Introduction of Deep Learning with Convolutional Neural Networks and Long Short Term Memory

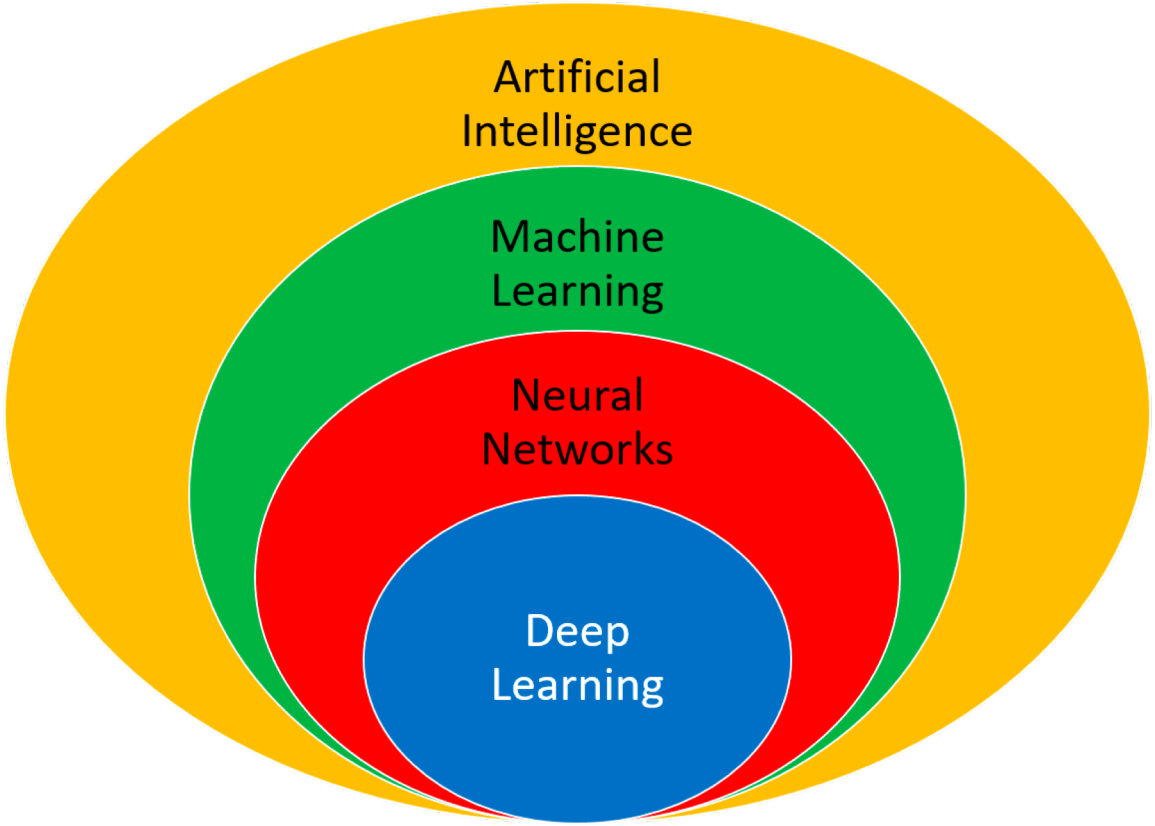
Lê Anh Cường

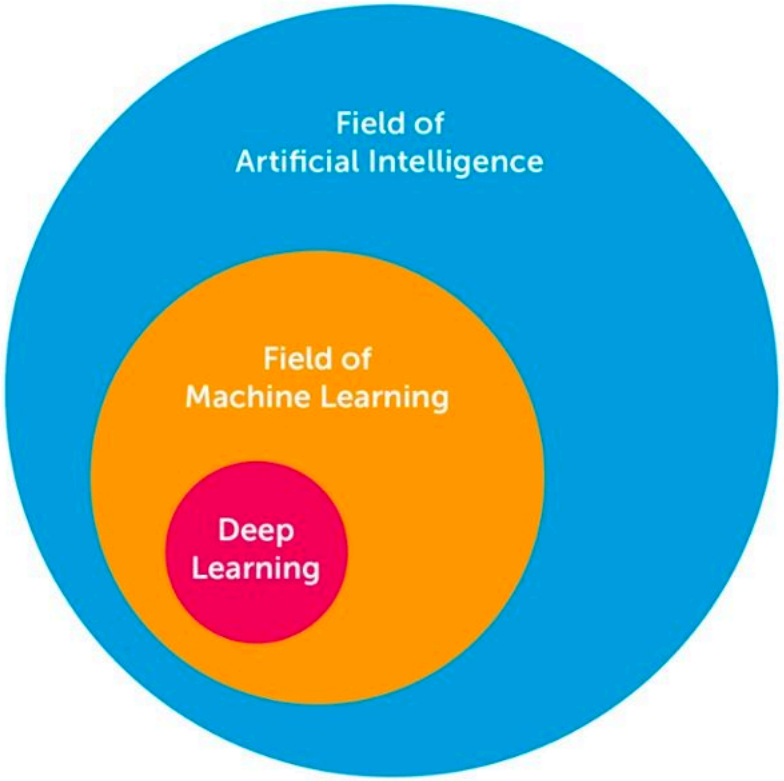
TDTU

Outline

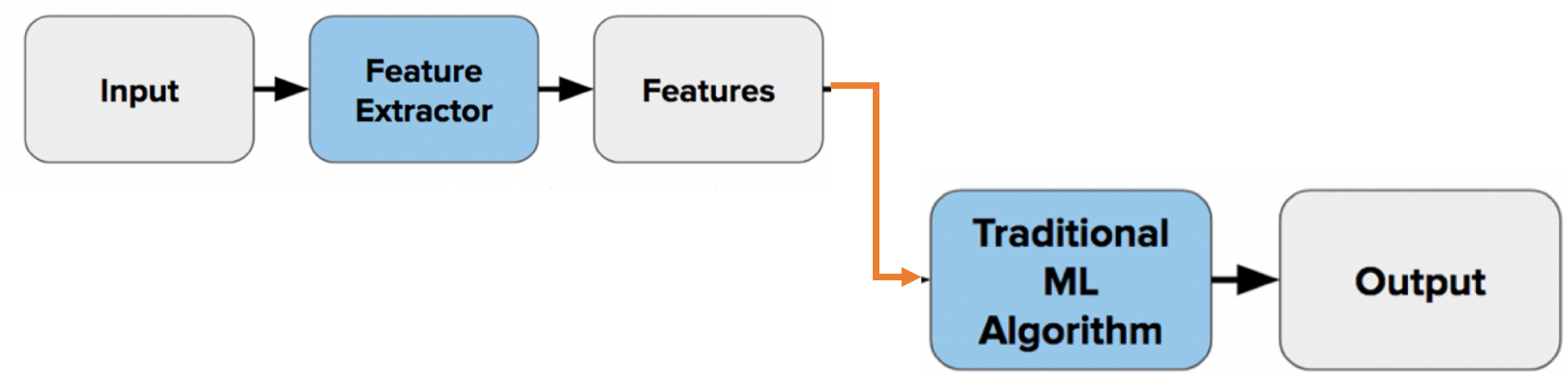
1. How to recognize Deep Learning networks/models:
   - Traditional statistical machine learning vs Deep Learning
   - Vanilla Neural Networks vs Deep Learning Networks
   - History of DL development
2. Convolutional Neural Network (CNN)
3. Reccurent Neural Network (RNN) and Long Short Term Memory (LSTM)
4. Transformer Models

AI, ML, NN and DL





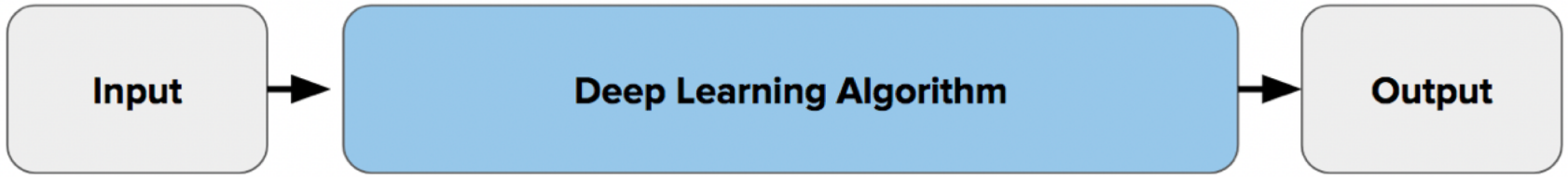
Traditional Statistical Machine Learning vs Deep Learning



Traditional Machine Learning Flow

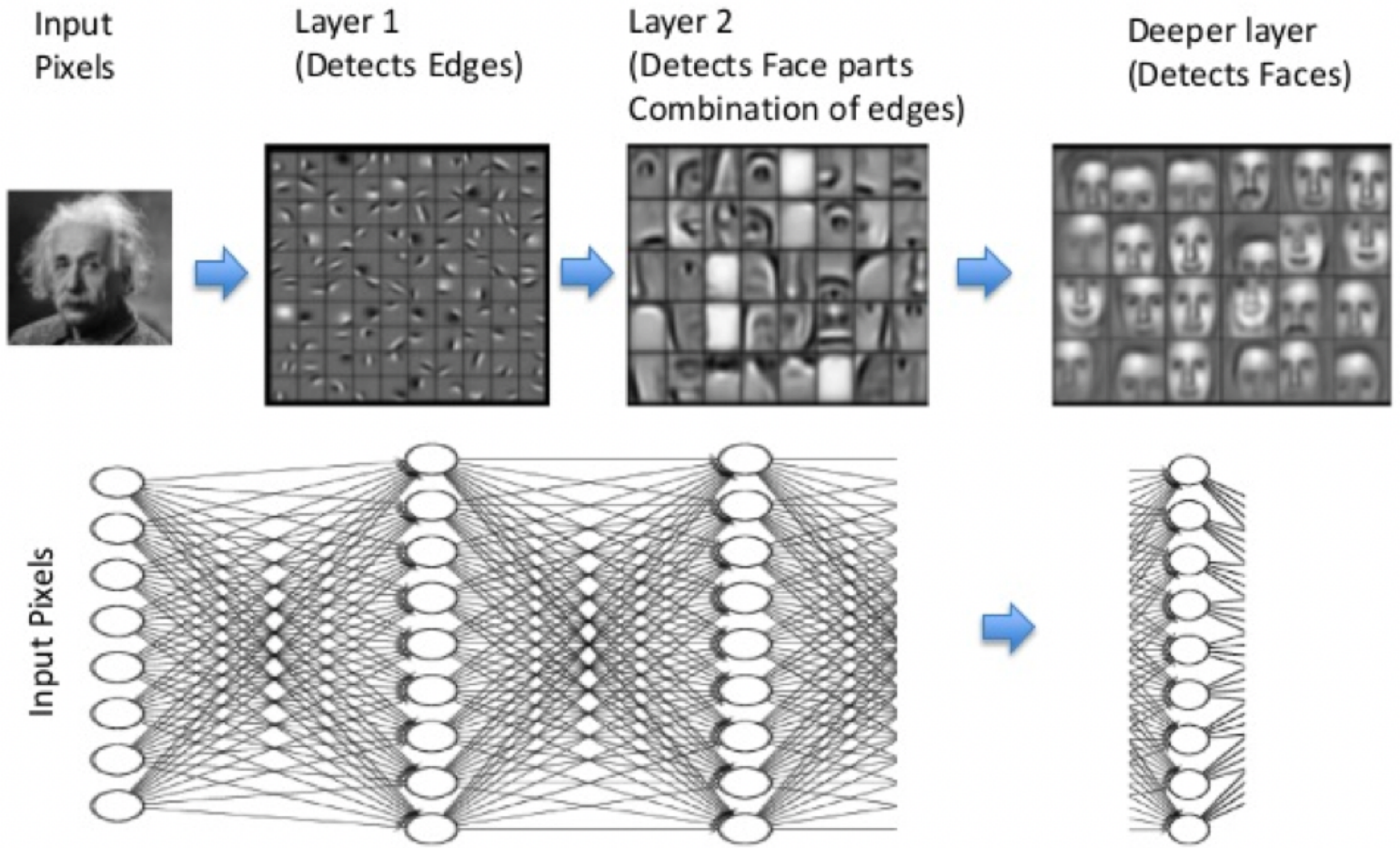
Traditional Statistical Machine Learning vs Deep Learning

Traditional Machine Learning Flow

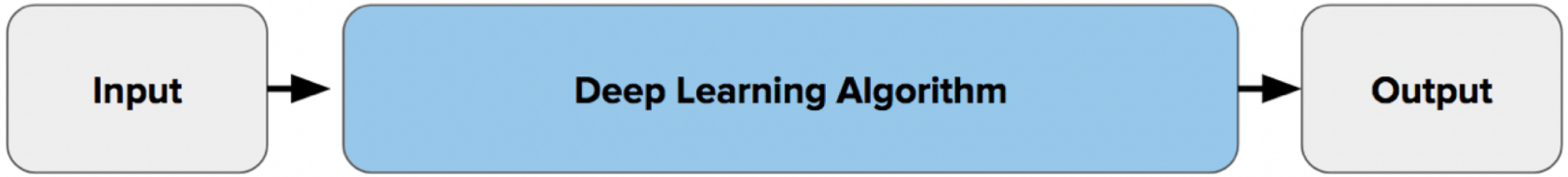


Deep Learning Flow

DL as Representation Learning

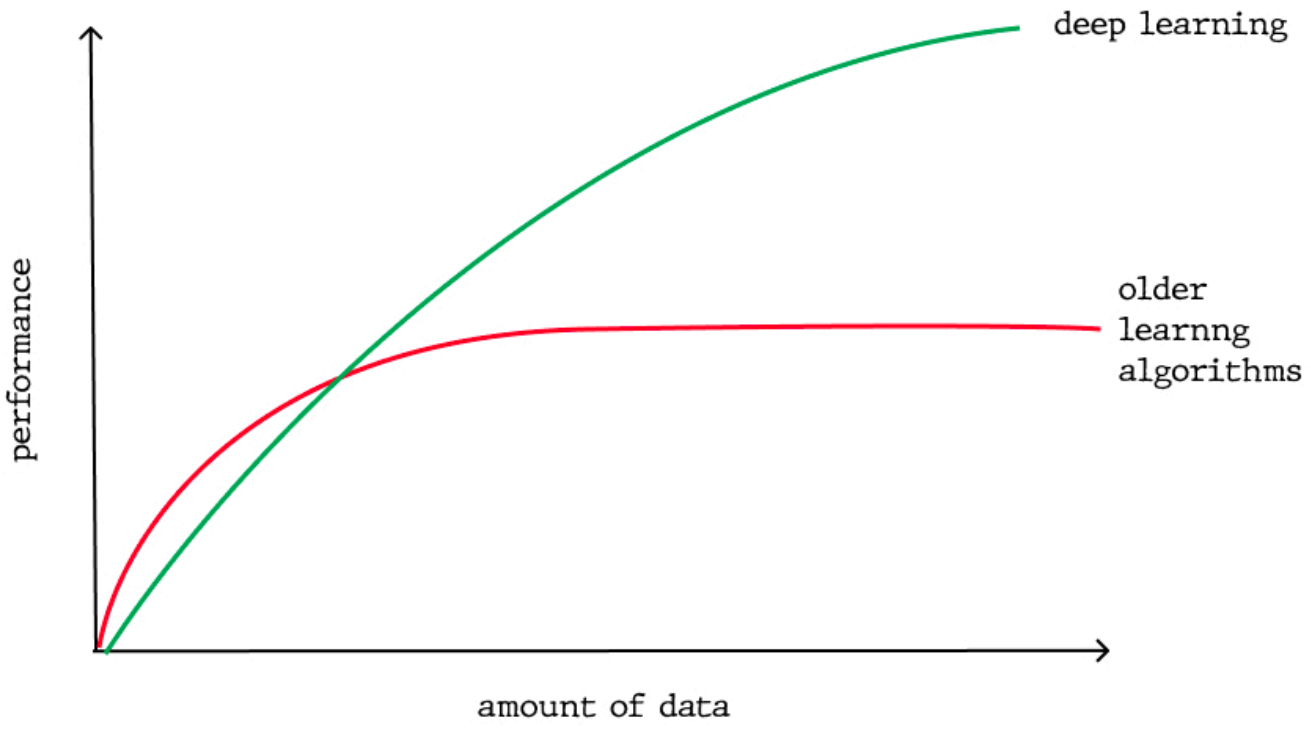


End-to-End Model

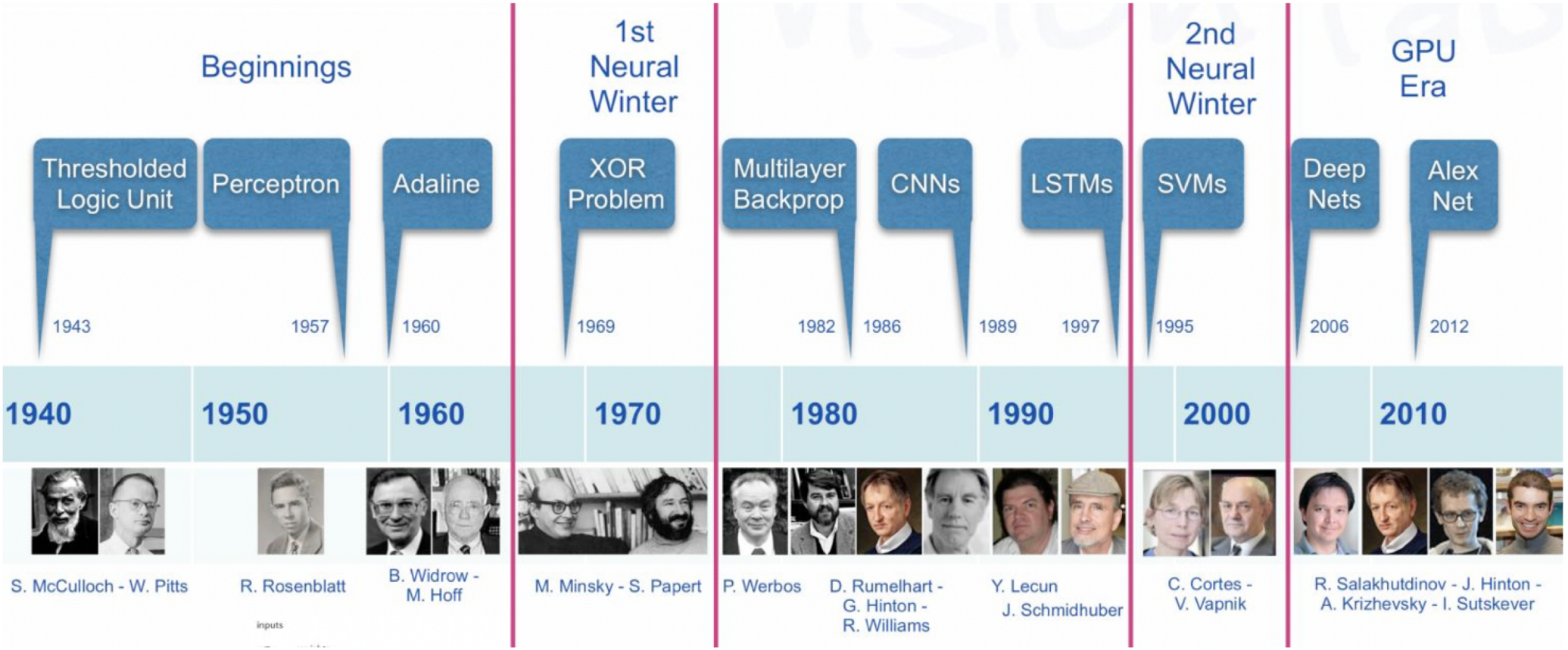


Deep Learning Flow

ML vs DL in Performance



History of Deep Learning



Recent Deep Learning models

- Generative Adversarial Network (2014)
- Residual Neural Network (2015)
- Transformer (2017)

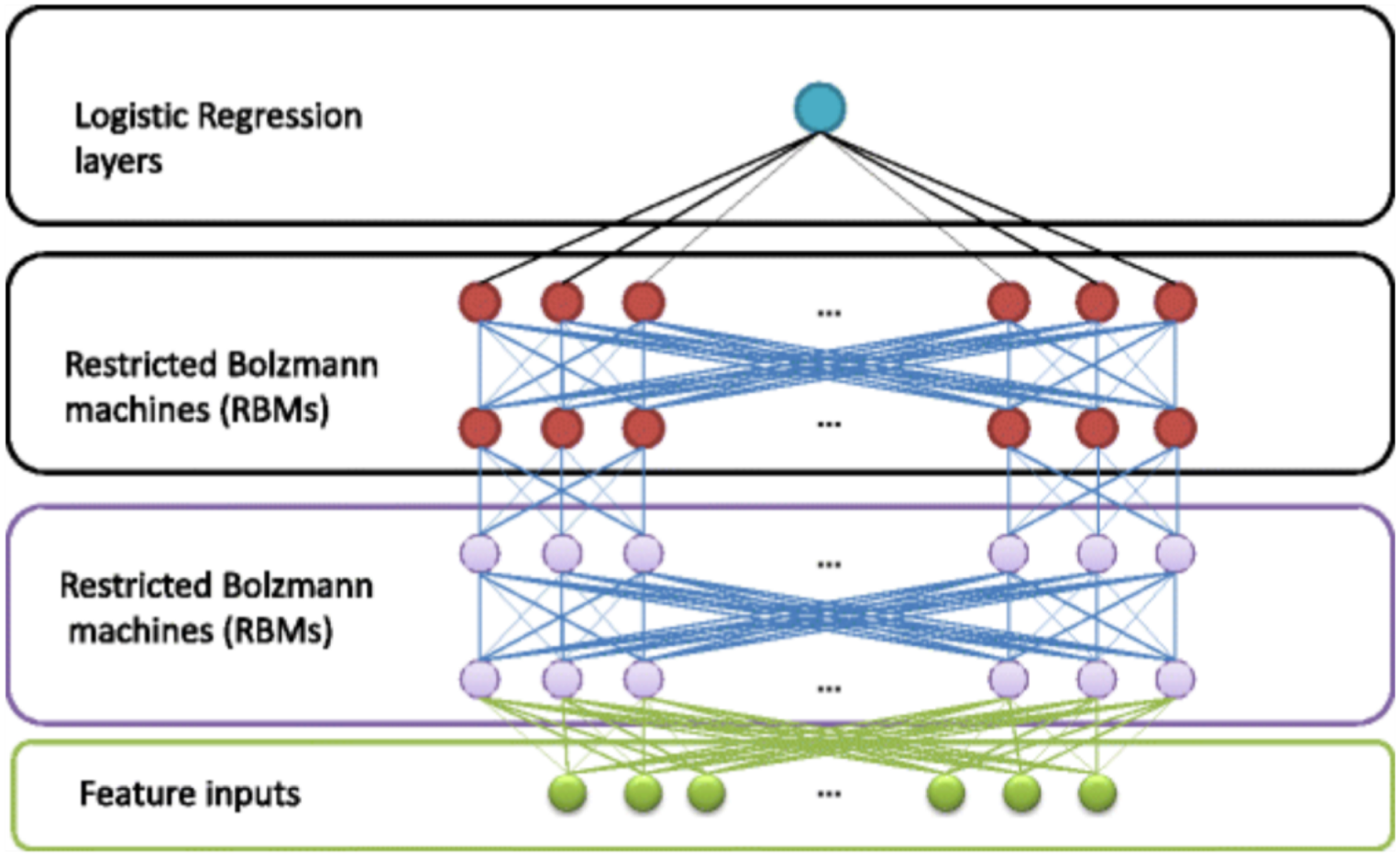
When the term Deep Learning?

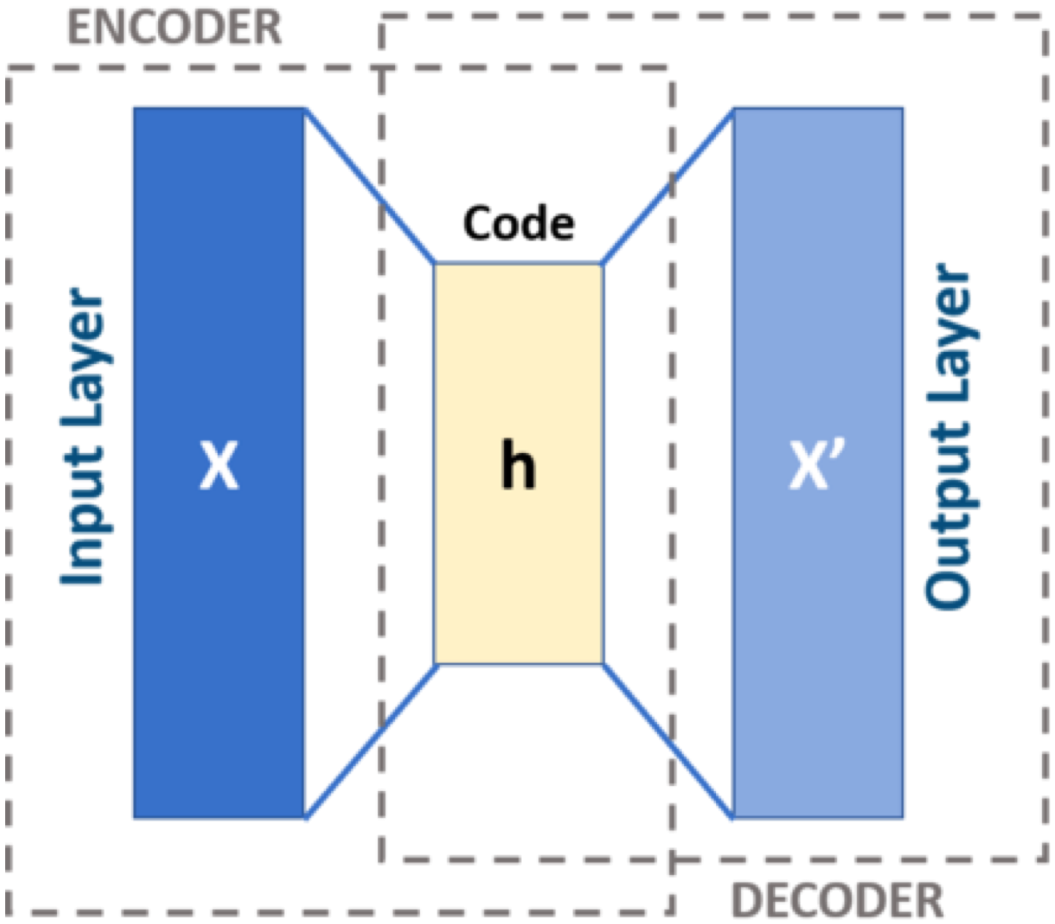
Geoffrey Hinton is a pioneer in the field of artificial neural networks and co-published the first paper on the backpropagation algorithm for training multilayer perceptron networks.

He may have started the introduction of the phrasing "*deep*" to describe the development of large artificial neural networks.

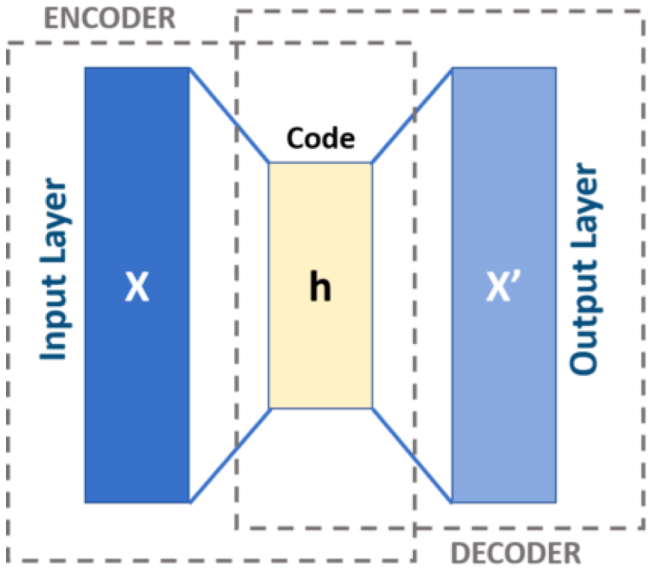
He co-authored a paper in 2006 titled "A Fast Learning Algorithm for Deep Belief Nets" in which they describe an approach to training "deep" (as in a many layered network) of restricted Boltzmann machines.

Deep Belief Network





An autoencoder is a type of artificial neural network used to learn efficient data codings in an unsupervised manner.[1] The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for dimensionality reduction



An autoencoder is a type of artificial neural network used to learn efficient data codings in an unsupervised manner.[1] The aim of an autoencoder is to learn
a representation (encoding) for a set of data, typically for dimensionality reduction

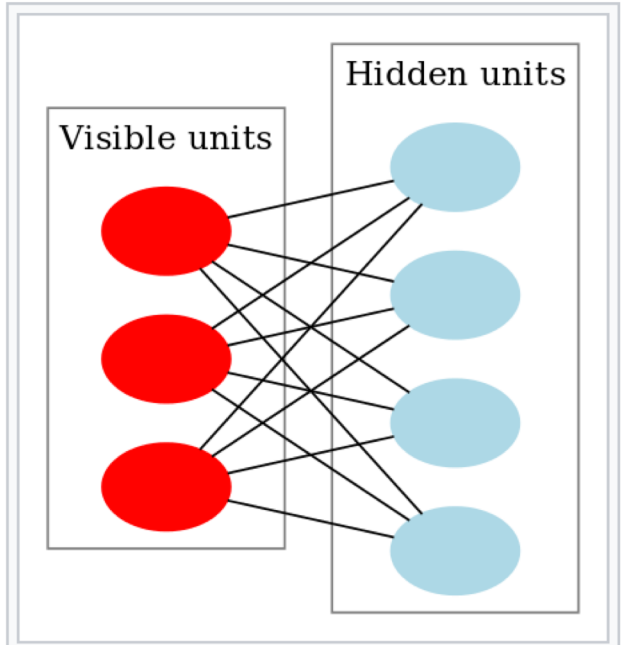


Diagram of a restricted Boltzmann machine with three visible units and four hidden units (no bias units).

A **restricted Boltzmann machine** (**RBM**) is a generative stochastic artificial neural network that can learn a probability distribution over its set of inputs.

Recognition of DL

- Deep: i.e. many layers/levels of data representation
- Big Data (large enough): for learning good features
- High Performance Computing: for doing with complex networks and big data

Convolutional Neural Network

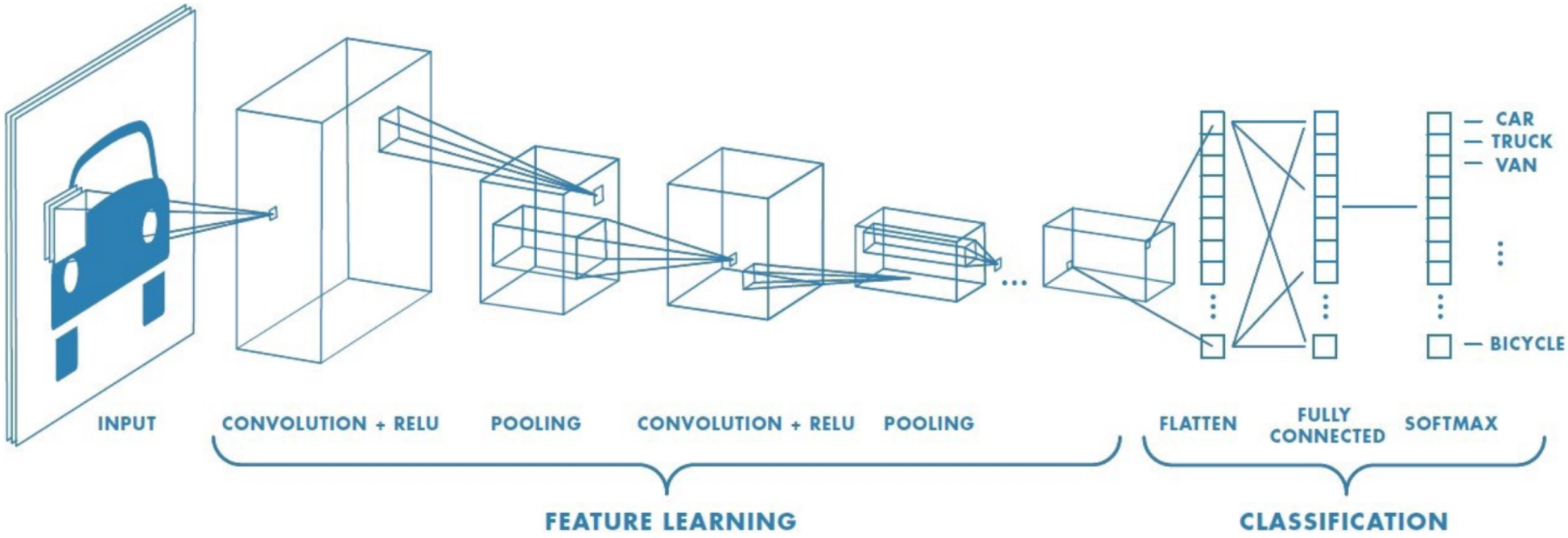
Outline

- What is CNN?
- Why CNN?
- CNN Explanation
- Implementation

ConvNet History

- Convolutional neural networks, also called ConvNets, were first introduced in the 1980s by Yann LeCun for building an image recognition neural network.
- LeNet-5, a pioneering 7-level convolutional network by LeCun et al. in 1998, that classifies digits, was applied by several banks to recognize hand-written numbers on checks.
- Although CNNs were invented in the 1980s, their breakthrough in the 2000s required fast implementations on graphics processing units (GPUs).
- A similar GPU-based CNN by Alex Krizhevsky et al. won the ImageNet Large Scale Visual Recognition Challenge 2012. A very deep CNN with over 100 layers by Microsoft won the ImageNet 2015 contest

A general architecture of Convolutional Neural Networks

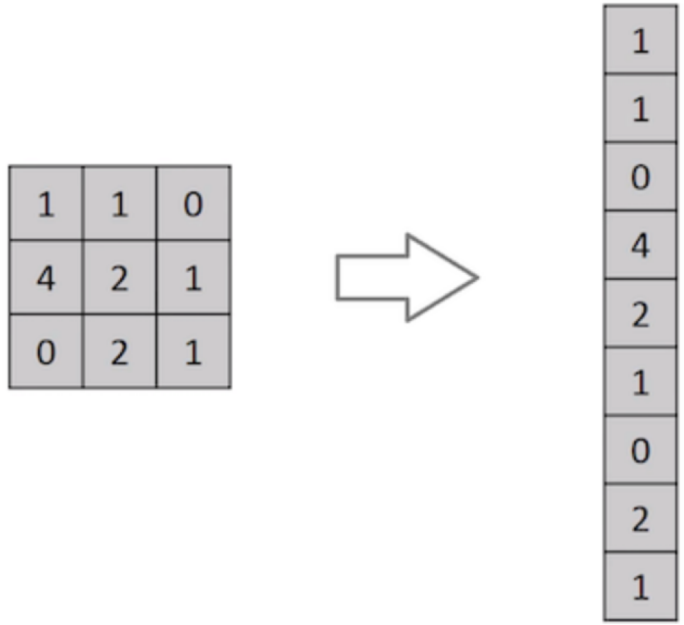


What is Convolutional Neural Network

- Convolutional Neural Networks (CNNs) are simply neural networks that use Convolution Operation which is a specialized kind of linear operation.
- CNNs are regularized versions of multilayer perceptrons.
- CNNs take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns.

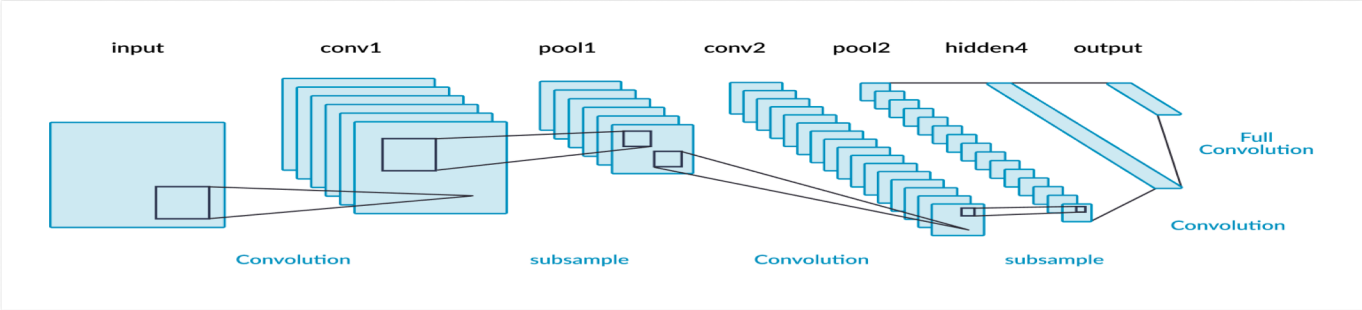
Why ConvNets over Feed-Forward Neural Nets?

- A ConvNet is able to **successfully capture the Spatial and Temporal dependencies.**
- The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters.
- The network can be trained to understand the sophistication of the image better.

