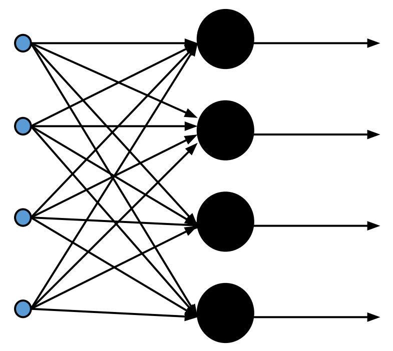
AI Course

Dr. Mürsel Taşgın

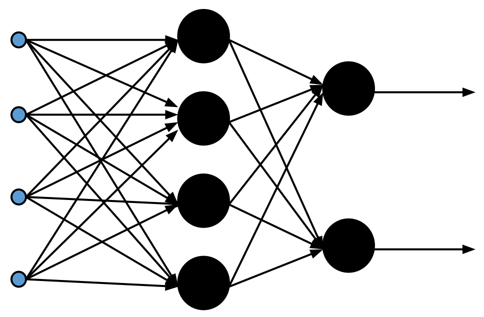
Neural Networks, Deep Learning

Neural Networks

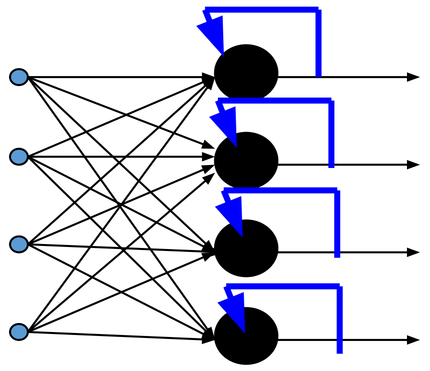
Types of neural networks



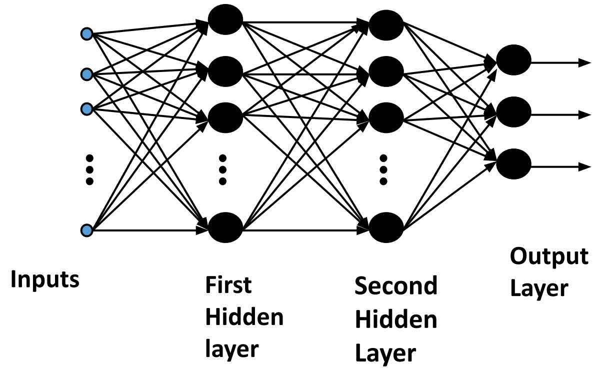
Multiple Inputs and Single Layer



Multiple Inputs and layers



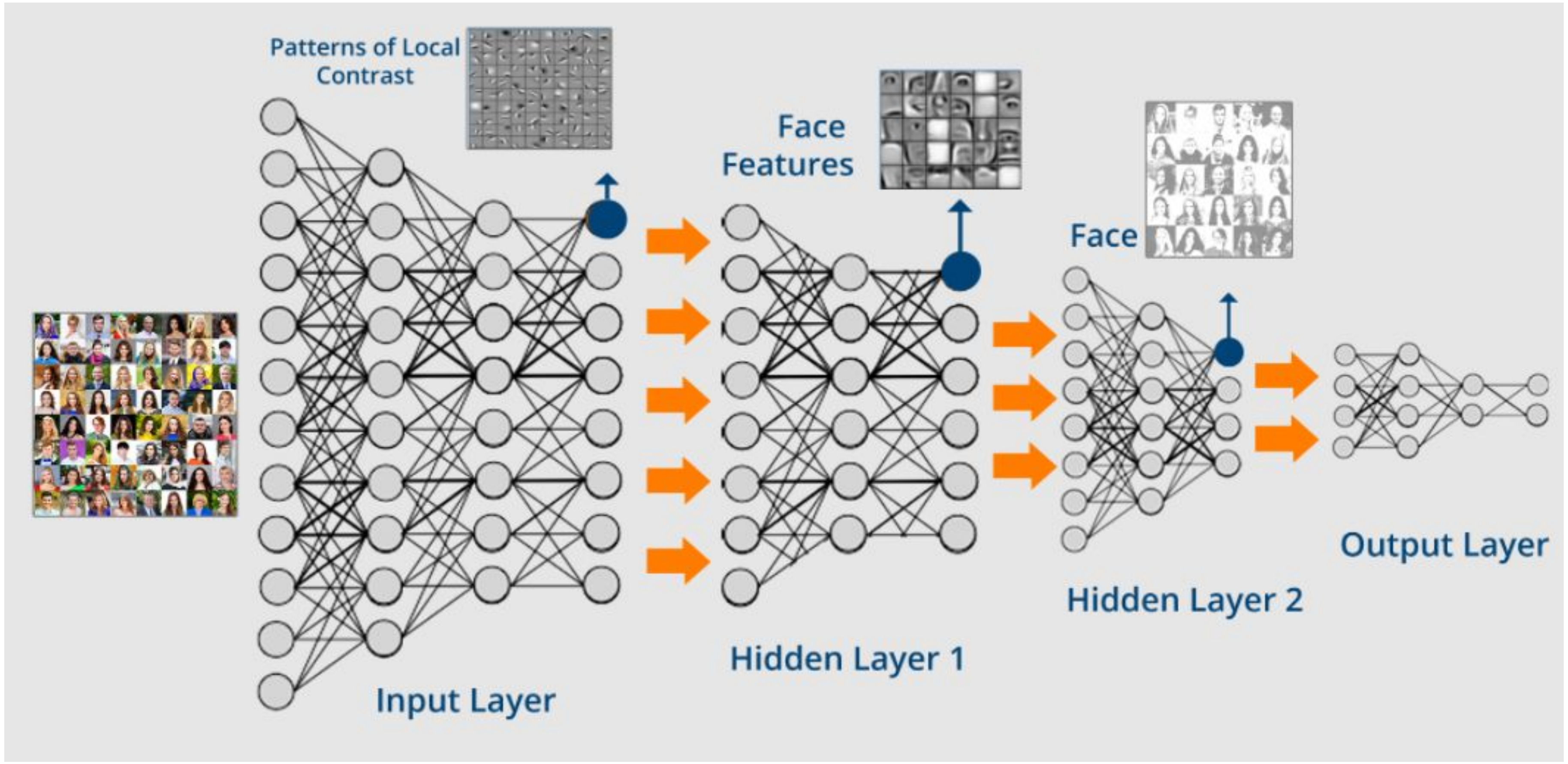
Recurrent Neural Networks



Multilayer Networks

Neural Networks

Deep Neural Networks



Many hidden layers!

Neural Networks

Deep Neural Networks

Why "*deep*"?

- Deep learning is a class of machine learning algorithms that uses multiple layers to progressively extract higher-level features from the raw input.

- The word "deep" in "deep learning" refers to the number of layers through which the data is transformed. More precisely, deep learning systems have a substantial *credit assignment path* (CAP) depth. The CAP is the chain of transformations from input to output. CAPs describe potentially causal connections between input and output.

"Neural Networks are universal approximators !"

The Universal Approximation Theorem tells us that Neural Networks has a kind of ***universality*** i.e. no matter what f(x) is, there is a network that can approximately approach the result and do the job! This result holds for any number of inputs and outputs.

Neural Networks

Deep Neural Networks

Backpropagation

- The backpropagation algorithm was originally introduced in the 1970s, but its importance wasn't fully appreciated until a famous 1986 paper by David Rumelhart, Geoffrey Hinton, and Ronald Williams.

