CSC4007 Advanced Machine Learning

Lesson 08: Neural Networks

by Vien Ngo
EEECS / ECIT / DSSC

Some slides adapted from Barry Devereux,Geoffrey Hinton, Andrew Ng, Kenrick Mock

Outline

- Neural network basics and representation
- Perceptron learning, multi-layer perceptron
- Nueral network training: Backpropagation
- Modern neural network architecture (a.k.a Deep learning):
  - Convolutional neural network (CNN)
  - Recurrent neural network (RNN), long-short term memory network (LSTM)

Neural Network History

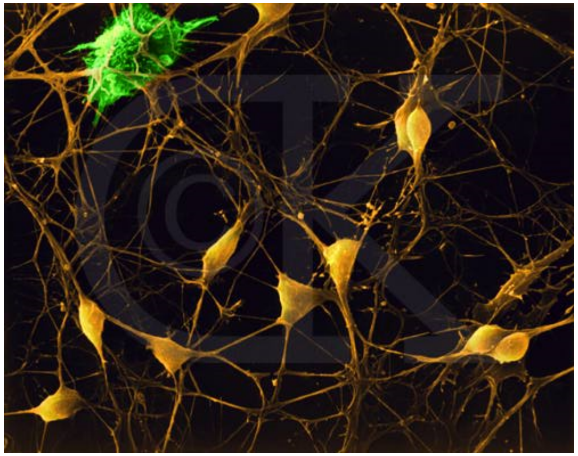
- History traces back to the 40’s but became popular in the 80’s with work by Hopfield, Rumelhart, Hinton, and Mclelland
- Peaked in the 90’s. Today:
  - Hundreds of variants
  - Less a model of the actual brain than a useful tool, but still some debate
- Numerous applications
  - Handwriting, face, speech recognition
  - Vehicles that drive themselves
  - Models of reading, sentence production, dreaming
- Recent major resurgence
  - NNs are computationally expensive, so only recently large scale neural networks became computationally feasible

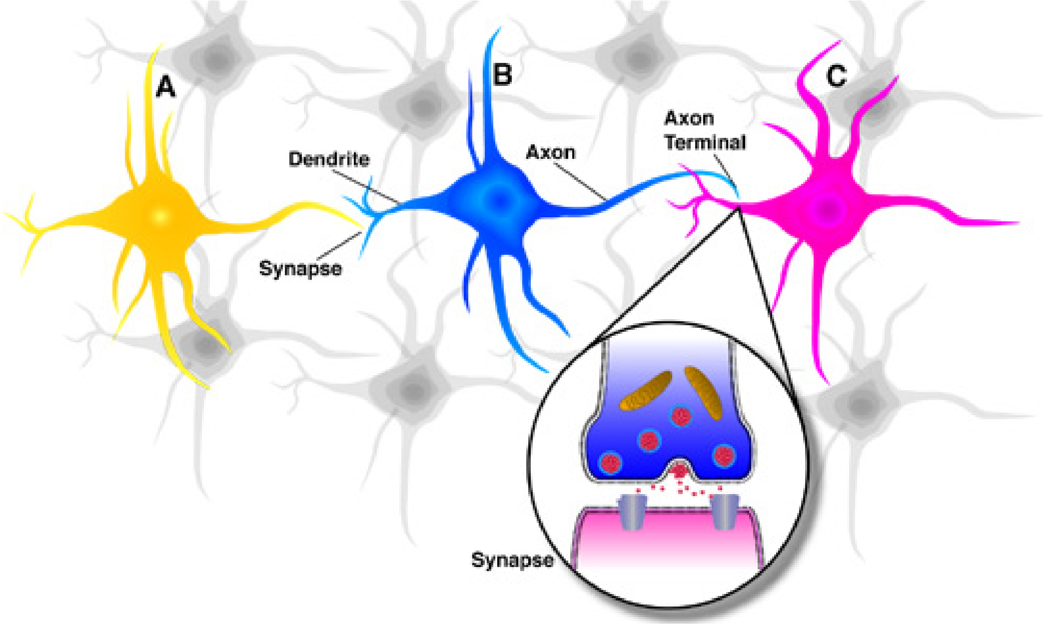
Reasons to study neural computation

- To understand how the brain actually works.
  - It's very big and very complicated and made of stuff that dies when you poke it around. So we need to use computer simulations.
- To understand a style of parallel computation inspired by neurons and their adaptive connections.
  - Very different style from sequential computation.
    - should be good for things that brains are good at (e.g. vision)
    - should be bad for things that brains are bad at (e.g. 23 x 71)
- To solve practical problems by using novel learning algorithms inspired by the brain (this lesson)
  - Learning algorithms can be very useful even if they are not how the brain actually works.

Neurons in the Brain

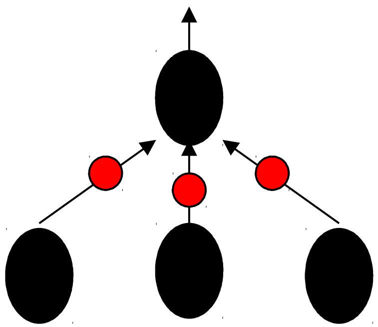
- Although heterogeneous, at a low level the brain is composed of neurons
  - A neuron receives input from other neurons (generally thousands) from its synapses (can be adapted ~ learning)
  - Inputs are approximately summed
  - When the input exceeds a threshold the neuron sends an electrical spike that travels from the body, down the axon, to the next neuron(s)





How the brain works on one slide!

- Each neuron receives inputs from other neurons
  - A few neurons also connect to receptors.
  - Cortical neurons use spikes to communicate.
- The effect of each input line on the neuron is controlled by a synaptic weight
  - The weights can be positive or negative.
- The synaptic weights adapt so that the whole network learns to perform useful computations
  - Recognizing objects, understanding language, making plans, controlling the body.
- You have about 10^{11} neurons each with about 10^{4} weights.
  - A huge number of weights can affect the computation in a very short time. Much better bandwidth than a workstation.



Idealized Neurons

- To model things we have to idealize them (e.g. atoms)
  - Idealization removes complicated details that are not essential for understanding the main principles.
  - It allows us to apply mathematics and to make analogies to other, familiar systems.
  - Once we understand the basic principles, its easy to add complexity to make the model more faithful.
- It is often worth understanding models that are known to be wrong (but we must not forget that they are wrong!)
  - E.g. neurons that communicate real values rather than discrete spikes of activity.

Linear Neurons

- These are simple but computationally limited
  - If we can make them learn we may get insight into more complicated neurons.

bias

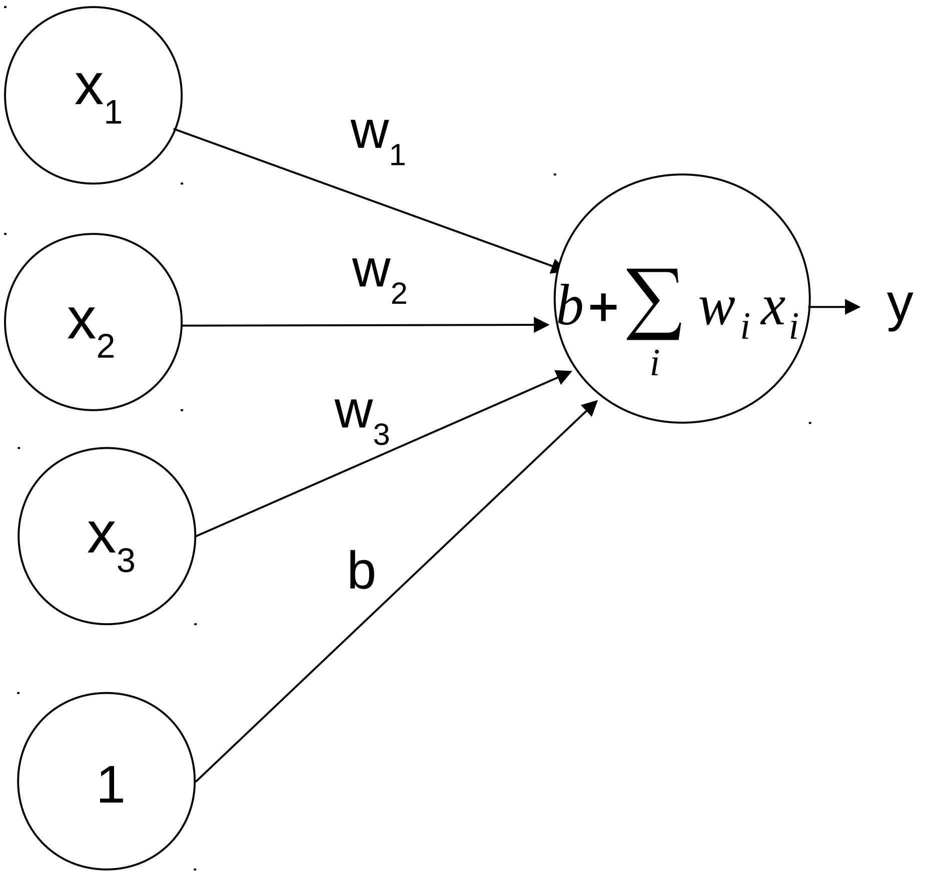
i^{th} input dimension

$$y = b + \sum_i x_i w_i$$

output

index over input connections

weight on i^{th} dimension

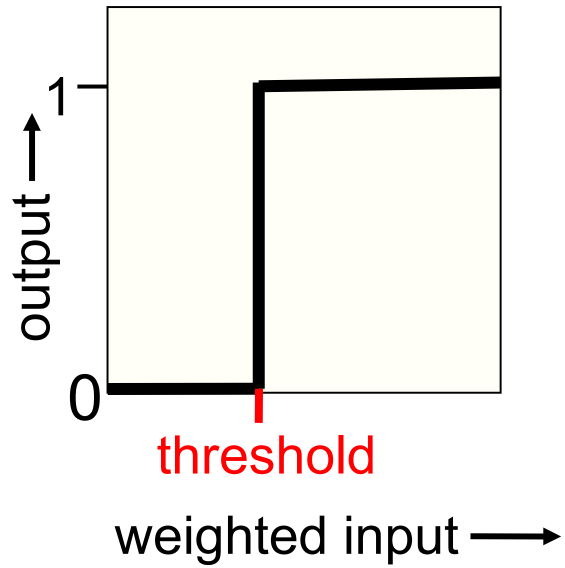


Binary threshold neurons

- There are two equivalent ways to write the equations for a binary threshold neuron:

$$z = \sum_i x_i w_i \qquad\qquad \boxed{\theta = -b} \qquad\qquad z = b + \sum_i x_i w_i$$