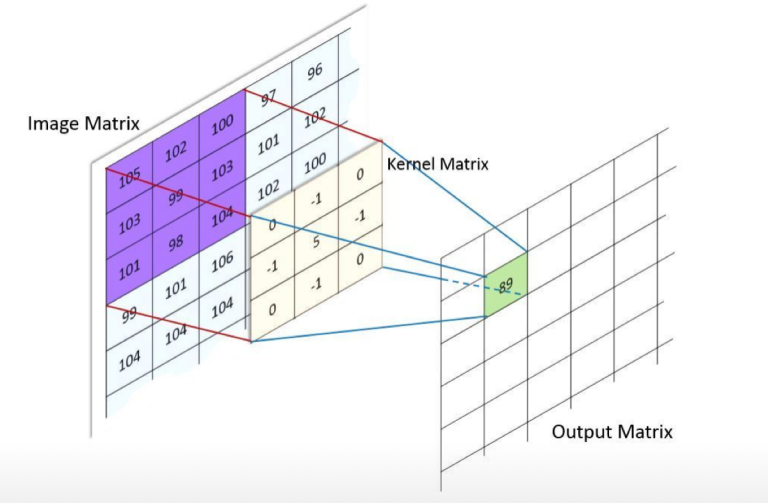


Flattening of a 3x3 image matrix into a 9x1 vector

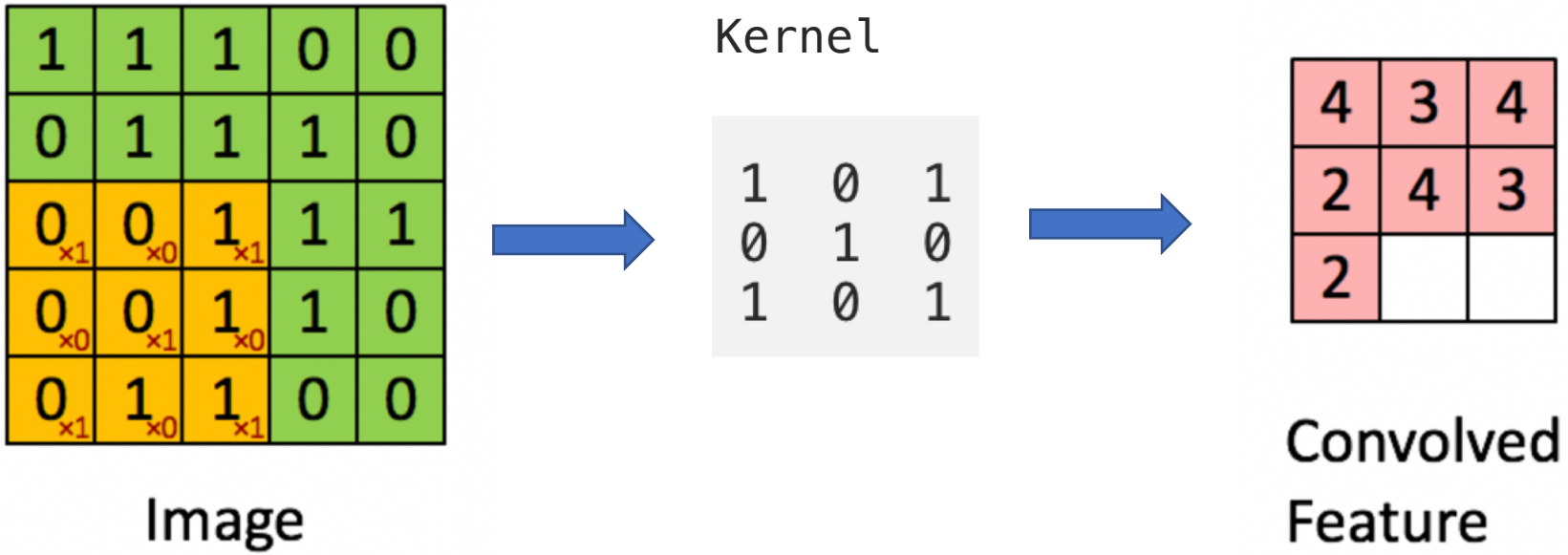
Convolution Layer — The Kernel



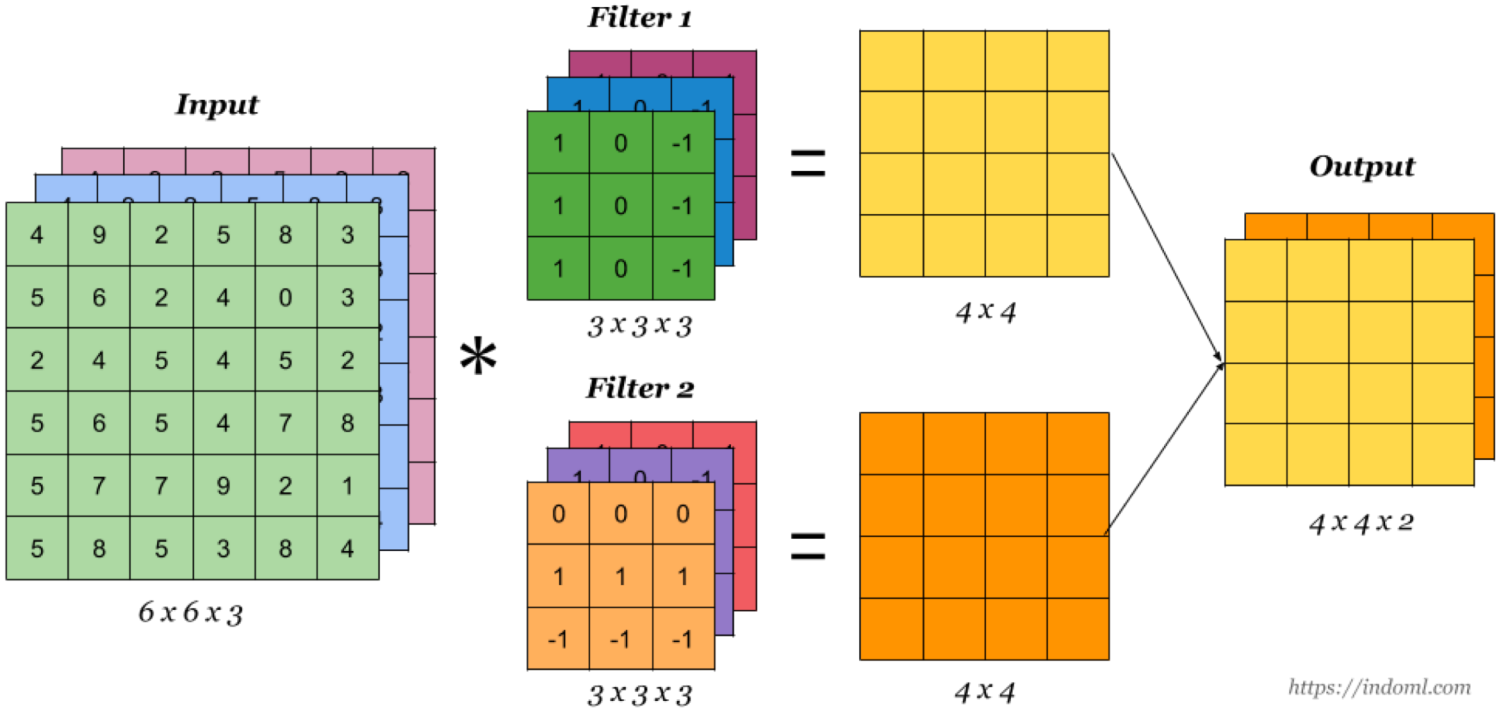
The objective of the Convolution Operation is to **extract the high-level features** such as edges, from the input image.



Convolution Layer — The Kernel

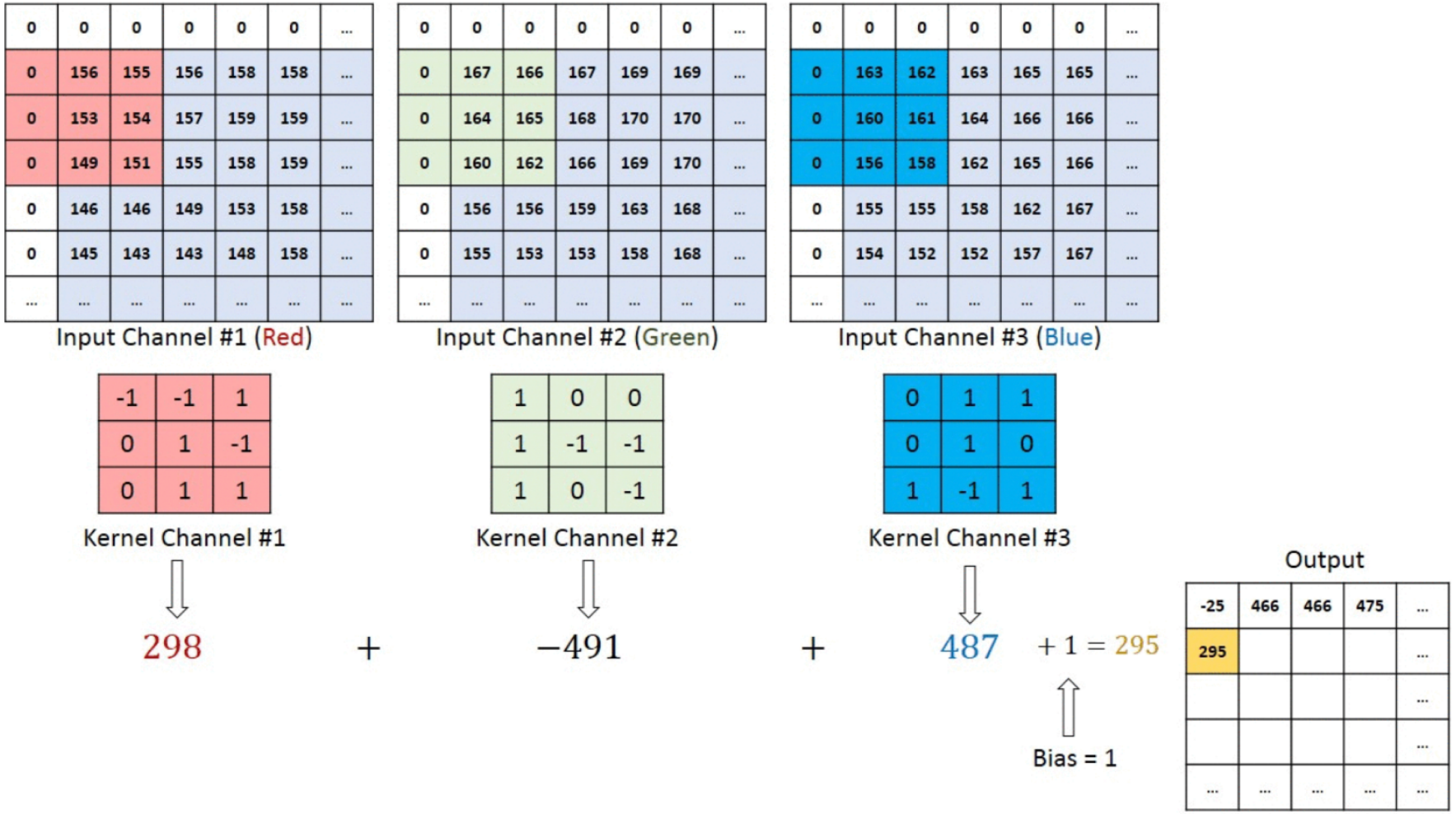


Multi-Channel Input and Filter



A common convolution layer actually consist of multiple such filters. *For the sake of simplicity in the discussion to follow, assume the presence of only one filter unless specified, since the same behavior is replicated across all the filters.*

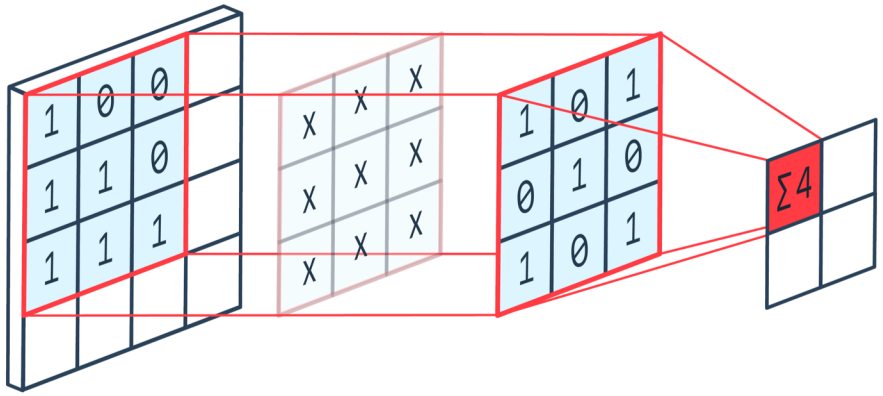
a *multi-input channel* convolution kernel

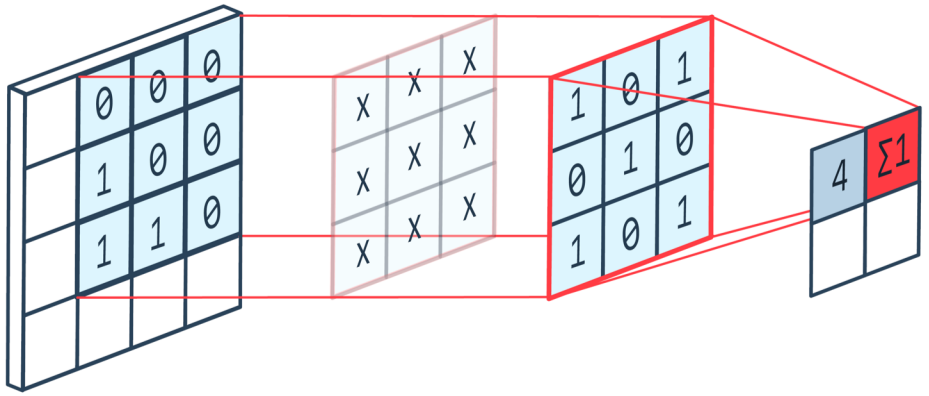


Convolution operation on a MxNx3 image matrix with a 3x3x3 Kernel

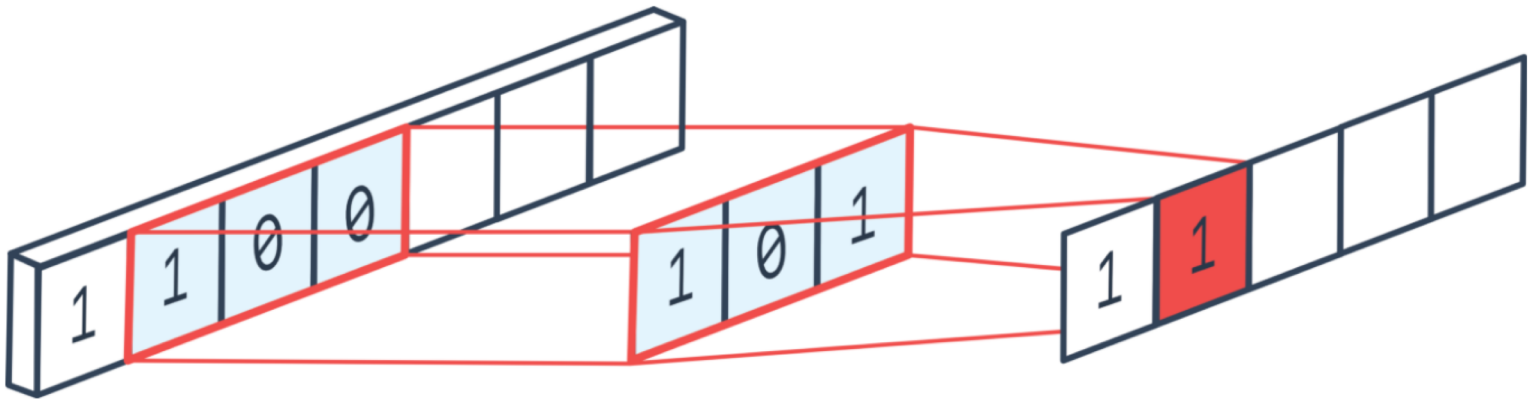
2D Convolutions

- The filter can move in 2 directions and thus the final output is 2D.
- 2D convolutions are the most common convolutions, and are heavily used in Computer Vision.

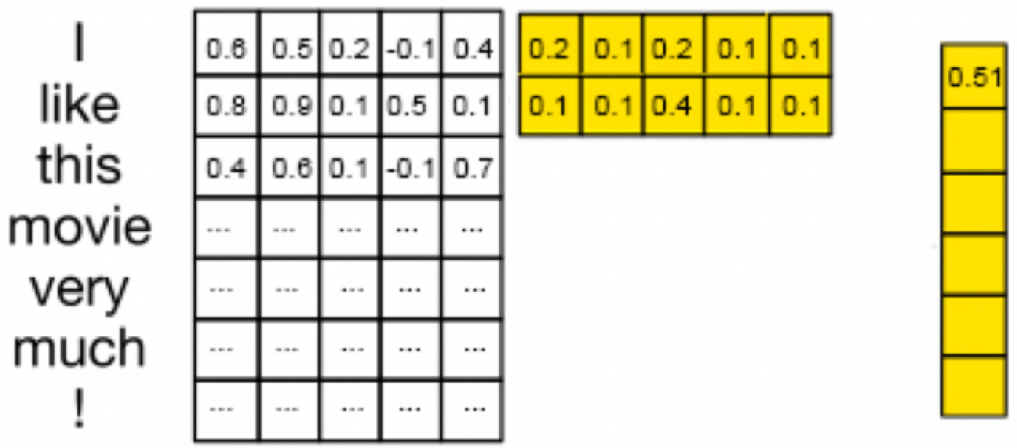


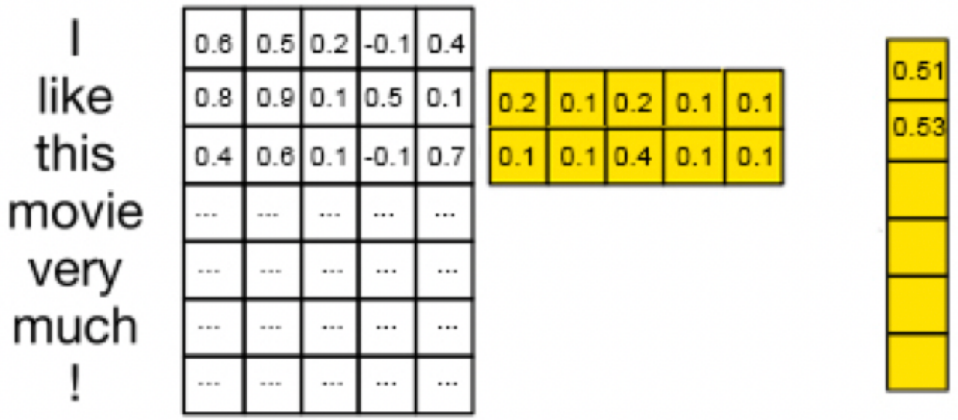


1D Convolutions

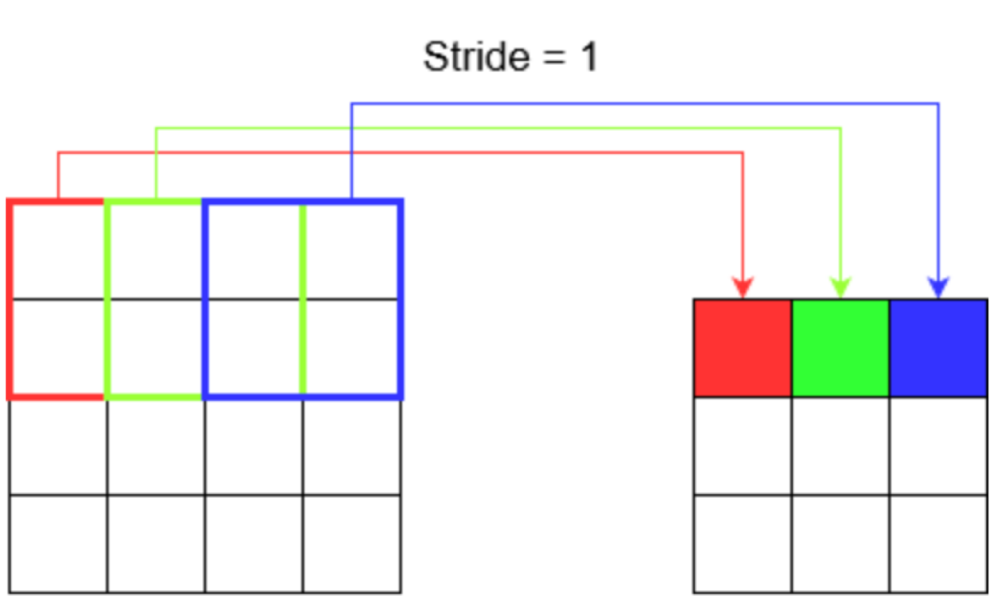


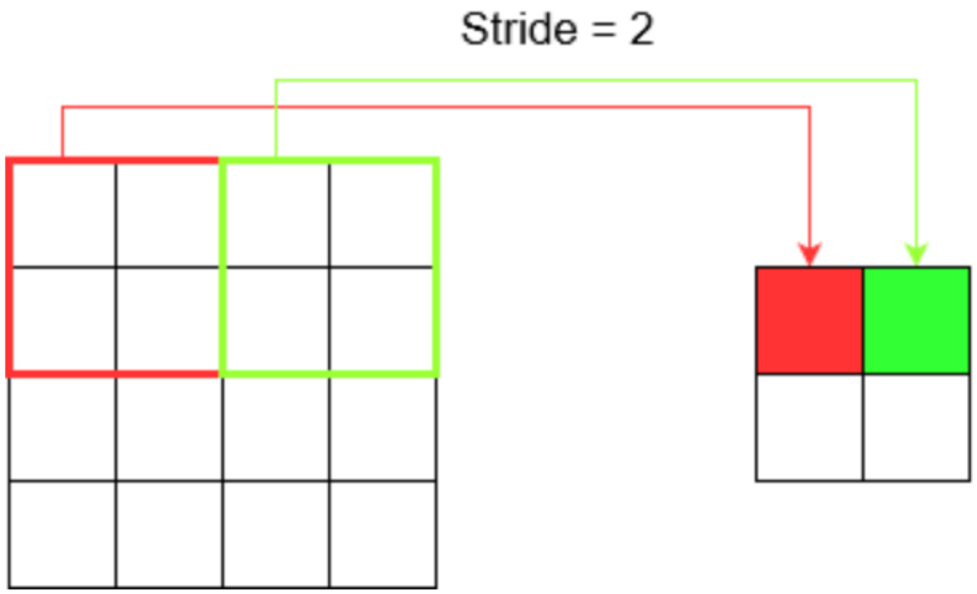
1D Convolution with 2D input



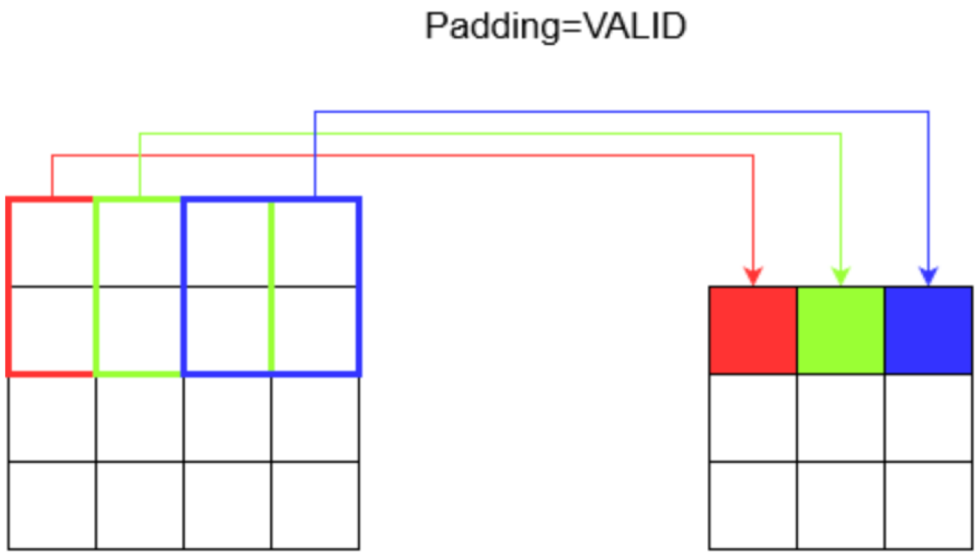


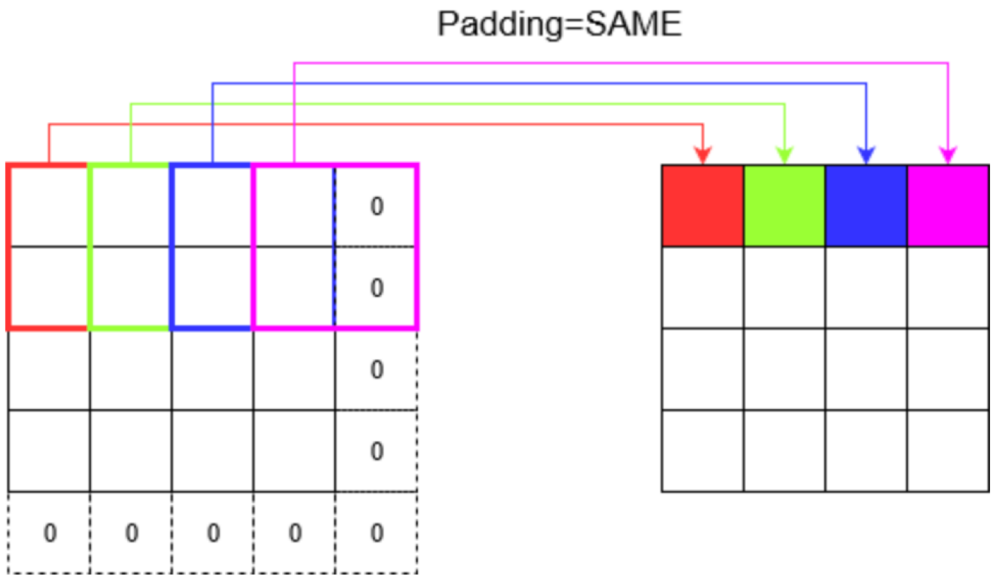
Stride





Padding

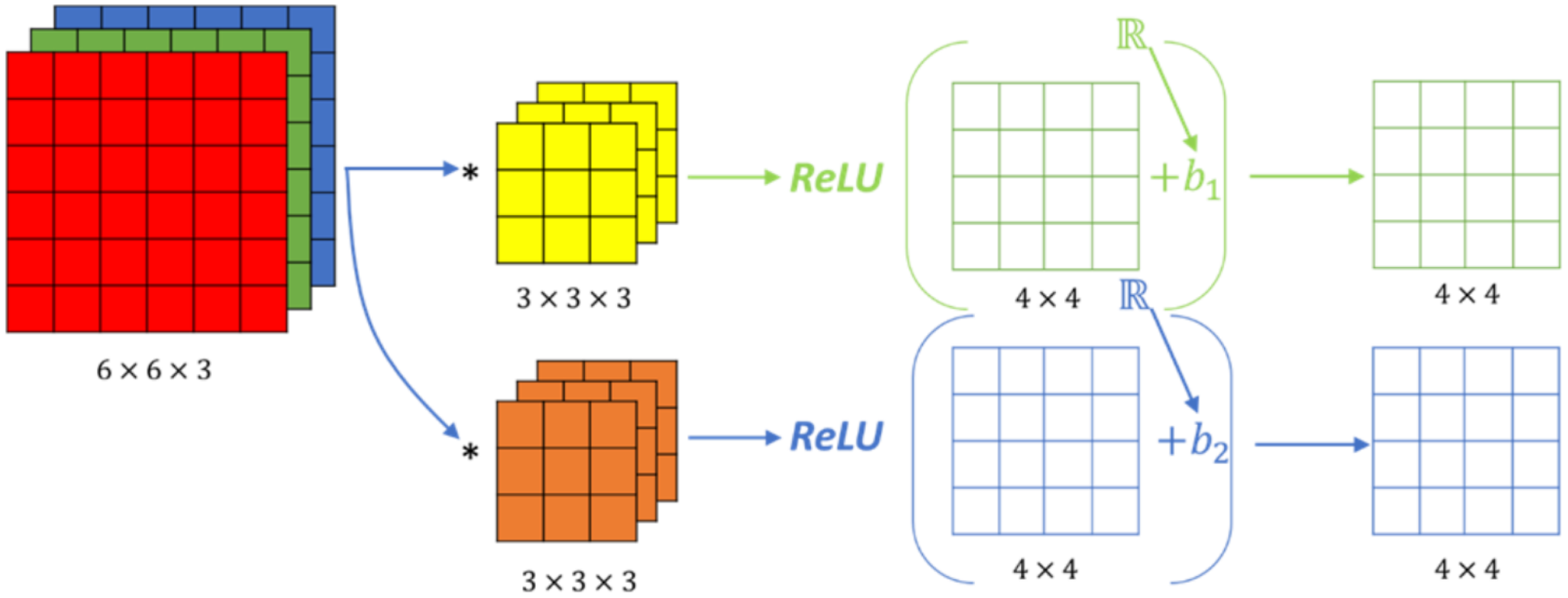




ReLU Operation

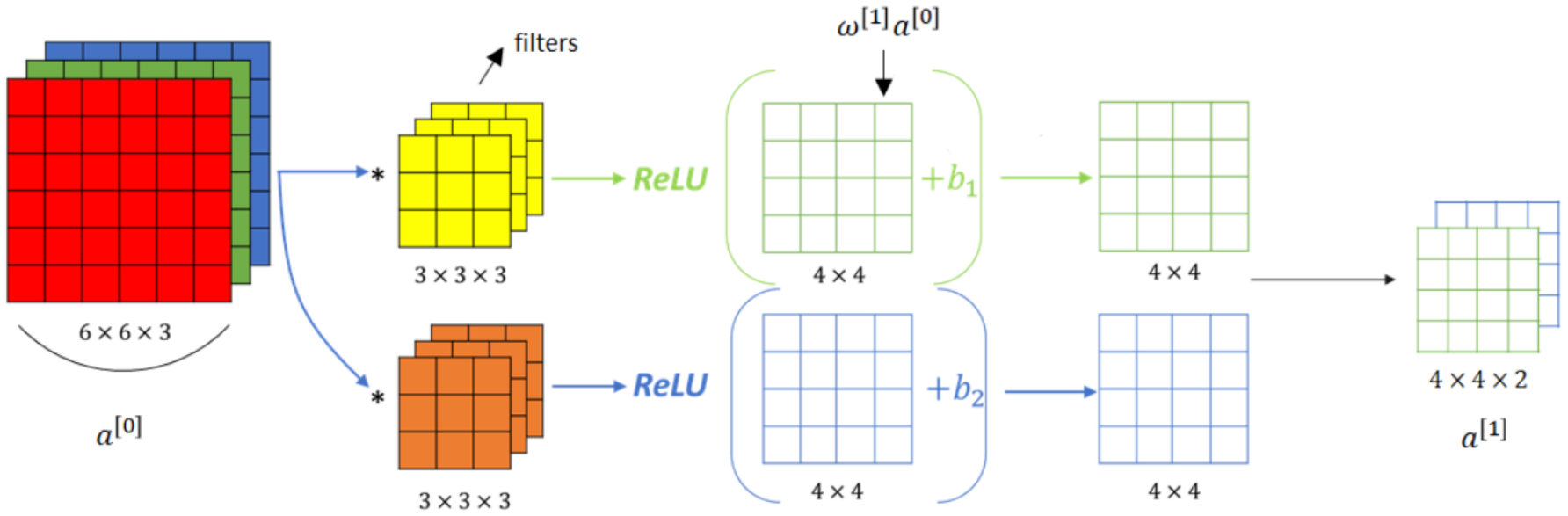
ReLU

$$\max(0, x)$$



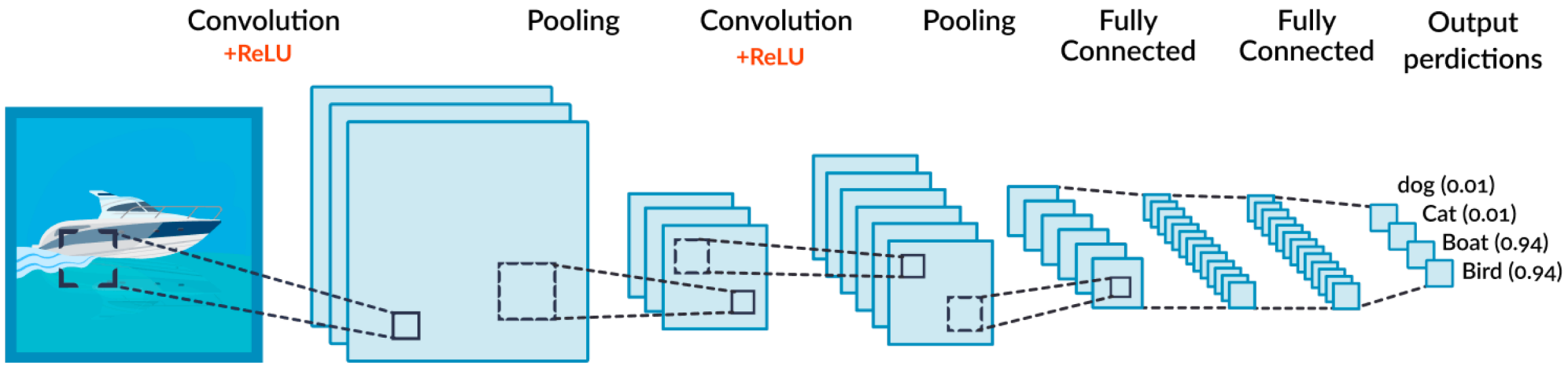
The result of convolution with two filters

ReLU



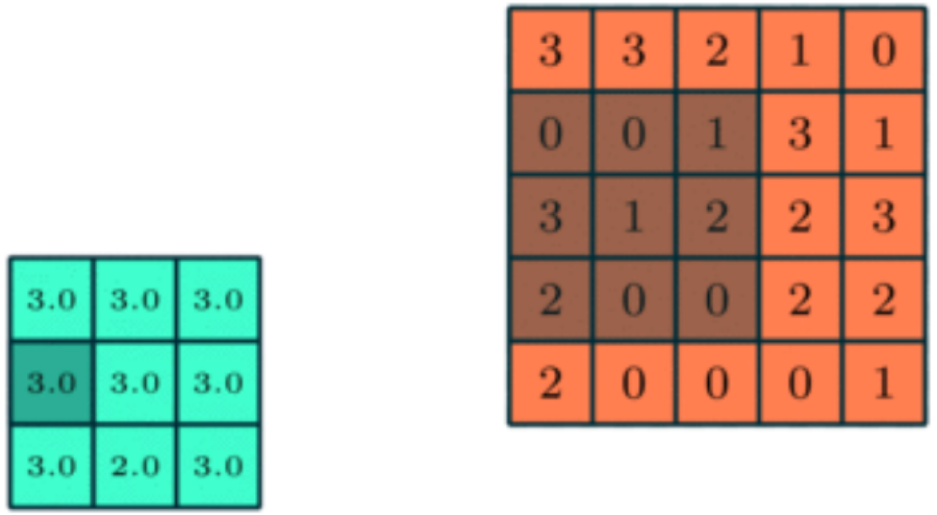
The result of a convolution of $6 \times 6 \times 3$ *with two* $3 \times 3 \times 3$ *is a volume of dimension* $4 \times 4 \times 2$

Pooling Layer



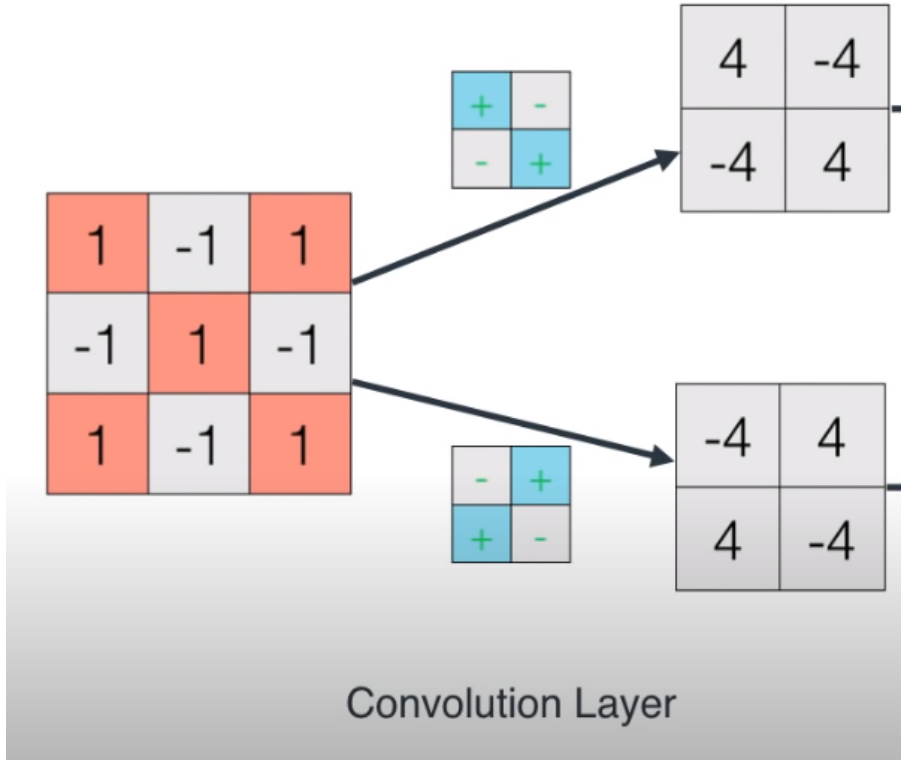
Pooling Layer

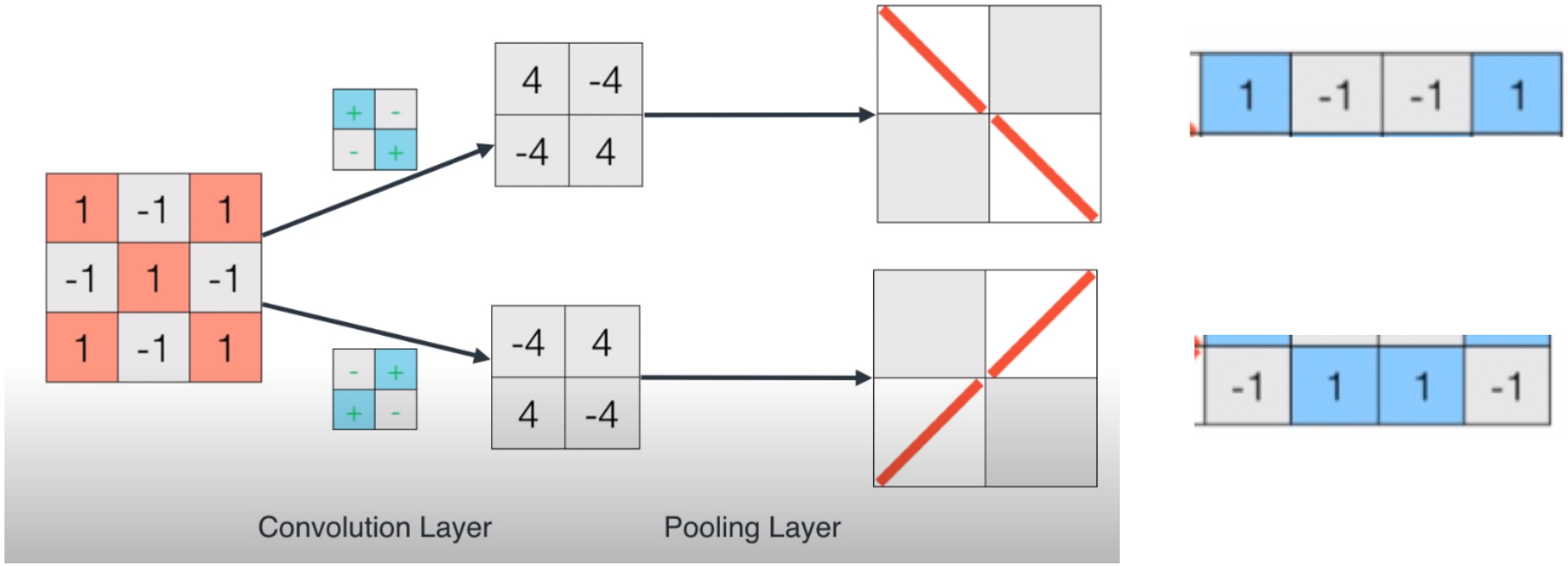
- The Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to **decrease the computational power required to process the data** through dimensionality reduction.
- Furthermore, it is useful for **extracting dominant features** .



