Deep Learning Architecture –CS812

- **(Elective Course – 8th Semester CS&E)**

- Dr.Srinath.S, Associate Professor
- Department of Computer Science and Engineering
- SJCE, JSS S&TU, Mysuru- 570006

Pre - Requisite

- Linear Algebra
- Elementary Probability and Statistics
- Machine Learning / Pattern Recognition.
- Programming skills – Python preferred

Course Outcomes

- After completing this course, students should be able to:
  - CO1: Identify the deep learning algorithm which are more appropriate for various types of learning tasks in various domains
  - CO2: Implement deep learning algorithm and solve real world problems
  - CO3: Execute performance metrics of deep learning techniques.

Text/Reference Books/web resource/CO mapping

Text Book:

Sl. No.	Author/s	Title	Publisher Details
1	Aurelien Geron	Hands on Machine Learning with Scikit-Learn &TensorFlow	O'Reilly, 2019

Reference Books:

Sl. No.	Author/s	Title	Publisher Details
1	Lan Good fellow and Yoshua Bengio and Aaron Courville	Deep Learning	MIT Press2016
2	Charu C. Aggarwal	Neural Networks and Deep Learning	Springer International Publishing, 2018
3	Andrew W. Trask	Grokking Deep Learning	Manning Publications
4	Sudharsan Ravichandran	Hands-On Deep Learning Algorithms with Python	--

Web Resources:

Sl. No.	Web link
1	https://onlinecourses.nptel.ac.in/noc20_cs62/preview
2	https://nptel.ac.in/courses/106/105/106105215/

Course Outcomes	Program Outcomes												PSO's		
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	P012	PSO1	PSO2	PSO3
CO-1	1	2	3	1	3	3	1	3	3	3	3	3	2	3	3
CO-2	3	3	3	3	3	2	2	2	1	2	3	2	1	2	3
CO-3	3	1	2	3	1	2	2	2	3	3	3	2	3	2	3

0 -- No association 1---Low association, 2--- Moderate association, 3---High association

Assessment Weightage in Marks

Class Test –I	10
Quiz/Mini Projects/ Assignment/ seminars	10
Class Test – II	10
Quiz/Mini Projects/ Assignment/ seminars	10
Class Test – III	10
Total	50

Question Paper Pattern

- **Semester End Examination (SEE)**
- **Semester End Examination (SEE) is a written examination of three hours duration of 100 marks with 50% weightage.**

- **Note:**
- **• The question paper consists of TWO parts PART- A and PART- B.**

- **• PART- A consists of Question Number 1-5 are compulsory (ONE question from each unit)**

- **• PART-B consists of Question Number 6-15 will have internal choice. (TWO question from each unit)**

- **• Each Question carries 10 marks and may consist of sub-questions.**
- **• Answer 10 full questions of 10 marks each**

Source:

- Material is based on *Hands-On Machine Learning with Scikit_Learn and TensorFlow: Concepts, Tools and Techniques* (by Aurelien Geron), Wikipedia, and other sources.

UNIT – 1 Introduction to ANN:

- **Introduction to ANN:** Biological to Artificial neuron, Training an MLP, training a DNN with TensorFlow, Fine tuning NN Hyper Parameters Up and Running with TensorFlow

Quick look into ML

Dr.Srinath.S

MACHINE LEARNING

Dr.Shrinath.S
Associate Professor, Department of CS&E
Sri Jayachamarajendra College of Engineering,
JSS Science and Technology University, Mysuru- 570006

Introduction

- Artificial Intelligence (AI)
- Machine Learning (ML)
- Deep Learning (DL)
- Data Science

Artificial Intelligence

- Artificial intelligence is intelligence demonstrated by machines, as opposed to natural intelligence displayed by animals including humans.

Machine Learning

- Machine Learning – Statistical Tool to explore the data.

Machine learning (ML) is a type of artificial intelligence (AI) that allows software applications to become more accurate at predicting outcomes without being explicitly programmed to do so. Machine learning algorithms use historical data as input to predict new output values.

If you are searching some item in amazon… next time… without your request… your choice will be listed.

Variants of Machine Learning:

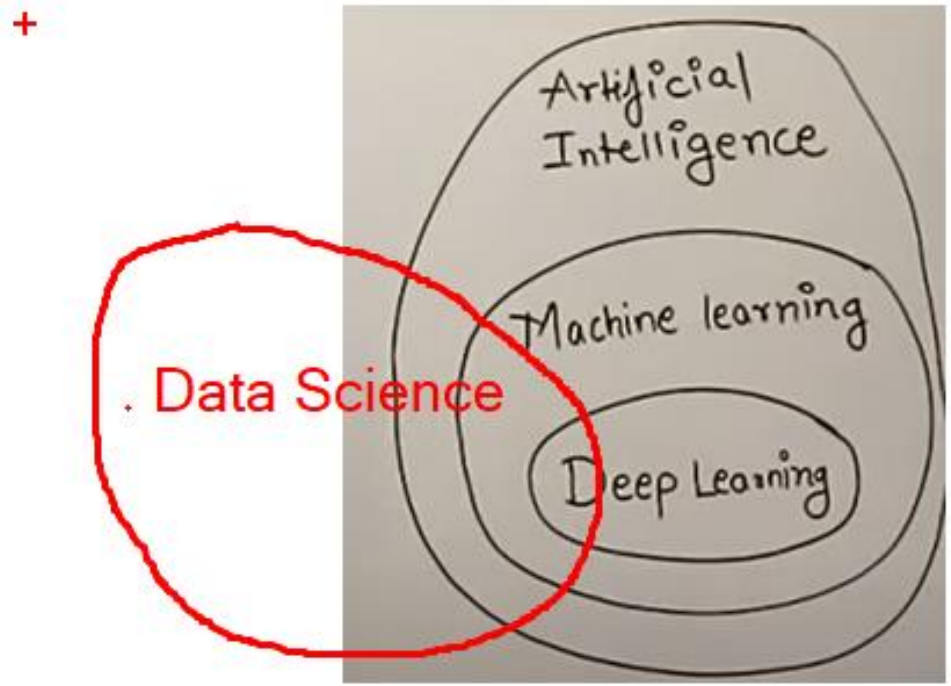
- Supervised
- Unsupervised
- Semi supervised
- Reinforcement Learning

Deep Learning

- It is the subset of ML, which mimic human brain.

- Three popular Deep Learning Techniques are:
  - ANN – Artificial Neural Network
  - CNN- Convolution Neural Network
  - RNN- Recurrent Neural Network

Summary:


Artificial Intelligence
Machine learning
Deep Learning
Data Science

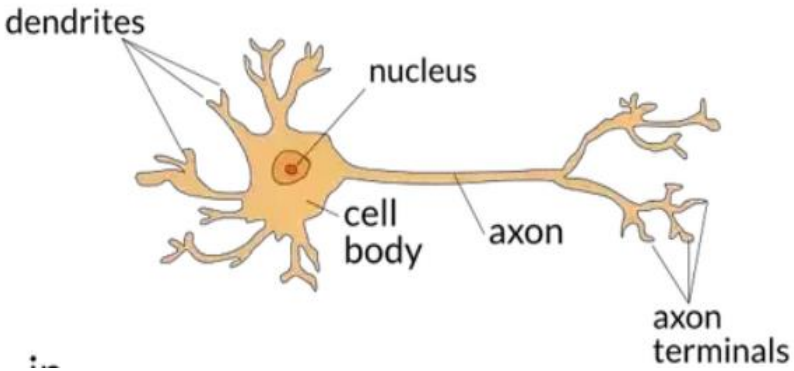
Introduction to ANN

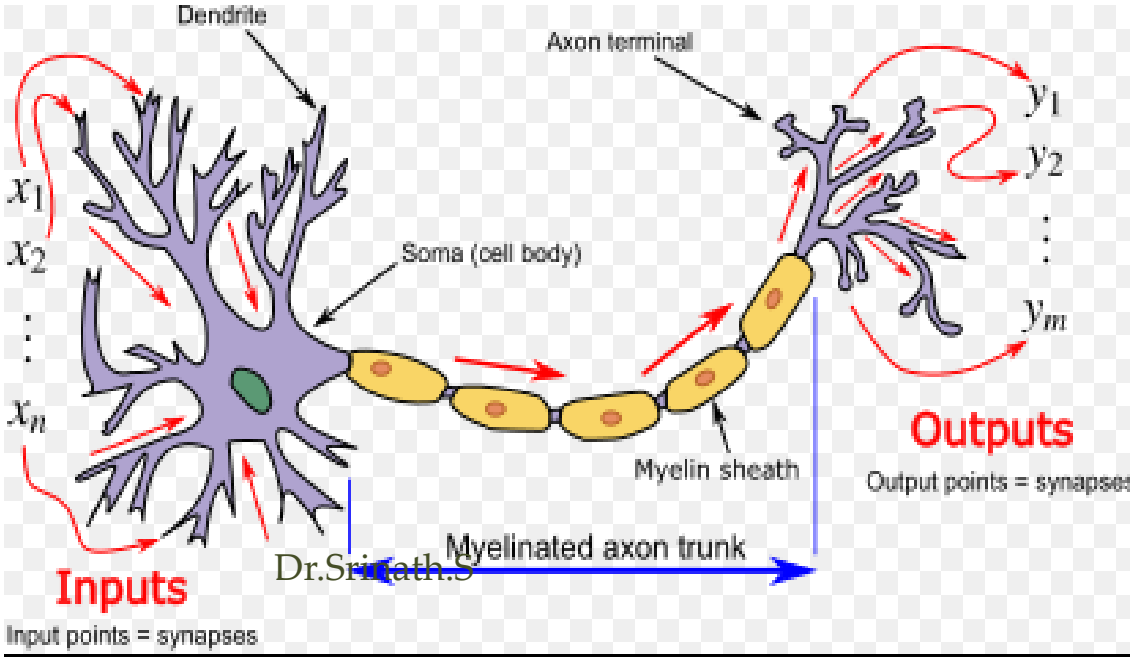
Dr.Srinath.S

Biological Neural Network to ANN

Dr.Srinath.S

Biological Neural Network (BNN)

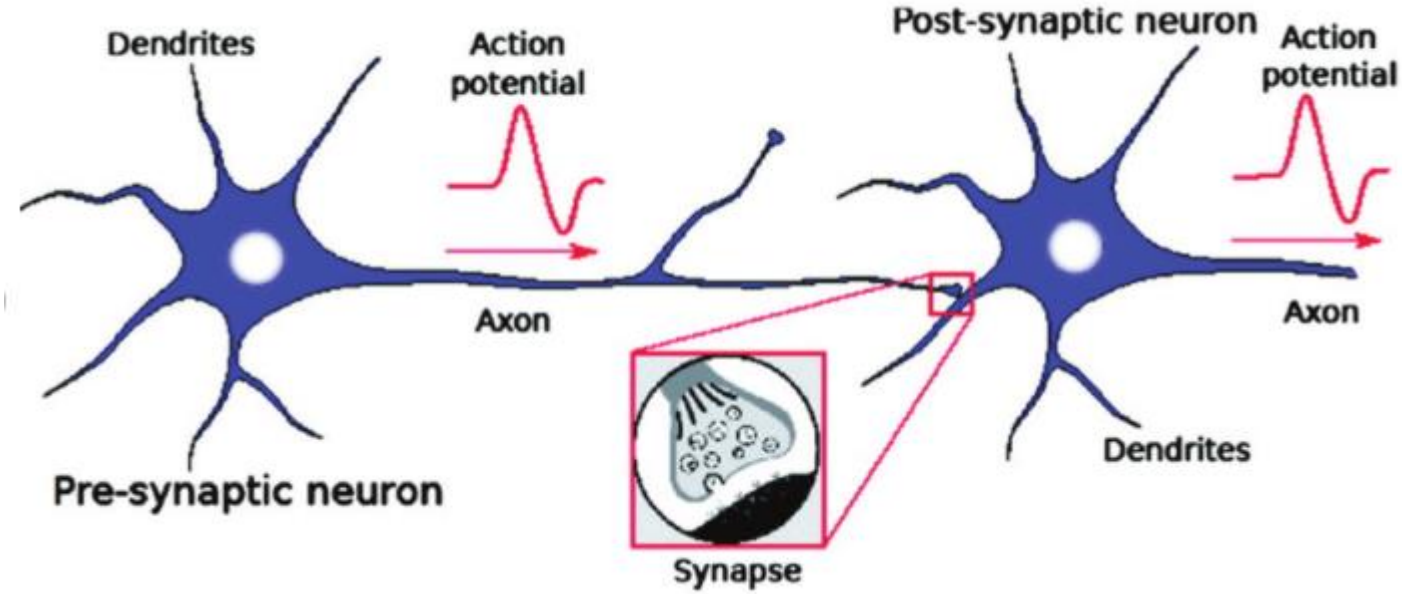




BNN parts

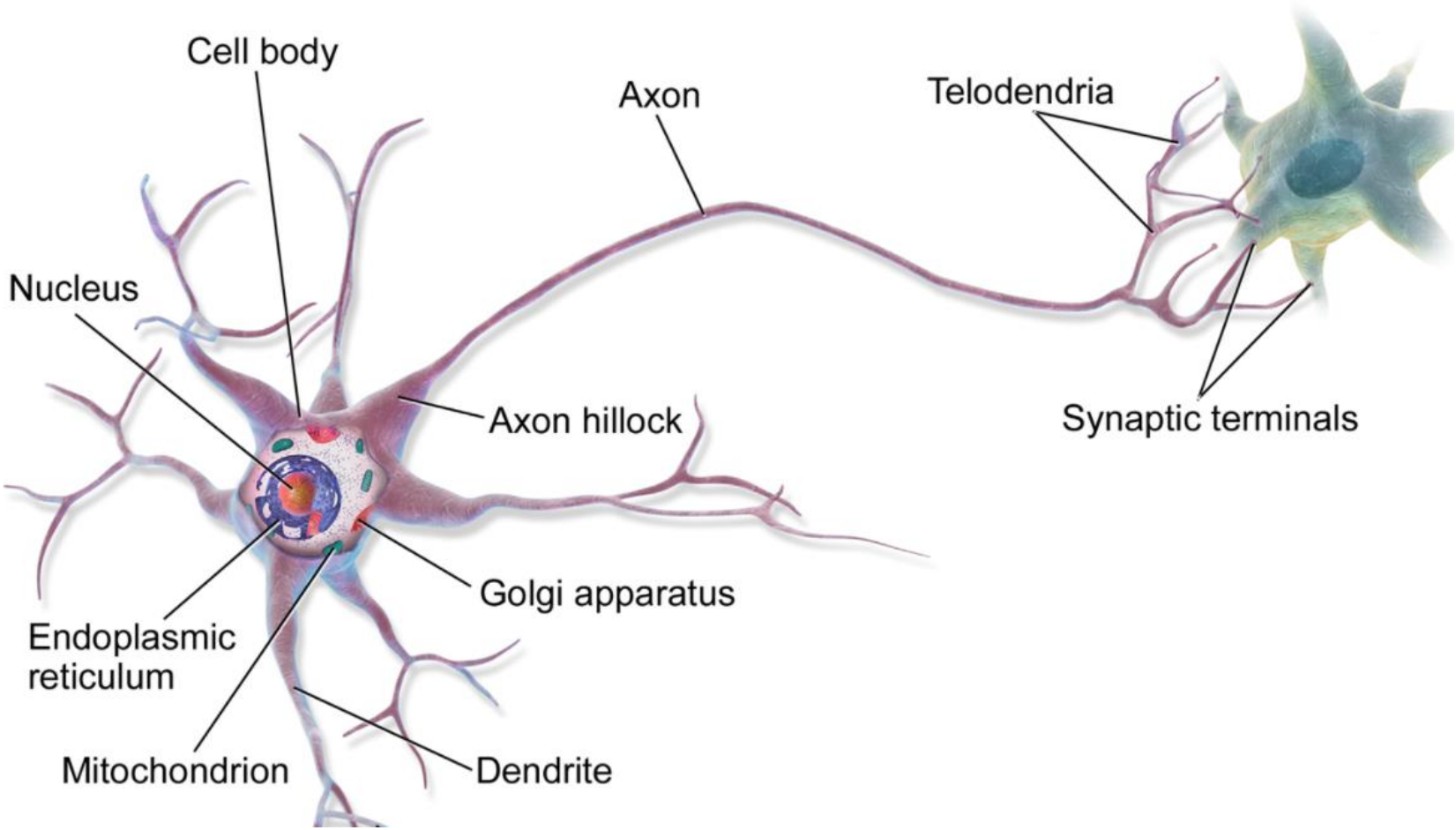
- BNN is composed of a cell body and many branching extensions called **dendrites** and one long extension called the **axon**.
- Primarily the parts of BNN are:
  - Cell body
  - Dendrites – Input part
  - Axon - output
- BNN is an interconnection of several biological neurons.
- Interconnection between two neurons is as shown in the next slide

Two neurons interconnected



- At the end AXON splits off into many branches called **telodendrion** and the tip of these branches are called synaptic terminals or simply **synapses.**
- The synapses of one neurons are connected to the dendrites of other neurons.
- Electric impulses called signals are passed from one neuron to another.
- BNN is a collection of billions of neurons, and each neurons are typically connected to thousands of other neurons.

Another view of BNN interconnection



Multiple layers in a biological network

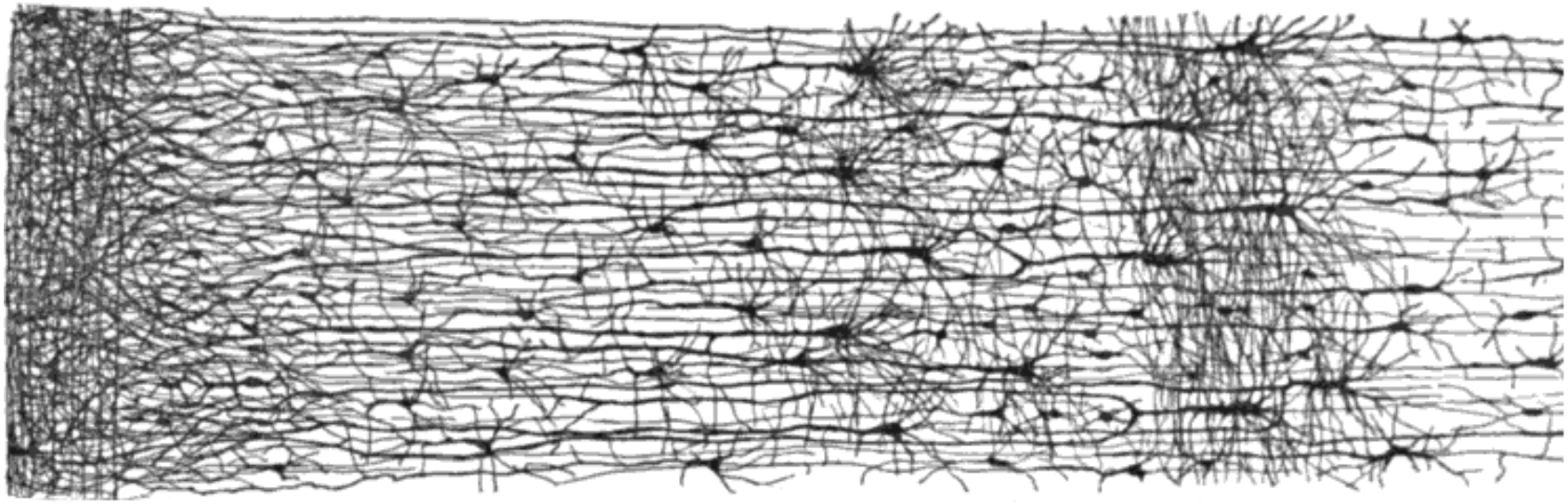


Figure 1-2. Multiple layers in a biological neural network (human cortex)[5]

Artificial Neural Network (ANN)

Dr.Srinath.S

Logical Computations with Neurons

- The artificial neuron simply activates its output when more than a certain number of its inputs are active.
- Let us see some of the ANNs performing simple logical computations.

ANNs performing simple logical computations

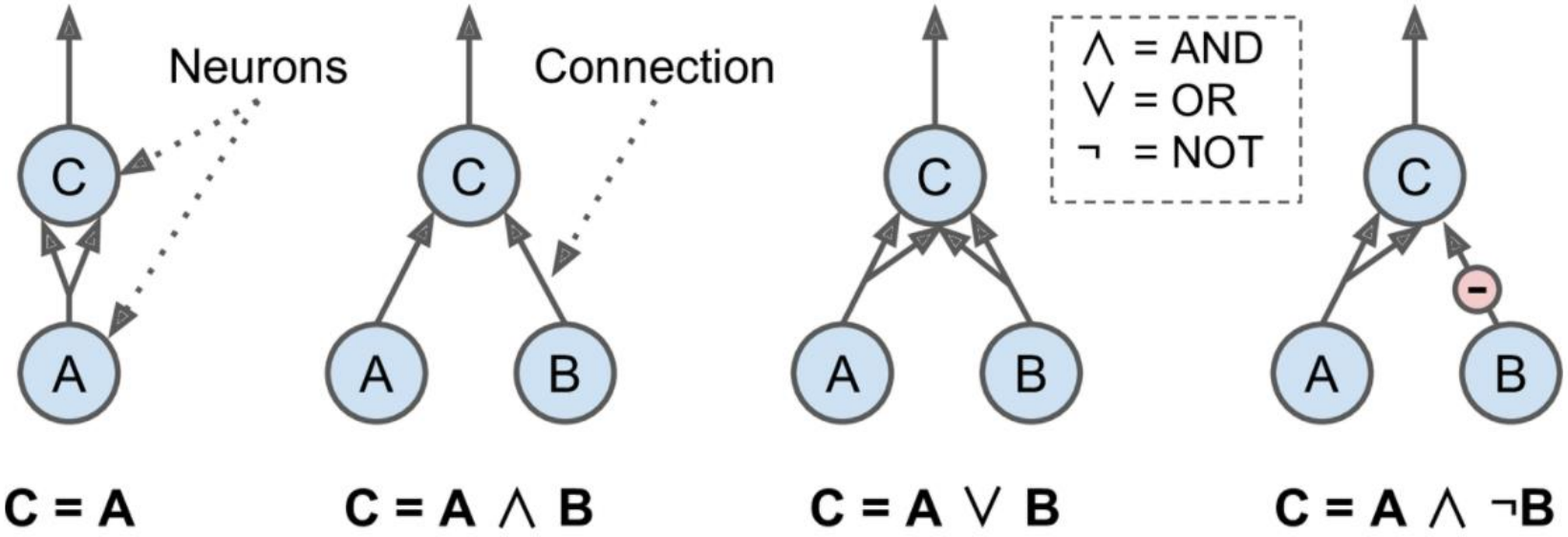


Figure 1-3. ANNs performing simple logical computations

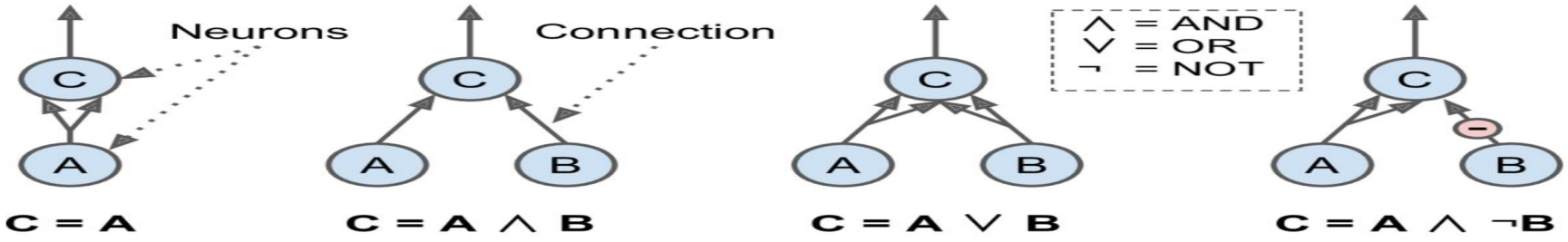


Figure 1-3. ANNs performing simple logical computations

- The first network on the left is simply the identity function: if neuron A is activated, then neuron C gets activated as well (since it receives two input signals from neuron A), but if neuron A is off, then neuron C is off as well.

- The second network performs a logical AND: neuron C is activated only when both neurons A and B are activated (a single input signal is not enough to activate neuron C).

- The third network performs a logical OR: neuron C gets activated if either neuron A or neuron B is activated (or both).

- Finally the fourth network computes a slightly more complex logical proposition: neuron C is activated only if neuron A is active and if neuron B is off. If neuron A is active all the time, then you get a logical NOT: neuron C is active when neuron B is off, and vice versa.

Perceptron

- ***Perceptron is a single layer neural network*** *or simply a neuron.*
- *So perceptron is a ANN with single layer neural network without having hidden layers.*

Perceptron consists of 4 parts

- input values
- weights and a Constant/Bias
- a weighted sum, and
- Step function / Activation function

Linear threshold unit (LTU)

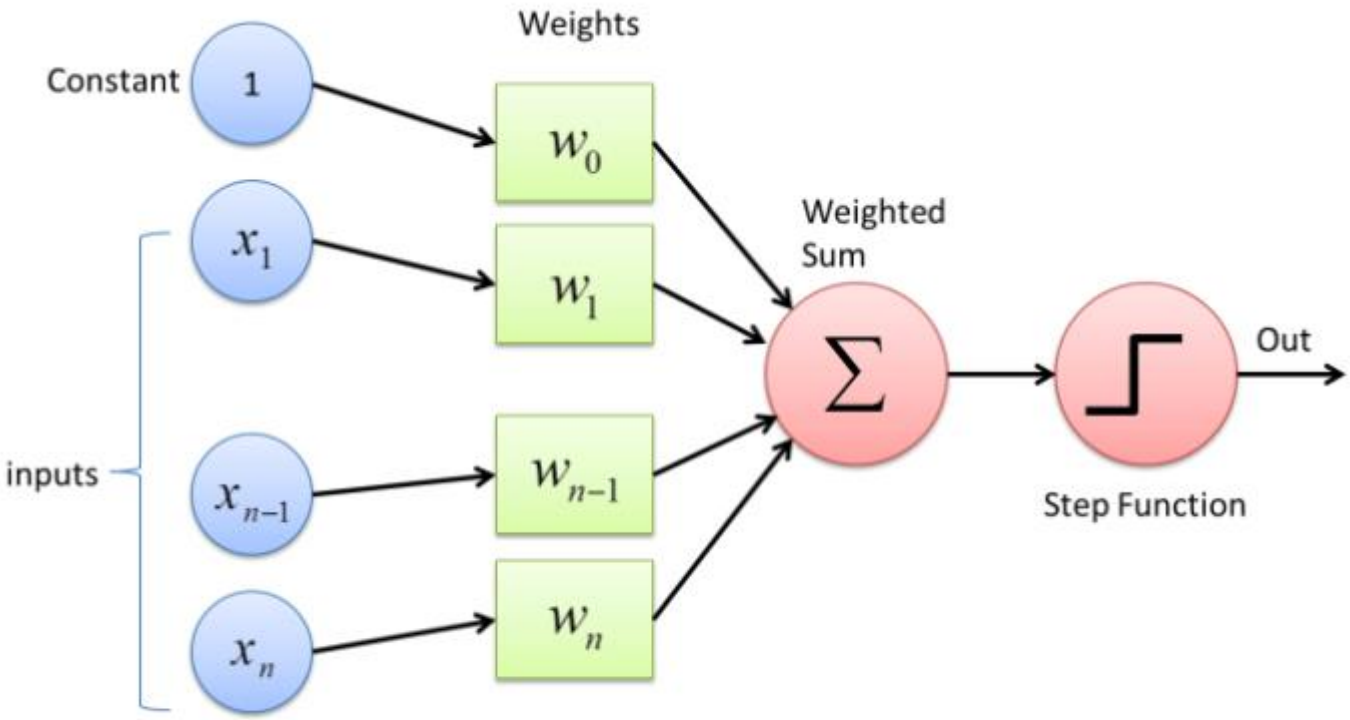


Fig : Perceptron

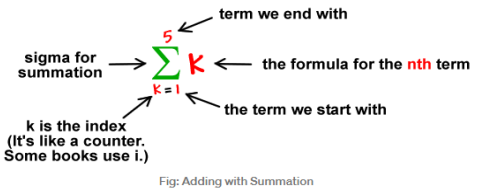


Fig: Adding with Summation

- A perceptron can have multiple input and single output as shown in the previous diagram (single LTU).
- Perceptron is simply composed of a single layer of LTUs.
- For example a 2 input and 3 output perceptron is as shown in the next slide.
- However a single layer perceptron will not have hidden layer.

A Perceptron with two inputs and three outputs is represented in Figure 10-5. This Perceptron can classify instances simultaneously into three different binary classes, which makes it a multioutput classifier.

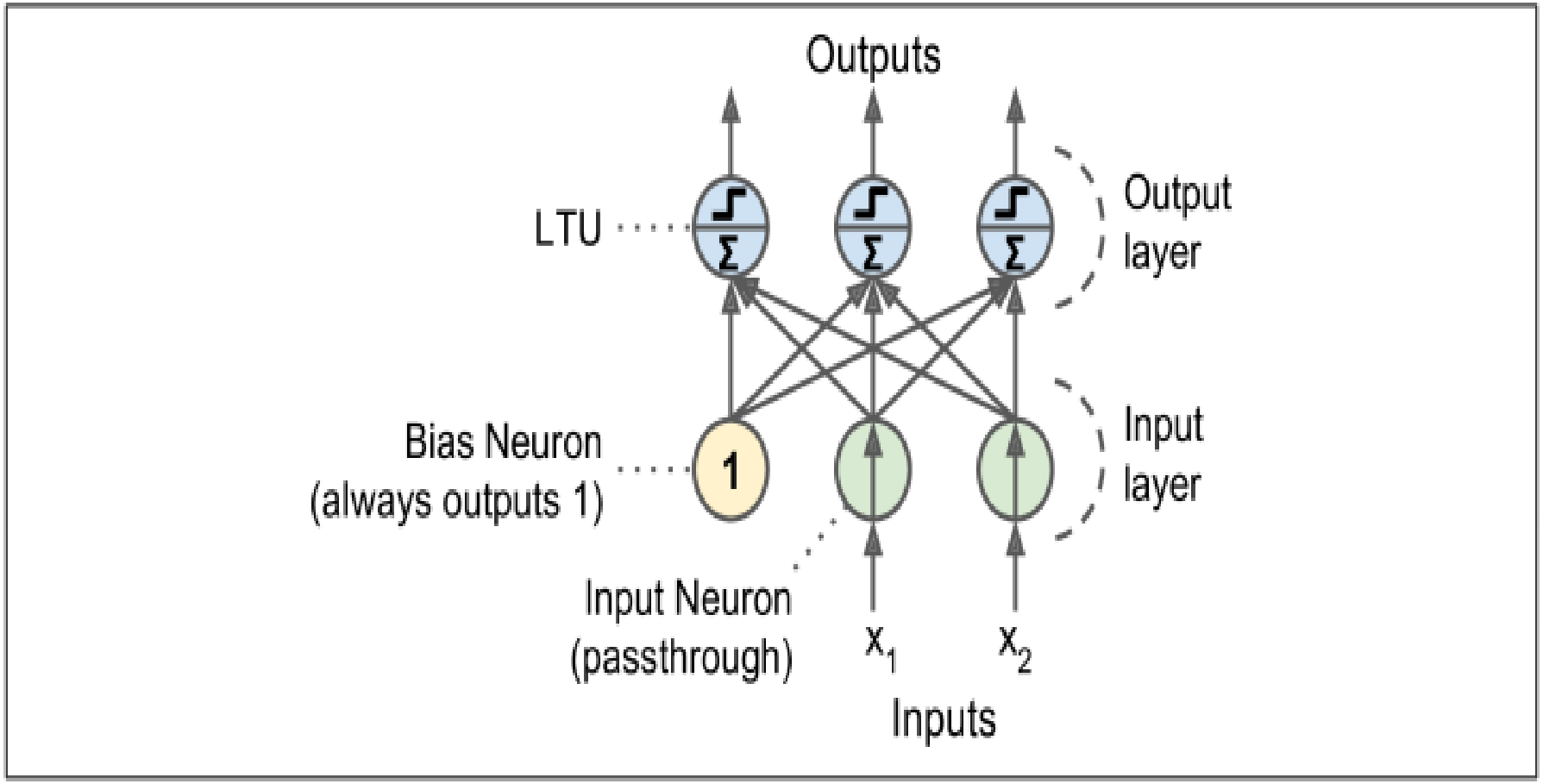


Figure 10-5. Perceptron diagram

Working of Perceptron

- The perceptron works on these simple steps:

- a. All the inputs ***x*** are multiplied with their weights ***w***. Let's call it ***k.***
- ***Add*** all the multiplied values and call them ***Weighted Sum.***
- ***Finally Apply*** that weighted sum to the correct ***Activation Function***.

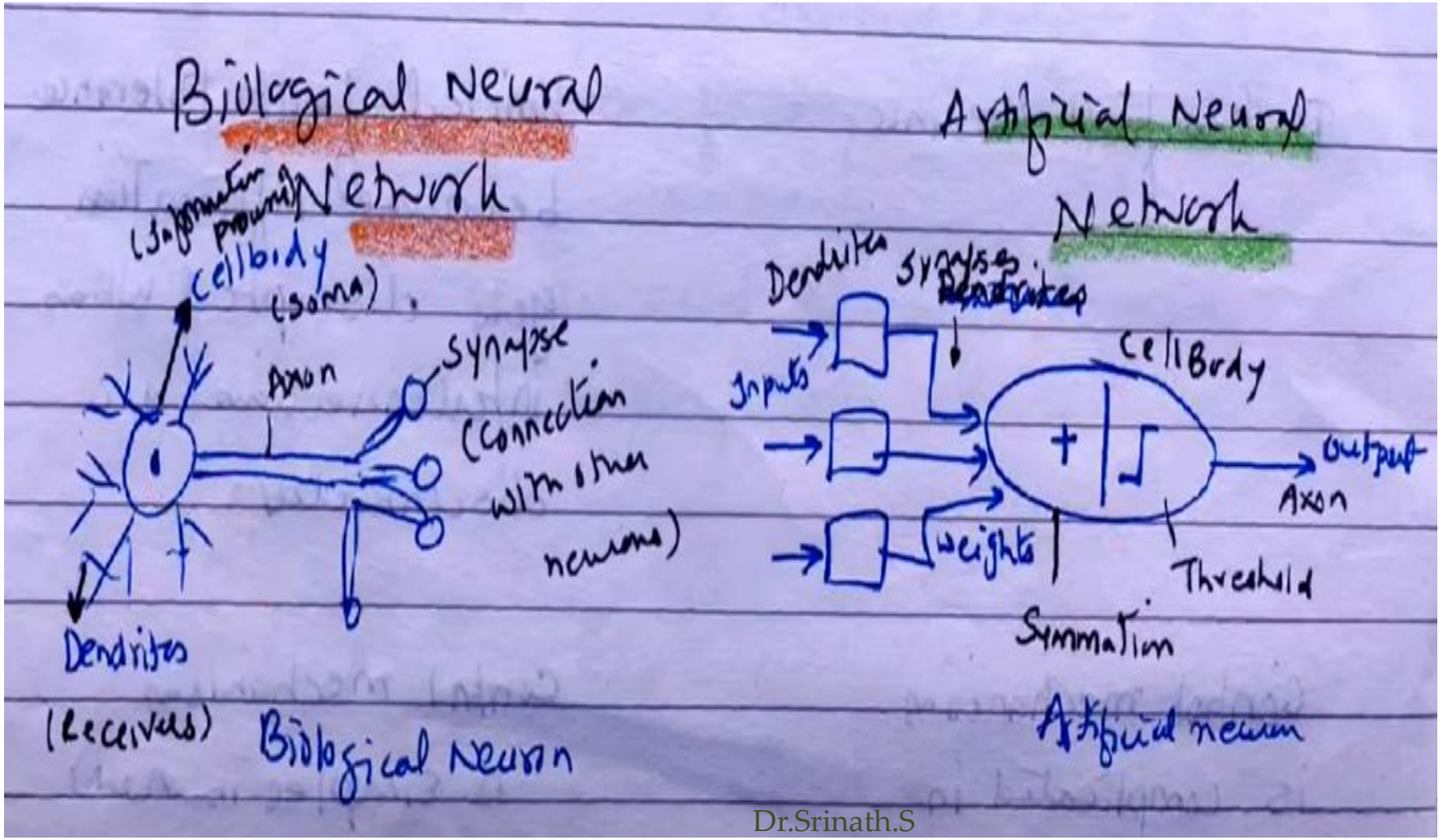
  For Example: Heaviside step function.

Step activation function

Equation 10-1. Common step functions used in Perceptrons

$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \qquad \text{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

Comparison between BNN and ANN



Equation for the perceptron learning rule

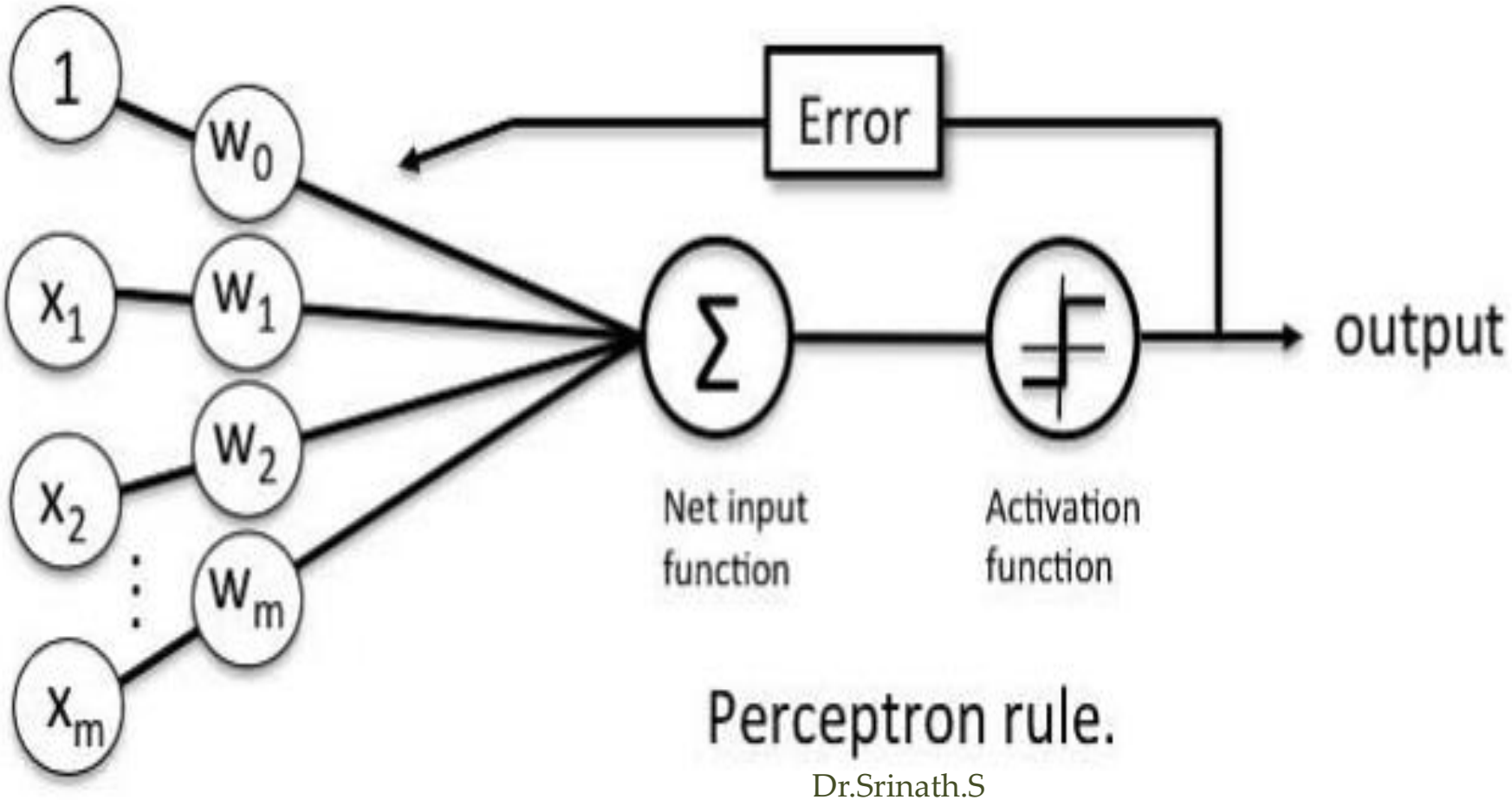
- Perceptrons are trained considering the error made by the network.
- For every output neuron that produced a wrong prediction, it reinforces the connection weights from the inputs that would have contributed to the correct prediction.
- Equation is given in the next slide

- Perceptron learning rule (weight update)

$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta(\hat{y}_j - y_j)x_i$$

- • wi, j is the connection weight between the ith input neuron and the jth output neuron.
- • xi is the ith input value of the current training instance.
- • ^yj is the output of the jth output neuron for the current training instance.
- • yj is the target output of the jth output neuron for the current training instance.
- • η is the learning rate.
- This process is repeated till the error rate is close to zero

Perceptron Learning Rule



Perceptron rule.

- A perceptron is simply composed of a single layer of LTUs, with each neuron connected to all the inputs.
- some of the limitations of Perceptrons can be eliminated by stacking multiple Perceptrons.
- The resulting ANN is called a Multi-Layer Perceptron (MLP).
- MLP will have one or more hidden layers.

Multi Layer Perceptron (MLP)

An MLP is composed of one (passthrough) input layer, one or more layers of LTUs, called *hidden layers*, and one final layer of LTUs called the *output layer* (see Figure 10-7). Every layer except the output layer includes a bias neuron and is fully connected to the next layer. When an ANN has two or more hidden layers, it is called a *deep neural network* (DNN).

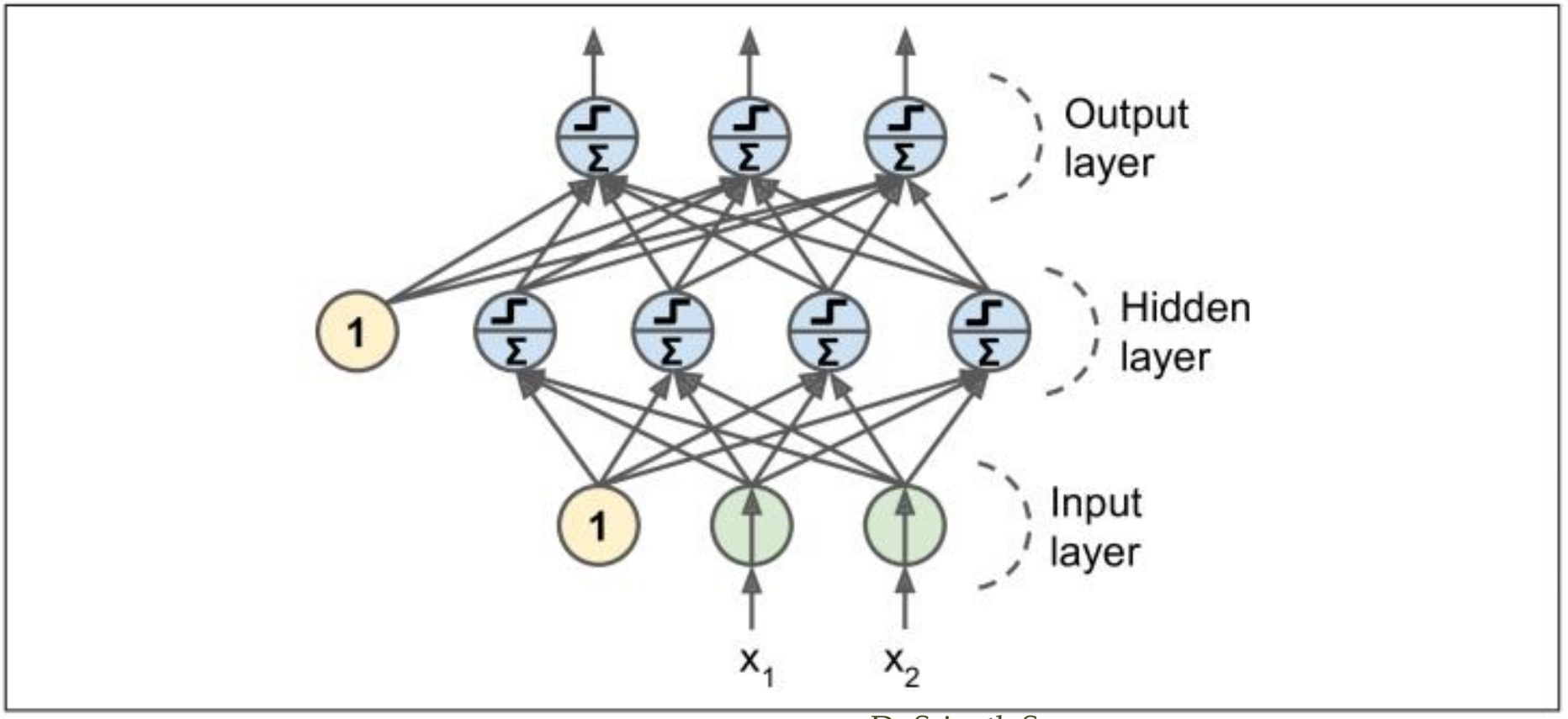
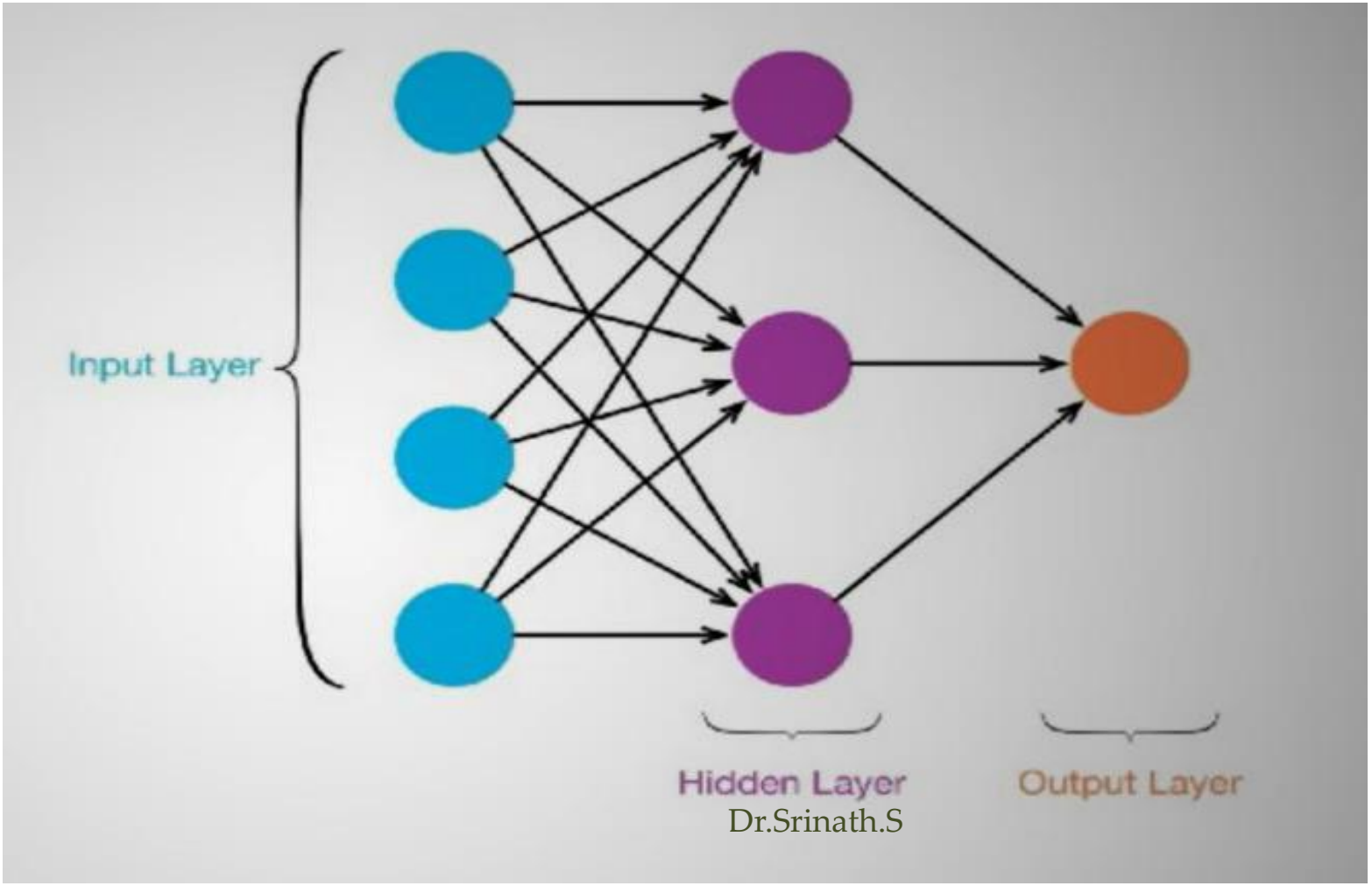
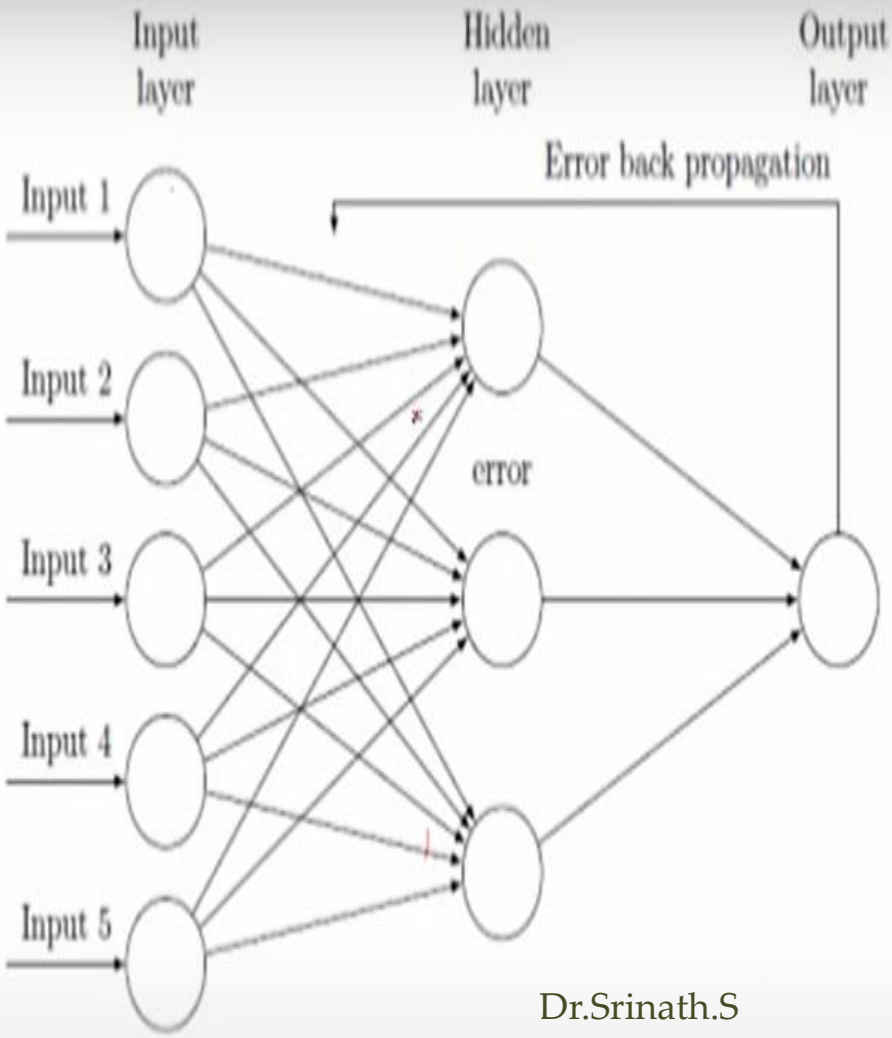


Figure 10-7. Multi-Layer Perceptron

MLP : Simplified view



Example for ANN



Banana?

Apple?

Mango?

Shallow or Deep ANN

- MLP can be either *shallow* or *deep*.
- They are called shallow when they have only one hidden layer (i.e. one layer between input and output).
- They are called deep when hidden layers are more than one. (Two or more)
- This is where the expression **DNN** (Deep Neural Network) comes.
- So DNN is a variant of ANN having 2 or more hidden layer.

