电子科技大学
格拉斯哥学院
Glasgow College, UESTC

Deep Learning & Neural Network

Supervised Learning

- Computation methods used in the supervised learning process
  - Regression
  - Naïve Bayes
  - K-Nearest Neighbors
  - Support Vector Machines
  - Decision Tree
  - Neural Networks

Deep Learning

What is Deep Learning?

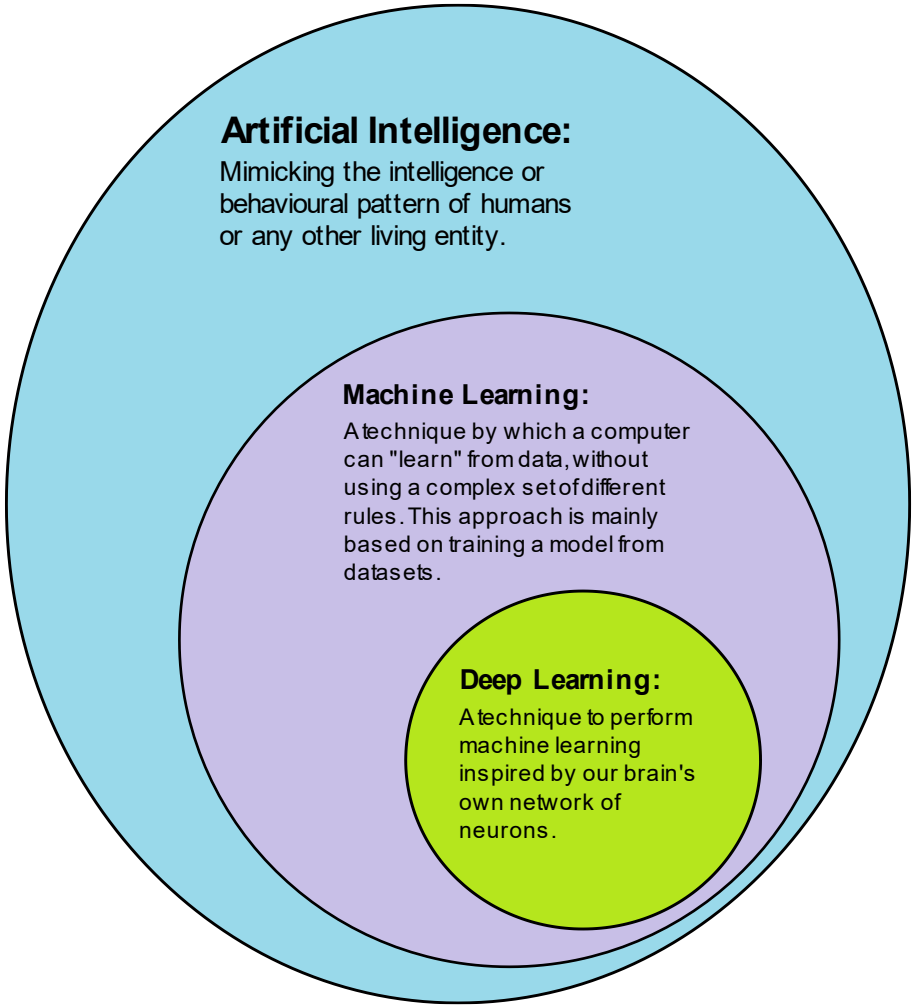
“Deep learning attempts to mimic the human brain—albeit far from matching its ability—enabling systems to cluster data and make predictions with incredible accuracy.”

Part of the Bigger Picture

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example.

A computer model learns to perform classification tasks directly from images, text, or sound.

Models are trained by using a large set of labeled data and neural network architectures that contain many layers.

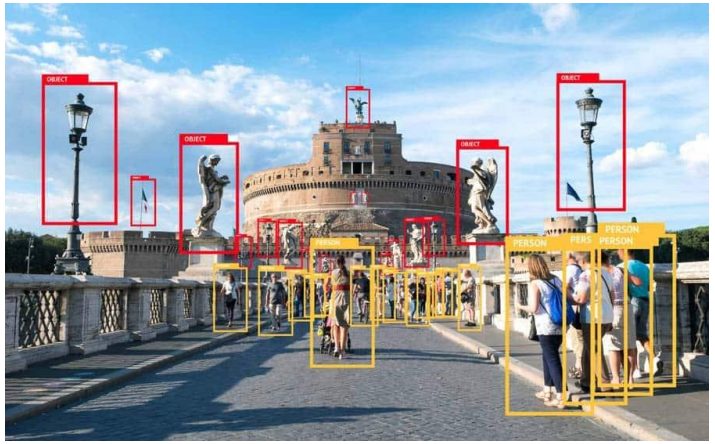


Why now?

- There are two main reasons DL has only recently become useful:
  1) Deep learning requires large amounts of labelled data.
  2) Deep learning requires substantial computing power.
- Examples:
  - Autonomous Driving
  - Objects identify/classification
  - Automatically detection of cancer and other diseases
  - Automated hearing and speech translation

Deep Learning Applications

- Face & Object Recognition



- Bioinformatics & Drug Discovery

- Natural Language & Speech

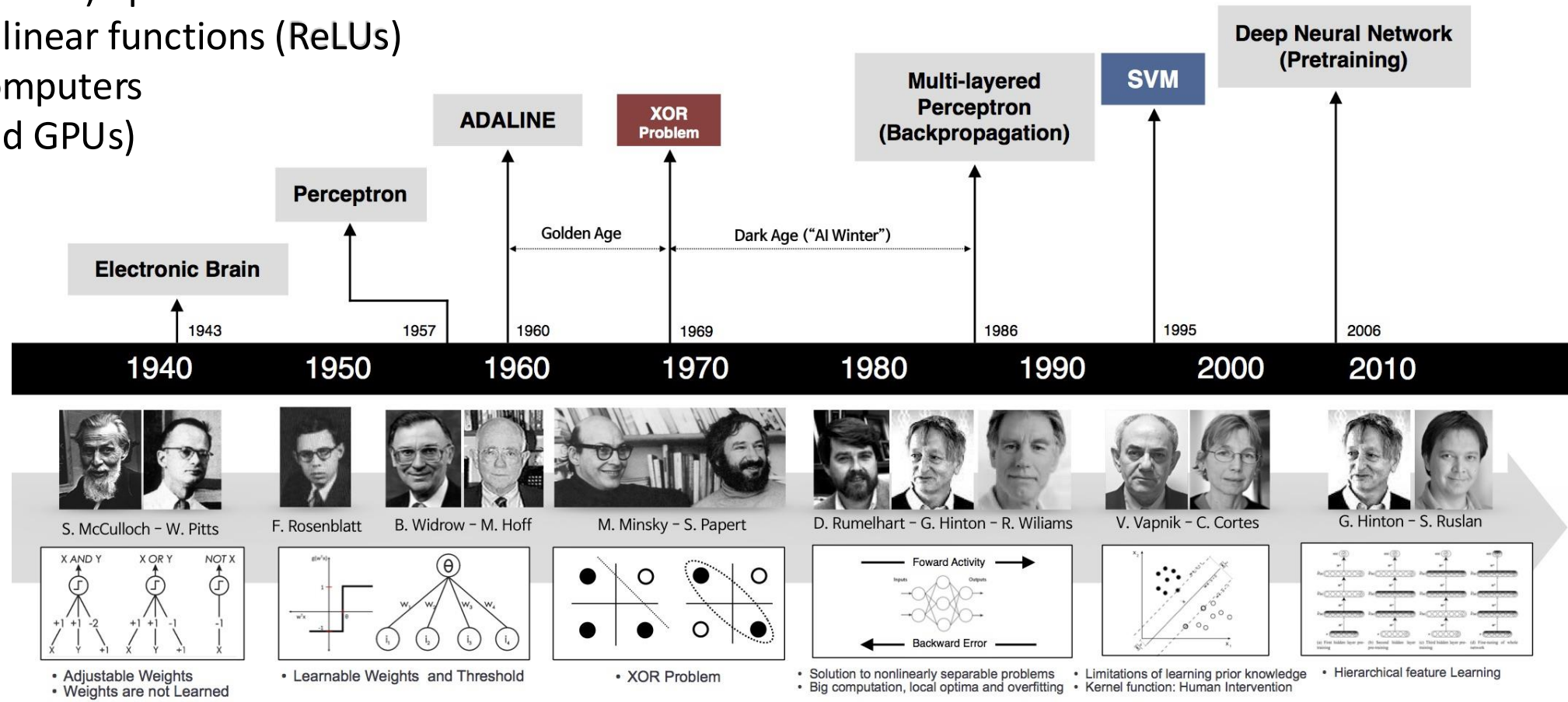
- Fraud Detection

Why deep learning is so Popular?

This wasn't always the case!

Since 1980s: Form of models hasn't changed much, but lots of new tricks…

- More hidden units
- Better (online) optimization
- New nonlinear functions (ReLUs)
- Faster computers
- (CPUs and GPUs)

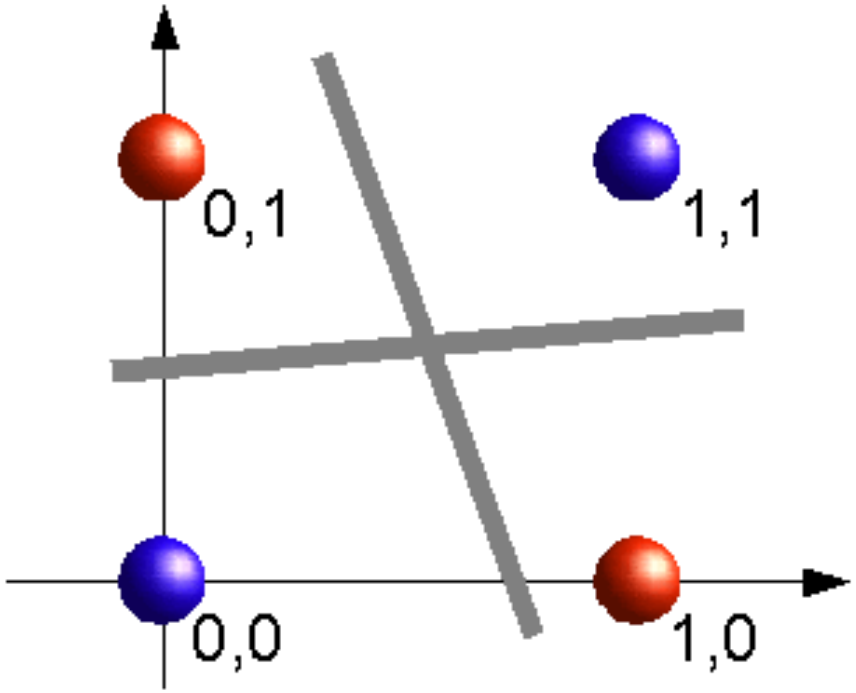


Why deep learning is so Popular?

- Has won numerous pattern recognition competitions.
- Does so with minimal feature engineering.
- A lot of money is invested in it:
  - DeepMind: Acquired by Google for $400 million
  - DNNResearch: Three person startup (including Geoff Hinton) acquired by Google for unknown price tag
  - Enlitic, Ersatz, MetaMind, Nervana, Skylab:
    Deep Learning startups commanding millions of VC dollars

Learning Highly Non-Linear Functions

- We have a mapping $y = F(x)$
- F might be non-linear function
- x (vector of) continuous and/or discrete vars
- y (vector of) continuous and/or discrete vars



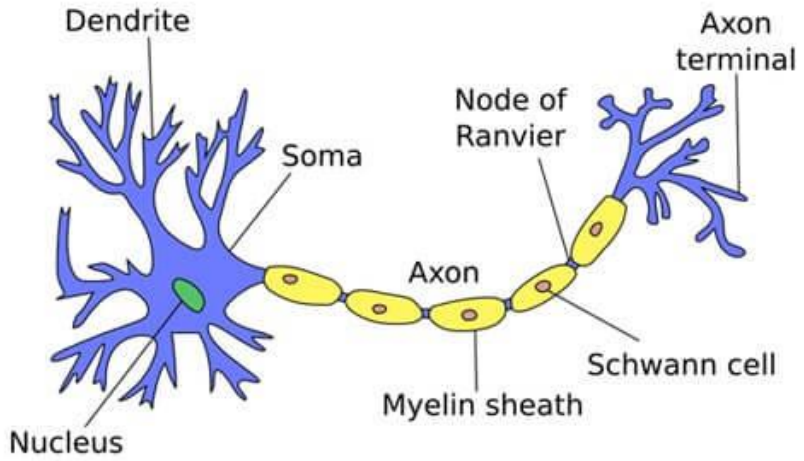
XOR gate:
You can't split 0 (red) and 1 (blue) with a single straight line

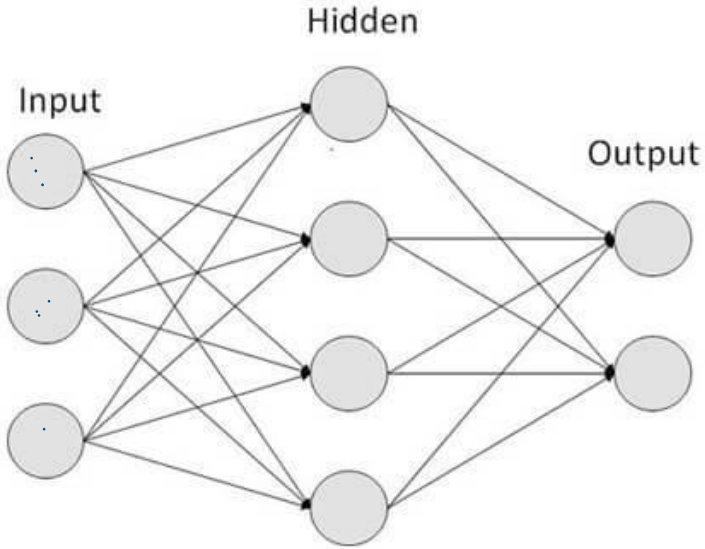
Learning Highly Non-Linear Functions

- In SVM, we solve such problem by applying Kernel function.
- An alternative approach is to use a fixed number of basis function in which the parameter values are adapted during training.
- The most successful model of this type in the context of pattern recognition is the feedforward neural network, also known as the multilayer perceptron.

Perceptron and Neural Nets

- From biological neuron to artificial neuron (perceptron)

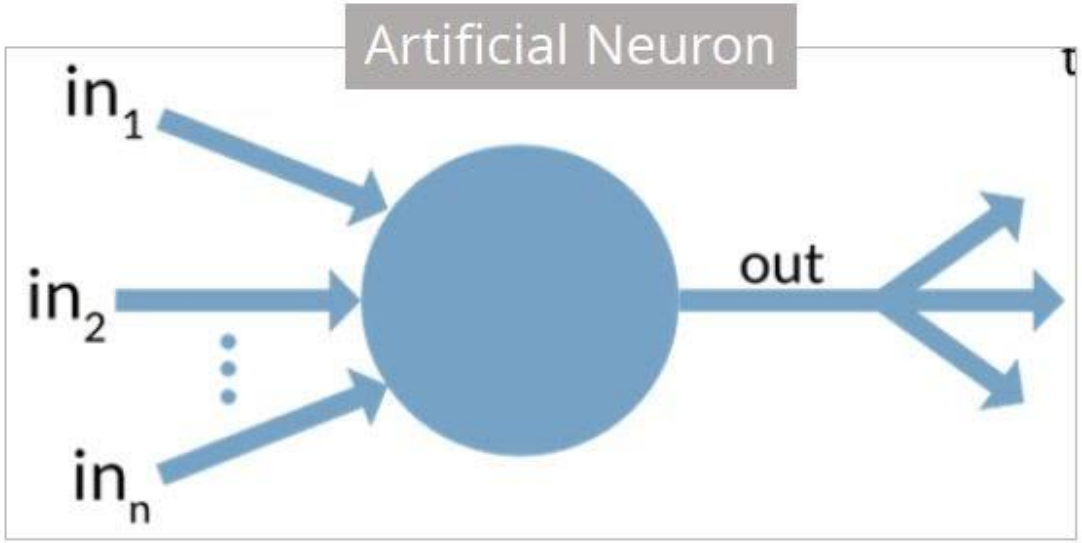




Artificial Neuron

Artificial Neuron

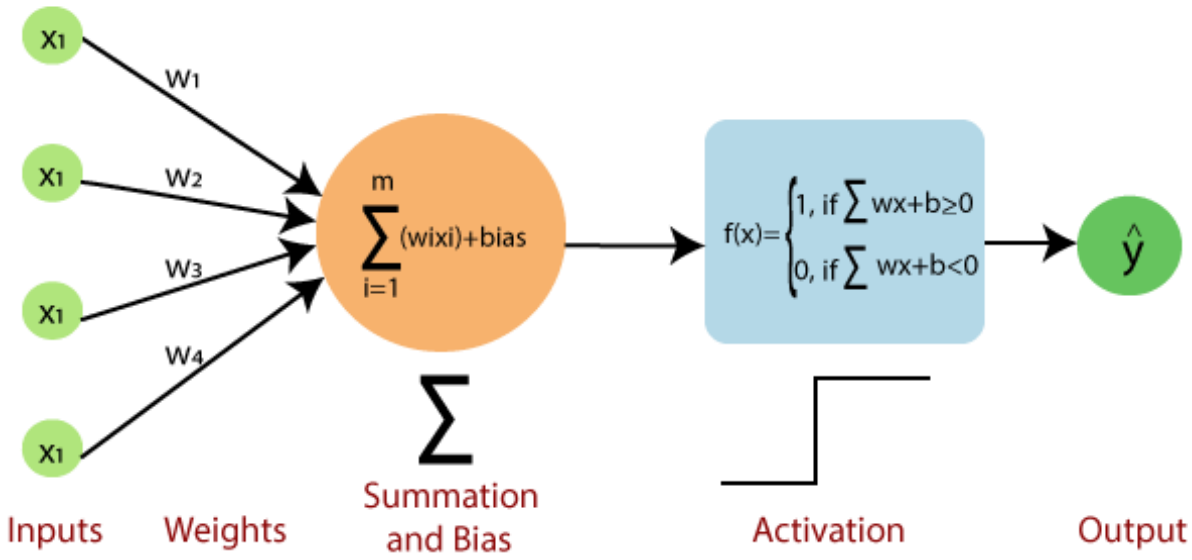
- An artificial neuron is a mathematical function based on a model of biological neurons.
- Each neuron
  - takes inputs,
  - weighs them separately,
  - sums them up,
  - passes this sum through a nonlinear function to produce output.



