Natural Language Processing Workshop

Amy Hemmeter, MSA Class of ‘18
Artificial Intelligence Engineer, Interactions Digital Roots

Preparing Linguistic Data for Deep Learning NLP Solutions

1. Words are not numbers
2. Input can be different lengths

1. **Words are not numbers**
2. Input can be different lengths

Word Vectors

- How do we turn a word into a number?
- We could use “one-hot” vectors - each vector is the length of your vocabulary, and a 1 denotes your word in that vocabulary

Vocabulary:
Man, woman, boy, girl, prince, princess, queen, king, monarch

	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
queen	0	0	0	0	0	0	1	0	0
king	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1

Each word gets a 1x9 vector representation

Source

Problems with One-Hot

- Real vocabularies can be millions of words, meaning your vectors could be incredibly high-dimensional
- Very clunky to work with
- No information encoded in the word vector
- All vectors are orthogonal, no one is more similar to the other than any of the others
- What happens if a word occurs that wasn't in your training set?

Distributional Semantics

“You shall know a word by the company it keeps” - J.R. Firth

- We draw on a linguistic notion called “distributional semantics”
- We can get a large number of contexts for a word and use it to build a vector representation of that word

…government debt problems turning into ***banking*** *crises as happened in 2009…*

…saying that Europe needs unified ***banking*** *regulation to replace the hodgepodge…*

…India has just given its ***banking*** *system a shot in the arm…*

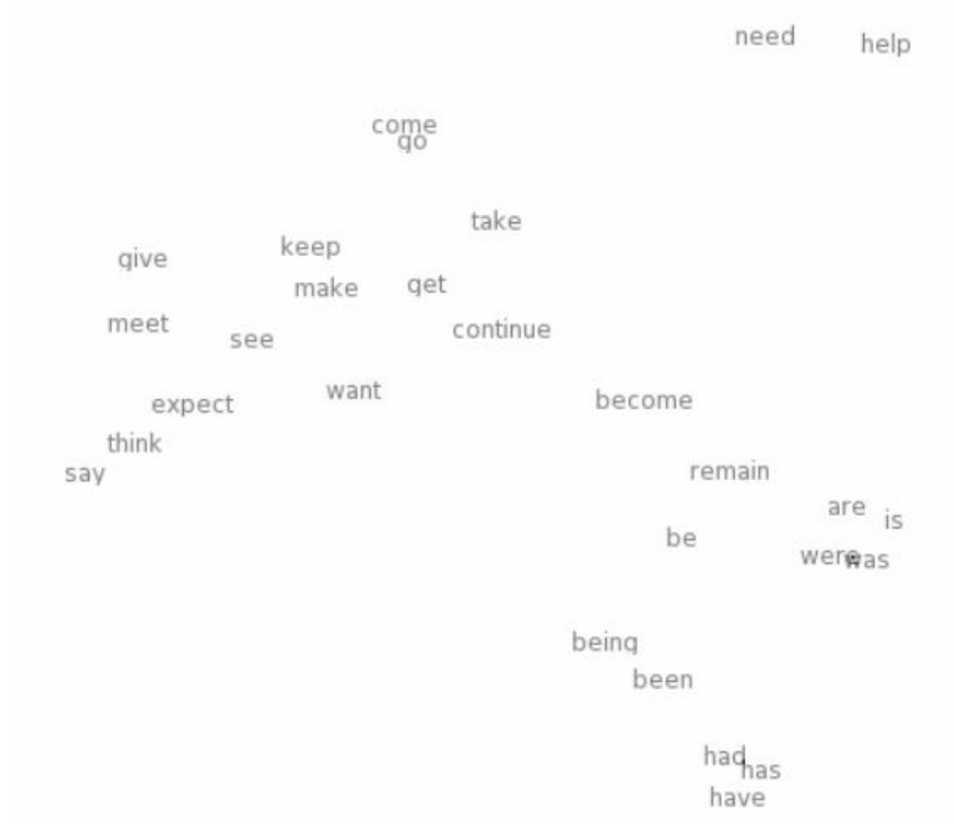
Source: CS224n lecture notes at Stanford

Quick quiz:

Does this sound like anything you've learned about in your text analytics class?

Defining word vectors by their context?

Word meaning visualized

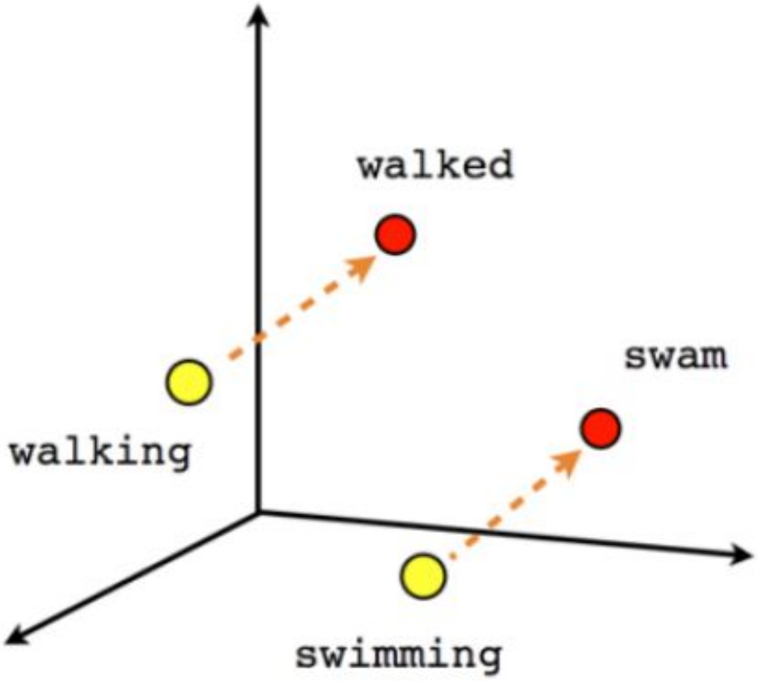


Source: Stanford cs224 notes

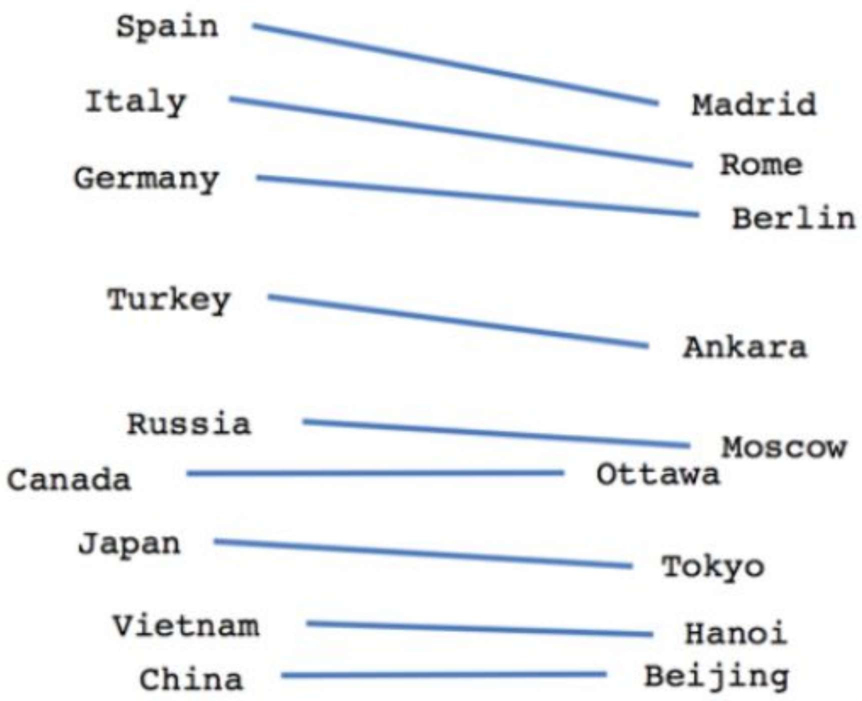
Word Vector Arithmetic

Source: [the morning paper]

Word Vector Arithmetic



Word Vector Arithmetic



How do we get these vectors?

