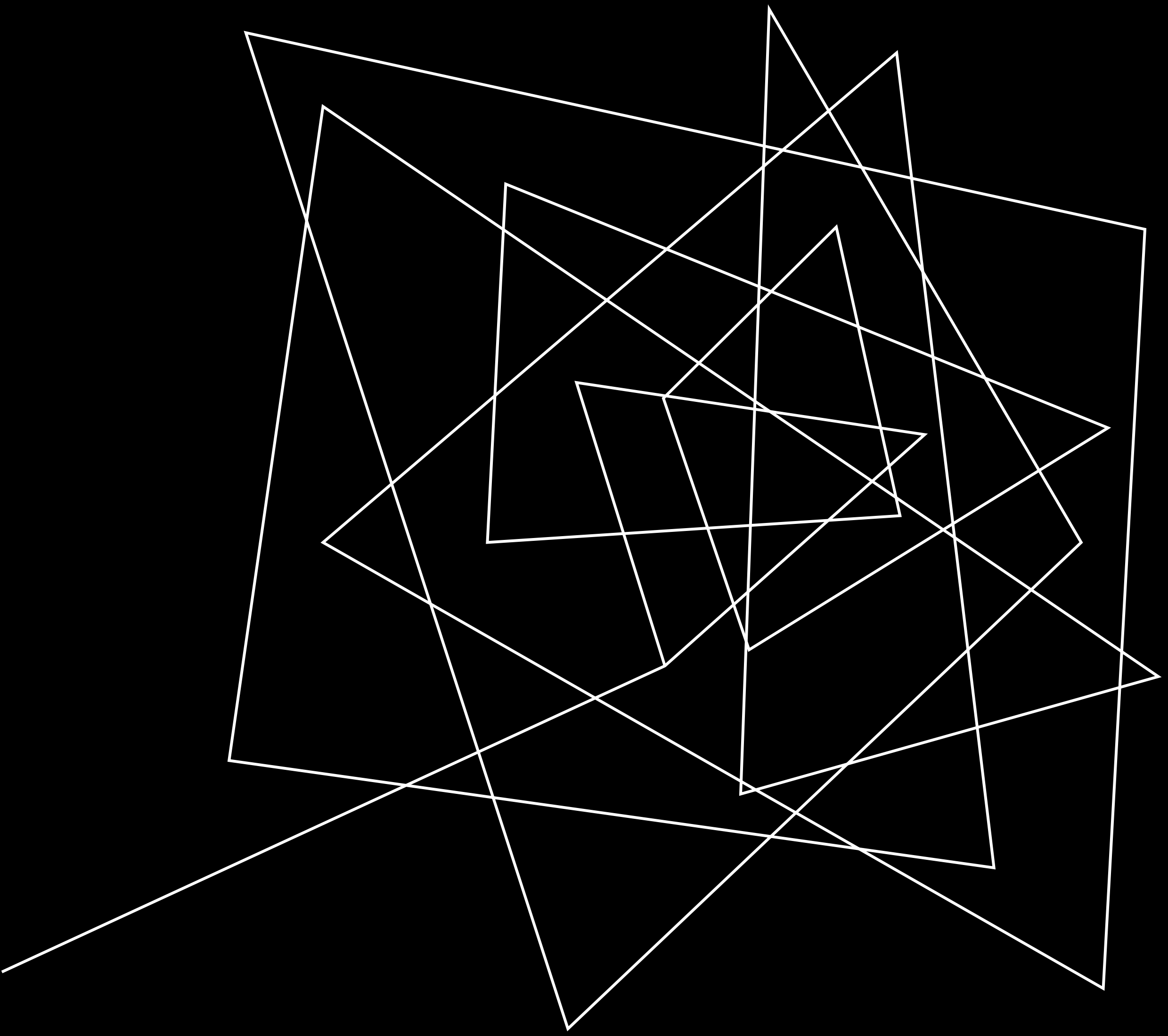


DEEP LEARNING

Perceptron, Artificial Neural Networks, and Backpropagation

Deep Learning: Introduction

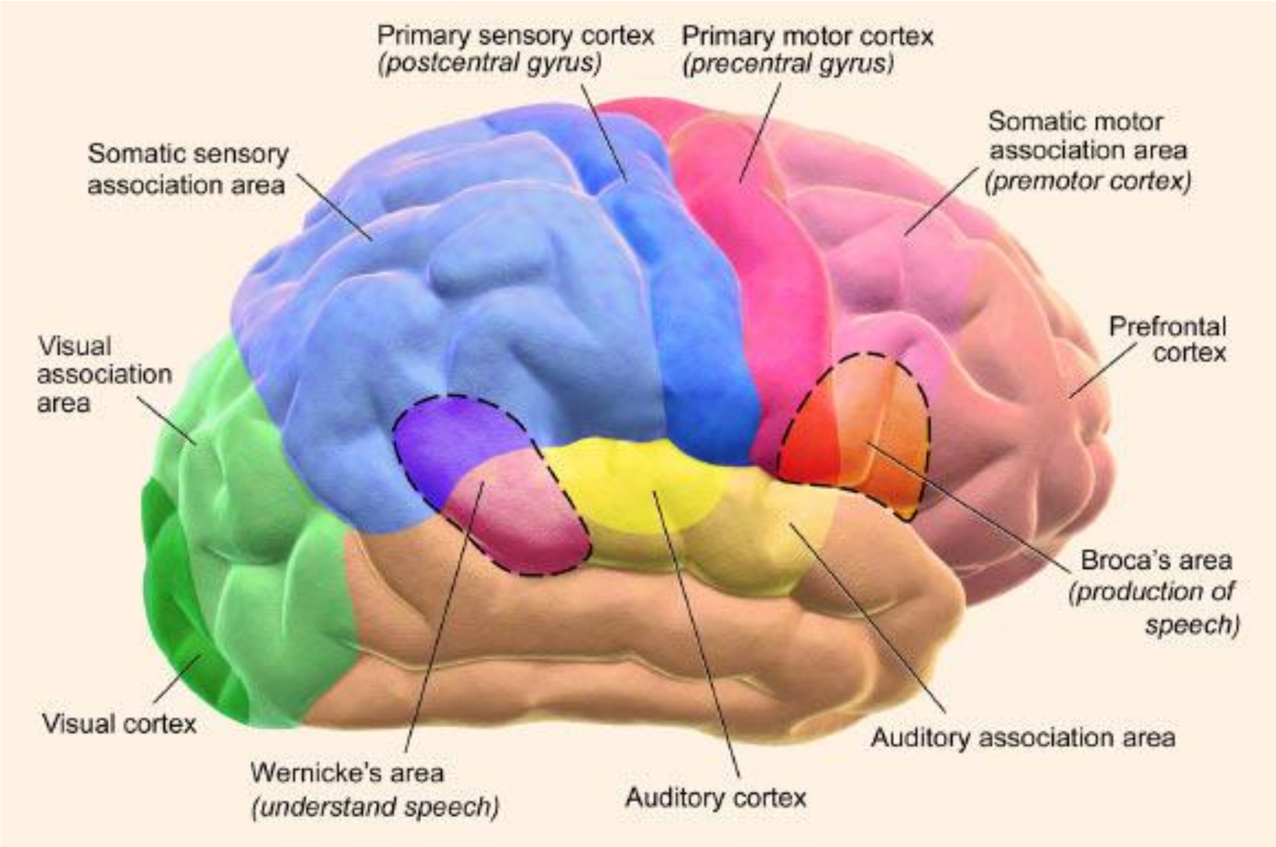
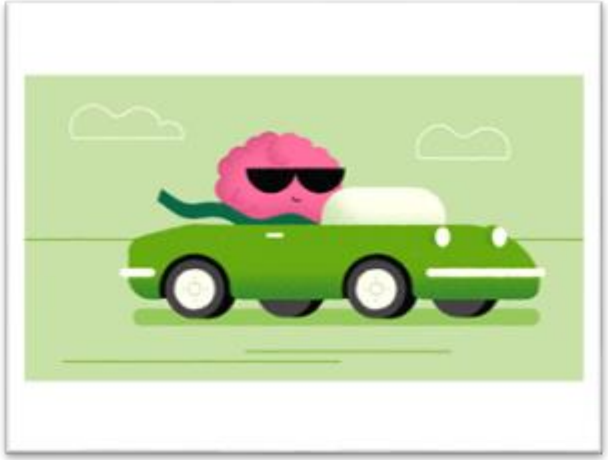


Fig. Human brain

Deep Learning: Introduction

The human brain is amazing:

1. Robust and powerful even with the existence of some problems (Neuroplasticity)
2. The ability to learn and to adapt with new and unfamiliar environments
3. The ability to deal with incomplete and noisy information
4. Parallel Processing/Computing



