Unit	Topic	Proposed Lecture
I	**INTRODUCTION** – Learning, Types of Learning, Well defined learning problems, Designing a Learning System, History of ML, Introduction of Machine Learning Approaches – (Artificial Neural Network, Clustering, Reinforcement Learning, Decision Tree Learning, Bayesian networks, Support Vector Machine, Genetic Algorithm), Issues in Machine Learning and Data Science Vs Machine Learning;	08
II	**REGRESSION:** Linear Regression and Logistic Regression **BAYESIAN LEARNING -** Bayes theorem, Concept learning, Bayes Optimal Classifier, Naïve Bayes classifier, Bayesian belief networks, EM algorithm. **SUPPORT VECTOR MACHINE:** Introduction, Types of support vector kernel – (Linear kernel, polynomial kernel,and Gaussiankernel), Hyperplane – (Decision surface), Properties of SVM, and Issues in SVM.	08
III	**DECISION TREE LEARNING -** Decision tree learning algorithm, Inductive bias, Inductive inference with decision trees, Entropy and information theory, Information gain, ID-3 Algorithm, Issues in Decision tree learning. **INSTANCE-BASED LEARNING** – k-Nearest Neighbour Learning, Locally Weighted Regression, Radial basis function networks, Case-based learning.	08
IV	**ARTIFICIAL NEURAL NETWORKS** – Perceptron's, Multilayer perceptron, Gradient descent and the Delta rule, Multilayer networks, Derivation of Backpropagation Algorithm, Generalization, Unsupervised Learning – SOM Algorithm and its variant; **DEEP LEARNING -** Introduction,concept of convolutional neural network , Types of layers – (Convolutional Layers , Activation function , pooling , fully connected) , Concept of Convolution (1D and 2D) layers, Training of network, Case study of CNN for eg on Diabetic Retinopathy, Building a smart speaker, Self-deriving car etc.	08
V	**REINFORCEMENT LEARNING**–Introduction to Reinforcement Learning , Learning Task,Example of Reinforcement Learning in Practice, Learning Models for Reinforcement – (Markov Decision process , Q Learning - Q Learning function, Q Learning Algorithm), Application of Reinforcement Learning,Introduction to Deep Q Learning. **GENETIC ALGORITHMS:** Introduction, Components, GA cycle of reproduction, Crossover, Mutation, Genetic Programming, Models of Evolution and Learning, Applications.	08

Text books:

1. Tom M. Mitchell, —Machine Learning, McGraw-Hill Education (India) Private Limited, 2013.
2. Ethem Alpaydin, —Introduction to Machine Learning (Adaptive Computation and Machine Learning), The MIT Press 2004.
3. Stephen Marsland, —Machine Learning: An Algorithmic Perspective, CRC Press, 2009.
4. Bishop, C., Pattern Recognition and Machine Learning. Berlin: Springer-Verlag.

Artificial Neural Networks

- Neural Network learning method is a good approach to approximate real valued, discrete valued and vector valued target functions.
- Its complex as compared to other machine learning algorithms but once its get trained, it can solve the problems very fast and efficiently.
- Neural network is best for many complex practical problems like learning to recognize handwritten characters, learning to recognize spoken words or learning to recognize faces.
- It can also be used in traditional classification problem but as it is complex and takes a lot of time to train so preferably not used for traditional problems.

Artificial Neural Networks

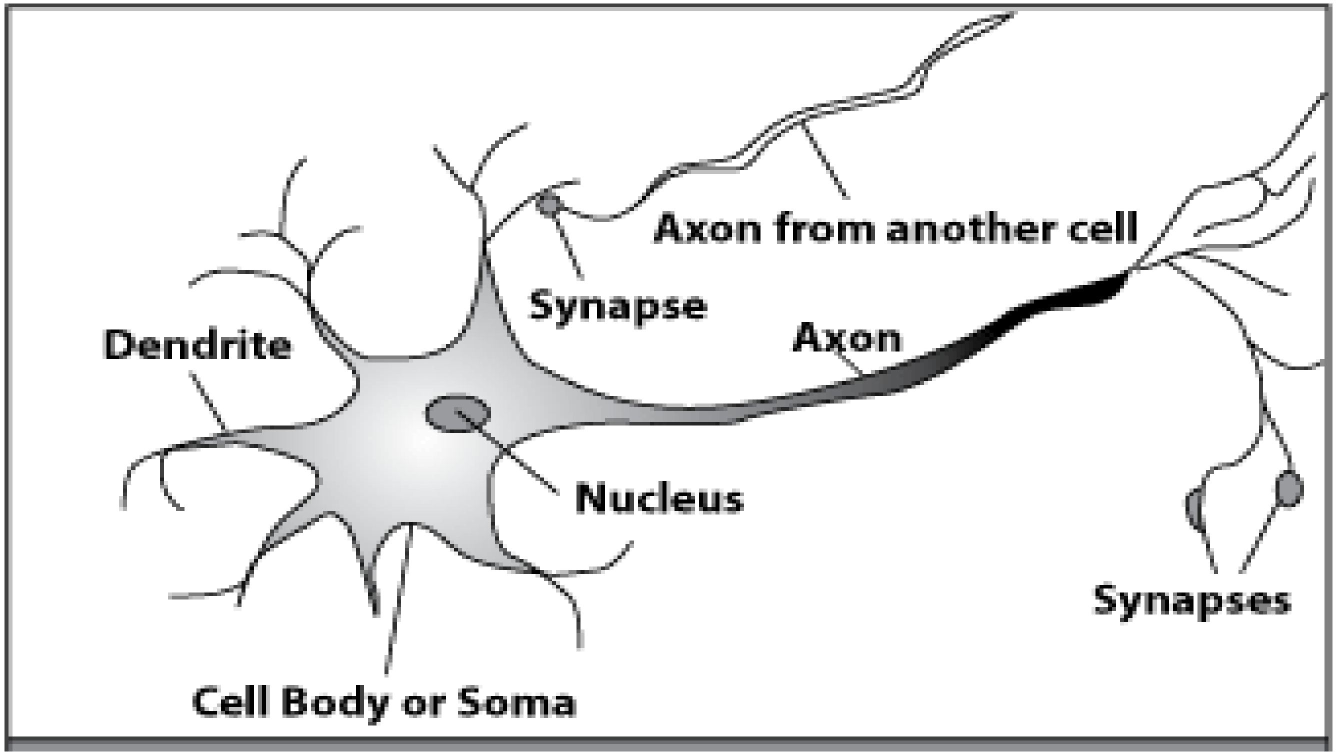
- Artificial Neural networks is an interconnected group of units which performs computation.
- Inspired by biological neural networks.
- The “building blocks” of neural networks are the neurons.
- In technical systems, we also refer to them as **units** or **nodes**.
- Each unit takes a number of real-valued inputs and produces a single real-valued output (which may become the input to many other units).
- The information-processing abilities of biological neural systems must follow from highly parallel processes operating on representations that are distributed over many neurons.
- Artificial Neural Networks are an imitation of the biological neural networks, but much simpler ones.
- The computing would have a lot to gain from neural networks. Their ability to learn by example makes them very flexible and powerful.

How do our brains work?

- The Brain is a massively parallel information processing system.
- Our brains is a huge network of processing elements. A typical brain contains a network of 10 billion neurons.

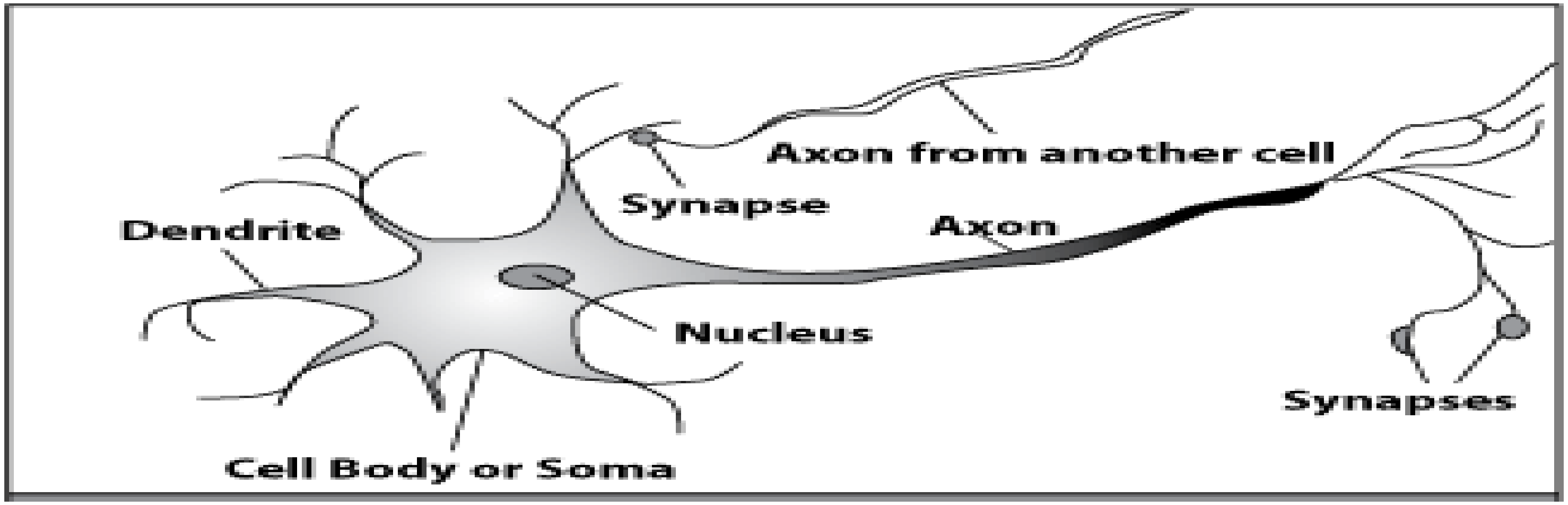
How do our brains work?

- A processing element



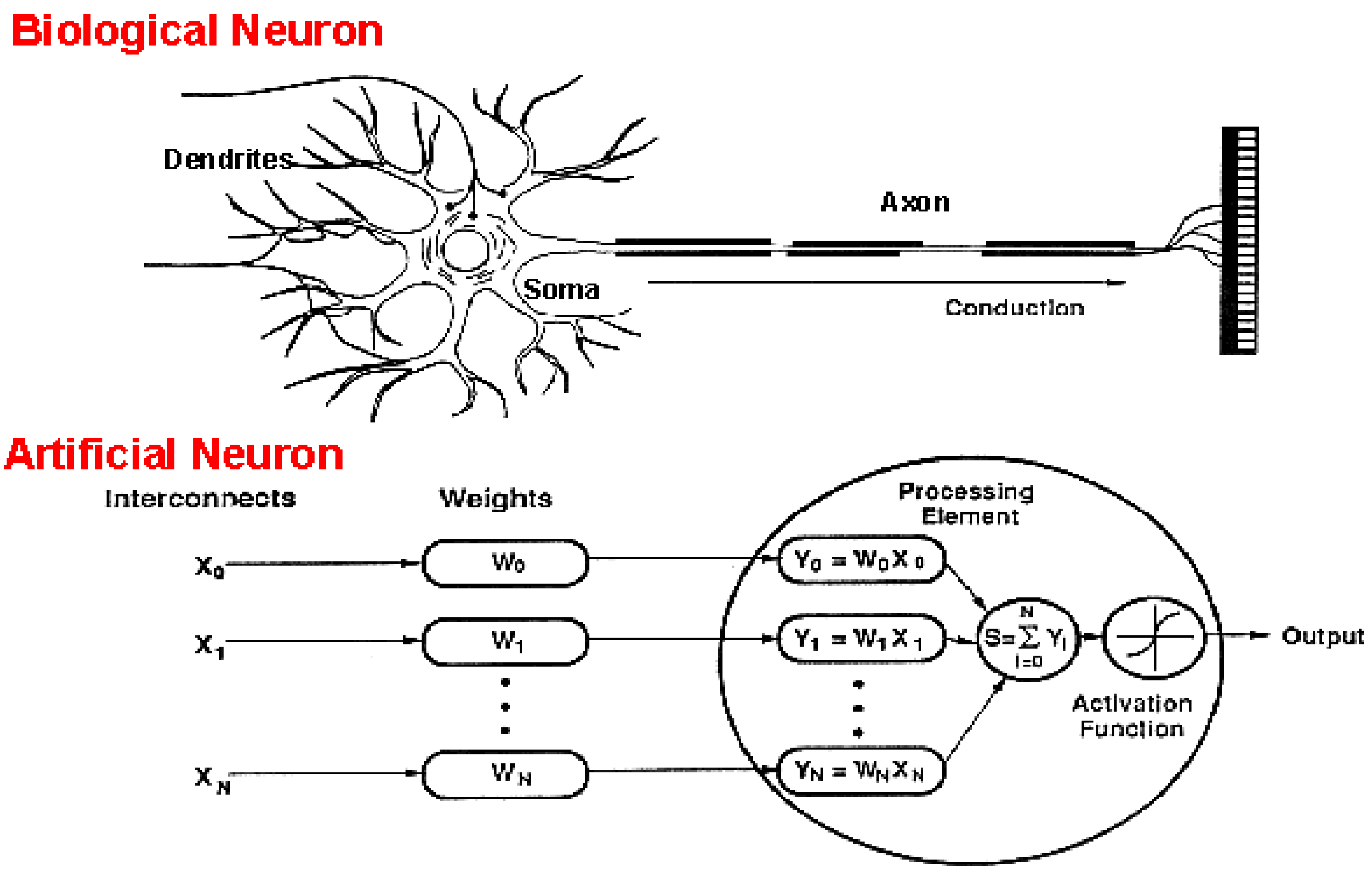
Dendrites: Input
Cell body: Processor
Synaptic: Link
Axon: Output

How do our brains work?



- A neuron is connected to other neurons through about *10,000 synapses*
- A neuron receives input from other neurons. Inputs are combined.
- Once input exceeds a critical level, the neuron discharges a spike - an electrical pulse that travels from the body, down the axon, to the next neuron(s)
- The axon endings almost touch the dendrites or cell body of the next neuron.
- Transmission of an electrical signal from one neuron to the next is effected by neurotransmitters.
- Neurotransmitters are chemicals which are released from the first neuron and which bind to the Second.
- This link is called a synapse. The strength of the signal that reaches the next neuron depends on factors such as the amount of neurotransmitter available.

How do ANNs work?



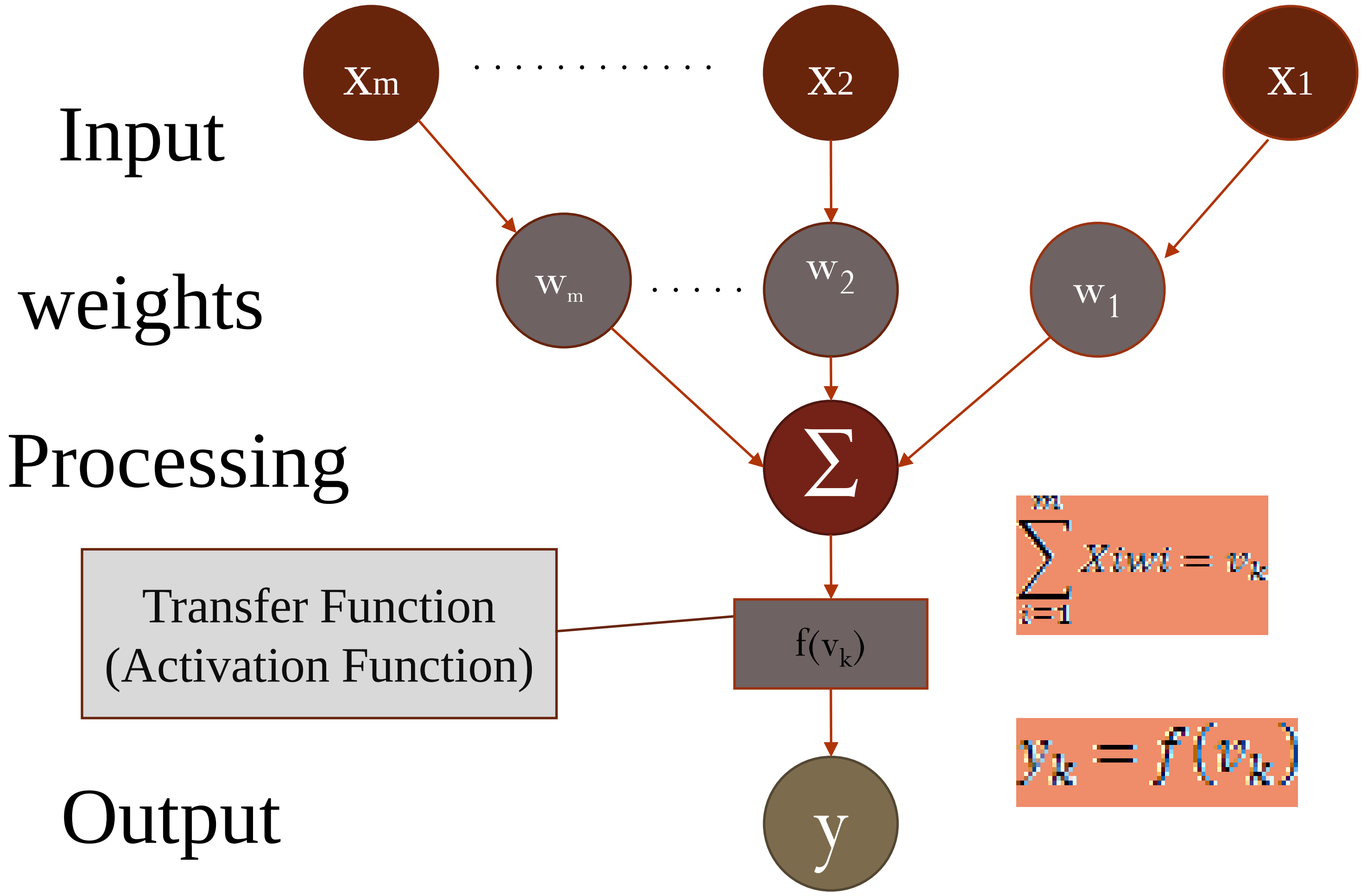
An artificial neuron is an imitation of a human neuron

ANNs

- An artificial neural network (ANN) is either a hardware implementation or a computer program which strives to simulate the information processing capabilities of its biological exemplar. ANNs are typically composed of a great number of interconnected artificial neurons. The artificial neurons are simplified models of their biological counterparts.
- ANN is a technique for solving problems by constructing software that works like our brains.

How do ANNs work?

