

# Introduction to Machine Learning Applications

Spring 2023

Neural networks

**Minor Gordon**

[gordom6@rpi.edu](mailto:gordom6@rpi.edu)



**Rensselaer**

# Neural Networks Have been Around

- In “A Logical Calculus of Ideas Immanent in Nervous Activity,” McCulloch and Pitts presented a simplified computational model of how biological neurons might work together in animal brains to perform complex computations using propositional logic.
- Published in 1943!

# Neural Networks Modeled Off Biological Processes

- Neuron typically connected to thousands of other neurons
- Organized in Layers

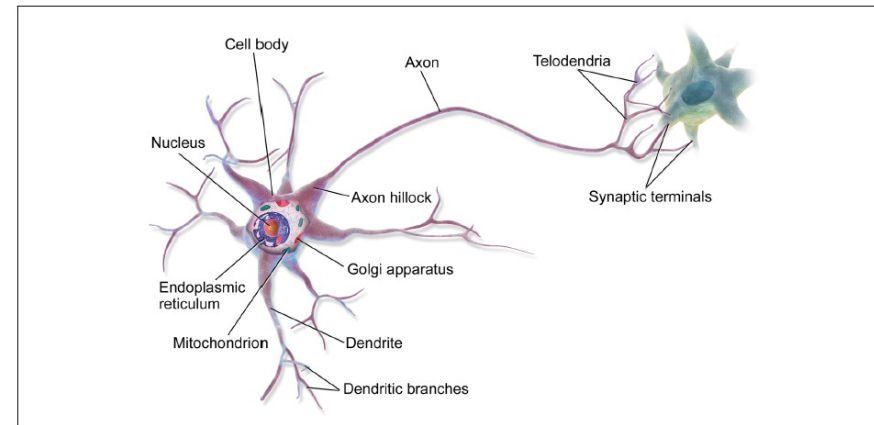


Figure 10-1. Biological neuron<sup>3</sup>

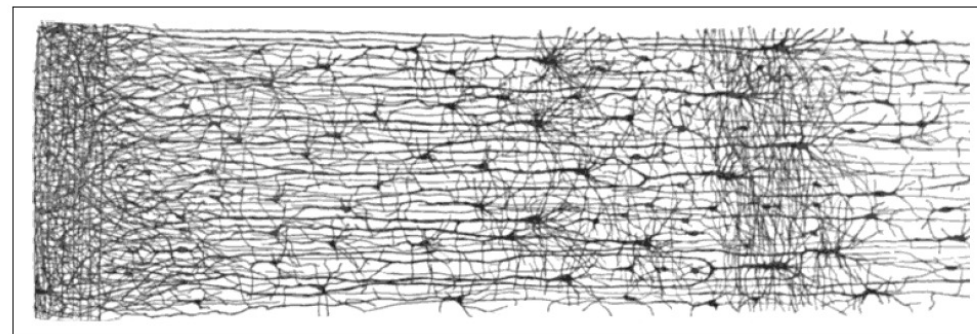


Figure 10-2. Multiple layers in a biological neural network (human cortex)<sup>5</sup>

