ANNs

Artificial Neural Networks

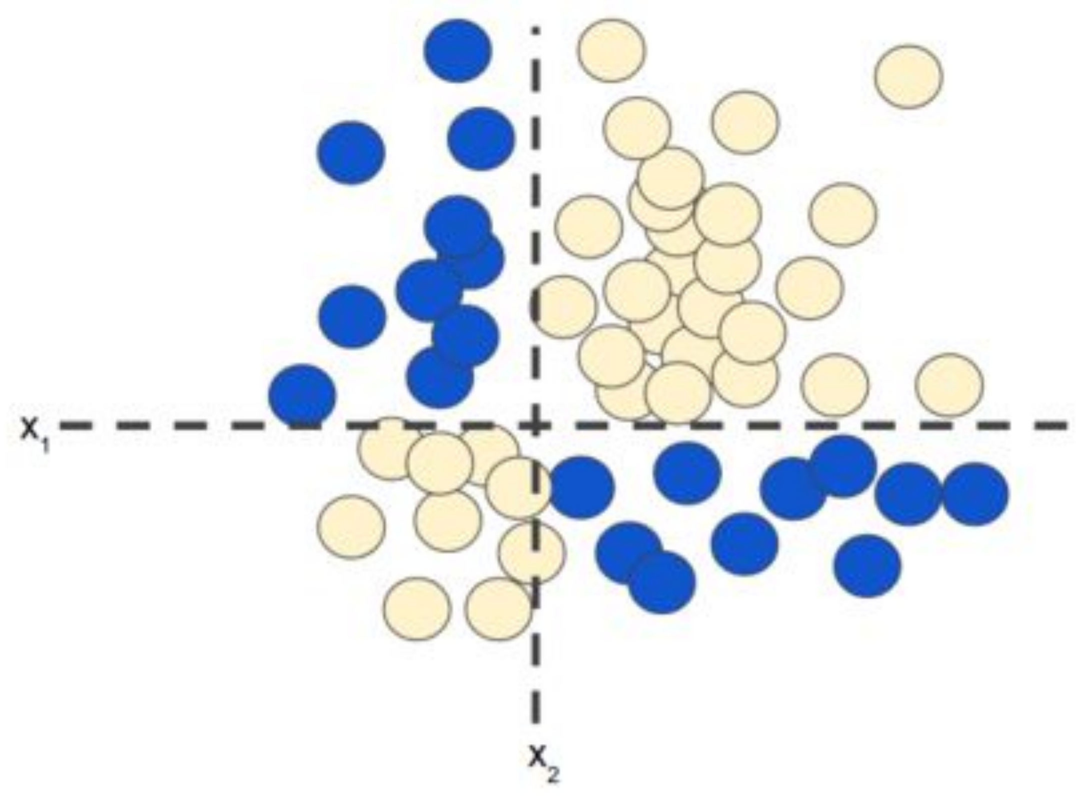
Part 1

Agenda

- Motivation
- What are Neural Networks?
- Activation Functions
- Backpropagation
- NN Types
  - Feed-Forward Neural Networks
  - Recurrent Neural Networks (RNN)
  - Convolutional Neural Networks (CNN)
  - Generative Adversarial Networks (GANs)
  - Transformer Networks
- NN for Finance

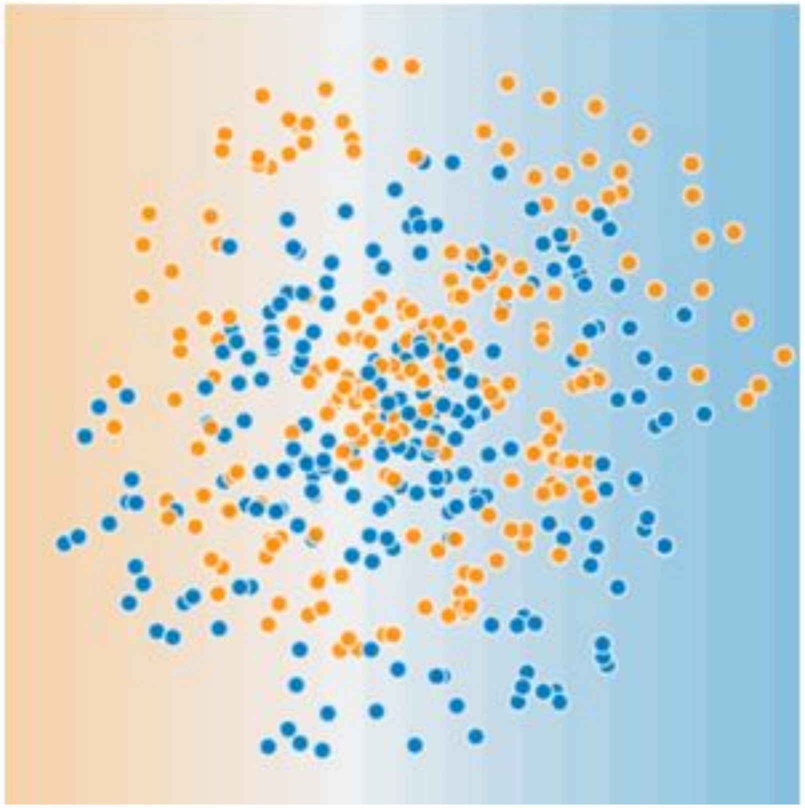
MOTIVATION

Revisit Non-Linear Problems



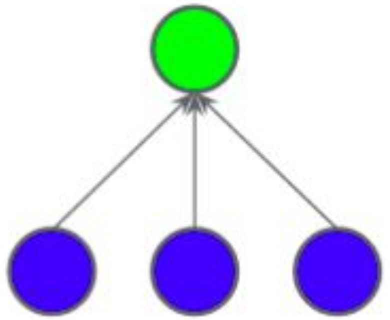
How did we solve this *simple* non-linear problem? Feature Cross

How to solve this one?



More complex non-linear problem. Would be great if feature crosses were done automatically!

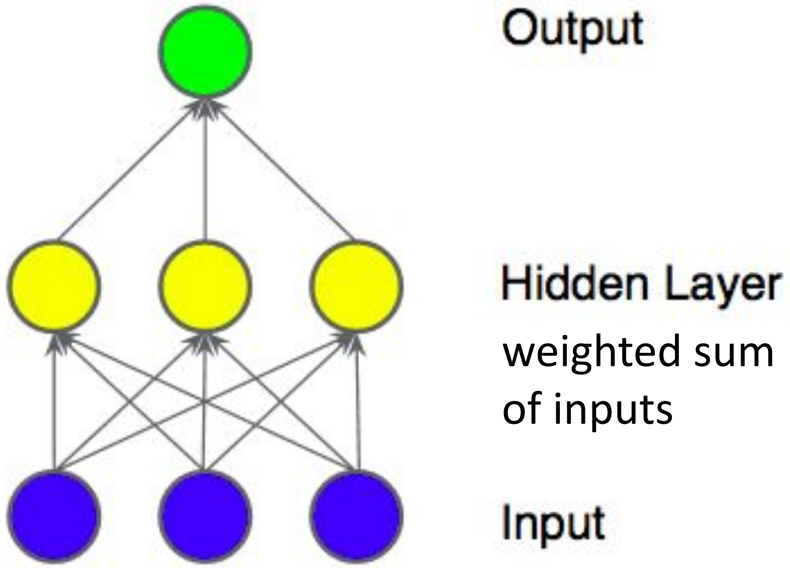
Graph representation of models



= weighted sum of the inputs

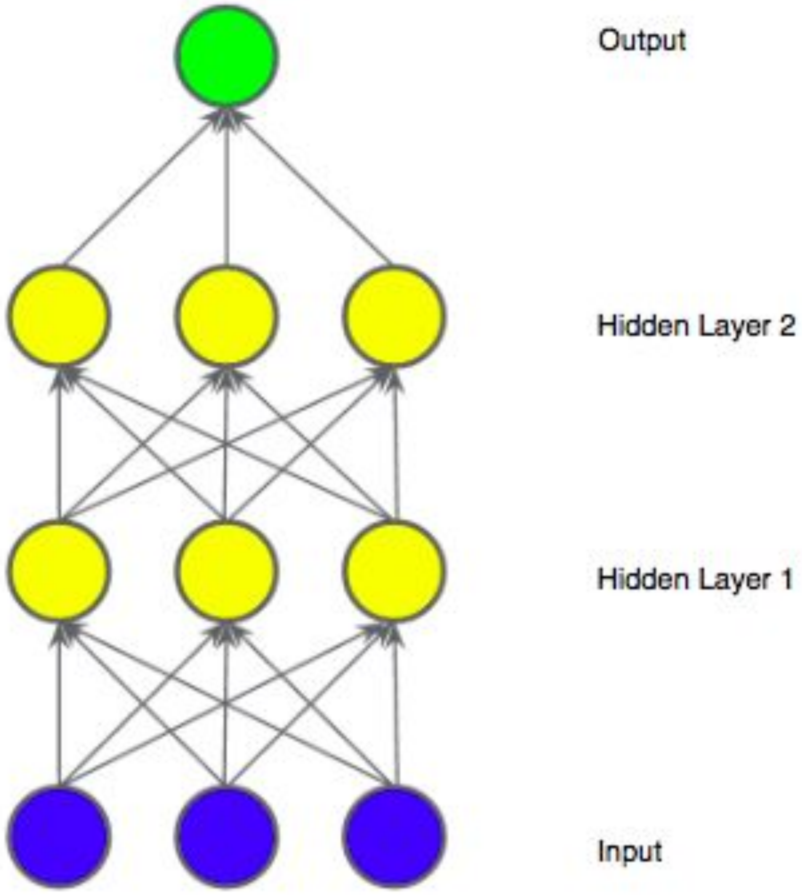
= feature

Linear



weighted sum of inputs

Linear?

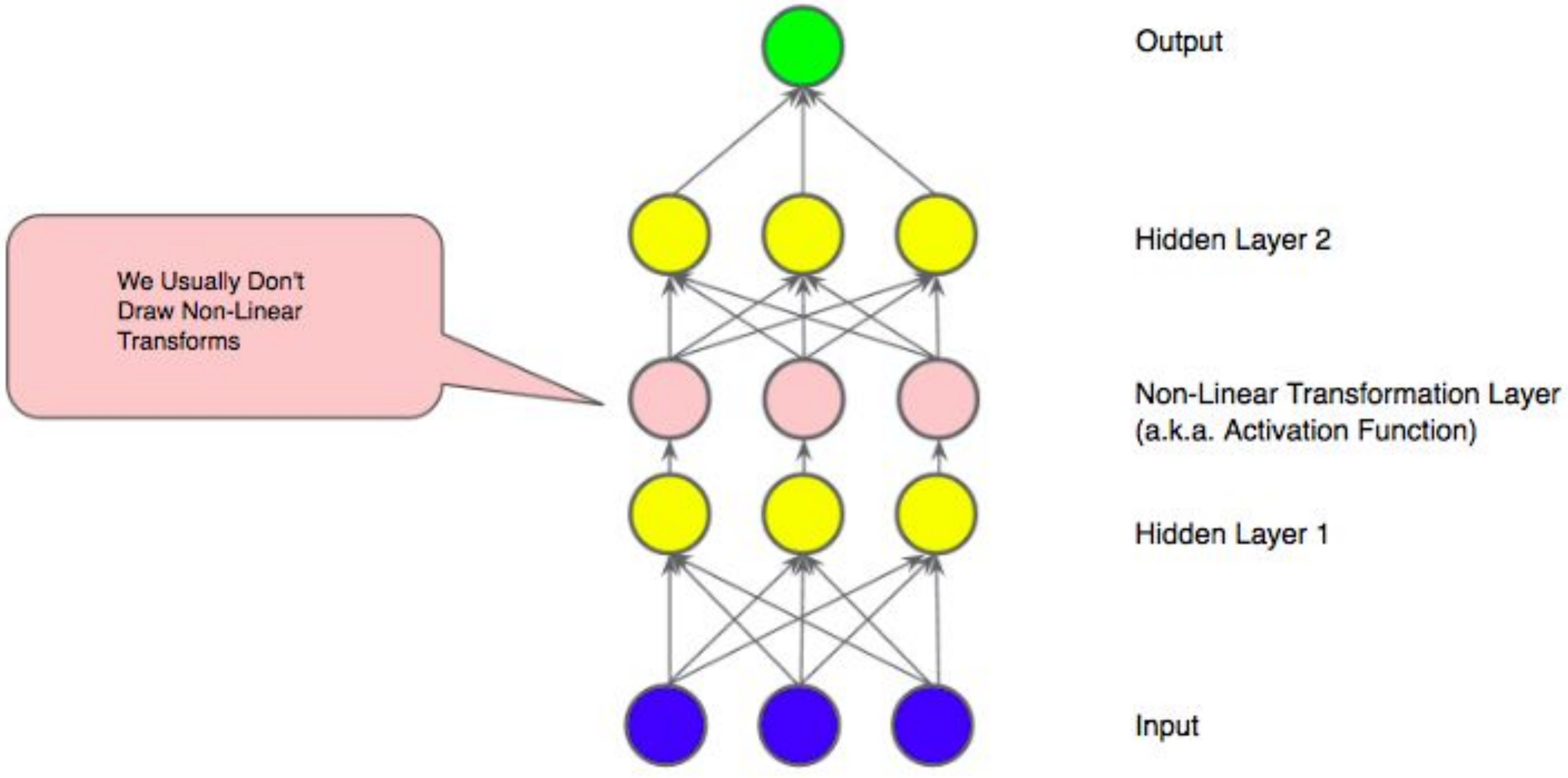


Linear?