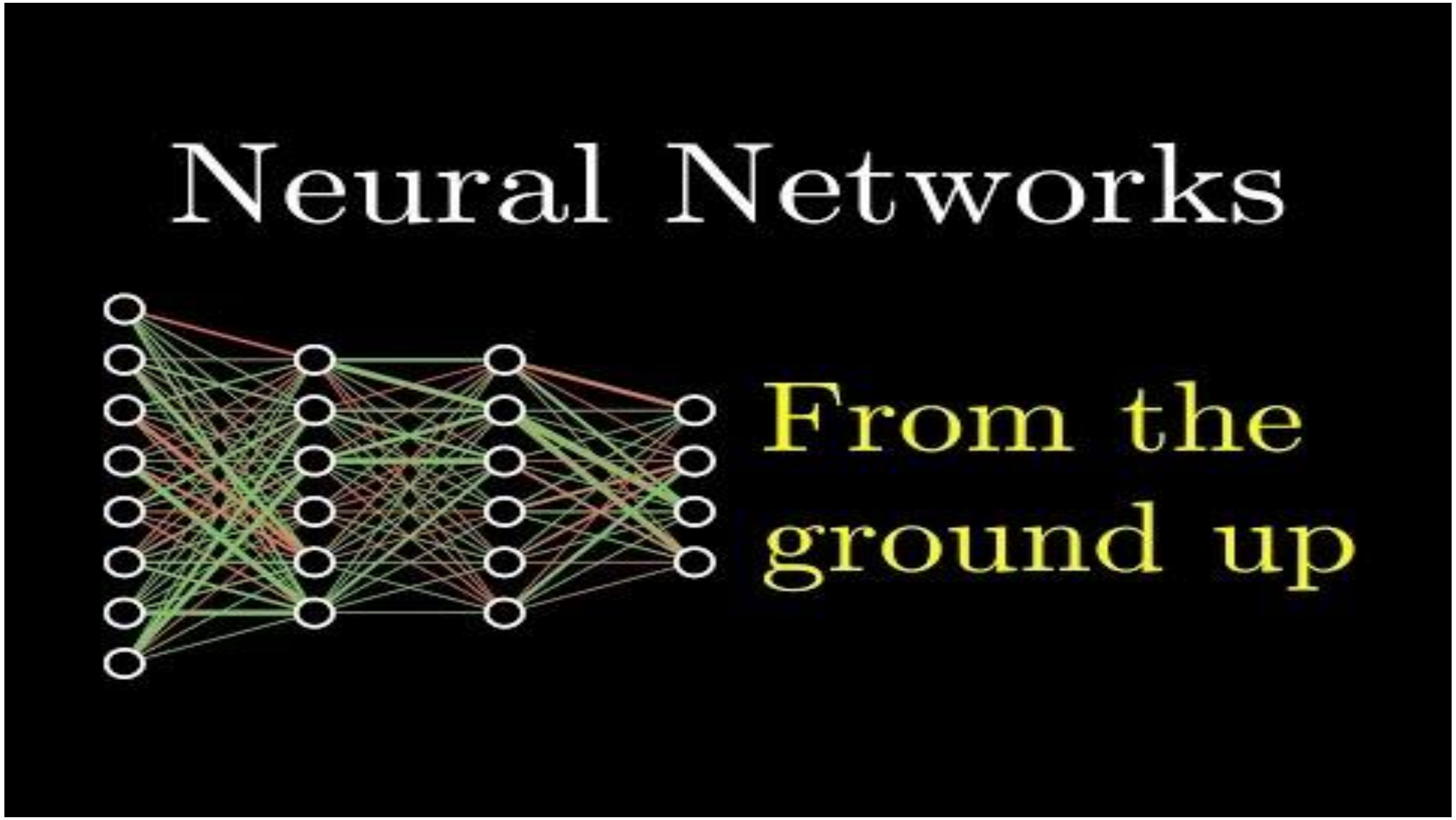
- That paper describes several neural networks where backpropagation works far faster than earlier approaches to learning, making it possible to use neural nets to solve problems which had previously been insoluble. Today, the backpropagation algorithm is the workhorse of learning in neural networks.

- In 2012, AlexNet gained much popularity with its success on ImageNet (*More data, more hardware power, better algorithms*) .

left: Alex Krizhevsky, Middle: Ilya Sutskever, Right: Geoffery Hinton

Neural Networks

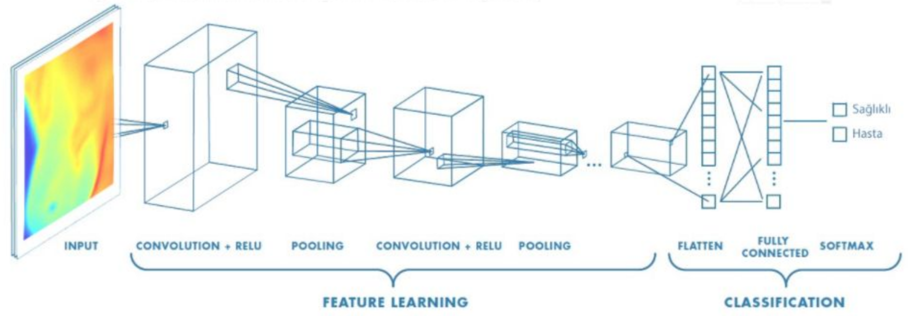
Deep Neural Networks



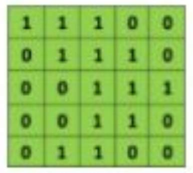
https://www.youtube.com/watch?v=aircAruvnKk

Neural Networks

Convolutional Neural Networks

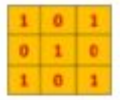


Convolutional Neural Networks



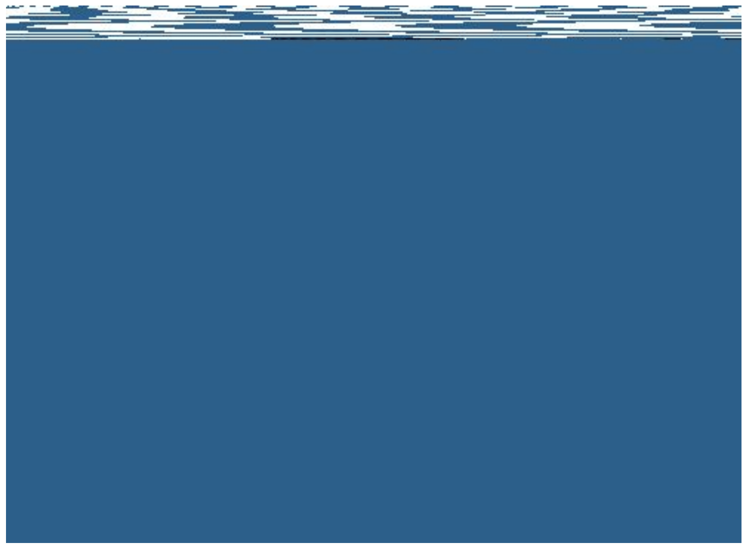
1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