- Now, let us have a look at the model of an artificial neuron. I

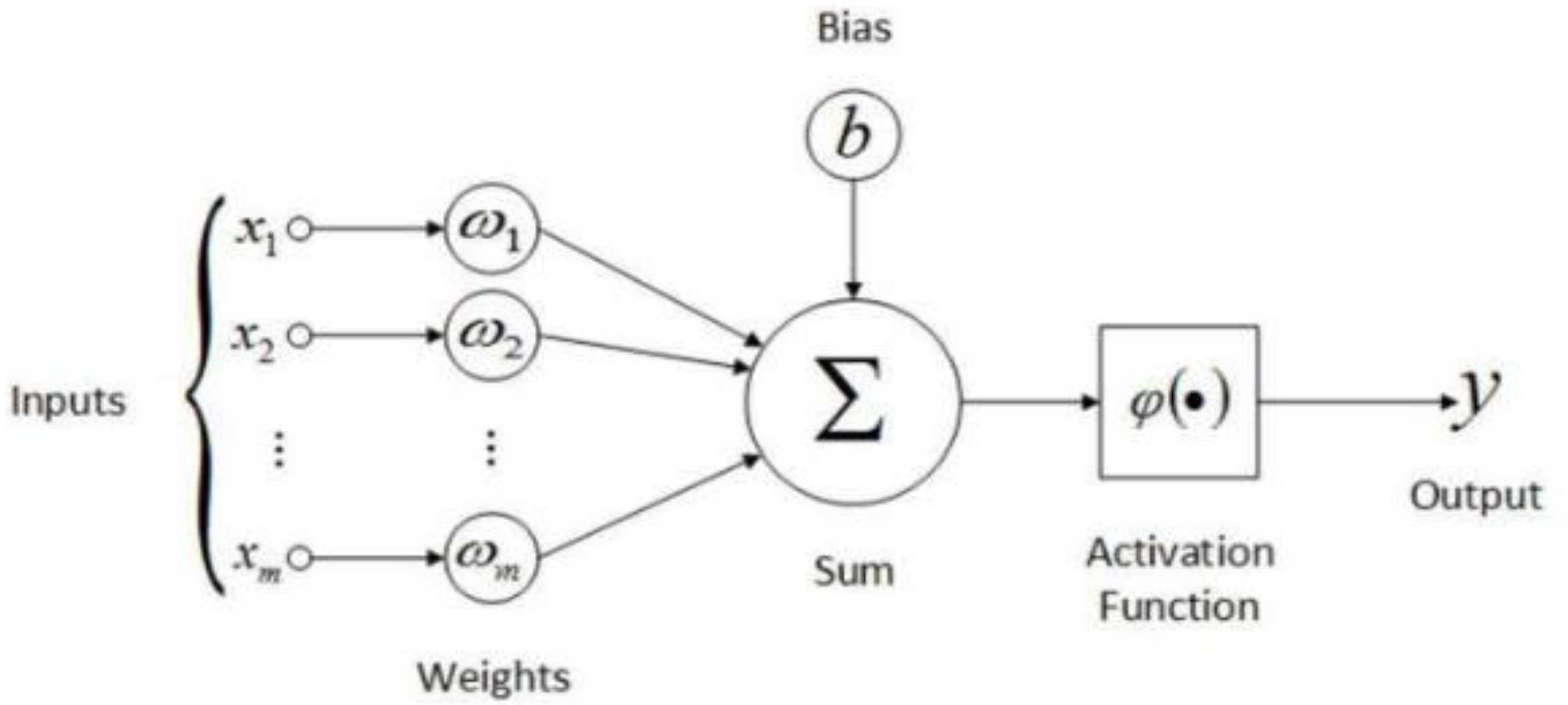


$$\sum_{i=1}^{m} Xiwi = v_k$$

$$y_k = f(v_k)$$

Perceptron

Perceptron is a simplest possible neural network.
It is a linear machine learning algorithm used for supervised learning for binary classifiers. It learns the weights in order to draw a linear decision boundary



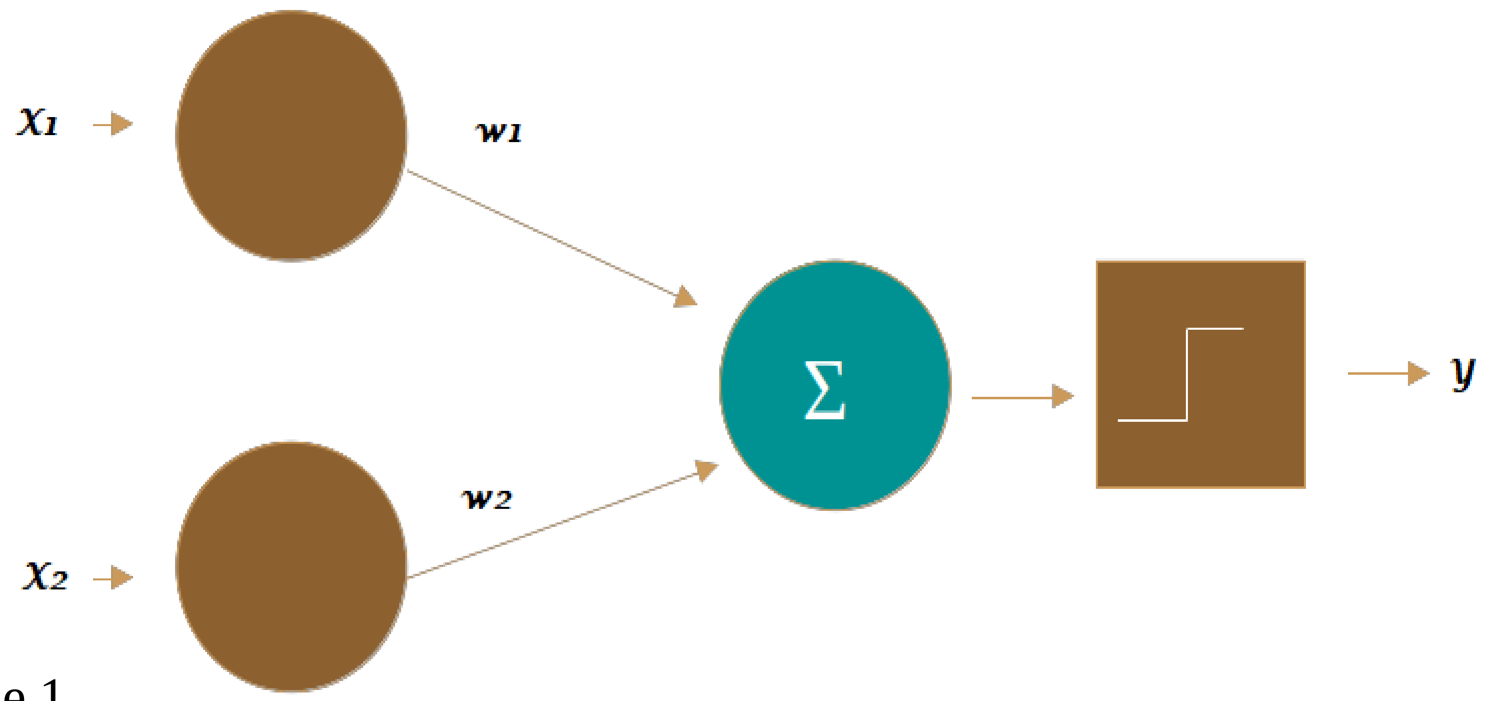
Characteristics of Perceptron Model

1. Perceptron is a machine learning algorithm for supervised learning for binary classifiers.
2. In Perceptron, the weight coefficient is automatically learned.
3. Weights multiplied with input features, decides whether the neuron is fired or not.
4. The activation function applies rule to check the weighted sum is compared with the preset threshold.
5. If the added sum of all input values is more than the threshold value, it must have an output signal; otherwise, no output will be shown.
6. The linear decision boundary is drawn, enabling the distinction between the two linearly separable classes +1 and -1.

Implementation of AND gate with Perceptron

X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1

Assume the weight values are
w1=1, w2=1&
threshold value=1.5



For case 1

y= w1*x1+w2*x2

=0*1+0*0

=0<1.5 so final output is 0

Case 4

y= w1*x1+w2*x2

=1*1+1*1

=2>1.50 so final output is 1

Training Rule

There are several algorithms for learning weights for single perceptron. Out of which one is **Perceptron Rule** and second is **Delta Rule**.

Perceptron Rule

- To learn an acceptable weight vector, begin with random weights, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example. This process is repeated, iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly. Weights are modified at each step according to the ***perceptron training rule,*** which revises the weight ***wi*** associated with input ***xi*** according to the rule

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

- Here ***t*** is the target output for the current training example, ***o*** is the output generated by the perceptron, and η is a positive constant called the ***learning rate.*** The role of the learning rate is to moderate the degree to which weights are changed at each step. It is usually set to some small value (e.g., 0.1) and is sometimes made to decay as the number of weight-tuning iterations increases.

Gradient Descent & Delta Rule

- Although the perceptron rule finds a successful weight vector when the training examples are linearly separable, it can fail to converge if the examples are not linearly separable. **A** second training rule, called the ***delta rule,*** is designed to overcome this difficulty.

- This rule is important because gradient descent provides the basis for the BACKPROPAGATION algorithm, which can learn networks with many interconnected units. It is also important because gradient descent can serve as the basis for learning algorithms that must search through hypothesis spaces containing many different types of continuously parameterized hypothesis.

- In order to derive a weight learning rule for linear units, let us begin by specifying a measure for the ***training error*** of a hypothesis (weight vector), relative to the training examples. Although there are many ways to define this error, one common measure that will turn out to be especially convenient is

$$E(\vec{w}) \equiv \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$$

- D is the set of training examples, ***td*** is the target output for training example *d*, and ***od*** is the output of the linear unit for training example d. By this definition, ***E(w)*** is simply half the squared difference between the target output ***td*** and the linear unit output ***od,*** summed over all training examples.

DERIVATION OF THE GRADIENT DESCENT RULE

- The steepest descent along the error surface can be calculated by computing the derivative of ***E*** with respect to each component of the vector w **.** This vector derivative is called the ***gradient*** of ***E*** with respect to w, written as ∇E(w)

$$\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_n}\right]$$

- ∇E(w) is itself a vector, whose components are the partial derivatives of ***E*** with respect to each of the ***wi*.** The training rule for gradient descent is

$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$$

- where $$\Delta\vec{w} = -\eta\nabla E(\vec{w})$$
- Here η is a positive constant called the learning rate, which determines the step size in the gradient descent search. The negative sign is present because we want to move the weight vector in the direction that ***decreases*** E. This training rule can also be written in its component form

$$w_i \leftarrow w_i + \Delta w_i$$

- where $$\Delta w_i = -\eta\frac{\partial E}{\partial w_i}$$

- The vector of derivatives that form the $\frac{\partial E}{\partial wi}$ gradient can be obtained by differentiating E

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\
&= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x_d}) \\
\frac{\partial E}{\partial w_i} &= \sum_{d \in D} (t_d - o_d)(-x_{id})
\end{aligned}$$

- where xid denotes the single input component xi for training example d.

- The weight update rule for gradient descent

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d)\ x_{id}$$

- To summarize, the gradient descent algorithm for training linear units is as follows:
- Pick an initial random weight vector.
- Apply the linear unit to all training examples, then compute Δw_i for each weight
- Update each weight ***wi*** by adding Δw_i , then repeat this process

Important note:

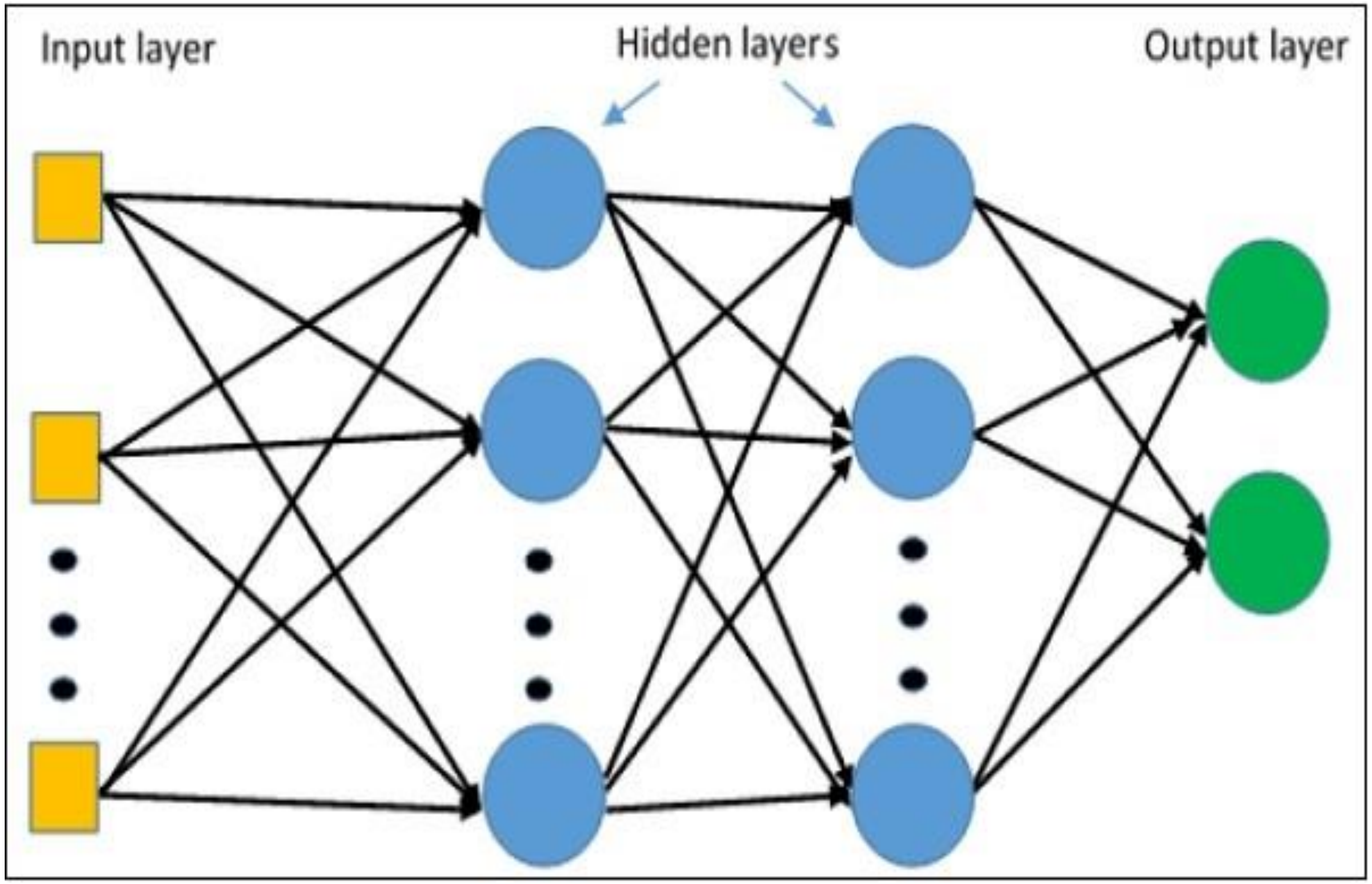
- In this algorithm the error surface contains only a single global minimum, this algorithm will converge to a weight vector with minimum error, regardless of whether the training examples are linearly separable, given a sufficiently small learning rate η is used. If η is too large, the gradient descent search runs the risk of overstepping the minimum in the error surface rather than settling into it. For this reason, one common modification to the algorithm is to gradually reduce the value of η as the number of gradient descent steps grows.

Multilayer Layer Perceptron (MLP)

MLP networks are usually used for supervised learning format.

A typical learning algorithm for MLP networks is also called back propagation’s algorithm

It train the model using Gradient Descent Rule..



- Single perceptron can only express linear decision surfaces. In contrast, the kind of multilayer networks learned by the **BACKPROPACATION** algorithm are capable of expressing a rich variety of nonlinear decision surfaces.
- Multilayer networks are studied using BACKPROPAGATION algorithm.

BACKPROPAGATION ALGORITHM

- For each training example d every weight ***wji*** is updated by adding to it ∆***wji***

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

where ***Ed*** is the error on training example d, summed over all output units in the network

$$E_d(\vec{w}) \equiv \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

- Here ***outputs*** is the set of output units in the network, ***tk*** is the target value of unit k for training example d, and ***ok*** is the output of unit k given training example d.

BACKPROPAGATION ALGORITHM

x_{ji} = the ith input to unit ***j***

w_{ji} = the weight associated with the ith input to unit ***j***

$net_j = \sum_i w_{ji} x_{ji}$ (the weighted sum of inputs for unit ***j)***

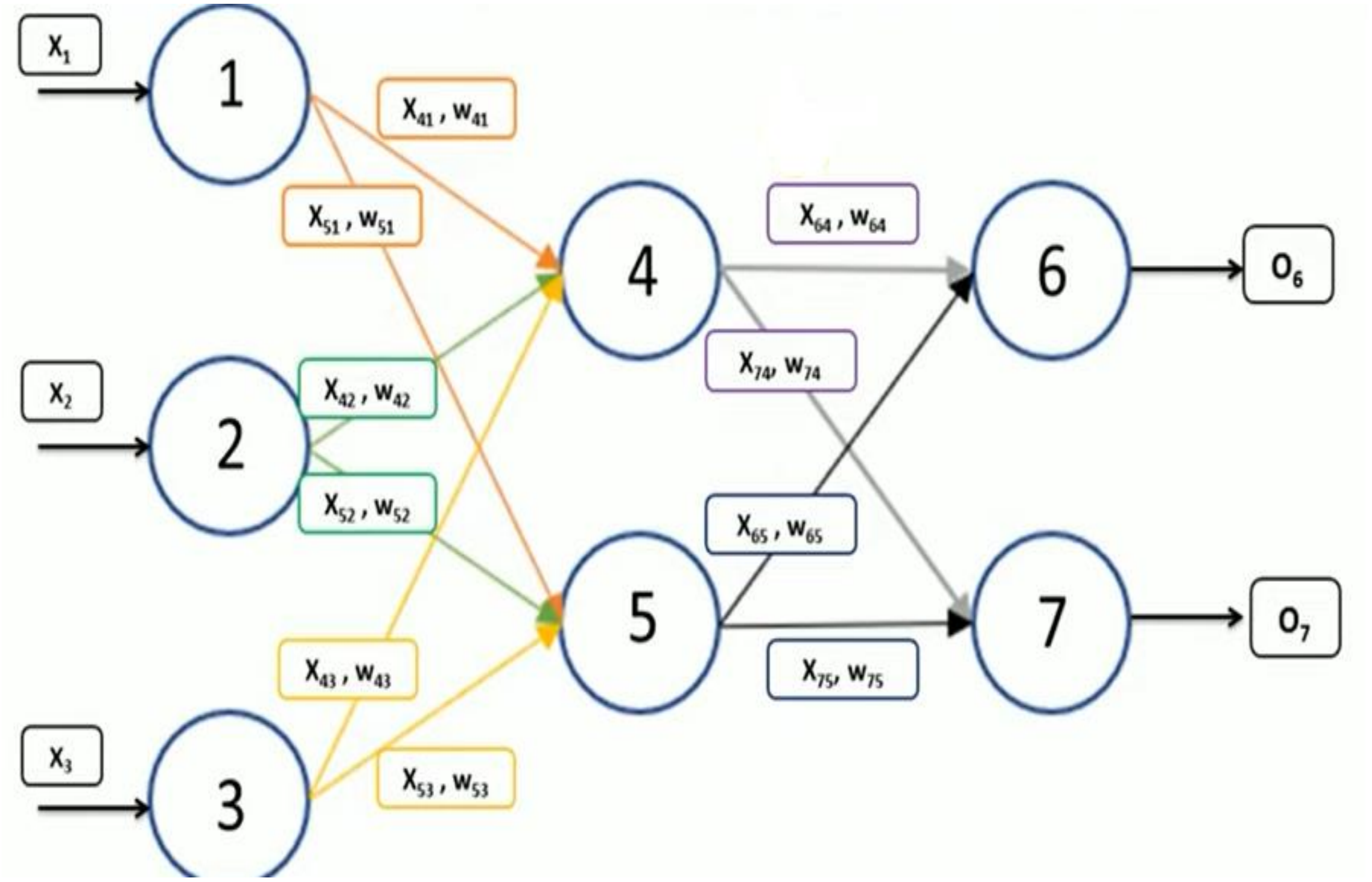
o_j = the output computed by unit ***j***

t_j = the target output for unit ***j***

σ = the sigmoid function

outputs = the set of units in the final layer of the network

Downstream(j) = the set of units whose immediate inputs include the output of unit ***j***



X1
X2
X3
1
2
3
X41 , w41
X51 , w51
X42 , w42
X52 , w52
X43 , w43
X53 , w53
4
5
X64 , w64
X74, w74
X65 , w65
X75, w75
6
7
o6
o7

- Weight wji can influence the rest of the network only through netj. Therefore, we can use the chain rule to write

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} x_{ji}$$

$$net_j = \sum_i w_{ji} X_{ji}$$

$$\frac{\partial net_j}{\partial w_{ji}} = x_{ji}$$

- Task is to derive a convenient expression for $\frac{\partial E_d}{\partial net_j}$

We consider two cases in turn: the case where unit j is an output unit for the network, and the case where j is an internal unit.

- **Case 1: Training Rule for Output Unit Weights,** when unit j is the output unit of the network.
- **Case 2: Training Rule for Hidden Unit Weights, ,** when unit j is an internal unit of the network.

- **Case 1: Training Rule for Output Unit Weights.** Just **as *wji*** can influence the rest of the network only through ***netj, netj*** can influence the network only through ***oj.*** Therefore, we can invoke the chain rule again to write

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}\sum_{k\in outputs}(t_k - o_k)^2$$

- The derivatives $\frac{\partial}{\partial o_j}$ ***(tk** – **ok)2*** will be zero for all output units k except when k = j.

