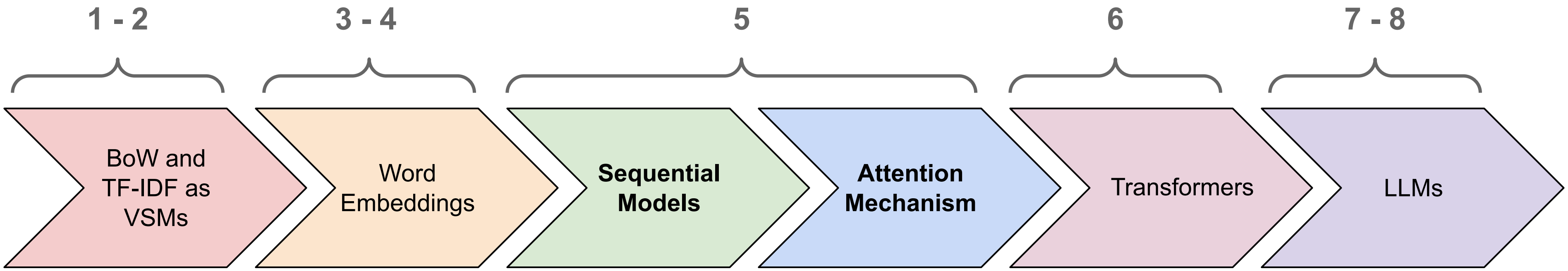
Code.Hub

The first Hub for Developers

Sequential Models

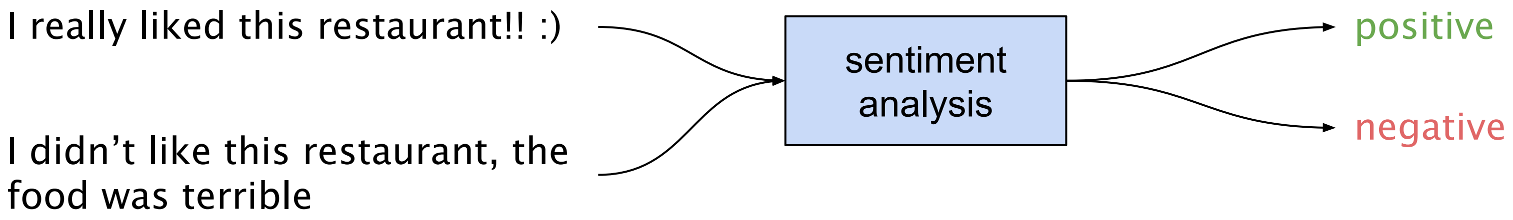
Thanos Tagaris

NLP timeline up till today…

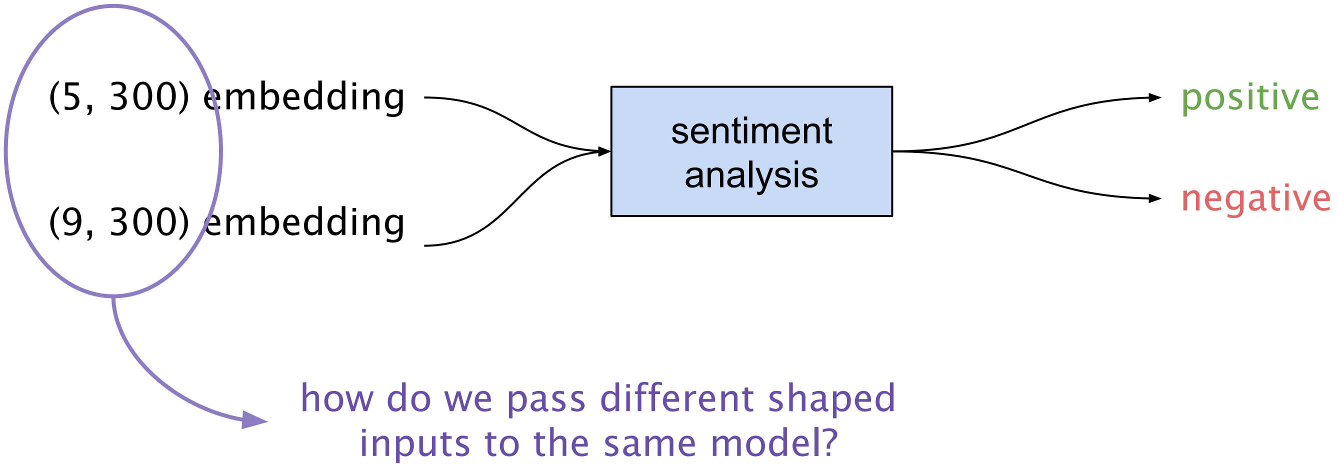


Trainable word embeddings gave rise to extended Neural Network usage for NLP tasks.

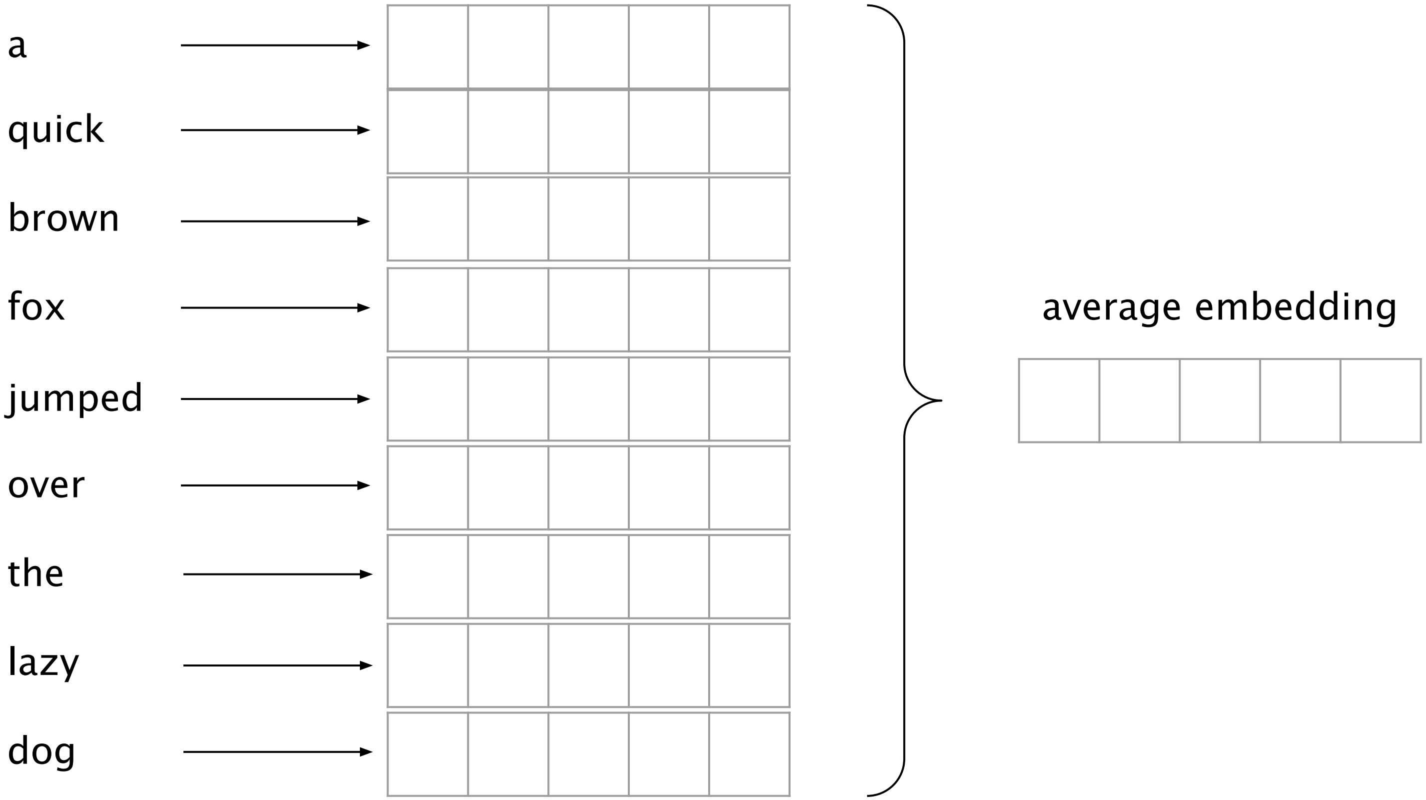
How do we use word embeddings for our downstream tasks?



