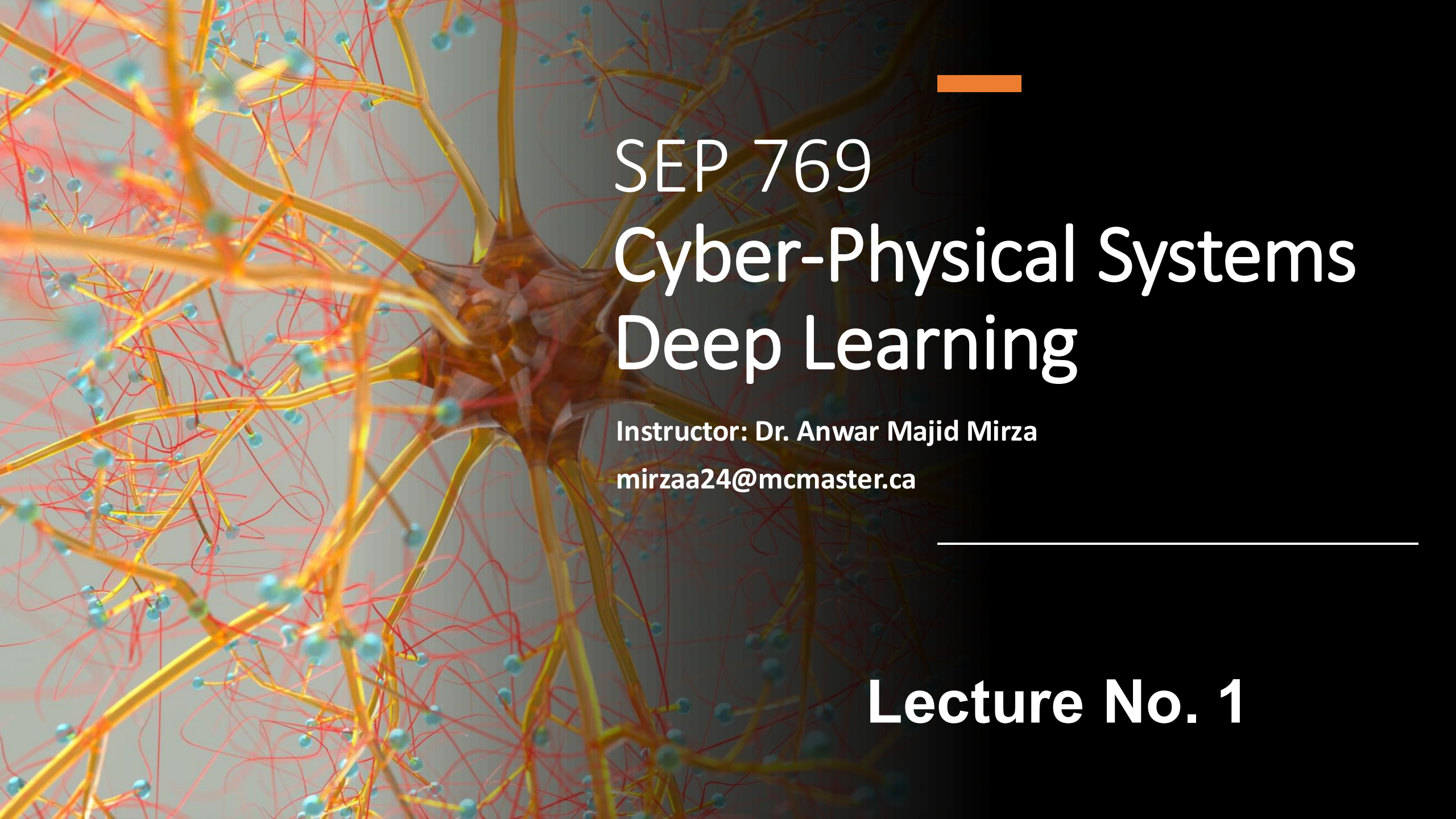


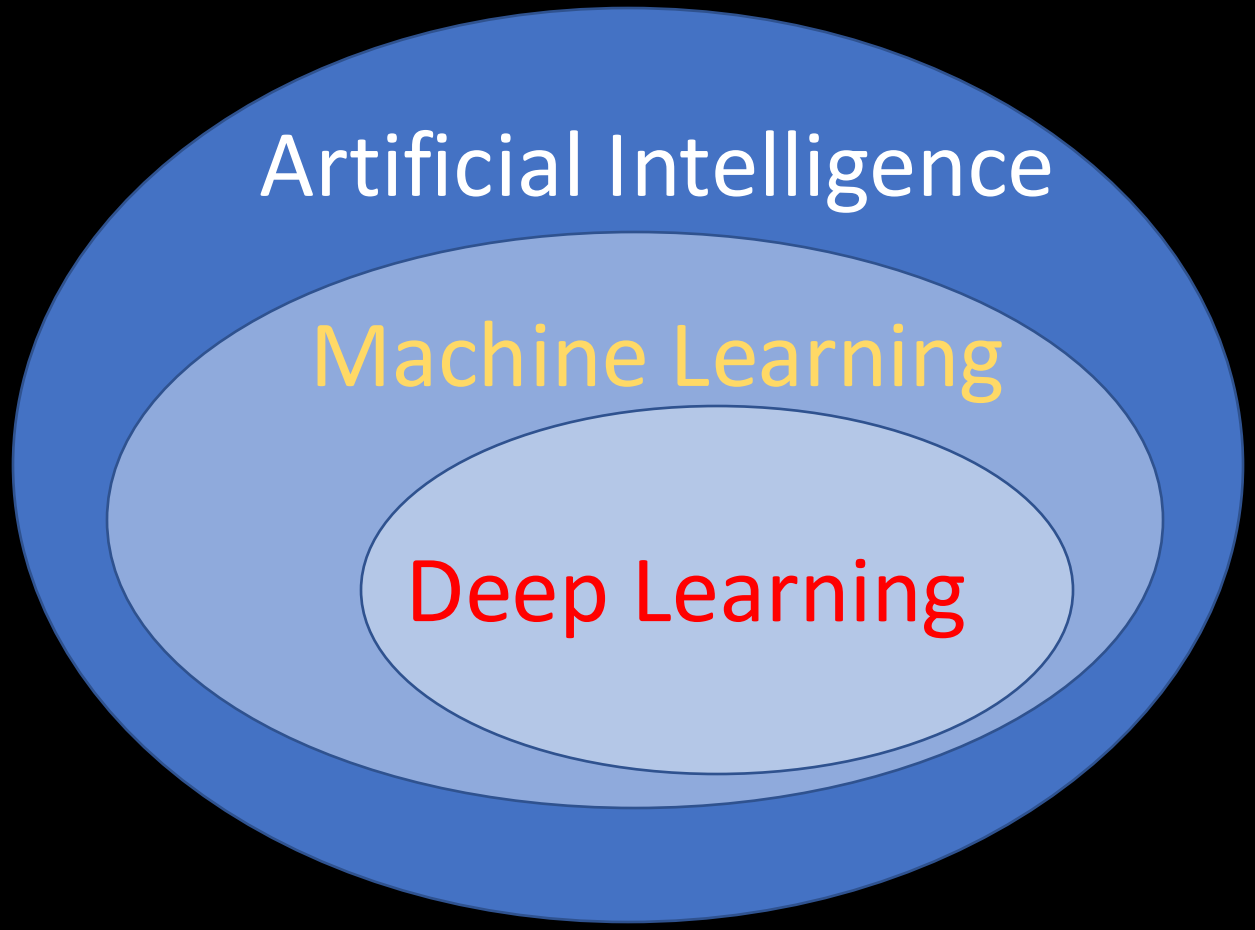
SEP 769
Cyber-Physical Systems Deep Learning

Instructor: Dr. Anwar Majid Mirza

mirzaa24@mcmaster.ca

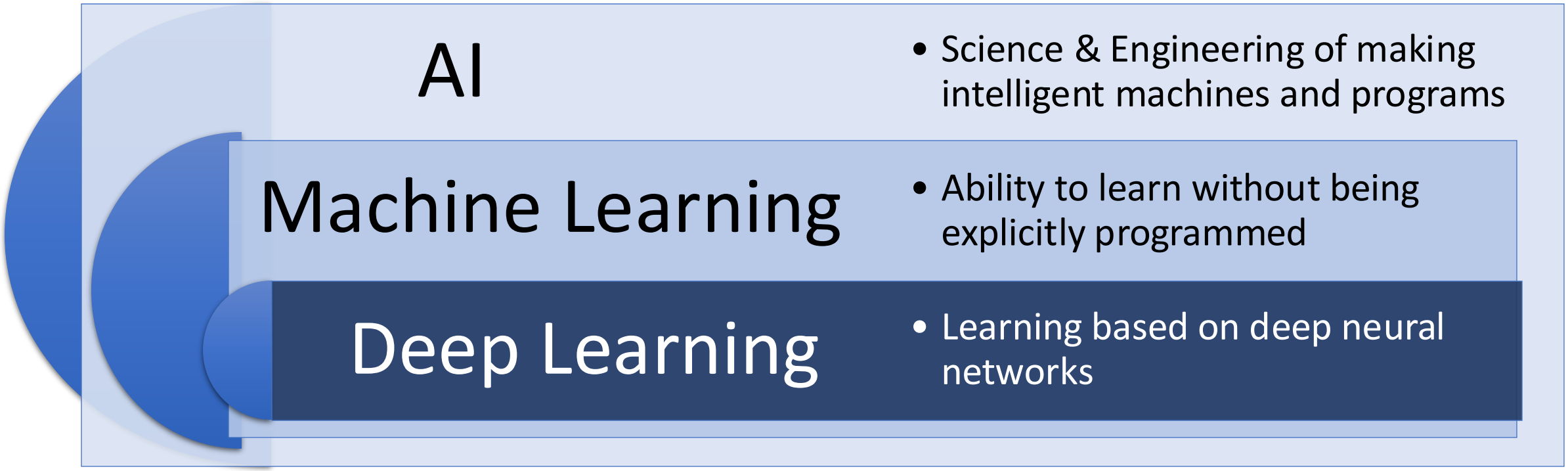
Lecture No. 1

What is Deep Leaning?



- Deep learning is a subset of machine learning (which in turn is a part of the field of Artificial Intelligence).
- Uses Artificial Neural Networks with multiple layers to model and solve complex problems

AI, Machine Learning and Deep Learning



Building artificial systems that process, perceive and reason about data.

A Brief History of AI / Machine Learning / Deep Learning

- **1943:** Warren **McCulloch** and Walter **Pitts** propose the **first model of artificial neural networks** (ANNs), which were inspired by the structure of the human brain.

- **1957:** Frank **Rosenblatt** introduces the **perceptron**, a single-layer neural network capable of learning simple linear patterns.

