446 Section 09

Plans for today!

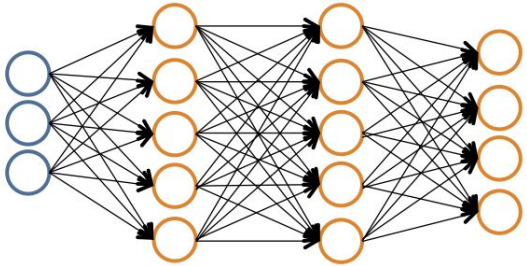
1. This
2. Reminders
3. CNNs
4. RNNs / LSTMs
5. Attention / Transformers

Reminders

- HW4 due June 3rd!
- Final is Wednesday June 10!
  - < 2 weeks from today

CNNs

Neural Networks



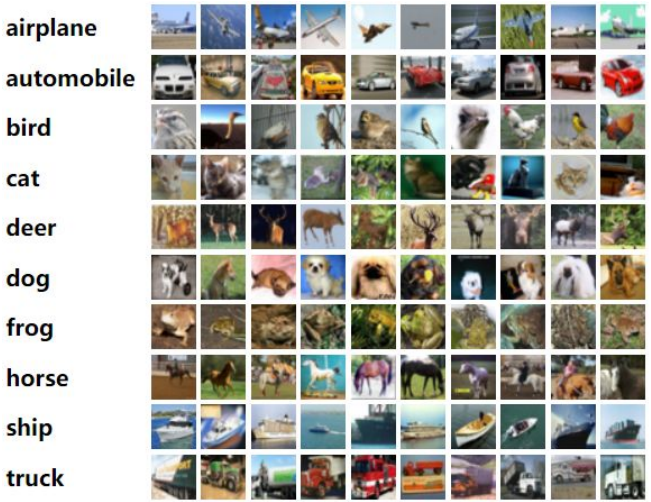
So far, we've looked at neural networks that look like this:

for the XOR problem and on the MNIST dataset (HW3).

- Both of these are pretty simple problems and our models were fairly small.

Now, consider more complex image data, like:

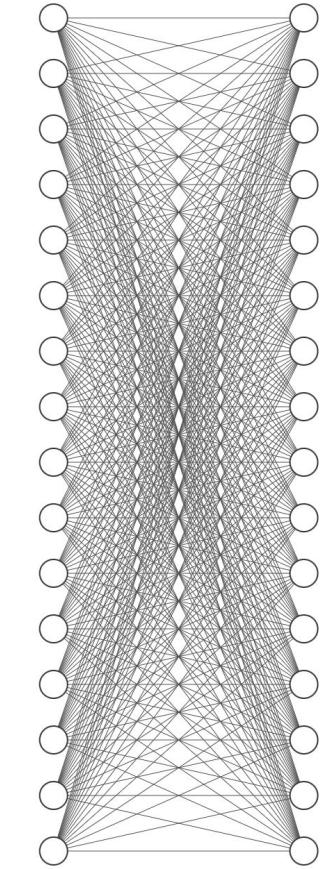
- Why might the same fully connected neural network architecture not work very well for these images?



Neural Networks: Images

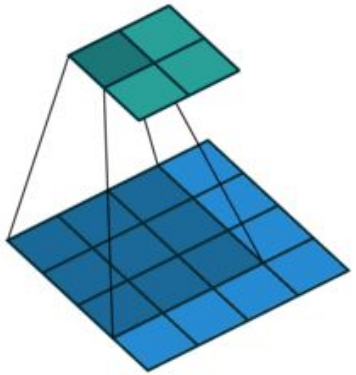
- A dense neural network has *a lot* of parameters to learn.
  - Moderate sized images might be 3 x 200 x 200 pixels each - 1200 inputs!

- With lots of images, we'd like to have fewer weights
- Typically, pixels that are closer to each other are more related while regions that are farther apart are less related

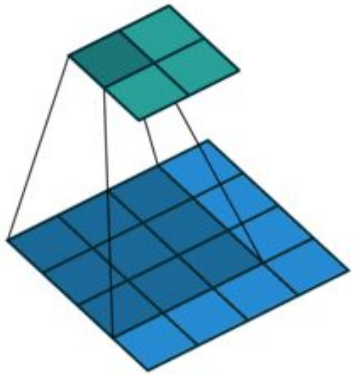


Convolutions! (crash course)

Taking advantage of the structured nature of image data, we can use weighted sums in small areas of images to process the data.



Convolutions! (crash course)



Taking advantage of the structured nature of image data, we can use weighted sums in small areas of images to process the data.

- We use a **kernel** filter that slides over the image and compute the sum of the element-wise product between the kernel and the image

Image

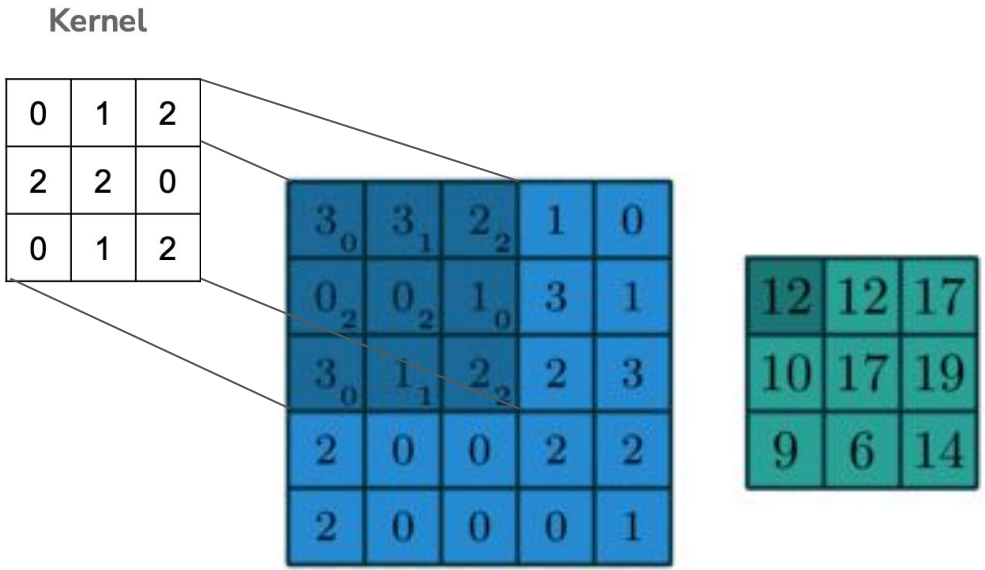
3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

Kernel

0	1	2
2	2	0
0	1	2

Convolutions - cont'd.

Convolutions - cont'd.

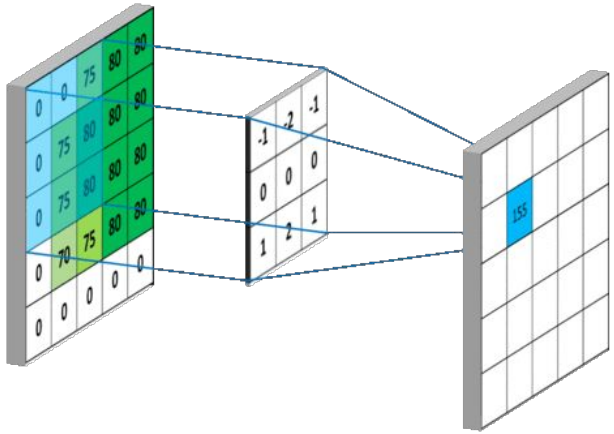


$$(3*0) + (3*1) + (2*2)$$
$$+ (0*2) + (0*2) + (1*0)$$
$$+ (3*0) + (1*1) + (2*2) = 12$$

Convolutions - cont’d.

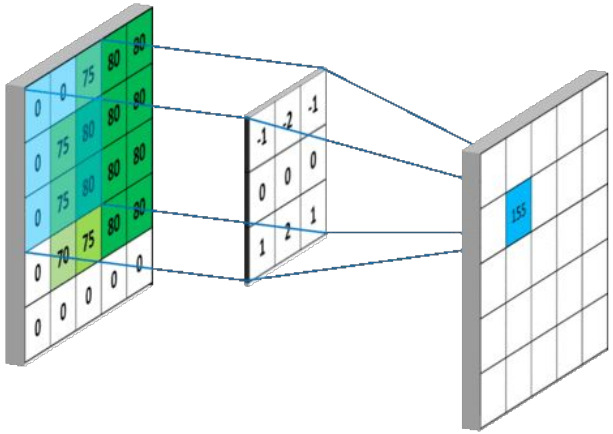
- What 3x3 kernel would result in the same image? (ignoring the edges)

- Can you think of any cool operations convolutions could do with an image?