Deep Learning: Introduction

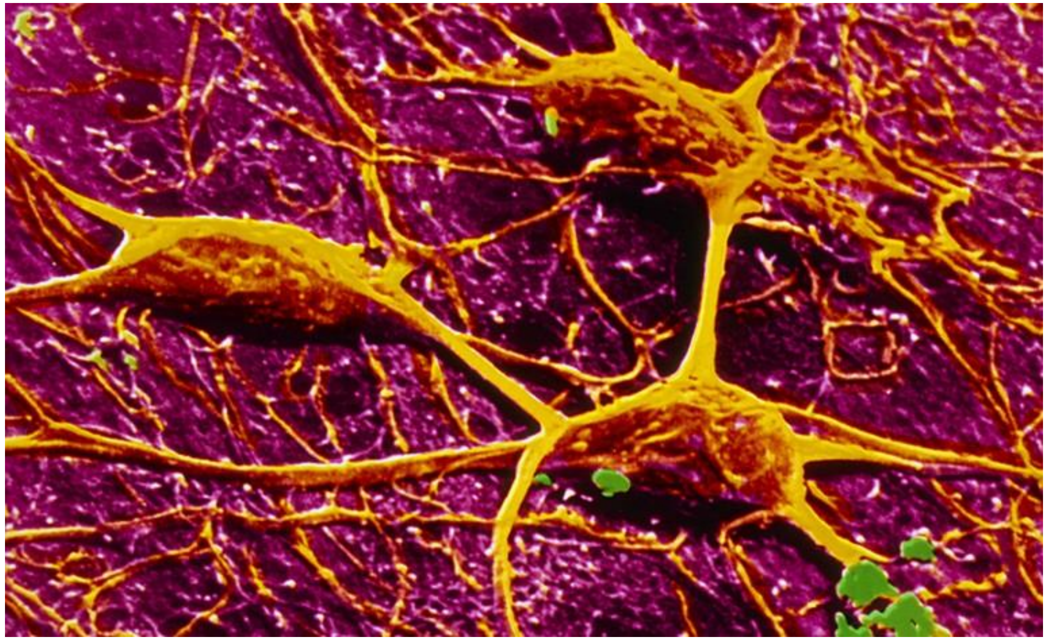


Fig. Neurons store and transmit information in the brain.
Credit: CNRI/SPL

Gif. Biological neurons

Deep Learning: Introduction

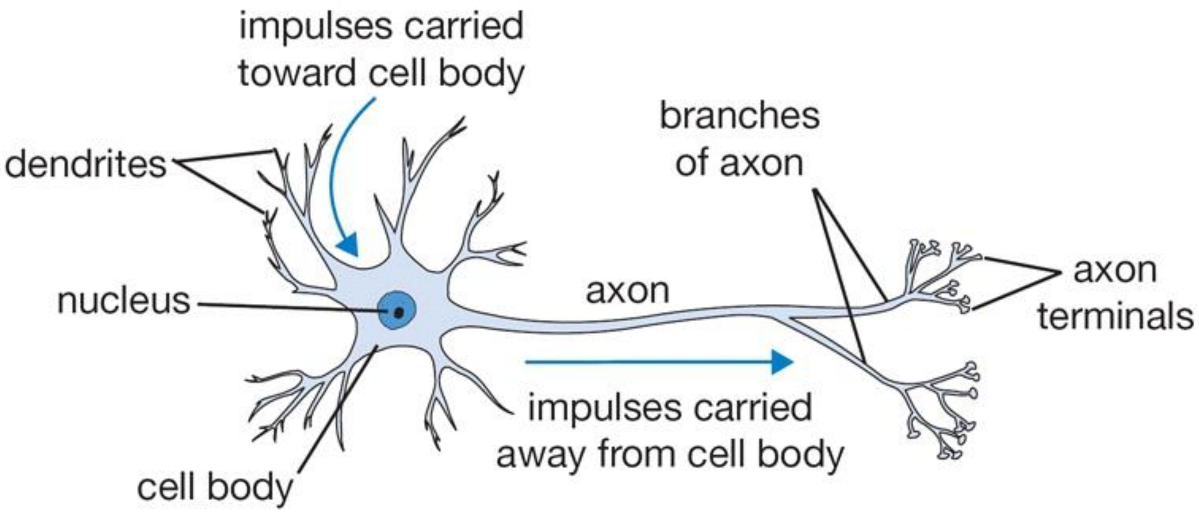


Fig. A cartoon drawing of a biological neuron

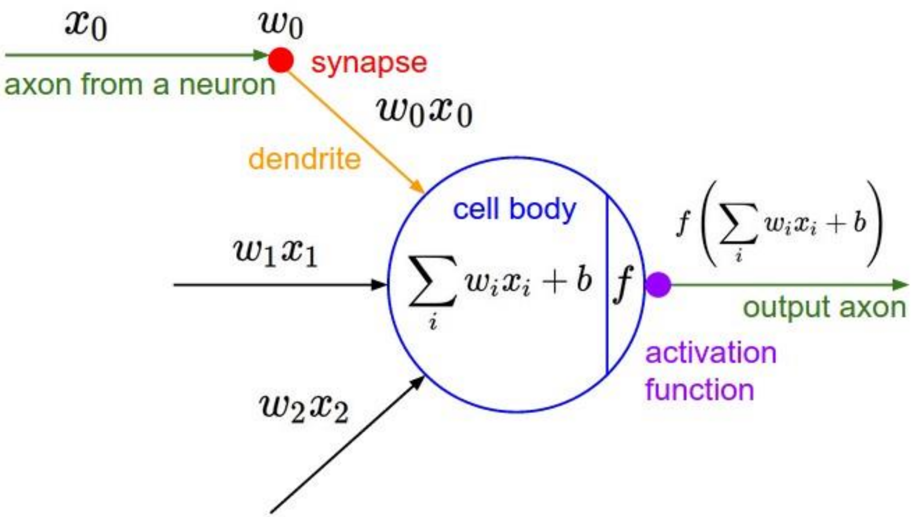


Fig. Mathematical model of the biological neuron

Deep Learning: Perceptron

Perceptron is a simplified model of a biological neuron proposed by **Rosenblatt** (1957-58)

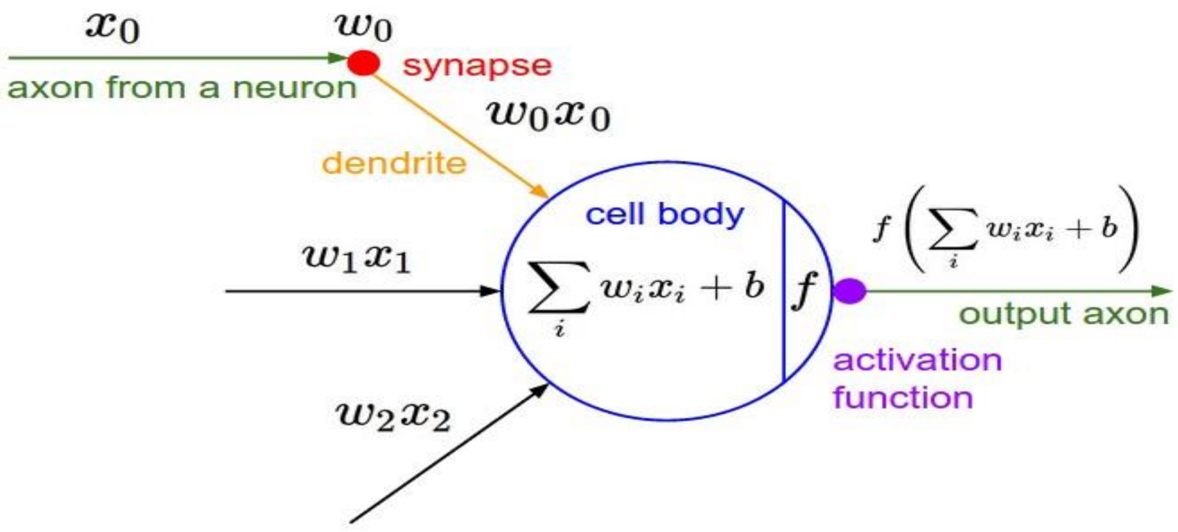


Fig. Mathematical model of the biological neuron (Perceptron)

Fig. Frank Rosenblatt

Deep Learning: Perceptron

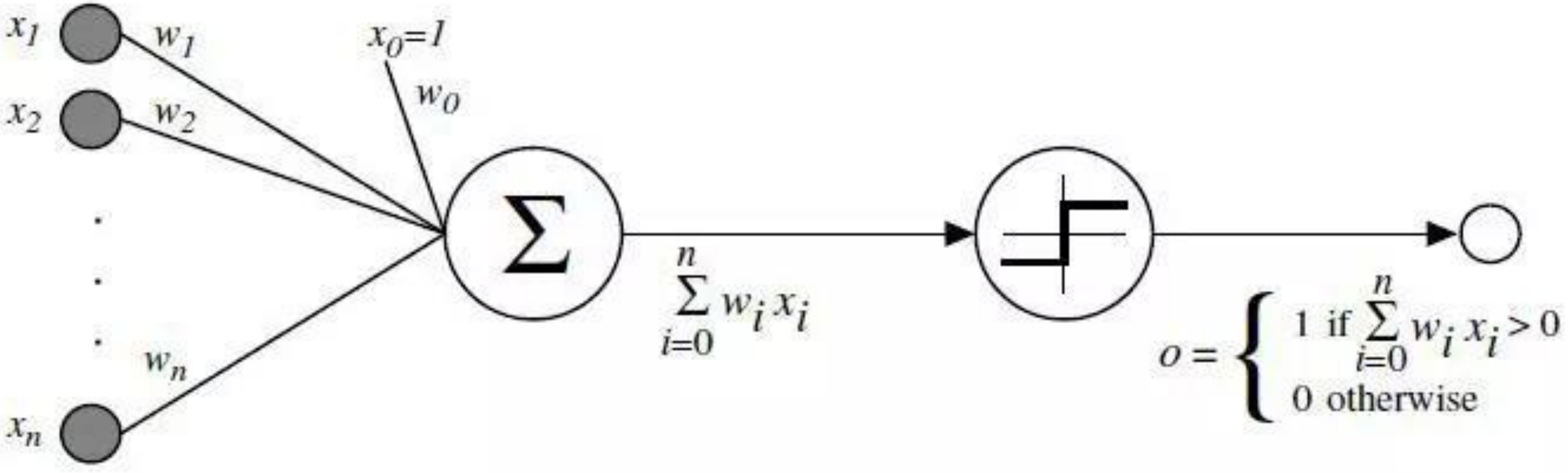


Fig. A diagram showing how the Perceptron works.

Deep Learning: Mathematical Model of The Perceptron

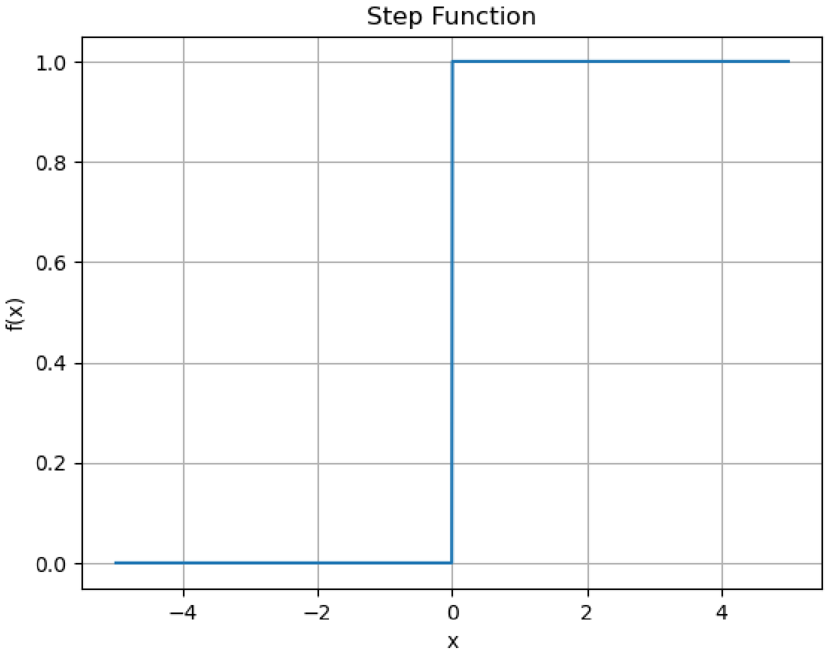
$$output = f(\sum_{i=1}^{n} w_i x_i + b)$$