Activation Functions

- To model a nonlinear problem, we can directly introduce a nonlinearity.
- We can pipe each hidden layer node through a nonlinear function (a.k.a activation function).
- Common activation functions
  - Sigmoid
  - ReLU
  - Tanh

Activation Functions


We Usually Don't Draw Non-Linear Transforms
Output
Hidden Layer 2
Non-Linear Transformation Layer (a.k.a. Activation Function)
Hidden Layer 1
Input

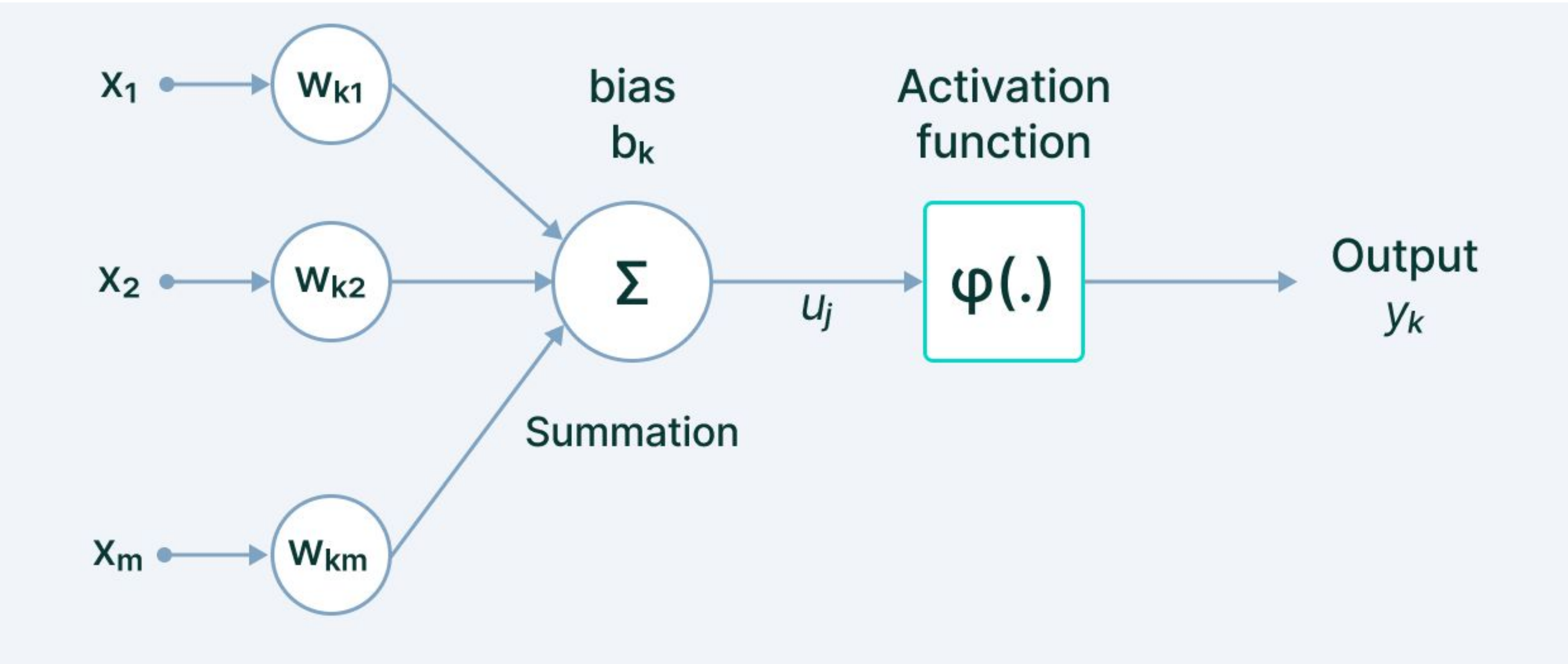
NEURAL NETWORKS

Deep Learning

- Deep Learning is a sub-field of machine learning that works in a manner inspired by the neurons of the brain.
- The learning method based on an artificial neural network imitates the working of the human brain in processing data and deriving meaningful information to make decisions.
- Recall how many neurons in the brain?
  - 86B

Neurons

- The NN architecture is made of individual units called neurons that mimic the biological behavior of the brain.
- Here are the various components of a neuron.



Neuron Components

- **Input** - It is the set of features that are fed into the model for the learning process. For example, the input in object detection can be an array of pixel values pertaining to an image.
- **Weight** - Its main function is to give importance to those features that contribute more towards the learning. It does so by introducing scalar multiplication between the input value and the weight matrix. For example, a negative word would impact the decision of the sentiment analysis model more than a pair of neutral words.
- **Bias** - The role of bias is to **shift** (left or right) the value produced by the activation function. Its role is similar to the role of a constant in a linear function.
- **Transfer function** - The job of the transfer function is to combine multiple inputs into one output value so that the activation function can be applied. It is done by a simple summation of all the inputs to the transfer function.
- **Activation Function** - It introduces non-linearity in the working of perceptrons to consider varying linearity with the inputs. Without this, the output would just be a linear combination of input values and would not be able to introduce non-linearity in the network.
  - A perceptron is an artificial neuron (simplified)

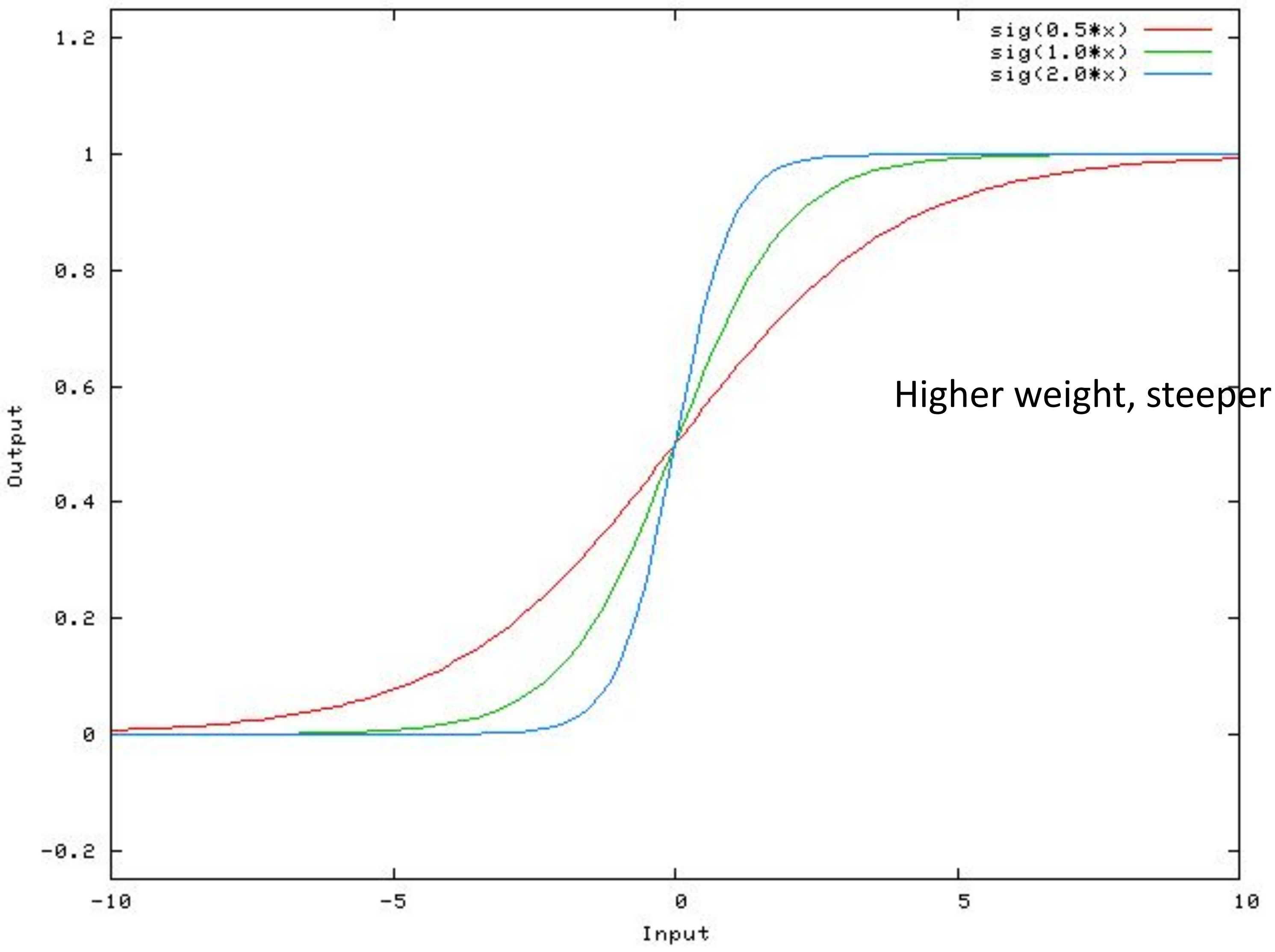
What are Neural Networks?

- Neural Networks are the functional unit of Deep Learning and are known to mimic the behavior of the human brain to solve complex data-driven problems.
- The input data is processed through different layers of artificial neurons stacked together to produce the desired output.
- From speech recognition and person recognition to healthcare and marketing, Neural Networks have been used in a varied set of domains.

NN Standard Components

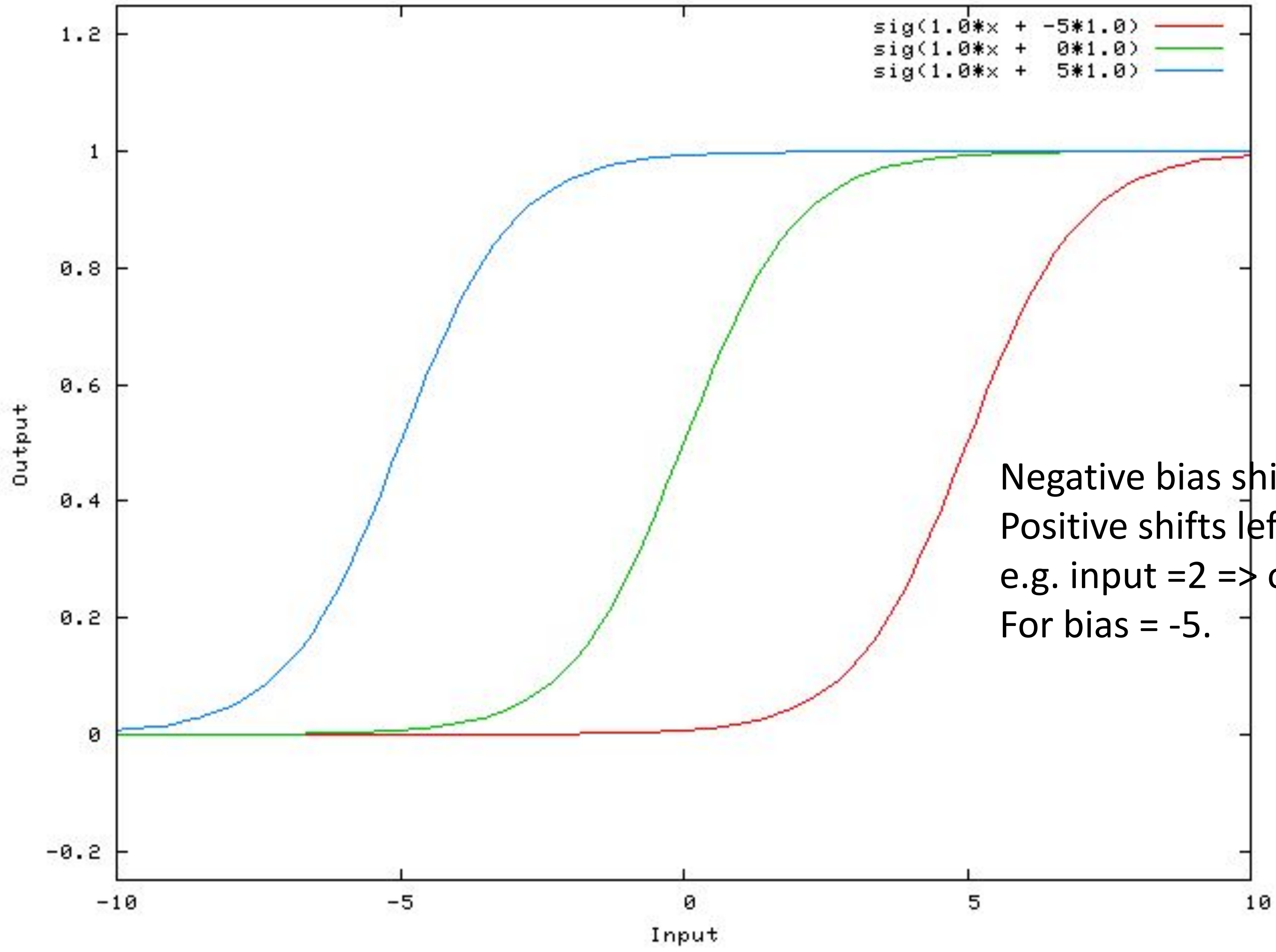
- **A set of nodes**, analogous to neurons, organized in layers.
- **A set of weights** representing the connections between each neural network layer and the layer beneath it. The layer beneath may be another neural network layer, or some other kind of layer.
- **A set of biases**, one for each node.
- **An activation function** that transforms the output of each node in a layer. Different layers may have different activation functions.

Weight Effects (sigmoid)



Higher weight, steeper sigmoid

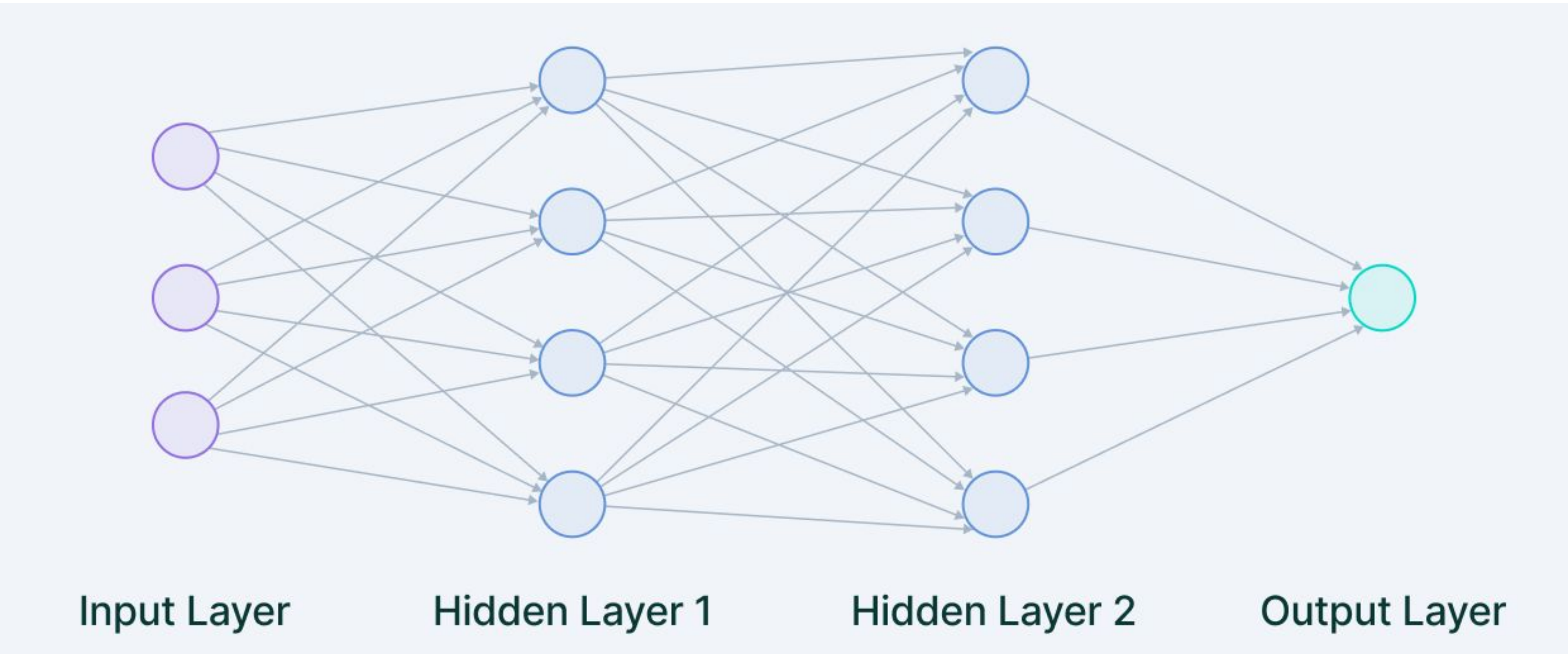
Bias Effects (sigmoid)



Negative bias shifts right,
Positive shifts left,
e.g. input =2 => output = 0
For bias = -5.

