

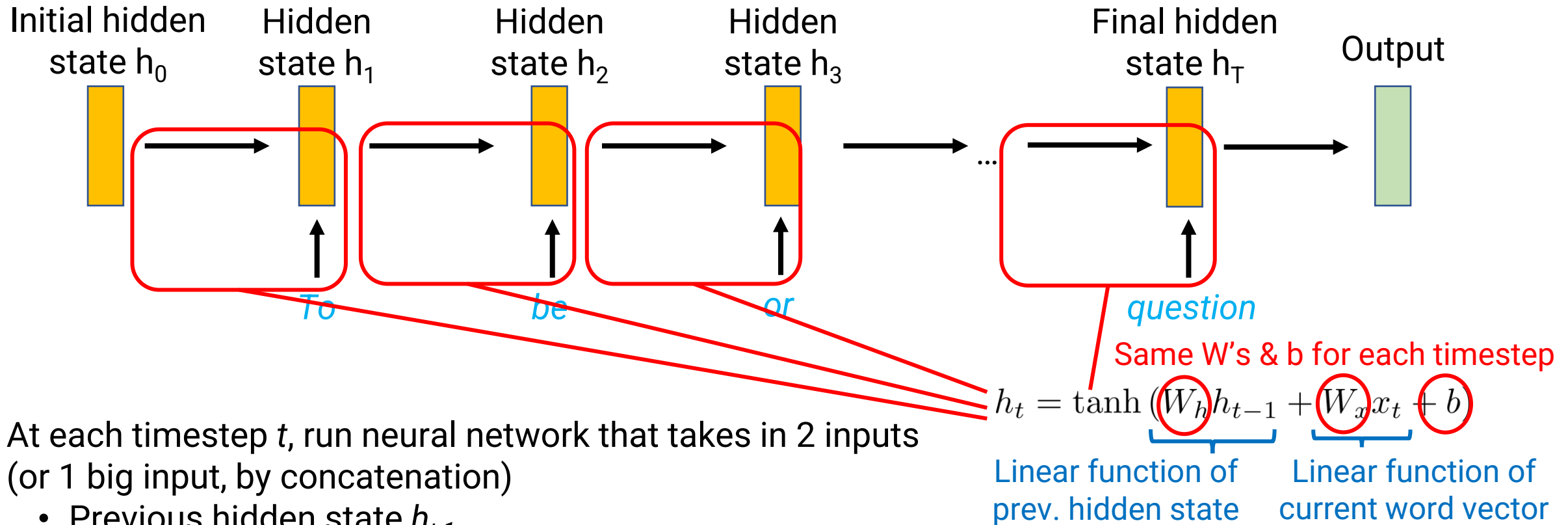
Deep Learning for Language: GRUs/LSTMs, Attention

Robin Jia

USC CSCI 467, Spring 2025

March 6, 2025

Review: “Vanilla”/“Elman” RNN



- At each timestep t , run neural network that takes in 2 inputs (or 1 big input, by concatenation)
 - Previous hidden state h_{t-1}
 - Vector for current word x_t
- Learn linear function of both inputs, add bias, apply non-linearity
- Parameters: Recurrence params (W_h , W_x , b), initial hidden state h_0 , word vectors

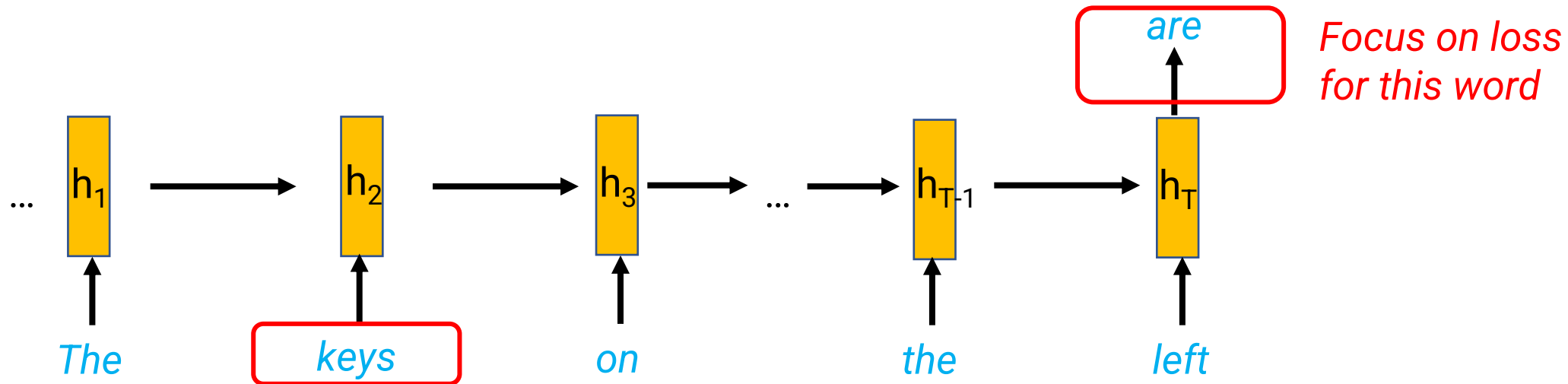
Review: Long-Range Dependencies

- Every step, you update the hidden state with the current word
- Over time, information from many words ago can easily get lost!
- This means RNNs can struggle to model **long-range dependencies**

*The **keys** to the cabinet by the door on the left **are** (on the table)*

Review: Vanishing Gradient Problem

- Gradient through “*keys*” word vector: $\delta \text{Loss} / \delta(h_T) * \delta(h_T) / \delta(h_{T-1}) * \delta(h_{T-1}) / \delta(h_{T-2}) * \dots * \delta(h_3) / \delta(h_2) * \delta(h_2) / \delta(x_2)$
 - What is each individual $\delta(h_t) / \delta(h_{t-1})$ term?
 - Elman network: $h_t = \tanh(W_h h_{t-1} + W_x x_t + b)$, $\frac{\delta h_t}{\delta h_{t-1}} = \underbrace{\tanh'(W_h h_{t-1} + W_x x_t + b)}_{\text{Ignore for now}} \cdot \underbrace{W_h}_{\text{The same parameter over and over!}}$
 - After t timesteps, have a factor of $(W_h)^t$ (to the t -th power)!
 - If $W_h \ll 1$, this quickly becomes 0 (“vanishes”)



Outline

- More on reducing the effect of vanishing gradients
- Sequence-to-sequence learning
- Attention

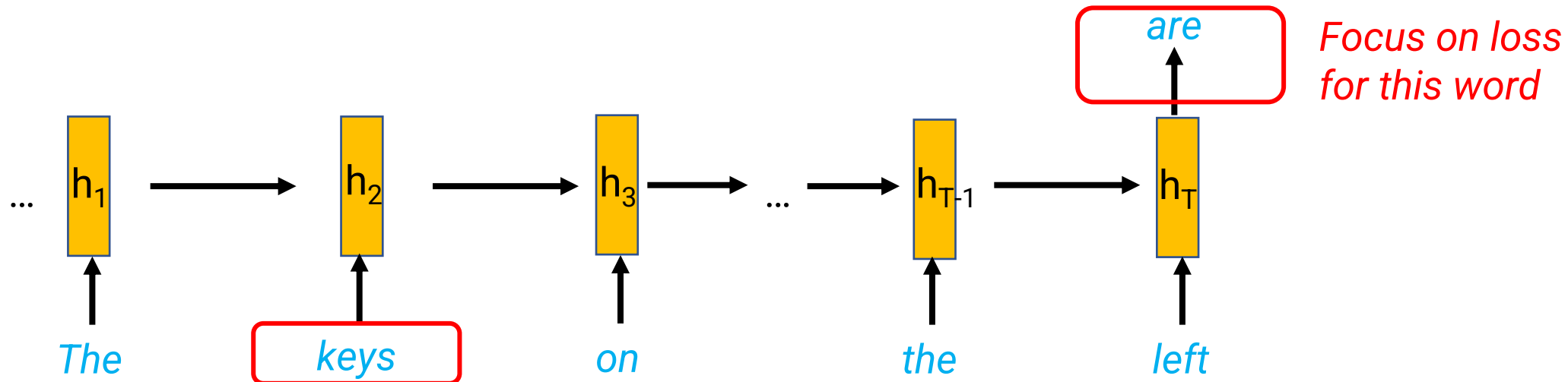
Review: Avoiding Vanishing Gradients

Where did we go wrong?