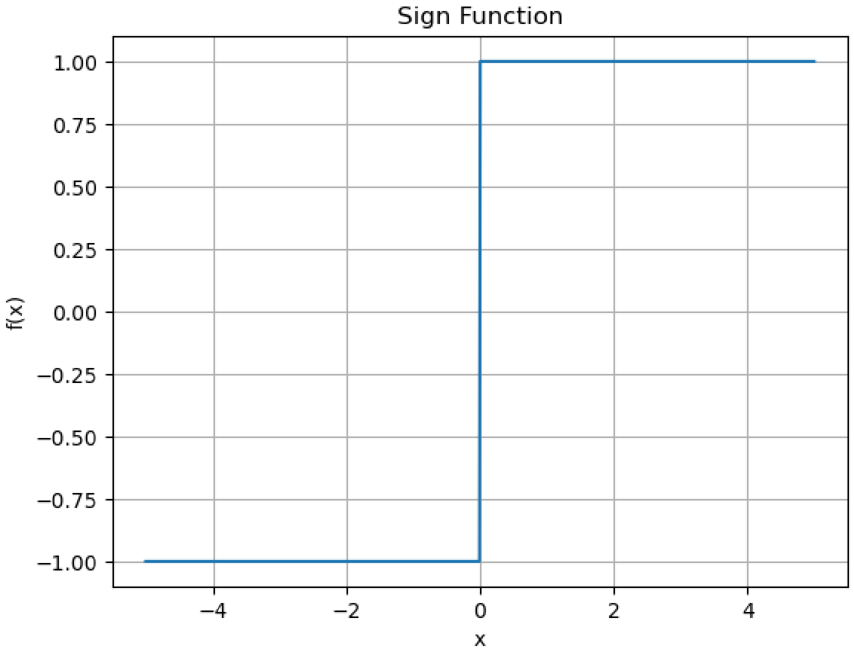
Weights

Bias

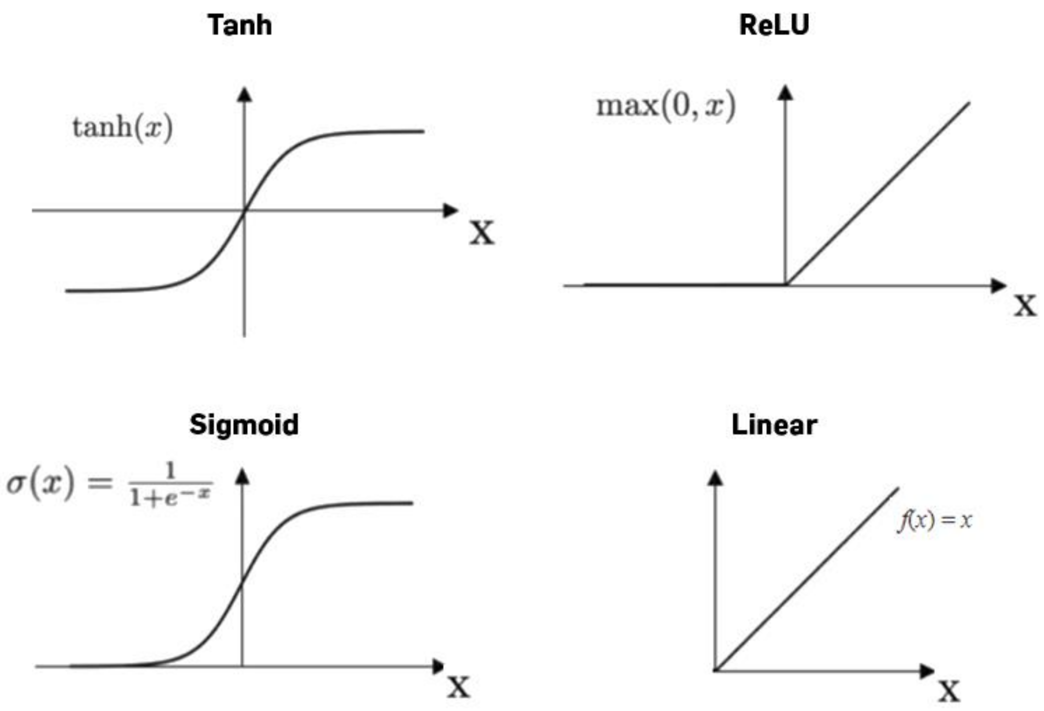
Activation function

Perceptron: Activation Functions



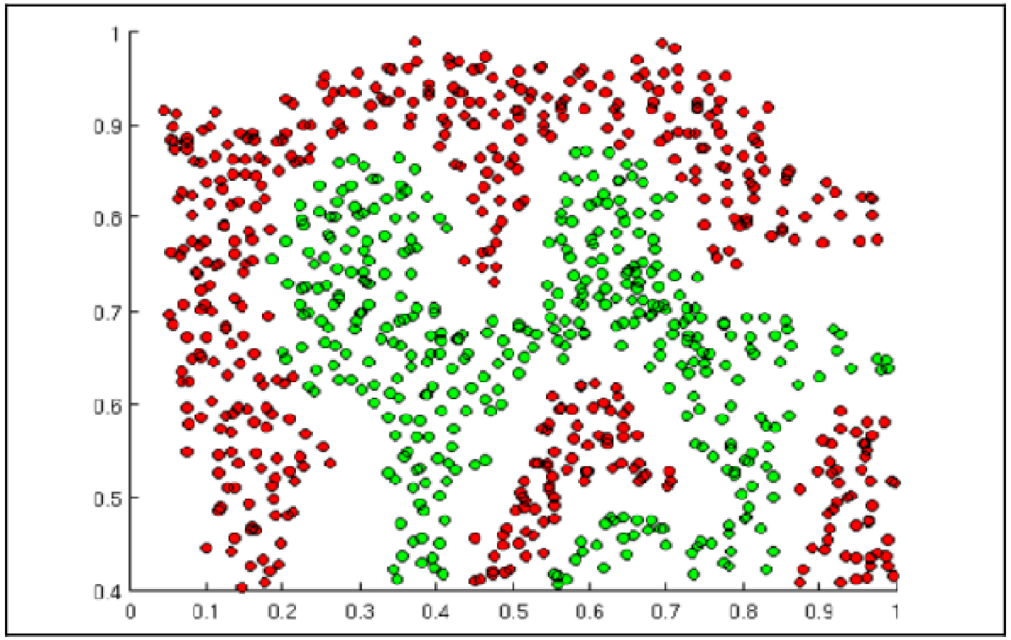


Perceptron: Activation Functions



Perceptron: Importance of Activation Functions

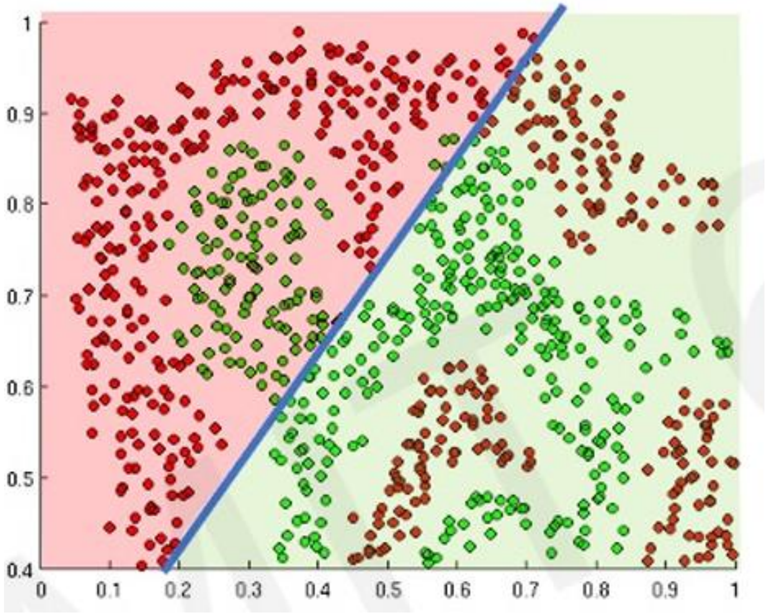
The importance of the activation functions is to introduce non-linearities into the network.



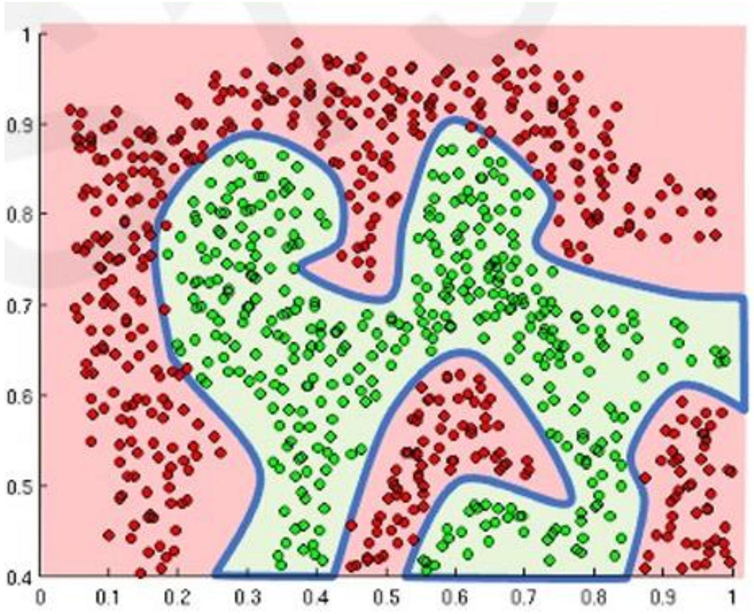
What if we wanted to build a neural network to distinguish green vs red points?

Perceptron: Importance of Activation Functions

The importance of the activation functions is to introduce non-linearities into the network.



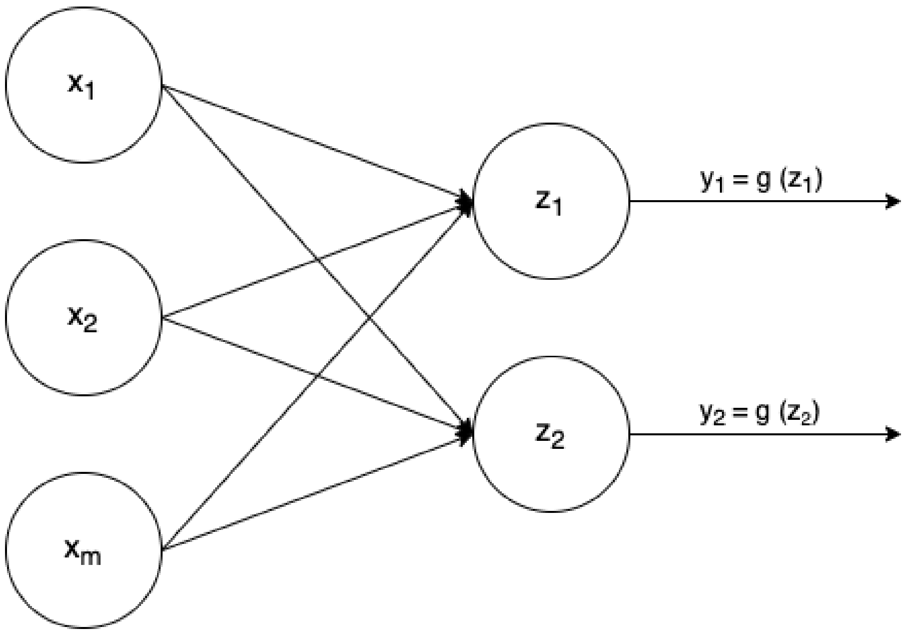
Linear activation functions produce linear decision boundaries no matter the network size



Non-linear activation functions allow to approximate complex function

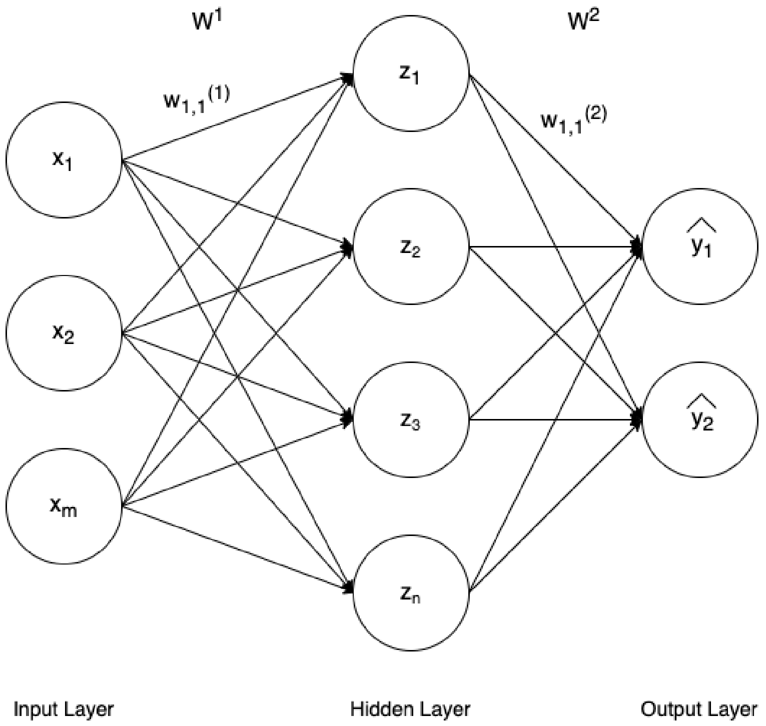
Perceptron: Multi-output Perceptron

- The **problem** of **XOR** logic function demonstrated the **limitations** of the **perceptron.**
- We can link two or more **perceptrons** with each other.
- Each perceptron is connected and linked with each input, it is called **fully connected layer** or **dense layer**.



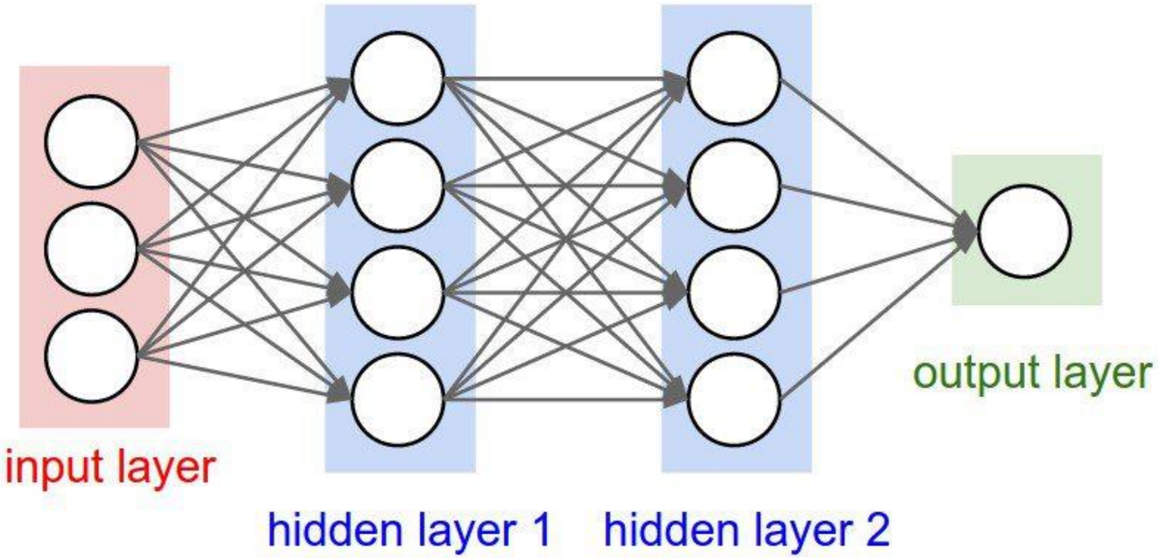
Neural Networks: Single Layer Neural Network

- Like in the human brain, the power and robustness of the biological neurons is when they are fully connected to each other.
- The figure in the right shows the simplest architecture of single layer neural network