How do we use word embeddings for our downstream tasks?



Average embedding



Average embedding

Pros:

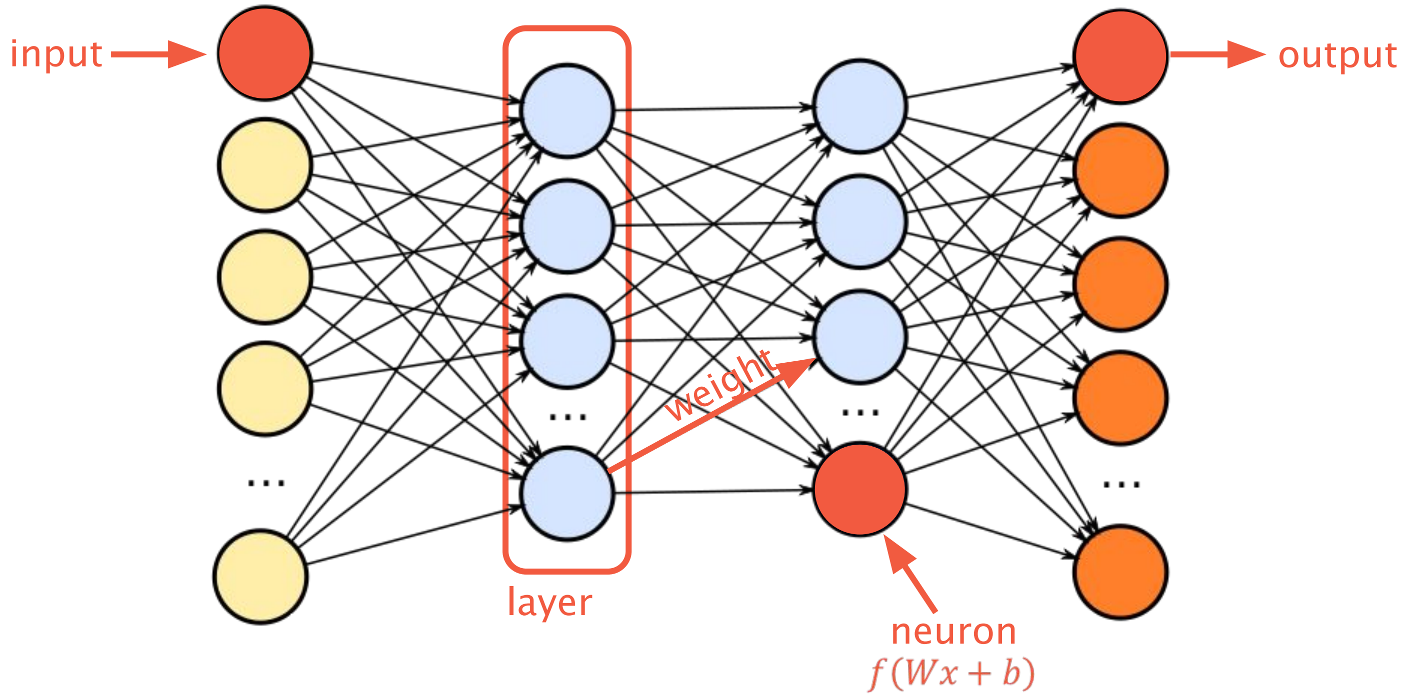
- always the same dim, can be used to train ML models
- cheap to compute

Cons:

- cannot capture word order
- degenerates with long sentences

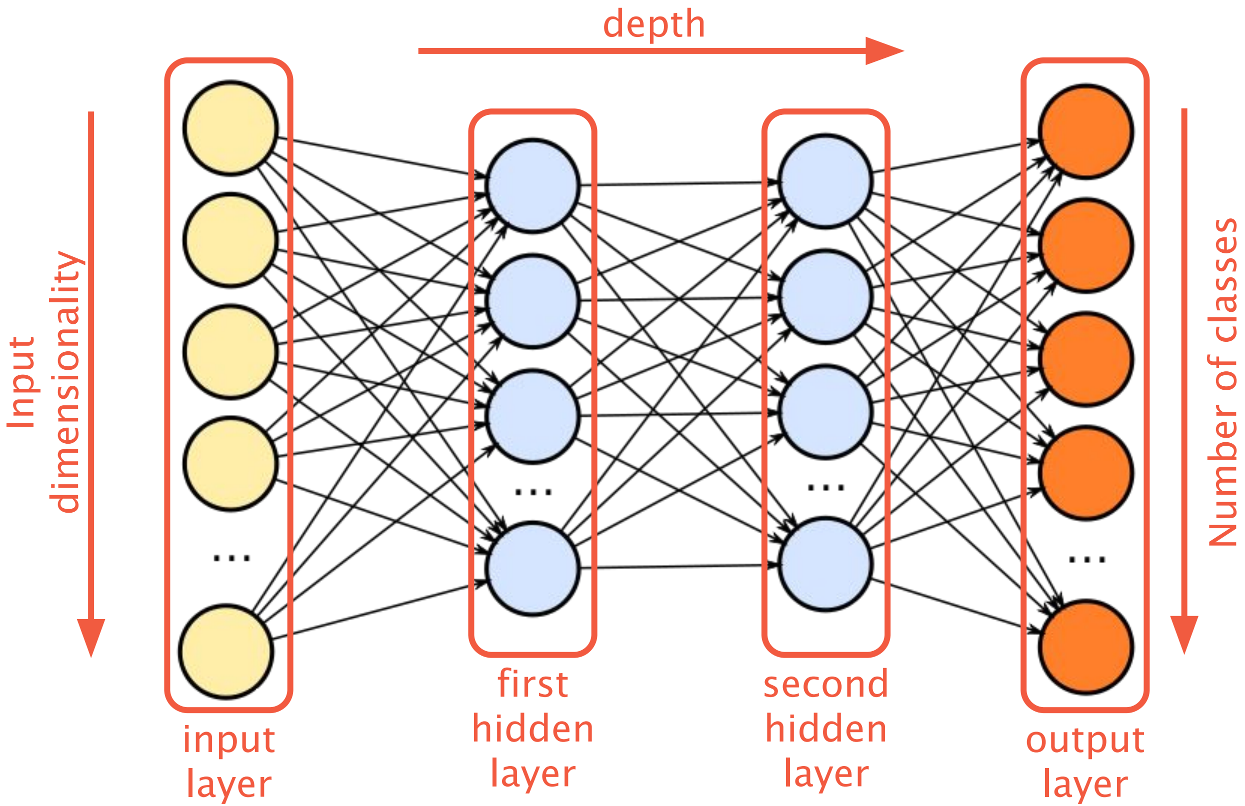
Is there a better way to encode a sentence?

