EE-608: Deep Learning For Natural Language Processing

James Henderson

Idiap Research Institute

DLNLP, Lecture 1

Outline

Course Introduction

Course Overview

Word Embeddings

Language Modelling

Recurrent Neural Networks

LSTMs

Attention Function

Outline

Course Introduction

Course Overview

Word Embeddings

Language Modelling

Recurrent Neural Networks

LSTMs

Attention Function

EE-608: Deep Learning For Natural Language Processing

Lecturer: James Henderson

TAs: Haruki Shirakami, TBD

Emails: *first.last*@idiap.ch

Lectures: normally Mondays 09:15–11:00, MXG 110
Slides will be uploaded before each lecture.

Assistance: normally Mondays 11:15–13:00, MXG 110
Exercise sessions will be used for help with projects.

Next week lecture will be at 11:15.

Course website:
https://go.epfl.ch/EE-608

Evaluation

We will do evaluation with a course project

- Teams of 2-4 people
- Written proposal for the project (10% of grade)
- Written report on the project (60% of grade)
- Oral presentation and answers to questions (30% of grade)

What do we plan to teach?

Provide an in-depth analysis of attention-based neural network architectures for text

- An understanding of the basic properties of human language
  - structured, variable length (unbounded),
  - categorical and continuous, large vocabulary
- An understanding of attention-based deep learning methods used for language
  - Seq2seq, attention, Transformers
  - Graph2graph, attention, Transformers
  - Variational-Bayesian methods, attention, Transformers
  - Multi-task learning, pretraining, finetuning
  - Diffusion and denoising
  - . . .
- The ability to build systems for NLP tasks
  - Project of your choice

This course is work in progress

- This is the first time we have given an advanced version of DLNLP
- For further reading and some alternative topics, see the Stanford course “Natural Language Processing with Deep Learning”
  `http://web.stanford.edu/class/cs224n/`
- Choice of topics will partly depend on **student input**

Expected Student Background

This year, I will not accept MSc students.

- What are you studying?
- What topics do you already know?
  - Probability and statistics
  - Neural networks
  - Transformers
  - Natural language processing
- What programming experience do you have?
  - Python, NumPy
  - PyTorch

Outline

Course Introduction

Course Overview

Word Embeddings

Language Modelling

Recurrent Neural Networks

LSTMs

Attention Function

Deep Learning is Representation Learning

The most important property of deep learning models is their ability to learn their own latent representations.

- vector representations embed features and categories (and possibly more) in a low-dimensional continuous vector space
- set-of-vectors representations can also embed relationships between vectors, and can be arbitrarily large
- sequence-of-vectors representations add an explicit ordering on vectors
- graphs-of-vectors add explicit graph relations between vectors

Deep Learning is Representation Learning

Backpropagation training allows both the encoding into a representation and the decoding from a representation to be trained jointly.
Any encoding and decoding functions can be used, provided they are differentiable.

- multi-layered perceptrons
- recurrent neural networks, LSTMs
- attention functions, pooling
- ...

When the decoding for one learned representation is the encoding of another, it is a *deep* model.

Deep Learning For Natural Language Processing

Models	Properties	NLP Tasks e.g.
word embeddings	continuous space, semantic similarity	information access
RNN, LSTM	unbounded sequences, induced state	language modelling
BiLSTM with pooling	unbounded input	text classification
Seq2Seq	unbounded output, end-to-end training	machine translation
attention	unbounded memory, content-based access	machine translation
Transformer	set-of-vectors representations, deep	language modelling

Deep Learning For Natural Language Processing

Models	Properties	NLP Tasks e.g.
NN transition-based parsing	structured prediction, unfactorised	syntactic parsing
NN graph-based parsing	structured prediction, conditionally factorised	syntactic parsing
language-model pretraining	representation learning	many tasks
BERT, GPTn, chatGPT	transfer learning	many tasks
...	...	...

The Central Question for DLNLP

How can

- the continuous, distributed, learnable representations of **neural networks**

capture

- the unbounded, structured, interpretable representations of **computational linguistics**?

My answer:

- probability distributions over mixture distributions and an **attention** mechanism.

The unbounded integer number of components in a mixture distribution captures the unbounded integer number of nodes in a graph structure. The attention function embeds and extracts relations between graph nodes using pairs of vectors generated by pairs of mixture components. The integer number of mixture components is made learnable by learning distributions over mixture distributions.

Outline

Course Introduction

Course Overview

Word Embeddings

Language Modelling

Recurrent Neural Networks

LSTMs

Attention Function

Representing words as discrete symbols

In traditional NLP, we regard words as discrete symbols:

hotel, conference, motel – a localist representation

Means one 1, the rest 0s

Such symbols for words can be represented by one-hot vectors:

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]

Vector dimension = number of words in vocabulary (e.g., 500,000+)

Slide from Christopher Manning

Problem with words as discrete symbols

Example: in web search, if a user searches for "Seattle motel", we would like to match documents containing "Seattle hotel"

But:

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

There is no natural notion of **similarity** for one-hot vectors!

Solution:

- Could try to rely on WordNet's list of synonyms to get similarity?
  - But it is well-known to fail badly: incompleteness, etc.
- **Instead: learn to encode similarity in the vectors themselves**

Slide from Christopher Manning

Representing words by their context

- 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)
  - One of the most successful ideas of modern statistical NLP!
- When a word *w* appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).
- We 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***

Slide from Christopher Manning

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, measuring similarity as the vector dot (scalar) product

$$\textit{banking} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix} \qquad \textit{monetary} = \begin{pmatrix} 0.413 \\ 0.582 \\ -0.007 \\ 0.247 \\ 0.216 \\ -0.718 \\ 0.147 \\ 0.051 \end{pmatrix}$$

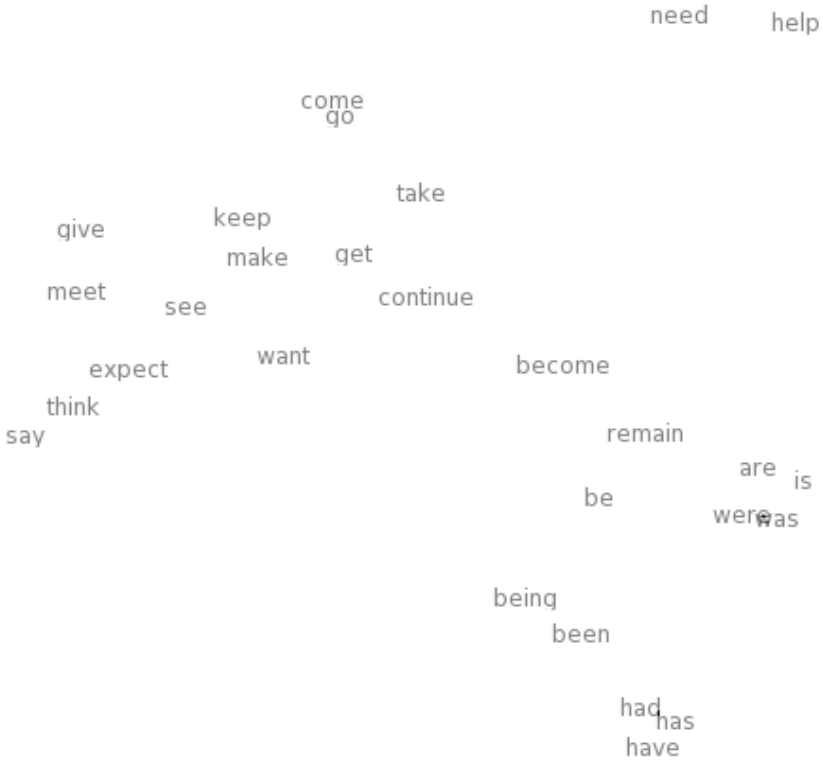
Note: word vectors are also called (word) embeddings or (neural) word representations
They are a distributed representation

Slide from Christopher Manning

Word meaning as a neural word vector – visualization

expect =

$$\begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$



Slide from Christopher Manning

Intrinsic word vector evaluation

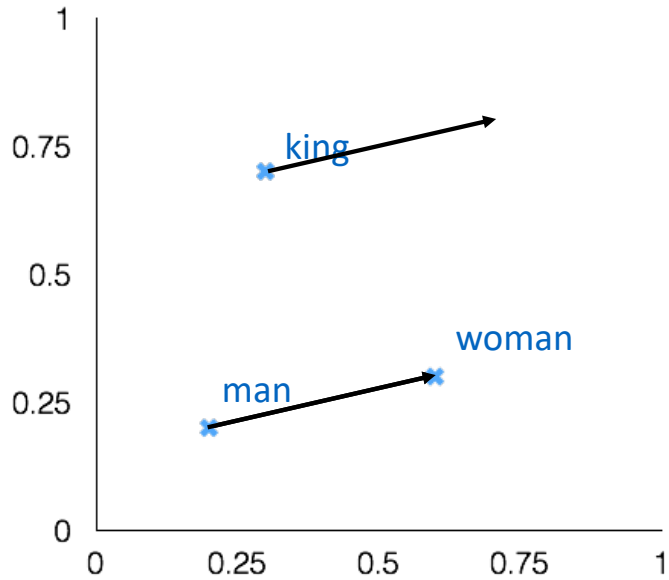
- Word Vector Analogies

a:b :: c:?

man:woman :: king:?

$$d = \arg\max_i \frac{(x_b - x_a + x_c)^T x_i}{||x_b - x_a + x_c||}$$

- Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactic analogy questions
- Discarding the input words from the search (!)
- Problem: What if the information is there but not linear?



Slide from Christopher Manning

Extrinsic word vector evaluation

- One example where good word vectors should help directly: **named entity recognition**: identifying references to a person, organization or location: Chris Manning lives in Palo Alto.

Model	Dev	Test	ACE	MUC7
Discrete	91.0	85.4	77.4	73.4
SVD	90.8	85.7	77.3	73.7
SVD-S	91.0	85.5	77.6	74.3
SVD-L	90.5	84.8	73.6	71.5
HPCA	92.6	**88.7**	81.7	80.7
HSMN	90.5	85.7	78.7	74.7
CW	92.2	87.4	81.7	80.2
CBOW	93.1	88.2	82.2	81.1
GloVe	**93.2**	88.3	**82.9**	**82.2**

- Subsequent NLP tasks in this class are other examples. So, more examples soon.

Slide from Christopher Manning

Summary of Word Embeddings

- Similarity of word meanings is a continuous, multi-dimentional phenomenon which is hard to define
- This is addressed by embedding words in a continuous vector space, called “word embeddings”, and using the dot product to predict word similarity
- Word embeddings can be learned automatically by training them to predict word coocurrance in (very) large corpora
- Words are similar if they have similar distributions of context words
- Intrinsic evaluations measure useful properties in word embeddings (e.g. similarity as dot product, analogy as vector difference)
- Extrinsic evaluations test embeddings for their usefulness in other tasks

Outline

Course Introduction

Course Overview

Word Embeddings

Language Modelling

Recurrent Neural Networks

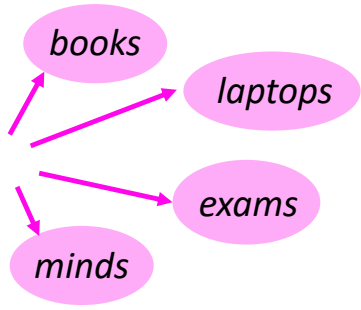
LSTMs

Attention Function

2. Language Modeling

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

the students opened their _____



- 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 $V = \{\boldsymbol{w}_1, ..., \boldsymbol{w}_{|V|}\}$

- A system that does this is called a **Language Model**

Slide from Christopher Manning

Language Modeling

- You can also think of a Language Model as a system that assigns a probability to a piece of text

- For example, if we have some text $x^{(1)}, \ldots, x^{(T)}$, 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

Slide from Christopher Manning

Evaluating Language Models

- The standard evaluation metric for Language Models is perplexity.

$$\text{perplexity} = \prod_{t=1}^{T} \left(\frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)^{1/T}$$

Normalized by number of words

Inverse probability of corpus, according to Language Model

- This is equal to the exponential of the cross-entropy loss $J(\theta)$:

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

Lower perplexity is better!