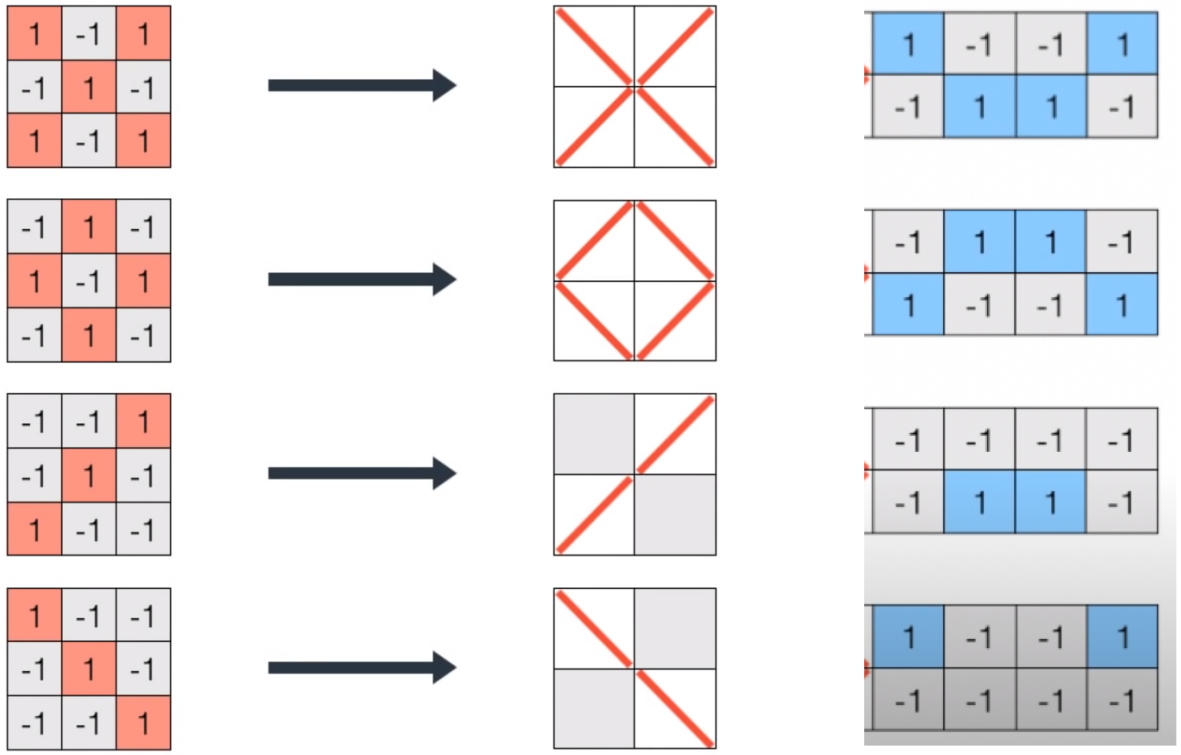
3x3 pooling over 5x5 convolved feature

Example for Convolutional Operations




1 -1 1
-1 1 -1
1 -1 1
+ -
- +
4 -4
-4 4
- +
+ -
-4 4
4 -4
Convolution Layer
Pooling Layer
1 -1 -1 1
-1 1 1 -1

Recognition of characters: X , O, /, \



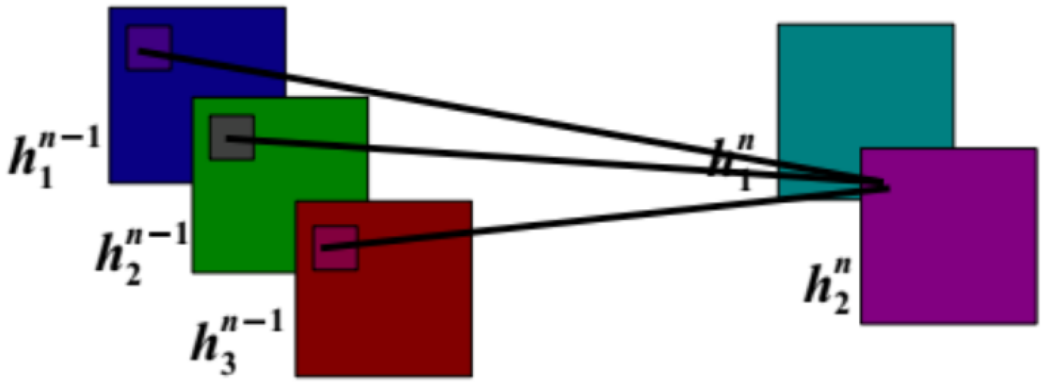
Convolutional Layer

$$h_j^n = max\left(0, \sum_{k=1}^{K} h_k^{n-1} * w_{kj}^n\right)$$

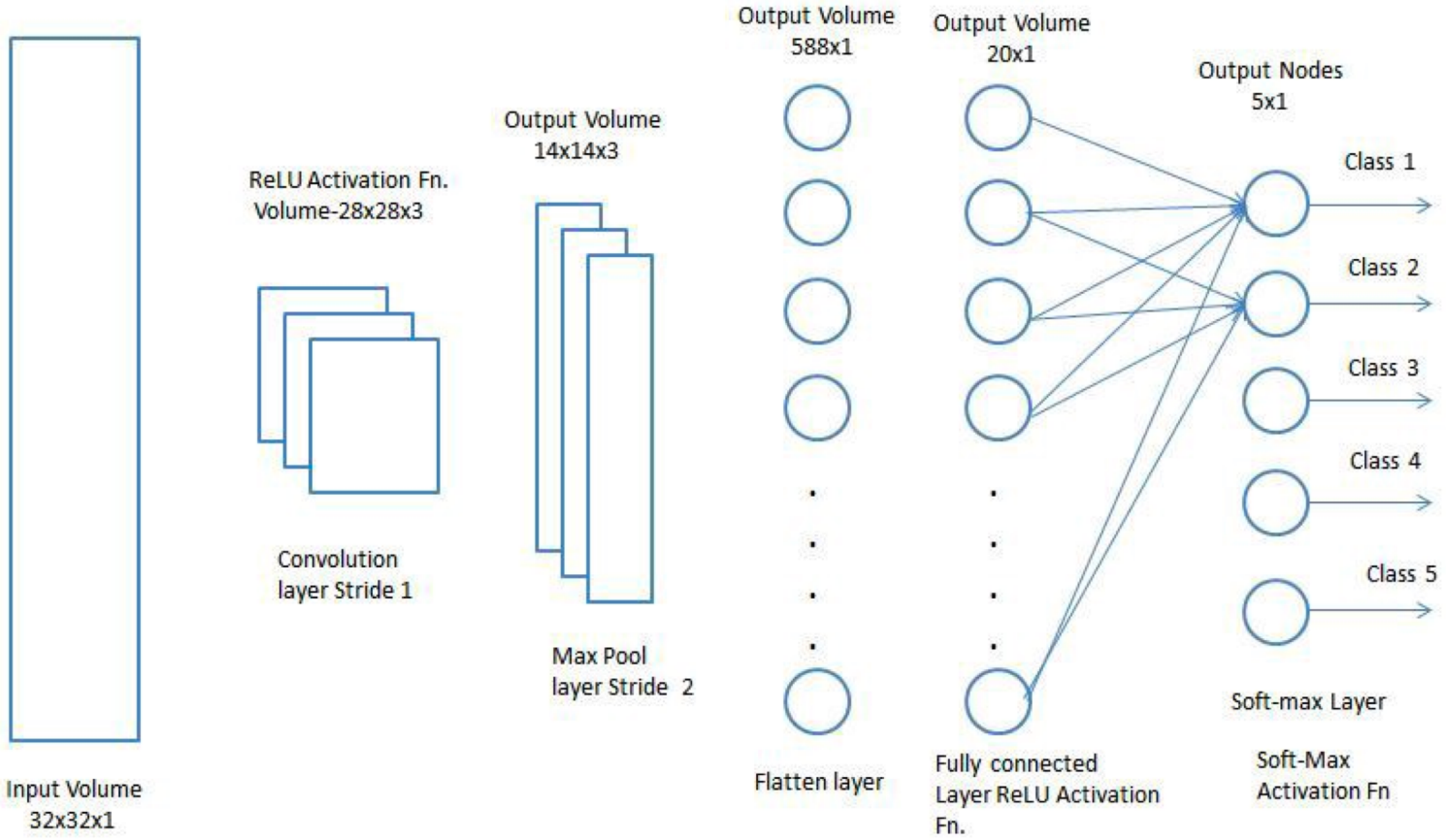
output feature map

input feature map

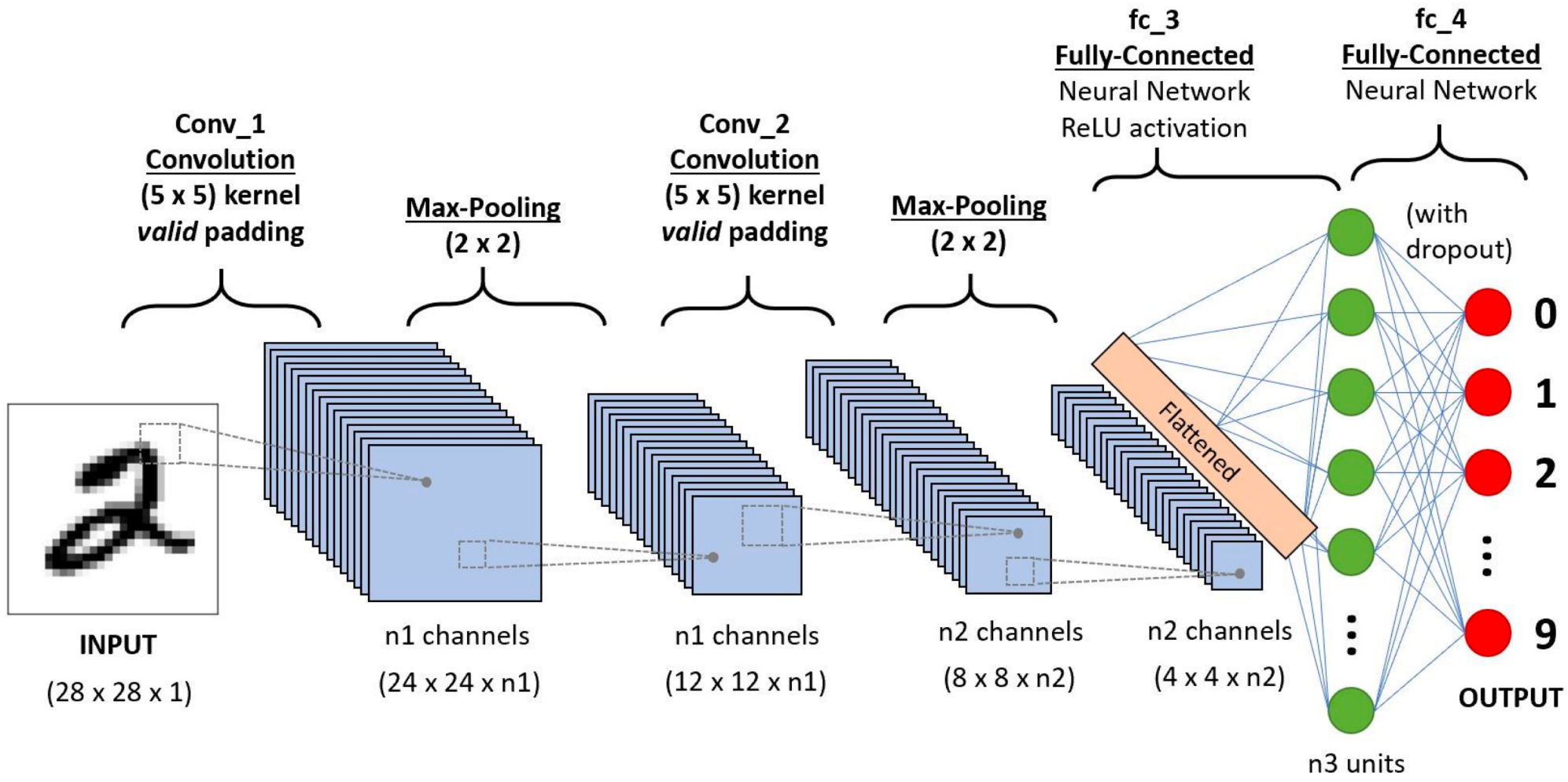
kernel

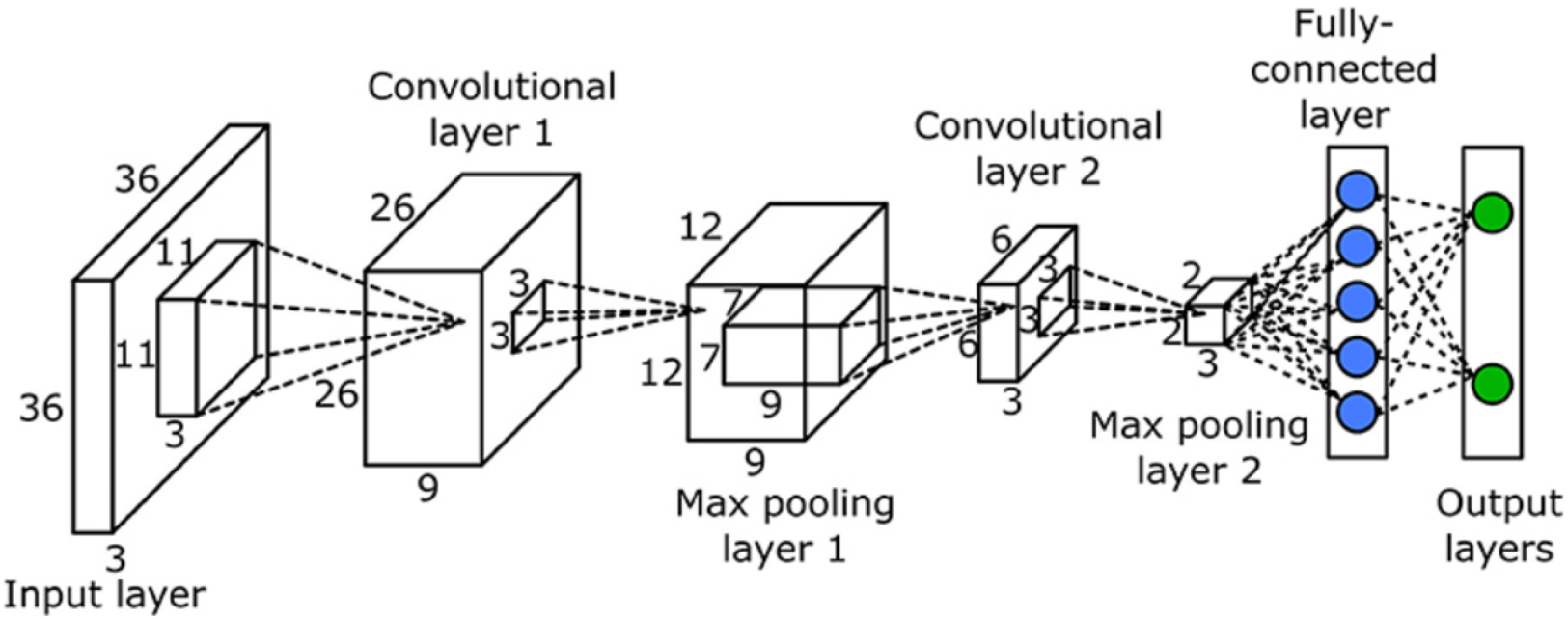


Classification — Fully Connected Layer (FC Layer)



Multi-Layer CNNs




36
11
11
36
3
3
Input layer
Convolutional
layer 1
26
3
3
26
9
12
7
7
12
9
9
Max pooling
layer 1
Convolutional
layer 2
6
3
3
6
3
Max pooling
layer 2
2
2
3
Fully-
connected
layer
Output
layers

Well known CNNs

There are various architectures of CNNs available which have been key in building algorithms which power and shall power AI as a whole in the foreseeable future.

1. LeNet
2. AlexNet
3. VGGNet
4. GoogLeNet
5. ResNet
6. ZFNet

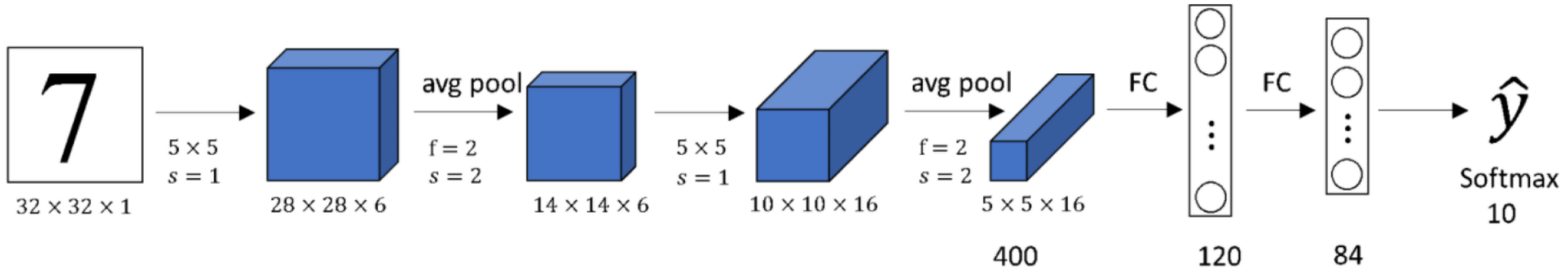
LeNet

LeNet is a convolutional neural network structure proposed by Yann LeCun et al. in 1998. In general, LeNet refers to lenet-5 and is a simple convolutional neural network.

LeNet5 was one of the earliest convolutional neural networks and promoted the development of deep learning. Since 1988, after years of research and many successful iterations, the pioneering work has been named LeNet5.

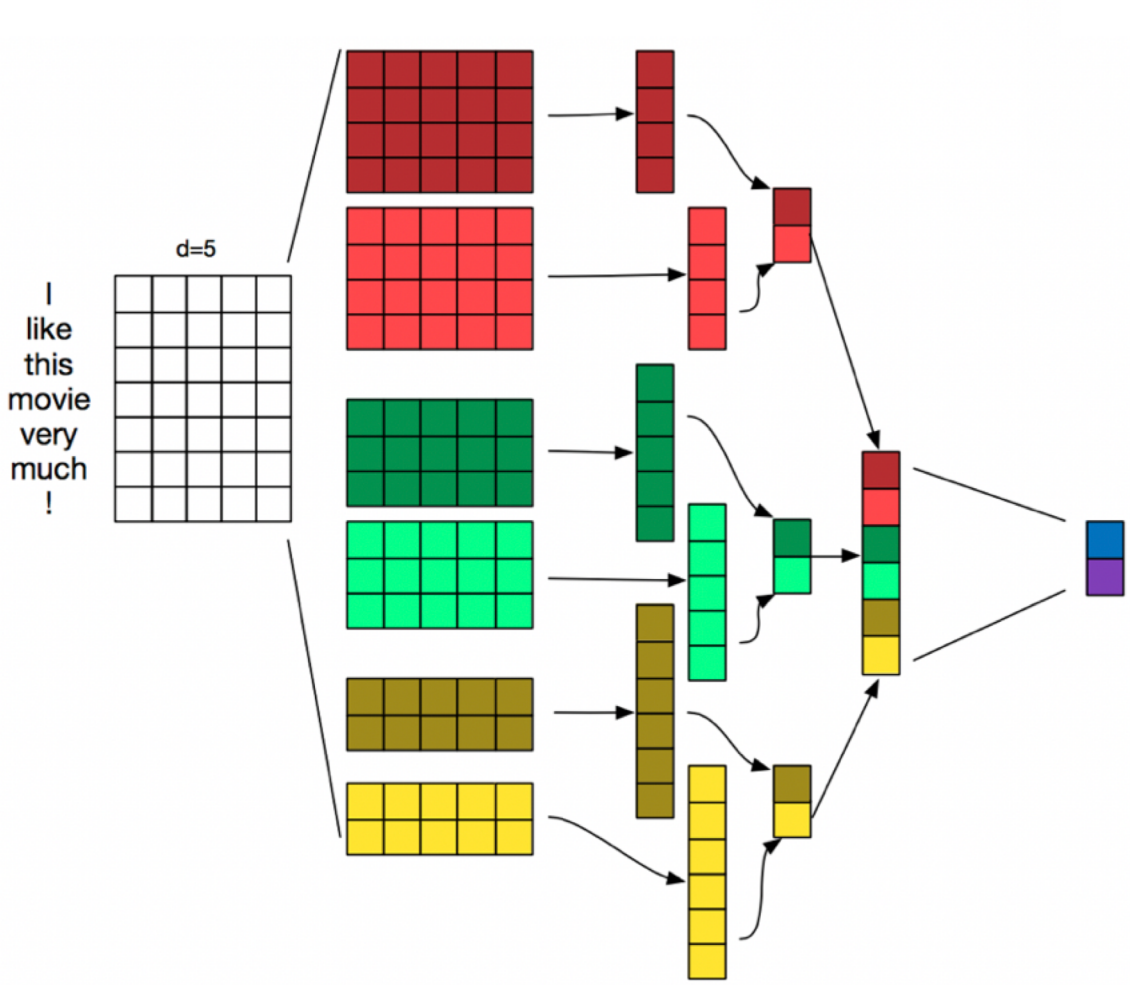
LeNet-5

http://datahacker.rs/deep-learning-lenet-5-architecture/



The goal of $LeNet{-}5$ was to recognize handwritten digits. So, it takes as an input $32{\times}32{\times}1$ image. It is a grayscale image, thus the number of channels is 1. Here is a picture of it's arhitecture. In the first step we use 65×5 filters with a stride s=1 and $nopadding$. Therefore we end up with a 28×28×6 volume. Notice that, because we are using s=1 and $nopadding$, the image dimensions reduce from 32×32 to 28×28.
Next, $LeNet$ applies $pooling$. When this paper was written $Averagepooling$ was much more in use, so here we will use $Averagepooling$. However, nowadays we would probably use $Maxpooling$ instead. So, here we will implement $Averagepool$ with filter f=2 and stride s=2. We get a 14×14×6 volume, so we reduced dimensions of an image by a factor of 2 and due to use of a stride of 2.

