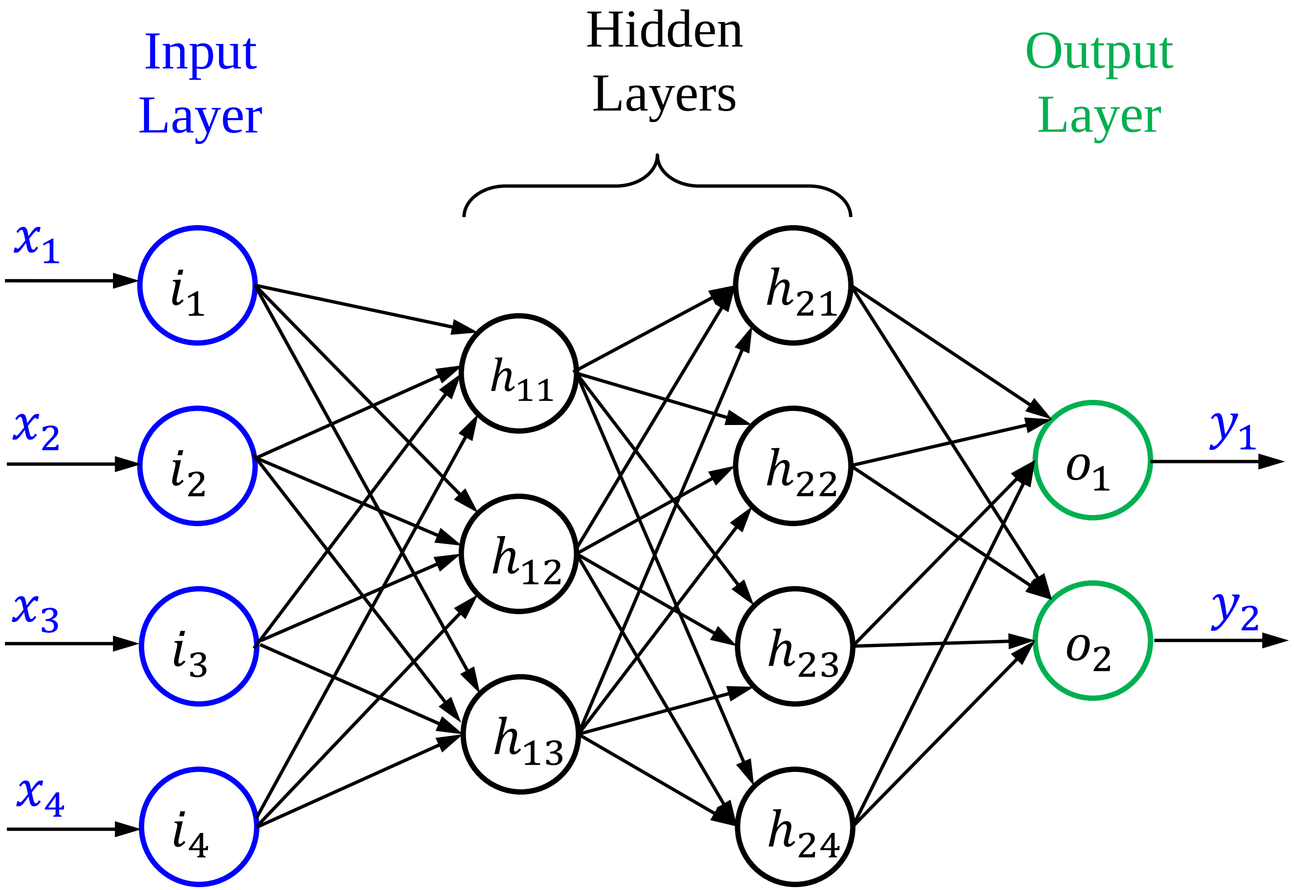
Module 2
Artificial Neural Networks

Artificial Neural Networks

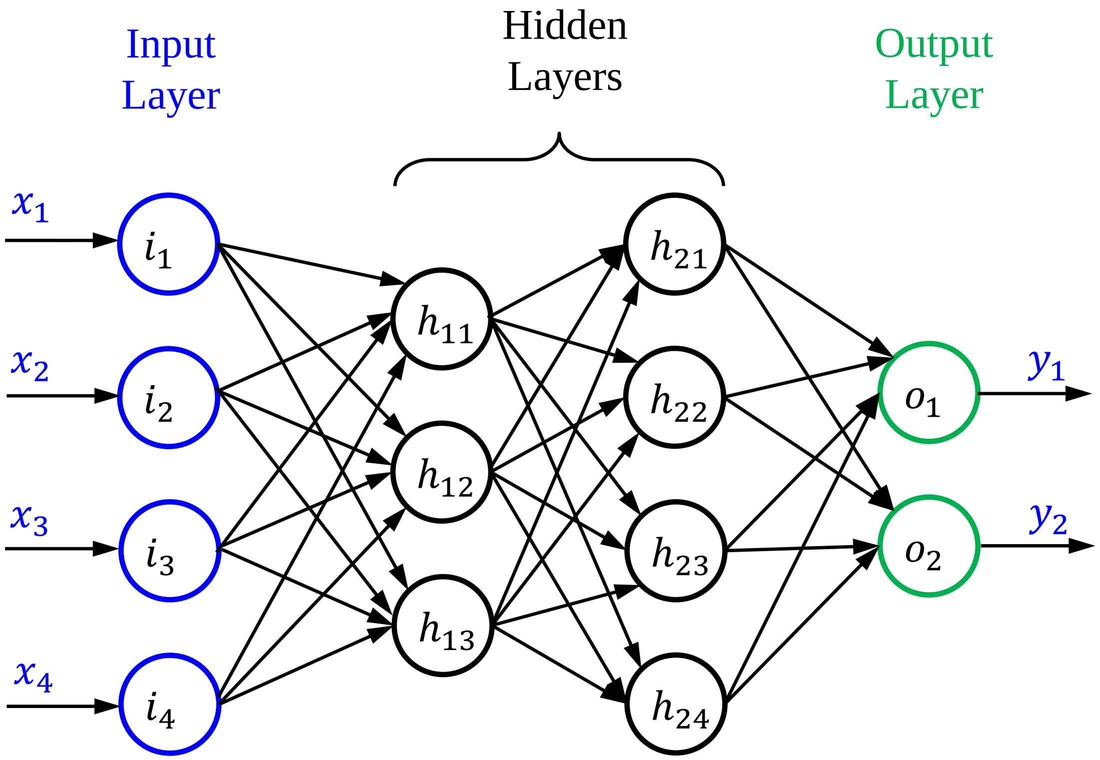
https://neurosurgerycnj.com/a-guide-to-brain-cells/



Artificial Neural Network (ANN)

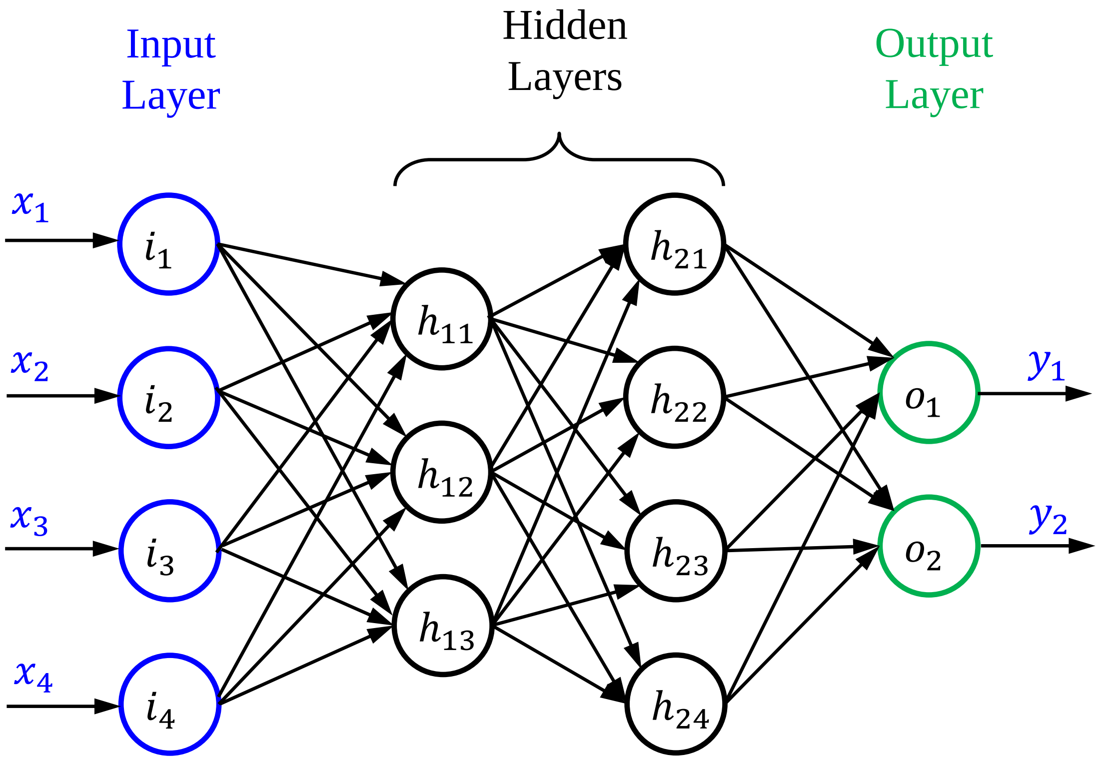
Santa Clara University

Multilayer Perceptron Networks (MLP)



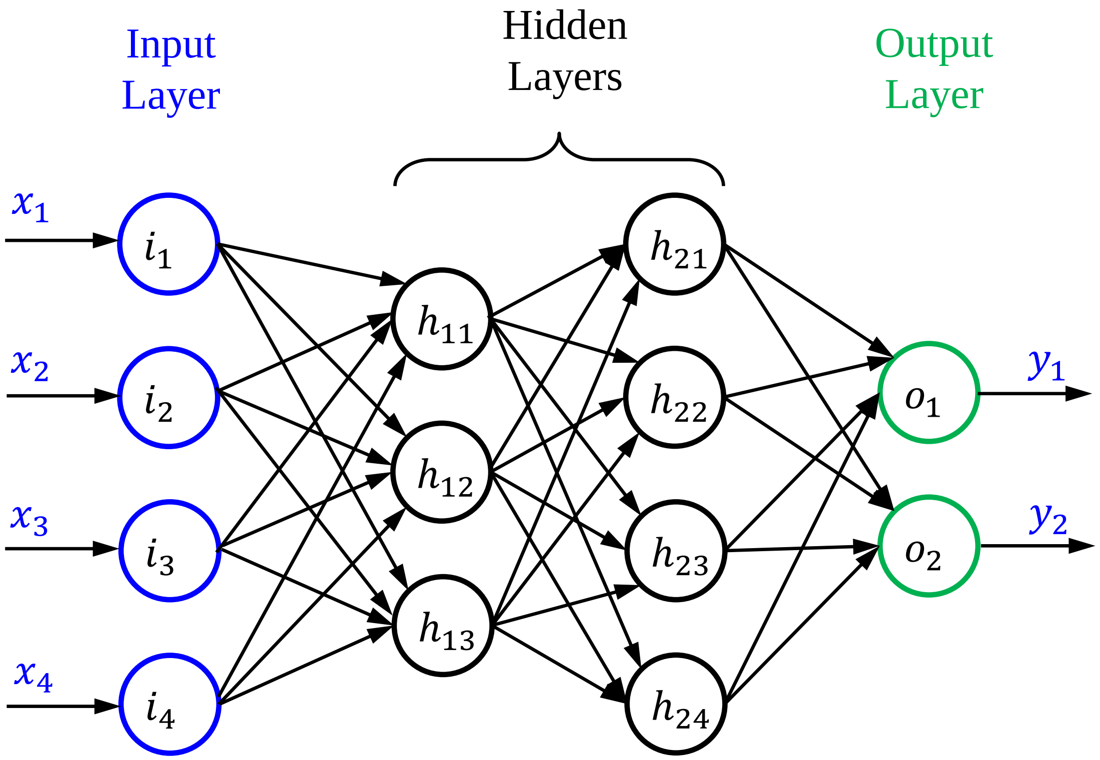
- A multilayer perceptron (MLP) is a fully connected feedforward artificial neural network (ANN) with, at least, three layers and a nonlinear activation function.
- Each perceptron or neuron in one layer is connected to all neurons in a subsequent layer.
- In a feedforward network, information travels in one direction.

Radial Basis Function Networks (RBF)



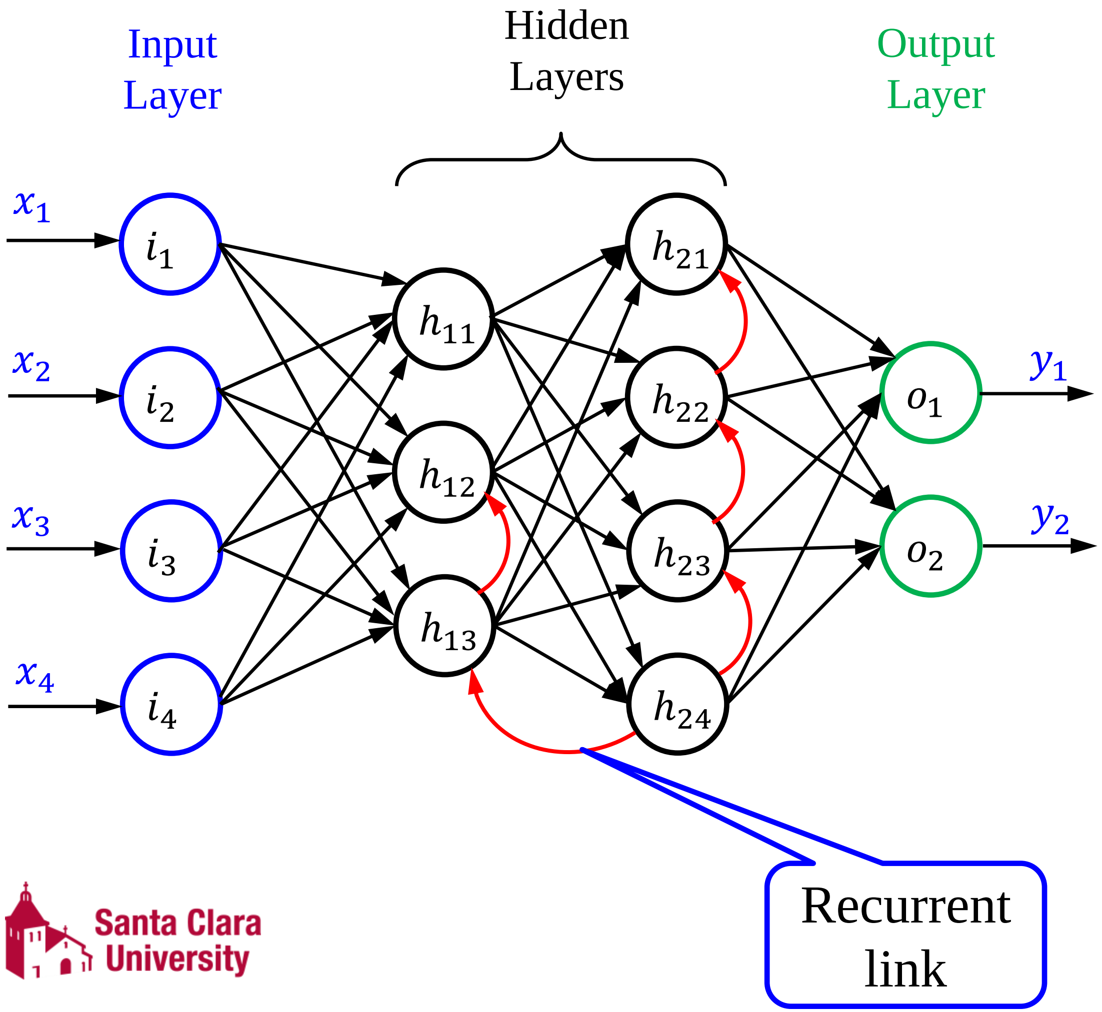
- A Radial Basis Function (RBF) network is an artificial neural network with radial basis activation functions.
- The output is a linear combination of radial basis functions of the inputs and neuron parameters.
- RBF networks are used in approximation, time series prediction, classification, and system control.

