CSCE 689: Special Topics in Trustworthy NLP

Lecture 4: Natural Language Processing Basics (3)

Kuan-Hao Huang
khhuang@tamu.edu

(Some slides adapted from Chris Manning, Dan Jurafsky, Danqi Chen, and Vivian Chen)

Presentation Sign-Up

- We have 10 students
  - Each student present two papers

W13	11/11	Robustness of Multimodal Models	[Instructor] Learning Transferable Visual Models From Natural Language Supervision, ICML 2021 [Instructor] BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation, ICML 2022 [Instructor] Visual Instruction Tuning, NeurIPS 2023	Summary Due
	11/13	Robustness of Multimodal Models (Remote)	[Instructor] When and why vision-language models behave like bags-of-words, and what to do about it?, ICLR 2023 [Instructor] Text encoders bottleneck compositionality in contrastive vision-language models, EMNLP 2023 [Instructor] Paxion: Patching Action Knowledge in Video-Language Foundation Models, NeurIPS 2023	
	11/15	Robustness of Multimodal Models	[Student] Robust CLIP: Unsupervised Adversarial Fine-Tuning of Vision Embeddings for Robust Large Vision-Language Models, ICML 2024 [Student] On the Robustness of Large Multimodal Models Against Image Adversarial Attacks, CVPR 2024	
W14	11/18	Robustness of Multimodal Models	[Student] CleanCLIP: Mitigating Data Poisoning Attacks in Multimodal Contrastive Learning, ICCV 2023 [Student] Eyes Wide Shut? Exploring the Visual Shortcomings of Multimodal LLMs, CVPR 2024	

Presentation Sign-Up

- Sign-up: https://tinyurl.com/34e27fjx
  - Deadline: Friday 8/30 before lecture
- We will decide the assignment during lecture on 8/30

Week	Date	Topic	Paper ID	Paper Title
4	9/13	Adversarial Attacks and Defenses	1	Adversarial Example Generation with Syntactically Controlled Paraphrase Networks, NAACL 2018
4	9/13	Adversarial Attacks and Defenses	2	Jailbreaking Black Box Large Language Models in Twenty Queries, arXiv 2023
5	9/20	Backdoor Attacks and Data Poisoning	3	Poison Attacks against Text Datasets with Conditional Adversarially Regularized Autoencoder, EMNLP-Findings 2020
5	9/20	Backdoor Attacks and Data Poisoning	4	RAP: Robustness-Aware Perturbations for Defending against Backdoor Attacks on NLP Models, EMNLP 2021
6	9/27	AI-Generated Text Detection	5	RADAR: Robust AI-Text Detection via Adversarial Learning, NeurIPS 2023
6	9/27	AI-Generated Text Detection	6	Paraphrasing Evades Detectors of AI-Generated Text, But Retrieval is An Effective Defense, NeurIPS 2023
7	10/4	Model Uncertainty	7	Semantic Uncertainty: Linguistic Invariances for Uncertainty Estimation in Natural Language Generation, ICLR 2023
7	10/4	Model Uncertainty	8	Decomposing Uncertainty for Large Language Models through Input Clarification Ensembling, ICML 2024
9	10/18	Model Explainability and Interpretability	9	Reframing Human-AI Collaboration for Generating Free-Text Explanations, NAACL 2022
9	10/18	Model Explainability and Interpretability	10	Self-Consistency Improves Chain of Thought Reasoning in Lang
10	10/25	Bias Detection and Mitigation	11	Null It Out: Guarding Protected Attributes by Iterative Nullspace
10	10/25	Bias Detection and Mitigation	12	From Pretraining Data to Language Models to Downstream Task
11	11/1	Human Preference Alignment	13	SimPO: Simple Preference Optimization with a Reference-Free
11	11/1	Human Preference Alignment	14	Self-Play Fine-Tuning Converts Weak Language Models to Stron
12	11/8	Hallucinations and Misinformation Con	15	SAC3: Reliable Hallucination Detection in Black-Box Language
12	11/8	Hallucinations and Misinformation Con	16	Characterizing Truthfulness in Large Language Model Generatio
13	11/15	Robustness of Multimodal Models	17	Robust CLIP: Unsupervised Adversarial Fine-Tuning of Vision E
13	11/15	Robustness of Multimodal Models	18	On the Robustness of Large Multimodal Models Against Image
14	11/18	Robustness of Multimodal Models	19	CleanCLIP: Mitigating Data Poisoning Attacks in Multimodal Co
14	11/18	Robustness of Multimodal Models	20	Eyes Wide Shut? Exploring the Visual Shortcomings of Multimo

Name	Preference 1	Preference 2	Preference 3	Preference 4	Preference 5	Preference 6	Preference 7	Preference 8
Agrawal, Saransh								
Baid, Rahul								
Bajaj, Divij								
Balasubramanian, Sriram								
Harden, Dylan								
Hu, Chan-Wei								
Lee, Jaehoon								
Liu, Junru								
Rajagopalan, Vicram								
Samudra, Arunim Chaitanya								
Please fill in paper IDs								

Lecture Plan

- Natural Language Processing Basics
- Tokenization
  - Byte-Pair Encoding
- Common NLP Models
  - Convolutional Neural Network (CNN)
  - Recurrent Neural Network (RNN)
  - Long Short-Term Memory (LSTM)

Recap: Tokenization

- Currently, we use word (and punctuation) as the basic unit to tokenize a text
  - I like this movie so much. → I + like + this + movie + so + much + .

What is the size of word embeddings (how many words)?

Recap: Unknown Token

- We create an unknown token for all the words that have never been seen or low frequency words
  - <UNK>
- <UNK> has its own embedding
  - I like this movie &*# so much → I + like + this + movie + <UNK> + so + much + .
  - I like this movie sooooo much. → I + like + this + movie + <UNK> + much + .
- We can reduce the size of vocabulary
- We can handle unseen words

Recap: Subword Tokenization

- We use subword (and punctuation) as the basic unit to tokenize a text
- Subword: parts of words
  - happy, happier, happiest: happ-, -y, -ier, -iest
  - drive, driving, driven: driv-, -e, -ing, -en
  - beautiful, trustful, grateful: -ful

Byte-Pair Encoding

- Byte-Pair Encoding (BPE) is a simple method to decide subword
  - Originally designed for compression
  - Use fewer subwords to cover more words
- Motivation: discover the most common pair of consecutive bytes of data
  - Start with a vocabulary containing only characters and a “end-of-word” symbol
  - Find the most common pair of adjacent characters “x” and “y”; add subword “xy” to the vocabulary
  - Replace instances of the character pair with the new subword; repeat until desired vocabulary size

Byte-Pair Encoding Example

- Start with a vocabulary containing only characters and a "end-of-word" symbol

End-of-word symbol

l o w </w>	5 times
l o w e r </w>	2 times
n e w e s t </w>	6 times
w i d e s t </w>	3 times

Vocabulary

</w> l o w e
r n s t i d

Byte-Pair Encoding Example

- Find the most common pair of adjacent characters “x” and “y”; add subword “xy” to the vocabulary

7 times

l o w </w>	5 times
l o w e r </w>	2 times
n e w e s t </w>	6 times
w i d e s t </w>	3 times

9 times 9 times

Vocabulary

</w> l o w e
r n s t i d
es

Byte-Pair Encoding Example

- Replace instances of the character pair with the new subword

l o w </w>	5 times
l o w e r </w>	2 times
n e w es t </w>	6 times
w i d es t </w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es

Byte-Pair Encoding Example

- Find the most common pair of adjacent characters “x” and “y”; add subword “xy” to the vocabulary

l o w </w>	5 times
l o w e r </w>	2 times
n e w es t </w>	6 times
w i d es t </w>	3 times

9 times

Vocabulary

</w> l o w e
r n s t i d
es est

Byte-Pair Encoding Example

- Replace instances of the character pair with the new subword

l o w </w>	5 times
l o w e r </w>	2 times
n e w est </w>	6 times
w i d est </w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es est

Byte-Pair Encoding Example

- Find the most common pair of adjacent characters “x” and “y”; add subword “xy” to the vocabulary

l o w </w>	5 times
l o w e r </w>	2 times
n e w est </w>	6 times
w i d est </w>	3 times

9 times

Vocabulary

</w> l o w e
r n s t i d
es est est</w>

Byte-Pair Encoding Example

- Replace instances of the character pair with the new subword

l o w </w>	5 times
l o w e r </w>	2 times
n e w est</w>	6 times
w i d est</w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es est est</w>

Byte-Pair Encoding Example

- Find the most common pair of adjacent characters “x” and “y”; add subword “xy” to the vocabulary

7 times

l o w </w>	5 times
l o w e r </w>	2 times
n e w est</w>	6 times
w i d est</w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es est est</w>
lo

Byte-Pair Encoding Example

- Replace instances of the character pair with the new subword

`lo w </w>`	5 times
`lo w e r </w>`	2 times
`n e w est</w>`	6 times
`w i d est</w>`	3 times

Vocabulary

```
</w> l o w e
r n s t i d
es est est</w>
lo
```

Byte-Pair Encoding Example

- Find the most common pair of adjacent characters “x” and “y”; add subword “xy” to the vocabulary