$$h_t = \tanh(\underbrace{W_h h_{t-1}}_{\text{Multiplicative}} + W_x x_t + b), \quad \frac{\delta h_t}{\delta h_{t-1}} = \tanh'(W_h h_{t-1} + W_x x_t + b) \cdot \underbrace{W_h}_{\text{Leads to repeated multiplication by } W_h}$$

Multiplicative
relationship between previous
state and next state

Leads to repeated
multiplication by W_h

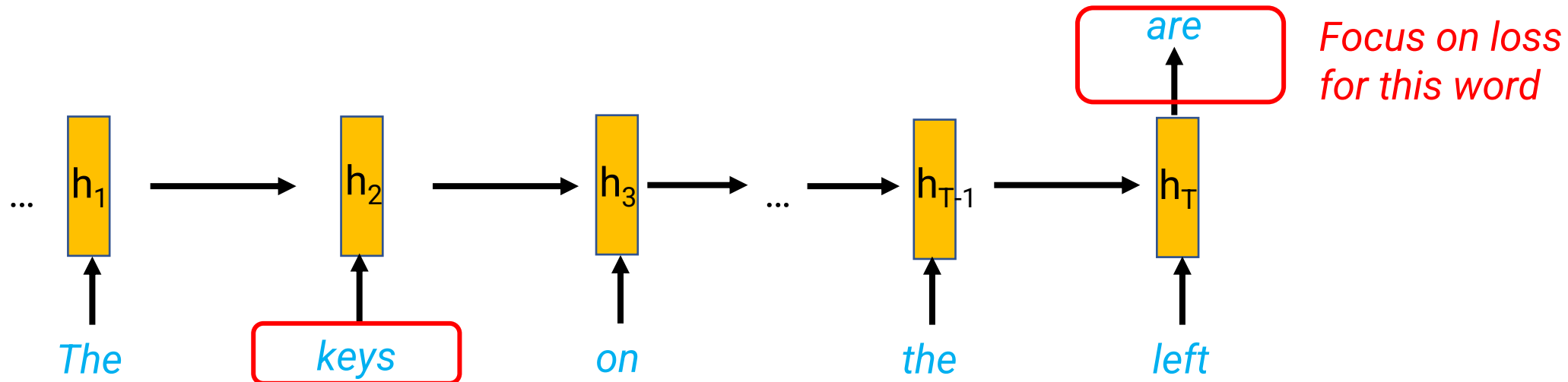


Review: Avoiding Vanishing Gradients

- Extreme idea: A purely additive relationship
 - Pro: No vanishing gradients
 - Pro: Old hidden state carried through to all future times
 - Con: May be good to “forget” irrelevant information about old states

$$h_t = h_{t-1} + \underbrace{g(h_{t-1}, x_t)}_{\text{Additive relationship}}$$

$$\frac{\delta h_t}{\delta h_{t-1}} = 1 + \underbrace{\frac{\delta}{\delta h_{t-1}} g(h_{t-1}, x_t)}_{\text{Gradients also add, not multiply}}$$

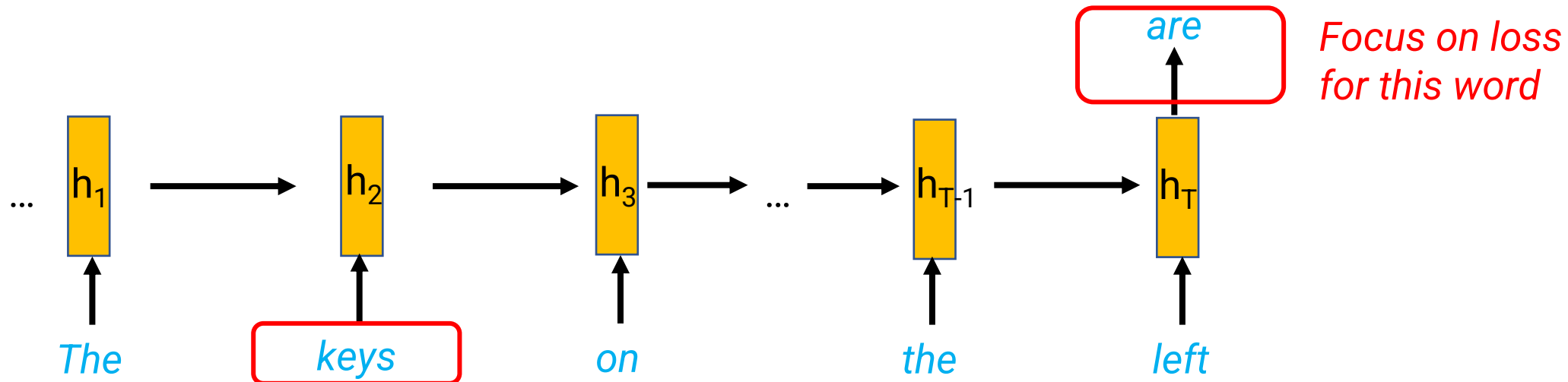


Avoiding Vanishing Gradients

- Middle-ground: **Gated** recurrence relationship
 - Additive component makes gradients add, not multiply = less vanishing gradients
 - Forget gate allows for selectively “forgetting” some neurons within hidden state
 - When forget gate is all 1’s, becomes the purely additive model (no vanishing)

$$h_t = h_{t-1} \odot \underbrace{f(h_{t-1}, x_t)}_{\substack{\text{“forget gate”} \\ \text{in } [0, 1]}} + \underbrace{g(h_{t-1}, x_t)}_{\substack{\text{Additive} \\ \text{relationship}}}$$

Elementwise multiplication



Gated Recurrent Units (GRUs)

- One type of gated RNN
 - Here z_t is the “forget gate” vector
 - If $z_{ti} = 1$:
 - Forget the i -th neuron
 - Allow updating its value to \tilde{h}_{ti} , computed from r_{ti}
 - If $z_{ti} = 0$:
 - Don't forget the i -th neuron
 - Do not allow updating its value
 - Additive relationship between h_{t-1} and h_t
 - Parameters: W_z W_r W

Sigmoid ensures gate is between 0 and 1

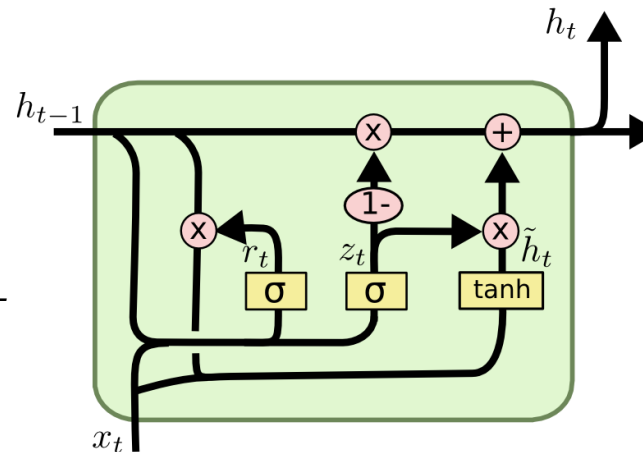
Forget gate $z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

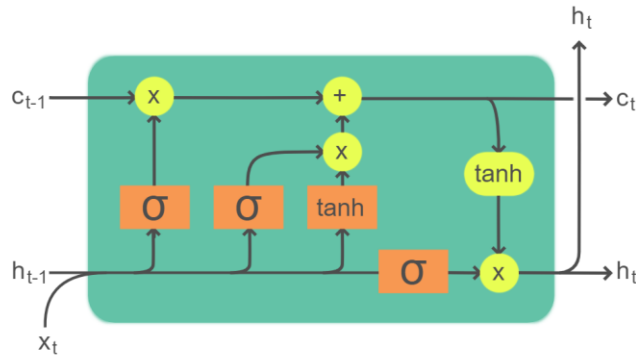
Planned update to h_t $\tilde{h}_t = \tanh(W \cdot [r_t \odot h_{t-1}, x_t])$

Actual update to h_t $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$

Forget parts of h_{t-1} Add update to parts that were forgotten



Long Short-Term Memory (LSTM)



Legend:

Layer	Componentwise	Copy	Concatenate
Orange box	Yellow circle	\updownarrow	\rightarrow

Forget gate $f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

Planned update to c_t $\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$

Cell state $c_t = f_t \odot c_t + i_t \odot \tilde{c}_t$

Hidden state $h_t = o_t \odot \tanh(c_t)$

Add the previous cell state * forget gate

- Another, more complicated gated RNN
- Commonly used in practice
- Overall idea:
 - Split the hidden state into normal hidden state h_t and “cell” state c_t
 - Cell state uses gated recurrence with forget gate f_t
 - Hidden state is gated function of cell state
 - Also has input and output gates i_t & o_t

What do LSTMs learn?

