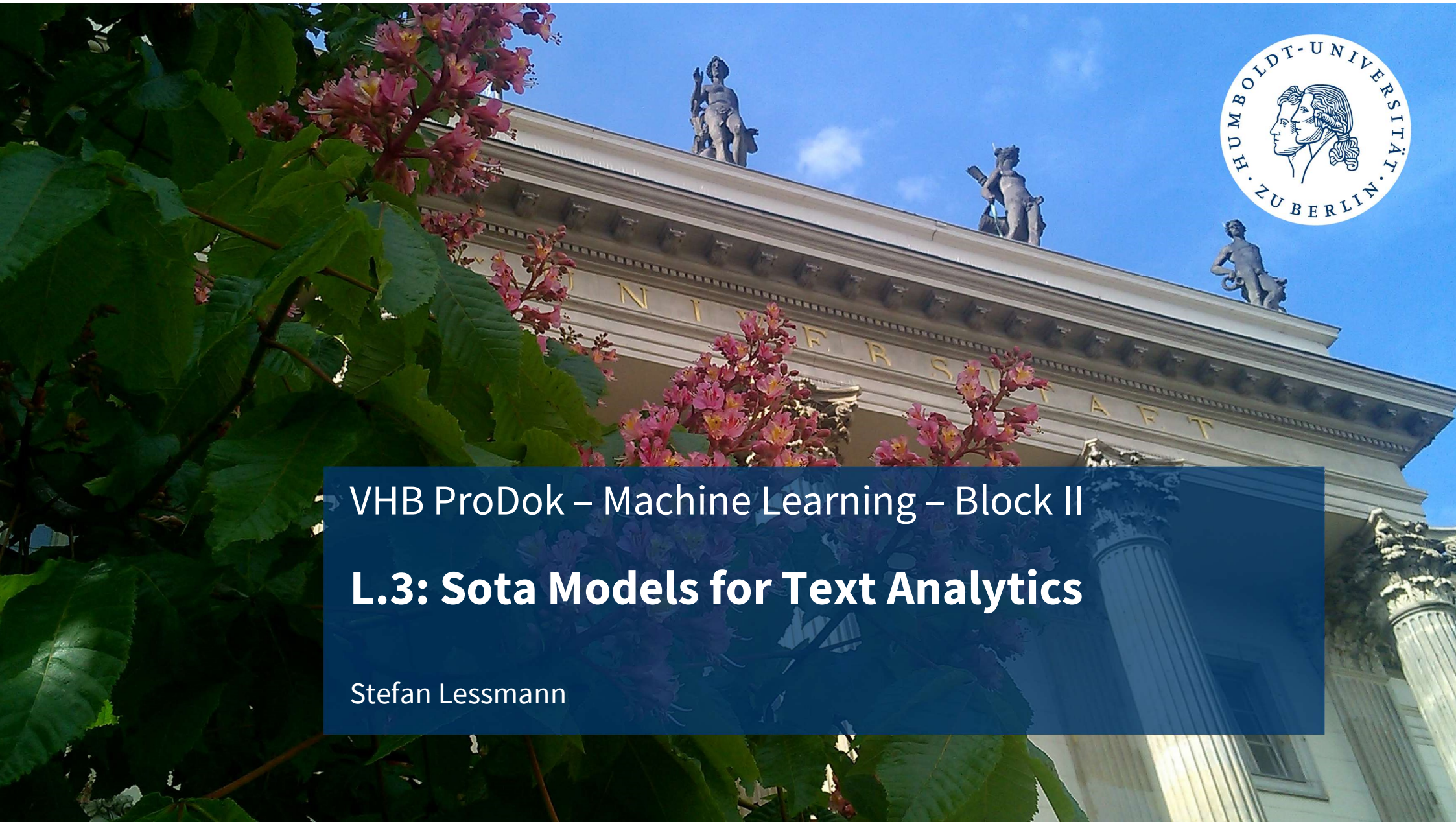
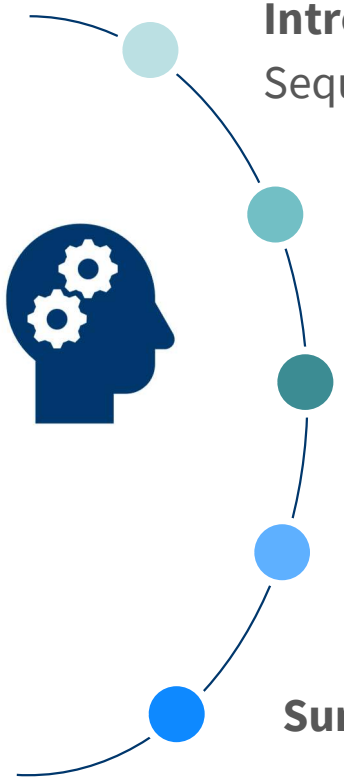


VHB ProDok – Machine Learning – Block II

L.3: Sota Models for Text Analytics

Stefan Lessmann

Agenda



Introduction
Sequential data, statistical techniques, and language models

Recurrent Neural Networks (RNN)
Recurrent cells and language models, training, and the vanishing gradient problem

Gated RNNs for the rescue
Design idea, gated cells, GRM & LSTM, and extensions

NLP Transfer Learning and State-of-the-Art
Attention mechanism and transformer architecture

Summary

Introduction

Sequential data, statistical techniques, and language models

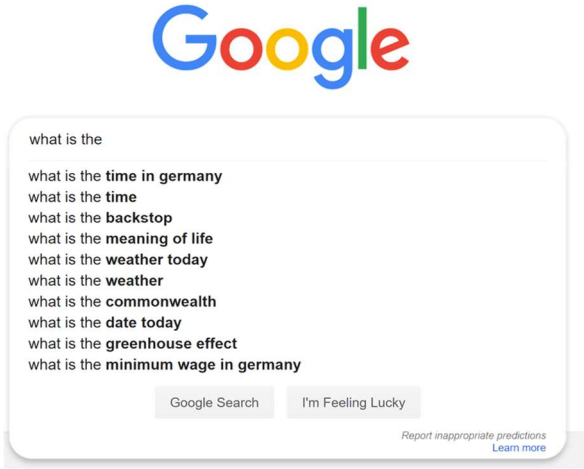
An Introduction to Sequential Data

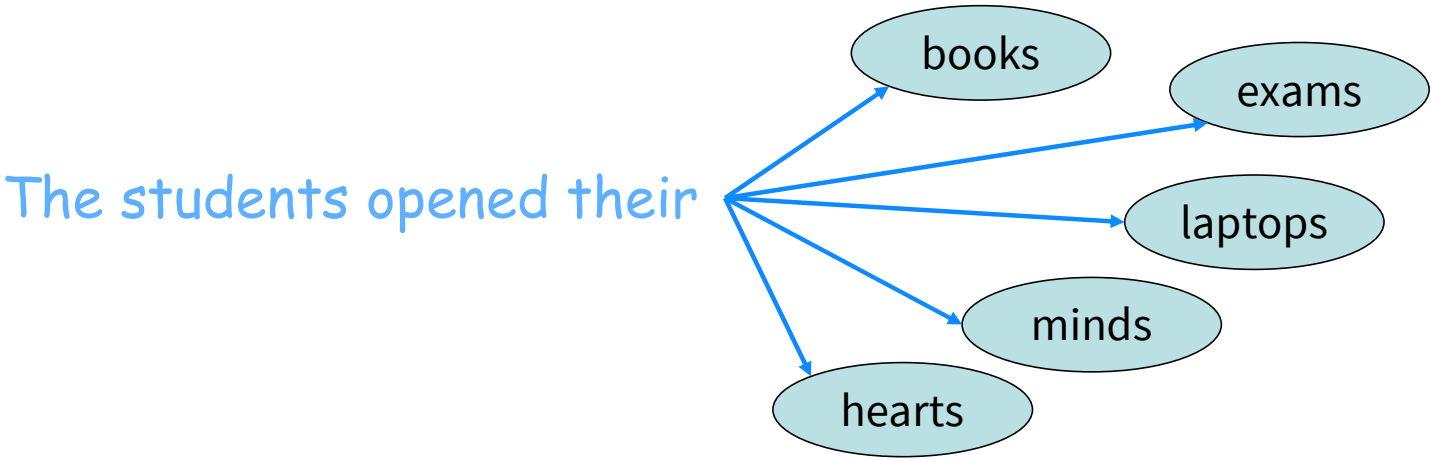
Time-ordered data is omnipresent

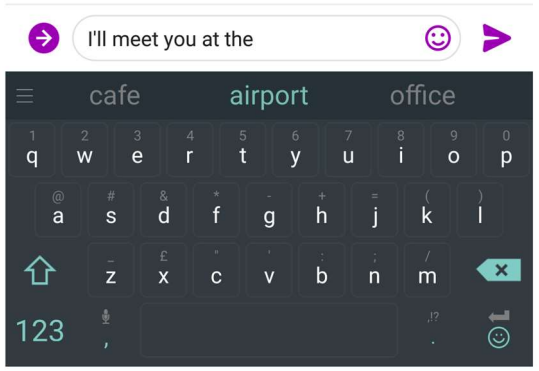
Language Models

The task of predicting the next word in a sequence

- **Language models are a cornerstone of modern NLP**
- **Large language models (LLMs) are a cornerstone of AI**
- **NLP transfer learning relies on language models**







An Introduction to Sequential Data

A more formal perspective

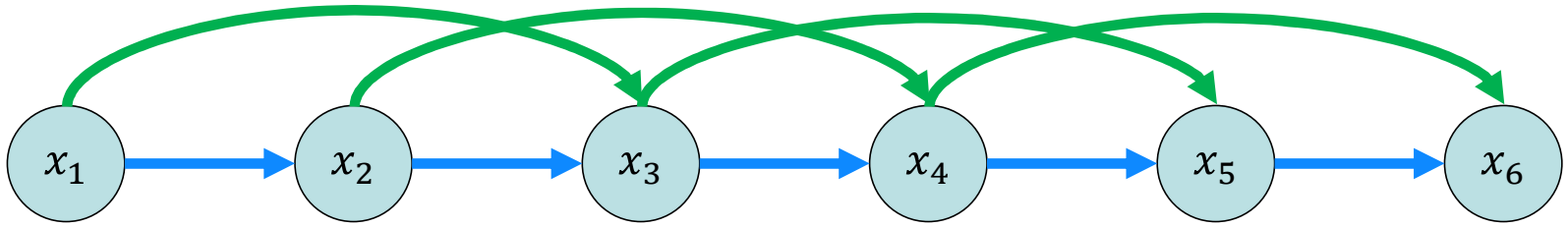
- **Modeling task**
  - Estimate of $p(x_1, \ldots, x_T) = \prod_{t=1}^{T} p(x_t | x_{t-1}, x_{t-2}, \ldots, x_1)$
  - Where x_t denotes the realization of random variable X at time t
  - Just like a word (or token) in a sentence
- **Modeling challenges**
  - Temporal dependencies violate the IID assumption
    - The value of observation x_t depends on previous realizations x_{t-1}
    - Therefore, there is no independence between x_t and x_{t-1}
    - Many statistical models depend on the IID assumption
  - When estimating $x_t \sim p(x_t | x_{t-1}, x_{t-2}, \ldots, x_1)$, the length of the sequence varies with t

Statistical Tools for Sequential Data

Plan A: Markov Assumption

- **Maybe we do not need all past observations** $x_{t-1}, x_{t-2}, \dots, x_1$
  - Using only the last τ observations ensures fixed input length for all $t > \tau$
  - Parameter τ represents a believe of the dependence structure

$$p(x) = p(x_1) \cdot p(x_2|x_1) \cdot p(x_3|x_1, x_2) \cdot \dots \cdot p(x_t|x_{t-\tau}, \dots, x_{t-1})$$



First order Markov model

Second order Markov model

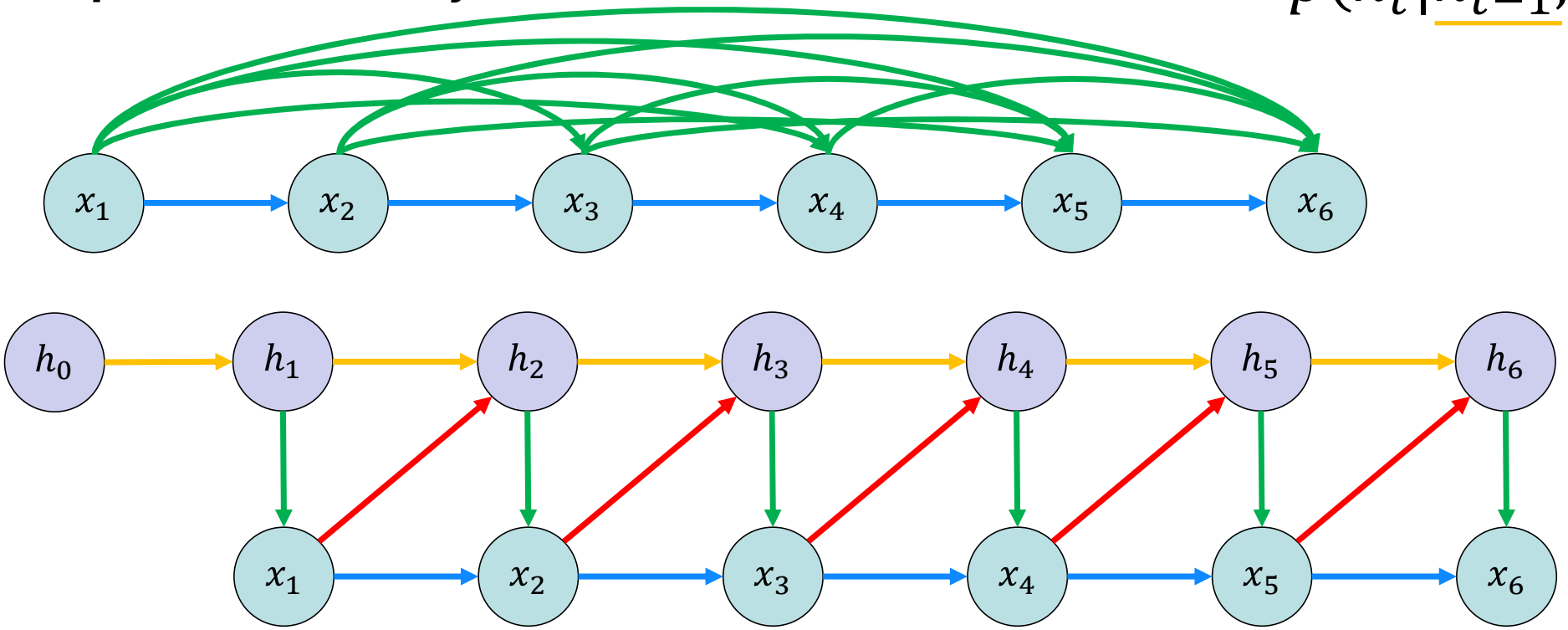
Statistical Tools for Sequential Data

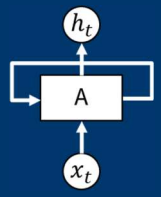
Plan B: Latent variable or state space model

- **Keep some memory h_t of past observations**
- **Update the memory when new information arrives**

$$p(x_t|h_t, x_{t-1})$$

$$p(h_t|h_{t-1}, x_{t-1})$$





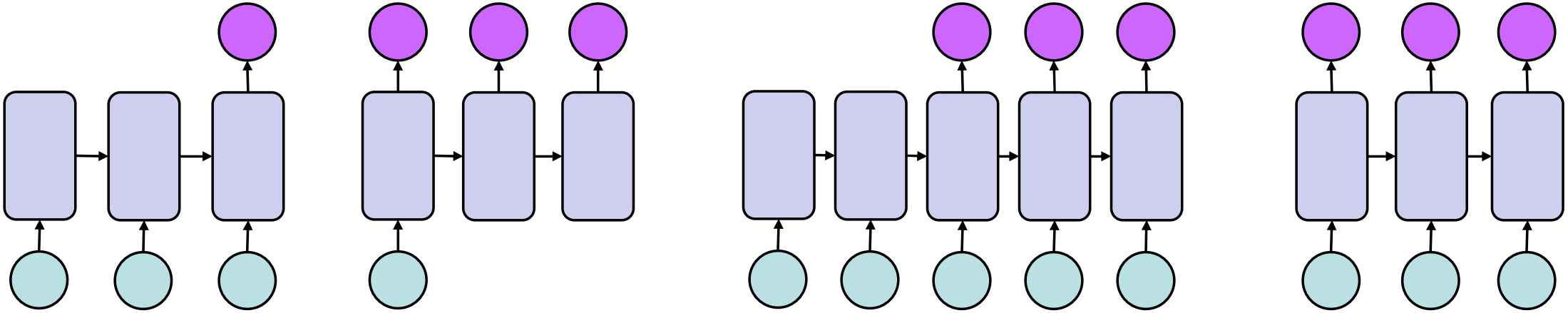
Recurrent Neural Networks (RNN)

Recurrent cells and language models, training, and the vanishing gradient problem

Recurrent Versus Feedforward Neural Networks

- **Feedforward networks**
  - Input to output mapping using vectors of fixed size
  - Fixed number of computational steps (e.g., hidden layer(s))

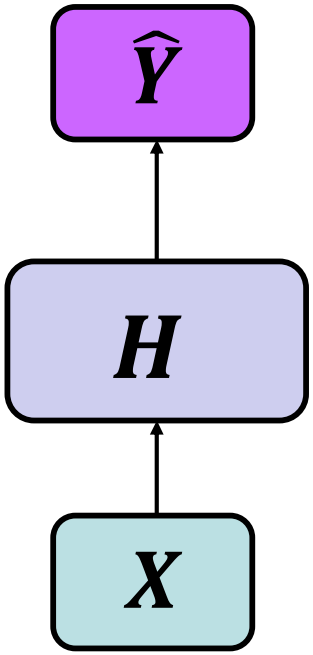
- **Recurrent networks provide a flexible sequence modeling framework**
  - Operate over sequences of vectors
  - Sequences in the input layer, in the output layer, or in both layers



Many to one | One to many | Many to many without sync. | Synced many to many

Feedforward Neural Network Revisited

Hidden layer has no memory and considers each case individually



Classification case

$$softmax(\hat{Y})$$

$$\hat{Y} = HW_{hy} + b_y$$

$$H = g(XW_{xh} + b_h)$$

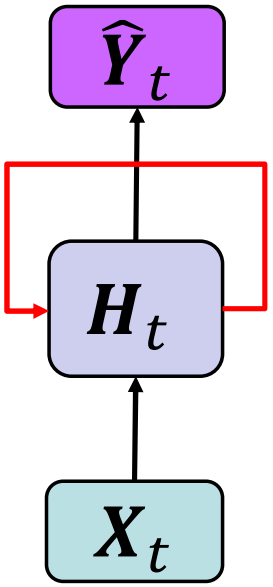
Notation

$X \in \mathbb{R}^{n \times m}$
$W_{xh} \in \mathbb{R}^{m \times h}$
$b_h \in \mathbb{R}^{1 \times h}$
$H \in \mathbb{R}^{n \times h}$
$W_{hy} \in \mathbb{R}^{h \times c}$
$b_y \in \mathbb{R}^{1 \times c}$

n	No. of cases; e.g., minibatch size
m	No. of input variables
c	No. of classes / nuerons in output layer
h	No. of hidden units

A Recurrent Neural Network

The recurrent cell accumulates information over multiple time steps



$$softmax(\hat{Y})$$

$$\hat{Y}_t = H_t W_{hy} + b_y$$

$$H_t = g(X_t W_{xh} + H_{t-1} W_{hh} + b_h)$$

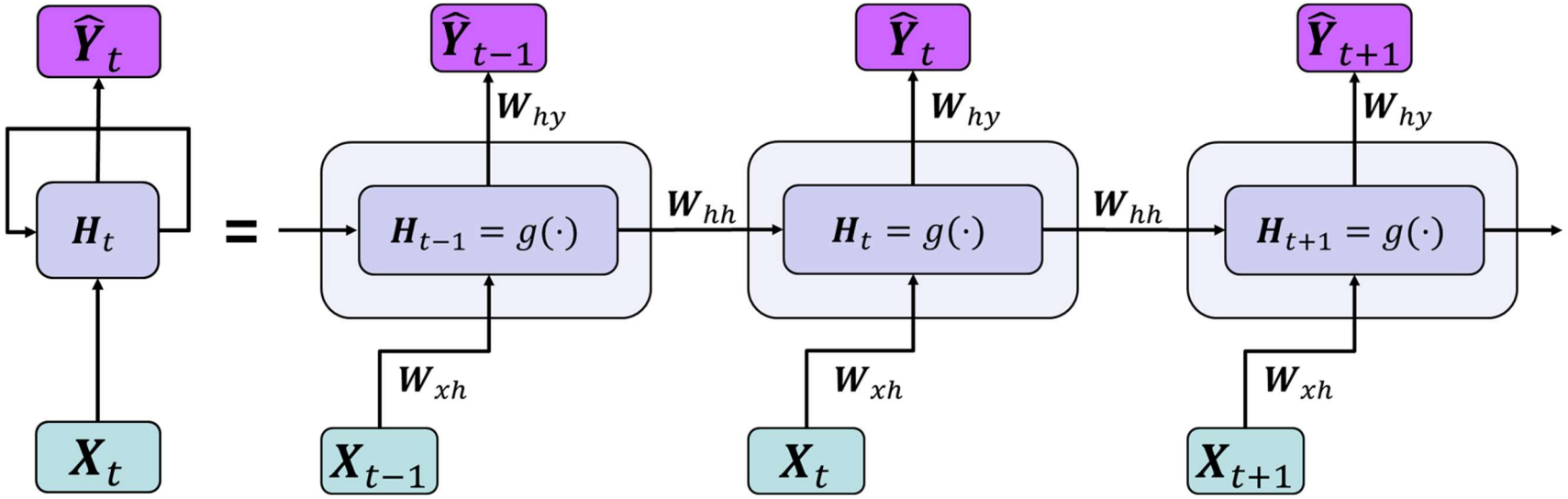
Notation:

$X_t \in \mathbb{R}^{n \times m}$
$W_{xh} \in \mathbb{R}^{m \times h}$
$b_h \in \mathbb{R}^{1 \times h}$
$H_t \in \mathbb{R}^{n \times h}$
$W_{hh} \in \mathbb{R}^{h \times h}$
$W_{hy} \in \mathbb{R}^{h \times c}$
$b_y \in \mathbb{R}^{1 \times c}$

RNN Represent a Latent Variable Autoregressive Model

$$x_t \sim p(x_t | x_{t-1}, h_{t-1})$$

$$\begin{cases} \hat{\boldsymbol{Y}}_t = \boldsymbol{H}_t \boldsymbol{W}_{hy} + \boldsymbol{b}_y \\ \boldsymbol{H}_t = g(\boldsymbol{X}_t \boldsymbol{W}_{xh} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hh} + \boldsymbol{b}_h) \end{cases}$$



An RNN Language Model

output distribution

$softmax(\hat{\boldsymbol{y}}_t)$

$\hat{\boldsymbol{y}}_t = \boldsymbol{W}_{hy}\boldsymbol{H}_t + \boldsymbol{b}_y$, with $\hat{\boldsymbol{y}}_t \in \mathbb{R}^{|V|}$

hidden state

$\boldsymbol{H}_t = g(\boldsymbol{e}_1\boldsymbol{W}_{eh} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{hh} + \boldsymbol{b}_h)$

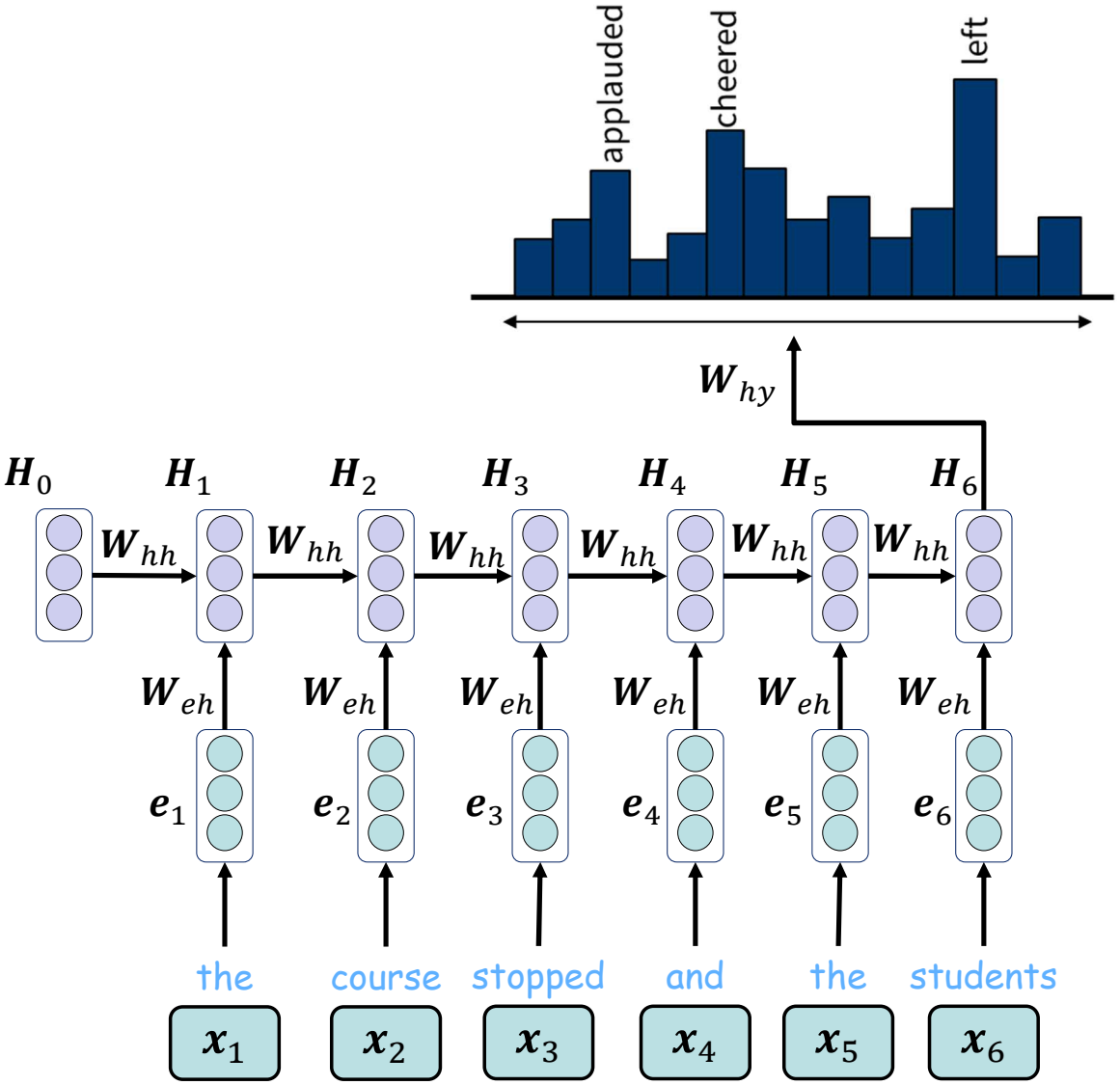
$\boldsymbol{H}_0 =$ is the initial hidden state

word embedding

$\boldsymbol{e}_t = \boldsymbol{x}_t^\top \boldsymbol{E} \in \mathbb{R}^d$

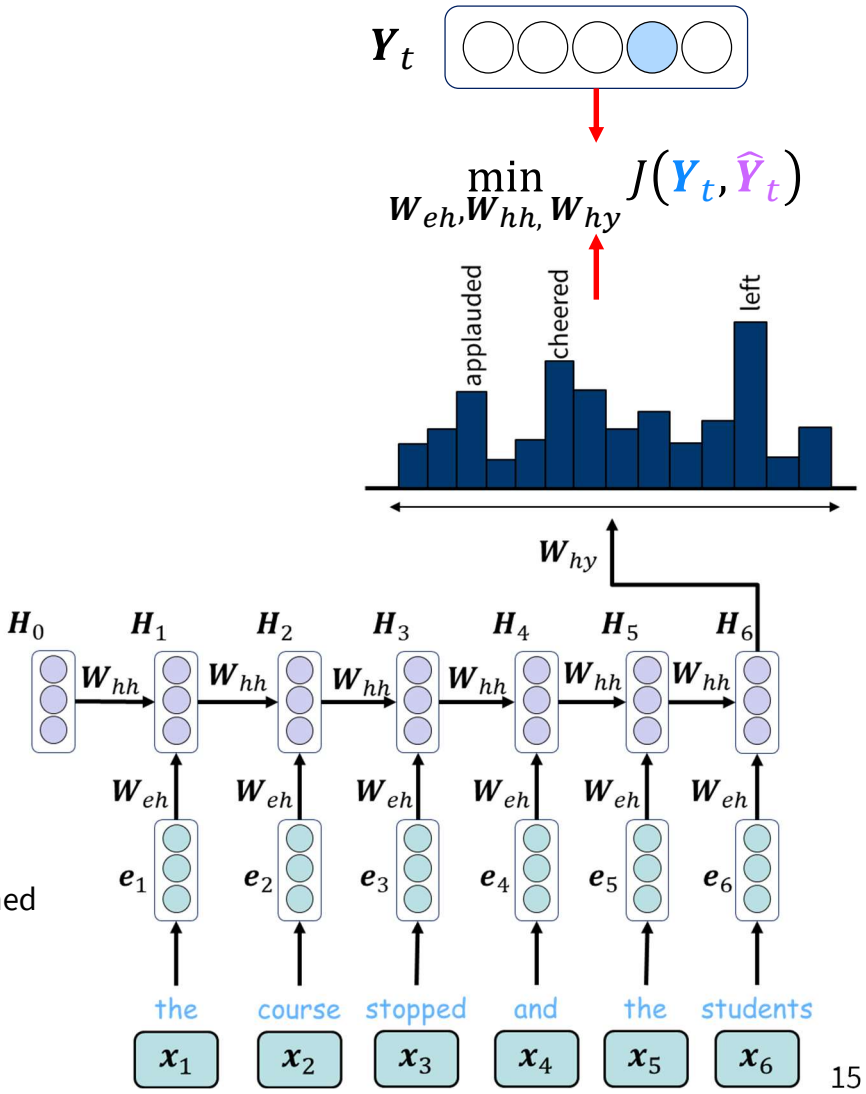
word as one-hot vector

$\boldsymbol{x}_t \in \{0,1\}^{|V|}$



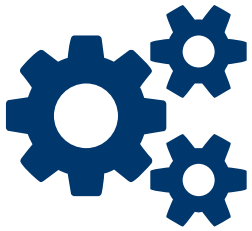
Training an RNN Language Model

- Gather text corpus
- Parse the text and feed windows into RNN
- Compute output distribution for every step t
- Assess loss using cross-entropy loss function
- Update weights using back-propagation (through time; see later)



Corpus

Sampling



started as early as eight in
in the morning <comma> talking about
about RNNs <dot> After a daunting
daunting two hours it eventually turned
eventually turned ten and the course
the course stopped and the students
students closed their books and left

Generating Text with RNN Language Model

RNN-LM trained on Obama speaches

The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done. The promise of the men and women who were still going to take out the fact that the American people have fought to make sure that they have to be able to protect our part. It was a chance to stand together to completely look for the commitment to borrow from the American people. And the fact is the men and women in uniform and the millions of our country with the law system that we should be a strong stretcks of the forces that we can afford to increase our spirit of the American people and the leadership of our country who are on the Internet of American lives.

Thank you very much. God bless you, and God bless the United States of America.

https://medium.com/@samim/obama-rnn-machine-generated-political-speeches-c8abd18a2ea0

Generating Text with RNN Language Model

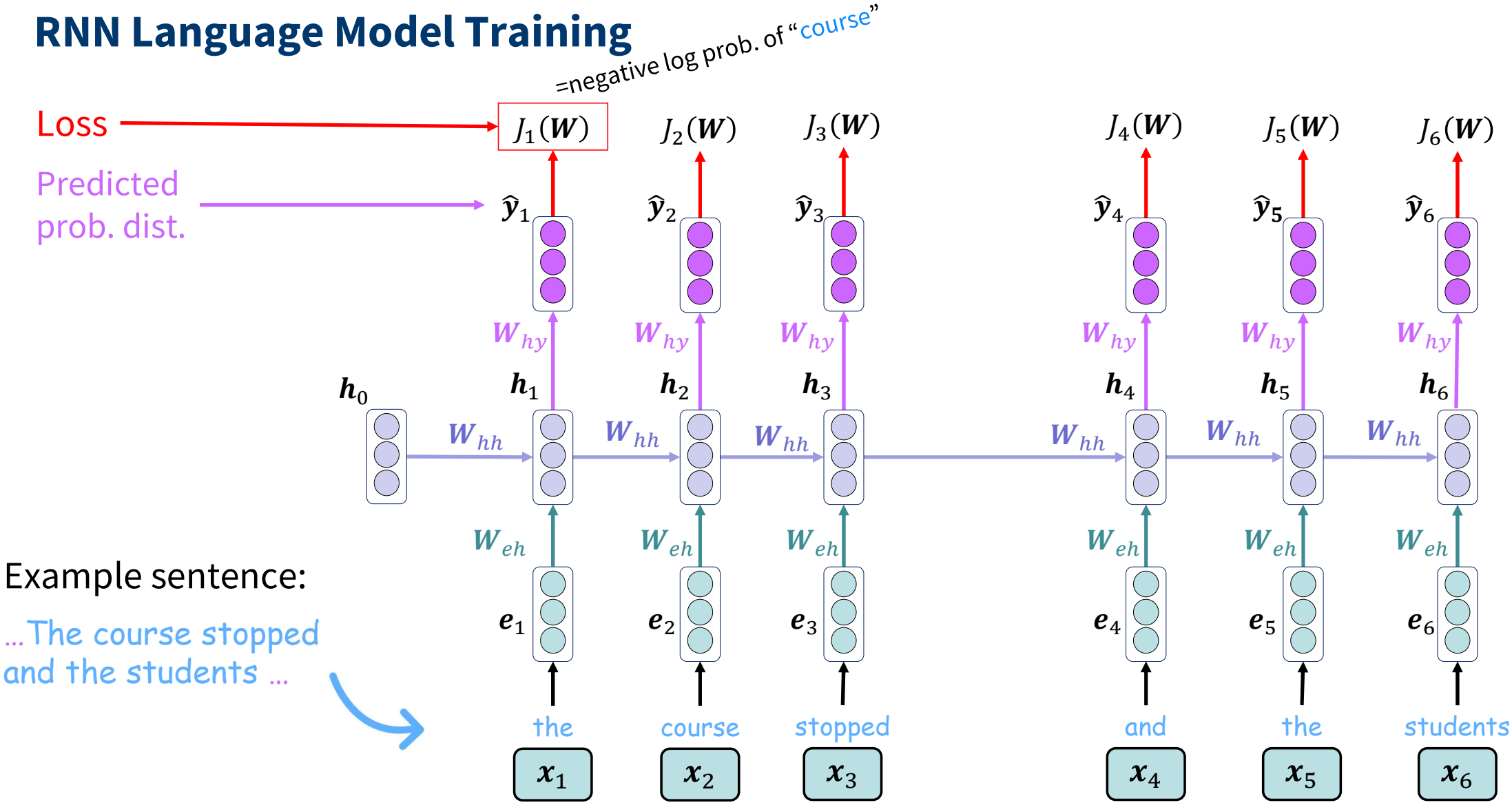
RNN-LM trained on Harry Potter I - IV

"The Malfoys!" said Hermione.