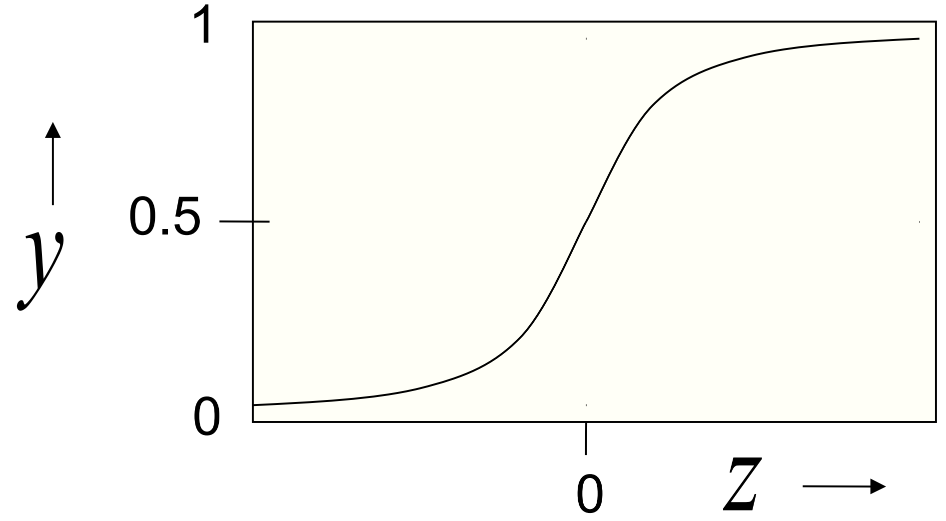
$$y = \begin{cases} 1 \text{ if } & z > \theta \\ 0 \text{ otherwise} & \end{cases} \qquad\qquad y = \begin{cases} 1 \text{ if } & z > 0 \\ 0 \text{ otherwise} & \end{cases}$$



Sigmoid neurons

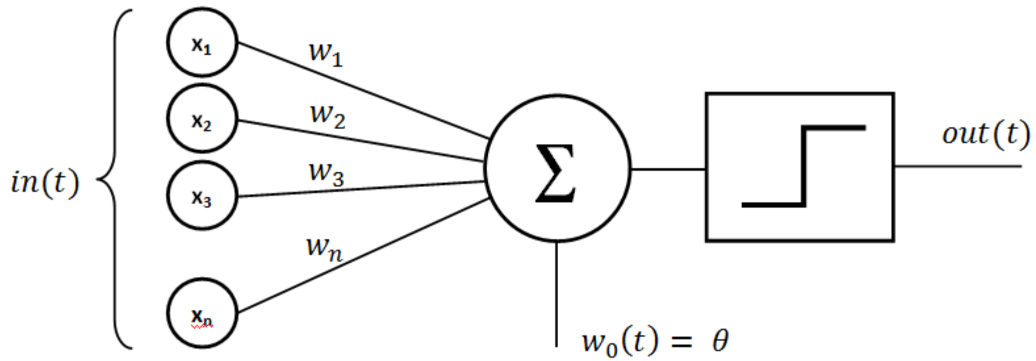
- These give a real-valued output that is a smooth and bounded function of their total input.
  - Typically they use the logistic function
  - They have nice derivatives which make learning easy (we will learn backpropagation).

