Introduction to Deep Learning

Outline

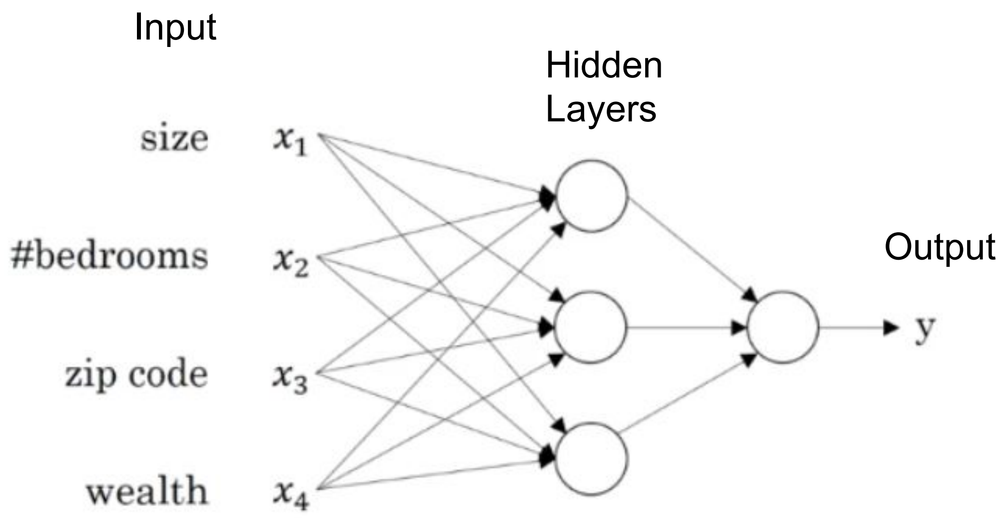
- **Deep Learning**
  - **RNN**
  - **CNN**
  - **Attention**
  - **Transformer**
- Pytorch
  - Introduction
  - Basics
  - Examples

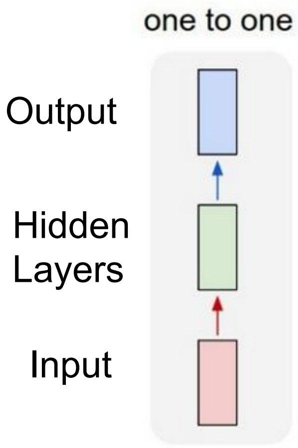
RNNs

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

Vanilla Neural Networks

House Price Prediction



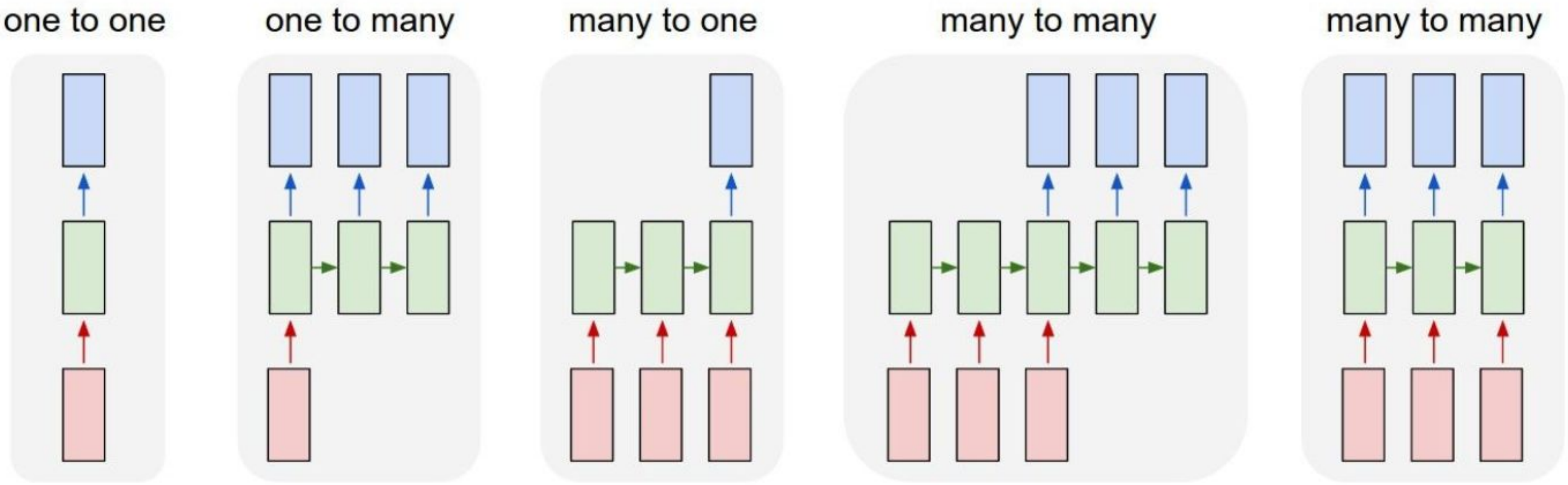


Vanilla
Neural
Networks

How to model sequences?

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

RNN- Recurrent Neural Networks



Vanilla Neural Networks

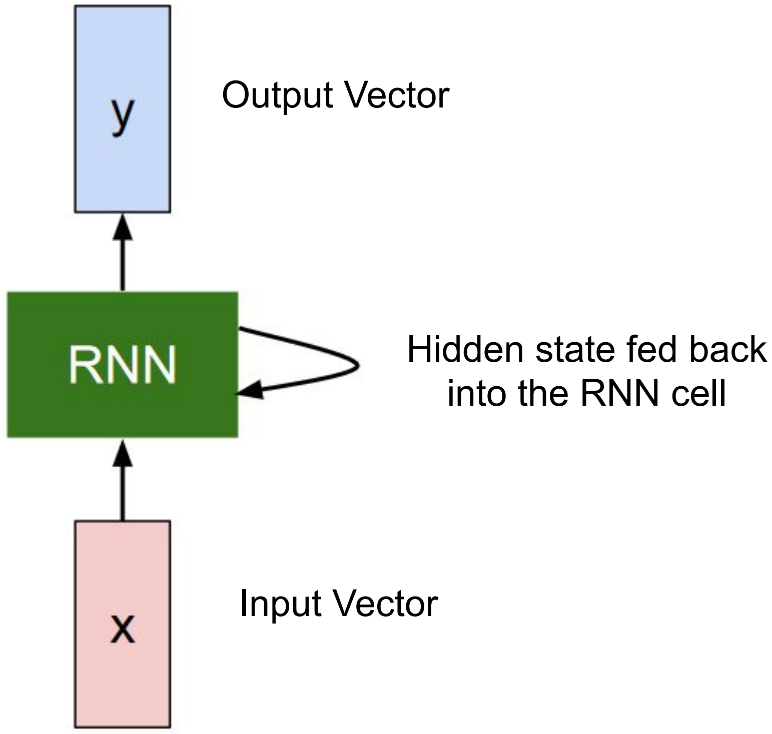
e.g.- Image Captioning

e.g.- Text Classification

e.g.- Translation

e.g.- POS tagging

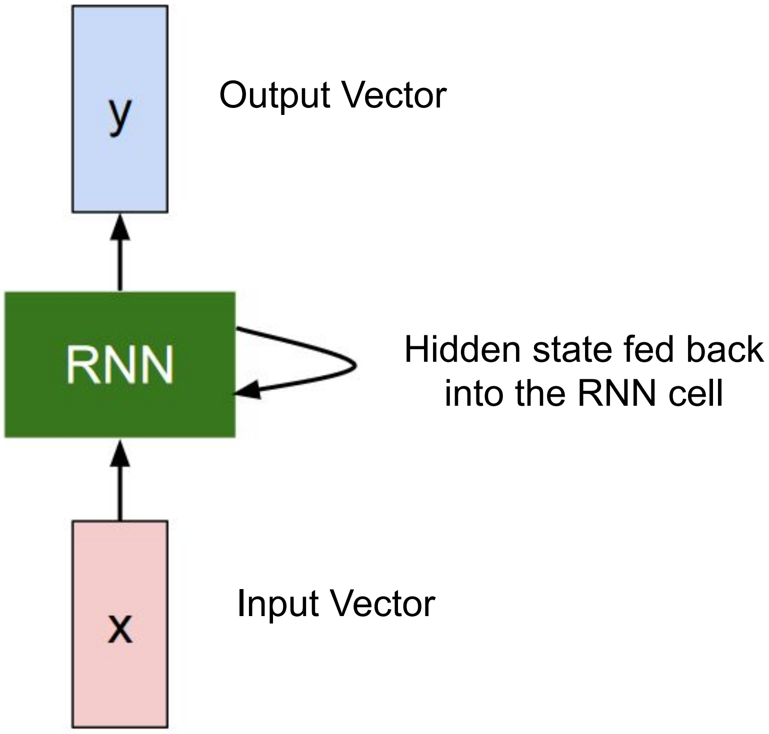
RNN- Representation



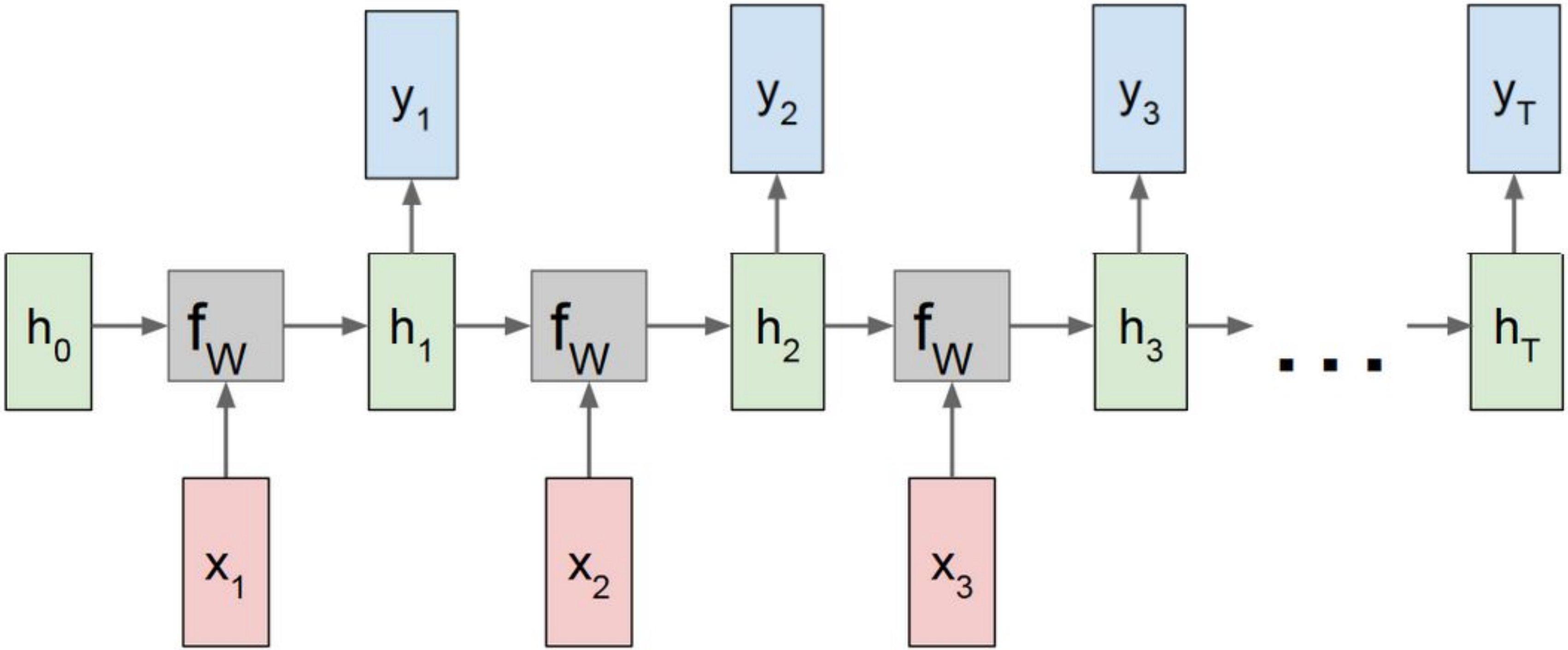
RNN- Recurrence Relation

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.

$$\underbrace{\boxed{h_t}}_{\text{new state}} = \underbrace{\boxed{f_W}}_{\text{some function with parameters W}}(\underbrace{\boxed{h_{t-1}}}_{\text{old state}}, \underbrace{\boxed{x_t}}_{\text{input vector at some time step}})$$