Harry was watching him. He looked like Madame Maxime. When she strode up the wrong staircase to visit himself.
"I'm afraid I've definitely been suspended from power, no chance—indeed?" said Snape. He put his head back behind them and read groups as they crossed a corner and fluttered down onto their ink lamp, and picked up his spoon. The doorbell rang. It was a lot cleaner down in London.
Hermione yelled. The party must be thrown by Krum, of course.
Harry collected fingers once more, with Malfoy. "Why, didn't she never tell me. …" She vanished. And then, Ron, Harry noticed, was nearly right.
"Now, be off," said Sirius, "I can't trace a new voice."

https://medium.com/deep-writing/harry-potter-written-by-artificial-intelligence-8a9431803da6

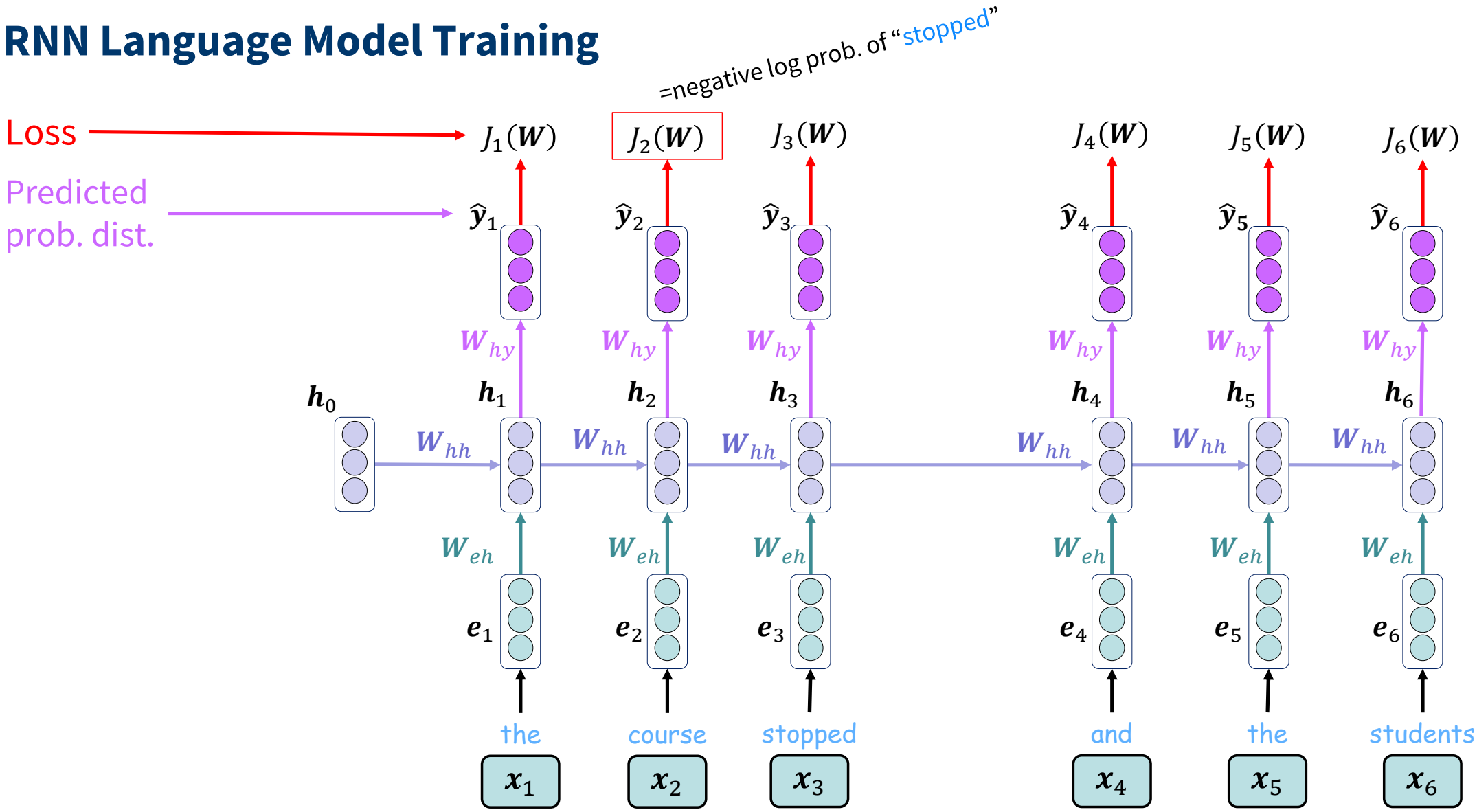
RNN Language Model Training



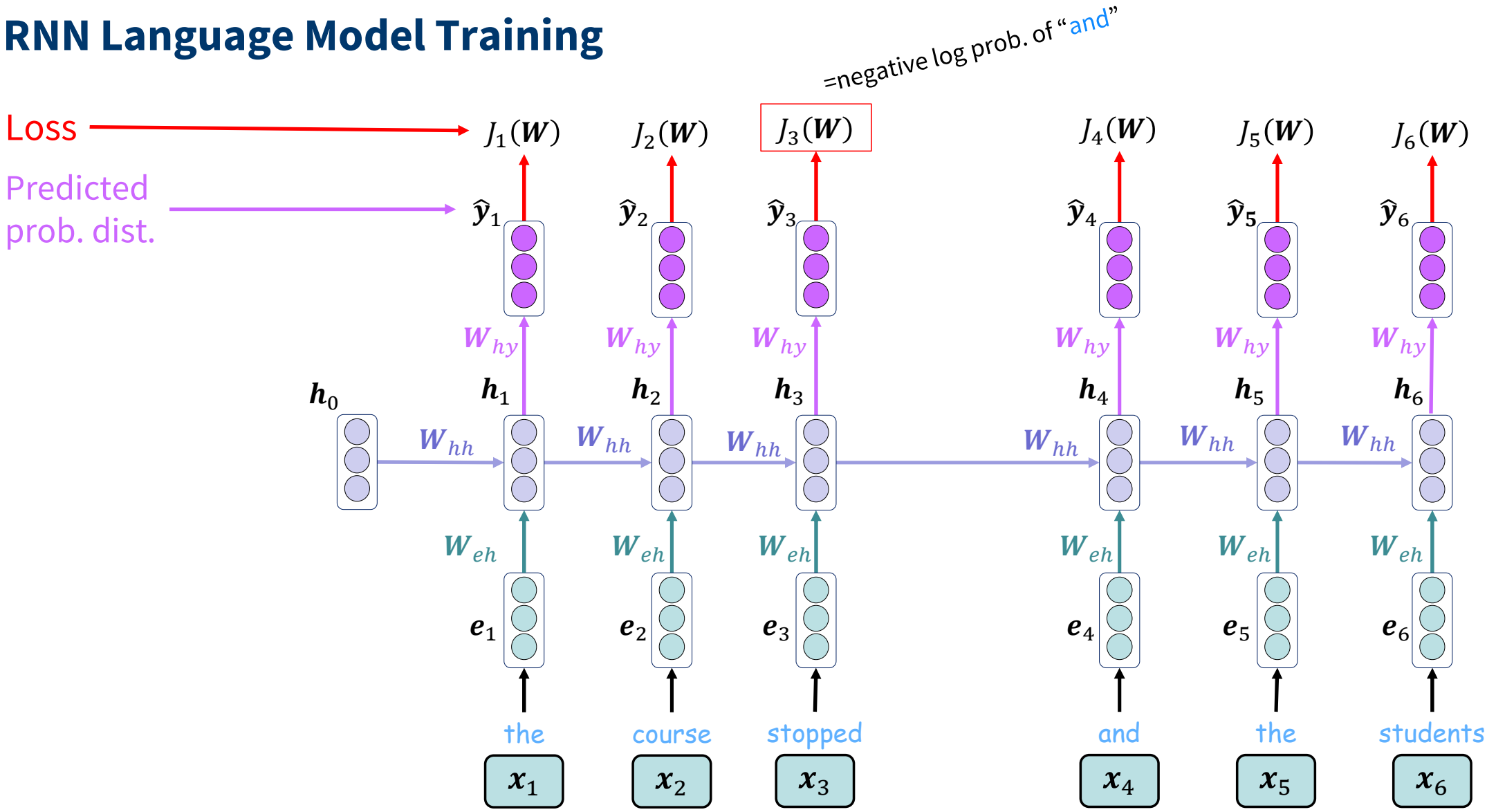
Example sentence:

...The course stopped and the students ...

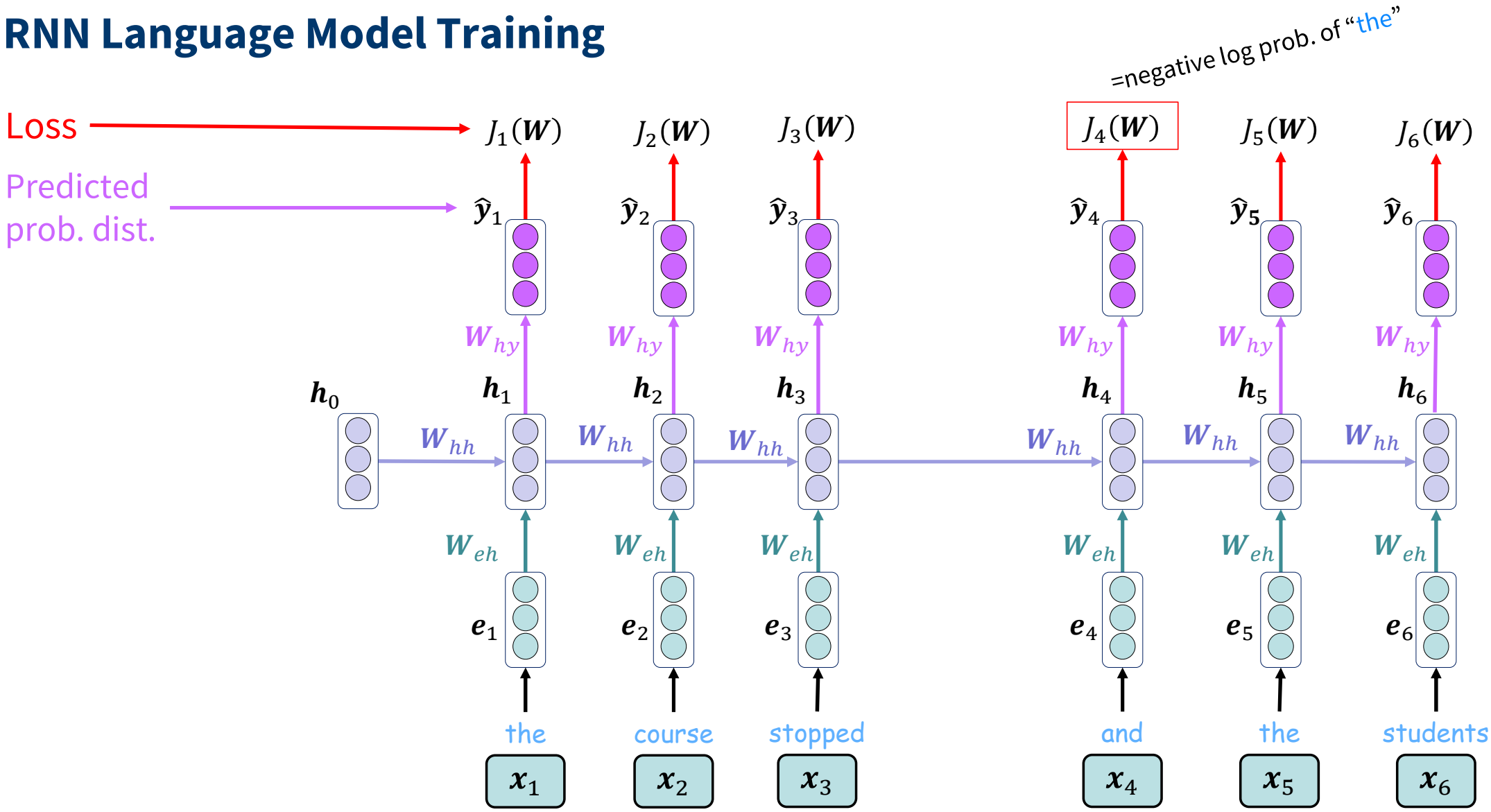
RNN Language Model Training



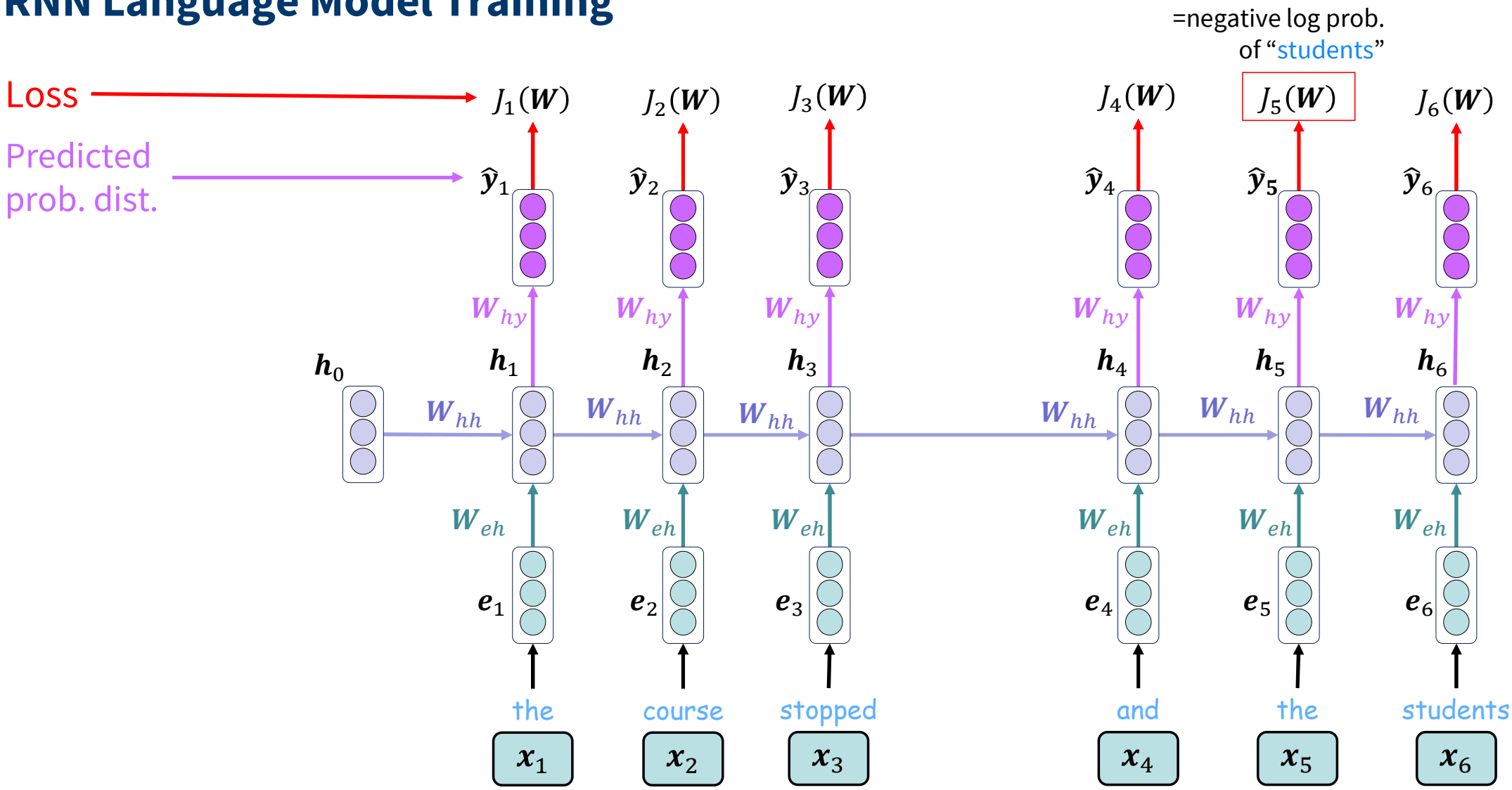
RNN Language Model Training



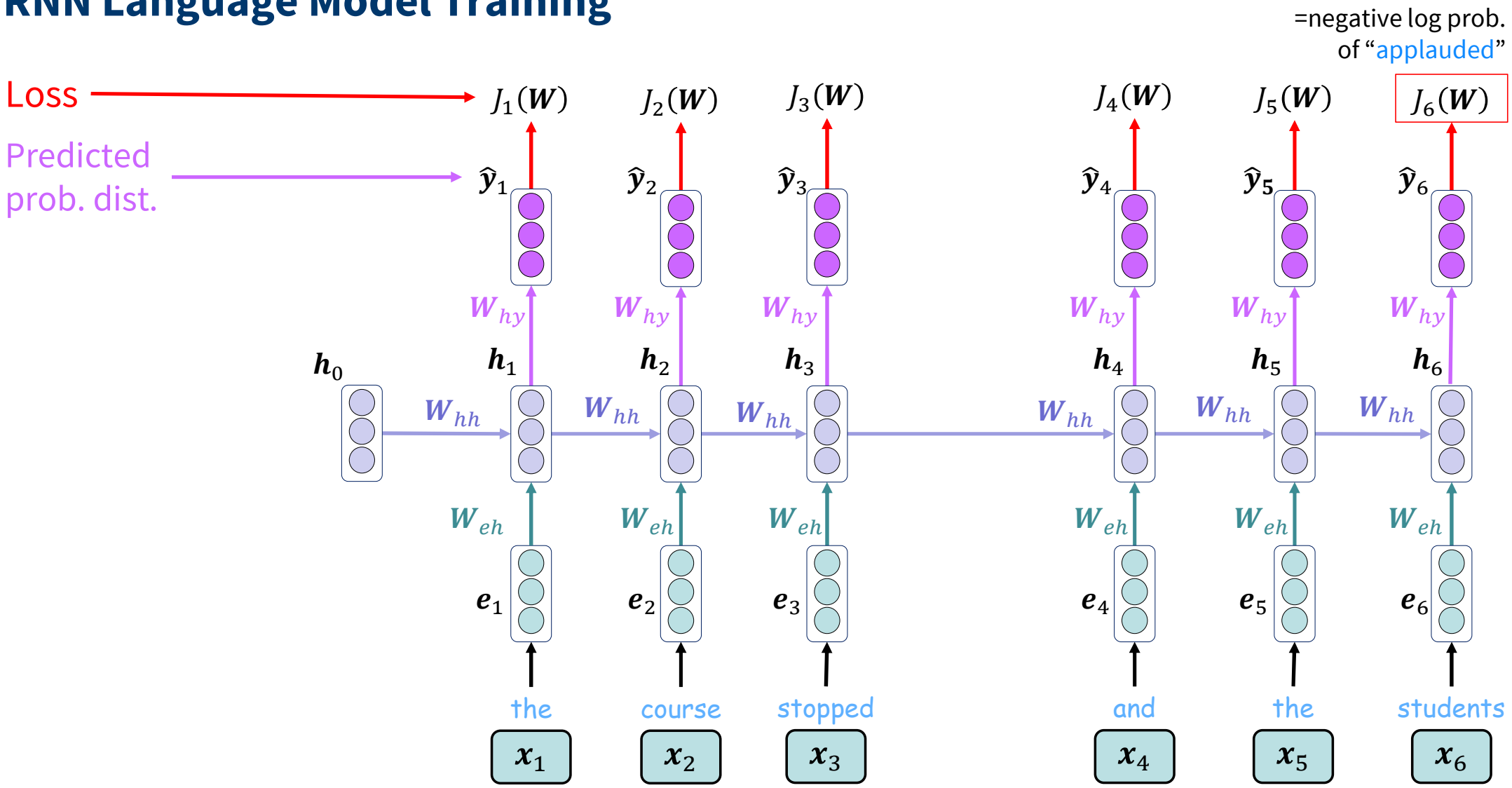
RNN Language Model Training



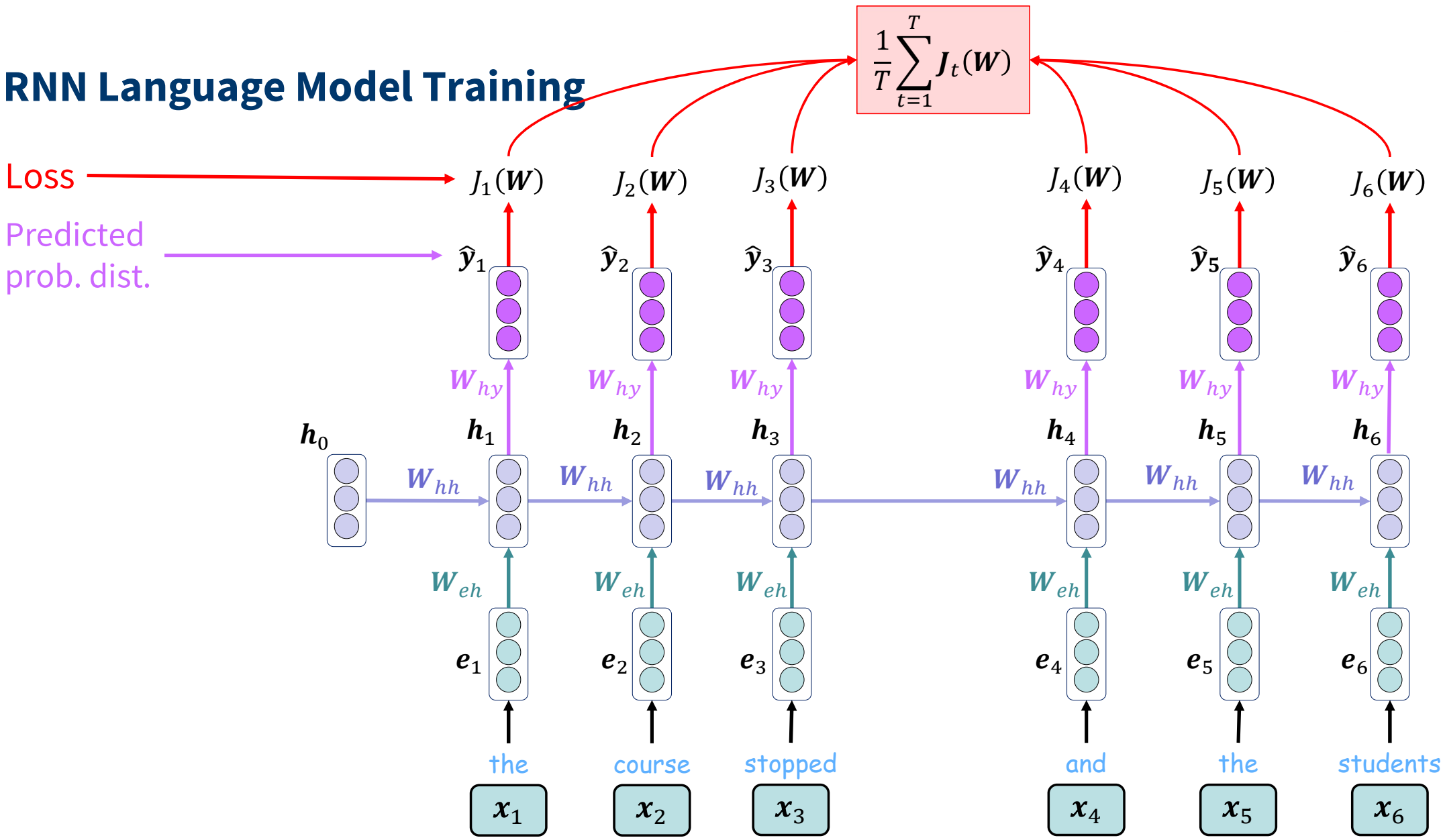
RNN Language Model Training



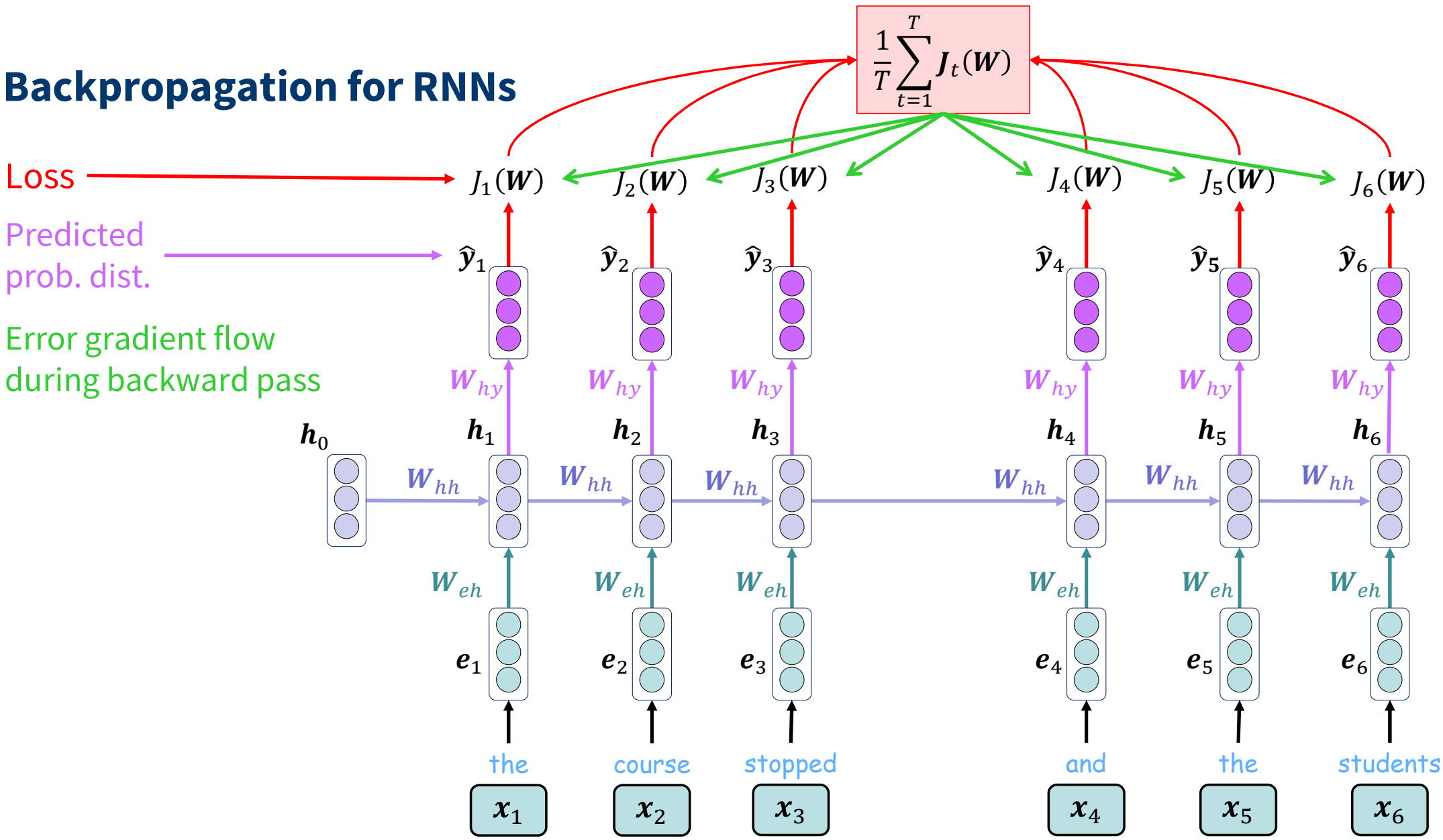
RNN Language Model Training



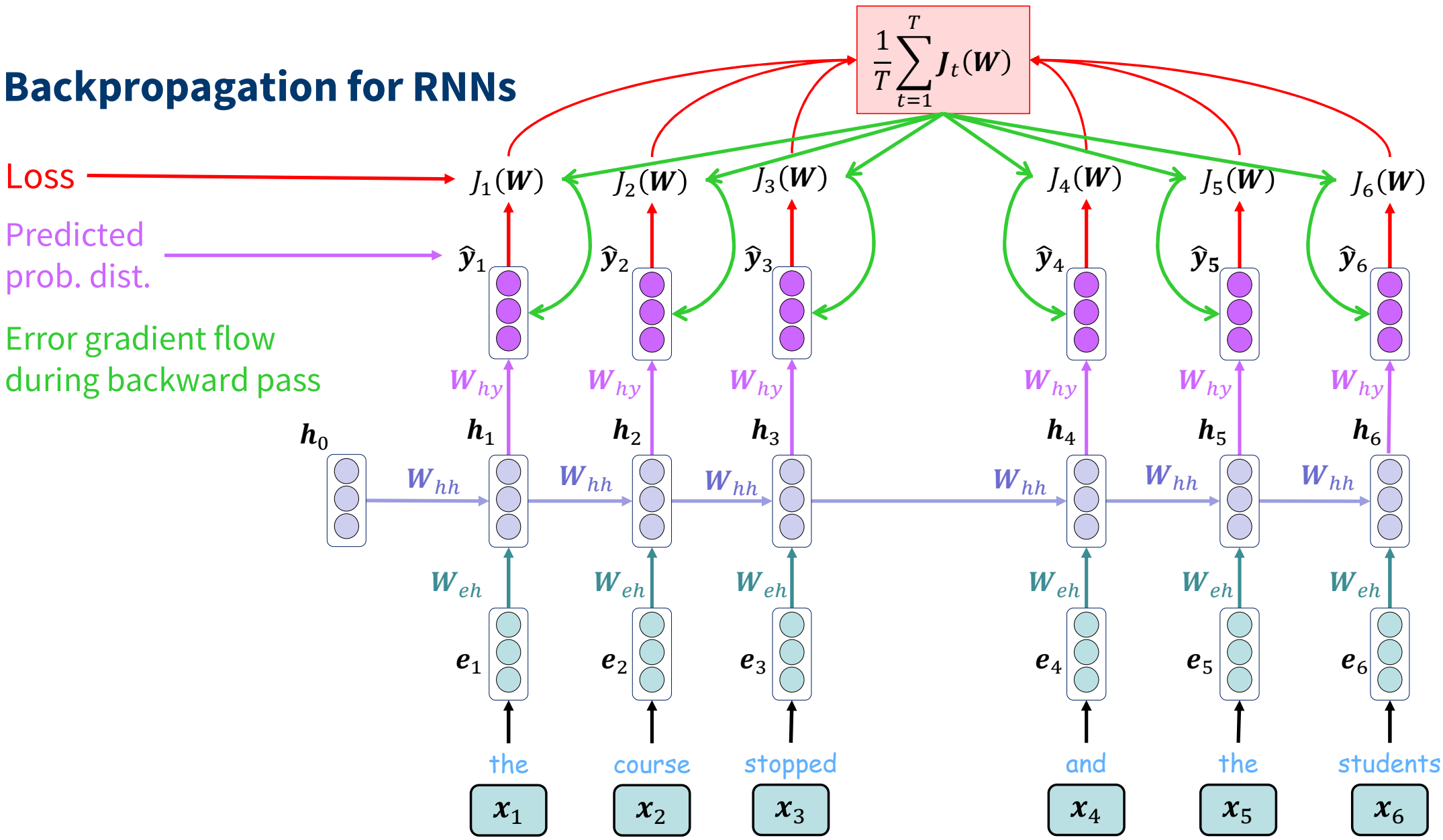
RNN Language Model Training



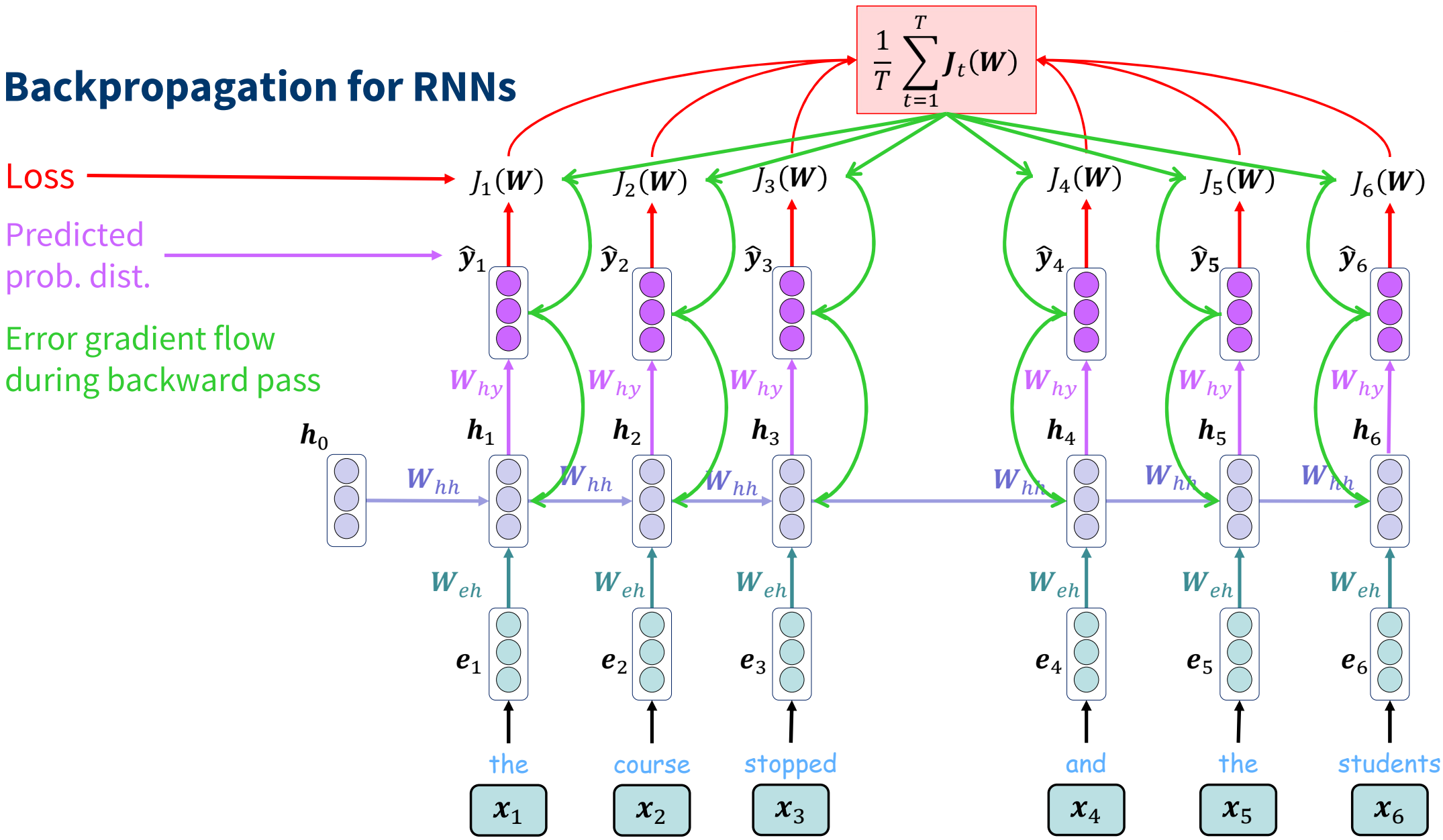
Backpropagation for RNNs



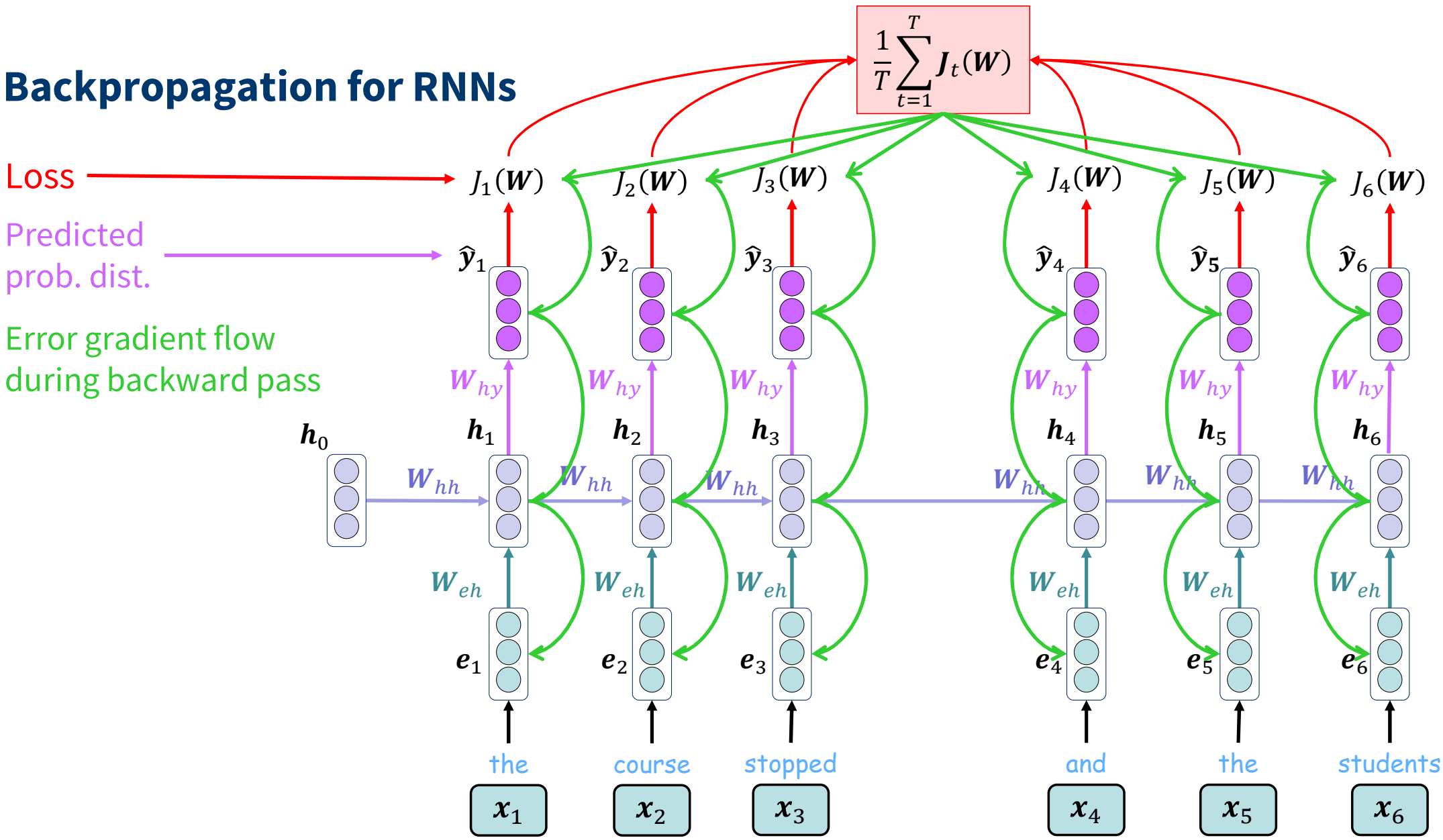
Backpropagation for RNNs



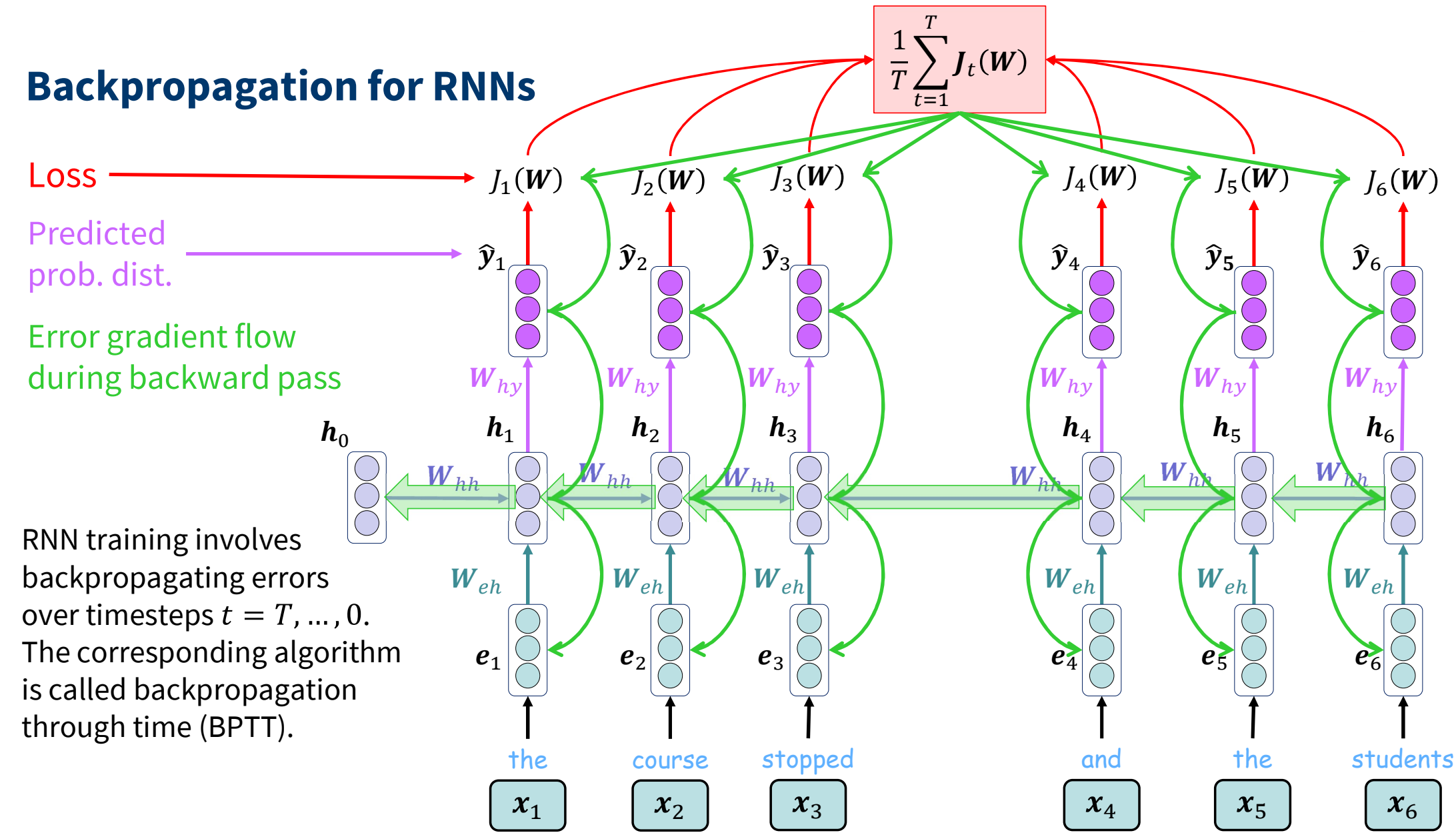
Backpropagation for RNNs



Backpropagation for RNNs



Backpropagation for RNNs



RNN training involves backpropagating errors over timesteps $t = T, \ldots, 0$. The corresponding algorithm is called backpropagation through time (BPTT).