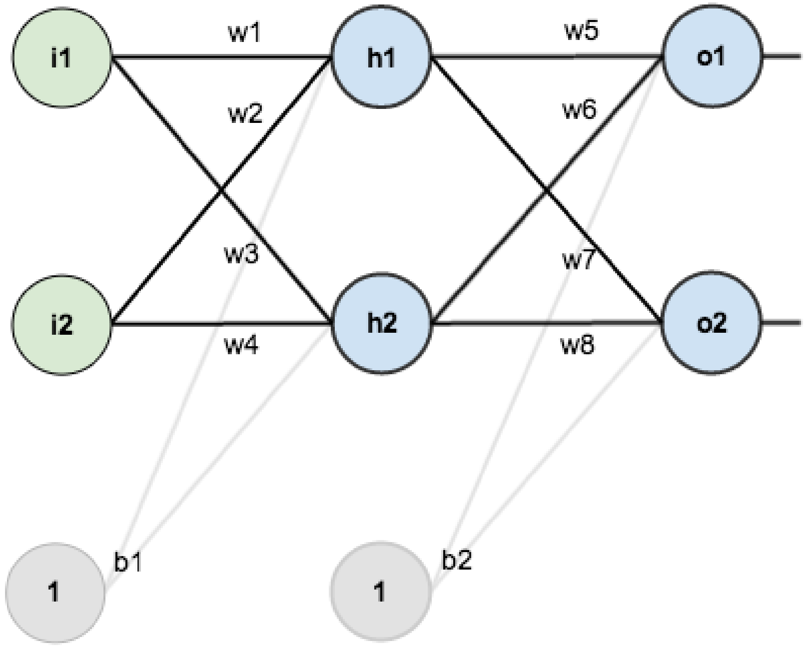


Neural Networks: Multi-Layer Neural Network

Like in the human brain, the power and robustness of the biological neurons is when they are fully connected to each other, it starts to become more and more complex by adding/stacking more and more hidden layers, at this level we are talking about **Multi-layer perceptron.** **(DEEP LEARNING)**



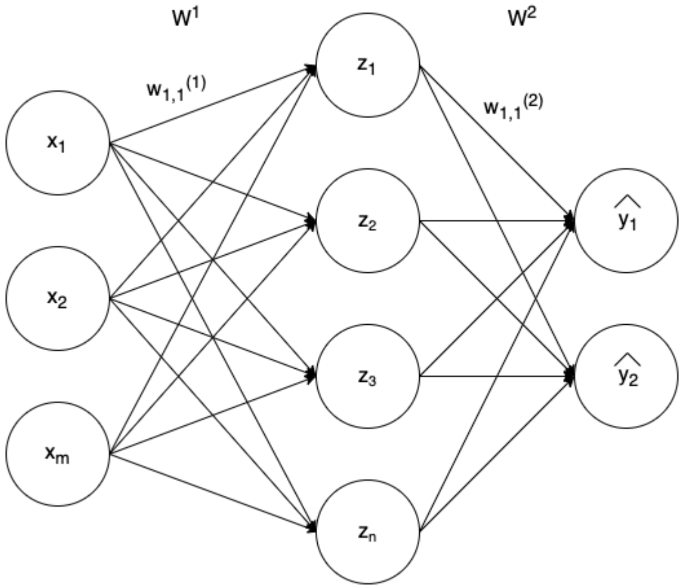
Deep Learning: Weights and Biases in a Neural Network



- The ultimate goal is to **find** the **optimal** values for the **weights** and **biases** that provide **good** predictions and thus **high accuracy**
- **Training** the artificial neural network to find the **weights** and **biases**

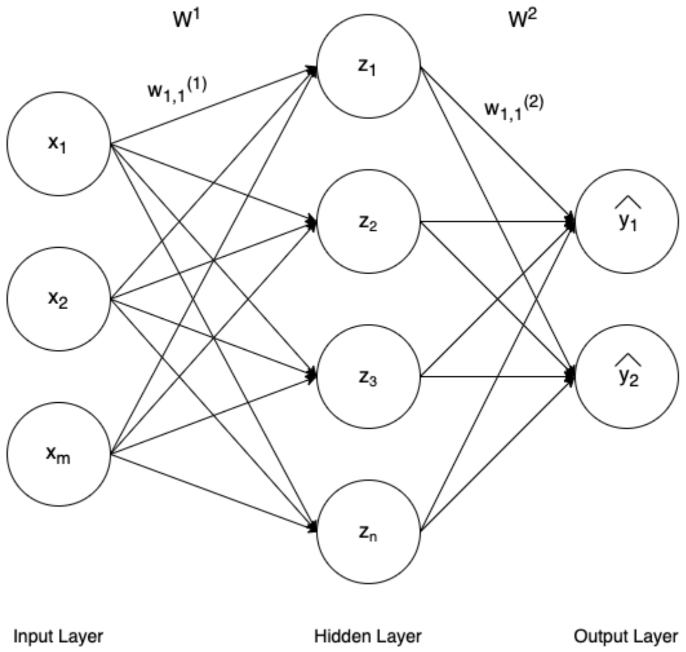
Deep Learning: How is a neural network "classically" trained?

How do humans learn?



Note: The term "classically" refers to one of the widely used techniques employed to train a neural network. There are many other "advanced " techniques

Deep Learning: How is a neural network "classically" trained?



- Before training a neural network, we have to define the prediction target (regression or classification) in order to determine the **loss/cost function** to be **minimized**

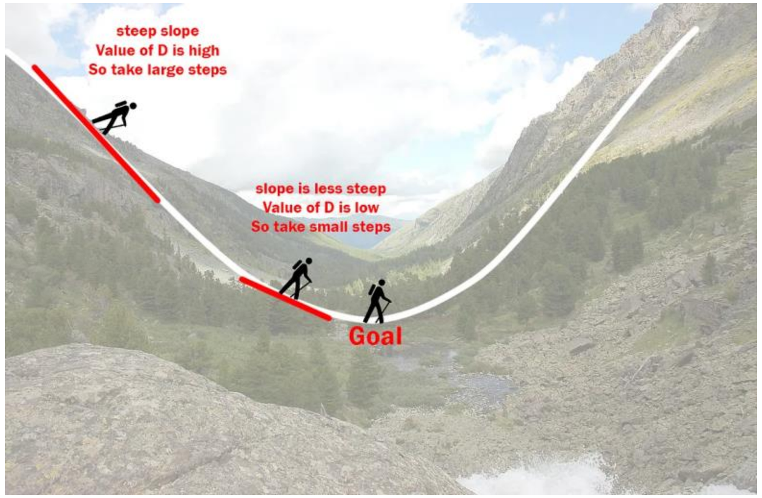
Deep Learning: How is a neural network "classically" trained? <Loss Optimization>

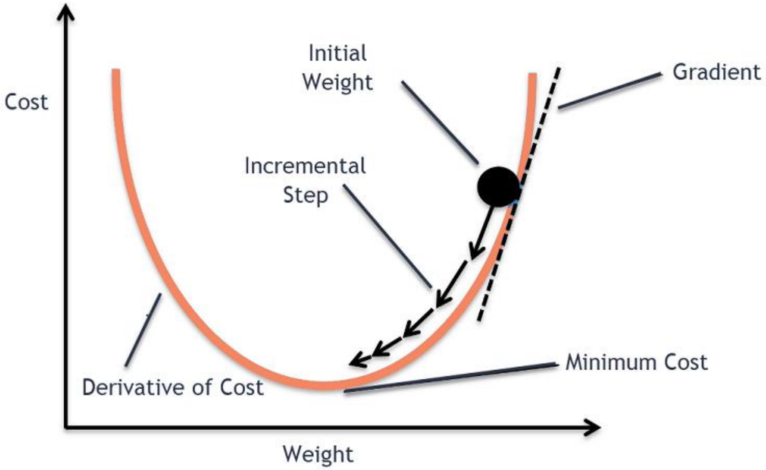
- Let's break this down, and let's suppose that we have a supervised learning task **T**, with inputs **X** and output **y.**
- Let **f** represent the neural network, **L** the loss function, and **W** the ensemble of weights and biases
- The **goal** is to find the parameters (weights and biases) that minimize the loss function **L.**

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}}\, J(\boldsymbol{W})$$

Deep Learning: Gradient Descent Is Back
<Loss Optimization>



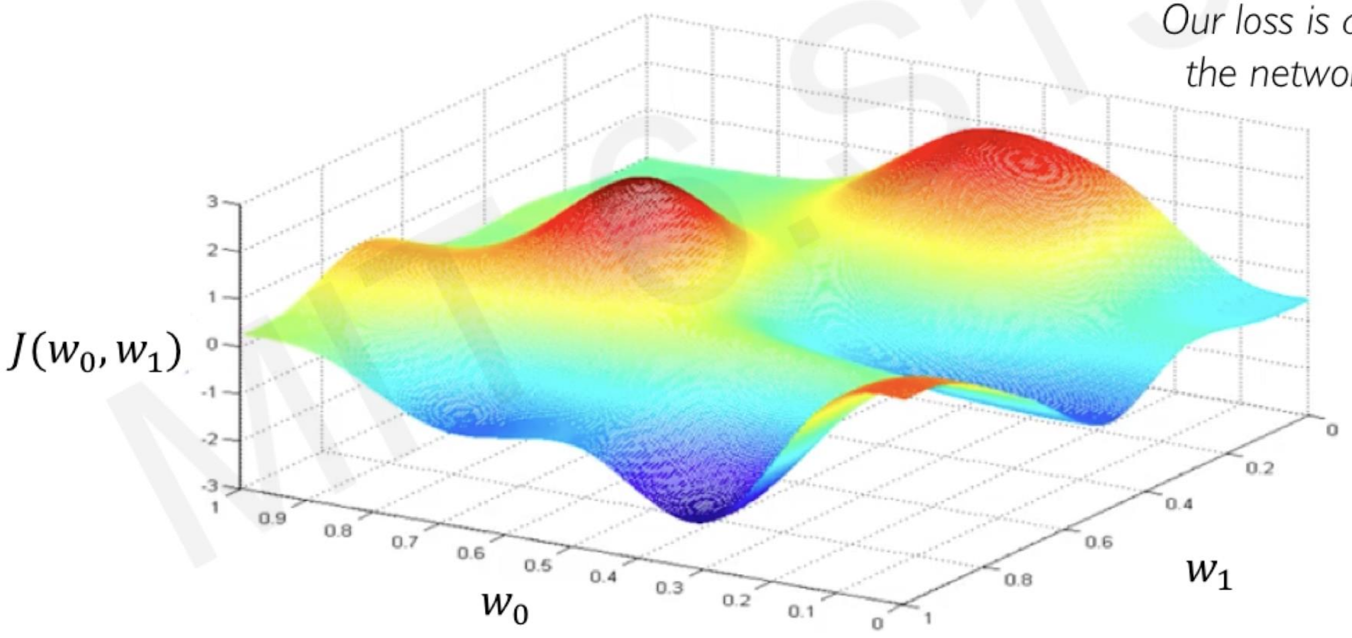


Note: Gradient descent is one of the optimization algorithms to find the parameters (weights and biases) that minimize the loss function (Local or Global Minimum)

Deep Learning: Gradient Descent <Loss Optimization>

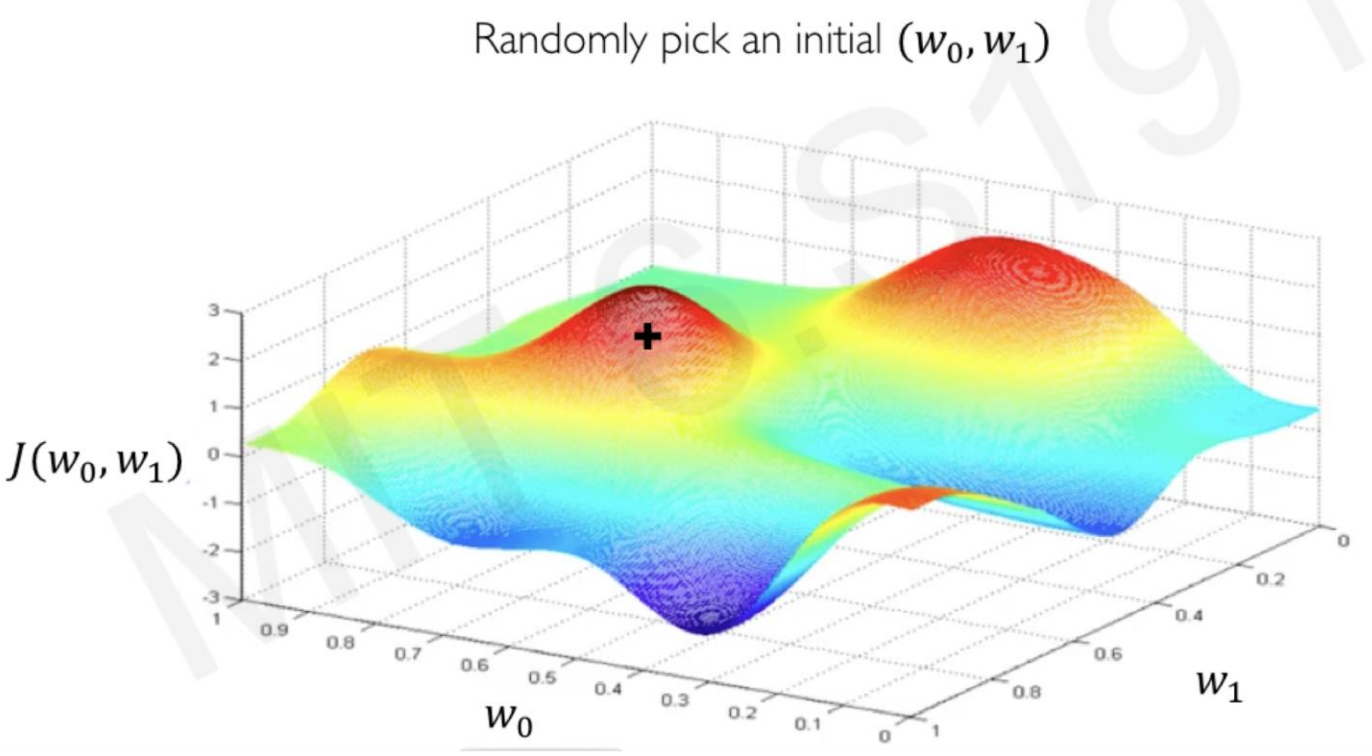
$$W^* = \underset{W}{\operatorname{argmin}}\, J(W)$$

Remember:
Our loss is a function of the network weights!