Multi-layer neural network

- When multiple neurons are stacked together, they constitute a layer, and multiple layers piled next to each other are called a multi-layer neural network.



NN Layers

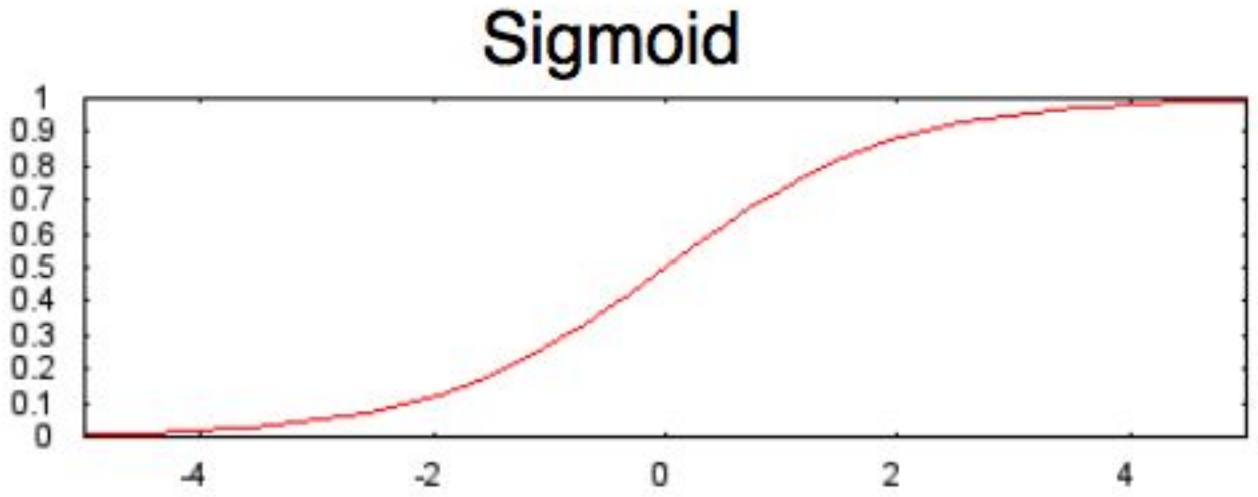
- Input Layer
  - The data that we feed to the model is loaded into the input layer from external sources like a CSV file or a web service. It is the only visible layer in the complete Neural Network architecture that passes the complete information from the outside world without any computation.
- Hidden Layers
  - The hidden layers are what makes deep learning what it is today. They are intermediate layers that do all the computations and extract the features from the data.
  - There can be multiple interconnected hidden layers that account for searching different hidden features in the data. For example, in image processing, the first hidden layers are responsible for higher-level features like edges, shapes, or boundaries. On the other hand, the later hidden layers perform more complicated tasks like identifying complete objects (a car, a building, a person).
- Output Layer
  - The output layer takes input from preceding hidden layers and comes to a final prediction based on the model’s learnings. It is the most important layer where we get the final result.
  - In the case of classification/regression models, the output layer generally has a single node. However, it is completely problem-specific and dependent on the way the model was built.

ACTIVATION FUNCTIONS

Sigmoid (revisit) Activation Function

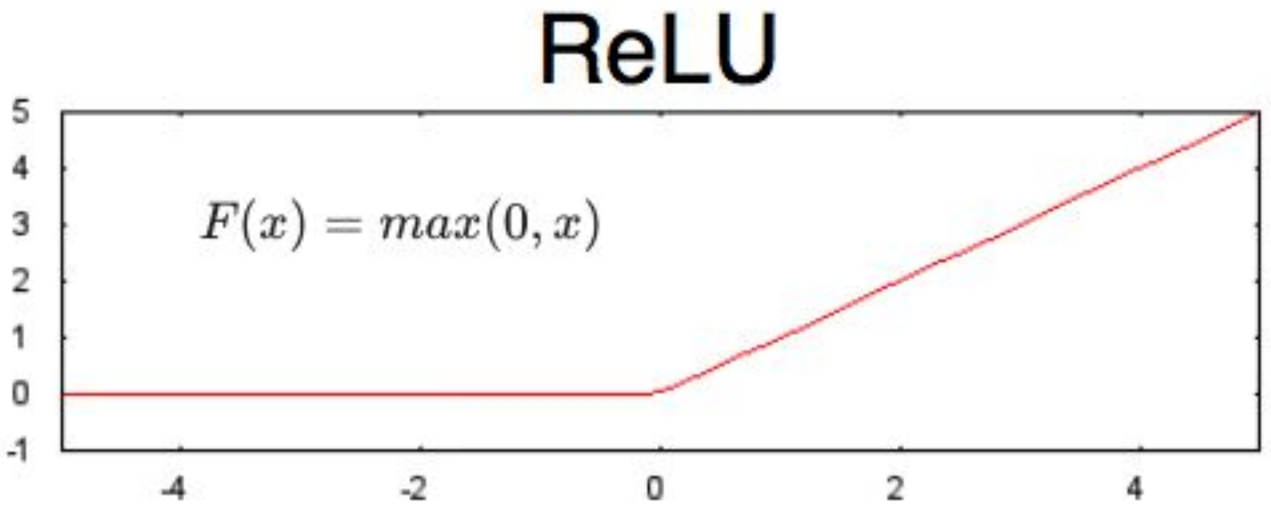
- The following sigmoid activation function converts the weighted sum to a value between 0 and 1.

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



ReLU Activation Function

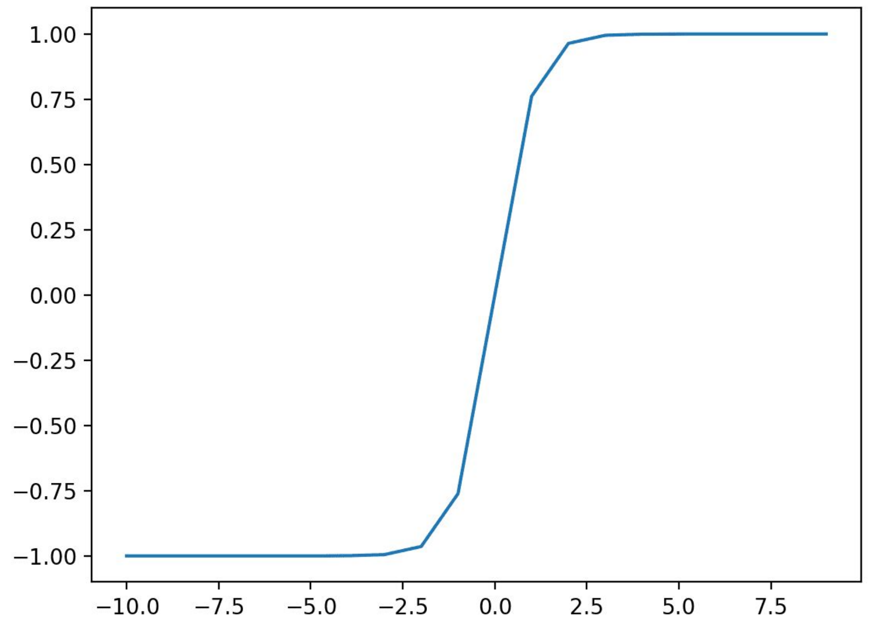
- The following **rectified linear unit (ReLU)** activation function often works a little better than a smooth function like the sigmoid, while also being significantly easier to compute.
- The superiority of ReLU is based on empirical findings, probably driven by ReLU having a more useful range of responsiveness. A sigmoid's responsiveness falls off relatively quickly on both sides.



tanh

- The hyperbolic tangent activation function is also referred to simply as the Tanh (also "tanh" and "TanH") function.
- It is very similar to the sigmoid activation function and even has the same S-shape.
- The function takes any real value as input and outputs values in the range -1 to 1. The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to -1.0.

$$f(x) = \frac{\left(e^{x} - e^{-x}\right)}{\left(e^{x} + e^{-x}\right)} = \tanh(x)$$

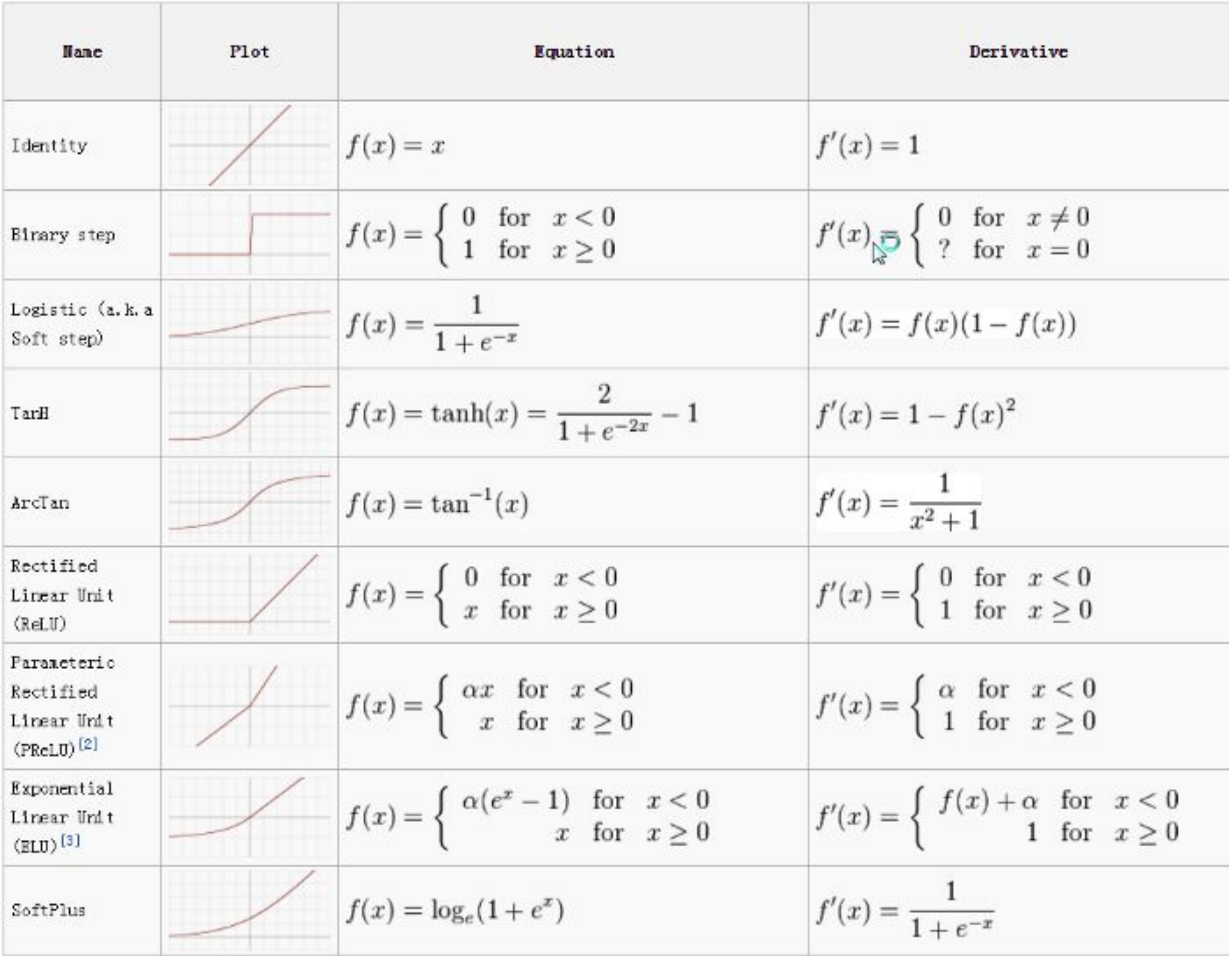


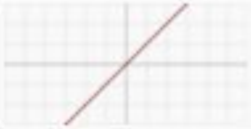
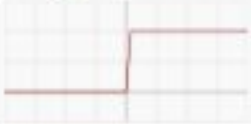
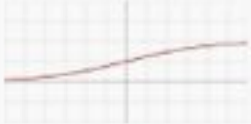
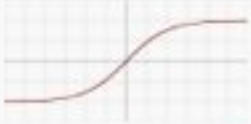
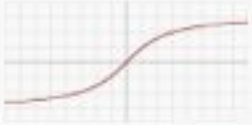
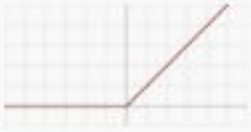
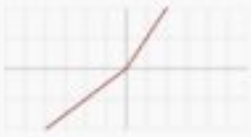
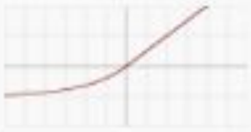
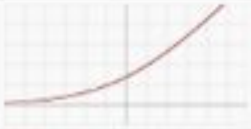
What’s your favorite Activation Function?

