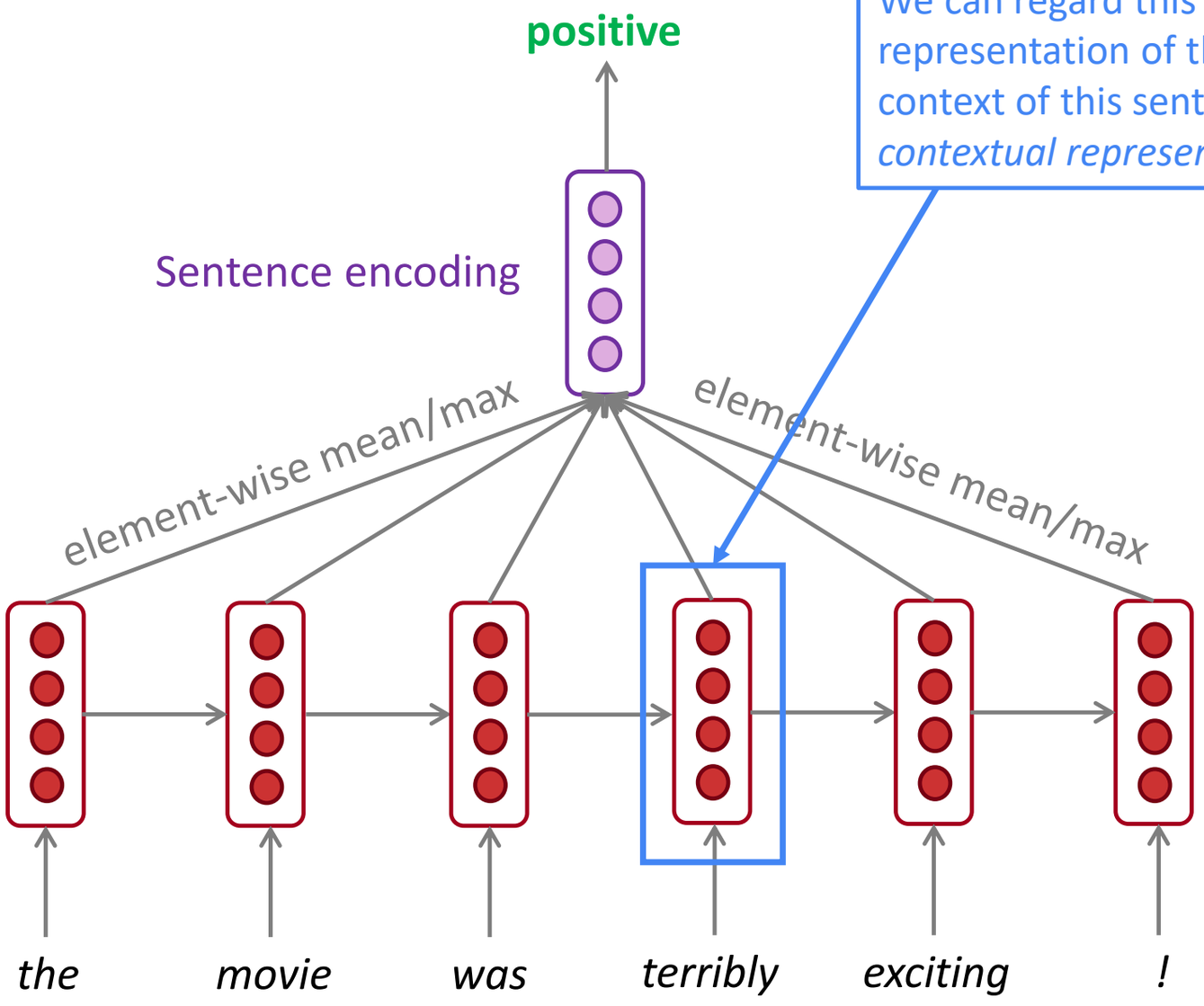
- Sentiment Classification:



Use max/average pooling as the summary of the sentence

Source: Stanford 224n

Bi-directional RNN: Motivation



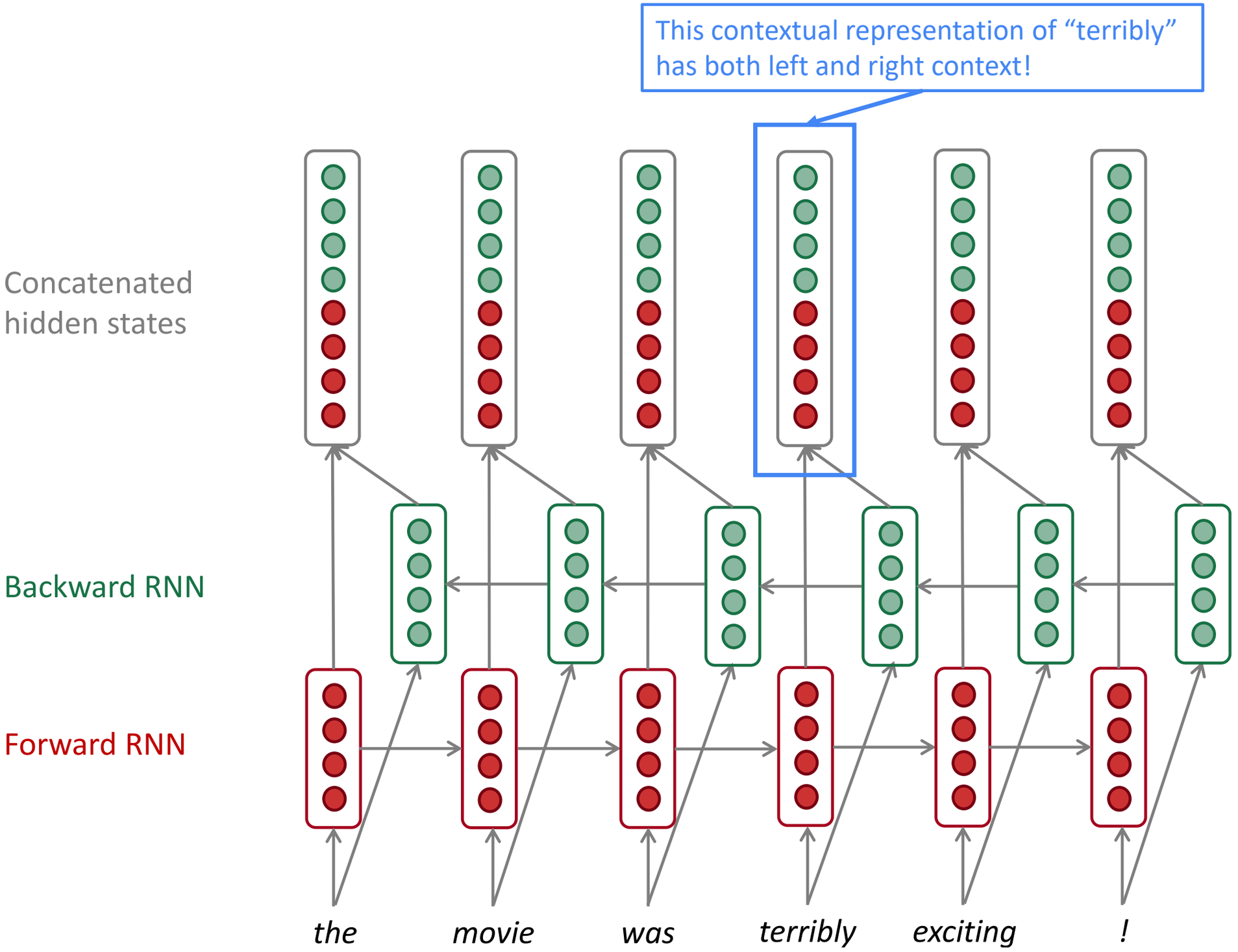
We can regard this hidden state as a representation of the word *"terribly"* in the context of this sentence. We call this a *contextual representation.*

These contextual representations only contain information about the *left* context (e.g. *"the movie was"*).

What about *right* context?

In this example, *"exciting"* is in the right context and this modifies the meaning of *"terribly"* (from negative to positive)

Bi-directional RNN



Bi-directional RNN

Forward RNN $\overrightarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{FW}}(\overrightarrow{\boldsymbol{h}}^{(t-1)}, \boldsymbol{x}^{(t)})$

Backward RNN $\overleftarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{BW}}(\overleftarrow{\boldsymbol{h}}^{(t+1)}, \boldsymbol{x}^{(t)})$

Generally, these two RNNs have separate weights

Concatenated hidden states $\boldsymbol{h}^{(t)} = [\overrightarrow{\boldsymbol{h}}^{(t)}; \overleftarrow{\boldsymbol{h}}^{(t)}]$

We regard this as “the hidden state” of a bidirectional RNN. This is what we pass on to the next parts of the network.

Source: Stanford 224n

Bi-directional RNN

- Bidirectional RNNs are only applicable when you have access to the entire input sequence. They are **not** applicable to Language Modeling as the future tokens are not accessible.
- If you do have entire input sequence, bidirectionality is powerful for encoding (you should use it by default).
- For example, BERT (**Bidirectional** Encoder Representations from Transformers) is a powerful pretrained contextual representation system built on bidirectionality (we'll learn more about BERT later)

Source: Stanford 224n

Where we are

Models/Algorithms

- Linear models
- Feed-forward Neural Nets (FNN)
- Recurrent Neural Nets
- **Seq2Seq**

NLP tasks/applications

- Word meaning
- Language modelling
- Sequence tagging
- Sequence encoding
- **Machine Translation**

Where we are

Models/Algorithms

- Seq2Seq

NLP tasks/applications

- Machine Translation

Two major changes:

- Tasks involve not only understanding (encoding) but also generation (decoding)
- We will put two models together — one encoder and one decoder

Machine Translation

- Machine that can translate a text in one language to another
- Classic test of language understanding
    - Language analysis and generation

- German <=> English

Automatische Textverarbeitung gefällt mir.

I like natural language processing.

- Flagship task in NLP
- The Transformer was proposed for MT!
    - Now everywhere in ML/AI including Biology, Finance.

The Need for MT

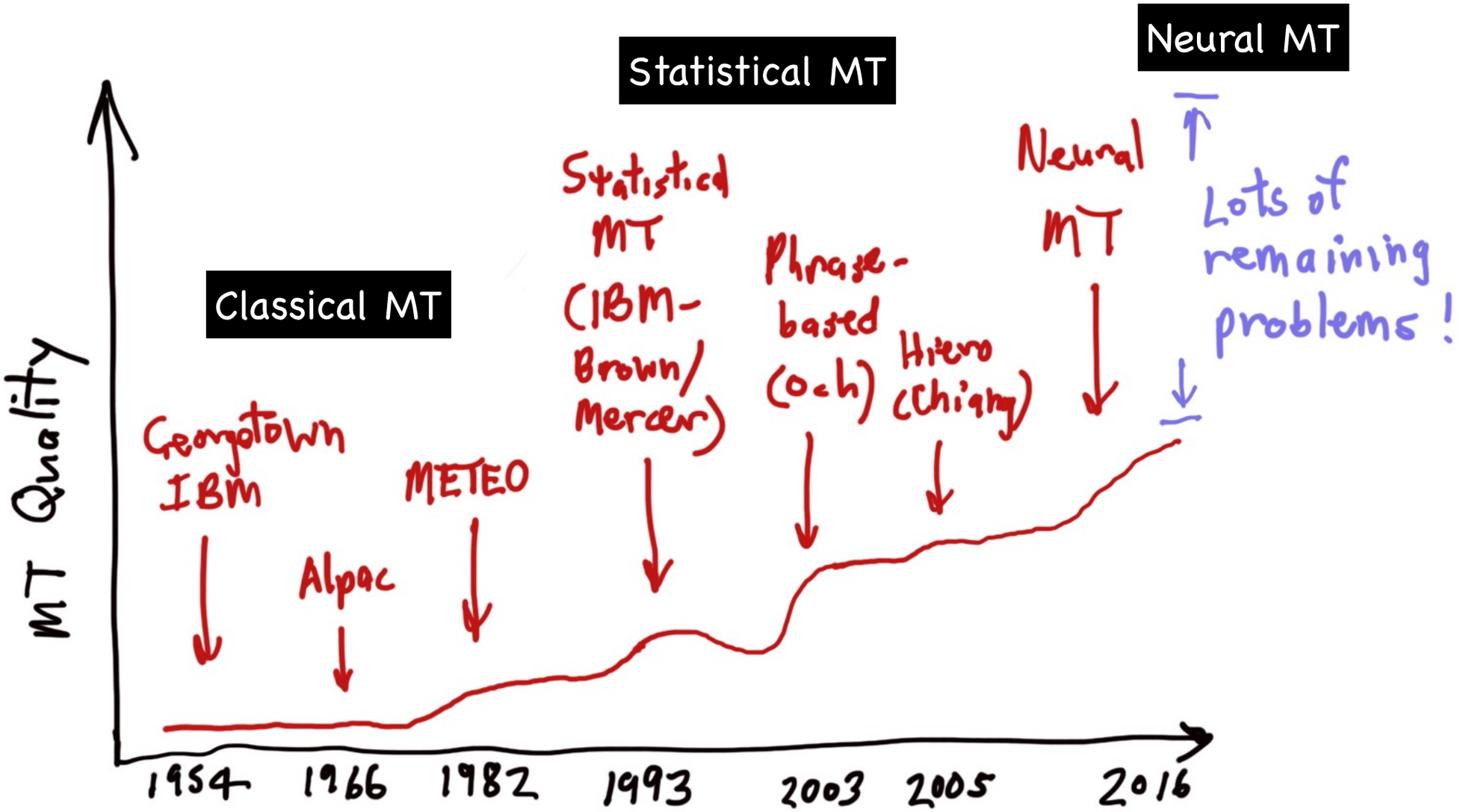
- Big MT needs: for humanity and for commerce
  - Translation is a US$90 billion a year industry
  - Huge in Europe and Asia
  - Large social/government/military as well as commercial needs
- Huge commercial use
  - Google translates over 100 billion words a day
  - eBay uses MT to enable cross-border trade
  - Facebook has their own homegrown MT

"When we turned [MT] off for some people, they went nuts!"

  - Microsoft, Amazon, Alibaba all are investing in MT

Source: Luong, Cho, Manning (ACL tutorial 2016)

Progress in MT



- Microsoft, Google, Yandex claimed **human parity** in MT in 2018 with NMT Only true in a controlled setup

Source: Luong, Cho, Manning (ACL tutorial 2016)

Neural Machine Translation (2014 -)

- New paradigm

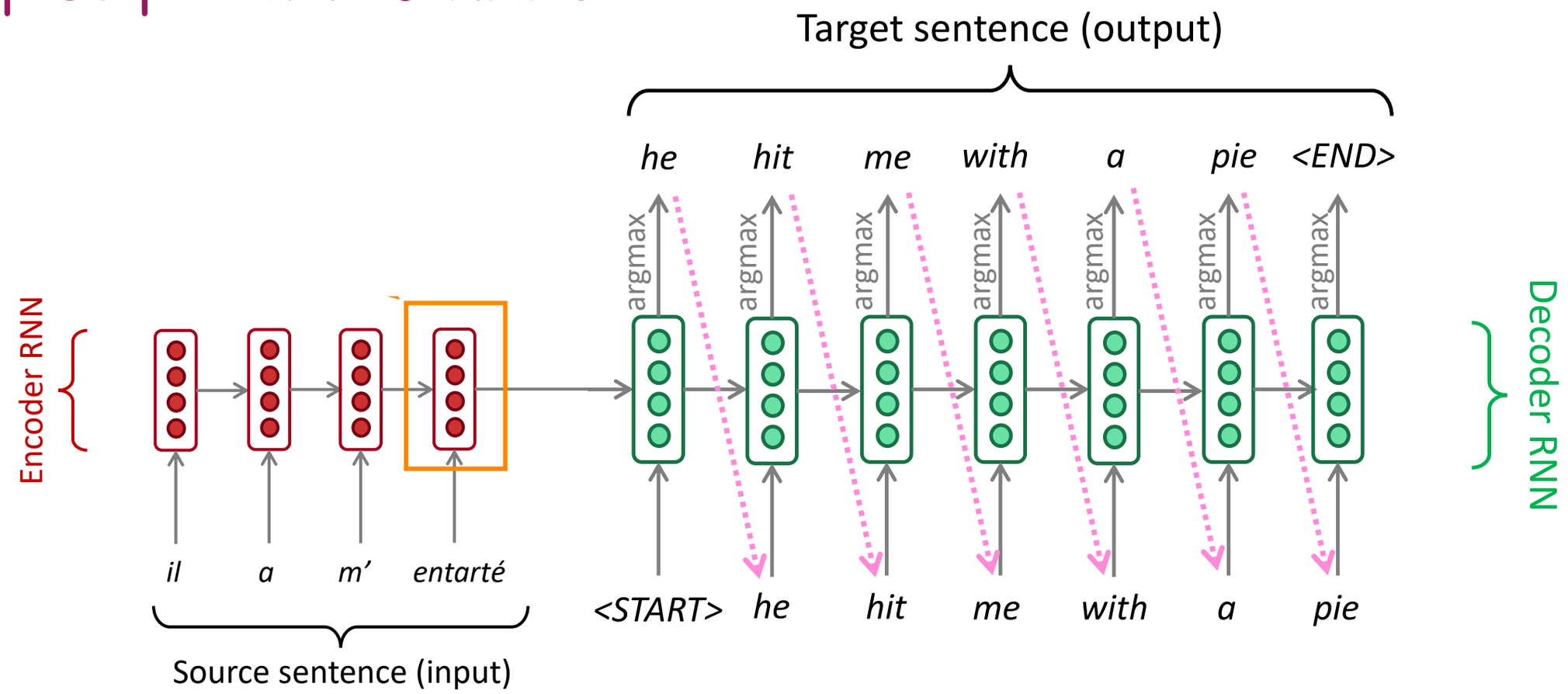
- One Network for everything!

