CSCI 566: Deep Learning and Its Applications

Jesse Thomason

Lecture 5: Recurrent Neural Networks

Map to the Midterm

January 2023

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

www.a-printable-calendar.com

February 2023

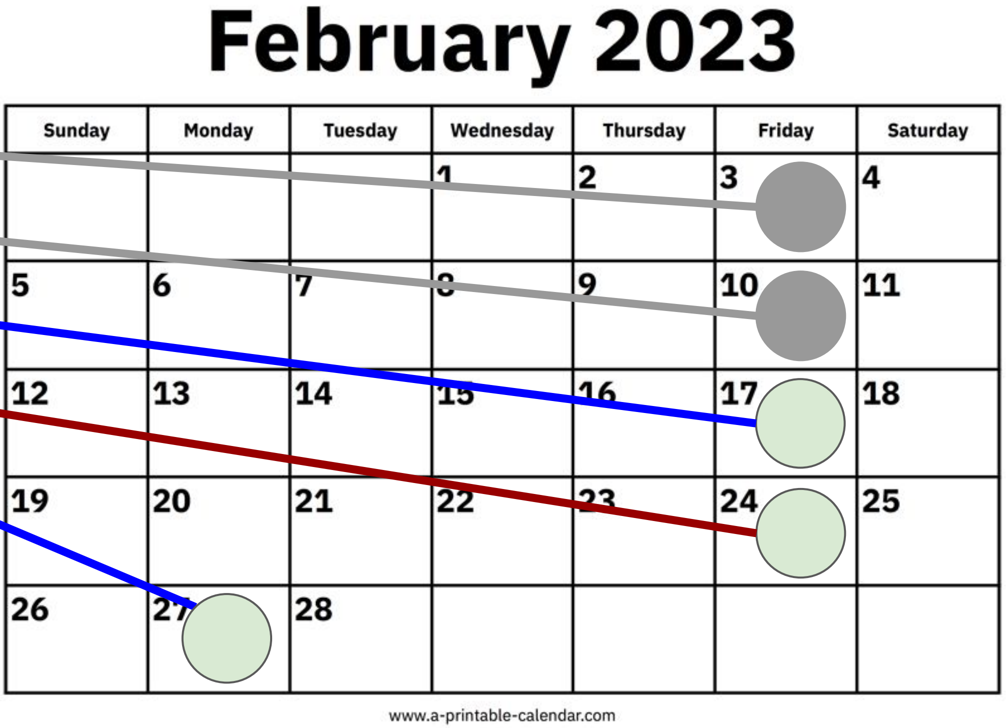
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28				

www.a-printable-calendar.com

Module 1: Neural Network Basics

[Feb 17] Wrapping Up Module One

Project Teams Formed	Feb 3
Assignment 1 Out	Feb 10
Project Proposal Due	Feb 17
Midterm Exam	Feb 24
Assignment 1 Due	*Feb 27*



Course Project Proposals Due **Today**

- See Piazza for assignment description details; 10 slides addressing:
  - Q1: What will your project aim to do? Articulate your objectives using absolutely no jargon.
  - Q2: What is new in your approach and why do you think it will be successful?
  - Q3: Who cares? If you are successful, what difference will it make?
  - Q4: What are the risks?
  - Q5: How much will it cost?
  - Q6: Who will do what? Outline your expectations for your team to hold yourselves accountable to one another and to us.
  - Q7: What is your expected timeline?

The Heilmeier Catechism

Midterm Next Week [Feb 24]

- The midterm exam will be conducted on Feb 24 during our class period after announcements (1:20pm to 4:20pm PT)
- You do not need to be physically present to take the exam
- The exam will be in the form of a link to a PDF (exam questions) and a link to a Google Form (to submit answers)
- The exam will be partially or entirely multiple choice
- The exam PDF and Google Form will be available *only* during the 1:20pm-4:20pm window; links will go up on Piazza
- We are still working on the exam, but we do not expect it to take most people the full class period to complete.

In-class “Pop-up” Quizzes

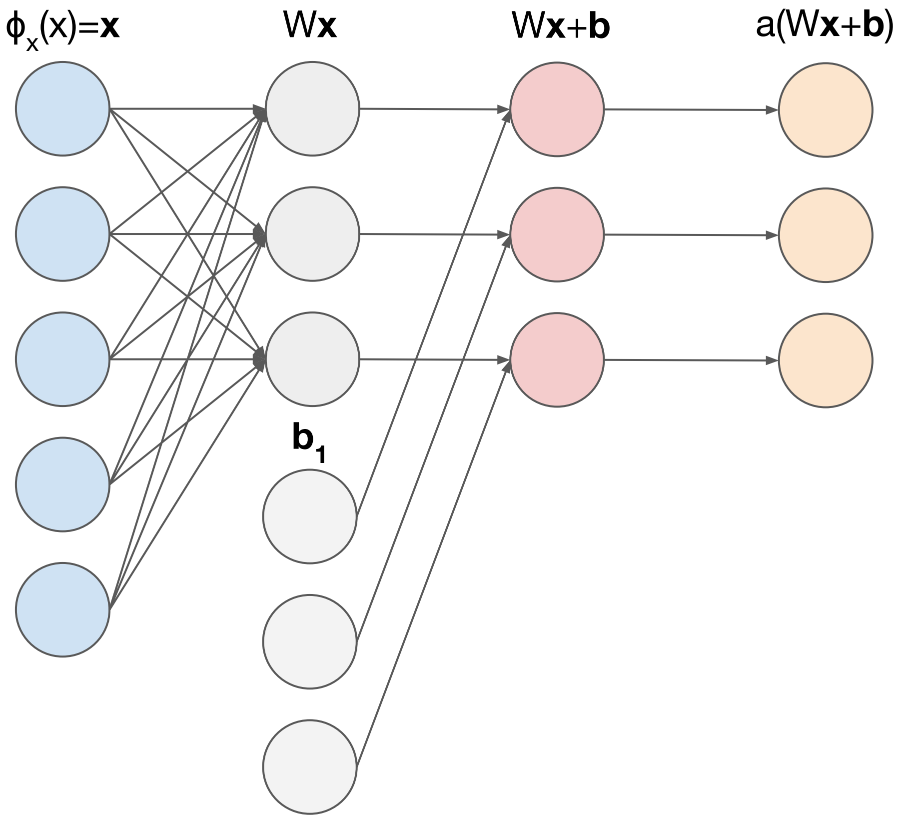
- These quizzes are a method of engaging with you while the class is happening
- They account for a very small fraction of your total grade
- There will probably be more than 5 of them
- They can, generally, not be made up or postponed
- If you usually attend class, this part of your evaluation will just be totally fine

Deliverable	**Points of the total grade**
Pop-up Quizzes	5
…	…
TOTAL	**100**

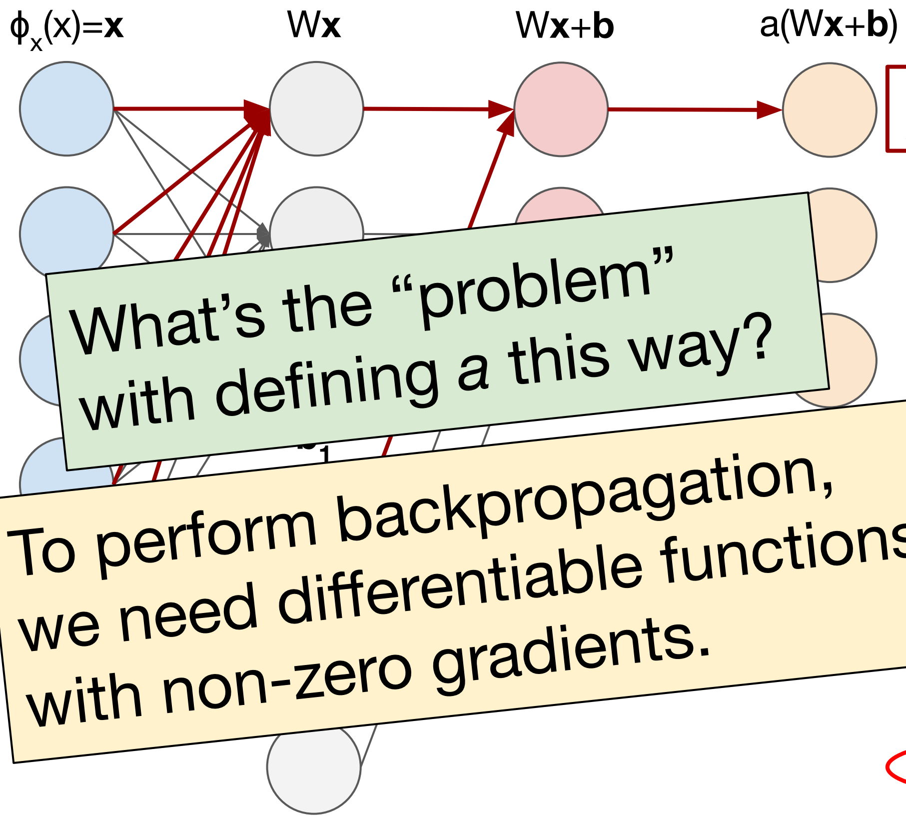
Overview of Today's Plan

- ~~Course organization and deliverables~~
    - Any questions before we move on?
- Recurrent Neural Networks
- Long-Short Term Memory
- Applications and Attention Mechanisms

Activation Functions



Activation Functions



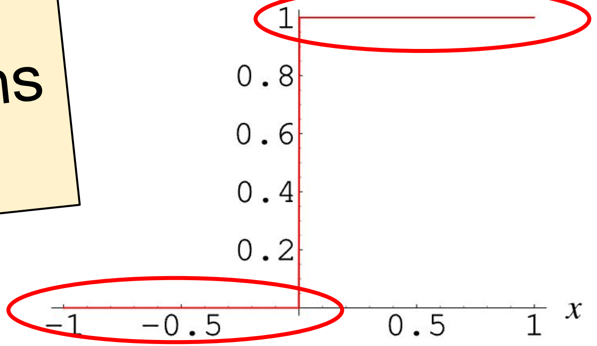
$= a(W_{1,1}\mathbf{x}_1 + W_{1,2}\mathbf{x}_2 + W_{1,3}\mathbf{x}_3 + W_{1,4}\mathbf{x}_4 + W_{1,5}\mathbf{x}_5 + \mathbf{b}_1)$

If we think about a "neuron" as inspiration, the output after *a* should be 1 if $W_{1,1}\mathbf{x}_1 + W_{1,2}\mathbf{x}_2 + W_{1,3}\mathbf{x}_3 + W_{1,4}\mathbf{x}_4 + W_{1,5}\mathbf{x}_5 + \mathbf{b}_1$ exceeds a threshold and 0 else.

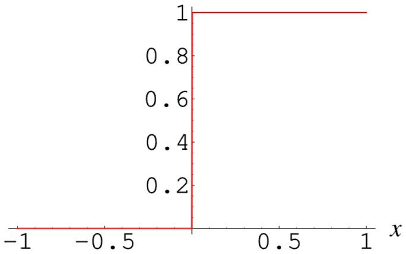
What's the "problem" with defining *a* this way?

To perform backpropagation, we need differentiable functions with non-zero gradients.

g., a(x) = {1 *if* x>0 *else* 0}

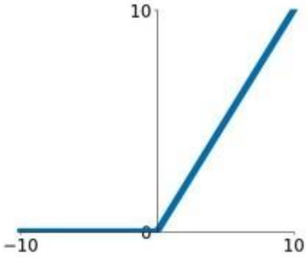


Activation Functions



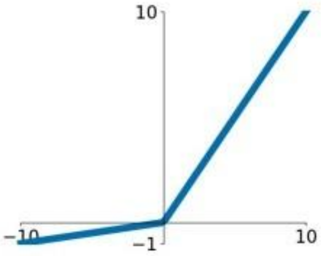
ReLU

$\max(0, x)$



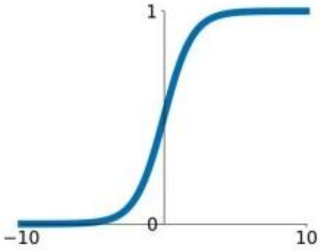
Leaky ReLU

$\max(0.1x, x)$



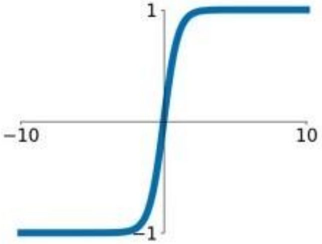
Sigmoid

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



tanh

$\tanh(x)$



$f: X \rightarrow Y;$

$f(x)=y$

$f: \phi_x(X) \rightarrow \phi_y(Y);$

$f(\mathbf{x})=\mathbf{y}$

$f: X \times \Theta \rightarrow Y;$

$f(x;\theta)=y;\ \theta \in \Theta$ for model M

$f: \phi_x(X) \times \Theta \rightarrow \phi_y(Y);$

$f(\mathbf{x};\theta)=\mathbf{y}$

$\theta^{*}=\min_{\theta \in \Theta}(\Sigma_{(x,y)} L(f(\mathbf{x};\theta), \mathbf{y})))$

$\theta^{*}=\min_{\theta \in \Theta}(\Sigma_{(x,y) \in D} L(f(\mathbf{x};\theta), \mathbf{y})))$

- DL enables function approximation for *f* and is constrained by:
  - Data representation ϕ
  - Model architecture M
  - Loss function L
  - Optimization algorithm
  - Training data D

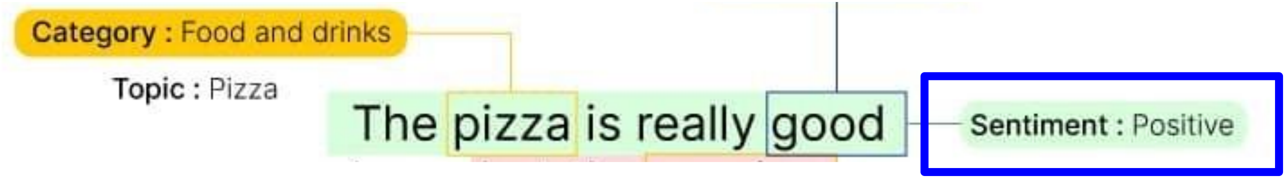
Video Topic Classification

→ Sports

- Input space X?
  - Images at every *timestep*
  - $\phi_x(X)=\boldsymbol{R}^{T,W,H,3}$
- Output space Y?
  - Output classes C
  - $X=C$

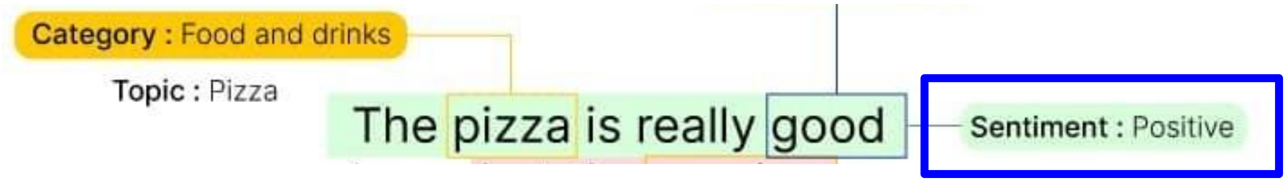
- Would a CNN make sense here?
  - 3D CNN
- Where would filters that span the time dimension fail?
- What if we processed every frame with a 2D CNN?
  - Would need some method to aggregate through time.

Text Classification



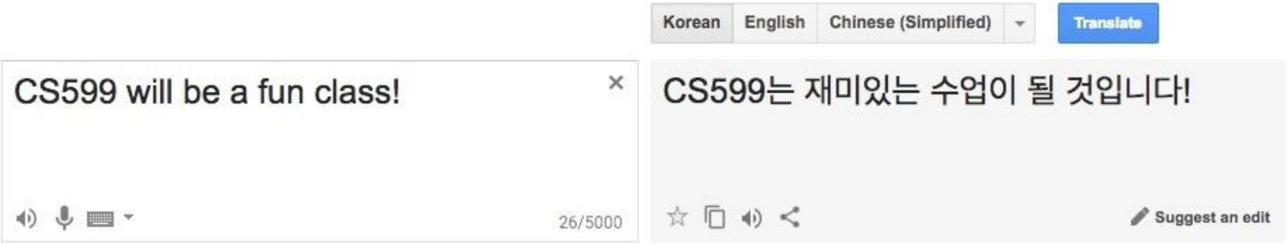
- Input space X?
  - Review *tokens* T
  - $X = T^N$ for max length N
- Output space Y?
  - Possible sentiments S
  - $Y = S$
- Classification via a multi-layer perceptron:
  - Represent **x** as a multi-hot "bag of words" vector that tells us which words are in the review and which aren't
- Weaknesses of an MLP model?
  - No ordering information
  - "cold ice cream and warm pizza"
  - "warm ice cream and cold pizza"

Text Classification



- Input space X?
  - Review *tokens* T
  - $X=T^N$ for max length N
- Output space Y?
  - Possible sentiments S
  - $Y=S$
- Classification via a CNN
  - Represent **x** as a sequence of token embedding vectors
  - Filters can combine words close together
- Weaknesses?
  - Words may have long-range dependencies that are hard to see with CNNs that favor neighborhood information
  - E.g., Main verb of “The horse raced past the barn fell”?

Machine Translation



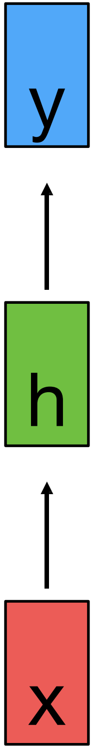
- Input space *X*?
  - Input *tokens T*
  - $X=T^N$ for max length N
- Output space *Y*?
  - Output *tokens V*
  - $X=V^N$ for max length M

- Would an MLP even make sense here?
  - What would our output classification be?
- Would a CNN even make sense here?
  - What might the output architecture look like?
- Probably we need an entirely different approach!

Classes of Sequence-to-Sequence Problems

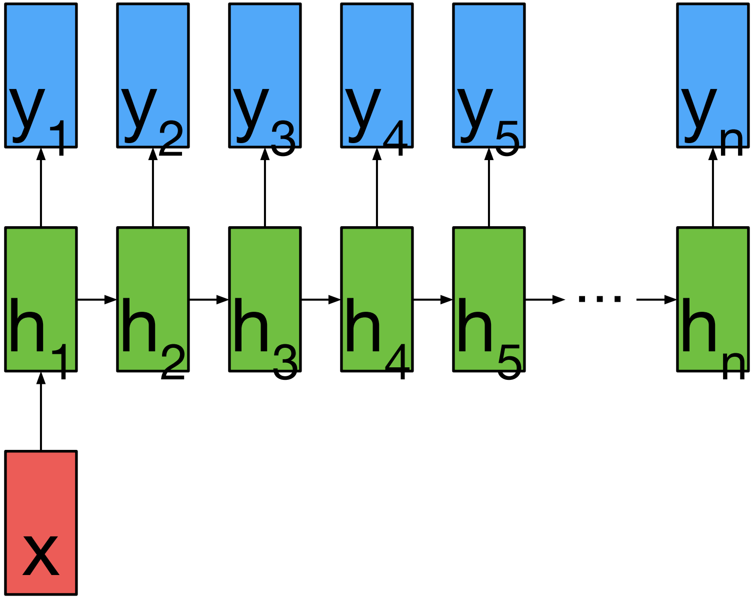
- One-to-one
  - One input produces one output
    - Image classification

Person



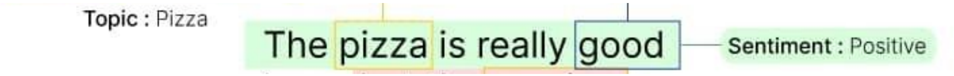
Classes of Sequence-to-Sequence Problems

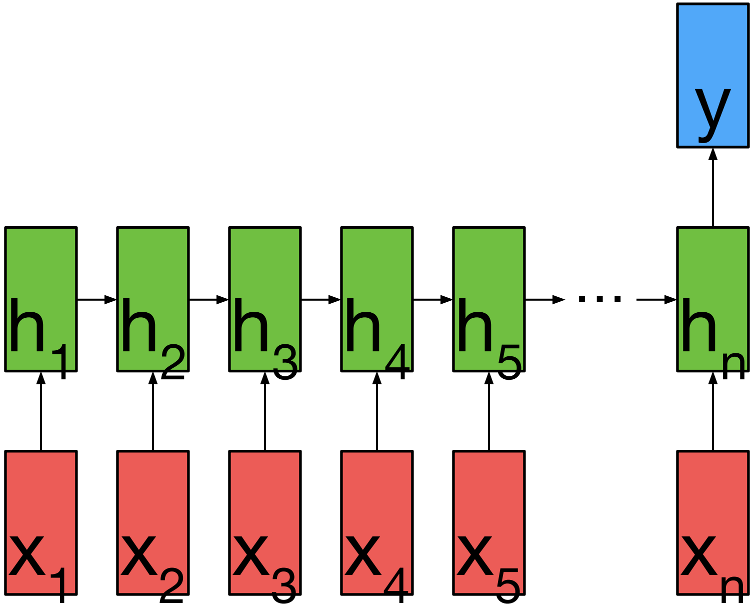
- One-to-many
  - One input produces a sequence of outputs
    - Image captioning



Classes of Sequence-to-Sequence Problems

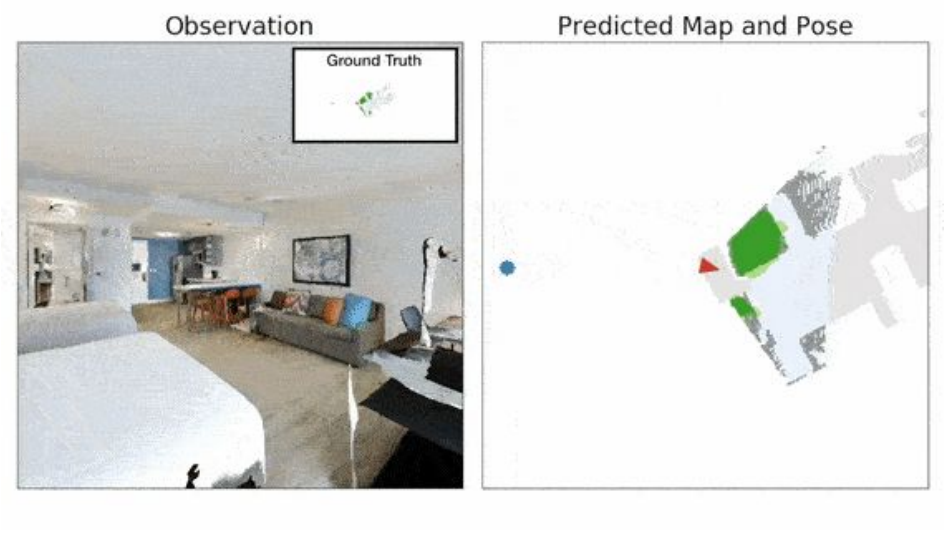
- Many-to-one
  - Sequence of inputs produces a single output
    - Text classification

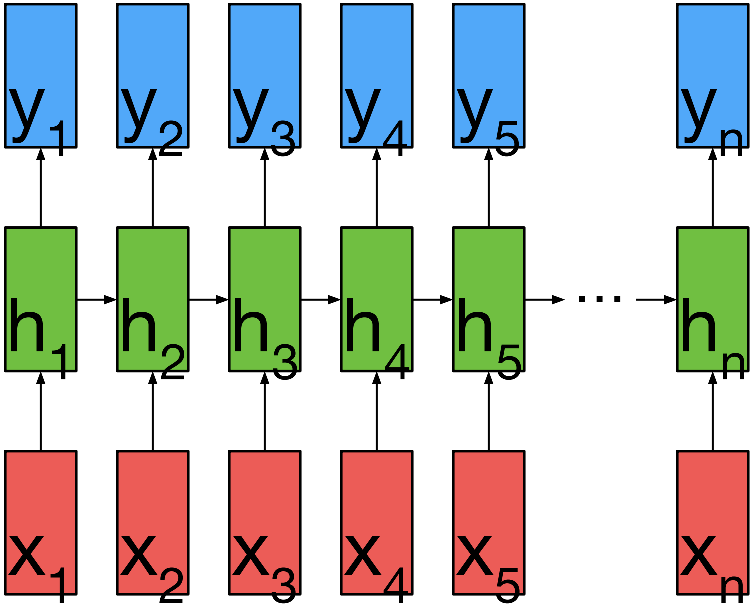




Classes of Sequence-to-Sequence Problems

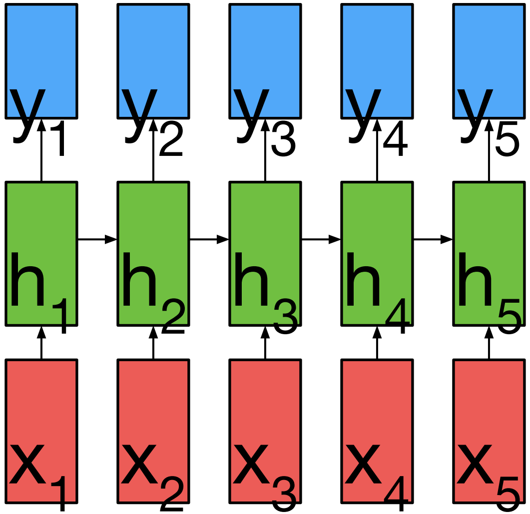
- Many-to-many
  - A sequence of inputs produces a sequence of outputs
    - Robot Actions