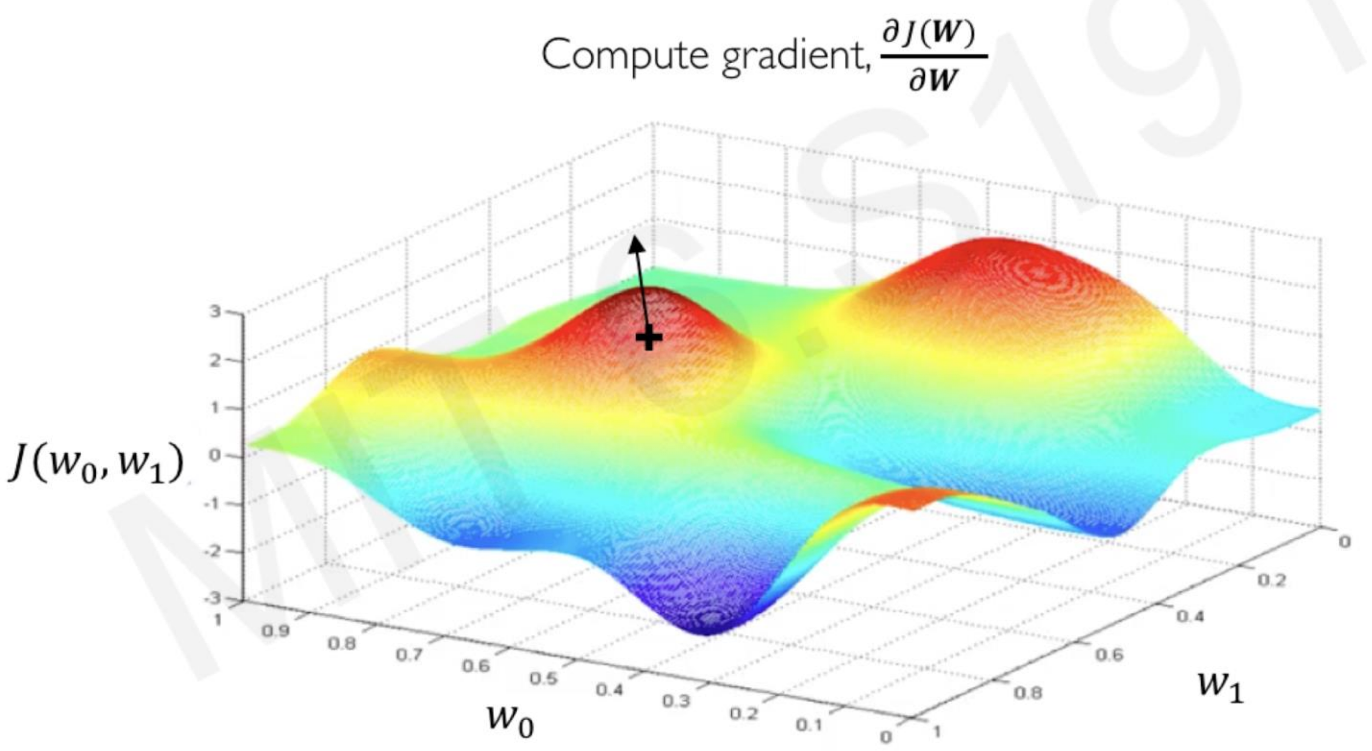
Source: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Deep Learning: Gradient Descent <Loss Optimization>



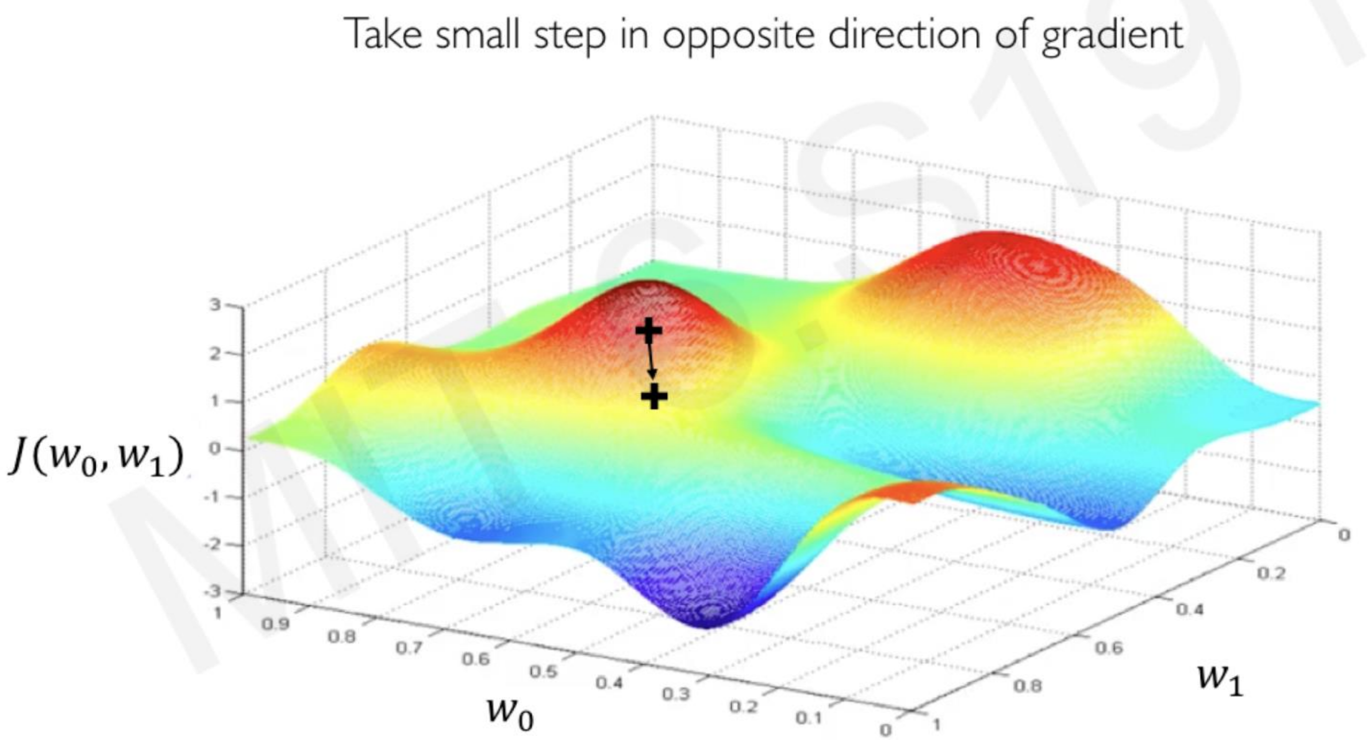
Source: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Deep Learning: Gradient Descent
<Loss Optimization>



Source: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

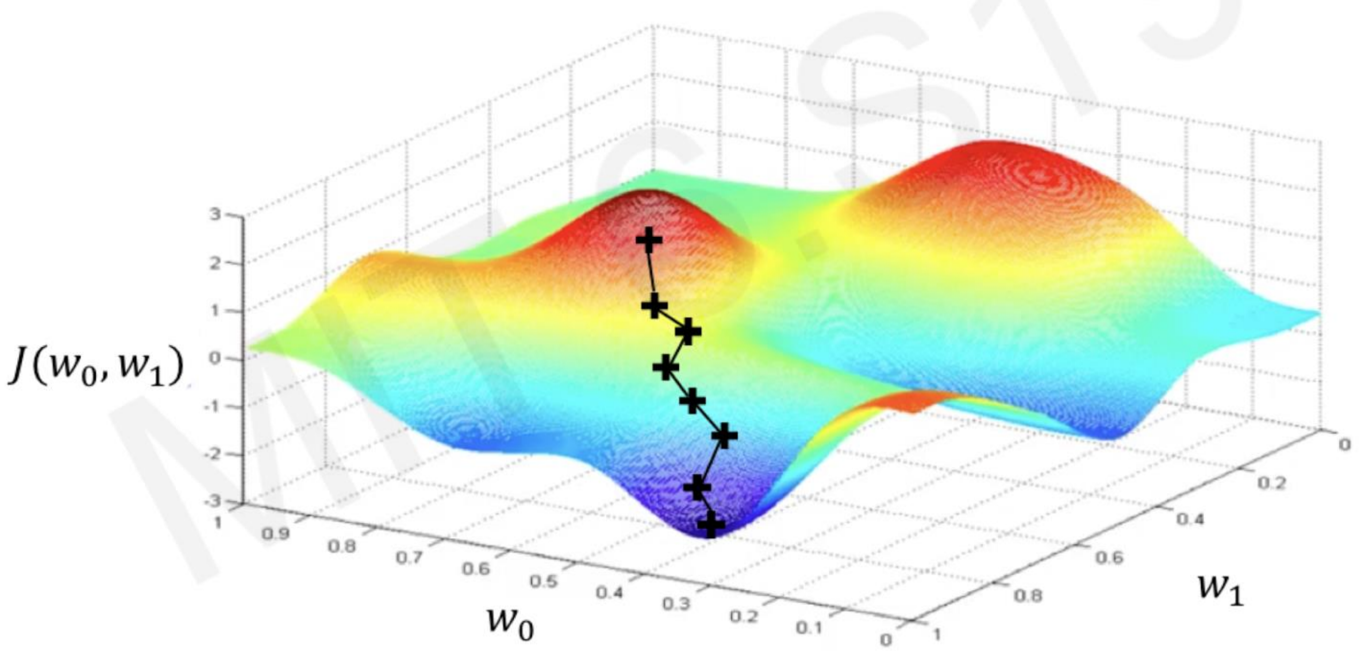
Deep Learning: Gradient Descent
<Loss Optimization>



Source: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Deep Learning: Gradient Descent <Loss Optimization>

Repeat until convergence



Source: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Deep Learning: Gradient Descent <Loss Optimization>

Gradient Descent

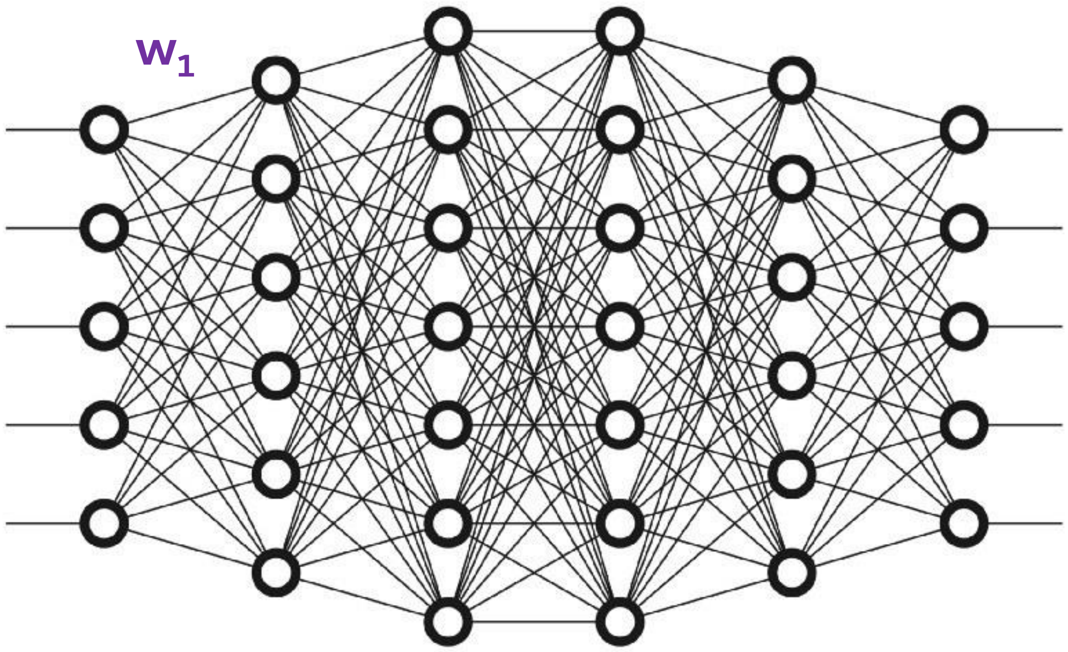
Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Problem: How to compute the gradient for a deep neural network?