We therefore drop the summation over output units and simply set k = j.

$$\begin{aligned}\frac{\partial E_d}{\partial o_j} &= \frac{\partial}{\partial o_j}\frac{1}{2}(t_j - o_j)^2\\ &= \frac{1}{2}2(t_j - o_j)\frac{\partial(t_j - o_j)}{\partial o_j}\\ &= -(t_j - o_j)\end{aligned}$$

- Since ***oj*** = ***σ(netj),*** the derivative is just the derivative of the sigmoid function, which is equal to ***σ((netj)(l*** - ***σ((netj)).*** Therefore

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial \sigma(net_j)}{\partial net_j}$$

$$= o_j(1 - o_j)$$

- So,

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)\ o_j(1 - o_j)$$

and

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} = \eta\ (t_j - o_j)\ o_j(1 - o_j)x_{ji}$$

$$\delta_j = (t_j - o_j)\ o_j(1 - o_j)$$

$$\Delta w_{ji} = \eta\, \delta_j\, x_{ji}$$

- **Case 2: Training Rule for Hidden Unit Weights.** In the case where ***j*** is an internal, or hidden unit in the network, the derivation of the training rule for ***wji*** must take into account the indirect ways in which ***wji*** can influence the network outputs and hence ***Ed.***
- we will find it useful to refer to the set of all units immediately downstream of unit j in the network (i.e., all units whose direct inputs include the output of unit j). We denote this set of units by ***Downstream(*** j)

$$\begin{aligned}
\frac{\partial E_d}{\partial net_j} &= \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \, w_{kj} \frac{\partial o_j}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \, w_{kj} \, o_j(1 - o_j)
\end{aligned}$$

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)\, o_j(1 - o_j)$$

$$\delta_j = (t_j - o_j)\, o_j(1 - o_j)$$

$$\frac{\partial o_j}{\partial (net_j)} = \frac{\partial \sigma(net_j)}{\partial (net_j)}$$

$$\frac{\partial\, net_k}{\partial o_j} = \frac{\partial x_{kj} w_{kj}}{\partial o_j} = \frac{\partial o_j w_{kj}}{\partial o_j}$$

$$= \sigma(net_j)(1 - \sigma(net_j))$$

$$= o_j (1 - o_j)$$

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial net_j} x_{ii}$$

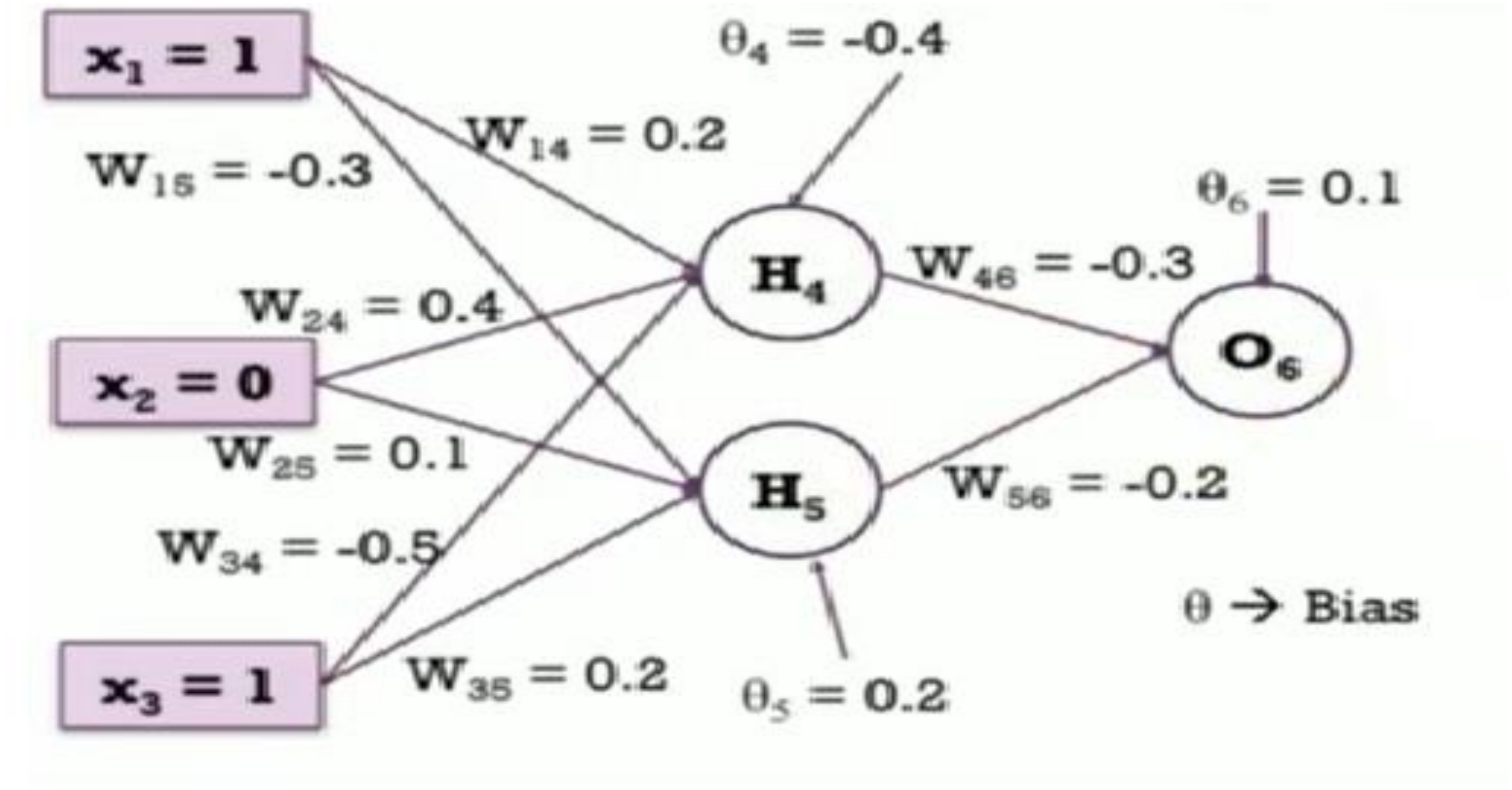
$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in Downstream(j)} -\delta_k \, w_{kj} \, o_j(1 - o_j)$$

$$\Delta w_{ji} = \eta \, o_j(1 - o_j) \sum_{k \in Downstream(j)} \delta_k \, w_{kj} \, x_{ji}$$

$$\Delta w_{ji} = \eta \, \delta_j \, x_{ji}$$

$$\delta_j = o_j(1 - o_j) \sum_{k \in Downstream(j)} \delta_k \, w_{kj}$$

If target output is 1 and learning rate is 0.9 then using sigmoid propagation function, apply backpropagation algorithm for the given network.



- **netj =** $\sum_j \boldsymbol{wij} * \boldsymbol{xi}$

O(H4) = σ (netj) = $\frac{1}{1+e^{-netj}}$

1. net4 = (w14*x1) + (w24*x2) + (w34*x3) + $\boldsymbol{\theta}$4

   = (0.2*1) + (0.4*0) + (-0.5*1) + (-0.4) = -0.7

O(H4) = σ (net4) = $\frac{1}{1+e^{0.7}} = 0.332$

2. net5 = (w15*x1) + (w25*x2) + (w35*x3) + $\boldsymbol{\theta}$5

   = (-0.3*1) + (0.1*0) + (0.2*1) + (0.2) = 0.1

O(H5) = σ (net5) = $\frac{1}{1+e^{-0.1}} = 0.525$

3. net6 = (w46*H4) + (w56*H5) + $\boldsymbol{\theta}$6

   = (-0.3*0.332) + (-0.2*0.525) + (0.1) = -0.105

O(H6) = σ (net6) = $\frac{1}{1+e^{0.105}} = 0.474$

Error = 1- 0.474 = 0.526

Now modify the weights to reduce the error.

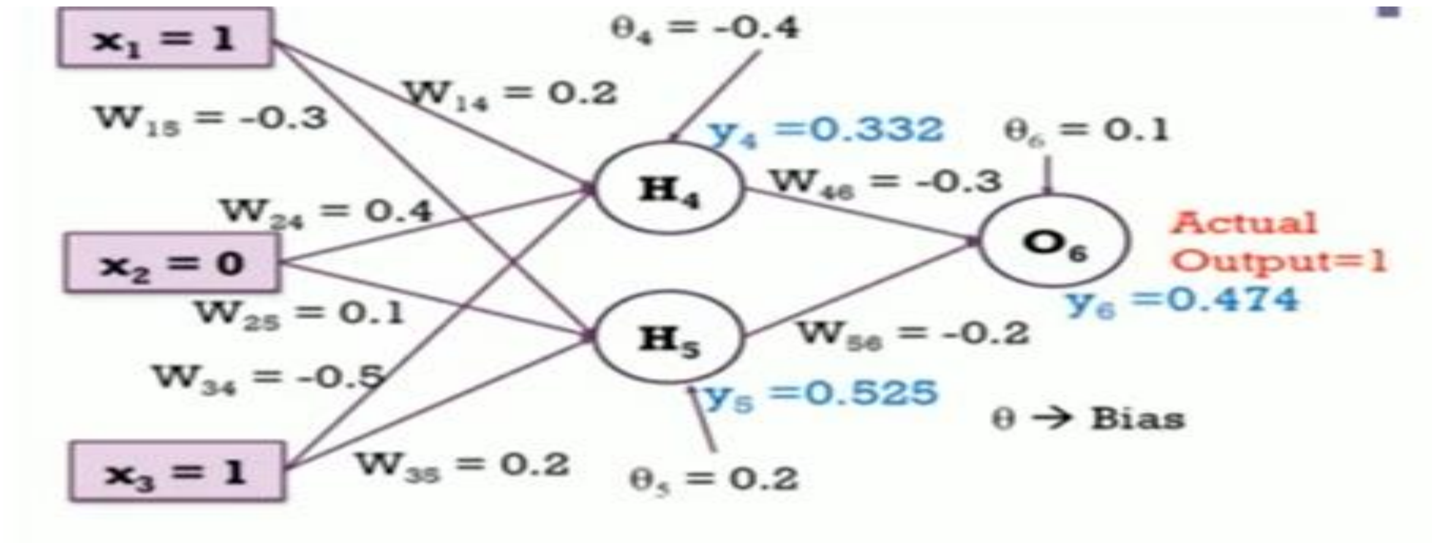
- Each weight changed by:

$$\Delta w_{ji} = \eta \delta_j o_i$$

$$\delta_j = o_j(1-o_j)(t_j - o_j) \qquad \text{if } j \text{ is an output unit}$$

$$\delta_j = o_j(1-o_j)\sum_k \delta_k w_{kj} \qquad \text{if } j \text{ is a hidden unit}$$

- where η is a constant called the learning rate
- tj is the correct teacher output for unit j
- δj is the error measure for unit j



- Backward Pass: Compute δ4, δ5 and δ6.

For output unit:

$\delta_6 = y_6(1-y_6)\,(y_{target} - y_6)$

$= 0.474*(1-0.474)*(1-0.474) = 0.1311$

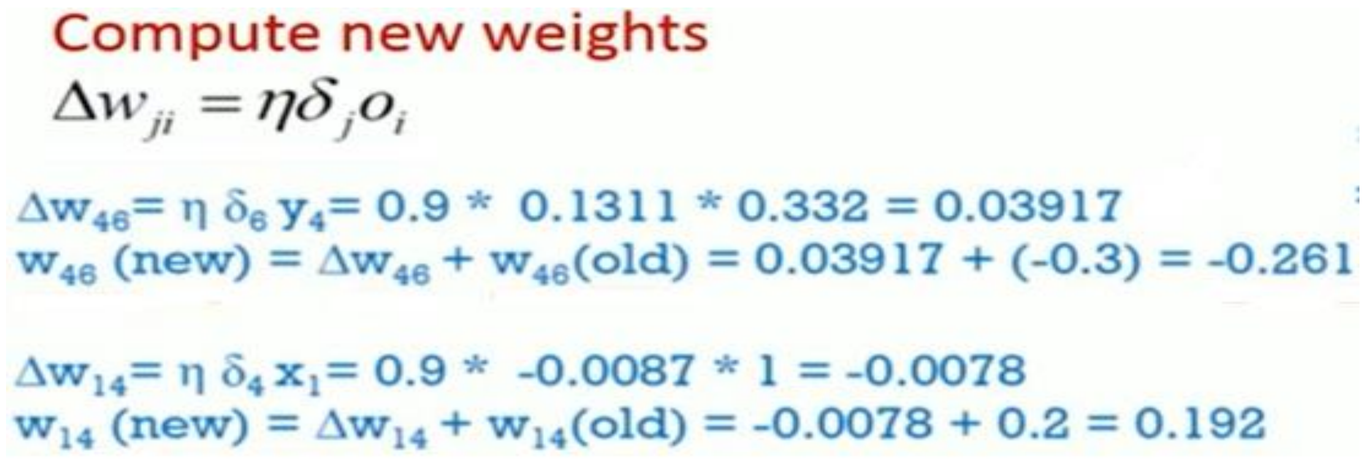
For hidden unit:

$\delta_5 = y_5(1-y_5)\, w_{56} * \delta_6$

$= 0.525*(1 - 0.525)*(-0.2 * 0.1311) = -0.0065$

$= y_4(1-y_4)\, w_{46} * \delta_6$

$= 0.332*(1 - 0.332)* (-0.3 * 0.1331) = -0.0087$



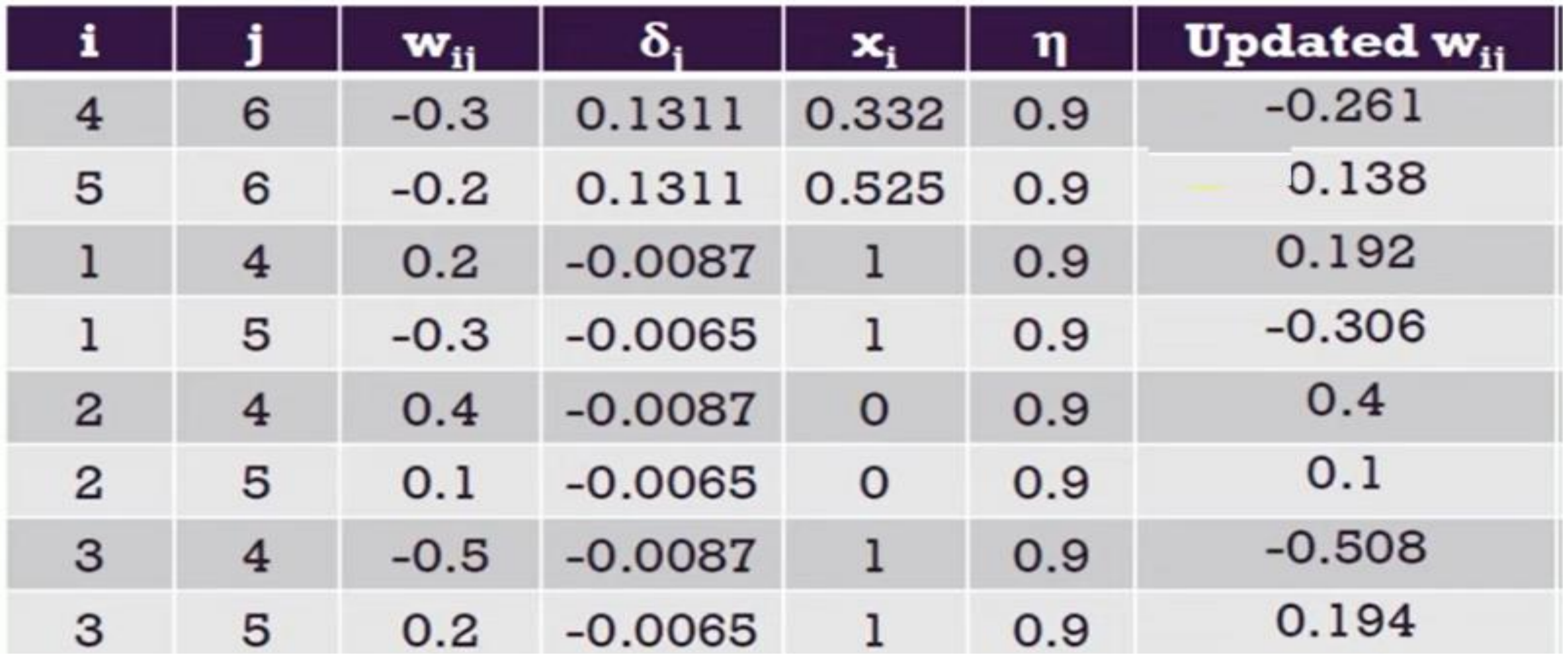
Compute new weights

$$\Delta w_{ji} = \eta \delta_j o_i$$

$\Delta w_{46} = \eta\ \delta_6\ y_4 = 0.9 * 0.1311 * 0.332 = 0.03917$
$w_{46}\ (\text{new}) = \Delta w_{46} + w_{46}(\text{old}) = 0.03917 + (-0.3) = -0.261$

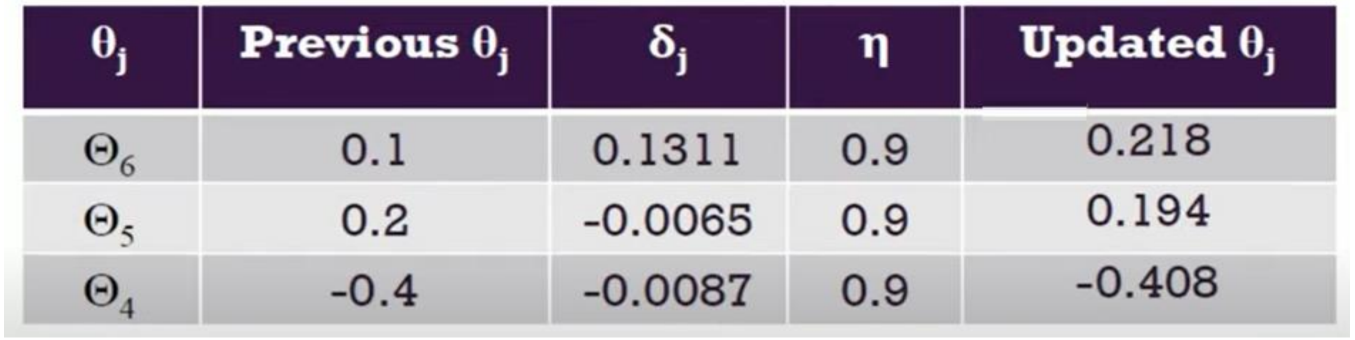
$\Delta w_{14} = \eta\ \delta_4\ x_1 = 0.9 * -0.0087 * 1 = -0.0078$
$w_{14}\ (\text{new}) = \Delta w_{14} + w_{14}(\text{old}) = -0.0078 + 0.2 = 0.192$

Update all the weights

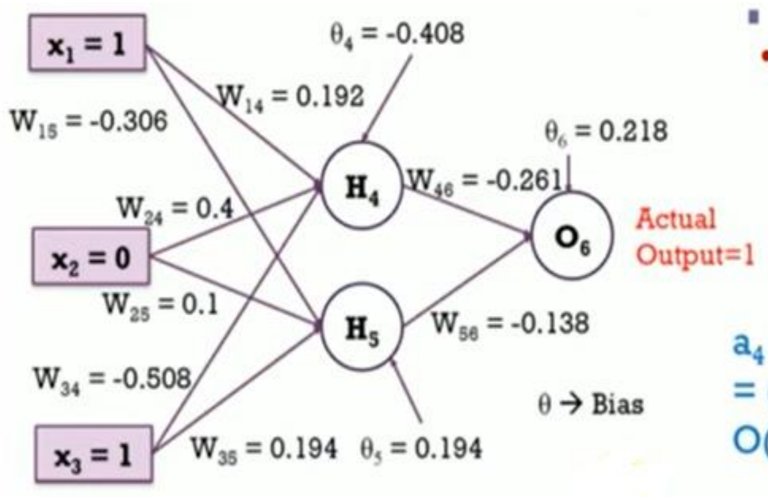


i	j	w_{ij}	δ_j	x_i	η	Updated w_{ij}
4	6	-0.3	0.1311	0.332	0.9	-0.261
5	6	-0.2	0.1311	0.525	0.9	0.138
1	4	0.2	-0.0087	1	0.9	0.192
1	5	-0.3	-0.0065	1	0.9	-0.306
2	4	0.4	-0.0087	0	0.9	0.4
2	5	0.1	-0.0065	0	0.9	0.1
3	4	-0.5	-0.0087	1	0.9	-0.508
3	5	0.2	-0.0065	1	0.9	0.194

Update the bias terms



θ_j	Previous θ_j	δ_j	η	Updated θ_j
Θ_6	0.1	0.1311	0.9	0.218
Θ_5	0.2	-0.0065	0.9	0.194
Θ_4	-0.4	-0.0087	0.9	-0.408



Error $= y_{target} - y_6 = 0.485$

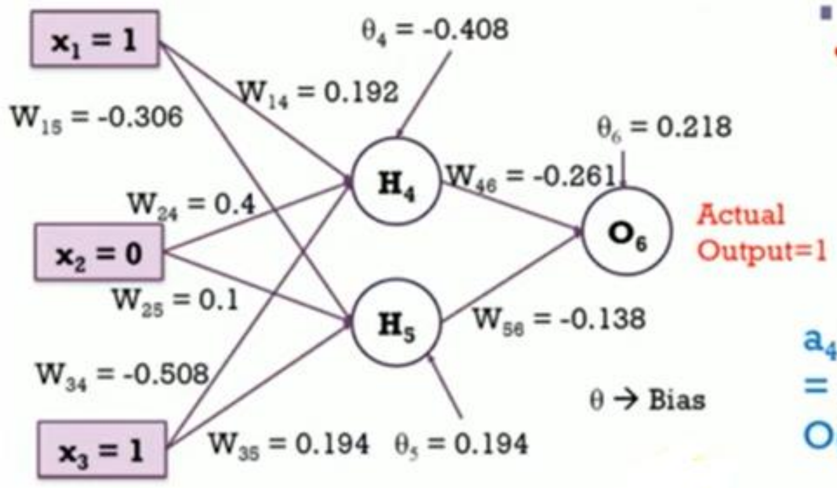
- Forward Pass: Compute output for y4, y5 and y6.

$$a_j = \sum_j (w_{i,j} * x_i) \qquad yj = F(aj) = \frac{1}{1 + e^{-a_j}}$$

$a_4 = (w_{14} * x_1) + (w_{24} * x_2) + (w_{34} * x_3) + \theta_4$
$= (0.192 * 1) + (0.4 * 0) + (-0.508 * 1) + (-0.408) = -0.724$
$O(H_4) = y_4 = f(a_4) = 1/(1 + e^{0.724}) = 0.327$

$a_5 = (w_{15} * x_1) + (w_{25} * x_2) + (w_{35} * x_3) + \theta_5$
$= (-0.306 * 1) + (0.1 * 0) + (0.194 * 1) + (0.194) = 0.082$
$O(H_5) = y_5 = f(a_5) = 1/(1 + e^{-0.082}) = 0.520$

$a_6 = (w_{46} * H_4) + (w_{56} * H_5) + \theta_6$
$= (-0.261 * 0.327) + (-0.138 * 0.520) + 0.218 = 0.061$
$O(O_6) = y_6 = f(a_6) = 1/(1 + e^{-0.061}) =$ **0.515** (Network Output)



Error $= y_{target} - y_6 = 0.485$

- Forward Pass: Compute output for y4, y5 and y6.

$$a_j = \sum_j (w_{i,j} * x_i) \qquad yj = F(aj) = \frac{1}{1 + e^{-a_j}}$$

$a_4 = (w_{14} * x_1) + (w_{24} * x_2) + (w_{34} * x_3) + \theta_4$
$= (0.192 * 1) + (0.4 * 0) + (-0.508 * 1) + (-0.408) = -0.724$
$O(H_4) = y_4 = f(a_4) = 1/\ (1 + e^{0.724}) = 0.327$

$a_5 = (w_{15} * x_1) + (w_{25} * x_2) + (w_{35} * x_3) + \theta_5$
$= (-0.306 * 1) + (0.1 * 0) + (0.194 * 1) + (0.194) = 0.082$
$O(H_5) = y_5 = f(a_5) = 1/\ (1 + e^{-0.082}) = 0.520$

$a_6 = (w_{46} * H_4) + (w_{56} * H_5) + \theta_6$
$= (-0.261 * 0.327) + (-0.138 * 0.520) + 0.218 = 0.061$
$O(O_6) = y_6 = f(a_6) = 1/\ (1 + e^{-0.061}) = 0.515$ (Network Output)

- Now error is reduced with the new weights. If it is acceptable then stop here otherwise again modify the weights.