- Here: a character-level LSTM (not word-level)
- Blue/Green: Low/high values of 1 neuron
- Below: Top-5 predictions for next character
- This neuron seems to detect whether we're inside a URL

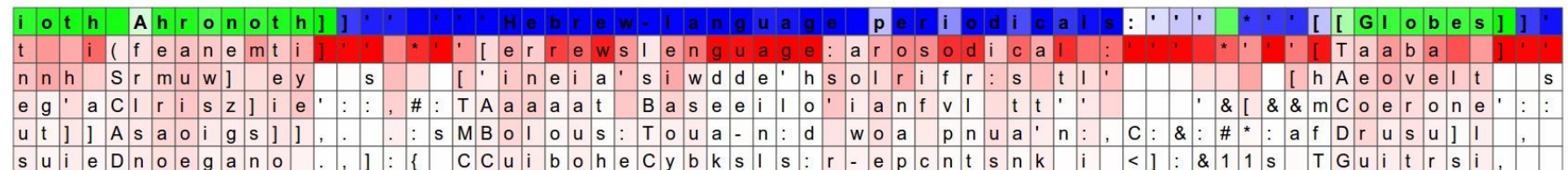
t	t	p	:	/	/	w	w	w	.	y	n	e	t	n	e	w	s	.	c	o	m	/]	E	n	g	l	i	s	h	-	l	a	n	g	u	a	g	e	w	e	b	s	i	t	e	o	f	i	s	r	a	e	l	'	s	i	a	r			
t	p	:	/	/	w	w	w	.	b	a	c	a	h	e	t	s	.	c	o	m	/		-	x	g	l	i	s	h	l	i	n	g	u	a	g	e	s	a	i	r	s	i	t	e	o	f	t	s	i	a	e	l	'	s	s	i	n	g			
d	:	x	n	e	.	w	a	e	a	.	a	w	a	t	o	a	.	s	&	n	t	i	a	c	a	-	s	a	r	d	e	e	l	h	o	a	n	t	b	i	s	a	n	f	a	n	r	e	i	f	'	a	a	t	d							
m	w	-	2		p	i	i	i	s	o	e	s	s	i	s	.	/	e	r	n	.	c]	(d	c	e	e	n	e	p	e	s	a	a	i	k	i	i	e	e	l	e	d	h	,	i	r	t	h	r	a	o	n	s	e	,	c	o	s	e	
d	r	.	<	:	a	h	b	-	n	p	t	w	t	.	x	i	g	h	/	m	a)	T	v	d	r	y	z	i	c	o	u	e	d	i	s	u	:	t	h	a	-	o	o		t	u	,	s	t	u	i	f	l	v	e	p	e	r	y		
s	t	p	,	t	c	o	a	2	d	r	u	i	w	o	c	l	e	n	s	r]	p	.	i	l	v	a	o	d	,	,	e	y	t	c	-	n	d	m	-	o	i	b	u	v	s]	b	b	i	m	s	u	l	t	a	t	l	y	b	n	

g	e	s	t		n	e	w	s	p	a	p	e	r		'	'	[[Y	e	d	i	o	t	h		A	h	r	o	n	o	t	h]]	'	'		'	'	H	e	b	r	e	w	-	l	a	n	g	u	a	g	e		p	e	r	i	o	d	
e	t		a	a	w	s	p	a	p	e	r	s	o	'	[[T	e	l	t		i	(f	e	a	n	e	m	t	i]	'	'	*	'	'	[e	r	r	e	w	s	l	e	n	g	u	a	g	e	:	a	r	o	s	o	d	i				
i	r		s	c	o	e		e	n	a		i	T	T	h	A	o	a	i	n	n	h		S	r	m	u	w]	e	y		s					['	i	n	e	i	a	'	s	i	w	d	d	e	'	h	s	o	l	r	i	f	r	:			
u	s	.	.	s	e	t	l	g	o	r		s	.	a	s	a	t	C	a	r	e	e	g	'	a	C	l	r	i	s	z]	i	e	'	:	:	,	#	:	T	A	a	a	a	a	t		B	a	s	e	e	i	l	o	'	i	a	n	f	v	l	
-			t	u	a	e	v	r	t	i	d	,	t	B	A	m	S	u	s	y	u	t]]	A	s	a	o	i	g	s]]	,	.	.	:	s	M	B	o	l	o	u	s	:	T	o	u	a	-	n	:	d		w	o	a		p	n	u		
a	,	d	,	i	i	u	i	t	i	c	p	.]	(l	S	v	H	v	t	u	s	u	i	e	D	n	o	e	g	a	n	o	.	,]	:	{		C	C	u	i	b	o	h	e	C	y	b	k	s	l	s	:	r	-	e	p	c	n	t	s	

i	c	a	i	s	:	'	'	'	*	'	'	[[G	l	o	b	e	s]]	'	'		[h	t	t	p	:	/	/	w	w	w	.	g	l	o	b	e	s	.	c	o	.	i	l	/]	b	u	s	i	n	e	s	s		d	a				
c	a	i	:	'	'	'	*	'	'	'	[T	a	a	b	a]	'	'	([t	t	p	:	/	/	w	w	w	.	b	u	o	b	a	l	.	c	o	m	u	n	/	s	A	-	y	t	i	n	e	s	s		a	e	t						
s	t	i	'									[h	A	e	o	v	e	l	t		s		a	h	a		d	:	x	g	e	.	w	a	o	i	r	.	r	t	o	a	.	e	l	.	i	T	&	a	i	e	g		e	o	o	y						
t	t	'	'									&	[&	&	m	C	o	e	r	o	n	e	'	:	:	i	'	o	d	w	.	,	:	n	i	i	s	a	a	u	e	.	e	n	i	/	o	m	l	c	C	.	(e	f	t	g	i	r		i	i	u	
a	'	n	:	,	C	:	&	:	#	:	*	:	a	f	D	r	u	s	u]	l	,	.	o	m	e	l		p	<	,	d	h	a	:	d	e	u	o	o	t	/	i	h	n	c	s	i	f	S	,	u	r	h	o	s		t	,	t	u	n			
n	k	i		<]	:	&	1	1	s		T	G	u	i	t	r	s	i	,		:	b	a	c	m	r	-	x	t	p	o	b	-	g	r	e	s	i	s	l	e	r	l	n	a	f	a	D]	l	o	s	p	t	a	d	,	i	f	r	m			

i	l	y		*	'	'	[[H	a	a	r	e	t	z]	H	a	'	A	r	e	t	z]]	'	'		[h	t	t	p	:	/	/	w	w	w	.	h	a	a	r	e	t	z	.	c	o	.	i	l	/]	R	e	l	a	t	i	v	
l	y		*	'	'	[[T	e	r	r	d	n		F	e	r	a	n	t	a	h]]	'	'	([t	t	p	:	/	/	w	w	w	.	b	o	n	m	d	s	t	.	c	o	m	u	n	/	s	-	e	s	a	t	e	o	i			
r	e		'		'	h	A	i	l	n	n	t	t	e	H	a	l	s	r	c	n	o	l	'	s		a	h	a		d	:	x	n	e	.	w	a	a	m	r	t	d	h	e	o	h	.	o	l	.	c	&	o	p	i	n	i	v	e				
k	i	:	*	s	C	O	S	a	n	l	t		h	i	T	i	m	'	l	i]	e		:	,	i	m	c	d	w	-	2		p	h	i	i	s	e	r	d	i	t	.	i	n	a	/	c	m	f	i	.	(a	f	l	c	a	n	a			
d	s	-	!	[t	B	T	C	o	m	m	g	d]]	W	o	n		a	a	e	,	:	.	b	a	e	r	r	.	<	t	a	i	b	-	d	u	l	c	n	n	c	/	a	r	n	e	s	i]	l	i	c	e	y	s	t	o				
n	d	s	#	&	:	G	l	D	u	v	c	c	s	a	o	S	u	c	l	t	e	l]	z]	,	:	o	'	o	m	t]	,	:	e	o	a	2	n	i	v	f	s	r	o	o	e	i	u	n	a	l	a)	u	v	v	r	o				

