

Neural Language Models & Attention

Lara J. Martin (she/they)

<https://laramartin.net/neurosymbolic-text-gen/>

Slides modified from Dr. Daphne Ippolito & Dr. Frank Ferraro

TA office hours

Arya Honraopatil

ahonrao1@umbc.edu

Wednesdays 2-4pm in ITE 340



Learning Objectives

Discover the basic function of language models

Determine why sequence-to-sequence models emerged from the regular RNN model

Explore the components of RNNs and seq2seq models

Understand the utility of attention mechanisms

Review

What are some examples of symbolic components?



When poll is active respond at
PollEv.com/laramartin527

Or

Send **laramartin527** and your message to **22333**

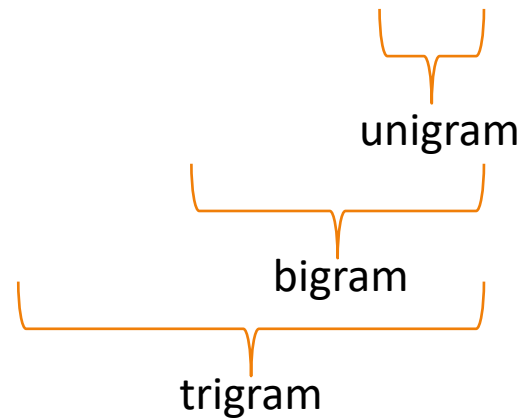
Language Models

What is a language model?

A model, given a history of words, that outputs likely next words.

Originally, they were statistical **n-gram models**.

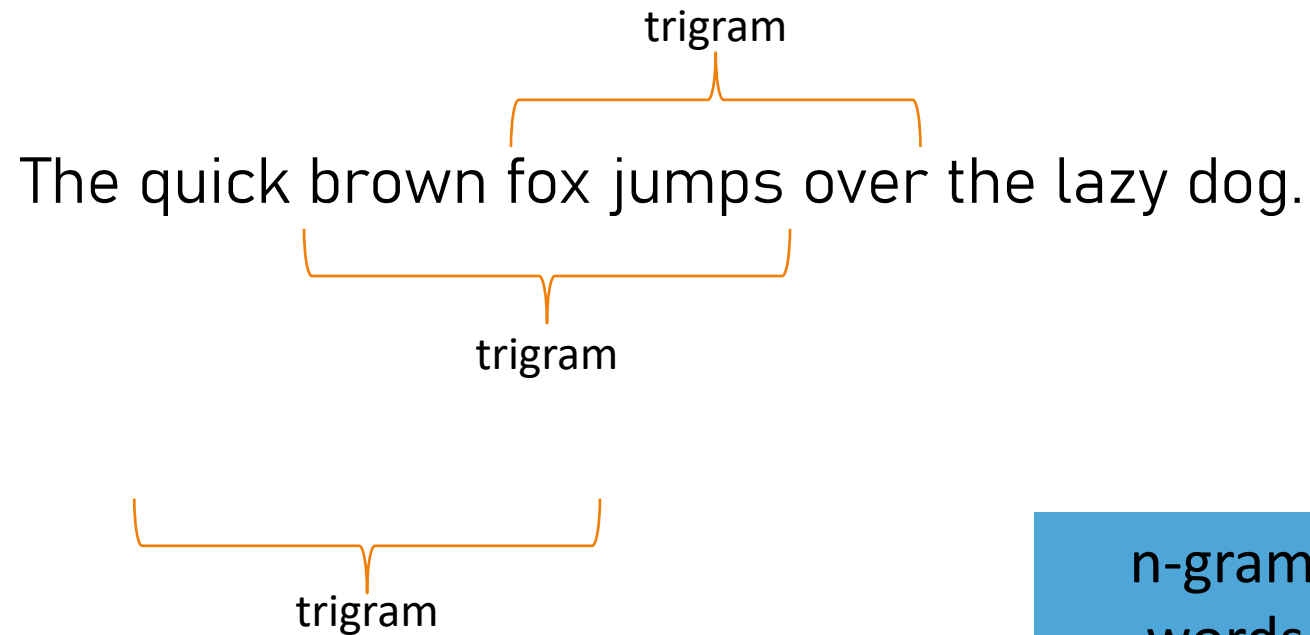
The quick brown fox jumps over the lazy dog.



What is a language model?

A model, given a history of words, that outputs likely next words.

Originally, they were statistical **n-gram models**.



n-gram: any consecutive n # of words, treated as a single unit

What is a language model?

A model, given a history of words, that outputs **likely** next words.

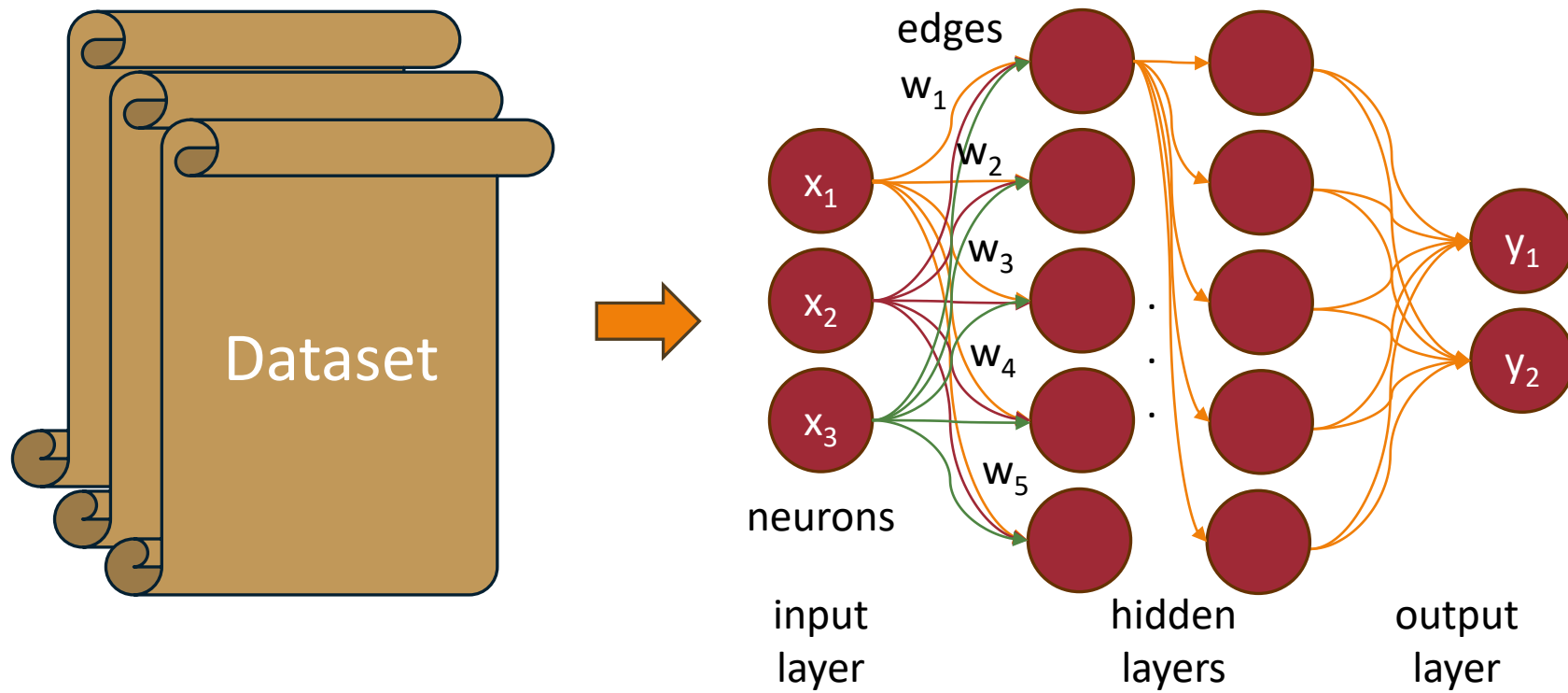
$$P(\text{"apple"} \mid \text{"eat the"}) = 0.02$$

$$P(\text{"pineapple"} \mid \text{"eat the"}) = 0.01$$

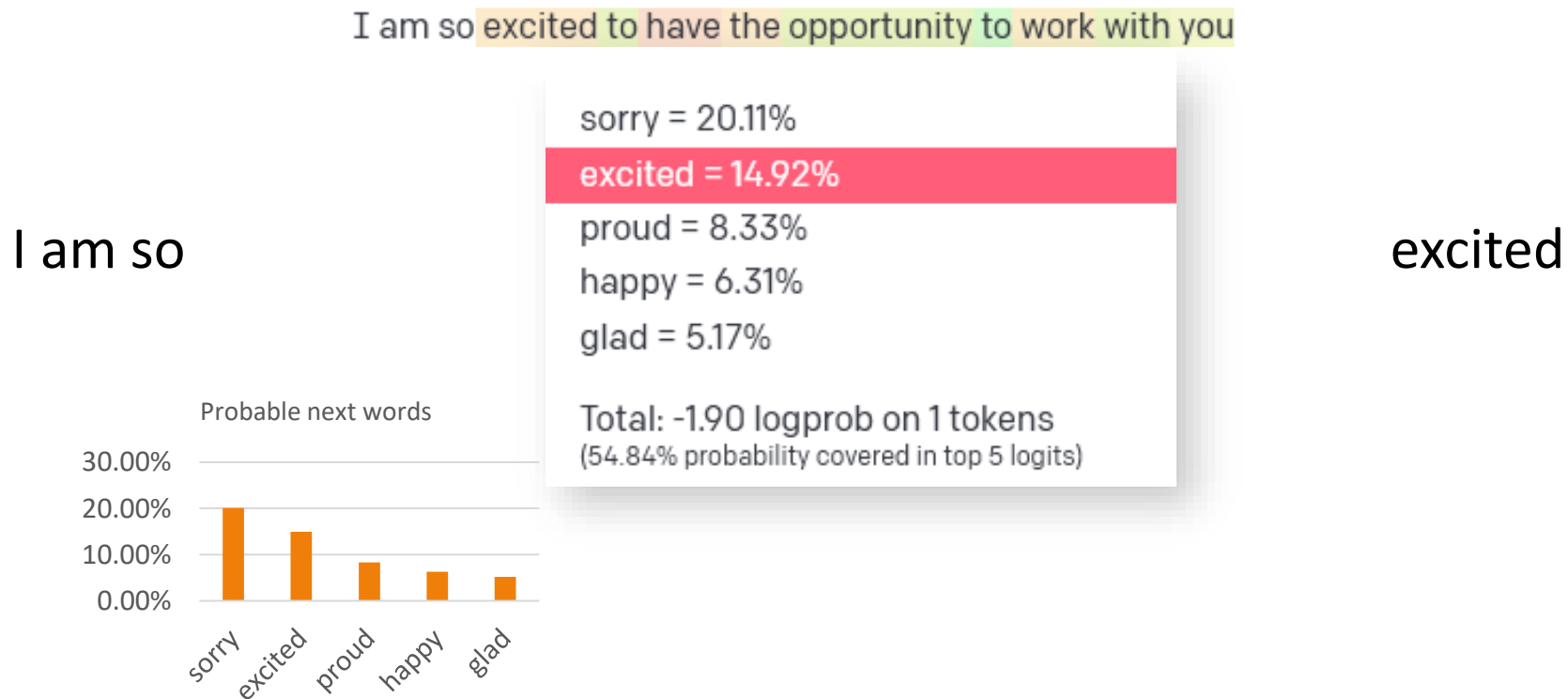
$$P(\text{"suitcase"} \mid \text{"eat the"}) = 0.0001$$

What is a *neural* language model?

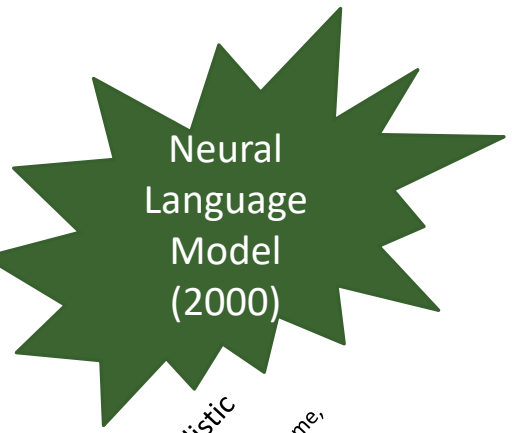
What's a neural network?