Neural Networks – reminder



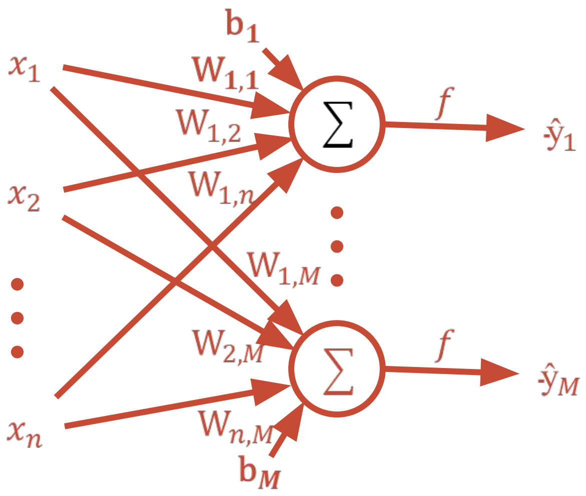
Code.Hub

Neural Networks – reminder



Code.Hub

Inside each layer…



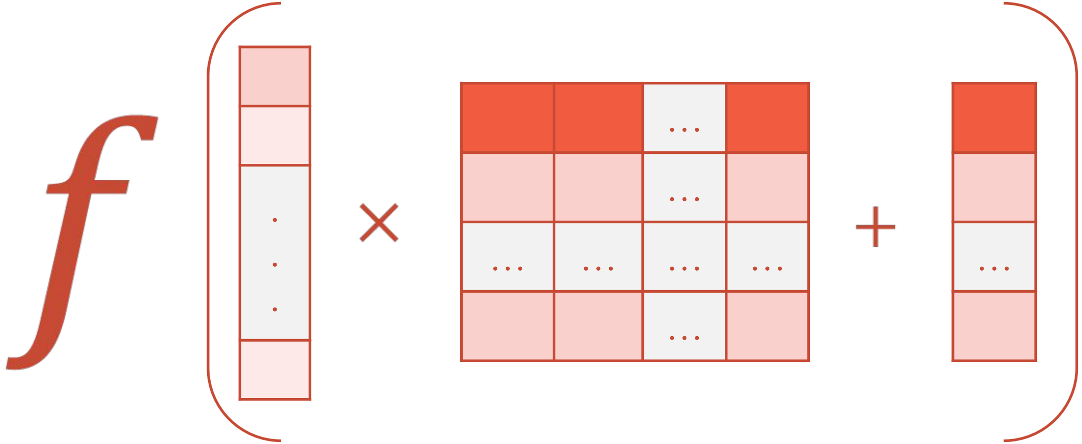
In matrix format

$$\hat{y} = f(XW + b)$$

If we have n input features and M neurons:

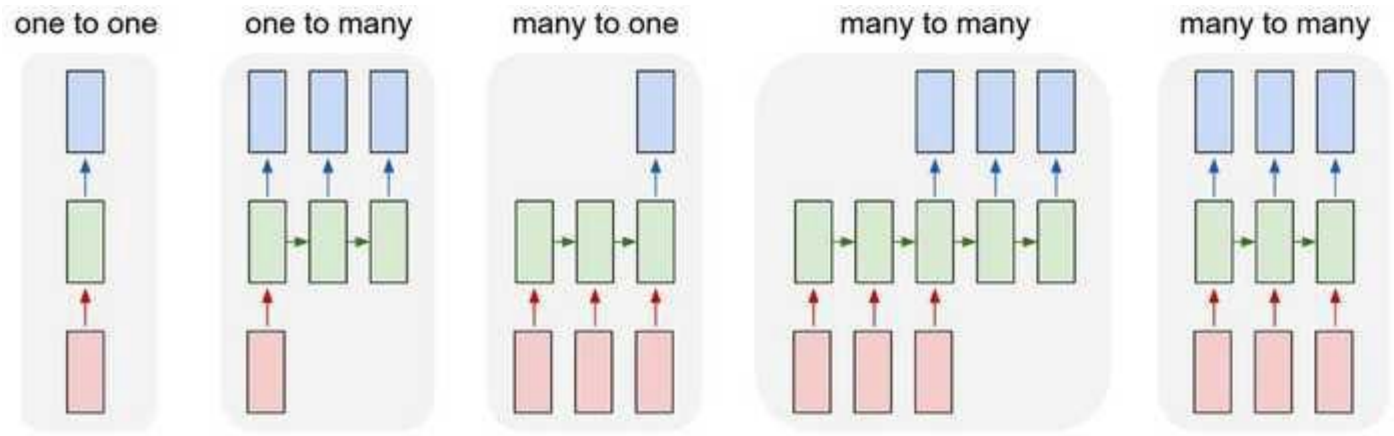
- Weight matrix has $M \times n$ values.
- Bias matrix has $M \times 1$ values.