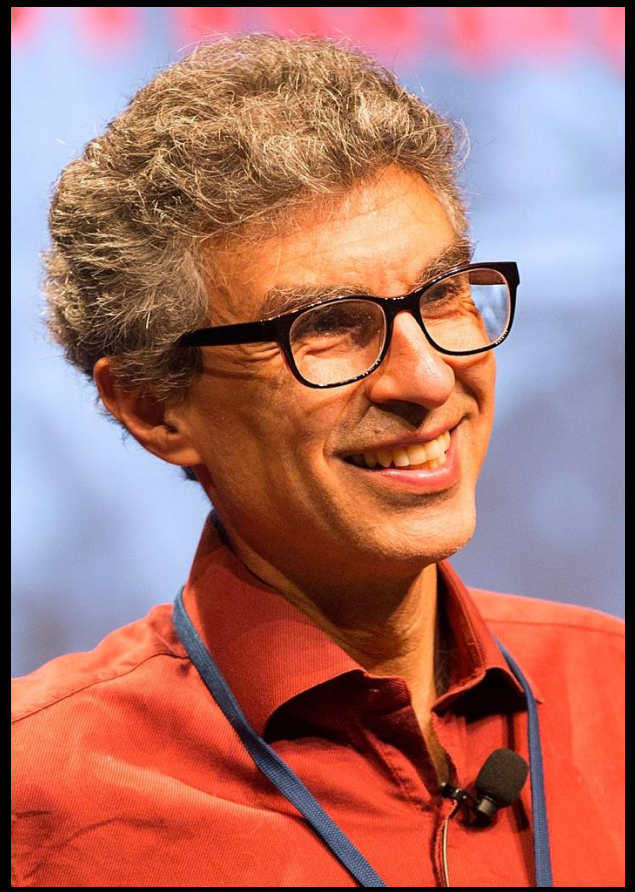
- **1969: Marvin Minsky** and Seymour Papert publish "**Perceptrons**," a book that shows the limitations of the perceptron and argues that Neural Networks are fundamentally limited in what they can learn.

- **1986:** Geoffrey **Hinton**, David **Rumelhart**, and Ronald Williams publish a paper on the **backpropagation algorithm**, which enables efficient training of multi-layer neural networks.

- **1990s: Neural Networks fall out of favor** due to difficulties in training them and the rise of other Machine Learning techniques such as Support Vector Machines (SVMs). **AI Winter**

- **2006: Hinton**, along with Yoshua **Bengio** and Yann **LeCun**, pioneers the use of Deep Neural Networks for speech recognition, achieving a significant reduction in error rates.
  (This trio is known as *Godfathers of Deep Leaning*)

A Brief History of AI / Machine Learning / Deep Learning



Yoshua Bengio

Geoffrey Hinton

Yann LeCun

A Brief History of AI / Machine Learning / Deep Learning …

- **2012:** Alex Krizhevsky, Ilya Sutskever, and **Hinton** use Deep *Convolutional Neural Networks* (CNNs) to win the *ImageNet* Large Scale Visual Recognition Challenge, dramatically improving image recognition accuracy. *LeNet*, *AlexNet*, and *Microsoft's Very Deep Net*.

- **2014: Google** acquires **DeepMind**, an AI research company founded by Demis Hassabis, Shane Legg, and Mustafa Suleyman, which goes on to make significant breakthroughs in *Reinforcement Learning*.

- **2015:** Andrej Karpathy uses *Recurrent Neural Networks* to generate image captions, showing the potential of Deep Learning for Natural Language Processing (NLP).

- **2016: AlphaGo**, a program developed by DeepMind, *defeats the world champion in the game of* ***Go***, demonstrating the power of Deep Learning for complex decision-making.
- 2018: **Hinton**, **Bengio**, and **LeCun** receive the **Turing Award** (referred to as *the Nobel Prize of Computing*), the highest honor in computer science, for their contributions to Deep Learning.

- **2024**: **John J. Hopfield** and **Geoffrey Hinton** were awarded the **Nobel Prize** in Physics, "for foundational discoveries and inventions that enable machine learning with artificial neural networks".

THE NOBEL PRIZE IN PHYSICS 2024

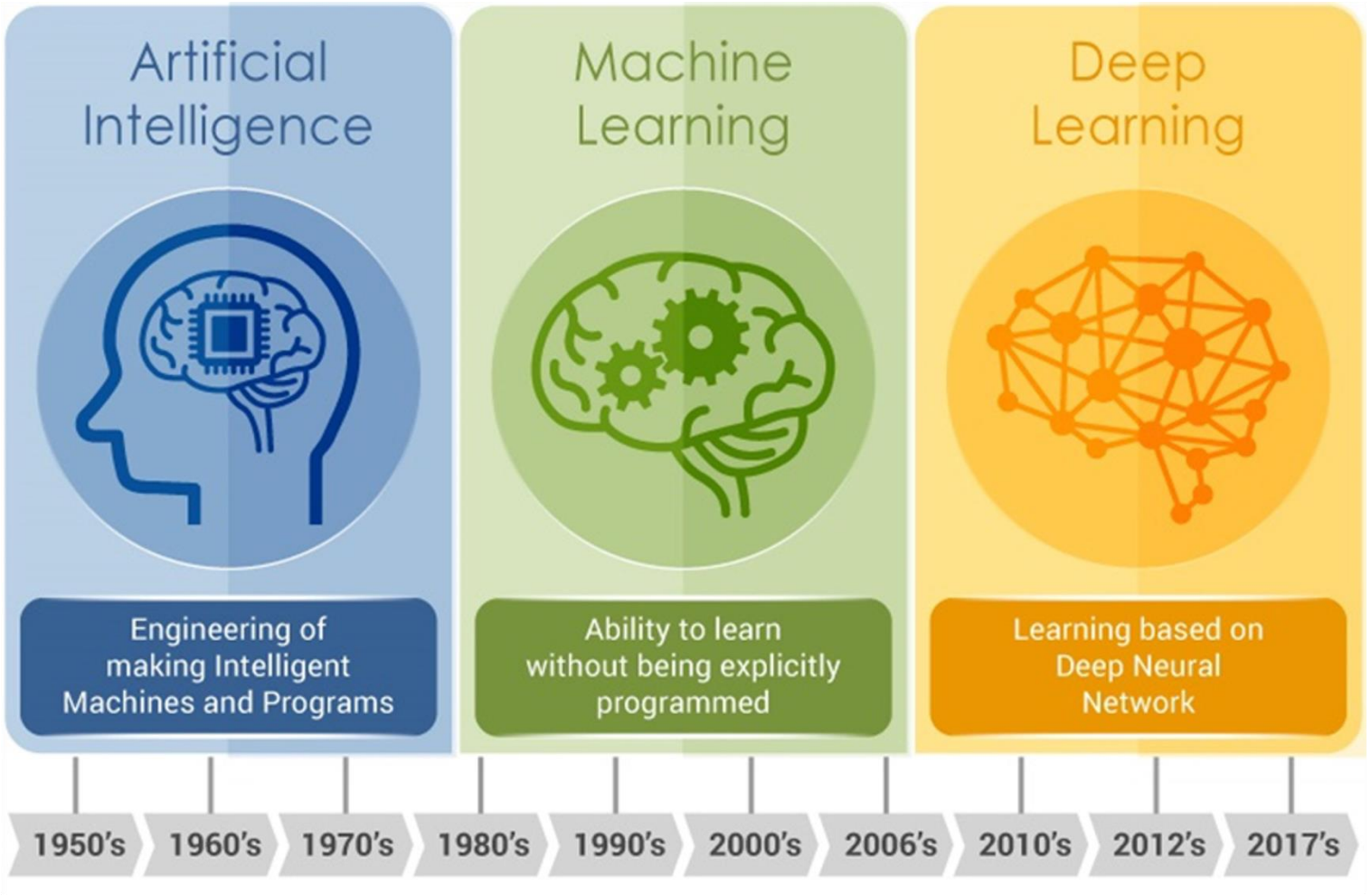
Illustrations: Niklas Elmehed

John J. Hopfield Geoffrey E. Hinton

"for foundational discoveries and inventions
that enable machine learning
with artificial neural networks"