Source: https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron

Perceptron

- A Perceptron is an algorithm for supervised learning of binary classifiers.
- This algorithm enables neurons to learn and processes elements in the training set one at a time.



Components of Perceptron

- Input Layer: the input layer consists of one or more input neurons, which receive input signals from the external world or other layers of the neural network.
- Weights: each input neuron is associated with a weight, representing the strength of the connection between the input and output neurons.
- Bias: a bias term is added to the input layer to give the perceptron additional flexibility in modelling complex patterns in the input data.
- Activation Function: the activation function determines the perceptron's output based on the weighted sum of the inputs and the bias term. Common activation functions used in perceptrons include the step, sigmoid, and ReLU functions.

Components of Perceptron

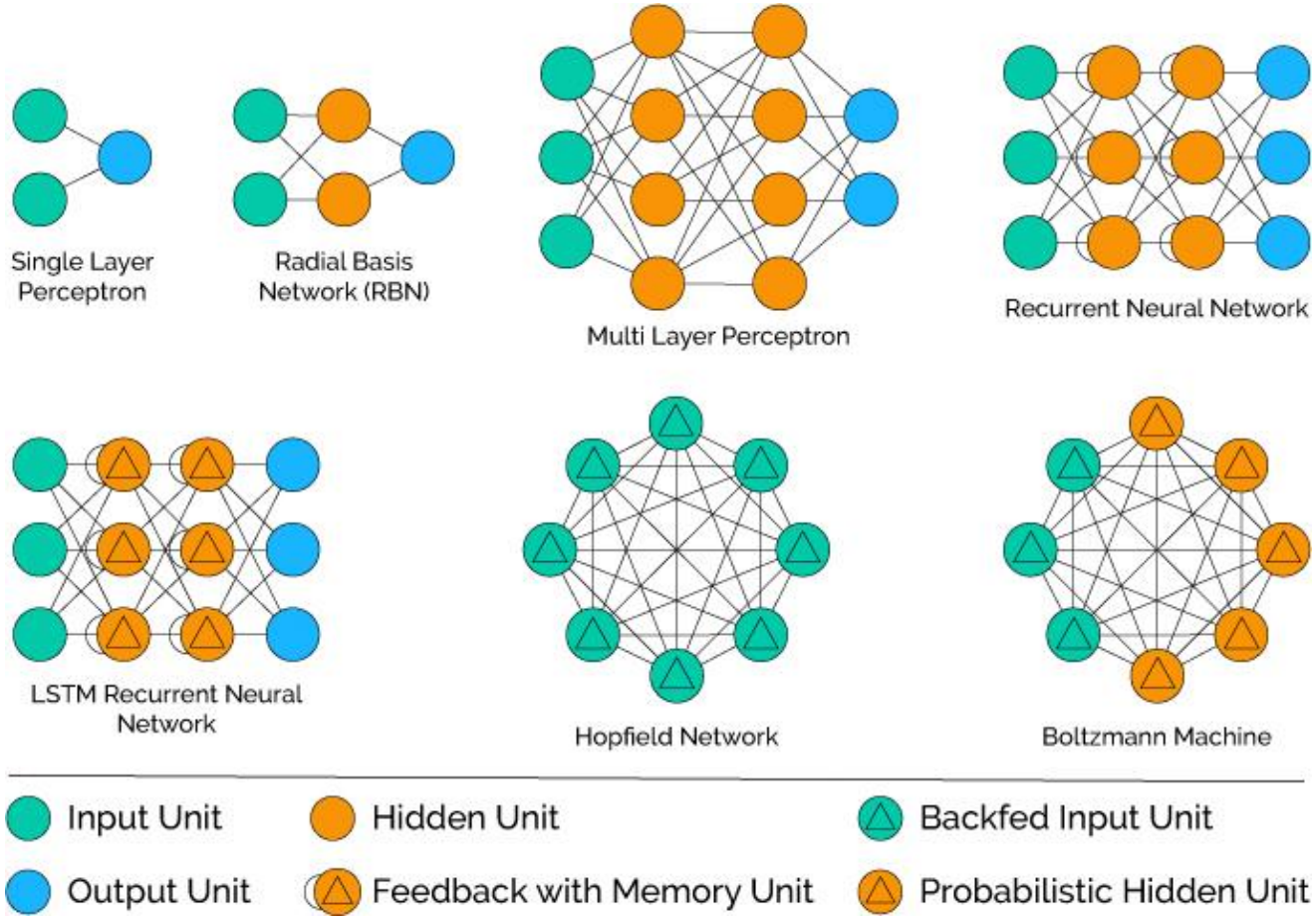
- Output: the output of the perceptron is a single binary value, either 0 or 1, which indicates the class or category to which the input data belongs.
- Training Algorithm: The perceptron is typically trained using a supervised learning algorithm such as the perceptron learning algorithm or backpropagation. During training, the weights and biases of the perceptron are adjusted to minimize the error between the predicted output and the true output for a given set of training examples.
- Overall, the perceptron is a simple yet powerful algorithm that can be used to perform binary classification tasks and has paved the way for more complex neural networks used in deep learning today.

Neural Networks Variations

- There are many types of artificial neural networks, each with their unique strengths.
- Different types of neural networks are used for different data and applications.
- They use different principles in determining their own rules.
- The different architectures of neural networks are specifically designed to work on those particular types of data or domain.

Types of Neural Networks

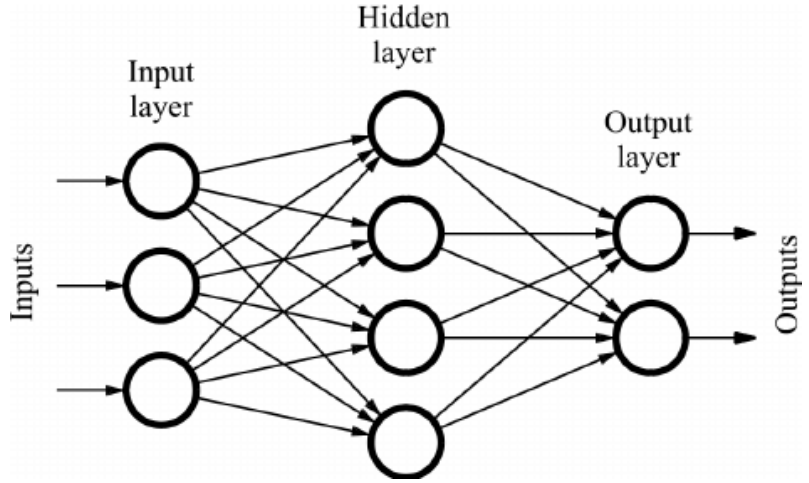
- Feed Forward Neural Network
- Multilayer Perceptron
- Convolutional Neural Network
- Radial Basis Functional Neural Network
- Recurrent Neural Network
- LSTM – Long Short-Term Memory
- Sequence to Sequence Models
- Modular Neural Network
- Graph Neural Network

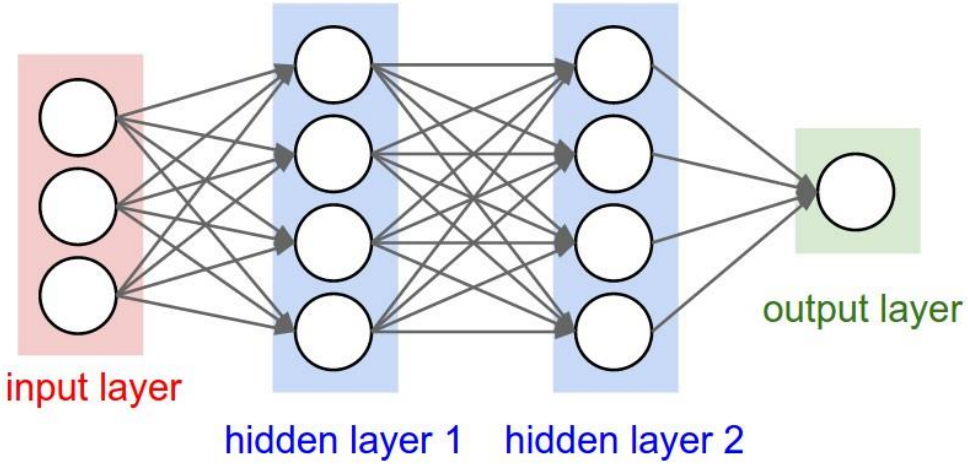


Neural Network

Neural Networks

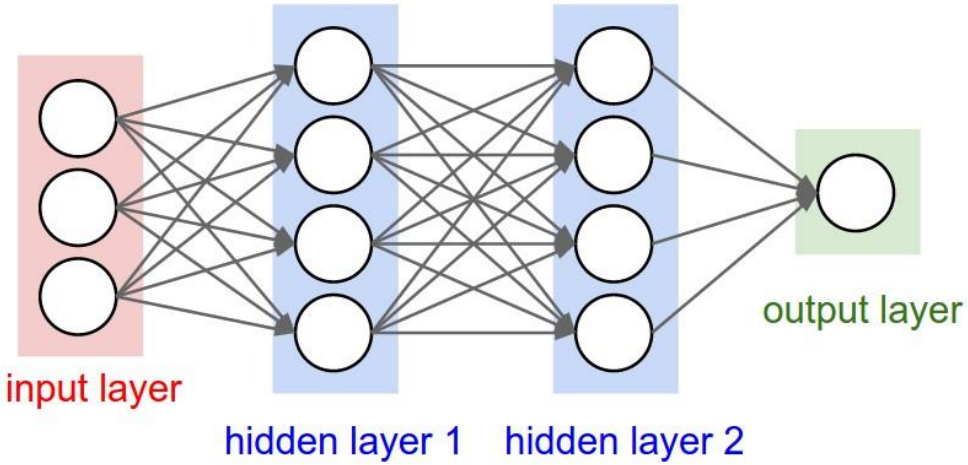
- Neural networks, also known as artificial neural networks (ANNs), or Feedforward Neural Networks (FNN), are a subset of machine learning and are at the heart of deep learning algorithms.
- Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.



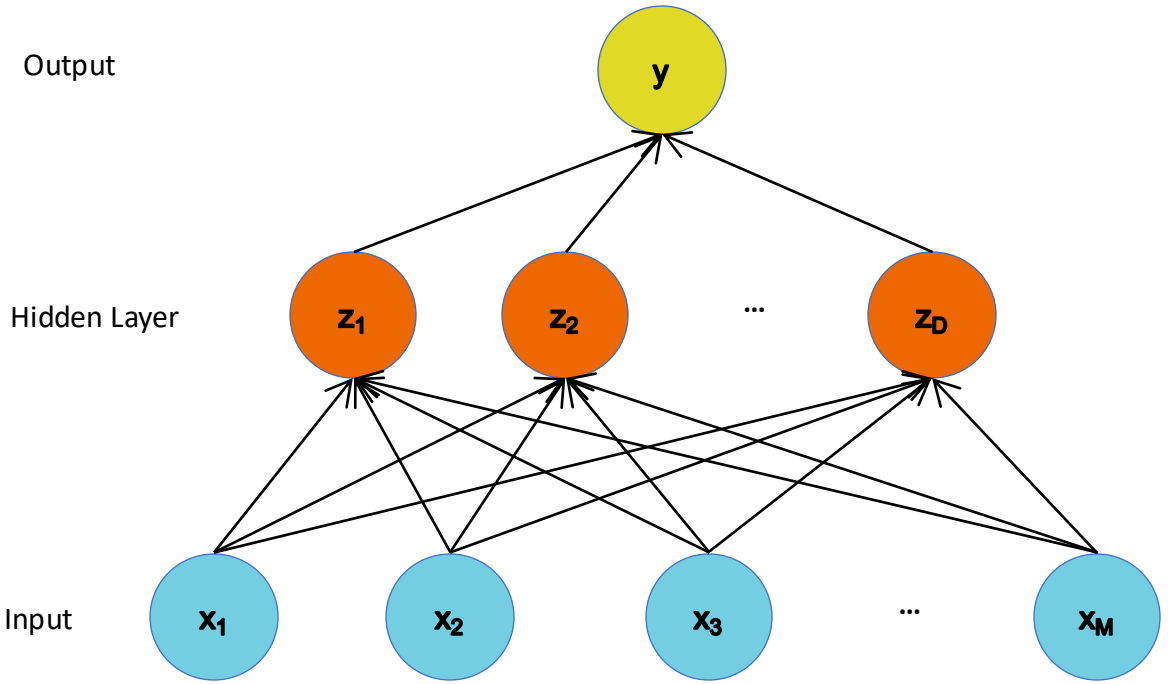


Network Structure

- NN/FNN comprises several layers containing an input layer, one or more hidden layers, and an output layer.
- Each node, or artificial neuron, connects to another and has an associated weight and threshold.
- If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network.
- Otherwise, no data is passed along to the next layer of the network.



Feedforward Networks



$$y = \hat{\sigma}(b)$$

$$b = \sum_{j=0}^{D} \widehat{w_j} z_j$$

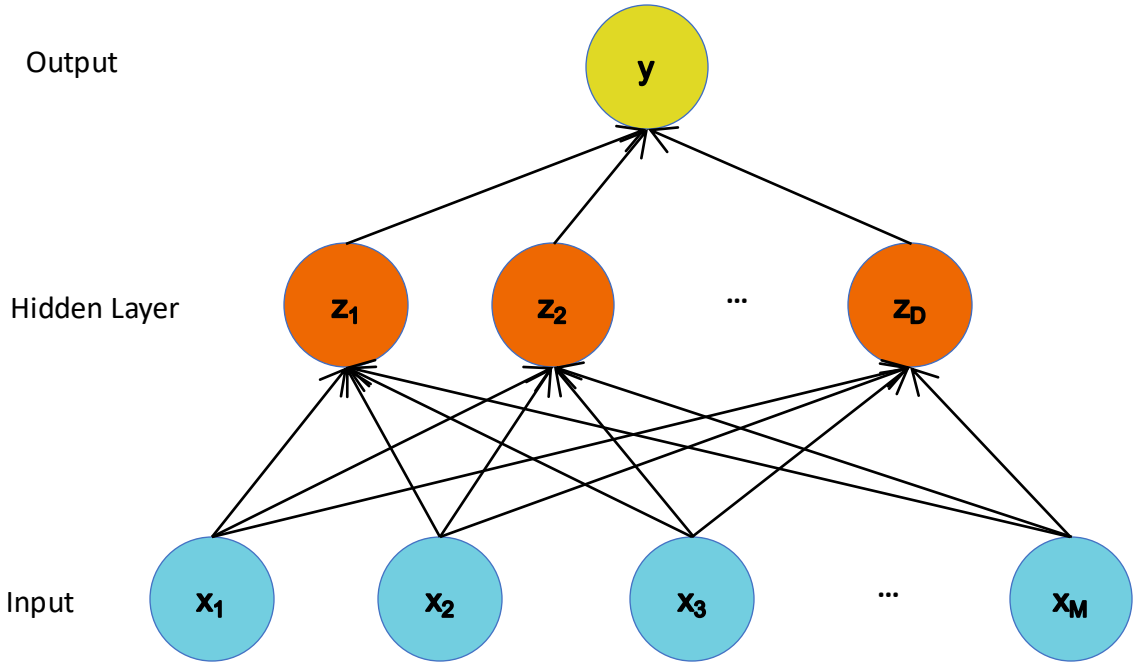
$$z_j = \sigma(a_j)$$

$$a_j = \sum_{i=0}^{M} w_{ji} x_i$$

$$x_i$$

Source: https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network

Feedforward Networks



$$y = \hat{\sigma}(b)$$

$$b = \sum_{j=0}^{D} \hat{w}_j z_j$$

$$z_j = \sigma(a_j)$$

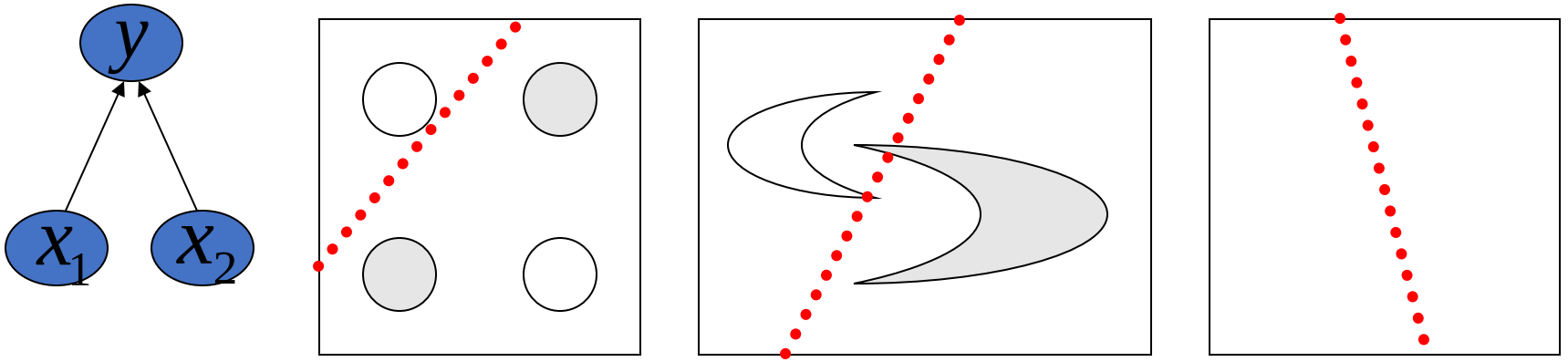
$$a_j = \sum_{i=0}^{M} w_{ji} x_i$$

$$x_i$$

σ and $\hat{\sigma}$ are activation functions

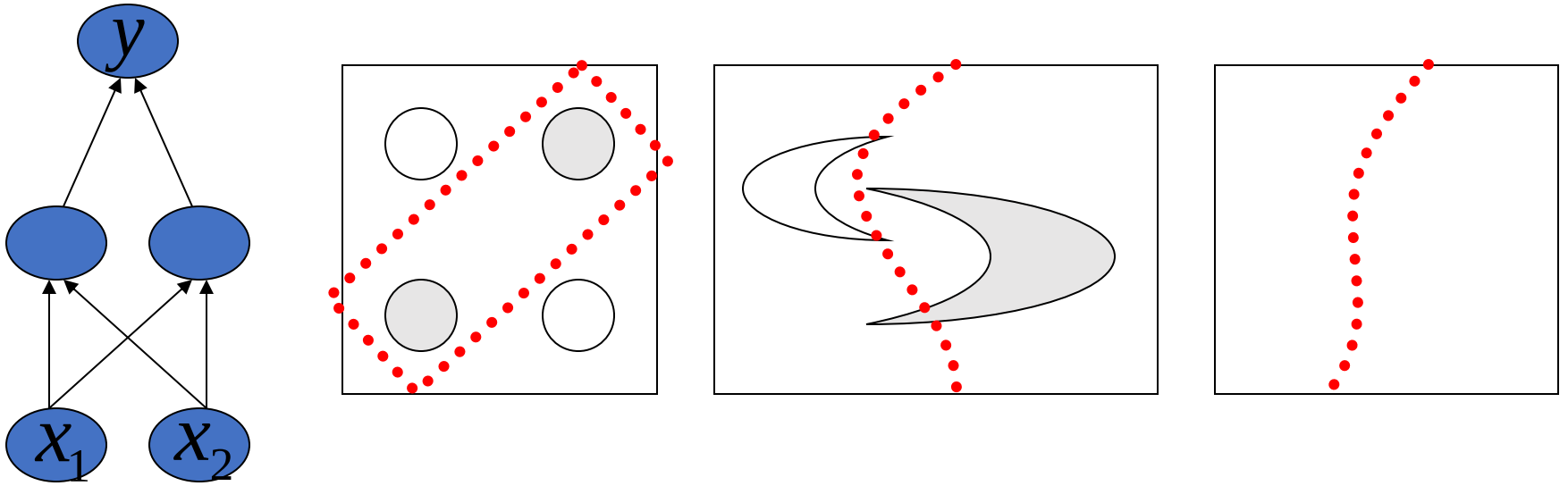
Decision Boundary

- 0 hidden layers: linear classifier (Hyperplanes)