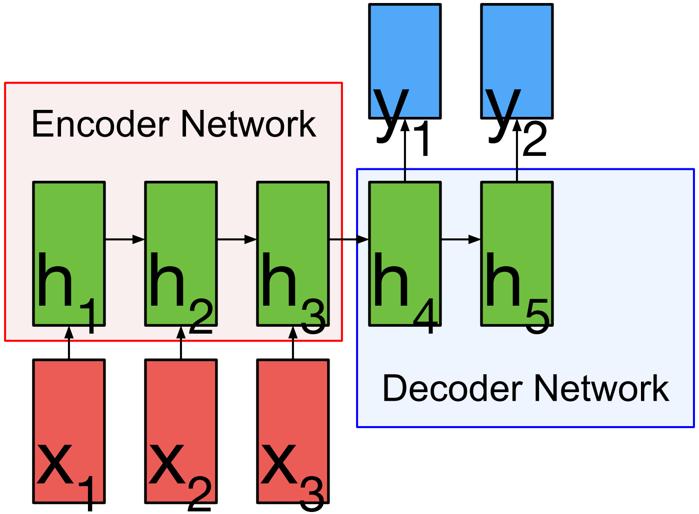


Classes of Sequence-to-Sequence Problems

- Many-to-many variants:
  - Sequential one-to-one
    - Robot actions

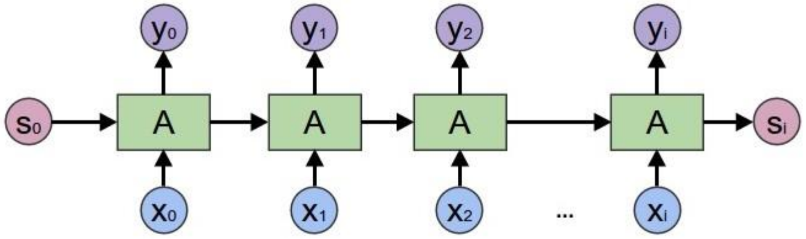


  - Encoder-decoder
    - Machine Translation



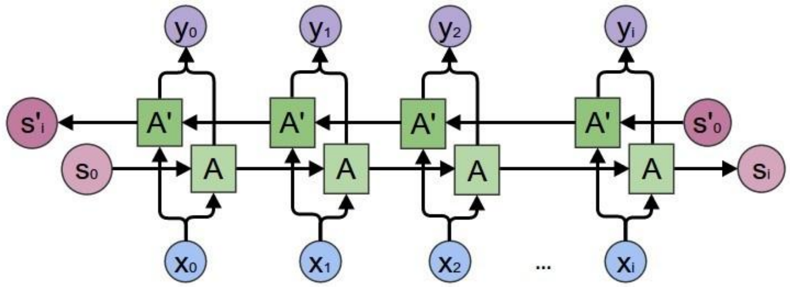
Unidirectional and Bidirectional Seq2Seq Models

Forward RNN



- Output y_t depends only on inputs so far $x_{0:t-1}$
  - E.g., robot actions

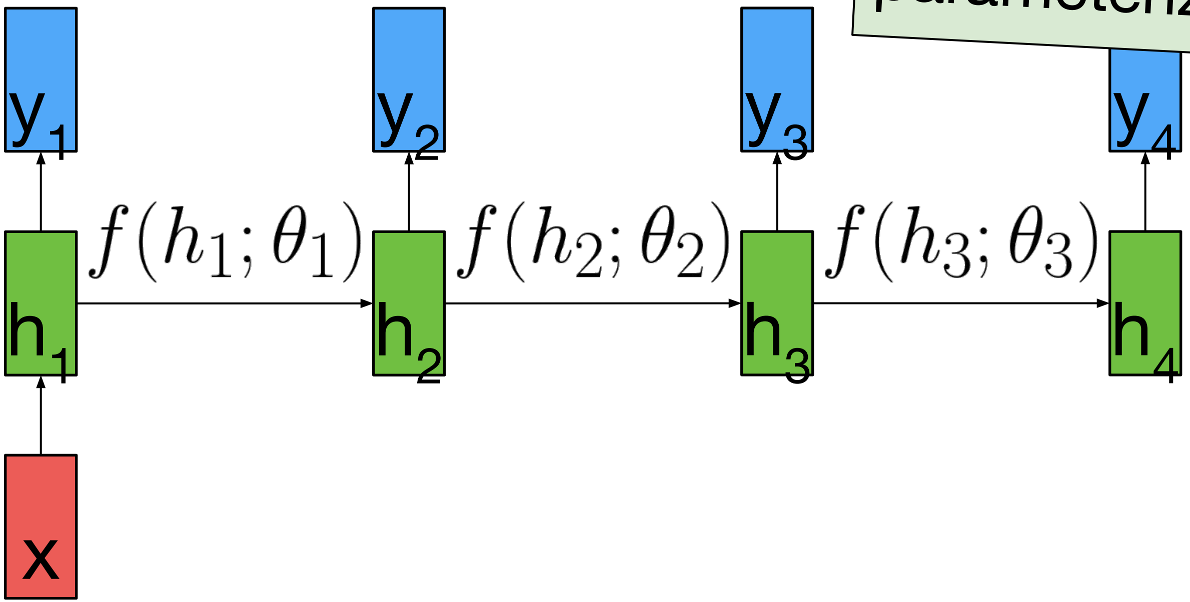
Bidirectional RNN



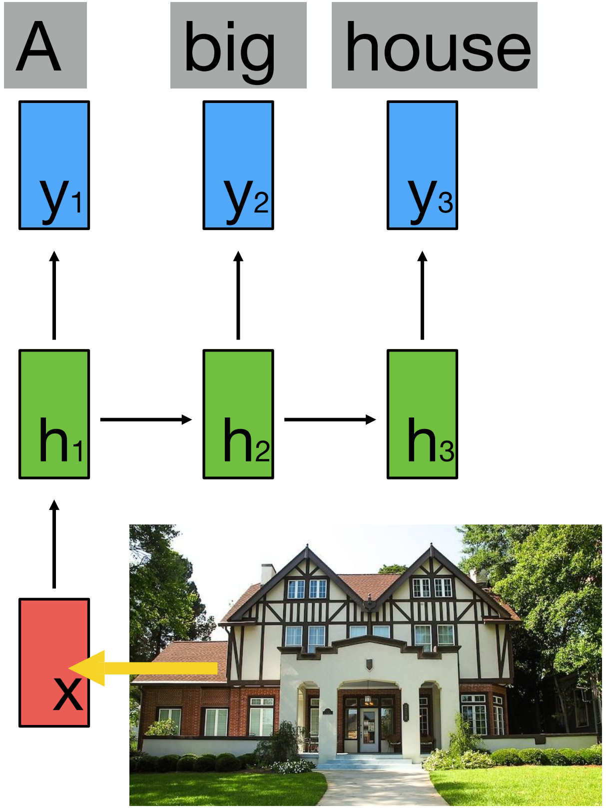
- Output y_t depends on all inputs $x_{0:L}$
  - E.g., machine translation

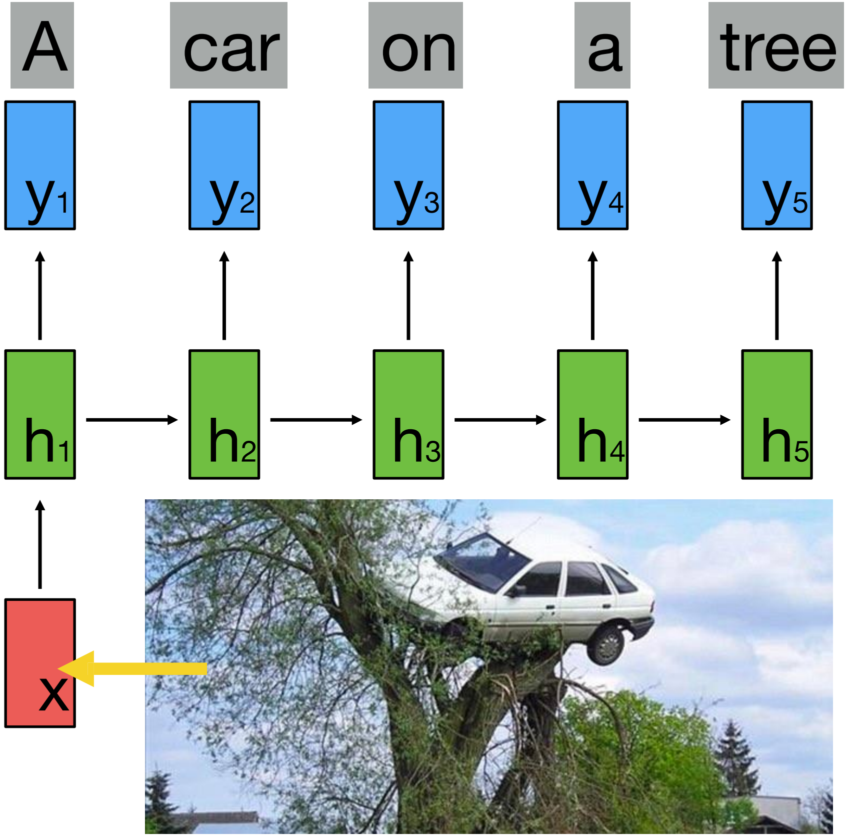
Sequence Modeling with Learned Neural Functions

What's a weakness of this parameterization?

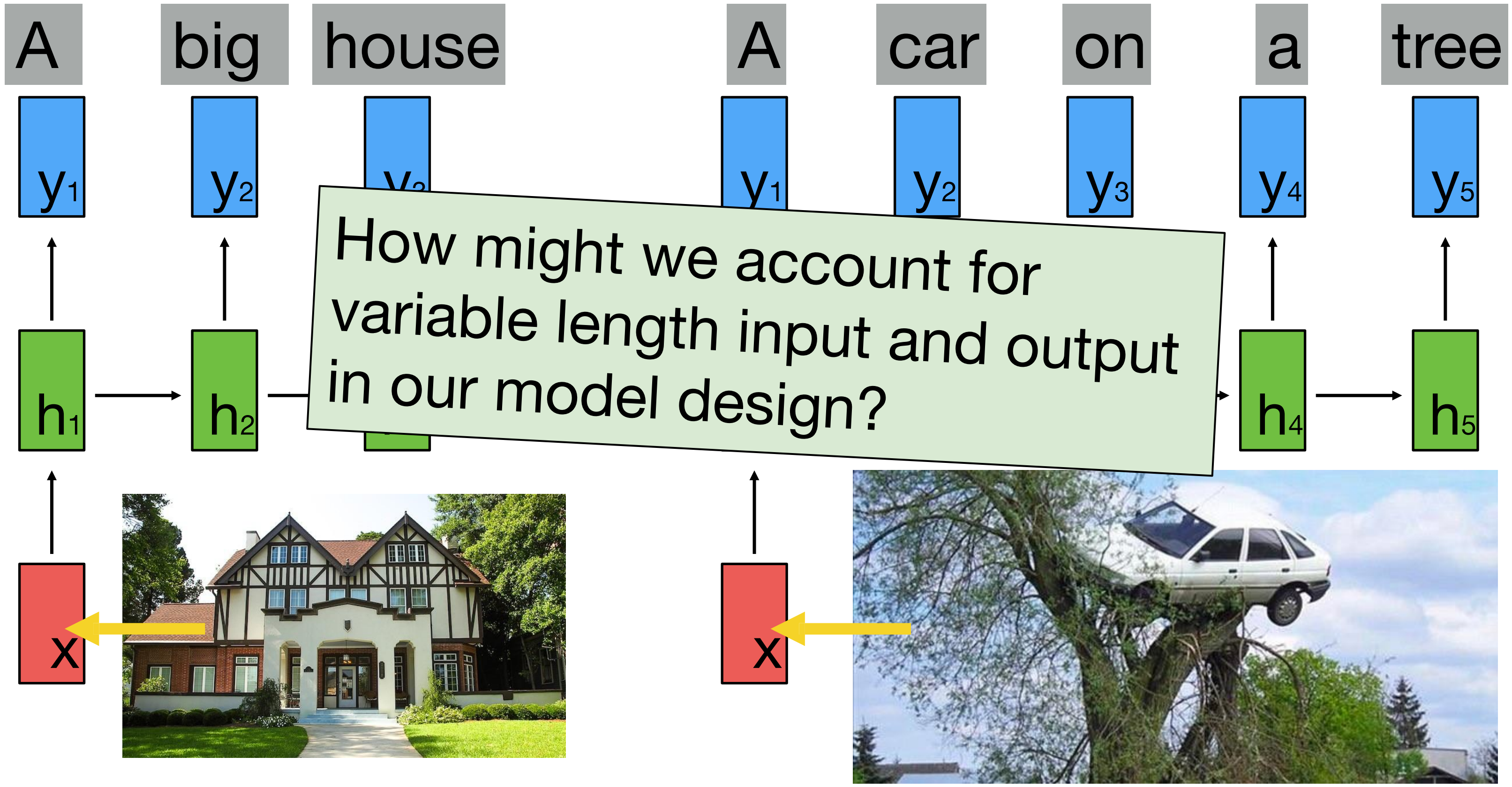


Sequence Modeling with Learned Neural Functions



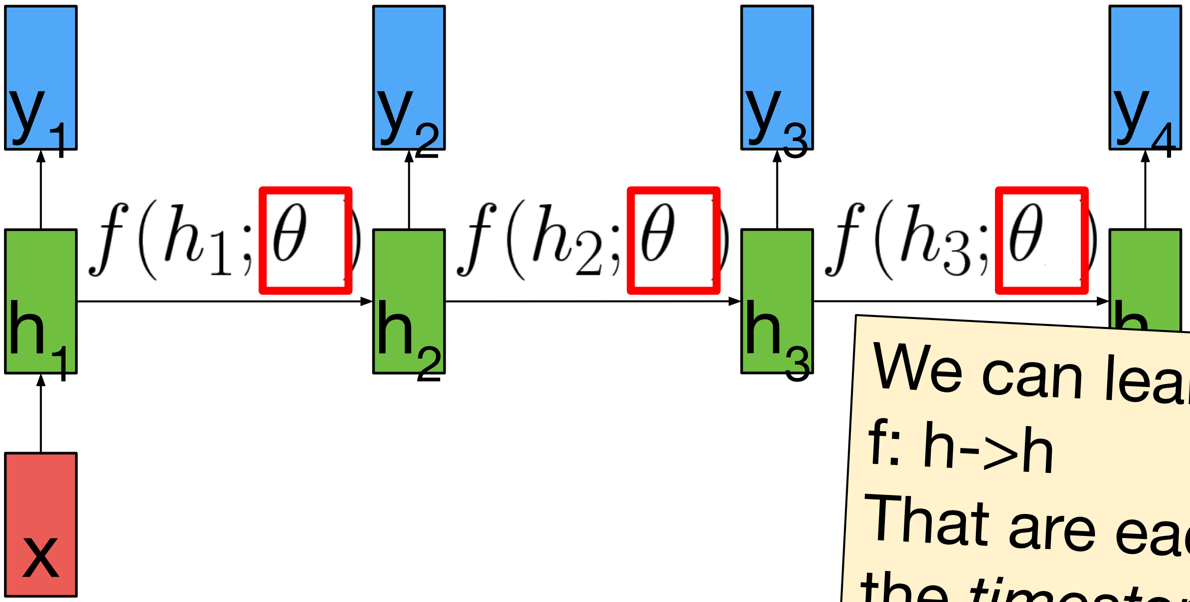


Sequence Modeling with Learned Neural Functions



How might we account for variable length input and output in our model design?

Sequence Modeling with Learned Neural Functions



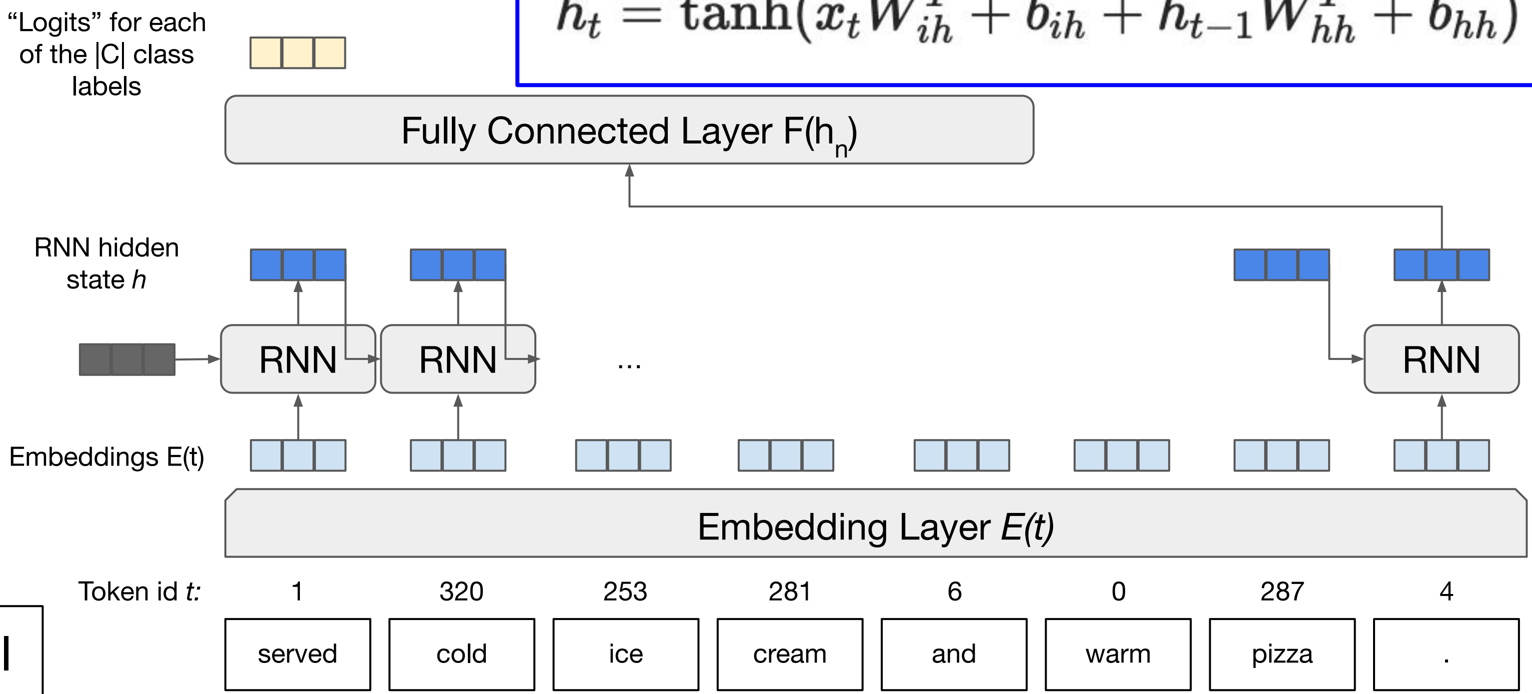
We can learn functions:
f: h->h   f: x->h   f: h->y
That are each independent of the *timestep* dimension.

Recurrent Neural Networks

- We encode **one input at a time** to iteratively build up our representation of the sequence
- After each new input, we produce a new *hidden state* from which we could decode outputs (e.g., many-to-many)
- For classification (e.g., many-to-one), we can decode from the final *hidden state* that contains information about the whole input sequence
- We only need to *learn* how to map inputs to hidden states, hidden states to outputs, and hidden states to hidden states

Text Classification with a Recurrent Neural Network

$$h_t = \tanh(x_t W_{ih}^T + b_{ih} + h_{t-1} W_{hh}^T + b_{hh})$$

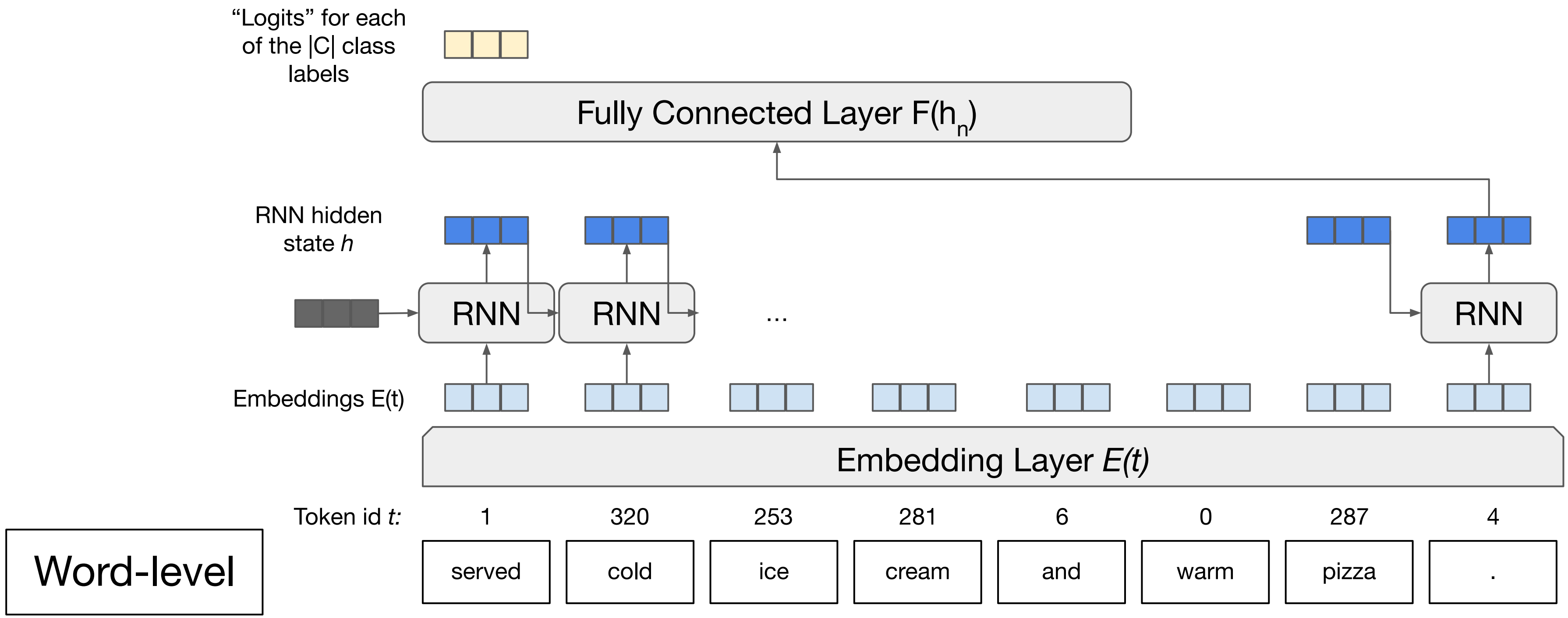


Word-level

NLP FUNDAMENTALS: Word Embeddings

- A word embedding is a fixed-length vector representation of a given string, such as *“dog”*; we’ll say length h
- Word embedding vectors stacked together as rows form a matrix of $E^{|V|*h}$ for V the set of words the model can understand as input
- That *embedding matrix* is differentiable and can be learned as part of our neural network architecture!
- We will cover word embeddings in more detail in Module 2

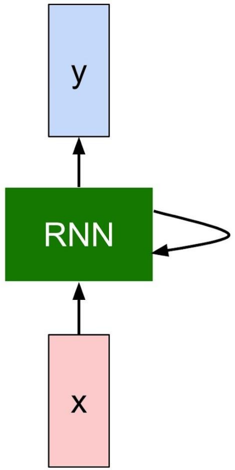
Text Classification with a Recurrent Neural Network



Recurrence In Neural Networks

- We can process a sequence of inputs one input at a time by defining a *recurrence function*
- Recurrence function takes two arguments: the next input and a *state* representing aggregated information from past inputs

$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

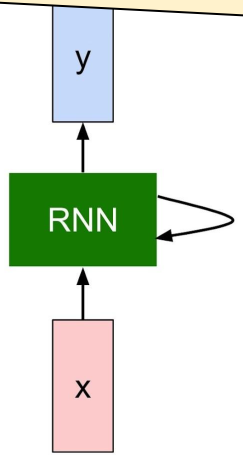


Recurrence In Neural Networks

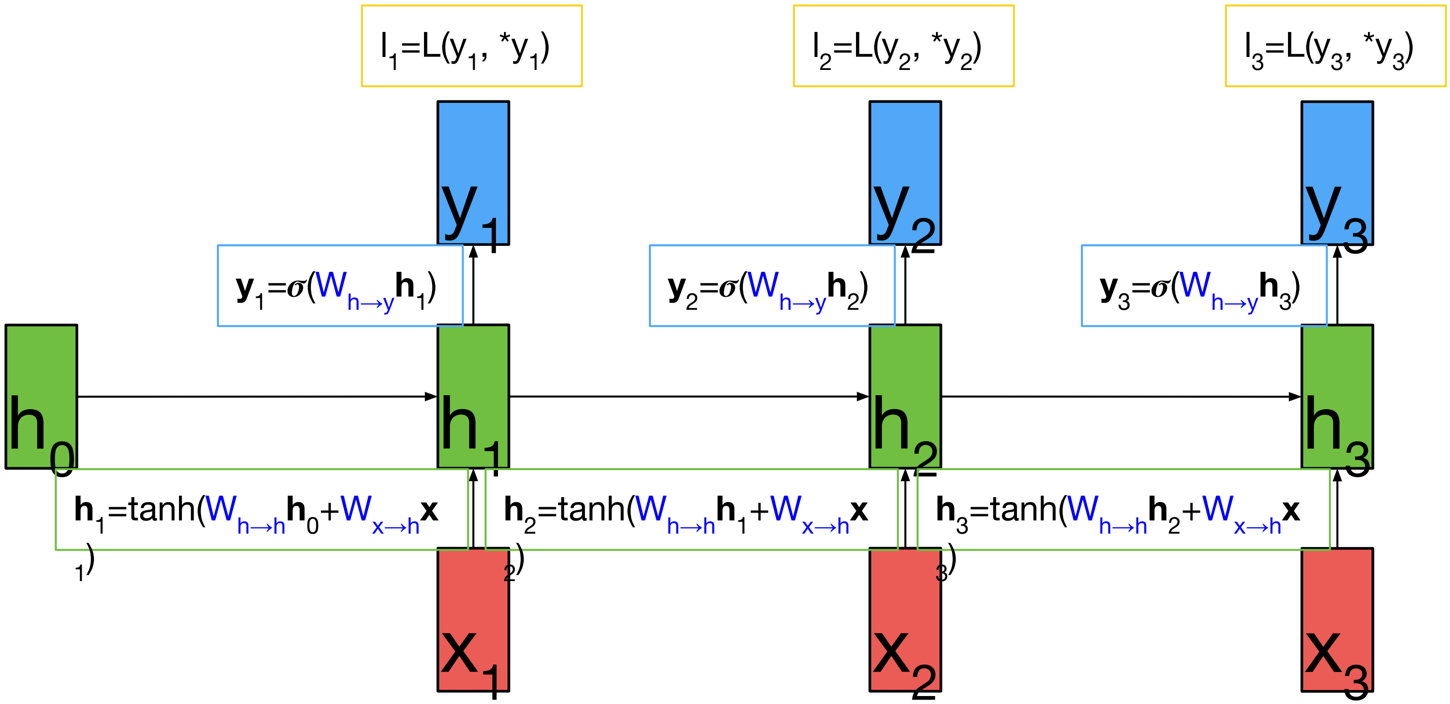
- Remember how CNN filters were just little linear layers?
- Same story for how we define f_W!
  - Vanilla RNN:
    - $\mathbf{h}_t = \tanh(W_{h\to h}\mathbf{h}_{t-1} + W_{x\to h}\mathbf{x}_t)$
    - $\mathbf{y}_t = \sigma(W_{h\to y}\mathbf{h}_t)$

Note that to compress notation we frequently drop bias vectors.