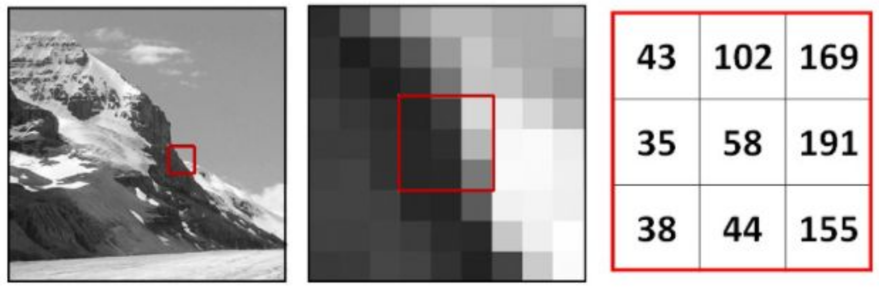
Image



1	0	1
0	1	0
1	0	1

Filter

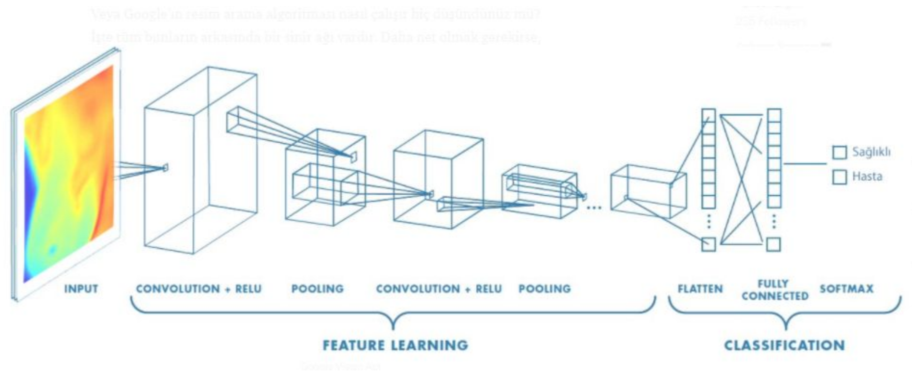




43	102	169
35	58	191
38	44	155

Neural Networks

Convolutional Neural Networks



Convolutional Neural Networks

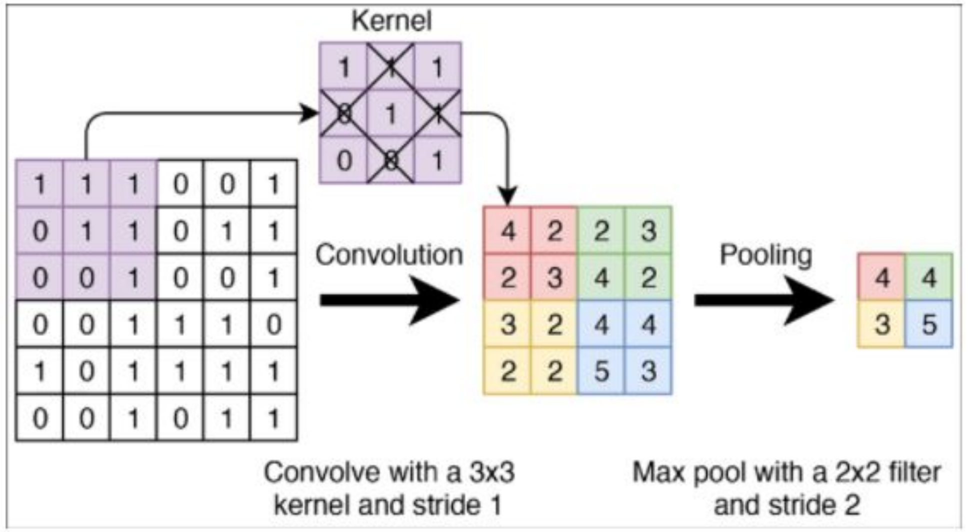
Convolution

Convolution applies a *filter* on the original image. Depending on the filter, we get another image and make some of the features more *visible/identified*.

Pooling

After convolution, we can group the pixels and perform an aggregation over them

- *Max pooling*
- *Average pooline*



Kernel
Convolution
Pooling
Convolve with a 3x3 kernel and stride 1
Max pool with a 2x2 filter and stride 2

1	1	1	0	0	1
0	1	1	0	1	1
0	0	1	0	0	1
0	0	1	1	1	0
1	0	1	1	1	1
0	0	1	0	1	1

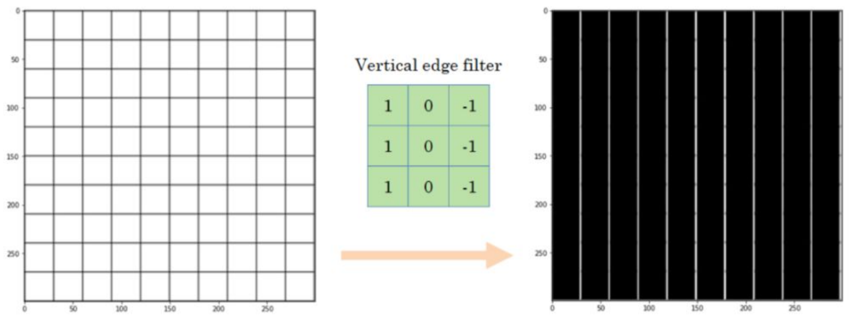
1	$\cancel{1}$	1
$\cancel{0}$	1	$\cancel{1}$
0	$\cancel{0}$	1