Inside each layer…



$$\hat{y} = f(xW + b)$$

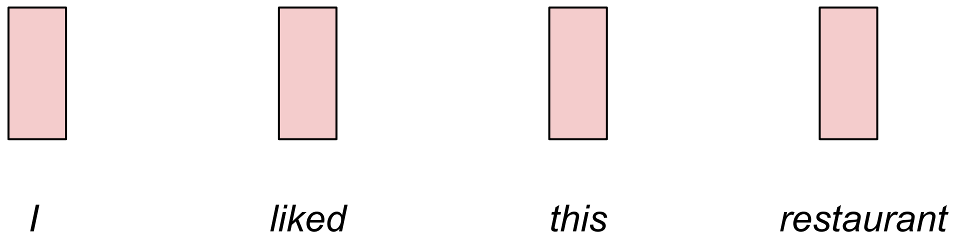
Recurrent Neural Networks



- input
- hidden
- output

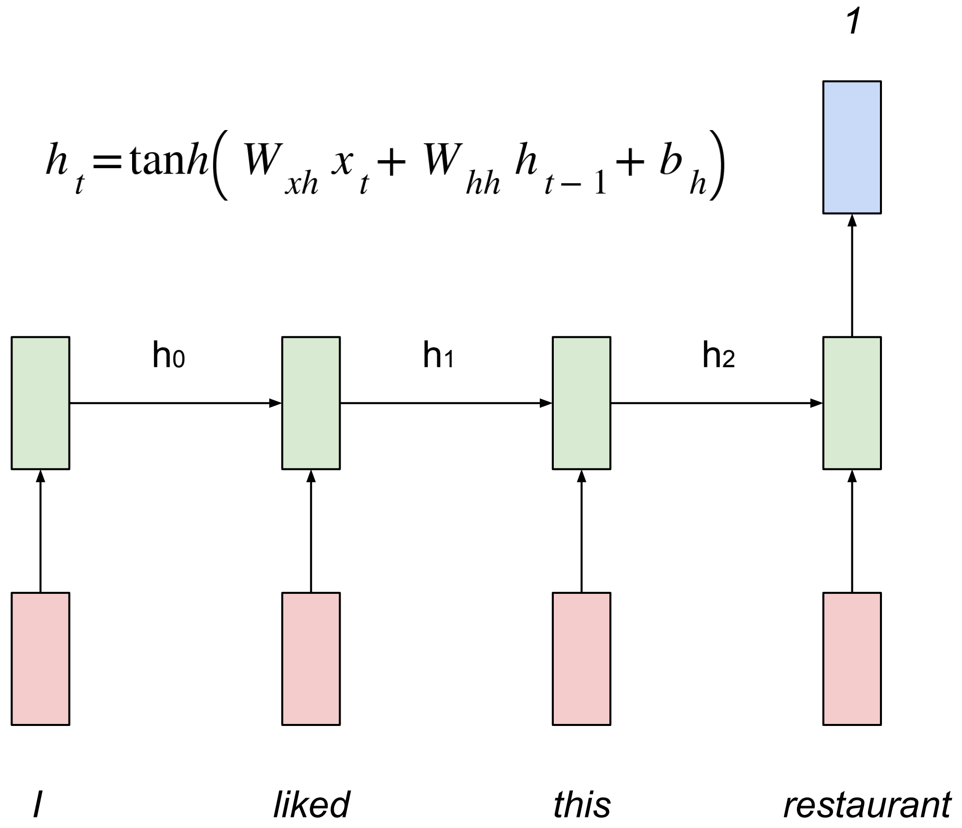
Recurrent Neural Networks

- Case study: sentiment analysis
- Predict if a sentence is positive or negative.
- Example *"I liked this restaurant"*
- *Network's input:*



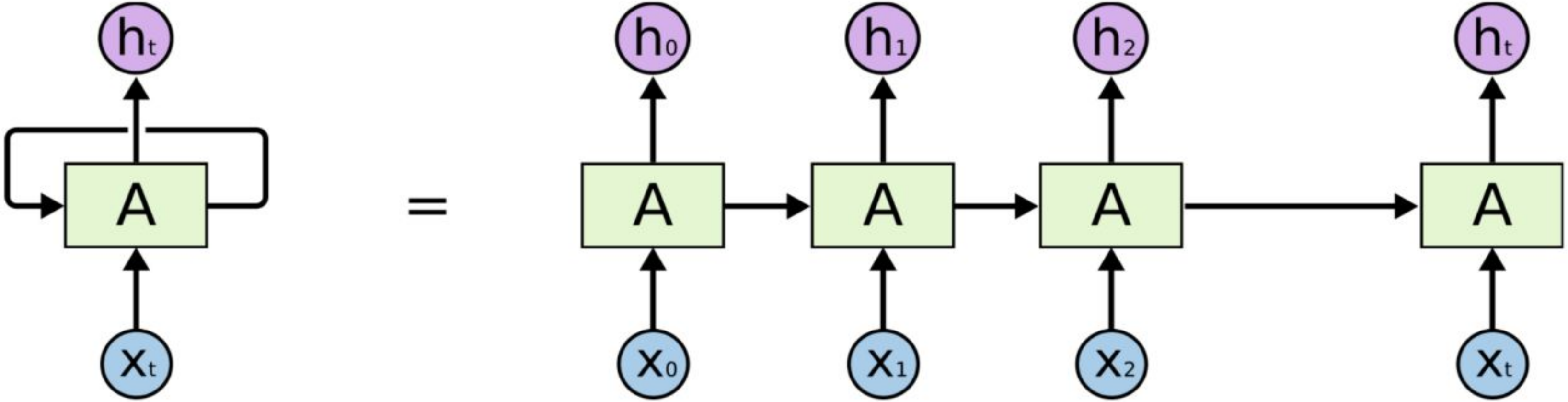
Recurrent Neural Networks

$$h_t = \tanh\left(W_{xh} x_t + W_{hh} h_{t-1} + b_h \right)$$



- Trained with Backpropagation Through Time
- Issues with vanishing/exploding gradients
- Partially solved with different "neuron" types, e.g. LSTM

