Encoder-Decoder

CS4248 Natural Language Processing

Week 08

Liangming PAN, Xinyuan LU and Min-Yen KAN

8

Recap of Week 07

Sequences – A primary form of natural language data with many applications.

The classic sequence model is the Hidden Markov Model, where a latent (unobserved) variable is key aspect of the inference.

- *What's the likelihood?* Solved by Forward
- *What's the best path?* Solved by Viterbi Decoding

Week 08 Agenda

Recap: Language Modeling

Language Modeling with RNNs

Encoder–Decoder Model

Attention Mechanism

Beam Search Decoding

Recap: Language Modeling

What are Language Models?

Language models are models that assign probabilities to a sentence.

Probability of sequence of words

$$P(W) = P(w_1, w_2, w_3, \dots, w_n)$$
$$P(\text{“please turn your homework”})$$

Probability of an upcoming word

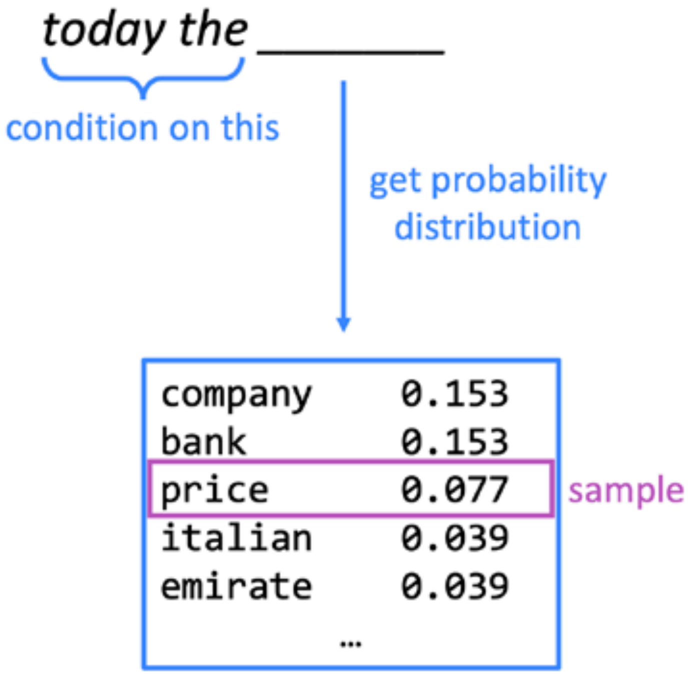
$$P(w_n | w_1, \dots, w_{n-1})$$
$$P(\text{“homework”} | \text{“please turn your”})$$

n-Gram models

Intuition: approximate the probability by looking at the n preceding words. Utilizing the Markov assumption.

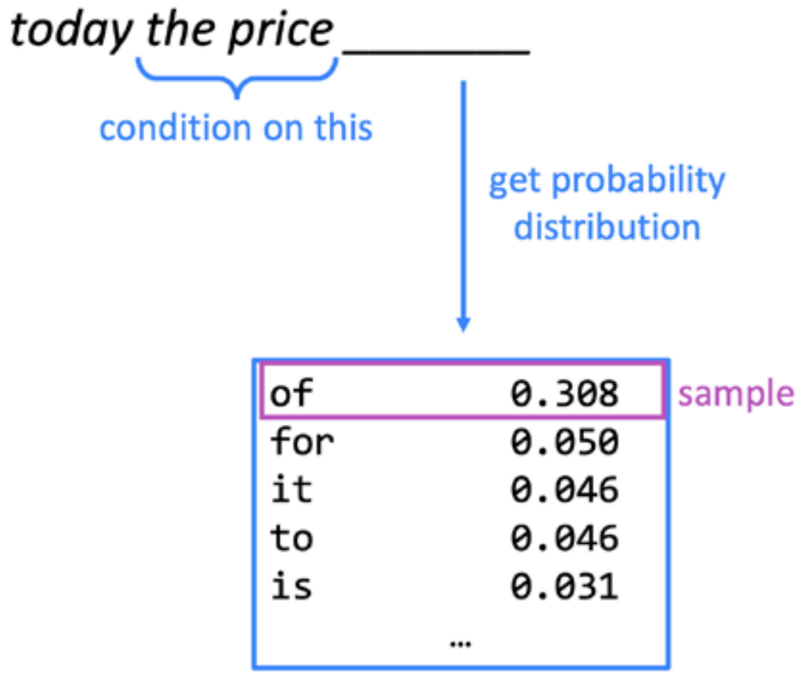
- Unigram (1-gram) : $P(A_i|A_{1:i-1}) \approx P(A_i)$
- Bigram (2-gram) : $P(A_i|A_{1:i-1}) \approx P(A_i|A_{i-1})$
- Trigram (3-gram) : $P(A_i|A_{1:i-1}) \approx P(A_i|A_{i-2}A_{i-1})$

n-Gram models



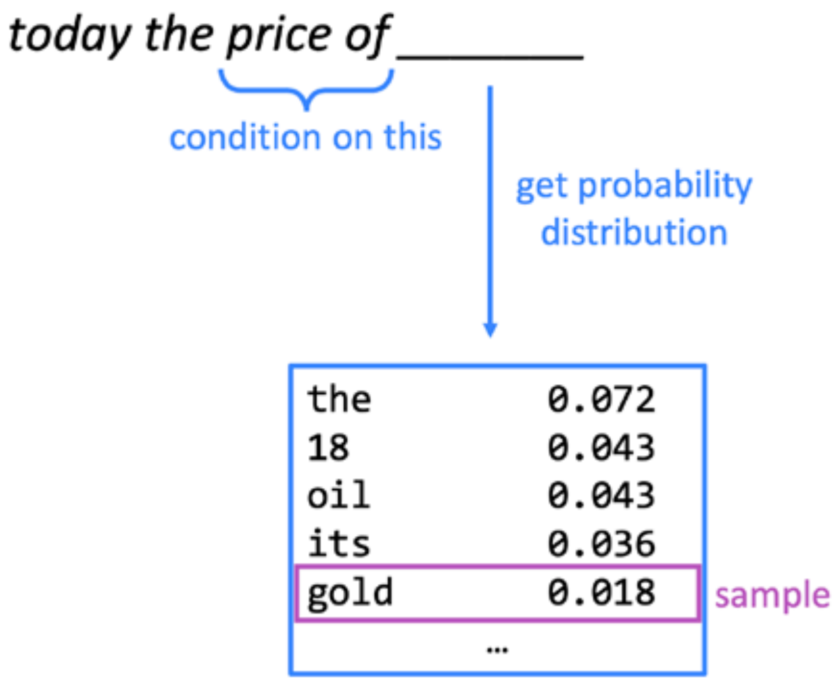
Slides adapted from Chris Manning (Stanford)

n-Gram models: Sample



Slides adapted from Chris Manning (Stanford)

n-Gram models: Sample Iteratively



Autoregressive Generation: Sample further words *conditioned on previous choices* until:

1. reaching a pre-determined length,
2. or an end of sequence token is generated.

Slides adapted from Chris Manning (Stanford)

n-Gram models

today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .

Slides adapted from Chris Manning (Stanford)

n-Gram models

today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .

Surprisingly grammatical!

...but **incoherent.** We need to consider more than three words at a time if we want to model language well.

Slides adapted from Chris Manning (Stanford)

Key shortcoming: long-distance dependencies

France is where I grew up, but I now live in Boston.

I speak fluent __________ .

We need information from the distant past to accurately predict the correct word.

Designing an ideal sequence model

To model sequences well, we need to:

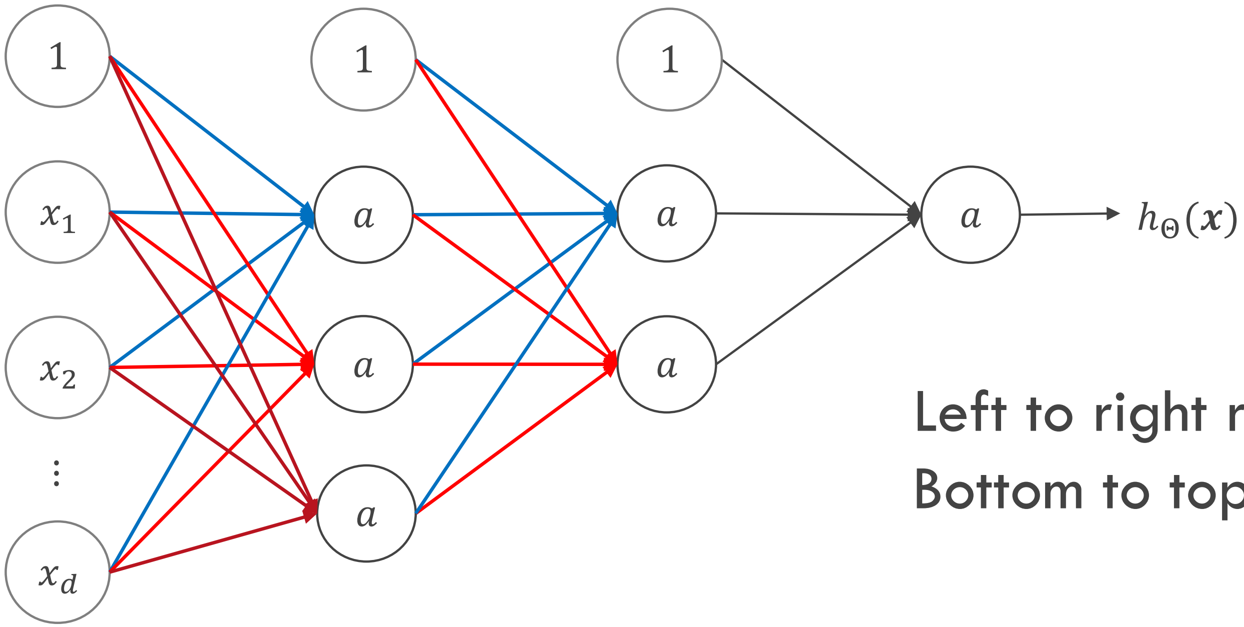
1. Handle variable-length sequences
2. Track long-distance dependencies ← The biggie!
3. Maintain information about token order
4. Share parameters across the sequence

Recurrent Neural Networks (RNNs) as a solution to this problem.

Adapted from H Suresh (MIT 6.S191)

Recurrent Neural Networks

Recap Week 05: The Neural Network

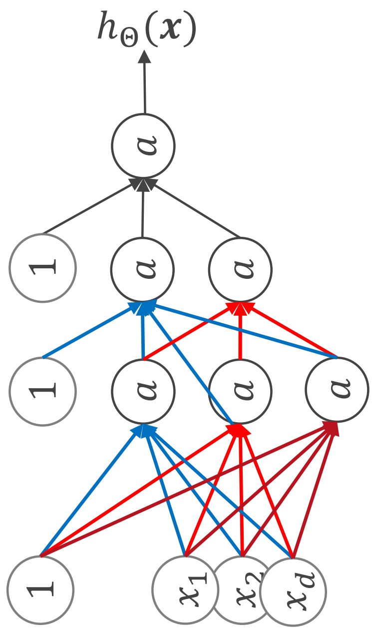


Left to right representation.
Bottom to top also common.

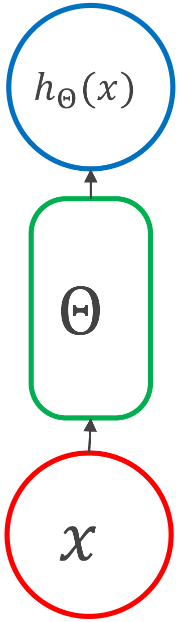
Input $\boldsymbol{x}$ Hidden layers $1 \leq l < L$ Output layer $l = L$

Slide Credits: CS3244

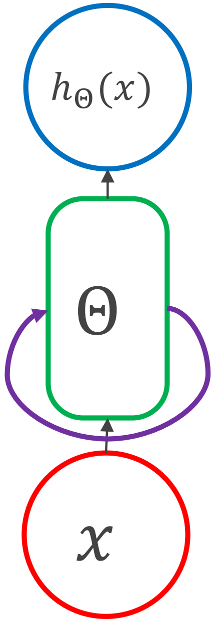
Abstracting the Neural Network



Recurrence: Adding Serial Dependencies



Standard, non-recurrent NN

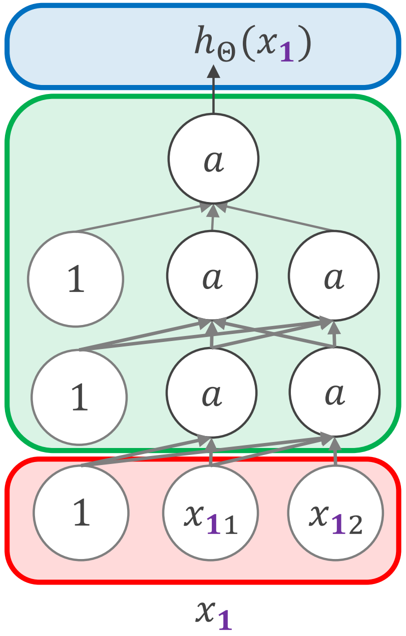


Recurrent NN

Adopted from section: Fei-Fei Li, Justin Johnson and Serena Yeung (Stanford CS231n)

Unrolling the Recurrence

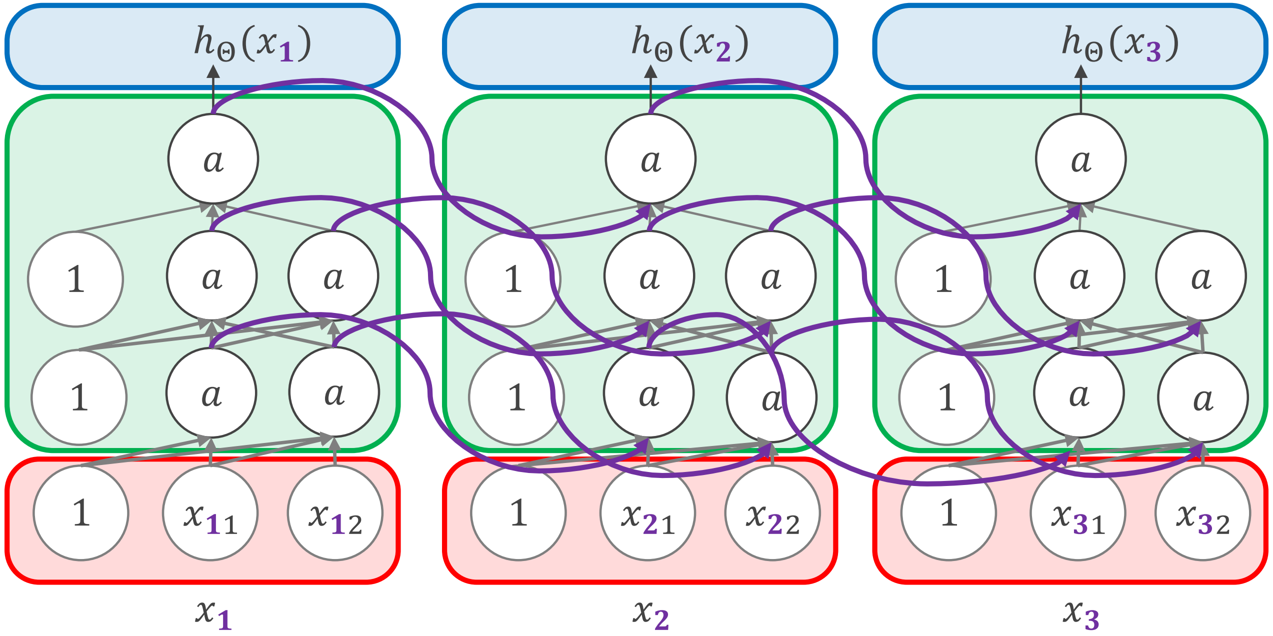
(Usually) adding an edge between a unit and a copy of itself at the next time step.