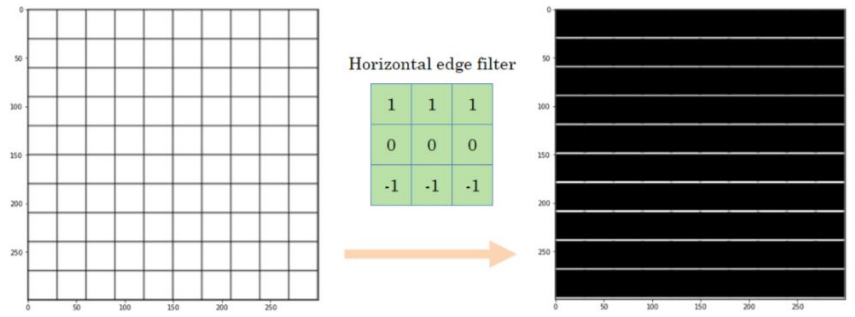
4	2	2	3
2	3	4	2
3	2	4	4
2	2	5	3

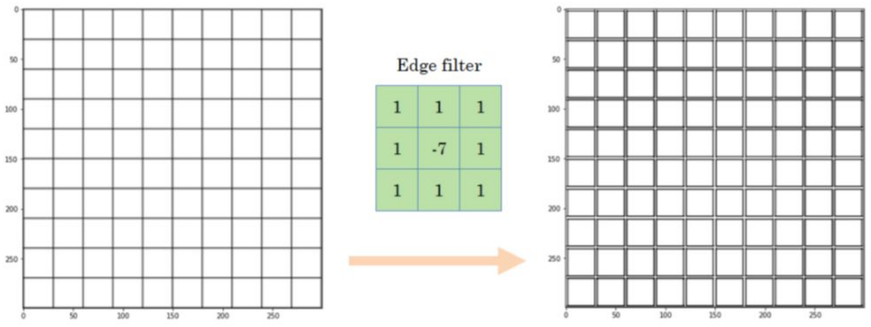
4	4
3	5

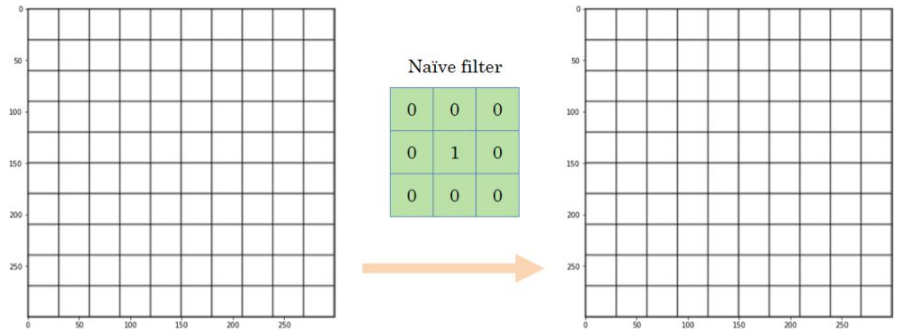
Neural Networks

Convolutional Neural Networks



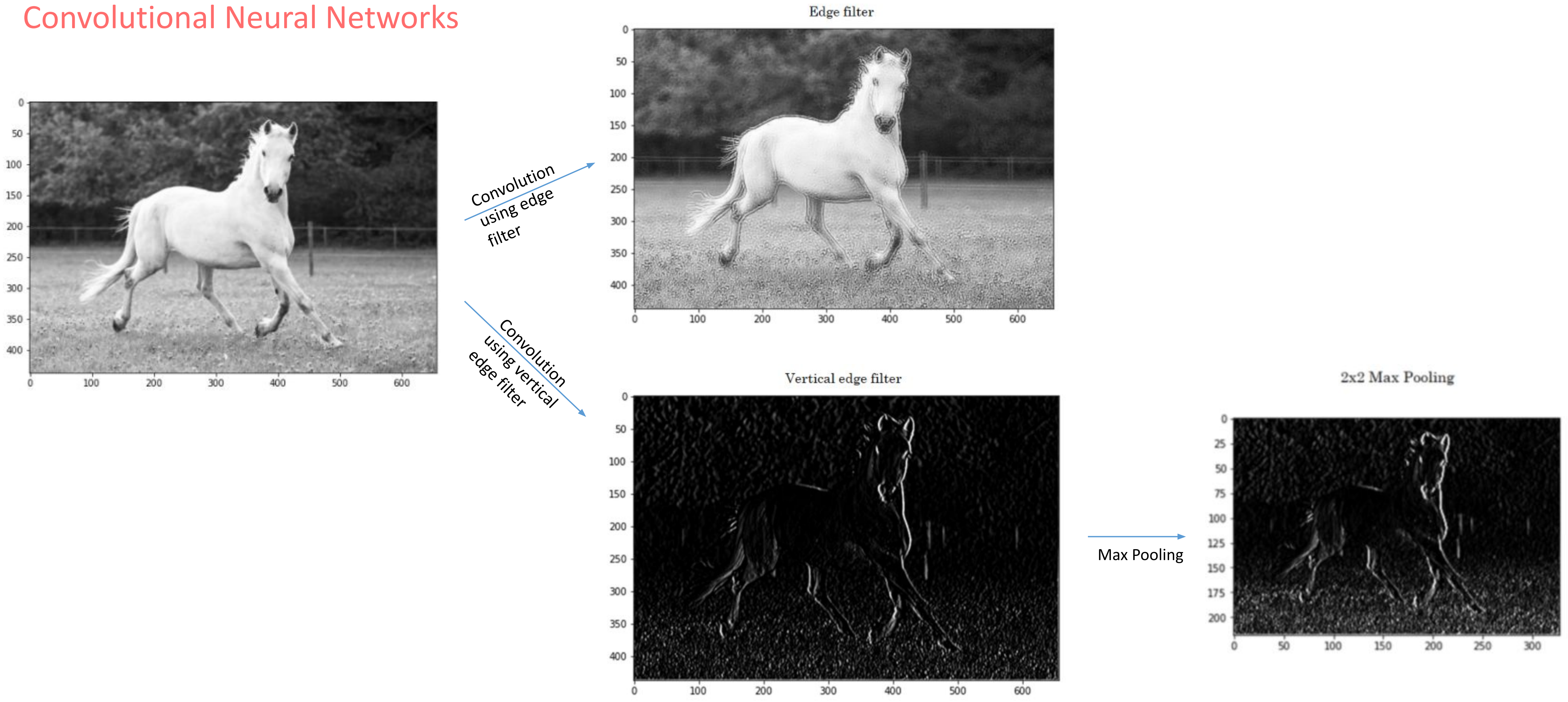






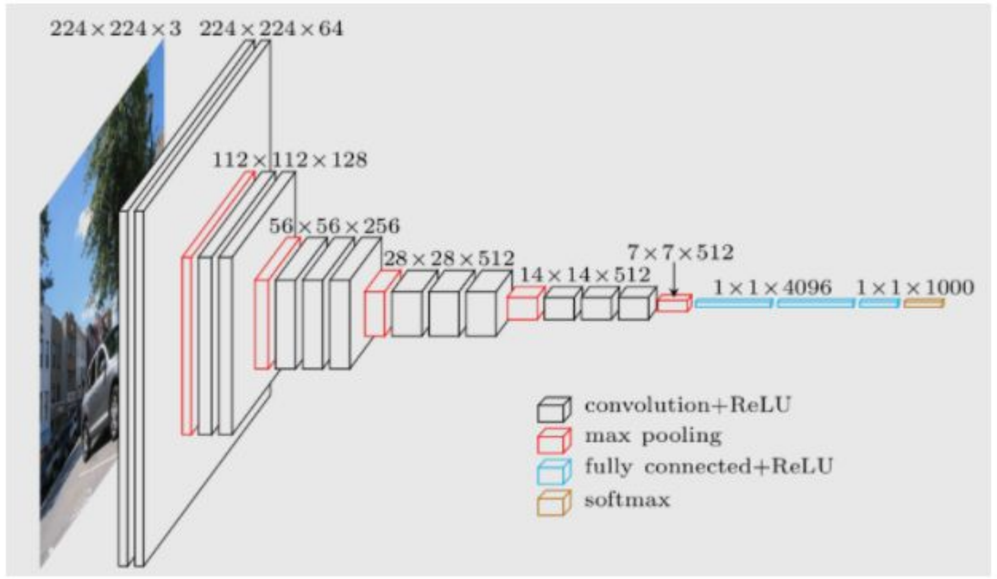
Neural Networks

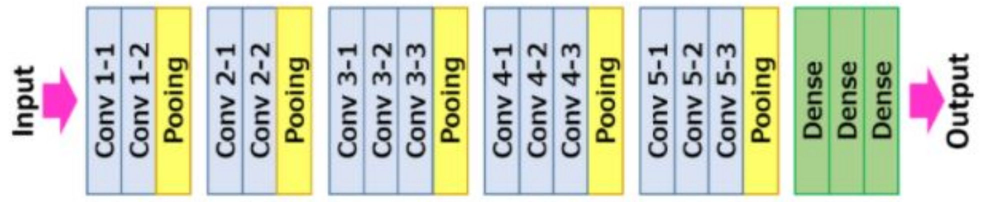
Convolutional Neural Networks

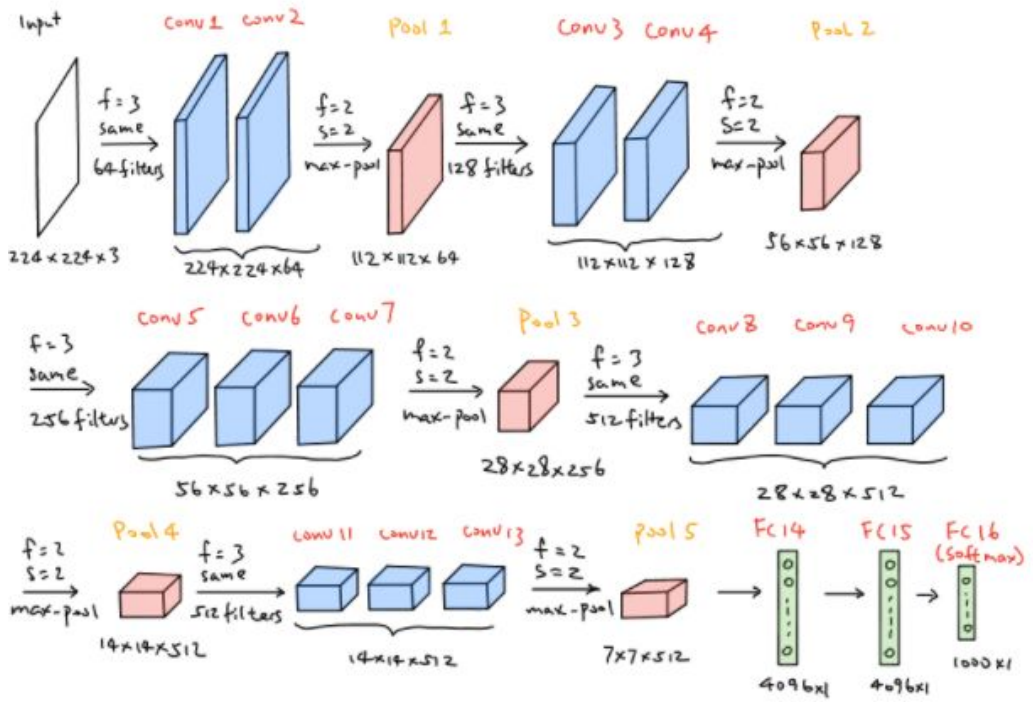

Convolution using edge filter
Edge filter
Convolution using vertical edge filter
Vertical edge filter
Max Pooling
2x2 Max Pooling

Neural Networks

Convolutional Neural Networks





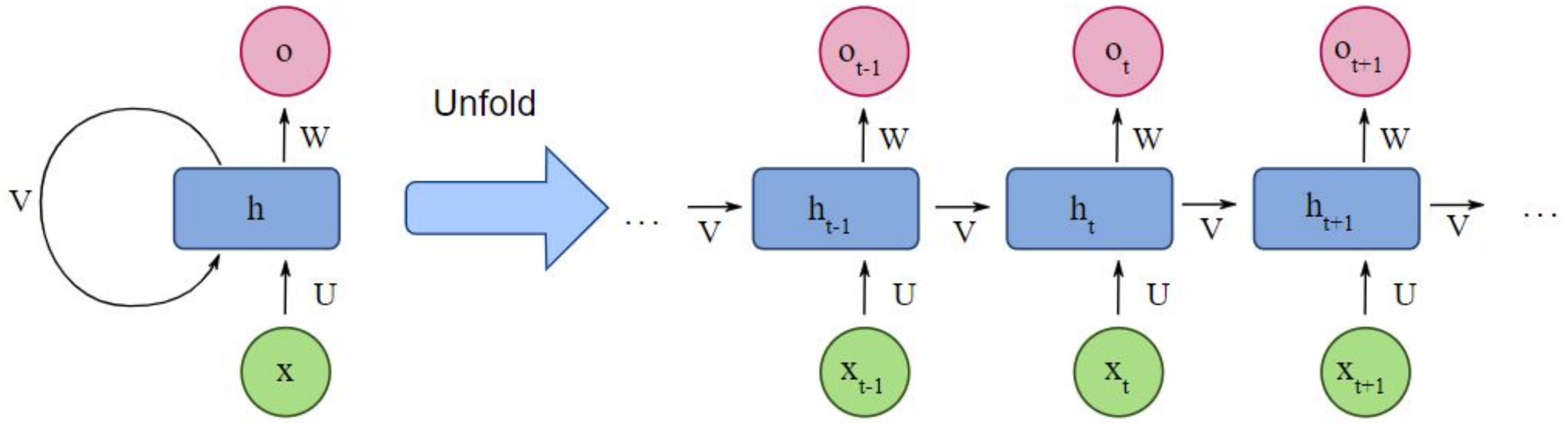


https://guandi1995.github.io/Classical-CNN-architecture/

Neural Networks

Recurrent Neural Networks

- A **recurrent neural network (RNN)** is a class of artificial neural networks where connections between nodes form a directed or undirected graph along a temporal sequence
- This allows it to exhibit temporal dynamic behavior
- RNNs can use their internal state (memory) to process variable length sequences of inputs