$$z = b + \sum_i x_i w_i \qquad\qquad y = \frac{1}{1+e^{-z}}$$

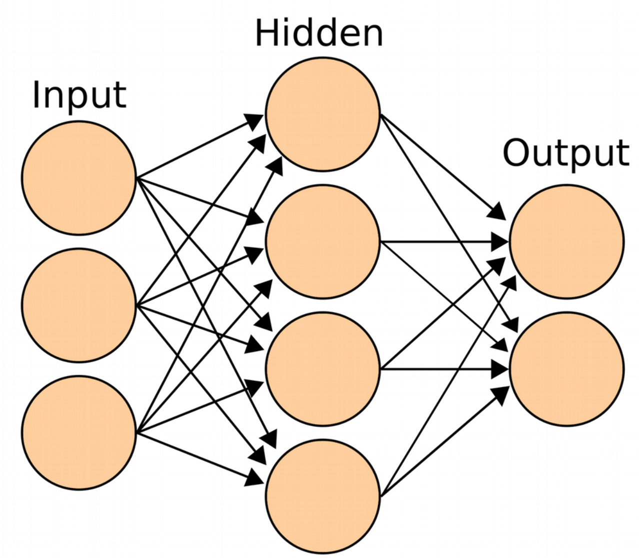


Simple Neural Networks

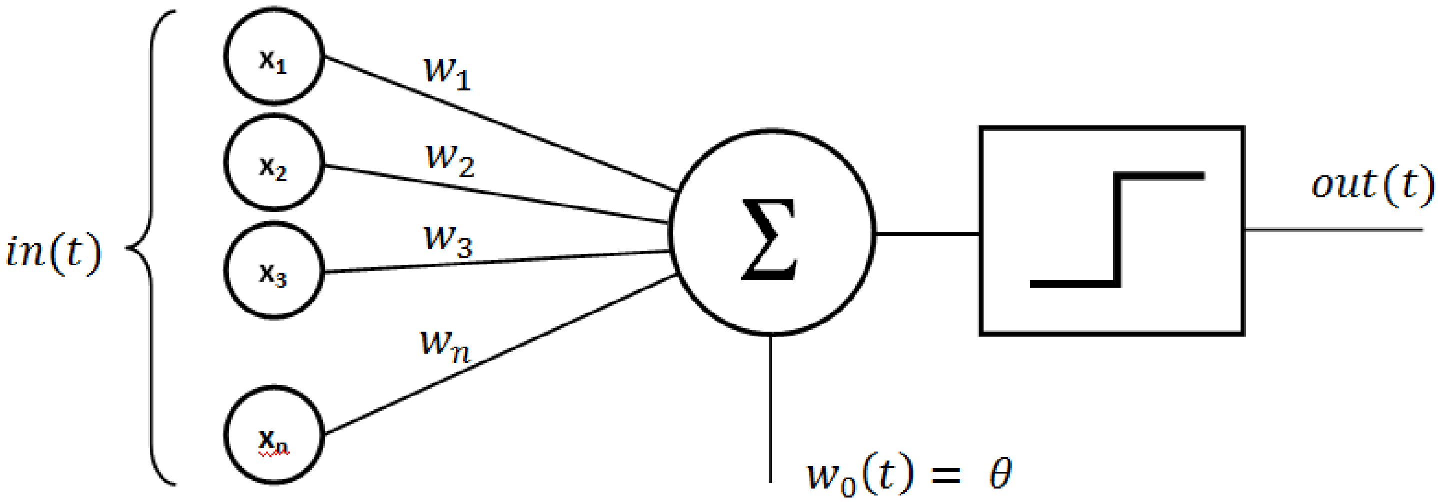
- Perceptron



- Multi-layer Perceptron

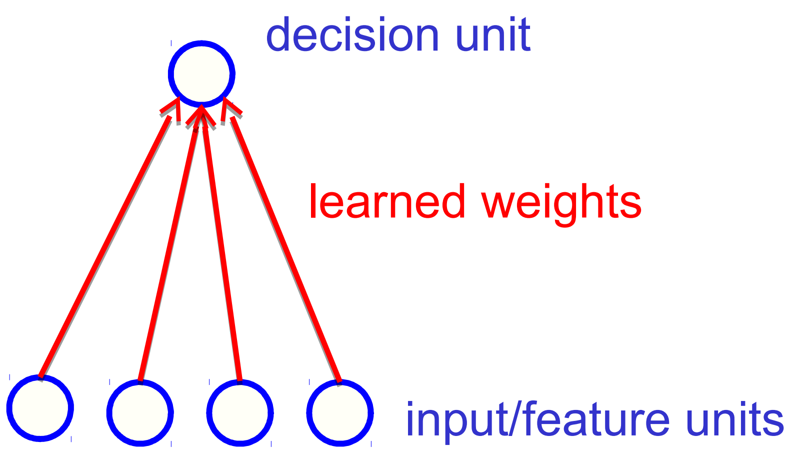


Perceptron



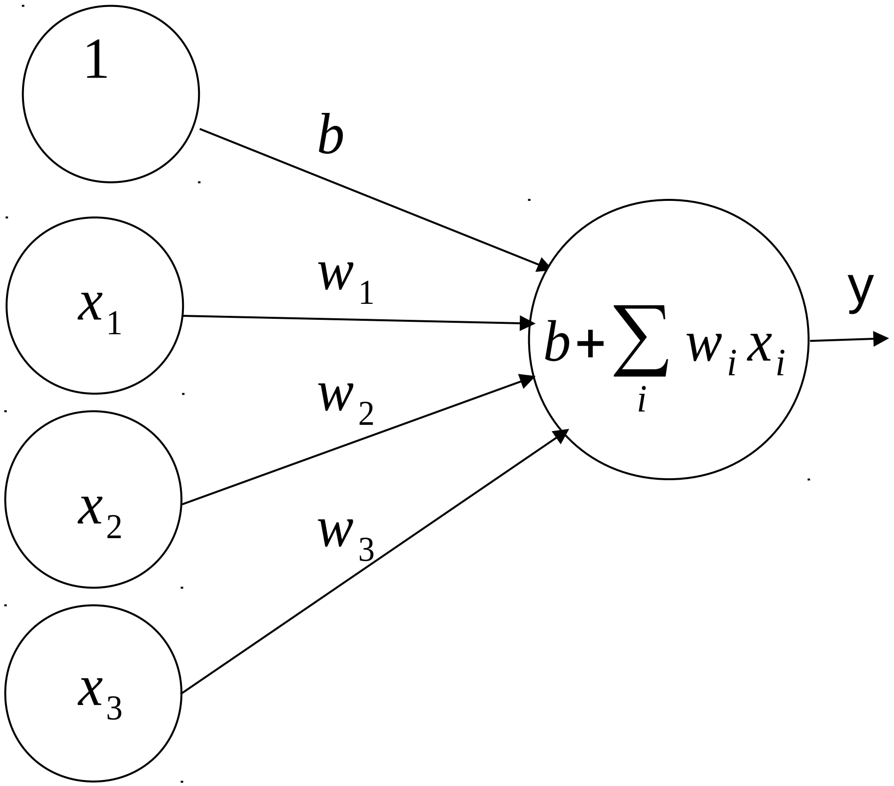
The standard Perceptron architecture

- Learn how to weight each of the input/feature variables to get a single scalar quantity.
- If this quantity is above some threshold, decide that the input vector is a positive example of the target class.
- The simplest neural network algorithm for supervised learning of **binary classification**



Perceptrons

- Essentially a linear discriminant composed of nodes, weights
- It is based on a linear neuron and a binary threshold neuron.



$$y = \begin{cases} 1 : \left(b + \sum_i w_i x_i\right) > 0 \\ 0 : otherwise \end{cases}$$

E.g.: Input x=[x_1, x_2, x_3] is a 3-dimensional vector

The weights are w=[b, w_1, w_2, w_3] is a 4-dimensional vector

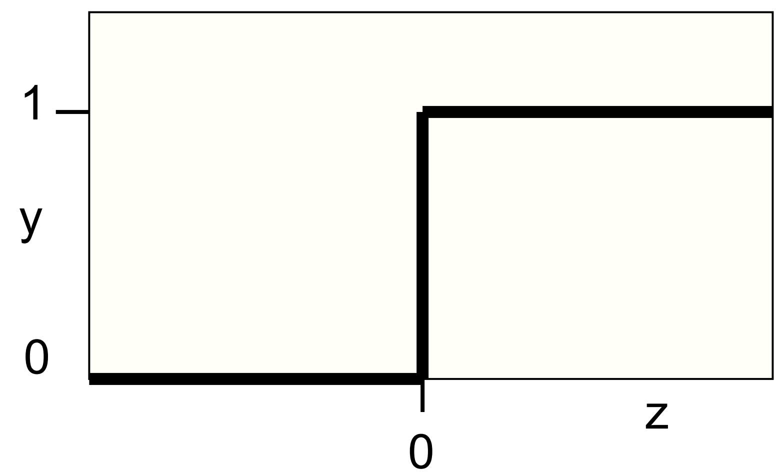
The bias is b

Perceptrons: Binary threshold neurons

- McCulloch-Pitts (1943)
  - First compute a weighted sum of the inputs from other neurons (plus a bias).
  - Then output a 1 if the weighted sum exceeds zero.

$$z = b + \sum_i x_i w_i$$

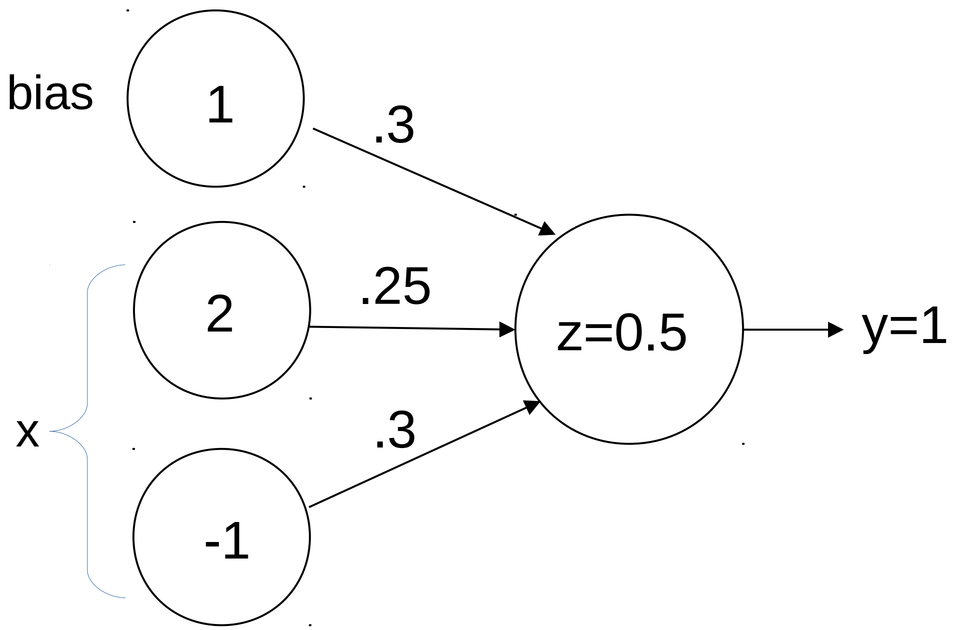
$$y = \begin{cases} 1 \text{ if } & z > 0 \\ 0 \text{ otherwise} & \end{cases}$$

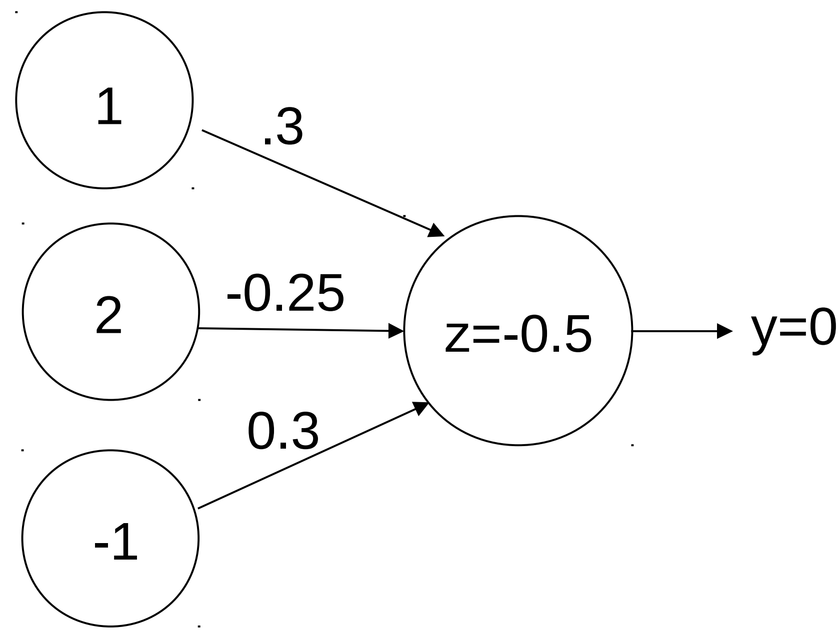


Perceptron: Example

Input x=[2,-1]   weights w=[0.3,0.25,0.3]

weights w=[0.3,-0.25,0.3]





$z = 1(0.3) + 2(0.25) - 1(0.3) = 0.5$

$z>0 \longrightarrow y=1$

$Z = 1(0.3) - 2(0.25) - 1(0.3) = -0.5$

$z<0 \longrightarrow y=0$

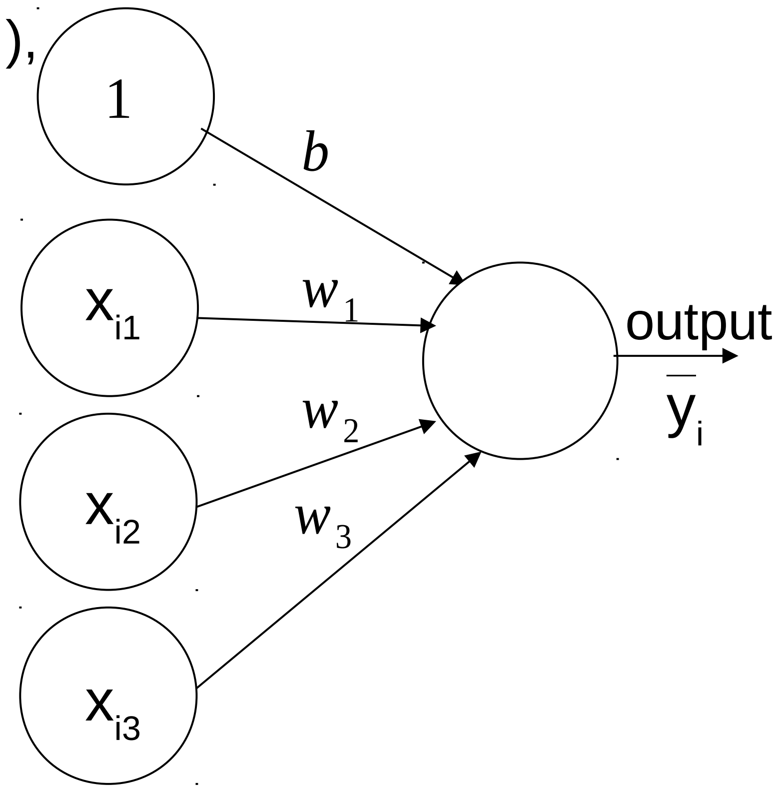
The Perceptron: Learning

- Given a dataset with N labeled data (x_1,y_1), (x_2,y_2), (x_3,y_3),….(x_N,y_N)
- Learning consists of modifying the weights.