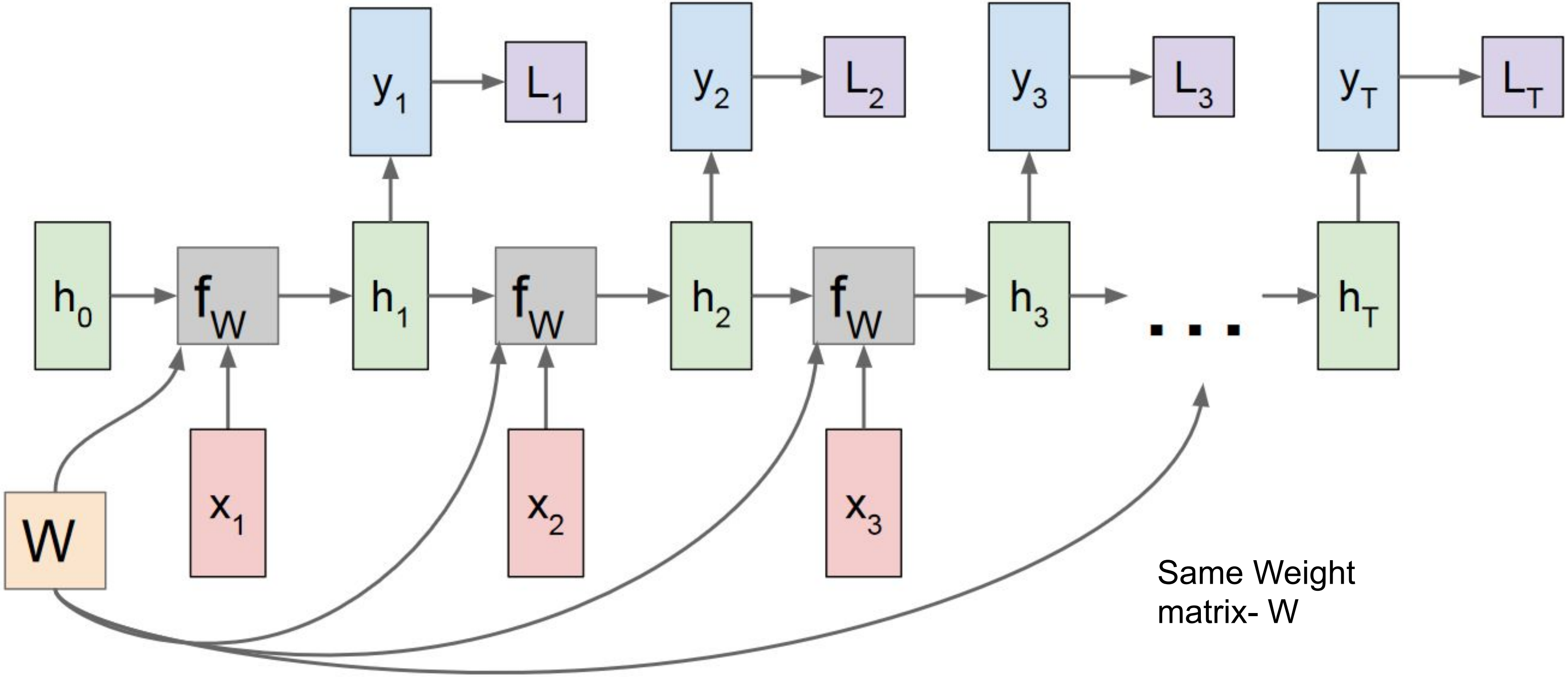


RNN- Rolled out representation



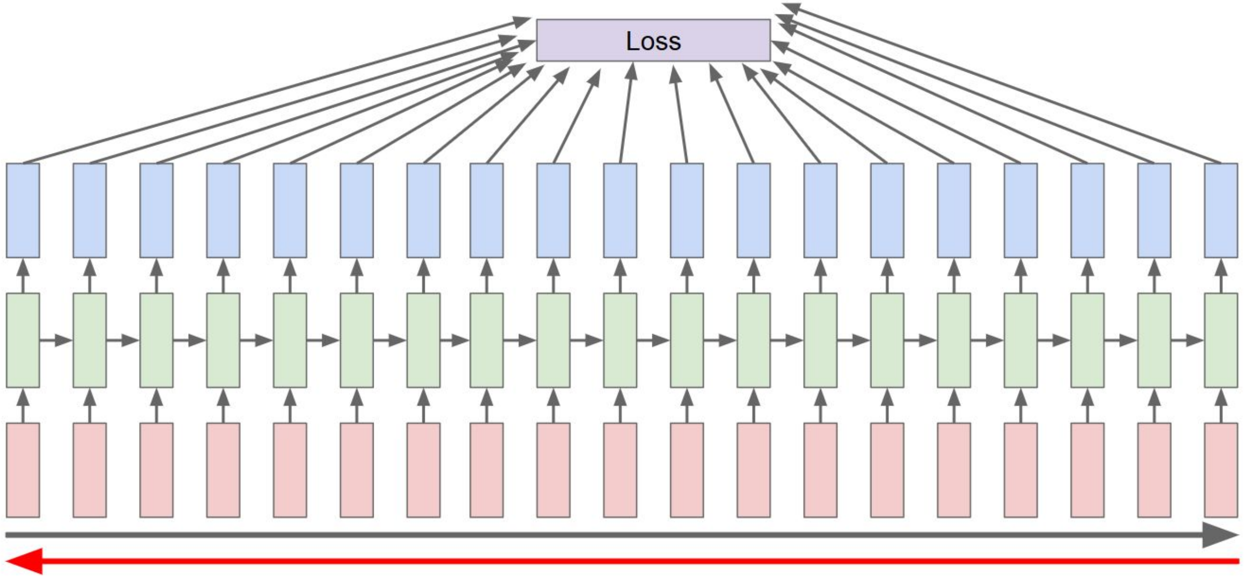
RNN- Rolled out representation

Individual Losses Li



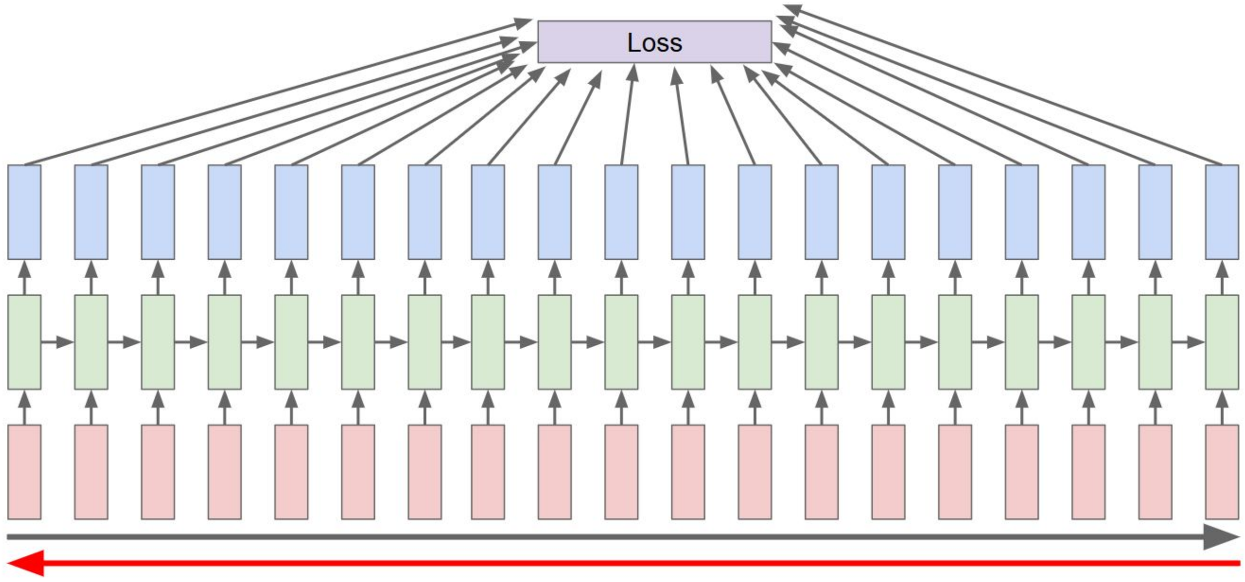
Same Weight
matrix- W

RNN- Backpropagation Through Time



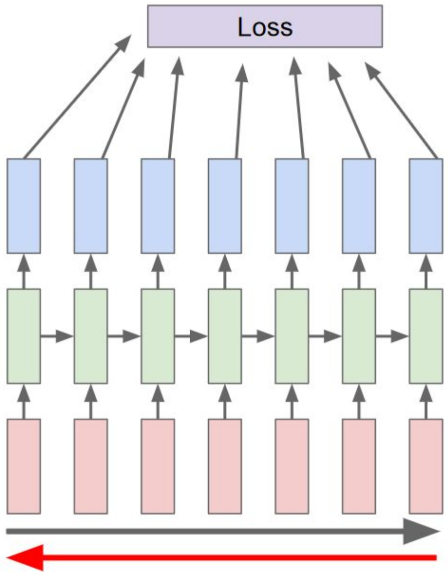
Forward pass through entire sequence to produce intermediate hidden states, output sequence and finally the loss. Backward pass through the entire sequence to compute gradient.

RNN- Backpropagation Through Time



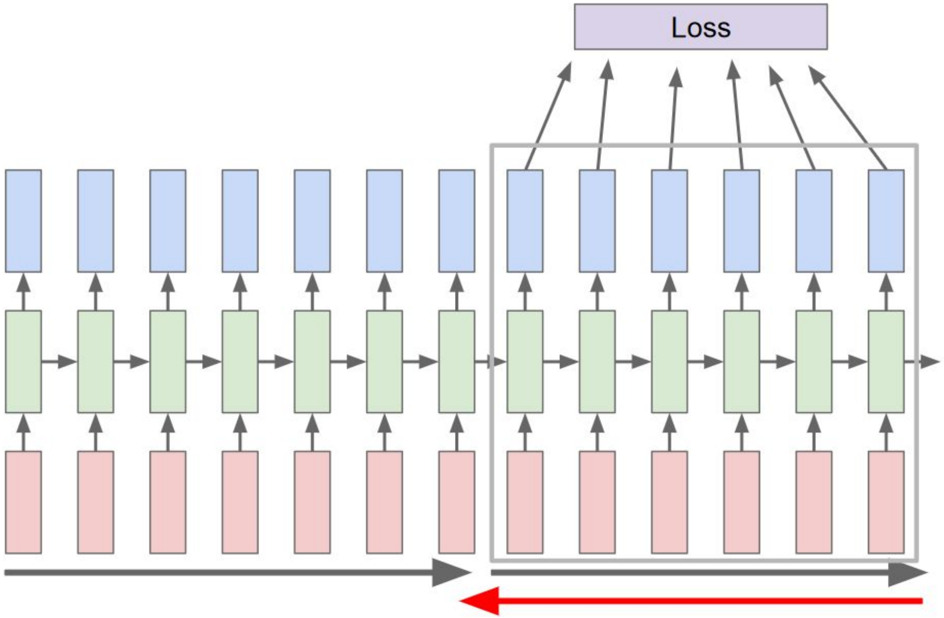
Running Backpropagation through time for the entire text would be very slow. Switch to an approximation- Truncated Backpropagation Through Time

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

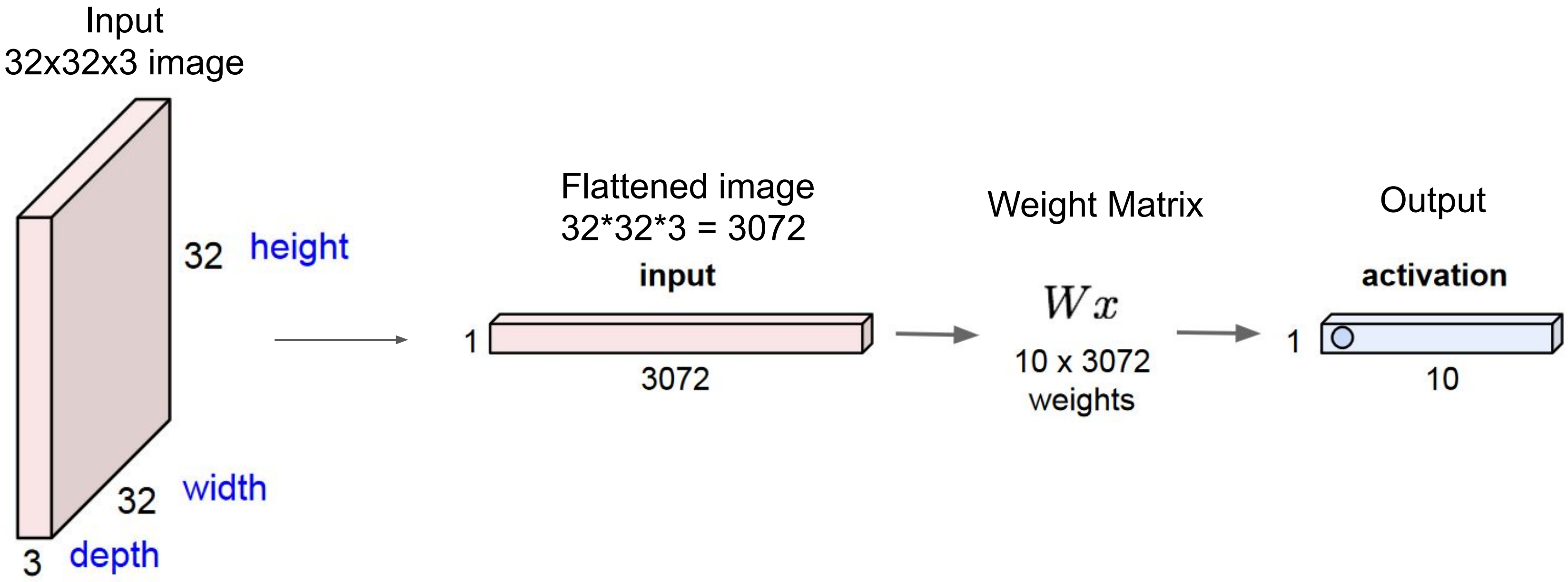
Recurrent Neural Network Tutorial, Part 4 – Implementing a GRU/LSTM RNN with Python and Theano

Stanford CS231n: Lecture 10 | Recurrent Neural Networks

CNNs

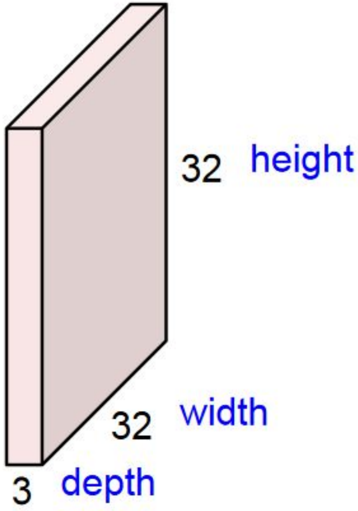
Some slides borrowed from Fei-Fei Li & Justin Johnson & Serena Yeung at Stanford.

