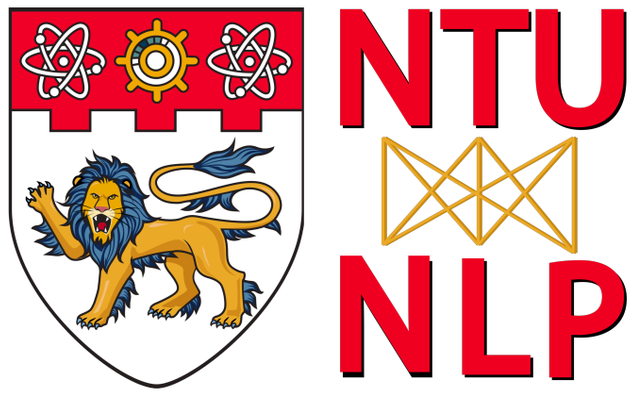
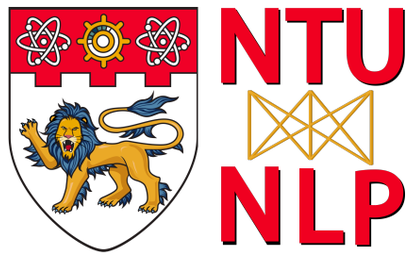
Deep Learning for Natural Language Processing

Shafiq Joty

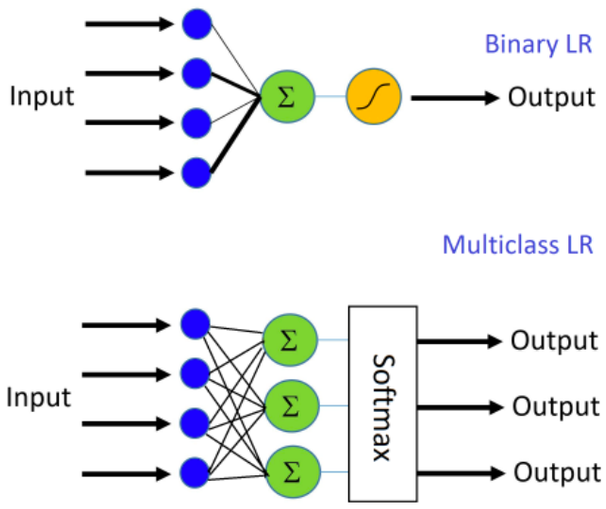


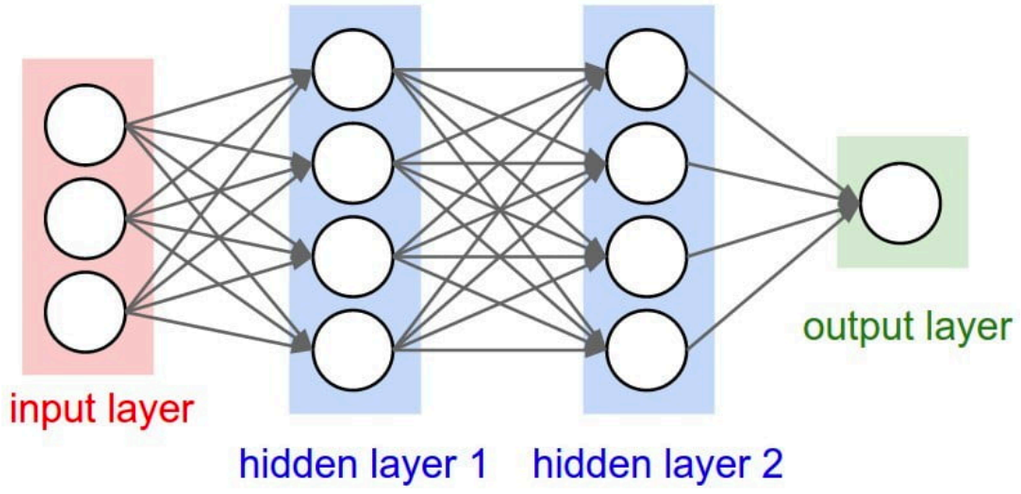
Lecture 6: Recurrent Neural Nets

Things Covered So Far



- Lecture 2: Linear Models (Linear & Logistic Regression)
  - Training with gradient descent and SGD
- Lecture 3: Extended linear models to MLP/FNN
  - Training with SGD (+ Backprop)





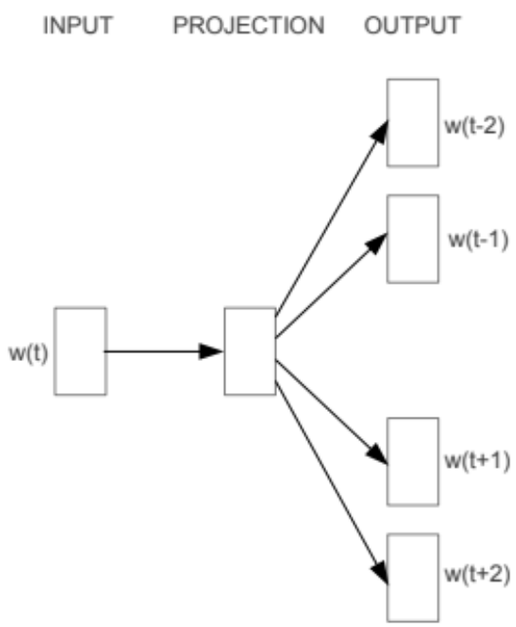
Things Covered So Far

NTU NLP

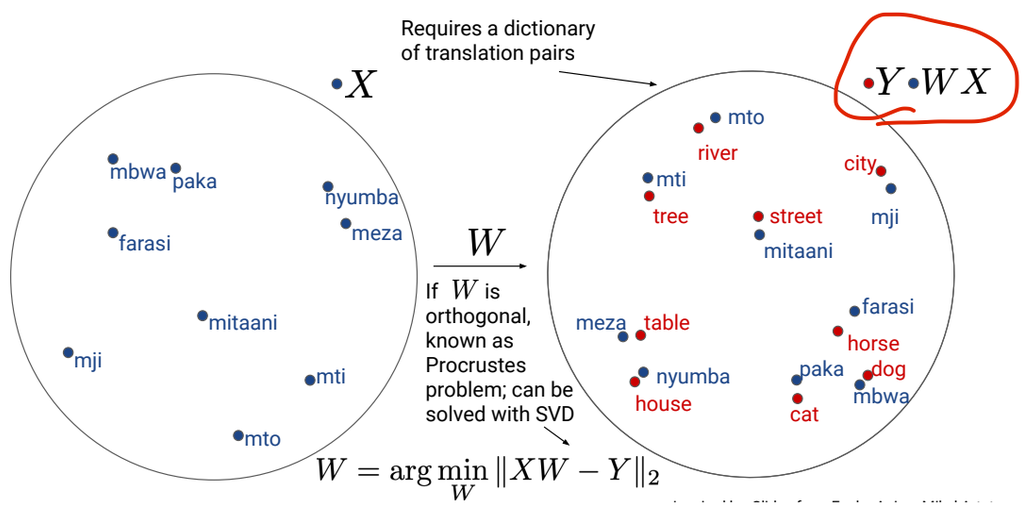
- Lecture 4: Used simple FNN as
  - A. Word2Vec Models
  - B. Cross-lingual Mapping

each word has a static embedding

learn mapping for these Embedding ?



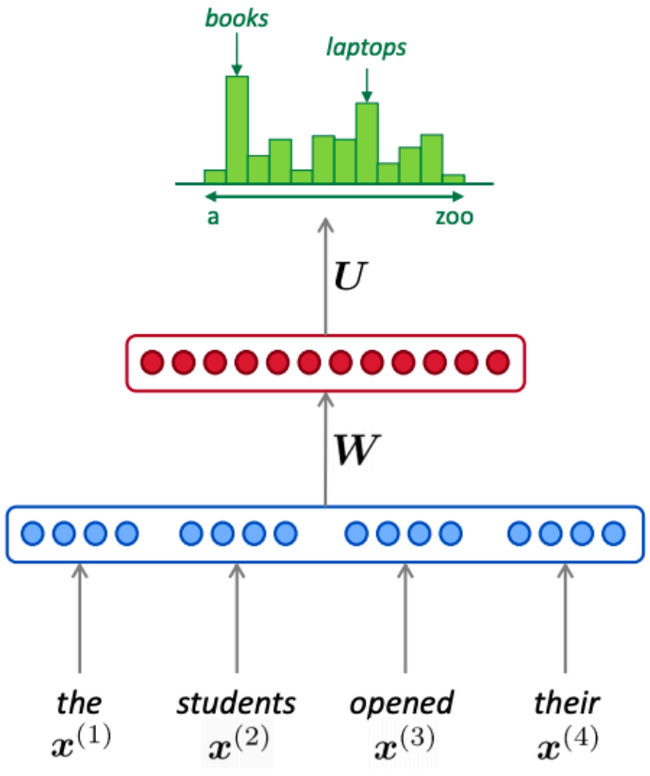
Skip-gram

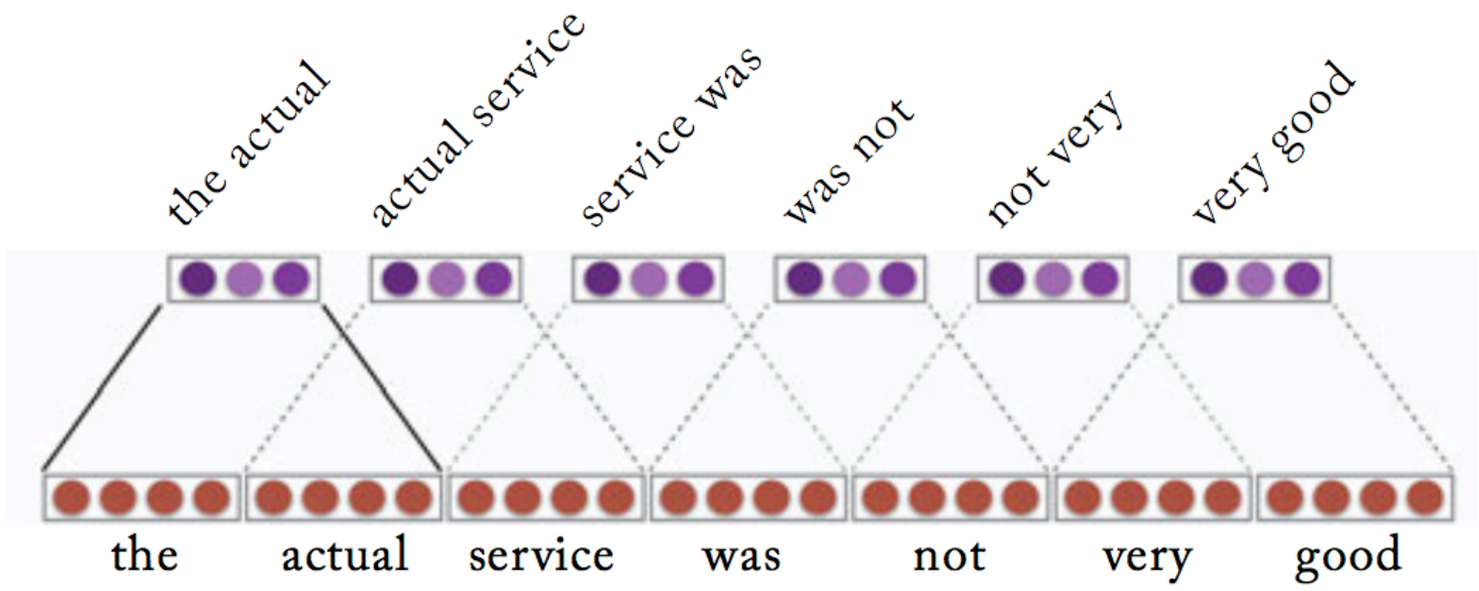


Things Covered So Far

NTU NLP

- Lecture 5: Window-based approach & Conv. Nets
  - Language modelling with window-based approach
  - Conv. Nets for encoding & classification





Where we are

NTU NLP

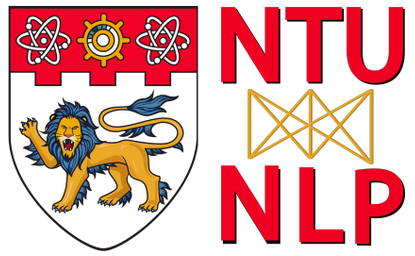
Models/Algorithms

- Linear models
- Feed-forward Neural Nets (FNN)
- Window-based methods
- Convolutional Nets
- **Recurrent Neural Nets**

NLP tasks/applications

- Word meaning
- **Language modelling**
- **Sequence tagging**
- **Sequence encoding**

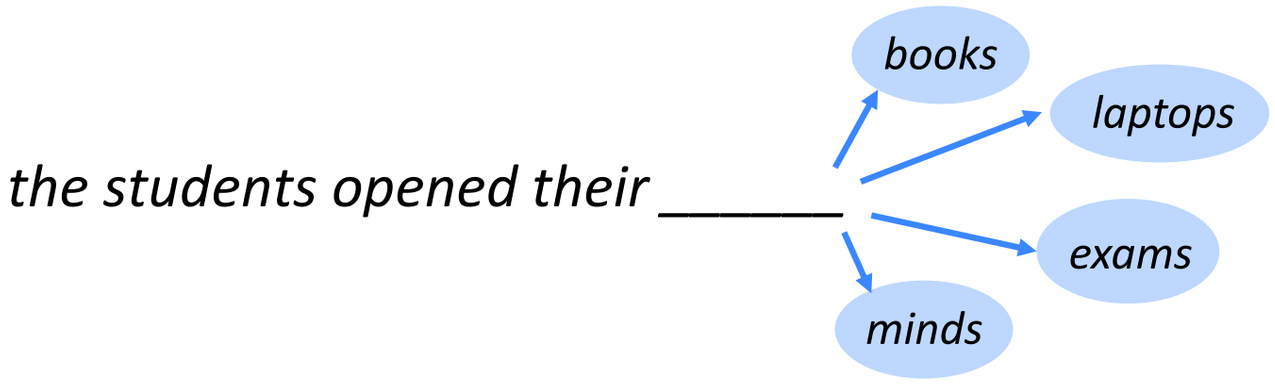
Lecture Plan

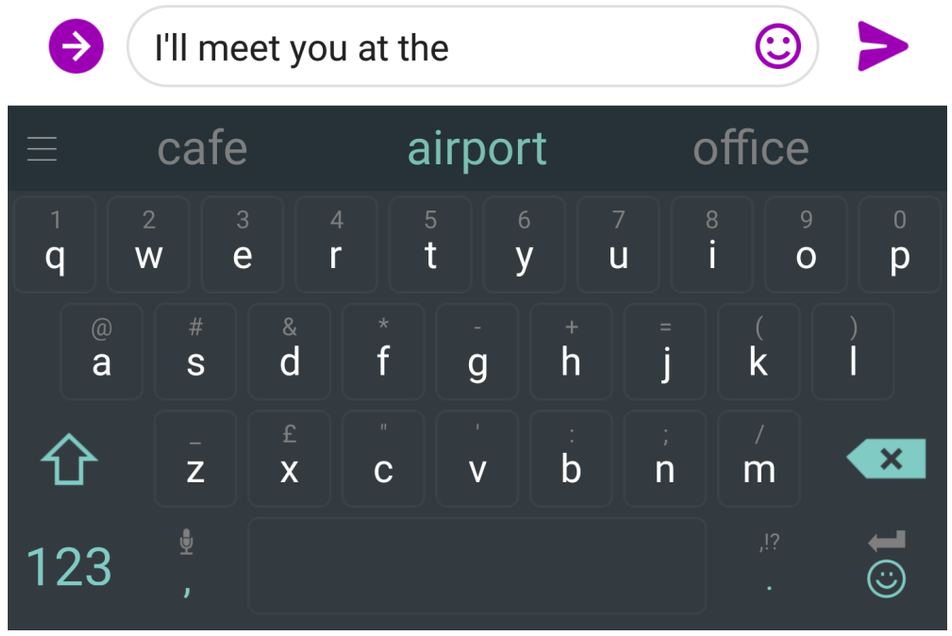


- Language Modeling
- Recurrent Neural Networks
- Backpropagation through time
- Text Generation with RNNs
- Sequence Tagging (NER, POS) with RNNs
- Sequence Classification with RNNs
- Bi-directional and stacked RNNs
- Gated RNNs (GRU, LSTM)

Language Modelling

A language model takes a list of words (history/context), and attempts to predict the word that follows them





Google

what is the

what is the **weather**
what is the **meaning of life**
what is the **dark web**
what is the **xfl**
what is the **doomsday clock**
what is the **weather today**
what is the **keto diet**
what is the **american dream**
what is the **speed of light**
what is the **bill of rights**

Google Search I'm Feeling Lucky

Source: Stanford 224n

Language Modelling

A language model takes a list of words (history/context), and attempts to predict the word that follows them

More formally: given a sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$, compute the probability distribution of the next word $\boldsymbol{x}^{(t+1)}$:

$$P(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

where $\boldsymbol{x}^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|V|}\}$

Source: Stanford 224n

Language Modeling

NTU NLP

Language Models (LM) also assigns a probability to a piece of text.

For example, if we have some text $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$, then the probability of this text (according to the Language Model) is:

$$P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}) = P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} | \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} | \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)})$$

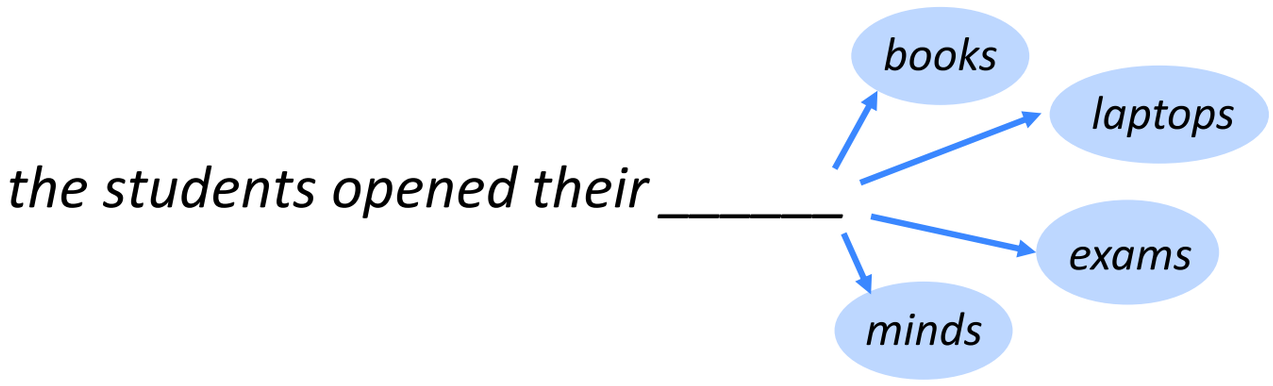
$$= \prod_{t=1}^{T} P(\boldsymbol{x}^{(t)} | \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)})$$

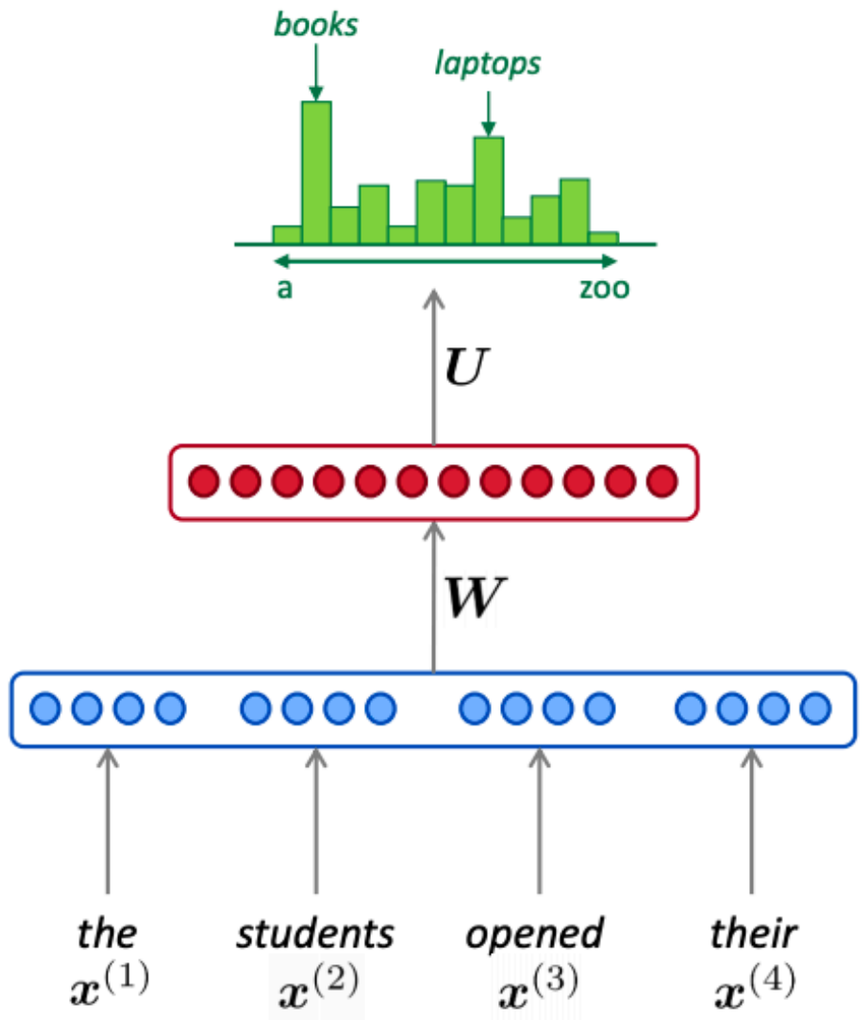
This is what our LM provides

Source: Stanford 224n

Language Modeling with Window Based Approach (Revisit)

A language model takes a list of words (history/context), and attempts to predict the word that follows them