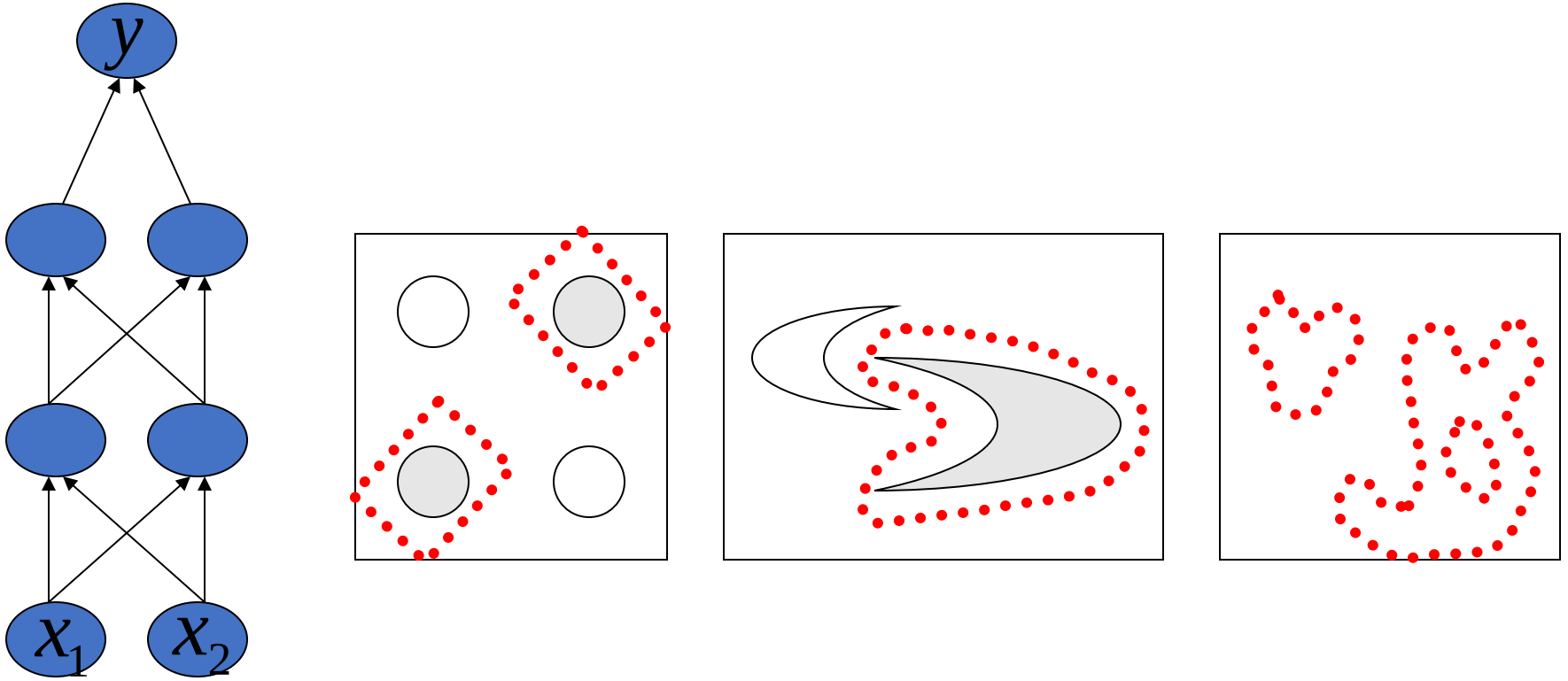
Decision Boundary

- 1 hidden layer: Boundary of convex region (open or closed)



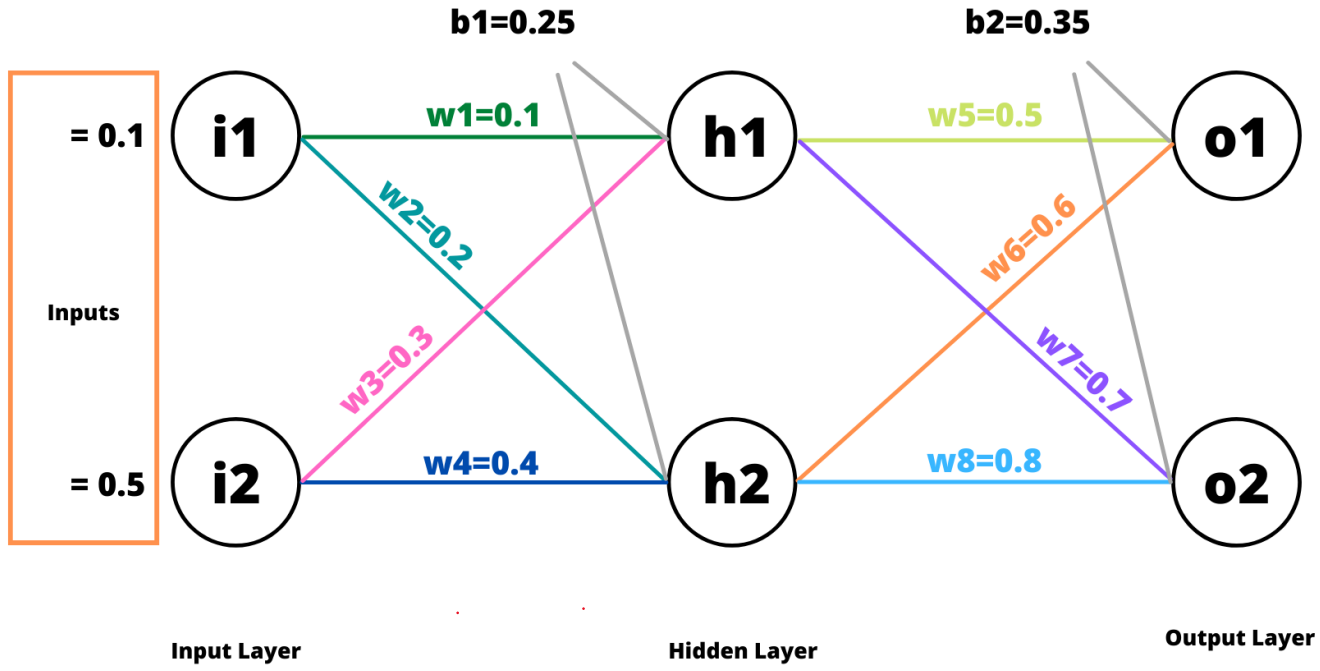
Decision Boundary

- 2 hidden layers: Combinations of convex regions

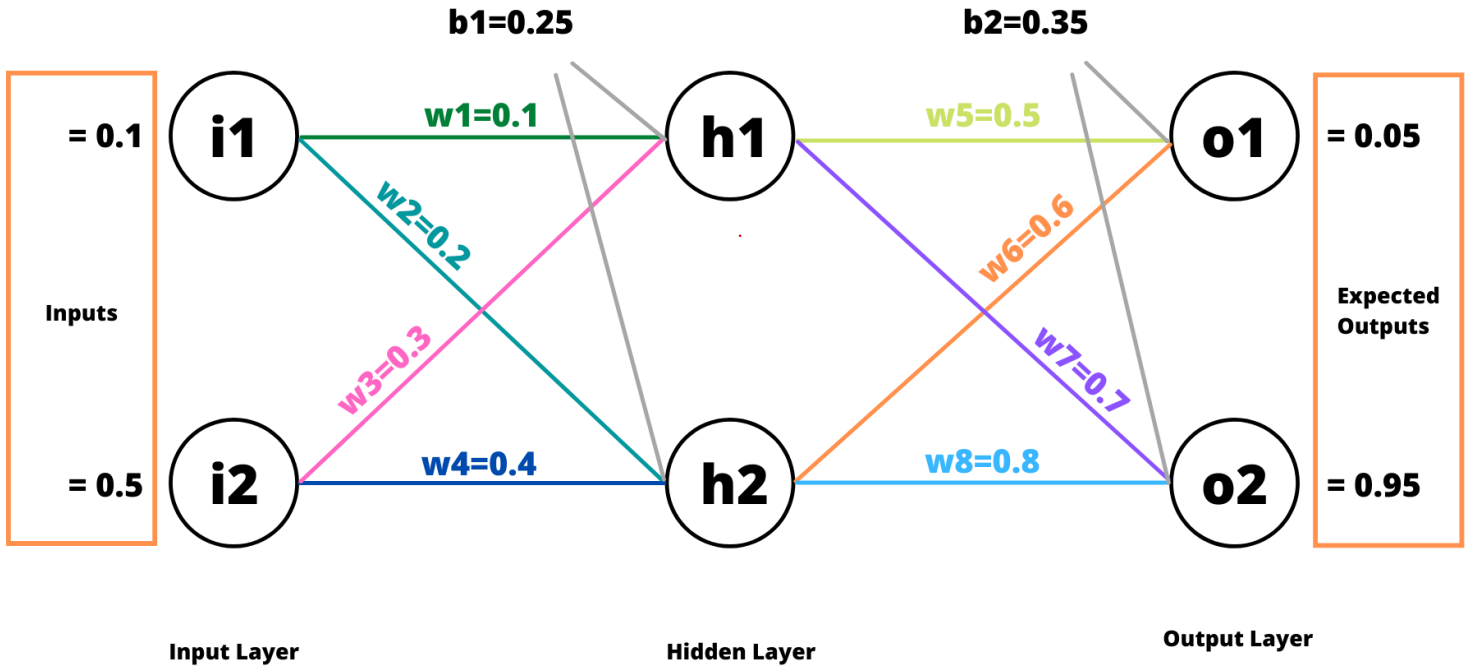


Example 1

- Use the given inputs and weights to calculate the outputs O1 and O2.



Example 1 - Solution



For **h1:**

$$sum_{h1} = i_1 * w_1 + i_2 * w_3 + b_1$$

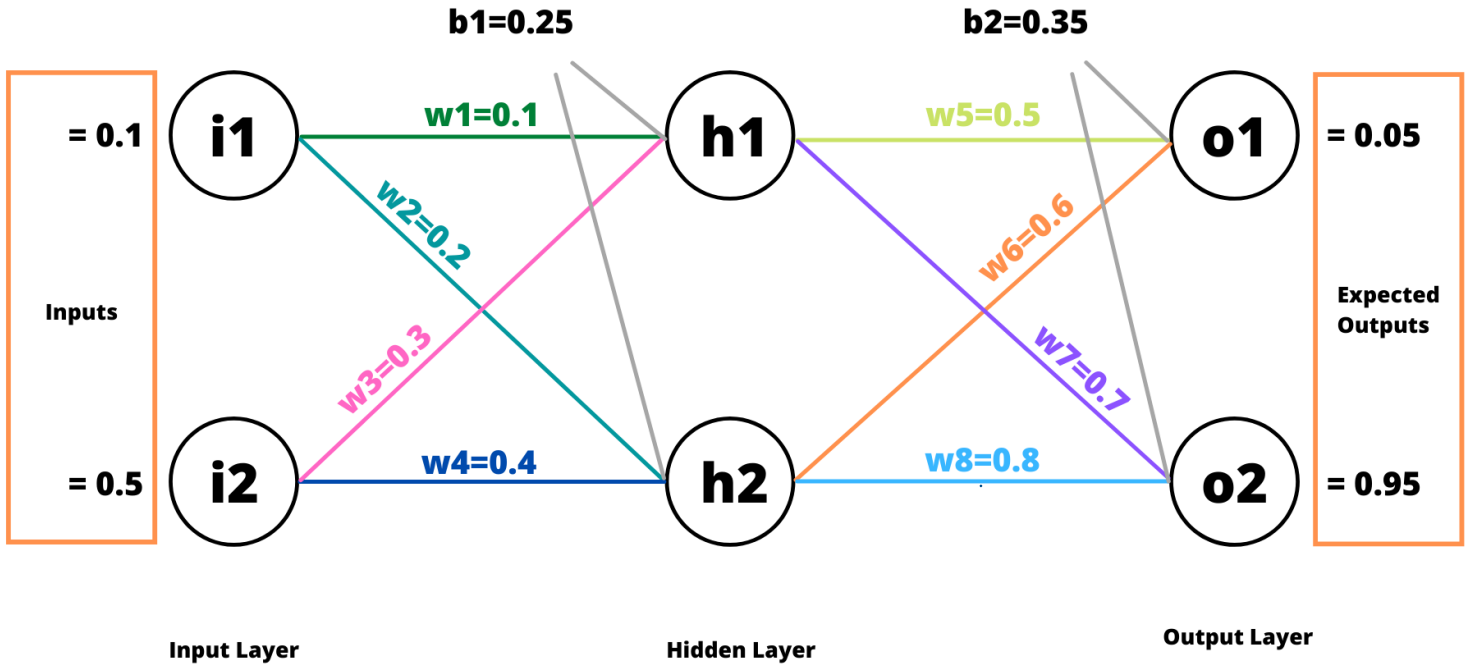
$$sum_{h1} = 0.1 * 0.1 + 0.5 * 0.3 + 0.25 = 0.41$$

For **h2:**

$$sum_{h2} = i_1 * w_2 + i_2 * w_4 + b_1 = 0.47$$

$$output_{h2} = \frac{1}{1 + e^{-sum_{h2}}} = 0.61538$$

Example 1 - Solution



For **O1:**

$$sum_{o1} = output_{h1} * w_5 + output_{h2} * w_6 + b_2 = 1.01977$$

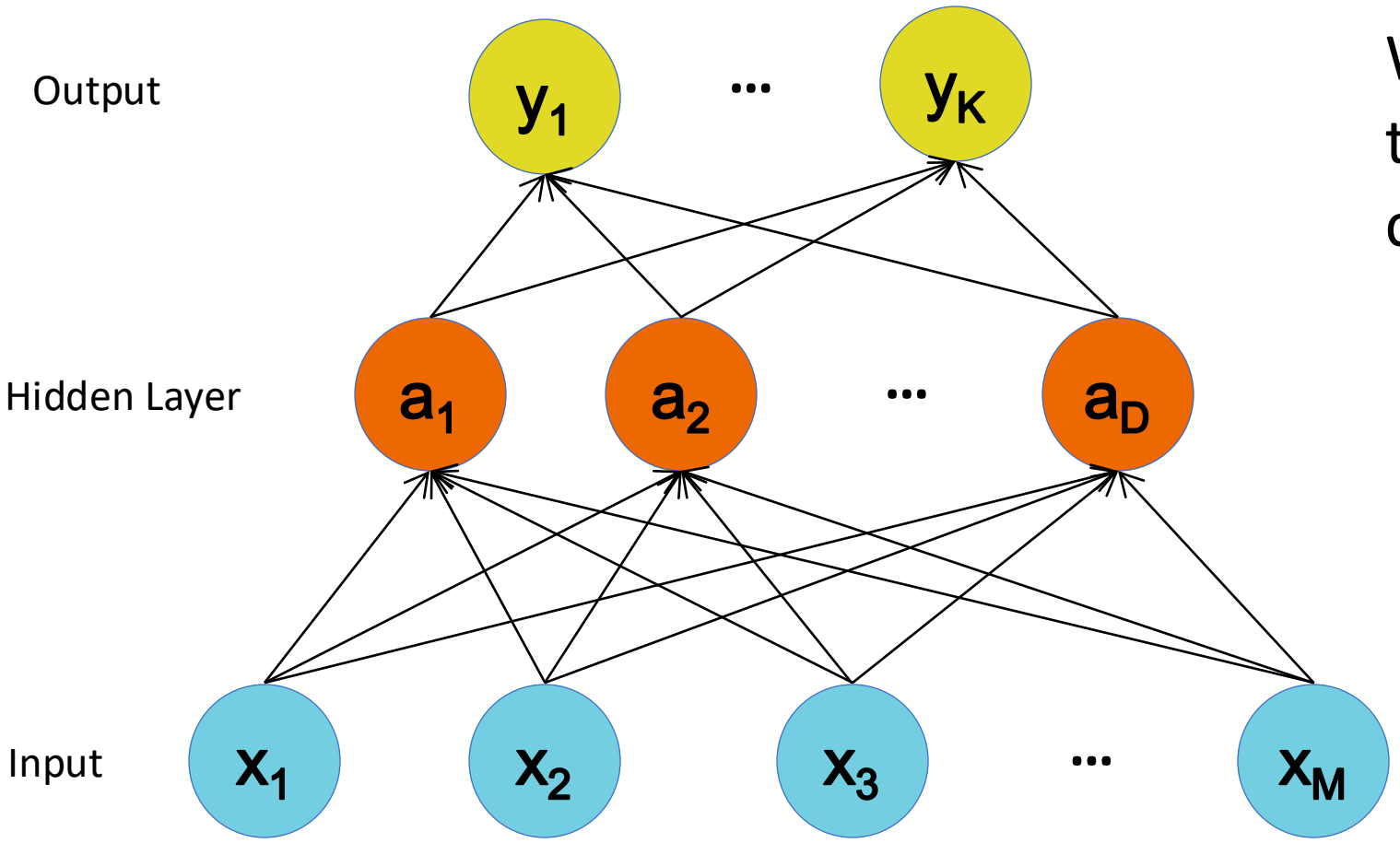
$$output_{o1} = \frac{1}{1 + e^{-sum_{o1}}} = 0.73492$$