- Training a neural network with one-hot vector encodings as inputs to do a “fake” task
- We then pop out the hidden layer of that neural network and use that as our embedding

Quick quiz:

What NLP task have we already talked about (hint: one that takes into account the context of words) might be a good fake task for this algorithm to use?

word2vec

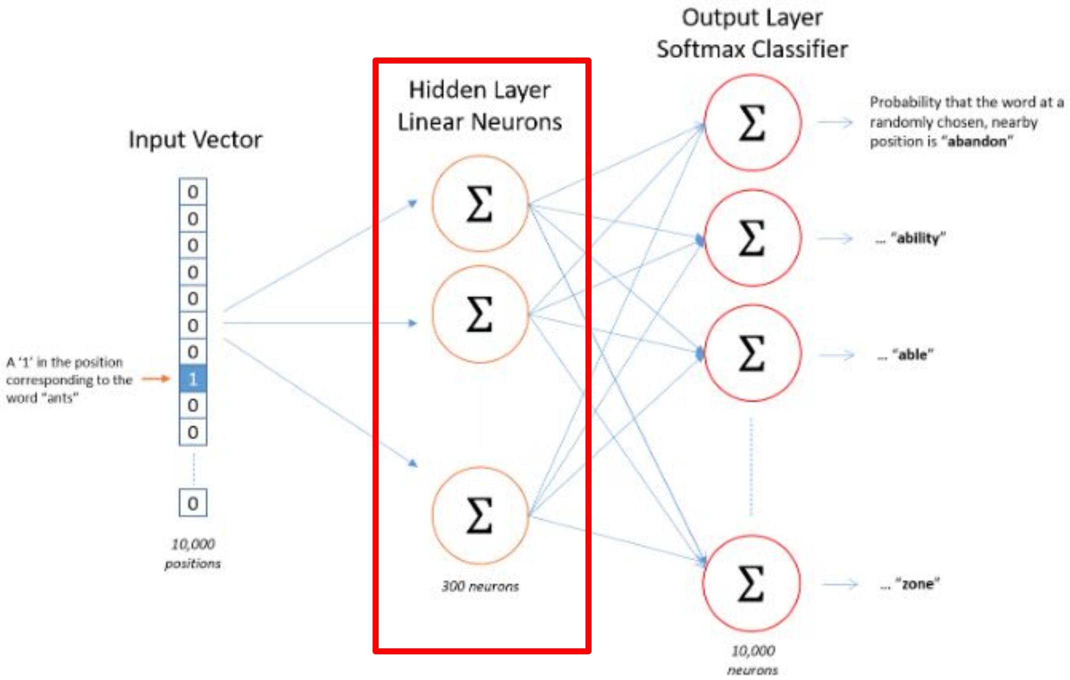
- Word2vec (Mikolov et al. 2013) is a framework for learning word vectors
- Inputs are vectors the same length as your vocabulary - so if you have 10,000 words you have a vectors of dimension 10,000
- There are two versions of word2vec
  - SkipGram - predict context words from center word
  - Continuous Bag of Words - predict center word from (bag of) context words

2. Sliding Window

derekchia.com

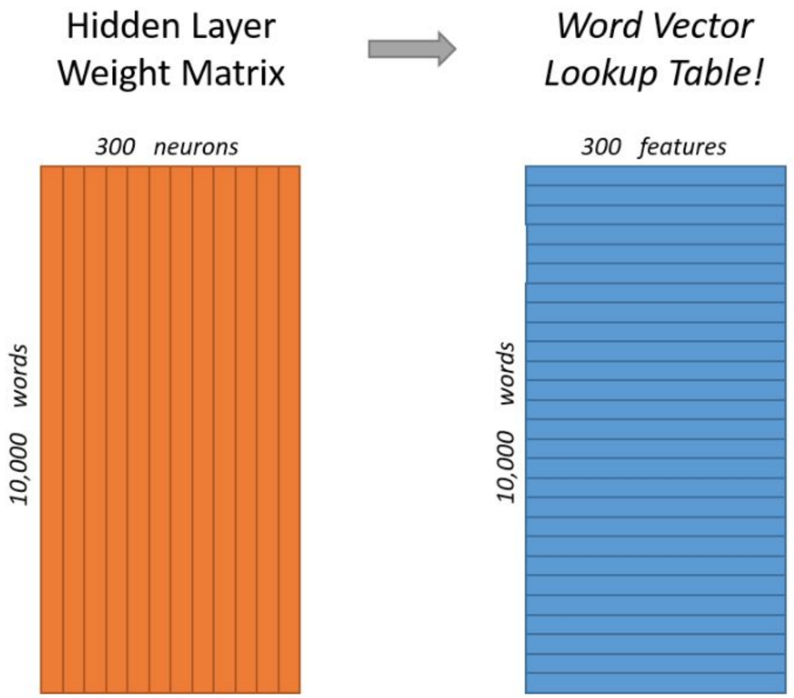
#1	natural	language	processing	and	machine	learning	is	fun	and	exciting	#1
	Xκ	Y(c=1)	Y(c=2)								
#2	natural	language	processing	and	machine	learning	is	fun	and	exciting	#2
	Y(c=1)	Xκ	Y(c=2)	Y(c=3)							
#3	natural	language	processing	and	machine	learning	is	fun	and	exciting	#3
	Y(c=1)	Y(c=2)	Xκ	Y(c=3)	Y(c=4)						
#4	natural	language	processing	and	machine	learning	is	fun	and	exciting	#4
		Y(c=1)	Y(c=2)	Xκ	Y(c=3)	Y(c=4)					
#5	natural	language	processing	and	machine	learning	is	fun	and	exciting	#5
			Y(c=1)	Y(c=2)	Xκ	Y(c=3)	Y(c=4)				
#6	natural	language	processing	and	machine	learning	is	fun	and	exciting	#6
				Y(c=1)	Y(c=2)	Xκ	Y(c=3)	Y(c=4)			
#7	natural	language	processing	and	machine	learning	is	fun	and	exciting	#7
					Y(c=1)	Y(c=2)	Xκ	Y(c=3)	Y(c=4)		
#8	natural	language	processing	and	machine	learning	is	fun	and	exciting	#8
						Y(c=1)	Y(c=2)	Xκ	Y(c=3)	Y(c=4)	
#9	natural	language	processing	and	machine	learning	is	fun	and	exciting	#9
							Y(c=1)	Y(c=2)	Xκ	Y(c=3)	
#10	natural	language	processing	and	machine	learning	is	fun	and	exciting	#10
								Y(c=1)	Y(c=2)	Xκ	

How do we get these vectors?



Source: McCormick ML

What do we do when we pop out the layer?



How do we evaluate word vectors?

- There are a couple different kinds of word vectors - word2vec and GloVe, as well as contextualized word embeddings like BERT or ELMo (see further reading)
- Intrinsic evaluation
  - Evaluate on an intermediate task -- such as word vector analogies (like man:woman, king:queen and visualizations
  - Allows you to understand more about the word embeddings but not downstream tasks
- Extrinsic evaluation
  - How do these vectors perform in the task you need them for? Does your classification accuracy go up when you use these vectors for the same data vs. another kind of word vector?

Let’s play around with word vectors!

If you want to follow along, open your “exploring-word-vectors.ipynb” file in the code folder I gave you.