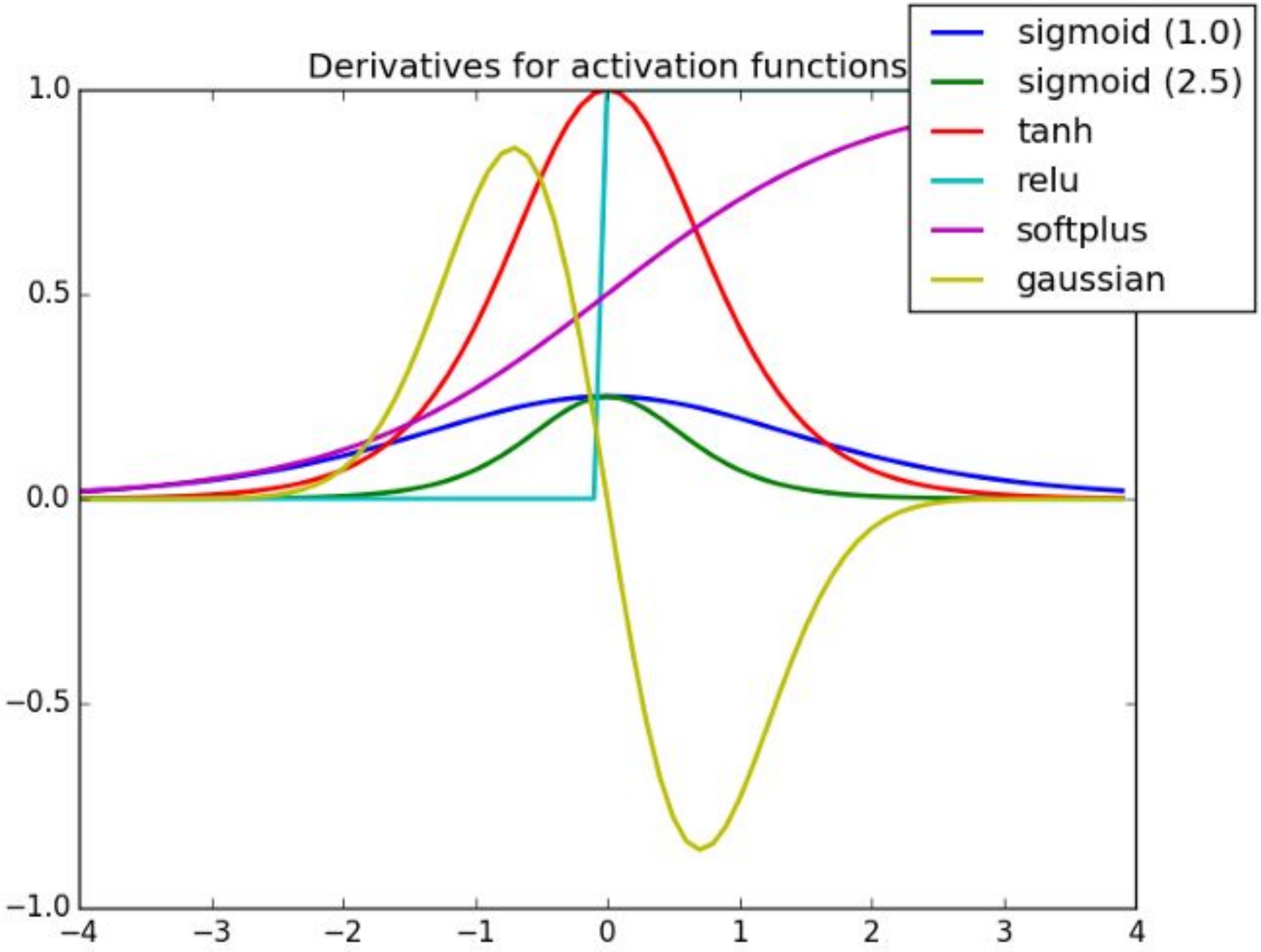
- Any mathematical function can serve as an activation function.
- Suppose that σ represents our activation function (Relu, Sigmoid, or whatever). Consequently, the value of a node in the network is given by the following formula:

$$\sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b)$$

- In general, you want it to be monotonic with known derivative (to gauge convergence)



Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

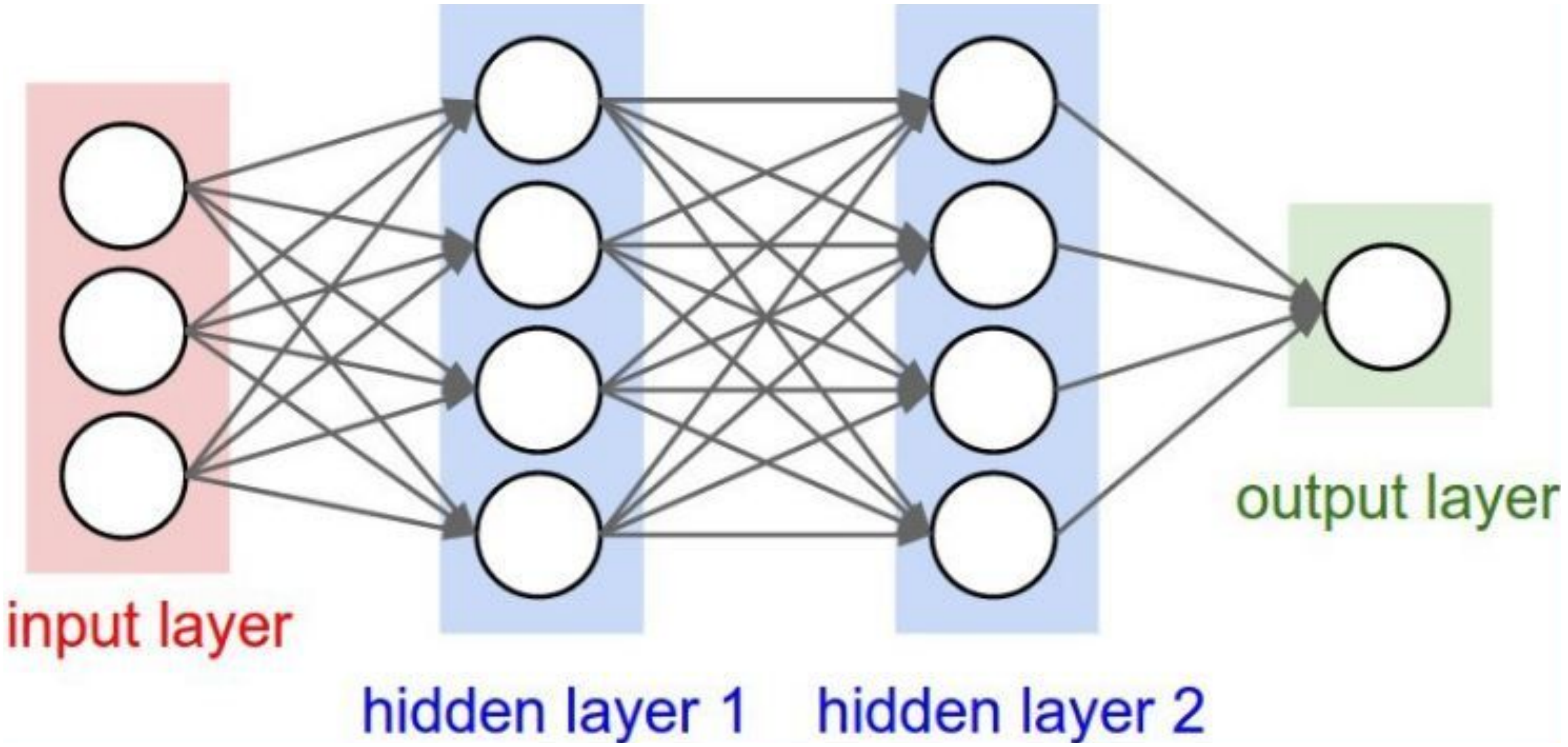

Derivatives for activation functions
sigmoid (1.0)
sigmoid (2.5)
tanh
relu
softplus
gaussian
1.0
0.5
0.0
−0.5
−1.0
−4
−3
−2
−1
0
1
2
3
4

How to Choose an Activation Function?

- Activation functions are a critical part of the design of a neural network.
- The choice of activation function in **the hidden layer** will control how well the network model learns the training dataset.
- The choice of activation function in **the output layer** will define the type of predictions the model can make.
- As such, a careful choice of activation function must be made for each deep learning neural network project.

Layers & Activation Functions

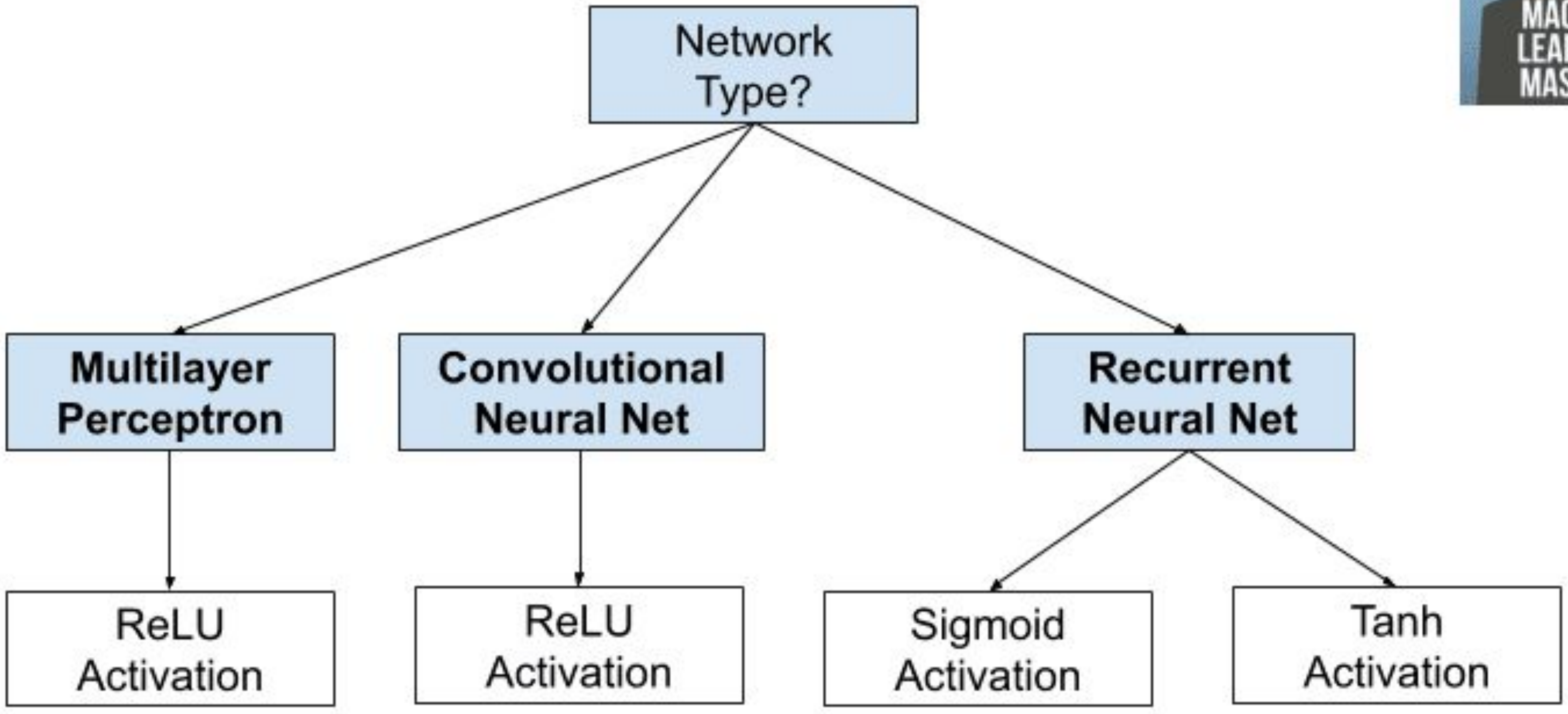
- Output layer has activation function that depends on the output required.
- Hidden layers usually have the same activation function.



How to Choose a Hidden Layer Activation Function

- A neural network will almost always have the same activation function in all hidden layers. It is most unusual to vary the activation function through a network model.
- Traditionally, the sigmoid activation function was the default activation function in the 1990s. Perhaps through the mid to late 1990s to 2010s, the Tanh function was the default activation function for hidden layers.
  - Both the sigmoid and Tanh functions can make the model more susceptible to problems during training, via the so-called **vanishing gradients problem**.
  - You can learn more about this problem in this tutorial:
    - A Gentle Introduction to the Rectified Linear Unit (ReLU)
    - https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/
- The activation function used in hidden layers is typically chosen based on the type of neural network architecture.
  - **In modern neural networks, the default recommendation is to use the rectified linear unit or ReLU**
  - Recurrent networks still commonly use Tanh or sigmoid activation functions, or even both. For example, the LSTM commonly uses the Sigmoid activation for recurrent connections and the Tanh activation for output.
- If you're unsure which activation function to use for your network, try a few and compare the results.

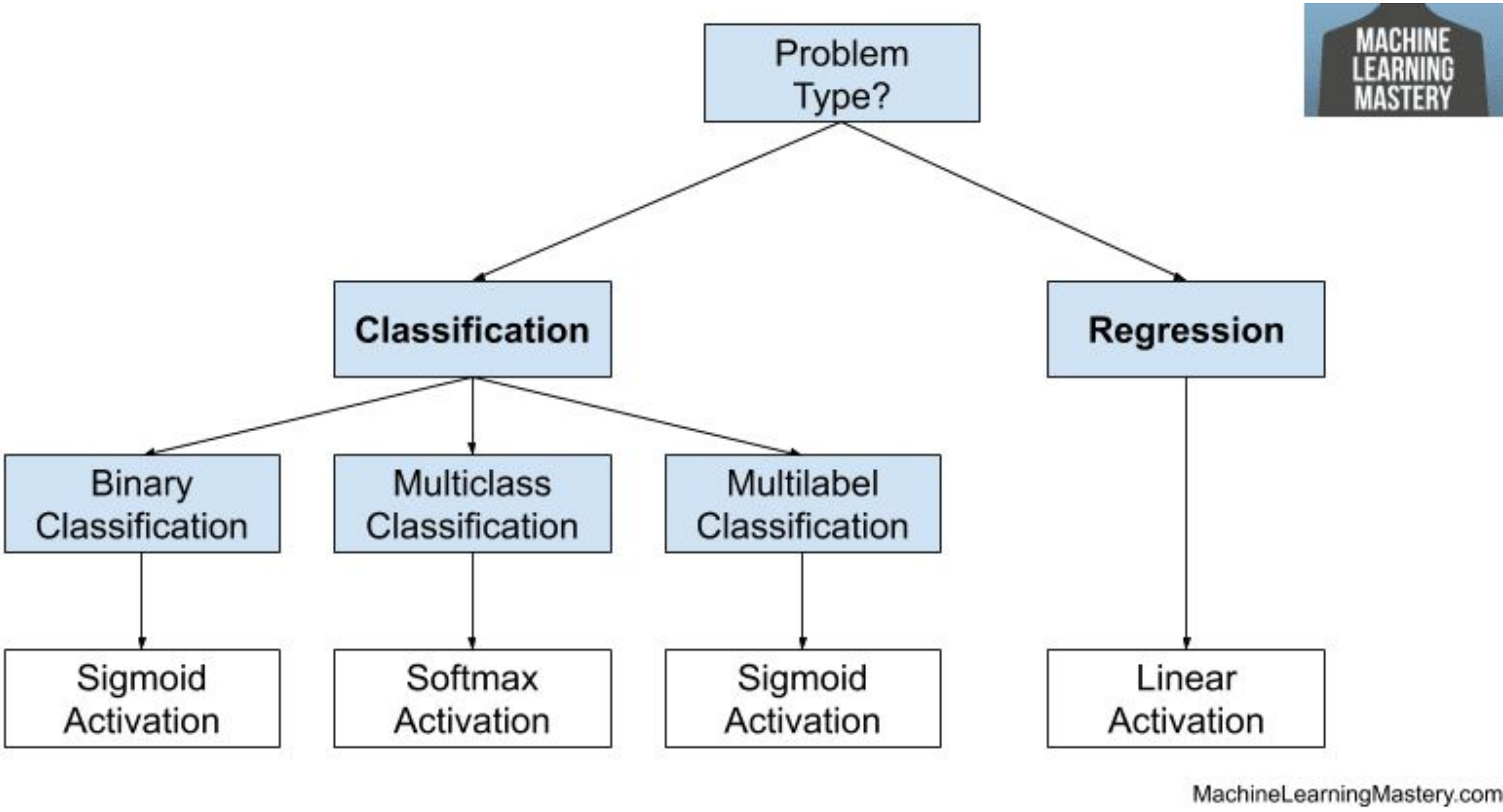
How to Choose a Hidden Layer Activation Function



How to Choose an Output Activation Function

- The **output layer** is the layer in a neural network model that directly outputs a prediction.
  - All feed-forward neural network models have an output layer.
- There are perhaps three activation functions you may want to consider for use in the output layer; they are:
  - Linear, eg for regression
  - Logistic (Sigmoid), eg for binary classification
  - Softmax, eg for multi class classification
- This is not an exhaustive list of activation functions used for output layers, but they are the most commonly used.