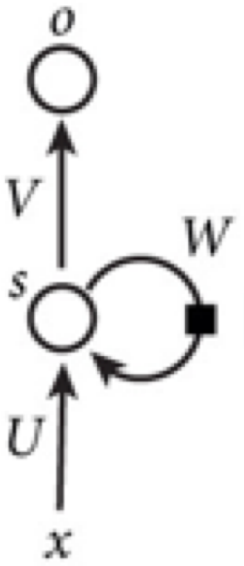
CNN for Natural Language Processing

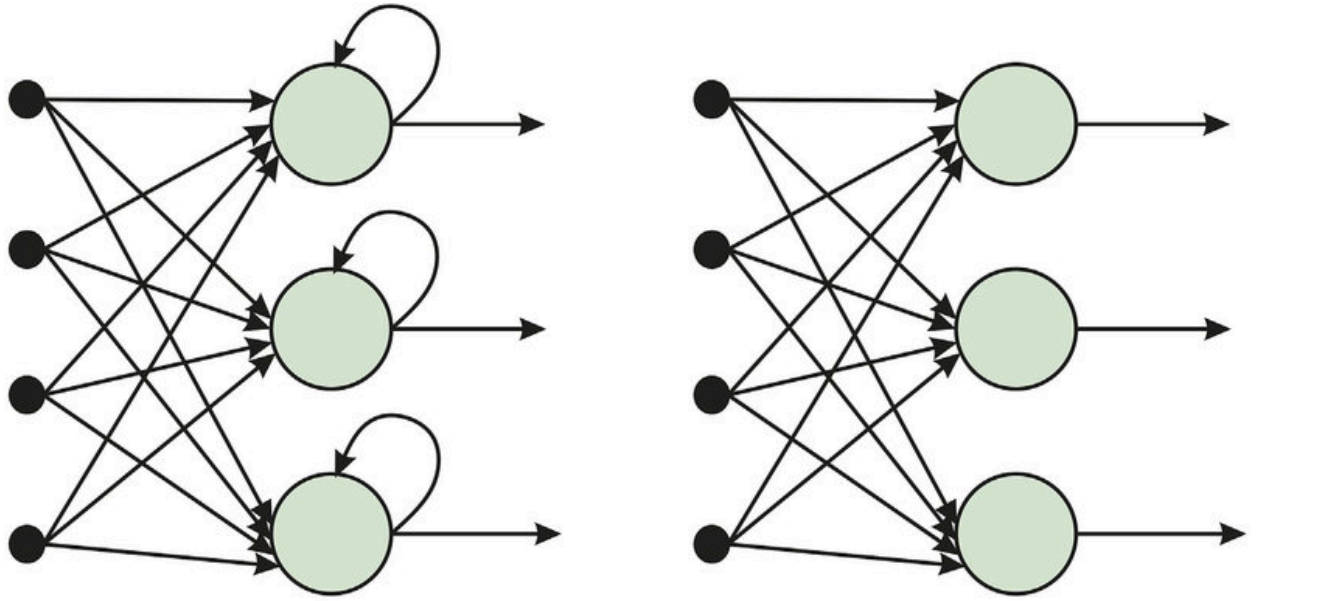


CNN Implementation

```
model = Sequential()

model.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Flatten())

model.add(Dense(units = 128, activation = 'relu'))
model.add(Dense(units = 1, activation = 'sigmoid'))

model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

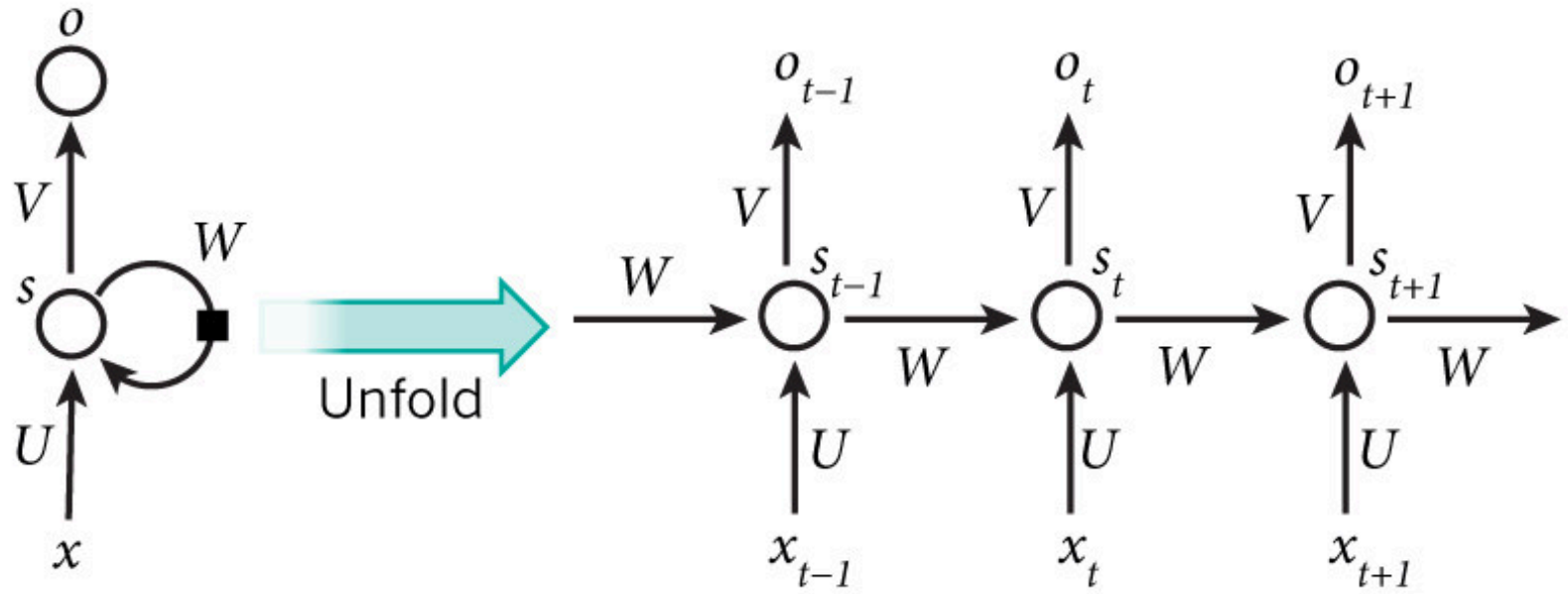
RECURRENT NEURAL NETWORKS



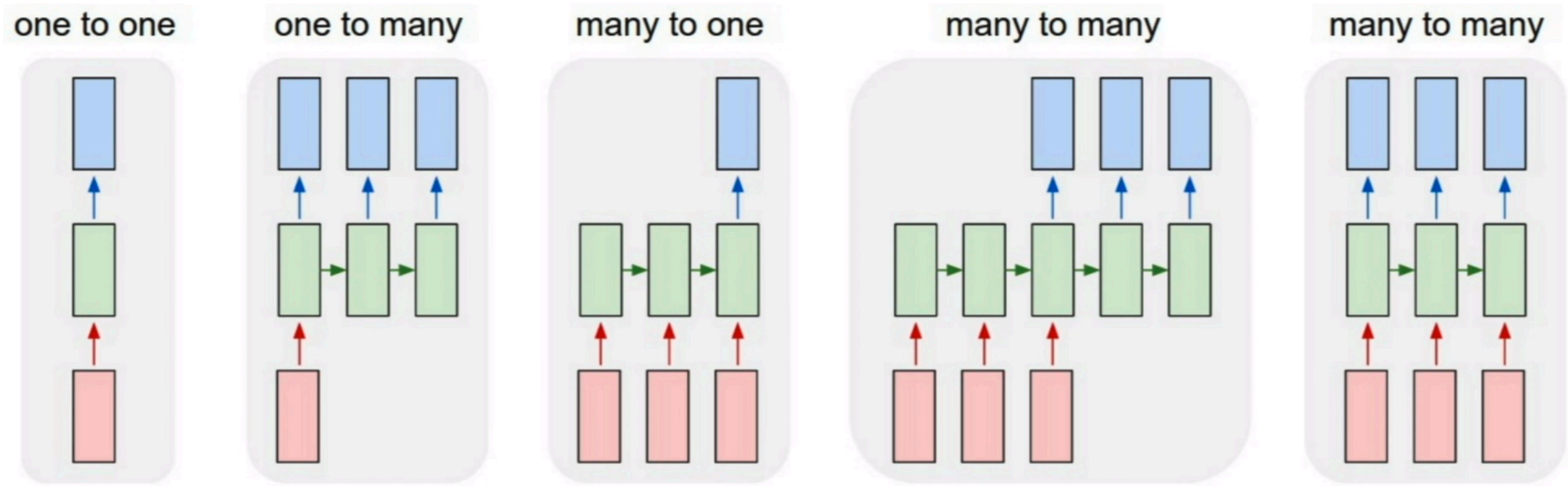


(a) Recurrent Neural Network (b) Feed-Forward Neural Network

RNN for Sequential Data



Different Types of RNN



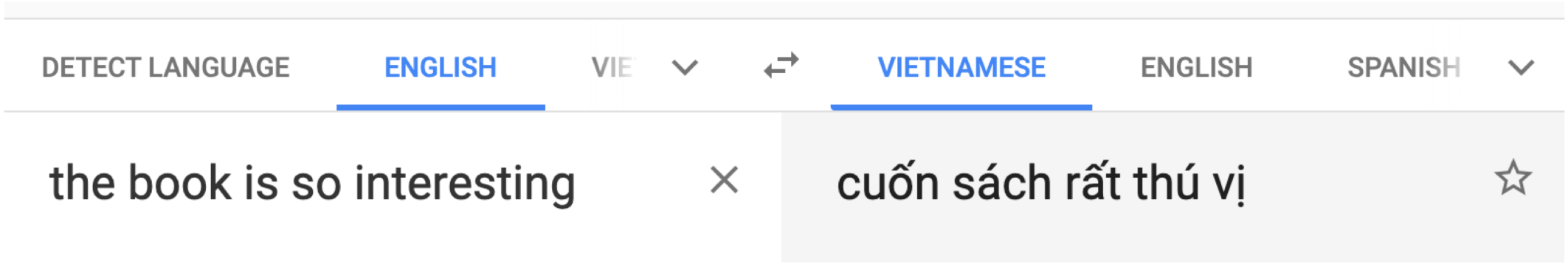
Outline

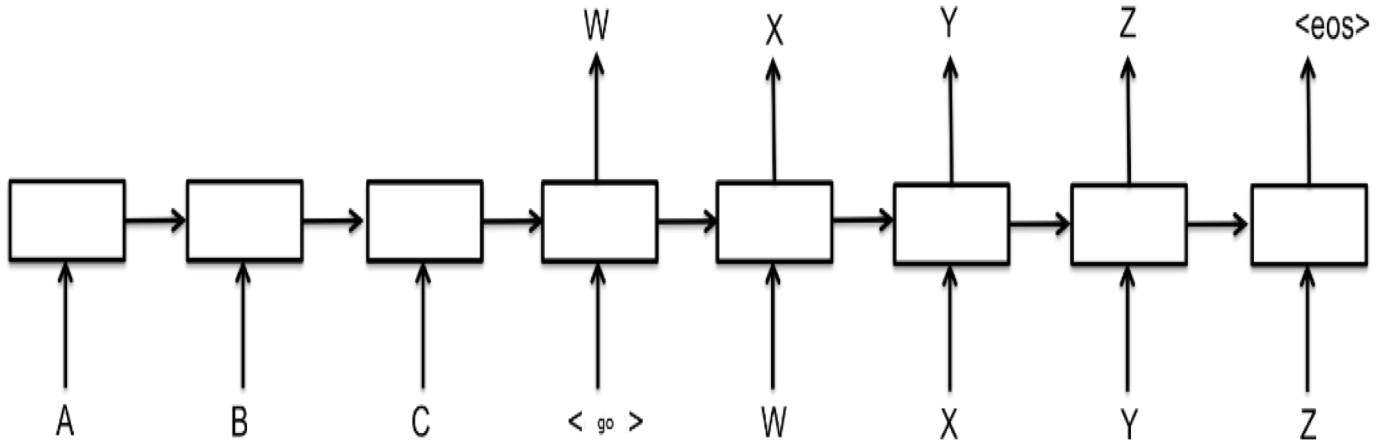
1. How to recognize Deep Learning networks/models:
   - Traditional statistical machine learning vs Deep Learning
   - Vanilla Neural Networks vs Deep Learning Networks
   - History of DL development
2. Convolutional Neural Network (CNN)
3. Reccurent Neural Network (RNN) and Long Short Term Memory (LSTM)
4. Transformer Models

Outline

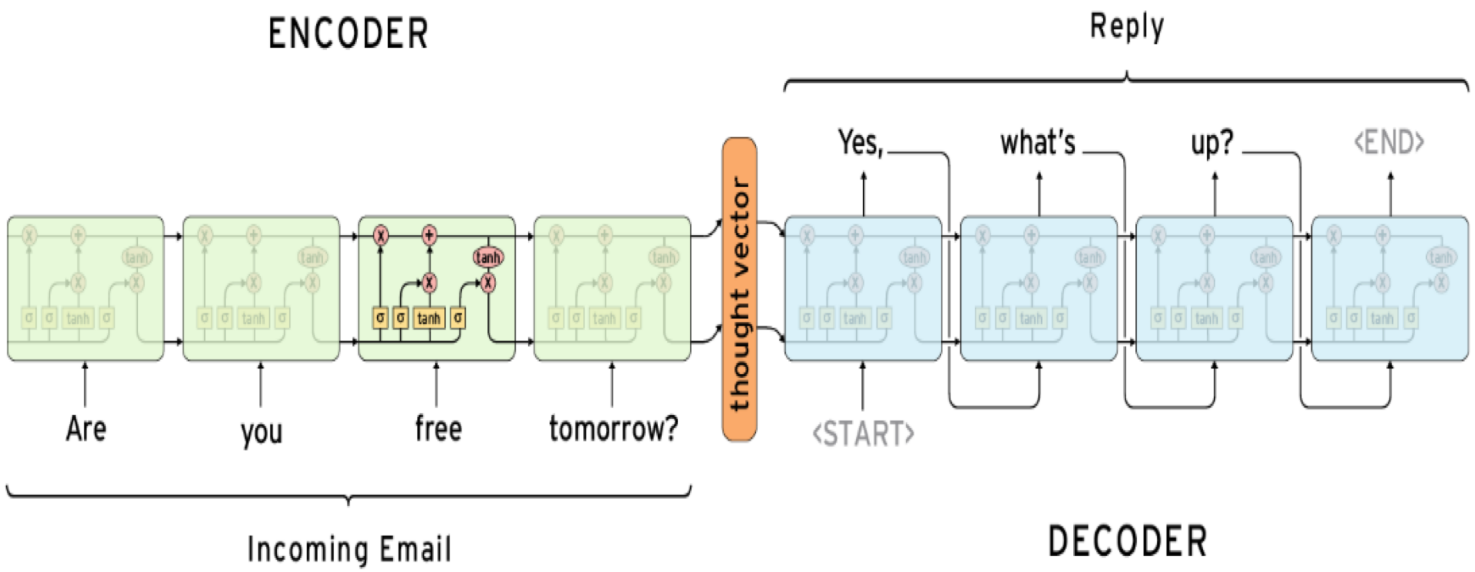
- What is RNN?
- RNN: the Architecture and Forward Computation
- RNN with Progapagation algorithm
- Long-Short Term Memory

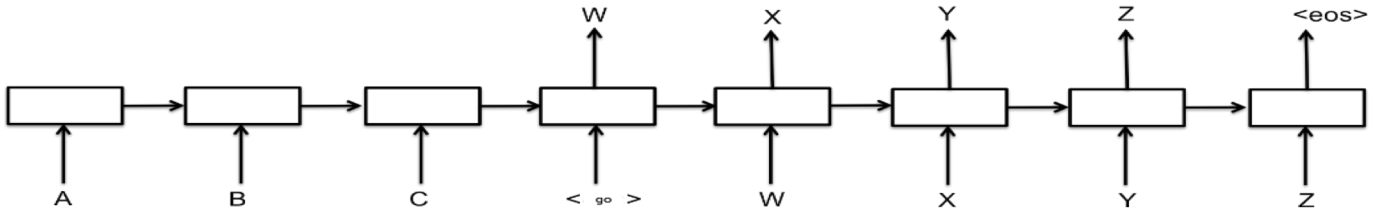
What is Recurrent Neural Network (RNN)?





Automatic Email Reply





Part Of Speech Tagging
Sentiment Analysis

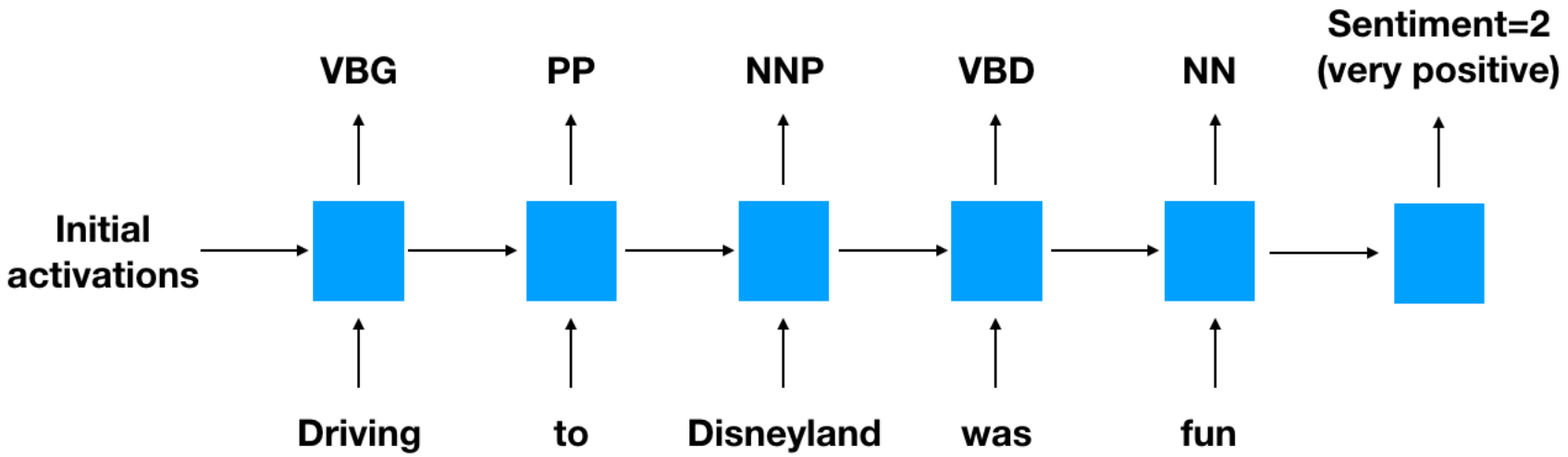
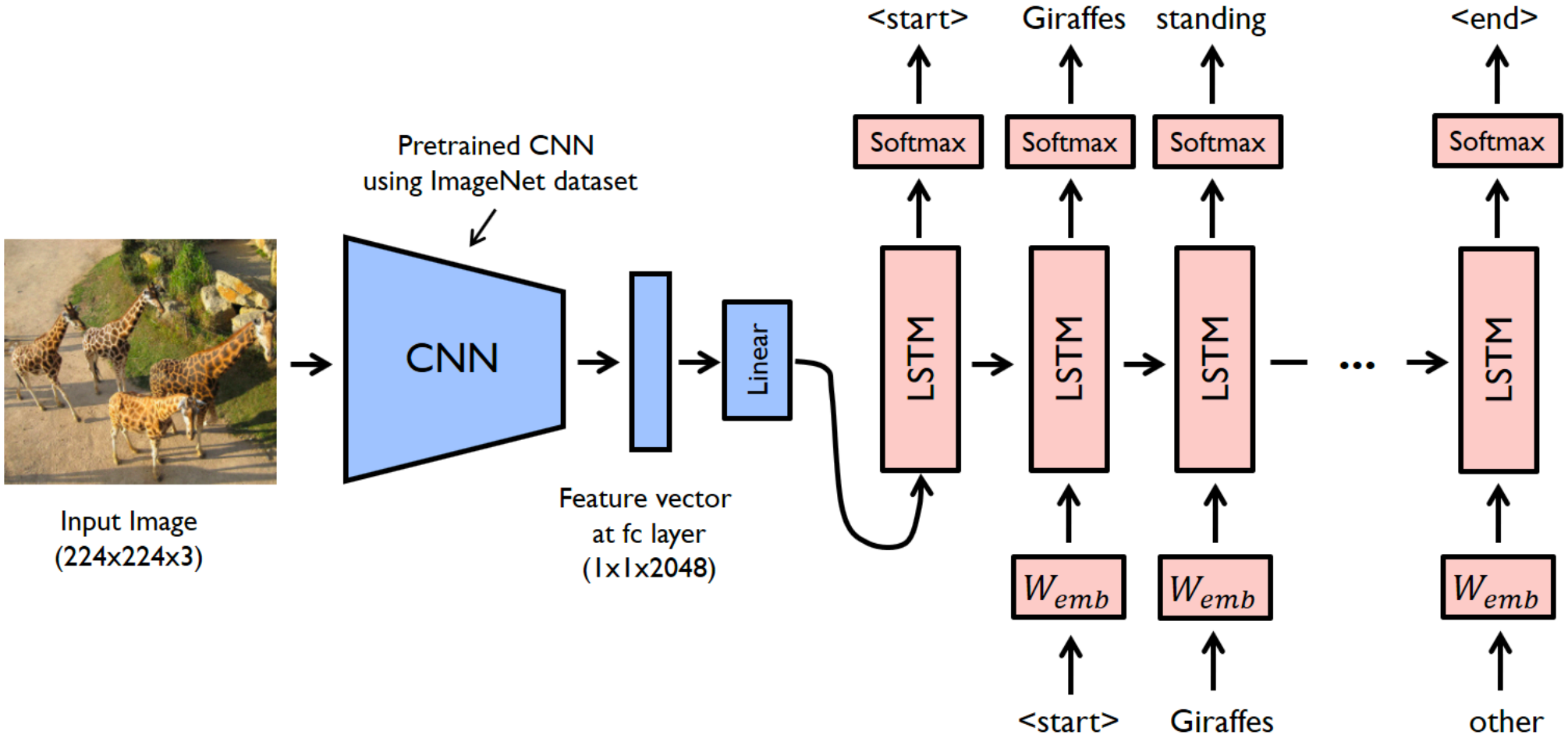
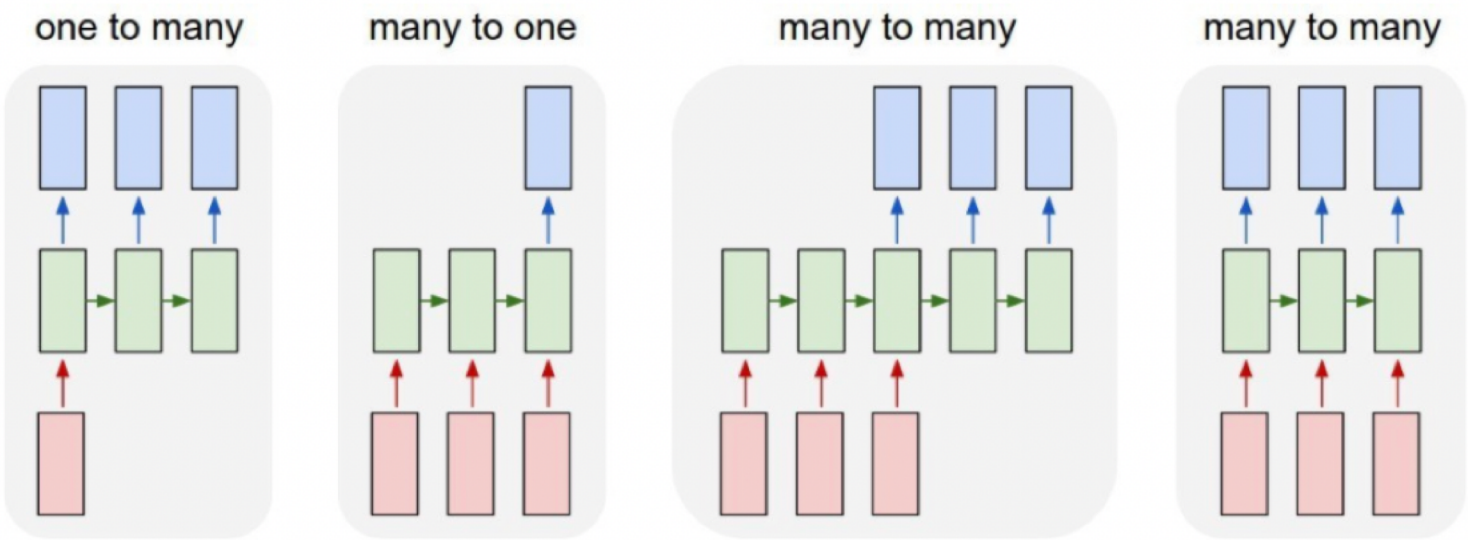


Image Captioning

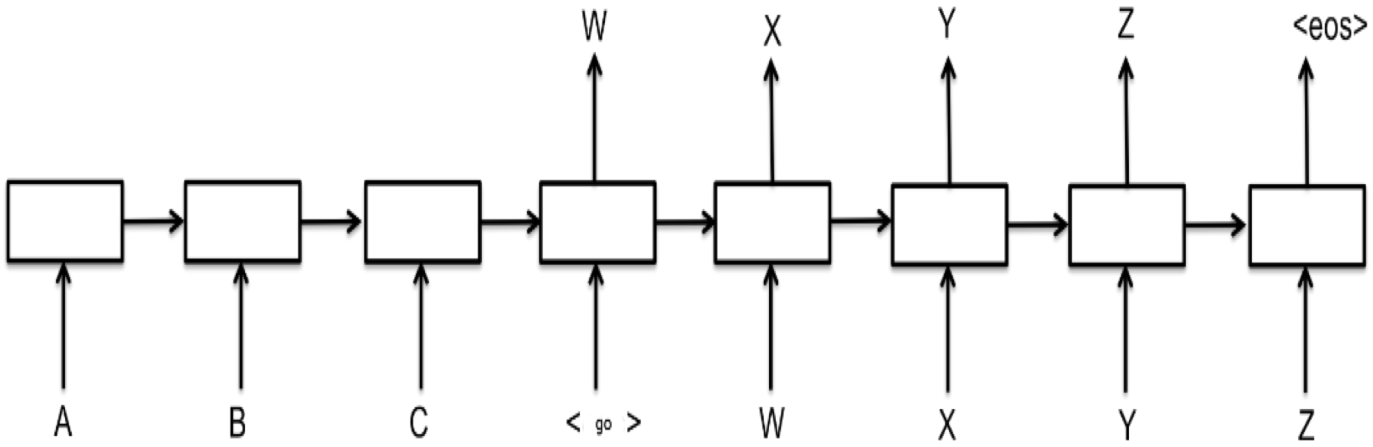


RNN Architectures

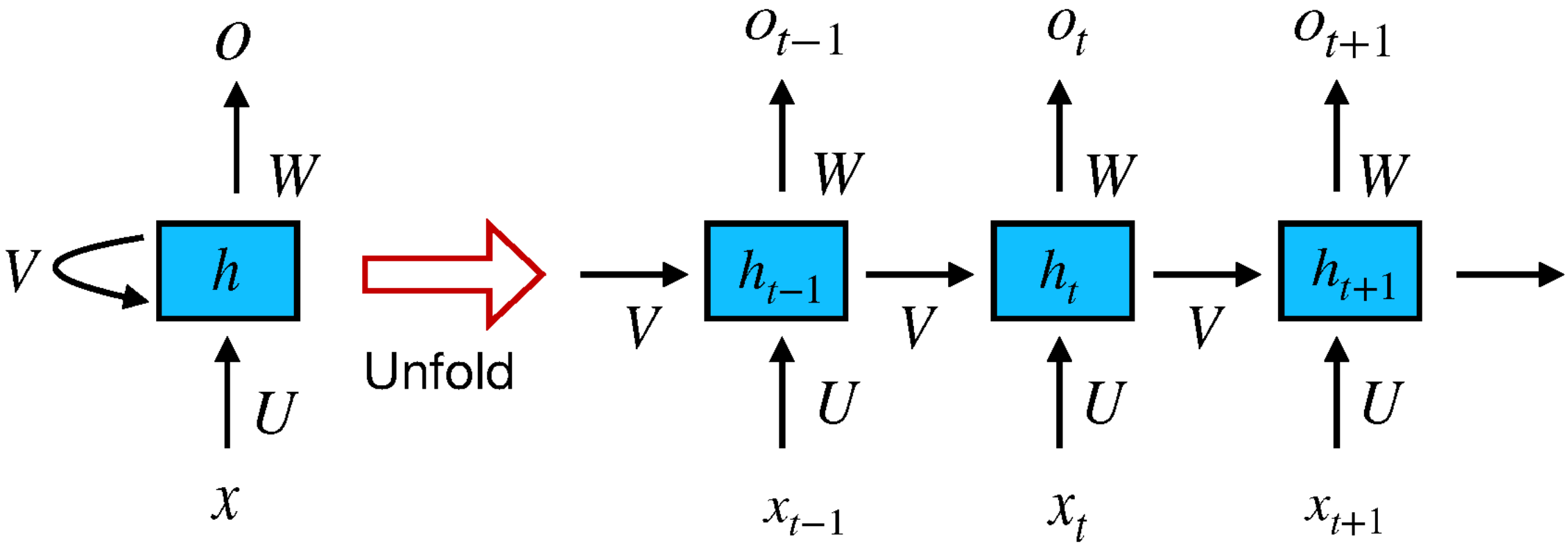


What is Recurrent Neural Network (RNN)?

- The idea behind RNNs is to make use of Sequential Information.
- RNNs are called *recurrent* because they perform the same task for every element of a sequence.
- RNNs have a "memory" which captures information about what has been calculated so far.

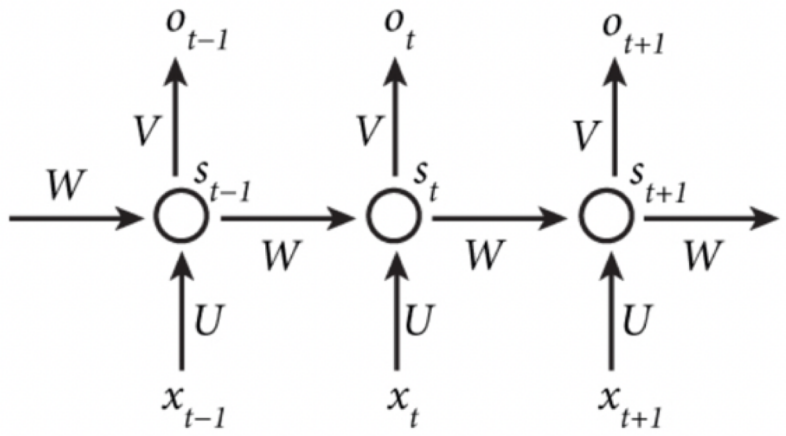


What is Recurrent Neural Network (RNN)?



Forward Computation in RNN

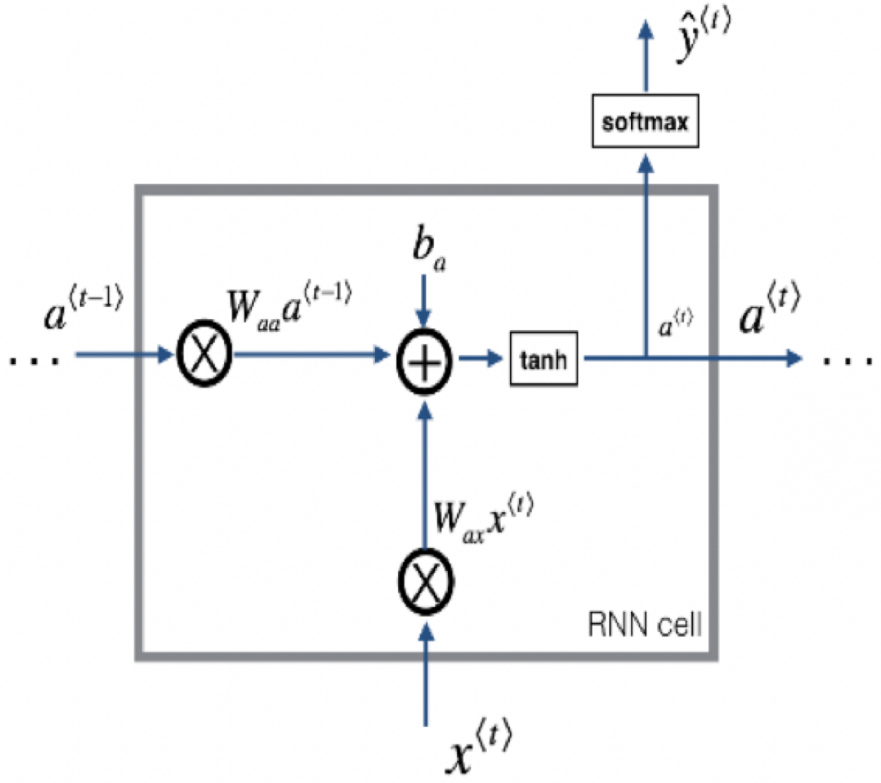
- x_t is the input at time step t. For example, x_1 could be a one-hot vector corresponding to the second word of a sentence.
- s_t is the hidden state at time step t. It's the "memory" of the network. s_t is calculated based on the previous hidden state and the input at the current step: $s_t = f(Ux_t + Ws_{t-1})$. The function f usually is a nonlinearity such as tanh or ReLU. s_{-1}, which is required to calculate the first hidden state, is typically initialized to all zeroes.
- o_t is the output at step t. For example, if we wanted to predict the next word in a sentence it would be a vector of probabilities across our vocabulary. $o_t = \text{softmax}(Vs_t)$.



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

$$o_t = \text{softmax}(Vs_t).$$

Forward Computation in RNN



$$a^{\langle t \rangle} = \tanh(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_a)$$

$$\hat{y}^{\langle t \rangle} = soft\max(W_{ya}a^{\langle t \rangle} + b_y)$$

Example

https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/

https://medium.com/towards-artificial-intelligence/whirlwind-tour-of-rnns-a11effb7808f

1	0	0	0
0	1	0	0
0	0	1	1
0	0	0	0
h	e	l	l

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

wxh			
0.287027	0.84606	0.572392	0.486813
0.902874	0.871522	0.691079	0.18998
0.537524	0.09224	0.558159	0.491528

×

1
0
0
0
h

=

0.287027
0.902874
0.537524

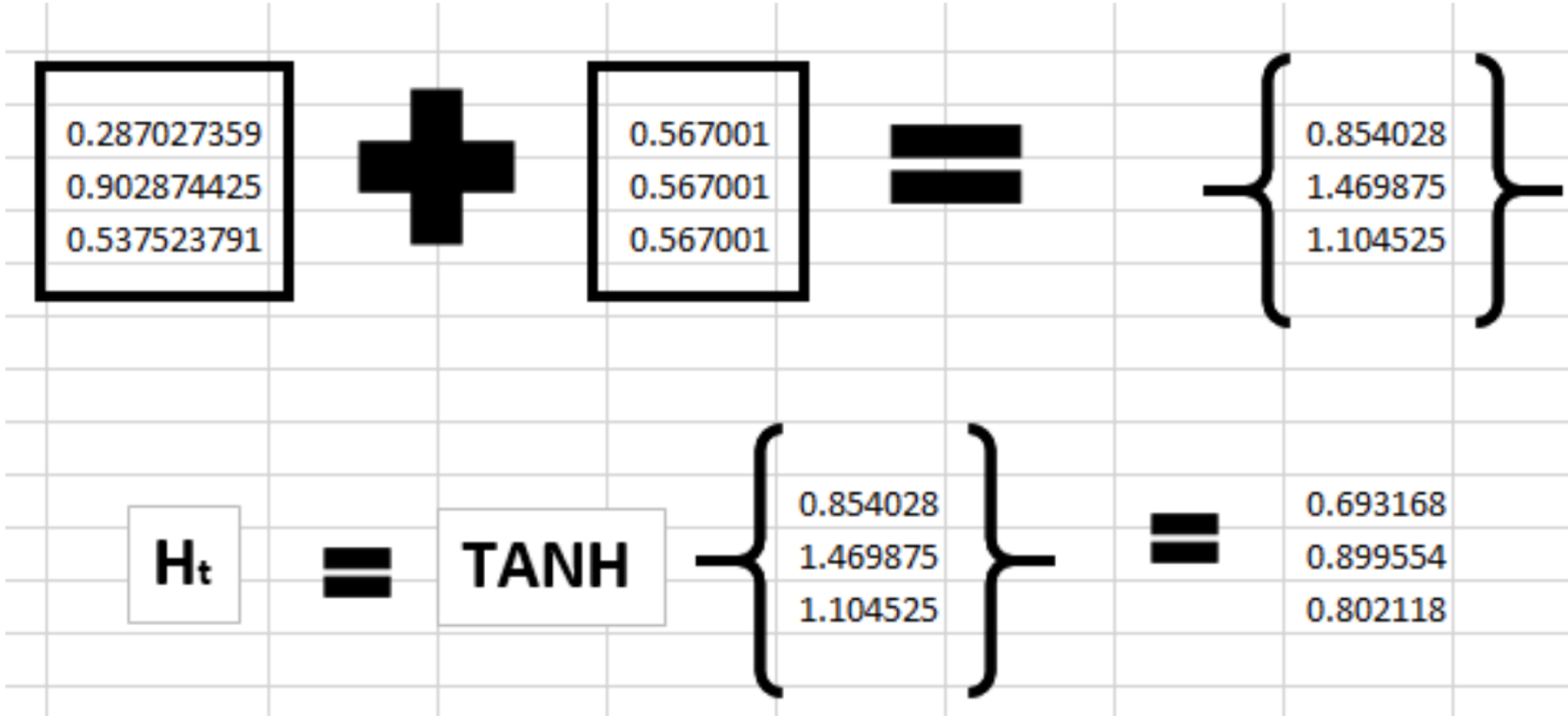
Weight(whh)	bias
0.427043	0.567001

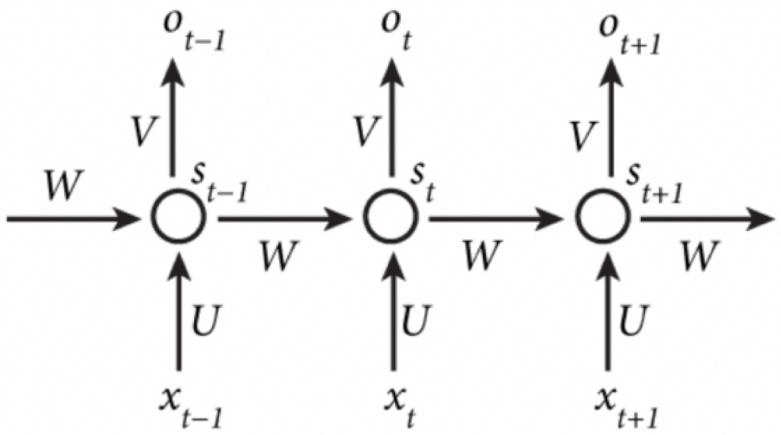
×

0
0
0
h_{t-1}

=

0.567001
0.567001
0.567001


0.287027359
0.902874425
0.537523791
0.567001
0.567001
0.567001
0.854028
1.469875
1.104525
Ht
TANH
0.854028
1.469875
1.104525
0.693168
0.899554
0.802118



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

$$o_t = \text{softmax}(Vs_t).$$

$$a_1 = \begin{pmatrix} 0.1 & 0.5 & 0.1 \\ 0.5 & 0.9 & 0.3 \\ 0.3 & 0.2 & 0.1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0.6 & 0.8 & 0.4 & 0.8 \\ 0.2 & 0.2 & 0.8 & 0.7 \\ 0.9 & 0.8 & 0.1 & 0.2 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.6 \\ 0.2 \\ 0.9 \end{pmatrix}$$

$$h_1 = tanh(\begin{pmatrix} 0.6 \\ 0.2 \\ 0.9 \end{pmatrix}) = \begin{pmatrix} 0.54 \\ 0.20 \\ 0.72 \end{pmatrix}$$

$$y_1 = softmax(\begin{pmatrix} 0.9 & 0.8 & 0.3 \\ 0.2 & 0.3 & 0.4 \\ 0.6 & 0.9 & 0.1 \\ 0.5 & 0.0 & 0.3 \end{pmatrix} \begin{pmatrix} 0.54 \\ 0.20 \\ 0.72 \end{pmatrix}) = \begin{pmatrix} 0.32 \\ 0.21 \\ 0.24 \\ 0.22 \end{pmatrix}$$

BackPropagation Through Time (BPTT)

The *loss*, or error, to be
the cross entropy loss, given by:

$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = \mathrm{softmax}(Vs_t)$$

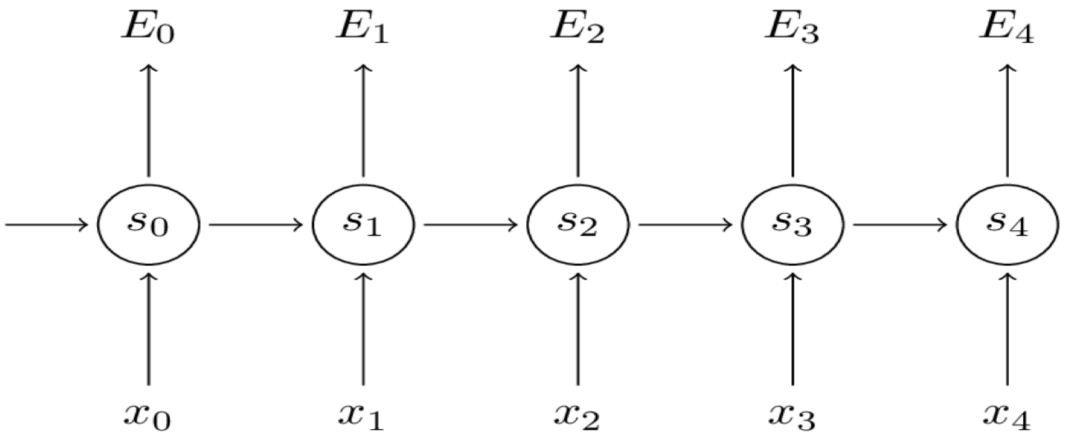
$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$
$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t)$$
$$= -\sum_t y_t \log \hat{y}_t$$

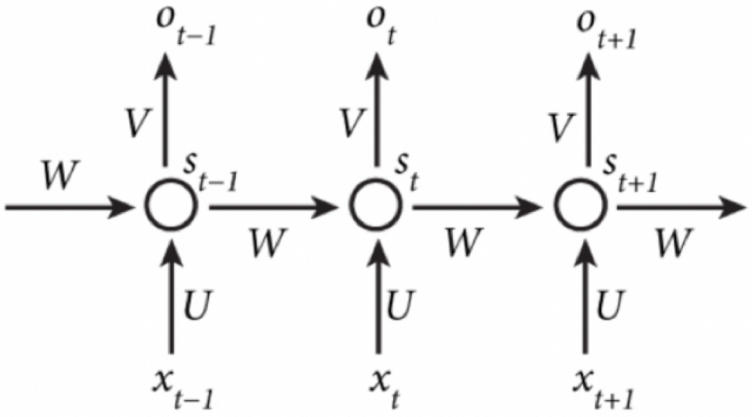
The cross entropy formula takes in two distributions, $p(x)$, the true distribution, and $q(x)$, the estimated distribution, defined over the discrete variable x and is given by

$$H(p, q) = -\sum_{\forall x} p(x) \log(q(x))$$

BackPropagation Through Time (BPTT)

Remember that our goal is to calculate the gradients of the error with respect to our parameters U, V and W and then learn good parameters using Stochastic Gradient Descent. Just like we sum up the errors, we also sum up the gradients at each time step for one training example: $\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$.



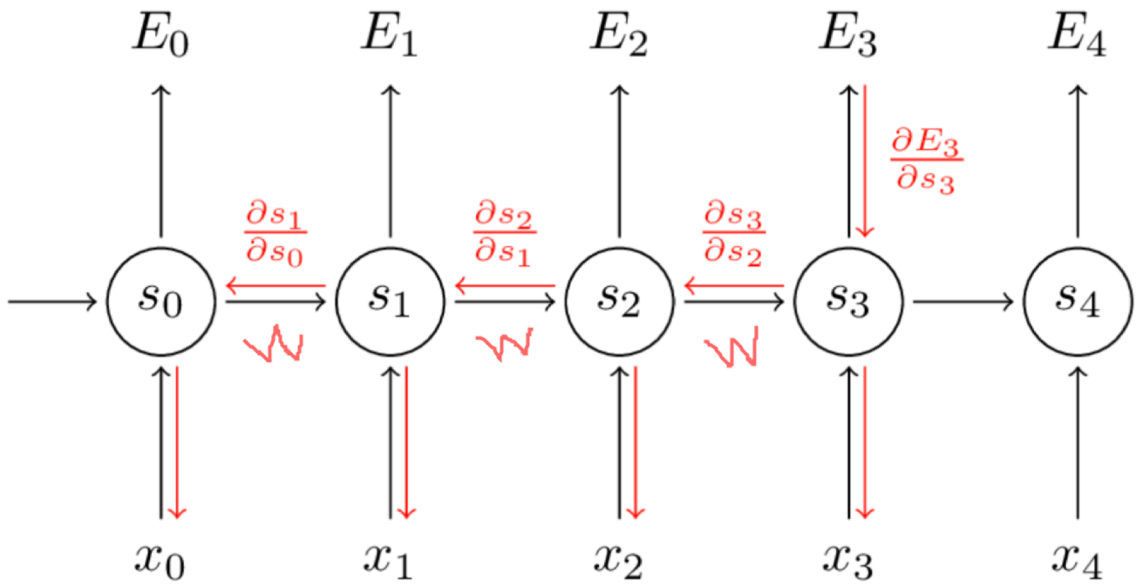


$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = \text{softmax}(Vs_t)$$

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$

$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t)$$

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$



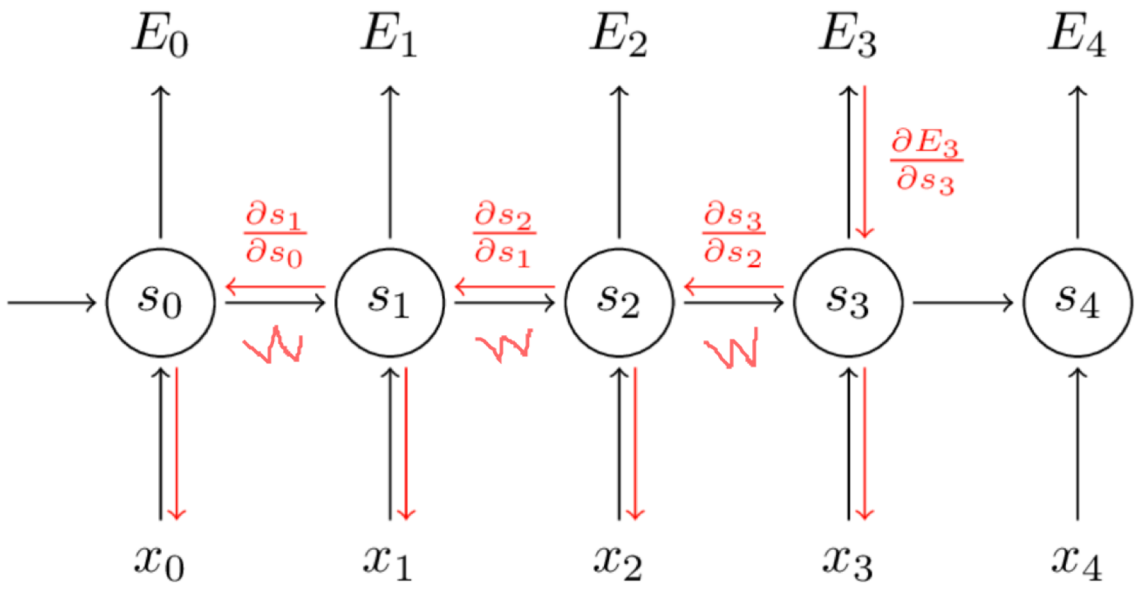
Note that $\frac{\partial s_3}{\partial s_k}$ is a chain rule in itself! For example,

$$\frac{\partial s_3}{\partial s_1} = \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1}.$$

$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = \text{softmax}(Vs_t)$$

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$

$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t)$$



$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$

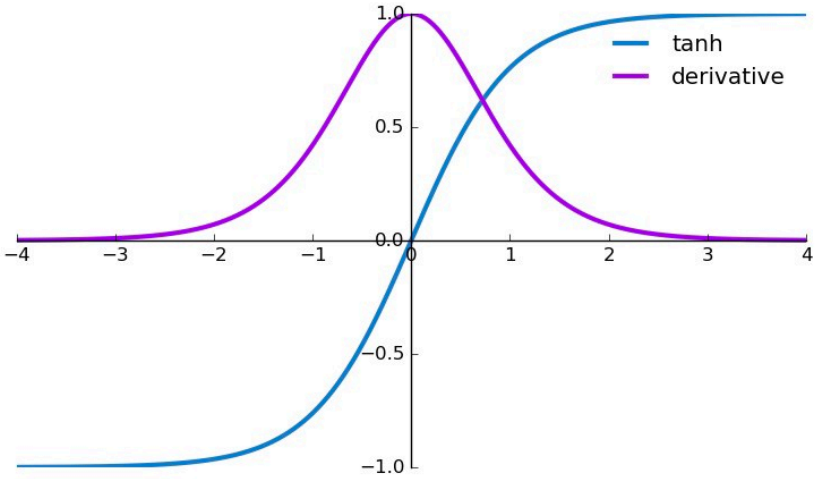
$$\Rightarrow \quad \frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \left(\prod_{j=k+1}^{3} \frac{\partial s_j}{\partial s_{j-1}} \right) \frac{\partial s_k}{\partial W}$$

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

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \left(\prod_{j=k+1}^{3} \frac{\partial s_j}{\partial s_{j-1}} \right) \frac{\partial s_k}{\partial W}$$

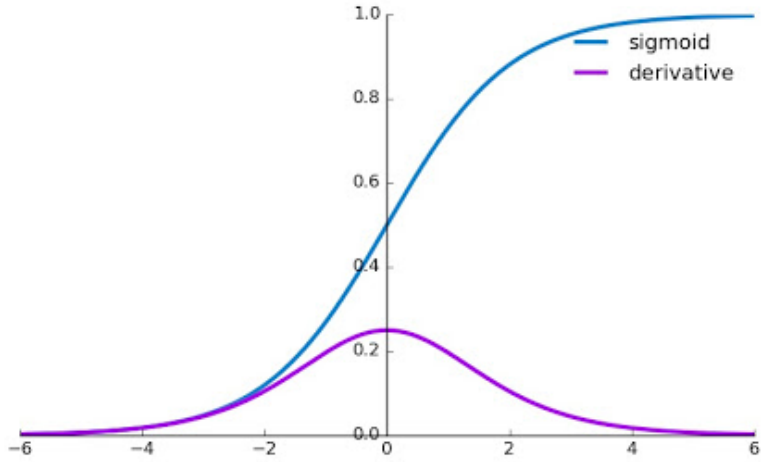
$$\frac{\partial s_j}{\partial s_{j-1}} = \tanh' * W$$

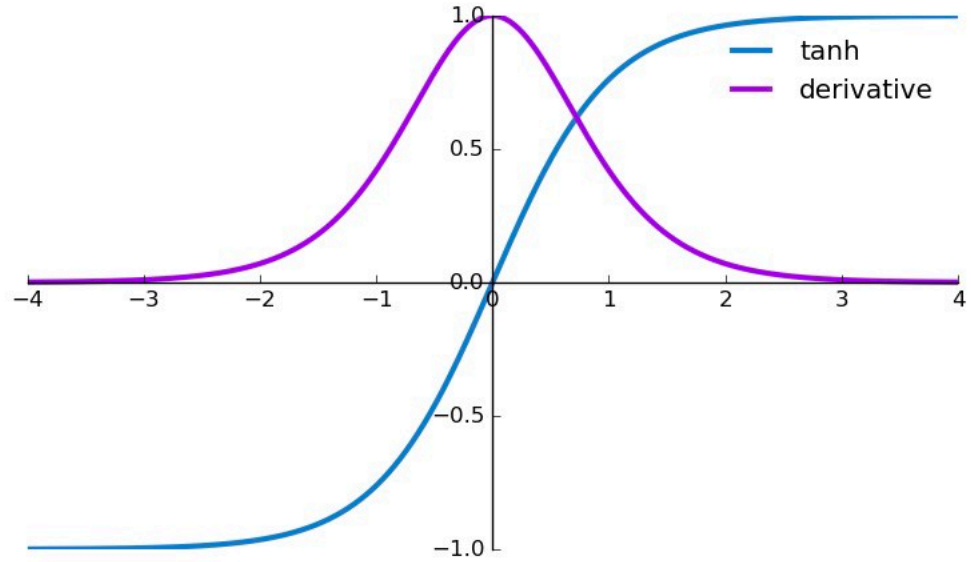
$$so \ \frac{\partial E_k}{\partial W} \longrightarrow 0$$



W ~ small -> Vanishing problem

Sigmoid and Tanh funtions





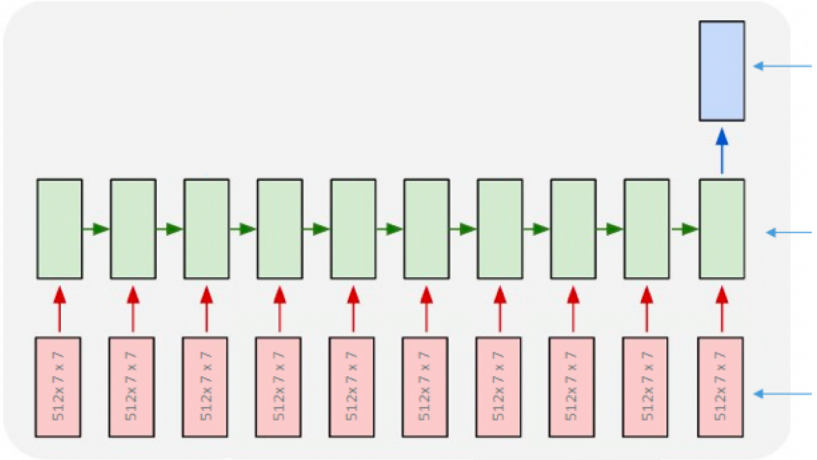
RNN: Vanishing Problem

"During back propagation, recurrent neural networks suffer from the vanishing gradient problem. Gradients are values used to update a neural networks weights.
The vanishing gradient problem is when the gradient shrinks as it back propagates through time. If a gradient value becomes extremely small, it doesn't contribute too much learning."

Limitation of RNN

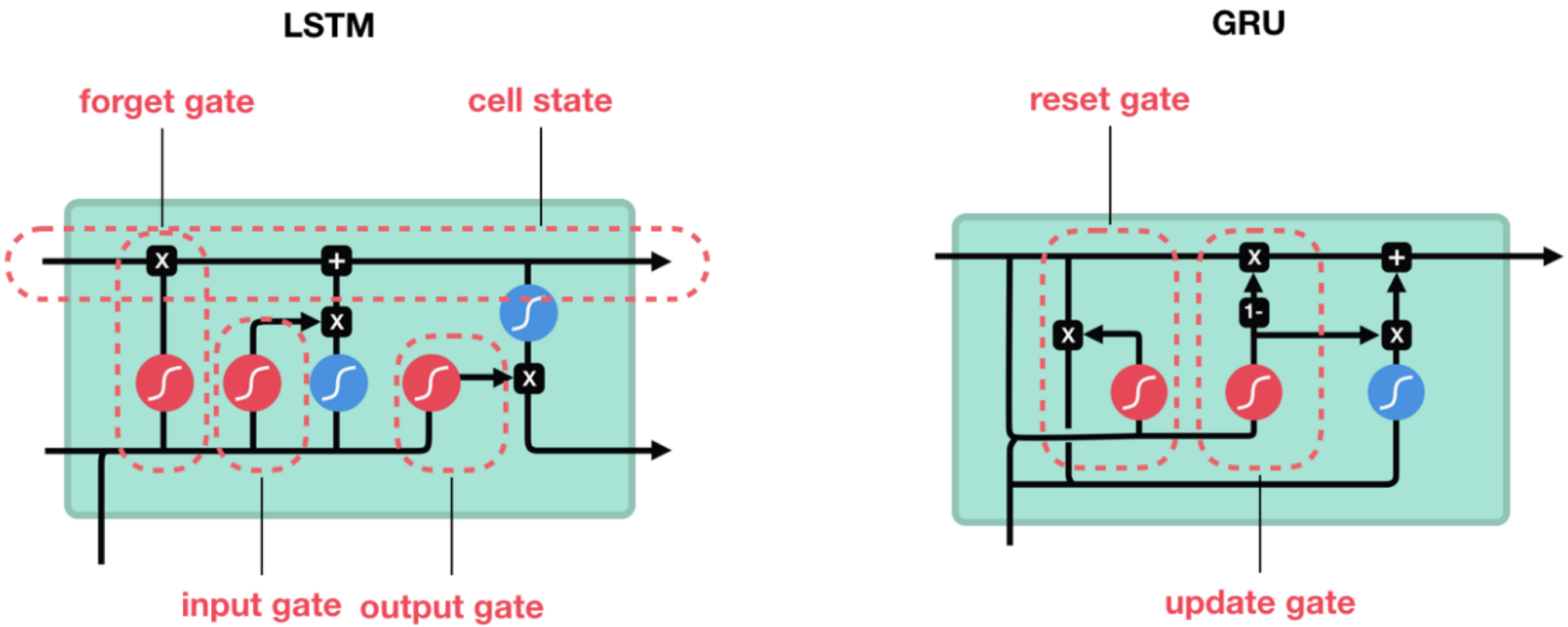
consider the sentence:

“ I grew up in France … I speak fluent French”



LSTM's and GRU's as a Solution

These gates can learn which data in a sequence is important to keep or throw away



LSTM Networks

- Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning Long-Term Dependencies.
- They were introduced by Hochreiter & Schmidhuber (1997), and were refined and popularized by many people, they work tremendously well on a large variety of problems, and are now widely used.
- LSTMs are explicitly designed to avoid the long-term dependency problem.

Customers Review 2,491

Thanos

September 2018

Verified Purchase

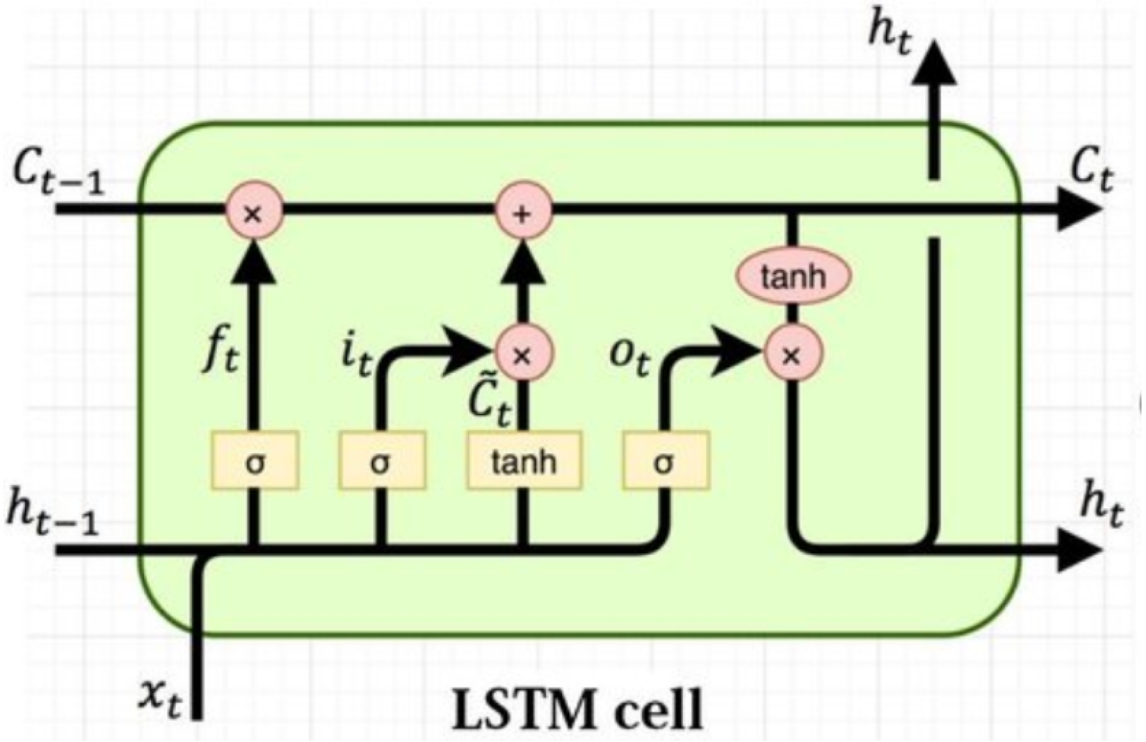
Amazing! This box of cereal gave me a perfectly balanced breakfast, as all things should be. I only ate half of it but will definitely be buying again!

When you read the review, your brain subconsciously only remembers important keywords. You pick up words like “amazing” and “perfectly balanced breakfast”. You don’t care much for words like “this”, “gave“, “all”, “should”, etc.

Longer Memories through LSTMs

- **Adding a forgetting mechanism.**
- **Adding a saving mechanism.**
  - When the model sees a new image, it needs to learn whether any information about the image is worth using and saving.

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_h(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

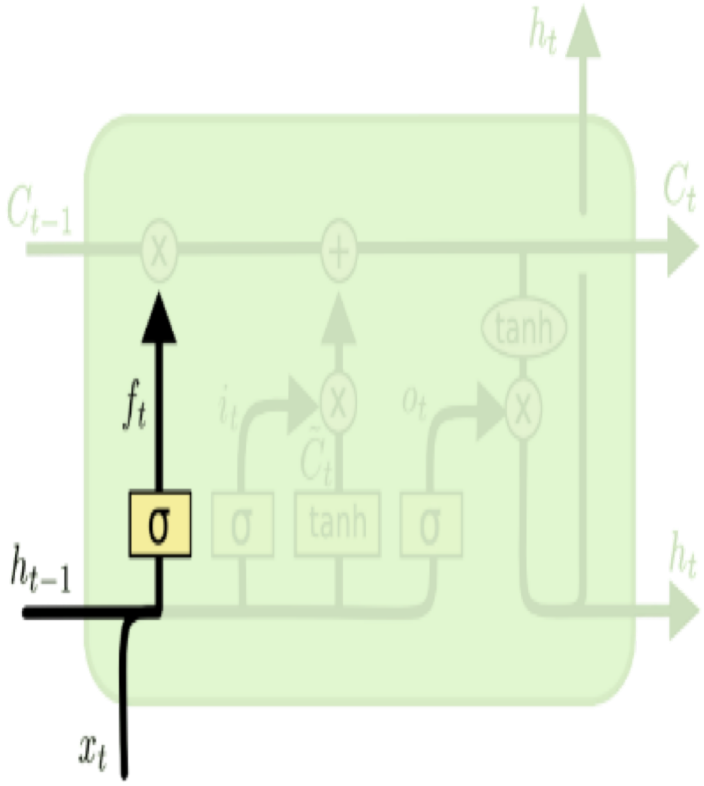


where the initial values are $c_0 = 0$ and $h_0 = 0$ and the operator $\circ$ denotes the Hadamard product (element-wise product). The subscript t indexes the time step.

https://en.wikipedia.org/wiki/Long_short-term_memory

LSTM Cell

- Forget gate

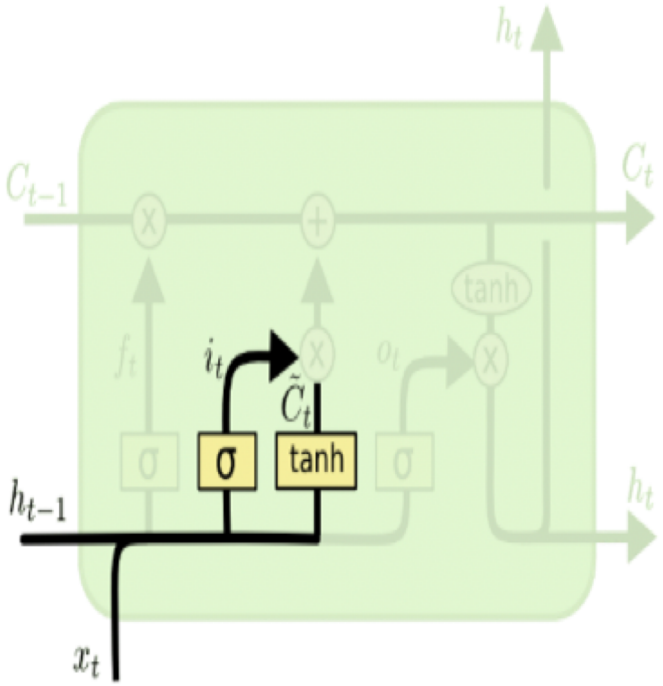


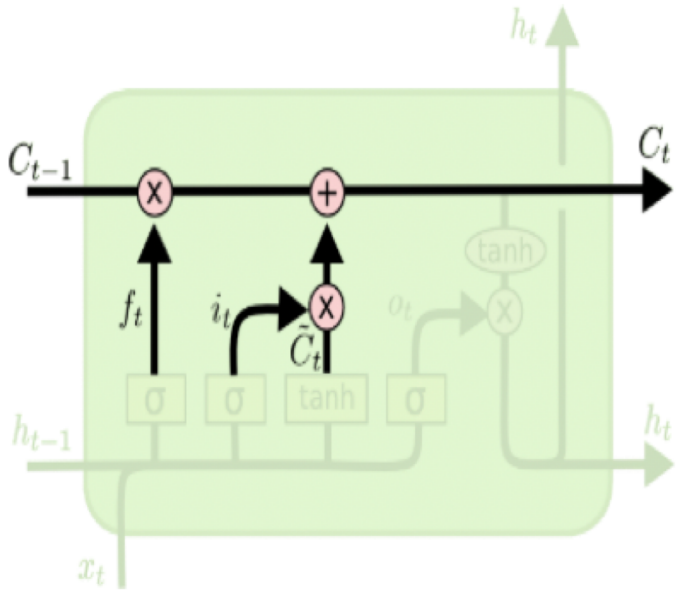