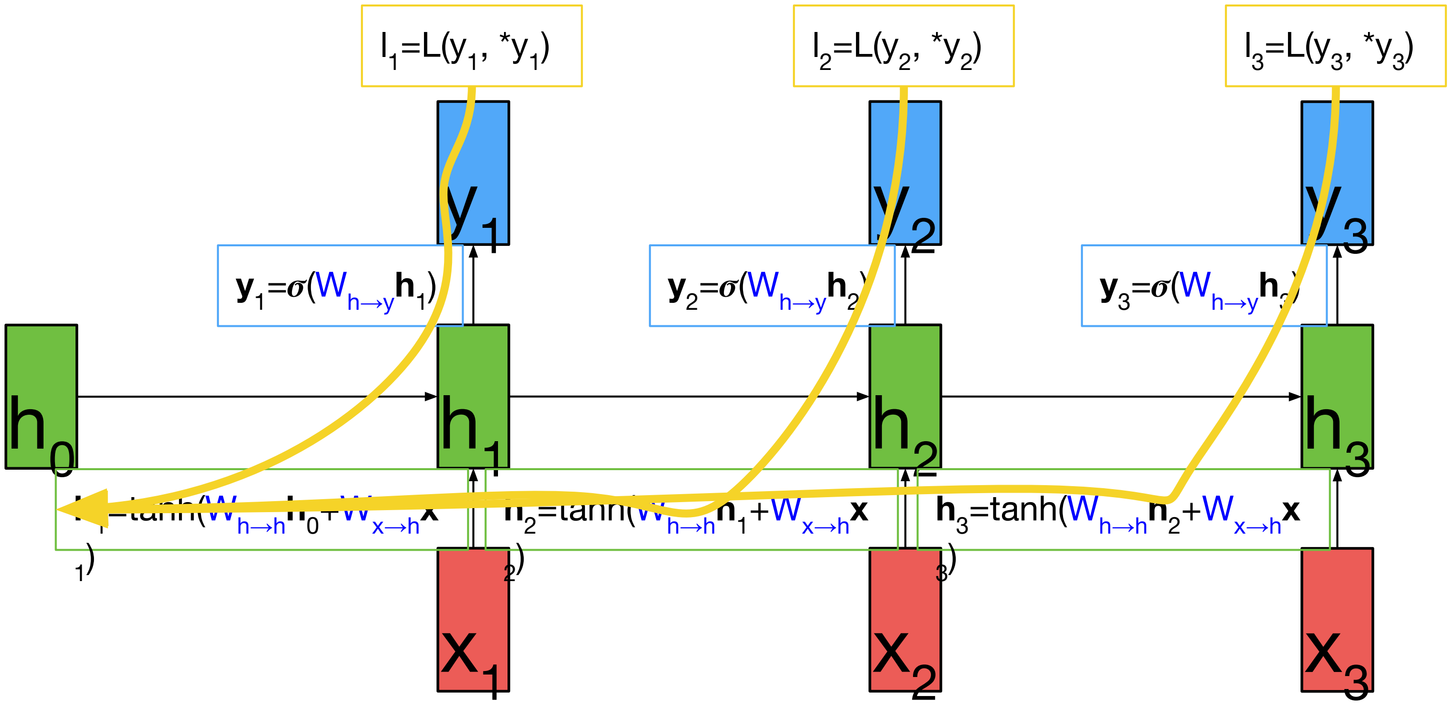
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$



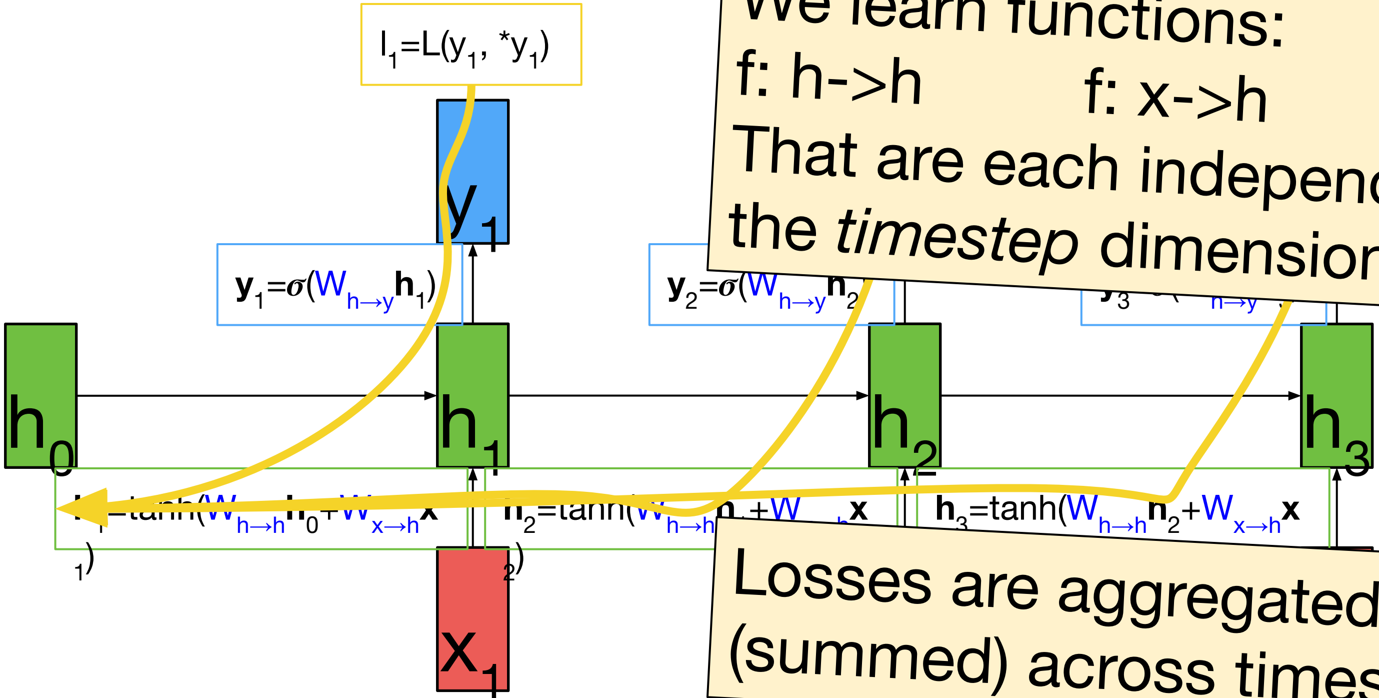
High Level Computation Graph for RNN



Backpropagation for RNN



Backpropagation for RNN



We learn functions:
f: h->h   f: x->h   f: h->y
That are each independent of the *timestep* dimension.

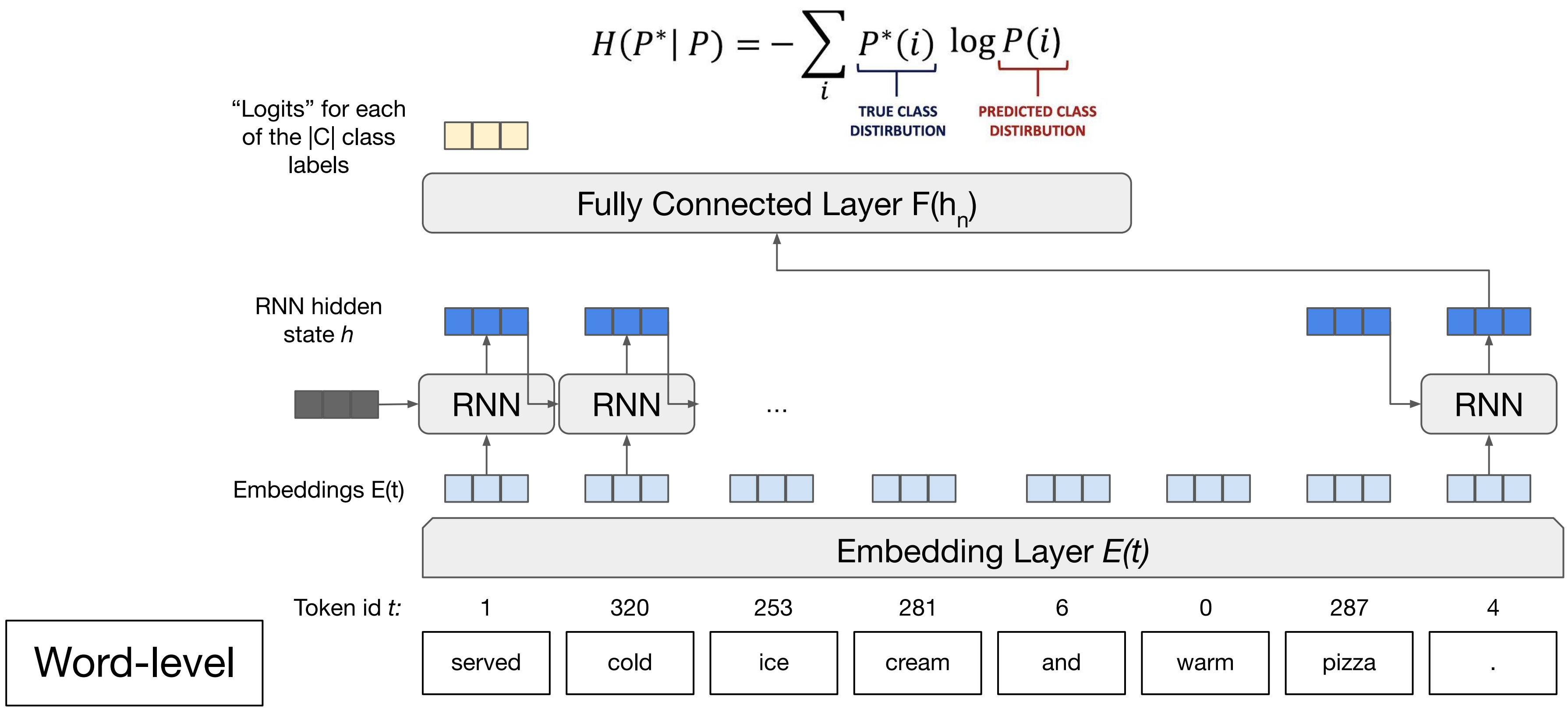
Losses are aggregated (summed) across timesteps.

Backpropagation for RNNs

- But we… don’t use RNNs?
- I know Transformers are all the rage, but even when RNNs were common we stopped using vanilla RNNs. Why?
- Let’s look at our gradient step again:
  - $\theta := \theta - \varepsilon \nabla_{di \ldots dj} L[M(\phi(d), \theta)]$
  - Need the derivative of, in this example, cross entropy
- So L is $H(P^*|P)$
- $P(i) = M(\phi(d), \theta)$
- *M* is our fancy new RNN

$$H(P^* | P) = -\sum_i \underbrace{P^*(i)}_{\text{TRUE CLASS DISTRIBUTION}} \log \underbrace{P(i)}_{\text{PREDICTED CLASS DISTRIBUTION}}$$

Text Classification with a Recurrent Neural Network



The Slide Where RNNs Lose

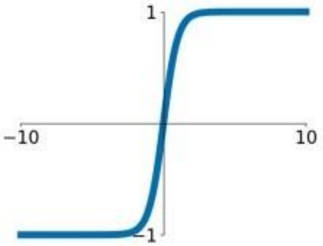
- $P(i) = M(\phi(d), \theta)$; R the RNN layer we're learning
  - … $E(t_1) \ldots E(t_L)$ ….
  - … $R(E(t_1), 0)$ …
  - … $R(E(t_2), R(E(t_1), 0))$ …
  - … $R(E(t_L), \ldots R(E(t_4), R(E(t_3), R(E(t_2), R(E(t_1), 0)))) \ldots)$ …
  - $P(i) = F(R(E(t_L), \ldots R(E(t_4), R(E(t_3), R(E(t_2), R(E(t_1), 0)))) \ldots))$
- Just thinking about the *size* of ∇, it'll be biggest at *F*
- For every nested call to *R*, the gradient *vanishes* further; remember that we're non-linearly squeezing with each *R* too

