

Introduction to Deep Learning

Georgia Tech CS 4650/7650
Fall 2020

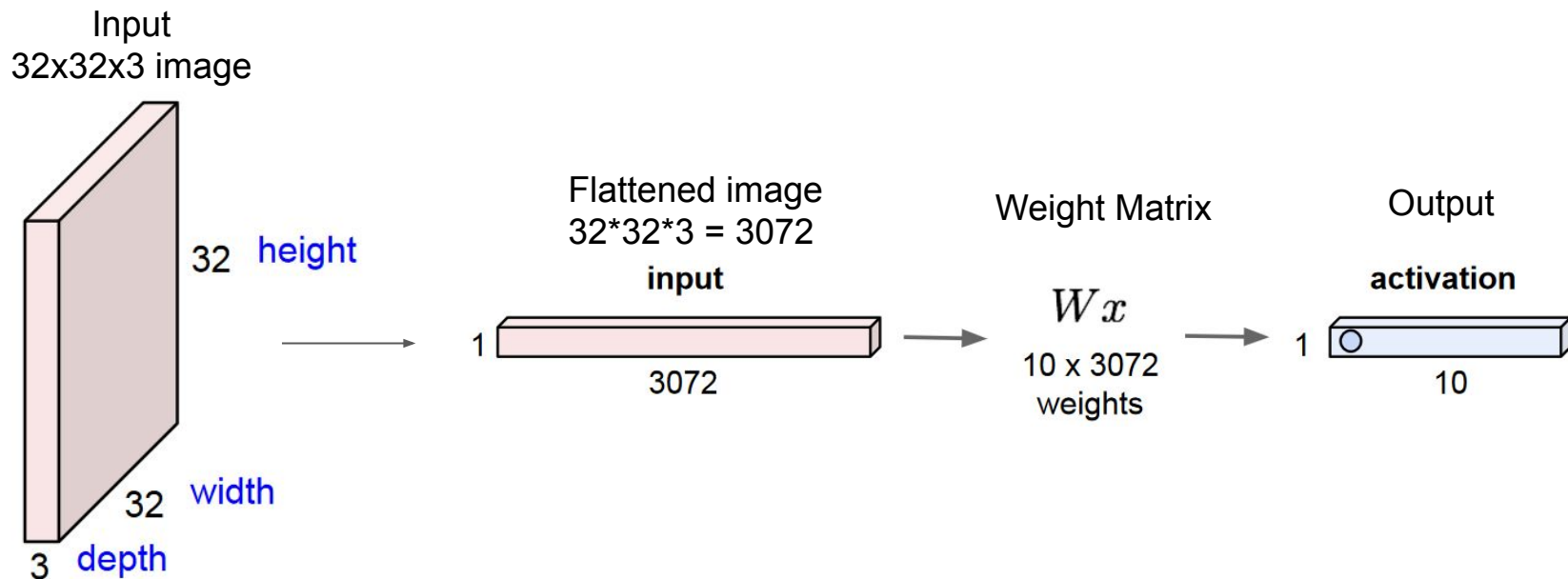
Outline

- **Deep Learning**
 - CNN
 - RNN
 - Attention
 - Transformer
- Pytorch
 - Introduction
 - Basics
 - Examples

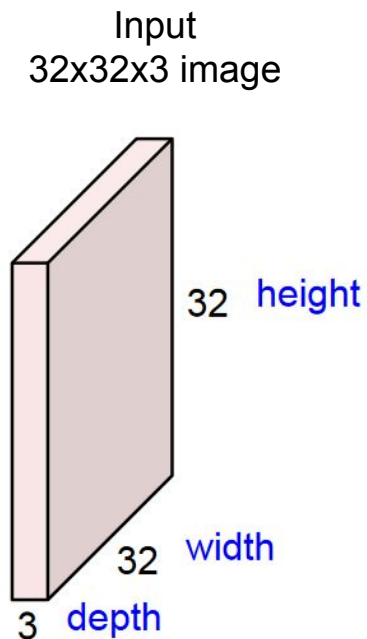
CNNs

Some slides borrowed from Fei-Fei Li & Justin
Johnson & Serena Yeung at Stanford.

Fully Connected Layer



Convolutional Layer



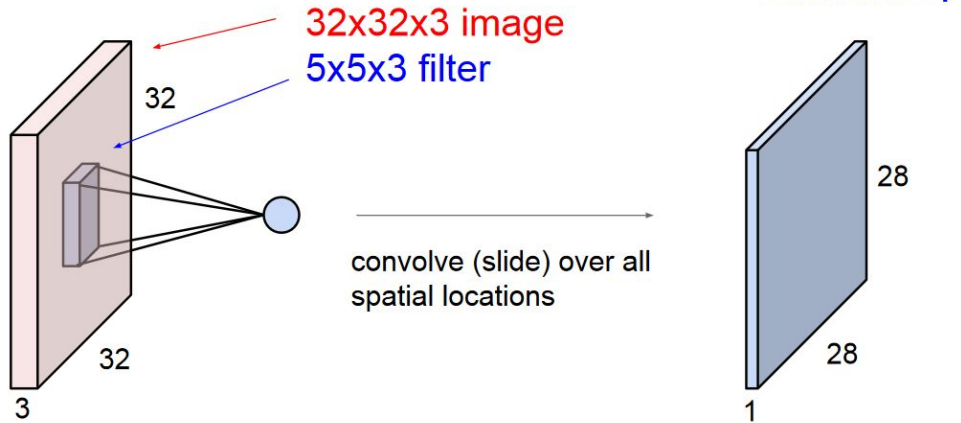
Filter
5x5x3



Convolve the filter with the image i.e.
“slide over the image spatially,
computing dot products”

Filters always extend the full depth of
the input volume.

Convolutional Layer

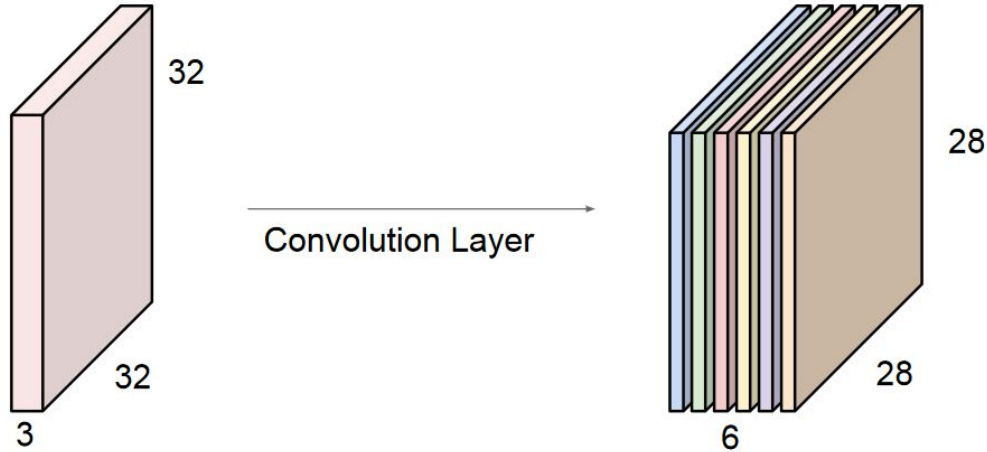


At each step during the convolution, the filter acts on a region in the input image and results in a single number as output.

This number is the result of the dot product between the values in the filter and the values in the 5x5x3 chunk in the image that the filter acts on.

Combining these together for the entire image results in the activation map.

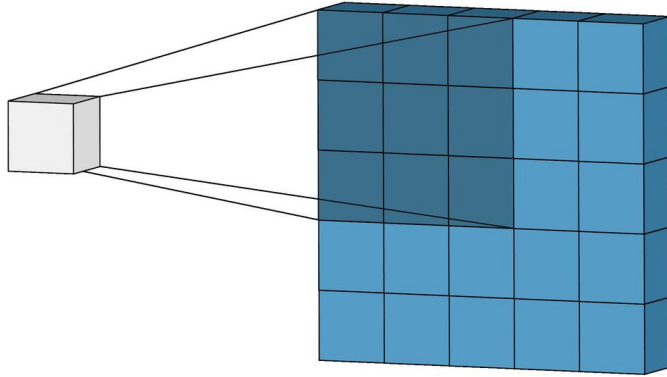
Convolutional Layer



Filters can be stacked together.

Example- If we had 6 filters of shape 5×5 , each would produce an activation map of $28 \times 28 \times 1$ and our output would be a “new image” of shape $28 \times 28 \times 6$.

Convolutional Layer

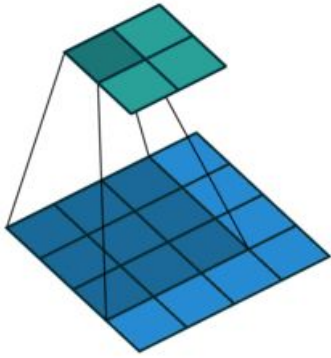


| | | | | |
|-------|-------|-------|---|---|
| 3_0 | 3_1 | 2_2 | 1 | 0 |
| 0_2 | 0_2 | 1_0 | 3 | 1 |
| 3_0 | 1_1 | 2_2 | 2 | 3 |
| 2 | 0 | 0 | 2 | 2 |
| 2 | 0 | 0 | 0 | 1 |

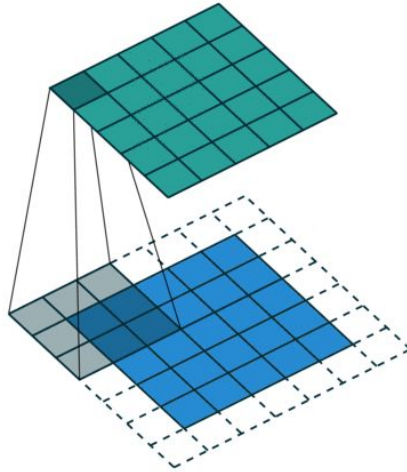
| | | |
|------|------|------|
| 12.0 | 12.0 | 17.0 |
| 10.0 | 17.0 | 19.0 |
| 9.0 | 6.0 | 14.0 |

Visualizations borrowed from Irhum Shafkat's [blog](#).