Deep Learning Solutions for NLP

Classification Review

- You have a training set of samples consisting of:
  - x_i inputs, e.g. words (indices to an embedding lookup table or vectors), sentences, documents, etc.
  - y_i outputs which are *labels* of a certain number of classes (binary or multi-class)
- Example:
  - Sentiment analysis
  - Intents in conversational AI
- Architecture
  - Various flavors of deep learning (we'll go over several in this class) topped off by a softmax layer
  - Generally we use cross-entropy loss as our objective function

Classification Review

Softmax

- You can think of it like an activation function (a la the sigmoid or the ReLU) for the final (output) layer of your network
- Normalizes the output of classifier to be some probability for each class, where they all add up to 1:

$$\boxed{\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}}$$

Sample output: $\begin{bmatrix} 0.25 \\ 0.21 \\ 0.9 \\ 0.45 \end{bmatrix}$

- This makes it easier to choose the y with the maximum probability

Softmax Example

Original output:

$$\sigma(x_j) = \frac{e^{x_j}}{\boxed{\sum_i e^{x_i}}}$$

$$\begin{bmatrix} 3.0 \\ 5.0 \\ 2.0 \end{bmatrix}$$

$$\begin{aligned} & e^3 + e^5 + e^2 \\ &= 20.0855 + 148.4132 + 7.3891 \\ &= \boxed{175.8878} \end{aligned}$$

Softmax Example

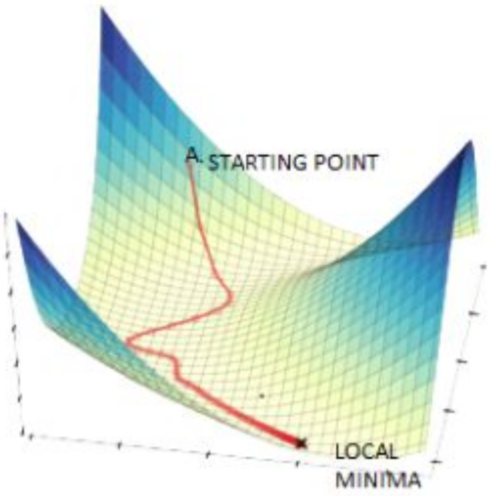
Original output:

$$\sigma(x_j) = \boxed{\frac{e^{x_j}}{\sum_i e^{x_i}}}$$

$$\begin{bmatrix} 3.0 \\ 5.0 \\ 2.0 \end{bmatrix} \qquad \begin{matrix} 20.0855/175.8878 = \\ 148.4132/175.8878 = \\ 7.3891/175.8878 = \end{matrix} \qquad \begin{bmatrix} 0.12 \\ 0.84 \\ 0.04 \end{bmatrix}$$

Gradient Descent Review (Hopefully)

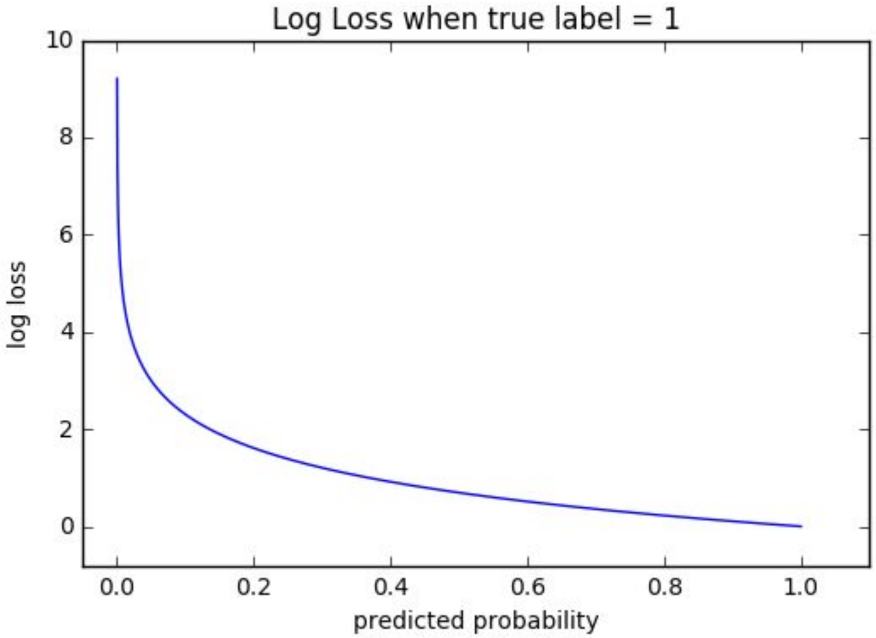
- How to find the minimum of your loss function
- Uses the derivative of the loss function with respect to your parameters to take a step in a direction towards the minimum
- How most machine learning algorithms work
- The rate at which you move down this slope is called the learning rate



Picture from Data Science Central,
https://www.datasciencecentral.com/profiles/blogs/alternatives-to-the-gradient-descent-algorithm

Cross Entropy Loss

- Cross-Entropy Loss is the loss function we're trying to minimize in classification
- We're attempting to maximize the probability of the correct class y
- Cross entropy loss measures the deviation of the predicted probability from the actual probability



Source: [ML cheatsheet]() on readthedocs.io

Cross-Entropy Loss

$$-\sum_{c=1}^{M} y_{o,c} log(p_{o,c})$$

Cross-Entropy Loss

$$-\sum_{c=1}^{M} y_{o,c} log(p_{o,c})$$

Output of softmax:

$$\begin{bmatrix} 0.12 \\ 0.84 \\ 0.04 \end{bmatrix}$$

Correct label:

$$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$-(1 \times log(0.84))$$

$$= 0.174353387$$

Cross-Entropy Loss

$$-\sum_{c=1}^{M} y_{o,c} log(p_{o,c})$$

Output of softmax:

$$\begin{bmatrix} 0.84 \\ 0.12 \\ 0.04 \end{bmatrix}$$

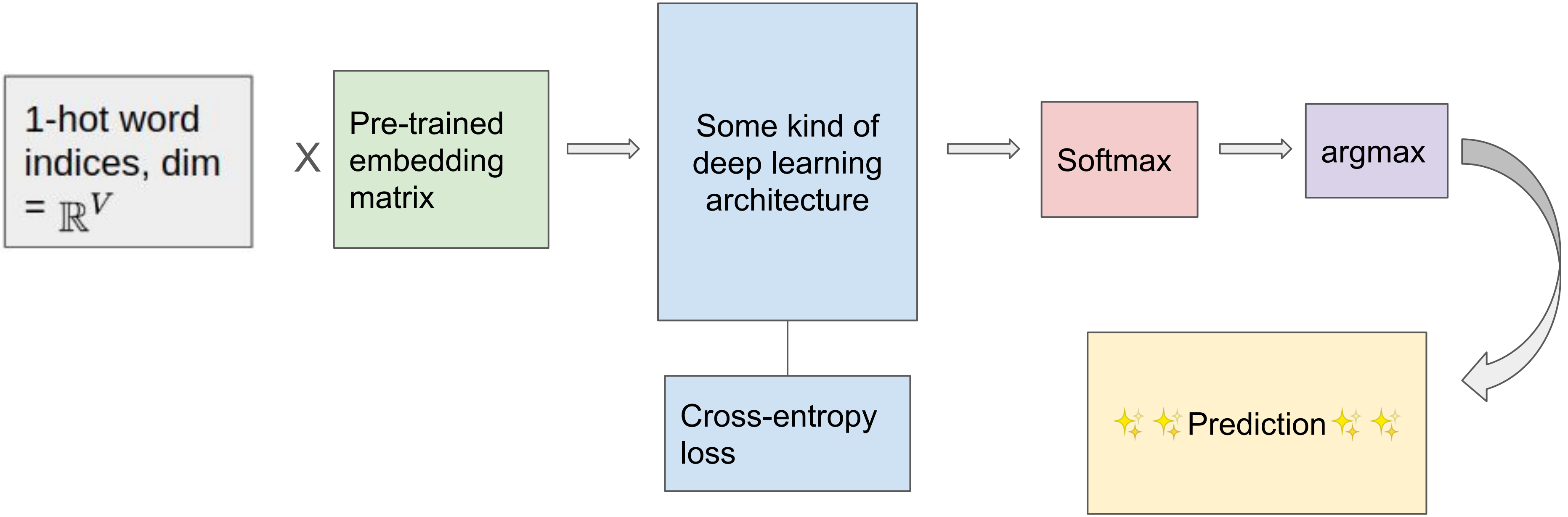
Correct label:

$$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

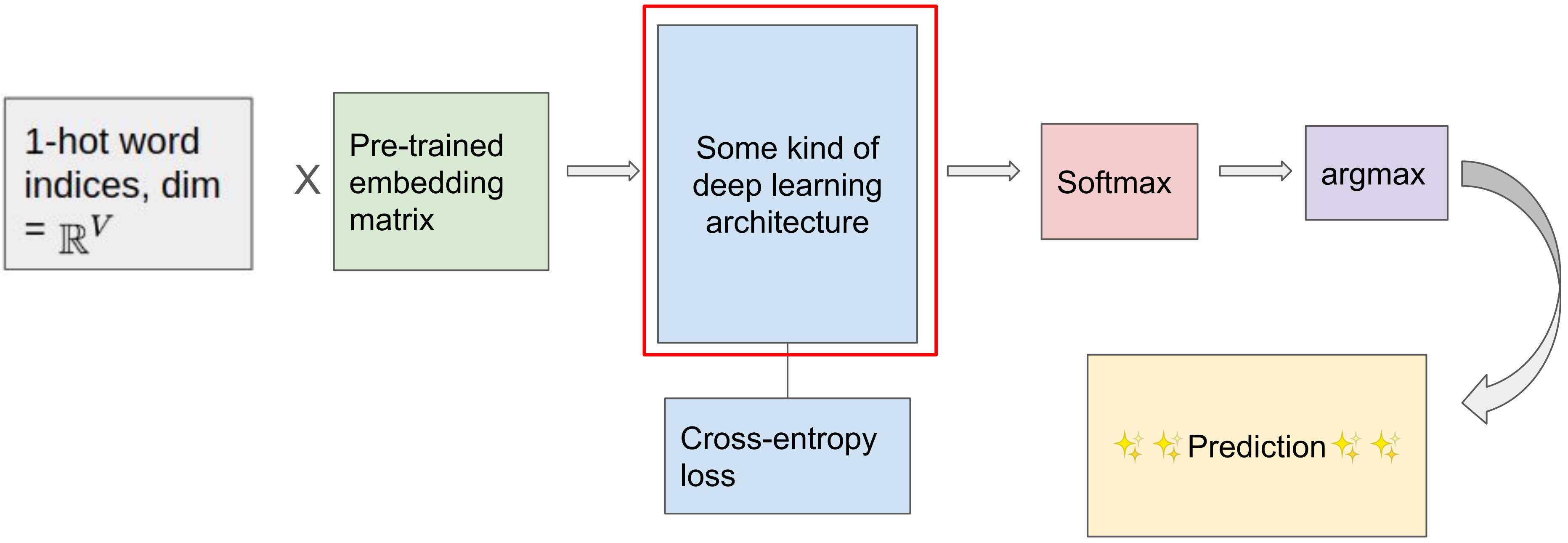
$$-(1 \times log(0.12))$$

$$= \mathbf{2.12026353}$$

Putting it together



Putting it together

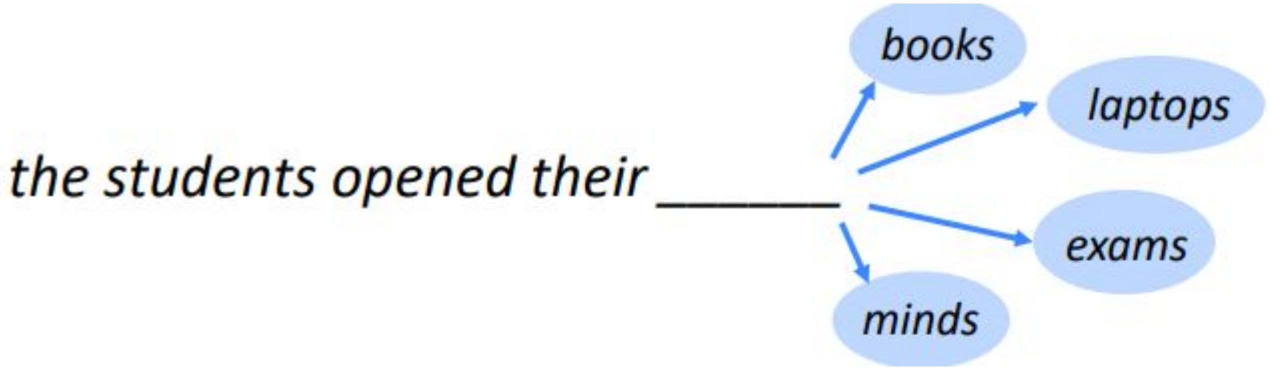


Deep Learning Architectures for NLP

- (Vanilla) Recurrent Neural Networks (RNNs)
- Long Short-Term Memory (LSTMs)
- CNNs - they’re not just for image processing anymore!

Language Modeling with Recurrent Neural Networks (RNNs)

- Words and characters are *not* independent in language, therefore they should not be treated as independent in your neural net
- We'll be revisiting language modeling



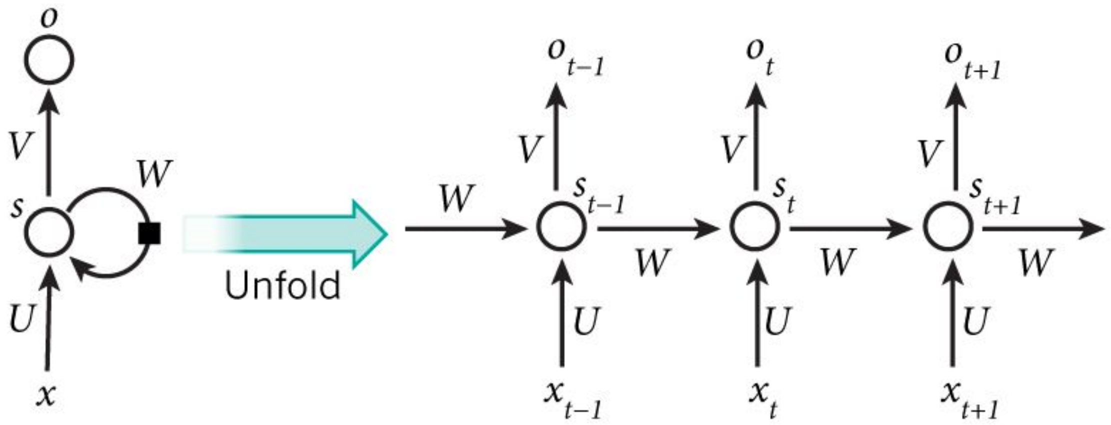
Language Modeling with Recurrent Neural Networks (RNNs)

- Increasing the size of your n-gram in an n-gram language model causes many problems with computation and sparsity
- However, in order to create output that is more coherent, you need bigger n-grams
- Consider the following sentence:

  “today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .”

Language Modeling with Recurrent Neural Networks (RNNs)

- RNNs allow us to use a “memory” for the information that came before



$$s_t = f(Ux_t + Ws_{t-1})$$

Language Modeling with Recurrent Neural Networks (RNNs)

- To train
  - Get large corpus of text data with a sequence of words x1, …. xT
  - Predict probability of distribution for each word, given words so far (classification with large number of classes)
  - Loss function is the cross entropy between the predicted next word and the true next word (one-hot vector of class index)
  - Average all of these values to get the overall loss for the training set

Quick quiz:

What metric do we use to evaluate language models again?

Language Modeling with Recurrent Neural Networks (RNNs)

- RNNs can process any length of input, the model size doesn’t increase for longer input
- However, RNNs are slow to compute -- they are difficult to parallelize because the inputs are dependent on one another
- Although they hold some memory, in practice it is difficult to access information from many steps back (5-10 steps)
- Vanishing gradient problem (more on this in a second)

Language Modeling with Recurrent Neural Networks (RNNs)

PANDARUS:

Alas, I think he shall be come approached and the day

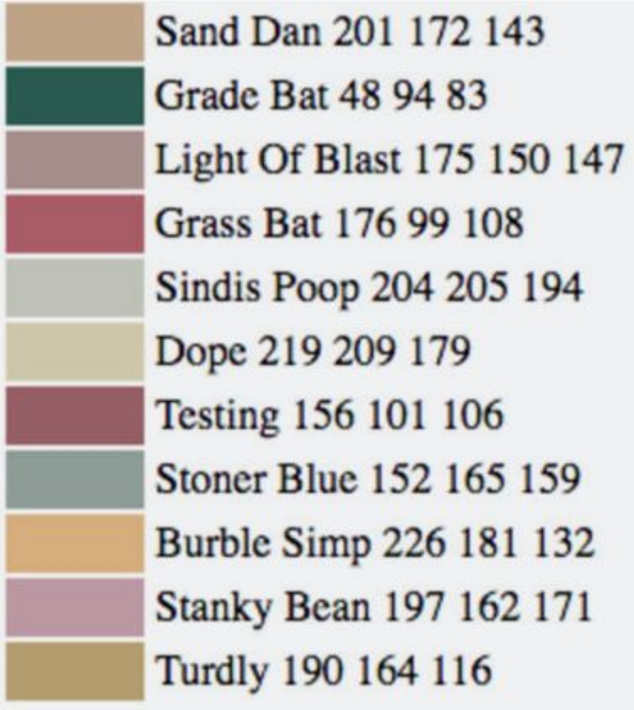
When little srain would be attain’d into being never fed,

And who is but a chain and subjects of his death,

I should not sleep

What are RNNs capable of?