- We initialise weights to be random.

  e.g. randomize $w=[b, w_1, w_2, w_3]$
- Training
  - Loop:
    - For each training item ***i***:
      1. Present x_i as input to the perceptron
      2. Compute the output $\bar{y}_i$ (predicted)
      3. Compute the **error** (mismatch between predicted and correct output):

         $$e_i = y_i - \bar{y}_i$$
      4. Adjust the weights according to the error:



$$\bar{y}_i = \begin{cases} 1 : (b + w_1 x_{i1} + w_2 x_{i2} + w_3 x_{i3}) > 0 \\ 0 : otherwise \end{cases}$$

$$w = w + \alpha e_i \begin{bmatrix} 1 \\ x_i \end{bmatrix}$$

α is a learning rate (step-size)

e.g.

$$\begin{bmatrix} 1 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} 1 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix} + \alpha e_i \begin{bmatrix} 1 \\ x_{i1} \\ x_{i2} \\ x_{i3} \end{bmatrix}$$

The Perceptron: Learning

- Given a dataset with N labeled data (x_1,y_1), (x_2,y_2), (x_3,y_3),….(x_N,y_N)
- Learning consists of modifying the weights.

- We initialise weights to be random.

  $w=[b, w_1, w_2, w_3]$

- Training
  - Loop:
    - For each training item ***i***:

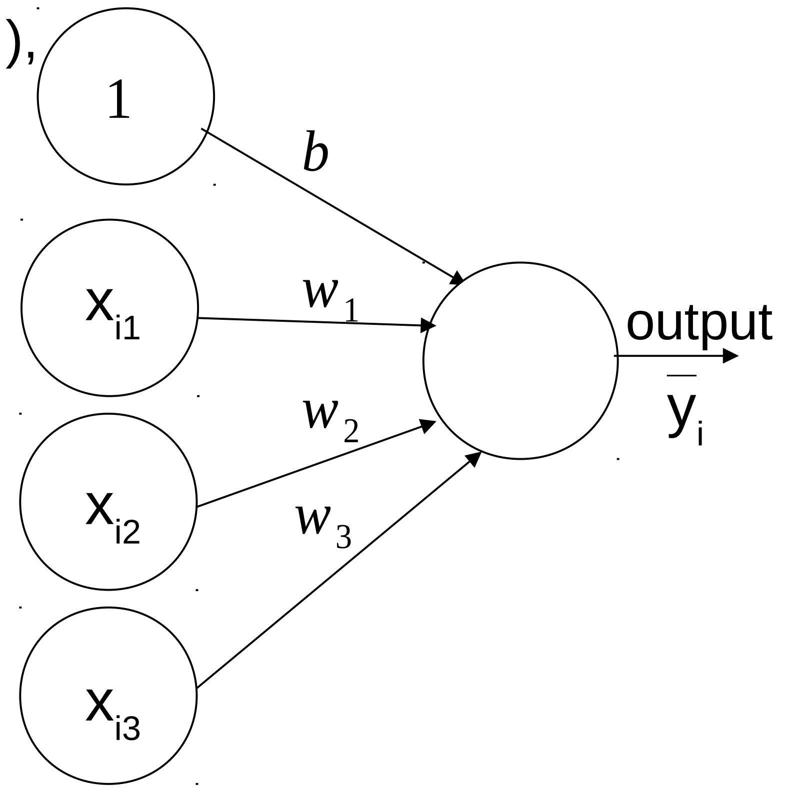
      1. Present x_i as input to the perceptron
      2. Compute the output $\bar{y}_i$ (predicted)
      3. Compute the **error** (mismatch between predicted and correct output):

         $$e_i = y_i - \bar{y}_i$$

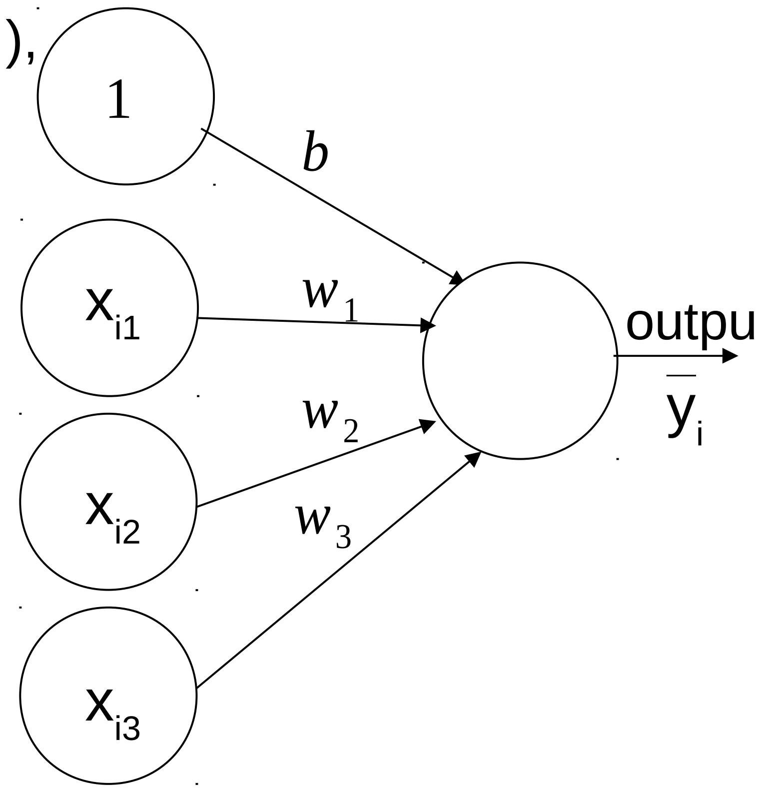
      4. Adjust the weights according to the error:

         $$w = w + \alpha e_i \begin{bmatrix} 1 \\ x_i \end{bmatrix}$$

    - Terminate when certain accuracy reached.



The Perceptron: Learning



- Given a dataset with N labeled data (x_1,y_1), (x_2,y_2), (x_3,y_3),….(x_N,y_N)
- Learning consists of modifying the weights.

- We initialise weights to be random.

  $w=[b, w_1, w_2, w_3]$
- Training
  - Loop:
    - For each training item ***i***:
      1. Present x_i as input to the perceptron
      2. Compute the output $\bar{y}_i$ (predicted)
      3. Compute the **error** (mismatch between predicted and correct output):

         $$e_i = y_i - \bar{y}_i$$
      4. Adjust the weights according to the error:

         $$w = w + \alpha\, e_i \begin{bmatrix} 1 \\ x_i \end{bmatrix}$$

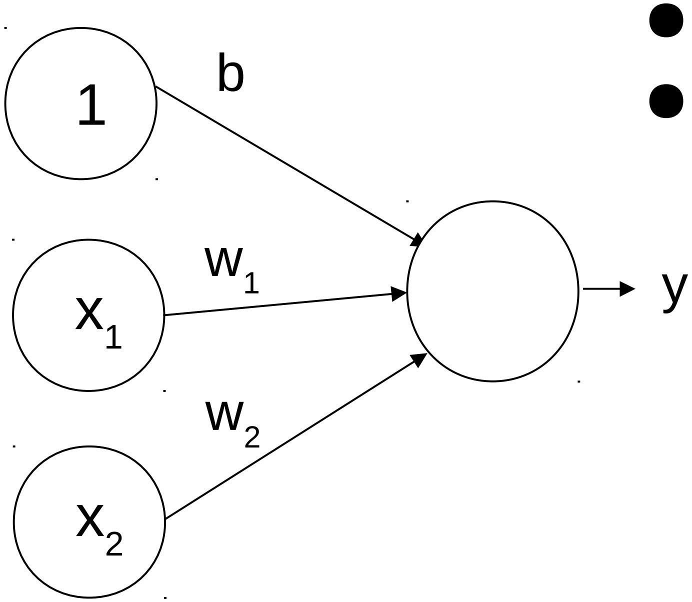
One iteration: called an epoch

(when the training process has one complete pass through the training dataset

Perceptron Example: AND function

- Two input nodes (with bias term): $x = [1, x_1, x_2]$
- No hidden layer
- One output node: y
- The weights of this network: $w=[b, w_1, w_2]$