Summary

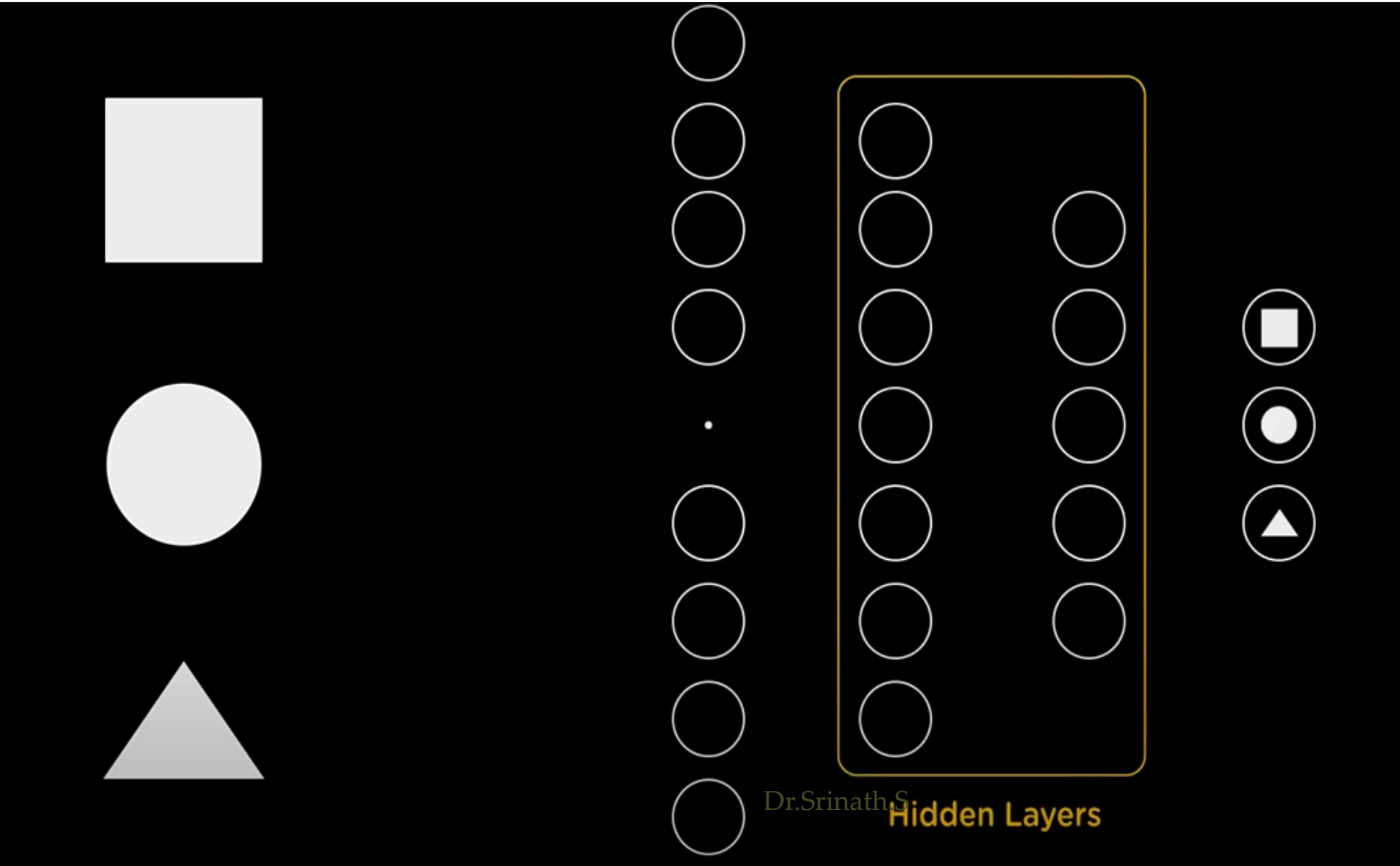
- Perceptron: It is a ANN with single layer neural network without having hidden layers. It will have only input and output layer.
- MLP – ANN with 2 or more layers called MLP
- MLP with only one hidden layer is called shallow ANN
- MLP with two or more hidden layers is called deep ANN, which is popularly known as Deep Neural Network.
- Perceptron, Shallow ANN, Deep ANN are all variants of ANN.

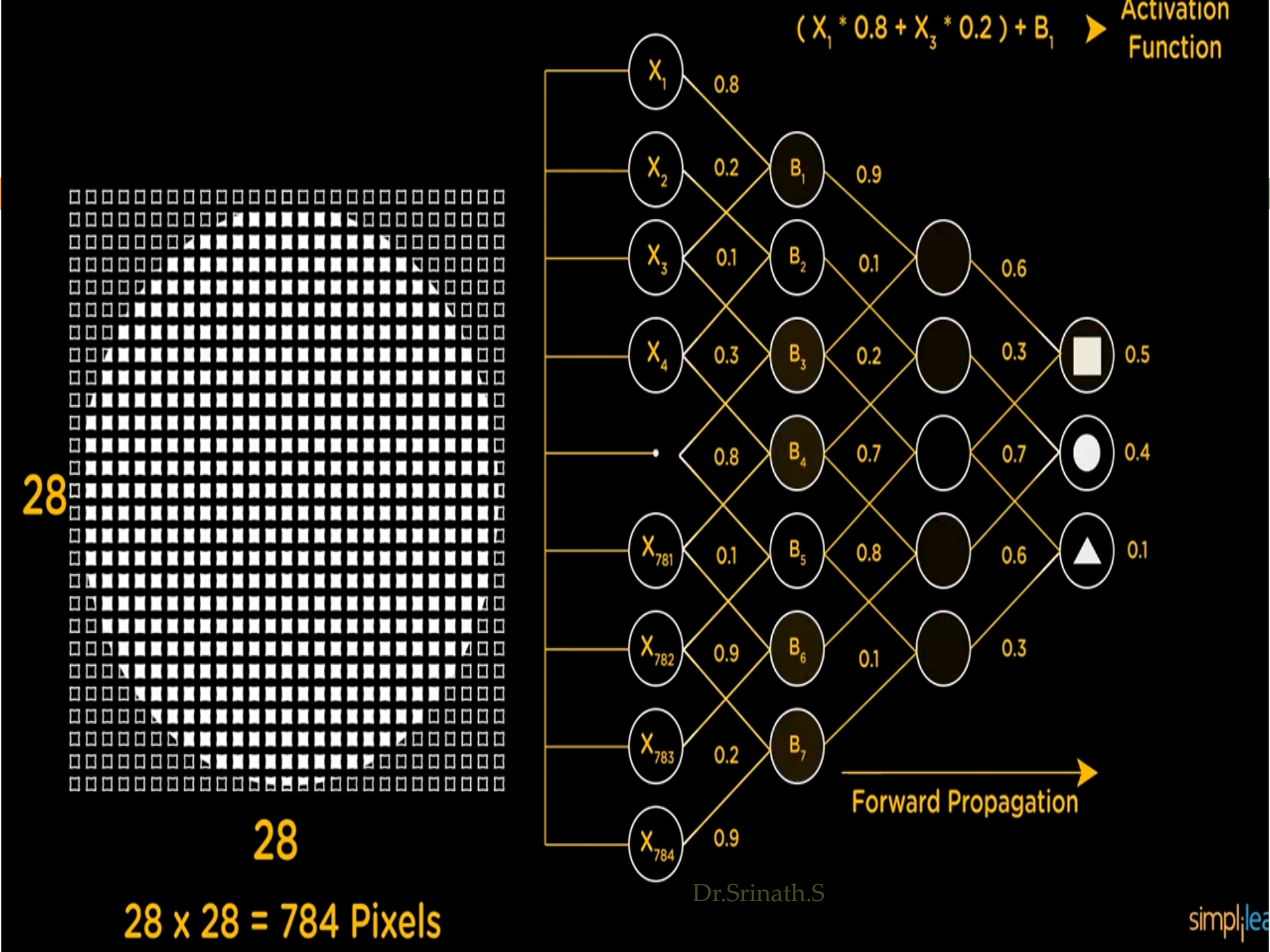
How many hidden layers?

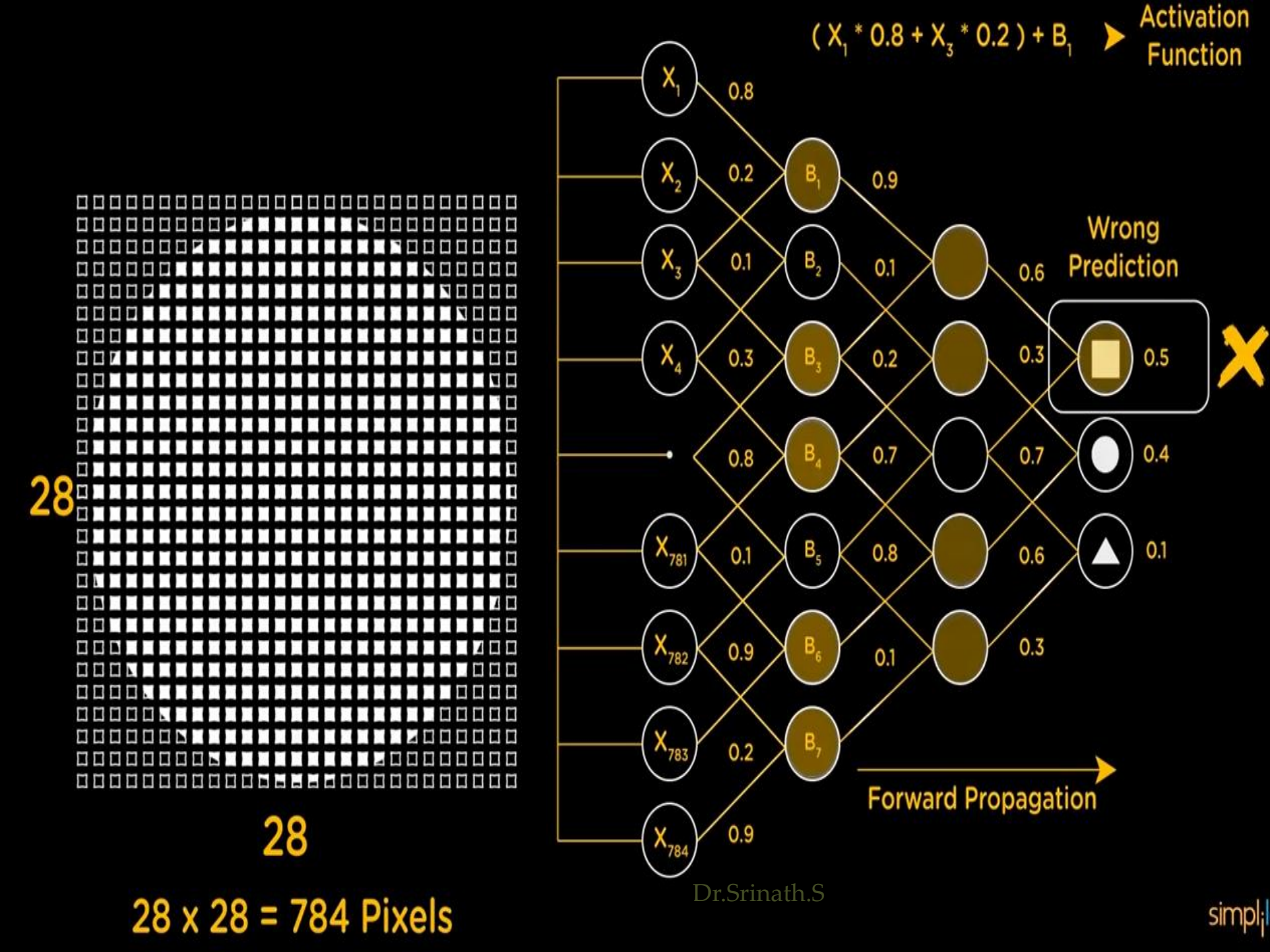
- For any application, number of hidden layers and number of nodes in each hidden layer is not fixed.

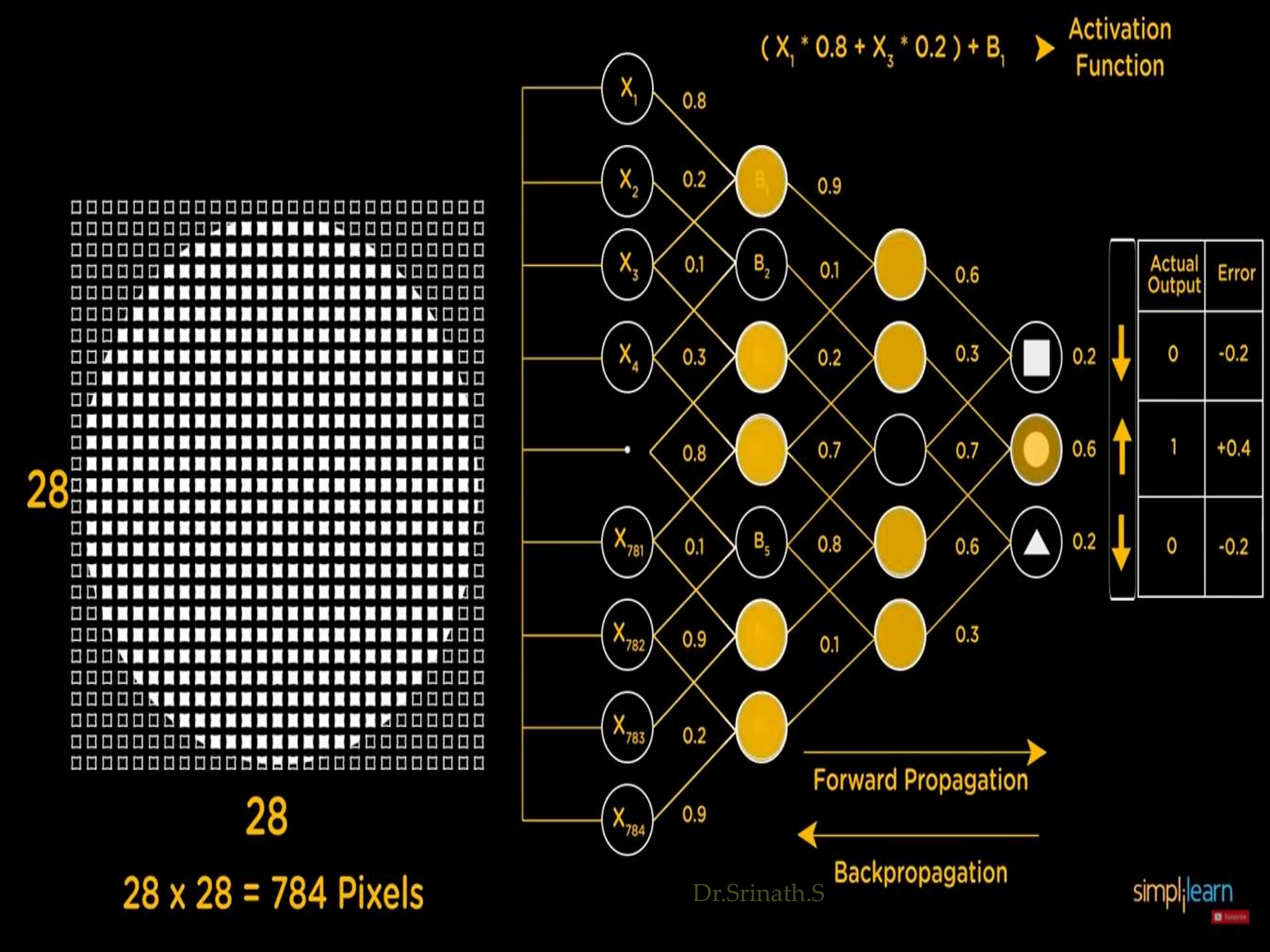
- It will be varied till the output moves towards zero error or till we get a satisfactory output.

Example: Neural Network to find whether the given input is Square or circle or triangle




(X1 * 0.8 + X3 * 0.2) + B1
Activation Function
X1
X2
X3
X4
X781
X782
X783
X784
0.8
0.2
0.1
0.3
0.8
0.1
0.9
0.2
B1
B2
B3
B4
B5
B6
B7
0.9
0.1
0.2
0.7
0.8
0.1
0.6
0.3
0.7
0.6
0.3
0.5
0.4
0.1
Forward Propagation
28
28
28 x 28 = 784 Pixels
Dr.Srinath.S
simplilea


(X1 * 0.8 + X3 * 0.2) + B1
Activation Function
X1
X2
X3
X4
X781
X782
X783
X784
0.8
0.2
0.1
0.3
0.8
0.1
0.9
0.2
B1
B2
B3
B4
B5
B6
B7
0.9
0.1
0.2
0.7
0.8
0.1
0.6
0.3
0.7
0.6
0.3
Wrong Prediction
0.5
0.4
0.1
Forward Propagation
28
28
28 x 28 = 784 Pixels


$(X_1 * 0.8 + X_3 * 0.2) + B_1$
Activation Function
28
28
28 x 28 = 784 Pixels
X_1
X_2
X_3
X_4
X_{781}
X_{782}
X_{783}
X_{784}
B_1
B_2
B_5
0.8
0.2
0.1
0.3
0.8
0.1
0.9
0.2
0.9
0.1
0.2
0.7
0.8
0.1
0.6
0.3
0.7
0.6
0.3
0.2
0.6
0.2
Actual Output
Error
0
-0.2
1
+0.4
0
-0.2
Forward Propagation
Backpropagation
Dr.Srinath.S
simplilearn

CNN

CNN (Convolutional Neural Network):

They are designed specifically for **computer vision** (they are sometimes applied elsewhere though).

Their name come from **convolutional layers.**

They have been invented to receive and process pixel data.

RNN

RNN (Recurrent Neural Network):

They are the "time series version" of ANNs.

They are meant to process *sequences* of data.

They are at the basis of forecast models and language models.

The most common kind of recurrent layers are called **LSTM** (Long Short Term Memory) and

GRU (Gated Recurrent Units): their cells contain small, in-scale ANNs that choose how much past information they want to let flow through the model. That's how they modeled "memory".

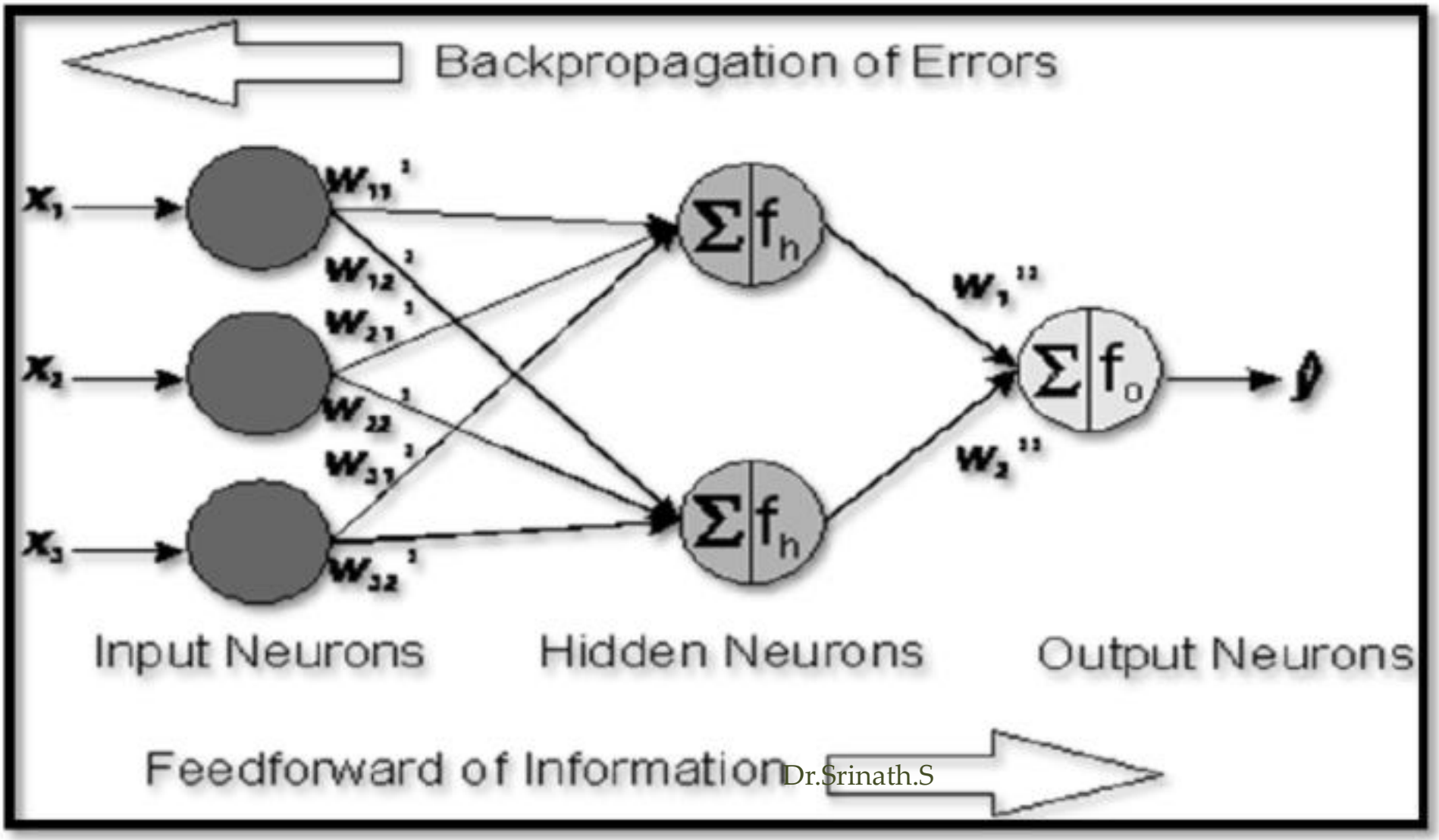
Forward and Backward Propagation

- Forward Propagation is the way to move from the Input layer (left) to the Output layer (right) in the neural network. It is also called as Feed forward.

- The process of moving from the right to left i.e backward from the Output to the Input layer is called the Backward Propagation.

- Backward propagation is required to correct the error or generally it is said to make the system to learn.

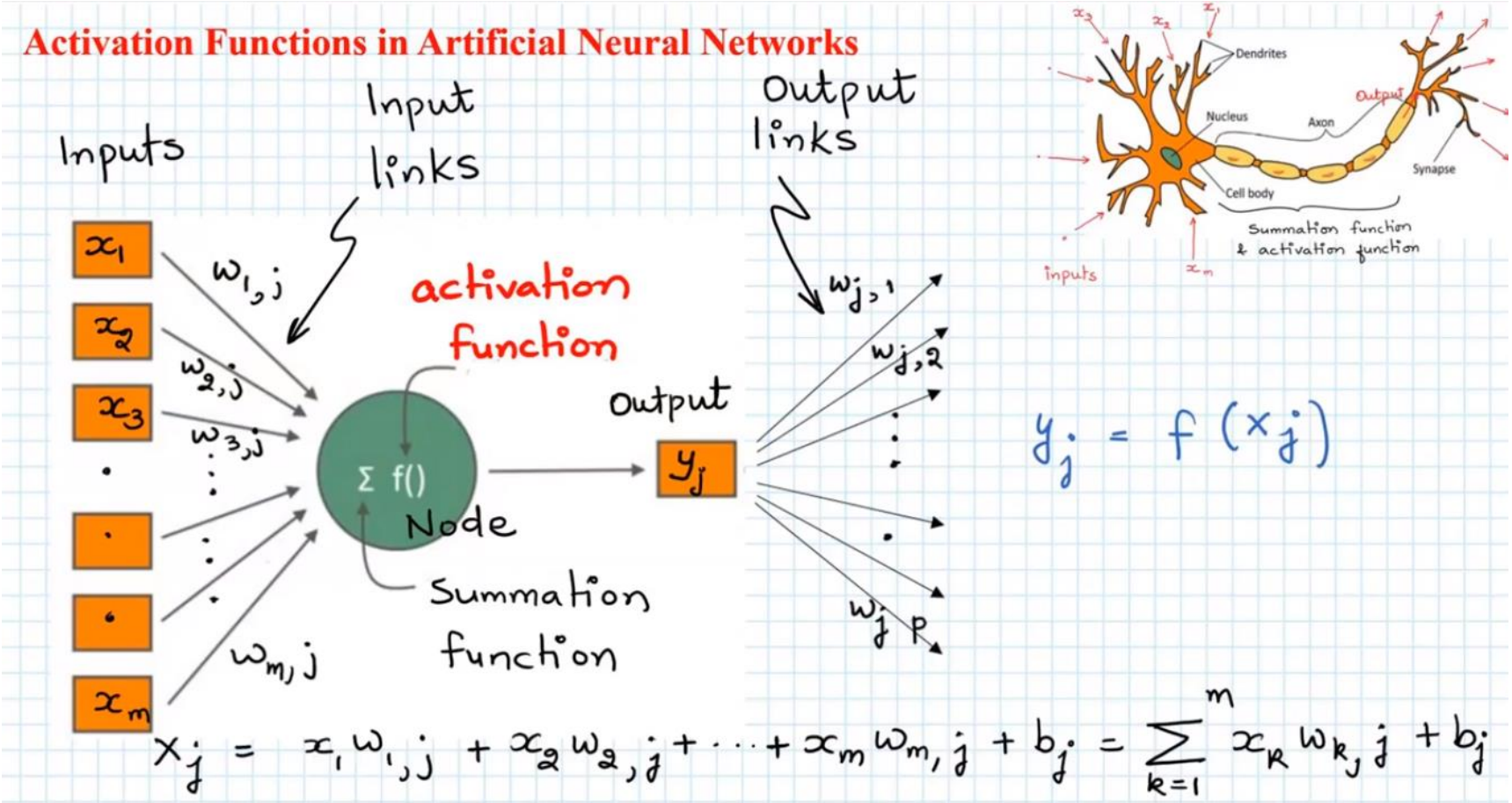
Feed forward and Backward propagation



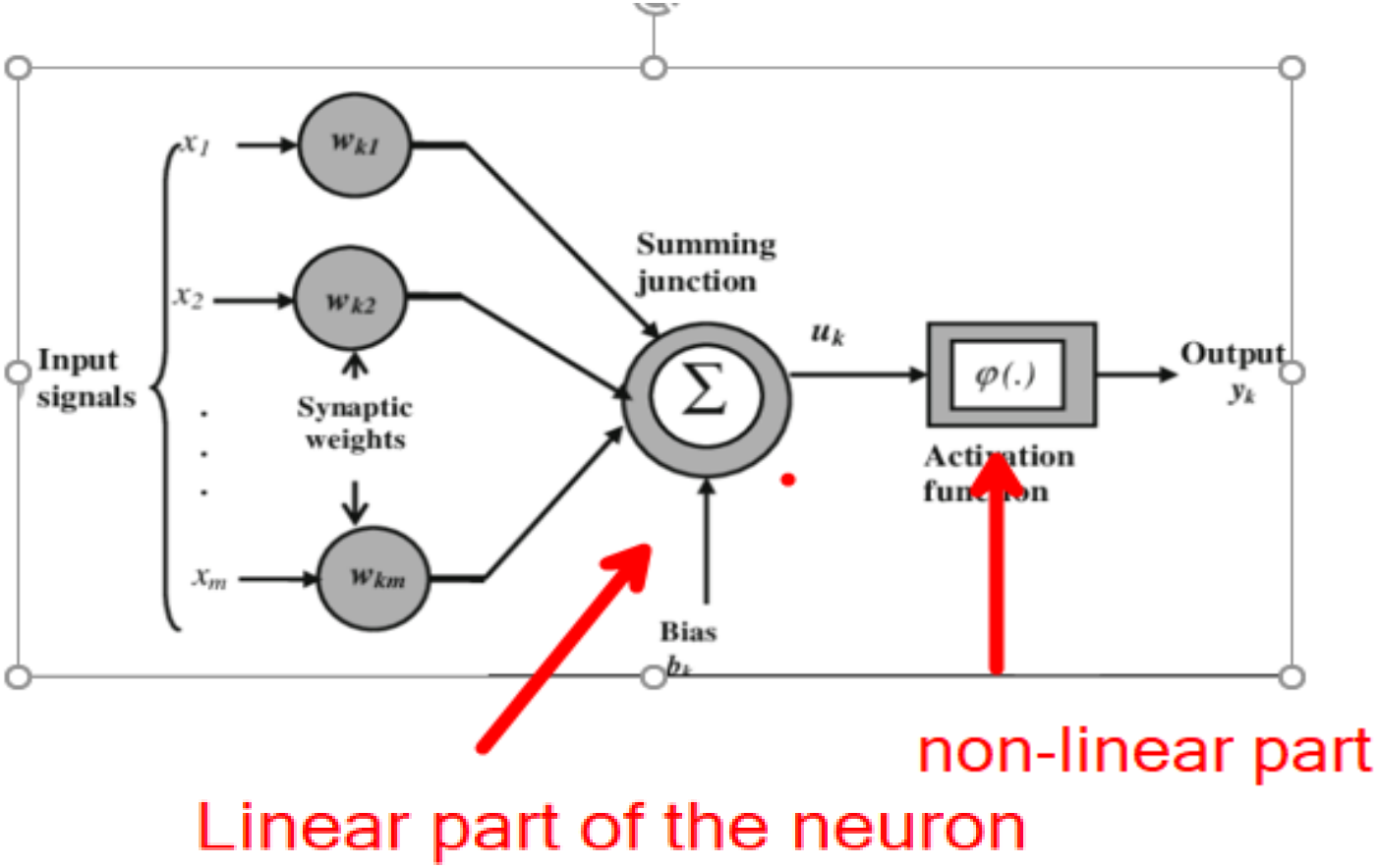
Backward propagation

- Measure the network's output error (difference between actual and obtained)
- Tweak the waits to correct or to reduce the error.
- Move from output layer to input layer one step at a time.
- Compute how much each neuron in the last hidden layer contributed to each output neuron's error.
- Later it moves to the next hidden layer in the reverse direction till the input layer and keeps updating the waits.
- Tweaking the weights to reduce the error is called gradient descent step

Summary… and moving toward activation functions



Linear and Non-linear part of neuron



Need of activation function:

- They are used in the hidden and output layers.
- Activation function is a function that is added into an artificial neural network in order **to help the network learn complex patterns in the data**.
- The activation function will decide what is to be fired to the next neuron

Can ANN work without an activation function?

- Then it becomes linear.
- Basically, the cell body has two parts, one is linear and another one is non-linear. Summation part will do linear activity, and activation function will perform non-linear activity.
- If activation function is not used then, every neuron will only be performing a linear transformation on the inputs using the weights and biases.
- Although linear transformations make the neural network simpler, but this network would be less powerful and will not be able to learn the complex patterns from the data. Hence the need for activation function.

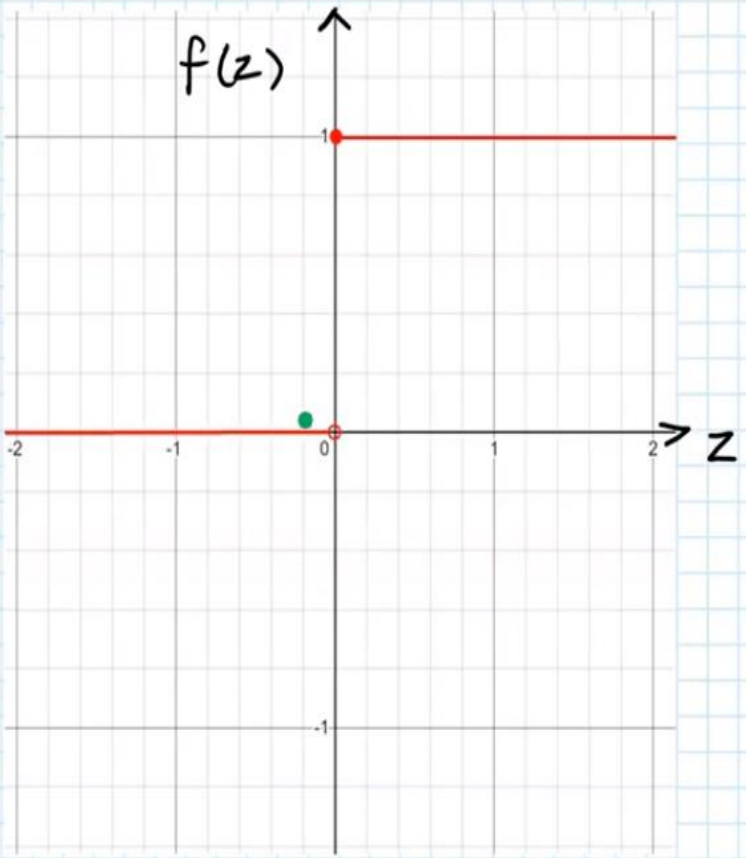
Popular Activation Functions

1. Popular types of activation functions are:
    1. Step function
    2. Sign function
    3. Linear function
    4. ReLU (Rectified Linear Unit): no –ve value
    5. Leaky ReLU
    6. Tanh
    7. Sigmoid
    8. softmax

1. Step Function

1. Step function

$$f(z) = \begin{cases} 0 & z < 0 \\ 1 & z \geq 0 \end{cases}$$



Range = Possible Output

$\{0, 1\}$

Used in: Hidden Layer, Output Layer for Classification

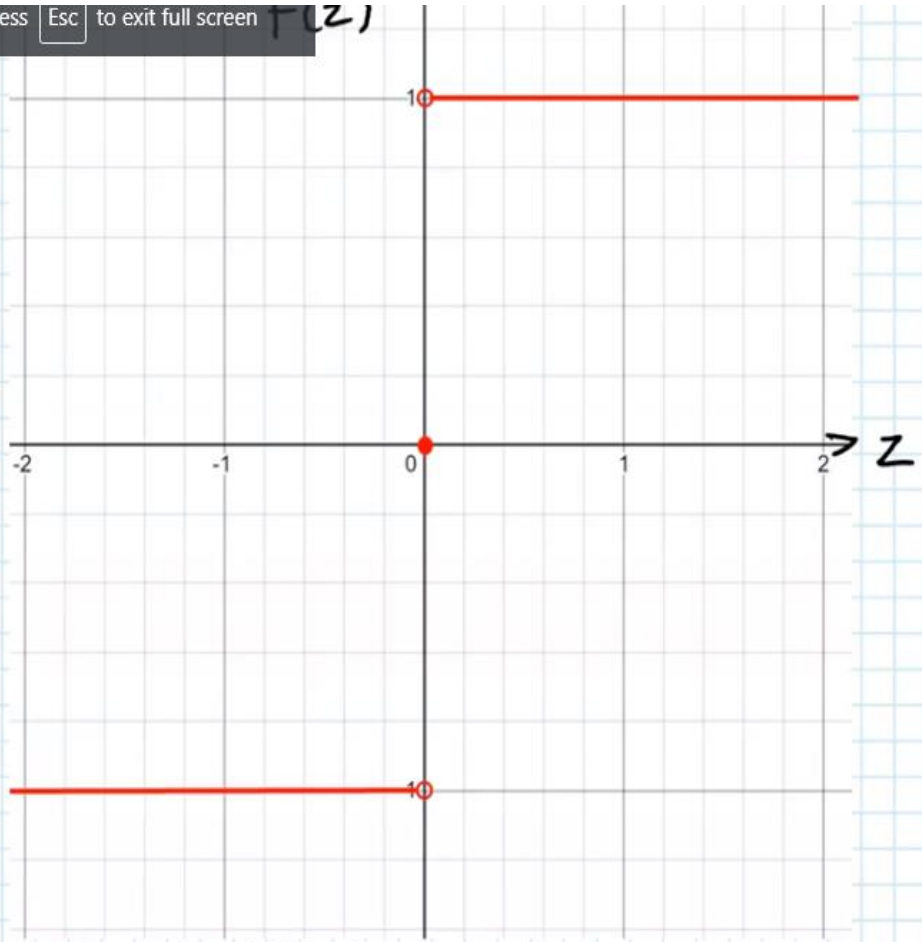
2. Sign function

2. Signum (sgn or sign) function

$$f(z) = \begin{cases} -1 & z < 0 \\ 1 & z > 0 \end{cases}$$

Range = Possible Outputs

$\{-1, 1\}$



Used in: Hidden Layer, Output Layer for Classification

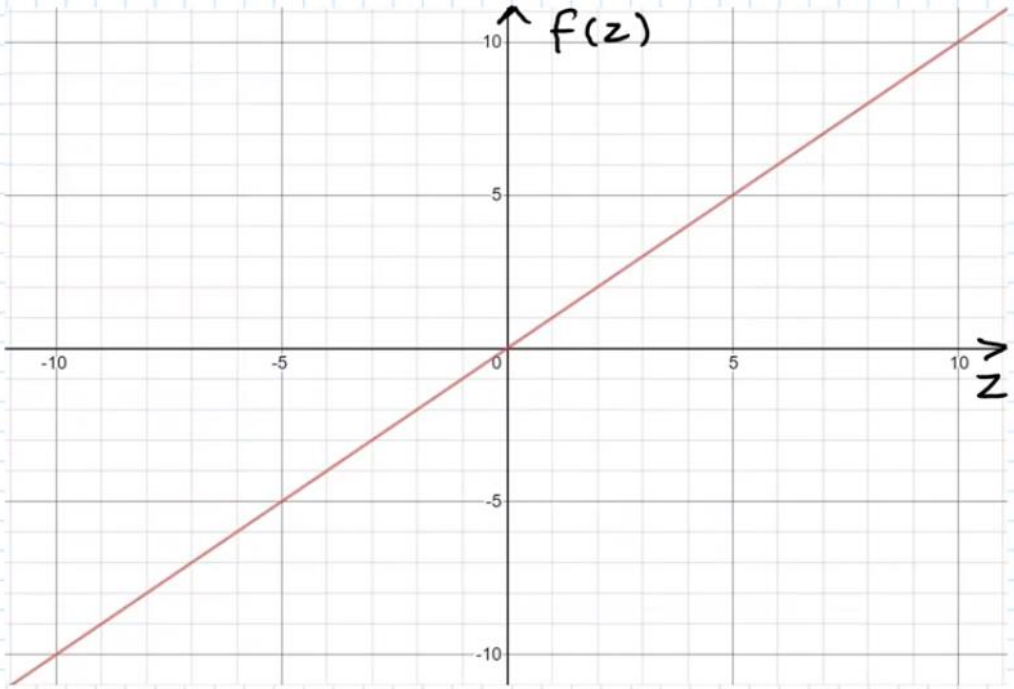
3. Linear function

3. Linear Function

$f(z) = z$

Range = Possible Outputs

$(-\infty, \infty)$



Used in: Hidden Layer, Output Layer for Regression

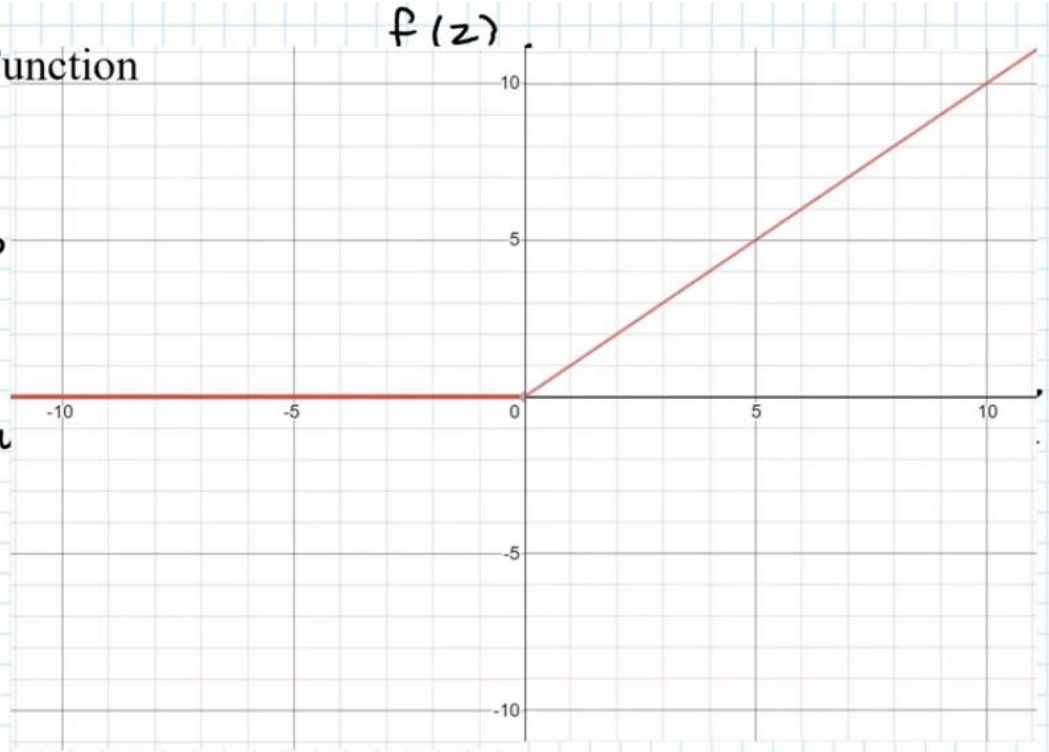
4. ReLU function

4a. Rectified Linear Unit (ReLU) Function

$$f(z) = \begin{cases} 0 & z < 0 \\ z & z \geq 0 \end{cases}$$

Range = Possible Outpu

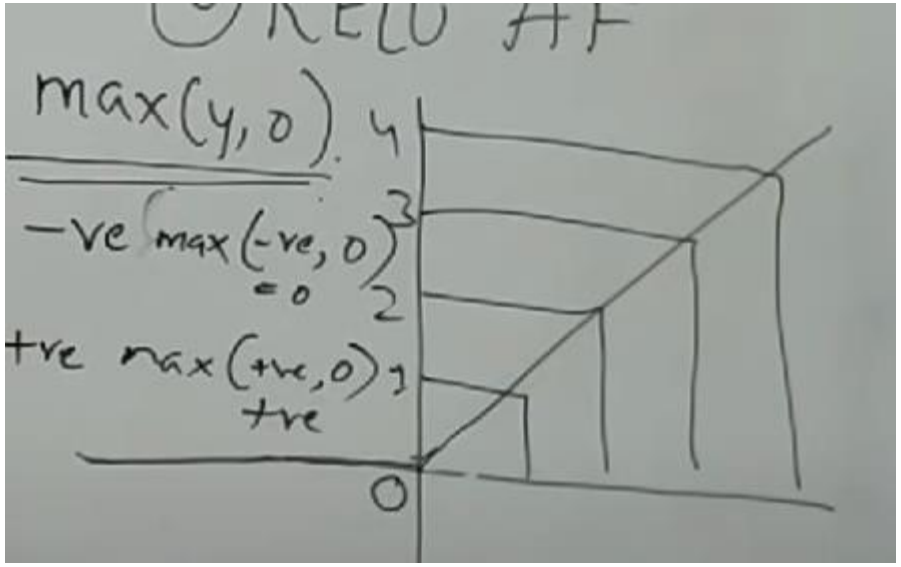
$[0, \infty)$



Used in: Hidden Layer, Output Layer for Regression (only positive output)

ReLU (Rectified Linear Unit)

- It will produce the same output for +ve value, and 0 for all –ve values.



5. Leaky Rectified Linear Unit

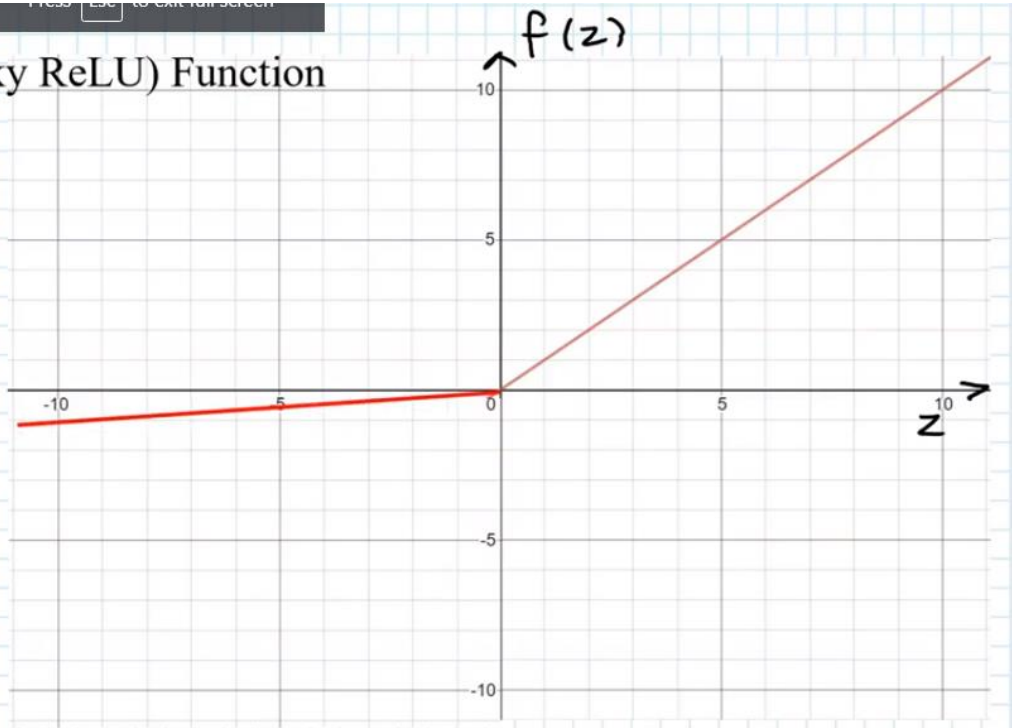
4b. Leaky Rectified Linear Unit (Leaky ReLU) Function

$$f(z) = \begin{cases} az & z < 0 \\ z & z \geq 0 \end{cases}$$

a is a small + number

Range = Possible outputs

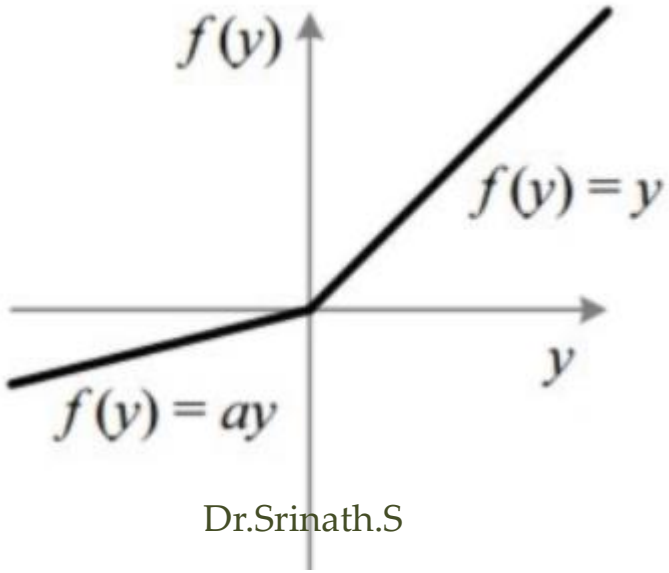
$(-\infty, \infty)$



Used in: Hidden Layer

Leaky ReLU

- Leaky Rectified Linear Unit, or Leaky ReLU, is **a type of activation function based on a ReLU, but it has a small slope for negative values instead of a flat slope**.



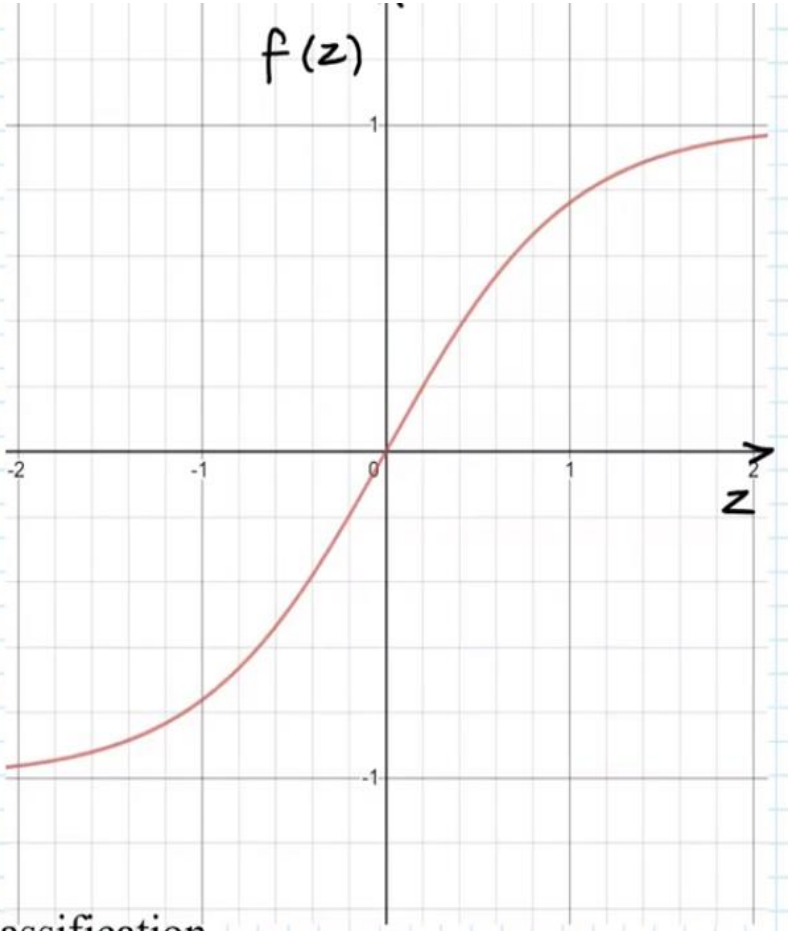
6. Tanh: Hyperbolic Tangent : any value between -1 to +1.