7 times

lo w </w>	5 times
lo w e r </w>	2 times
n e w est</w>	6 times
w i d est</w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es est est</w>
lo low

Byte-Pair Encoding Example

- Replace instances of the character pair with the new subword

low </w>	5 times
low e r </w>	2 times
n e w est</w>	6 times
w i d est</w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es est est</w>
lo low

Byte-Pair Encoding Example

- Find the most common pair of adjacent characters “x” and “y”; add subword “xy” to the vocabulary

	`low </w>`	5 times
	`low e r </w>`	2 times
6 times	`n e w est</w>`	6 times
	`w i d est</w>`	3 times

Vocabulary

`</w> l o w e`
`r n s t i d`
`es est est</w>`
`lo low ne`

Byte-Pair Encoding Example

- Replace instances of the character pair with the new subword

`low </w>`	5 times
`low e r </w>`	2 times
`ne w est</w>`	6 times
`w i d est</w>`	3 times

Vocabulary

`</w> l o w e`
`r n s t i d`
`es est est</w>`
`lo low ne`

Byte-Pair Encoding Example

- Find the most common pair of adjacent characters “x” and “y”; add subword “xy” to the vocabulary

	low </w>	5 times
	low e r </w>	2 times
6 times	ne w est</w>	6 times
	w i d est</w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es est est</w>
lo low ne new

Byte-Pair Encoding Example

- Replace instances of the character pair with the new subword

`low </w>`	5 times
`low e r </w>`	2 times
`new est</w>`	6 times
`w i d est</w>`	3 times

Vocabulary

```
</w> l o w e
r n s t i d
es est est</w>
lo low ne new
```

Byte-Pair Encoding Example

- Find the most common pair of adjacent characters "x" and "y"; add subword "xy" to the vocabulary

	low </w>	5 times
	low e r </w>	2 times
6 times	new est</w>	6 times
	w i d est</w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es est est</w>
lo low ne new
newest</w>

Byte-Pair Encoding Example

- Replace instances of the character pair with the new subword

low </w>	5 times
low e r </w>	2 times
newest</w>	6 times
w i d est</w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es est est</w>
lo low ne new
newest</w>

Byte-Pair Encoding Example

- Find the most common pair of adjacent characters “x” and “y”; add subword “xy” to the vocabulary

				Vocabulary
5 times	`low </w>`	5 times		`</w> l o w e`
	`low e r </w>`	2 times		`r n s t i d`
	`newest</w>`	6 times		`es est est</w>`
	`w i d est</w>`	3 times		`lo low ne new`
				`newest</w>`
				`low</w>`

Byte-Pair Encoding Example

- Replace instances of the character pair with the new subword

low</w>	5 times
low e r </w>	2 times
newest</w>	6 times
w i d est</w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es est est</w>
lo low ne new
newest</w>
low</w>

Byte-Pair Encoding Example

MERGES

```
e + s => es
es + t => est
est + </w> => est</w>
l + o => lo
lo + w => low
n + e => ne
ne + w => new
new + est</w> => newest</w>
low + </w> => low</w>
```

Vocabulary

```
</w> l o w e
r n s t i d
es est est</w>
lo low ne new
newest</w>
low</w>
```

Byte-Pair Encoding Example

MERGES

```
e + s => es
es + t => est
est + </w> => est</w>
l + o => lo
lo + w => low
n + e => ne
ne + w => new
new + est</w> => newest</w>
low + </w> => low</w>
```

Vocabulary

```
</w> l o w e
r n s t i d
es est est</w>
lo low ne new
newest</w>
low</w>
```

New unseen token: lowest → `low est</w>`

Byte-Pair Encoding Example

MERGES

```
e + s => es
es + t => est
est + </w> => est</w>
l + o => lo
lo + w => low
n + e => ne
ne + w => new
new + est</w> => newest</w>
low + </w> => low</w>
```

Vocabulary

```
</w> l o w e
r n s t i d
es est est</w>
lo low ne new
newest</w>
low</w>
```

New unseen token: powest → `<UNK> o w est</w>`

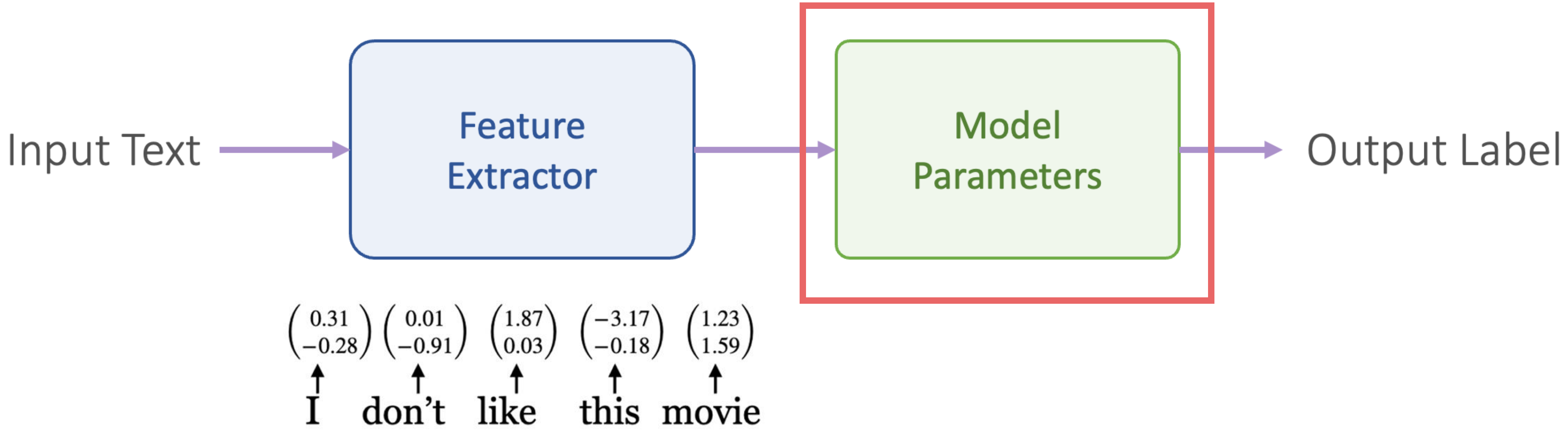
Subword Tokenization

- We use subword (and punctuation) as the basic unit to tokenize a text
- Subword: parts of words
  - happy, happier, happiest: happ-, -y, -ier, -iest
  - drive, driving, driven: driv-, -e, -ing, -en
  - beautiful, trustful, grateful: -ful
- A more effective way to construct vocabulary

Lecture Plan

- Natural Language Processing Basics
- Tokenization
  - Byte-Pair Encoding
- Common NLP Models
  - Convolutional Neural Network (CNN)
  - Recurrent Neural Network (RNN)
  - Long Short-Term Memory (LSTM)

Training NLP Models

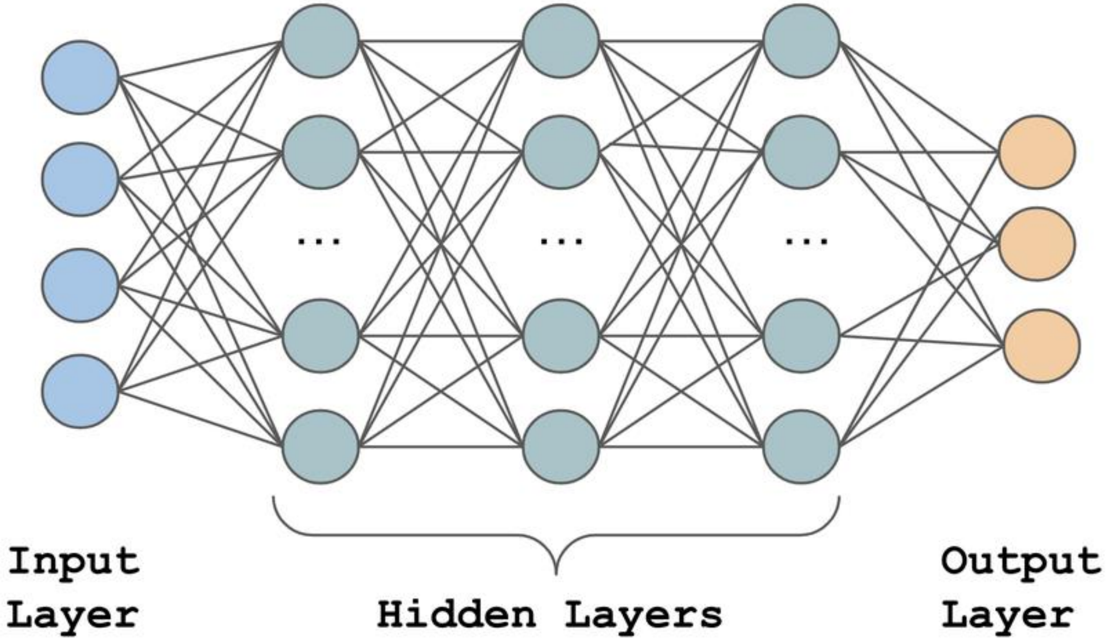


Input Lengths can be Different

$$\begin{array}{ccccc} \begin{pmatrix} 0.31 \\ -0.28 \end{pmatrix} & \begin{pmatrix} 0.01 \\ -0.91 \end{pmatrix} & \begin{pmatrix} 1.87 \\ 0.03 \end{pmatrix} & \begin{pmatrix} -3.17 \\ -0.18 \end{pmatrix} & \begin{pmatrix} 1.23 \\ 1.59 \end{pmatrix} \\ \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ \text{I} & \text{don't} & \text{like} & \text{this} & \text{movie} \end{array}$$