Using a neural language model



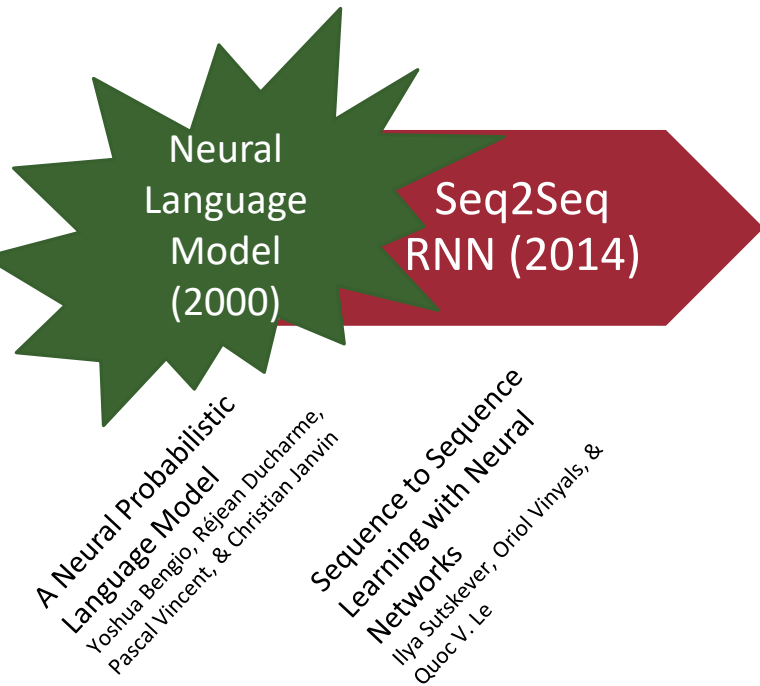
Neural Language Model Timeline



Neural
Language
Model
(2000)

A Neural Probabilistic
Language Model
Yoshua Bengio, Réjean Ducharme,
Pascal Vincent, & Christian Jauvin

Neural Language Model Timeline



Sequence-to-Sequence RNNs

Up until 2017 or so, neural language models were mostly built using recurrent neural networks.

Sequence to Sequence Learning with Neural Networks

Ilya Sutskever
Google
ilyasu@google.com

Oriol Vinyals
Google
vinyals@google.com

Quoc V. Le
Google
qvl@google.com

Abstract

Deep Neural Networks (DNNs) are powerful models that have achieved excellent performance on difficult learning tasks. Although DNNs work well whenever large labeled training sets are available, they cannot be used to map sequences to sequences. In this paper, we present a general end-to-end approach to sequence learning that makes minimal assumptions on the sequence structure. Our method uses a multilayered Long Short-Term Memory (LSTM) to map the input sequence to a vector of a fixed dimensionality, and then another deep LSTM to decode the target sequence from the vector. Our main result is that on an English to French translation task from the WMT'14 dataset, the translations produced by the LSTM achieve a BLEU score of 34.8 on the entire test set, where the LSTM's BLEU score was penalized on out-of-vocabulary words. Additionally, the LSTM did not have difficulty on long sentences. For comparison, a phrase-based SMT system achieves a BLEU score of 33.3 on the same dataset. When we used the LSTM to rerank the 1000 hypotheses produced by the aforementioned SMT system, its BLEU score increases to 36.5, which is close to the previous best result on this task. The LSTM also learned sensible phrase and sentence representations that are sensitive to word order and are relatively invariant to the active and the passive voice. Finally, we found that reversing the order of the words in all source sentences (but not target sentences) improved the LSTM's performance markedly, because doing so introduced many short term dependencies between the source and the target sentence which made the optimization problem easier.

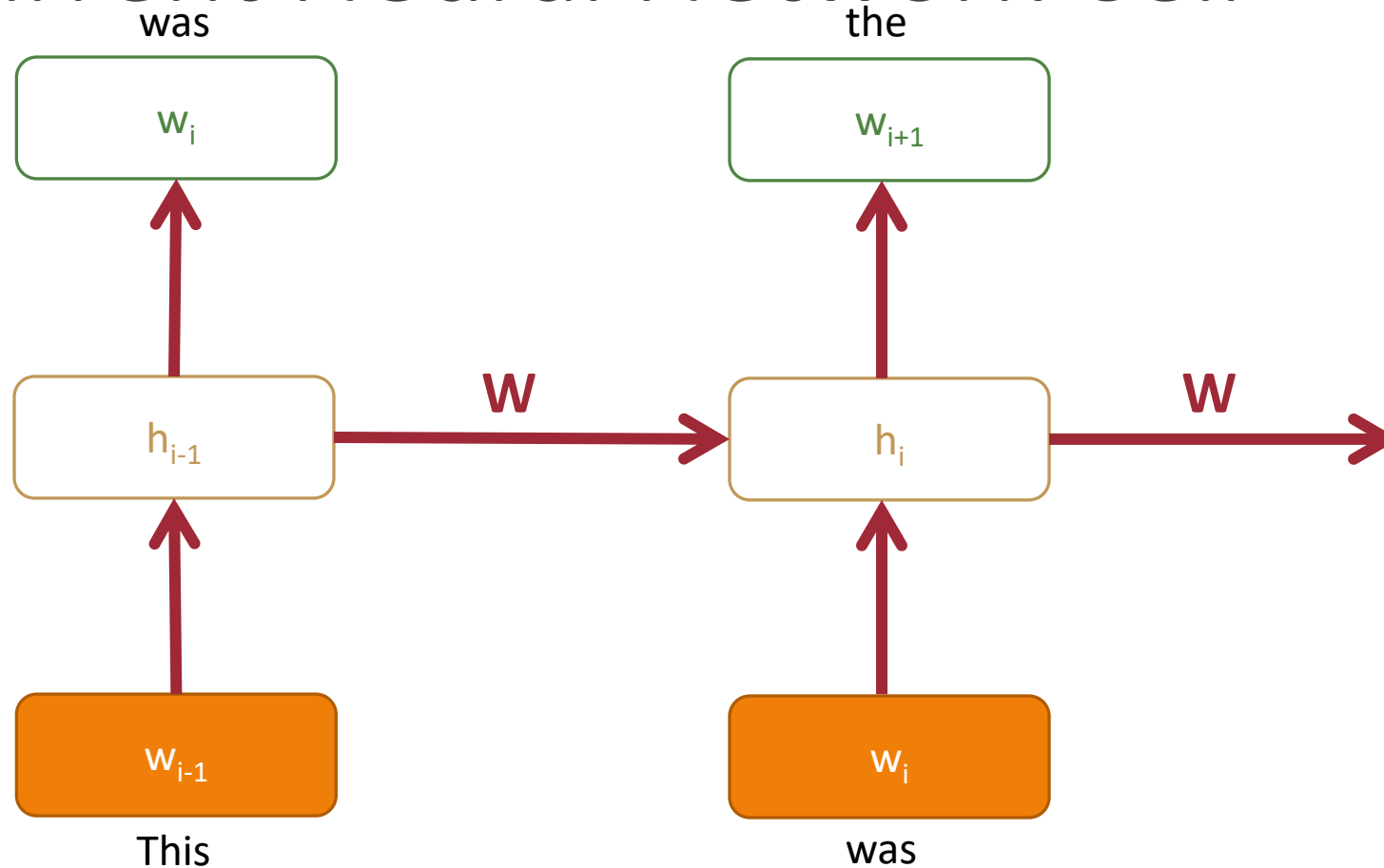
Generating Sequences With Recurrent Neural Networks

Alex Graves
Department of Computer Science
University of Toronto
graves@cs.toronto.edu

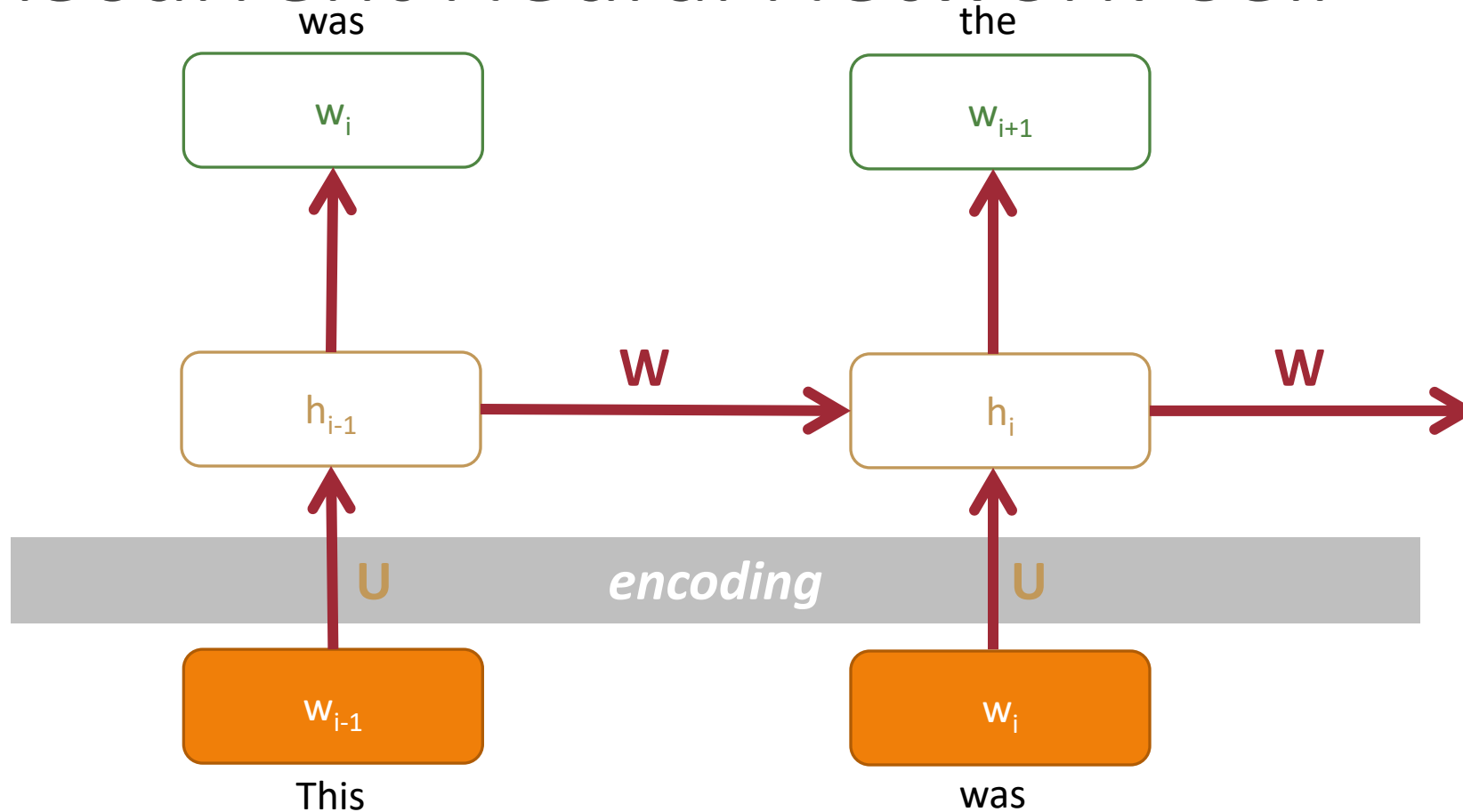
Abstract

This paper shows how Long Short-term Memory recurrent neural networks can be used to generate complex sequences with long-range structure, simply by predicting one data point at a time. The approach is demonstrated for text (where the data are discrete) and online handwriting (where the data are real-valued). It is then extended to handwriting synthesis by allowing the network to condition its predictions on a text sequence. The resulting system is able to generate highly realistic cursive handwriting in a wide variety of styles.