5. Hyperbolic tangent; tanh(z)

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

Range = Possible Outputs

(-1, 1)



Used in: Hidden Layer, Output Layer for Classification

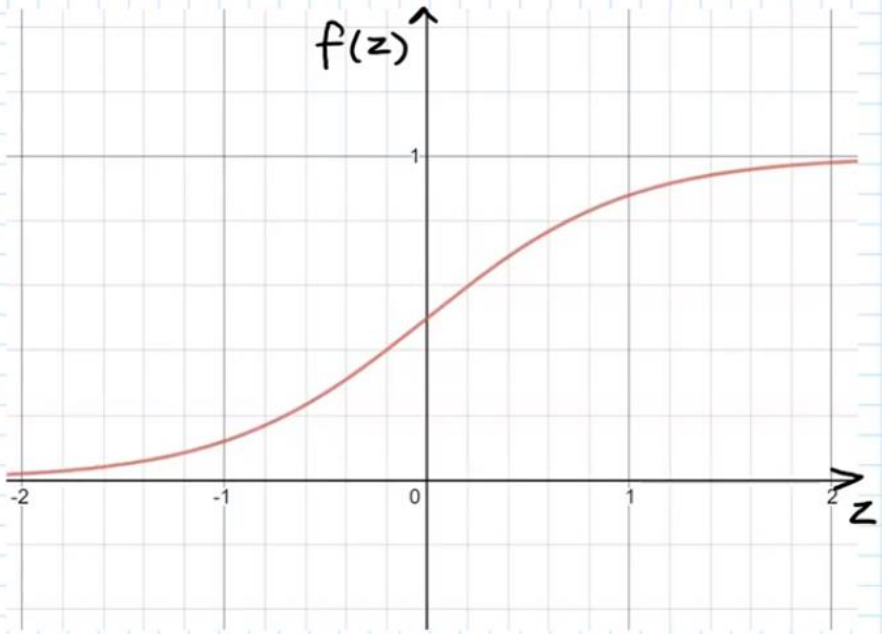
7 Sigmoid Function

6a. Sigmoid Function

$$f(z) = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}}$$

Range = Possible Outputs

(0, 1)



Used in: Hidden Layer, Output Layer for Classification

Sigmoid: It is used for classification

$$\sigma(x) = \frac{1}{1+e^{-x}} \qquad e = 2.71$$

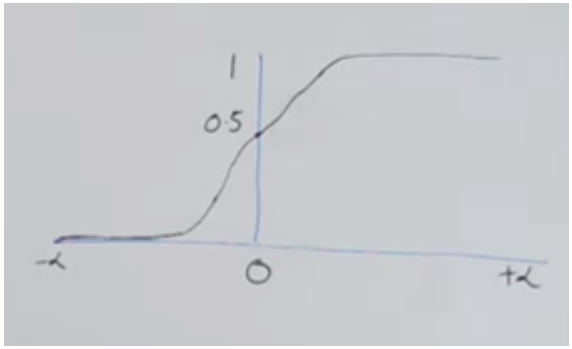
$x = 0$ $\quad \frac{1}{1+\frac{1}{e^0}} = \frac{1}{2} = 0.5$

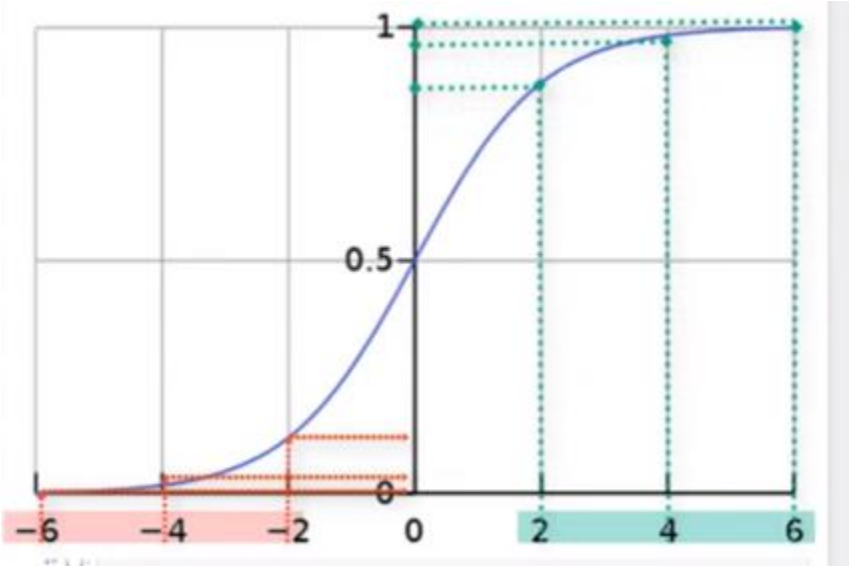
$x = 1$ $\quad \frac{1}{1+\frac{1}{e^1}} = 0.73$

$x = +\infty$ $\quad \frac{1}{1+\frac{1}{\infty}} = 1$

$x = -1$ $\quad \frac{1}{1+2.71} = 0.26$

$x = -\infty$ $\quad \frac{1}{1+\infty} = 0$





8. SoftMax function : It is the variant of sigmoid function with multi class classification

6b. Softmax Function

$$z_1, z_2, \cdots, z_n$$

$$f(z_1) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \cdots + e^{z_n}}$$

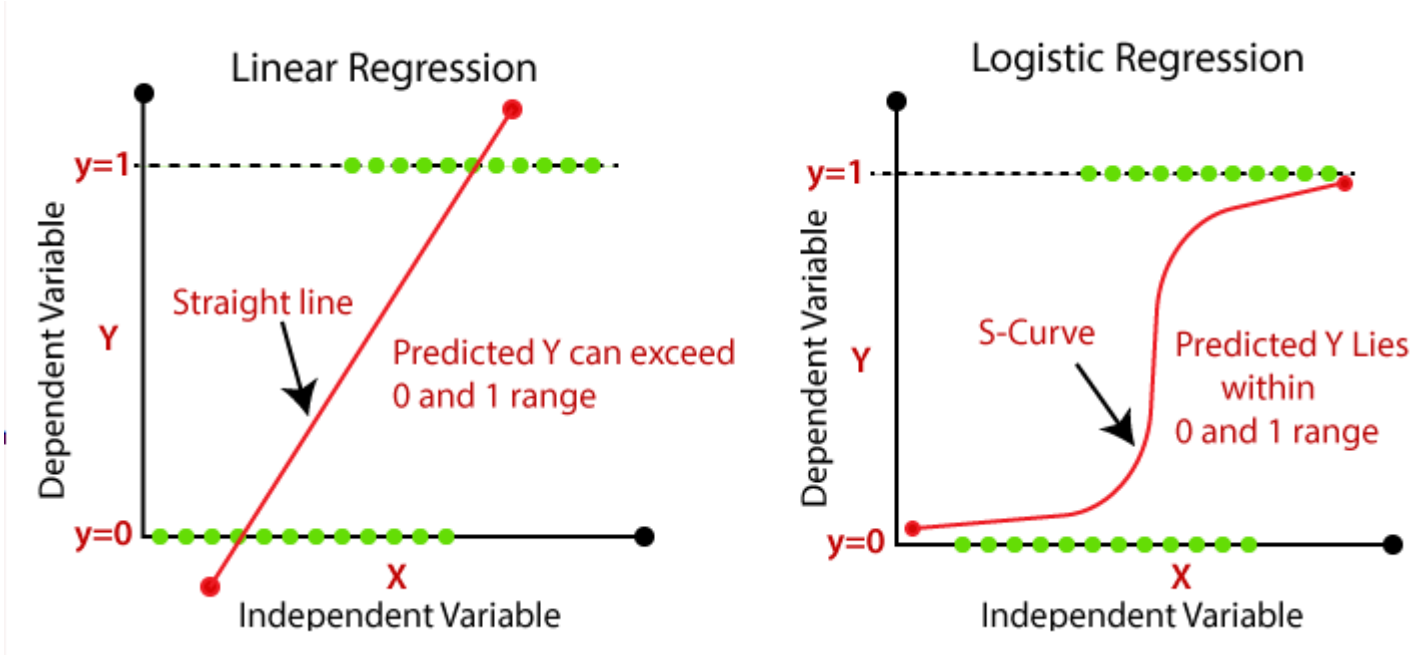
$$f(z_i) = \frac{e^{z_i}}{e^{z_1} + e^{z_2} + \cdots + e^{z_n}} = \frac{e^{z_i}}{\sum_{k=1}^{n} e^{z_k}}$$

Used in: Output Layer for Multiclass Classification

Logistic Regression

- **Linear Regression is used to handle regression problems whereas Logistic regression is used to handle the classification problems**.
- Linear regression provides a continuous output but Logistic regression provides discreet output

Compare linear vs Logistic regression



Linear Regression

- Linear Regression is one of the most simple Machine learning algorithm that comes under Supervised Learning technique and used for solving regression problems.
- It is used for predicting the continuous dependent variable with the help of independent variables.
- The goal of the Linear regression is to find the best fit line that can accurately predict the output for the continuous dependent variable

Logistic Regression

- Logistic regression is one of the most popular Machine learning algorithm that comes under Supervised Learning techniques.
- It can be used for Classification as well as for Regression problems, but mainly used for Classification problems.
- Logistic regression is used to predict the categorical dependent variable with the help of independent variables.

Training an MLP with TensorFlow's

- Training (60%)
- Validation (20%)
- Testing (20%)

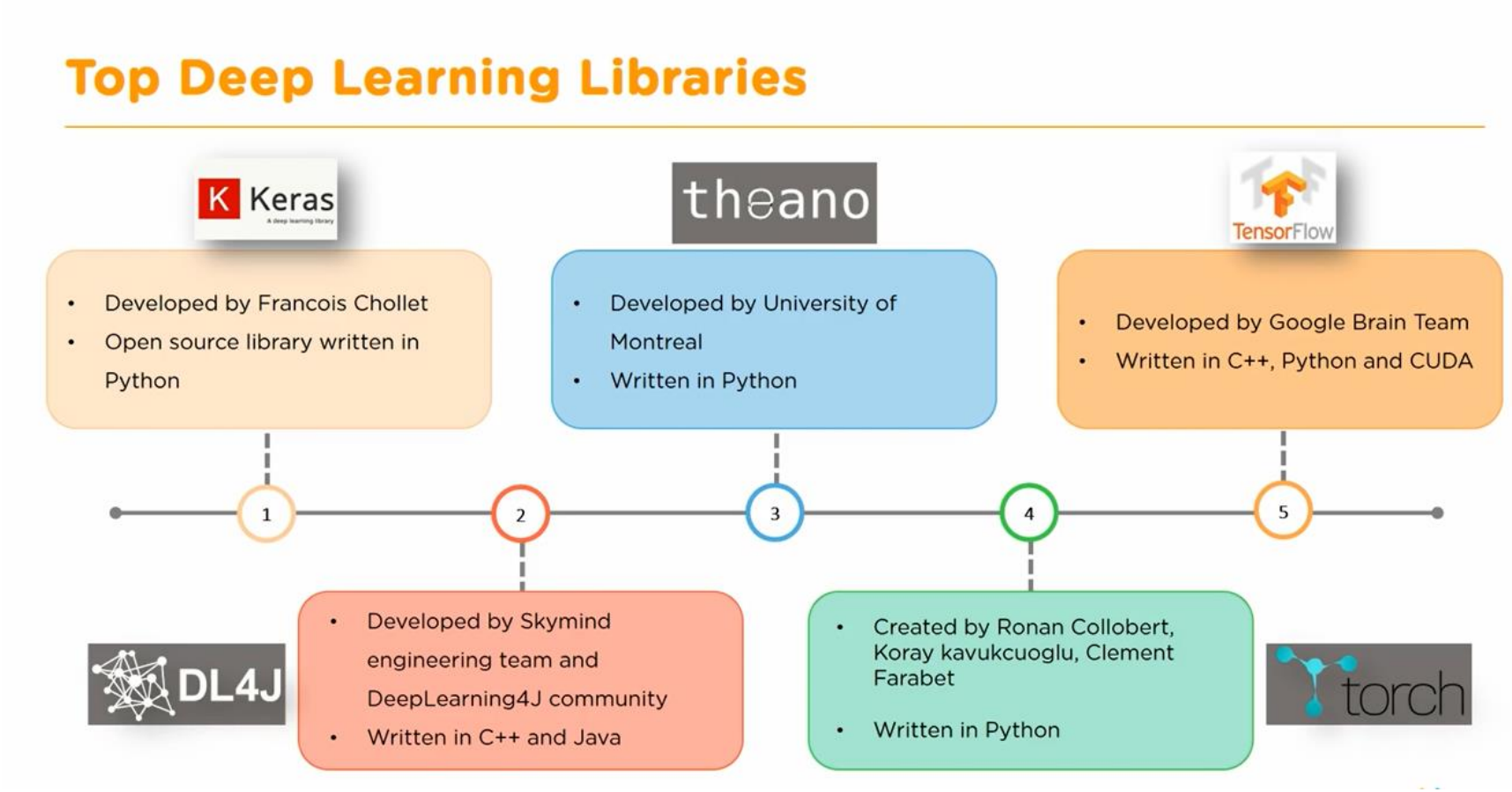
Tensorflow and Scikit-learn (SK learn)

- Scikit-learn (SK Learn) is a general-purpose machine learning library is better for traditional Machine Learning,
- While TensorFlow (tf) is positioned as a deep learning library is better for Deep Learning.
- The obvious and main difference is that TensorFlow does not provide the methods for a powerful feature engineering as sklearn such as dimensional compression, feature selection, etc.

How to work with DL algorithms?

- You need a programming language, preferred is Python.
- Lot of libraries available including Tensorflow.
- Others are Keras, Theano, torch, DL4J
- Tensorflow is from google and Keras is now embedded into Tensorflow.
- Tensorflow also supports traditional ML algorithms also.

What is Tensor Flow?

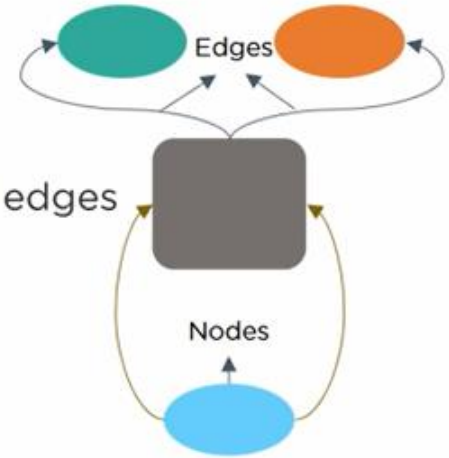


What is Tensorflow?

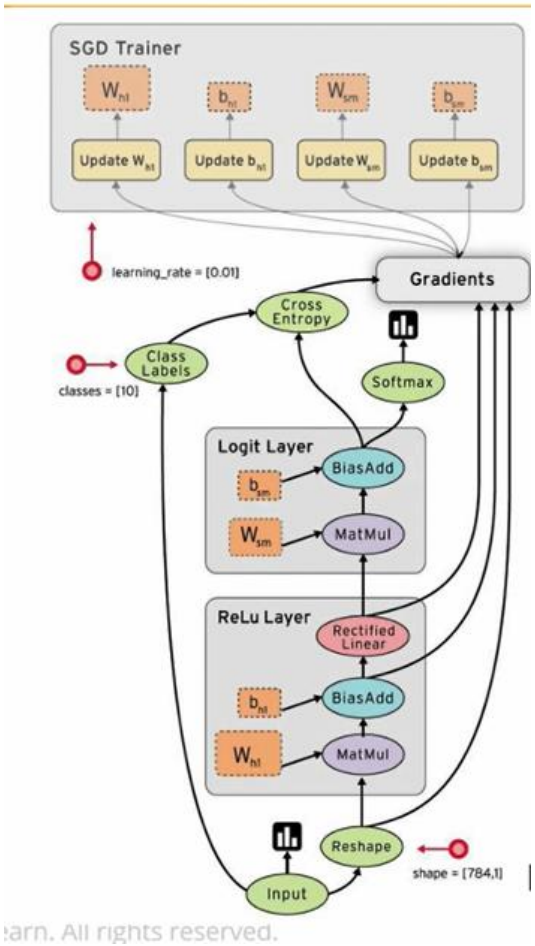
- It is from google
- It was originally developed for large numerical computations.
- Later it was introduced with ML and DL algorithms
- It accepts data in multidimensional array called "Tensor"

Tensorflow works on the basis of Dataflow graphs

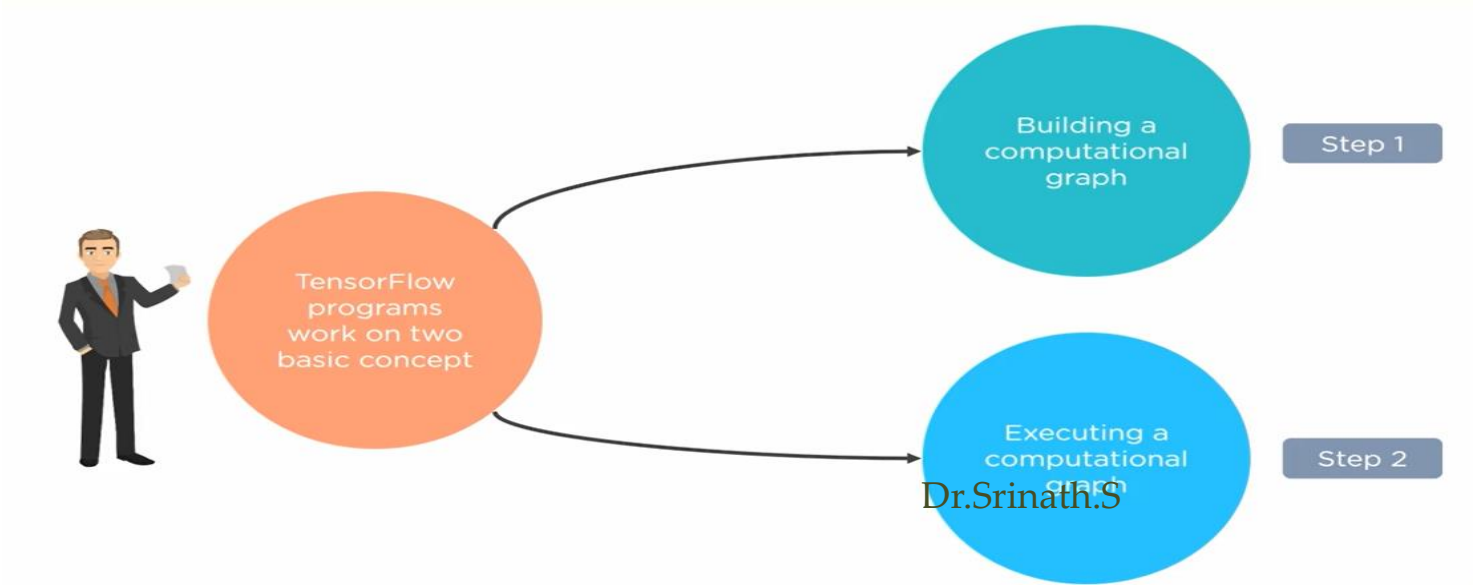
Works on the basis of Data Flow graphs that have nodes and edges



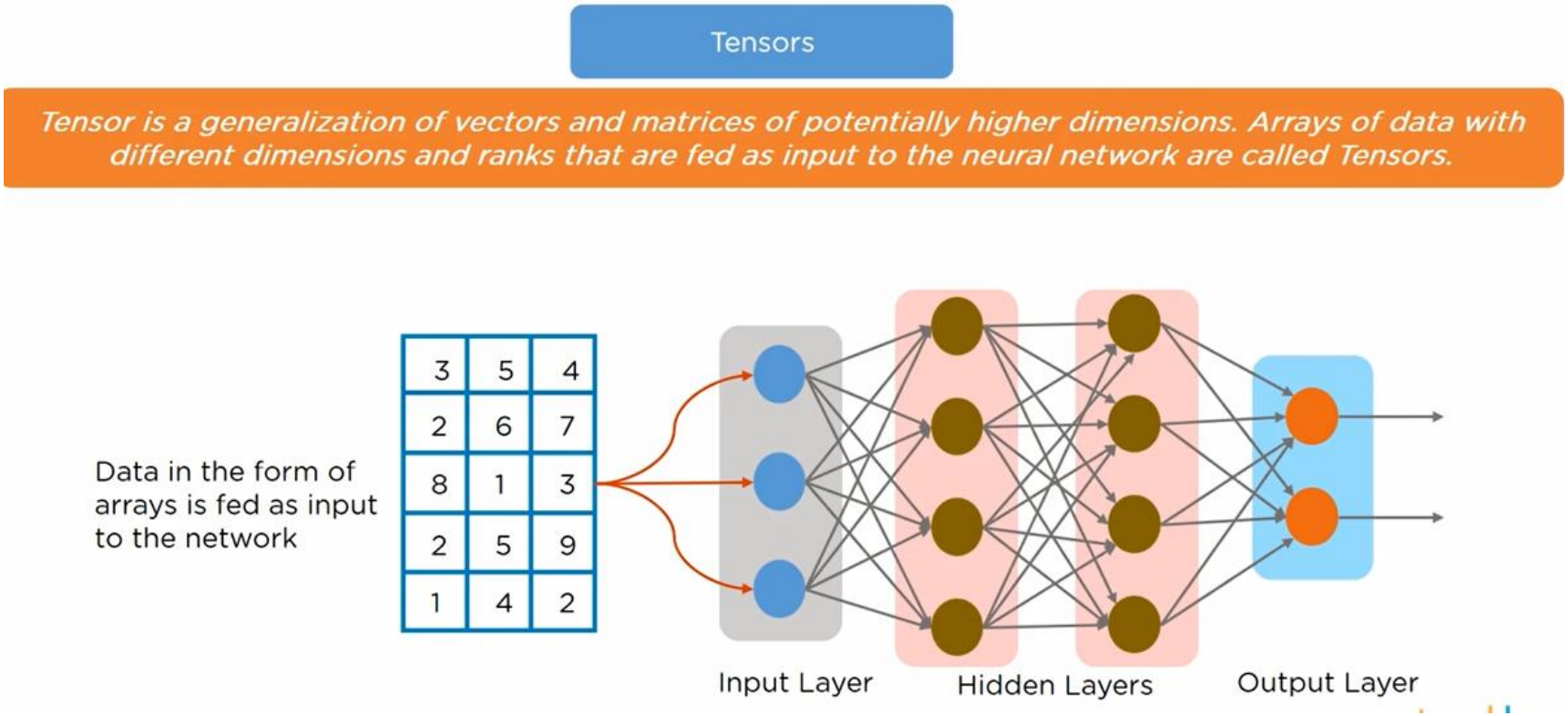
In tensor flow graphs are created and are executed by creating sessions



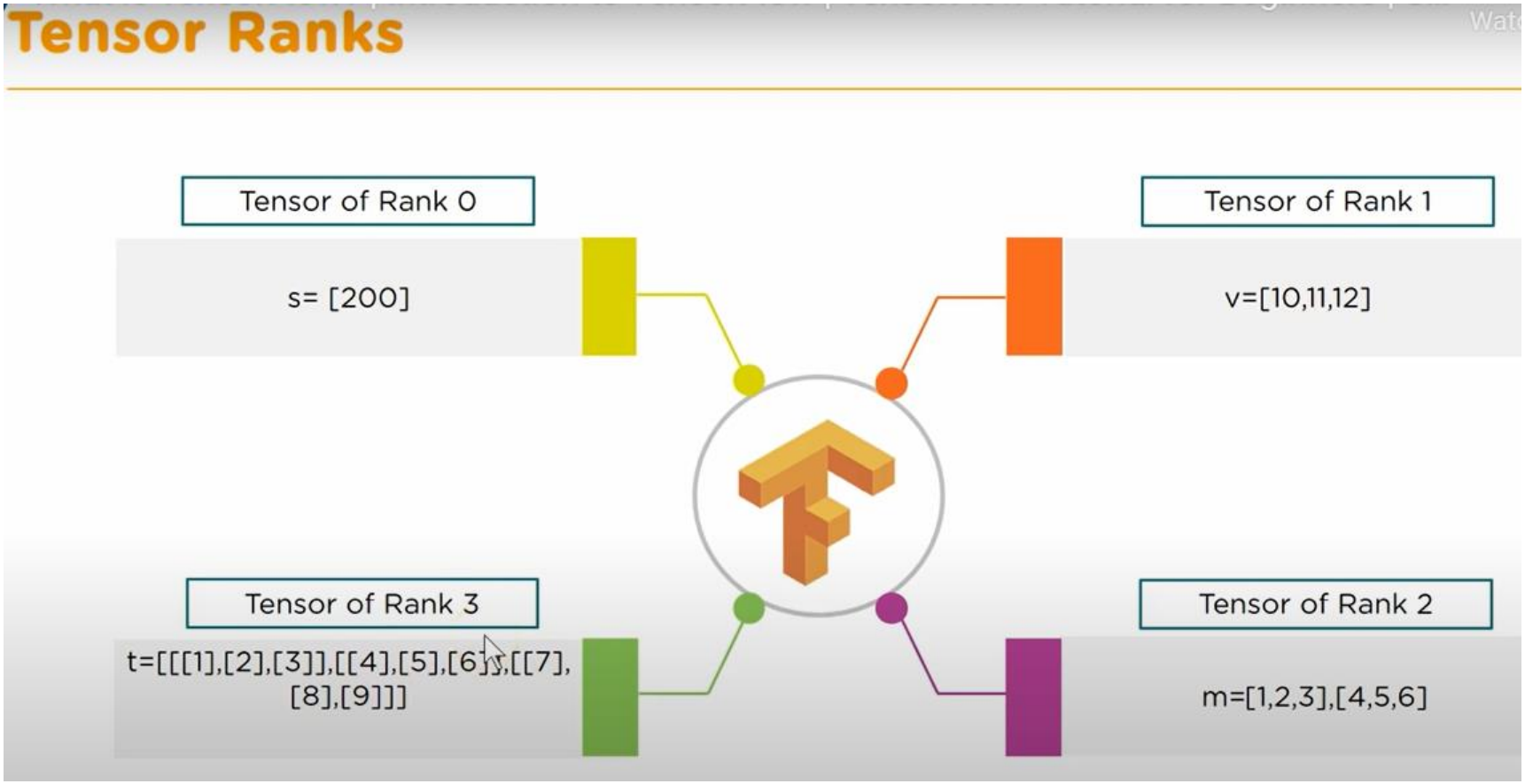
- All the external data at fed into what is know as placeholder, constants and variables.
- To summarize, Tensorflow starts building a computational graph and in the next step it executing the computational graph.



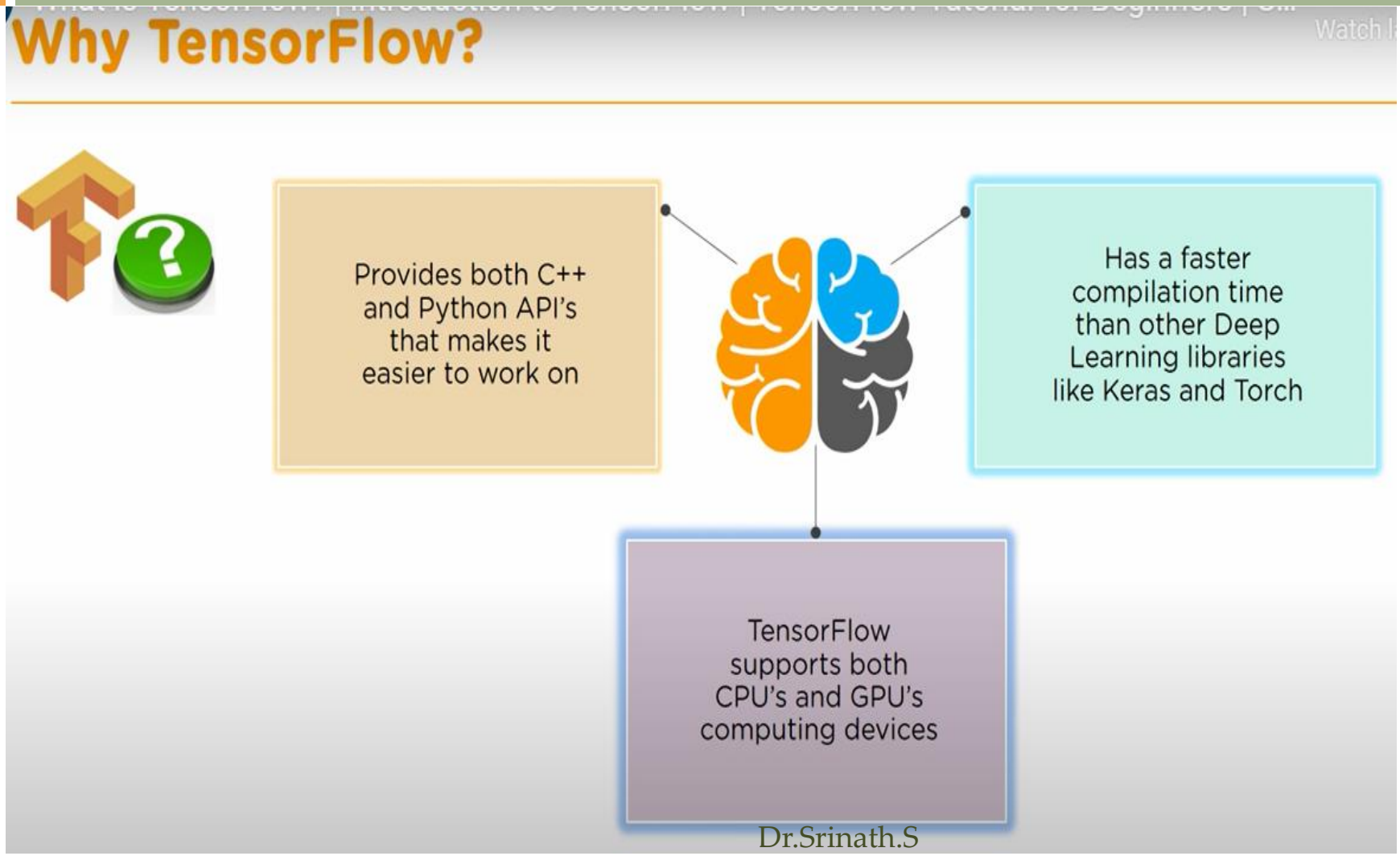
Tensors


Tensors
Tensor is a generalization of vectors and matrices of potentially higher dimensions. Arrays of data with different dimensions and ranks that are fed as input to the neural network are called Tensors.
Data in the form of arrays is fed as input to the network
3 5 4
2 6 7
8 1 3
2 5 9
1 4 2
Input Layer
Hidden Layers
Output Layer

Ranks (dimensions) of tensor



Why to use TensorFlow?



Components of Tensorflow: Constants

- Programming using Tensorflow is bit different from programming on SK Learn and also on python.
- In Tensorflow the storage consists of
  - Constants
  - Variables
  - Placeholders

Constants are parameters whose value does not change. To define a constant we use *tf.constant()* command.

Example:

```
a = tf.constant(2.0, tf.float32)
b = tf.constant(3.0)
Print(a, b)
```

Variables

- In variables, V must be in capital letters.
- Value of the variable can be changed.. But not of constant.

Variables allow us to add new trainable parameters to graph. To define a variable we use *tf.Variable()* command and initialize them before running the graph in a session.

Example:

```
W = tf.Variable([.3],dtype=tf.float32)
b = tf.Variable([-.3],dtype=tf.float32)
x = tf.placeholder(tf.float32)
linear_model = W*x+b
```

Placeholder

- They are used to feed the data from outside.
- Say from a file, from image file, CSV file and so on.
- Feed_dict id popularly used to feed the data to the placeholder.

Placeholders allow us to feed data to a tensorflow model from outside a model. It permits a value to be assigned later. To define a placeholder we use *tf.placeholder()* command.

Example:

```
a = tf.placeholder(tf.float32)
b = a*2
with tf.Session() as sess:
result = sess.run(b,feed_dict={a:3.0})

print result
```

- Constants, Variable and placeholder…
- Crate Graph using the above, then you will have session and session object and run it.
- Every computation you perform is a node in a graph.
- Initially tf object is created, which is the default graph, which will not have any constant, variable …

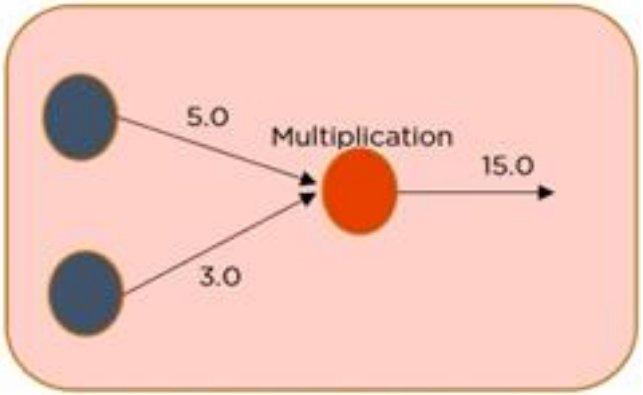
Running a session in Tensorflow

- Multiplication of ‘a’ and ‘b’ are done while running the session (last statement)

A *session* is run to evaluate the nodes. This is called as the *TensorFlow runtime*.

Example:

```
a = tf.constant(5.0)
b = tf.constant(3.0)
c = a*b
# Launch Session
sess = tf.Session()
# Evaluate the tensor c
print(sess.run(c))
```



Running a Computation Graph

Tensor flow – where to execute?

- **TensorFlow is already pre-installed**
- When you create a new notebook on colab.research.google.com, TensorFlow is already pre-installed and optimized for the hardware being used. Just import tensorflow as tf , and start coding.

Tensor flow can also be executed in Jupyter Notebook

- Inside the notebook, you can **import TensorFlow in Jupyter Notebook with the tf alias**. Click to run.

Training an MLP with TensorFlow's

- The simplest way to train an MLP with TensorFlow is to use the high-level API TF.Learn.
- The DNNClassifier class makes it trivial to train a deep neural network with any number of hidden layers, and a softmax output layer to output estimated class probabilities.
- For example, the following code trains a DNN for classification with two hidden layers (one with 300 neurons, and the other with 100 neurons) and a softmax output layer with 10 neurons.

Piece of code of for training MLP

```
import tensorflow as tf

feature_columns = tf.contrib.learn.infer_real_valued_columns_from_input(X_train)
dnn_clf = tf.contrib.learn.DNNClassifier(hidden_units=[300, 100], n_classes=10,
                                         feature_columns=feature_columns)
dnn_clf.fit(x=X_train, y=y_train, batch_size=50, steps=40000)
```

tf is tensorflow

Code creates set of real valued columns from the training set.
Then create the DNNClassifier, with two hidden layers of 300 and 100 neurons and with output layer of 10 neurons.
Finally program is run for 40,000 epochs in a batch of 50.

Fine tuning NN Hyper Parameters - Up and Running with TensorFlow

- In a simple MLP you can change the number of layers, number of neurons per layer, the type of activation function and also the weight initialization logic.
- The above are the Hyper Parameters to be fine tuned in a Neural Network.

Number of Hidden Layers

- For many problems, you can just begin with a single hidden layer and you will get reasonable results.
- It has actually been shown that an MLP with just one hidden layer can model even the most complex functions provided it has enough neurons.
- For a long time, these facts convinced researchers that there was no need to investigate any deeper neural networks.
- But they overlooked the fact that deep networks have a much higher parameter efficiency than shallow ones.
- They can model complex functions using exponentially fewer neurons than shallow nets, making them much faster to train

Number of hidden layers..cont

- Very complex tasks, such as large image classification or speech recognition, typically require networks with dozens of layers (or even hundreds) and they need a huge amount of training data.
- However, you will rarely have to train such networks from scratch: it is much more common to reuse parts of a pretrained state-of-the-art network that performs a similar task. Training will be a lot faster and require much less data

Number of Neurons per Hidden Layers

- Usually the number of neurons in the input and output layers is determined by the type of input and output your task required.
- For the hidden layer the common practice is to size them to form a funnel, with fewer and fewer neurons at each layer.
- For example a typical neural network may have two hidden layers, the first with 300 neurons and the second with 100.
- However, this practice is not as common now, and you may simply use the same size for all hidden layers; for example all hidden layers with 150 neurons.
- Neurons can be gradually increased until the network starts overfitting.

Activation Functions

- In most of the cases you can use the ReLU activation function in the hidden layers. It is a bit faster to compute than other activation functions.
- For the output layer, the softmax activation function is generally a good choice for classification tasks.

End of Unit - 1

UNIT - 2

Deep Neural Network

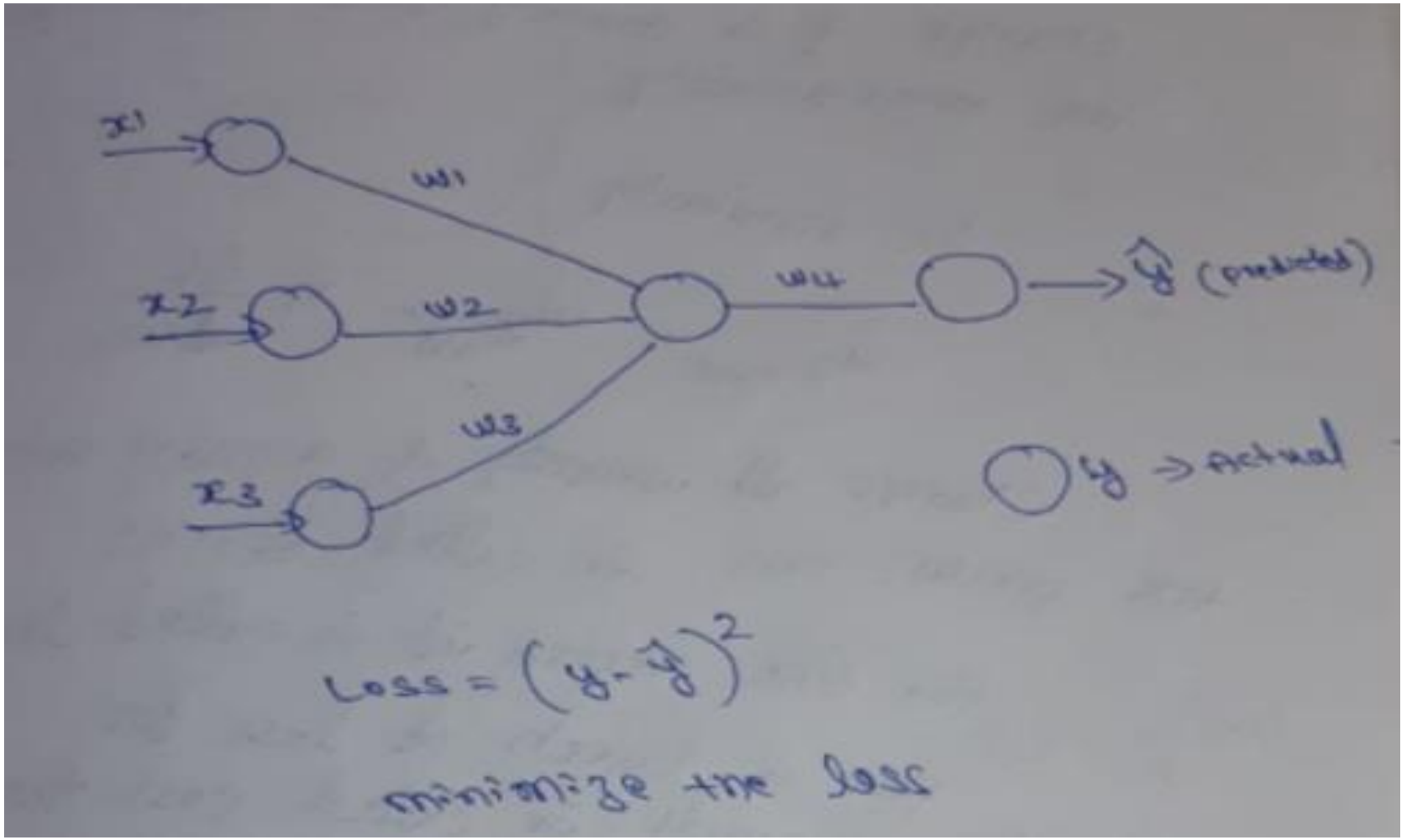
Unit – 2 : Syllabus

- Deep Neural network:
- Introduction,
- Vanishing Gradient problems,
- Reusing Pretrained layers,
- Faster optimizers,
- avoiding over fitting through regularization

Deep Neural Networks

- **Introduction:**
- Neural network with 2 or more hidden layers, can be called Deep Neural Network.
- While handling a complex problem such as detecting hundreds of types of objects in high resolution images, you may need to train a much deeper DNN, perhaps say 10 layers, each containing hundreds of neurons, connected by hundreds of thousands of connections.
- This leads to a problem of vanishing gradients.

Training the Neural Network

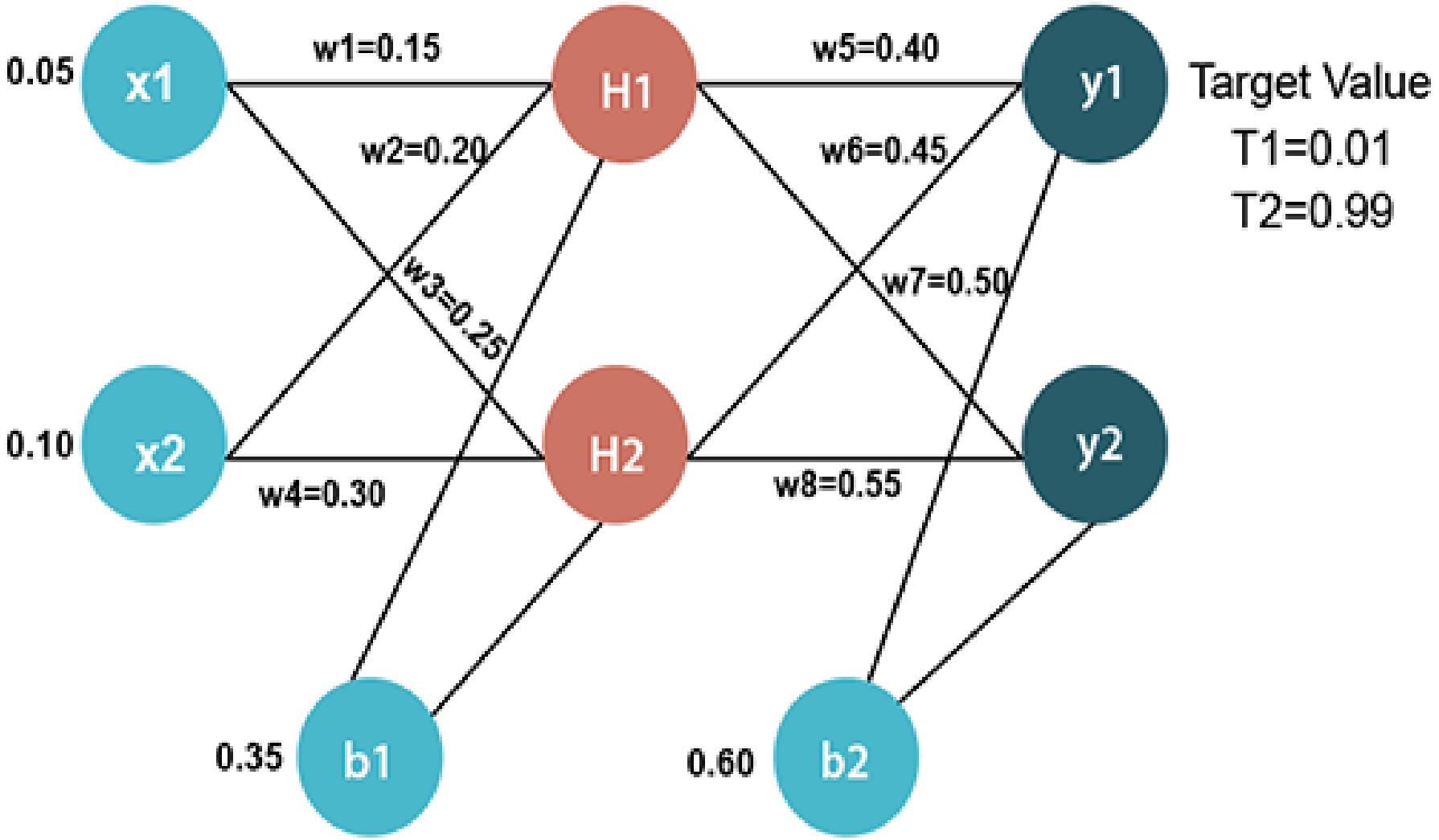


Back propagation

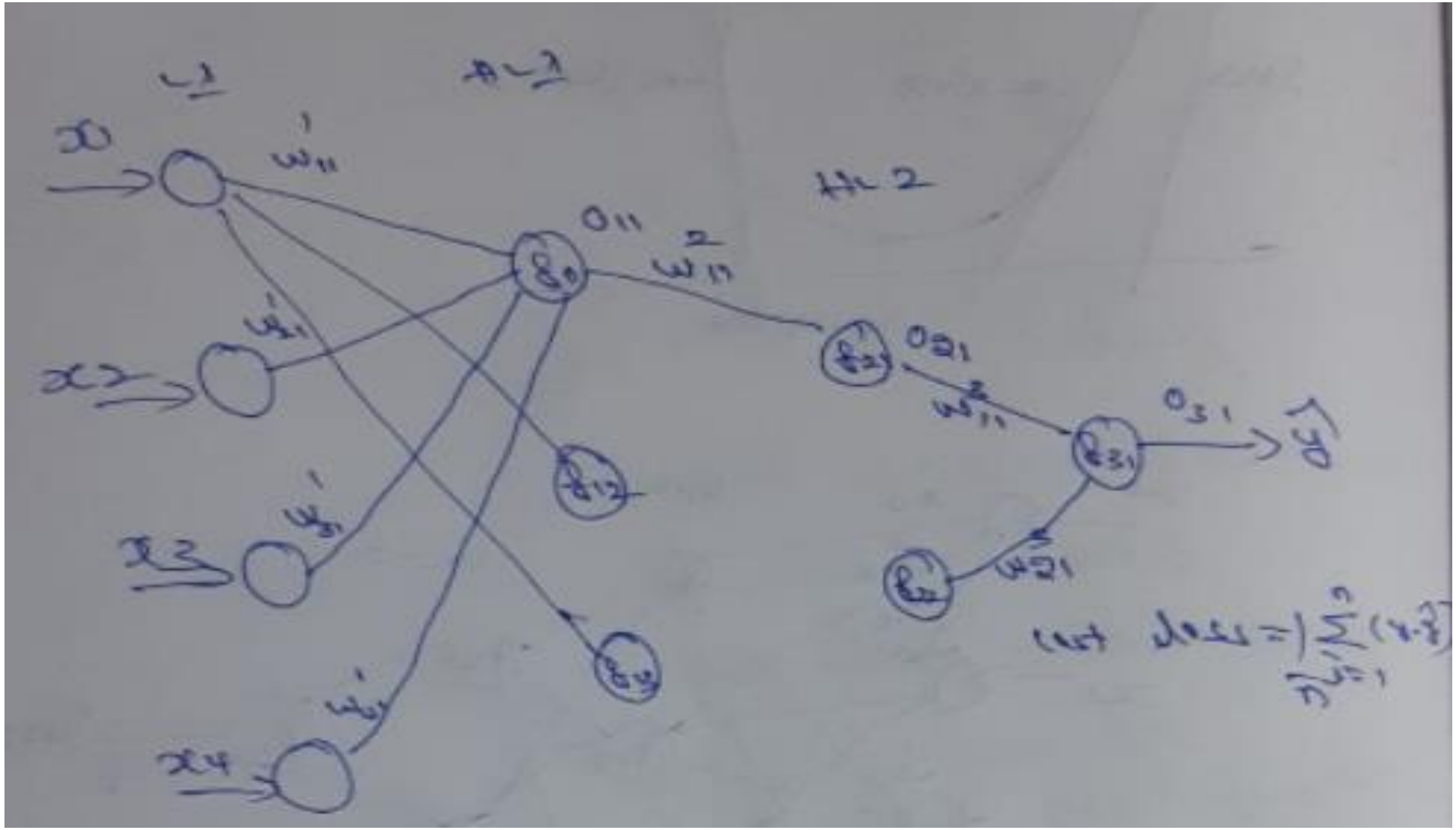
- **Backpropagation** is the essence of neural network training. It is the method of fine-tuning the weights of a neural network based on the error rate obtained in the previous epoch (i.e., iteration).
- Proper tuning of the weights allows you to reduce error rates and make the model reliable by increasing its generalization.

- Backpropagation in neural network is a short form for “backward propagation of errors.” It is a standard method of training artificial neural networks. This method helps calculate the gradient of a loss function with respect to all the weights in the network.

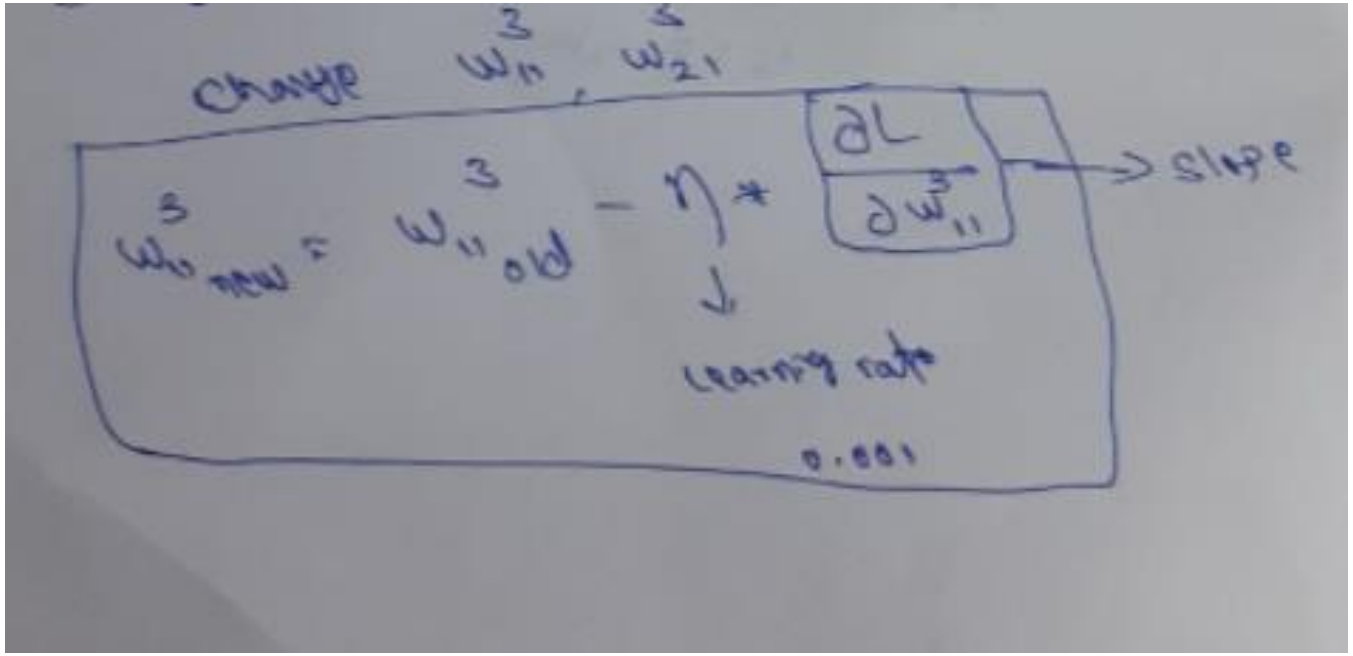
Back propagation is the method of adjusting weights after computing the loss value.



Chain rule in backpropagation



Backpropagate to change the weight

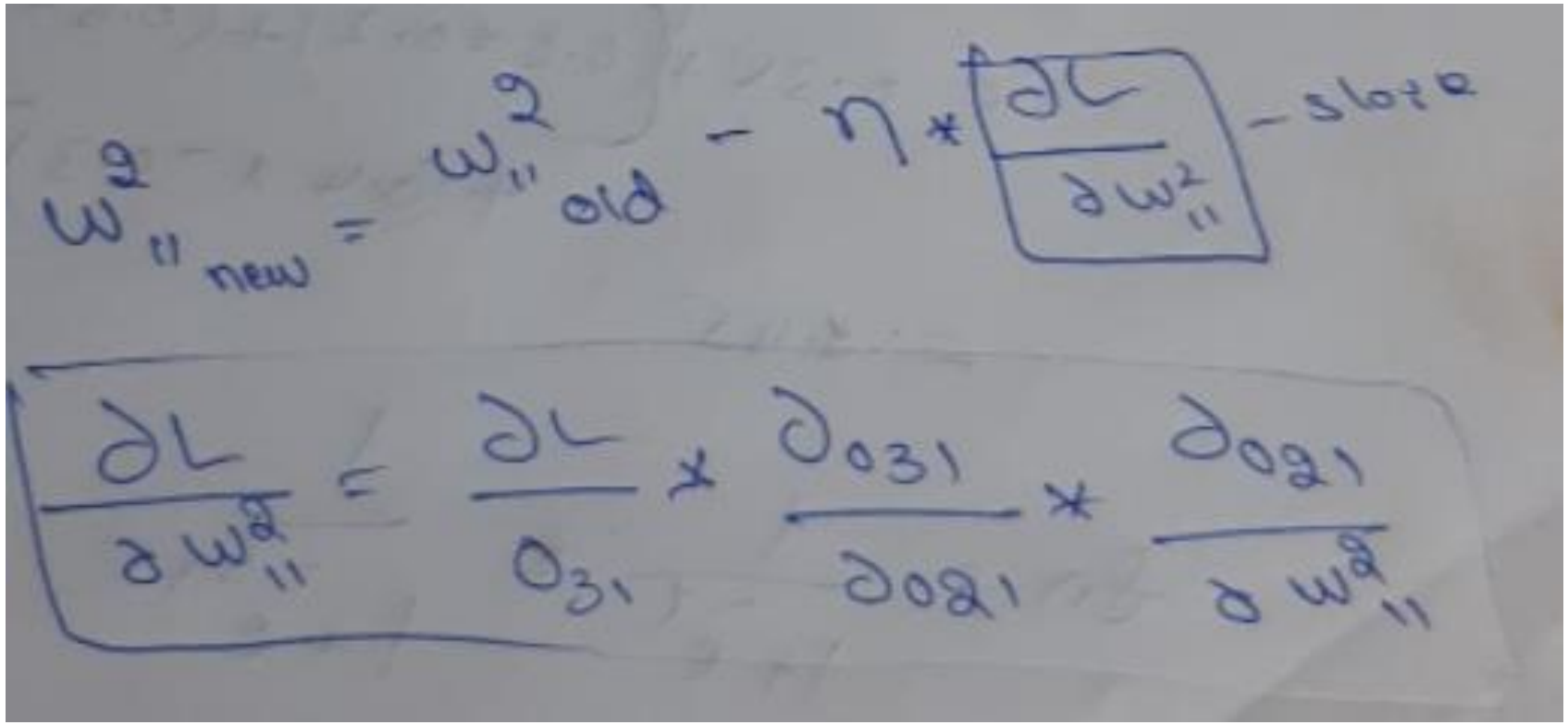


$$\frac{\partial L}{\partial w^3_{11}} = \frac{\partial L}{\partial o_{31}} * \frac{\partial o_{31}}{\partial w^3_{11}}$$

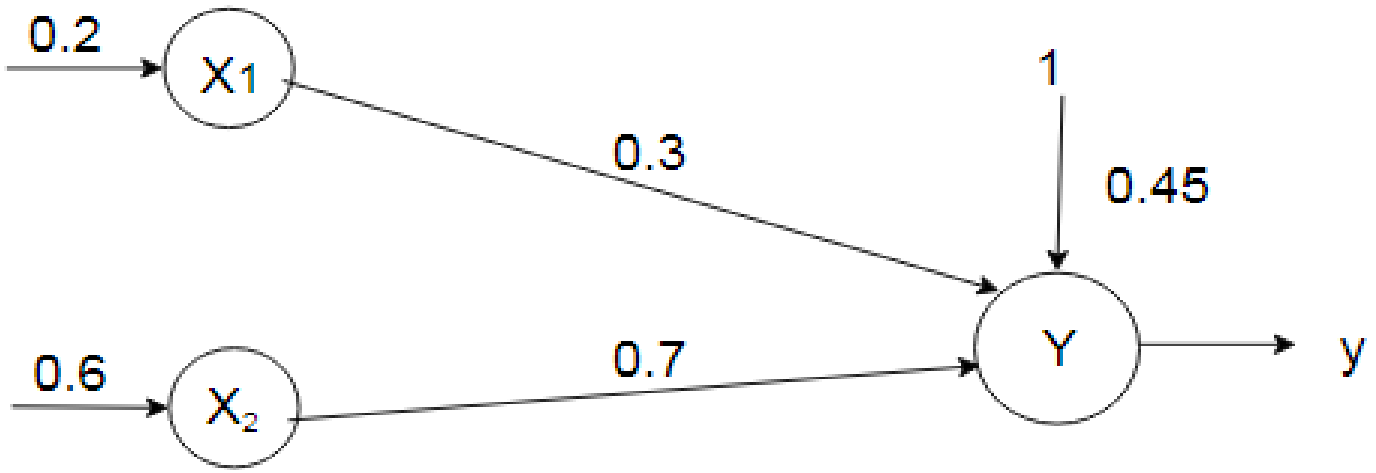
Further

$$\frac{\partial L}{\partial w^3_{21}} = \frac{\partial L}{\partial o_{31}} * \frac{\partial o_{31}}{\partial w_{21}}$$

Similarly :


$w^2_{11\,new} = w^2_{11\,old} - \eta * \frac{\partial L}{\partial w^2_{11}}$ - slope
$\frac{\partial L}{\partial w^2_{11}} = \frac{\partial L}{o_{31}} * \frac{\partial o_{31}}{\partial o_{21}} * \frac{\partial o_{21}}{\partial w^2_{11}}$

Q2. Calculate the net input for the network shown in figure with bias included in the network?