Fully Connected Layer

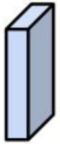


Convolutional Layer

Input
32x32x3 image



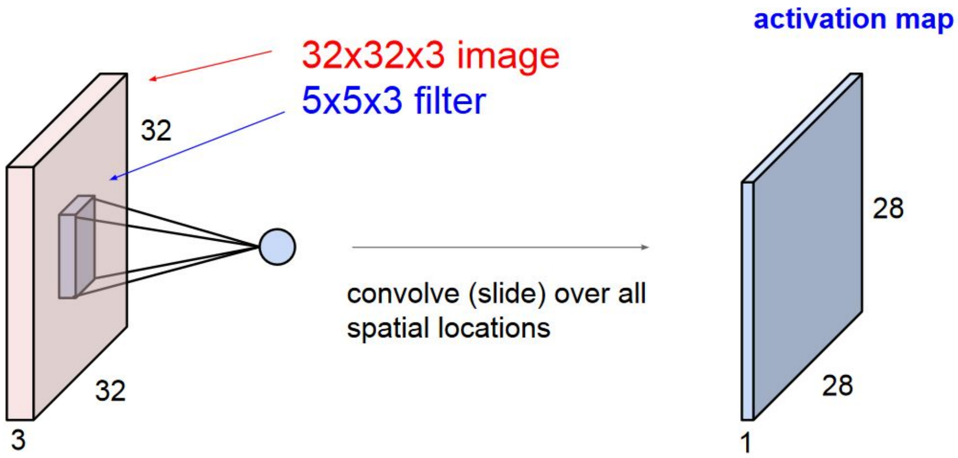
Filter
5x5x3



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

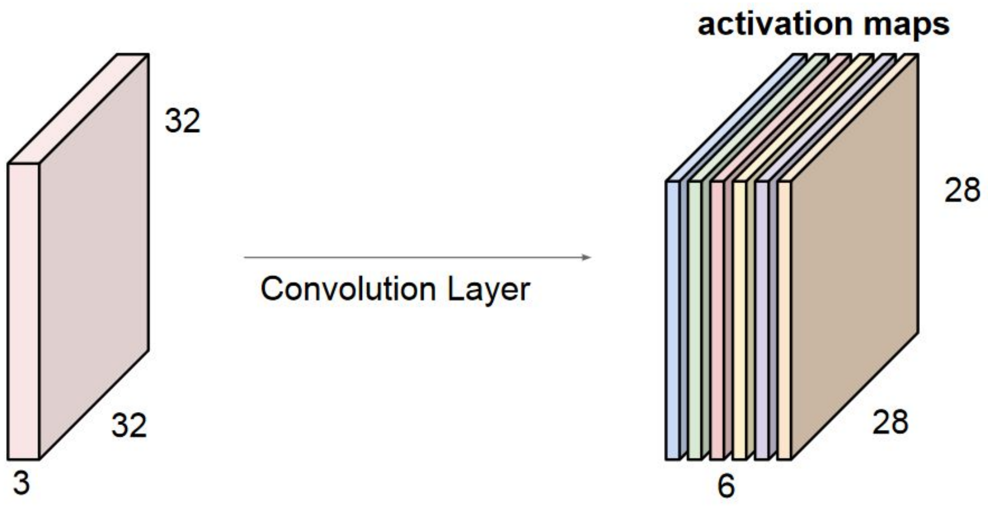


At each step during the convolution, the filter acts on a region in the input image and results in a single number as output.

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.

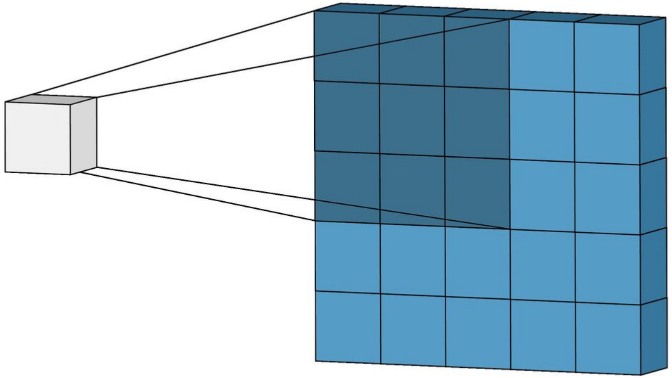
Convolutional Layer

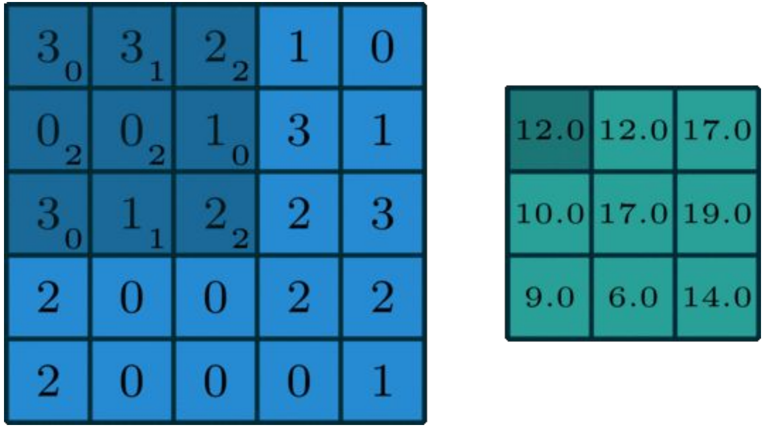


Filters can be stacked together.

Example- If we had 6 filters of shape 5x5,
each would produce an activation map of 28x28x1 and our output would be a “new image” of shape 28x28x6.

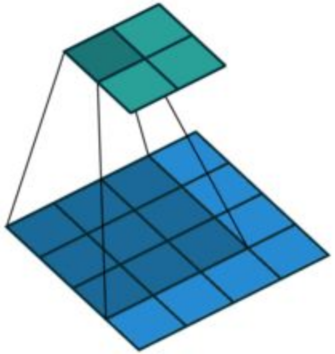
Convolutional Layer



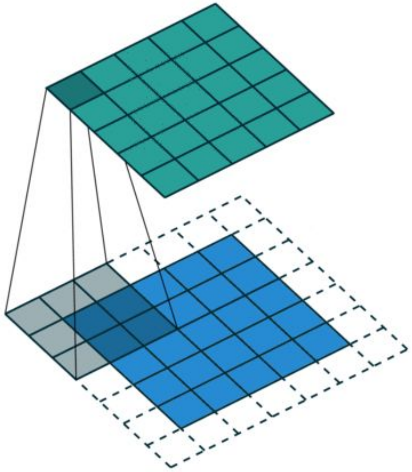


Visualizations borrowed from Irhum Shafkat's blog.

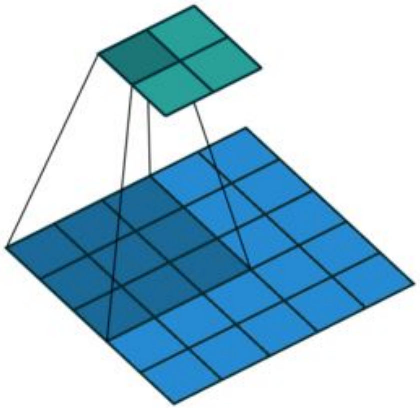
Convolutional Layer



Standard
Convolution



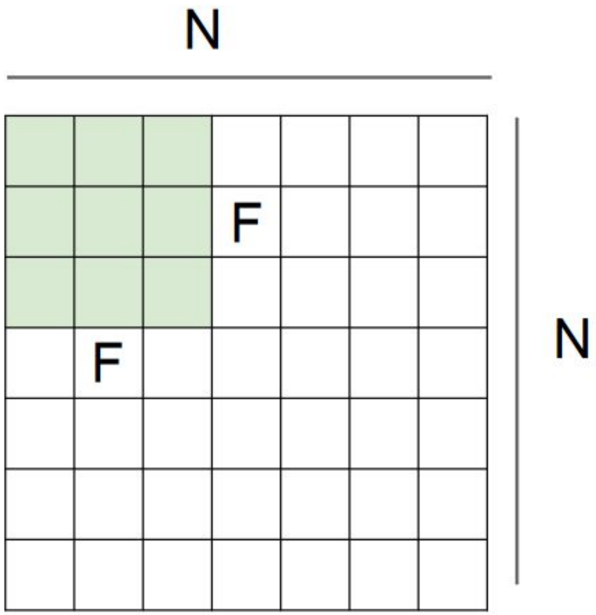
Convolution
with Padding



Convolution
with strides

Visualizations borrowed from vdumoulin's [github repo](#).

Convolutional Layer

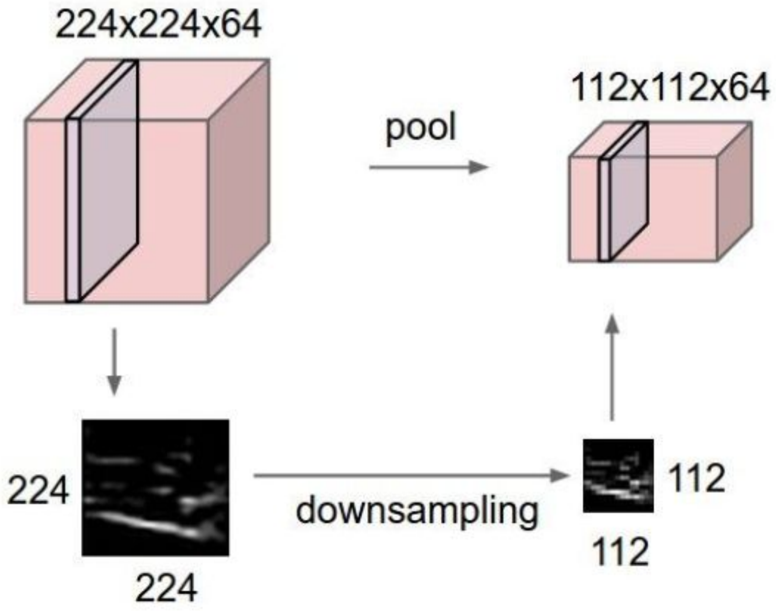


Output Size:
(N - F)/stride + 1

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

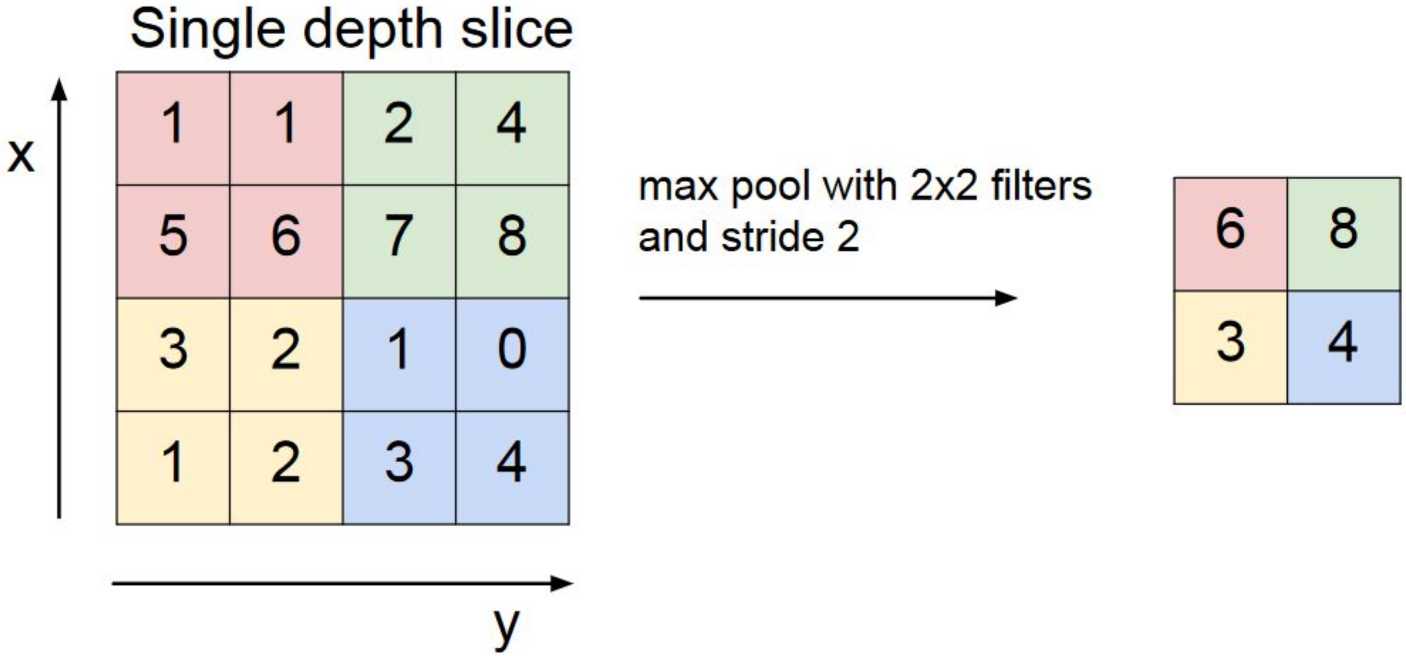
e.g. N = 7, F = 3, stride 2
=> (7 - 3)/2 + 1 = 3

Pooling Layer



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

Max Pooling


Single depth slice
x
y
1
1
2
4
5
6
7
8
3
2
1
0
1
2
3
4
max pool with 2x2 filters
and stride 2
6
8
3
4

ConvNet Layer

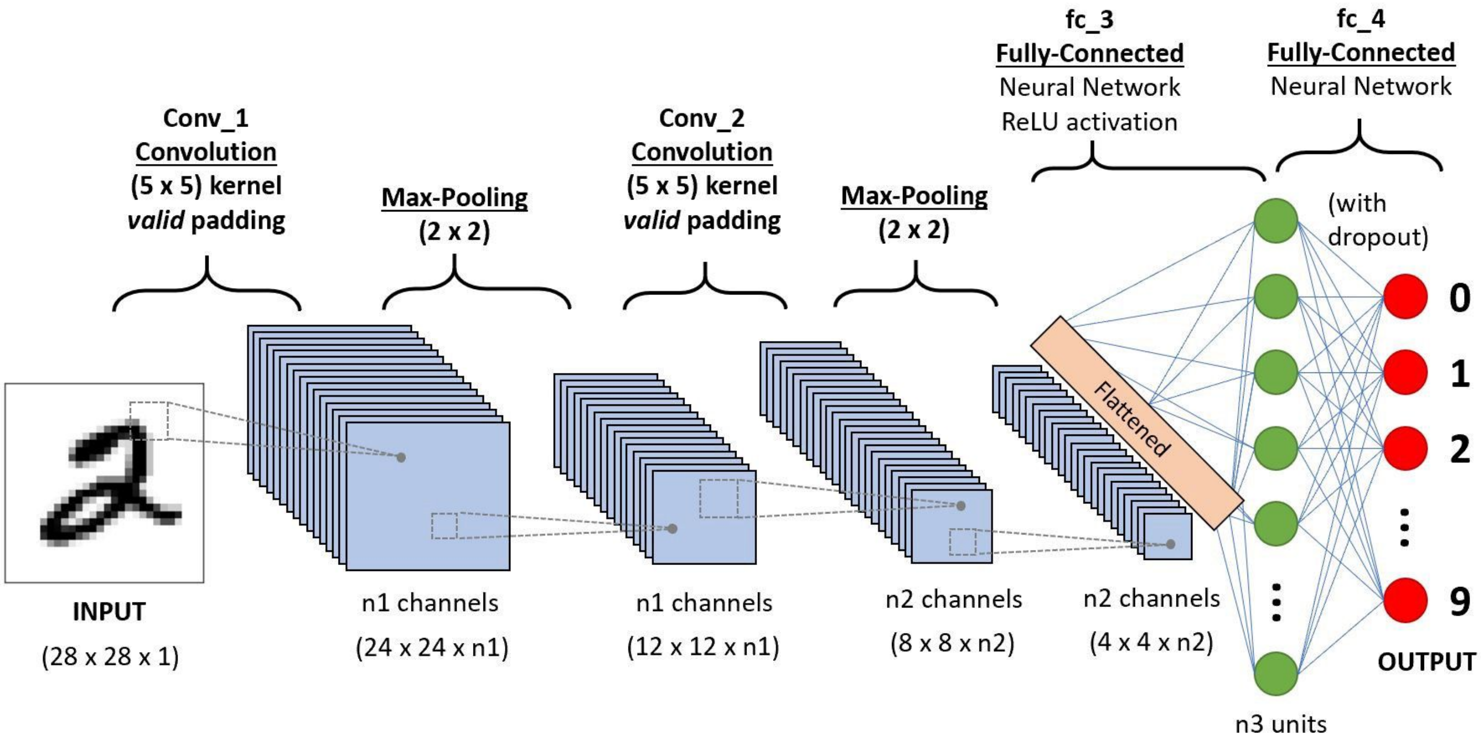


Image credits- Saha's blog.