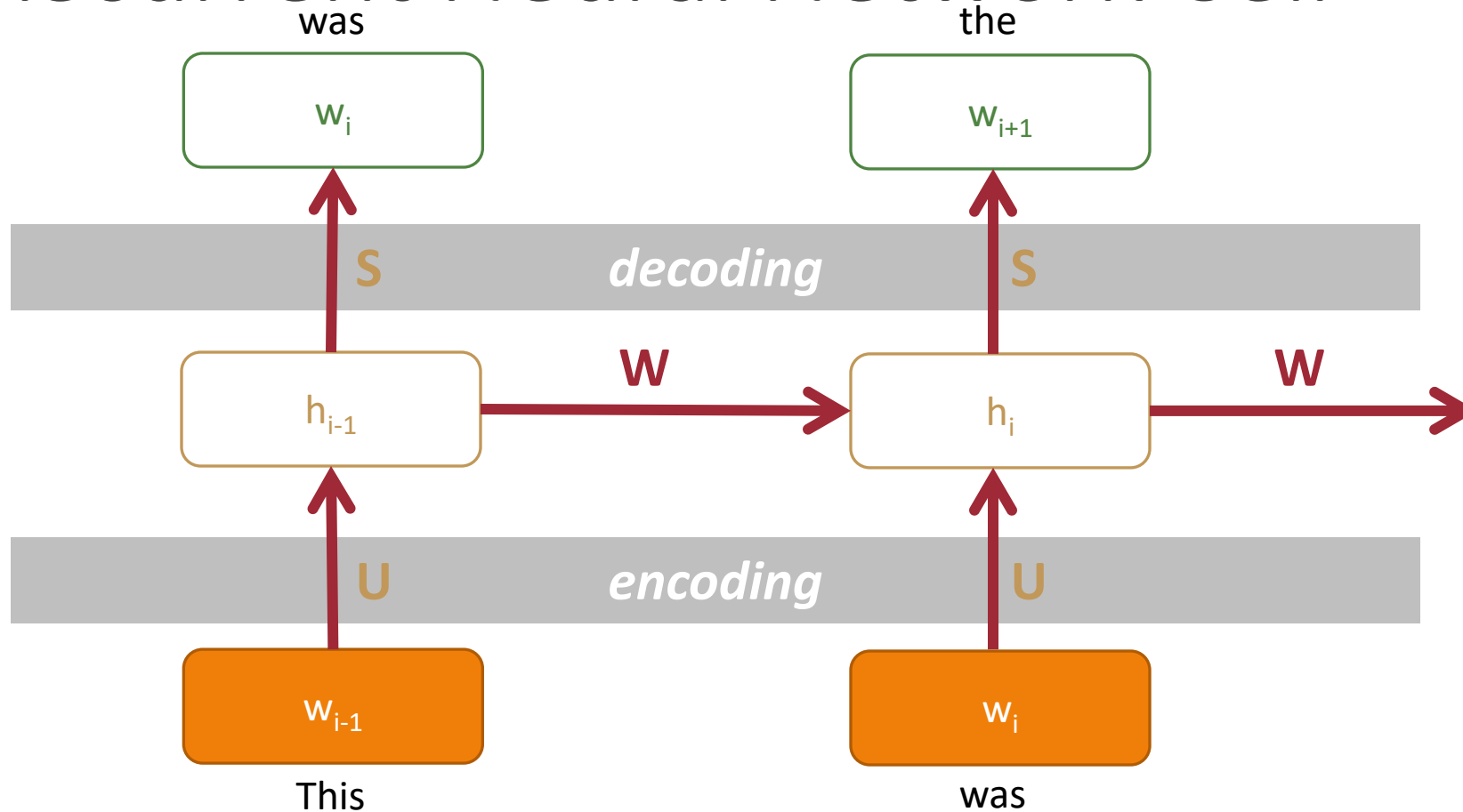
A Recurrent Neural Network Cell



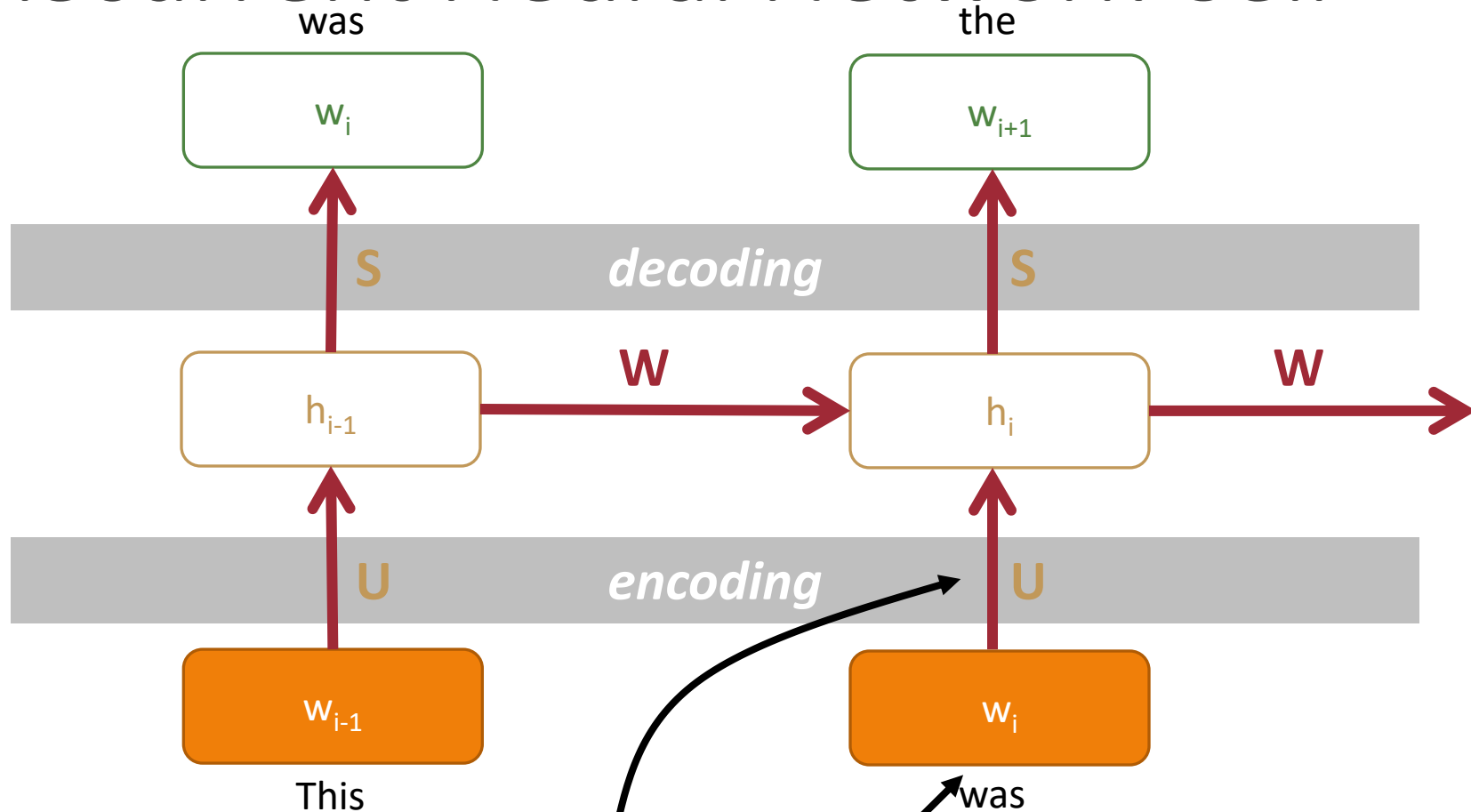
A Recurrent Neural Network Cell



A Recurrent Neural Network Cell



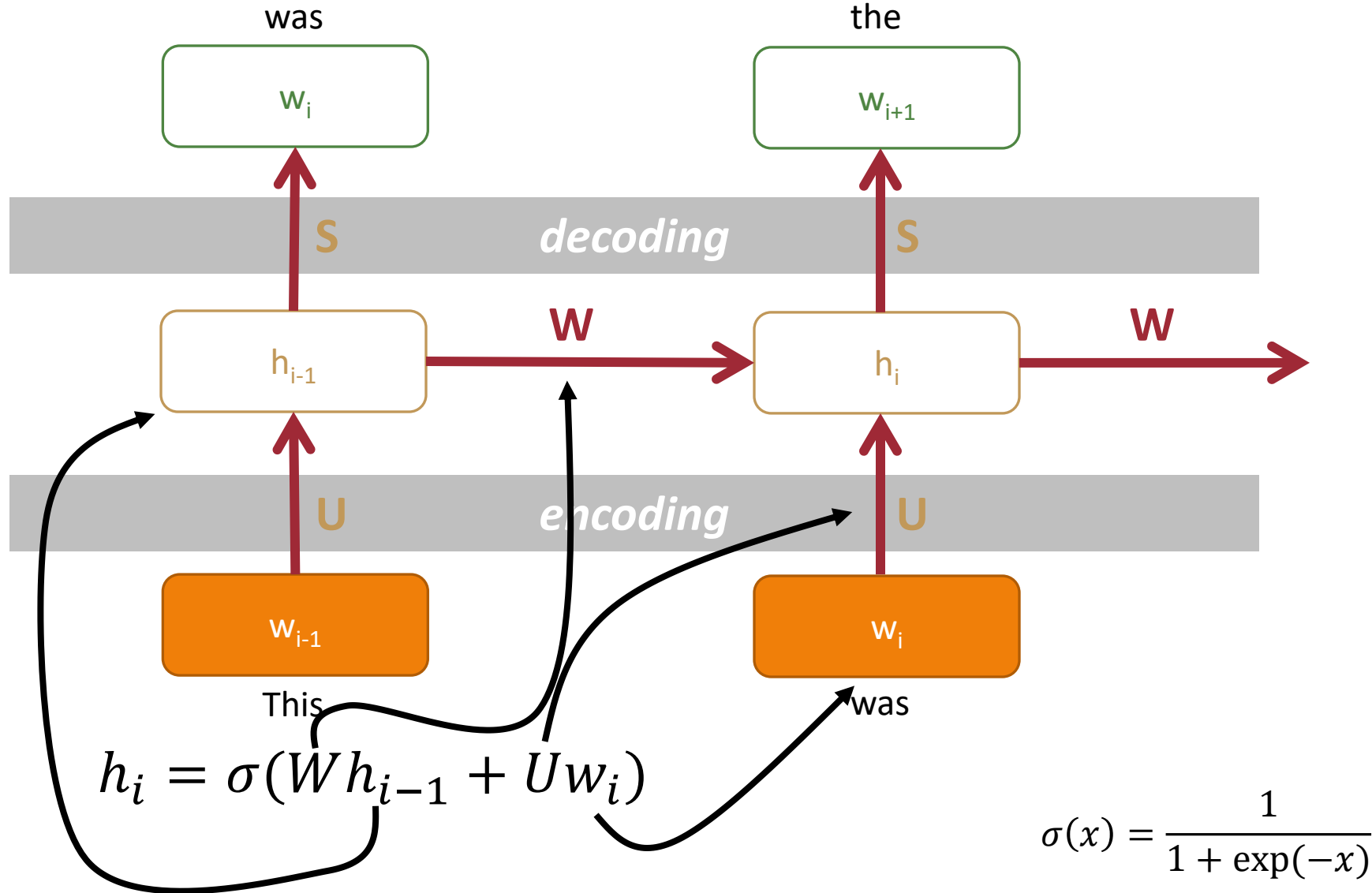
A Recurrent Neural Network Cell



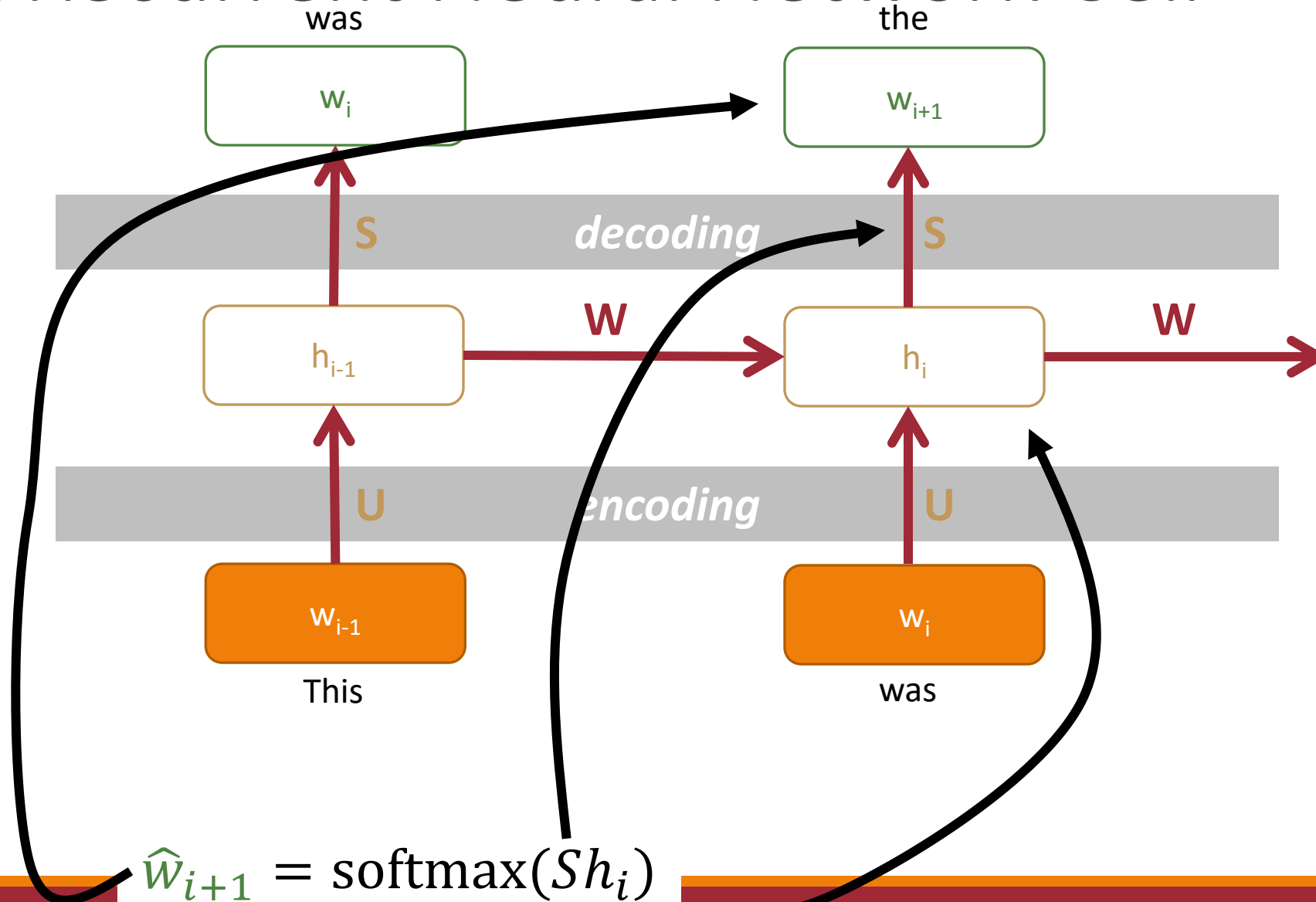
$$h_i = \sigma(W h_{i-1} + U w_i)$$

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

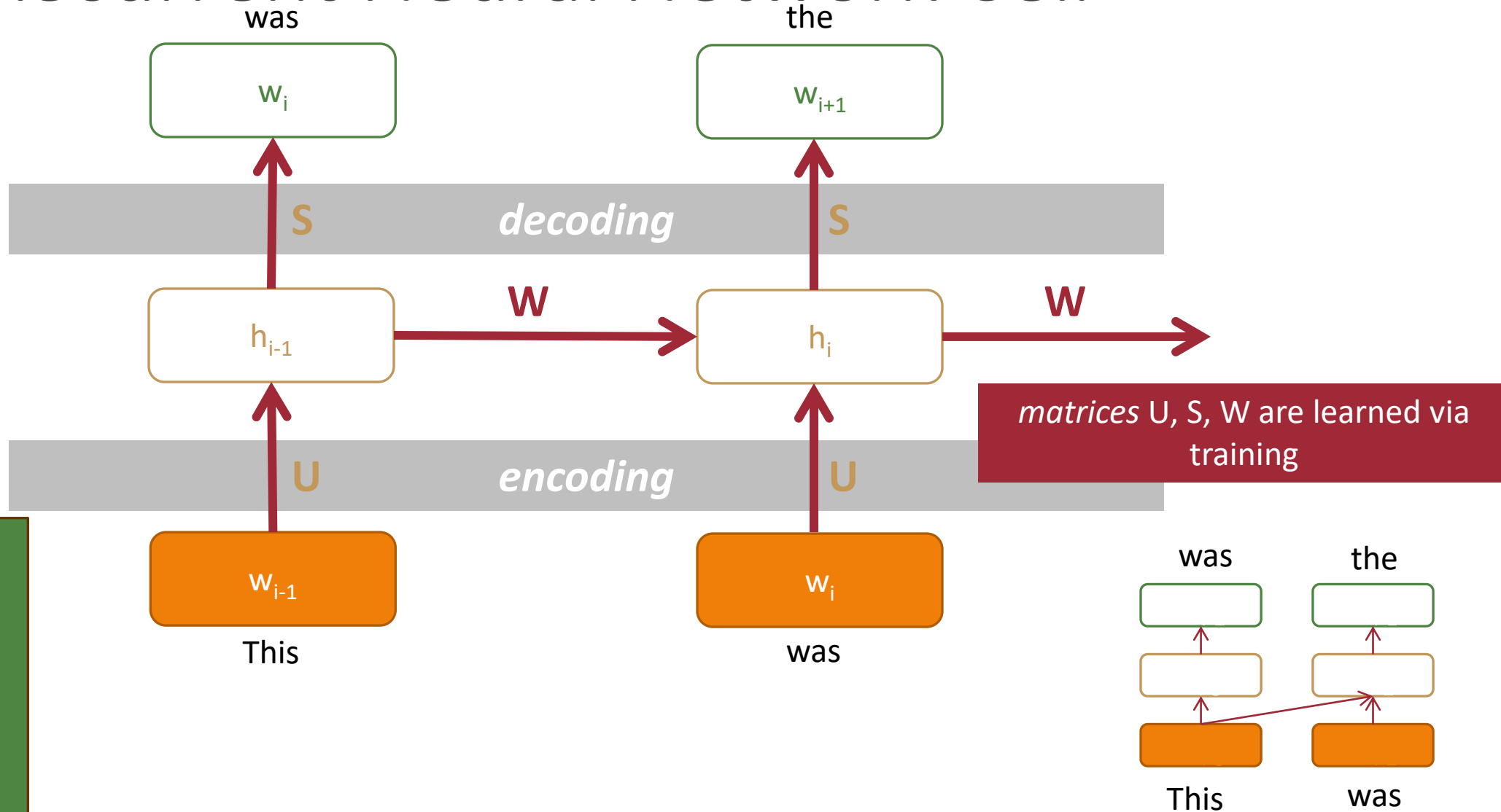
A Recurrent Neural Network Cell