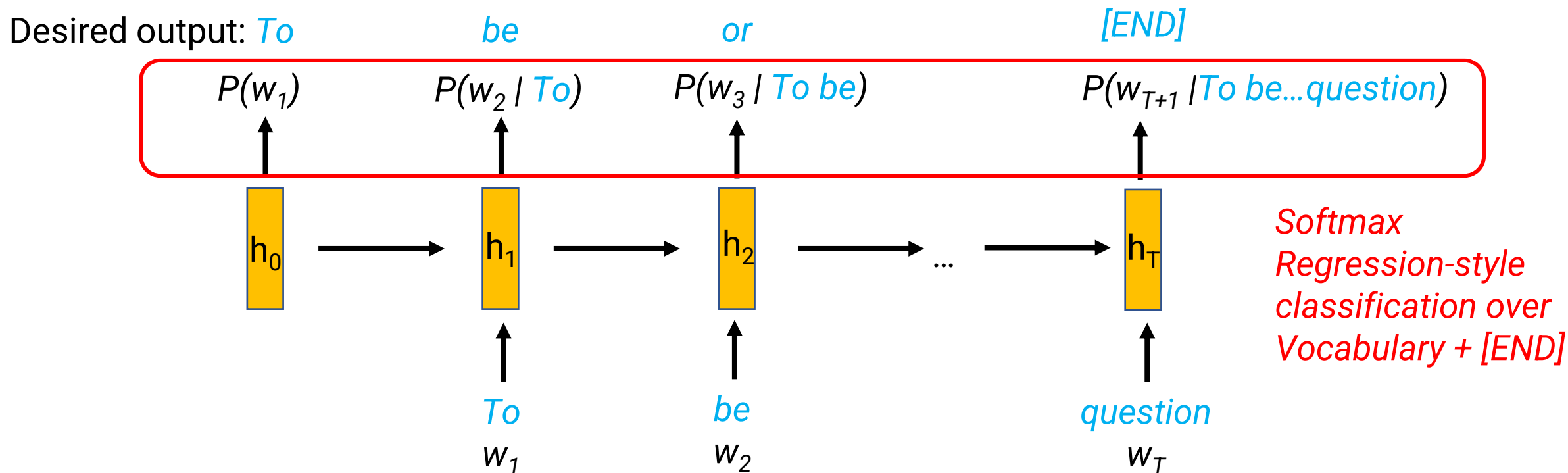
- Here: a character-level LSTM (not word-level)
- Blue/Green: Low/high values of 1 neuron
- Below: Top-5 predictions for next character
- This neuron fires only inside a Markdown `[[link]]` (so it knows when to close the square brackets?)



Outline

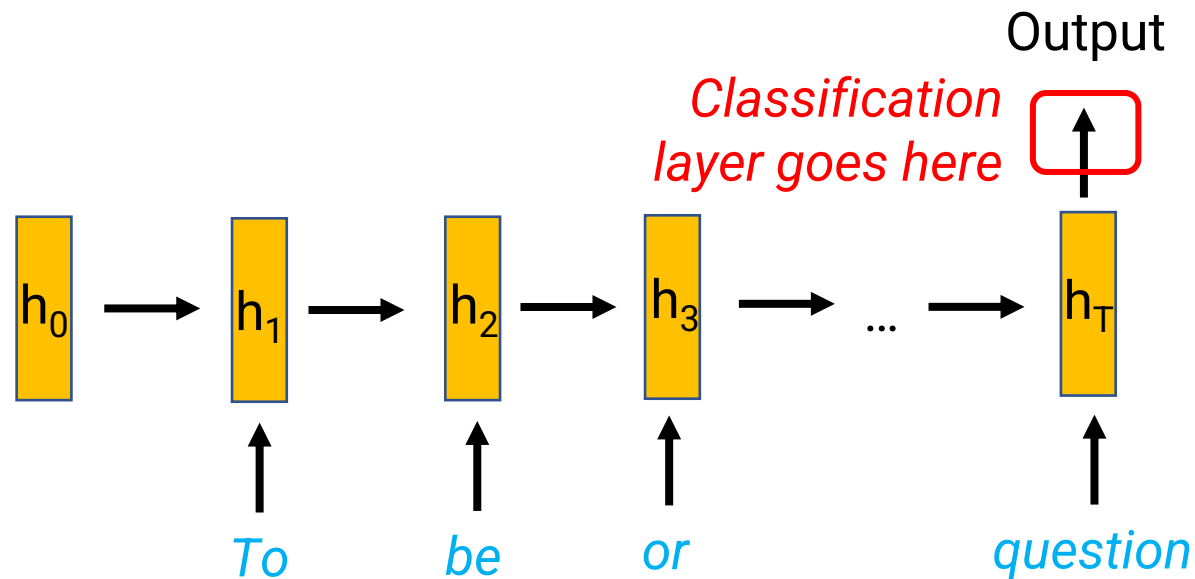
- More on reducing the effect of vanishing gradients
- **Sequence-to-sequence learning**
- Attention

Review: Autoregressive Language Modeling



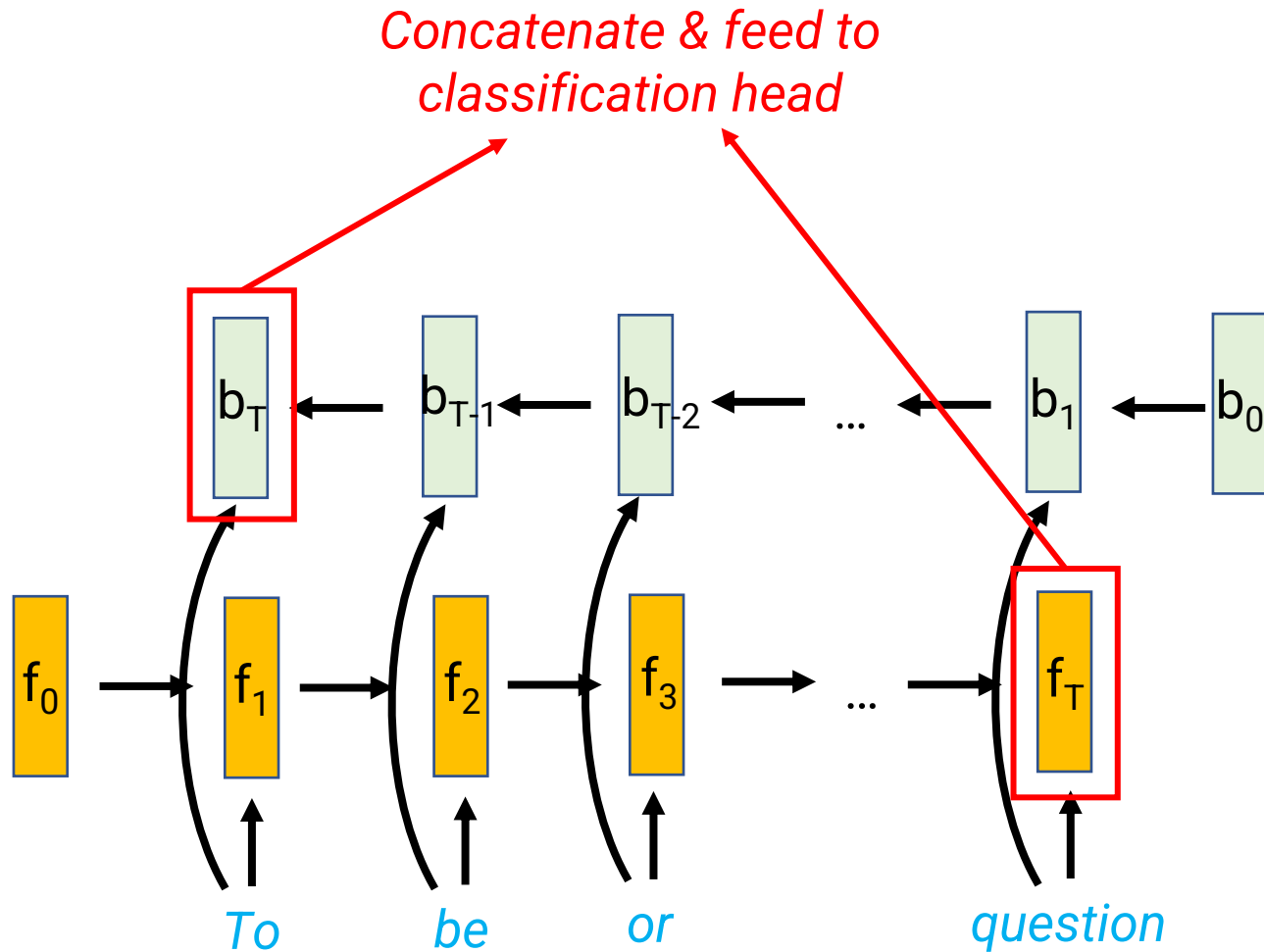
- At each step, probabilistically predict the next word given current hidden state
- One step's desired output is the next step's input ("autoregressive")
- To mark end of sequence, model should predict the *[END]* token
- Called a "Decoder": Looks at the hidden state and "decodes" next word