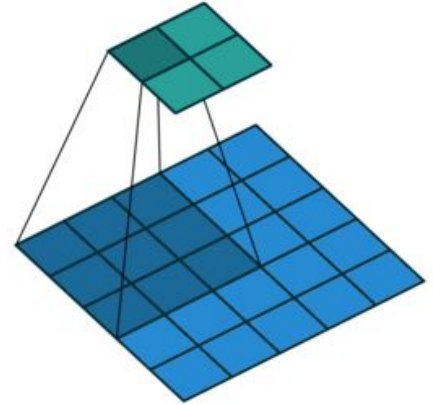
Convolutional Layer



Standard
Convolution

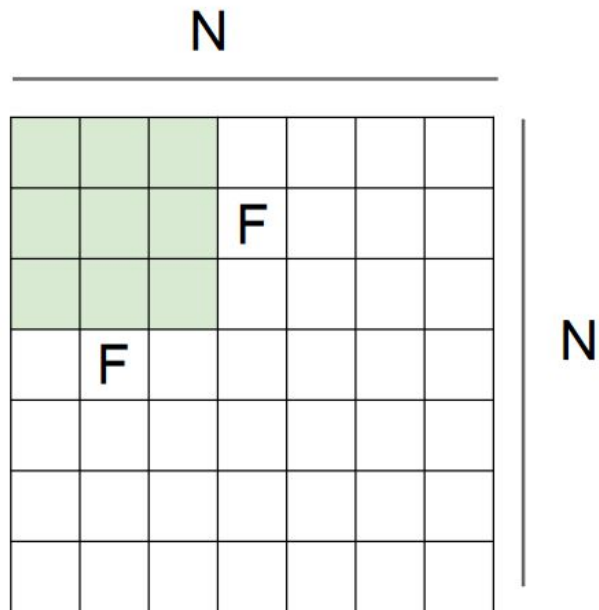


Convolution
with Padding



Convolution
with strides

Convolutional Layer

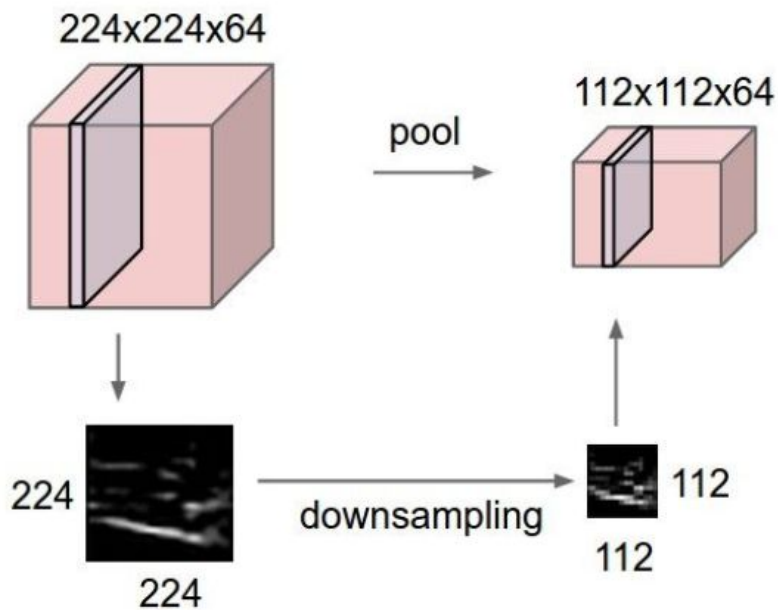


Output Size:
 $(N - F)/\text{stride} + 1$

e.g. $N = 7$, $F = 3$, stride 1
 $\Rightarrow (7 - 3)/1 + 1 = 5$

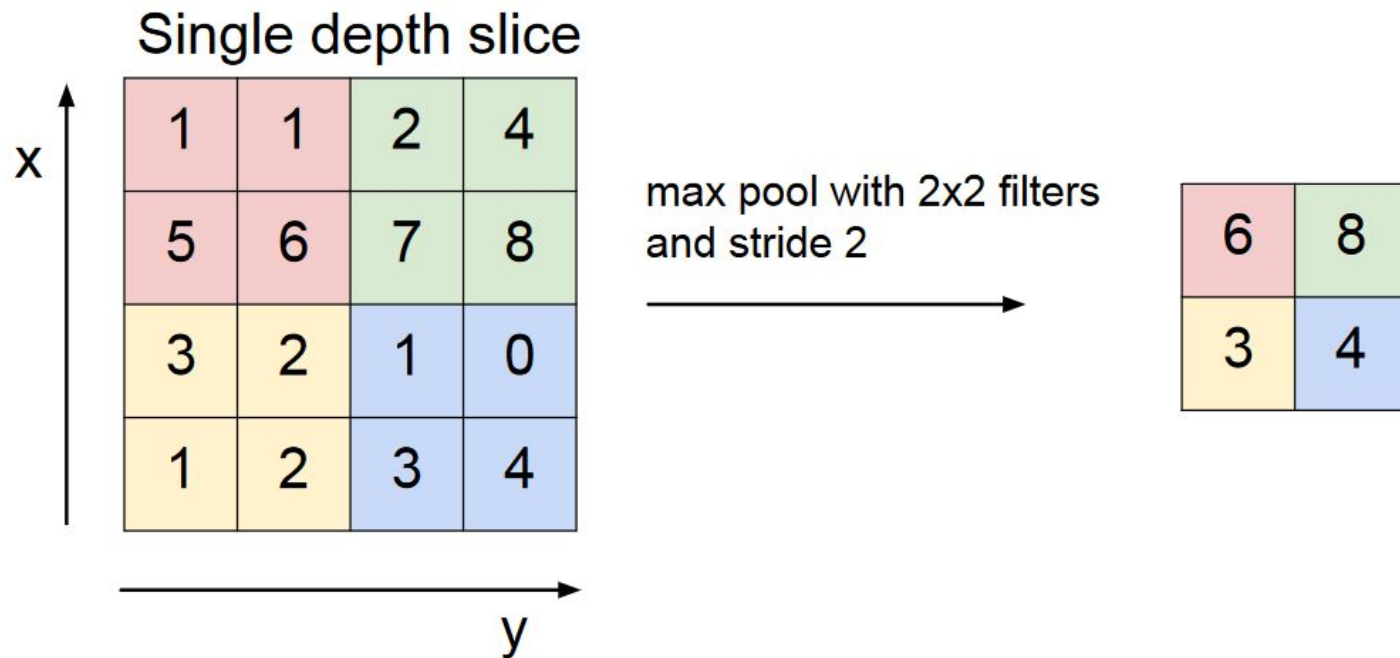
e.g. $N = 7$, $F = 3$, stride 2
 $\Rightarrow (7 - 3)/2 + 1 = 3$

Pooling Layer

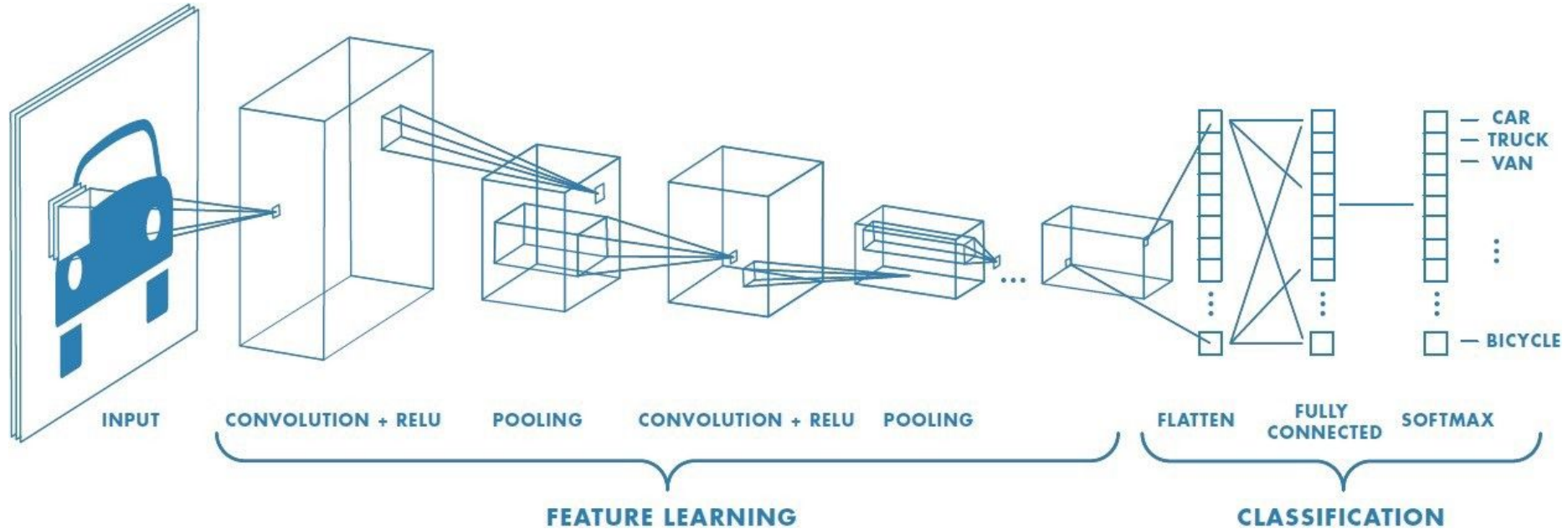


- makes the representations smaller and more manageable
- operates over each activation map independently

Max Pooling

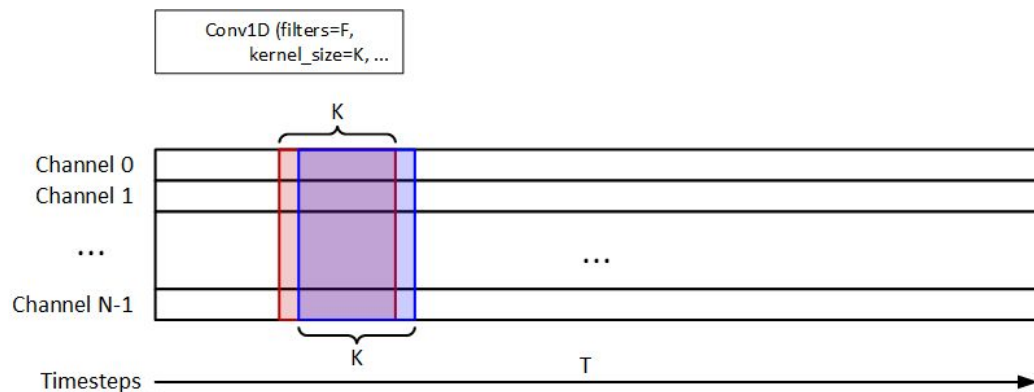


ConvNet Layer



Application in text

- NLP doesn't use convolutional nets a lot
- Some adjacent applications exist, such as graph convolutions or image-to-text
- For text sequences, it sometimes helps to use **1-dimensional** convolutions (because embedding dimension ordering has no intrinsic meaning)
- What does this basically amount to?
- **N-gram features.**

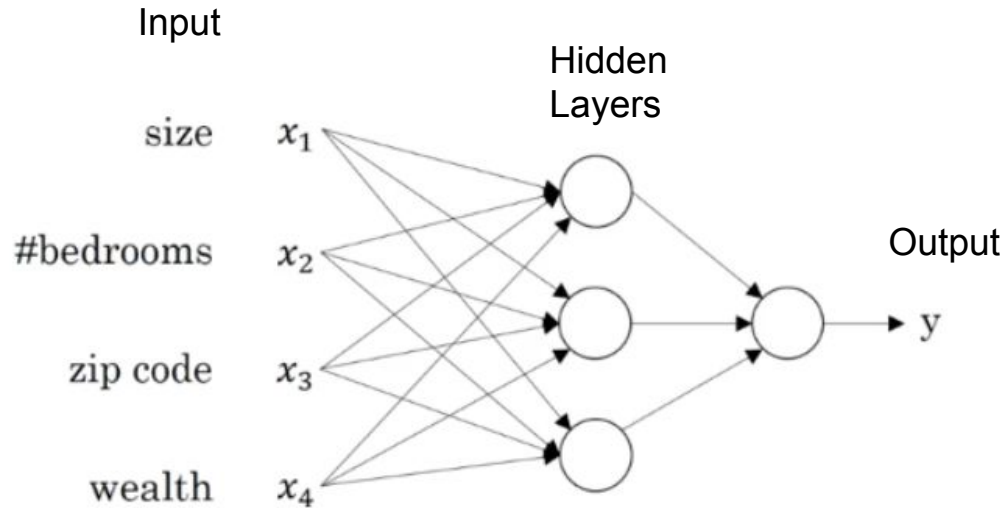


RNNs

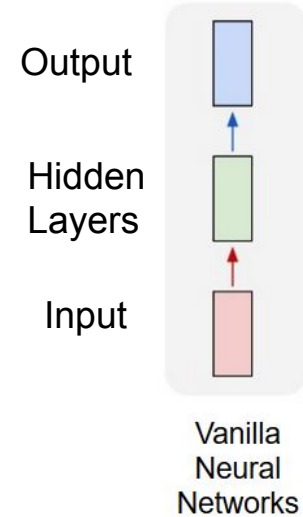
Some slides borrowed from Fei-Fei Li & Justin
Johnson & Serena Yeung at Stanford.

Vanilla Neural Networks

House Price Prediction



one to one

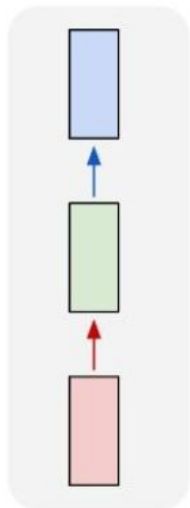


How to model sequences?

- Text Classification: Input Sequence \rightarrow Output label
- Translation: Input Sequence \rightarrow Output Sequence
- Image Captioning: Input image \rightarrow Output Sequence