A Recurrent Neural Network Cell



A Recurrent Neural Network Cell



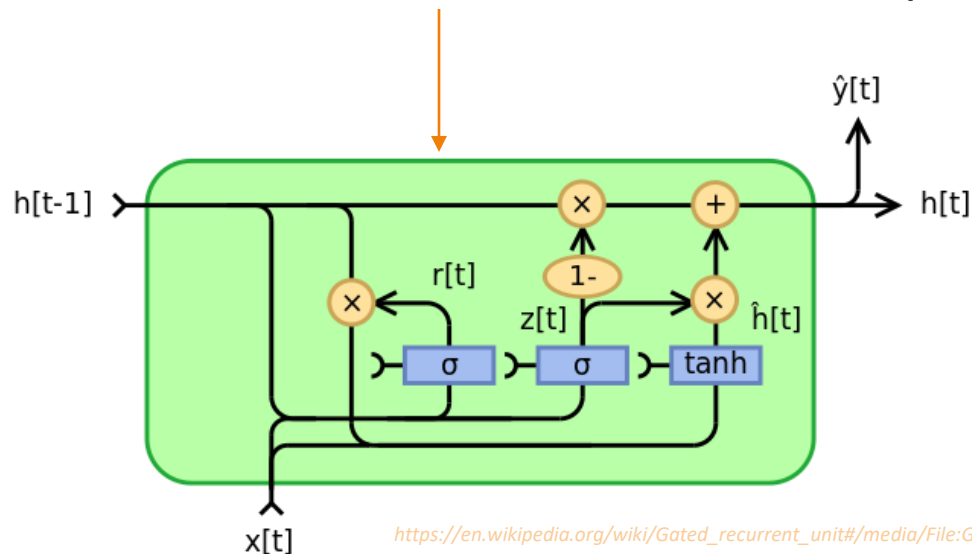
Why is an RNN considered "recurrent"?

Types of RNN cells: LSTMs/GRUs

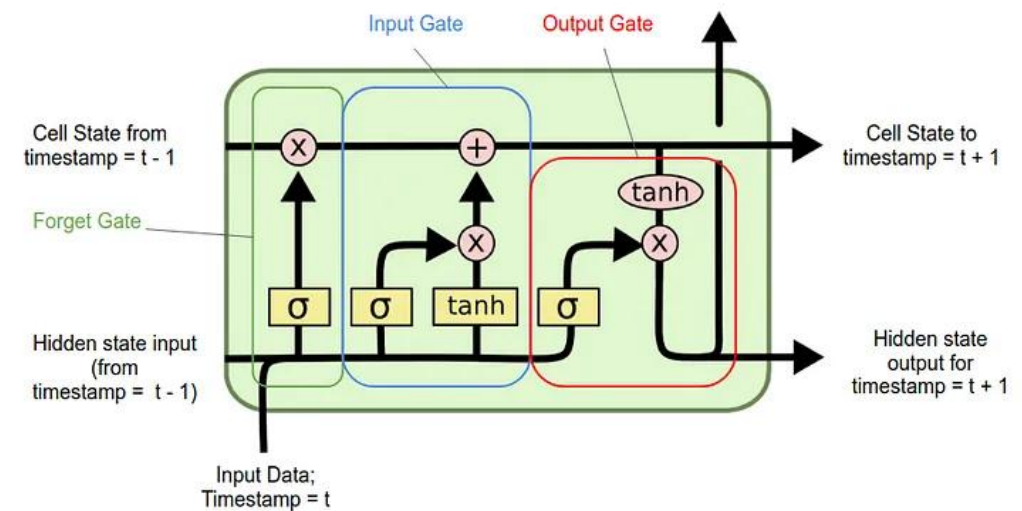
LSTM: Long Short-Term Memory (Hochreiter & Schmidhuber, 1997)

LSTMs were originally designed to keep around information for longer in the hidden state as it gets repeatedly updated.

GRU: Gated Recurrent Unit (Cho et al., 2014)