# The Perceptron 1957 Frank Rosenblatt

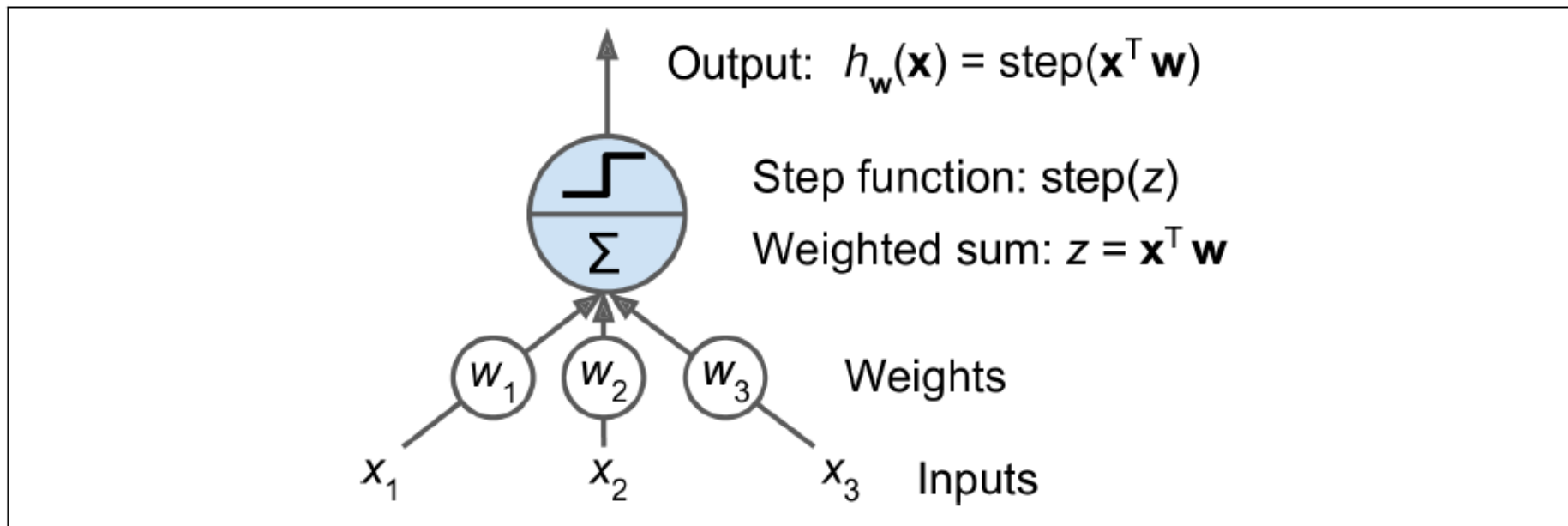


Figure 10-4. Threshold logic unit

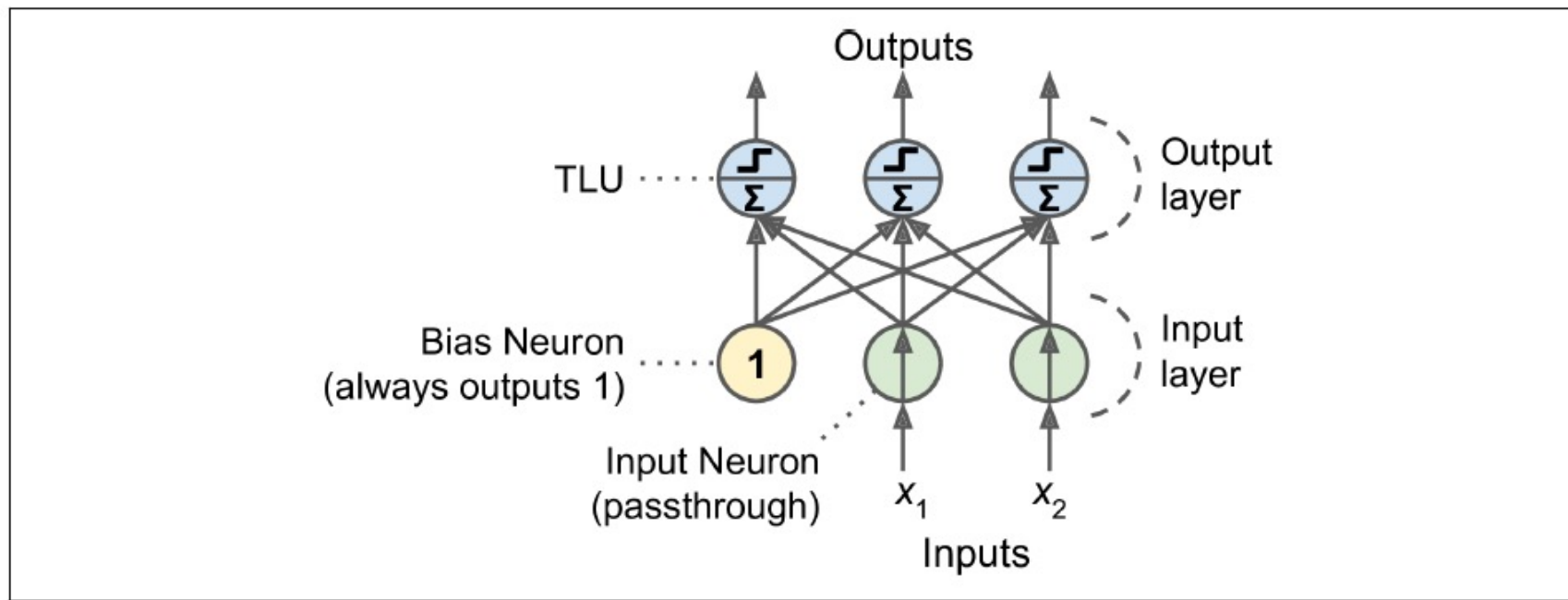


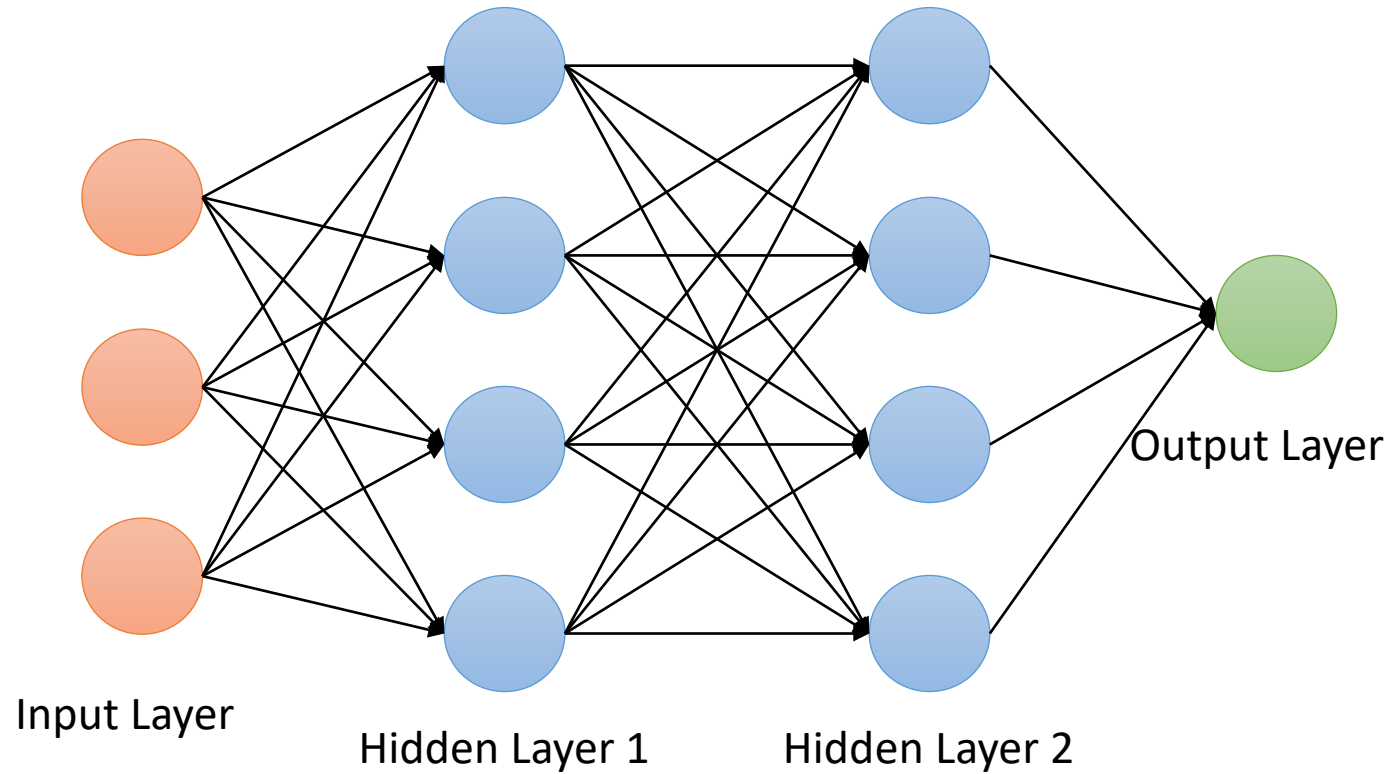
Figure 10-5. Perceptron diagram

Equation 10-2. Computing the outputs of a fully connected layer

$$h_{\mathbf{W}, \mathbf{b}}(\mathbf{X}) = \phi(\mathbf{XW} + \mathbf{b})$$

- As always,  $\mathbf{X}$  represents the matrix of input features. It has one row per instance, one column per feature.
- The weight matrix  $\mathbf{W}$  contains all the connection weights except for the ones from the bias neuron. It has one row per input neuron and one column per artificial neuron in the layer.
- The bias vector  $\mathbf{b}$  contains all the connection weights between the bias neuron and the artificial neurons. It has one bias term per artificial neuron.
- The function  $\phi$  is called the *activation function*: when the artificial neurons are TLUs, it is a step function (but we will discuss other activation functions shortly).

# Simple 3-layer neural network



Traditionally the input layer is not counted in the number of layers.

# Layers

- **Input layer**

- Accepts the information for the network to process
- Number of nodes present in this layer is equal to the number of features present in the dataset

- **Hidden layers**

- Do most of the processing
- Every layer has a set of nodes that receive information from the previous layer's nodes