ConvNet Layer

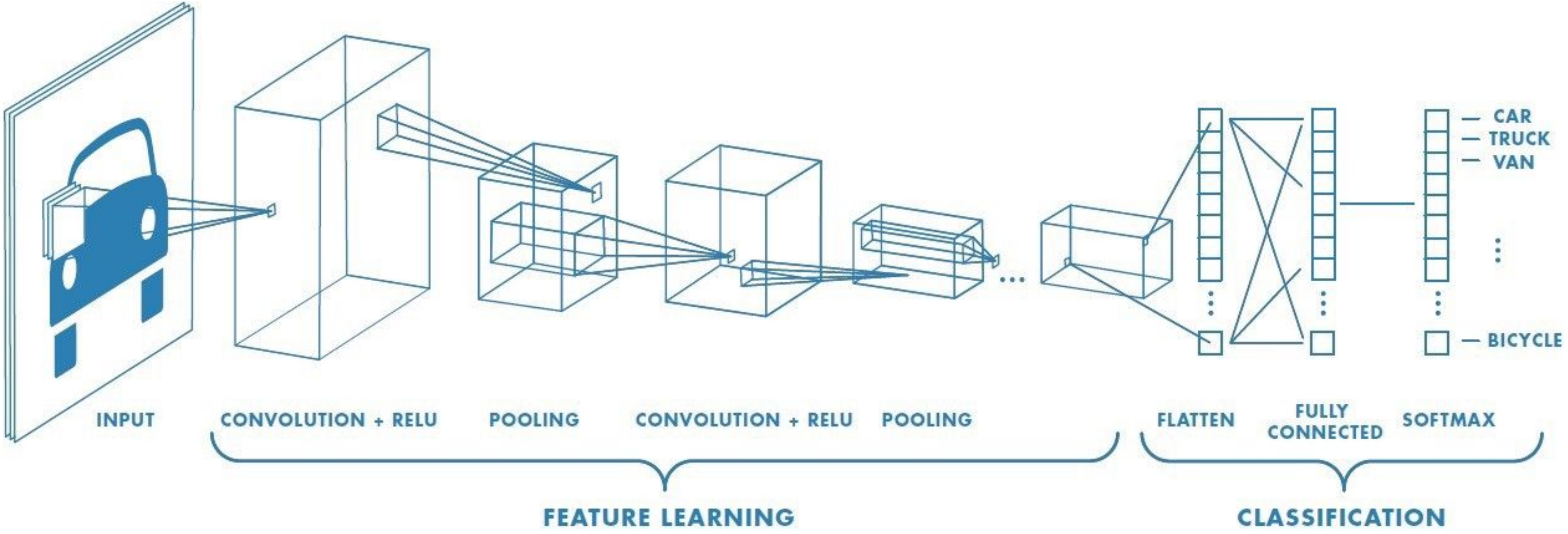
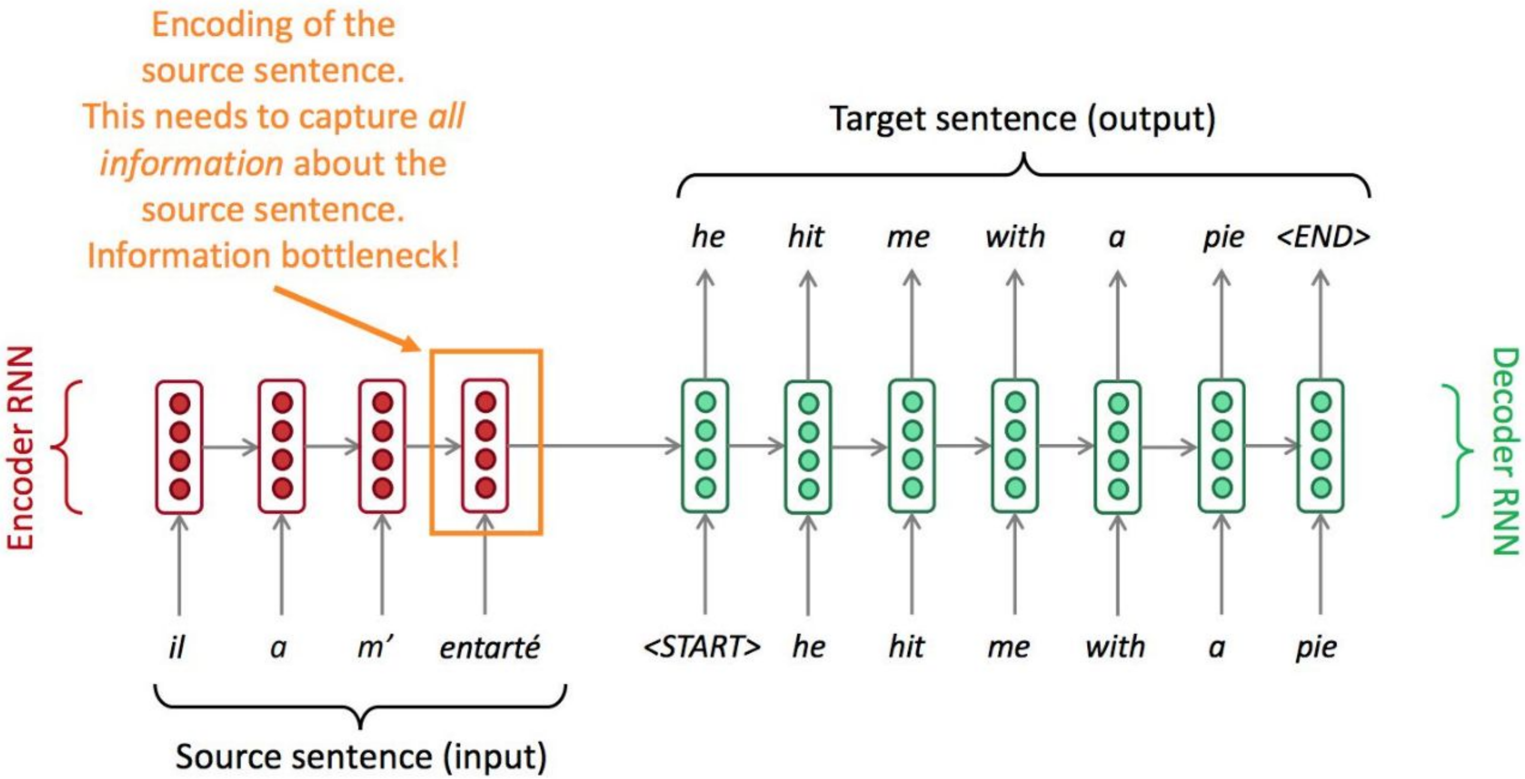


Image credits- Saha's blog.

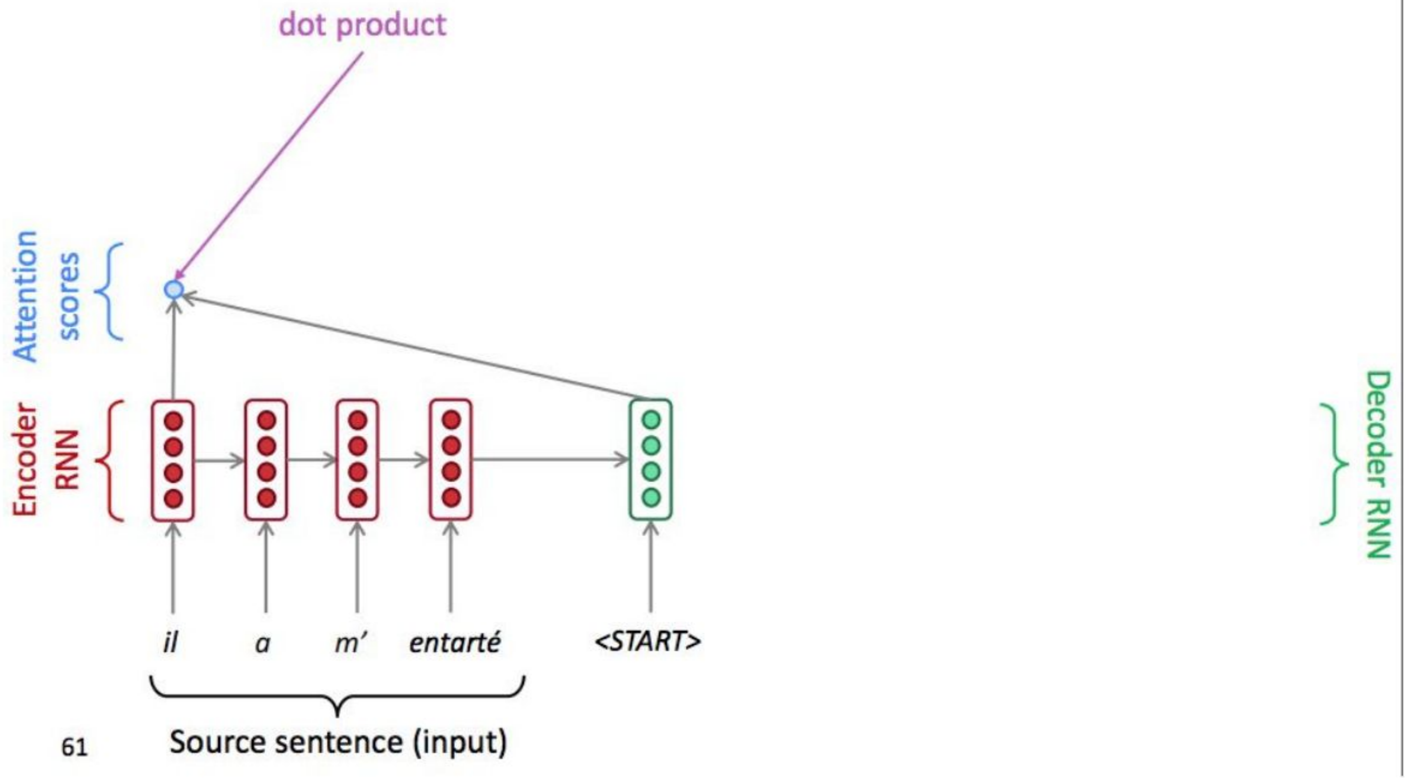
Attention

Some slides borrowed from Sarah Wiegreffe
at Georgia Tech.

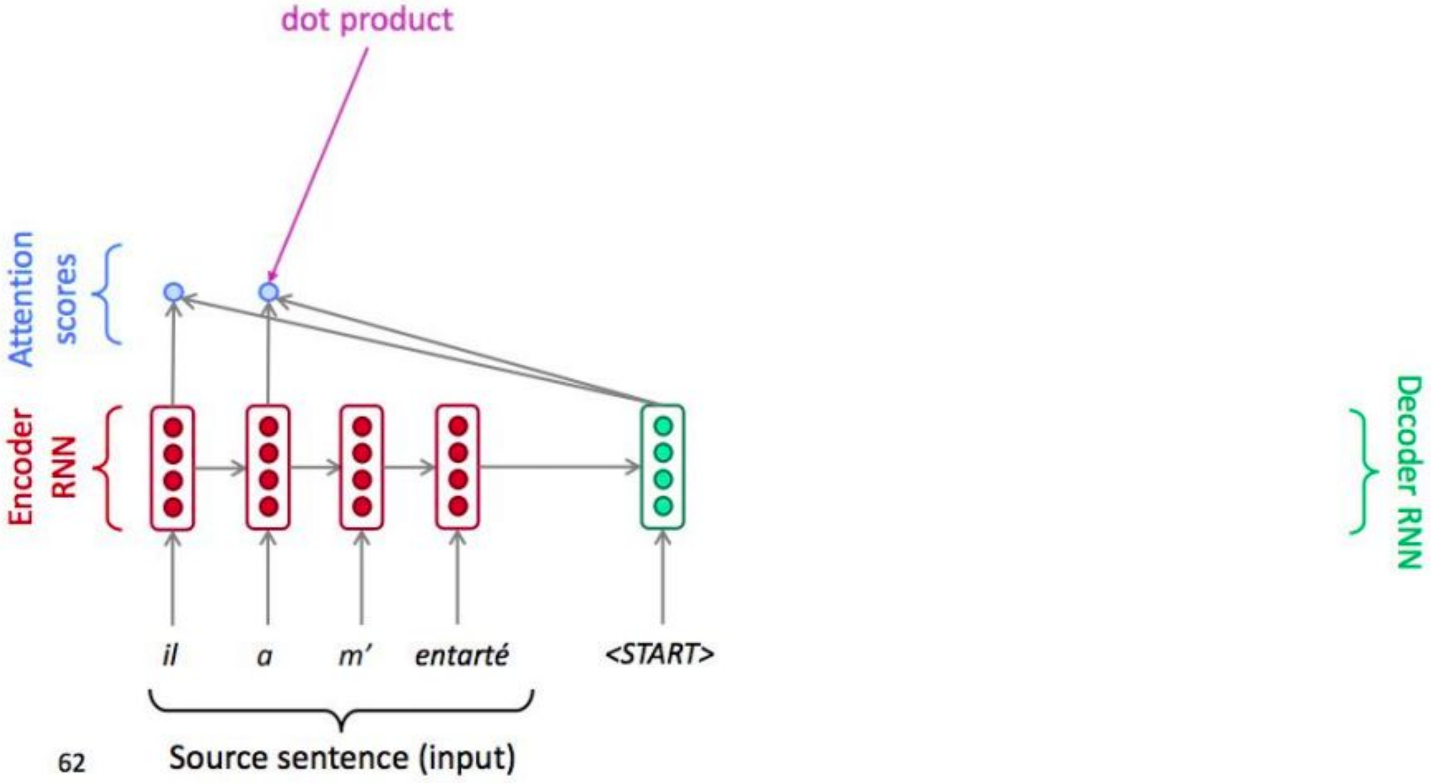
RNN


Encoding of the source sentence. This needs to capture *all information* about the source sentence. Information bottleneck!
Target sentence (output)
he hit me with a pie <END>
Encoder RNN
Decoder RNN
il a m' entarté
<START> he hit me with a pie
Source sentence (input)