Clardic Fug	112 113 84
Snowbonk	201 199 165
Catbabel	97 93 68
Bunflow	190 174 155
Ronching Blue	121 114 125
Bank Butt	221 196 199
Caring Tan	171 166 170
Stargoon	233 191 141
Sink	176 138 110
Stummy Beige	216 200 185
Dorkwood	61 63 66
Flower	178 184 196



Sand Dan	201 172 143
Grade Bat	48 94 83
Light Of Blast	175 150 147
Grass Bat	176 99 108
Sindis Poop	204 205 194
Dope	219 209 179
Testing	156 101 106
Stoner Blue	152 165 159
Burble Simp	226 181 132
Stanky Bean	197 162 171
Turdly	190 164 116

Source: [AI Weirdness]

What is machine learning not capable of?

DINERS, DRIVE-INS, AND DIVES

INT. PARKING LOT

GUY FIERI sits in a convertible. He looks like America.

GUY FIERI
I’m Guy Fieri and there’s nothing you can do about it. Today I’m eating it all.

Guy takes a bite out of his hair.

INT. DINER’S KITCHEN

Guy and a CHEF stand in a kitchen. Guy has 3 pairs of sunglasses on. The sun can’t get him.

GUY FIERI
Prove to me you can panini!

The chef starts boiling a pot of milk. He’s scared.

CHEF
Flavortown is near.

Drawbacks of RNNs

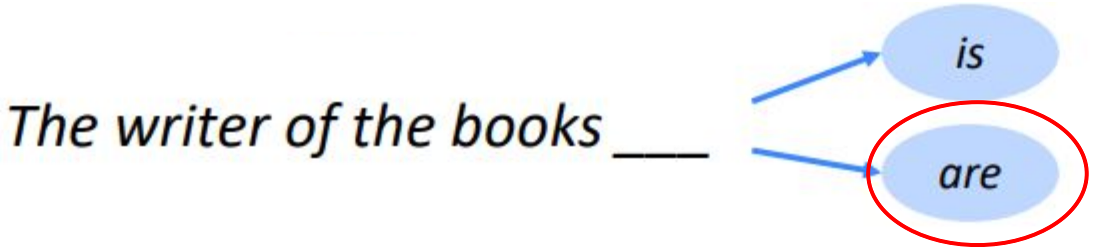
Consider the following word prediction task:

“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 _________”

Drawbacks of RNNs

- We get something called the Vanishing Gradient Problem
- The gradient can be viewed as a measure of the effect of the past on the future
- If the gradient becomes too small, it’s impossible to tell whether there’s no dependency between two distant time steps or we have the wrong parameters to capture the dependency between two distant time steps
- This is an artifact of using the chain rule for backpropagation

Drawbacks of RNNs



Long Short-Term Memory (LSTMs)

- Like an RNN, but includes a cell state in addition to a hidden state
- The cell stores long-term information
- The LSTM can erase, write and read information from the cell with gates, which are vectors of length n - the same length as the hidden state and the cell state

Long Short-Term Memory (LSTMs)

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

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

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

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

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

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

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

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

All these are vectors of same length n

Gates are applied using element-wise product

Source: [Stanford class]

Long Short-Term Memory (LSTMs)

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

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

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

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

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

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

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

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

All these are vectors of same length n

Gates are applied using element-wise product

Source: Stanford class

Long Short-Term Memory (LSTMs)

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

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

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

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

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

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

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

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

All these are vectors of same length n

Gates are applied using element-wise product

Source: [Stanford class]

Long Short-Term Memory (LSTMs)

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

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

Sigmoid function: all gate values are between 0 and 1

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

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

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

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

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

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

All these are vectors of same length n

Gates are applied using element-wise product

Source: [Stanford class]

Long Short-Term Memory (LSTMs)

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

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

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

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

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

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

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

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

All these are vectors of same length n

Gates are applied using element-wise product

Source: [Stanford class]

Long Short-Term Memory (LSTMs)

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

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

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

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

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

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

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

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

All these are vectors of same length *n*

Gates are applied using element-wise product

Source: [Stanford class]

Long Short-Term Memory (LSTMs)

Sigmoid function: all gate values are between 0 and 1

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

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

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

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

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

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

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

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

All these are vectors of same length n

Gates are applied using element-wise product

Source: [Stanford class]

Long Short-Term Memory (LSTMs)

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

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

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

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

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

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

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

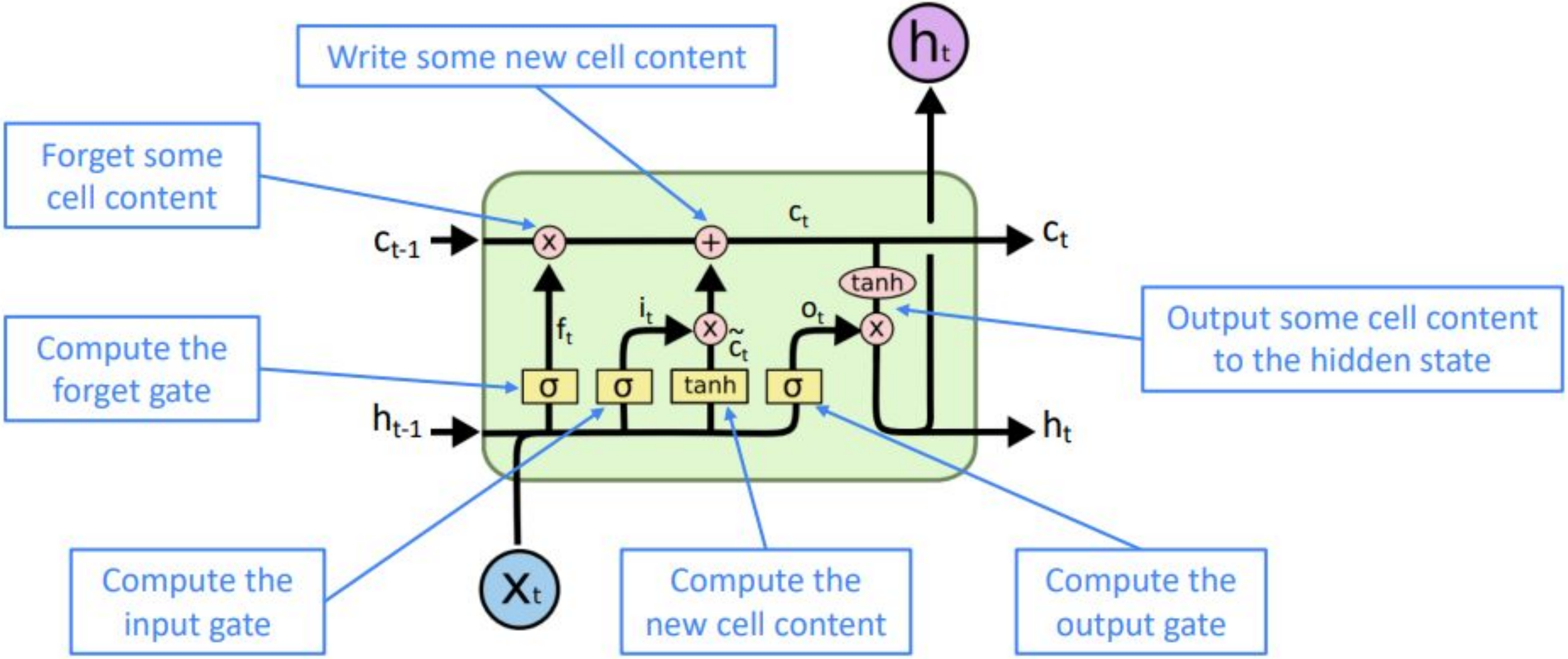
$$\boxed{\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}}$$