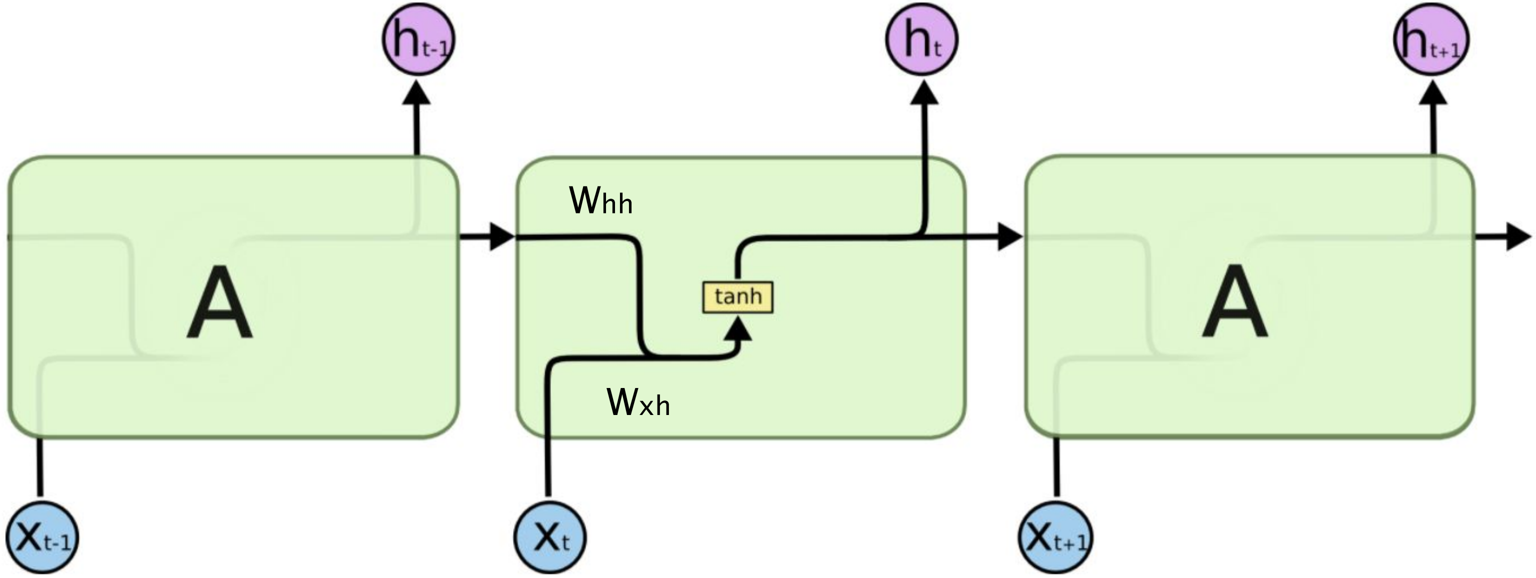
Recurrent Neural Networks



Must read: *“The unreasonable effectiveness of RNNs”* - Andrej Karpathy

Code.Hub

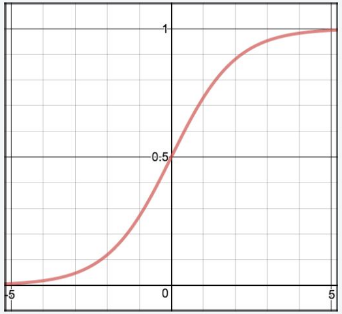
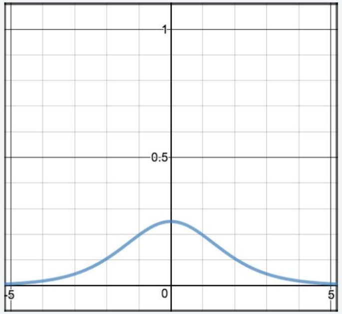
Recurrent Neural Networks



$$h_t = \tanh(W_{xh} x_t + W_{hh} h_{t-1} + b_h)$$

Vanishing/Exploding Gradients

- Historically, we couldn't model very long sequences due to numerical instability.
- Example: derivative of sigmoid

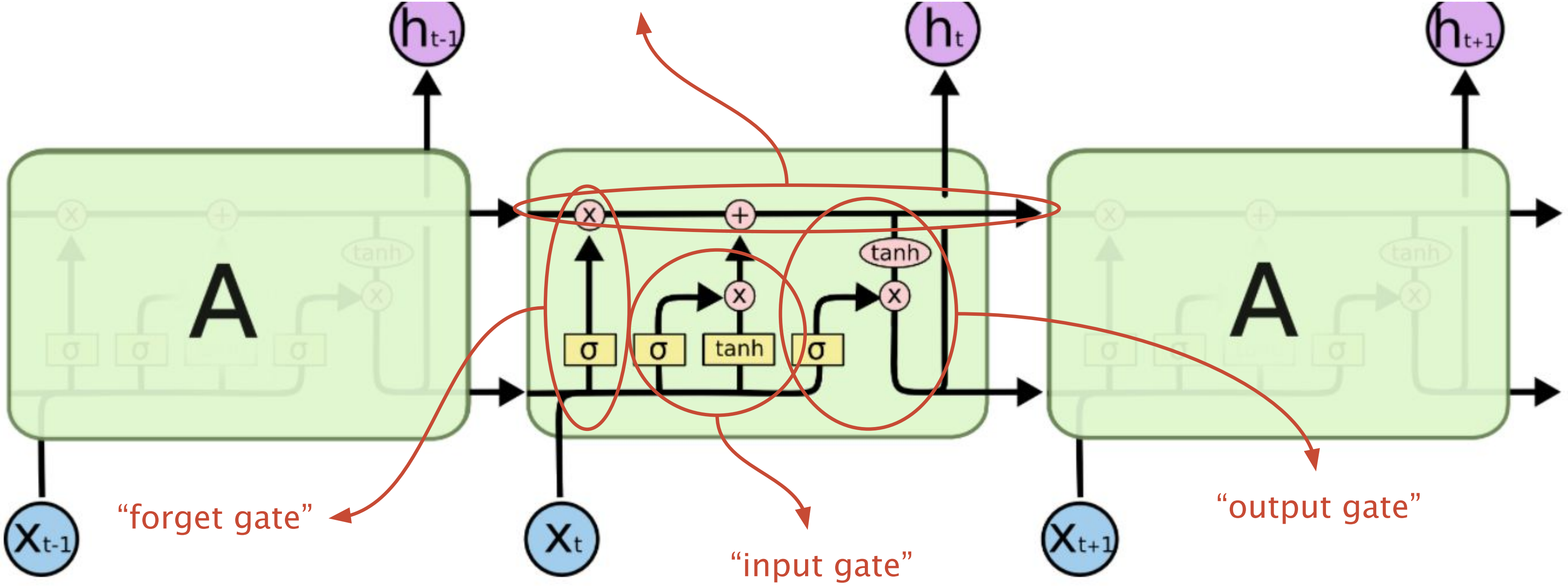
Function	Derivative
$S(z) = \frac{1}{1+e^{-z}}$	$S'(z) = S(z) \cdot (1 - S(z))$
	

```python
def sigmoid(z):
  return 1.0 / (1 + np.exp(-z))
```

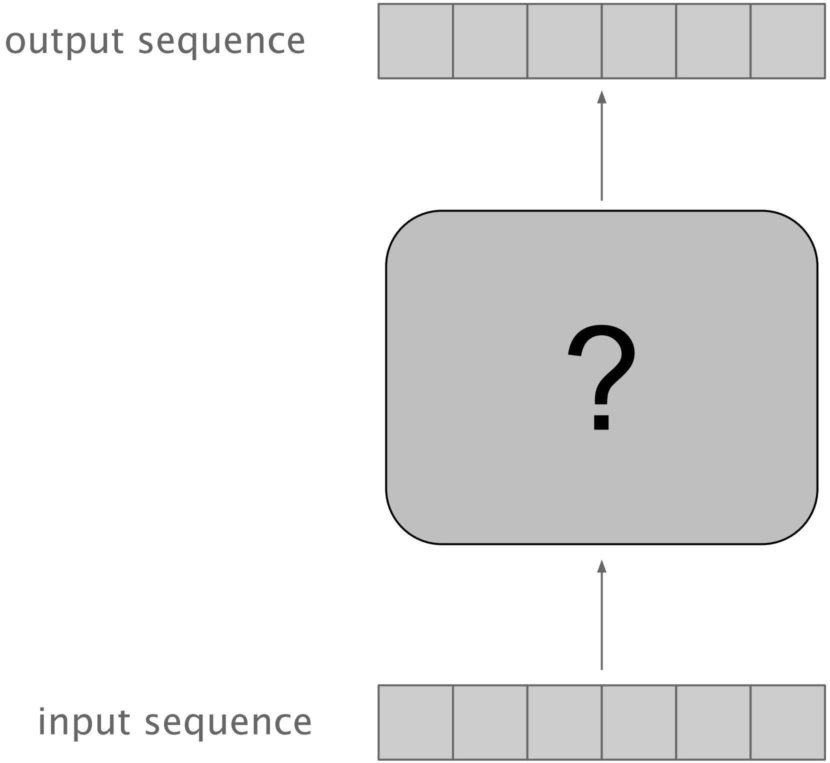
```python
def sigmoid_prime(z):
  return sigmoid(z) * (1-sigmoid(z))
```