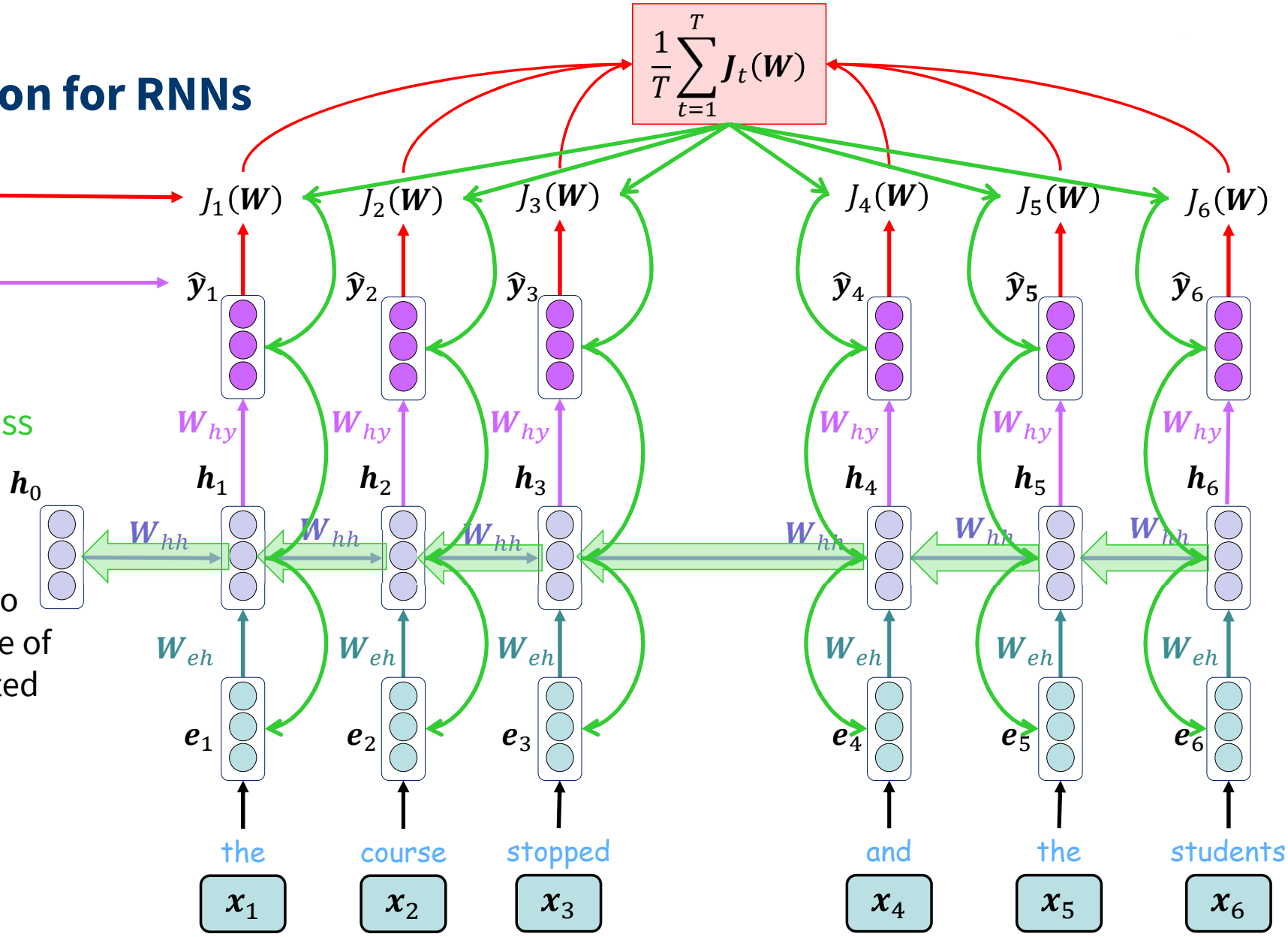
Backpropagation for RNNs

Loss

Predicted prob. dist.

Error gradient flow during backward pass

To that end, we need to calculate the derivative of $J_t(\boldsymbol{W})$ w.r.t. the repeated weight matrix $\boldsymbol{W}_{hh}$.



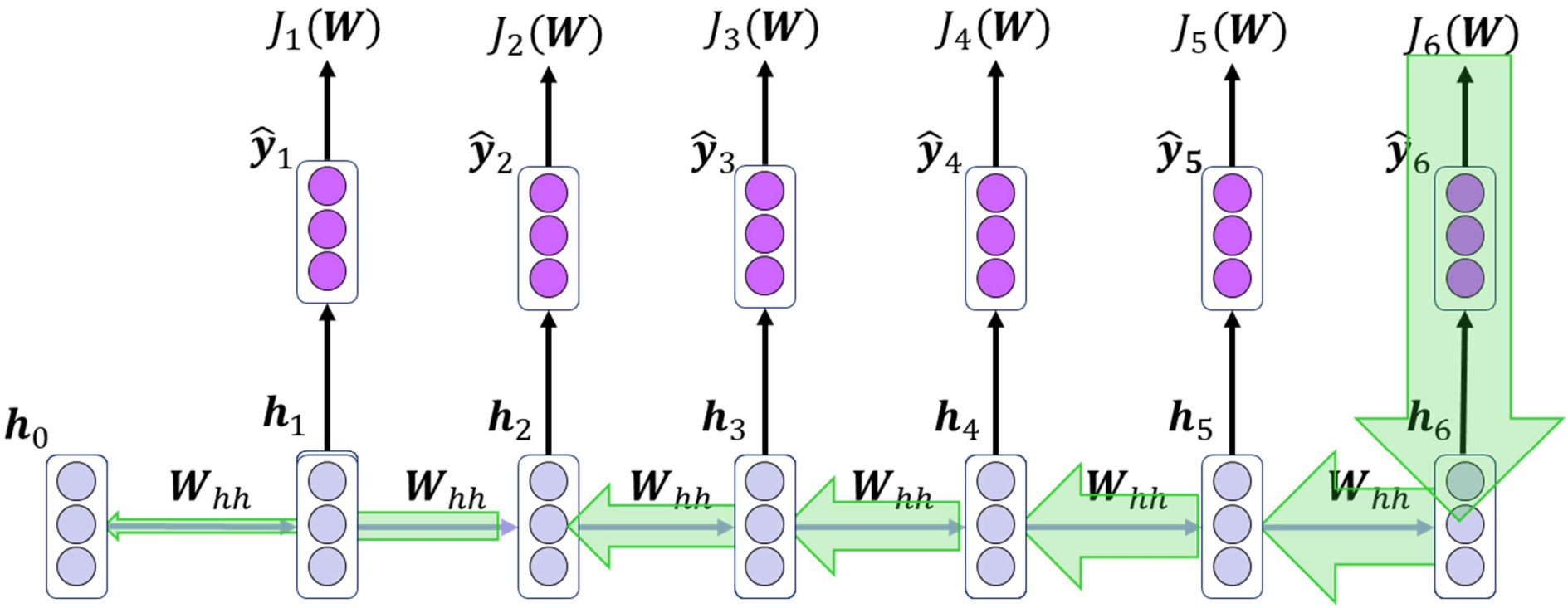
Backpropagation Through Time

Summary

- **An application of backpropagation to latent variable sequence models**
- **Involves long chains of derivatives**
- **In practice, BPTT is typically truncated for computational reasons**
  - Do not perform calculations all the way back to $\boldsymbol{h}_0$
  - Terminate the chain of computations at some – arbitrarily selected – threshold
  - See Tallec and Ollivier (2017) for details
- **Especially long sequences lead to large powers of $\boldsymbol{W}_{\boldsymbol{hh}}$**
  - Weights smaller than 1 led to the problem of gradient vanishing
  - Weights larger than 1 led to the problem of gradient explosion
  - See Pascanu et al. (2013) for a proof

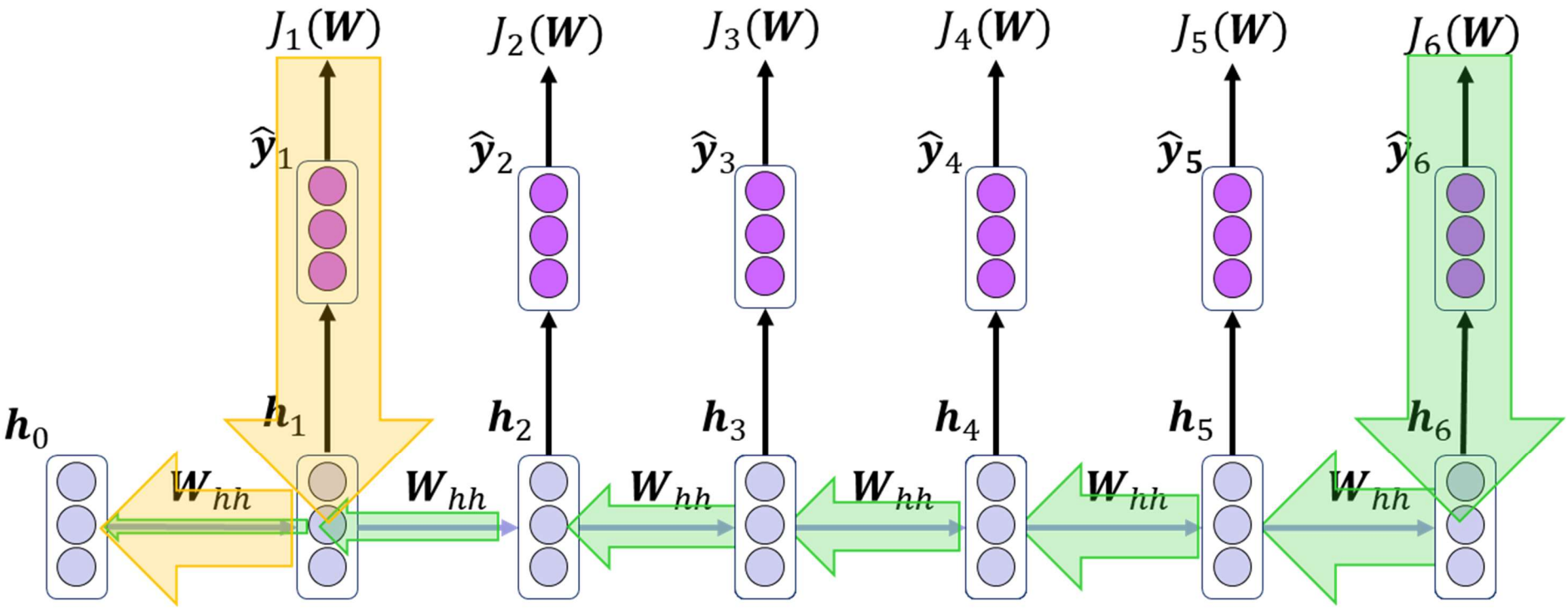
$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{W}_{hh}} = \sum_{j=1}^{t} \left(\boldsymbol{W}_{hh}^T\right)^{t-j} \boldsymbol{h}_j$$

The Problem of Vanishing Gradients



The gradient signal from faraway is lost because of repeated multiplications of the weight matrix when applying the chain rule.

The Problem of Vanishing Gradients

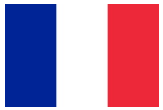


The gradient signal from faraway is lost and outweighed by close by gradient signal. Thus, weight updates emphasize near effects while the network faces difficulties in learning long-term dependencies.

The Problem of Vanishing Gradients

- **Can interpret gradient as a measure of how much the past affects the future**
- **If the gradient signal vanishes over longer distances t to $t + j$**
- **There is no way to determine whether**
  - There is no dependency between step t and $t + j$ in the data
  - We have the wrong parameters to capture the dependency between t and $t + j$
- **Language modeling example**
  - Model needs to learn the dependency between “France” and “speak fluent”
  - Gradient vanishing prohibits learning this dependency
  - Rendering the model unable to predict dependencies of such length

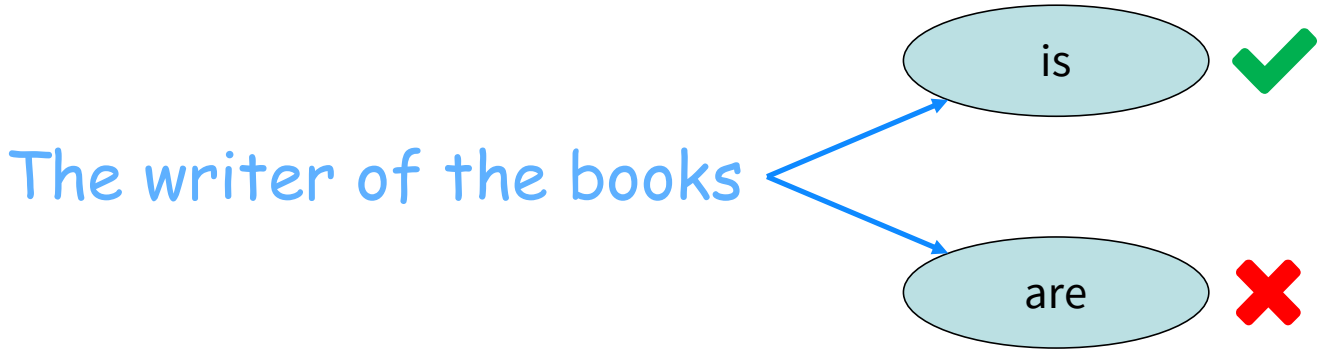
I grew up in France where my father works as a teacher in primary school. My mom is a criminal lawyer. Just recently, I moved to Berlin for my undergraduate studies. German proves difficult but I speak fluent ???.



Another Language Modeling Example

Linzen et al. (2016)

- **Syntactic recency suggests ”is“**
- **Sequential recency suggests “are”**
- **Due to the vanishing gradient problem, RNNs are better at learning from sequential recency than syntactic recency**



Gated RNNs for the rescue

Design idea, gated cells, GRM & LSTM, and extensions

Gated RNNs

Address problem of gradient vanishing & learning long-term dependencies

- **RNNs update the hidden state in every step**
  - $\boldsymbol{H}_t = g(\boldsymbol{X}_t \boldsymbol{W}_{xh} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hh} + \boldsymbol{b}_h)$
  - New information enters network memory even if not relevant
  - Complicates/prohibits learning long-term dependencies
- **Solution**
  - Update memory selectively
  - Learn – as part of the training – what information is relevant and deserves being kept
- **Approaches**
  - Long short-term memory cell proposed by Hochreiter, Schmidhuber (1997)
  - Gated recurrent unit (GRU) proposed by Chung et al. (2014)
  - Think of them as an enhanced version of an RNN cell
  - Building a network out of RNN cells works essentially as before

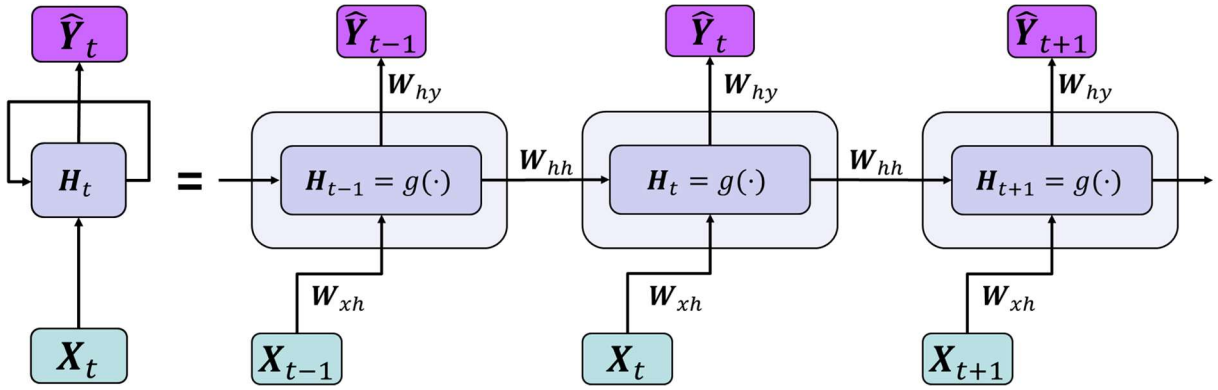
Gated Recurrent Units (GRU)¶

Chung et al. (2014)

- **Gating the hidden state**
  - Dedicated mechanism to decide when to update or reset the hidden state
  - Learn this mechanism as part of training
- **What actually is meant by "gating"?**

$$\widehat{\boldsymbol{Y}}_t = \boldsymbol{H}_t \boldsymbol{W}_{hy} + \boldsymbol{b}_y$$

$$\boldsymbol{H}_t = g(\boldsymbol{X}_t \boldsymbol{W}_{xh} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hh} + \boldsymbol{b}_h)$$



$\boldsymbol{X}_t \in \mathbb{R}^{n \times m}$
$\boldsymbol{W}_{xh} \in \mathbb{R}^{m \times h}$
$\boldsymbol{b}_h \in \mathbb{R}^{1 \times h}$
$\boldsymbol{H}_t \in \mathbb{R}^{n \times h}$
$\boldsymbol{W}_{hh} \in \mathbb{R}^{h \times h}$
$\boldsymbol{W}_{hy} \in \mathbb{R}^{h \times c}$
$\boldsymbol{b}_y \in \mathbb{R}^{1 \times c}$

Gated Recurrent Units (GRU)¶

A dedicated matrix or gate moderates these updates

- **Gating the hidden state**
  - Dedicated mechanism to decide when to update or reset the hidden state
  - Learn this mechanism as part of training
- **What actually is meant by "gating"?**

$$\widehat{\boldsymbol{Y}}_t = \boldsymbol{H}_t \boldsymbol{W}_{hy} + \boldsymbol{b}_y$$

$$\boldsymbol{H}_t = g(\boldsymbol{X}_t \boldsymbol{W}_{xh} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hh} + \boldsymbol{b}_h)$$

$$\boldsymbol{H}_t = \begin{bmatrix} g(z_{11}) & \cdots & g(z_{1h}) \\ \vdots & \ddots & \vdots \\ g(z_{1n}) & \cdots & g(z_{nh}) \end{bmatrix} \odot \begin{bmatrix} \sigma(u_{11}) & \cdots & \sigma(u_{1h}) \\ \vdots & \ddots & \vdots \\ \sigma(u_{1n}) & \cdots & \sigma(u_{nh}) \end{bmatrix} = \boldsymbol{U}_t$$

$\boldsymbol{X}_t \in \mathbb{R}^{n \times m}$
$\boldsymbol{W}_{xh} \in \mathbb{R}^{m \times h}$
$\boldsymbol{b}_h \in \mathbb{R}^{1 \times h}$
$\boldsymbol{H}_t \in \mathbb{R}^{n \times h}$
$\boldsymbol{W}_{hh} \in \mathbb{R}^{h \times h}$
$\boldsymbol{W}_{hy} \in \mathbb{R}^{h \times c}$
$\boldsymbol{b}_y \in \mathbb{R}^{1 \times c}$
$\boldsymbol{U}_t \in \mathbb{R}^{n \times h}$

Gated Recurrent Networks

Design ideas

- **Update the memory of the network selectively**
  - Gating mechanism to decide when to update or reset the memory
  - Design gates as basic neural networks and learn weights during RNN training
    - Matrices of same dimension as the hidden state/memory
    - All weights between zero (gate is closed) and one (gate is open) due to sigmoid activation
  - Perform element-wise multiplication
- **GRUs gate the hidden state (→ network memory) using two gates**
  - Update gate → governs which information is added to the hidden state
  - Reset gate → governs which information is removed from the hidden state
- **LSTMs add a dedicated cell memory and introduce three gates**
  - Dedicated memory cell to store long-term information
  - Input gate, output gate, forget gate

Formal Derivation of BPTT for LSTM and GRU

Key take away: avoid high powers of weight matrices in BPTT

LSTM¶

We first start by taking the neatly separated equations (1) to (6) and plug them all in, in order to get a more granular understanding of how our predictions are formed.

$$
\begin{aligned}
\hat{y}_t &= w^T \cdot h_t = w^T \cdot [o_t * g(c_t)] \\
&= w^T \left[\sigma\left(W_o + U_o \cdot h_{t-1}\right) * g(f_t * c_{t-1} + i_t * \tilde{c}_t)\right] \\
&= w^T \cdot \left[\begin{pmatrix} \sigma\left(w_1^o x_t + u_{11}^o h_1^{t-1} + u_{12}^o h_2^{t-1}\right) \\ \sigma\left(w_2^o x_t + u_{21}^o h_1^{t-1} + u_{22}^o h_2^{t-1}\right) \end{pmatrix} \right. \\
&\qquad * g \left. \begin{pmatrix} \sigma\left(w_1^f x_t + u_{11}^f h_1^{t-1} + u_{12}^f h_2^{t-1}\right) c_1^{t-1} + \sigma\left(w_1^i x_t + u_{11}^i h_1^{t-1} + u_{12}^i h_2^{t-1}\right) + g\left(w_1^c x_t + u_{11}^c h_1^{t-1} + u_{12}^c h_2^{t-1}\right) \\ \sigma\left(w_2^f x_t + u_{21}^f h_1^{t-1} + u_{22}^f h_2^{t-1}\right) c_2^{t-1} + \sigma\left(w_2^i x_t + u_{21}^i h_1^{t-1} + u_{22}^i h_2^{t-1}\right) + g\left(w_2^c x_t + u_{21}^c h_1^{t-1} + u_{22}^c h_2^{t-1}\right) \end{pmatrix} \right]
\end{aligned}
$$

which, if written in its single equation form, gives

$$
\begin{aligned}
\hat{y}_t = {} & w_1 \sigma\left(w_1^o x_t + u_{11}^o h_1^{t-1} + u_{12}^o h_2^{t-1}\right) g\left(\sigma\left(w_1^f x_t + u_{11}^f h_1^{t-1} + u_{12}^f h_2^{t-1}\right) c_1^{t-1} + \right. \\
& \left. \sigma\left(w_1^i x_t + u_{11}^i h_1^{t-1} + u_{12}^i h_2^{t-1}\right) g\left(w_1^c x_t + u_{11}^c h_1^{t-1} + u_{12}^c h_2^{t-1}\right)\right) \\
& + w_2 \sigma\left(w_2^o x_t + u_{21}^o h_1^{t-1} + u_{22}^o h_2^{t-1}\right) g\left(\sigma\left(w_2^f x_t + u_{21}^f h_1^{t-1} + u_{22}^f h_2^{t-1}\right) c_2^{t-1} + \right. \\
& \left. \sigma\left(w_2^i x_t + u_{21}^i h_1^{t-1} + u_{22}^i h_2^{t-1}\right) g\left(w_2^c x_t + u_{21}^c h_1^{t-1} + u_{22}^c h_2^{t-1}\right)\right)
\end{aligned} \tag{7}
$$

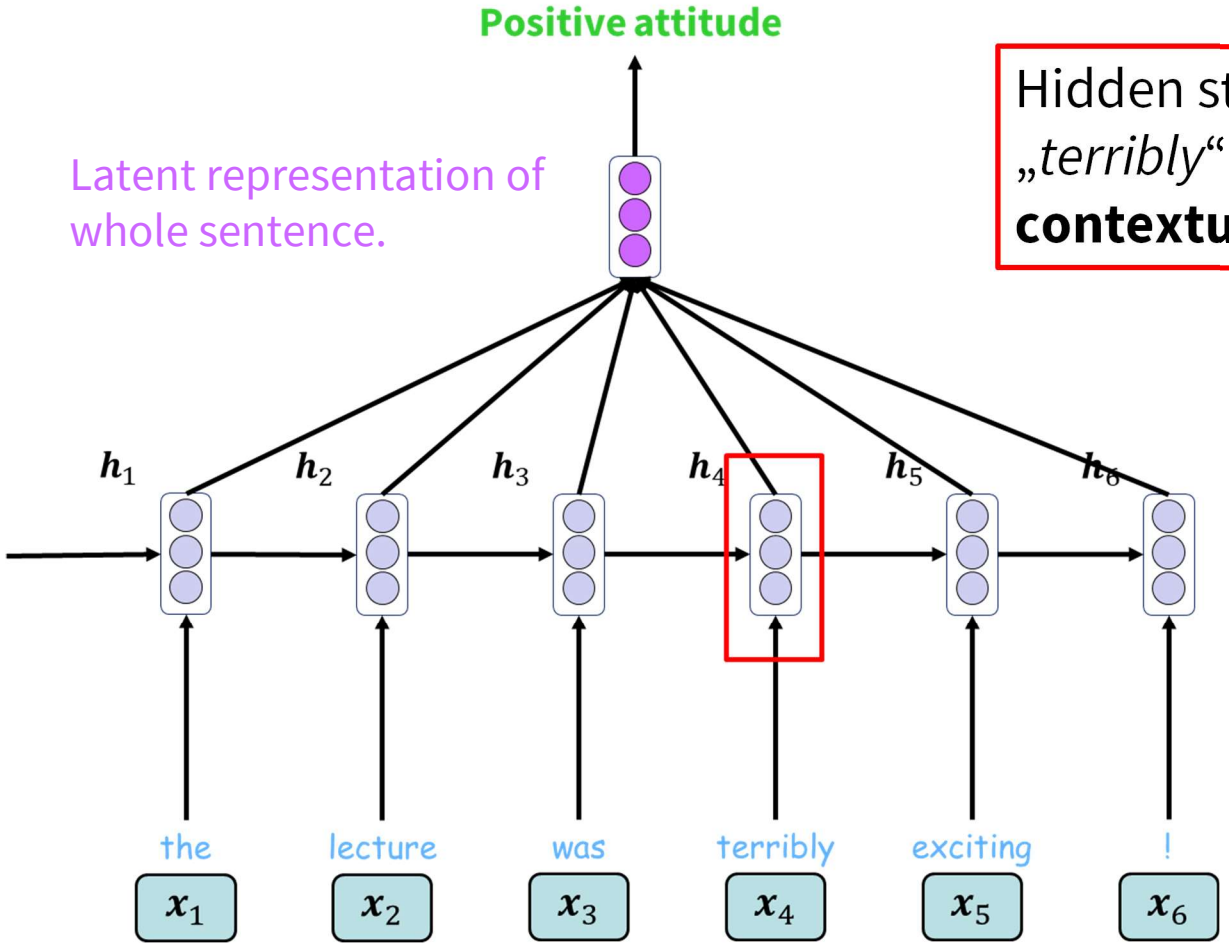
https://humboldt-wi.github.io/blog/research/information_systems_1819/02lstmgruandbeyond/

Further Information on GRUs / LSTMs

A small selection

- **Dive into Deep Learning Chapter 9 and 10: https://d2l.ai/chapter_recurrent-neural-networks/index.html**
- **Famous blog on LSTMs: https://colah.github.io/posts/2015-08-Understanding-LSTMs/**
- **Critical perspective on LSTMs, suggesting CNN (see later) as superior alternative: https://towardsdatascience.com/the-fall-of-rnn-lstm-2d1594c74ce0**
- **Animated RNN, LSTM and GRU https://towardsdatascience.com/animated-rnn-lstm-and-gru-ef124d06cf45**
- **Illustrated Guide to LSTM's and GRU's: A step by step explanation https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21**
- **Stock modeling example https://towardsdatascience.com/predicting-stock-price-with-lstm-13af86a74944**

Bidirectional Recurrent Neural Networks



Hidden state h_4 represents word „*terribly*“ in the context of the sequence: **contextual representation**

The contextual representation is based on the left context.

What about the right context?

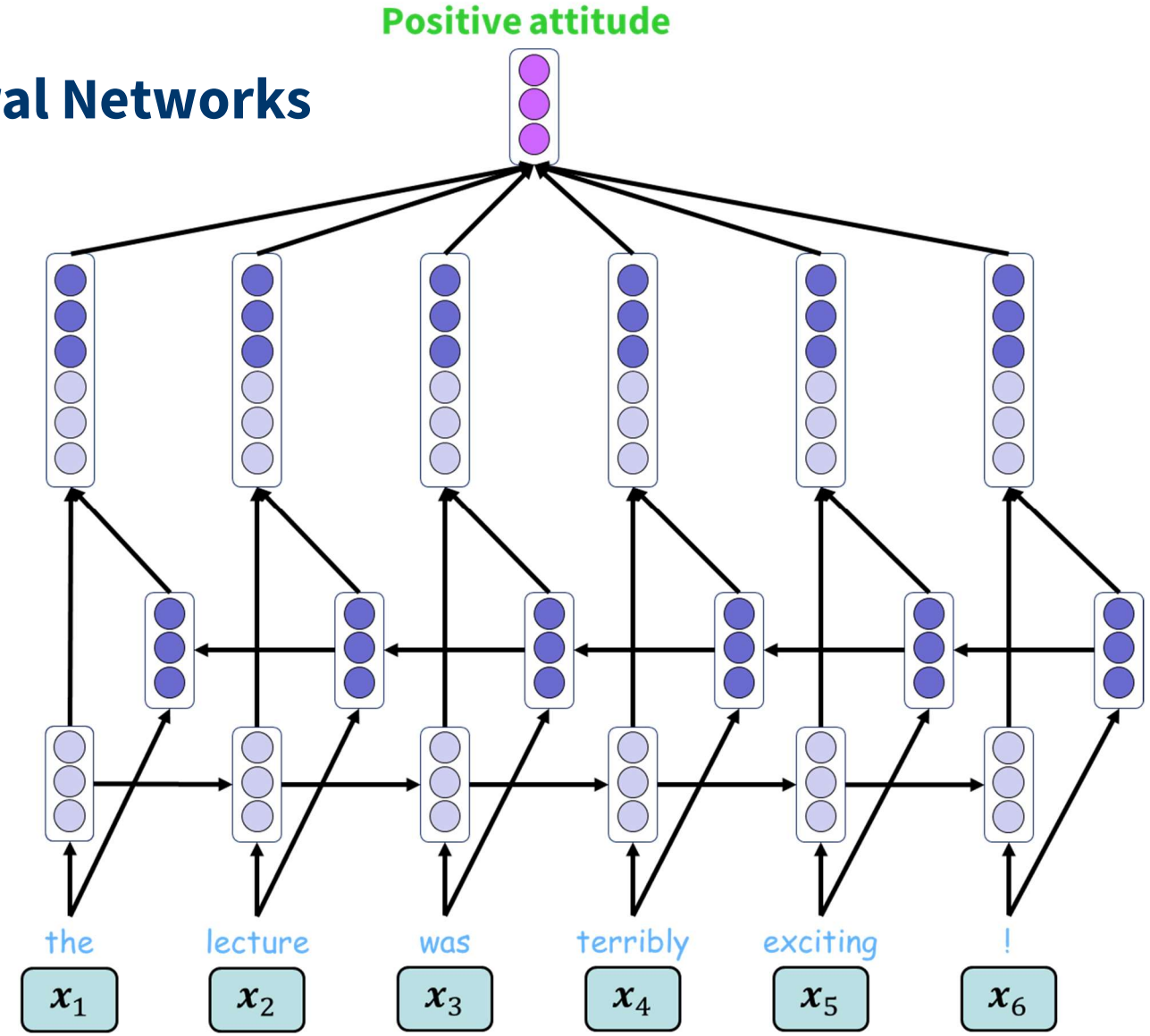
Here, the right context word „*exciting*“ shifts the polarity of the word „*terribly*“.

Bidirectional Recurrent Neural Networks

Concatenated hidden states

Backward RNN

Forward RNN



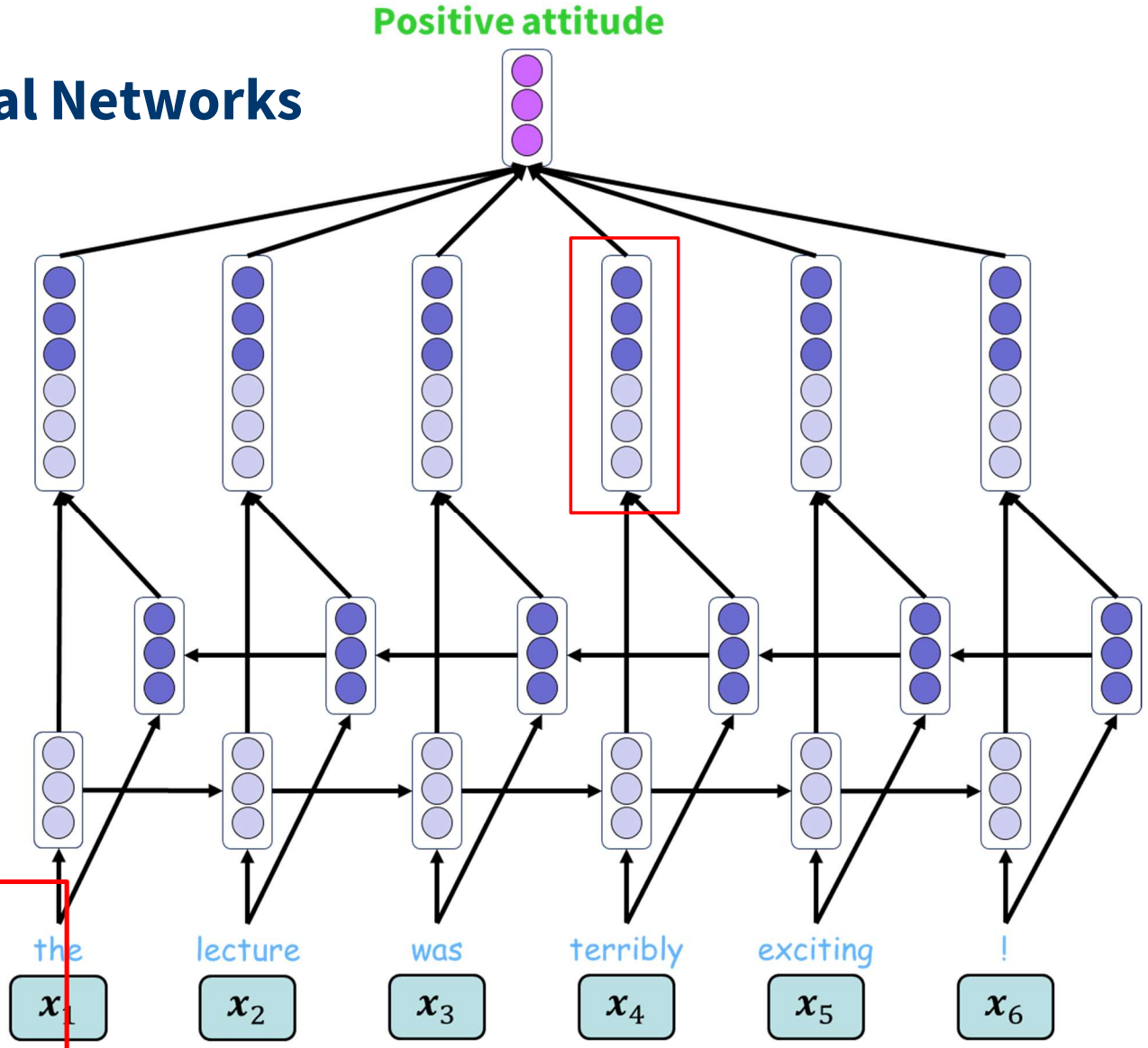
Bidirectional Recurrent Neural Networks

Concatenated hidden states

Backward RNN

Forward RNN

Contextual representation of word „*terribly*“ with both left and right context.