All these are vectors of same length n

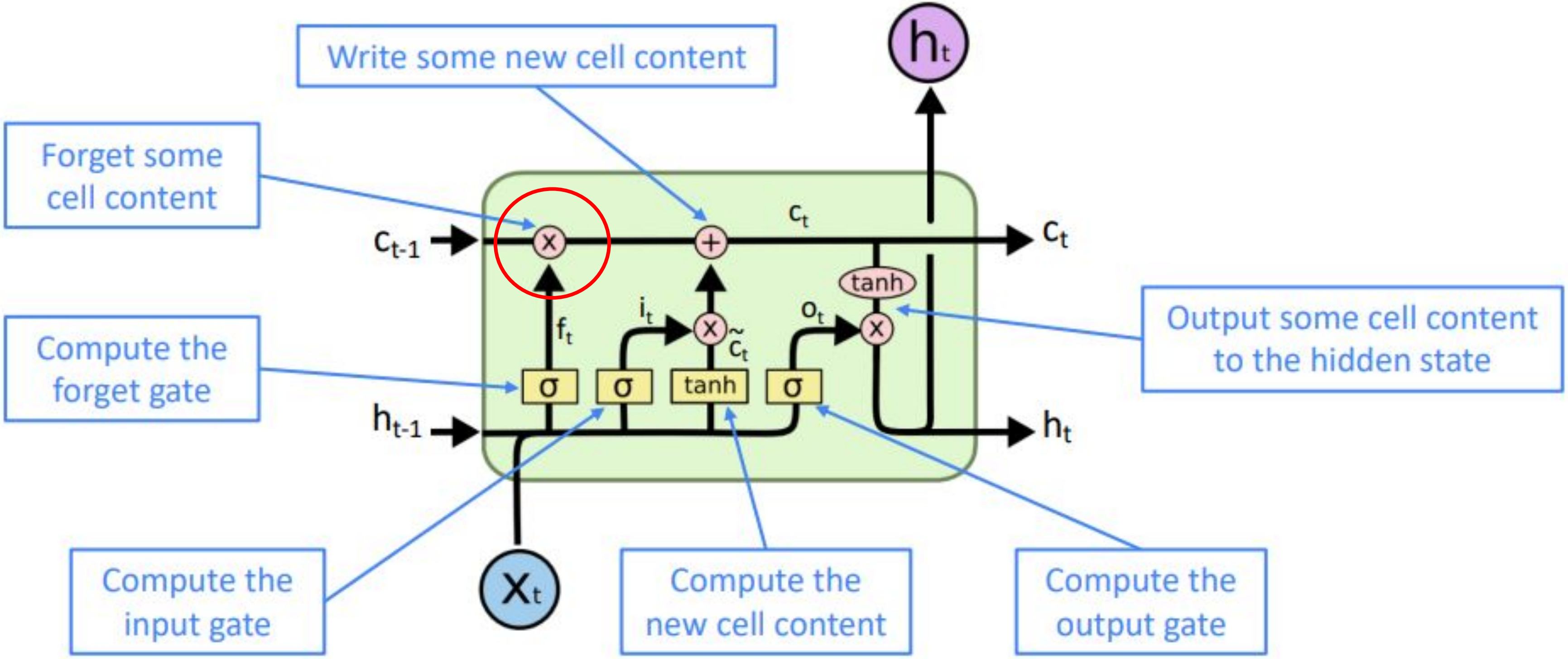
Gates are applied using element-wise product

Source: [Stanford class]

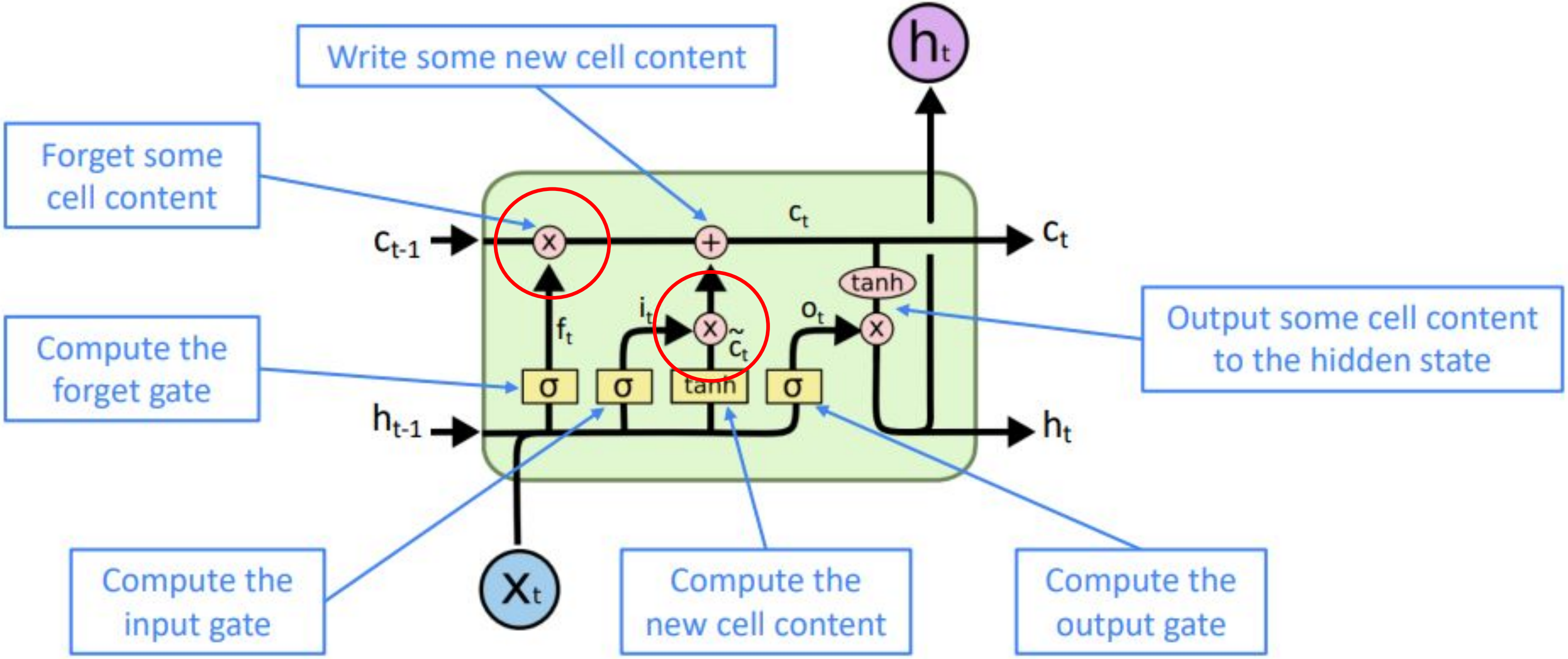
Long Short-Term Memory (LSTMs)