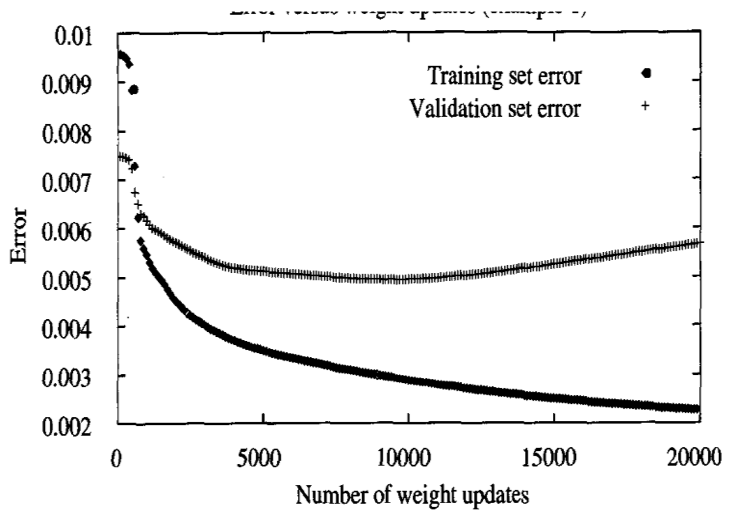
Generalization

- The termination condition for the algorithm has been left unspecified.
- Generalization is affected by three factors

1. Training set size
2. Architecture of the neural network
3. Complexity of the problem

Most of the time for generalization training set size and architecture is considered as Complex problems are always solved using ANN.

- One obvious choice of Generalization is to continue training until the error E on the training examples falls below some predetermined threshold. In fact, this is a poor strategy because BACKPROPAGATION susceptible to overfitting the training examples at the cost of decreasing generalization accuracy over other unseen examples.
- The lower of the two lines shows the monotonically decreasing error E over the training set, as the number of gradient descent iterations grows. The upper line shows the error E measured over a different ***validation set*** of examples, distinct from the training examples. This line measures the ***generalization accuracy*** of the network-the accuracy with which it fits examples beyond the training data.
- generalization accuracy measured over the validation examples first decreases, then increases, even as the error over the training examples continues to decrease. How can this occur? This occurs because the weights are being tuned to fit idiosyncrasies of the training examples that are not representative of the general distribution of examples.



The large number of weight parameters in ANNs provides many degrees of freedom for fitting such idiosyncrasies.

- **Overfitting tends to occur during later iterations but not during earlier iterations.**
- With weights of nearly identical value, only very smooth decision surfaces are describable. As training proceeds, some weights begin to grow in order to reduce the error over the training data, and the complexity of the learned decision surface increases. Thus, the effective complexity of the hypotheses that can be reached by **BACKPROPAGATION** increases with the number of weight-tuning iterations. Given enough weight-tuning iterations, **will** often be able to create overly complex decision surfaces that fit noise in the training data or unrepresentative characteristics of the particular training sample. This overfitting problem is analogous to the overfitting problem in decision tree learning.

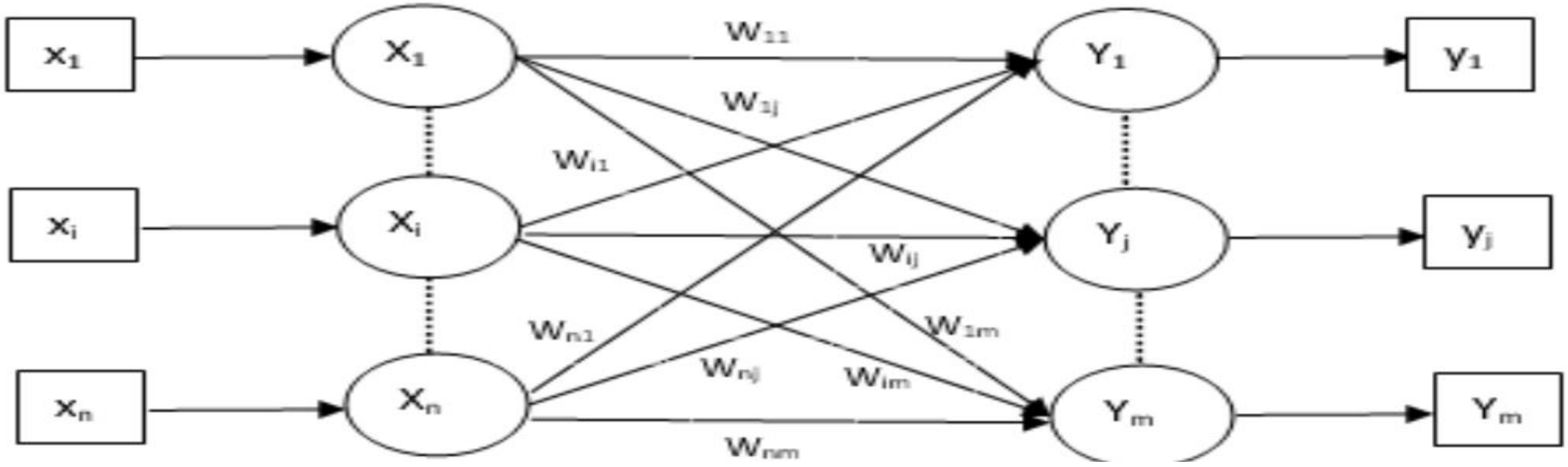
Approaches to solve overfitting

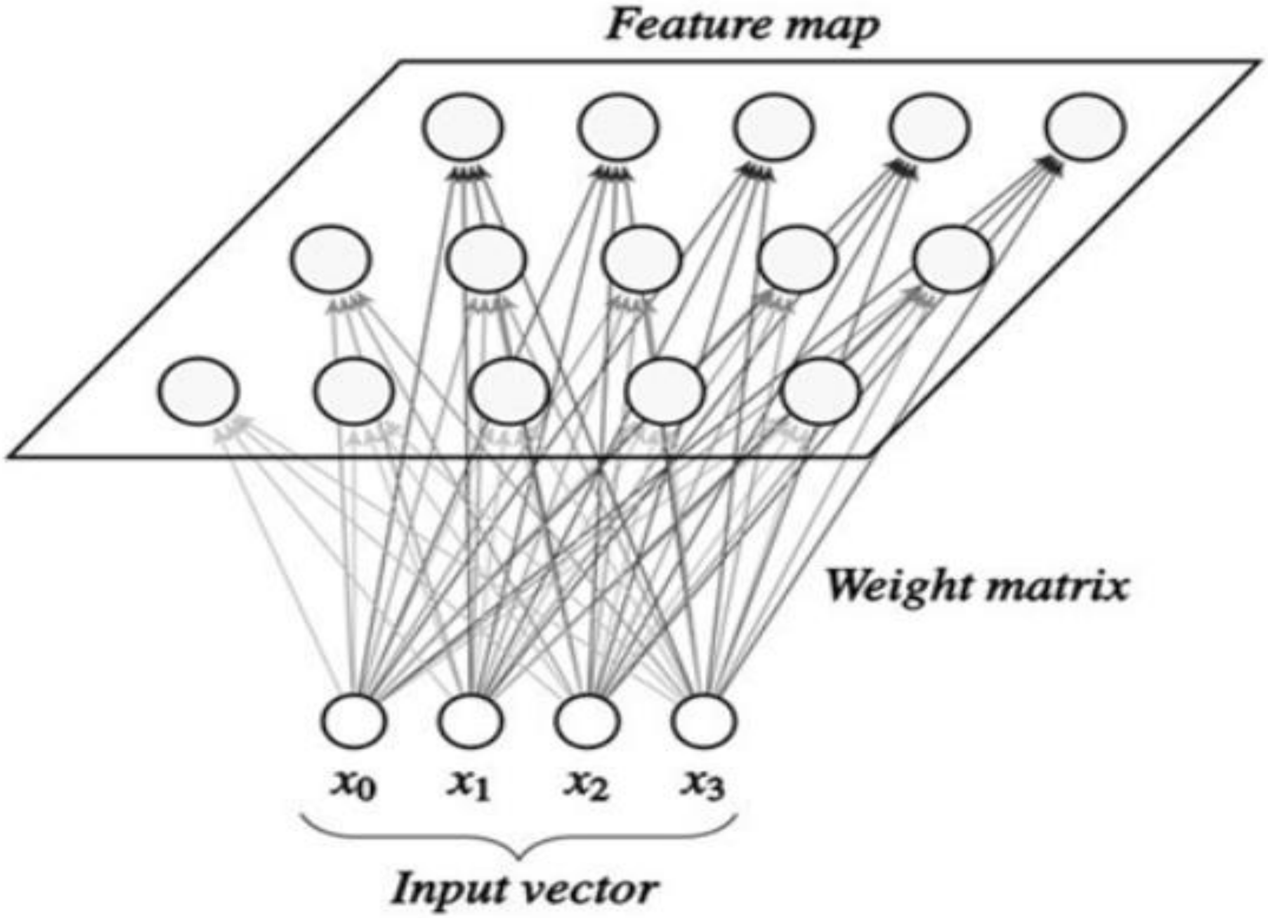
- **Weight Decay : D**ecrease each weight by some small factor during each iteration. The motivation for this approach is to keep weight values small, to bias learning against complex decision surfaces.
- One of the most successful methods for overcoming the overfitting problem is to simply provide a set of validation data to the algorithm in addition to the training data. The algorithm monitors the error with respect to this validation set, while using the training set to drive the gradient descent search. In essence, this allows the algorithm itself to plot the two curves shown in the previous slide.
- How many weight-tuning iterations should the algorithm perform? It should use the number of iterations that produces the lowest error over the validation set, since this is the best indicator of network performance over unseen examples. In typical implementations of this approach, two copies of the network weights are kept one copy for training and a separate copy of the best-performing weights thus far, measured by their error over the validation set.

Self Organizing Maps(SOM)

- SOM was developed by T. Kohonen in 1982.
- Its an unsupervised learning technique. It does not require any supervision during learning, so its called self organizing maps. It is also called feature map.
- SOM learn on their own through unsupervised competitive learning.
- Self Organizing map are the neural networks that uses unsupervised learning approach and trained its network through a competitive learning algorithm to map multidimensional data into lower dimensional data, which allows easy interpretation of the problems.

- Self Organizing maps have two layers.
- 1) Input Layer 2) Output Layer
- There are n units in the input layer and m units in the output layer.
- SOM Training Algorithm




Feature map
Weight matrix
x_0 x_1 x_2 x_3
Input vector

Self Organizing Maps(SOM) Algorithm

1) Initialization- Choose random values for the initial weights wij.

2) Sampling- Take a sample training input vector for the input layer (x1, x2, x3……xn)

3) Matching- Find the winning neuron from the output layer that has weight vector closest to the input vector. It can be calculated taking the square of the Euclidean distance for each output unit and find the output unit that has minimum Euclidean distance from the input vector. That output unit is called the winning unit ie maximum features of input are matching to that output unit.

$$D(j) = \sum_{i=1}^{n} \sum_{j=1}^{m} (x_i - w_{ij})^2$$

$$D(j) = \sum_{i=1}^{n} \sum_{j=1}^{m} (x_i - w_{ij})^2$$

- **$D(1)=(x_1 - w_{11})^2+(x_2-w_{21})^2+(x_3- w_{31})^2+\ldots\ldots..+(x_n-w_{n1})^2$**
- **$D(2)=(x_1 - w_{12})^2+(x_2-w_{22})^2+(x_3- w_{32})^2+\ldots\ldots..+(x_n-w_{n2})^2$**
- **$D(m)=(x_1 - w_{1m})^2+(x_2-w_{2m})^2+(x_3- w_{3m})^2+\ldots\ldots..+(x_n-w_{nm})^2$**
- If D(2) is minimum, it means Y(2) is closest to the present input vector.

4) New Weight Calculation- Find the new weights between input vector sample and winning output unit or neuron.

$$w_{ij}(new) = w_{ij}(old) + \alpha[x_i - w_{ij}(old)]$$

$$w_{ij}(new) = w_{ij}(old) + \alpha[x_i - w_{ij}(old)]$$

If D(2) is minimum then w12, w22, w32…..wn2 are modified.

Here ***α*** is a constant learning rate whose value lies between 0 to 1.

- w_{12}(new)= w_{12}(old) + ***α*** (x_1-w_{12}(old))
- w_{22}(new)= w_{22}(old) + ***α*** (x_2-w_{22}(old))
- w_{32}(new)= w_{32}(old) + ***α*** (x_3-w_{32}(old))

5) Repetition- Repeat step 2 to step 4 until weight updation is negligible or new weights are similar to the old weights. When new weights stop changing, it means features are mapped with the input vector.

- After feature mapping two layers can be combined to have single layer ie higher dimensional data is converted to lower dimensional data. So dimensions can be decreased by mapping the features.

Deep Learning

- Deep learning is just a subset of machine learning. In fact, deep learning is machine learning and functions in a similar way. However, its capabilities are different.
- While basic machine learning models do become progressively better at performing their specific functions as they take in new data, they still need some human intervention. If an AI algorithm returns an inaccurate prediction, then an engineer has to step in and make adjustments. With a deep learning model, an algorithm can determine whether or not a prediction is accurate through its own neural network—no human help is required.

Difference between Machine Learning & Deep Learning

1. Machine learning models mostly require data in a structured form.	1. Deep Learning models can work with structured and unstructured data both as they rely on the layers of the Artificial neural network.
2. Machine learning models are suitable for solving simple or bit-complex problems.	2. Deep learning models are suitable for solving complex problems.
3. Machine learning algorithm takes less time to train the model than deep learning, but it takes a long-time duration to test the model.	3. Deep Learning takes a long execution time to train the model, but less time to test the model.

Convolutional Neural Network

- In neural networks, Convolutional neural network (Conv Nets or CNNs) is one of the main categories to do images recognition, images classifications. Objects detections, recognition faces etc., are some of the areas where CNNs are widely used.
- A convolutional neural network is a feed-forward neural network, often with up to 20 or 30 layers. The power of a convolutional neural network comes from a special kind of layer called the convolutional layer.

- Convolutional neural networks are widely used in computer vision and have become the state of the art for many visual applications such as image classification, and have also found success in natural language processing for text classification.
- Convolutional neural networks are very good at picking up on patterns in the input image, such as lines, gradients, circles, or even eyes and faces. It is this property that makes convolutional neural networks so powerful for computer vision. Unlike earlier computer vision algorithms, convolutional neural networks can operate directly on a raw image and do not need any preprocessing.

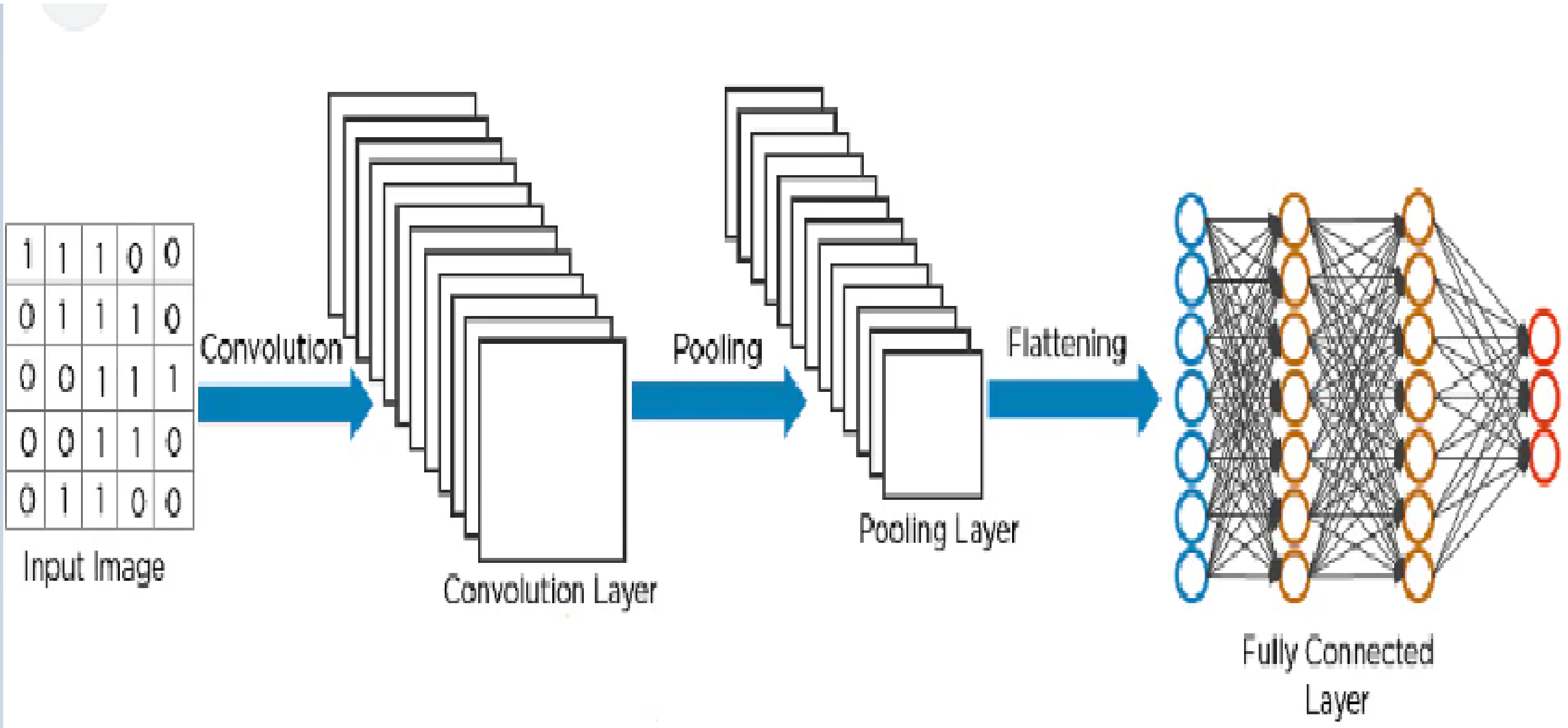
How Do Convolutional Layers Work in Deep Learning Neural Networks?

Convolutional layers are the major building blocks used in convolutional neural networks.

Convolution in Convolutional Neural Networks

- Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel.

CNN- Convolutional Neural Network



5 Steps of CNN

1. Kernel
2. Stride
3. Padding
4. Pooling
5. Flatten

Kernel

- Kernel is nothing but a filter that is used to extract the features from the images.
- The kernel is the matrix that moves over the input data, performs the product and additions with the sub region of input data and get the new matrix.
- Size of the output matrix= N-K+1

where N is the size of input matrix & K is the size of kernel or filter

- Different filter kernels are applied to extract different features. Kernel weights play very important role in finding particular feature.