THE ROYAL SWEDISH ACADEMY OF SCIENCES

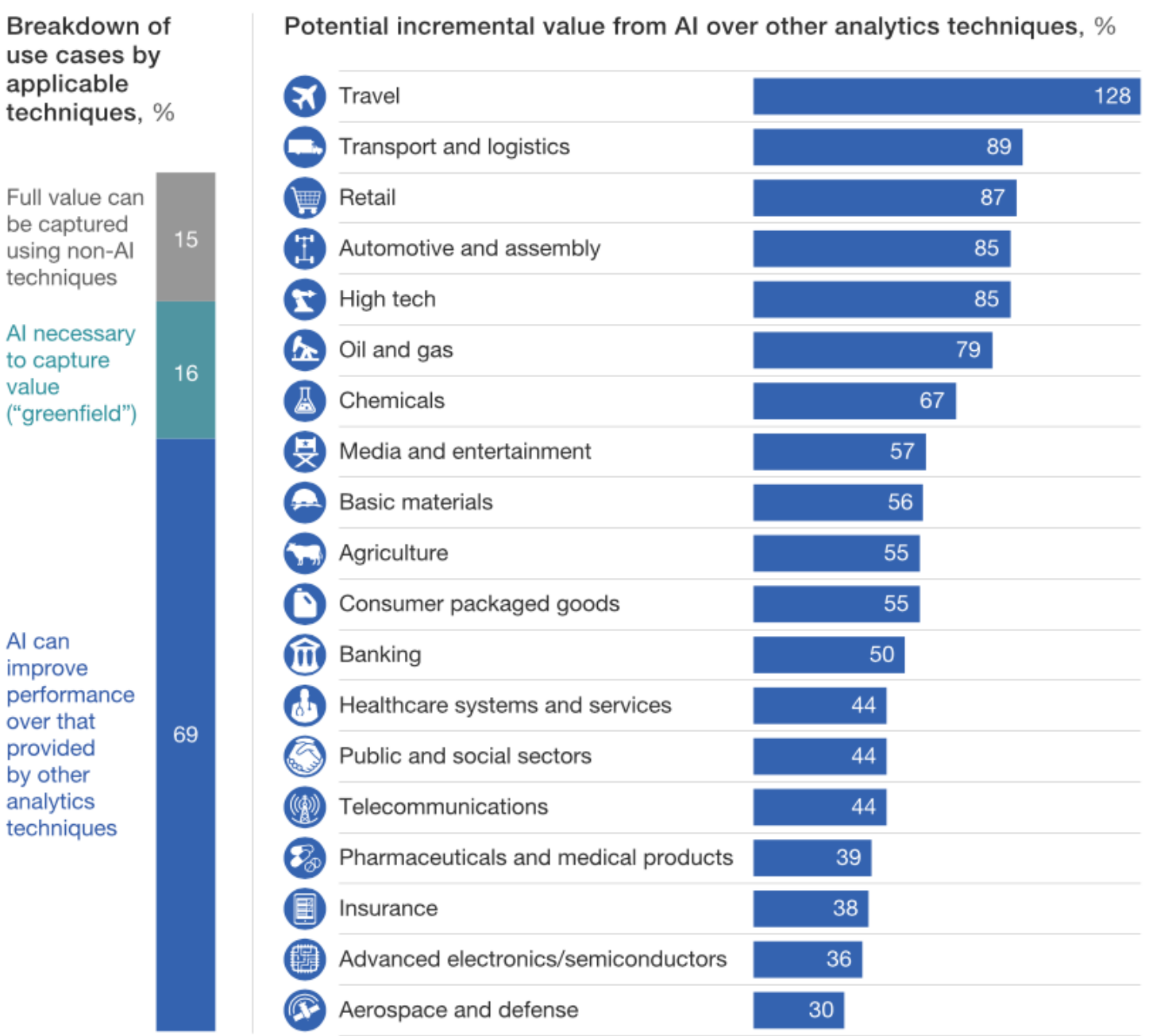
Brief History



AI versus Other Analytics Techniques

McKinsey 2018

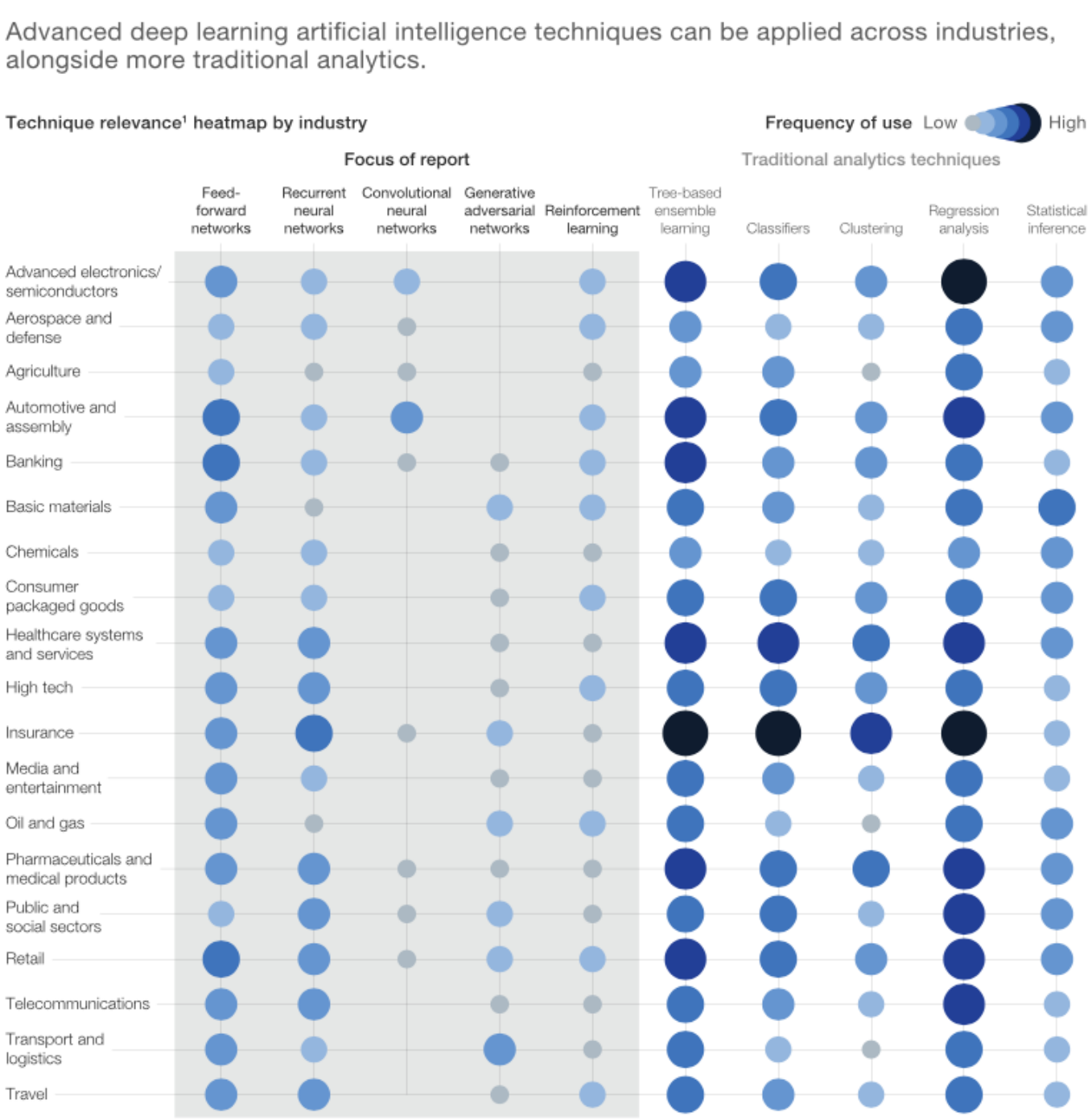
Click here for the link.



Insights from 400 use cases across 19 industries

McKinsey
2018

Click here for the link.



What Are The Most Important Areas of Deep Learning?

1. Neural Networks and Deep Leaning Fundamentals
2. Feedforward Neural Networks (FNNs)
3. Recurrent Neural Networks (RNNs)
4. Convolutional Neural Networks (CNNs)
5. Reinforcement Learning (RL)
6. Deep Learning Frameworks
7. Applications of Deep Learning
8. Advanced Deep Learning Topics

What Are The Most Important Things to Learn / Study in Becoming an Expert of Deep Learning?

1. Mathematics
2. Programming
3. Neural Networks
4. Model Design and Evaluation
5. Data Preparation and Presentation
6. Research and Innovation
7. Optimization Techniques
8. Computer Vision
9. Natural Language Processing
10. Transfer Learning
11. Collaborative and Open-Source Tools
12. Communication Skills

Remaining Lecture …

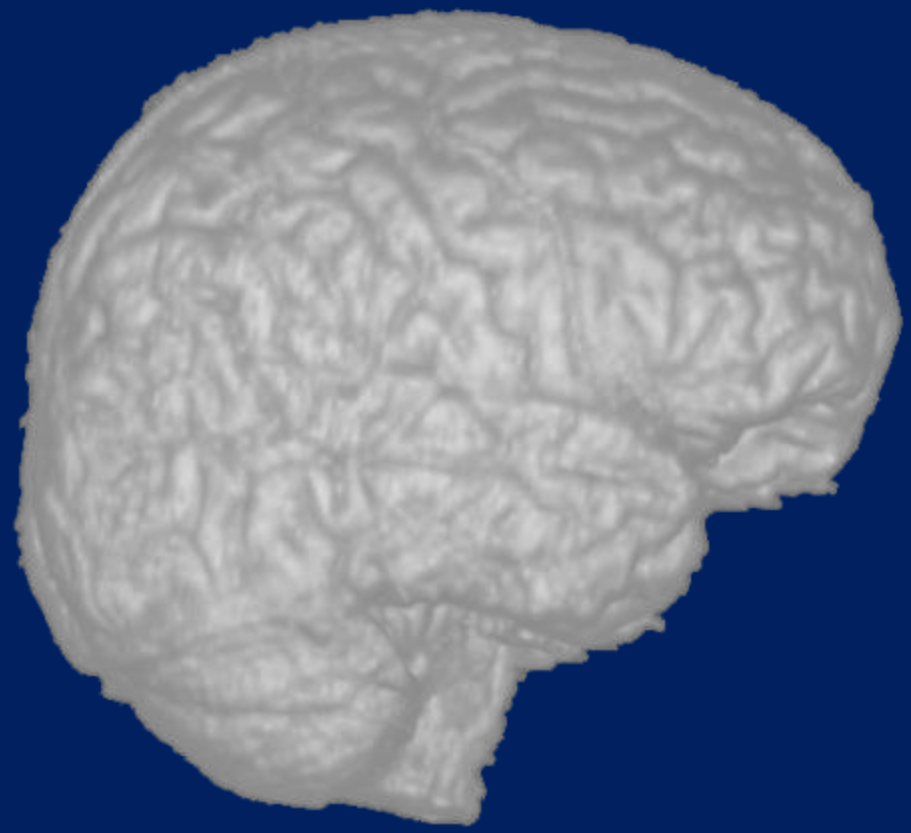
- Introduction
- What is a Neural Network?
- Mathematical Model of a Neuron
- Network Architectures
- Training
- Commonly Used Activation Functions
- Typical Problem Areas
- Perceptron: Simple Perceptron for Pattern Classification
- Perceptron: Algorithm
- Perceptron: Example
- A Perceptron to classify letters from different fonts:
  One Output Class
- A Perceptron to classify letters from different fonts:
  Several output classes
- Modified Perceptron Algorithm
- Problem with Perceptron Learning
- Derivation of Delta-Rule
  a) Single output unit case
  b) Multiple output units' case
- Application of Delta Rule:
  1. Adaline

Tools: Practice Session

Download Jupyter Notebook "adaline.ipynb" from A2L course website and run it on Anaconda / CoLab

Welcome to the World of Artificial Neural Networks

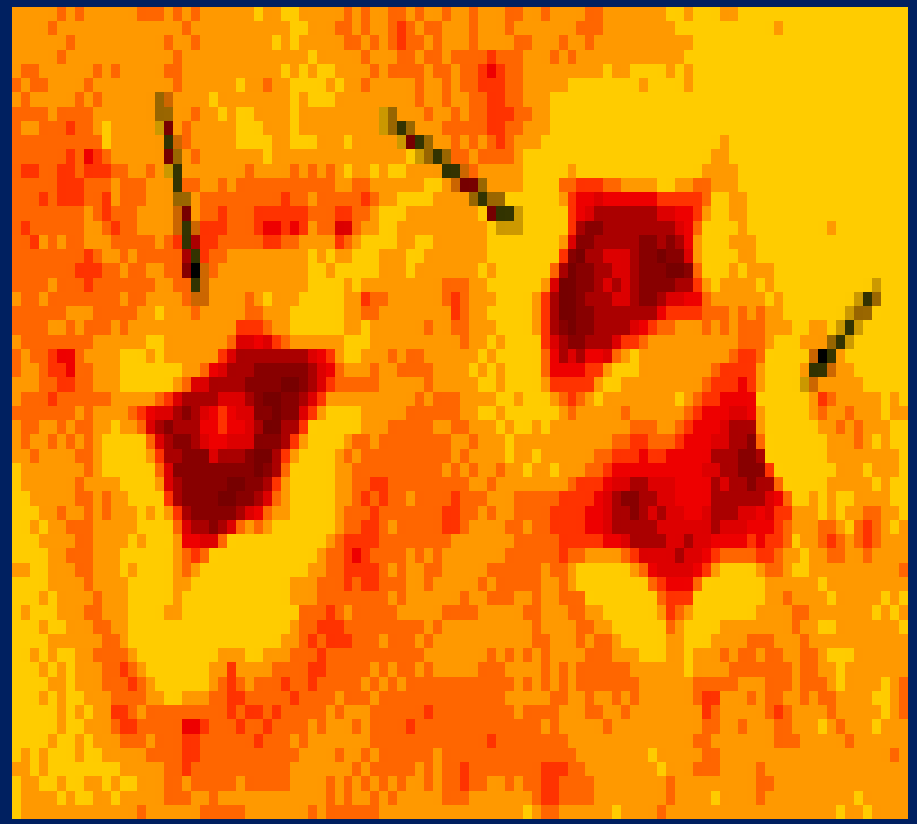
- Brain
- A marvelous piece of architecture and design.
- In association with a nervous system, it controls the life patterns, communications, interactions, growth and development of hundreds of billion of life forms.



Some Statistics

- There are about 10^{10} to 10^{14} nerve cells (called neurons) in an adult human brain.
- Neurons are highly connected with each other. Each nerve cell is connected to hundreds of thousands of other nerve cells.
- Passage of information between neurons is slow (in comparison to transistors in an IC). It takes place in the form of electrochemical signals between two neurons in milliseconds.
- Energy consumption per neuron is low (approximately 10^{-6} Watts).

How do the actual neurons look like?



