Deep Learning an Introduction

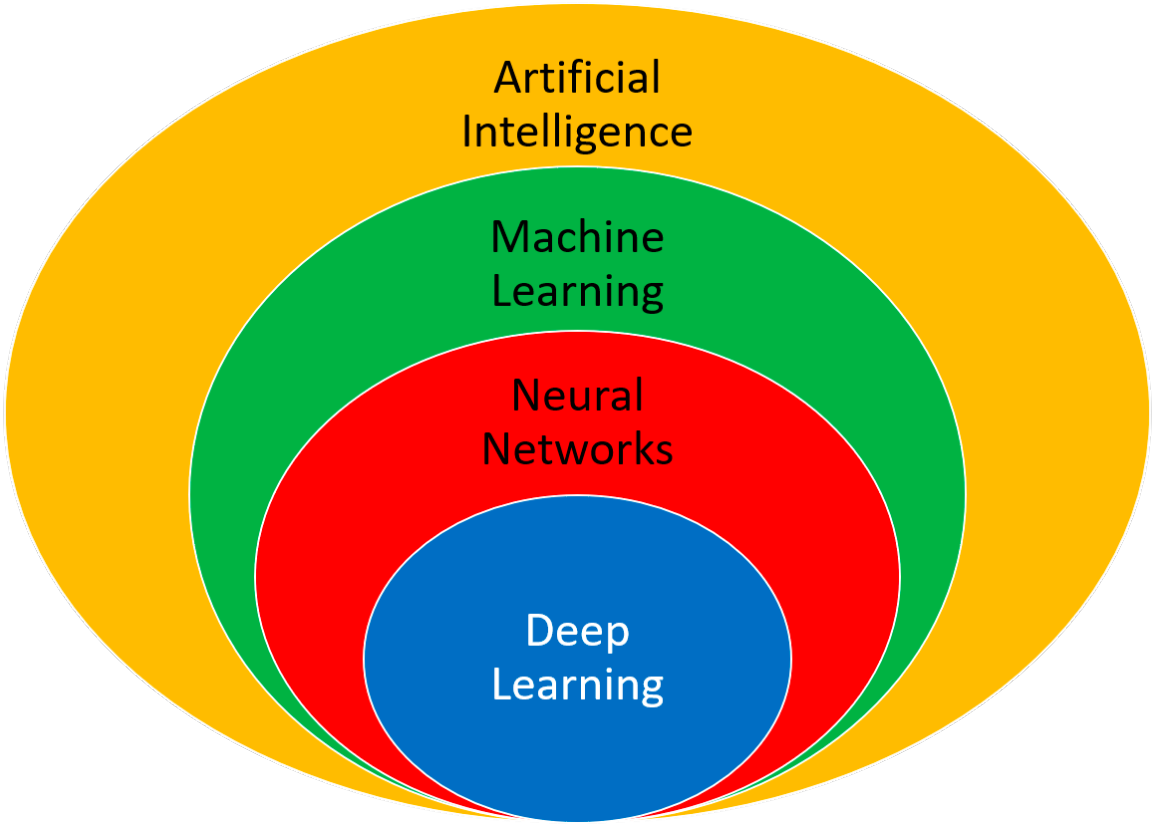
Lê Anh Cường

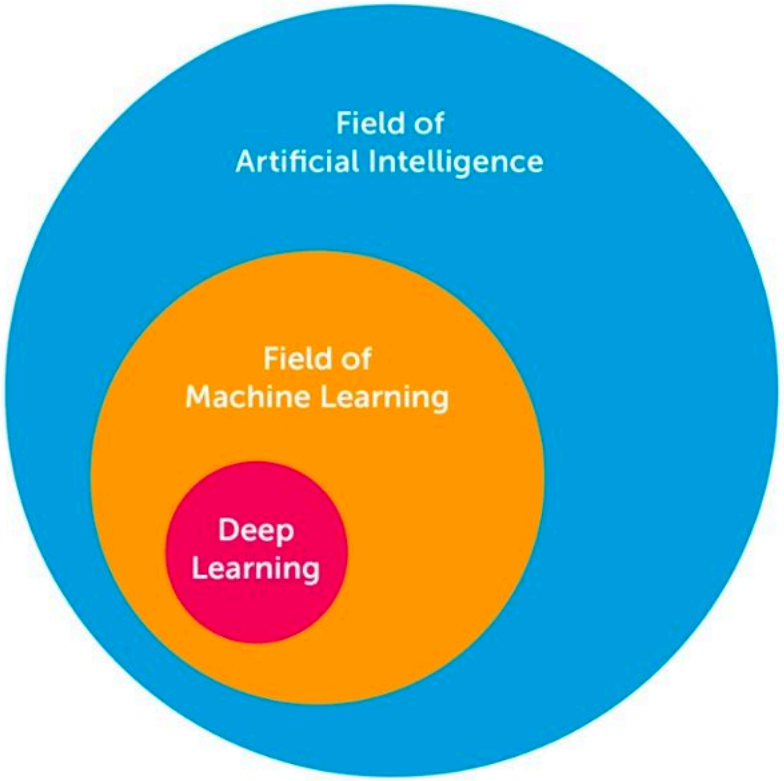
2020

Outline

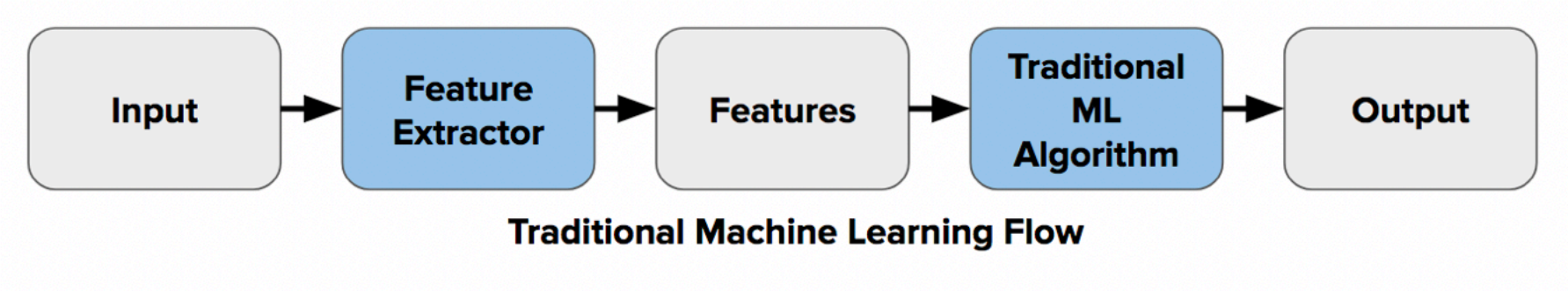
1. How to recognize Deep Learning networks/models:
   - Traditional statistical machine learning vs Deep Learning
   - Vanilla neural networks vs DL networks
   - Deep Belief Network
   - History of DL development
2. Convolutional Neural Network (CNN)
3. Long Short Term Memory (LSTM)

1. Introduction of Machine Learning





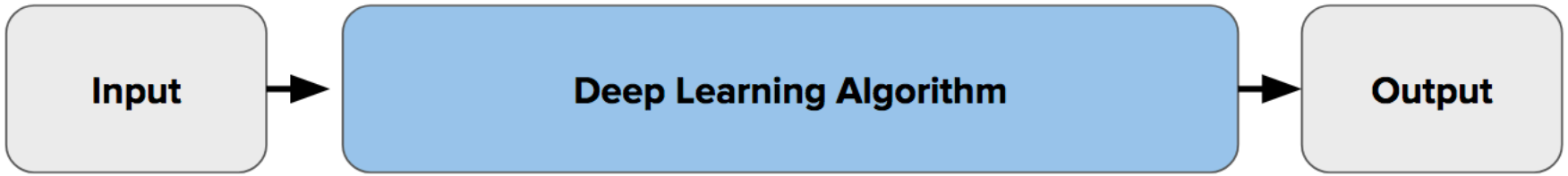
Traditional Statistical Machine Learning vs Deep Learning



Traditional Machine Learning Flow

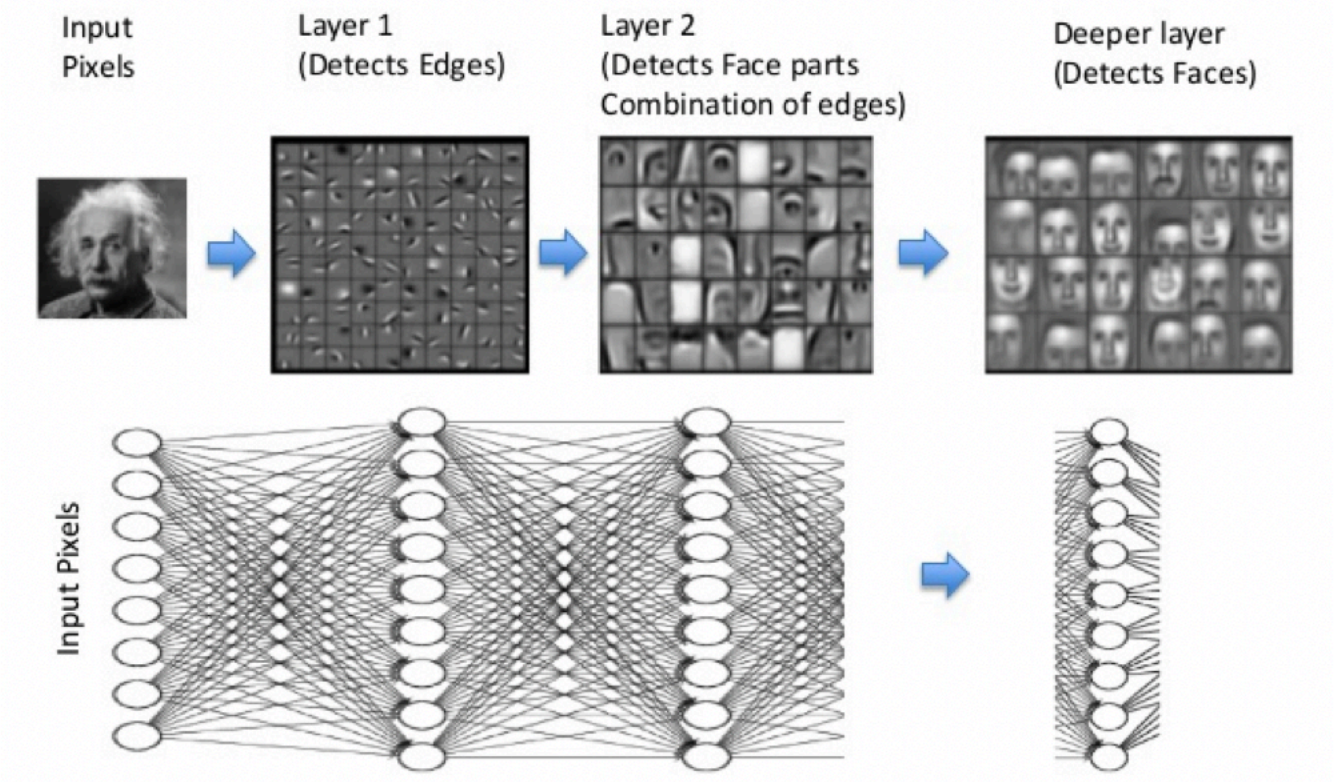
Traditional Statistical Machine Learning vs Deep Learning