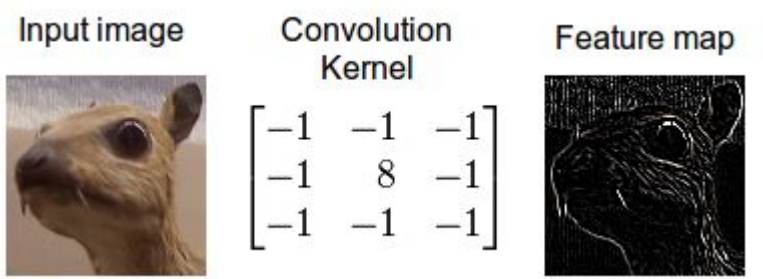


Convolutions - cont’d.

- What 3x3 kernel would result in the same image? (ignoring the edges)

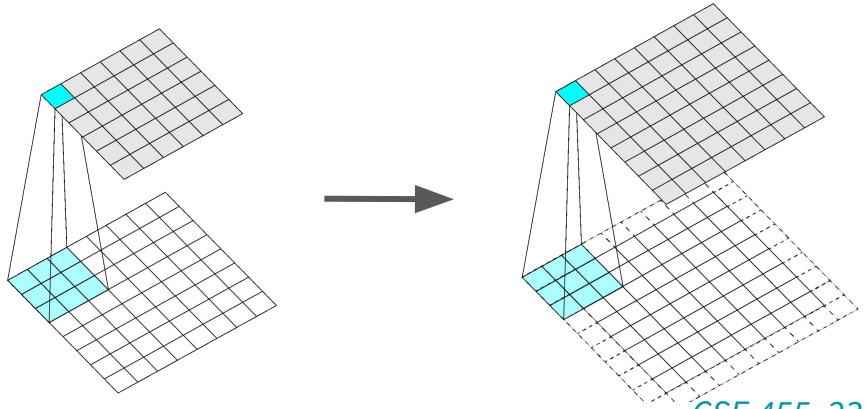


- Can you think of any cool operations convolutions could do with an image?



Padding & Stride

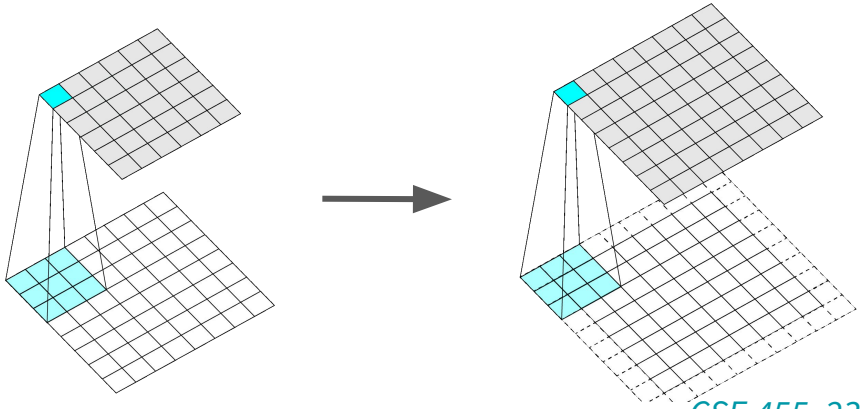
- Convolutions can have problems on the edges. So, we can add “padding” to the edges
  - Can vary the value and amount of padding to include



CSE 455, 23Sp

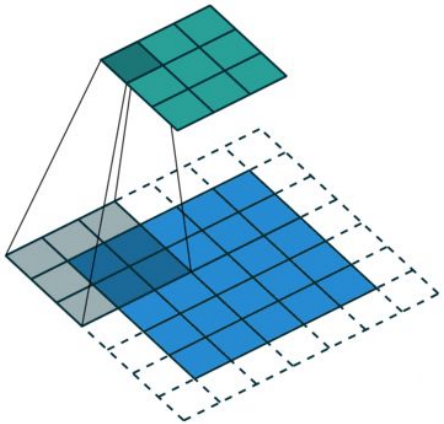
Padding & Stride

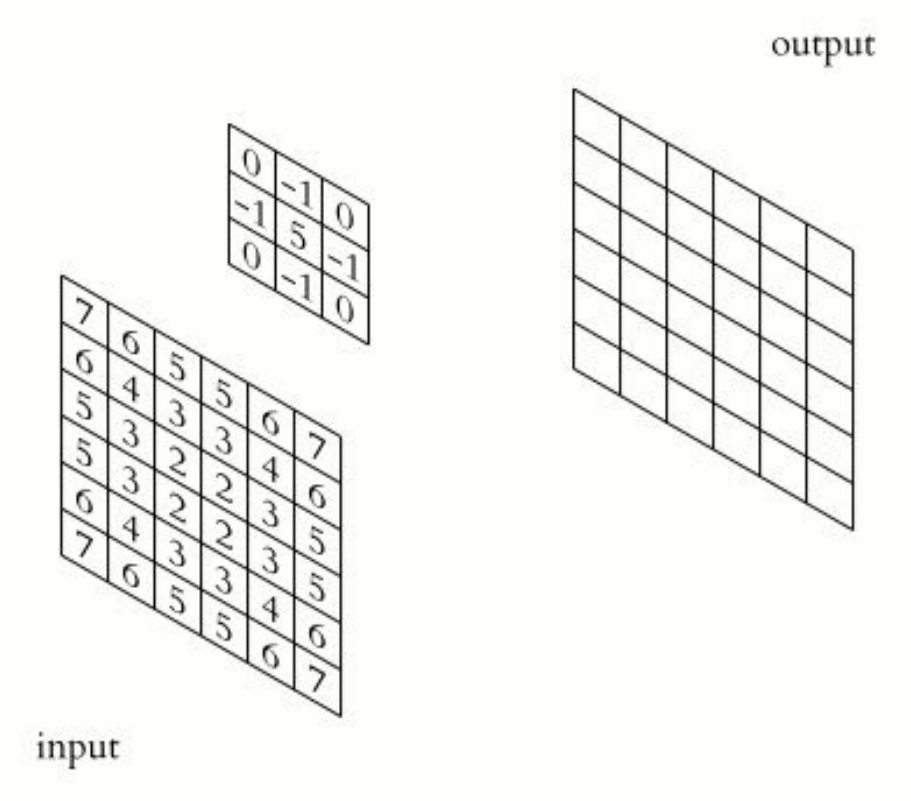
- Convolutions can have problems on the edges. So, we can add "padding" to the edges
  - Can vary the value and amount of padding to include



CSE 455, 23Sp

- Can also set a "stride", which determines how far to "jump" values. We've looked at convolutions with stride 1 so far; how would a larger stride affect your convolution output?





This video shows a convolutional pass being done. The kernel they use here is a “sharpening kernel”

Shape of a convolutional layer / maxpooling output: For a $n \times n$ input, $f \times f$ filter, padding p and stride s, the output size is $o \times o$ where:

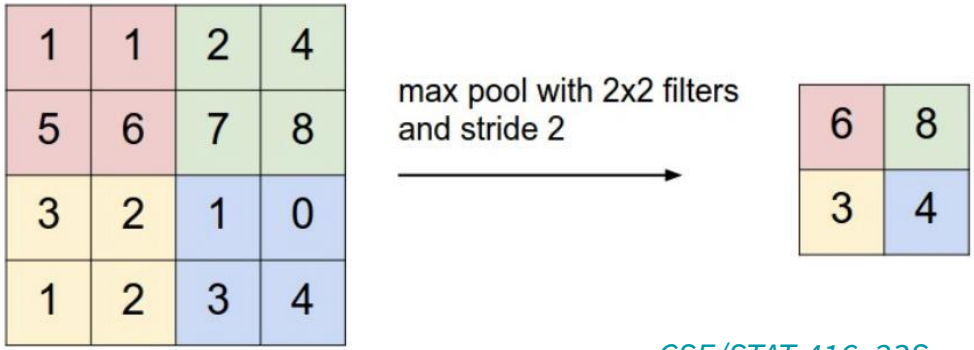
$$o = \frac{n - f + 2p}{s} + 1$$

An equation worth memorizing…

Pooling

Similar to convolutions, **pooling** is an operation that is commonly used to **downsample** images.