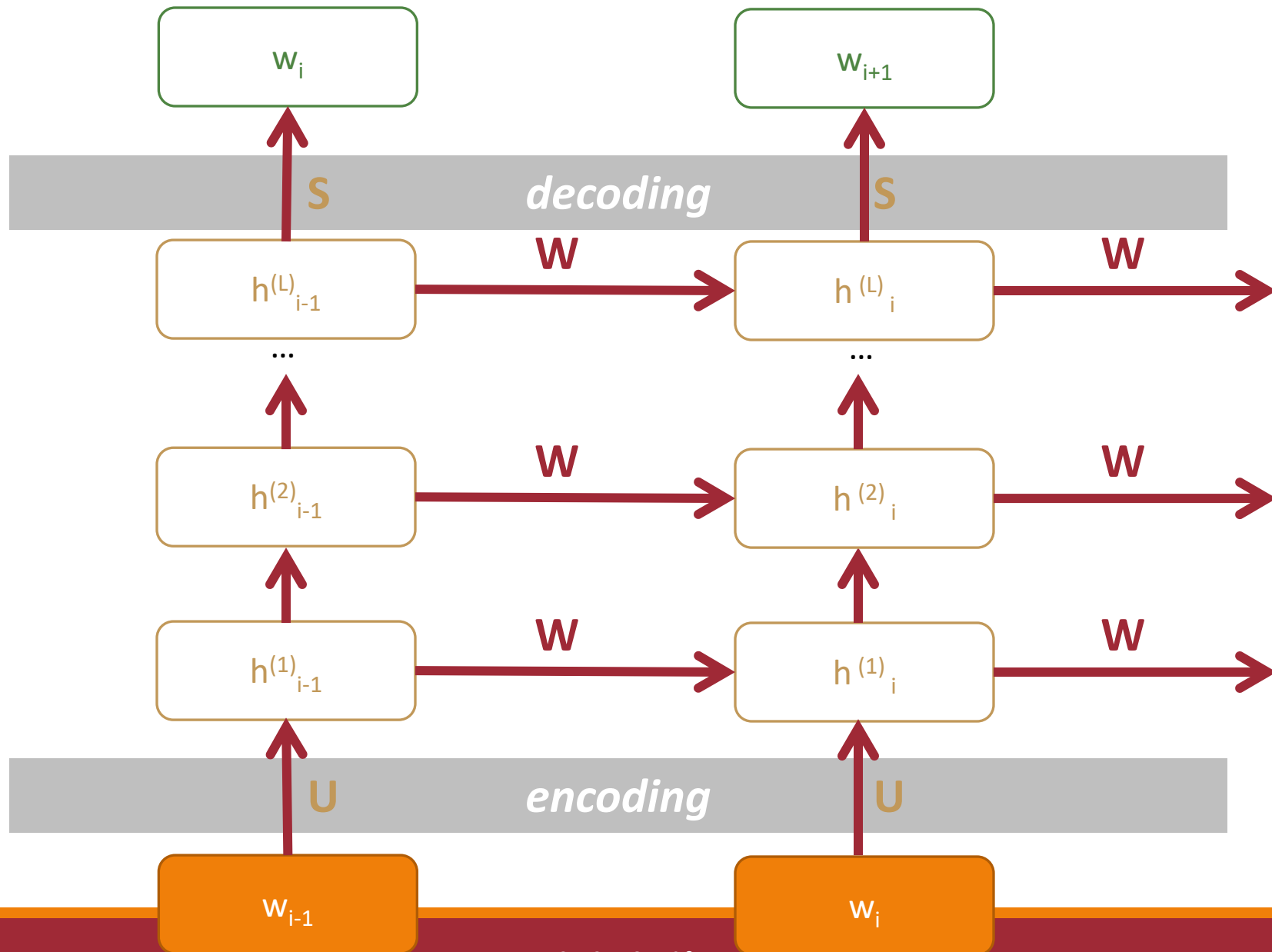


https://en.wikipedia.org/wiki/Gated_recurrent_unit#/media/File:Gated_Recurrent_Unit_base_type.svg



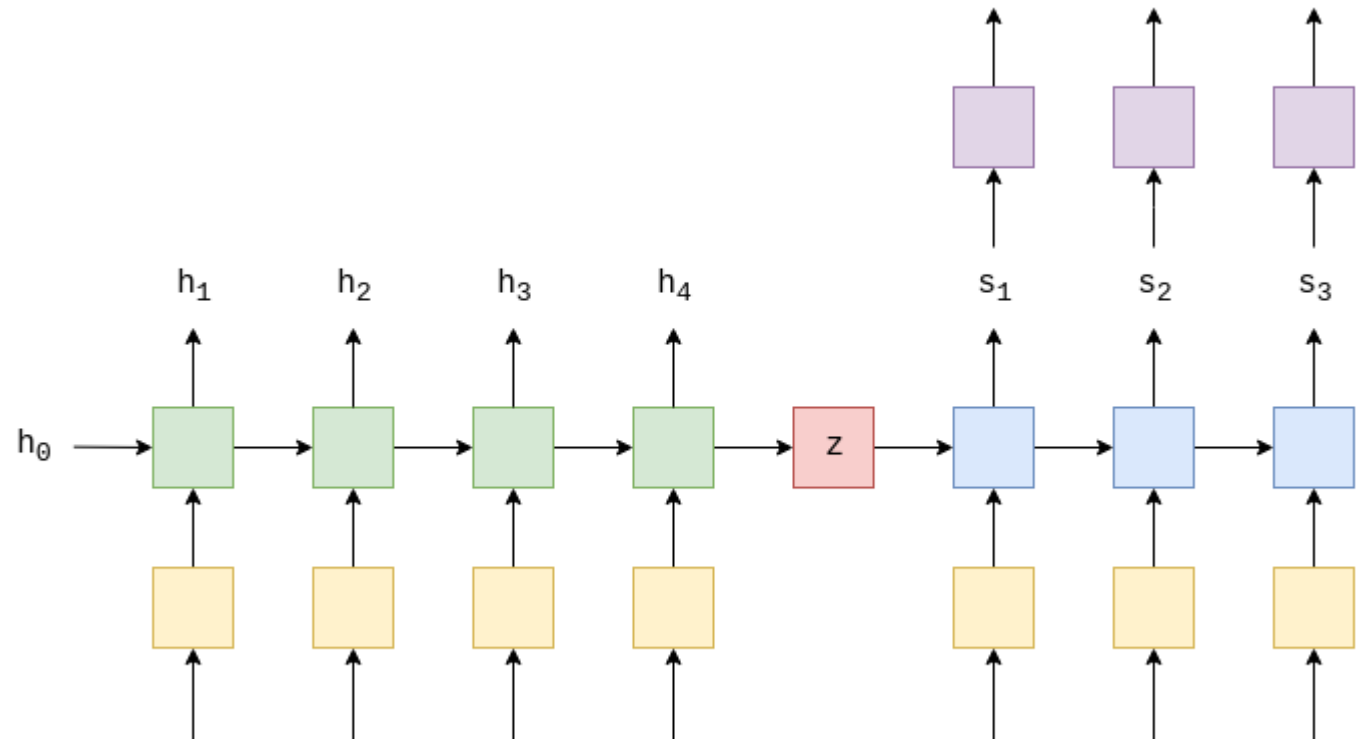
<https://medium.com/analytics-vidhya/lstms-explained-a-complete-technically-accurate-conceptual-guide-with-keras-2a650327e8f2>

A Multi-Layer Recurrent Neural Network Cell

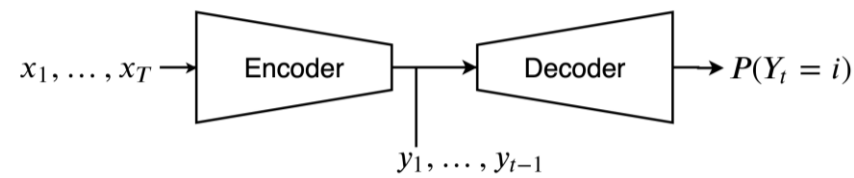


Sequence-to-Sequence / Encoder-Decoder Models

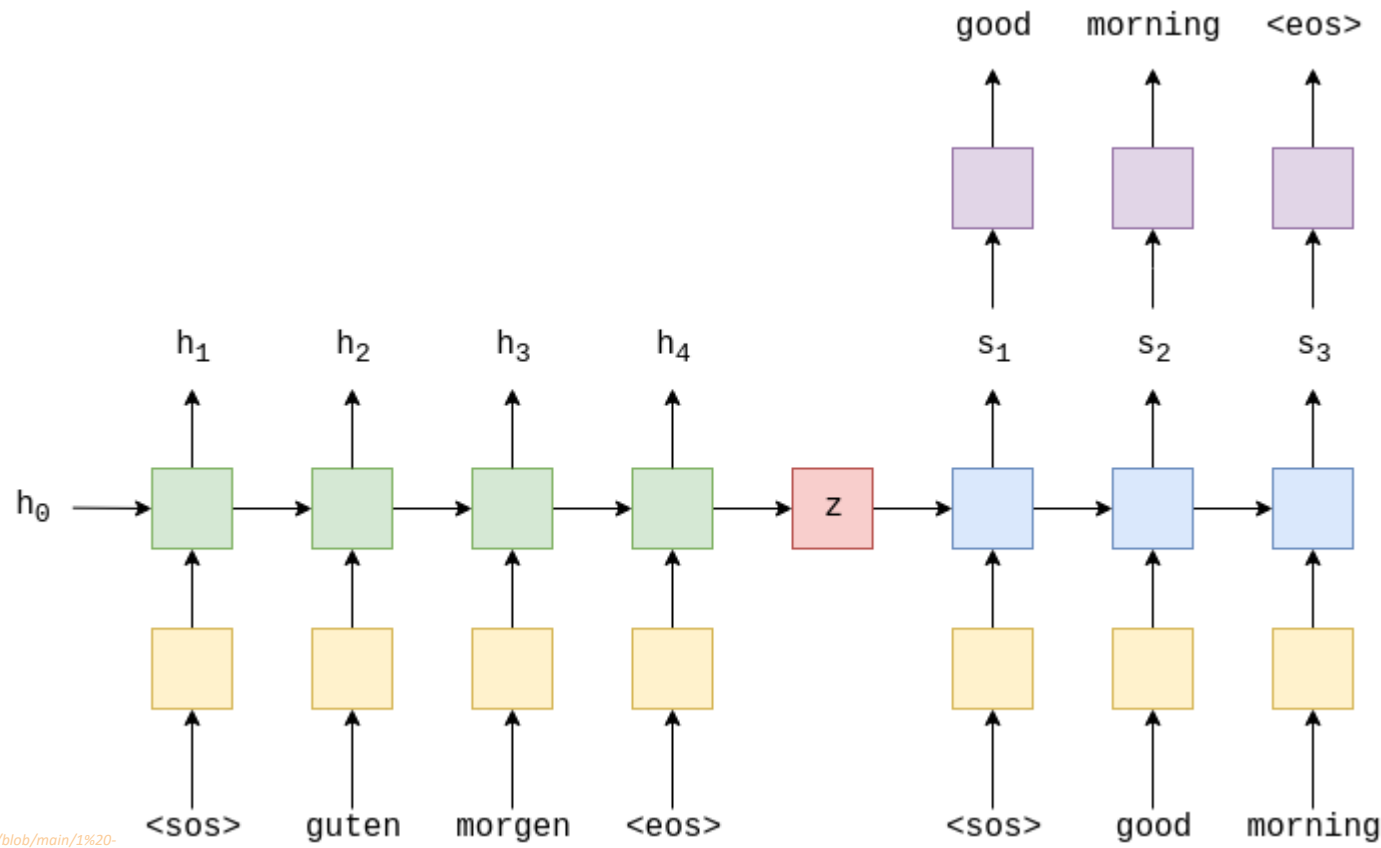
Think-pair-share:
Sequence-to-sequence models
**divided up the encoder and
decoder** components of RNNs
and
added another input to the
decoder.
Why do you think this was useful?



I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *Conference on Advances in Neural Information Processing Systems (NeurIPS)*, Montréal, Canada, 2014, pp. 3104–3112. https://proceedings.neurips.cc/paper_files/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html

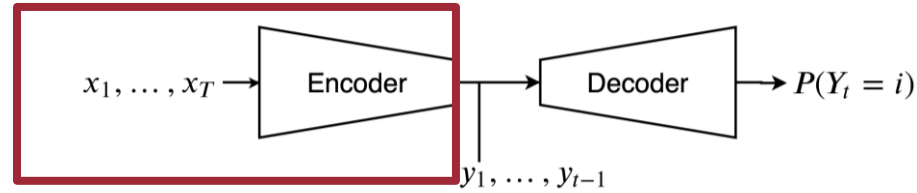


Sequence-to-Sequence / Encoder-Decoder Models



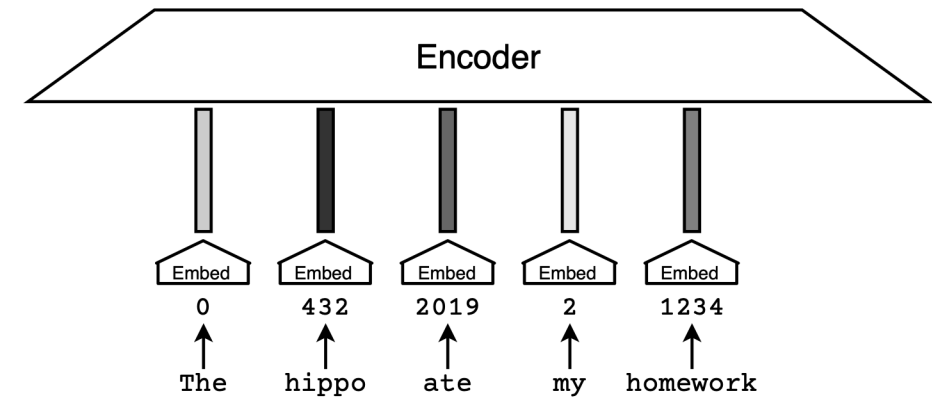
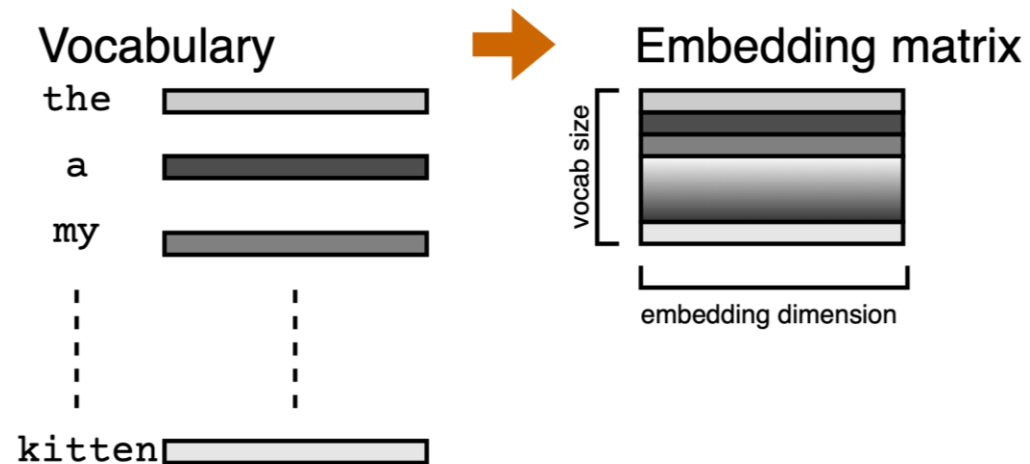
<https://colab.research.google.com/github/bentrevett/pytorch-seq2seq/blob/main/1%20-%20Sequence%20to%20Sequence%20Learning%20with%20Neural%20Networks.ipynb#scrollTo=k6sRrL4wKsmI>