Text classification (“Encoder only”)



- First run an RNN over text
- Use the final hidden state as an “encoding” of the entire sequence
- Use this as features, train a classifier on top
- Downside: Later words processed better than early words (long range dependency issues)

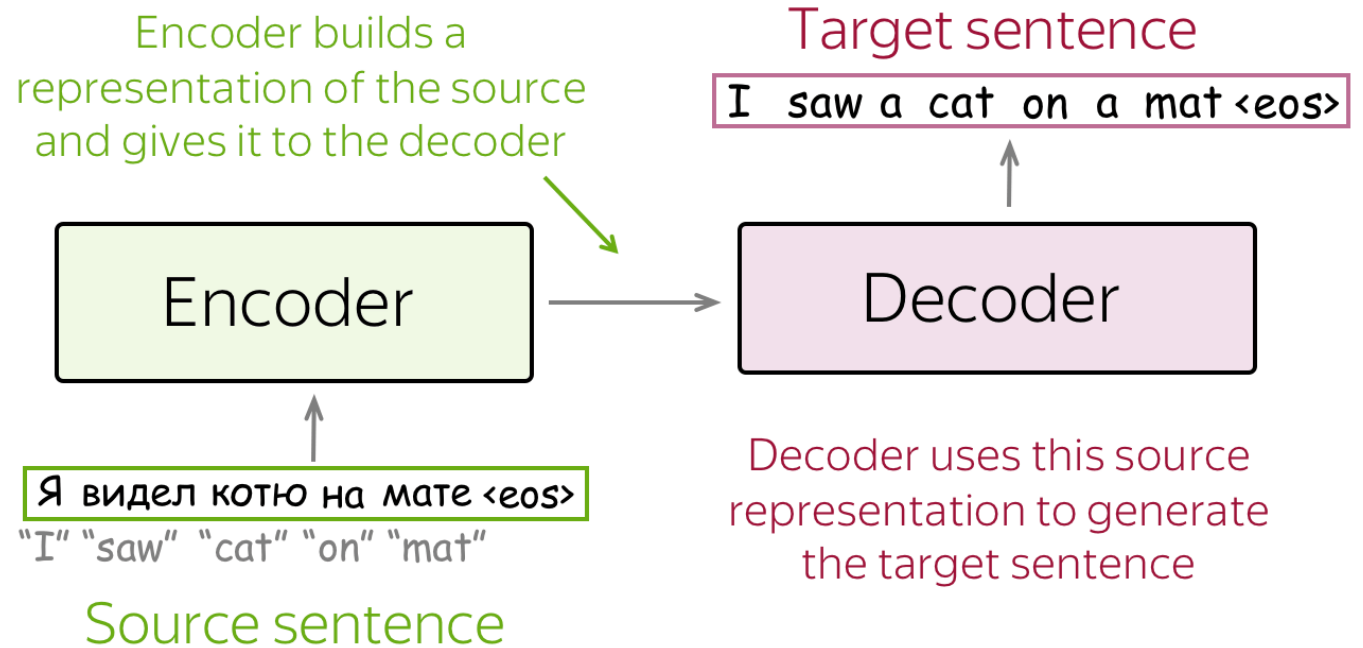
Bi-directional encoders



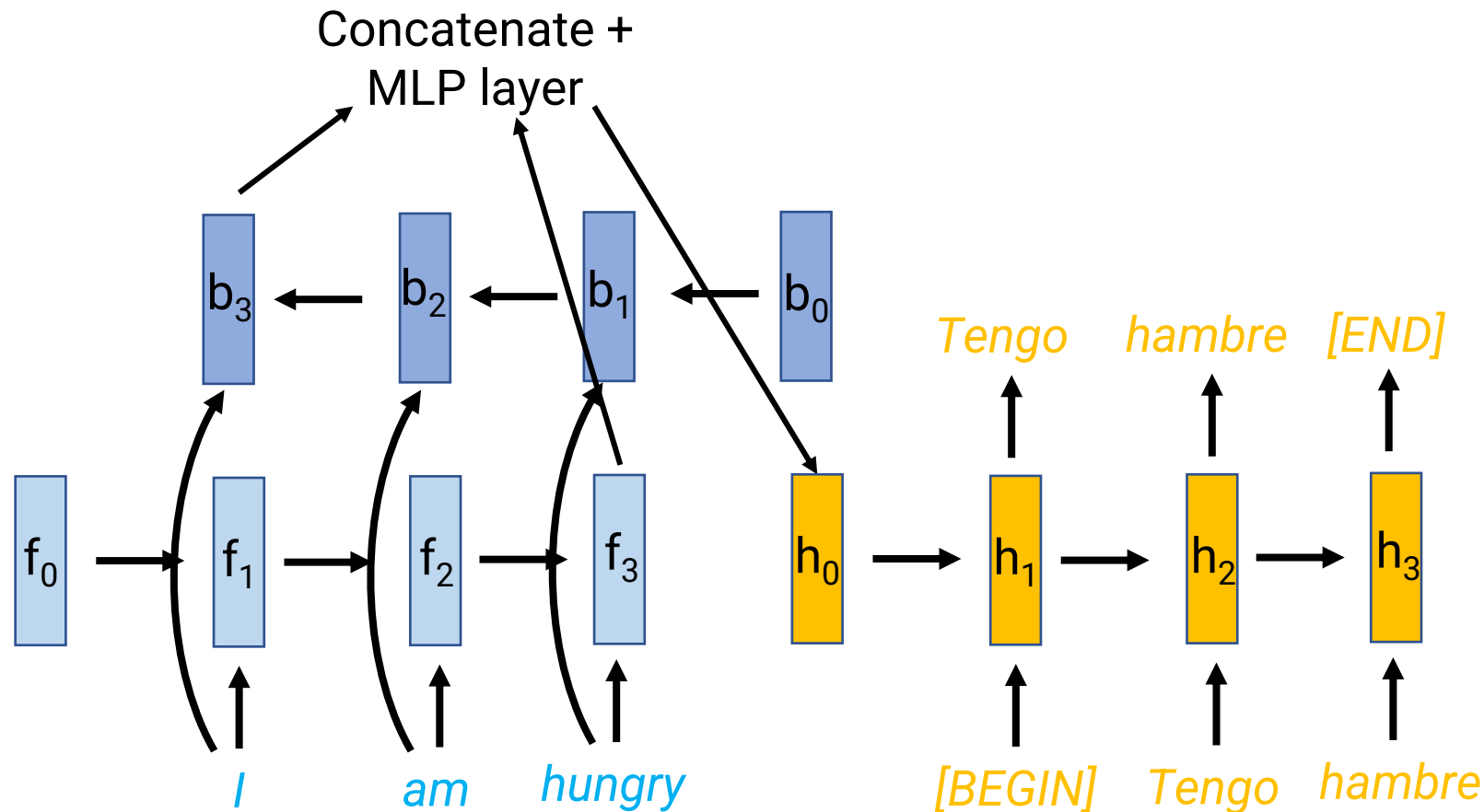
- Run one RNN left-to-right, and another one right-to-left
 - (I'll call forward-direction hidden states f_t , backward-direction hidden states b_t)
- Concatenate the 2 final hidden states as final representation
 - Note: This encoding is twice as large now—we've doubled the number of features passed to the final classifier