Deep Neural Networks (DNNs)



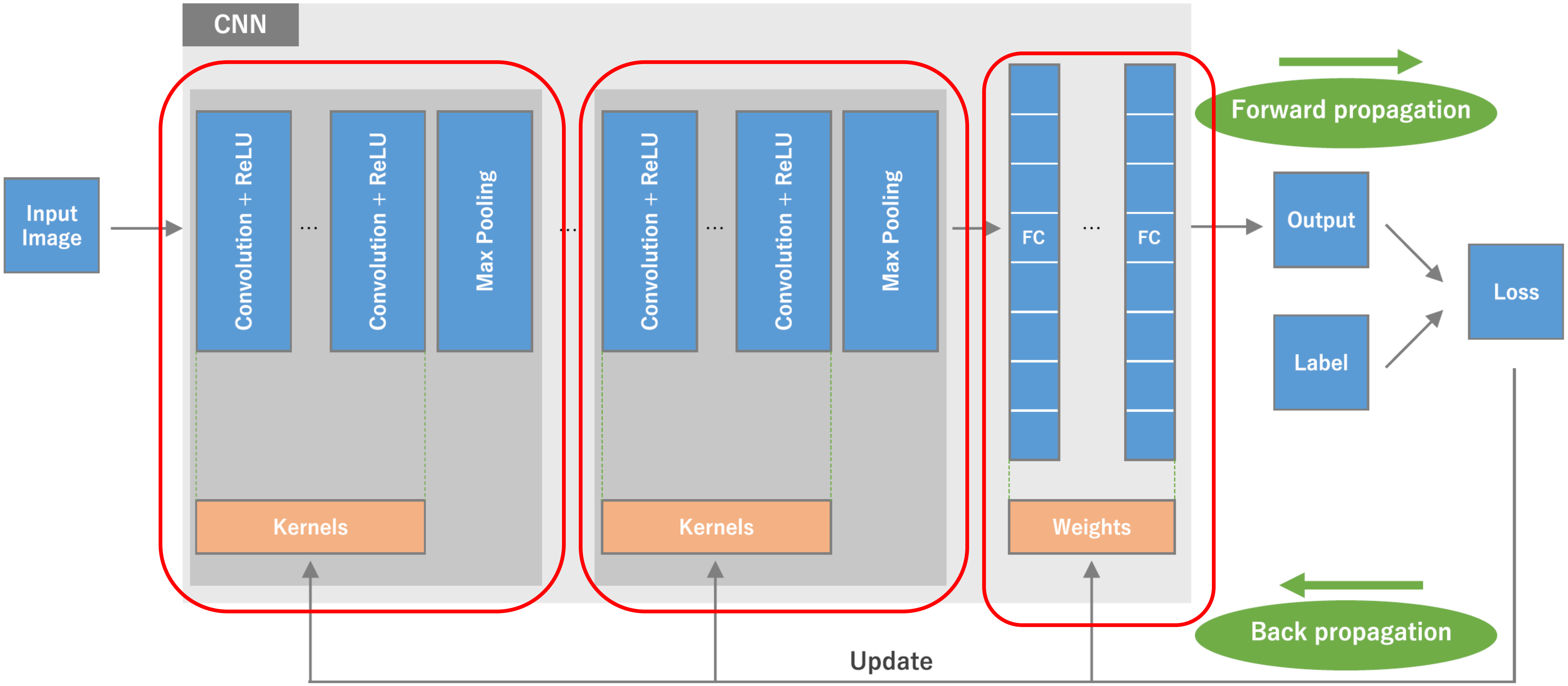
- ANNs with 1 or 2 hidden layers are called shallow networks.
- ANNs with more hidden layers are called Deep Neural Networks (DNNs).
- Shallow ANNs are for simple tasks such as image classification.
- DNNs are for complex tasks such as image segmentation and natural language processing.

Recurrent Neural Networks



- Recurrent Neural Networks (RNNs) are ANNs with recurrent or feedback connections.
- RNNs overcome the problem of maintaining temporal information in ANNs but suffer the complexity in training.
- RNNs are applicable to learning sequences and tree structures in natural language processing.

Convolutional Neural networks (1)



Santa Clara University

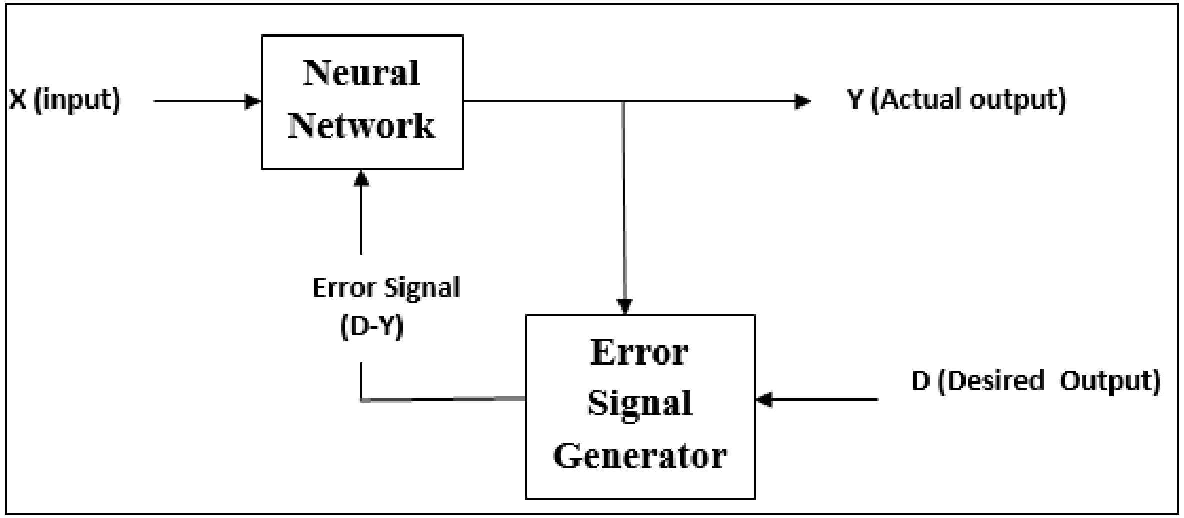
Yamashita, Rikiya, et al. "Convolutional neural networks: an overview and application in radiology." Insights into imaging 9 (2018): 611-629.

Convolutional Neural networks (2)

- Convolutional neural networks (CNNs) are regularized feed-forward neural networks that learn feature engineering via filter (or kernel) optimization.
- Regularization is a process that changes the resulting answer to be "simpler."
- CNN is a mathematical construct typically composed of three types of layers (or building blocks): convolution, pooling, and fully connected layers.
  a. Convolution and pooling layers perform feature extraction
  b. The fully connected layer maps the extracted features into the final output, such as classification.

Supervised Learning

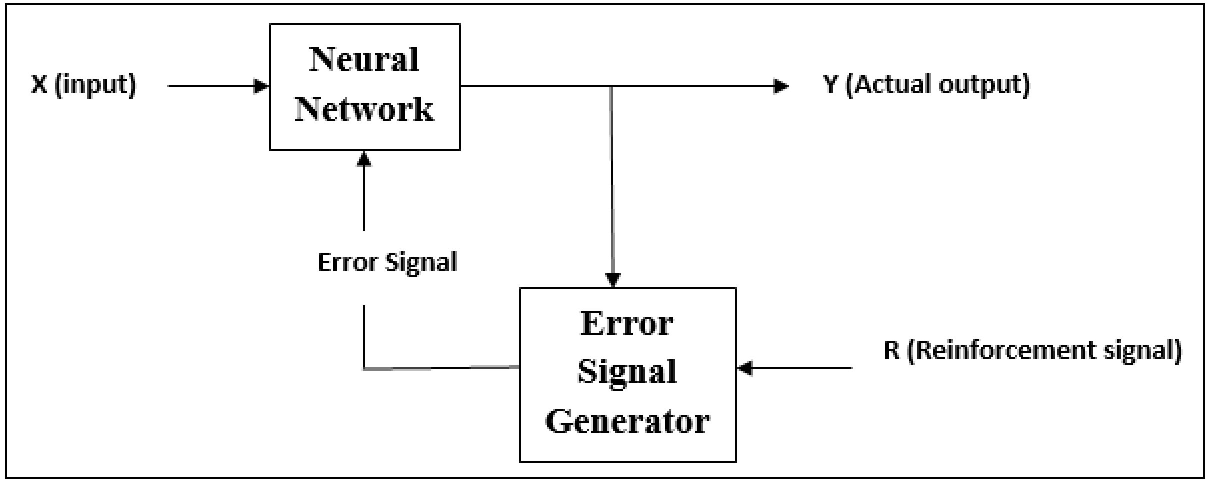
- The output vector Y is compared to the desired output D.
- Based on the error, weights are adjusted until the actual output is matched with (or close to) the desired output.
- The dataset contains input vectors and labels (or desired outputs)
- Common ML applications.



https://www.tutorialspoint.com/artificial_neural_network/artificial_neural_network_building_blocks.htm

Reinforcement Learning

- The network receives a feedback signal R based on the output Y.
- The network performs weight adjustments for better feedback (reward and punishment mechanism).
- Self-driving Cars, Data Center Google Cooling System, Trading and Finance, Learning in NLP (Natural Language Processing), Healthcare applications, etc.

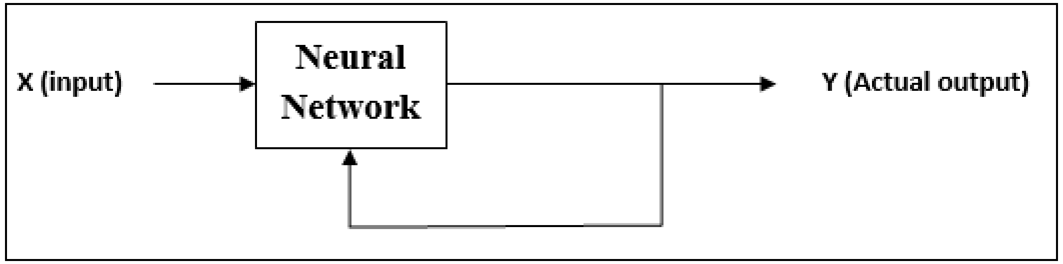


https://www.tutorialspoint.com/artificial_neural_network/artificial_neural_network_building_blocks.htm

Santa Clara University

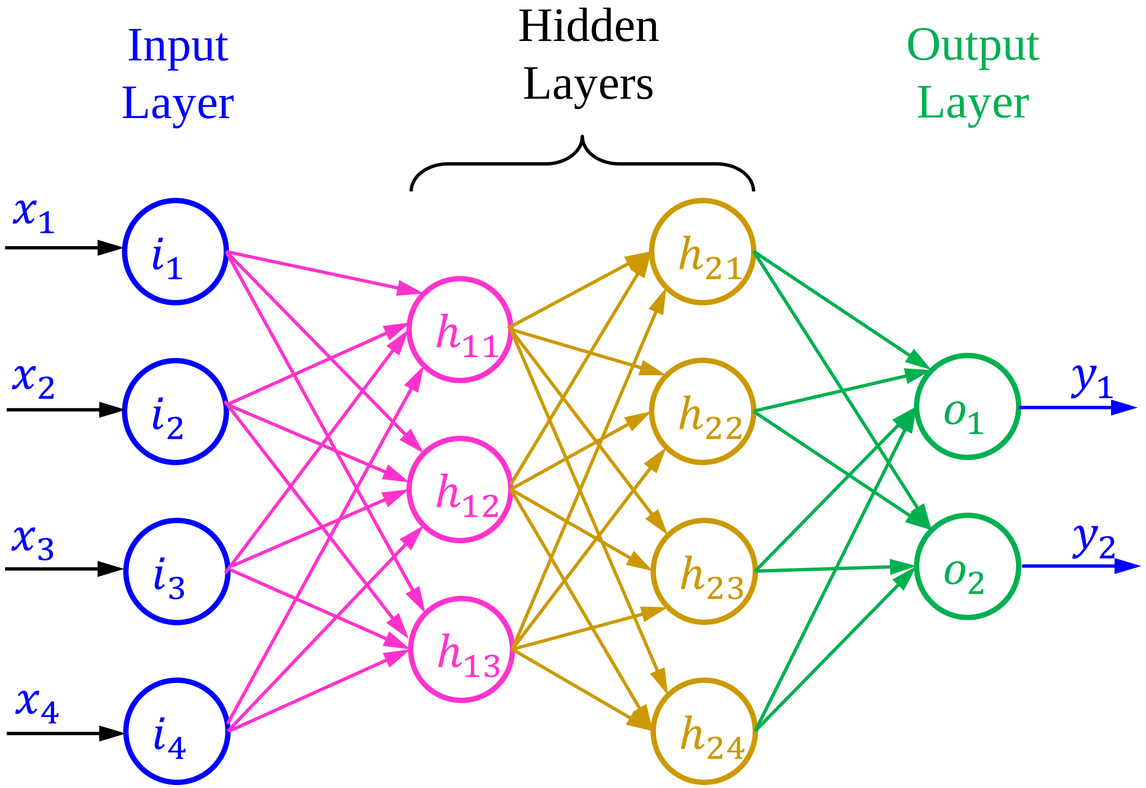
Unsupervised Learning

- The network discovers by itself the patterns and features from the input data and the relation between the input data and the output.
- Anomaly detection, hierarchical cluster relation, association rule, recommendation system, customer segmentation, etc.



https://www.tutorialspoint.com/artificial_neural_network/artificial_neural_network_building_blocks.htm

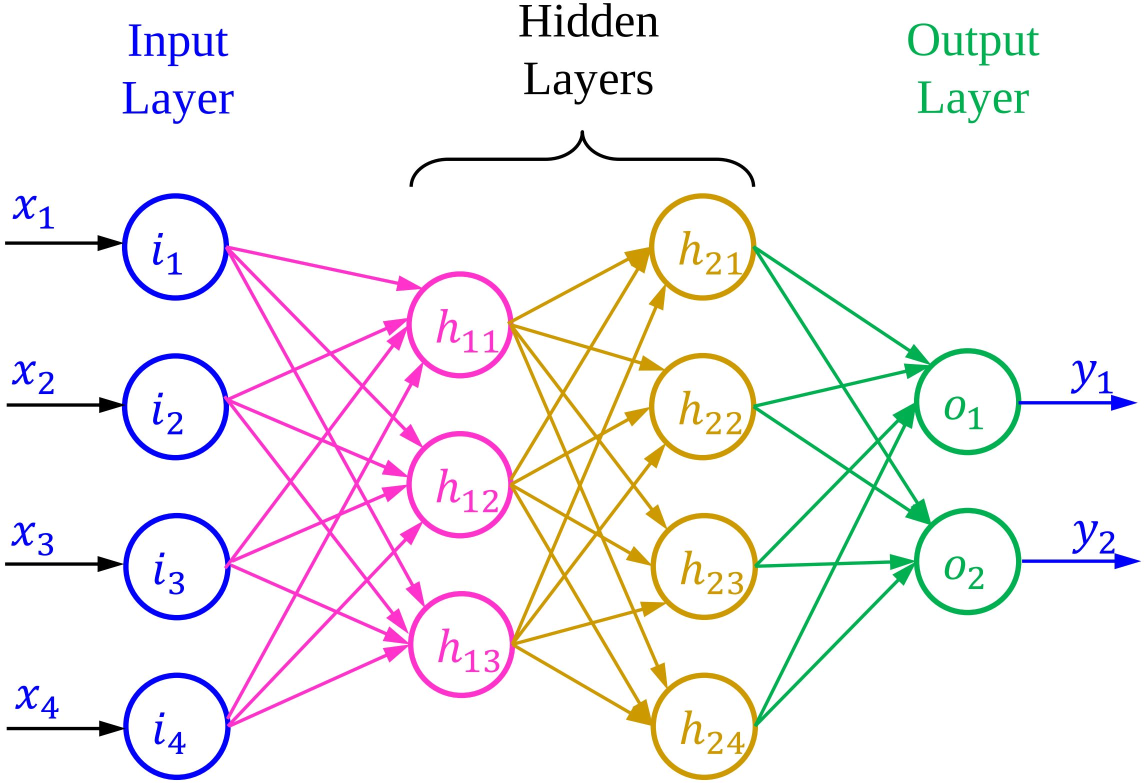
Implementation of FF Networks (1)



- $w_{12}^{h1} \rightarrow$ weight from neuron i_1 to neuron h_{12} (neuron 2 in the hidden layer 1).
- $w_{13}^{h2} \rightarrow$ weight from neuron h_{11} to neuron h_{23} (neuron 1 in the hidden layer 1 to neuron 3 in the hidden layer 2).
- $w_{12}^{o} \rightarrow$ weight from neuron h_{21} to output neuron o_2.
- $\varphi()$ is activation function of a neuron output.