Traditional Machine Learning Flow

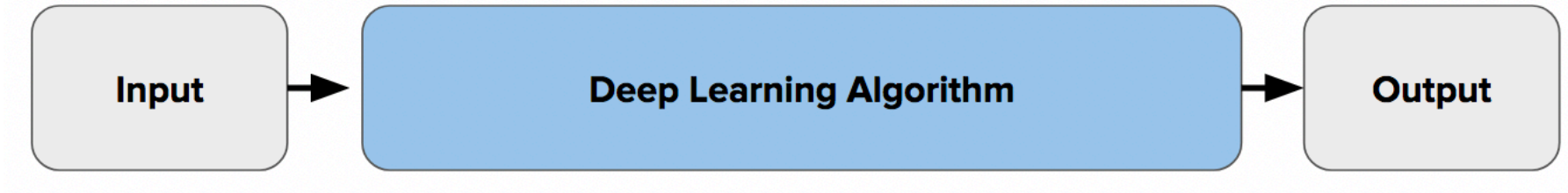


Deep Learning Flow

DL as Representation Learning

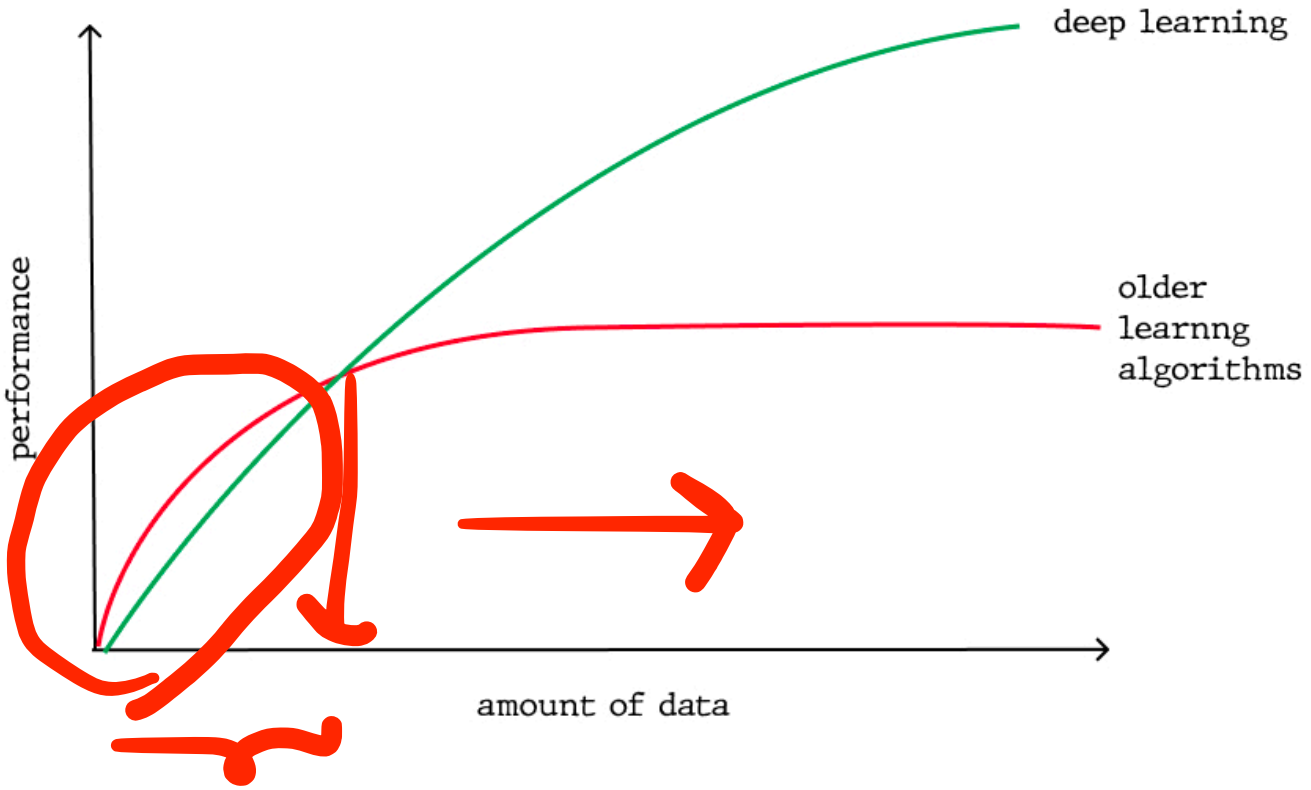


Deep Learning
End-to-end model

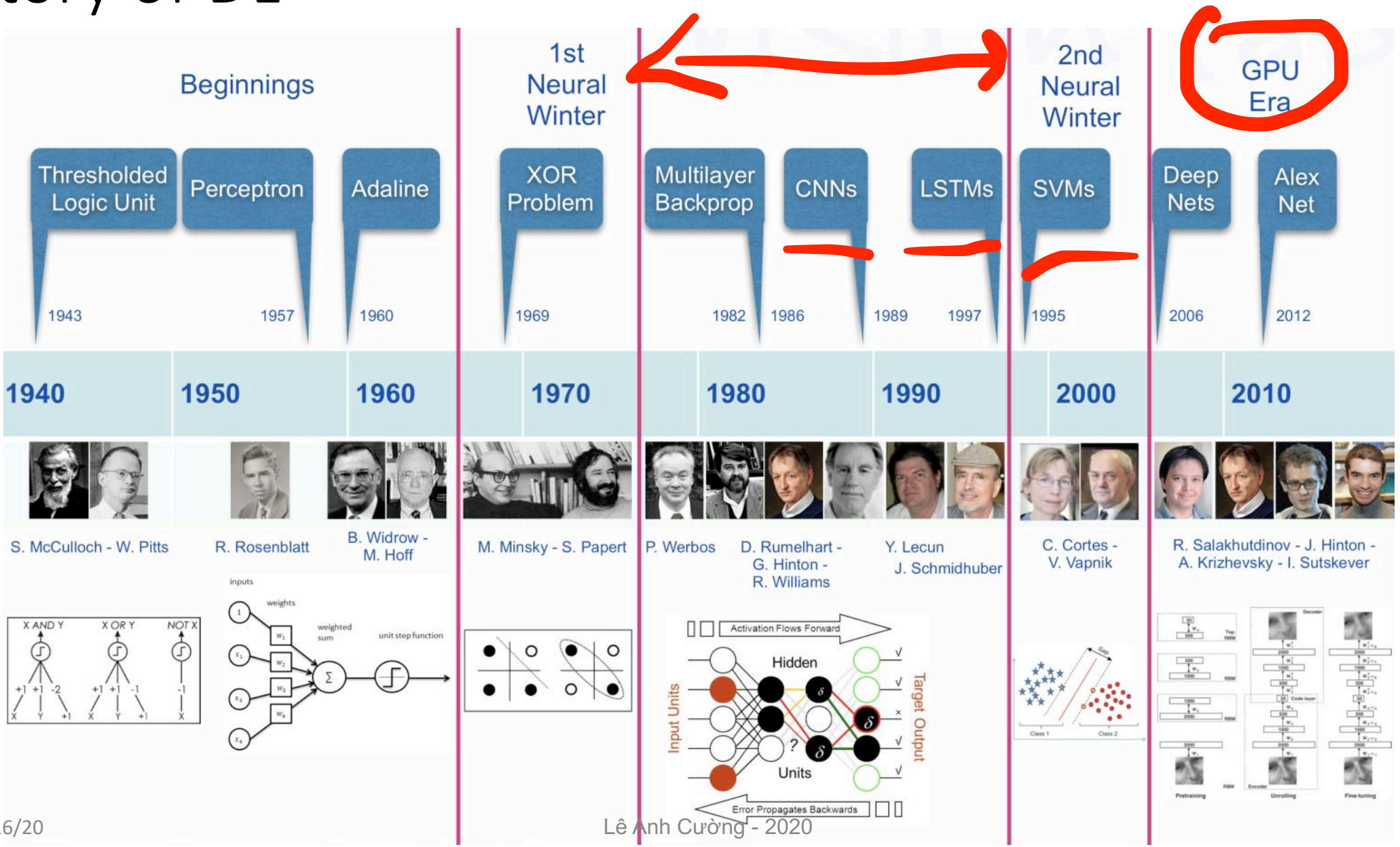


Deep Learning Flow

Result


deep learning
older
learnng
algorithms
performance
amount of data

History of DL



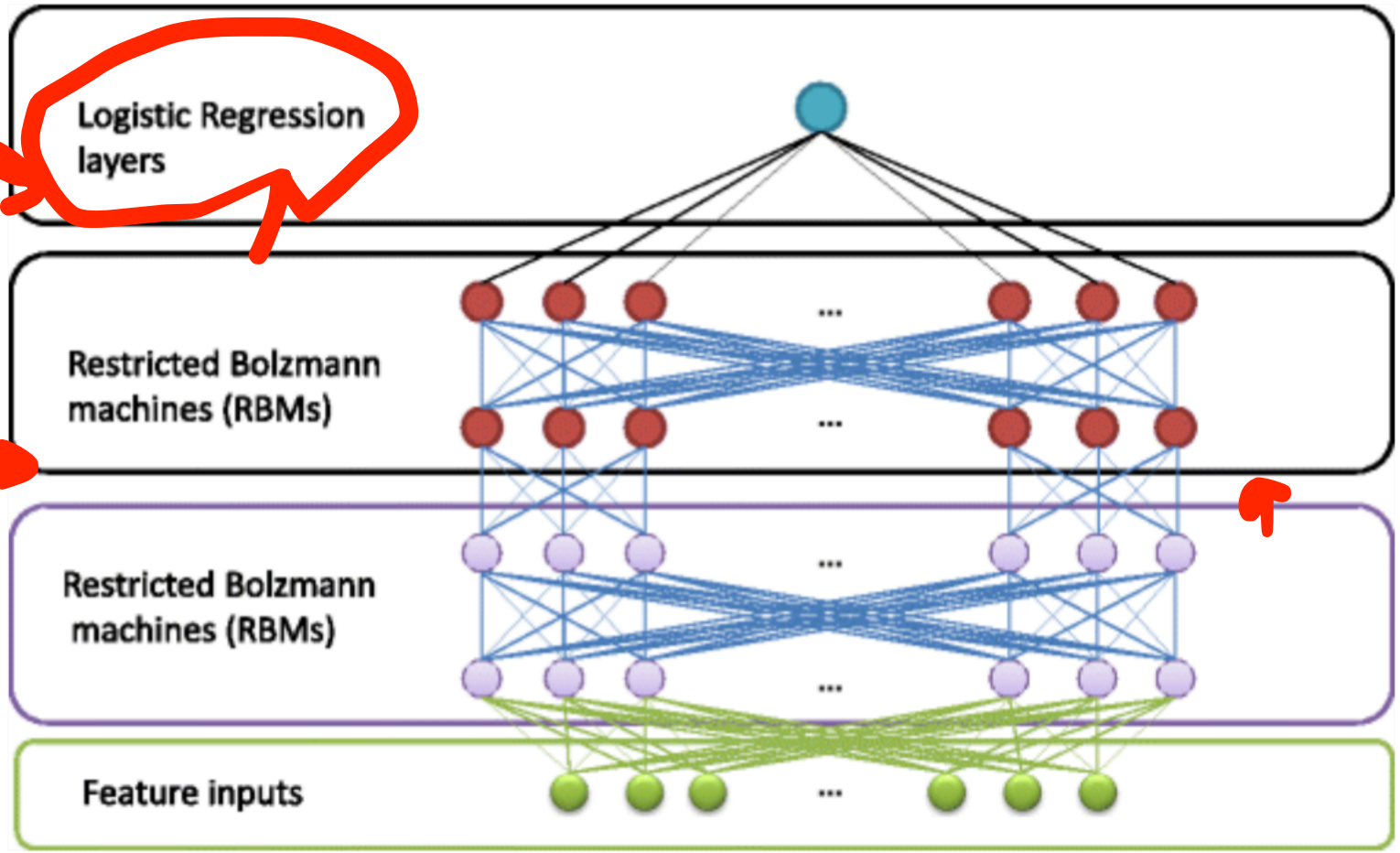
When does the term “Deep Learning” begin?

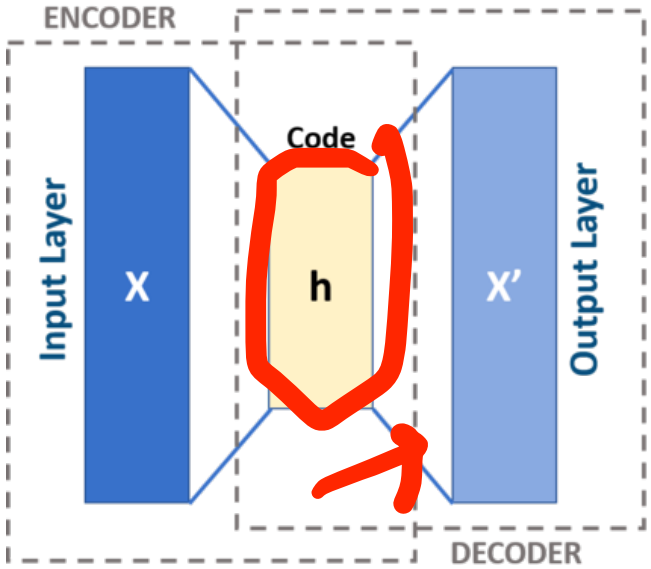
Geoffrey Hinton is a pioneer in the field of artificial neural networks and co-published the first paper on the backpropagation algorithm for training multilayer perceptron networks.

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

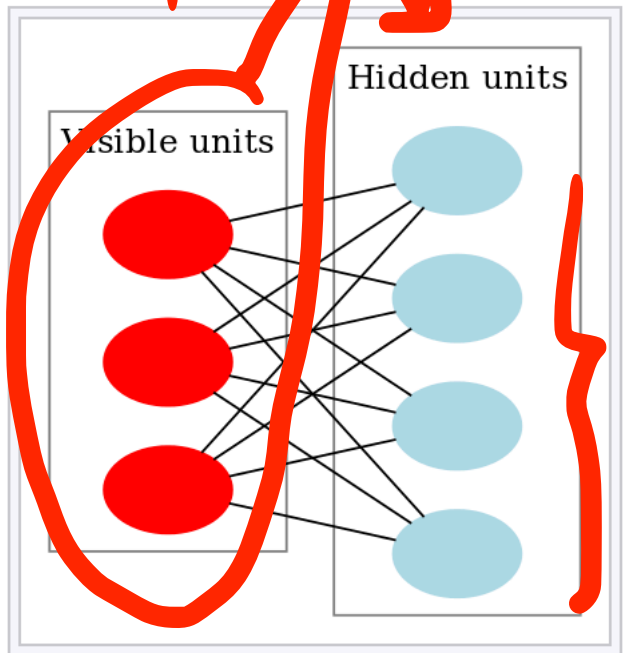
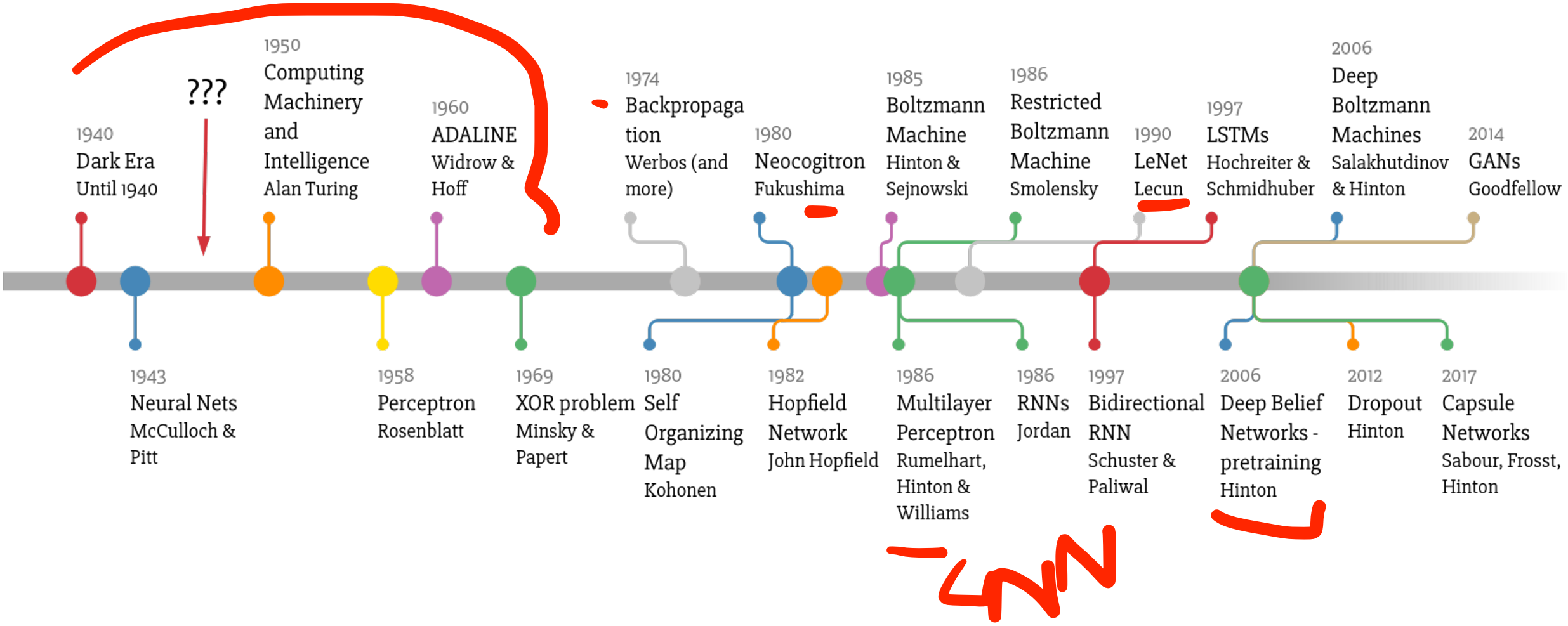


Diagram of a restricted Boltzmann machine with three visible units and four hidden units (no bias units).

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

Deep Learning Timeline



Made by Favio Vázquez

Characteristics of DL

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

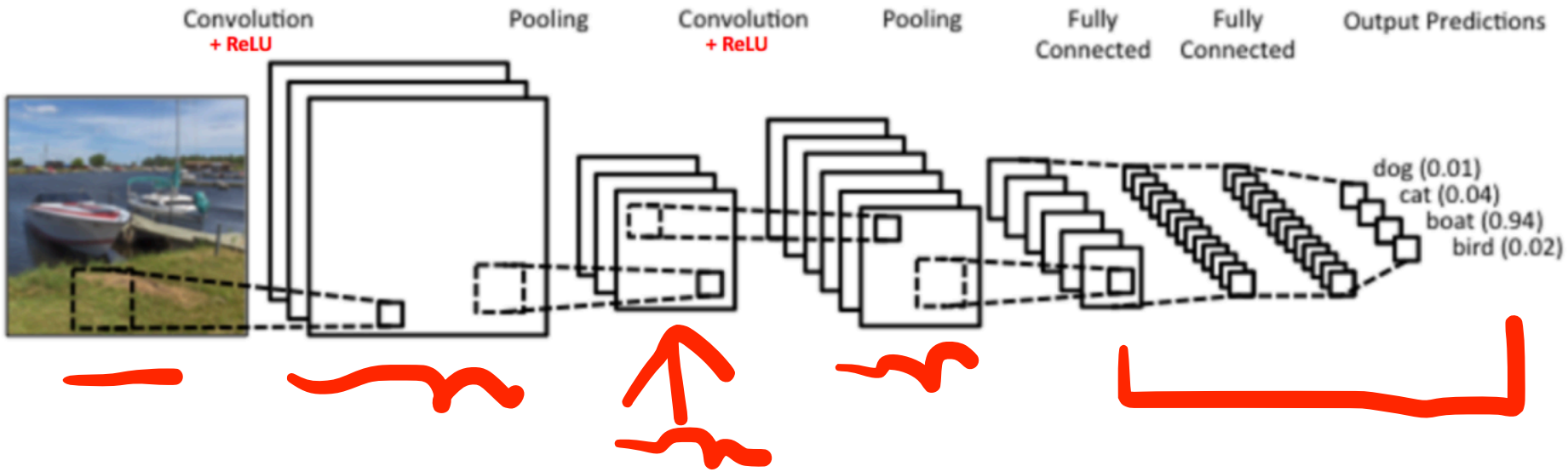
Convolutional Neural Network

Outline

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

What is Convolutional Neural Network

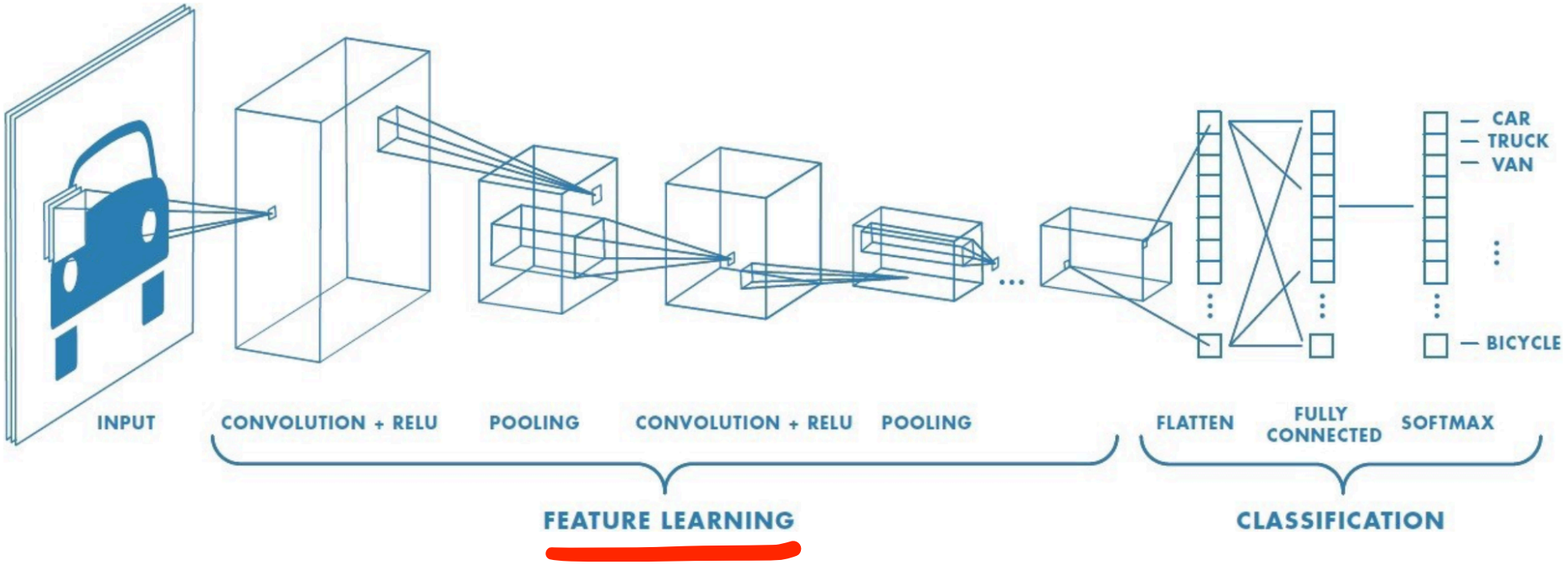
- Convolutional networks are simply neural networks that use convolution operation which is a specialized kind of linear operation.
- CNNs are regularized versions of multilayer perceptrons.
- CNNs take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and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



ConvNet History

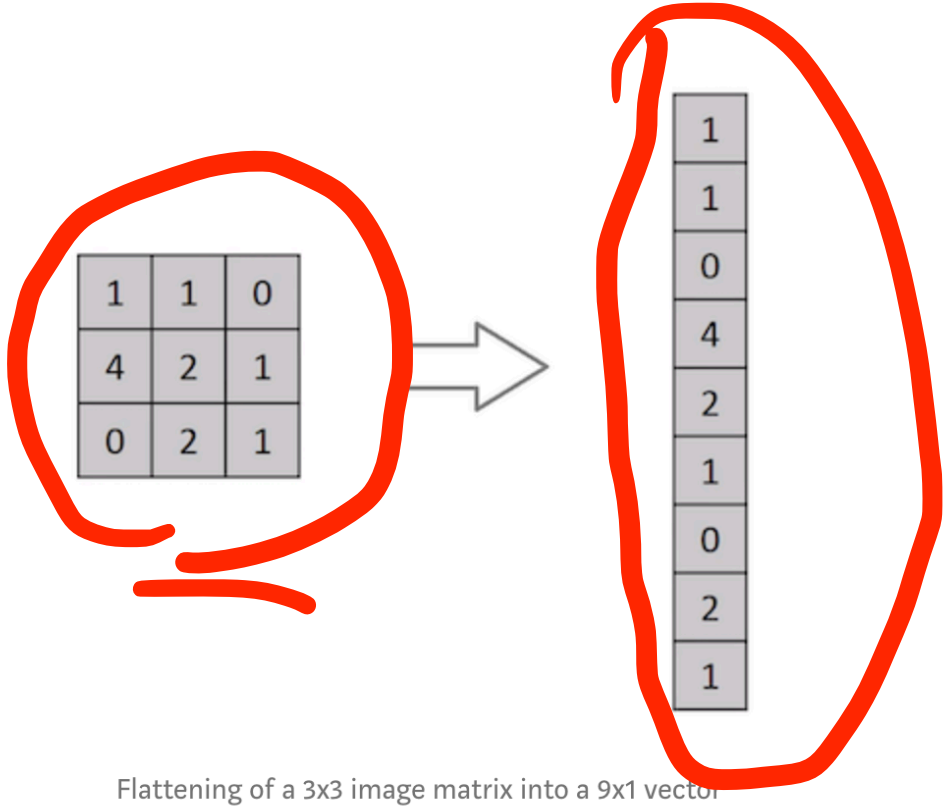
- Convolutional neural networks, also called ConvNets, were first introduced in the 1980s by Yann LeCun for building an image recognition neural network.
- LeNet-5, a pioneering 7-level convolutional network by LeCun et al. in 1998, that classifies digits, was applied by several banks to recognize hand-written 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



Why ConvNets over Feed-Forward Neural Nets?

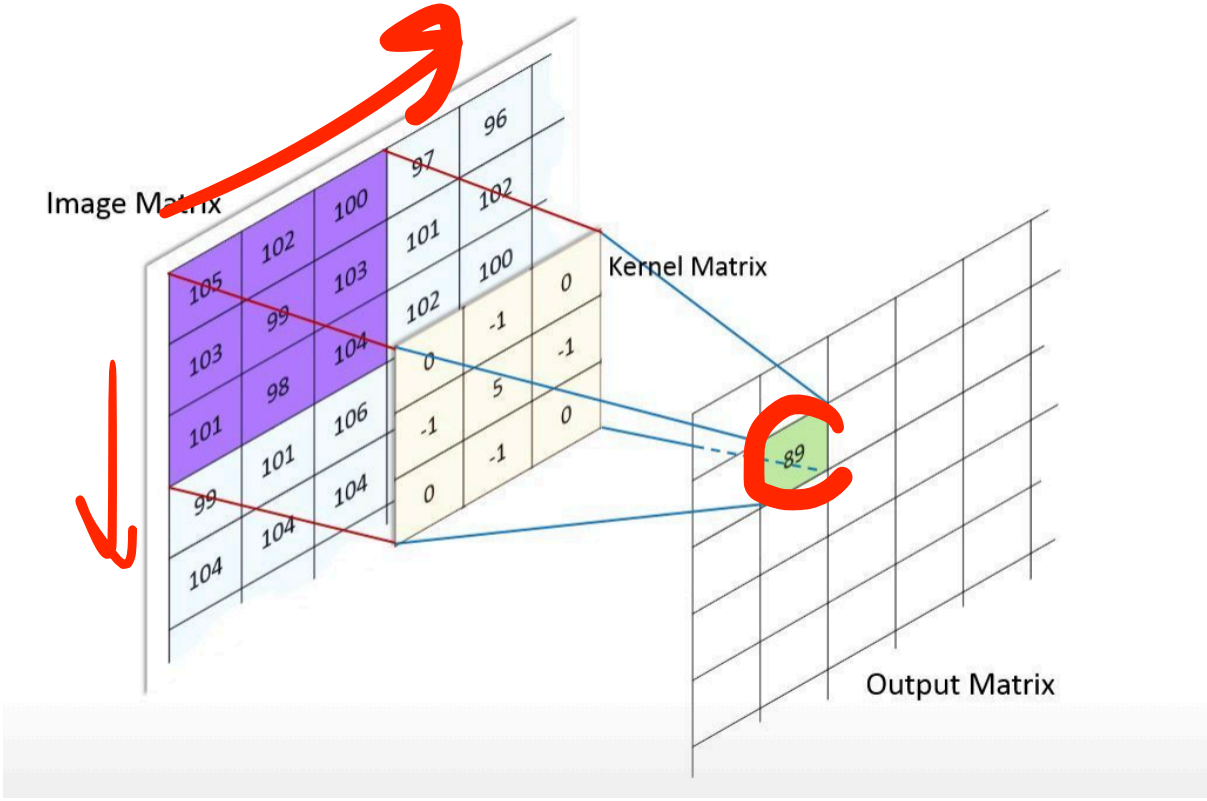
- A ConvNet is able to **successfully capture the Spatial and Temporal dependencies.**
- The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters.
- The network can be trained to understand the sophistication of the image better.