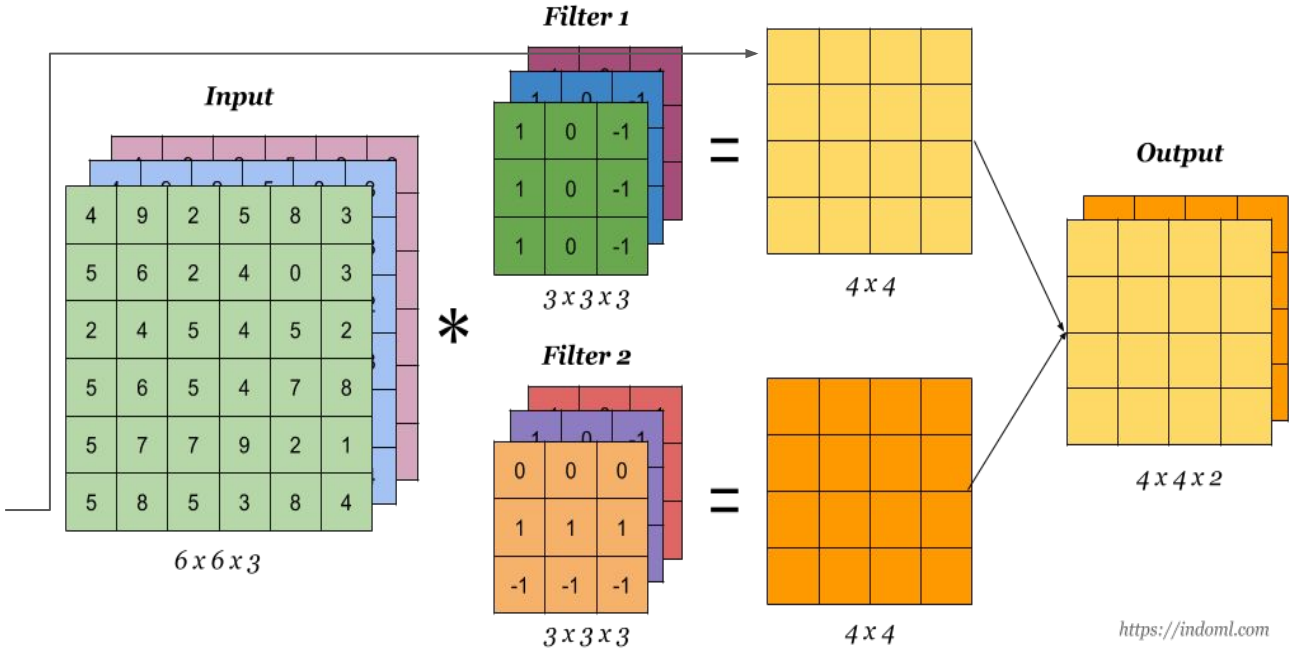
A pooling layer will iterate over an image like a convolution, pooling pixels in each region. You can average pixels, take the minimum value, take the median, etc. But typically, we will use **max pooling**, where we take the maximum value at each region.



What each filter looks like

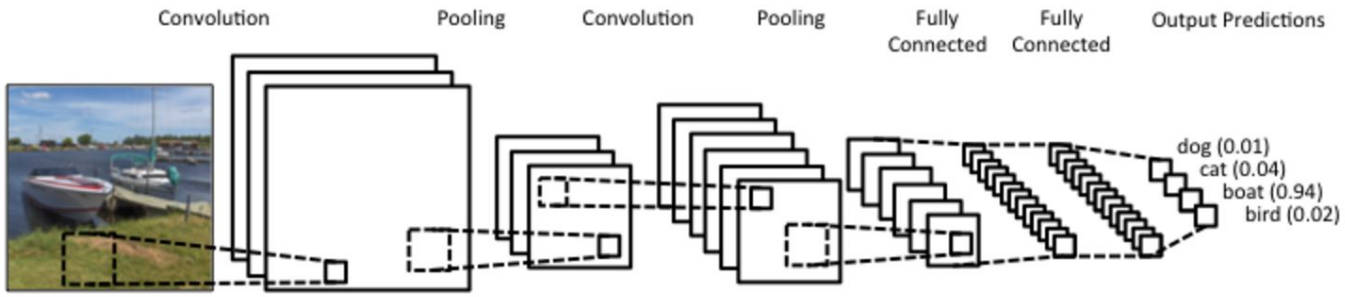
Each filter learns some feature of the input.

For this example, input is a 6x6 image with RGB channels. The filter will have the size of third dimension equal to the input channel (3). Output is just the elementwise dot-product.



Convolutional Neural Networks

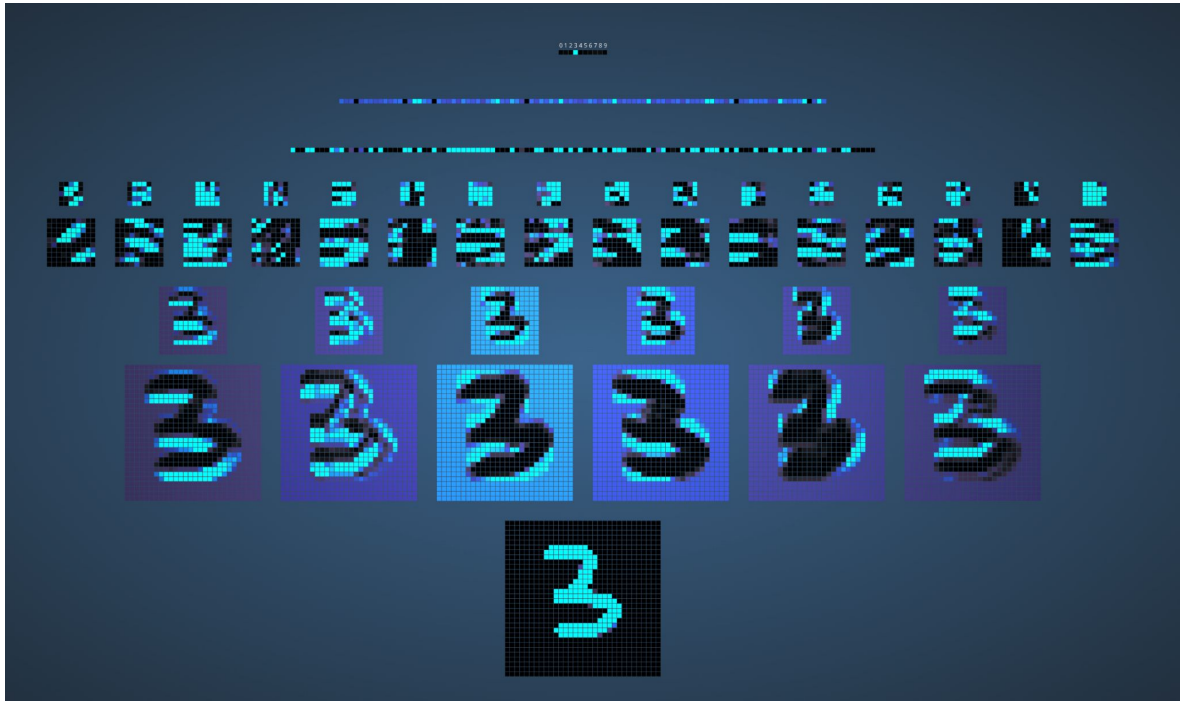
- Lots of different architectures possible for CNNs! In general, they often take the form:



  - The basic building blocks are convolutional layers, pooling layers, and fully connected layers. Fully connected layers are often used as a last layer to map your image features to predictions.

Filters visualized

You can clearly see the feature mapping here as a result of the convolutional operation with the kernel filters.



[CNN visualization demo](#) by Adam Harley

Section Participation Question

What operation in neural networks uses filters to process images?

Convolution

Practice

Question 1

(a) Discuss the advantages of a convolutional layer compared to a fully connected one. **Solution:**

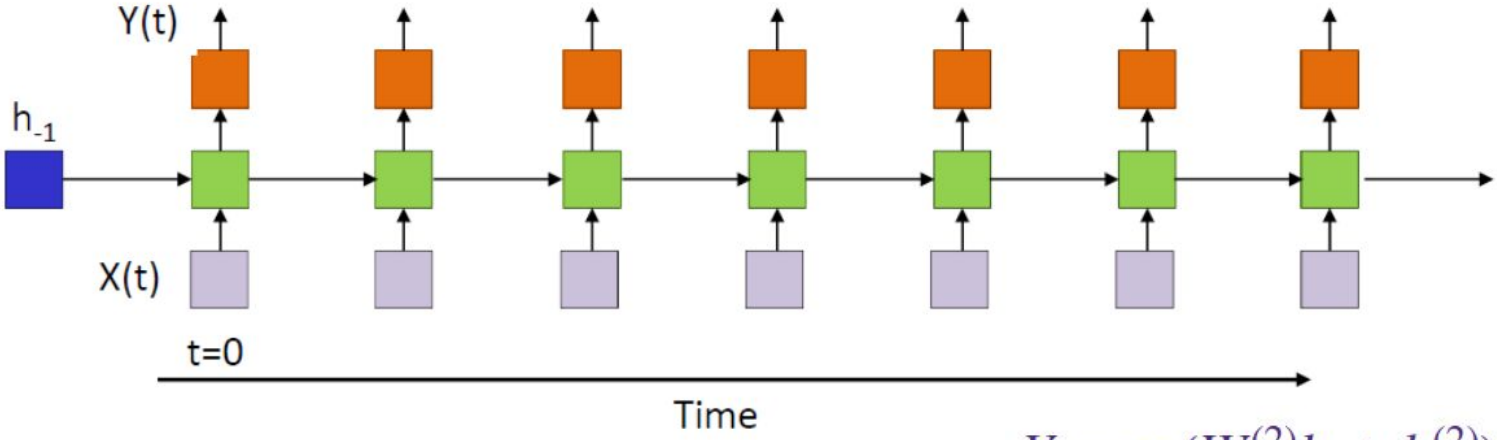
Convolutional layers are more flexible than fully connected ones since not all input neurons affect all output neurons. In addition, the number of weights per layer is smaller than that of linear layers, which would ease computation with high-dimensional data.