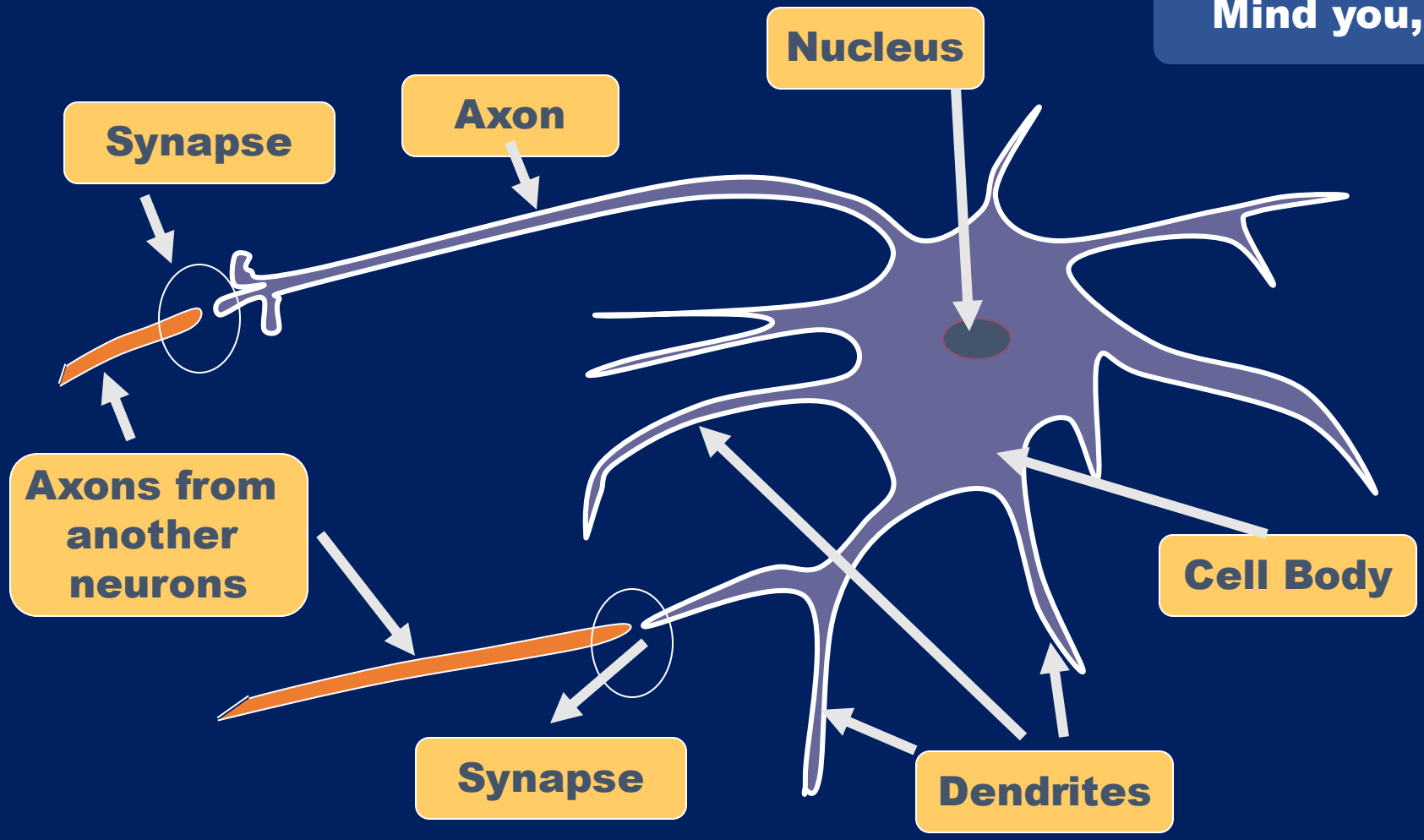
Look more like some blobs of ink... aren't they!

Taking a more closer look reveals that there is a large collection of different molecules, working together coherently, in an organized manner.

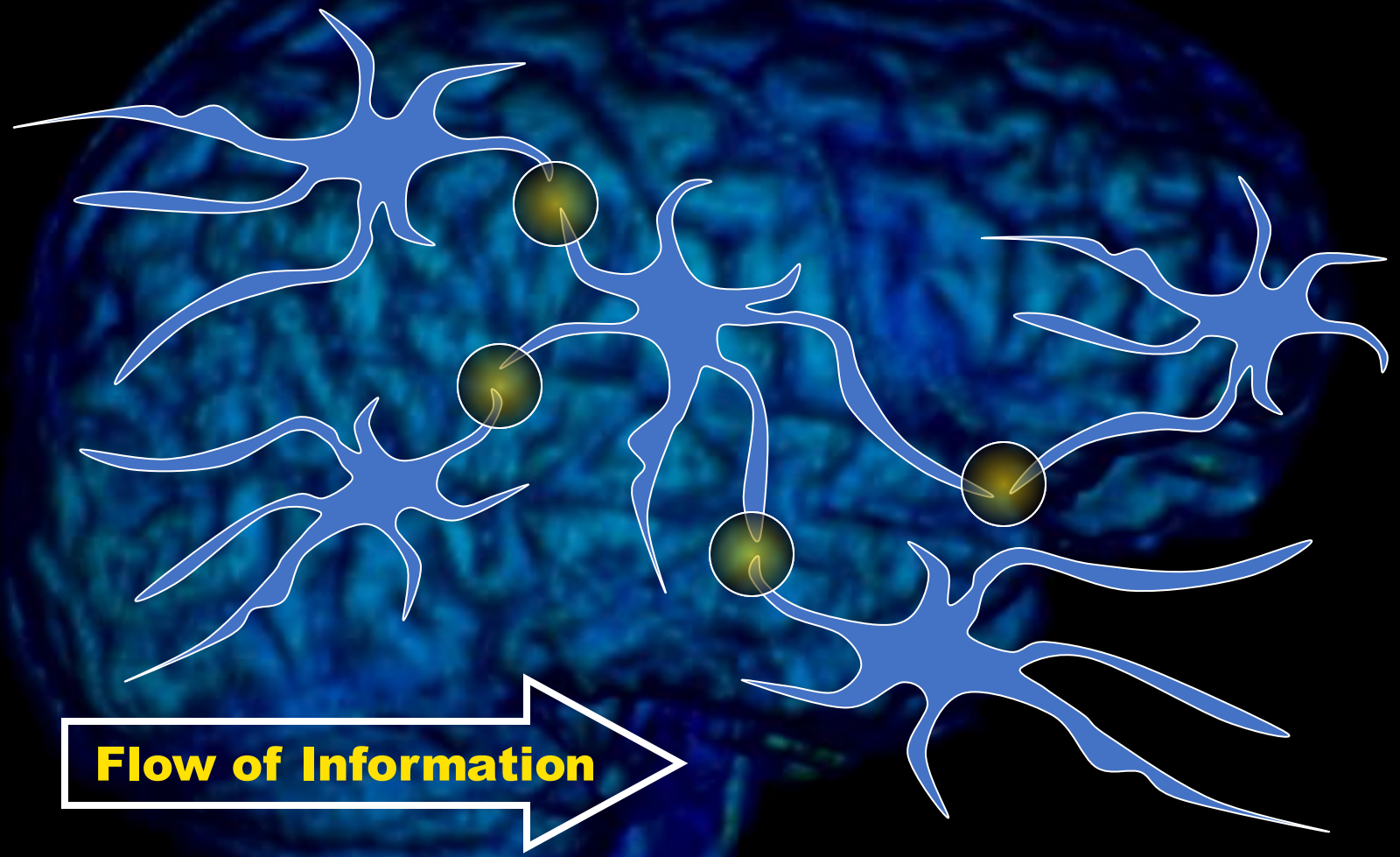
Put together, they form the best information processing system in the known universe.

Salient Features of a Neuron

Mind you, a neuron is a 3Dentity!