Solution: $[x_1, x_2, b] = [0.2, 0.6, 0.45]$

$[w_1, w_2] = [0.3, 0.7]$

The net input can be calculated as,

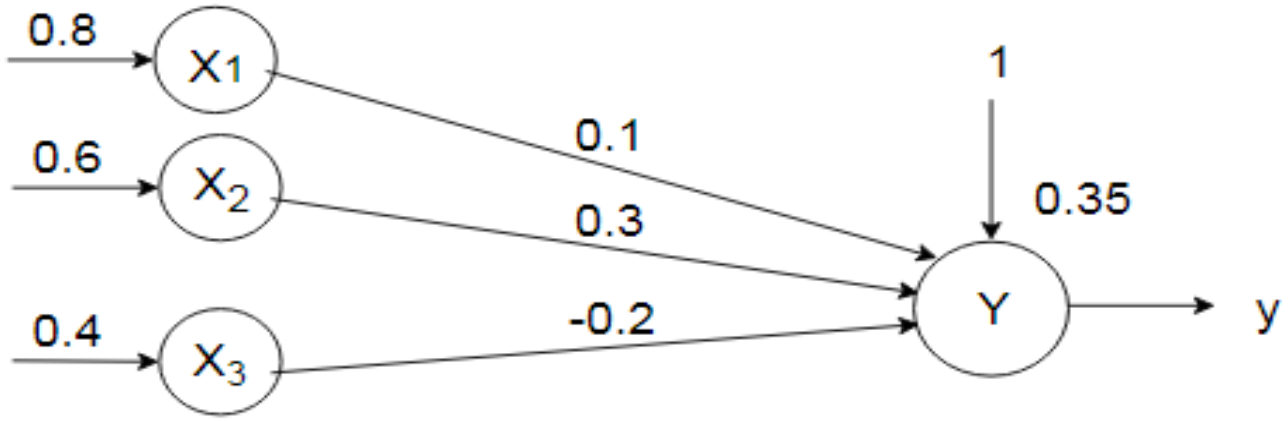
$y_{in} = b + x_1 w_1 + x_2 w_2$

$y_{in} = 0.45 + 0.2 \times 0.3 + 0.6 \times 0.7$

$y_{in} = 0.45 + 0.06 + 0.42 = 0.93$

Problem: (to compute Y out): apply activation function

- Calculate the output ‘y’ of a 3 input neuron with bias, with data as given in the diagram. Use sigmoidal activation function



$$[x_1, x_2, x_3] = [0.8, 0.6, 0.4]$$

The weights are,

$$[w_1, w_2, w_3] = [0.1, 0.3, -0.2]$$

The net input can be calculated as,

$$y_{in} = b + \sum_{i=1}^{n} (x_i w_i)$$

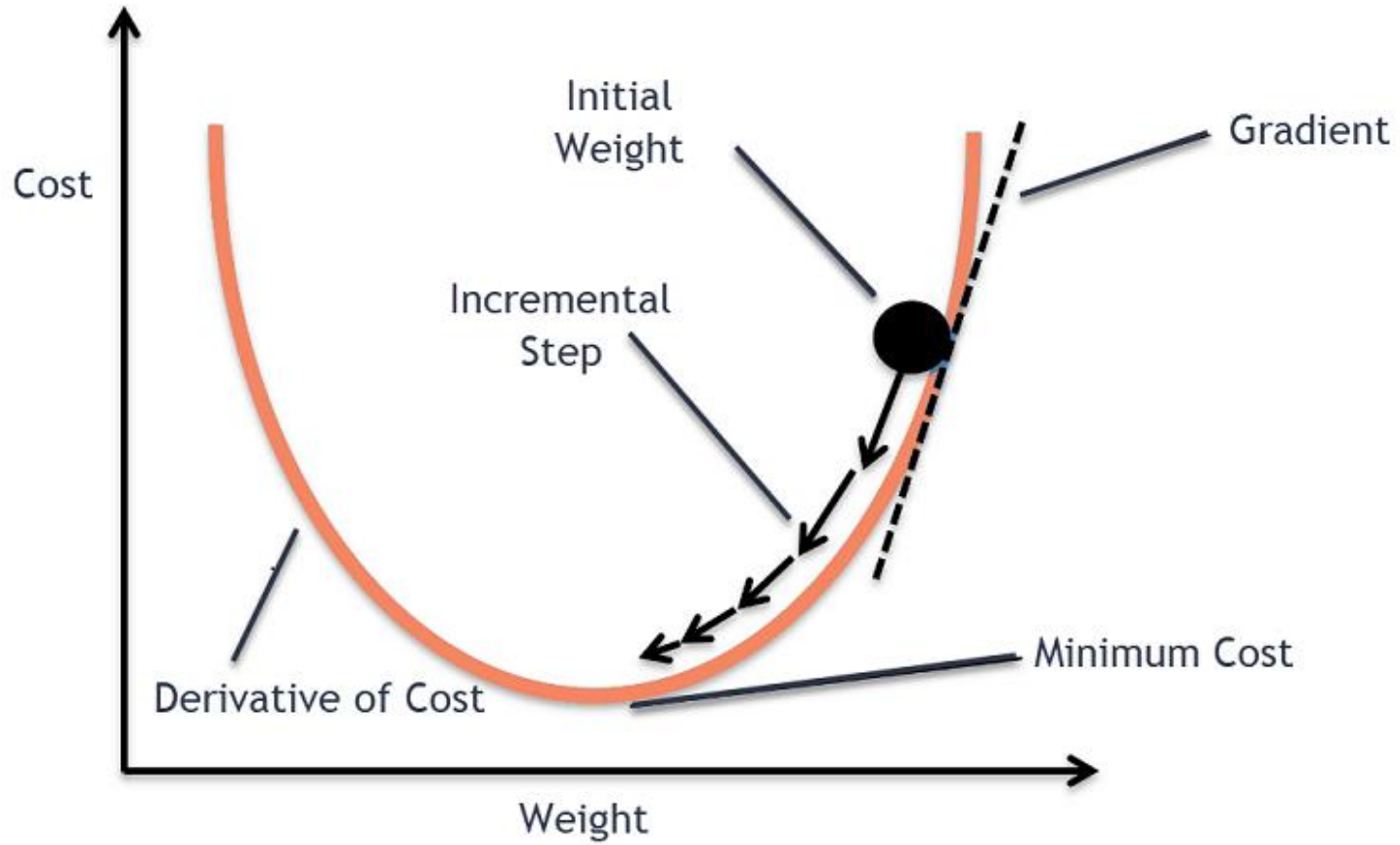
$$y_{in} = 0.35 + 0.8 \times 0.1 + 0.6 \times 0.3 + 0.4 \times (-0.2)$$

$$y_{in} = 0.35 + 0.08 + 0.18 - 0.08 = 0.53$$

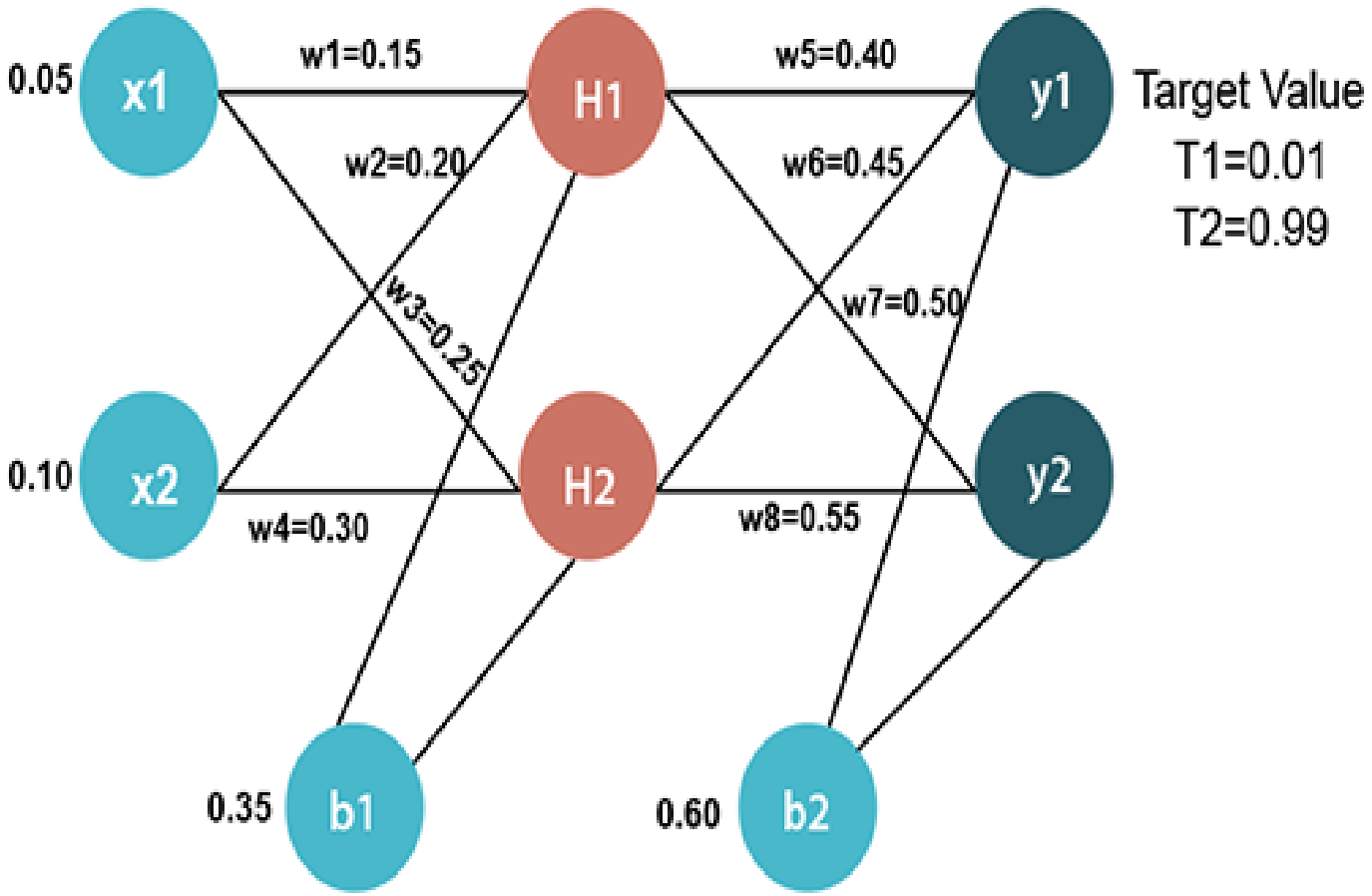
(i) For Binary Sigmoidal Function,

$$y = f(y_{in}) = \frac{1}{1 + e^{-y_{in}}} = \frac{1}{1 + e^{-0.53}} = 0.62$$

Gradient Descent



Compute the weights



Input values

X1=0.05

X2=0.10

Initial weight

W1=0.15 w5=0.40

W2=0.20 w6=0.45

W3=0.25 w7=0.50

W4=0.30 w8=0.55

Bias Values

b1=0.35 b2=0.60

Target Values

T1=0.01

T2=0.99

The technique used to determine how much a weight can be changed is known as gradient descent method

Forward propagation:

net input for $h_1 = (x_1 * w_1) + (x_2 * w_2) + b_1$

$= [(0.05 * 0.15) + (0.10 * 0.20)] + 0.35$

$= 0.3775$

output of h_1 after applying Sigmoidal activation function

$f(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-0.3775}} = 0.595$

h_1 output = 0.593269992

$h_2 = [(x_1 * w_3) + (x_2 * w_4)] + b_1$

$= [(0.05 * 0.25) + (0.10 * 0.30)] + 0.35$

$= 0.3925$

$h_2 = f(z) = \frac{1}{1 + e^{0.3925}} = \frac{1}{2.480677865}$

$= \frac{1}{1.675366346} = 0.4031156821$

$= 0.596884$

Similarly

$o1 = (w5 * h1) + (w6 * h2) + b2$

$= (0.40 * 0.593269992) +$

$(0.45 * 0.596884378) + 0.6$

$= 1.105909967$

$out\ o1 = \frac{1}{1 + e^{-1.105909967}} = \frac{1}{1 + 0.330910952}$

$= \frac{1}{1.330910952}$

$\boxed{o1 = 0.751365069}$

$o2 = (w7 * h1) + (w8 * h2) + b2$

$= (0.5 * 0.593269992) +$

$(0.55 * 0.596884378) + 0.6$

$= 1.224921404$

$out\ o2 = \frac{1}{1 + e^{-1.224921404}}$

$\boxed{o2 = 0.772928465}$

Target

$T_1 = 0.01$

$T_2 = 0.99$

output

$y_{1,out} = 0.751365507$

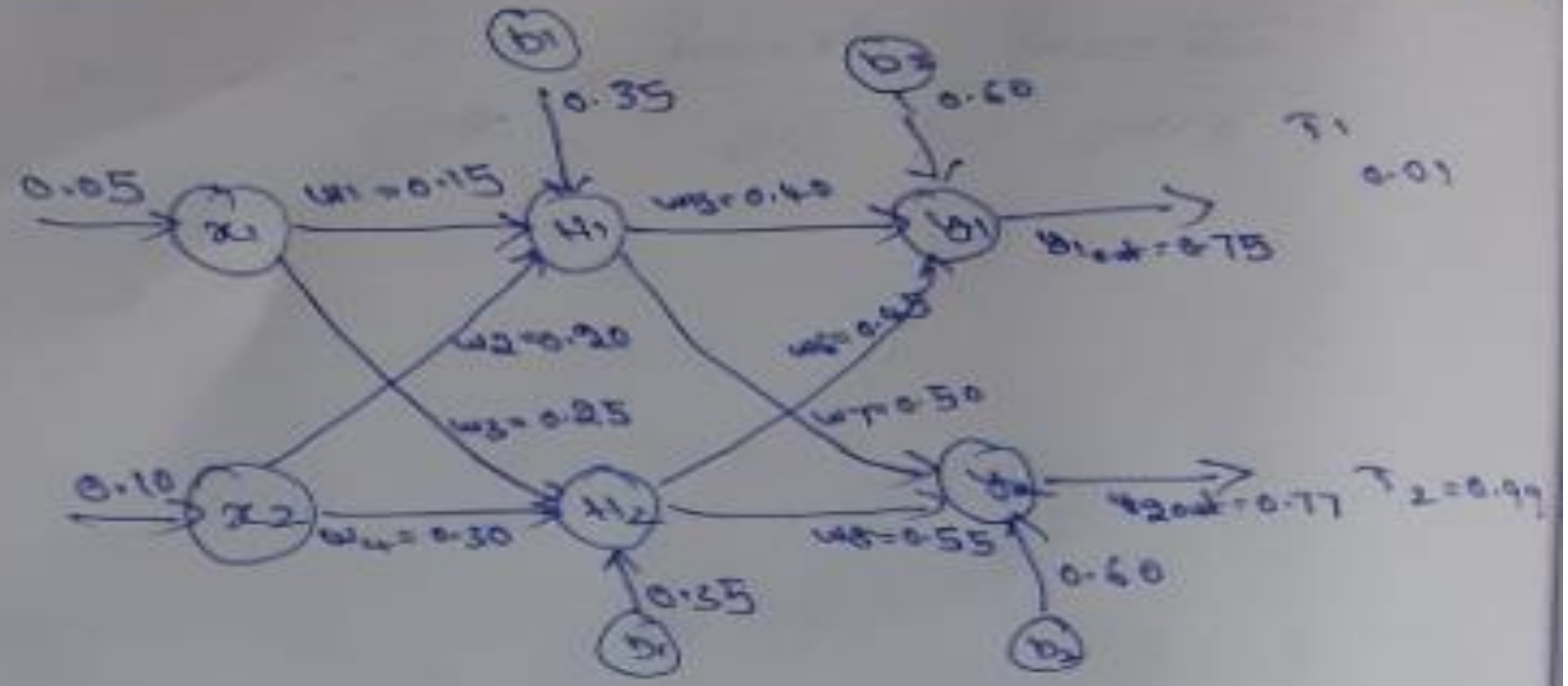
$y_{2,out} = 0.772928465$

$$\text{Error} = \sum \frac{1}{2}(\text{target} - \text{output})^2$$

$$= \frac{1}{2}(0.01 - 0.751365507)^2 + \frac{1}{2}(0.99 - 0.772928465)^2$$

$$= 0.274811084 + 0.023560025?$$

$$E_{total} = 0.298371111$$



T1
0.01

$y_{1out} = 0.75$

$y_{2out} = 0.77$ $T_2 = 0.99$

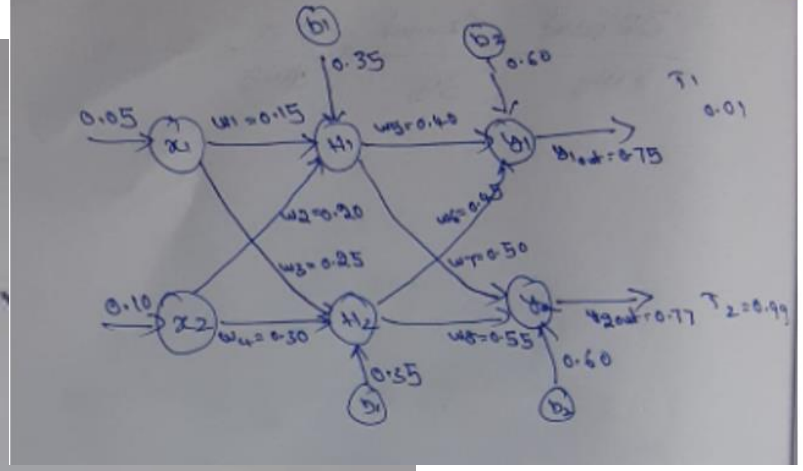
20

$y_1 = 1.105905967$

$y_2 = 1.224921404$

$y_{1out} = 0.751365069$

$y_{2out} = 0.772928465$



compute

$$\frac{\partial E_{total}}{\partial w_5}$$

It is not possible, because w_5 is not present in E_{total}, so

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial y_{1,out}} * \frac{\partial y_{1,out}}{\partial y_{1,}} * \frac{\partial y_{1,}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial y_{1,out}} = \frac{\partial \left(\frac{1}{2}(T_1 - y_{1,out})^2 + \left(\frac{1}{2}(T_2 - y_{2,out})^2\right)\right)}{\partial y_{1,out}}$$

$$= 2 * \frac{1}{2}(T_1 - y_{1,out})^{2-1} * (0-1) + 0$$

$$= (T_1 - y_{1,out}) * (-1)$$

$$= -(T_1 - y_{1,out})$$

$$= -(0.01 - 0.751365069)$$

$$\boxed{\frac{\partial E_{total}}{\partial y_{1,out}} = 0.741365069}$$

$$\frac{\partial y_{1,out}}{\partial y_1} = \frac{\partial\left(\frac{1}{(1+e^{-y_1})}\right)}{\partial y_1}$$

$$= \frac{e^{-y_1}}{(1+e^{-y_1})^2}$$

$$= e^{-y_1} \times \frac{1}{(1+e^{-y_1})^2}$$

$$= e^{-y_1} \times (y_{1,out})^2 \quad -(1)$$

$$y_{1,out} = \frac{1}{1+e^{-y_1}}$$

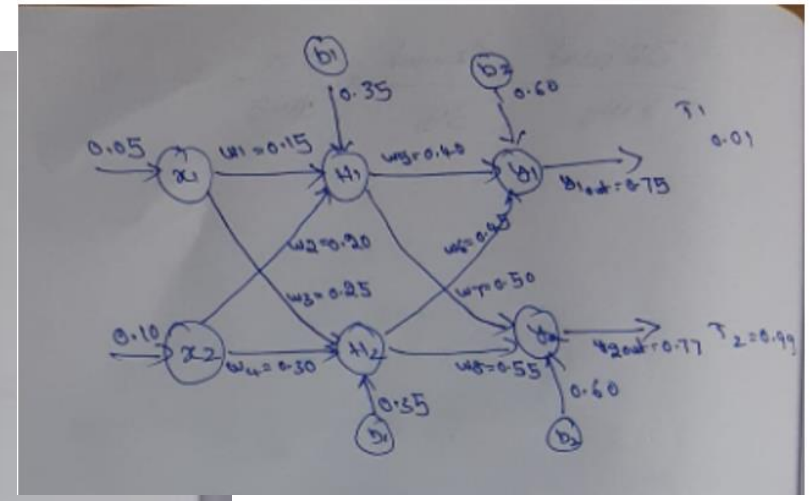
$$y_{1,out}(1+e^{-y_1}) = 1$$

$$y_{1,out} + y_{1,out} e^{-y_1} = 1$$

$$y_{1,out} e^{-y_1} = 1 - y_{1,out}$$

$$e^{-y_1} = \frac{1 - y_{1,out}}{y_{1,out}} \quad -(2)$$

replace (1) in (2)



$$\frac{\partial y_{1,out}}{\partial y_1} = e^{-y_1} * (y_{1,out})^2$$

$$= \left(\frac{1 - y_{1,out}}{y_{1,out}}\right) * (y_{1,out})^2$$

$$\boxed{\frac{\partial y_{1,out}}{\partial y_1} = y_{1,out}(1 - y_{1,out})}$$

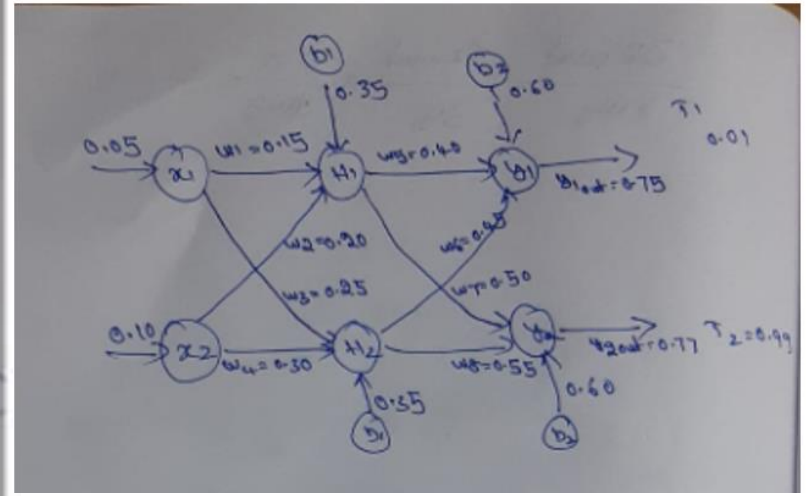
$$= 0.75136507 * (1 - 0.75136507)$$

$$\boxed{\frac{\partial y_{1,out}}{\partial y_1} = 0.18681562}$$

$$\frac{\partial y_1}{\partial w_5} = \frac{\partial (H_{1,out} * w_5 + H_{2,out} * w_6 + b_2)}{\partial w_5}$$

Simplify

$$\boxed{\frac{\partial y_1}{\partial w_5} = H_{1,out} = 0.593269992}$$



So

$$\frac{\delta E_{total}}{\delta w_5} = \frac{\delta E_{total}}{\delta y_{o1,out}} * \frac{\delta y_{o1,out}}{\delta y_{o1}} * \frac{\delta y_{o1}}{\delta w_5}$$

$$= 0.741365907 * 0.18681 * 0.5932$$

$$\boxed{\frac{\delta E_{total}}{\delta w_5} = 0.082167041407}$$

new w_5 weight

$$= \boxed{w_{5\,new} = w_5 - \eta * \frac{\delta E_{total}}{\delta w_5}}$$

consider $\eta = 0.5$

$$= w_{5\,new} = 0.4 - 0.5 * 0.082167041$$

$$\boxed{w_{5\,new} = 0.35891648}$$

Similarly change other weights.

Vanishing Gradient

- Back propagation algorithm works by going from the output layer to the input layer, propagating the error gradient on the way.
- Gradients often get smaller and smaller as the algorithm progresses down to the lower layers.
- As a result the gradient descent updates in the lower layer weights virtually unchanged.
- This makes training never converges to a good solution.
- This is called vanishing gradient.

Exploding Gradients

- Some times the gradients can grow bigger and bigger, so many layers gets large weight updates and the algorithm diverges.
- This is called exploding Gradients, which is popularly seen in recurrent neural networks.

Gradient clipping

- A popular technique to lessen the exploding gradients problem is to simply clip the gradients during backpropagation so that they never exceed some threshold (this is mostly useful for recurrent neural networks)

Reusing Pretrained Layers

- Saved models with its weights after training
- They are usually a very deep Neural Network models
- Trained on very large and generalized dataset used on large classification tasks.

Reusing Pretrained Layers

- It is generally not a good idea to train a very large DNN from scratch:.
- instead you should always try to find an existing neural network that accomplishes a similar task to the one you are trying to tackle, .
- then just reuse the lower layers of this network this is called Transfer Learning.
- It will not only speed up training considerably, but will also require much less training data.

Reusing Pretrained Layers

- For example, suppose that you have access to a DNN that was trained to classify pictures into 100 different categories, including animals, plants, vehicles, and everyday objects.
- You now want to train a DNN to classify specific types of vehicles. These tasks are very similar, so you should try to reuse parts of the first network

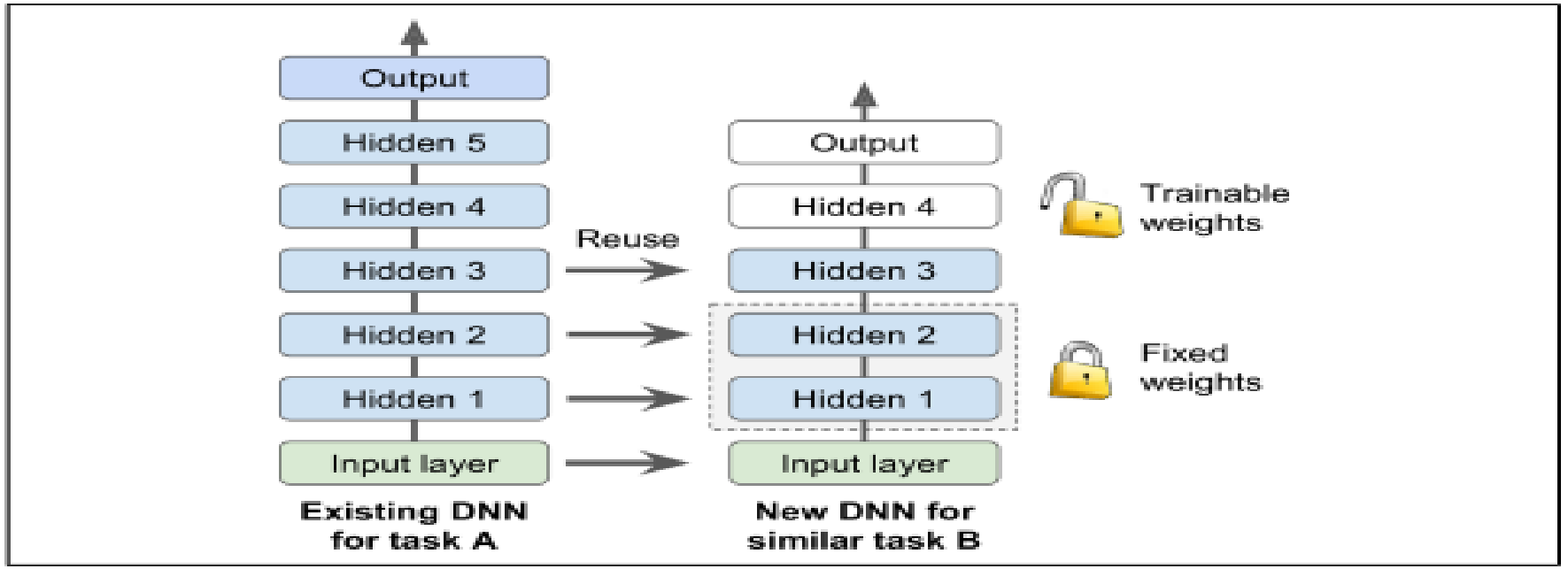


Figure 11-4. Reusing pretrained layers

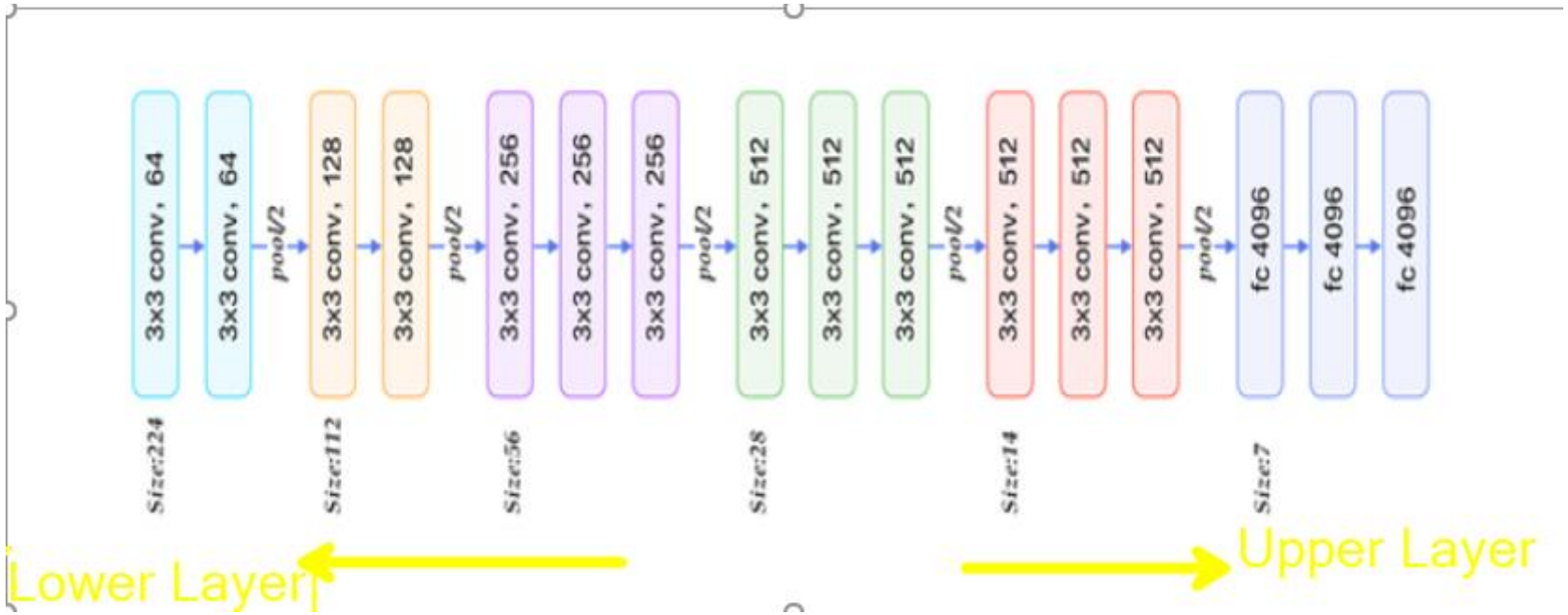
Observations to be made While training the lower layers.

- If the input pictures of your new task don’t have the same size as the ones used in the original task, you will have to add a preprocessing step to resize them to the size expected by the original model. More generally, transfer learning will work only well if the inputs have similar low-level features.

Pre-trained Models – How it is useful?

- Lower layers learn basic features from very large and generalized training images like color, lines in various angles
- These lower layer features are almost same for our most of our task, changes will be made in the upper layers, Hence training will be made for the upper layers.
- Examples of Pre-trained models:
- Xception
- VGG16
- VGG19
- ResNet50
- InceptionV3
- InceptionResNetV2
- MobileNet
- MobileNetV2
- DenseNet
- NASNet

VGG 16 Layers.

- 

Freezing the lower layers

- It is likely that the lower layer of the first DNN have learned to detect low –level features in pictures that will be useful across both image classification tasks, so you can just reuse these layers as they are.

- It is generally good idea to “Freeze” their weights.
- If the lower layers weights are fixed, then the higher layers weights will be easier to train.

Tweaking, Dropping or replacing the upper layers

- The output layers of the original model should usually be replaced since it is most likely not useful at all for the new task and it may not even have the right number of outputs for the new task.

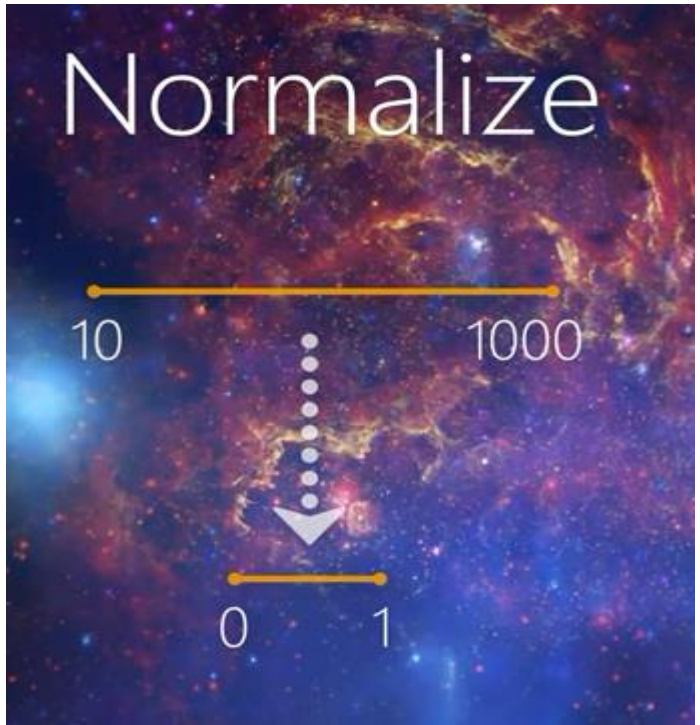
- If performance is not good you may drop or replace the upper layer for god performance.

Normalization

- In some cases, feature will have varied values, which may lead to inconsistent results.
- For example, observe the following table: where wine quality is being verified.
- However the values of “Alcohol” and “Malic” are having huge difference.

	Class	Alcohol	Malic
0	1	14.23	1.71
1	1	13.20	1.78
2	1	13.16	2.36
3	1	14.37	1.95
4	1	13.24	2.59

- In such cases, the system may fail to perform properly.
- Hence normalizing all the feature between 0 to 1 is required.
- Min-max normalization is popularly used, observe the values of alcohol and malic are now normalized between 0 to 1.
- This will be done as soon as the input is fed into the system, and before summation and activation function. It is also called pre-processing in neural network.



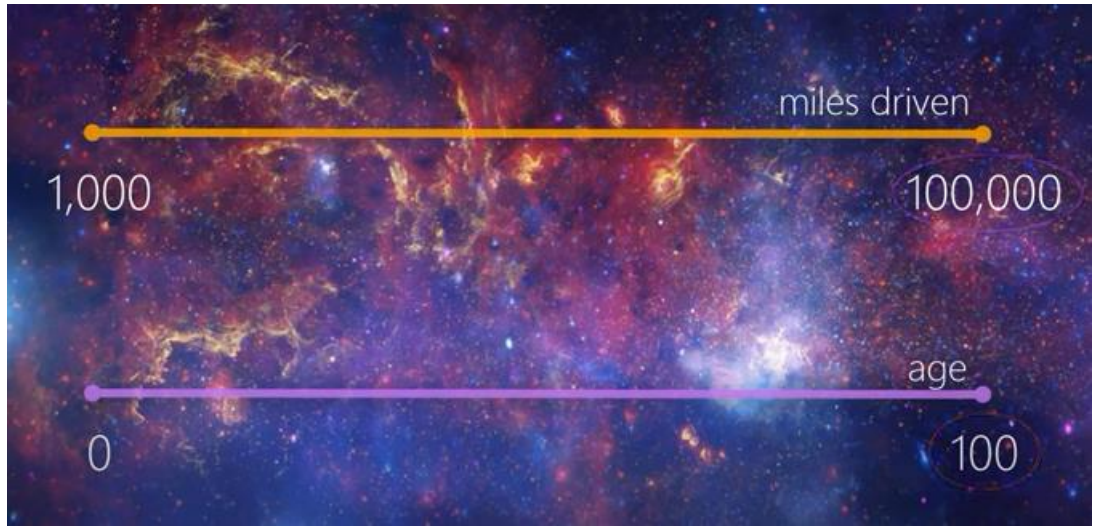
	Class	Alcohol	Malic
0	1	14.23	1.71
1	1	13.20	1.78
2	1	13.16	2.36
3	1	14.37	1.95
4	1	13.24	2.59

```
In [4]: from sklearn.preprocessing import MinMaxScaler

In [5]: scaling=MinMaxScaler()

In [8]: scaling.fit_transform(df[['Alcohol','Malic']])
[0.64473684, 0.18379447],
[0.35      , 0.61067194],
[0.7       , 0.49802372],
[0.47894737, 0.5       ],
[0.50789474, 0.53557312],
[0.72368421, 0.39920949],
```

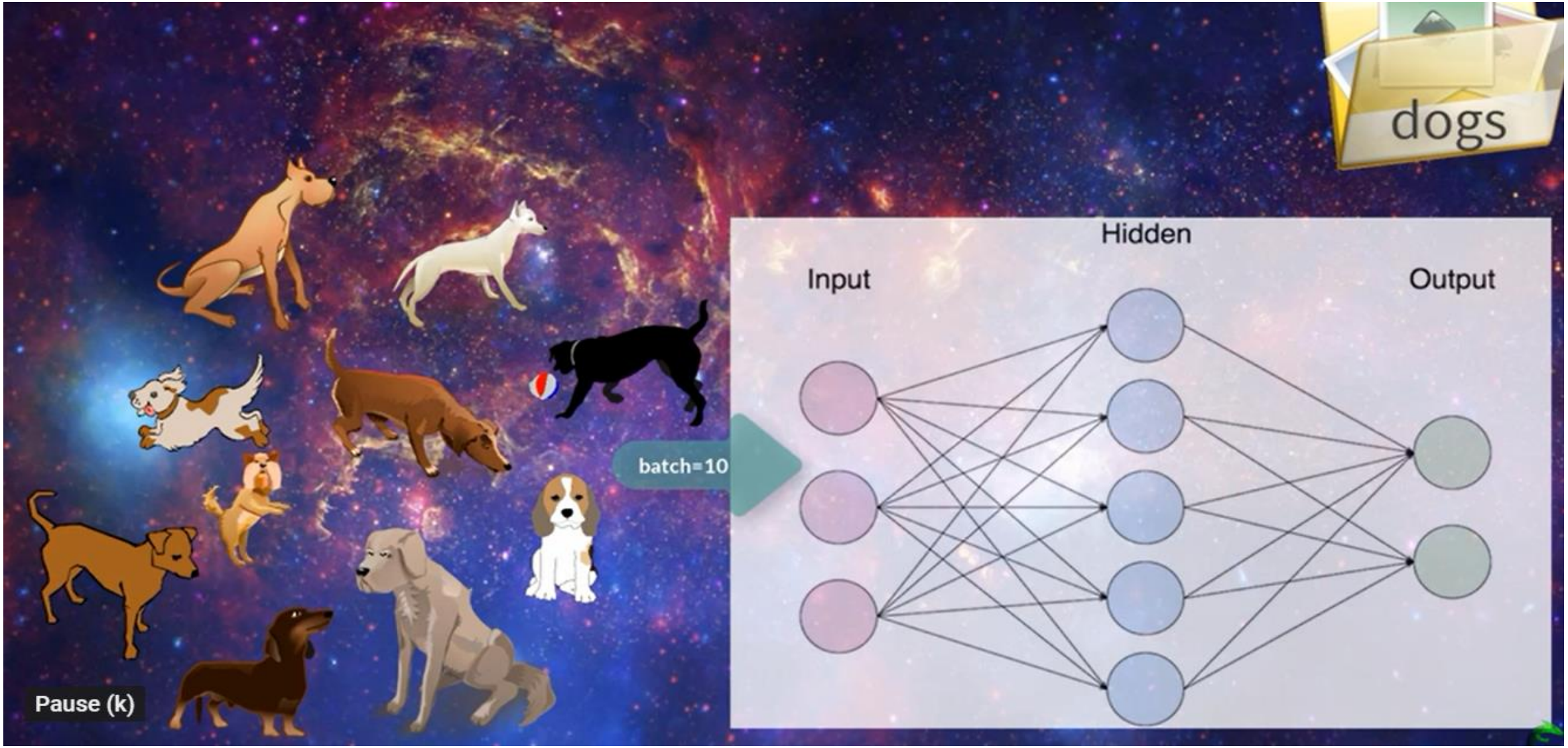
- In the example given below: age and number of mile of driving are two parameters.
- They are on different scaling.
- If they are used as it is without normalization, may lead to imbalance of neural network.
- To handle this we need to normalize the data.
- Right hand side you have the same data which is now normalized.



Batch size and epoch

- The weights are update after one iteration of every batch of data.
- For example, if you have 1000 samples and you set a batch size of 200, then the neural network's weights gets updated after every 200 samples.
- Batch is also called as 'Mini Batch'.
- An epoch completes after it has seen the full data set, so in the example above, in 1 epoch, the neural network gets updated 5 times.
- Batch size will be fixed based on the processing capacity.
- In the above example weights will not be updated till your training set receives 200 samples.

Batch size = 10

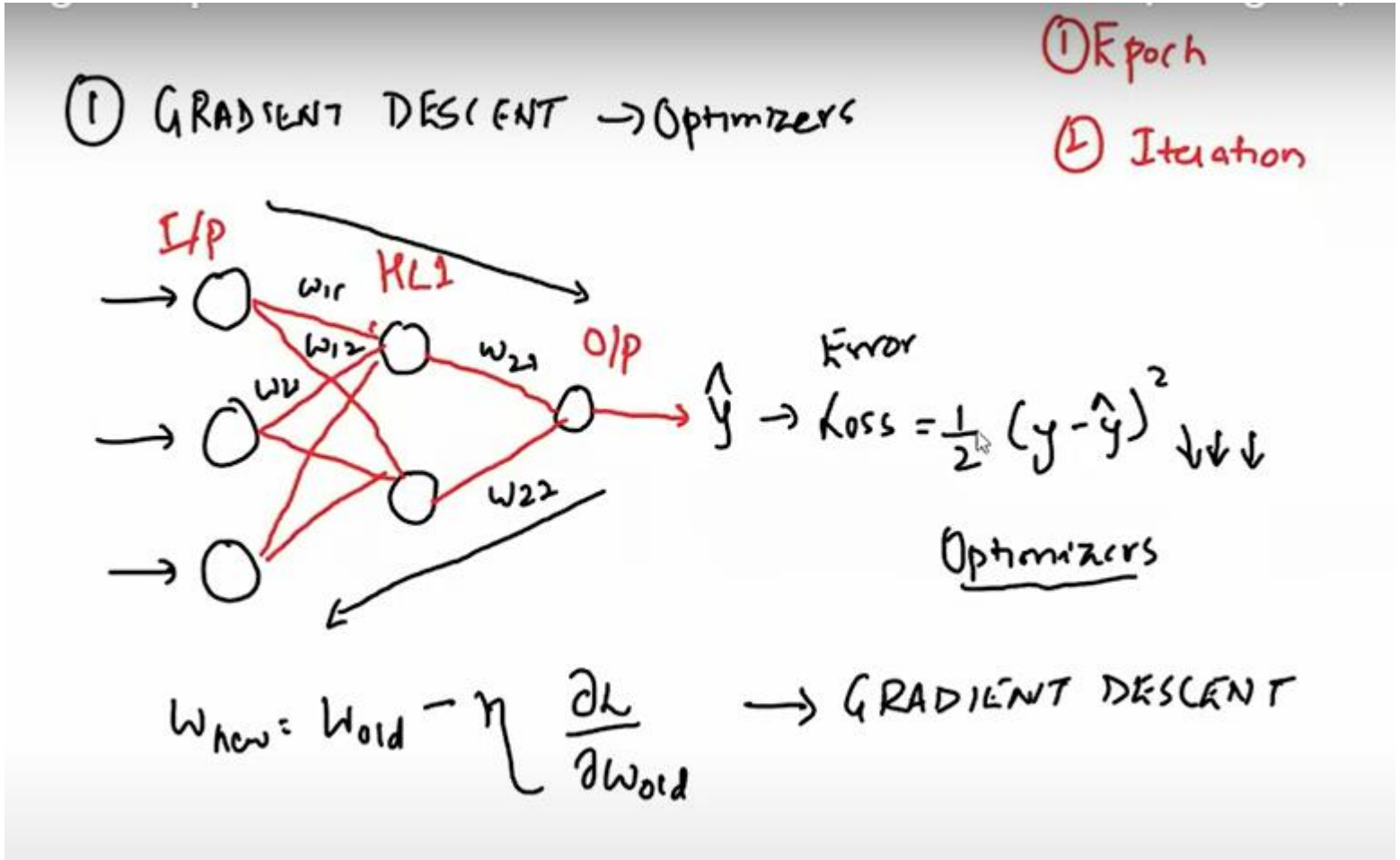


Faster Optimizers

- Training a very large deep neural network can be slow.
- Four ways to speed up training :
  - Applying good initialization strategy for the connection weights.
  - Using a good activation function
  - Using Batch Normalization and
  - Reusing parts of a pretrained network
- Another huge speed boost comes from using a faster optimizer than the gradient descent optimizer.
- Some of the popular fast optimizers are: Momentum Optimization, Nesterov Accelerated Gradient, AdaGrad, RMSProp and Adam Optimization

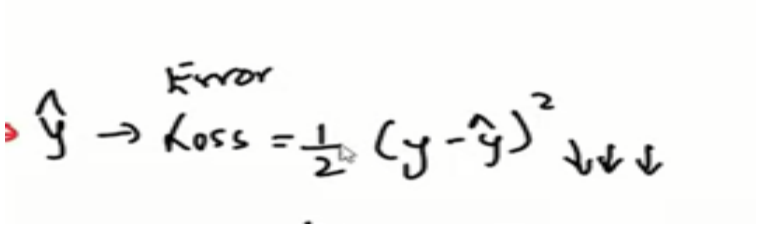
Reason to use fast optimizer?

- Let us consider the following case:

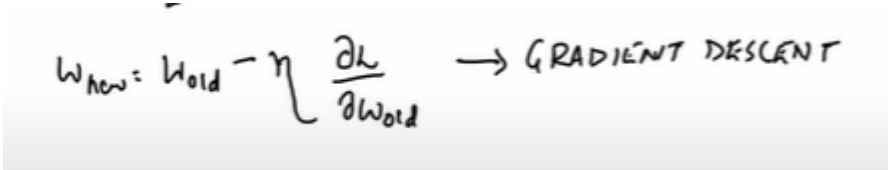


Gradient Descent updating…

- When we pass one sample, loss is given by:



- And gradient descent is, updating the weights with reference to loss.



Iteration and Epoch

- Forward propagate and later update the weights in the backward propagation for one sample is called ITERATION.

- If the iteration is completed for all the training samples available it is called ONE EPOCH.

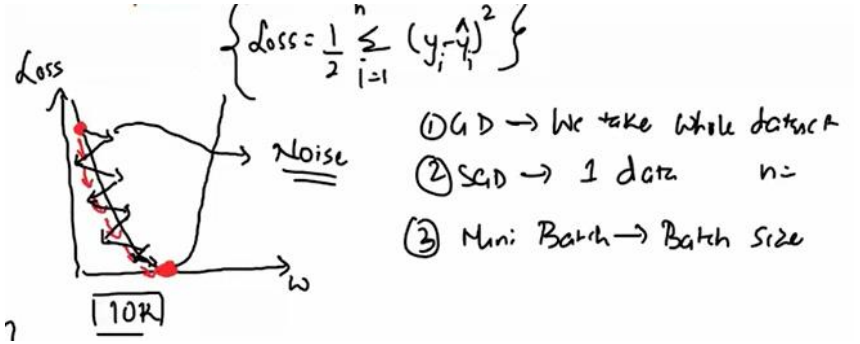
- Say for example if we have 10,000 training samples are available, then if we decide to update weights for every sample, we will be having 10,000 iterations and this is called one EPOCH.

General Gradient Descent and Stochastic Gradient Descent (SGD)

- In general Gradient Descent, loss will be collected for all the training samples and weights will be updated by taking average of all the loss.

- Say if we have 10,000 samples to be trained, then we will not be having 10,000 iterations. Will be collecting the loss of all the 10,000 samples. This completes one epoch and at the end of one epoch weights will be updated.
- The problem in the above method is, if the training data is too huge like 10 Lakhs or 50 Lakhs.. Then we need have huge RAM space to load all the samples and space is also required to hold the loss value for all the samples. As a solution researchers found another method called stochastic Gradient Descent.
- In Stochastic Gradient Descent (SGD), weights will be updated in every iteration and weights will be updated in every iteration. Though it requires less memory, it is time consuming to reach the global minima of error.

Solution to SGD is the ‘Mini Batch’



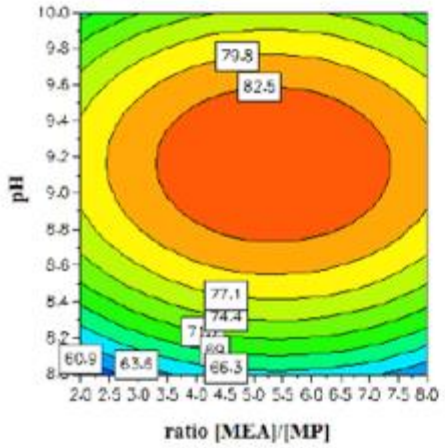
- Researchers have introduced a technique called “Min Batch” or “Mini Batch SGD”
- In mini batch, a batch of training data is considered for weight updates. This is the iteration value.