(b) Discuss the advantages of maxpooling in CNN. **Solution:**

Pooling layers are used to downsample feature maps, which make processing more efficient by reducing the number of parameters to learn.

RNNs / LSTMs

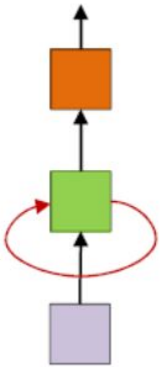
Recurrent Neural Network



- h_t: hidden state
- X_t: input
- Y_t: output
- $Y_t, h_t = f(h_{t-1}, X_t; \theta)$
- h_{-1}: initial state

$$Y_t = \sigma_2(W^{(2)}h_t + b^{(2)})$$

$$h_t = \sigma_1(W^{(1)}X_t + W^{(11)}h_{t-1} + b^{(1)})$$



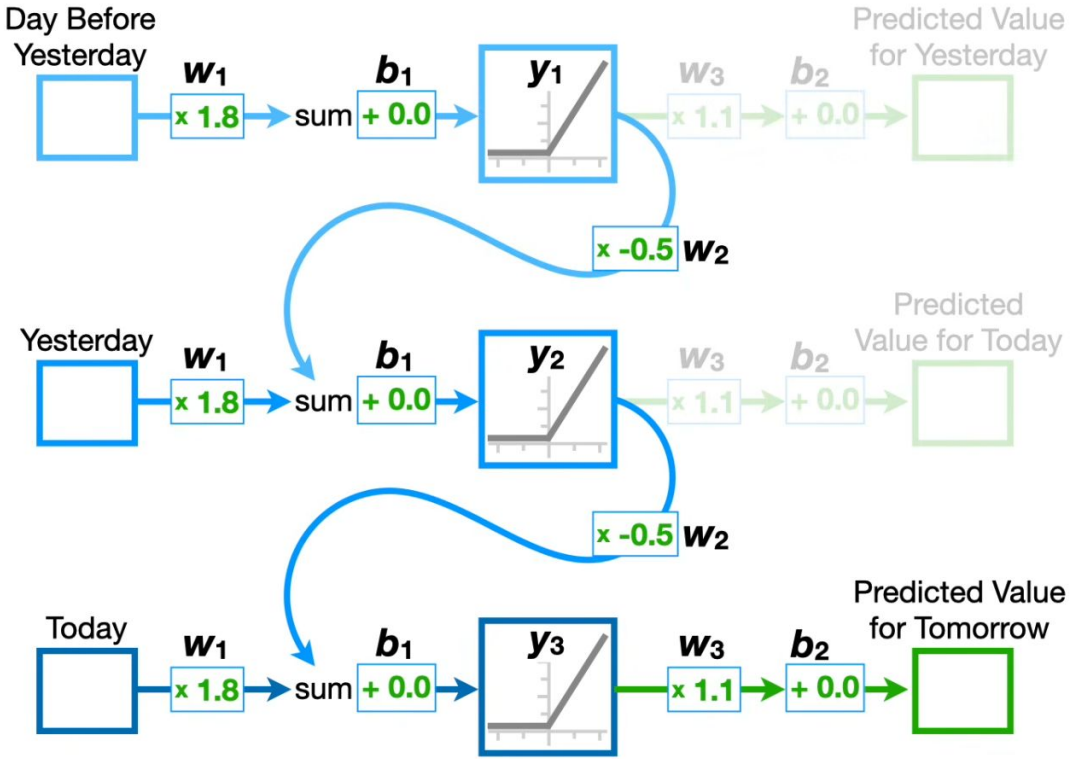
Feedback loop where also have t_{i-1}

RNNs

Similar to repeatedly applying a fully connected NN

aka

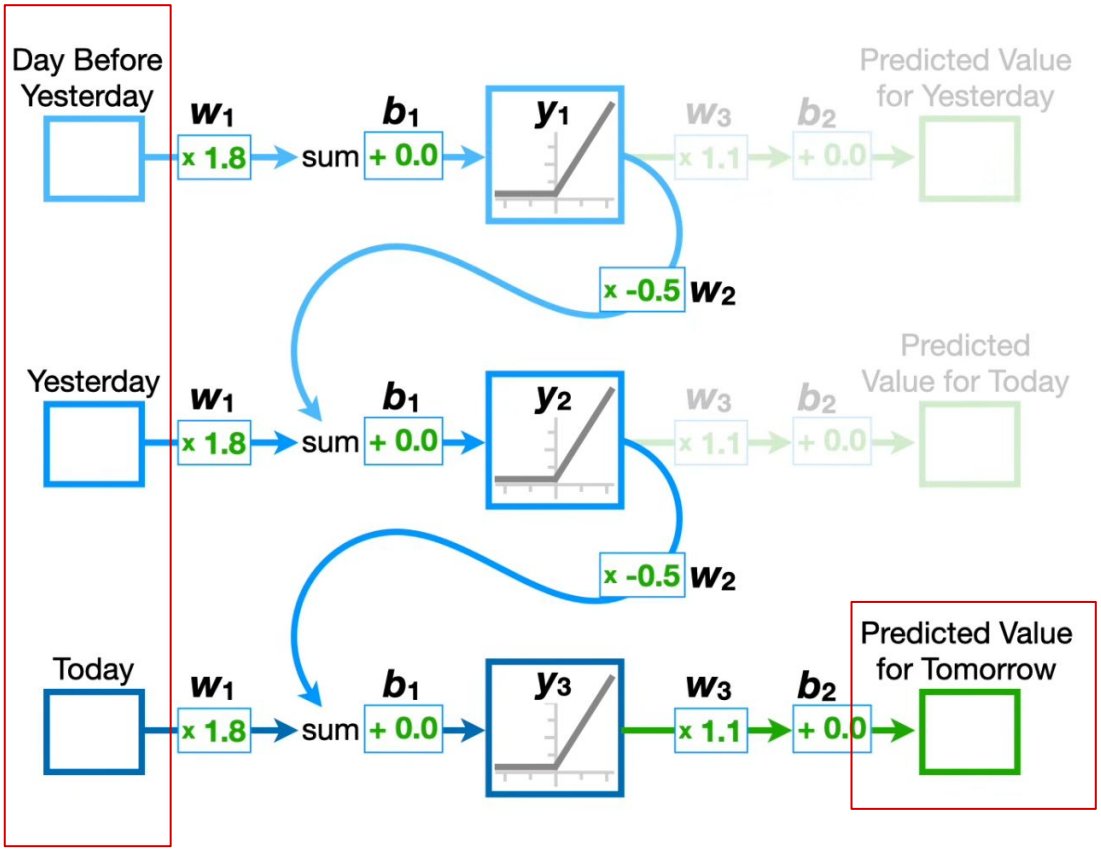
“Unrolling” the inputs



RNNs

Here we have 3 inputs to model stock prices

“Unrolling” the network we have three inputs that helps us predict tomorrow’s stock price