A Simple Approach: Averaged Embeddings + DNN

	Alice	treats	Bob	well	
Dimension 1	0.7	2.7	-0.1	-5.7	-0.6
Dimension 2	8.6	-3.9	6.7	-9.8	0.4
Dimension 3	-2.4	-5.6	1.5	-1.6	-1.6
Dimension 4	2.3	1.1	2.0	-1.0	1.1

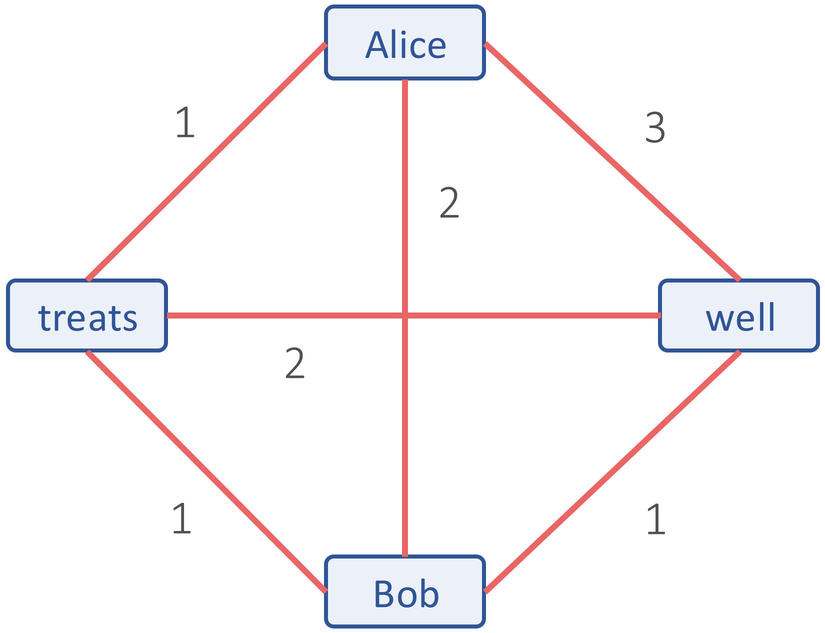


Disadvantages?

$$\text{Loss} = -\sum_{i=1}^{C} y_i \cdot \log(p_i)$$

Averaged Embeddings Lose Order Information

	Alice	treats	Bob	well	
	Bob	treats	Alice	well	
	treats	Bob	well	Alice	
Dimension 1	0.7	2.7	-0.1	-5.7	-0.6
Dimension 2	8.6	-3.9	6.7	-9.8	0.4
Dimension 3	-2.4	-5.6	1.5	-1.6	-1.6
Dimension 4	2.3	1.1	2.0	-1.0	1.1



Capture local information!
(neighbor information)

Convolutional Neural Network (CNN)

Learnable Weight (Filter)
Filter Size = 3

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ \cdots & \cdots & \cdots \\ w_{4,1} & w_{4,2} & w_{4,3} \end{bmatrix}$$

	Alice	treats	Bob	well
Dimension 1	0.7	2.7	-0.1	-5.7
Dimension 2	8.6	-3.9	6.7	-9.8
Dimension 3	-2.4	-5.6	1.5	-1.6
Dimension 4	2.3	1.1	2.0	-1.0

$w_{1,1}$*0.7 + $w_{1,2}$*2.7 + $w_{1,3}$*-0.1

$w_{2,1}$*8.6 + $w_{2,2}$*-3.9 + $w_{2,3}$*6.7

$w_{3,1}$*-2.4 + $w_{3,2}$*-5.6 + $w_{3,3}$*1.5

$w_{4,1}$*2.3 + $w_{4,2}$*1.1 + $w_{4,3}$*2.0

$v_{1,1}$ + $v_{1,2}$ + $v_{1,3}$ + $v_{1,4}$

v_1

Convolutional Neural Network (CNN)

Learnable Weight (Filter)
Filter Size = 3

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ \cdots & \cdots & \cdots \\ w_{4,1} & w_{4,2} & w_{4,3} \end{bmatrix}$$

	Alice	treats	Bob	well
Dimension 1	0.7	2.7	-0.1	-5.7
Dimension 2	8.6	-3.9	6.7	-9.8
Dimension 3	-2.4	-5.6	1.5	-1.6
Dimension 4	2.3	1.1	2.0	-1.0

$w_{1,1}$*2.7 + $w_{1,2}$*-0.1 + $w_{1,3}$*-5.7

$w_{2,1}$*-3.9 + $w_{2,2}$*6.7 + $w_{2,3}$*-9.8

$w_{3,1}$*-5.6 + $w_{3,2}$*1.5 + $w_{3,3}$*-1.6

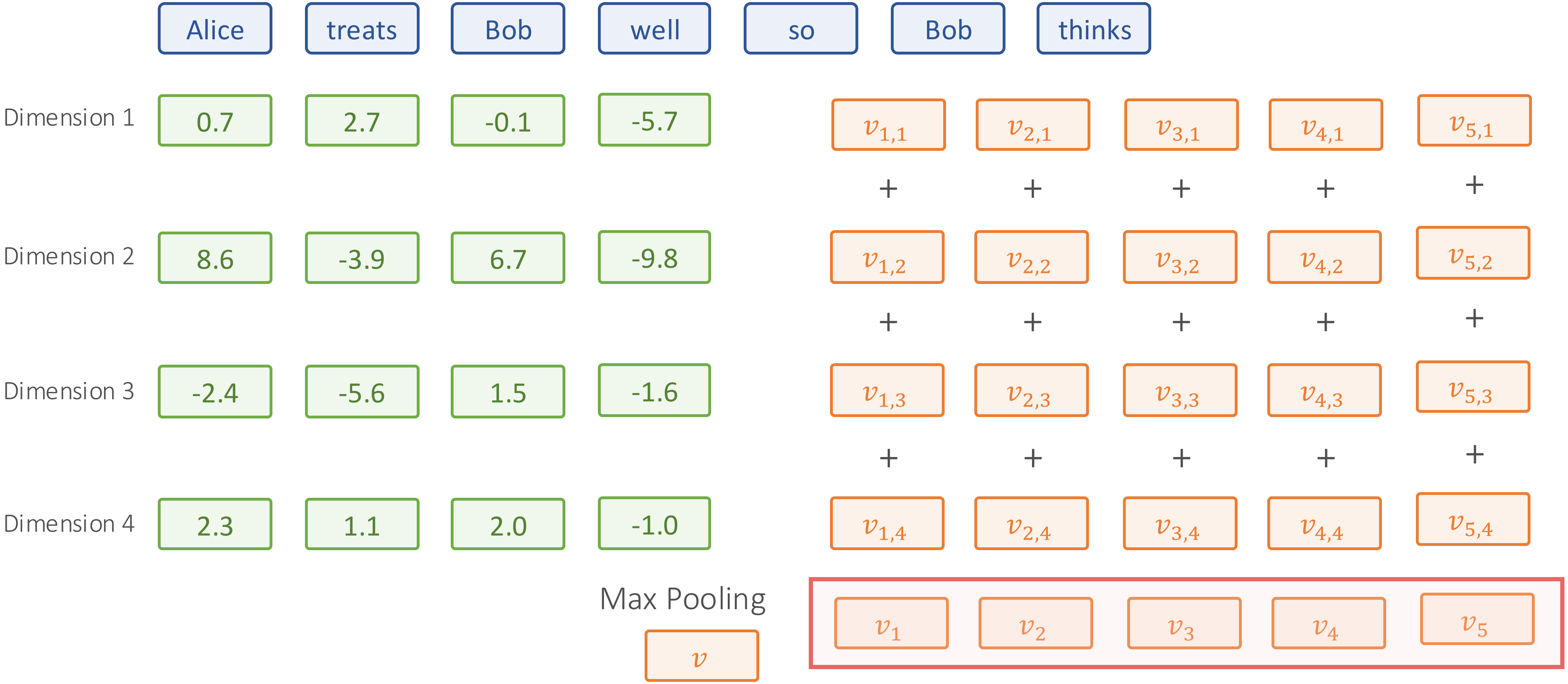
$w_{4,1}$*1.1 + $w_{4,2}$*2.0 + $w_{4,3}$*-1.0

$v_{1,1}$	$v_{2,1}$
+	+
$v_{1,2}$	$v_{2,2}$
+	+
$v_{1,3}$	$v_{2,3}$
+	+
$v_{1,4}$	$v_{2,4}$
v_1	v_2

Convolutional Neural Network (CNN)

Learnable Weight (Filter)
Filter Size = 3

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ \cdots & \cdots & \cdots \\ w_{4,1} & w_{4,2} & w_{4,3} \end{bmatrix}$$



Convolutional Neural Network (CNN)