How to Choose an Output Activation Function



How to Choose an Output Activation Function

- Regression – predicting a numerical variable
  - one node
  - linear activation
- Classification – predicting class membership (categorical value)
  - Binary - two classes
    - one node
    - sigmoid activation
  - Multiclass – multiple (two or more) mutually **exclusive** classes
    - one node per class
    - softmax activation (sum of probabilities = 1)
  - Multilabel – multiple (two or more) mutually **inclusive** classes
    - One node per class
    - sigmoid activation (probability it has label)

NNs with Activation Functions

- Now that we've added an activation function, adding layers has more impact.
- Stacking nonlinearities on nonlinearities lets us model very complicated relationships between the inputs and the predicted outputs.
- Each layer is effectively learning a more complex, higher-level function over the raw inputs.
- If you'd like to develop more intuition on how this works, see Chris Olah's excellent blog post.
  - http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Further Reading

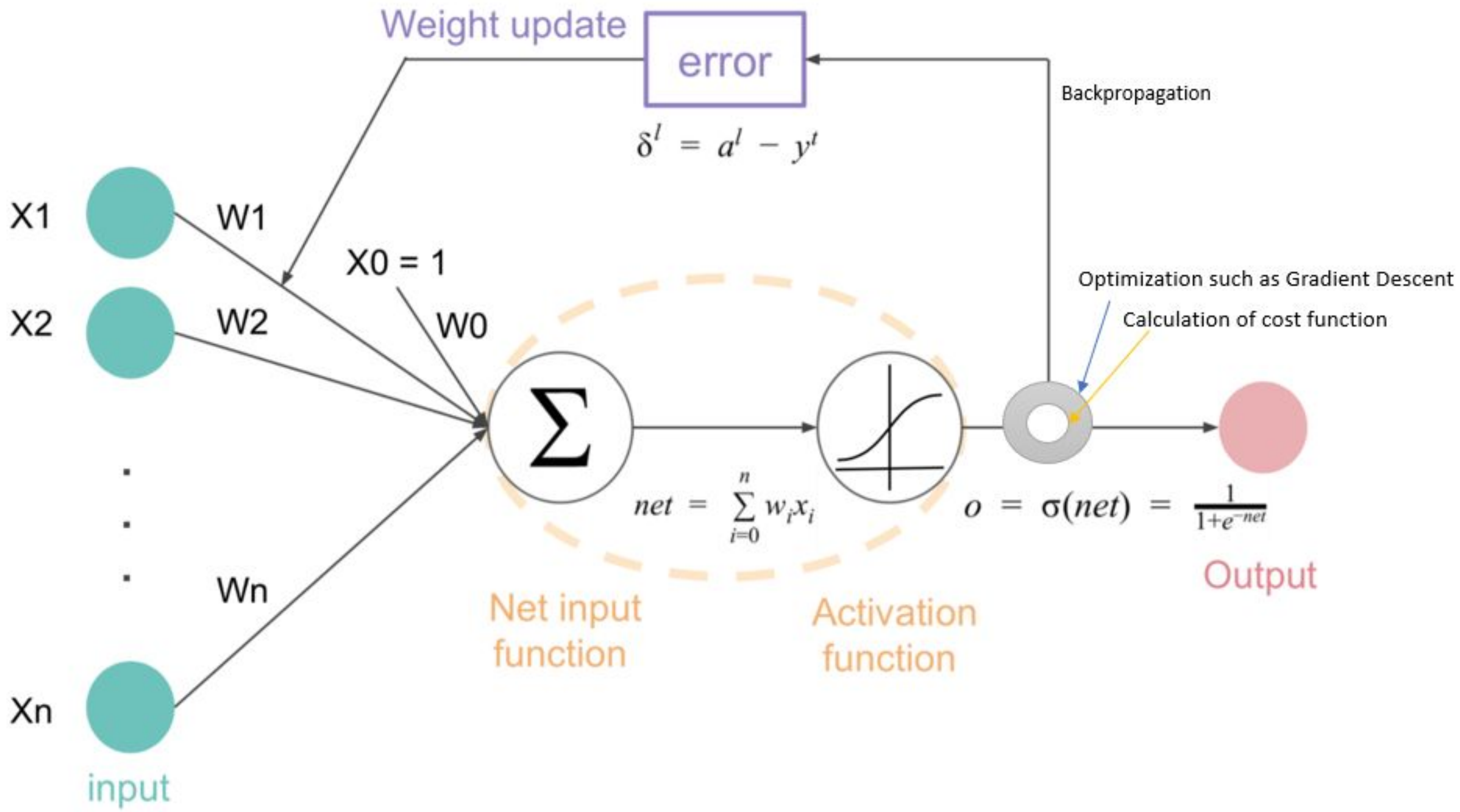
- https://www.v7labs.com/blog/neural-networks-activation-functions
- https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/
- https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

BACKPROPAGATION

Backpropagation

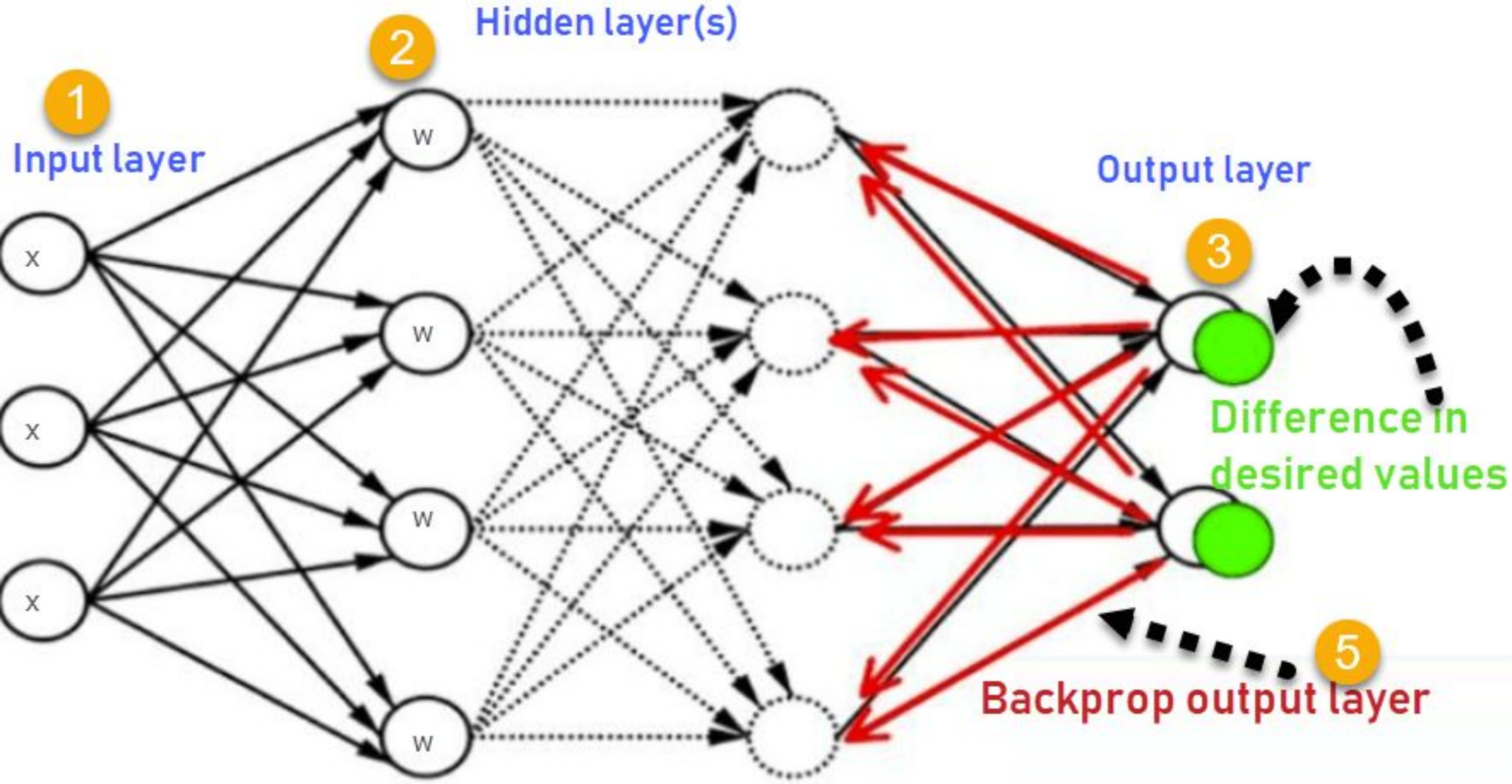
- The primary algorithm for performing gradient descent on neural networks.
  - Backpropagation is a technique to improve the performance of the network. It backpropagates the error and updates the weights to reduce the error.
- The goals of backpropagation are straightforward: adjust each weight in the network in proportion to how much it contributes to overall error.
- If we iteratively reduce each weight’s error, eventually we’ll have a series of weights that produce good predictions.
- First, the output values of each node are calculated (and cached) in a forward pass.
- Then, the partial derivative of the error with respect to each parameter is calculated in a backward pass through the graph.
- Details: https://ml-cheatsheet.readthedocs.io/en/latest/backpropagation.html

Backpropagation

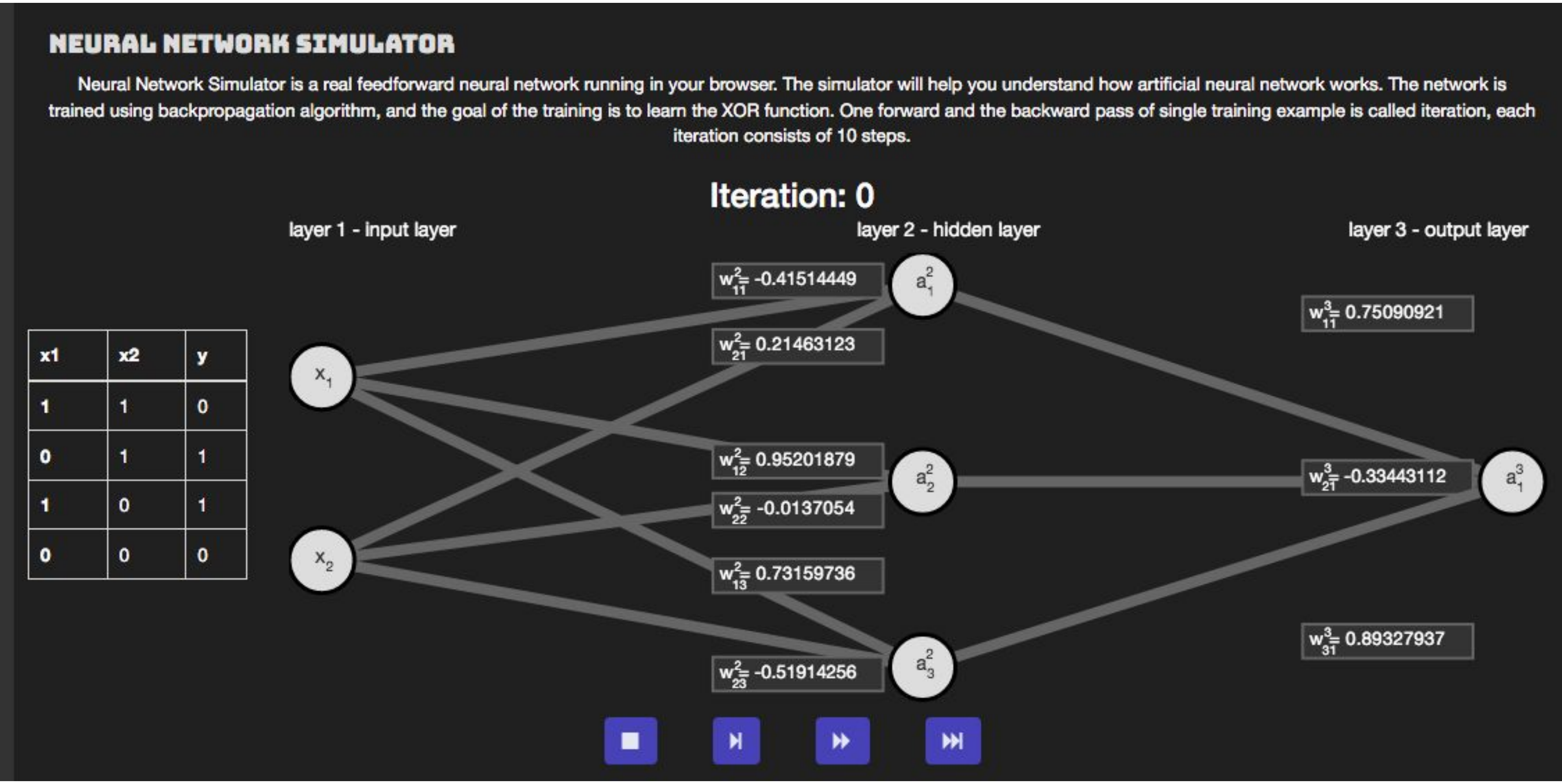

Weight update
error
$\delta^l = a^l - y^t$
Backpropagation
X1
W1
X0 = 1
X2
W2
W0
Optimization such as Gradient Descent
Calculation of cost function
Σ
$net = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(net) = \frac{1}{1+e^{-net}}$
Output
Wn
Net input function
Activation function
Xn
input

https://intellipaat.com/blog/tutorial/artificial-intelligence-tutorial/back-propagation-algorithm/

Backpropogation



NN Backpropagation Simulation



NEURAL NETWORK SIMULATOR

Neural Network Simulator is a real feedforward neural network running in your browser. The simulator will help you understand how artificial neural network works. The network is trained using backpropagation algorithm, and the goal of the training is to learn the XOR function. One forward and the backward pass of single training example is called iteration, each iteration consists of 10 steps.