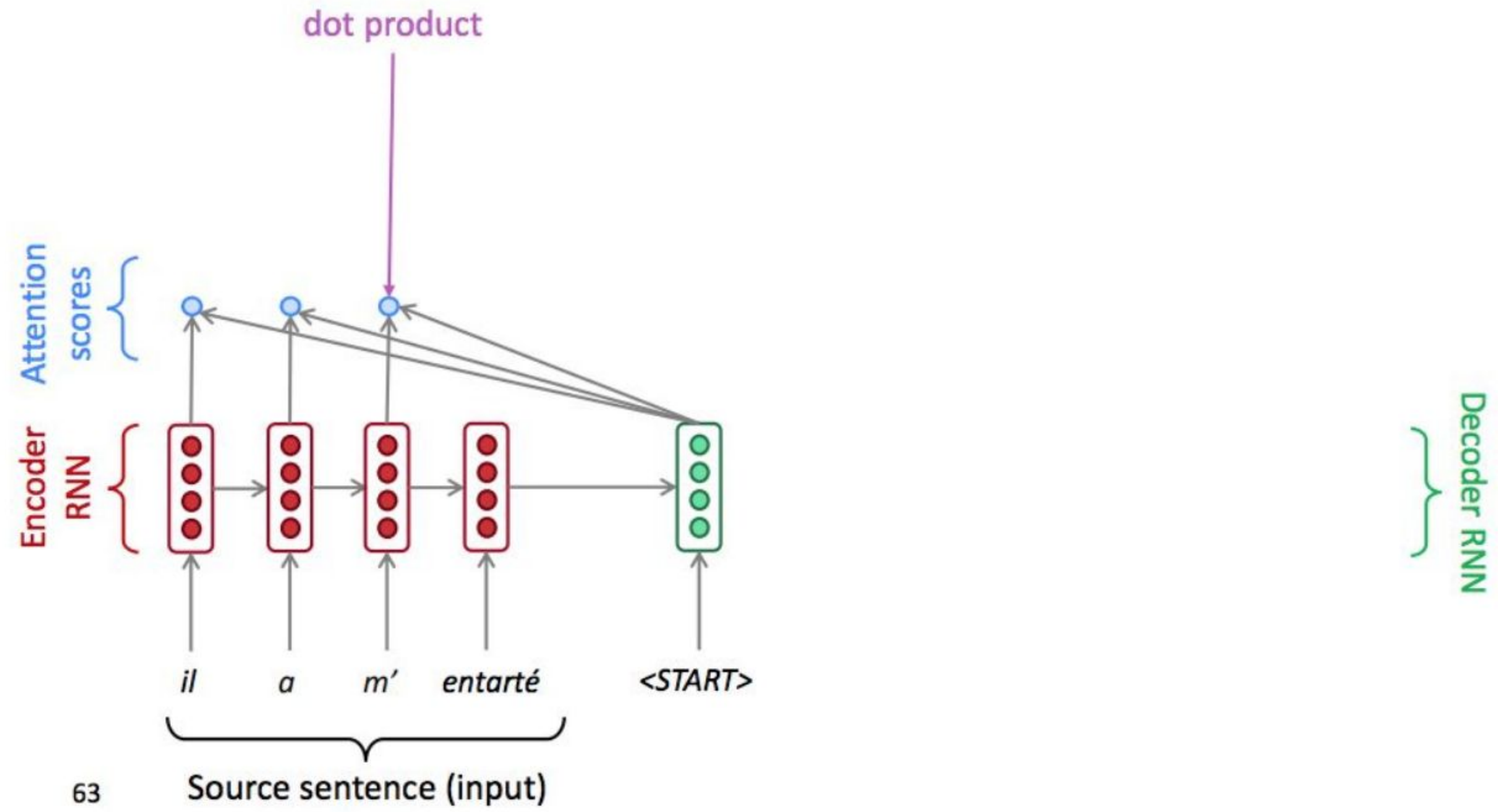
RNN - Attention



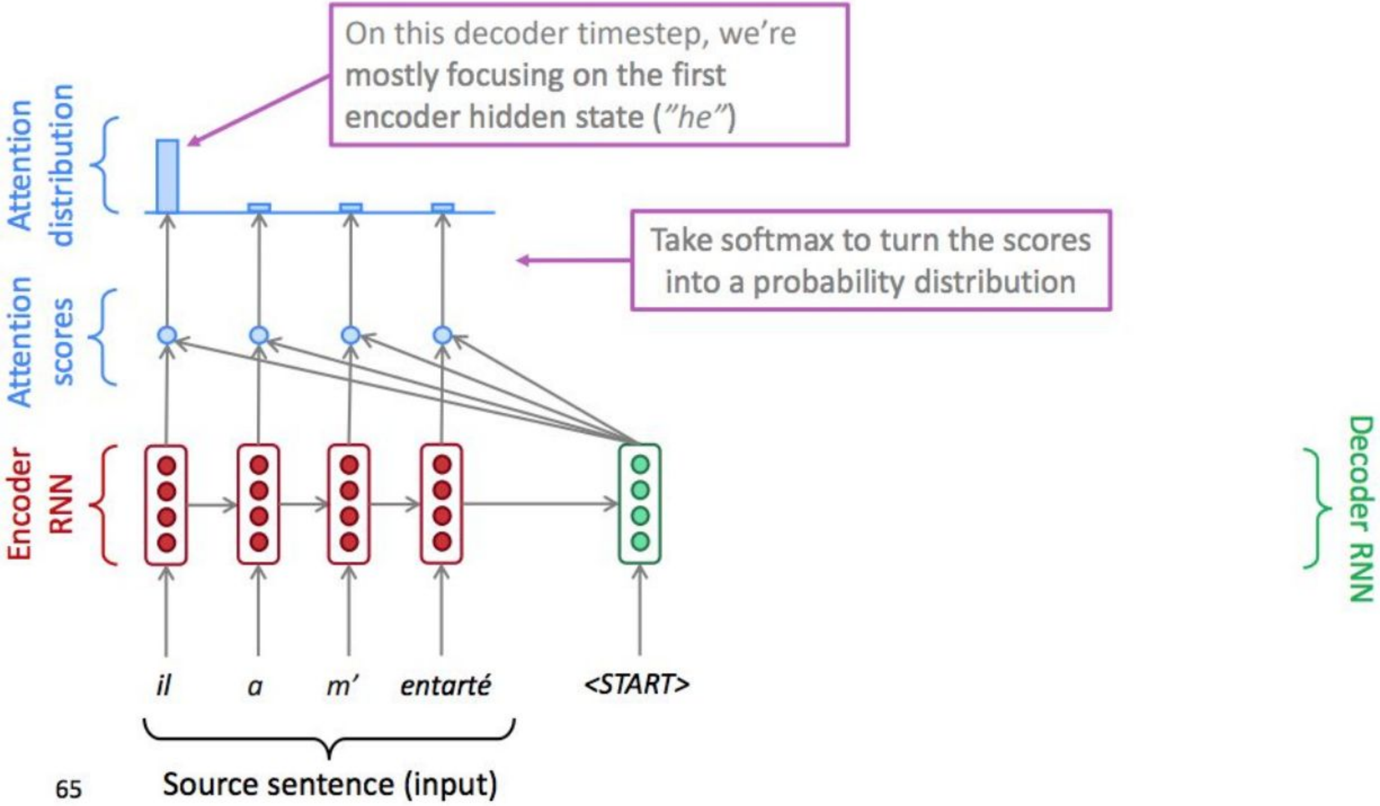
RNN - Attention



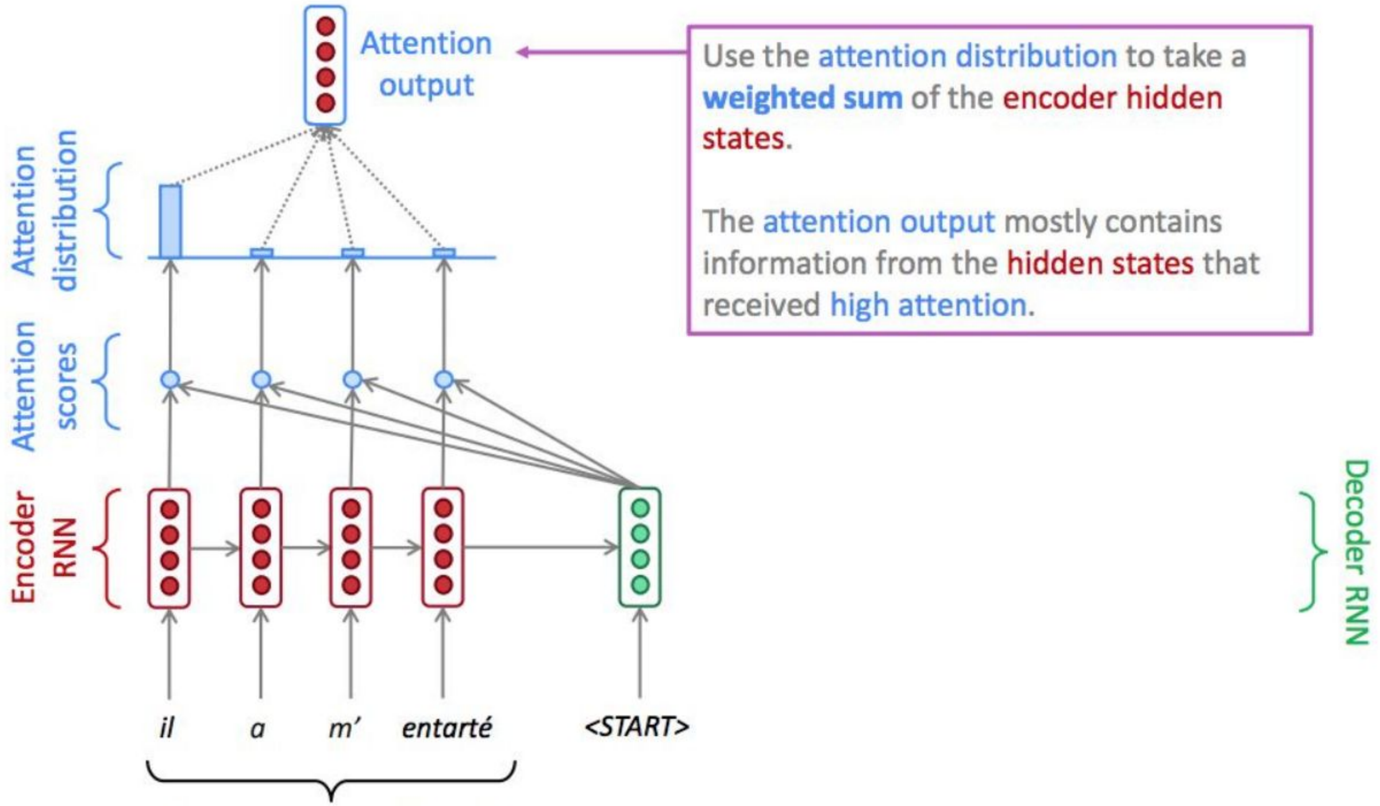
RNN - Attention



RNN - Attention



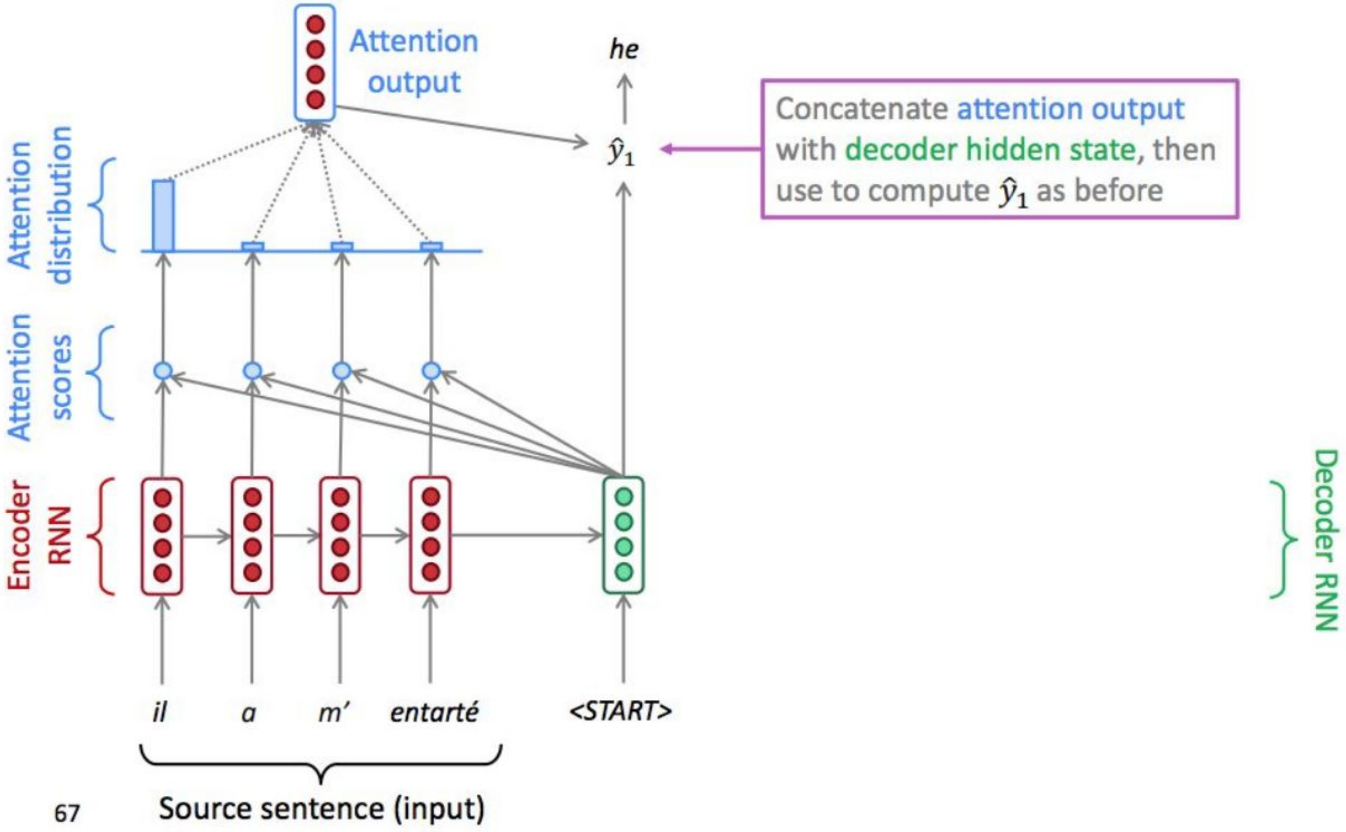
RNN - Attention



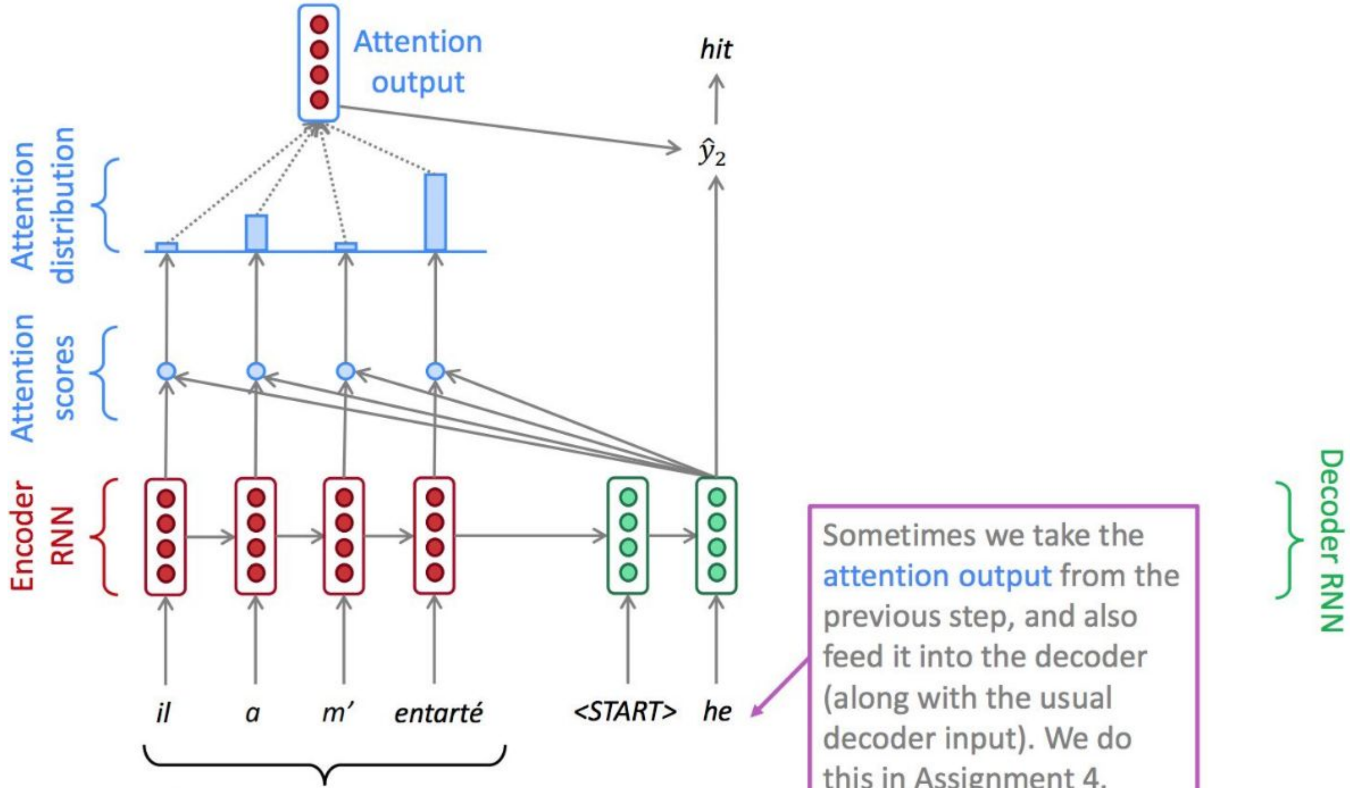
Use the attention distribution to take a **weighted sum** of the encoder hidden states.

The attention output mostly contains information from the hidden states that received high attention.

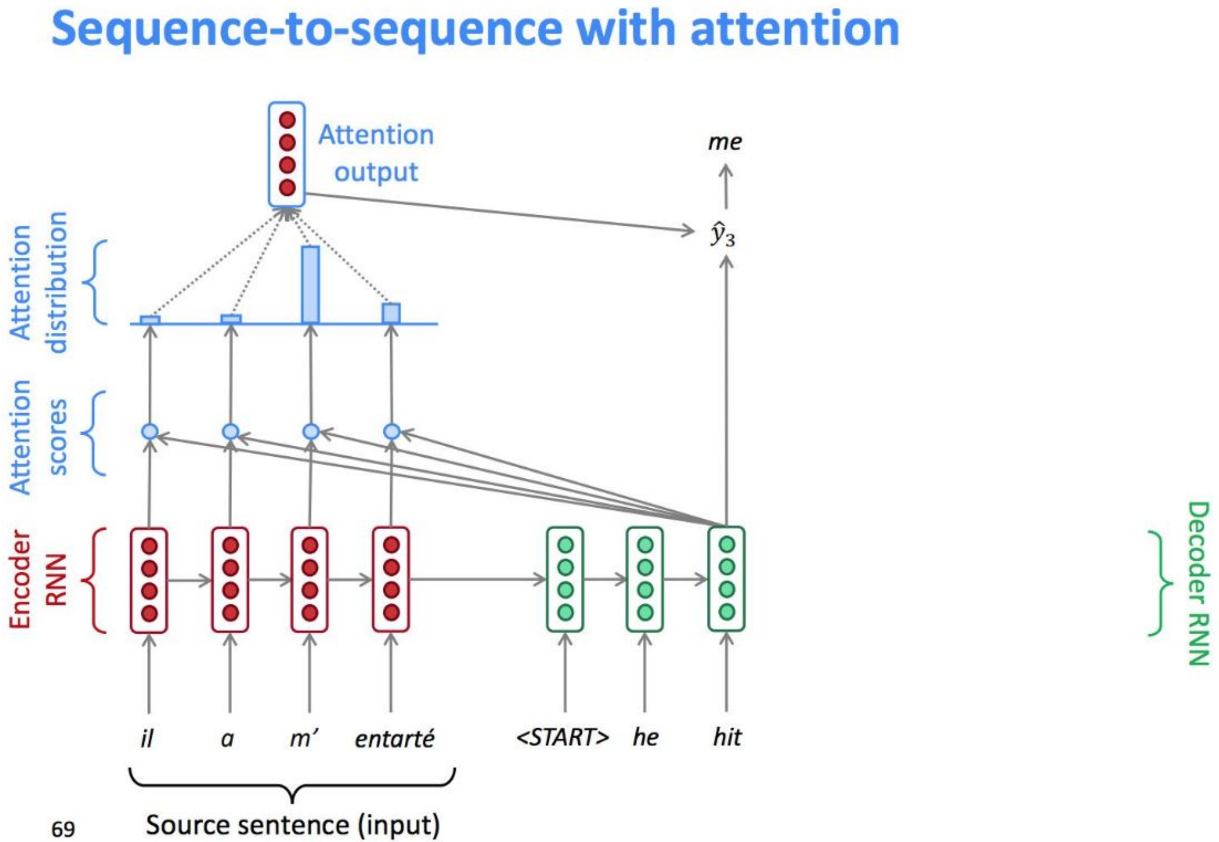
RNN - Attention



RNN - Attention

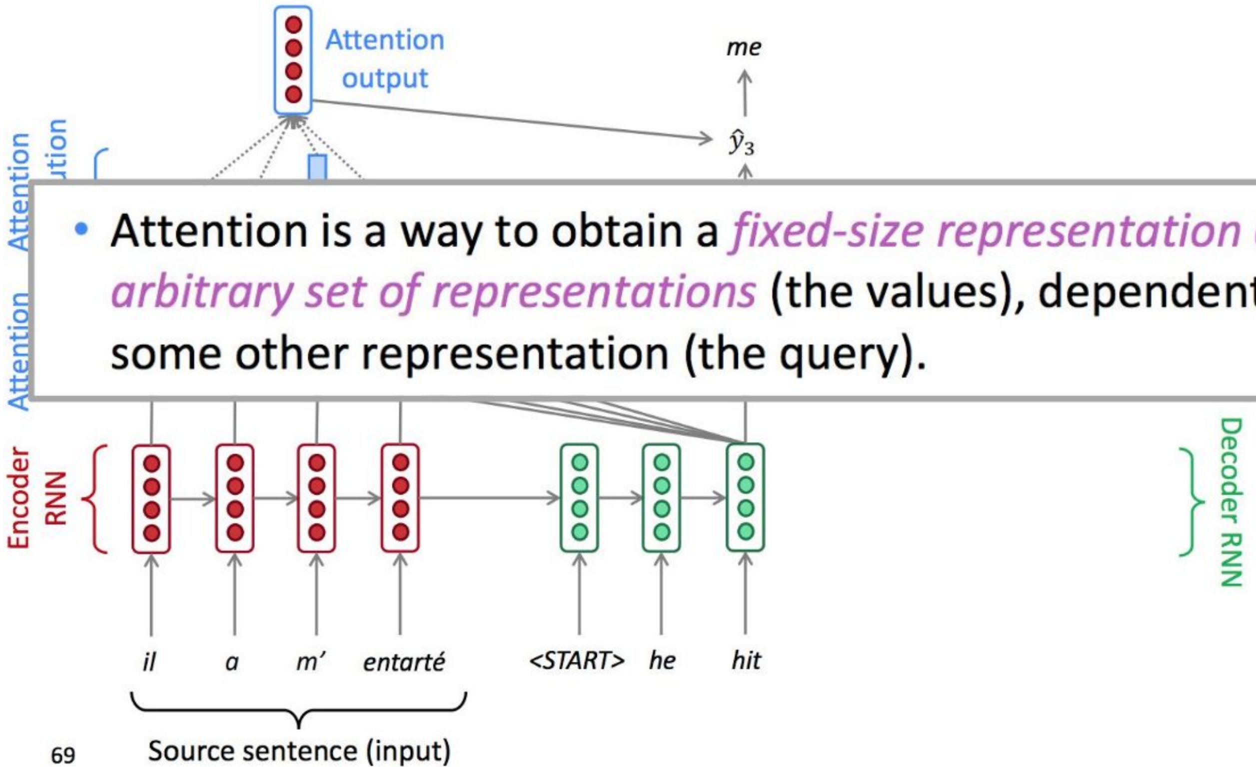


RNN - Attention