For **O2:**

$$sum_{o2} = output_{h1} * w_7 + output_{h2} * w_8 + b_2 = 1.26306$$

$$output_{o2} = \frac{1}{1 + e^{-sum_{o2}}} = 0.77955$$

Multi-Class Output



Where y_i should be the possibility of each category, thus:

$$0 < y_i < 1$$

$$\sum y_i = 1$$

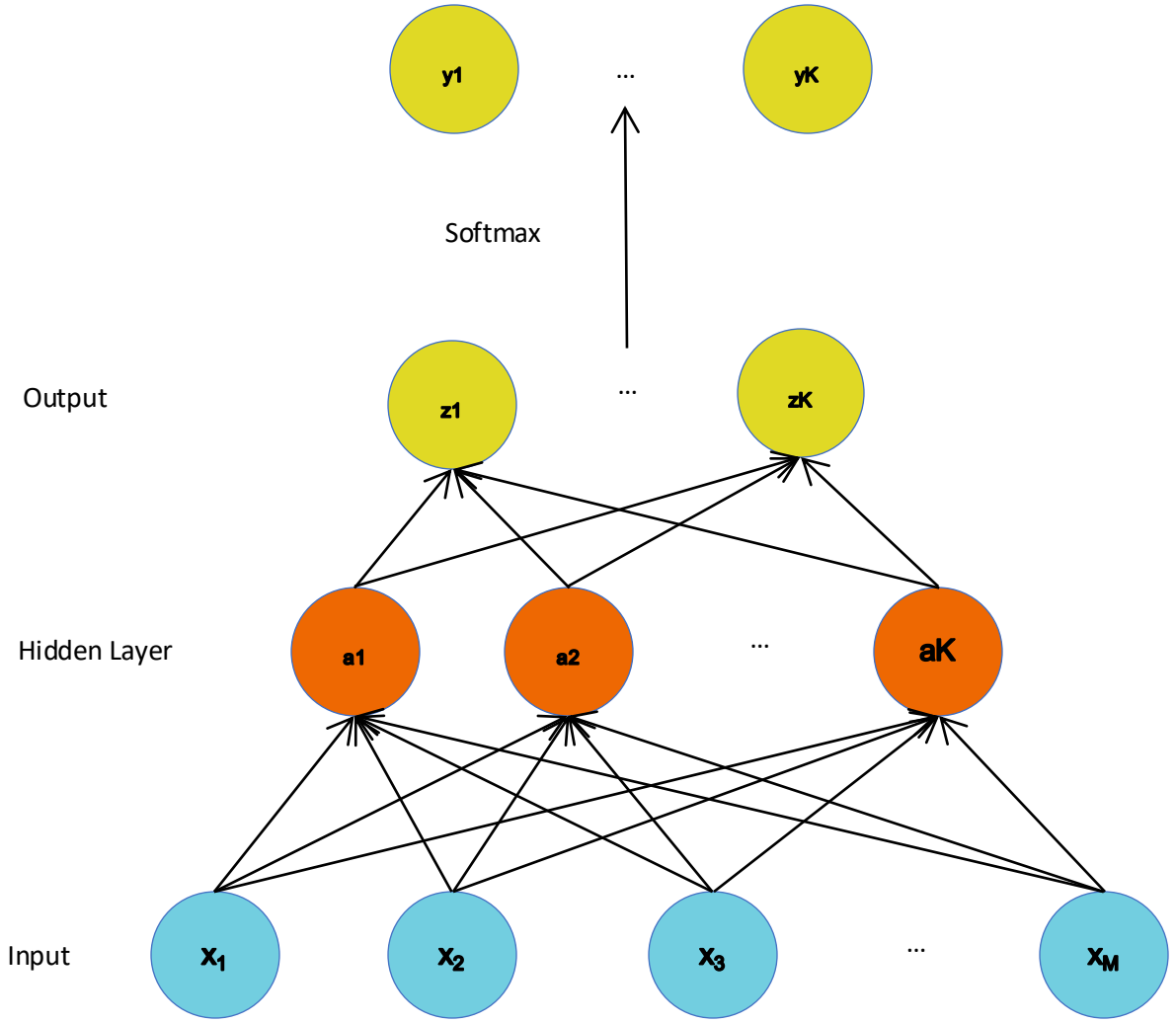
Multi-Class Output

- Softmax activation function:

$$y_i = \sigma(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

$$0 < y_i < 1$$

$$\sum y_i = 1$$

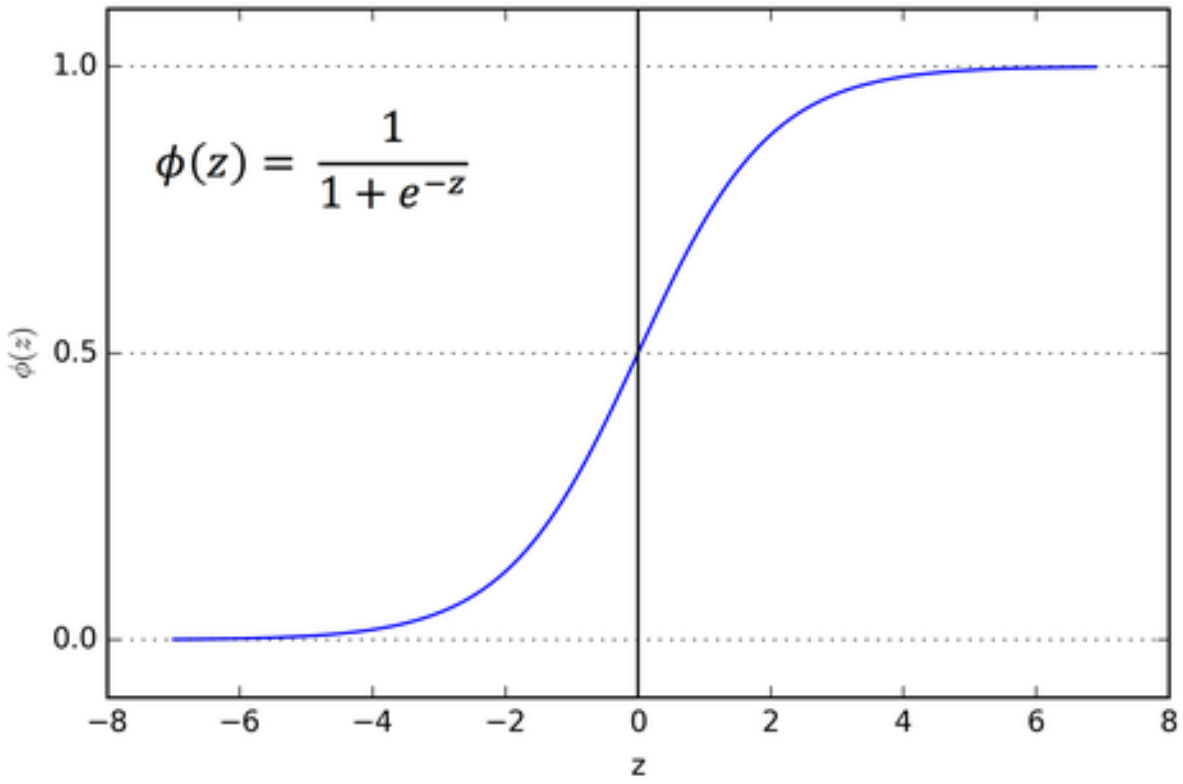


Terminologies

Activation Functions

- Sigmoid or Logistic Activation Function.

- So far, we've assumed that the activation function (nonlinearity) is always the sigmoid.

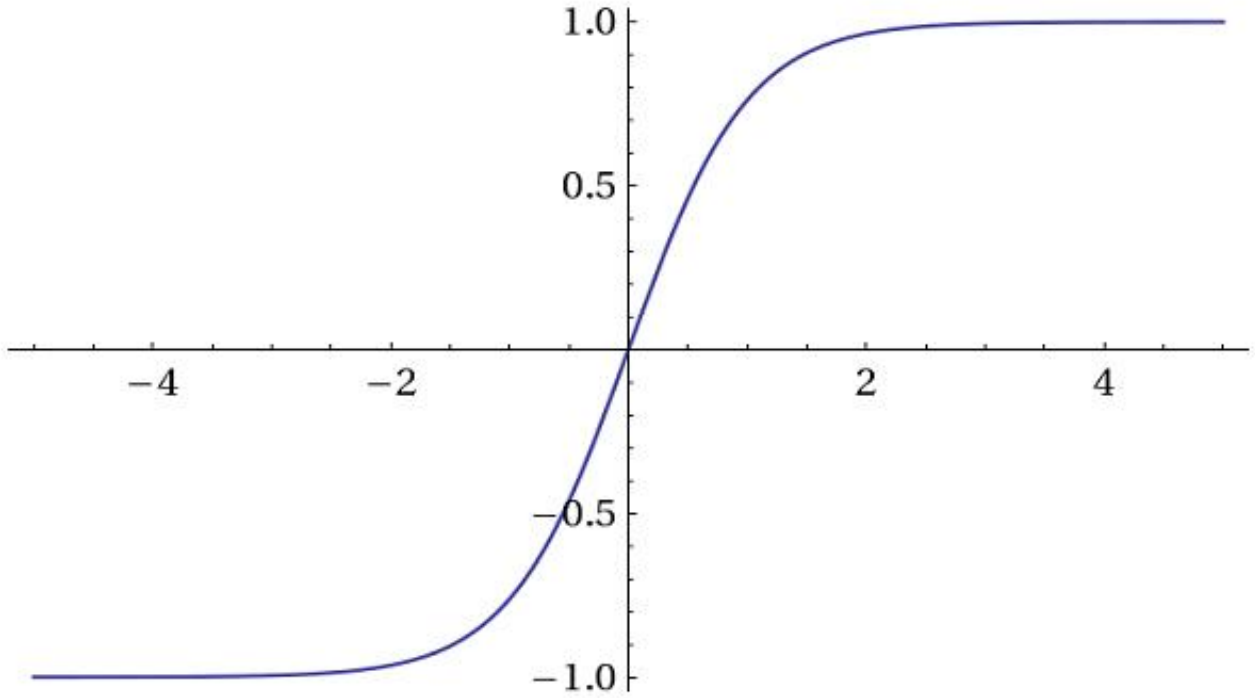


Activation Functions

- Tangent hyperbolic function simply referred to as tanh function

$$z = \tanh(x)$$

- Shifted the output range int (-1, +1)



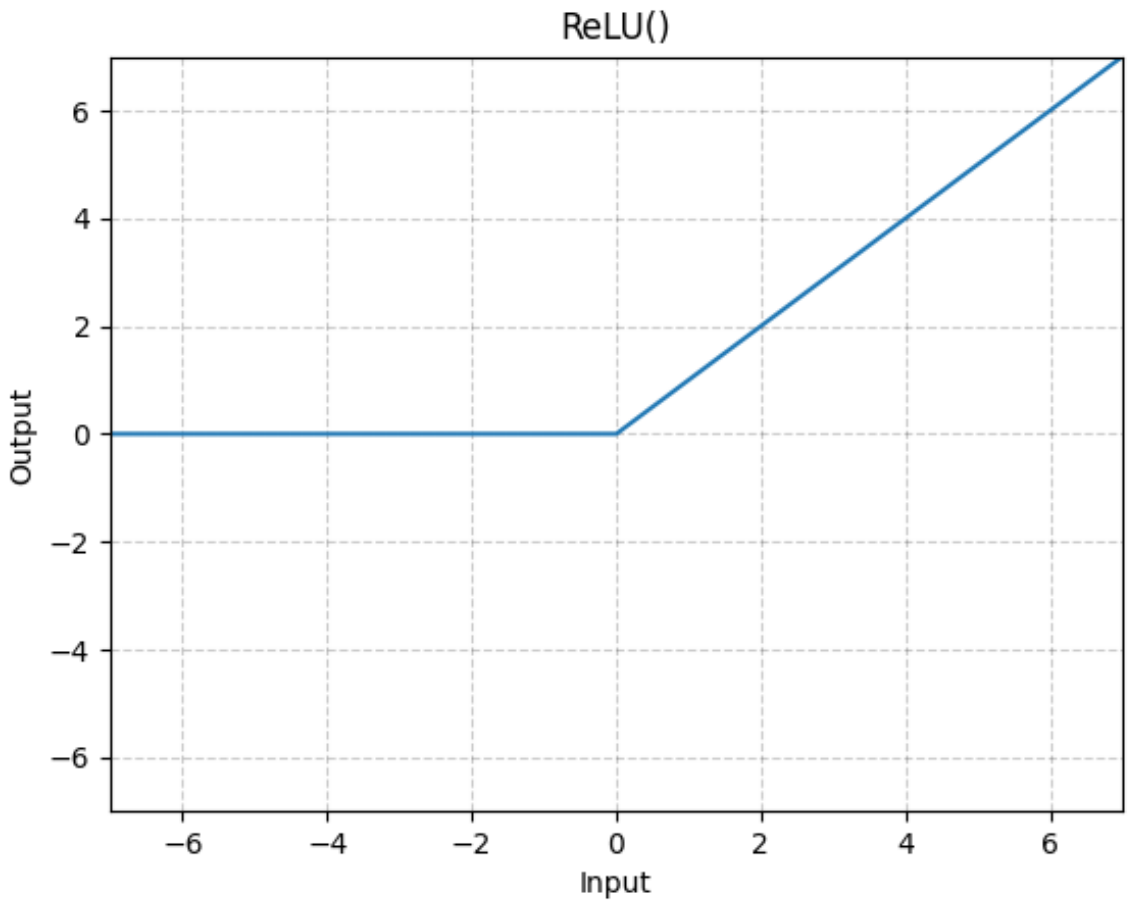
Activation Functions

- Intend: Reducing the gradiaent vanish problem

$$Relu = \max(0, x)$$

ReLU: Rectified Linear Unit

The gradient of sigmoids becomes increasingly small as the absolute value of *x* increases. The constant gradient of ReLUs results in faster learning.



Dataset Split

- Training Dataset

  The sample of data used to fit the model.

- Validation Dataset

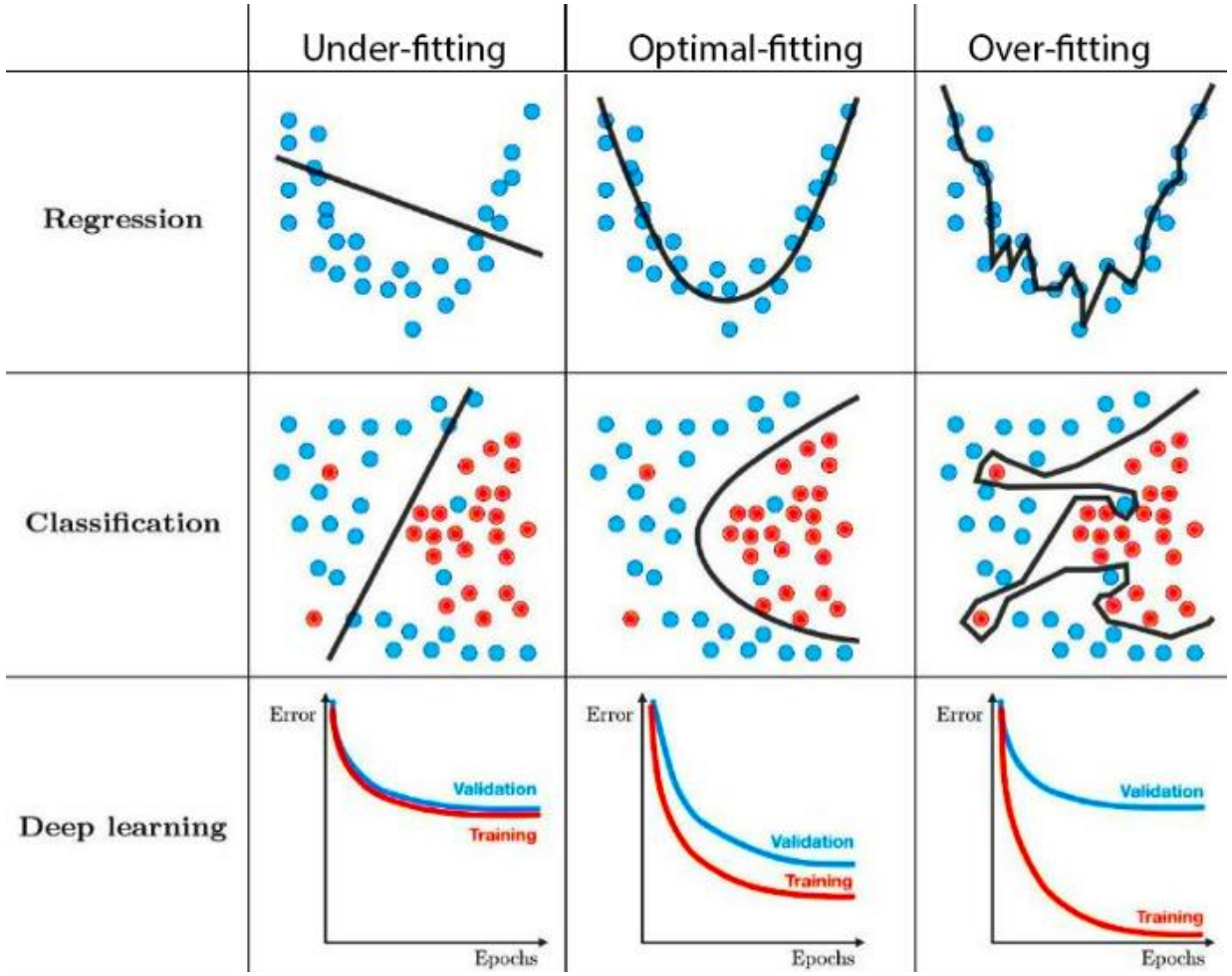
  The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters.