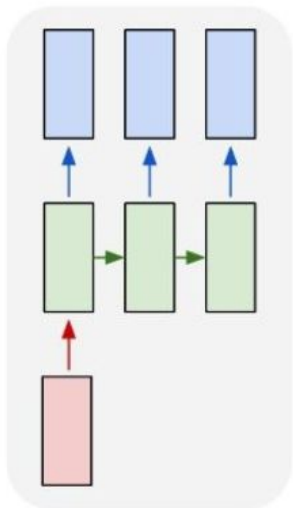
RNN - Recurrent Neural Networks

one to one



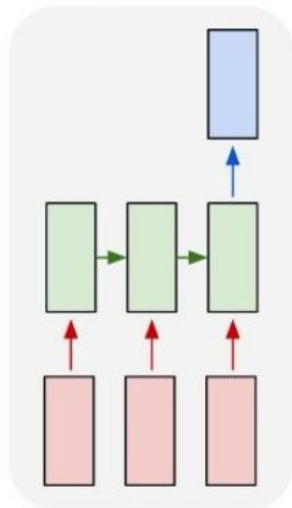
Vanilla
Neural
Networks

one to many



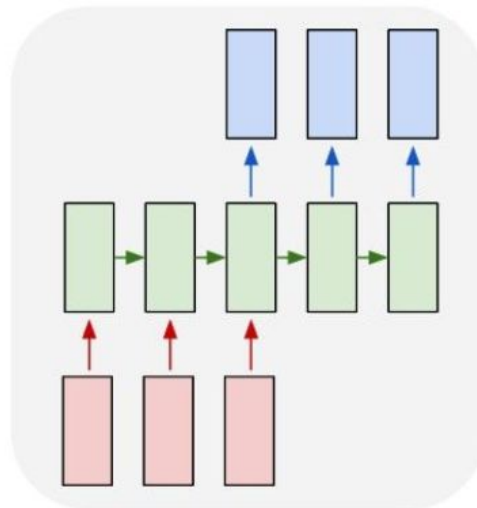
e.g.
Image captioning

many to one



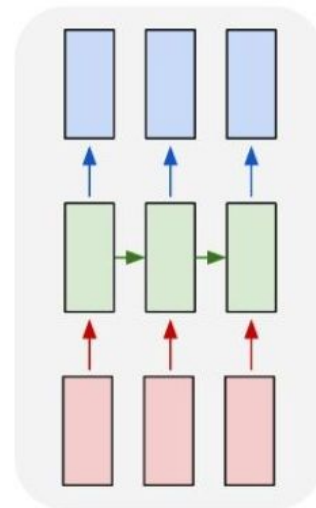
e.g.
Text classification

many to many



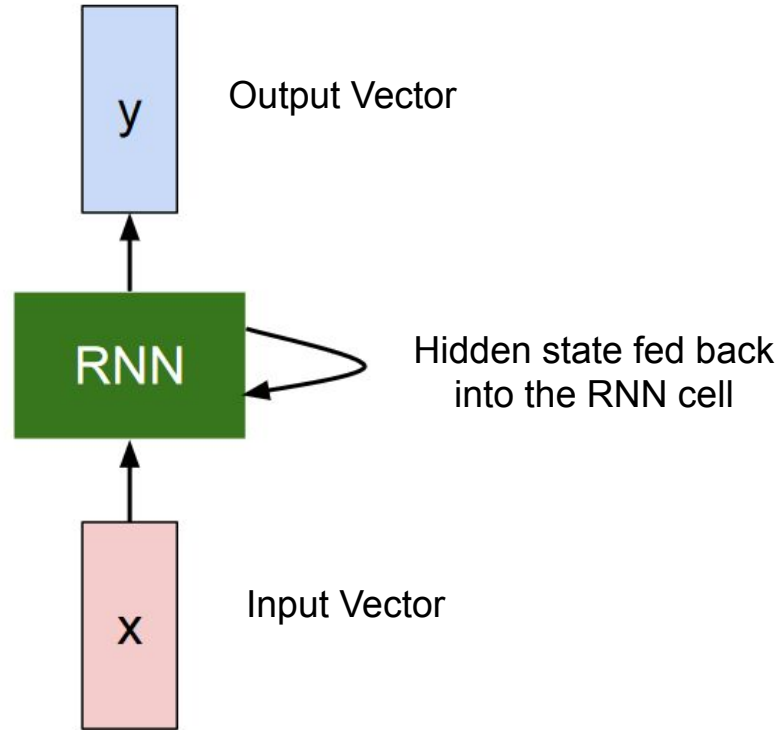
e.g.
Translation

many to many



e.g.
POS tagging

RNN - Representation

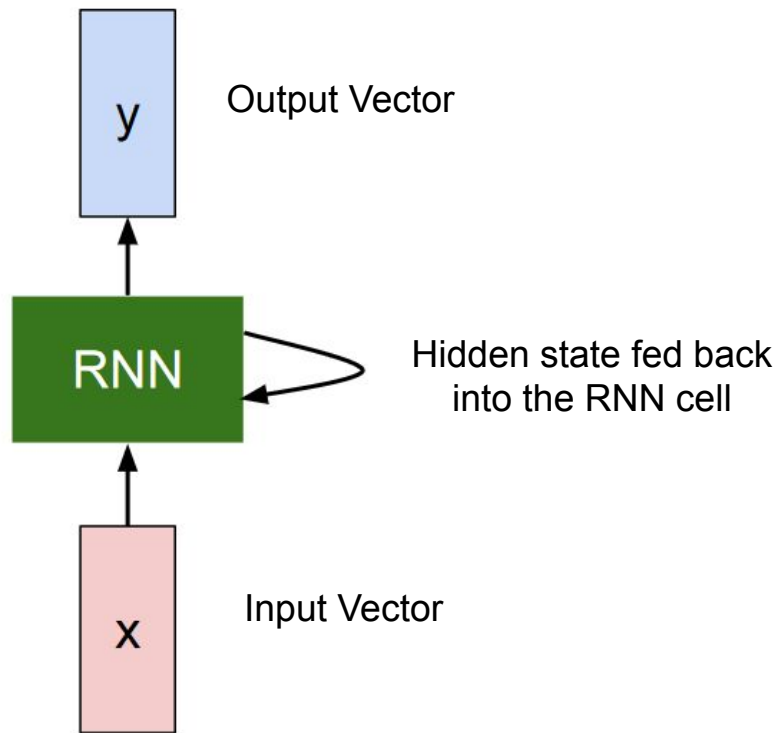


RNN - Recurrence Relation

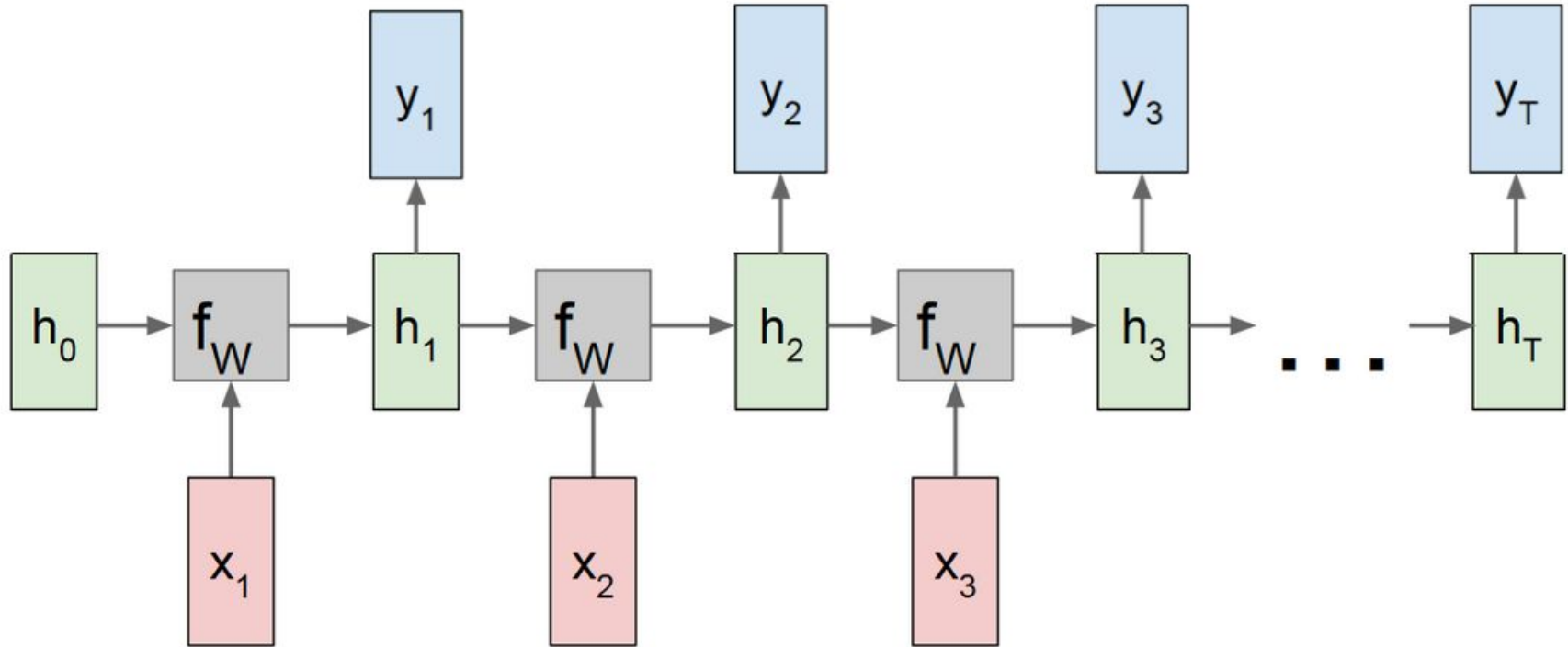
The RNN cell consists of a hidden state that is updated whenever a new input is received. At every time step, this hidden state is fed back into the RNN cell.

$$h_t = f_W(h_{t-1}, x_t)$$

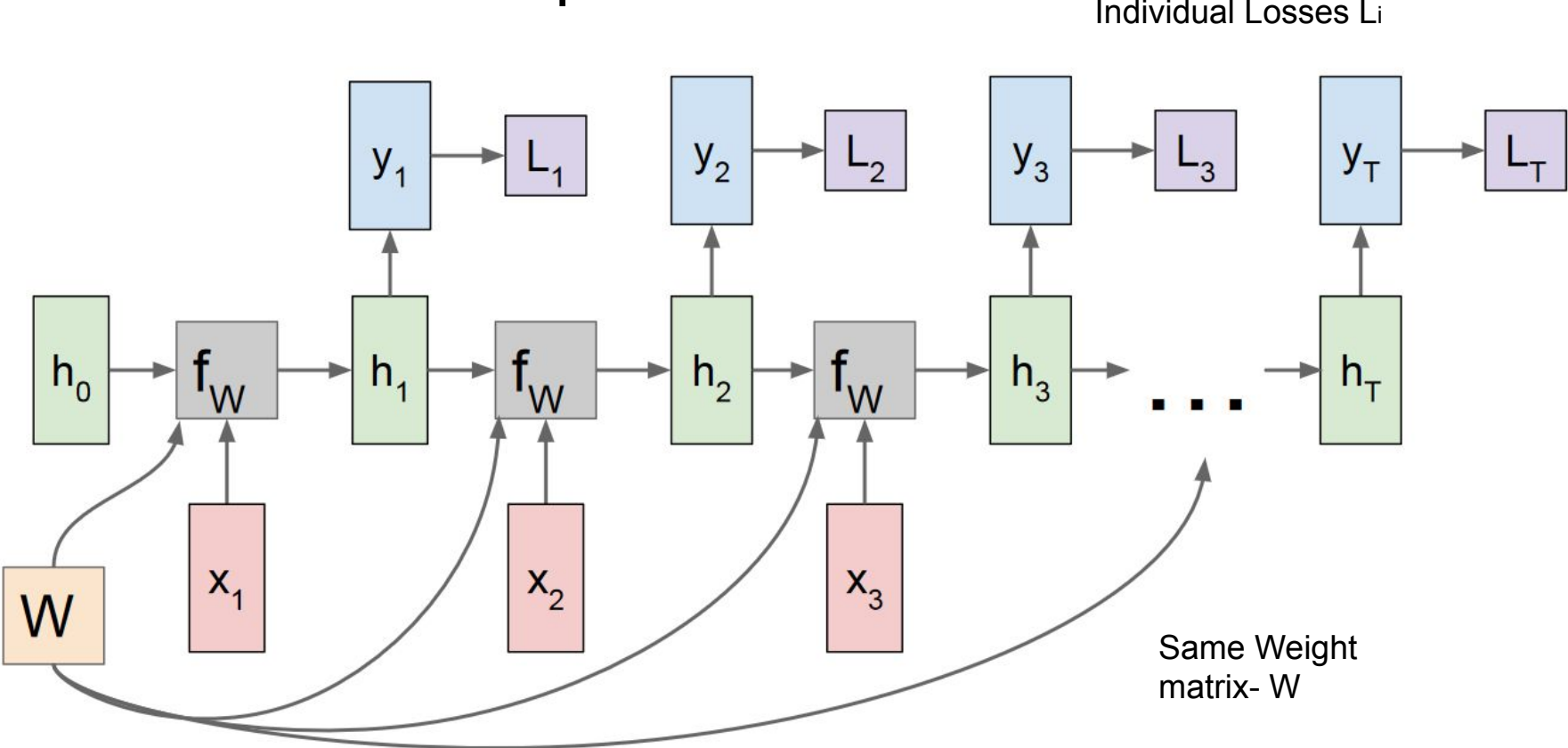
new state some function with parameters W old state input vector at some time step



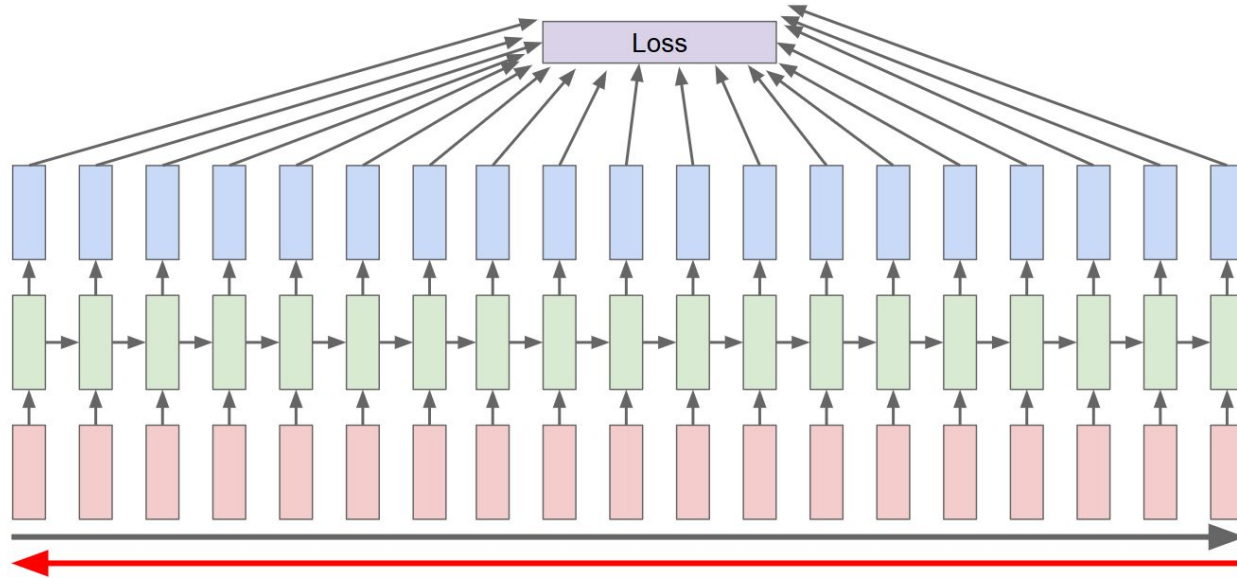
RNN - Rolled out representation



RNN - Rolled out representation

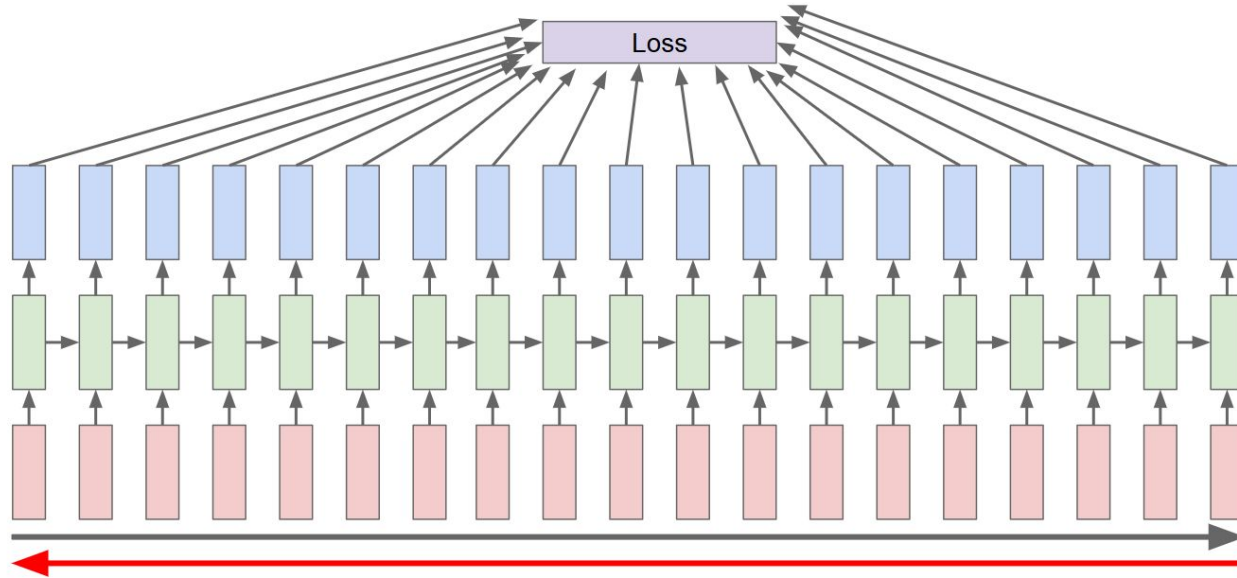


RNN - Backpropagation Through Time



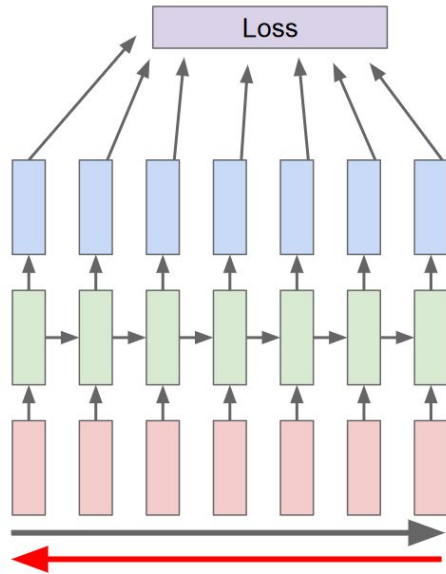
Forward pass through entire sequence to produce intermediate hidden states, output sequence and finally the loss. Backward pass through the entire sequence to compute gradient.