Sequence-to-sequence Tasks

- **Sequence-to-sequence** tasks
 - Machine translation (Russian -> English)
 - Summarization (Document -> Summary)
 - Personal Assistants (Command -> Action)
- Encoder: "Reads" the input sentence, produces a feature vector summarizing the input
- Decoder: Uses that vector as its initial state, predicts output tokens one at a time



Encoder-decoder model



- Example: Machine Translation
 - Input = English text
 - Output = Spanish text
- Encoder: Process English sentence into vector
 - E.g. Bidirectional encoder + MLP layer to generate decoder's initial state
- Decoder: Use vector as initial hidden state and start doing language modeling in Spanish
- Vector space acts as a "shared language"

The Power of Building Blocks

- We now know about a lot of components
- We can assemble in any way we think makes sense, given the input and desired output
- We only have to think about the forward pass!
- Code to learn parameters is always the same:
 - Get a batch of training examples
 - Compute the loss (forward pass)
 - Run backpropagation to get gradient of loss w.r.t. parameters
 - Gradient descent to update parameters



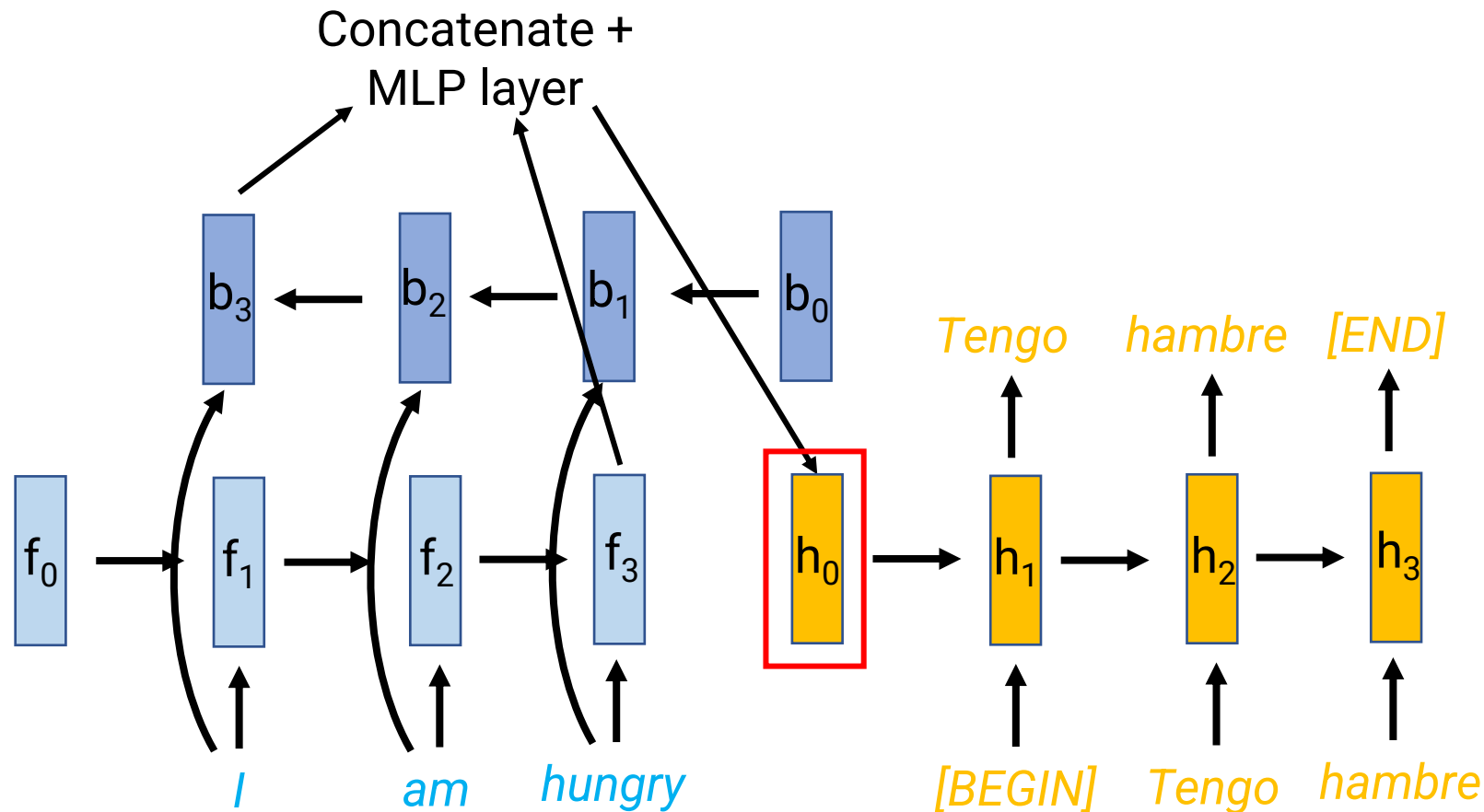
Announcements

- HW2 due today @ 11:59pm
- Section Friday: Midterm Review (practice exam + questions)
- Midterm exam: Thursday March 13
 - Practice exams released on website
 - Everything through end of today's lecture is fair game
 - Will post spreadsheet of lecture video links on Piazza

Outline

- More on reducing the effect of vanishing gradients
- Sequence-to-sequence learning
- Attention

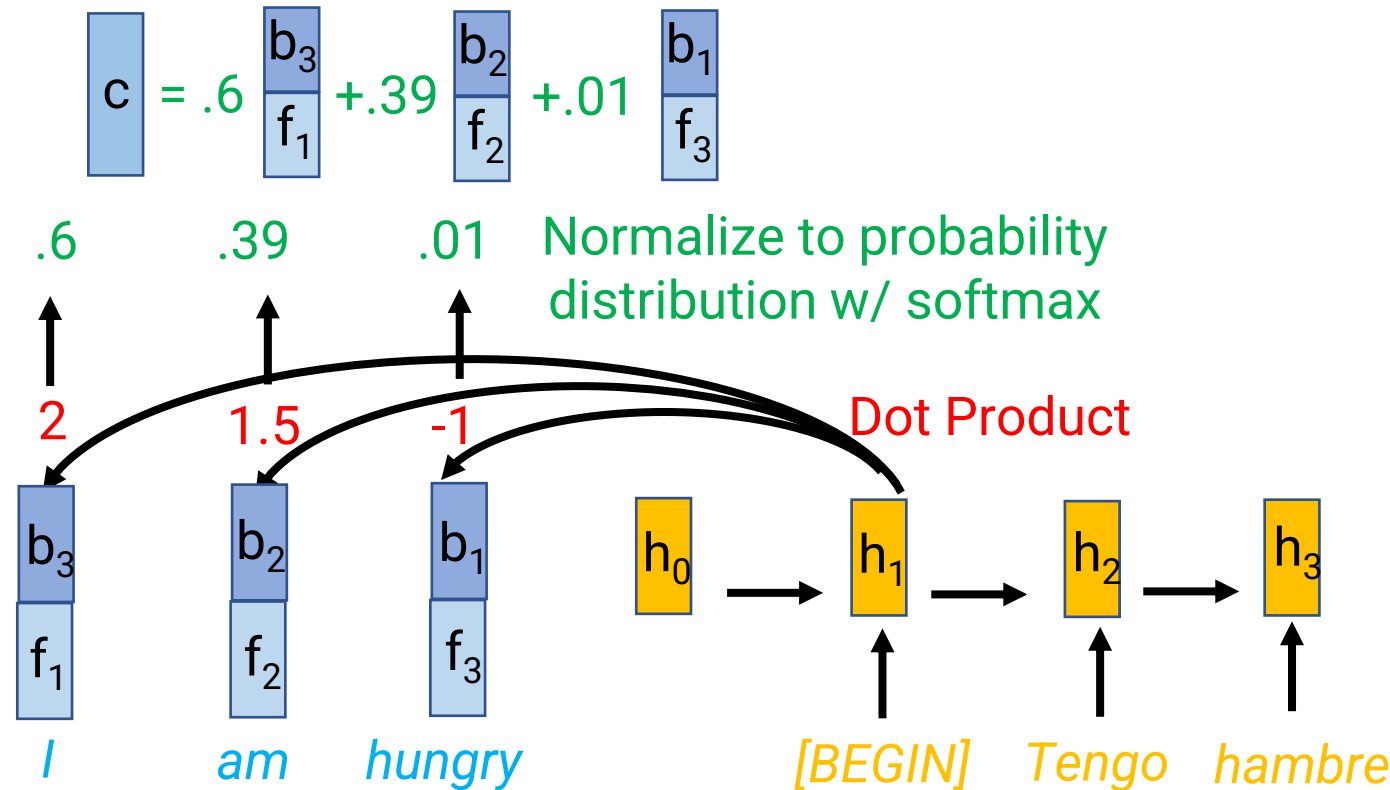
What's missing? Alignment



- Challenge: The single encoder output has to store information about the entire sentence in a single vector
- Better strategy: Look for the next input word to translate, then translate that word
- Traditional MT: Alignment between input & output sentences
- Can we get a neural network to model alignments?