- **Output layer**

- Outputs the information processed by the neural network

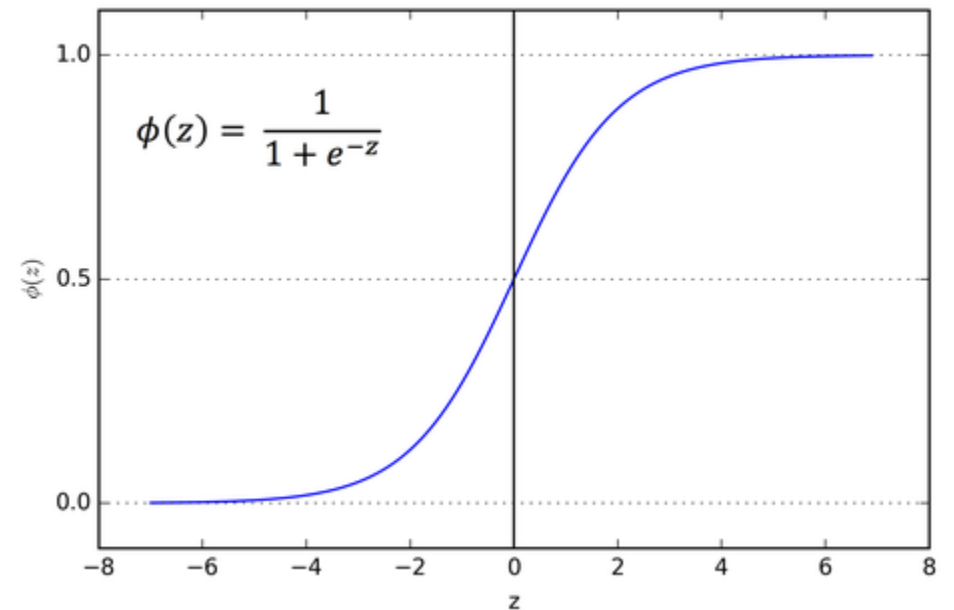
# Activation function

- At every (non-input) node, information received by the node is processed using an activation function
- This function allows the network to learn complex aspects of the data and thereby represent the non-linear complex functional mappings between inputs and outputs
- Hence, we consider non-linear functions as activation functions to solve complex and challenging problems
- However, these functions should also be differentiable in order to backward propagation



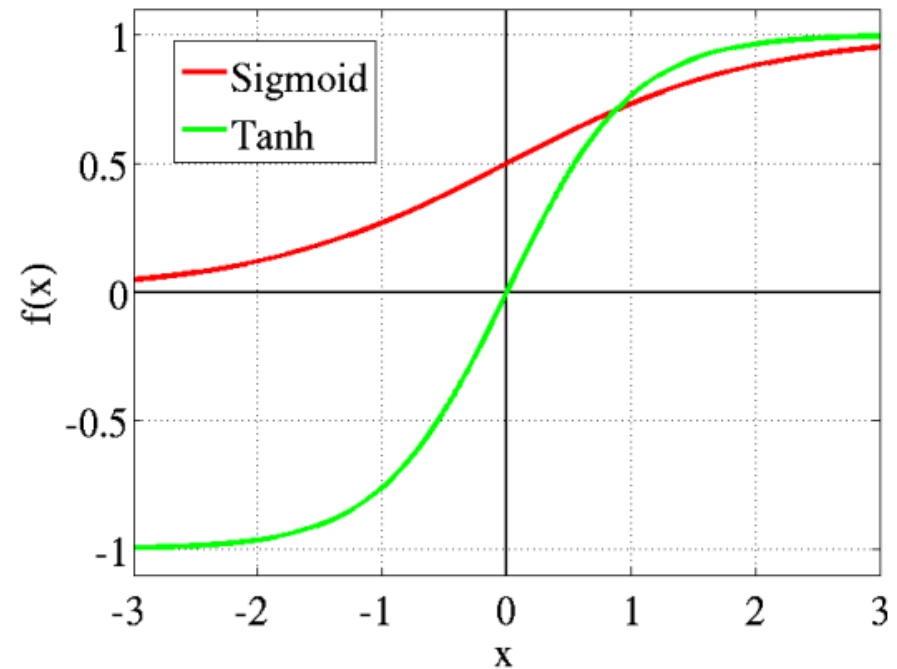
# Sigmoid activation function

- Generates values between 0 and 1
- Used mainly for models that predict probability
- Differentiable



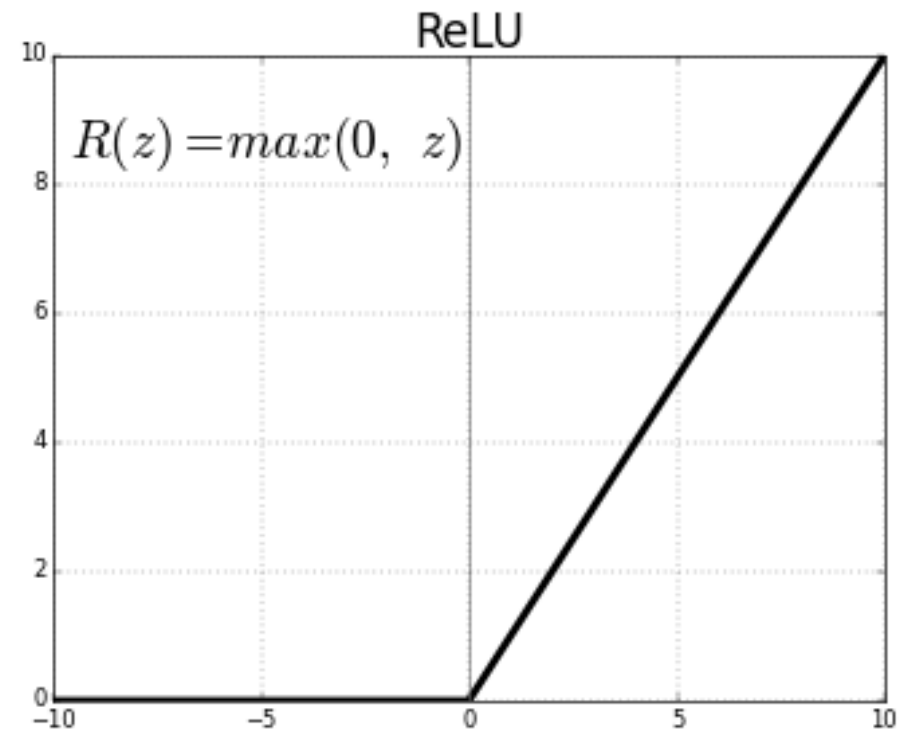
# Tanh or hyperbolic activation function