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

- Input Gate

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$

$$\tilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right)$$





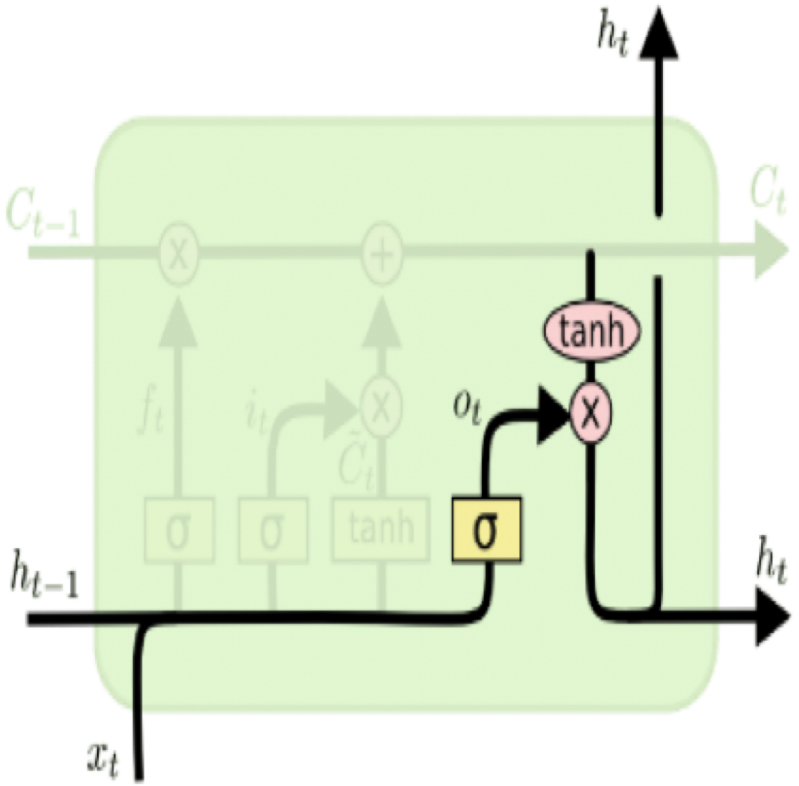
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Output Gate

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

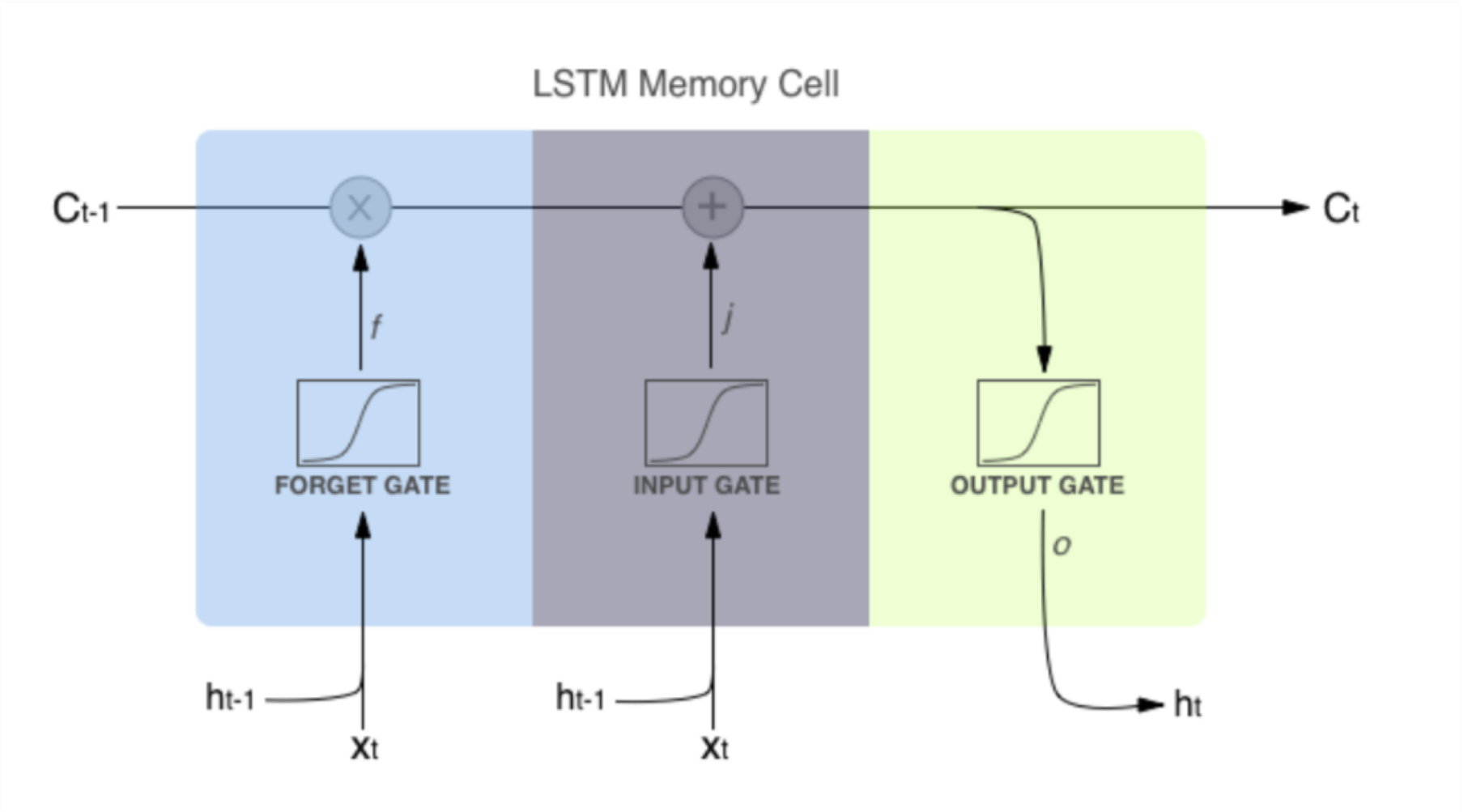
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] \; + \; b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

LSTM Memory Cell



How does LSTM prevent Vanishing

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t]\right)$$

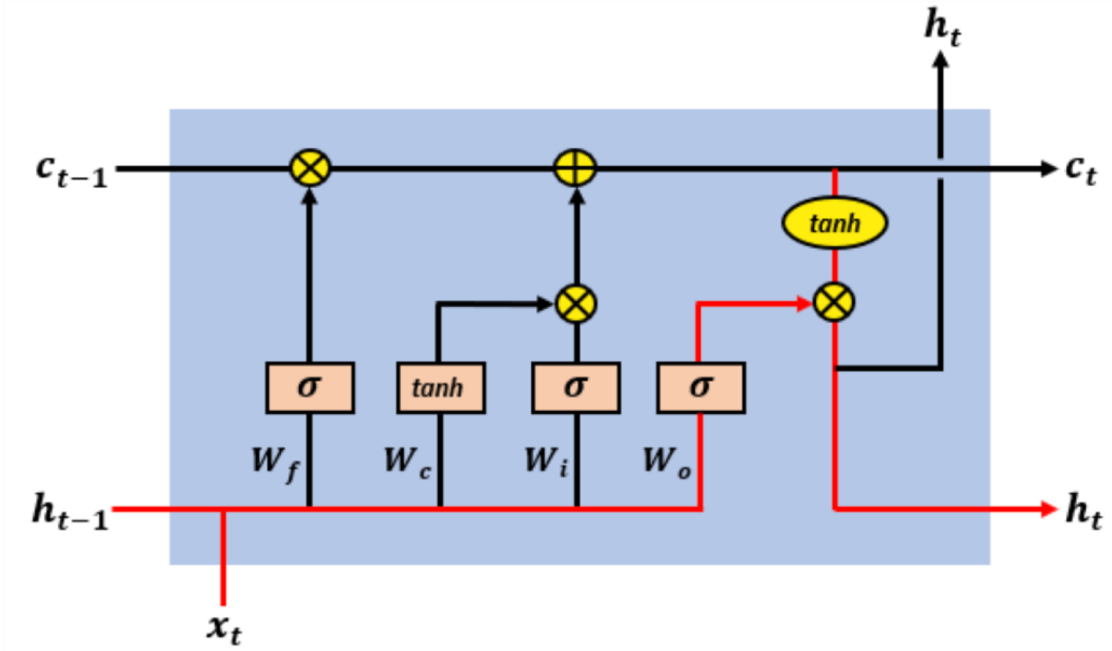
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t])$$

$$\tilde{c}_t = tanh\ (W_c \cdot [h_{t-1}, x_t])$$

$$c_t = c_{t-1} \otimes f_t \oplus \tilde{c}_t \otimes i_t$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t])$$

$$h_t = o_t \otimes tanh(c_t)$$



$$tanh(W_c \cdot [h_{t-1}, x_t]) \otimes \sigma(W_i \cdot [h_{t-1}, x_t])$$

BackPropagation Through Time in LSTM

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W} \qquad (3)$$

The gradient of the error in an LSTM

$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial h_k} \frac{\partial h_k}{\partial c_k} \dots \frac{\partial c_2}{\partial c_1} \frac{\partial c_1}{\partial W}$$

$$= \frac{\partial E_k}{\partial h_k} \frac{\partial h_k}{\partial c_k} \left(\prod_{t=2}^{k} \frac{\partial c_t}{\partial c_{t-1}} \right) \frac{\partial c_1}{\partial W} \qquad (4)$$

Backpropagation through time in LSTM

In an LTSM, the state vector c(t), has the form:

$$c_t = c_{t-1} \otimes \sigma(W_f \cdot [h_{t-1}, x_t]) \oplus$$

$$tanh\,(W_c \cdot [h_{t-1}, x_t]) \otimes \sigma(W_i \cdot [h_{t-1}, x_t])$$

which can be written compactly as

$$c_t = c_{t-1} \otimes f_t \oplus \tilde{c}_t \otimes i_t \qquad (5)$$

$$c_t = c_{t-1} \otimes f_t \oplus \tilde{c}_t \otimes i_t$$

vanilla RNN

$$C_t = tanh(W_c C_{t-1} + W_x X_t)$$

Compute the derivative of (5) and get:

$$\frac{\partial c_t}{\partial c_{t-1}} = \frac{\partial}{\partial c_{t-1}} [c_{t-1} \otimes f_t \oplus \tilde{c}_t \otimes i_t]$$

$$= \frac{\partial}{\partial c_{t-1}} [c_{t-1} \otimes f_t] + \frac{\partial}{\partial c_{t-1}} [\tilde{c}_t \otimes i_t]$$

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

Backpropagating through time for gradient computation

$$\frac{\partial C_t}{\partial C_{t-1}} \approx \sigma(W_f \cdot [H_{t-1}, X_t]) \qquad (2)$$

Now, notice equation (2) means that **the gradient behaves similarly to the forget gate**, and if the forget gate decides that a certain piece of information should be remembered, it will be open and have values closer to 1 to allow for information flow.
For simplicity, we can think of the forget gate's action as:

$$\sigma(W_f \cdot [H_{t-1}, X_t]) \approx \vec{1}$$

So we get:

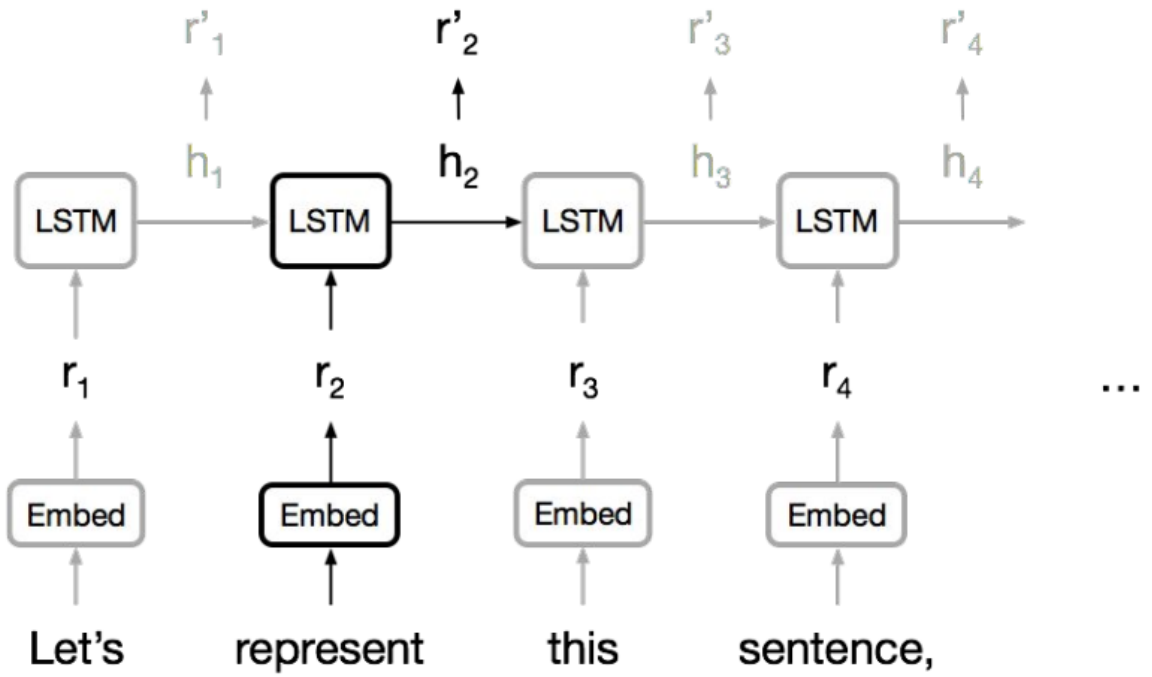
$$\frac{\partial C_t}{\partial C_{t-1}} \nrightarrow 0$$

Finally:

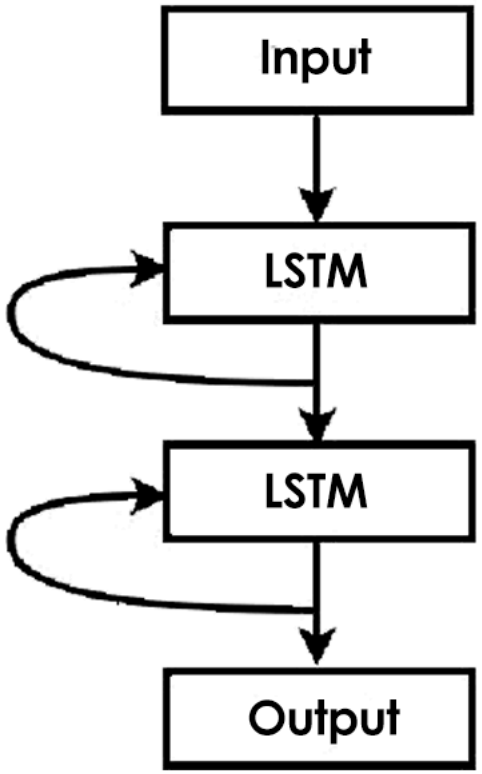
$$\frac{\partial E_k}{\partial W} \approx \frac{\partial E_k}{\partial H_k}\frac{\partial H_k}{\partial C_k}\left(\prod_{t=2}^{k} \sigma(W_f \cdot [H_{t-1}, X_t])\right)\frac{\partial C_1}{\partial W} \nrightarrow 0$$

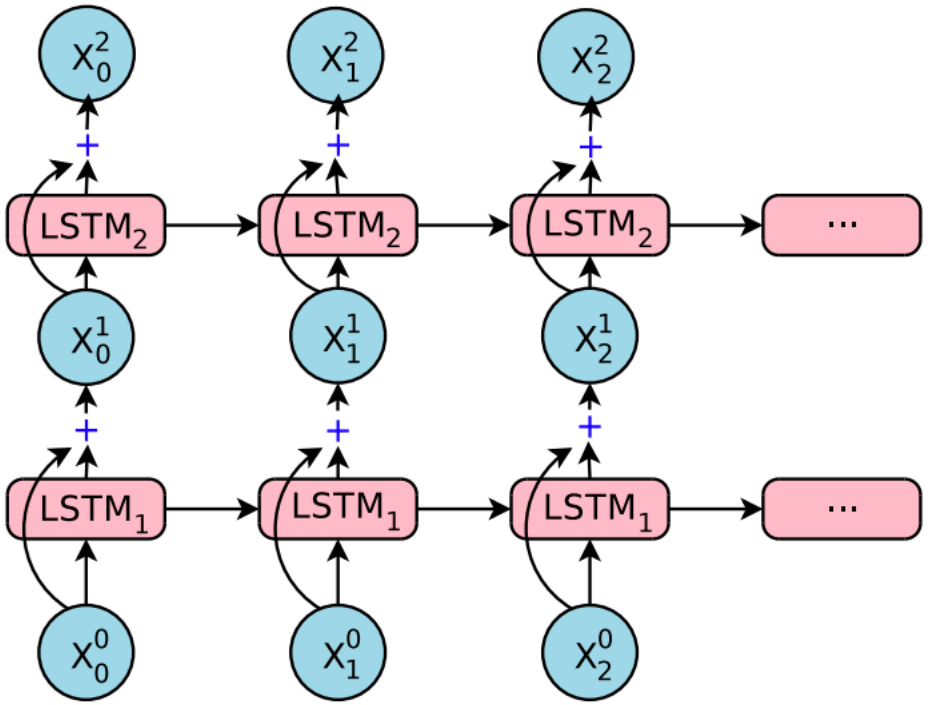
And the gradients do not vanish!

LSTM Architecture

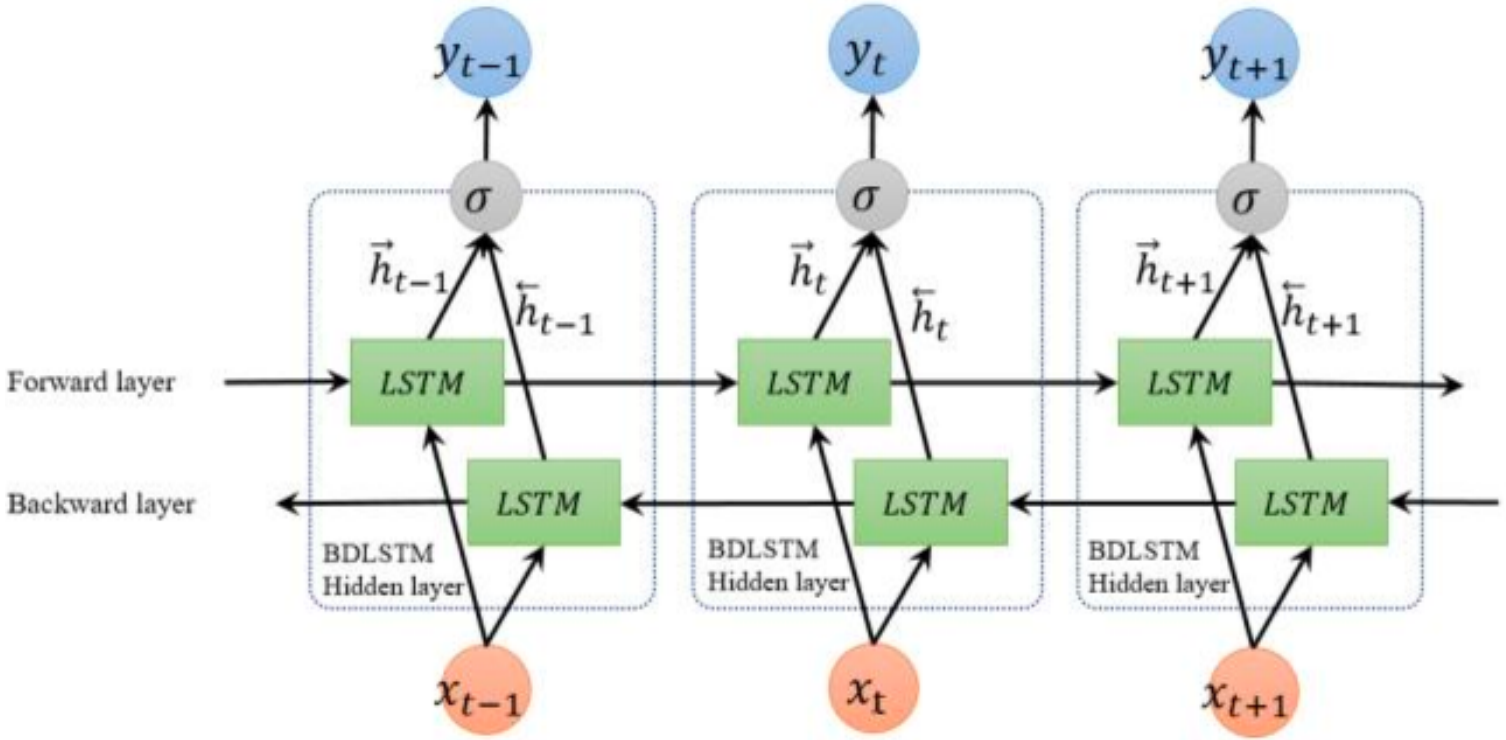


Stacked LSTM

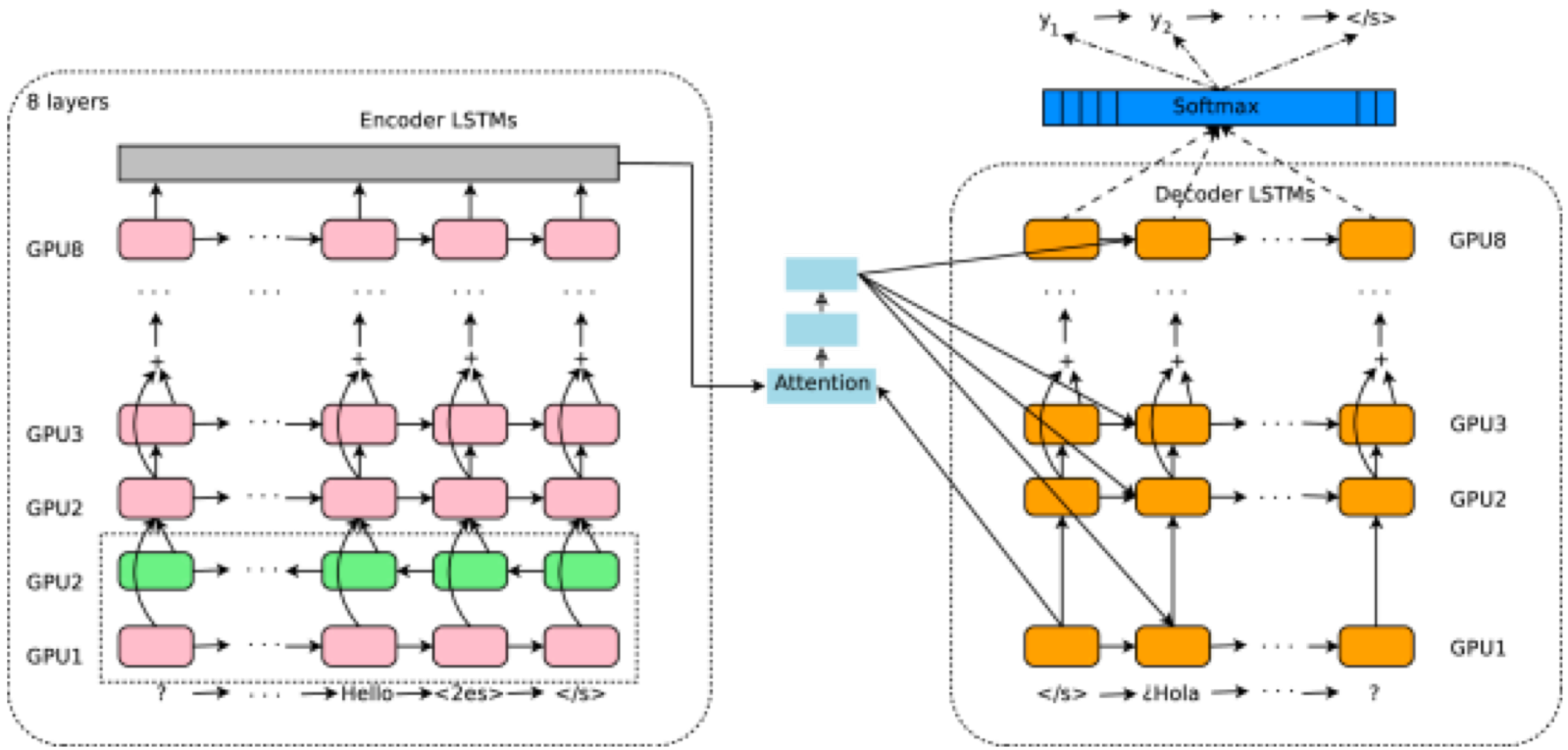




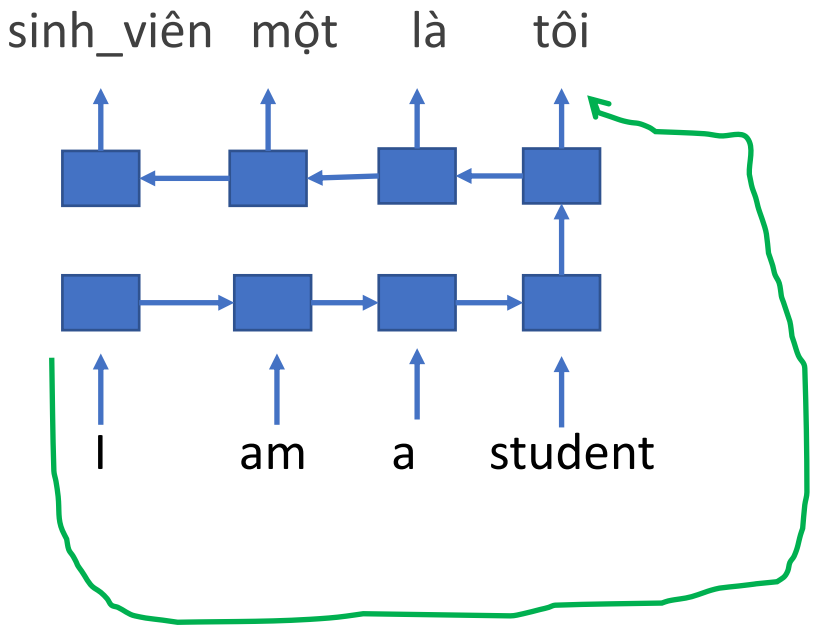
BiLSTM

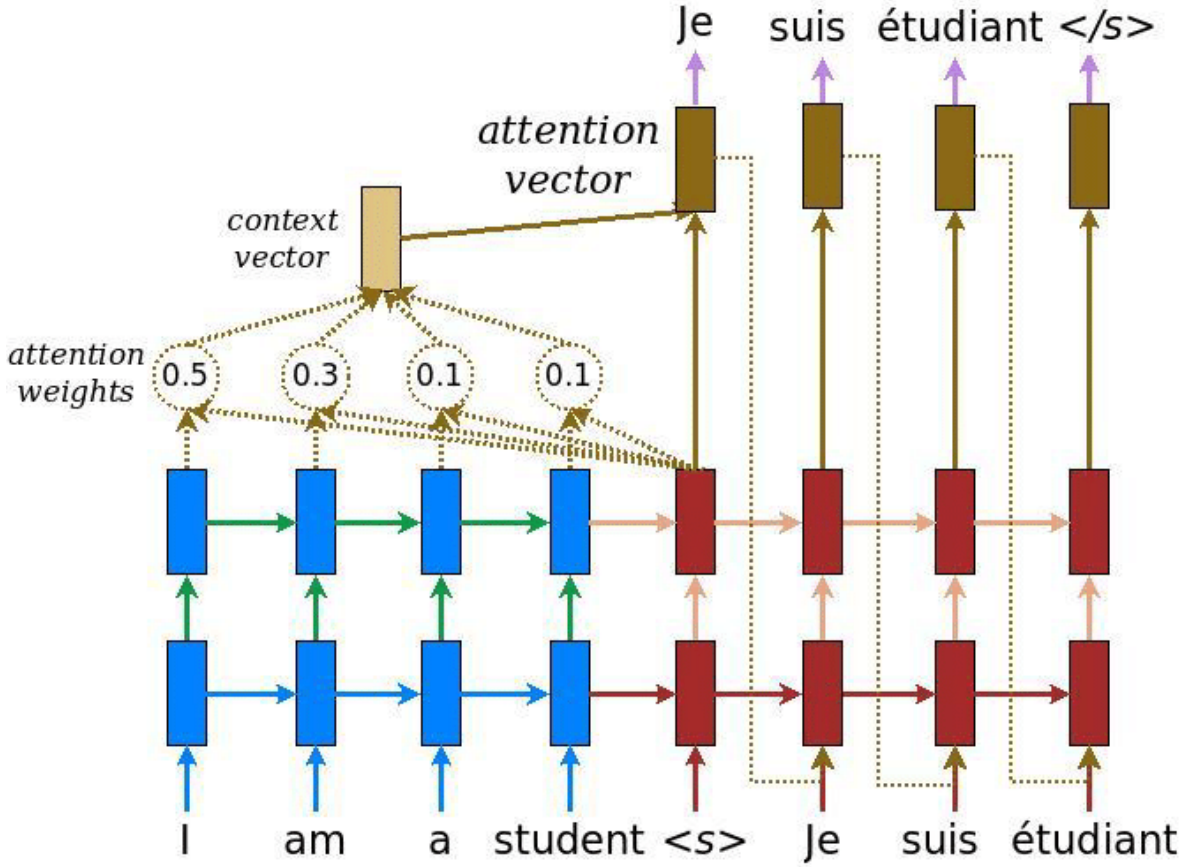

y_{t-1}
y_t
y_{t+1}
σ
$\vec{h}_{t-1}$
$\overleftarrow{h}_{t-1}$
$\vec{h}_t$
$\overleftarrow{h}_t$
$\vec{h}_{t+1}$
$\overleftarrow{h}_{t+1}$
Forward layer
Backward layer
LSTM
BDLSTM Hidden layer
x_{t-1}
x_t
x_{t+1}

Neural Machine Translation



Attention based NMT



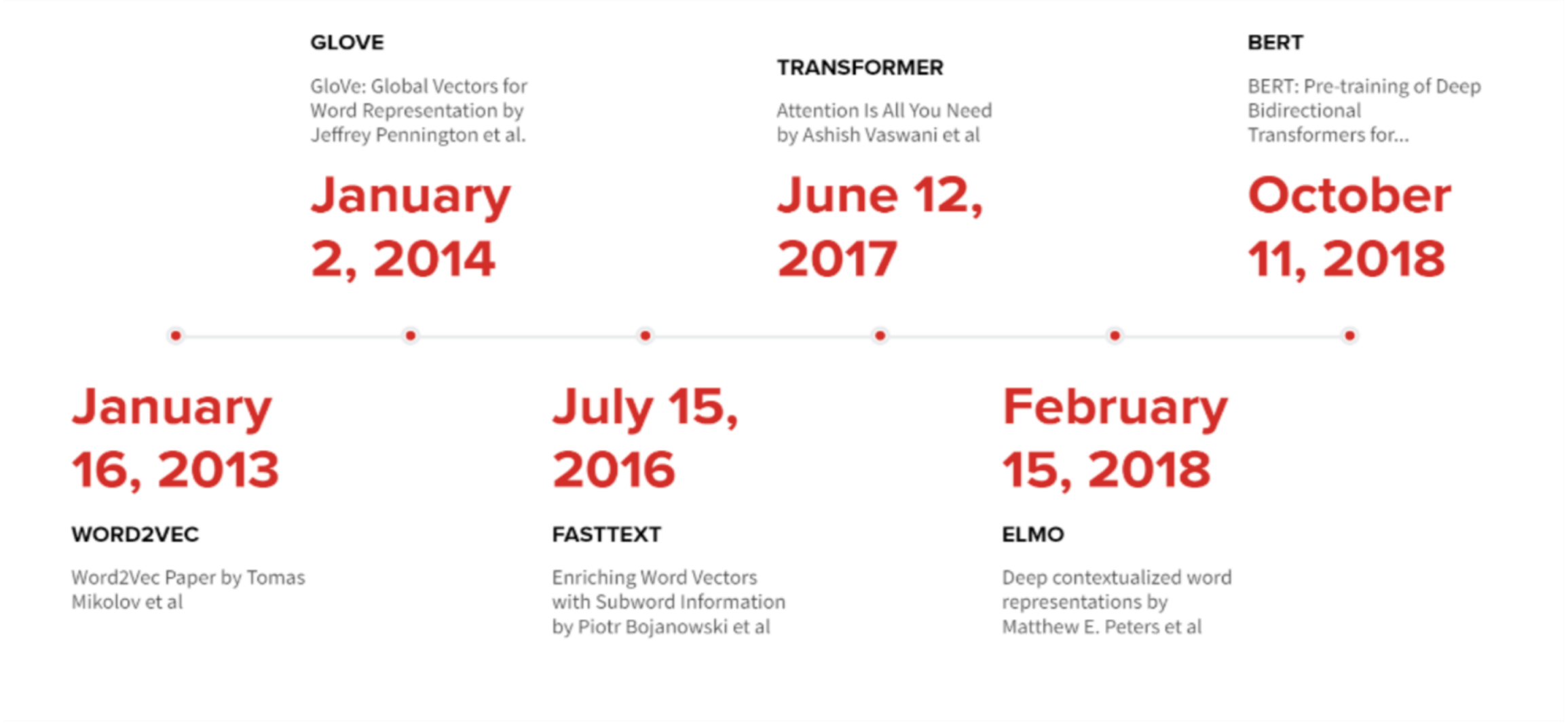


Outline

1. How to recognize Deep Learning networks/models:
   - Traditional statistical machine learning vs Deep Learning
   - Vanilla Neural Networks vs Deep Learning Networks
   - History of DL development
2. Convolutional Neural Network (CNN)
3. Reccurent Neural Network (RNN) and Long Short Term Memory (LSTM)
4. Transformer Models

Word Representation

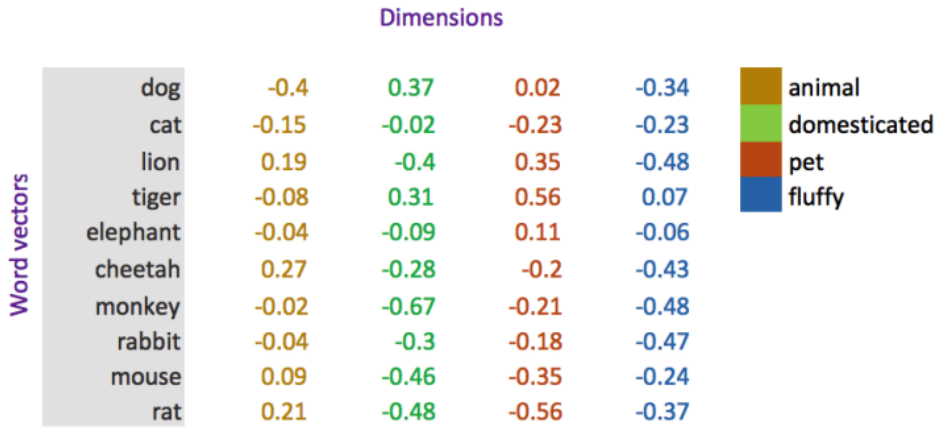
1. Word Embeddings/Word2vec
2. Word pre-train models
3. Transformer based models

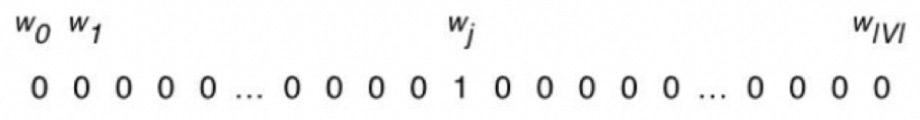


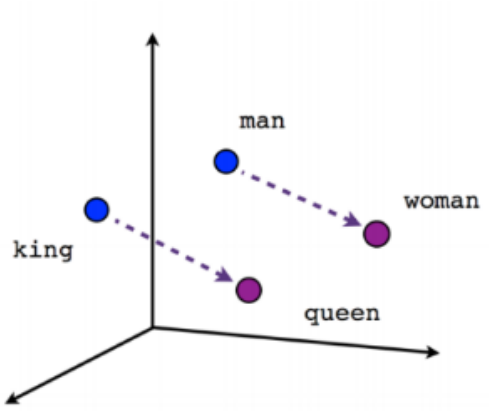
Timeline of some major NLP Projects

Word Embeddings (Semantic Distributed Representation)

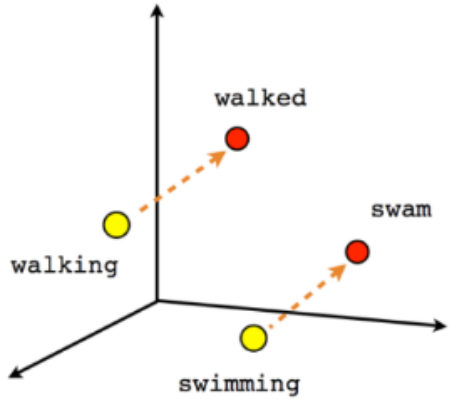
- Embed the words into a space of abstract concepts
- The words are placed in a space of n<< |V| dimensions.