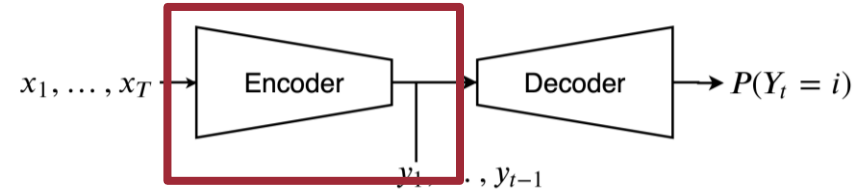
I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *Conference on Advances in Neural Information Processing Systems (NeurIPS)*, Montréal, Canada, 2014, pp. 3104–3112. https://proceedings.neurips.cc/paper_files/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html



Inputs to the Encoder

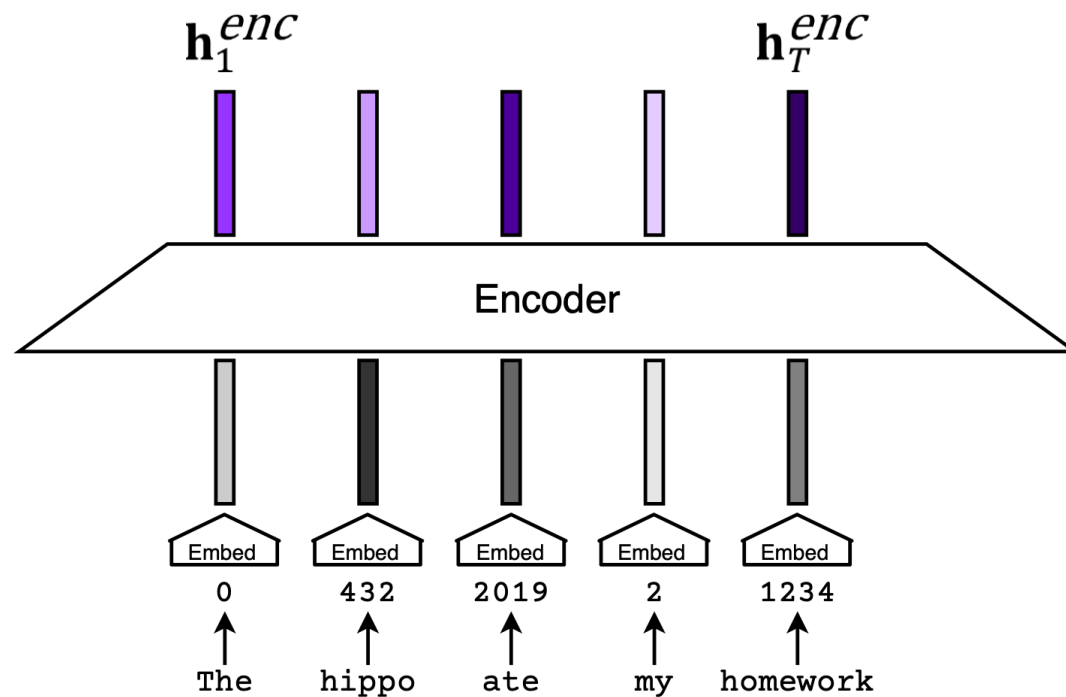
The encoder takes as input the embeddings corresponding to each token in the sequence.

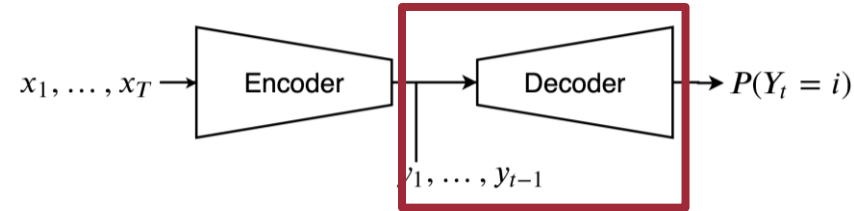




Outputs from the Encoder

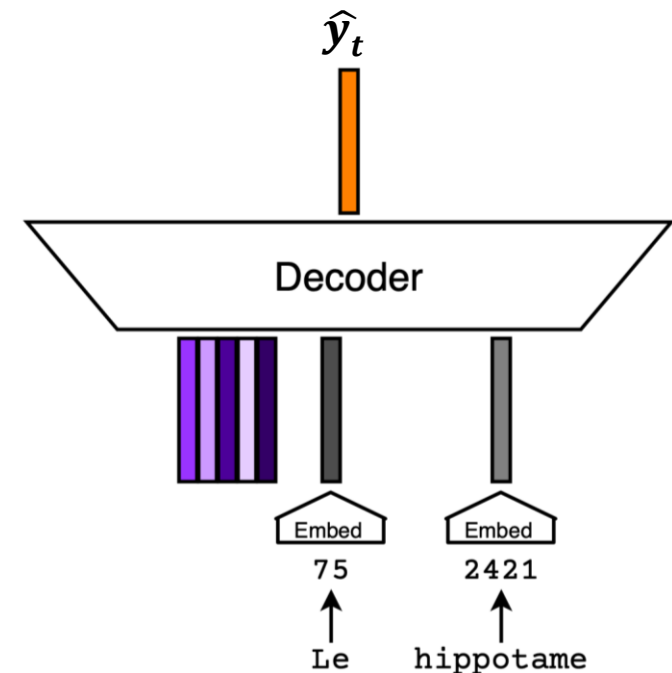
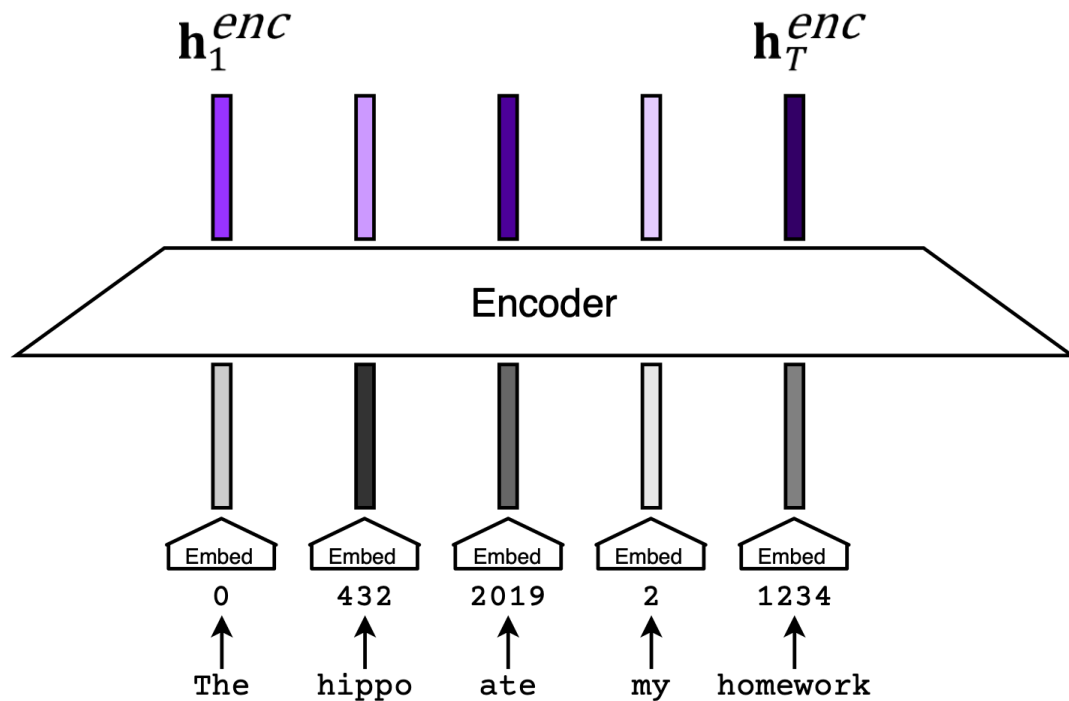
The encoder outputs a sequence of vectors. These are called the hidden state of the encoder.

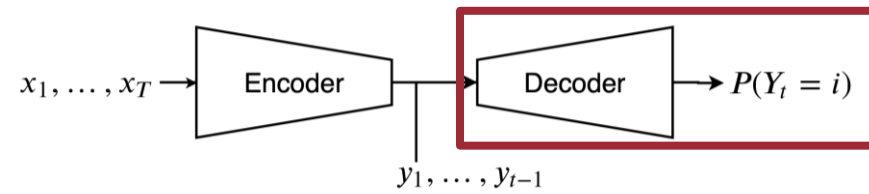




Inputs to the Decoder

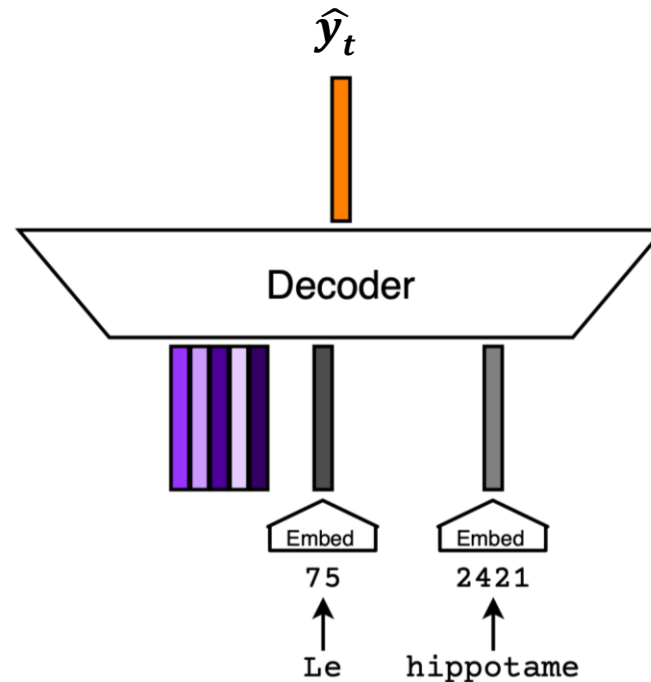
The decoder takes as input the hidden states from the encoder as well as the embeddings for the tokens seen so far in the target sequence.