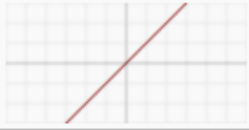



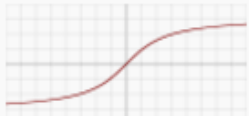




- Similar to sigmoid, but generates values between -1 and 1
- Used mainly for binary classification
- Differentiable



# Rectified Linear Unit (ReLU) activation function

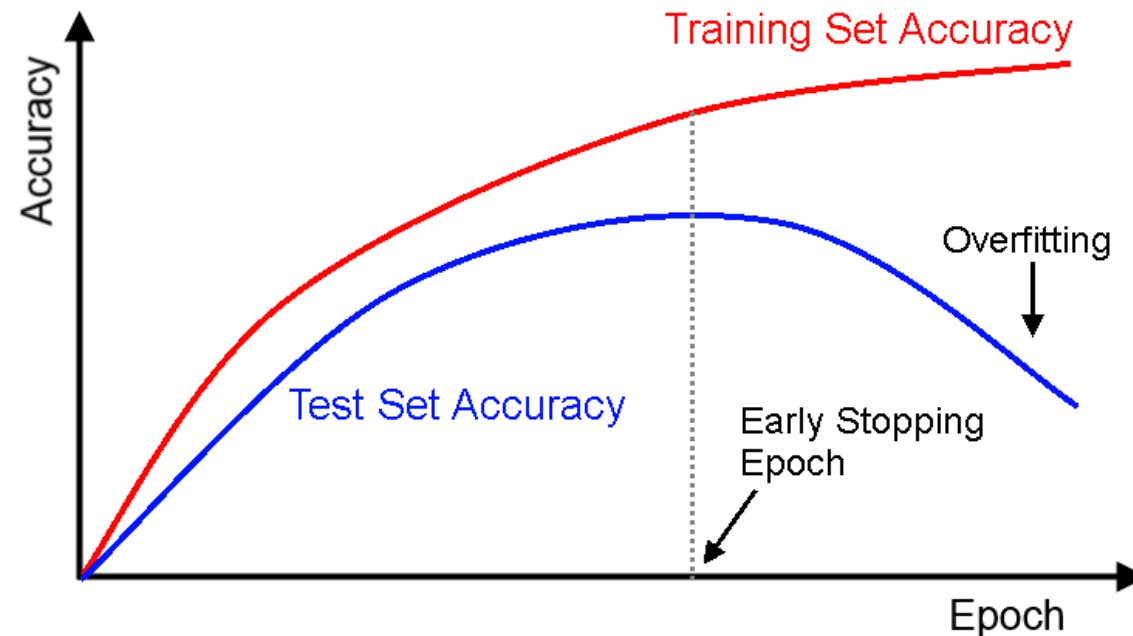
- Returns 0 when  $z$  is less than 0 and is  $z$  when it is  $\geq 0$
- Most commonly used



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		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