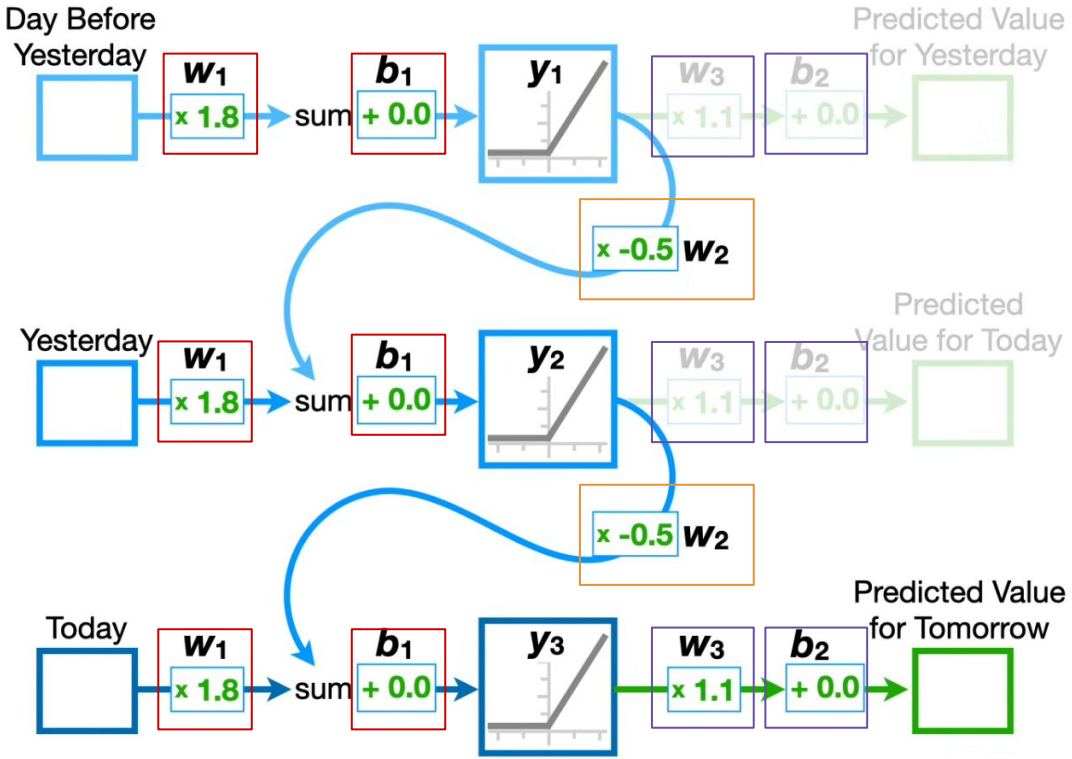


RNNs

3 inputs but like training one network

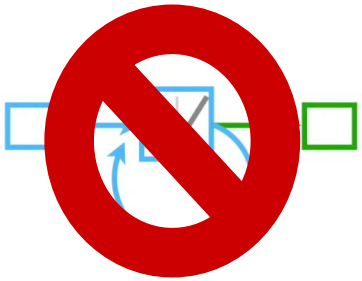
Weights and biases are shared

Easily leads to exploding/vanishing gradient (i.e. what if $w_2 < 1$? $w_2 > 1$?)

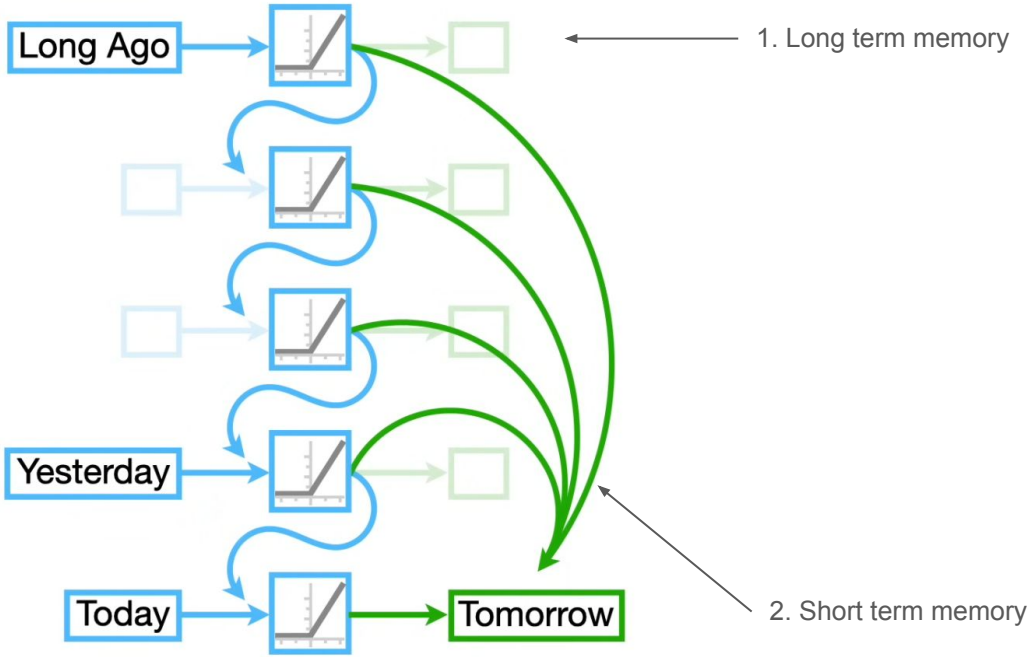


LSTMs

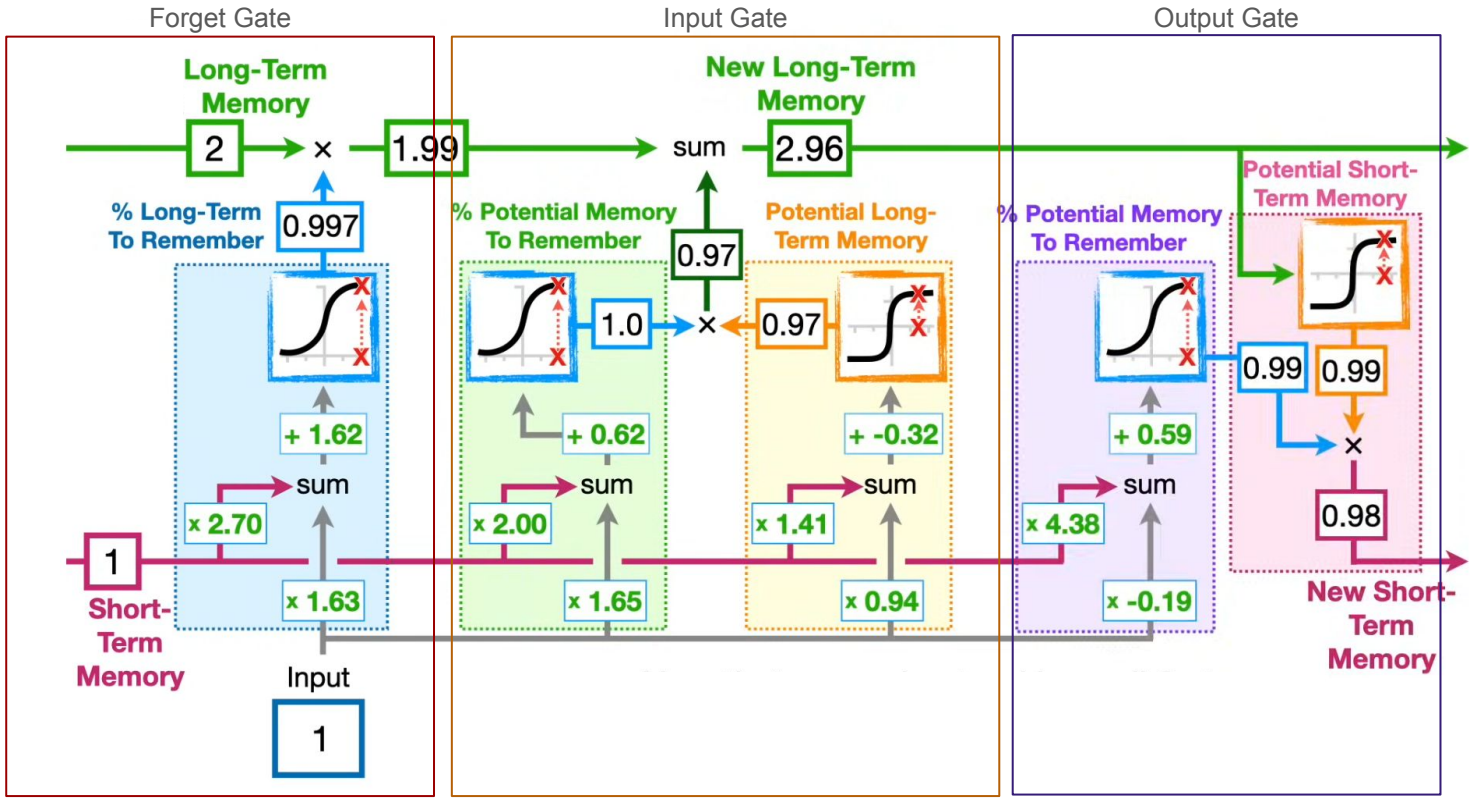
We can easily solve the exploding/vanishing gradient problems in RNNs with LSTMs