Learnable Weight (Filter)
Filter Size = 3

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ \cdots & \cdots & \cdots \\ w_{4,1} & w_{4,2} & w_{4,3} \end{bmatrix} \quad \mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ \cdots & \cdots & \cdots \\ w_{4,1} & w_{4,2} & w_{4,3} \end{bmatrix} \quad \mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ \cdots & \cdots & \cdots \\ w_{4,1} & w_{4,2} & w_{4,3} \end{bmatrix}$$

	Alice	treats	Bob	well
Dimension 1	0.7	2.7	-0.1	-5.7
Dimension 2	8.6	-3.9	6.7	-9.8
Dimension 3	-2.4	-5.6	1.5	-1.6
Dimension 4	2.3	1.1	2.0	-1.0

v v v

Convolutional Neural Network (CNN)

Filter Size = 3 $\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ \cdots & \cdots & \cdots \\ w_{4,1} & w_{4,2} & w_{4,3} \end{bmatrix}$ Filter Size = 2 $\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} \\ \cdots & \cdots \\ w_{4,1} & w_{4,2} \end{bmatrix}$ Filter Size = 4 $\mathbf{W} = \begin{bmatrix} w_{1,1} & \cdots & w_{1,4} \\ \cdots & \cdots & \cdots \\ w_{4,1} & \cdots & w_{4,4} \end{bmatrix}$

	Alice	treats	Bob	well
Dimension 1	0.7	2.7	-0.1	-5.7
Dimension 2	8.6	-3.9	6.7	-9.8
Dimension 3	-2.4	-5.6	1.5	-1.6
Dimension 4	2.3	1.1	2.0	-1.0

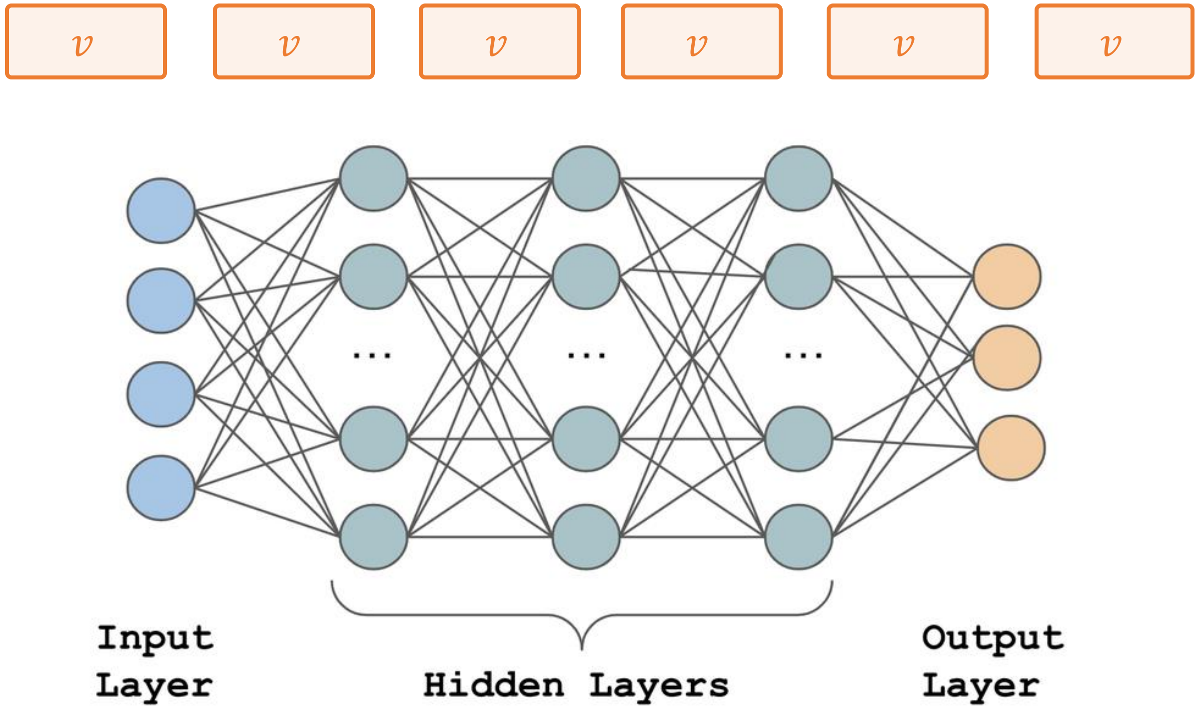
$w_{1,1}$*0.7 + $w_{1,2}$*2.7

$w_{2,1}$*8.6 + $w_{2,2}$*-3.9 + $w_{2,3}$*6.7 + $w_{2,4}$*-9.8

v v v

Convolutional Neural Network (CNN)

	Alice	treats	Bob	well
Dimension 1	0.7	2.7	-0.1	-5.7
Dimension 2	8.6	-3.9	6.7	-9.8
Dimension 3	-2.4	-5.6	1.5	-1.6
Dimension 4	2.3	1.1	2.0	-1.0



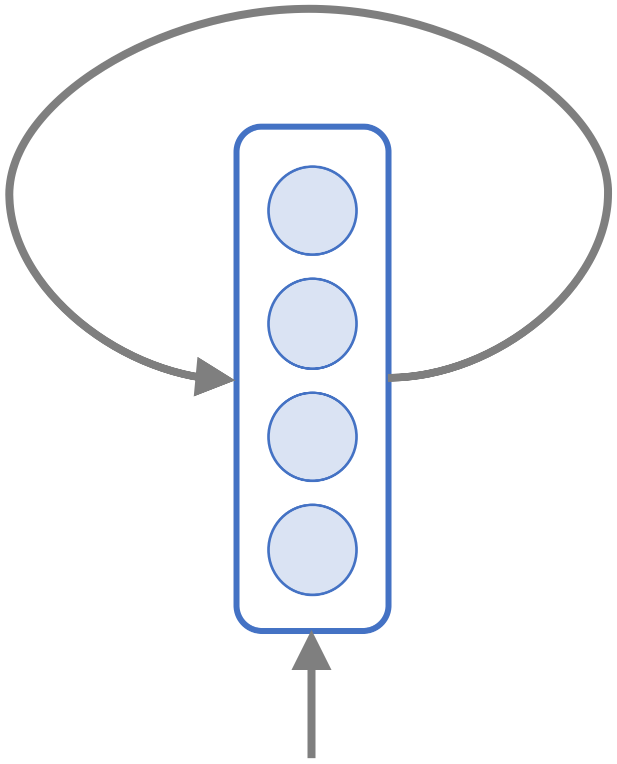
$$\text{Loss} = -\sum_{i=1}^{C} y_i \cdot \log(p_i)$$

Convolutional Neural Network (CNN)

- The whole process is still not similar to how human read texts
- Can we model reading texts in a sequential way?

Recurrent Neural Network (RNN)

- More idea: apply the same weights repeatedly at different positions



Recurrent Neural Network (RNN)

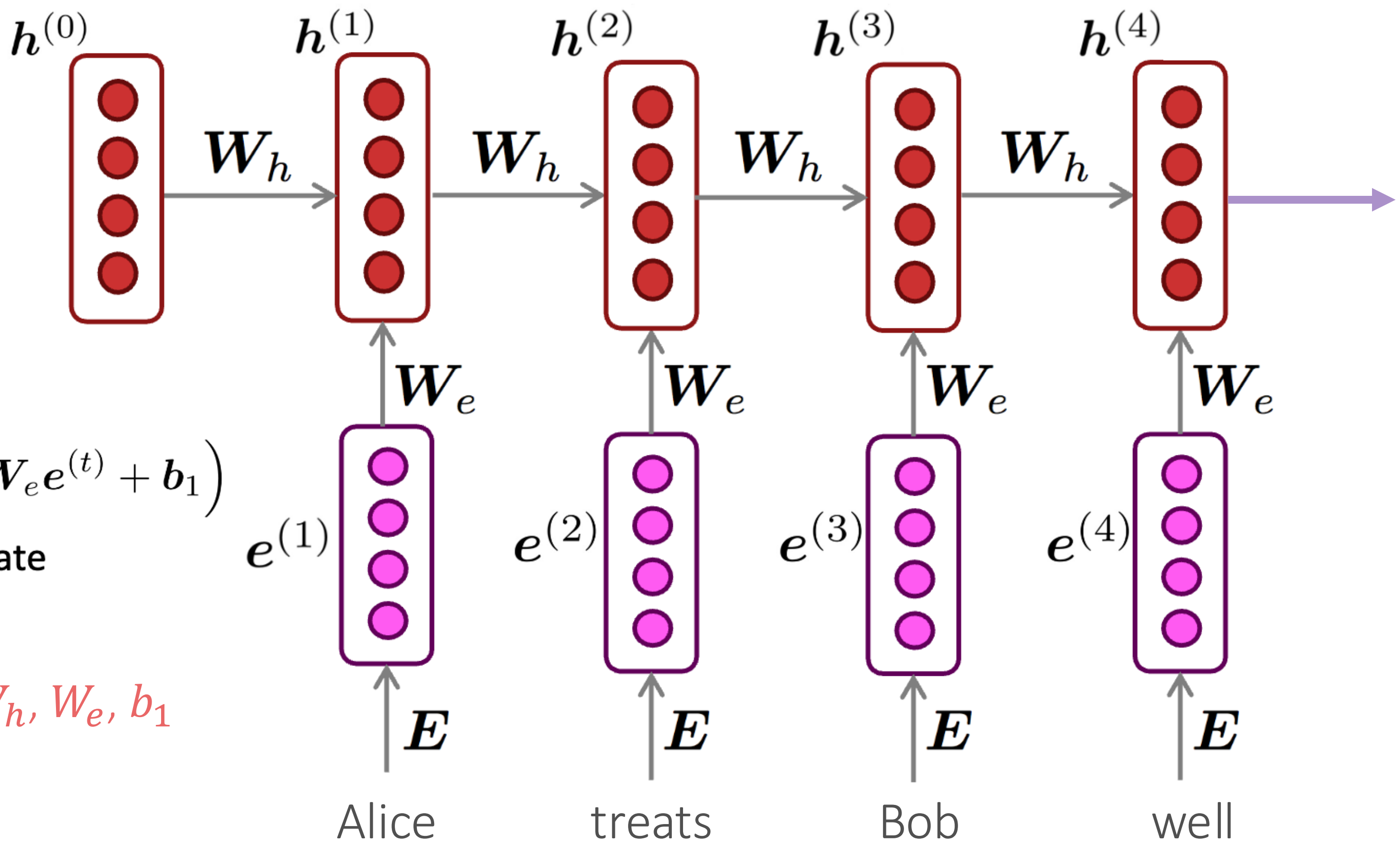
Activation Function
(tanh, sigmoid)