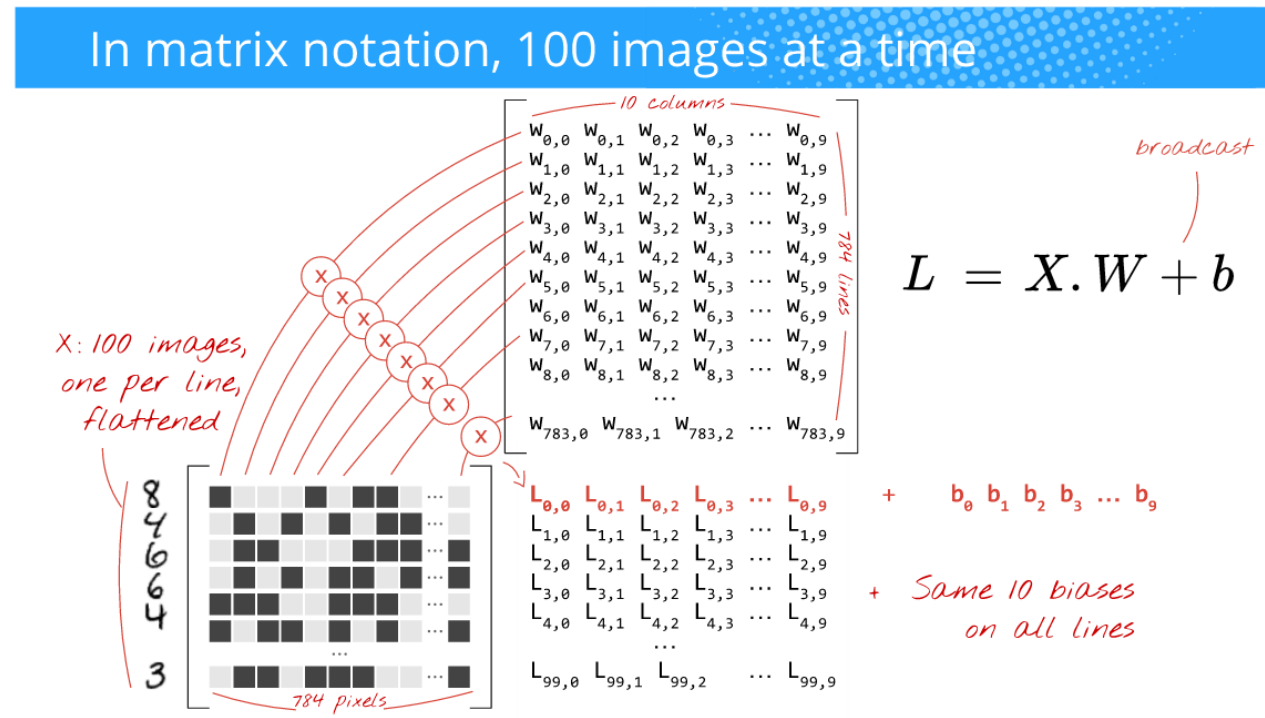
# Epoch

- An **epoch** is one complete presentation of the data set to be learned to a learning machine.



# Batch

- A batch is how many inputs you perform computations on at a time

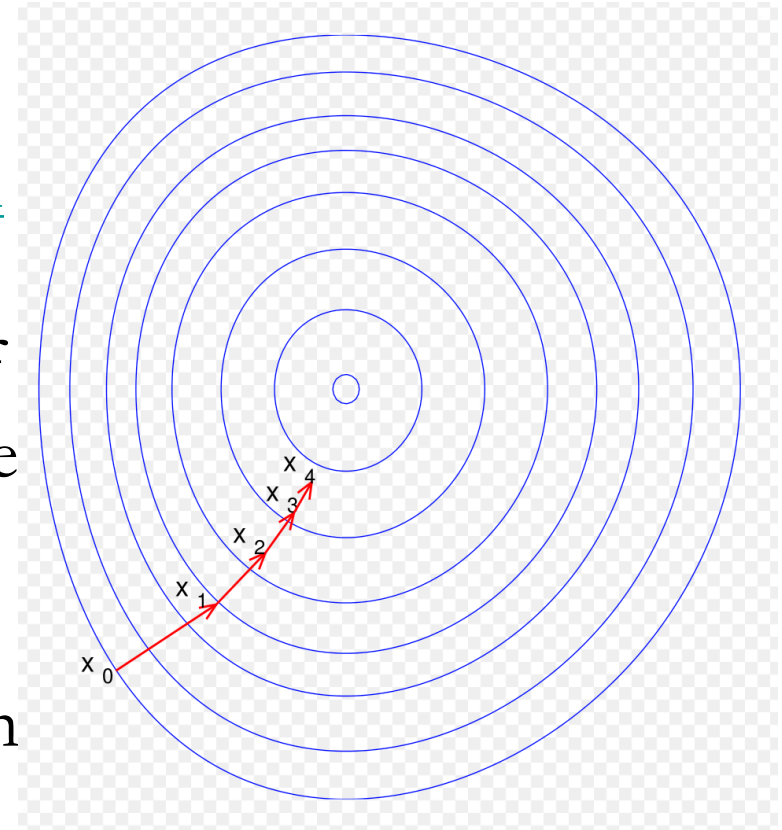


(Author: Martin Gerner)

# Gradient descent

”Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the *negative* of the gradient (or of the approximate gradient) of the function at the current point.” (Wikipedia)