RNN - Backpropagation Through Time



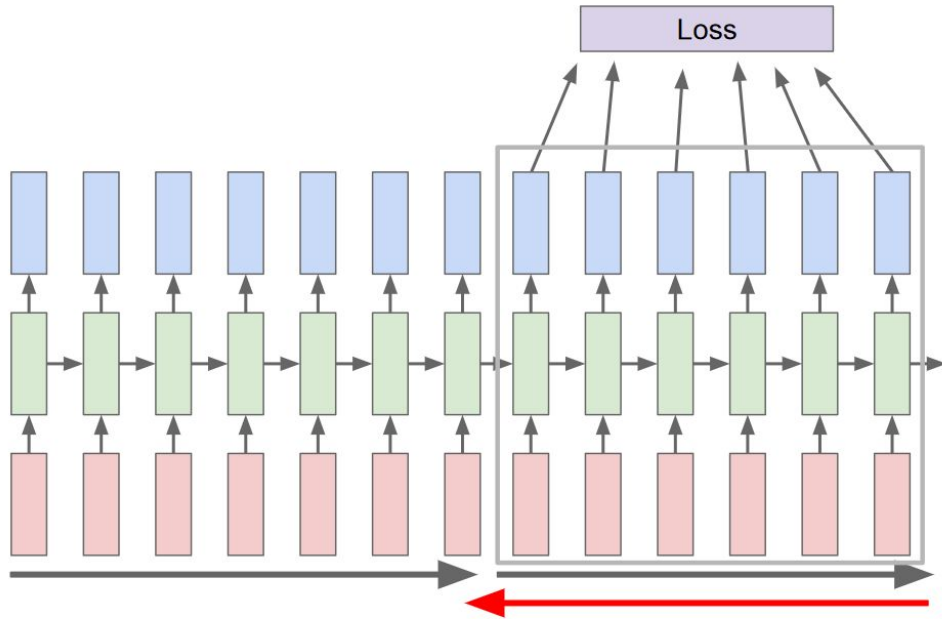
Running Backpropagation through time for the entire text would be very slow. Switch to an approximation-
Truncated Backpropagation Through Time

RNN - Truncated Backpropagation Through Time



Run forward and backward through chunks of the sequence instead of whole sequence

RNN - Truncated Backpropagation Through Time



Carry hidden states forward
in time forever, but only
backpropagate for some
smaller number of steps

RNN Types

The 3 most common types of Recurrent Neural Networks are:

1. Vanilla RNN
2. LSTM (Long Short-Term Memory)
3. GRU (Gated Recurrent Units)

Some good resources:

[Understanding LSTM Networks](#)

[An Empirical Exploration of Recurrent Network Architectures](#)

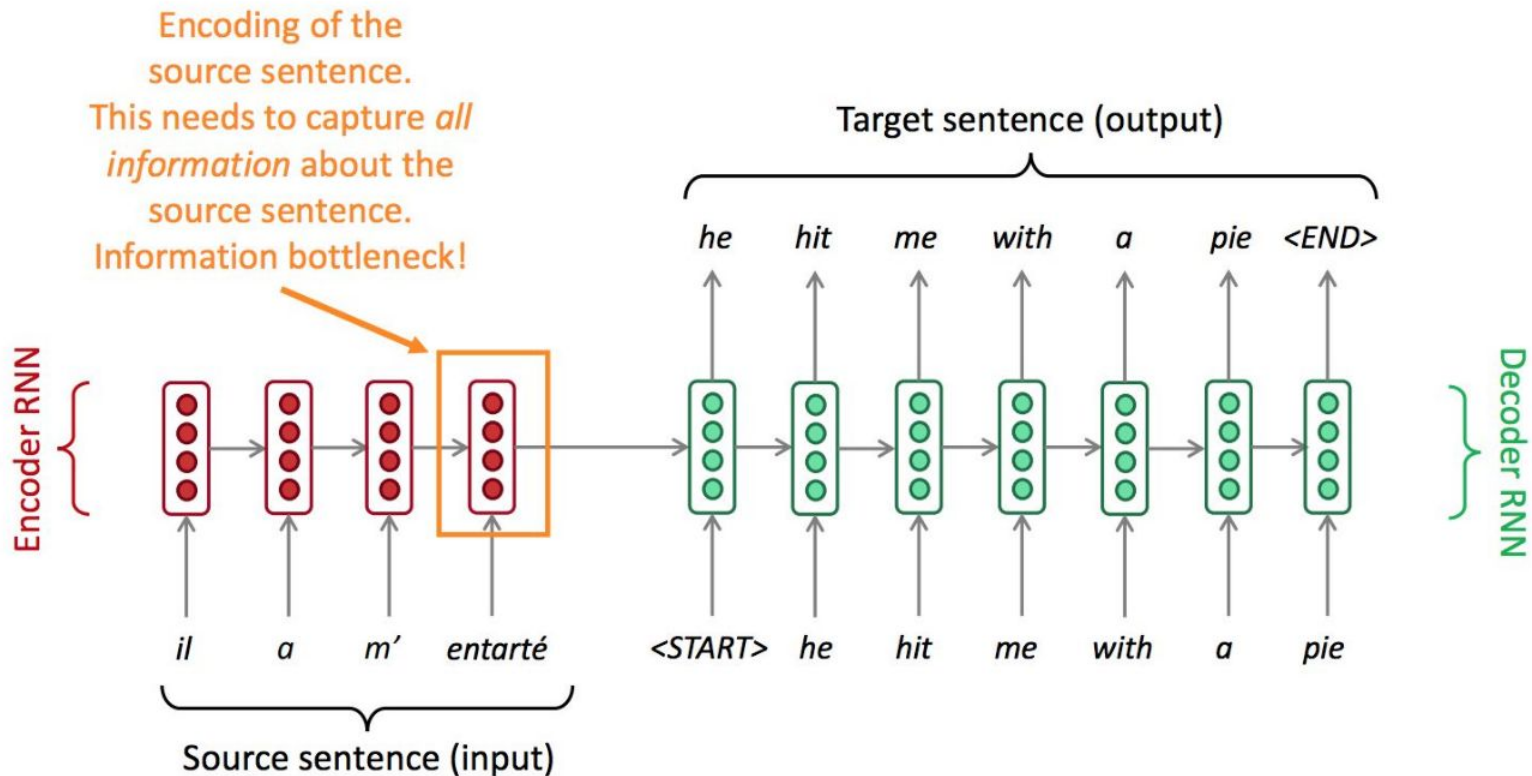
[Recurrent Neural Network Tutorial, Part 4 – Implementing a GRU/LSTM RNN with Python and Theano](#)

[Stanford CS231n: Lecture 10 | Recurrent Neural Networks](#)

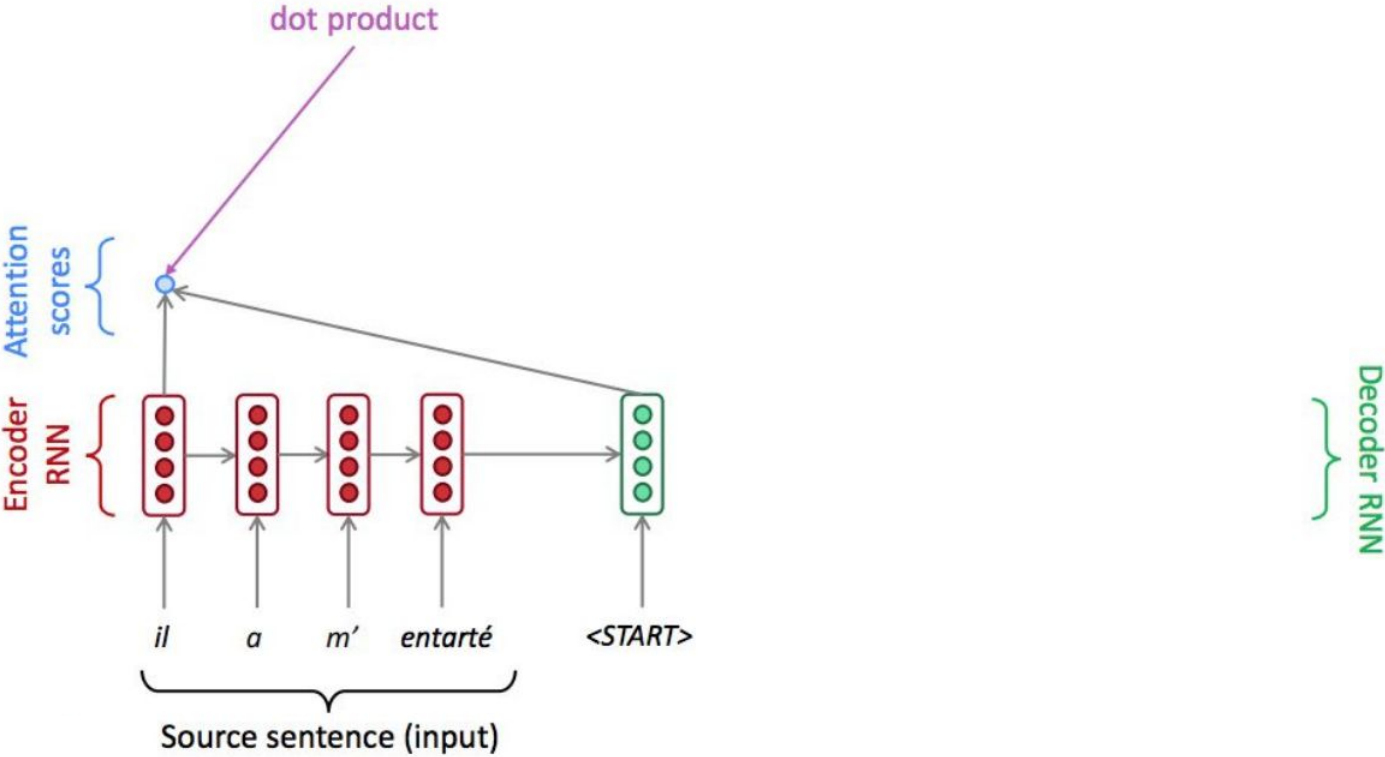
Attention

Some slides borrowed from Sarah Wiegrefe
at Georgia Tech and Abigail See, Stanford CS224n.