Stride

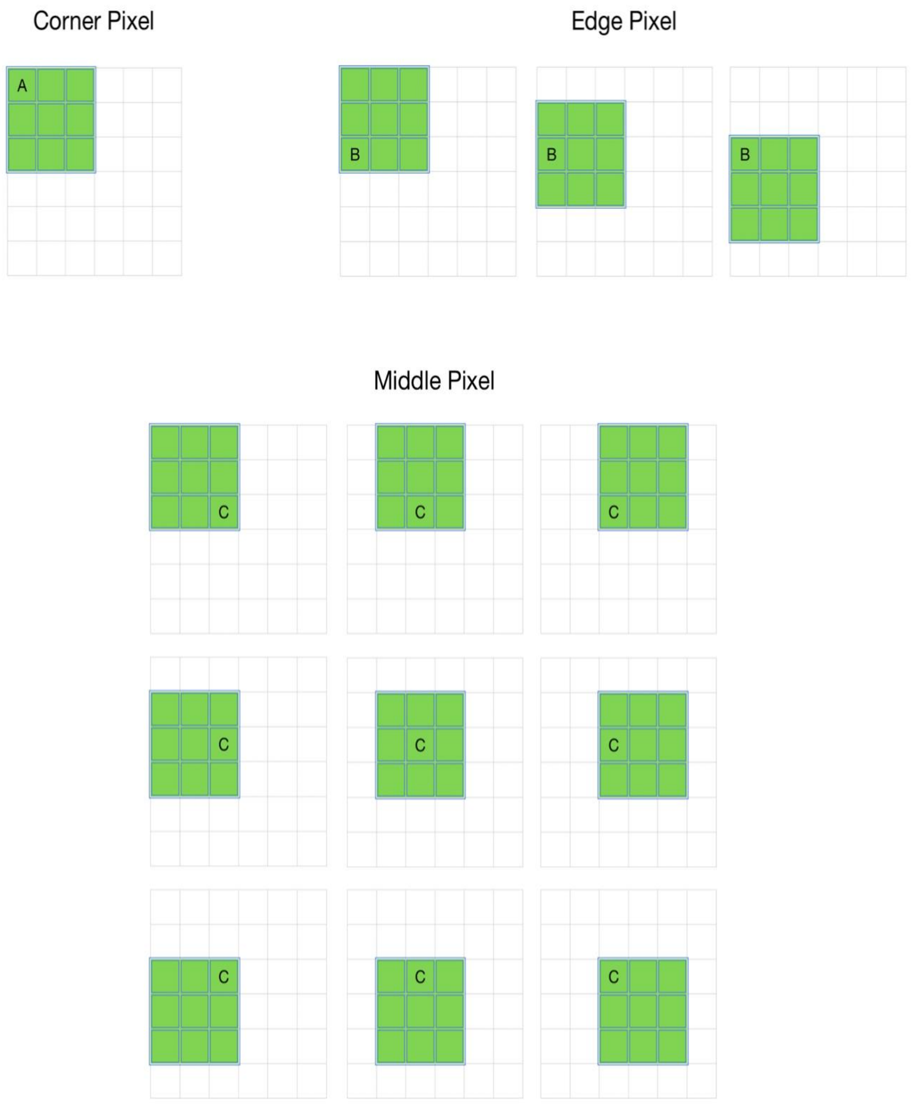
- The filter is moved across the image left to right, top to bottom with one pixel column change on the horizontal movements then a one pixel row change on the vertical movements.
- The amount of movement between applications of the filter to the input image is referred to as the stride and it is always symmetrical in height and width dimensions.
- The default stride in two dimensions is (1,1) for the height and the width movement.
- Size of the output matrix if value of stride (S) is is greater than one is (N-K)/S+1

where N is the size of input matrix, K is the size of kernel or filter & S is size of stride.

- Stride is a component of convolutional neural networks, or neural networks tuned **for the compression of images and video data**.

Padding

- For an (8 x 8) image and (3 x 3) filter, the output resulting after convolution operation would be of size (6 x 6). Thus, the image shrinks every time a convolution operation is performed. This places an upper limit to the number of times such an operation could be performed before the image reduces to nothing thereby precluding us from building deeper networks.
- The pixels on the corners and the edges are used much less than those in the middle.

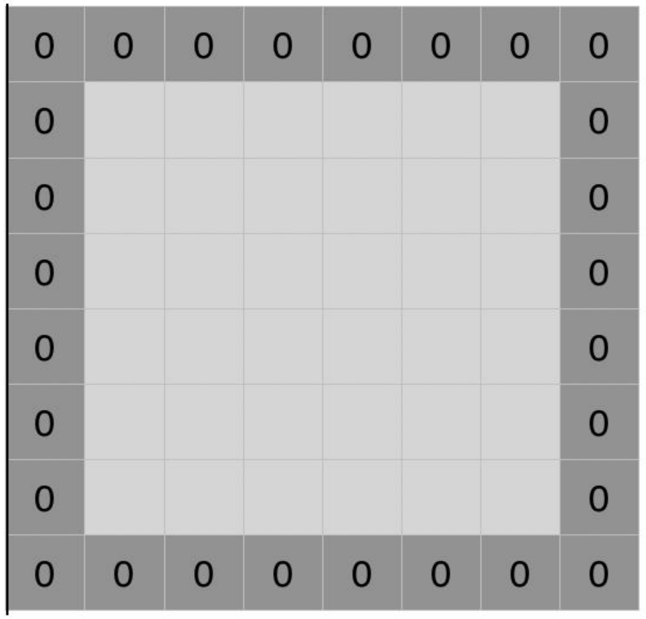


Padding

- Padding is the best approach, where the number of pixels needed for the convolution kernel to process the edge, pixels are added on to the outside, copying the pixels from the edge of the image.
- Fix the border effect problem with padding.
- Padding is simply a process of adding layers of zeros to our input images
- Size of Output matrix

$$= \frac{N-K+2P}{S} + 1;$$

Where P is size of padding.



Zero-padding added to image

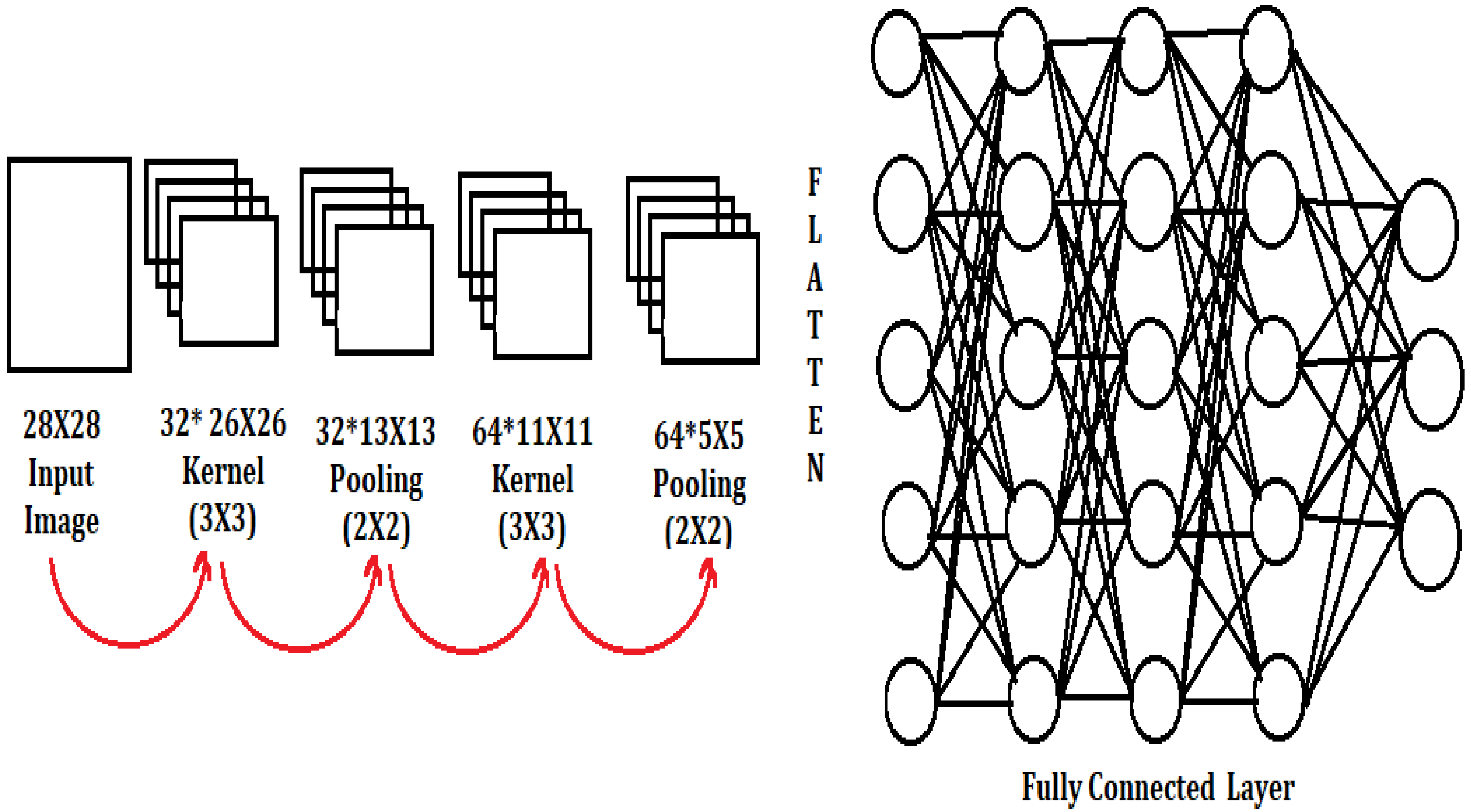
Pooling

- Pooling is required to down sample the detection of feature maps.
- Pooling layers provide an approach to down sampling feature maps by summarizing the presence of features in patches of the feature map.
- Two common pooling methods are Max Pooling and Average Pooling.
- Max Pooling summarizes most activated presence of the feature and Average Pooling summarizes the average presence of the feature.

Flattening

- Once the pooled feature map is obtained, the next step is to flatten it.
- It involves transforming the entire pooled feature matrix into single column which is then fed to the neural network for processing.

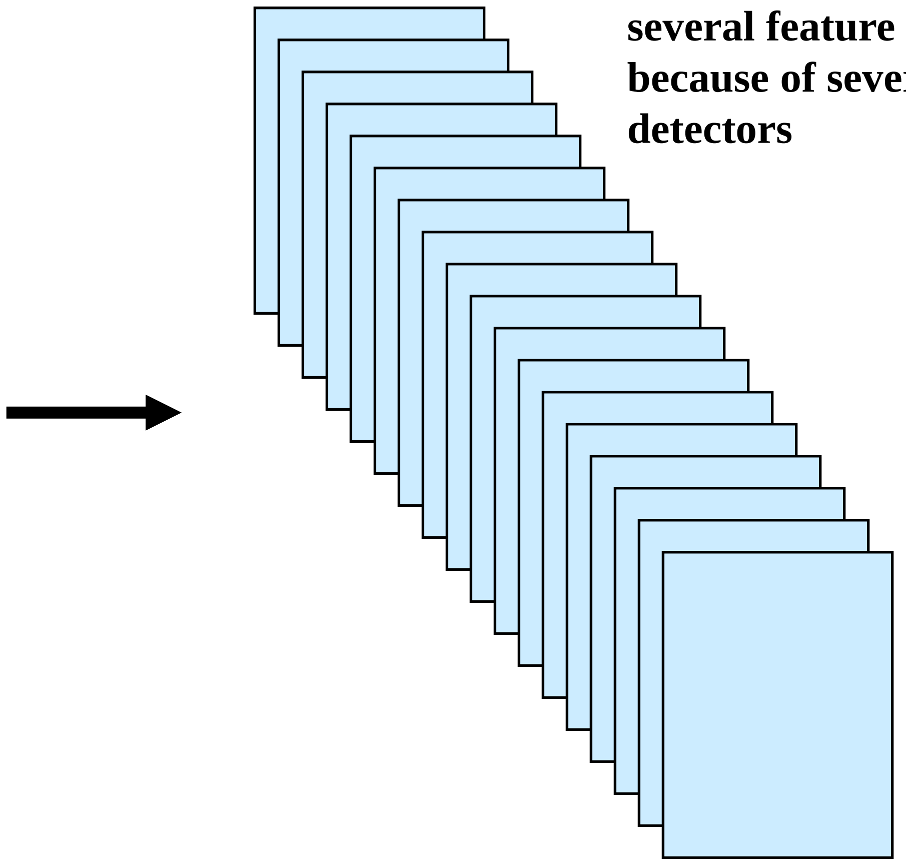
CNN Building Block



CNN Example

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

image

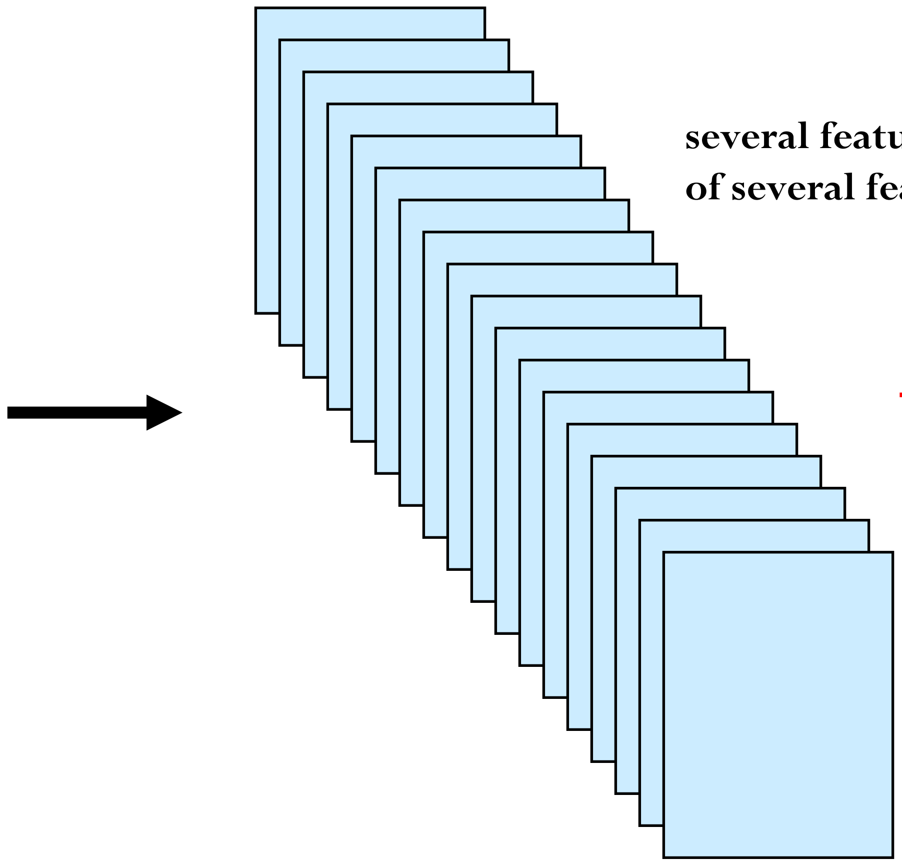


several feature maps because of several feature detectors

CONVOLUTIONAL NEURAL NETWORKS

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

image

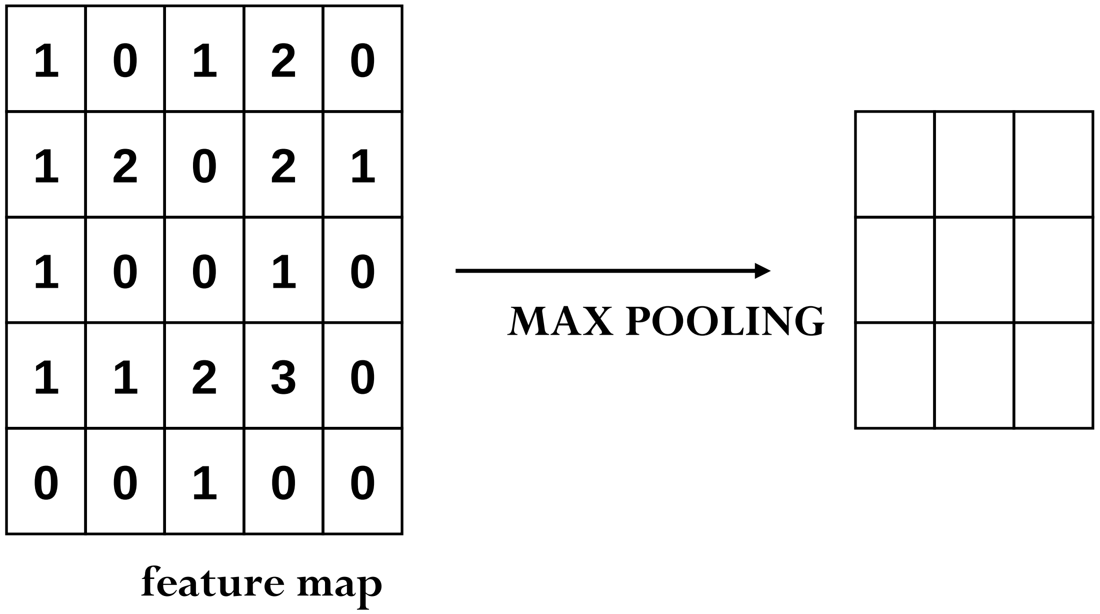


several feature maps becau
of several feature detectors

+ReLU

CONVOLUTIONAL NEURAL NETWORKS

With **max pooling** we select the most relevant features: this is how we deal with spatial invariance. We just care about the most relevant features !!!

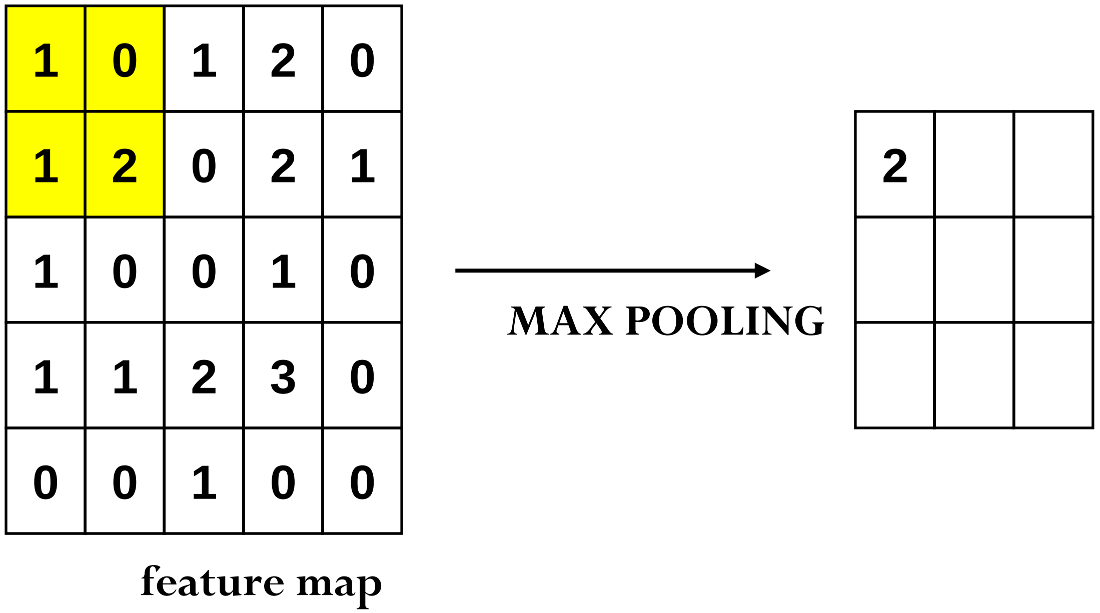


~ we can reduce the dimension of the image in order to end with a dataset containing the important pixel values without the unnecessary noise

+ we reduce number of parameters: reduce overfitting !!!

<u>CONVOLUTIONAL NEURAL NETWORKS</u>

With **max pooling** we select the most relevant features: this is how we deal with spatial invariance. We just care about the most relevant features !!!

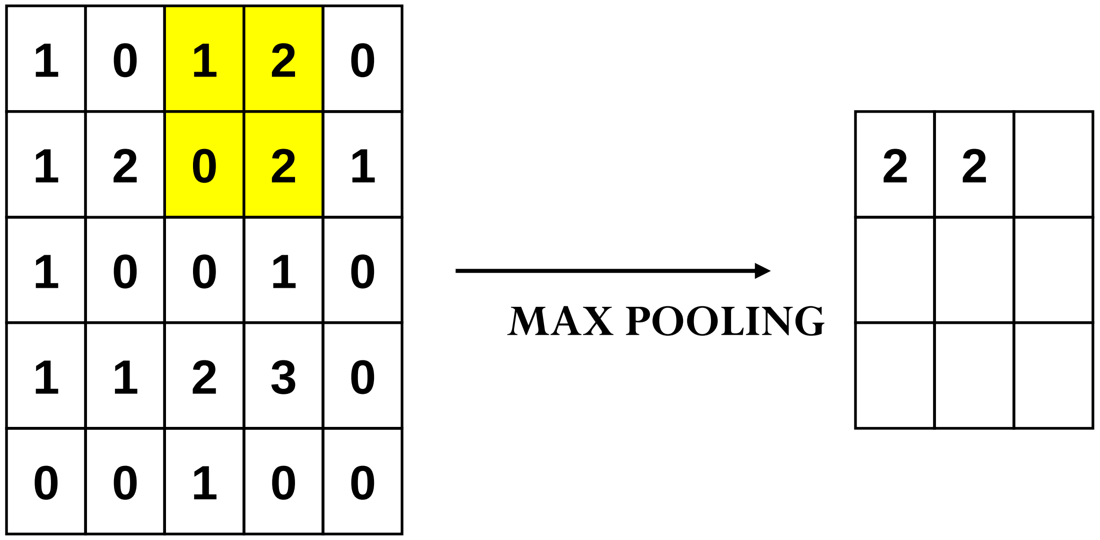


~ we can reduce the dimension of the image in order to end with a dataset containing the important pixel values without the unnecessary noise

+ we reduce number of parameters: reduce overfitting !!!