Speed of gradient descent is set by  
the learning rate



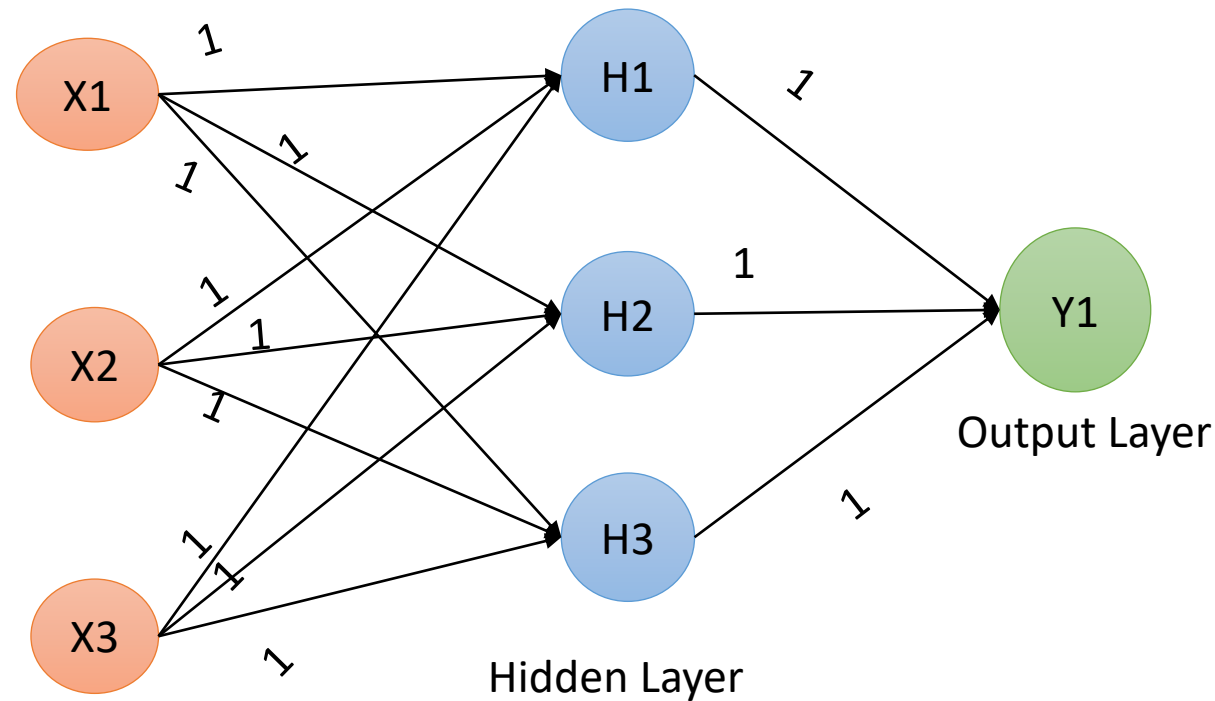
# Training a neural network

- Loop through the training dataset for  $x$  epochs.
- In each epoch feed training data in batches of  $x$ .
- Learning occurs through gradient descent with *backpropagation*.
- Backpropagation uses the partial derivative of the cost functions with respect to the weights and bias terms to update the model.
- Potential issues with a *vanishing or exploding gradient*.



# Forward propagation

- Neuron output =  $\text{activation}(\sum(\text{weight}_i * \text{input}_i) + \text{bias})$



Input Layer with X3 as the bias term of value 1

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

# Forward propagation

- At node H1:
  - $W1*X1 + W2*X2 + X3$
  - $1*0 + 1*0 + 1$
  - 1
- At node H2:
  - $W1*X1 + W2*X2 + X3$
  - $1*0 + 1*0 + 1$
  - 1
- At node H3:
  - $W1*X1 + W2*X2 + X3$
  - $1*0 + 1*0 + 1$
  - 1
- At node Y1:
  - $W1*H1 + W2*H2 + W3*H3$
  - $1*1 + 1*1 + 1*1$
  - 3
- According to the table, the output should be 0 but we got 3
- **Loss** = actual – predicted  
=  $0 - 3$   
= **-3**

# Backpropagation

Feed the loss backwards to adjust the weights

Two main steps:

- 1) Find the gradients
- 2) Update the weights using the gradients and learning rate

$W := W - \alpha \cdot J'(W)$  where,

- $W$  is the weight at hand,
- $\alpha$  is the learning rate
- $J'(W)$  is the partial derivative of the cost function  $J(W)$

# Chain rule

- In forward propagation:  $f(x)=A(B(C(x)))$  if we assume A, B and C are the activation functions in different layers
- Using chain rule, we first find the derivative of  $f(x)$  w.r.t  $x$ 
  - $f'(x)=f'(A) \cdot A'(B) \cdot B'(C) \cdot C'(x)$
- Derivative w.r.t B
  - $f'(B)=f'(A) \cdot A'(B)$  where we assume that  $B(C(x))$  is a constant replaced by B