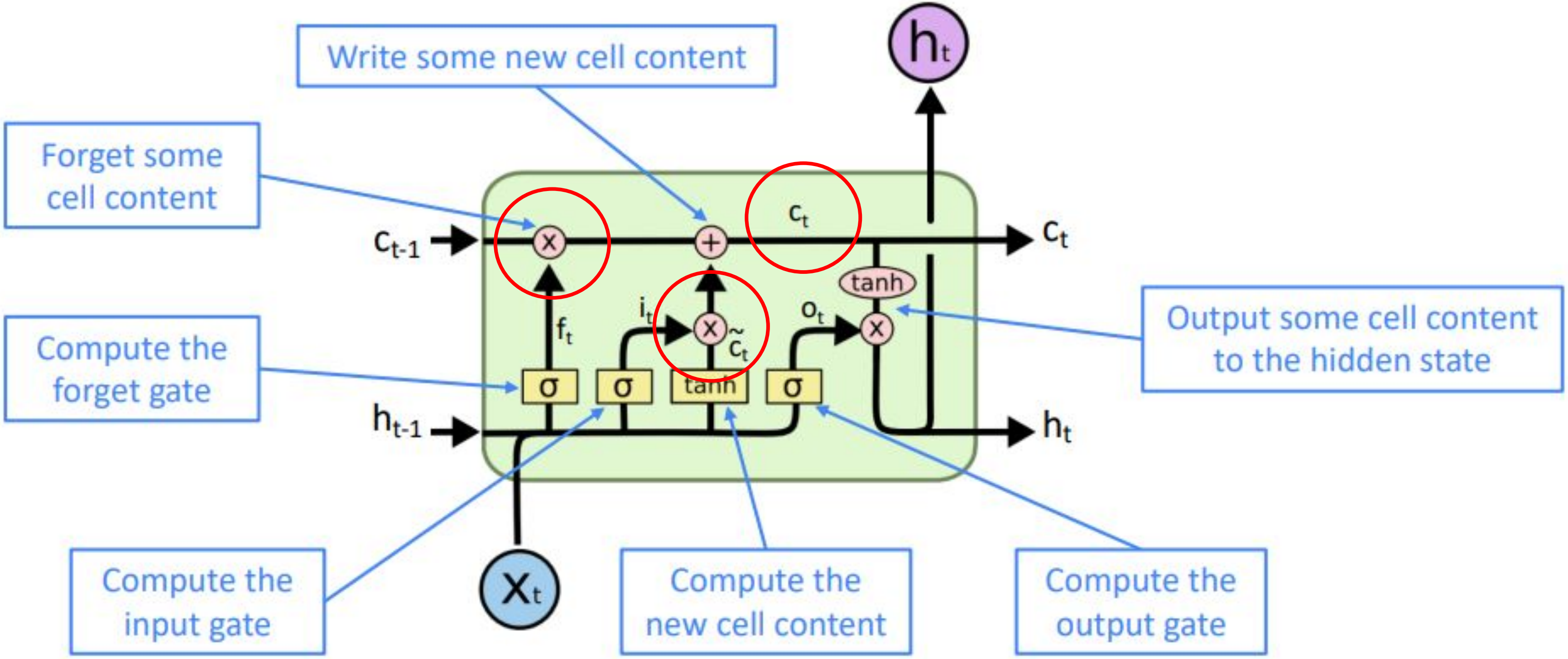
Long Short-Term Memory (LSTMs)



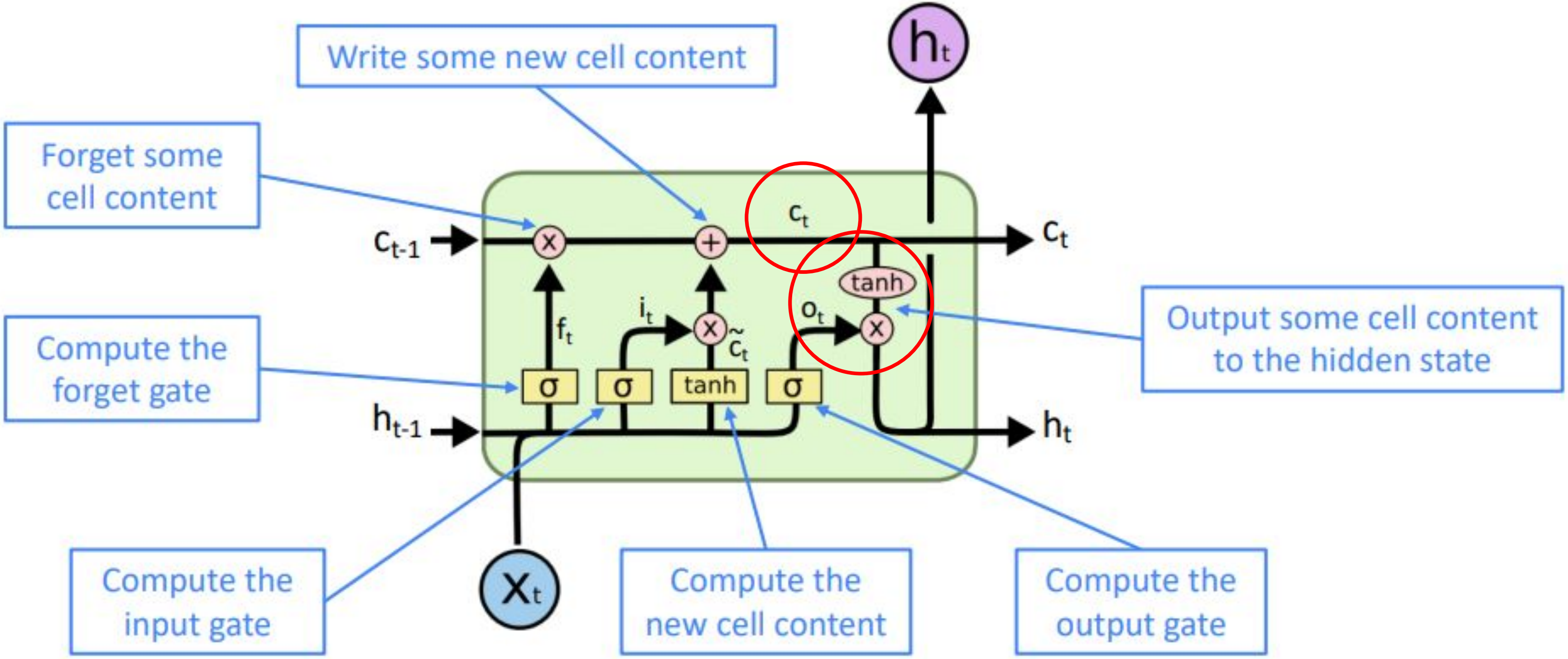
Long Short-Term Memory (LSTMs)



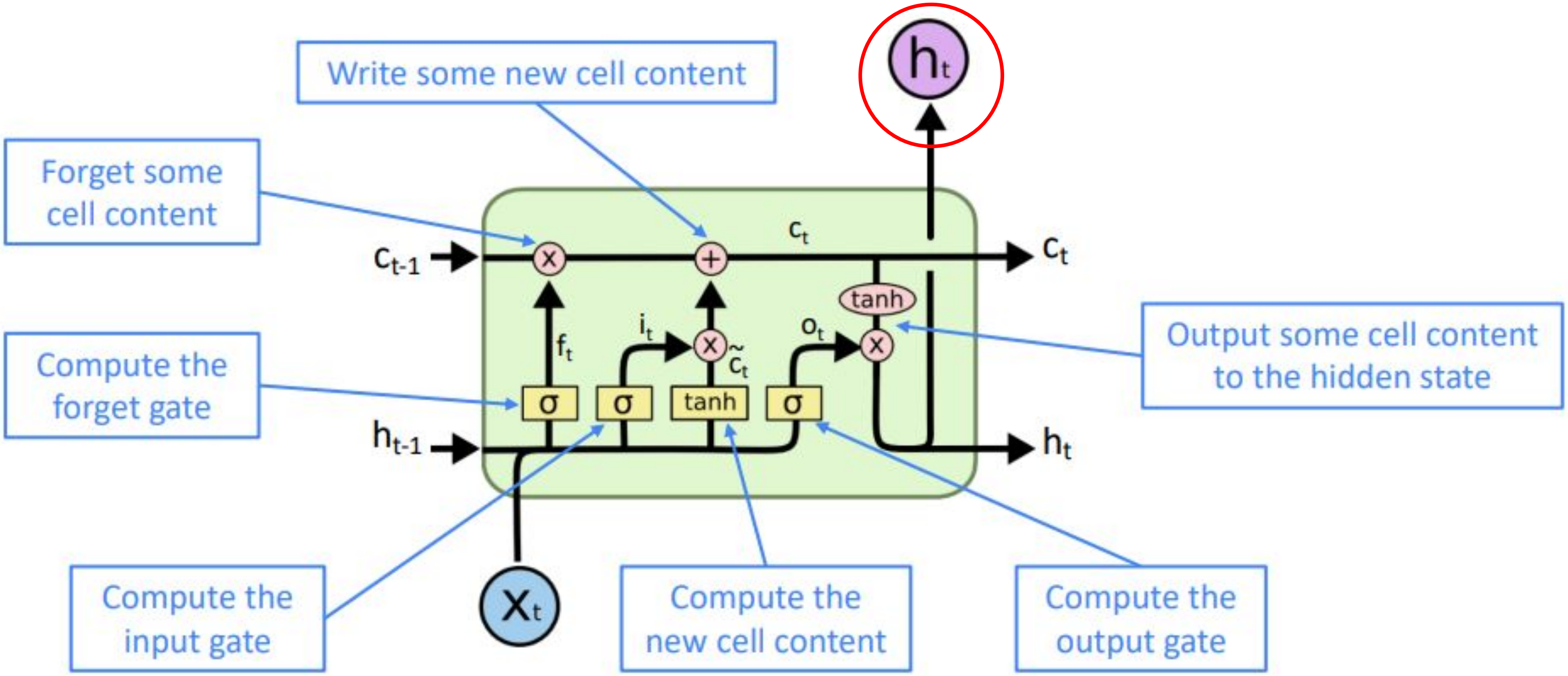
Long Short-Term Memory (LSTMs)