Neural Machine Translation (2014 -)

- Machine Translation with a single neural network
- Two models (e.g., RNNs) are put together
  - One acts as an Encoder
  - The other acts as a Decoder
- The neural architecture is called sequence-to-sequence (seq2seq)
  - **Distributed representation** of words and phrases
  - **End-to-end** training
  - Allows **larger** context (even extra sentential)

Sequence-to-sequence Model

- Seq2Seq with Two RNNs

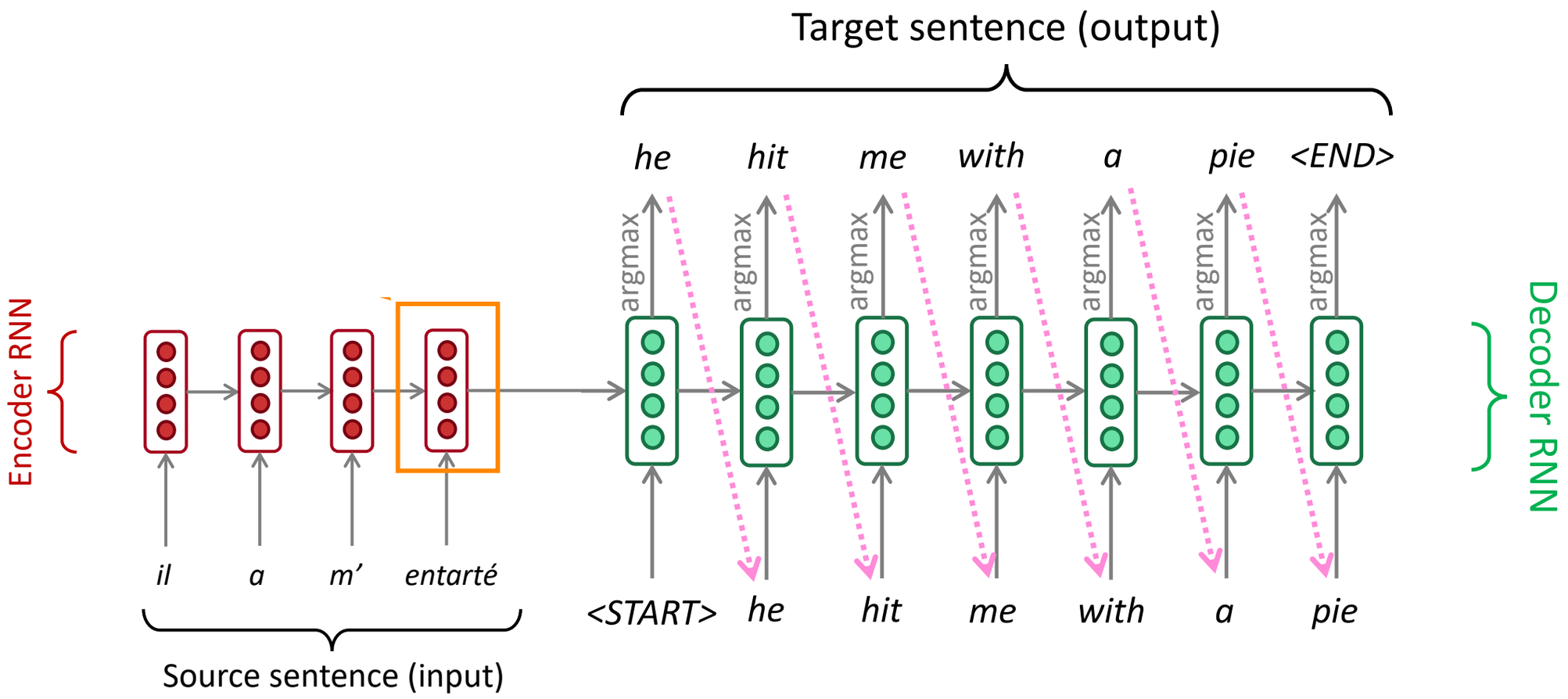


Encoder produces an encoding of the source sentence.

Provides Initial hidden state for Decoder

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

Sequence-to-sequence Model



- Directly models the conditional probability p(y|x) of translating a source sequence x1, …, xn to a target sequence y1, …, ym.

$$p(y_j|y_{<j}, \mathbf{s}) = softmax(g(h_j))$$

Training a Seq2Seq Model

$$J = \frac{1}{T}\sum_{t=1}^{T} J_t \quad = \quad J_1 + J_2 + J_3 + J_4 + J_5 + J_6 + J_7$$

= negative log prob of "he"

= negative log prob of "with"

= negative log prob of <END>

$\hat{y}_1$ $\hat{y}_2$ $\hat{y}_3$ $\hat{y}_4$ $\hat{y}_5$ $\hat{y}_6$ $\hat{y}_7$

Encoder RNN

Decoder RNN

il *a* *m'* *entarté* *<START>* *he* *hit* *me* *with* *a* *pie*

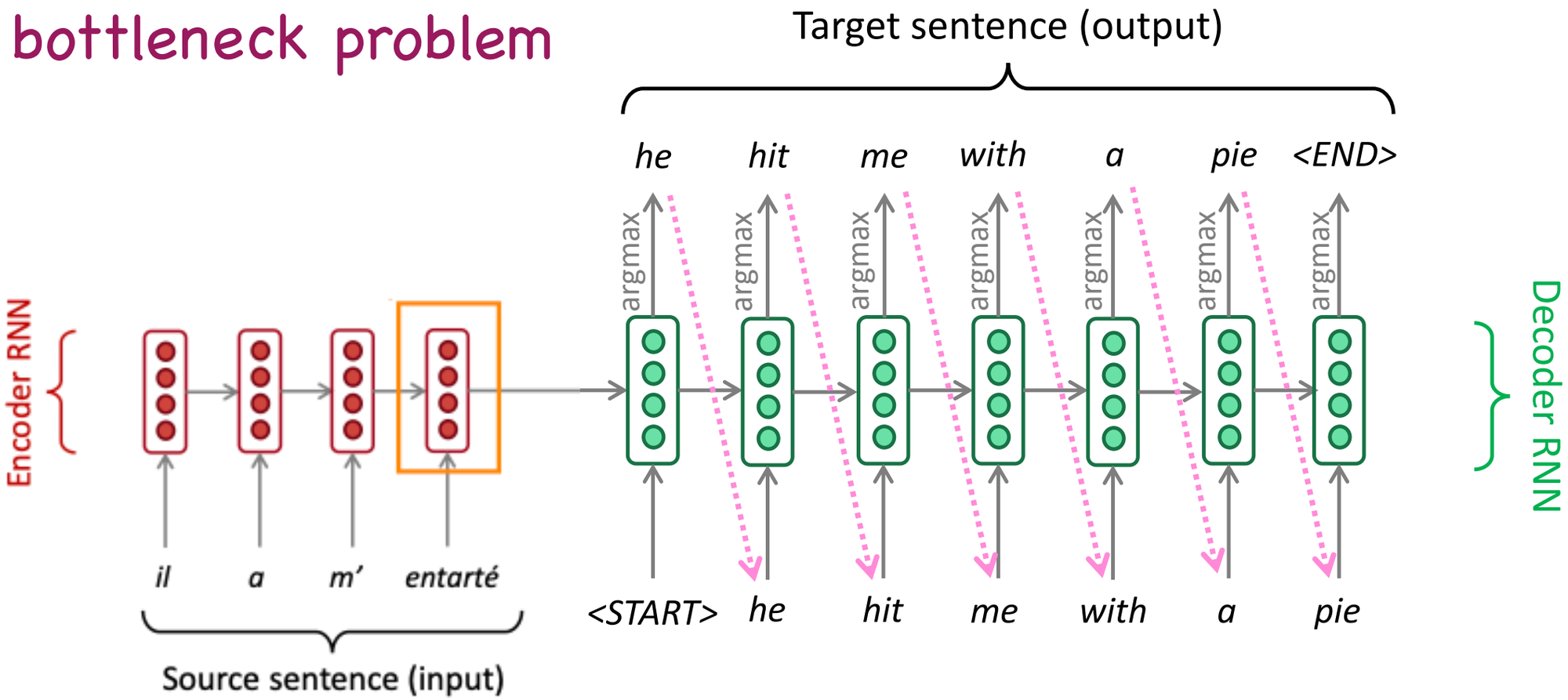
Source sentence (from corpus)

Target sentence (from corpus)

Source: Stanford 224n

Sequence-to-sequence Model

- The bottleneck problem

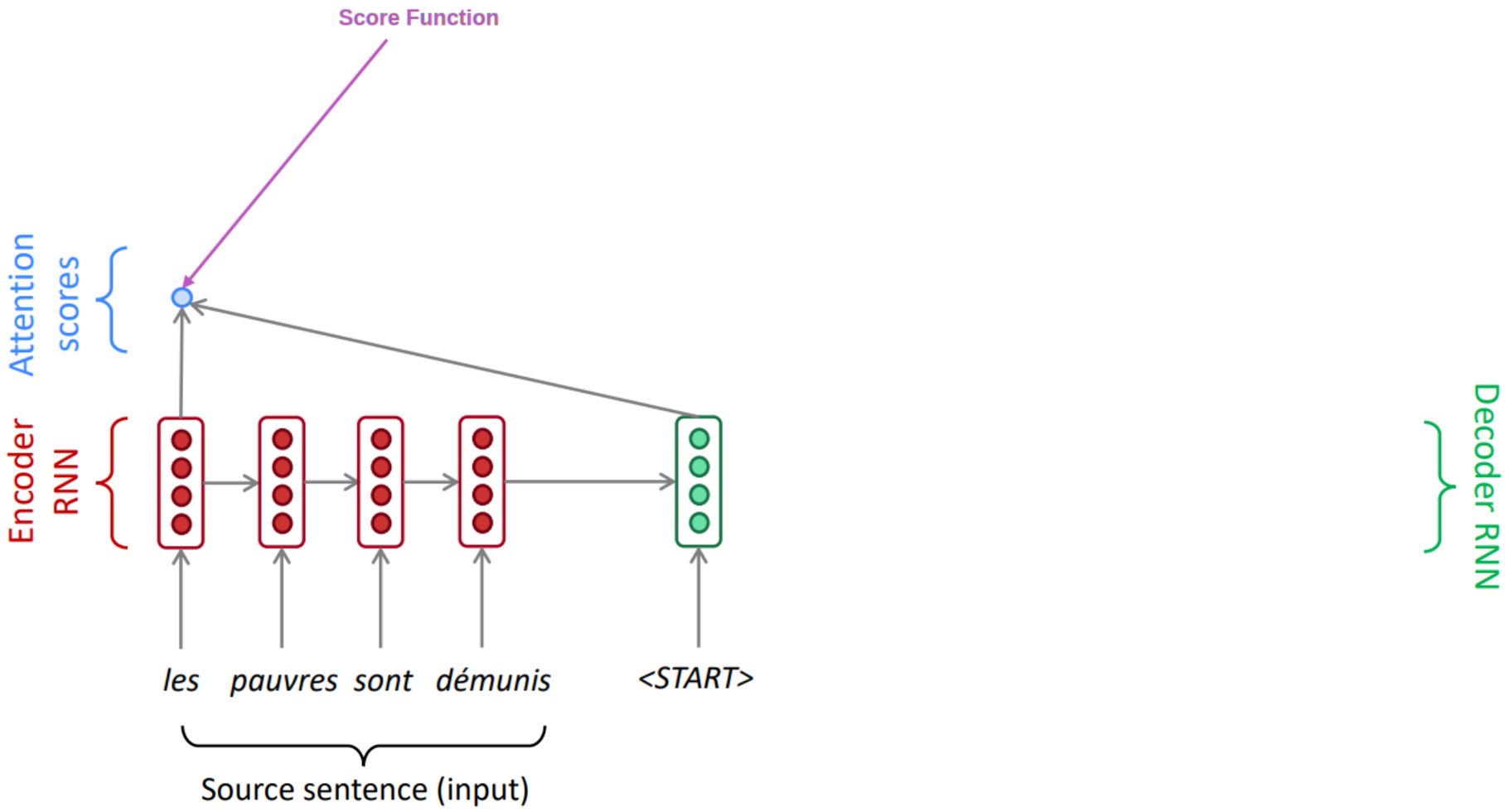


Encoding of the source sentence. This needs to capture all information about the source sentence. Information bottleneck!

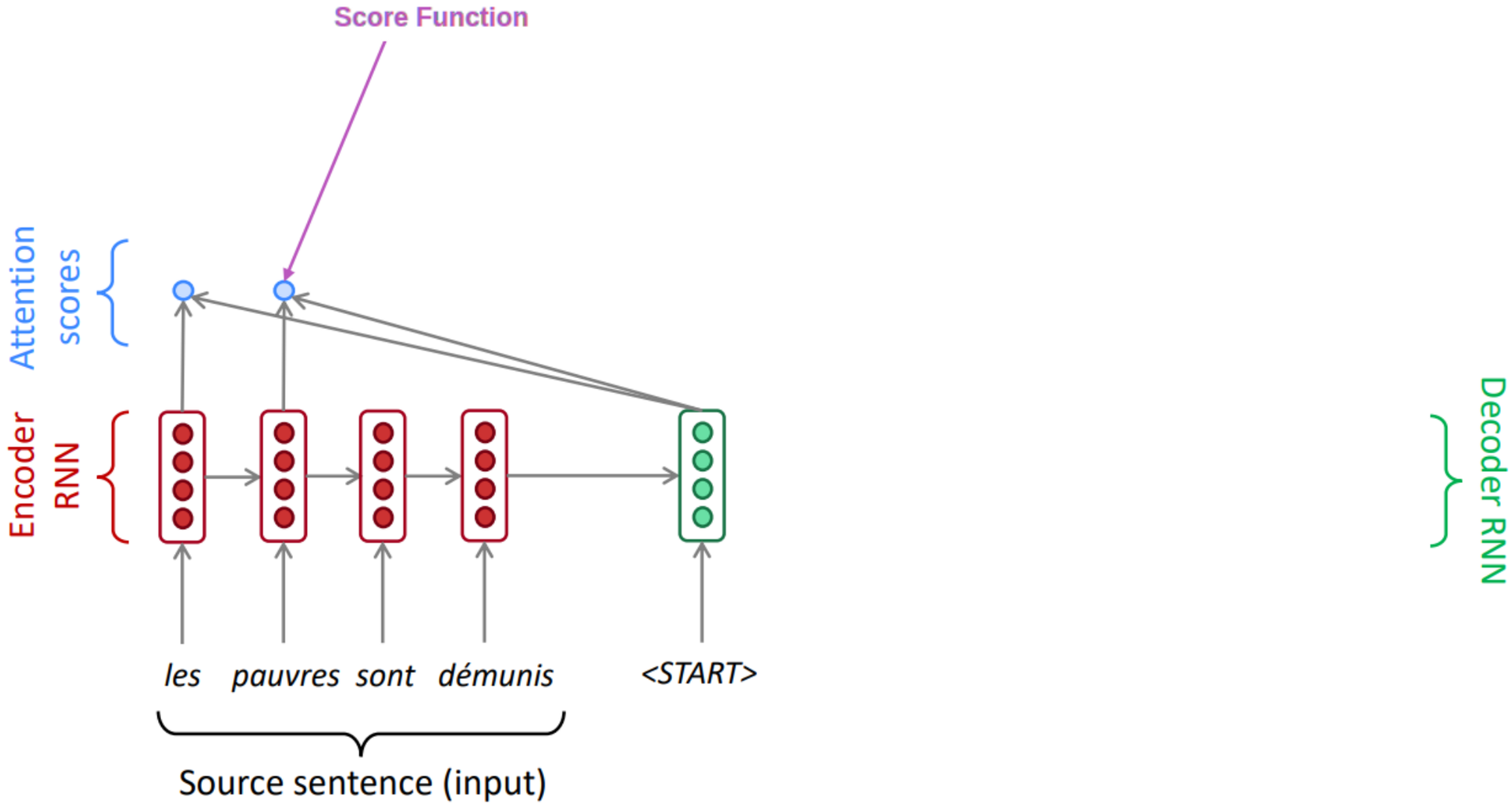
Attention Mechanism

- **Attention** provides a solution to the bottleneck problem.
- **Core idea:** on each step of the decoder, use direct connection to the encoder to focus on the relevant part of the source sequence