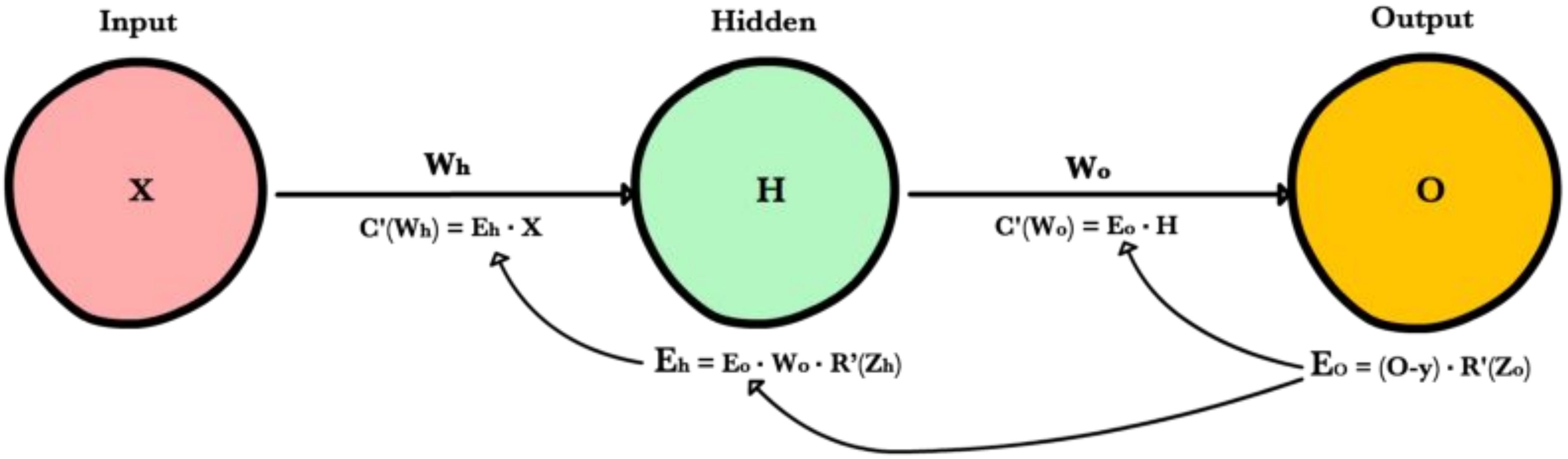
Iteration: 0

x1	x2	y
1	1	0
0	1	1
1	0	1
0	0	0

https://www.mladdict.com/neural-network-simulator

Backpropagation

- Back-propagation is called like this because to calculate the derivative you use the chain rule from the last layer (which is the one directly connected to the loss function, as it is the one that provides the prediction) to the first layer, which is the one that takes the input data. You are "moving from back to front”.



Backpropagation Failure Scenarios

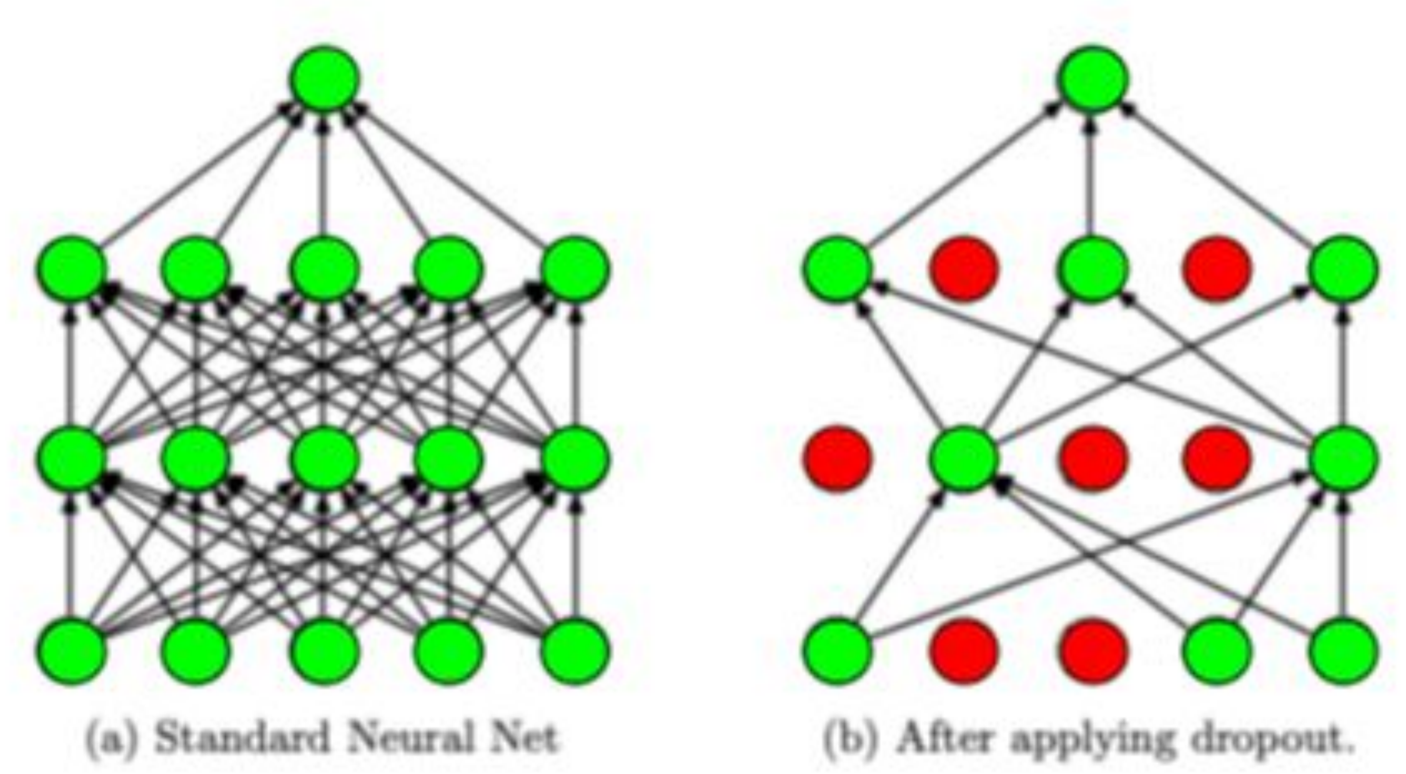
- Vanishing Gradients
  - The gradients for the lower layers (closer to the input) can become **very small**. In deep networks, computing these gradients can involve taking the product of many small terms.
  - When the gradients vanish toward 0 for the lower layers, these layers train very slowly, or not at all.
  - The ReLU activation function can help prevent vanishing gradients.
- Exploding Gradients
  - If the weights in a network are very large, then the gradients for the lower layers involve products of many large terms. In this case you can have exploding gradients: gradients that get **too large** to converge.
  - Batch normalization can help prevent exploding gradients, as can lowering the learning rate.
    - Batch normalization smooths the optimization landscape, it gets rid of the extreme gradients that accumulate, leading to the elimination of the major weight fluctuations that result from gradient build-up. This dramatically stabilizes learning.
    - https://towardsdatascience.com/batch-normalization-the-greatest-breakthrough-in-deep-learning-77e64909d81d

Backpropagation Failure Scenarios

- Dead ReLU Units
  - Once the weighted sum for a ReLU unit falls below 0, the ReLU unit can get stuck. It outputs 0 activation, contributing nothing to the network's output, and gradients can no longer flow through it during backpropagation. With a source of gradients cut off, the input to the ReLU may not ever change enough to bring the weighted sum back above 0.
  - Lowering the learning rate can help keep ReLU units from dying.
- Dropout Regularization
  - Yet another form of regularization, called Dropout, is useful for neural networks. It works by **randomly** "dropping out" unit activations in a network for a single gradient step. The more you drop out, the stronger the regularization:
  - 0.0 = No dropout regularization.
  - 1.0 = Drop out everything. The model learns nothing.
  - Values between 0.0 and 1.0 = More useful.

Dropout Regularization

- "Dropout" in machine learning refers to the process of randomly ignoring certain nodes in a layer during training.


(a) Standard Neural Net (b) After applying dropout.

https://medium.com/analytics-vidhya/a-simple-introduction-to-dropout-regularization-with-code-5279489dda1e

Vanishing Gradients Problem

- The vanishing gradients problem is one example of unstable behavior that you may encounter when training a deep neural network.
- It describes the situation where a deep multilayer network is unable to propagate useful gradient information from the output end of the model back to the layers near the input end of the model.
- The result is the general inability of models with many layers to learn on a given dataset, or for models with many layers to prematurely converge to a poor solution.
- Details:
  - https://machinelearningmastery.com/how-to-fix-vanishing-gradients-using-the-rectified-linear-activation-function/

EXAMPLES

A First Neural Network

- Simulation: https://developers-dot-devsite-v2-prod.appspot.com/machine-learning/crash-course/playground/
- Task 1: The model as given combines our two input features into a single neuron. Will this model learn any nonlinearities? Run it to confirm your guess.
- Task 2: Try increasing the number of neurons in the hidden layer from 1 to 2, and also try changing from a Linear activation to a nonlinear activation like ReLU. Can you create a model that can learn nonlinearities? Can it model the data effectively?
- Task 3: Try increasing the number of neurons in the hidden layer from 2 to 3, using a nonlinear activation like ReLU. Can it model the data effectively? How does model quality vary from run to run?
- Task 4: Continue experimenting by adding or removing hidden layers and neurons per layer. Also feel free to change learning rates, regularization, and other learning settings. What is the smallest number of neurons and layers you can use that gives test loss of 0.177 or lower?
- Does increasing the model size improve the fit, or how quickly it converges? Does this change how often it converges to a good model? For example, try the following architecture:
  - First hidden layer with 3 neurons.
  - Second hidden layer with 3 neurons.
  - Third hidden layer with 2 neurons.

Neural Net Initialization

- This exercise uses the XOR data again, but looks at the repeatability of training Neural Nets and the importance of initialization.
- Task 1: Run the model as given four or five times. Before each trial, hit the Reset the network button to get a new random initialization. (The Reset the network button is the circular reset arrow just to the left of the Play button.) Let each trial run for at least 500 steps to ensure convergence. What shape does each model output converge to? What does this say about the role of initialization in non-convex optimization?
- Task 2: Try making the model slightly more complex by adding a layer and a couple of extra nodes. Repeat the trials from Task 1. Does this add any additional stability to the results?

Neural Net Spiral

- This data set is a noisy spiral. Obviously, a linear model will fail here, but even manually defined feature crosses may be hard to construct.
- Task 1: Train the best model you can, using just X1 and X2. Feel free to add or remove layers and neurons, change learning settings like learning rate, regularization rate, and batch size. What is the best test loss you can get? How smooth is the model output surface?
- Task 2: Even with Neural Nets, some amount of feature engineering is often needed to achieve best performance. Try adding in additional cross product features or other transformations like sin(X1) and sin(X2). Do you get a better model? Is the model output surface any smoother?

NN TYPES

NN Types

- Standard Neural Networks
  - The Perceptron
  - Feed-Forward Networks
  - Residual Networks (ResNet)
- Recurrent Neural Networks (RNNs)
  - The Recurrent Neural Networks (RNN)
  - The Long Short Term Memory Network (LSTM)
  - Echo State Networks (ESN)
- Convolutional Neural Networks (CNNs)
  - The Deconvolutional Neural Networks (DNN)
  - AlexNet, Overfeat, VGG, Network-in-network, GoogLeNet and Inception, SqueezeNet, Xception, MobileNets, Capsule Networks
- Generative Adversarial Network (GAN)
- Transformer Neural Networks

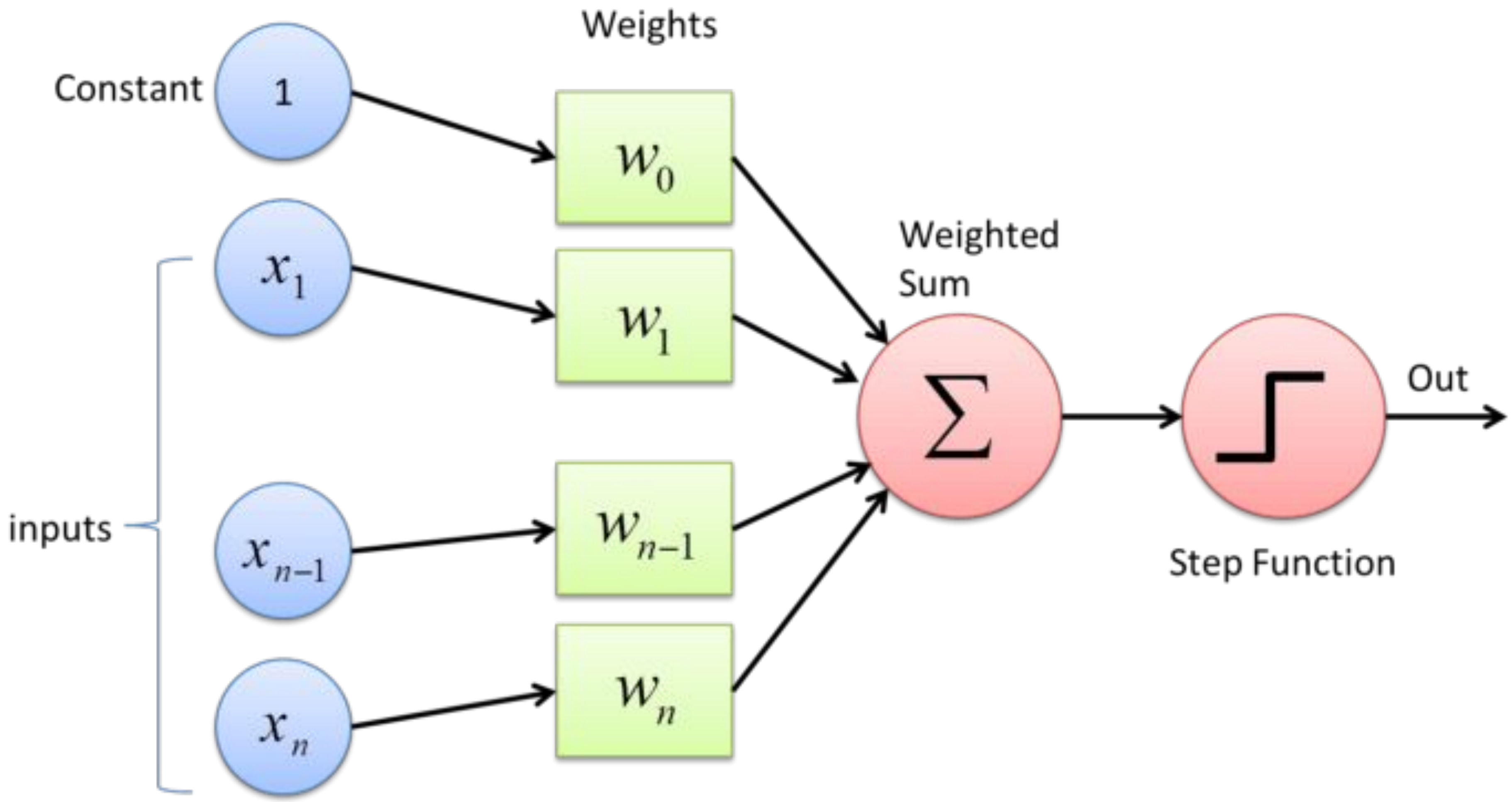
NN Type Overview

- Each Neural Networks architecture has its own set of pros and cons.
- Standard Neural Networks like Feed-Forward Neural Networks are most commonly used in solving classification and regression problems related to simple structured data.
- Recurrent Neural Networks are much more powerful in terms of remembering information for a long time and are used in sequential data like text, audio, video etc.
- Recent research shows that Transformers based on Attention Mechanism outperform RNNs and have almost replaced RNNs in every field.
- For complex data like images we can use ConvNets in classification tasks and for generation of images or style transfer related tasks Generative Adversarial Networks performs the best.