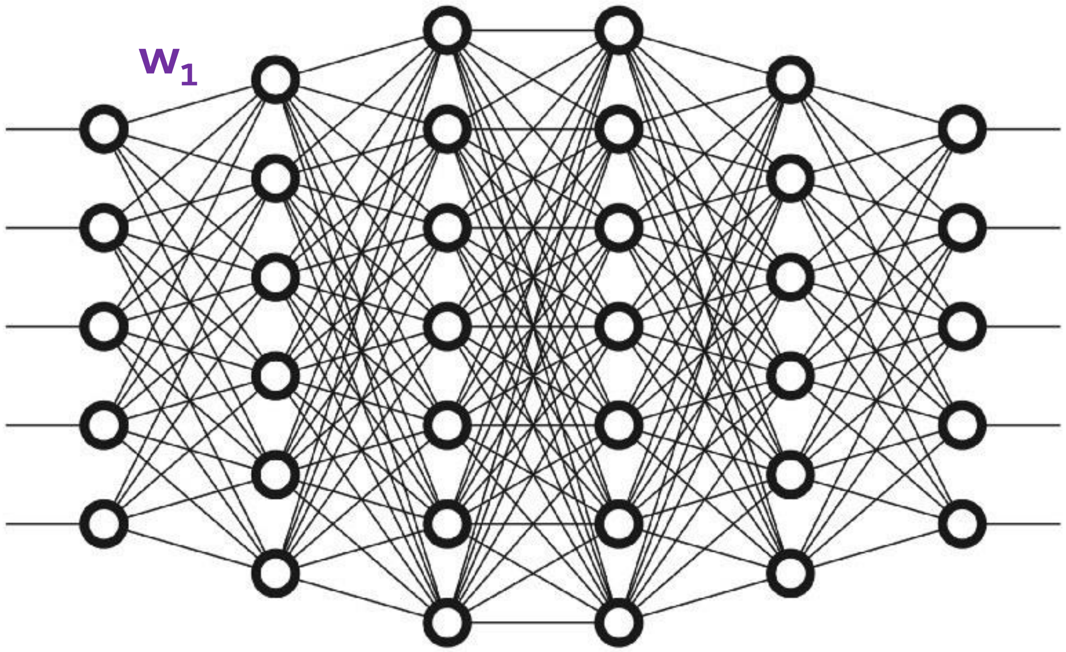
Source: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Deep Learning: Gradient Descent



$$\frac{\partial J(W)}{\partial w1}?$$

Deep Learning: Backpropagation

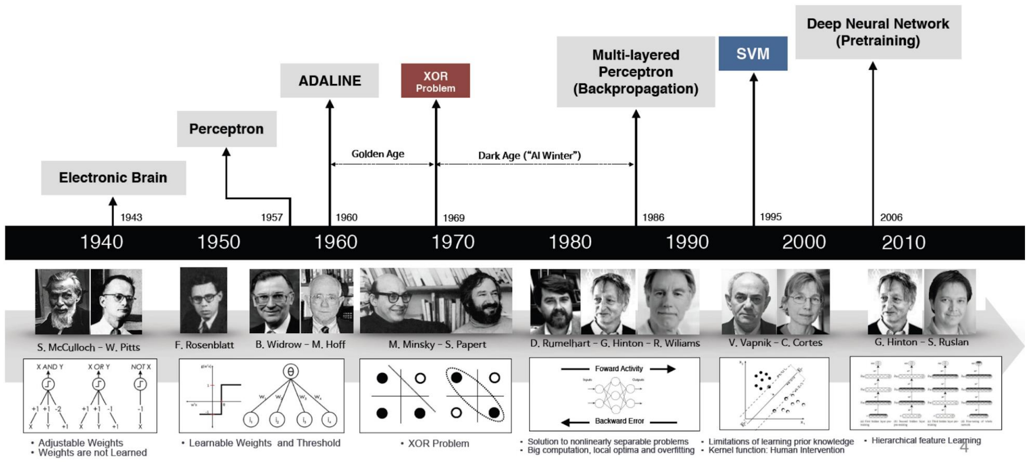


Backpropagation is the solution

$$\frac{\partial J(W)}{\partial w1}?$$

Deep Learning: Backpropagation

- **Backpropagation** is one of the reasons that made training deep neural network possible.
- **Backpropagation** is basically based on **chain rule.**



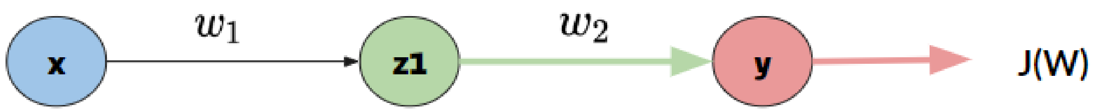
The Chain Rule If f and g are both differentiable and $F = f \circ g$ is the composite function defined by $F(x) = f(g(x))$, then F is differentiable and F' is given by the product

$$F'(x) = f'(g(x))g'(x)$$

In Leibniz notation, if $y = f(u)$ and $u = g(x)$ are both differentiable functions, then

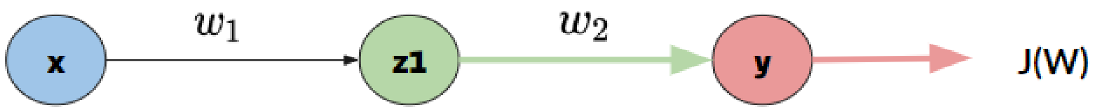
$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

Deep Learning: Backpropagation
<Example>



$$\frac{\partial J(W)}{\partial w_2} = \frac{\partial J(W)}{\partial y} * \frac{\partial y}{\partial w_2}$$

Deep Learning: Backpropagation
<Example>



$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial y} * \frac{\partial y}{\partial w_1}$$

Re-applying the chain rule

$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial y} * \frac{\partial y}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

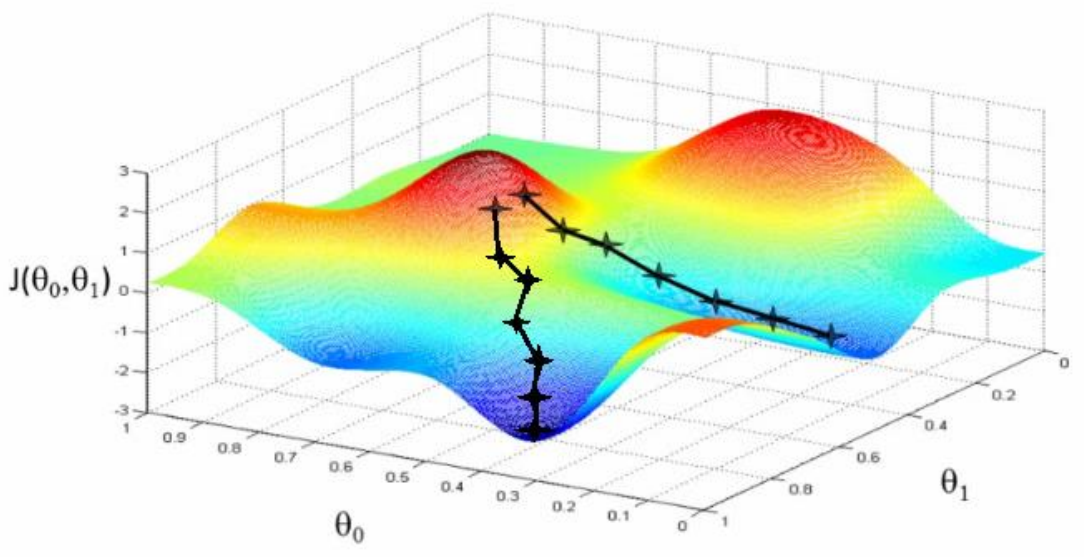
Deep Learning: What is the difference between Gradient Descent and Backpropagation?

- **Gradient descent** is an optimization algorithm for minimizing the loss of a predictive model with regard to a training dataset.
- **Back-propagation** is an automatic differentiation algorithm for calculating gradients for the weights in a neural network graph structure.
- **Gradient descent** and the **back-propagation of error** algorithms together are used to train neural network models.

	Backpropagation	Gradient Descent
Definition	An algorithm for calculating the gradients of the cost function	Optimization algorithm used to find the weights that minimize the cost function
Requirements	Differentiation via the chain rule	•Gradient via Backpropagation •Learning rate
Process	Propagating the error backwards and calculating the gradient of the error function with respect to the weights	Descending down the cost function until the minimum point and find the corresponding weights

Deep Learning: Gradient Descent Algorithms

- **Gradient Descent**
- Stochastic Gradient Descent **(SGD)**
- **Adam** (Adaptable moment estimation)
- Adaptive Gradient Algorithm (**AdaGrad**)
- Root Mean Square Propagation (**RMSProp**)



Deep Learning: Stochastic Mini-batch Gradient Descent

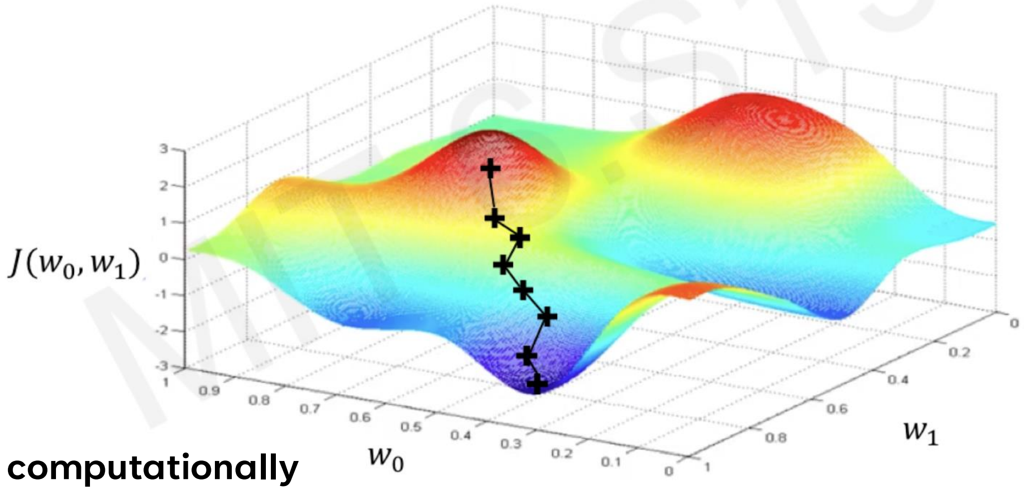
- **Problem:** Imagine you are building a deep learning model for image classification (Millions of images), you cannot feed all these images directly to the model and start training on all these images at once (computing the loss function), even if you have a solid/strong computer, this would be challenging if not impossible to train the model in that way.
- **Solution:** Feeding Mini batches of the initial dataset to the model.

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0,\sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Can be very **computationally intensive** to compute!