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. ConvNets need not be limited to only one Convolutional Layer.

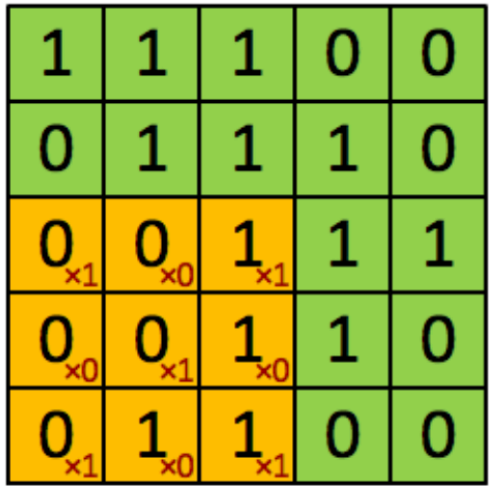


Convolution Layer — The Kernel

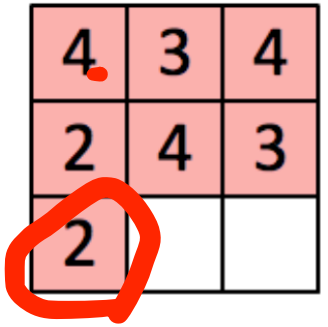
A kernel is, a matrix of weights. The dimensions of the kernel matrix is *how the convolution gets it's name*. For example, in 2D convolutions, the kernel matrix is a 2D matrix.

Kernel

```
1  0  1
0  1  0
1  0  1
```

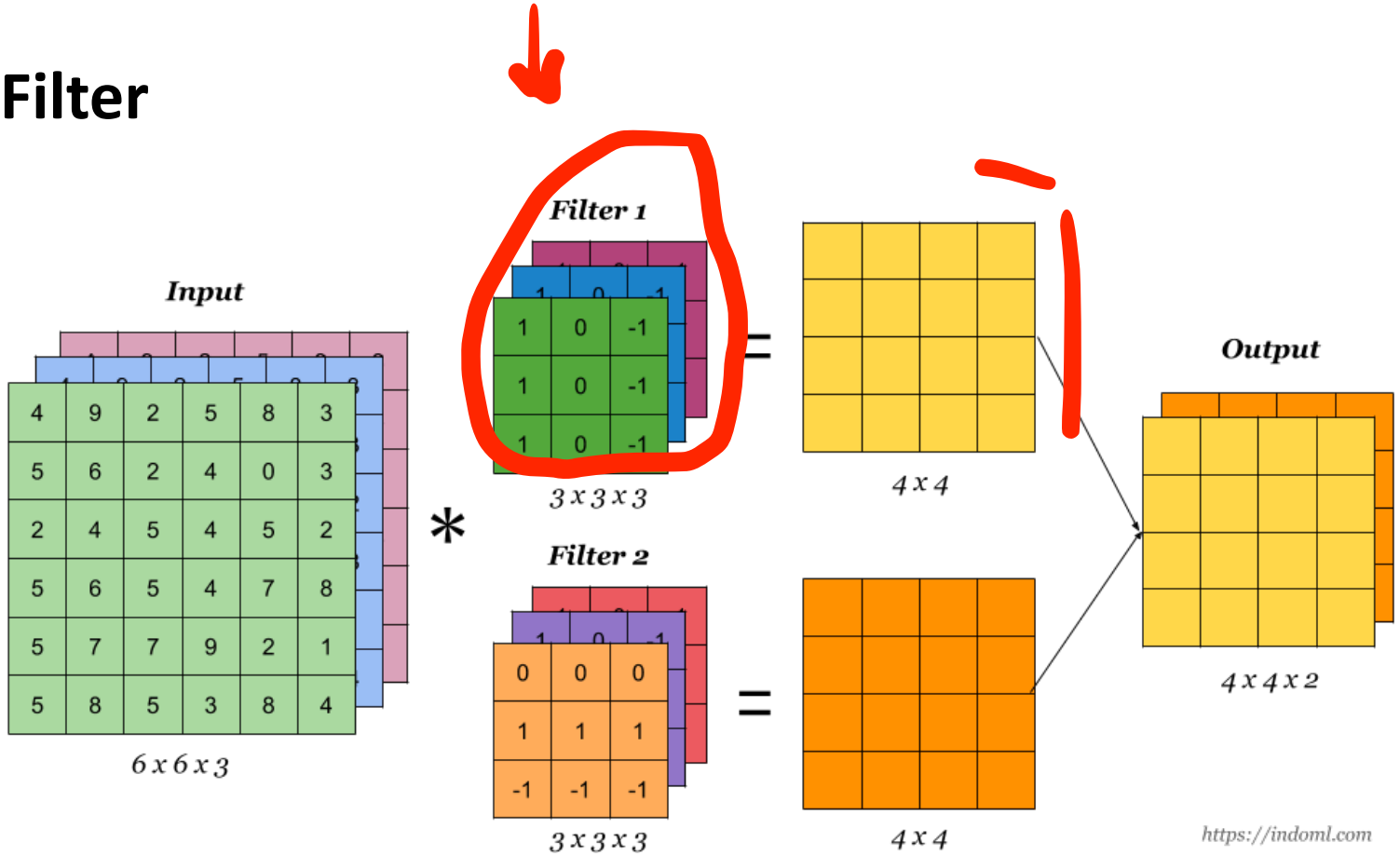


Image



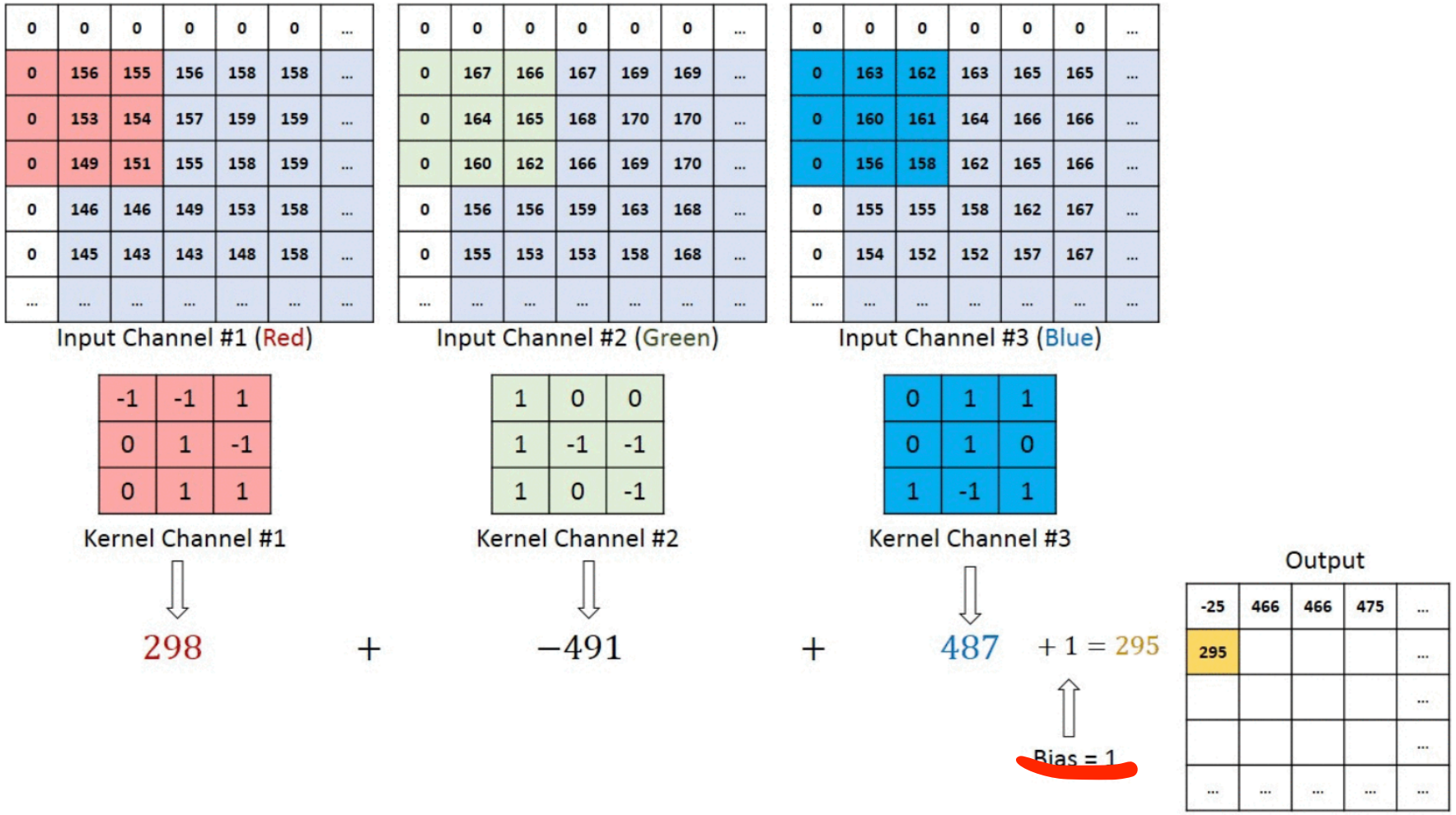
Convolved Feature

Kernel and Filter



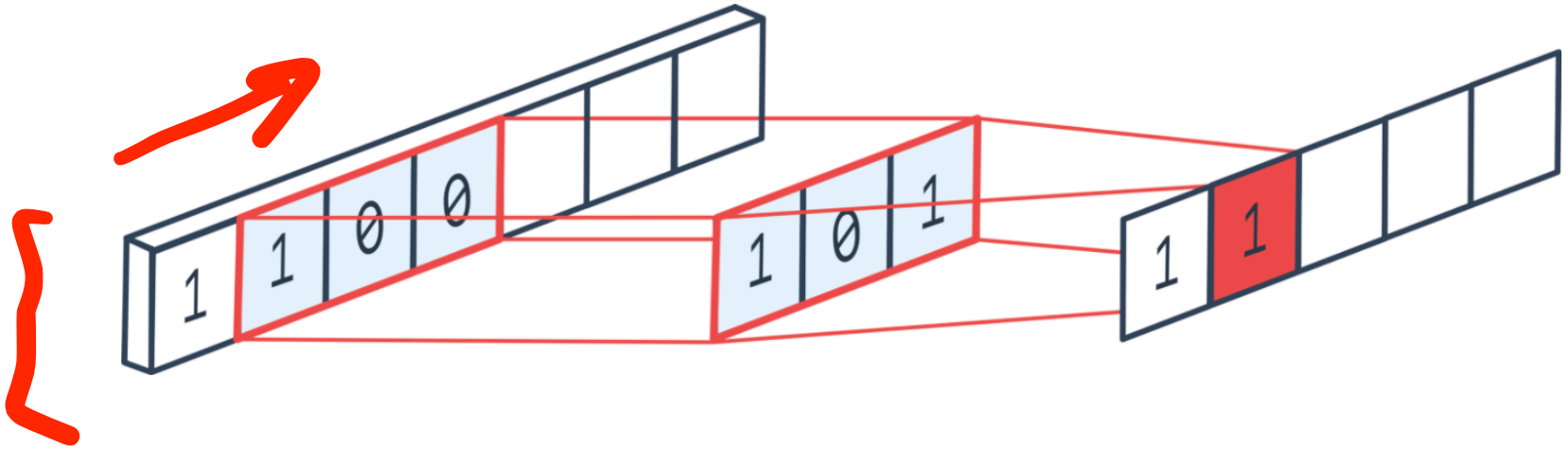
A common convolution layer actually consist of multiple such filters. *For the sake of simplicity in the discussion to follow, assume the presence of only one filter unless specified, since the same behavior is replicated across all the filters.*

a *multi-input channel* convolution kernel

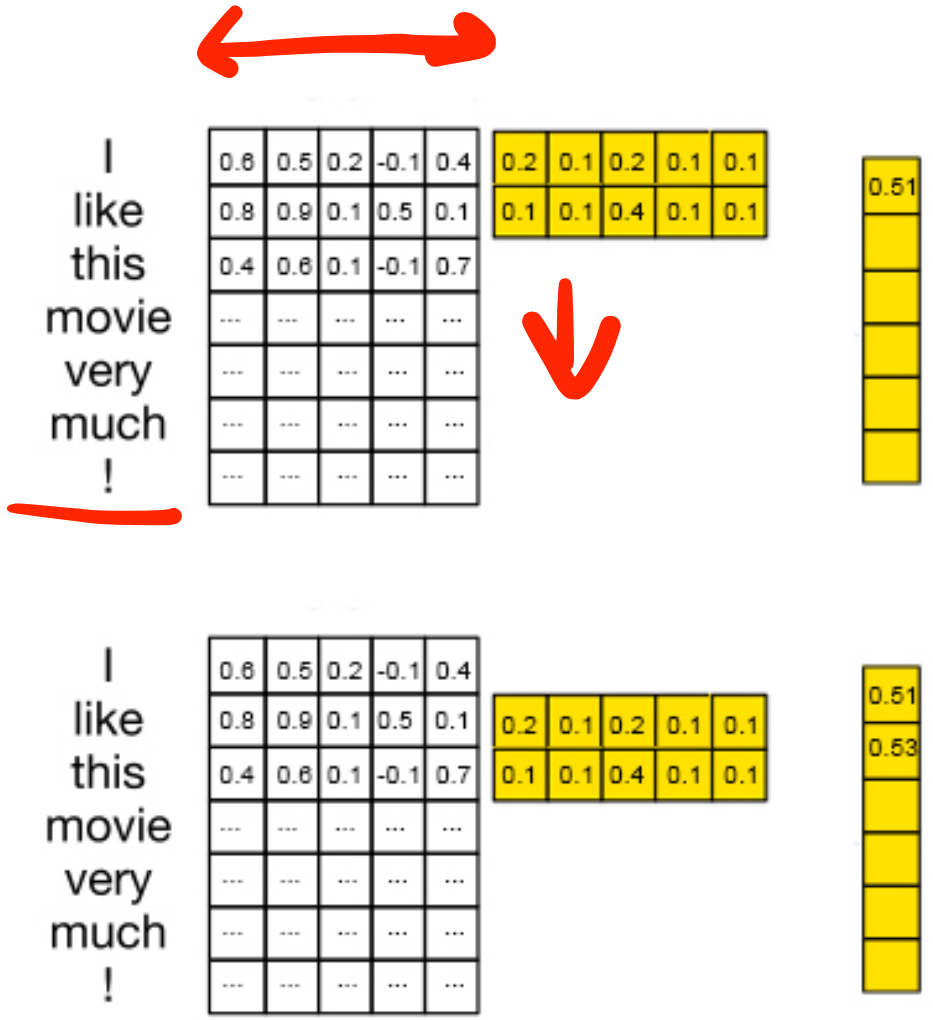


Convolution operation on a MxNx3 image matrix with a 3x3x3 Kernel

1D Convolutions

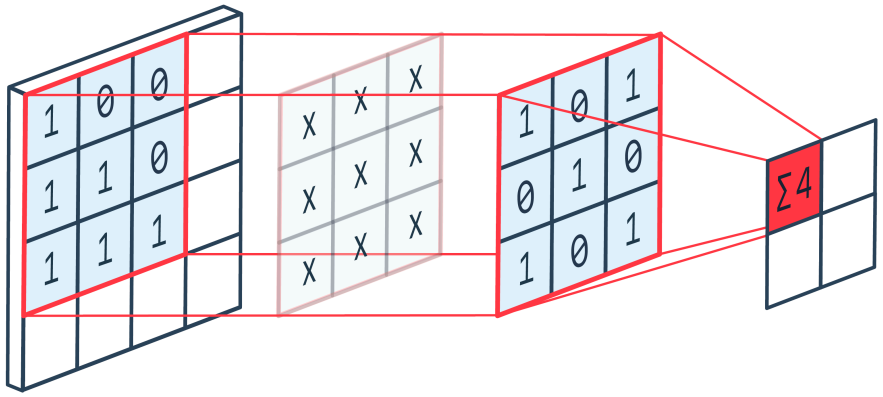


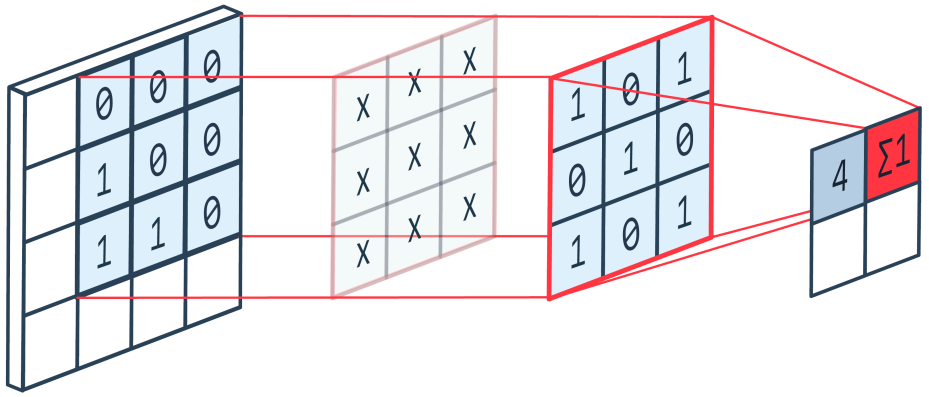
1D Convolution with 2D input



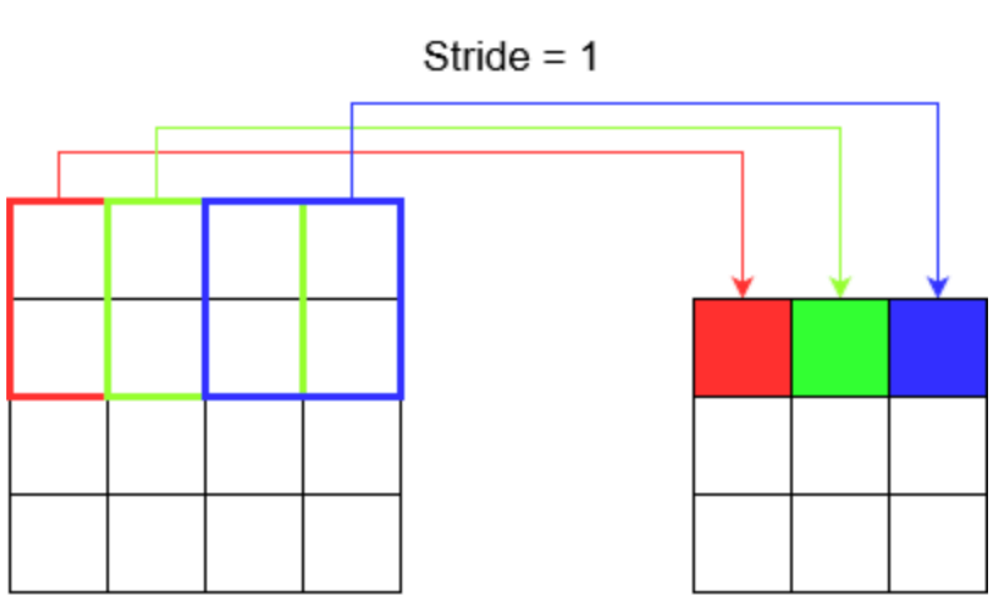
2D Convolutions

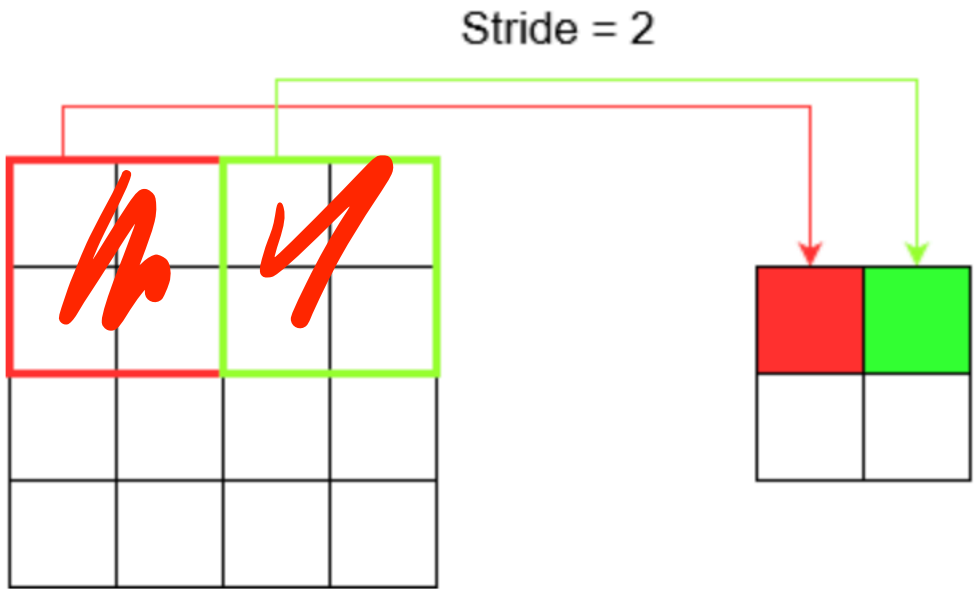
- The filter can move in 2 directions and thus the final output is 2D. 2D convolutions are the most common convolutions, and are heavily used in Computer Vision.