Adopted from section: Fei-Fei Li, Justin Johnson and Serena Yeung (Stanford CS231n)

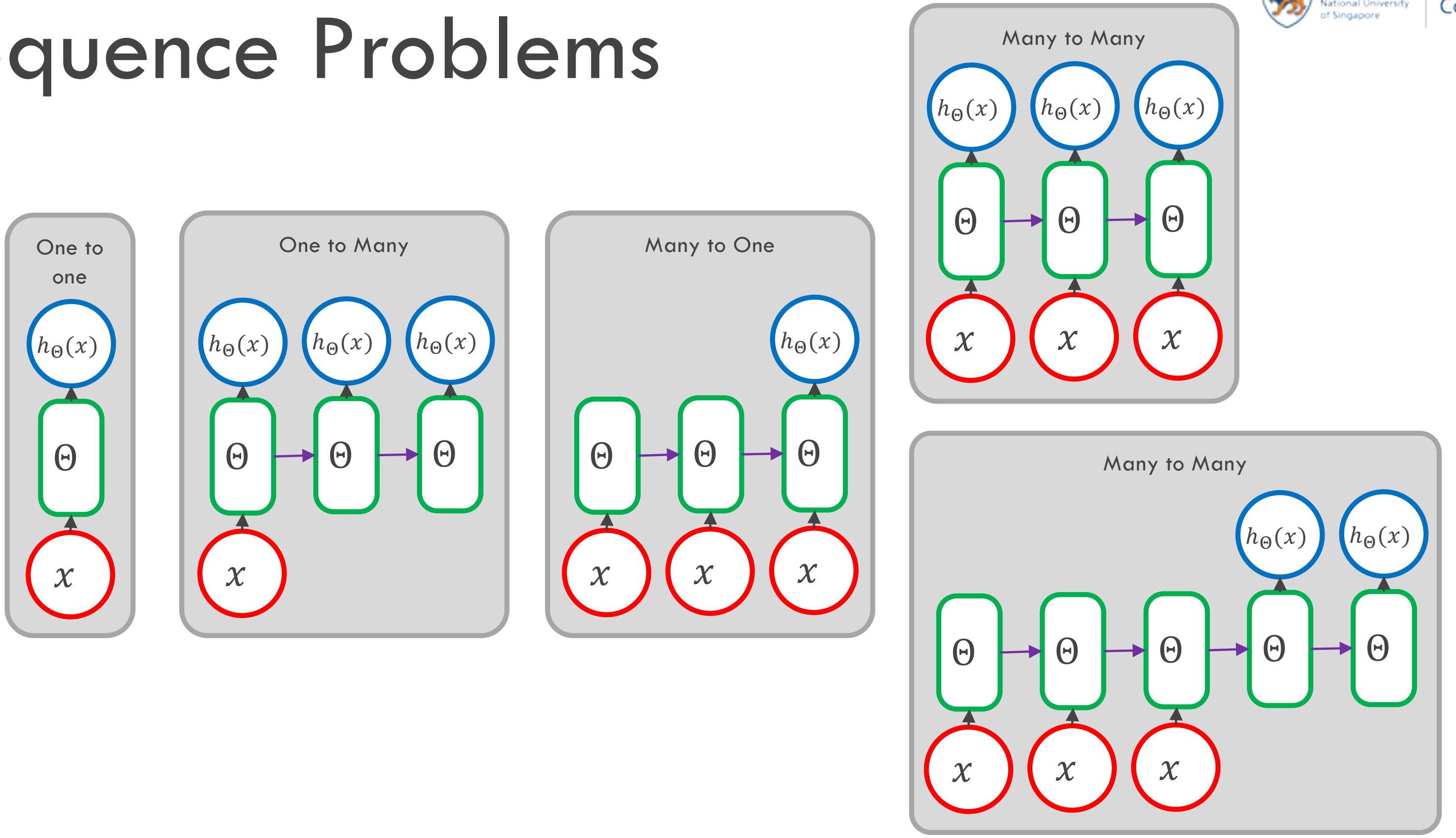
Unrolling the Recurrence

(Usually) adding an edge between a unit and a copy of itself at the next time step.



Adopted from section: Fei-Fei Li, Justin Johnson and Serena Yeung (Stanford CS231n)

Sequence Problems



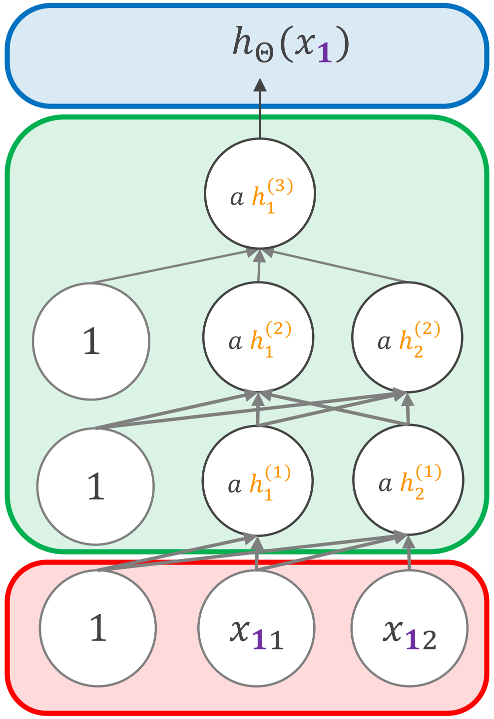
Adopted from section: Fei-Fei Li, Justin Johnson and Serena Yeung (Stanford CS231n)

Recurrent NNs keep state

In addition to the weights Θ in the network, an RNN incorporates a (hidden) state h, as a vector.

Each unit in the network has a respective dimension of h (unit state).

Randomly initialized, and to be tuned through training (backpropagation).

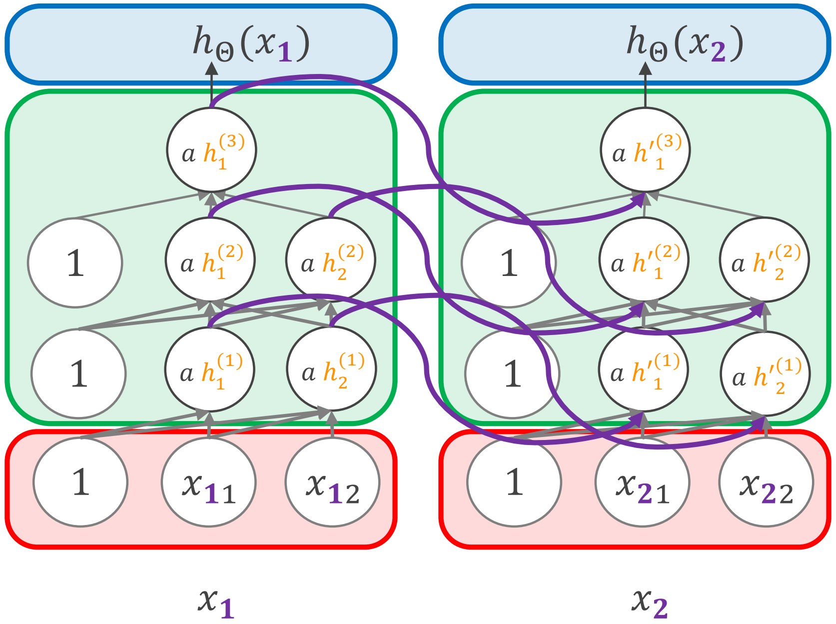


Recurrent NNs keep state

We process a sequence of inputs x by applying the recurrence at each time step.

$$h' = g_{\Theta}(h, x')$$

Using a single set of Θ for all time steps.



One state, three weights

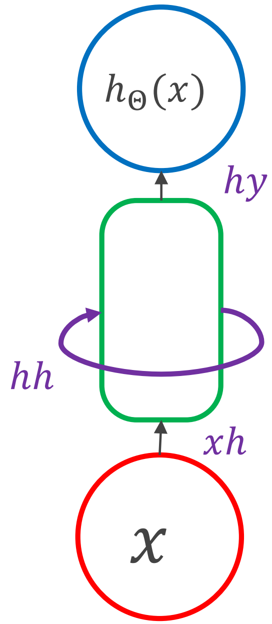
We then need individual weight matrices to represent each of the three sets of edges.

$h' = g_{\Theta}(h, x')$

$$h' = g(\Theta_{hh} h + \Theta_{xh} x')$$

$$\hat{y}' = \Theta_{hy} h'$$

Notation: you'll see both Θ and $W + b$.



Contrast: History-based LMs (Week 03)

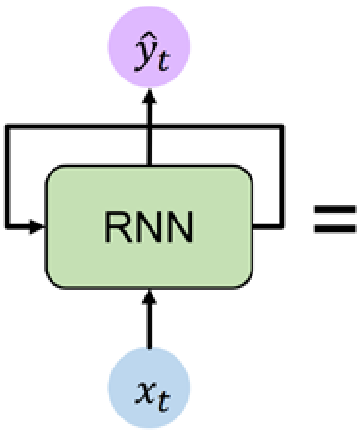
$$
\begin{aligned}
P(\boldsymbol{w}) \approx\ & P(w_1)\times \\
& P(w_2|w_1)\times \\
& P(w_3|w_1, w_2)\times \\
& P(w_4|w_1, w_2, w_3)\times \\
& \ldots
\end{aligned}
$$

Why RNNs are great for NLP: no more Markov assumptions!

We have state h'. *"Bye bye bigrams"*

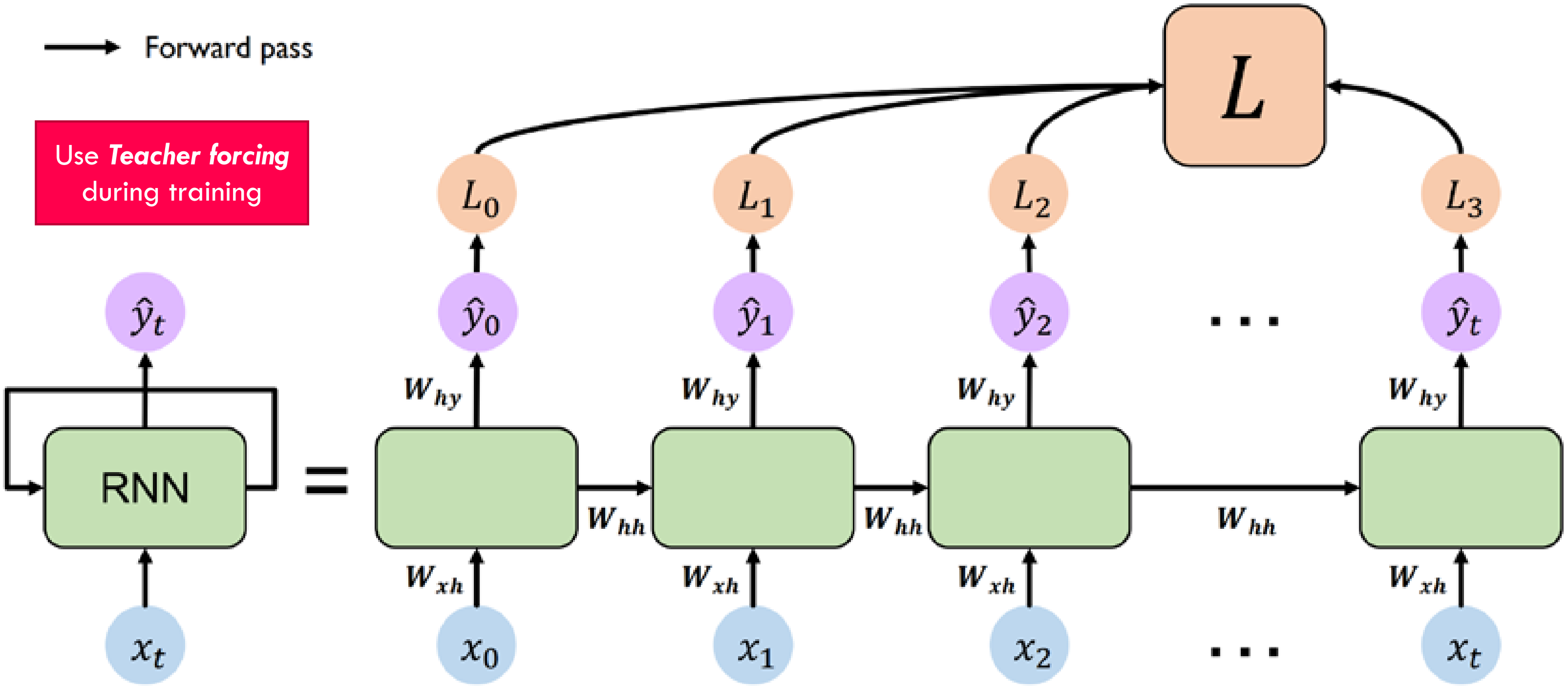
Forward Computation in Time

→ Forward pass



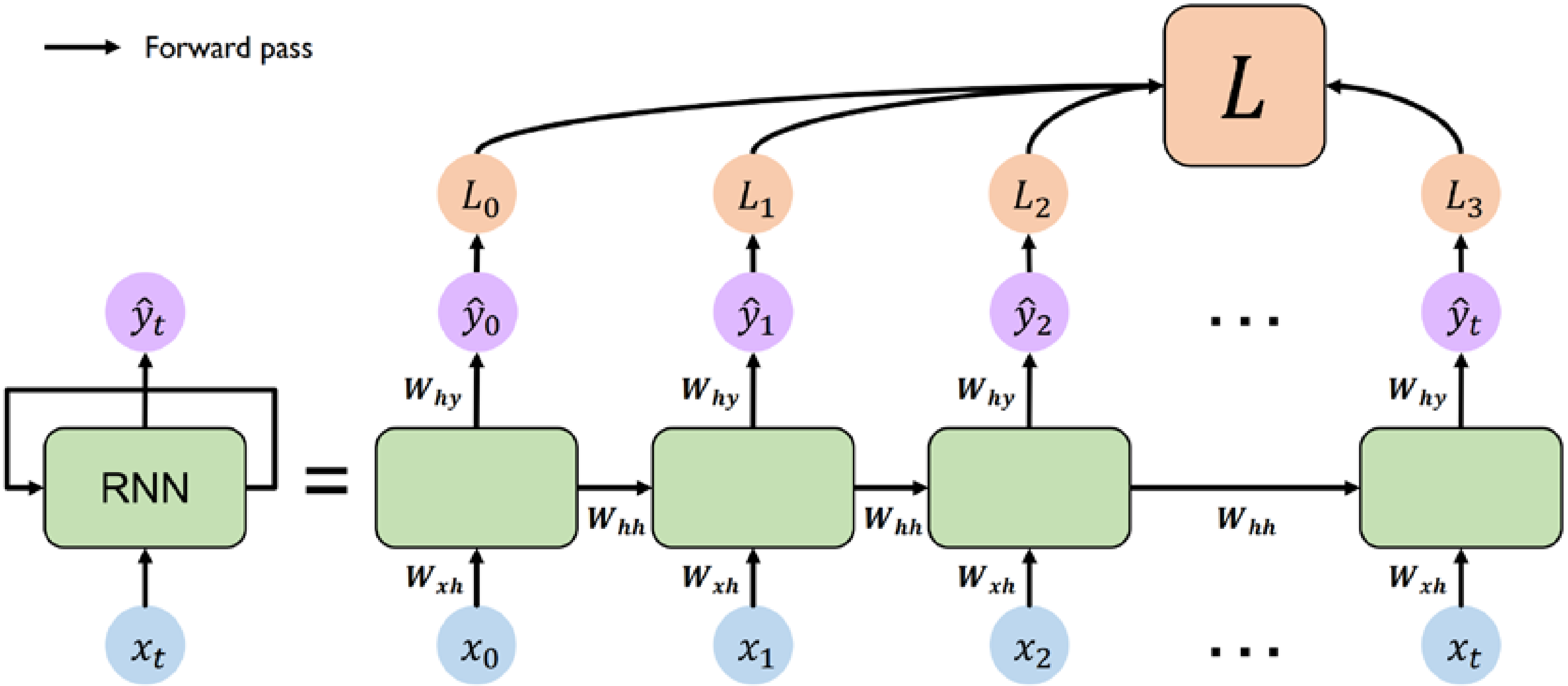
Slide Credit: MIT 6.S191 Intro to Deep Learning

Forward Computation in Time



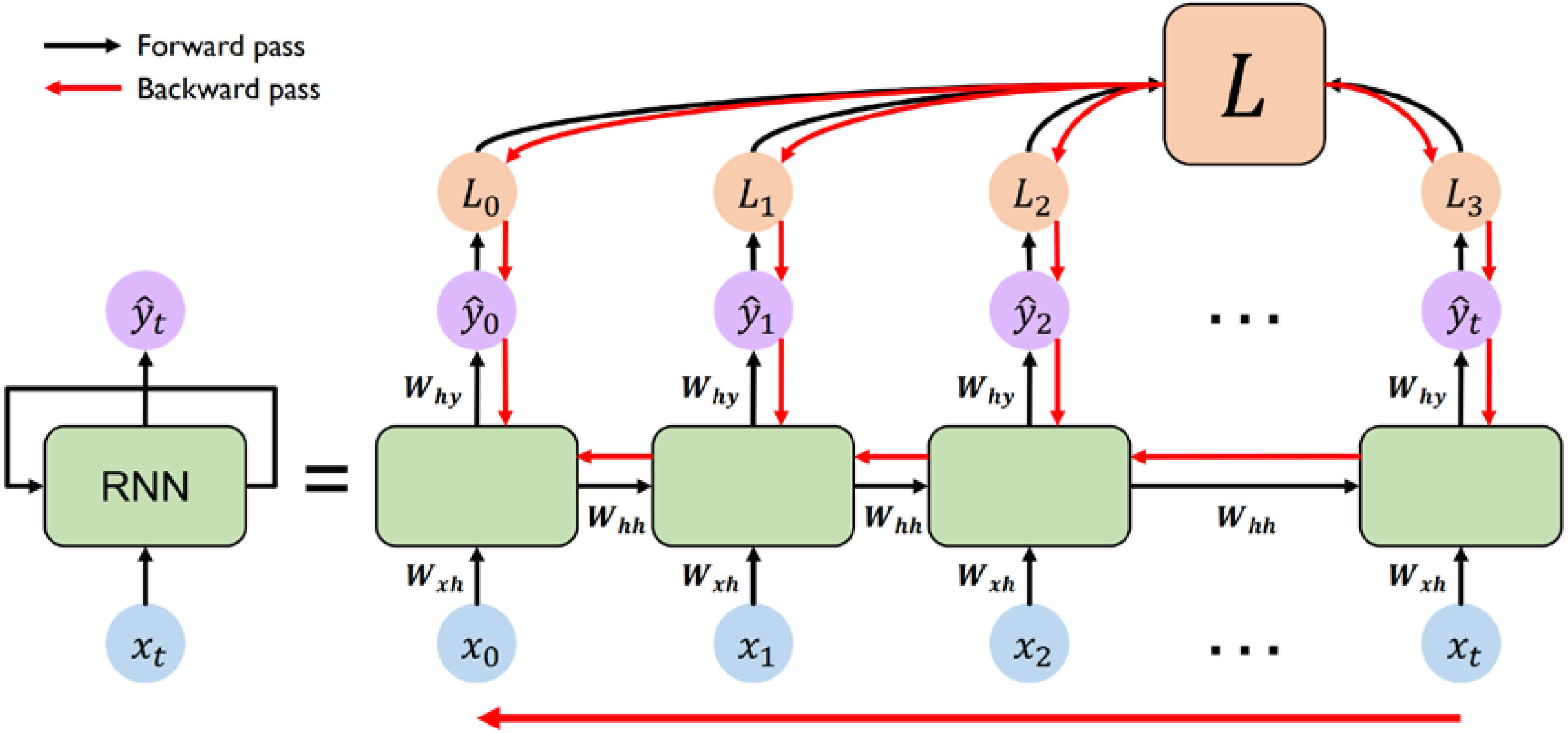
Slide Credit: MIT 6.S191 Intro to Deep Learning