Repeat until convergence



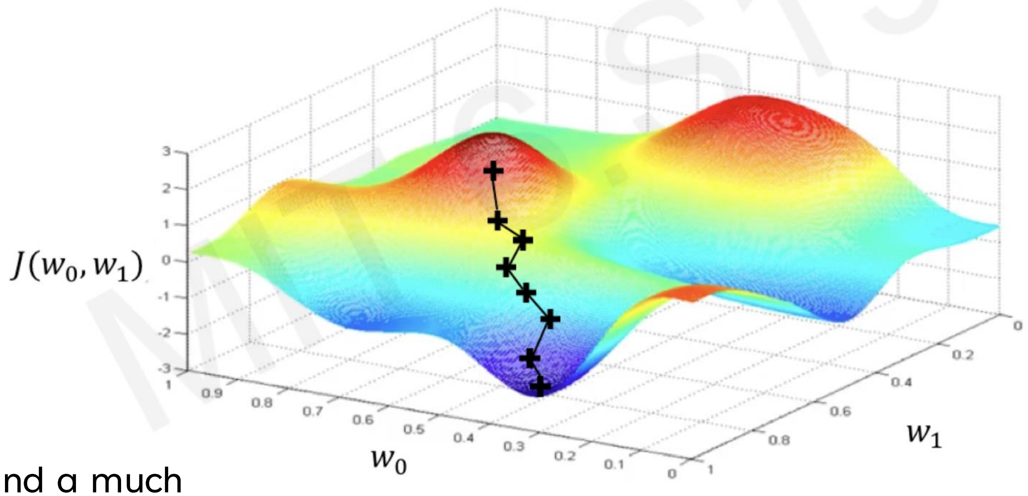
Deep Learning: Stochastic Mini-batch Gradient Descent

Algorithm

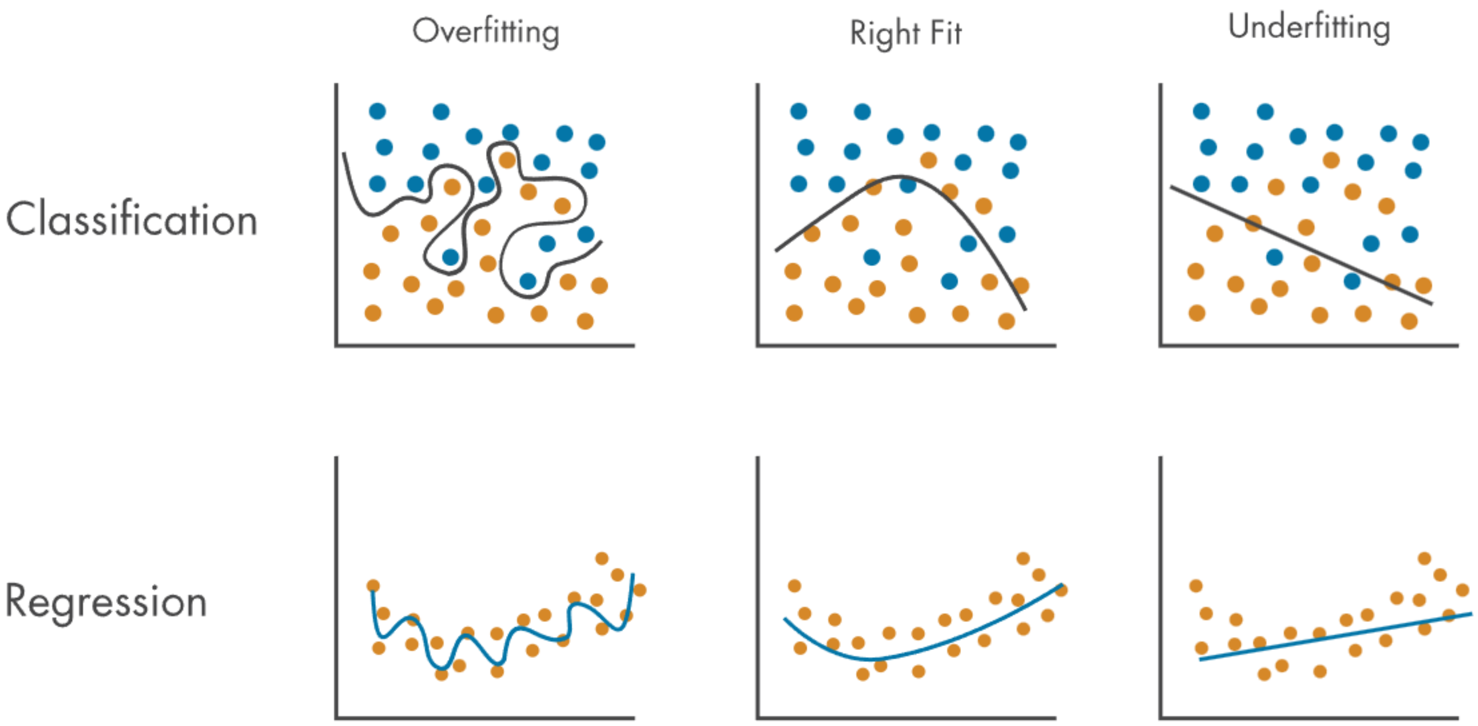
1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \boxed{\frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights

Fast to compute and a much better estimate of the true gradient!

Repeat until convergence

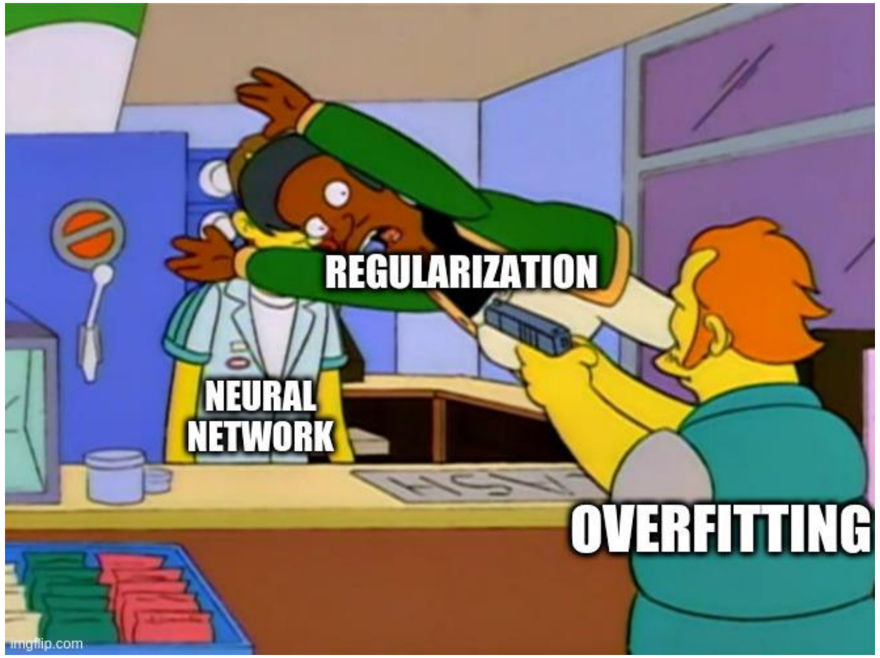


Deep Learning: The Problem of Overfitting



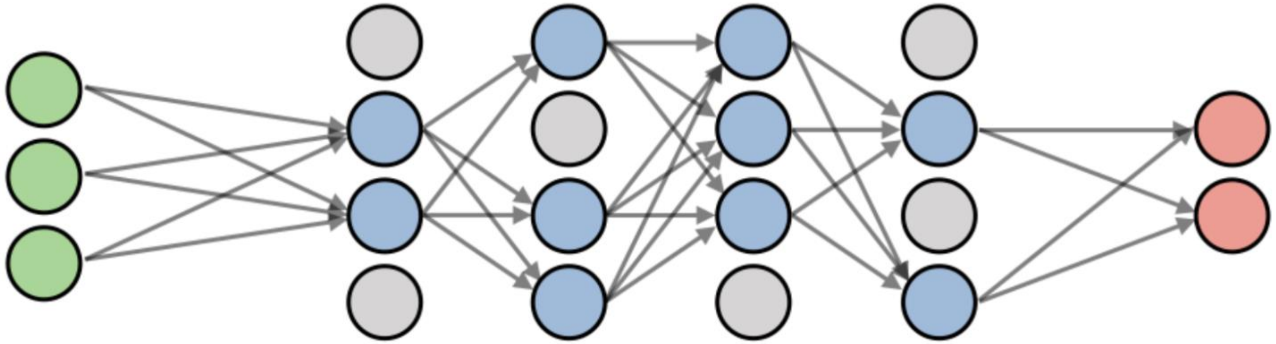
Deep Learning: Regularization

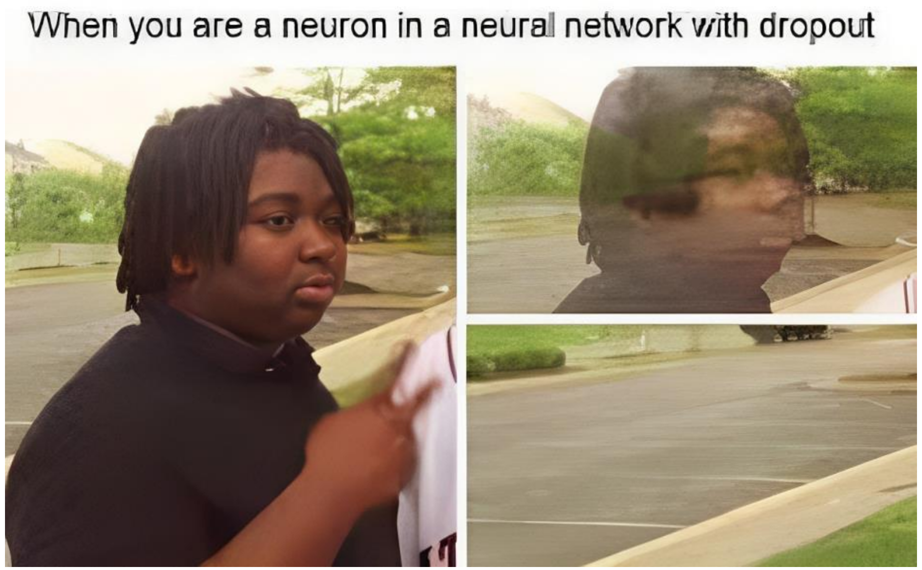
- **What is Regularization?**
  - Technique that constraints our optimization problem to discourage complex models
- **Why do we need it ?**
  - To improve generalization of our model on unseen data



Regularization: Dropout

- Dropout is one of the simplest techniques used to regularize the models and prevent overfitting.
- The idea is simple, you set randomly some neurons(activations) to 0.

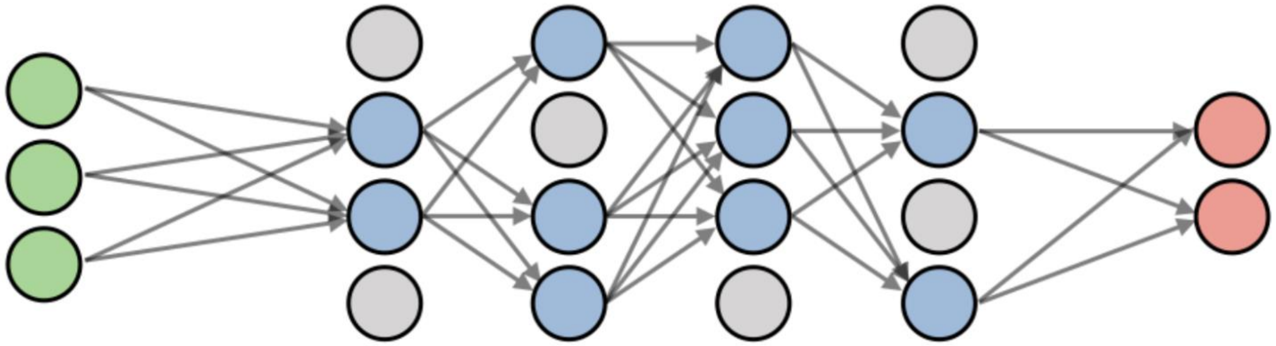




Regularization: Dropout

- During training, randomly set some activations to 0
  - Typically, 'drop' 50% of neurons/activations in a layer
  - Forces the network to not rely on any node

```
tf.keras.layers.Dropout(p=0.5)
```