- Test Dataset

  The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.

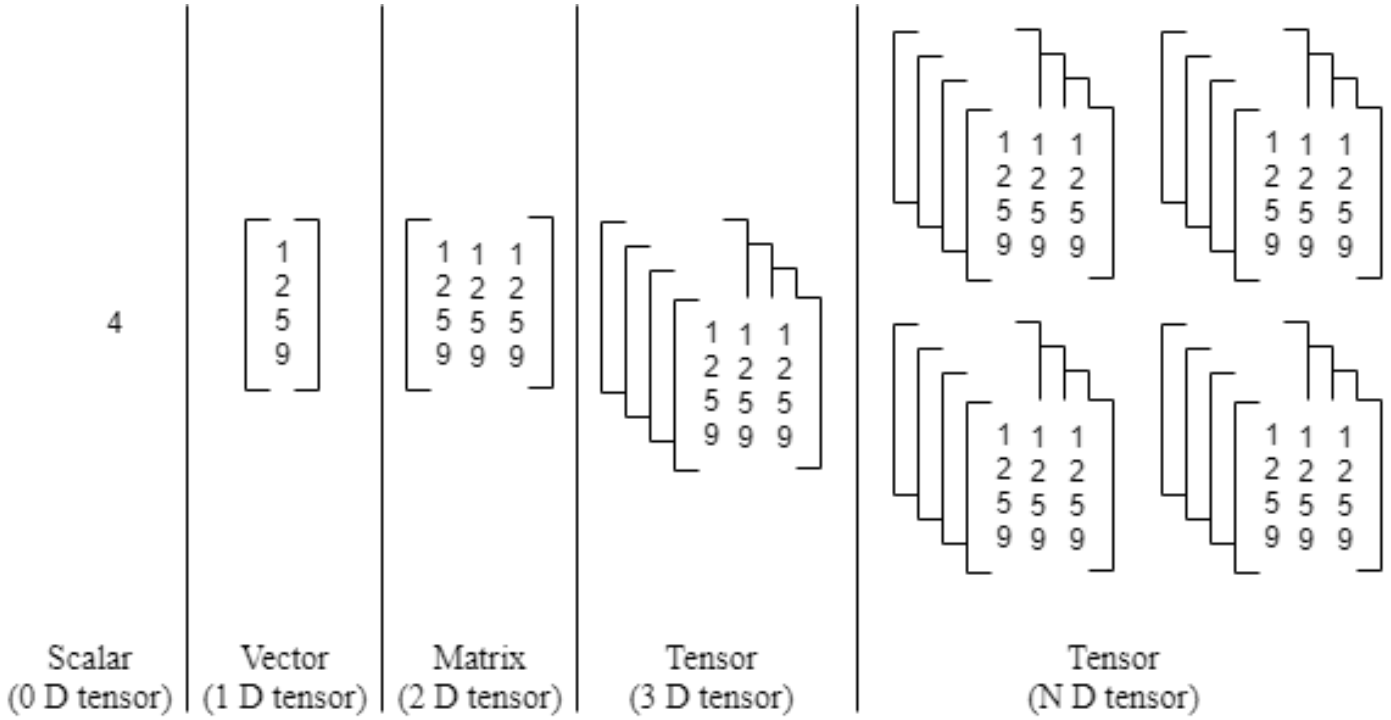
Overfitting and Underfitting

- Underfitting happens when the model has a very high bias and is unable to capture the complex patterns in the data.
- Overfitting is the opposite in the sense that the model is too complex (or higher model) and captures even the noise in the data.



Tensor

- A tensor is just a container for data, typically numerical data.
- Tensors are a generalization of matrices to any number of dimensions.
- You may already be familiar with matrices, which are 2D tensors.
- Note that in the context of tensors, a dimension is often called an axis.

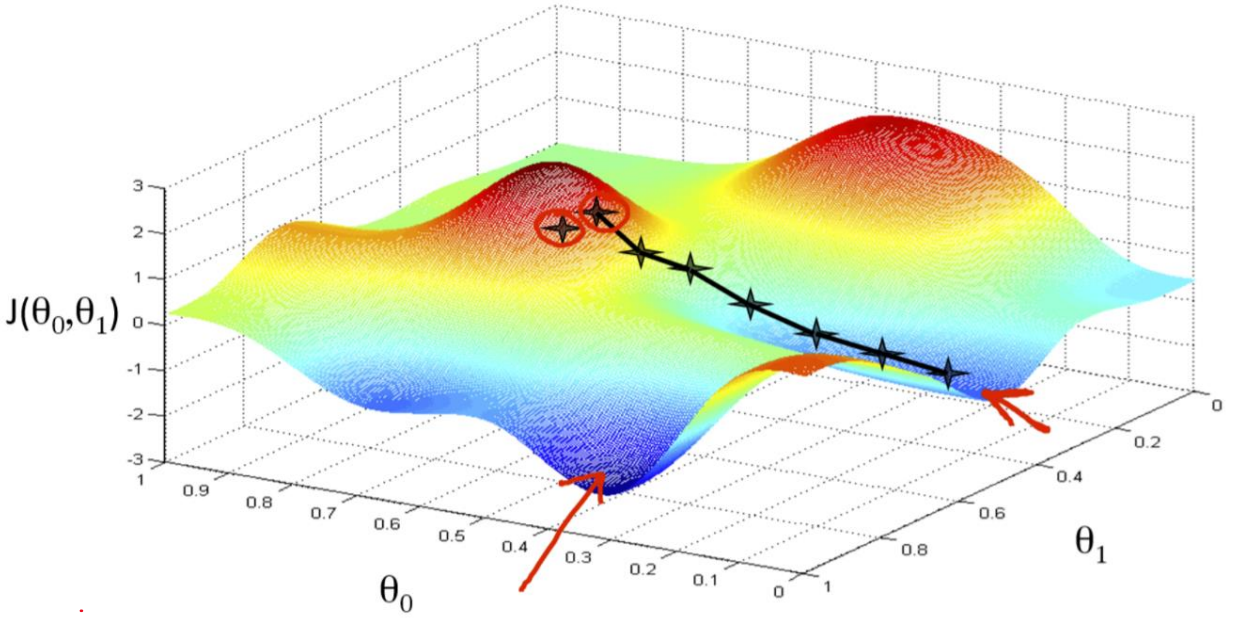


Gradient Descent

- Gradient descent is an optimization algorithm which is commonly used to train machine learning models and neural networks.
- Training data helps these models learn over time, and the cost function within gradient descent specifically acts as a barometer, gauging its accuracy with each iteration of parameter updates.
- Until the function is close to or equal to zero, the model will continue to adjust its parameters to yield the smallest possible error.

Gradient Descent

- The starting point is just an arbitrary point for us to evaluate the performance.
- From that starting point, we will find the derivative (or slope), and from there, we can use a tangent line to observe the steepness of the slope.
- The slope will inform the updates to the parameters—i.e. the weights and bias.



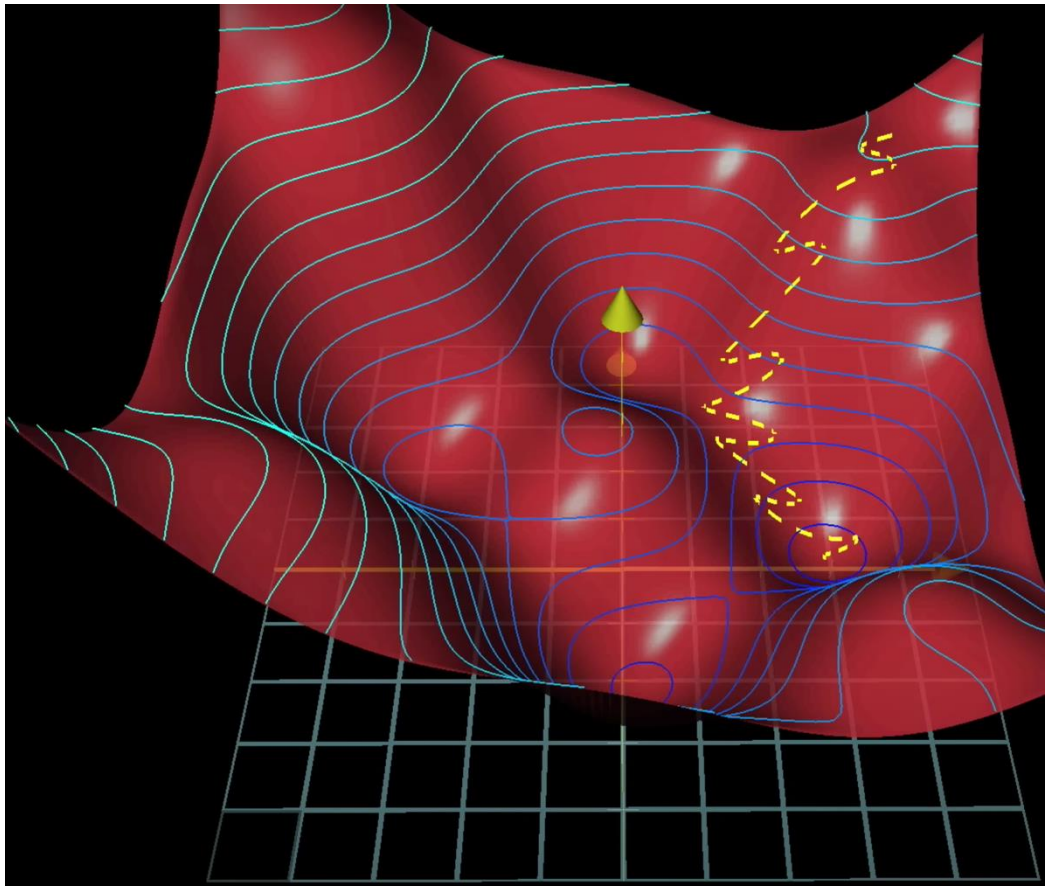
- The slope at the starting point will be steeper, but as new parameters are generated, the steepness should gradually reduce until it reaches the lowest point on the curve, known as the point of convergence.

Learning Rate

- Learning rate (also referred to as step size or the alpha) is the size of the steps that are taken to reach the minimum.
- This is typically a small value, and it is evaluated and updated based on the behaviour of the cost function.
- High learning rates result in larger steps but risks overshooting the minimum.
- Conversely, a low learning rate has small step sizes. While it has the advantage of more precision, the number of iterations compromises overall efficiency as this takes more time and computations to reach the minimum.

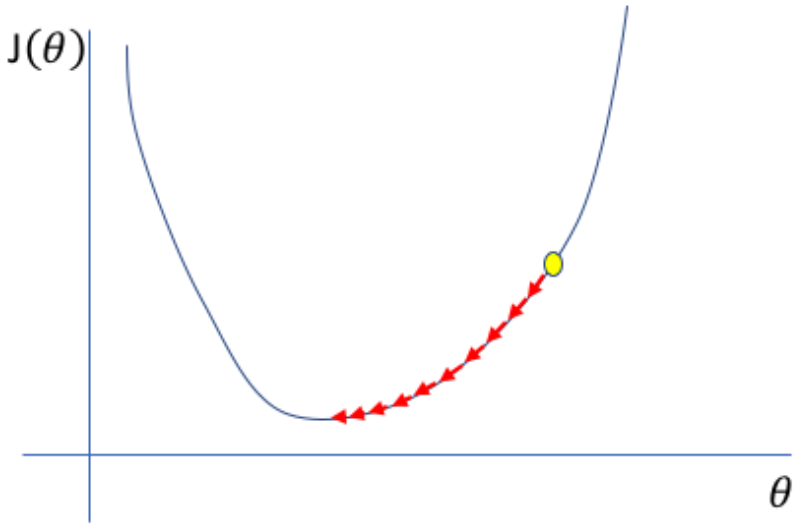
Learning Rate

- Learning rate is a hyper-parameter that controls how much we are adjusting the weights of our network with respect the loss gradient.



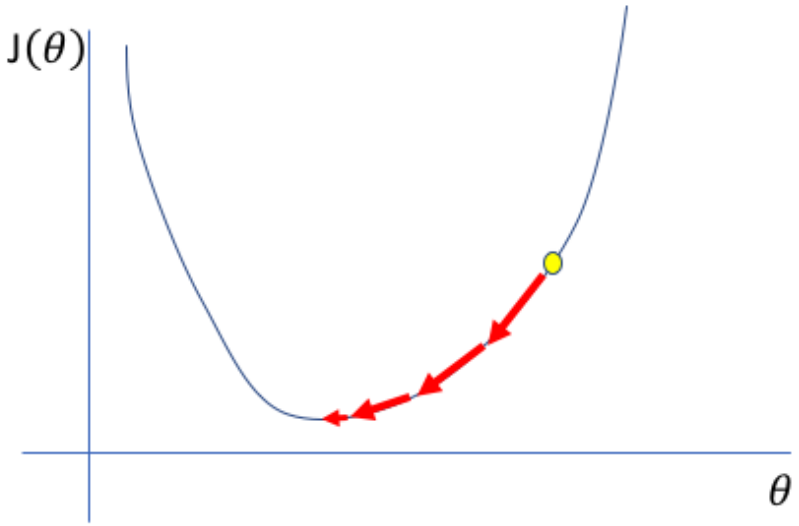
Learning Rate

Too low



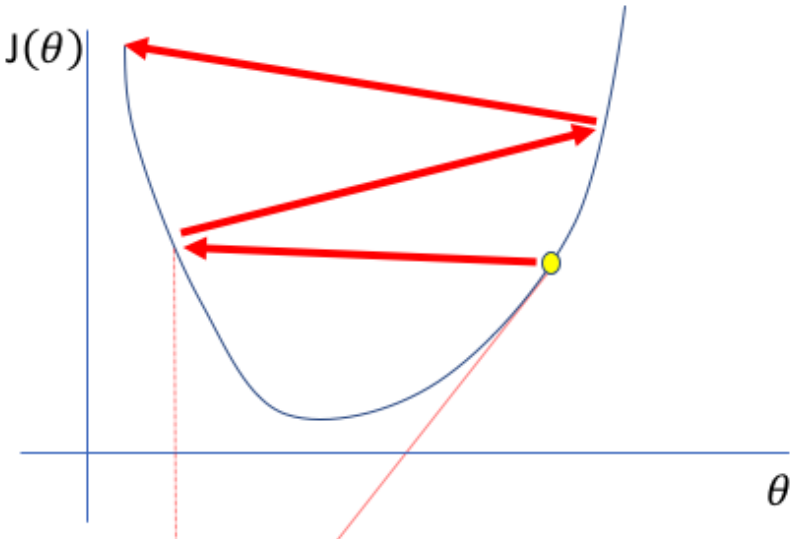
A small learning rate requires many updates before reaching the minimum point

Just right



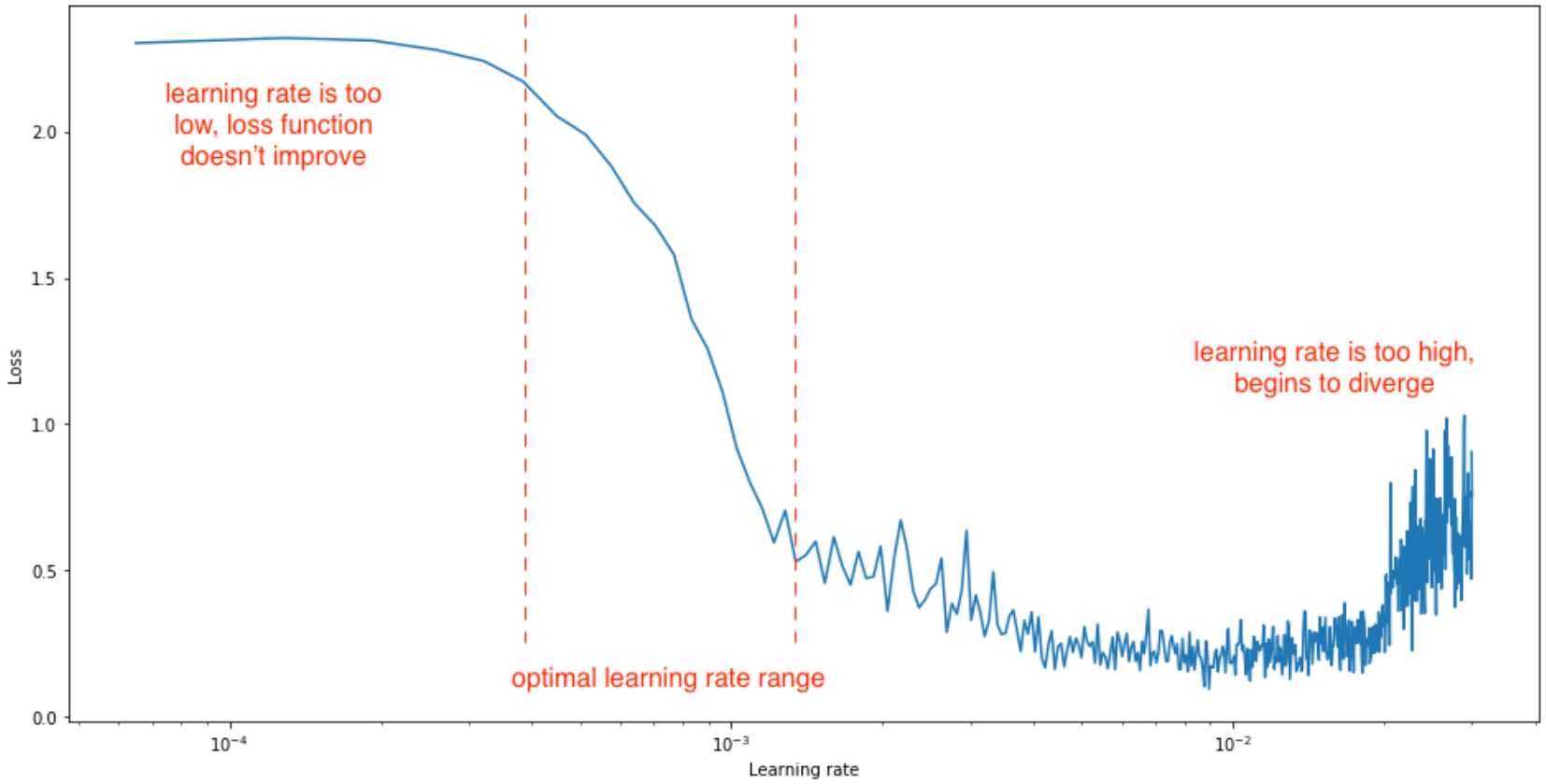
The optimal learning rate swiftly reaches the minimum point