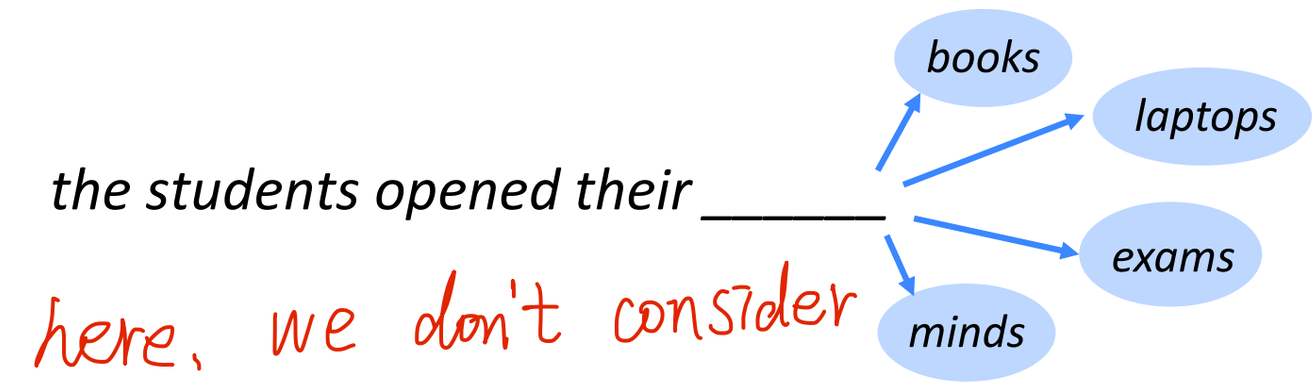


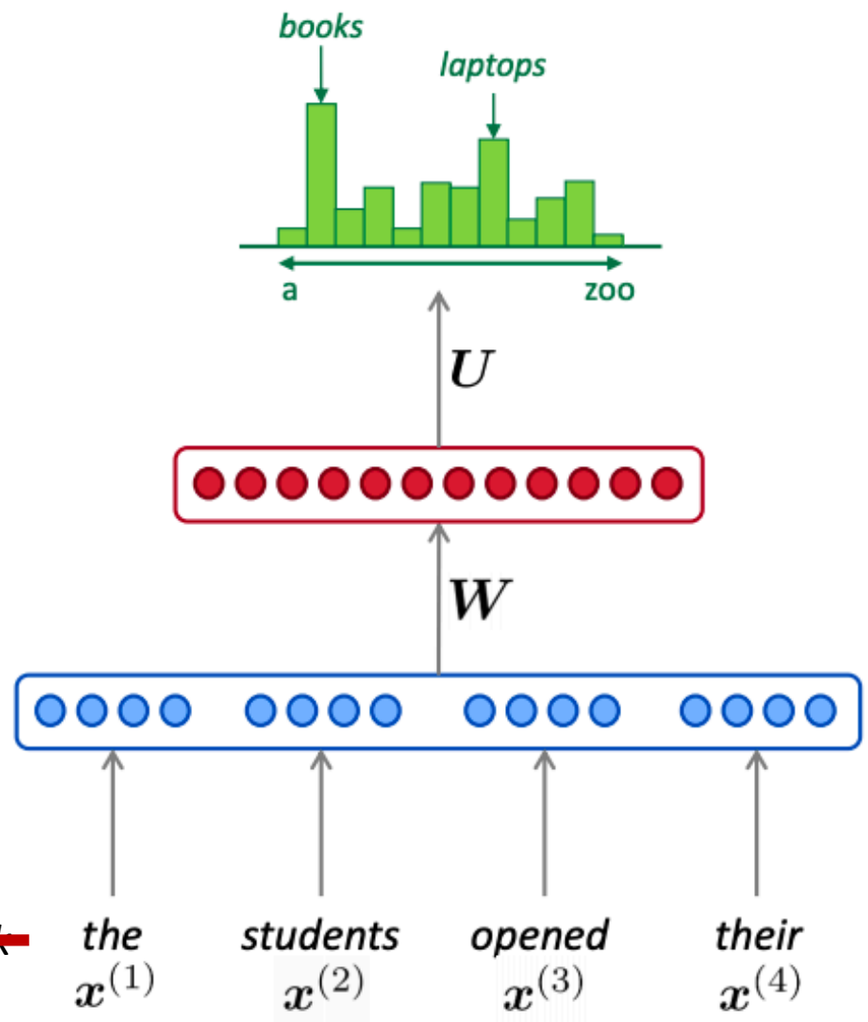


Language Modeling with Window Based Approach (Revisit)

NTU NLP

A language model takes a list of words (history/context), and attempts to predict the word that follows them





~~the proctor started the clock~~

discard

here, we don't consider the entire context.
if so our net architecture would be huge and in this case we should increase the volumn of dataset in case overfitting.

Source: Stanford 224n

Language Modeling with Window Based Approach (Revisit)

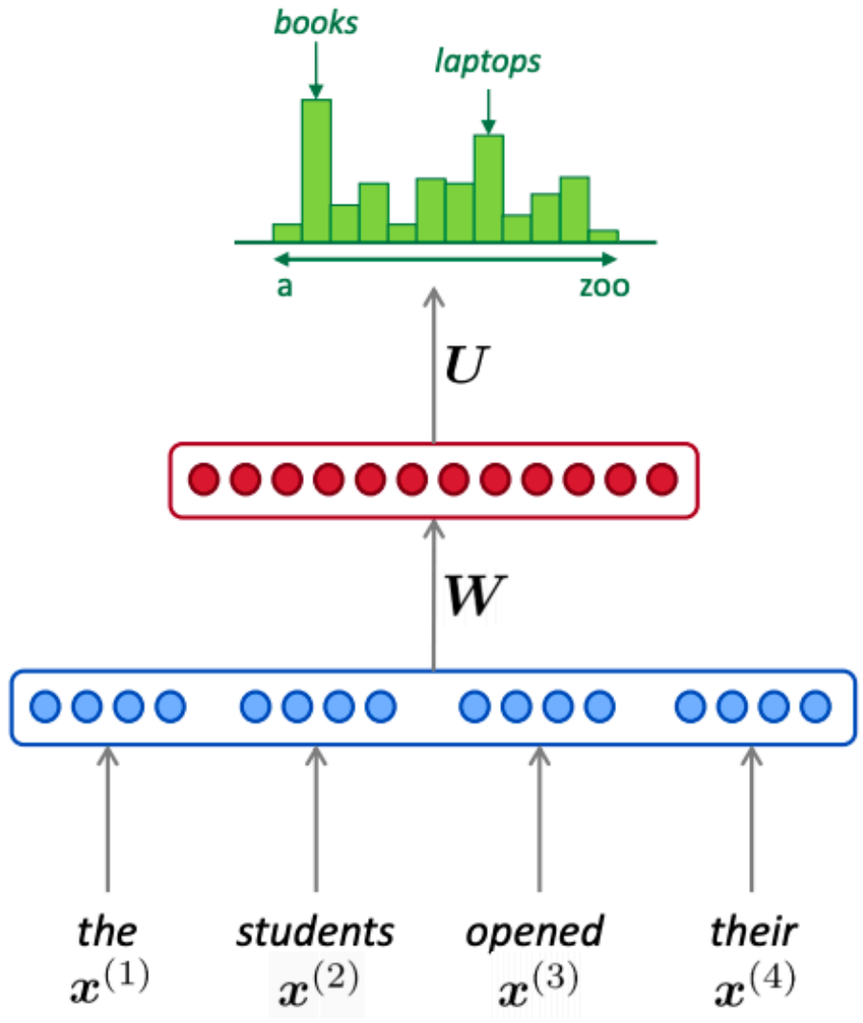
NTU NLP

Improvements over n-gram LM:

- No sparsity problem
- Don't need to store the n-grams

Remaining **problems**:

- Fixed window is too small
- Window can never be large enough! Enlarging window enlarges W
- Each input vector is multiplied by completely different weights. No symmetry in how the inputs are processed.
- Convolution? Not suitable for variable length sequences



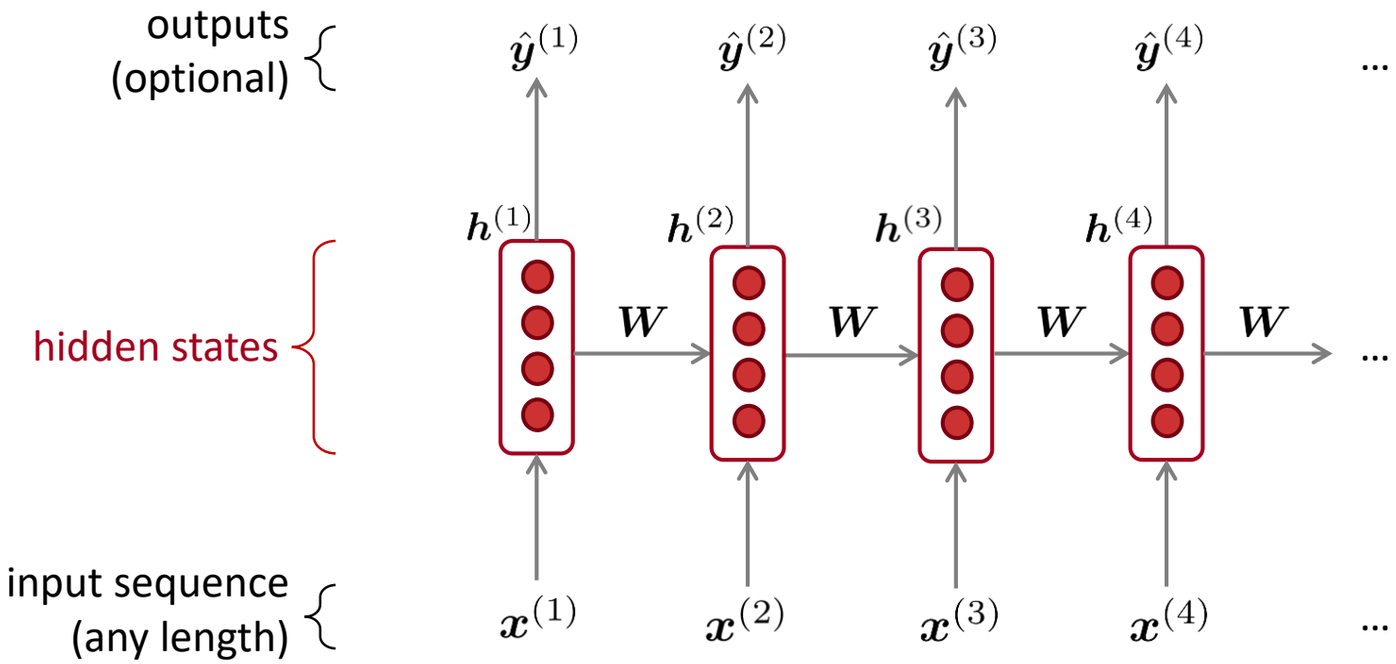
We need a neural architecture that can process any length input

Source: Stanford 224n

Recurrent Neural Networks

NTU NLP

Main idea: Apply the same weights repeatedly (recurrently)



Notice:

- Output at each step is optional
- Recurrence is done with hidden states

Source: Stanford 224n

A RNN Language Model

NTU NLP

output distribution

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

$\boldsymbol{h}^{(0)}$ is the initial hidden state

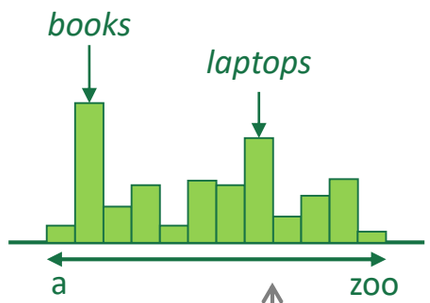
word embeddings

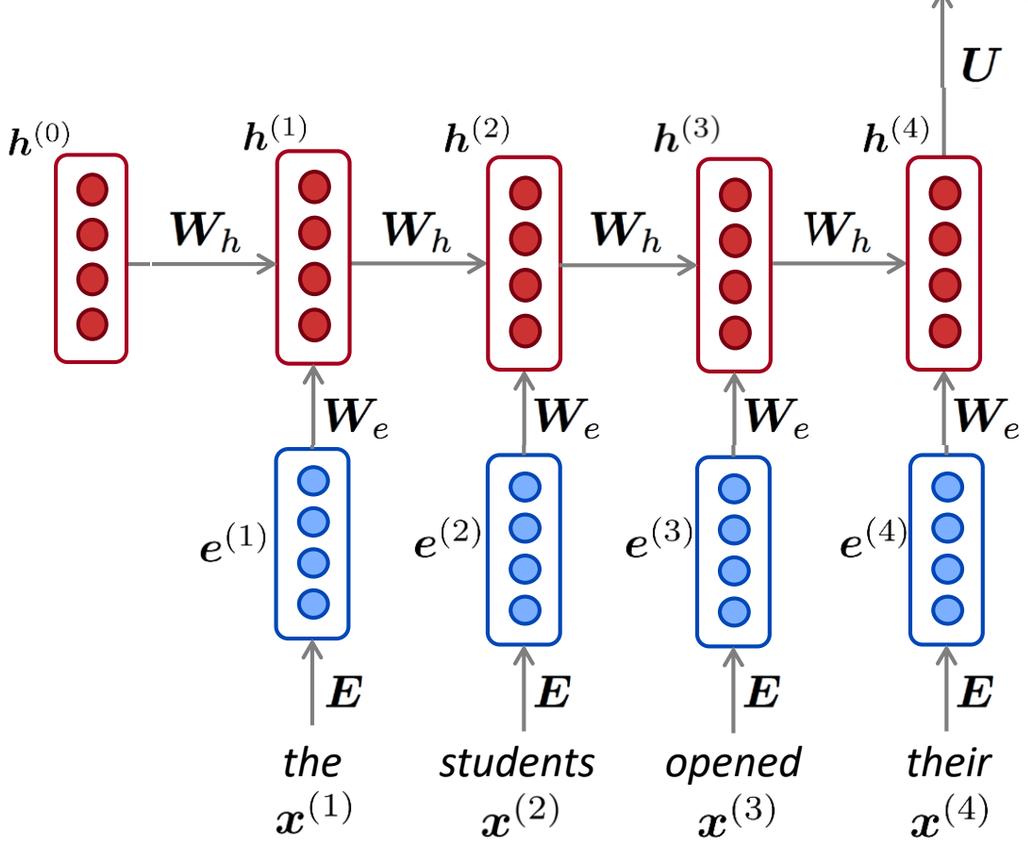
$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

words / one-hot vectors

$$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$$

$\hat{\boldsymbol{y}}^{(4)} = P(\boldsymbol{x}^{(5)}|\text{the students opened their})$

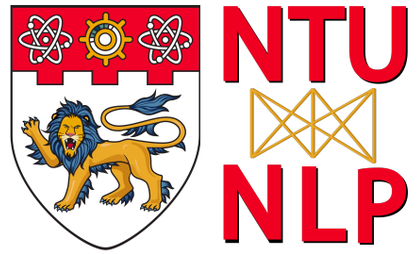




the input sequence can be of arbitrary length

Source: Stanford 224n

Language Modeling with RNNs

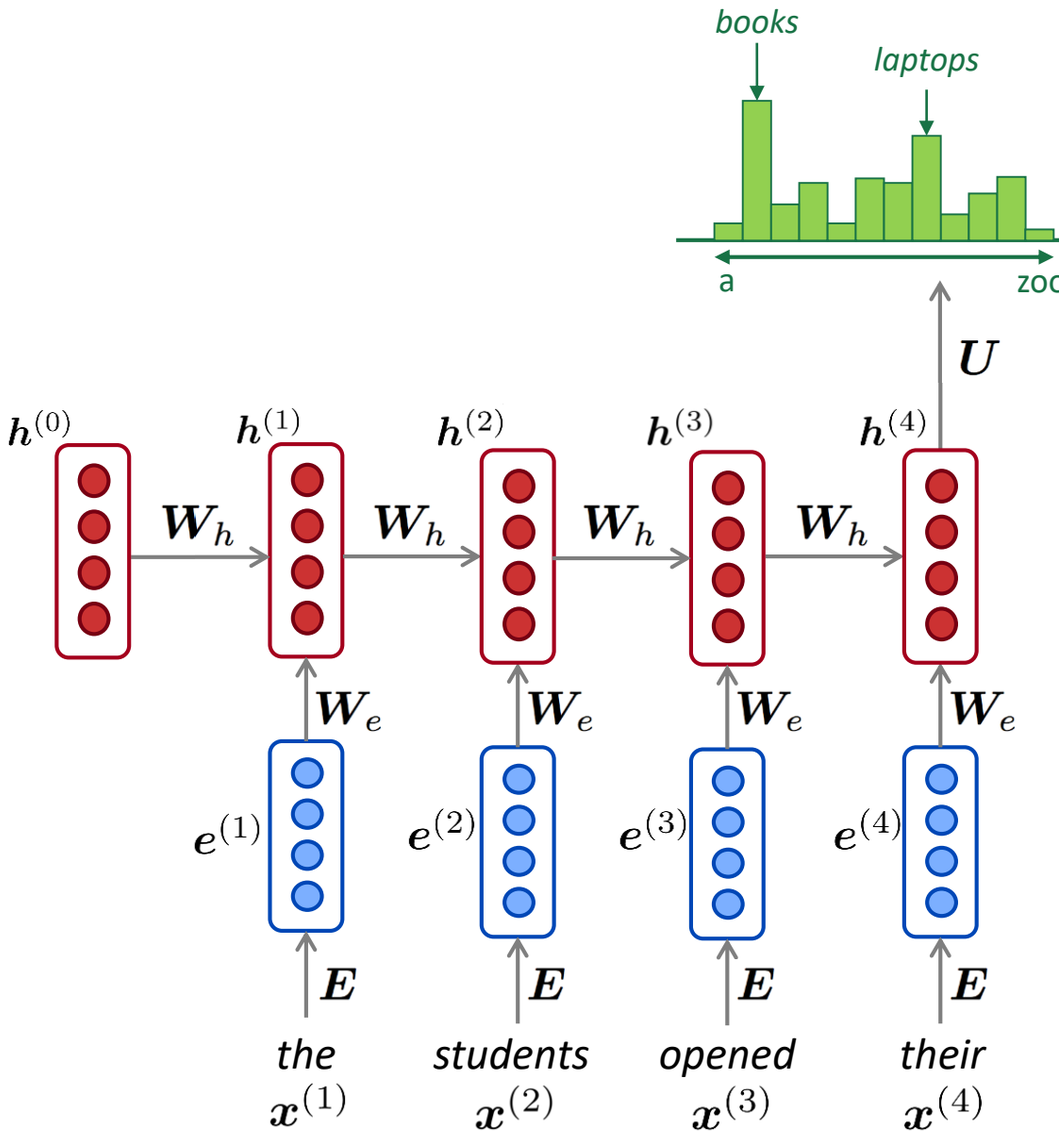


Advantages:

- Can process any length input
- Computation at step t can (in theory) use information from many steps back
- Model size doesn't increase for longer input
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

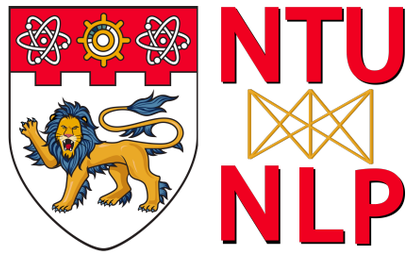
Problems:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back (remembers only recent states)



Source: Stanford 224n

Training a RNN-LM



- Get a big corpus of text which is a sequence of words $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$
- Feed into RNN-LM; compute output distribution $\hat{\boldsymbol{y}}^{(t)}$ for *every step t*.
  - i.e. predict probability dist of *every word*, given words so far

- Loss function on step *t* is cross-entropy between predicted probability distribution $\hat{\boldsymbol{y}}^{(t)}$, and the true next word $\boldsymbol{y}^{(t)}$ (one-hot for $\boldsymbol{x}^{(t+1)}$):

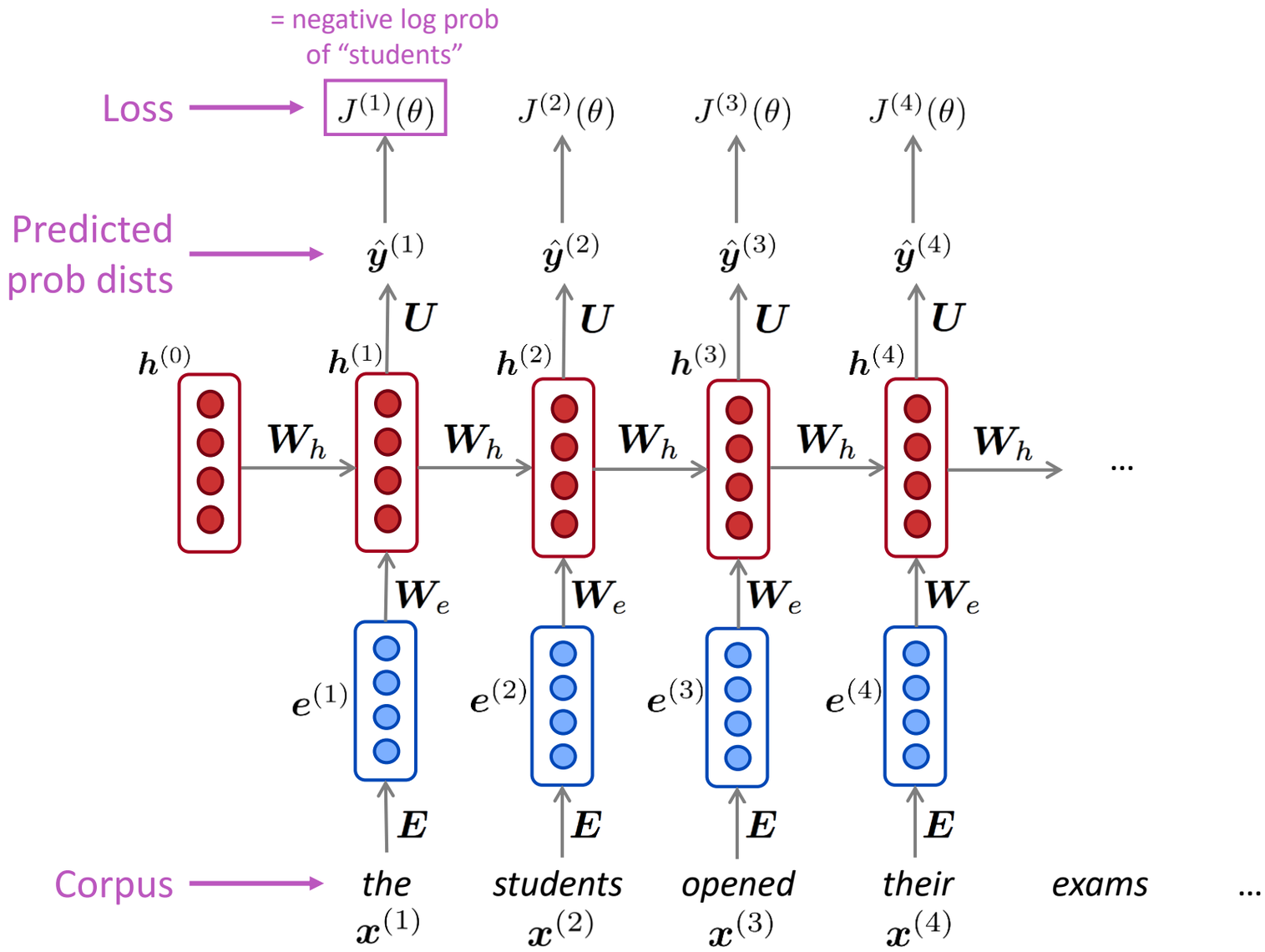
$$J^{(t)}(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{w \in V} \boldsymbol{y}_w^{(t)} \log \hat{\boldsymbol{y}}_w^{(t)} = -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