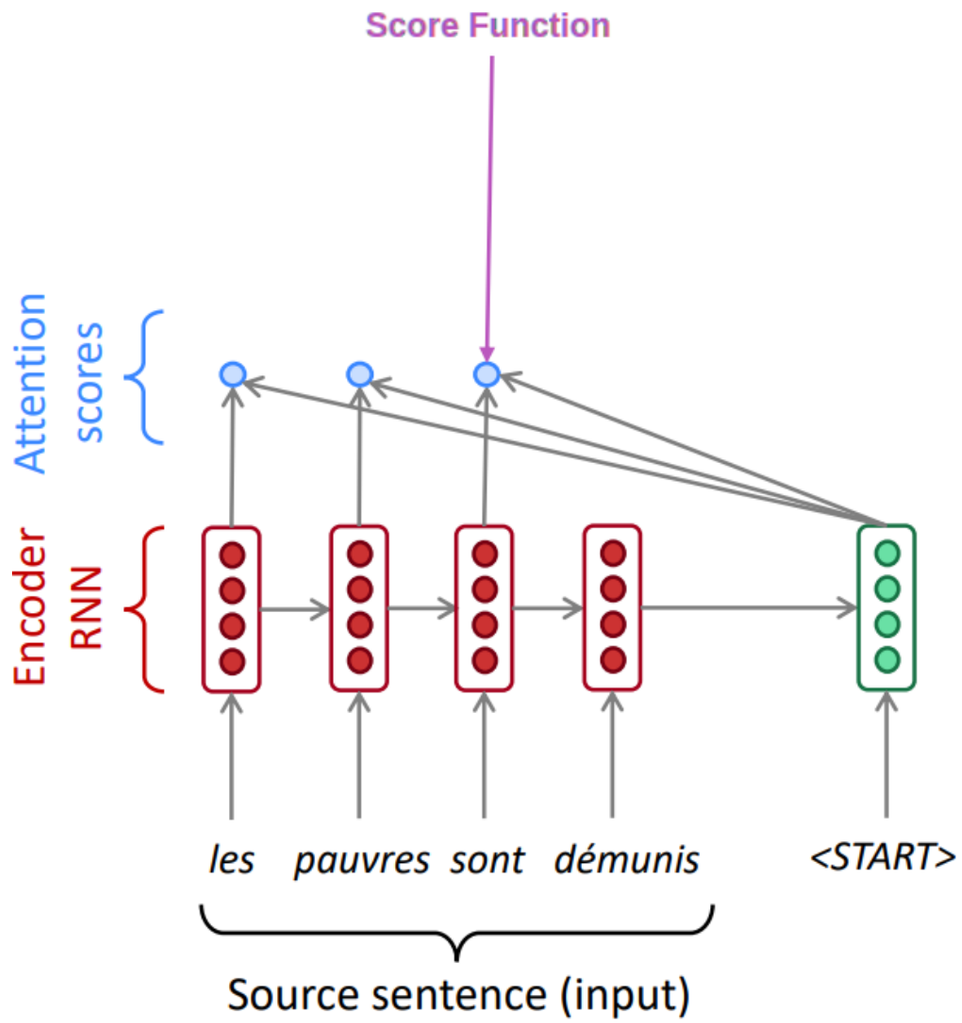
Attention Mechanism



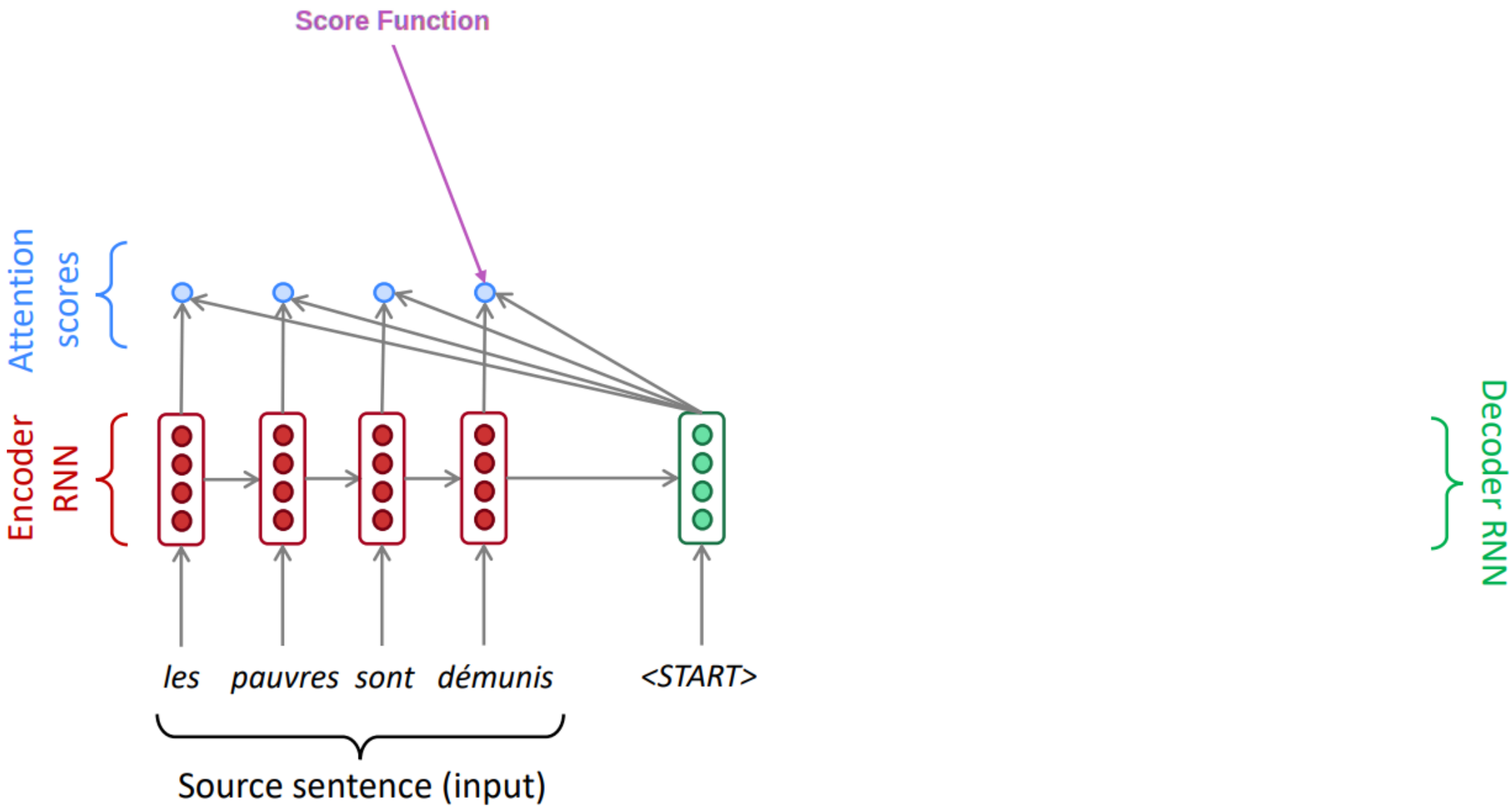
Attention Mechanism



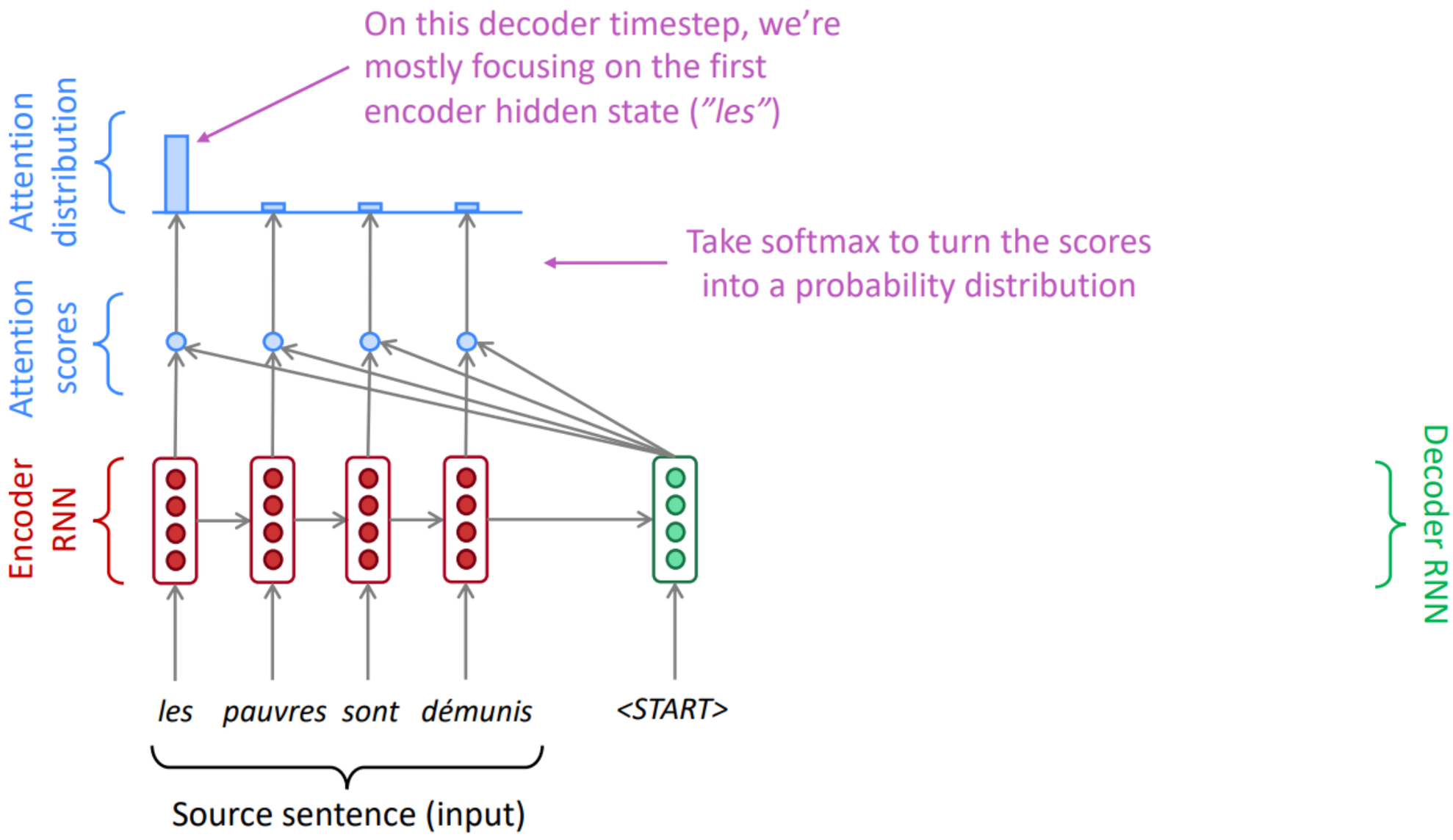
Attention Mechanism



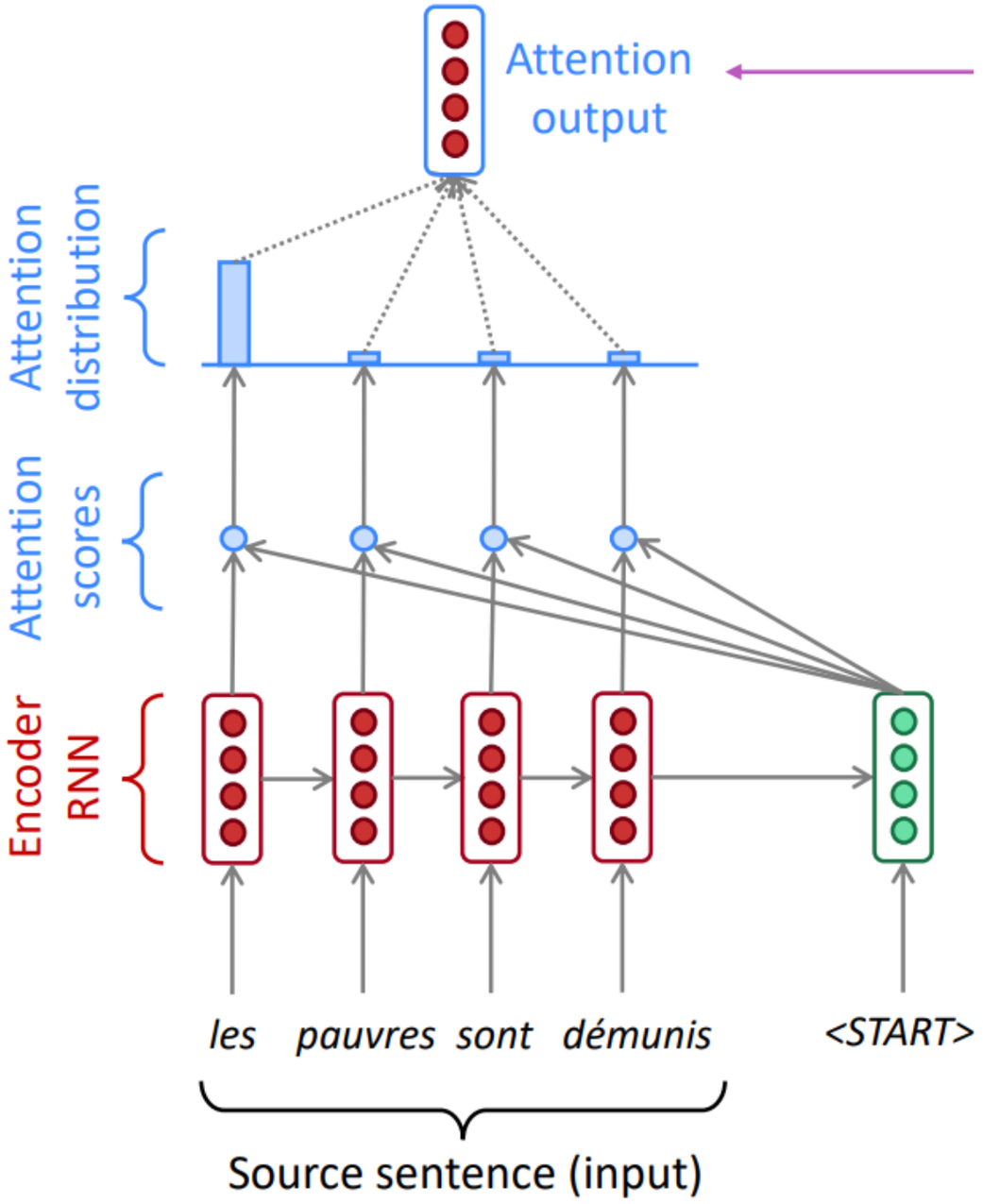
Attention Mechanism



Attention Mechanism



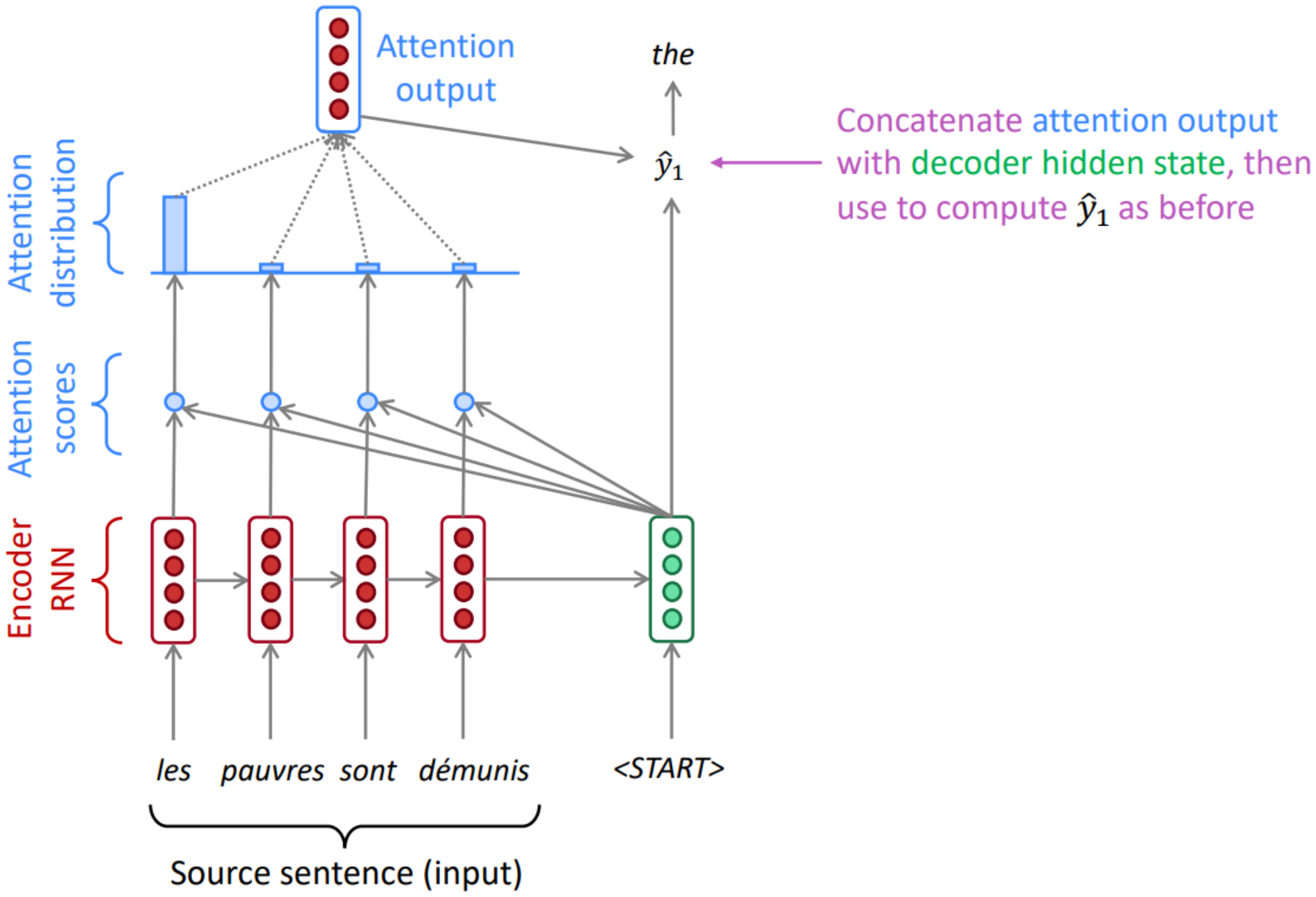
Attention Mechanism



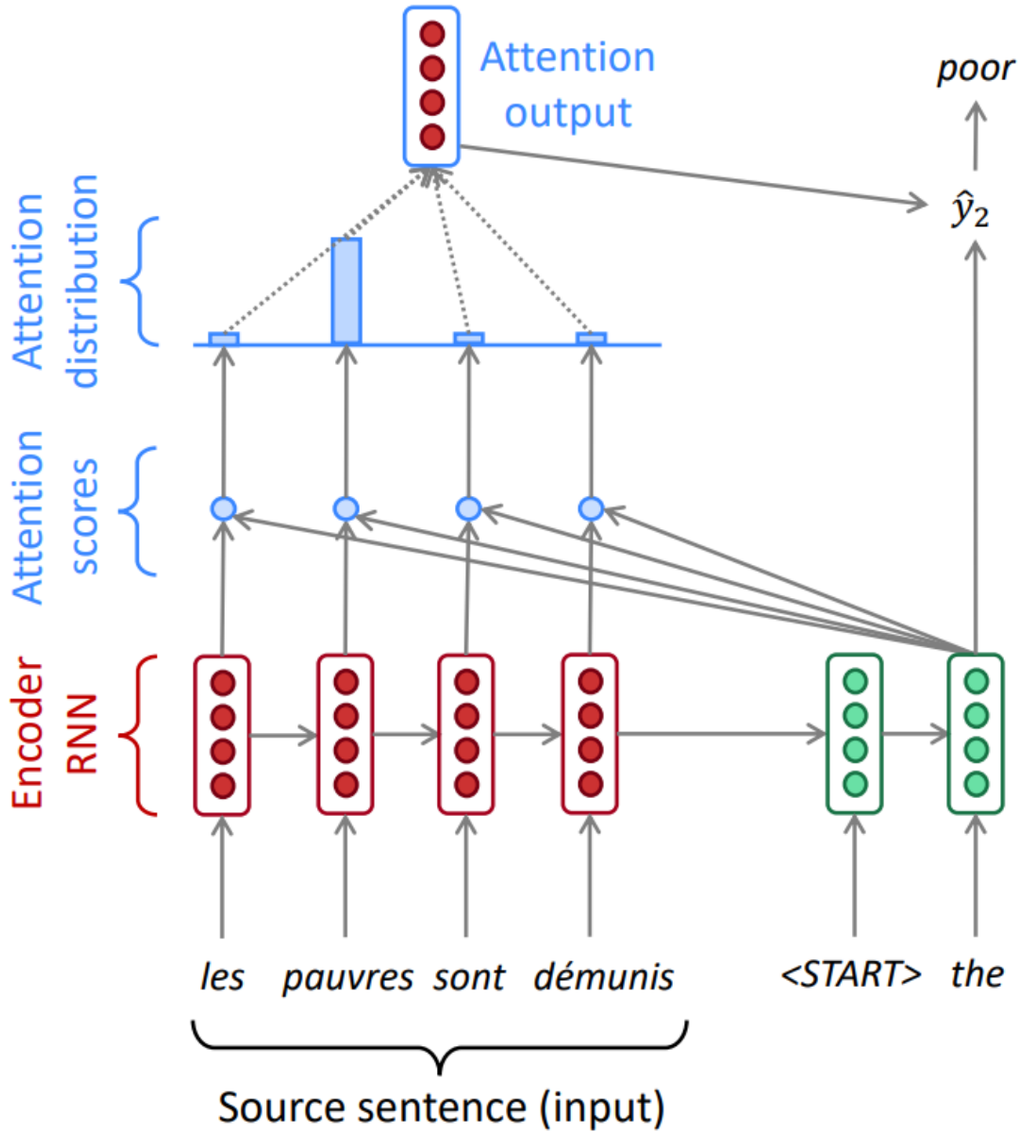
Use the attention distribution to take a **weighted sum** of the encoder hidden states.