Long Short-Term Memory (LSTM)

allows for information to pass straight through



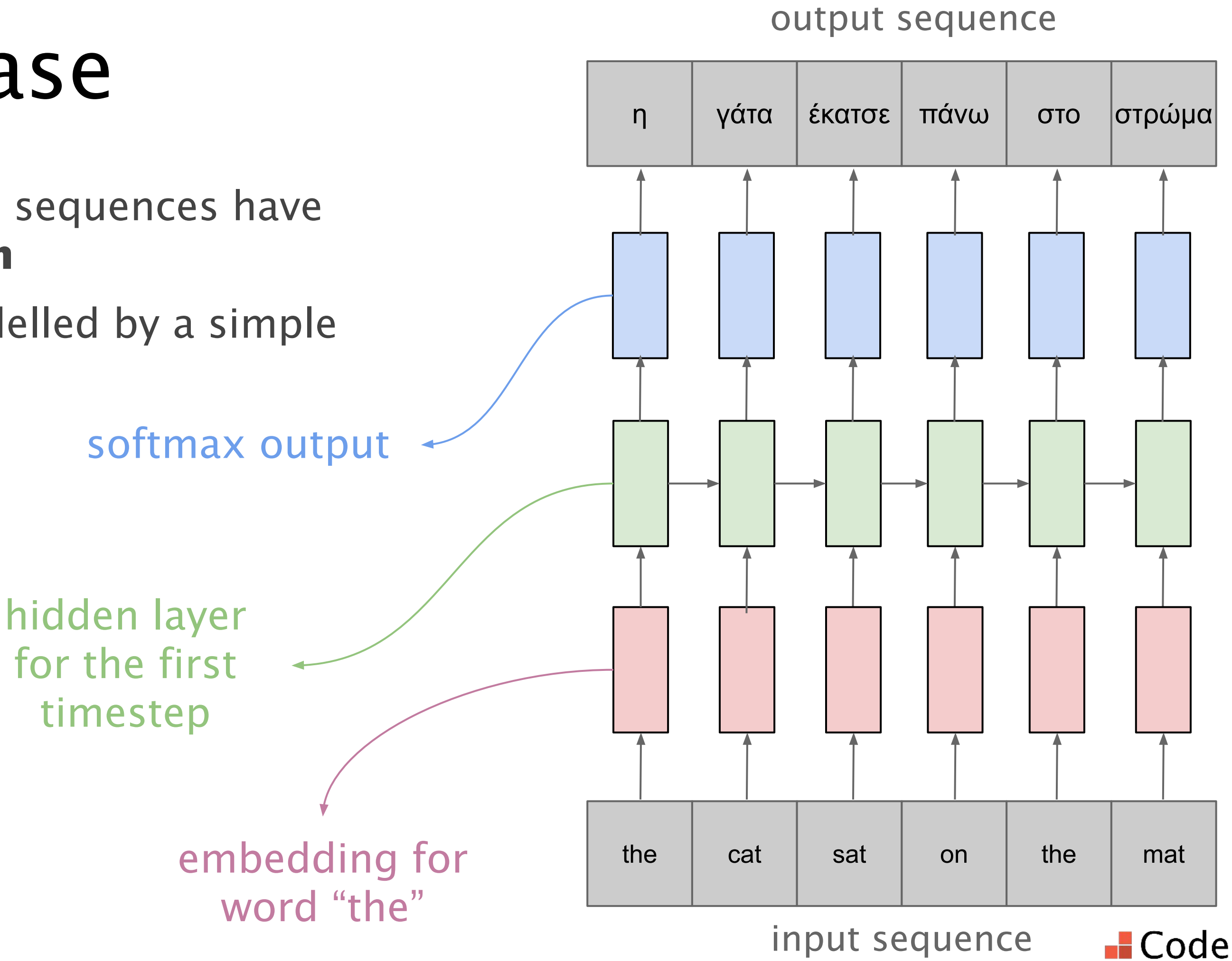
Sequence-to-sequence problems



- Input is a sequence, target is also a sequence
- Most language problems are like this
- E.g.
  - machine translation
  - question answering
  - summarization
- How do we model such tasks?

Trivial case

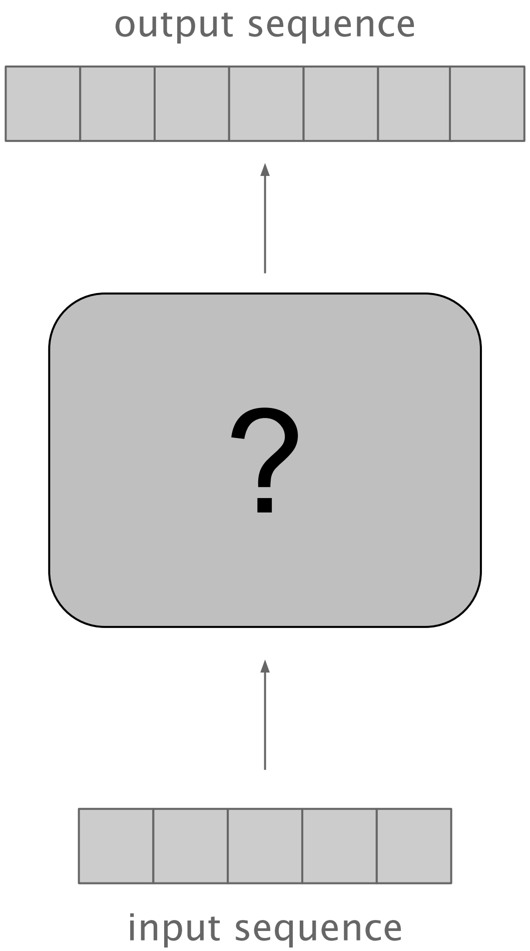
- Input and output sequences have the **same length**
- They can be modelled by a simple LSTM network



Code.Hub

General case

- What if we have a different length of input and output sequences?
- What architecture would we need to model these cases?