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

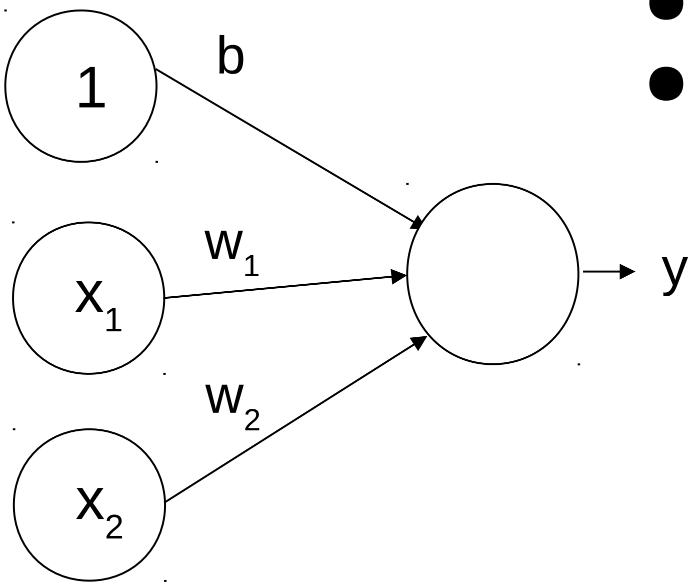


- Initialize: w=[0.1, 0.2, -0.2], set $\alpha = 0.5$
- Loop:
  - For each training instance:
    - x=[1,0,0],
      - Compute 1+0*0.2-0*0.2=1>0, predict $\bar{y}$ = 1.
      - The error is e = y- $\bar{y}$ = 0-1 = -1
      - Update w = w – 0.5*1 * [1,0,0] = [-0.4,0.2,-0.2]
    - x=[1,0,1],
      - Compute -0.4+0*0.2-1*0.2=-0.6<0, predict $\bar{y}$ = 0
      - The error is e = y- $\bar{y}$ = 0-0 = 0
      - Update w = w + 0.5*0 * [1,0,1] = [-0.4,0.2,-0.2]

Perceptron Example: AND function

- Two input nodes (with bias term): x = [1,x_1, x_2]
- No hidden layer
- One output node: y
- The weights of this network: w=[b, w_1, w_2]

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

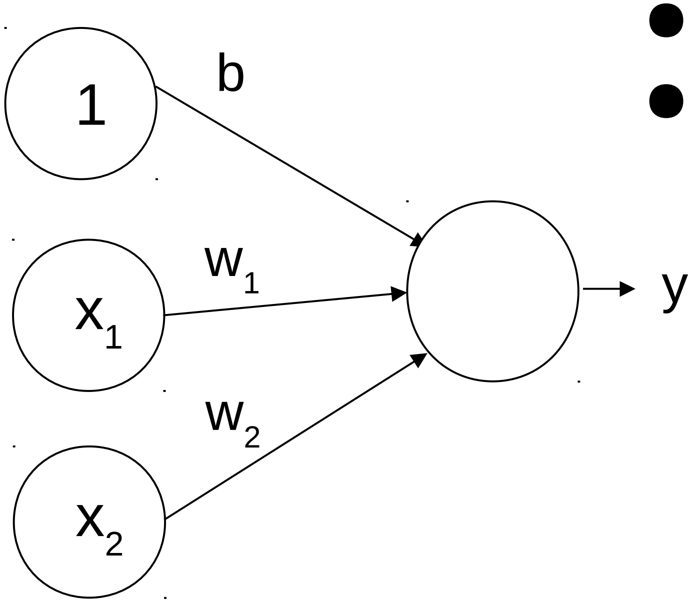


- Initialize: w=[0.1, 0.2, -0.2], set $\alpha = 0.5$
- Loop:
  - For each training instance (cont'd):
    - x=[1,1,0],
      - Compute -0.4+1*0.2-0*0.2=-0.2<0, predict $\bar{y}$=0
      - The error is e = y- $\bar{y}$ = 0-0 = 0
      - Update w = w + 0.5*0 * [1,0,0] = [-0.4,0.2,-0.2]
    - x=[1,1,1],
      - Compute -0.4+1*0.2-1*0.2=-0.4<0, predict $\bar{y}$ = 0
      - The error is e = y- $\bar{y}$ = 1-0 = 1
      - Update w = w + 0.5*1 * [1,1,1] = [0.1,0.7,0.3]

Perceptron Example: AND function

- Two input nodes (with bias term): x = [1,x_1, x_2]
- No hidden layer
- One output node: y
- The weights of this network: w=[b, w_1, w_2]

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

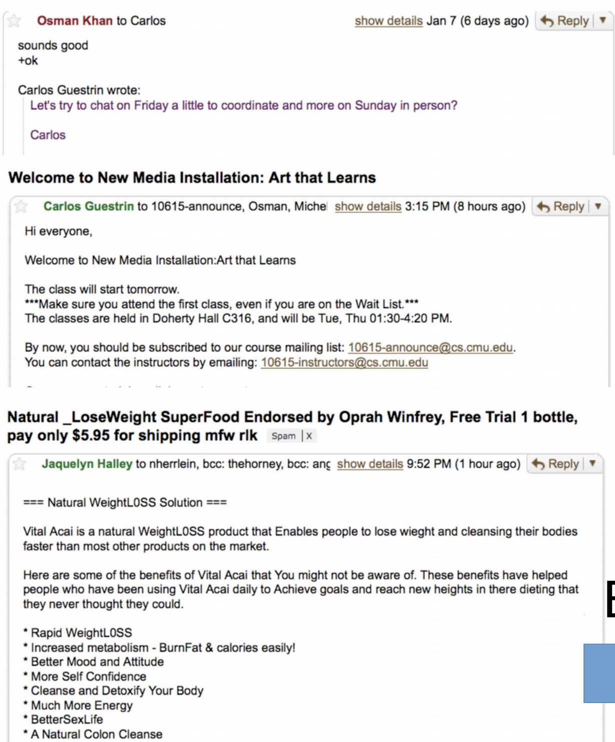


- Initialize: w=[0.1, 0.2, -0.2], set $\alpha = 0.5$
- Loop:
  - After the first epoch
    - w = [0.1,0.7,0.3]

Perceptron: Applications?

- Spam filtering: given an email → **spam** or **not spam**

data



Osman Khan to Carlos show details Jan 7 (6 days ago) Reply

sounds good
+ok

Carlos Guestrin wrote:
Let's try to chat on Friday a little to coordinate and more on Sunday in person?

Carlos

Welcome to New Media Installation: Art that Learns

Carlos Guestrin to 10615-announce, Osman, Michel show details 3:15 PM (8 hours ago) Reply

Hi everyone,

Welcome to New Media Installation:Art that Learns

The class will start tomorrow.
Make sure you attend the first class, even if you are on the Wait List.
The classes are held in Doherty Hall C316, and will be Tue, Thu 01:30-4:20 PM.

By now, you should be subscribed to our course mailing list: 10615-announce@cs.cmu.edu.
You can contact the instructors by emailing: 10615-instructors@cs.cmu.edu

Natural _LoseWeight SuperFood Endorsed by Oprah Winfrey, Free Trial 1 bottle, pay only $5.95 for shipping mfw rlk Spam | X

Jaquelyn Halley to nherrlein, bcc: thehorney, bcc: ang show details 9:52 PM (1 hour ago) Reply

=== Natural WeightL0SS Solution ===

Vital Acai is a natural WeightL0SS product that Enables people to lose wieght and cleansing their bodies faster than most other products on the market.

Here are some of the benefits of Vital Acai that You might not be aware of. These benefits have helped people who have been using Vital Acai daily to Achieve goals and reach new heights in there dieting that they never thought they could.

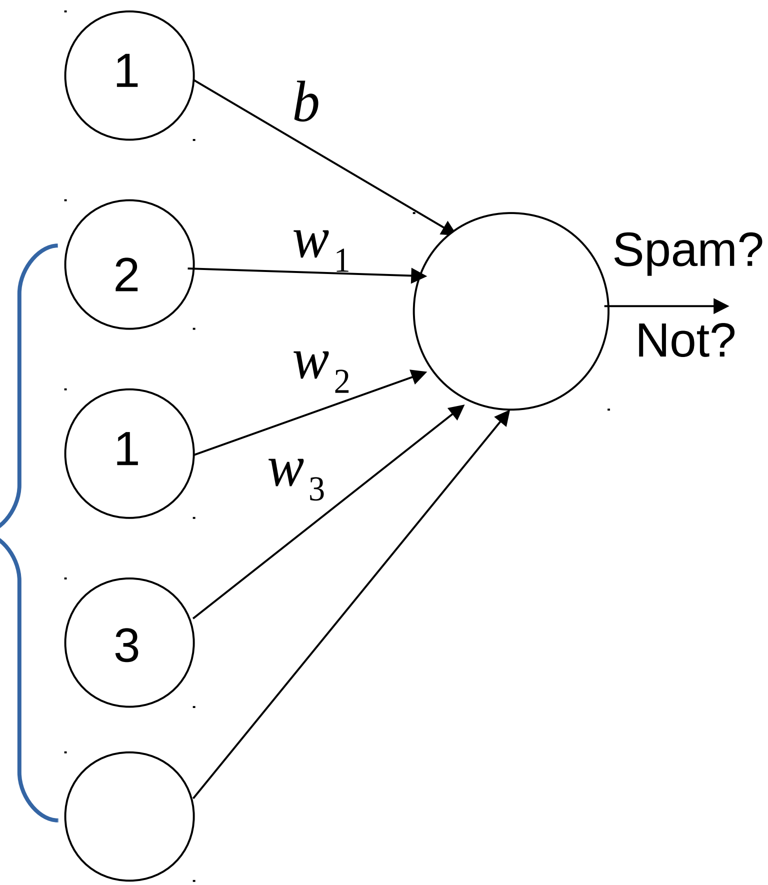
- Rapid WeightL0SS
- Increased metabolism - BurnFat & calories easily!
- Better Mood and Attitude
- More Self Confidence
- Cleanse and Detoxify Your Body
- Much More Energy
- BetterSexLife
- A Natural Colon Cleanse

prediction

Spam
vs.
Not Spam

Bag of words

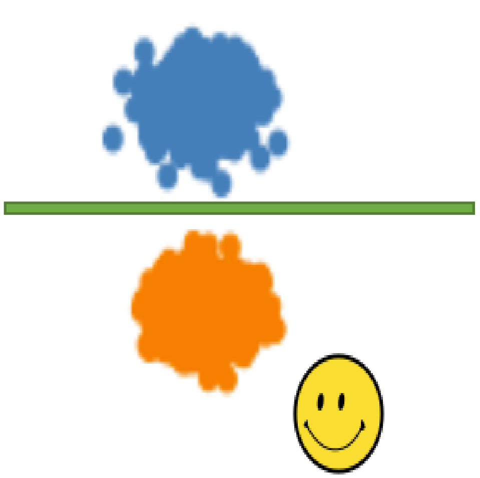
$$x = \begin{bmatrix} winner = 2 \\ prize = 1 \\ \$\$ = 3 \\ etc. \end{bmatrix}$$