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

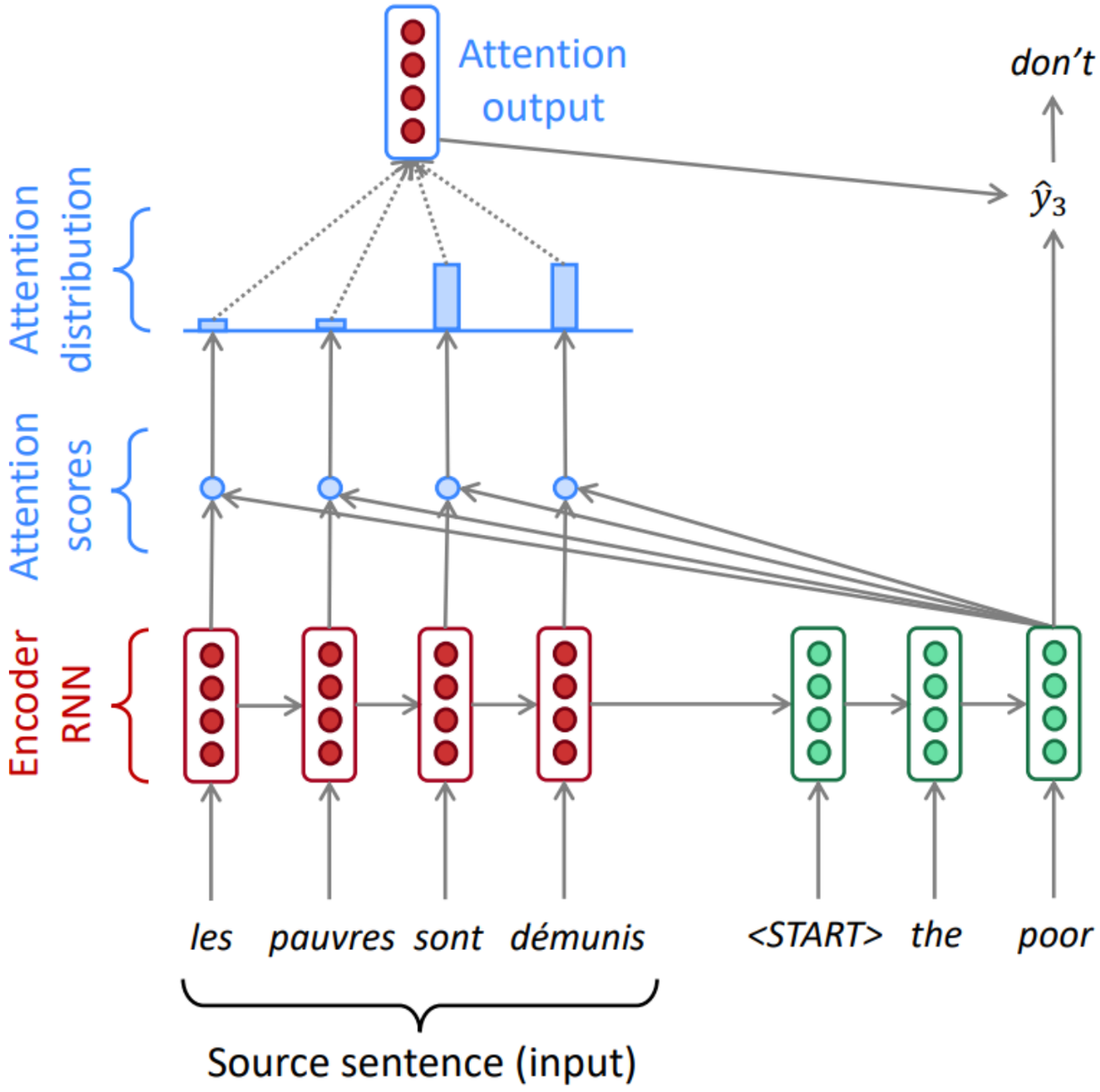
Attention Mechanism



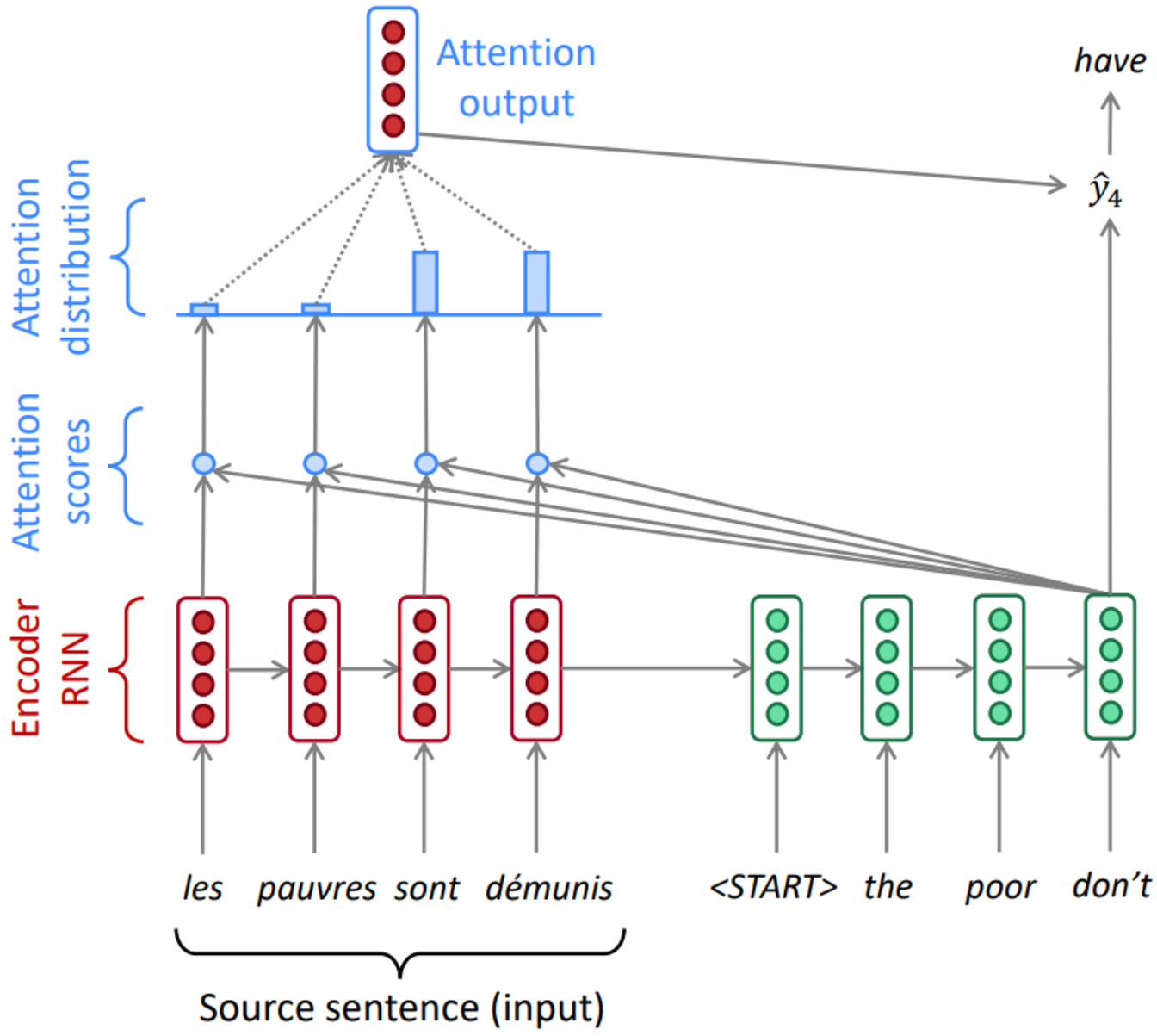
Attention Mechanism



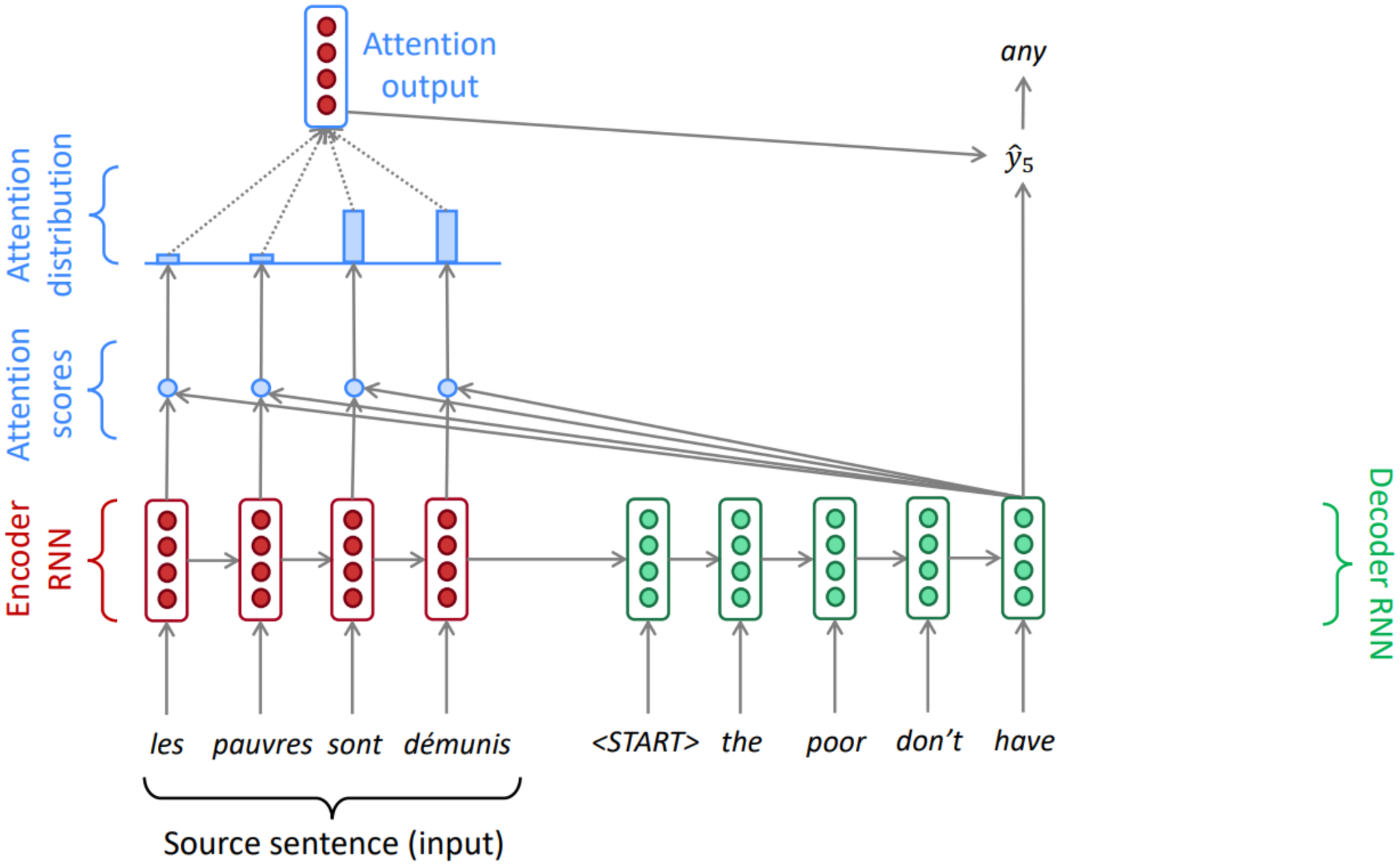
Attention Mechanism



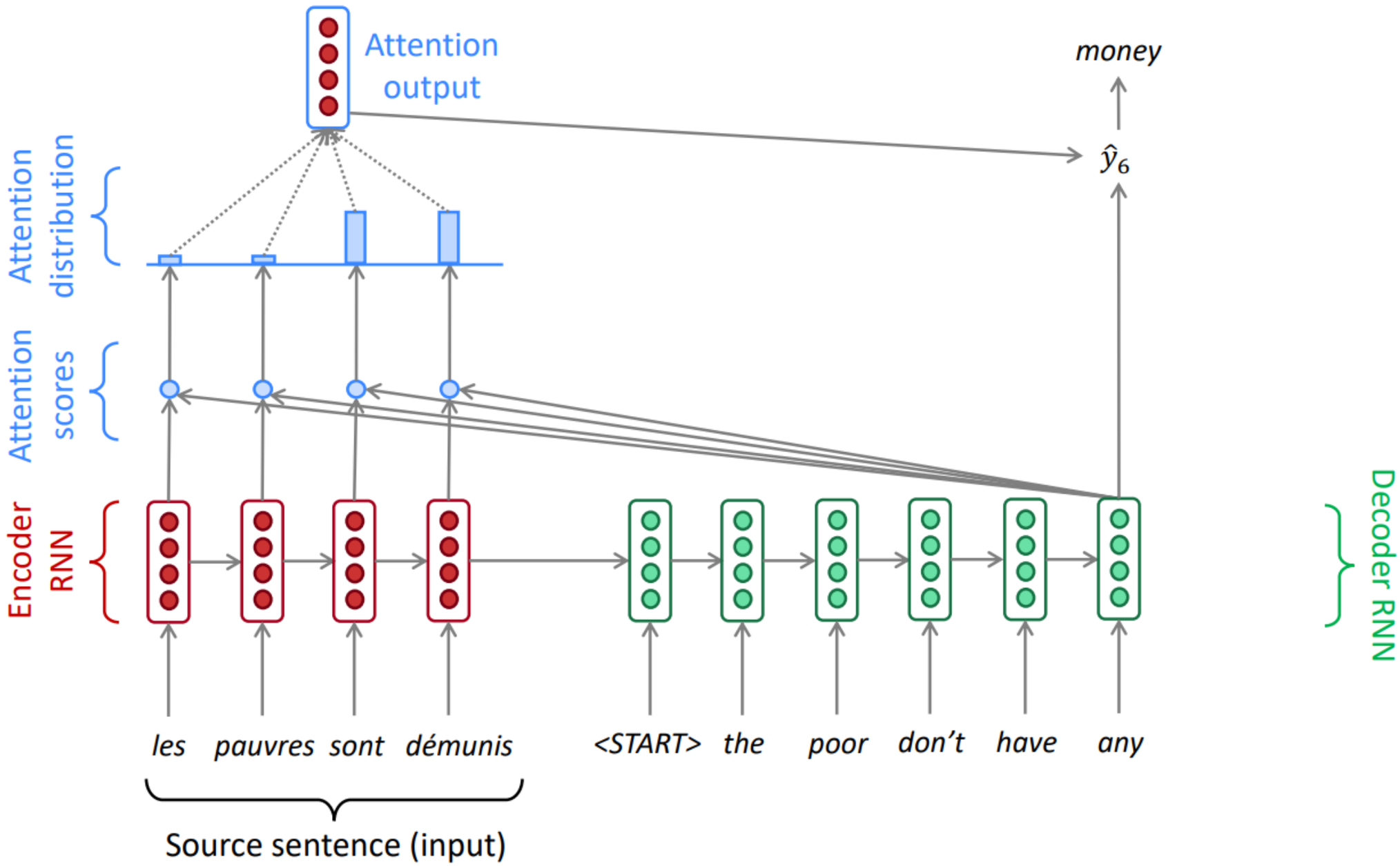
Attention Mechanism



Attention Mechanism



Attention Mechanism



Attention Mechanism (Formally)

- 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 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)

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

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

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

Attention is a general Deep Learning technique

- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$ **Keys/Values**
- On timestep *t*, we have decoder hidden state $s_t \in \mathbb{R}^h$ **Query**

- Given a set of key-value vectors, and a query vector, **attention** is a technique to compute a weighted sum of the value vectors, dependent on the query-key relevance.