Slide from Christopher Manning

A fixed-window neural Language Model

output distribution

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

hidden layer

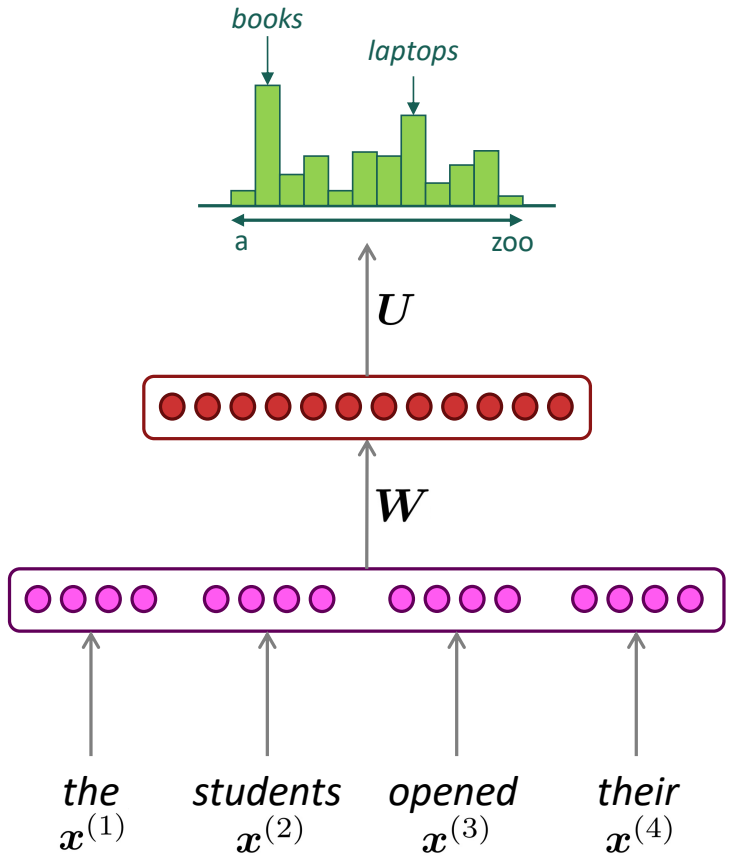
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



Slide from Christopher Manning

A fixed-window neural Language Model

Approximately: Y. Bengio, et al. (2000/2003): A Neural Probabilistic Language Model

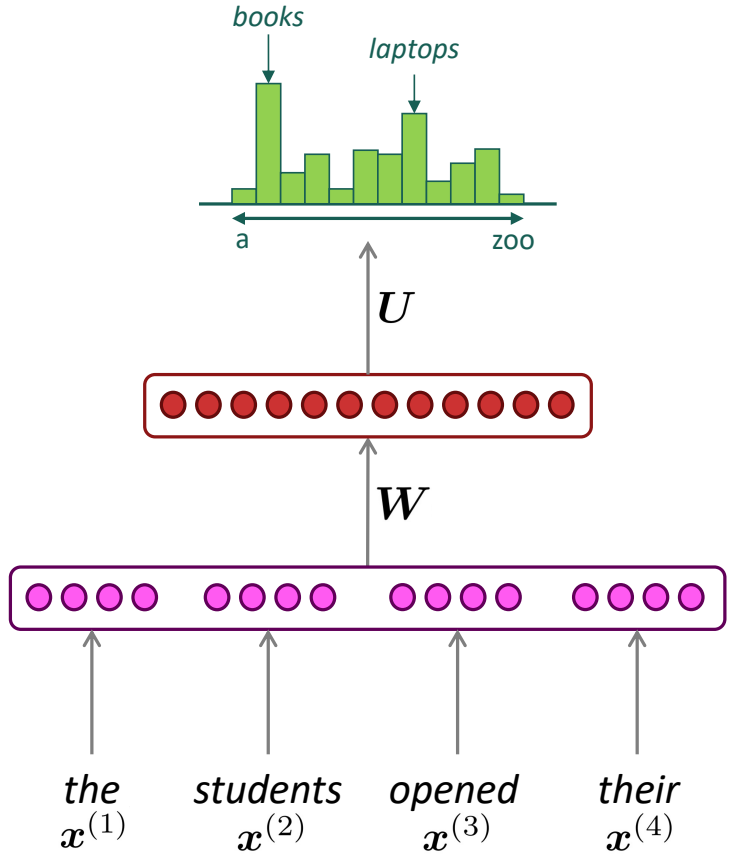
Improvements over *n*-gram LM:

- No sparsity problem
- Don't need to store all observed *n*-grams

Remaining **problems**:

- Fixed window is too small
- Enlarging window enlarges W
- Window can never be large enough!
- $x^{(1)}$ and $x^{(2)}$ are multiplied by completely different weights in W. No symmetry in how the inputs are processed.