Loss Backprogated through time (BPTT)

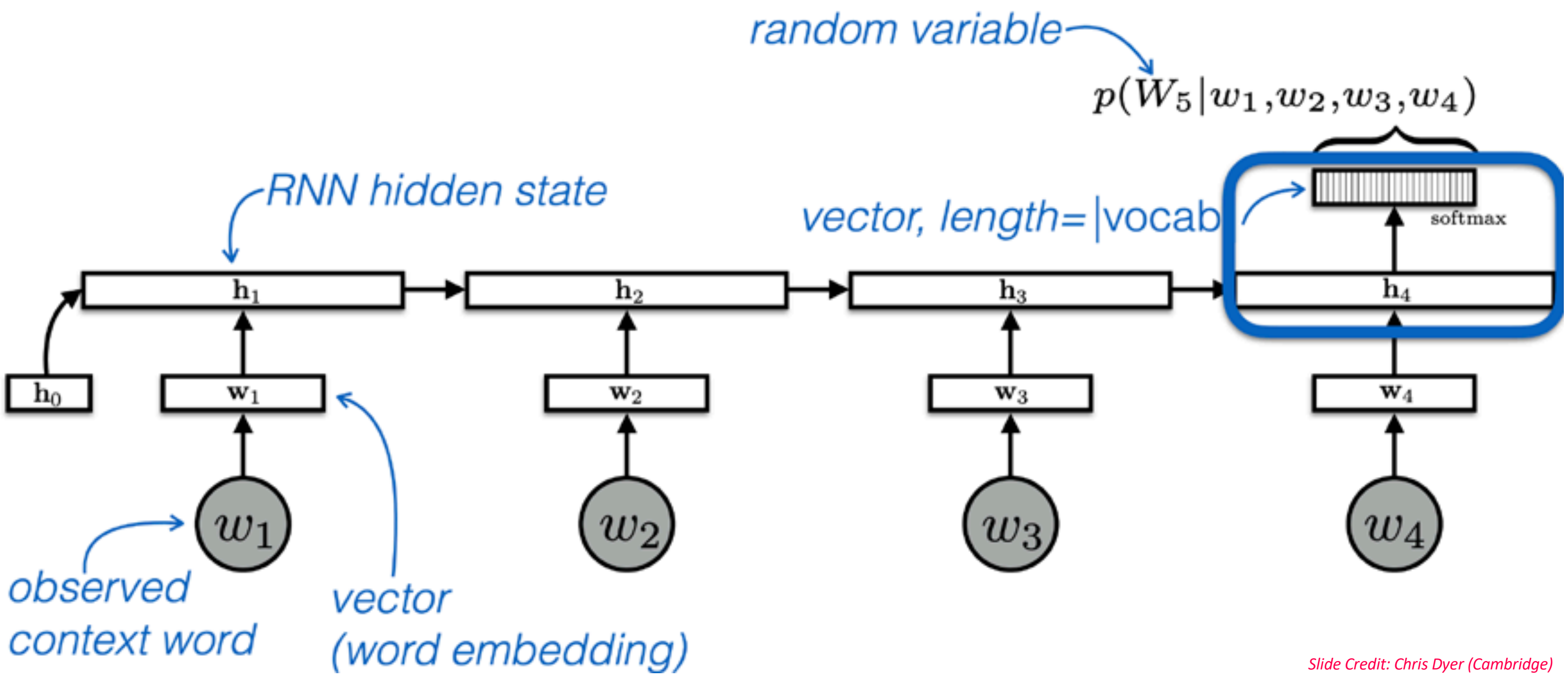


Slide Credit: MIT 6.S191 Intro to Deep Learning

Loss Backprogated through time (BPTT)



Slide Credit: MIT 6.S191 Intro to Deep Learning

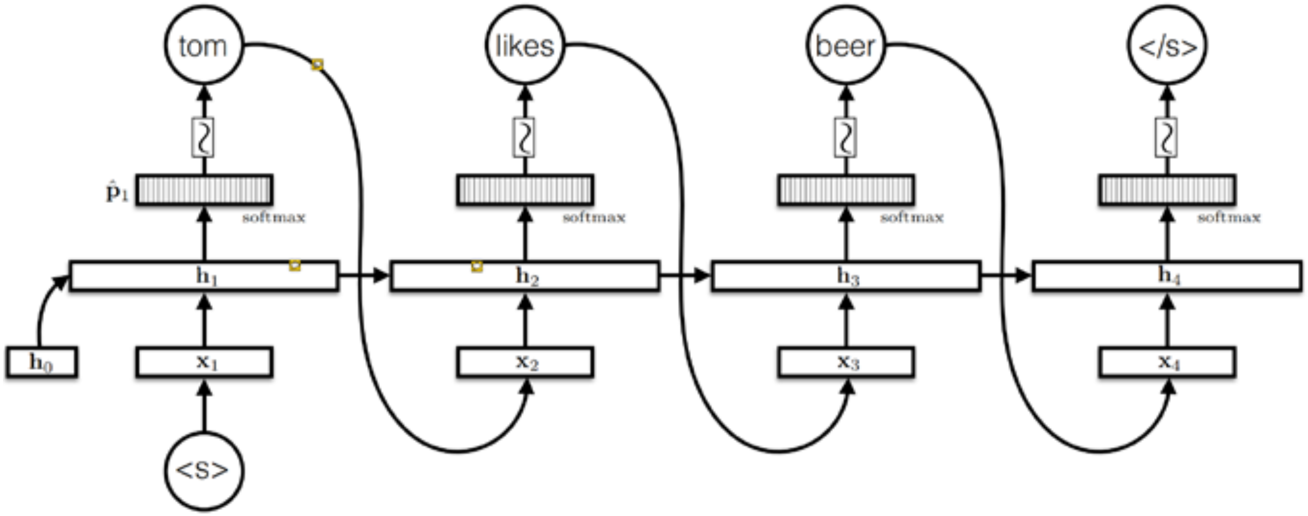


Slide Credit: Chris Dyer (Cambridge)

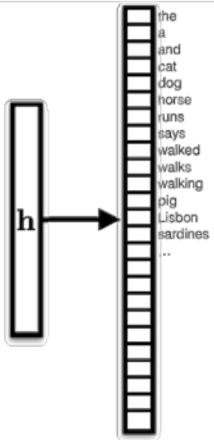
RNN for Language Modeling

Check your understanding: Not Markovian. Why is that?

$$p(tom \mid \langle \mathbf{s} \rangle) \times p(likes \mid \langle \mathbf{s} \rangle, tom)$$
$$\times p(beer \mid \langle \mathbf{s} \rangle, tom, likes)$$
$$\times p(\langle / \mathbf{s} \rangle \mid \langle \mathbf{s} \rangle, tom, likes, beer)$$



Slide Credit: Chris Dyer (Cambridge)



softmax

$$\mathbf{u} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

$$p_i = \frac{\exp u_i}{\sum_j \exp u_j}$$

$\mathbf{h} \in \mathbb{R}^d$

$|V| = 100{,}000$

$$p(\boldsymbol{w}) = p(w_1) \times$$
$$p(w_2 \mid w_1) \times$$
$$p(w_3 \mid w_1, w_2) \times$$
$$p(w_4 \mid w_1, w_2, w_3) \times$$

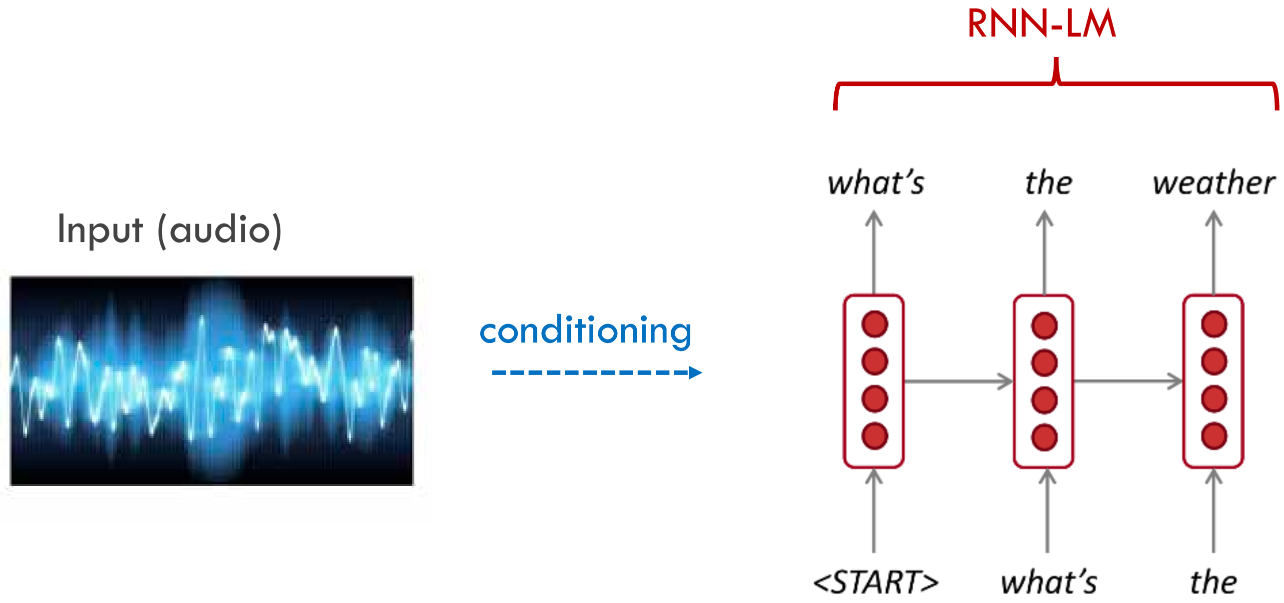
histories are sequences of words…

Slide Credit: Chris Dyer (Cambridge)

Conditional LMs

Conditioning our sequential LM

Conditional Generation



This is an example of a *conditional language model.*

Conditional LMs

A conditional language model assigns probabilities to sequences of words, $\boldsymbol{w} = (w_1, w_2, \dots, w_m)$, given some conditioning context, $\boldsymbol{x}$.

As with unconditional models, it is again helpful to use the chain rule to decompose this probability:

$$P(\boldsymbol{w}|\boldsymbol{x}) = \prod_{t=1}^{T} P(w_t|\boldsymbol{x}, w_1, w_2, \dots, w_{t-1})$$

i.e., What is the probability of the next word, given the history of previously generated words and conditioning context $\boldsymbol{x}$?

Slide Credit: Chris Dyer (Cambridge)

Conditional LMs

x "input"	w "**text** output"
An author	A document written by that author
A topic label	An article about that topic
An email	{SPAM, NOT_SPAM}
A sentence in French	Its English translation
A sentence in English	Its French translation
A sentence in English	Its Chinese translation
An image	A text description of the image
A document	Its summary
A document	Its translation
Meteorological measurements	A weather report
Acoustic signal	Transcription of speech
Conversational history + database	Dialogue system response
A question + a document	Its answer
A question + an image	Its answer

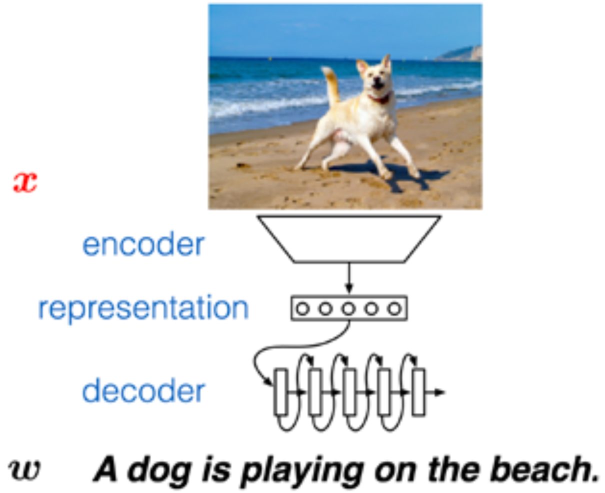
Slide Credit: Chris Dyer (Cambridge)

Encoder–Decoder

Putting RNNs to work for Conditional LMs

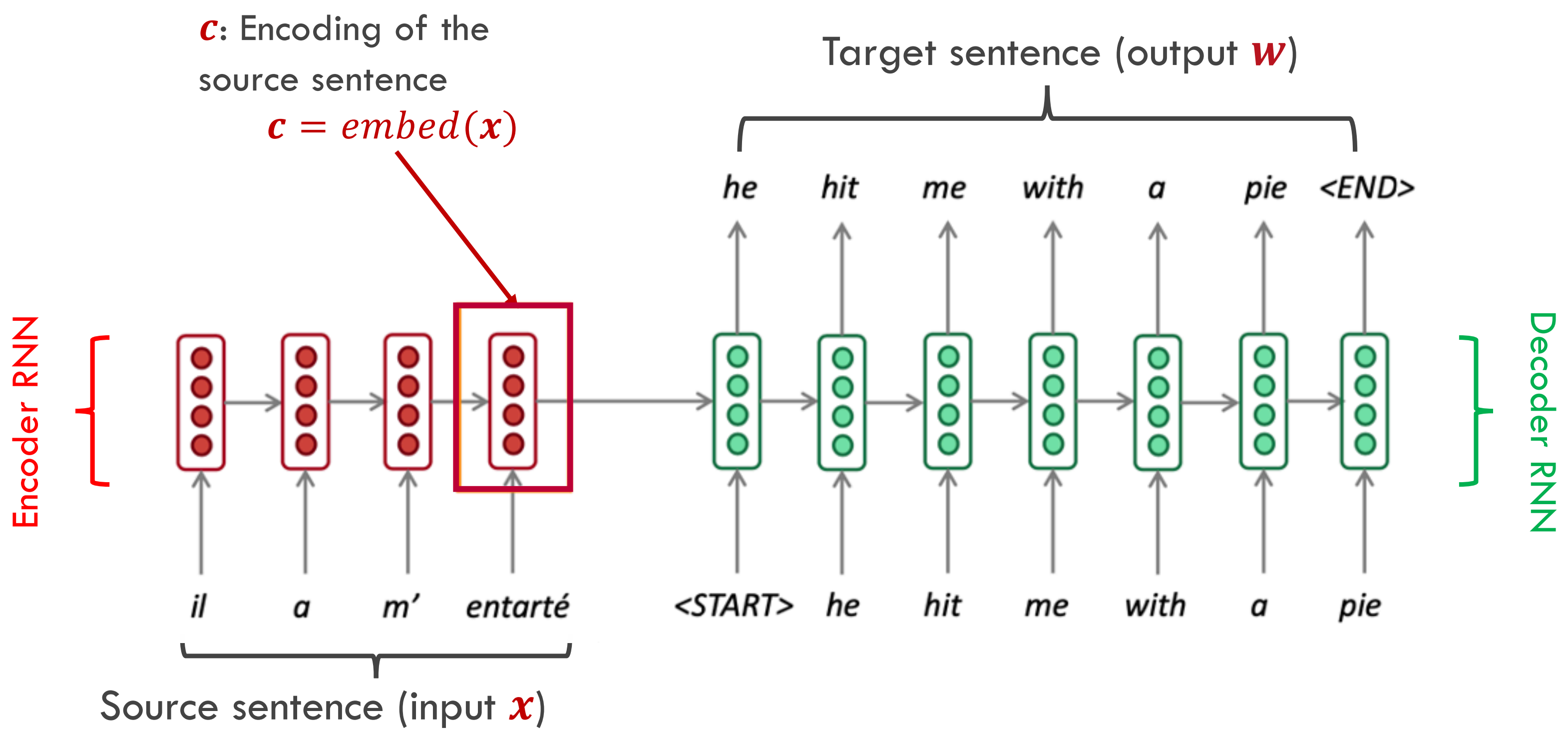
Encoder–Decoder

Models that learn a function that maps x into a fixed-sized vector representation c and then uses a language model to "decode" that vector into a sequence of output words w.