- Average this to get overall loss for entire training set:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

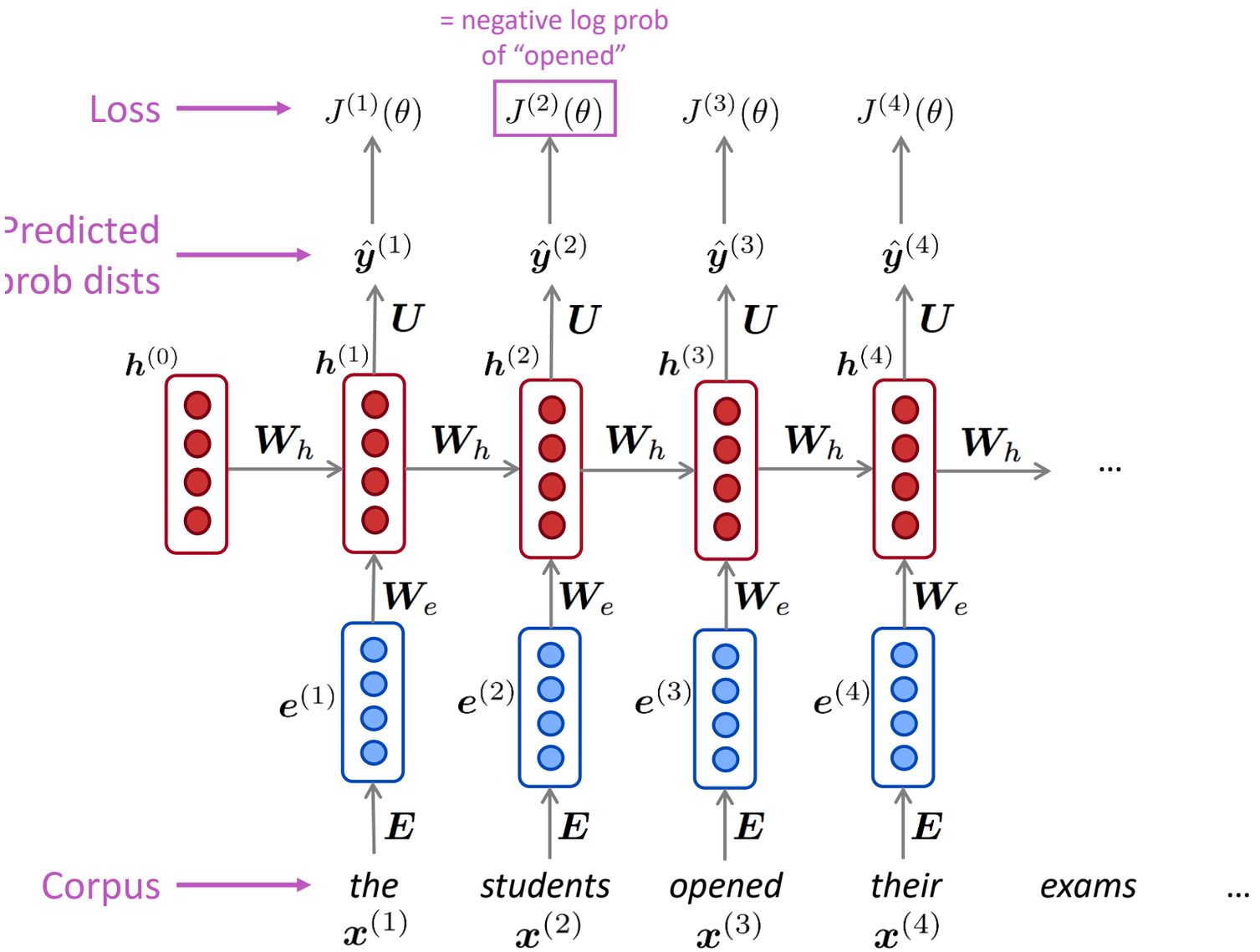
Source: Stanford 224n

Training a RNN-LM



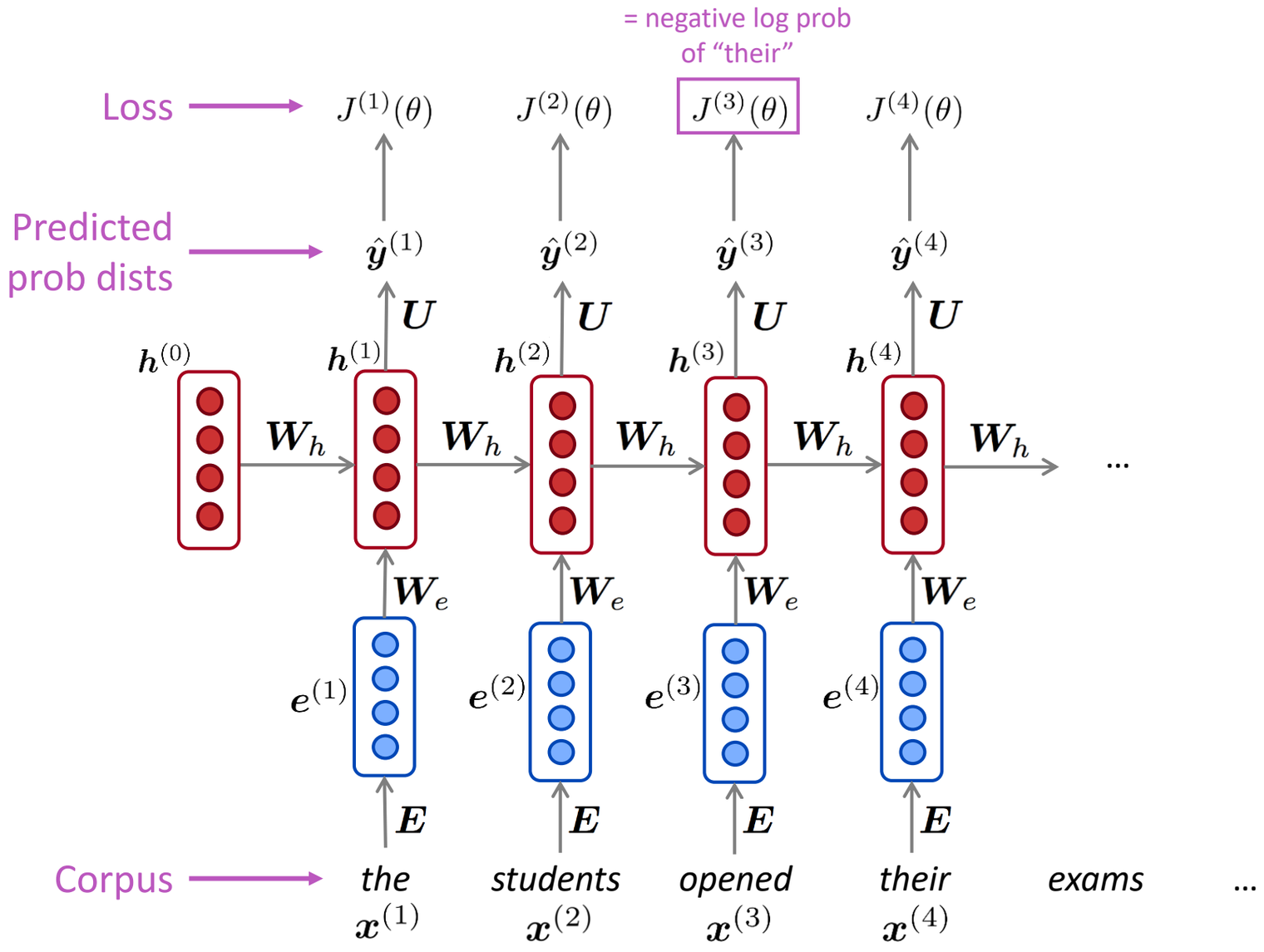
Source: Stanford 224n

Training a RNN-LM



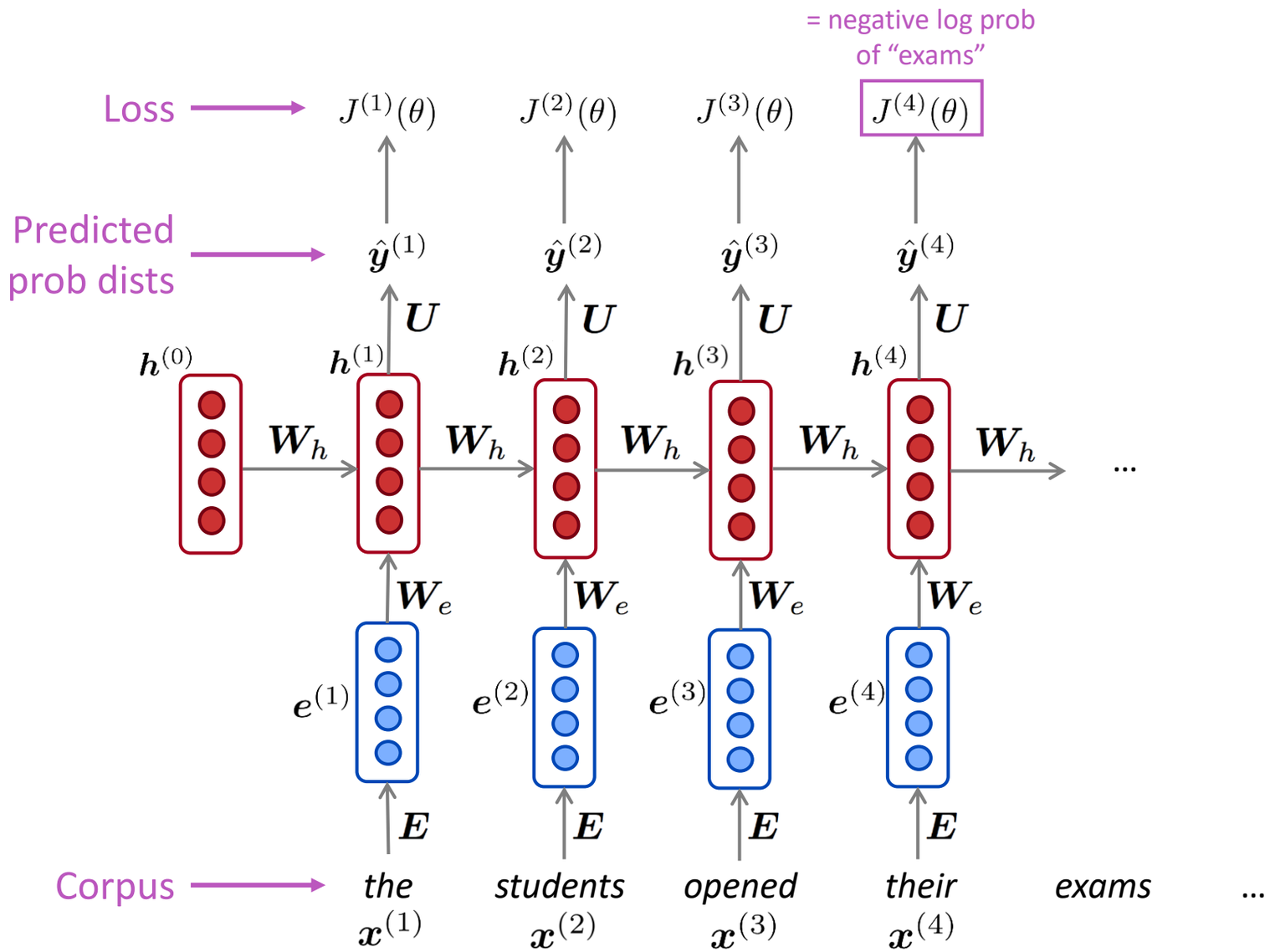
Source: Stanford 224n

Training a RNN-LM



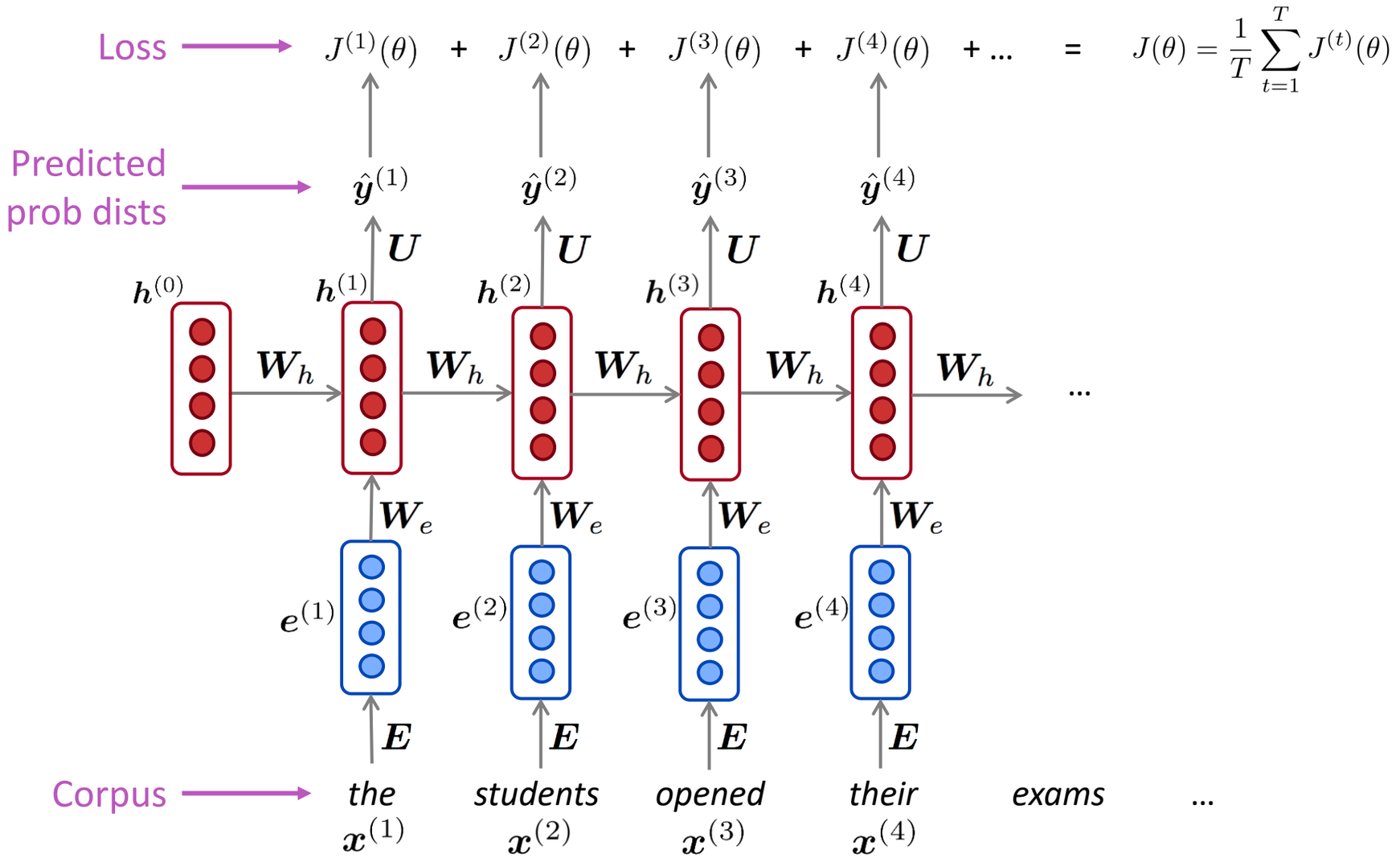
Source: Stanford 224n

Training a RNN-LM



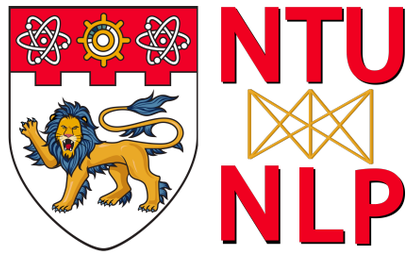
Source: Stanford 224n

Training a RNN-LM



Source: Stanford 224n

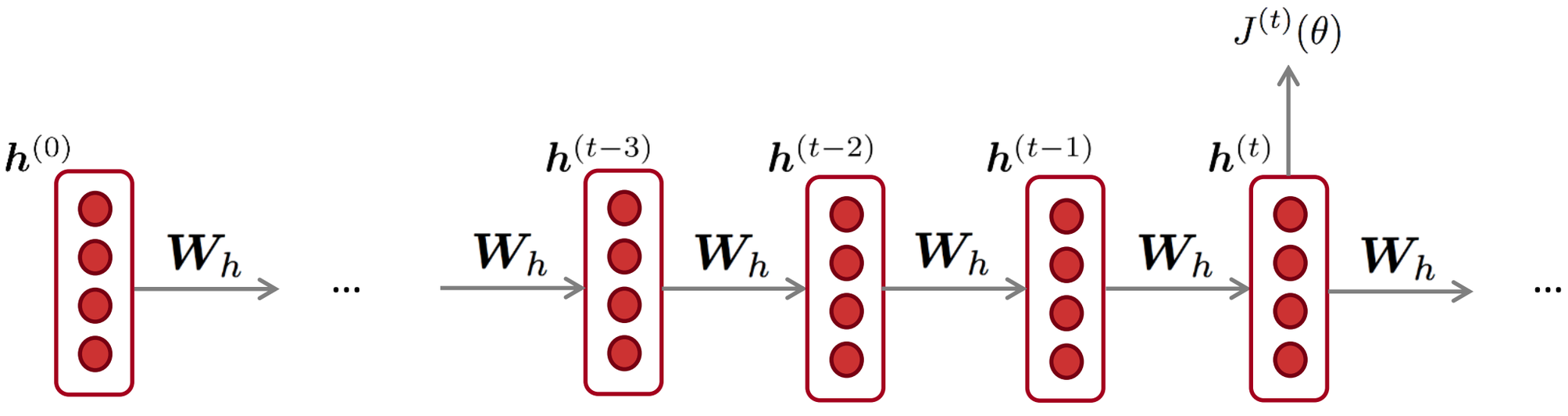
Training a RNN-LM



- However: Computing loss and gradients across the entire corpus is way too expensive (no parallelisation)
- In practice, consider as a sentence (or a fixed length sequence)
- In SGD, we compute loss and gradients for batches and update, and then repeat

Source: Stanford 224n

Backpropagation for RNNs



The gradient w.r.t. a repeated weight is the sum of the gradient w.r.t. each time it appears

$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}}\bigg|_{(i)}$$

Source: Stanford 224n

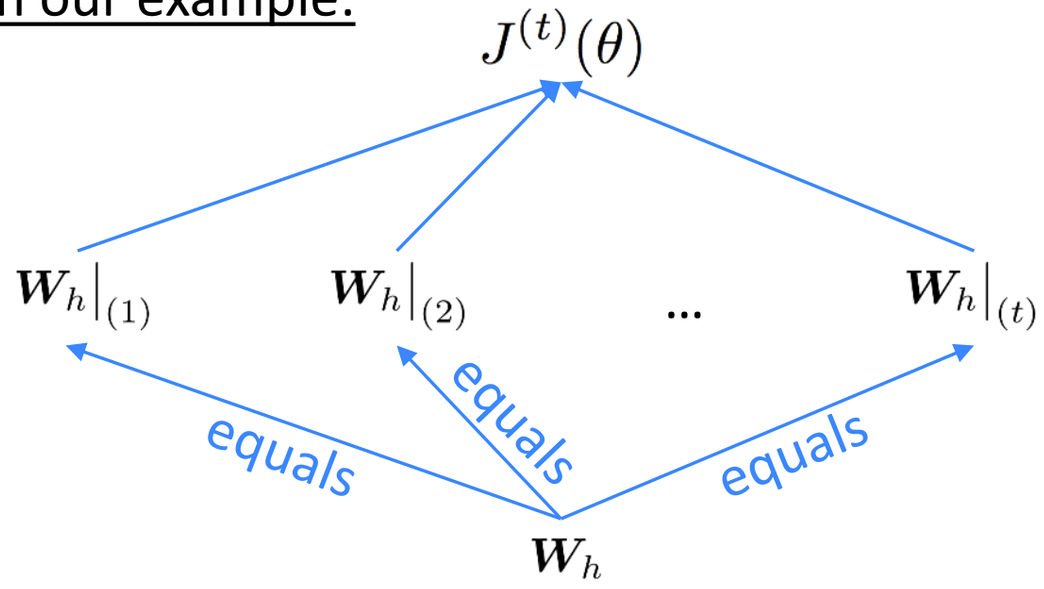
Backpropagation for RNNs

NTU NLP

- Given a multivariable function $f(x, y)$, and two single variable functions $x(t)$ and $y(t)$, here's what the multivariable chain rule says:

$$\underbrace{\frac{d}{dt} f(x(t), y(t))}_{\text{Derivative of composition function}} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$