We need a neural architecture that can process *any length input*



Slide from Christopher Manning

Outline

Course Introduction

Course Overview

Word Embeddings

Language Modelling

Recurrent Neural Networks

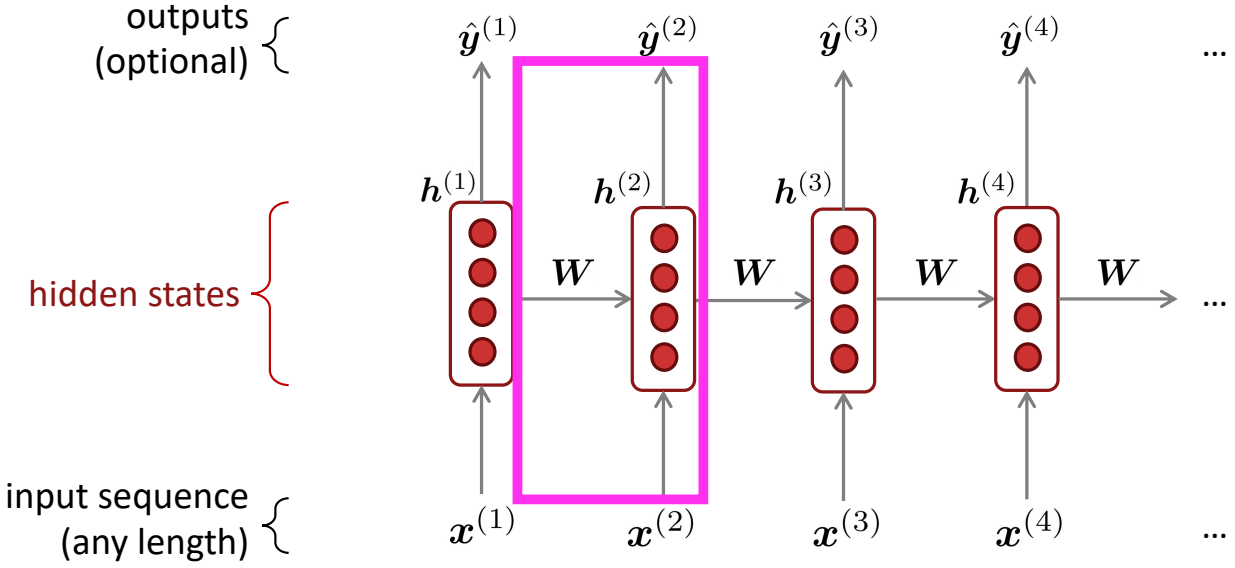
LSTMs

Attention Function

3. Recurrent Neural Networks (RNN)

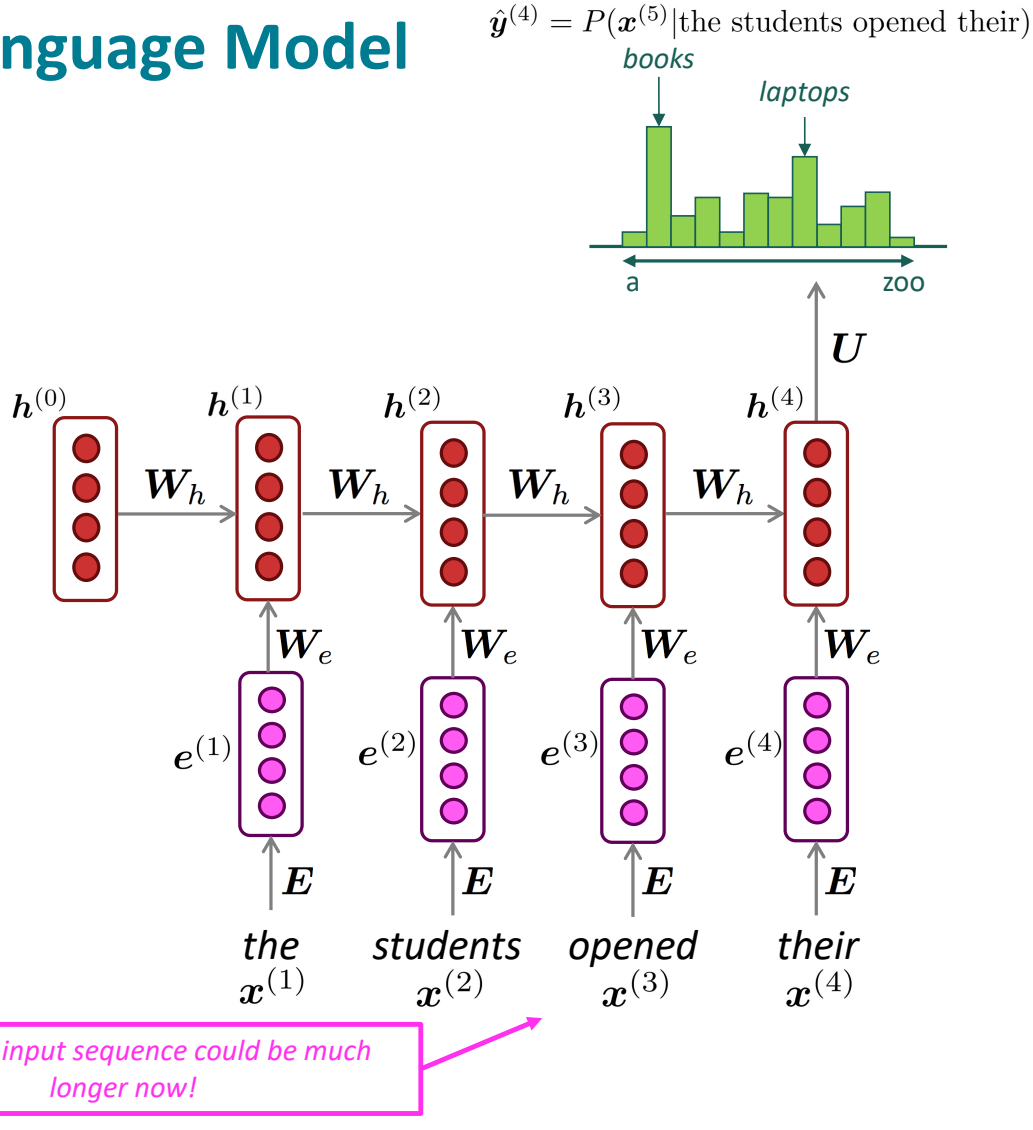
A family of neural architectures

Core idea: Apply the same weights *W repeatedly*



Slide from Christopher Manning

A Simple 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

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

Note: *this input sequence could be much longer now!*

Slide from Christopher Manning

RNN Language Models

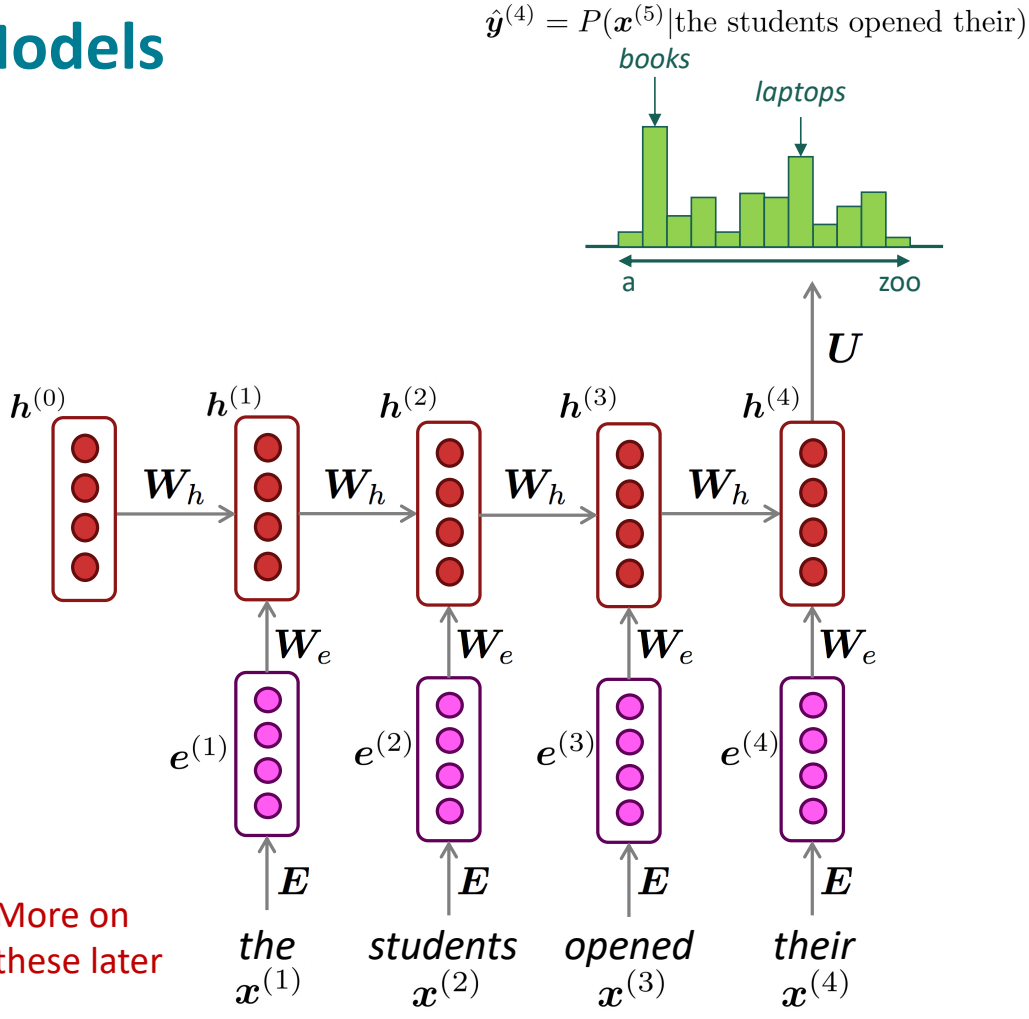
RNN **Advantages**:

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

RNN **Disadvantages**:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back

More on these later



Slide from Christopher Manning

Summary of Language Modelling and RNNs

- Modelling the distribution over strings of words in natural language is a fundamental task
- This task can be decomposed into predicting the next word given the prefix of previous words (as in auto-regressive generation)
- Recurrent neural networks (RNNs) can encode arbitrarily long prefixes by passing information through arbitrarily many hidden representations
- Log-likelihood training gives probability estimates for the next word
- Using the same weights for every position allows RNNs to generalise across sequence positions and lengths

Outline

Course Introduction

Course Overview

Word Embeddings

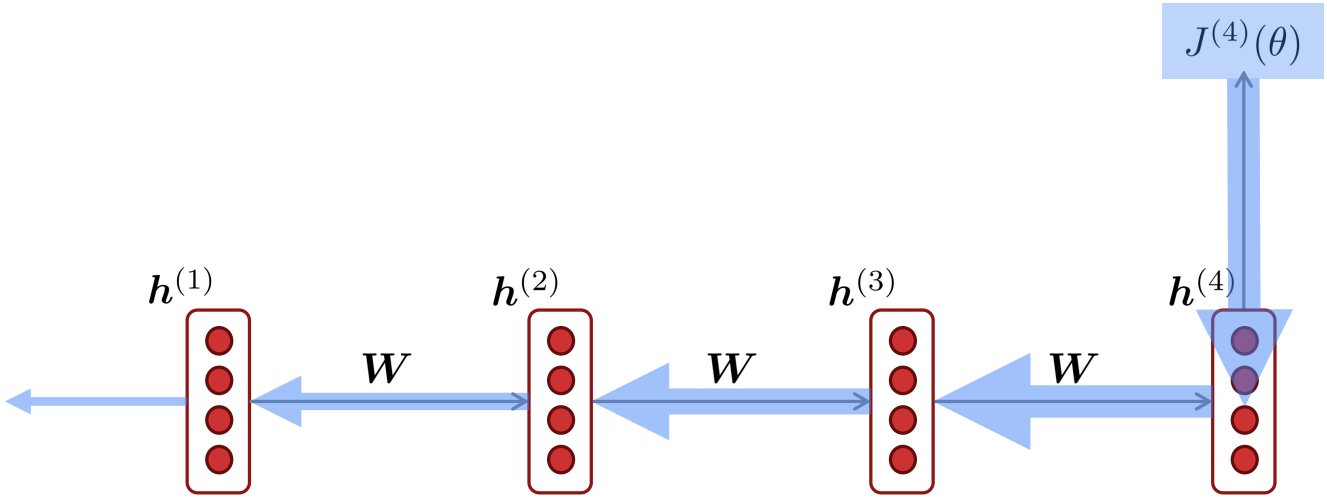
Language Modelling

Recurrent Neural Networks

LSTMs

Attention Function

Vanishing gradient intuition



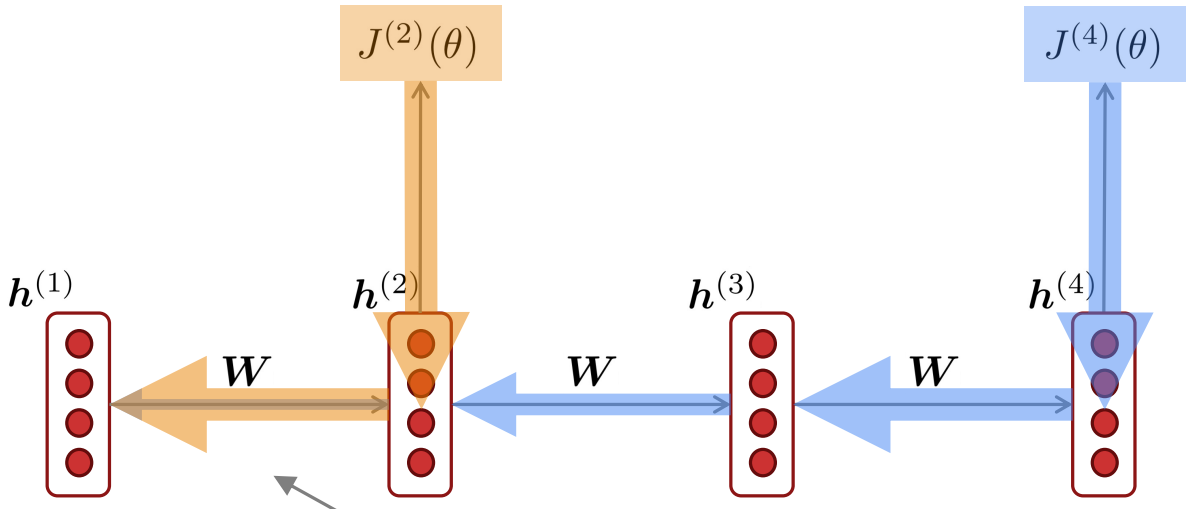
$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

What happens if these are small?

Vanishing gradient problem:
When these are small, the gradient signal gets smaller and smaller as it backpropagates further

Slide from Christopher Manning

Why is vanishing gradient a problem?



Gradient signal from far away is lost because it's much smaller than gradient signal from close-by.

So, model weights are basically updated only with respect to near effects, not long-term effects.

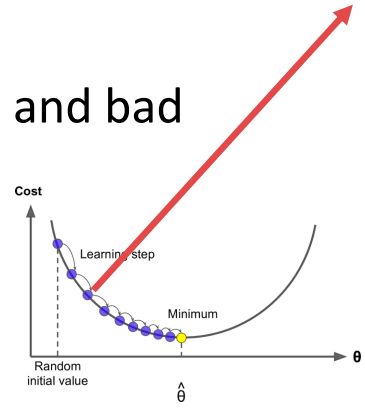
Slide from Christopher Manning

Why is exploding gradient a problem?

- If the gradient becomes too big, then the SGD update step becomes too big:

$$\theta^{new} = \theta^{old} - \underbrace{\overbrace{\alpha}^{\text{learning rate}} \nabla_\theta J(\theta)}_{\text{gradient}}$$

- This can cause bad updates: we take too large a step and reach a weird and bad parameter configuration (with large loss)
  - You think you've found a hill to climb, but suddenly you're in Iowa



- In the worst case, this will result in Inf or NaN in your network (then you have to restart training from an earlier checkpoint)

Slide from Christopher Manning

Gradient clipping: solution for exploding gradient

- **Gradient clipping**: if the norm of the gradient is greater than some threshold, scale it down before applying SGD update

Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if

- **Intuition**: take a step in the same direction, but a smaller step

- In practice, **remembering to clip gradients is important**, but exploding gradients are an easy problem to solve

Source: "On the difficulty of training recurrent neural networks", Pascanu et al, 2013. http://proceedings.mlr.press/v28/pascanu13.pdf

Slide from Christopher Manning

How to fix the vanishing gradient problem?

- The main problem is that *it's too difficult for the RNN to learn to preserve information over many timesteps*.

- In a vanilla RNN, the hidden state is constantly being rewritten

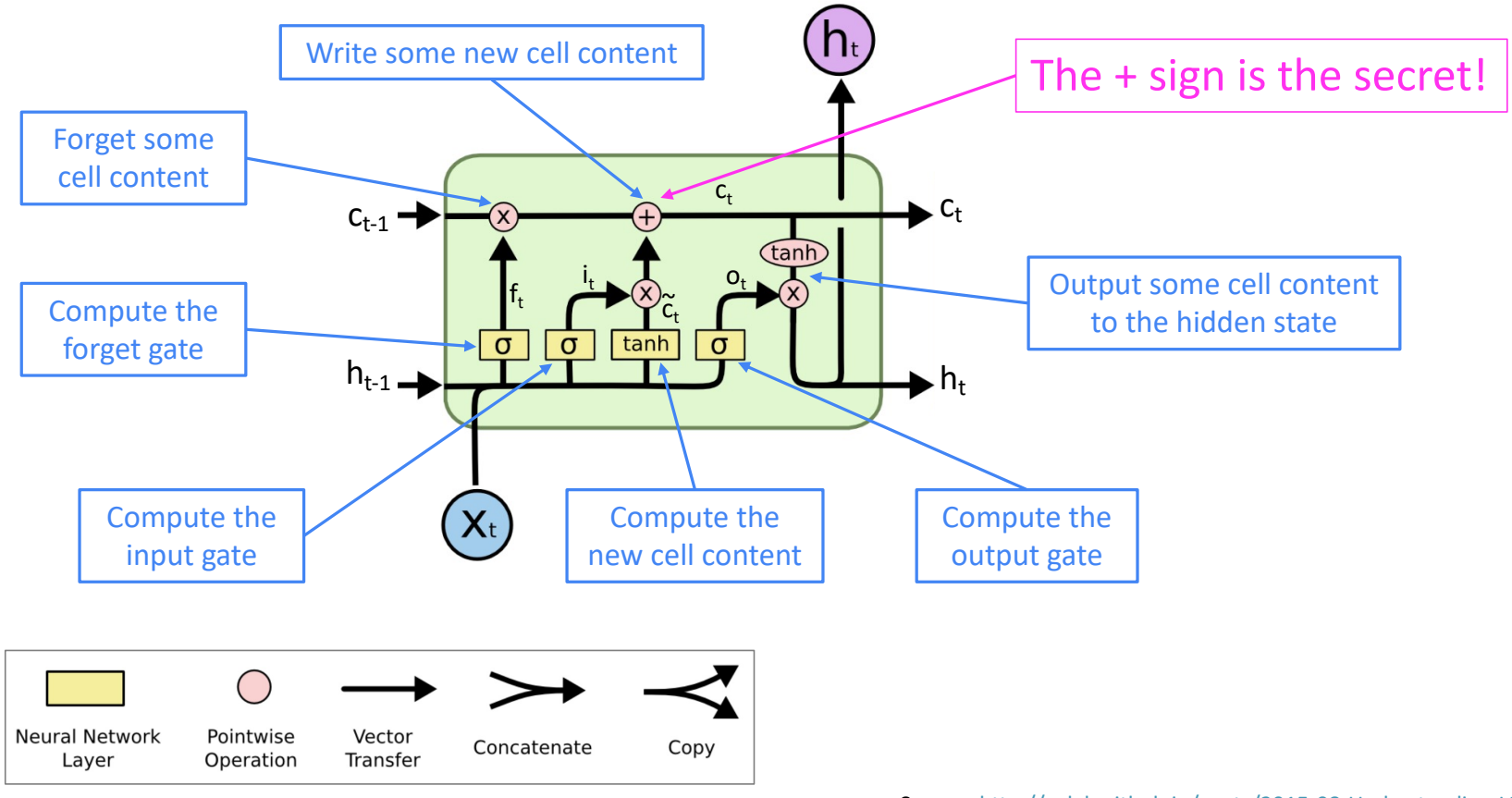
$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}\right)$$

- Could we design an RNN with separate memory which is added to?