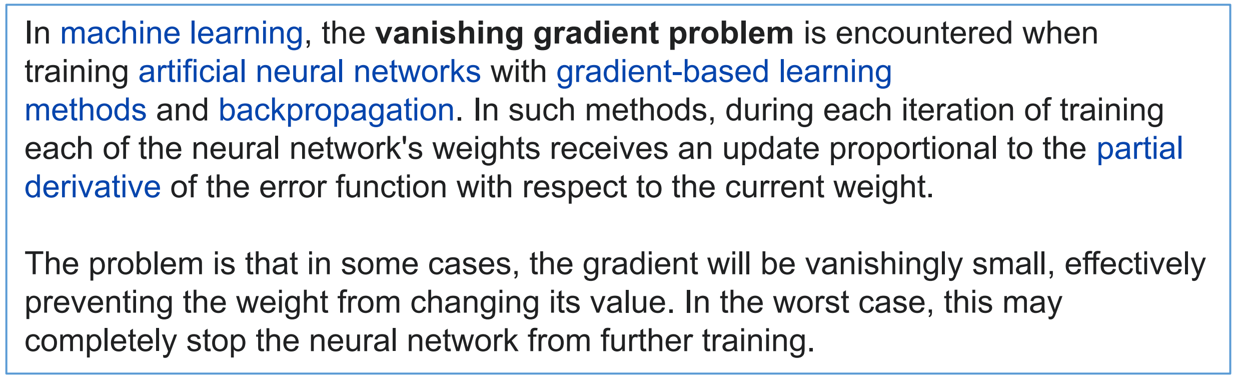


Source: Wikipedia

Neural Networks

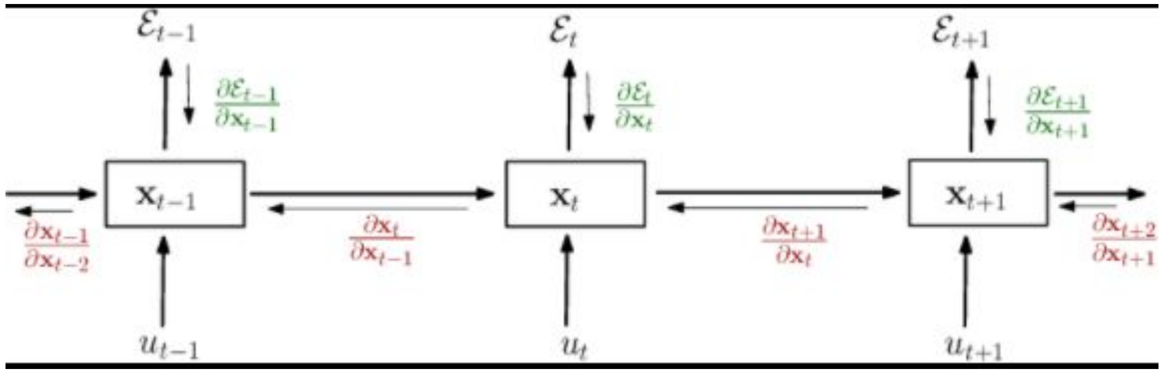
Recurrent Neural Networks

- RNNs can learn short sequences, but can be limited for longer periods.
- RNNs can have vanishing *(exploding)* gradient problems.



In machine learning, the **vanishing gradient problem** is encountered when training artificial neural networks with gradient-based learning methods and backpropagation. In such methods, during each iteration of training each of the neural network's weights receives an update proportional to the partial derivative of the error function with respect to the current weight.

The problem is that in some cases, the gradient will be vanishingly small, effectively preventing the weight from changing its value. In the worst case, this may completely stop the neural network from further training.

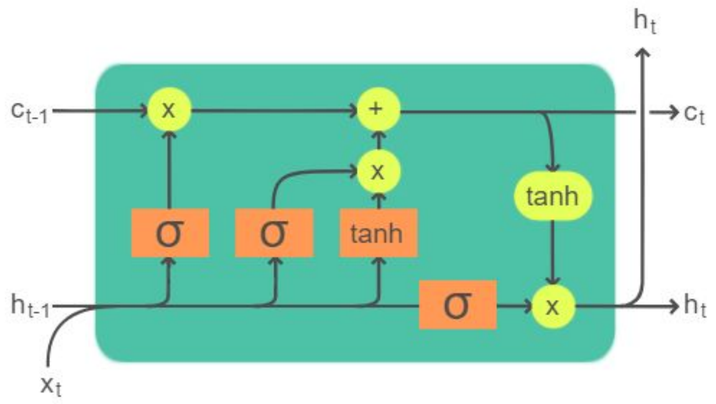


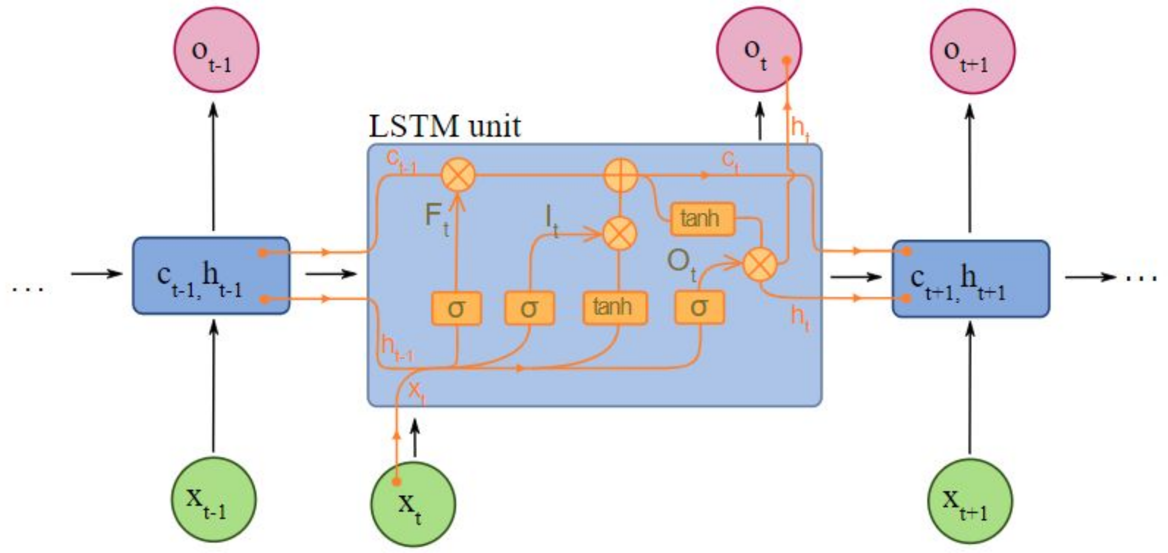
$\mathcal{E}_{t-1}$
$\frac{\partial \mathcal{E}_{t-1}}{\partial \mathbf{x}_{t-1}}$
$\mathcal{E}_t$
$\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t}$
$\mathcal{E}_{t+1}$
$\frac{\partial \mathcal{E}_{t+1}}{\partial \mathbf{x}_{t+1}}$
$\mathbf{x}_{t-1}$
$\mathbf{x}_t$
$\mathbf{x}_{t+1}$
$\frac{\partial \mathbf{x}_{t-1}}{\partial \mathbf{x}_{t-2}}$
$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_{t-1}}$
$\frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{x}_t}$
$\frac{\partial \mathbf{x}_{t+2}}{\partial \mathbf{x}_{t+1}}$
u_{t-1}
u_t
u_{t+1}