Santa Clara University

Implementation of FF Networks (2)



- Input layer:

$$i_1 = \varphi(x_1), i_2 = \varphi(x_2), i_3 = \varphi(x_3)$$
$$i_4 = \varphi(x_4)$$

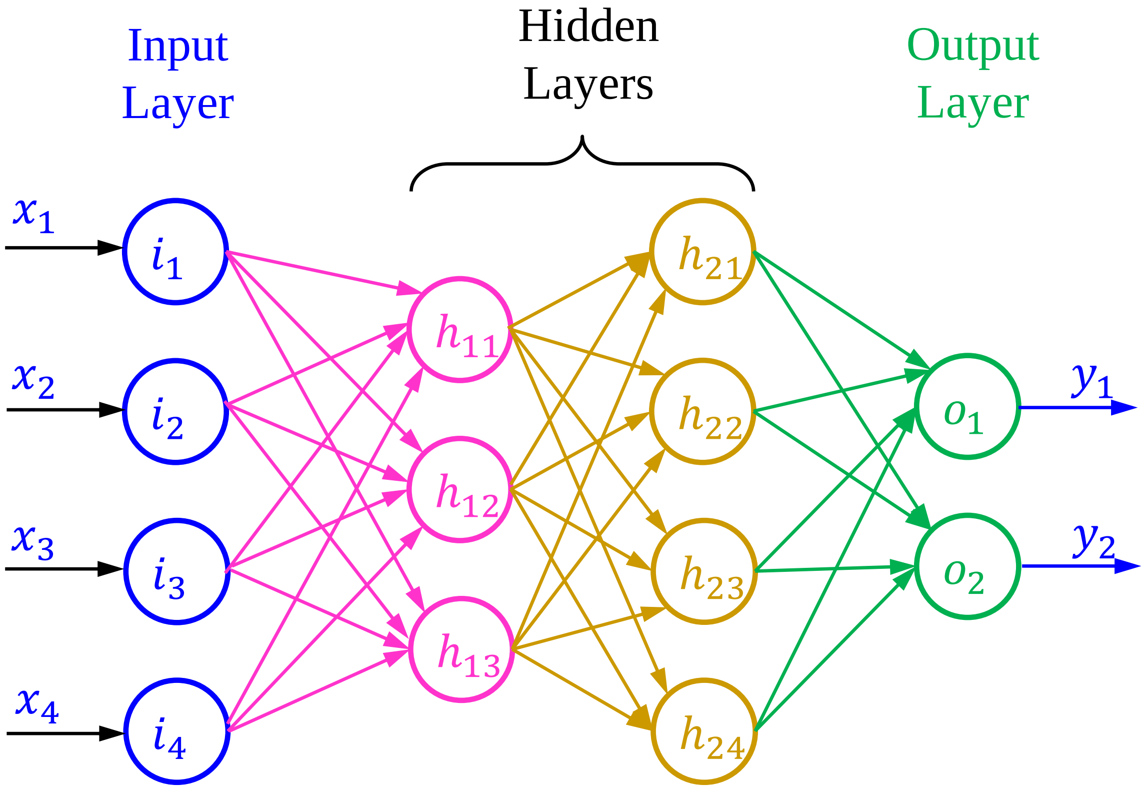
- Hidden layer 1:

$$h_{11} = \varphi\left(i_1 w_{11}^{h1} + i_2 w_{21}^{h1} + i_3 w_{31}^{h1} + i_4 w_{41}^{h1} + b_1\right)$$
$$h_{12} = \varphi\left(i_1 w_{12}^{h1} + i_2 w_{22}^{h1} + i_3 w_{32}^{h1} + i_4 w_{42}^{h1} + b_2\right)$$
$$h_{13} = \varphi\left(i_1 w_{13}^{h1} + i_2 w_{23}^{h1} + i_3 w_{33}^{h1} + i_4 w_{43}^{h1} + b_3\right)$$

- Hidden layer 2:

$$h_{21} = \varphi\left(h_{11} w_{11}^{h2} + h_{12} w_{21}^{h2} + h_{13} w_{31}^{h2} + b_1\right)$$
$$h_{22} = \varphi\left(h_{11} w_{12}^{h2} + h_{12} w_{22}^{h2} + h_{13} w_{32}^{h2} + b_2\right)$$
$$h_{23} = \varphi\left(h_{11} w_{13}^{h2} + h_{12} w_{23}^{h2} + h_{13} w_{33}^{h2} + b_3\right)$$
$$h_{24} = \varphi\left(h_{11} w_{14}^{h2} + h_{12} w_{24}^{h2} + h_{13} w_{34}^{h2} + b_4\right)$$

Implementation of FF Networks (3)



- Output layer:

$$o_1 = \varphi(h_{21}w^o_{11} + h_{22}w^o_{21} + h_{23}w^o_{31} + h_{24}w^o_{41} + b_1)$$
$$o_2 = \varphi(h_{21}w^o_{12} + h_{22}w^o_{22} + h_{23}w^o_{32} + h_{24}w^o_{42} + b_2)$$
$$y_1 = o_1$$
$$y_2 = o_2$$

Matrix Multiplication

- The multiplication of two matrices is the "dot product" of rows and columns. Assuming we have $P = A \times B \Longrightarrow A \rightarrow 2 \times 3, B \rightarrow 3 \times 2, P \rightarrow 2 \times 2$

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} \times \begin{bmatrix}7 & 8\\9 & 10\\11 & 12\end{bmatrix} = \begin{bmatrix}p_{11} & p_{12}\\p_{21} & p_{22}\end{bmatrix} = \begin{bmatrix}58 & 64\\139 & 154\end{bmatrix}$$

$$p_{11} = (1,2,3) \cdot (7,9,1) = 1 \times 7 + 2 \times 9 + 3 \times 1 = 58$$
$$p_{12} = (1,2,3) \cdot (8,10,12) = 1 \times 8 + 2 \times 10 + 3 \times 12 = 64$$
$$p_{21} = (4,5,6) \cdot (7,9,1) = 4 \times 7 + 5 \times 9 + 6 \times 1 = 139$$
$$p_{22} = (4,5,6) \cdot (8,10,12) = 5 \times 8 + 5 \times 10 + 6 \times 12 = 154$$

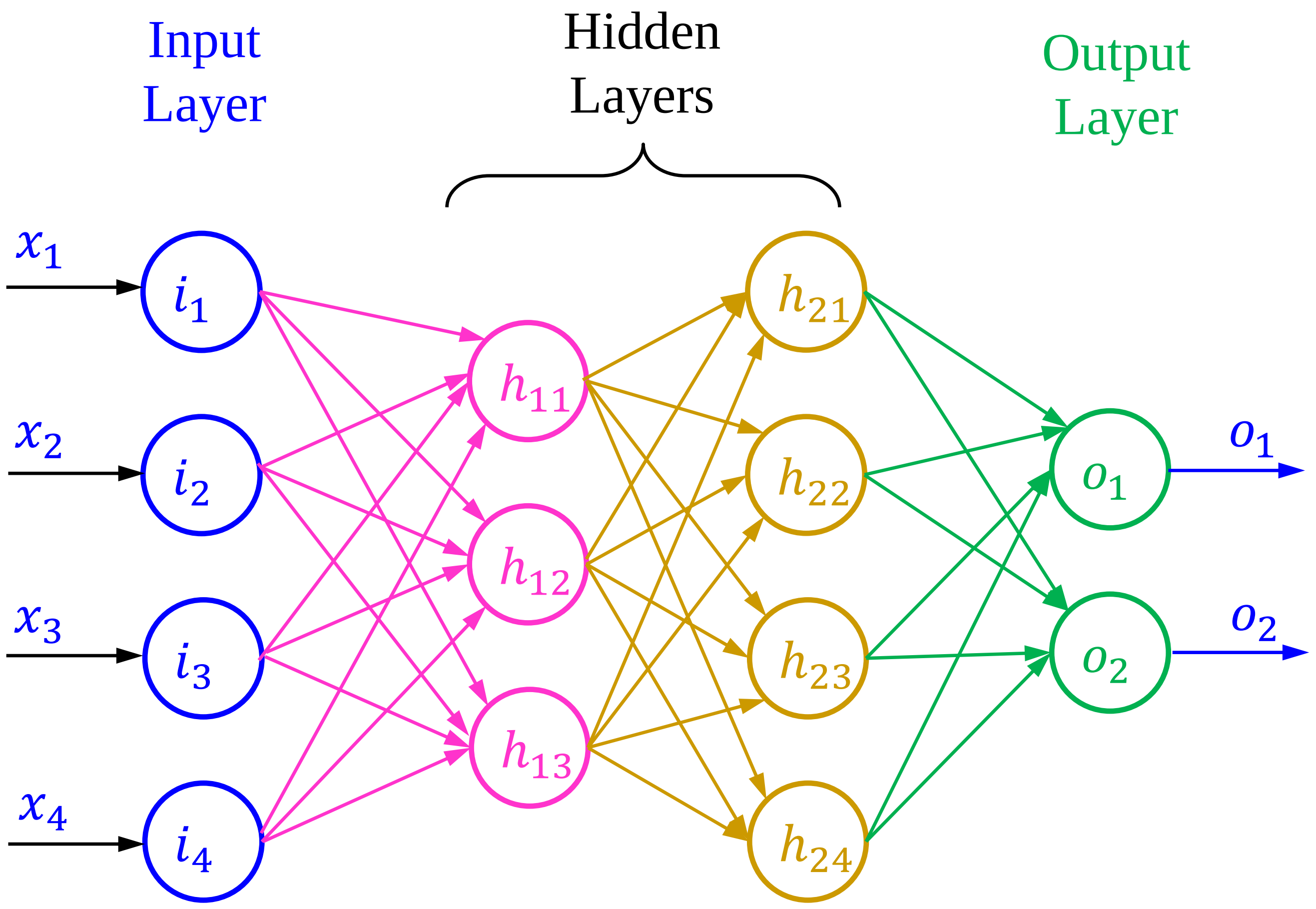
Matrix Multiplication Example

- Determine the dot product of two matrices below:

$$A(3 \times 4) * B(4 \times 2)$$

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} \times \begin{bmatrix} 13 & 14 \\ 15 & 16 \\ 17 & 18 \\ 19 & 20 \end{bmatrix} = \begin{bmatrix} 170 & 180 \\ 426 & 452 \\ 682 & 724 \end{bmatrix}$$

Matrix Representation of FF Networks (1)



Matrix Representation of FF Networks (2)

- Hidden layer 1:

$$h_{11} = \varphi\left(i_1 w_{11}^{h1} + i_2 w_{21}^{h1} + i_3 w_{31}^{h1} + i_4 w_{41}^{h1} + b_1\right) = \varphi(p_{11} + b_1)$$
$$h_{12} = \varphi\left(i_1 w_{12}^{h1} + i_2 w_{22}^{h1} + i_3 w_{32}^{h1} + i_4 w_{42}^{h1} + b_2\right) = \varphi(p_{12} + b_2)$$
$$h_{13} = \varphi\left(i_1 w_{13}^{h1} + i_2 w_{23}^{h1} + i_3 w_{33}^{h1} + i_4 w_{43}^{h1} + b_3\right) = \varphi(p_{13} + b_3)$$

$$\Leftrightarrow [i_1 \quad i_2 \quad i_3 \quad i_4] \times \begin{bmatrix} w_{11}^{h1} & w_{12}^{h1} & w_{13}^{h1} \\ w_{21}^{h1} & w_{22}^{h1} & w_{23}^{h1} \\ w_{31}^{h1} & w_{32}^{h1} & w_{33}^{h1} \\ w_{41}^{h1} & w_{42}^{h1} & w_{43}^{h1} \end{bmatrix} = [p_{11} \quad p_{12} \quad p_{13}]$$

Matrix Representation of FF Networks (3)

- Hidden layer 2:

$$h_{21} = \varphi\left(h_{11}w_{11}^{h2} + h_{12}w_{21}^{h2} + h_{13}w_{31}^{h2} + b_1\right) = \varphi(p_{11} + b_1)$$
$$h_{22} = \varphi\left(h_{11}w_{12}^{h2} + h_{12}w_{22}^{h2} + h_{13}w_{32}^{h2} + b_2\right) = \varphi(p_{12} + b_2)$$
$$h_{23} = \varphi\left(h_{11}w_{13}^{h2} + h_{12}w_{23}^{h2} + h_{13}w_{33}^{h2} + b_3\right) = \varphi(p_{13} + b_3)$$
$$h_{24} = \varphi\left(h_{11}w_{14}^{h2} + h_{12}w_{24}^{h2} + h_{13}w_{34}^{h2} + b_4\right) = \varphi(p_{14} + b_4)$$

$$\Leftrightarrow [h_{11} \quad h_{12} \quad h_{13}] \times \begin{bmatrix} w_{11}^{h2} & w_{12}^{h2} & w_{13}^{h2} & w_{14}^{h2} \\ w_{21}^{h2} & w_{22}^{h2} & w_{23}^{h2} & w_{24}^{h2} \\ w_{31}^{h2} & w_{32}^{h2} & w_{33}^{h2} & w_{34}^{h2} \end{bmatrix}$$
$$= [p_{11} \quad p_{12} \quad p_{13} \quad p_{14}]$$

Matrix Representation of FF Networks (4)

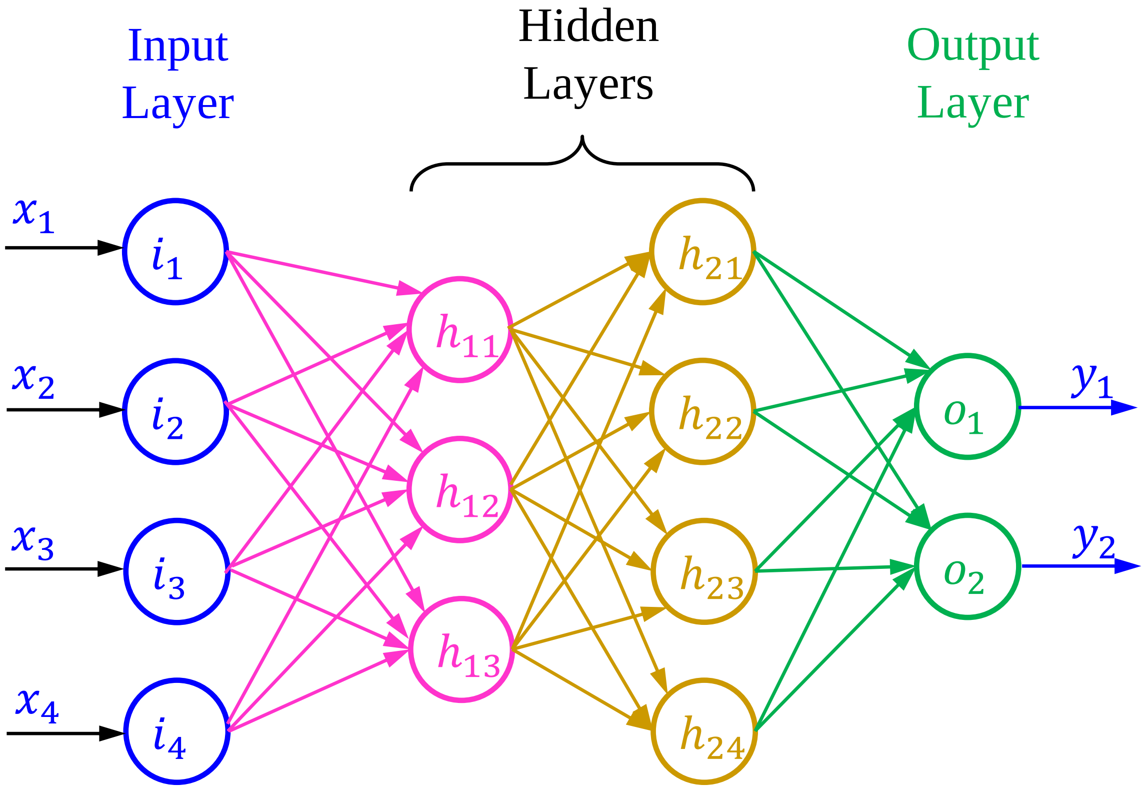
- Output layer:

$$o_1 = \varphi(h_{21}w_{11}^o + h_{22}w_{21}^o + h_{23}w_{31}^o + h_{24}w_{41}^o + b_1) = \varphi(p_{11} + b_1)$$
$$o_2 = \varphi(h_{21}w_{12}^o + h_{22}w_{22}^o + h_{23}w_{32}^o + h_{24}w_{42}^o + b_2) = \varphi(p_{12} + b_2)$$

$$\Leftrightarrow [h_{21} \quad h_{22} \quad h_{23} \quad h_{24}] \times \begin{bmatrix} w_{11}^o & w_{12}^o \\ w_{21}^o & w_{22}^o \\ w_{31}^{h2} & w_{32}^{h2} \\ w_{41}^o & w_{42}^o \end{bmatrix} = [p_{11} \quad p_{12}]$$