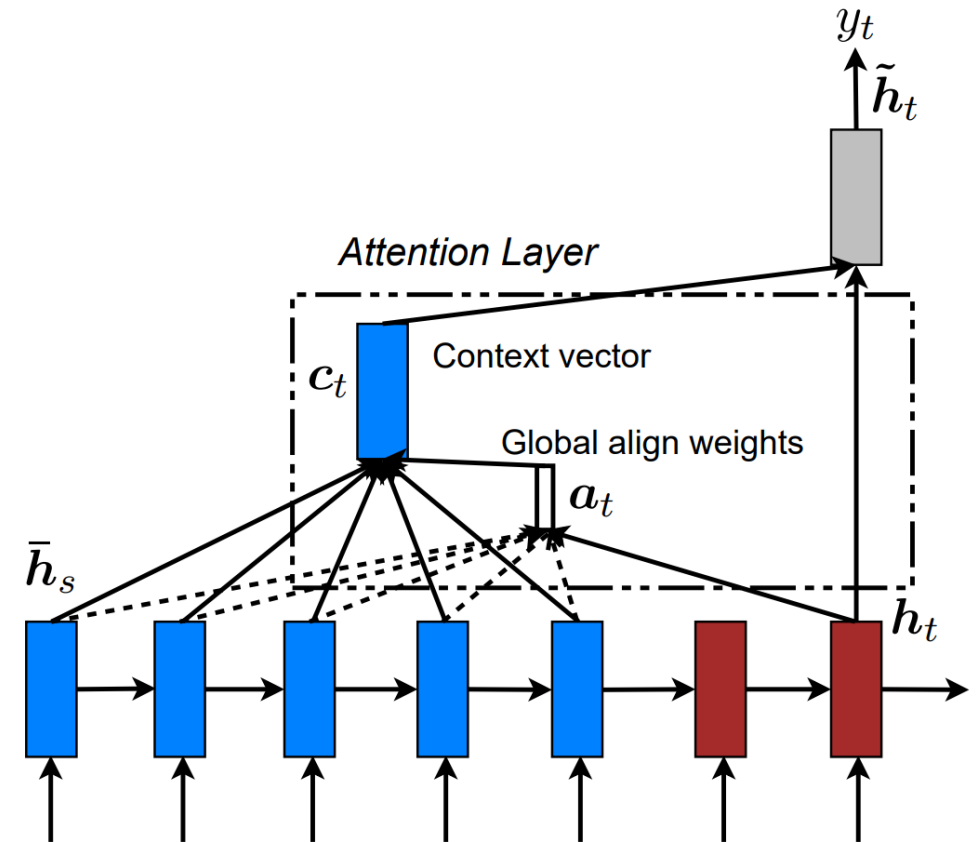
Attention

- Compute **similarity** between **decoder** hidden state and each **encoder** hidden state
 - E.g., **dot product**, if same size
- Normalize similarities to **probability distribution with softmax**
- Output: “Context” vector c = weighted average of encoder states based on the probabilities
 - No new parameters (like ReLU/max pool)
- Use c when computing decoder outputs or transitions
- Intuition
 - Step 1: Find similar input words
 - Step 2: Grab the encoder representation of those words
 - Step 3: Tell the decoder that this is relevant

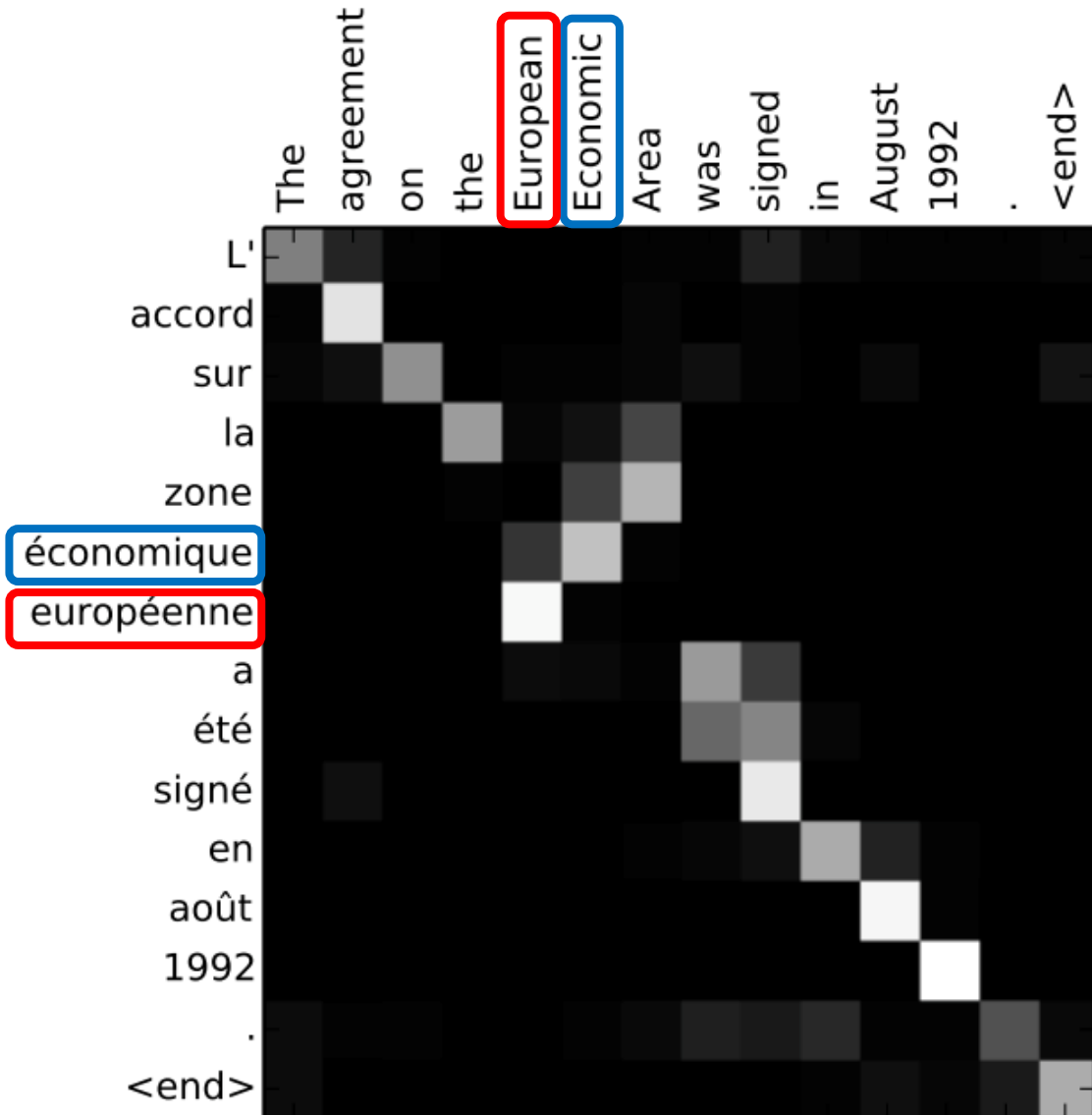


Using Attention in Seq-to-seq model

- Many similar ways one could implement an attention mechanism
- Example from a well-known 2015 paper by Luong et al. on machine translation
 - Blue = encoder states
 - Red = decoder states
 - Note: Encoder was unidirectional here
- Dot-product decoder state h_t with encoder states, then apply softmax to produce weights a_t
- Weighted sum of encoder states yields context vector c_t
- Context vector c_t concatenated with decoder state h_t , fed through 1 MLP layer to generate \tilde{h}_t
- \tilde{h}_t used to make prediction y_t



Visualizing attention



- Source is English, Target is French
- Each row is a probability distribution over the English text
- Alignment makes sense, overcomes word order differences
 - When generating “**économique**” attend to “**Economic**”
 - When generating “**européenne**” attend to “**European**”

Attention as Retrieval

The screenshot shows a Google search interface. The search bar contains the text "training a machine translation model". Below the search bar, there are tabs for "Images", "Videos", "Perspectives", "Python", "Example", "Online", "Github", "Shopping", and "News". The search results show "About 174,000,000 results (0.18 seconds)". The first result is from Pangeanic, titled "How to train your machine translation engine", dated Oct 20, 2021. The second result is from Machine Learning Mastery, titled "How to Develop a Neural Machine Translation System from ...", dated Oct 6, 2020. The third result is from GitHub, titled "Tutorial: Neural Machine Translation - seq2seq".