Slide Credit: Chris Dyer (Cambridge)

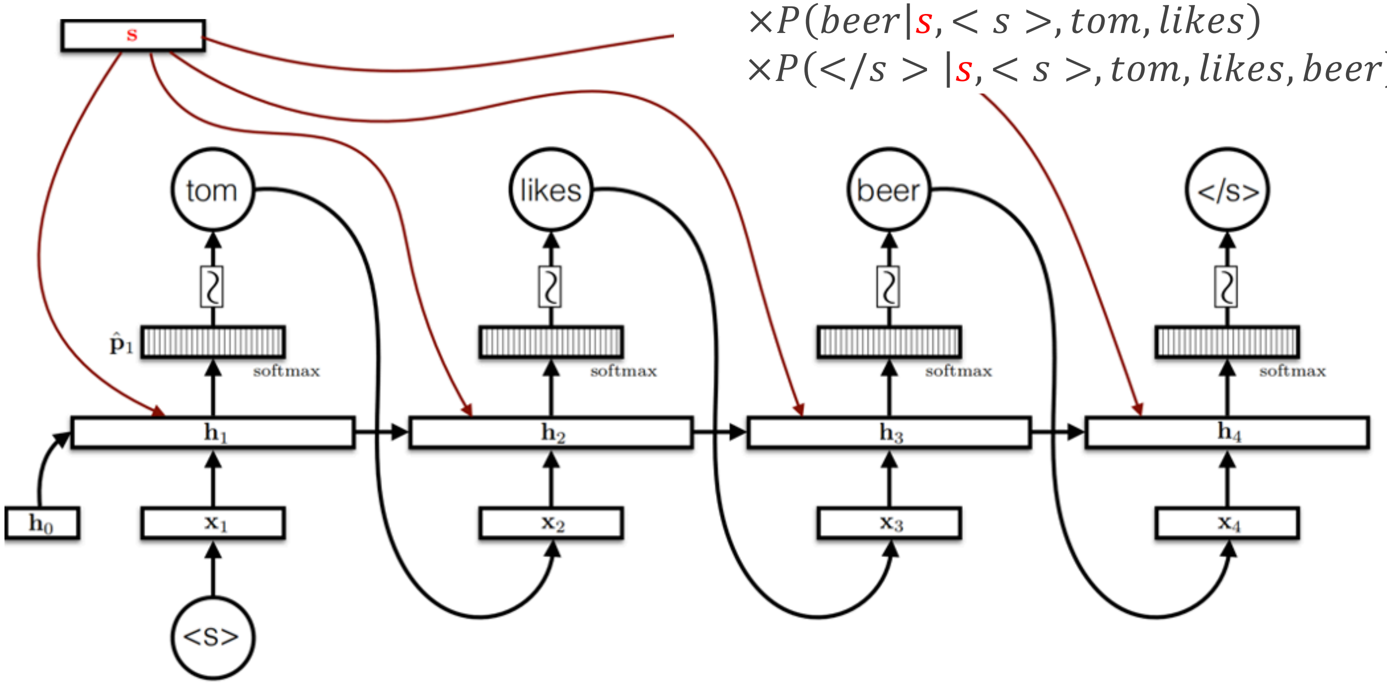
When conditioning on a sequence…



Adaptedd from Chris Manning (Stanford) CS224N

Alternative, inject extra input *s*

$$P(tom|s, < s >) \times P(likes|s, < s >, tom)$$
$$\times P(beer|s, < s >, tom, likes)$$
$$\times P(</s > |s, < s >, tom, likes, beer)$$



Click to edit Master Attribution style.

Encoder–Decoder

Encoder:

$$c = embed\ (x)$$

$$s = V_c$$

Recurrent Decoder:

Recurrent connection

Embedding of output w_{t-1}

$$h_t = g\ (\Theta[h_{t-1}; w_{t-1}] + s + b)$$

Learnt bias

Source sentence

$$u_t = \Theta_{hy}[h_t + b']$$

$$P(w_t | x, w_{<t}) = softmax(u_t)$$

Compare against the vanilla RNN:

$$h_t = g\ (W[h_{t-1}; W_{t-1}] + b)$$

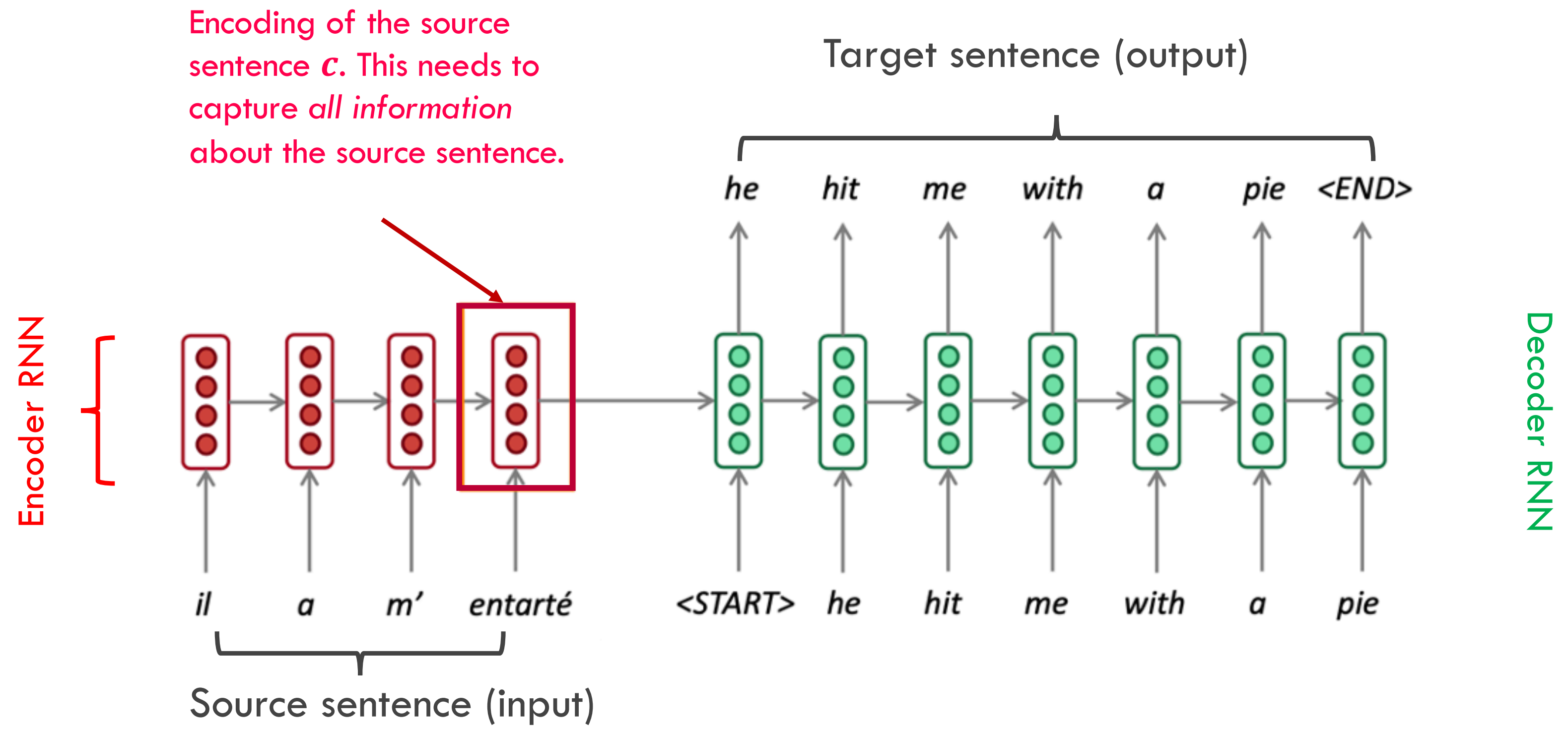
Click to edit Master Attribution style.

Attention Mechanism

Focus on what's important for the current decision

Slide Credits: Abigail See (Stanford) CS224N

Encoding c as an Information Bottleneck



Slide Credits: Abigail See (Stanford) CS224N

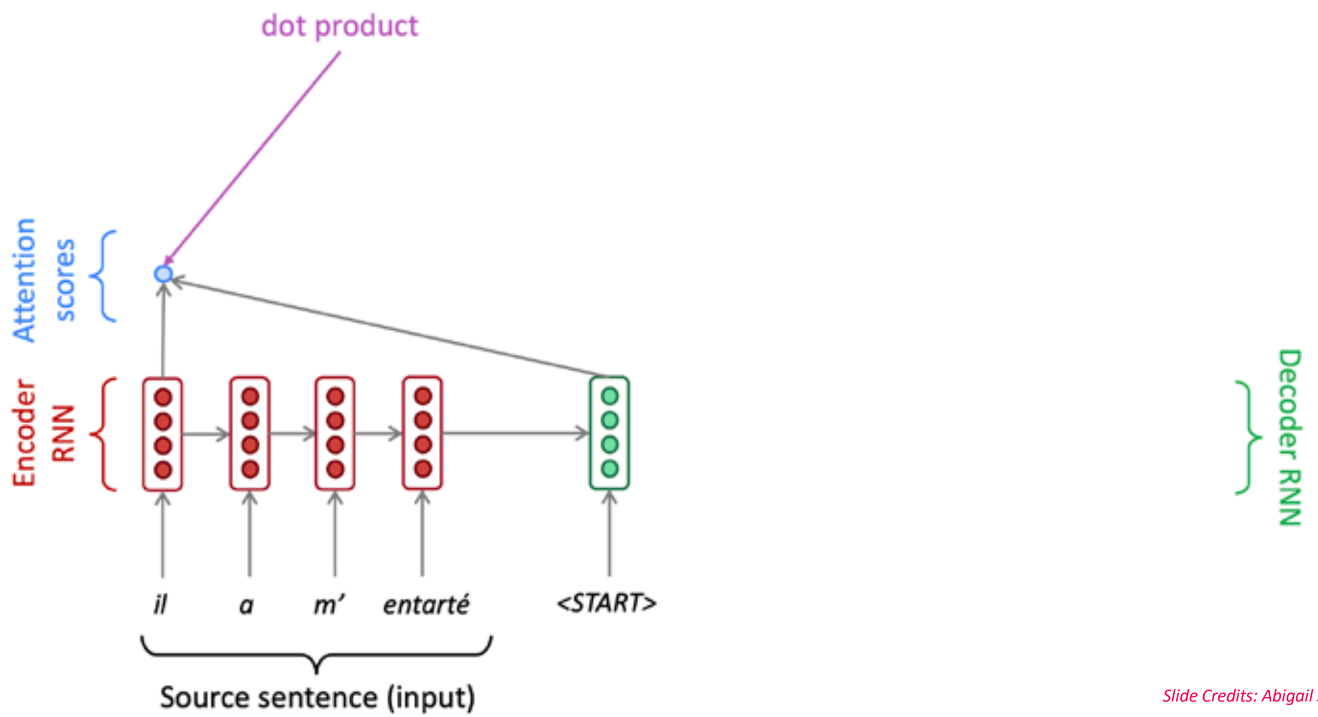
Attention

Attention provides a solution to the bottleneck problem.

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

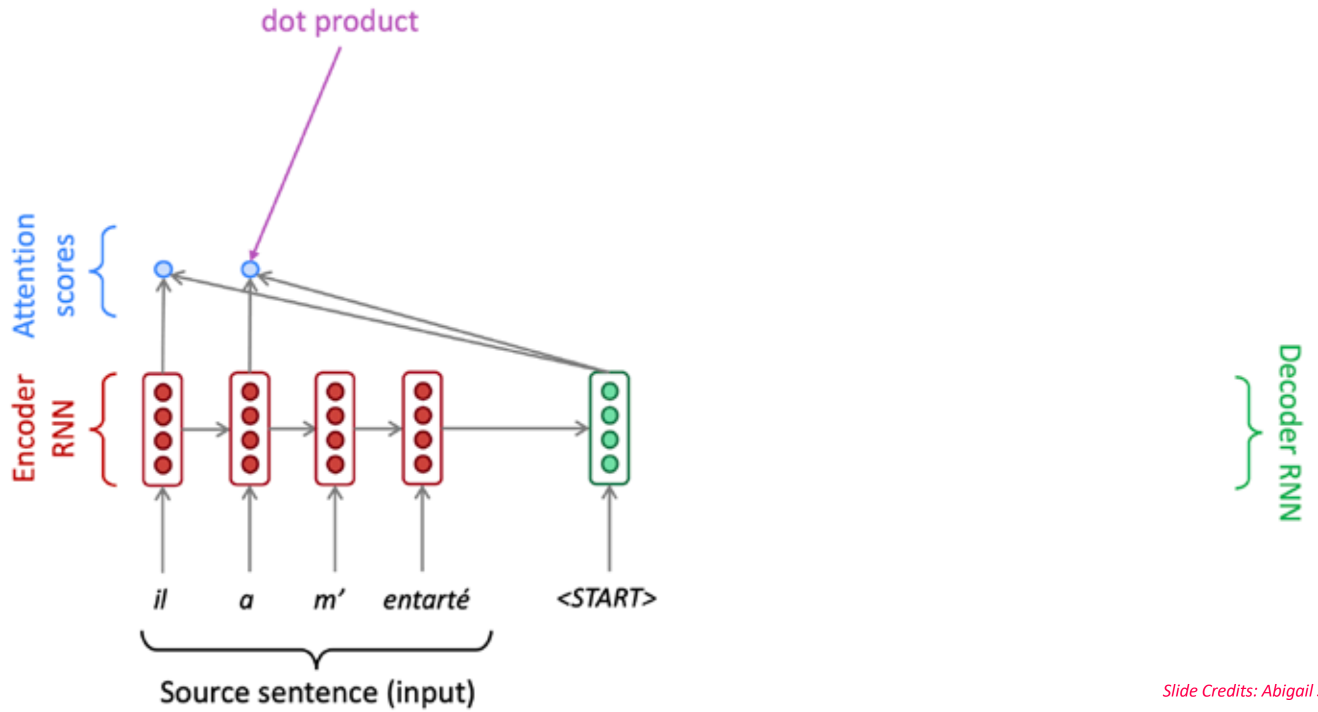
Slide Credits: Abigail See (Stanford) CS224N

Attention Walkthrough: Encoding Score



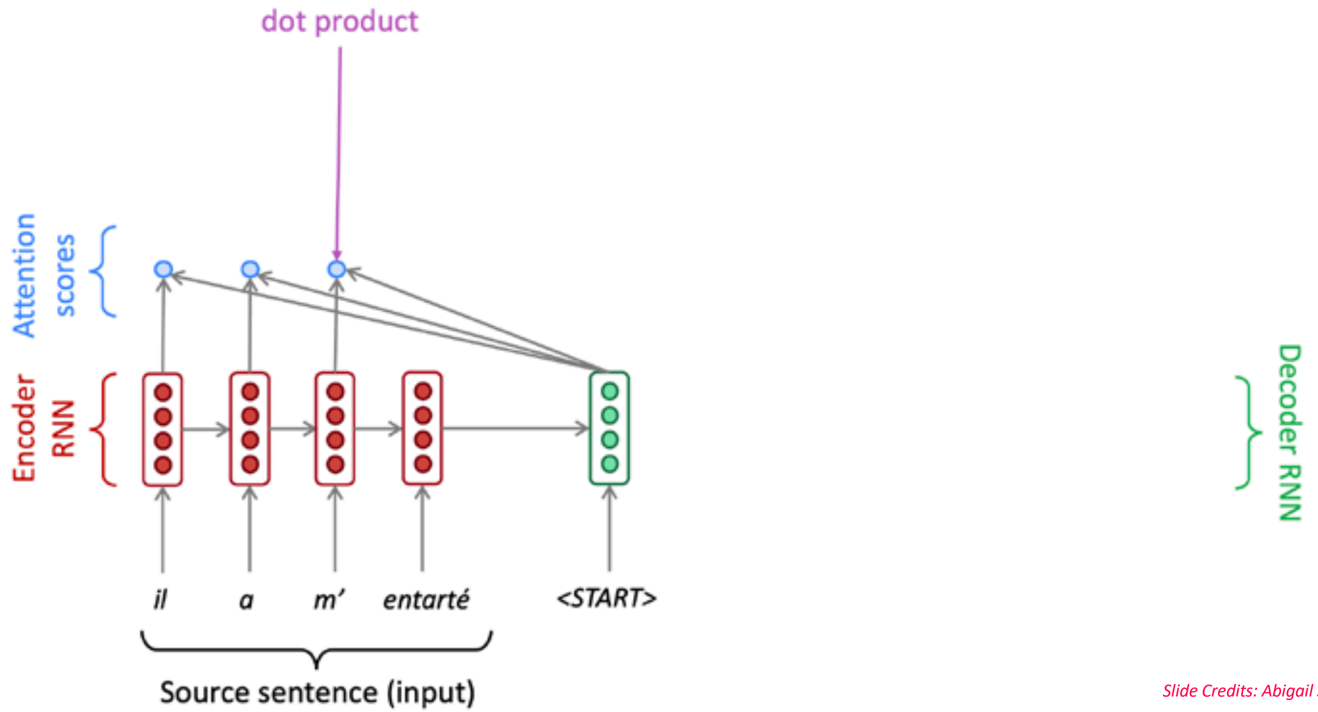
Slide Credits: Abigail See (Stanford) CS224N