Outputs from the Decoder

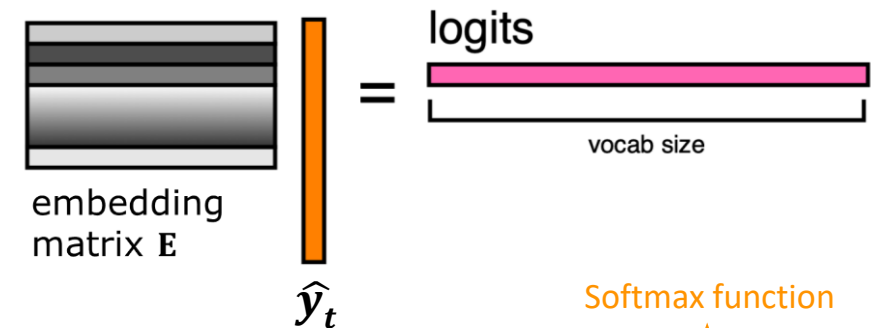
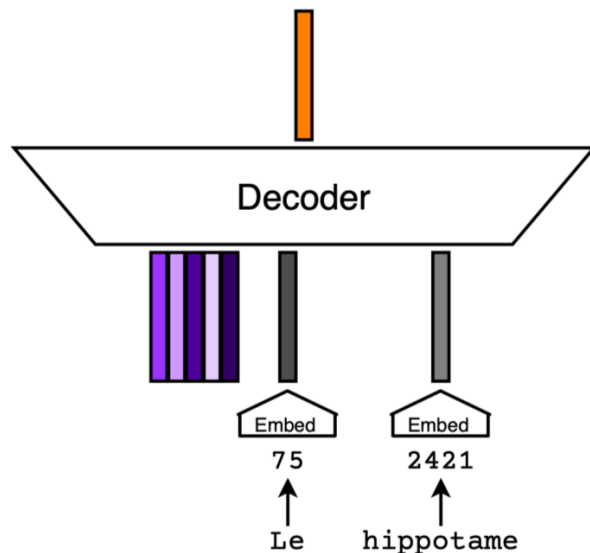
The decoder outputs an embedding $\hat{y}_t$. The goal is for this embedding to be as close as possible to the embedding of the true next token.



Turning $\hat{\mathbf{y}}_t$ into a Probability Distribution

We can multiply the predicted embedding $\hat{\mathbf{y}}_t$ by our vocabulary embedding matrix to get a score for each vocabulary word. These scores are referred to as logits.

The **softmax function** then lets us turn the logits into probabilities.



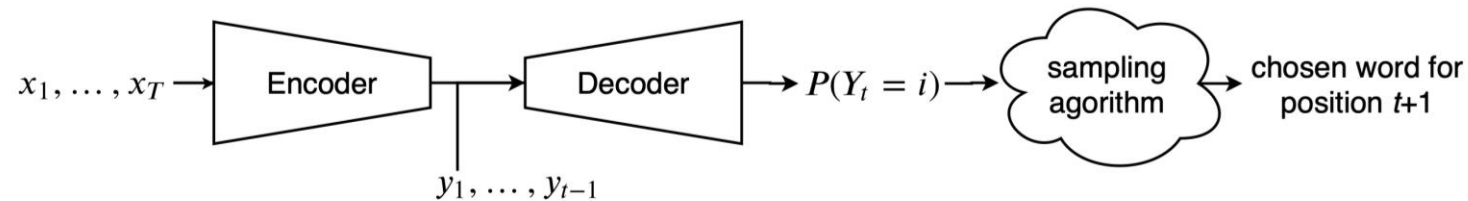
Softmax function

$$P(Y_t = i | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1}) = \frac{\exp(\mathbf{E}\hat{\mathbf{y}}_{t[i]})}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_{t[j]})}$$

Generating Text

Also sometimes called decoding 🙄

To generate text, we need an algorithm that selects tokens given the predicted probability distributions.



More on this
in the next
lecture!

Loss Function

$$\mathcal{L} = - \sum_{t=1}^T \log P(Y_t = i^* | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})$$

The index of the true
 t th word in the target
sequence.

Loss Function

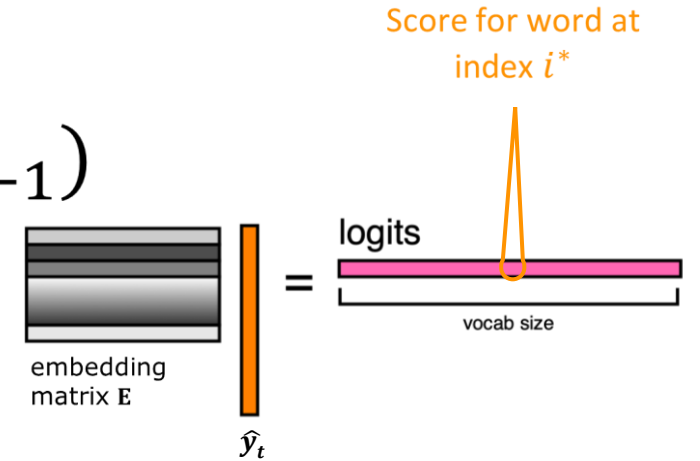
$$\mathcal{L} = - \sum_{t=1}^T \log P(Y_t = i^* | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})$$

The probability the language model assigns to the true t th word in the target sequence.

Loss Function

$$\mathcal{L} = - \sum_{t=1}^T \log P(Y_t = i^* | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})$$

$$= - \sum_{t=1}^T \log \frac{\exp(\mathbf{E} \hat{\mathbf{y}}_t[i^*])}{\sum_j \exp(\mathbf{E} \hat{\mathbf{y}}_t[j])}$$



$$P(Y_t = i | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1}) = \frac{\exp(\mathbf{E} \hat{\mathbf{y}}_t[i])}{\sum_j \exp(\mathbf{E} \hat{\mathbf{y}}_t[j])}$$

Loss Function

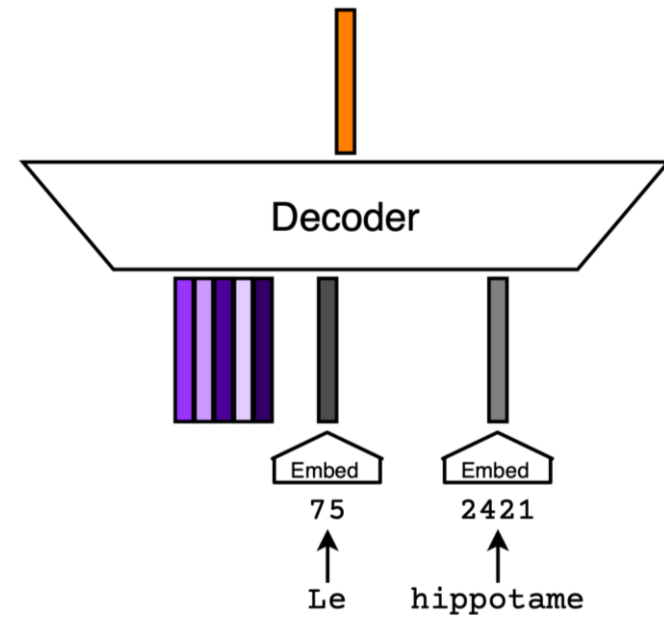
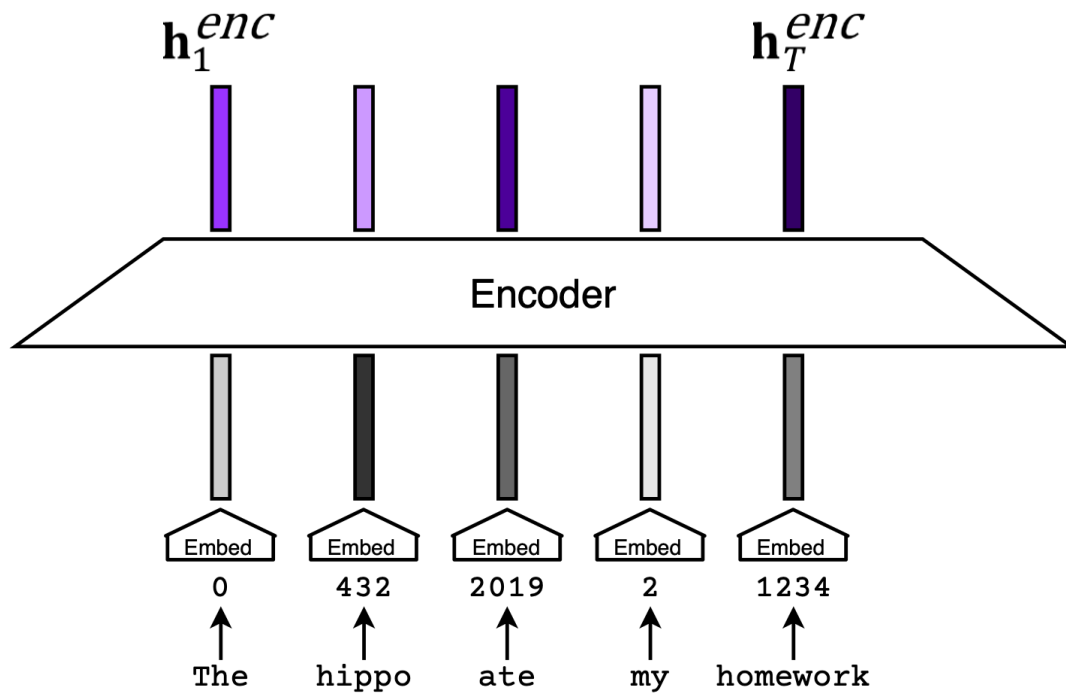
$$\mathcal{L} = - \sum_{t=1}^T \log P(Y_t = i^* | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})$$

$$= - \sum_{t=1}^T \log \frac{\exp(\mathbf{E} \mathbf{y}_t[i^*])}{\sum_j \exp(\mathbf{E} \hat{\mathbf{y}}_t[j])}$$

$$= - \sum_{t=1}^T \mathbf{E} \hat{\mathbf{y}}_t[i^*]$$

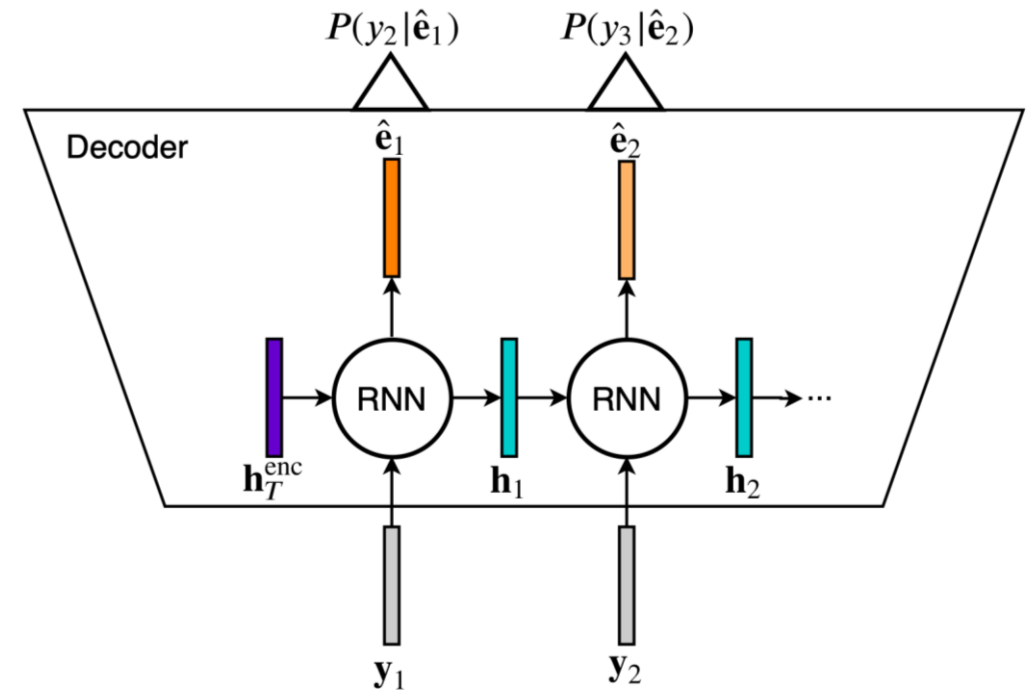
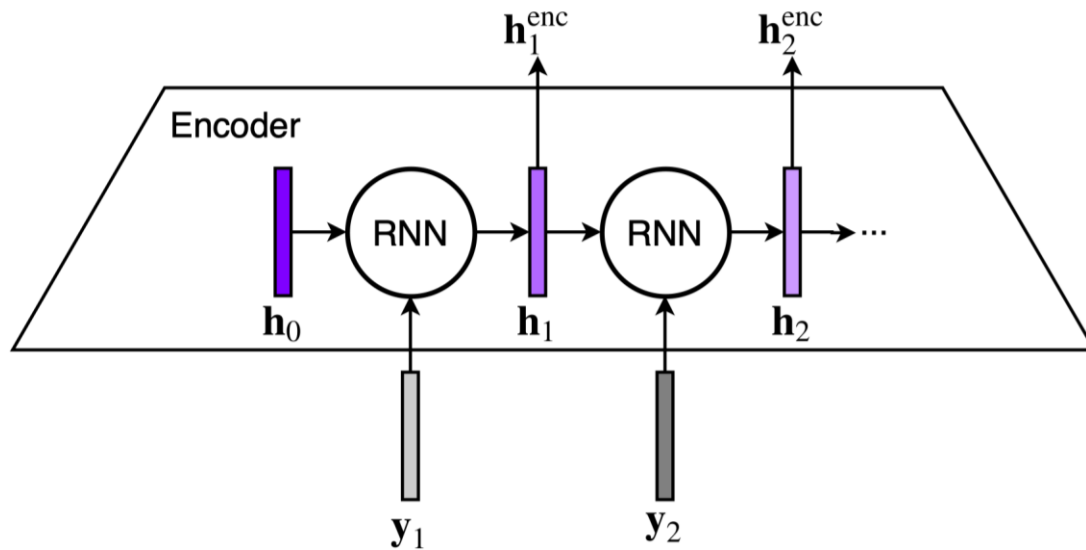
RNN Encoder-Decoder Architectures

How might we combine the two parts to make an encoder-decoder model?