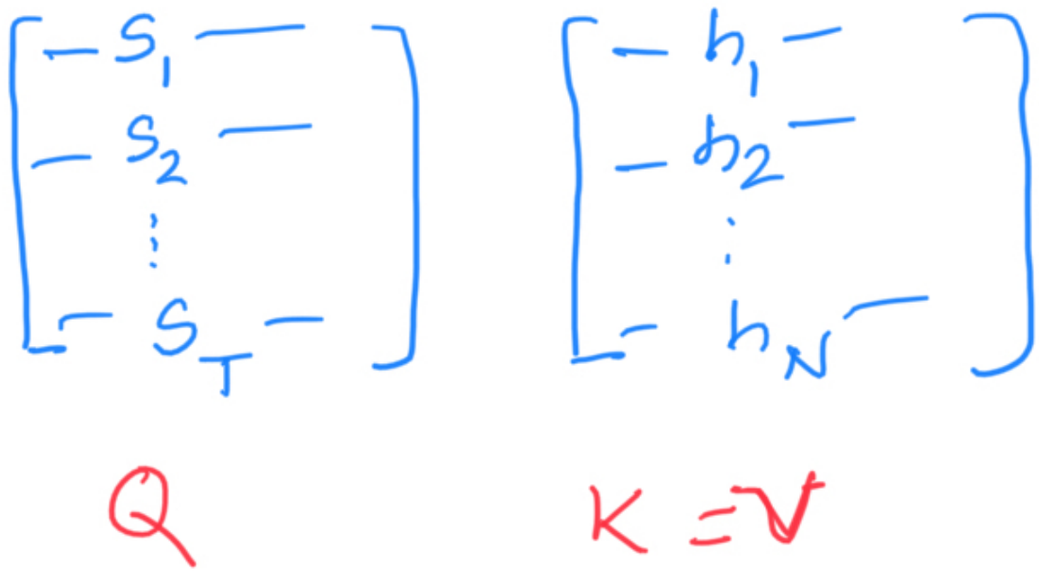
Intuition:

- The weighted sum is a selective summary of the information contained in the values, where the query-key determines which values to focus on.
- Attention is a way to obtain a fixed-size representation of an arbitrary set of representations (values), dependent on some other representation (query-key).

Attention in Matrix Notation

- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep *t*, we have decoder hidden state $s_t \in \mathbb{R}^h$

- Dot-product attention in matrix notation



$$A = S(QK^T)V$$

Plan

- NLP & why deep learning for NLP [10 min]
- FFNs & word vectors [15 min]
- RNNs & sequence level NLP tasks [20 min]
- Break [30 min]
- Transformers [13 min]
- Self-supervised learning [12 min]

Where we are

Models/Algorithms

- Linear models
- Feed-forward Neural Nets (FNN)
- Recurrent Neural Nets
- **Transformers**

NLP tasks/applications

- Word meaning
- **Language modelling**
- **Sequence tagging**
- **Sequence encoding**
- **Machine Translation**

Recurrent Neural Networks

Issues with RNN Computations

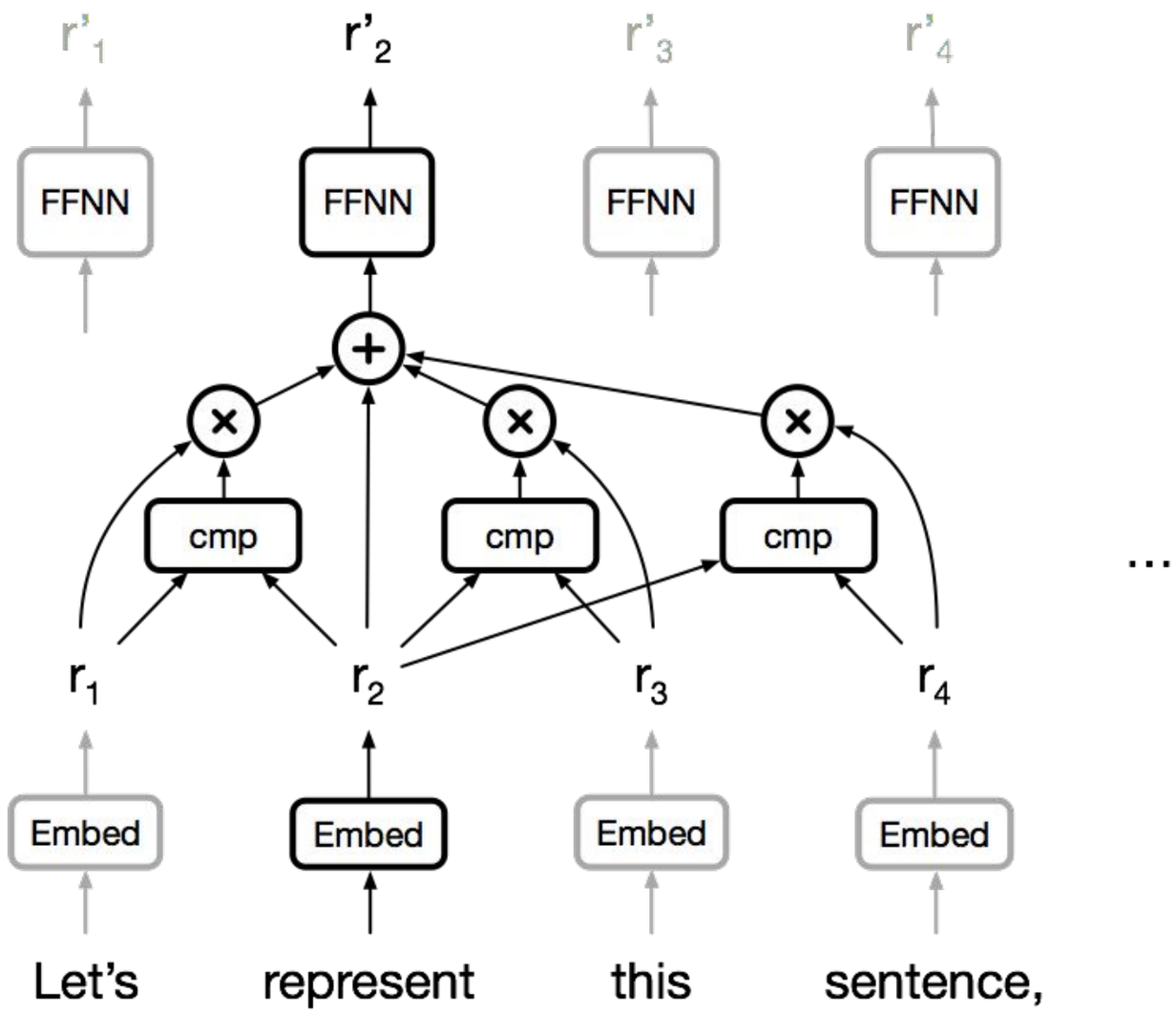
- Sequential computation inhibits parallelization.
- No explicit modeling of long and short range dependencies
- Not many hierarchy of features.
- RNNs are a bit wasteful!

Attentions?

- Attention between encoder and decoder is crucial in NMT

Why not use attention for representations?

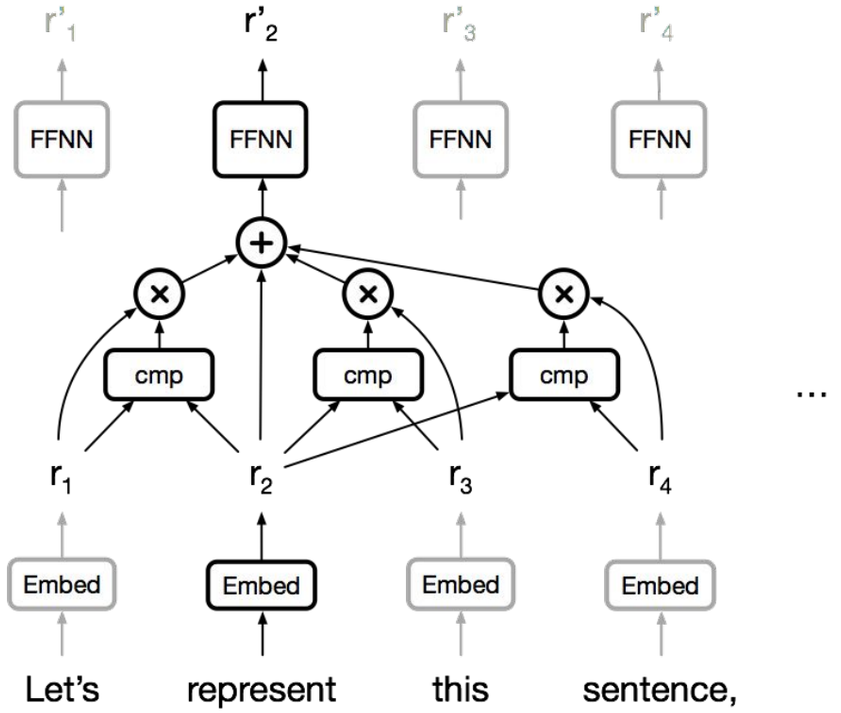
Self Attentions



source: Vaswani et al.

Self Attentions

- Constant 'path length' between any two positions.
- (Soft) Gating & multiplicative interactions.
- Easy to parallelize (per layer).
- Can replace sequential computation entirely

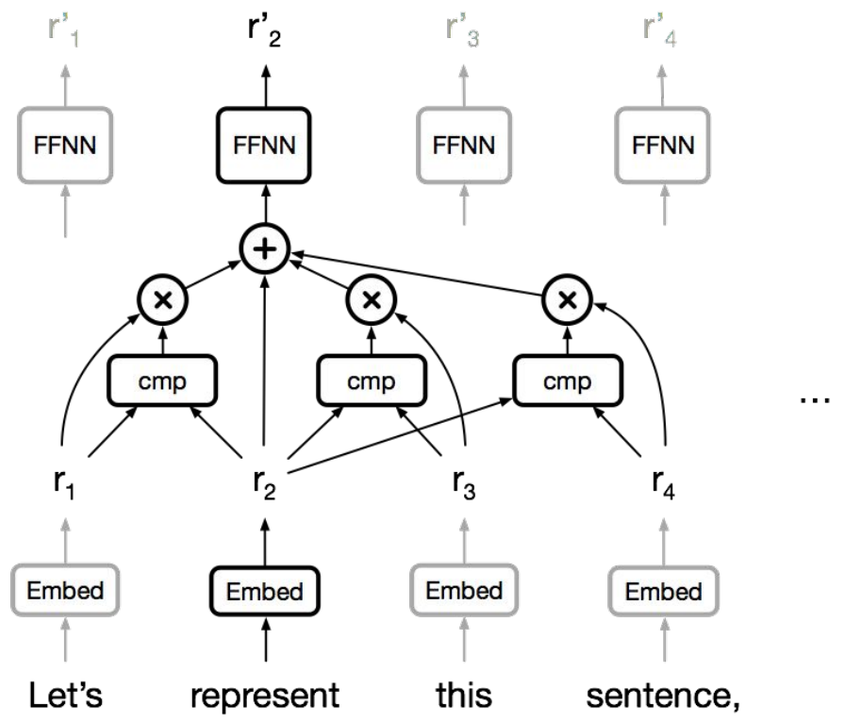


source: Vaswani et al.

Attention is Cheap

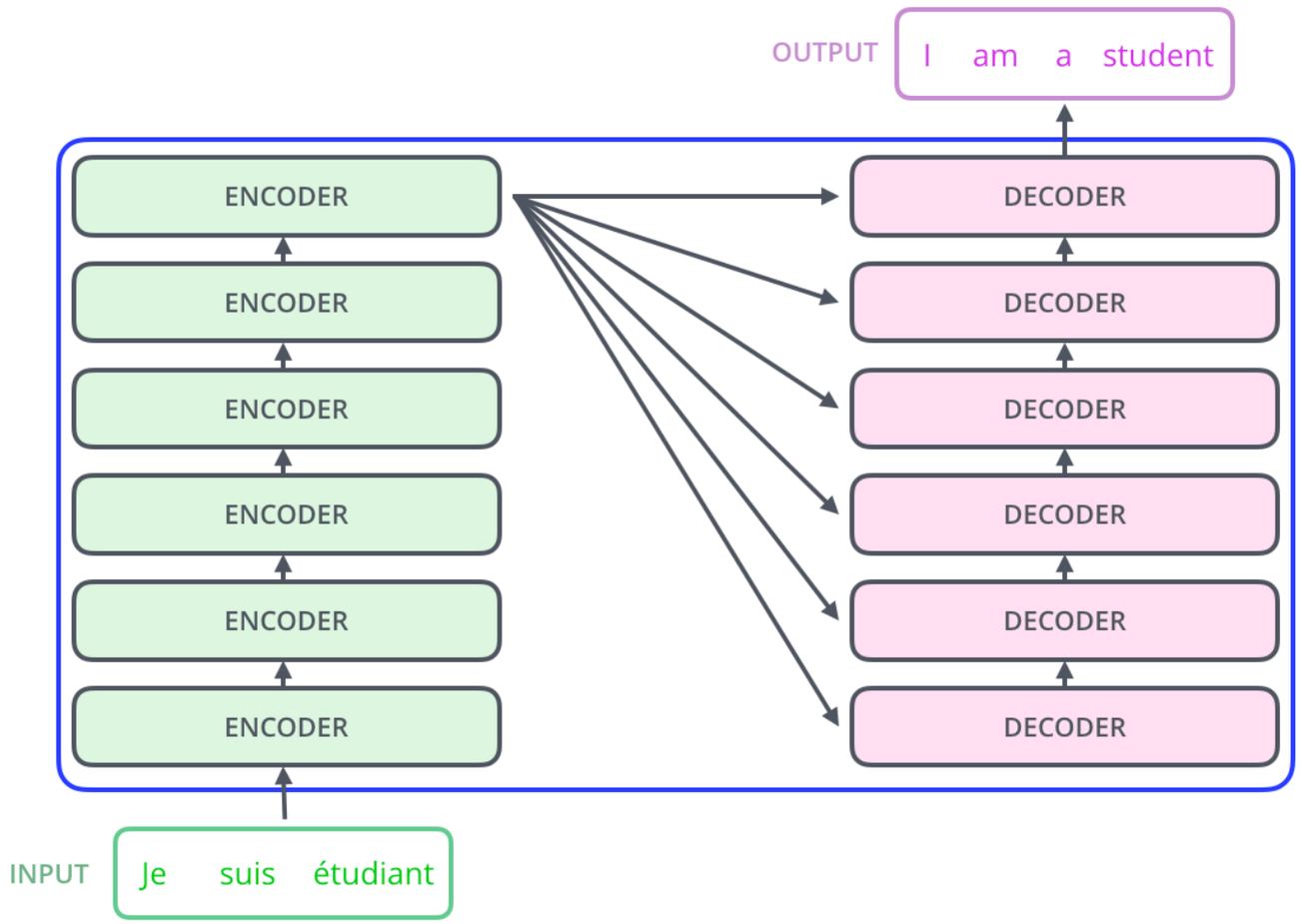
FLOPs

Self-Attention	O(length^2 · dim)
RNN (LSTM)	O(length · dim^2)
Convolution	O(length · dim^2 · kernel_width)



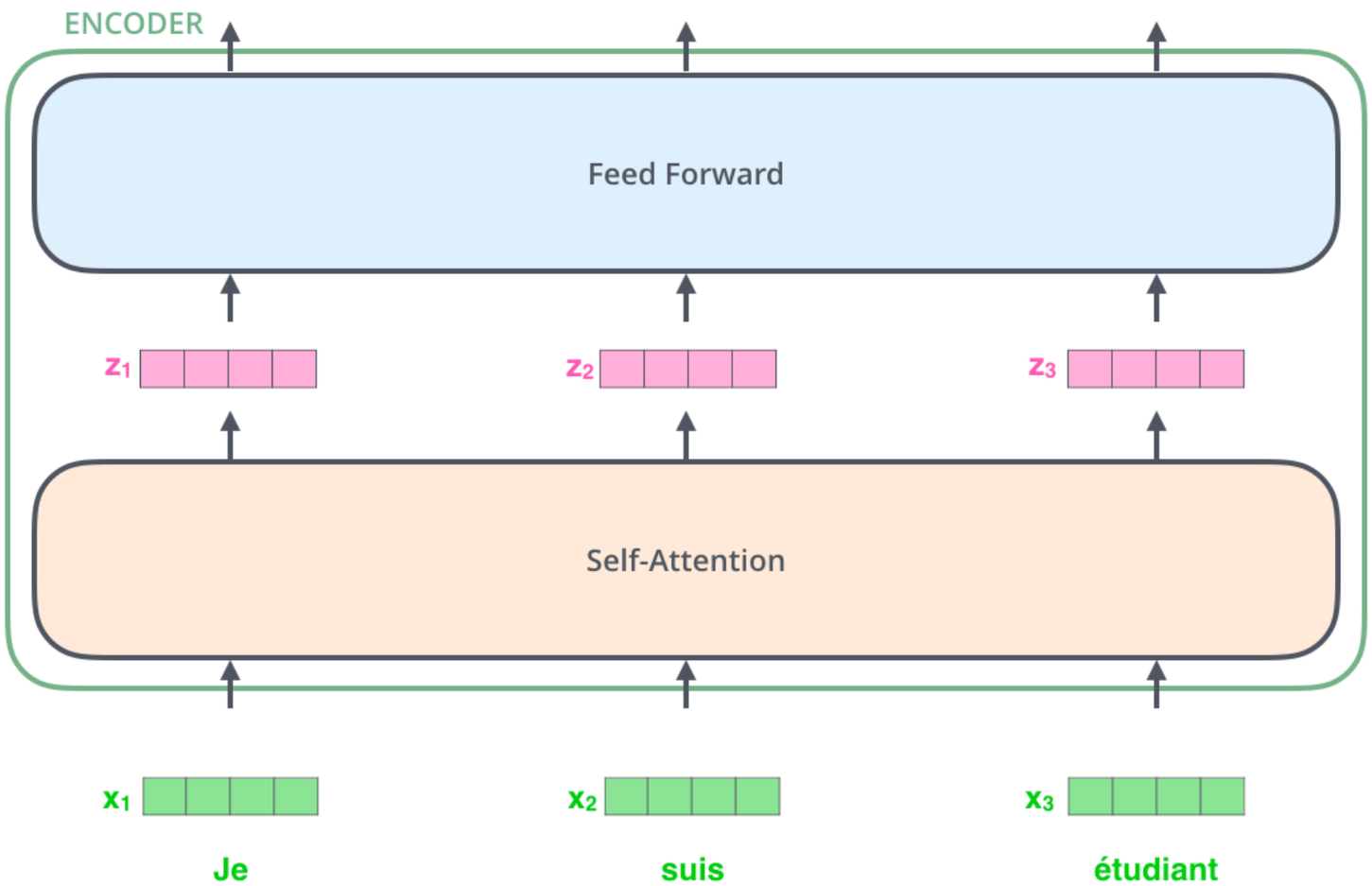
source: Vaswani et al.