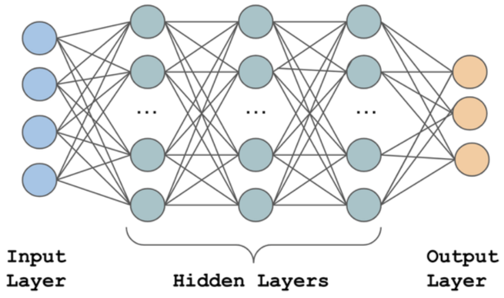
hidden states

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

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

Model Parameters W_h, W_e, b_1



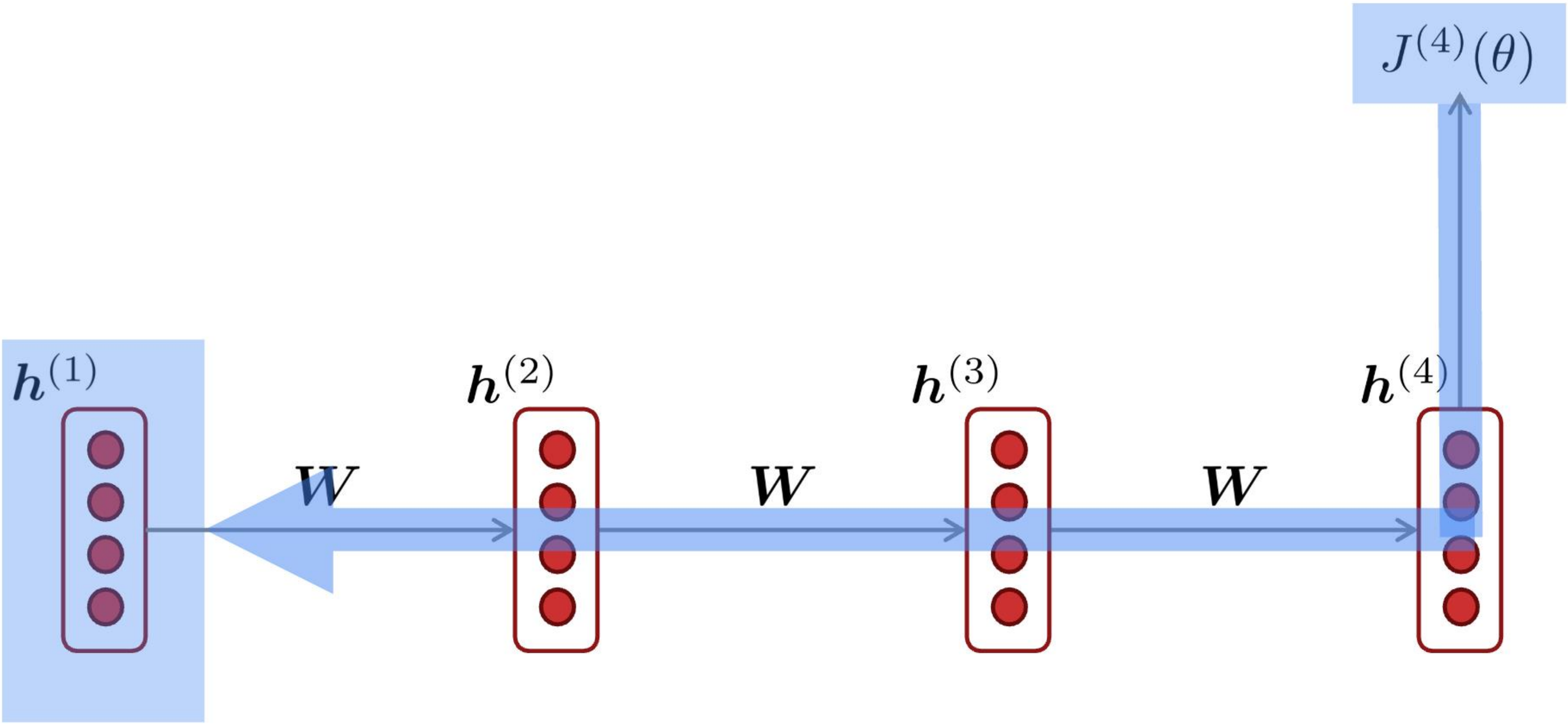


$\text{Loss} = -\sum_{i=1}^{C} y_i \cdot \log(p_i)$

Recurrent Neural Network (RNN)

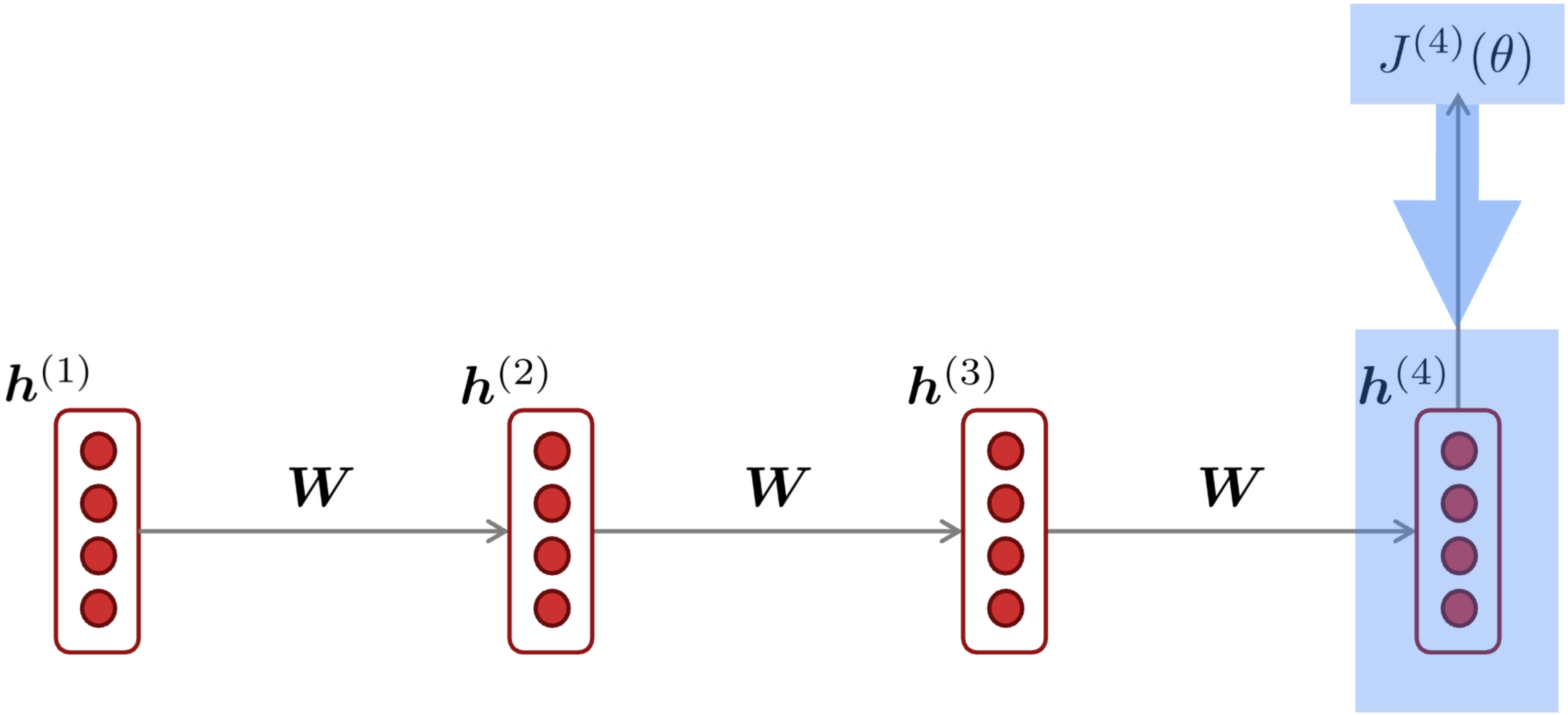
- Advantages
  - Can process any length input
  - Model size doesn't increase for longer input context
  - Computation for step t can (in theory) use information from many steps back
- Disadvantages
  - Recurrent computation is slow
  - In practice, difficult to access information from many steps back
  - Vanishing gradient