Slide from Christopher Manning

Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:

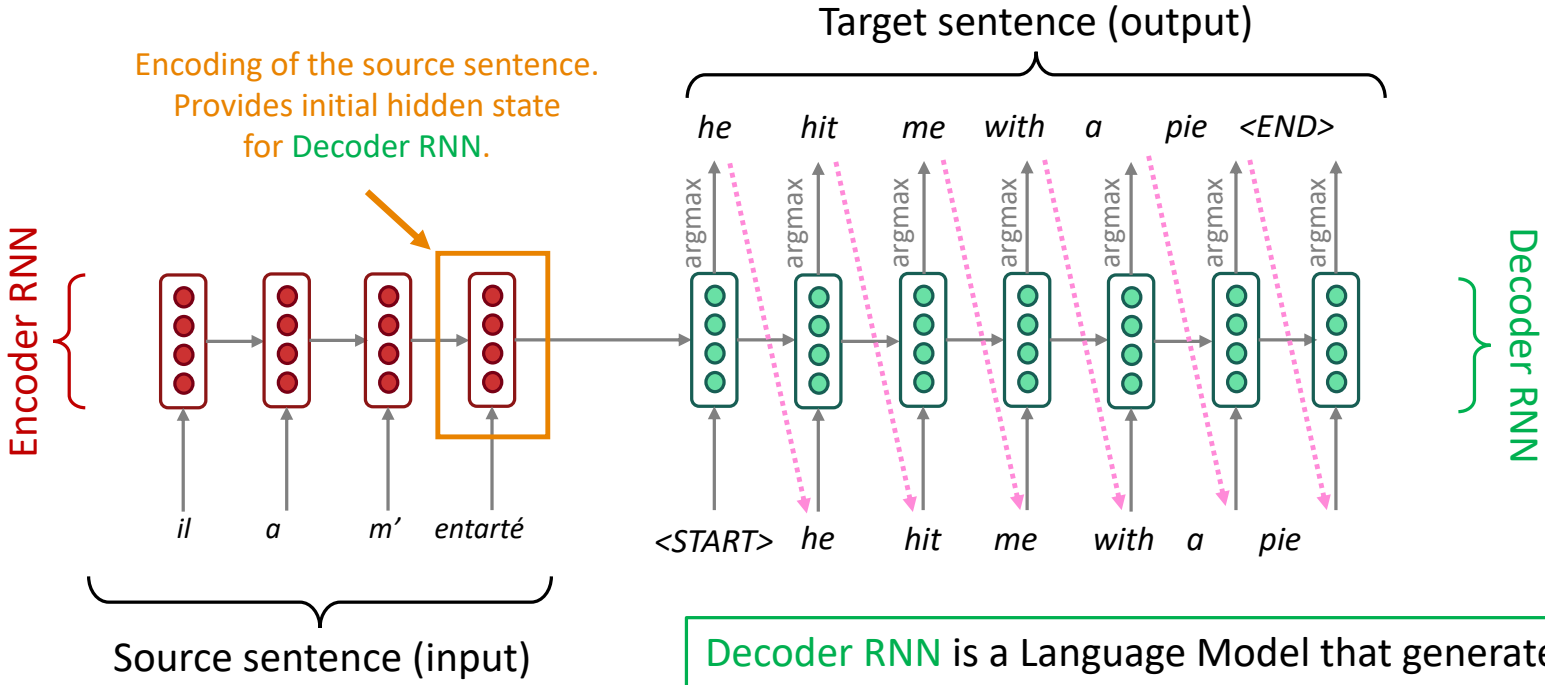


Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Slide from Christopher Manning

Neural Machine Translation (NMT)

The sequence-to-sequence model



Encoder RNN produces an encoding of the source sentence.

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

Note: This diagram shows **test time** behavior: decoder output is fed in ⇢ as next step's input

Slide from Christopher Manning

Sequence-to-sequence is versatile!

- The general notion here is an encoder-decoder model
  - One neural network takes input and produces a neural representation
  - Another network produces output based on that neural representation
  - If the input and output are sequences, we call it a seq2seq model

- Sequence-to-sequence is useful for *more than just MT*
- Many NLP tasks can be phrased as sequence-to-sequence:
  - Summarization (long text → short text)
  - Dialogue (previous utterances → next utterance)
  - Parsing (input text → output parse as sequence)
  - Code generation (natural language → Python code)

Slide from Christopher Manning

Neural Machine Translation (NMT)

- The sequence-to-sequence model is an example of a **Conditional Language Model**
  - **Language Model** because the decoder is predicting the next word of the target sentence *y*
  - **Conditional** because its predictions are *also* conditioned on the source sentence *x*

- NMT directly calculates $P(y|x)$:

$$P(y|x) = P(y_1|x)\, P(y_2|y_1, x)\, P(y_3|y_1, y_2, x) \ldots \underbrace{P(y_T|y_1, \ldots, y_{T-1}, x)}$$

Probability of next target word, given target words so far and source sentence *x*

- **Question:** How to train an NMT system?
- **(Easy) Answer:** Get a big parallel corpus…
  - But there is now exciting work on "unsupervised NMT", data augmentation, etc.

Slide from Christopher Manning

Summary of Deep Backprop with RNNs

- Backprop through time can be arbitrarily deep, which can cause vanishing or exploding gradients
- Gated identity recurrent connections can address vanishing gradients
- Gradient clipping can address exploding gradients
- LSTMs are effective ways to learn (relatively) long dependencies
- Similar problems exist in all deep neural networks, requiring skip connections or identity biases
- RNNs can be used as encoders, as well as decoders, allowing sequence-to-sequence models

Outline

Course Introduction

Course Overview

Word Embeddings

Language Modelling

Recurrent Neural Networks

LSTMs

Attention Function

Attention in NMT: Idea

- For long sentences, a fixed-length vector encoding introduces a bottleneck.
- Even for shorter sentences, conditioning on the entire sentence is hard.
- Solution: reintroduce a model of latent **alignment**, as in SMT.
- **Attention** is a soft latent alignment.

Attention

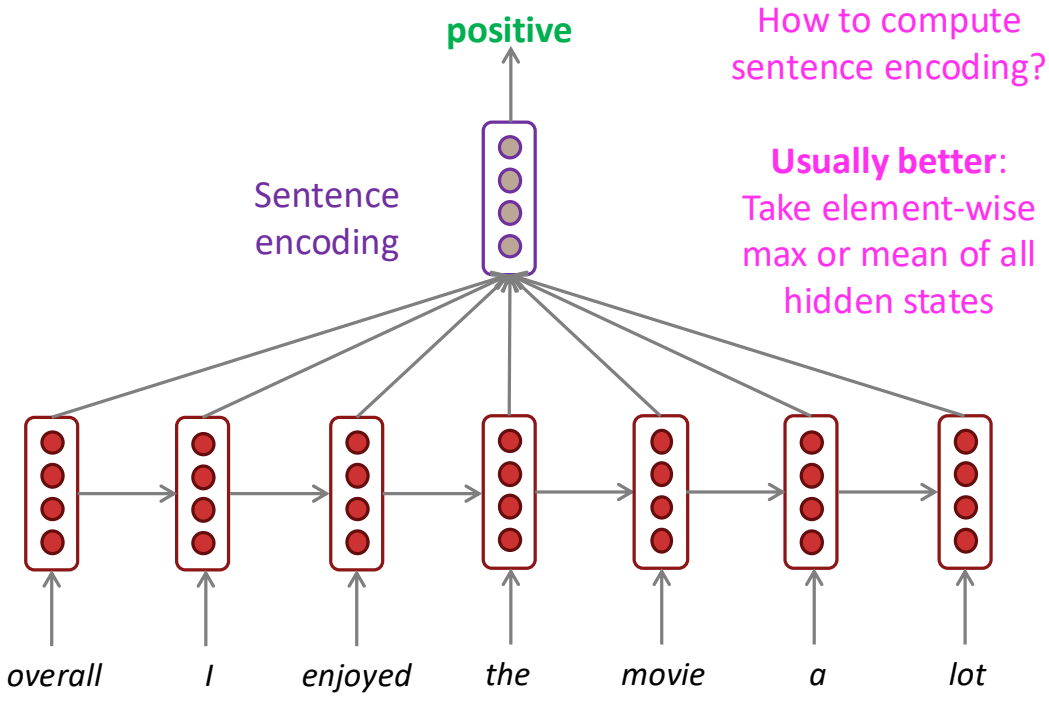
- **Attention** provides a solution to the bottleneck problem.

- **Core idea**: on each step of the decoder, use *direct connection to the encoder* to *focus on a particular part* of the source sequence

- First, we will show via diagram (no equations), then we will show with equations

Slide from Tatsunori Hashimoto

The starting point: mean-pooling for RNNs



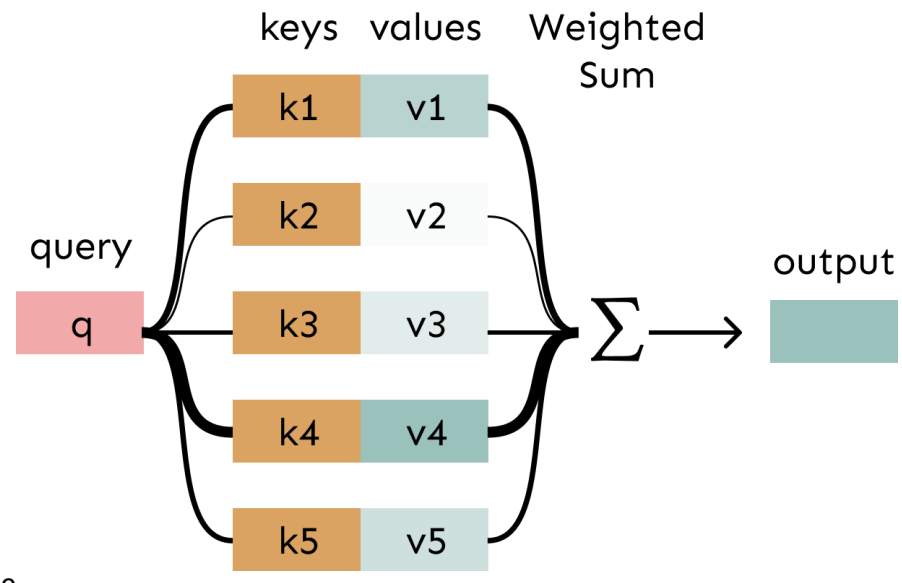
- Starting point: a *very* basic way of 'passing information from the encoder' is to *average*

Slide from Tatsunori Hashimoto

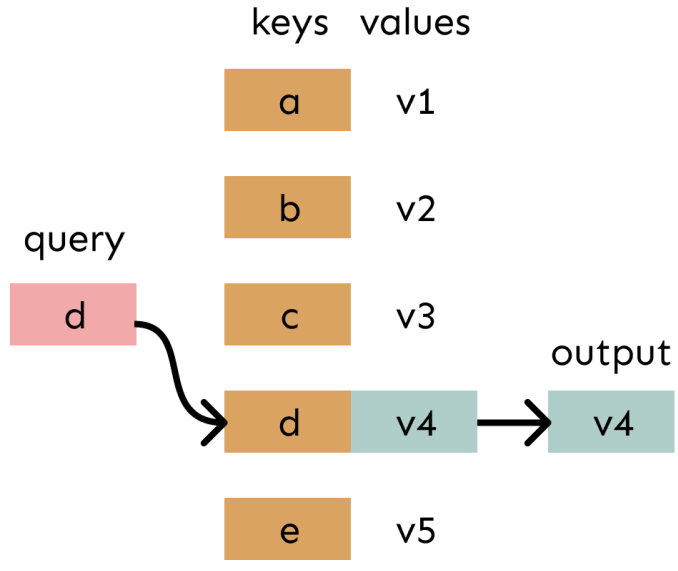
Attention is *weighted* averaging, which lets you do lookups!