RNN Encoder-Decoder Architectures

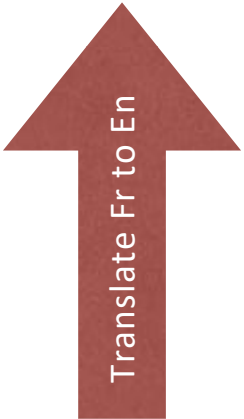
Simplest approach: Use the final hidden state from the encoder to initialize the first hidden state of the decoder.



RNN Encoder-Decoder Architectures

[The, hippopotamus, ...

When predicting the next English word, how much weight should the model put on each French word in the source sequence?

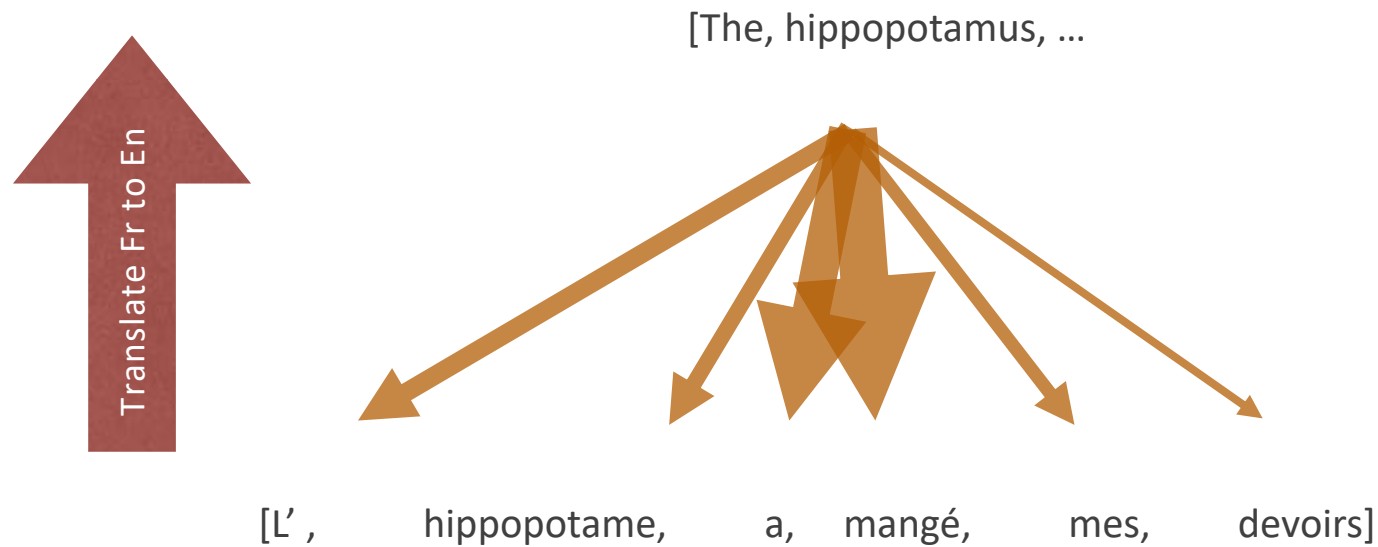


Translate Fr to En

[L', hippopotame, a, mangé, mes, devoirs]

Attention

Better approach: an attention mechanism



Compute a linear combination of the encoder hidden states.

$$\mathbf{c}_t = \alpha_1 \mathbf{h}_1 + \alpha_2 \mathbf{h}_2 + \alpha_3 \mathbf{h}_3 + \dots + \alpha_T \mathbf{h}_T$$

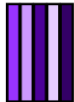
Decoder's prediction at position t is based on both the context vector and the hidden state outputted by the RNN at that position.

$$\hat{\mathbf{e}}_t = f_{\theta}(\mathbf{h}_t^{\text{dec}} \parallel \mathbf{c}_t)$$

RNN Encoder-Decoder Architectures

The t^{th} context vector is computed as $\mathbf{c}_t = \mathbf{H}^{\text{enc}} \mathbf{a}_t$

$$\mathbf{a}_t[i] = \text{softmax}(\text{att_score}(\mathbf{h}_t^{\text{dec}}, \mathbf{h}_i^{\text{enc}}))$$

There are a few different options for the attention score: $\mathbf{H}^{\text{enc}} =$ 

Compute a linear combination of the encoder hidden states.

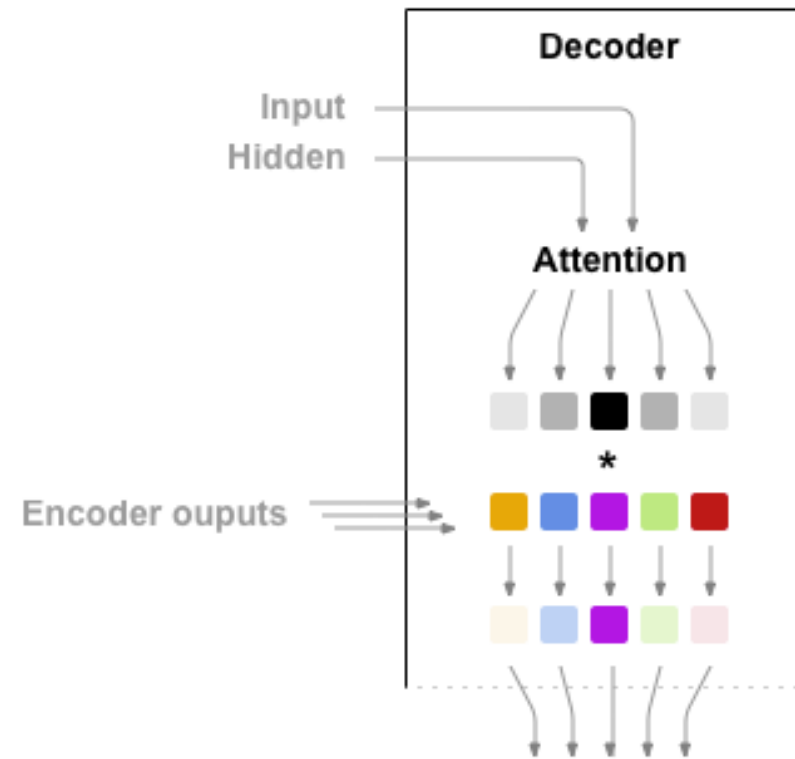
$$\mathbf{c}_t = \alpha_1 \mathbf{h}_1^{\text{enc}} + \alpha_2 \mathbf{h}_2^{\text{enc}} + \alpha_3 \mathbf{h}_3^{\text{enc}} + \dots + \alpha_T \mathbf{h}_T^{\text{enc}}$$

Decoder's prediction at position t is based on both the context vector and the hidden state outputted by the RNN at that position.

$$\hat{\mathbf{e}}_t = f_{\theta}(\mathbf{h}_t^{\text{dec}} \parallel \mathbf{c}_t)$$

$$\text{att_score}(\mathbf{h}_t^{\text{dec}}, \mathbf{h}_i^{\text{enc}}) = \begin{cases} \mathbf{h}_t^{\text{dec}} \cdot \mathbf{h}_i^{\text{enc}} & \text{dot product} \\ \mathbf{h}_t^{\text{dec}} \mathbf{W}_a \mathbf{h}_i^{\text{enc}} & \text{bilinear function} \\ w_{a1}^{\top} \tanh(\mathbf{W}_a \mathbf{2} [\mathbf{h}_t^{\text{dec}}, \mathbf{h}_i^{\text{enc}}]) & \text{MLP} \end{cases}$$

Attention Decoder



https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

Limitations of Recurrent architecture

Slow to train.

- Can't be easily parallelized because of the recurrence
- The computation at position t is dependent on first doing the computation at position $t-1$.

Difficult to access information from many steps back.

- If two tokens are K positions apart, there are K opportunities for knowledge of the first token to be erased from the hidden state before a prediction is made at the position of the second token.

Seq2Seq Output (2017)

R2-D2 carrying some drinks on a tray strapped to his back passes Yoda who uses his force powers to hog the drinks

Expected:

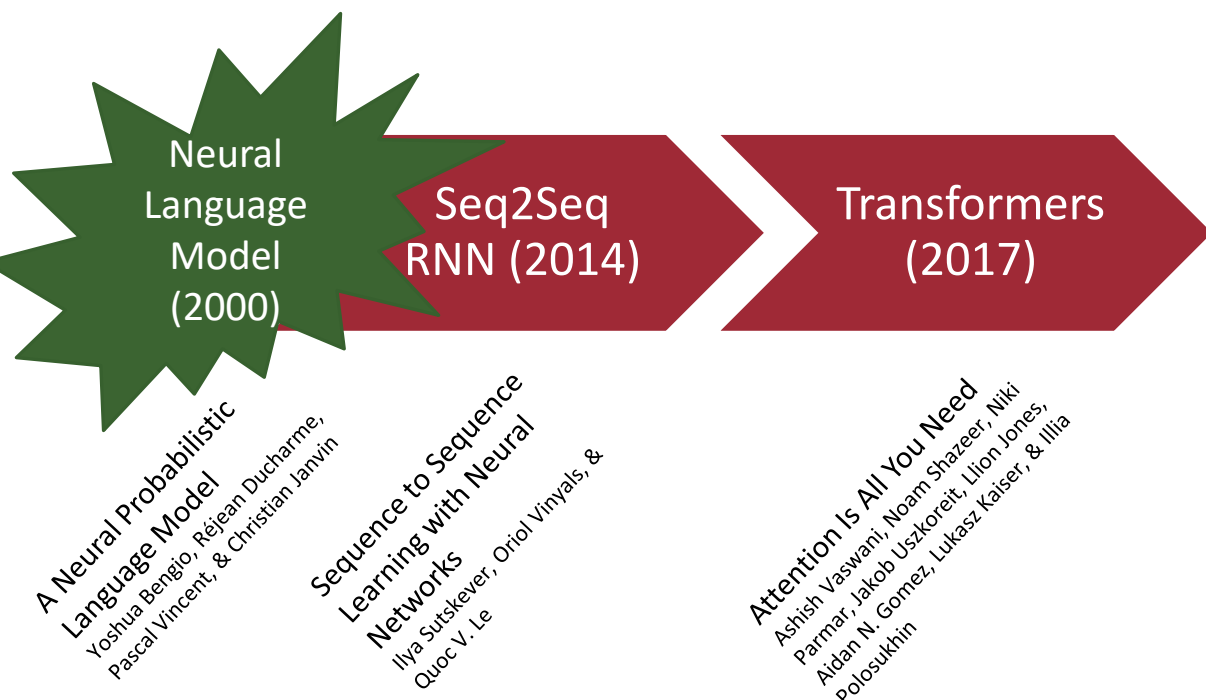
Obi Wan and Anakin are drinking happily when Chewbacca takes a Polaroid picture of Anakin and Obi Wan

Predicted:

Can this block gives him the advantage to personally run around with a large stick of cheese



Neural Language Model Timeline



Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Lukasz Kaiser*
Google Brain
lukaszkaier@google.com

Illia Polosukhin* †
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

Transformers

The Transformer is a **non-recurrent** non-convolutional (feed-forward) neural network designed for language understanding

- introduces self-attention in addition to encoder-decoder attention