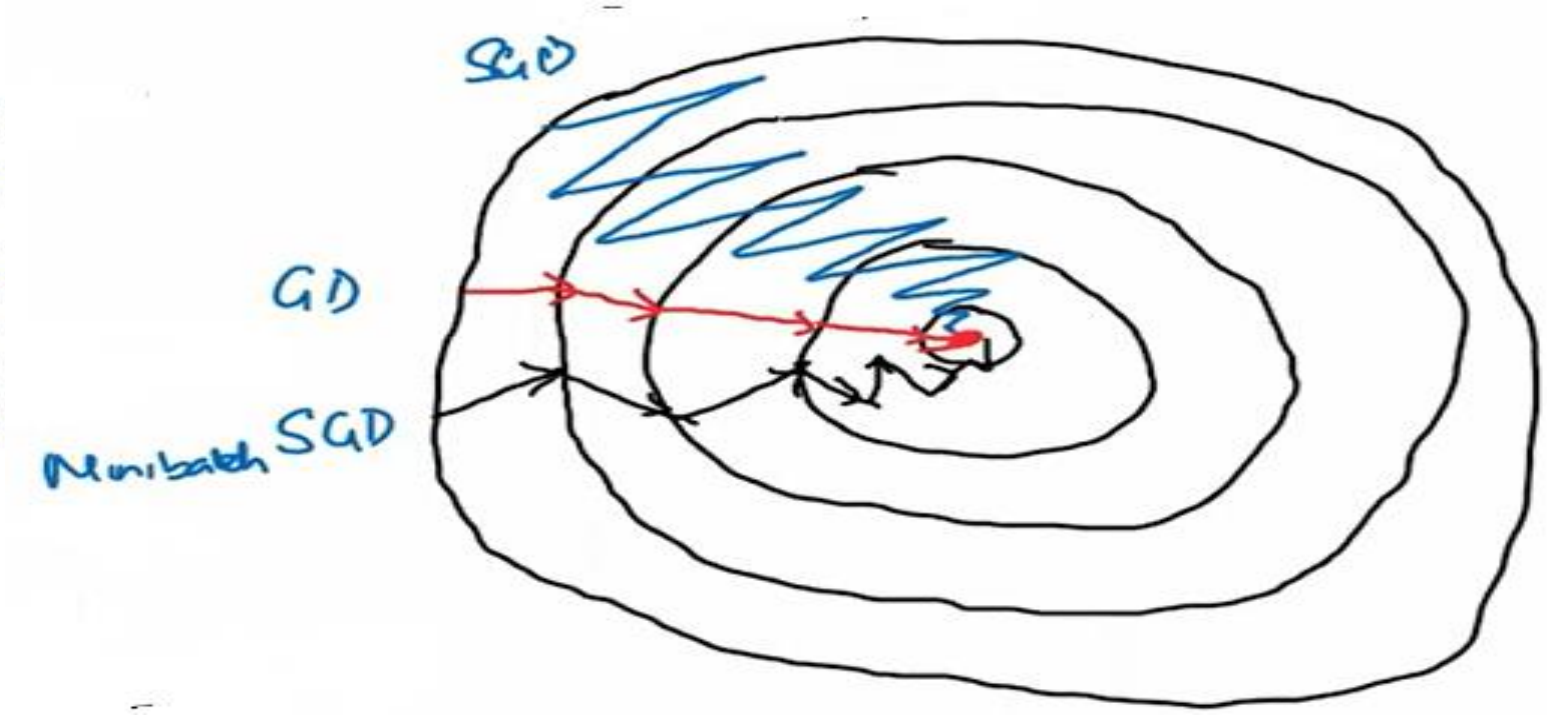
- For example, if we have 10,000 training samples, and if batch size is 1000, weights will be updated after every 1000 samples are trained.
- In this case to complete one epoch, we need to have 10 iterations.

- On the other hand in one epoch, there will be weight updation for 10 times, but this might have some noise as shown in the diagram.

Counter Plot

- If you draw a counter plot, which is the top view of the gradient descent, it will be smoother for GD (Red color), and for Minibatch SGD it will not be smooth (Black color) and blue colour is the SG.. Which will have more error. The left hand side picture is an illustration of counter plot using python.





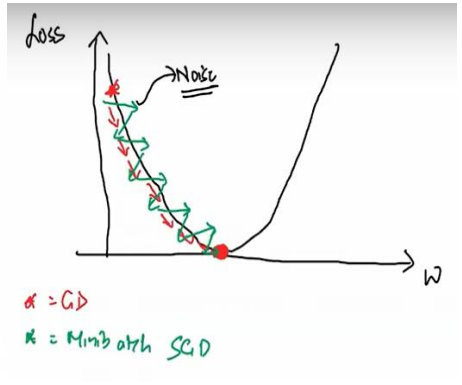
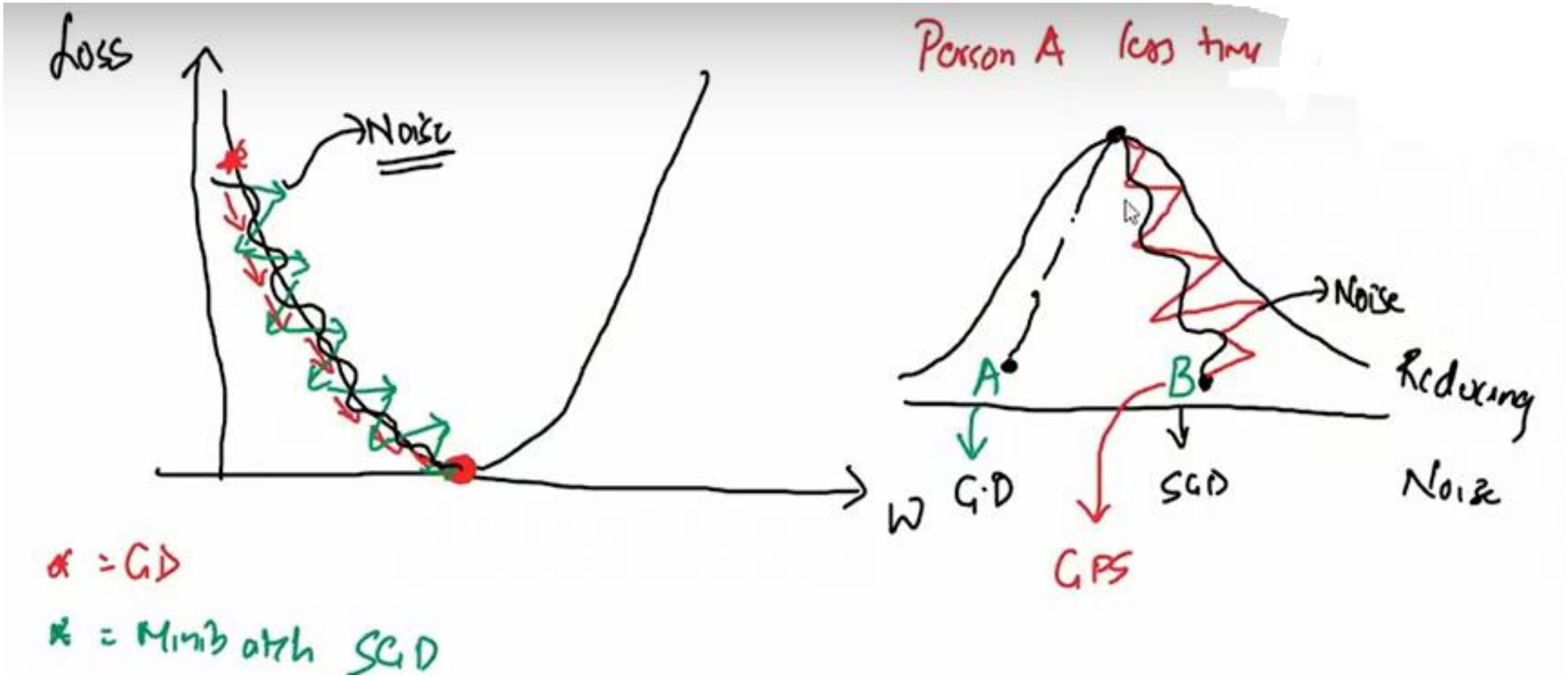


Illustration of noise, while climbing the hill, SGD or Mini SGD will have noise, which can be smoothened using some technique called Optimizers.



Fast Optimizer – Momentum Optimizer

- Consider the physics solution to smoothen the velocity:
- If beta is 0.95 and for the next step it will become 0.05, and hence the velocity of moment will be smoothened.

$t_1 \quad t_2 \quad t_3 \quad t_4 \cdots t_n$

$a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_n$

$V_{t_1} = a_1$

$V_{t_2} = \beta * V_{t_1} + (1-\beta) * a_2$

$V_{t_1} = a_1$

$V_{t_2} = \beta * V_{t_1} + (1-\beta) * a_2$

$= \beta * a_1 + (1-\beta) * a_2$

$= 0.1\, a_1 + (0.9) * a_2$

$V_{t_3} = \beta * V_{t_2} + (1-\beta) * a_3$

$= \beta(\beta * V_{t_1} + (1-\beta) * a_2) + (1-\beta) * a$

- Similarly weights will be updated in the momentum optimization:
- The below computation shows the weight calculation for a single weight. It can also be applied for bias. Wt-1 is the Wold.
- Vdw is the exponential weight change.

$$w_{new} = w_{old} - \eta \frac{\partial h}{\partial w_{old}}$$

$$w_t = w_{t-1} - \eta \frac{\partial h}{\partial w_{t-1}}$$

Exponential Weighted Average

$$w_t = w_{t-1} - \eta V_{dw}$$

Final concept of GD with Momentum for fast and smooth optimization

$$w_{new} = w_{old} - \eta \frac{\partial h}{\partial w_{old}}$$

$$w_t = w_{t-1} - \eta \frac{\partial h}{\partial w_{t-1}}$$

Exponential Weighted Average

$$w_t = w_{t-1} - \eta V_{dw}$$

$$V_{dw_t} = \beta V_{dw_{t-1}} + (1-\beta) * \frac{\partial h}{\partial w_{t-1}}$$

$$V_{db_t} = \beta V_{db_{t-1}} + (1-\beta) * \frac{\partial h}{\partial b_{t-1}}$$

Summary of GD, SGD and Mini Batch SGD

- GD – Weight updation will be done after all samples are passed through the model
- SGD – weight updation takes place for every sample
- Mini Batch – Weight updation takes place for every batch.

- Batch size should not be too small or too big. Depending on the available sample size, program should decide about the batch size.

AdaGrad Faster Optimizer

- AdaGrad – Adaptive Gradient.
- In Adagrad Optimizer the core idea is that **each weight has a different learning rate (η)**.
- This modification has great importance.
- In the real-world dataset, some features are sparse (for example, in Bag of Words most of the features are zero so it's sparse) and some are dense (most of the features will be noon-zero).
- So keeping the same value of learning rate for all the weights is not good for optimization. The weight updating formula for adagrad looks like:

- Weight updation in Ada Grad is given by:

$$w_t = w_{t-1} - \eta'_t \frac{\partial L}{\partial w(t-1)} \qquad \eta'_t = \frac{\eta}{sqrt(\alpha_t + \epsilon)}$$

- Where **alpha(t)** denotes different learning rates for each weight at each iteration.
- Here, **η** is a constant number, **epsilon** is a small positive value number to avoid divide by zero error if in case **alpha(t)** becomes 0 because if alpha(t) become zero then the learning rate will become zero which in turn after multiplying by derivative will make w(old) = w(new), and this will lead to small convergence.

- **Advantages of Adagrad:**
  - No manual tuning of the learning rate required.
  - Faster convergence
  - More reliable

Avoiding Overfitting Through Regularization

- Deep Neural Network typically have tens of thousands of parameters.
- With so many parameters, the network is prone to overfitting the training set.

- This will be done using “Regularization” techniques.
- Some of the popular regularization techniques are:
  - Early Stopping
  - Dropout
  - Max-Norm Regularization and
  - Data Augmentation.

Early Stopping

- To avoid Overfitting the training set, good solution is early stopping.
- Interrupt training when its performance on the validation set starts dropping.

- Evaluate the model on a validation set at regular intervals.
- If the performance is improved compared to the previous interval, go back to the pervious values and stop training.

Dropout

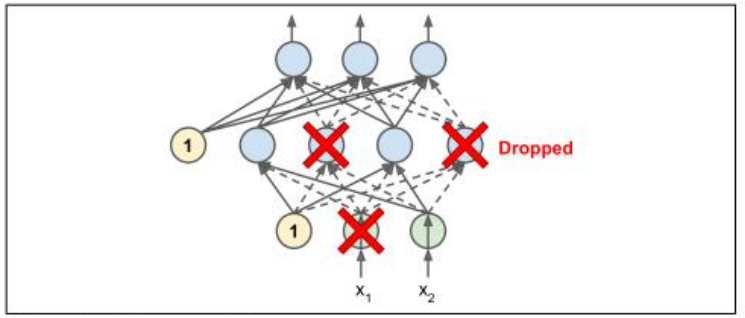


Figure 11-9. Dropout regularization

- Another popular regularization technique for deep neural network is arguably dropout.
- At every training step, every neuron has a probability P of being temporarily “Dropped Out”, meaning it will be entirely ignored during this training step.
- But it may be active during the next step.
- The hyperparameter ‘P’ is called the dropout rate and it is typically set to 50%.
- After training neurons don’t get dropped anymore.
- It is found that many a times this technique has worked well.

Max-Norm Regularization

- Another regularization technique that is quite popular for neural networks s called max-norm regularization..
- It **constrains** the weights w of the incoming connections such that** $\| w \|_2 \leq r$**, where r is the max-norm hyperparameter and** $\| \cdot \|_2$ is the ℓ2 norm**.
- It is typically implemented by computing **$\|w\|_2$ after each training step** and **clipping w** if needed.
- **Reducing r** increases the amount of regularization and helps **reduce overfitting**.

Data Augmentation

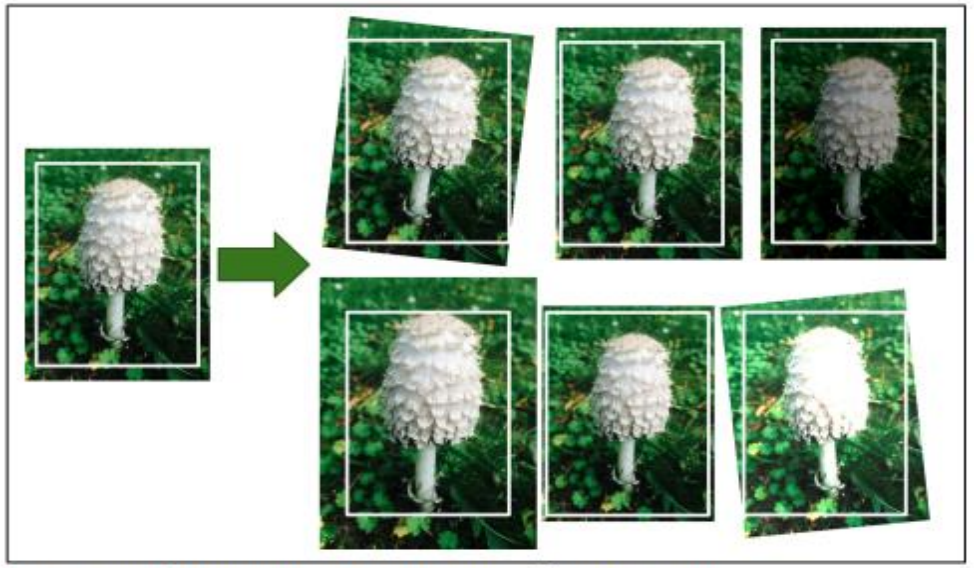


Figure 11-10. Generating new training instances from existing ones

- One last regularization technique is data augmentation.
- It consists of generating new training instances from existing ones.
- Artificially boosting the size of the training set.
- This will reduce overfitting making this a regularization technique.
- The trick is to generate realistic training instances, ideally a human should not be able to tell which instances were generated and which ones were not.

End of Unit - 2

Distributing Tensor flow across devices and servers

Unit - 3

Syllabus : Unit 3

Multiple devices on a single machine
multiple servers
parallelizing NN on a Tensor Flow cluster
Convolution Neural Network:
Architecture of the visual cortex
Convolutional layer
Pooling layer
CNN architecture

Multiple Devices on a single machine

- Tensorflow provides facility to execute multiple tasks across multiple devices present in the same machine.
- Multiple devices can be multiple GPUs' and CPUs'
- You can do this by adding multiple GPU cards to a single machine.

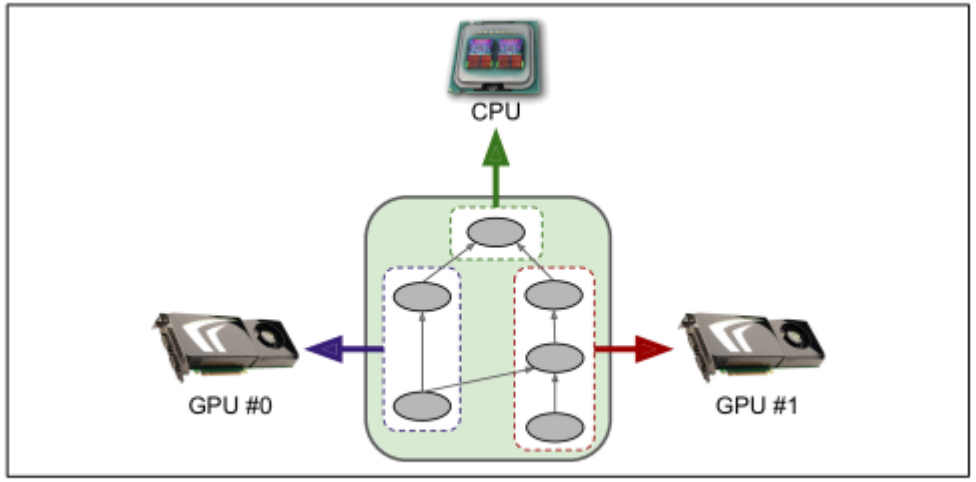


Figure 12-1. Executing a TensorFlow graph across multiple devices in parallel

- Major performance boost is obtained by adding single machine.
- Nvidia’s Compute Unified Device Architecture Library (CUDA) allows developers to use CUDA enabled GPUs for all sorts of computations.
- Nvidia’s CUDA Deep Neural Network Library (cuDNN) is a GPU accelerated library of accelerated library of primitive for DNN.
- It provides optimized implementations of common DNN commutations such as activation layers, normalization, forward and backward convolutions and pooling .
- Block diagram of multiple device on single machine for ANN execution is shown in the next slide.

Block diagram to show the use of multiple devices on a single machine.

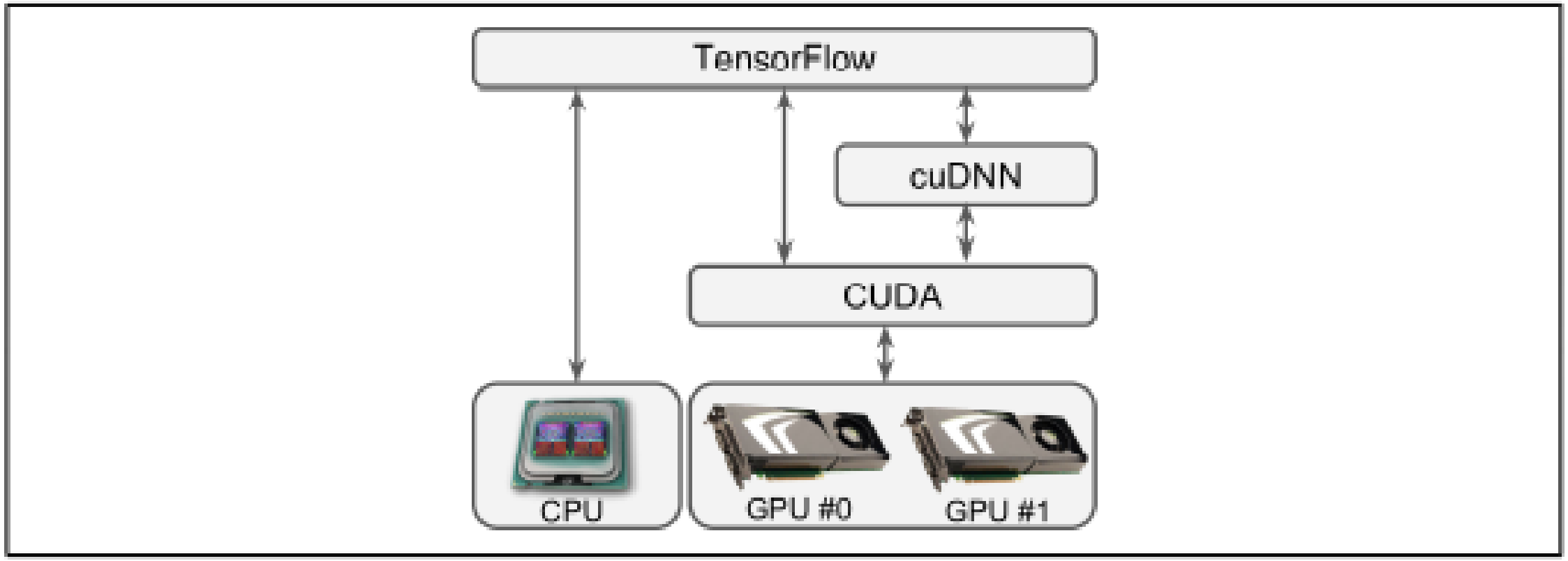


Figure 12-2. TensorFlow uses CUDA and cuDNN to control GPUs and boost DNNs

Multiple devices across multiple servers

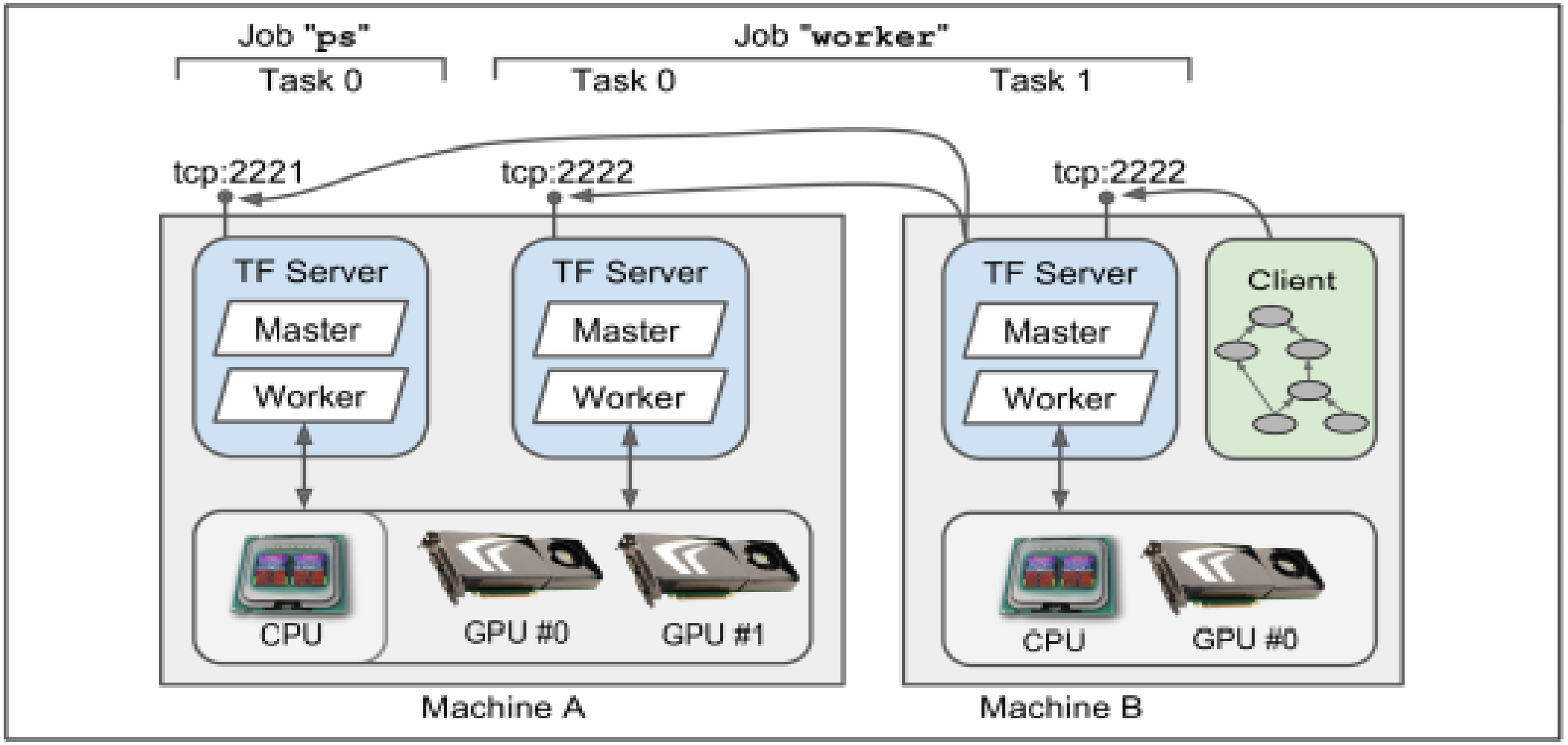


Figure 12-6. TensorFlow cluster

- To run a graph across multiple servers, you first need to define a cluster.
- A, cluster is composed of one or more Tensorflow servers, called tasks, which are typically spread across several machines as shown in the picture presented in the previous slide.
- Each task belongs to a job.
- A job is just a named group of tasks that typically have a common role such as “ps” or “worker” in the diagram shown.

Parallelizing Neural Networks

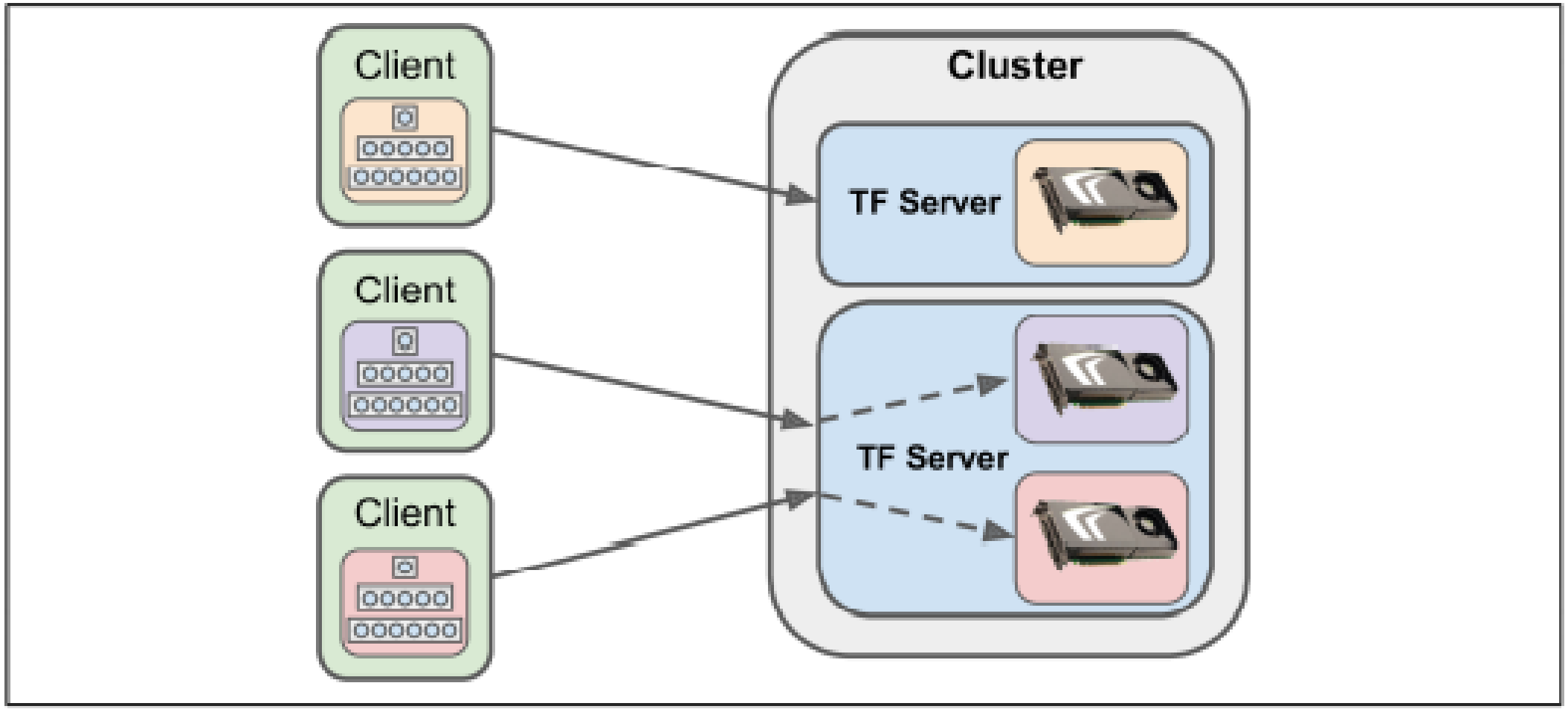


Figure 12-11. Training one neural network per device

- Initially use the code for a single device and specify the master server’s address when creating the session.
- Now the program will be running on the server’s default device.
- Later, by running several client sessions in parallel in different threads or different process, connecting them to different servers and configuring them to use different devices you can quite easily train or run many neural networks in parallel across all devices and all machines in your cluster.

Convolution Neural Networks

- Convolutional neural networks (CNN) emerged from the study of brain's visual cortex and they have been used in image recognition since the 1980's.
- In the last few years due to increase in computational power, the amount of available training data, CNN have managed to achieve superhuman performance on some complex visual trasks.

- CNN's are now not just restricted to visual perception, they are also successful at other tasks such as voice recognition or natural language processing (NLP)

Why CNN for image recognition?

- A regular Deep Neural Network with fully connected layers for image recognition task is fine if it is a small image.
- It breaks downs for larger images because of the huge number of parameters it requires, for example a 100x100 image has 10,000 pixels.
- If we have 1000 neurons in the first layer, then will be having 10 million (1 crore) connections.
- This is just he first layer…and think of subsequent layers.
- CNN solves this problems using partially connected layers.

The Architecture of the Visual Cortex

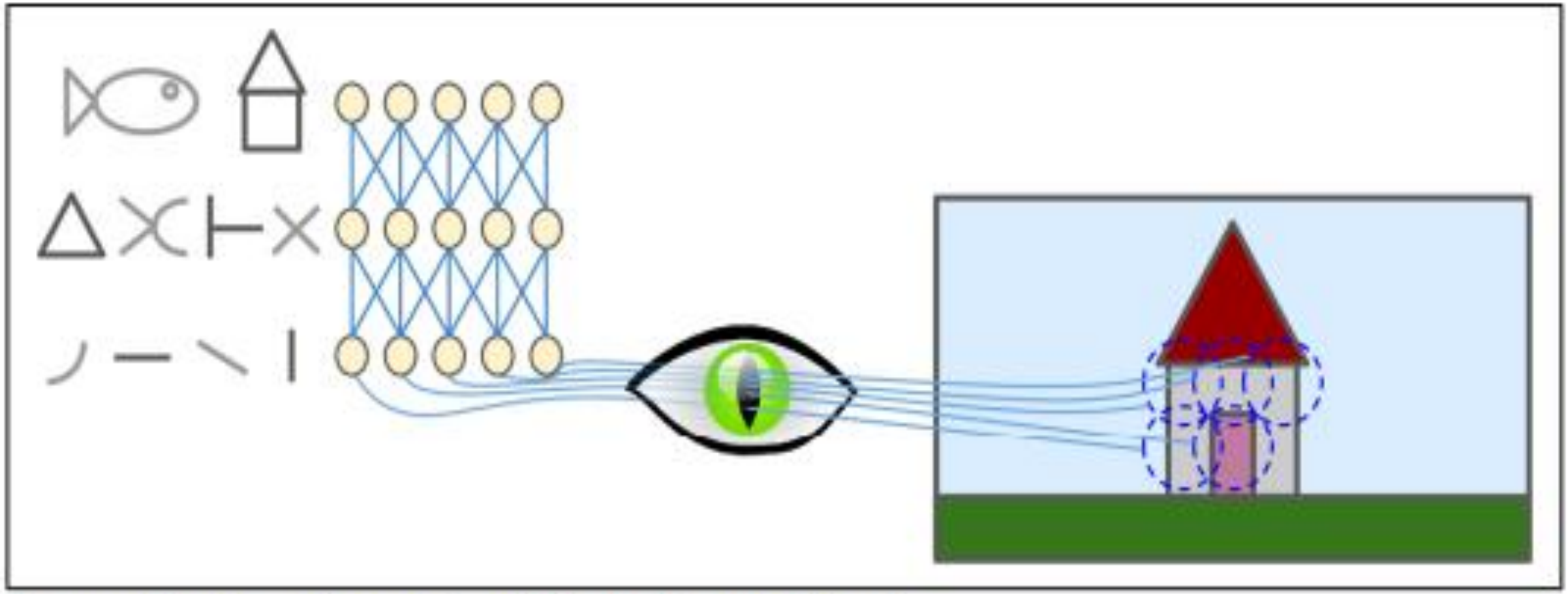
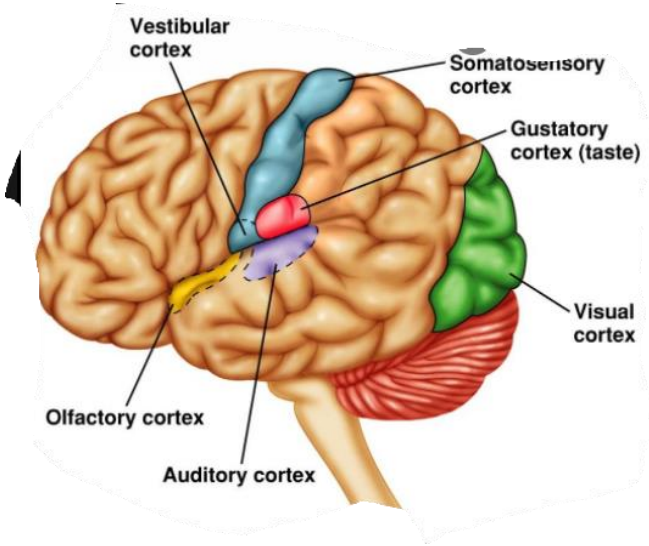
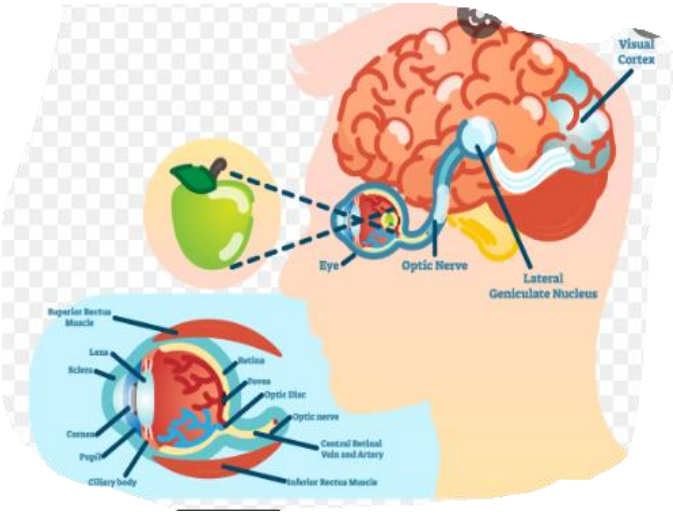


Figure 13-1. Local receptive fields in the visual cortex

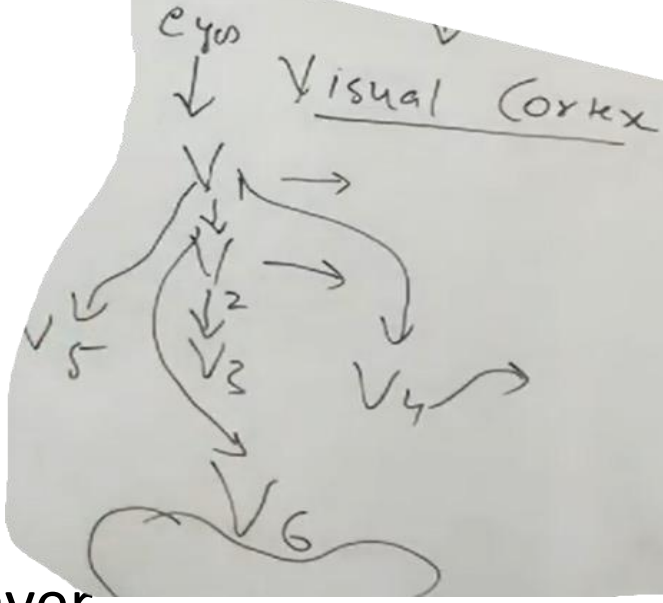
Visual Cortex Architecture





- Whenever your input is in the form of images and even frames of the video CNN is highly preferred.
- It is very efficient in recognizing the face, object recognition and so on.
- Cerebral Cortex is in the back part of our brain.
- Inside the cerebral cortex visual cortex is present, which is responsible for creating the impression of vision.
- Eye ball are only responsible for passing the picture data of the image.

Visual Cortex



- Visual Cortex has multiple layers like V1, V2,…V6
- They play very important role in recognition of object.
- For example, If we are trying to recognize the animal CAT, V1 layer identifies the borders.
- It next passes data to the next layer, say layer V2 and so on.. It tries to identify any other object associated with the primary object… and so on…
- Similarly each layer does a specific task. Some times data is passed from V1 to V4 and or any other layer directly instead of passing sequentially.
- After data passed through several layers, final picture is created in our brain.
- Similar to visual cortex, different filters in the CNN does different tasks.

Convolution Layer

- The most important building block of CNN is the convolution layer.

- All multilayer neural networks we looked at had layers composed of a long line of neurons, and we had to flatten input images to 1D before feeding them to the neural network.
- In CNN each layer is represented in 2D, which makes it easier to match neurons with their corresponding inputs.

- Neurons in the first convolution layer are not connected to every single pixel in the input image, but only to pixels in their receptive fields.

- In turn, each neuron in the second convolution layer, connected only to neurons located within a small rectangle in the first hidden layer. (shown in the picture in the next slide)

CNN Layers

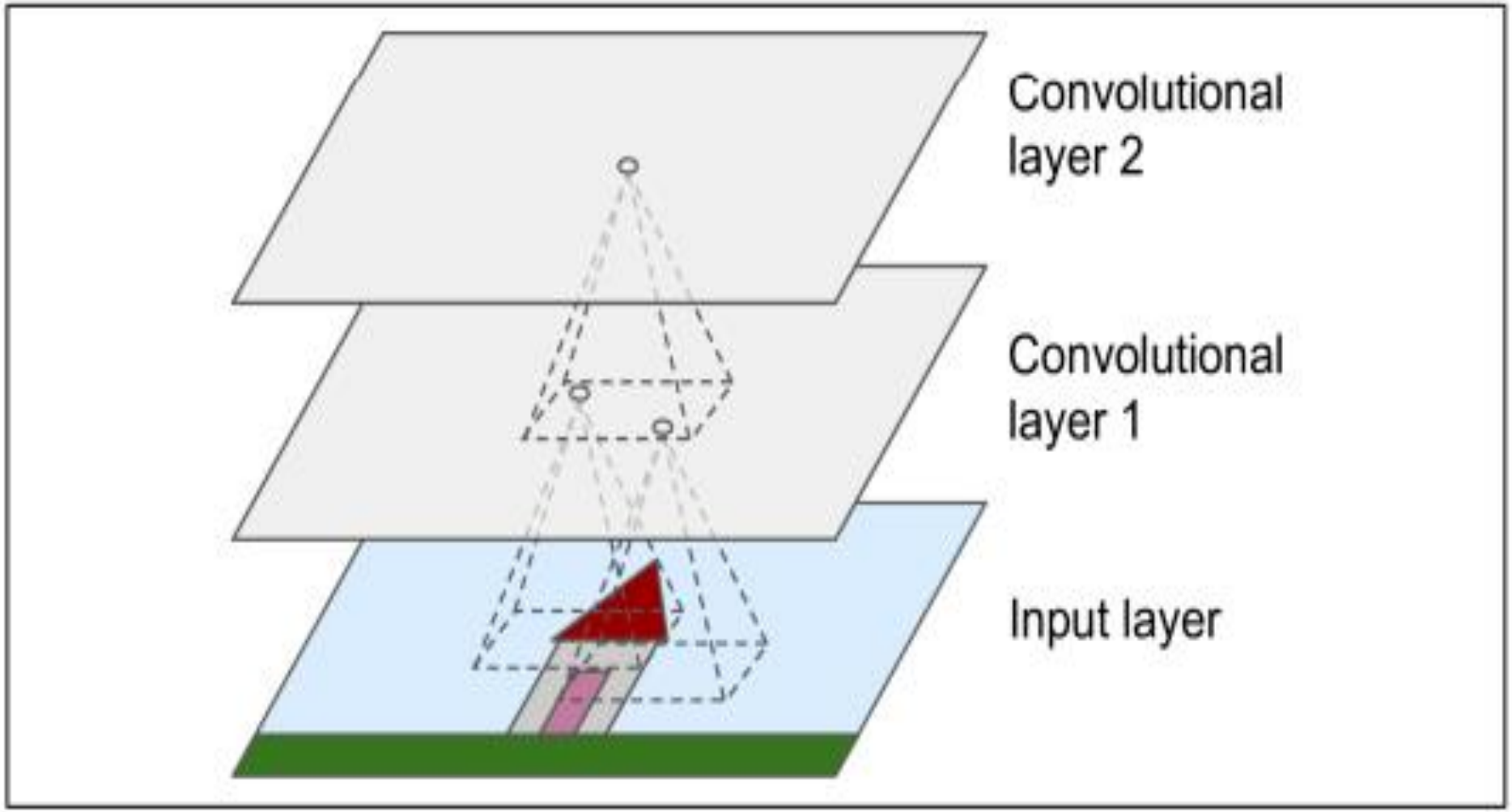


Figure 13-2. CNN layers with rectangular local receptive fields

Connection between layers and zero padding

- Filter is passed through the input or any other subsequent layer will generate the next level convolution layer.
- The figure shown below is a 3x3 filter.

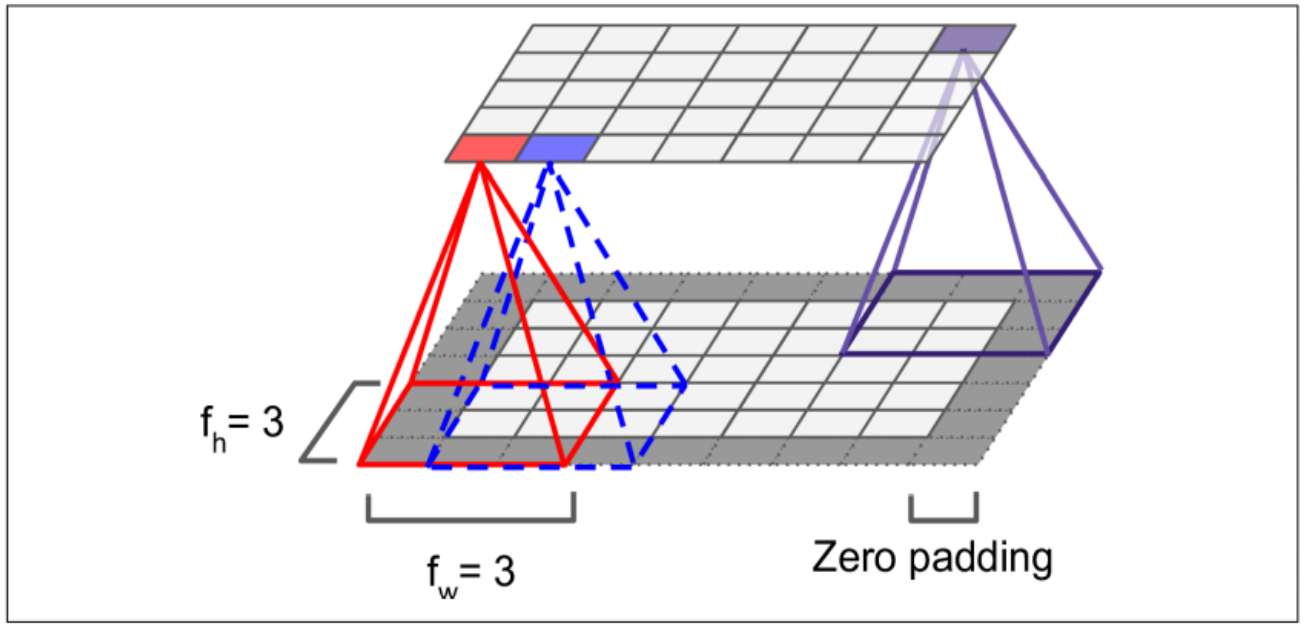


Figure 13-3. Connections between layers and zero padding

Connection between layers and zero padding

- A neuron located in row i, column j of a given layer is connected to the outputs of the neurons in the previous layer located in

  rows i to (i + fh – 1), and columns j to (j + fw – 1)

- Where fh and fw are the height and width of the receptive field (see Figure 13-3 in the previous slide) or we can say filter width and filter height.

- In order for a layer to have the same height and width as the previous layer, it is common to add zeros around the inputs, as shown in the diagram. This is called zero padding. (Explained in detailed in later part)

Reducing dimensionality using stride

- Stride is **the number of pixels shifts over the input matrix**.
- When the stride is 1 then we move the filters to 1 pixel at a time.
- When the stride is 2 then we move the filters to 2 pixels at a time and so on.
- As the stride value increases the dimension of the next convolution layer decreases.
- The figure shows convolution would work with a stride of 2.

Stride can have different size on x and y direction.

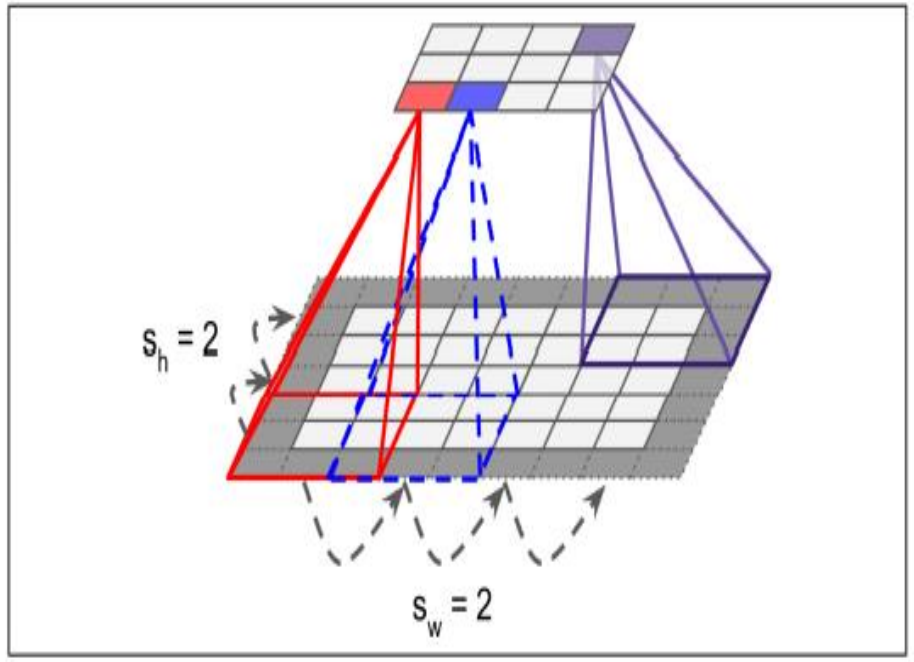
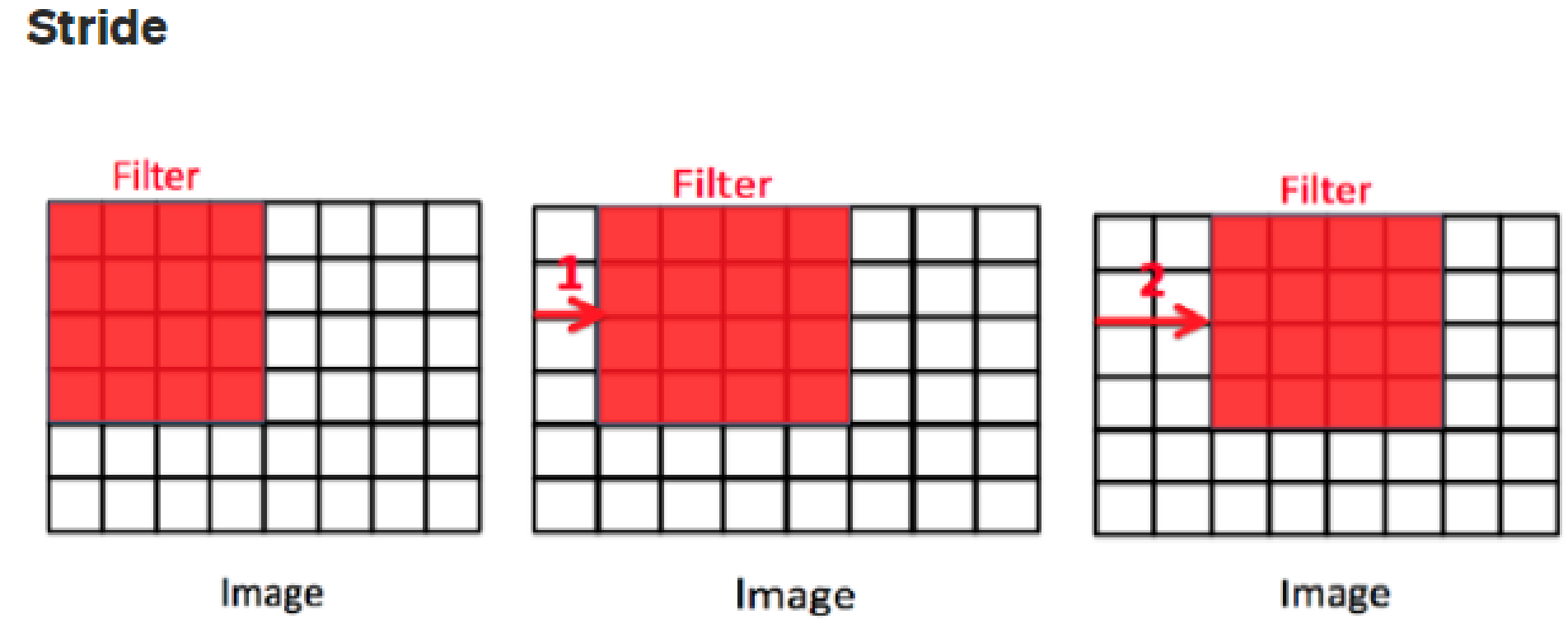


Figure 13-4. Reducing dimensionality using a stride

Stride movement



left image: stride =0, middle image: stride = 1, right image: stride =2

Filters

- In CNN neuron's weights can be represented as a small image called filters.

- Different filters are used for different purpose. Say for example Vertical filter for identifying the vertical lines and horizontal filter for highlighting the horizontal lines and so on.