Attention Walkthrough: Encoding Scores



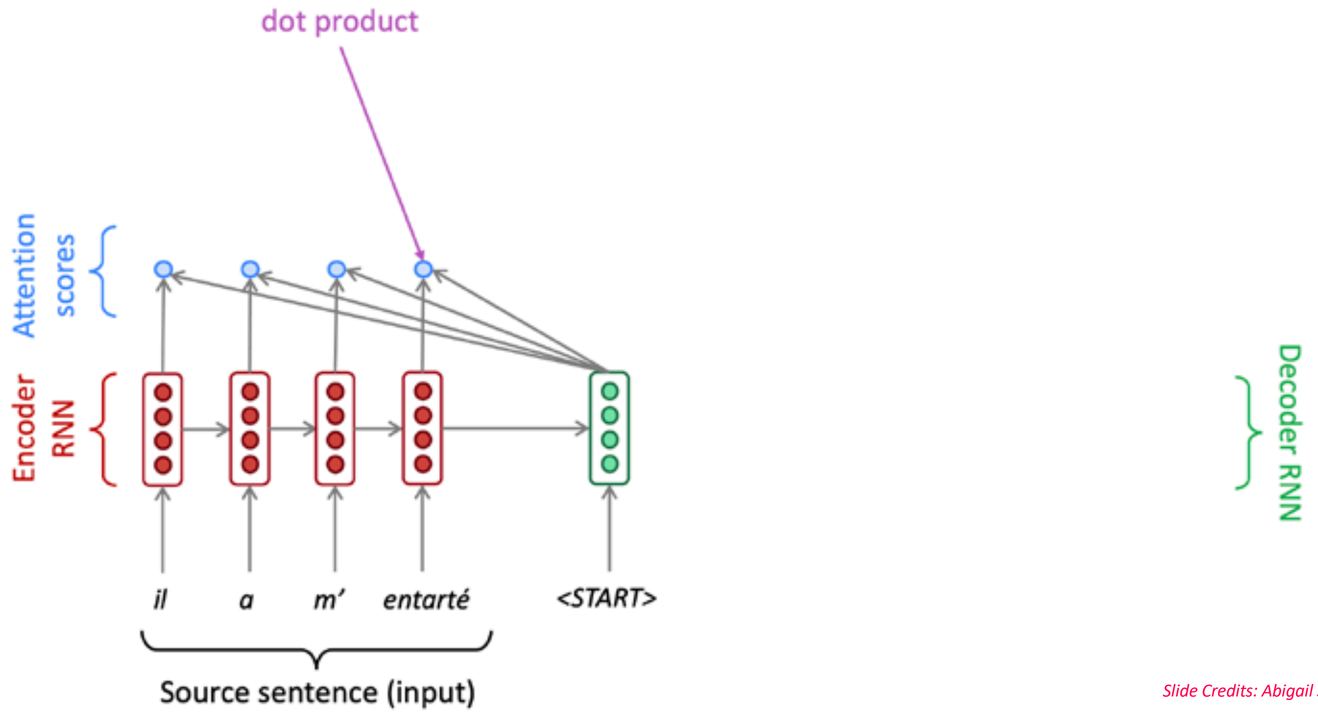
Slide Credits: Abigail See (Stanford) CS224N

Attention Walkthrough: Encoding Scores



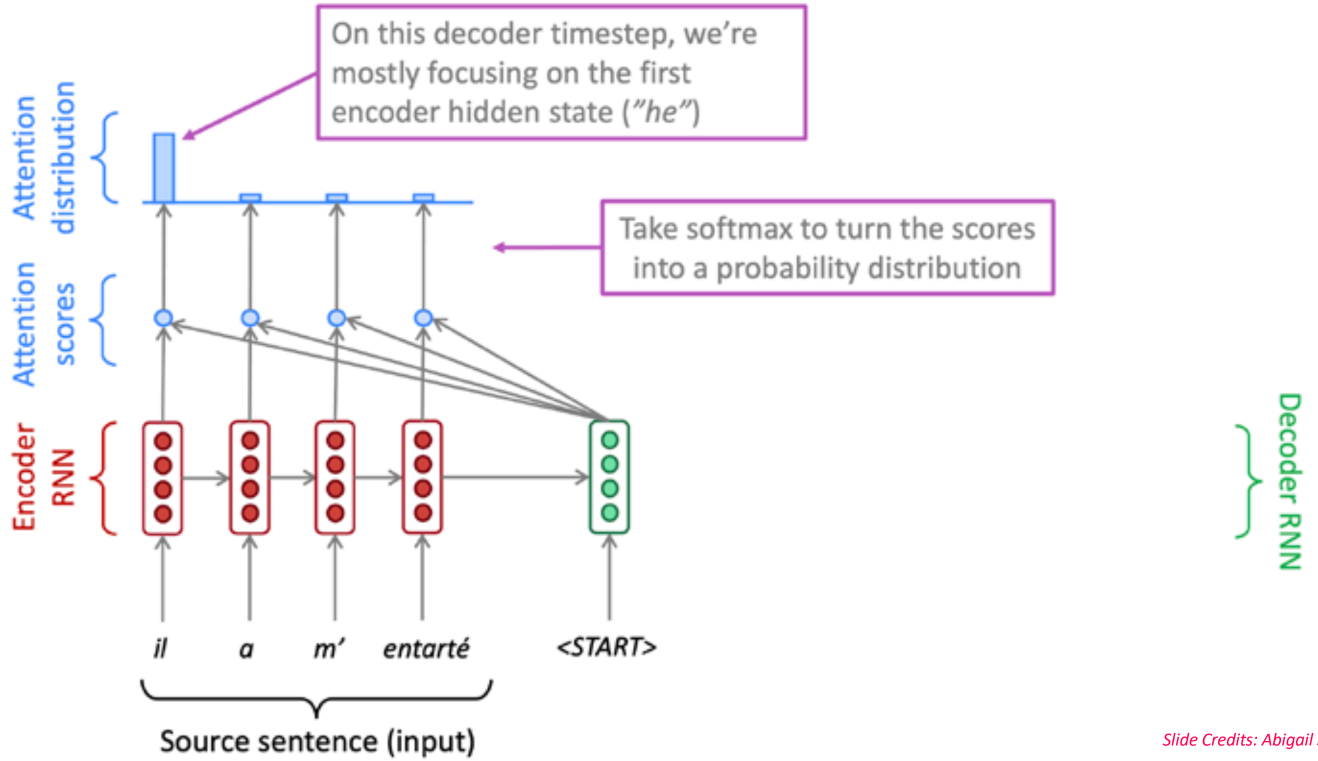
Slide Credits: Abigail See (Stanford) CS224N

Attention Walkthrough: Encoding Scores



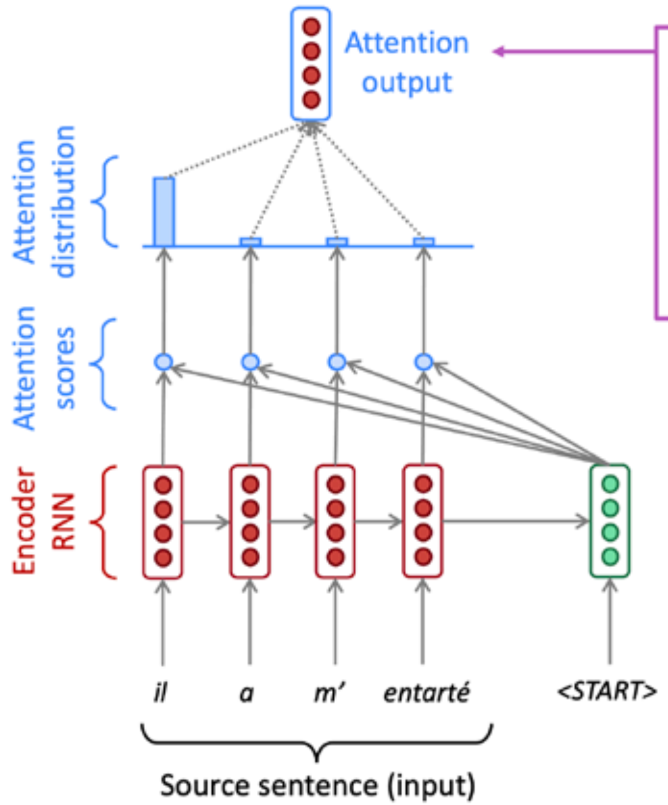
Slide Credits: Abigail See (Stanford) CS224N

Attention Walkthrough: Decoding



Slide Credits: Abigail See (Stanford) CS224N

Attention Output

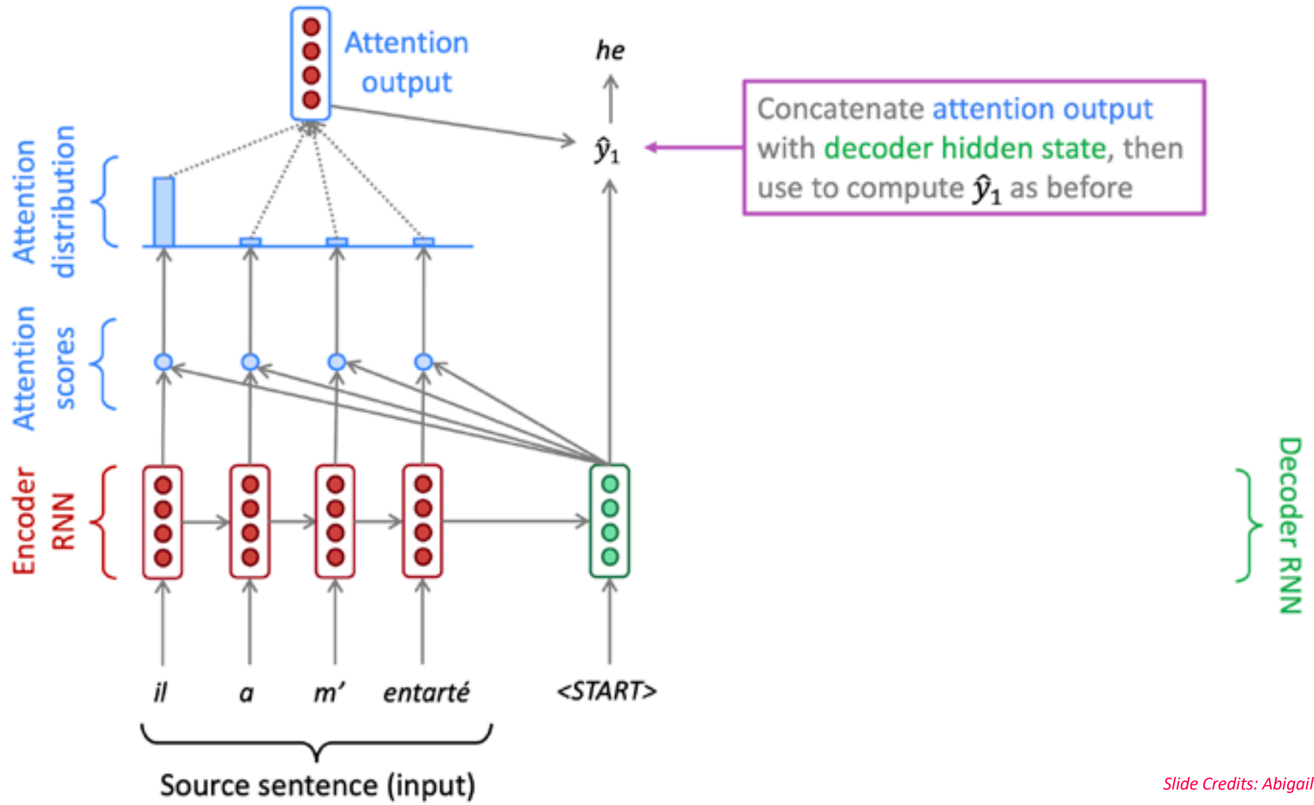


Use the attention distribution to take a **weighted sum** of the encoder hidden states.

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

Slide Credits: Abigail See (Stanford) CS224N

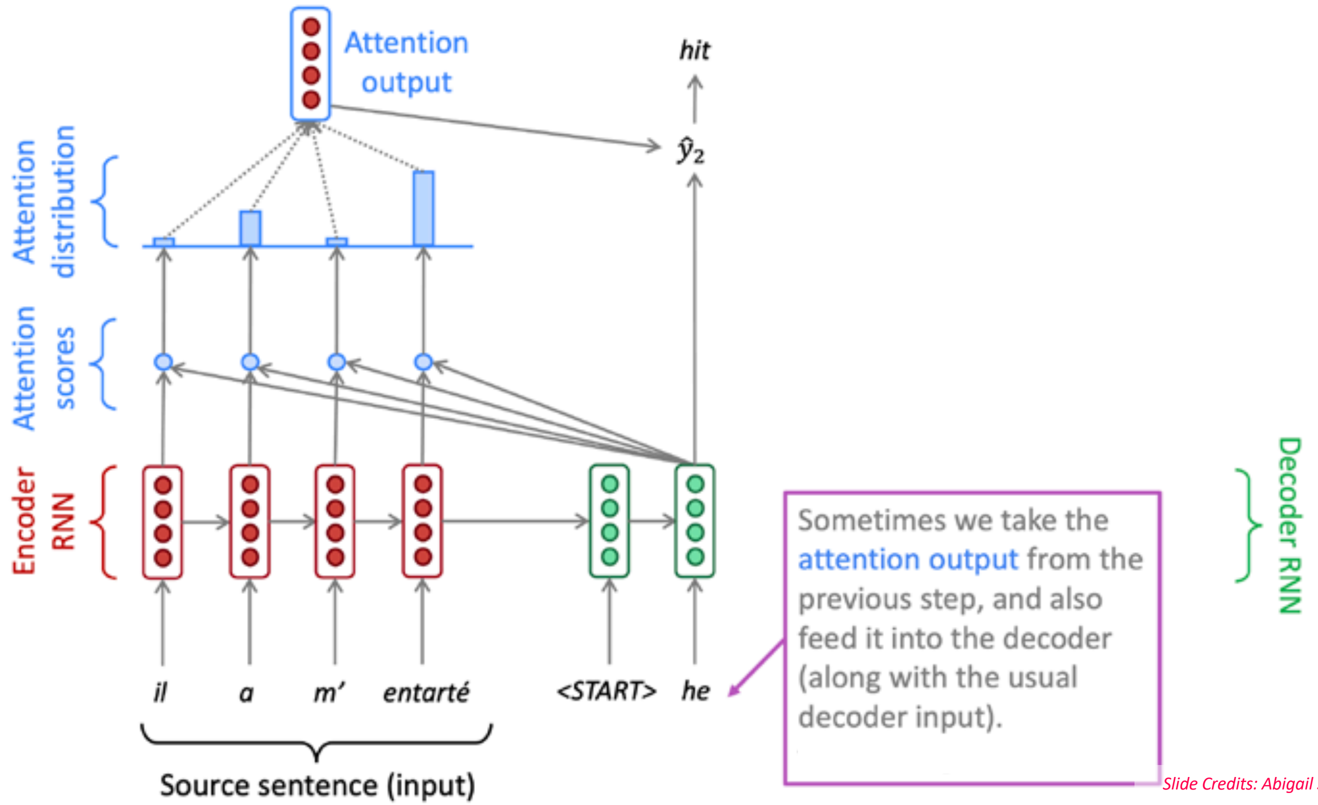
Attention: Predicting $\widehat{y_1} = w_1$



Concatenate attention output with decoder hidden state, then use to compute $\hat{y}_1$ as before