Code.Hub

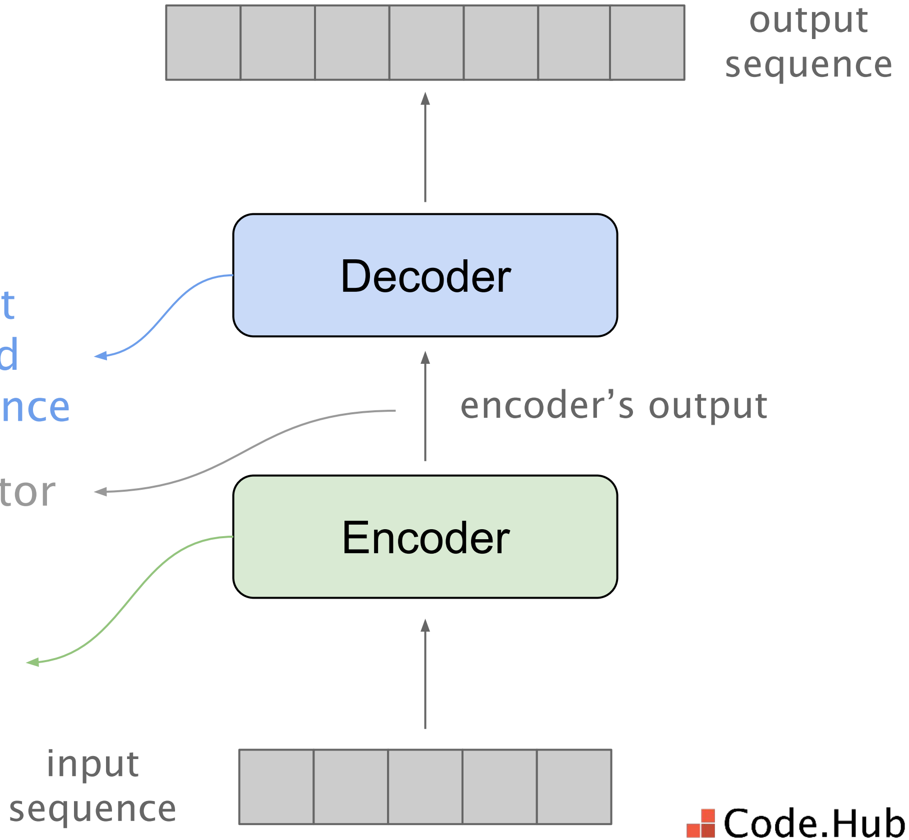
Sequence-to-sequence models

- Typically have an “encoder-decoder” architecture
- Encoder and decoder are sequential models

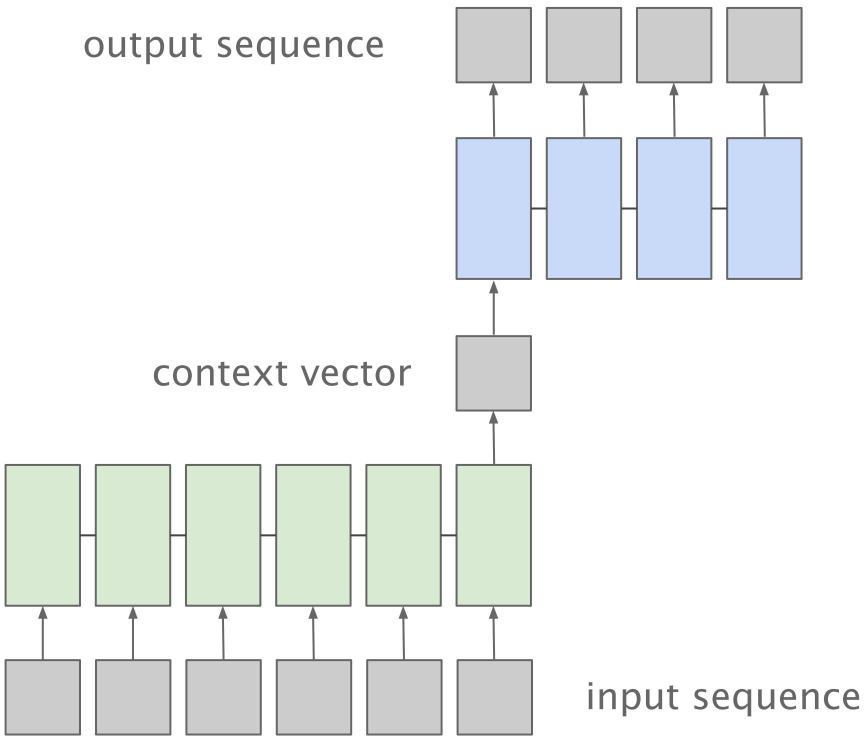
decoder takes a constant length contect vector and produces the output sequence

context vector

encoder takes an arbitrary length input and “compresses” it into a constant-length context vector

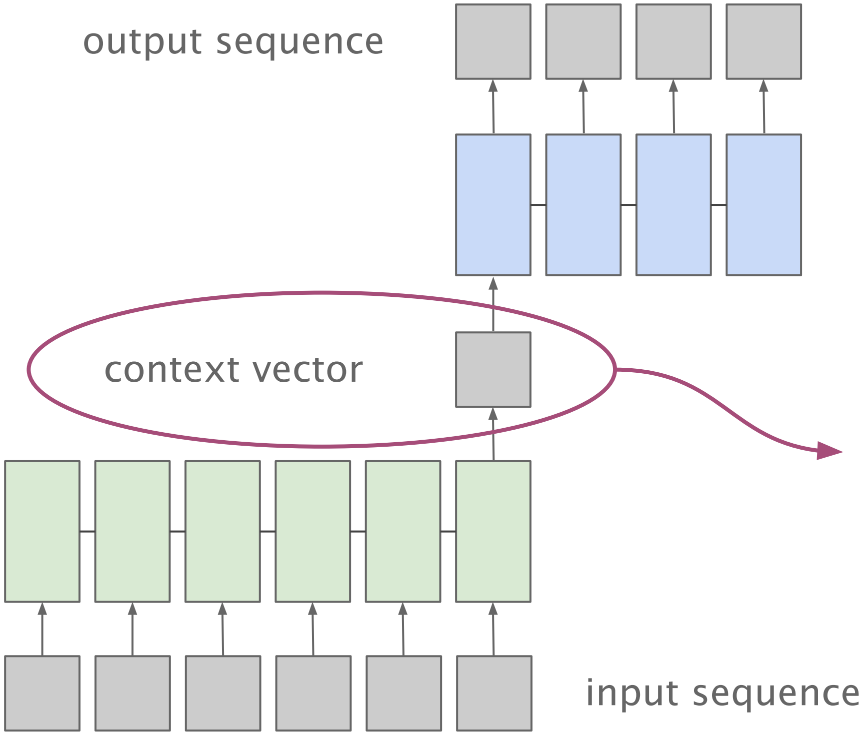


A deeper look...