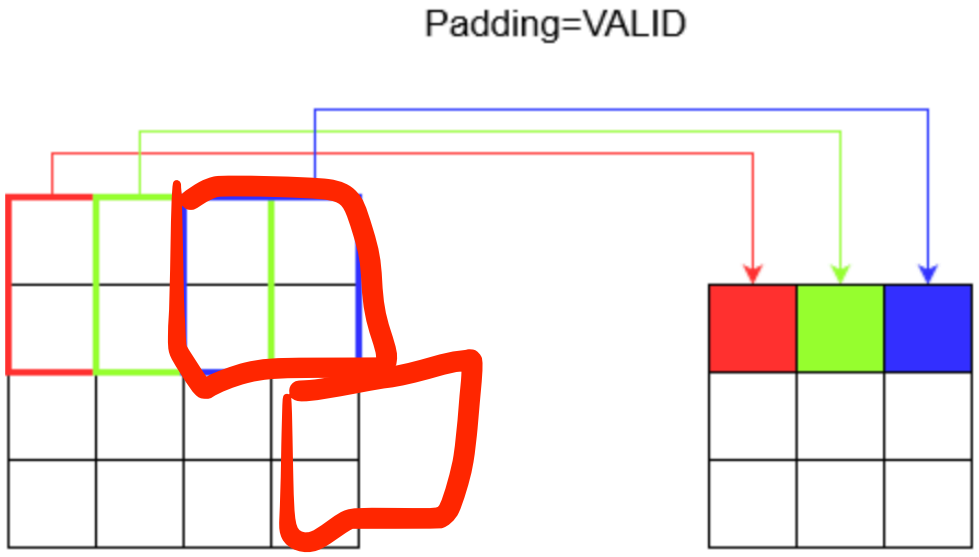


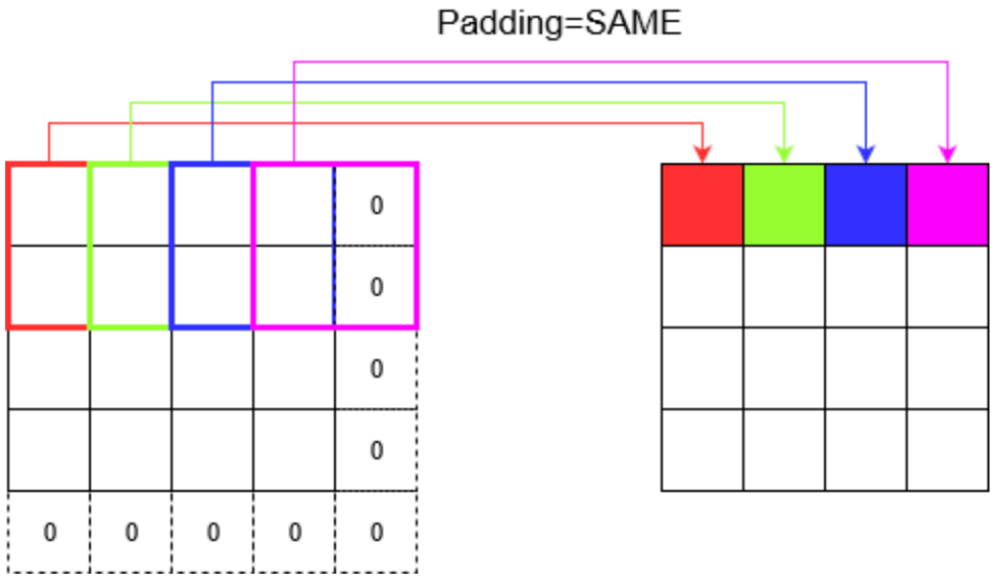
Stride



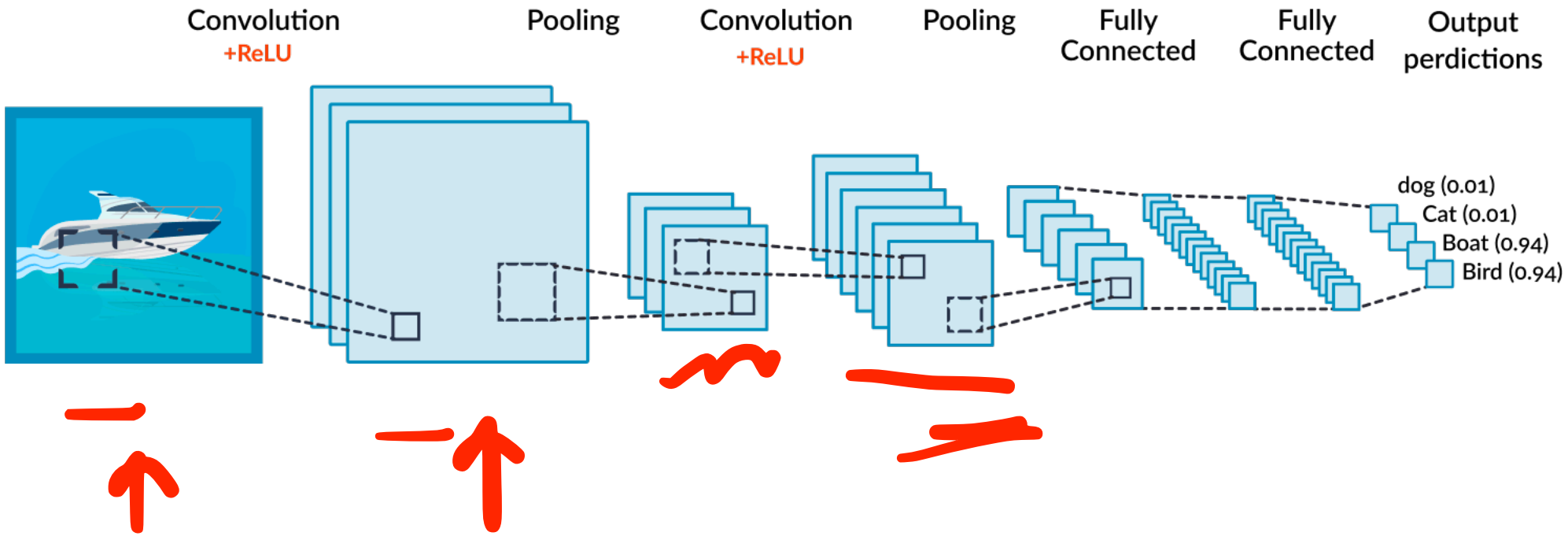


Padding



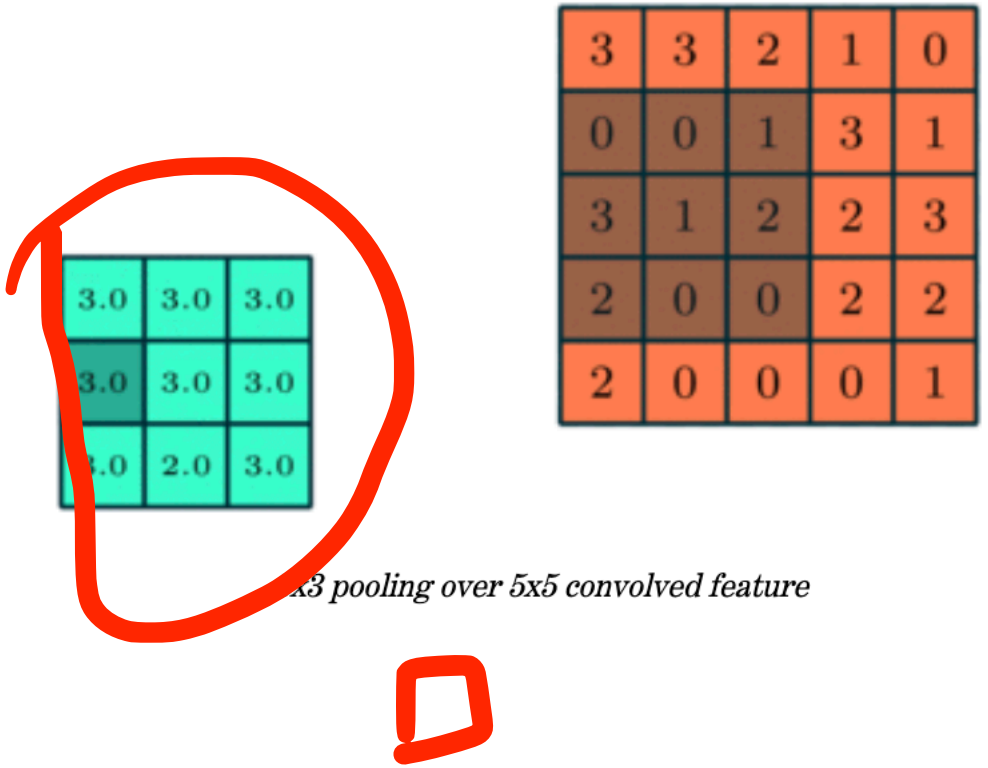


Pooling Layer

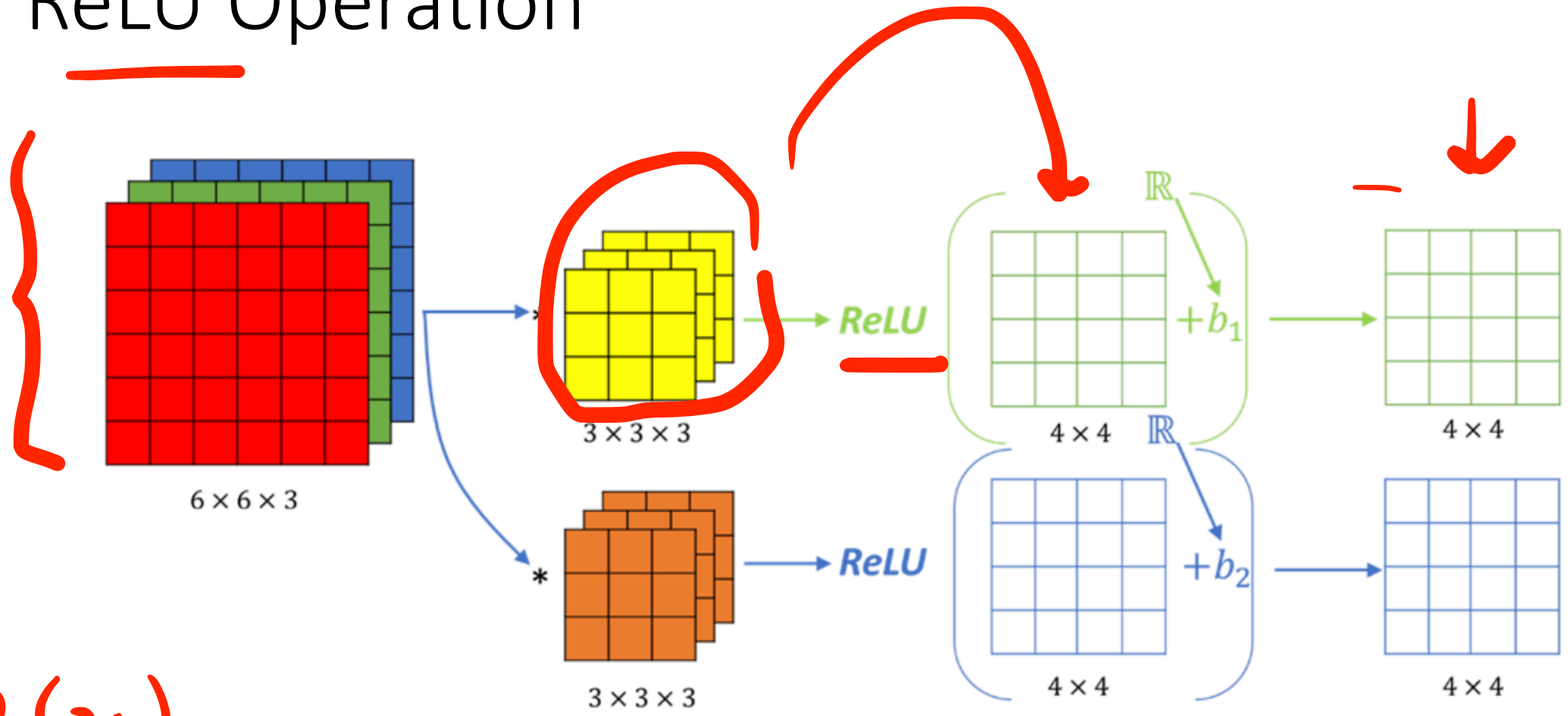


Pooling Layer

- 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** which are rotational and positional invariant, thus maintaining the process of effectively training of the model.