Encoder-Decoder Network



source: https://jalammar.github.io/

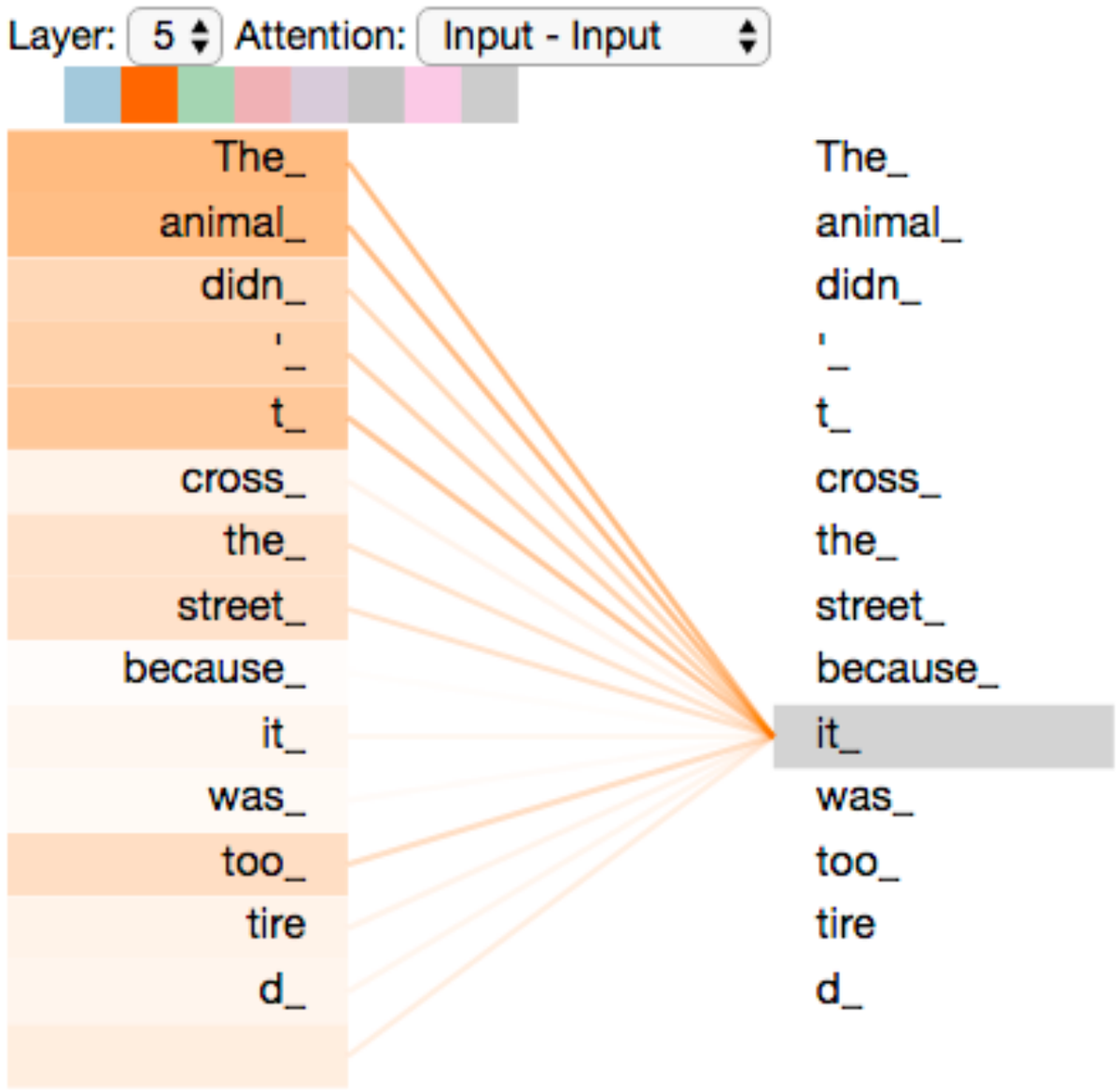
The First Encoder Layer



- The dependencies between word vectors are modelled by the self-attention layer.

source: https://jalammar.github.io/

One Self-Attention Layer

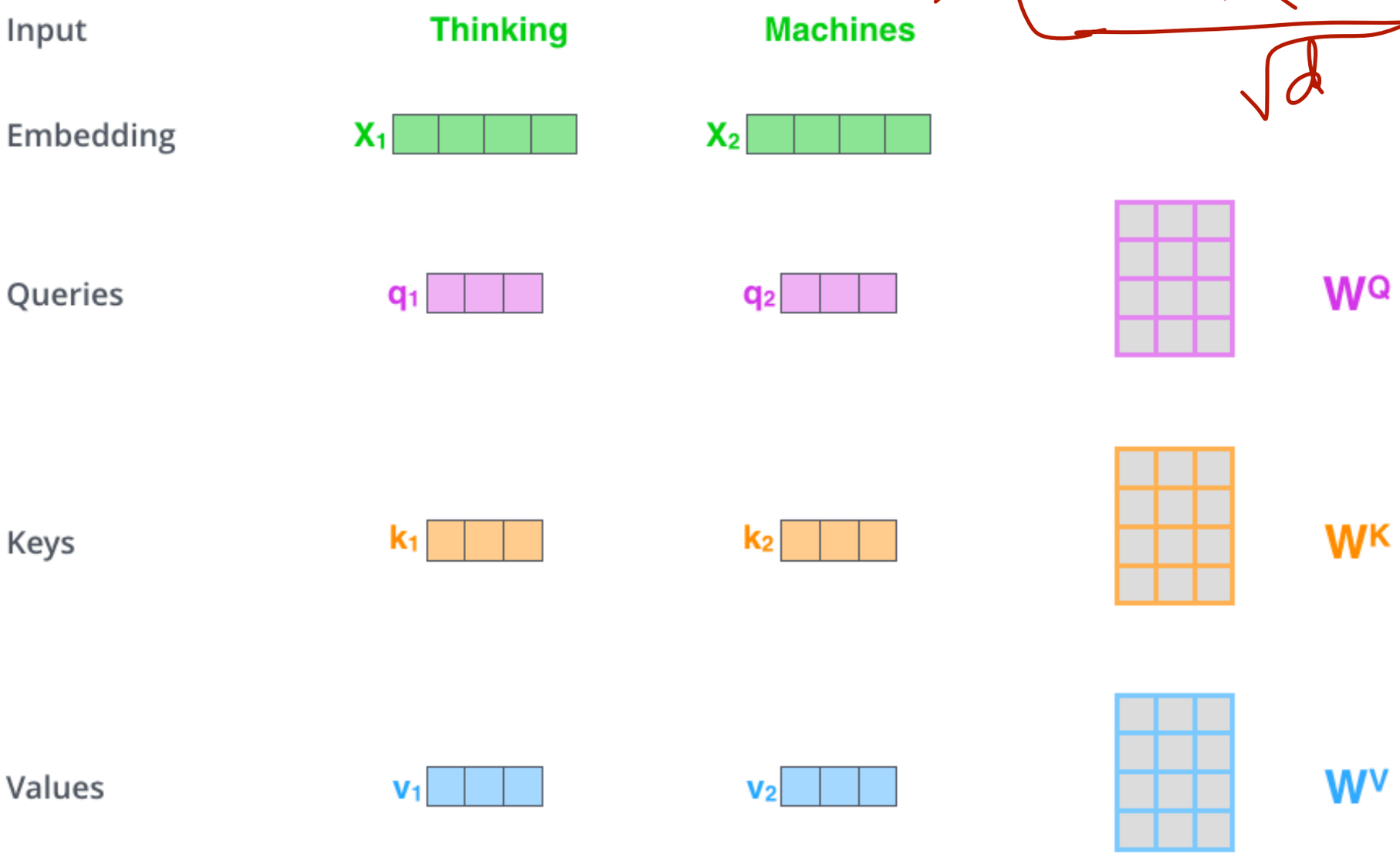


- The dependencies between word vectors are modelled by the self-attention layer.

source: https://jalammar.github.io/

One Self-Attention Layer

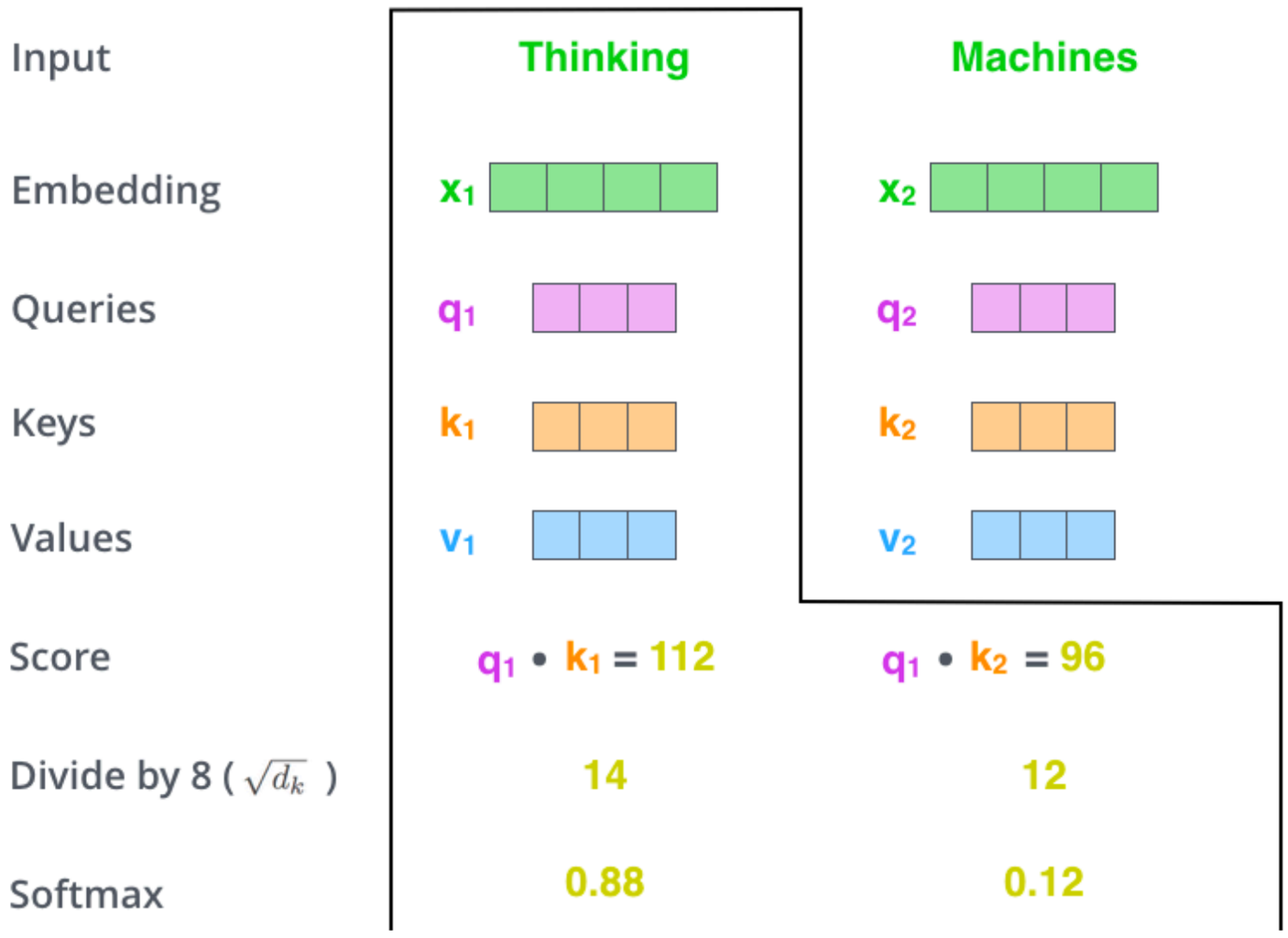
$$\mathrm{Soft}\left[\frac{(QW^Q)(KW^K)^T}{\sqrt{d}}\right](VW^V)$$



- The dependencies between word vectors are modelled by the self-attention layer.

source: https://jalammar.github.io/

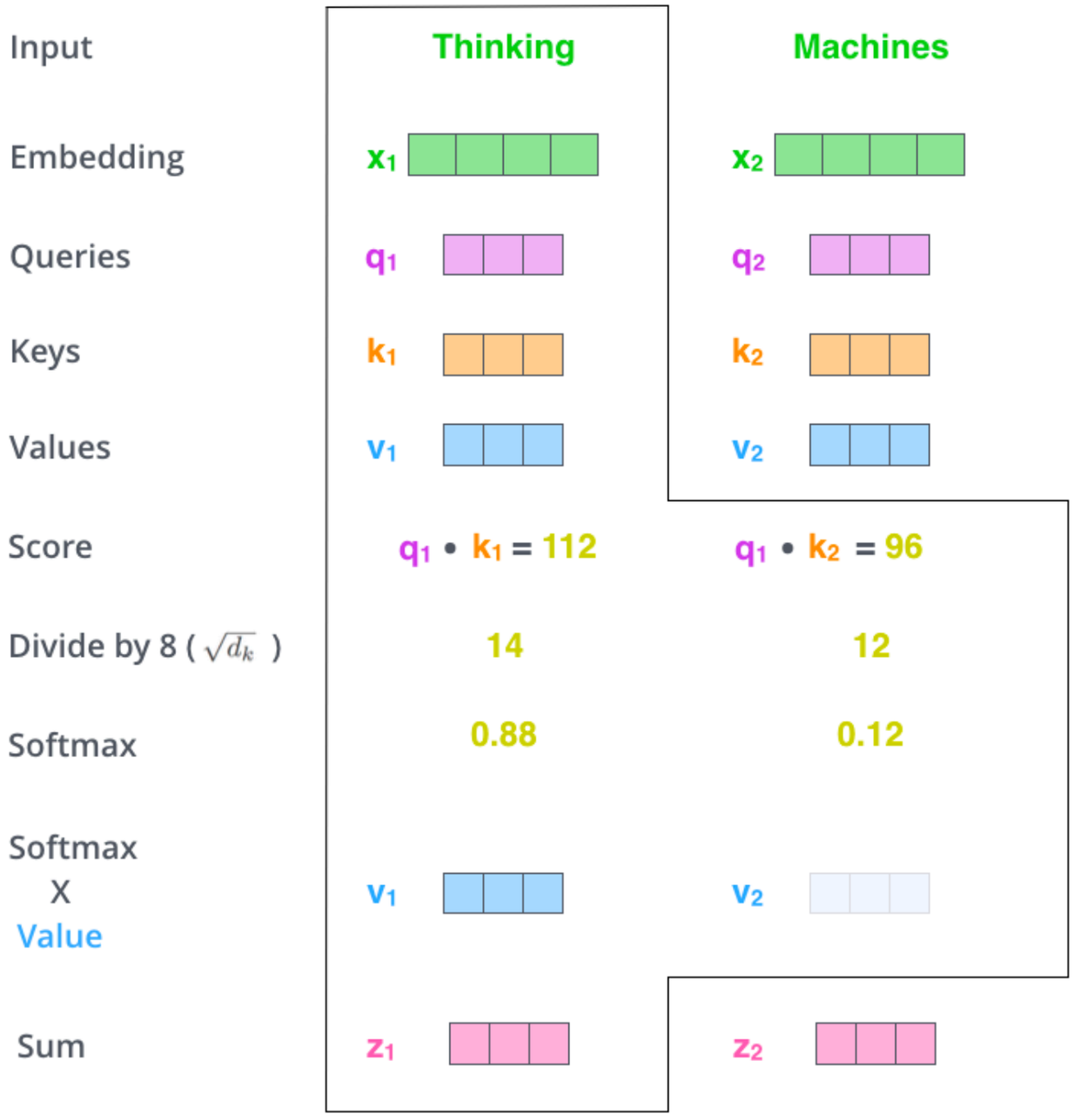
One Self-Attention Layer



- The dependencies between word vectors are modelled by the self-attention layer.

source: https://jalammar.github.io/

One Self-Attention Layer



- The dependencies between word vectors are modelled by the self-attention layer.