Santa Clara University

Matrix Representation of FF Networks (5)



- Matrix representation of each layer:

$$\left.\begin{aligned} H_1 &= \varphi\left(I \times W^{h1} + B_{h1}\right) \\ H_2 &= \varphi\left(H_1 \times W^{h2} + B_{h2}\right) \\ Y &= \varphi\left(H_2 \times W^{o} + B_{0}\right) \end{aligned}\right\}$$

- Matrix Dimension:

$$Y \Longrightarrow [1 \times 4] * [4 \times 3] * [3 \times 4] * [4 \times 2]$$

$$I \Longrightarrow [1 \times 4]$$

$$H_1 \Longrightarrow [4 \times 3]$$

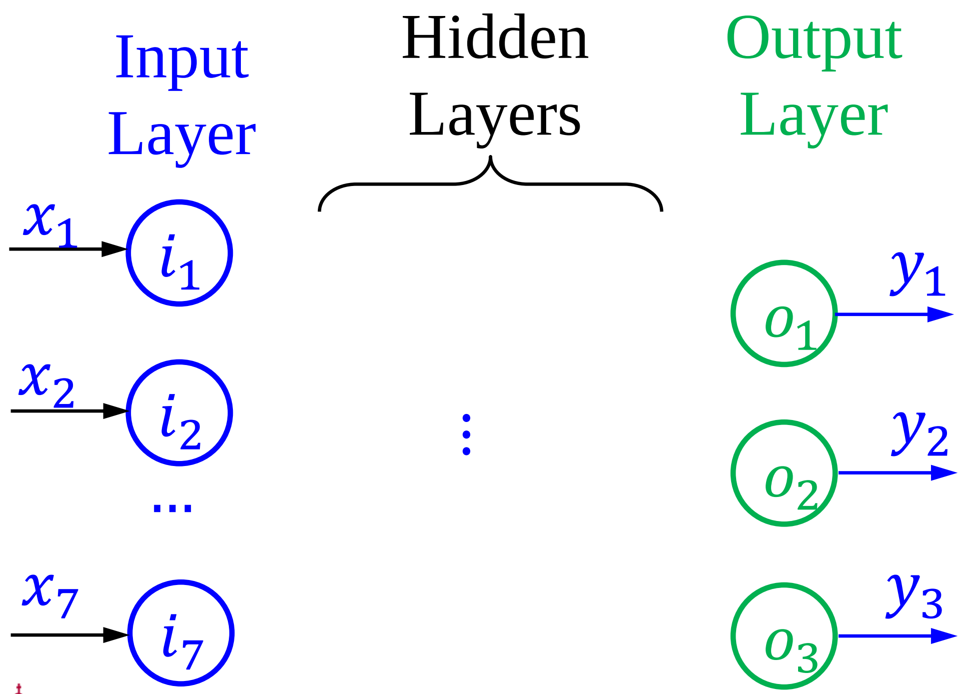
$$H_2 \Longrightarrow [3 \times 4]$$

$$Y \Longrightarrow [1 \times 2]$$

Supervised Learning Input Data

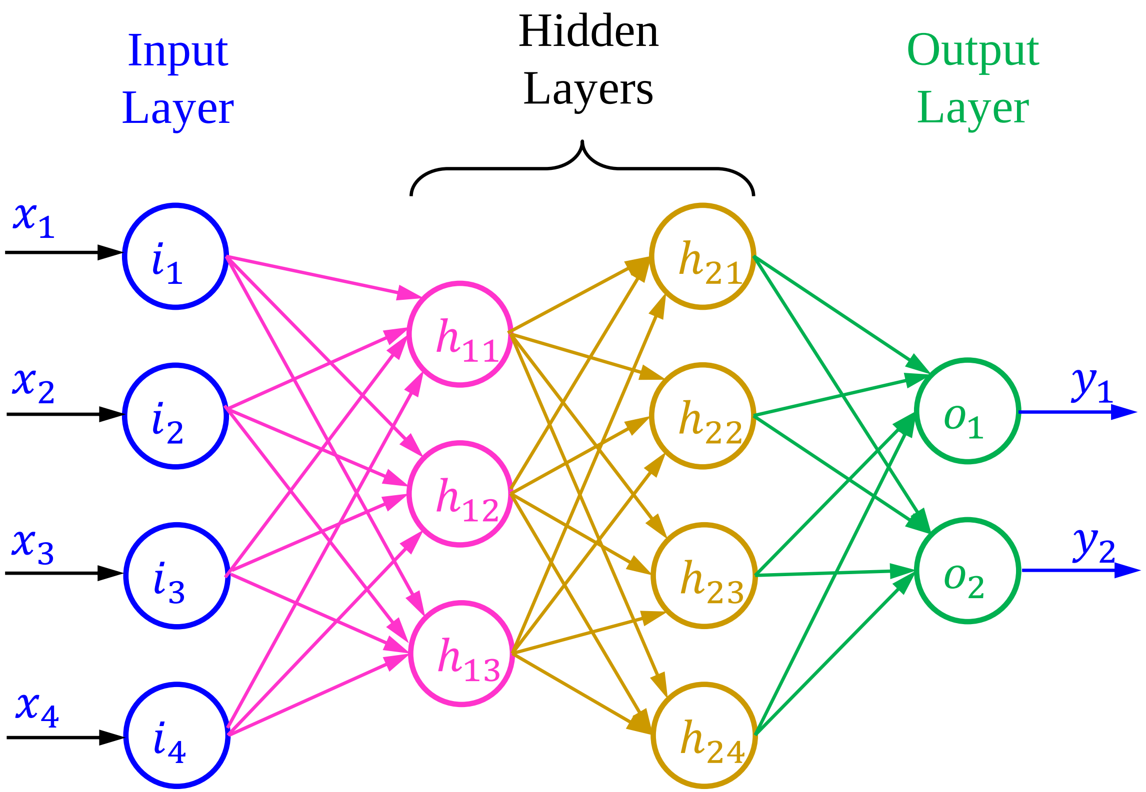
x_1	x_2	x_3	x_4	x_5	x_6	x_7	y_1
15.26	14.84	0.871	5.763	3.312	2.221	5.22	**1**

Expected Outputs	
o_1	1.0
o_2	0.0
o_3	0.0



- Assuming input data are preprocessed as row or column vectors for 3 classes.
- From the dataset, the network needs 7 input and 3 output neurons.

Feed-forward Networks



- How do we train an FF network?
- The most common training method is the backpropagation gradient descent algorithm.

Gradient

- Supposing a scalar-valued differentiable function $\mathcal{L}()$ has several variables (or N components) that influence the direction and rate of change of $\mathcal{L}()$, the vector component Θ is:

$$\Theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_N \end{bmatrix}$$

- The gradient (∇ nabla) of function $\mathcal{L}()$ is a vector contains the partial derivatives of $\mathcal{L}()$ with respect to each single component, θ_i for $i = 1, \ldots, N$:

$$\nabla\mathcal{L} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \theta_1} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial \theta_N} \end{bmatrix} = \left(\frac{\partial \mathcal{L}}{\partial \theta}\right)^T$$

https://en.wikipedia.org/wiki/Gradient

Stochastic Gradient Descent (SGD) (1)

Given a single pair of input-target (X_i, Y_i), where X_i is a vector input, $O_\Theta(X_i)$ is the vector outputs of a neural network, and Y_i is the vector of target outputs $(y_{i1}, y_{i2}, \dots, y_{iK}$ are the target outputs of K output neurons $O_{\Theta 1}(X_i), o_{\Theta 2}(X_i), \dots, o_{\Theta k}(X_i))$:

- The squared-error loss function for regression error is:

$$\mathcal{L}_i[O_\Theta(X_i), Y_i] = \frac{1}{K}\sum_{k=1}^{K}[y_{ik} - o_{\Theta k}(X_i)]^2$$

- The cross-entropy (or log) loss function for classification is:

$$\mathcal{L}_i[O_\Theta(X_i), Y_i] = -\frac{1}{K}\sum_{k=1}^{K}\left[y_{ik} log(o_{\Theta k}(X_i)) + (1 - y_{ik})\left(1 - log(o_{\Theta k}(X_i))\right)\right]$$

Santa Clara University

https://en.wikipedia.org/wiki/Gradient

Stochastic Gradient Descent (SGD) (2)

- Given a single pair of input-target (X_i, Y_i), the gradient descent (stochastic gradient descent) is:

$$\Delta\Theta = -\eta \nabla \mathcal{L}_i$$

where η is the learning rate ($\eta = 0.01$), and Θ and $\nabla \mathcal{L}_i$ have the same dimension.

Batch Gradient Descent (1)

- It is inefficient to update the components (or weights) of a neural network with each pair of input-target input (X_i, Y_i). Rather, input pairs are collected into a batch, and network weights are updated after each batch:

$$B = \{(X_1, Y_1), (X_2, Y_2), \dots, (X_M, Y_M)\}$$

where M is the size of batch B. When M is smaller than the total number of input pairs, it is called mini-batch.

- Experimental results show that when M is in a power of 2 such as 32, 64, 128, or 256, the computation utilizes fully the capacity of CPU cores or GPU cards in a high process computing (HPC) system.

https://en.wikipedia.org/wiki/Gradient

Batch Gradient Descent (2)

- In the batch mode, the loss function with respect to a component vector Θ of a neural network is defined as:

$$\mathcal{L}_M[\Theta] = \frac{1}{M}\sum_{k=1}^{M} \mathcal{L}_i[\Theta]$$

- The batch update in Gradient Descent is:

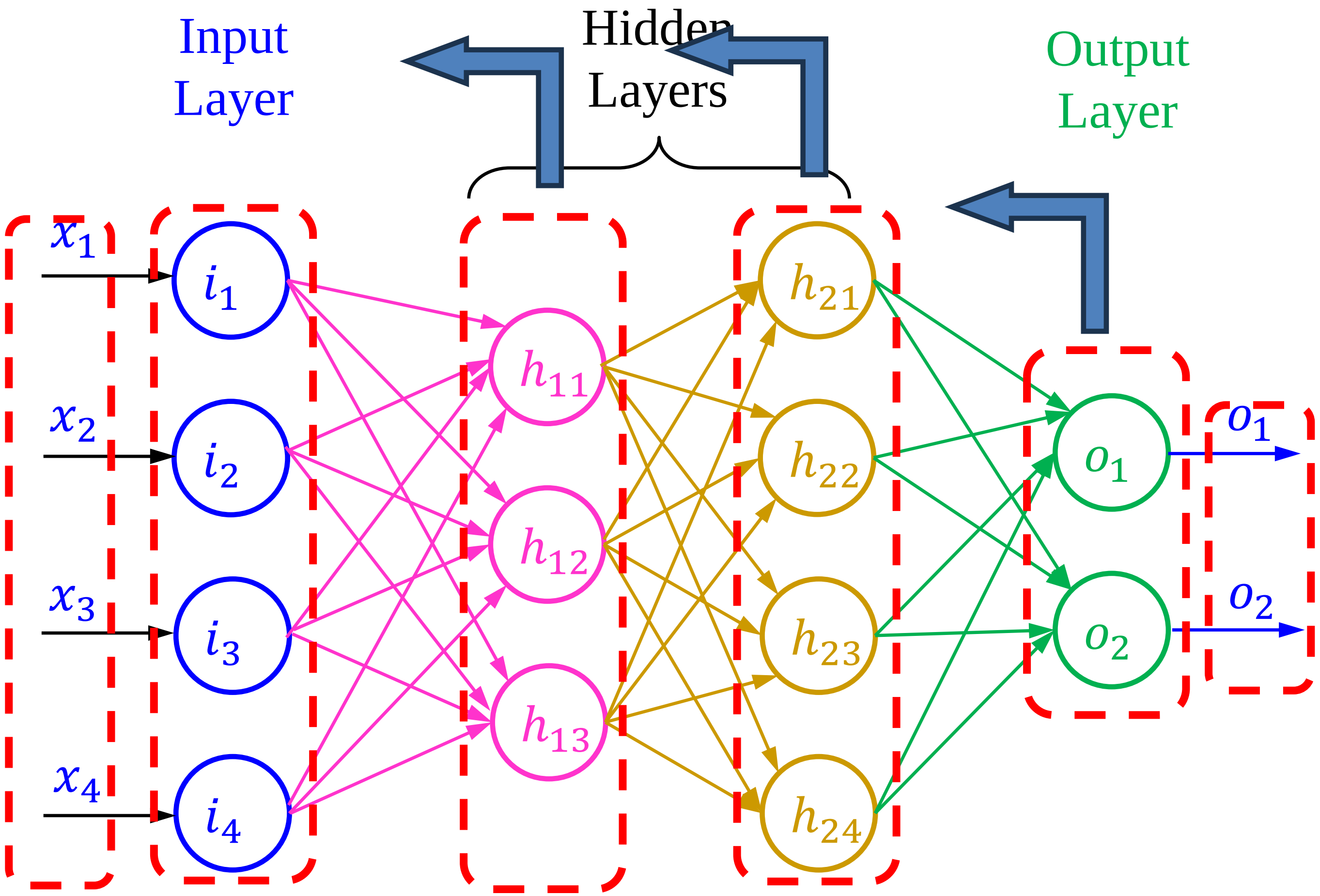
$$\Delta\Theta = -\eta\nabla\mathcal{L}_M$$

where η is the learning rate ($\eta = 0.01$), and Θ and $\nabla\mathcal{L}_M$ have the same dimension.

Backpropagation Gradient Descent (1)

- Backpropagation is a fundamental algorithm for training feedforward neural networks in supervised learning. It adjusts the weights of connections between neurons to minimize the error between the outputs and expected values (targets).
- The algorithm includes:
  1. Feedforward Pass: The network calculates outputs based on input vectors.
  2. Calculate Error: Using a cost function, the network computes errors based on the outputs and targets.
  3. Backpropagation: The network propagates the error backward and computes the gradient errors for weights in each layer.
  4. Gradient Descent: the network updates weights with a small learning rate (η)
  5. Repeat: repeat steps 1 – 4 many times (epochs) or until the error converges
  6. Termination: terminate the process and test the network for classification.