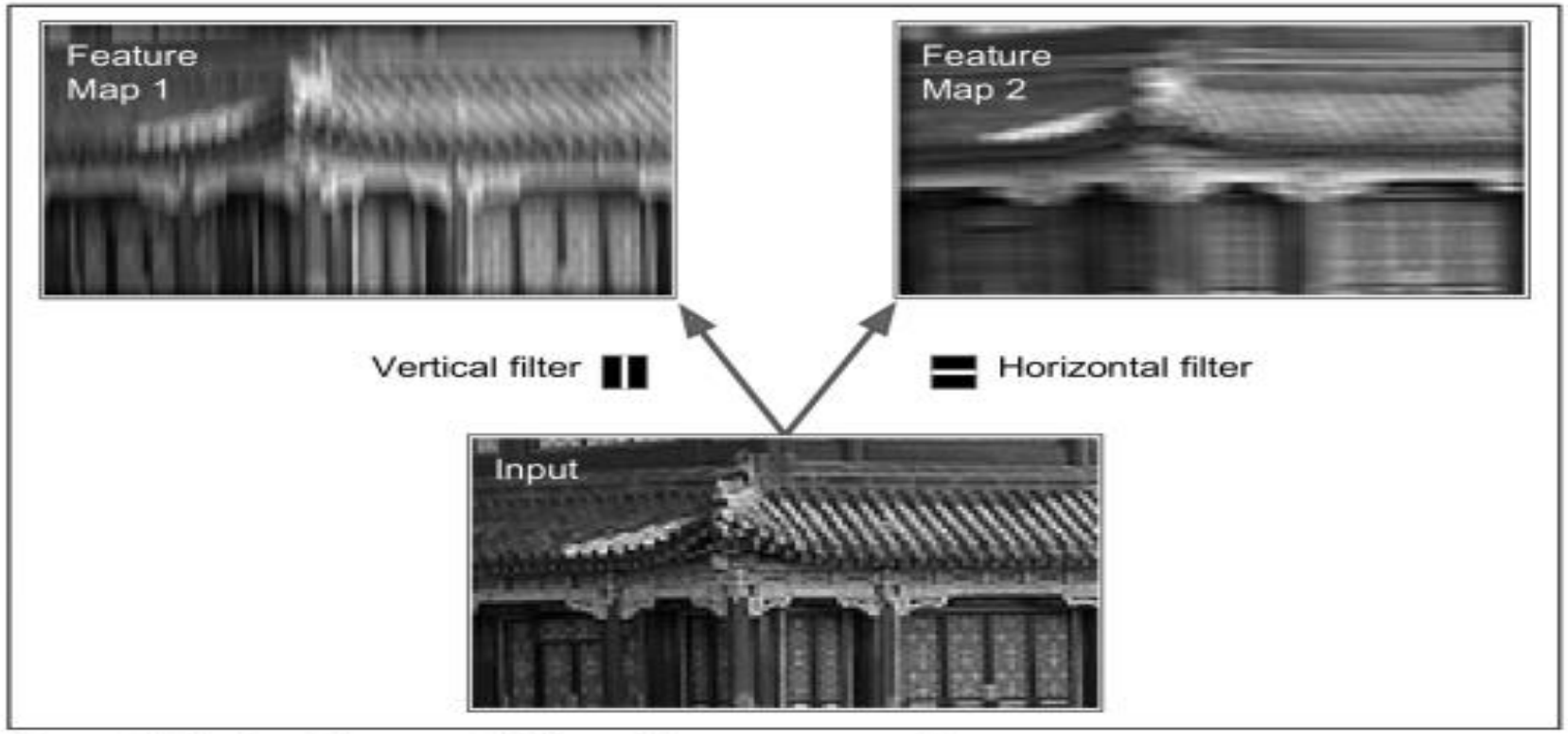


Figure 13-5. Applying two different filters to get two feature maps

Filters continued…

- Filters are popularly called as convolution kernels.
- This slide illustrates the use of vertical filters.

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

1	0	-1
1	0	-1
1	0	-1

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

*

Note: Higher pixel values represent the brighter portion of the image and the lower pixel values represent the darker portions. This is how we can detect a vertical edge in an image.

Other popular filters

1	0	-1
1	0	-1
1	0	-1

Vertical

1	1	1
0	0	0
-1	-1	-1

Horizontal

Some of the commonly used filters are:

1	0	-1
2	0	-2
1	0	-1

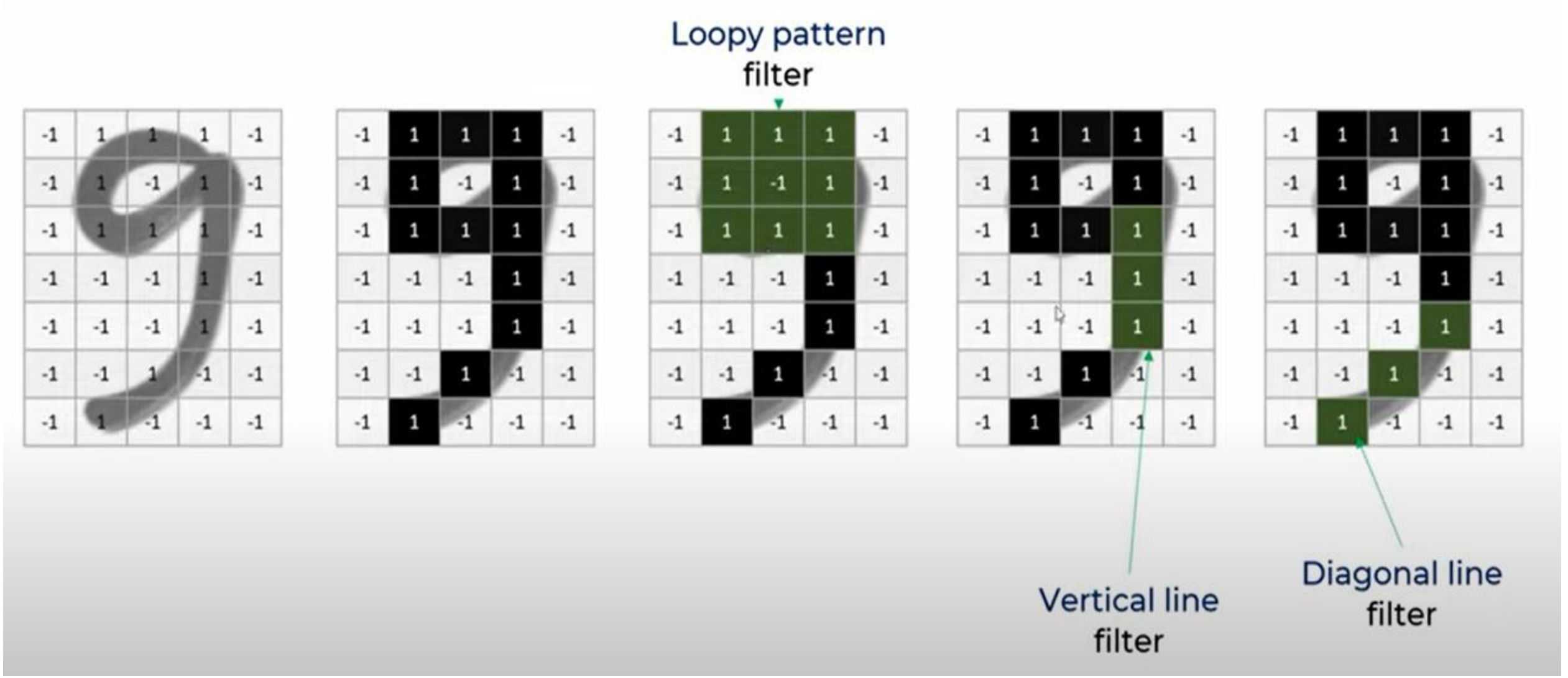
Sobel filter

3	0	-3
10	0	-10
3	0	-3

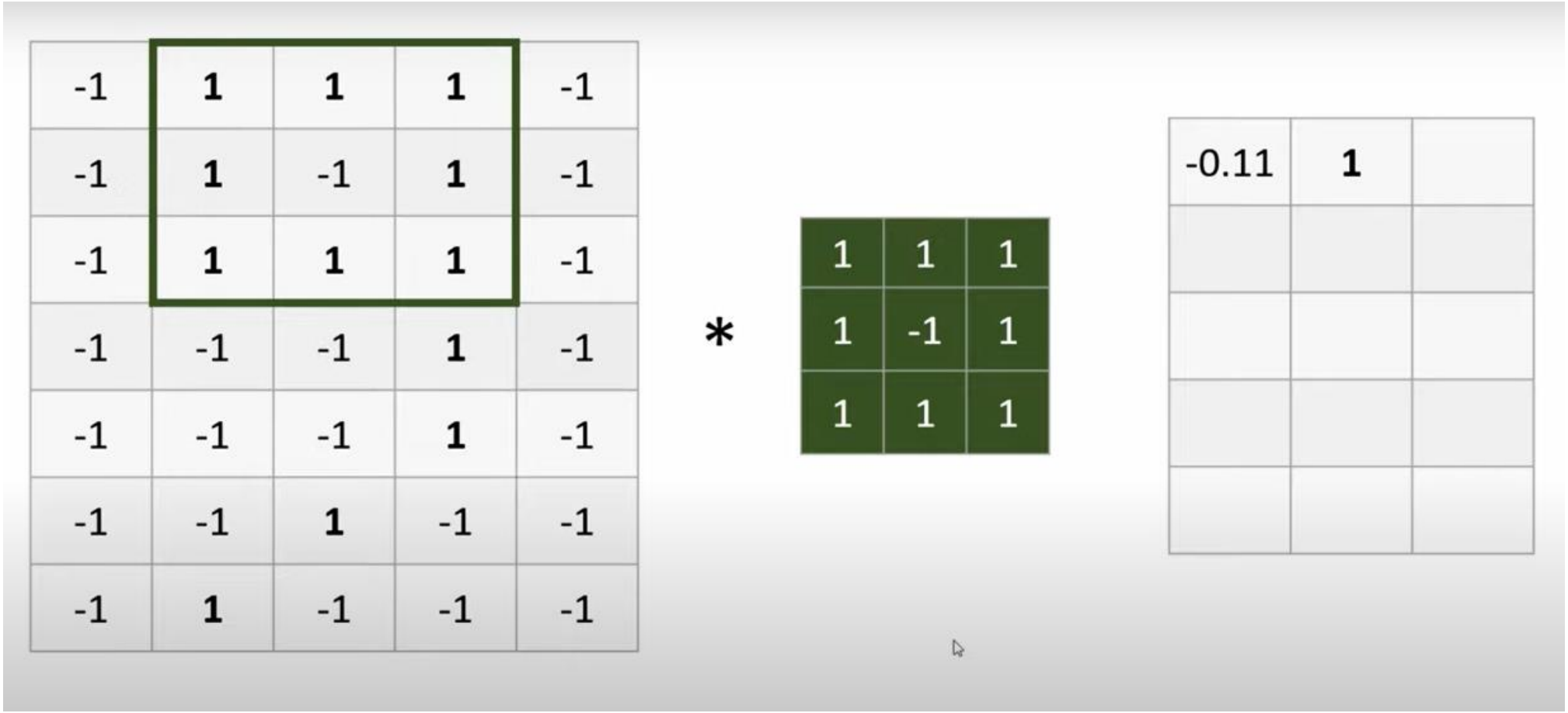
Scharr filter

- It is required to create our own filter to identify different objects in a image.
- Further the values in the filter will be updated during back propagation of CNN.

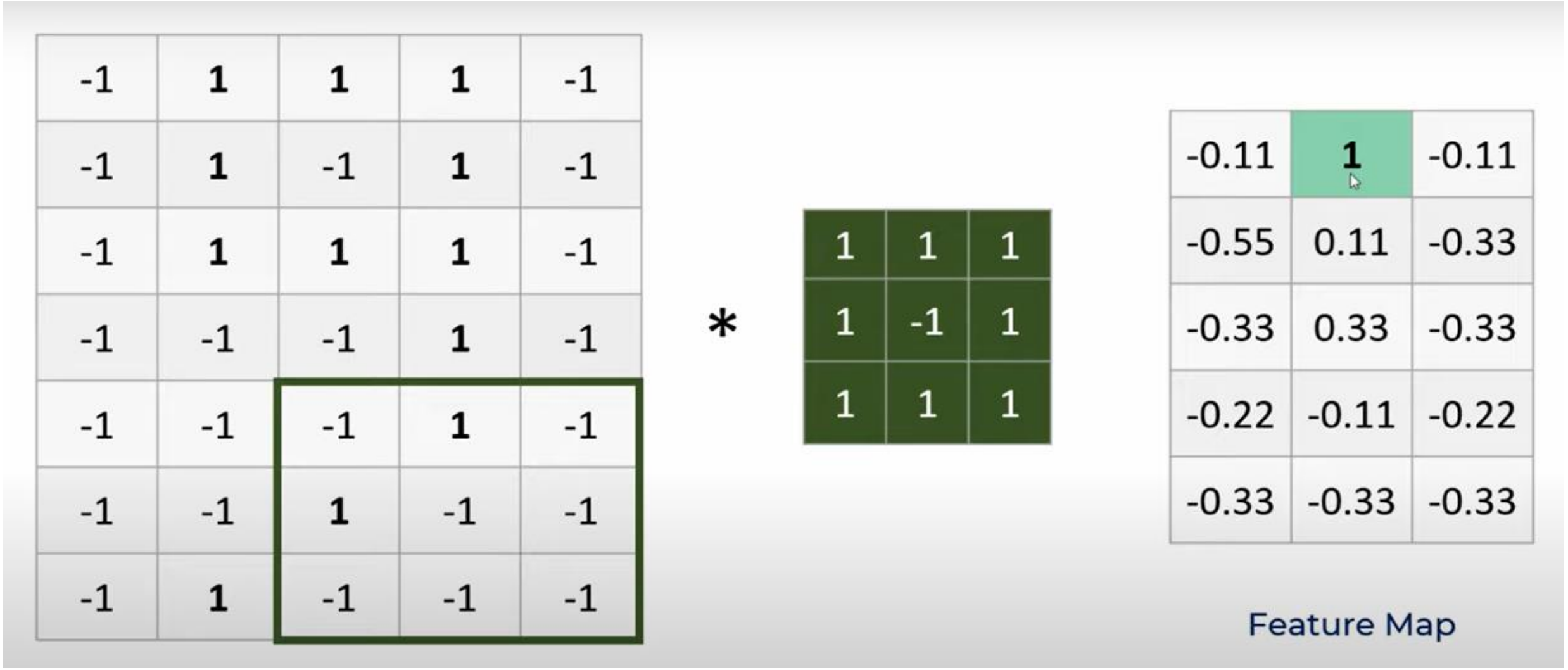
Applications of different filters



Loopy filter applied

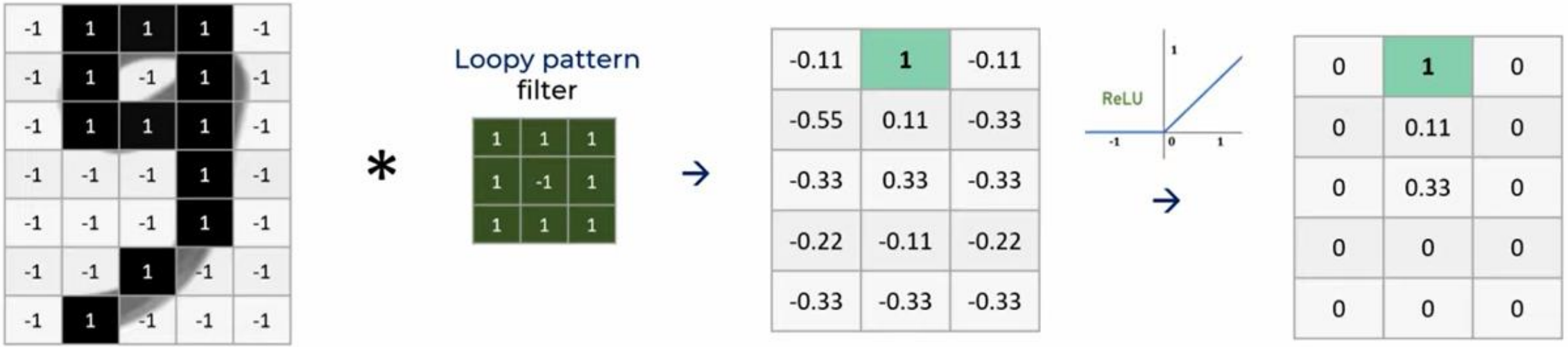


Loopy filter



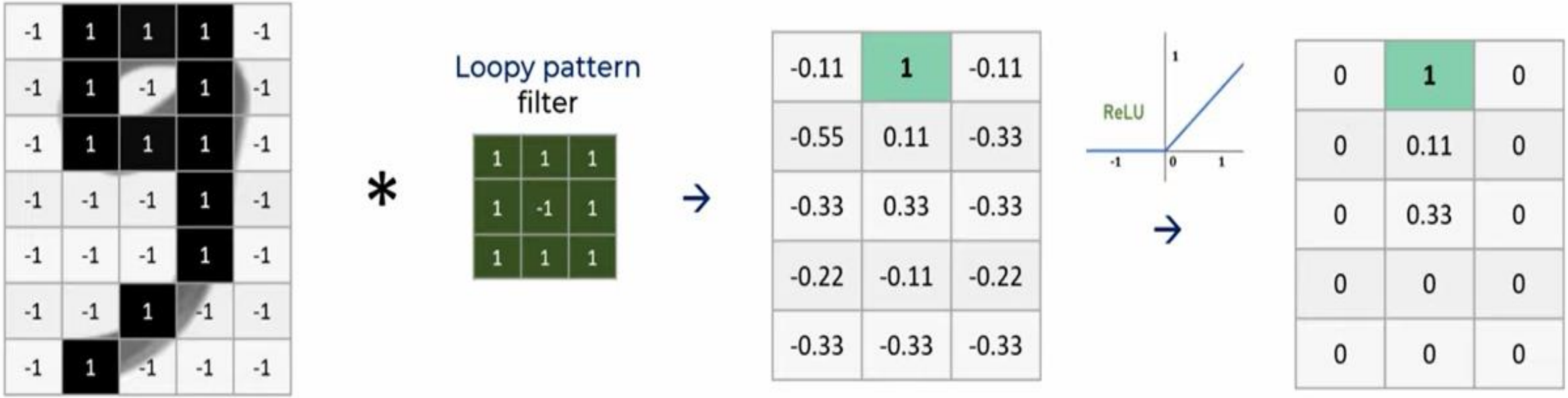
Activation function on the feature mapped image

- ReLU activation function is popularly used after feature mapping.
- All –ve values are replaced by zero and the remaining values will be retained as it is.



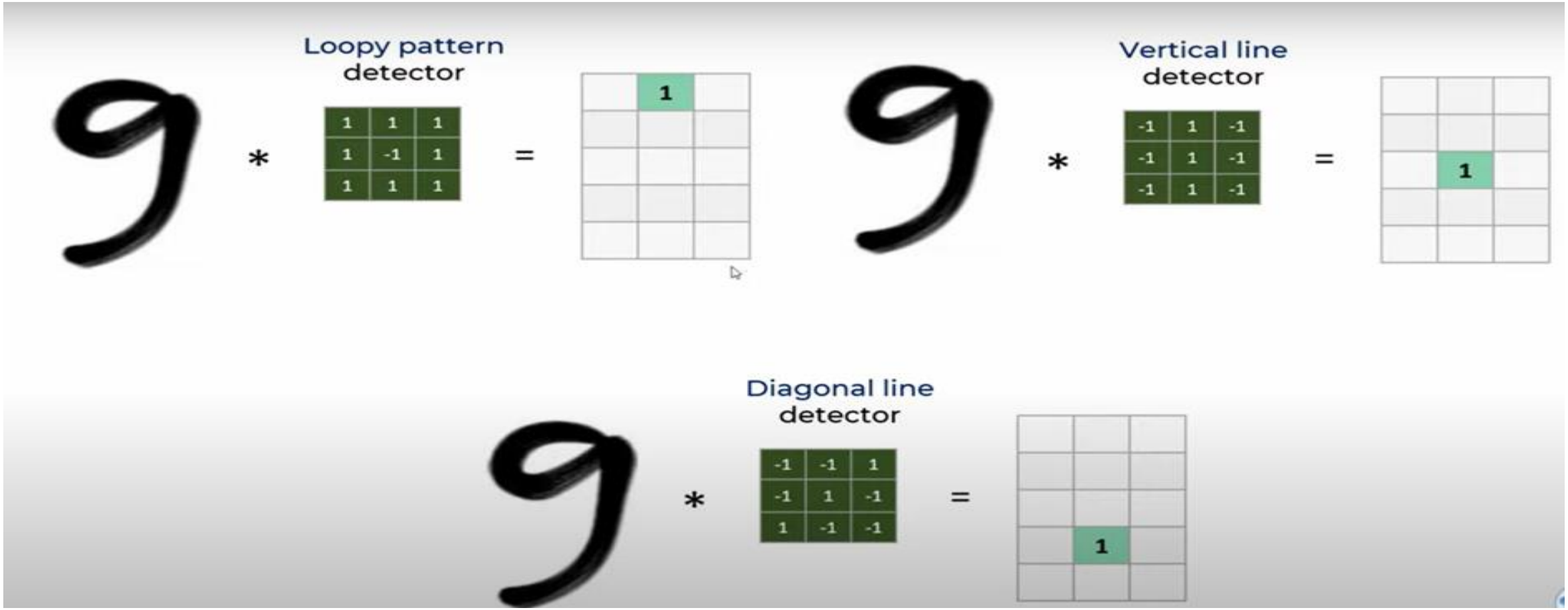
Application of filter to identify different patterns

- Example shown below is the application of filter to identify the presence of loop in different numbers:

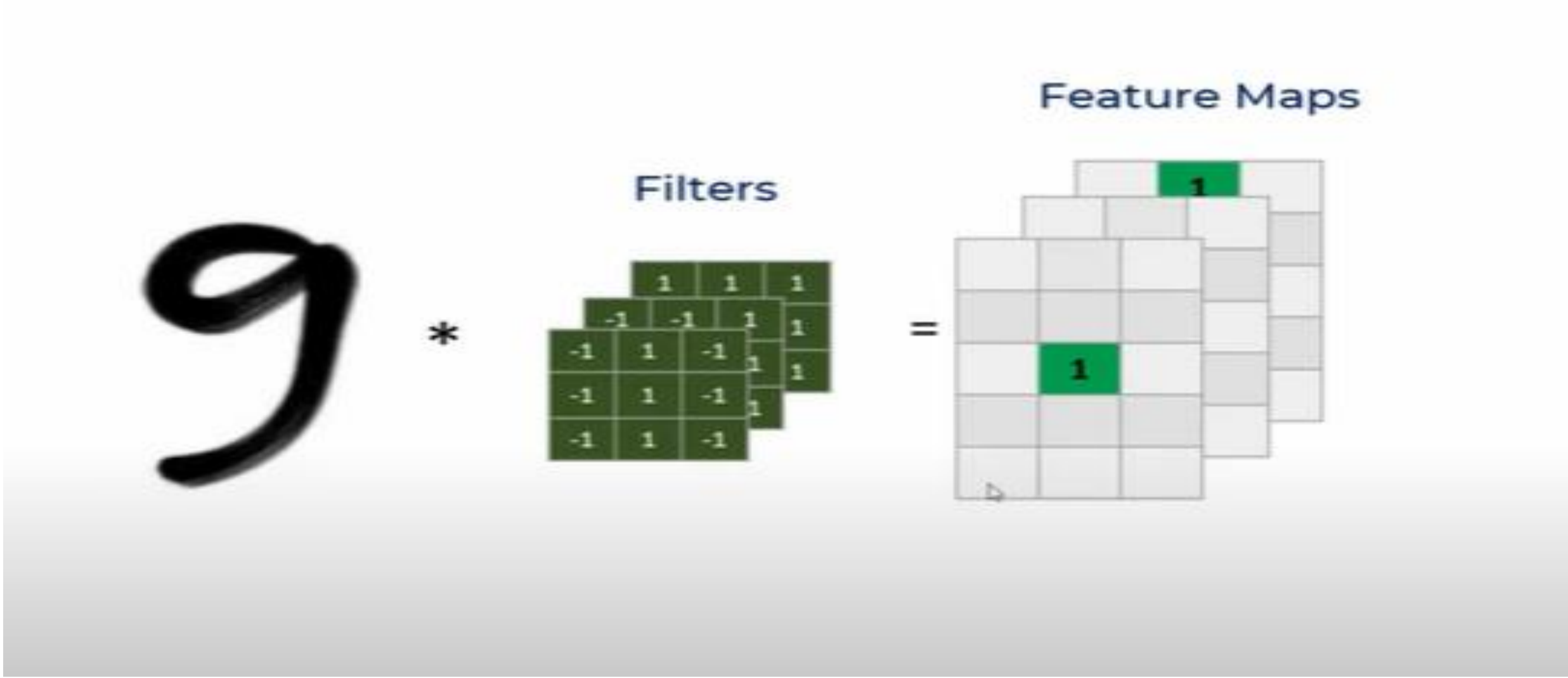


Application of different filters on the same input image:

- Application of multiple filters like loopy, vertical and diagonal line filter on the feature map is shown in the figure below.



Stacking of filters:
Filters are applied and are stacked as shown below



Formula for computing the matrix size of the output image or convolution layer.

- Formula for computing the output, given nxn input image is

  - **Input:** n X n
  - **Filter size:** f X f
  - **Output:** (n-f+1) X (n-f+1)

- The main drawbacks are:

1. Every time we apply a convolutional operation, the size of the image shrinks
2. Pixels present in the corner of the image are used only a few number of times during convolution as compared to the central pixels. Hence, we do not focus too much on the corners since that can lead to information loss

-

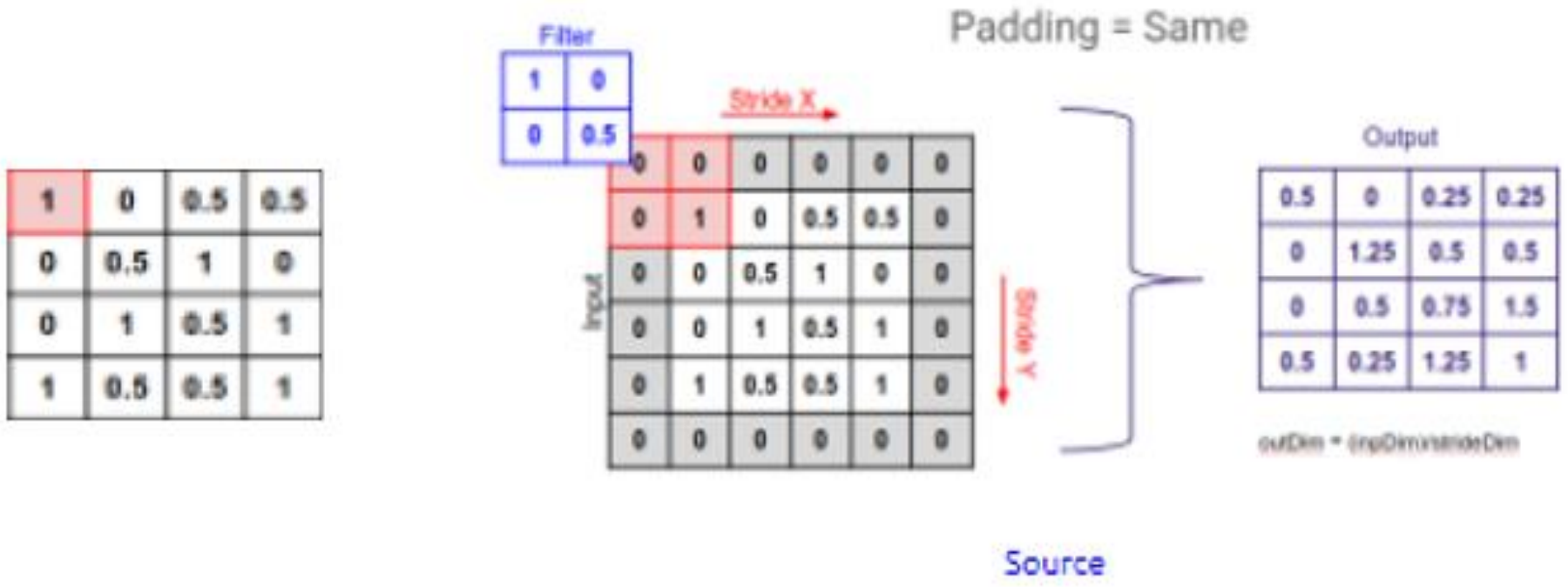
- Solution for the above problem is to use padding

Padding

- Padding is adding extra columns and rows on either sides with value zero.
- Padding makes the output size same as the input size, i.e.,
  $n+2p-f+1 = n$
- How many rows and columns should be padded depends on the size of the filter.
- If filter is of size 3x3 then 1 row and both on top and bottom will be added and also one column to the left and right will be added.
  So, $p = (f-1)/2$
- Adding padding to an image processed by a CNN allows for more accurate analysis of images.

Original image – Padding – After applying filter

- Original image is of size 4x4 and the image generated after filter is also of size 4x4.



Another example for padding

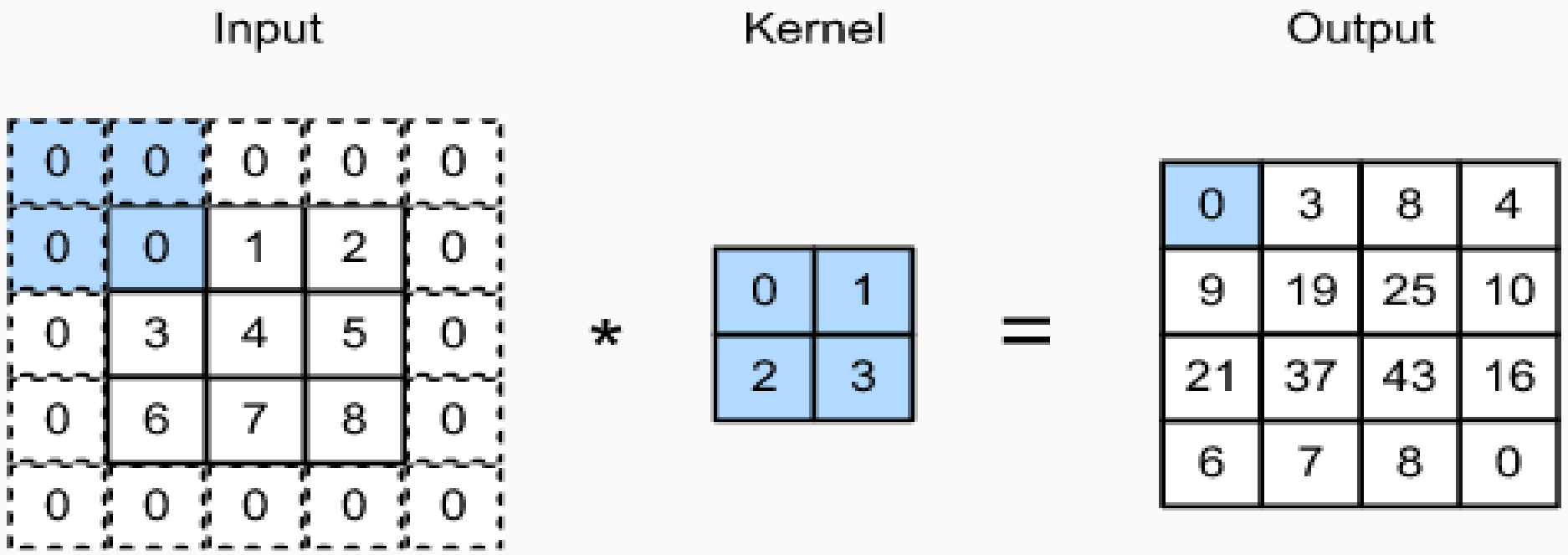
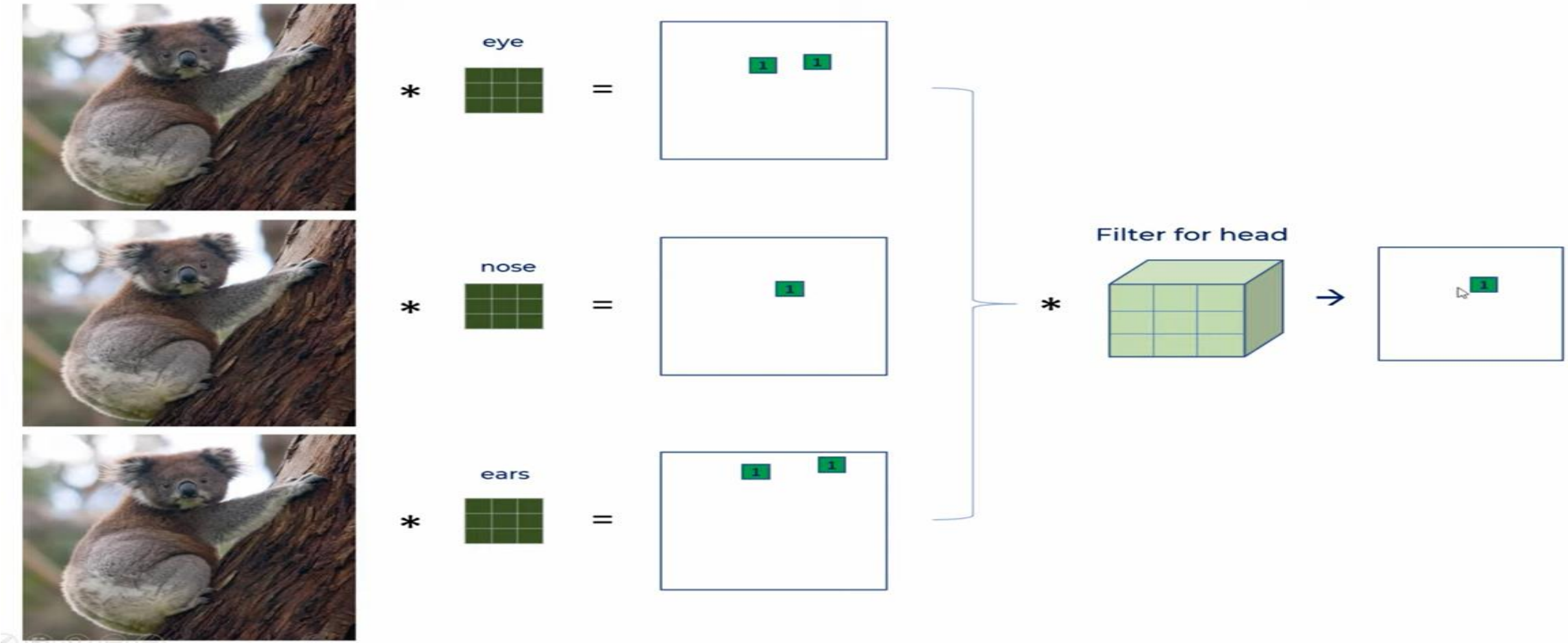


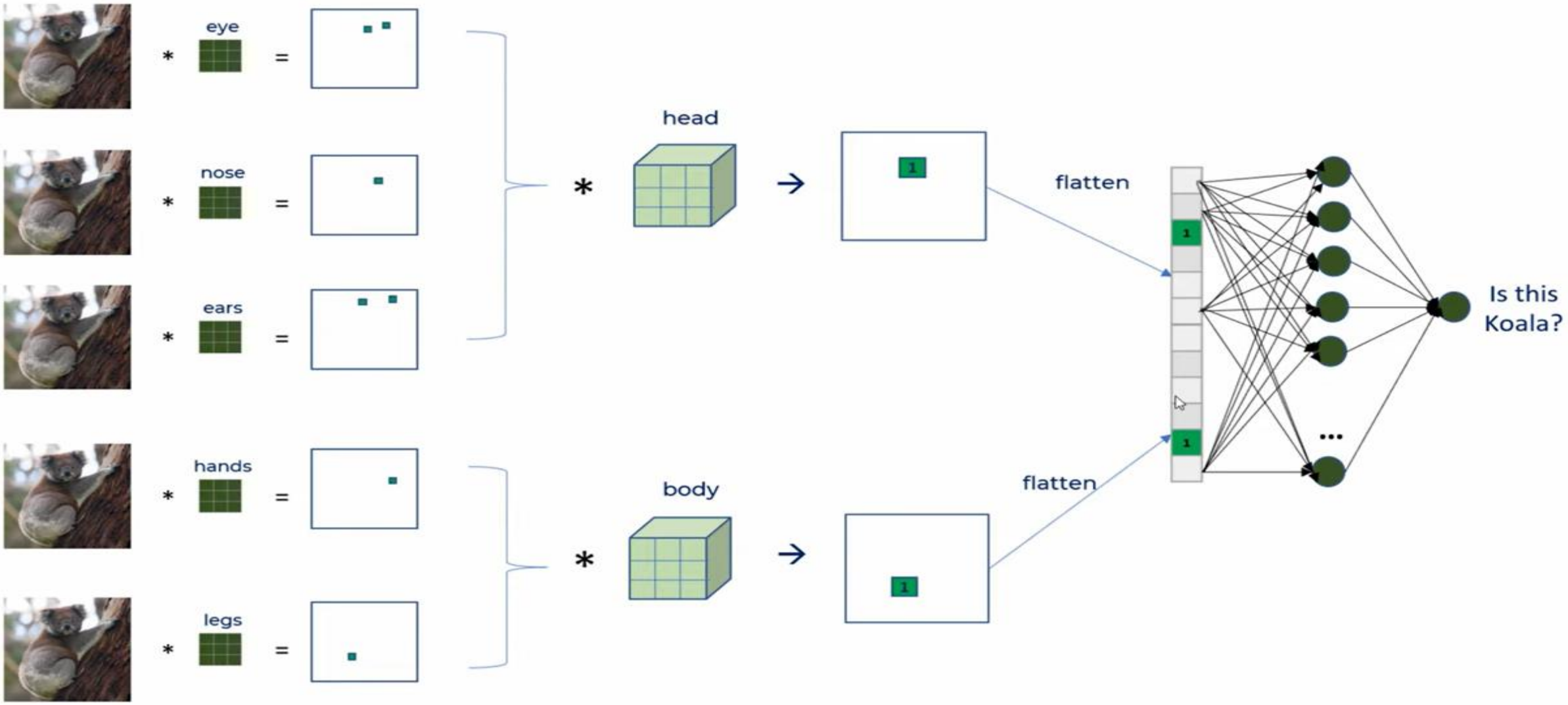
Fig. 6.3.1 Two-dimensional cross-correlation with padding.

Another example: Applying different filters to the image

Filters applied on an image to detect the head part of Koala

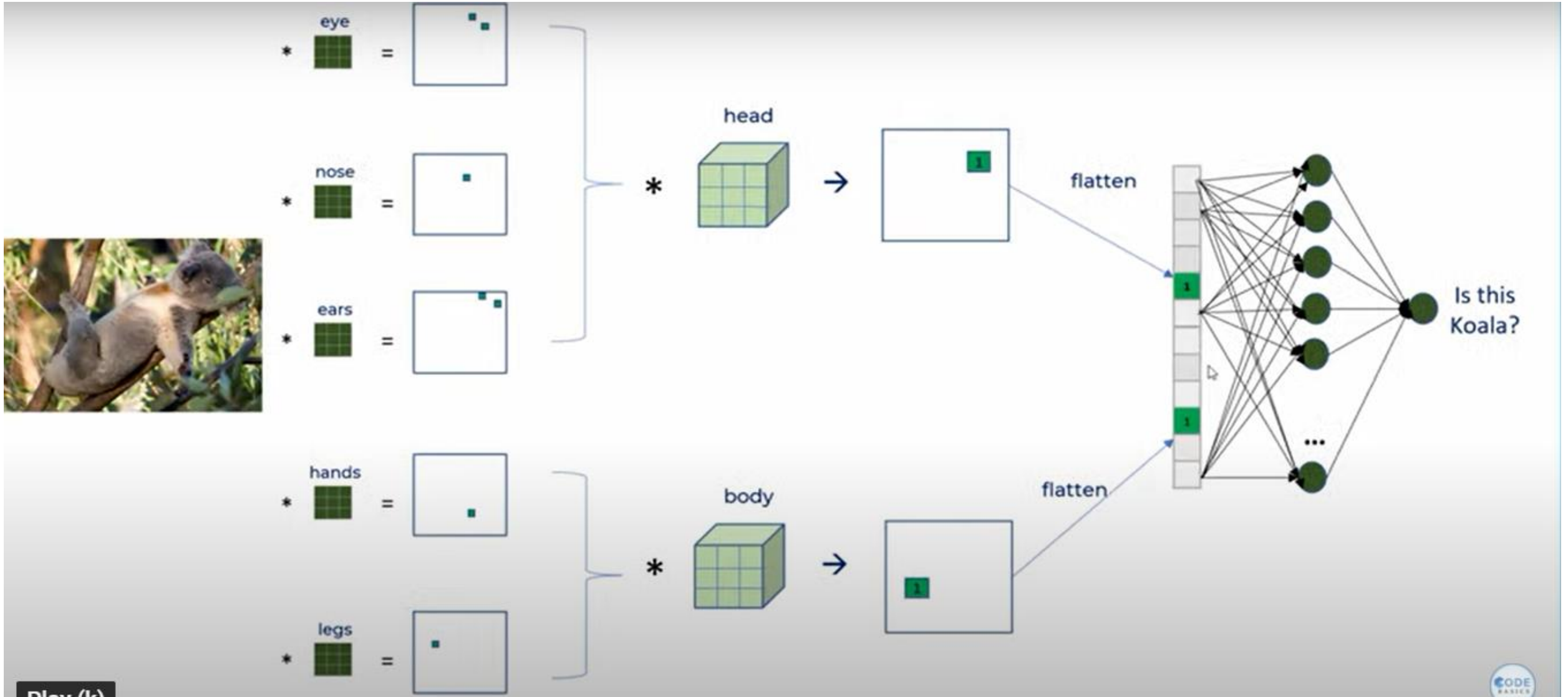


CNN applied on image to detect the presence of animal Koala

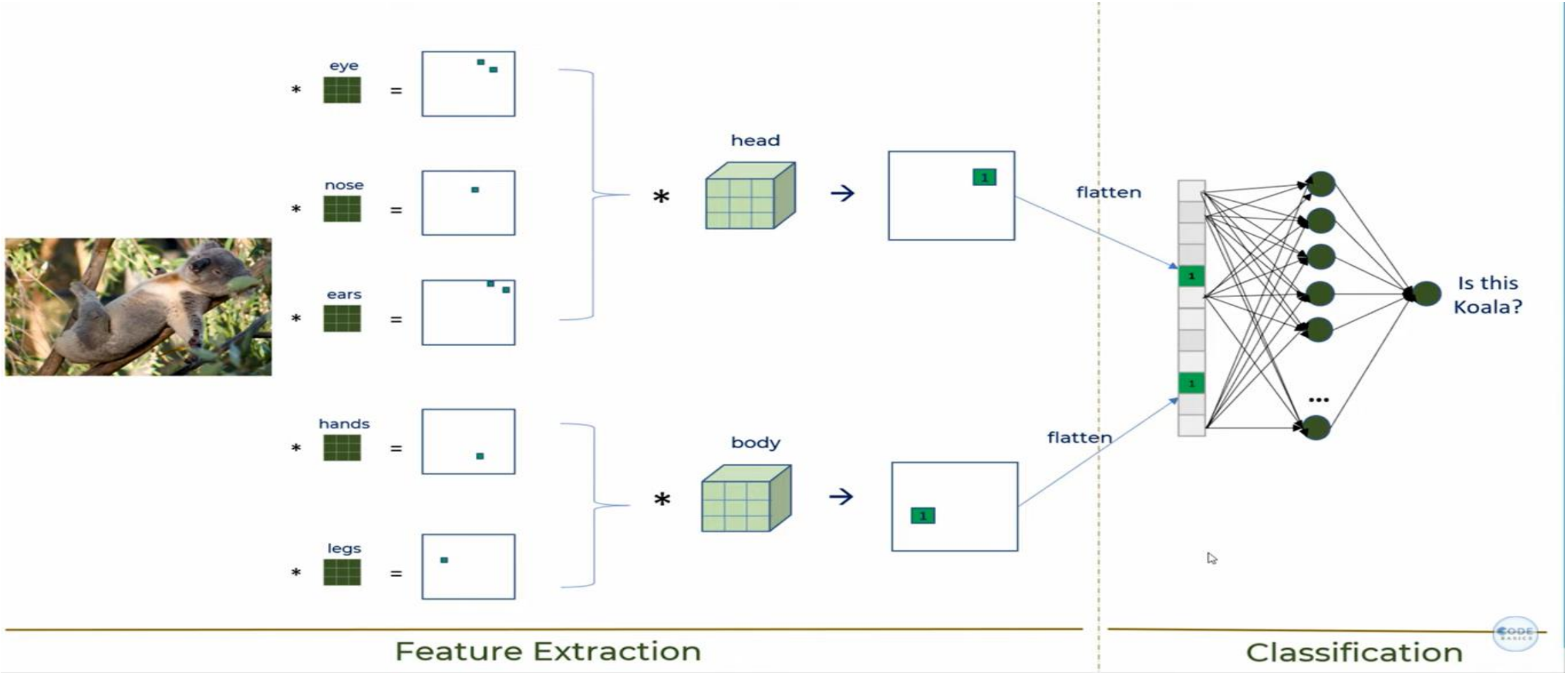


Even if the animal position changes the feature maps will identify different parts of animal Koala.

(Observe the change in feature map position comparing it with the previous slide)



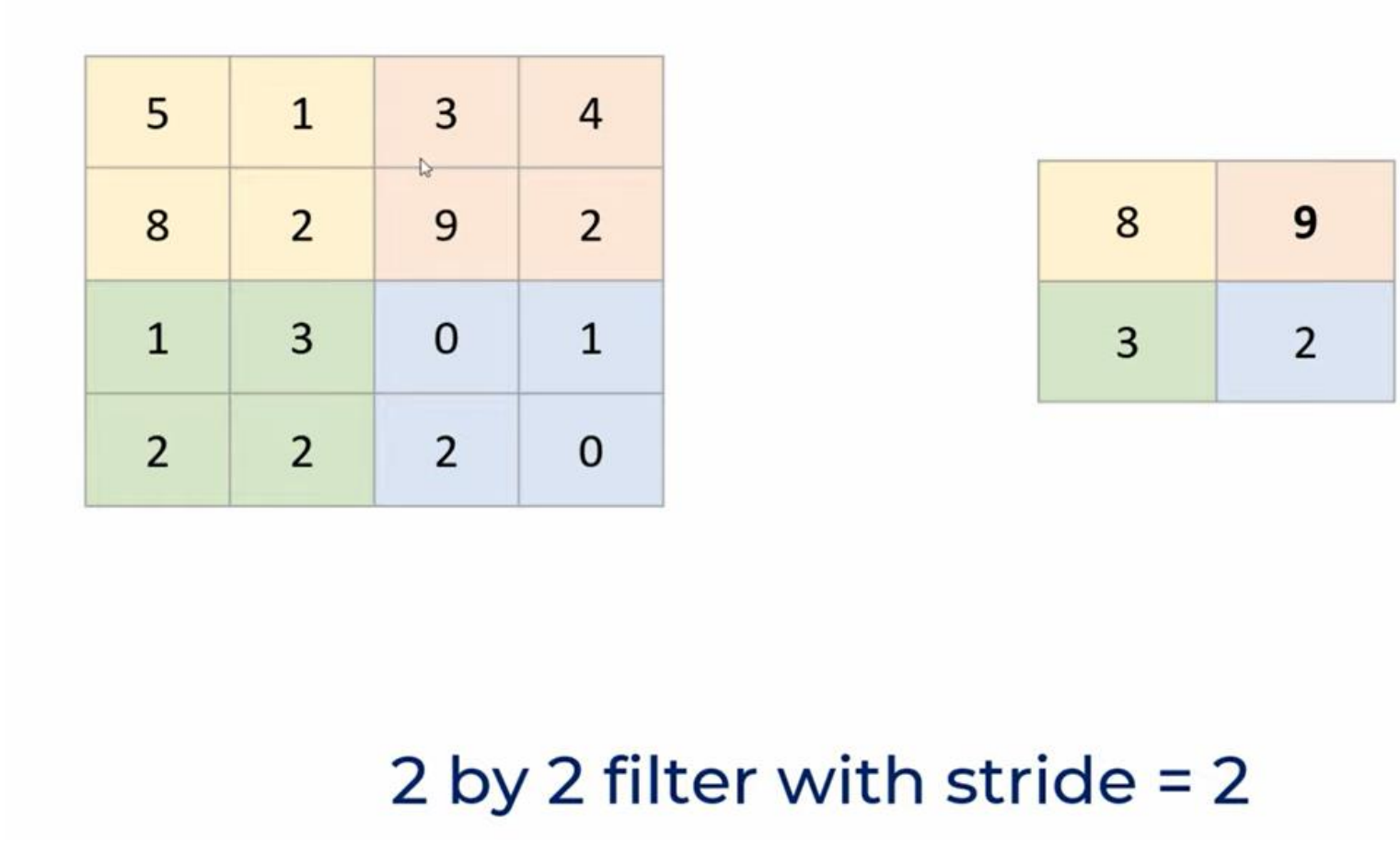
Complete CNN architecture, which does Feature extraction and classification using fully connected Neural network at the end.