RNN - Attention

Sequence-to-sequence with attention



- Attention is a way to obtain a *fixed-size representation of an arbitrary set of representations* (the values), dependent on some other representation (the query).

Attention

- For query vector **q**, key vector $\mathbf{k_i}$ representing value $\mathbf{v_i}$
  - s_i is the similarity score between **q** and $\mathbf{k_i}$
- Normalize the similarity scores to sum to 1
  - p_i = Softmax(s_i)
- Compute **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

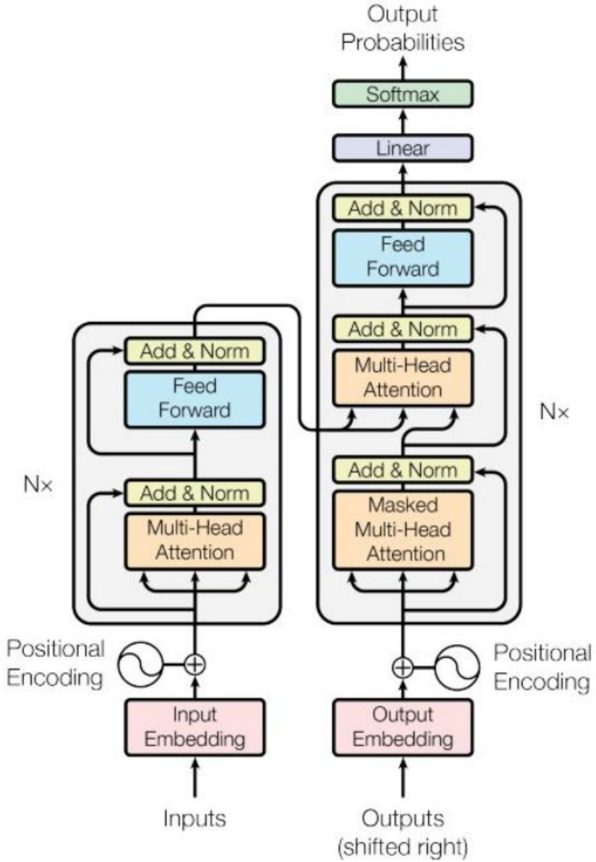
Some slides borrowed from Sarah Wiegreffe
at Georgia Tech.