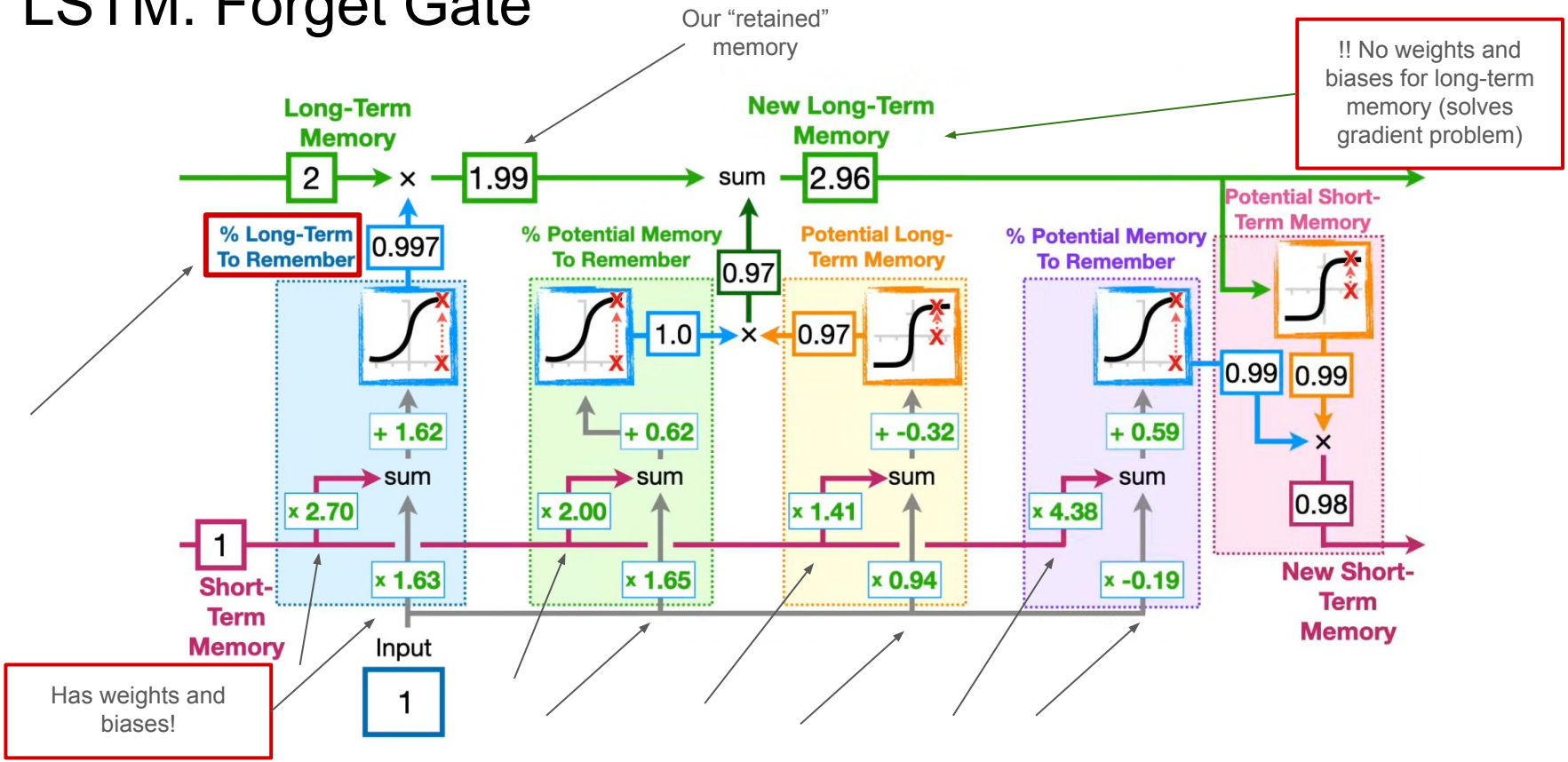
RNN Feedback Loop



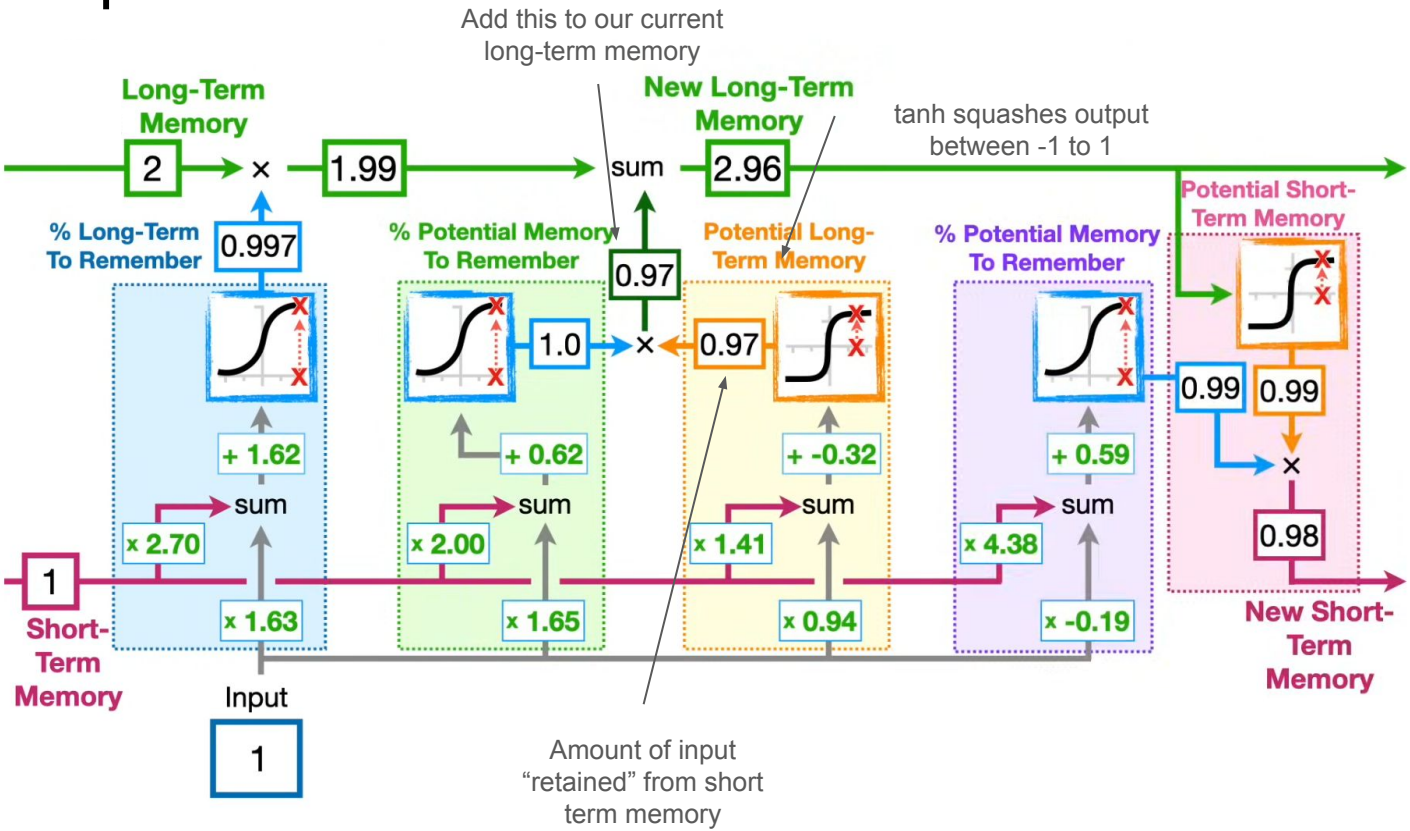
LSTMs


Forget Gate
Input Gate
Output Gate
Long-Term Memory
2
×
1.99
New Long-Term Memory
sum
2.96
% Long-Term To Remember
0.997
% Potential Memory To Remember
0.97
Potential Long-Term Memory
% Potential Memory To Remember
Potential Short-Term Memory
1.0
0.97
0.99
0.99
+ 1.62
+ 0.62
+ -0.32
+ 0.59
sum
sum
sum
sum
x 2.70
x 2.00
x 1.41
x 4.38
0.98
1
x 1.63
x 1.65
x 0.94
x -0.19
Short-Term Memory
New Short-Term Memory
Input
1