Attention is just a **weighted** average – this is very powerful if the weights are learned!

In **attention**, the query matches all keys *softly*, to a weight between 0 and 1. The keys' values are multiplied by the weights and summed.



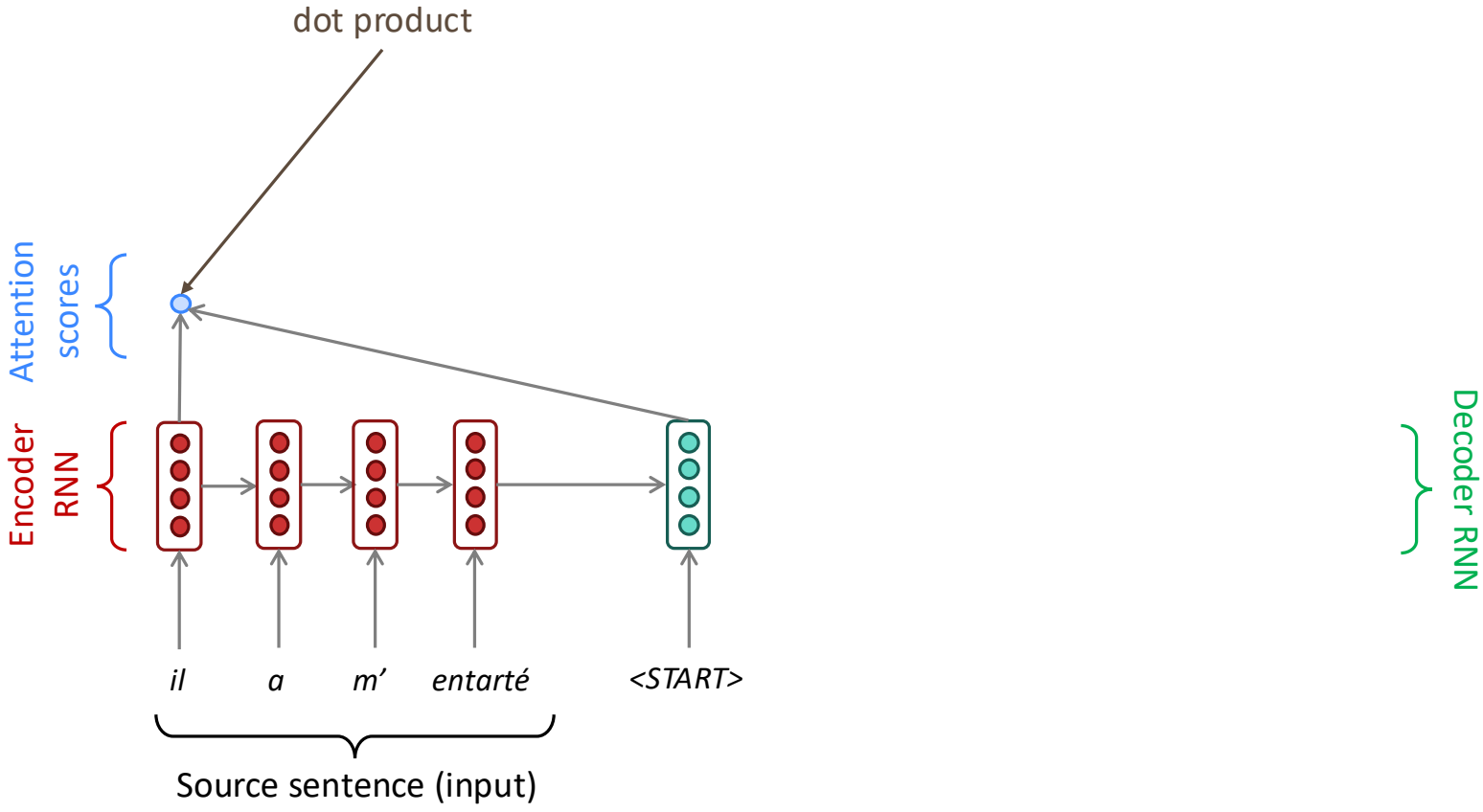
In a **lookup table**, we have a table of keys that map to values. The query matches one of the keys, returning its value.



Slide from Tatsunori Hashimoto

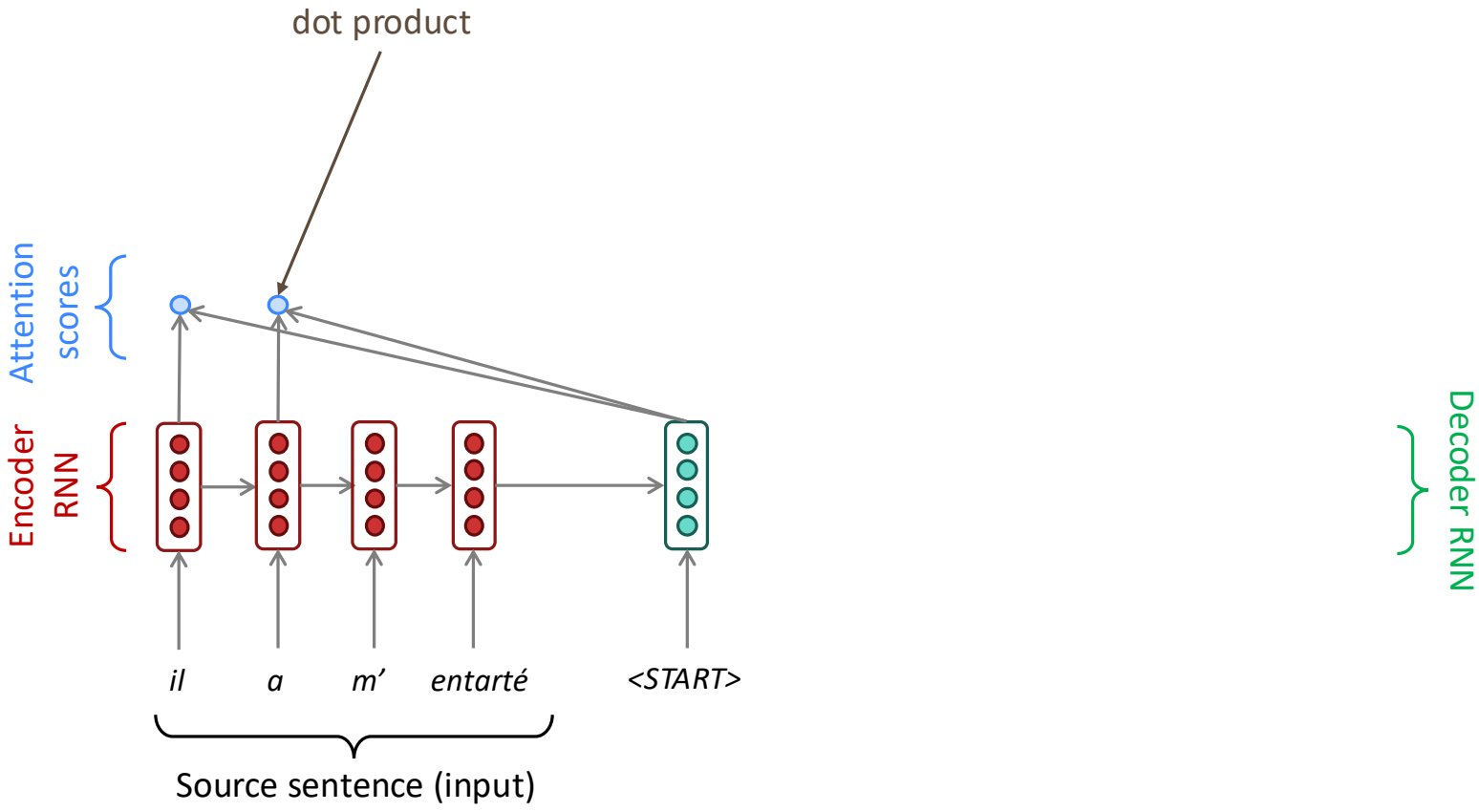
Sequence-to-sequence with attention

Core idea: on each step of the decoder, use *direct connection to the encoder* to *focus on a particular part* of the source sequence