Perceptron

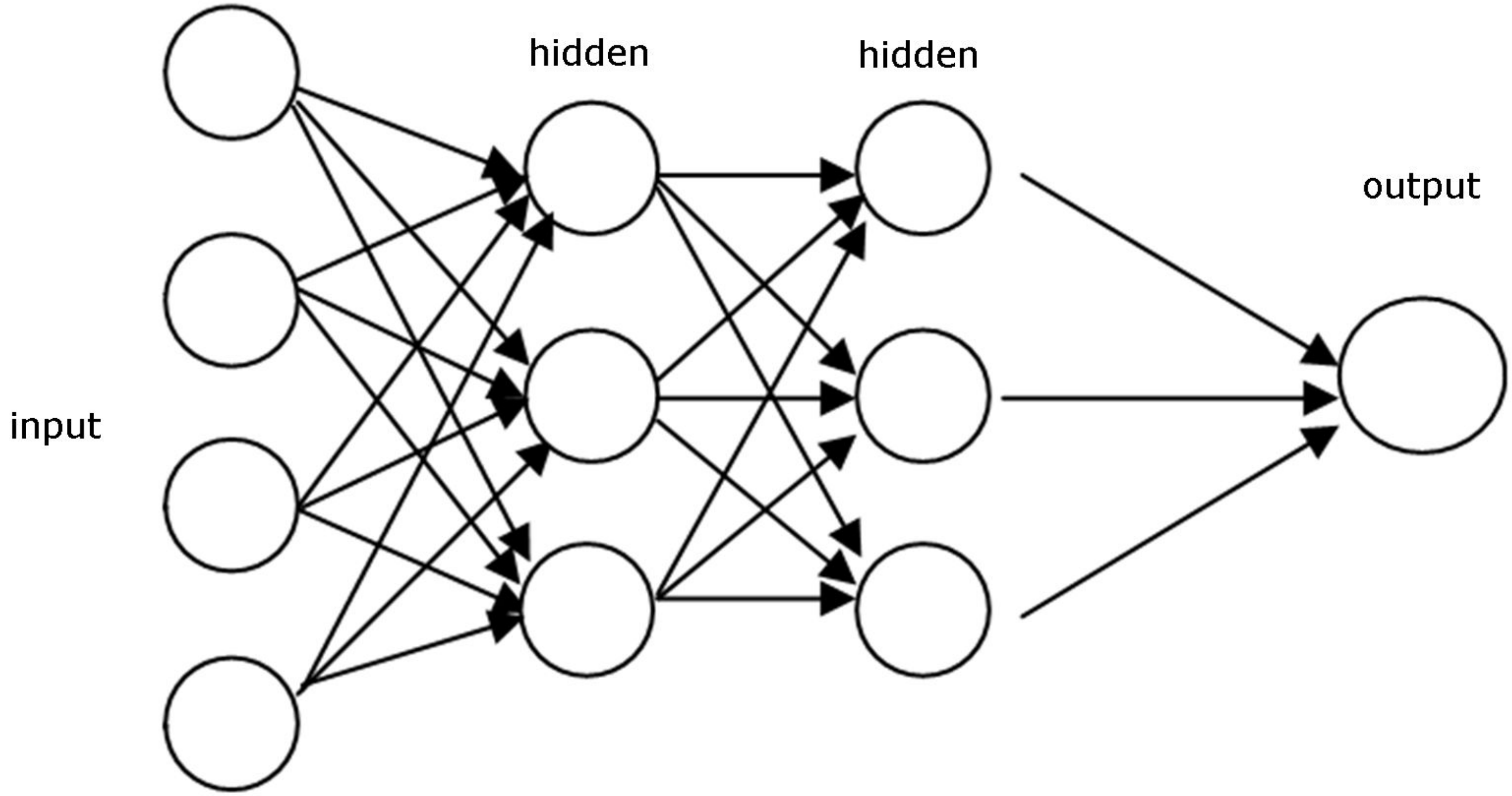
- Perceptron is the simplest Neural Network architecture.
  - Single or multi layer neural network
- It is a type of Neural Network that takes a number of inputs, applies certain mathematical operations on these inputs, and produces an output. It takes a vector of real values inputs, performs a linear combination of each attribute with the corresponding weight assigned to each of them.
- The weighted input is summed into a single value and passed through an activation function.
- These perceptron units are combined to form a bigger Artificial Neural Network architecture.
- Details: https://towardsdatascience.com/perceptron-learning-algorithm-d5db0deab975

Perceptron



Details: https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

Feed-Forward Neural Networks



Feed-Forward Neural Networks

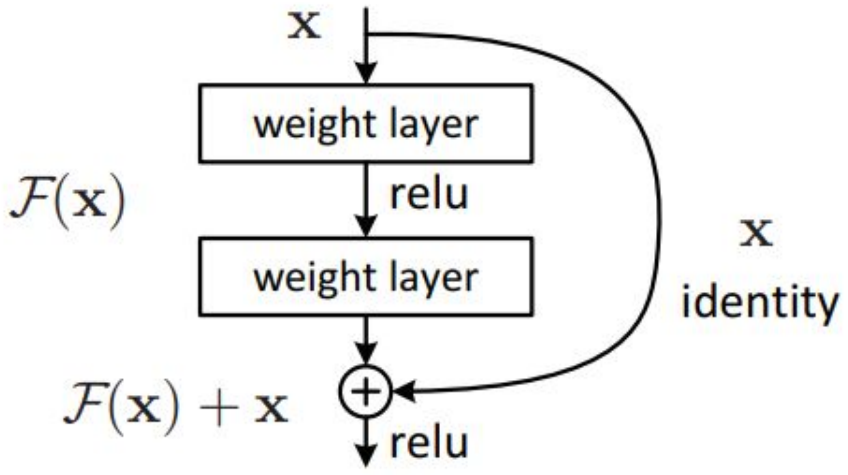
- Perceptron represents how a single neuron works. But—
  - What about a series of perceptrons stacked in a row and piled in different layers? How does the model learn then?
- It is a multi-layer Neural Network, and, as the name suggests, the information is passed in the forward direction—from left to right.
- In the forward pass, the information comes inside the model through the input layer, passes through the series of hidden layers, and finally goes to the output layer. This Neural Networks architecture is forward in nature—the information does not loop with two hidden layers.
- The later layers give no feedback to the previous layers. The basic learning process of Feed-Forward Networks remain the same as the perceptron.

Residual Networks (ResNet)

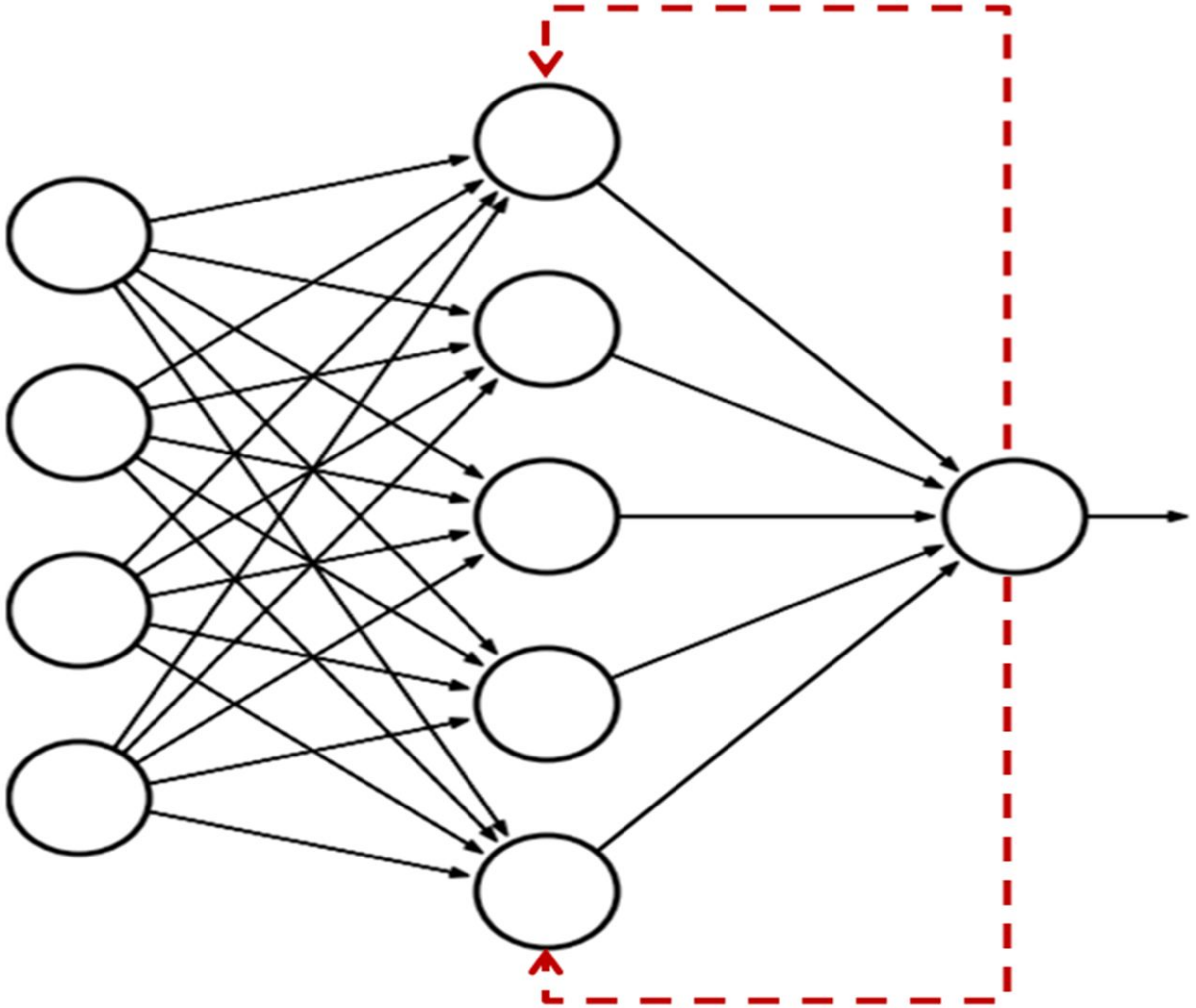
- how to decide on the number of layers in our neural network architecture?
- Very deep Neural Networks are extremely difficult to train due to vanishing and exploding gradient problems.
- ResNets provide an alternate pathway for data to flow to make the training process much faster and easier.
- This is different from the feed-forward approach of earlier Neural Networks architectures.
- The core idea behind ResNet is that a deeper network can be made from a shallow network by copying weight from the shallow counterparts using identity mapping.
- The data from previous layers is fast-forwarded and copied much forward in the Neural Networks. This is what we call skip connections first introduced in Residual Networks to resolve vanishing gradients.

ResNet

- In this network we use a technique called skip connections . The skip connection skips training from a few layers and connects directly to the output.
- The advantage of adding this type of skip connection is because if any layer hurt the performance of architecture then it will be skipped by regularization.
- So, this results in training very deep neural network without the problems caused by vanishing/exploding gradient.


x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Recurrent Neural Networks



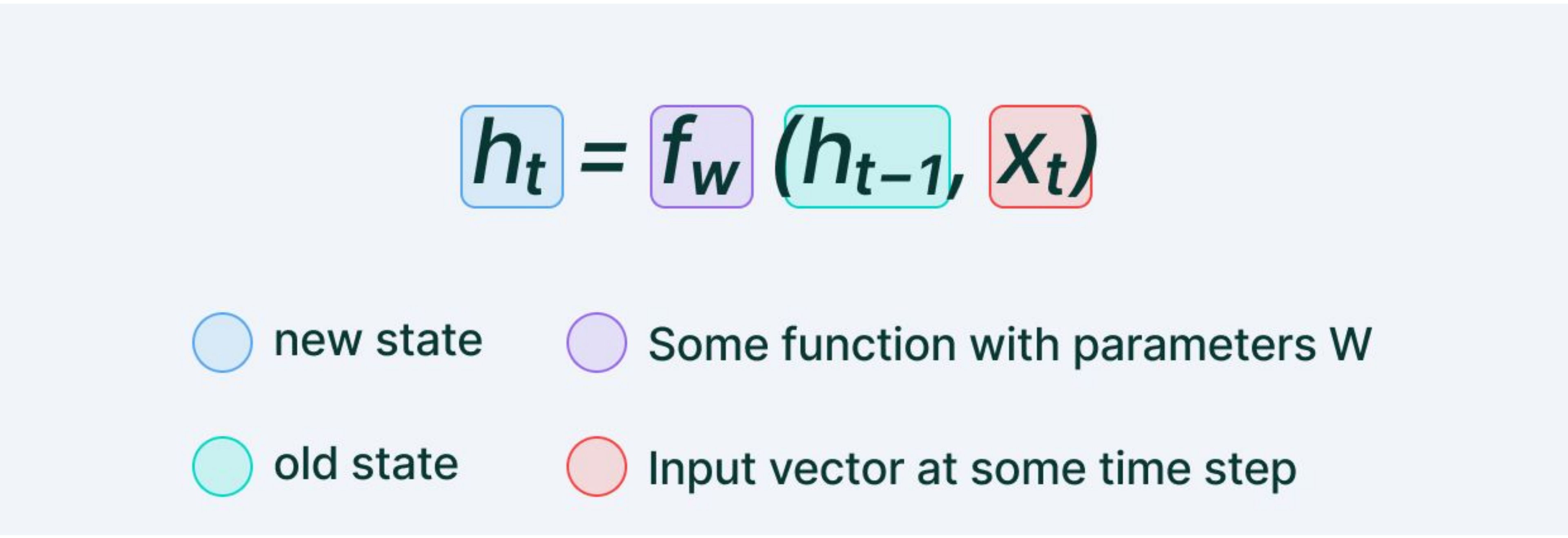
Recurrent Neural Networks (RNN)

- Recurrent Neural Networks have the power to remember what it has learned in the past and apply it in future predictions.
- The input is in the form of sequential data that is fed into the RNN, which has a hidden internal state that gets updated every time it reads the following sequence of data in the input.
- The internal hidden state will be fed back to the model. The RNN produces some output at every timestamp.