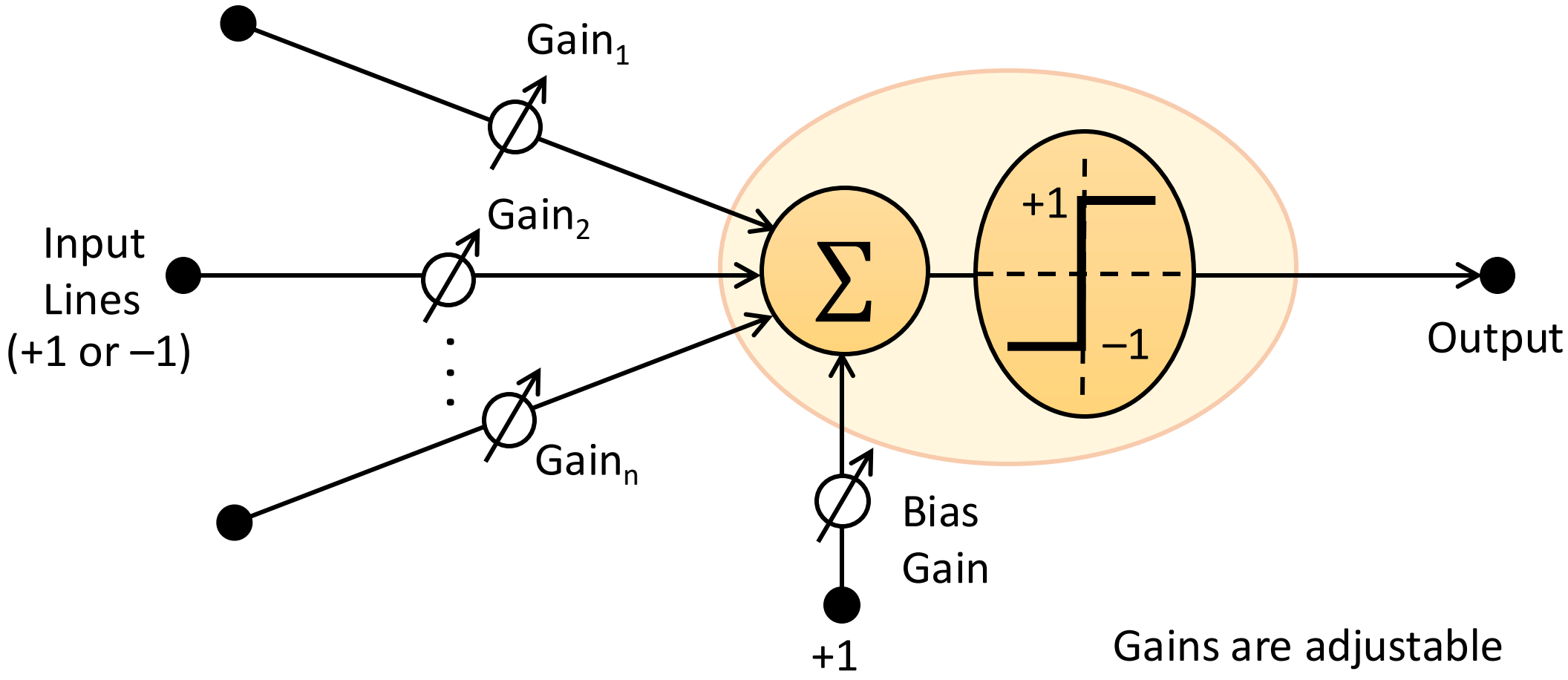


Information Processing in a Neural Network



A few Neurons and their synaptic junctions

Mathematical Model of A Typical Neuron



Basics of an Artificial Neural Network

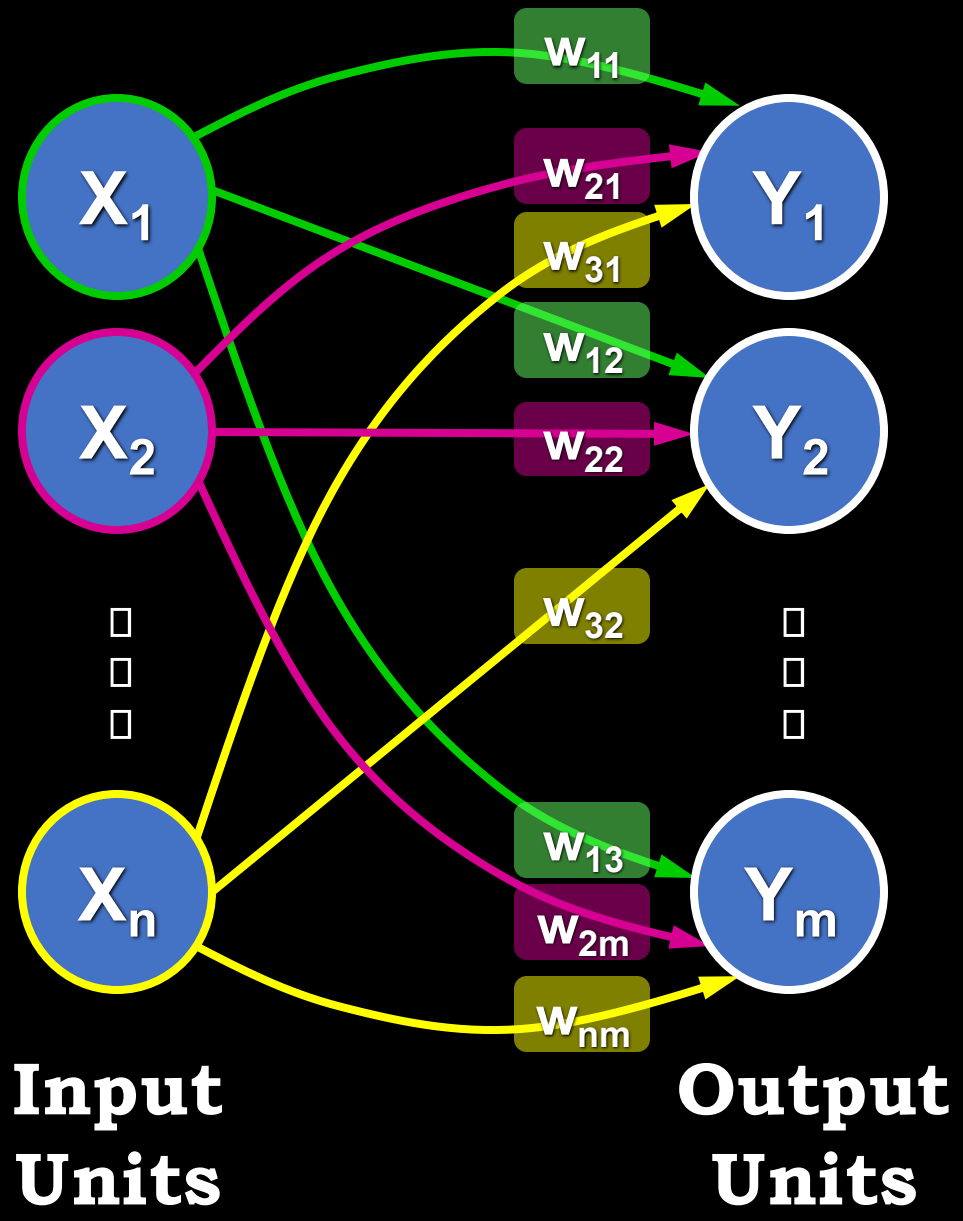
- An artificial neural network is an information processing system that has certain performance characteristics in common with biological neural networks.
- An ANN can be characterized by:
  1. Architecture: The pattern of connections between different neurons.
  2. Training or Learning Algorithms: The method of determining weights on the connections.
  3. Activation Function: The nature of function used by a neuron to become activated.

Architecture

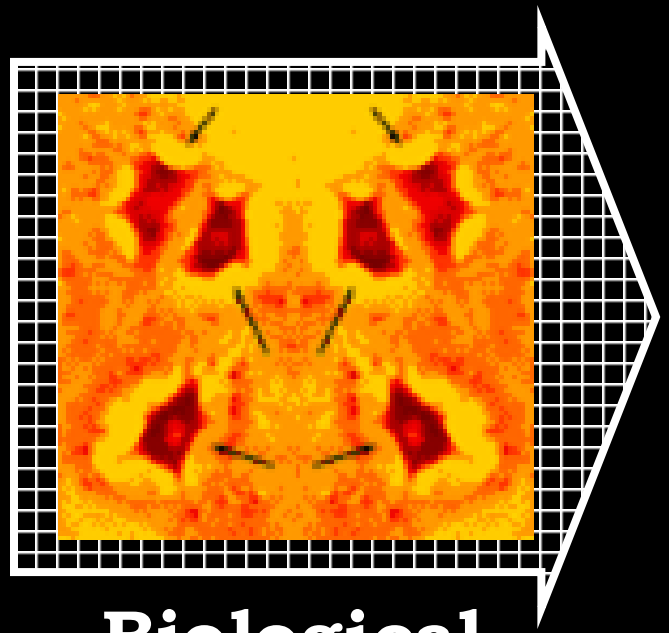
- There are two basic categories:

1. Feed-forward Neural Networks
   - These are the nets in which the signals flow from the input units to the output units, in a forward direction.
   - They are further classified as:
     1. Single Layer Nets
     2. Multi-layer Nets
2. Recurrent Neural Networks
   - These are the nets in which the signals can flow in both directions from the input to the output or vice versa.

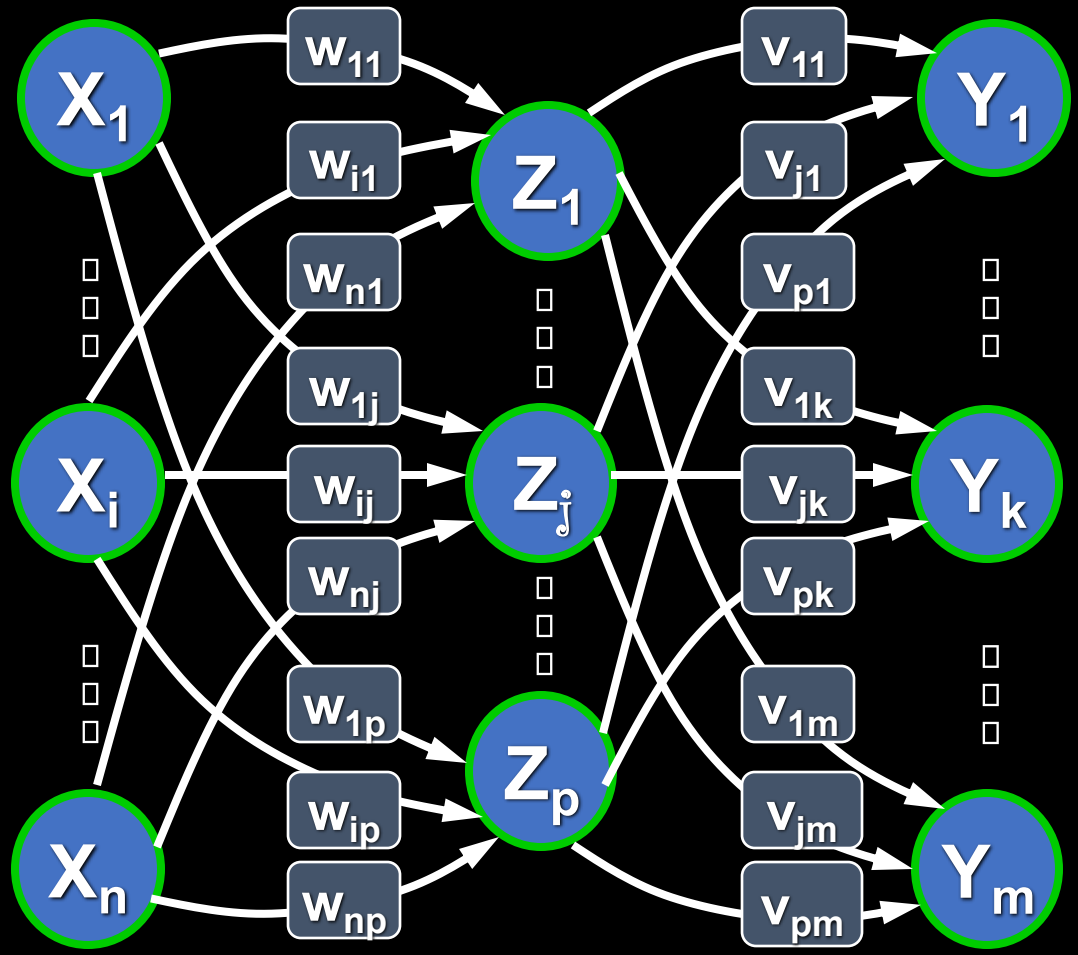
Model 1: A Single Layer Net



Model 2: A Multi-Layer Net

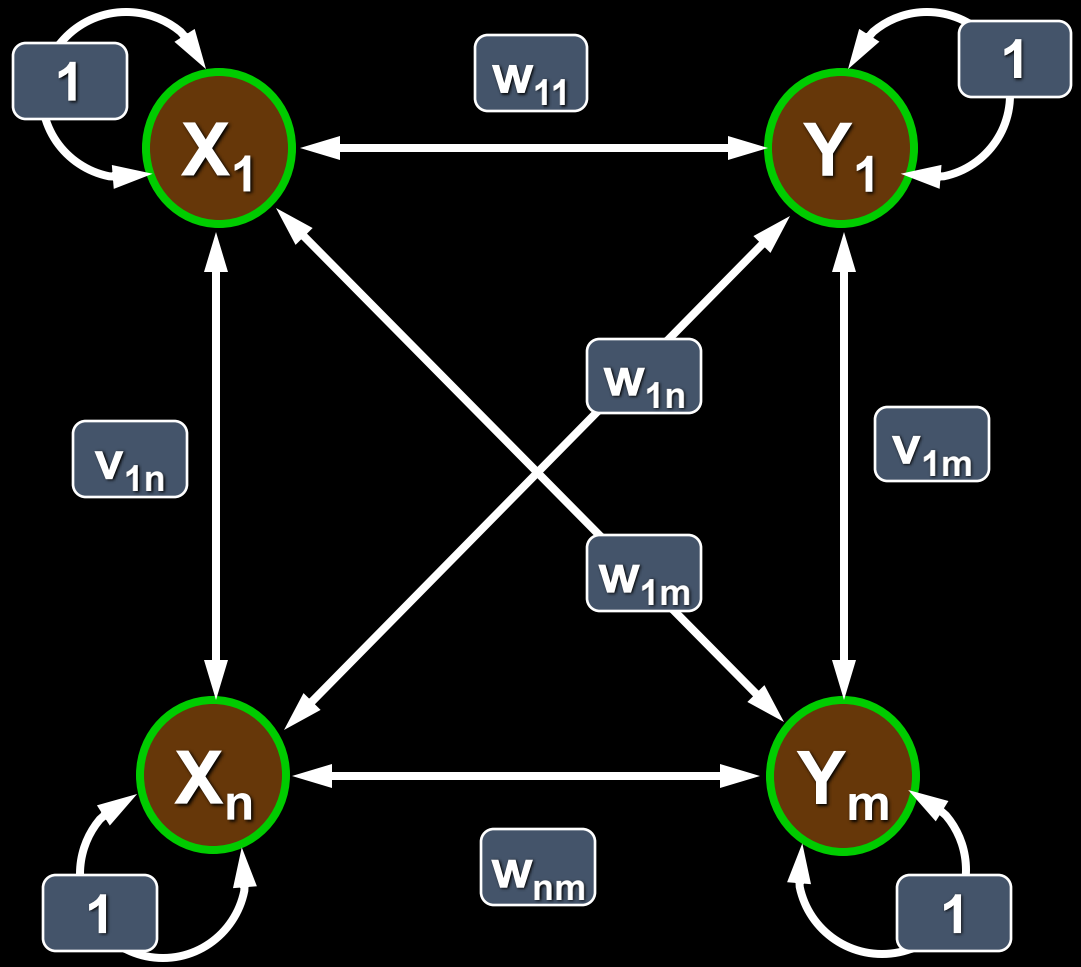


Biological Neurons In Action



Input Units | Hidden Units | Output Units

Model 3: A Recurrent Net



Training

Supervised Training

- Training is accomplished by presenting a sequence of training vectors or patterns, each with an associated target output vector.
- The weights are then adjusted according to a learning algorithm.
- During training, the network develops an associative memory. It can then recall a stored pattern when it is given an input vector that is sufficiently similar to a vector it has learned.