Vanishing Gradient



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

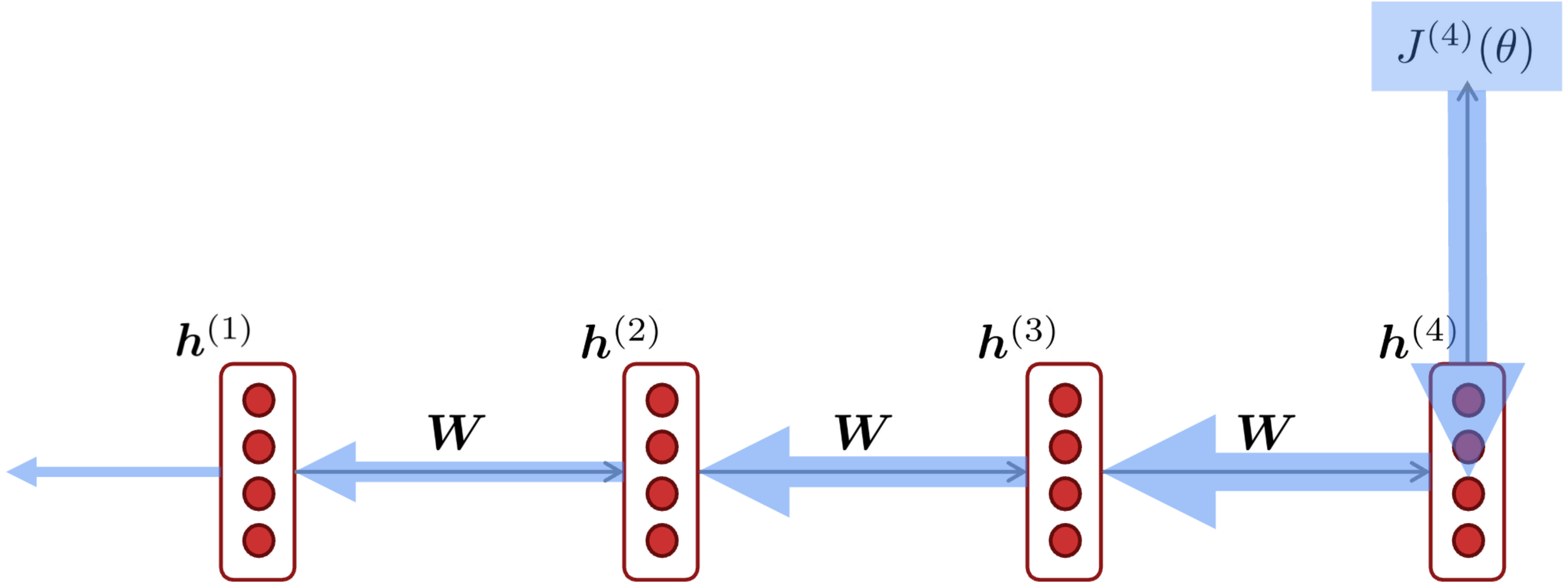
Vanishing Gradient



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

chain rule!

Vanishing Gradient



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

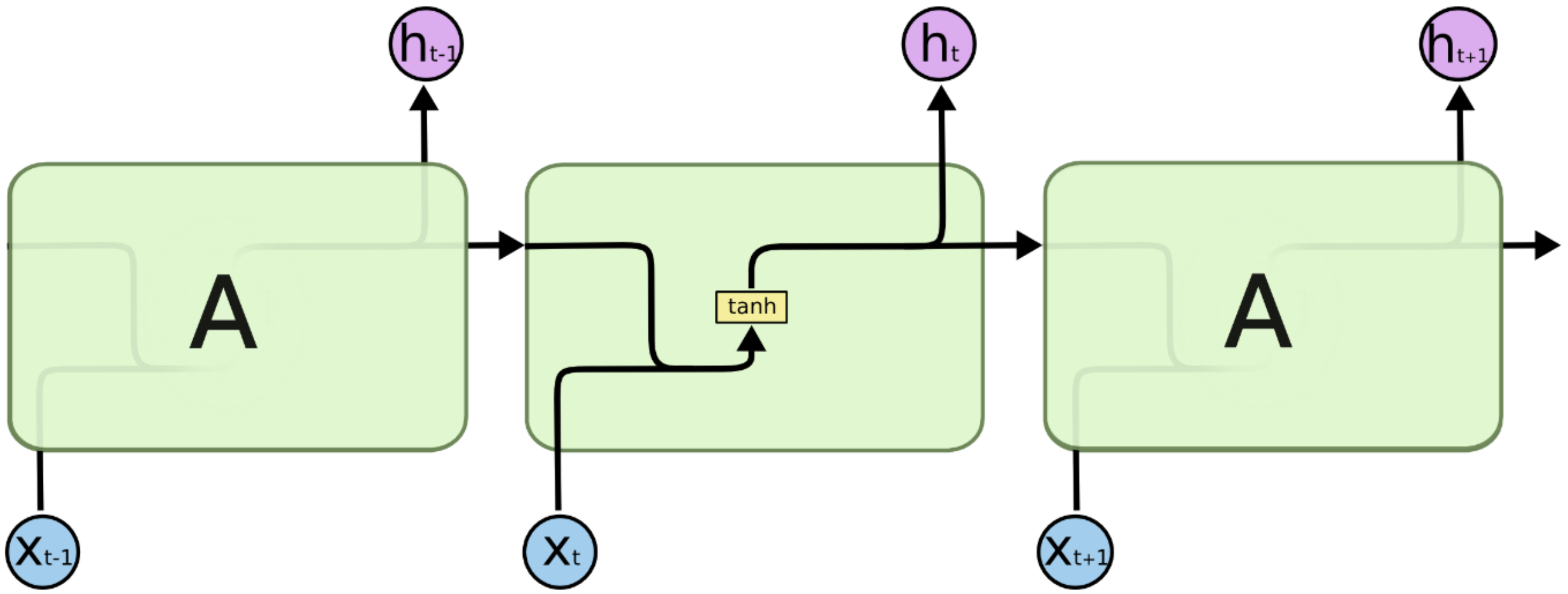
What happens if these are small?

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

Long Short-Term Memory (LSTM)

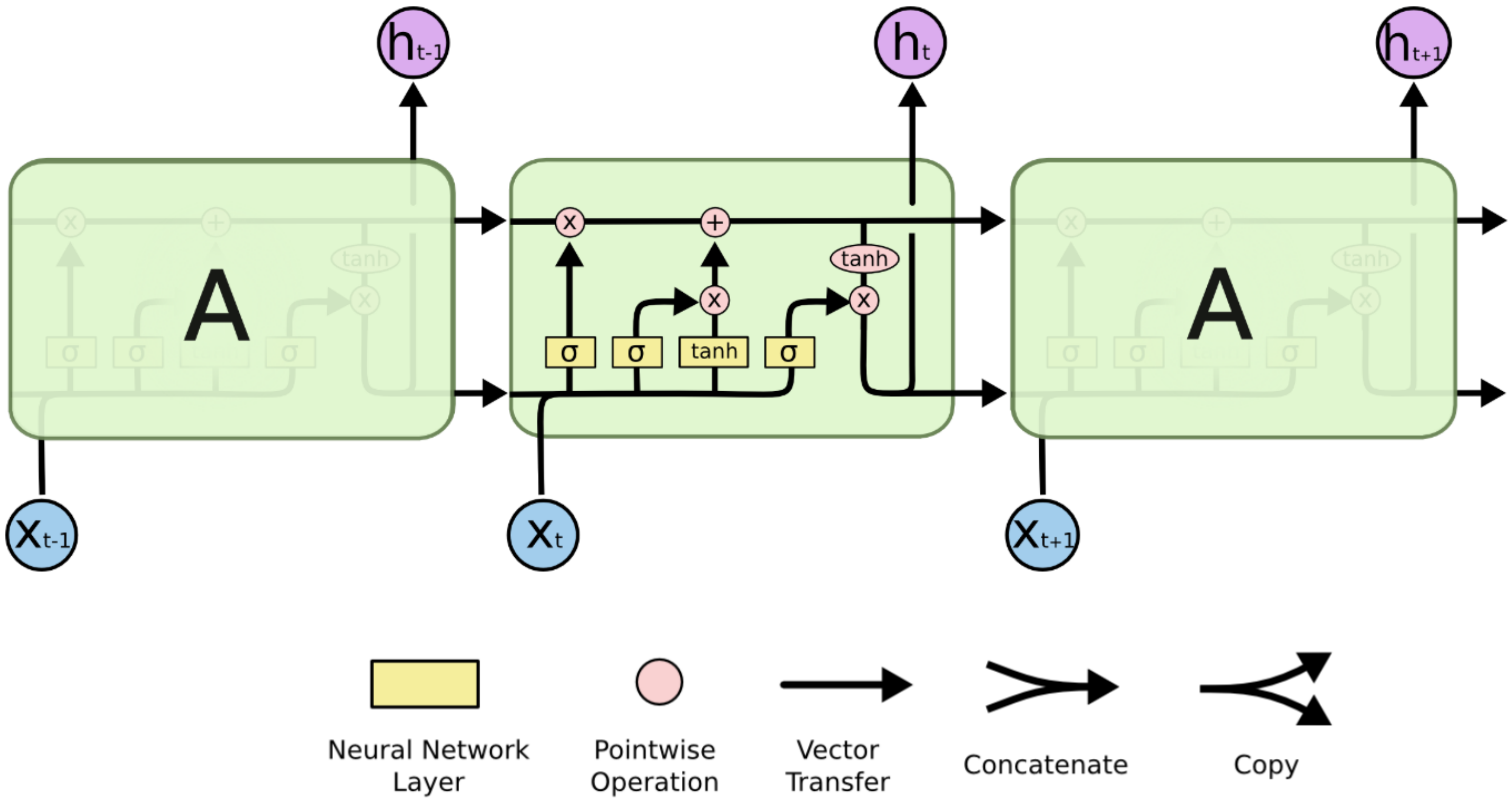
- On step t, there is a hidden state $\boldsymbol{h}^{(t)}$ and a cell state $\boldsymbol{c}^{(t)}$
  - Both are vectors of length n
  - The cell stores long-term information
  - The LSTM can read, erase, and write information from the cell
- The selection of which information is erased/written/read is controlled by three corresponding gates
  - The gates are also vectors of 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