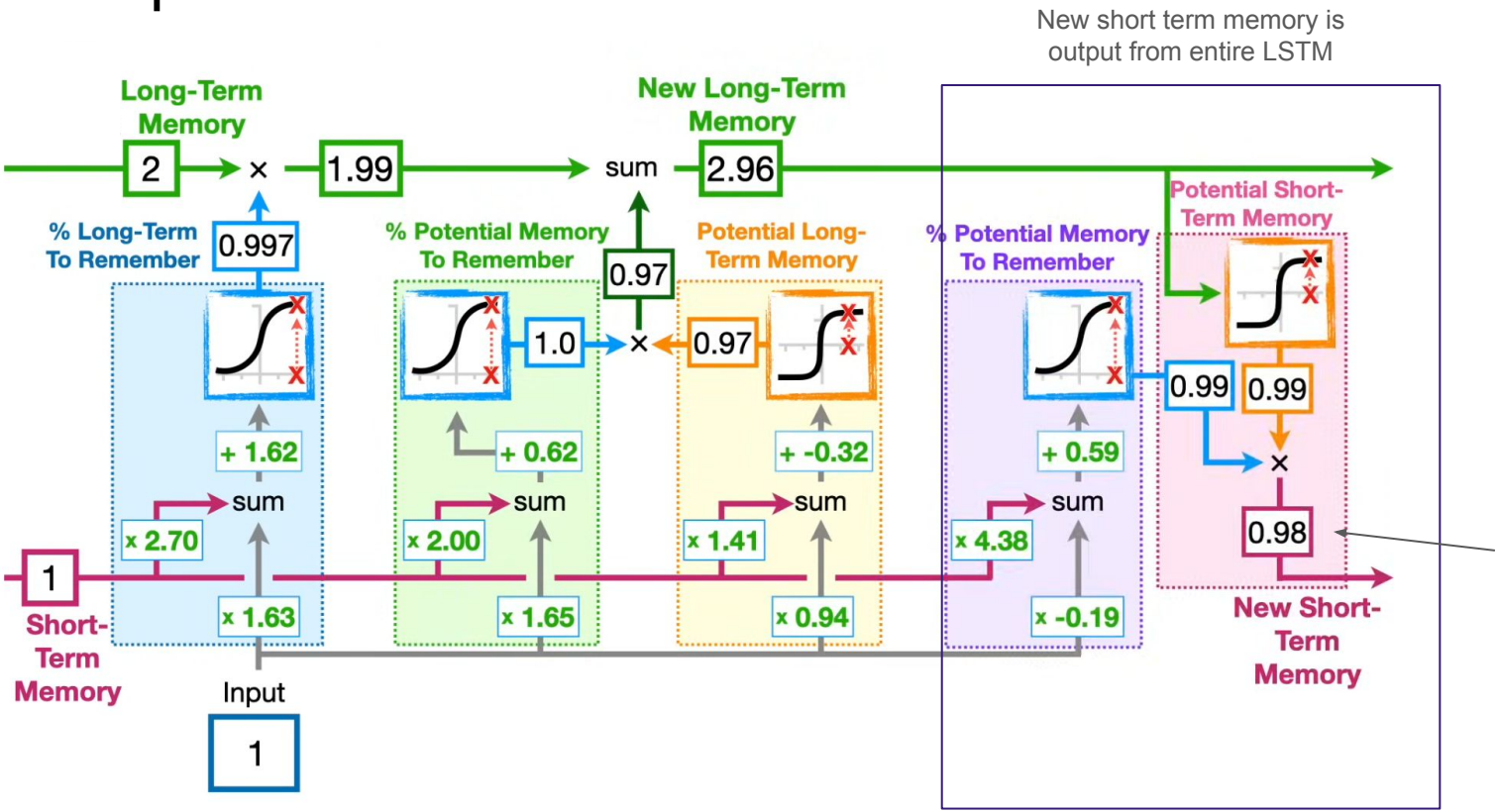
LSTM: Forget Gate



LSTM: Input Gate



LSTM: Output Gate



Practice

Question 3a

Suppose we have a toy scalar RNN with the recurrence

$$h_t = \text{ReLU}(wh_{t-1} + ux_t)$$

where $h_t, x_t, w, u \in \mathbb{R}$ are weights shared across all timesteps. Assume all pre-activation values are positive, meaning $\text{ReLU}' = 1$ throughout the network.

(a) Compute $\frac{\partial h_3}{\partial h_1}$ treating x_t as a constant, with $w = 0.3$ and with $w = 3$.

Solution:

Note that for each layer $h_t = \text{ReLU}(wh_{t-1}+ux_t)$, we have $\frac{\partial h_t}{\partial h_{t-1}} = \text{ReLU'}(wh_{t-1}+ux_t)\cdot w = 1\cdot w = w$.
We can compute $\frac{\partial h_3}{\partial h_1}$ using chain rule:

$$\frac{\partial h_3}{\partial h_1} = \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} = w \cdot w = w^2 = (0.3)^2 = 0.09$$

$$\frac{\partial h_3}{\partial h_1} = \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} = w \cdot w = w^2 = (3)^2 = 9$$

Question 3b

(b) Compute $\frac{\partial h_{20}}{\partial h_1}$, with $w = 0.3$ and $w = 3$, and state how the magnitude of w impacts the model's ability to learn long-range dependencies.

Solution:

$$\frac{\partial h_{20}}{\partial h_1} = \frac{\partial h_{20}}{\partial h_{19}} \cdot \frac{\partial h_{19}}{\partial h_{18}} \cdots \frac{\partial h_2}{\partial h_1} = w^{19} = (0.3)^{19} \approx 0$$

$$\frac{\partial h_{20}}{\partial h_1} = \frac{\partial h_{20}}{\partial h_{19}} \cdot \frac{\partial h_{19}}{\partial h_{18}} \cdots \frac{\partial h_2}{\partial h_1} = w^{19} = (3)^{19} \approx 1 \times 10^9$$

The main takeaway here is that as we propagated back through more layers, we face the issue of vanishing gradients or exploding gradients. This is the inherent problem with RNNs, and why LSTMs are far more popular.

Question 3c

(c) To address the issue of vanishing gradients, the LSTM architecture uses a cell update

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t$$

where $f_t \in [0, 1]$ is the learned forget gate. Suppose the network learns $f_t = 0.99$ for all t. Compute $\frac{\partial c_{20}}{\partial c_1}$, treating $i_t \cdot \tilde{c}_t$ as a constant w.r.t. c_{t-1}.

Solution:

$$\frac{\partial c_t}{\partial c_{t-1}} = f_t \rightarrow \frac{\partial c_{20}}{\partial c_1} = \prod_{t=2}^{20} f_t = (0.99)^{19} \approx 0.83$$

Note: This architecture creates a "gradient highway". Because f_t is learned, the network can choose to maintain a gradient close to 1 (by learning $f_t \approx 1$), which prevents the exponential decay seen in the RNN example. For exploding gradients, we need to resort to gradient clipping (which can be done for RNNs as well).

Attention & Transformers

Attention

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

$$e_{ij} = a(s_{i-1}, h_j)$$

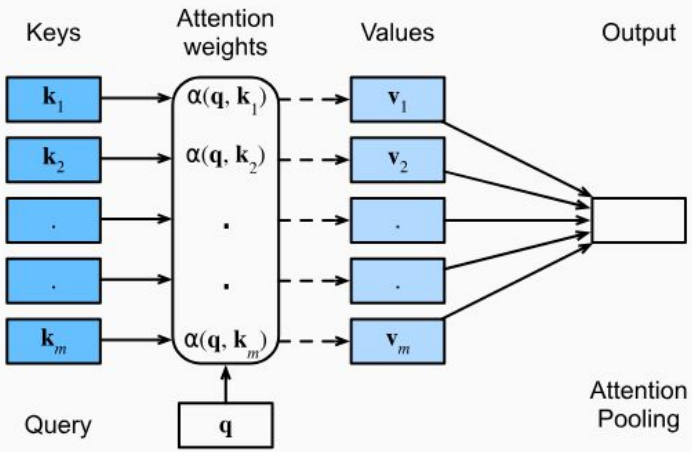


Fig. 11.1.1 The attention mechanism computes a linear combination over values $\mathbf{v}_i$ via attention pooling, where weights are derived according to the compatibility between a query $\mathbf{q}$ and keys $\mathbf{k}_i$.

- Used in transformers
- A way of remembering initial inputs
- Computing the similarity between the query and all input keys

Transformers

I like watching tabby cats sunbathe.

Encodes the position a word is in the sentence.