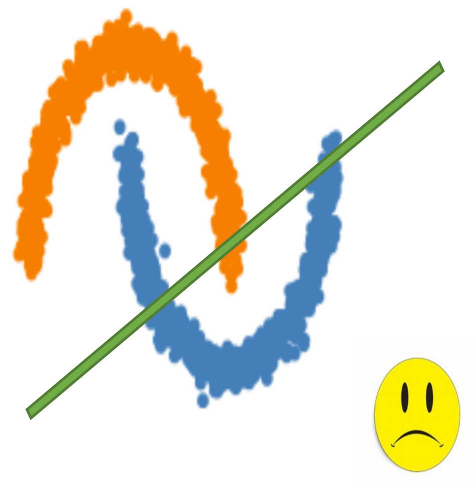


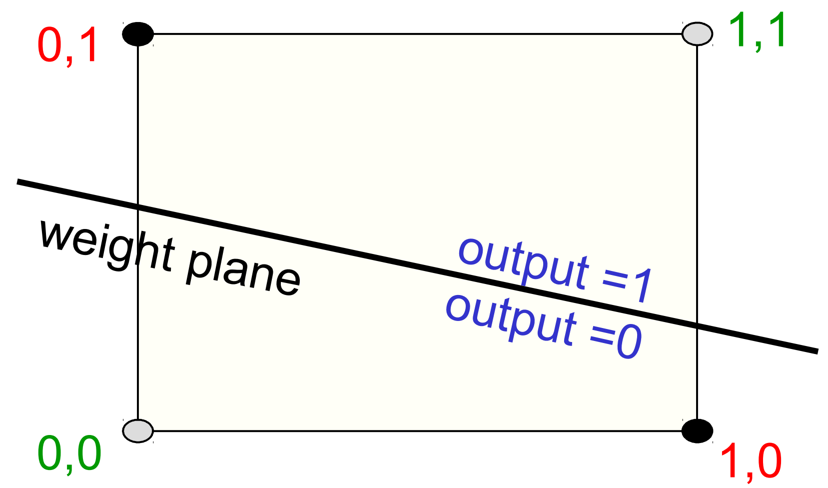
The Perceptron: Limitations

- Perceptrons are incredibly limited in their abilities.

  Can only solve linearly separable problems



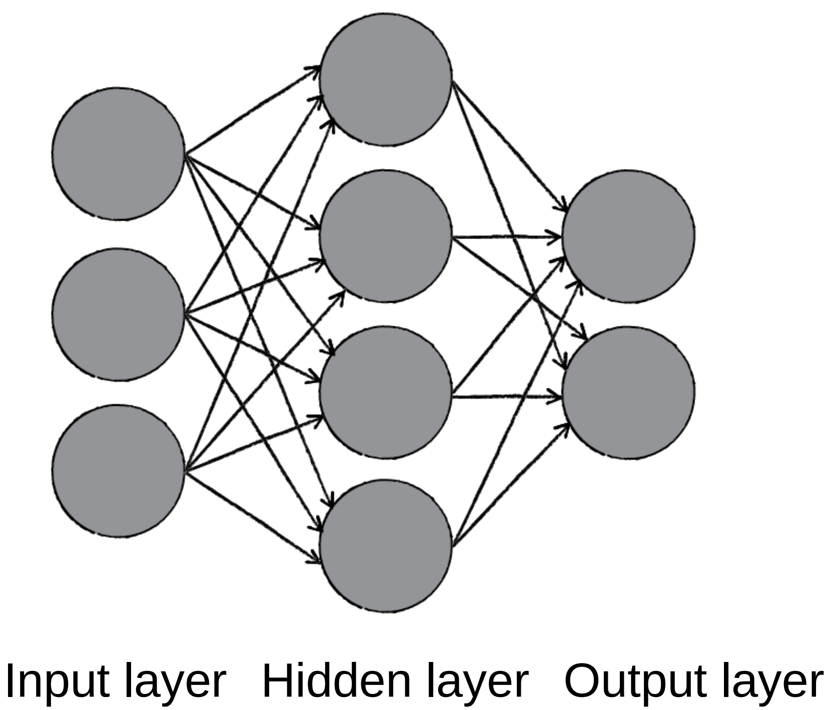




The positive and negative cases cannot be separated by a plane

The Perceptron: Limitations

We can improve things with a ***multi-layered perceptron*** (learning with hidden units)

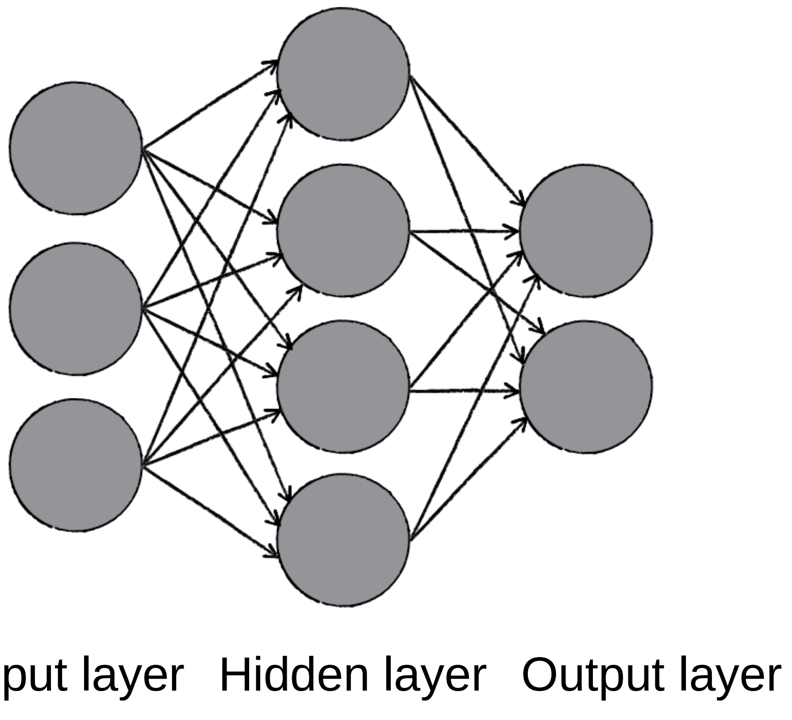


A network of many neurons. Each node is one **Linear Neuron**

- **Input Layer (layer 1):** neurons that receive the inputs
- **Hidden layers (layers 2,3, ...):** connected to neither the inputs nor the outputs of the network directly
- **Output layer (last layer):** neurons from which we read the results.

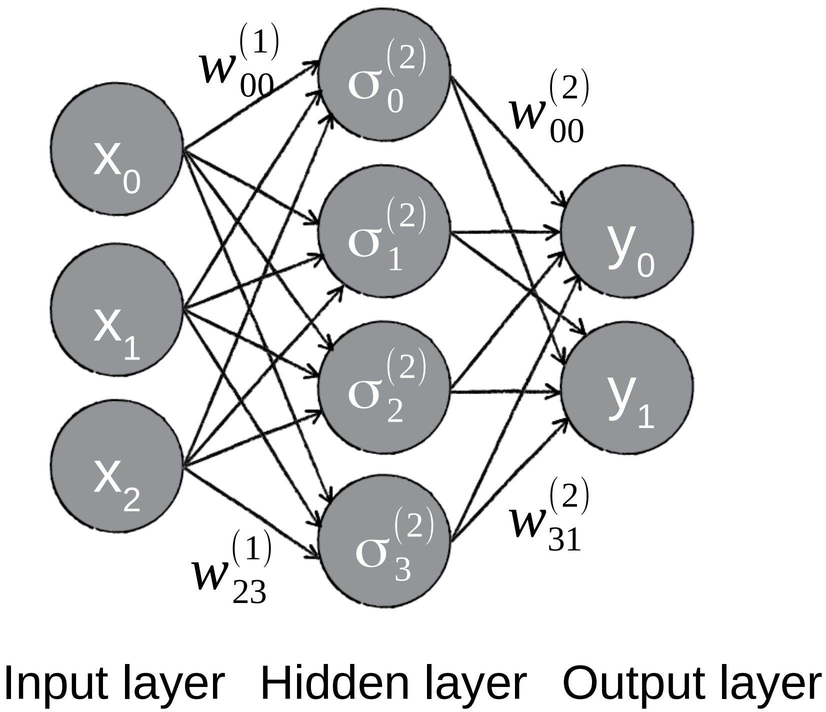
Multi-layered perceptron

We can improve things with a ***multi-layered perceptron***



- However, training is now much more complicated.
- With the simple perceptron, we could easily evaluate how to change the weights according to the error.
- Now there are so many different connections, each in a different layer of the network.
- **How does one know how much each neuron or connection contributed to the overall error of the network?**