Bidirectional Recurrent Neural Networks

On one timestep t

Forward RNN

$$\overrightarrow{\boldsymbol{H}}_t = RNN_{FW}\left(\overrightarrow{\boldsymbol{H}}_{t-1}, \boldsymbol{X}_t\right)$$

Backward RNN

$$\overleftarrow{\boldsymbol{H}}_t = RNN_{BW}\left(\overleftarrow{\boldsymbol{H}}_{t+1}, \boldsymbol{X}_t\right)$$

Compute forward step of the RNN (vanilla, LSTM, GRU)

Two RNNs with separate weights

Concatenated hidden states

$$\boldsymbol{H}_t = \left[\overrightarrow{\boldsymbol{H}}_t, \overleftarrow{\boldsymbol{H}}_t\right]$$

Hidden state of a bidirectional RNN, which we forward to the next part of the network.

Bidirectional Recurrent Neural Networks

Considerations on applicability

- **Bidirectional RNNs require access to the entire input sequence**
  - Not applicable in language modeling where the goal is to predict the next word
  - Language model has access to left context only
  - Same in time series forecasting
- **Often more powerful than unidirectional RNNs if requirement is met**

Multi-Layer Recurrent Neural Networks

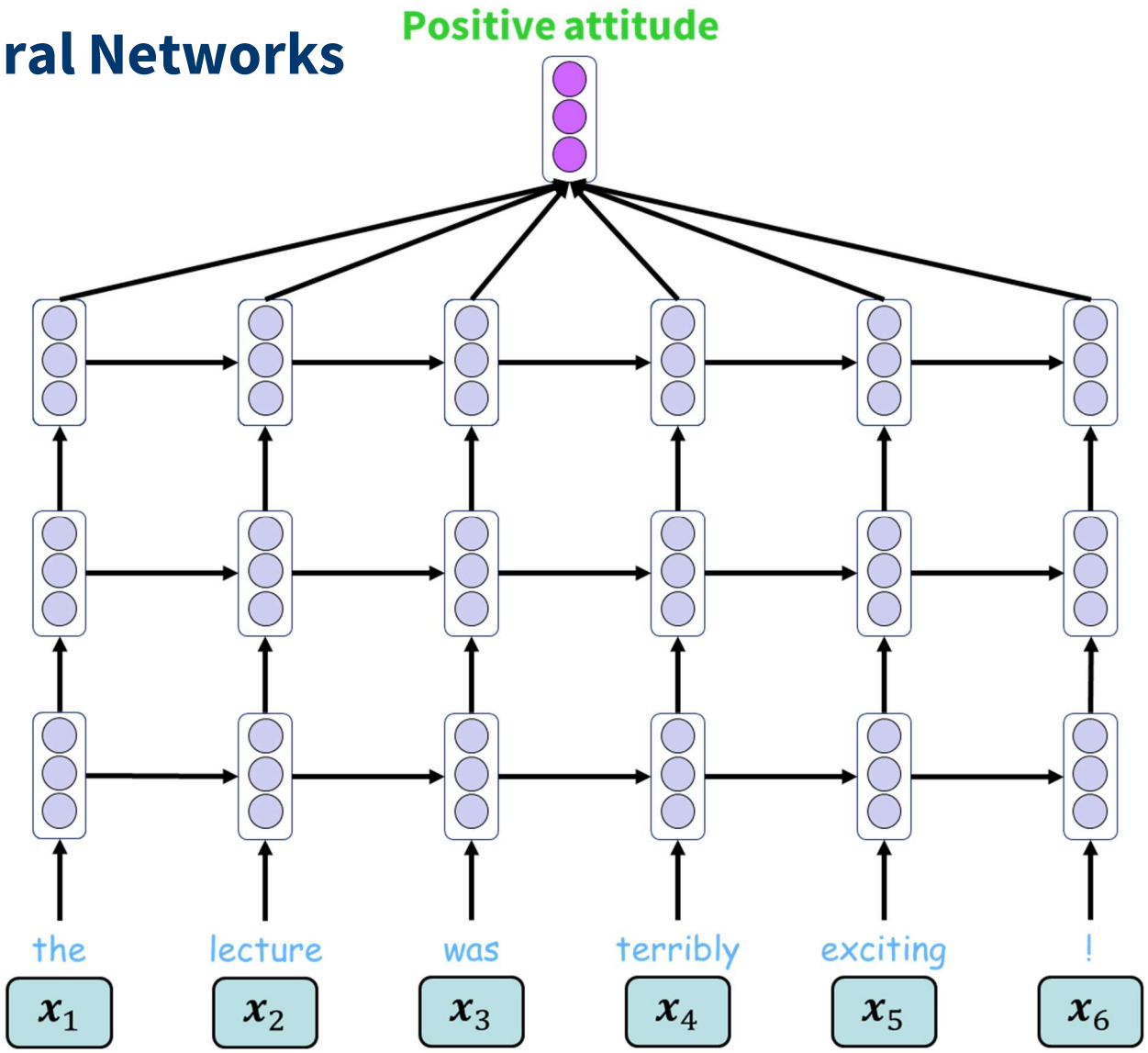
- **RNNs exhibit temporal depth by definition**
  - Enrolling computation over several time steps (i.e., sequence length)
  - Example: Fischer & Krauss (2018) use a „deep“ network consisting of one LSTM cell
- **Can also obtain depth in a more conventional sense by stacking multiple layers of RNNs on top of each others**
- **May enable network to compute more complex representations**
  - Lower network layers learn low-level features
  - Deep layers learn more abstract, high-level features

Multi-Layer Recurrent Neural Networks

Third layer RNN

Second layer RNN

First layer RNN



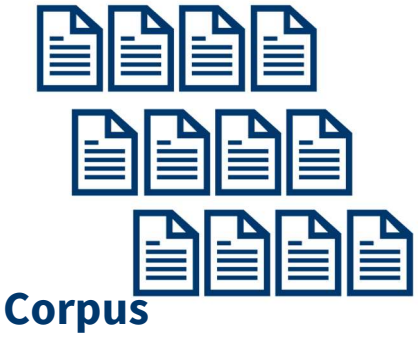
NLP Transfer Learning and State-of-the-Art

Attention mechanism and transformer architecture

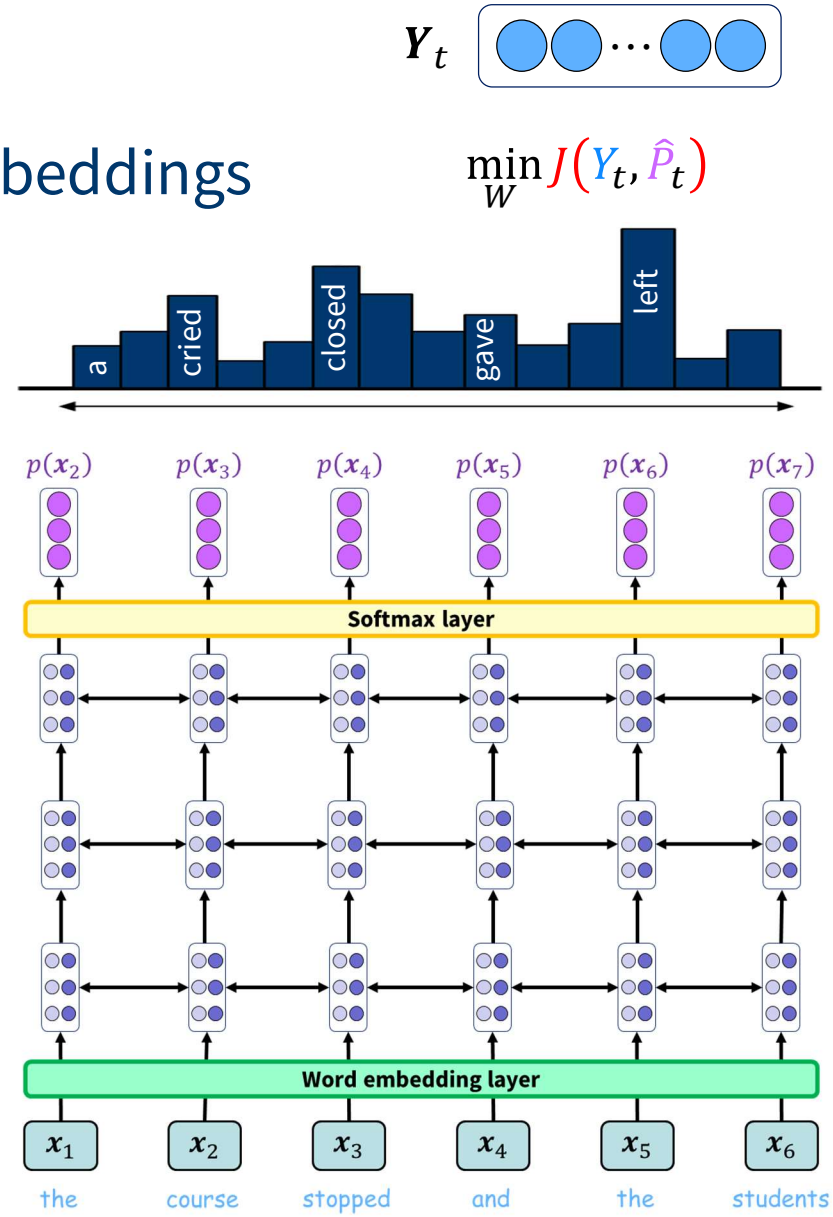
Status-Quo with NLP and RNNs by ~2018

Stacked, bidirectional RNN-type networks & embeddings

- **Embeddings to represent tokens**
- **LSTM or GRU cells to address gradient vanishing**
- **Bidirectionality to capture left & right context**
- **Stacking layers to learn more abstract patterns**
- **Self-training on large corpora**
  - Window-wise parsing of documents to obtain sequences
  - Using next token prediction as objective (i.e., LM)
  - Train weights using cross-entropy loss by backprop.

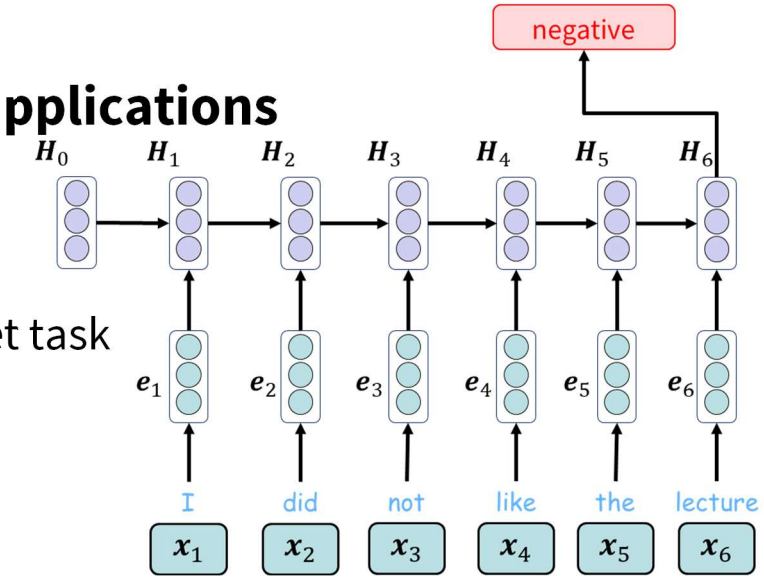


started as early as eight in
in the morning <comma> talking about
about RNNs <dot> After a daunting
daunting two hours it eventually turned
eventually turned ten and the course
the course stopped and the students
students closed their books and left



Transfer Learning in NLP

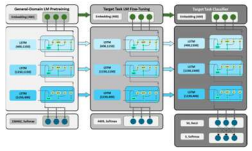
A 2018 perspective



- **Word embeddings established in many downstream applications**
- **Word embeddings are a form of transfer learning**
  - Pretrain embeddings on some (big) corpus
  - Use those embeddings as input to a network (e.g., RNN) in a target task
- **Improvement 1: raise depth**
  - Embeddings stem from a shallow network (c.f. W2V)
  - Experiences from computer vision show that top-performance is achieved with deep pretrained networks
- **Improvement 2: consider context**
  - The embedding of a word is constant no matter in which context it occurs
  - "*She sits on the bank next to the river*" vs. "*he visits his bank to apply for a loan*"

2018 – The Birth of NLP Transfer Learning

First approaches for deep NLP transfer learning

ULMFiT	ELMo	GPT	BERT	GPTv2
				
Jan 2018	Feb 2018	Jun 2018	Oct 2018	Feb 2019
1 GPU day	~ 42 GPU days	240 GPU days	256 TPU days	~2048 TPU days
fast.ai	AI2	OpenAI	Google AI	OpenAI

Stanford NLP with Deep Learning – Lecture 13, https://web.stanford.edu/class/cs224n/
Image source: http://jalammar.github.io/illustrated-bert/

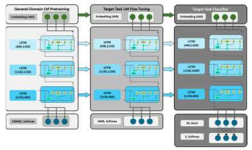
1 TPU day ~ 1.25 – 2.2 GPU days
according to Stanford lecture.

2018 – The Birth of NLP Transfer Learning

RNN-based language models dethroned

First approaches for deep NLP transfer learning

ULMFiT



Jan 2018

1 GPU day

ELMo

Feb 2018

~ 42 GPU days

Based on transformers

GPT

Jun 2018

240 GPU days

BERT

Oct 2018

256 TPU days

GPTv2

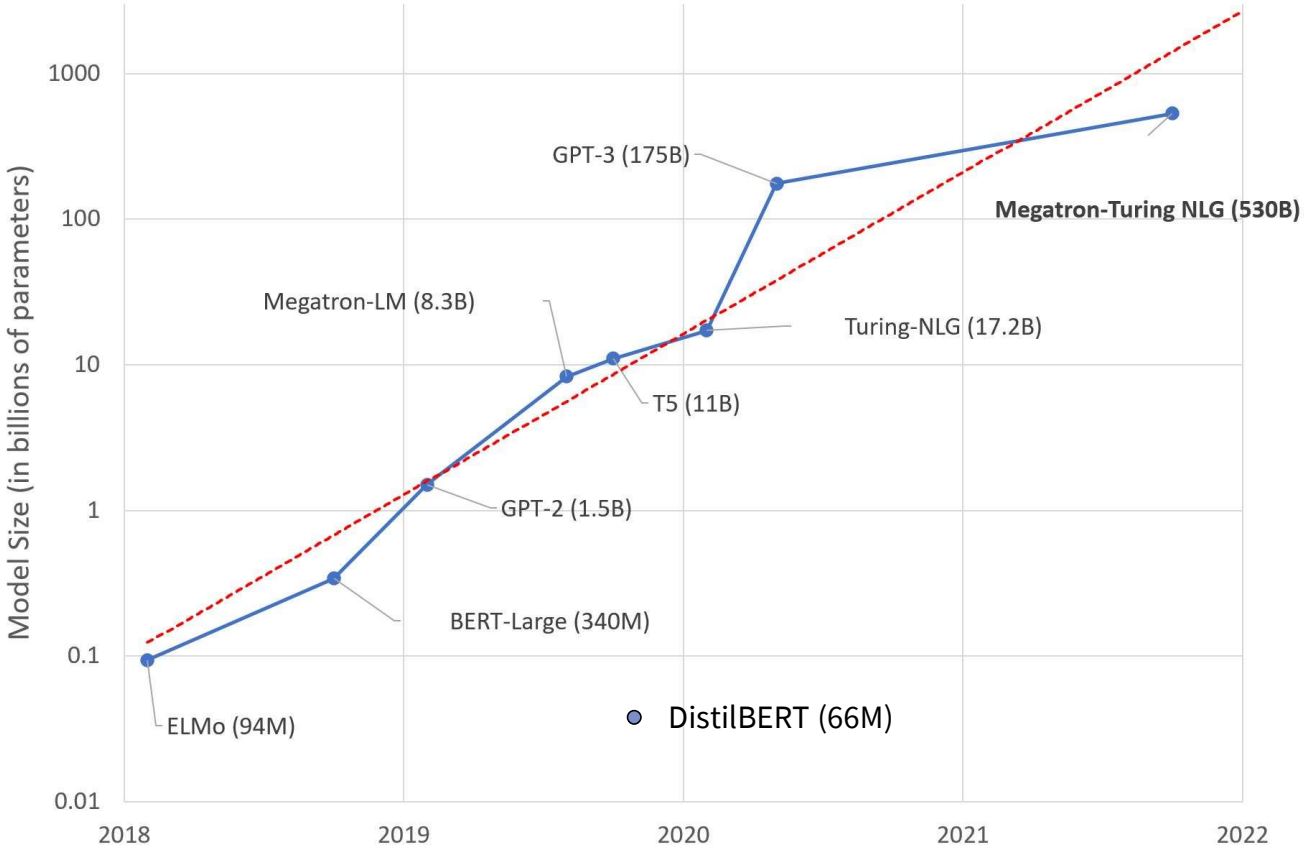
Feb 2019

~2048 TPU days

Stanford NLP with Deep Learning – Lecture 13, https://web.stanford.edu/class/cs224n/
Image source: http://jalammar.github.io/illustrated-bert/

1 TPU day ~ 1.25 – 2.2 GPU days
according to Stanford lecture.

Trend Toward Larger Models Continues



Other mega models:

- PaLM, 540B (April 4 2022) https://ai.googleblog.com/2022/04/pathways-language-model-palm-scaling-to.html
- Switch Transformer, up to 1.5T (2022) https://jmlr.org/papers/v23/21-0998.html

And increasing criticism

- Environmental footprint
- Economic sanity
- See, e.g., Merity (2019) for a critical perspective on the benefits
- Strubell et al. (2019) and Dodge et al. (2022) touch on energy consumption

Image source: https://www.microsoft.com/en-us/research/blog/using-deepspeed-and-megatron-to-train-megatron-turing-nlg-530b-the-worlds-largest-and-most-powerful-generative-language-model/

A Fifteen-Page Paper Revolutionizing Deep Learning and AI

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

Łukasz Kaiser* Google Brain lukaszkaiser@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

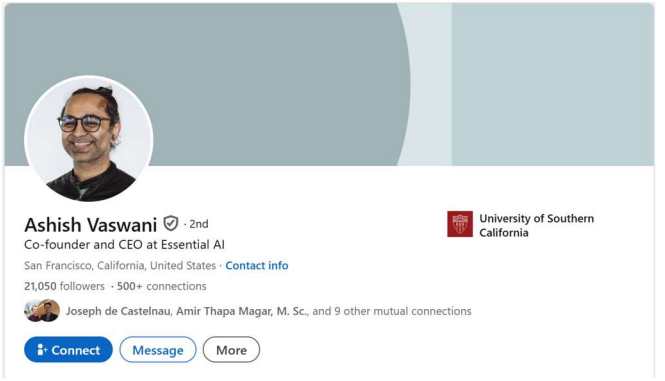
Annotated paper by Havard NLP group
http://nlp.seas.harvard.edu/2018/04/03/attention.html

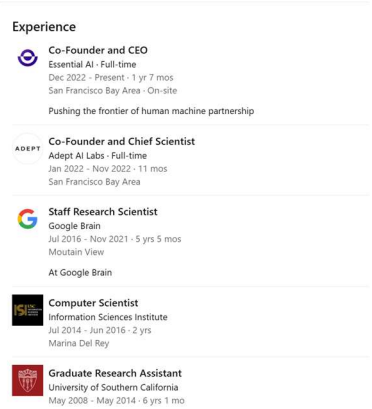
Ashish Vaswani

Startup
Verified email at fastmail.com
Deep Learning

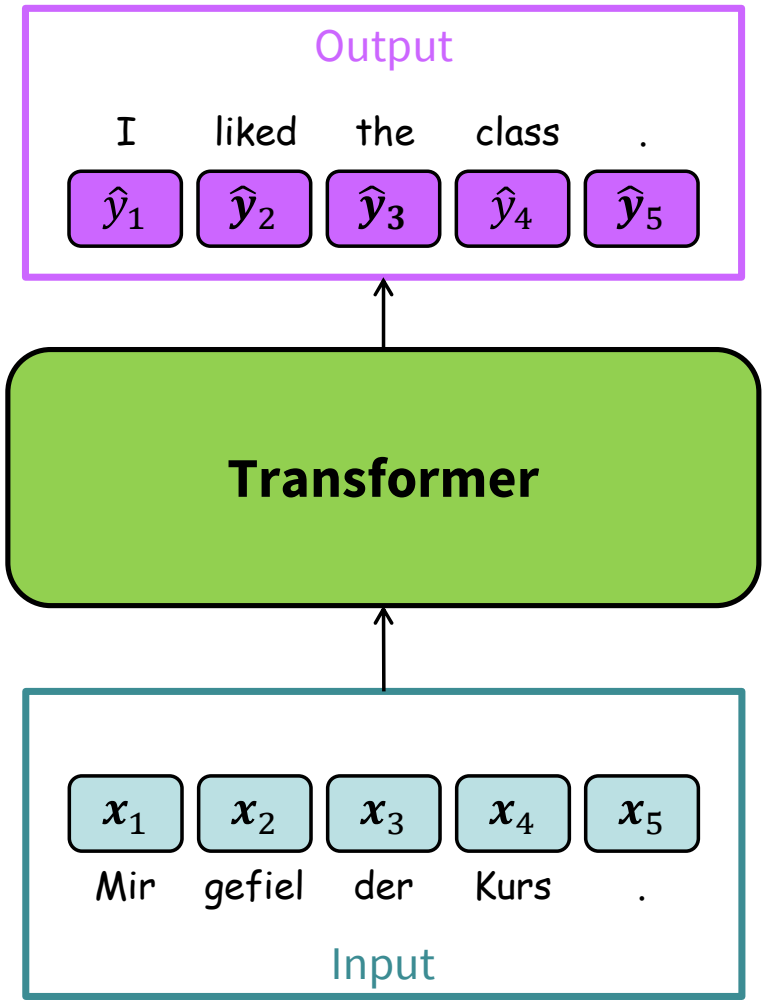
FOLLOW

TITLE	CITED BY	YEAR
Attention is all you need A Vaswani, N Shazeer, N Parmar, J Uszkoreit, L Jones, AN Gomez, ... Advances in neural information processing systems 30	127769	2017
Relational inductive biases, deep learning, and graph networks PW Battaglia, JB Hamrick, V Bapst, A Sanchez-Gonzalez, V Zambaldi, ... arXiv preprint arXiv:1806.01261	3547	2018
Self-attention with relative position representations P Shaw, J Uszkoreit, A Vaswani arXiv preprint arXiv:1803.02155	2406	2018





A Transformer Transforms Input Sequences to Output Sequences



Attention Is All You Need

NIPS 2017, pp. 6000-6010.

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
Łukasz Kaiser* Google Brain lukaszkaiser@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

Annotated paper by Havard NLP group
http://nlp.seas.harvard.edu/2018/04/03/attention.html

Sequence-to-Sequence (Seq2Seq) Architecture

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

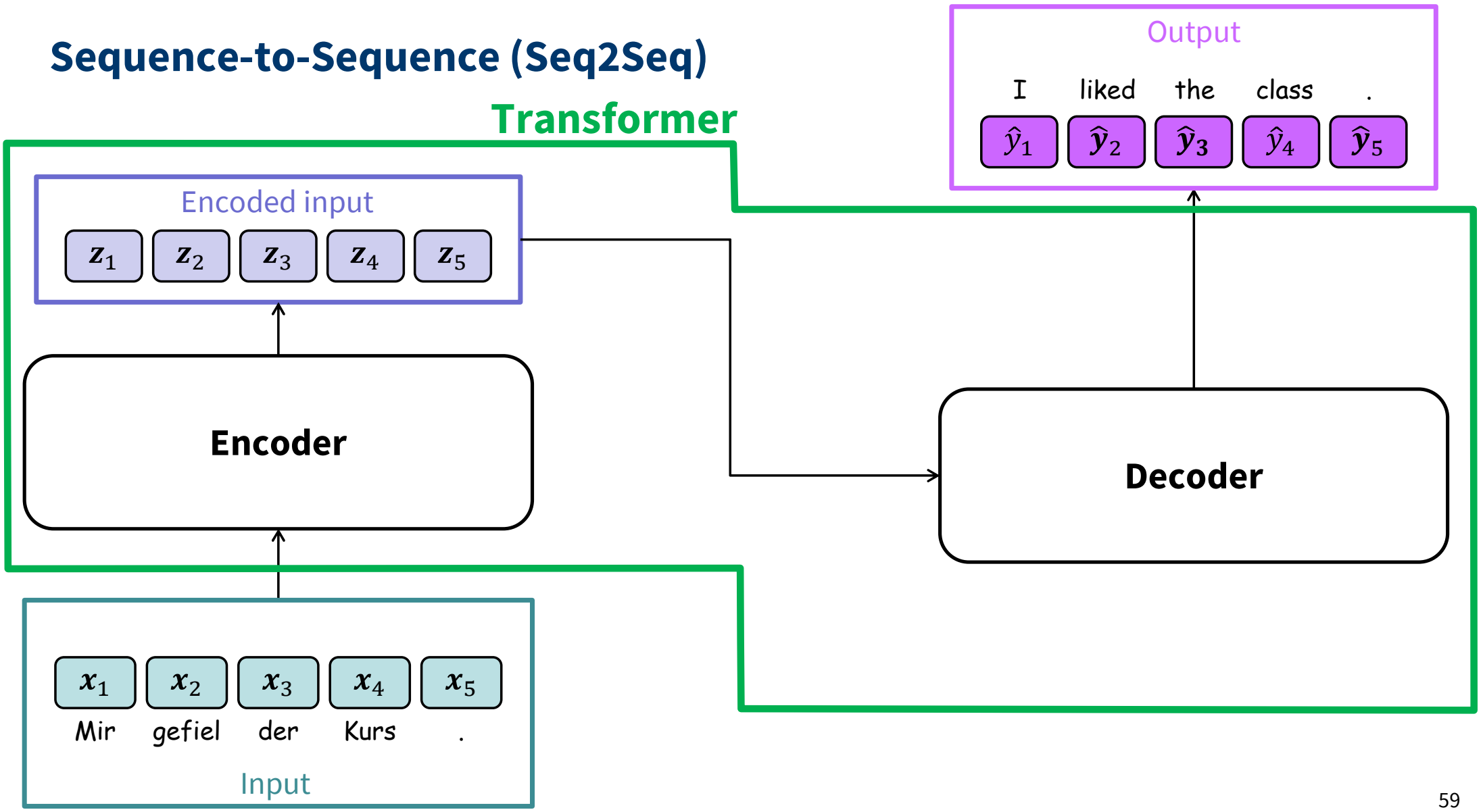
Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

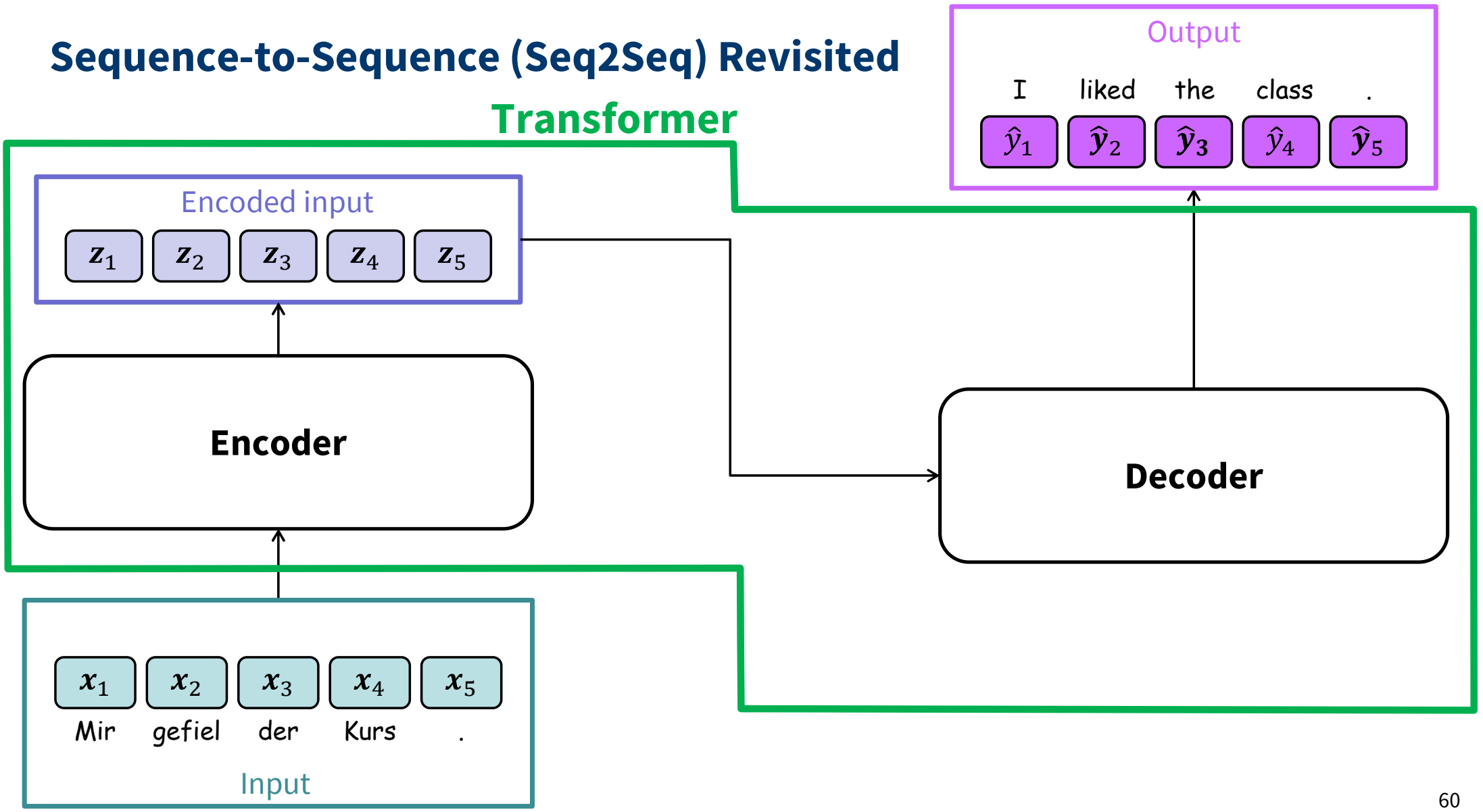
Most competitive neural sequence transduction models have an encoder-decoder structure [5, 2, 35]. Here, the encoder maps an input sequence of symbol representations $(x_1, \dots, x_n)$ ***to a sequence of continuous representations*** $z = (z_1, \dots, z_n)$***. Given*** z***, the decoder then generates an output sequence*** $(y_1, \dots, y_m)$ ***of symbols one element at a time. At each step the model is auto-regressive [10], consuming the previously generated symbols as additional input when generating the next.***

Vaswani et al. (2017) p.2

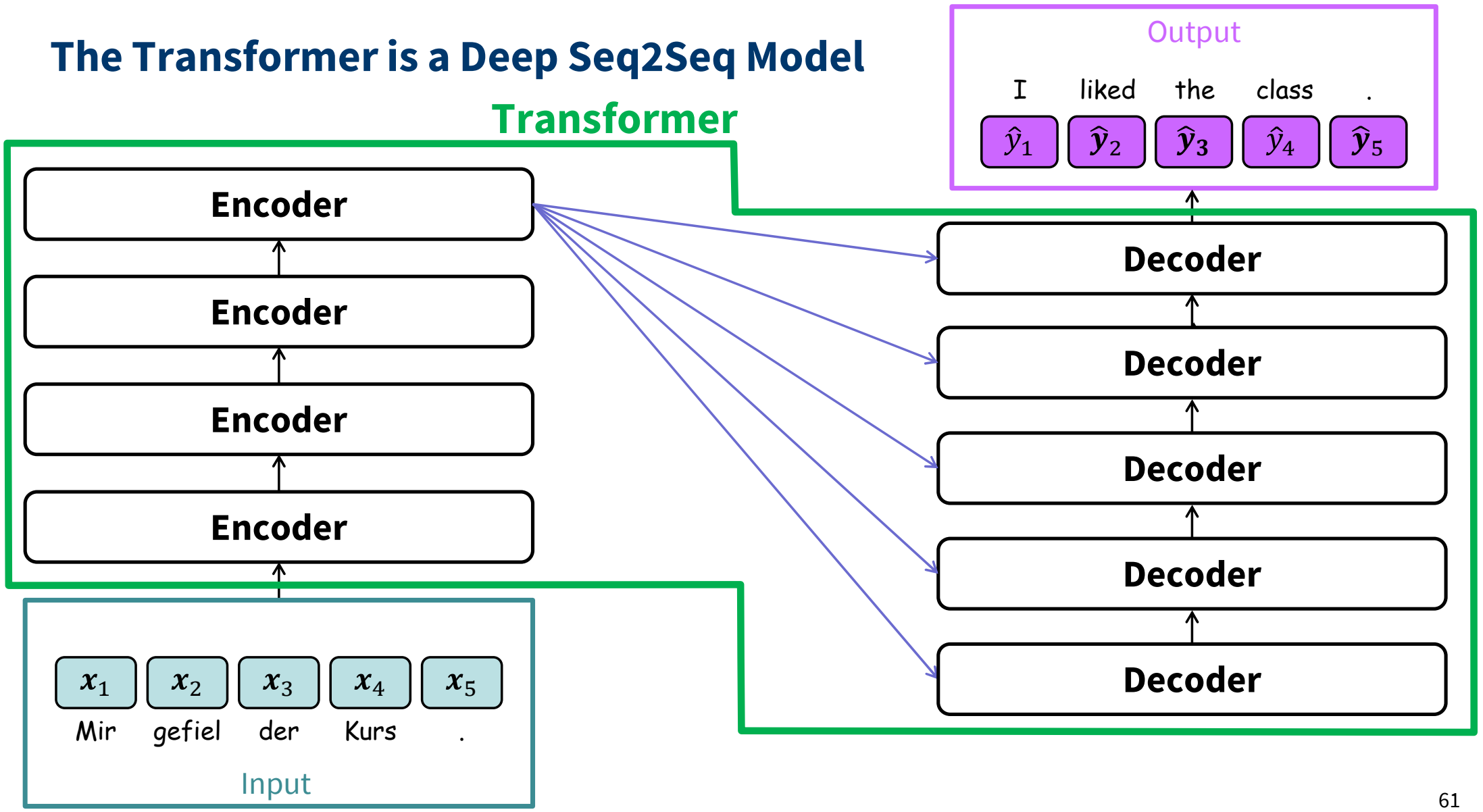
Sequence-to-Sequence (Seq2Seq)