Too high



Too large of a learning rate causes drastic updates which lead to divergent behaviors

Find a Good Learning Rate

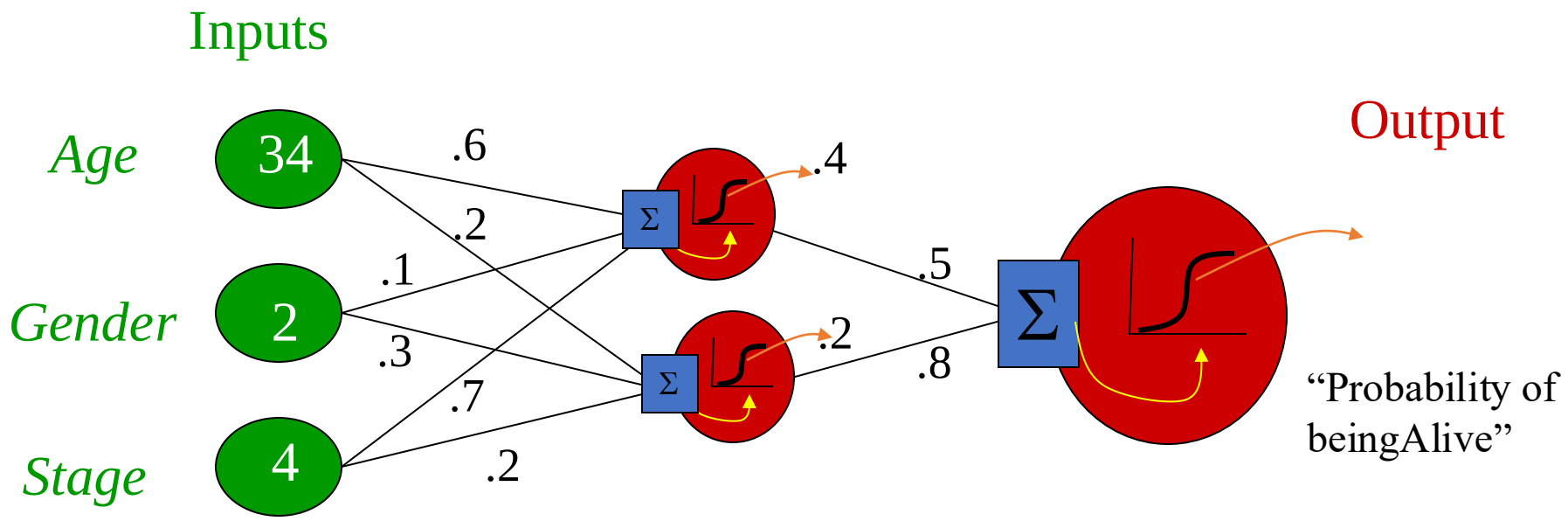


Momentum in Neural Network

- Gradient descent is an optimization algorithm that follows the negative gradient of an objective function in order to locate the minimum of the function.

- Momentum is an extension to the gradient descent optimization algorithm that allows the search to build inertia in a direction in the search space and overcome the oscillations of noisy gradients and coast across flat spots of the search space.

Example 2



Independent variables

Weights

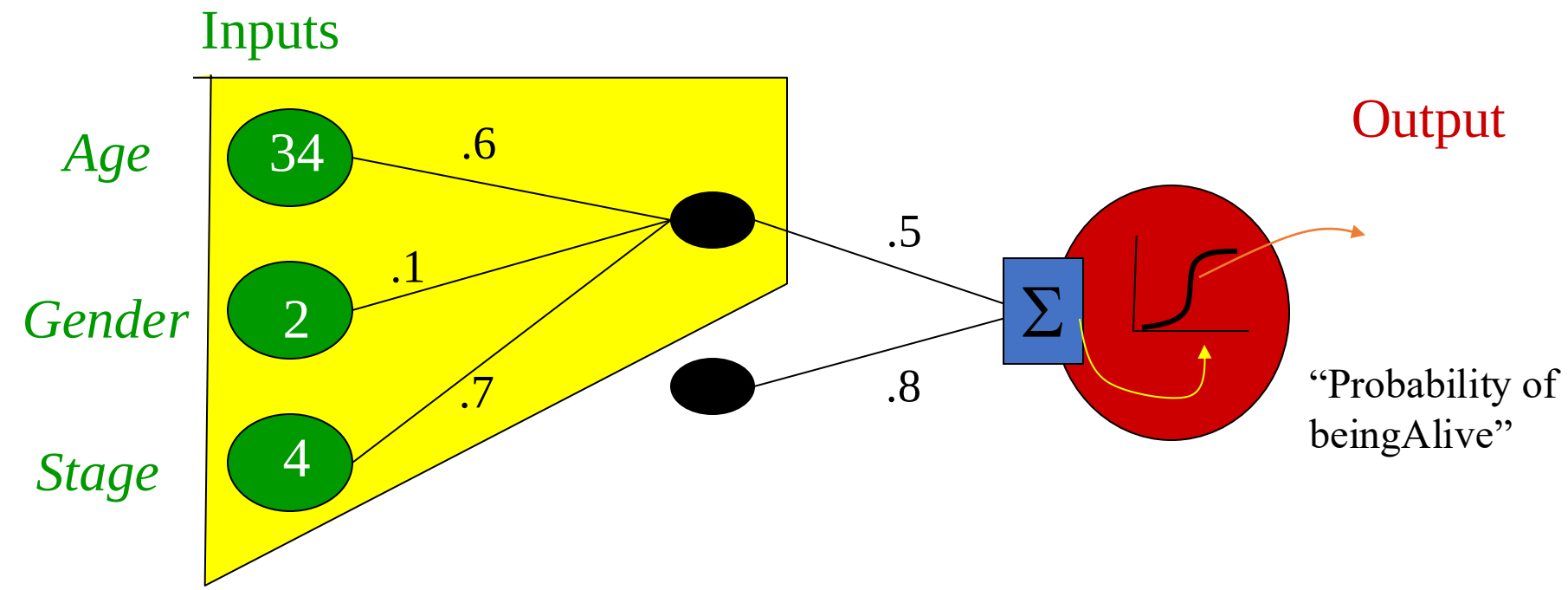
Hidden Layer

Weights

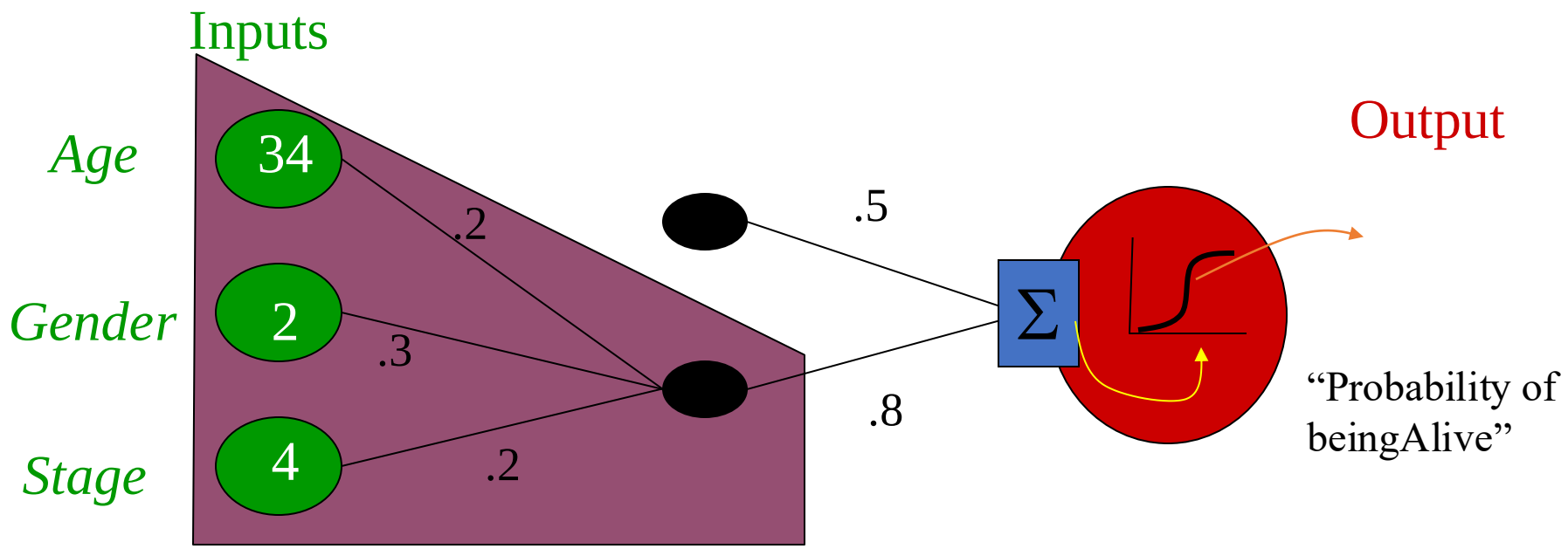
Dependent variable

Prediction

Example 2


Inputs
Age
Gender
Stage
34
2
4
.6
.1
.7
.5
.8
Σ
Output
"Probability of beingAlive"
Independent variables
Weights
Hidden Layer
Weights
Dependent variable
Prediction

Example 2



Independent variables

Weights

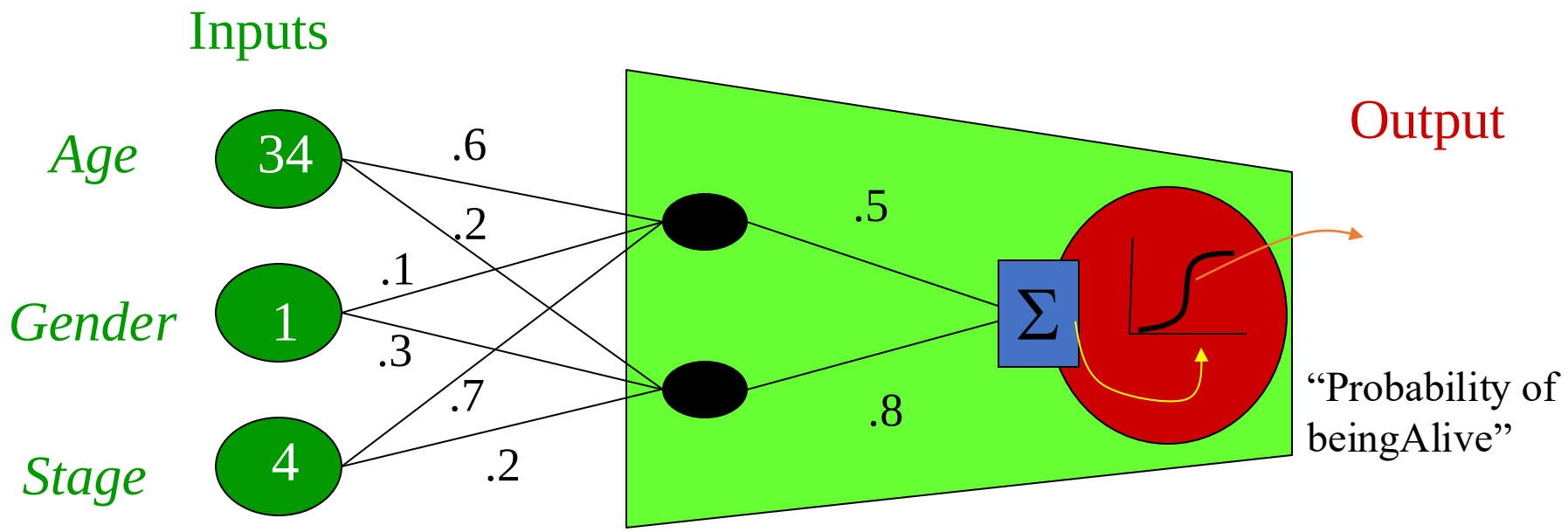
Hidden Layer

Weights

Dependent variable

Prediction

Example 2



Independent variables

Weights

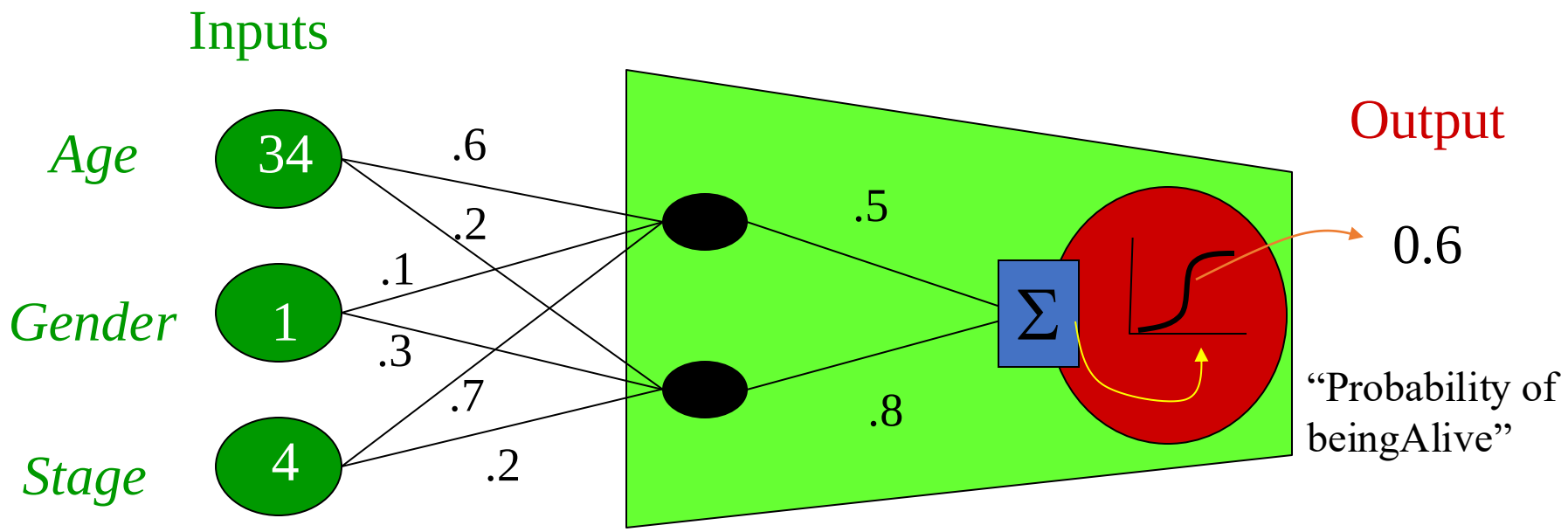
Hidden Layer

Weights

Dependent variable

Prediction

Example 2



Independent variables

Weights

Hidden Layer

Weights

Dependent variable

Prediction

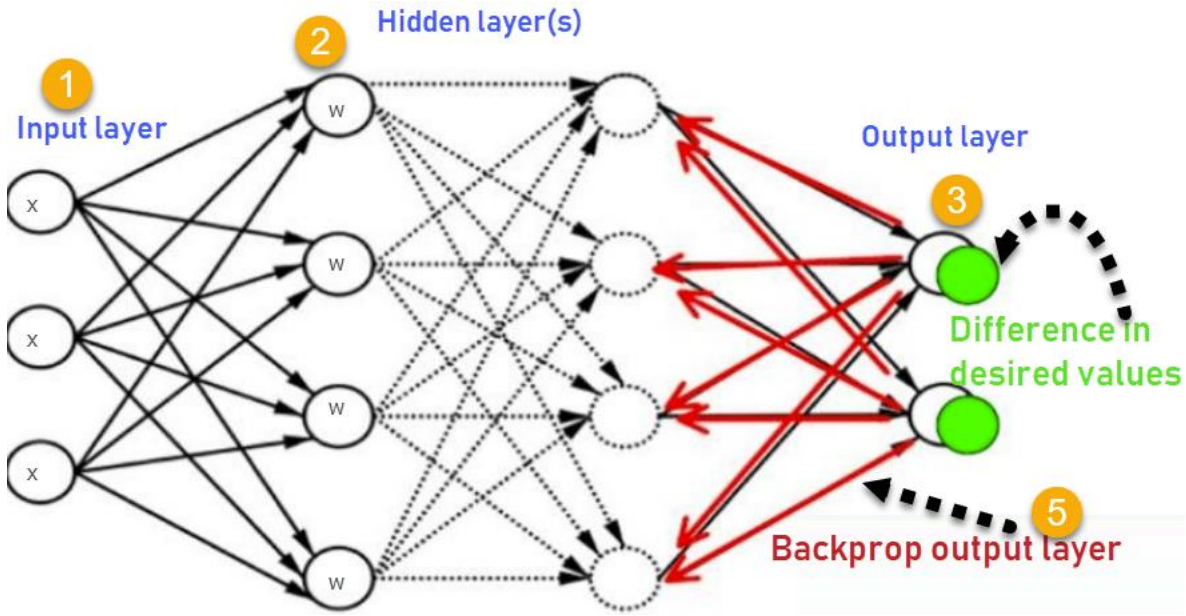
Backpropagation: Multi-Layer Perceptron

What is Backpropagation?

- Backpropagation in neural network is a short form for “backward propagation of errors”
- Backpropagation 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)
- This method helps calculate the gradient of a loss function with respect to all the weights in the network
- Proper tuning of the weights allows you to reduce error rates and make the model reliable by increasing its generalization

How Backpropagation Works?