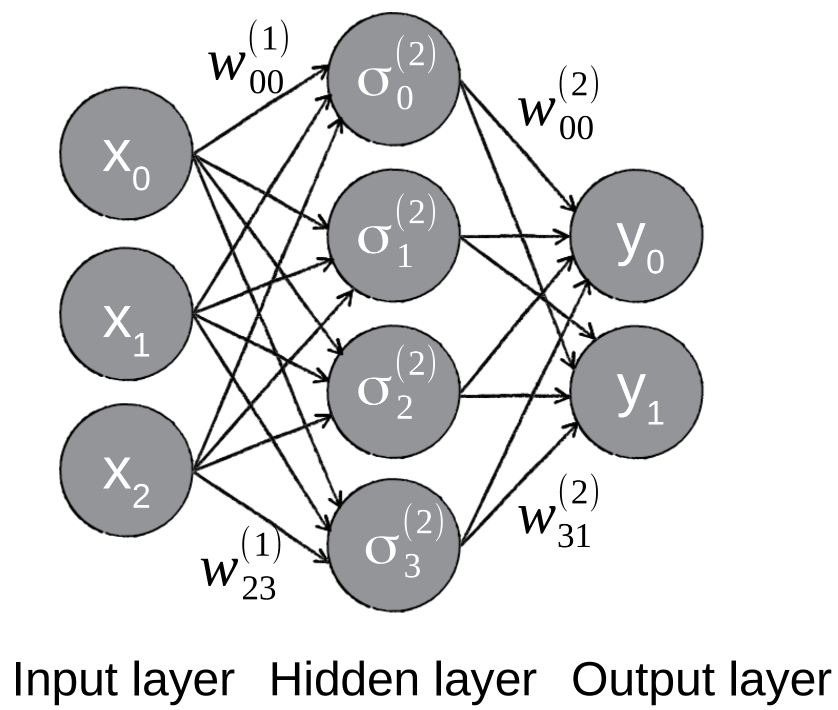
Multi-layered perceptron



- **Input variables**: x= [x_0, x_1, x_2] (x_0 is often defined as bias term: x_0 =1)
- **Parameter matrix** $w^{(j)}$ controlling the function mapping from layer j to layer j+1.
  - If layer j has m units, and j+1 has n units, then $w^{(j)}$ is mxn matrix
  - e.g. $w^{(1)}$ is 3x4 matrix, $w^{(2)}$ is 4x2 matrix
- **Activation function** $\sigma_i^{(j)}$ at node i of the layer j (By activation, we mean the value which is computed and output by that node).
  - It receives the input to the node i, and transforms it.
  - Input layer could also use activation
- **Output variables**: y= [y_0, y_1] (with activations $\sigma_0^{(3)}, \sigma_1^{(3)}$

Each node is a linear neuron with activations (e.g. binary threshold neuron, sigmoid neuron, etc.). Therefore the output after each node is the activation function applied to the linear combination of its inputs

Multi-layered perceptron



- We calculate each of the layer-2 activations based on the input values.
- We then calculate the output (prediction) (i.e. two nodes in layer 3) using exactly the same logic, except in input is not x values, but the activation values from the preceding layer.
- The activation value on each hidden unit is equal to the activation function applied to the linear combination of its inputs
- Every input/activation goes to every node in following layer

Calculations of four hidden nodes:

Hidden node 1: $g_0 = \sigma_0^{(2)}(x_0 w_{00}^{(1)} + x_1 w_{10}^{(1)} + x_2 w_{20}^{(1)})$

Hidden node 2: $g_1 = \sigma_1^{(2)}(x_0 w_{01}^{(1)} + x_1 w_{11}^{(1)} + x_2 w_{21}^{(1)})$

Hidden node 3: $g_2 = \sigma_2^{(2)}(x_0 w_{02}^{(1)} + x_1 w_{12}^{(1)} + x_2 w_{22}^{(1)})$

Hidden node 4: $g_3 = \sigma_3^{(2)}(x_0 w_{03}^{(1)} + x_1 w_{13}^{(1)} + x_2 w_{23}^{(1)})$

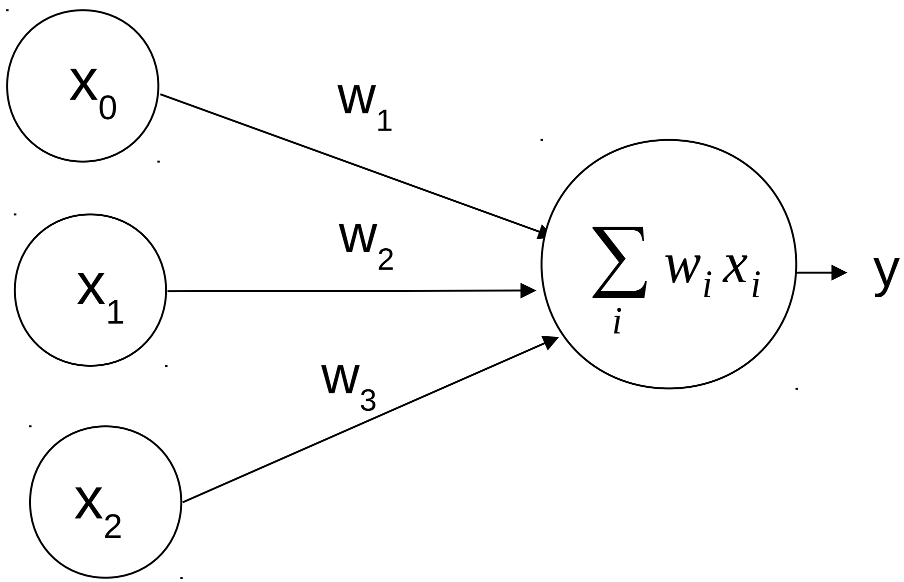
Calculations of two output nodes:

$$y_0 = \sigma_0^{(3)}(g_0 w_{00}^{(2)} + g_1 w_{10}^{(2)} + g_2 w_{20}^{(2)} + g_3 w_{30}^{(2)})$$

$$y_1 = \sigma_1^{(3)}(g_0 w_{01}^{(2)} + g_1 w_{11}^{(2)} + g_2 w_{21}^{(2)} + g_3 w_{31}^{(2)})$$

Multi-layered perceptron: Activations

- **Activation function** $\sigma_i^{(j)}$ at node i of the layer j (By activation, we mean the value which is computed and output by that node).



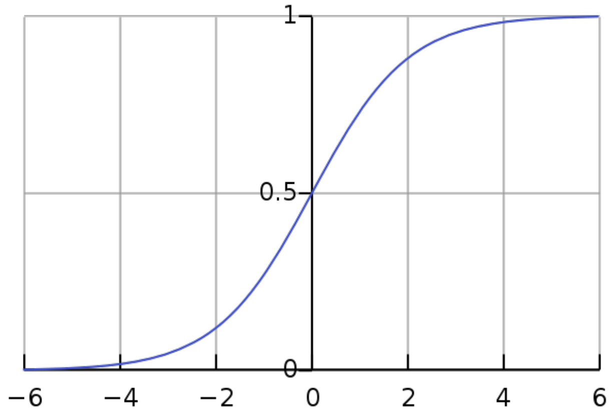
$$y = \sigma\left(\sum_i x_i w_i\right) = \begin{cases} 1 : if \sum_i x_i w_i > 0 \\ 0 : otherwise \end{cases}$$

Perceptron's activation function: $\sigma(z) = \begin{cases} 1 : if\ z > 0 \\ 0 : otherwise \end{cases}$

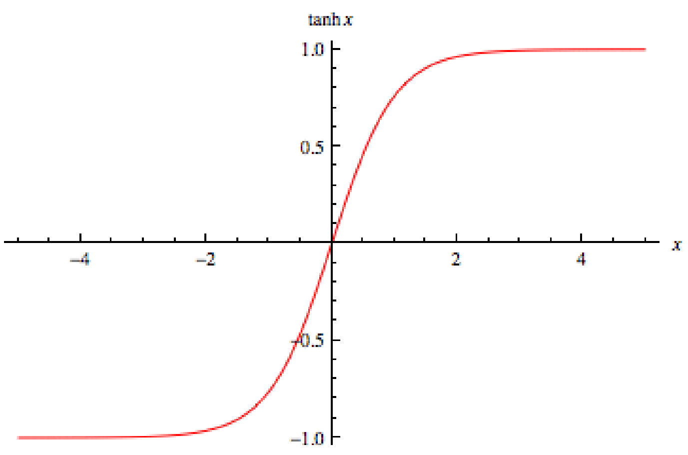
Multi-layered perceptron: Activations

- Two common activation functions used in multi-layered perceptron
  - sigmoid and tanh functions

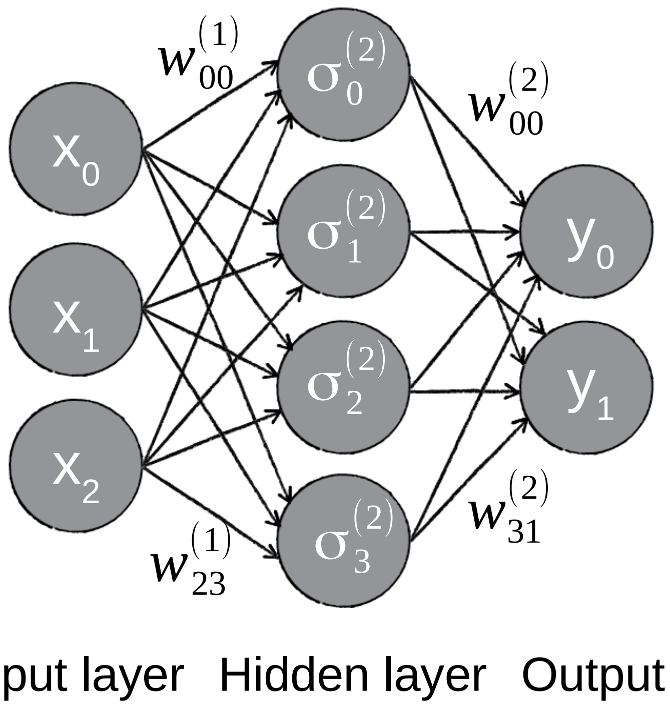
$$\sigma(x) = sigmoid(x) = \frac{1}{1+e^{-x}}$$



$$\sigma(x) = \tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$



Multi-layered perceptron (MLP)



- If the activations at last layer are sigmoid functions, this layer is just logistic regression
  - The only difference is, instead of input a feature vector, the features are just values calculated by the hidden layer
- NN is representation learning: Instead of being constrained by the original input features, a neural network can learn its own features to feed into logistic regression.
  - So we feed the hidden layers our input values, and let them learn whatever gives the best final result to feed into the final output layer.