Backpropagation Gradient Descent (2)



$[1 \times 4] * [4 \times 3] * [3 \times 4] * [4 \times 2] \implies [1 \times 2]$

- Feedforward Pass: $O = [o_1, o_2]$
- Calculate Error:

$$target \implies Y = [y_1, y_2]$$

$$C(O,Y) \implies \nabla C(O,Y) = \begin{bmatrix} \frac{\partial C(o_1, y_1)}{\partial o_1} \\ \frac{\partial C(o_2, y_2)}{\partial o_2} \end{bmatrix}$$

- Backpropagation:
  - H2: $\nabla_{H2} \implies \nabla C(O,Y) \times [4 \times 2]^T$
  - H1: $\nabla_{H1} \implies \nabla_{H2} \times [3 \times 4]^T$
  - I: $\nabla_{I} \implies \nabla_{H1} \times [4 \times 3]^T$
- Gradient Descent

Santa Clara University

Backpropagation Gradient Descent (3)

Notation:

- L: the number of layers.
- W^l: the matrix weight between layer $l-1$ and l.
- φ^l: the activation function at layer l:
  a. In the last layer (or output layer), the activation function is sigmoid (logistic) for binary classification softmax for multi-class classification.
  b. In the hidden layers, the activation function is sigmoid or ReLU.
- a^l: the activation at layer l.
- δ^l: the error at the layer l.
- $\nabla_L C$ or $\nabla_L \mathcal{L}$: the gradient of the cost (or loss) function at the output layer.
- $\odot$ or $\circ$: is the Hadamard product (elementwise products of two matrices).

Santa Clara University

https://en.wikipedia.org/wiki/Backpropagation

Backpropagation Gradient Descent (4)

- The derivative of the cost function C at the output layer (or last layer L) is:

$$\frac{dC}{dO} = \frac{dC}{da^L}$$

- The derivative of the cost function C in terms of input X with a chain rule:

$$\frac{dC}{da^L} * \frac{da^L}{dW^L} * \frac{dW^L}{da^{L-1}} * \frac{da^{L-1}}{dW^{L-1}} * \frac{dW^{L-1}}{da^{L-2}} * \cdots * \frac{da^1}{dW^1} * \frac{dW^1}{dX} \qquad (1)$$

- We have:

$$\frac{da^l}{dW^l} = \circ\,(\varphi^l)' \; and \; \frac{dW^l}{da^{l-1}} = (W^l x)' = W^l$$

- Changing terms in Eq 1:

$$\frac{dC}{da^L} \circ (\varphi^L)' * W^L \circ (\varphi^{L-1})' * W^{L-1} \circ \cdots \circ (\varphi^1)' * W^1 \qquad (2)$$

Santa Clara University

https://en.wikipedia.org/wiki/Backpropagation

Backpropagation Gradient Descent (5)

- The error at layer l :

$$\delta^1 = (\varphi^1)' \circ (W^2)^T \cdot (\varphi^2)' \circ \cdots \circ (W^{L-1})^T \cdot (\varphi^{L-1})' \circ (W^L)^T \cdot (\varphi^L)' \circ \nabla_L C$$
$$\delta^2 = (\varphi^2)' \circ \cdots \circ (W^{L-1})^T \cdot (\varphi^{L-1})' \circ (W^L)^T \cdot (\varphi^L)' \circ \nabla_L C$$
$$\vdots$$
$$\delta^{L-1} = (\varphi^{L-1})' \circ (W^L)^T \cdot (\varphi^L)' \circ \nabla_L C$$
$$\delta^L = (\varphi^L)' \circ \nabla_L C$$

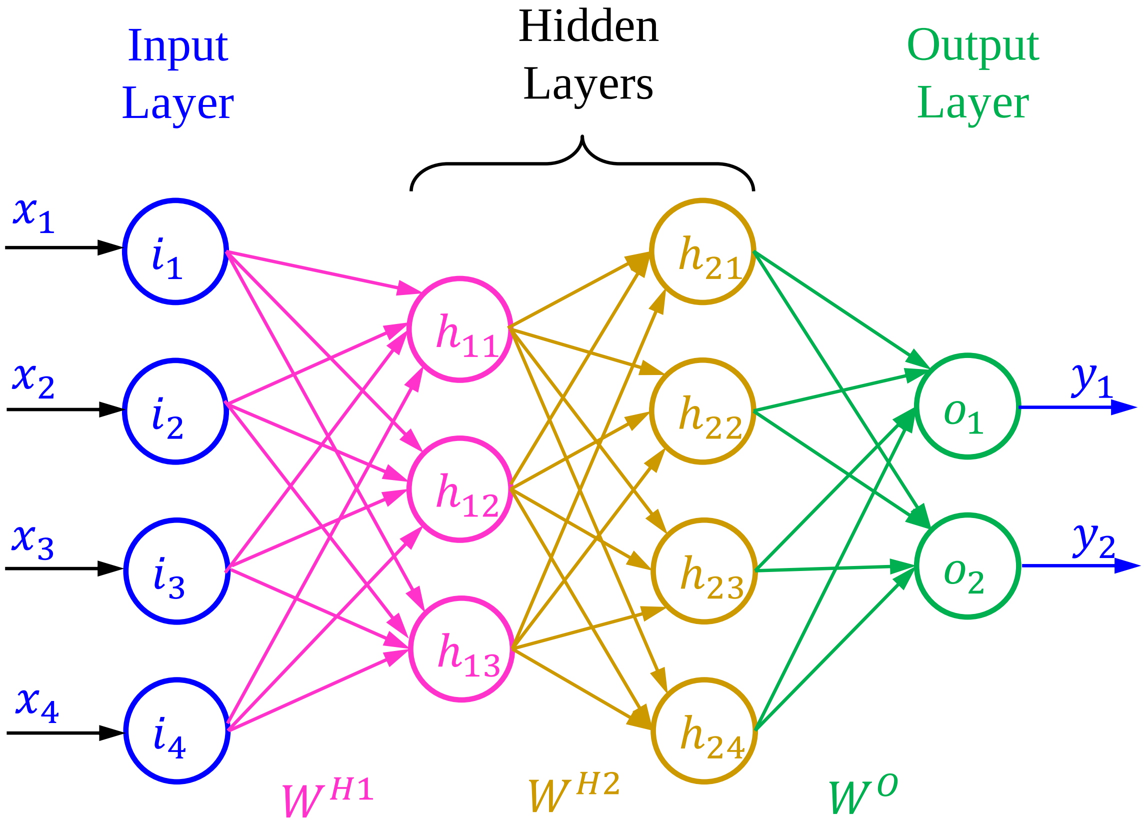
- The gradient of the weights and bias in layer l is then:

$$\nabla_{W^l} C = \delta^l \left(a^{l-1}\right)^T, \quad \nabla_{B^l} C = \delta^l$$

- The weight and bias update in layer l is:

$$\Delta W^l = -\eta \nabla_{W^l} C, \quad \Delta B^l = -\eta \nabla_{B^l} C$$

1. Feedforward Pass (1)



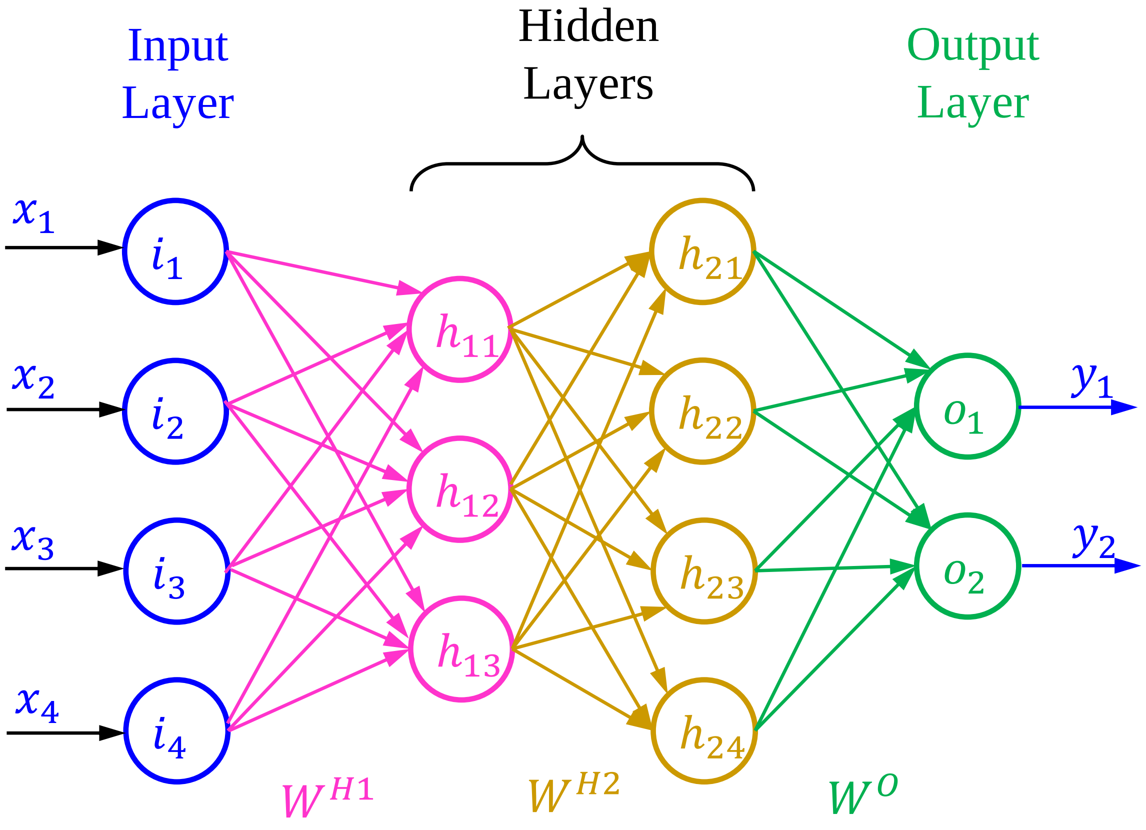
$$P^{h1} = X \times \overbrace{\begin{bmatrix} w_{11}^{h1} & w_{12}^{h1} & w_{13}^{h1} \\ w_{21}^{h1} & w_{22}^{h1} & w_{23}^{h1} \\ w_{31}^{h1} & w_{32}^{h1} & w_{33}^{h1} \\ w_{41}^{h1} & w_{42}^{h1} & w_{43}^{h1} \end{bmatrix}}^{W^{H1}}$$

$$H^{H1} = \varphi\left(P^{h1} + B_{h1}\right)$$

$$P^{h2} = H^{H1} \times \overbrace{\begin{bmatrix} w_{11}^{h2} & w_{12}^{h2} & w_{13}^{h2} & w_{14}^{h2} \\ w_{21}^{h2} & w_{22}^{h2} & w_{23}^{h2} & w_{24}^{h2} \\ w_{31}^{h2} & w_{32}^{h2} & w_{33}^{h2} & w_{34}^{h2} \end{bmatrix}}^{W^{H2}}$$

$$H^{H2} = \varphi\left(P^{h1} + B_{h2}\right)$$

1. Feedforward Pass (2)

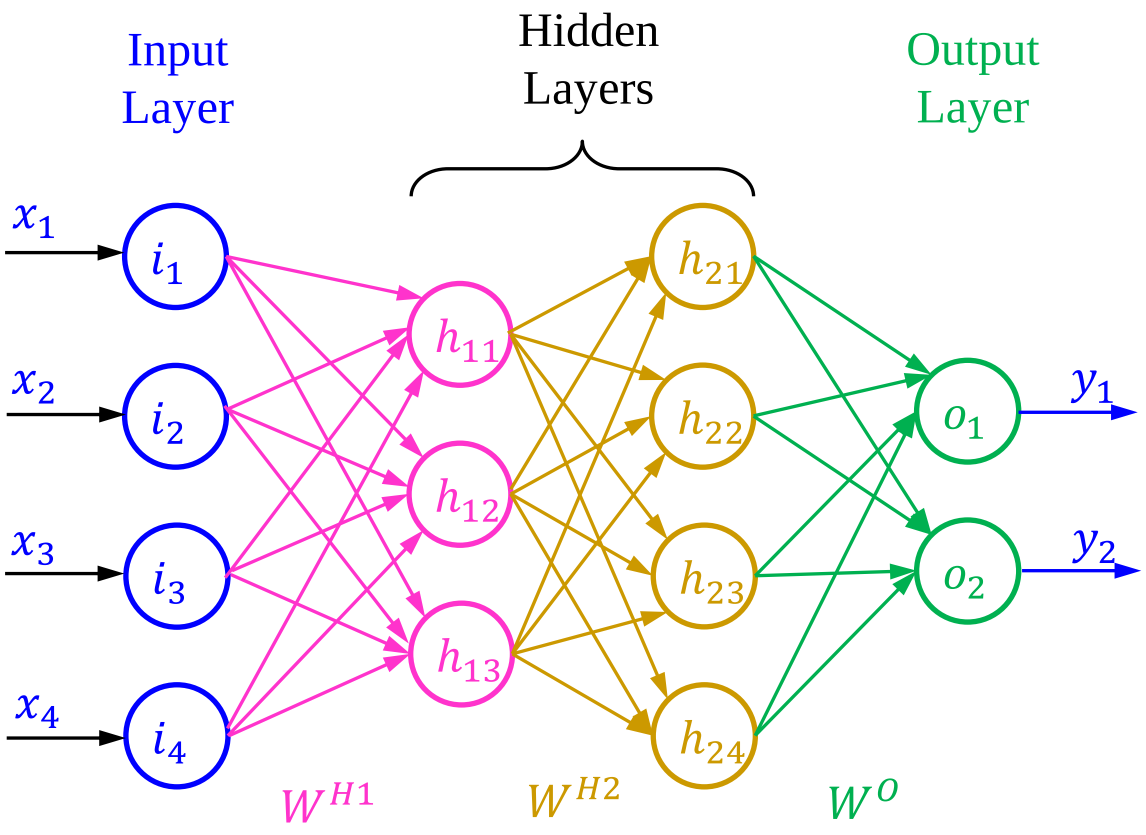


$$P^{O} = H^{H2} \times \overbrace{\begin{bmatrix} w^{o}_{11} & w^{o}_{12} \\ w^{o}_{21} & w^{o}_{22} \\ w^{h2}_{31} & w^{h2}_{32} \\ w^{o}_{41} & w^{o}_{42} \end{bmatrix}}^{W^{O}}$$

$$Y = O = \varphi(P^{O} + B_{o})$$

Santa Clara University

2. Calculate Error (1)



- Error (E) is a cost function or loss function (C):

$$Y = O = \varphi(P^O + B_o)$$
$$E = C(Y_t, Y)$$

- For classification, the loss function (C) is cross-entropy (log loss):

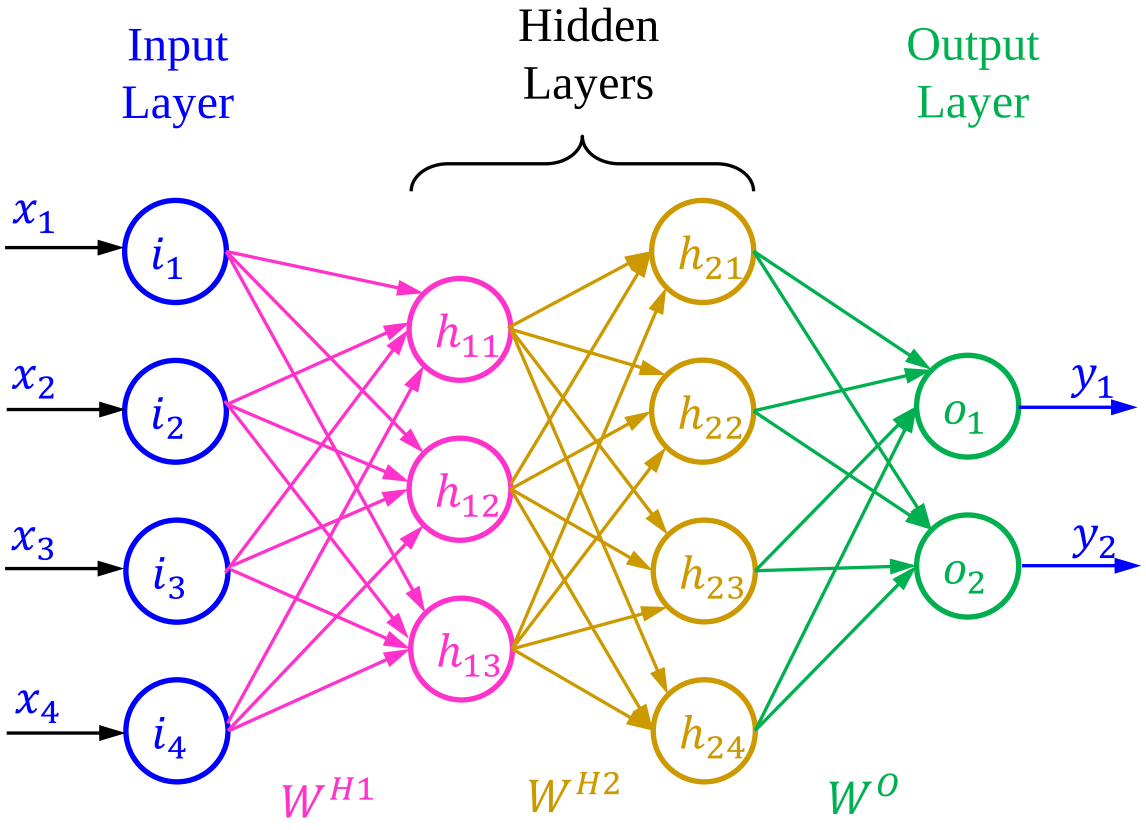
$$C(Y_i, O) = -\frac{1}{n_o}\sum_{k=1}^{n_o}[y_k log(o_k) + (1 - y_k) log(1 - o_k)]$$

where n is the number of batches, n_o is the number of output neurons, y_k and o_k are the target and actual output values at neron i

https://en.wikipedia.org/wiki/Cross-entropy

Santa Clara University

2. Calculate Error (2)



- For regression, the loss function is usually squared error loss (SE):