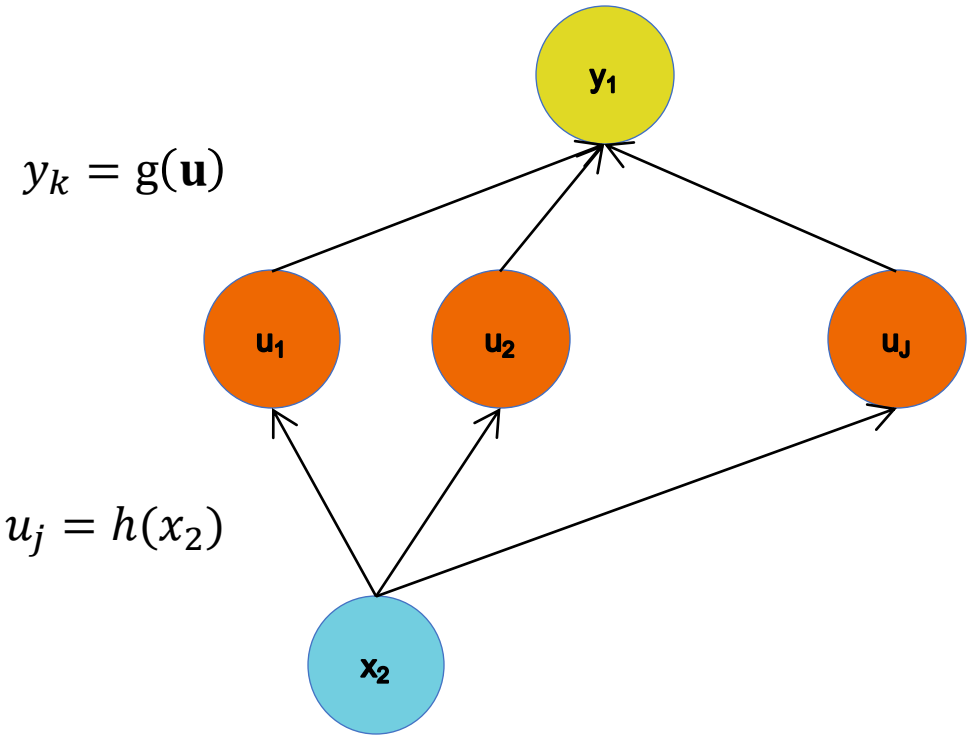
- Inputs X, arrive through the preconnected path
- Input is modelled using real weights W. The weights are usually randomly selected.
- Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
- Calculate the error in the outputs
- Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.
- Keep repeating the process until the desired output is achieved



Chain Rule

Given $y_i = \sum_j u_j * \widehat{w_j} \quad u_j = x_k * w_j$

Chain rule: $\frac{dy_i}{dx_k} = \sum_j \frac{dy_i}{du_j} \frac{du_j}{dx_k}$



Backpropagation is just repeated application of the **chain rule** from Calculus.

Backpropagation Algorithm

Chain rule: $\frac{dy_i}{dx_k} = \sum_j \frac{dy_i}{du_j}\frac{du_j}{dx_k}$

1. **Instantiate the computation as a directed acyclic graph**, where each intermediate quantity is a node
2. At each node, store (a) the quantity computed in the forward pass and (b) the **partial derivative** of the goal with respect to that node's intermediate quantity.
3. **Initialize** all partial derivatives to 0.
4. Visit each node in **reverse topological order**. At each node, add its contribution to the partial derivatives of its parents

This algorithm is also called **automatic differentiation in the reverse-mode**

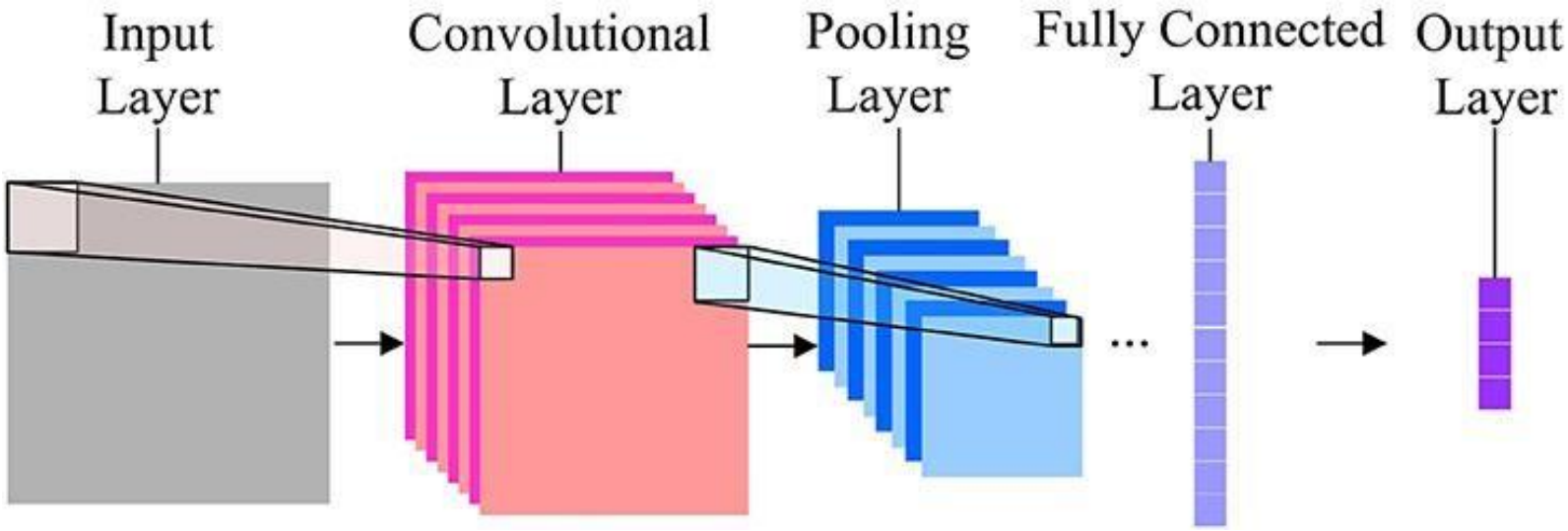
Convolutional Neural Networks

What is CNNs?

- When it comes to image classification, the most used neural networks are Convolution Neural Networks (CNN).
- CNNs contain multiple convolution layers, which are responsible for the extraction of important features from the image
- The earlier layers are responsible for low-level details, and the later layers are responsible for more high-level features

Applications:

- Image processing
- Computer Vision
- Speech Recognition
- Machine translation

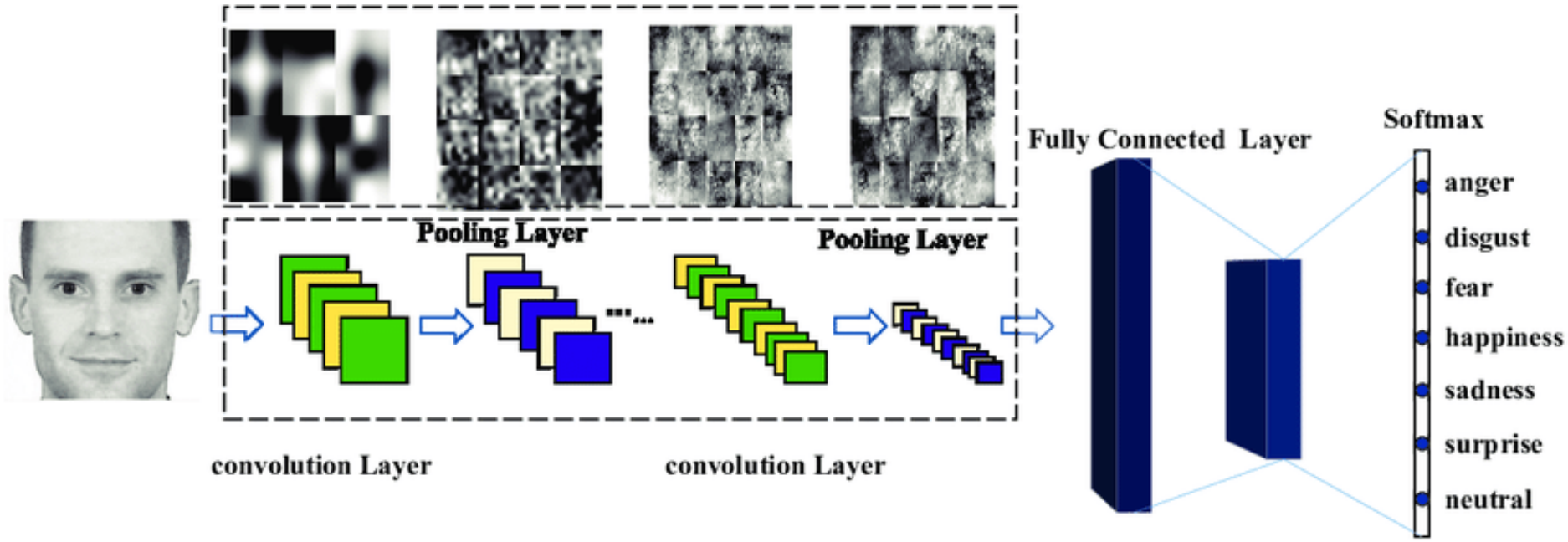


Pros and Cons of CNN

- Advantages:
  - Used for deep learning with few parameters
  - Less parameters to learn as compared to fully connected layer

- Disadvantages:
  - Comparatively complex to design and maintain
  - Comparatively slow [depends on the number of hidden layers]

CNN Example



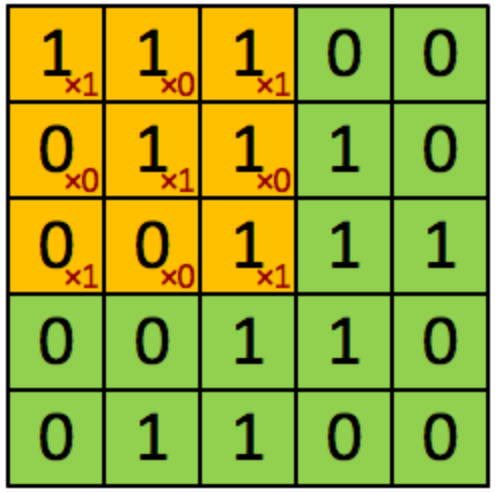
Emotion Recognition from face image

Convolutional Neural Network

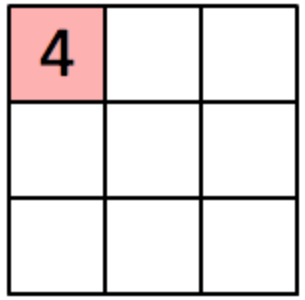
- How to compute the convolution of an image?

Suppose the kernel K=

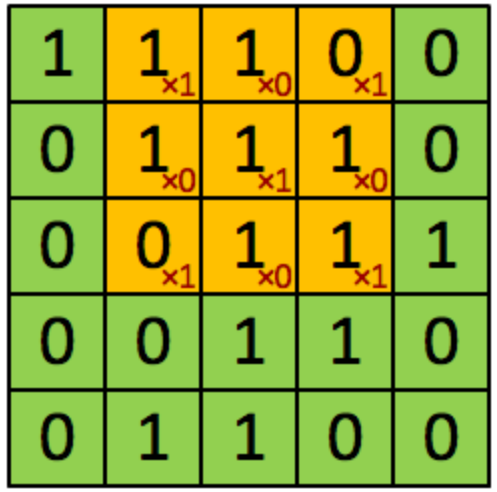
$$\begin{matrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{matrix}$$



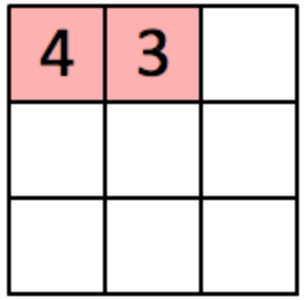
Image



Convolved Feature



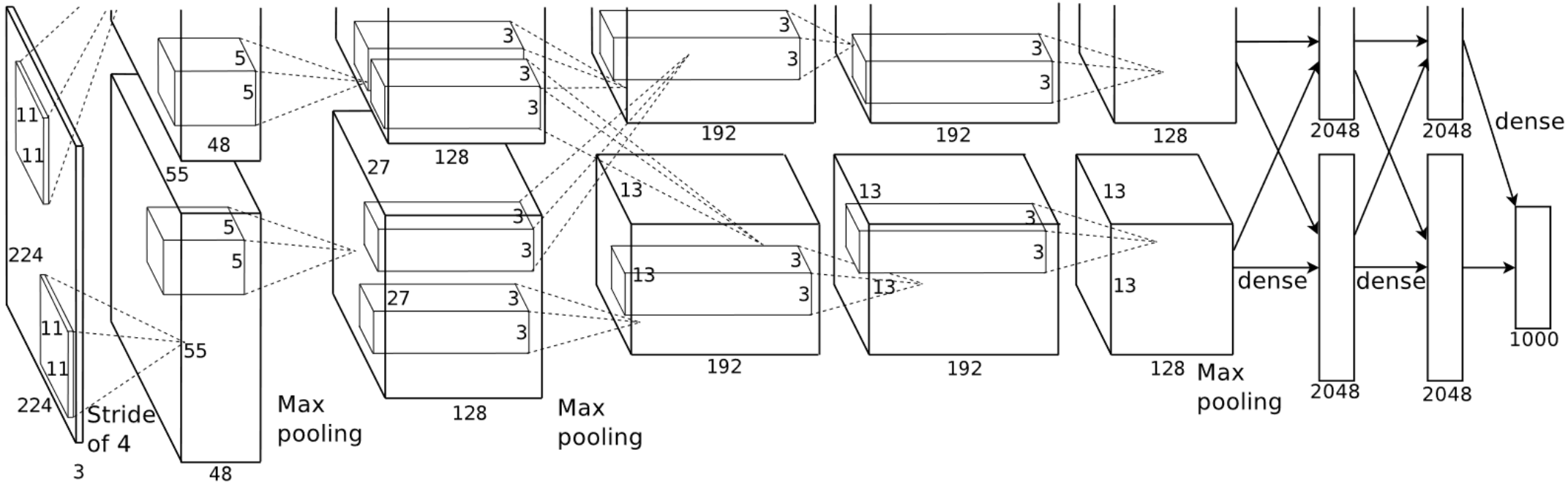
Image



Convolved Feature

AlexNet

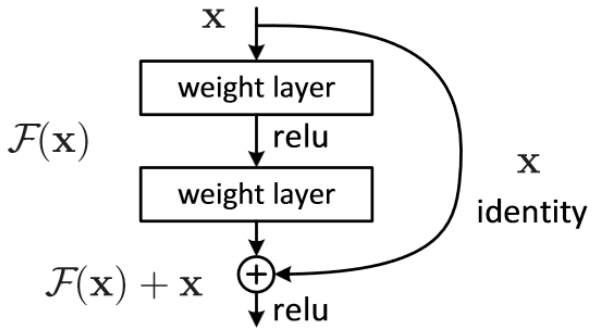
We want the model to **learn the kernel**



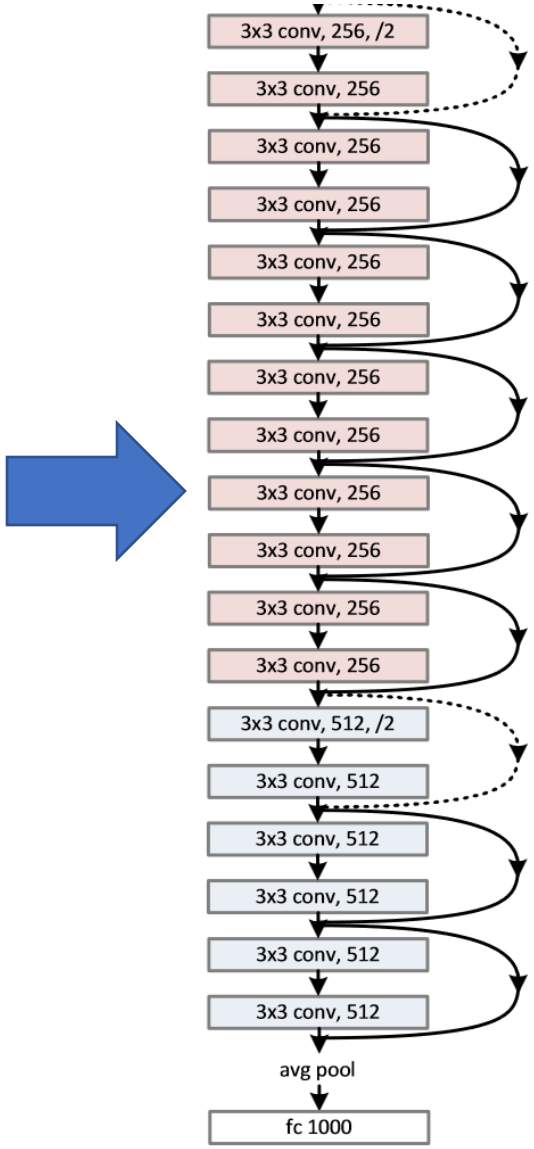
Achieved best performance in image classification competition 2010

Source: A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,”

Residual Network



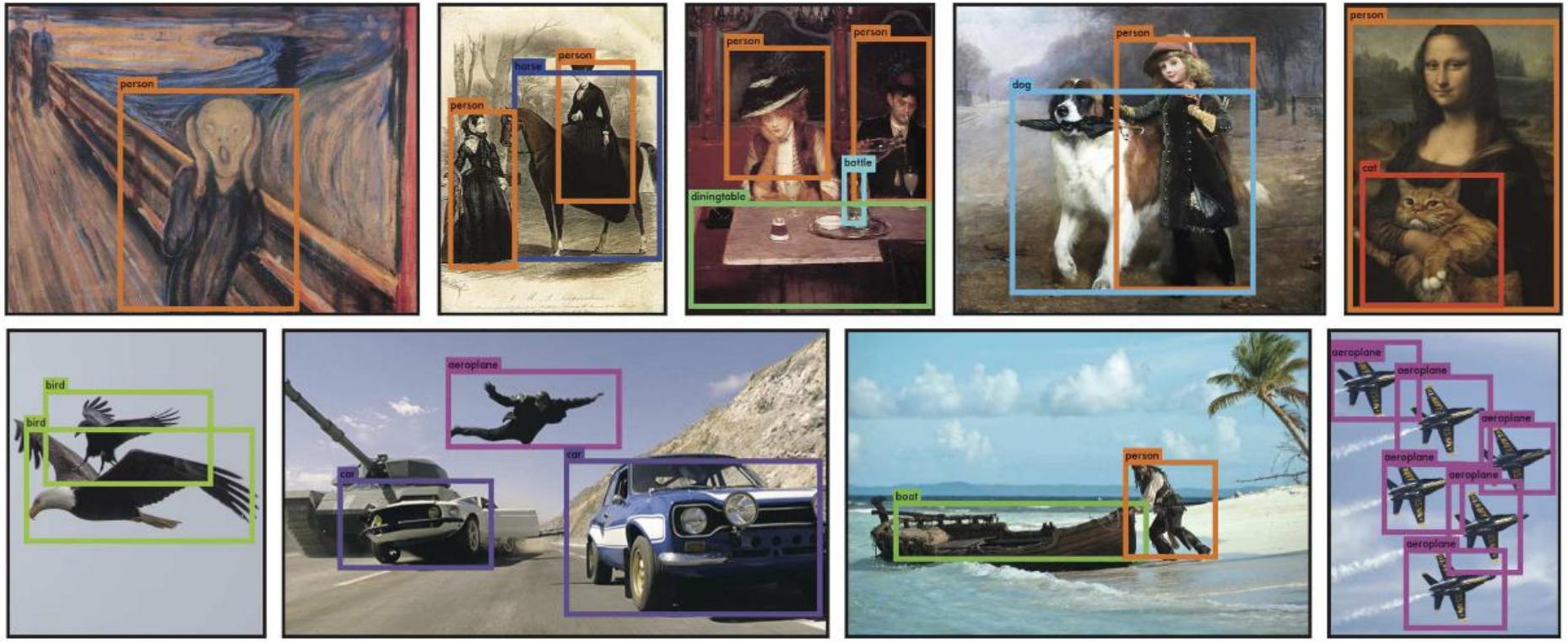
K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition"



By adding an identity mapping in the between the convolutional layer, the authors created a deeper network for image classification

This model won the 1st place on the ILSVRC 2015 classification task.