Long Short-Term Memory (LSTM)



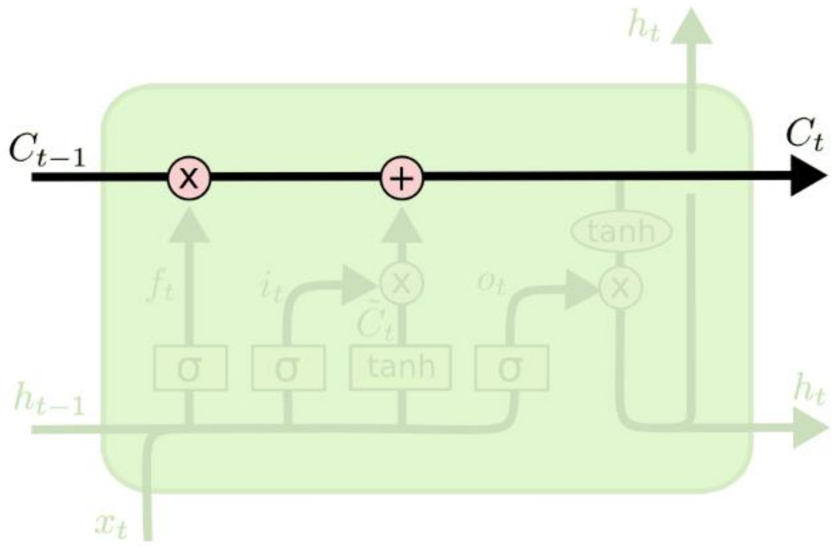
Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM)



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

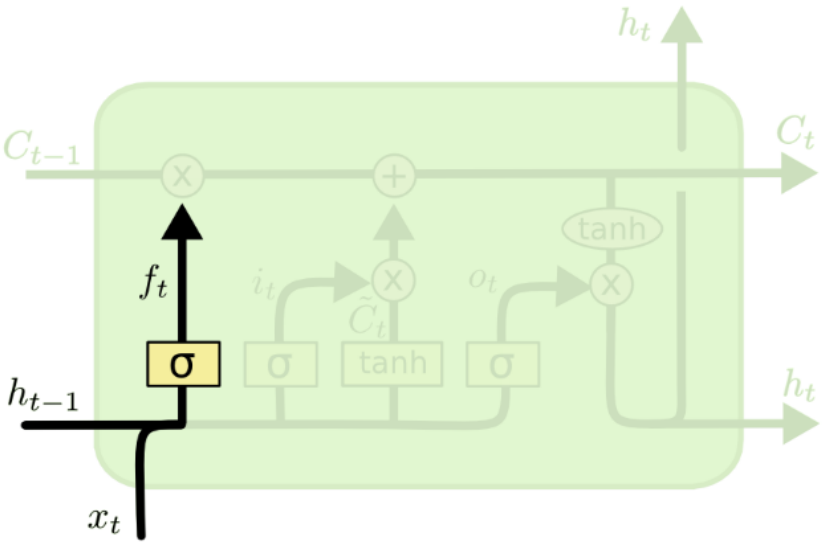
Long Short-Term Memory (LSTM)



The cell stores long-term information

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

Long Short-Term Memory (LSTM)



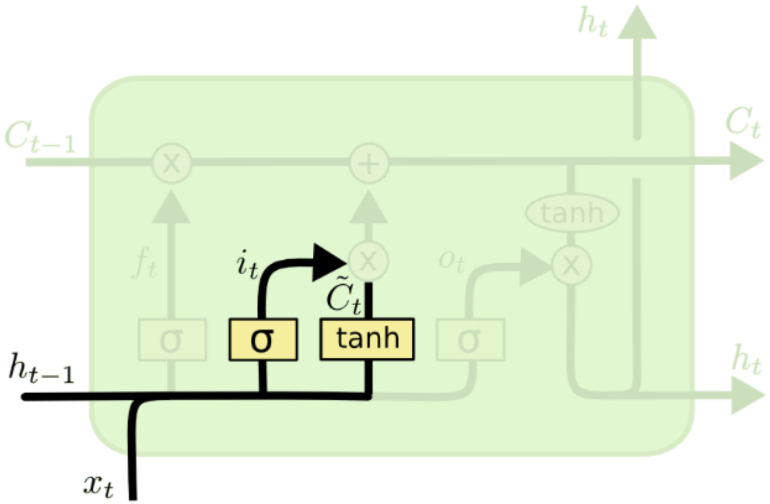
Sigmoid function: gate values are between 0 and 1

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

Decide how much we should forget for each dimension

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

Long Short-Term Memory (LSTM)



Sigmoid function: gate values are between 0 and 1

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

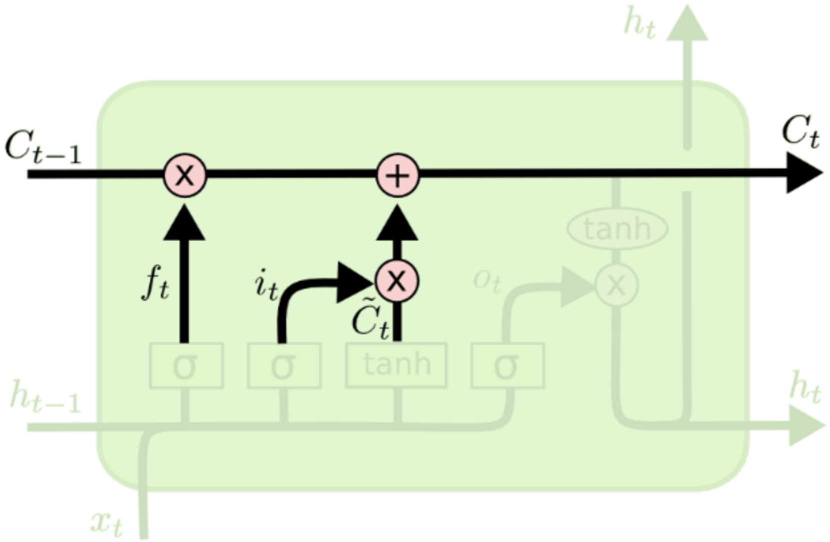
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Decide how much we should write for each dimension

Decide what content we should write

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

Long Short-Term Memory (LSTM)



Forget

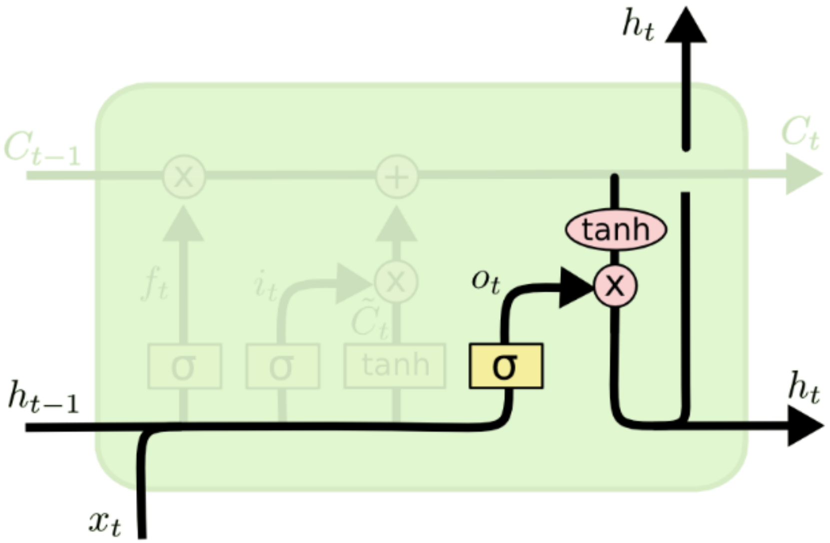
Write

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

Update long-term information (cell)