Google

training a machine translation model

Images Videos Perspectives Python Example Online Github Shopping News

About 174,000,000 results (0.18 seconds)

Pangeanic
https://blog.pangeanic.com › train-machine-translation-e...

How to train your machine translation engine
Oct 20, 2021 — A **machine translation** engine is software capable of translating texts from a source language to a target language. Applying artificial ...
How To Train Your Machine... · 1. Incorporation Of The Base... · Tips For Improving The...

Machine Learning Mastery
https://machinelearningmastery.com › Blog

How to Develop a Neural Machine Translation System from ...
Oct 6, 2020 — **Machine translation** is a challenging task that traditionally involves large statistical **models** developed using highly sophisticated linguistic ...

GitHub
https://google.github.io › nmt

Tutorial: Neural Machine Translation - seq2seq
For more details on the theory of Sequence-to-Sequence and **Machine Translation models**, we recommend the following resources: ... The **training** script will save ...
Neural Machine Translation... · Alternative: Generate Toy Data · Training

- Consider a search engine:
 - **Queries**: What you are looking for
 - E.g., What you type into Google search
 - **Keys**: Summary of what information is there
 - E.g., Text from each webpage
 - **Values**: What to give the user
 - E.g., The URL of each webpage

General Form of Attention

(8) Attention Layer

- Inputs (all vectors of length d):
 - Query vector q
 - Key vectors k_1, \dots, k_T
 - Value vectors v_1, \dots, v_T
- Output (also vector of length d)
 - Dot product q with each key vector k_t to get score s_t :

$$s_t = q^\top k_t$$

- Softmax to get probability distribution p_1, \dots, p_T :

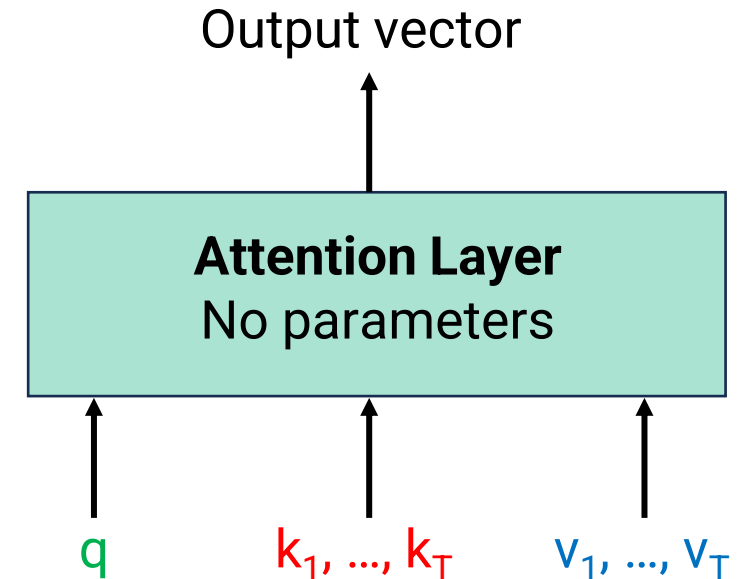
$$p_t = \frac{e^{s_t}}{\sum_{j=1}^T e^{s_j}}$$

- Return weighted average of value vectors:

$$\sum_{t=1}^T p_t v_t$$

Dominated by the values corresponding to the “best-matching” keys

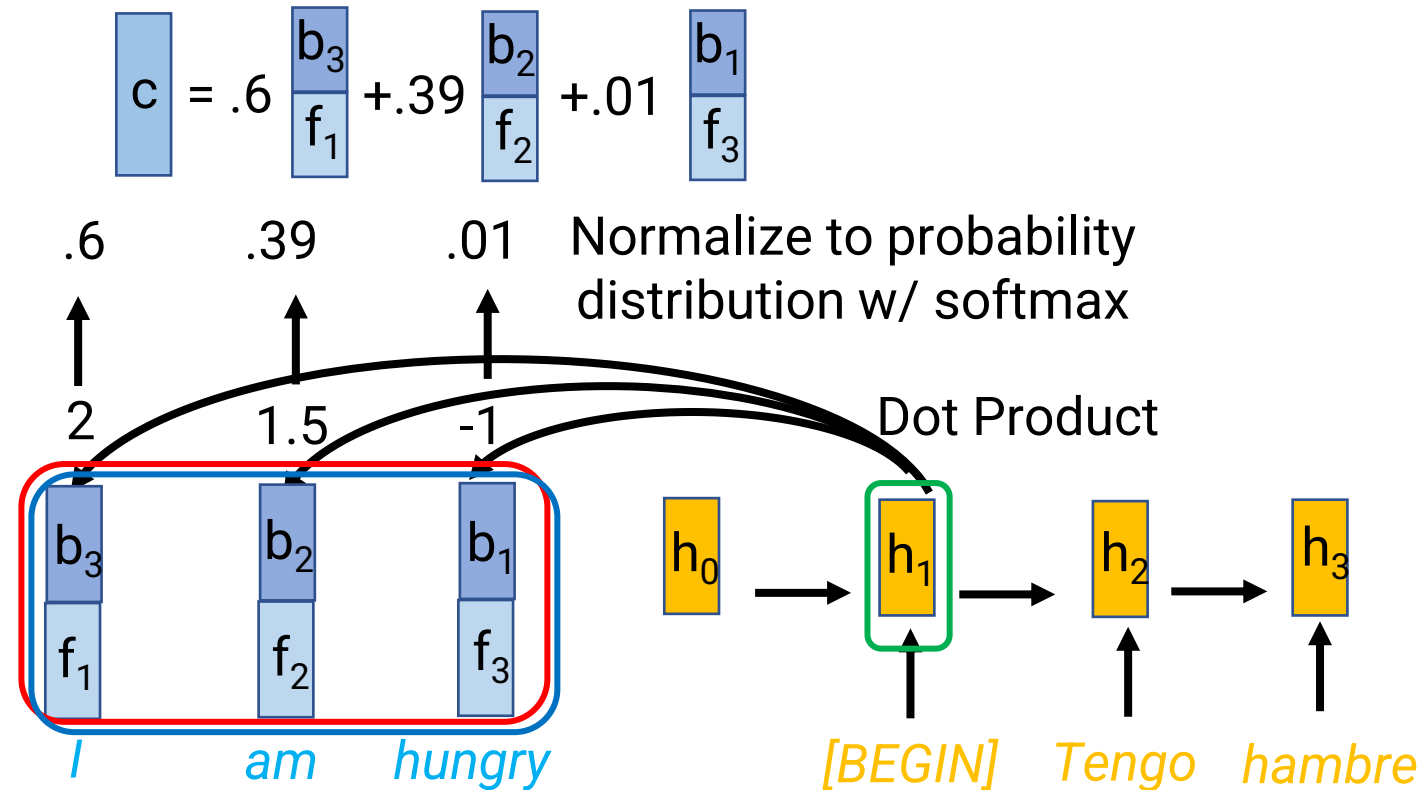
How well does the query match each key?



Attention in Seq-to-seq RNNs

- Applies a general attention layer where:

- Query** = Current decoder hidden state
- Keys** = Encoder hidden states
- Values** = Encoder hidden states (same as keys)



Conclusion

- GRUs, LSTMs: Add gates + additive connections to reduce vanishing gradients
- Ways to use RNNs
 - As a decoder: To generate text
 - As an encoder: To produce feature vectors for text
 - Sequence-to-sequence: Use 2 RNNs, one for each purpose
- Attention: Know which part of the input matters when generating each word of the output
 - After Spring Break: Can we get rid of RNN's, and only use attention?