Unsupervised Training

- A sequence of input vectors is provided, but no traget vectors are specified in this case.
- The net modifies its weights and biases, so that the most similar input vectors are assigned to the same output unit.

Common Activation Functions

1. Binary Step Function
(a "threshold" or "Heaviside" function)

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

2. Bipolar Step Function

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

3. Binary Sigmoid Function (Logistic Sigmoid)

$$f(x) = \frac{1}{1+\exp(-\sigma x)}; \ f'(x) = \sigma f(x)[1 - f(x)]$$

3. Bipolar Sigmoid

$$g(x) = \frac{1-\exp(-\sigma x)}{1+\exp(-\sigma x)}; \ g'(x) = \frac{\sigma}{2}[1+g(x)][1-g(x)]$$

4. Hyperbolic - Tangent

$$h(x) = \frac{1-\exp(-2x)}{1+\exp(-2x)}; \ h'(x) = [1+h(x)][1-h(x)]$$

Typical Problem Areas

- The number of application areas in which artificial neural networks are used is growing daily.
- Some of the representative problem areas, where neural networks have been used are:

1. Pattern Completion: ANNs can be trained on a set of visual patterns represented by pixel values.
   - If subsequently, a part of an individual pattern (or a noisy pattern) is presented to the network, we can allow the network's activation to propagate through the net till it converges to the original (memorized) visual pattern.
   - The network is acting like a "content- addressable" memory.

Typical Problem Areas

2. Classification: An early example of this type of network was trained to differentiate between male and female faces.
   - It is actually very difficult to create an algorithm to do so yet, but an ANN has shown to have near-human capacity to do so.
3. Optimization: It is notoriously difficult to find algorithms for solving optimization problems (e.g., TSP).
   - There are several types of neural networks which have been shown to converge to 'good-enough' solutions i.e., solutions which may not be globally optimal but can be shown to be close to the global optimum for any given set of parameters.

Typical Problem Areas

4. Feature Detection: An early example of this is the phoneme producing feature map of Kohonen:
   - The network is provided with a set of inputs and must learn to pronounce the words; in doing so, it must identify a set of features which are important in phoneme production
5. Data Compression: There are many ANNs which have been shown to be capable of representing input data in a compressed format loosing as little of the information as possible.
6. Approximation: Given examples of an input to output mapping, a neural network can be trained to approximate the mapping so that a future input will give approximately the correct answer i.e., the answer which the mapping should give.

Typical Problem Areas

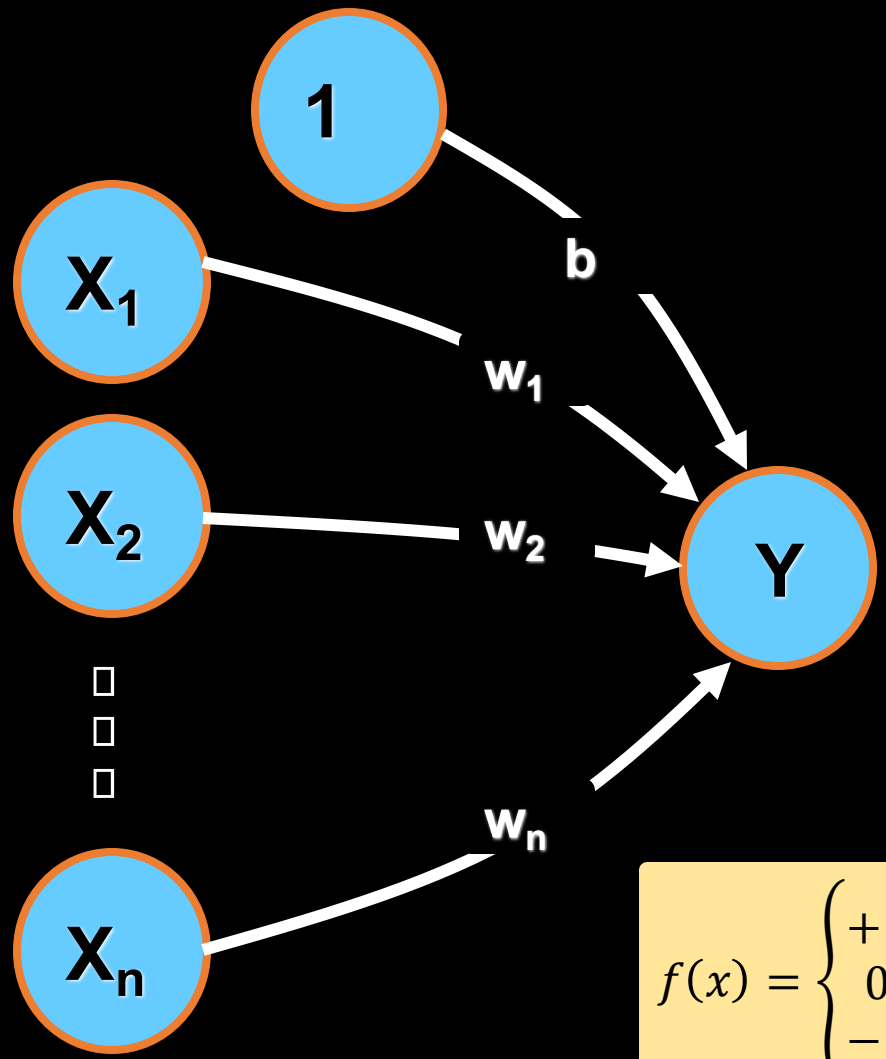
7. Association: We may associate a particular input with a particular output so that given the same (or similar) input again, the net will give the same (or a similar) output again.
8. Prediction: This task may be stated as: given a set of previous examples from a time series, such as a set of closing prices of a Stock Exchange, to predict the next (future) sample.
9. Control: For example, to control the movement of a robot arm (or truck, or any non-linear process) to learn what inputs (actions) will have the correct outputs (results)

Perceptron

- Perceptrons had perhaps the most far-reaching impact of any of the early neural nets.
- A number of different types of Perceptrons have been used and described by various workers.
- The original perceptrons had three layers of neurons – sensory units, associator units and a response unit – forming an approximate model of a retina.
- Under suitable assumptions, its iterative learning procedure can be proved to converge to the correct weights i.e., the weights that allow the net to produce the correct output value for each of the training input patterns.

Simple Perceptron for Pattern Classification

- The architecture of a simple perceptron for performing single classification is shown in the figure.
- The goal of the net is to classify each input pattern as belonging, or not belonging, to a particular class.
- Belonging is signified by the output unit giving a response of +1; not belonging is indicated by a response of -1.
- A zero output means that it is not decisive.



$$f(x) = \begin{cases} +1 & if \quad x > \theta \\ 0 & if \quad -\theta \le x \le \theta \\ -1 & if \quad x < -\theta \end{cases}$$

Perceptron: Algorithm

1. Initialize weights, bias and the threshold θ. Also set the learning rate α such that $(0 < \alpha \leq 1)$.
2. While stopping condition is false, do the following steps.
3. For each training pair s:t, do steps 4 to 7.
4. Set the activations of input units. $x_i = s_i$
5. Compute the response of the output unit.
6. Update weights and bias if an error occurred for this pattern.

   If y is not equal to t (under some limit) then

   $w_i(new) = w_i(old) + \alpha\ x_i\ t$ for i = 1 to n

   $b(new) = b(old) + \alpha\ t$

   end if
7. Test stopping condition: If no weight changed in step 3, stop, else continue.

Perceptron: Example

- Lets consider the following training data:

Inputs		Target
x_1	x_2	t
1	1	1
1	0	-1
0	1	-1
0	0	-1

- We initialize the weights to be $w_1 = 0$, $w_2 = 0$ and $b = 0$. Also, we set $\alpha = 1$, and $\theta = 0.2$.
- The following table shows the sequence in which the net is provided with the input one by one and checked for the required target.

Perceptron: Example

	Input		Bias	Net Input	Output	Target	Weights		
	x_1	x_2		y_{in}	y	t	w_1	w_2	b
1							0	0	0
	1	1	1	0	0	1	1	1	1
	1	0	1	2	1	-1	0	1	0
	0	1	1	1	1	-1	0	0	-1
	0	0	1	-1	-1	-1	0	0	-1
2	1	1	1	-1	-1	1	1	1	0
	1	0	1	1	1	-1	0	1	-1
	0	1	1	0	0	-1	0	0	-2
	0	0	1	-2	-1	-1	0	0	-2

Perceptron: Example

10

Input		Bias	Net Input	Output	Target	Weights		
x_1	x_2		y_{in}	y	t	w_1	w_2	b
1	1	1	1	1	1	2	3	-4
1	0	1	-2	-1	-1	2	3	-4
0	1	1	-1	-1	-1	2	3	-4
0	0	1	-4	-1	-1	2	3	-4

Thus, the positive response is given by all the points such that

$$2x_1 + 3x_2 - 4 > 0.2$$

And the negative response is given by all the points such that

$$2x_1 + 3x_2 - 4 < -0.2$$

Decision Boundaries

- We note that the output of the Perceptron is 1(positive), if the net input 'y_{in}' is greater than θ.
- Also that the output is -1(negative), if the net input 'y_{in}' is less than $-\theta$.
- We also know that

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

  in this case
- Thus, we have

  $2*x_1 + 3*x_2 - 4 > 0.2$ (+ve output)

  $2*x_1 + 3*x_2 - 4 < -0.2$ (-ve output)
- Removing the inequalities and simplifying them gives us the following two equations:

  $2*x_1 + 3*x_2 = 4.2$

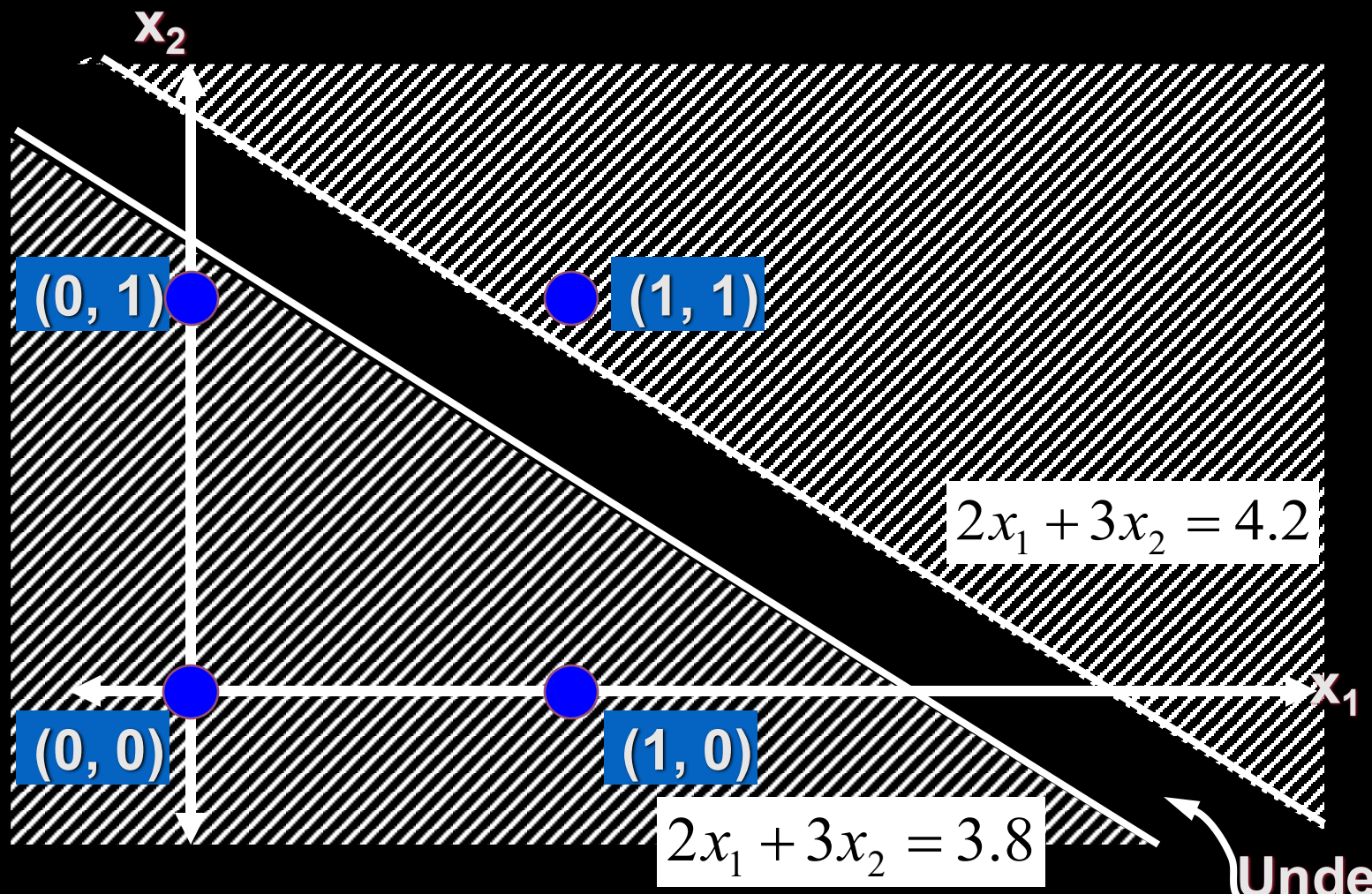
  $2*x_1 + 3*x_2 = 3.8$

Decision Boundaries

- These two equations are equations of straight lines in x1,x2-plane.
- These lines geometrically separate out the input training data into two classes.
- Hence, they serve as acting like decision boundaries.
- The two lines have the same slope, but their y-intercepts are different. So essentially, we have two parallel straight lines acting line decision boundaries for out input data.
- The parameter θ, determines the separation between two lines (and hence the width of the indecisive region).

Perceptron: Example

- Graphically

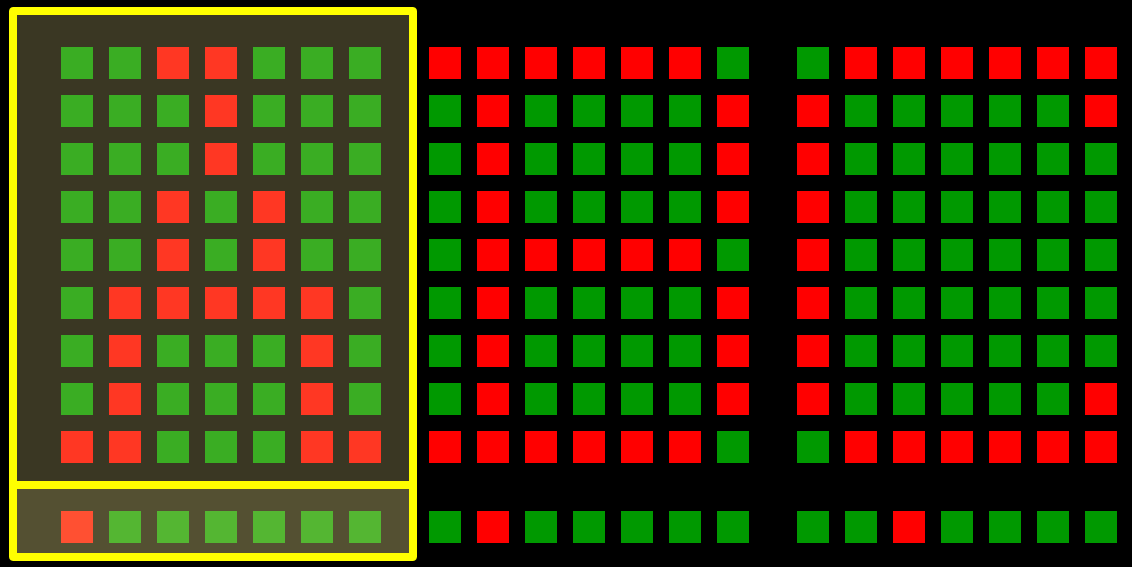


A Perceptron to classify letters from different fonts: one output class

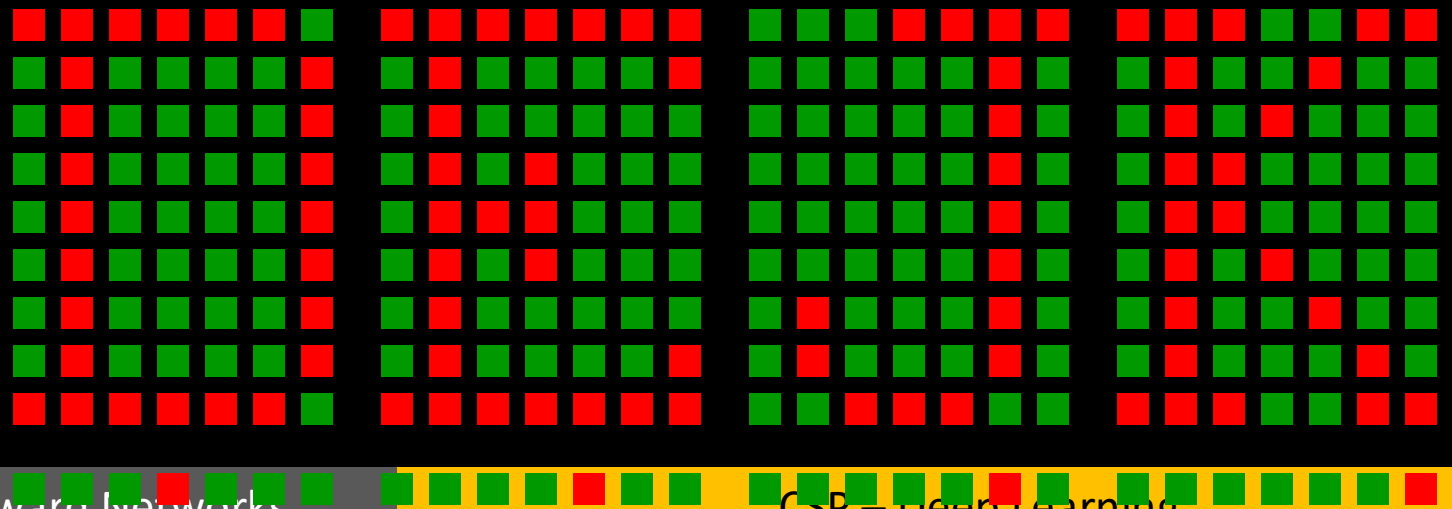
1

0

Input from Font 1



$$s_1 = [\begin{matrix}-1,-1,+1,+1,-1,-1,-1,\\ -1,-1,-1,+1,-1,-1,-1,\\ -1,-1,-1,+1,-1,-1,-1,\\ -1,-1,+1,-1,+1,-1,-1,\\ -1,-1,+1,-1,+1,-1,-1,\\ -1,+1,+1,+1,+1,+1,-1,\\ -1,+1,-1,-1,-1,+1,-1,\\ -1,+1,-1,-1,-1,+1,-1,\\ +1,+1,-1,-1,-1,+1,+1\end{matrix}]$$



$$t_1 = [\begin{matrix}+1,\\ -1,\\ -1,\\ -1,\\ -1,\\ -1,\\ -1,\\ -1,\\ -1\end{matrix}]$$

A Perceptron to classify letters from different fonts: one output class

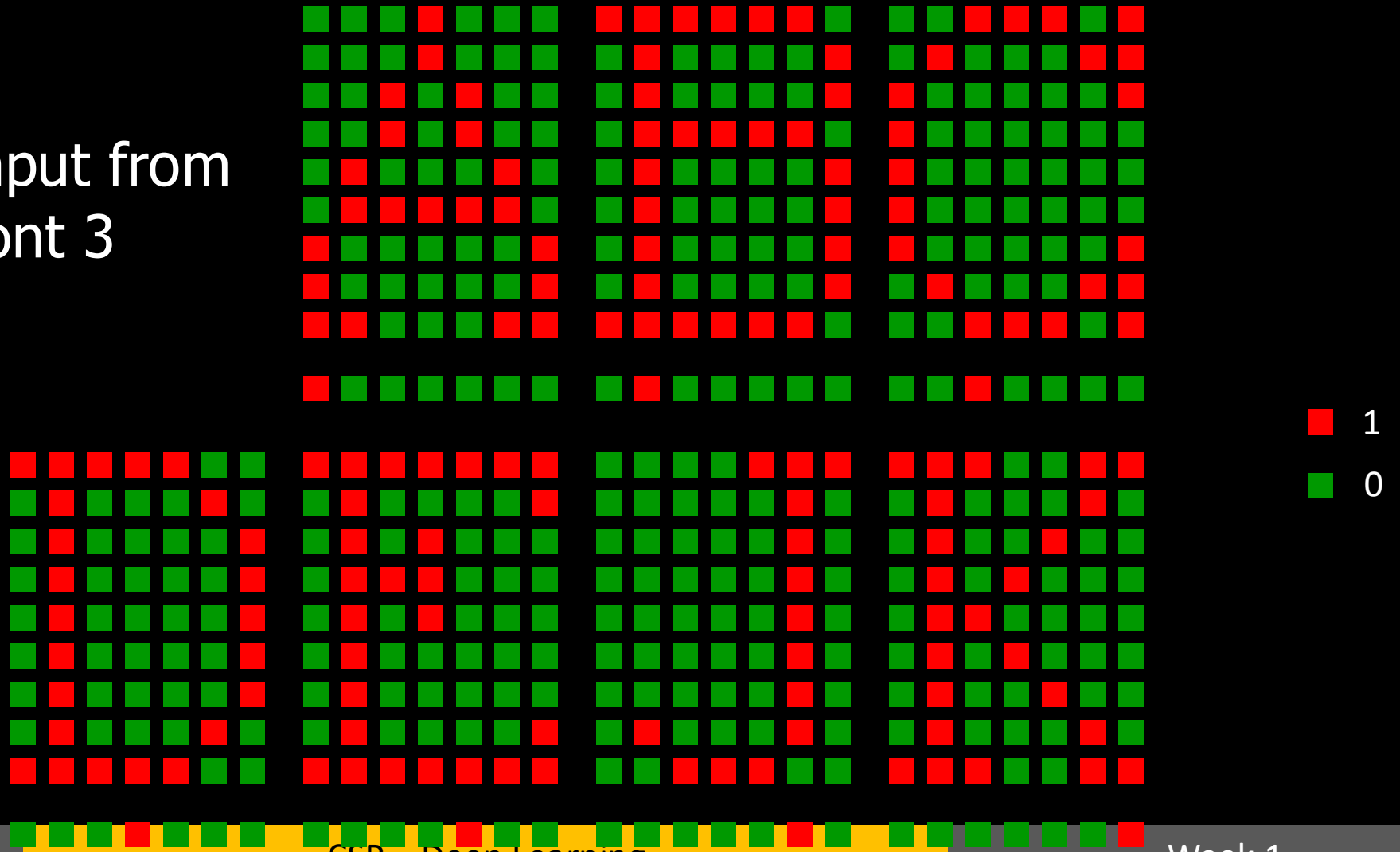
Input from Font 2

1

0

A Perceptron to classify letters from different fonts: one output class

Input from Font 3



A Perceptron to classify letters from different fonts: one output class

- Consider the 21 input patterns as shown on the last three slides.
- Let us first assume that we want to train a simple perceptron to classify each of the above input patterns as belonging, or not belonging to the class A (letters which are very similar to A).
- In that case, the target value for each pattern is either +1 or -1. There are three examples of A and 18 examples of not-A in last three slides.
- We could, of course, use the same vectors as examples of B or not-B and train the net in a similar way.
- Note, however, that because we are using a single layer net, the weights for the output unit signifying A, do not have any interaction with the weights for the output unit signifying B.