Male-Female



Verb tense

Learning Word2Vec

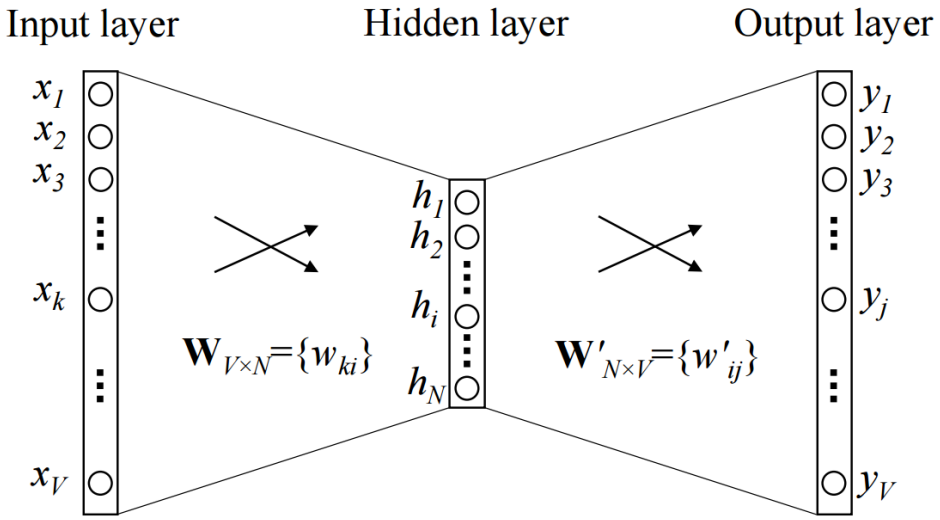
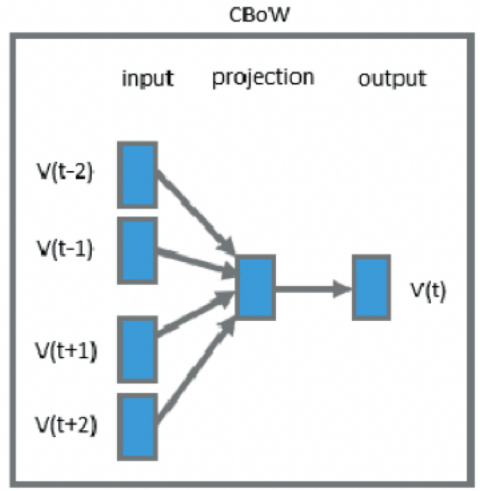
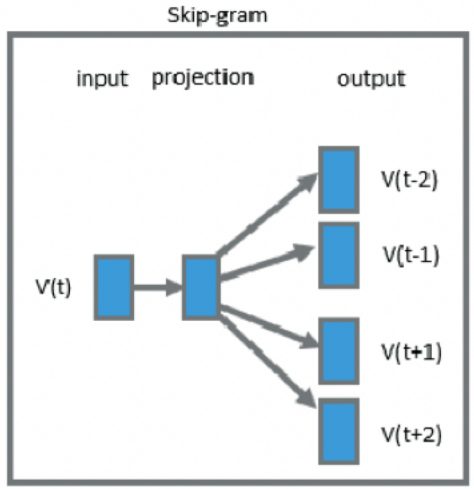


Figure 1: A simple CBOW model with only one word in the context





Pretrain word representations data and tools

- Word2vec: CBOW, Skip-gram
- Glove (word embeddings)
- FastText (FastText is an open-source library developed by the Facebook AI Research

Some Experiments

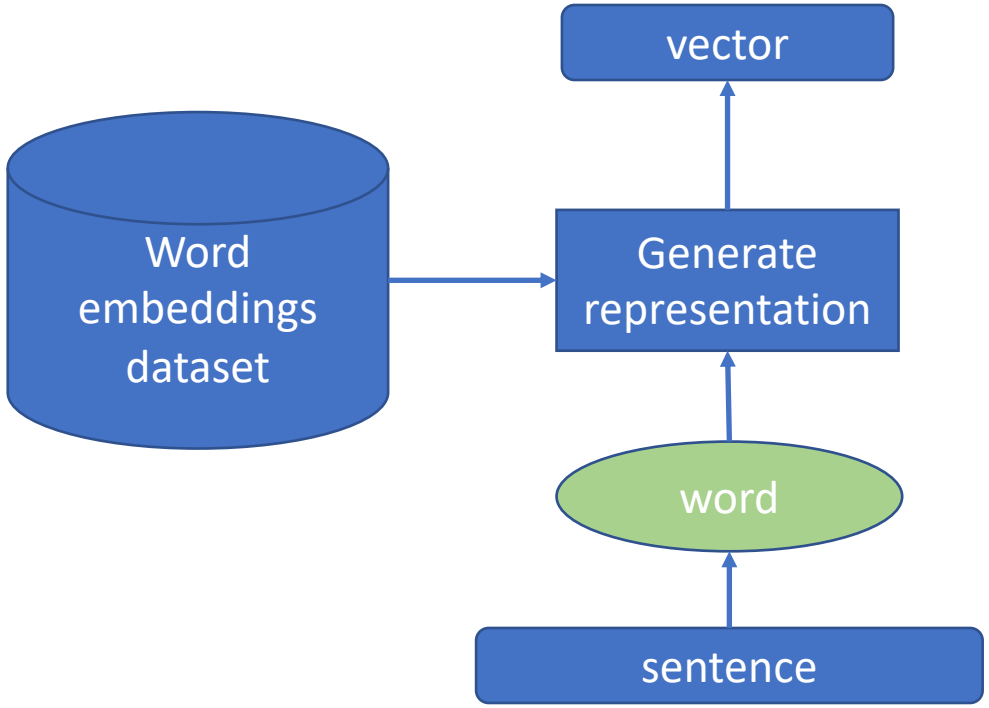
Model	Size	WS353	MC	RG	SCWS	RW
SVD	6B	35.3	35.1	42.5	38.3	25.6
SVD-S	6B	56.5	71.5	71.0	53.6	34.7
SVD-L	6B	65.7	72.7	75.1	56.5	37.0
CBOW†	6B	57.2	65.6	68.2	57.0	32.5
SG†	6B	62.8	65.2	69.7	58.1	37.2
GloVe	6B	65.8	72.7	77.8	53.9	38.1
SVD-L	42B	74.0	76.4	74.1	58.3	39.9
GloVe	42B	**75.9**	**83.6**	**82.9**	**59.6**	**47.8**
CBOW*	100B	68.4	79.6	75.4	59.4	45.5

	Corpora	MRPC	MR	CR	SUBJ	MPQA	SST	TREC	Average
GloVe	Wiki+news	71.9/81.0	75.7	78.1	91.5	86.9	78.1	66.6	79.7
GloVe	Crawl	72.0/80.7	78.0	79.6	91.8	88.0	80.0	84.2	82.0
fastText	Wiki+news	72.9/**81.6**	77.8	80.3	92.2	**88.3**	81.1	**85.0**	82.5
fastText	Crawl	**73.4/81.6**	**78.2**	**81.1**	**92.5**	87.8	**82.0**	84.0	**82.7**

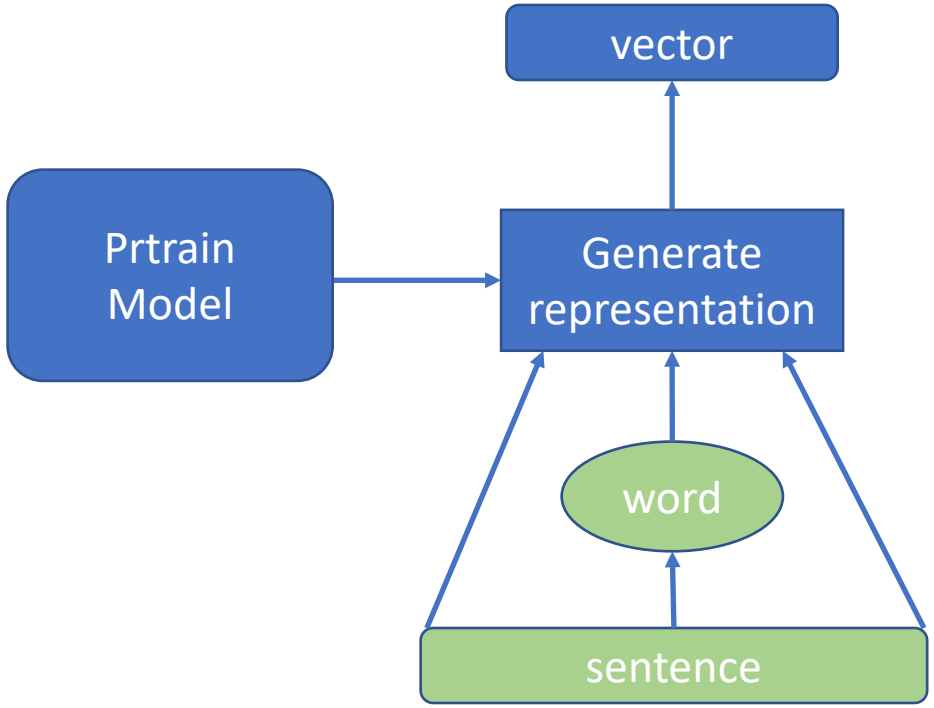
Table 4: Comparison of different pre-trained models on supervised text classification tasks.

Stage 2: Pre-train Models

- Pretrain word presentation: word2vec, word embeddings



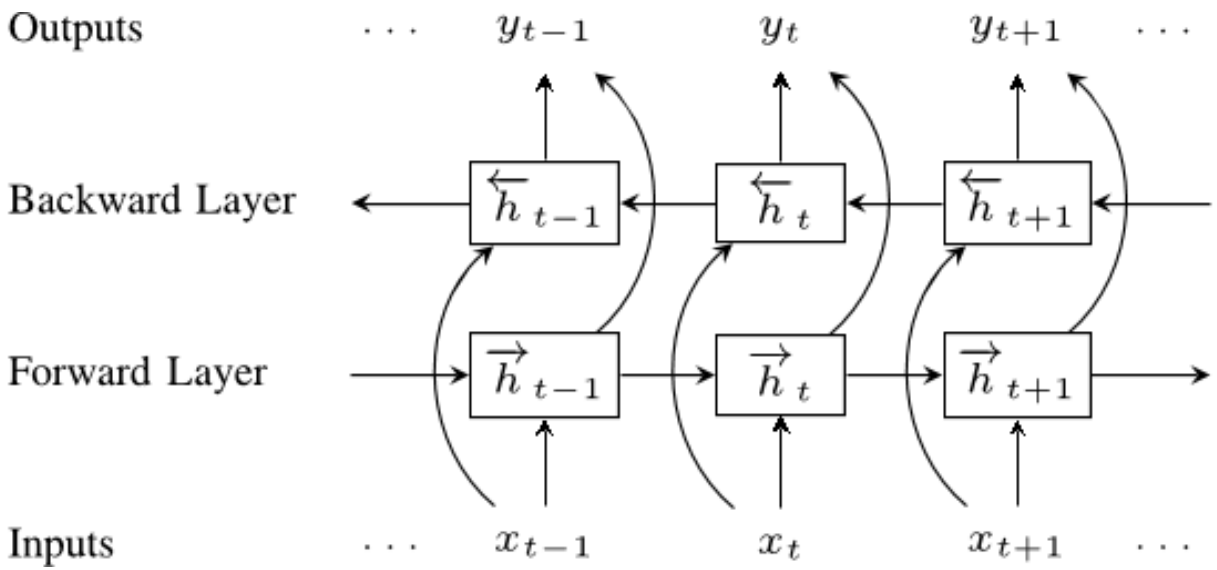
- Pretrain models: generate context-word2ve (dynamic word embeddings)



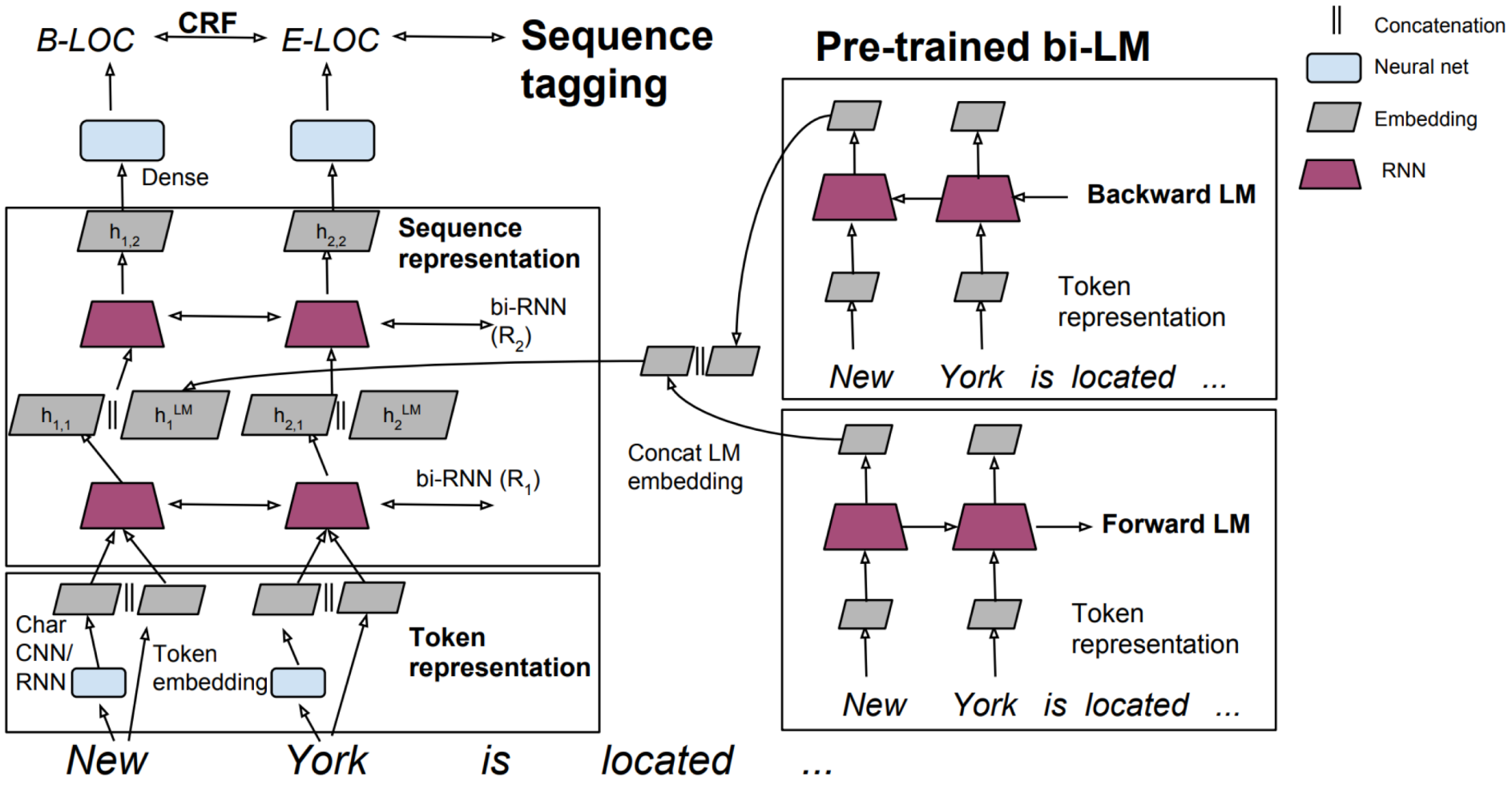
ELMo (Embeddings from Language Models) - 2018

ELMo: Deep contextualized word representations (2018)

- The main idea of the Embeddings from Language Models (ELMo) can be divided into two main tasks, first we train an LSTM-based language model on some corpus, and then we use the hidden states of the LSTM for each token to generate a vector representation of each word.
- It's been shown to outperform GloVe and Word2Vec embeddings on a wide variety of NLP tasks.



Using ELMo


B-LOC
CRF
E-LOC
Sequence tagging
Dense
Sequence representation
bi-RNN (R2)
bi-RNN (R1)
Concat LM embedding
Char CNN/ RNN
Token embedding
Token representation
New
York
is
located
...
Pre-trained bi-LM
Backward LM
Token representation
Forward LM
Concatenation
Neural net
Embedding
RNN

Using ELMo: Experiments

TASK	PREVIOUS SOTA		OUR BASELINE	ELMO + BASELINE	INCREASE (ABSOLUTE/ RELATIVE)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Table 1: Test set comparison of ELMo enhanced neural models with state-of-the-art single model baselines across six benchmark NLP tasks. The performance metric varies across tasks – accuracy for SNLI and SST-5; F_1 for SQuAD, SRL and NER; average F_1 for Coref. Due to the small test sizes for NER and SST-5, we report the mean and standard deviation across five runs with different random seeds. The "increase" column lists both the absolute and relative improvements over our baseline.

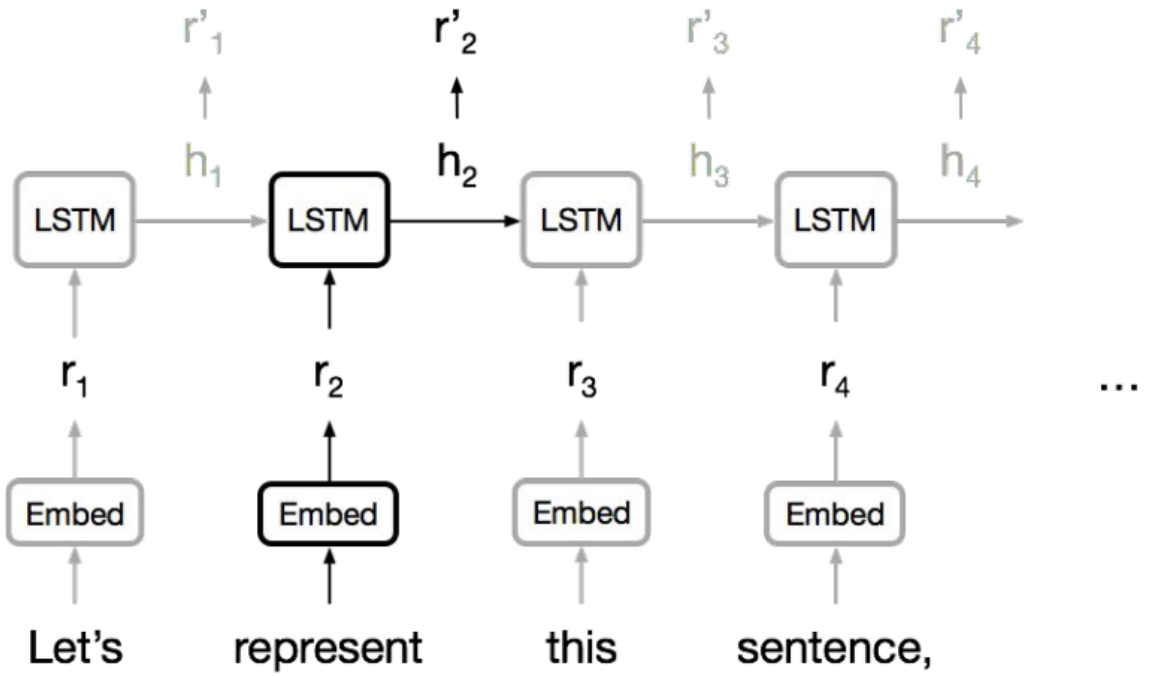
The Transformer
Self & Multi-Head Attention

State-Of-The-Art models in Natura Language Processing

Why? Motivation

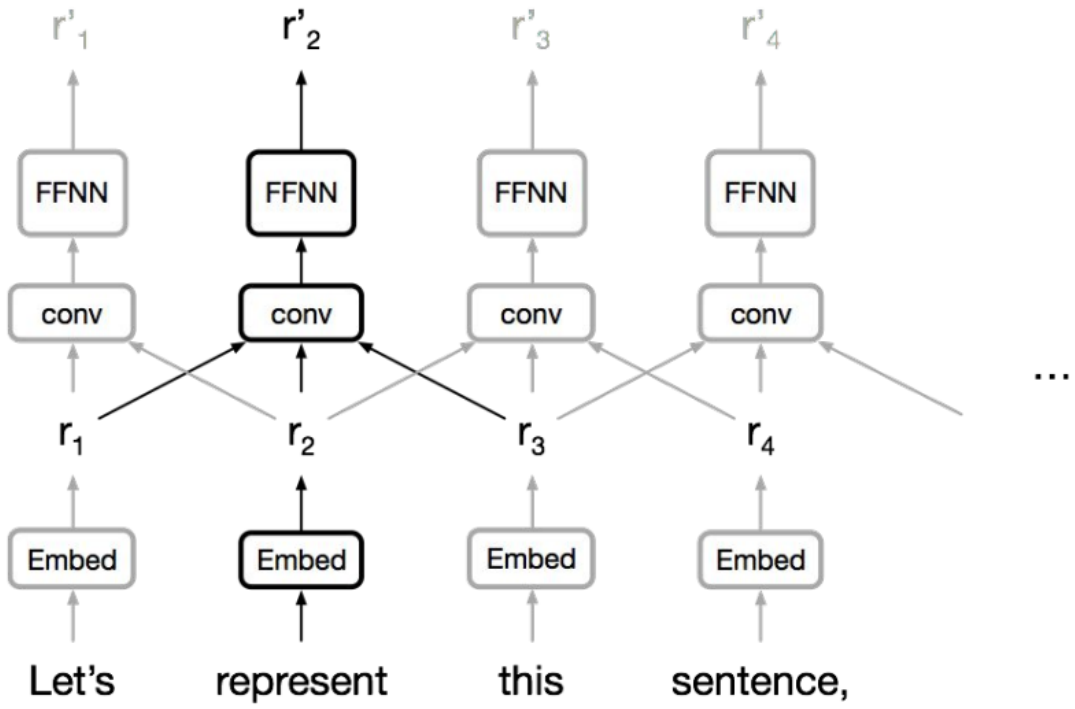
RNN

- Sequential computation inhibits parallelization.
- No explicit modeling of long and short range dependencies.



Why? Motivation

- Convolutional Neural Network
  - Exploits local dependencies
  - Long-distance dependencies require many layers.



Why? Motivation

- Attention
  - Attention between encoder and decoder is crucial in NMT.
  - Why not use attention for representations?

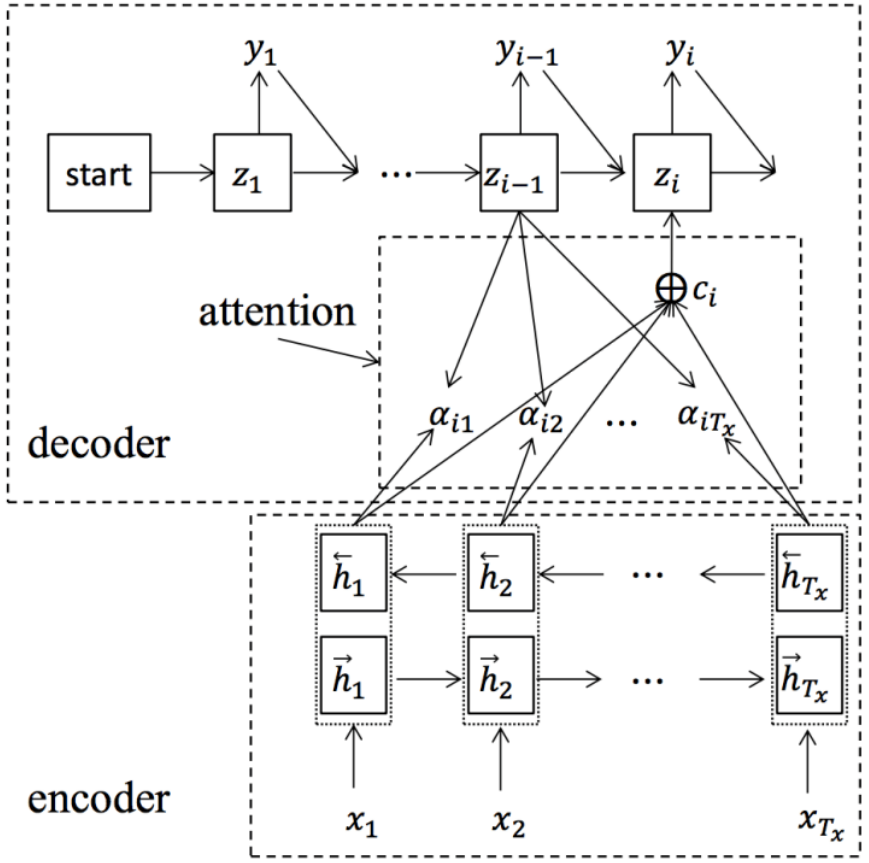
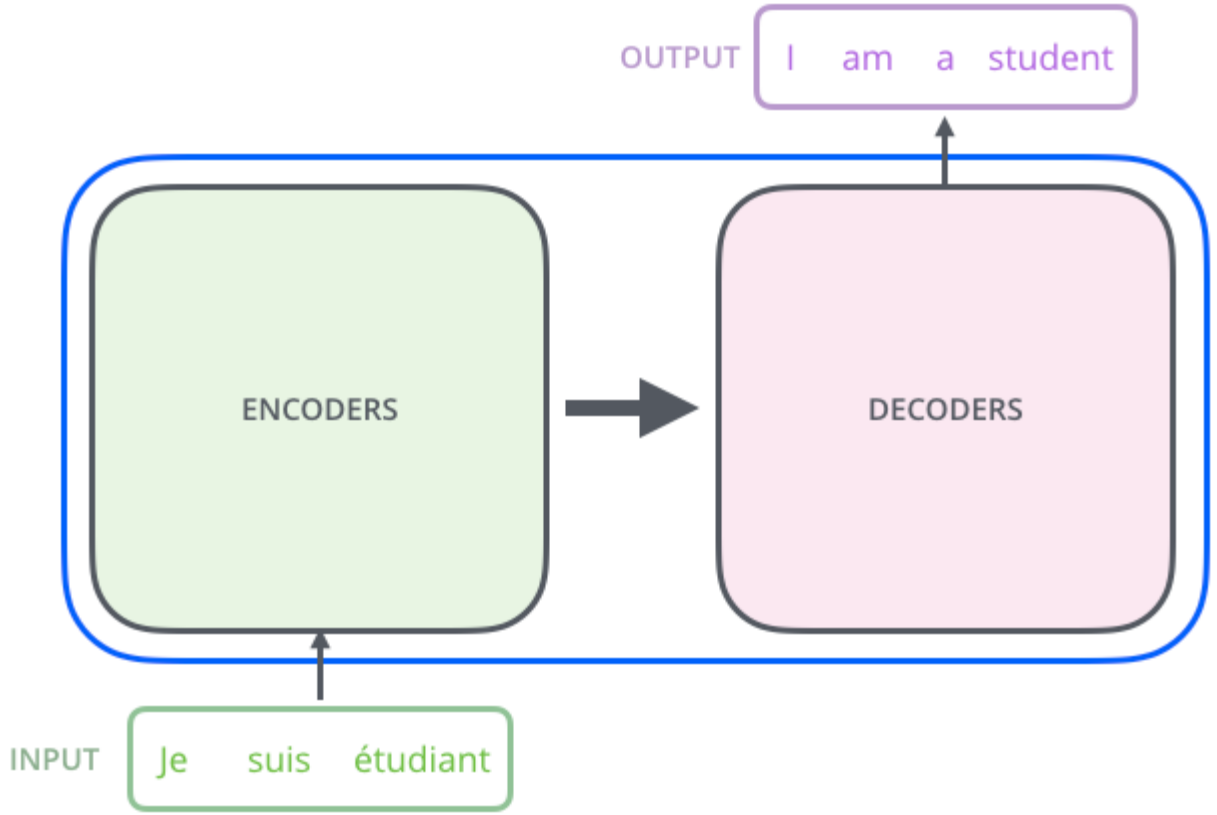


Figure 1: The encoder-decoder NMT with attention.

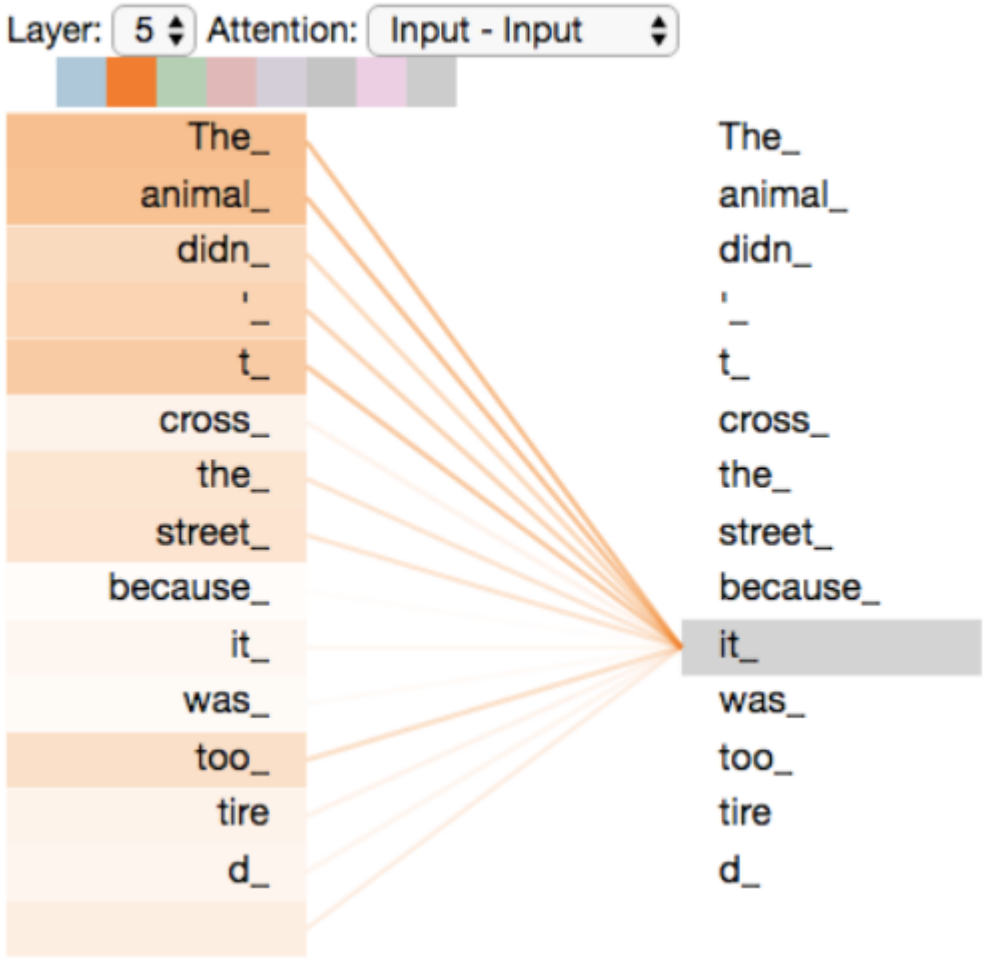
Solution: Transformer Model (Mô hình chuyển đổi)

- Self-Attention
- Multi-Head Attention

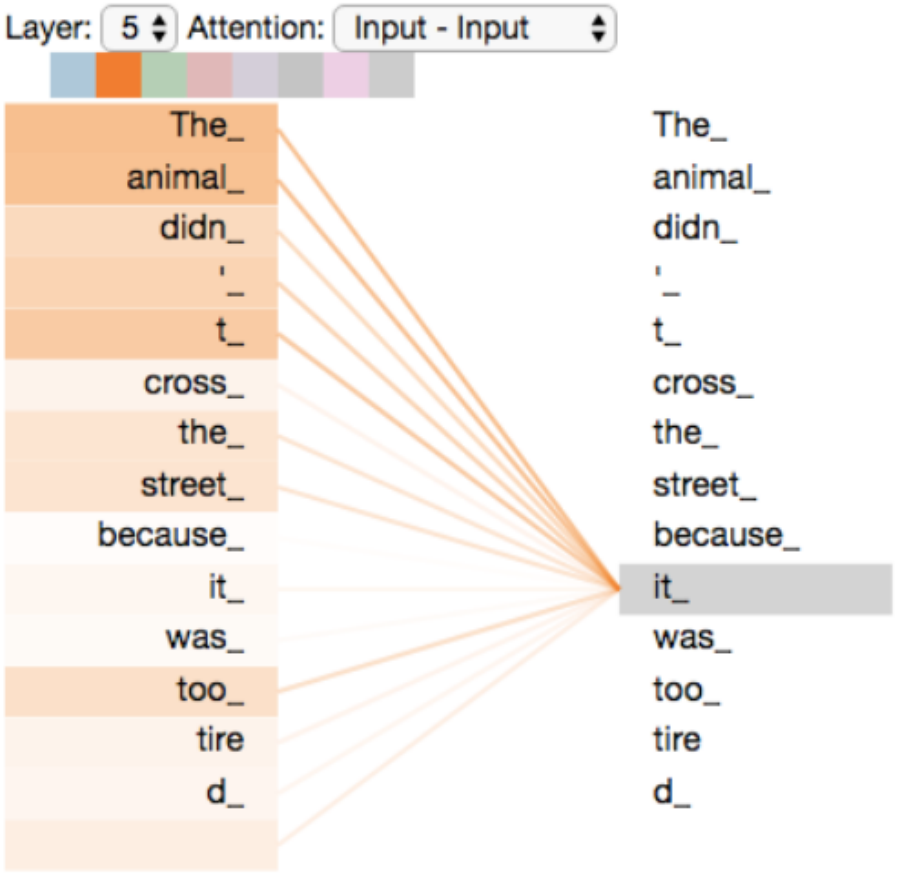
Transformer Model

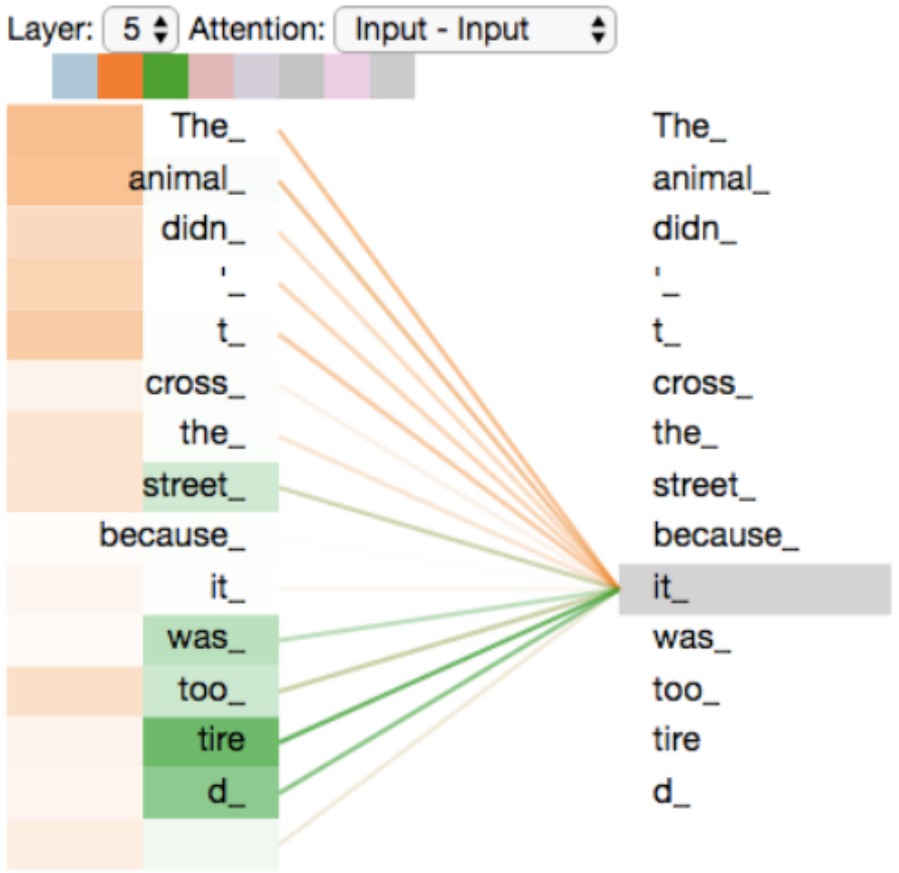


What is Self-Attention?
For encoding

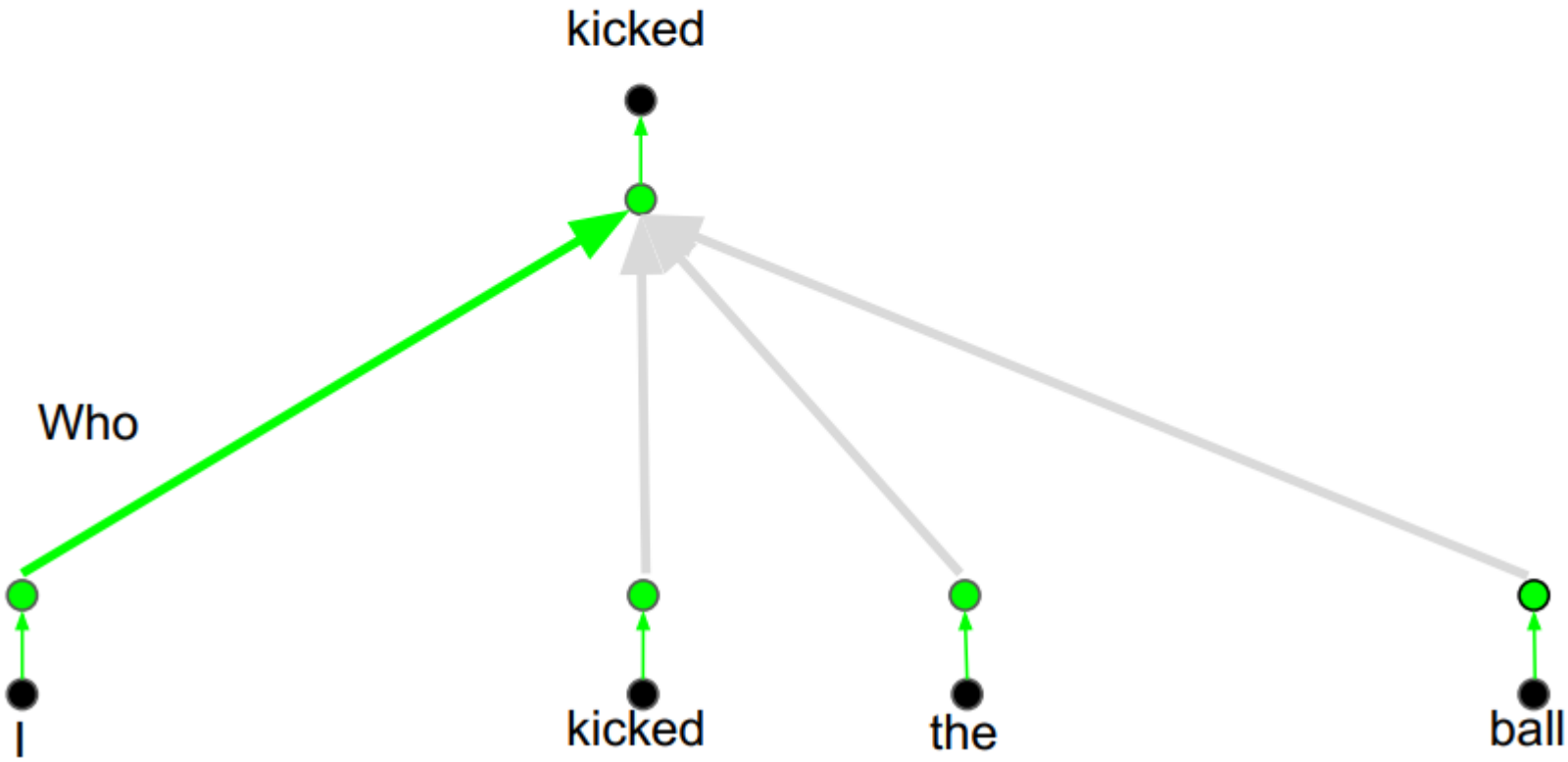


Multi-Head Attention

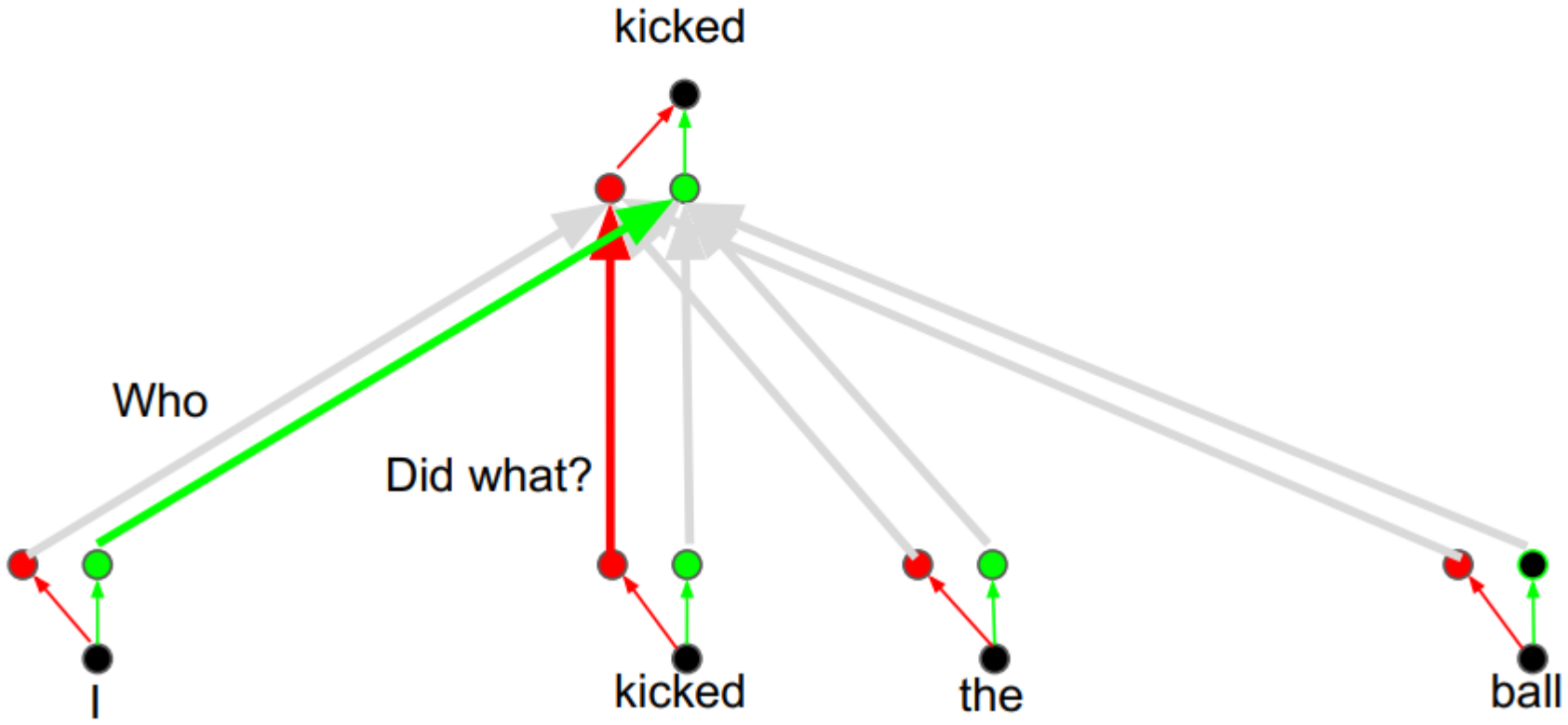




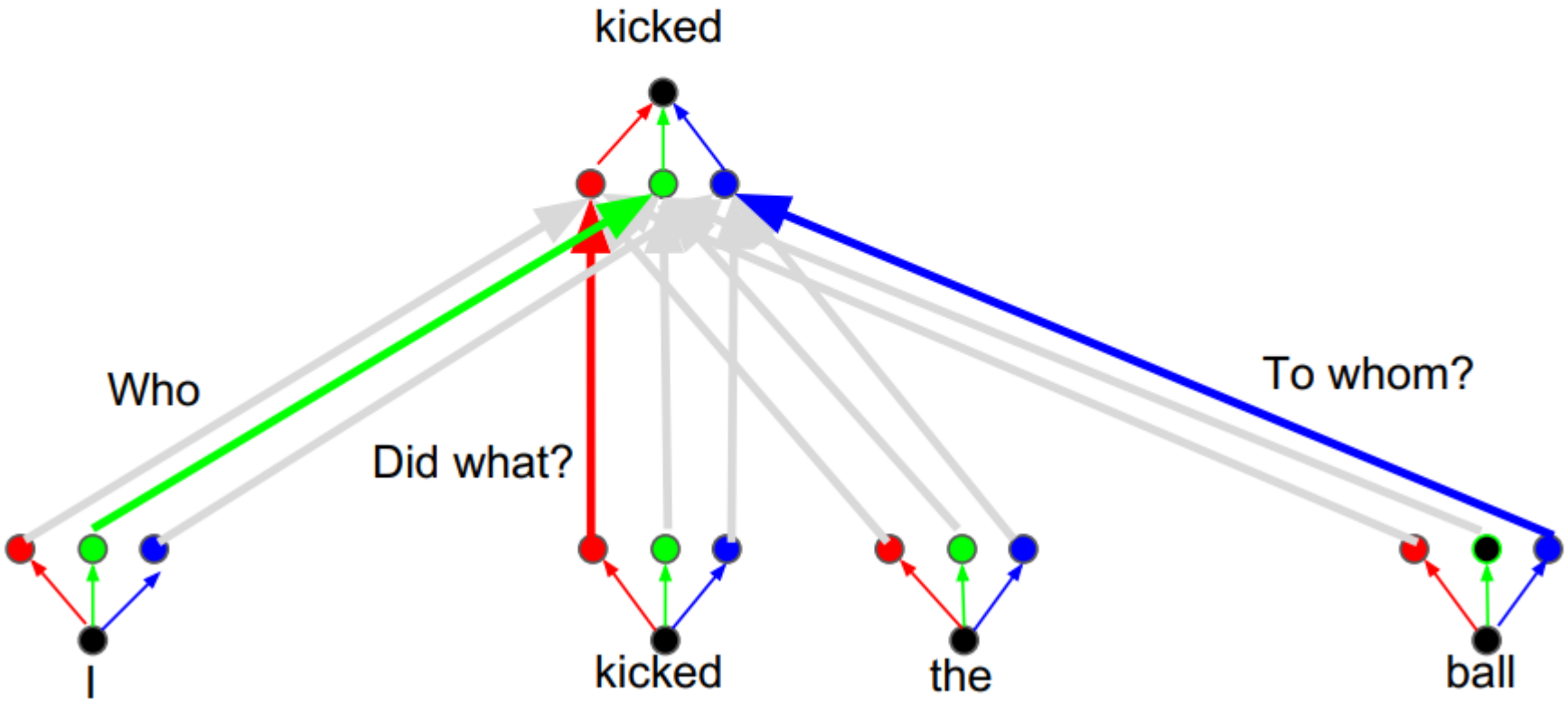
Attention head: Who



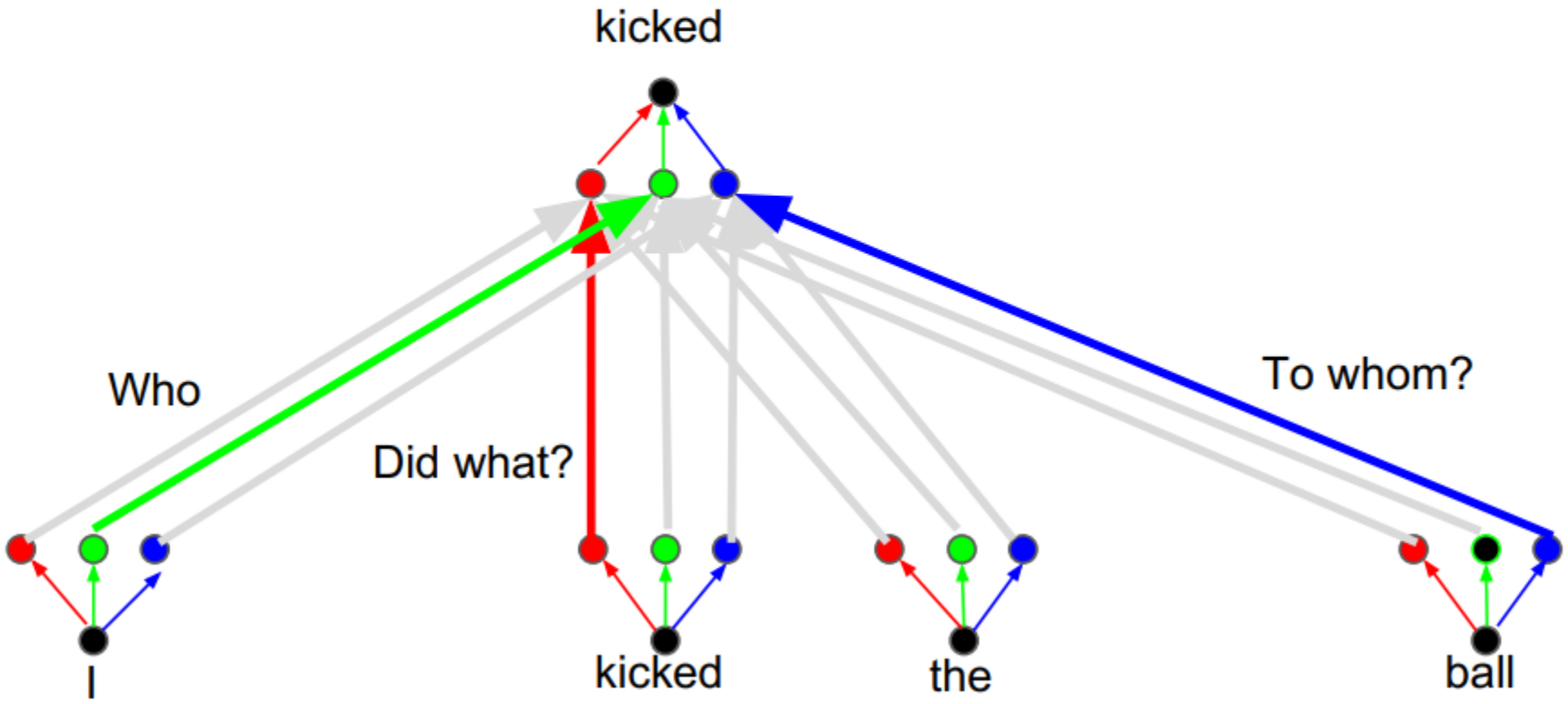
Attention head: Did What?



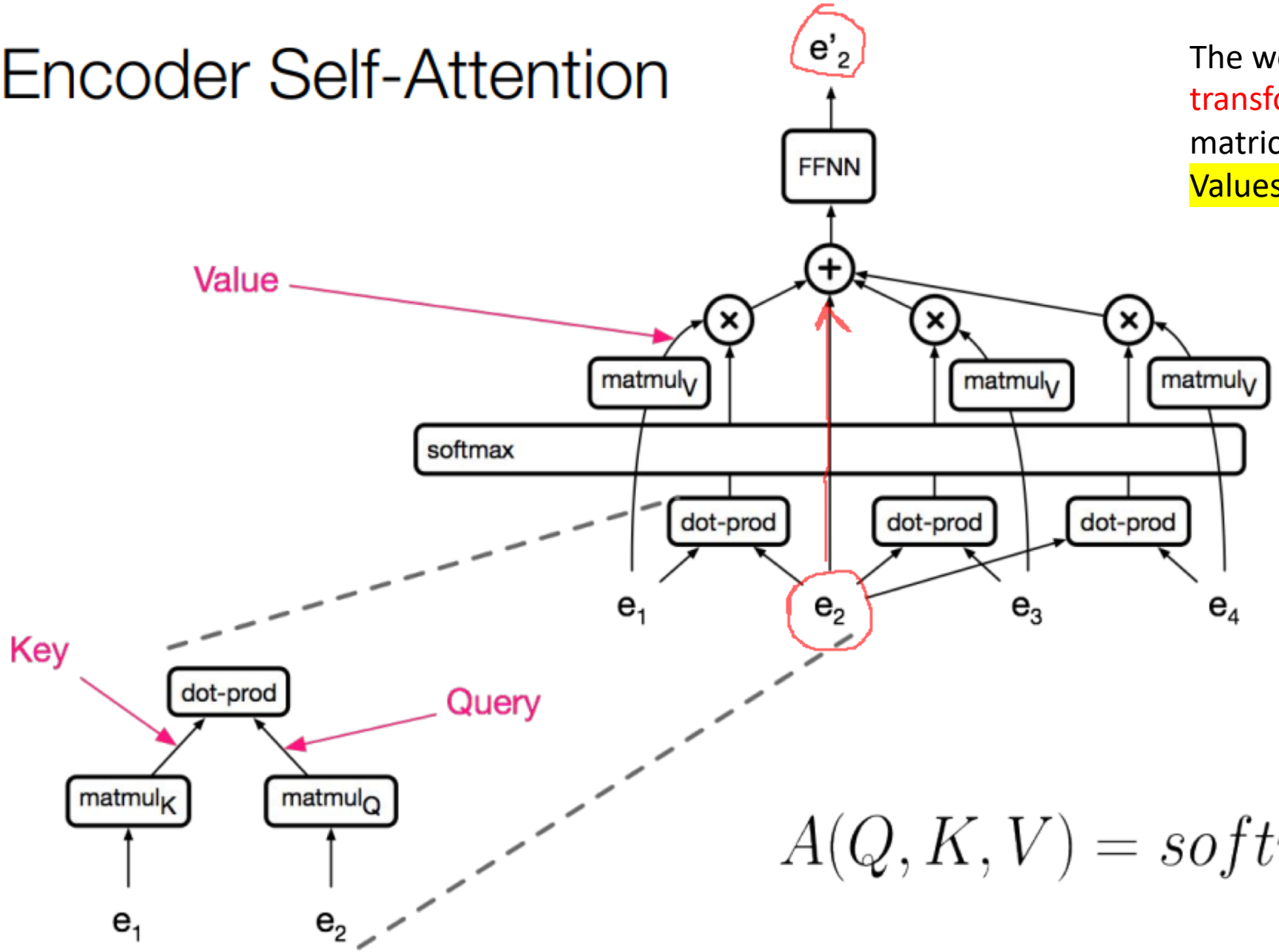
Attention head: To Whom?



Multihead Attention

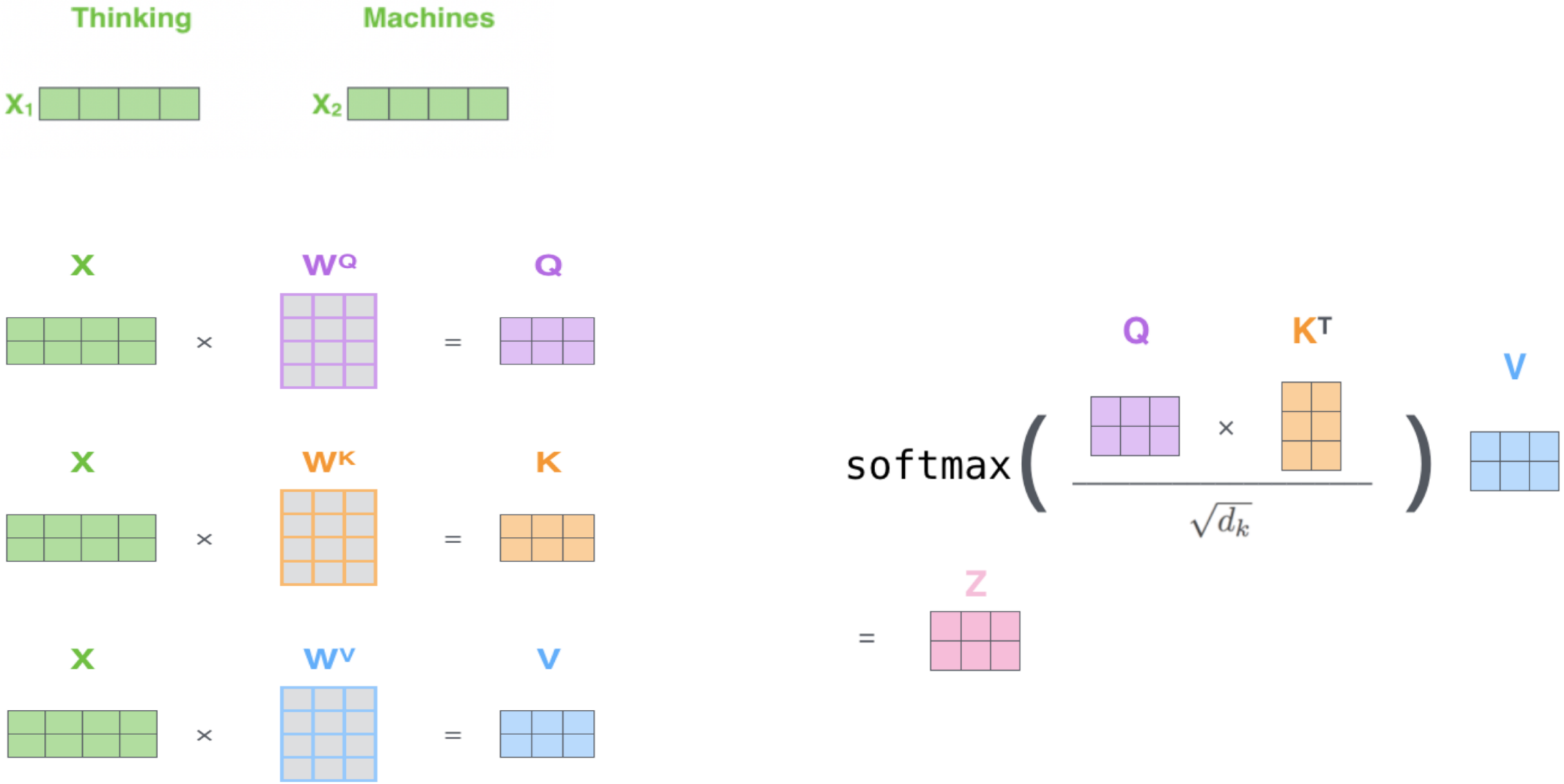


Encoder Self-Attention

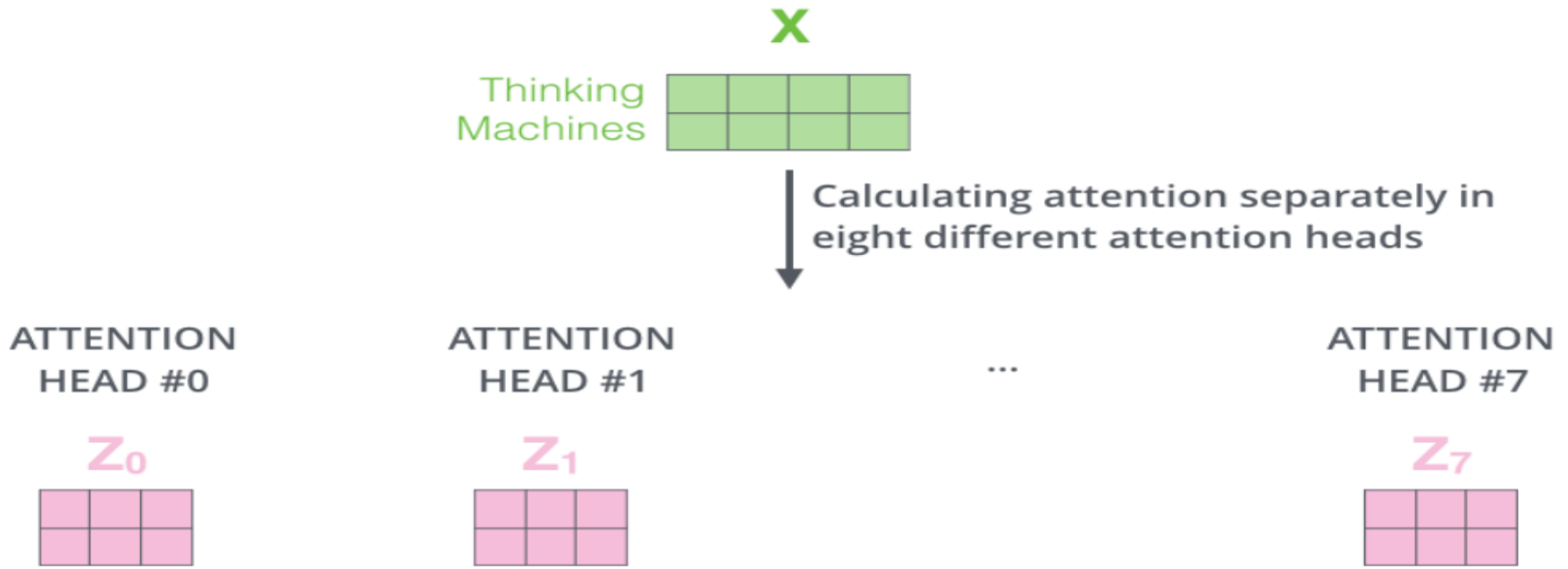


The word embedding is transformed into three separate matrices -- Queries, Keys, and Values.

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


Thinking
Machines
X_1
X_2
X
W^Q
Q
X
W^K
K
X
W^V
V
softmax
Q
K^T
V
$\sqrt{d_k}$