Transformer

“Attention is All You Need”

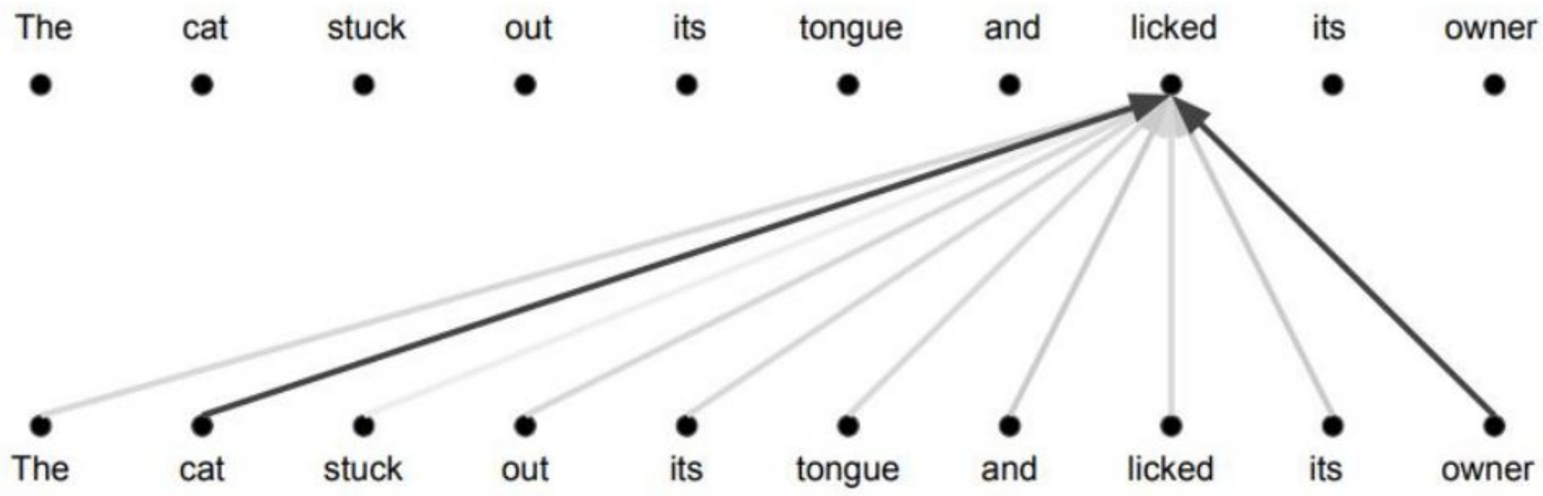
(Vaswani et. al 2017)

Encoder →

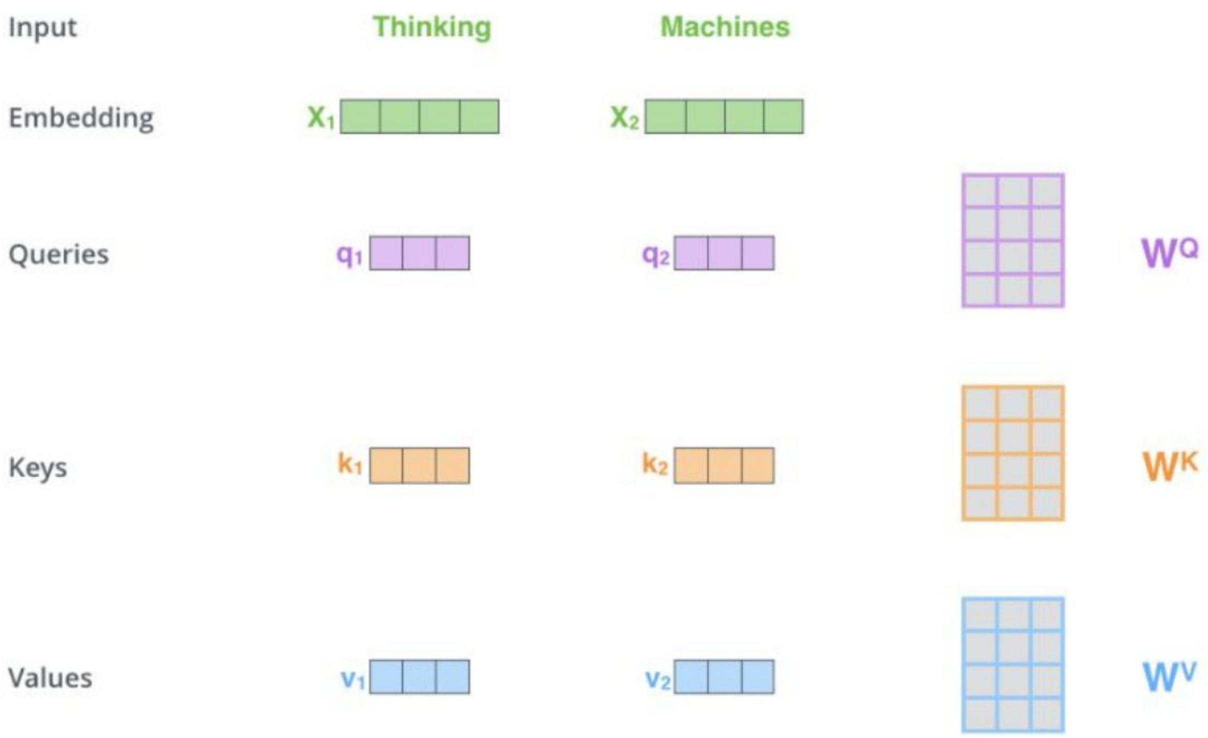


← Decoder

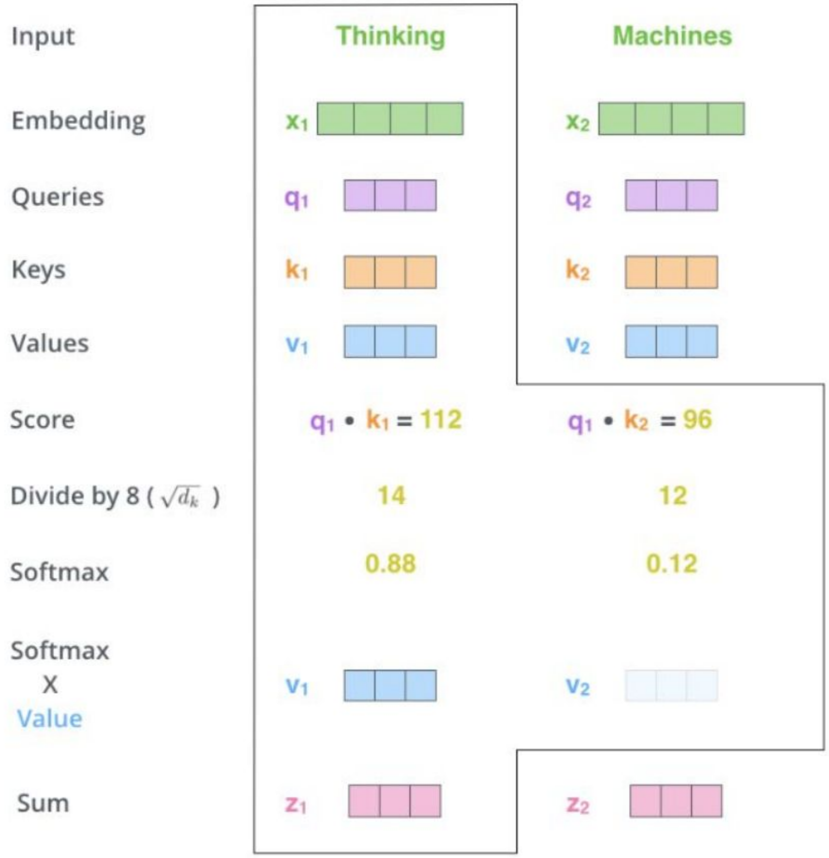
Self-Attention



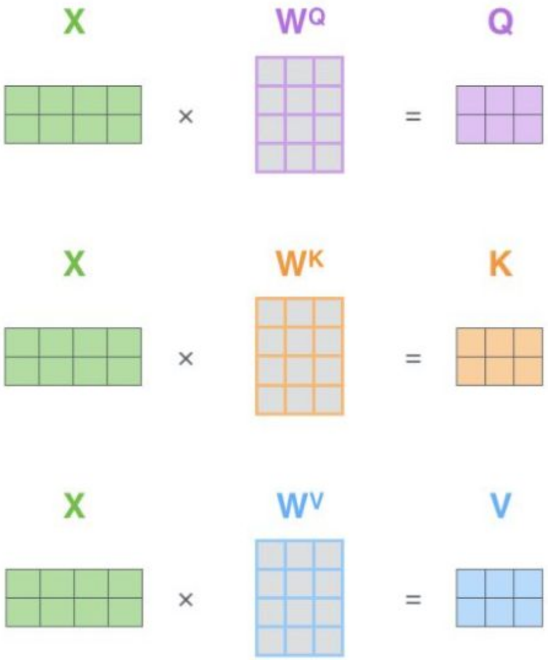
Self-Attention



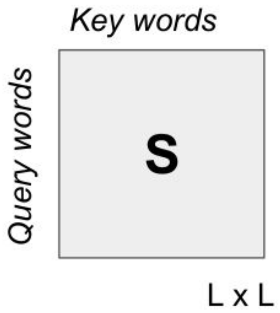
Self-Attention



Self-Attention

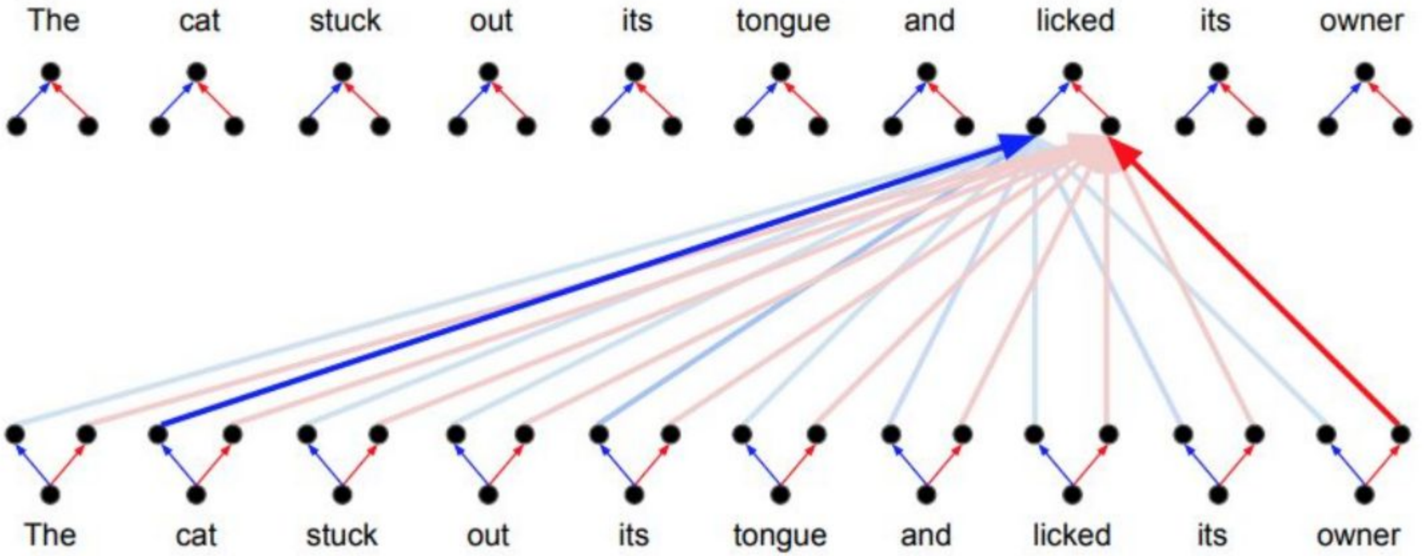


$$A(Q,K,V) = \boxed{softmax(\frac{QK^T}{\sqrt{d_k}})}V$$

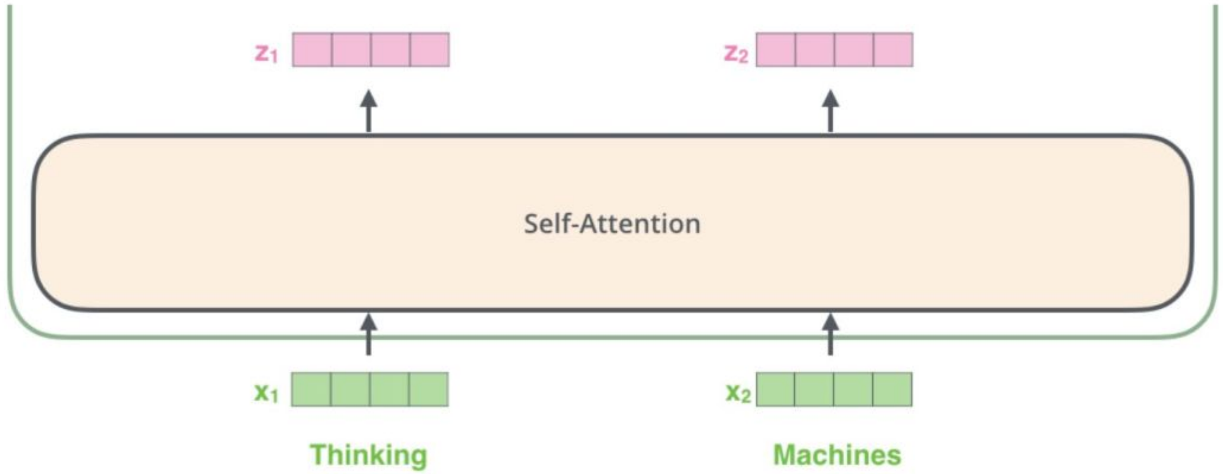


Multi-Head Self-Attention

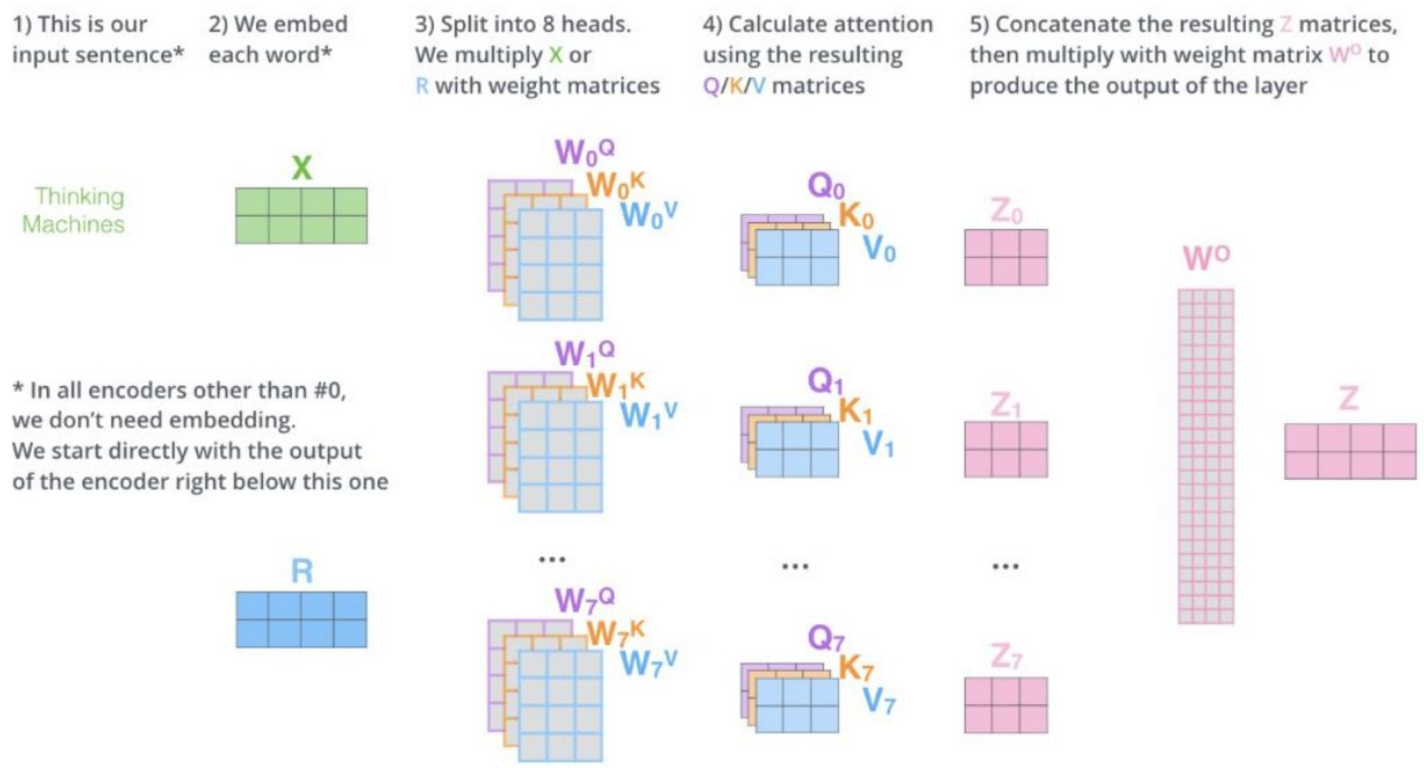
Parallel attention layers with different linear transformations on input and output.