Neural Networks

LSTM (Long short-term memory)

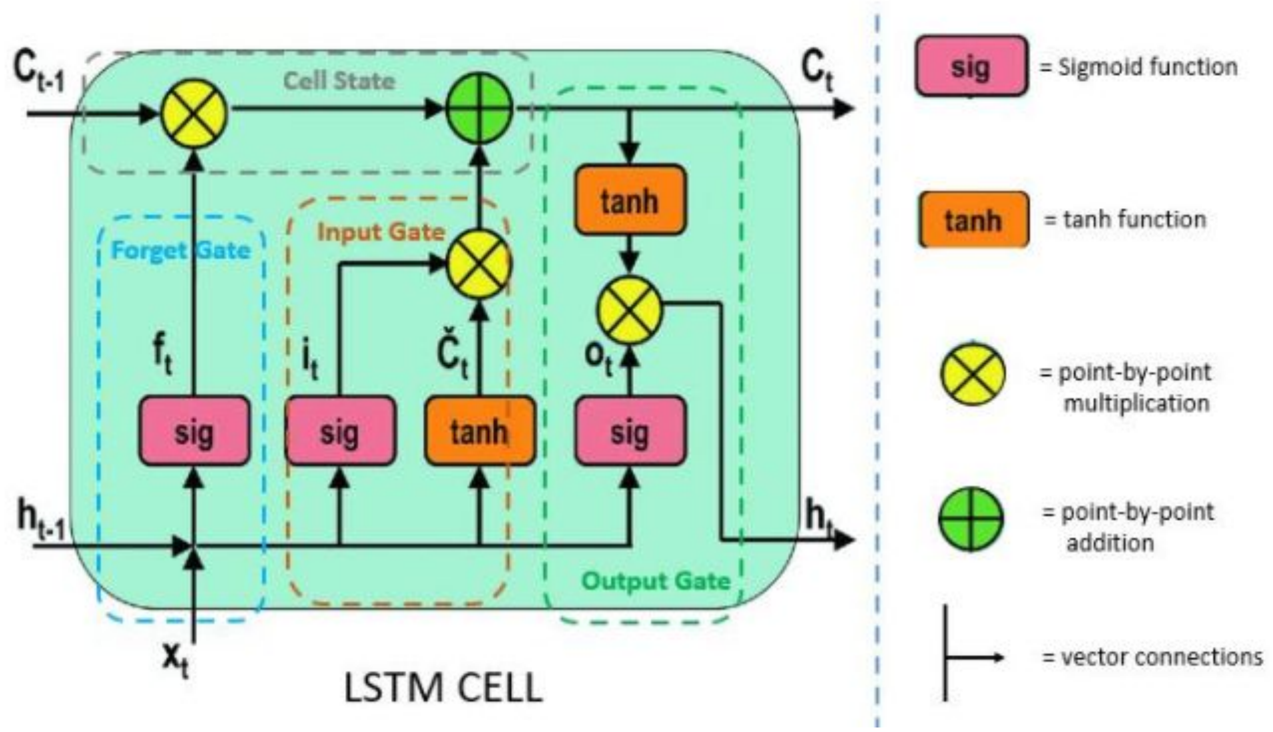
- Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning.
- Unlike standard feedforward neural networks, LSTM has feedback connections.
- A common LSTM unit is composed of;
  - a cell
  - an input gate
  - an output gate
  - a forget gate





Neural Networks

LSTM (Long short-term memory)



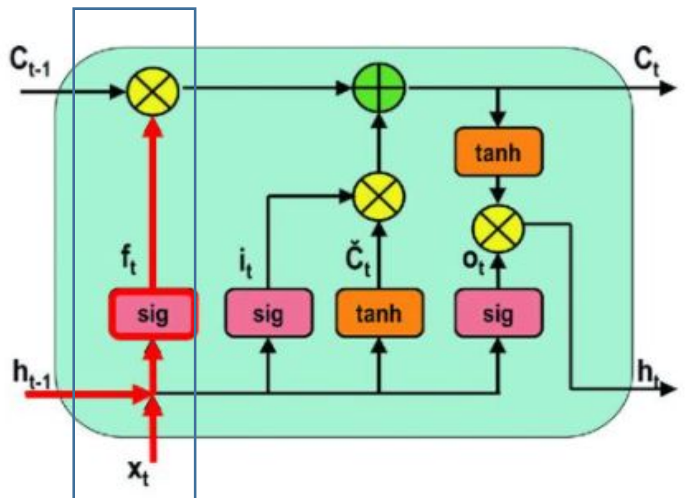
$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \quad (1)$$
$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \quad (2)$$
$$\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C) \quad (3)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \quad (4)$$
$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad (5)$$
$$h_t = o_t \odot \tanh(C_t) \quad (6)$$
$$\hat{y} = \text{softmax}(W_y h_t + b_y) \quad (7)$$

https://www.pluralsight.com/guides/introduction-to-lstm-units-in-rnn

Neural Networks

LSTM (Long short-term memory)

Forget Gate



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

t = timestep

f_t = forget gate at t

x_t = input

h_{t-1} = Previous hidden state

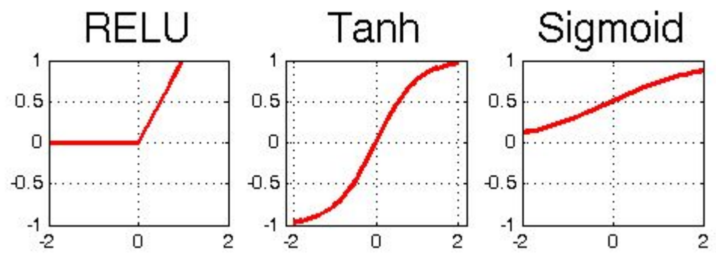
W_f = Weight matrix between forget gate and input gate

b_t = connection bias at t

Forget Gate Operation

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

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \qquad (4)$$



The forget gate decides which information needs attention and which can be ignored.

The information from the current input X(t) and hidden state h(t-1) are passed through the sigmoid function.

Sigmoid generates values between 0 and 1.

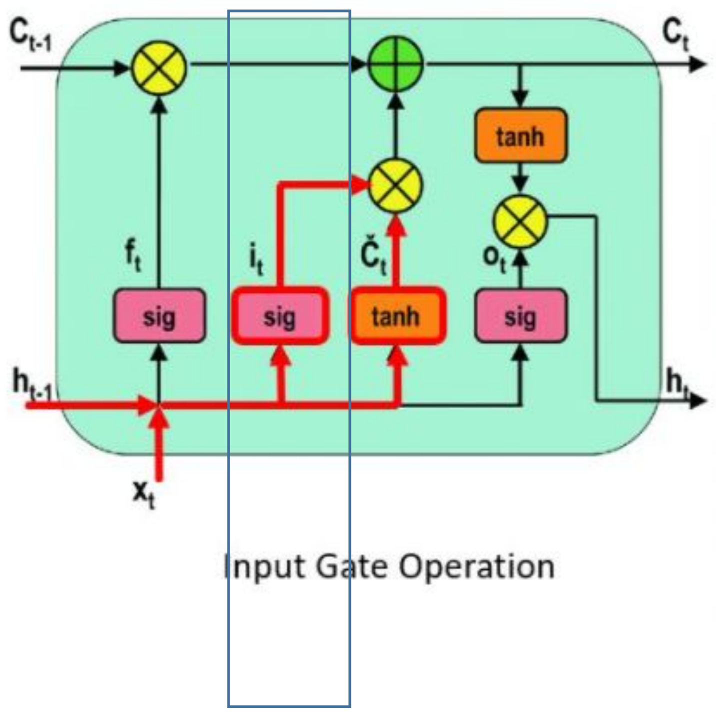
It concludes whether the part of the old output is necessary (by giving the output closer to 1).