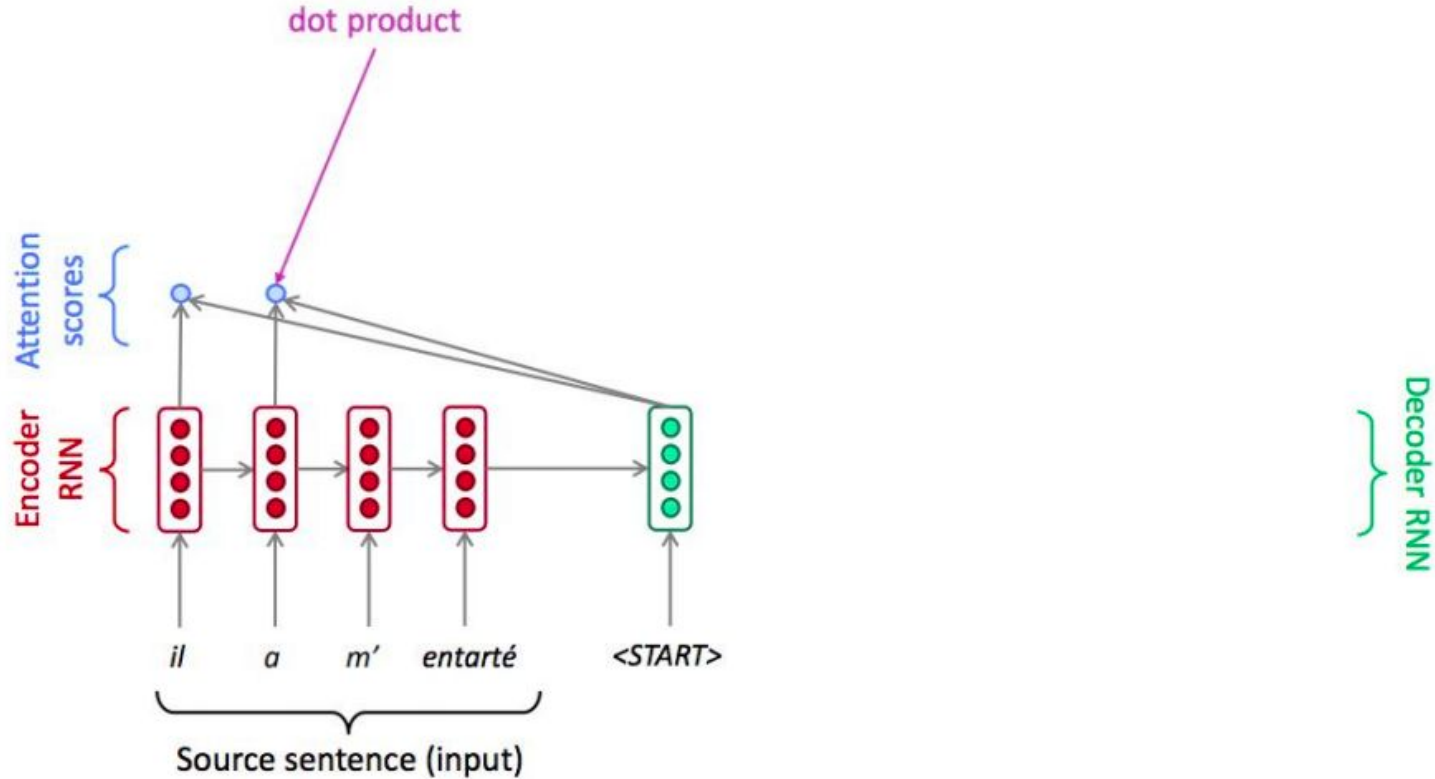
RNN



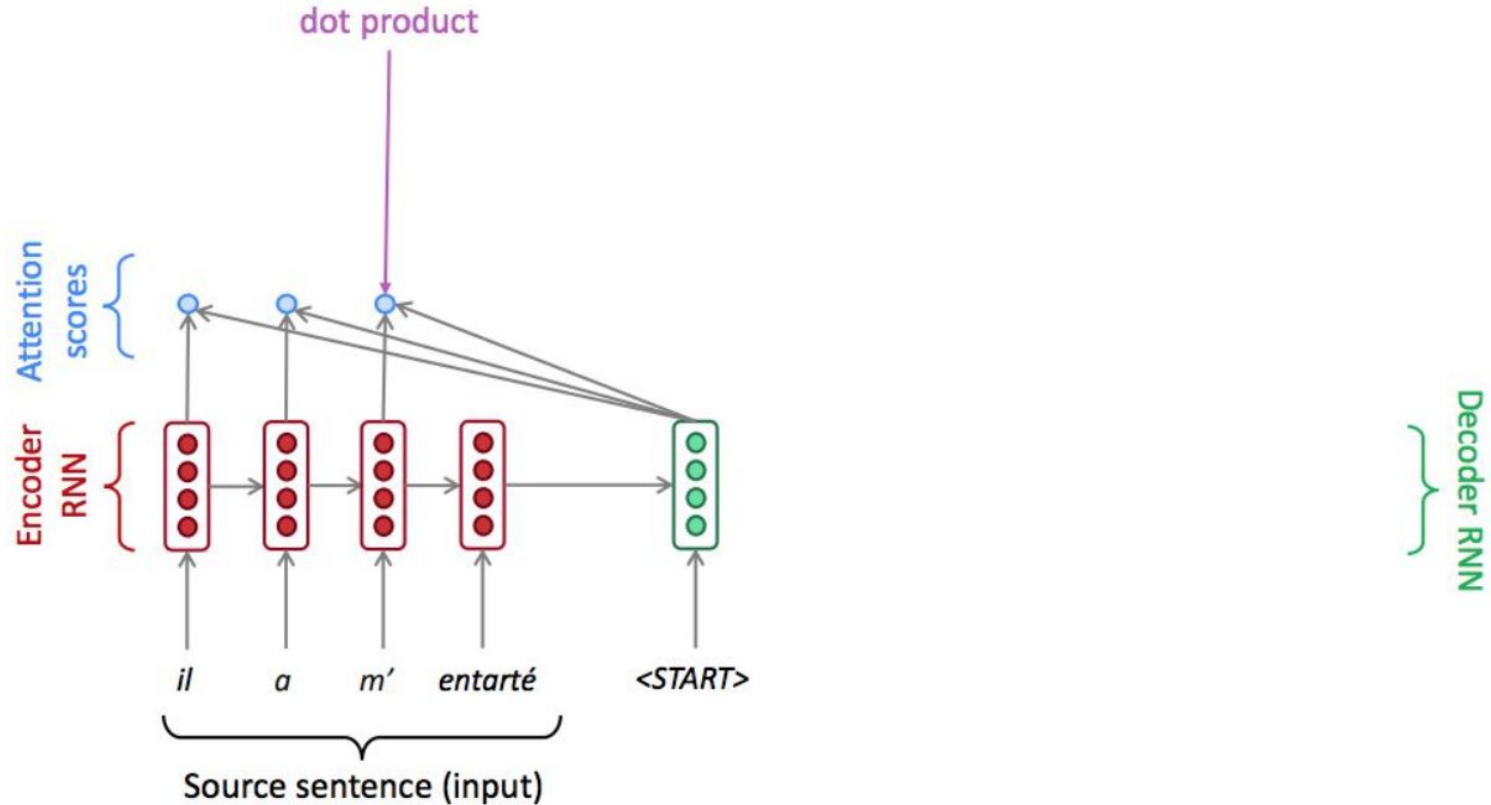
RNN - Attention



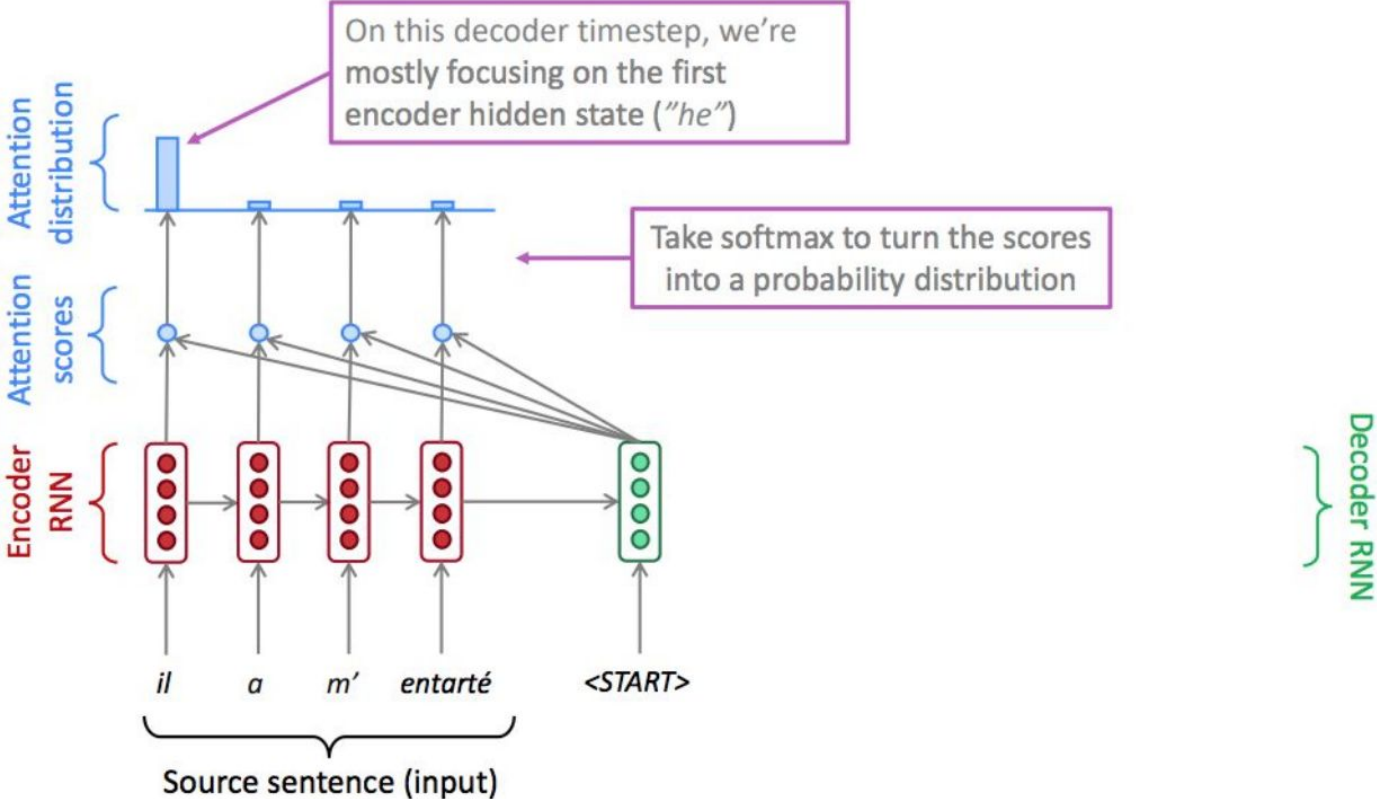
RNN - Attention



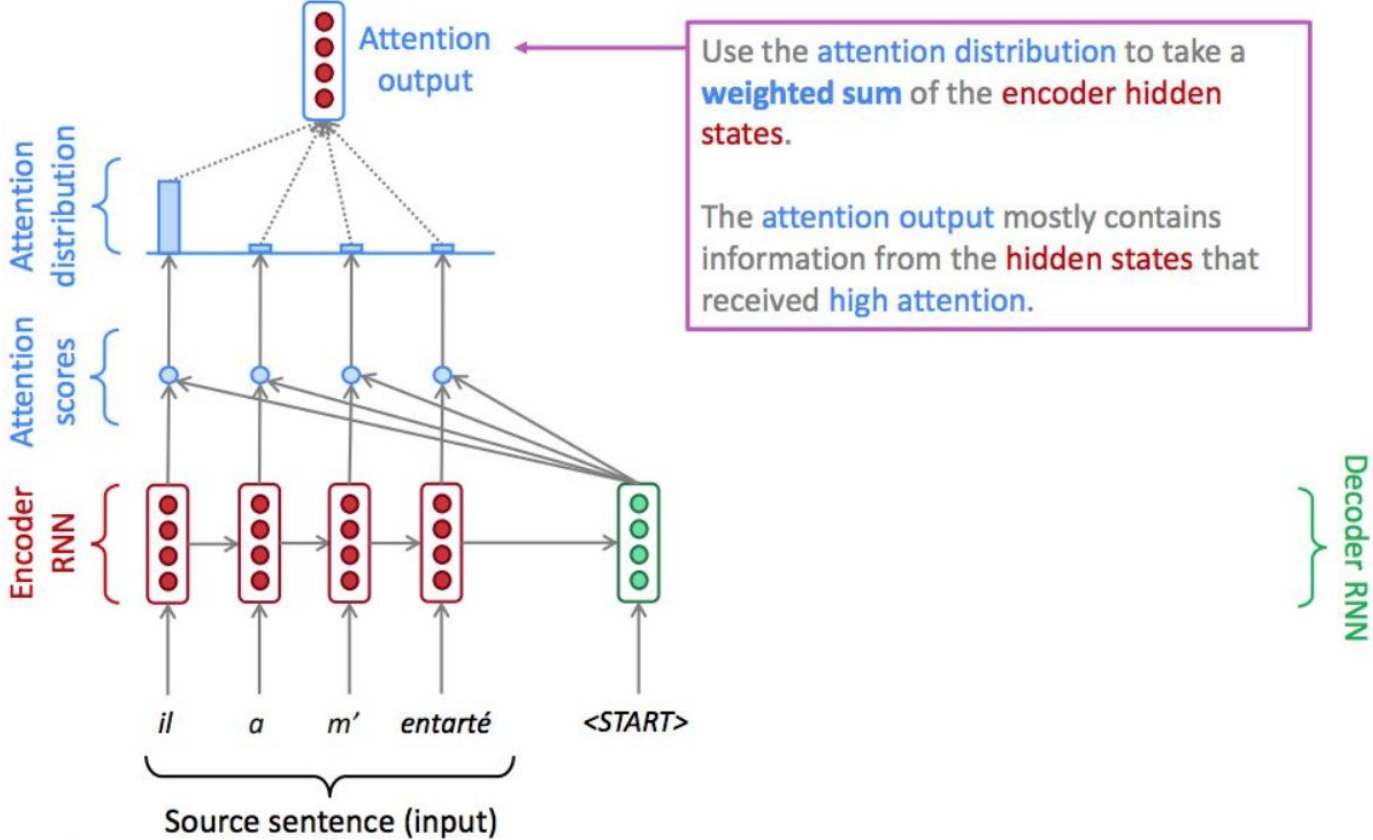
RNN - Attention



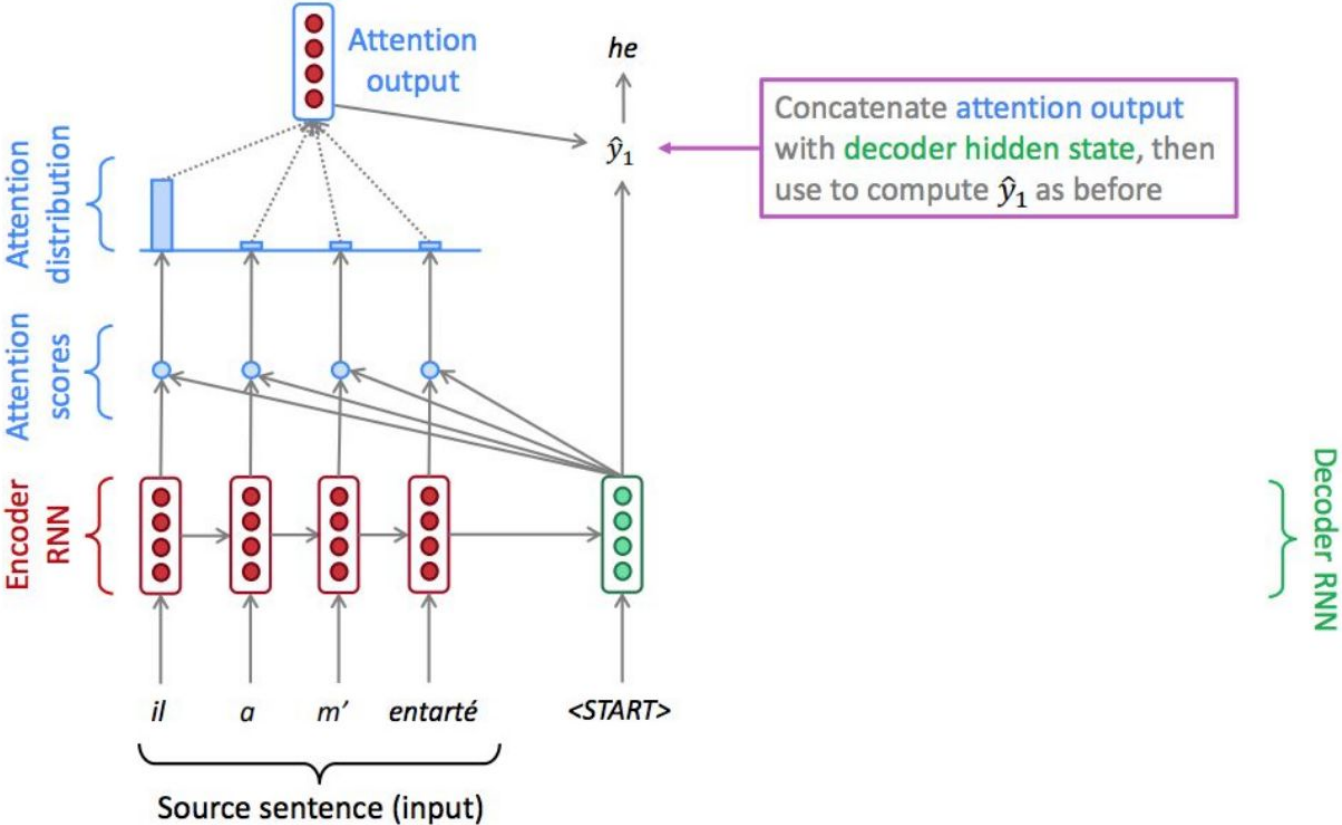
RNN - Attention



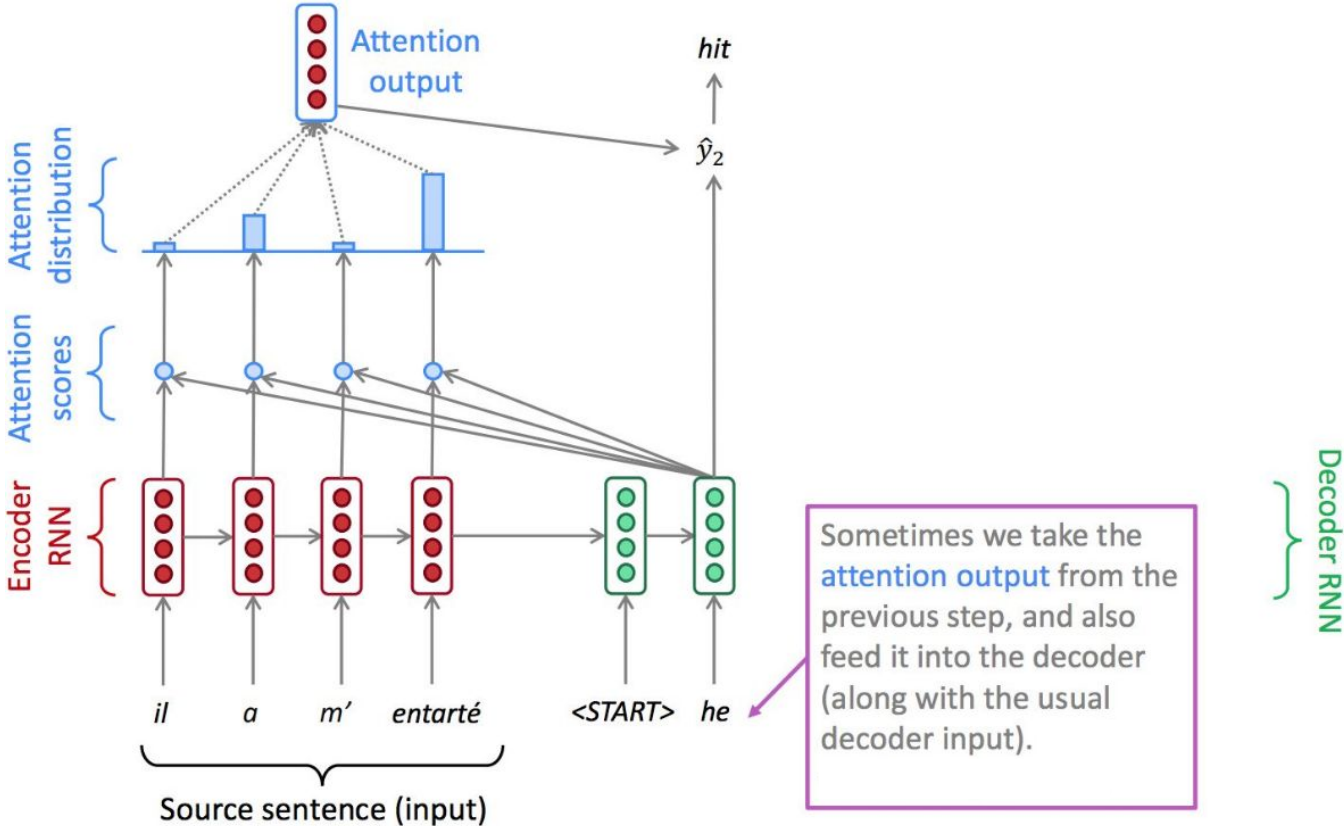
RNN - Attention



RNN - Attention

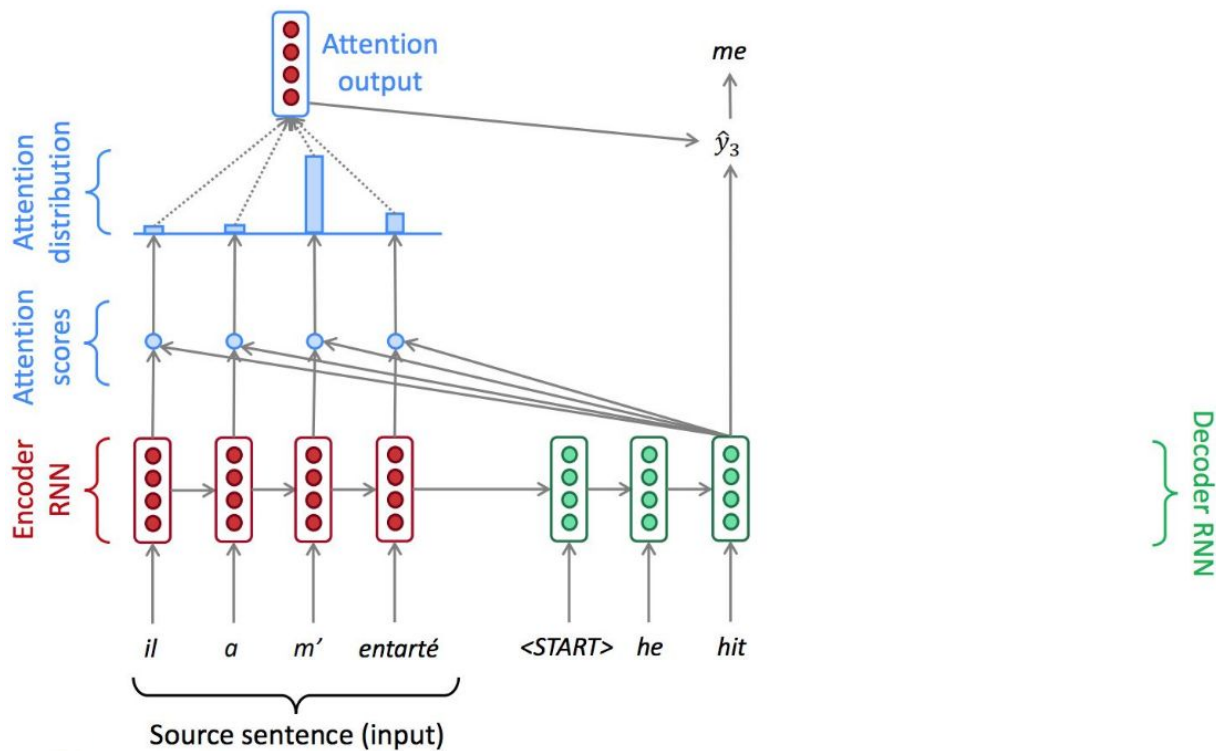


RNN - Attention



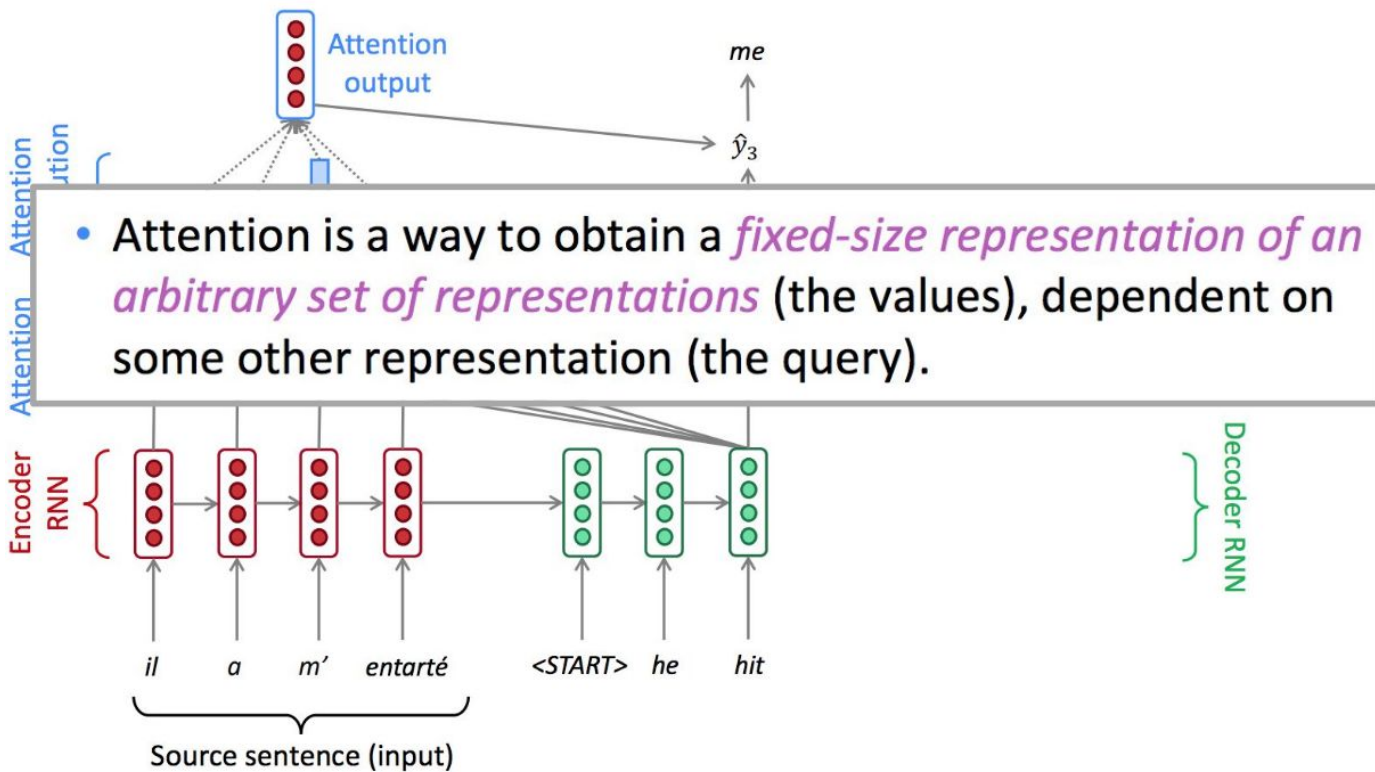
RNN - Attention

Sequence-to-sequence with attention



RNN - Attention

Sequence-to-sequence with attention



Attention

- For query vector \mathbf{q} , key vector \mathbf{k}_i representing value \mathbf{v}_i
 - s_i is the similarity score between \mathbf{q} and \mathbf{k}_i
- Normalize the similarity scores to sum to 1
 - $p_i = \text{Softmax}(s_i)$
- Compute \mathbf{z} as the weighted sum of the value vectors \mathbf{v}_i weighted by their scores p_i
- In Machine Translation & Image Captioning, the keys and values are the same.