You Only Look Once(**YOLO**): Unified, Real-Time Object Detection



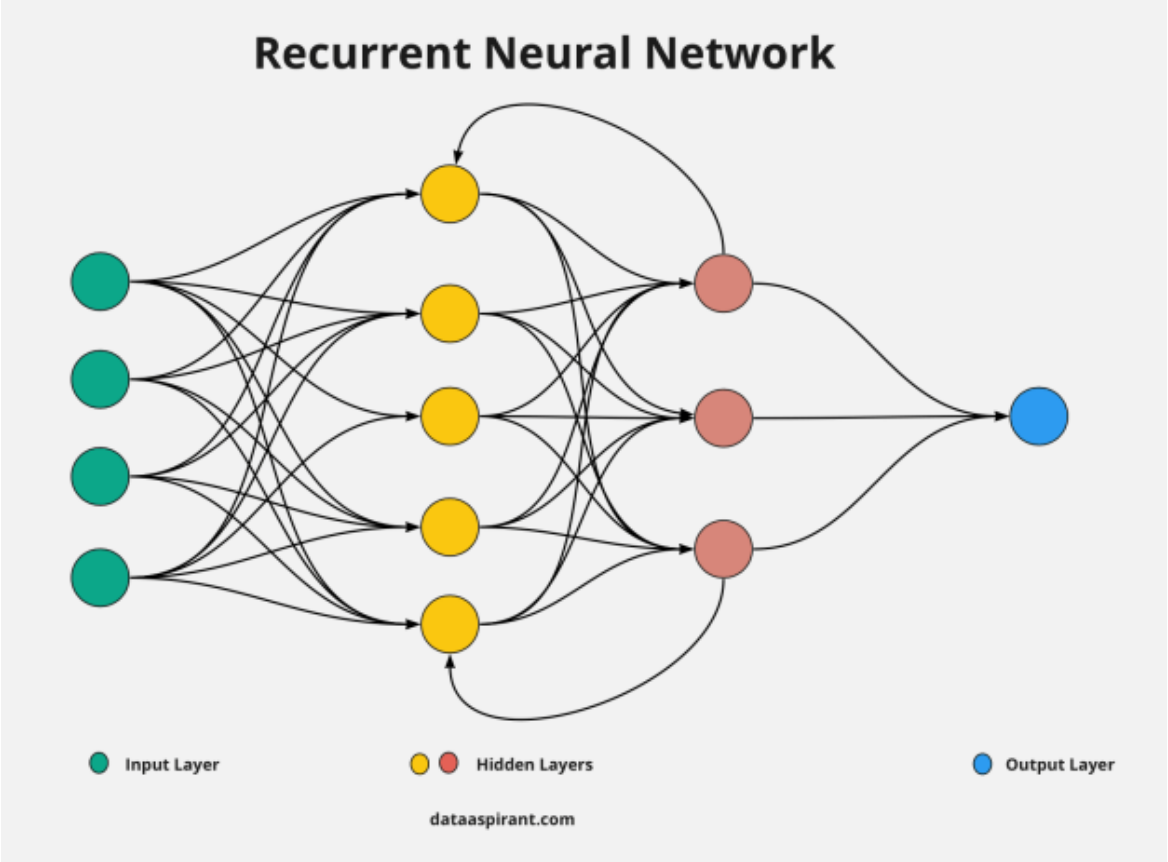
A **single neural network** predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, **it can be optimized end-to-end directly** on detection performance.

Source: J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection,"

Recurrent Neural Network

What is RNNs?

- RNNs come into picture when there's a need for predictions using sequential data
- Sequential data can be a sequence of images, words, etc
- The RNN have a similar structure to that of a Feed-Forward Network, except that the layers also receive a time-delayed input of the previous instance prediction
- This instance prediction is stored in the RNN cell which is a second input for every prediction



Applications of RNNs?

- Prediction problems
- Machine Translation
- Speech Recognition
- Language Modelling and Generating Text (Text processing like auto suggest, grammar checks, etc.)
- Video tagger
- Generating image descriptions
- Sentiment Analysis
- Text Summarization.
- Call Centre Analysis.

Pros and Cons

- Advantages:
  - Model sequential data where each sample can be assumed to be dependent on historical ones is one of the advantage
  - Used with convolution layers to extend the pixel effectiveness

- Disadvantages:
  - Gradient vanishing and exploding problems
  - Training recurrent neural nets could be a difficult task
  - Difficult to process long sequential data using ReLU as an activation function

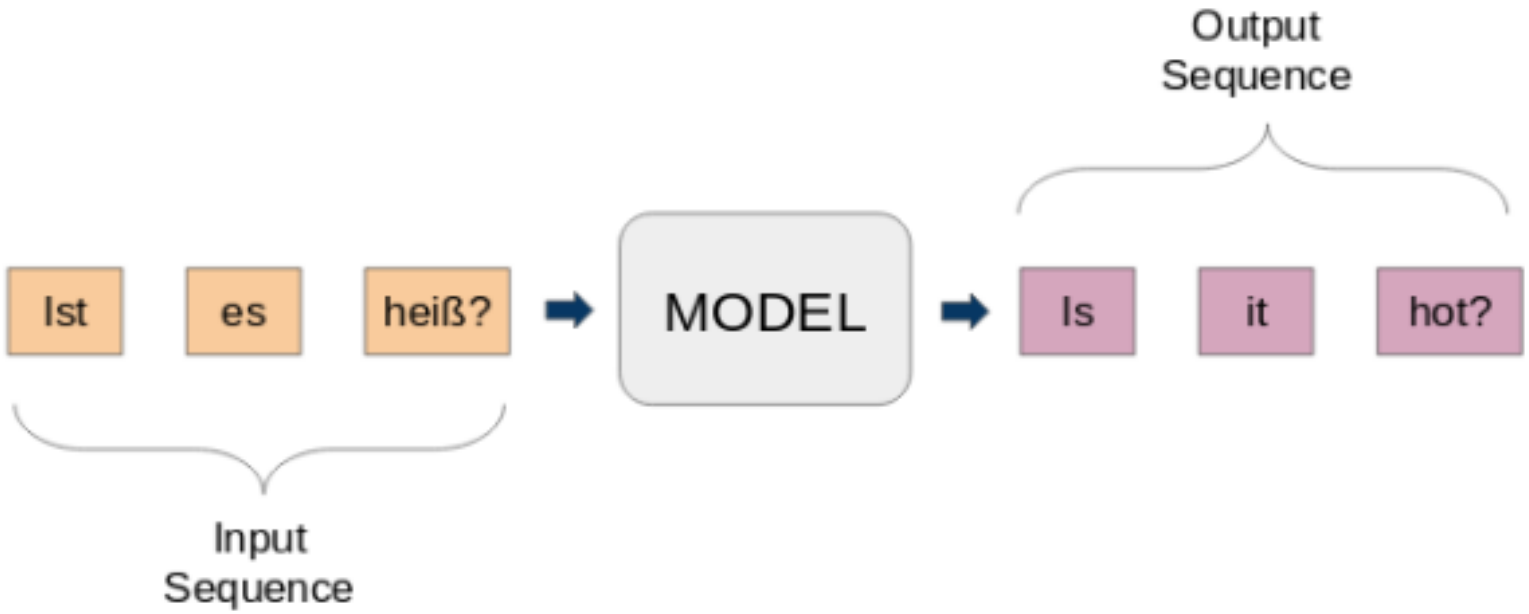
Sequence to Sequence Models

What is StS Models?

- A StS model consists of two Recurrent Neural Networks
- There exists an **encoder** that processes the input and a **decoder** that processes the output
- The encoder and decoder work simultaneously – either using the same parameter or different ones
- This model, on contrary to the actual RNN, is particularly applicable in those cases where the length of the input data is equal to the length of the output data
- These models are usually applied mainly in chatbots, machine translations, and question answering systems

Sequence to Sequence Models

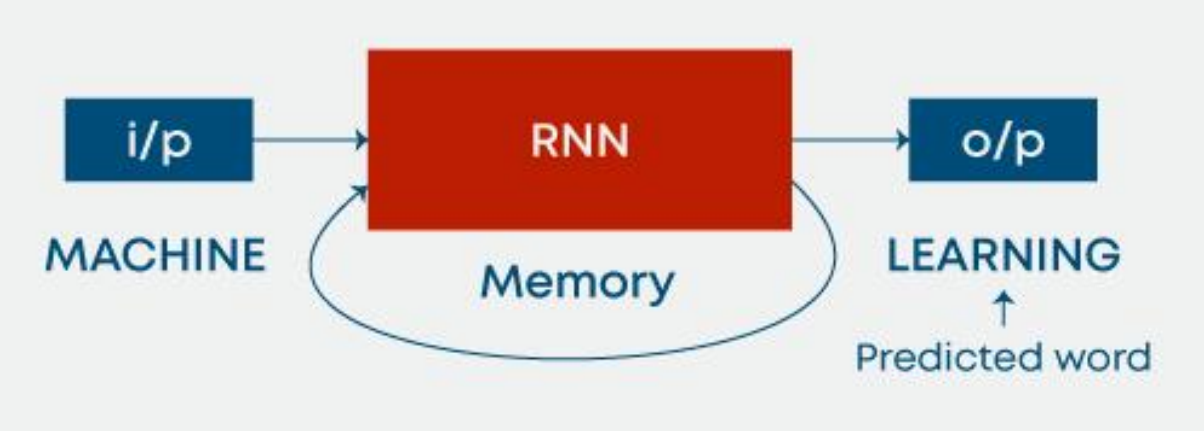
- Example: Machine translation



LSTM: Long Short-Term Memory

What is LSTM?

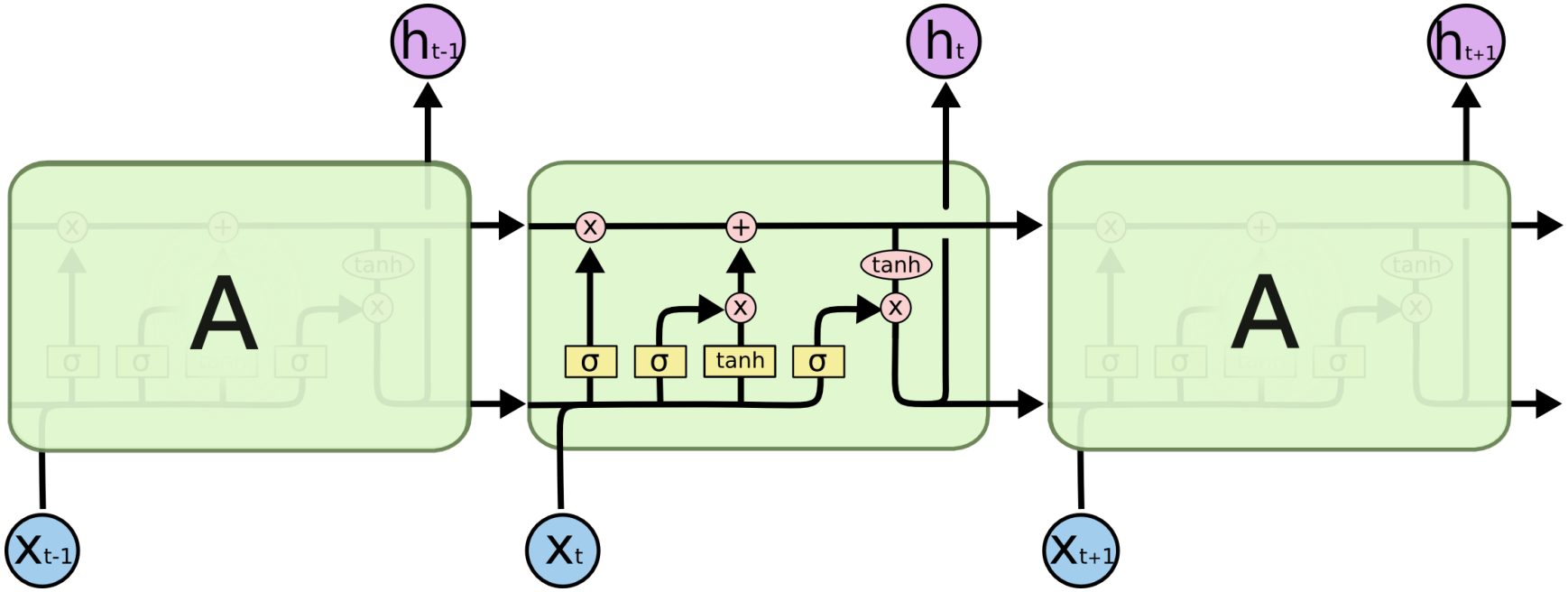
- LSTM Neural Networks overcome the issue of Vanishing Gradient in RNNs by adding a special memory cell that can store information for long periods of time
- LSTM uses 3 gates to define which output should be used or forgotten - Input gate, Output gate and a Forget gate
  - The Input gate controls what all data should be kept in memory
  - The Output gate controls the data given to the next layer
  - The forget gate controls when to dump/forget the data not required



Applications of LSTM?

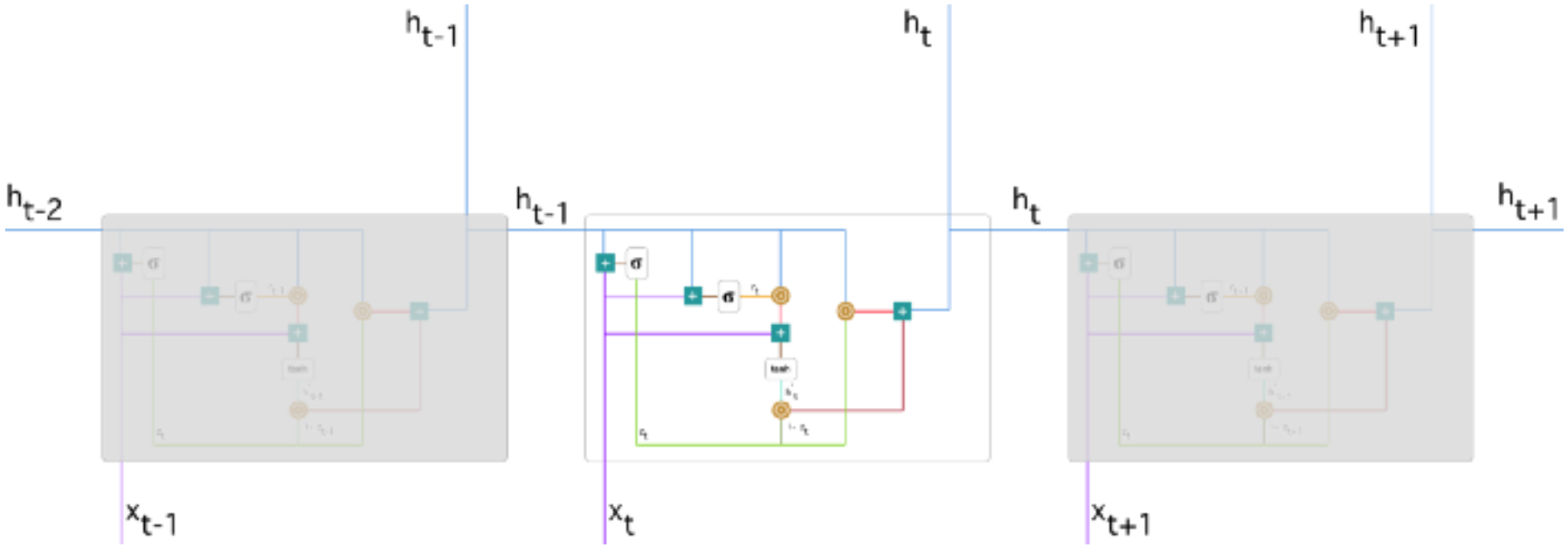
- Applications:
  - Gesture recognition
  - Speech recognition
  - Text prediction

Long Short-Term Memory (LSTM)



- LSTMs are explicitly designed to avoid the long-term dependency problem
- Remembering information for long periods of time is practically their default behavior, not something they struggle to learn
- They really work a lot better for most tasks!

Gated Recurrent Unit (GRU)



- Aims to solve the vanishing gradient problem which comes with a standard recurrent neural network. GRU can also be considered as a variation on the LSTM

Graph Neural Network

Graph Neural Network (GNN)

- Although counterintuitive, one can learn more about the symmetries and structure of images and text **by viewing them as graphs** and build an intuition that will help understand other less grid-like graph data