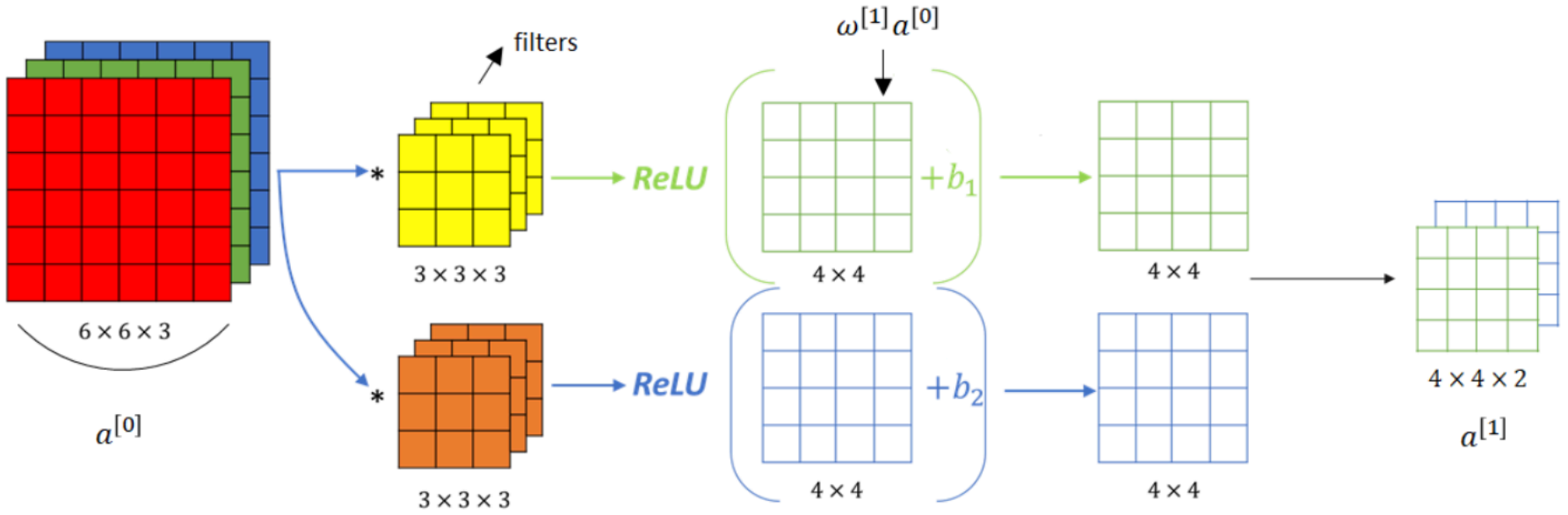
ReLU Operation



The result of convolution with two filters

$R(x) = \max(0, x)$

ReLU



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

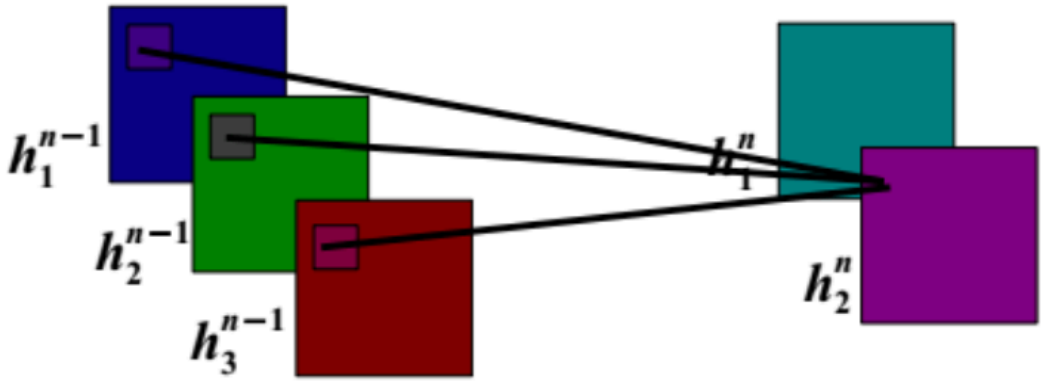
Convolutional Layer

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

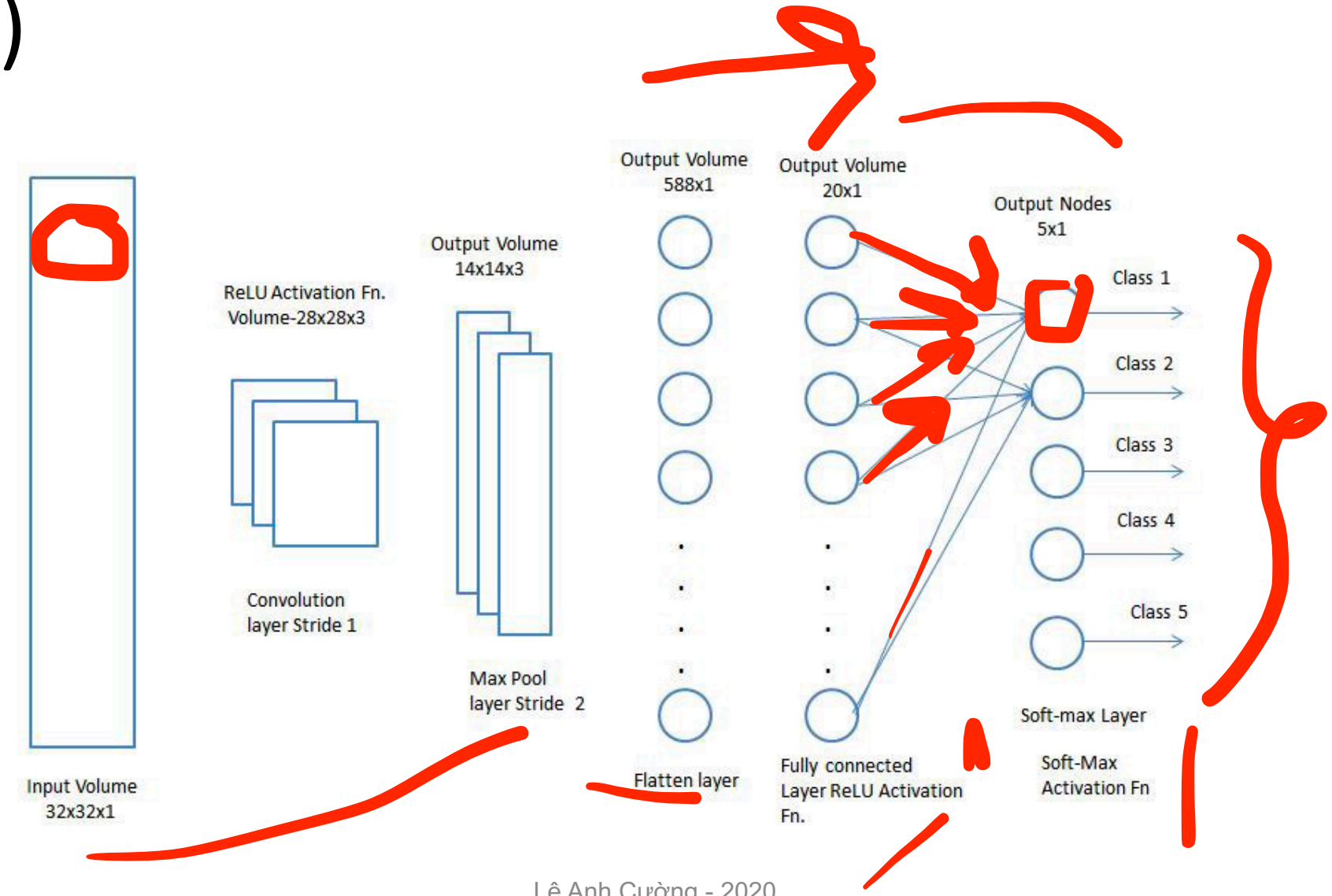
output feature map

input feature map

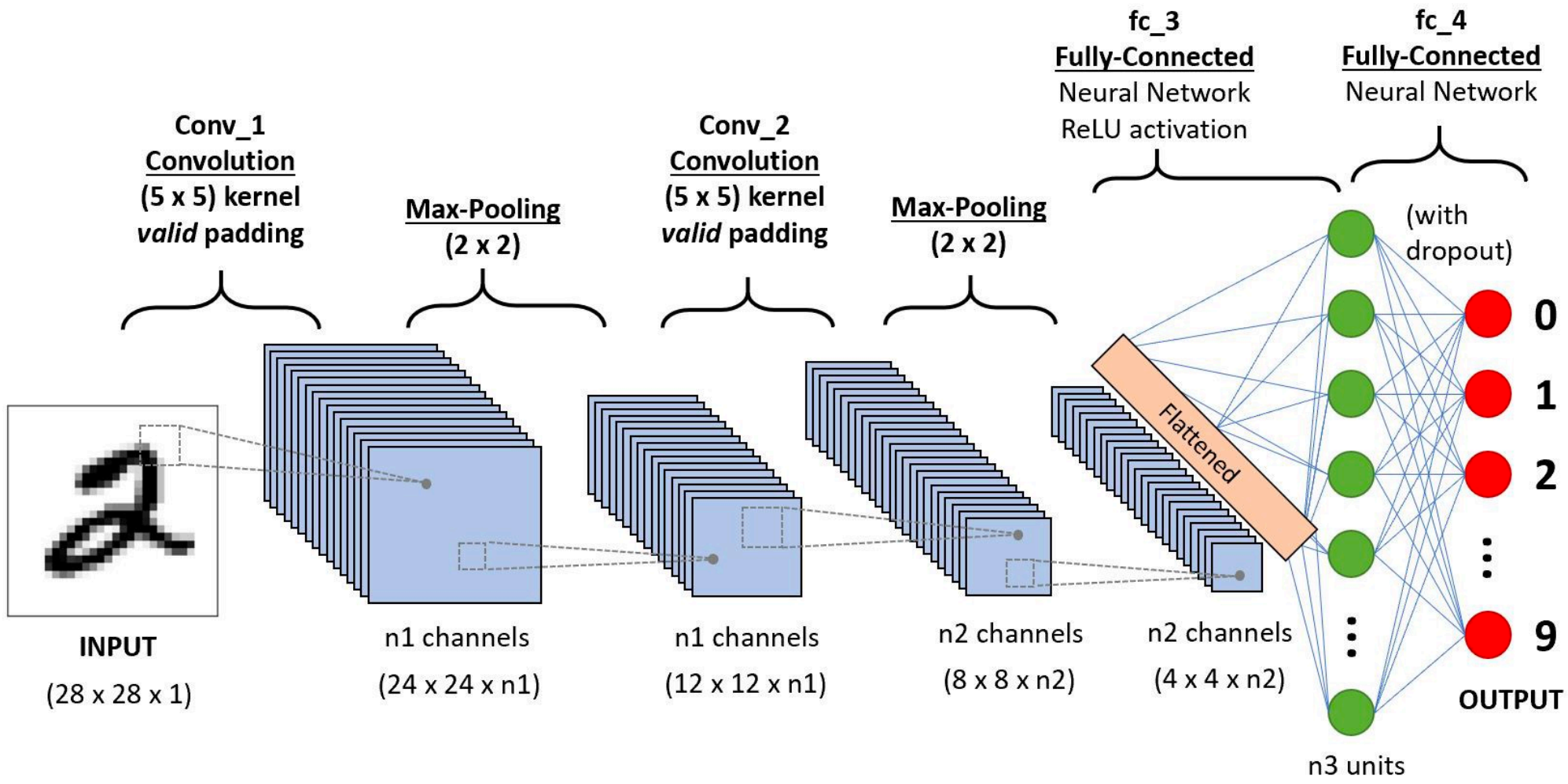
kernel

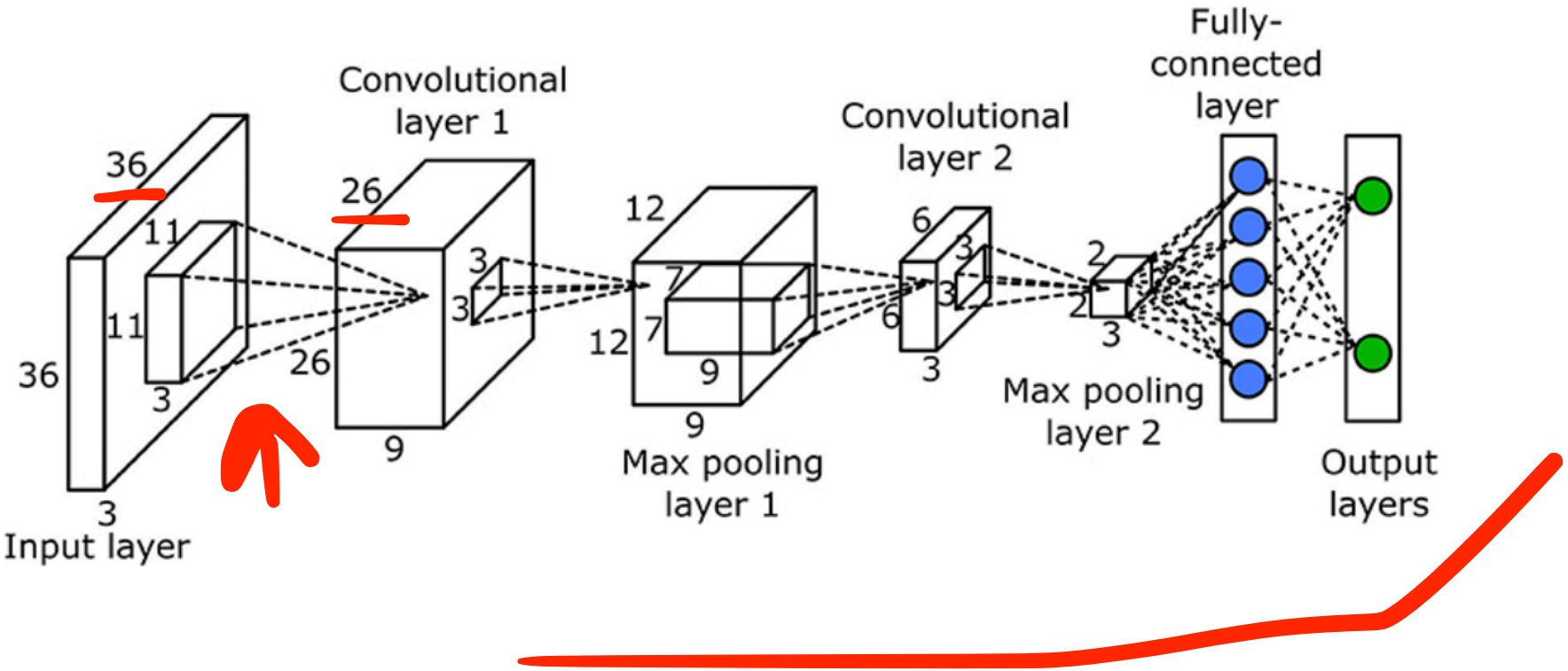


Classification — Fully Connected Layer (FC Layer)



Multi-Layer CNNs




36
36
11
11
3
3
Input layer
Convolutional
layer 1
26
26
3
3
9
12
12
7
7
9
9
Max pooling
layer 1
Convolutional
layer 2
6
6
3
3
3
2
2
3
Max pooling
layer 2
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. Some of them have been listed below:

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

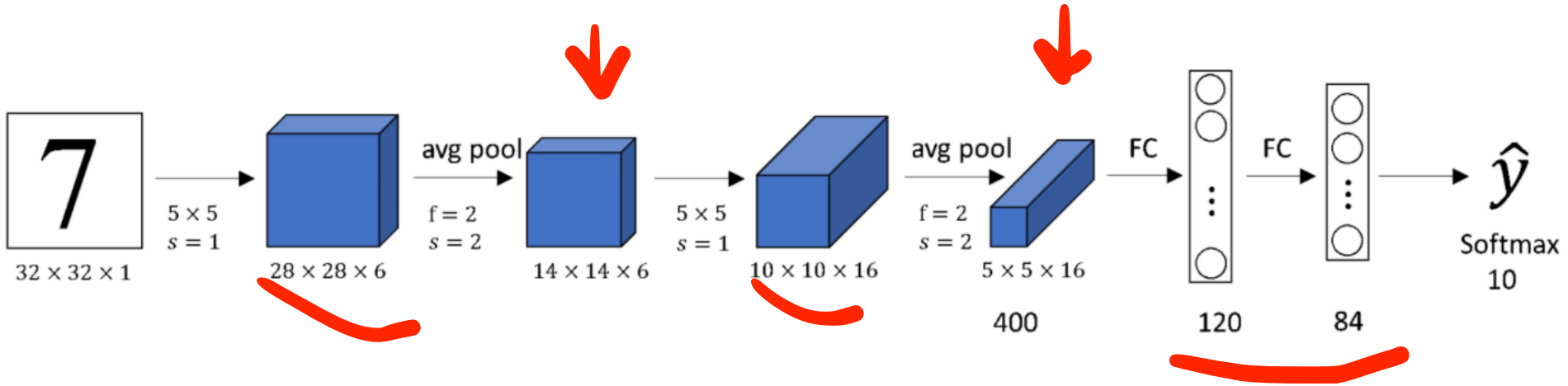
LeNet

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. Convolutional neural networks are a kind of feed-forward neural network whose artificial neurons can respond to a part of the surrounding cells in the coverage range and perform well in large-scale image processing.

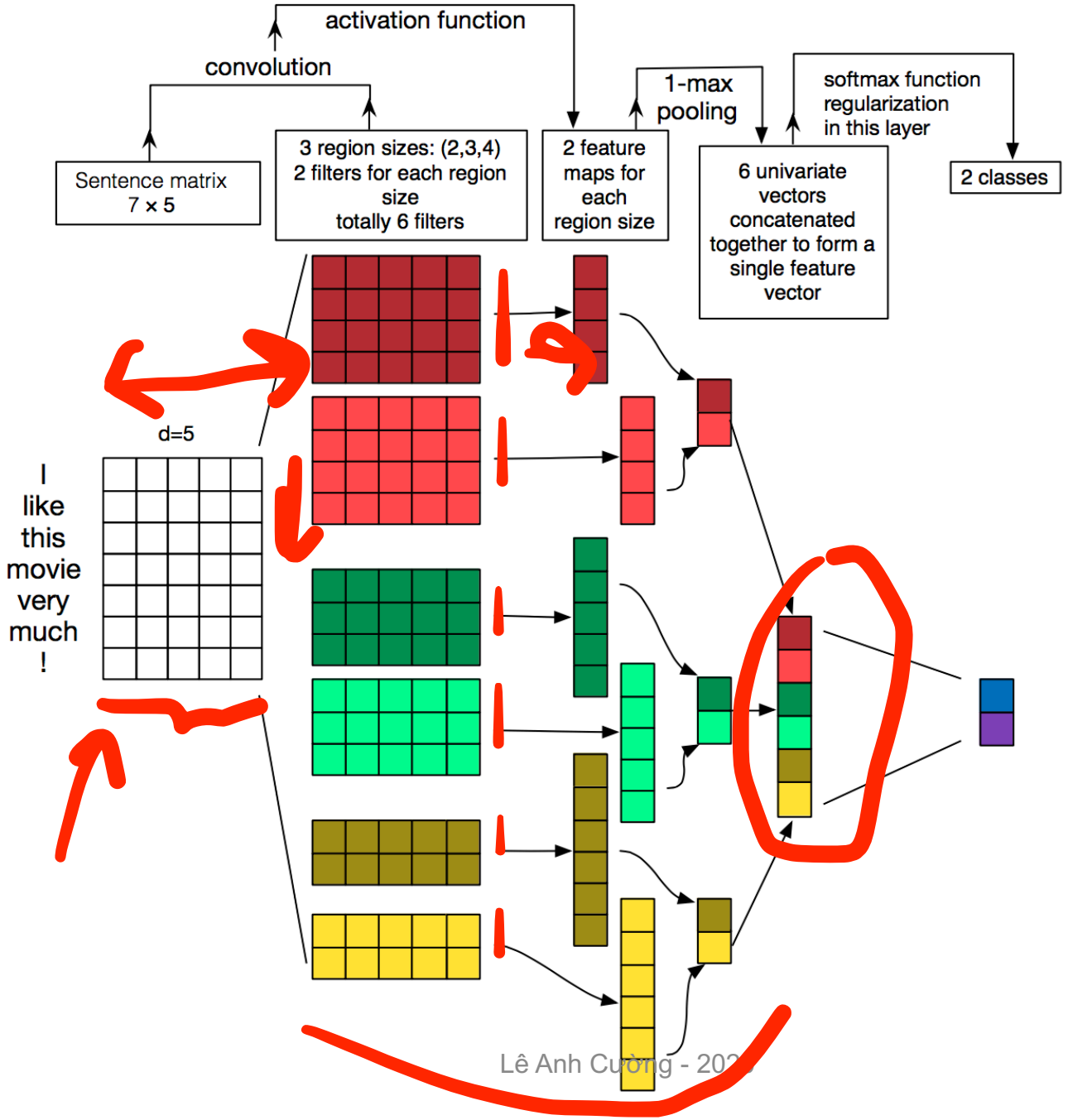
LeNet5 was one of the earliest convolutional neural networks and promoted the development of deep learning. Since 1988, after years of research and many successful iterations, the pioneering work has been named LeNet5.

LeNet-5

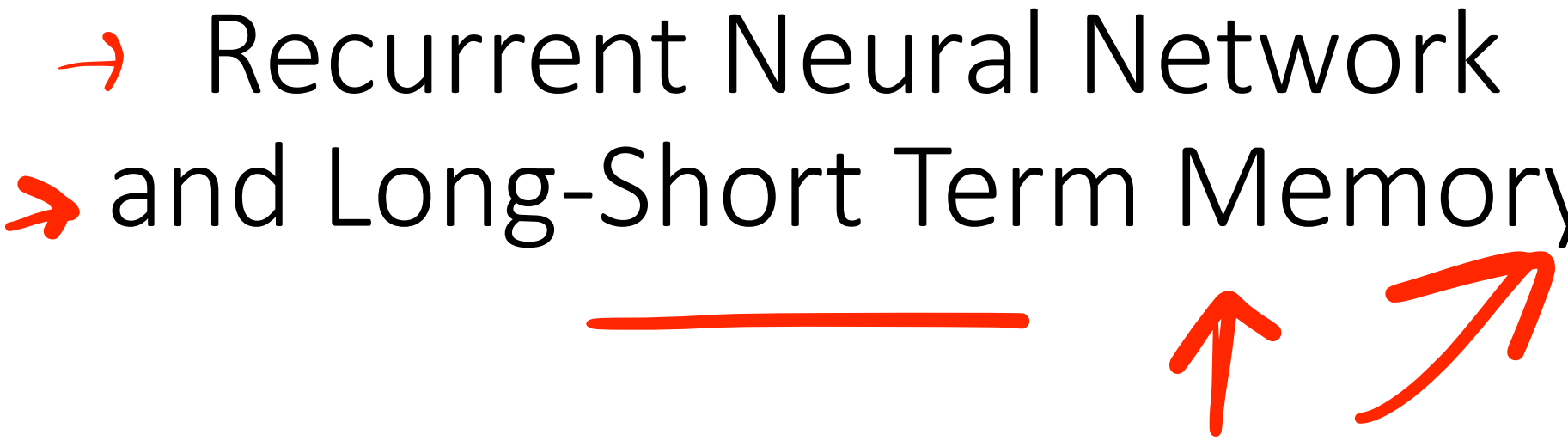
http://datahacker.rs/deep-learning-lenet-5-architecture/



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.


activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

Recurrent Neural Network and Long-Short Term Memory



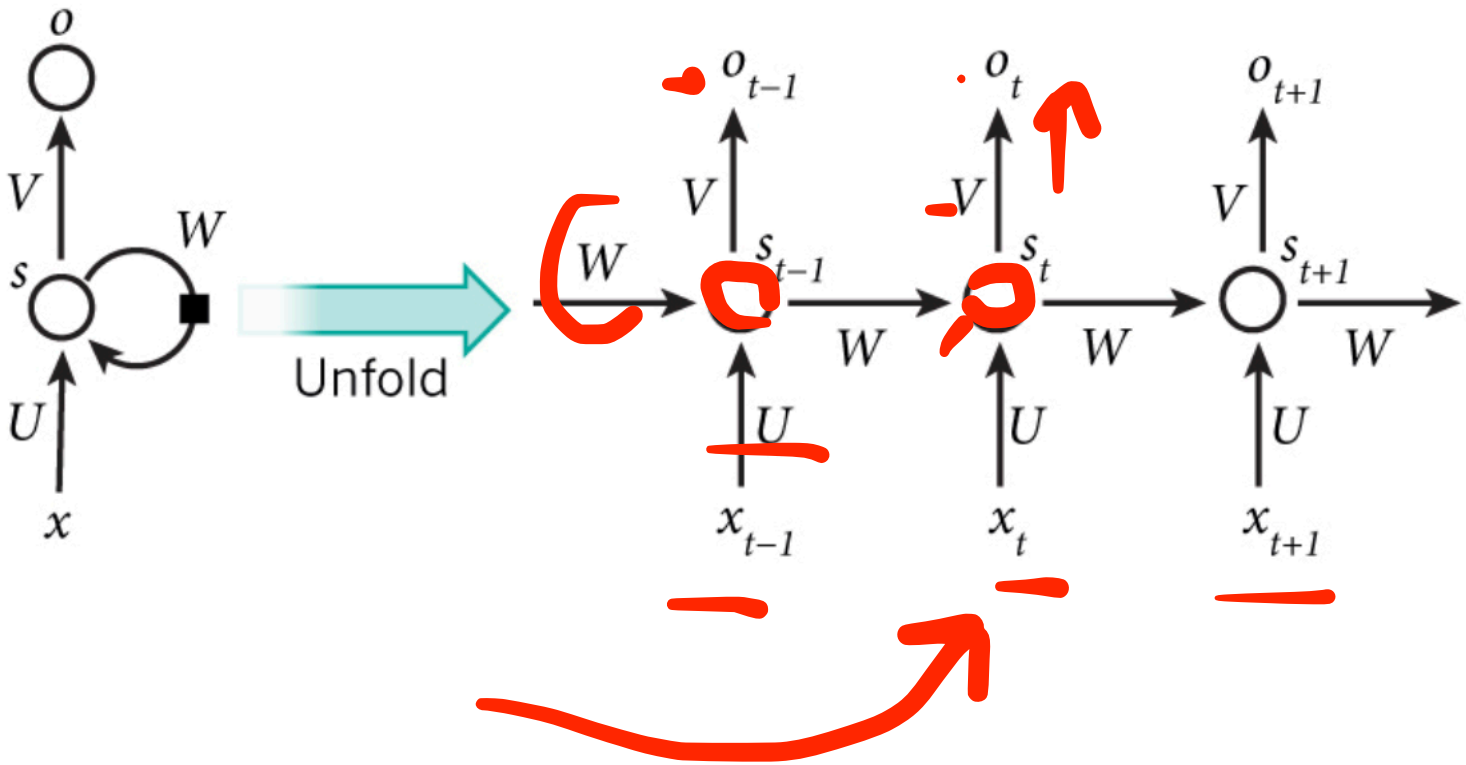
Outline

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

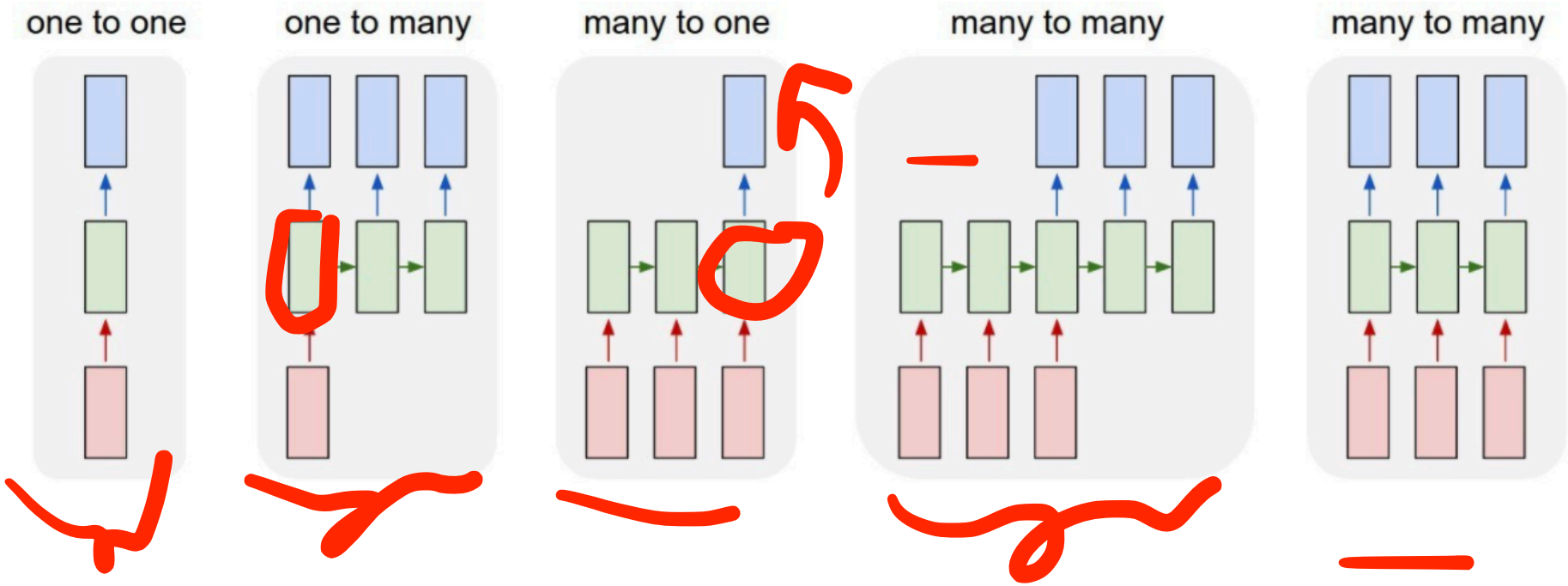
What is Recurrent Neural Network (RNN)?

- 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)?



RNN architectures



Applications of RNNs

1. One to Many

Image

RNN

A person is sitting in a beach holding guitar.

Sequence of text

2.Many to One

This movie is not good

Sequence of text

RNN

Negative

Sentiment

3.Many to Many

Main tenu pyar karda haan

Punjabi,India

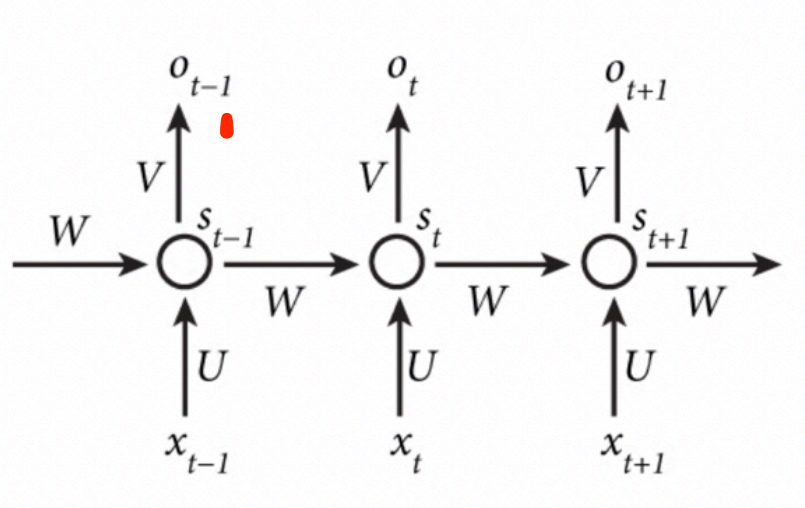
RNN

I love you

English, World

Forward Computation in RNNs

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



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

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

Backpropagation Through Time (BPTT)

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

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

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

$$= -\sum_t y_t \log \hat{y}_t$$

$$s_t = \tanh(U x_t + W s_{t-1})$$

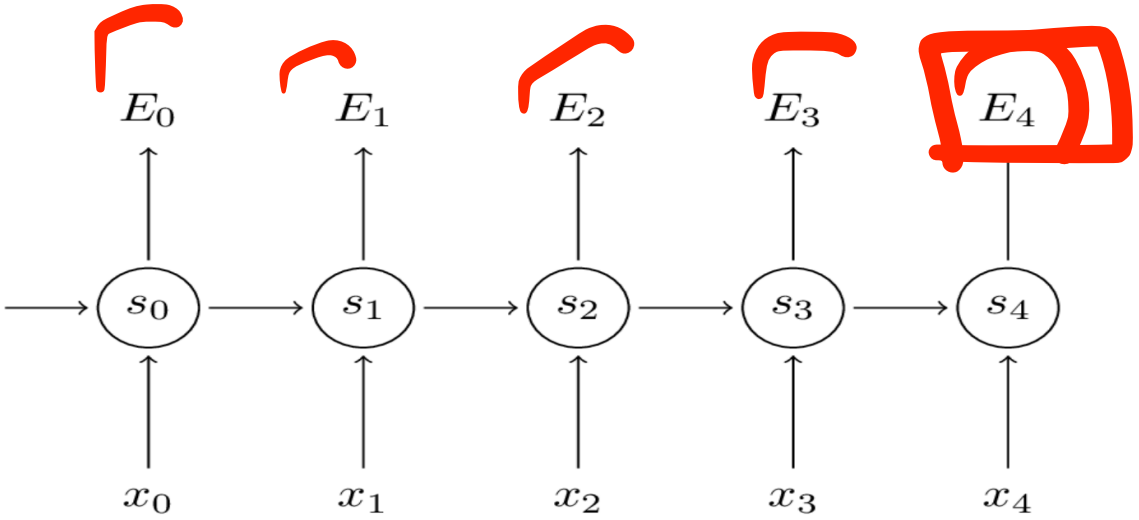
$$\hat{y}_t = \mathrm{softmax}(V s_t)$$

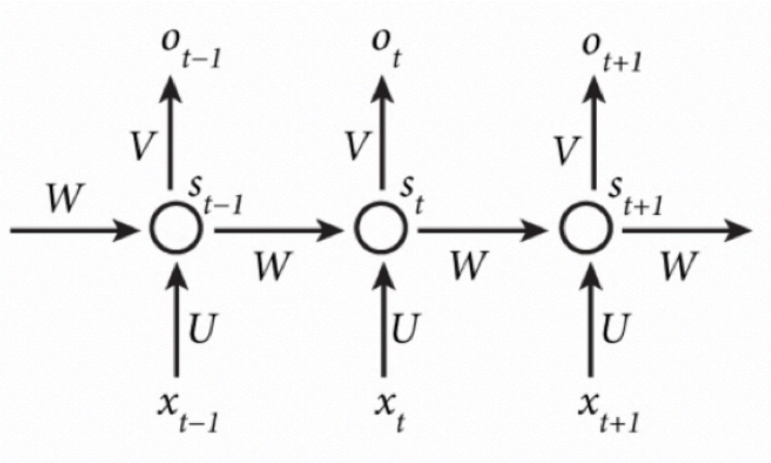
The cross entropy formula takes in two distributions, $p(x)$, the true distribution, and $q(x)$, the estimated distribution, defined over the discrete variable x and is given by

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

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: $\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$.

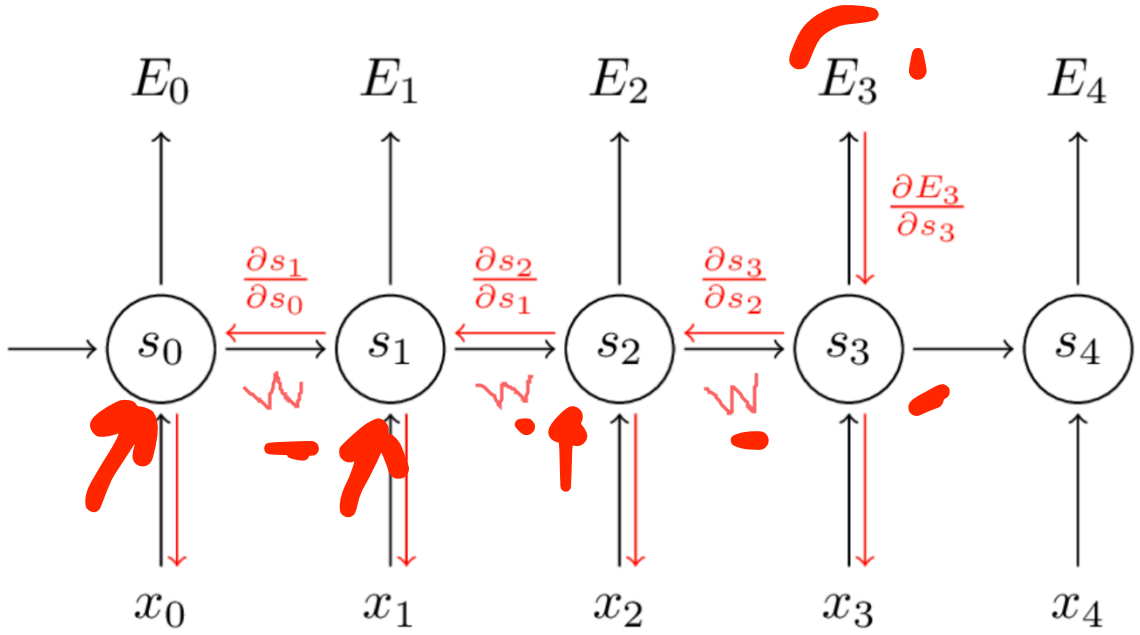




$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = \text{softmax}(Vs_t)$$

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

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



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

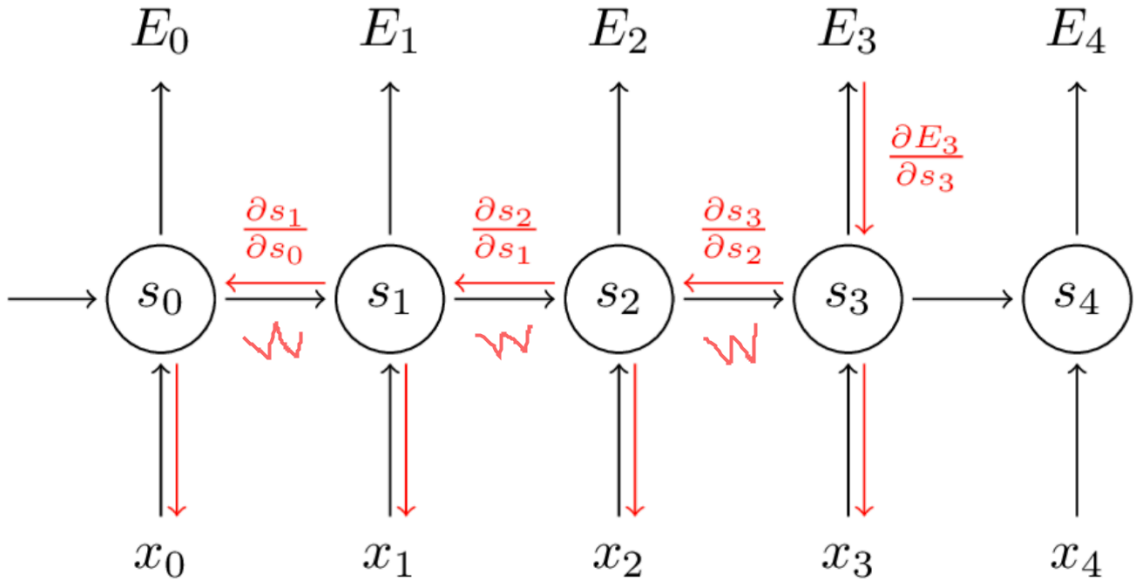
Note that $\frac{\partial s_3}{\partial s_k}$ is a chain rule in itself! For example,

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

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

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

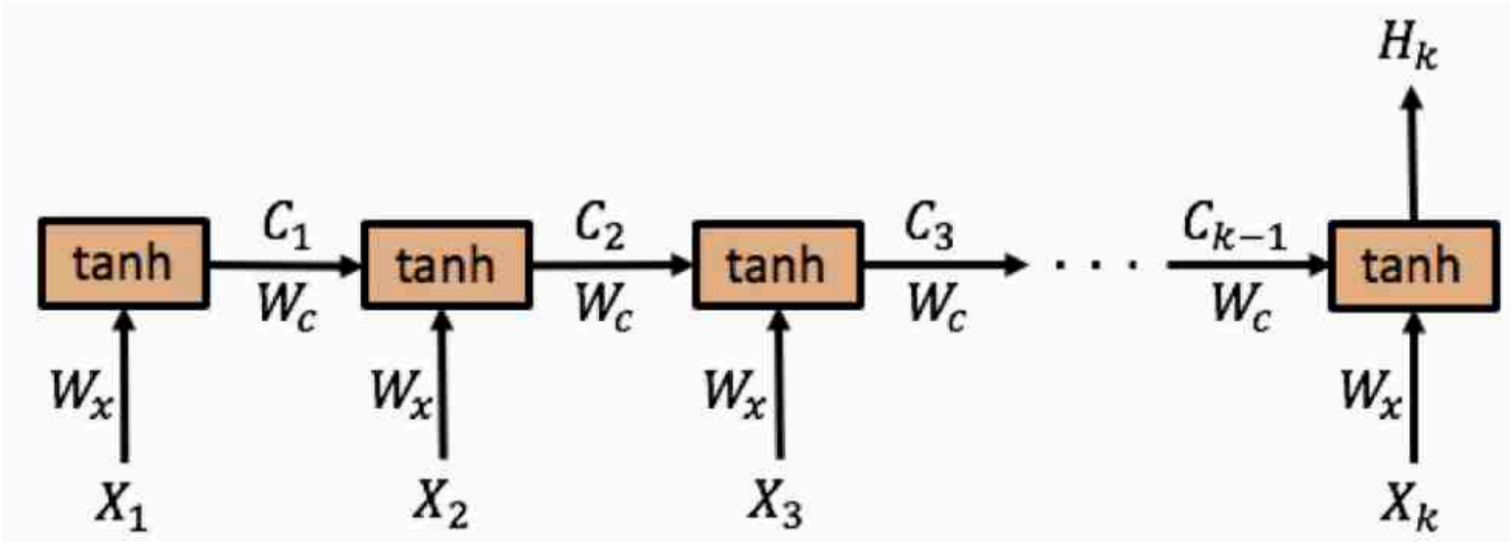
$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t)$$



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

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

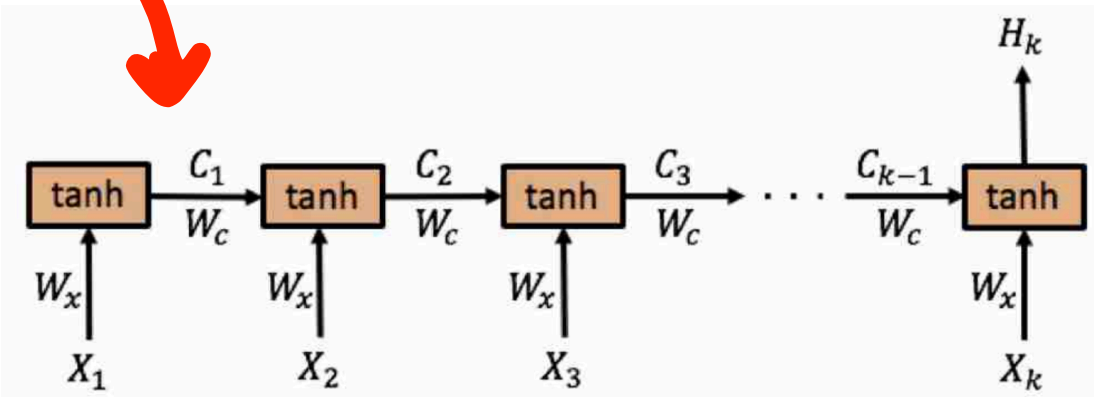
Vanishing Gradients



W = [Wc,Wx]

$$W \leftarrow W - \alpha \frac{\partial E_k}{\partial W}$$

Vanishing Gradients

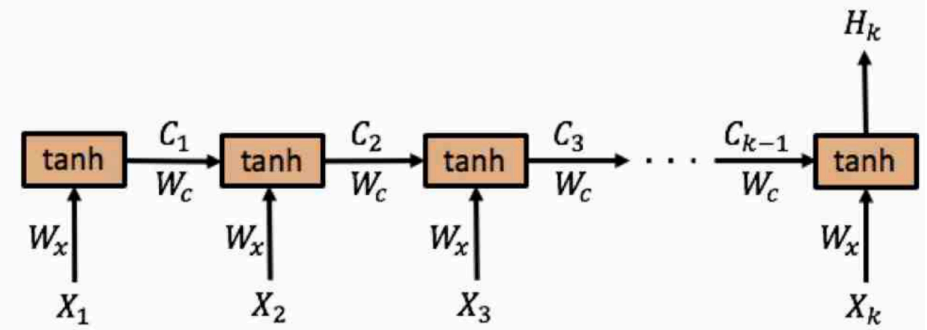


$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial H_k}\frac{\partial H_k}{\partial C_k}\frac{\partial C_k}{\partial C_{k-1}}\cdots\frac{\partial C_2}{\partial C_1}\frac{\partial C_1}{\partial W} =$$

$$\frac{\partial E_k}{\partial H_k}\frac{\partial H_k}{\partial C_k}\left(\prod_{t=2}^{k}\frac{\partial C_t}{\partial C_{t-1}}\right)\frac{\partial C_1}{\partial W}$$