Sequence-to-Sequence (Seq2Seq) Revisited



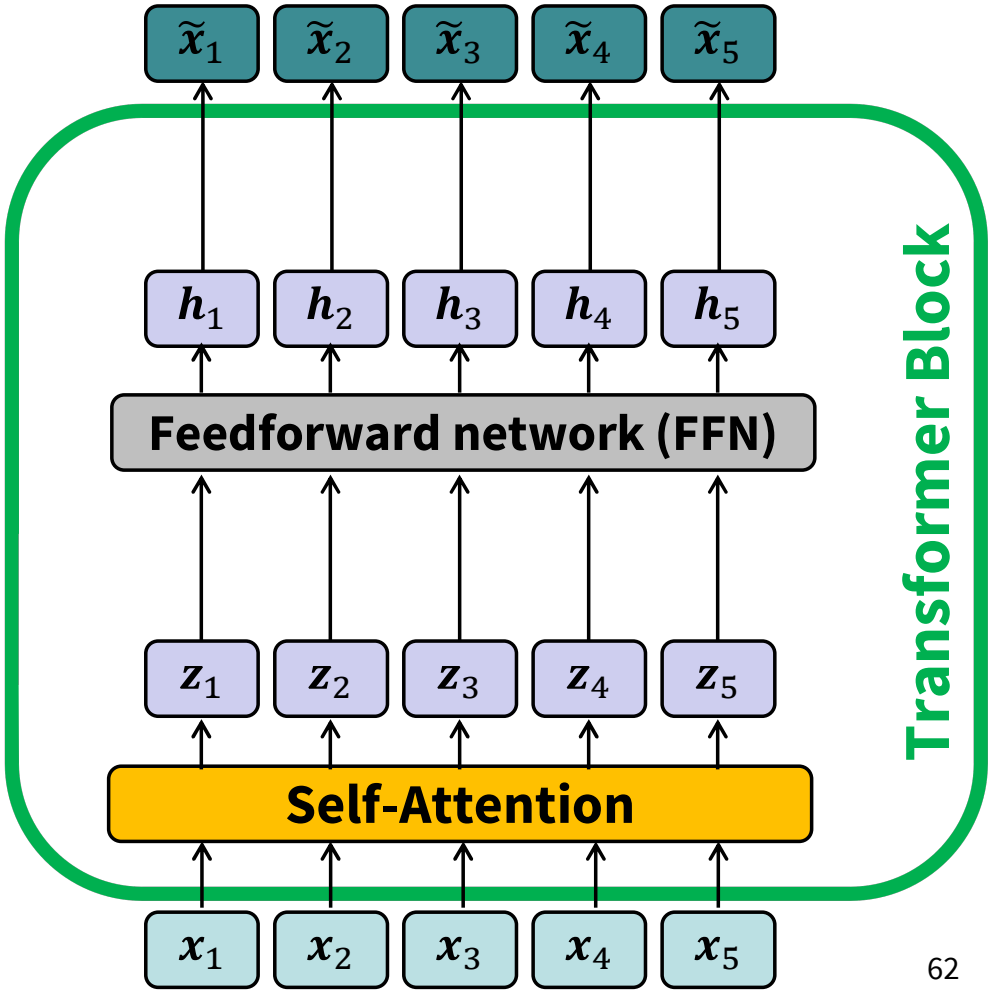
The Transformer is a Deep Seq2Seq Model



Encoder & Decoder Layers Consist of Transformer Blocks

Two sublayers in each block: attention and feedforward network (FNN)

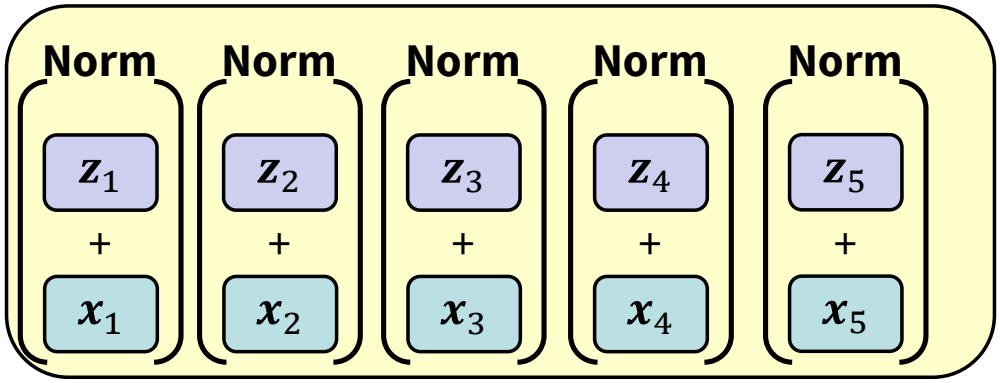
- **Input sequence of T tokens $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) \in \mathbb{R}^{T \times d}$**
  - with token embedding $\boldsymbol{x}_t \in \mathbb{R}^d$
  - Vaswani et al. (2017) set $d = 512$
- **Self-Attention layer produces a transformed sequence $\boldsymbol{Z} = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_T) \in \mathbb{R}^{T \times d}$**
  - Capture dependencies among tokens
  - Parallel processing of sequence tokens
- **Position-wise application of FFN**
  - Each token $\boldsymbol{z}_t \in \mathbb{R}^d$ as individual input
  - Adding nonlinearity (attention is a linear operation)
  - Same weights for all tokens enable parallel processing
  - Architecture ensures output $\boldsymbol{H} = (\boldsymbol{h}_1, \ldots, \boldsymbol{h}_T) \in \mathbb{R}^{T \times d}$
    - Input layer of size d
    - Hidden layer of size $4d$ with RELU activation
    - Output layer of size d
- **Transformer block sustains dimensionality**
  - Output sequence has same shape as input sequence
  - Every token is transformed (embedded)
- **Output $\widetilde{\boldsymbol{X}} = (\widetilde{\boldsymbol{x}}_1, \ldots, \widetilde{\boldsymbol{x}}_T) \in \mathbb{R}^{T \times d}$ becomes next block's input**

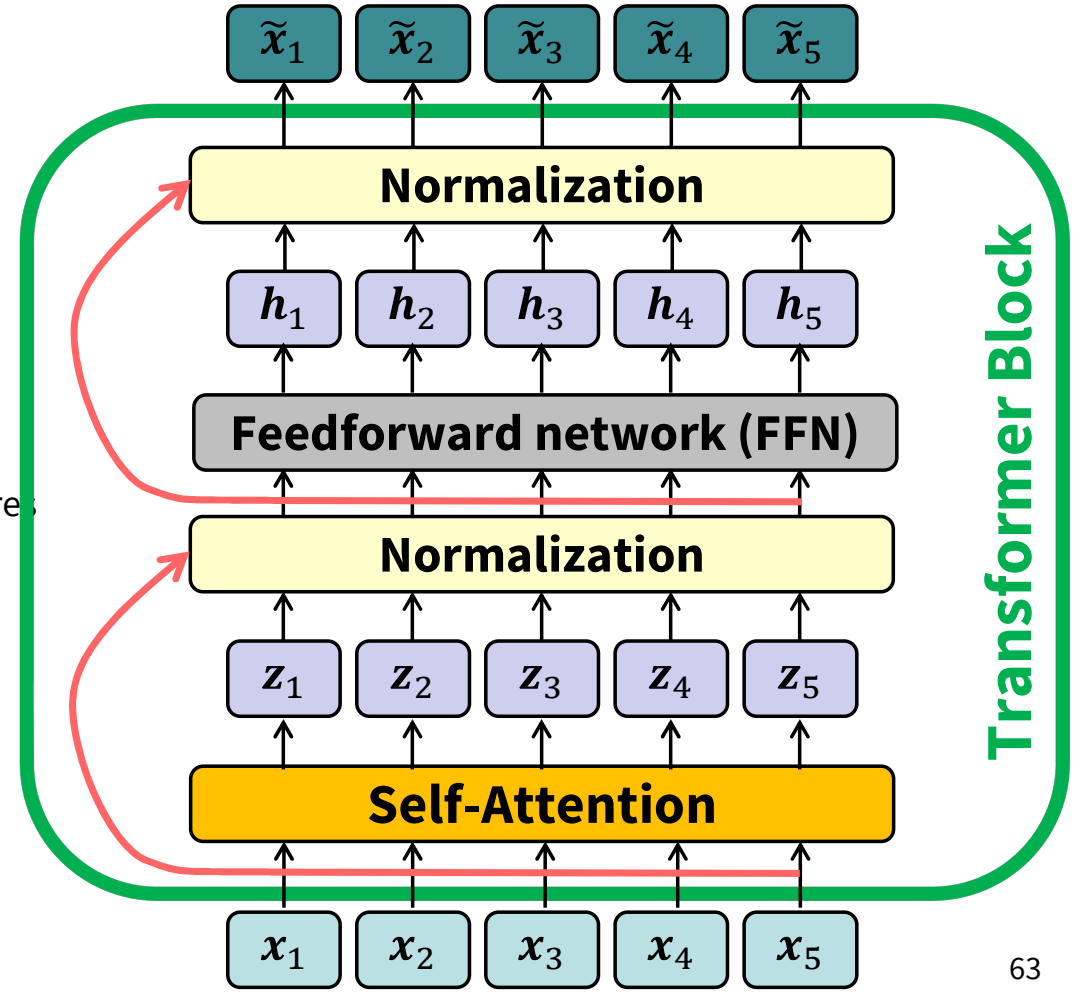


Encoder & Decoder Layers Consist of Transformer Blocks

Residual connections and layer normalization around each sublayer

- **Residual connections let data bypass a sublayer**
- **Output of sublayer becomes**
  - $\text{LayerNorm}\big(\boldsymbol{X} + \text{SubLayer}(\boldsymbol{X})\big)$
  - with $\text{SubLayer}()$ being either attention mechanism or FFN
- **Layer normalization**
  - Stabilize the learning and enable deeper networks
  - Standardizes each token independently
    - Token is represented as $\boldsymbol{x}_t \in \mathbb{R}^d$
    - Compute mean/standard deviation across d (latent) features
    - Idea: maintain contextual integrity (e.g., positional info.)





Attention Mechanism

Dynamically weighting input tokens using Query, Key, and Value

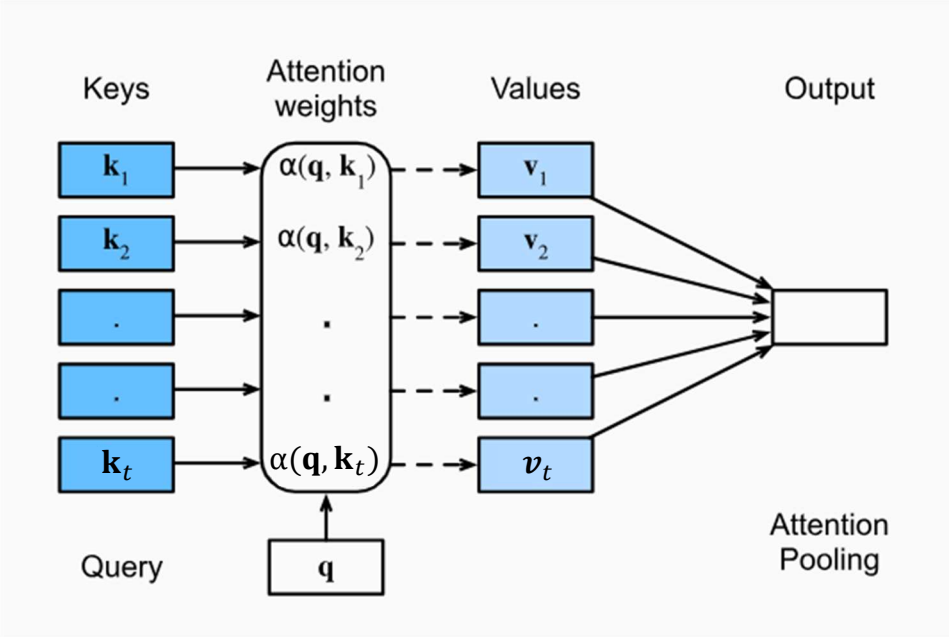
- **Formal characterization (Zhang et al. 2023)**
  - Let $\mathcal{D} \stackrel{\text{def}}{=} \{(\boldsymbol{k}_1, \boldsymbol{v}_1), \ldots, (\boldsymbol{k}_T, \boldsymbol{v}_T)\}$ denote a database of T tuples
  - and $\boldsymbol{q}$ denote a query

$$\boldsymbol{Attention}(\boldsymbol{q}, \mathcal{D}) \stackrel{\text{def}}{=} \sum_{t=1}^{T} \alpha'(\boldsymbol{q}, \boldsymbol{k}_t)\boldsymbol{v}_t$$

  - where $\alpha'(\boldsymbol{q}, \boldsymbol{k}_t) \in \mathbb{R}(t = 1, \ldots, T)$ are scalar attention weights
  - Also known as attention pooling
    - Compute a linear combination over database values
    - In a Python dictionary, for example, $\alpha'(\boldsymbol{q}, \boldsymbol{k}_t)$ is exactly 1 for one t
- **Standard setting in deep learning is to have attention weights that are nonnegative and sum to unity**
  - Easily achieved using softmax

$$\alpha(\boldsymbol{q}, \boldsymbol{k}_j) = \frac{\exp(\alpha'(\boldsymbol{q}, \boldsymbol{k}_j))}{\sum_t (\alpha'(\boldsymbol{q}, \boldsymbol{k}_t))}$$

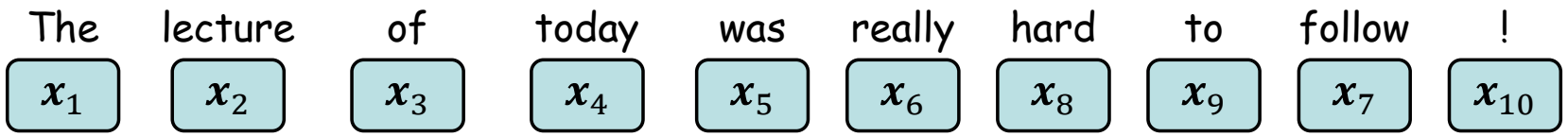
  - Implies that we can use any function $\alpha'(\boldsymbol{q}, \boldsymbol{k}_t)$
  - Several approaches/options (see, e.g., Zhang et al. (2023), Ch.11.2)

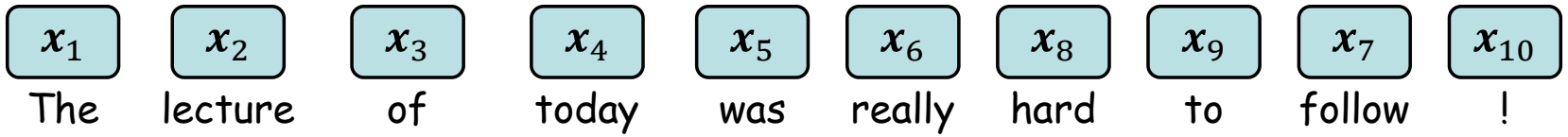


The attention mechanism computes a linear combination over values v_i via attention pooling, where weights are derived according to the compatibility between a query q and keys k_t.

The Transformer Uses *Self*-Attention

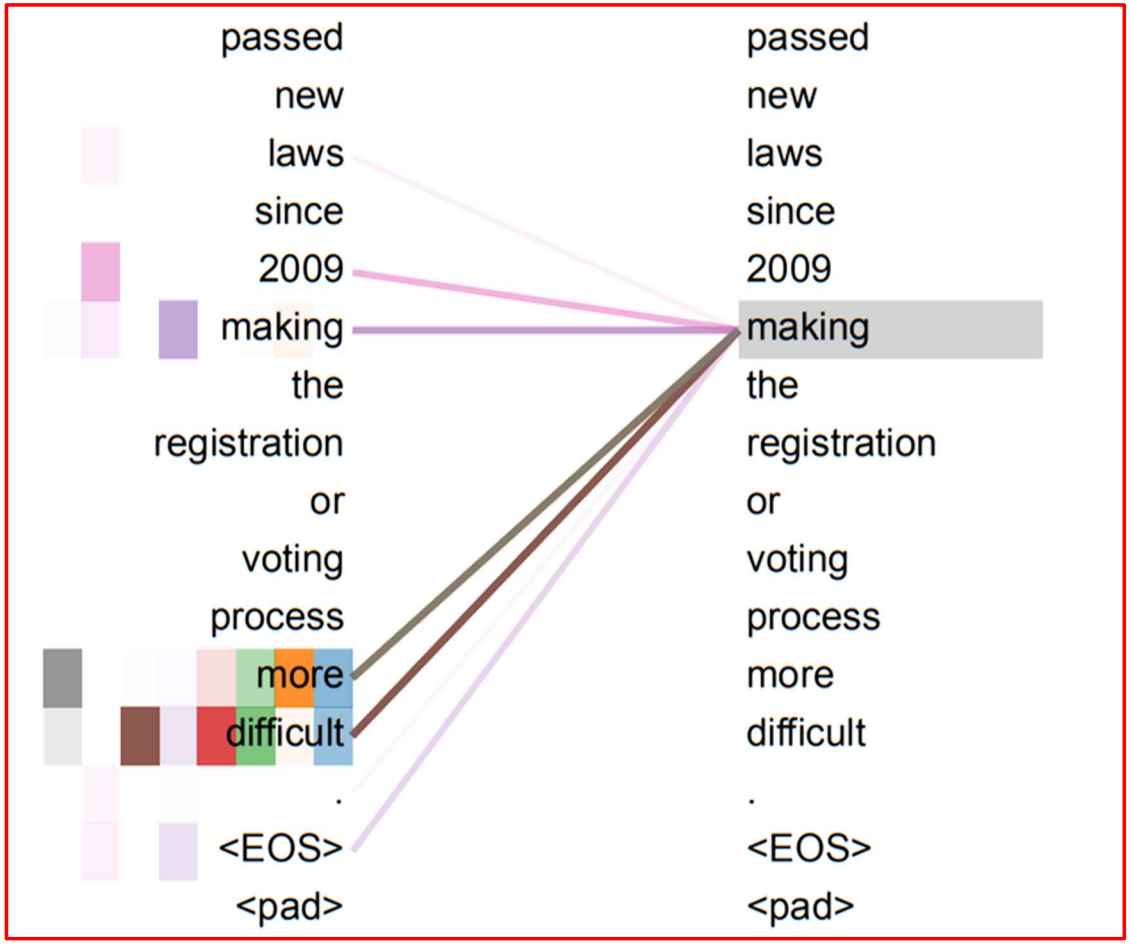
Which tokens *attend* to another?





Self-Attention in Action

Capturing dependencies among tokens of a sequence



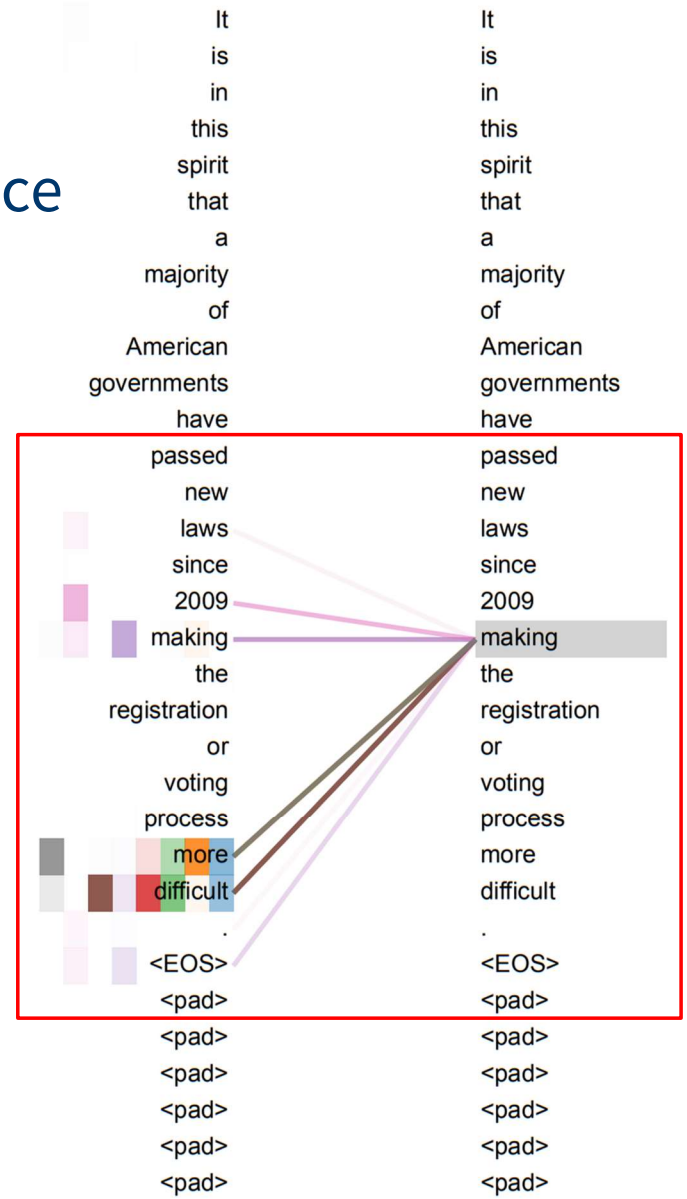
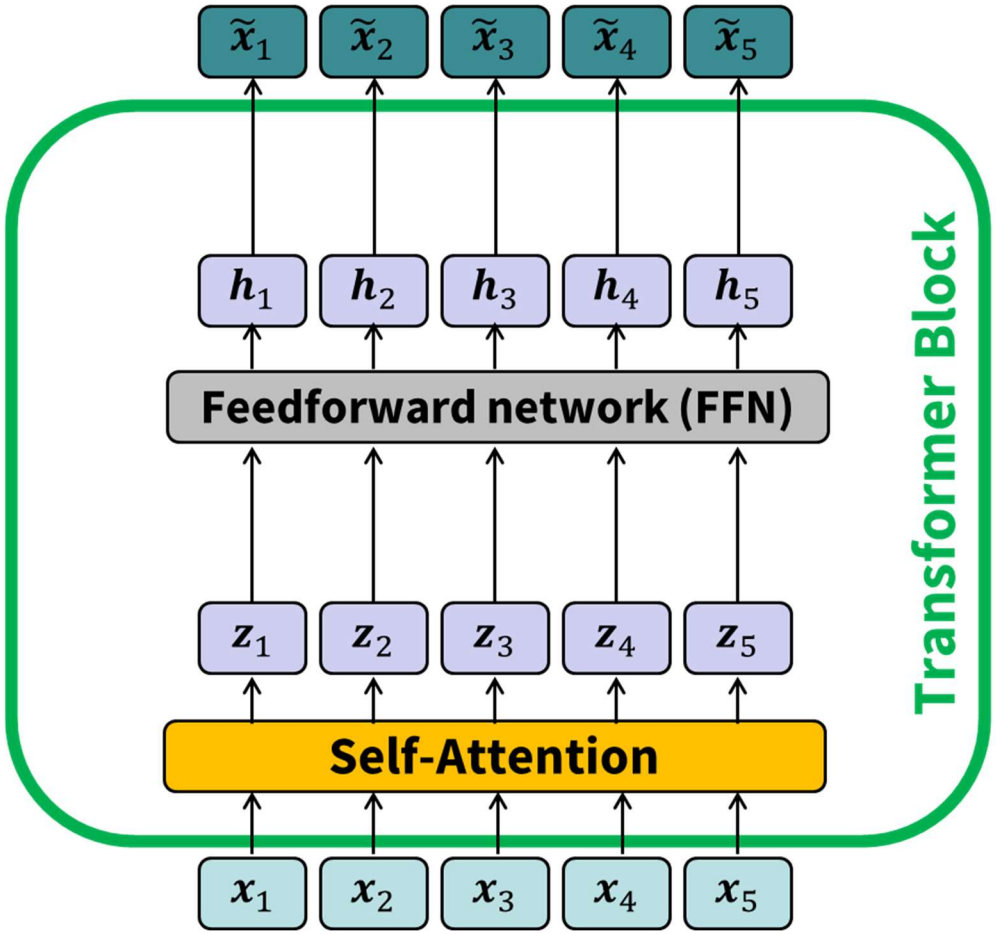


Image source: Vaswani et al. (2017)

Self-Attention in Detail

- **Input sequence of T tokens $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) \in \mathbb{R}^{T \times d}$ with token embedding $\boldsymbol{x}_t \in \mathbb{R}^d$**
- **Derive query, key, and value vectors from each token**
  - $\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t \in \mathbb{R}^{\tilde{d}}$
    - Vaswani et al. (2017) set $\tilde{d} = 64$
    - Yet another meta-parameter



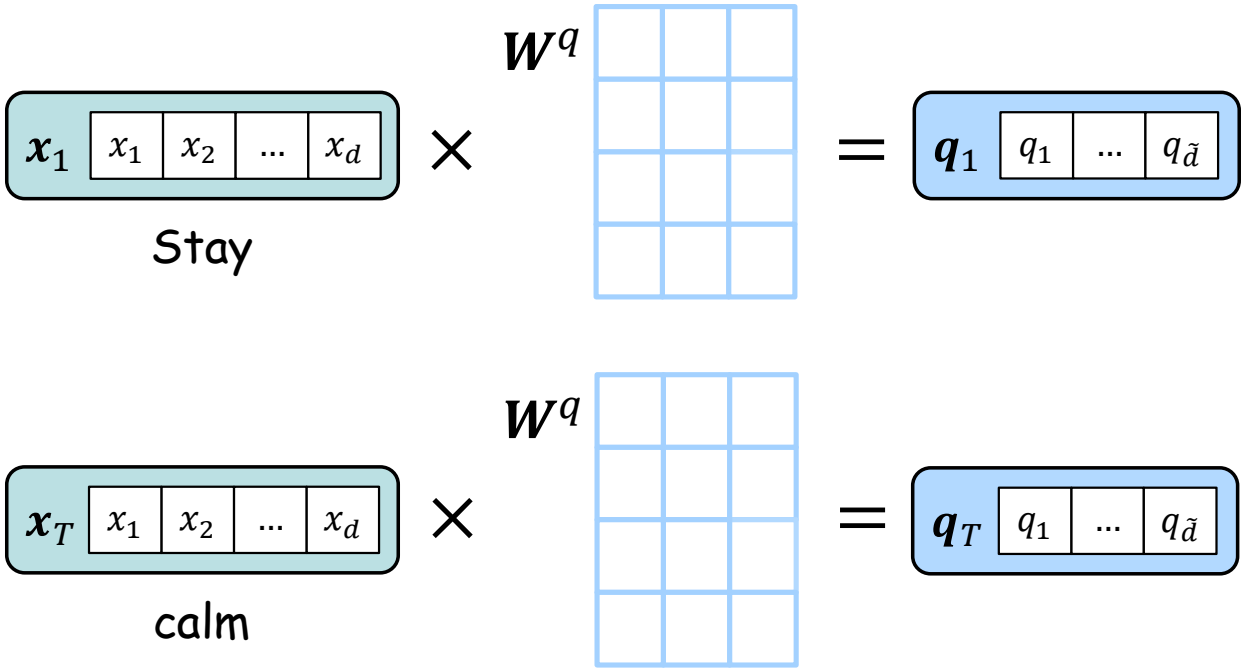
Inspired by Alammar (2020)

Self-Attention in Detail

Introduce weight matrices

- **Input sequence of T tokens $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) \in \mathbb{R}^{T \times d}$ with token embedding $\boldsymbol{x}_t \in \mathbb{R}^d$**
- **Derive query, key, and value vectors from each token**
  - $\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t \in \mathbb{R}^{\tilde{d}}$
    - Vaswani et al. (2017) set $\tilde{d} = 64$
    - Yet another meta-parameter
  - Compute as $\boldsymbol{q}_t = \boldsymbol{x}_t$

    with $\boldsymbol{W}^q \quad \in \mathbb{R}^{d \times \tilde{d}}$



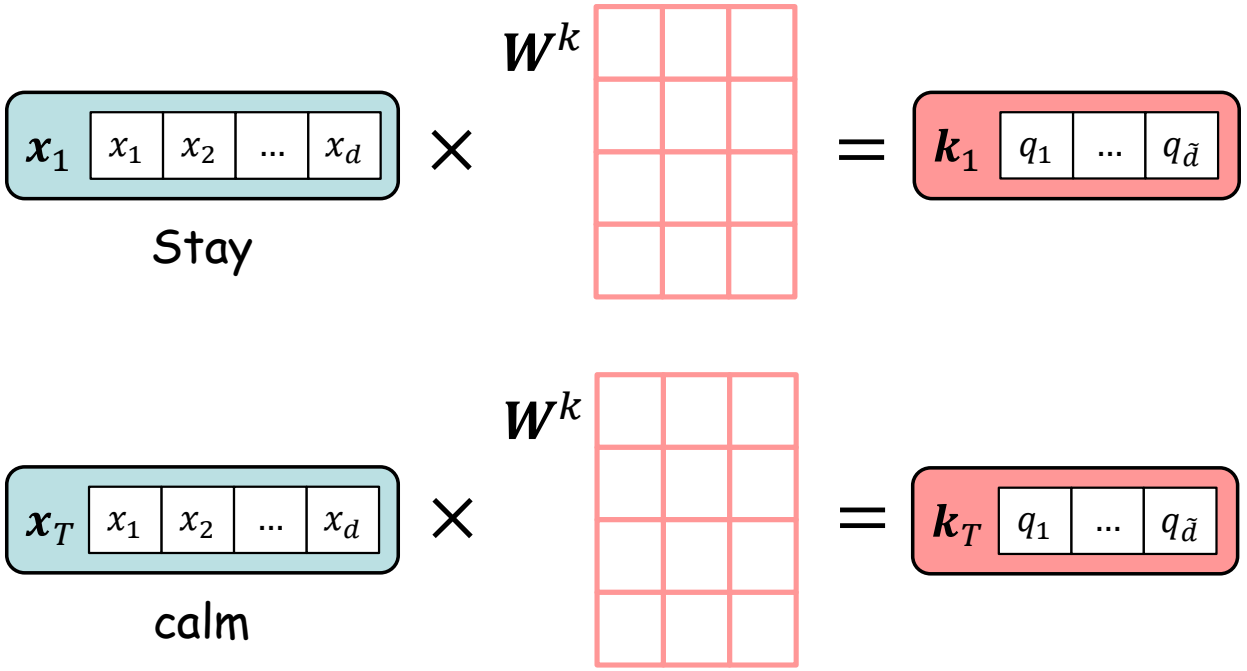
Inspired by Alammar (2020)

Self-Attention in Detail

Introduce weight matrices

- **Input sequence of T tokens $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) \in \mathbb{R}^{T \times d}$ with token embedding $\boldsymbol{x}_t \in \mathbb{R}^d$**
- **Derive query, key, and value vectors from each token**
  - $\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t \in \mathbb{R}^{\tilde{d}}$
    - Vaswani et al. (2017) set $\tilde{d} = 64$
    - Yet another meta-parameter
  - Compute as $\boldsymbol{q}_t = \boldsymbol{x}_t \boldsymbol{W}^q$, $\boldsymbol{k}_t = \boldsymbol{x}_t \boldsymbol{W}^k$,

    with $\boldsymbol{W}^q, \boldsymbol{W}^k \in \mathbb{R}^{d \times \tilde{d}}$

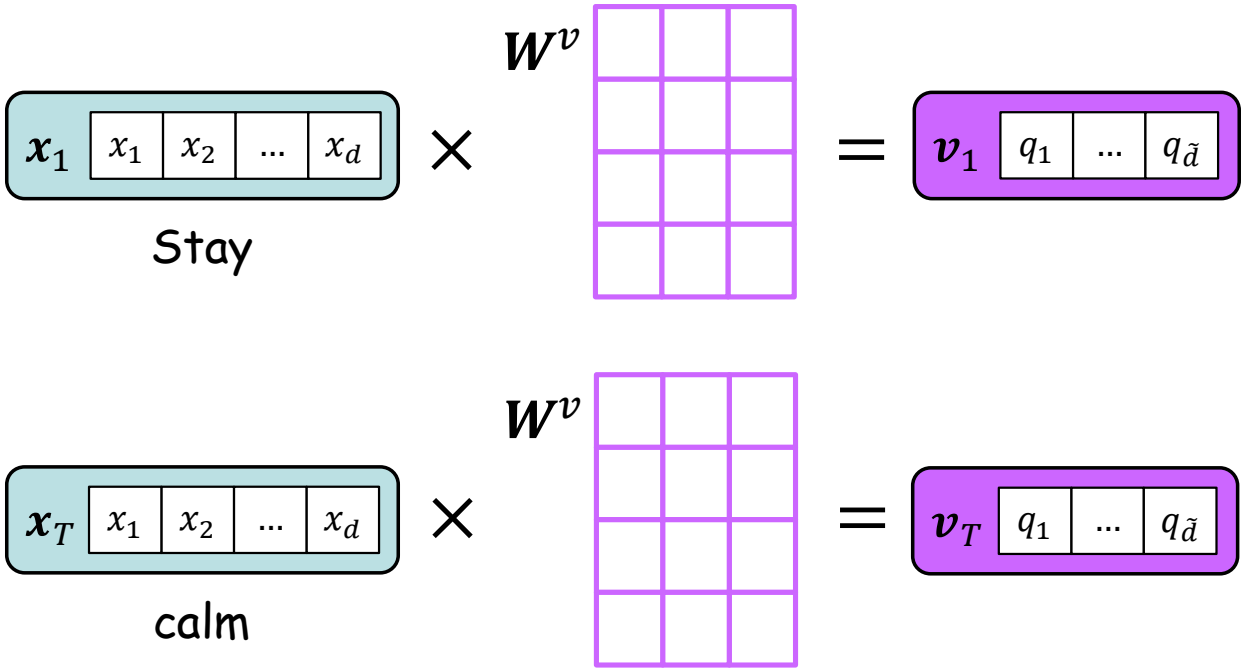


Inspired by Alammar (2020)

Self-Attention in Detail

Introduce weight matrices

- **Input sequence of T tokens $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) \in \mathbb{R}^{T \times d}$ with token embedding $\boldsymbol{x}_t \in \mathbb{R}^d$**
- **Derive query, key, and value vectors from each token**
  - $\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t \in \mathbb{R}^{\tilde{d}}$
    - Vaswani et al. (2017) set $\tilde{d} = 64$
    - Yet another meta-parameter
  - Compute as $\boldsymbol{q}_t = \boldsymbol{x}_t \boldsymbol{W}^q$, $\boldsymbol{k}_t = \boldsymbol{x}_t \boldsymbol{W}^k$, $\boldsymbol{v}_t = \boldsymbol{x}_t \boldsymbol{W}^v$ with $\boldsymbol{W}^q, \boldsymbol{W}^k, \boldsymbol{W}^v \in \mathbb{R}^{d \times \tilde{d}}$

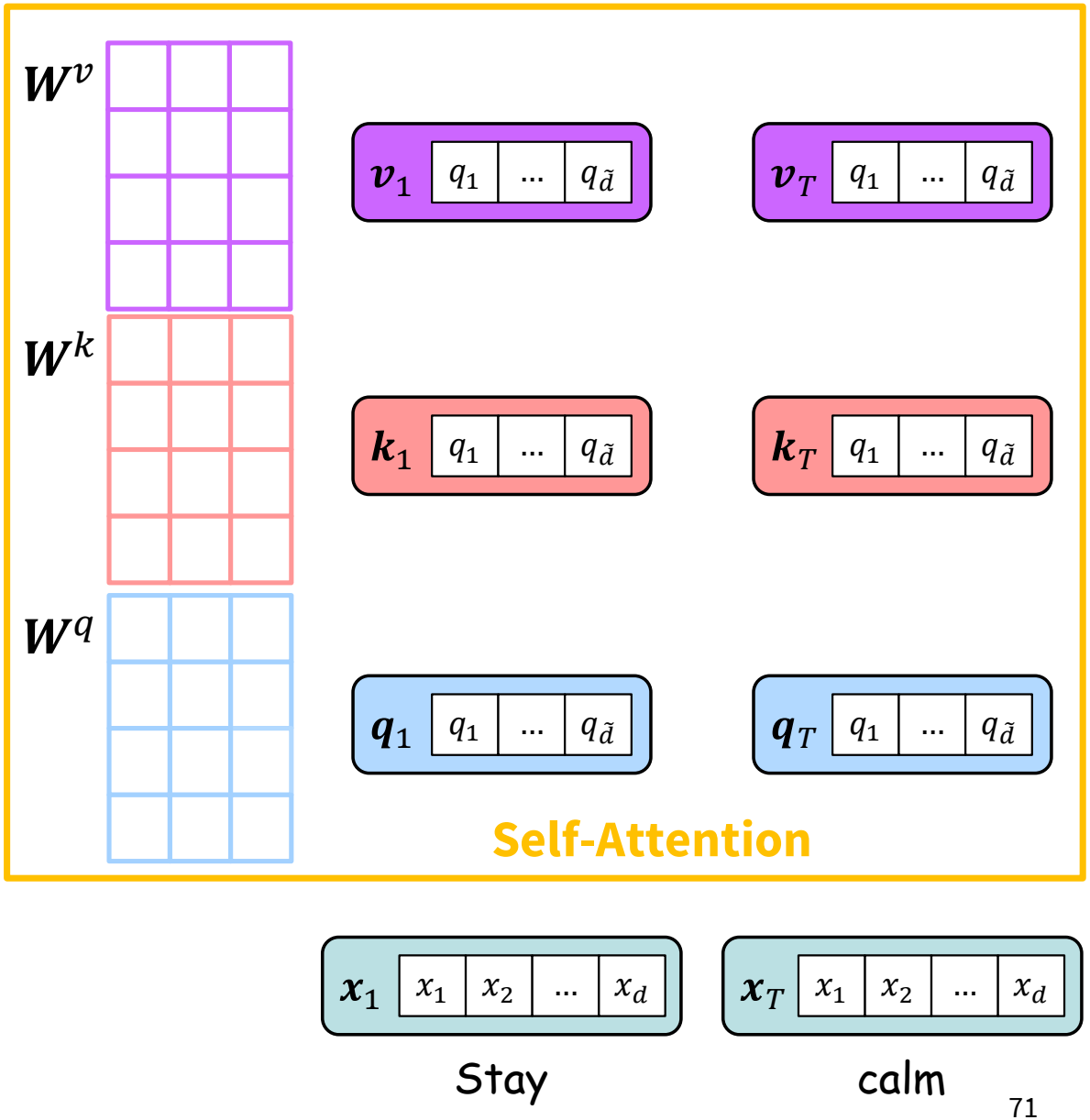


Inspired by Alammar (2020)