Slide Credits: Abigail See (Stanford) CS224N

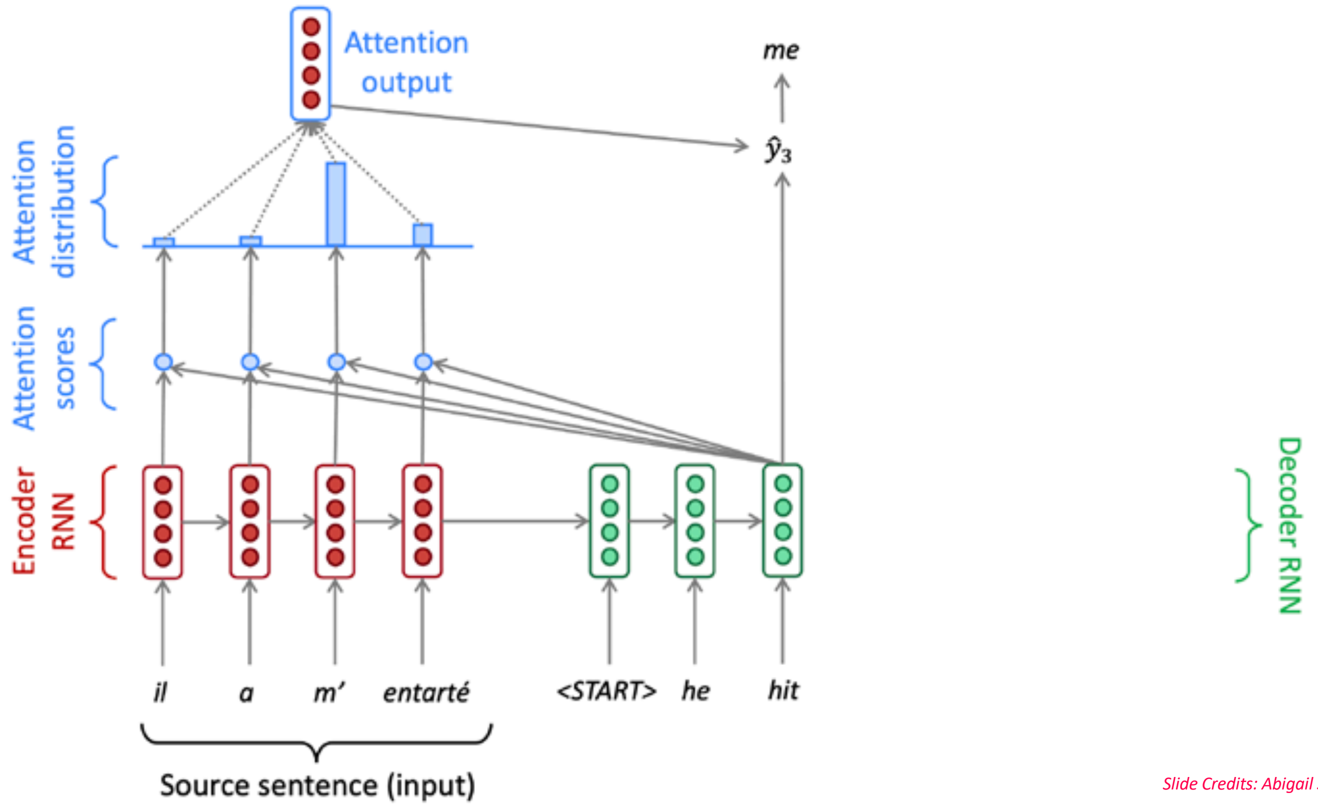
Attention



Sometimes we take the attention output from the previous step, and also feed it into the decoder (along with the usual decoder input).

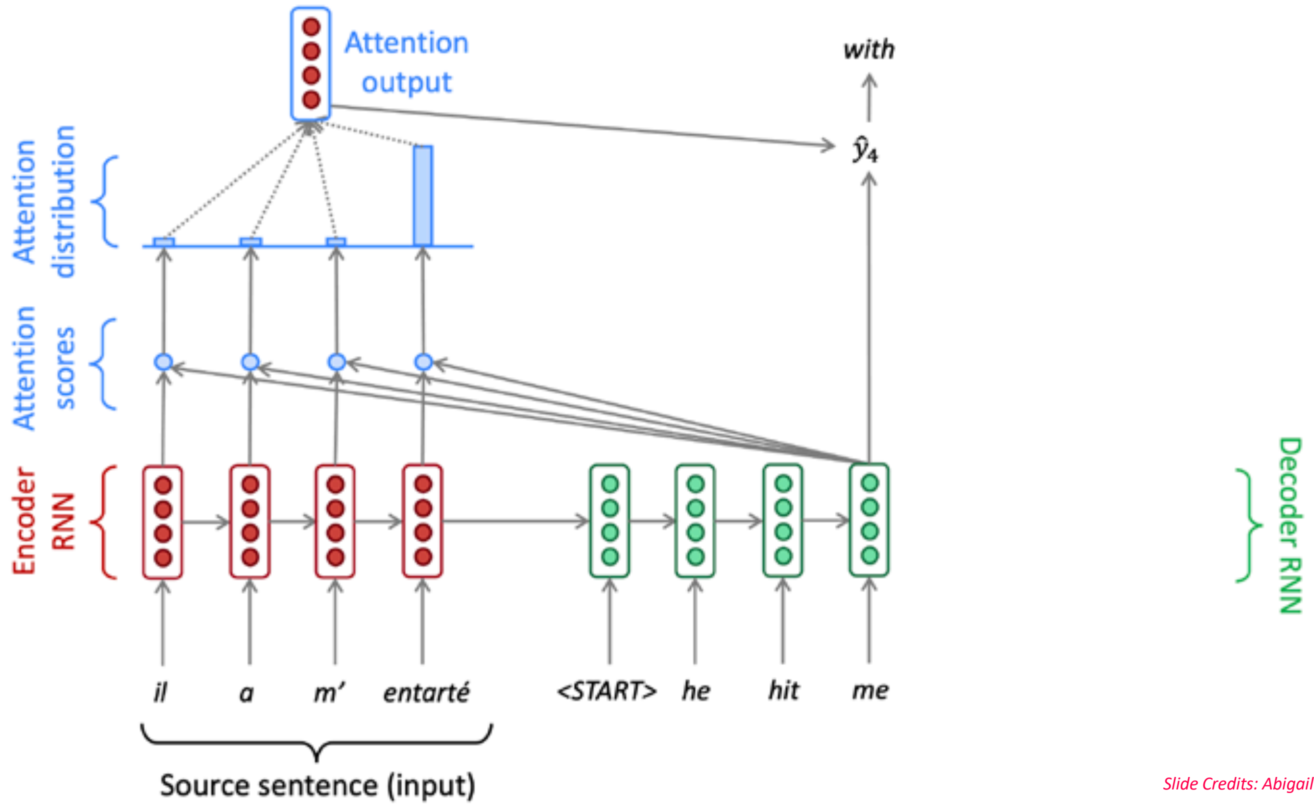
Slide Credits: Abigail See (Stanford) CS224N

Attention



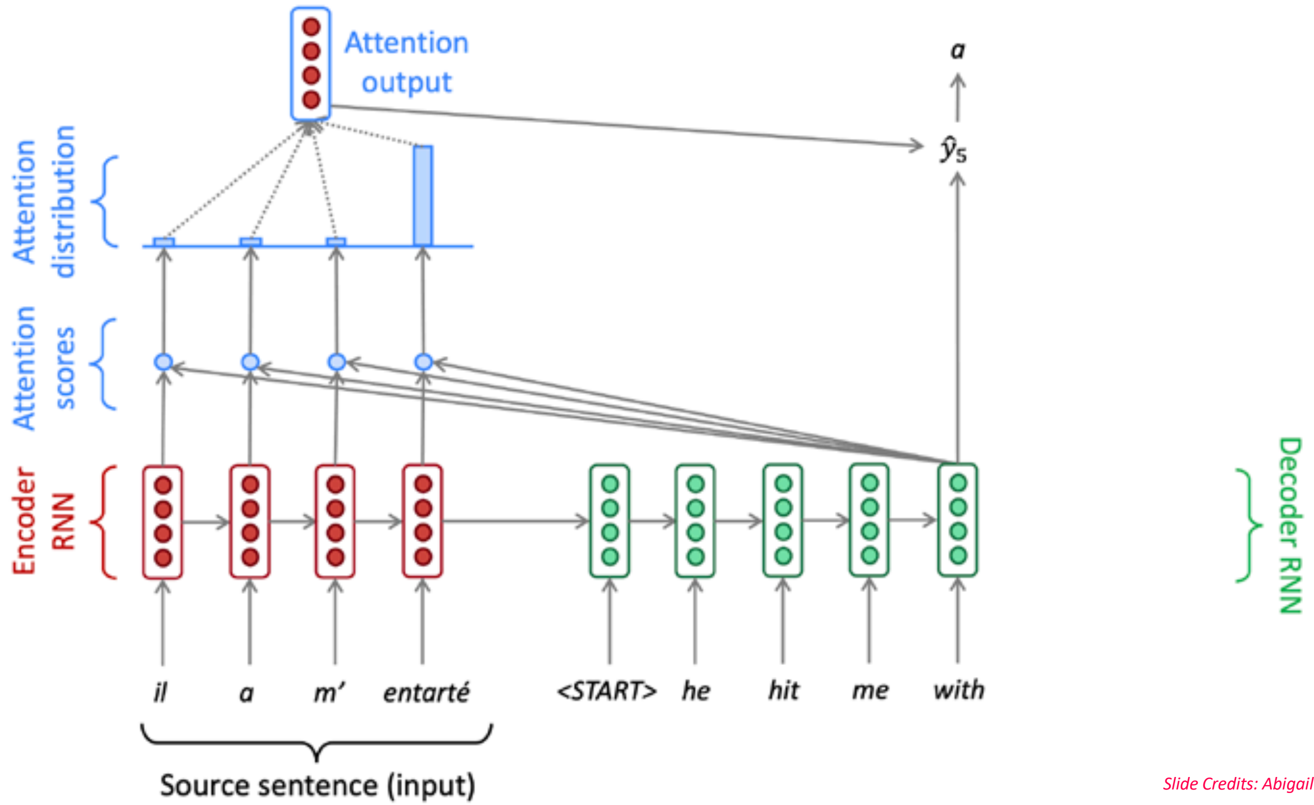
Slide Credits: Abigail See (Stanford) CS224N

Attention



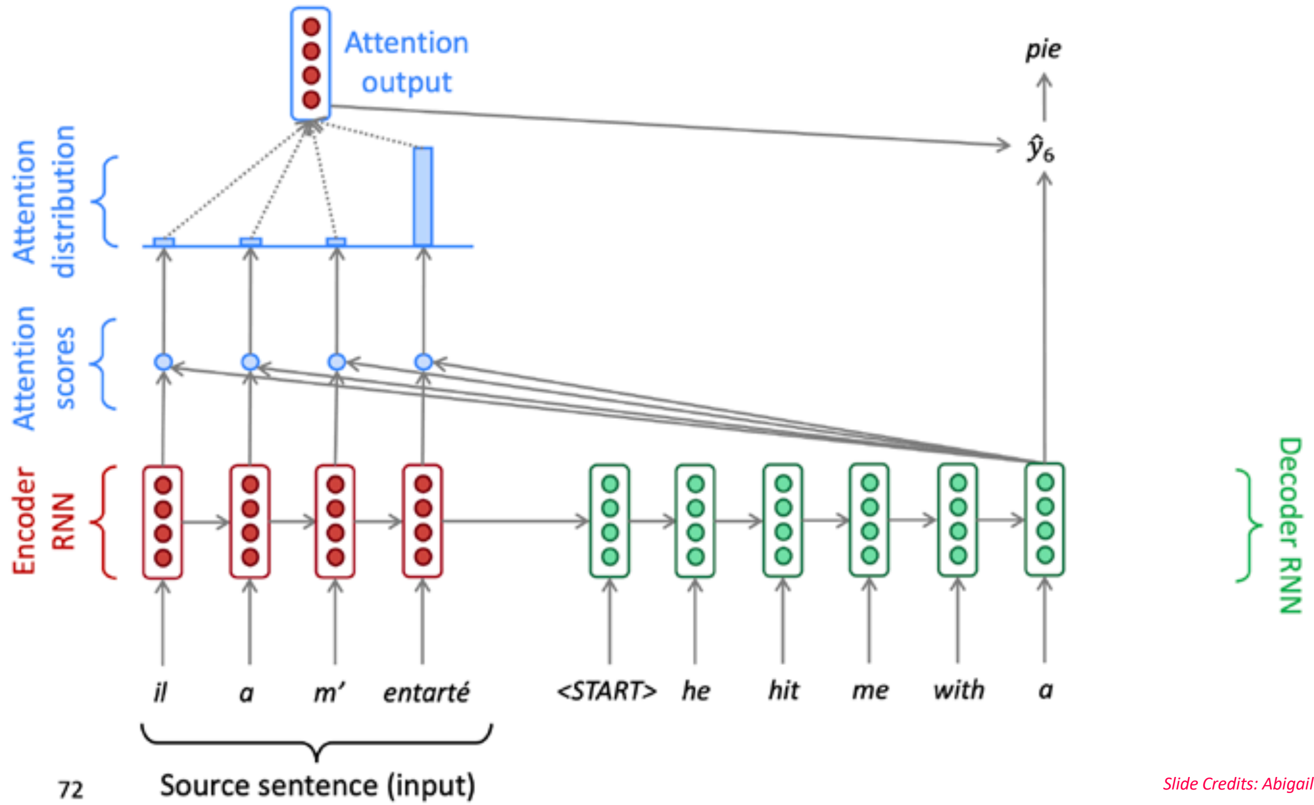
Slide Credits: Abigail See (Stanford) CS224N

Attention



Slide Credits: Abigail See (Stanford) CS224N

Attention

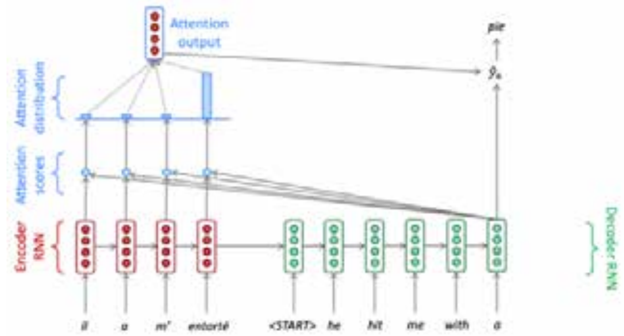


Slide Credits: Abigail See (Stanford) CS224N

Attention: in Equations

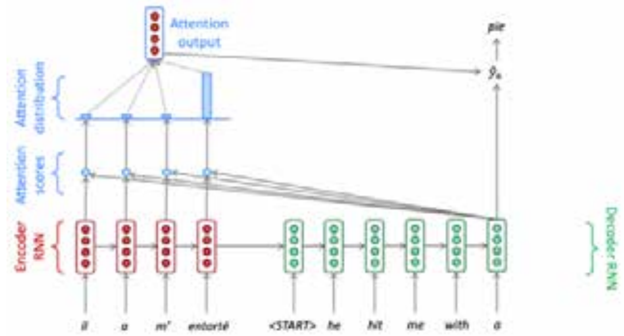
We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$

On each time step t, we have decoder hidden state $d_t \in \mathbb{R}^h$



Slide Credits: Abigail See (Stanford) CS224N

Attention: in Equations



We have encoder hidden states $h_1, \dots, h_N \in \mathbb{R}^h$

On each time step t, we have decoder hidden state $d_t \in \mathbb{R}^h$

We get the attention scores e^t for this step:

$$e^t = [d_t^T h_1, \dots, d_t^T h_N] \in \mathbb{R}^N$$

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

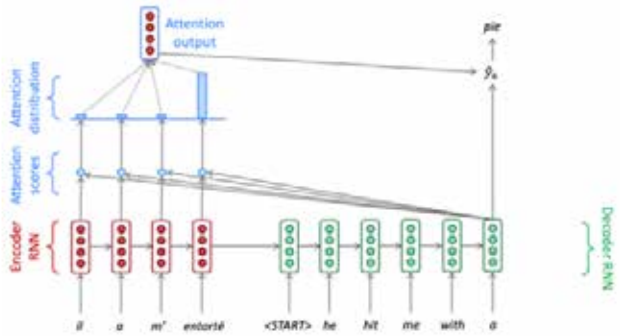
$$\alpha^t = softmax(e^t) \in \mathbb{R}^N$$

We use α^t to take a weighted sum of the encoder hidden states to get the attention output a^t

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

Slide Credits: Abigail See (Stanford) CS224N

Attention: in Equations



We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$

On each time step t, we have decoder hidden state $d_t \in \mathbb{R}^h$

We get the attention scores e^t for this step:

$$e^t = [d_t^T h_1, \ldots, d_t^T h_N] \in \mathbb{R}^N$$

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

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

We use α^t to take a weighted sum of the encoder hidden states to get the attention output a^t

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

Finally, we concatenate the attention output a^t with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

$$[a^t; s_t] \in \mathbb{R}^{2h}$$

Slide Credits: Abigail See (Stanford) CS224N

Attention, please! A summary

Significantly improves performance

- Useful to allow decoder to focus on certain parts of the source

Solves the bottleneck problem

- Attention allows the decoder to look directly at source; bypassing the bottleneck

Helps with vanishing gradient problem in training

- Provides shortcut to faraway states; RNN often “forgets”

Provides some interpretability

- By inspecting the attention distribution, we can see what the decoder was focusing on

Slide Credits: Abigail See (Stanford) CS224N

Generalizing Attention

There are variants of attention scores, but in general:

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).

Beam Search Decoding

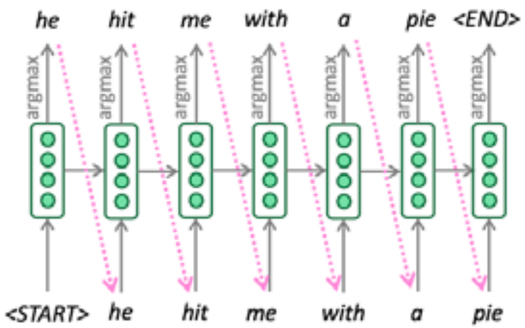
Approximately finding the best output.