In our example:



Apply the multivariable chain rule:

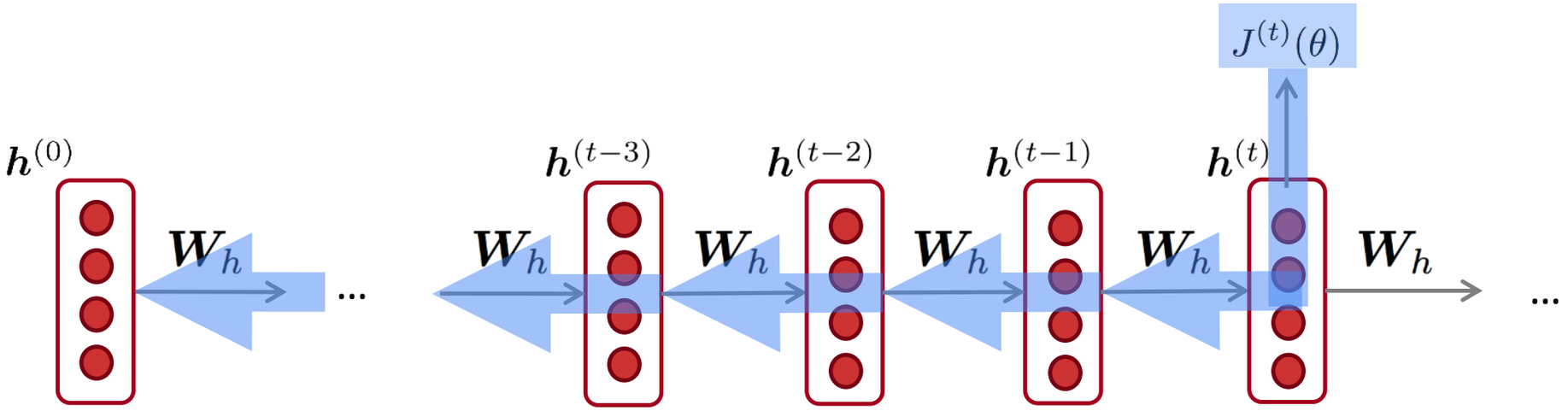
$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W}_h} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W}_h}\bigg|_{(i)} \frac{\partial \boldsymbol{W}_h|_{(i)}}{\partial \boldsymbol{W}_h}$$

= 1

$$= \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W}_h}\bigg|_{(i)}$$

Source: Stanford 224n

Backpropagation for RNNs

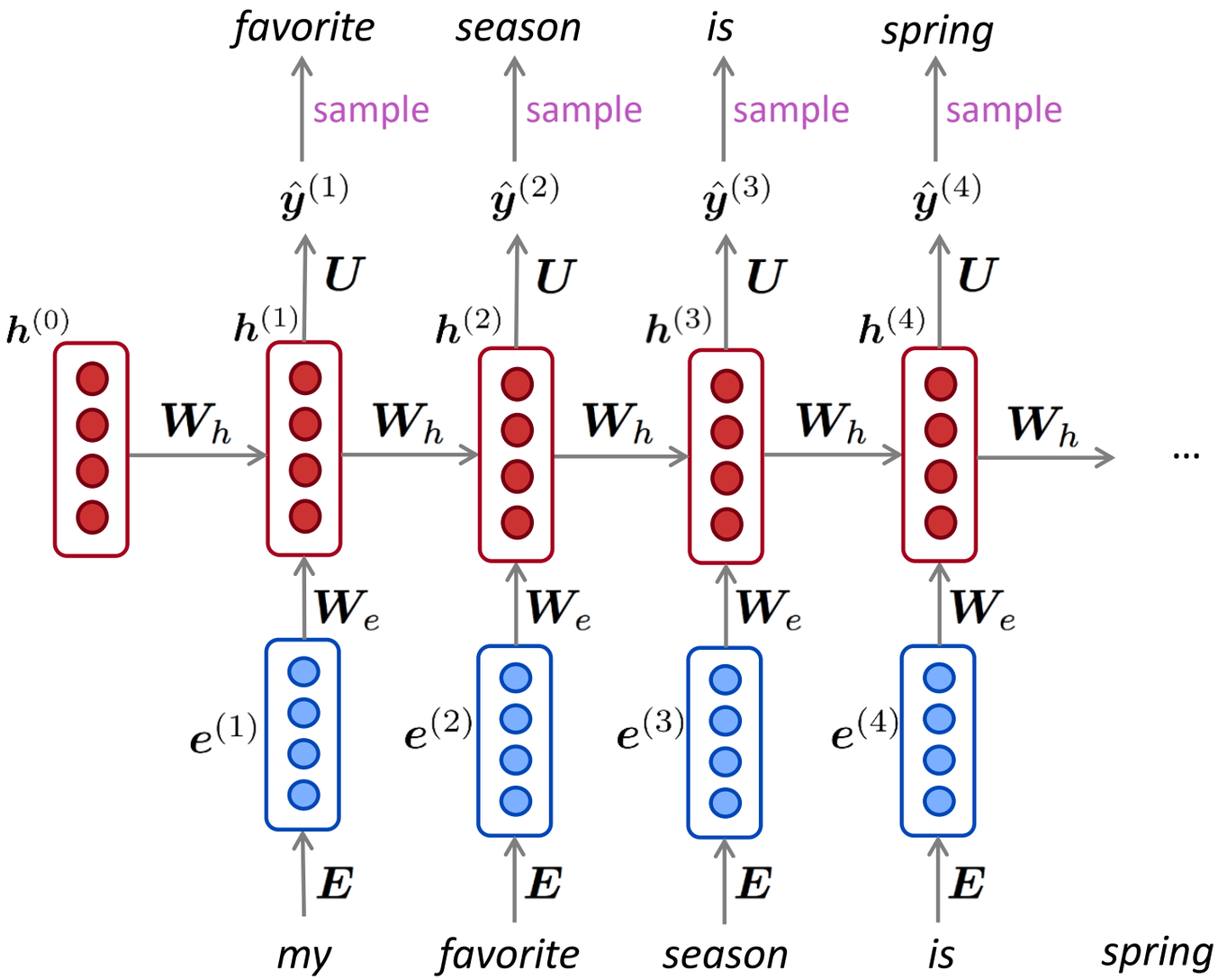


Backpropagate over timesteps i=t,...,0, summing gradients as you go.

This algorithm is called **"backpropagation through time"**

Source: Stanford 224n

Generating Text with RNN-LM



Just like a n-gram LM, you can use an RNN-LM to generate text by repeated sampling. Sampled output is next step's input.

Source: Stanford 224n

Fun with RNN Generation

NTU NLP

Char-RNN trained on Obama Speeches

SEED: Jobs

Good afternoon. God bless you.

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.

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

Fun with RNN Generation

NTU NLP

Char-RNN trained on Obama Speeches

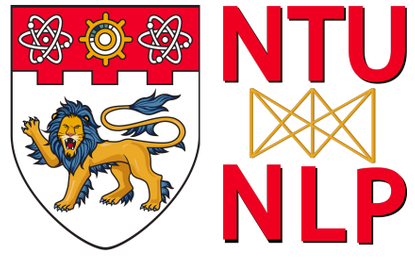
SEED: Democracy

Good morning. One of the borders will be able to continue to be here today. We have to say that the partnership was a partnership with the American people and the street continually progress that is a process and distant lasting peace and support that they were supporting the work of concern in the world. They were in the streets and communities that could have to provide steps to the people of the United States and Afghanistan. In the streets — the final decade of the country that will include the people of the United States of America. Now, humanitarian crisis has already rightly achieved the first American future in the same financial crisis that they can find reason to invest in the world.

Thank you very much. God bless you. God bless you. Thank you.

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

Fun with RNN Generation



Char-RNN trained on Harry Potter

Part 1

"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."

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

Fun with RNN Generation

NTU NLP

Char-RNN trained on Food Recipe

"MMMMM----- Recipe via Meal-Master (tm) v8.05

Title: CARAMEL CORN GARLIC BEEF
Categories: Soups, Desserts
Yield: 10 Servings

2 tb Parmesan cheese, ground
1/4 ts Ground cloves
-- diced
1 ts Cayenne pepper

Cook it with the batter. Set aside to cool. Remove the peanut oil in a small saucepan and pour into the margarine until they are soft. Stir in a
a mixer (dough). Add the chestnuts, beaten egg whites, oil, and salt
and brown sugar and sugar; stir onto the boqtly brown it.

The recipe from an oiled by fried and can. Beans, by Judil Cookbook, Source: Pintore, October, by Chocolates, Breammons of Jozen, Empt.com

And many more ...

Source: https://gist.github.com/nylki/1efbaa36635956d35bcc

Evaluating Language Models

NTU NLP

We use perplexity as the standard evaluation metric for LMs

$$\text{perplexity} = \prod_{t=1}^{T} \left(\frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)^{1/T}$$

Normalized by number of words

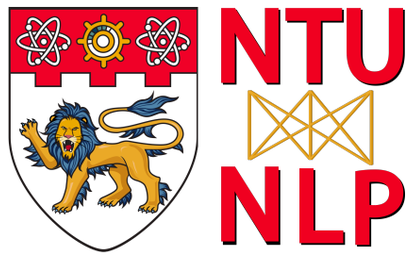
Inverse probability of corpus, according to Language Model

This is equal to the exponential of the cross-entropy loss

$$= \prod_{t=1}^{T} \left(\frac{1}{\hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}}} \right)^{1/T} = \exp\left(\frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}} \right) = \exp(J(\theta))$$

Source: Stanford 224n

Language Modeling Results

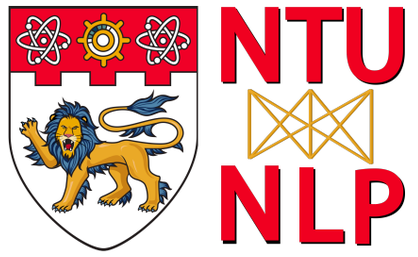


RNNs improve results quite significantly

Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

Source: https://research.fb.com/building-an-efficient-neural-language-model-over-a-billion-words/

Why Language Modeling is Important



- A benchmark task to track our progress on understanding language
- An important component of many NLP tasks, especially those involving generating text or estimating the probability of a text

- Speech recognition
- Spelling/grammar correction
- Authorship identification
- Machine translation
- Summarization
- Dialogue
- etc.

(BERT)
• to learn contextual embedding in unsupervised way.

Source: Stanford 224n

Lecture Plan

NTU NLP

- Language Modeling
- Recurrent Neural Networks
- Backpropagation through time
- Text Generation with RNNs

not just Language Modeling.

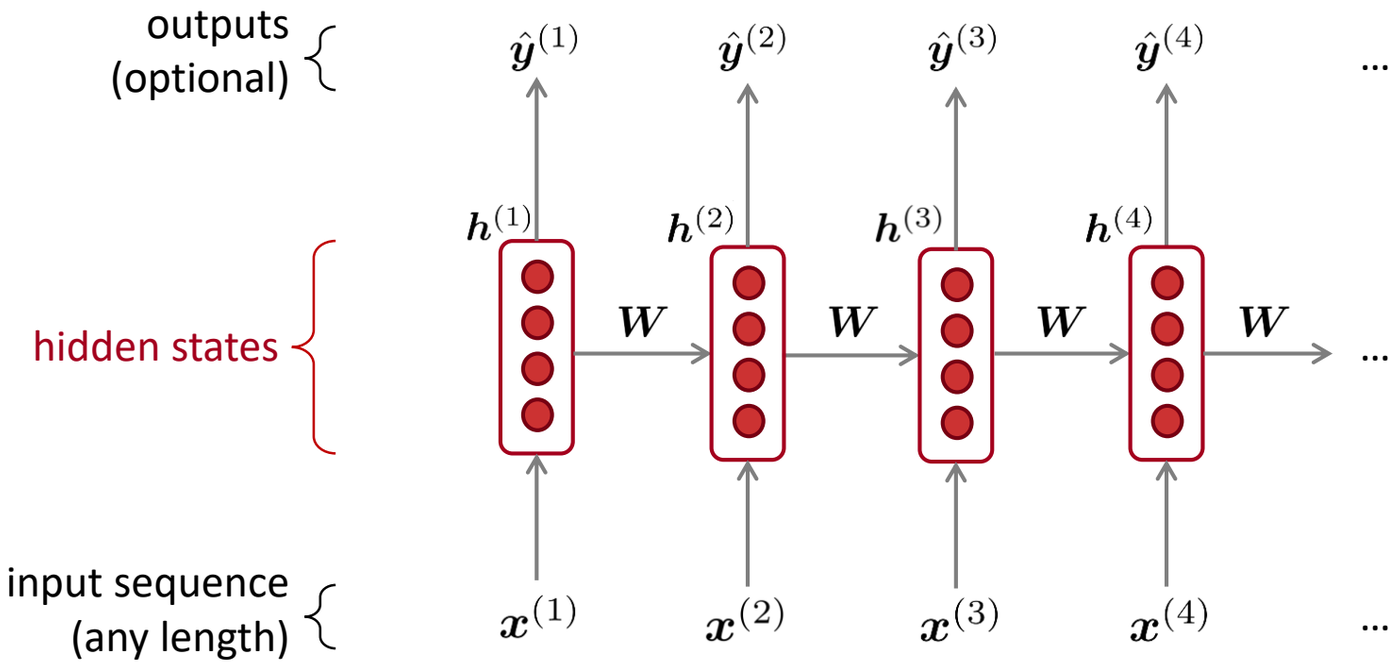
general model can used in many other purposes

- Sequence Tagging (NER, POS) with RNNs
- Sequence Classification with RNNs
- Bi-directional and stacked RNNs
- Gated RNNs (GRU, LSTM)

Recurrent Neural Networks (Revisit)

NTU NLP

Main idea: Apply the same weights repeatedly (recurrently)



Notice:

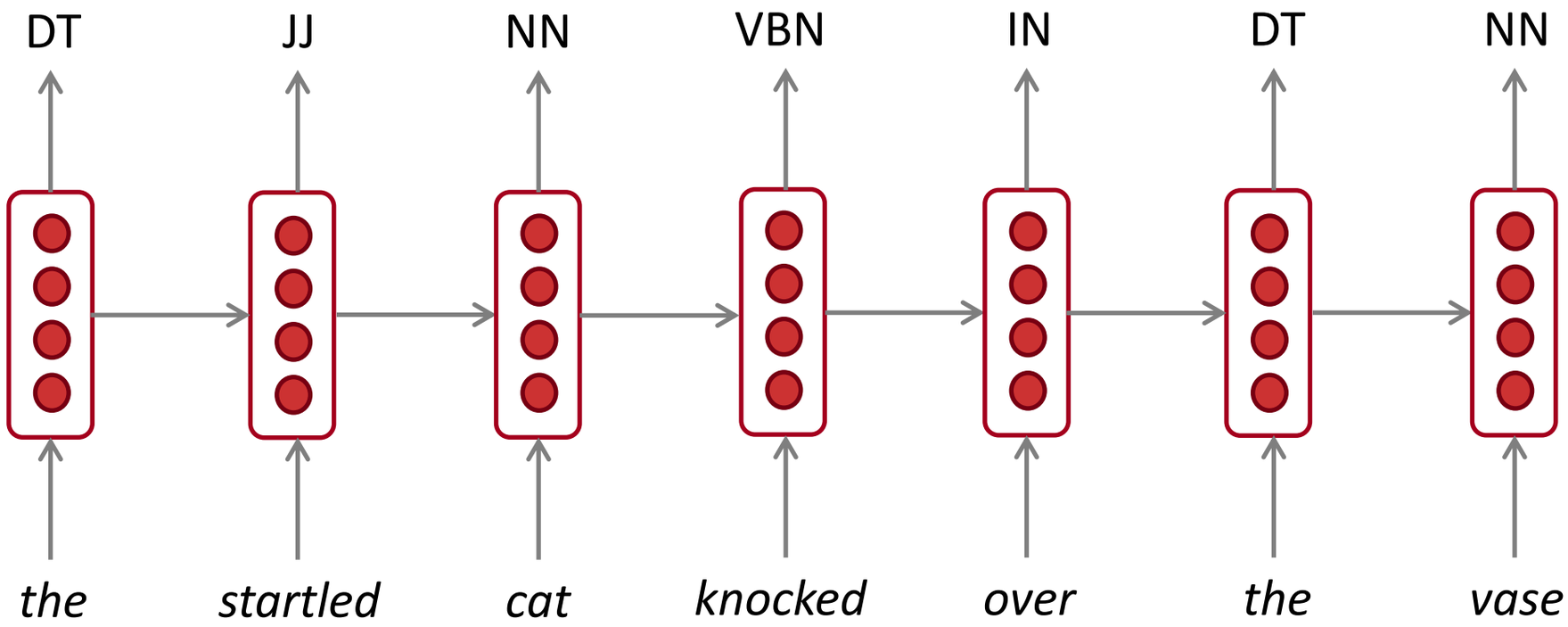
- Output at each step is optional
- Recurrence is done with hidden states

Source: Stanford 224n

Sequence Tagging with RNNs

NTU NLP

- POS Tagging:

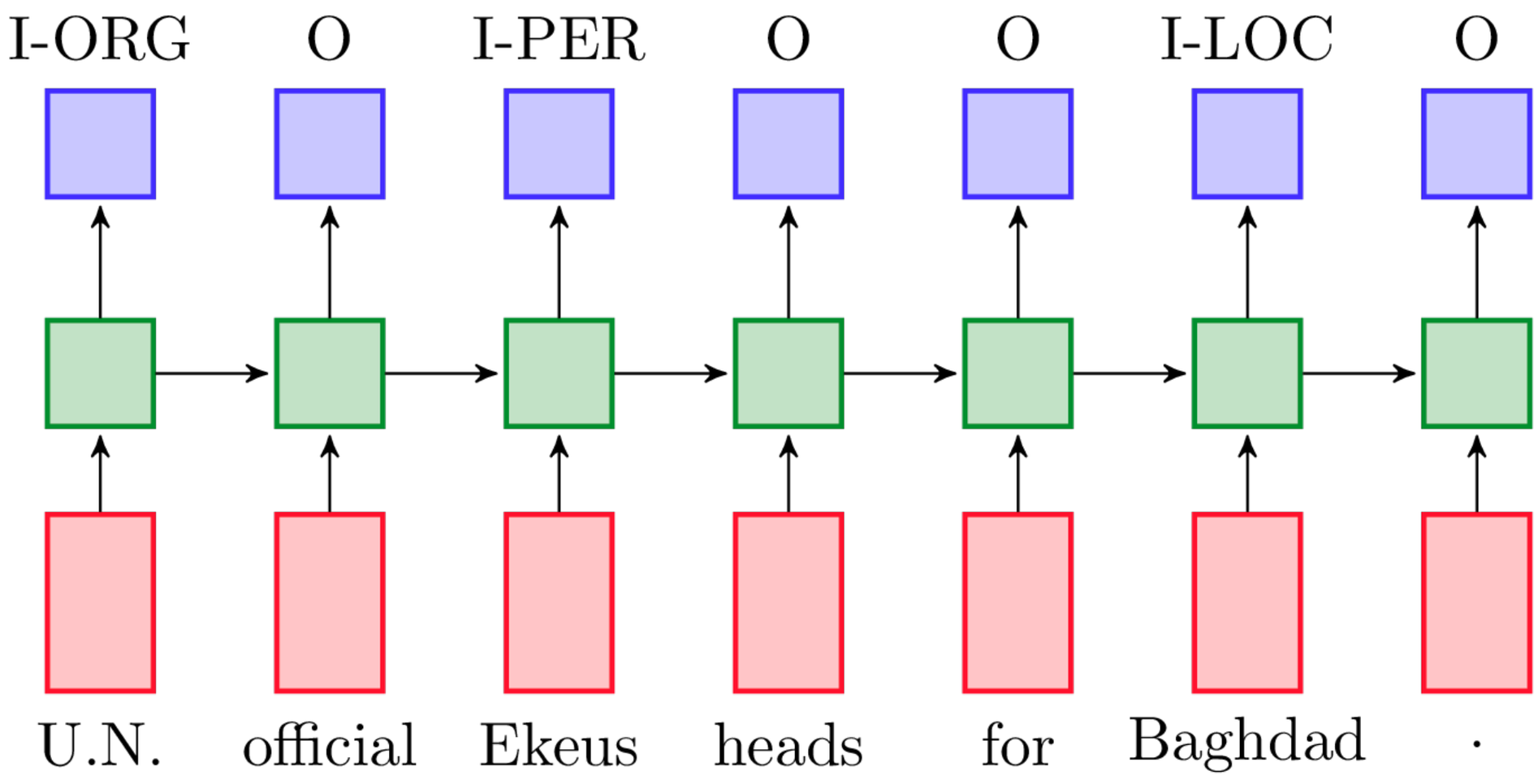


Source: Stanford 224n

Sequence Tagging with RNNs

NTU NLP

- NER Tagging:

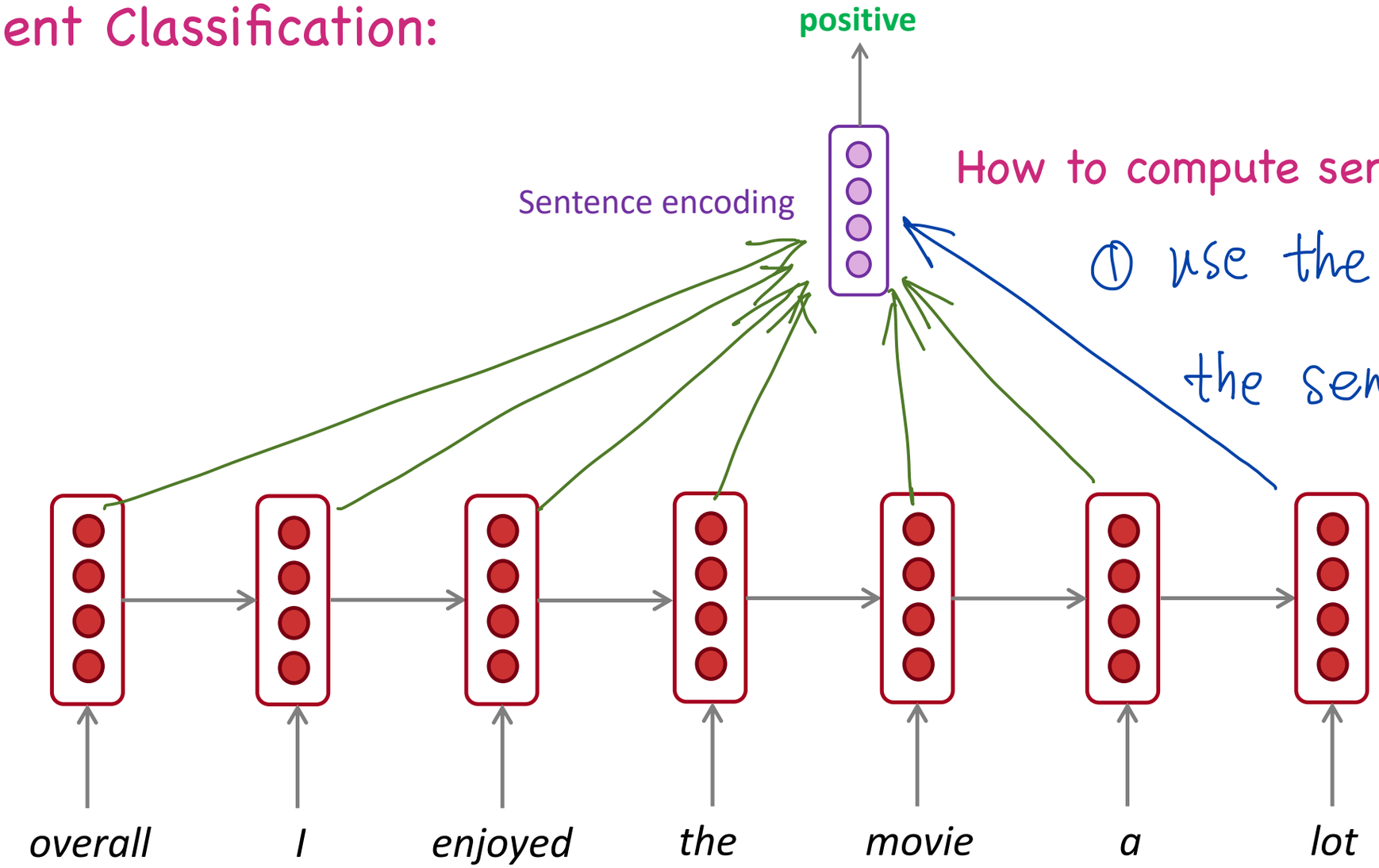


Source: https://www.depends-on-the-definition.com/guide-sequence-tagging-neural-networks-python/

Sequence Classification with RNNs

NTU NLP

- Sentiment Classification:



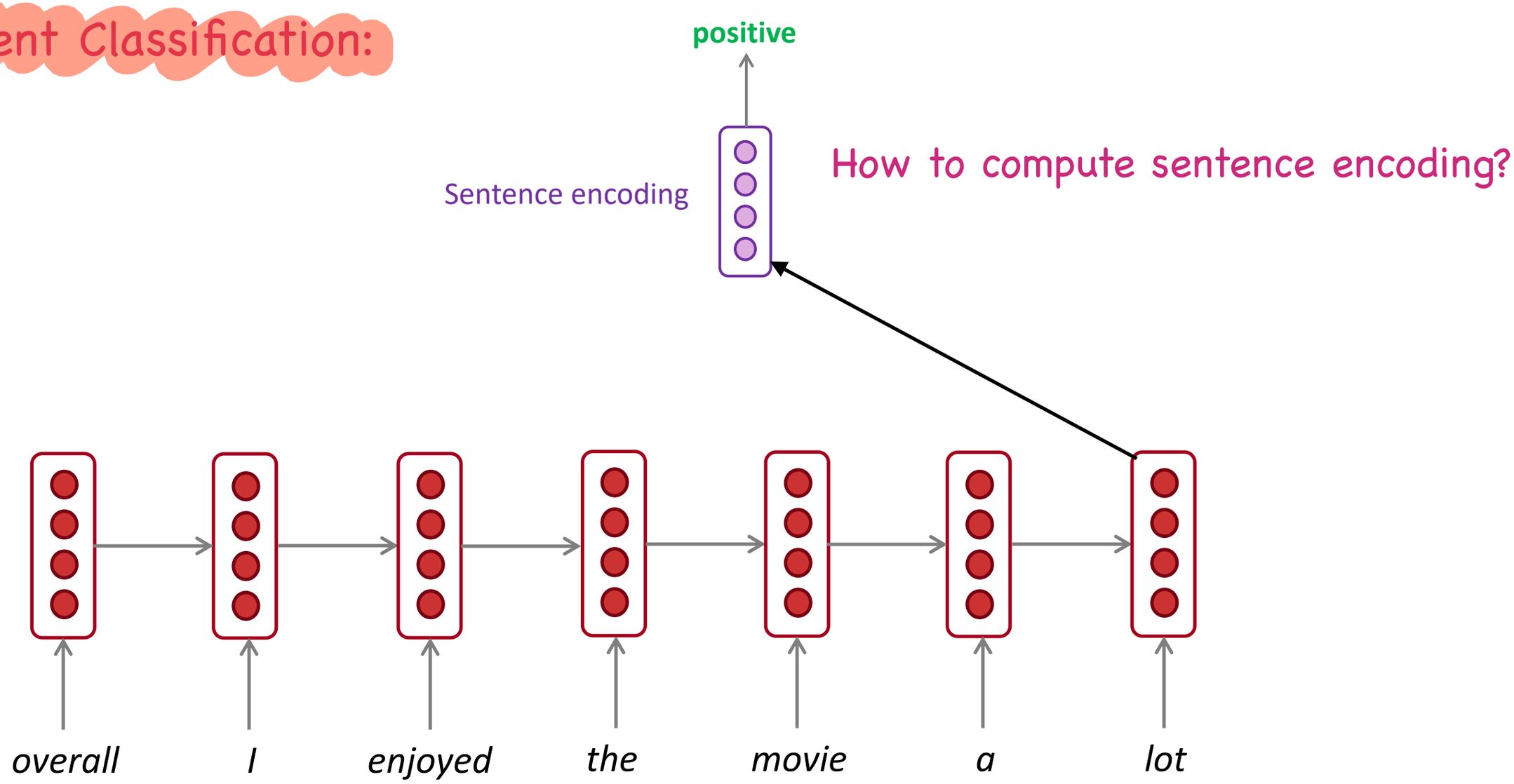
How to compute sentence encoding?

① use the last step as the sentence representation

Source: Stanford 224n

Sequence Classification with RNNs

- Sentiment Classification:

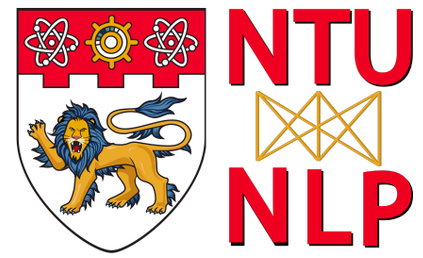


First choice:

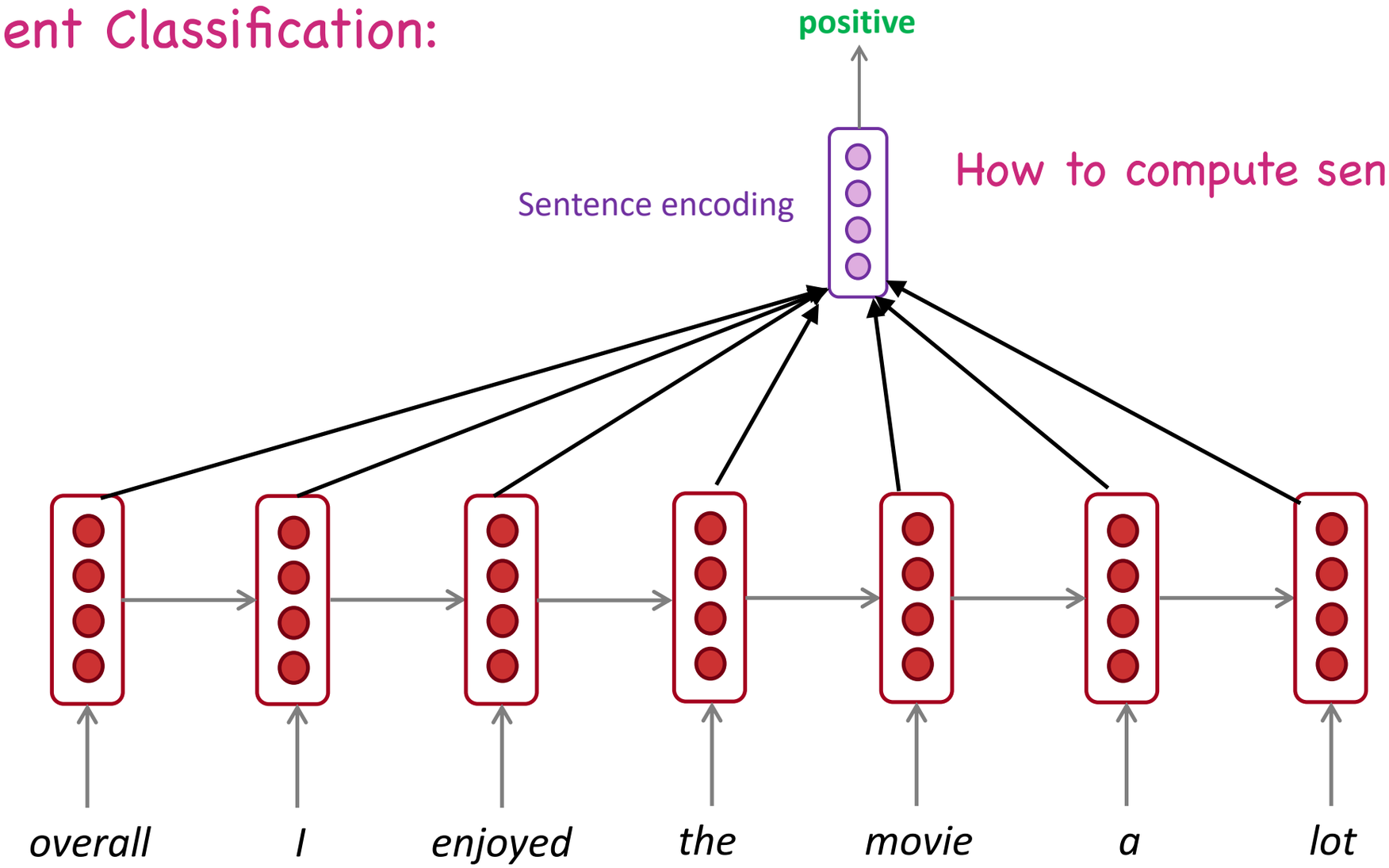
Use the final hidden state as the summary of the sentence

Source: Stanford 224n

Sequence Classification with RNNs



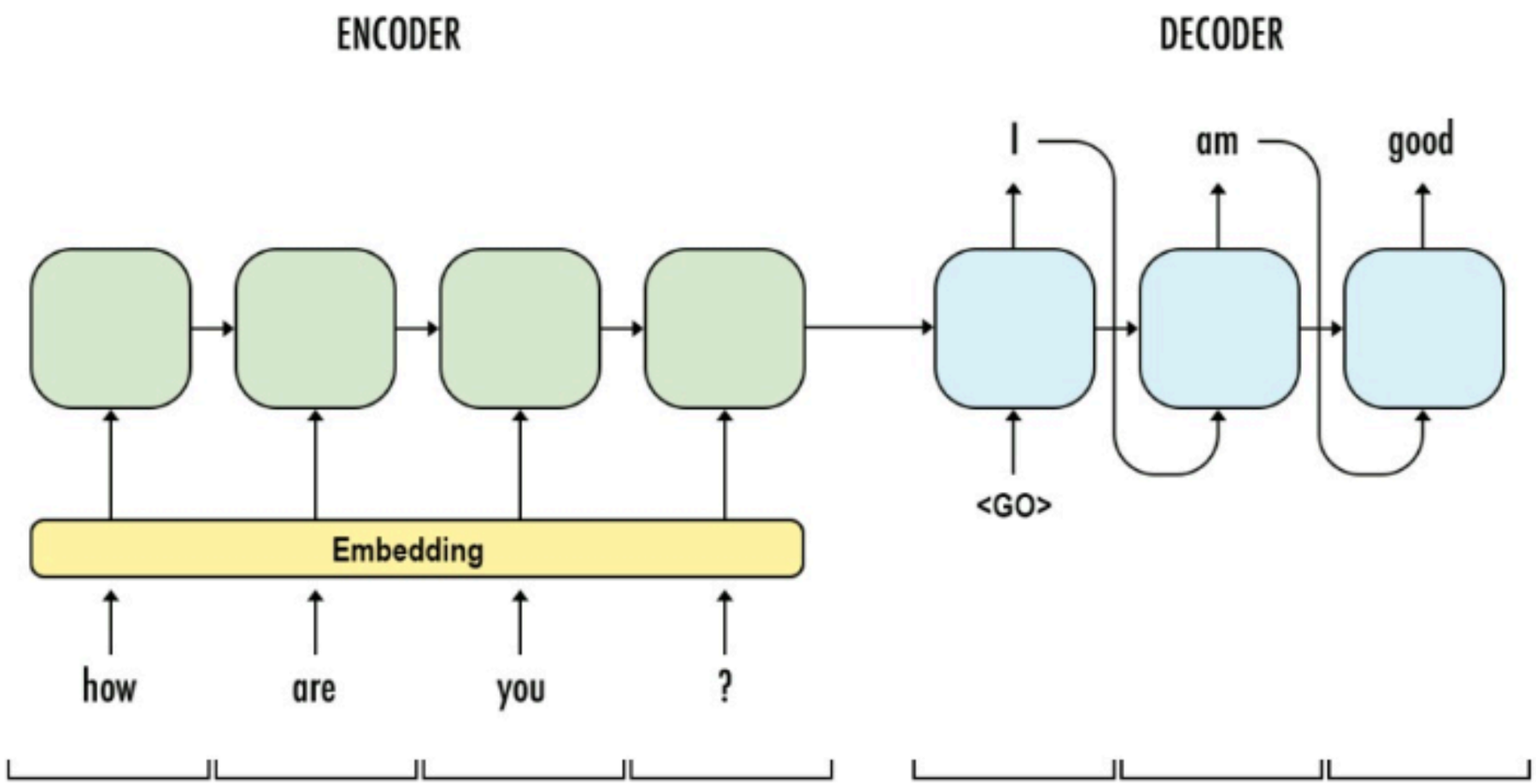
- Sentiment Classification:



2nd choice: Use max/average pooling as the summary of the sentence

Source: Stanford 224n

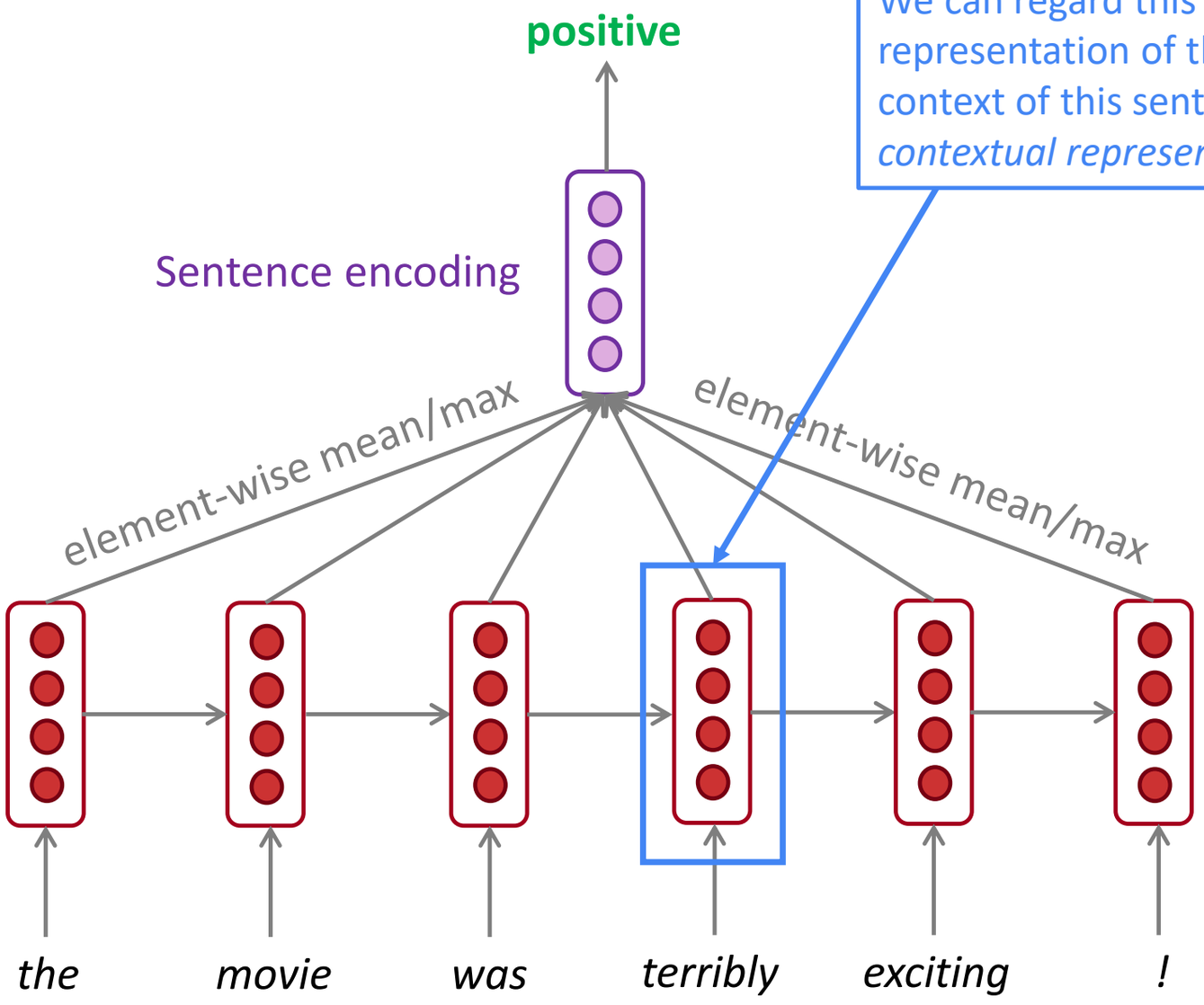
RNNs as General Purpose Encoders/Decoders



- We will see more details on this when we cover Seq2Seq model and Neural Machine Translation

Source: Stanford 224n

Bi-directional RNN: Motivation



We can regard this hidden state as a representation of the word *"terribly"* in the context of this sentence. We call this a *contextual representation.*

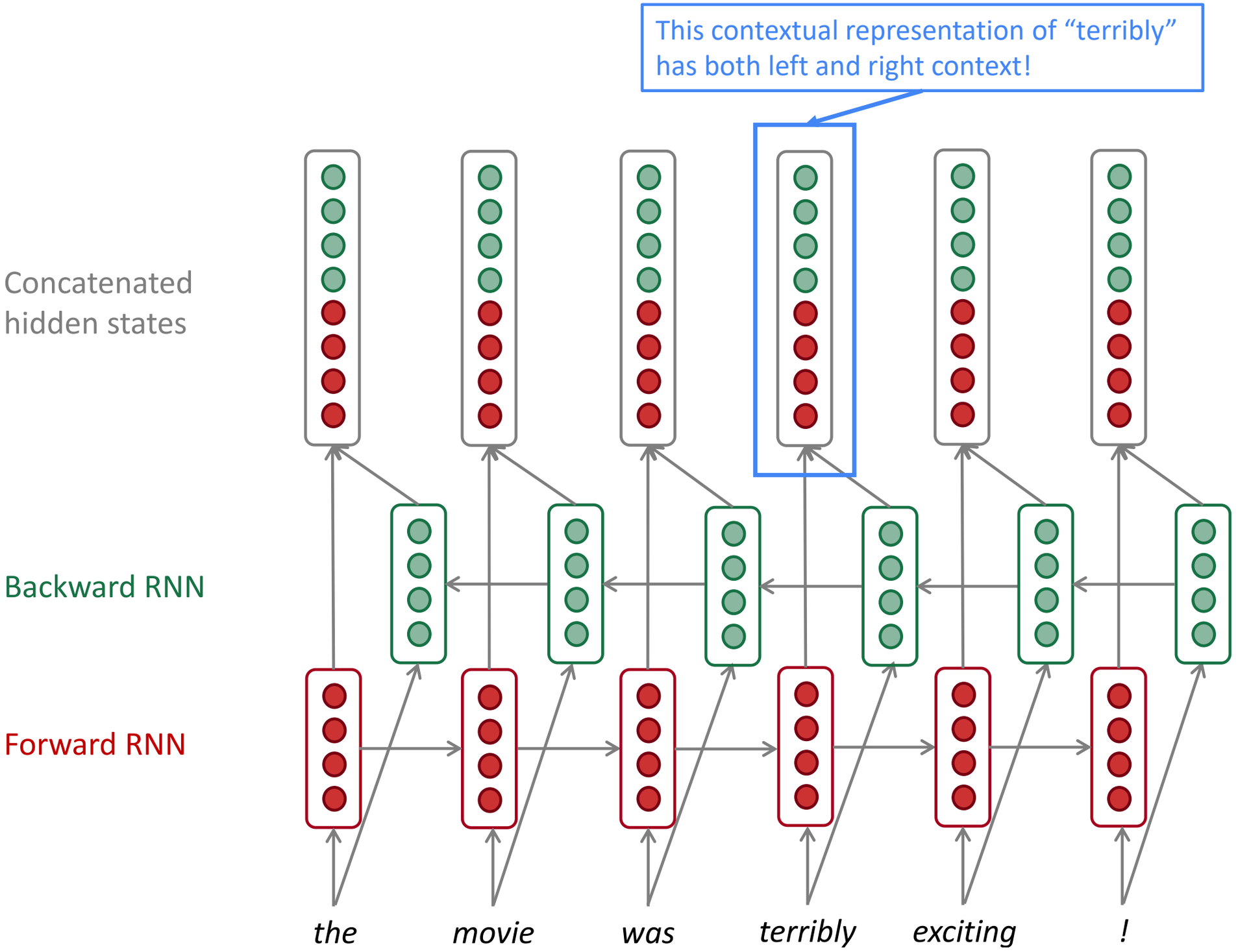
These contextual representations only contain information about the *left* context (e.g. *"the movie was"*).