Self-Attention in Detail

Introduce weight matrices

- **Input sequence of T tokens $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) \in \mathbb{R}^{T \times d}$ with token embedding $\boldsymbol{x}_t \in \mathbb{R}^d$**
- **Derive query, key, and value vectors from each token**
  - $\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t \in \mathbb{R}^{\tilde{d}}$
    - Vaswani et al. (2017) set $\tilde{d} = 64$
    - Yet another meta-parameter
  - Compute as $\boldsymbol{q}_t = \boldsymbol{x}_t \boldsymbol{W}^q$, $\boldsymbol{k}_t = \boldsymbol{x}_t \boldsymbol{W}^k$, $\boldsymbol{v}_t = \boldsymbol{x}_t \boldsymbol{W}^v$ with $\boldsymbol{W}^q, \boldsymbol{W}^k, \boldsymbol{W}^v \in \mathbb{R}^{d \times \tilde{d}}$
  - The matrices are trainable weights

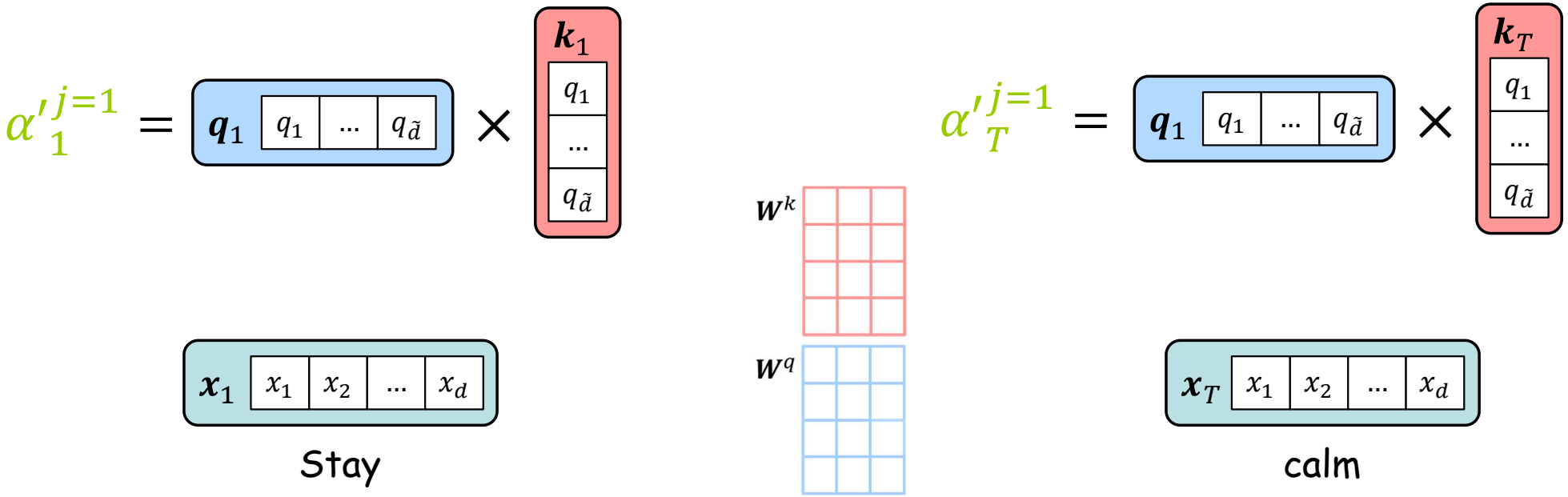


Inspired by Alammar (2020)

Self-Attention in Detail

Say we focus on token $j = 1$

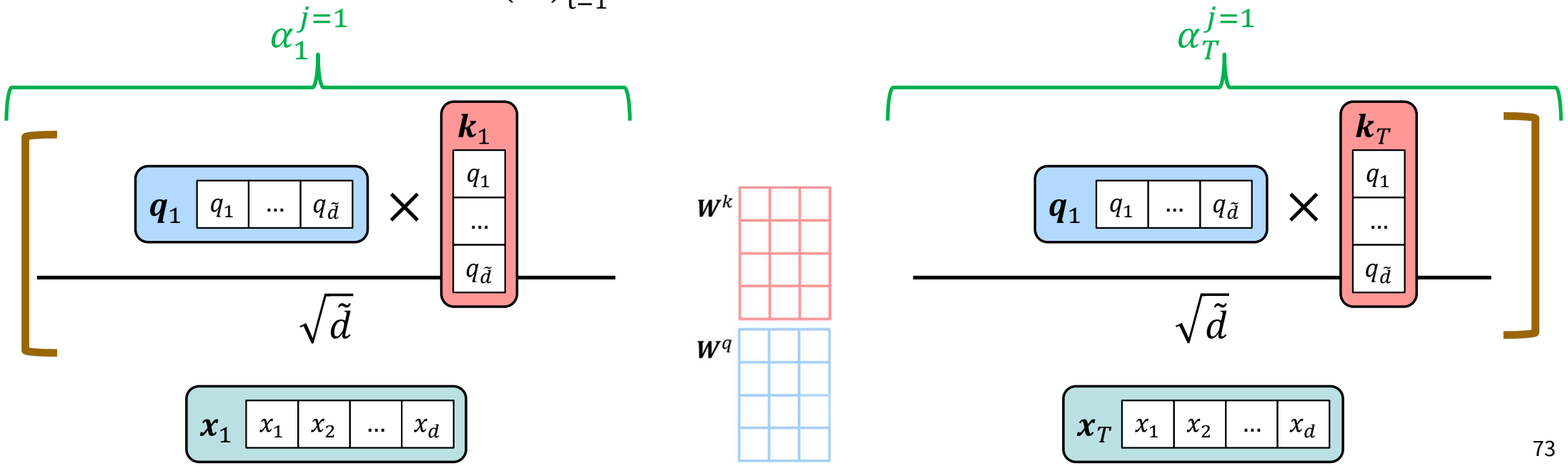
- **Compute raw scores α'^{j}_{t} as dot product of query and key vectors $\boldsymbol{q}_j \cdot \boldsymbol{k}_t^{\top} \forall t = 1, \ldots, T$**
  - Recall that the dot product is a measure of similarity
  - High values at same positions produce high output at this position and vice versa



Scaled Dot Product Attention

Say we focus on token $j = 1$

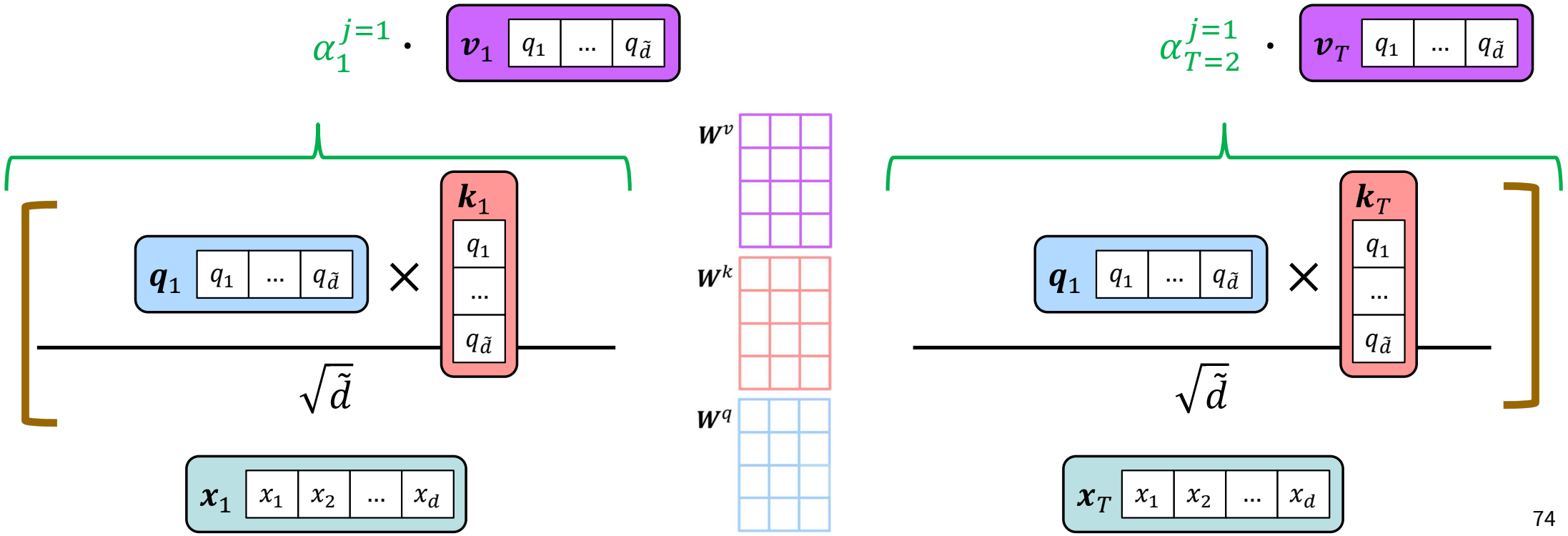
- **Normalize raw attention scores using softmax to ensure they sum to unity**
  - $\alpha_t^j = \exp\left(\frac{q_j \times k_t^\top}{\sqrt{\tilde{d}}}\right) \Big/ \sum_t \exp\left(\frac{q_j \times k_t^\top}{\sqrt{\tilde{d}}}\right)$
  - Normalized scores $\boldsymbol{\alpha}^j = \left(\alpha_t^j\right)_{t=1}^{\mathrm{T}}$ determine how much each token is expressed at position j



Scaled Dot Product Attention

Say we focus on token $j = 1$

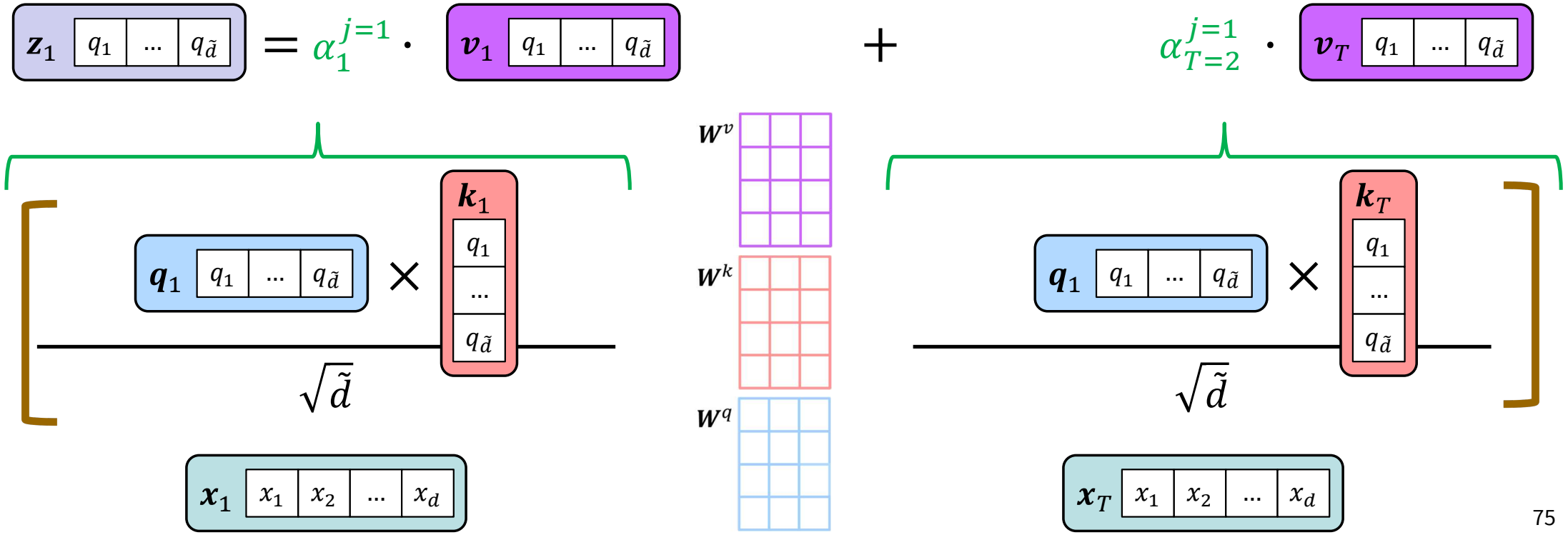
■ **Multiply attention scores with value vectors** $\alpha_t^j v_t$



Scaled Dot Product Attention

Say we focus on token $j = 1$

- **Multiply attention scores with value vectors $\alpha_t^j v_t$**
- **Compute attention output for this position, z_j, as weighted sum $z_j = \sum_{t=1}^{T} \alpha_t^j v_t$**



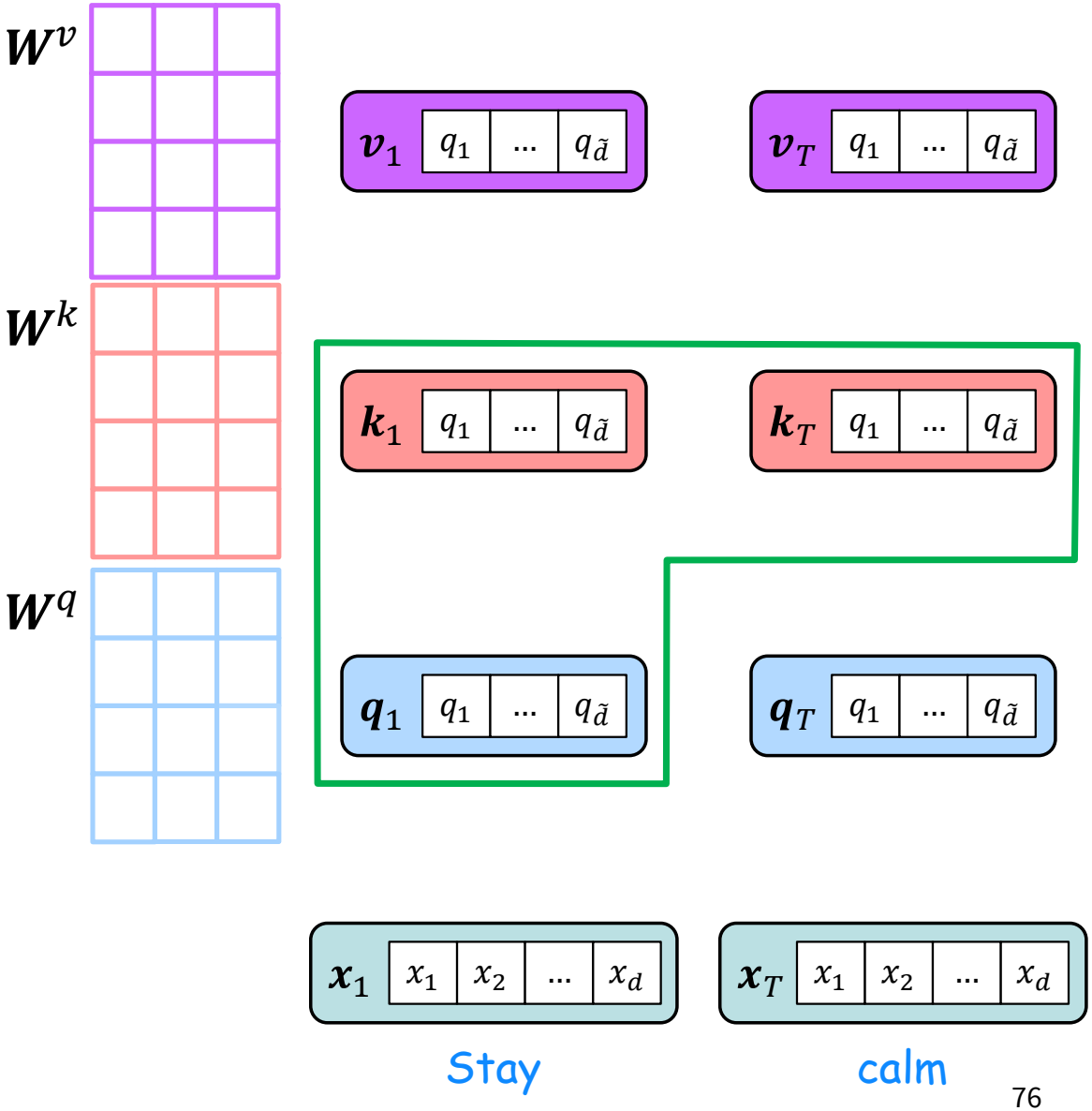
Scaled Dot Product Attention

Summary for tokens $j = 1$

- **For** $j = 1$ we use q_1 to compute scores

$$\boldsymbol{z}_1 \; [q_1 \; \dots \; q_{\tilde{d}}] = \alpha_1^{j=1} \times \boldsymbol{v}_1 \; [q_1 \; \dots \; q_{\tilde{d}}] + \alpha_2^{j=1} \times \boldsymbol{v}_T \; [q_1 \; \dots \; q_{\tilde{d}}]$$

$$\boldsymbol{\alpha}^{j=1} = \text{softmax}\left(\frac{\boldsymbol{q}_1 \boldsymbol{k}_1}{\sqrt{\tilde{d}}}, \dots, \frac{\boldsymbol{q}_1 \boldsymbol{k}_T}{\sqrt{\tilde{d}}}\right)$$



Inspired by Alammar (2020)

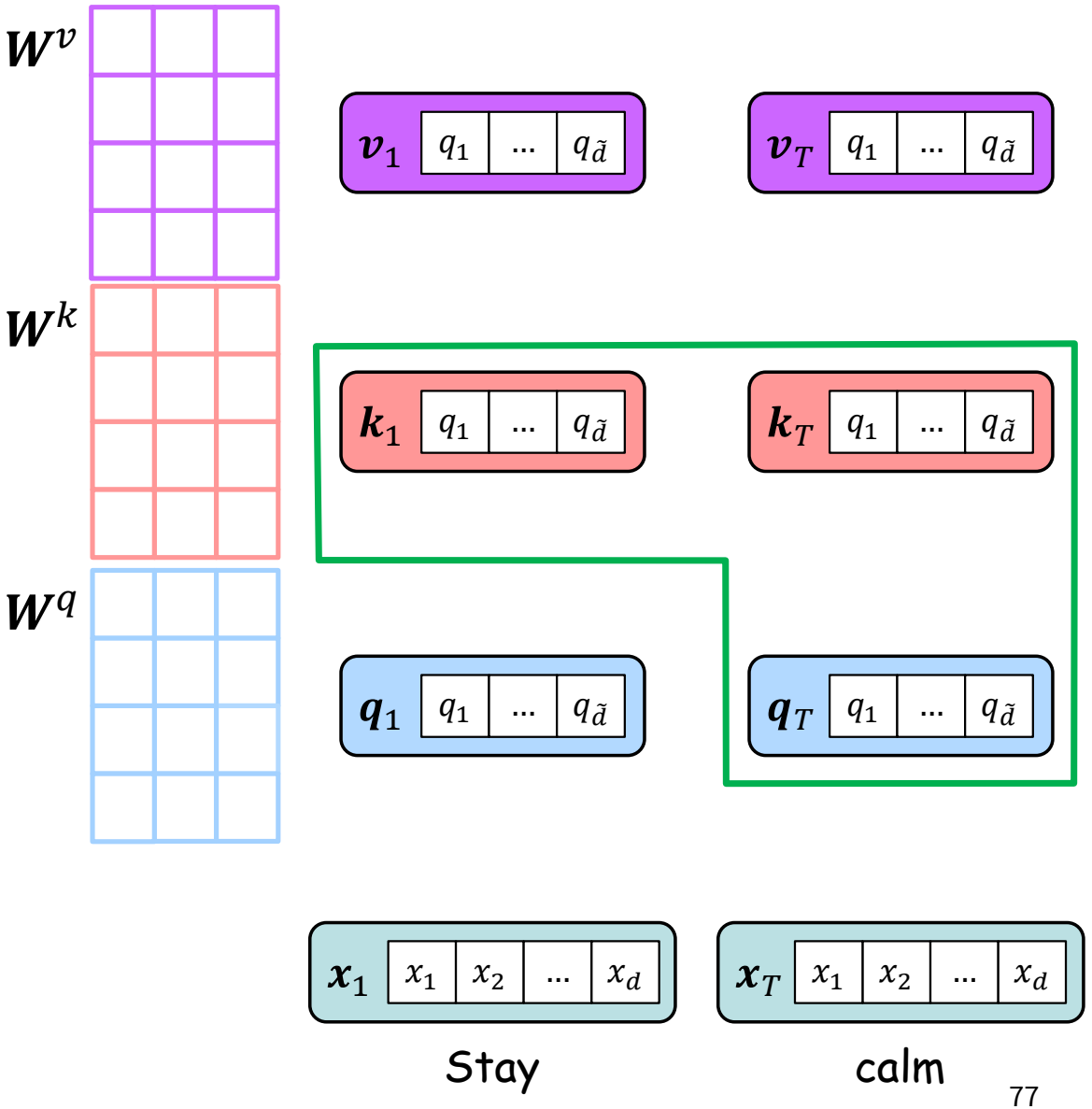
Scaled Dot Product Attention

Proceed alike for other tokens j

■ **Now let $j = 2 = T$**

$$\boldsymbol{z}_2 \; [q_1 \; \dots \; q_{\tilde{d}}] = \alpha_1^{j=2} \times \boldsymbol{v}_1 \; [q_1 \; \dots \; q_{\tilde{d}}] + \alpha_2^{j=2} \times \boldsymbol{v}_T \; [q_1 \; \dots \; q_{\tilde{d}}]$$

$$\boldsymbol{\alpha}^{j=2} = \text{softmax}\left(\frac{\boldsymbol{q}_2 \boldsymbol{k}_1}{\sqrt{\tilde{d}}}, \dots, \frac{\boldsymbol{q}_2 \boldsymbol{k}_T}{\sqrt{\tilde{d}}}\right)$$

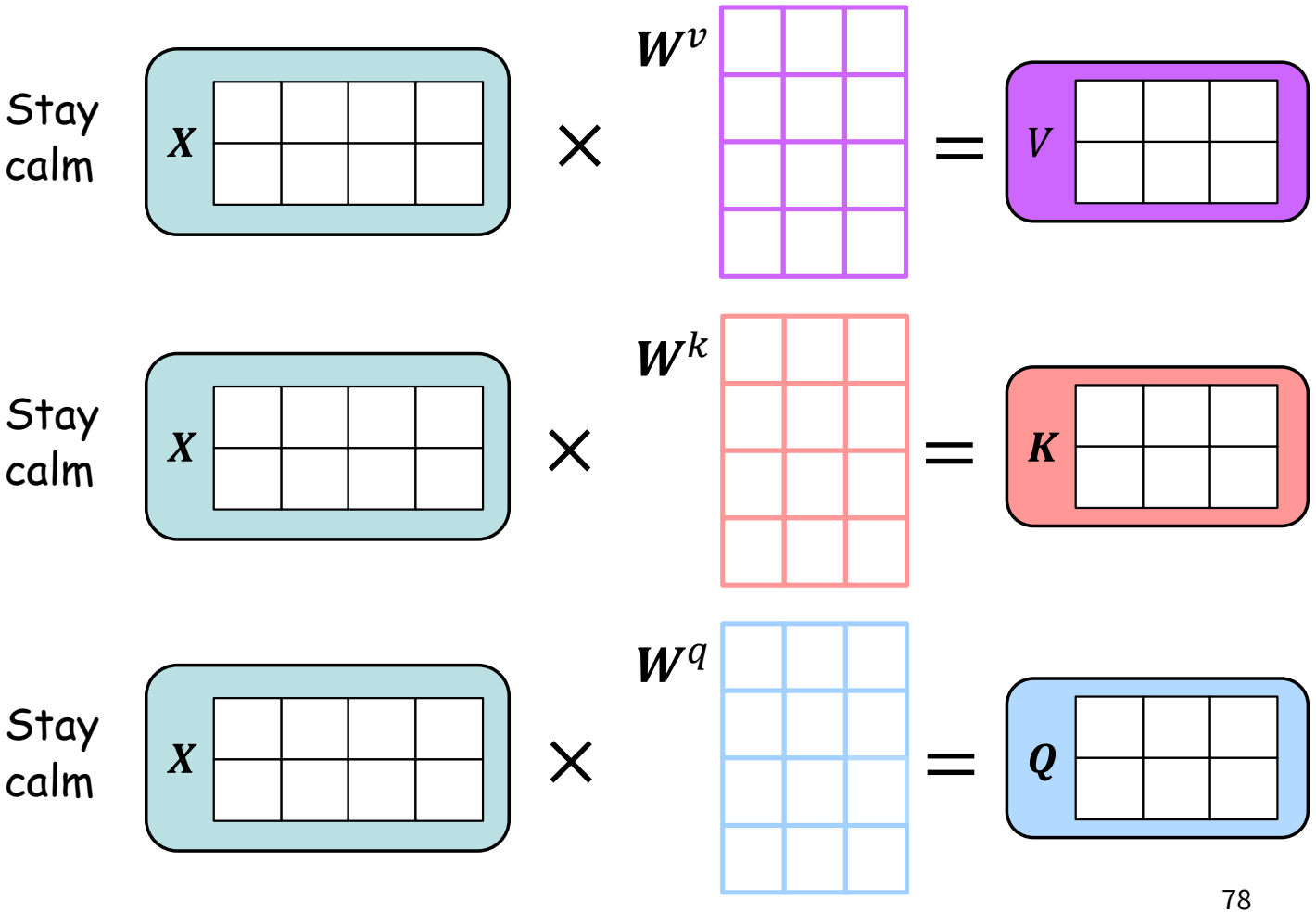


Inspired by Alammar (2020)

Vectorize Calculations for Efficiency

In practice, we compute all attention outputs in parallel

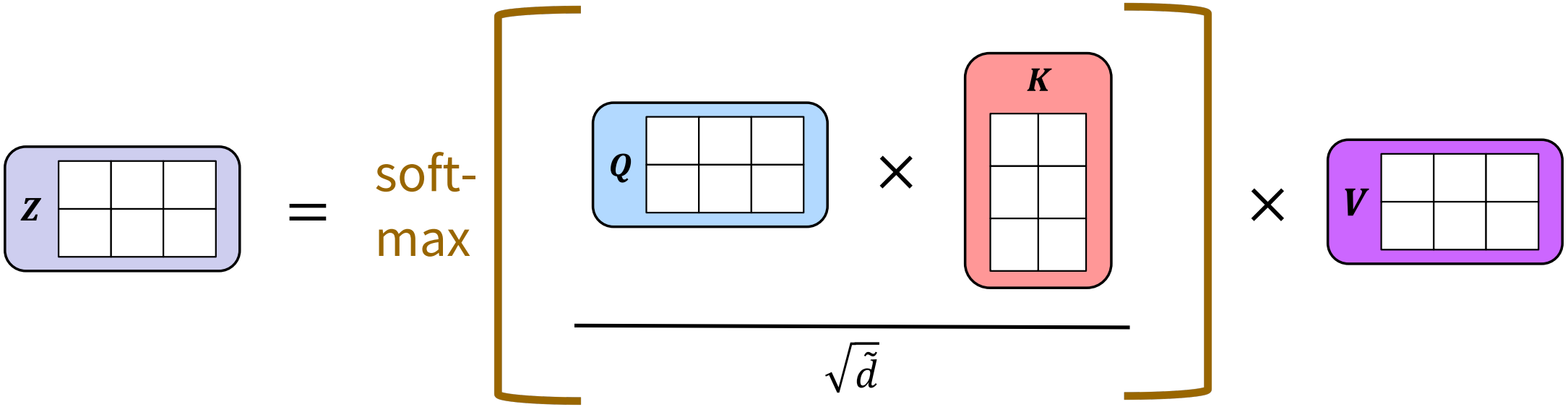
- **Input matrix of stacked token embeddings**
- **Compute query, key, and value matrices for entire input sequence**
  - No recurrence
  - Highly efficient



Inspired by Alammar (2020)

Vectorize Calculations for Efficiency

Scaled dot product attention in matric format



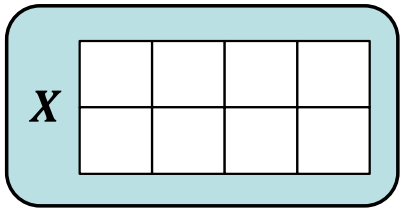
Inspired by Alammar (2020)

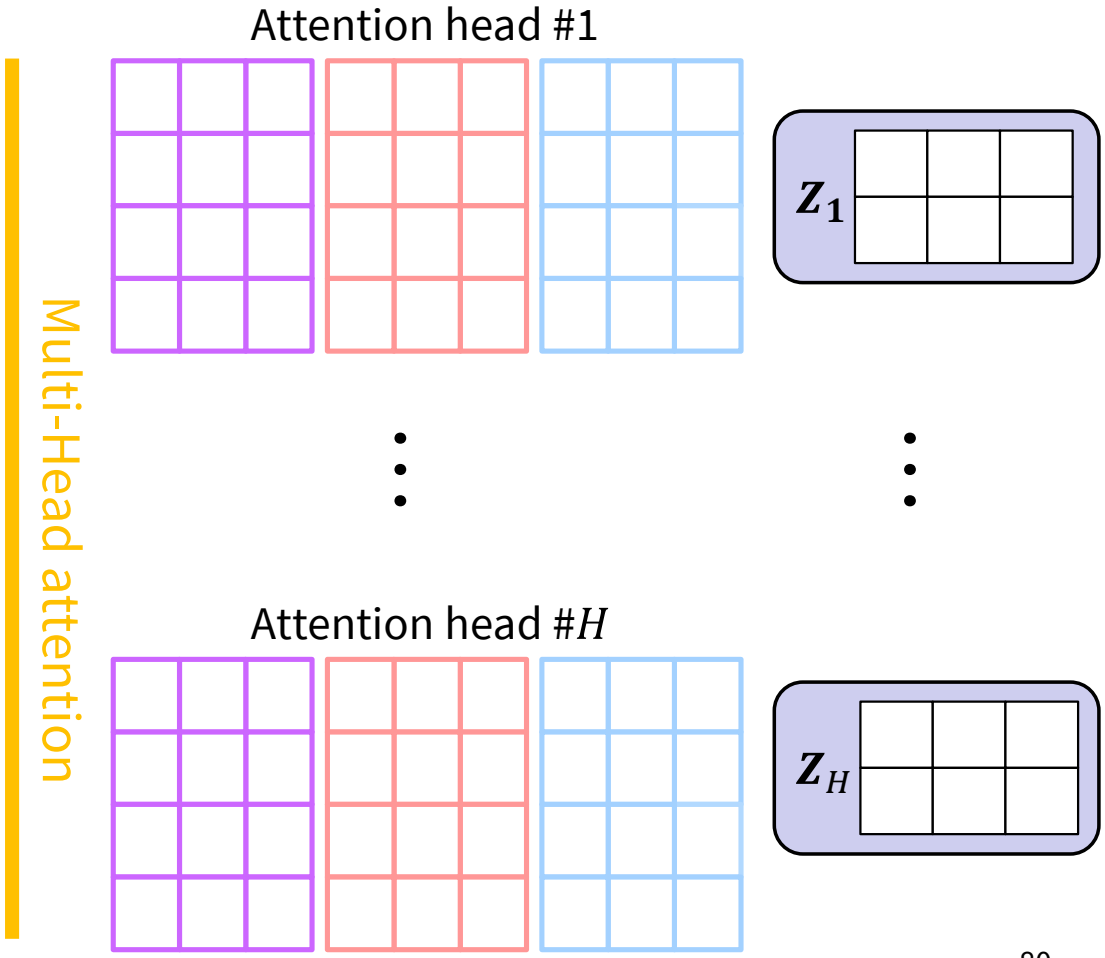
Multi-Head Attention

Repeat attention calculations raises representational capacity

- **Scaled dot product attention**
  - Transformation of of one input sequence, X into one output sequence, Z
  - The computational units performing this operations are called an *attention head*
- **The architecture of Vaswani et al. (2017) comprises 8 *attention heads***
  - More heads→more weights→more flexibility
  - Each head can focus on different dependencies
  - Attention layer can map input to multiple sub-spaces

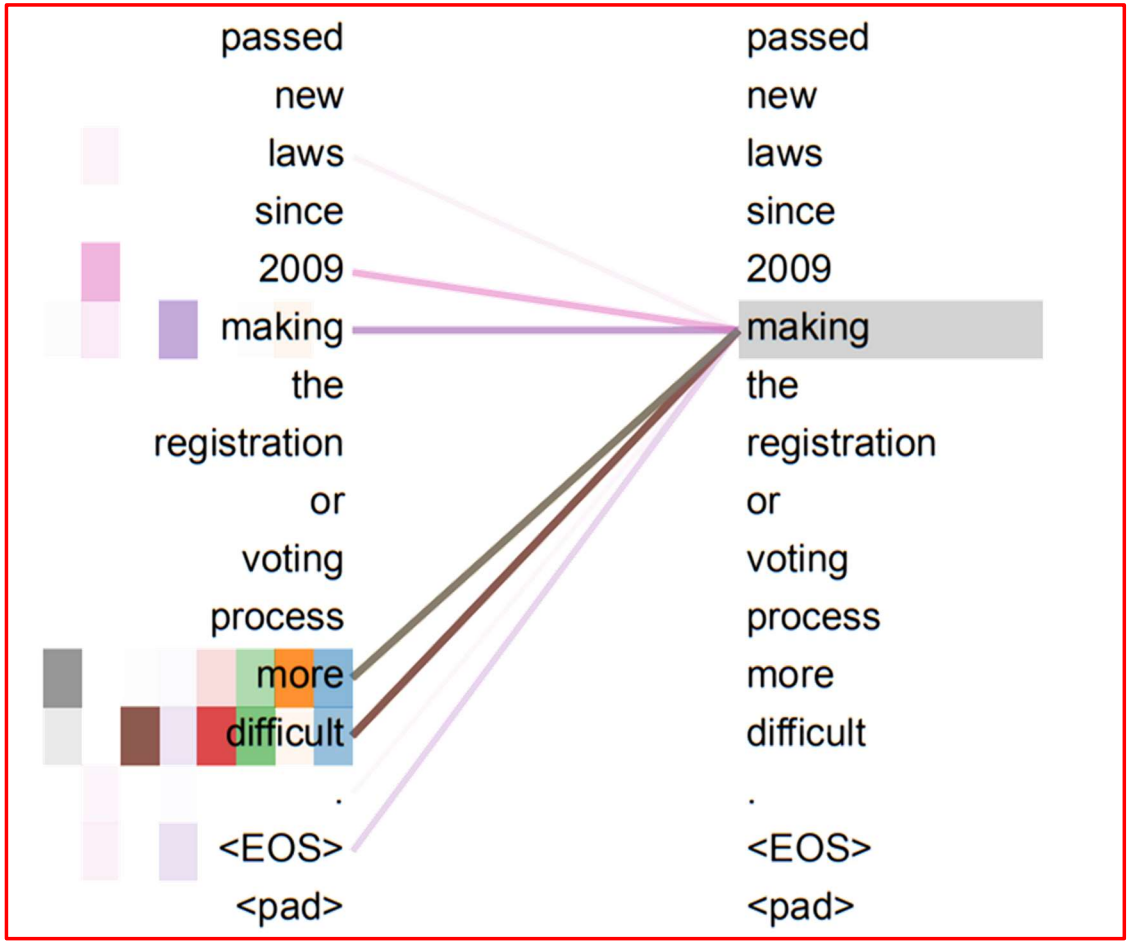
Stay calm





Multi-Head Attention

Different attention heads capture different dependencies



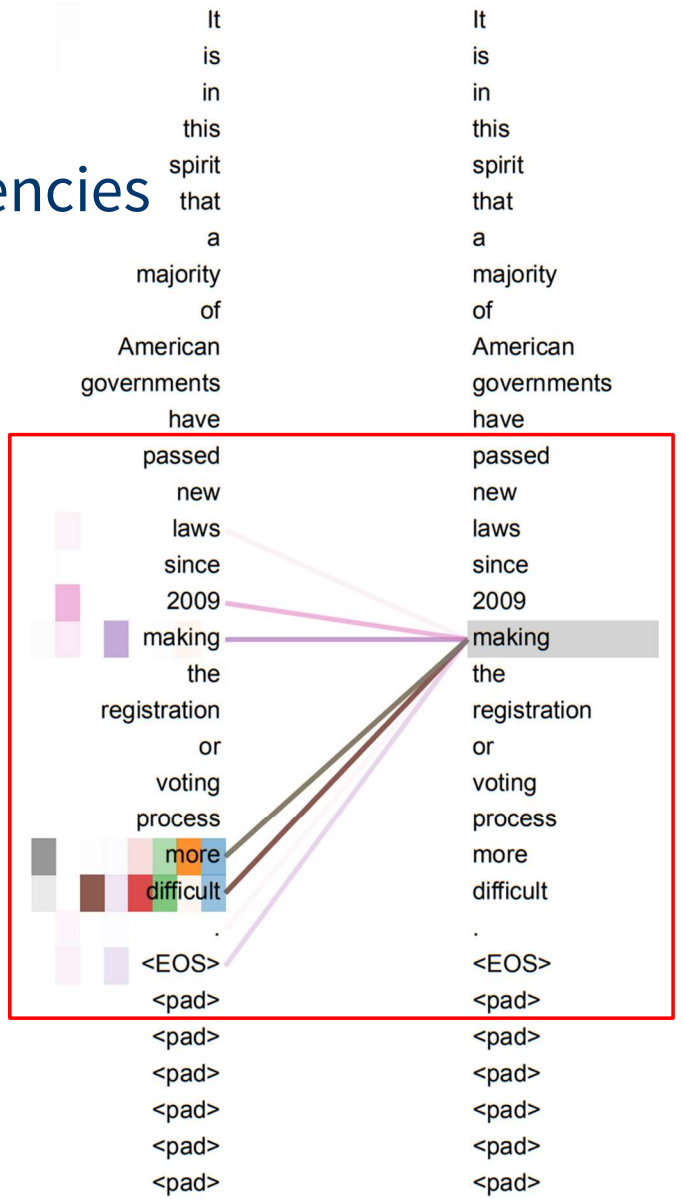
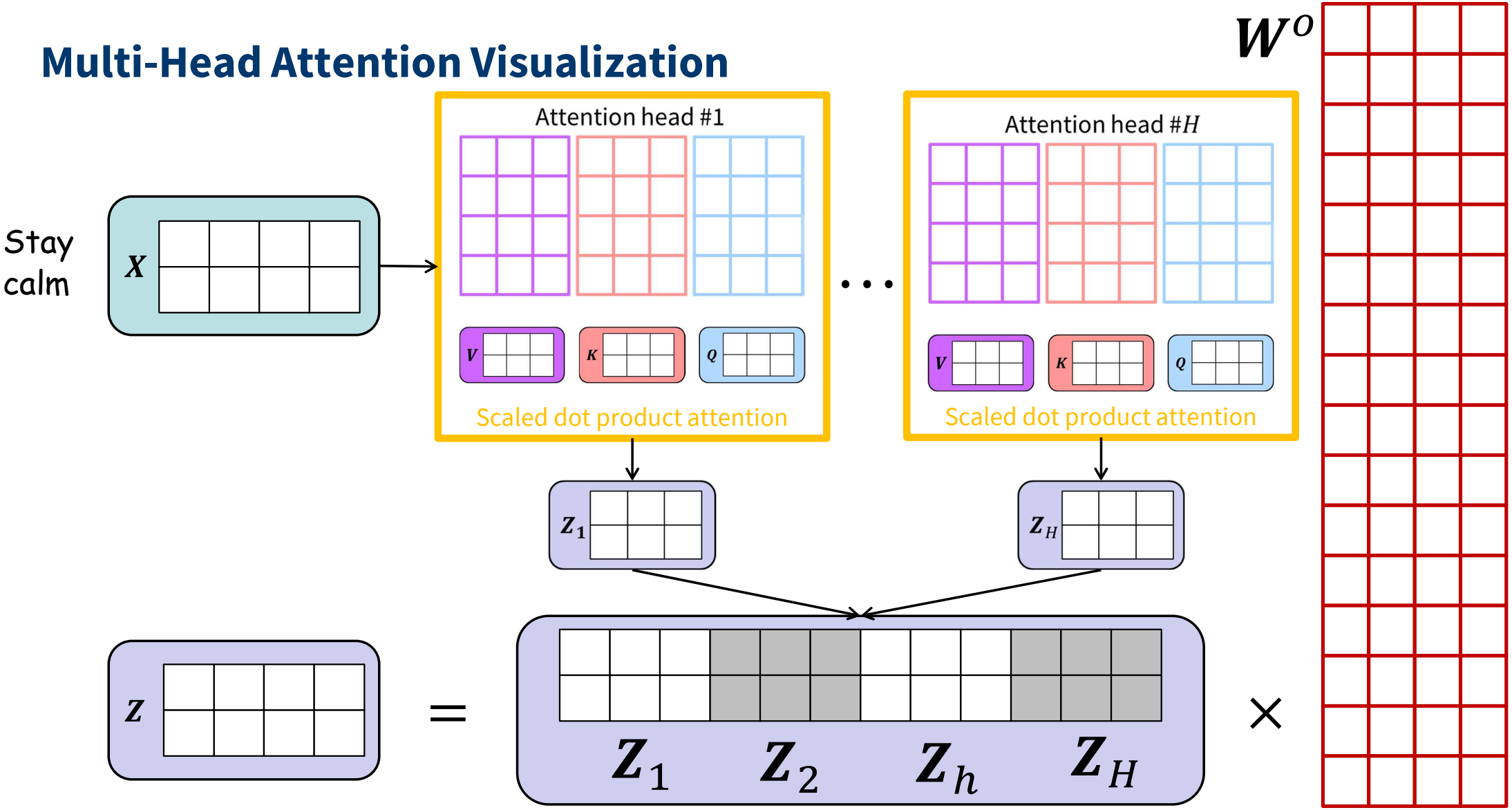