 - But, they could be different.

$$z = \sum_{i=1}^L p_i v_i$$

Attention is great

- Attention significantly **improves performance** (in many applications)
 - It's very useful to allow decoder to focus on certain parts of the source
- Attention **solves the bottleneck problem**
 - Attention allows decoder to look directly at source; bypass bottleneck
- Attention **helps with vanishing gradient problem**
 - Provides shortcut to faraway states
- Attention provides **some interpretability**
 - By inspecting attention distribution, we can see what the decoder was focusing on

Drawbacks of RNN

- RNNs involve sequential computation
 - can't parallelize = time-consuming
- RNNs “forget” past information
- No explicit modeling of long and short range dependencies

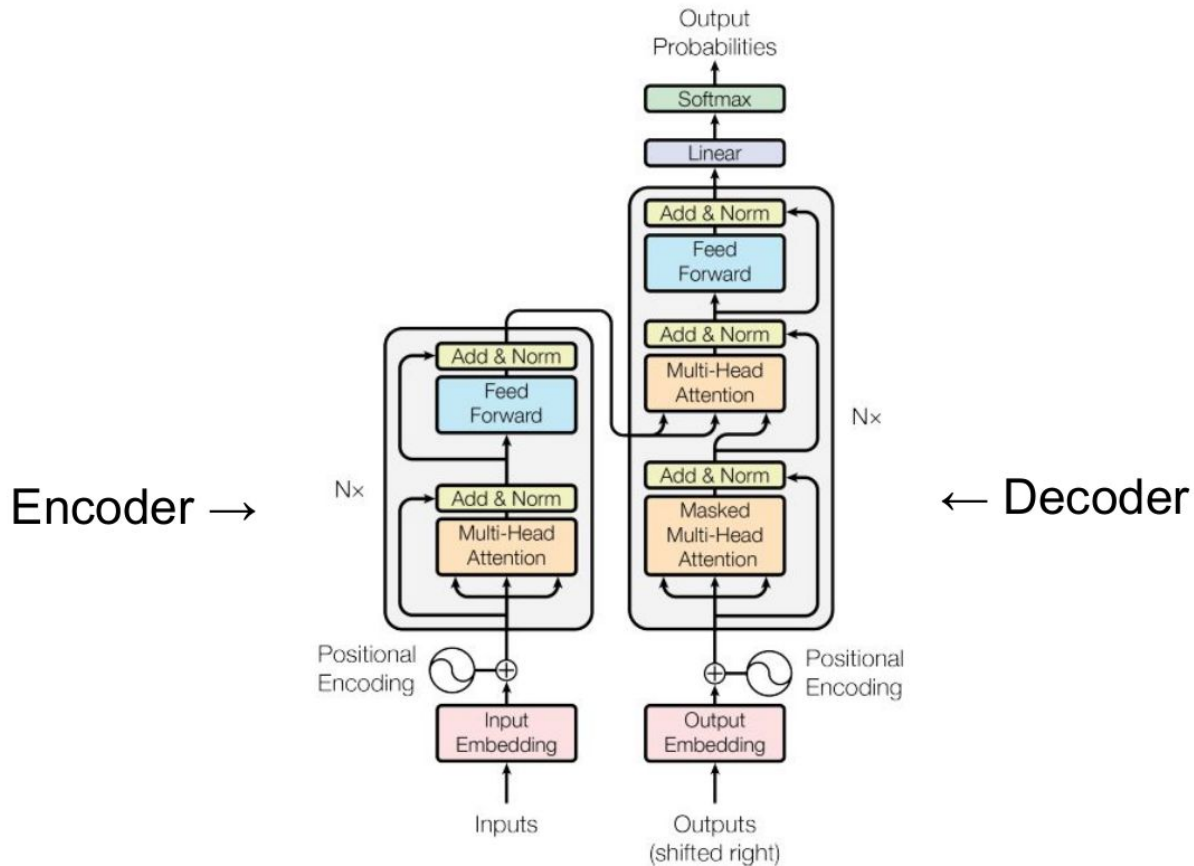
Transformer

Some slides borrowed from Sarah Wiegreffe
at Georgia Tech and “The Illustrated Transformer”
<https://jalammar.github.io/illustrated-transformer/>