Long Short-Term Memory (LSTMs)



Long Short-Term Memory (LSTMs)

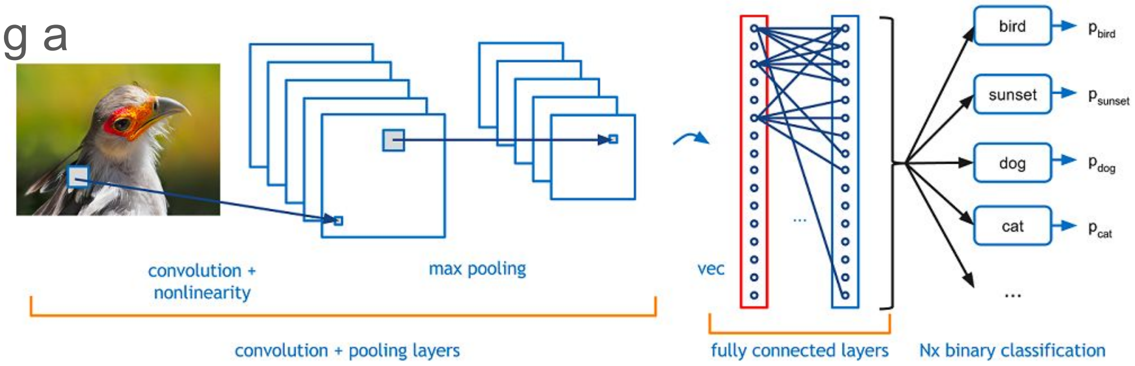


Long Short-Term Memory (LSTMs)

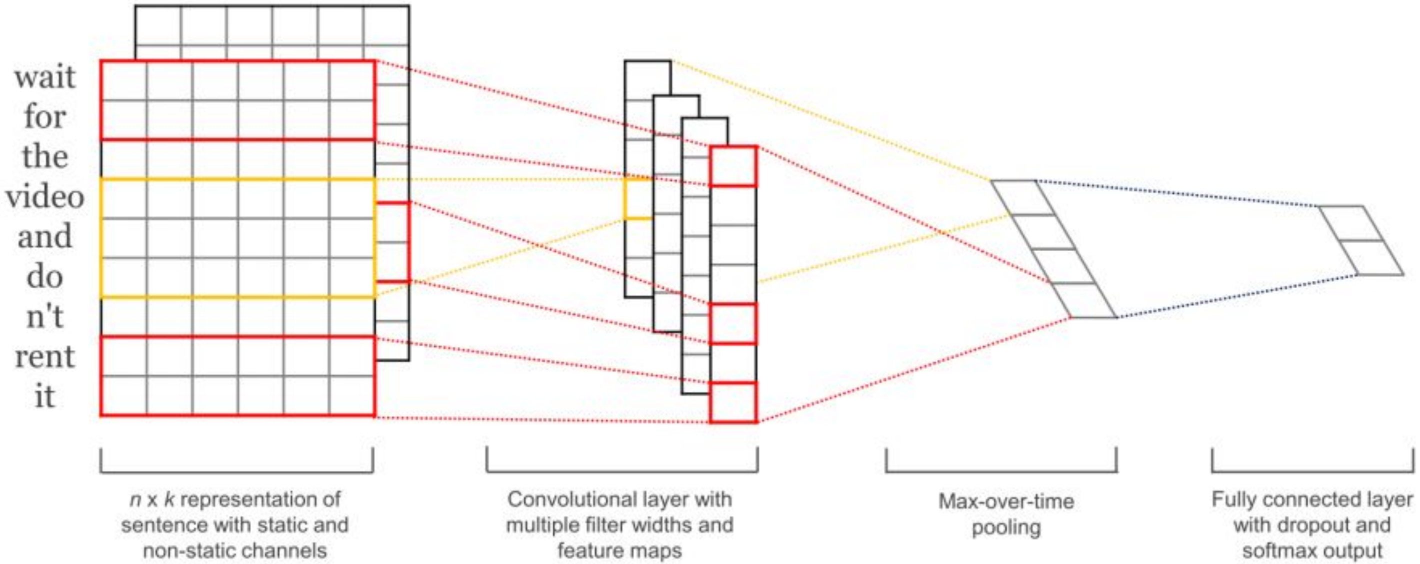
- Because we have the forgetting gate, it’s easier to keep things around for a long time, mostly solving our vanishing gradient problem
- Perform better on a large number of tasks
- Makes sense for language because of the time dependencies and long-range dependencies

CNN Review

- 2-dim image with sliding window
- At each point you create a convolved value by applying a kernel or filter to the output
- You apply some kind of pooling to the outputs to squash the dimensions
- You then use the smaller-dimensional layers to feed as input to your fully-connected layers



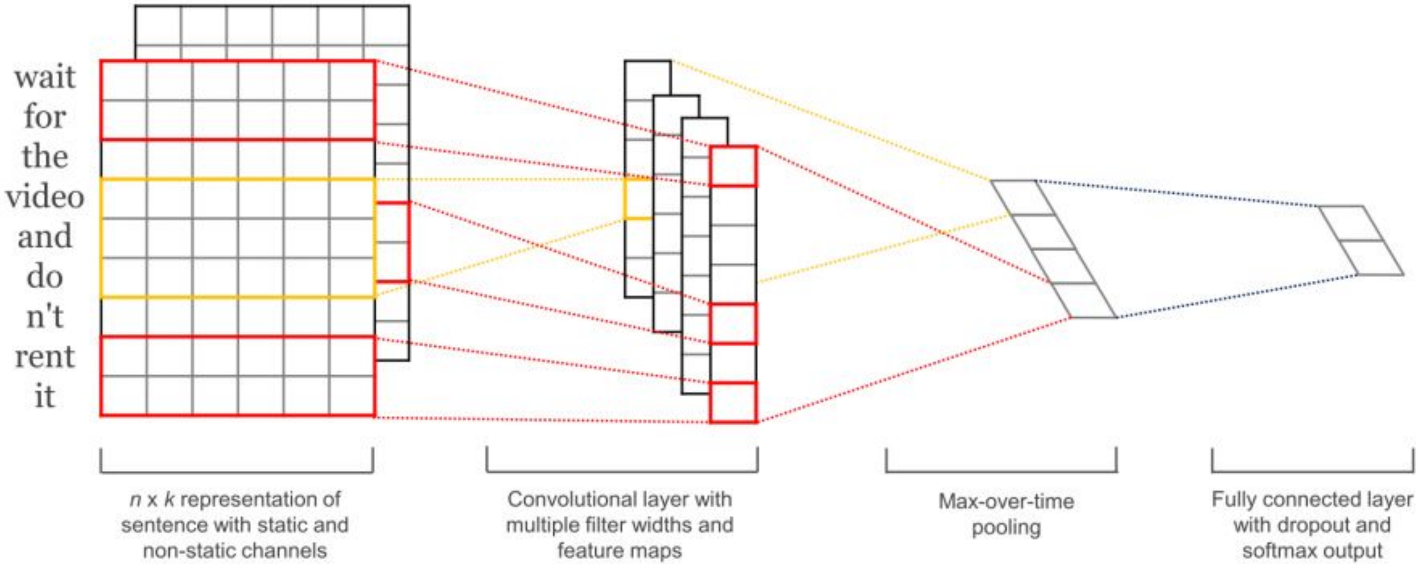
CNNs for NLP


wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Quick quiz

The filter mechanism in this CNN architecture is equivalent to what, from our first lecture?

CNNs for NLP



wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Next Time

- We'll talk more preprocessing and cut our teeth on some code!
- And we'll work on some more practical tips and tricks for any analytics professional dealing with text data