What about *right* context?

In this example, *"exciting"* is in the right context and this modifies the meaning of *"terribly"* (from negative to positive)

Source: Stanford 224n

Bi-directional RNN



Source: Stanford 224n

Bi-directional RNN

Forward RNN

Backward RNN

Concatenated hidden states

$$\overrightarrow{\boldsymbol{h}}^{(t)} = \overrightarrow{\mathrm{RNN}}_{\mathrm{FW}}(\overrightarrow{\boldsymbol{h}}^{(t-1)}, \boldsymbol{x}^{(t)})$$

$$\overleftarrow{\boldsymbol{h}}^{(t)} = \overleftarrow{\mathrm{RNN}}_{\mathrm{BW}}(\overleftarrow{\boldsymbol{h}}^{(t+1)}, \boldsymbol{x}^{(t)})$$

$$\boldsymbol{h}^{(t)} = [\overrightarrow{\boldsymbol{h}}^{(t)}; \overleftarrow{\boldsymbol{h}}^{(t)}]$$

Generally, these two RNNs have separate weights

We regard this as “the hidden state” of a bidirectional RNN. This is what we pass on to the next parts of the network.

Source: Stanford 224n

Bi-directional RNN

- Bidirectional RNNs are only applicable when you have access to the entire input sequence. They are **not** applicable to Language Modeling as the future tokens are not accessible.
- If you do have entire input sequence, bidirectionality is powerful for encoding (you should use it by default).
- For example, BERT (**Bidirectional** Encoder Representations from Transformers) is a powerful pretrained contextual representation system built on bidirectionality (we'll learn more about BERT later)

Source: Stanford 224n

Stacked RNNs

NTU NLP

- RNNs are already "deep" in one dimension (time)
- We can also make them "deep" in another dimension by putting an RNN on top of another – this is a multi-layer RNN.
- This allows the network to compute more complex representations
  - The lower RNNs should compute lower-level features and the higher RNNs should compute higher-level features.
- Multi-layer RNNs are also called stacked RNNs.

Source: Stanford 224n

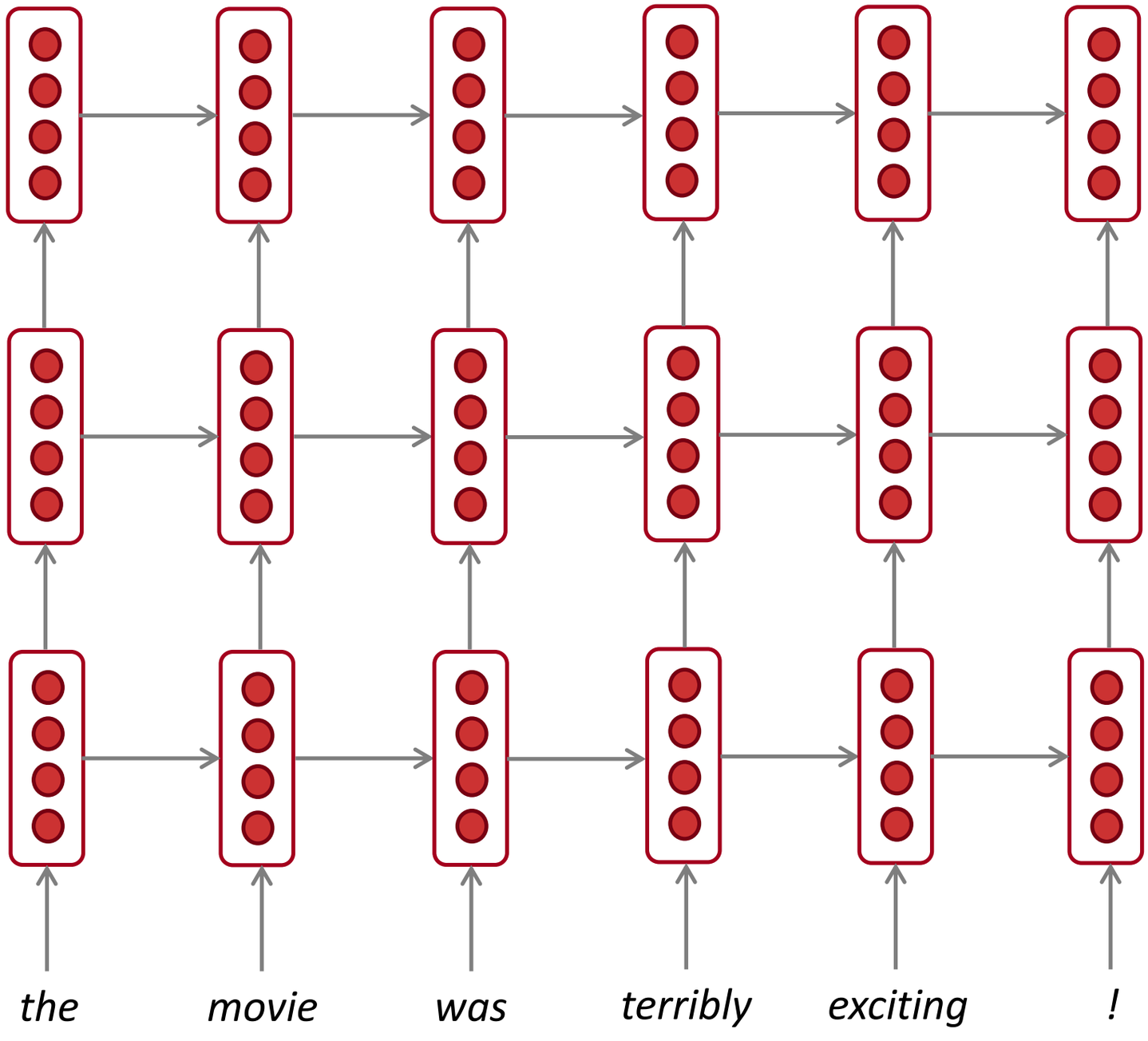
Stacked RNNs

The hidden states from RNN layer *i* are the inputs to RNN layer *i+1*

RNN layer 3

RNN layer 2

RNN layer 1



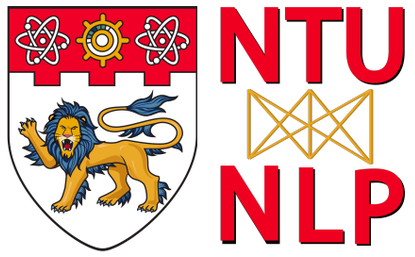
meaning depending on large context ...

semantics. discourse (pronounciation)

morphologic & syntatic info (e.g)

Source: Stanford 224n

Stacked RNNs

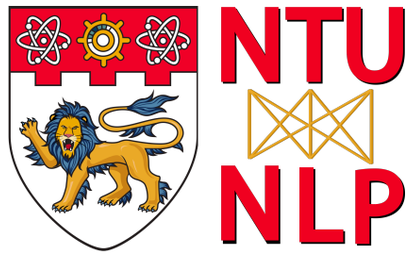


- High-performing RNNs are often multi-layer (but aren't as deep as convolutional or feed-forward networks) already deep in time dimension.
- For example: In a 2017 paper, Britz et al find that for Neural Machine Translation, 2 to 4 layers is best for the encoder RNN, and 4 layers is best for the decoder RNN
- However, skip-connections/dense-connections are needed to train deeper RNNs (e.g. 8 layers)
- Transformer-based networks (e.g. BERT) can be up to 24 layers

"Massive Exploration of Neural Machine Translation Architecutres", Britz et al, 2017. https://arxiv.org/pdf/1703.03906.pdf

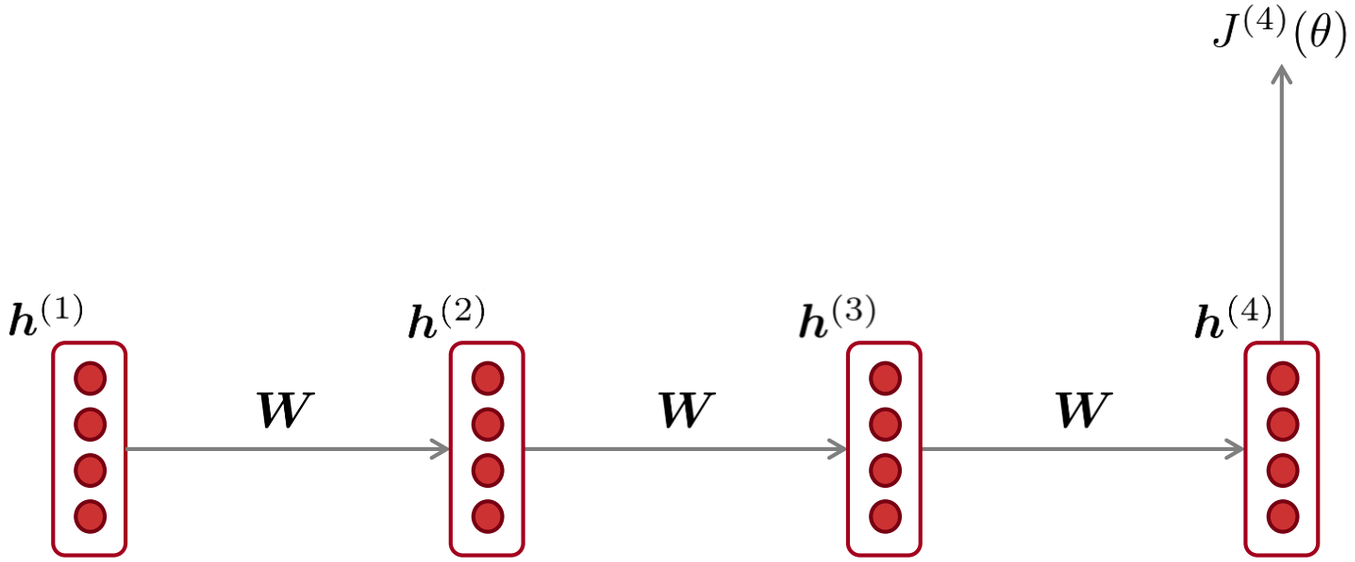
Source: Stanford 224n

Lecture Plan



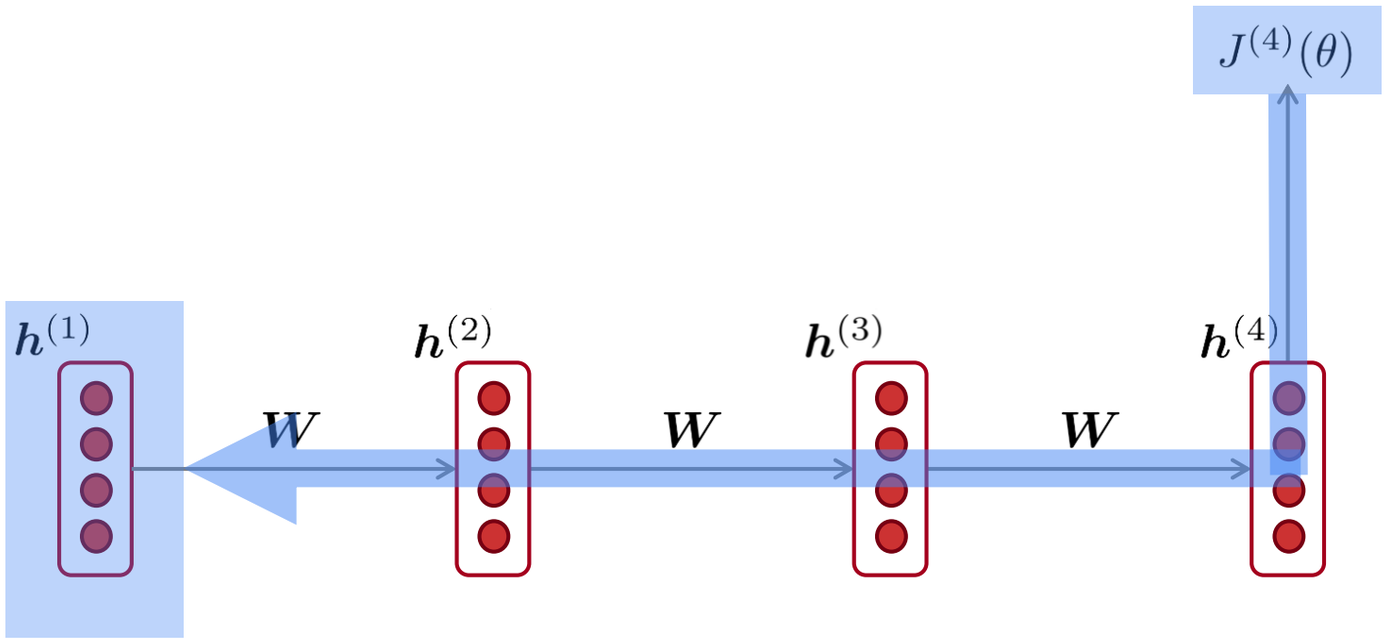
- Language Modeling
- Recurrent Neural Networks
- Backpropagation through time
- Text Generation with RNNs
- Sequence Tagging (NER, POS) with RNNs
- Sequence Classification with RNNs
- Bi-directional and stacked RNNs
- Gated RNNs (GRU, LSTM)

Vanishing Gradient Problem with Vanilla RNNs



Source: Stanford 224n

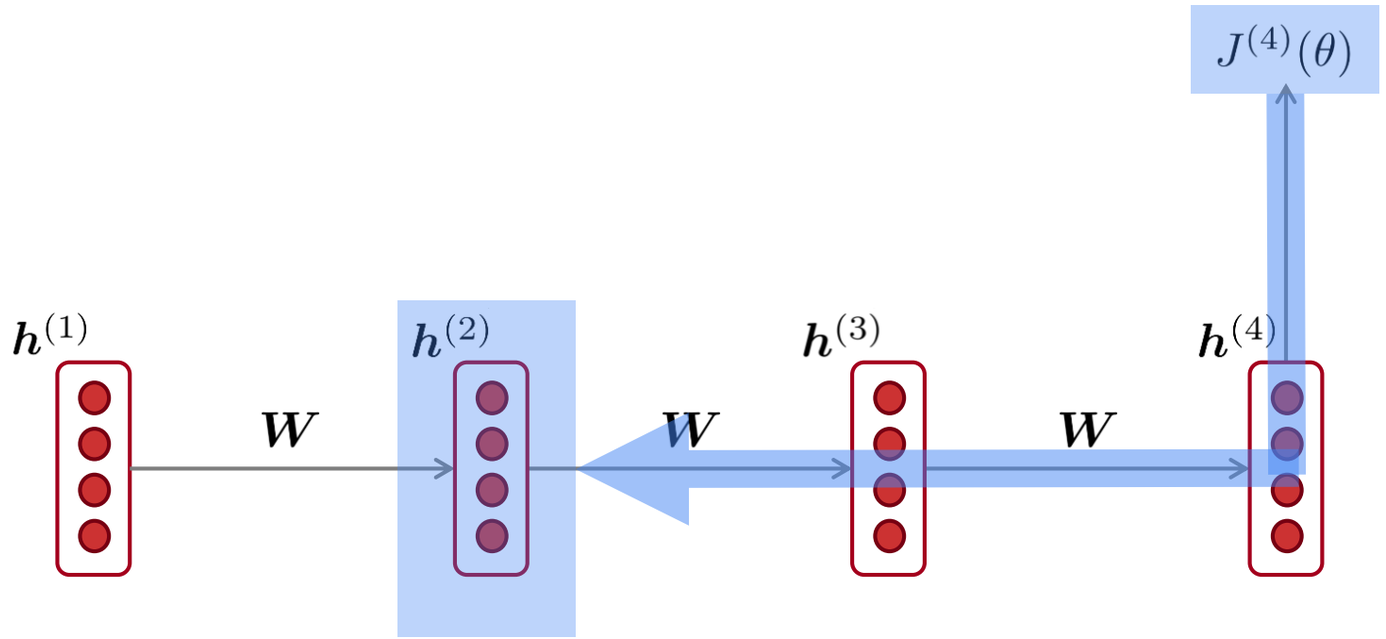
Vanishing Gradient Problem with Vanilla RNNs



$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \ ?$$

Source: Stanford 224n

Vanishing Gradient Problem with Vanilla RNNs

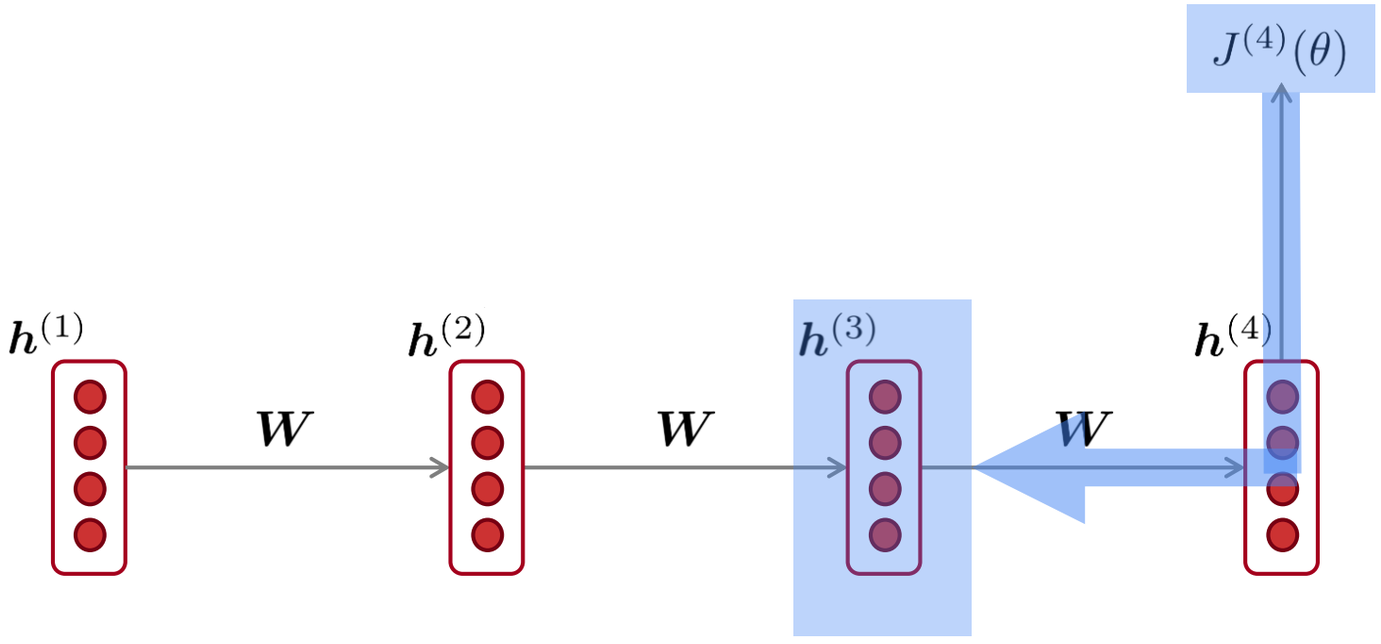


$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(2)}}$$

chain rule!

Source: Stanford 224n

Vanishing Gradient Problem with Vanilla RNNs



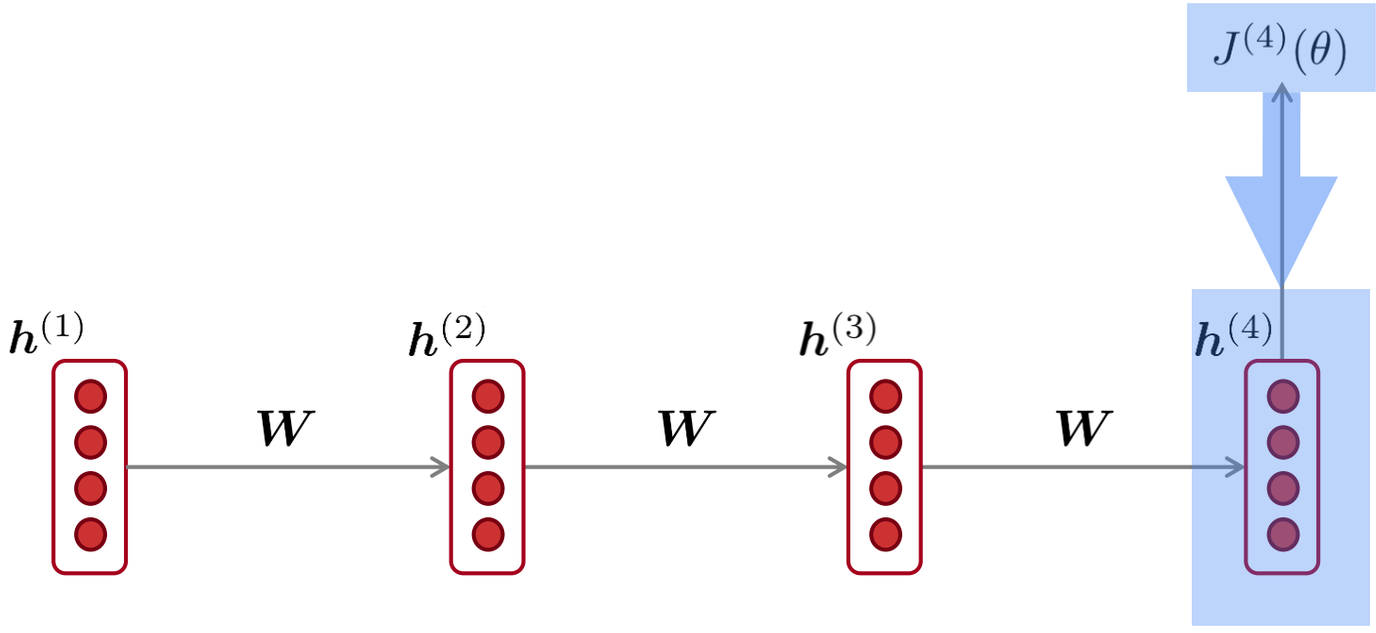
$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(3)}}$$

chain rule!