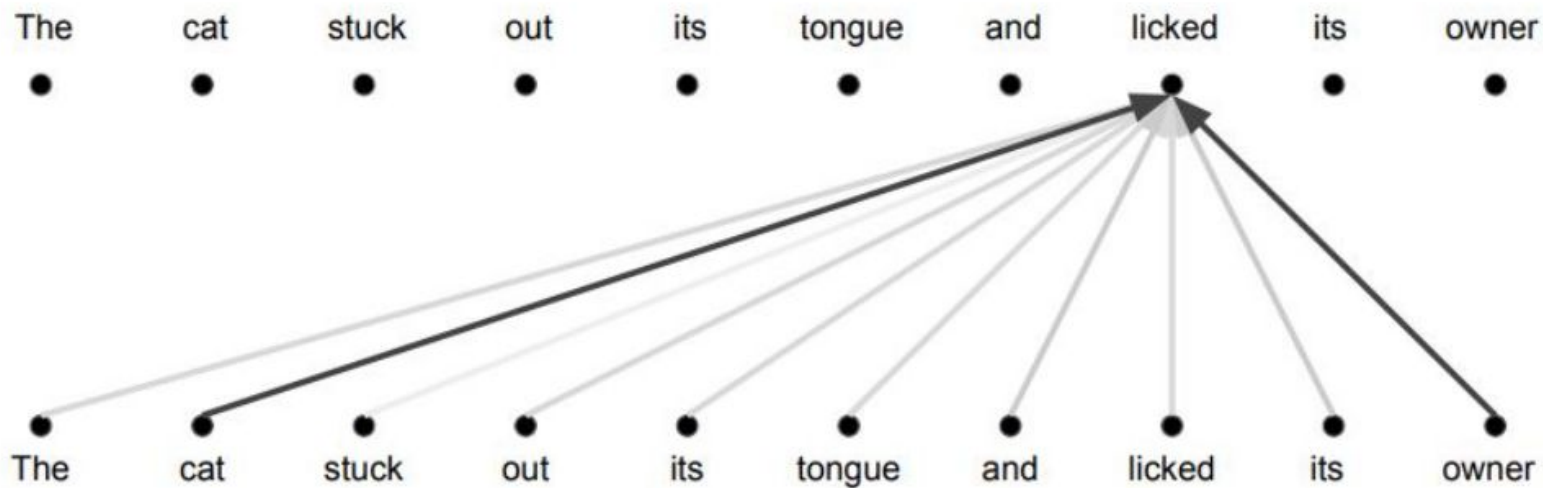
“Attention is All You Need”

(Vaswani et. al 2017)

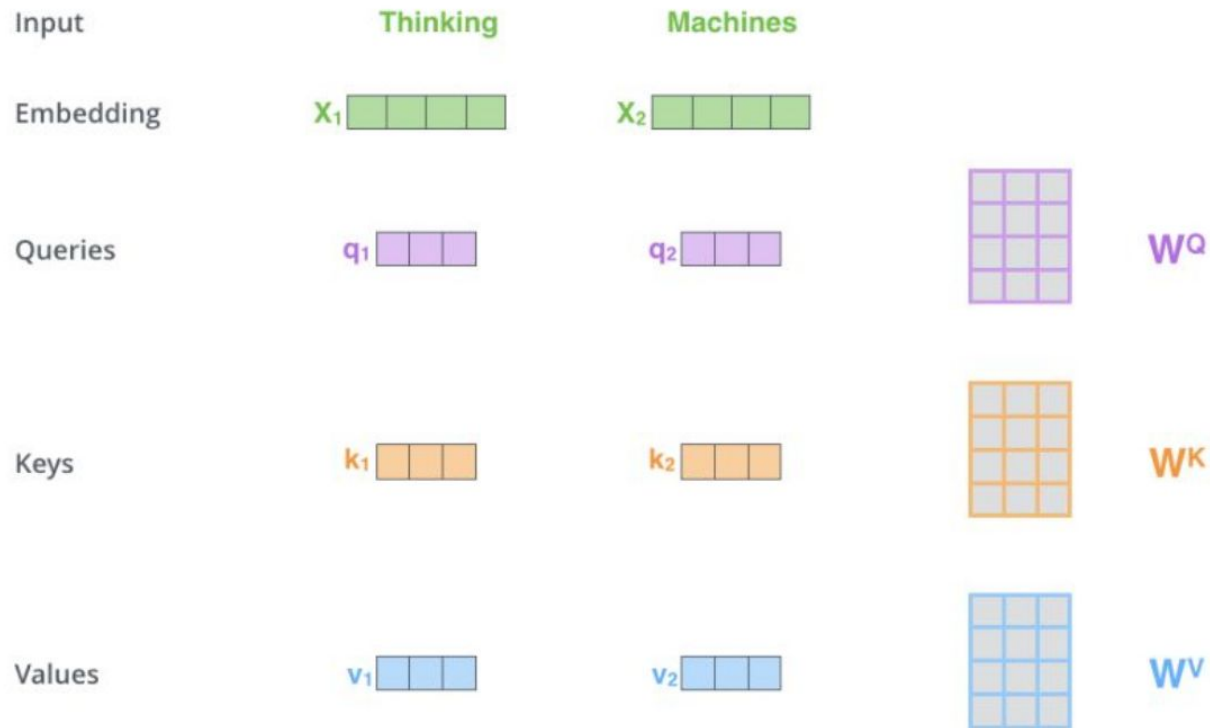
Transformer



Self-Attention



Self-Attention



Self-Attention

Input

Embedding

Queries

Keys

Values

Score

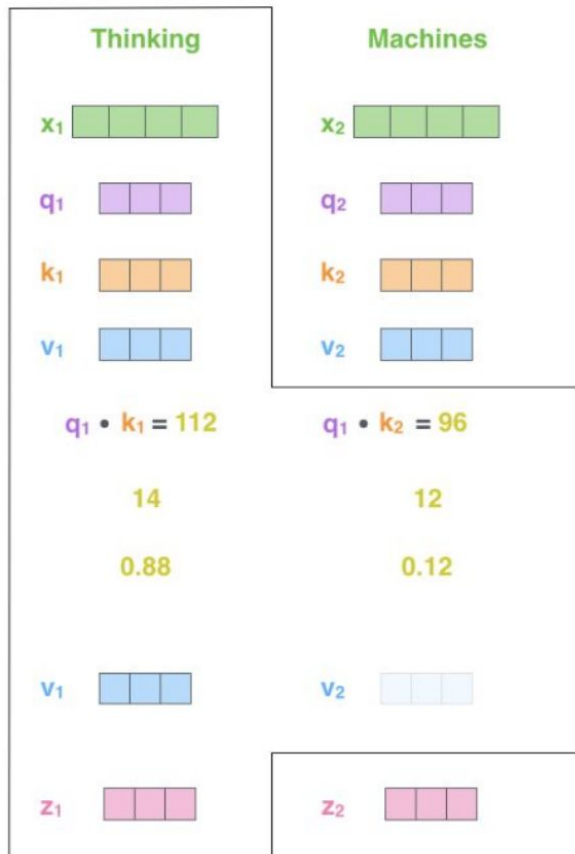
Divide by 8 ($\sqrt{d_k}$)

Softmax

Softmax

X
Value

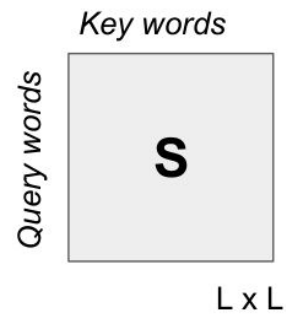
Sum



Self-Attention

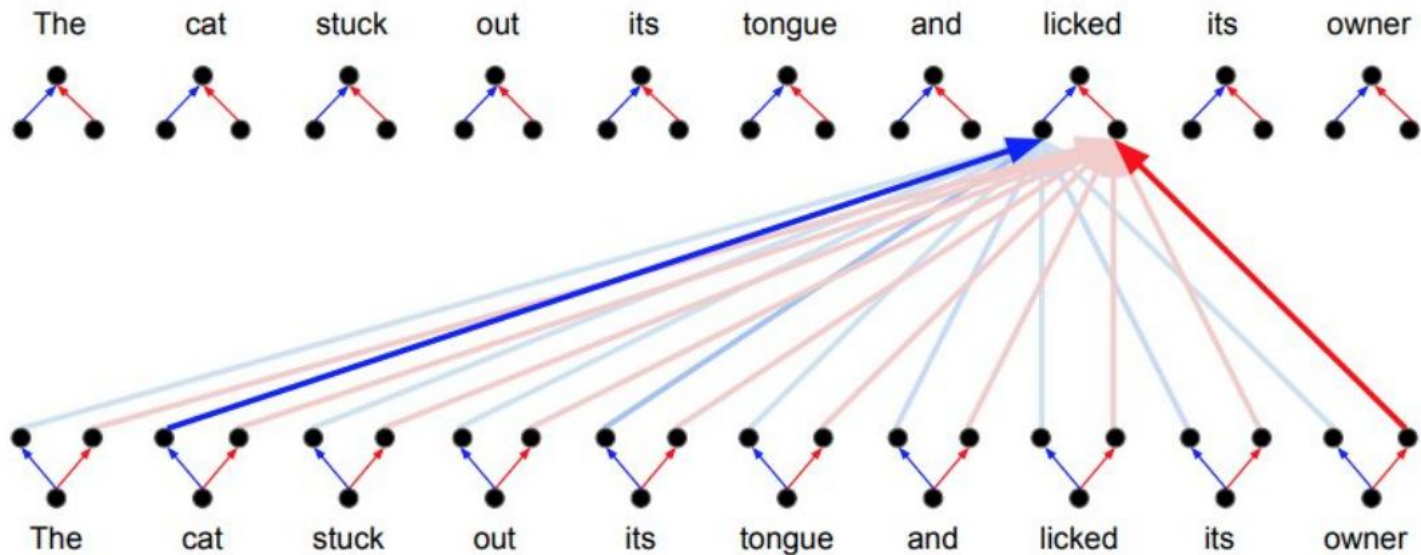


$$A(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

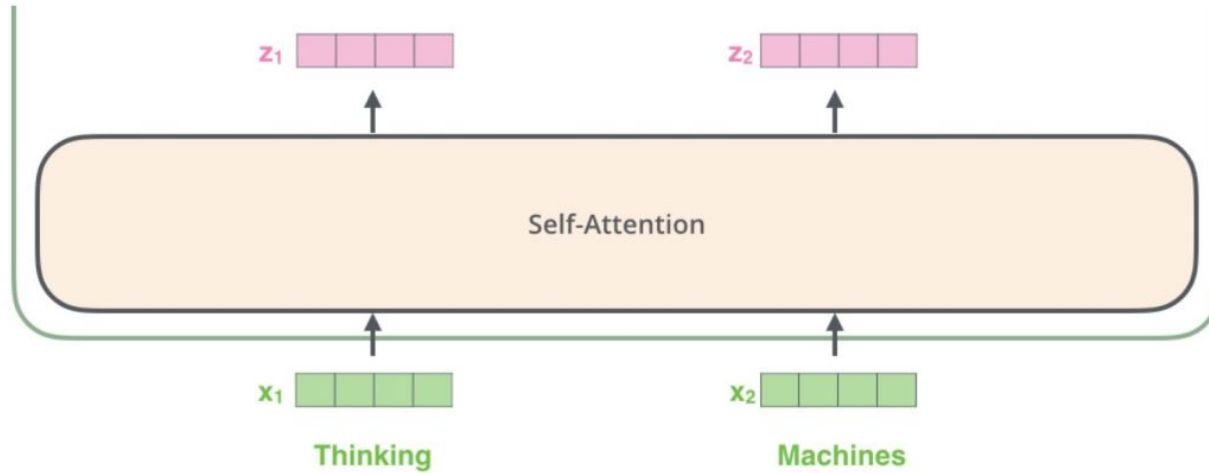


Multi-Head Self-Attention

Parallel attention layers with different linear transformations on input and output.



Retaining Hidden State Size



Details of Each Attention Sub-Layer of Transformer Encoder

1) This is our input sentence*

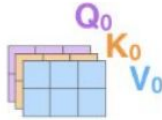
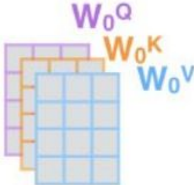
2) We embed each word*

3) Split into 8 heads. We multiply X or R with weight matrices

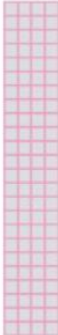
4) Calculate attention using the resulting $Q/K/V$ matrices

5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

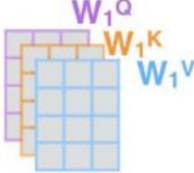
Thinking Machines



W^O



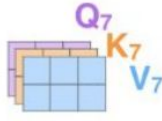
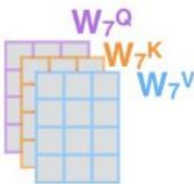
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



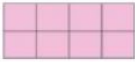
...

...