CONVOLUTIONAL NEURAL NETWORKS

With **max pooling** we select the most relevant features: this is how we deal with spatial invariance. We just care about the most relevant features !!!



1	0	1	2	0
1	2	0	2	1
1	0	0	1	0
1	1	2	3	0
0	0	1	0	0

2	2	

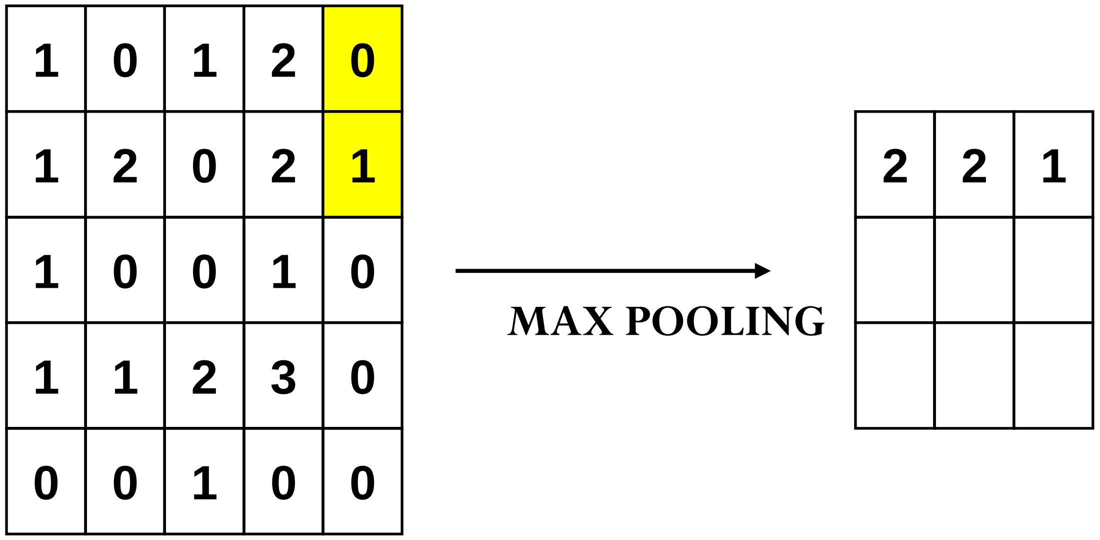
feature map

~ we can reduce the dimension of the image in order to end with a dataset containing the important pixel values without the unnecessary noise

+ we reduce number of parameters: reduce overfitting !!!

CONVOLUTIONAL NEURAL NETWORKS

With **max pooling** we select the most relevant features: this is how we deal with spatial invariance. We just care about the most relevant features !!!



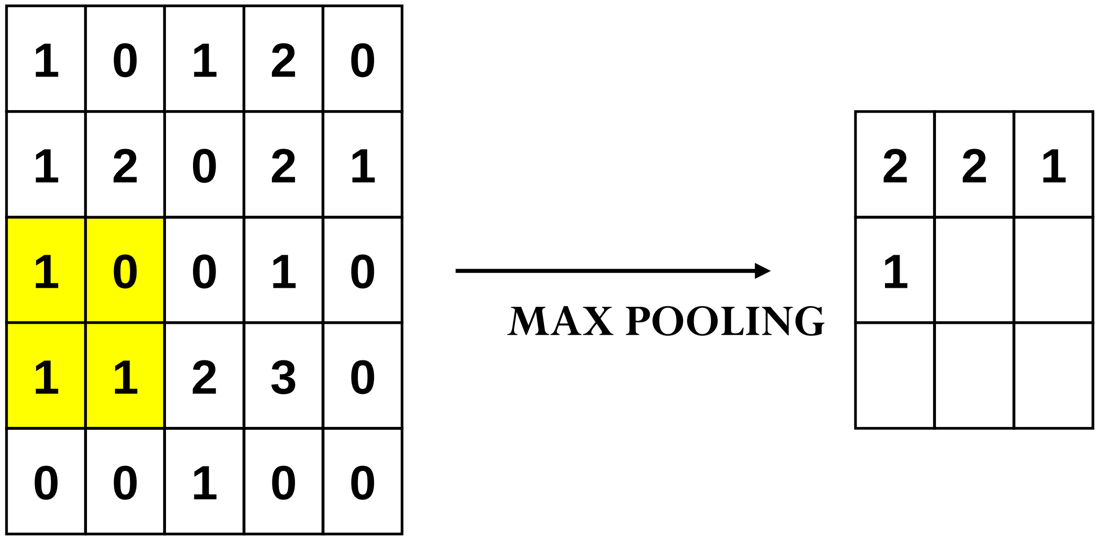
feature map

~ we can reduce the dimension of the image in order to end with a dataset containing the important pixel values without the unnecessary noise

+ we reduce number of parameters: reduce overfitting !!!

CONVOLUTIONAL NEURAL NETWORKS

With **max pooling** we select the most relevant features: this is how we deal with spatial invariance. We just care about the most relevant features !!!



1	0	1	2	0
1	2	0	2	1
1	0	0	1	0
1	1	2	3	0
0	0	1	0	0

2	2	1
1		

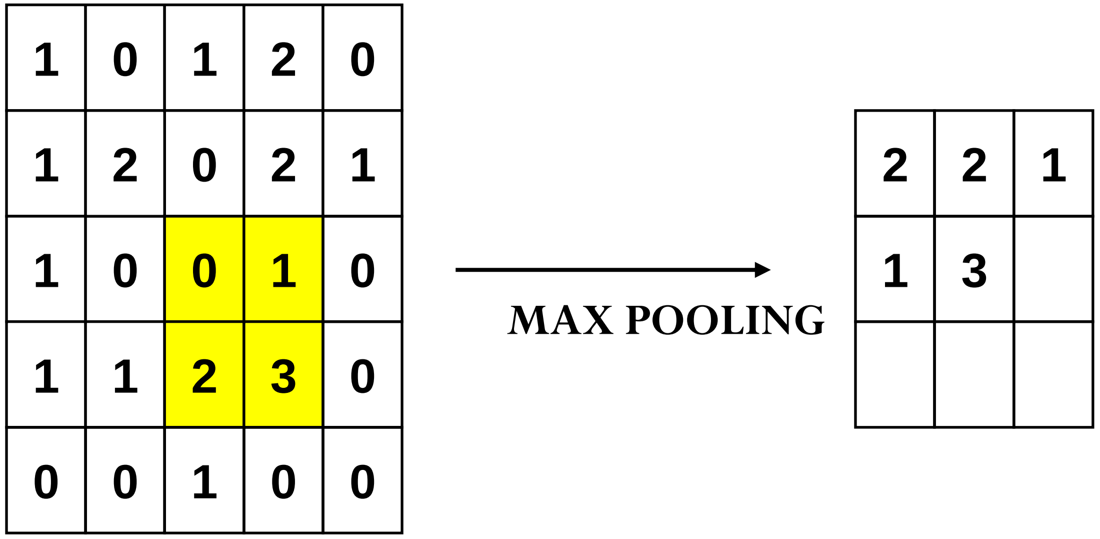
feature map

~ we can reduce the dimension of the image in order to end with a dataset containing the important pixel values without the unnecessary noise

+ we reduce number of parameters: reduce overfitting !!!

CONVOLUTIONAL NEURAL NETWORKS

With **max pooling** we select the most relevant features: this is how we deal with spatial invariance. We just care about the most relevant features !!!



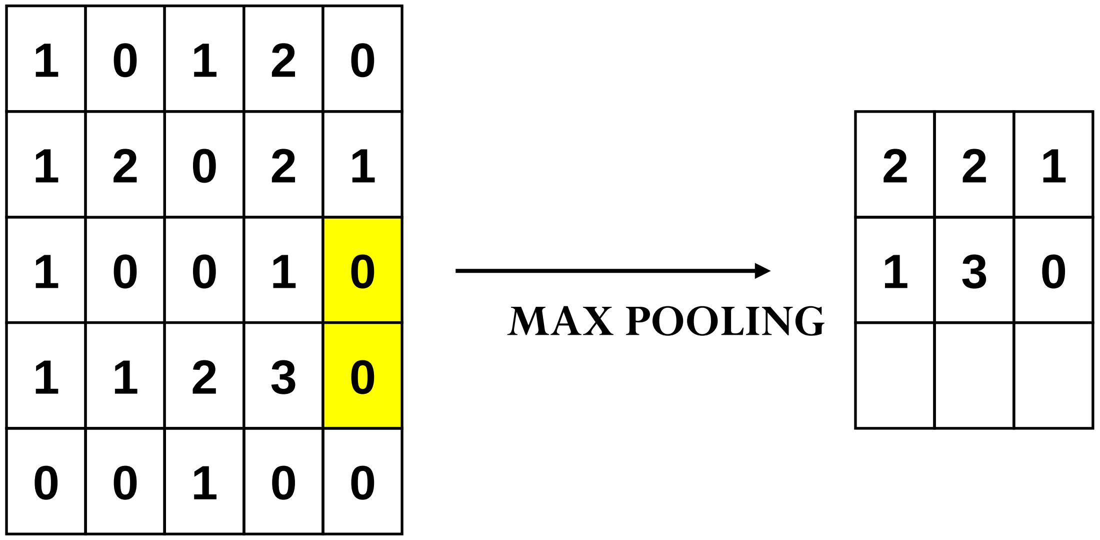
feature map

~ we can reduce the dimension of the image in order to end with a dataset containing the important pixel values without the unnecessary noise

+ we reduce number of parameters: reduce overfitting !!!

CONVOLUTIONAL NEURAL NETWORKS

With **max pooling** we select the most relevant features: this is how we deal with spatial invariance. We just care about the most relevant features !!!



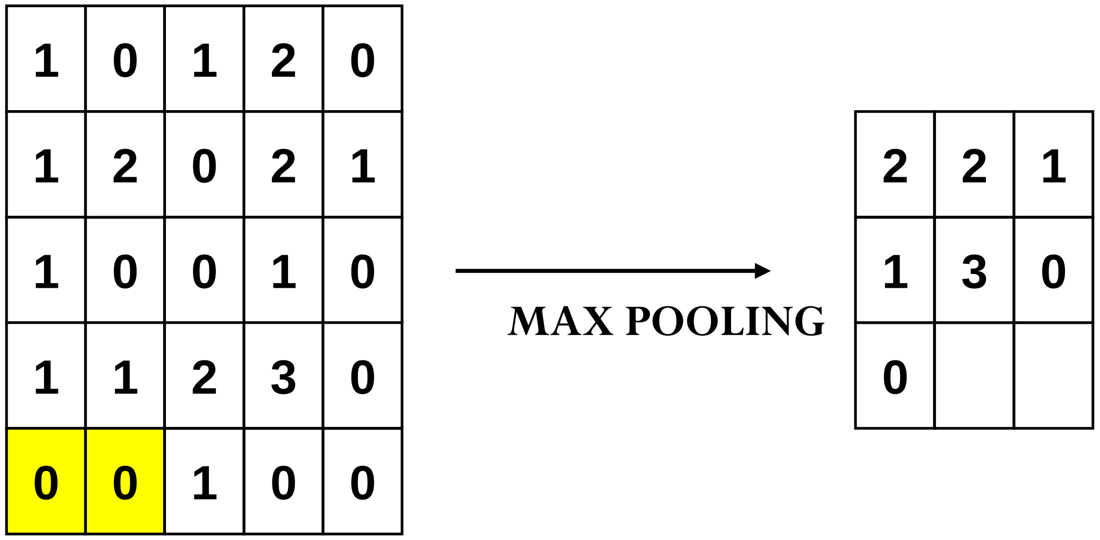
feature map

~ we can reduce the dimension of the image in order to end with a dataset containing the important pixel values without the unnecessary noise

+ we reduce number of parameters: reduce overfitting !!!

CONVOLUTIONAL NEURAL NETWORKS

With **max pooling** we select the most relevant features: this is how we deal with spatial invariance. We just care about the most relevant features !!!



1	0	1	2	0
1	2	0	2	1
1	0	0	1	0
1	1	2	3	0
0	0	1	0	0

feature map

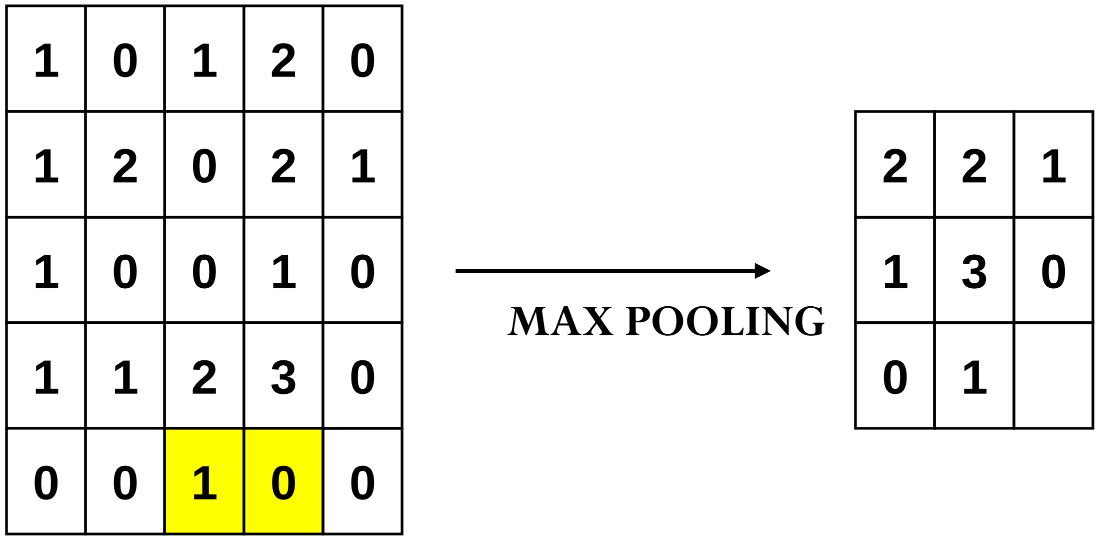
2	2	1
1	3	0
0		

~ we can reduce the dimension of the image in order to end with a dataset containing the important pixel values without the unnecessary noise

+ we reduce number of parameters: reduce overfitting !!!

CONVOLUTIONAL NEURAL NETWORKS

With **max pooling** we select the most relevant features: this is how we deal with spatial invariance. We just care about the most relevant features !!!



1	0	1	2	0
1	2	0	2	1
1	0	0	1	0
1	1	2	3	0
0	0	1	0	0

feature map

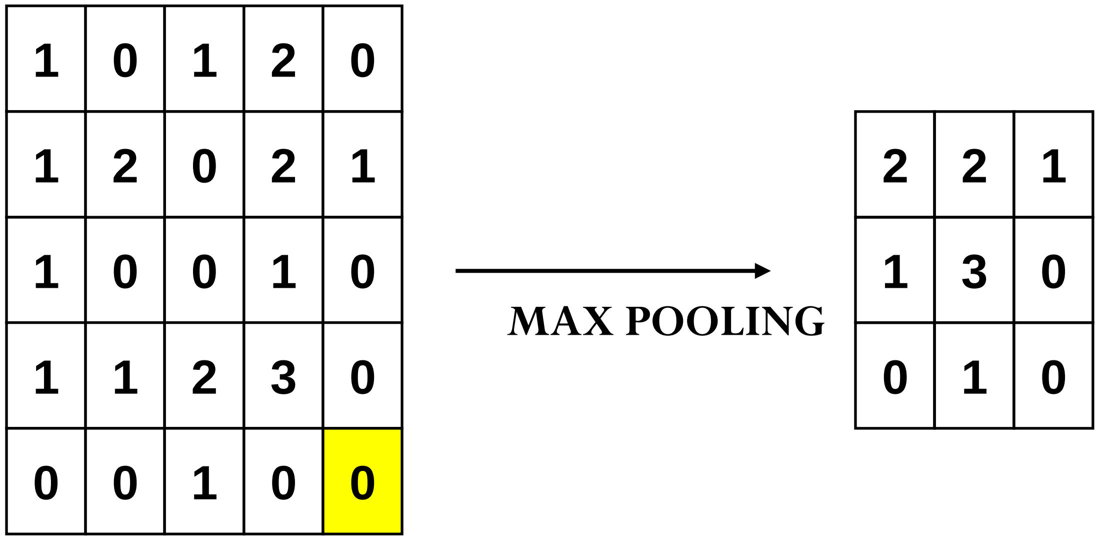
2	2	1
1	3	0
0	1	

~ we can reduce the dimension of the image in order to end with a dataset containing the important pixel values without the unnecessary noise

+ we reduce number of parameters: reduce overfitting !!!

CONVOLUTIONAL NEURAL NETWORKS

With **max pooling** we select the most relevant features: this is how we deal with spatial invariance. We just care about the most relevant features !!!



1	0	1	2	0
1	2	0	2	1
1	0	0	1	0
1	1	2	3	0
0	0	1	0	🟨 0

feature map

MAX POOLING

2	2	1
1	3	0
0	1	0

~ we can reduce the dimension of the image in order to end with a dataset containing the important pixel values without the unnecessary noise

+ we reduce number of parameters: reduce overfitting !!!

CONVOLUTIONAL NEURAL NETWORKS

With **max pooling** we select the most relevant features: this is how we deal with spatial invariance. We just care about the most relevant features !!!

1	0	1	2	0
1	2	0	2	1
1	0	0	1	0
1	1	2	3	0
0	0	1	0	0

feature map

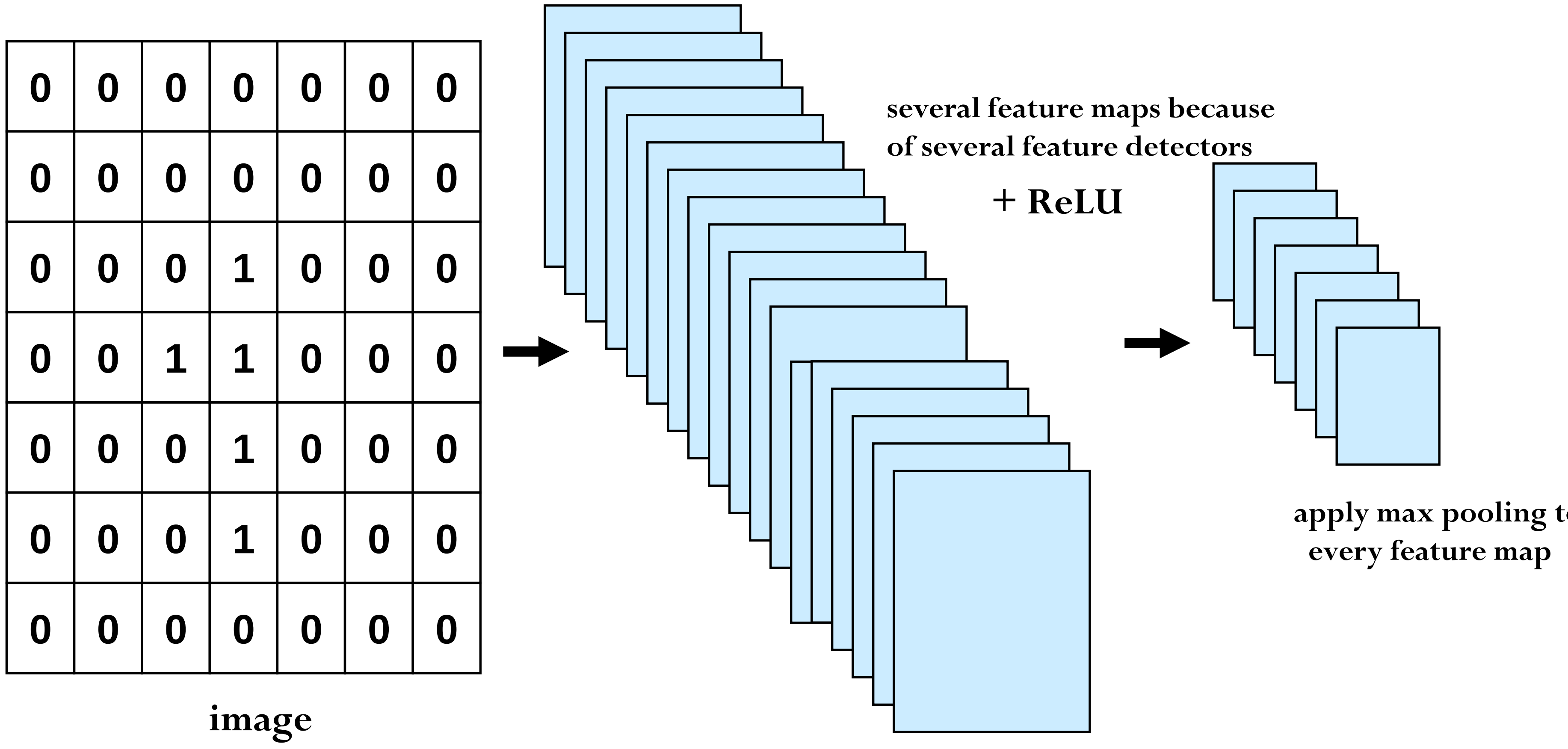
$\longrightarrow$ **MAX POOLING**

2	2	1
1	3	0
0	1	0

There are other techniques: **average pooling** is popular as well ~ instead of choosing the maximum value we calculate the average of the values present in the subset

CONVOLUTIONAL NEURAL NETWORKS

With **max pooling** we select the most relevant features: this is how we deal with spatial invariance. We just care about the most relevant features !!!