Z

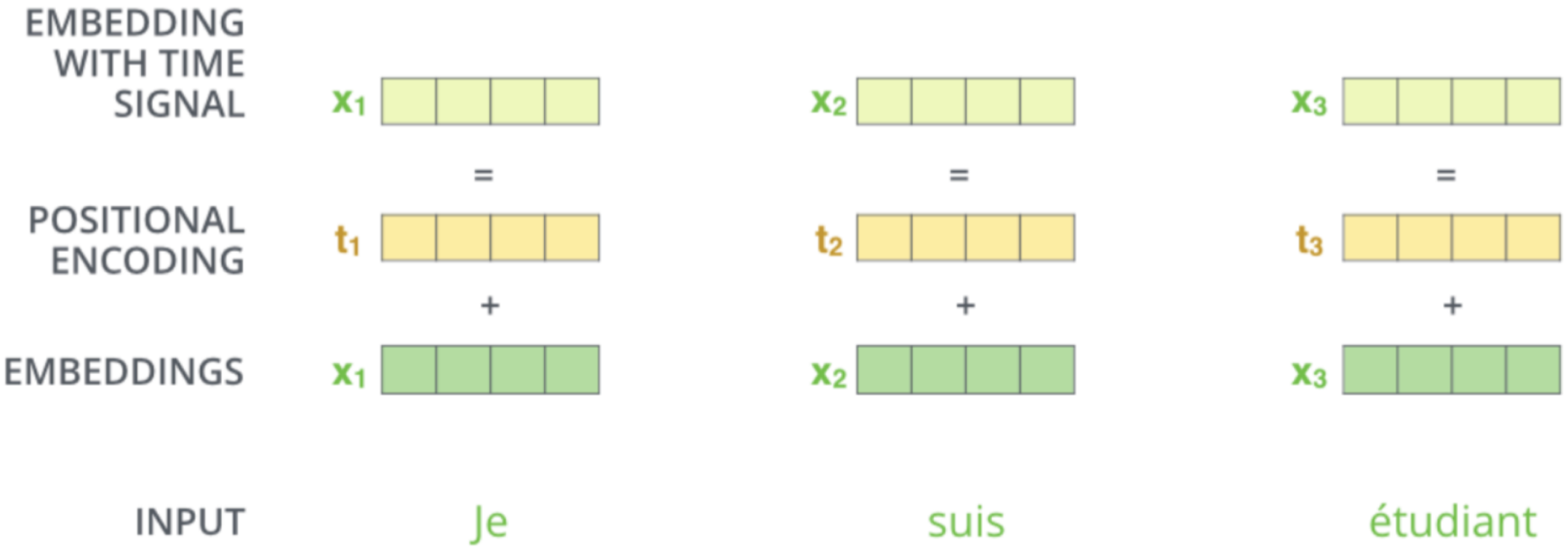
Multi-Head Attention

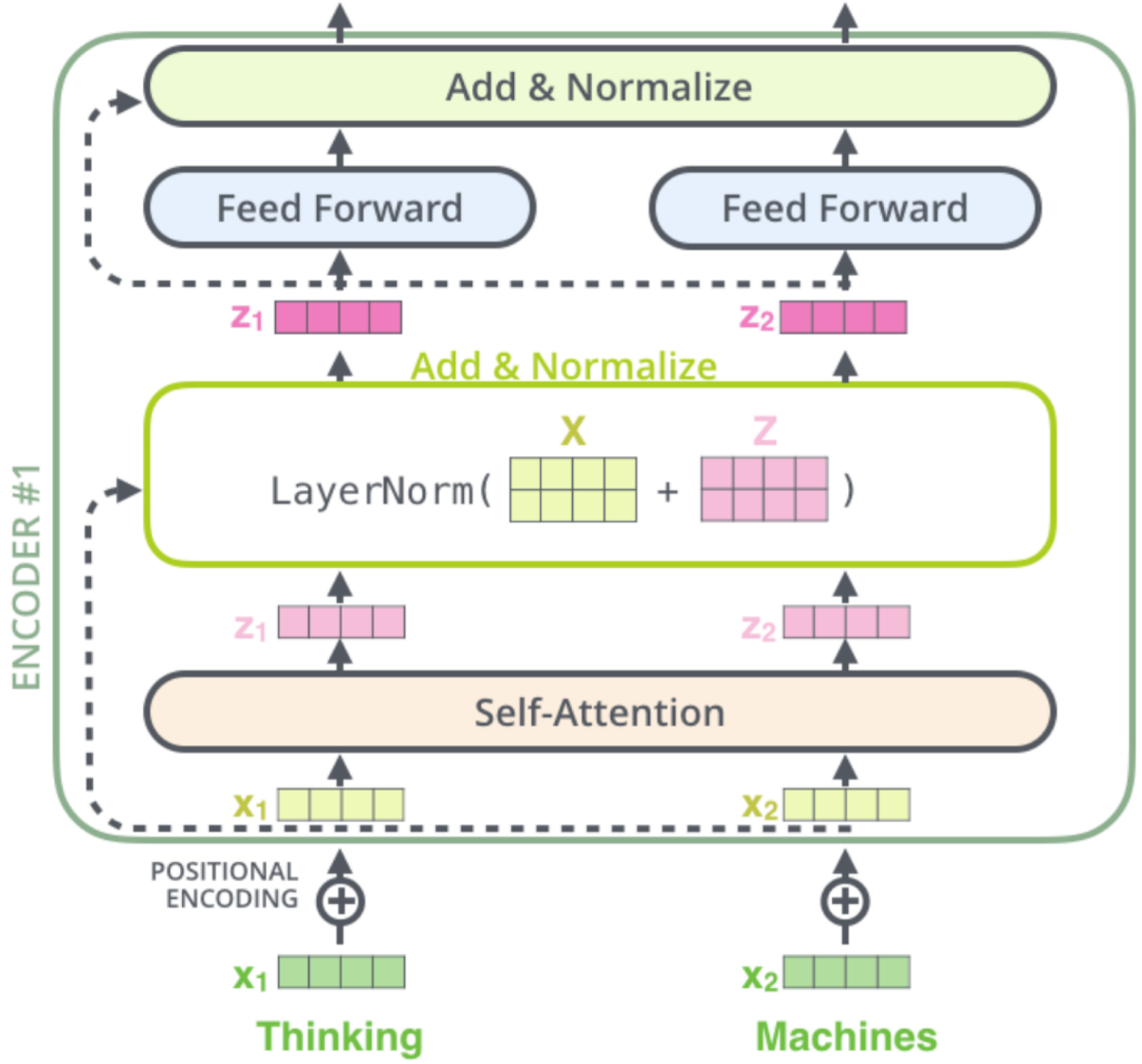


$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

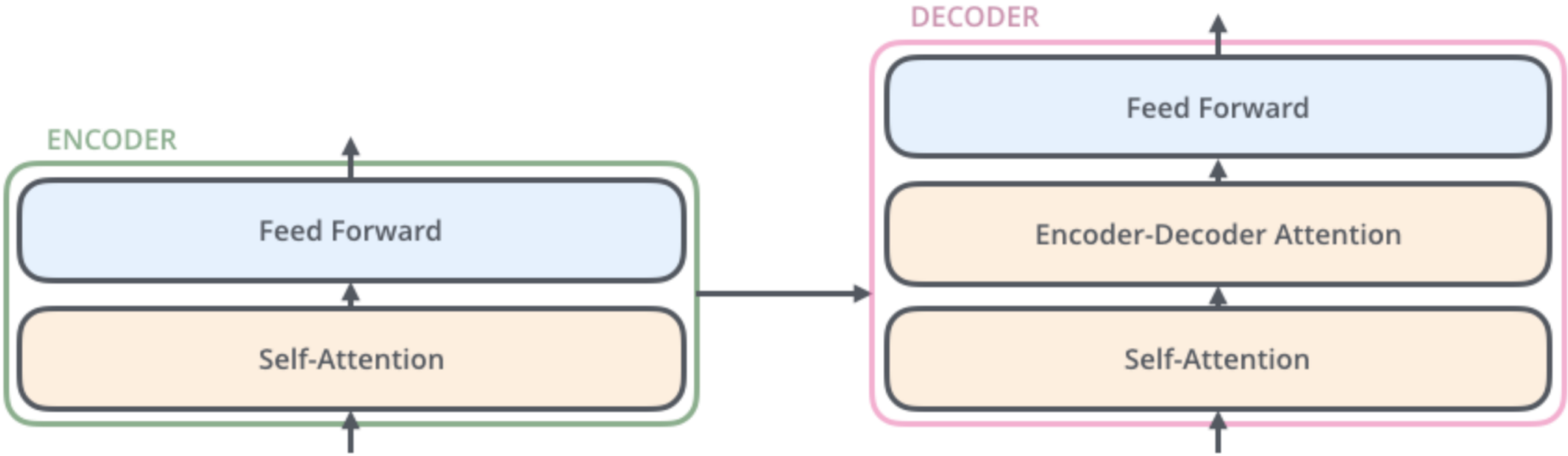
$$\textbf{where } \text{head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

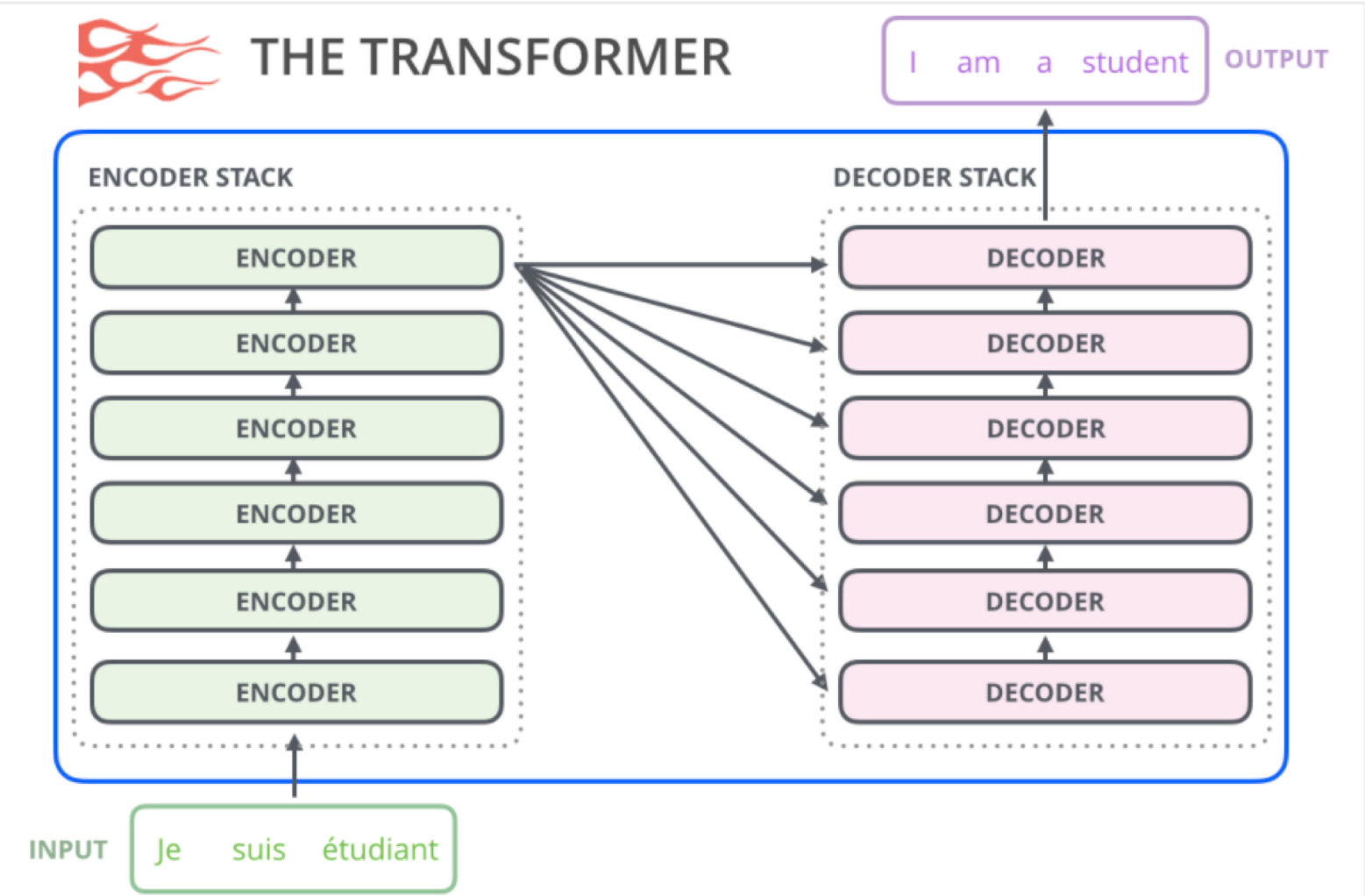
Positional Encoding



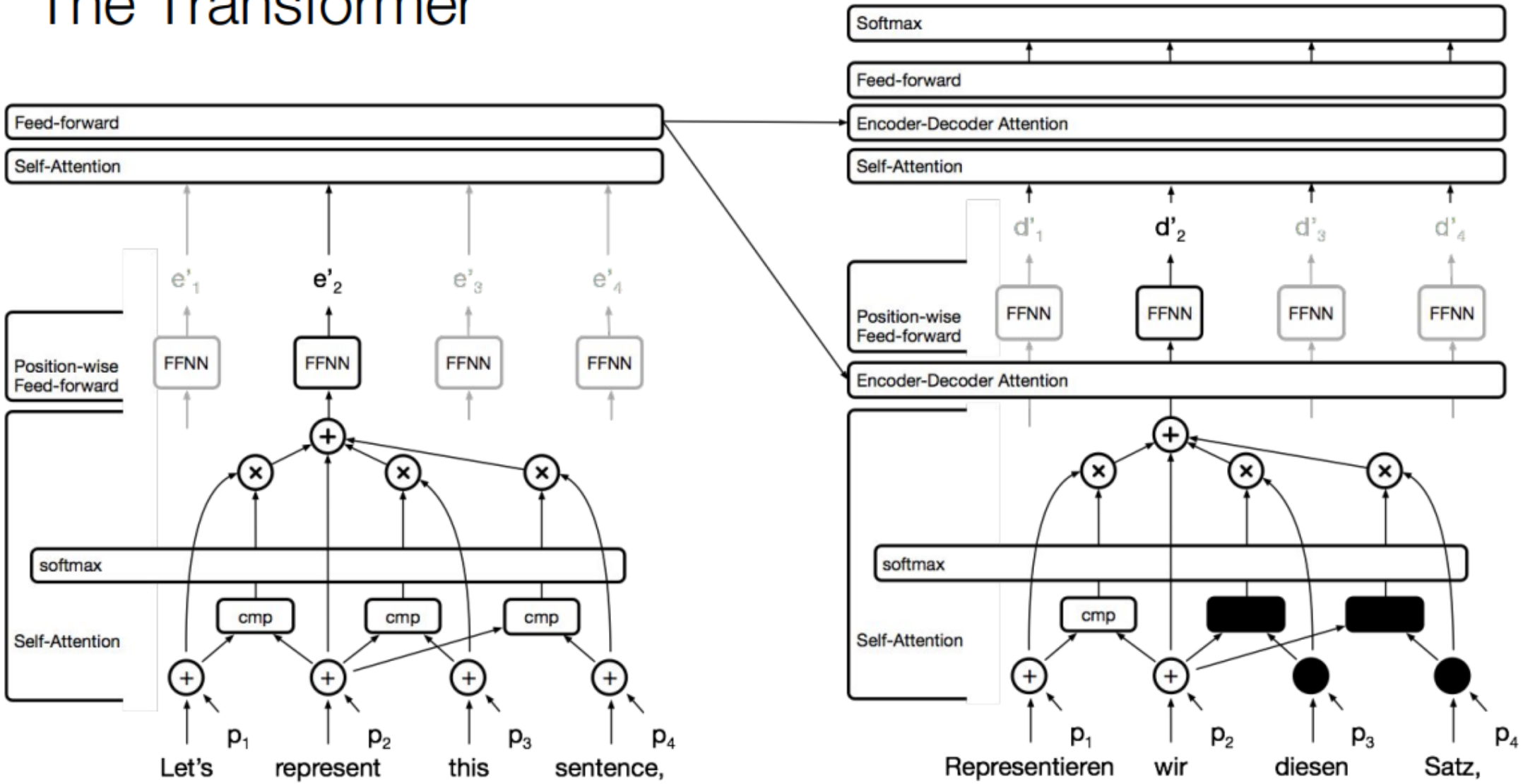

Add & Normalize
Feed Forward
Feed Forward
z1
z2
Add & Normalize
X
Z
LayerNorm(+)
z1
z2
Self-Attention
x1
x2
ENCODER #1
POSITIONAL ENCODING
x1
x2
Thinking
Machines

Encoder – Decoder Architecture

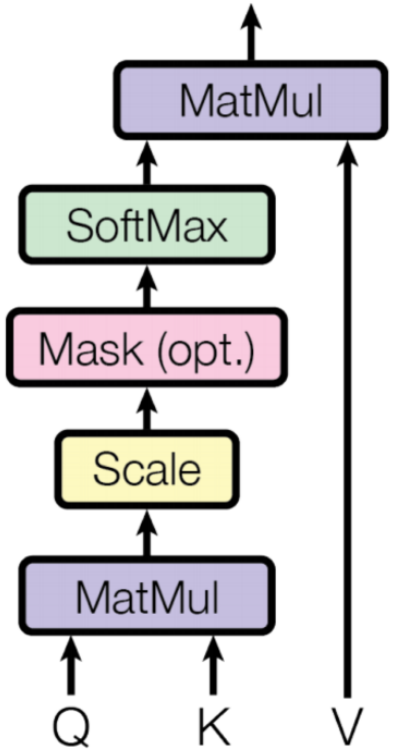



THE TRANSFORMER
I am a student
OUTPUT
ENCODER STACK
DECODER STACK
ENCODER
ENCODER
ENCODER
ENCODER
ENCODER
ENCODER
DECODER
DECODER
DECODER
DECODER
DECODER
DECODER
INPUT
Je suis étudiant

The Transformer



A Transformer Architecture



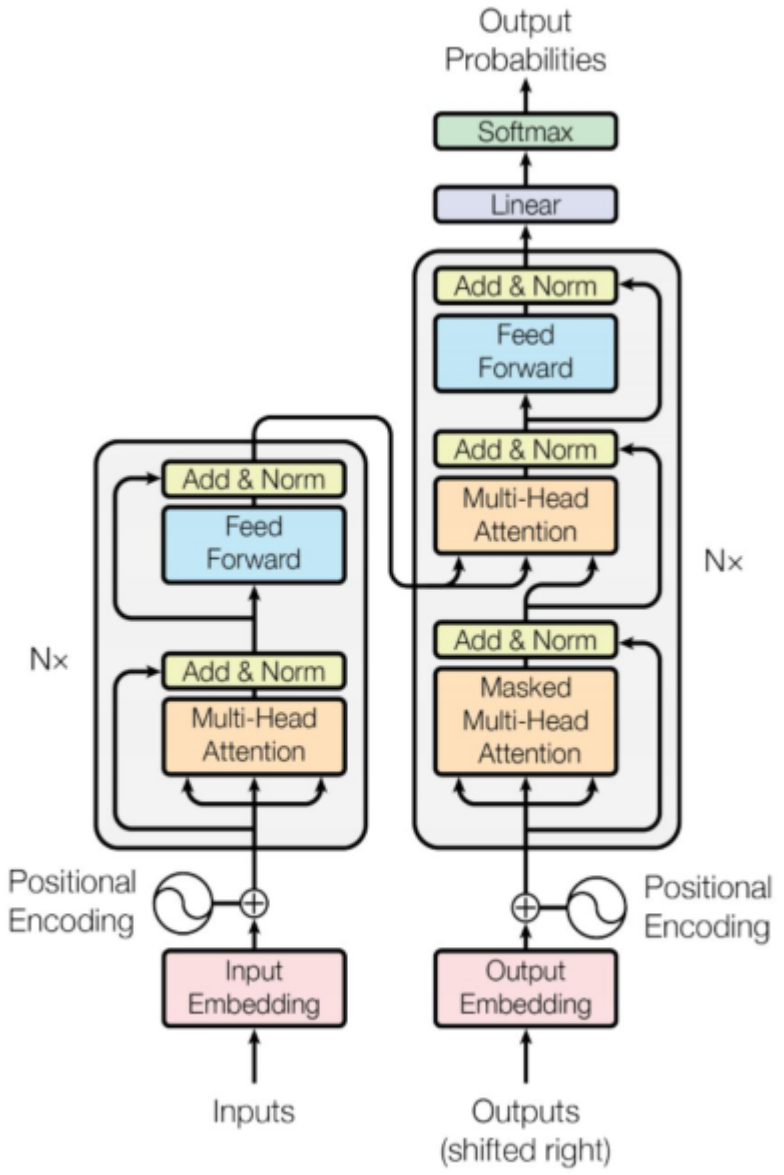


Figure 1: The Transformer - model architecture.

Attention is Cheap!

	FLOPs	
Self-Attention	$O(\text{length}^2 \cdot \text{dim})$	$= 4 \cdot 10^9$
RNN (LSTM)	$O(\text{length} \cdot \text{dim}^2)$	$= 16 \cdot 10^9$
Convolution	$O(\text{length} \cdot \text{dim}^2 \cdot \text{kernel\_width})$	$= 6 \cdot 10^9$

length=1000 dim=1000 kernel_width=3

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

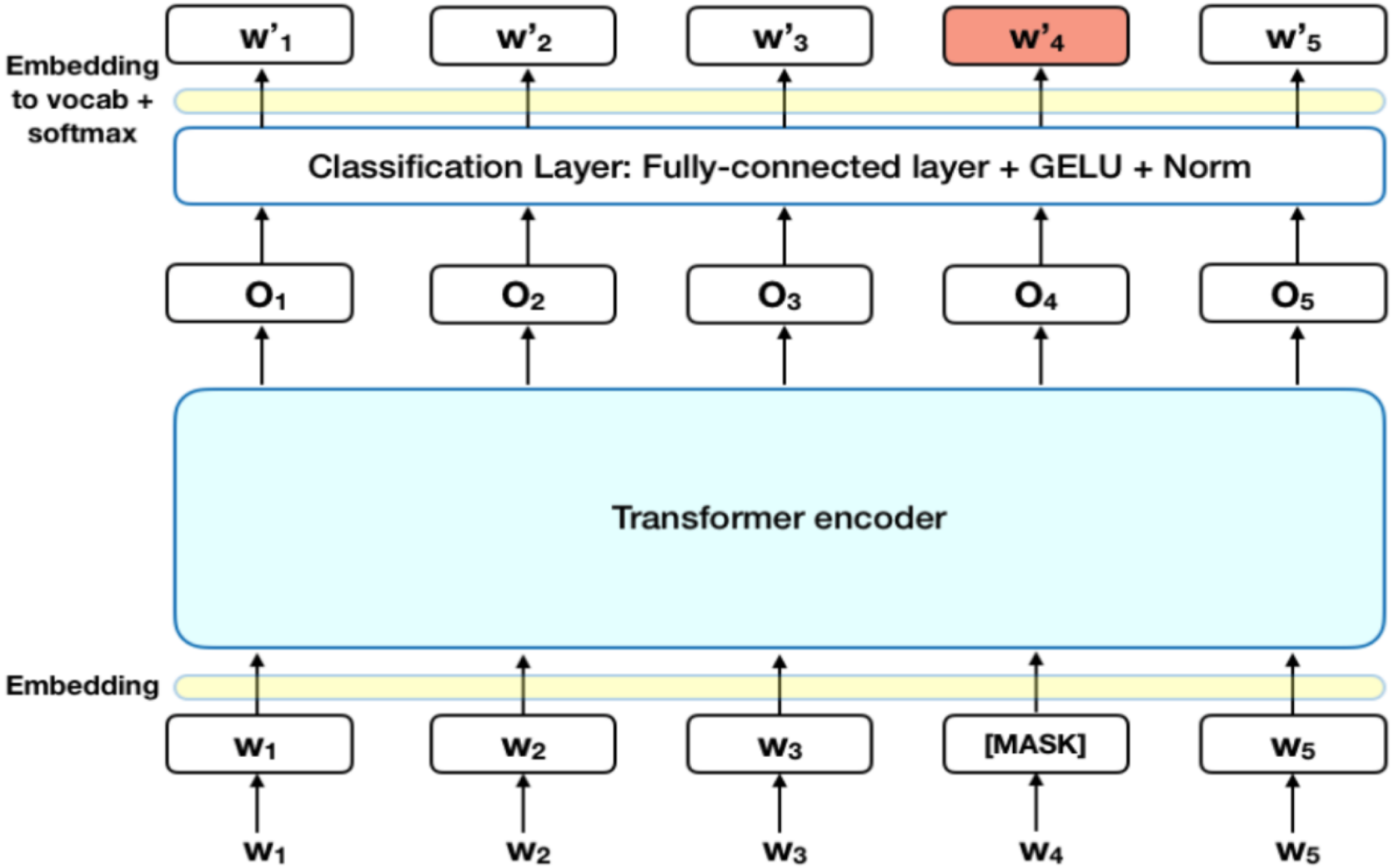
BERT(Bidirectional Encoder Representations from Transformers)

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

[Submitted on 11 Oct 2018 (v1), last revised 24 May 2019 (this version, v2)]

Masked LM (MLM)



Next Sentence Prediction (NSP)

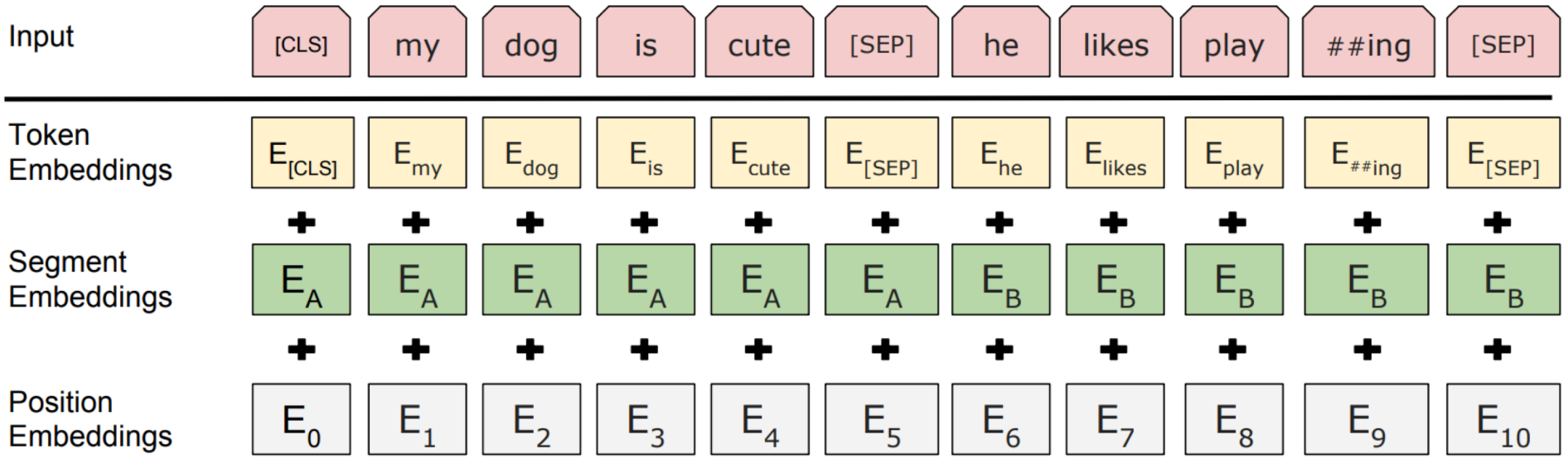


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

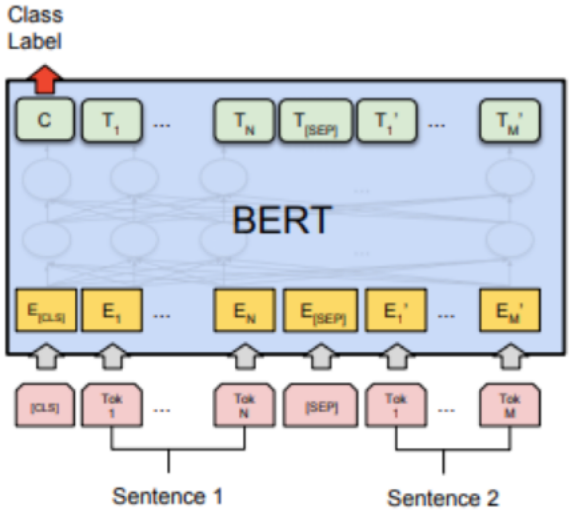
How to use BERT (Fine-tuning)

1. Classification tasks such as sentiment analysis are done similarly to Next Sentence classification, by adding a classification layer on top of the Transformer output for the [CLS] token.

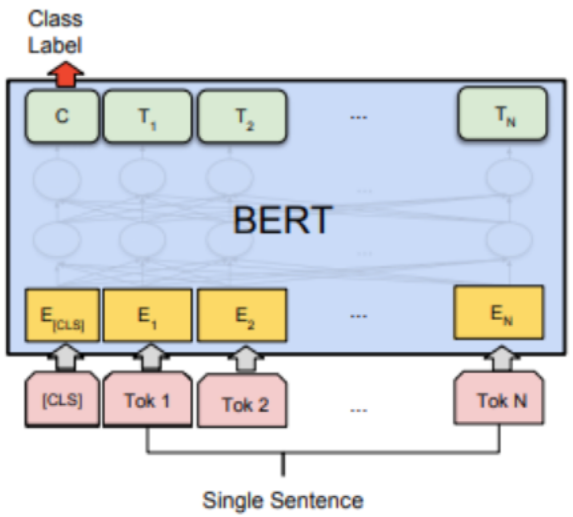
2. In Question Answering tasks (e.g. SQuAD v1.1), the software receives a question regarding a text sequence and is required to mark the answer in the sequence. Using BERT, a Q&A model can be trained by learning two extra vectors that mark the beginning and the end of the answer.

3. In Named Entity Recognition (NER), the software receives a text sequence and is required to mark the various types of entities (Person, Organization, Date, etc) that appear in the text. Using BERT, a NER model can be trained by feeding the output vector of each token into a classification layer that predicts the NER label.

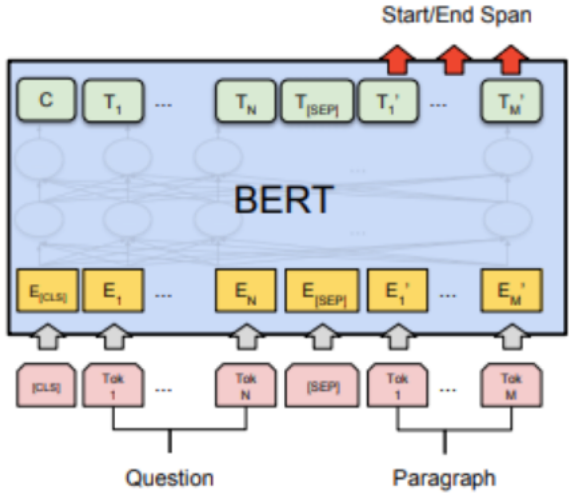
Using BERT



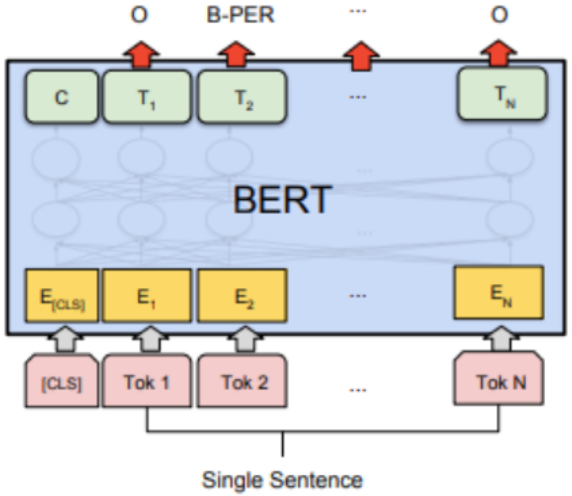
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



(b) Single Sentence Classification Tasks: SST-2, CoLA

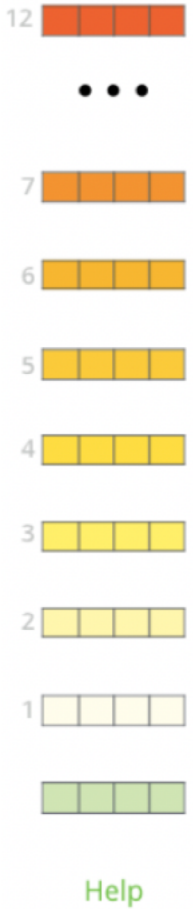


(c) Question Answering Tasks: SQuAD v1.1



(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Using BERT



For named-entity recognition task CoNLL-2003 NER

		Dev F1 Score
First Layer	Embedding	91.0
Last Hidden Layer	12	94.9
Sum All 12 Layers	12 + ... + 2 + 1 =	95.5
Second-to-Last Hidden Layer	11	95.6
Sum Last Four Hidden	12 + 11 + 10 + 9 =	95.9
Concat Last Four Hidden	9 10 11 12	96.1

Machine Translation: WMT-2014 BLEU

	EN-DE	EN-FR
GNMT (orig)	24.6	39.9
ConvSeq2Seq	25.2	40.5
Transformer*	**28.4**	**41.8**

*Transformer models trained >3x faster than the others.

Attention is All You Need (NeurIPS 2017) Vaswani*, Shazeer*, Parmar*, Uszkoreit*, Jones*, Kaiser*, Gomez*, Polosukhin*

GPT

June 2018

Training

240 GPU days

BERT

Oct 2018

Training

256 TPU days

~320–560 GPU days

GPT-2

Feb 2019

Training

~2048 TPU v3 days according to a reddit thread

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova

(Submitted on 11 Oct 2018)

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

Table 1: GLUE Test results, scored by the GLUE evaluation server. The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set. OpenAI GPT = (L=12, H=768, A=12); $\text{BERT}_{\text{BASE}}$ = (L=12, H=768, A=12); $\text{BERT}_{\text{LARGE}}$ = (L=24, H=1024, A=16). BERT and OpenAI GPT are single-model, single task. All results obtained from https://gluebenchmark.com/leaderboard and https://blog.openai.com/language-unsupervised/.

References

- https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21
- https://mc.ai/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients/
- https://medium.com/datadriveninvestor/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577
- http://jalammar.github.io/illustrated-transformer/

THANK YOU!