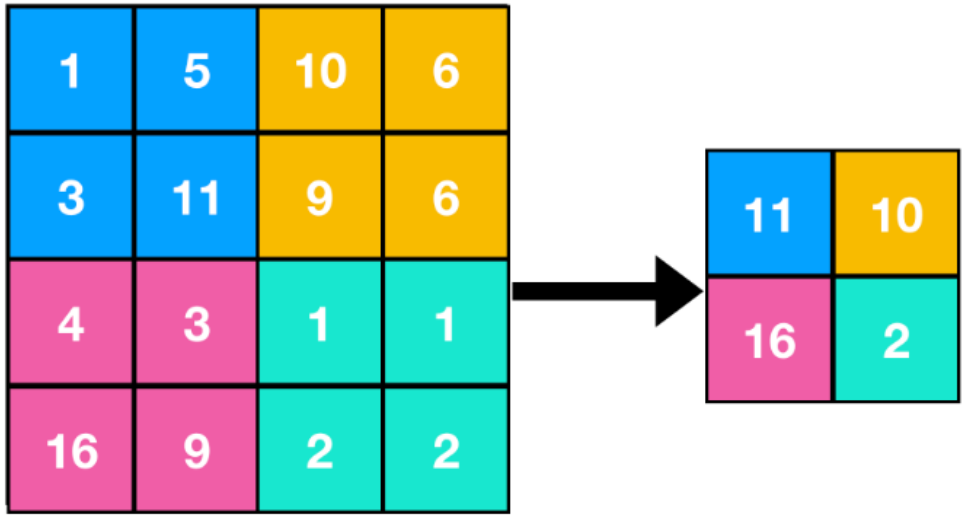
Pooling Layer

- Pooling layers are used to reduce the dimension.
- 3 Popular pooling methods are
  - Max pooling (Very popularly used)
  - Min Pooling
  - Average Pooling

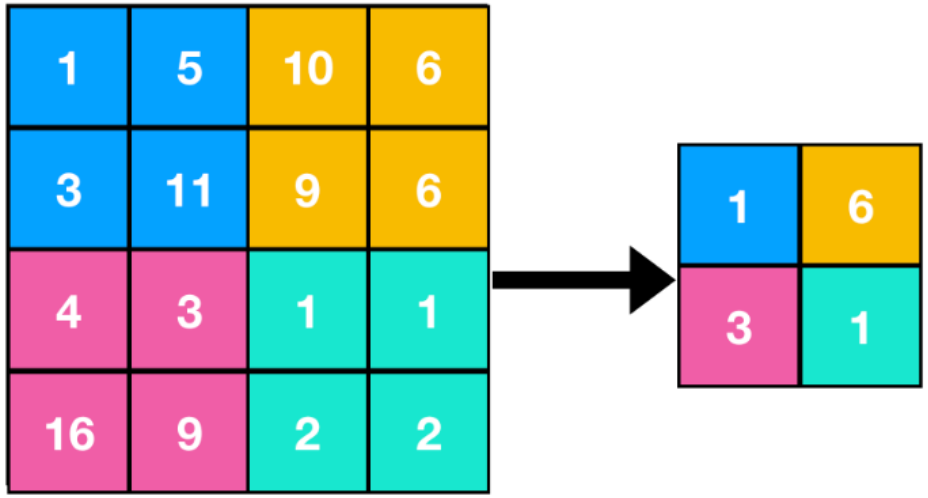
Max Pooling



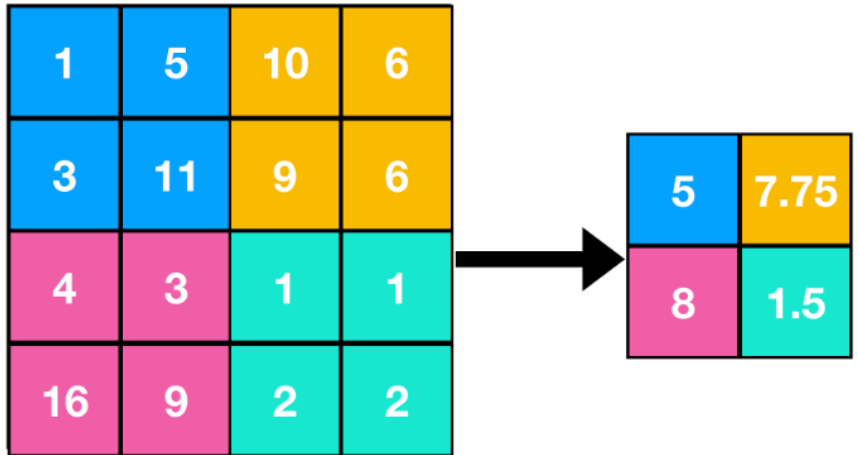
Max, Min and Average Pooling



Max Pooling



Min Pooling



Average Pooling

Location invariant loop detection : Use of filter, activation function and max pooling

Shifted 9 at different position

1	1	1	-1	-1
1	-1	1	-1	-1
1	1	1	-1	-1
-1	-1	1	-1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1
1	-1	-1	-1	-1

*

Loopy pattern filter

1	1	1
1	-1	1
1	1	1

→

1	-0.11	-0.11
0.11	-0.33	0.33
0.33	-0.33	-0.33
-0.11	-0.55	-0.33
-0.55	-0.33	-0.55

→

1	0	0
0.11	0	0.33
0.33	0	0
0	0	0
0	0	0

Max pooling

→

1	0.33
0.33	0.33
0.33	0
0	0

CNN Architecture

- A typical CNN Architecture is as shown in the figure below

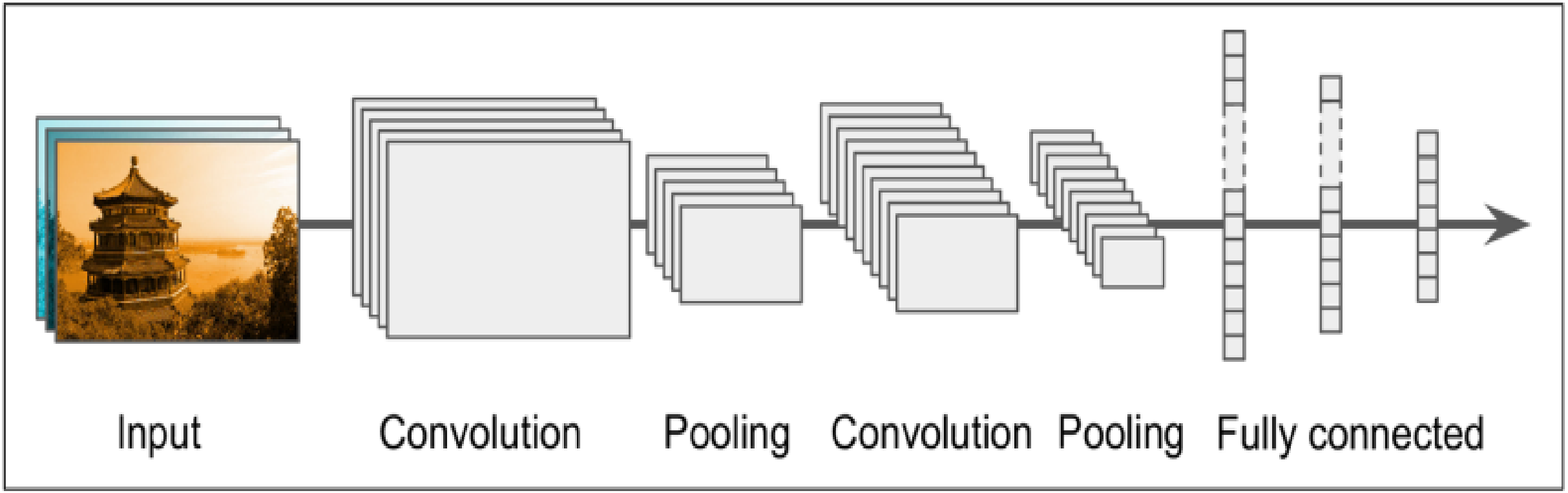
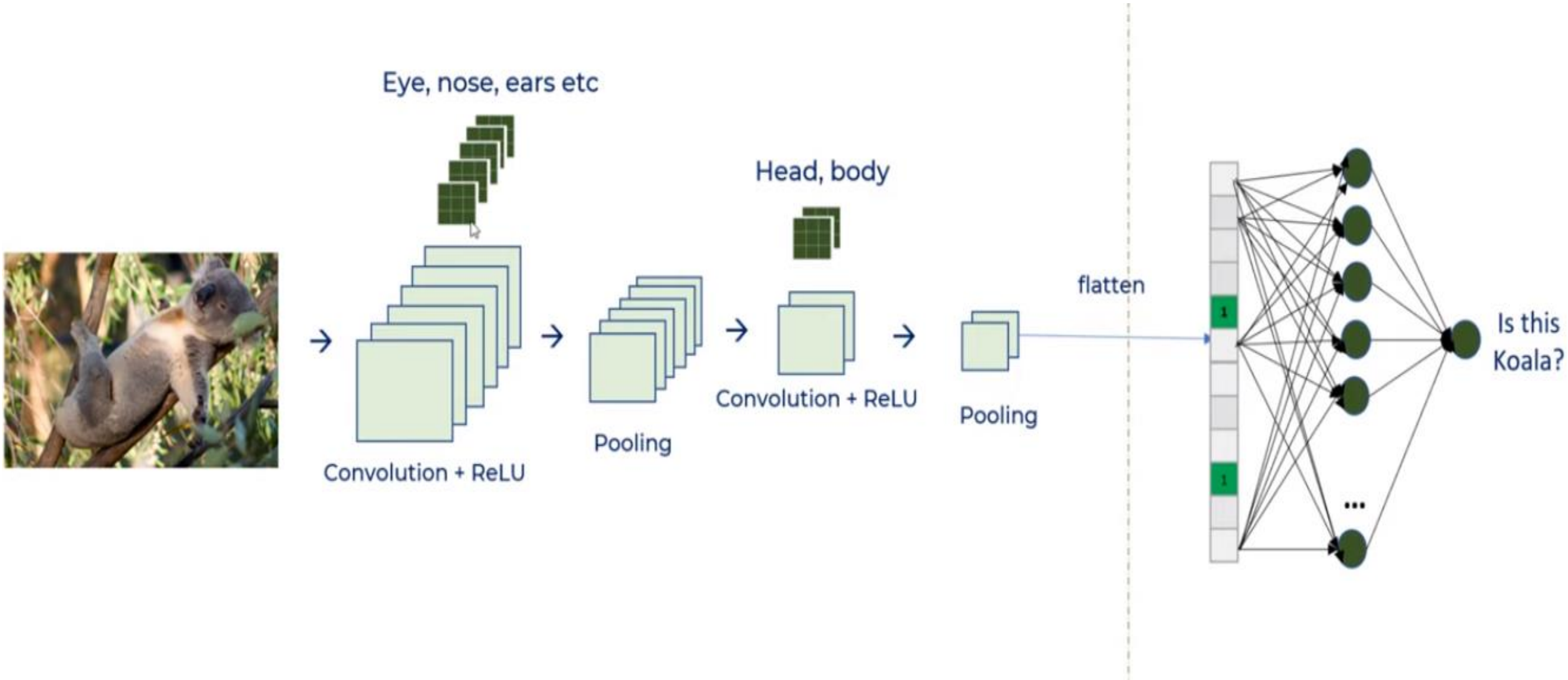


Figure 13-9. Typical CNN architecture

CNN architecture Explained

- A typical CNN architecture stack a few convolution layers each followed by ReLU layer.
- Then a pooling layer.
- Then another few convolutional layers , then another pooling layer, and so on.
- The image gets smaller and smaller as it progresses through the network, but it also typically gets deeper and deeper.
- At the top of the stack, a regular feedforward neural network is added composed of a few fully connected layers(+ ReLU), and the final layer outputs the prediction.
- Over the years, variants of this fundamental architecture have been developed , leading to amazing advances in the field.

Example for CNN Architecture.



End of Unit 3

UNIT – 4
Recurrent Neural Network

Recurrent Neural Network:

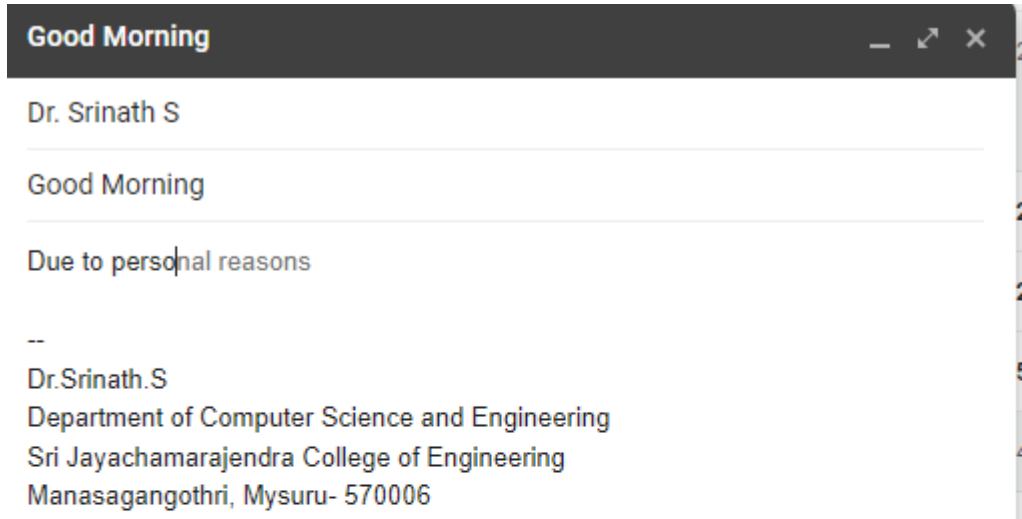
- Recurrent neurons,
- Basic RNN in Tensor Flow,
- Training RNN,
- Deep RNNs,
- LSTM Cell,
- GRU Cell,
- NLP

What is RNN?

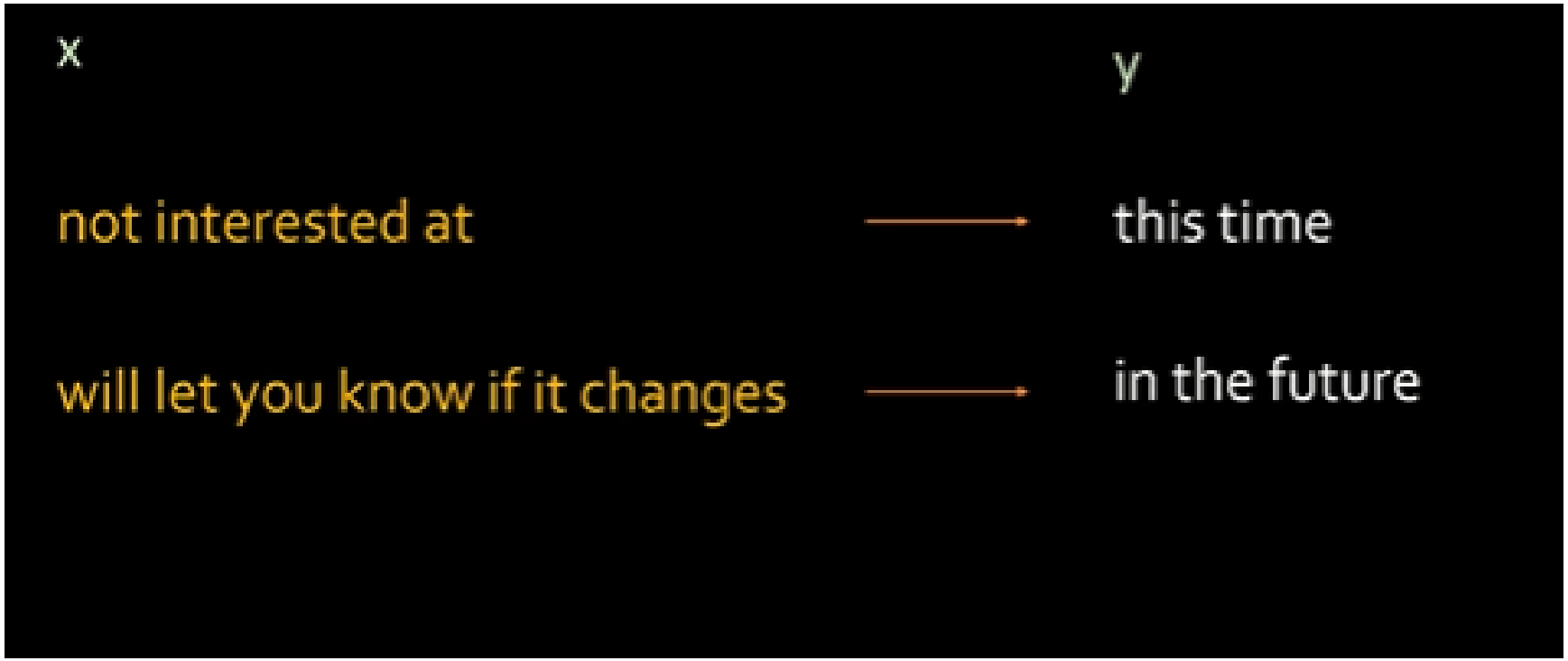
- RNN – Recurrent Neural Network.
- CNN can be used mainly for Image and also to some extent video.
- RNN is popularly used in NLP and other domains.
- When we type a sentence,

It automatically completes.

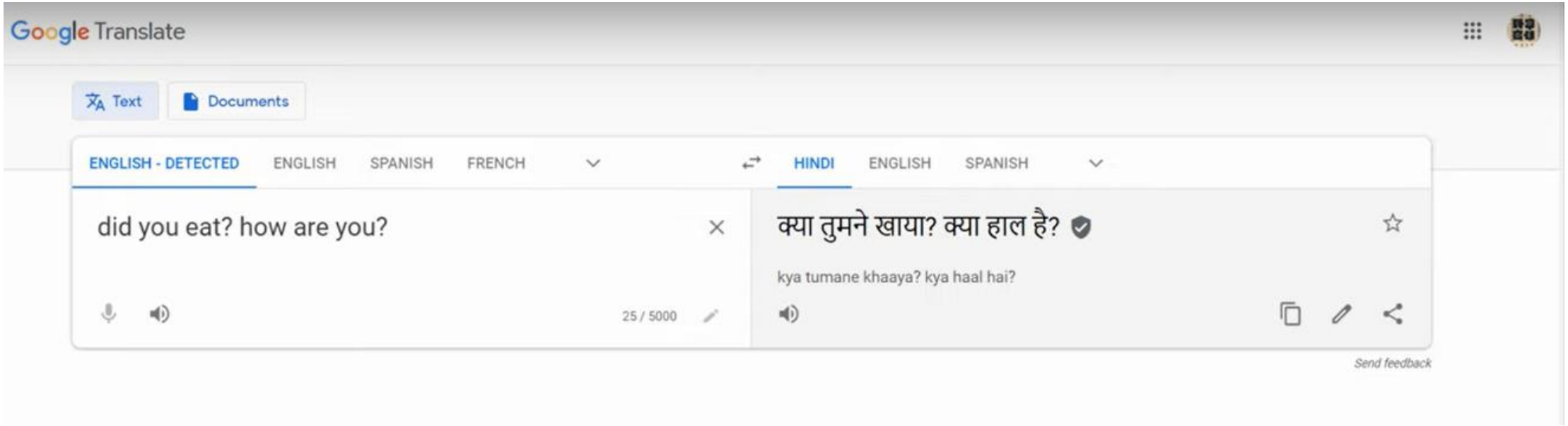
Google has RNN in it.



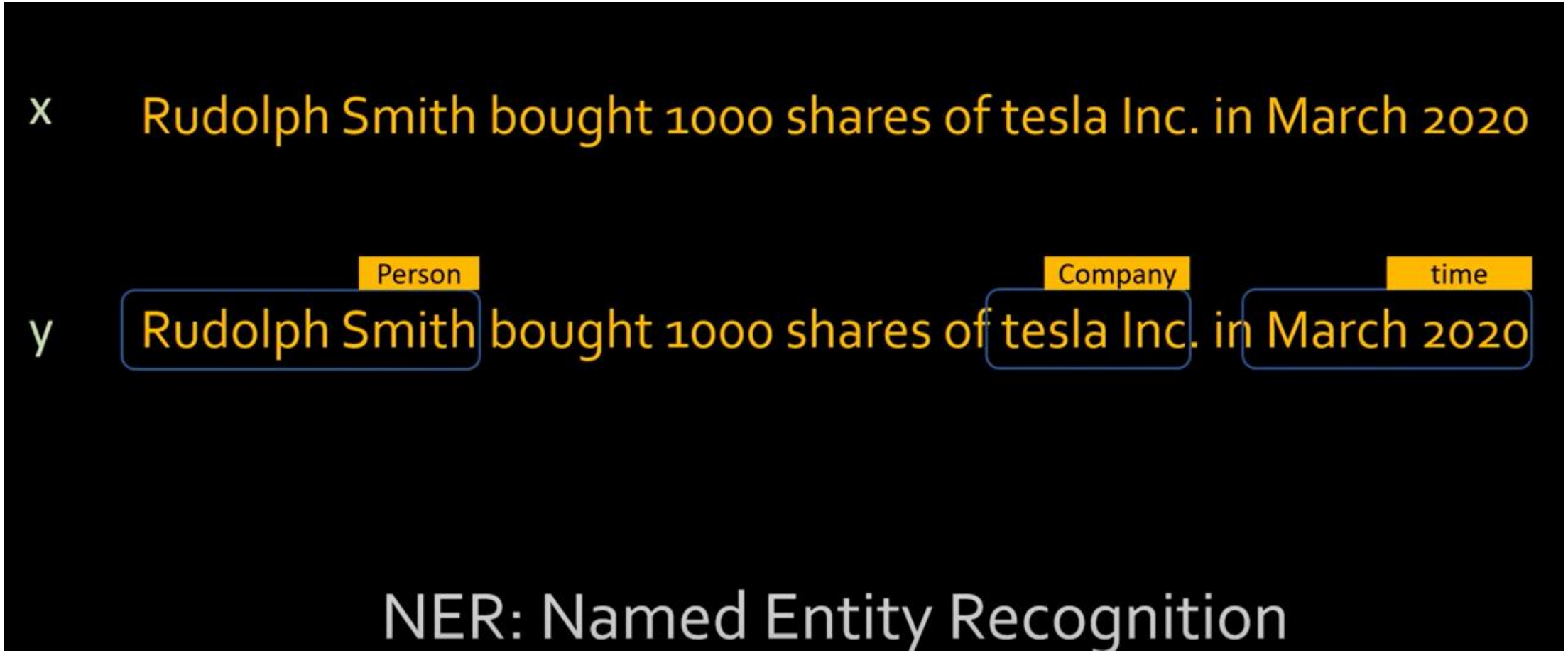
Application of RNN… auto completion.



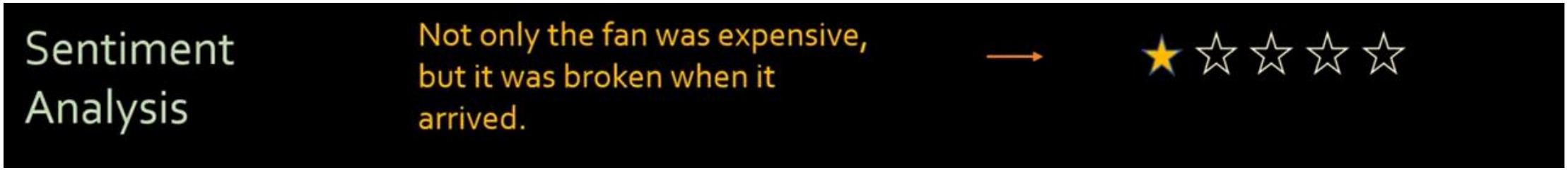
Application of RNN : Translation



Application of RNN : Named Entity Recognition

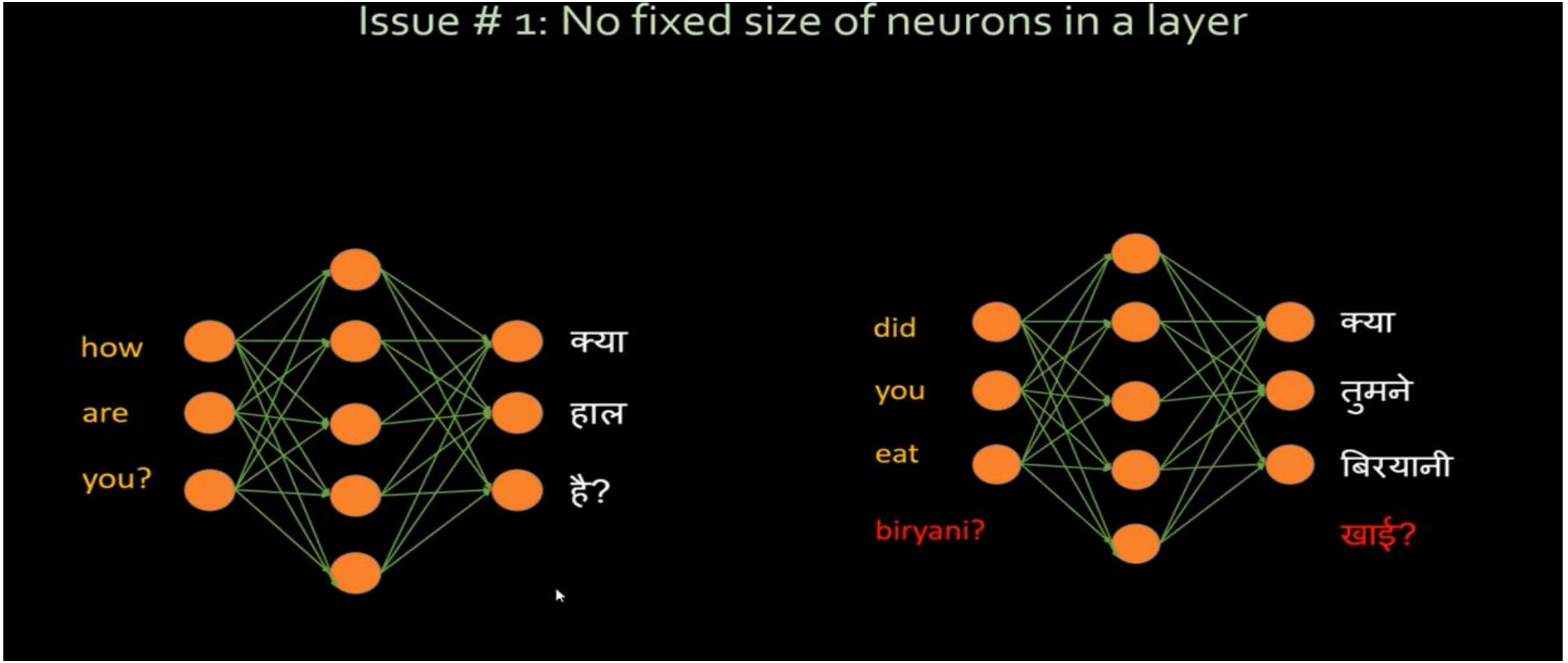


Sentimental Analysis



Summary of few applications

Why can’t an ordinary ANN is not enough?



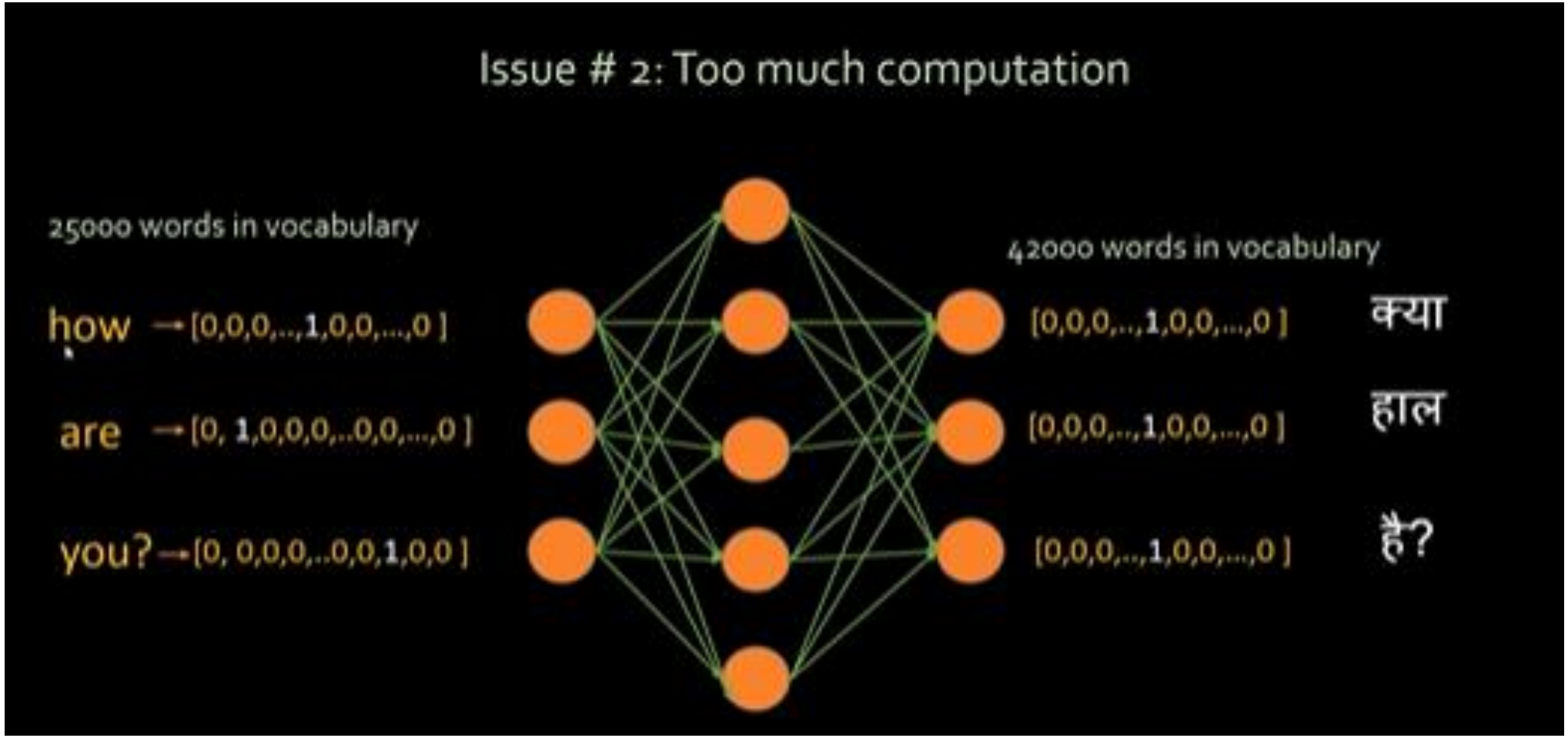
Ordinary ANN

- For an application like NLP, ordinary ANN may not be sufficient.
- The reason is ANN works on fixed number of inputs and outputs or classes.
- For applications like NLP, to work with fixed number of neurons is difficult. (refer picture in previous slide)
- On the other hand ..we can start with huge number neurons, If I am working with only few words… rest of the neurons will be having null values or zero values.
- But this is waste of computation time.

Some of the applications of RNN

- RNN is good for sequences of data
- Like NLP… say a sentence is positive or negative
- Time series
- Stock market prediction
- SPAM classifier.
- Machine Translation
- Video Tagging
- Text Summarization
- Call Centre Analysis
- Music composition…..

Too much of computation



Recurrent Neuron

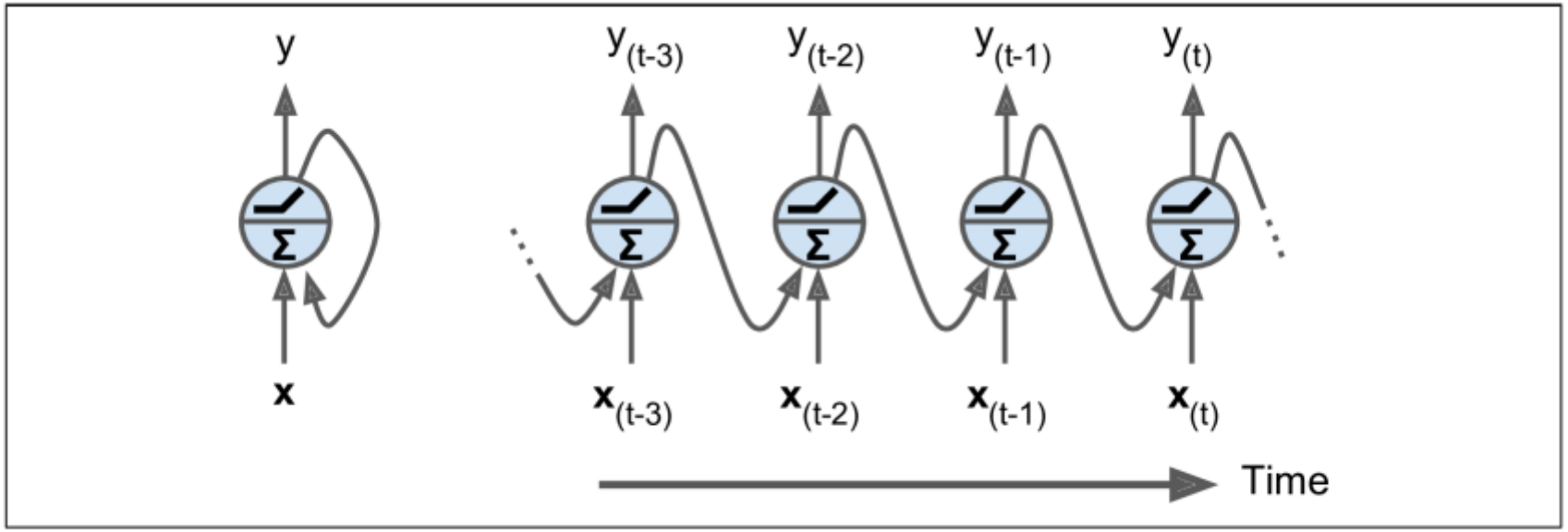
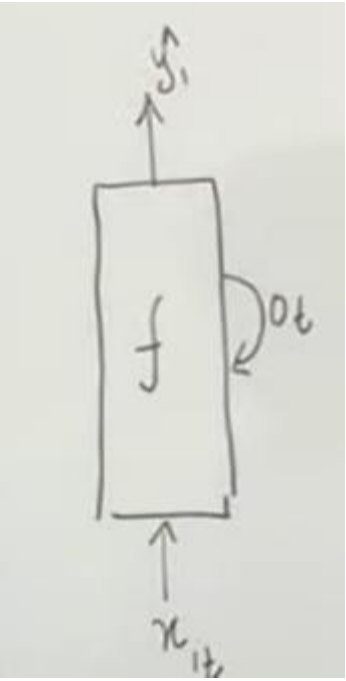


Figure 14-1. A recurrent neuron (left), unrolled through time (right)



A layer of recurrent neuron

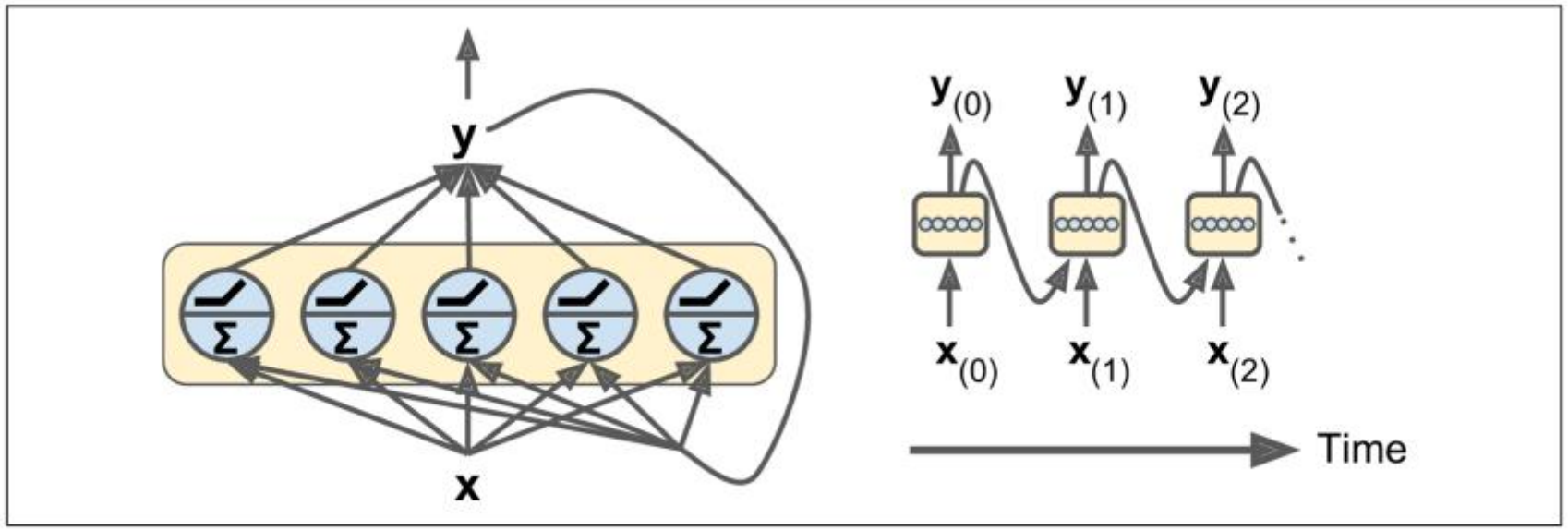
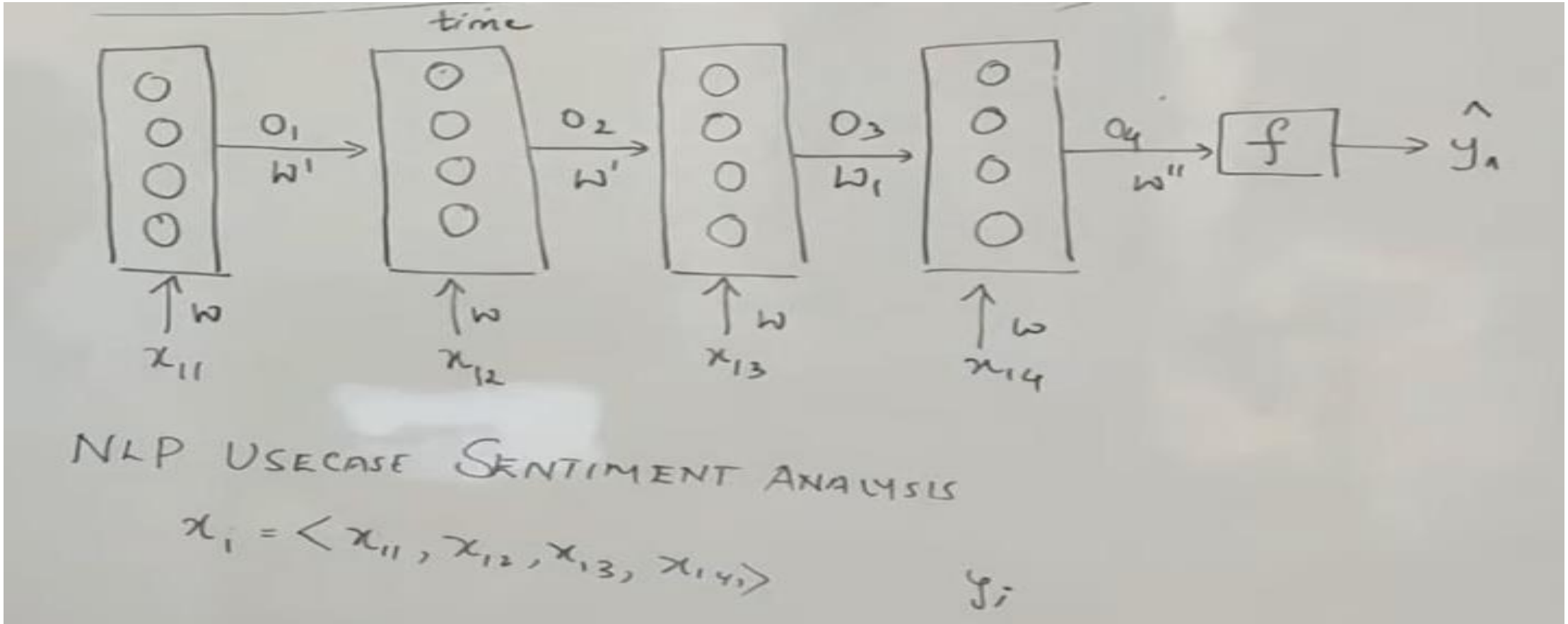


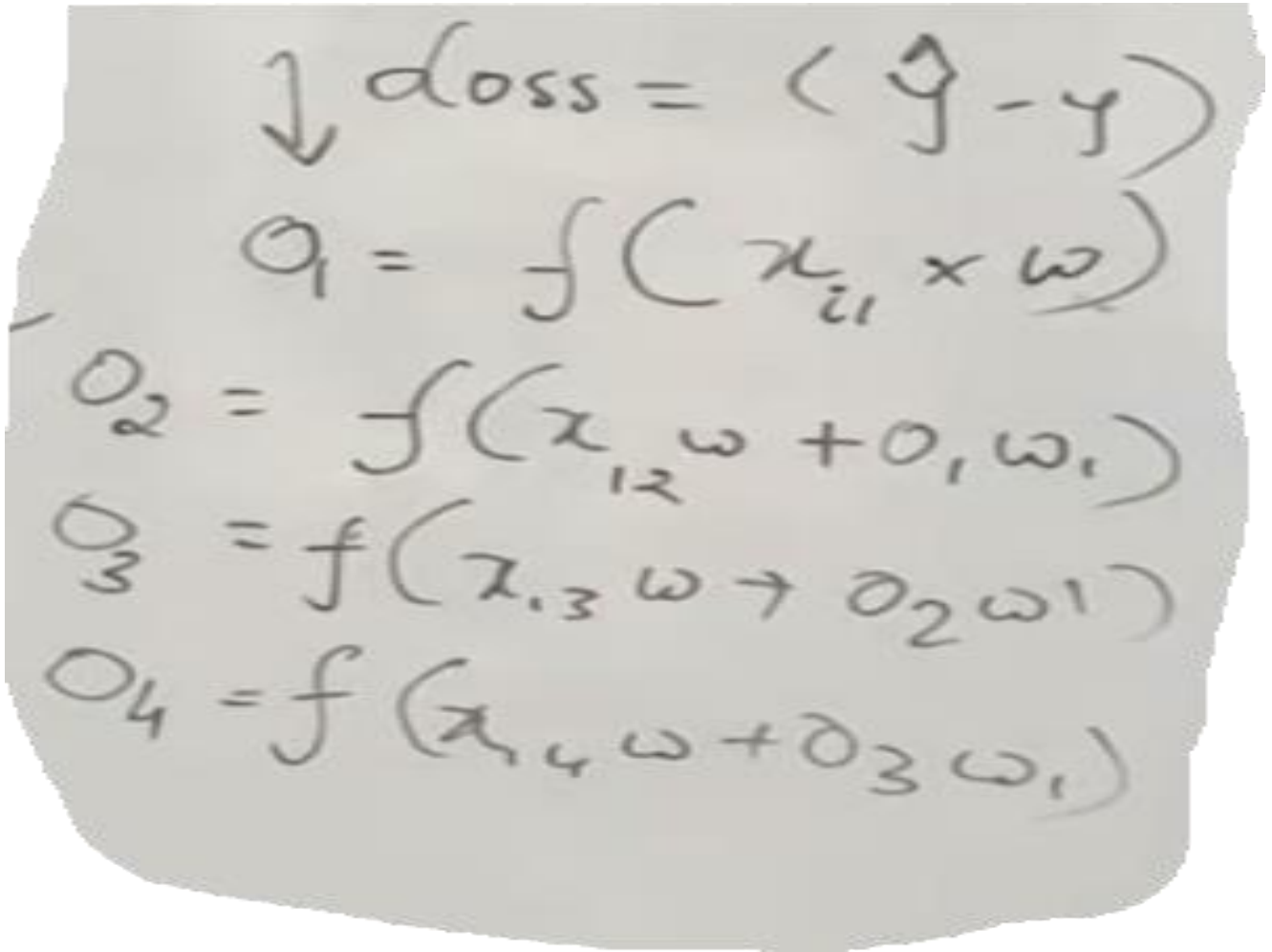
Figure 14-2. A layer of recurrent neurons (left), unrolled through time (right)

A layer of RNN with example

- Consider X1 as sentence with 4 words..x11,x12,x13,x14 … RNN in this example is to find out whether it is positive feedback or not.



Computation of intermediate outputs



Basic RNNs in TensorFlow

- The static_rnn() function creates an unrolled RNN network by chaining cells.

- Code for Basic RNN usage is as given below.

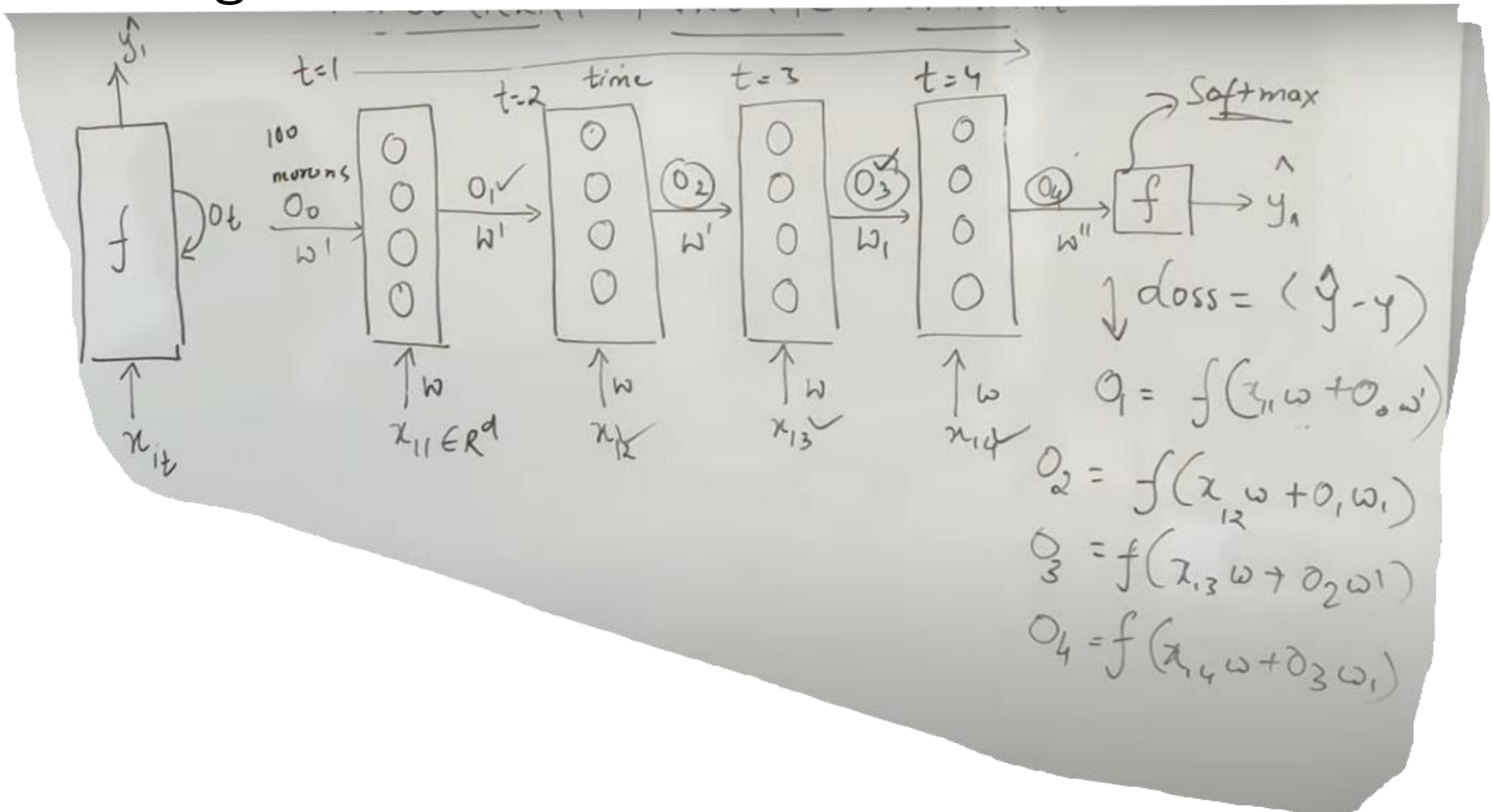
```
basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=n_neurons)
```

Training RNN

- To train an RNN, use regular B**ackpropagation.** This strategy is called backpropagation through time (BPTT).
- Consider the example discussed earlier.

(Presented in next slide also)

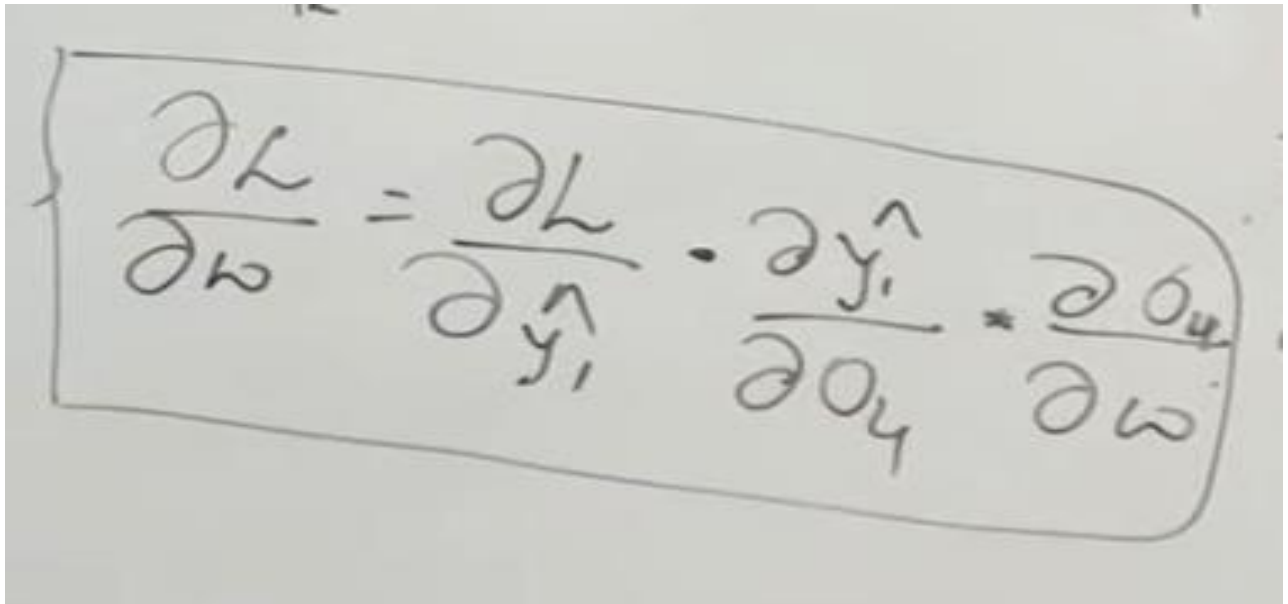
Training RNN



- Now weights are to be updated. Chain rule to update a wait is shown below. Similar changes are to be made for other waits.

- However during gradient descend, again problems like vanishing gradient and exploding gradient problem exists.

- To handle this, in RNN LSTM (Long short term memory) is used.



Deep RNN

- It is quite common to stack multiple layers of cells.
- This gives you a deep RNN.

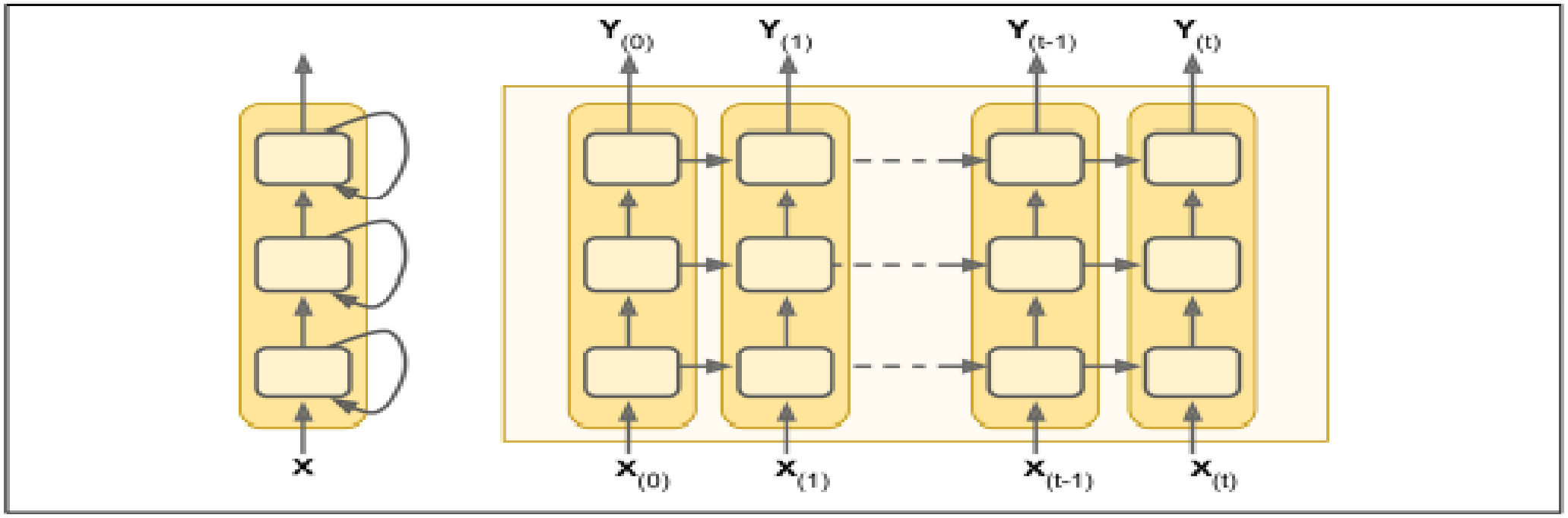
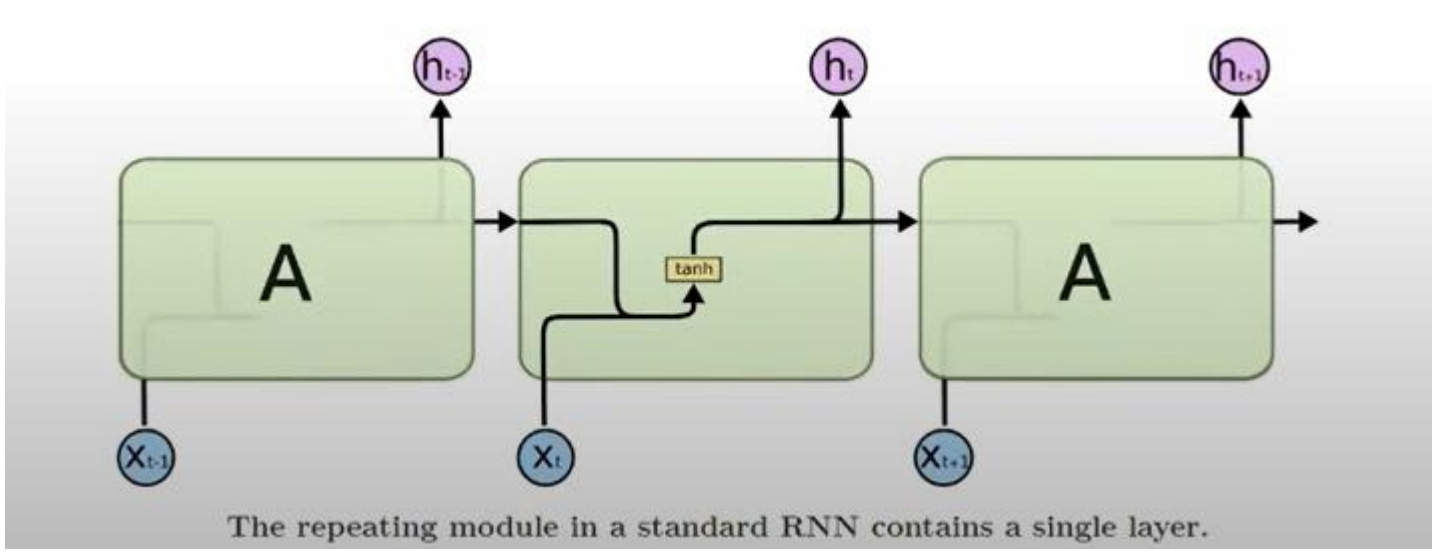


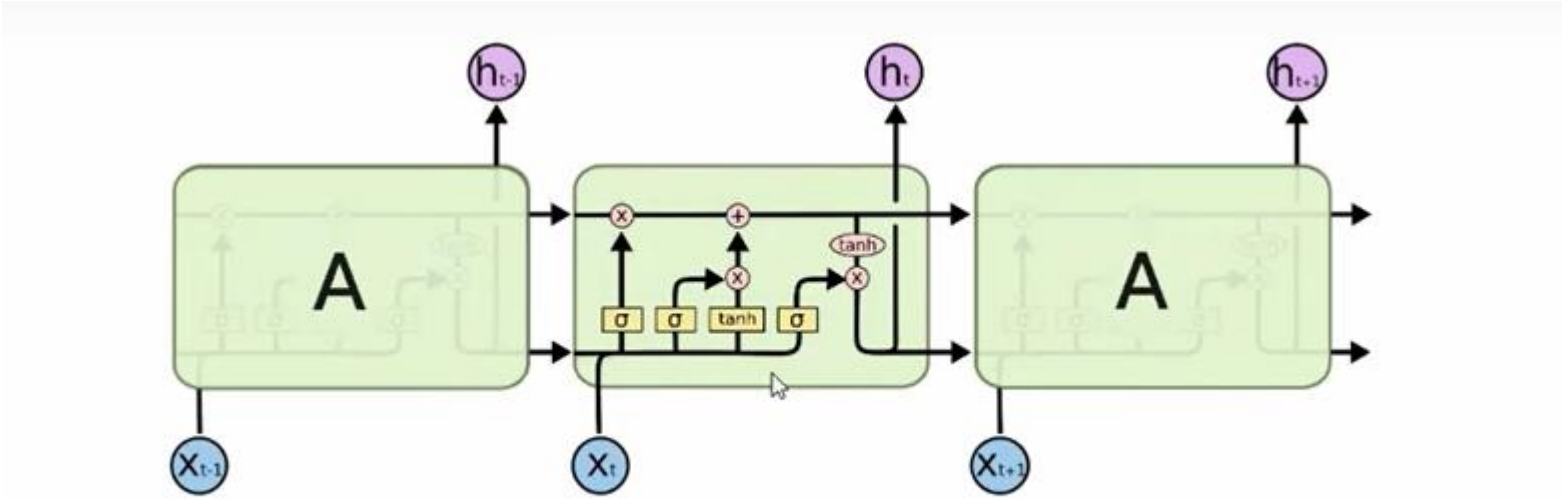
Figure 14-12. Deep RNN (left), unrolled through time (right)

LSTM

- Let us compare the architecture of regular RNN and LSTM



The repeating module in a standard RNN contains a single layer.