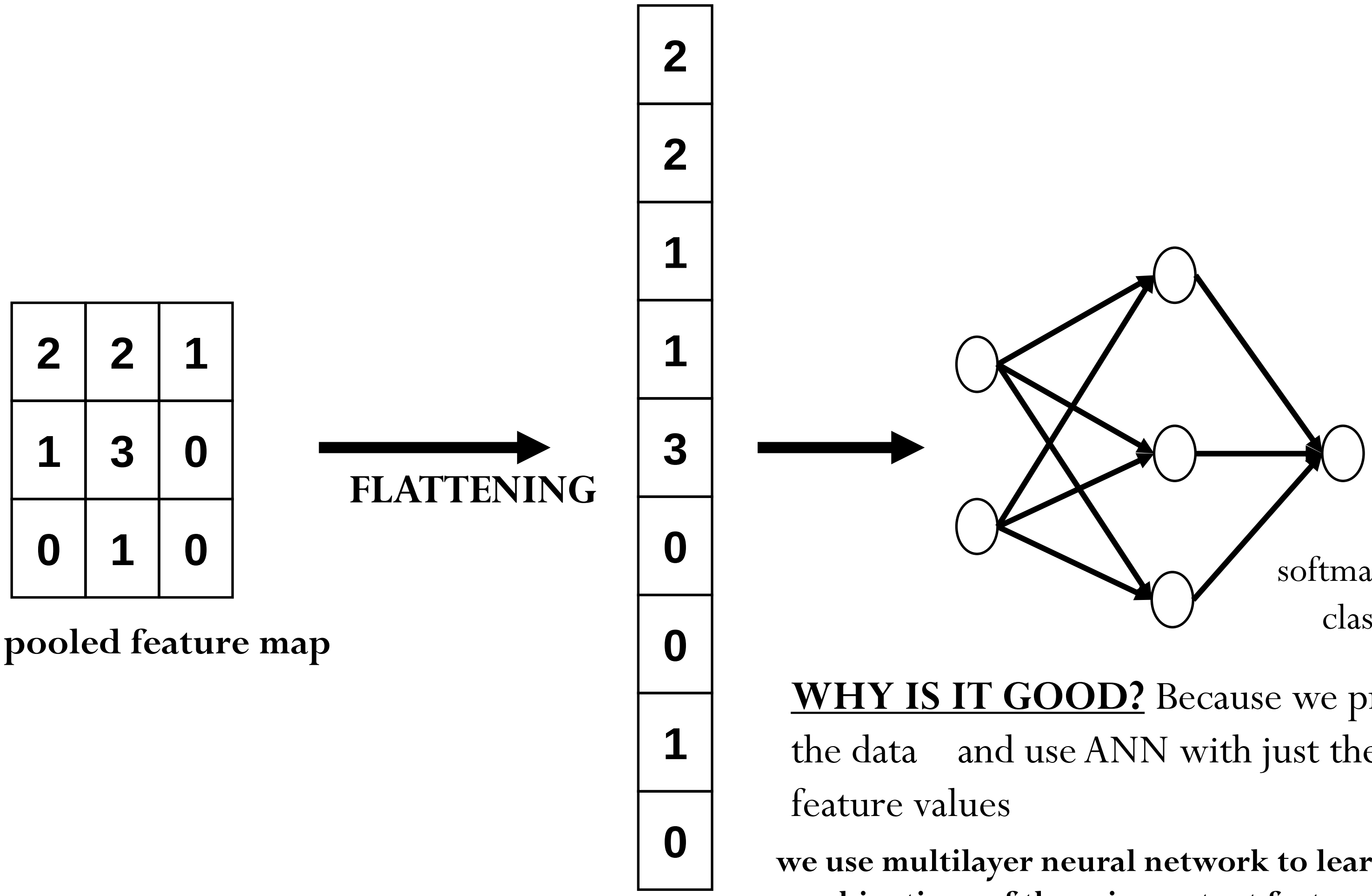


0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	1	0	0	0
0	0	0	1	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0

image

CONVOLUTIONAL NEURAL NETWORKS

The last operation we have to make is the **flattening** procedure: we transform the matrix into a one-dimensional vector: this is the input of a standard densely connected neural network



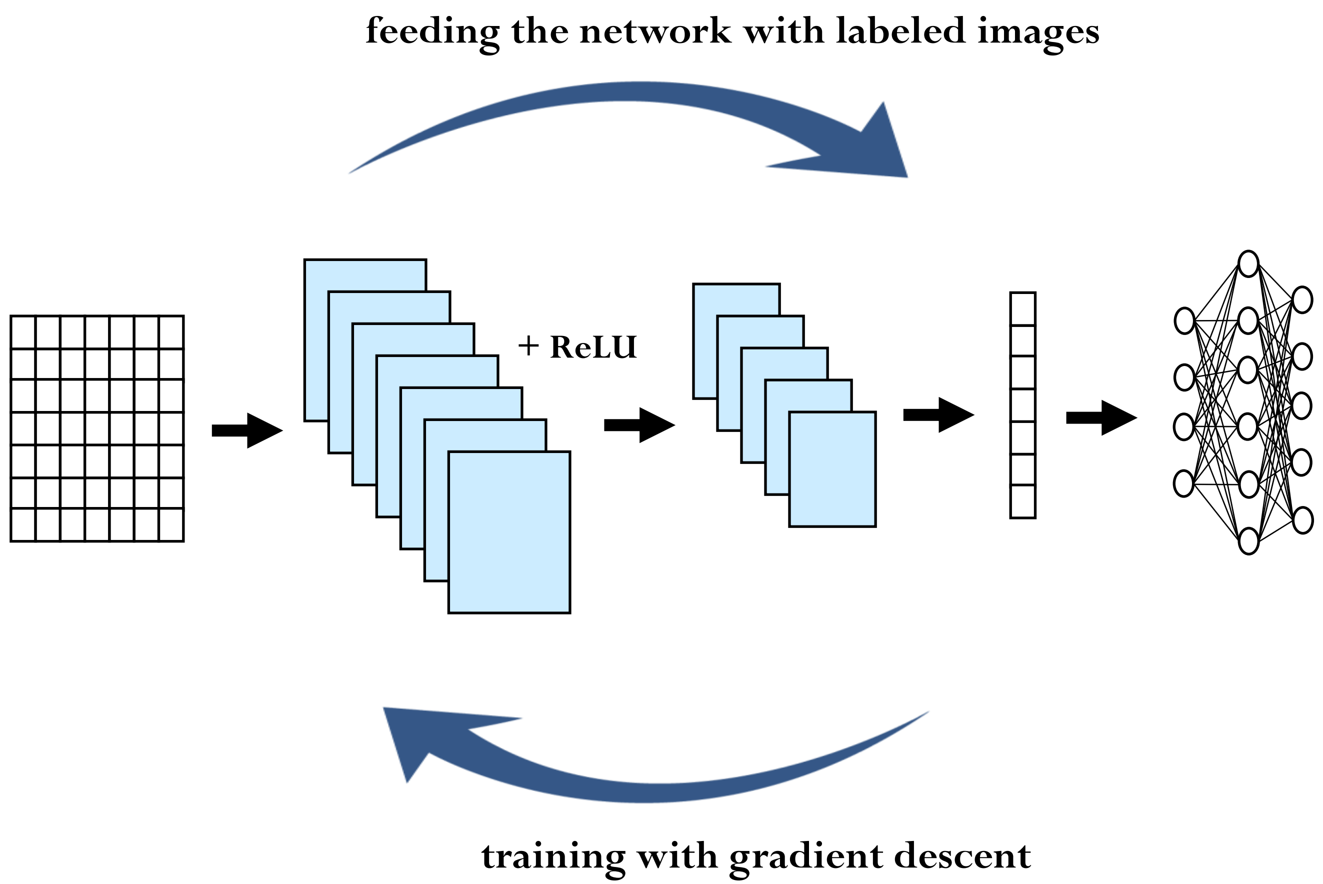
WHY IS IT GOOD? Because we preprocess the data and use ANN with just the relevant feature values

we use multilayer neural network to learn the non-linear combinations of these important features

CONVOLUTIONAL NEURAL NETWORKS- TRAINING

We use gradient descent (backpropagation) as usual as far as training the convolutional neural networks are concerned

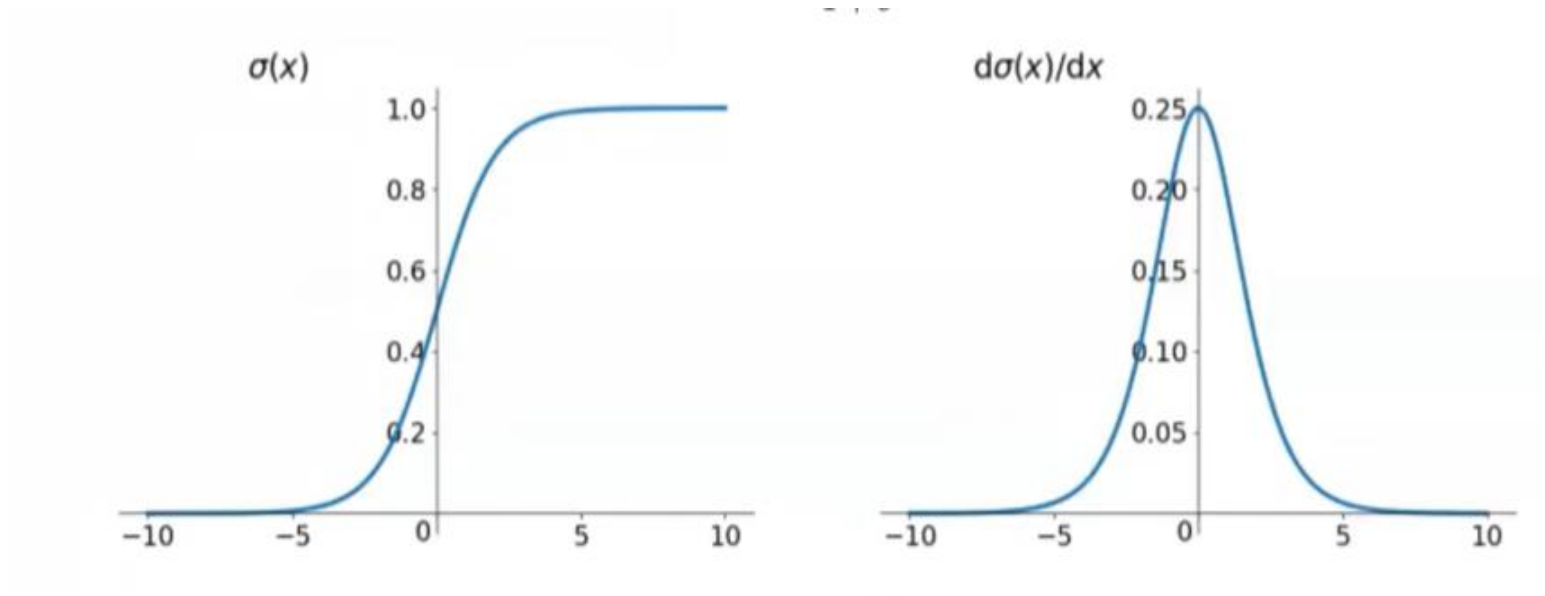
~ update the edge weights according to the error + choosing the right filter !!!


feeding the network with labeled images
+ ReLU
training with gradient descent

Activation Function

- **An Activation Function** decides whether a neuron should be activated or not. This means that it will decide whether the neuron’s input to the network is important or not in the process of prediction using simpler mathematical operations.
- The role of the Activation Function is to derive output from a set of input values fed to a node (or a layer).
- Activation Functions are used to introduce non-linearity in the network.
- A neural network will almost always have the same activation function in all hidden layers. This activation function should be differentiable so that the parameters of the network are learned in backpropagation.

Sigmoid



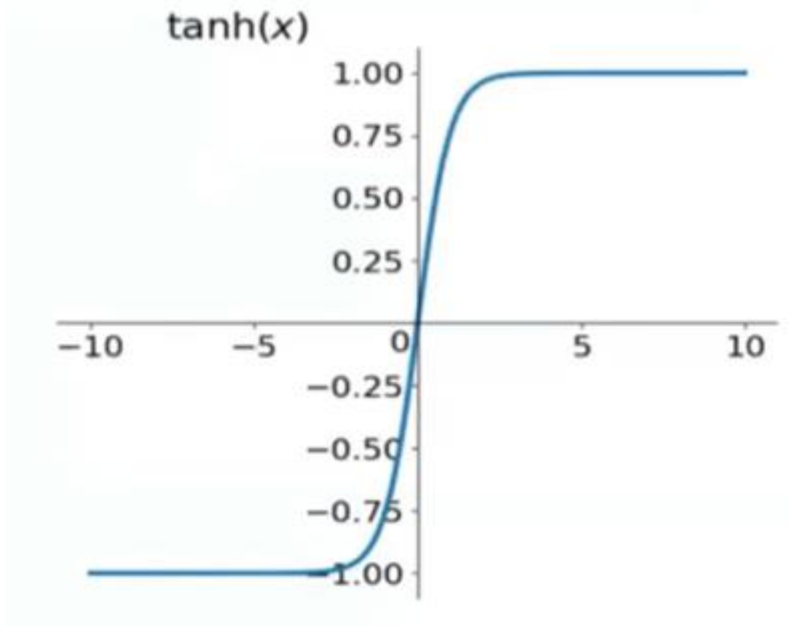
The Sigmoid function performs the role of an activation function in machine learning which is used to add non-linearity in a machine learning model. It is a mathematical function that has a characteristic that can take any real value and map it to between 0 to 1. The sigmoid function is also known as a logistic function and given as :

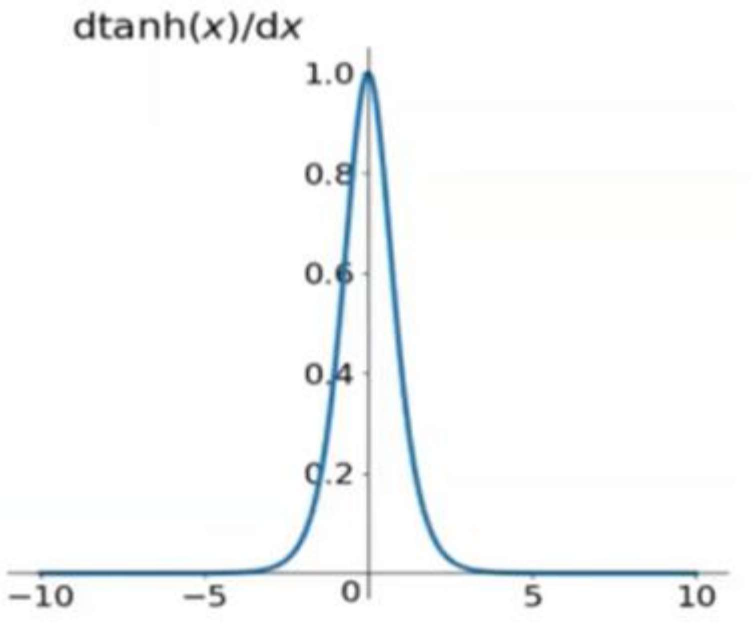
$$\boldsymbol{\sigma(x) = \frac{1}{1+e^{-x}}}$$

Tanh

Tanh function is very similar to the sigmoid/logistic activation function, and even has the same S-shape with the difference in output range of -1 to 1. It is given as:

$$\boldsymbol{Tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}}$$



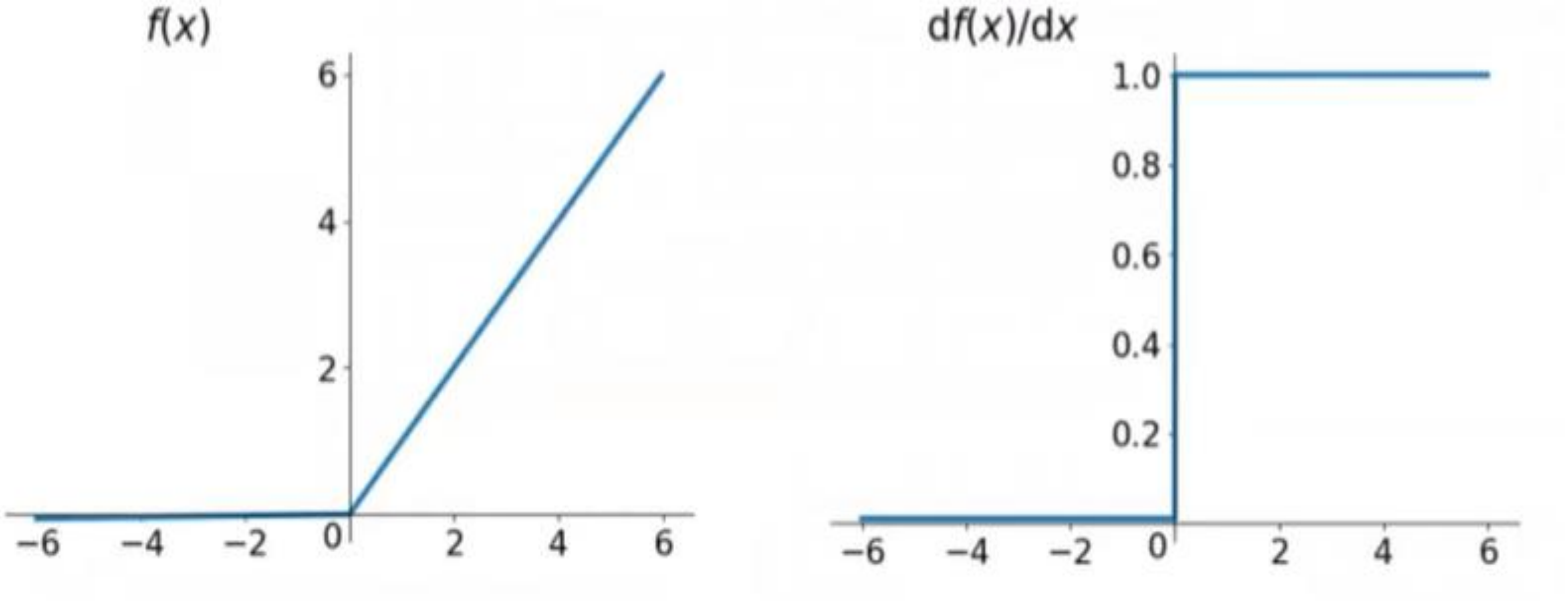


In sigmoid and tanh, when we do the backpropagation then weights are updated according to the derivative of activation function. But derivative of sigmoid and tanh is very small and zero for maximum values, so weights are changed by very small value or does not change and algorithm does not converge. So ReLU function is used.

ReLU Function

Rectified Linear Unit (ReLU) transform function only activates a node if the input is above a certain quantity, while the input is below zero, the output is zero, but when the input rises above a certain threshold, it has a linear relationship with the dependent variable

$$f(x) = \begin{cases} 0 \text{ if } x < 0 \\ x \text{ if } x >= 0 \end{cases}$$



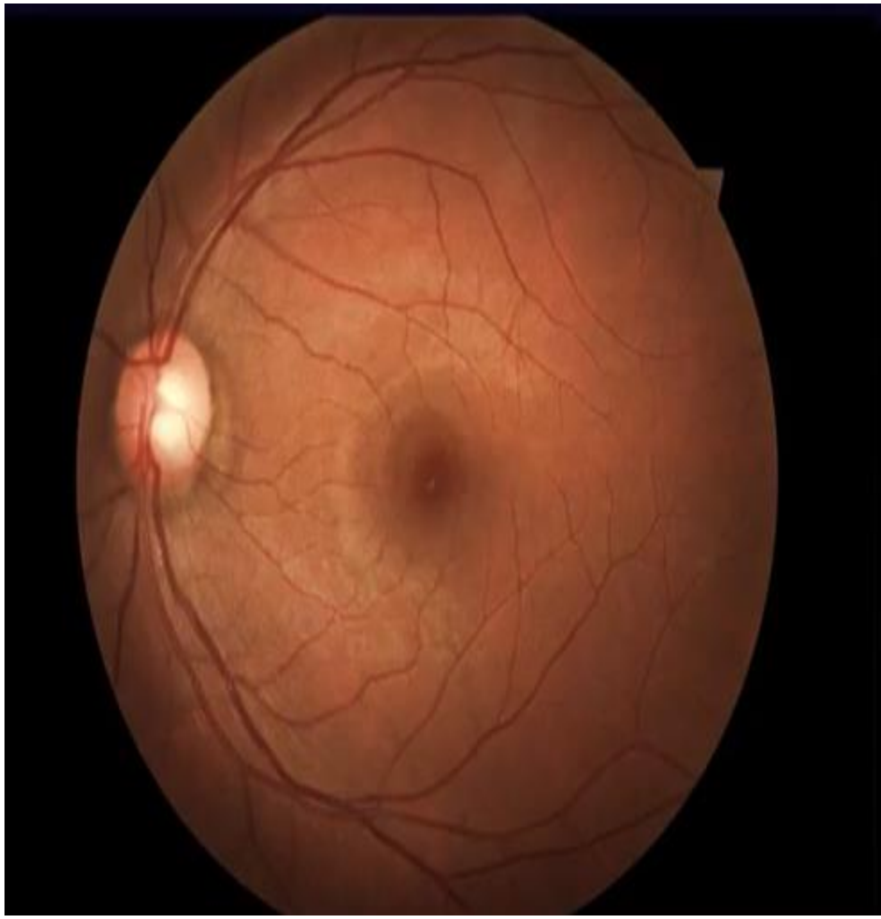
How Do Convolutional Neural Networks Work with Example?