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

Vanishing Gradients



$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial H_k}\frac{\partial H_k}{\partial C_k}\frac{\partial C_k}{\partial C_{k-1}}\cdots\frac{\partial C_2}{\partial C_1}\frac{\partial C_1}{\partial W} =$$

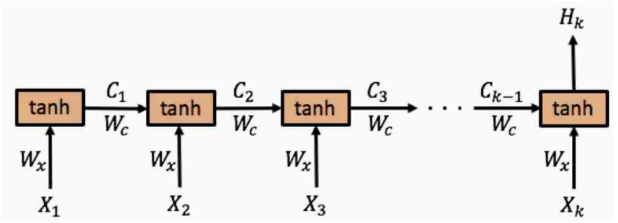
$$\frac{\partial E_k}{\partial H_k}\frac{\partial H_k}{\partial C_k}\left(\prod_{t=2}^{k}\frac{\partial C_t}{\partial C_{t-1}}\right)\frac{\partial C_1}{\partial W}$$

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

$$\frac{\partial C_t}{\partial C_{t-1}} = tanh'(W_c C_{t-1} + W_x X_t)\cdot\frac{d}{dC_{t-1}}[W_c C_{t-1} + W_x X_t] =$$

$$tanh'(W_c C_{t-1} + W_x X_t)\cdot W_c$$

Vanishing Gradients



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

$$\frac{\partial C_t}{\partial C_{t-1}} = tanh'(W_c C_{t-1} + W_x X_t) \cdot \frac{d}{dC_{t-1}}[W_c C_{t-1} + W_x X_t] =$$

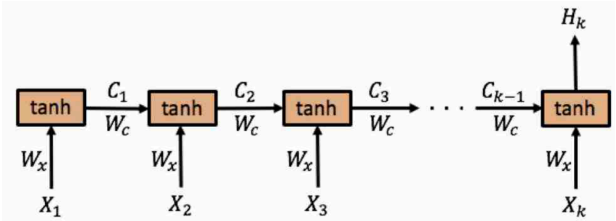
$$tanh'(W_c C_{t-1} + W_x X_t) \cdot W_c$$

$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial H_k}\frac{\partial H_k}{\partial C_k}\frac{\partial C_k}{\partial C_{k-1}} \cdots \frac{\partial C_2}{\partial C_1}\frac{\partial C_1}{\partial W} =$$

$$\frac{\partial E_k}{\partial H_k}\frac{\partial H_k}{\partial C_k}\left(\prod_{t=2}^{k}\frac{\partial C_t}{\partial C_{t-1}}\right)\frac{\partial C_1}{\partial W}$$

$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial H_k}\frac{\partial H_k}{\partial C_k}\left(\prod_{t=2}^{k} tanh'(W_c C_{t-1} + W_x X_t) \cdot W_c\right)\frac{\partial C_1}{\partial W}$$

Vanishing Gradients



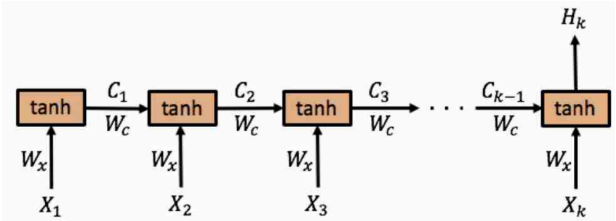
$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial H_k}\frac{\partial H_k}{\partial C_k}\left(\prod_{t=2}^{k} tanh'(W_c C_{t-1} + W_x X_t)\cdot W_c\right)\frac{\partial C_1}{\partial W}$$

The last expression **tends to vanish when k is large**, this is due to the derivative of the activation function tanh which is smaller or equal 1.

$$\prod_{t=2}^{k} tanh'(W_c C_{t-1} + W_x X_t)\cdot W_c \longrightarrow 0, \; so \; \frac{\partial E_k}{\partial W} \longrightarrow 0$$

$$W \leftarrow W - \alpha\frac{\partial E_k}{\partial W} \approx W$$

Vanishing Gradients



$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial H_k}\frac{\partial H_k}{\partial C_k}\left(\prod_{t=2}^{k} tanh'(W_c C_{t-1} + W_x X_t)\cdot W_c\right)\frac{\partial C_1}{\partial W}$$

The last expression **tends to vanish when k is large**, this is due to the derivative of the activation function tanh which is smaller or equal 1.

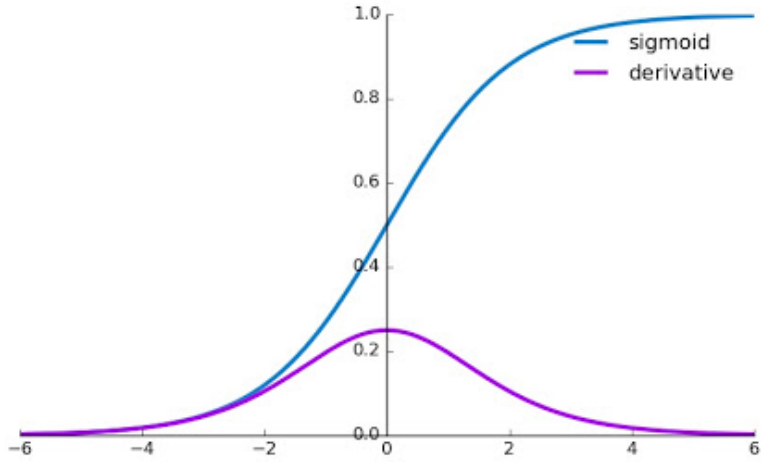
$$\prod_{t=2}^{k} tanh'(W_c C_{t-1} + W_x X_t)\cdot W_c \longrightarrow 0, \; so \; \frac{\partial E_k}{\partial W} \longrightarrow 0$$

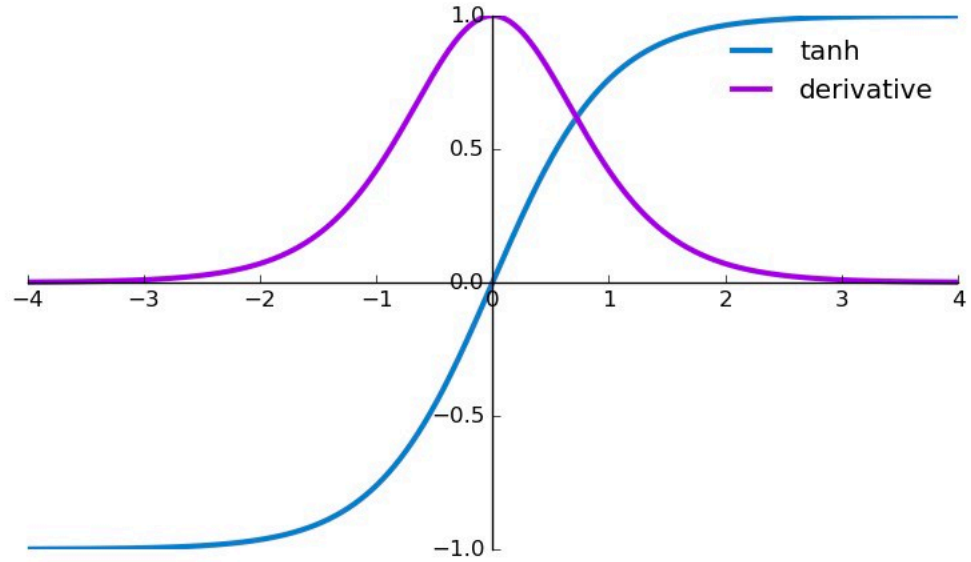
$$W \leftarrow W - \alpha\frac{\partial E_k}{\partial W} \approx W$$

W_C ~ small -> vanishing

W_C ~ large -> exploding

Sigmoid and Tanh funtions

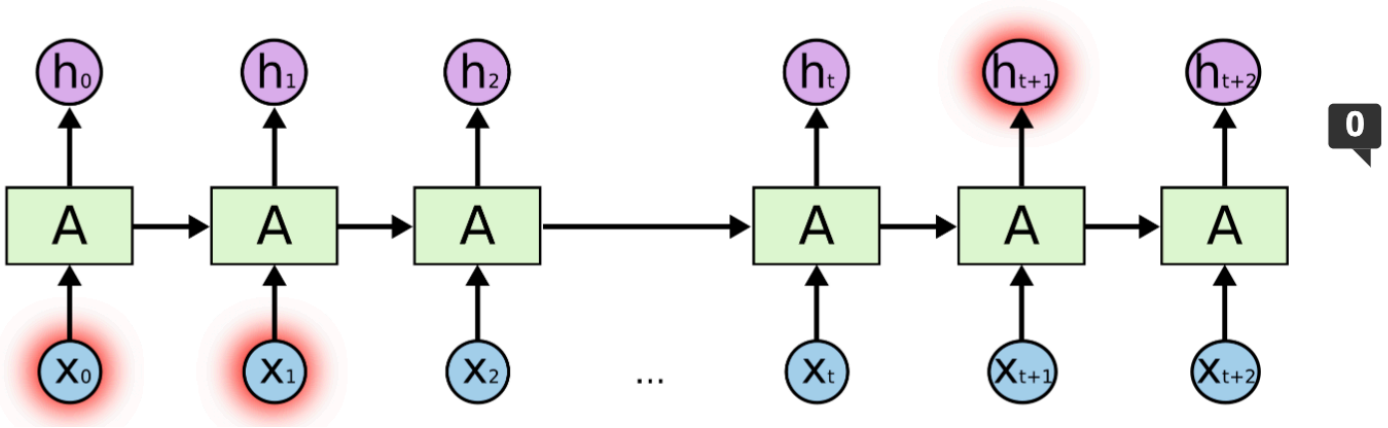




Limitation of RNN

But there are also cases where we need more context. Consider trying to predict the last word in the text "I grew up in France... I speak fluent *French*." Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It's entirely possible for the gap between the relevant information and the point where it is needed to become very large.

Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.



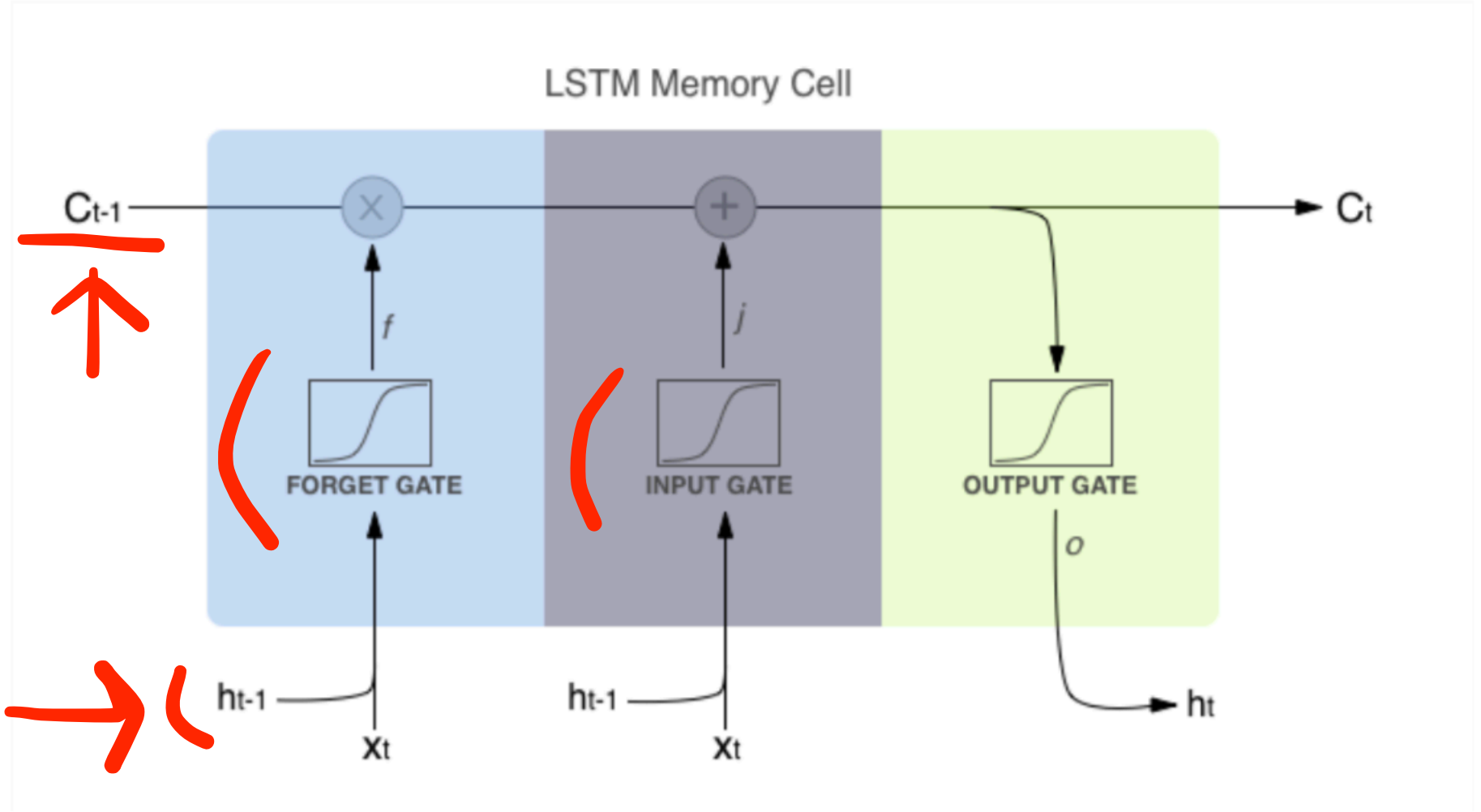
LSTM Networks

- 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.

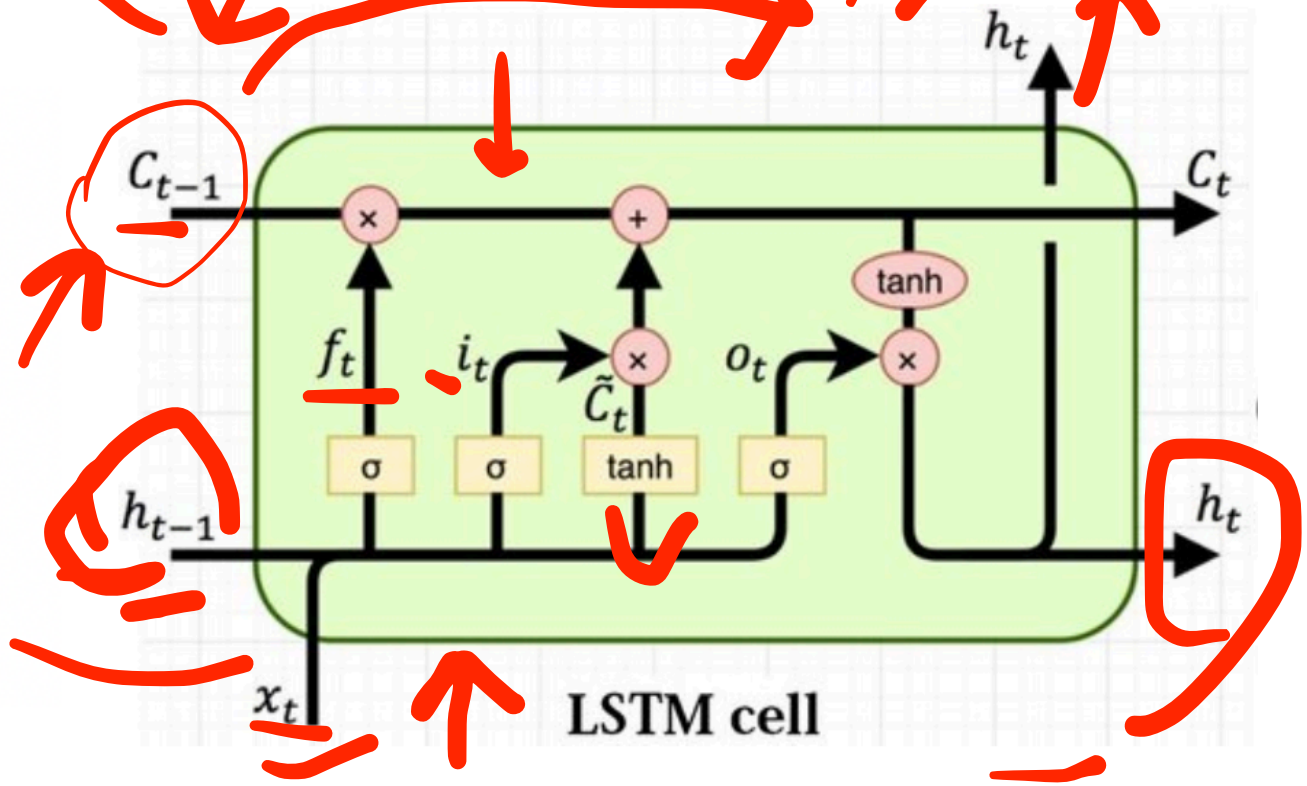
Longer Memories through LSTMs

- **Adding a forgetting mechanism.** If a scene ends, for example: "the model should forget the current scene location, the time of day, and reset any scene-specific information; however, if a character dies in the scene, it should continue remembering that he's no longer alive. Thus, we want the model to learn separate *forgetting/remembering* mechanism: when new inputs come in, it needs to know which beliefs to keep or throw away.
- **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.

LSTM Memory Cell



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



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

Variables [edit]

- $x_t \in \mathbb{R}^d$: input vector to the LSTM unit
- $f_t \in \mathbb{R}^h$: forget gate's activation vector
- $i_t \in \mathbb{R}^h$: input/update gate's activation vector
- $o_t \in \mathbb{R}^h$: output gate's activation vector
- $h_t \in \mathbb{R}^h$: hidden state vector also known as output vector of the LSTM unit
- $\tilde{c}_t \in \mathbb{R}^h$: cell input activation vector
- $c_t \in \mathbb{R}^h$: cell state vector
- $W \in \mathbb{R}^{h \times d}$, $U \in \mathbb{R}^{h \times h}$ and $b \in \mathbb{R}^h$: weight matrices and bias vector parameters which need to to be learned during training

where the superscripts d and h refer to the number of input features and number of hidden units, respectively.

Activation functions [edit]

- σ_g: sigmoid function.
- σ_c: hyperbolic tangent function.
- σ_h: hyperbolic tangent function or, as the peephole LSTM paper[29][30] suggests, $\sigma_h(x) = x$.