This value of f(t) will later be used by the cell for *point-by-point multiplication (Hadamard Product).*

Neural Networks

LSTM (Long short-term memory)

Input Gate



Input Gate Operation

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

t = timestep

i_t = input gate at t

W_i = Weight matrix of sigmoid operator between input gate and output gate

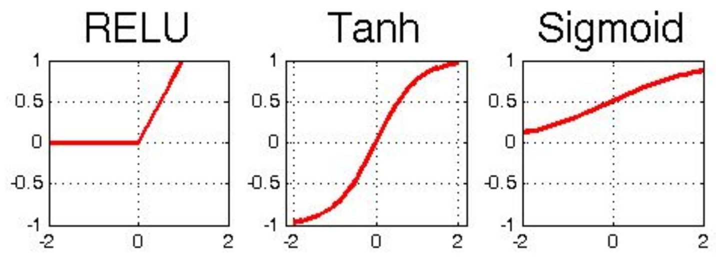
b_t = bias vector at t

$C\sim_t$ = value genrated by tanh

W_c = Weight matrix of tanh operator between cell state information and network output

b_c = bias vector at t.w.r.t W_c

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \qquad (2)$$
$$\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C) \qquad (3)$$



The input gate performs the following operations to update the cell status.

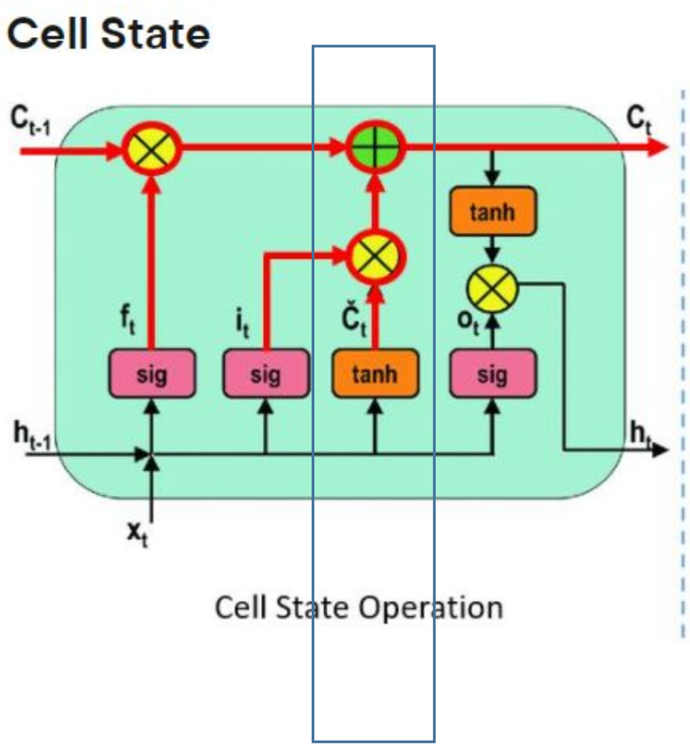
First, the current state X(t) and previously hidden state h(t-1) are passed into the second sigmoid function. The values are transformed between 0 (important) and 1 (not-important).

Next, the same information of the hidden state and current state will be passed through the tanh function. To regulate the network, the tanh operator will create a vector (C~(t)) with all the possible values between -1 and 1.

The output values generated form the activation functions are ready for point-by-point multiplication.

Neural Networks

LSTM (Long short-term memory)



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

t = timestep

C_t = cell state information

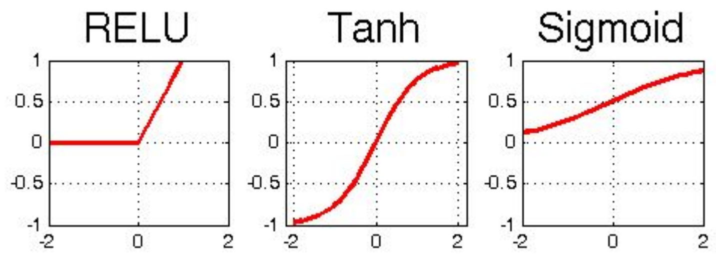
f_t = forget gate at t

i_t = input gate at t

C_{t-1} = Previous timestemp

$C\sim_t$ = value genrated by tanh

Cell State Operation



The network has enough information form the forget gate and input gate. The next step is to decide and store the information from the new state in the cell state.

The previous cell state C(t-1) gets multiplied with forget vector f(t). If the outcome is 0, then values will get dropped in the cell state.

Next, the network takes the output value of the input vector *i(t)* and performs point-by-point addition, which updates the cell state giving the network a *new cell state C(t).*

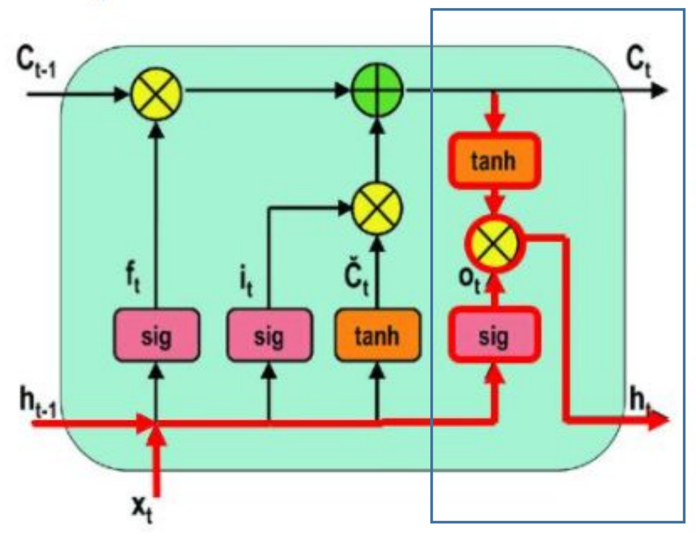
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \qquad (4)$$

Neural Networks

LSTM (Long short-term memory)

Output states, hidden states: Output may be
- a single LSTM cell hidden state
- several LSTM cell hidden states
- all the hidden states outputs

Output Gate



Output Gate Operation

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

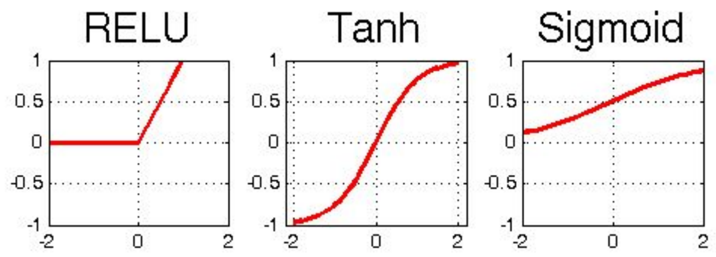
$t = timestep$

$O_t = output\ gate\ at\ t$

$W_o = Weight\ matrix\ of\ output\ gate$

$b_o = bias\ vector, w.r.t\ W_o$

$h_t = LSTM\ output$



The output gate determines the value of the next hidden state. This state contains information on previous inputs.

First, the values of the current state and previous hidden state are passed into the third sigmoid function. Then the new cell state generated from the cell state is passed through the *tanh* function.

Both these outputs are multiplied point-by-point. Based upon the final value, the network decides which information the hidden state should carry. This hidden state is used for prediction.

Finally, the new cell state and new hidden state are carried over to the next time step.

To conclude, the forget gate determines which relevant information from the prior steps is needed.