https://en.wikipedia.org/wiki/Vanishing_gradient_problem

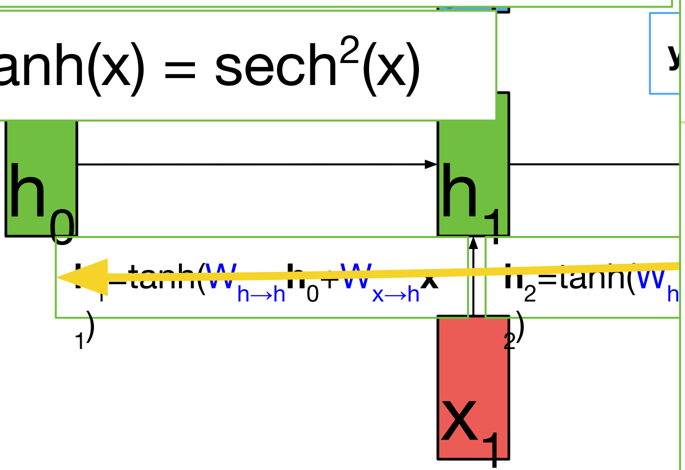
The Vanishing Gradient Problem

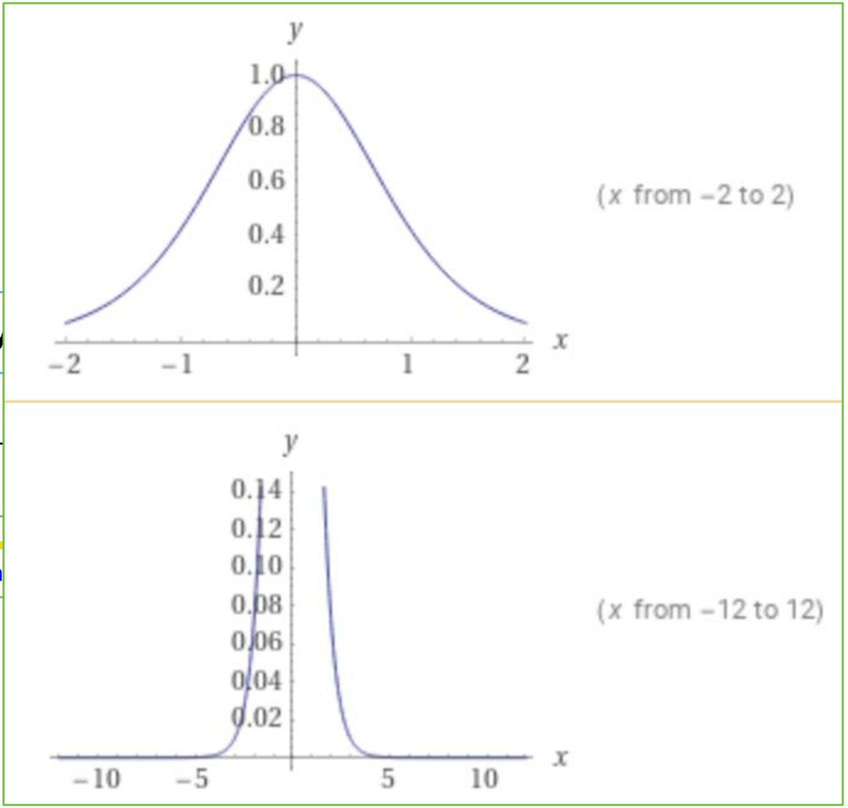
tanh

$\tanh(x)$

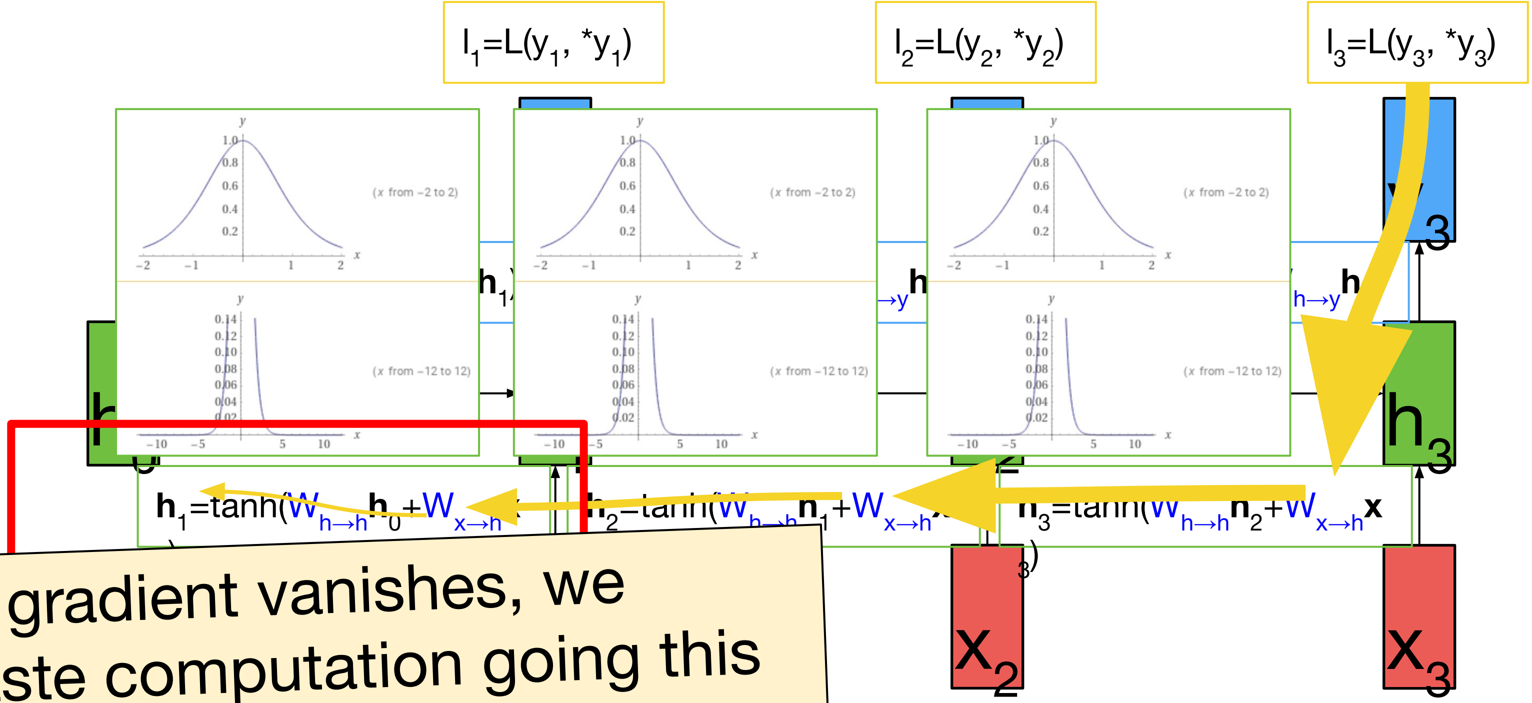


$\frac{d}{dx}\tanh(x) = \mathrm{sech}^2(x)$



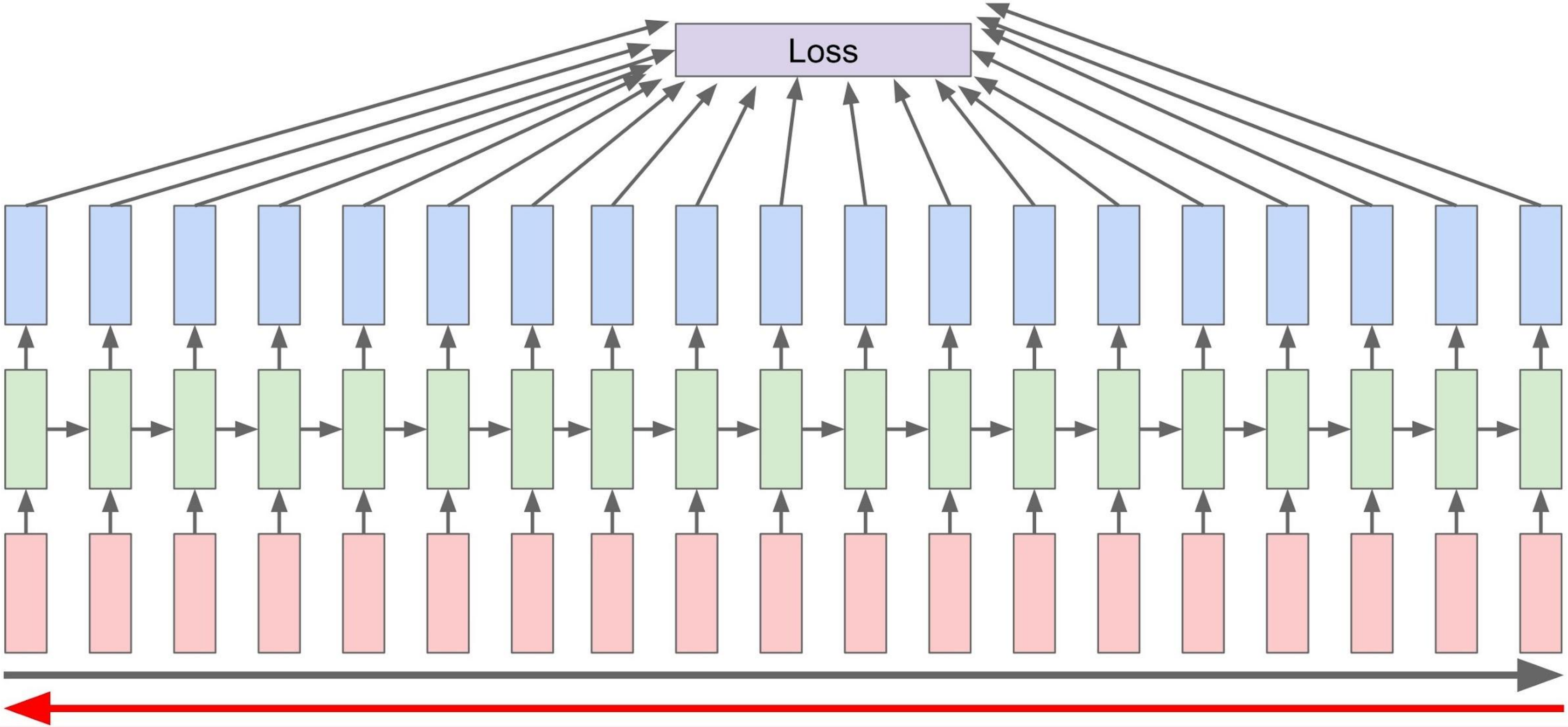


The Vanishing Gradient Problem

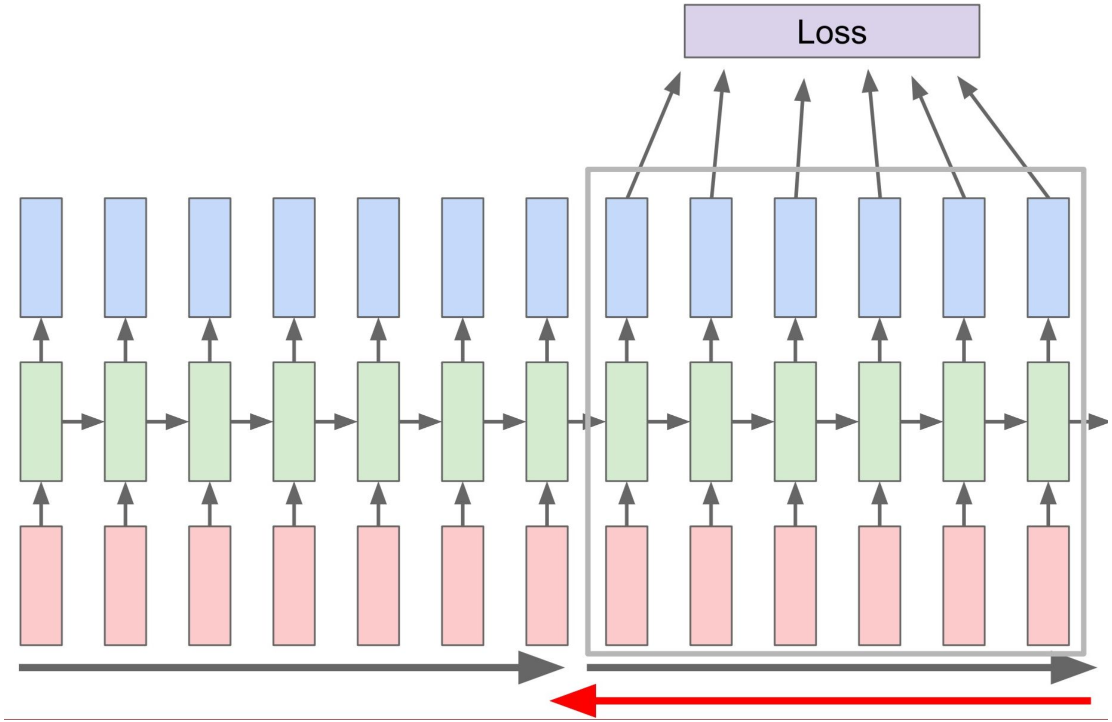


As gradient vanishes, we waste computation going this far back in time…

The Vanishing Gradient Problem

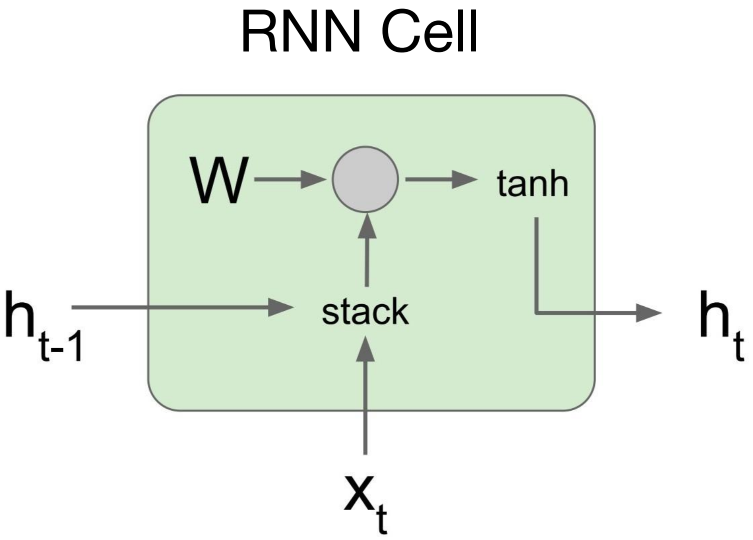


Truncated Backpropagation



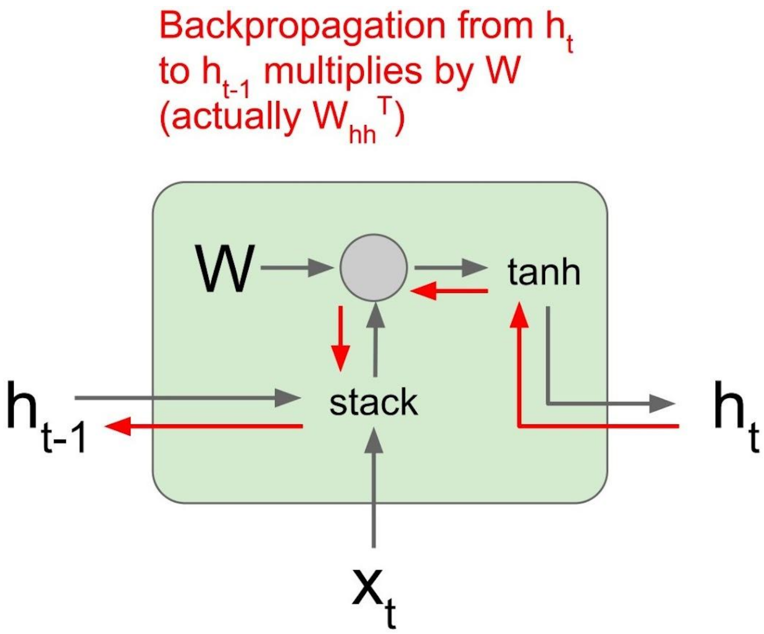
- Run forward and backward through chunks of the sequence instead of the whole sequence
- Carry hidden states forward through chunks
- Truncate backprop after a few steps

The Vanishing Gradient Problem



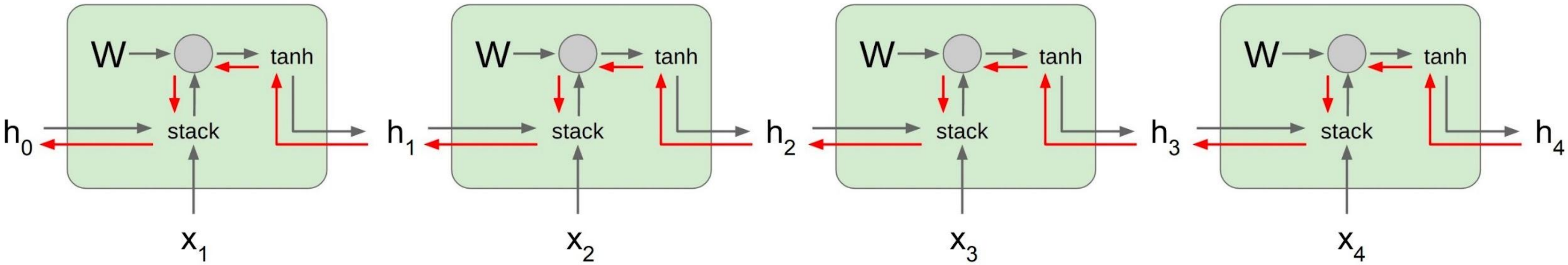
$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
&= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\
&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)
\end{aligned}
$$

The Vanishing Gradient Problem



$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
&= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\
&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)
\end{aligned}
$$

The Vanishing Gradient Problem



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Gradient clipping: scale gradient if its norm is too big

To not squash downstream gradients, we need a new architecture.

Overview of Today's Plan

- ~~Course organization and deliverables~~
- ~~Recurrent Neural Networks~~
    - Any questions before we move on?
- Long-Short Term Memory
- Applications and Attention Mechanisms

Long short-term memory

- A dumb name for some complex cell architecture
  - The details of LSTMs have taken a backseat in NLP lately
  - Upshot: preserve more gradient by keeping more of the hidden state around between updates
- Gated Recurrent Units (GRU) are in the same family of attempts to get around this issue with sequence problems
- In many settings, the lower parameter count of LSTMs and GRUs makes them preferable to Transformers
- So we will still learn LSTMs!

https://en.wikipedia.org/wiki/Vanishing_gradient_problem

Long short-term memory

- Intuition of LSTMs is to "flip" the default recurrence behavior from "squeeze out the history" to "mostly consider the history"

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation 1997

Long short-term memory

LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

$$i = \sigma(W_i \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix})$$

$$f = \sigma(W_f \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix})$$

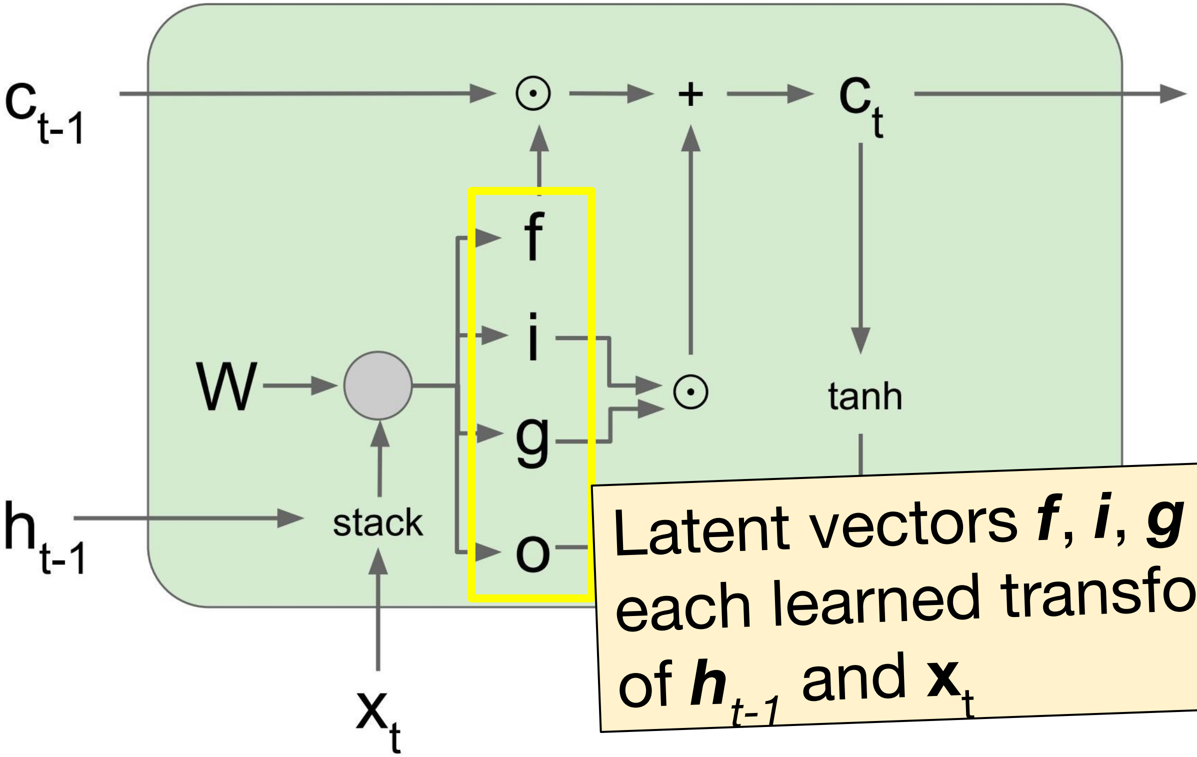
$$o = \sigma(W_o \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix})$$

$$g = tanh(W_g \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix})$$

Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997

Long short-term memory

LSTM Cell



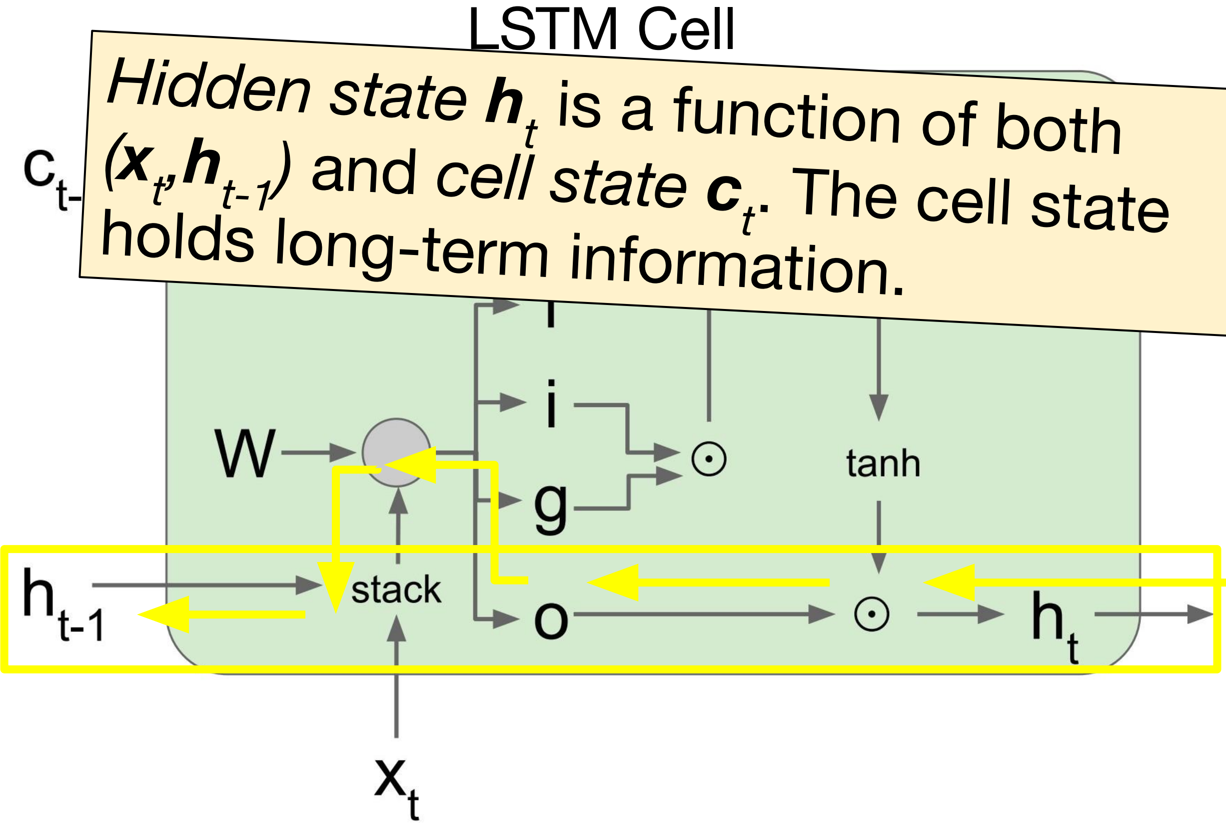
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$\tanh(c_t)$$

Latent vectors ***f***, ***i***, ***g*** and ***o*** are each learned transformations of $\boldsymbol{h}_{t-1}$ and $\mathbf{x}_t$

Long short-term memory



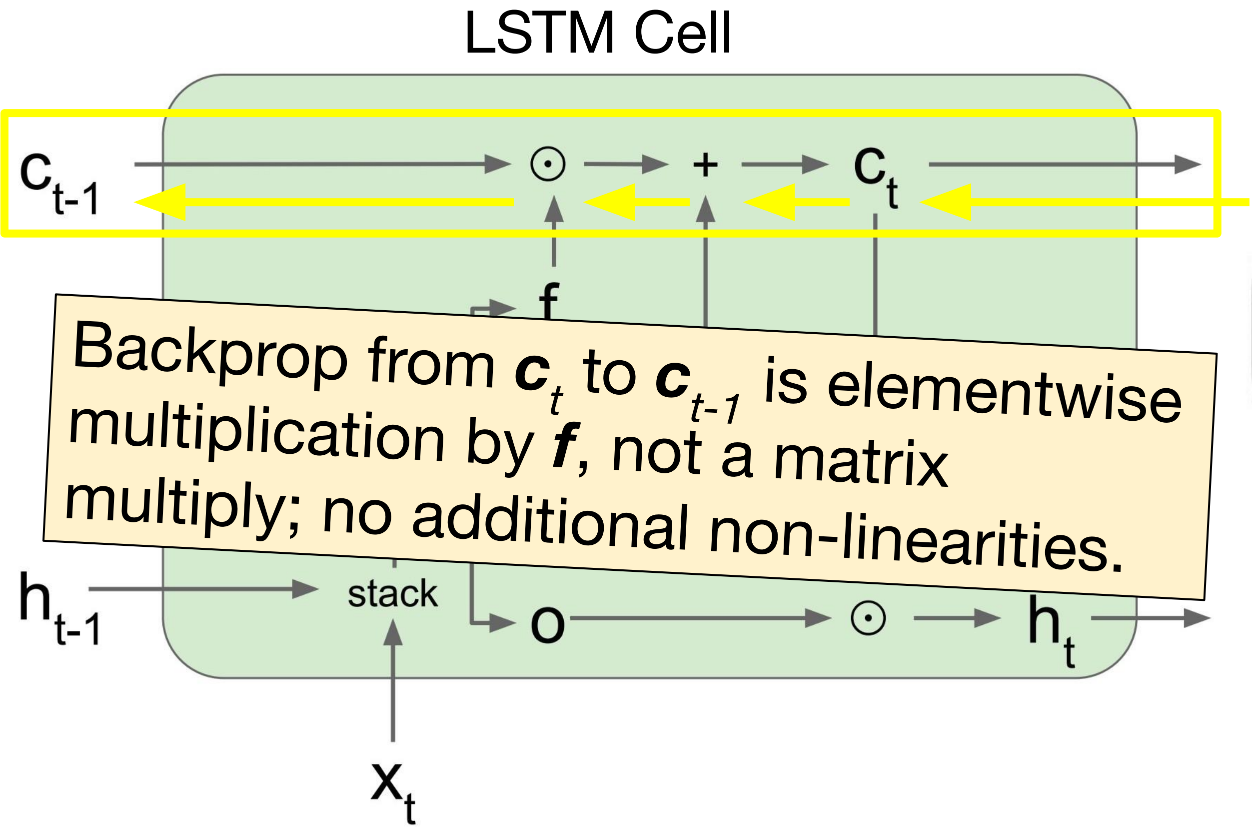
Hidden state $\boldsymbol{h}_t$ is a function of both ($\boldsymbol{x}_t$,$\boldsymbol{h}_{t-1}$) and *cell state* $\boldsymbol{c}_t$. The cell state holds long-term information.

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$\boxed{h_t = o \odot \tanh(c_t)}$$

Long short-term memory



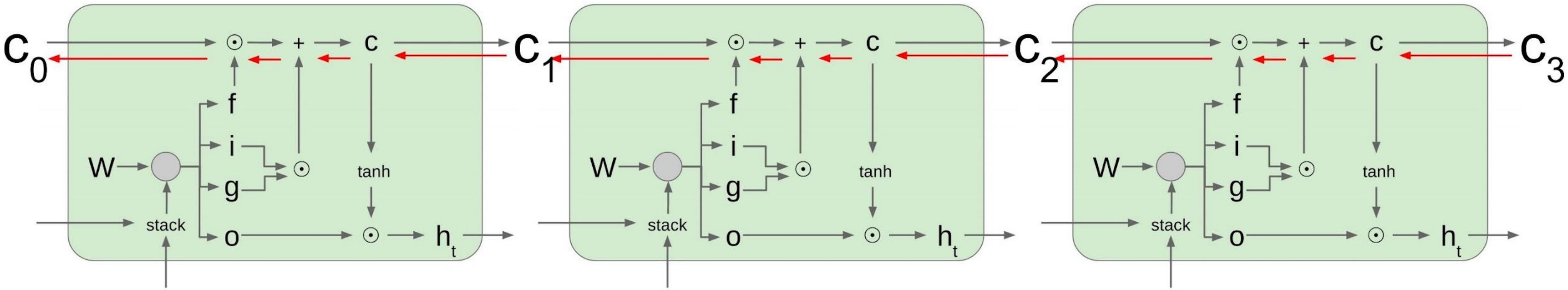
Backprop from $\boldsymbol{c}_t$ to $\boldsymbol{c}_{t-1}$ is elementwise multiplication by $\boldsymbol{f}$, not a matrix multiply; no additional non-linearities.

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$\boxed{c_t = f \odot c_{t-1} + i \odot g}$$

$$h_t = o \odot \tanh(c_t)$$

Long short-term memory

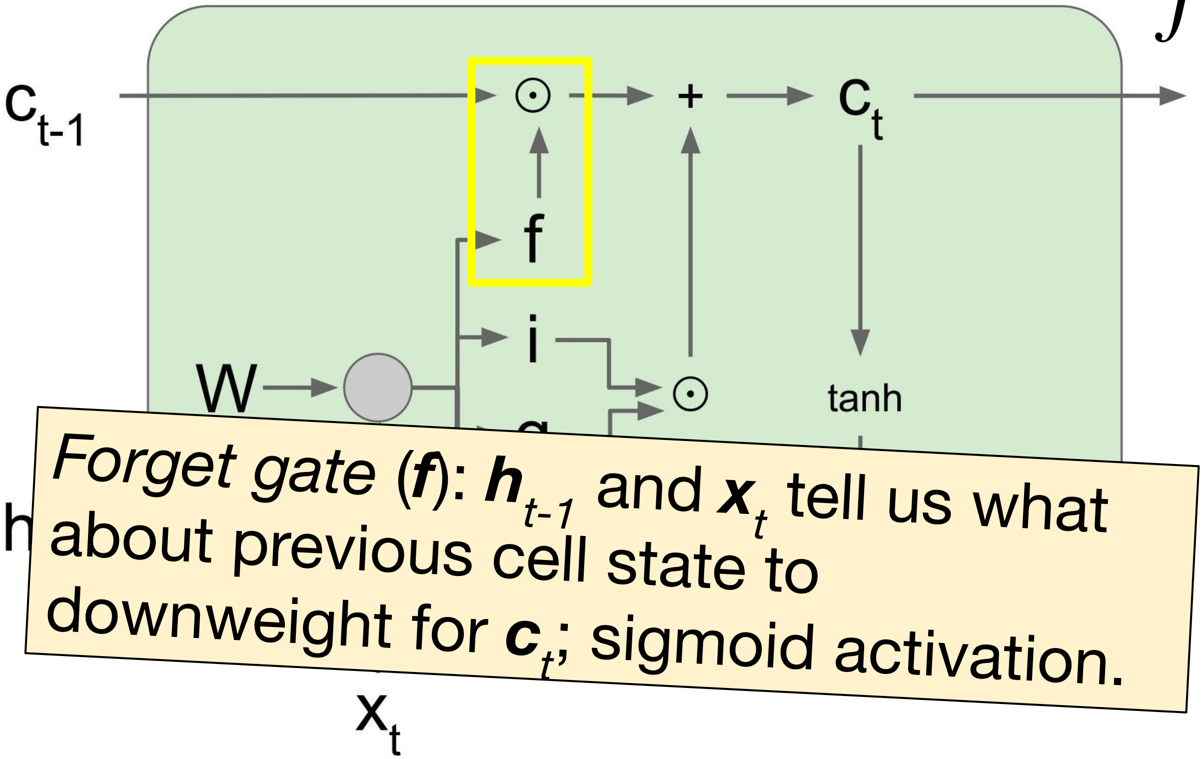


$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long short-term memory

LSTM Cell



*Forget gate (**f**): **h***

$$f = \sigma\left(W_f \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix}\right)$$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

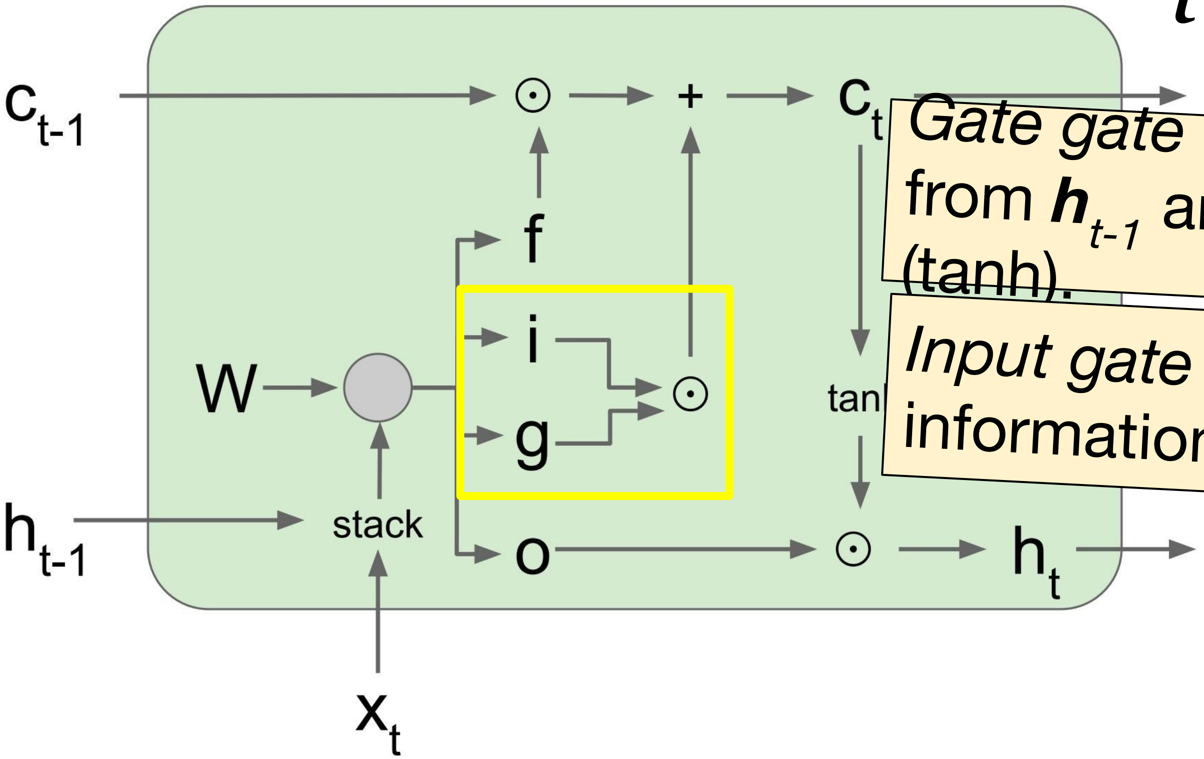
$$h_t = o \odot \tanh(c_t)$$

Long short-term memory

$$g = tanh(W_g \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix})$$

$$\boldsymbol{i} = \sigma(W_i \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix})$$

LSTM Cell



Gate gate (*g*): What information from $\boldsymbol{h}_{t-1}$ and $\boldsymbol{x}_t$ to add to $\boldsymbol{c}_t$ (tanh).

Input gate (*i*): How much information to add to $\boldsymbol{c}_t$ (sigmoid).