- There are **four** layered **concepts** we should understand in Convolutional Neural Networks:
  - Convolution,
  - ReLu,
  - Pooling and
  - Full Connectedness (Fully Connected Layer).

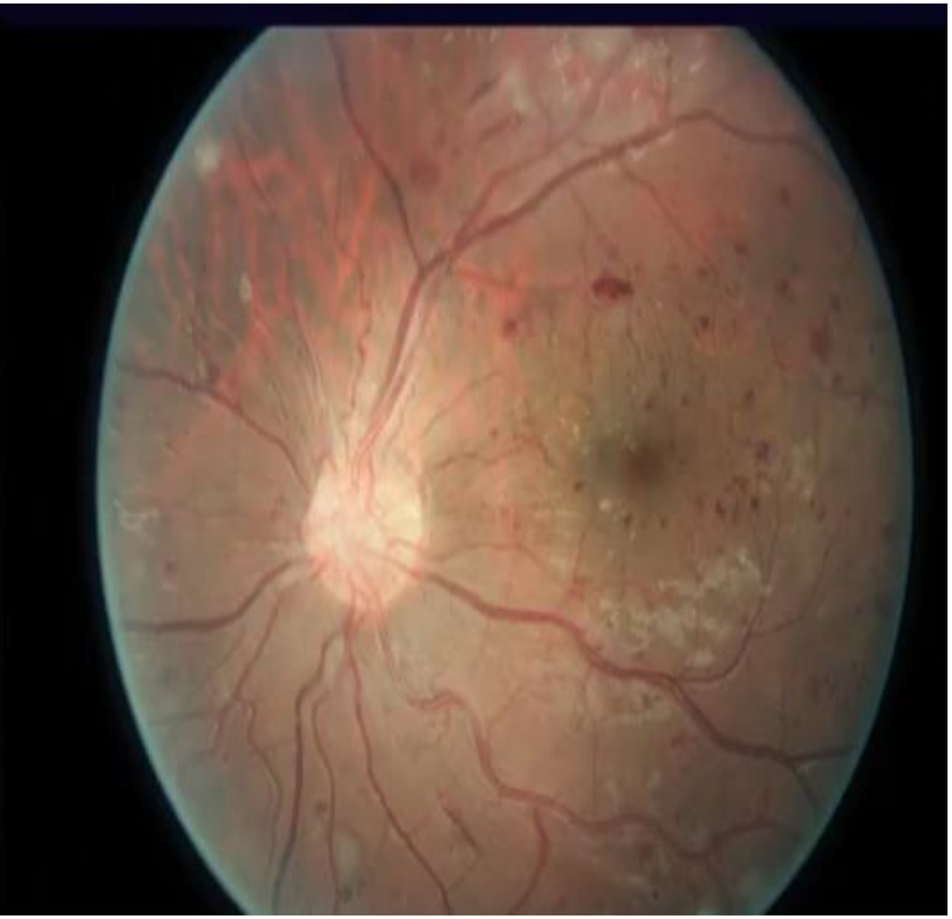
Case Studies

1. **Diabetic Retinopathy**

- It is an eye disease that affects the eyes of diabetic person.
- If blood glucose levels are left untreated then blood vessels in the eyes constricted and blood packet formed inside the eyes. It leads to retinopathy which can result in blindness.
- 93 Million people live with diabetic retinopathy worldwide.
- Current diagnosis process is very long and its difficult to analyse the images.
- So classifier using neural network can be the assistant to doctor but not the replacement.

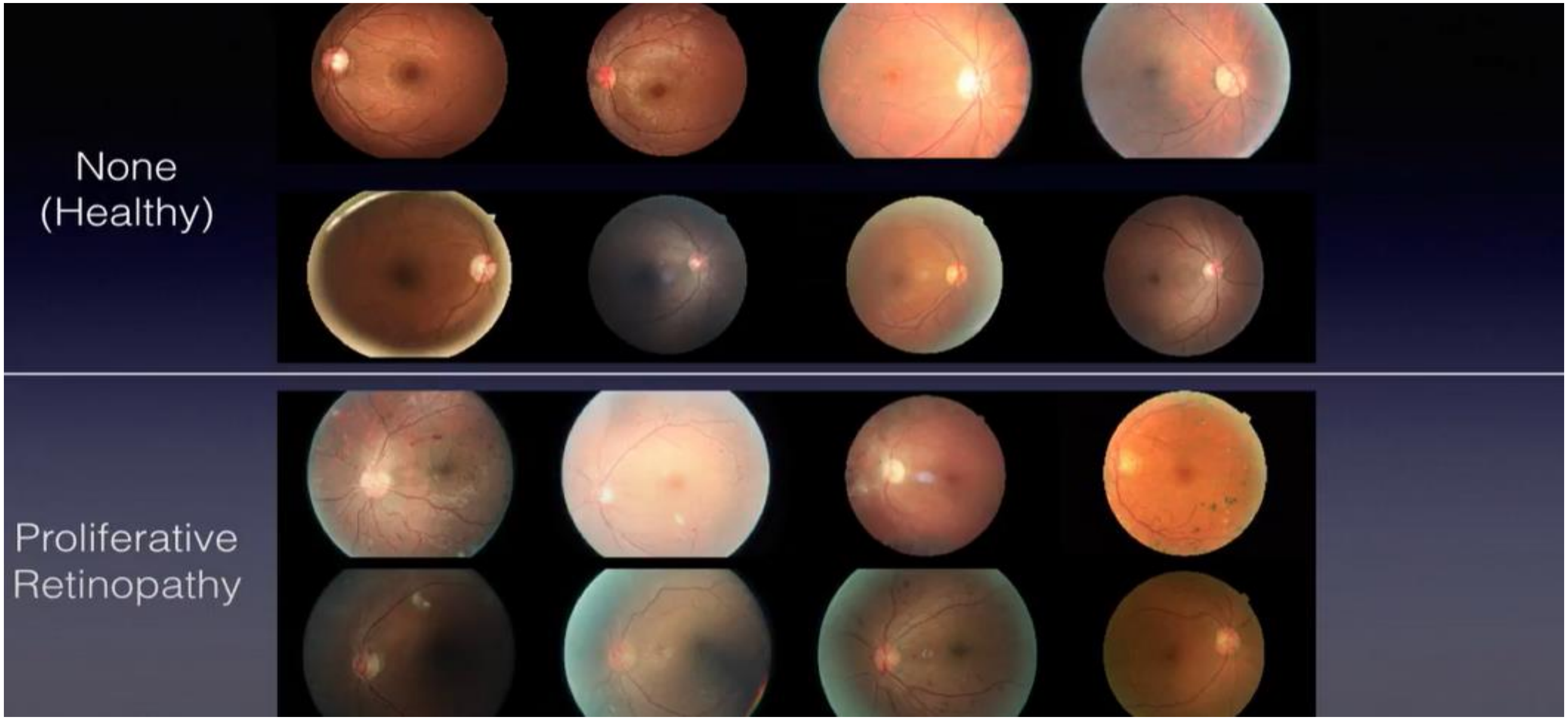


Healthy



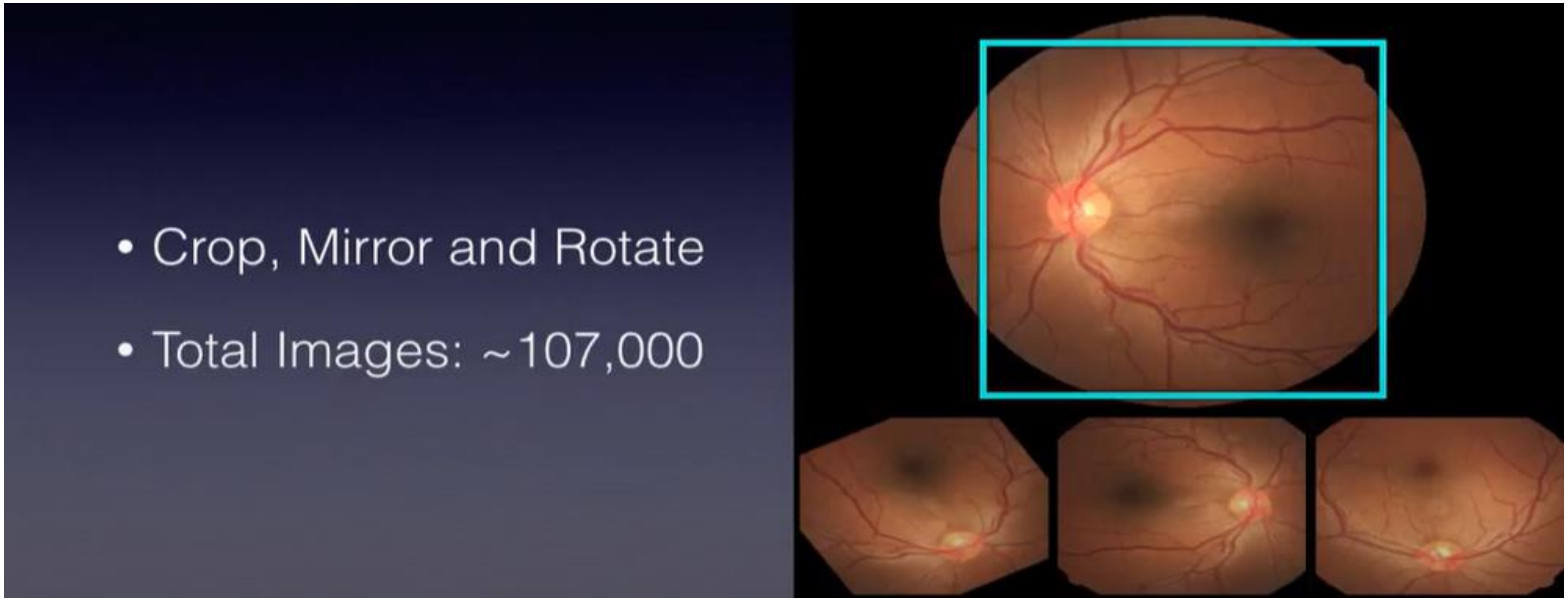
Retinopathy

- The images taken in different states are hard to classify.
- Convolutional Neural network are so effective that it can classify the images



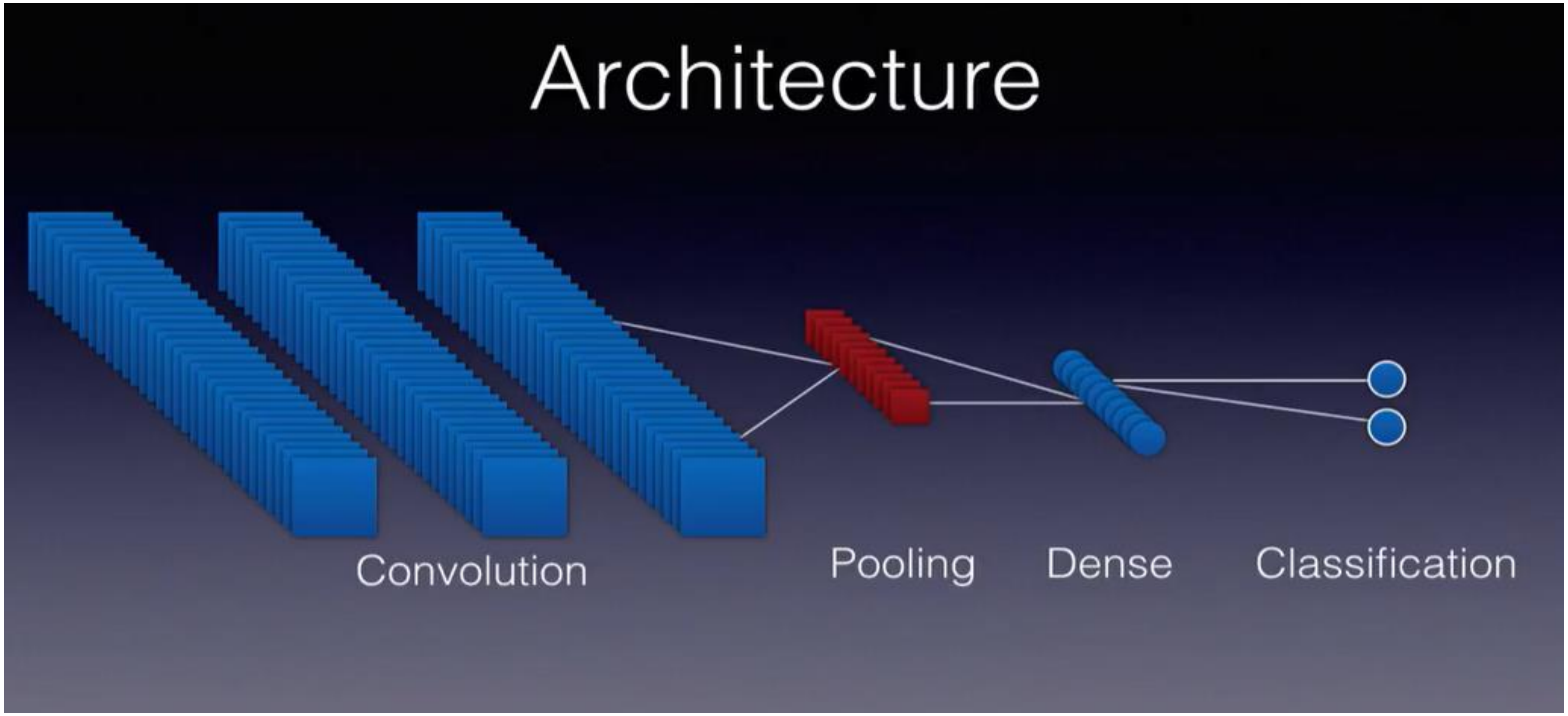
Pre processing

- Without preprocessing algorithms do not converge.
- In preprocessing images are cropped to remove noise. Then images are mirrored and rotated to have different data types of data set.
- Then the images are normalized assign 1 to the images with retinopathy and 0 to the healthy eyes.



- Crop, Mirror and Rotate
- Total Images: ~107,000

- After normalization, images are fed to convolutional neural network. After 3 layer convolution, single pooling layer data is flattened and given to dense neural network and in the result we will get whether the person is having retinopathy or not.



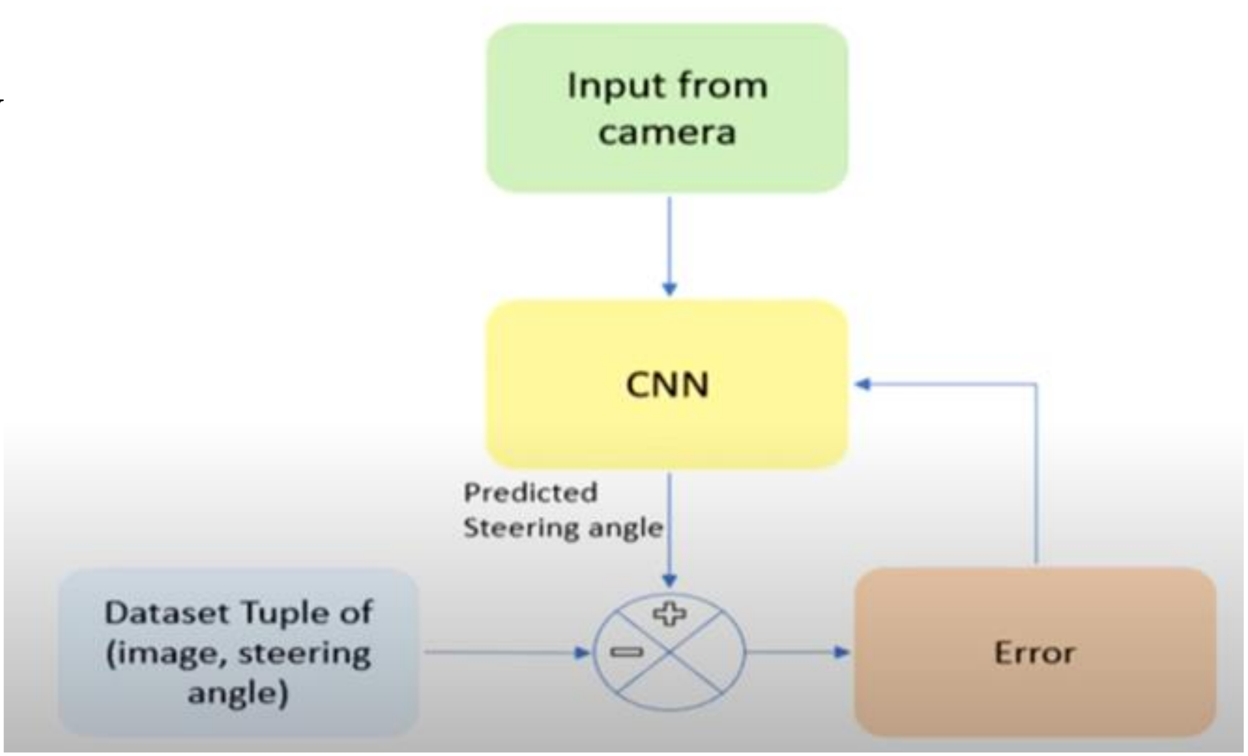
2. Self Driving Cars using CNN

- In Autonomous cars 5 levels are there depending upon the human intervention.
- In driving two parameters are there one is the steering parameter and other is the speed acceleration parameter.
- Convolutional Neural networks and Ultrasonic sensors are used in this project.

CNN Flow

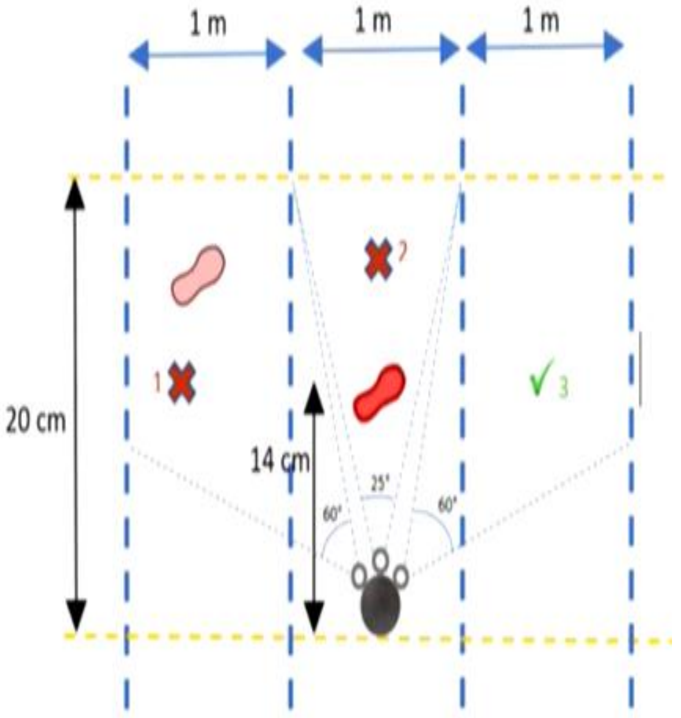
- Images taken from camera are fed to CNN, steering angle is predicted.
- Data set is present with the images and corresponding steering angles.
- Predicted steering angle compared with the steering angle given by data set. Mean square error is calculated and fed back to CNN to train it again

  for better efficiency



Working of the Model

- Two Ultrasonic Sensors placed at 60 degrees with the one in centre
- Camera: capturing images
- Raspberry pi: processing to obtain steering angle
- Input:
    - Images
    - Obstacle distance
- Output:
    - Steering angle
    - Speed of motors



3. Smart speaker system

Smart speaker

Amazon
Echo / Alexa

Google
Home

Apple
Siri

Baidu
DuerOS

"Hey device, tell me a joke"

Key steps:

1. Trigger/wakeword detection
2. Speech recognition
3. Intent recognition
4. Specialized program to execute command