Recurrent Neural Networks (RNN)

- The basic deep learning architecture has a fixed input size, and this acts as a blocker in scenarios where the input size is not fixed.
- Also, the decisions made by the model were based on the current input with no memory of the past.
- Recurrent Neural Networks work very well with sequences of data as input.
- Its functionality can be seen in solving NLP problems like sentiment analysis, spam filters, time series problems like sales forecasting, stock market prediction, etc.

RNNs



$$h_t = f_W(h_{t-1}, x_t)$$

🔵 new state

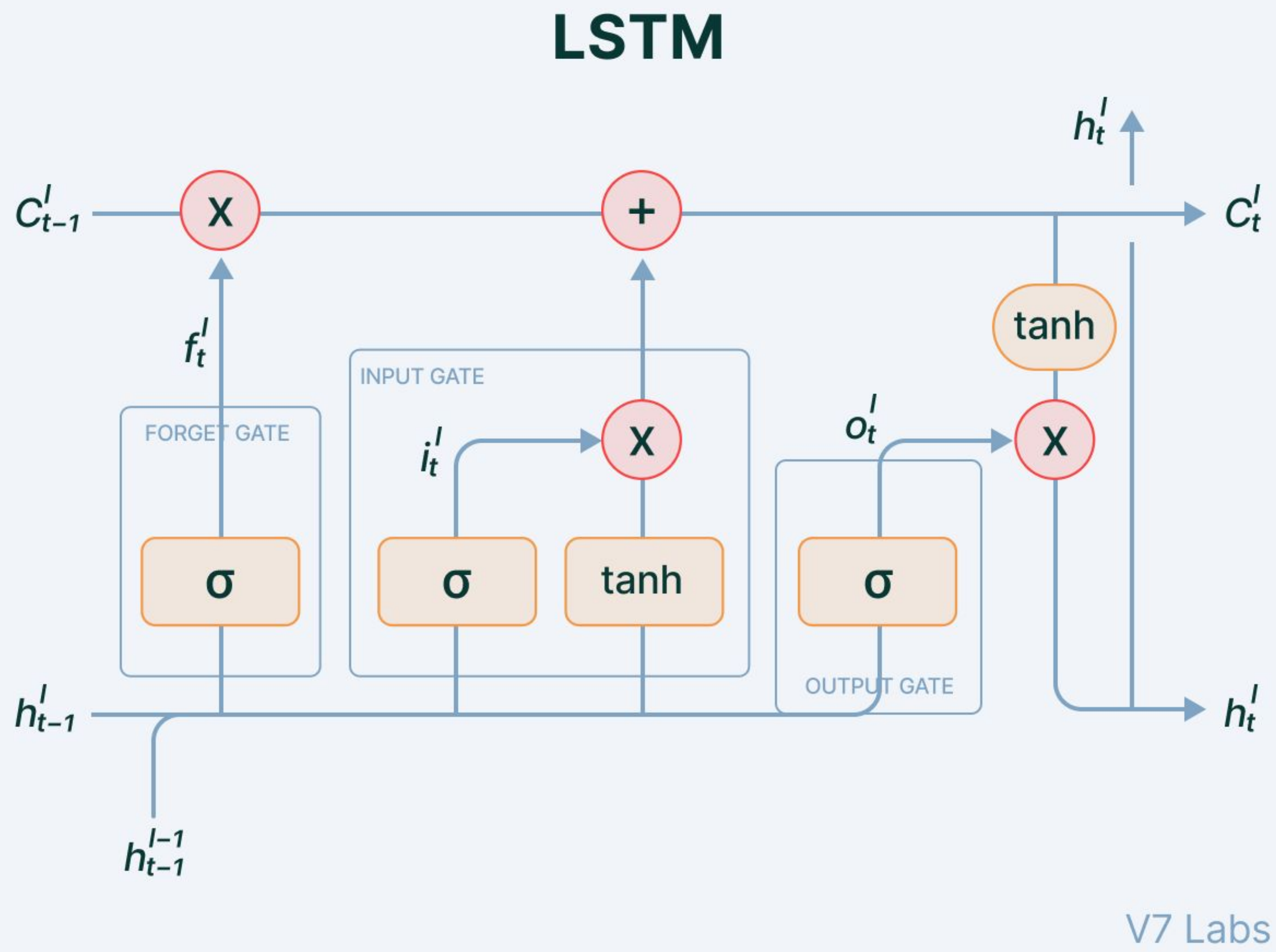
🟣 Some function with parameters W

🟢 old state

🔴 Input vector at some time step

The Long Short Term Memory Network (LSTM)

- In RNN each of our predictions looked only one timestamp back, and it has a very short-term memory. It doesn't use any information from further back.
- To rectify this, we can take our Recurrent Neural Networks structure and expand it by adding some more pieces to it.
- The critical part that we add to this Recurrent Neural Networks is **memory**. We want it to be able to remember what happened many timestamps ago. To achieve this, we need to add extra structures called gates to the artificial neural network structure.


LSTM
C^l_{t-1}
C^l_t
h^l_t
f^l_t
INPUT GATE
FORGET GATE
i^l_t
o^l_t
X
+
tanh
σ
OUTPUT GATE
h^l_{t-1}
h^{l-1}_{t-1}
V7 Labs

LSTM Components

- Cell state (c_t): It corresponds to the long-term memory content of the network.
- Forget Gate: Some information in the cell state is no longer needed and is erased. The gate receives two inputs, x_t (current timestamp input) and h_t-1 (previous cell state), multiplied with the relevant weight matrices before bias is added. The result is sent into an activation function, which outputs a binary value that decides whether the information is retained or forgotten.
- Input gate: It decides what piece of new information is to be added to the cell state. It is similar to the forget gate using the current timestamp input and previous cell state with the only difference of multiplying with a different set of weights.
- Output gate: The output gate's job is to extract meaningful information from the current cell state and provide it as an output.

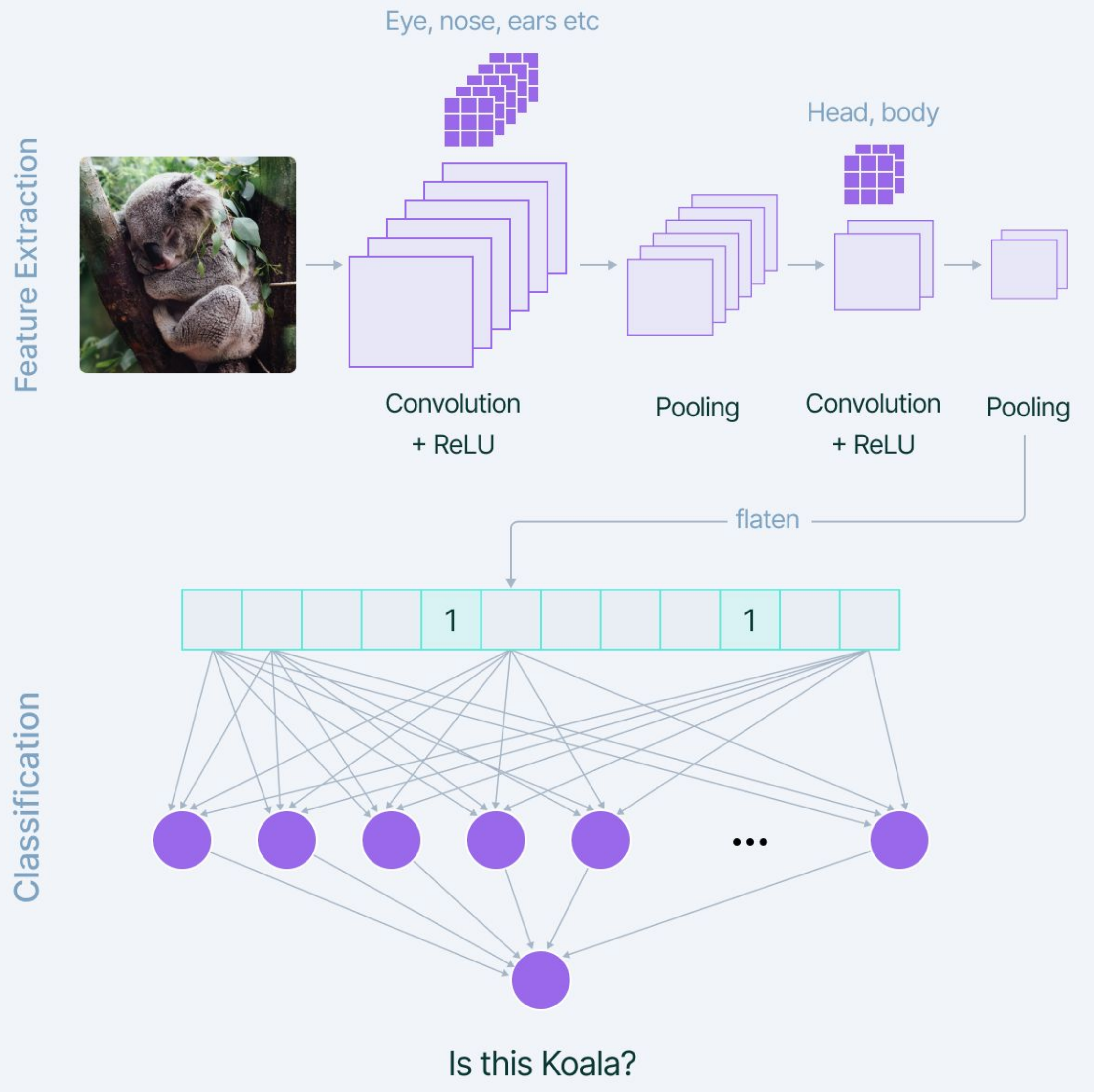
Echo State Networks (ESN)

- Echo state Networks is a RNN with sparsely connected hidden layers with typically 1% connectivity.
- The connectivity and weight of hidden neurons are fixed and randomly assigned. The only weight that needs to be learned is that of the output layer. It can be seen as a linear model of the weighted input passed through all the hidden layers and the targeted output. The main idea is to keep the early layers fixed.
- The only weights that will be modified during the training are for the synopsis that connects the hidden layers to the output layers. This makes the loss function simple and easy to differentiate.
- The training becomes uncomplicated, assuming linear output units. The only thing to keep in mind is to set the random connections very carefully.

Convolutional Neural Networks (CNN)

- Convolutional Neural Networks is a type of Feed-Forward Neural Networks used in tasks like image analysis, natural language processing, and other complex image classification problems.
- A CNN has hidden layers of convolutional layers that form the base of ConvNets.
- Features refer to minute details in the image data like edges, borders, shapes, textures, objects, circles, etc.
- At a higher level, convolutional layers detect these patterns in the image data with the help of filters. The higher-level details are taken care of by the first few convolutional layers.
- The deeper the network goes, the more sophisticated the pattern searching becomes.
- For example, in later layers rather than edges and simple shapes, filters may detect specific objects like eyes or ears, and eventually a cat, a dog, and what not.

Feature Extraction & Classification



V7 Labs

CNN Filters

- When adding a convolutional layer to a network, we need to specify the number of filters.
- A **filter** can be thought of as a relatively small matrix for which we decide the number of rows and columns this matrix has.
- The value of this feature matrix is initialized with random numbers. When this convolutional layer receives pixel values of input data, the filter will convolve over each patch of the input matrix.
- The output of the convolutional layer is usually passed through the **ReLU** activation function to bring non-linearity to the model. It takes the feature map and replaces all the negative values with zero.
- Pooling is a very important step in the ConvNets as it reduces the computation and makes the model tolerant towards distortions and variations. A Fully Connected Dense Neural Networks would use a flattened feature matrix and predict according to the use case.

Generative Adversarial Networks (GANs)

- Generative modeling comes under the umbrella of unsupervised learning, where new/synthetic data is generated based on the patterns discovered from the input set of data.
- GAN is a generative model and is used to generate entirely new synthetic data by learning the pattern and hence is an active area of AI research.
- They have two components - a generator and a discriminator that work in a competitive fashion.
  - The generator's job is to create synthetic data based on the model's features during its learning phase. It takes in random data as input and returns a generated image after performing certain transformations.
  - The discriminator acts as a critic and has an overall idea of the problem domain with a clear understanding of generated images.
  - These generated images are classified into fake/genuine images by the discriminator.

GANs

- The discriminator returns a probabilistic prediction for the images to be noisy/free-of-noise by a value in the range of 0 to 1, where 1 is an authentic image and 0 a fake image.
- The generator network produces samples based on its learning.
- Its adversary, the discriminator, strives to distinguish between samples from the training data and samples produced from the generator. There is feedback from the discriminator fed to the generator to improve the performance.
- When the discriminator successfully distinguishes between real and fake examples, the component is working well and no changes need to be applied to its parameters.
- The generator is given a penalty when it fails to generate an image as real such that it could fool the discriminator. However, if it succeeds in making the discriminator categorize the generated image as real, it shows that the training of the generator is moving in the right direction. So the ultimate aim for the generator is to fool the discriminator while for the discriminator is to surpass the accuracy of the generator.
- It is used in scenarios like predicting the next frame in a video, text to image generation, image to image translation like style transfer, denoising of the image, etc.

Transformer Networks

- The truth is RNNs are slow and take too much time in training.
  - They are not very good with large sequenced data and lead to vanishing gradients.
  - LSTMs that were introduced to bring memory in the RNN became even slower to train.
  - For both RNN and LSTM, we need to feed the data sequentially or serially. This does not make use of GPUs.
- How to parallelize the training on sequential data?
  - The answer is Transformers.
  - These networks employ an encoder-decoder structure with a difference that the input data can be passed in parallel.
  - In RNN structure, one word at a time was passed through the input layer. But in Transformers, there is no concept of timestamps for passing the input. We feed the complete sentence together and get the embeddings for all the words together.

Transformer Networks

- BERT (Bidirectional Encoder Representations from Transformers) outperform LSTM.
- These models are faster as the words can be processed simultaneously. The context of words is better learned as they can learn from both directions simultaneously. If we stack the encoders, we get the BERT model.

NNs for Finance

- NLP - FinBERT (from last lecture)
- https://www.investopedia.com/articles/trading/06/neuralnetworks.asp
  - In some areas, such as fraud detection or risk assessment, they are the indisputable leaders. The major fields in which neural networks have found application are financial operations, enterprise planning, trading, business analytics, and product maintenance.
  - A 10% increase in efficiency is probably the most a trader can ever expect from a neural network.
- https://www.accenture.com/us-en/insights/financial-services/technology-advisory-neural-networks
- https://blog.aspiresys.com/banking-and-finance/financial-applications-neural-networks/
  - Stock Market Prediction/Stock Market Index Prediction
  - Loan Application Evaluation & Underwriting
  - Credit Card Customers Search

Further Reading

- Deep Learning with Python
  - https://amzn.to/3qVpSin
- Activation functions
  - https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/
  - https://www.v7labs.com/blog/neural-networks-activation-functions
  - https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6