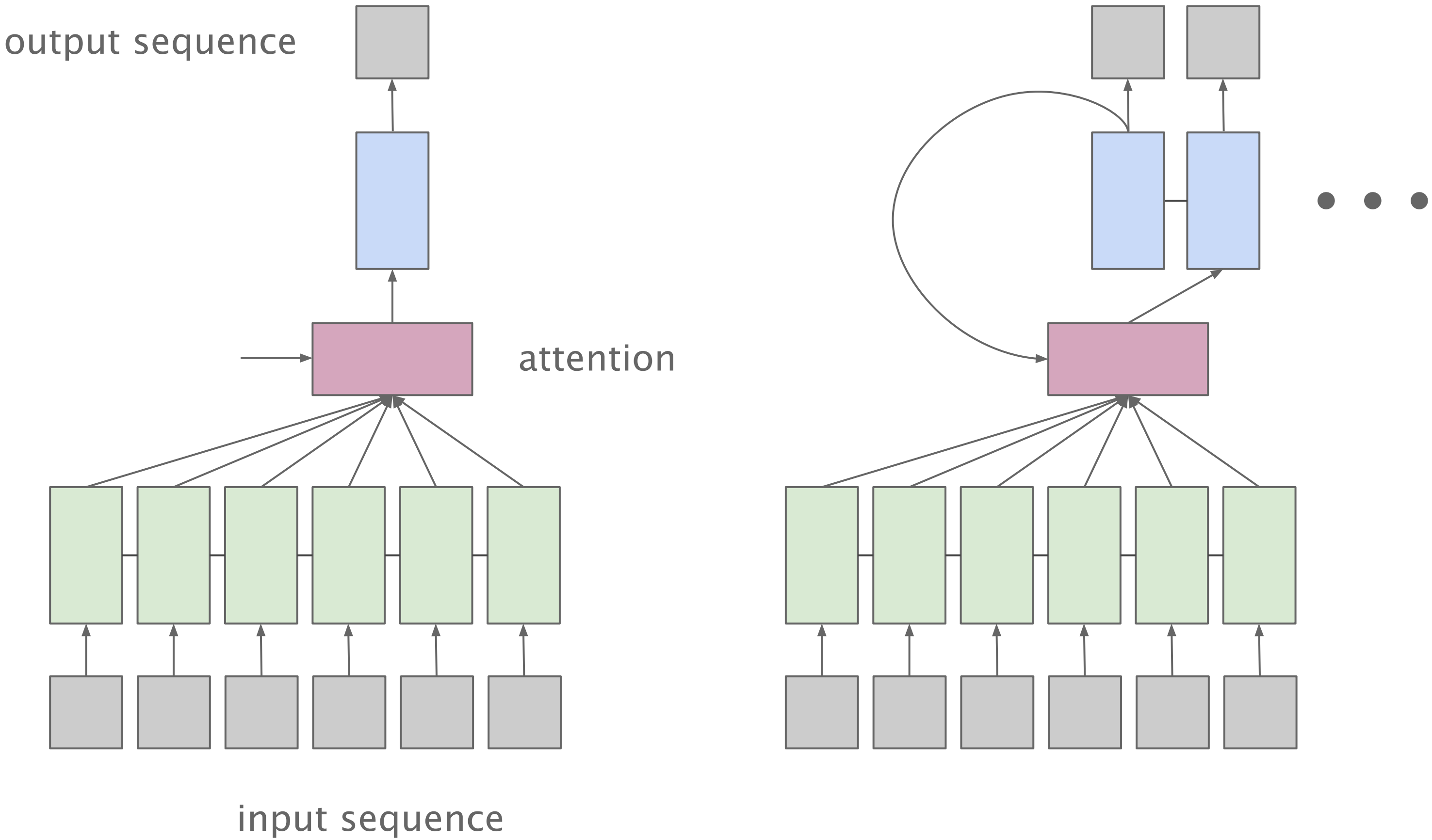
Code.Hub

Limitations

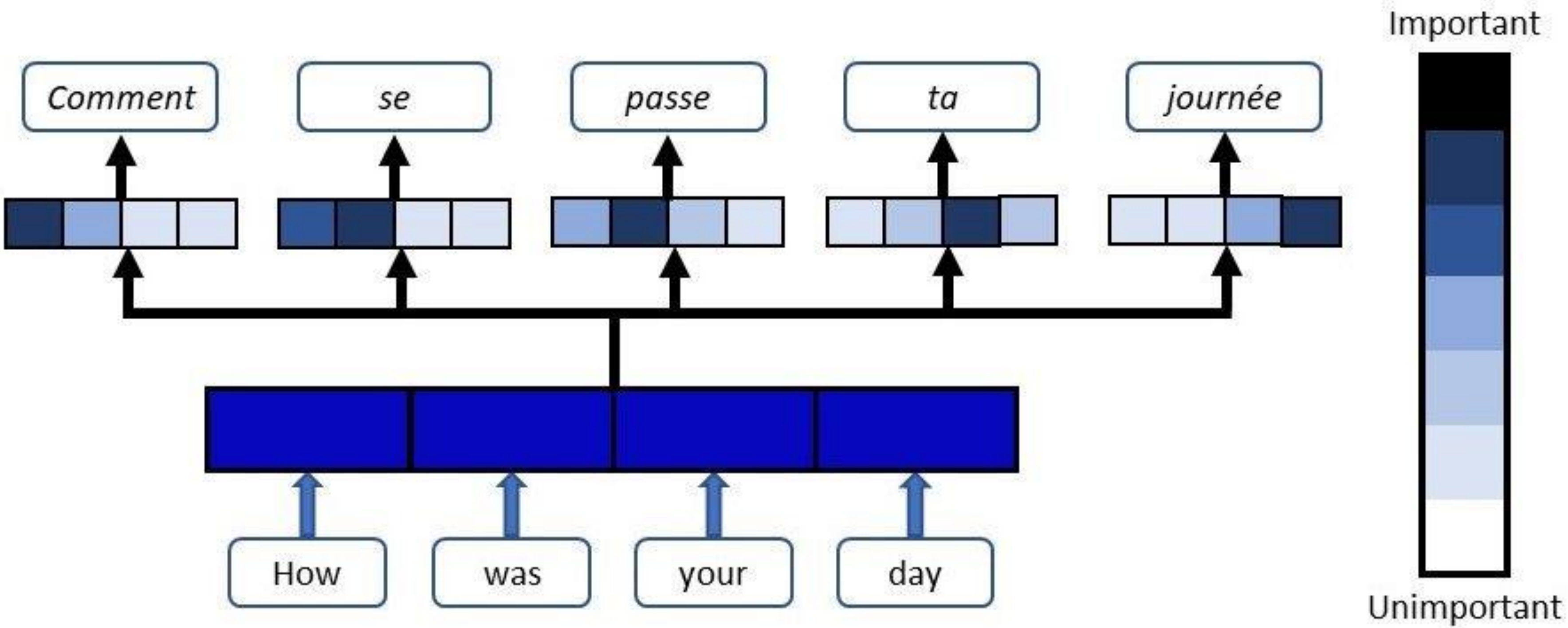


the “context vector” needs to encode all relevant information about the input sequence

Attention Mechanism



Attention Mechanism



Code.Hub