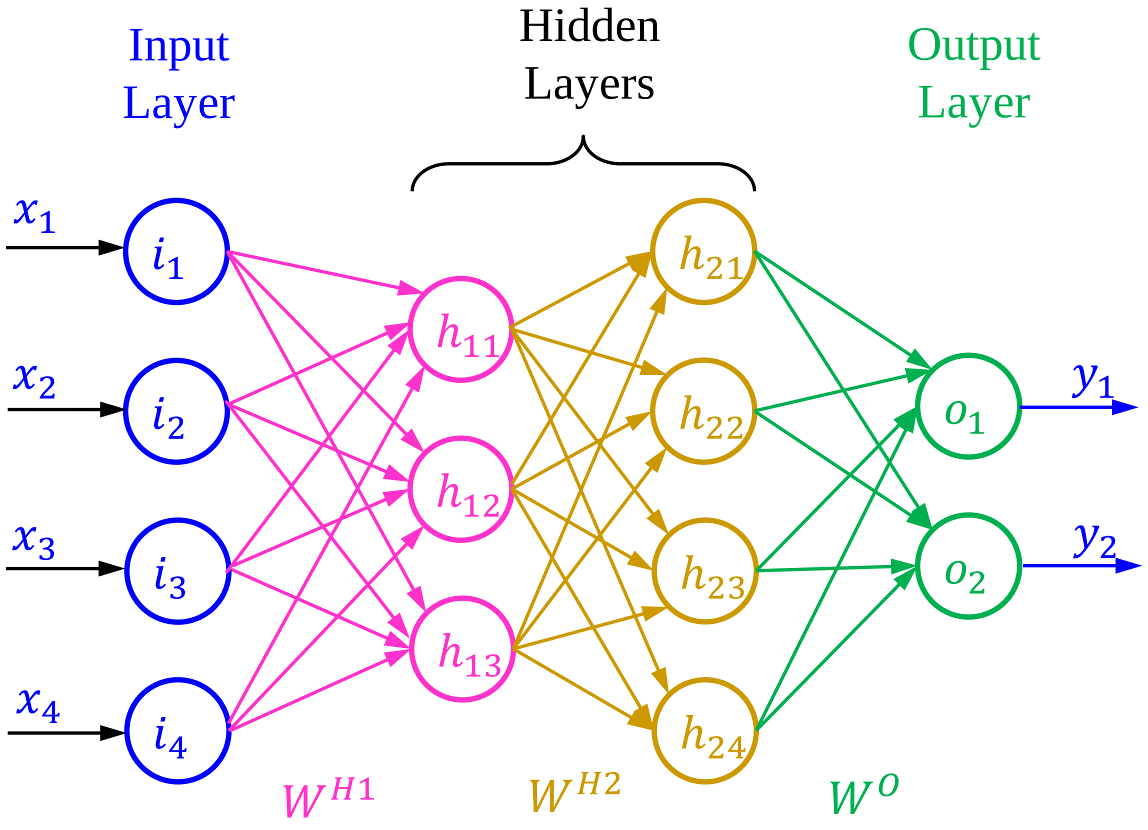
$$C(Y_i, O) = SE = \frac{1}{2}\sum_{k=1}^{n_o}(y_k - o_k)^2$$

https://en.wikipedia.org/wiki/Least_squares

- Assuming for classification, we use the cross-entropy loss function $\nabla_O C$:

$$\begin{bmatrix} \frac{\partial e_1}{\partial o_1} \\ \frac{\partial e_2}{\partial o_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial\left[(-1)\left(y_1 log(o_1) + (1-y_1)log(1-o_1)\right)\right]}{\partial o_1} \\ \frac{\partial\left[(-1)\left(y_2 log(o_2) + (1-y_2)log(1-o_2)\right)\right]}{\partial o_2} \end{bmatrix}$$

2. Calculate Error (3)



- $\nabla_O C$:

$$\begin{bmatrix} \frac{\partial e_1}{\partial o_1} \\ \frac{\partial e_2}{\partial o_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial[(-1)(y_1 log(o_1) + (1-y_1) log(1-o_1))]}{\partial o_1} \\ \frac{\partial[(-1)(y_2 log(o_2) + (1-y_2) log(1-o_2))]}{\partial o_2} \end{bmatrix}$$

- We have:

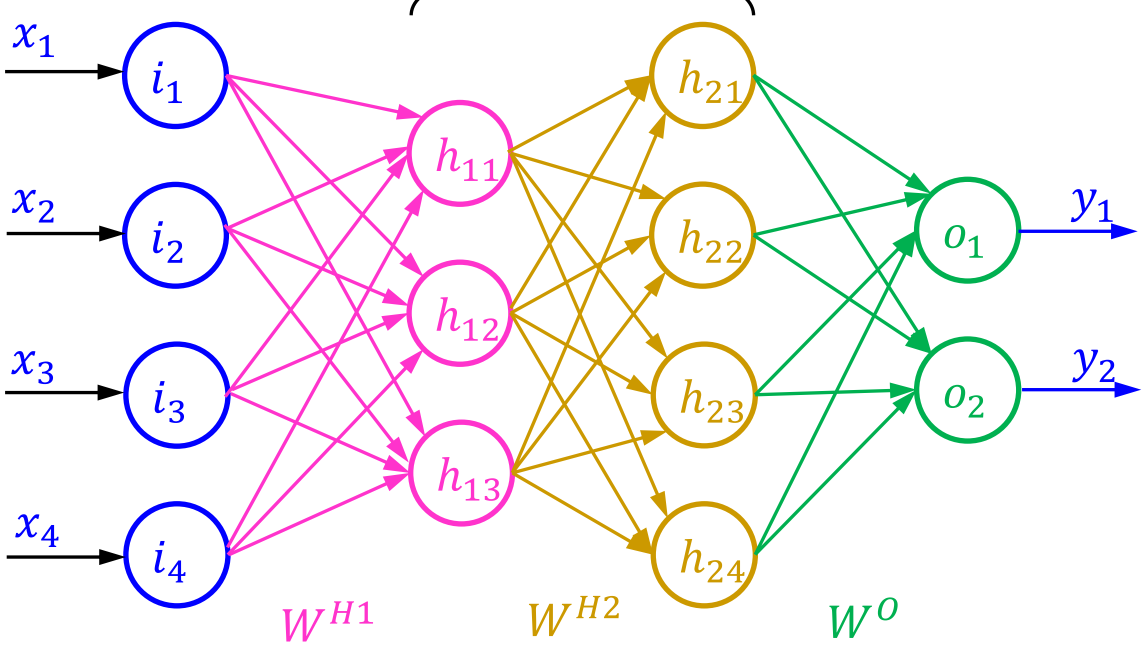
$$\frac{d}{dx} log_a(x) = \frac{1}{x * ln(a)}$$

$$\frac{d}{dx} log_a(1-x) = -\frac{1}{ln(a) - x * ln(a)}$$

- $\nabla_O C$:

$$\begin{bmatrix} \frac{\partial e_1}{\partial o_1} \\ \frac{\partial e_2}{\partial o_2} \end{bmatrix} = \begin{bmatrix} (-y_1/[o_1 ln(10)]) + ([1-y_1]/[ln(10) - o_1 ln(10)]) \\ (-y_2/[o_2 ln(10)]) + ([1-y_2]/[ln(10) - o_2 ln(10)]) \end{bmatrix}$$

3. Backpropagate Error to Weights



- Derivative of activation function:

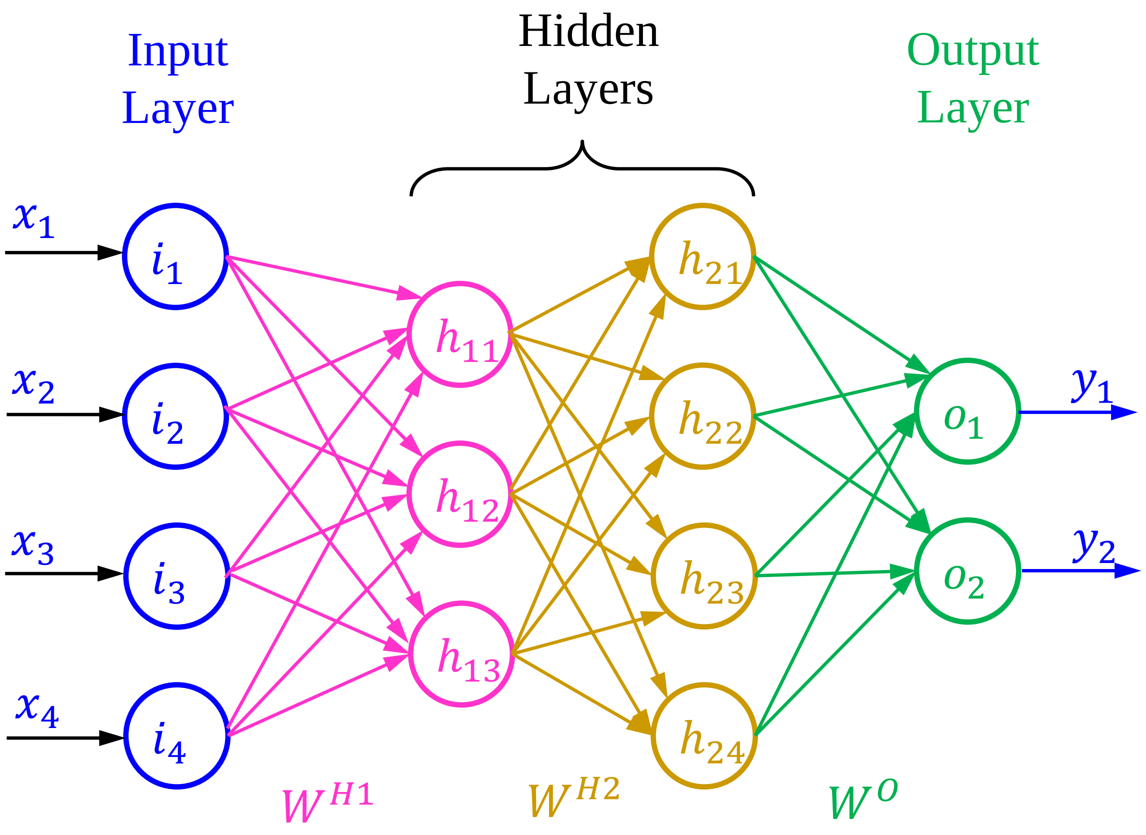
$$\varphi(x) = \frac{1}{1+e^{-x}}$$
$$\varphi'(x) = \varphi(x)[1-\varphi(x)]$$

- Calculate δ^l (error function) at each layer:

$$\delta^O = [\varphi^O(W^O + B_O)]' \circ \nabla_O C$$
$$\delta^{H2} = \{[\varphi^{H2}(W^{H2} + B_{H2})]' \circ (W^O)^T\} \cdot \delta^O$$
$$\delta^{H1} = \{[\varphi^{H1}(W^{H1} + B_{H1})]' \circ (W^{H2})^T\} \cdot \delta^{H2}$$

http://neuralnetworksanddeeplearning.com/chap2.html

4. Calculate Gradient Descent Weight and Bias Updates



- Learing rate $\eta = 0.01$
- Calculate weight and bias updates:

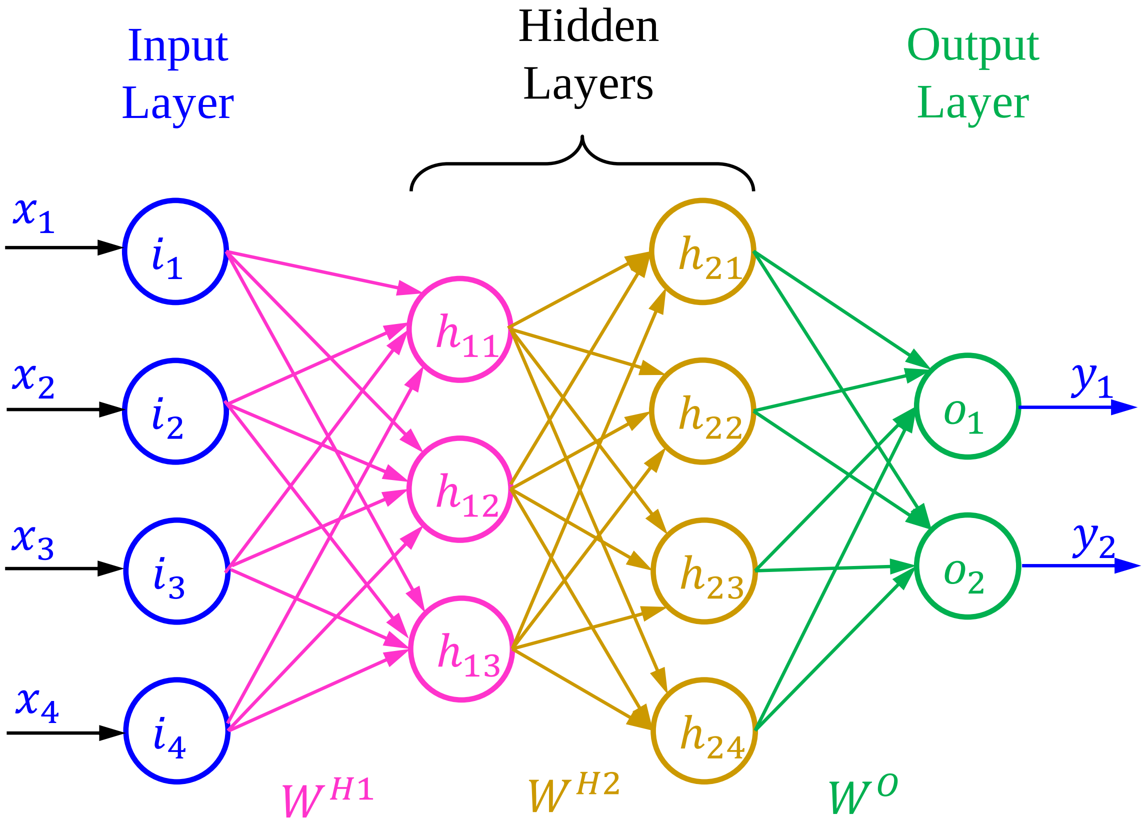
$$\Delta W^{O} = -\eta\delta^{O}(H_2)^T$$
$$\Delta B^{O} = -\eta\delta^{O}$$

$$\Delta W^{H2} = -\eta\delta^{H2}(H_1)^T$$
$$\Delta B^{H2} = -\eta\delta^{H2}$$

$$\Delta W^{H1} = -\eta\delta^{H1}(I)^T$$
$$\Delta B^{H1} = -\eta\delta^{H1}$$

http://neuralnetworksanddeeplearning.com/chap2.html

5. Update Weights and Bias and Repeat



- Update weights and biases:

$$W^O = W^O + \Delta W^O$$
$$B^O = B^O + \Delta B^O$$

$$W^{H2} = W^{H2} + \Delta W^{H2}$$
$$B^{H2} = B^{H2} + \Delta B^{H2}$$

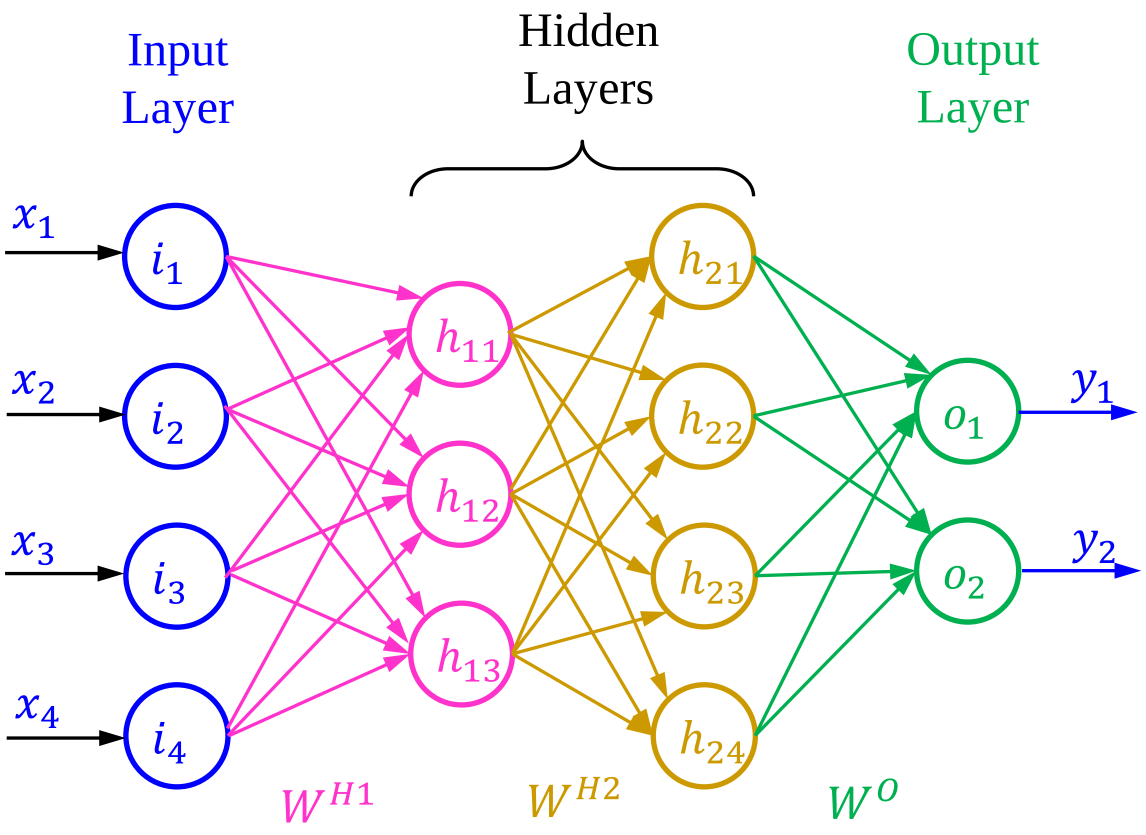
$$W^{H1} = W^{H1} + \Delta W^{H1}$$
$$B^{H1} = B^{H1} + \Delta B^{H1}$$

- Repeat the process until the error converges

http://neuralnetworksanddeeplearning.com/chap2.html

6. Test Network

Winner-take-all (WTA)
$o_1 \rightarrow o_1 = 1, o_2 = 0$



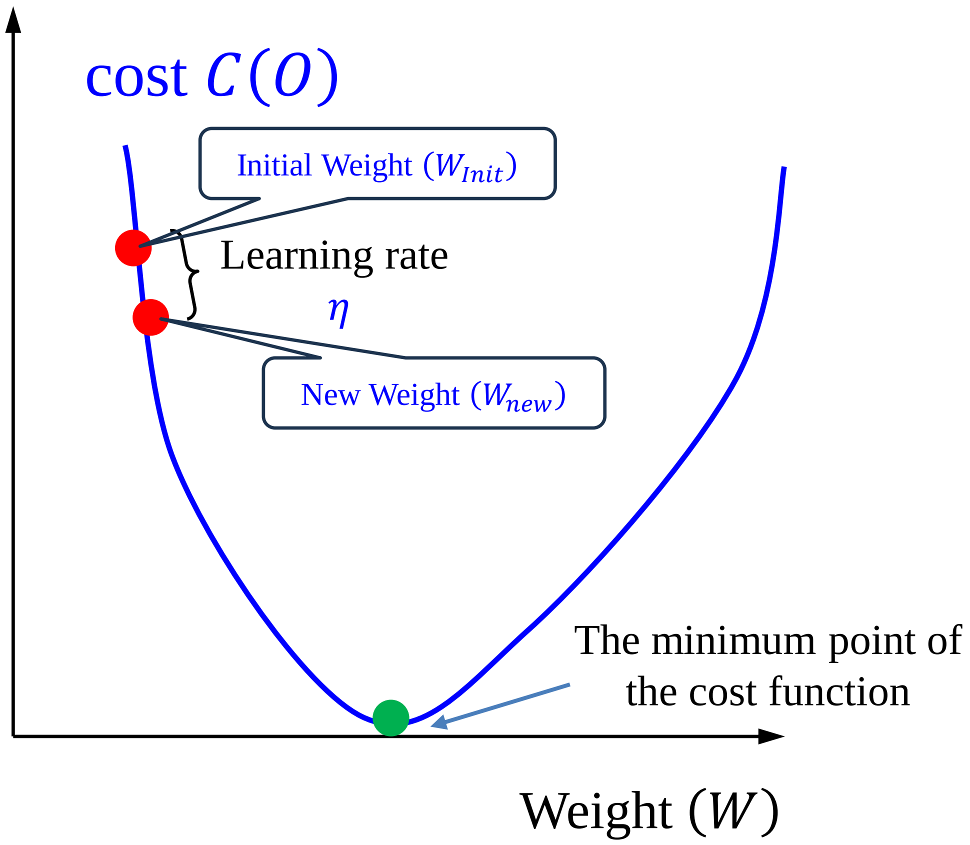
- Terminate the process.
- Test the network with test inputs for classification.

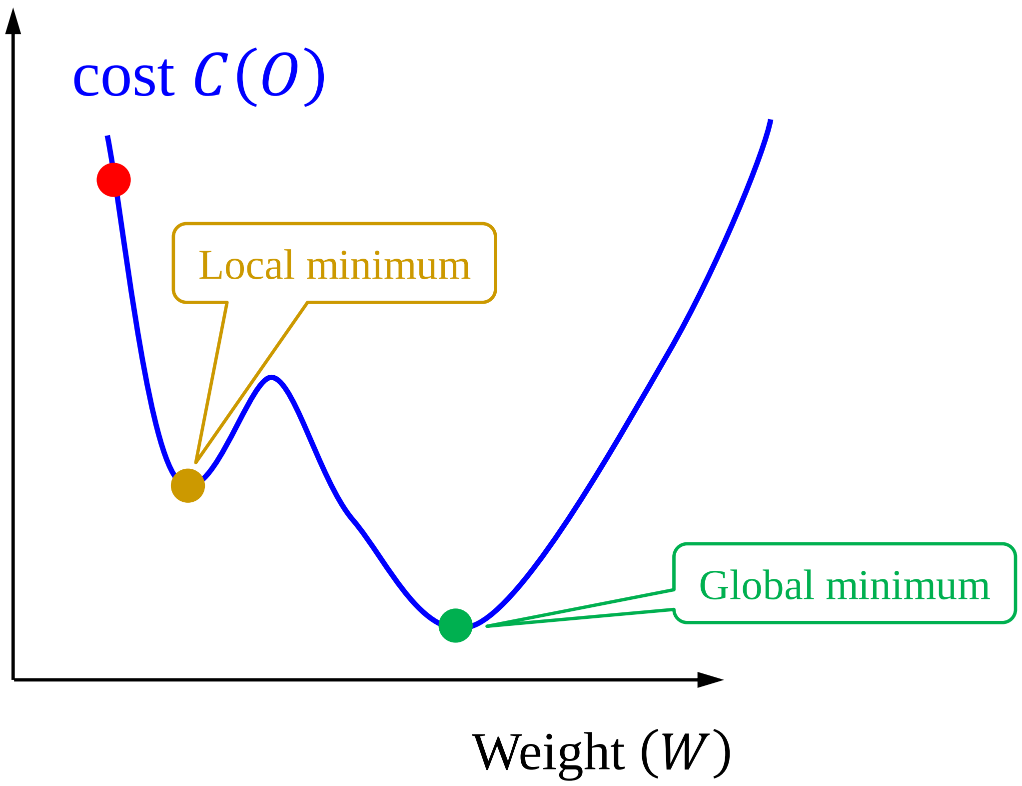
	Outputs	Targets
o_1	0.82	1.0
o_2	0.31	0.0

- Report the classification performance.

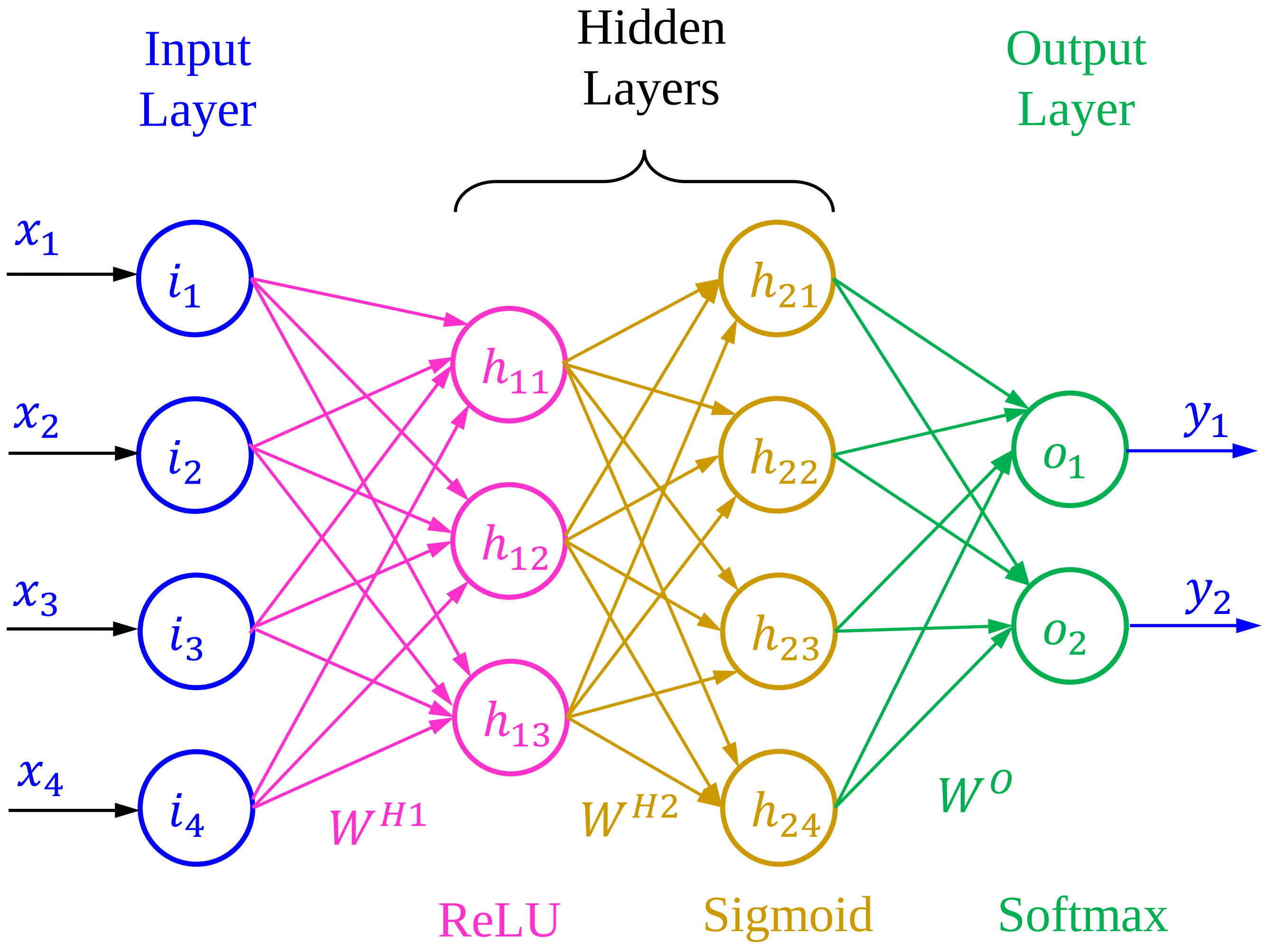
http://neuralnetworksanddeeplearning.com/chap2.html

Backpropagation Gradient Descent

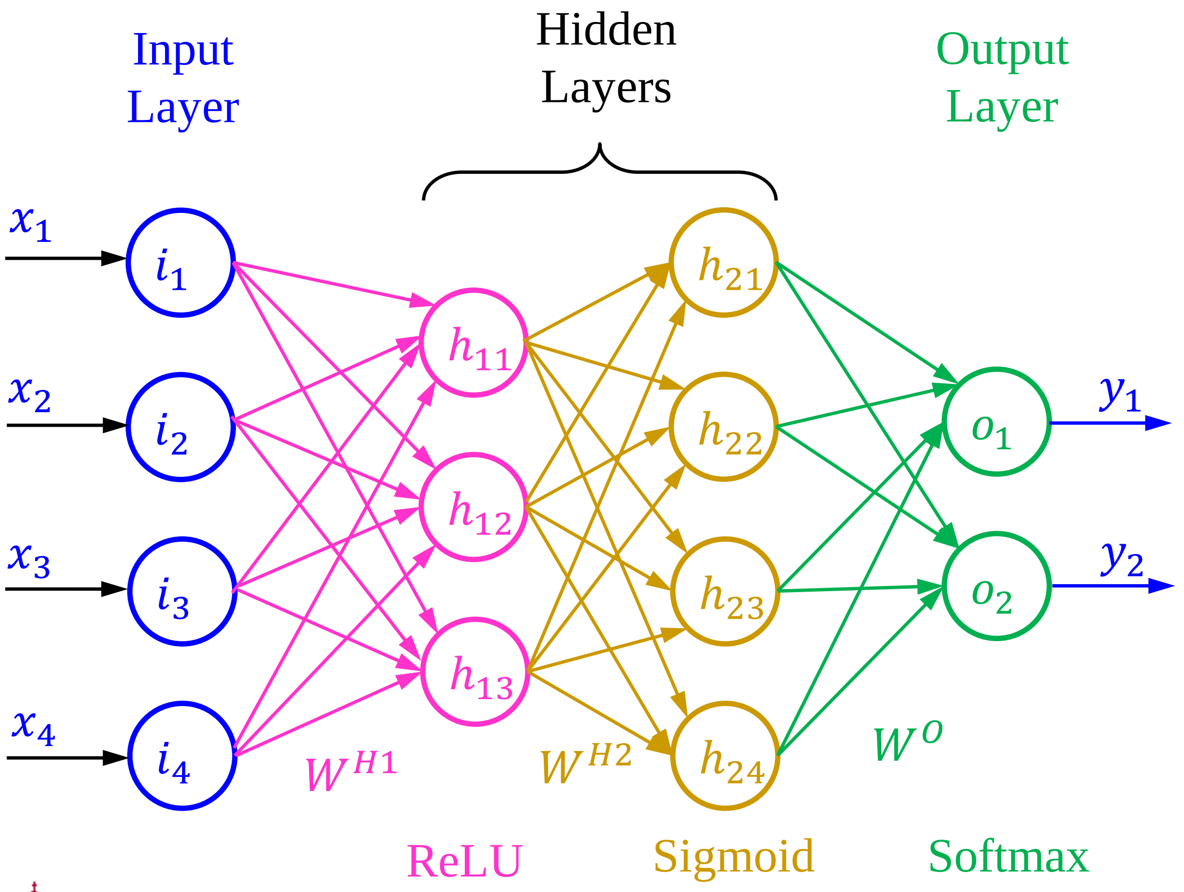




Backpropagation Gradient Descent Example (1)