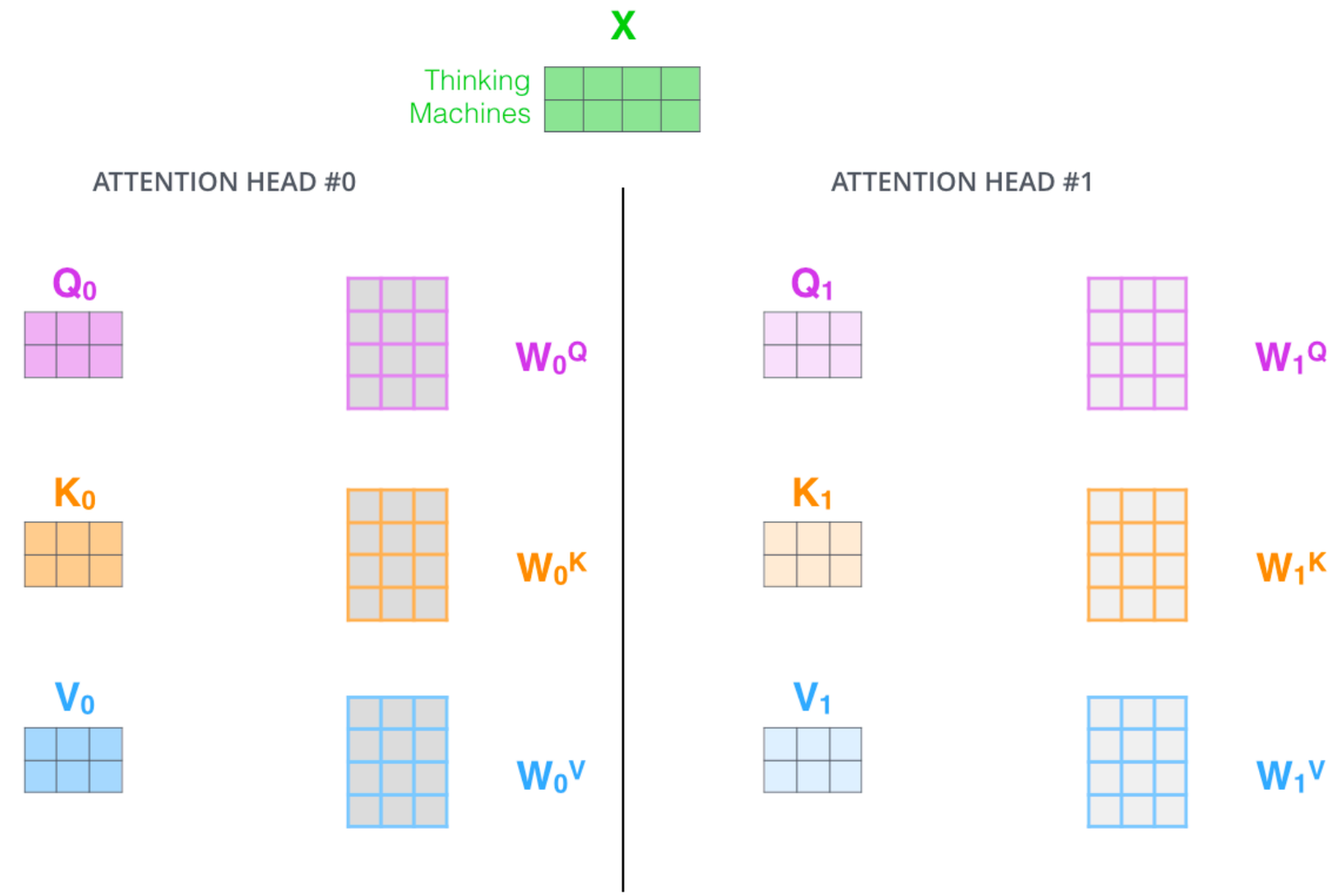
source: https://jalammar.github.io/

One Self-Attention Layer

- The dependencies between word vectors are modelled by the self-attention layer.

source: https://jalammar.github.io/

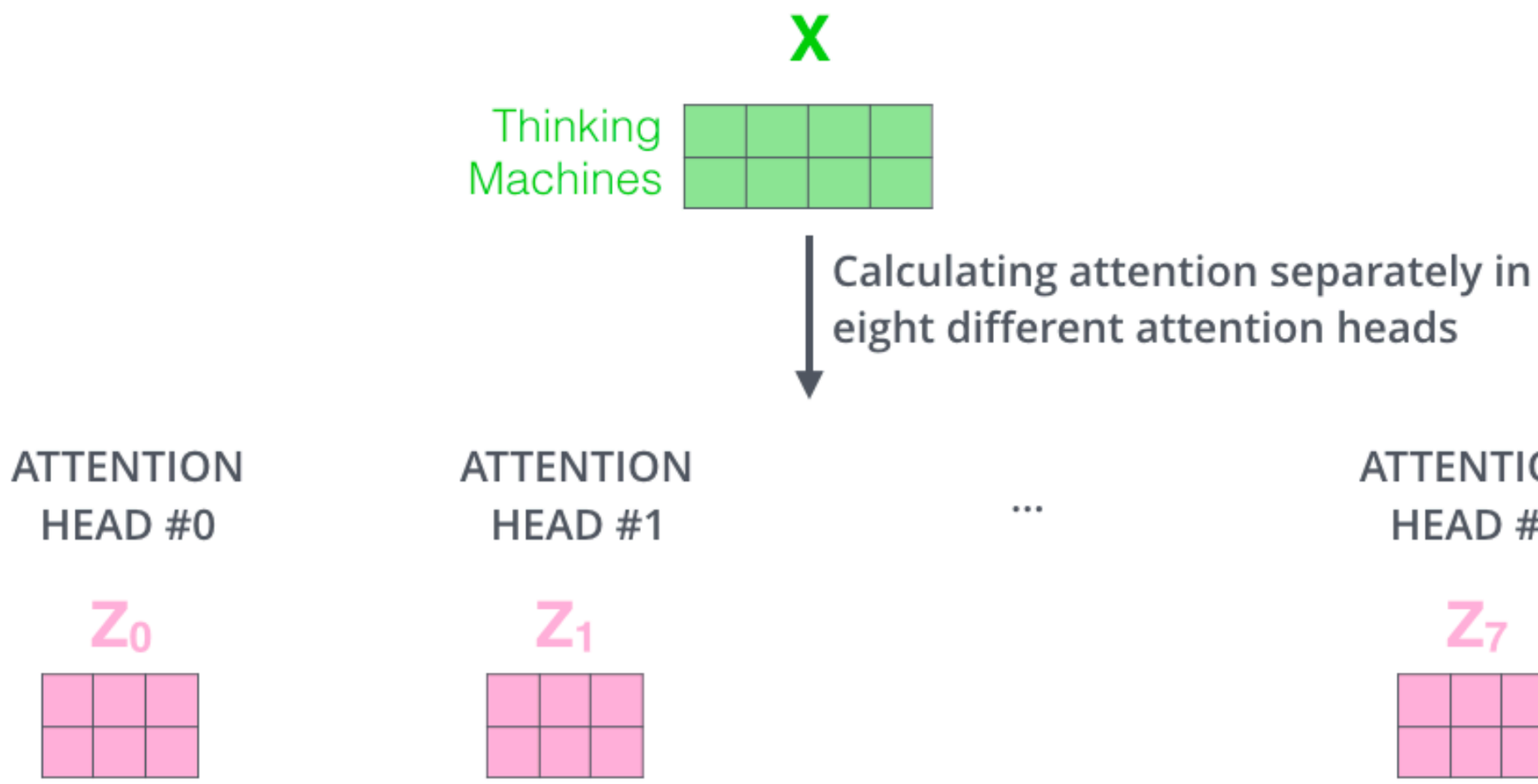
Using Multiple Heads



- Multiple heads expand the model's ability to focus on different positions.
- Each head gives the attention layer multiple representation subspaces

source: https://jalammar.github.io/

Using Multiple Heads



- Transformer uses 8 different heads

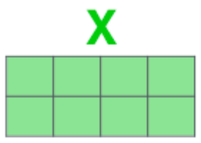
source: https://jalammar.github.io/

Using Multiple Heads

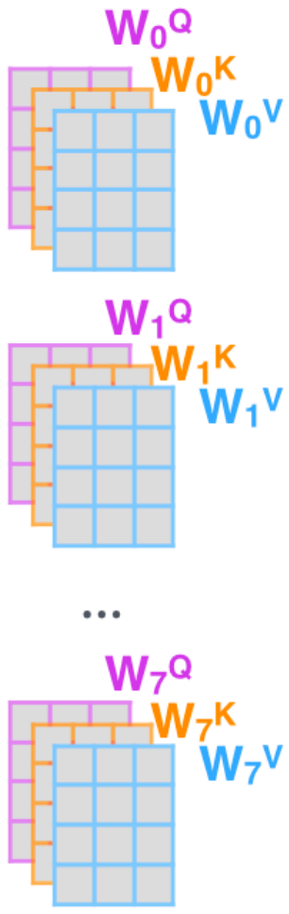
1) This is our input sentence*

Thinking
Machines

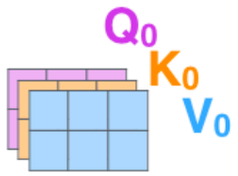
2) We embed each word*

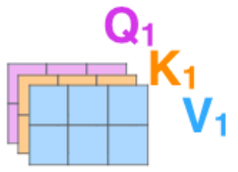


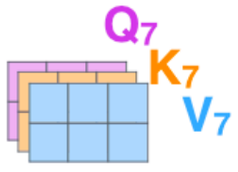
3) Split into 8 heads. We multiply X or R with weight matrices



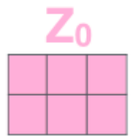
4) Calculate attention using the resulting Q/K/V matrices

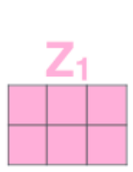


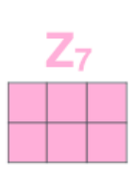


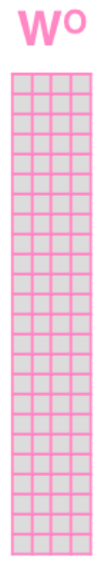


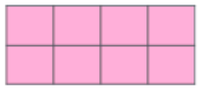
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer









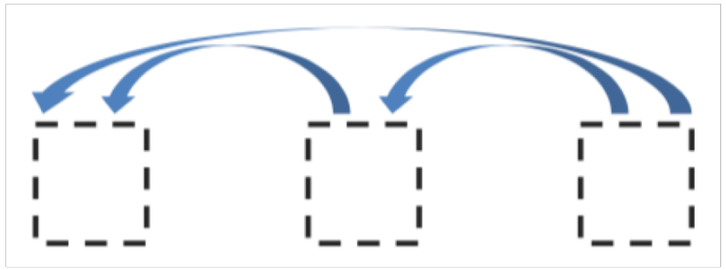


* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

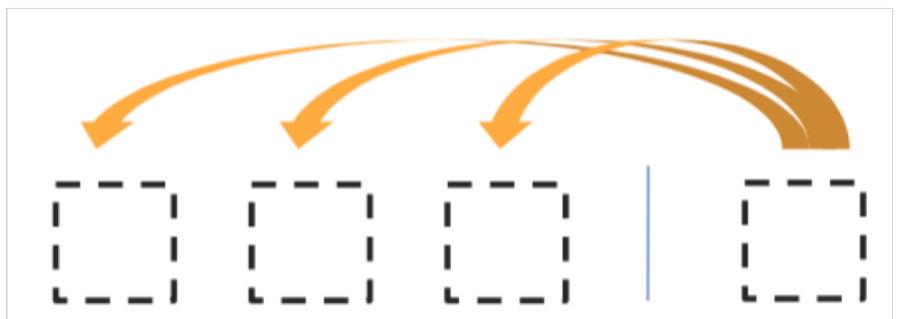
source: https://jalammar.github.io/

Decoder Layers

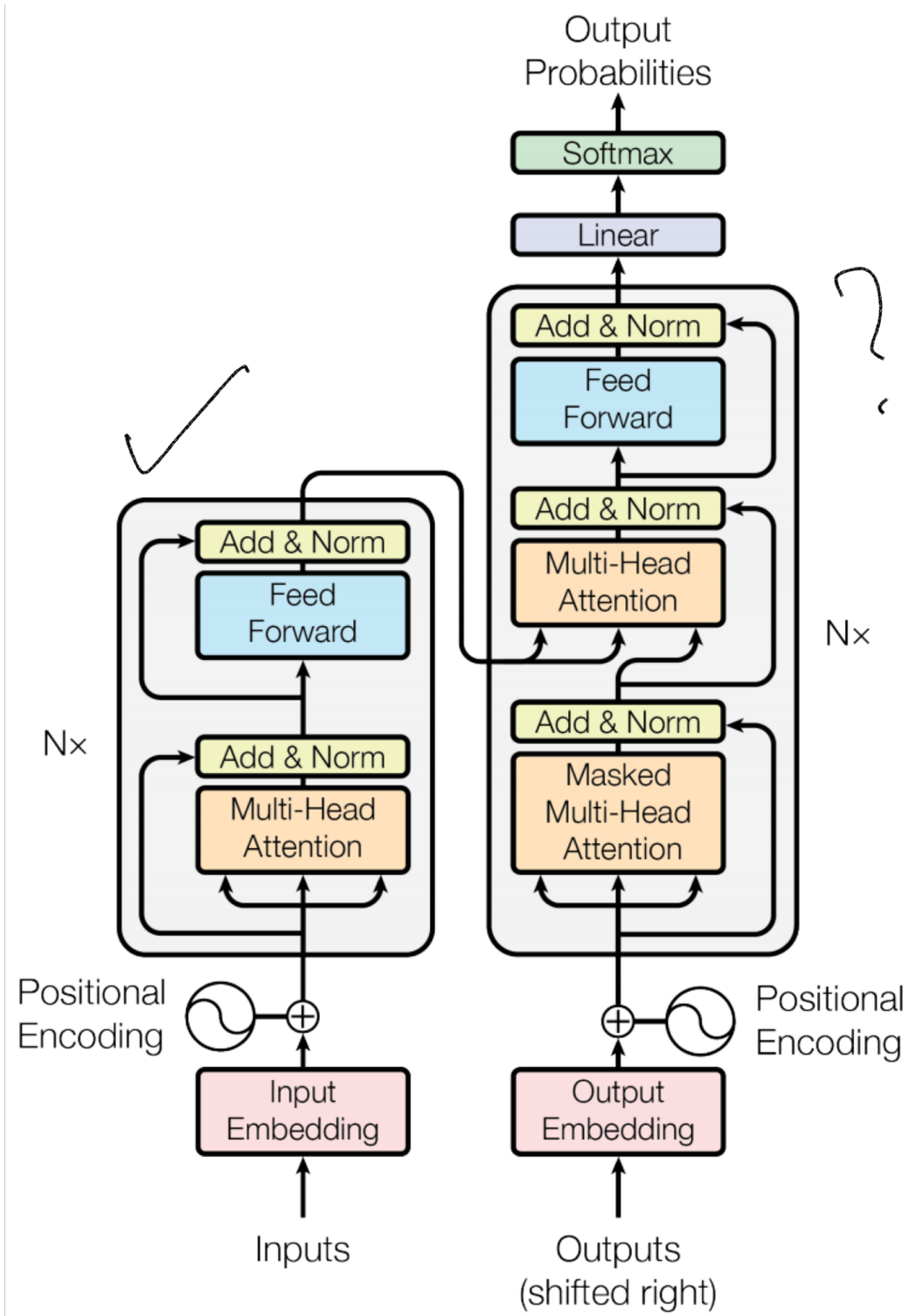
- 2 sublayer changes in decoder
- Masked decoder self-attention on previously generated outputs:



- Encoder-Decoder Attention, where queries come from previous decoder layer and keys and values come from output of encoder



- Blocks repeated 6 times also



Transformer Resources

Learning about transformers on your own?

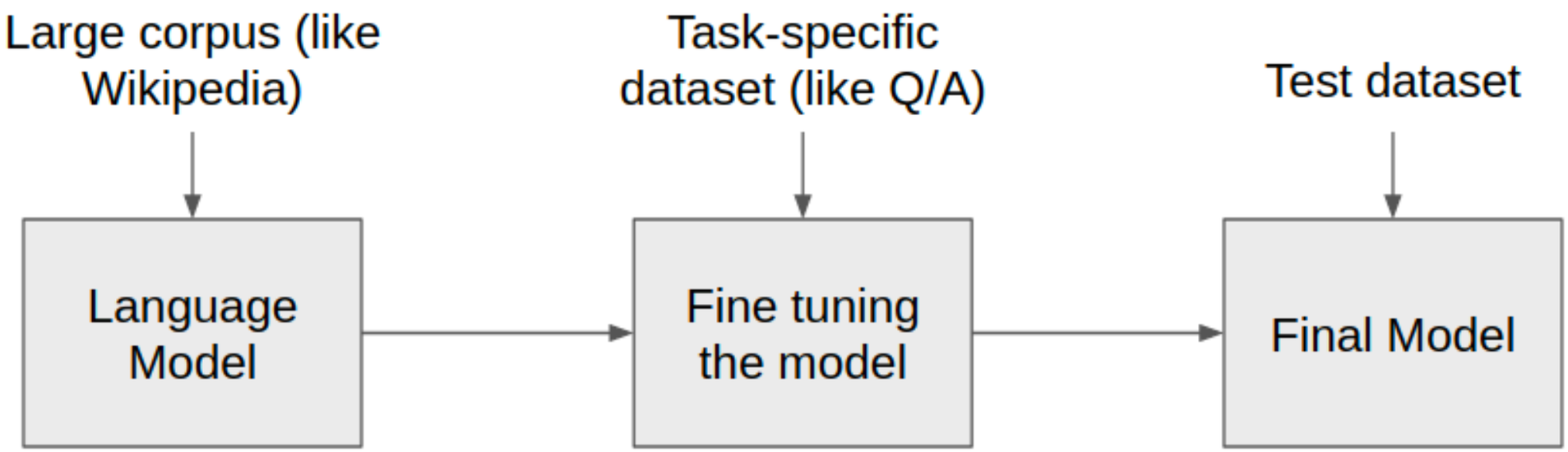
- Key recommended resource:
  - http://nlp.seas.harvard.edu/2018/04/03/attention.html
- The Annotated Transformer by SashaRush

  Jupyter Notebook using PyTorch that explains everything!

Plan

- NLP & why deep learning for NLP [10 min]
- FFNs & word vectors [15 min]
- RNNs & sequence level NLP tasks [20 min]
- Break [30 min]
- Transformers [13 min]
- Self-supervised learning [12 min]

Pretrain-finetune Paradigm



- What do we transfer?
  - Model vs. Embeddings
- What do we pre-train
  - Encoder or Decoder or Encoder-Decoder?
- What is the pre-training objective
  - Language Model? Translation? Cloze? ….