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

Long Short-Term Memory (LSTM)

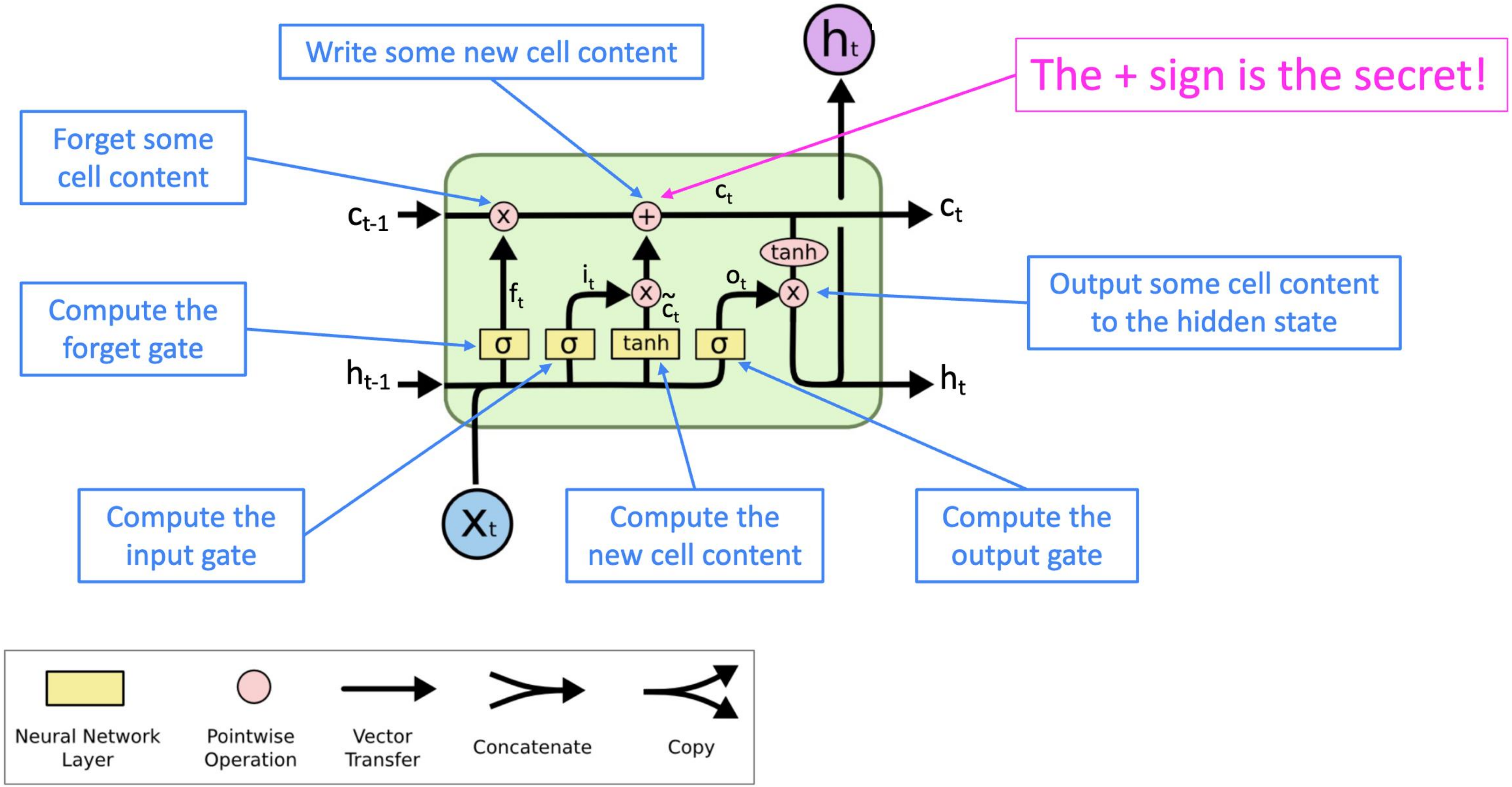


$$h_t = o_t * \tanh\left(C_t\right)$$

Update hidden state

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

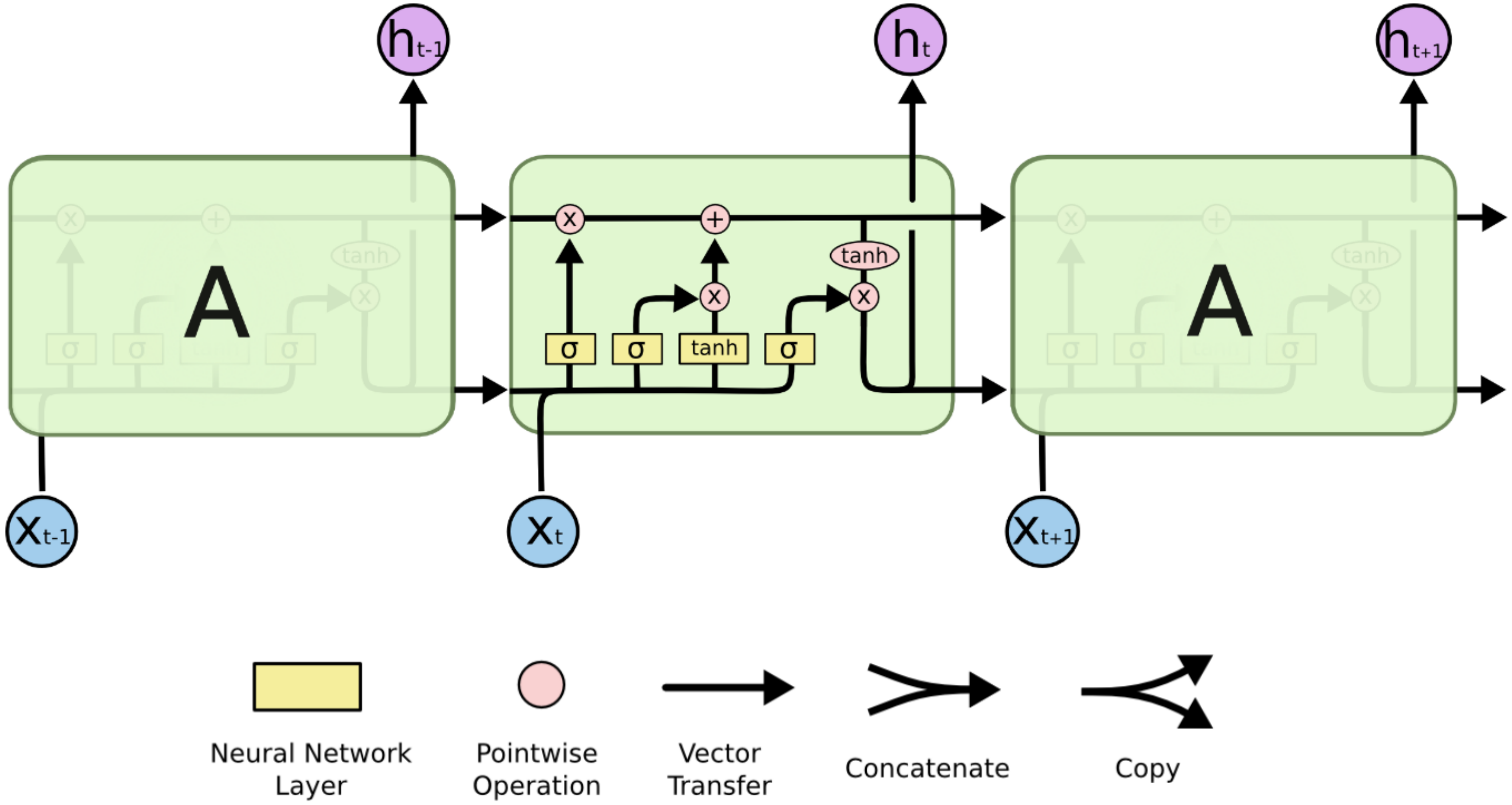
Long Short-Term Memory (LSTM)

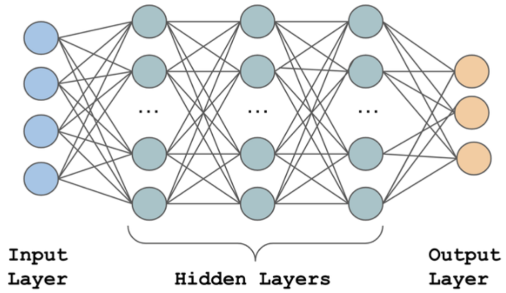


Long Short-Term Memory (LSTM)

- How does LSTM solve vanishing gradients?
  - The LSTM architecture makes it much easier for an RNN to preserve information over many timesteps
  - e.g., if the forget gate is set to 1 for a cell dimension and the input gate set to 0, then the information of that cell is preserved indefinitely

Long Short-Term Memory (LSTM)





$$\text{Loss} = -\sum_{i=1}^{C} y_i \cdot \log(p_i)$$

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

Next Lecture

- Natural Language Processing Basics
- Long Short-Term Memory (LSTM) for generation
- Attention mechanism
- Transformers