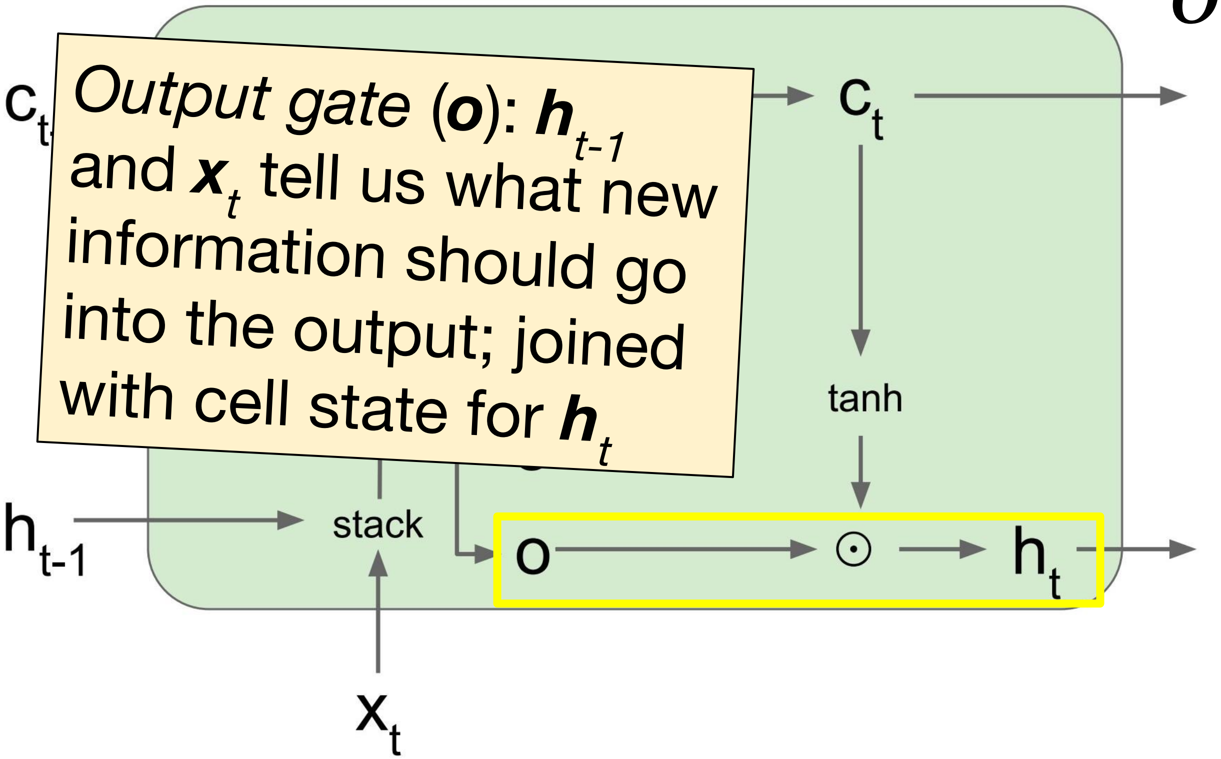
$$\begin{pmatrix} o \\ g \end{pmatrix} \quad \begin{pmatrix} \sigma \\ \tanh \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long short-term memory

LSTM Cell



Output gate (***o***): $\boldsymbol{h}_{t\text{-}1}$ and $\boldsymbol{x}_t$ tell us what new information should go into the output; joined with cell state for $\boldsymbol{h}_t$

$$o = \sigma\left(W_o \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix}\right)$$

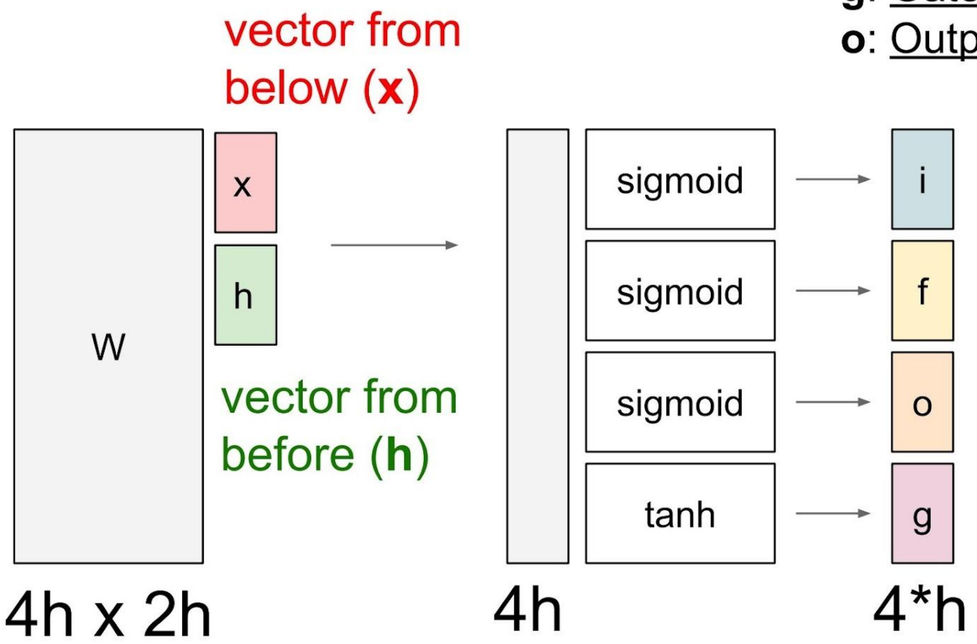
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long short-term memory

f: Forget gate, Whether to erase cell
i: Input gate, whether to write to cell
g: Gate gate (?), How much to write to cell
o: Output gate, How much to reveal cell

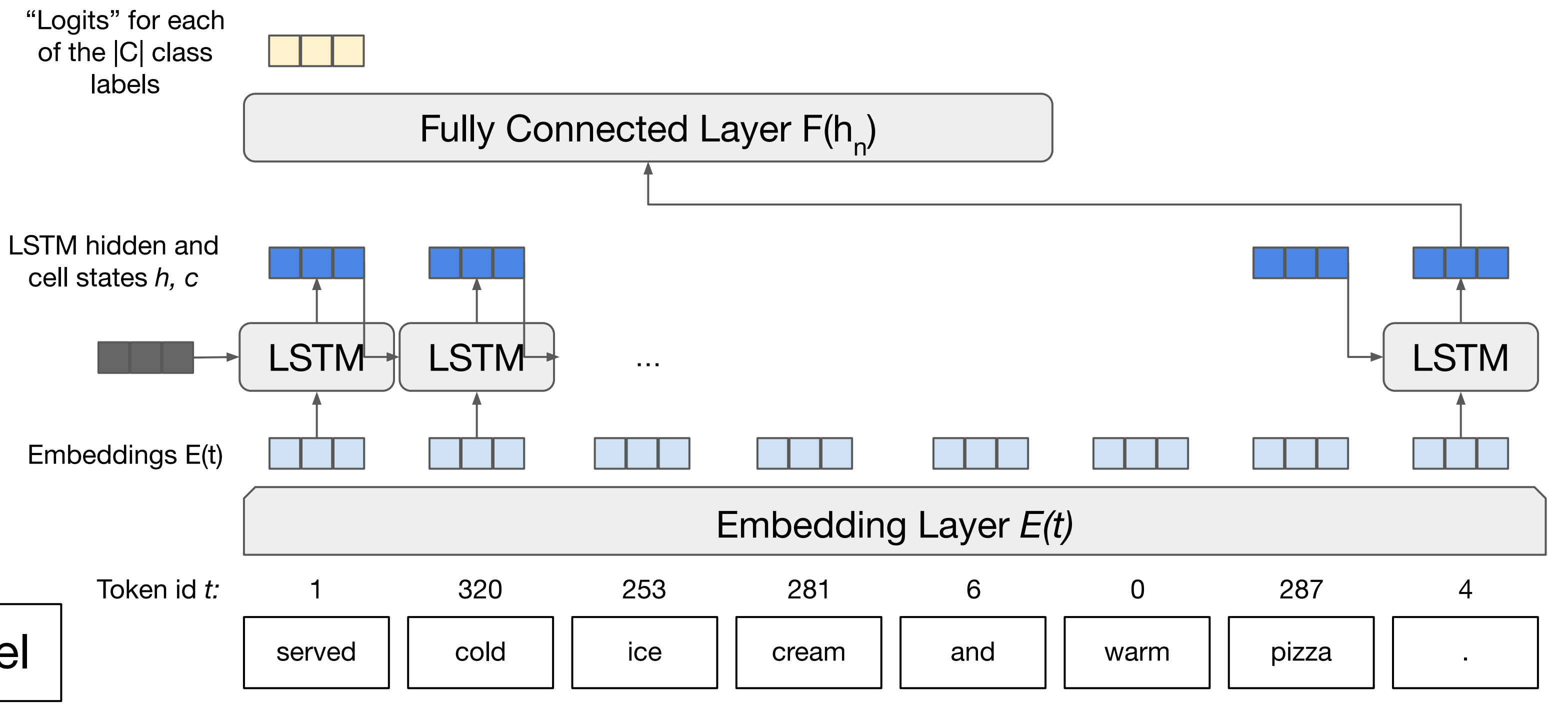


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

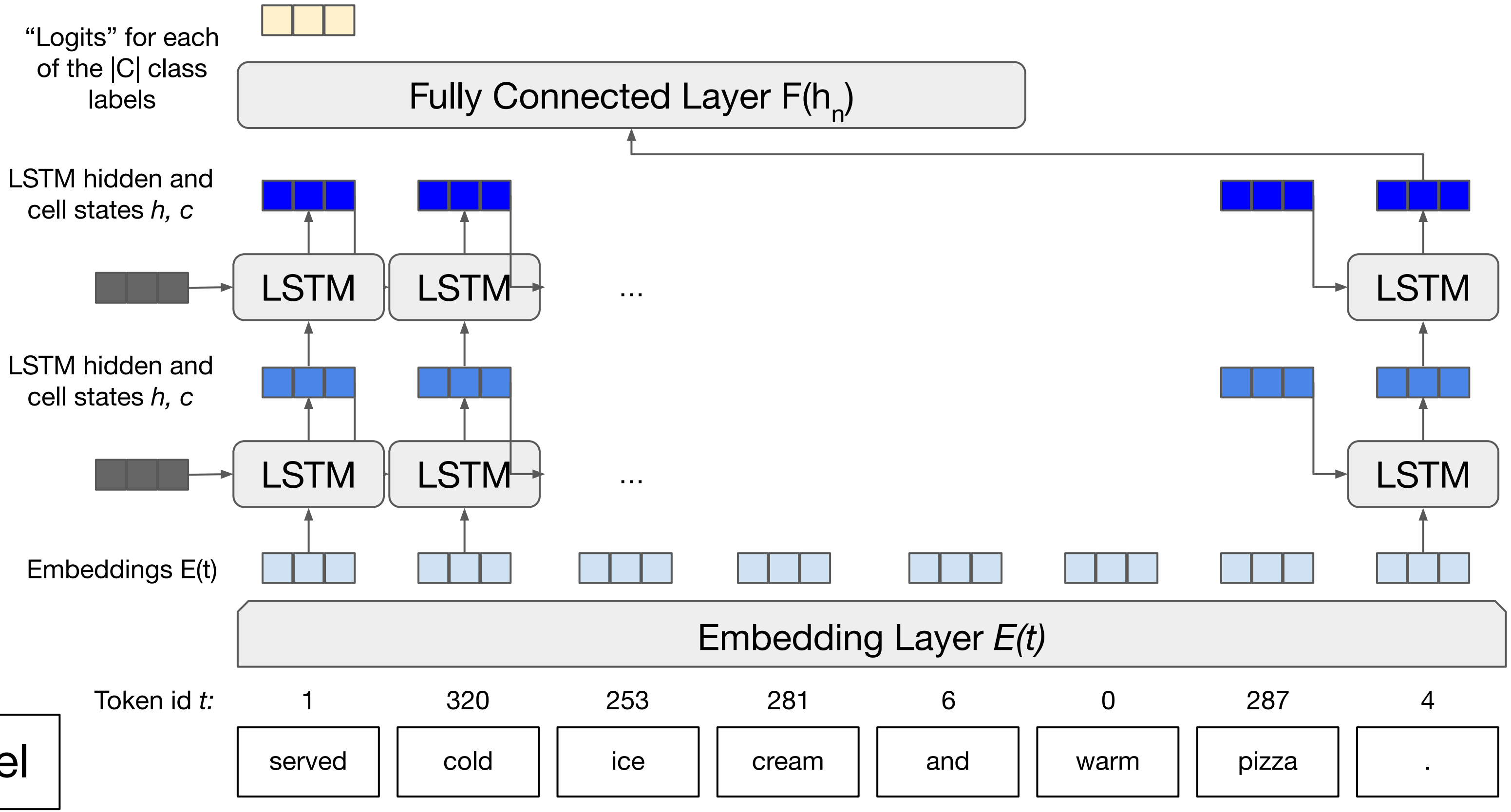
$$h_t = o \odot \tanh(c_t)$$

Text Classification with an LSTM



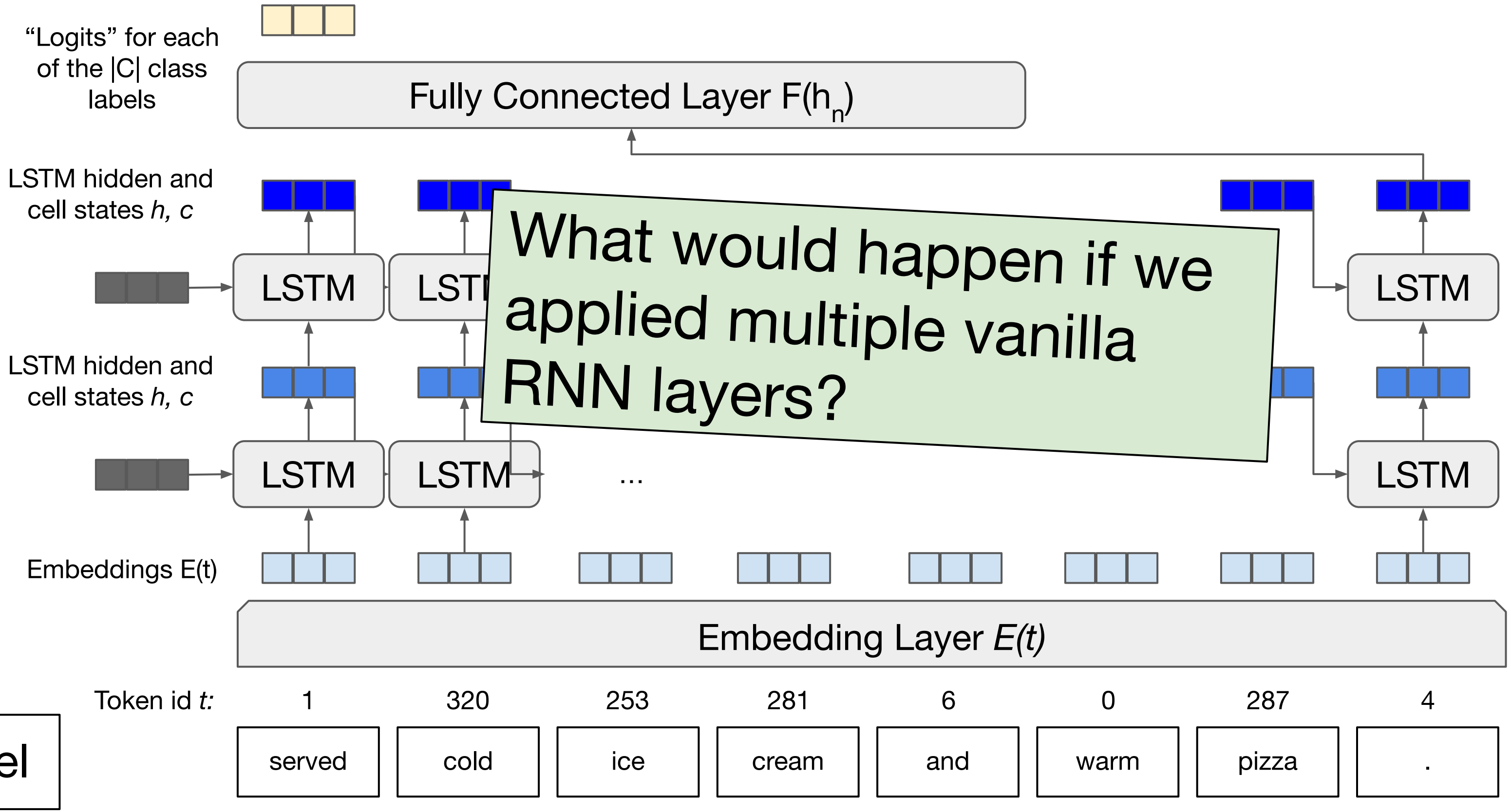
Word-level

Text Classification with a Multi-Layer LSTM



Word-level

Text Classification with a Multi-Layer LSTM



What would happen if we applied multiple vanilla RNN layers?

Word-level

Addressing the Vanishing Gradient Problem

- What do you need to know?
  - LSTMs > vanilla RNNs basically all the time
  - Multi-layer LSTM makes sense; multi-layer vanilla RNN will have such bad gradient vanishing it probably won’t train
  - LSTMs ~= GRUs; think of this choice as a hyperparameter you can tune on your validation data
  - Using an RNN/LSTM/GRU is *different* from using convolutional layers to process text (e.g., short segments combined with some kind of pooling layer)

Overview of Today's Plan

- ~~Course organization and deliverables~~
- ~~Recurrent Neural Networks~~
- ~~Long-Short Term Memory~~
    - Any questions before we move on?
- Applications and Attention Mechanisms

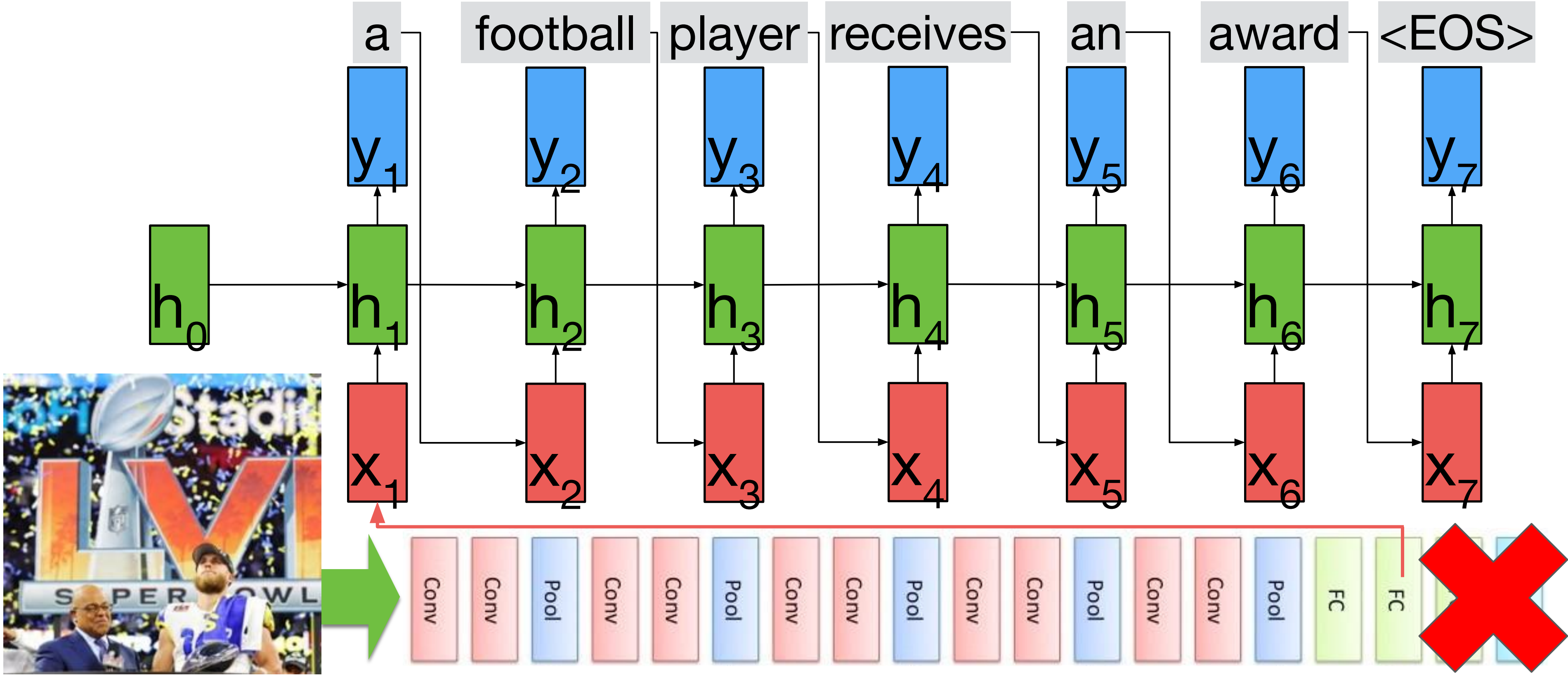
Recurrent Neural Network Applications

- Image Captioning
- Question Answering
- Visual Question Answering
- Speech Recognition
- Action Recognition in Videos
- Text Parsing
- Machine Translation
- Genomic Sequence Classification

Image Captioning with Sequential Decoder

A football player receives an award.

Image Captioning with Sequential Decoder



Question Answering with Encoder-Decoder

Text

Mary moved to the bathroom.

John went to the hallway.

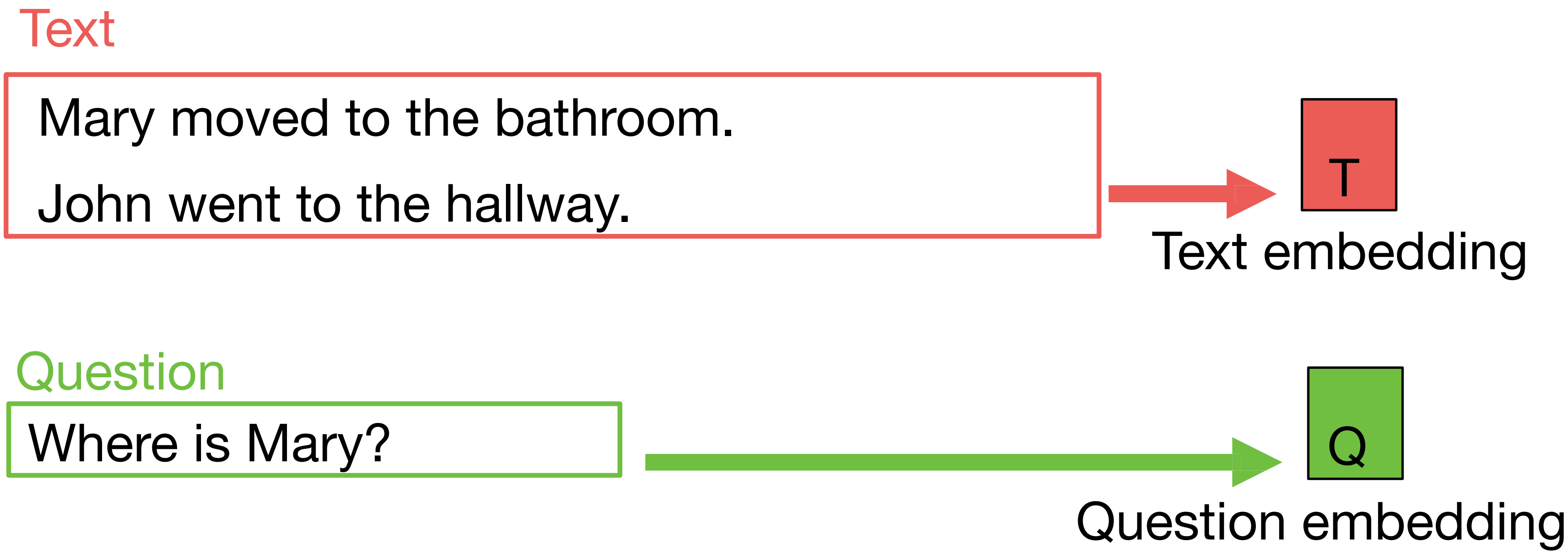
Question

Where is Mary?