A Perceptron to classify letters from different fonts: one output class

- Therefore, we can solve these problems at the same time, by allowing a column of weights for each output unit.
- Our net would have 63 input units and 2 output units.
- The first output unit would correspond to "A" or "not-A", the second unit to "B" or "not-B".
- Continuing this idea, we can identify 7 output units, one for each of the 7 categories into which we wish to classify out input.

A Perceptron to classify letters from different fonts: several output classes

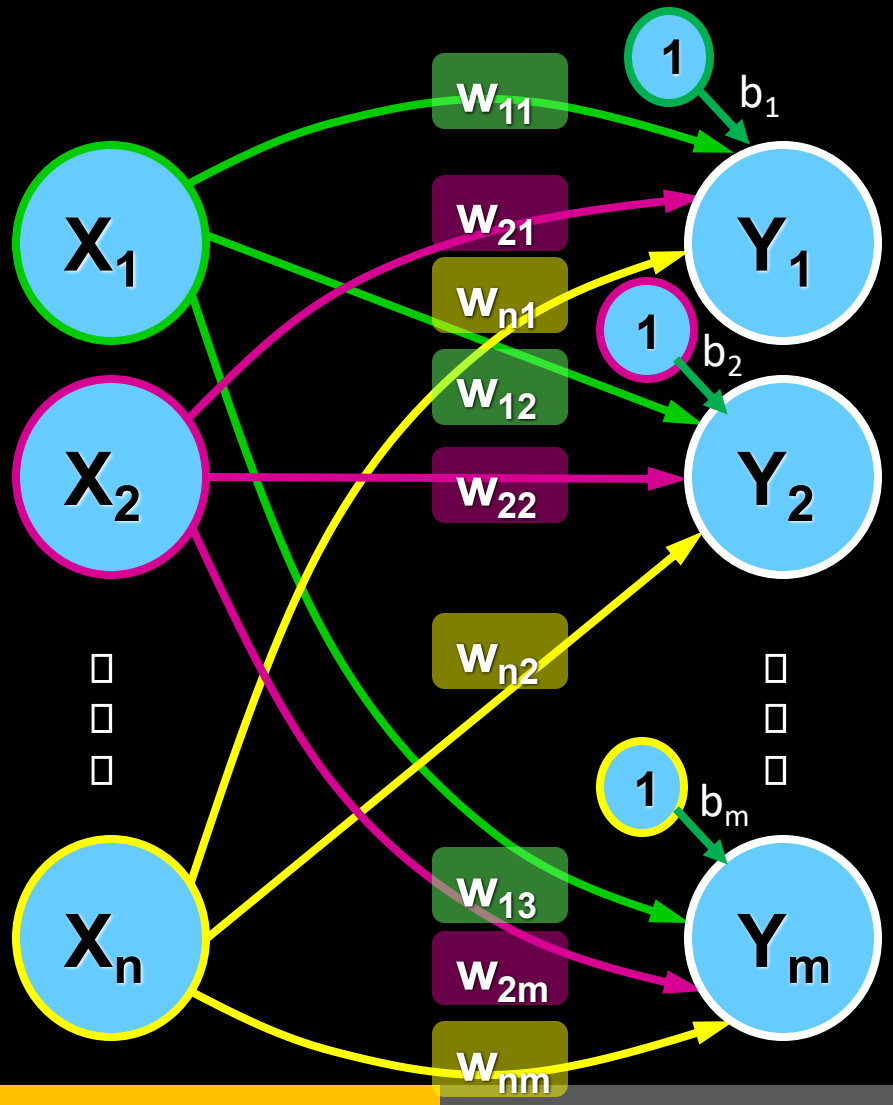
- **The simple perceptron discussed so far, can be extended easily to the case where the input vectors belong to the one (or more) of several categories to which the input vectors may belong.**
- **The architecture of such a net is shown in the figure.**

Net Inputs

$$y_{1\_in} = b_1 + w_{11}x_1 + w_{21}x_2 \ldots + w_{n1}x_n$$
$$y_{2\_in} = b_2 + w_{12}x_1 + w_{22}x_2 \ldots + w_{n2}x_n$$
$$\vdots$$
$$y_{m\_in} = b_m + w_{1m}x_1 + w_{2m}x_2 \ldots + w_{nm}x_n$$

$$\Rightarrow \begin{bmatrix} y_{1\_in} \\ y_{2\_in} \\ \vdots \\ y_{m\_in} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} + \begin{bmatrix} w_{11} & w_{21} & \cdots & w_{n1} \\ w_{12} & w_{22} & \cdots & w_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1m} & w_{2m} & \cdots & w_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$\Rightarrow \boldsymbol{y_{in} = b + WX}$$



A Perceptron to classify letters from different fonts: several output classes

- For this example, each input vector is a 63-tuple representing a letter expressed as a pattern on a 7x9 grid of pixels.
- There are seven categories to which each input vector may belong, so there are seven components to the output vector, each representing a letter: A, B, C, D, E, K or J.
- The training input patterns, and target responses must be converted to an appropriate form for the neural net to process.
- A bipolar representation has better computational characteristics than does a binary representation.
- We would also require a 'modified training algorithm' for several output categories. One such algorithm is given on the next slide.

Modified Perceptron Algorithm for Several Output Units

1. Initialize weights, bias and the threshold θ. Also set the learning rate α such that (0 < α <=1).
2. While stopping condition is false, do the following steps.
3. For each training pair **s**:**t**, do steps 4 to 6.
4. Set the activations of input units. $x_i = s_i$
5. Compute the response y_j of each output unit Yj, j=1,…,m.
6. Update weights and bias if an error occurred for this pattern for any of the output unit.

   If y_j is not equal to t_j (under some limit) then

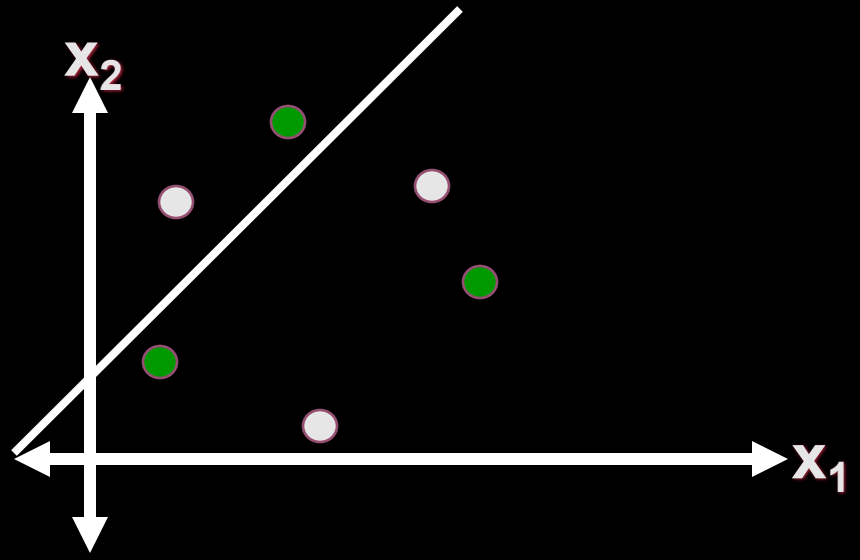
   $$w_{ij}(new) = w_i(old) + \alpha\, x_i\, t_j \qquad \text{for } j = 1, 2, \dots, m; i = 1, 2, \dots, n$$

   $$b_j(new) = b_j(old) + \alpha t_j$$

   end if
7. Test stopping condition: If no weight changed in step 3, stop, else continue.

Problem with Perceptron Learning

1. The major problem with Perceptron learning is that it cannot separate out those classes which are not linearly separable.



2. Thus, there is a need to improve upon the existing model and develop a technique which could be useful for a wider range of problems. … Solution to this problem was provided by Widrow's Delta Rule…

Delta Rule

- The delta rule changes the weights of the neural network connections so as to minimize the difference between the net input y_{in} (which is considered to be equal to the output 'y' in this case) to the output unit and the target value 't'.
- The aim is to minimize the error over all training patterns. This is accomplished by reducing the error for each pattern one at a time.

Case 1: Single Output Unit

- In this case the square of the error for a particular training pattern is:

$$E = (t - y_{in})^2 = E(w_1, w_2, \cdots, w_n)$$

Delta Rule

Case 1: Single Output Unit

- The gradient of a scalar E, is a vector having the components:

$$\nabla E = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \frac{\partial E}{\partial w_3}, \cdots, \frac{\partial E}{\partial w_n}\right]$$

Gives maximum rate of change of E

- and it gives the direction of most rapid increase in E. The negative gradient of E gives the direction of most rapid decrease in E. The error can be reduced by adjusting the weights w_i in the direction of negative of the gradient:

$$y_{in} = b + \sum_i w_i x_i$$

$$\frac{\partial E}{\partial w_I} = -2(t - y_{in})\frac{\partial y_{in}}{\partial w_I} = -2(t - y_{in})x_I$$

Delta Rule

- **Thus, the local error will be reduced most rapidly (for a given learning rate α) by adjusting the weights according to the delta rule:**

$$\Delta w_I = \alpha (t - y_{in}) x_I$$

Delta Rule

Case 2: Several Output Units

- In this case:

$$y_{in\_j} = b_j + \sum_i w_{ij} x_i \quad and \quad E = \sum_{j=1}^{m} (t_j - y_{in\_j})^2$$

$$\frac{\partial E}{\partial w_{IJ}} = \frac{\partial}{\partial w_{IJ}} (t_J - y_{in\_J})^2 = -2(t_J - y_{in\_J}) x_I$$

- Thus, the delta rule for multiple output units' case is:

$$\Delta w_{IJ} = \alpha (t_J - y_{in\_J}) x_I$$

Adaline: Algorithm

ADALINE stands for ADAptive LInear NEuron.

1. Initialize weights and bias. Also set the learning rate α such that ($0 < \alpha <= 1$).
2. While stopping condition is false, do the following steps.
3. For each training pair s:t, do steps 4 to 6.
4. Set the activations of input units. $x_i = s_i$
5. Compute the response of the output unit.
6. Update weights and bias if an error occurred for this pattern.

   $w_i(new) = w_i(old) + \alpha (t - y_{in})x_i$ for i = 1 to n

   $b(new) = b(old) + \alpha (t - y_{in})$
7. Test stopping condition: If the largest weight change that occurred in step 2 is smaller than a specified tolerance, then stop; else continue.

Adaline

Learning Rate α:

- Take small values of α (say α = 0.1)
- If α is too large, learning process will not converge.
- If α is too small, learning will be extremely slow.
- A practical range for α is

  $0.1 <= n\ \alpha <= 1.0$,

  where ‘n’ is the number of input units.

How to use Jupyter Notebook on Google Colab?

- This requires you to have a google account (if you have a gmail account, you are good to go). Please make one before proceeding further.
- Open your favorite browser and go to the address: https://colab.research.google.com
- You will see Colab site opening up with a pop-up window as shown in the adjacent image.
- Use the tab item “Upload” the Jupyter notebook that you have downloaded from the course website.