Be greedy. Very greedy. Greedy b times.

Slide Credits: Abigail See (Stanford) CS224N

Review: Greedy Decoding

How to generate (decode) the target output?
By taking an argmax on each step of the decoder



This is greedy decoding (take most probable word on each step)

Problems with this method?

Slide Credits: Abigail See (Stanford) CS224N

Greedy Decoding

Greedy decoding has no way to undo decisions!

- Input: *il a m'entarté* (he hit me with a pie)
- → *he* ____
- → *he hit* ____
- → *he hit a* ____ (whoops! no going back now…)

How to fix this?

Slide Credits: Abigail See (Stanford) CS224N

Exhaustive search decoding

Ideally, we want to find a (length T) translation y that maximizes

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

$$= \prod_{t=1}^{T} P(y_t|y_1, \ldots, y_{t-1}, x)$$

We could try computing all possible sequences y

- This means that on each step t of the decoder, we are tracking V^t possible partial translations, where V is the vocabulary size
- This $O(V^T)$ complexity is intractable!

cf n-gram models without Markov assumption)

Slide Credits: Abigail See (Stanford) CS224N

Beam Search Decoding

Core idea: on each step of the decoder, keep track of the k most probable partial translations (which we call hypotheses)

- k is the beam size (in practice around 5 to 10)

A hypothesis $y_1, \dots, y_t$ has a score which is its log probability:

$$score(y_1, \dots, y_t) = log P_{LM}(y_1, \dots, y_t | x) = \sum_{i=1}^{t} log P_{LM}(y_i | y_1, \dots, y_{i-1}, x)$$

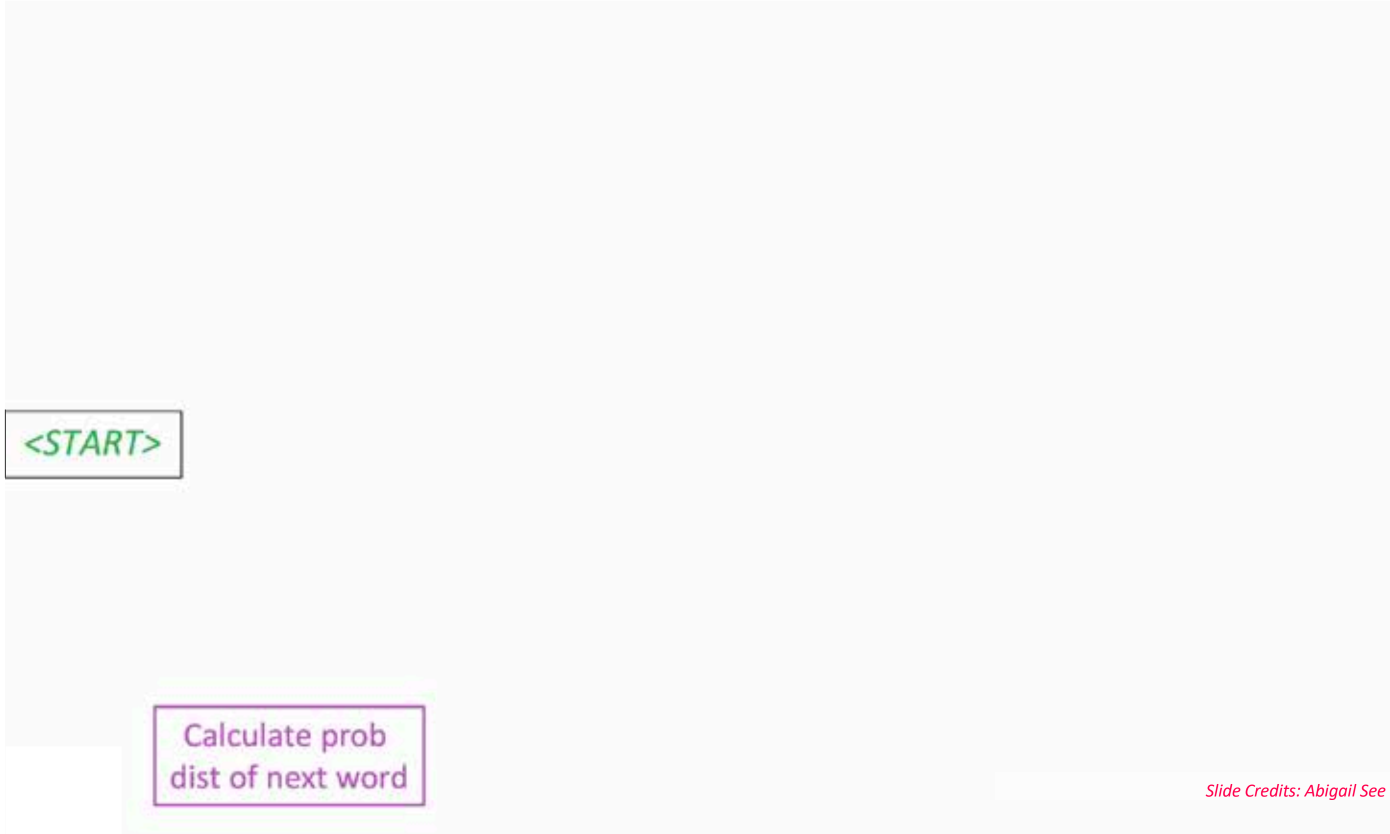
- Scores are all negative, and higher score is better
- We search for high-scoring hypotheses, tracking top k on each step

Beam search is not guaranteed to find optimal solution

But its much more efficient than exhaustive search!

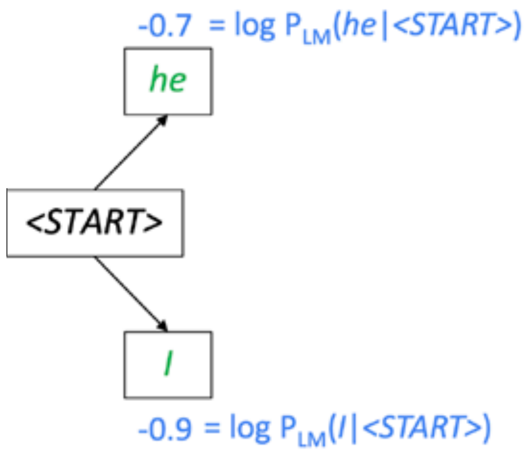
Slide Credits: Abigail See (Stanford) CS224N

Beam Search Decoding



Slide Credits: Abigail See (Stanford) CS224N

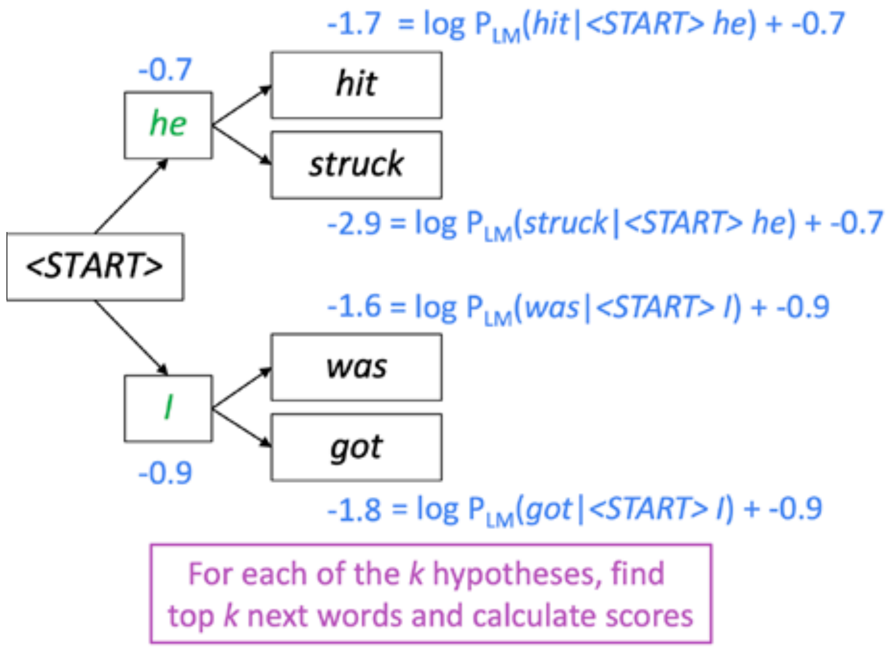
Beam Search Decoding



Take top *k* words
and compute scores

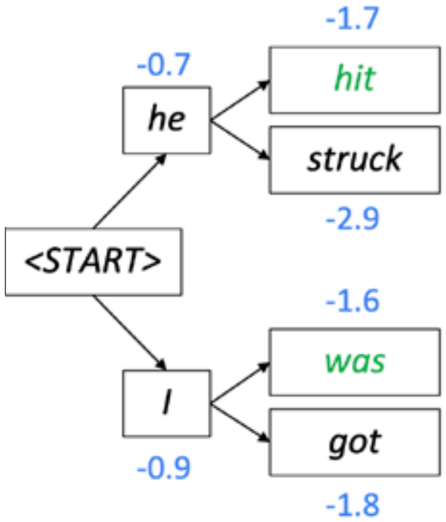
Slide Credits: Abigail See (Stanford) CS224N

Beam Search Decoding



Slide Credits: Abigail See (Stanford) CS224N

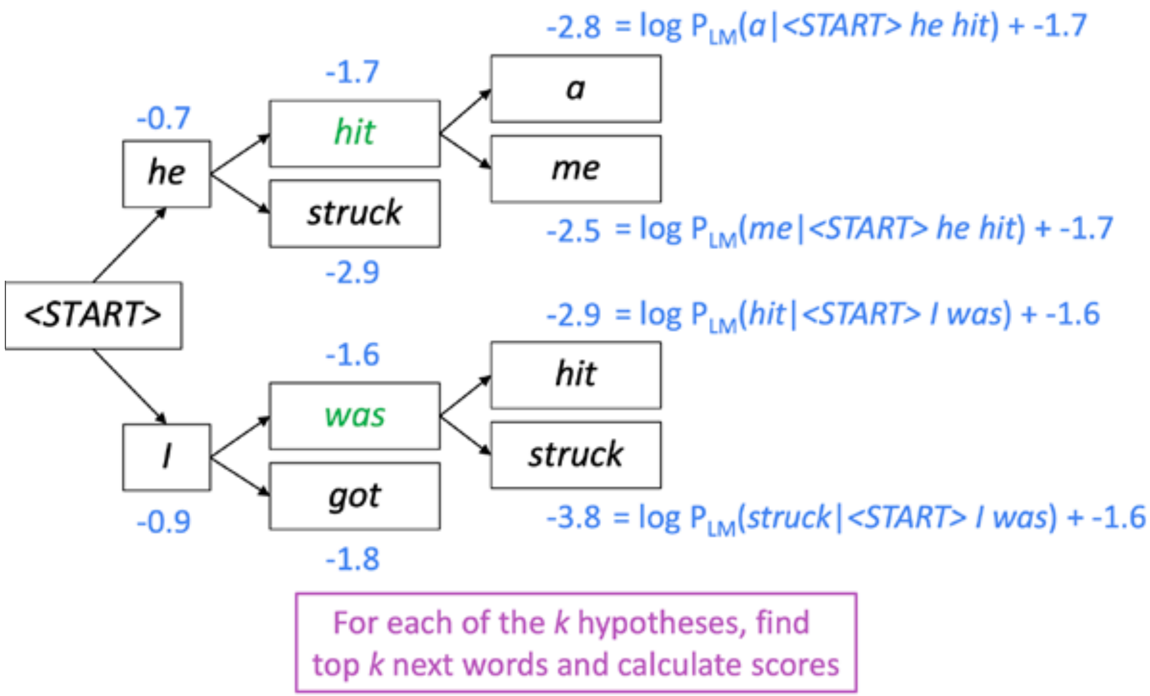
Beam Search Decoding



Of these k^2 hypotheses,
just keep k with highest scores

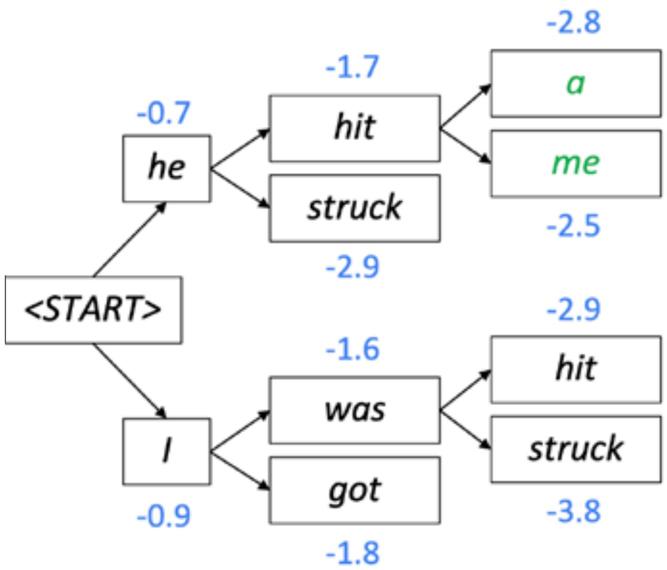
Slide Credits: Abigail See (Stanford) CS224N