How does LSTM prevent vanishing

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

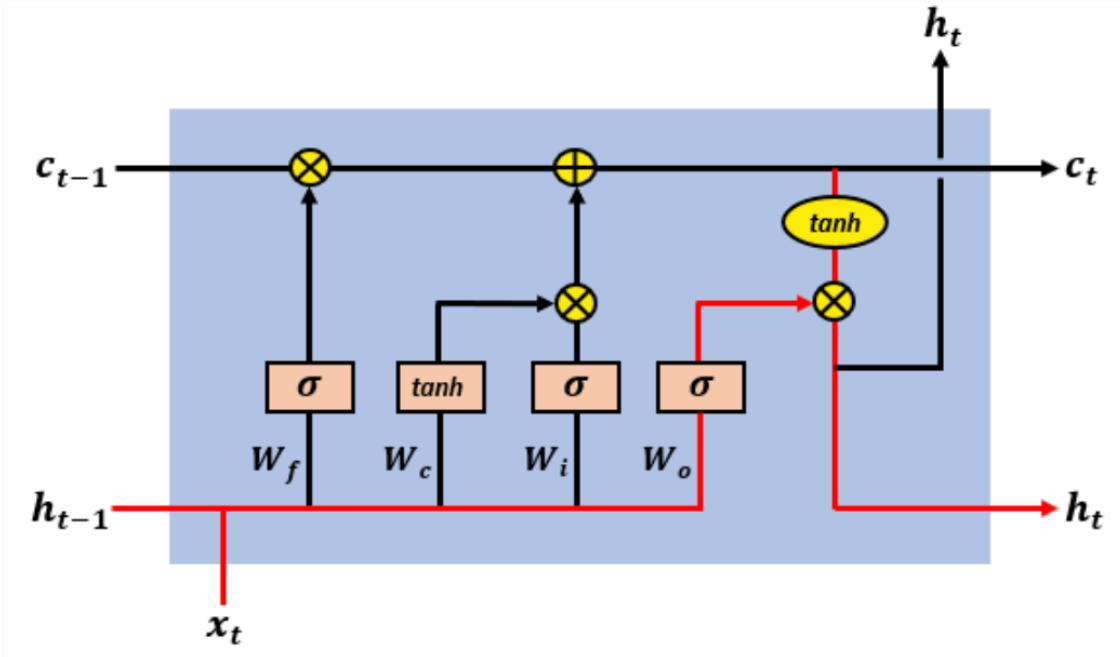
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t])$$

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

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

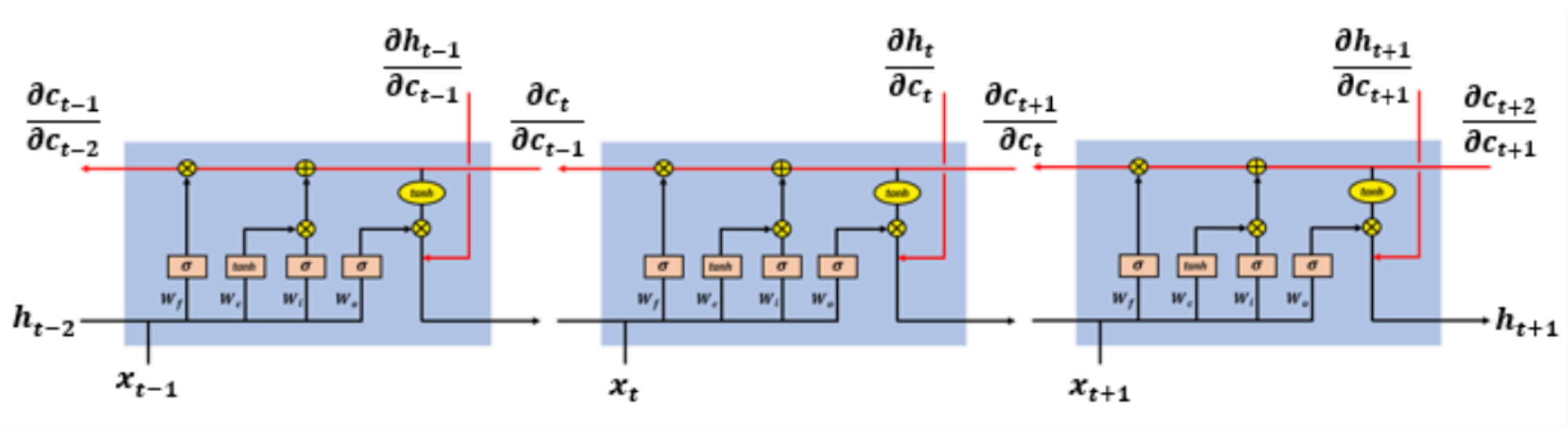
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t])$$

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



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

Backpropagation through time in LSTM



Backpropagating through time for gradient computation

https://medium.com/datadriveninvestor/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577

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

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

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

Compute the derivative of (5) and get:

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

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

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

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

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

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

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

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

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

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

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

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

$$\frac{\partial c_t}{\partial c_{t-1}} = \sigma'(W_f \cdot [h_{t-1}, x_t]) \cdot W_f \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot c_{t-1}$$

$$+ f_t$$

$$+ \sigma'(W_i \cdot [h_{t-1}, x_t]) \cdot W_i \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot \tilde{c}_t$$

$$+ \sigma'(W_c \cdot [h_{t-1}, x_t]) \cdot W_c \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot i_t$$

$$\frac{\partial c_t}{\partial c_{t-1}} = A_t + B_t + C_t + D_t$$

Backpropagation through time in LSTM

$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial h_k}\frac{\partial h_k}{\partial c_k}\left(\prod_{t=2}^{k}[A_t + B_t + C_t + D_t]\right)\frac{\partial c_1}{\partial W}$$

$$\frac{\partial E_{k+1}}{\partial W} \nrightarrow 0$$

How LSTM avoid Vanishing

The LSTM forgate gate update of the cell state Ct. Notice the forget gate's output which is given by

$$\sigma(W_f \cdot [H_t, X_{t+1}])$$

The LSTM input gate update of the cell state Ct. The input gate's output has the form:

$$tanh(W_c \cdot [H_t, X_{t+1}]) \otimes \sigma\,(W_i \cdot [H_t, X_{t+1}])$$

Backpropagation through time in LSTM

- One more explanation:

https://mc.ai/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients/

Backpropagating through time for gradient computation

$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial H_k}\frac{\partial H_k}{\partial C_k}\frac{\partial C_k}{\partial C_{k-1}}\cdots\frac{\partial C_2}{\partial C_1}\frac{\partial C_1}{\partial W} =$$

$$\frac{\partial E_k}{\partial H_k}\frac{\partial H_k}{\partial C_k}\left(\prod_{t=2}^{k}\frac{\partial C_t}{\partial C_{t-1}}\right)\frac{\partial C_1}{\partial W}$$

In an LTSM, the state vector Ct, has the form:

$$C_t = C_{t-1} \otimes \sigma\left(W_f \cdot [H_{t-1}, X_t]\right) \oplus$$

$$tanh\left(W_c \cdot [H_{t-1}, X_t]\right) \otimes \sigma\left(W_i \cdot [H_{t-1}, X_t]\right)$$

Backpropagating through time for gradient computation

vanilla RNN

In an LTSM, the state vector Ct, has the form:

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

$$C_t = C_{t-1} \otimes \sigma\,(W_f \cdot [H_{t-1}, X_t]) \oplus$$

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

$$\frac{\partial C_t}{\partial C_{t-1}} = \sigma(W_f \cdot [H_{t-1}, X_t]) + \frac{d}{dC_{t-1}}(tanh(W_c \cdot [H_{t-1}, X_t]) \otimes \sigma(W_i \cdot [H_{t-1}, X_t]))$$

How LSTM prevent vanishing problem

vanilla RNN

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

LSTM

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

Backpropagating through time for gradient computation

$$C_t = C_{t-1} \otimes \sigma\,(W_f \cdot [H_{t-1}, X_t]) \oplus$$

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

$$\frac{\partial C_t}{\partial C_{t-1}} = \sigma(W_f \cdot [H_{t-1}, X_t]) + \frac{d}{dC_{t-1}}(tanh(W_c \cdot [H_{t-1}, X_t]) \otimes \sigma(W_i \cdot [H_{t-1}, X_t]))$$

For simplicity, we leave out the computation of:

$$\frac{d}{dC_{t-1}}(tanh(W_c \cdot [H_{t-1}, X_t]) \otimes \sigma(W_i \cdot [H_{t-1}, X_t]))$$

So we just write:

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

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:

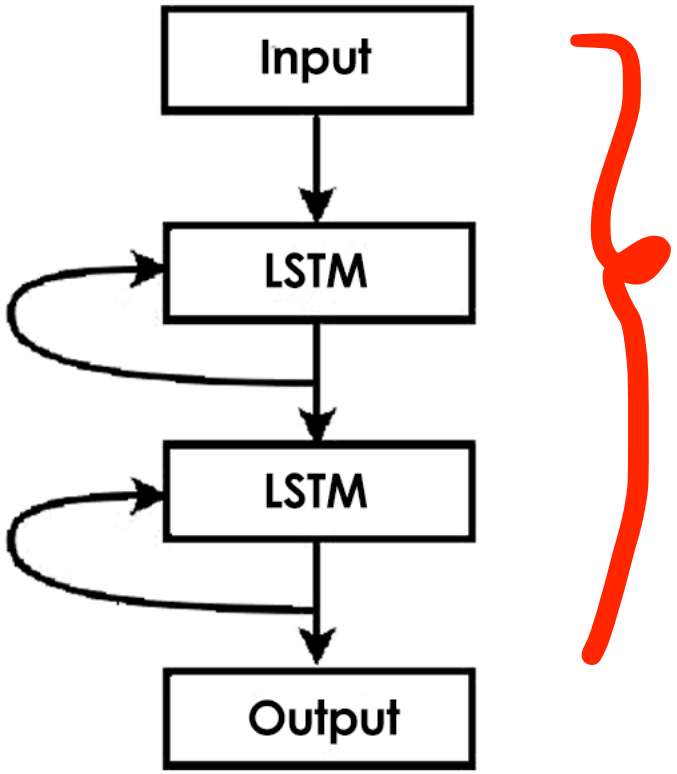
$$\frac{\partial C_t}{\partial C_{t-1}} \nrightarrow 0$$

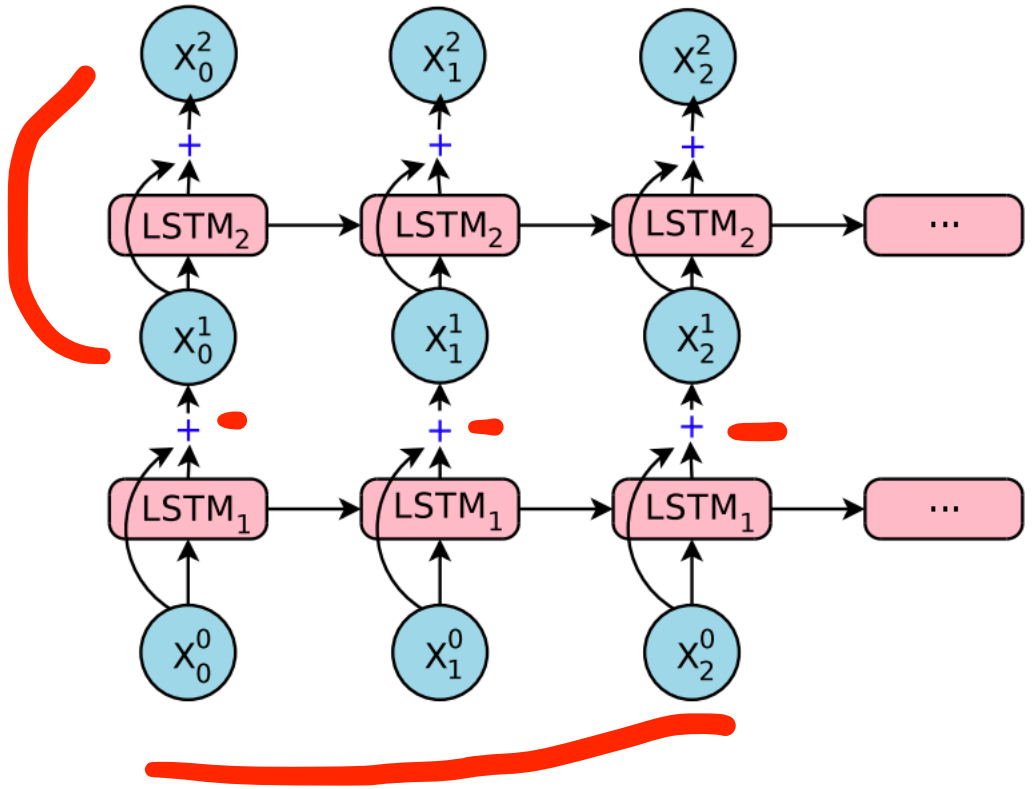
Finally:

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

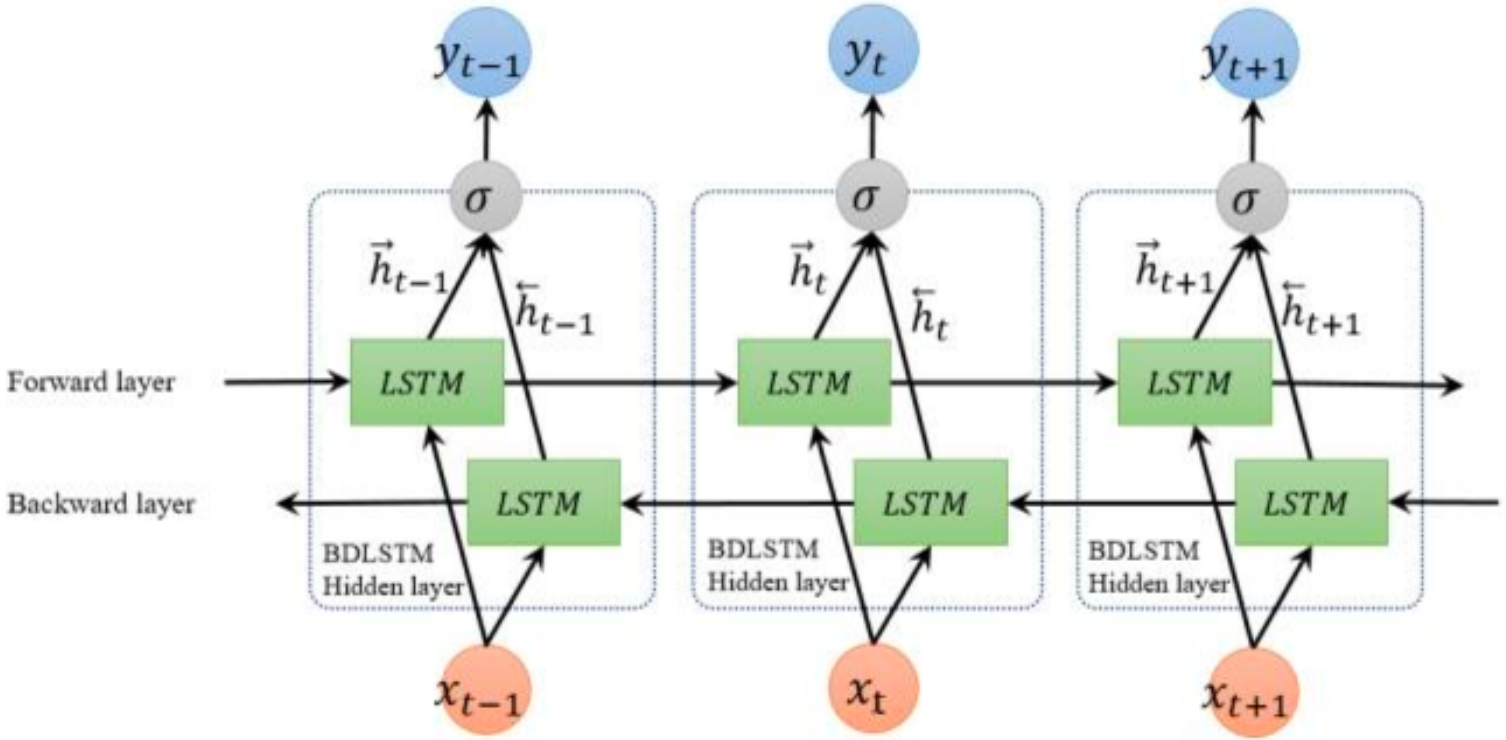
And the gradients do not vanish!

Stacked LSTM

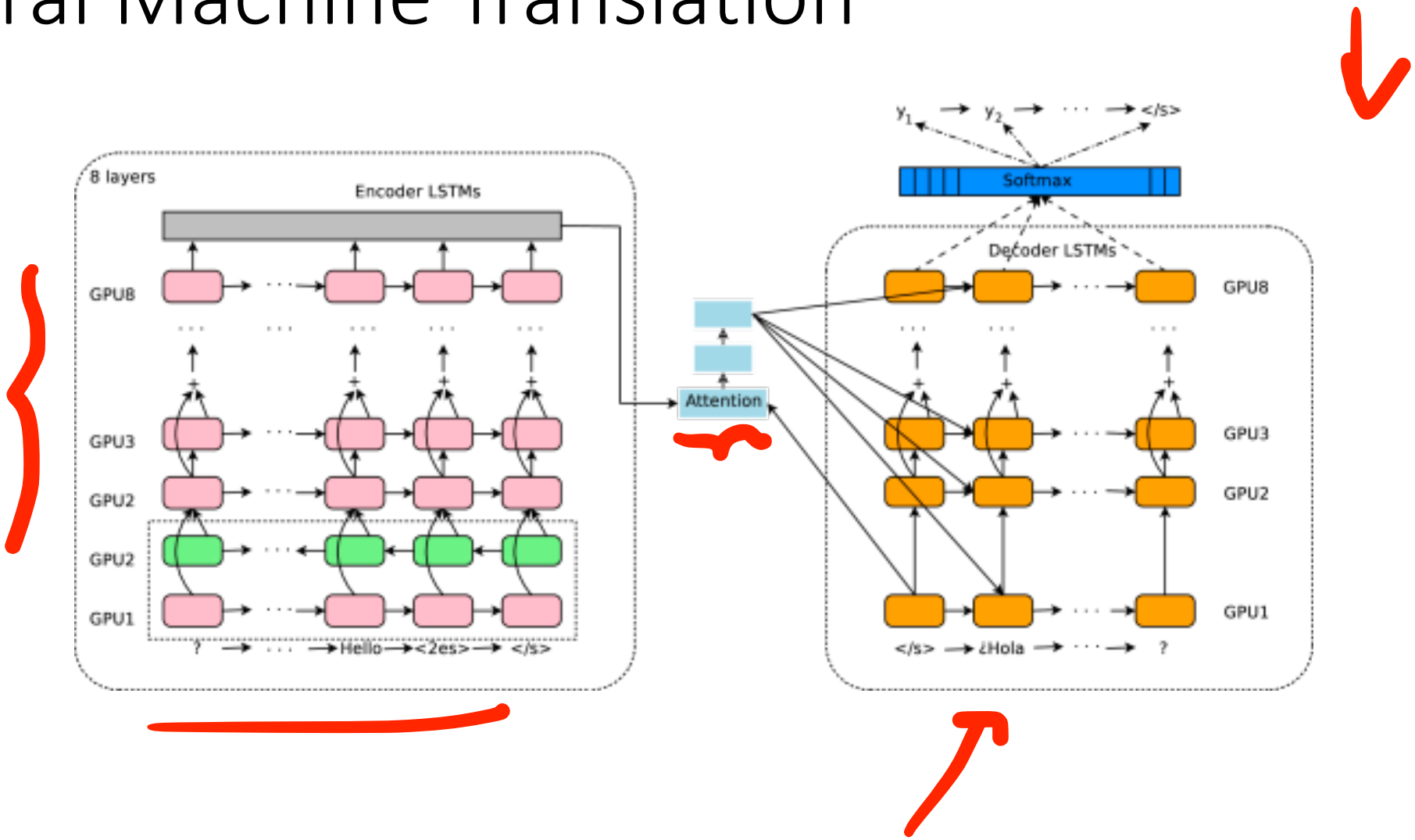




BiLSTM


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



Thank you!

- Comment & questions?