The input gate decides what relevant information can be added from the current step, and the output gates finalize the next hidden state

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad (5)$$
$$h_t = o_t \odot \tanh(C_t) \quad (6)$$

Neural Networks

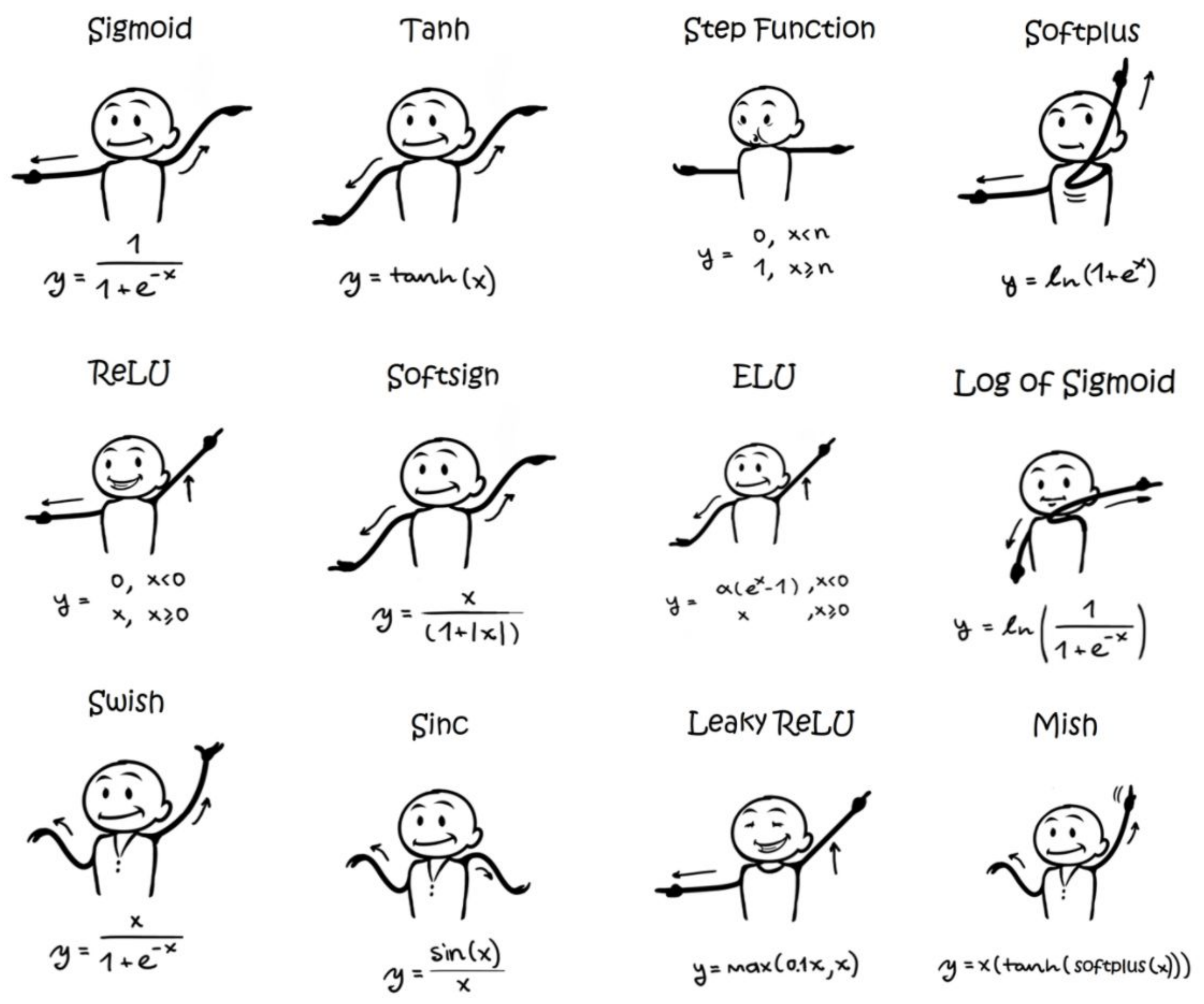
Other types of neural networks

- Gated Recurrent Units (GRU)
- Graph Neural Networks
- Transformer Networks
- Sequence models
- Capsule Networks
- Multi layer perceptron (MLP)
- Bayesian Networks
- ....

Concepts & Topics to have a look deeper

Neural Networks

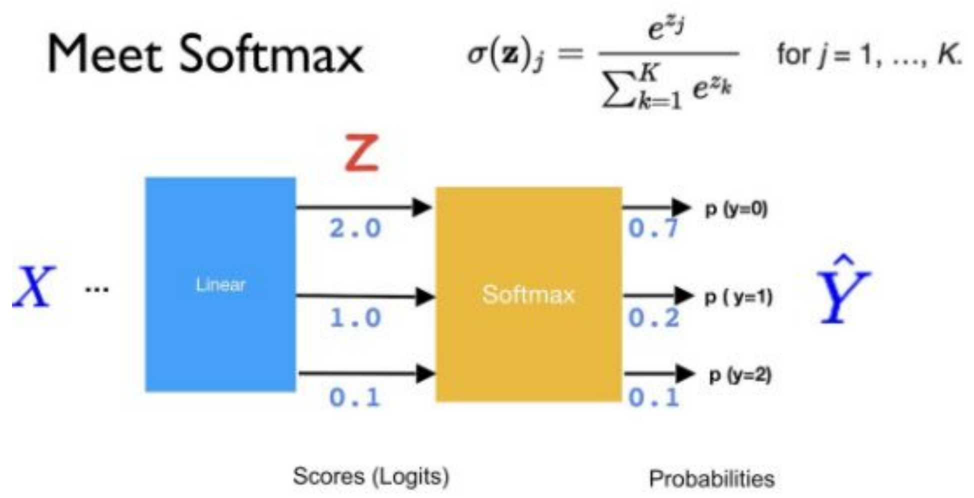
Activation Functions


Sigmoid
$y = \frac{1}{1+e^{-x}}$
Tanh
$y = \tanh(x)$
Step Function
$y = \begin{cases} 0, & x<n \\ 1, & x \geq n \end{cases}$
Softplus
$y = \ln(1+e^{x})$
ReLU
$y = \begin{cases} 0, & x<0 \\ x, & x \geq 0 \end{cases}$
Softsign
$y = \frac{x}{(1+|x|)}$
ELU
$y = \begin{cases} \alpha(e^{x}-1), & x<0 \\ x, & x \geq 0 \end{cases}$
Log of Sigmoid
$y = \ln\left(\frac{1}{1+e^{-x}}\right)$
Swish
$y = \frac{x}{1+e^{-x}}$
Sinc
$y = \frac{\sin(x)}{x}$
Leaky ReLU
$y = \max(0.1x, x)$
Mish
$y = x(\tanh(\text{softplus}(x)))$

Also, have a look at Softmax !

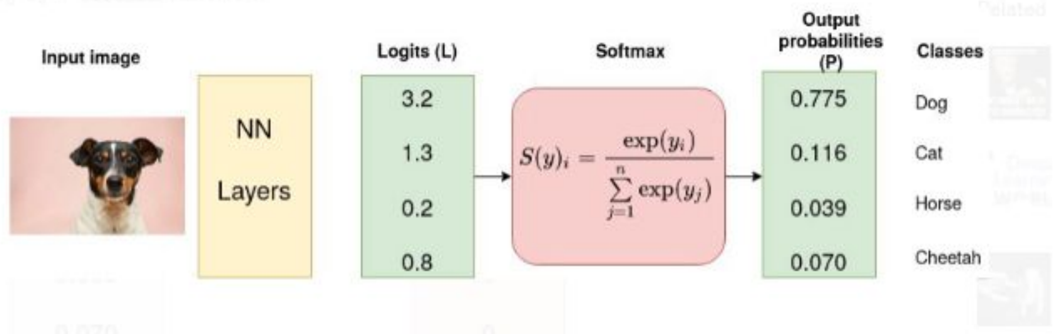
Neural Networks

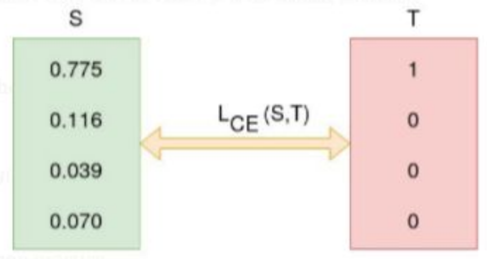
Softmax



The softmax function, also known as *softargmax*, or normalized exponential function is a generalization of the logistic function to multiple dimensions.

It is used in multinomial logistic regression and is often used as the last activation function of a neural network to normalize the output of a network to a probability distribution over predicted output classes, based on Luce's choice axiom.





Neural Networks

Neural Architecture Search

Neural architecture search (NAS) is a technique for automating the design of artificial neural networks (ANN), a widely used model in the field of machine learning.

NAS has been used to design networks that are on par or outperform hand-designed architectures. Methods for NAS can be categorized according to the search space, search strategy and performance estimation strategy used:

- The search space defines the type(s) of ANN that can be designed and optimized.
- The search strategy defines the approach used to explore the search space.
- The performance estimation strategy evaluates the performance of a possible ANN from its design (without constructing and training it).

NAS is closely related to hyperparameter optimization and meta-learning and is a subfield of automated machine learning (AutoML).