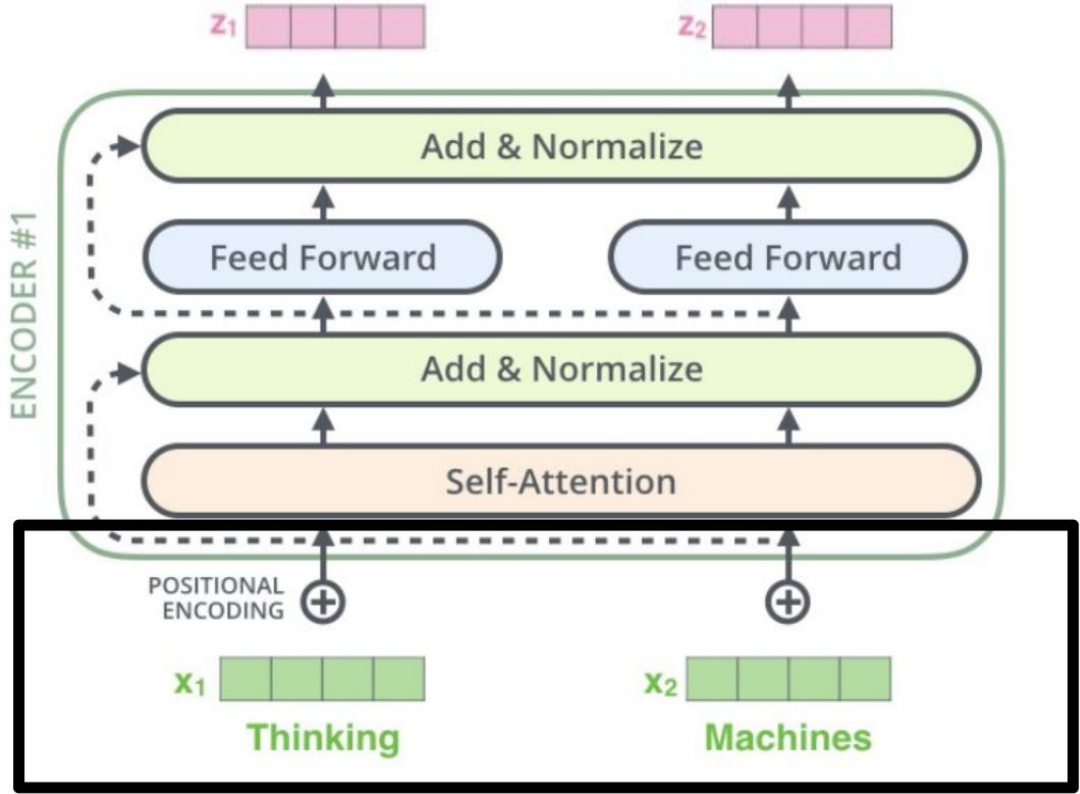
...



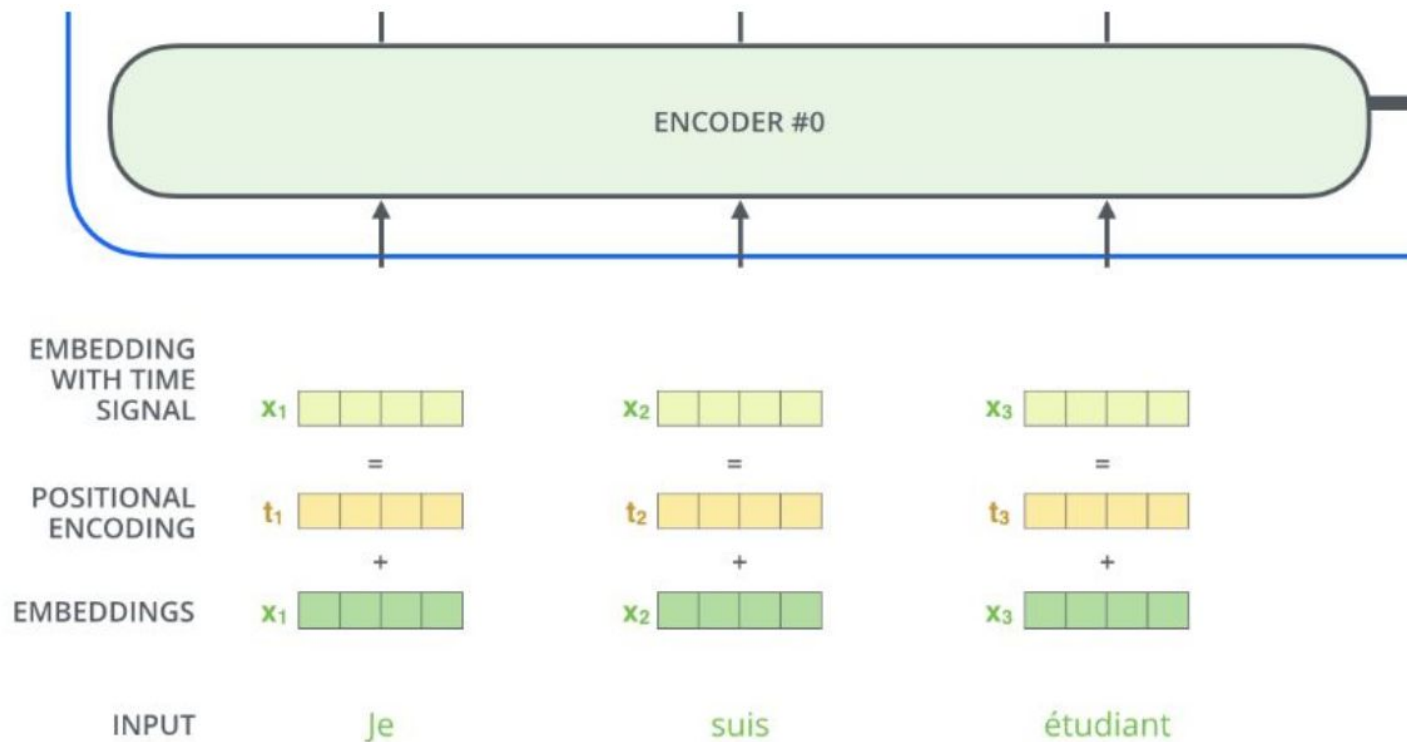
Z



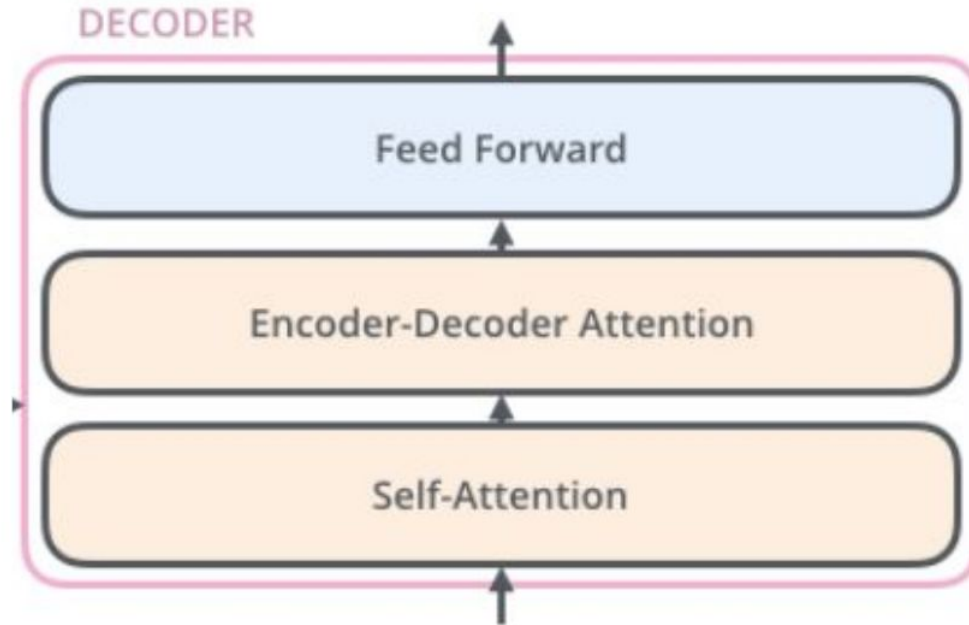
Each Layer of Transformer Encoder



Positional Encoding

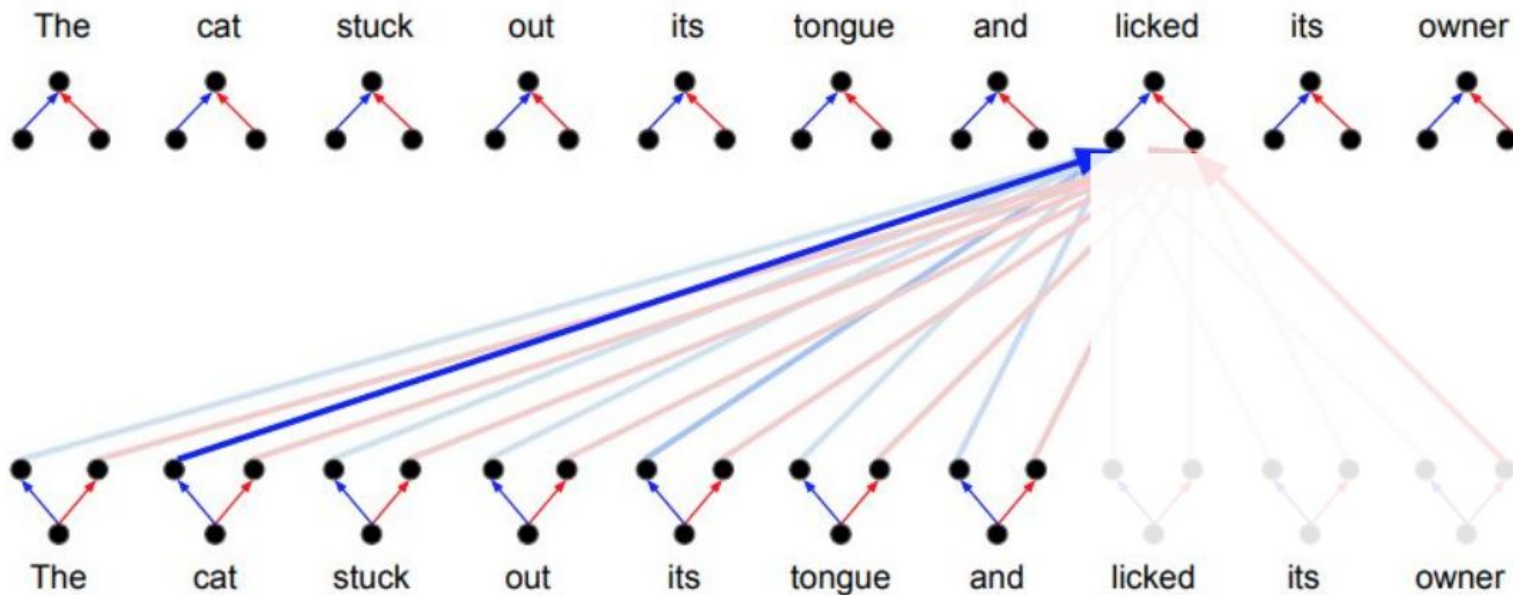


Each Layer of Transformer Decoder



Transformer Decoder - Masked Multi-Head Attention

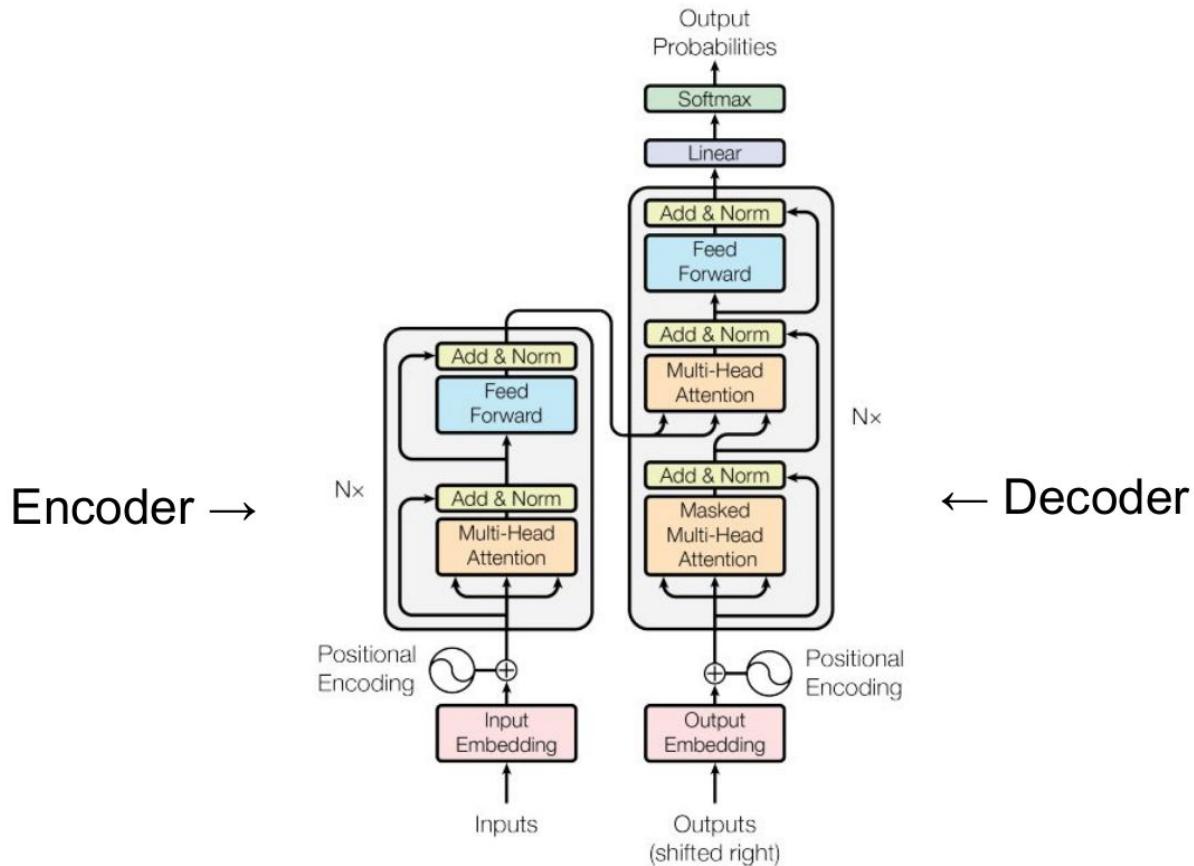
Problem of Encoder self-attention: we can't see the future !



“Attention is All You Need”

(Vaswani et. al 2017)

Transformer



Thank you!