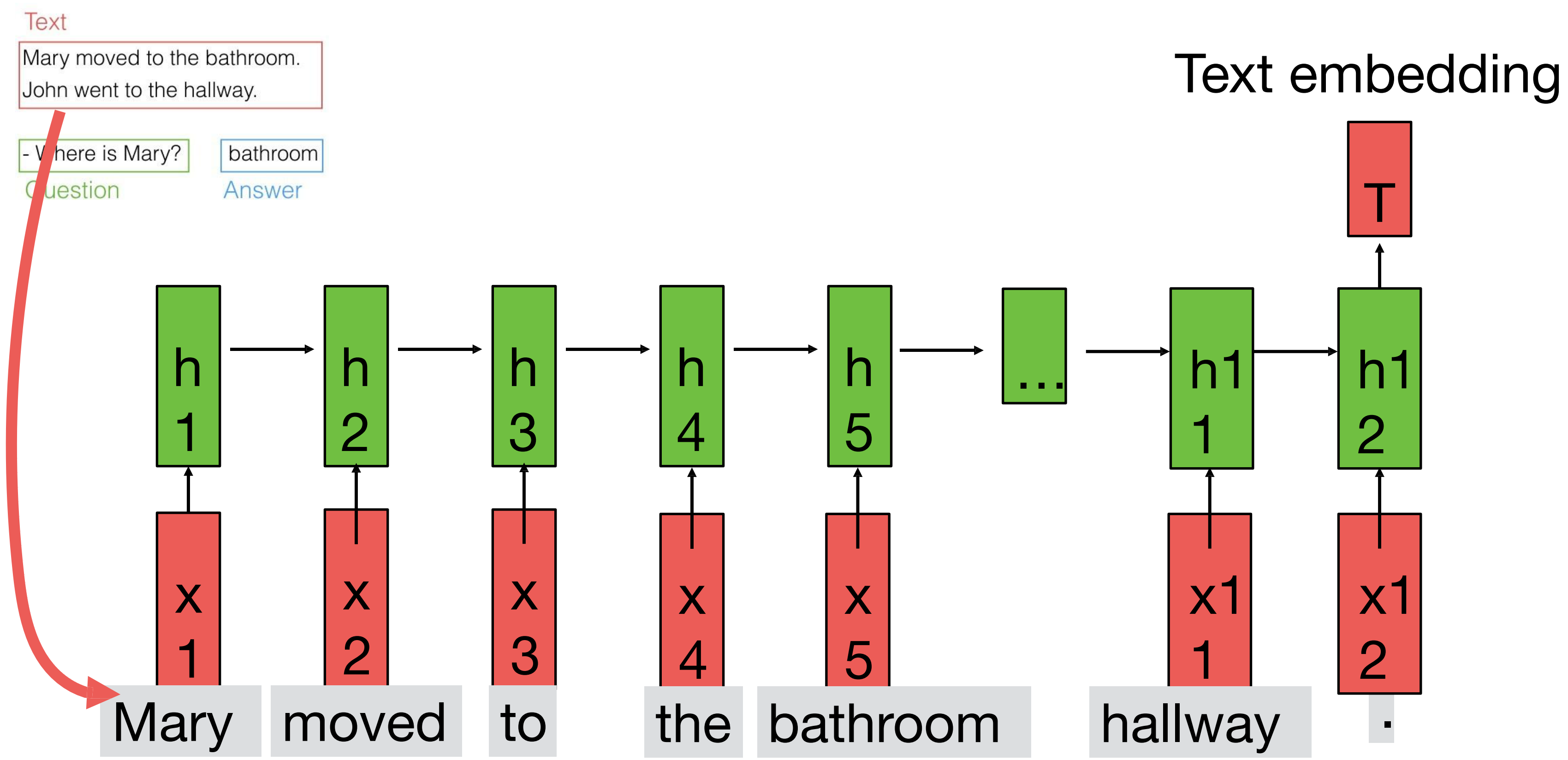
Answer

bathroom

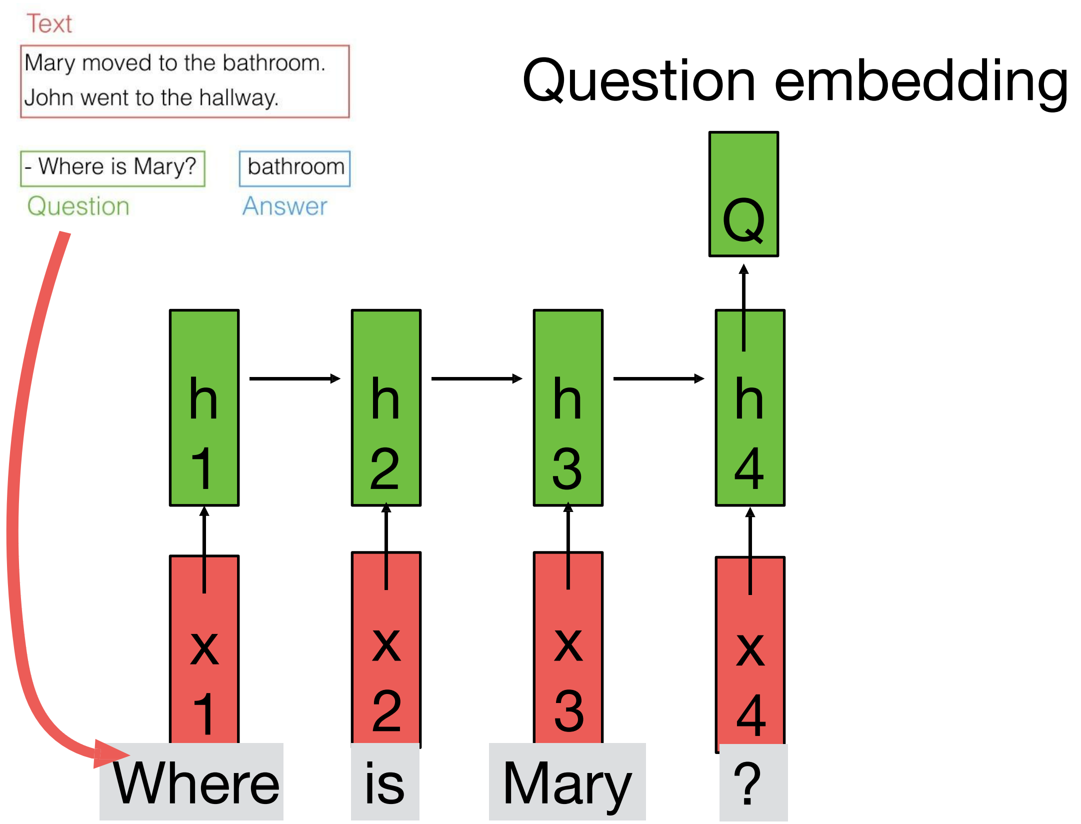
Question Answering with Encoder-Decoder



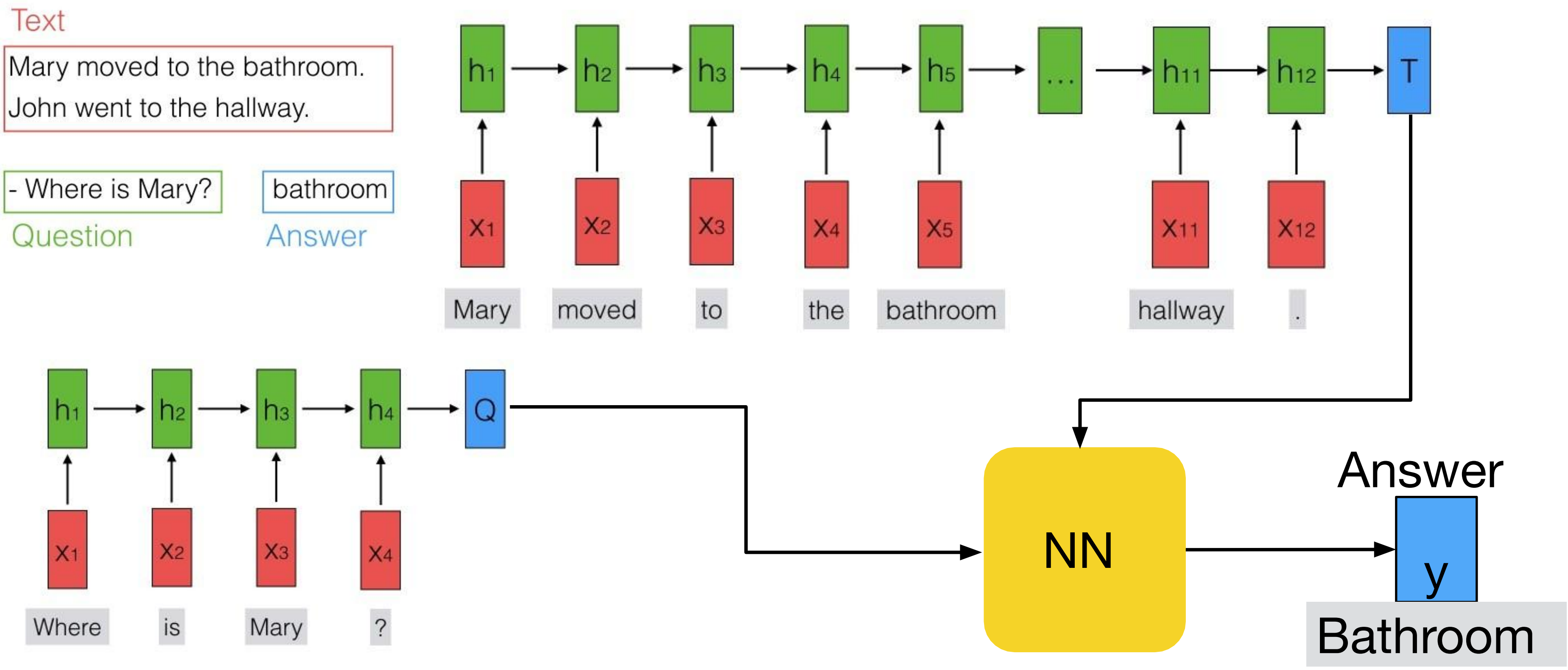
Question Answering with Encoder-Decoder



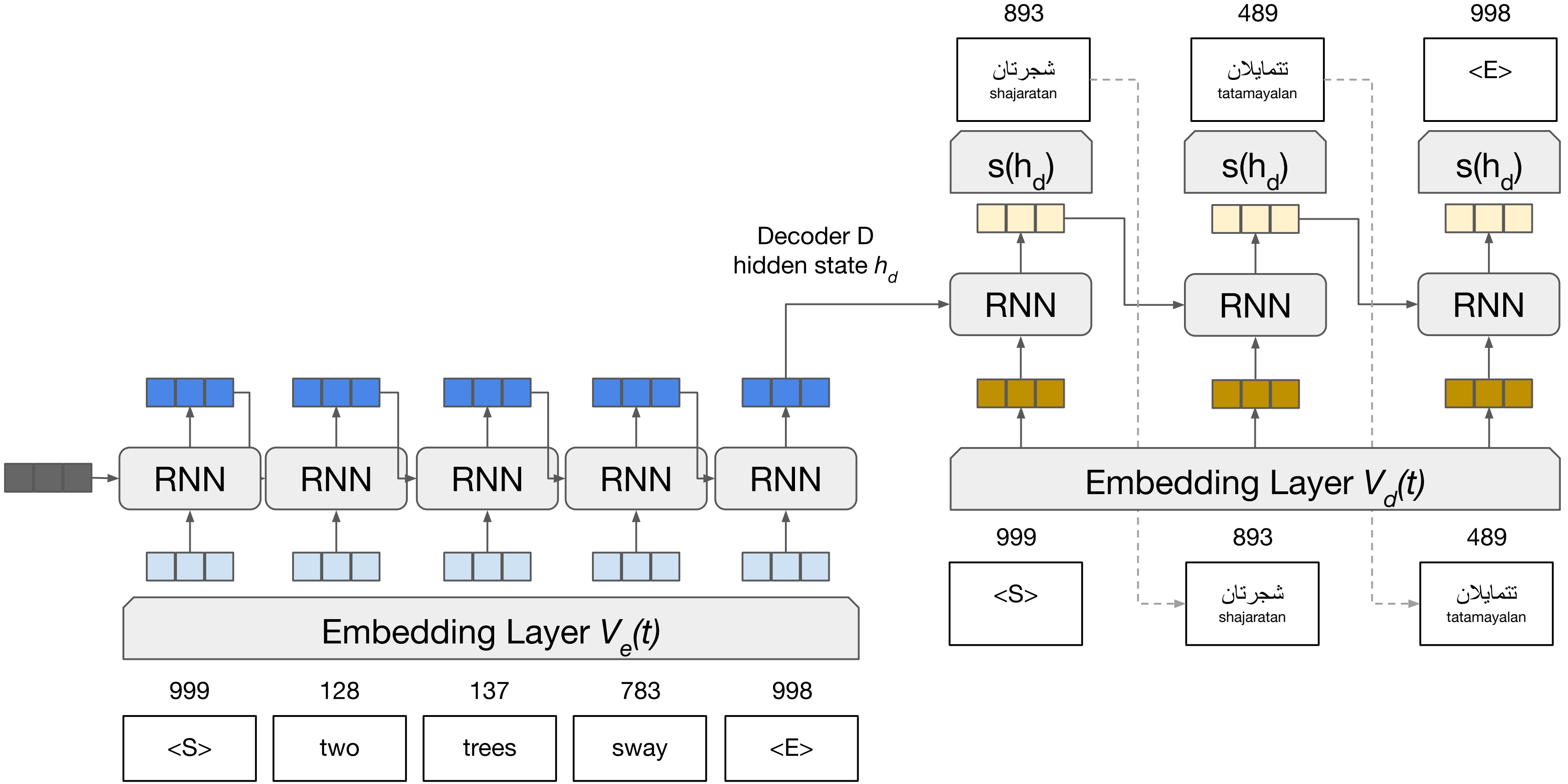
Question Answering with Encoder-Decoder



Question Answering with Encoder-Decoder



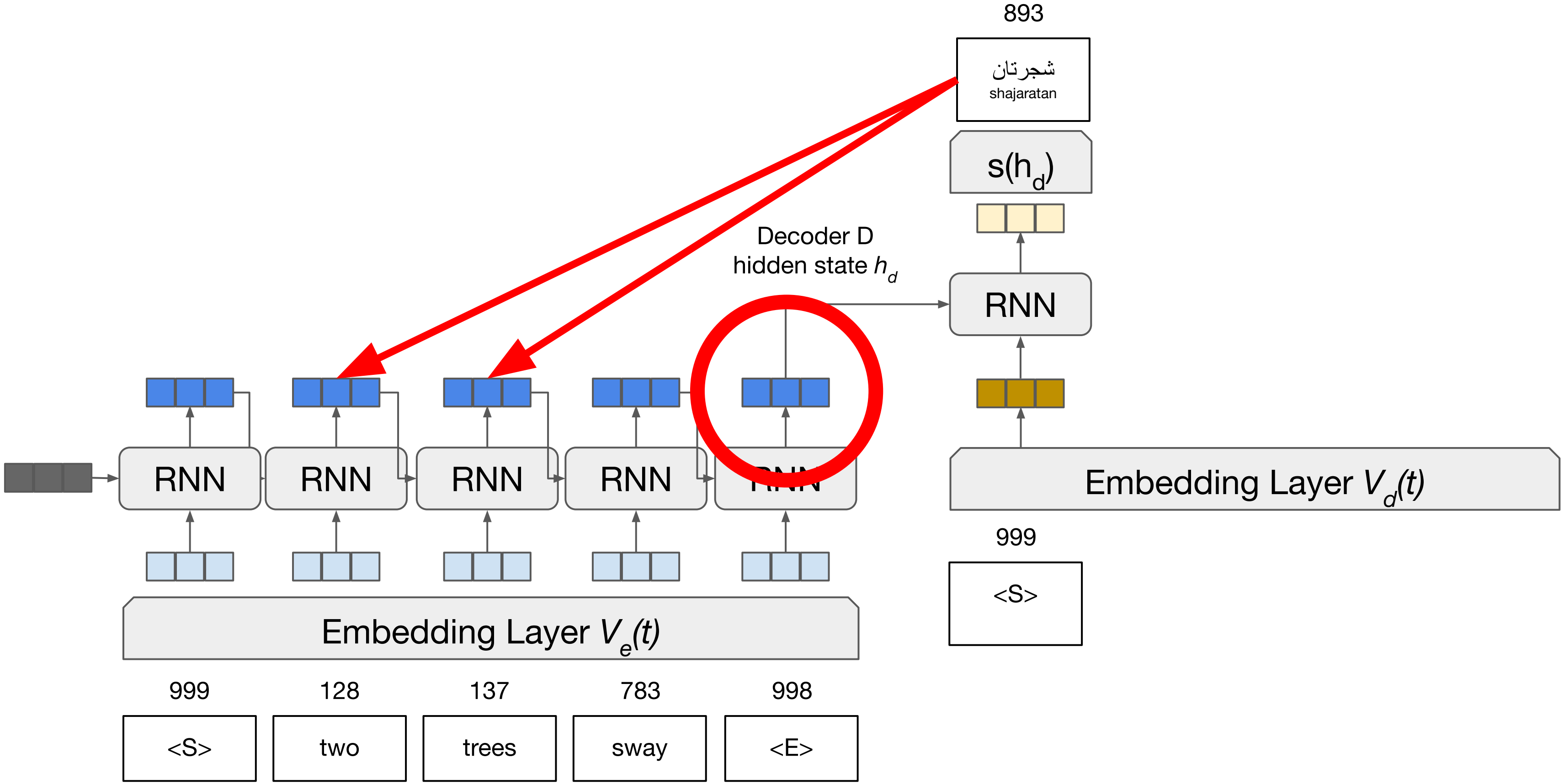
Machine Translation and The Case for Contextual Info



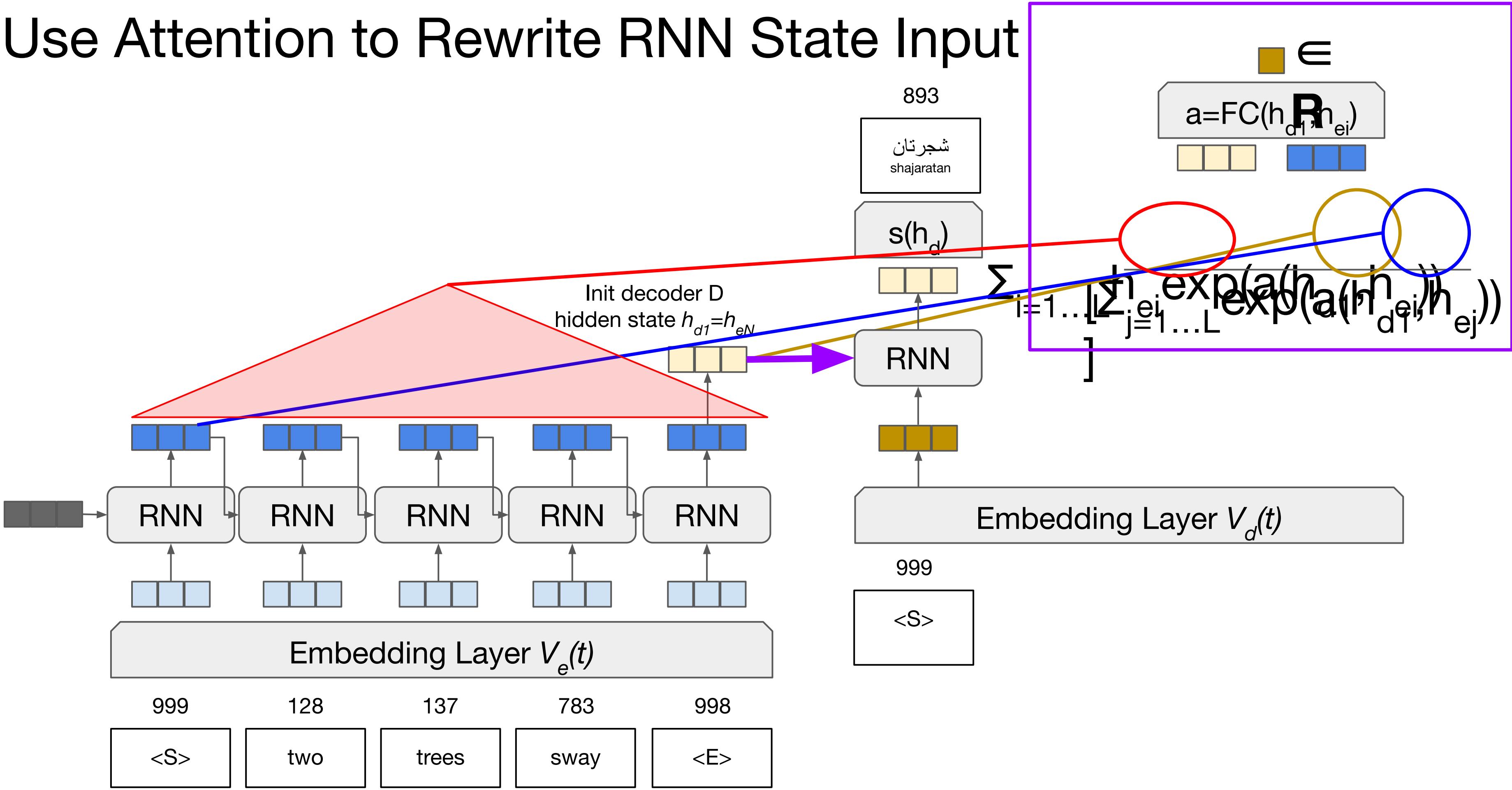
The Case for Contextual Information

- English to Arabic translation
- Arabic has a few things English doesn't have, including the *dual case*, a noun- and verb- form for *pairs*
  - A tree sways
    - شجرة تتأرجح [shajarat tata'arjah]
  - Two trees sway
    - شجرتان تتمايلان [shajaratan tatamayalan]
  - The trees sway
    - الأشجار تتأرجح [al'ashjar tata'arjah]

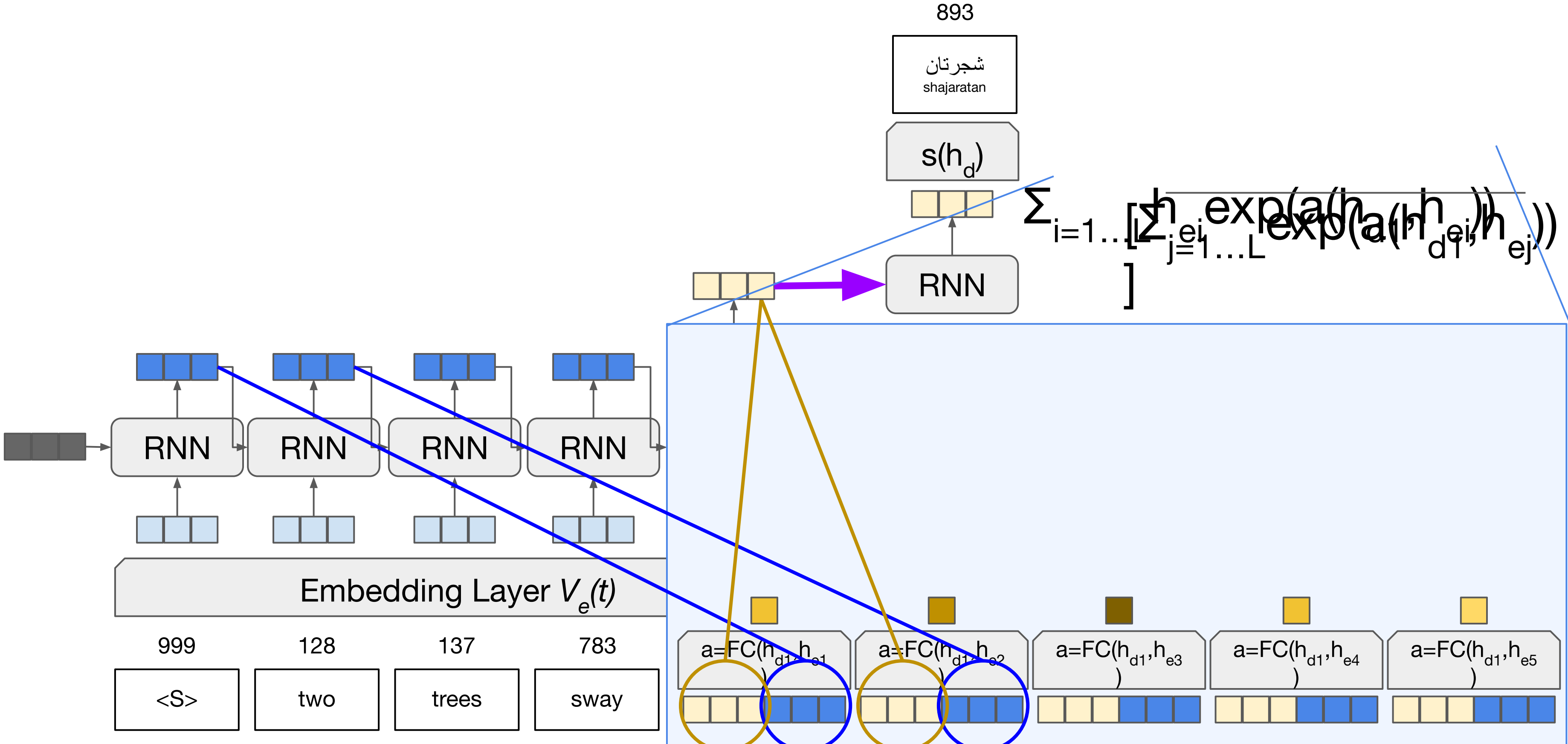
Machine Translation and The Case for Contextual Info



Use Attention to Rewrite RNN State Input

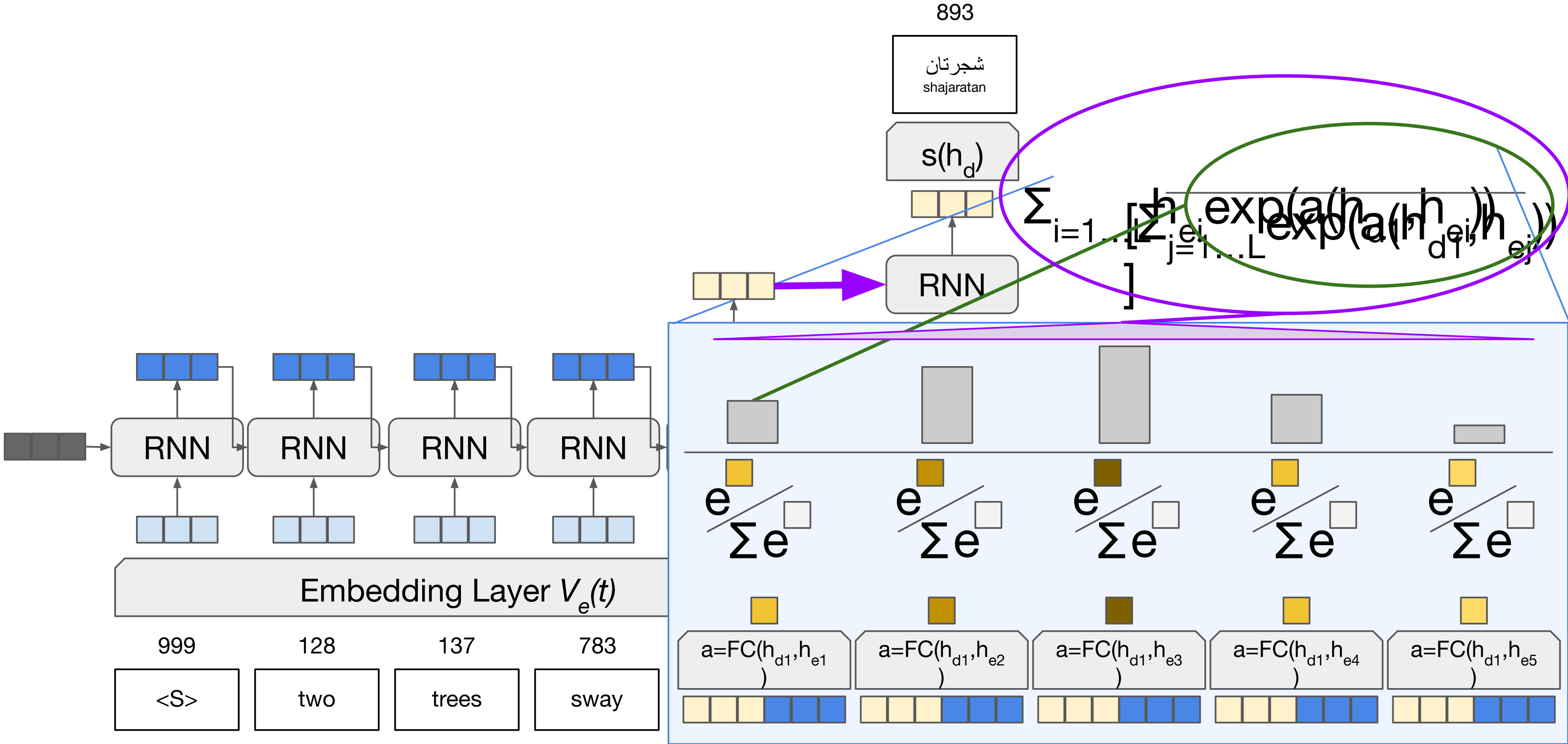


Attention



893
شجرتان
shajaratan
s(h_d)
RNN
Σ_i=1…L[h_ei exp(a(h_d1,h_ei)) / Σ_j=1…L exp(a(h_d1,h_ej))]
RNN
RNN
RNN
RNN
Embedding Layer V_e(t)
999
128
137
783
<S>
two
trees
sway
a=FC(h_d1,h_e1)
a=FC(h_d1,h_e2)
a=FC(h_d1,h_e3)
a=FC(h_d1,h_e4)
a=FC(h_d1,h_e5)