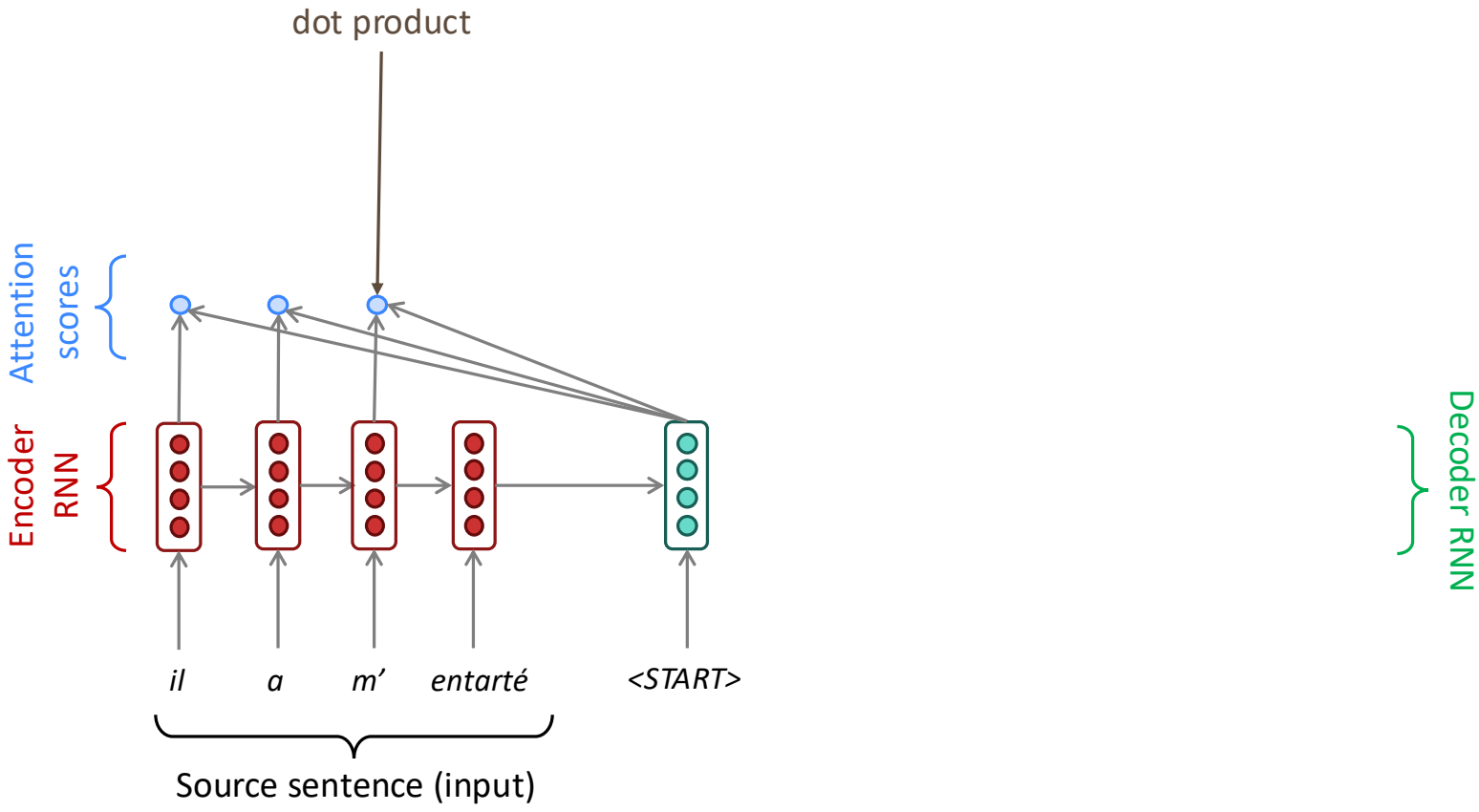
Slide from Tatsunori Hashimoto

Sequence-to-sequence with attention



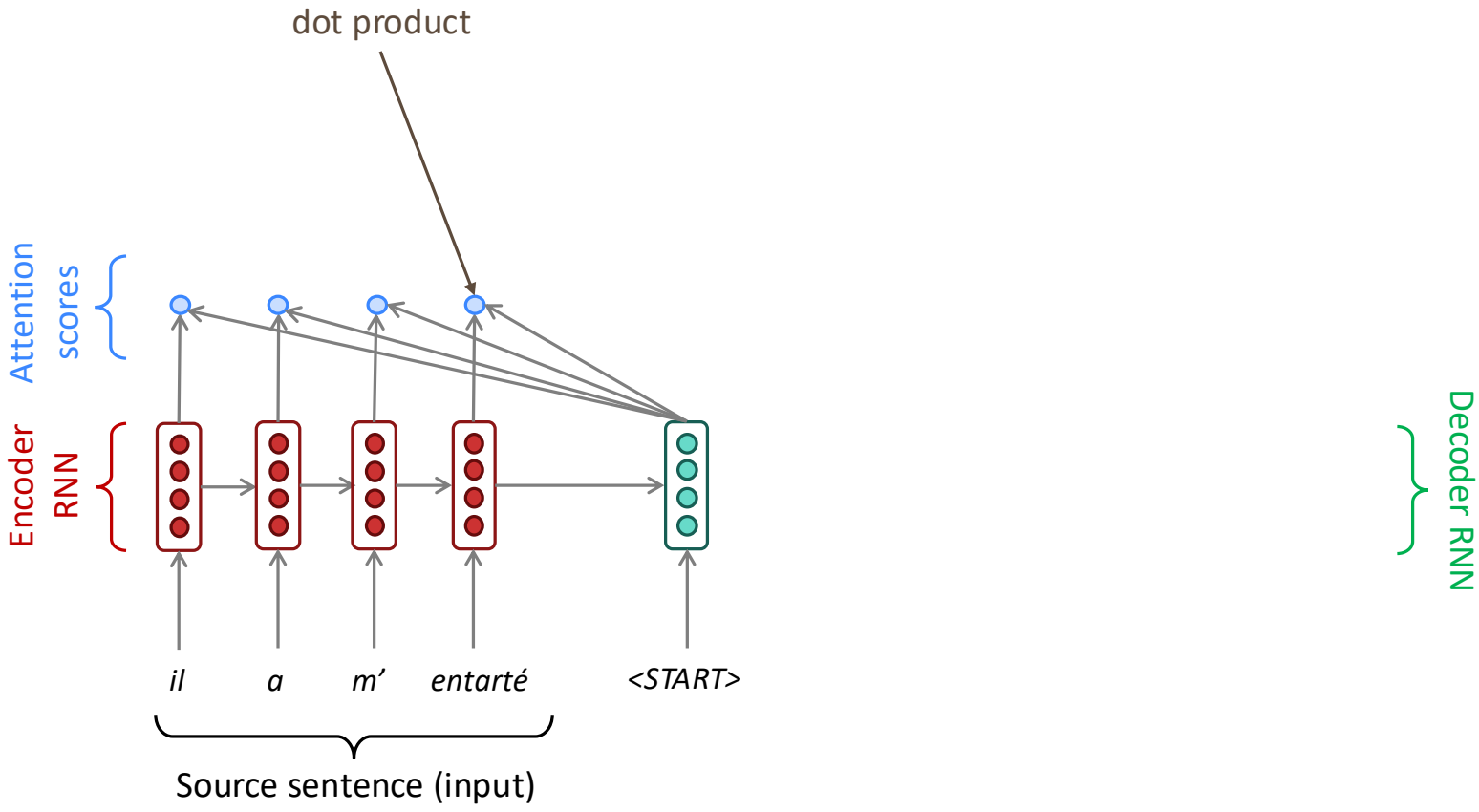
Slide from Tatsunori Hashimoto

Sequence-to-sequence with attention



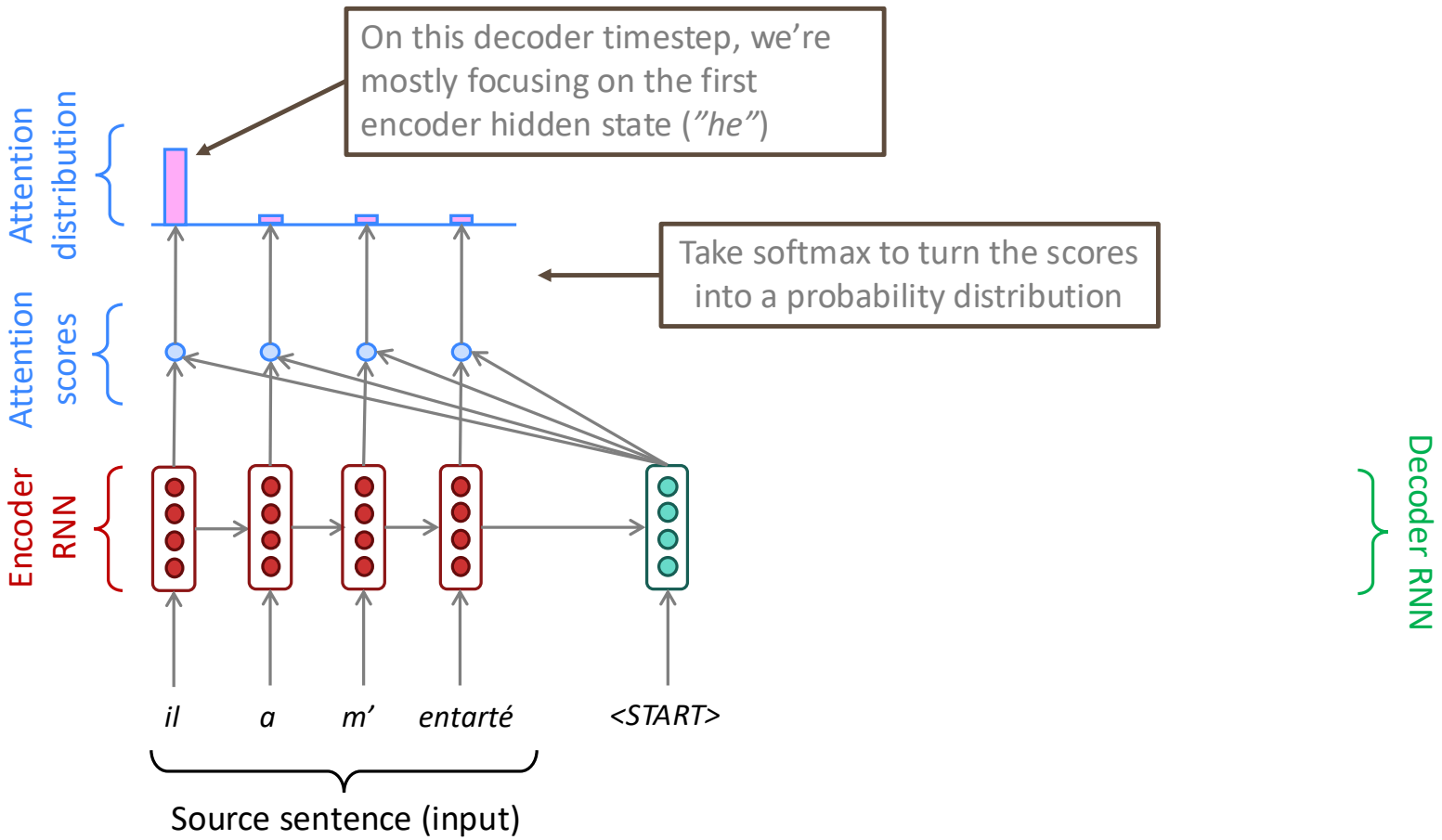
Slide from Tatsunori Hashimoto

Sequence-to-sequence with attention



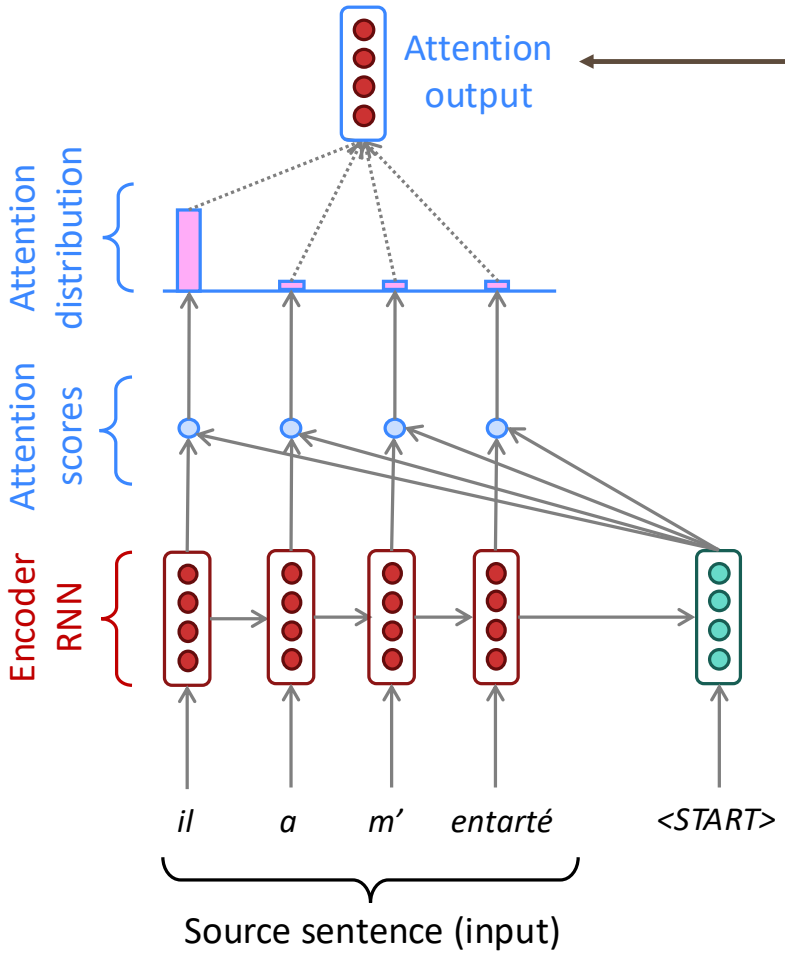
Slide from Tatsunori Hashimoto

Sequence-to-sequence with attention



Slide from Tatsunori Hashimoto

Sequence-to-sequence with attention



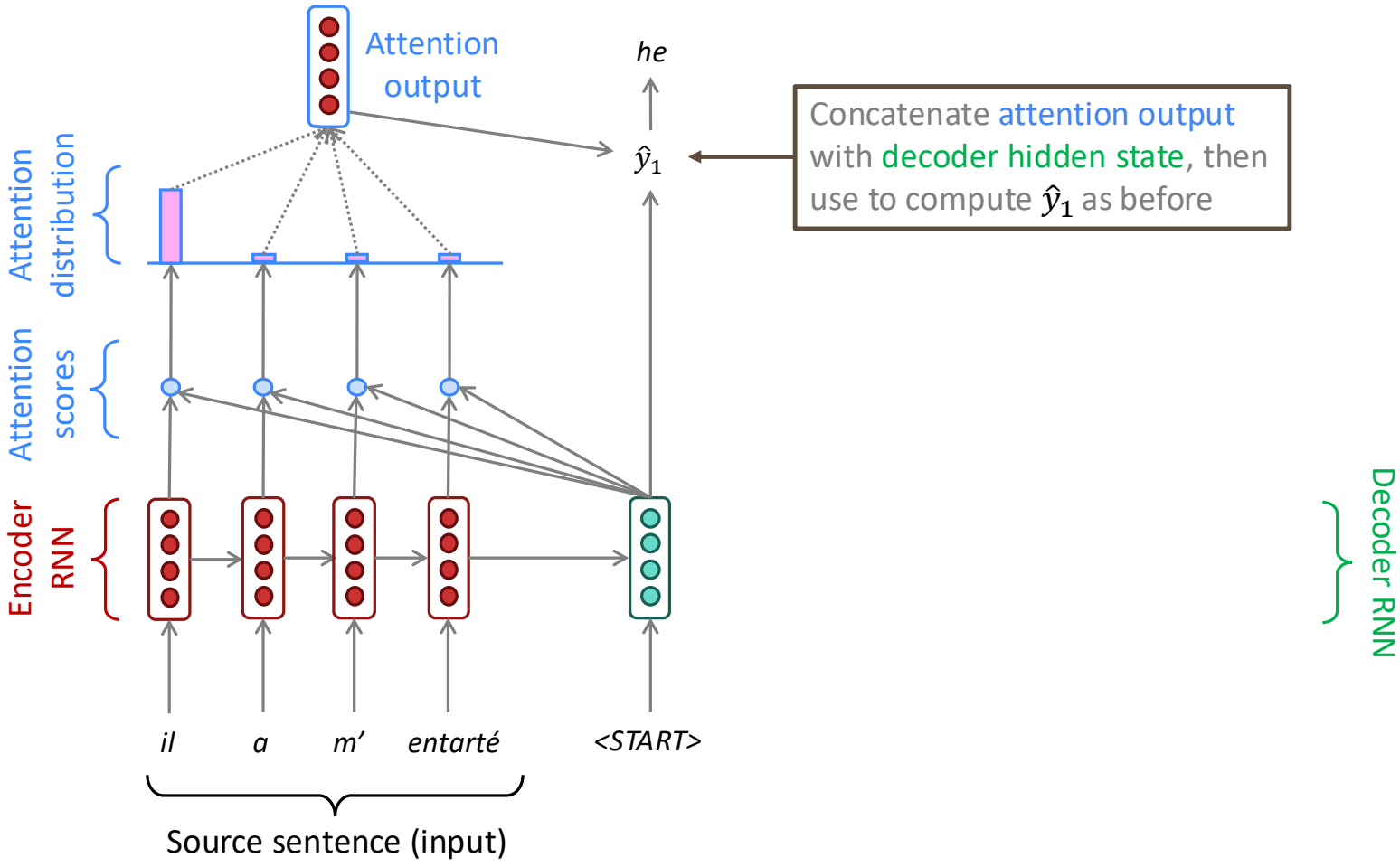
Use the attention distribution to take a **weighted sum** of the encoder hidden states.