Source: Stanford 224n

Vanishing Gradient Problem with Vanilla RNNs

NTU NLP



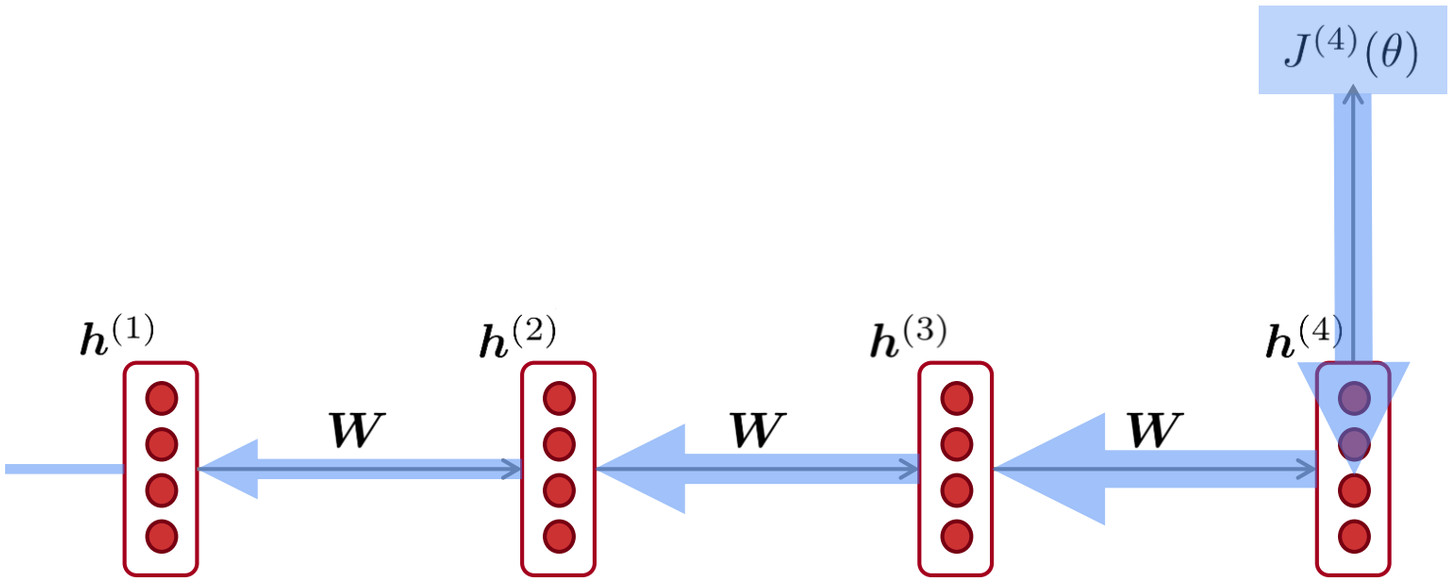
$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \times \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(4)}}$$

chain rule!

Source: Stanford 224n

Vanishing Gradient Problem with Vanilla RNNs

NTU NLP



$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \times \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(4)}}$$

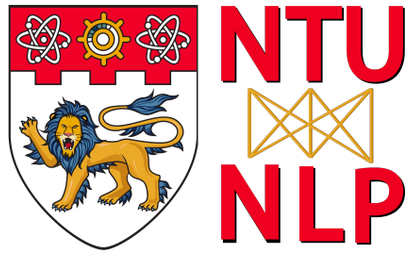
What happens if these are small?

Vanishing gradient problem:

When these are small, the gradient signal gets smaller and smaller as it backpropagates further

Source: Stanford 224n

Vanishing Gradient Problem with Vanilla RNNs



- A hidden layer in RNN: $$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x\boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)$$

- Taking the derivative: $$\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} = \operatorname{diag}\left(\sigma'\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x\boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right)\boldsymbol{W}_h$$

- The gradient of the loss on step i with respect to the hidden state on some previous step j:

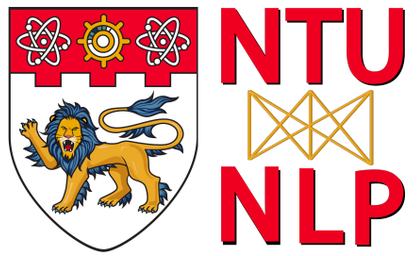
$$\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j<t\leq i} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} \qquad \text{(chain rule)}$$

$$= \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \boldsymbol{W}_h^{(i-j)} \prod_{j<t\leq i} \operatorname{diag}\left(\sigma'\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x\boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right) \qquad \text{(value of } \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}\text{)}$$

If W_h is small, then this term gets vanishingly small as *i* and *j* get further apart

Source: Stanford 224n

Vanishing Gradient Problem with Vanilla RNNs



- With the Matrix L2 norm:

$$\left\|\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}}\right\| \leq \left\|\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}}\right\| \|\boldsymbol{W}_h\|^{(i-j)} \prod_{j<t\leq i} \left\|\operatorname{diag}\left(\sigma'\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right)\right\|$$

- Pascanu et al showed that if the largest eigenvalue of W_h is less than 1 (bound is 1 for sigmoid nonlinearity), then the gradient $\left\|\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}}\right\|$ will shrink exponentially

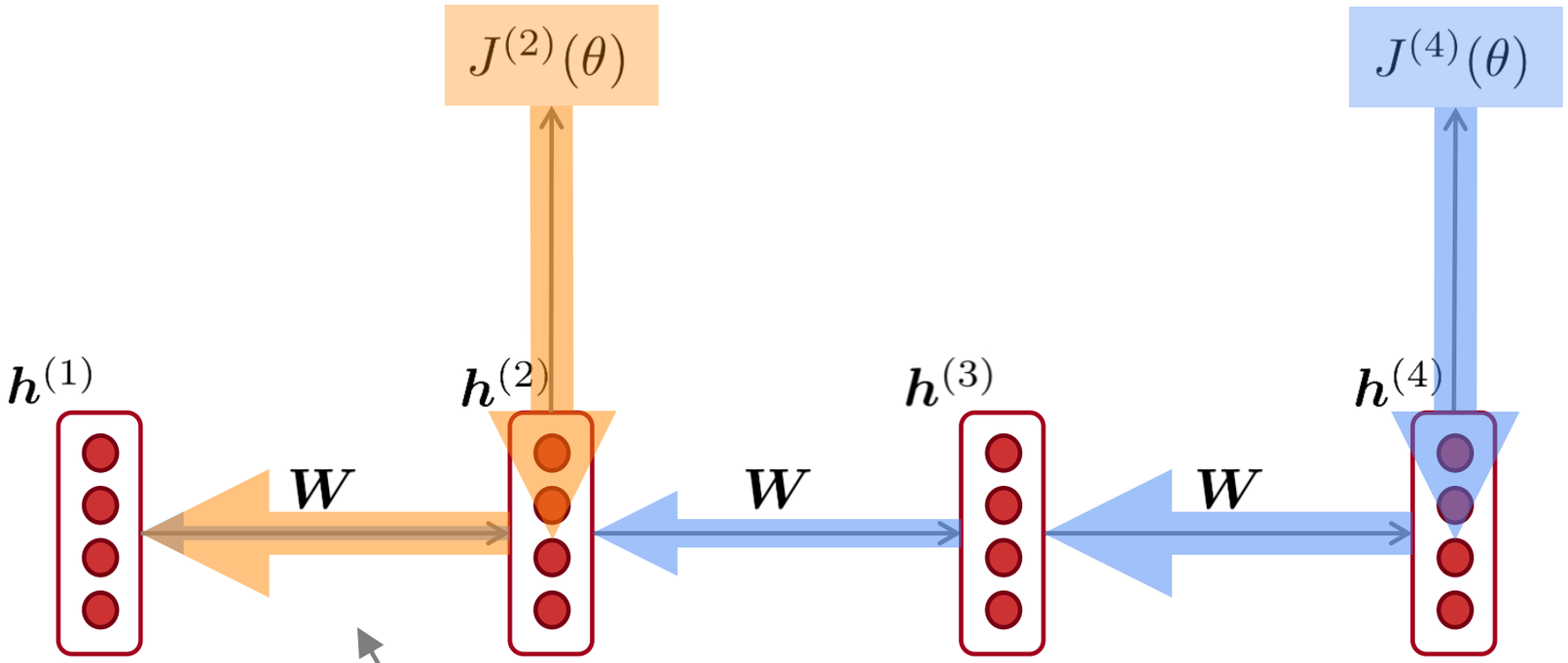
- Similarly a largest eigenvalue >1 results in exploding gradients

Pascanu et al, 2013. http://proceedings.mlr.press/v28/pascanu13.pdf

Source: Stanford 224n

Why Vanishing Gradient is a Problem

NTU NLP



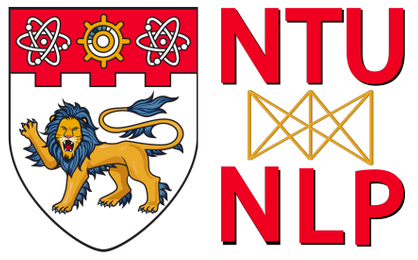
Gradient signal from faraway is lost because it's much smaller than gradient signal from close-by.

So model weights are only updated only with respect to near effects, not long-term effects.

not taking for weights into account. fails to capture the defined answers. (supervised)

Source: Stanford 224n

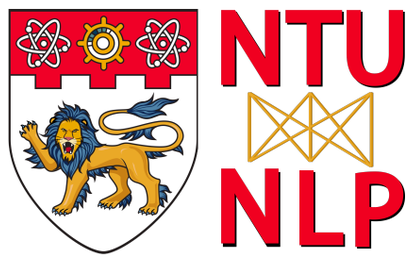
Why Vanishing Gradient is a Problem



- Gradient can be viewed as a measure of the effect of the past on the future.
- If the gradient becomes vanishingly small over longer distances (step t to step t+n), then we can't tell whether:
- There's no dependency between step t and t+n in the data or
- We have wrong parameters to capture the true dependency between t and t+n

Source: Stanford 224n

Why Vanishing Gradient is a Problem



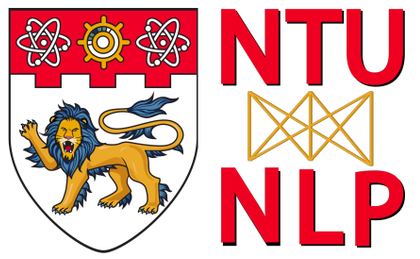
- **LM task:** When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her _____

- To learn from this training example, the RNN-LM needs to model the dependency between "tickets" on the 7th step and the target word "tickets" at the end.

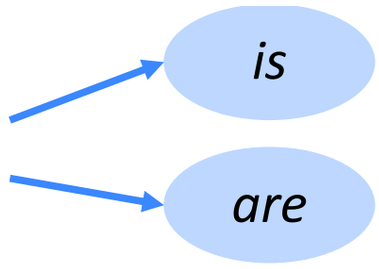
- But if gradient is small, the model can't learn this dependency

Source: Stanford 224n

Why Vanishing Gradient is a Problem



- **LM task:** *The writer of the books ___*



- **Correct answer**: *The writer of the books <u>is</u> planning a sequel*

- **Syntactic recency:** *The <u>writer</u> of the books <u>is</u>* (correct)

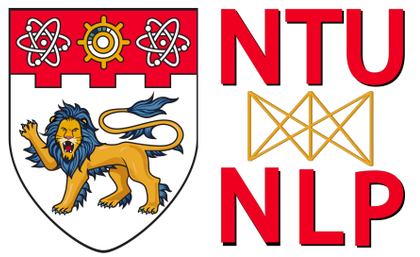
- **Sequential recency:** *The writer of the <u>books</u> <u>are</u>* (incorrect)

Due to vanishing gradient, RNN-LMs are better at learning from sequential recency than syntactic recency, so they make this type of error more often [Linzen et al 2016]

"Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies", Linzen et al, 2016. https://arxiv.org/pdf/1611.01368.pdf

Source: Stanford 224n

Why Exploding Gradient is a Problem



- If the gradient becomes too big, then the SGD update step becomes too big:

$$\theta^{new} = \theta^{old} - \underbrace{\overbrace{\alpha}^{\text{learning rate}} \nabla_\theta J(\theta)}_{\text{gradient}}$$

- This can cause bad updates: we take too large a step and reach a bad parameter configuration (with large loss)

- In the worst case, this will result in Inf or NaN in your network (then you have to restart training from an earlier checkpoint)

Source: Stanford 224n

Solution for Exploding Gradient

NTU NLP

Gradient clipping: if the norm of the gradient is greater than some threshold, scale it down before applying SGD update

Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$

if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**

$\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$

end if

- Intuition: take a step in the same direction, but a smaller step

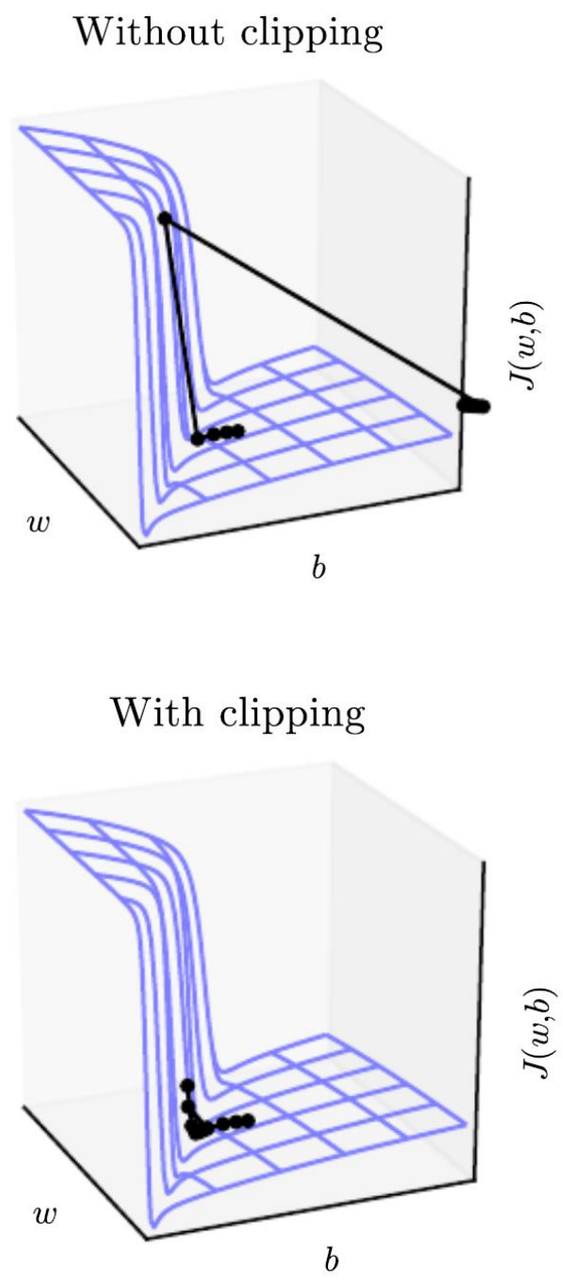
""On the difficulty of training recurrent neural networks", Pascanu et al, 2013. http://proceedings.mlr.press/v28/pascanu13.pdf

Source: Stanford 224n

Solution for Exploding Gradient

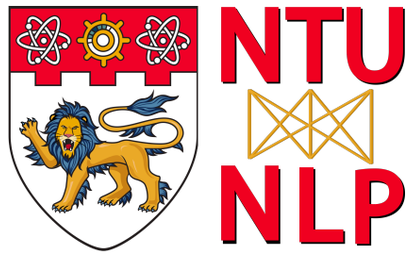
NTU NLP

- This shows the loss surface of a simple RNN (hidden state is a scalar not a vector)
- The "cliff" is dangerous because it has steep gradient
- In the Fig. at the top, gradient descent takes two very big steps due to steep gradient, resulting in climbing the cliff then shooting off to the right (both bad updates)
- At the bottom, gradient clipping reduces the size of those steps, so effect is less drastic



Source: Stanford 224n

Tackling Vanishing Gradient



- The main problem is that it's too difficult for the vanilla RNNs to learn to preserve information over many timesteps. The "same" hidden state is constantly updated

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}\right)$$

- How about a RNN with separate memory?

Source: Stanford 224n

Long Short-Term Memory (LSTM)

NTU NLP

- On step t, there is a hidden state $\boldsymbol{h}^{(t)}$ and a cell state $\boldsymbol{c}^{(t)}$
  - Both are vectors length n (neurons in h)
  - The cell stores long-term information
  - The LSTM can erase, write and read information from the cell

- The selection of which information is erased/written/read is controlled by three corresponding gates

- The gates are also vectors length n

- On each timestep, each element of the gates can be open (1), closed (0), or somewhere in-between.

- The gates are dynamic: their value is computed based on the current context (previous text and current input)

"Long short-term memory", Hochreiter and Schmidhuber, 1997. https://www.bioinf.jku.at/publications/older/2604.pdf

Source: Stanford 224n

Long Short-Term Memory (LSTM)

NTU NLP

- Given current input and previous hidden state, we compute the gates

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

Sigmoid function: all gate values are between 0 and 1

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

- Now we compute how much to forget, update and output

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

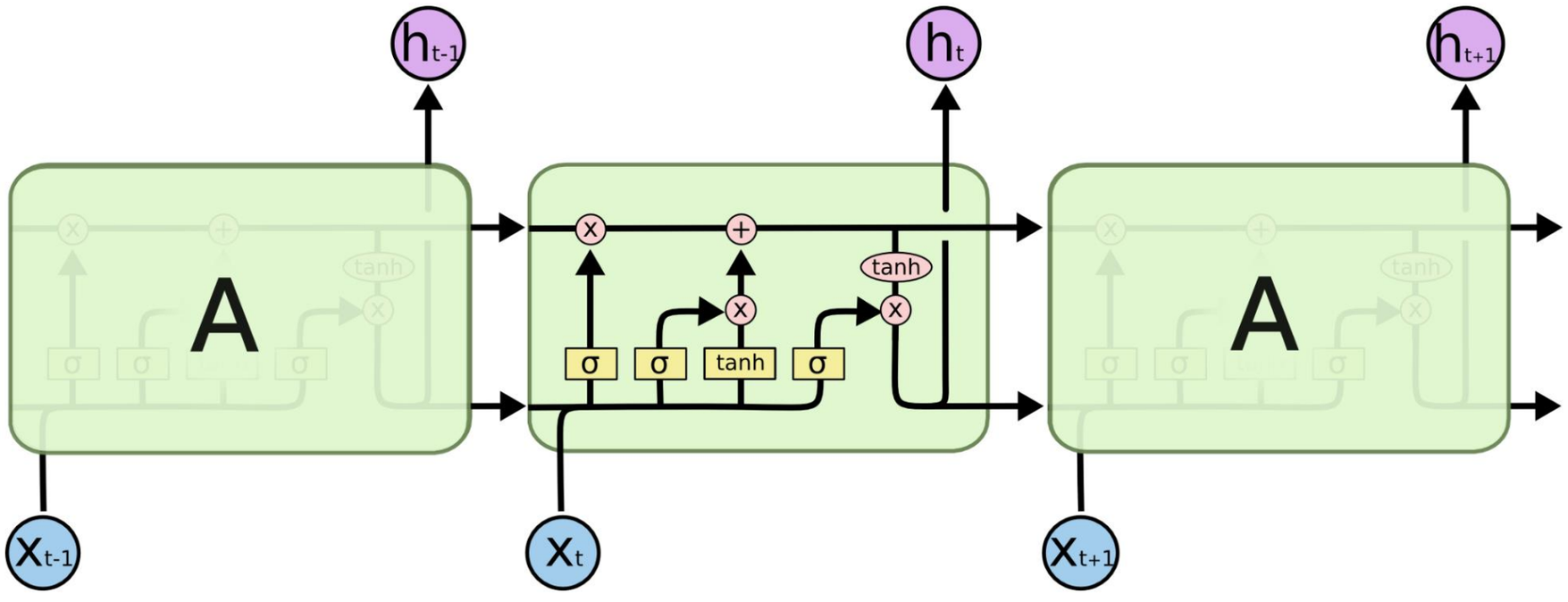
$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

Gates are applied using element-wise product

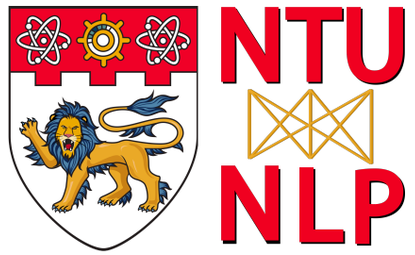
Source: Stanford 224n

Long Short-Term Memory (LSTM)