Attention

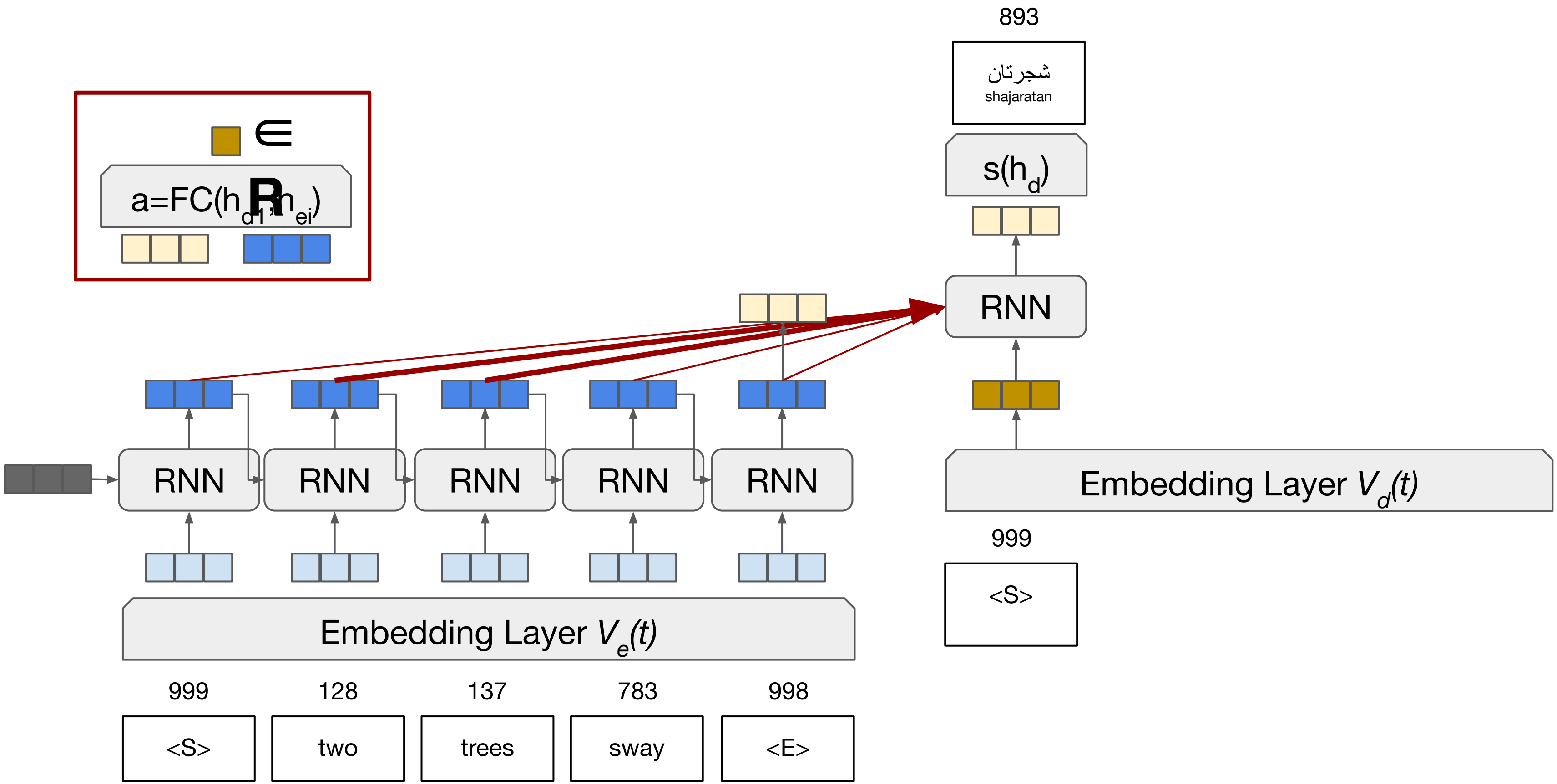


893
شجرتان
shajaratan
s(h_d)
RNN
RNN
RNN
RNN
RNN
Embedding Layer V_e(t)
999
128
137
783
<S>
two
trees
sway
a=FC(h_d1,h_e1)
a=FC(h_d1,h_e2)
a=FC(h_d1,h_e3)
a=FC(h_d1,h_e4)
a=FC(h_d1,h_e5)

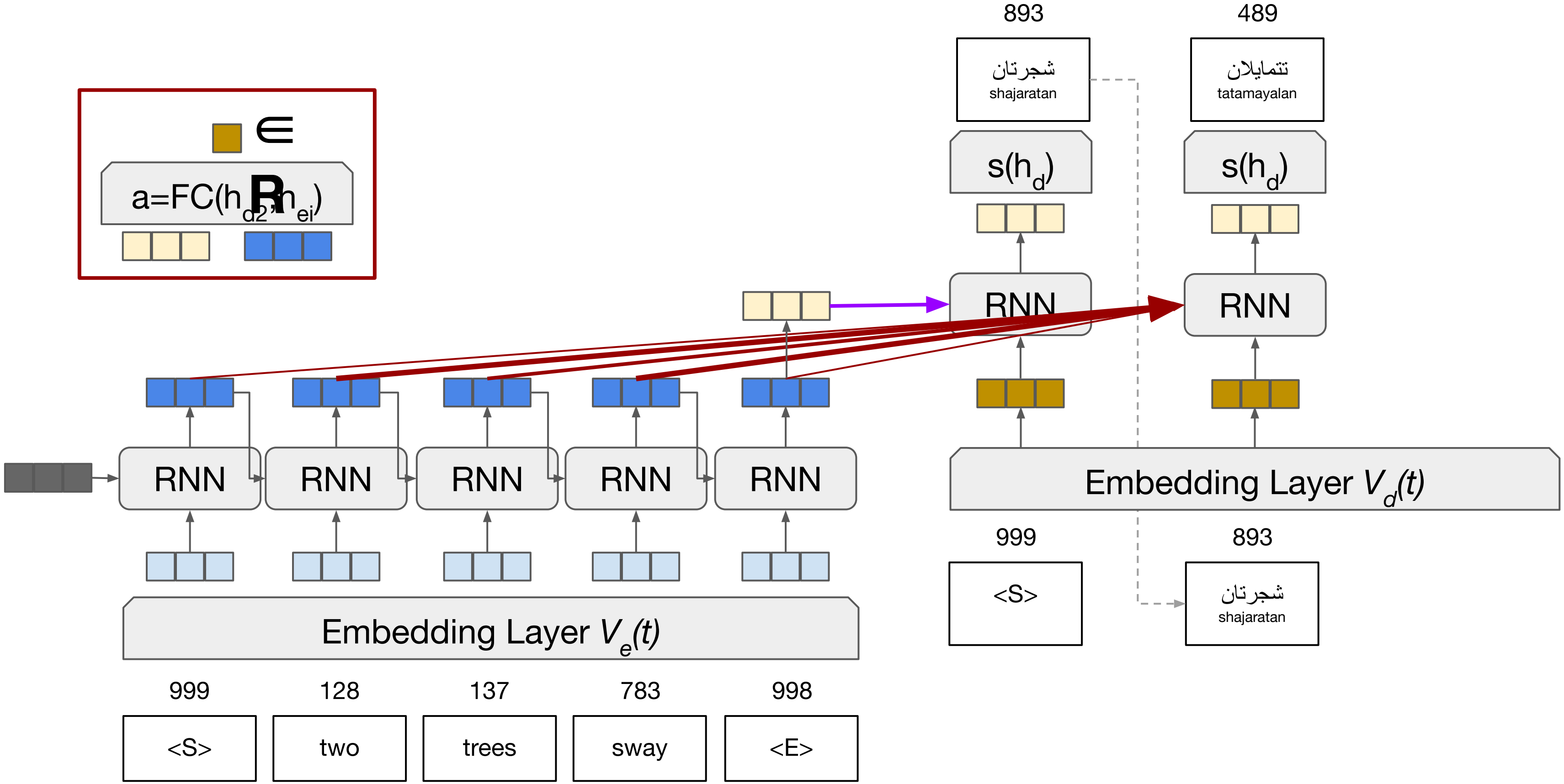
Encoder-Decoder Attention

- The final hidden state of an encoder may not carry enough information about previous tokens to perform decoding
- *Attention* reweights encoder states $h_{e[1:L]}$ as a function of the decoder hidden state $h_{d[t-1]}$ at each timestep
  - $\mathbf{h}_{d[t]}=[\Sigma_{i=1 \ldots L}\mathbf{h}_{e[i]}\exp(a(\mathbf{h}_{d[t-1]},h_{e[i]}))\,]\,/\,[\Sigma_{j=1 \ldots L}\exp(a(\mathbf{h}_{d[t-1]},\mathbf{h}_{e[j]}))]$
- A learned attention *head* determines the matching score between the decoder's last output state and each candidate encoder hidden state
- An attention *head* has shared weights for all encoder states, so it's cheap to add, parameter-wise

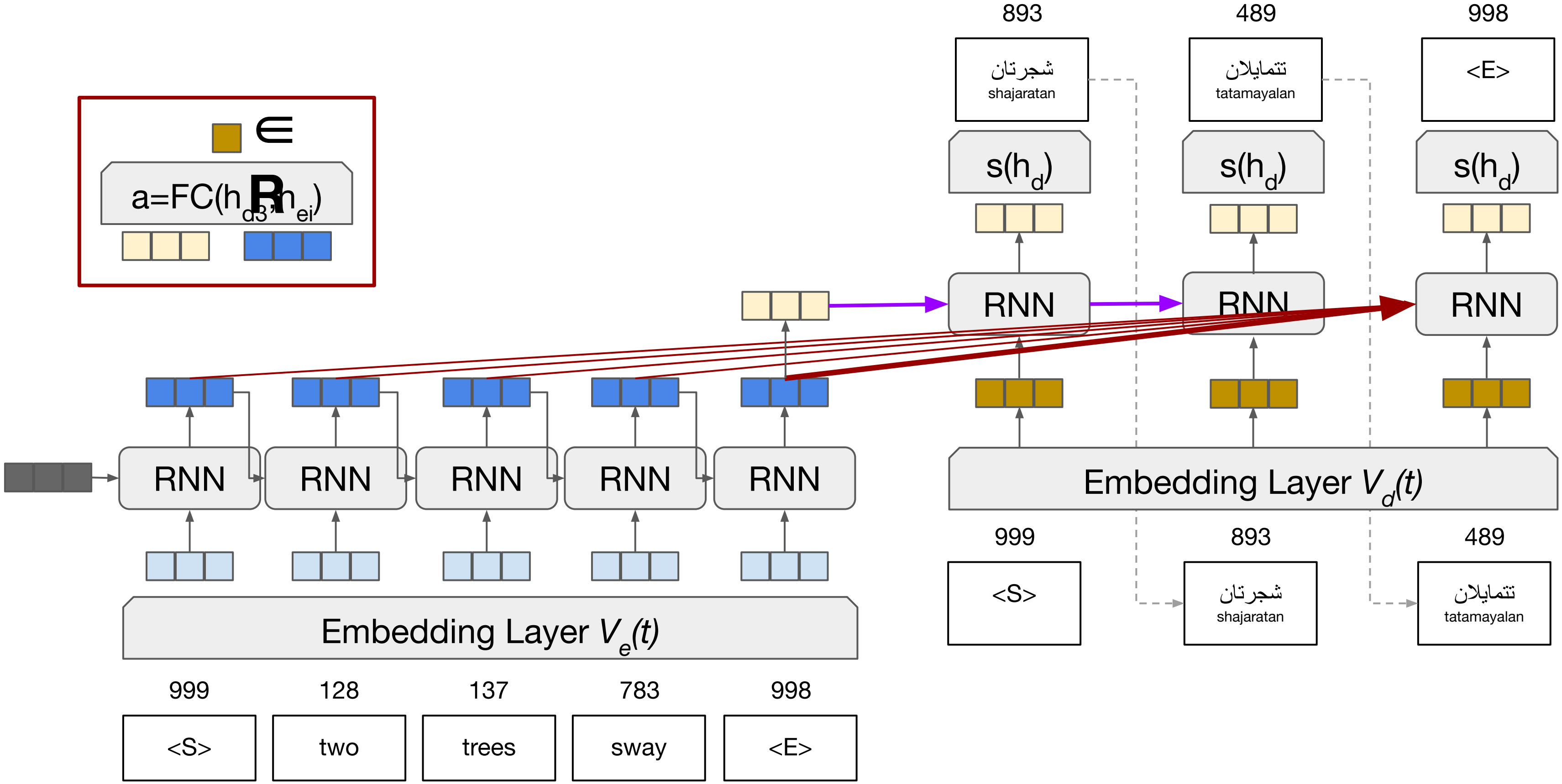
Encoder-Decoder Attention



Encoder-Decoder Attention



Encoder-Decoder Attention



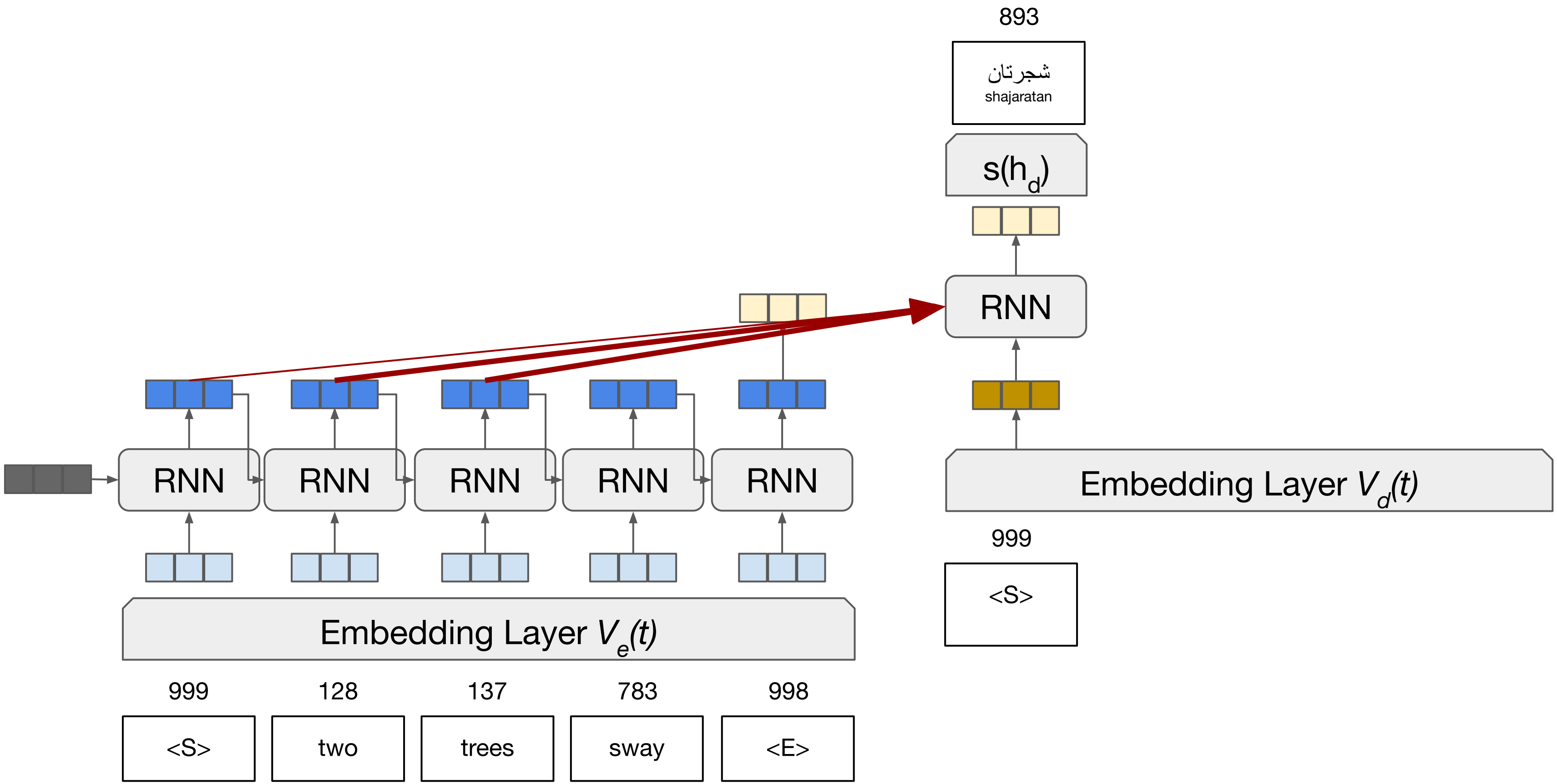
Encoder-Decoder Attention

- What are some pitfalls or weaknesses we can anticipate with the attention mechanism described so far?
- *Pitfalls*:
  - We probably still don’t need the entire sequence for every decoding step
  - A lot of our weights will be near zero, probably
  - The shared parameters of the attention head might only learn to pick out certain dependencies (e.g., noun-verb agreement, determiner-gender agreement, …)

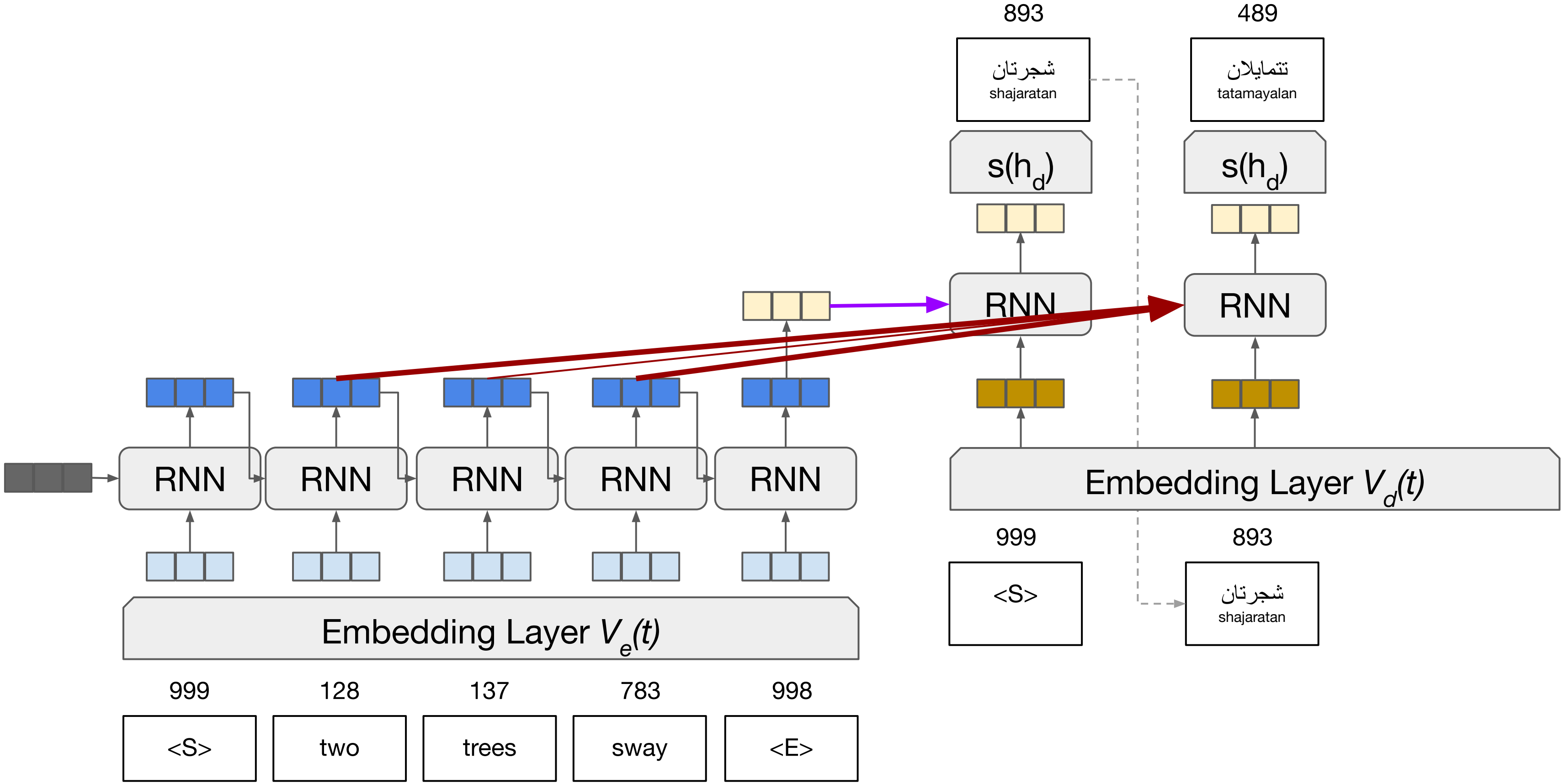
Encoder-Decoder Attention: Global Versus Local

- We probably still don’t need the entire sequence for every decoding step
- Can limit attention to a window of *k* around current decoding index to limit computation

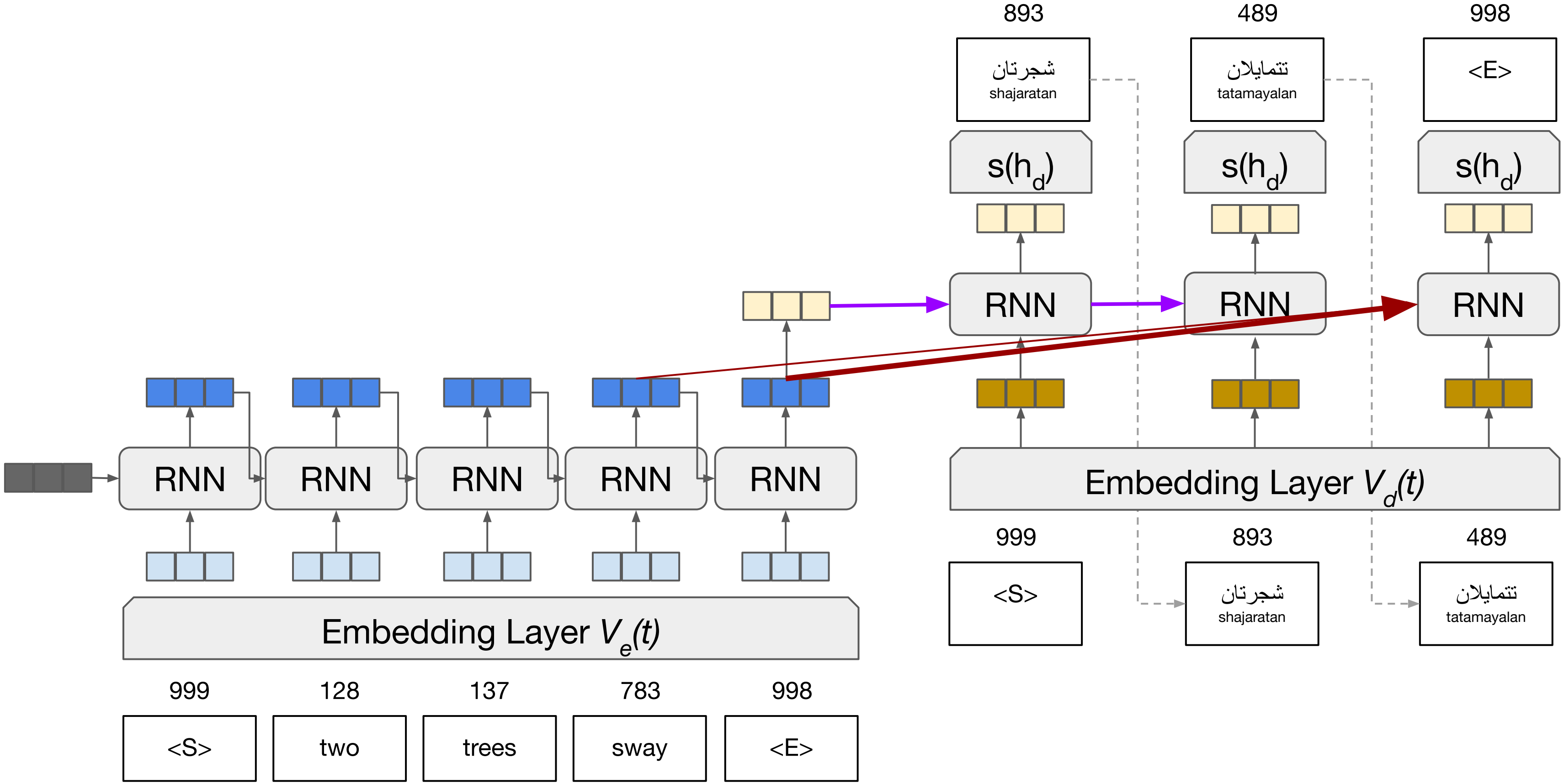
Encoder-Decoder Attention: Global Versus Local