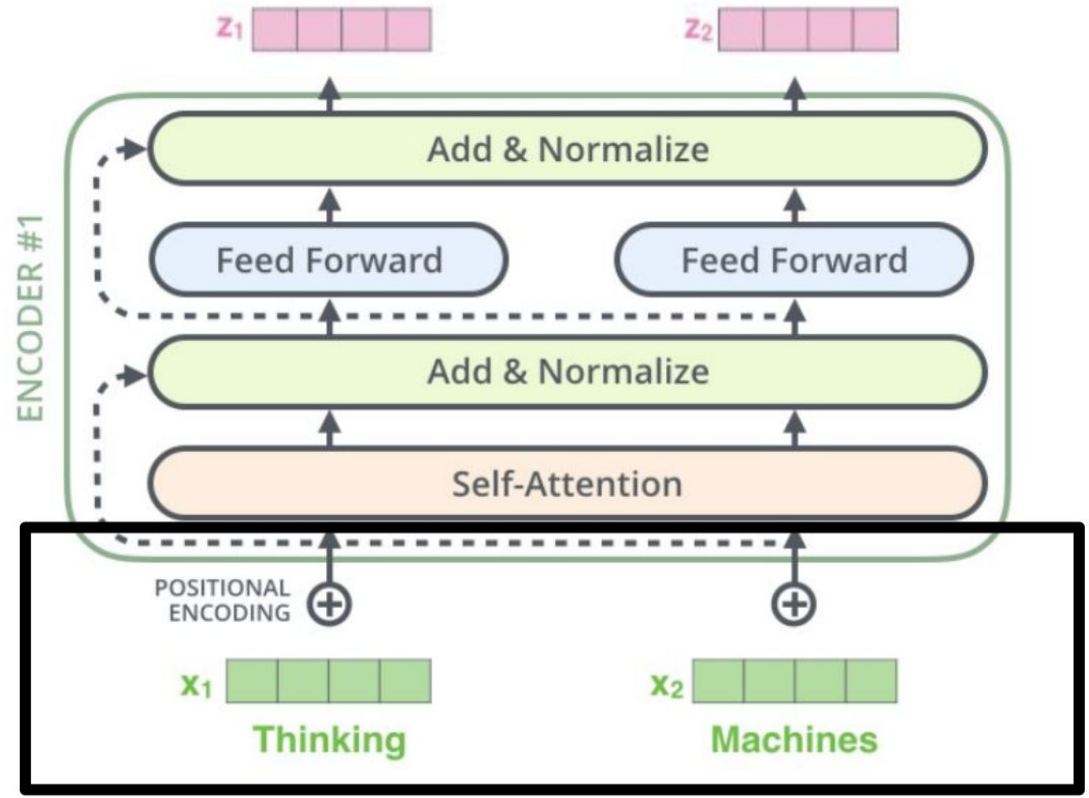
Retaining Hidden State Size



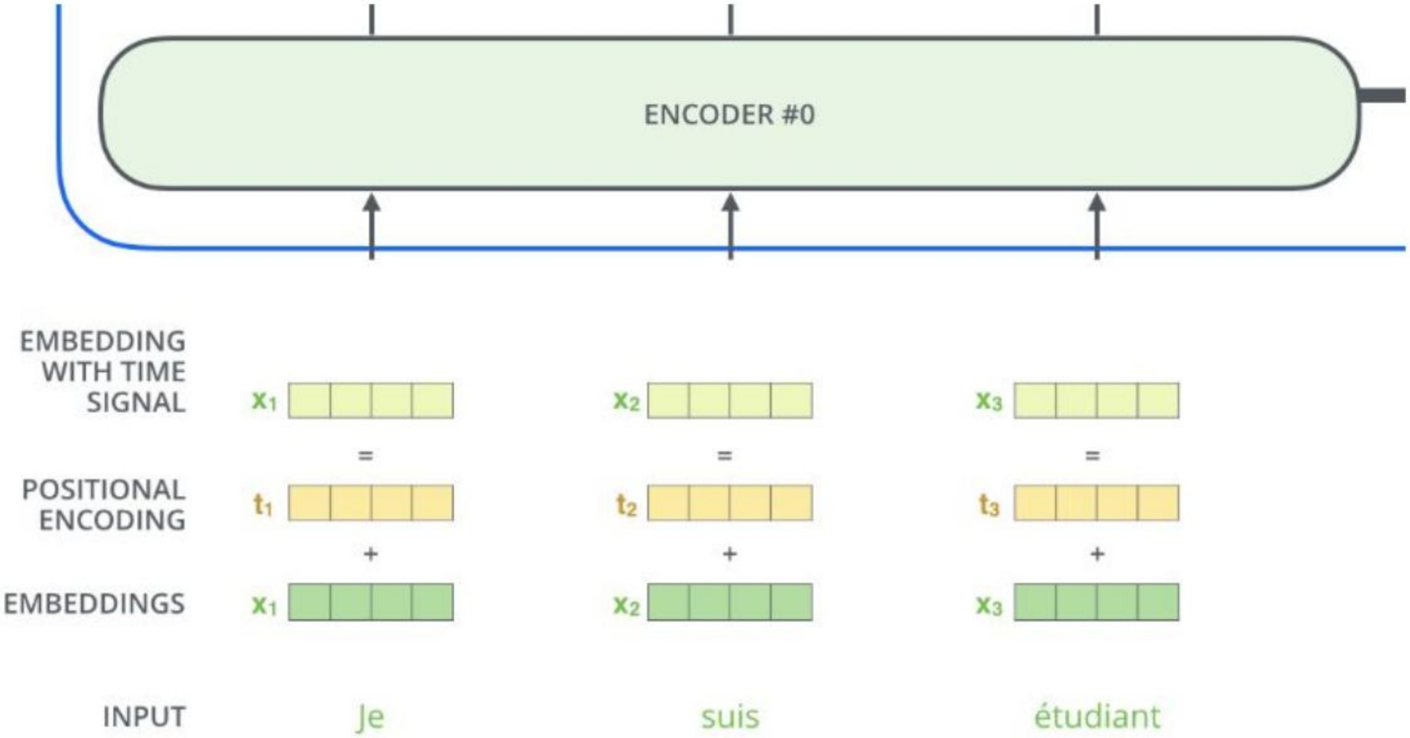
Details of Each Attention Sub-Layer of Transformer Encoder



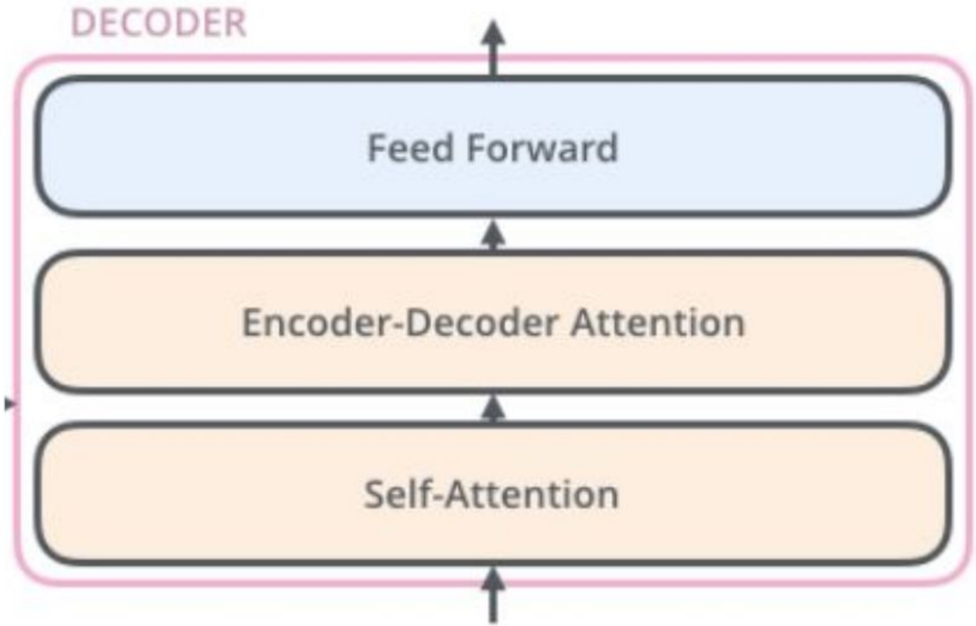
Each Layer of Transformer Encoder



Positional Encoding

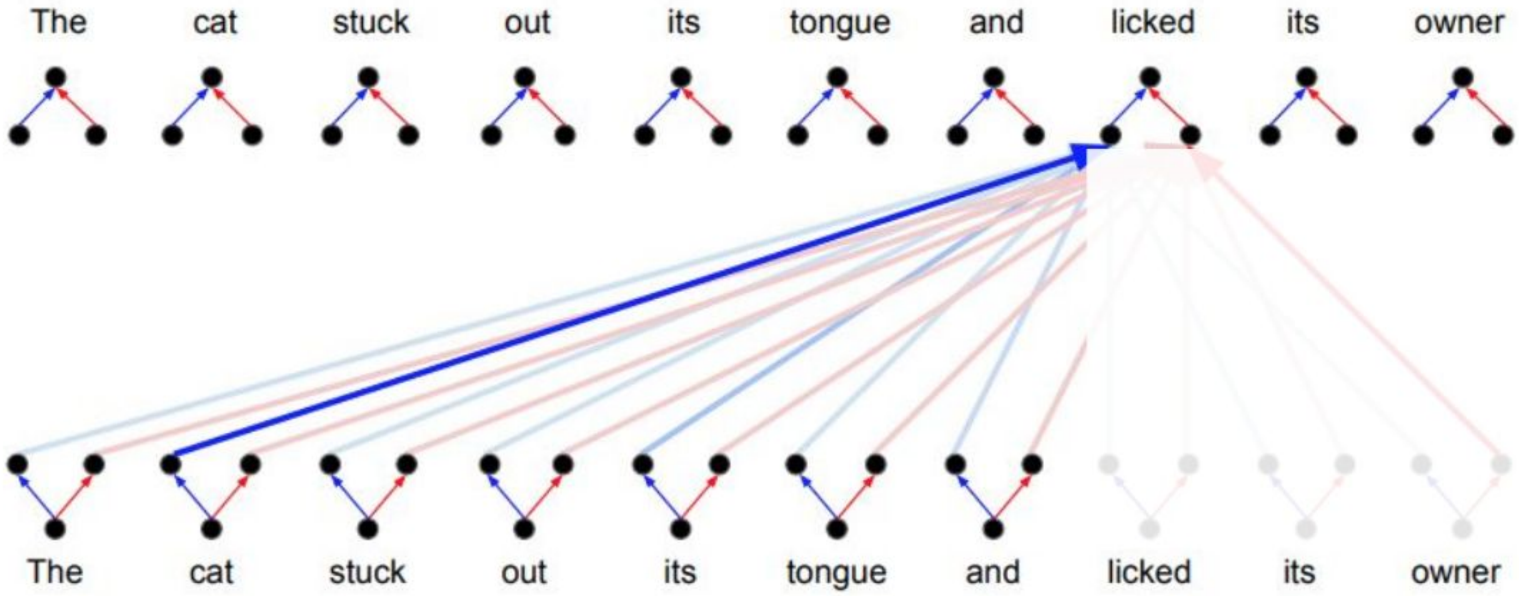


Each Layer of Transformer Decoder



Transformer Decoder - Masked Multi-Head Attention

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

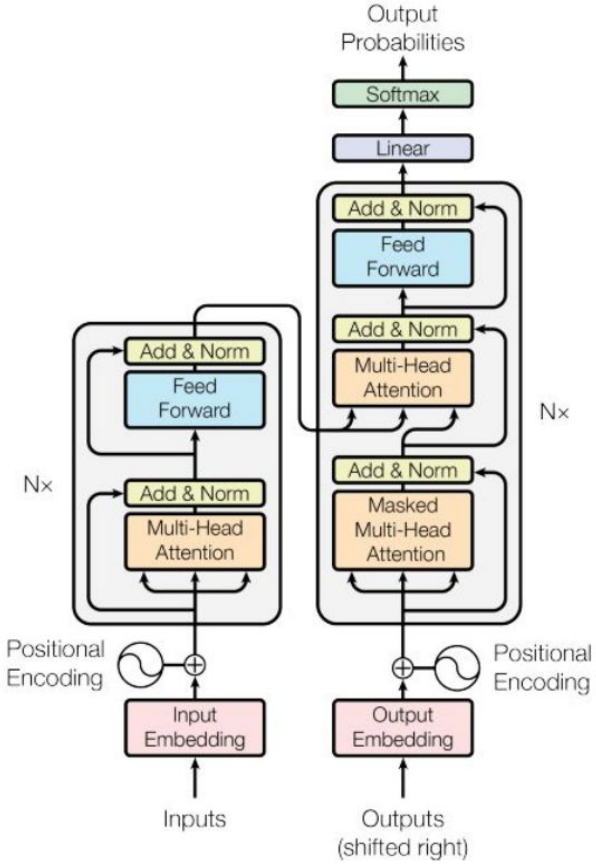


Transformer

“Attention is All You Need”

(Vaswani et. al 2017)

Encoder →



← Decoder