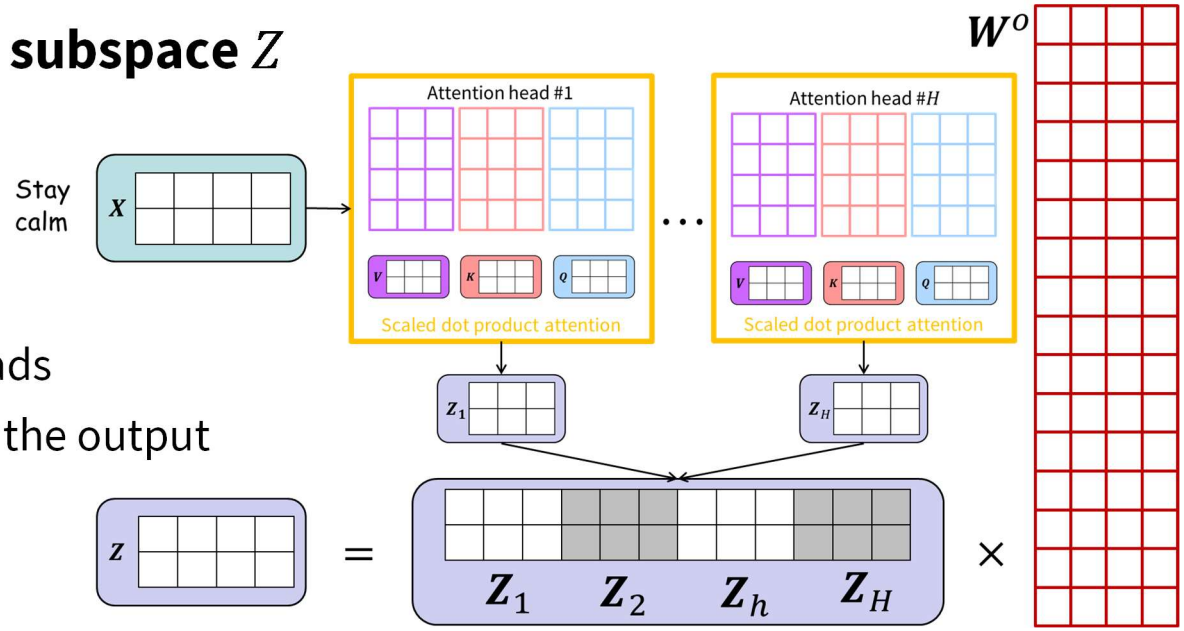
Image source: Vaswani et al. (2017)

Multi-Head Attention Visualization



Multi-Head Attention Summary

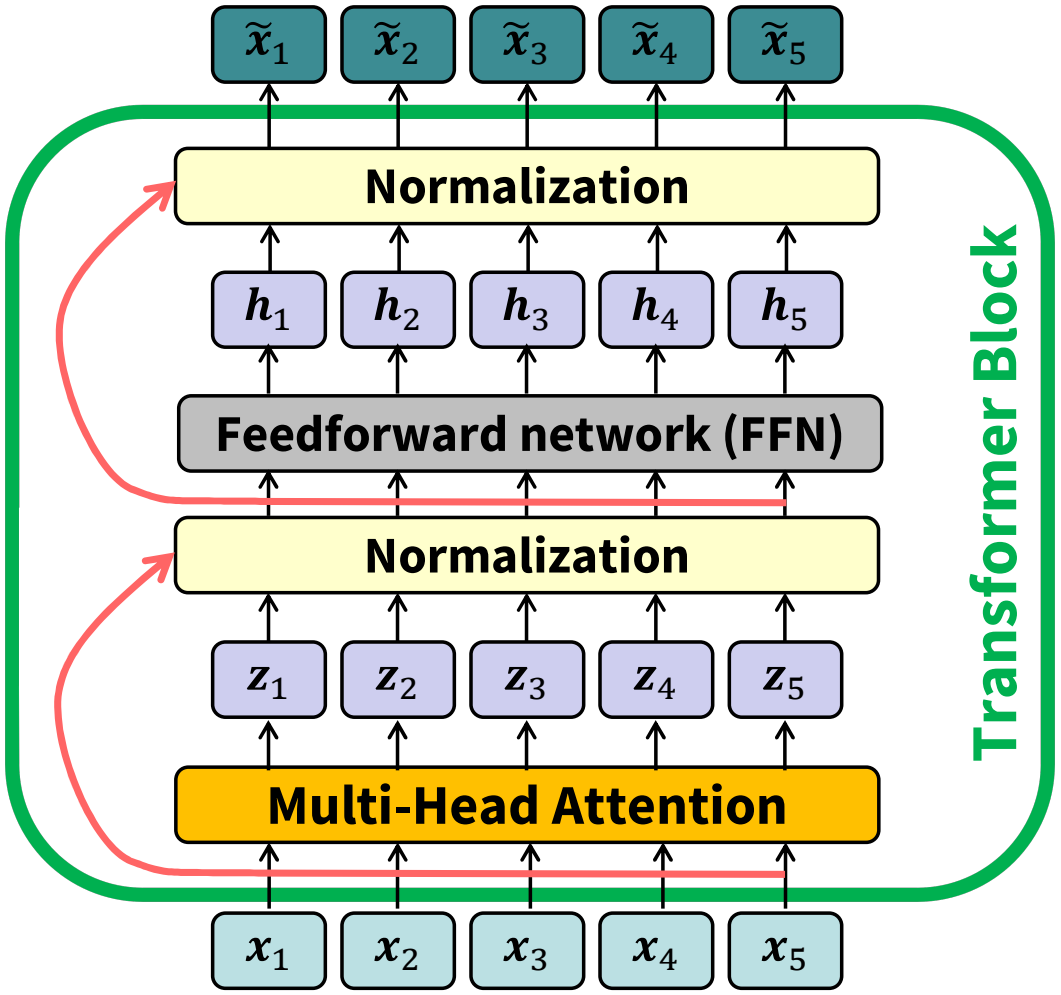
- **Each attention head maps input X to subspace Z**
- **Stack subspaces**
- **Multiply with properly shaped *output* weight matrix $\boldsymbol{W^O} \in \mathbb{R}^{\tilde{d}H \times d}$**
  - Aggregate outputs of different attention heads
  - Gives full control over the dimensionality of the output
- **Attention layer does not change the dimensionality of the input**
  - Useful to devise a deep architecture with many stacked layers
  - All calculation run in parallel w/o any sort of recurrence
- **Multi-head attention (MHA) formally defined**



$$\mathrm{MHA}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \mathrm{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{\tilde{d}}}\right)\boldsymbol{V}$$

Transformer Block Summary

- **Multi-head attention layer**
  - Capture dependencies among input token through linear projections
  - Designed to sustain dimensionality
- **Normalization step**
  - Add attention layer output and its input coming from residual connection
  - Aim is to stabilize gradients
- **Feedforward neural network**
  - Applied position wise to input sequence
    - Weight sharing across tokens
  - Adds nonlinearity through RELU activation
  - Designed to sustain dimensionality
- **Another normalization step**



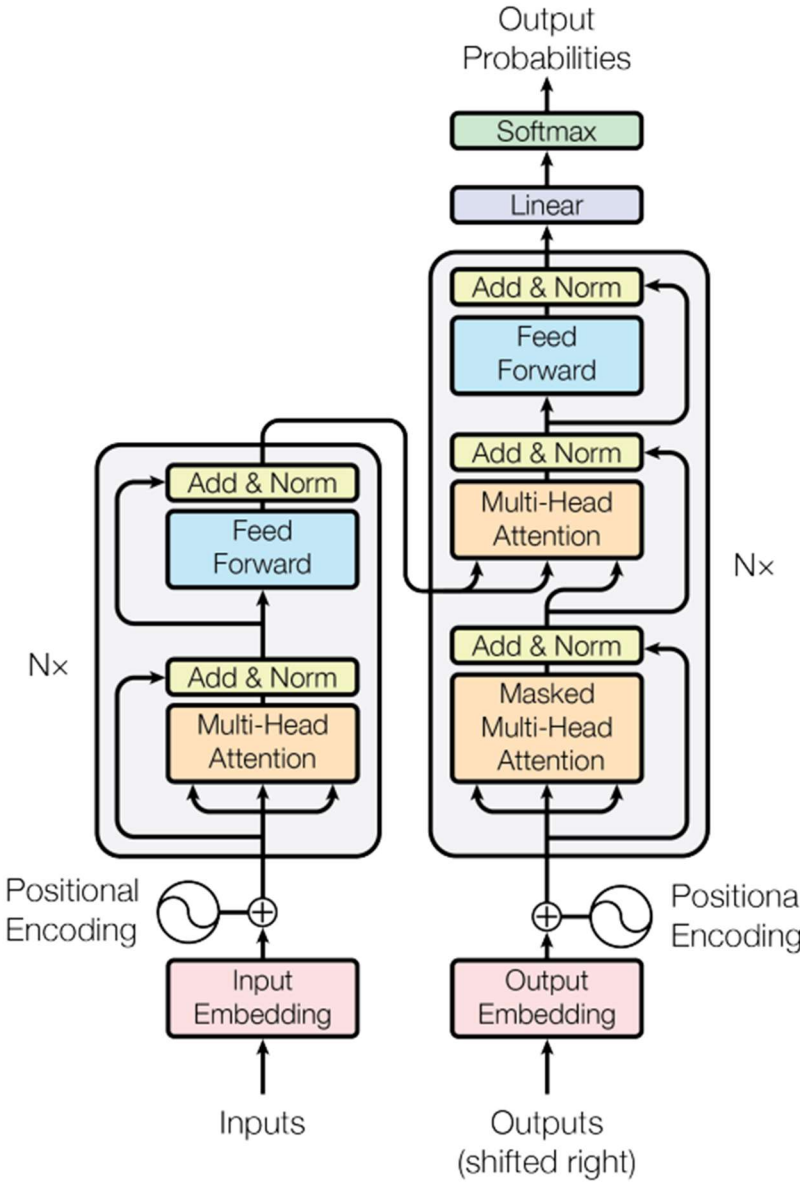
Almost There

Encoder also uses masked multi-head attention

- **Self-attention layer must not look ahead**
  - At position j, we can only attend to positions i $\leq j$
    - Recall machine translation example
    - It is unknown how the sentences continues
  - Achieved through masking
- **Masked multi-head attention (MMHA)**

$$\mathrm{MMHA}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \mathrm{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{\top}}{\sqrt{\tilde{d}}} + \boldsymbol{M}\right)\boldsymbol{V}$$

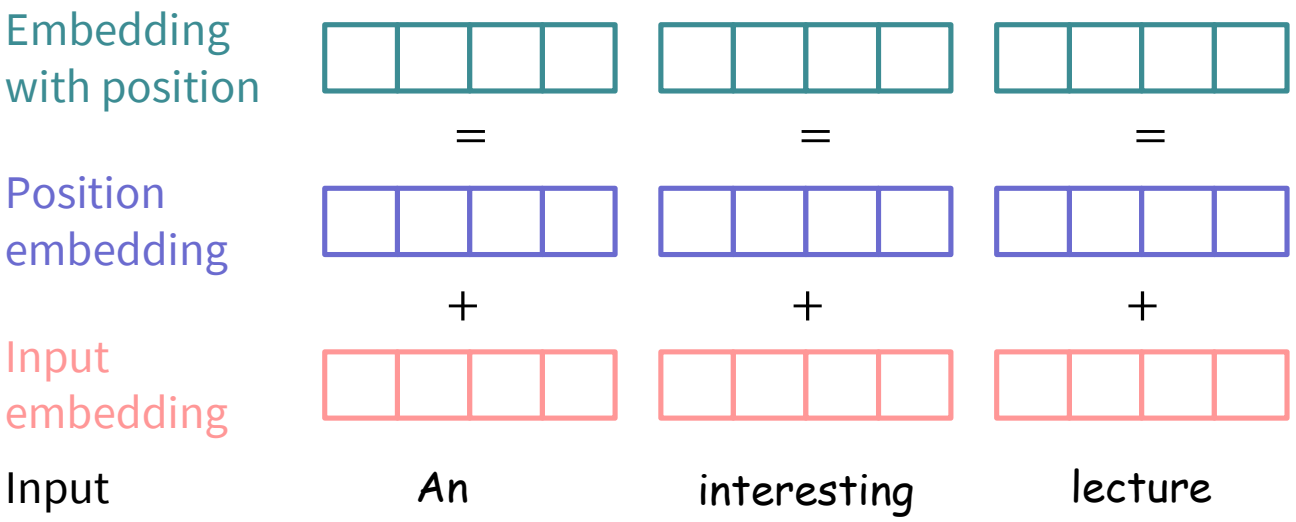
  - with $\boldsymbol{M}$ denoting a triangular matrix
  - $M_{ji} = \begin{cases} 0 & if\ j \geq i \\ -\infty & if\ j < i \end{cases}$ where j is the current position
  - Softmax assigns zero weight to positions $> j$

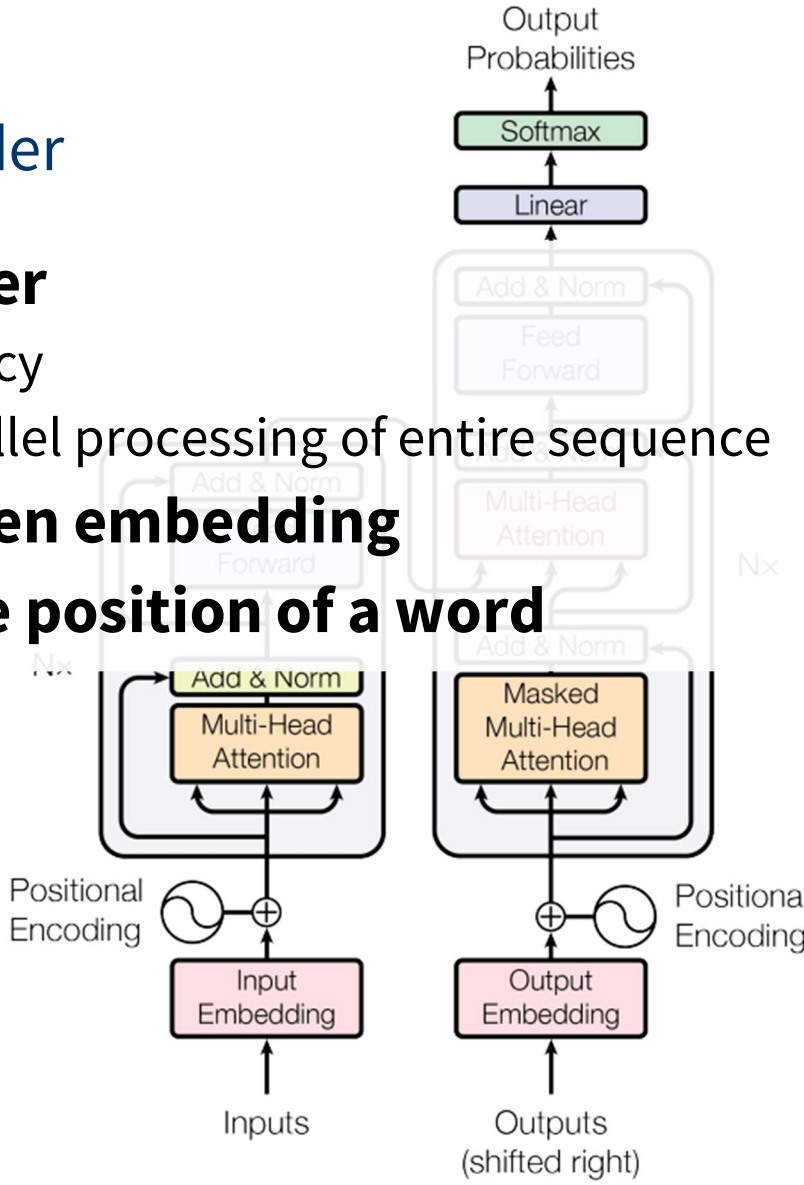


Positional Encoding (PE)

PE equip the transformer with a sense of *word* order

- **Self-attention does not account for word order**
  - Recurrent operation was dropped to increase efficiency
  - No inherent understanding of word order due to parallel processing of entire sequence
- **Add auxiliary input of same dimension to token embedding**
- **Design this input to allow the model learn the position of a word**





Positional Encoding (PE)

Obtain a unique embedding vector for each position in the input sequence

- **Different options to devise PEs (e.g., fixed or trainable)**
- **Vaswani et al. (2017) use sine and cosine functions of different frequencies**
  - $PE_{(t,2i)} = \sin\left(t/n^{\frac{2i}{d}}\right)$
  - $PE_{(t,2i+1)} = \cos(t/n^{2i/d})$

t is the position of a token in the input sequence and i indexes the PE dimension, $i = 1, \ldots, d$

n is a user-defined scaler; set to 10,000 in Vaswani et al. (2017)

Image Source: Havard NLP, The annotated transformer, https://nlp.seas.harvard.edu/2018/04/03/attention.html

A Step Toward Consolidating Modern NLP

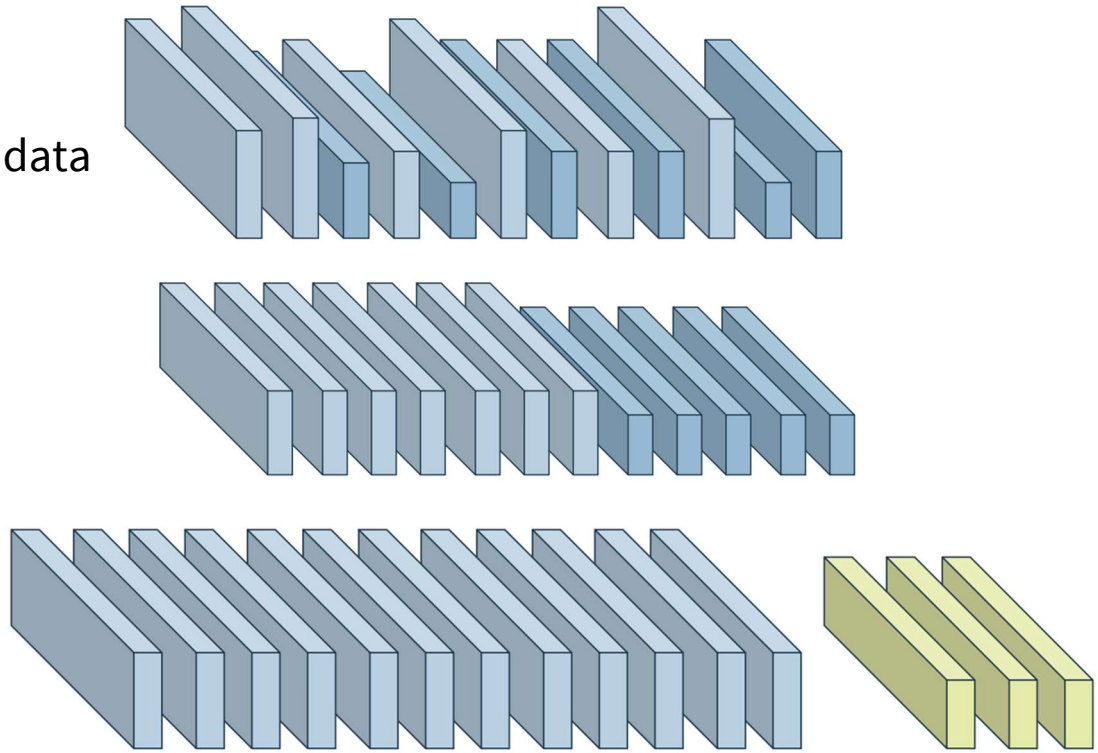
Some considerations and take-aways

- **When working on a NLP task, start from a pre-trained model**
  - The pre-training was done on corpus far larger than what we can access or process
  - Therefore, a pre-trained model possesses an advanced understanding of language
  - Even if your text data is very specific, the general understanding of language in a pre-trained model will almost certainly be valuable (exception: unsupported languages)
- **Pre-trained (transformer) models are available for many tasks**
  - Text classification, sentiment analysis, topic modeling, keyword extraction, summarization, etc.
  - How to adjust such models to your data?

A Step Toward Consolidating Modern NLP

Strategies to adjust a pre-trained model to your text

- **Start from a pre-trained model**
- **Fine-tuning**
  - Adjust parameters of all layers based on target data
  - Initialize training with pre-trained weights
- **Freezing**
  - Freeze earlier layers
  - Retrain later layers based on your data
- **Progressive learning**
  - Keep all layers unchanged
  - Add more layers (progression)



A Step Toward Consolidating Modern NLP

Is adjustment (aka fine-tuning) still state-of-art?

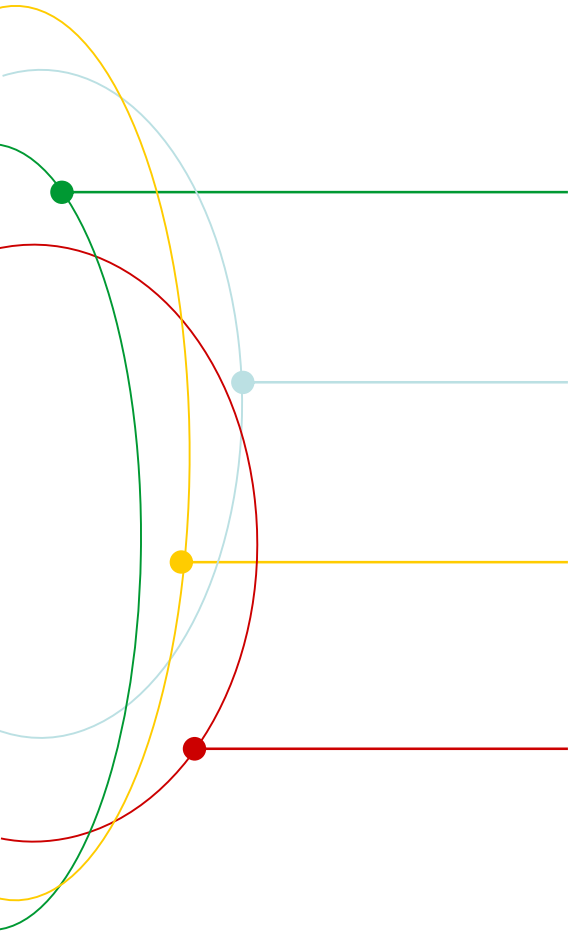
- **Zero-shot learning**
  - NLP models build for generality (e.g., BART-MNLI)
  - NLI = Natural language inference is the task of determining whether a “hypothesis” is true (entailment), false (contradiction), or undetermined (neutral) given a “premise”.
  - No fine-tuning
- **General-purpose text classifier**
  - Output is a probability
  - Financial forecasting example
    - Premise: some text (e.g., tweet)
    - Hypothesis: “This example is bullish for XYZ”
    - Different from sentiment analysis!
- **And how about prompting an LLM?**

Premise	Label	Hypothesis
A man inspects the uniform of a figure in some East Asian country.	contradiction	The man is sleeping.
An older and younger man smiling.	neutral	Two men are smiling and laughing at the cats playing on the floor.
A soccer game with multiple males playing.	entailment	Some men are playing a sport.

Source: Natural language inference | NLP-progress (nlpprogress.com)

Summary

Summary



Learning goals

- Idea and issues of RNNs
- Gated RNNs, Attention, and Transformers

Findings

- Autoregression and latent state models
- Recurrent cells maintain a memory
- Gating hidden state/memory to overcome vanishing gradient problem and capture long-term dependencies
- GRUs, LSTMs, bidirectionality, and stacked RNNs
- NLP transfer learning
- Attention to aggregate information in seq-to-seq models
- Transformer networks rely exclusively on (self-)attention
- BERT is a state-of-the-art, transformer-based text classifier

What next

- Demo of various NLP models for sentiment analysis
- Interpretable and causal machine learning

Literature

- Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *ArXiv preprint*, arXiv:1409.0473.
- Cho, K., Merrienboer, B. v., Bahdanau, D., & Bengio, Y. (2014). *On the Properties of Neural Machine Translation: Encoder-Decoder Approaches.* In D. Wu, M. Carpuat, X. Carreras & E. M. Vecchi (Eds.). Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translatio, ACL, pp. 103-111.
- Chung J, Gulcehre C, Cho K, Bengio Y (2014): Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. CoRR arXiv:1412.3555v1
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *ArXiv preprint,* arXiv:1810.04805v2.
- Dodge, J., Prewitt, T., Combes, R. T. d., Odmark, E., Schwartz, R., Strubell, E., . . . Buchanan, W. (2022). Measuring the Carbon Intensity of AI in Cloud Instances. ACM Conference on Fairness, Accountability, and Transparency, Seoul, Republic of Korea.
- Fischer T, Krauss C (2018): Deep learning with long short-term memory networks for financial market predictions. EJOR 270(2): 654-669
- Hochreiter S, Schmidhuber J (1997): Long Short-Term Memory. Neural Computation 9(8): 1735-1780
- Howard, J., & Ruder, S. (2018). Universal Language Model Fine-tuning for Text Classification. *CoRR,* arXiv:1801.06146.
- Kalchbrenner, N., & Blunsom, P. (2013). *Recurrent Continuous Translation Models*. Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, ACL, pp. 1700-1709.
- Karim, R. (2019)Attn: Illustrated Attention. https://towardsdatascience.com/attn-illustrated-attention-5ec4ad276ee3
- Linzen et al. (2016) Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies. Archive preprint. https://arxiv.org/pdf/1611.01368.pdf
- Merity, S. (2019). Single Headed Attention RNN: Stop Thinking With Your Head. *ArXiv preprint*, arXiv:1911.11423v2.
- R. Pascanu et al. (2013) On the difficulty of training recurrent neural networks. ICML. Available at http://proceedings.mlr.press/v28/pascanu13.pdf
- C. Tallec, Y. Ollivier (2017) Unbiasing Truncated Backpropagation Through. Archive preprint available at https://arxiv.org/abs/1705.08209
- Strubell, E., Ganesh, A., & McCallum, A. (2019). Energy and Policy Considerations for Deep Learning in NLP. Proceedings of the 57th Conference of the Association for Computational Linguistics (ACL'2019), Florence, Italy.
- Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. *ArXiv preprint,* arXiv:1409.3215v3.
- Vaswani et al. (2017). Attention Is All You Need. *NIPS'2017,*pp. 6000-6010

Thank you for your attention!

Stefan Lessmann

Chair of Information Systems
School of Business and Economics
Humboldt-University of Berlin, Germany

Tel. +49.30.2093.5742
Fax. +49.30.2093.5741

stefan.lessmann@hu-berlin.de
http://bit.ly/hu-wi

www.hu-berlin.de

Photo: Heike Zappe

Gated RNNs for the rescue

Gated recurrent units (GRU) and long short-term memory (LSTM)

Gated Recurrent Units (GRU)¶

Introduce two gates for updating and resetting the hidden state

- **Design gates as neural networks**

$$\boldsymbol{U}_t = g(\boldsymbol{X}_t\boldsymbol{W}_{xu} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{hu} + \boldsymbol{b}_u)$$

$$\boldsymbol{R}_t = g(\boldsymbol{X}_t\boldsymbol{W}_{xr} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{hr} + \boldsymbol{b}_r)$$

- **GRU first generate a new candidate hidden state $\widehat{\boldsymbol{H}}_t$**

$$\widetilde{\boldsymbol{H}}_t = tanH(\boldsymbol{X}_t\boldsymbol{W}_{xh} + (\boldsymbol{R}_t \odot \boldsymbol{H}_{t-1})\boldsymbol{W}_{hh} + \boldsymbol{b}_h)$$

- **The candidate hidden state governs updates of the actual hidden state**

$$\boldsymbol{H}_t = \boldsymbol{U}_t \odot \boldsymbol{H}_{t-1} + (\boldsymbol{1} - \boldsymbol{U}_t) \odot \widetilde{\boldsymbol{H}}_t$$

$\boldsymbol{X}_t \in \mathbb{R}^{n\times m}$

$\boldsymbol{W}_{xh} \in \mathbb{R}^{m\times h}$

$\boldsymbol{b}_h \in \mathbb{R}^{1\times h}$

$\boldsymbol{H}_t \in \mathbb{R}^{n\times h}$

$\boldsymbol{W}_{hh} \in \mathbb{R}^{h\times h}$

$\boldsymbol{W}_{hy} \in \mathbb{R}^{h\times c}$

$\boldsymbol{b}_y \in \mathbb{R}^{1\times c}$

$\boldsymbol{U}_t \in \mathbb{R}^{n\times h}$

$\boldsymbol{b}_u \in \mathbb{R}^{1\times h}$

$\boldsymbol{W}_{xu} \in \mathbb{R}^{m\times h}$

$\boldsymbol{W}_{hu} \in \mathbb{R}^{h\times h}$

$\boldsymbol{R}_t \in \mathbb{R}^{n\times h}$

$\boldsymbol{b}_r \in \mathbb{R}^{1\times h}$

$\boldsymbol{W}_{xr} \in \mathbb{R}^{m\times h}$

$\boldsymbol{W}_{hr} \in \mathbb{R}^{h\times h}$

Gated Recurrent Units (GRU)¶

Chung et al. (2014)

- **Gating the hidden state**
  - Dedicated mechanism to decide when to update or reset the hidden state
  - Learn this mechanism as part of training
- **What actually is meant by "gating"?**

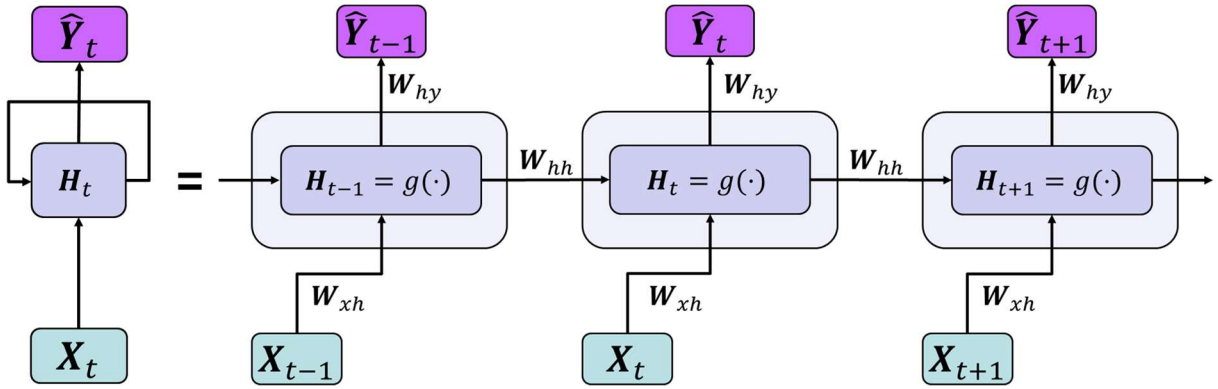
Gated Recurrent Units (GRU)¶

RNNs update their hidden state in every step

- **Gating the hidden state**
  - Dedicated mechanism to decide when to update or reset the hidden state
  - Learn this mechanism as part of training
- **What actually is meant by "gating"?**

$$\widehat{\boldsymbol{Y}}_t = \boldsymbol{H}_t \boldsymbol{W}_{hy} + \boldsymbol{b}_y$$

$$\boldsymbol{H}_t = g(\boldsymbol{X}_t \boldsymbol{W}_{xh} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hh} + \boldsymbol{b}_h)$$



$\boldsymbol{X}_t \in \mathbb{R}^{n \times m}$
$\boldsymbol{W}_{xh} \in \mathbb{R}^{m \times h}$
$\boldsymbol{b}_h \in \mathbb{R}^{1 \times h}$
$\boldsymbol{H}_t \in \mathbb{R}^{n \times h}$
$\boldsymbol{W}_{hh} \in \mathbb{R}^{h \times h}$
$\boldsymbol{W}_{hy} \in \mathbb{R}^{h \times c}$
$\boldsymbol{b}_y \in \mathbb{R}^{1 \times c}$

Gated Recurrent Units (GRU)¶

A dedicated matrix or gate could moderate these updates

- **Gating the hidden state**
  - Dedicated mechanism to decide when to update or reset the hidden state
  - Learn this mechanism as part of training
- **What actually is meant by "gating"?**

$$\widehat{\boldsymbol{Y}}_t = \boldsymbol{H}_t \boldsymbol{W}_{hy} + \boldsymbol{b}_y$$

$$\boldsymbol{H}_t = g(\boldsymbol{X}_t \boldsymbol{W}_{xh} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hh} + \boldsymbol{b}_h)$$

$$\boldsymbol{H}_t = \begin{bmatrix} g(z_{11}) & \cdots & g(z_{1h}) \\ \vdots & \ddots & \vdots \\ g(z_{1n}) & \cdots & g(z_{nh}) \end{bmatrix} \odot \begin{bmatrix} u_{11} & \cdots & u_{1h} \\ \vdots & \ddots & \vdots \\ u_{1n} & \cdots & u_{nh} \end{bmatrix} = \boldsymbol{U}_t$$

$\boldsymbol{X}_t \in \mathbb{R}^{n \times m}$
$\boldsymbol{W}_{xh} \in \mathbb{R}^{m \times h}$
$\boldsymbol{b}_h \in \mathbb{R}^{1 \times h}$
$\boldsymbol{H}_t \in \mathbb{R}^{n \times h}$
$\boldsymbol{W}_{hh} \in \mathbb{R}^{h \times h}$
$\boldsymbol{W}_{hy} \in \mathbb{R}^{h \times c}$
$\boldsymbol{b}_y \in \mathbb{R}^{1 \times c}$
$\boldsymbol{U}_t \in \mathbb{R}^{n \times h}$

Gated Recurrent Units (GRU)¶

A dedicated matrix or gate could moderate these updates

- **Gating the hidden state**
  - Dedicated mechanism to decide when to update or reset the hidden state
  - Learn this mechanism as part of training
- **What actually is meant by "gating"?**

$$\widehat{\boldsymbol{Y}}_t = \boldsymbol{H}_t \boldsymbol{W}_{hy} + \boldsymbol{b}_y$$

$$\boldsymbol{H}_t = g(\boldsymbol{X}_t \boldsymbol{W}_{xh} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hh} + \boldsymbol{b}_h)$$

$$\boldsymbol{H}_t = \begin{bmatrix} g(z_{11}) & \cdots & g(z_{1h}) \\ \vdots & \ddots & \vdots \\ g(z_{1n}) & \cdots & g(z_{nh}) \end{bmatrix} \odot \begin{bmatrix} \sigma(u_{11}) & \cdots & \sigma(u_{1h}) \\ \vdots & \ddots & \vdots \\ \sigma(u_{1n}) & \cdots & \sigma(u_{nh}) \end{bmatrix} = \boldsymbol{U}_t$$

$\boldsymbol{X}_t \in \mathbb{R}^{n \times m}$
$\boldsymbol{W}_{xh} \in \mathbb{R}^{m \times h}$
$\boldsymbol{b}_h \in \mathbb{R}^{1 \times h}$
$\boldsymbol{H}_t \in \mathbb{R}^{n \times h}$
$\boldsymbol{W}_{hh} \in \mathbb{R}^{h \times h}$
$\boldsymbol{W}_{hy} \in \mathbb{R}^{h \times c}$
$\boldsymbol{b}_y \in \mathbb{R}^{1 \times c}$
$\boldsymbol{U}_t \in \mathbb{R}^{n \times h}$

Gated Recurrent Units (GRU)¶

Design ideas

- **Gating the hidden state**
  - Dedicated mechanism to decide when to update or reset the hidden state
  - Learn this mechanism as part of training
- **What actually is meant by "gating"?**
  - Introduce a matrix of the same dimension as the hidden state
  - Ensure elements of this matrix are between zero and one
  - Perform element-wise multiplication
- **GRUs introduce two gates**
  - Update gate → governs which information is added to the hidden state
  - Reset gate → governs which information is removed from the hidden state
- **Hidden state represents the memory of the network (same as RNN)**

Gated Recurrent Units (GRU)¶

How to learn the gating mechanism?