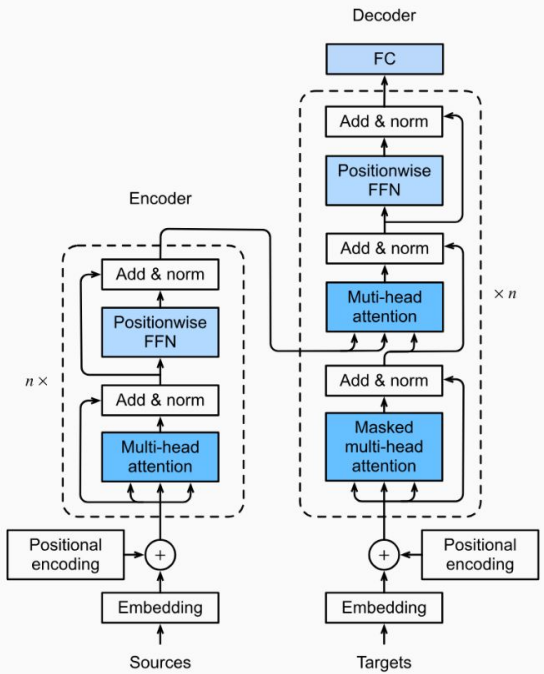
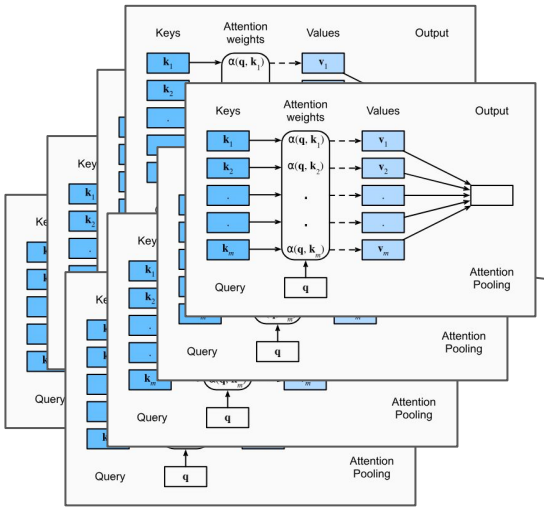


Fig. 11.7.1 The Transformer architecture.

Transformers

I like watching tabby cats sunbathe.



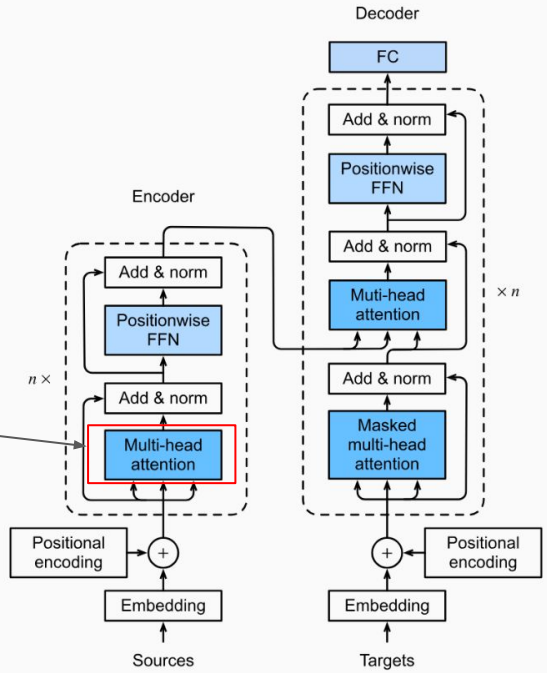


Fig. 11.7.1 The Transformer architecture.

Transformers

I like watching tabby cats sunbathe.

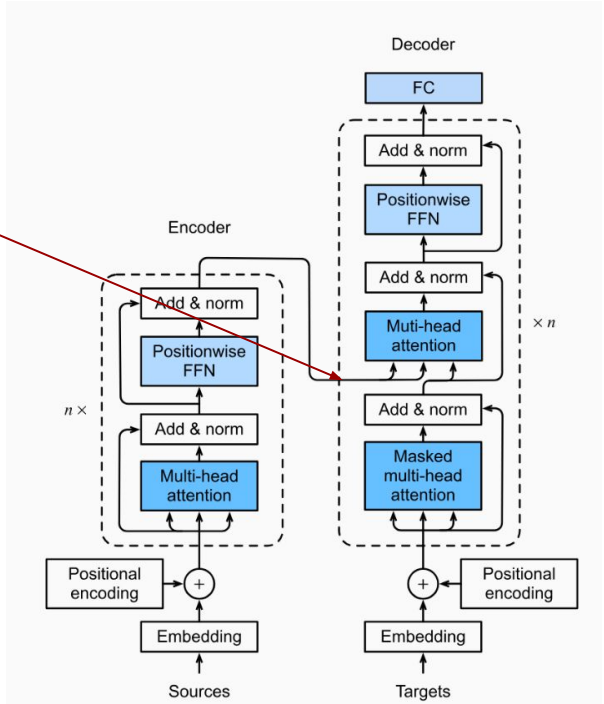


Fig. 11.7.1 The Transformer architecture.

Transformers

I like watching tabby cats sunbathe.

Encodes words in parallel (not sequentially!!)
We can reuse same weights & biases in attention

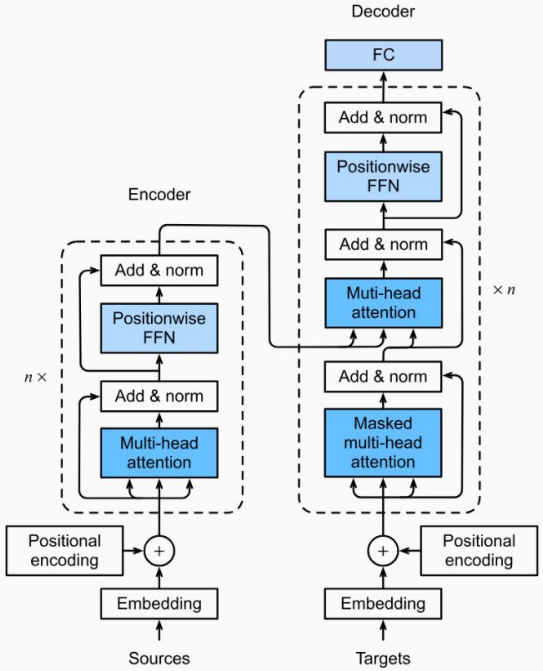


Fig. 11.7.1 The Transformer architecture.

Practice

Question 4a

In this example, we will do a full forward pass through the self-attention mechanism, which is often used in Transformer architectures, such as ChatGPT. Our input sequence will be ”The red panda is cute.”, where our data matrix is

$$X = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Our weights will be given as

$$W_Q = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad W_K = \begin{bmatrix} \sqrt{3}\ln(80) & 0 & 0 \\ 0 & 0 & 0 \\ \sqrt{3}\ln(17) & 0 & 0 \end{bmatrix}, \quad W_V = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

As you can see, $d_{model} = 3$ here.

(a) Calculate the Query, Key, and Value matrices Q, K, V.

Question 4a

Solution:

$$Q = XW_Q = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$K = XW_K = \begin{bmatrix} 0 & 0 & 0 \\ \sqrt{3}\ln(80) & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \sqrt{3}\ln(17) & 0 & 0 \end{bmatrix}$$

$$V = XW_V = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Question 4b

(b) Compute the pre-softmax attention scores, i.e., $\frac{QK^\top}{\sqrt{d_{model}}}$.

Solution:

$$QK^\top = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & \sqrt{3}\ln(80) & 0 & 0 & \sqrt{3}\ln(17) \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \sqrt{3}\ln(80) & 0 & 0 & \sqrt{3}\ln(17) \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\frac{QK^\top}{\sqrt{d_{model}}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \ln(80) & 0 & 0 & \ln(17) \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Question 4c

(c) Compute the softmax to obtain our final attention scores.

Solution:

For all zero rows, the softmax is just a vector of all $\frac{1}{5}$s. This is because $\text{softmax}(\mathbf{0}_i) = \frac{e^0}{\sum_j e^0} = \frac{1}{5}$

For row 3, the denominator of the softmax will be $e^0+e^{\ln(80)}+e^0+e^0+e^{\ln(17)} = 1+80+1+1+17 = 100$. Therefore, the softmax of this row will be

$$\begin{bmatrix} e^0/100 & e^{\ln(80)}/100 & e^0/100 & e^0/100 & e^{\ln(17)}/100 \end{bmatrix} = \begin{bmatrix} 0.01 & 0.8 & 0.01 & 0.01 & 0.17 \end{bmatrix}$$

Thus, our final attention scores are

$$\begin{bmatrix} 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.01 & 0.80 & 0.01 & 0.01 & 0.17 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \end{bmatrix}$$

Question 4d

(d) Finally, compute $\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_{model}}}\right)V$.

Solution:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_{model}}}\right)V = \begin{bmatrix} 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.01 & 0.80 & 0.01 & 0.01 & 0.17 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.20 & 0.20 & 0.20 \\ 0.20 & 0.20 & 0.20 \\ 0.80 & 0.01 & 0.17 \\ 0.20 & 0.20 & 0.20 \\ 0.20 & 0.20 & 0.20 \end{bmatrix}$$

Question 4e

(e) What information does it seem like this attention head is transferring between word embeddings?

Solution:

The only word embedding with a meaningful update is ”panda”, which is a noun. When we look at the attention scores, it is attending to two words: ”red” and ”cute”. We can infer that this attention head is getting nouns to attend to the relevant adjectives, such that after this self-attention block, the embedding for ”panda” now also encodes that it is ”cute” and ”red”.