Encoder-Decoder Attention: Global Versus Local



Encoder-Decoder Attention: Global Versus Local



Encoder-Decoder Attention: Soft versus Hard

- A lot of our weights will be near zero, probably
- We could instead just use the *most relevant* hidden state

Soft Attention

$$\Sigma_{i=1\ldots L} \frac{h_{ei}\exp(a(h_{d1},h_{ei}))}{[\Sigma_{j=1\ldots L}\exp(a(h_{d1},h_{ej}))]}$$

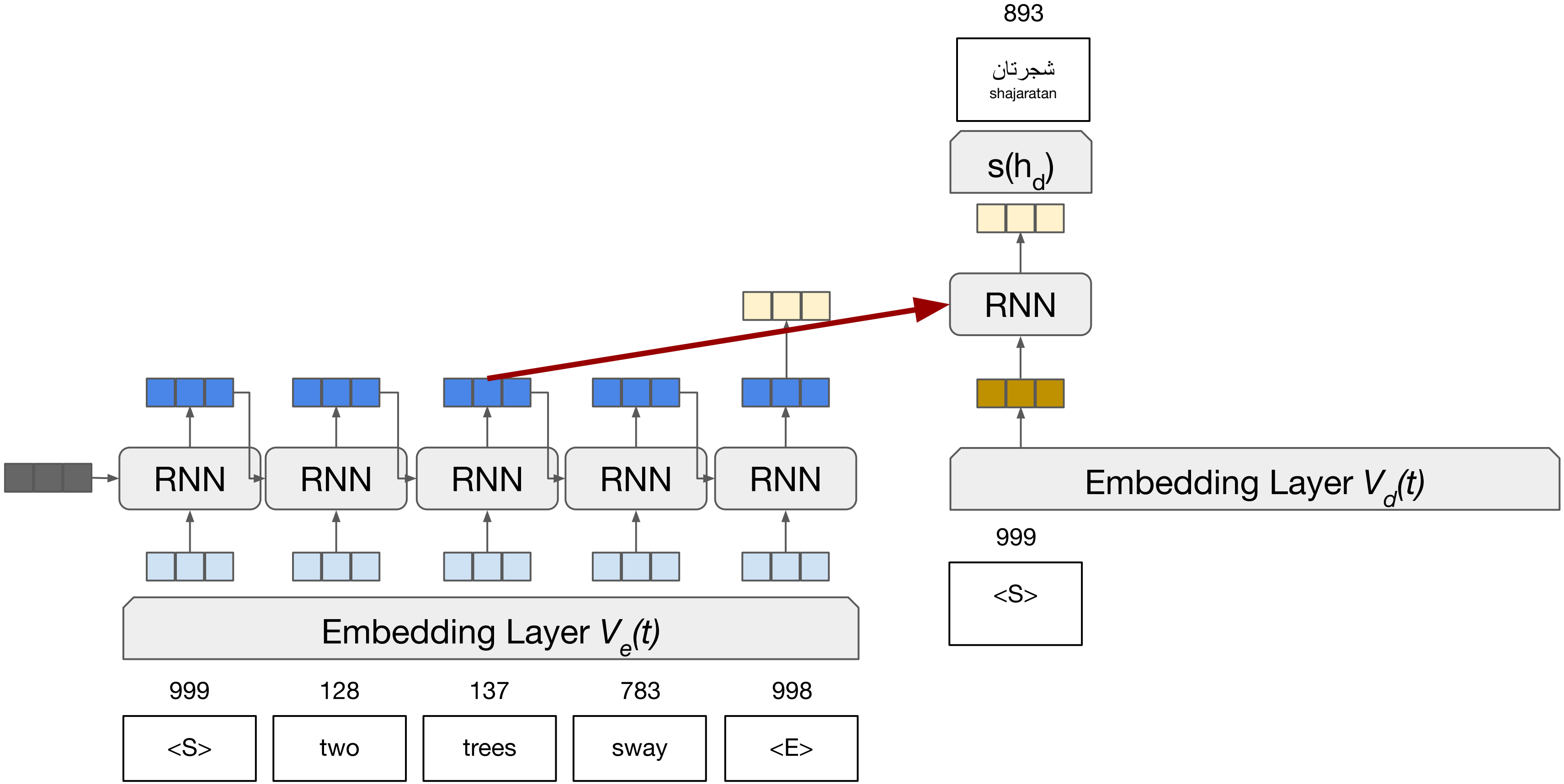
Hard Attention

$$\text{argmax}^{*}_{he.i=1\ldots L} \frac{\exp(a(h_{d1},h_{ei}))}{[\Sigma_{j=1\ldots L}\exp(a(h_{d1},h_{ej}))]}$$

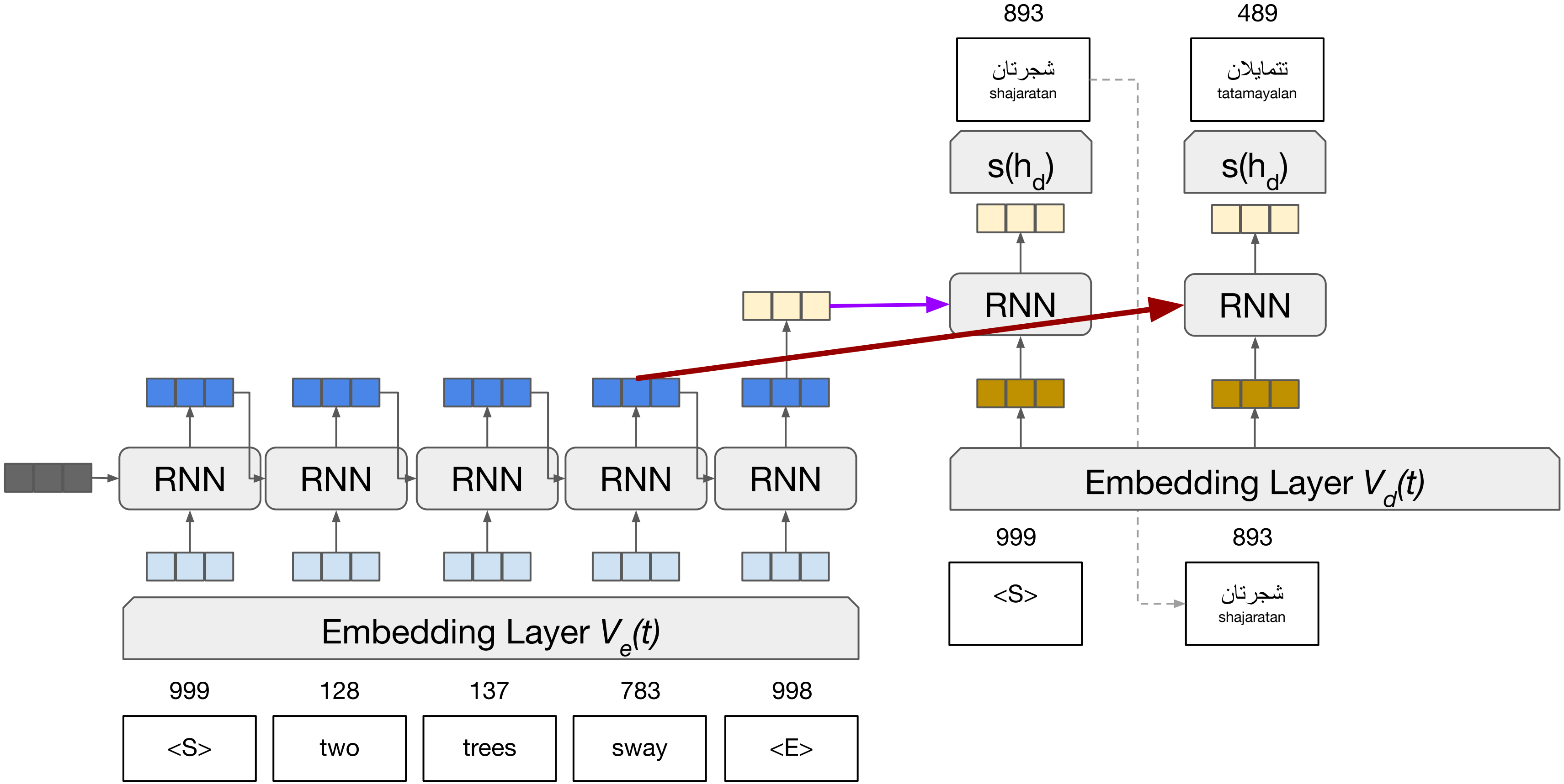
* with some tricks to ensure the whole pipeline stays differentiable

Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, ICML 2015

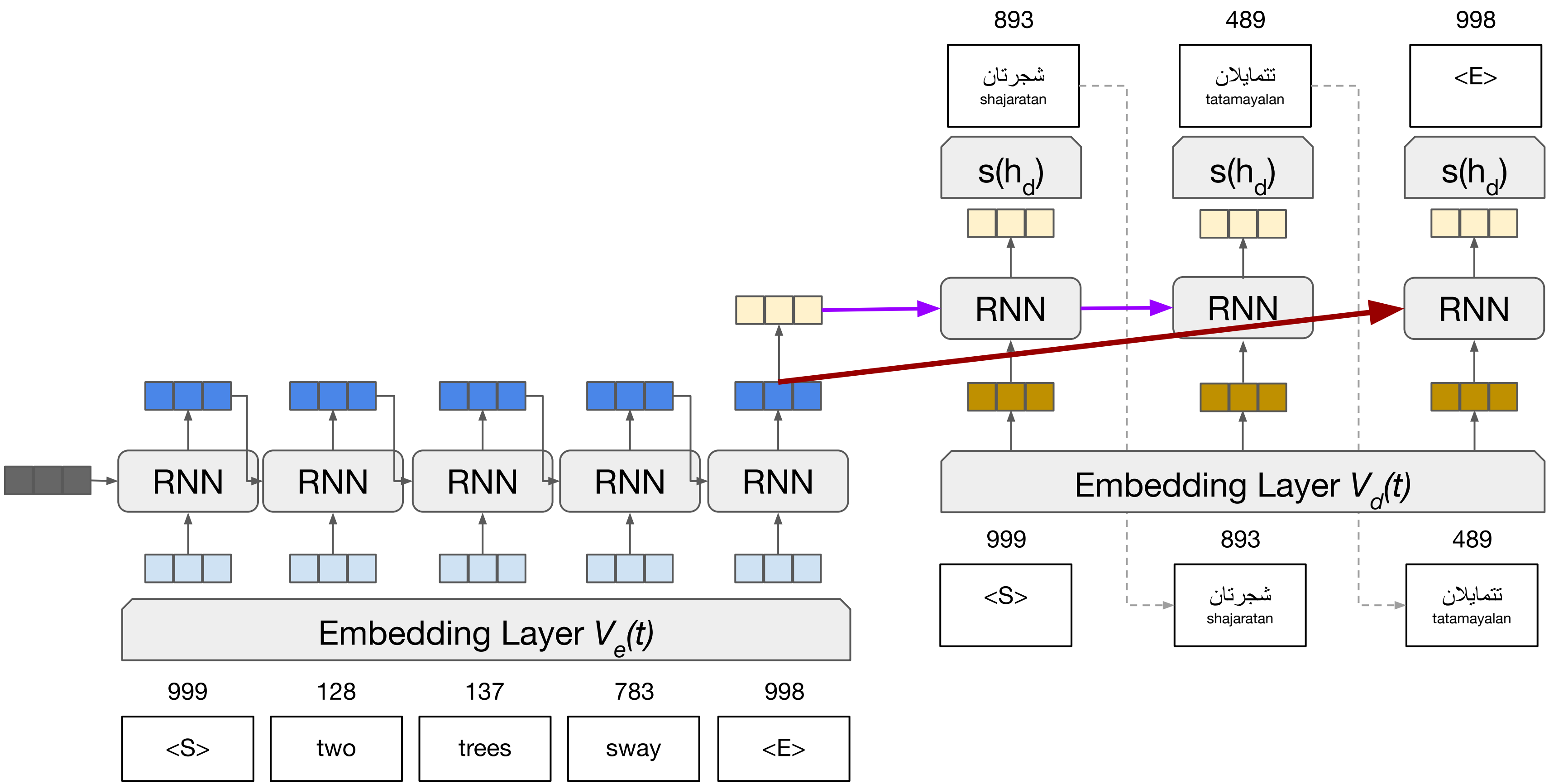
Encoder-Decoder Attention: Hard Versus Soft



Encoder-Decoder Attention: Hard Versus Soft



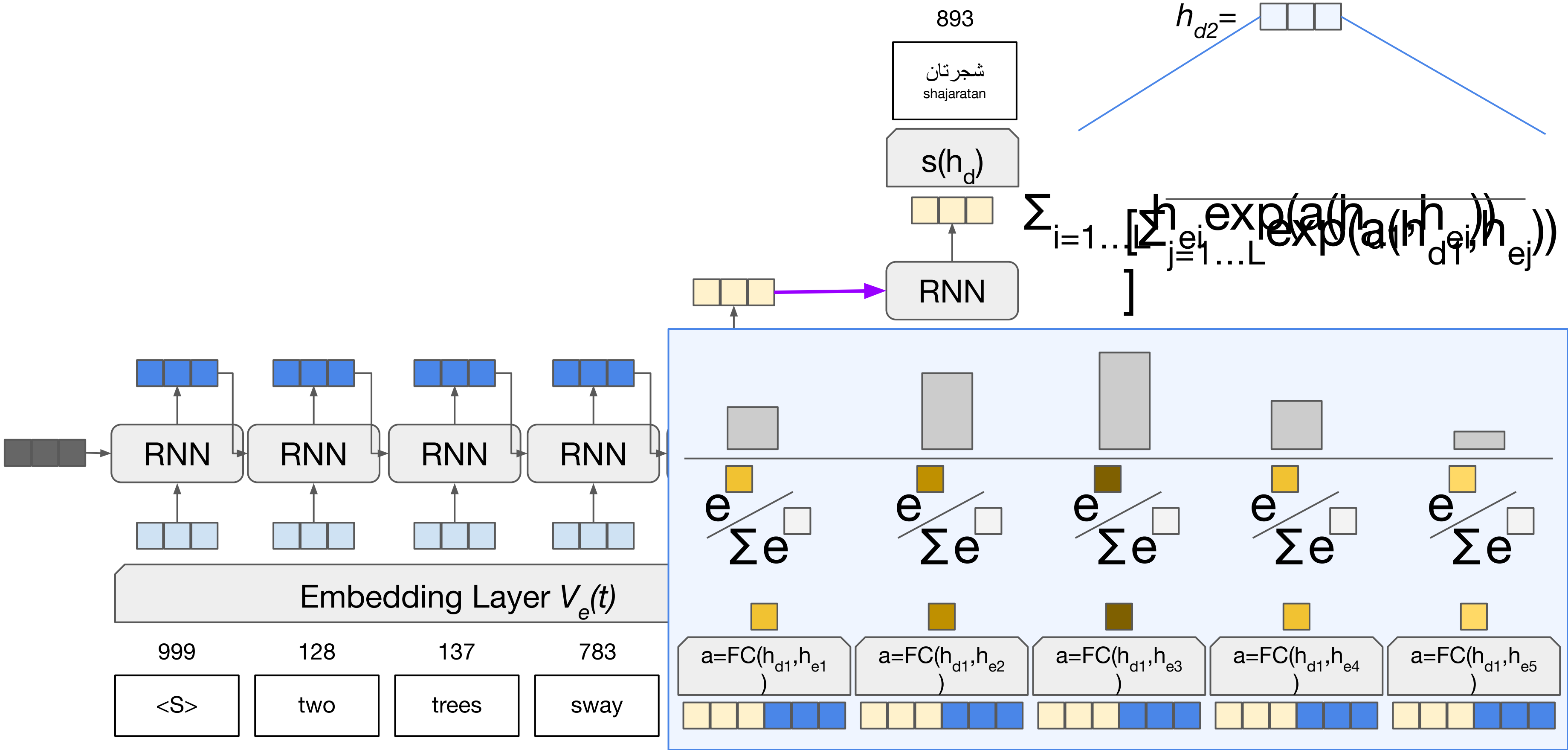
Encoder-Decoder Attention: Hard Versus Soft



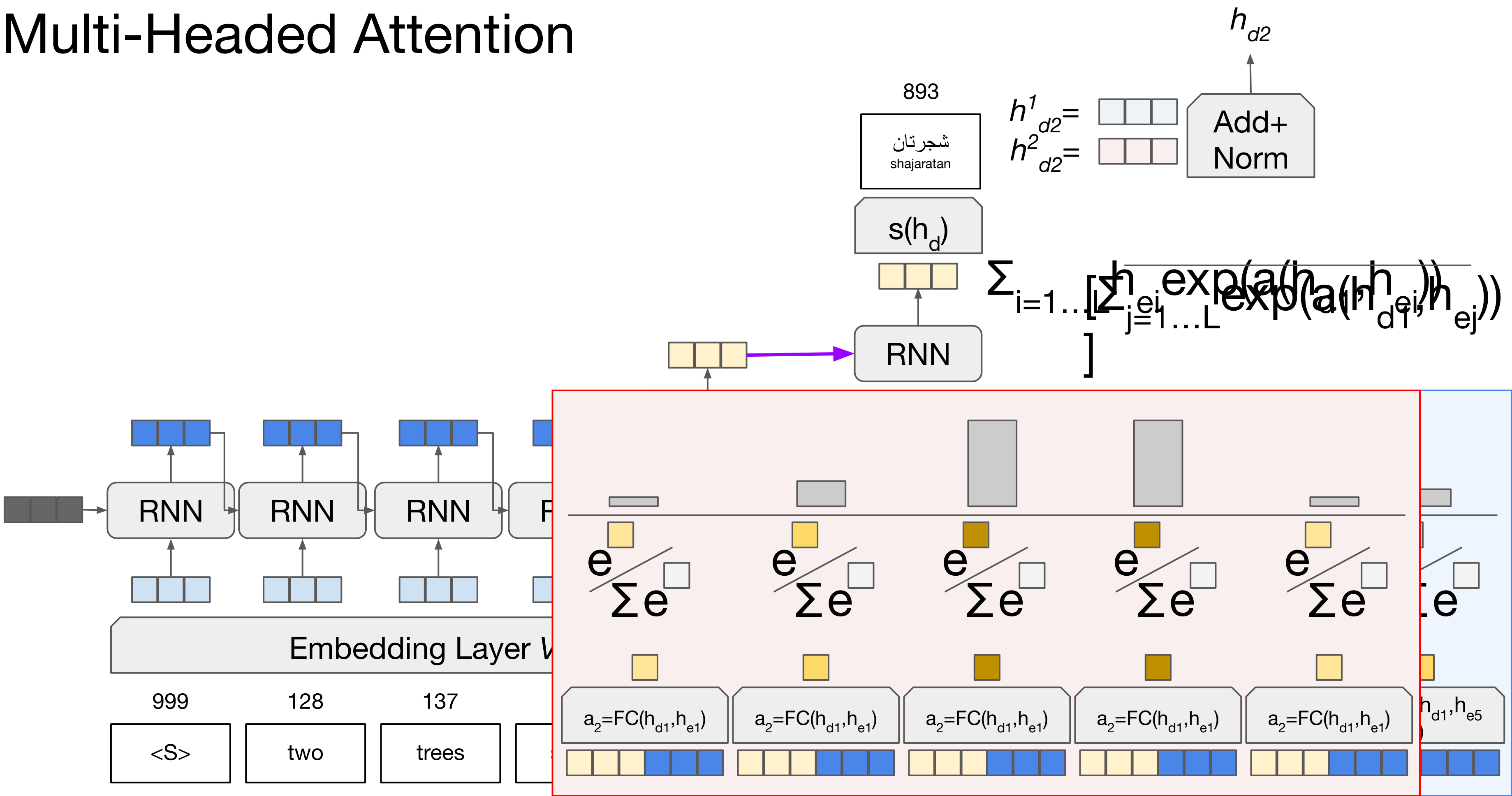
Encoder-Decoder Attention: Multi-headed Attention

- The shared parameters of the attention head might only learn to pick out certain dependencies (e.g., noun-verb agreement, determiner-gender agreement, …)
- We can create *multi-headed attention*, where we choose a number of attention heads *k* to learn independent of one another

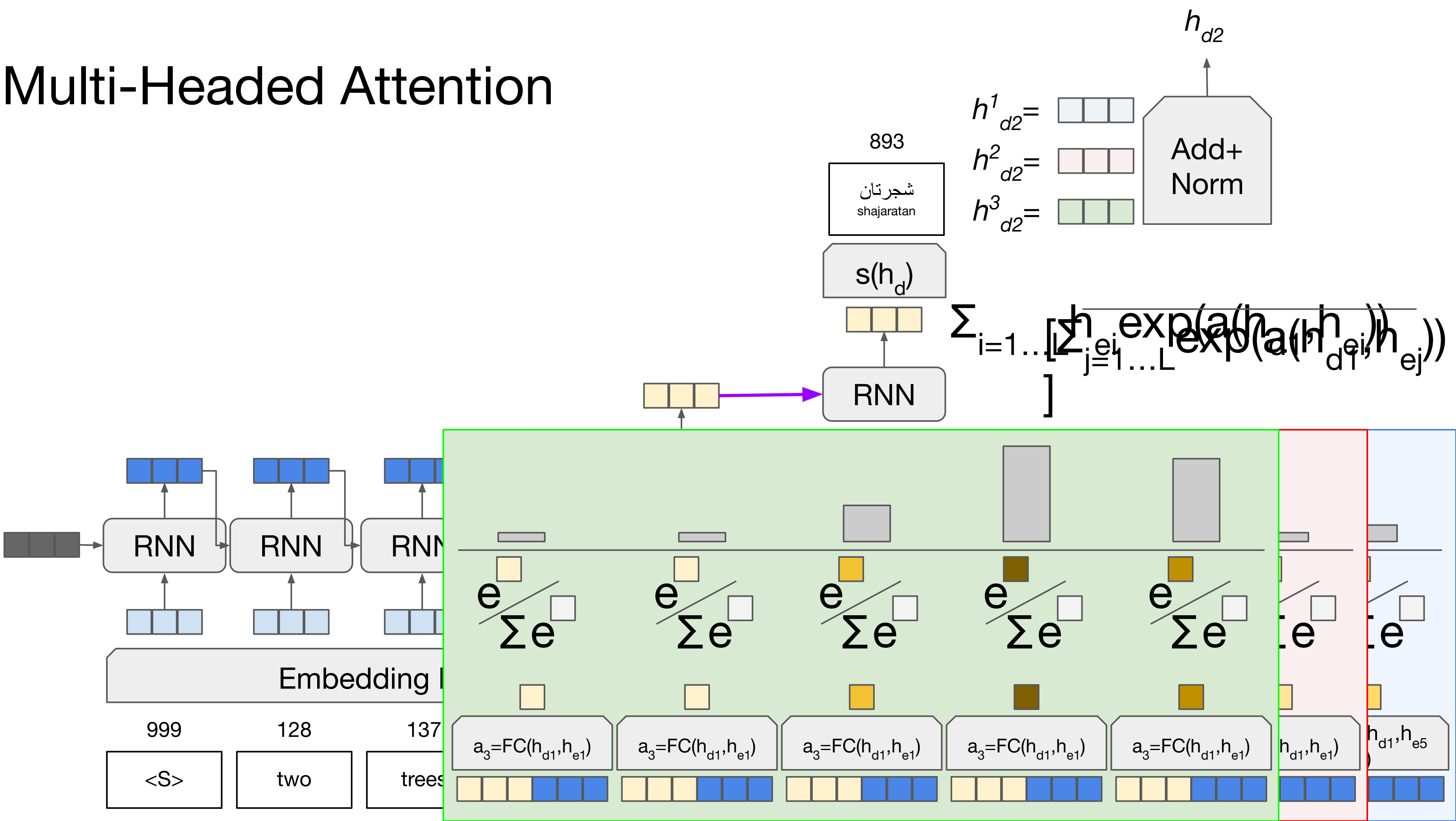
Multi-Headed Attention



Multi-Headed Attention



Multi-Headed Attention



Encoder-Decoder Attention: Recap

- The final hidden state of an encoder may not carry enough information about previous tokens to perform decoding
- *Global Soft Attention* creates a new conditional decoding vector *x* that is a weighted sum of all encoder hidden states, rather than only the final hidden state only
    - Subset of states: “Local” attention
    - Max score instead of weighted: “Hard” attention
- *Multi-headed Attention* learns multiple such scoring functions and sums the final vectors from each as the input hidden state

Action Items for You

- Your project proposals are due **today**
- The midterm exam is *next week* during class; Feb 24th
    - Covers material in Module 1 and assumes knowledge of material from prerequisite courses like CSCI 567
- Coding Assignment 1 is due **Monday Feb 27th**
- If you want to start thinking ahead, the next deliverable after Coding Assignment 1 is the *Project Survey Report* [March 10]
    - See syllabus for details

CSCI 566: Deep Learning and Its Applications

Jesse Thomason

Lecture 5: Recurrent Neural Networks