Source: Internet

GPT & GPT-2

Main Idea: Train a **Transformer Decoder** as conditional LM

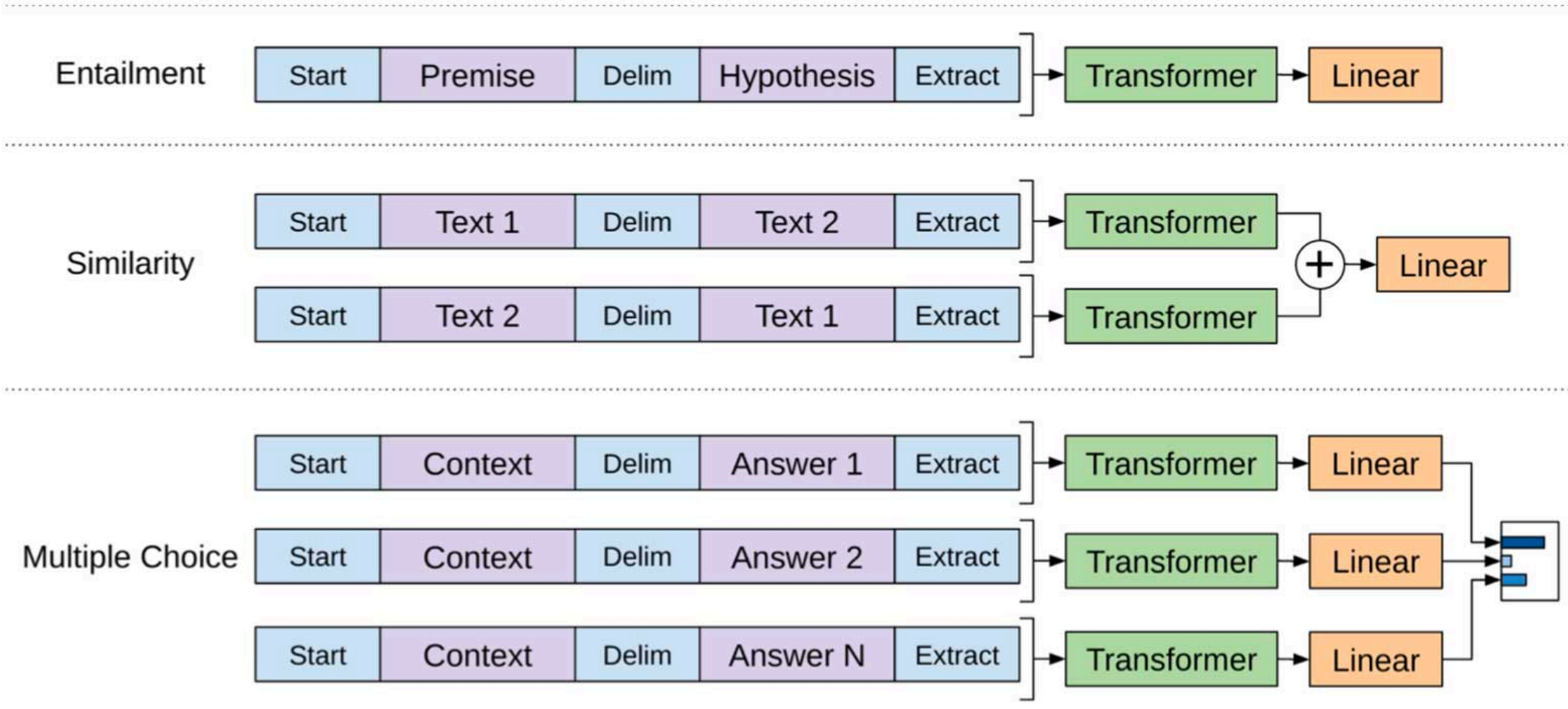
Train Deep (12-layer) Transformer LM

Fine-tune on Classification Task

Improving Language Understanding by Generative Pre-Training, OpenAI, 2018

GPT & GPT-2

- Fine-tuning



Scaling Up!

All of these models are Transformer architecture models; we already know Transformer

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

BERT

(Bidirectional Encoder Representations from Transformers)

Devlin, Chang, Lee, Toutanova (2018)

Why Bidirectionality?

- **Problem**: Language models only use left context or right context, but language understanding is bidirectional.

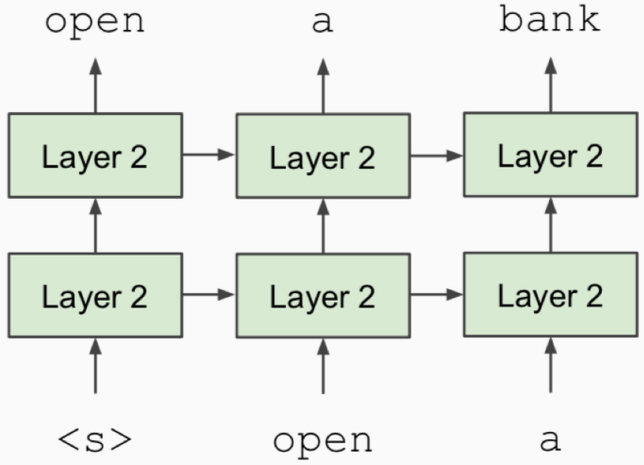
- Why are LMs unidirectional?

- 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.

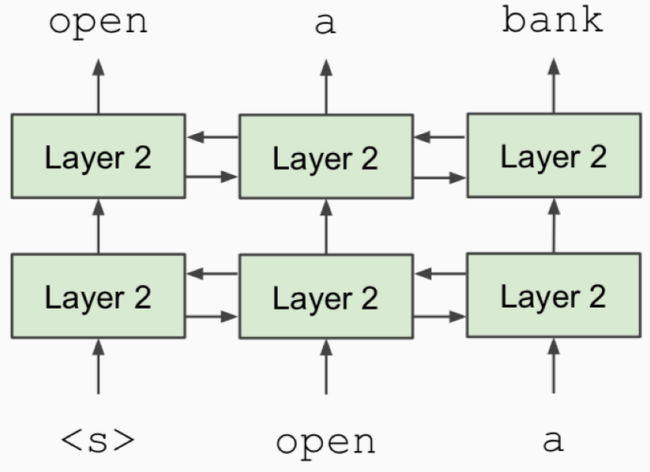
source: Devlin et al. 2019

Bidirectionality?

Unidirectional context
Build representation incrementally



Bidirectional context
Words can “see themselves”



source: Devlin et al. 2019

Masked LM

- **Solution**: Mask out k% of the input words, and then predict the masked words
  - They always use k = 15%

- Too little masking: Too expensive to train
- Too much masking: Not enough context

source: Devlin et al. 2019

Masked LM

- Problem: Mask token never seen at fine-tuning
- Solution: 15% of the words to predict, but don't replace with [MASK] 100% of the time. Instead:
  - 80% of the time, replace with [MASK]

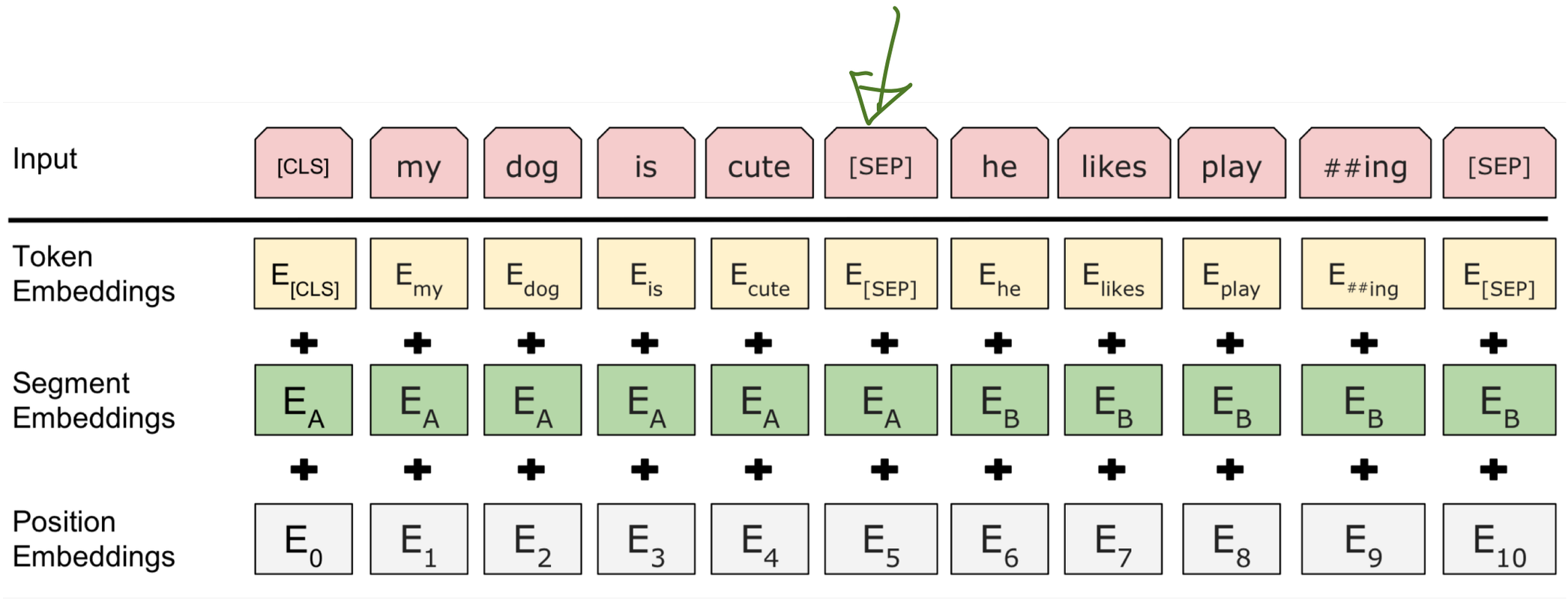
    went to the store → went to the [MASK]
  - 10% of the time, replace random word

    went to the store → went to the running
  - 10% of the time, keep the same

    went to the store → went to the store

source: Devlin et al. 2019

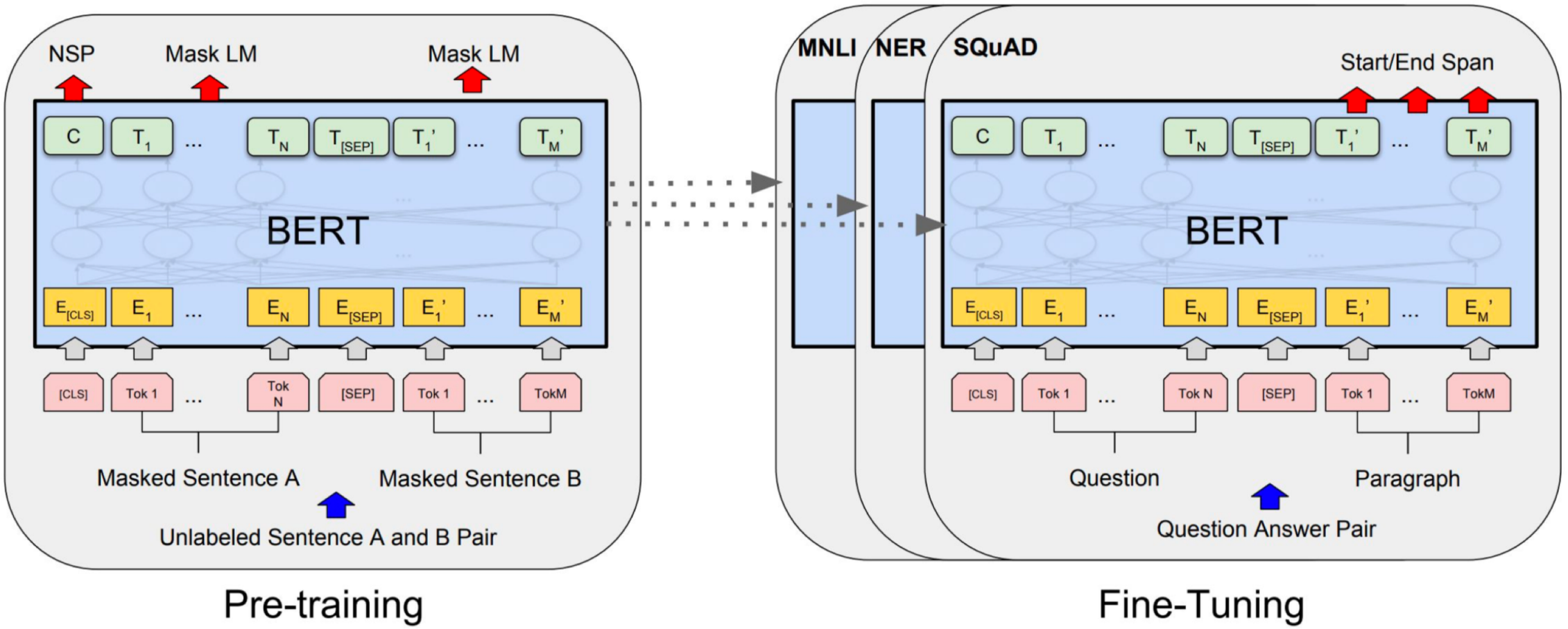
BERT sentence pair encoding



- Token embeddings are word pieces
- Learned segmented embedding represents each sentence
- Positional embedding is as for other Transformer architectures

BERT Fine-tuning

Simply learn a classifier built on the top layer for each task that you fine tune for



BERT Results (GLUE tasks)

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

- GLUE benchmark is dominated by natural language inference tasks, but also has sentence similarity and sentiment
- MultiNLI
  - Premise: Hills and mountains are especially sanctified in Jainism.
    Hypothesis: Jainism hates nature.
    Label: Contradiction
- CoLa
  - Sentence: The wagon rumbled down the road. Label: Acceptable
    Sentence: The car honked down the road. Label: Unacceptable

BERT Results (NER)

CoNLL 2003 Named Entity Recognition (en news testb)

Name	Description	Year	F1
Flair (Zalando)	Character-level language model	2018	93.09
BERT Large	Transformer bidi LM + fine tune	2018	92.8
CVT Clark	Cross-view training + multitask learn	2018	92.61
BERT Base	Transformer bidi LM + fine tune	2018	92.4
ELMo	ELMo in BiLSTM	2018	92.22
TagLM Peters	LSTM BiLM in BiLSTM tagger	2017	91.93
Ma + Hovy	BiLSTM + char CNN + CRF layer	2016	91.21
Tagger Peters	BiLSTM + char CNN + CRF layer	2017	90.87
Ratinov + Roth	Categorical CRF+Wikipeda+word cls	2009	90.80
Finkel et al.	Categorical feature CRF	2005	86.86
IBM Florian	Linear/softmax/TBL/HMM ensemble, gazettes++	2003	88.76

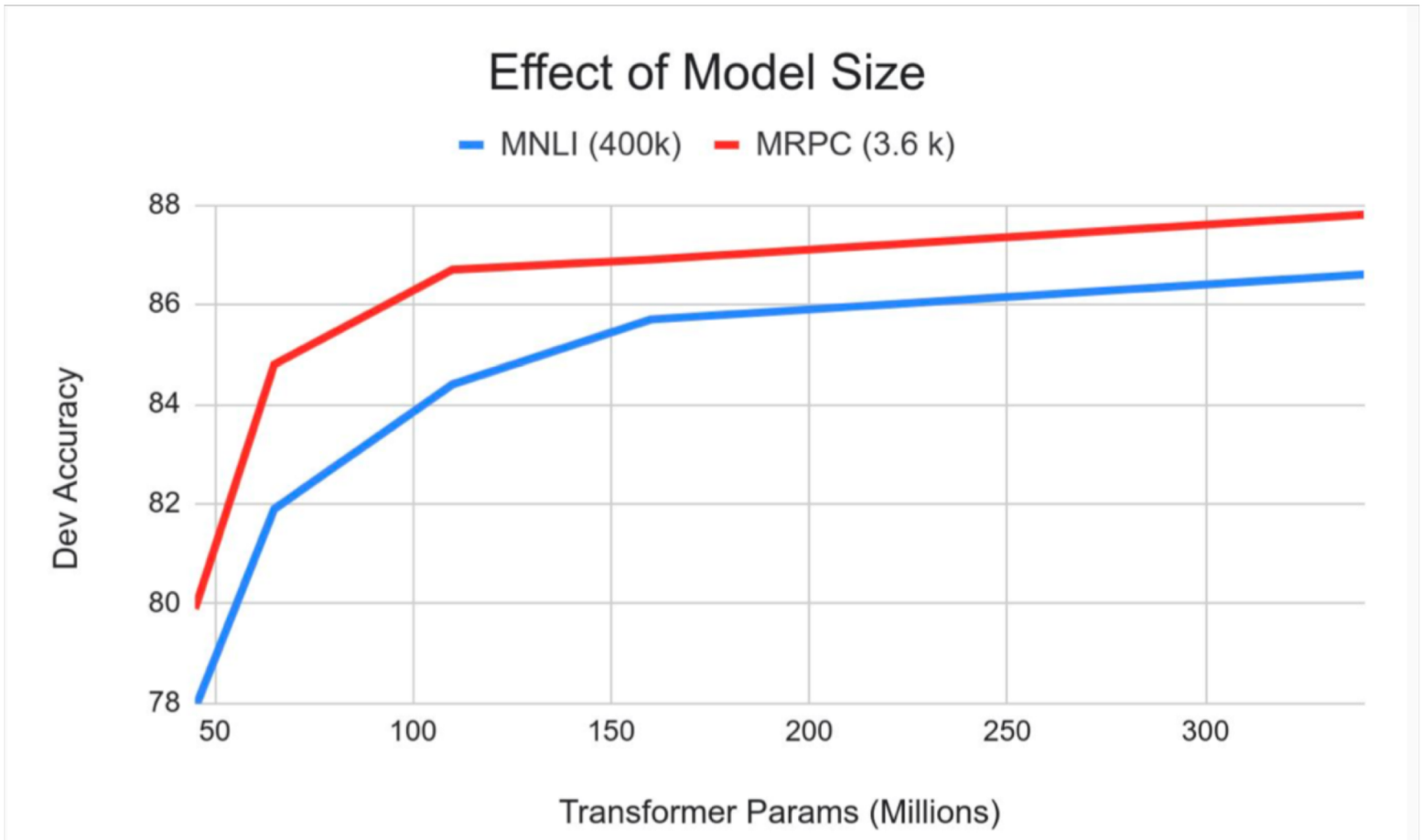
BERT Results (SQuAD 1.1)

Rank	Model	EM	F1
	Human Performance *Stanford University* (Rajpurkar et al. '16)	82.304	91.221
1 Oct 05, 2018	BERT (ensemble) *Google AI Language* https://arxiv.org/abs/1810.04805	**87.433**	**93.160**
2 Oct 05, 2018	BERT (single model) *Google AI Language* https://arxiv.org/abs/1810.04805	85.083	91.835
2 Sep 26, 2018	nlnet (ensemble) *Microsoft Research Asia*	85.954	91.677
5 Sep 09, 2018	nlnet (single model) *Microsoft Research Asia*	83.468	90.133
3 Jul 11, 2018	QANet (ensemble) *Google Brain & CMU*	84.454	90.490

BERT Results

Size matters

- Going from 110M to 340M parameters helps a lot
- Improvements have not yet asymptoted



Questions?