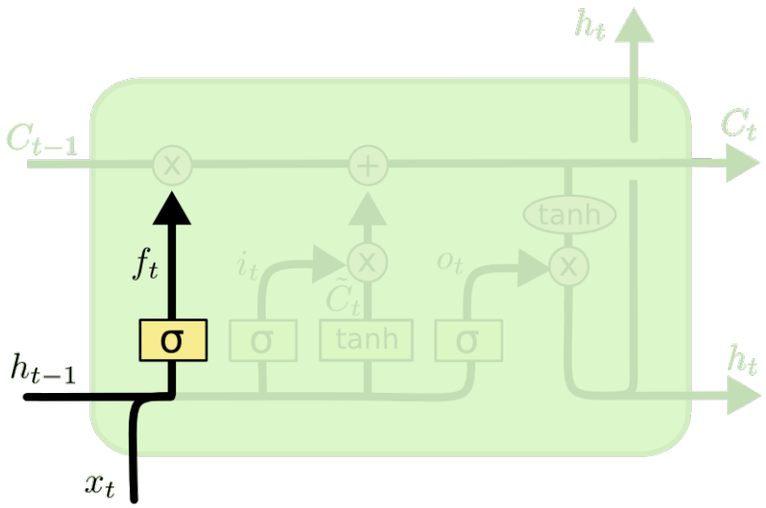


Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM)



- Compute forget gate

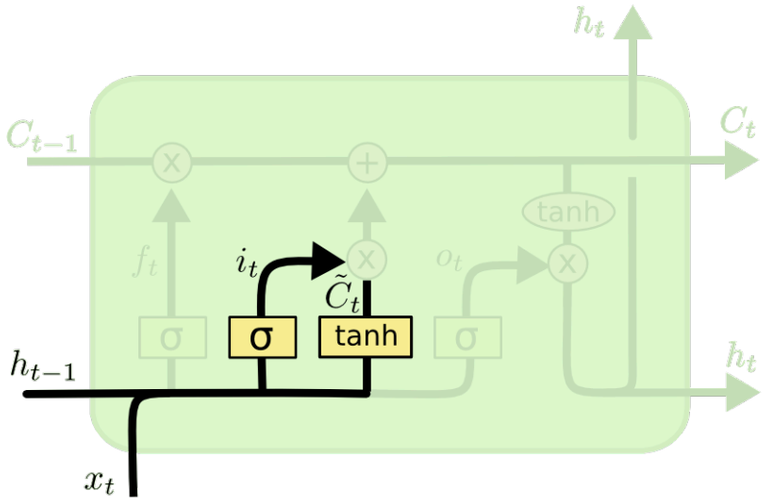


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM)

- Compute current cell state and input gate

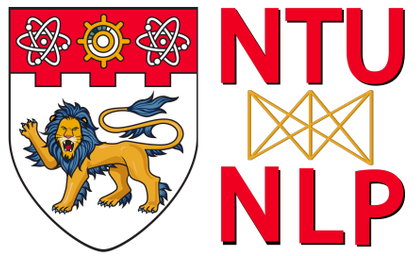


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

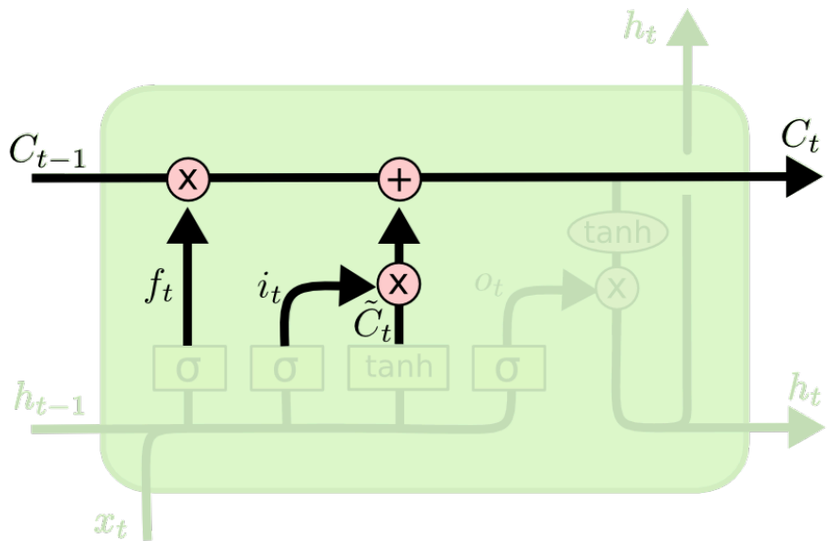
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM)



- Forget (erase) something and write (update) something new



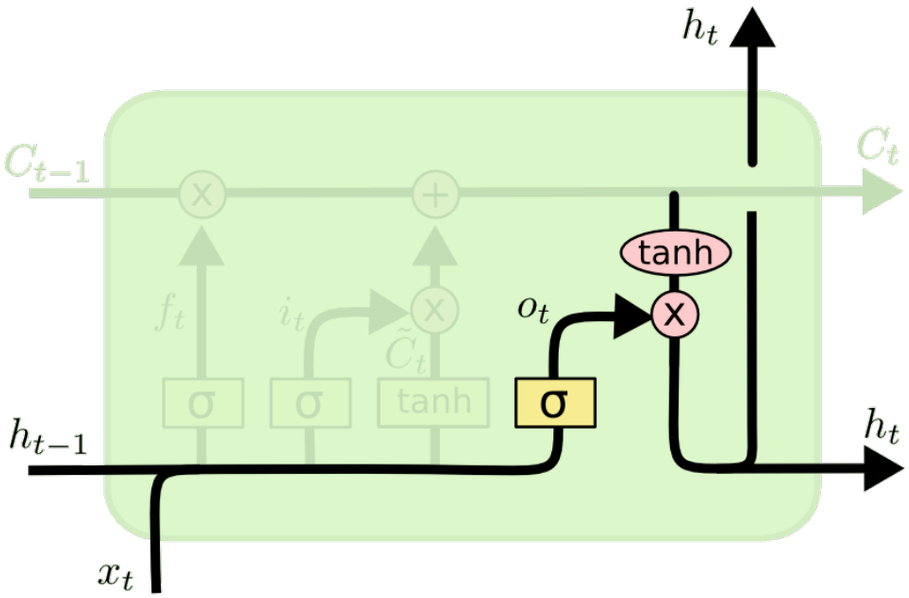
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM)

NTU NLP

- Compute output gate and output state



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

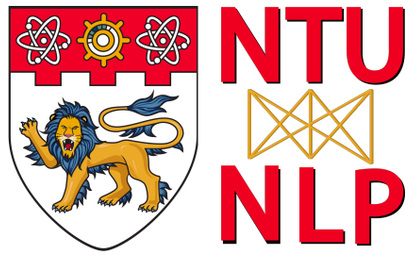
Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

How LSTM Remembers

- The LSTM architecture makes it easier for the RNN to preserve information over many time steps.
  - If the forget gate is set to remember everything on every time step, then the info in the cell is preserved indefinitely.
  - In practice, LSTM uses adaptive gates, which are also learned (i.e., using their respective parameters)
- LSTM however doesn't guarantee that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies.

Source: Stanford 224n

Success of LSTMs



- In 2013-2015, LSTMs started achieving state-of-the-art results
  - Successful tasks: handwriting recognition, speech recognition, machine translation, parsing, image captioning.
  - Became the dominant approach
- In recent years (2017-2020), other approaches (e.g. Transformers) have become more dominant for certain tasks. For example in **WMT** (a MT conference + competition):
  - WMT 2016, the summary report contains "RNN" 44 times
  - In WMT 2018, the report contains "RNN" 9 times and "Transformer" 63 times

Source: Stanford 224n

Gated Recurrent Unit (GRU)

A simpler alternative to LSTM

NTU NLP

- No cell state, remembers through two gates

Update gate: controls what parts of hidden state are updated vs preserved

$$\boldsymbol{u}^{(t)} = \sigma\left(\boldsymbol{W}_u \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_u \boldsymbol{x}^{(t)} + \boldsymbol{b}_u\right)$$

Reset gate: controls what parts of previous hidden state are used to compute new content

$$\boldsymbol{r}^{(t)} = \sigma\left(\boldsymbol{W}_r \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_r \boldsymbol{x}^{(t)} + \boldsymbol{b}_r\right)$$

New hidden state content: reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content.

$$\tilde{\boldsymbol{h}}^{(t)} = \tanh\left(\boldsymbol{W}_h(\boldsymbol{r}^{(t)} \circ \boldsymbol{h}^{(t-1)}) + \boldsymbol{U}_h \boldsymbol{x}^{(t)} + \boldsymbol{b}_h\right)$$

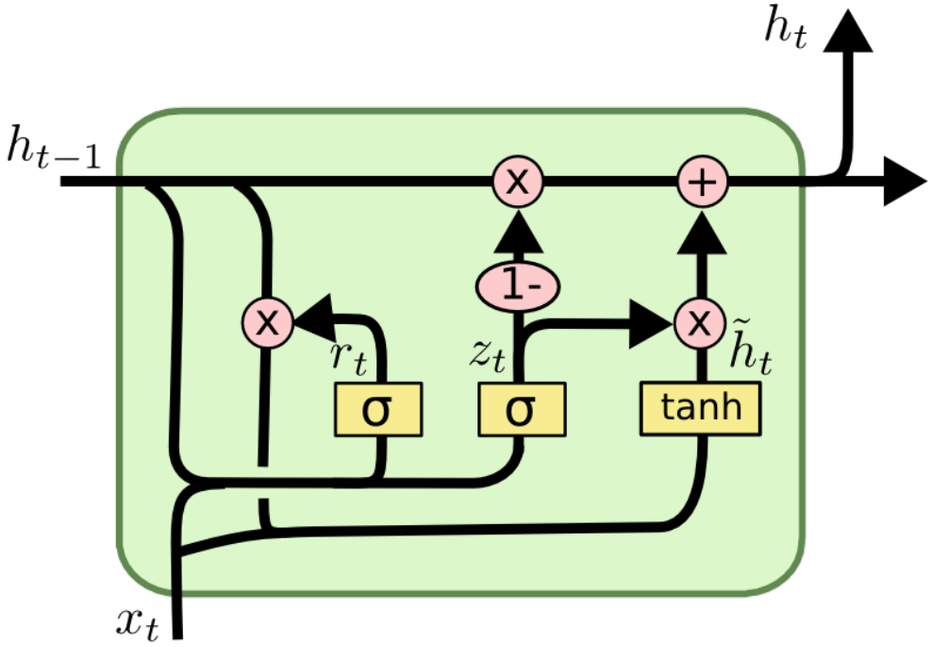
reset previous

Hidden state: update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

$$\boldsymbol{h}^{(t)} = (1 - \boldsymbol{u}^{(t)}) \circ \boldsymbol{h}^{(t-1)} + \boldsymbol{u}^{(t)} \circ \tilde{\boldsymbol{h}}^{(t)}$$

Source: Stanford 224n

Gated Recurrent Unit (GRU)



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

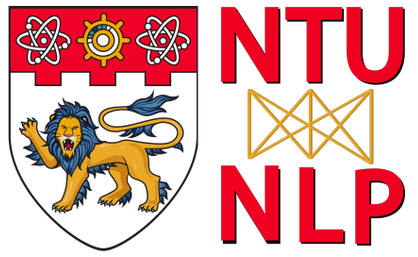
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

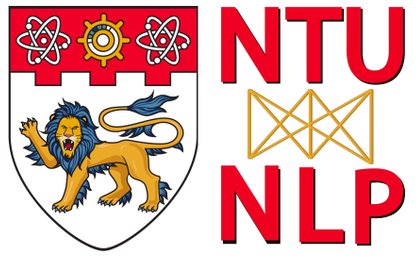
LSTM vs. GRU



- GRU is quicker to compute and has fewer parameters
- No conclusive evidence that one consistently performs better than the other
- LSTM is a good default choice (especially if your data has particularly long dependencies, or you have lots of training data)
- Rule of thumb: start with LSTM, but switch to GRU if you want something more efficient

Source: Stanford 224n

Vanishing/Exploding Gradient in Other Architectures

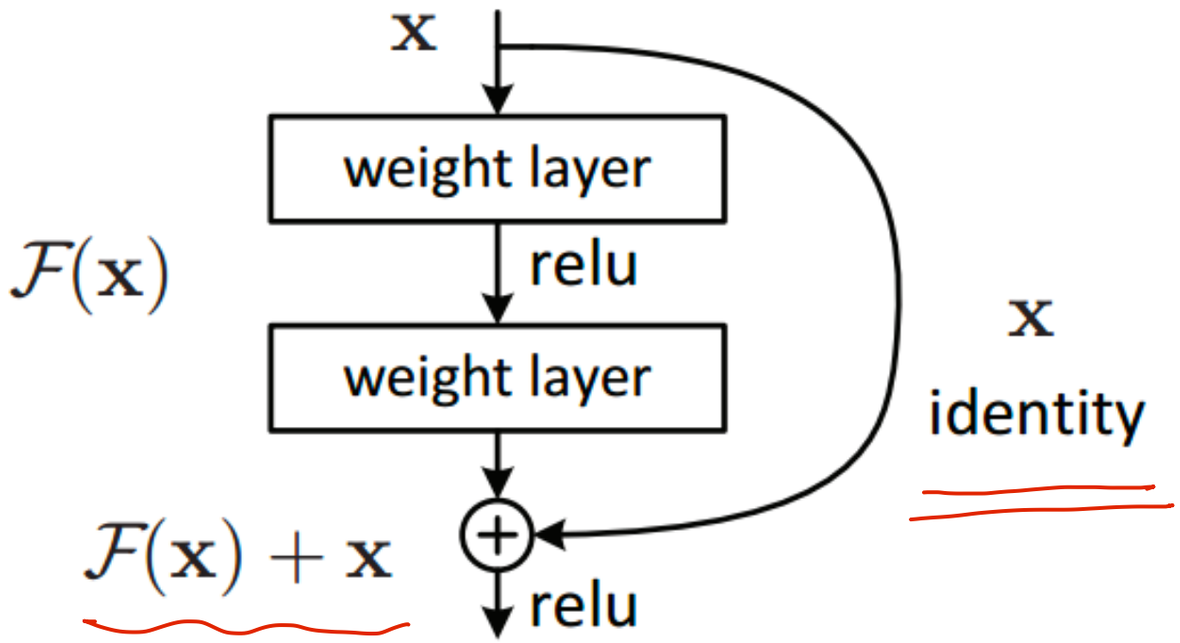


- Vanishing/Exploding gradient problems are not specific to only RNNs
- Can happen in all neural deep architectures (including feed-forward and convolutional; RNNs are deep in time).
  - Due to chain rule and/or choice of nonlinearity function, gradient can become vanishingly small as it backpropagates.
  - Thus lower layers are learnt very slowly (hard to train)
- Solution: lots of new deep feedforward/convolutional architectures add more direct connections (thus allowing the gradient to flow)

Source: Stanford 224n

Direct Connections: Residual Links

- Solution: lots of new deep feedforward/convolutional architectures add more direct connections (thus allowing the gradient to flow)
  - Residual connections aka "ResNet"
  - Also known as skip-connections
  - The identity connection preserves information by default
  - This makes deep networks much easier to train



$H(x) = F(x) + x$

$H(x) - x = F(x)$

before going to final Non-Linear layer

Why?

https://arxiv.org/pdf/1512.03385.pdf

Source: Stanford 224n

Direct Connections: Highway Connections

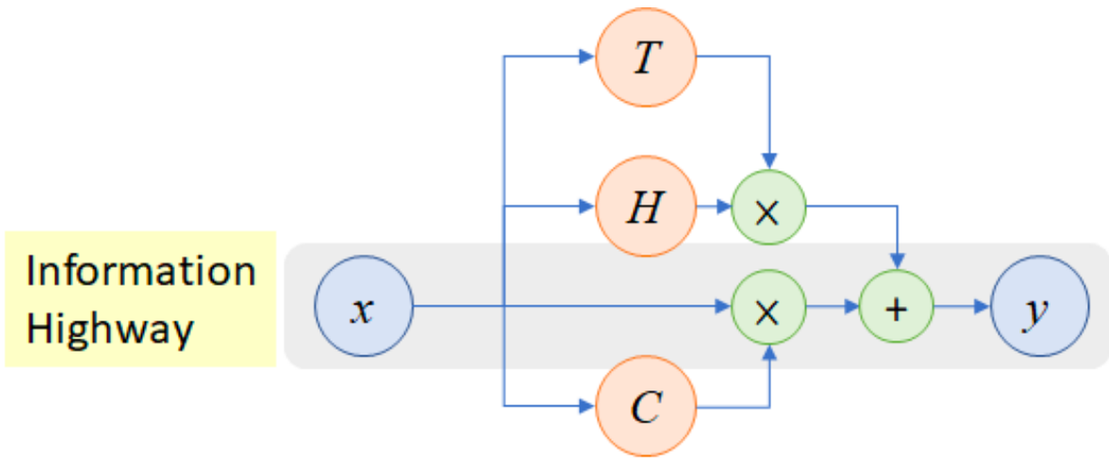
- Solution: lots of new deep feedforward/convolutional architectures add more direct connections (thus allowing the gradient to flow)

  - Similar to residual connections, but the identity connection vs the regular (transformation) layer is controlled by a dynamic gate

- Inspired by LSTM's gating but applied to other architectures.

- Two non-linear transforms **T** and **C** are introduced: T is the Transform Gate and C is the Carry Gate; C = 1 - T

- H(x, W_H) is any hidden layer with x as input and W_H as weights



$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot C(\mathbf{x}, \mathbf{W_C}).$$

$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot (1 - T(\mathbf{x}, \mathbf{W_T})).$$

https://arxiv.org/pdf/1505.00387.pdf

Source: Stanford 224n

Final Remark

- Though vanishing/exploding gradients are a general problem, RNNs are particularly unstable due to the repeated multiplication by the same weight matrix [Bengio et al, 1994]

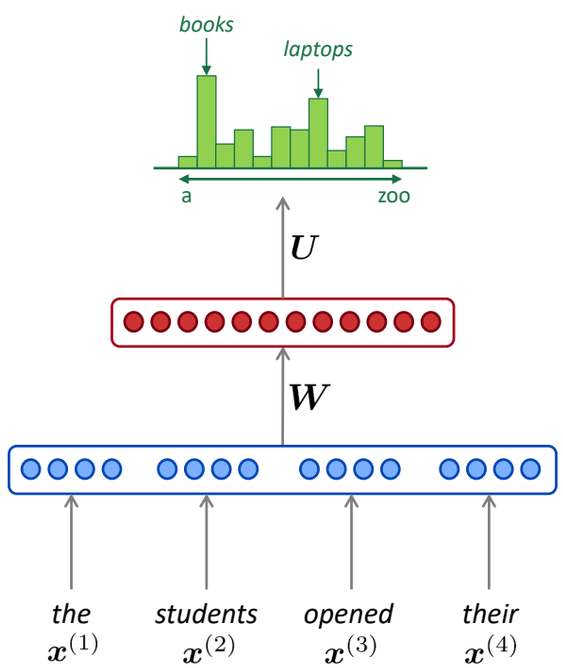
"Learning Long-Term Dependencies with Gradient Descent is Difficult", Bengio et al. 1994, http://ai.dinfo.unifi.it/paolo//ps/tnn-94-gradient.pdf

Source: Stanford 224n

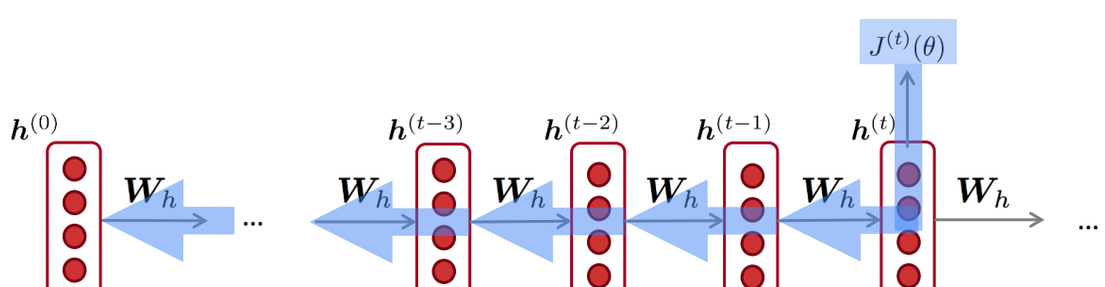
Summary

NTU NLP

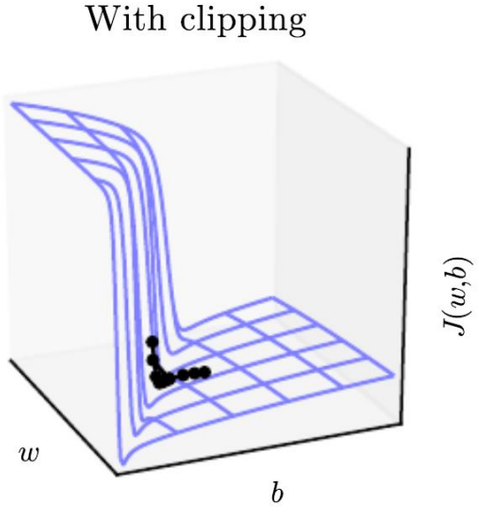
RNN-LM



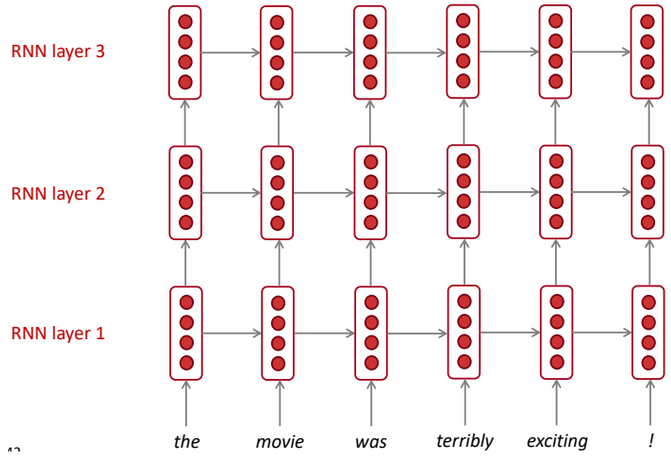
BPTT



Gradient clipping for exploding gradient



Multi-layer RNNs



LSTM/GRU