centralelille

Deep Learning: Frameworks

PyTorch

Keras

scikit learn

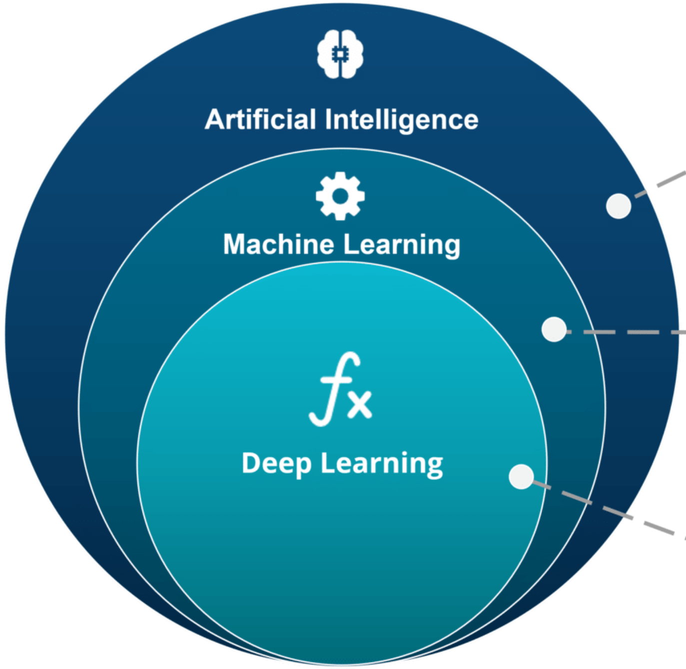
TensorFlow

Deep Learning: Perceptron and Artificial Neural Network <DEMO>

DEMO: Perceptron and Artificial Neural Networks

From Classical Machine Learning to Deep Learning

From Classical Machine Learning to Deep Learning

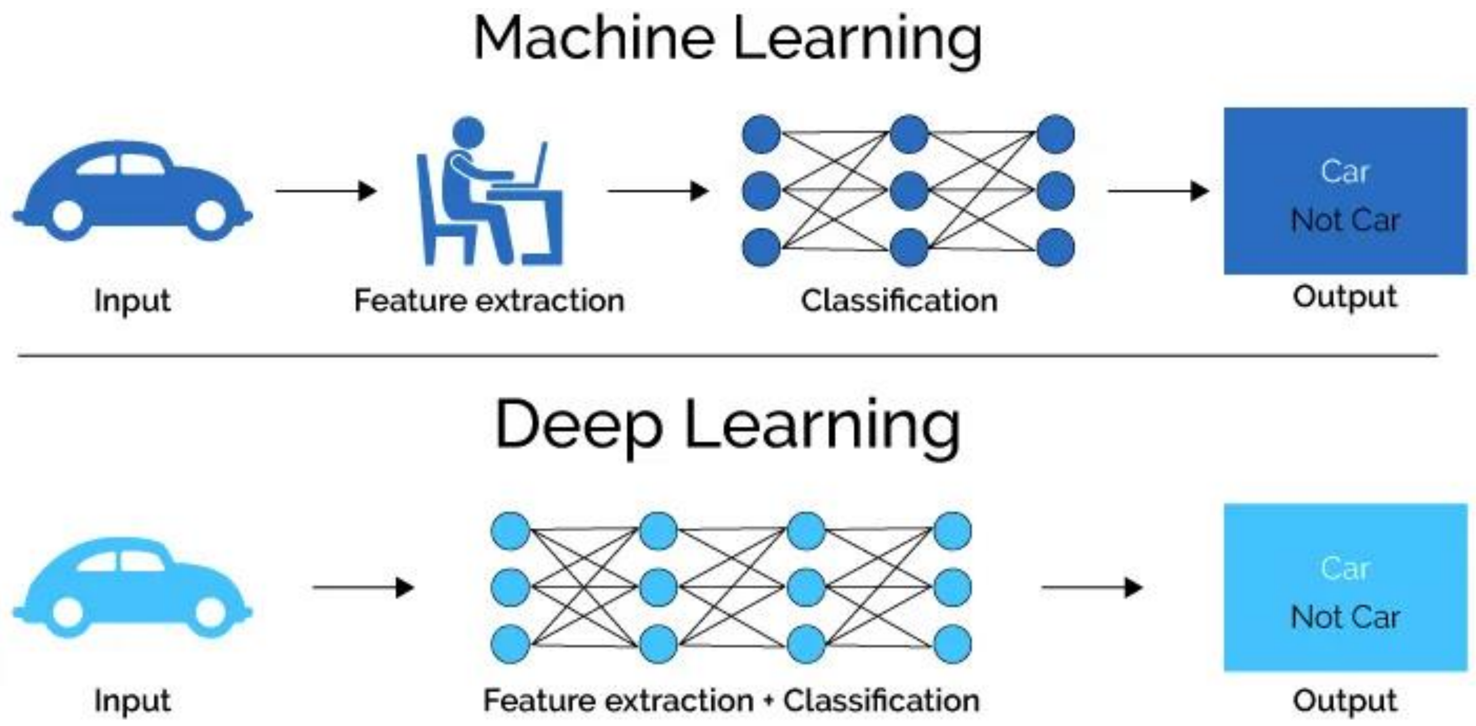


ARTIFICIAL INTELLIGENCE
A technique which enables machines to mimic human behaviour

MACHINE LEARNING
Subset of AI technique which use statistical methods to enable machines to improve with experience

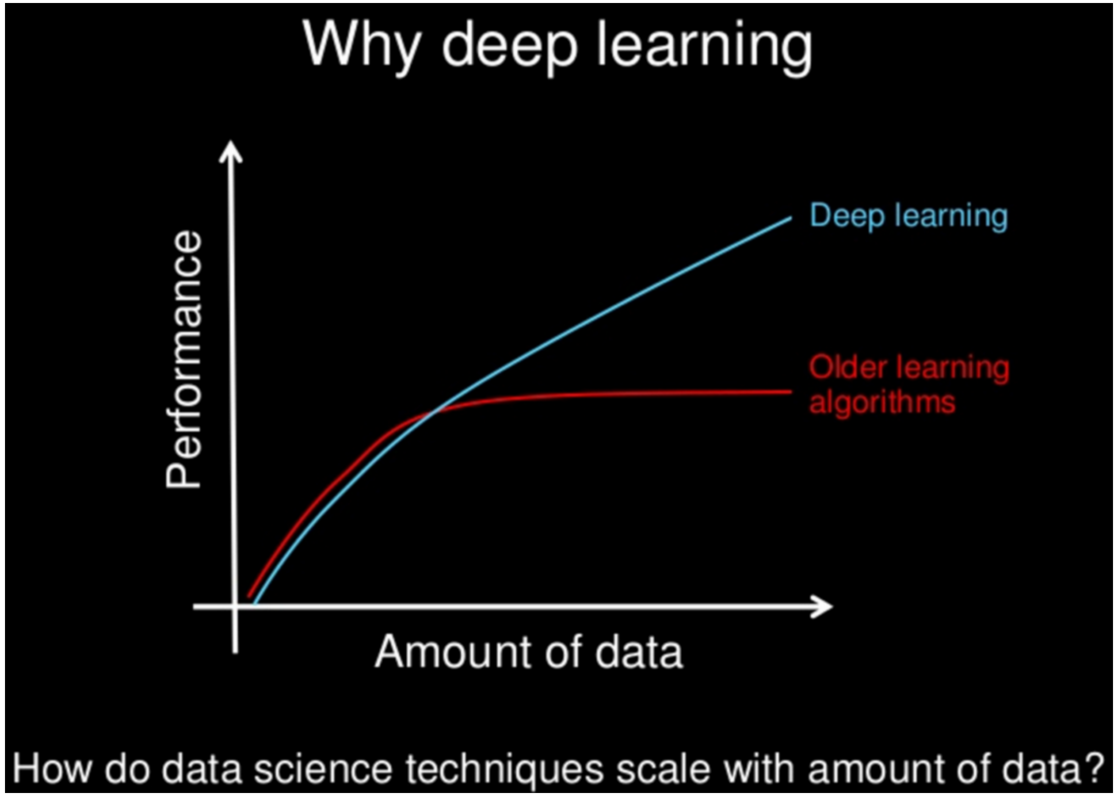
DEEP LEARNING
Subset of ML which make the computation of multi-layer neural network feasible

From Classical Machine Learning to Deep Learning



Feature extraction: Machine learning vs Deep learning

From Classical Machine Learning to Deep Learning



From Classical Machine Learning to Deep Learning

Aspect	Classical machine learning	Deep learning
Data size	Can perform well with smaller datasets.	Often requires large amounts of data for effective training.
Feature Engineering	Often requires domain knowledge for manual feature creation.	Automatically learns hierarchical features.
Computation	Computationally less intensive compared to deep learning.	Requires powerful hardware, like GPUs, for efficient training.
NLP + Computer vision + Audio/Speech recognition	Limited and less effective in these tasks	Highly effective and represents state-of-the-art for these tasks
Human-like Learning	Relies on human-engineered features and rules.	Can automatically learn complex patterns like humans.

Table. The comparative table between classical machine learning (ML) and deep learning (DL) based on various aspects

From Classical Machine Learning to Deep Learning: Examples and Case Studies

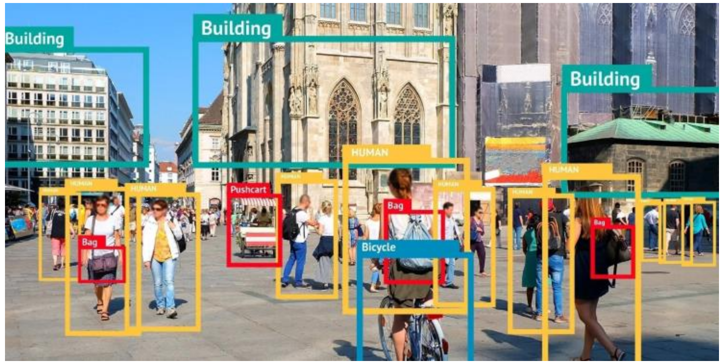
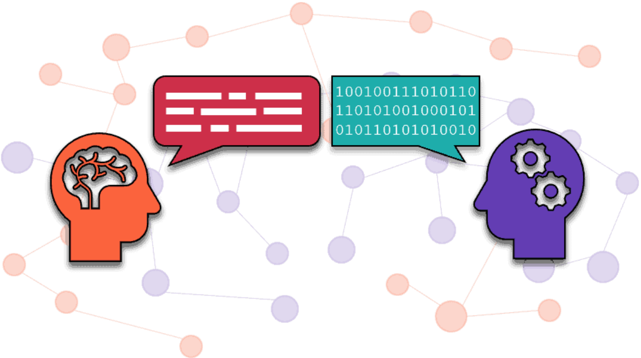


Image Recognition and Computer Vision

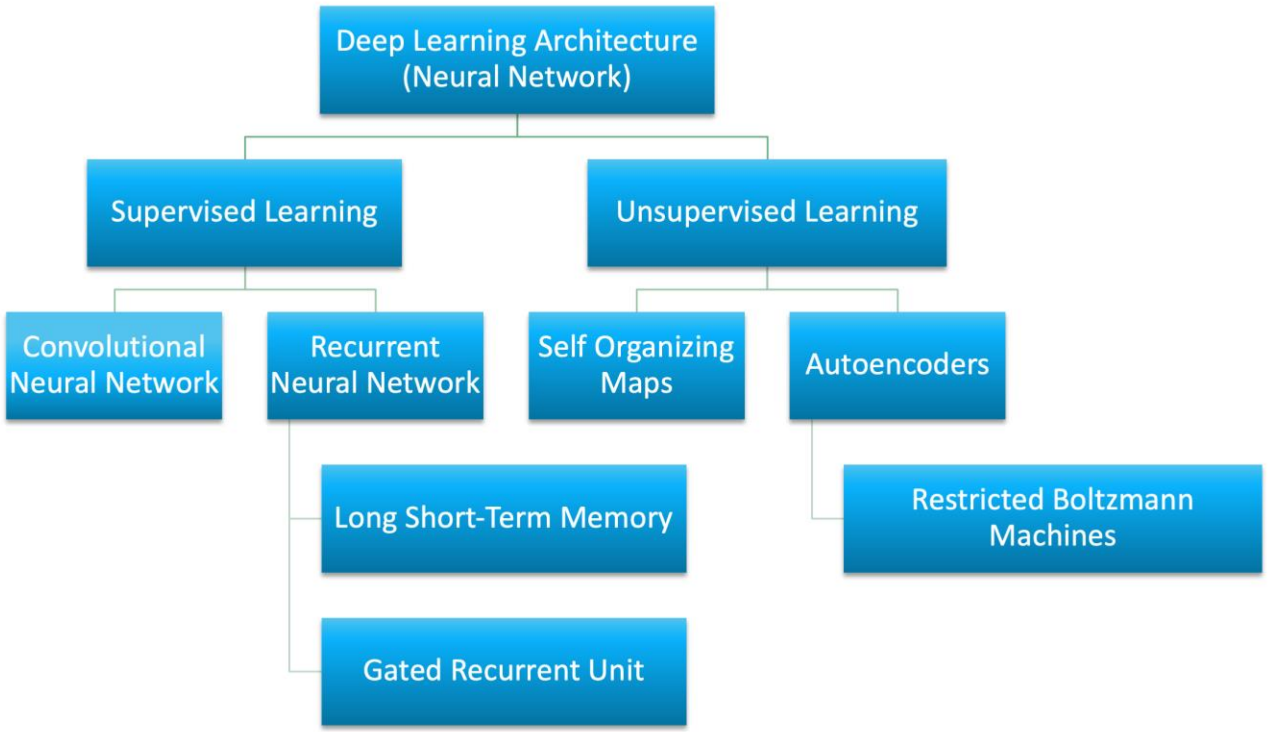


Natural Language Processing (NLP)

Speech Recognition

Healthcare (DeepMind's AlphaFold), Game Playing, Recommendation System, Robotics, ...etc.

Deep Learning: Architectures



Source: https://developer.ibm.com/articles/cc-machine-learning-deep-learning-architectures/

Deep Learning: TO-DO Project

Classification using ANN: Heart Disease Prediction