Beam Search Decoding



Slide Credits: Abigail See (Stanford) CS224N

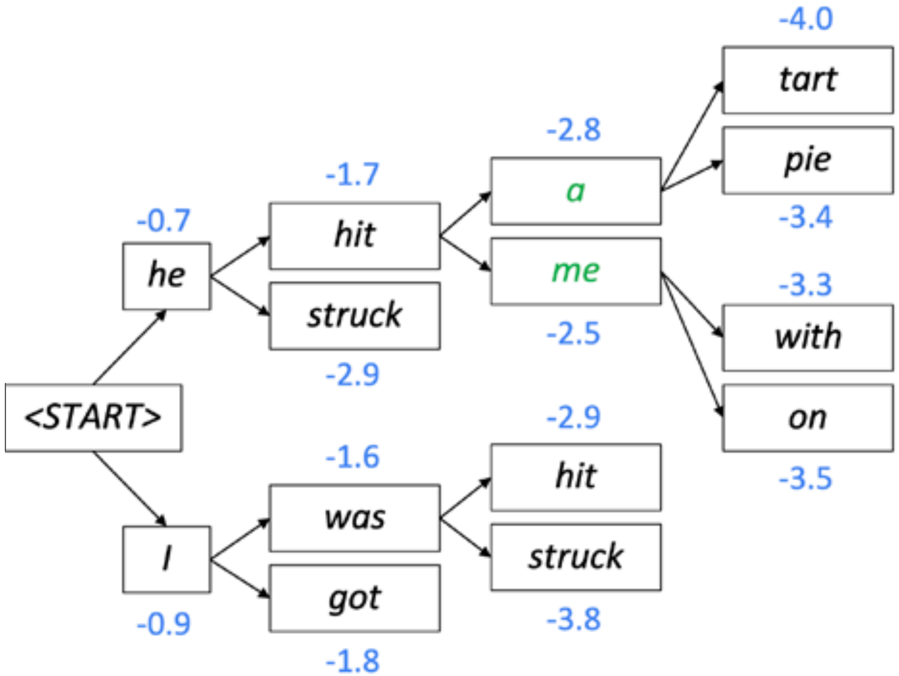
Beam Search Decoding: Pruning



Of these k^2 hypotheses, just keep k with highest scores

Slide Credits: Abigail See (Stanford) CS224N

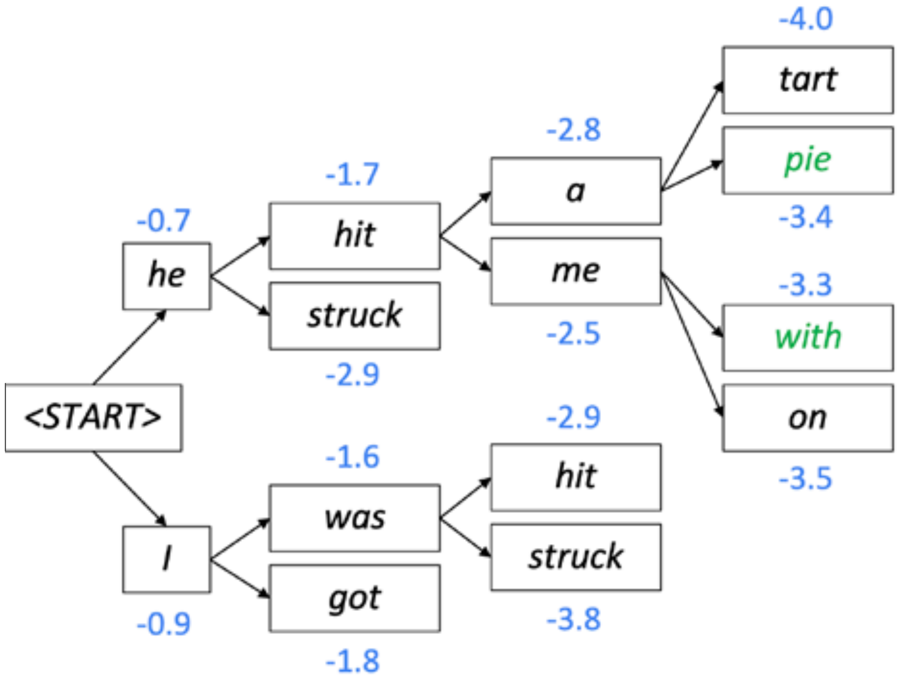
Beam Search Decoding



For each of the *k* hypotheses, find top *k* next words and calculate scores

Slide Credits: Abigail See (Stanford) CS224N

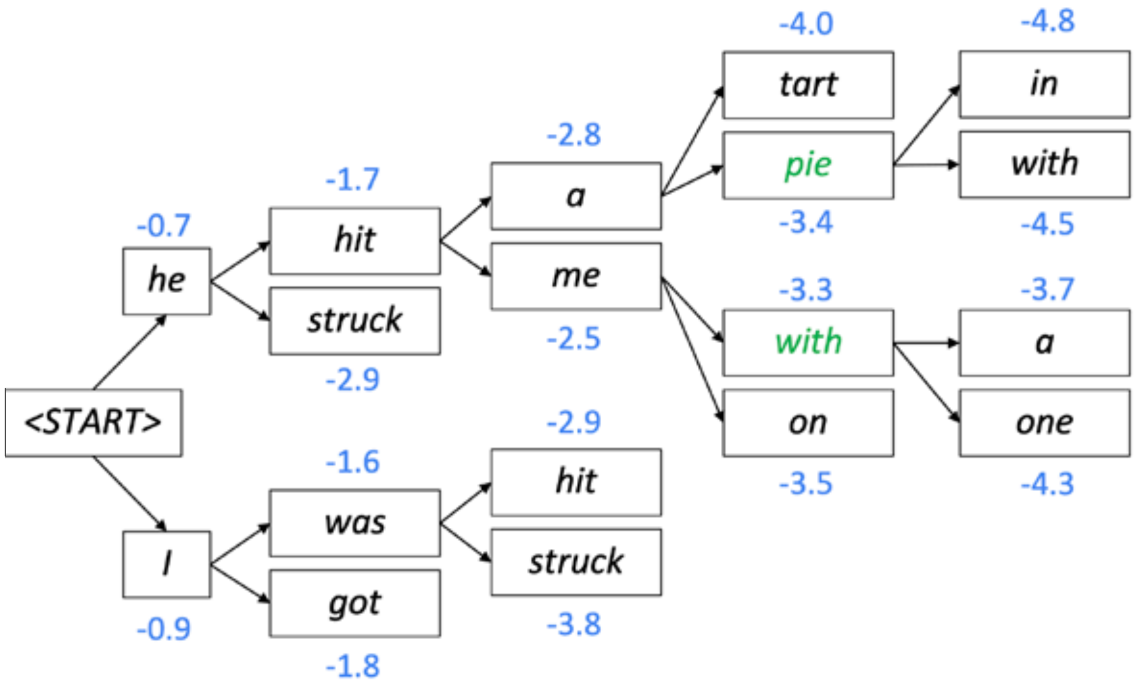
Beam Search Decoding



Of these k^2 hypotheses,
just keep k with highest scores

Slide Credits: Abigail See (Stanford) CS224N

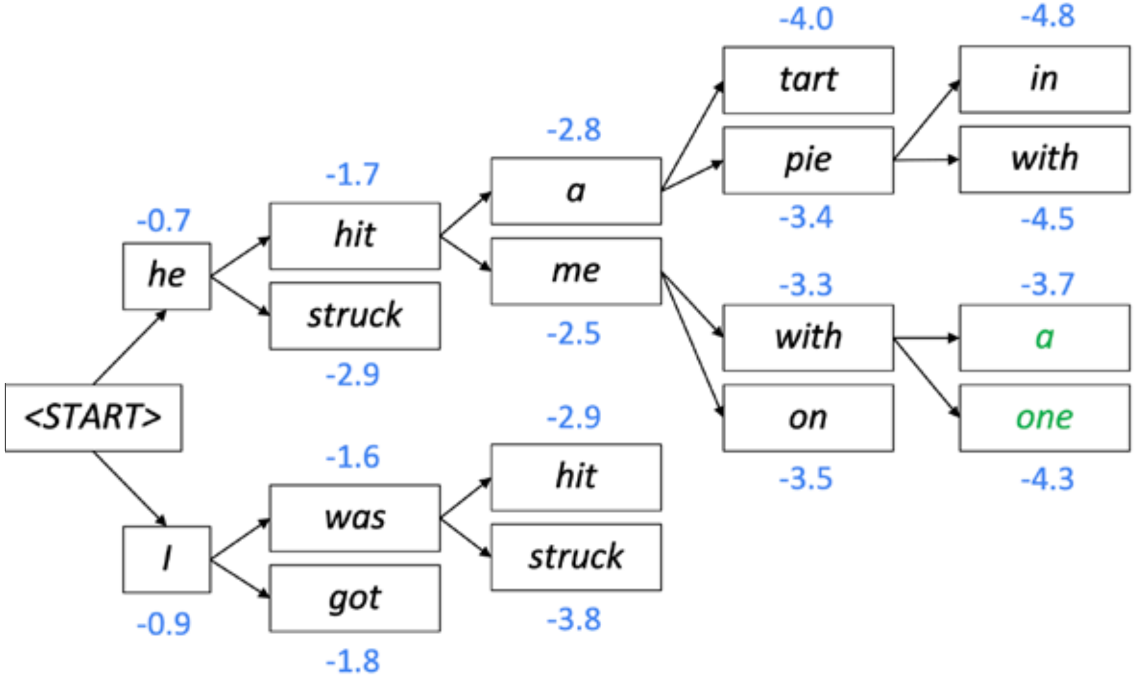
Beam Search Decoding



Slide Credits: Abigail See (Stanford) CS224N

For each of the *k* hypotheses, find top *k* next words and calculate scores

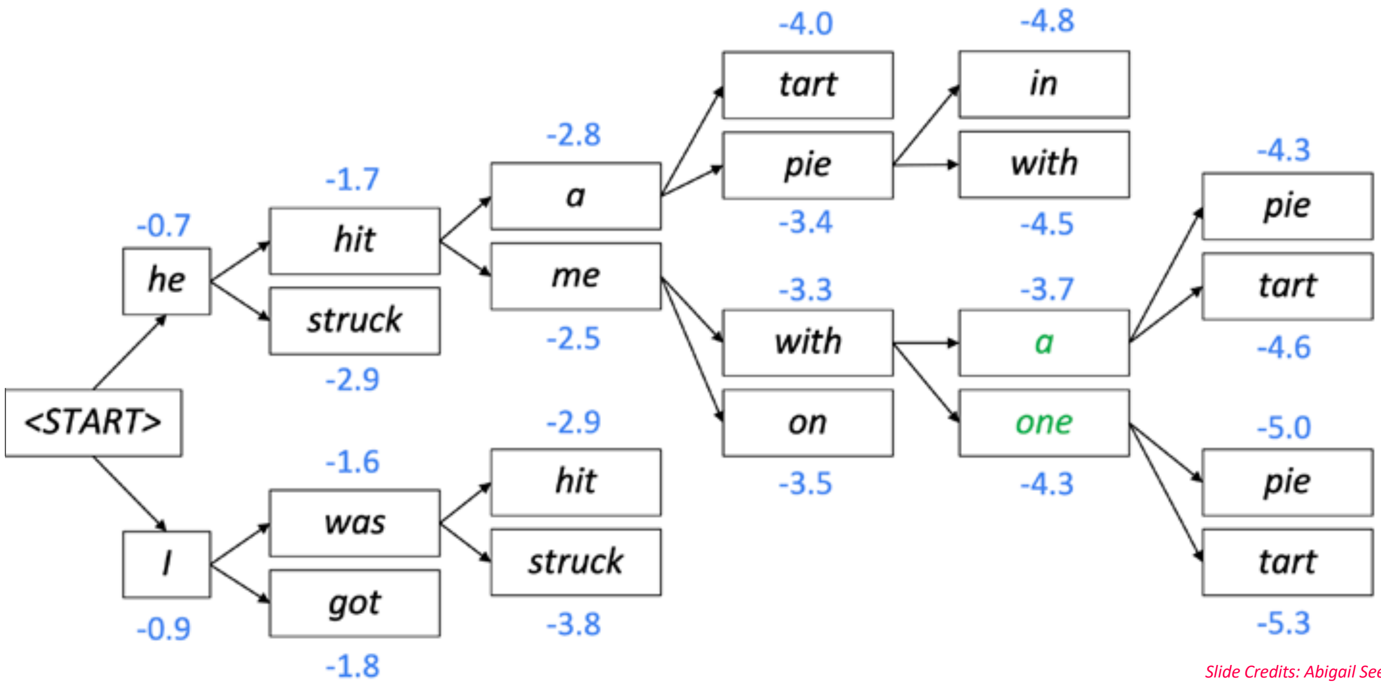
Beam Search Decoding



Slide Credits: Abigail See (Stanford) CS224N

Of these k^2 hypotheses, just keep k with highest scores

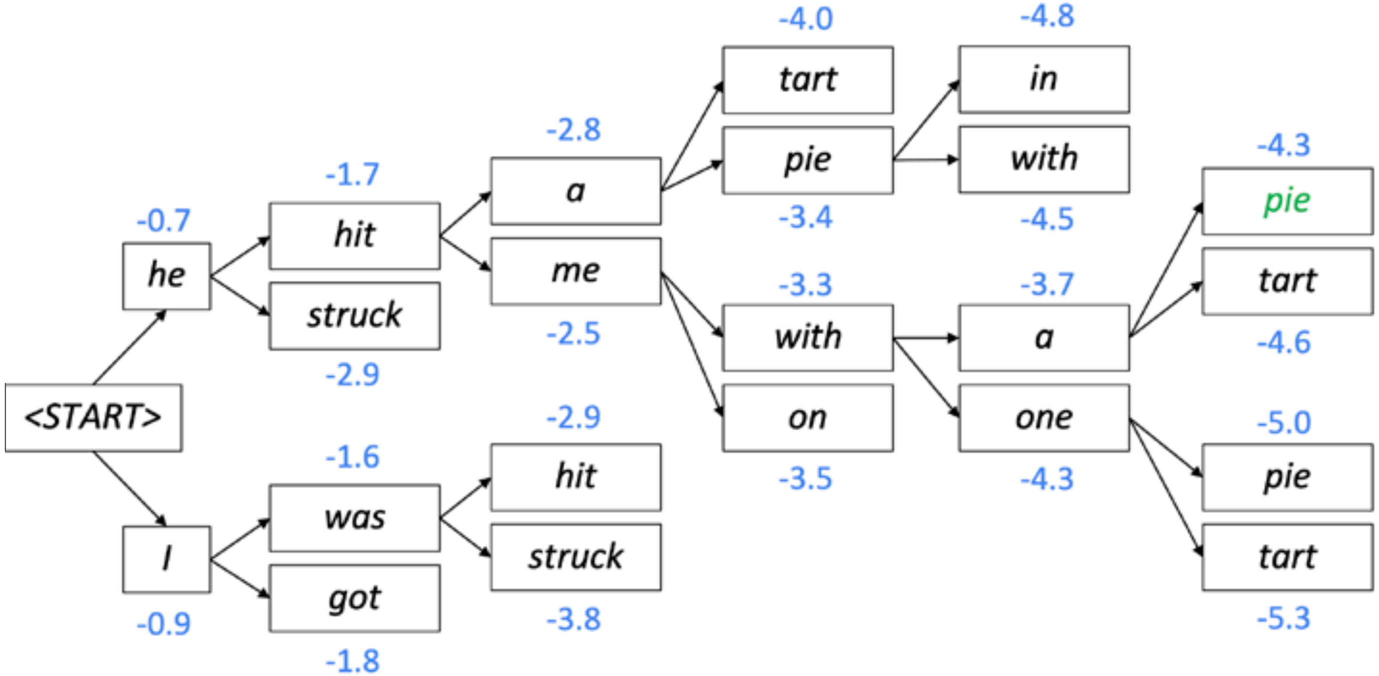
Beam Search Decoding



Slide Credits: Abigail See (Stanford) CS224N

For each of the *k* hypotheses, find top *k* next words and calculate scores

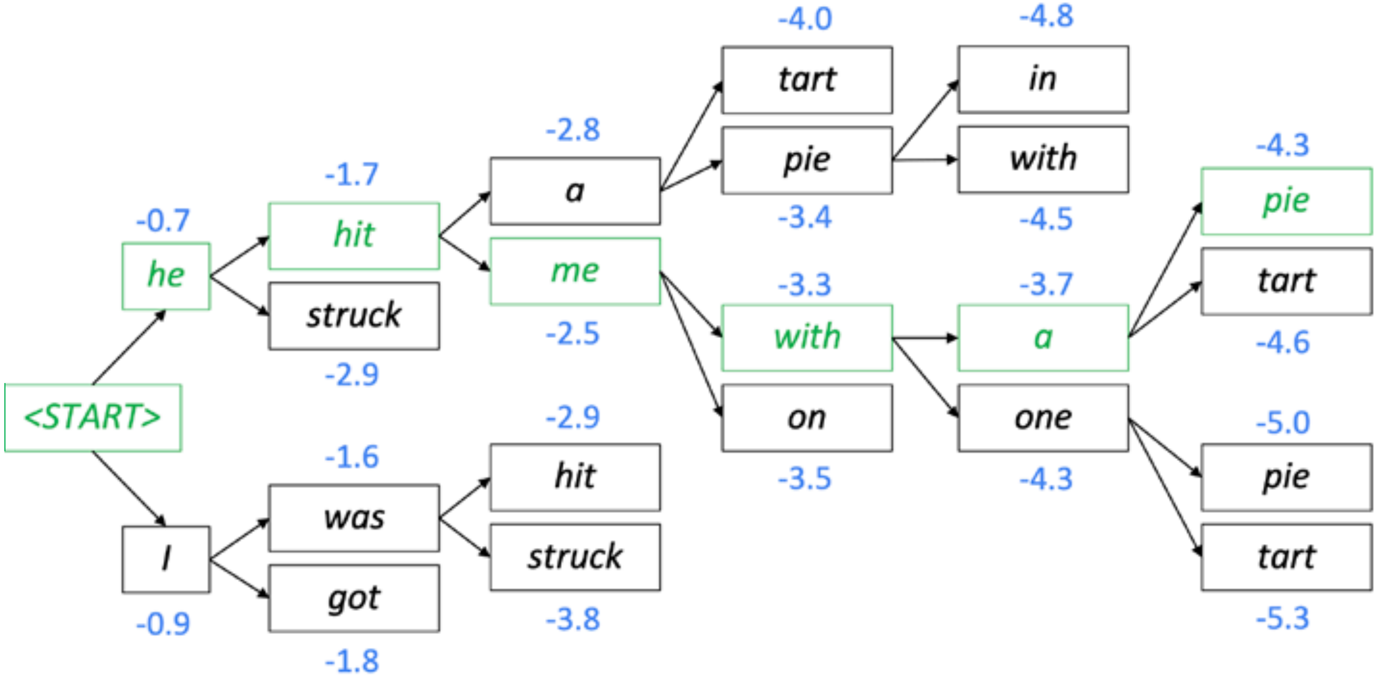
Beam Search Decoding



Slide Credits: Abigail See (Stanford) CS224N

This is the top-scoring hypothesis!

Beam Search Decoding: Backtrack



Slide Credits: Abigail See (Stanford) CS224N

Backtrack to obtain the full hypothesis

Beam Search Decoding

In greedy decoding, usually we decode until the model produces a <END> token

- For example: <START> *he hit me with a pie* <END>

In beam search decoding, different hypotheses may produce <END> tokens on different timesteps

- When a hypothesis produces <END>, that hypothesis is complete.
- Place it aside and continue exploring other hypotheses via beam search.

Usually we continue beam search until:

- We reach timestep T (where T is some pre-defined cutoff), or
- We have at least n completed hypothesis (where n is a pre-defined cutoff)

Slide Credits: Abigail See (Stanford) CS224N

Sampling-based Decoding

Pure sampling: On each step t, randomly sample from the probability distribution P_t to obtain your next word.

- Like greedy decoding, but sample instead of argmax.

Top-n sampling: On each step t, randomly sample from P_t, restricted to just the top-n most probable words

- Like pure sampling, but truncate the probability distribution
- n=1 is greedy search, n=V is pure sampling
- Increase n to get more diverse/risky output
- Decrease n to get more generic/safe output

Slide Credits: Abigail See (Stanford) CS224N

Search Summary

Greedy decoding is a simple method; gives low quality output

Beam search (especially with high beam size) searches for high-probability output

- Delivers better quality than greedy, but if beam size is too high, can return high-probability but unsuitable output (e.g., generic, short)

Sampling methods are a way to get more diversity and randomness

- Good for open-ended/creative generation (poetry, stories)
- Top-n sampling allows you to control diversity

Slide Credits: Abigail See (Stanford) CS224N