Backpropagation Gradient Descent Example (2)



- Activation functions and their derivatives:
- ReLU:

$$\varphi_{H1}(x) = max(0, x)$$

$$\varphi'_{H1}(x) = \begin{cases} 1 \; if \; x > 0 \\ 0 \; if \; x \leq 0 \end{cases}$$

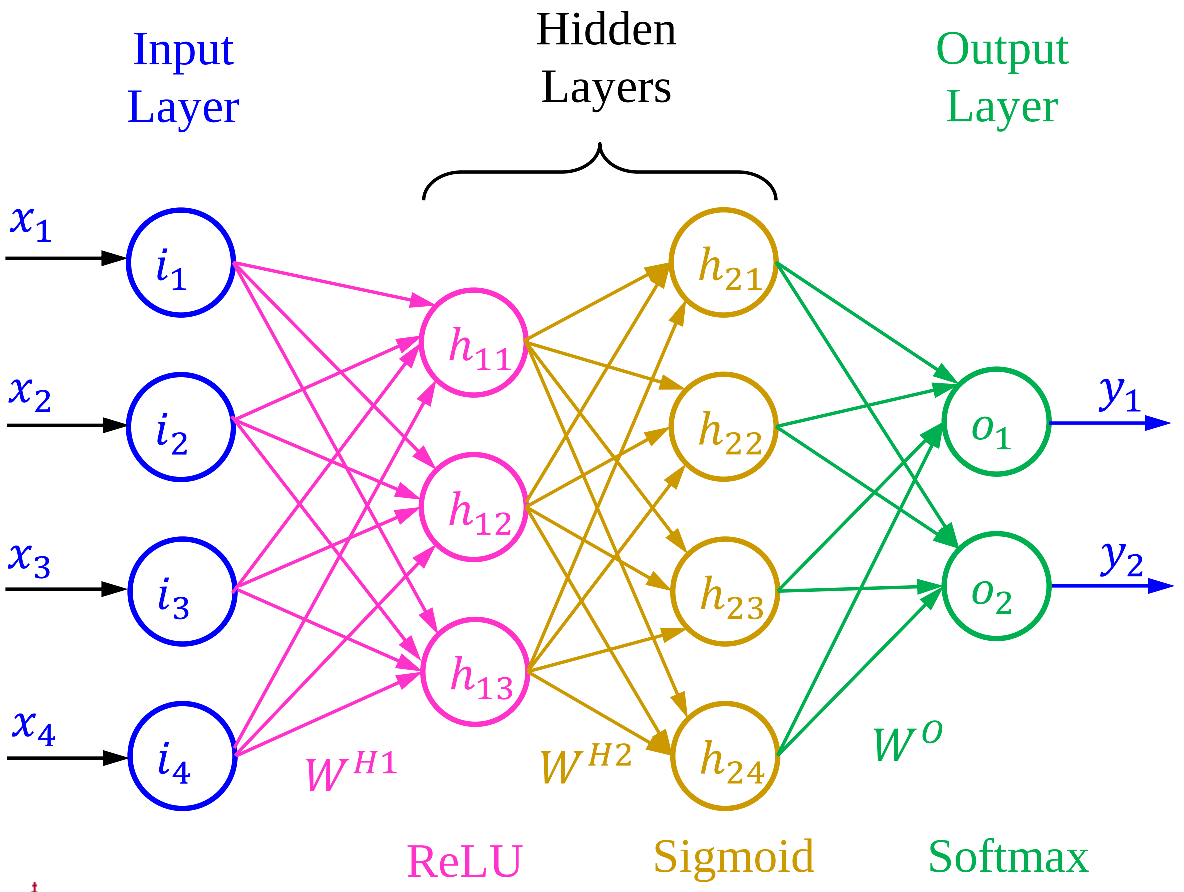
- Sigmoid:

$$\varphi_{H2}(x) = \frac{1}{1 + e^{-x}}$$

$$\varphi'_{H2}(x) = \varphi(x)[1 - \varphi(x)]$$

https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html

Backpropagation Gradient Descent Example (3)



- Softmax:

$$O_v = H2 \times W^O + B_O$$

$$o_i = \frac{e^{o_{vi}}}{\sum_{k=1}^{n_o} e^{o_{vk}}}$$

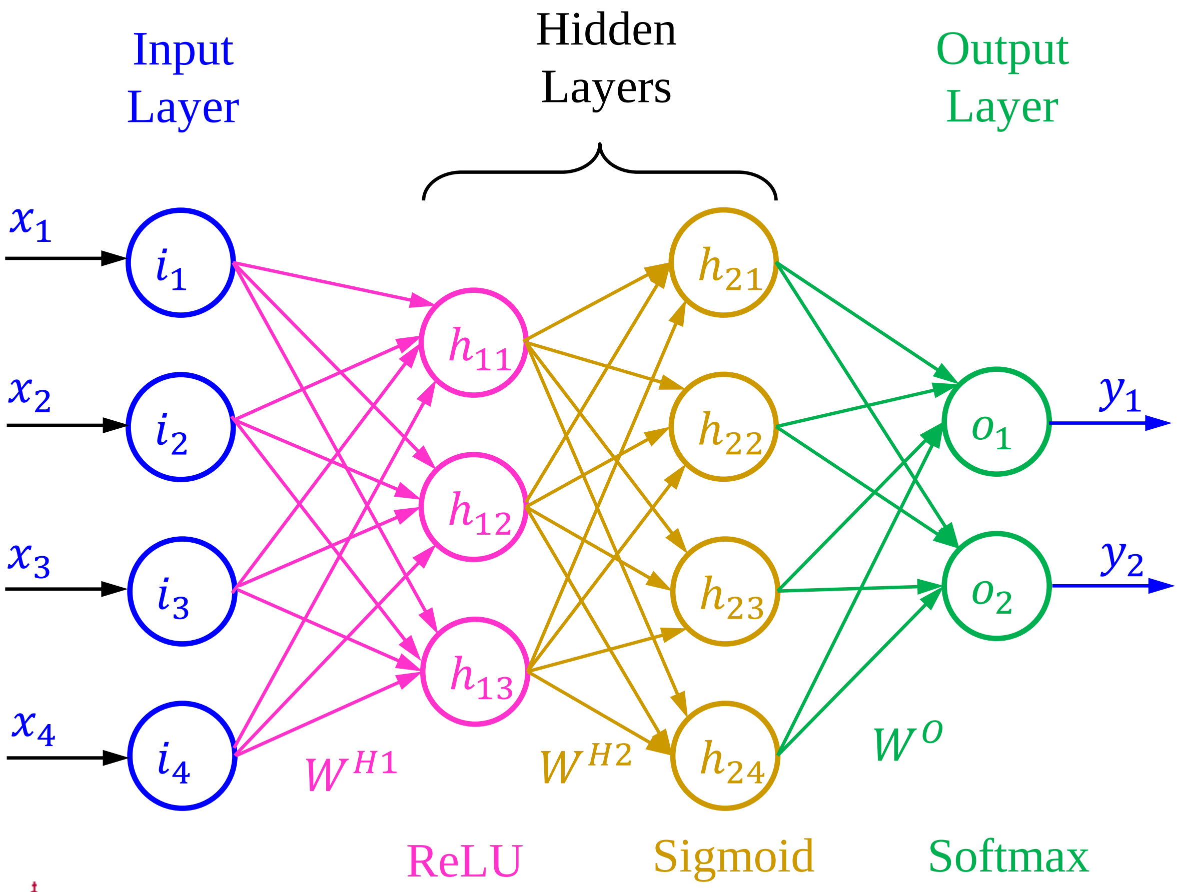
where O_v is the vector output, o_{vi} is the output at neuron i, n_o is the number of output neurons, o_i is the softmax value of output neuron i

- Derivative of Softmax:

$$\frac{\partial s_i}{\partial z_j} = s_i \frac{\partial log(s_i)}{\partial z_j}$$

https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html

Backpropagation Gradient Descent Example (4)

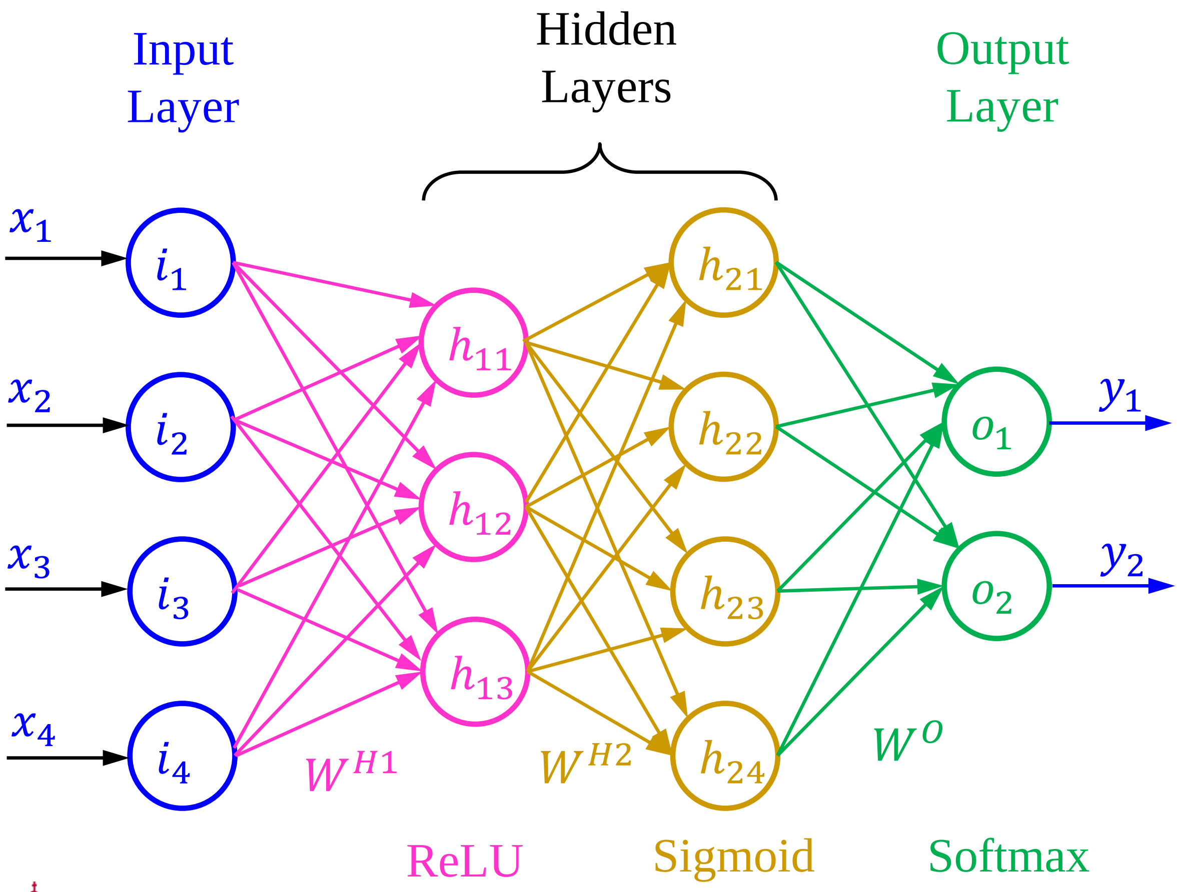


- derivative:

$$\begin{bmatrix} \frac{\partial o_1}{\partial o_{v1}} \\ \frac{\partial o_2}{\partial o_{v2}} \\ \vdots \\ \frac{\partial o_{n_o}}{\partial o_{vn_o}} \end{bmatrix} = \begin{bmatrix} \frac{e^{o_{v1}}\left(\sum_{k\neq 1}^{n_o} e^{o_{vk}}\right)}{\left(\sum_{k=1}^{n_o} e^{o_{vk}}\right)^2} \\ \frac{e^{o_{v2}}\left(\sum_{k\neq 2}^{n_o} e^{o_{vk}}\right)}{\left(\sum_{k=1}^{n_o} e^{o_{vk}}\right)^2} \\ \vdots \\ \frac{e^{o_{vn_o}}\left(\sum_{k\neq n_o}^{n_o} e^{o_{vk}}\right)}{\left(\sum_{k=1}^{n_o} e^{o_{vk}}\right)^2} \end{bmatrix}$$

https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html

Backpropagation Gradient Descent Example (5)



- Follow the similar steps to propagate error backward to each layer.

https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html

Python Code Activation Function and Its derivatives

```
# ***************************************************
def ReLU(Z):
    '''ReLU function
    Inputs:
        Z: a 2d matrix (mxn): m mini-batch, n output neurons
    Returns: ReLU values
    '''
    return np.maximum(0, Z)

# ****************************************************
def ReLUDervivative(Z):
    '''Derivative of ReLU function
    Inputs:
        Z: a 2d matrix (mxn): m mini-batch, n output neurons
    Returns: derivative of ReLU values
    '''
    Z[Z <= 0.0] = 0.0
    Z[Z > 0.0]  = 1.0
    return Z
```

```
# ***************************************************
def Sigmoid(Z):
    '''Sigmoid function
    Inputs:
        Z: a 2d matrix (mxn): m mini-batch, n output neurons
    Returns: Sigmoid values
    '''
    return np.divide(1.0, np.add(1.0, np.exp(-Z)))

# ***************************************************
def SigmoidDerivative(Z):
    '''Derivative Sigmoid function
    Inputs:
        Z: a 2d matrix (mxn): m mini-batch, n output neurons
    Returns: derivative Sigmoid values
    '''
    return np.multiply(Sigmoid(Z), np.sub(1.0, Sigmoid(Z)))
```

Python Code Activation Function and Its derivatives

```
# ************************************************
def Softmax(Z):
    '''SoftmaxDerivative function
    Inputs:
        Z: a 2d matrix (mxn): m mini-batch, n output neurons
    Returns: softmax values
    '''
    ExpVals     = np.exp(np.subtract(Z, np.max(Z)))
    ExpValSum   = np.sum(ExpVals)
    return np.divide(ExpVals, ExpValSum)
```

```
# ************************************************
def SoftmaxDerivative(Z): # Best implementation (VERY FAST)
    '''Returns the jacobian of the Softmax function.
    Inputs:
        x: should be a 2d (mxn) matrix where m corresponds to             the
samples (or mini-batch), and n is the number
                    of nodes.
    Returns: jacobian derivative of softmax
    reference: https://www.bragitoff.com/2021/12/efficient-implementation-of-
softmax-activation-function-and-its-derivative-jacobian-in-python/
    '''
    s     = Softmax(Z)
    a     = np.eye(s.shape[-1])
    Temp1 = np.zeros((s.shape[0], s.shape[1], s.shape[1]),dtype=np.float32)
    Temp2 = np.zeros((s.shape[0], s.shape[1], s.shape[1]),dtype=np.float32)
    Temp1 = np.einsum('ij,jk->ijk', s, a)
    Temp2 = np.einsum('ij,ik->ijk', s, s)
    return np.subtract(Temp1, Temp2)
```

Santa Clara University

Python Code Activation Function and Its derivatives

```
# ***************************************************
def CrossEntropy(Outputs, Targets):
    '''Cross entropy loss function
    Inputs:
        Outputs: a 2d matrix (mxn): m mini-batch, n output
                neurons
        Targets: a 2d matrix (mxn): m mini-batch, n expected
                values
    Returns: average derivative of cross entropy loss function
    '''
        Loss    = -np.mean(np.add(np.multiply(Targets, np.log(Outputs)),
        \
         np.multiply((1.0 - Targets), np.log(1.0 - Outputs))))
    return Loss
```

```
# ***********************************************************
def CrossEntropyDerivative(Outputs, Targets):
    '''Derivative of Cross entropy loss function
    Inputs:
        Outputs: a 2d matrix (mxn): m mini-batch, n output neurons
        Targets: a 2d matrix (mxn): m mini-batch, n expected values
    Returns: average derivative of cross entropy loss function
    '''
    DeriVector = np.add(np.divide(-Targets, (Outputs * np.log(10))), \
          np.divide((1.0 - Targets), (np.log(10) * (1 - Outputs))))
    return DeriVector
```