- **A gate is a matrix**
- **Learning a gate equates to learning the elements of a matrix**
- **Similar to learning a weight matrix**
- **Design gates as neural network**

$$\boldsymbol{U}_t = g(\boldsymbol{X}_t\boldsymbol{W}_{xu} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{hu} + \boldsymbol{b}_u)$$

$$\boldsymbol{R}_t = g(\boldsymbol{X}_t\boldsymbol{W}_{xr} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{hr} + \boldsymbol{b}_r)$$

$\boldsymbol{X}_t \in \mathbb{R}^{n\times m}$

$\boldsymbol{W}_{xh} \in \mathbb{R}^{m\times h}$

$\boldsymbol{b}_h \in \mathbb{R}^{1\times h}$

$\boldsymbol{H}_t \in \mathbb{R}^{n\times h}$

$\boldsymbol{W}_{hh} \in \mathbb{R}^{h\times h}$

$\boldsymbol{W}_{hy} \in \mathbb{R}^{h\times c}$

$\boldsymbol{b}_y \in \mathbb{R}^{1\times c}$

$\boldsymbol{U}_t \in \mathbb{R}^{n\times h}$

$\boldsymbol{b}_u \in \mathbb{R}^{1\times h}$

$\boldsymbol{W}_{xu} \in \mathbb{R}^{m\times h}$

$\boldsymbol{W}_{hu} \in \mathbb{R}^{h\times h}$

$\boldsymbol{R}_t \in \mathbb{R}^{n\times h}$

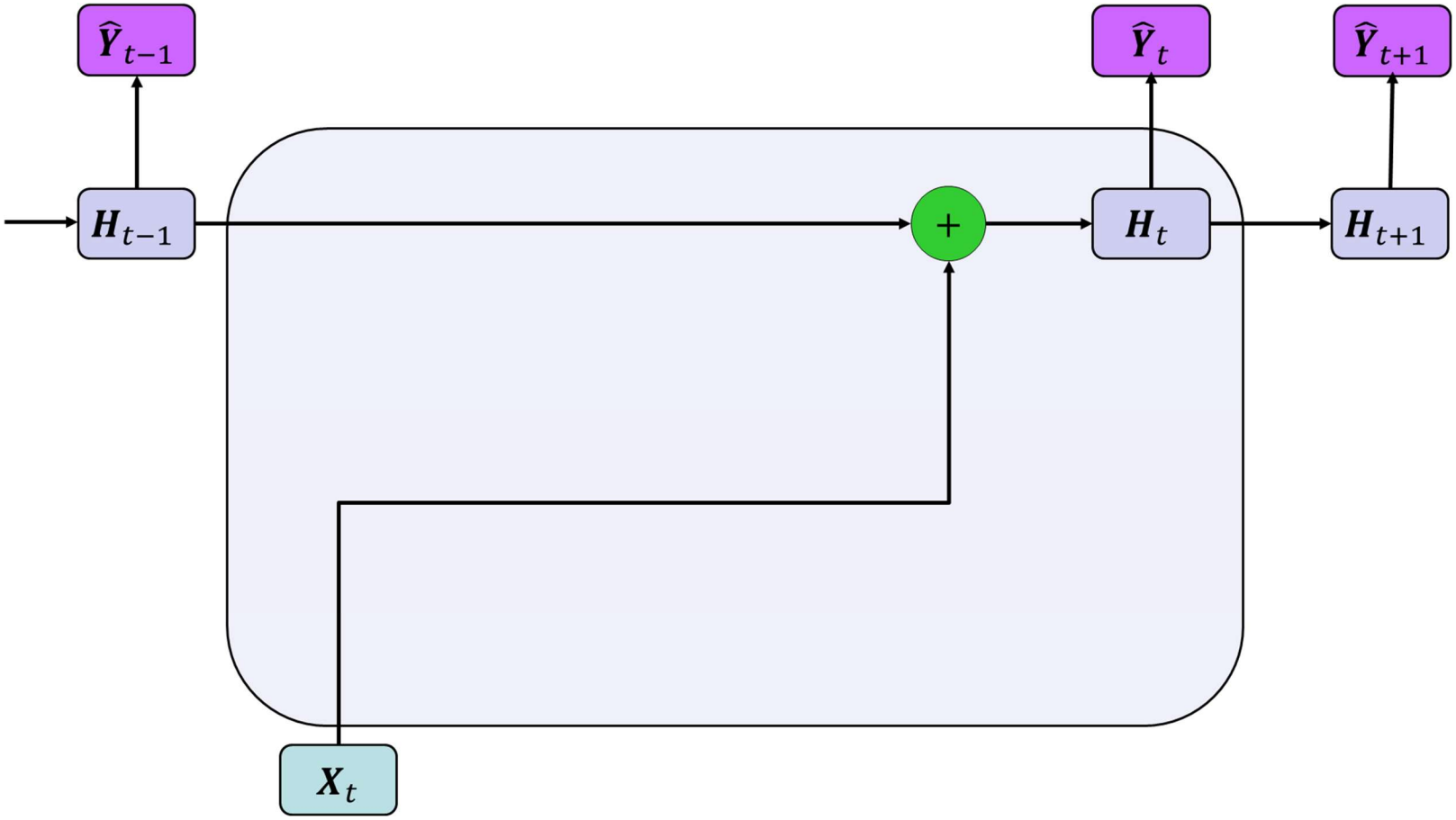
$\boldsymbol{b}_r \in \mathbb{R}^{1\times h}$

$\boldsymbol{W}_{xr} \in \mathbb{R}^{m\times h}$

$\boldsymbol{W}_{hr} \in \mathbb{R}^{h\times h}$

Gated Recurrent Units (GRU)¶

Graphical representation of RNN

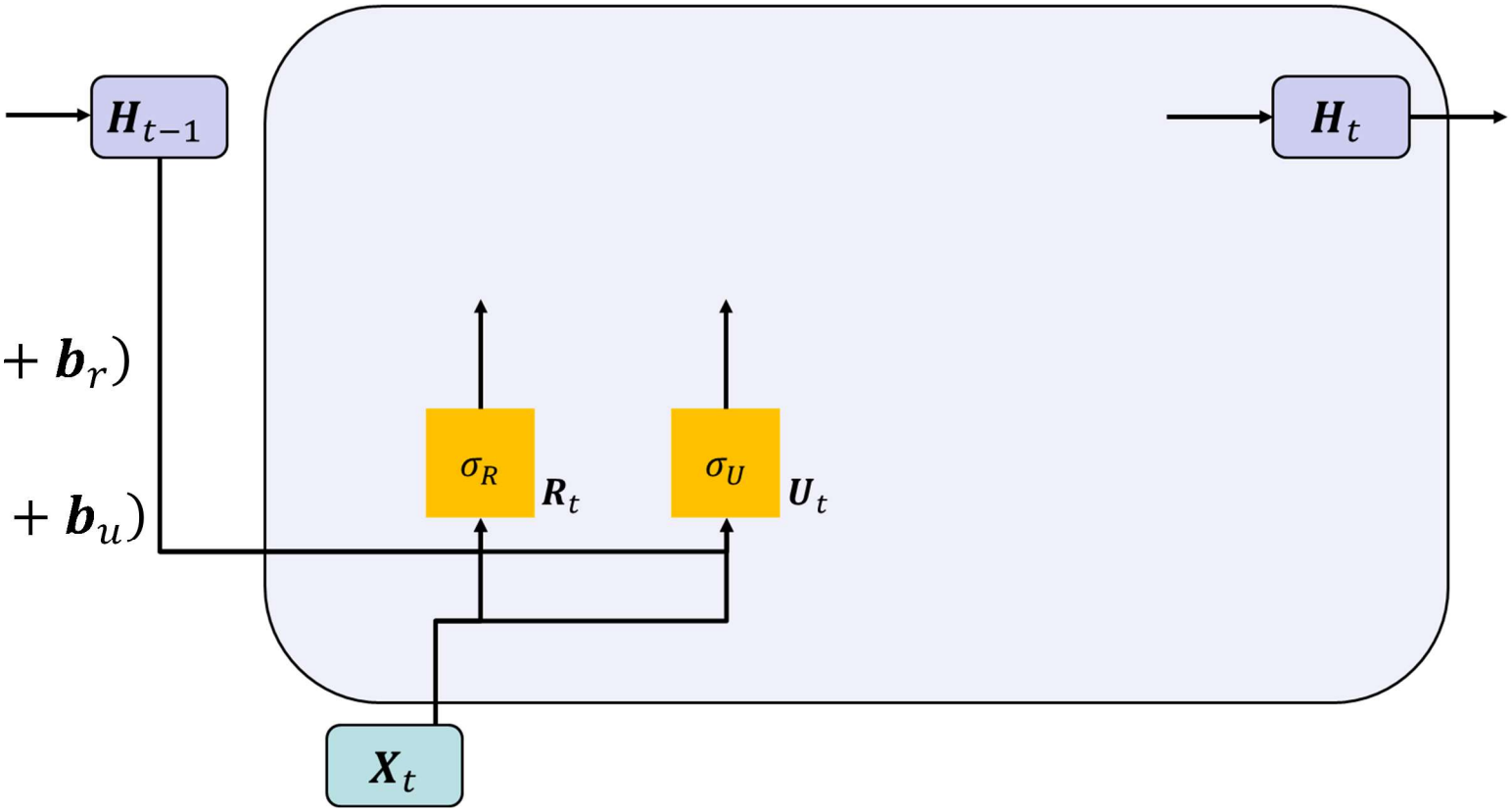


Gated Recurrent Units (GRU)¶

Graphical representation of RNN with reset and update gate

$$\boldsymbol{R}_t = \sigma(\boldsymbol{X}_t\boldsymbol{W}_{xr} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{hr} + \boldsymbol{b}_r)$$

$$\boldsymbol{U}_t = \sigma(\boldsymbol{X}_t\boldsymbol{W}_{xu} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{hu} + \boldsymbol{b}_u)$$



Introducing a Reset Gate Into the GRU Cell

- **Hidden state equation in RNN (note that $g(\cdot)$ is typically tanH)**

$$\boldsymbol{H}_t = g(\boldsymbol{X}_t\boldsymbol{W}_{xh} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{hh} + \boldsymbol{b}_h)$$

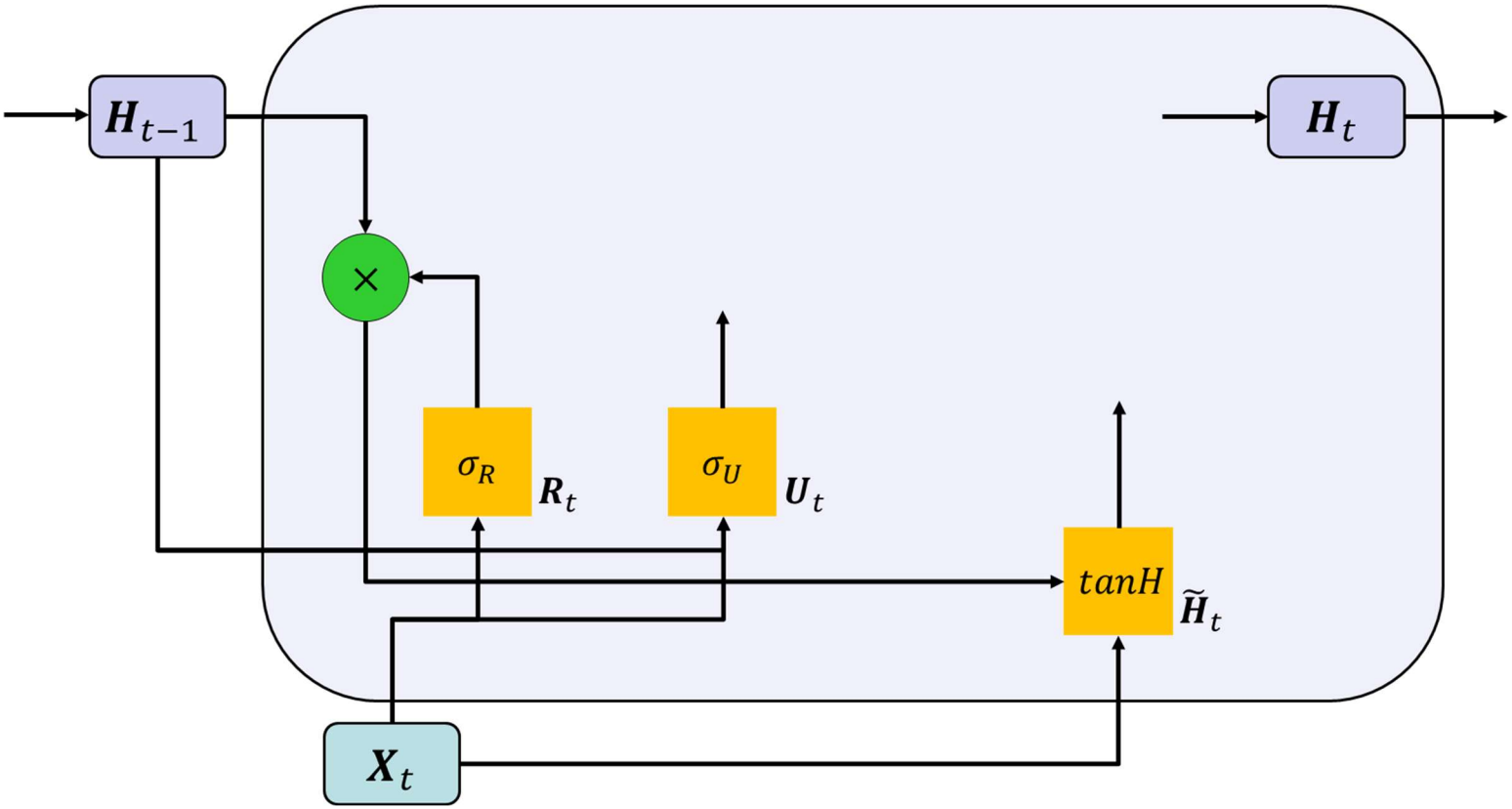
- **GRU first generate a new candidate hidden state $\widehat{\boldsymbol{H}}_t$**

$$\widetilde{\boldsymbol{H}}_t = tanH(\boldsymbol{X}_t\boldsymbol{W}_{xh} + (\boldsymbol{R}_t \odot \boldsymbol{H}_{t-1})\boldsymbol{W}_{hh} + \boldsymbol{b}_h)$$

  - Elements of $\boldsymbol{R}_t$close to one:
  - Elements of $\boldsymbol{R}_t$close to zero:

Introducing a Reset Gate Into the GRU Cell

$$\widetilde{\boldsymbol{H}}_t = tanH(\boldsymbol{X}_t\boldsymbol{W}_{xh} + (\boldsymbol{R}_t \odot \boldsymbol{H}_{t-1})\boldsymbol{W}_{hh} + \boldsymbol{b}_h)$$



GRU Hidden State Update

- GRU first generate a new candidate hidden state $\widehat{H}_t$

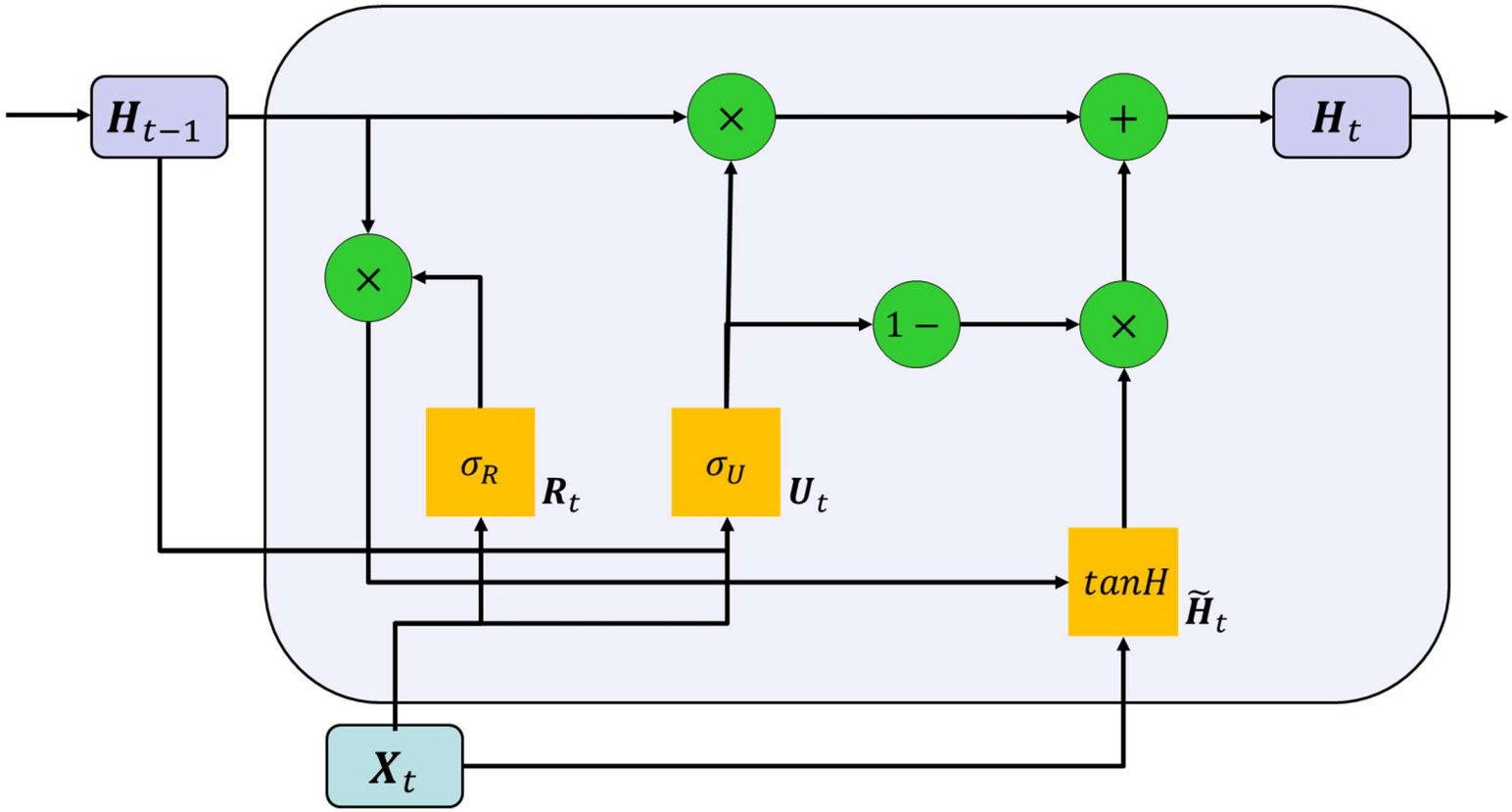
$$\widetilde{\boldsymbol{H}}_t = tanH(\boldsymbol{X}_t\boldsymbol{W}_{xh} + (\boldsymbol{R}_t \odot \boldsymbol{H}_{t-1})\boldsymbol{W}_{hh} + \boldsymbol{b}_h)$$

- The candidate hidden state is then used to update the actual hidden state

$$\boldsymbol{H}_t = \boldsymbol{U}_t \odot \boldsymbol{H}_{t-1} + (\boldsymbol{1} - \boldsymbol{U}_t) \odot \widetilde{\boldsymbol{H}}_t$$

Full GRU Cell With Reset and Update Gate

$$\boldsymbol{H}_t = \boldsymbol{U}_t \odot \boldsymbol{H}_{t-1} + (\mathbf{1} - \boldsymbol{U}_t) \odot \tilde{\boldsymbol{H}}_t$$



Chung J, Gulcehre C, Cho K, Bengio Y (2014): Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. CoRR arXiv:1412.3555v1

Formal Summary of the GRU

- **Reset Gate** $\boldsymbol{R}_t = \sigma(\boldsymbol{X}_t\boldsymbol{W}_{xr} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{hr} + \boldsymbol{b}_r)$
- **Update Gate** $\boldsymbol{U}_t = \sigma(\boldsymbol{X}_t\boldsymbol{W}_{xu} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{hu} + \boldsymbol{b}_u)$
- **Candidate hidden state**

$$\widetilde{\boldsymbol{H}}_t = tanH(\boldsymbol{X}_t\boldsymbol{W}_{xh} + (\boldsymbol{R}_t \odot \boldsymbol{H}_{t-1})\boldsymbol{W}_{hh} + \boldsymbol{b}_h)$$

- **Hidden state update**

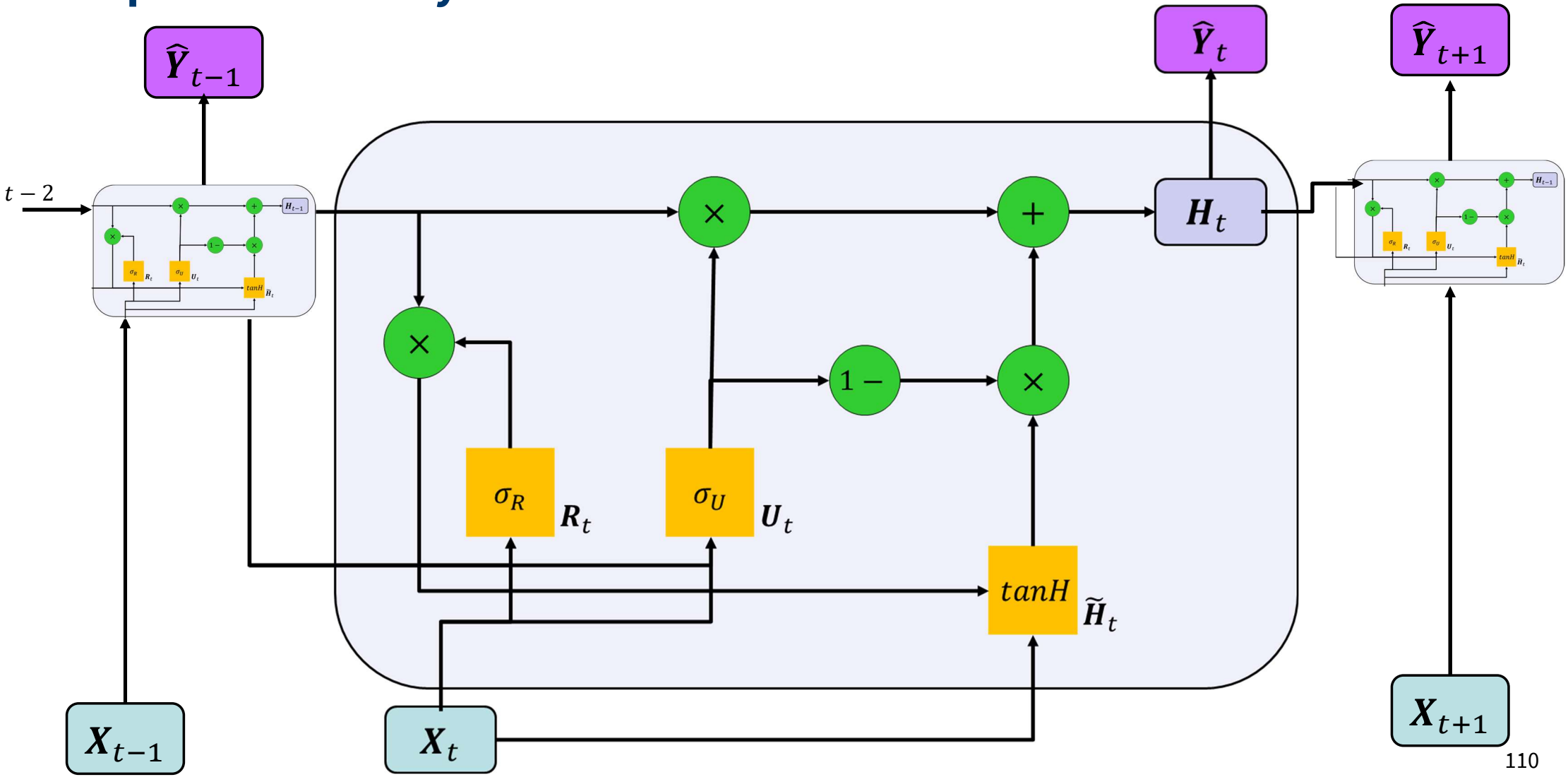
$$\boldsymbol{H}_t = \boldsymbol{U}_t \odot \boldsymbol{H}_{t-1} + (\boldsymbol{1} - \boldsymbol{U}_t) \odot \widetilde{\boldsymbol{H}}_t$$

- **Network output**

$$\widehat{\boldsymbol{Y}}_t = \boldsymbol{H}_t\boldsymbol{W}_{hy} + \boldsymbol{b}_y$$

$\boldsymbol{X}_t \in \mathbb{R}^{n\times m}$
$\boldsymbol{W}_{xh} \in \mathbb{R}^{m\times h}$
$\boldsymbol{b}_h \in \mathbb{R}^{1\times h}$
$\boldsymbol{H}_t \in \mathbb{R}^{n\times h}$
$\boldsymbol{W}_{hh} \in \mathbb{R}^{h\times h}$
$\boldsymbol{W}_{hy} \in \mathbb{R}^{h\times c}$
$\boldsymbol{b}_y \in \mathbb{R}^{1\times c}$
$\boldsymbol{U}_t \in \mathbb{R}^{n\times h}$
$\boldsymbol{b}_u \in \mathbb{R}^{1\times h}$
$\boldsymbol{W}_{xu} \in \mathbb{R}^{m\times h}$
$\boldsymbol{W}_{hu} \in \mathbb{R}^{h\times h}$
$\boldsymbol{R}_t \in \mathbb{R}^{n\times h}$
$\boldsymbol{b}_r \in \mathbb{R}^{1\times h}$
$\boldsymbol{W}_{xr} \in \mathbb{R}^{m\times h}$
$\boldsymbol{W}_{hr} \in \mathbb{R}^{h\times h}$

Graphical Summary of the GRU



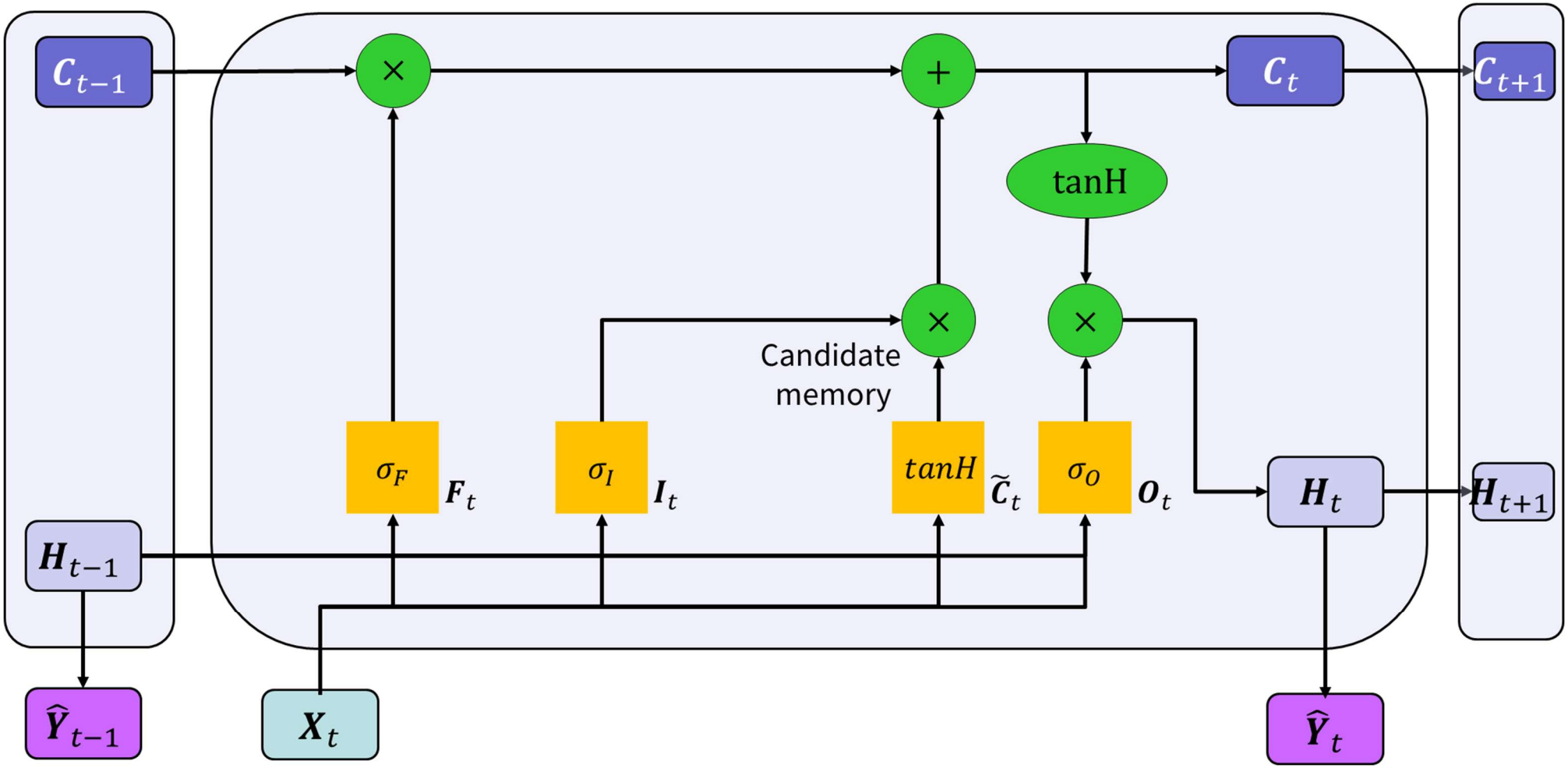
Long Short-Term Memory
Hochreiter, Schmidhuber (1997)

- **Related approach to address long-term information preservation and short-term input skipping in latent variable models and solve the vanishing gradient problem**
- **Introduce an additional cell memory, which stores long-term information**
- **LSTM can erase, read, and write the memory via three gates**
  - Input gate
  - Output gate
  - Forget gate

  } All feedforward neural networks as with GRUs
- **Gates are dynamic in that their value is computed based on the current context, t**
  - Gates is closed → all matrix elements close to zero
  - Gates is open → all matrix elements close to one
  - Somewhere in between

Long Short-Term Memory

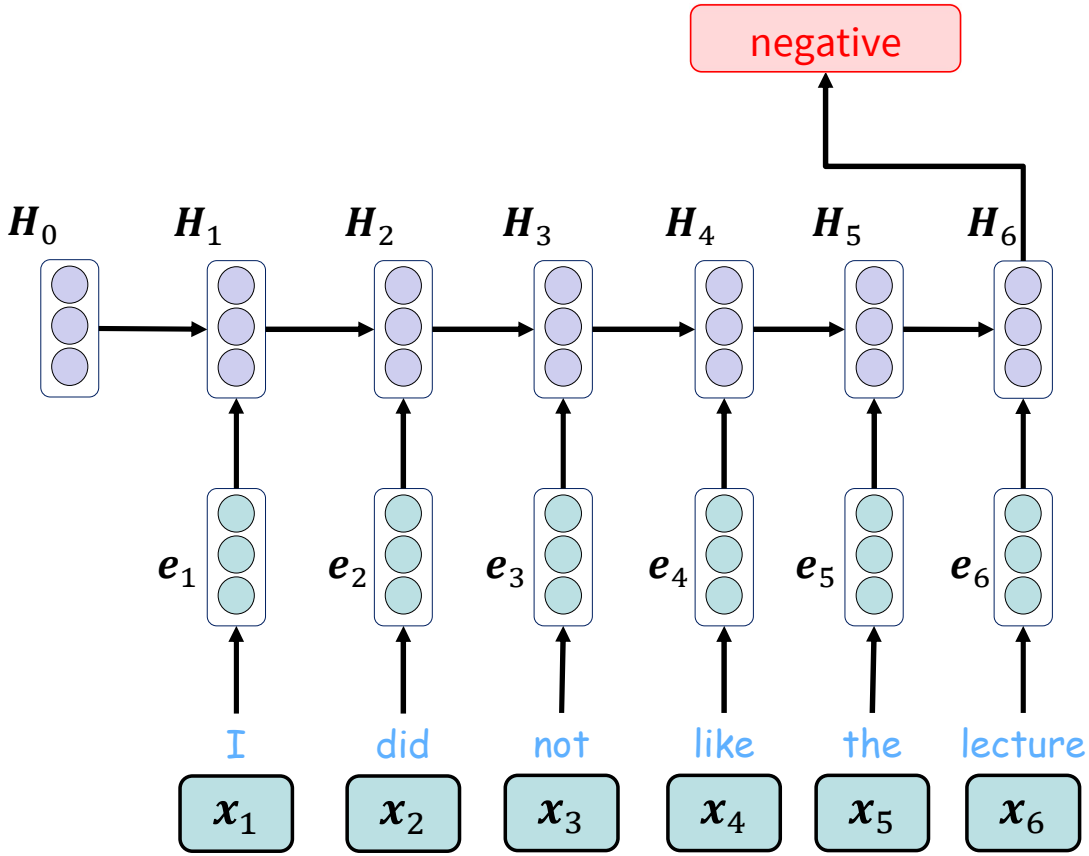
Visual representation of LSTM cell



Long Short-Term Memory

Formal representation of LSTM cell

- **Forget gate** $\boldsymbol{F}_t = \sigma\left(\boldsymbol{X}_t\boldsymbol{W}_{xf} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{hf} + \boldsymbol{b}_f\right)$ Controls what is kept vs. forgotten from previous memory state
- **Input gate** $\boldsymbol{I}_t = \sigma(\boldsymbol{X}_t\boldsymbol{W}_{xi} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{hi} + \boldsymbol{b}_i)$ Controls what part of candidate memory is written to memory
- **Output gate** $\boldsymbol{O}_t = \sigma(\boldsymbol{X}_t\boldsymbol{W}_{xo} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{ho} + \boldsymbol{b}_o)$ Controls what part of cell memory enters hidden state
- **New candidate memory** $\widetilde{\boldsymbol{C}}_t = tanH(\boldsymbol{X}_t\boldsymbol{W}_{xc} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{hc} + \boldsymbol{b}_c)$
- **Memory update** $\boldsymbol{C}_t = \boldsymbol{F}_t \odot \boldsymbol{C}_{t-1} + \boldsymbol{I}_t \odot \widetilde{\boldsymbol{C}}_t$ Erase some part of memory and write some new content
- **Hidden state** $\boldsymbol{H}_t = \boldsymbol{O}_t \odot tanH(\boldsymbol{C}_t)$ Read some part from memory


negative
H_0
H_1
H_2
H_3
H_4
H_5
H_6
e_1
e_2
e_3
e_4
e_5
e_6
I
did
not
like
the
lecture
x_1
x_2
x_3
x_4
x_5
x_6