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

Decoder RNN

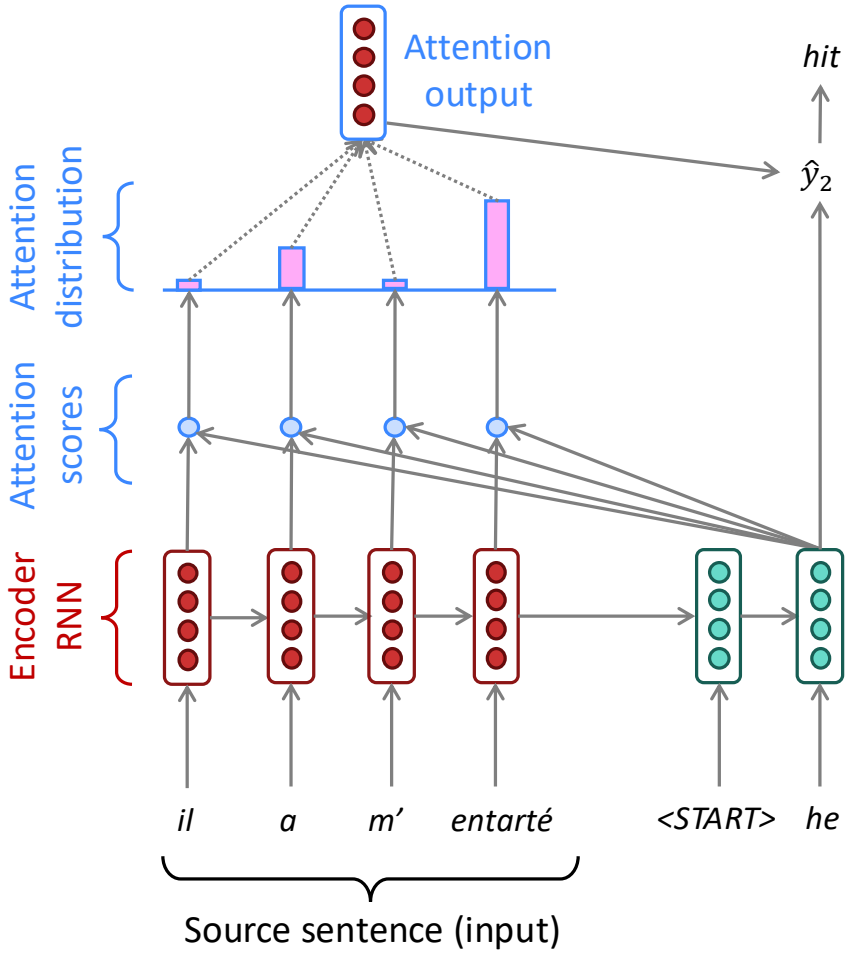
Slide from Tatsunori Hashimoto

Sequence-to-sequence with attention



Slide from Tatsunori Hashimoto

Sequence-to-sequence with attention

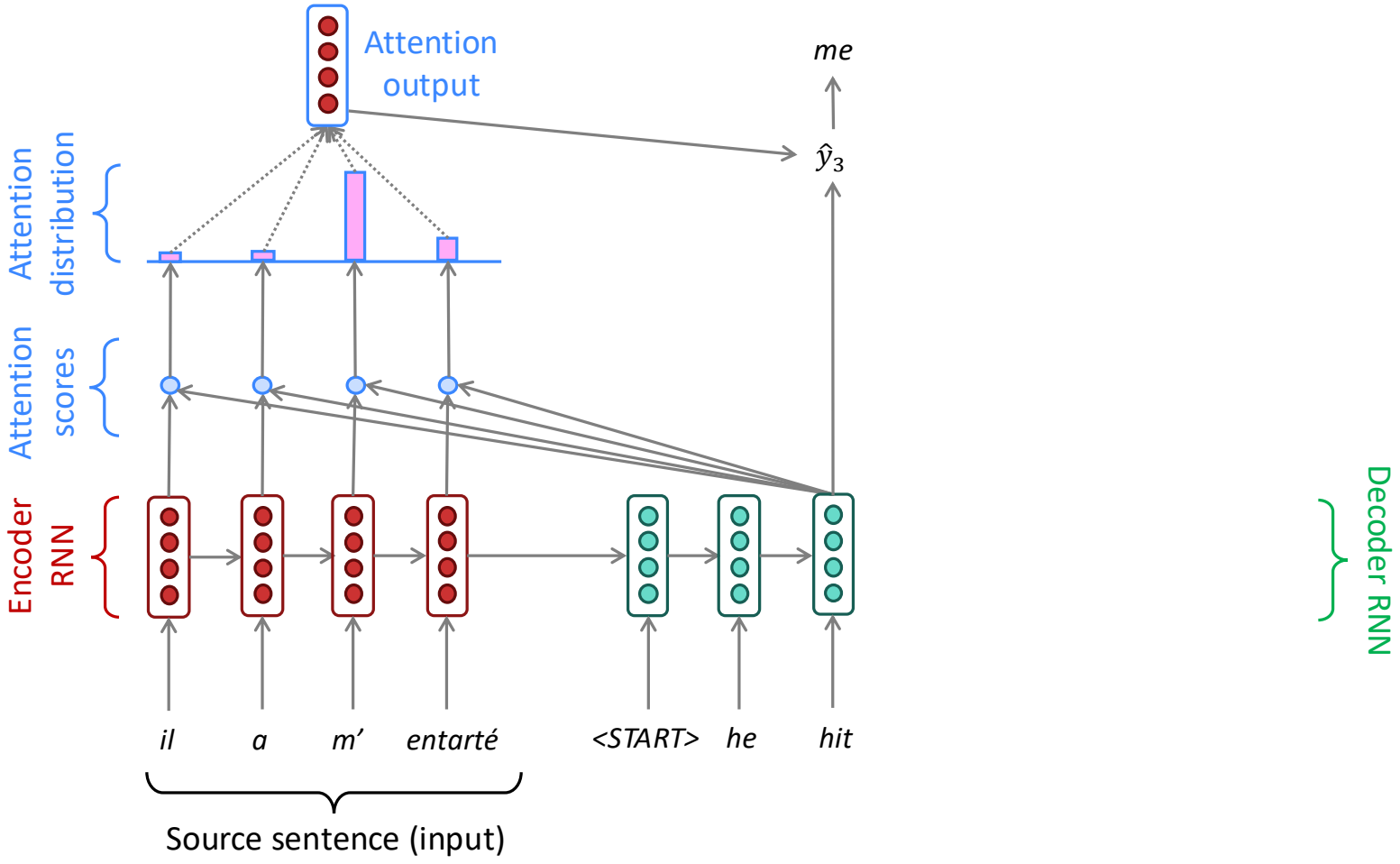


Sometimes we take the attention output from the previous step, and also feed it into the decoder (along with the usual decoder input). We do this in Assignment 4.