The repeating module in an LSTM contains four interacting layers.

LSTM will have…

- Memory cell
- Forget Gate
- Input Gate
- Output Gate

Memory Cell

- Consider the statement:

- “I am Srinath. I am 48 years old”…… Here Noun is Srinath… The RNN system memory cell should remember Noun as Srianth

- If the next statement “ My brother is Raveesh”.
- Now the context is changed… now the system should forget the old data and remember the new data. This activity is Forget gate.

Unit - 5

Auto Encoders

Syllabus

- **Autoencoders:**
- Efficient data representation,
- stacked autoencoders,
- Unsupervised pretraining using SA,
- Denoising, Sparse autoencoders,
- variational and other autoencoders.

- **Reinforcement Learning:**
- Learning to optimize rewards,
- policy search,
- Neural network polices,
- Evaluating actions,
- Policy gradients,
- Markov decision processes,
- TDL and Q-learning

What is Autoencoders?

- Autoencoders are artificial neural networks capable of learning efficient representations of the input data, called coding's, without any supervision (i.e., the training set is unlabeled).

- These coding's typically have a much lower dimensionality than the input data, making autoencoders useful for dimensionality reduction.

- More importantly, autoencoders act as powerful feature detectors, and they can be used for unsupervised pretraining of deep neural networks.

- Lastly, they are capable of randomly generating new data that looks very similar to the training data; this is called a generative model. For example, you could train an autoencoder on pictures of faces, and it would then be able to generate new faces

Efficient Data Representations

- An autoencoder is always composed of two parts:

  An encoder (or recognition network) that converts the inputs to an internal representation,

  followed by a decoder (or generative network) that converts the internal representation to the outputs.

- An autoencoder typically has the same architecture as a Multi-Layer Perceptron except that the number of neurons in the output layer must be equal to the number of inputs.
- In this example, there is just one hidden Autoencoders layer composed of two neurons (the encoder), and one output layer composed of three neurons (the decoder).
- The outputs are often called the reconstructions since the autoencoder tries to reconstruct the inputs.

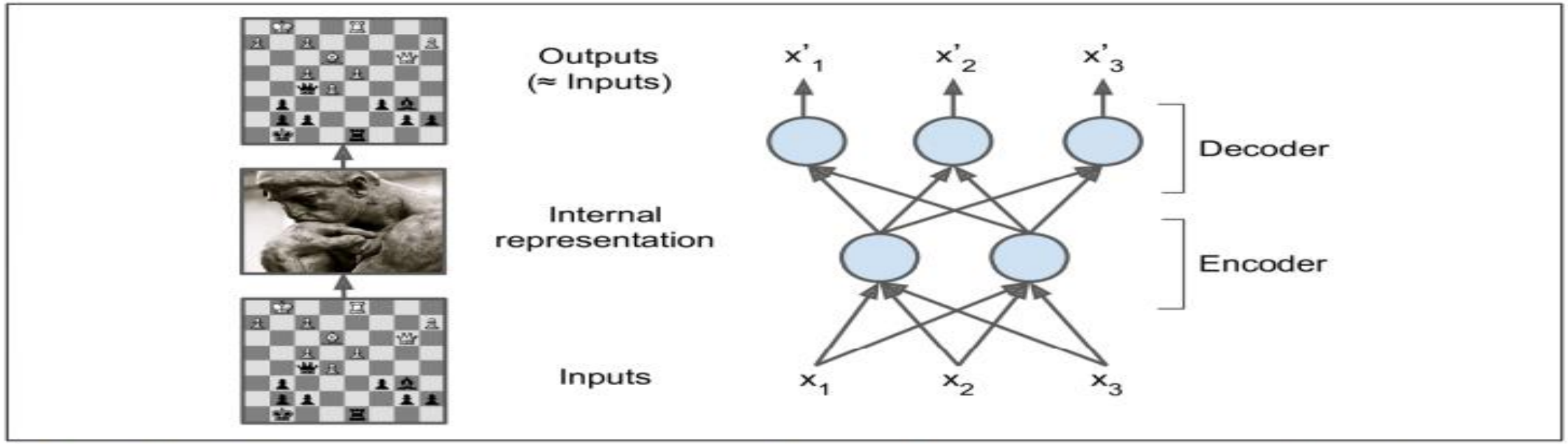


Figure 15-1. The chess memory experiment (left) and a simple autoencoder (right)

How autoencoder works?

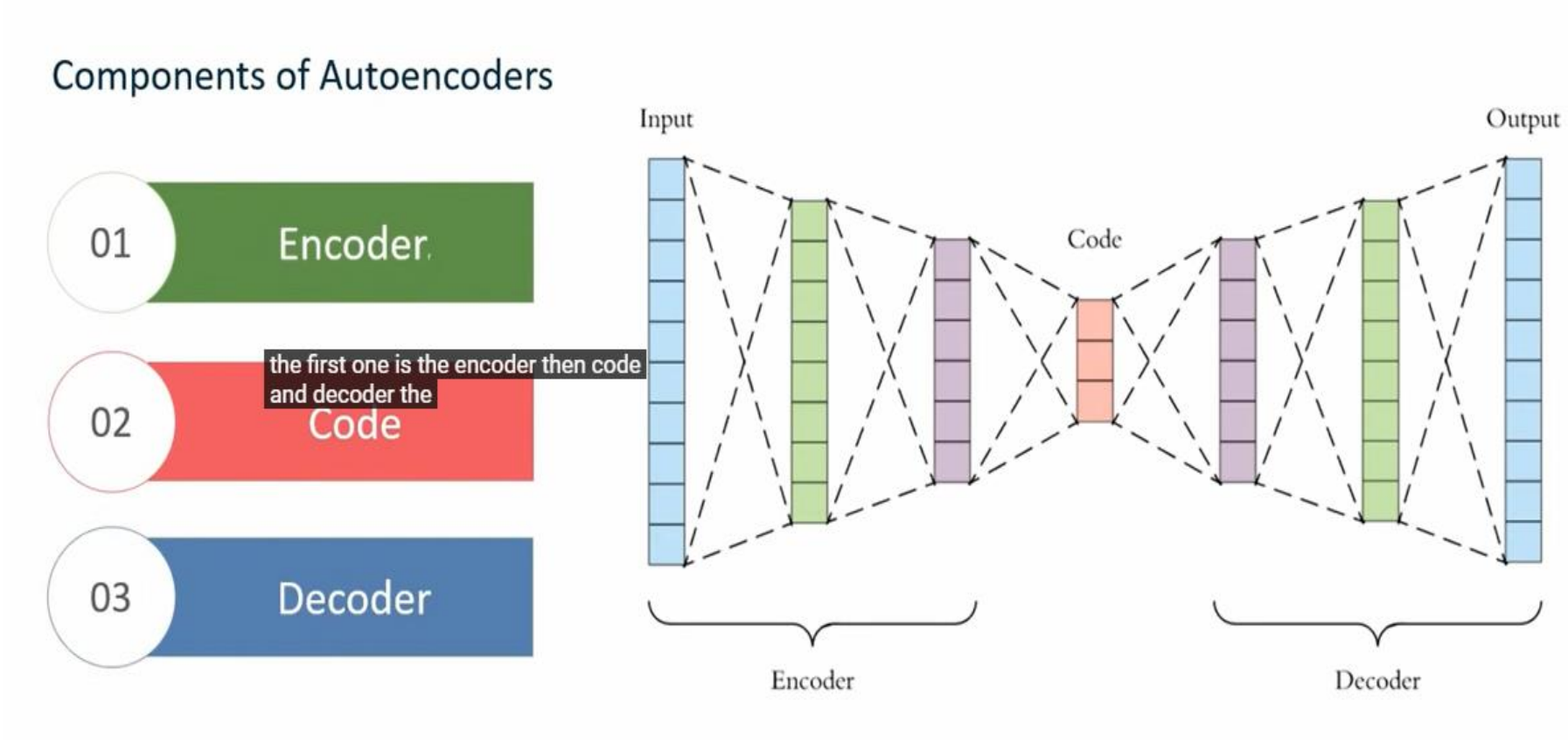
- Compression is to convert the input data into lower dimension, and later using the lower dimension data it can be reconstructed which will be close to original data.
- Autoencoders are more accurate compared to PCA

How autoencoder works?

- Autoencoder are popularly used for dimensionality reduction and noise elimination.
- Autoencoders are simple network, which converts input into output with minimal possible errors.
- It is unsupervised ML algorithm that applied back propagation, which sets target value similar to input.

Components of Autoencoders

- Three components of Autoencoders are:
- 1. Encoder
- 2. Code
- 3. Decoder.



Components

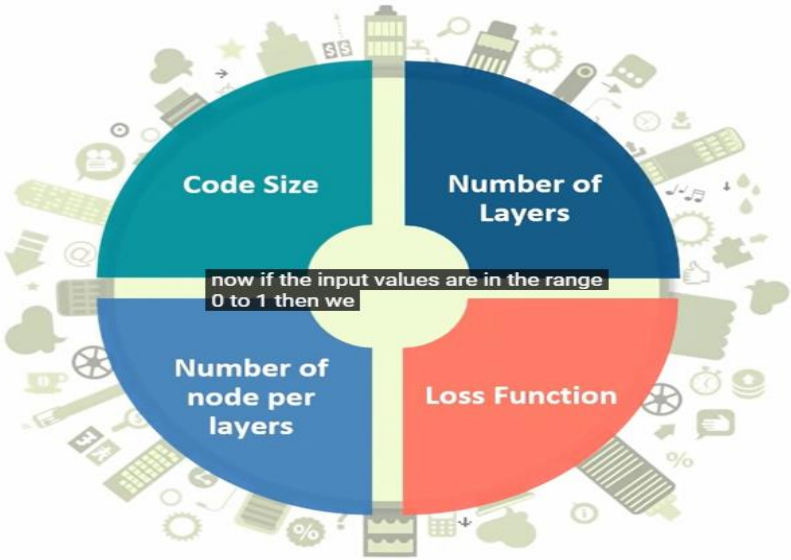
- Encoder layer compresses the input into a different representation, which is of reduced dimension.
- The next component is the code which is the compressed data, which is called latent space data.
- Decoder, decodes the code and reconstructs the original data.

Properties of Autoencoders

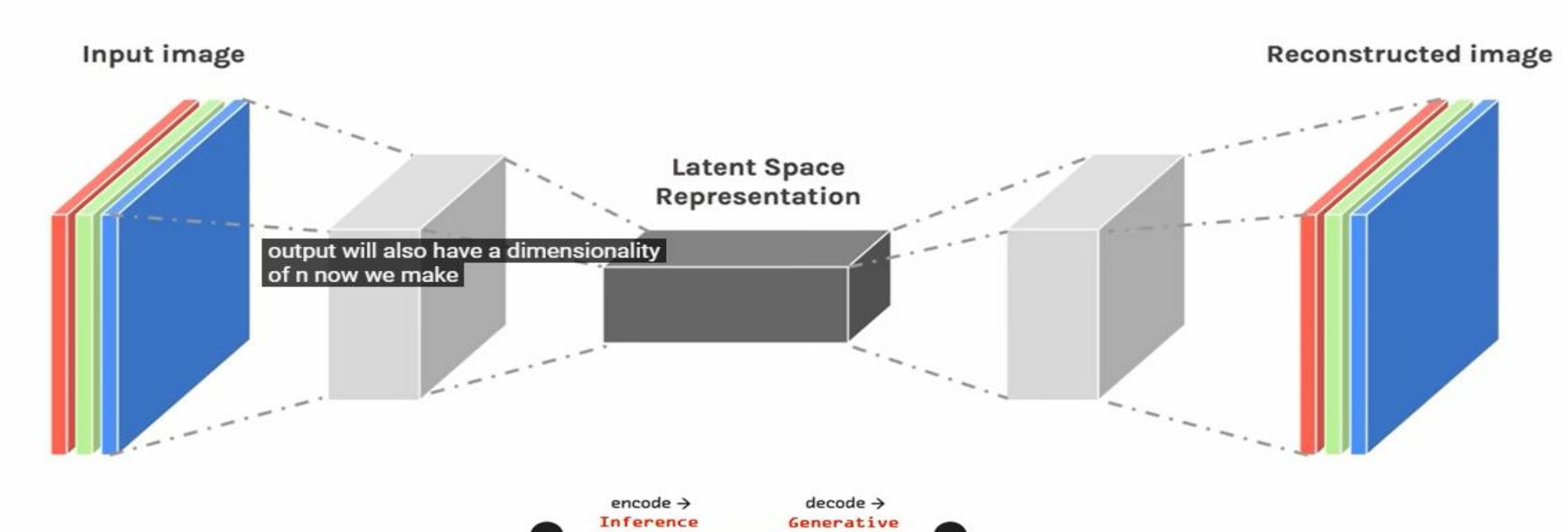
- Autoencoders are basically used for compression.
- They are unsupervised as they don’t require explicit labels to train
- Autoencoders are lossy, which means that the decompressed data will be degraded when compared to original data.

Training the autoencoders

- 4 parameters are required while training the auto encoders.
  - Code Size : Data in the middle of the auto encoder after compression.
  - Number of layers : Number of neural layers used
  - Number of nodes per layer : Number of neurons per layer
  - Loss function : It can be mean squared error or binary entropy



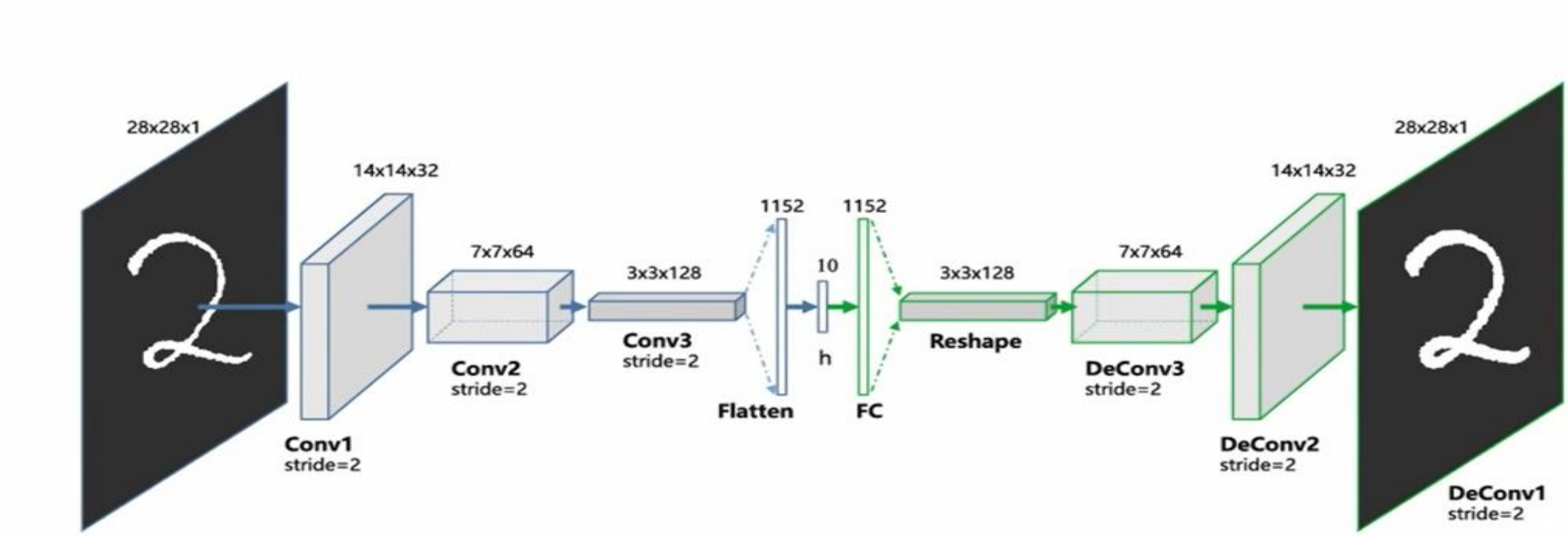
Architecture of Autoencoders



Encoder

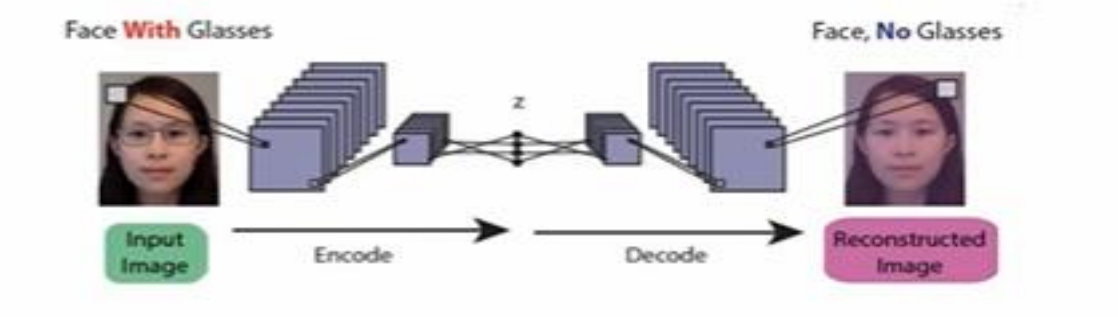
- The layers between input and output of autoencoders are of lesser dimension.
- Input if it is of 28x28 image= 784 pixels, let us say it as X.
- Its output is Z which is much smaller compared to X

Convolution Autoencoders:
uses convolution operator to learn to encode the input in a set of simple singles and then try to reconstruct the input in a set of simple signals and then try to reconstruct the input from them



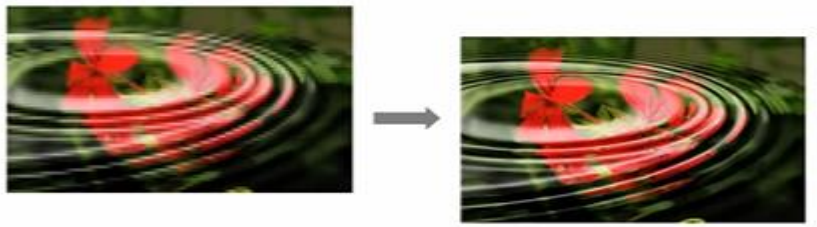
Uses of Convolution autoencoders

- 1. Image reconstruction



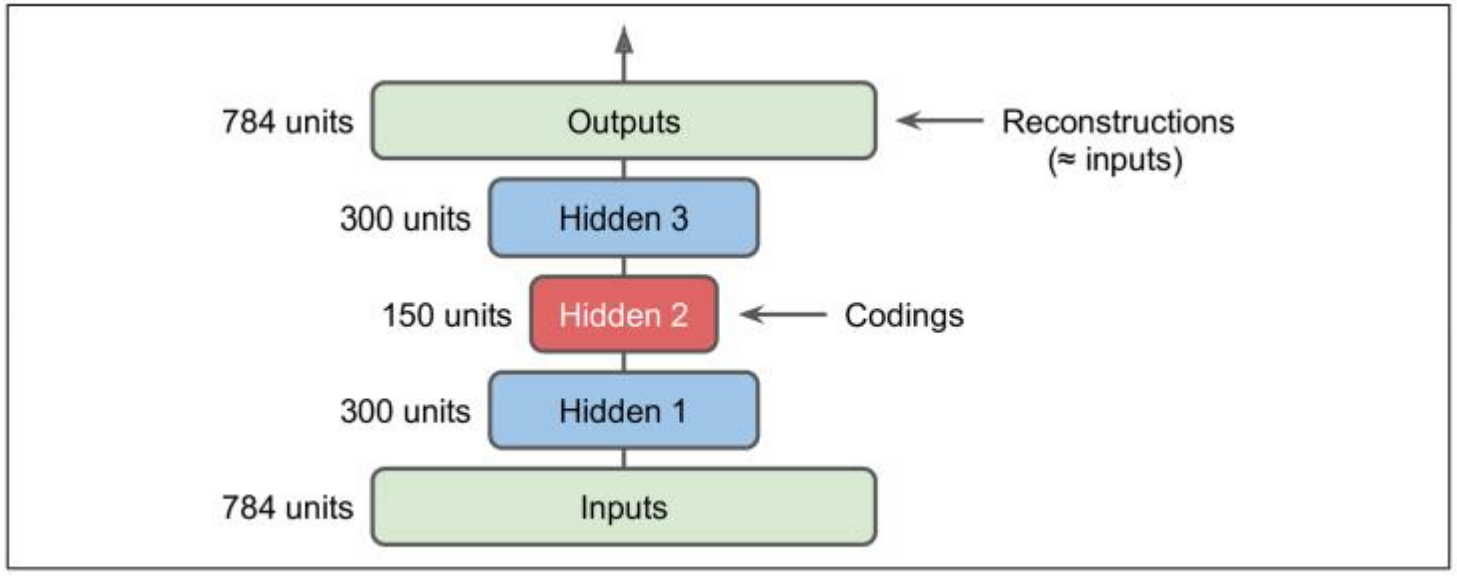
- 2. Image Colorization

- 3. Advanced application like generating high resolution image



Stacked Autoencoders

- Just like other neural networks we have discussed, autoencoders can have multiple hidden layers.
- In this case they are called stacked autoencoders (or deep autoencoders).
- Adding more layers helps the autoencoder learn more complex coding's.



Unsupervised Pretraining using Stacked Autoencoders

- Tackling a complex supervised task when do not have a lot of labeled training data, one solution is to find a neural network that performs a similar task, and then reuse its lower layers.

- This makes it possible to train a high-performance model using only little training data because your neural network won’t have to learn all the low-level features; it will just reuse the feature detectors learned by the existing net.

unsupervised Pretraining Using Stacked Autoencoders

- Similarly, if you have a large dataset but most of it is unlabeled, you can first train a stacked autoencoder using all the data, then reuse the lower layers to create a neural network for your actual task, and train it using the labeled data.

Unsupervised pretraining using autoencoders

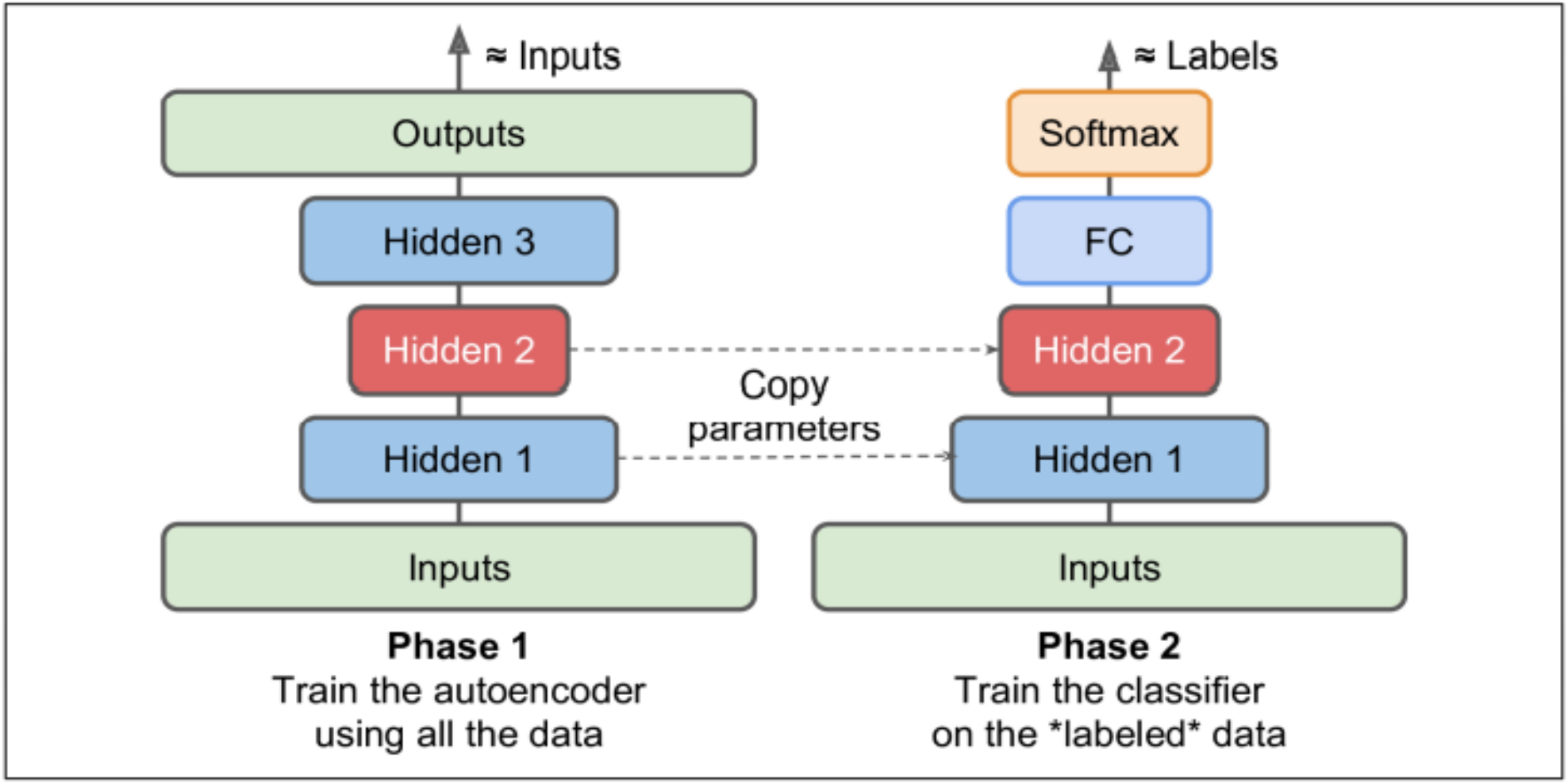


Figure 15-8. Unsupervised pretraining using autoencoders

Denoising Autoencoders

- Another way to force the autoencoder to learn useful features is to add noise to its inputs, training it to recover the original, noise-free inputs. This prevents the autoencoder from trivially copying its inputs to its outputs, so it ends up having to find patterns in the data.

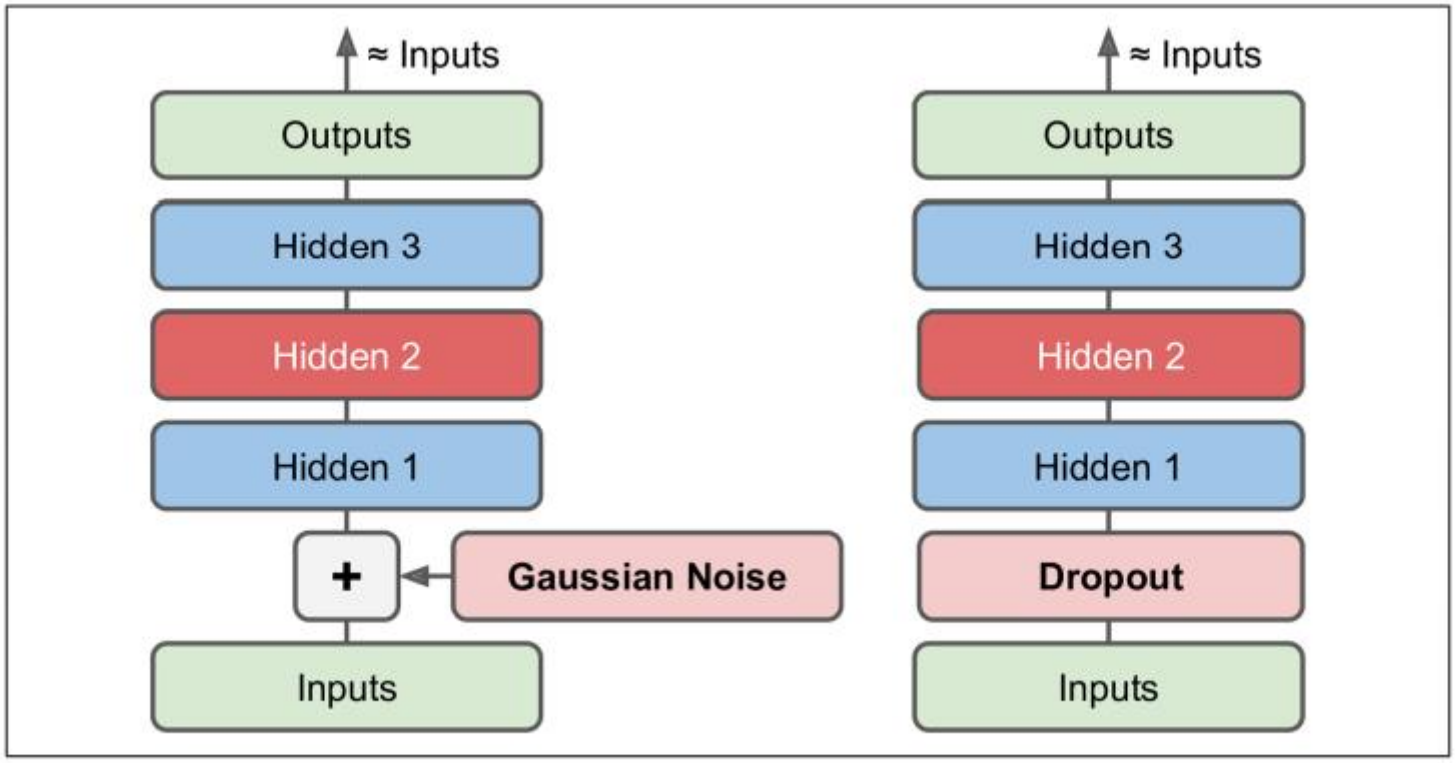


Figure 15-9. Denoising autoencoders, with Gaussian noise (left) or dropout (right)

Variational Autoencoders

- Another important category of autoencoders variational autoencoders.

- They are probabilistic autoencoders, meaning that their outputs are partly determined by chance, even after training

- Most importantly, they are generative autoencoders, meaning that they can generate new instances that look like they were sampled from the training set.

Reinforcement Learning

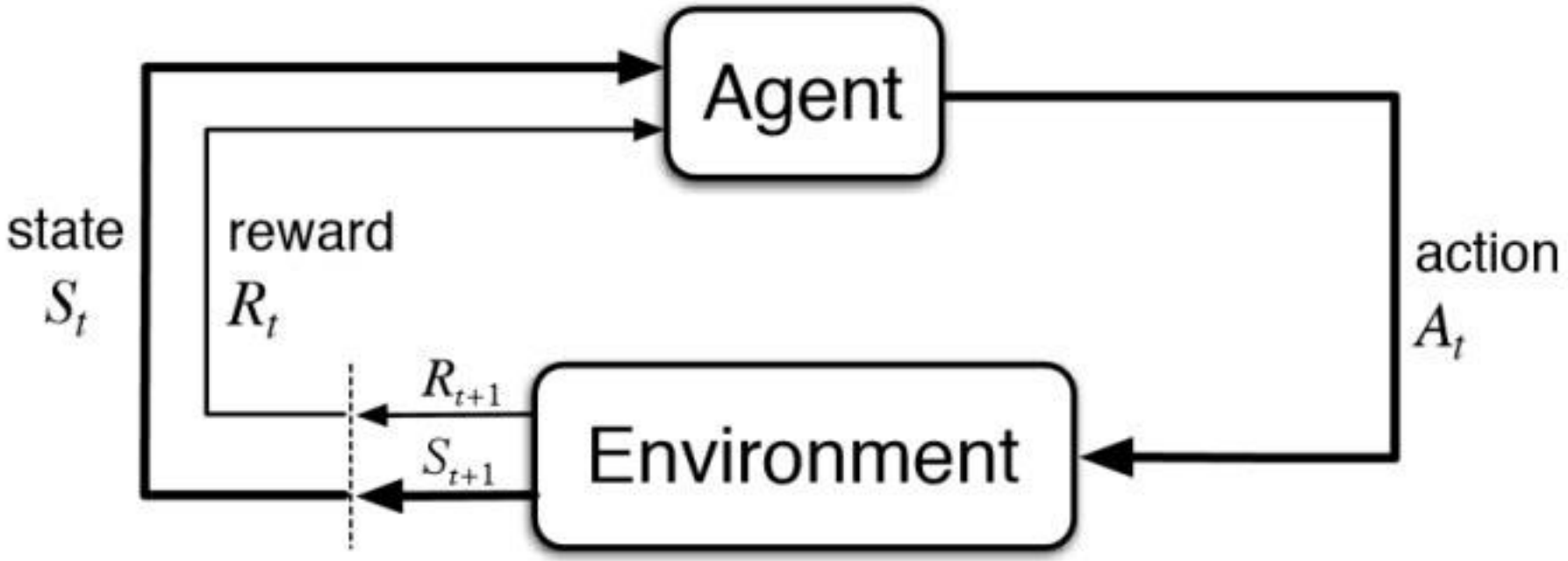
- What is reinforcement Learning?

- Reinforcement learning is **a machine learning training method based on rewarding desired behaviors and/or punishing undesired ones**.

- In general, a reinforcement learning agent is able to perceive and interpret its environment, take actions and learn through trial and error.

- Reinforcement Learning (RL) is **the science of decision making**. It is about learning the optimal behavior in an environment to obtain maximum reward.

Reinforcement Learning Architecture



Some of the important terms used

- **Agent:** It is an assumed entity which performs actions in an environment to gain some reward.
- **Environment (e):** A scenario that an agent has to face.
- **Reward (R):** An immediate return given to an agent when he or she performs specific action or task. (can be award or punishment)
- **State (s):** State refers to the current situation returned by the environment.

Example:

- The agent can be the program controlling a walking robot.
- In this case, the environment is the real world, the agent observes the environment through a set of sensors such as cameras and touch sensors.
- Its actions consist of sending signals to activate motors.
- It may be programmed to get positive rewards whenever it approaches the target destination, and negative rewards whenever it wastes time, goes in the wrong direction, or falls down.

Policy Search

- The algorithm used by the software agent to determine its actions is called its policy.

- For example, the policy could be a neural network taking observations as inputs and outputting the action to take (see Figure 16-2).

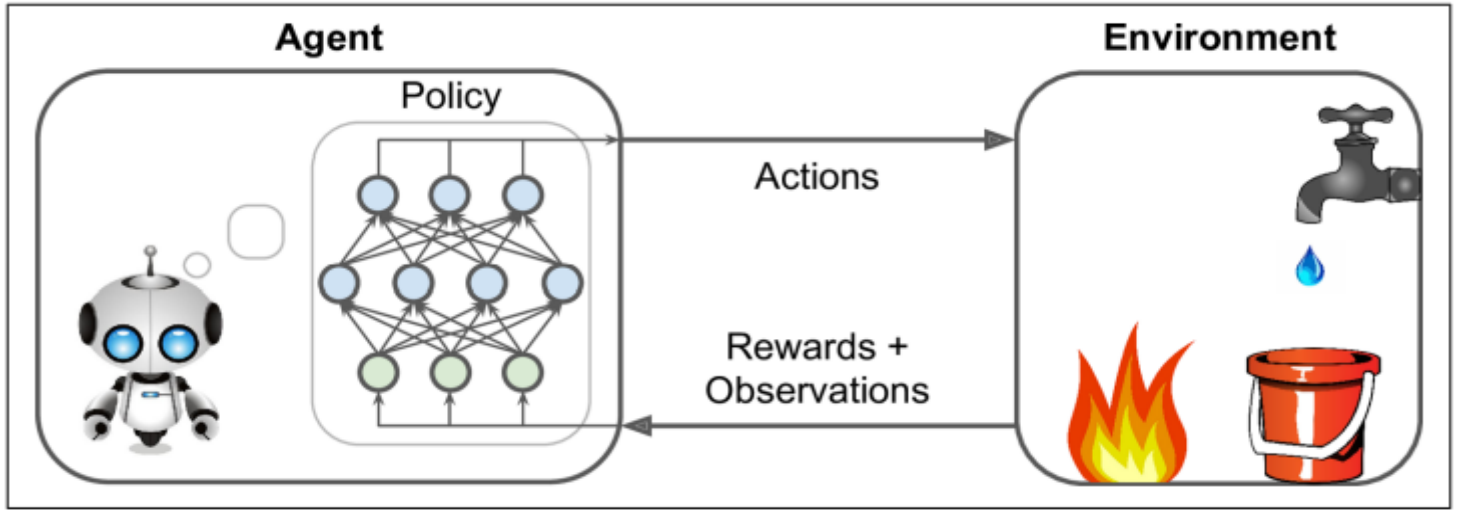


Figure 16-2. Reinforcement Learning using a neural network policy

Neural Network Policy

- Neural network will take an observation as input, and it will output the action to be executed.

- More precisely, it will estimate a probability for each action, and then we will select an action randomly according to the estimated probabilities.

Neural Network Policy

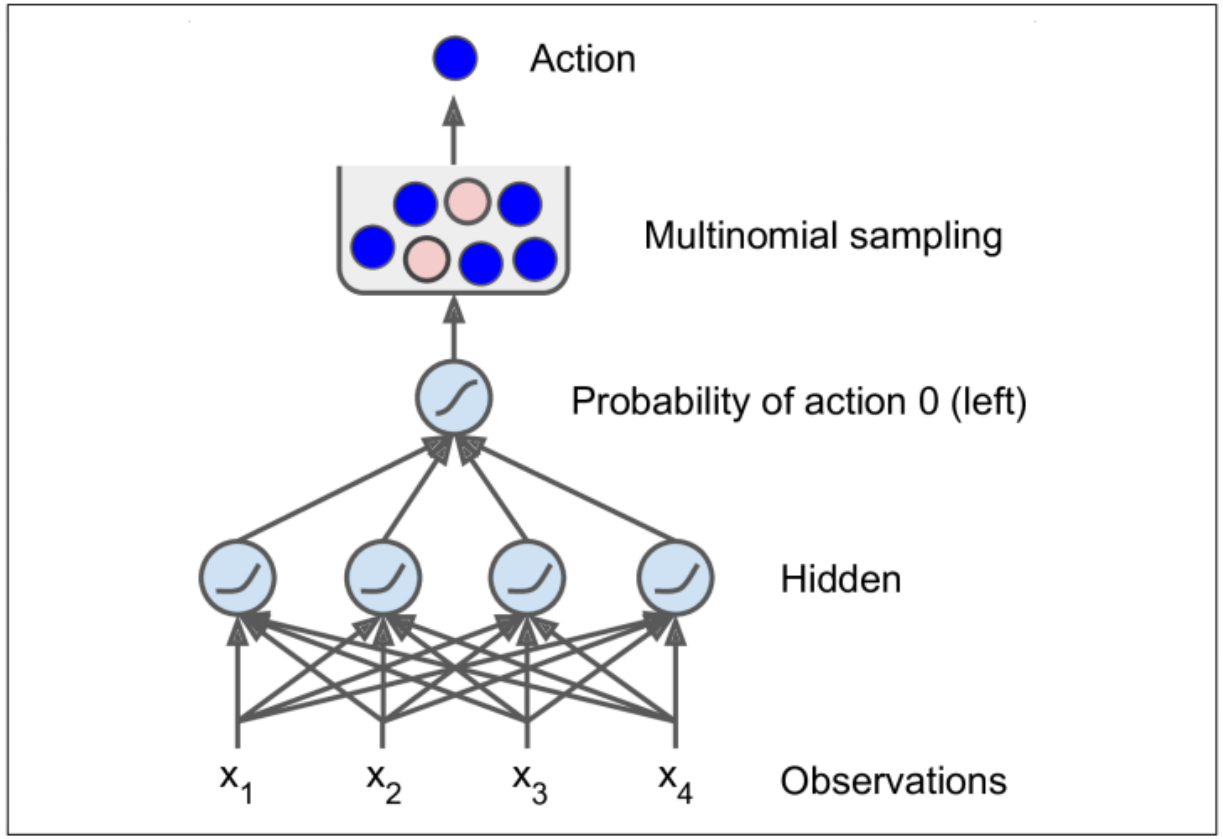


Figure 16-5. Neural network policy

Evaluating Actions:

- If we knew what the best action was at each step, we could train the neural network as usual, by minimizing the cross entropy between the estimated probability and the target probability.

- It would just be regular supervised learning.

- However, in Reinforcement Learning the only guidance the agent gets is through rewards, and rewards are typically sparse and delayed.

- To tackle this problem, a common strategy is to evaluate an action based on the sum of all the rewards that come after it, usually applying a discount rate r at each step.

- if an agent decides to go right three times in a row and gets +10 reward after the first step, 0 after the second step, and finally –50 after the third step, then assuming we use a discount rate r = 0.8, the first action will have a total score of $10 + r \times 0 + r^2 \times (-50) = -22$.

Policy Gradients (PG)

- PG algorithms optimize the parameters of a policy by following the grad
- First, let the neural network policy play the game several times and at each step compute the gradients that would make the chosen action even more likely, but don't apply these gradients yet.

- Once you have run several episodes, compute each action's score.

- If an action's score is positive, it means that the action was good and you want to apply the gradients computed earlier to make the action even more likely to be chosen in the future.

- However, if the score is negative, it means the action was bad and you want to apply the opposite gradients to make this action slightly less likely in the future. The solution is simply to multiply each gradient vector by the corresponding action's score.

- Finally, compute the mean of all the resulting gradient vectors, and use it to per- form a Gradient Descent step

Markov Decision Process (MDP)

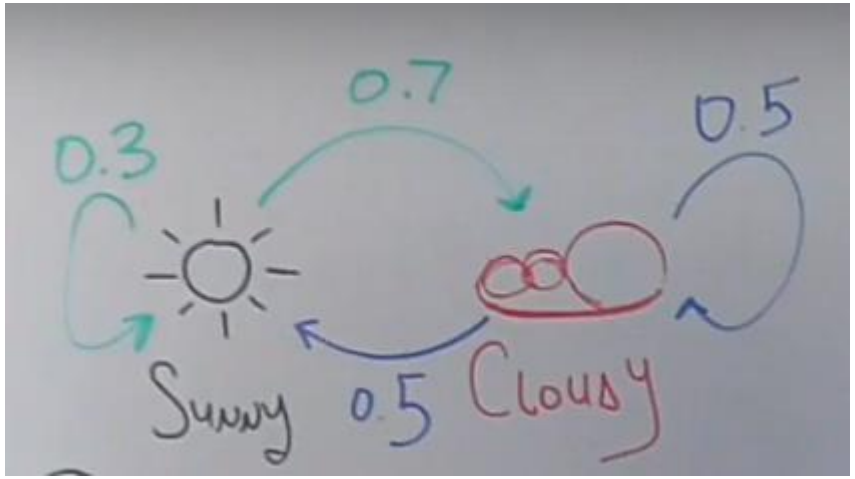
- If the information provided by the state is sufficient to determine the future states of the environment given any action, then the state is deemed to have the Markov property (or in some literature, the state is deemed as Markovian).

- This comes from the fact that the environments we deal with while doing Reinforcement Learning are modeled as Markov Decision Processes.

Markovian Decision Process continued..

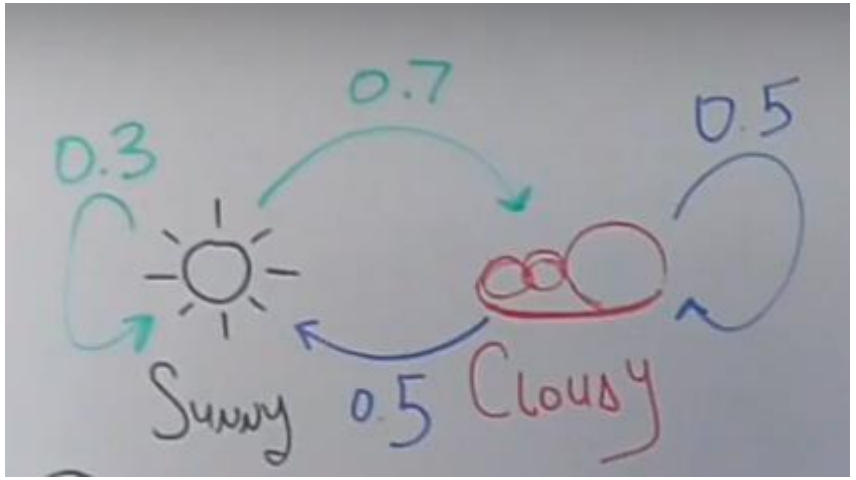
- Reinforcement Learning has:
  - Agent
  - Actions
  - Environment
  - Rewards

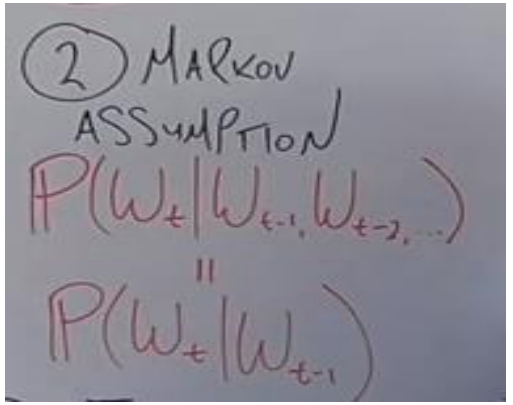
  - Consider the following to explain MDP



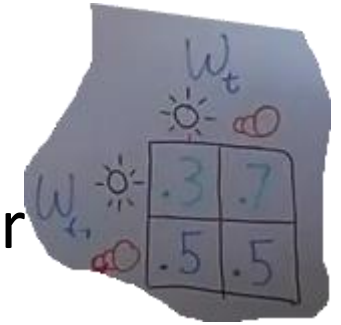
MDP

- Consider the state of the climate can be Sunny or Cloudy.





- The state change is shown in the diagram above
- The state space has only two states ..Ref (1)
- Markov assumption finally says that the current state depends only on the previous state, not on any other state earlier to it.
- In Reinforcement Learning, state change decision
- Depends only on the previous state not on any other earlier



TDL

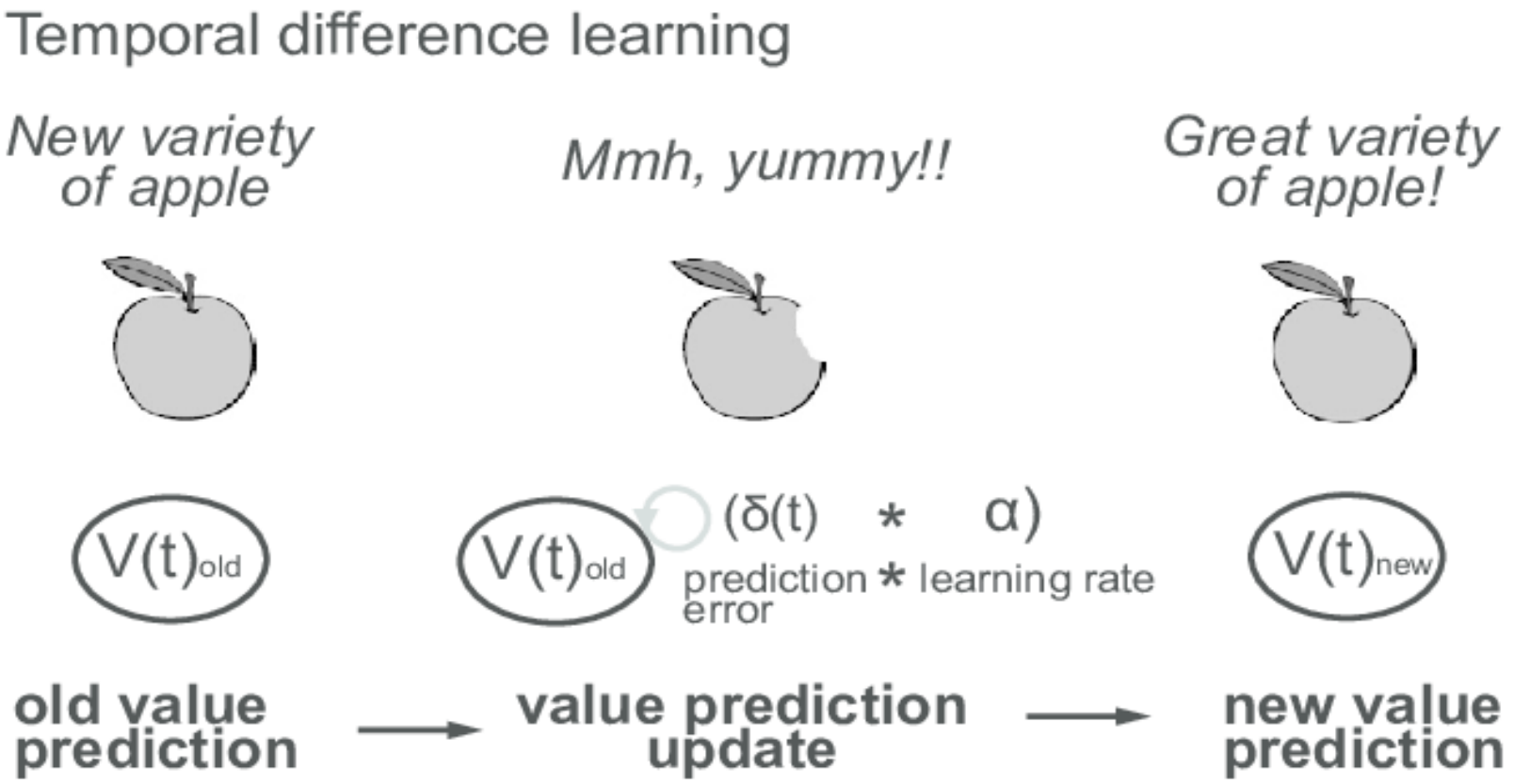
- TDL – Temporal Difference Learning

- Temporal difference (TD) learning is **an approach to learning how to predict a quantity that depends on future values of a given signal**. The name TD derives from its use of changes, or differences, in predictions over successive time steps to drive the learning process.

- Temporal Difference Learning is an unsupervised learning technique that is very commonly used in reinforcement learning for the purpose of predicting the total reward expected over the future.

TDL example

- TDL estimates the quantity, which depends on the feature signals



Q-Learning (Quality Learning)

- Q-learning is a variant of reinforcement learning algorithm that seeks to find the best action to take given the current state.
- It uses Monte Carlo Policy.
- It observes reward for all the steps in the episode, where as TDL observes only one step.

End of Unit - 5

End of the Course
Deep Learning Architecture

Wishing You
All the best!

Dr.Srinath.S
Associate Professor and Head
Dept. of CS&E
SJCE, JSS S&TU
Mysuru- 570006