# Vanishing gradient

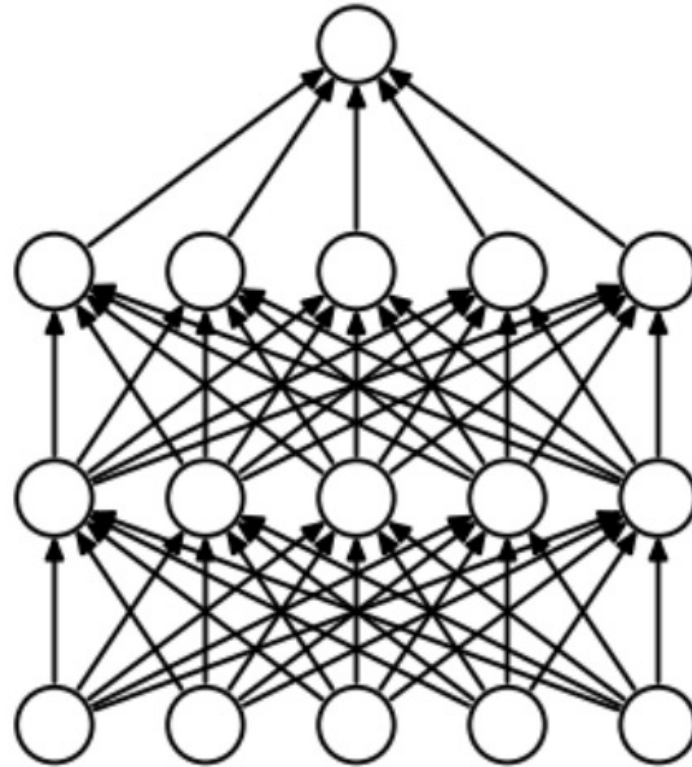
- By adding more layers, the gradients of loss function will approach zero – training the network gets harder
- Activation functions such as sigmoid function converts a large input space to the range  $[0, 1]$ 
  - A large change in the input reflects as a small change in the output
  - This also means the derivative will be small

# Vanishing gradient

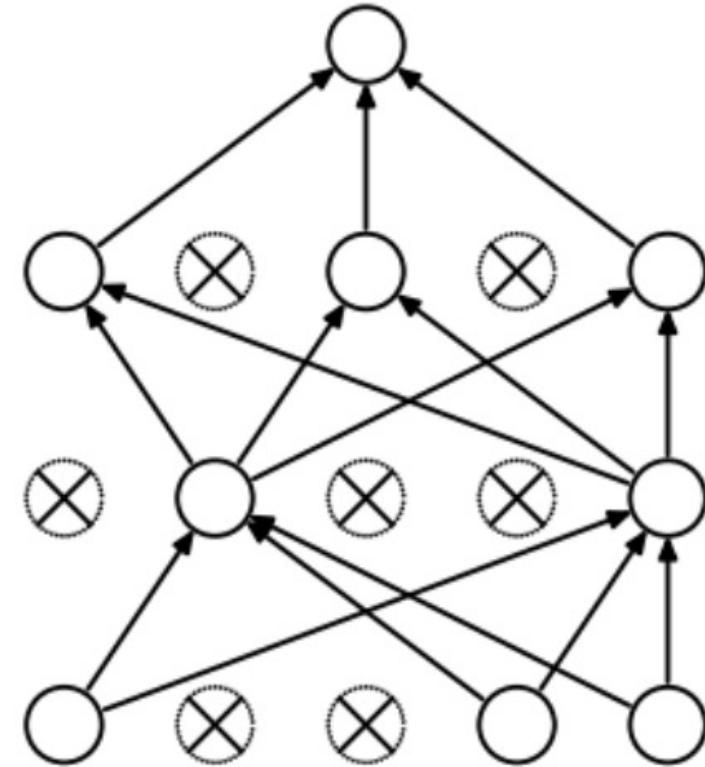
- In backpropagation using the chain rule, derivatives of each layer are multiplied down the network to compute the derivatives of the initial layers
  - From the final layer to the initial layer
- In a scenario where  $N$  hidden layers use sigmoid function,  $N$  small derivatives are multiplied together leads to the gradient decreasing exponentially
- Small gradient values do not help updating the weights and biases especially of initial layers

# Avoiding overfitting: dropout

- A regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data
  - Randomly-selected neurons are ignored (dropped out) during training
  - Their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.



(a) Standard Neural Net



(b) After applying dropout.

# Softmax function

- A function that computes the probability distribution of a task over k different class labels
- In a multi-class classification, assigns probabilities to each class.
  - These probabilities should add up to 1.0





# Playground

<https://playground.tensorflow.org>

# Playground exercise

- Perform a machine learning experiment.
- For a given problem, test 2 or more different hyperparameters for your model.
- Report on result