Decoder RNN

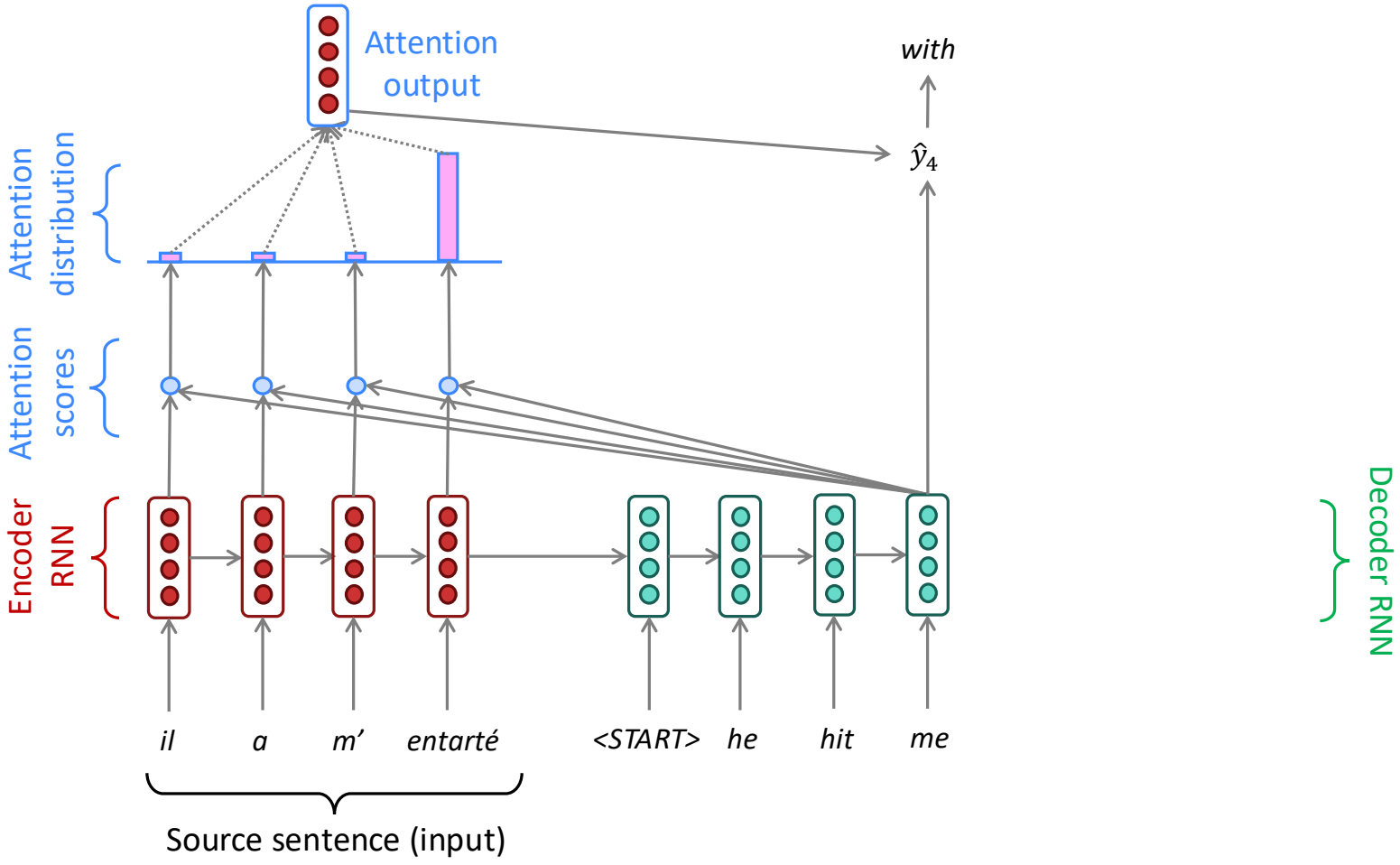
Slide from Tatsunori Hashimoto

Sequence-to-sequence with attention



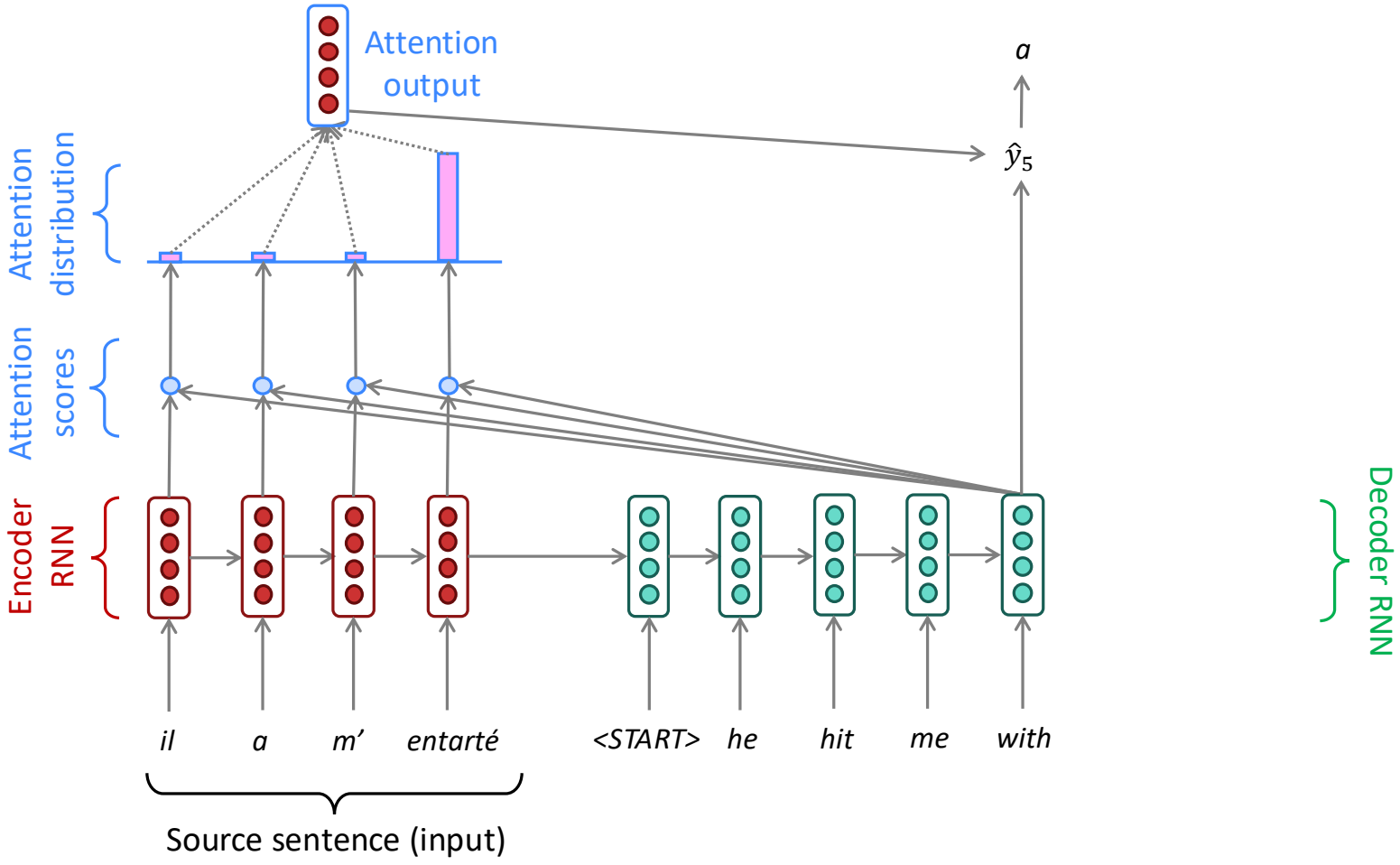
Slide from Tatsunori Hashimoto

Sequence-to-sequence with attention



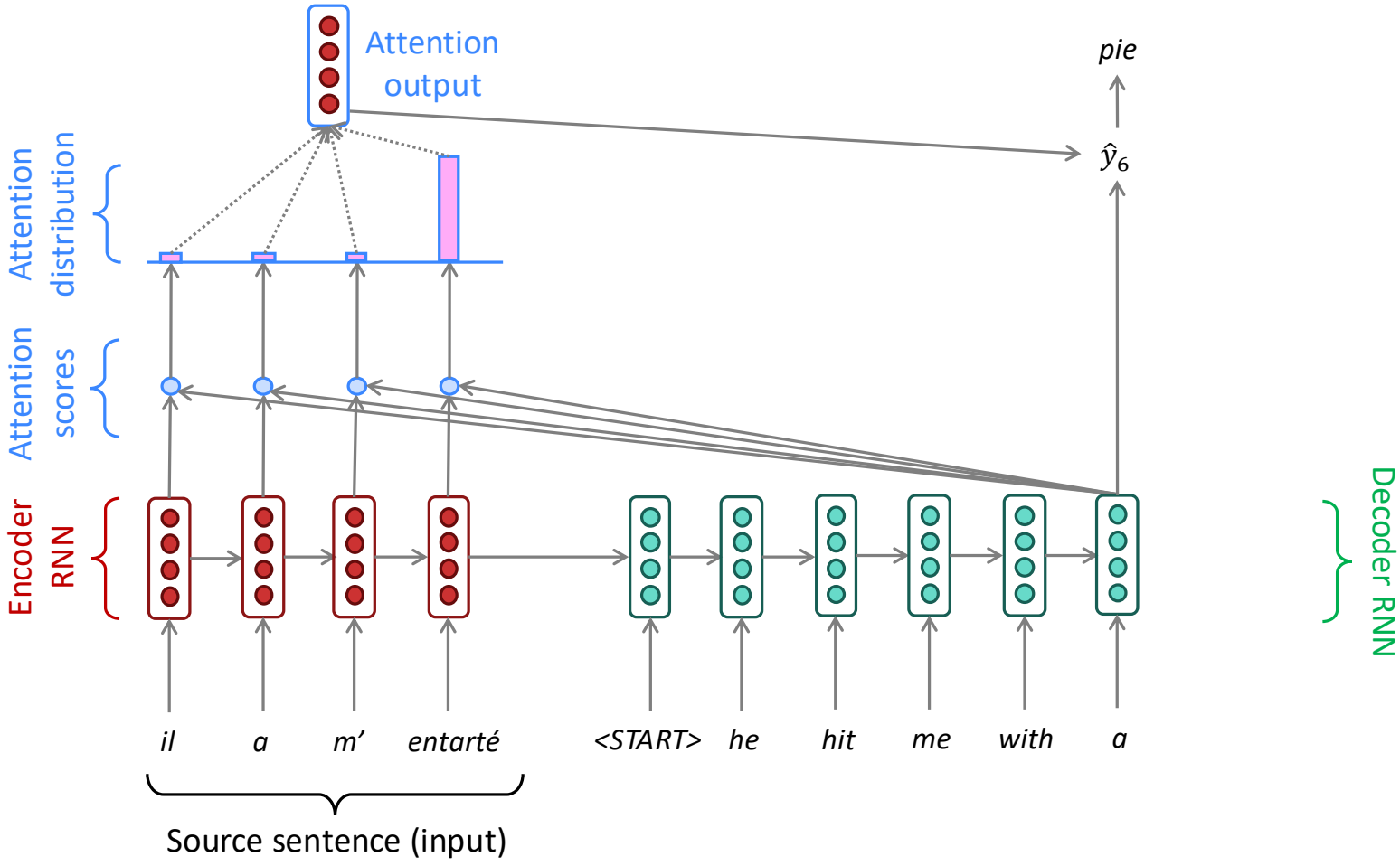
Slide from Tatsunori Hashimoto

Sequence-to-sequence with attention



Slide from Tatsunori Hashimoto

Sequence-to-sequence with attention



Slide from Tatsunori Hashimoto

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)

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

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

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

- 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

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

Slide from Tatsunori Hashimoto

Query-Key-Value Attention

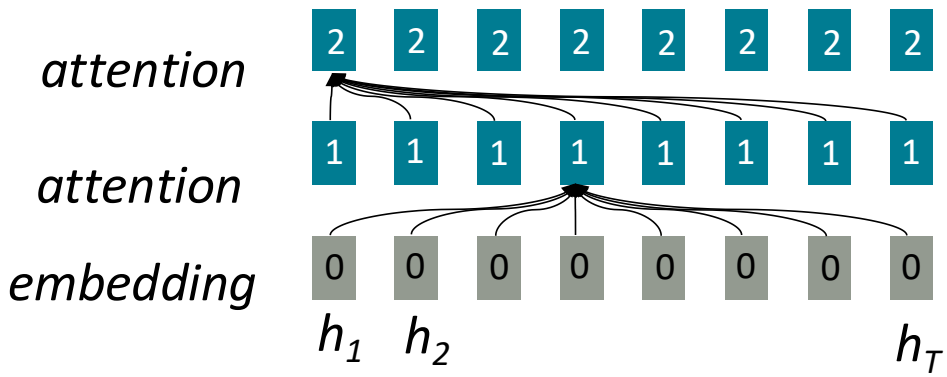
- Given a sequence-of-vectors $\langle h_1, \ldots, h_N \rangle$ and a state vector s_t,
- and three parameter matrices W^q, W^k, W^v,
- $$e_i^t = (W^q s_t)^T W^k h_i$$
  $$\alpha^t = \mathit{softmax}(e^t)$$
  $$a_t = \sum_{i=1}^{N} \alpha_i^t W^v h_i$$

Attention function:

- permutation invariant, so $\langle h_1, \ldots, h_N \rangle$ is a set
- size invariant, so $\langle h_1, \ldots, h_N \rangle$ is unbounded
- normalised weighting, so $\langle \alpha_1^t, \ldots, \alpha_N^t \rangle$ is a distribution

Attention is parallelizable, and solves bottleneck issues.

- **Attention** treats each word's representation as a **query** to access and incorporate information from **a set of values.**
  - We saw attention from the **decoder** to the **encoder**; today we'll think about attention **within a single sentence**.
- Number of unparallelizable operations does not increase with sequence length.
- Maximum interaction distance: O(1), since all words interact at every layer!

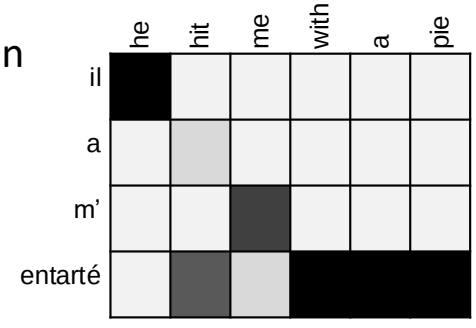


All words attend to all words in previous layer; most arrows here are omitted

Slide from Tatsunori Hashimoto

Attention is great!

- Attention significantly improves NMT performance
  - It's very useful to allow decoder to focus on certain parts of the source
- Attention provides a more "human-like" model of the MT process
  - You can look back at the source sentence while translating, rather than needing to remember it all
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with the vanishing gradient problem
  - Provides shortcut to faraway states
- Attention provides some interpretability
  - By inspecting attention distribution, we see what the decoder was focusing on
  - We get (soft) alignment for free!
  - The network just learned alignment by itself
- (One issue – attention has *quadratic* cost with respect to sequence length)



Slide from Tatsunori Hashimoto

Attention is a *general* Deep Learning technique

- We've seen that attention is a great way to improve the sequence-to-sequence model for Machine Translation.
- However: You can use attention in many architectures (not just seq2seq) and many tasks (not just MT)

- **More general definition of attention**:
  - Given a set of vector *values*, and a vector *query*, **attention** is a technique to compute a weighted sum of the values, dependent on the query.

- We sometimes say that the query *attends to* the values.
- For example, in the seq2seq + attention model, each decoder hidden state (query) *attends to* all the encoder hidden states (values).

Slide from Tatsunori Hashimoto

Attention is a *general* Deep Learning technique

- More general definition of attention:
  - Given a set of vector *values*, and a vector *query*, attention is a technique to compute a weighted sum of the values, dependent on the query.

Intuition:

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

Upshot:

- Attention has become the powerful, flexible, general way pointer and memory manipulation in all deep learning models. A new idea from after 2010! From NMT!

Slide from Tatsunori Hashimoto

Summary of Attention in NMT

- Attention in NMT learns a soft alignment between output and input tokens
- Attention uses a (non-parametric) **set-of-vector** representation, instead of a (parametric) vector representation, which is more appropriate for representing language
- Attention accesses vectors in the set based only on their content
- Attention is very effective whenever conditioning on (arbitrarily long) text