Calculations of four hidden nodes:

$$g_0 = \sigma_0^{(2)}(x_0 w_{00}^{(1)} + x_1 w_{10}^{(1)} + x_2 w_{20}^{(1)})$$

$$g_1 = \sigma_1^{(2)}(x_0 w_{01}^{(1)} + x_1 w_{11}^{(1)} + x_2 w_{21}^{(1)})$$

$$g_2 = \sigma_2^{(2)}(x_0 w_{02}^{(1)} + x_1 w_{12}^{(1)} + x_2 w_{22}^{(1)})$$

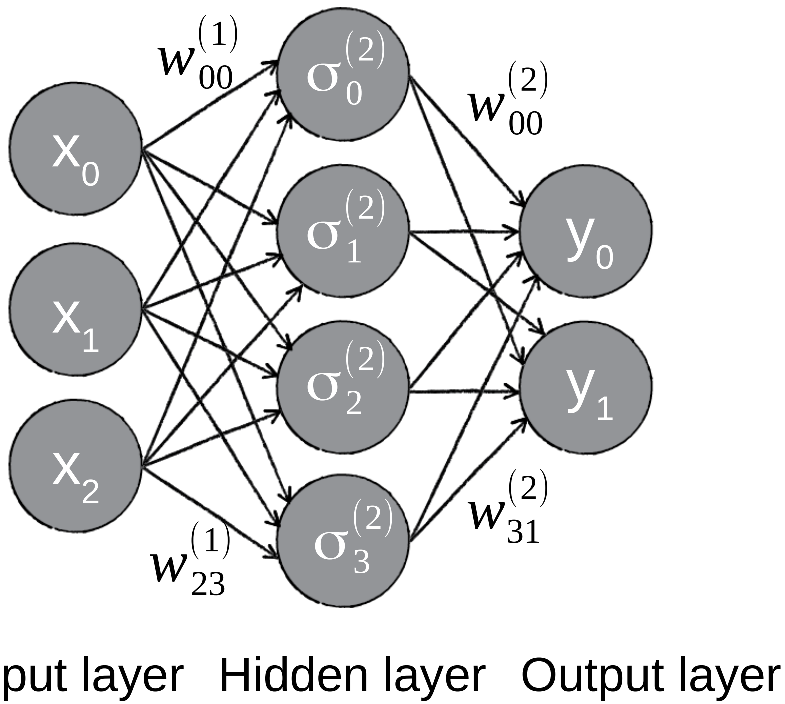
$$g_3 = \sigma_3^{(2)}(x_0 w_{03}^{(1)} + x_1 w_{13}^{(1)} + x_2 w_{23}^{(1)})$$

Calculations of two output nodes:

$$y_0 = \sigma_0^{(3)}(g_0 w_{00}^{(2)} + g_1 w_{10}^{(2)} + g_2 w_{20}^{(2)} + g_3 w_{30}^{(2)})$$

$$y_1 = \sigma_1^{(3)}(g_0 w_{01}^{(2)} + g_1 w_{11}^{(2)} + g_2 w_{21}^{(2)} + g_3 w_{31}^{(2)})$$

MLP: Vectorization



- Note that: $w^{(j)}$ is the paramter matrix mapping from layer j to layer j+1
  - e.g.: $w^{(1)}_{ij}$ is the parameter from note i (layer 1) to node j (layer 2)
  - Denote $w^{(j)}_i$ is the i^{th} column of $w^{(j)}$
    - e.g.: $w^{(1)}_0 = [w^{(1)}_{00}, w^{(1)}_{10}, w^{(1)}_{20}]$
- Assume all nodes in the same layer using the same activations function, e.g. $\sigma^{(2)}_i = \sigma^{(2)}$

input $x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$ denote $z = \begin{bmatrix} x^T w^{(1)}_0 \\ x^T w^{(1)}_1 \\ x^T w^{(1)}_2 \\ x^T w^{(1)}_3 \end{bmatrix} = w^{(1)} x$

Calculations of four hidden nodes:

$$g_0 = \sigma^{(2)}(x_0 w^{(1)}_{00} + x_1 w^{(1)}_{10} + x_2 w^{(1)}_{20})$$
$$g_1 = \sigma^{(2)}(x_0 w^{(1)}_{01} + x_1 w^{(1)}_{11} + x_2 w^{(1)}_{21})$$
$$g_2 = \sigma^{(2)}(x_0 w^{(1)}_{02} + x_1 w^{(1)}_{12} + x_2 w^{(1)}_{22})$$
$$g_3 = \sigma^{(2)}(x_0 w^{(1)}_{03} + x_1 w^{(1)}_{13} + x_2 w^{(1)}_{23})$$

vectorize

$$g_0 = \sigma^{(2)}(x^T w^{(1)}_0)$$
$$g_1 = \sigma^{(2)}(x^T w^{(1)}_1)$$
$$g_2 = \sigma^{(2)}(x^T w^{(1)}_2)$$
$$g_3 = \sigma^{(2)}(x^T w^{(1)}_3)$$

$$g = \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix} = \begin{bmatrix} \sigma^{(2)}(x^T w^{(1)}_0) \\ \sigma^{(2)}(x^T w^{(1)}_1) \\ \sigma^{(2)}(x^T w^{(1)}_2) \\ \sigma^{(2)}(x^T w^{(1)}_3) \end{bmatrix}$$

MLP: Vectorization

input $x=\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$ denote $z=\begin{bmatrix} x^T w_0^{(1)} \\ x^T w_1^{(1)} \\ x^T w_2^{(1)} \\ x^T w_3^{(1)} \end{bmatrix} = w^{(1)} x$

Calculations of four hidden nodes:

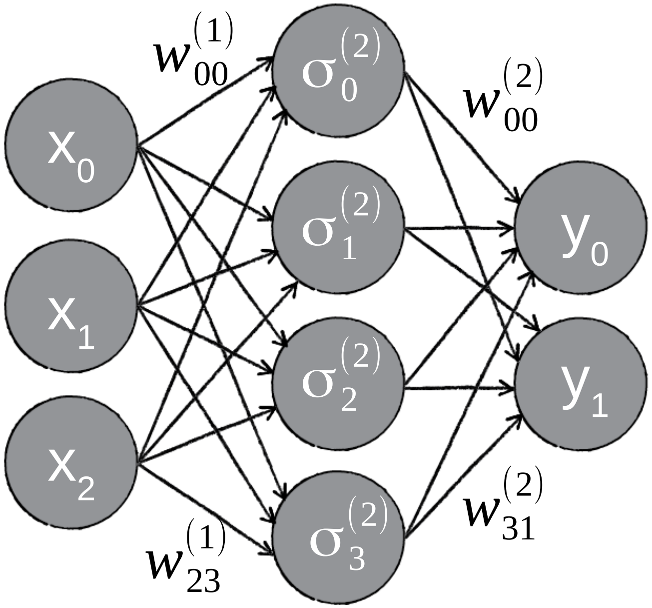
$$g_0 = \sigma^{(2)}(x_0 w_{00}^{(1)} + x_1 w_{10}^{(1)} + x_2 w_{20}^{(1)})$$
$$g_1 = \sigma^{(2)}(x_0 w_{01}^{(1)} + x_1 w_{11}^{(1)} + x_2 w_{21}^{(1)})$$
$$g_2 = \sigma^{(2)}(x_0 w_{02}^{(1)} + x_1 w_{12}^{(1)} + x_2 w_{22}^{(1)})$$
$$g_3 = \sigma^{(2)}(x_0 w_{03}^{(1)} + x_1 w_{13}^{(1)} + x_2 w_{23}^{(1)})$$

vectorize

$$g_0 = \sigma^{(2)}(x^T w_0^{(1)})$$
$$g_1 = \sigma^{(2)}(x^T w_1^{(1)})$$
$$g_2 = \sigma^{(2)}(x^T w_2^{(1)})$$
$$g_3 = \sigma^{(2)}(x^T w_3^{(1)})$$

$$g=\begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix} = \begin{bmatrix} \sigma^{(2)}(x^T w_0^{(1)}) \\ \sigma^{(2)}(x^T w_1^{(1)}) \\ \sigma^{(2)}(x^T w_2^{(1)}) \\ \sigma^{(2)}(x^T w_3^{(1)}) \end{bmatrix}$$

$$g=\sigma^{(2)}\begin{bmatrix} x^T w_0^{(1)} \\ x^T w_1^{(1)} \\ x^T w_2^{(1)} \\ x^T w_3^{(1)} \end{bmatrix} = \sigma^{(2)}(z) = \sigma^{(2)}(w^{(1)} x)$$

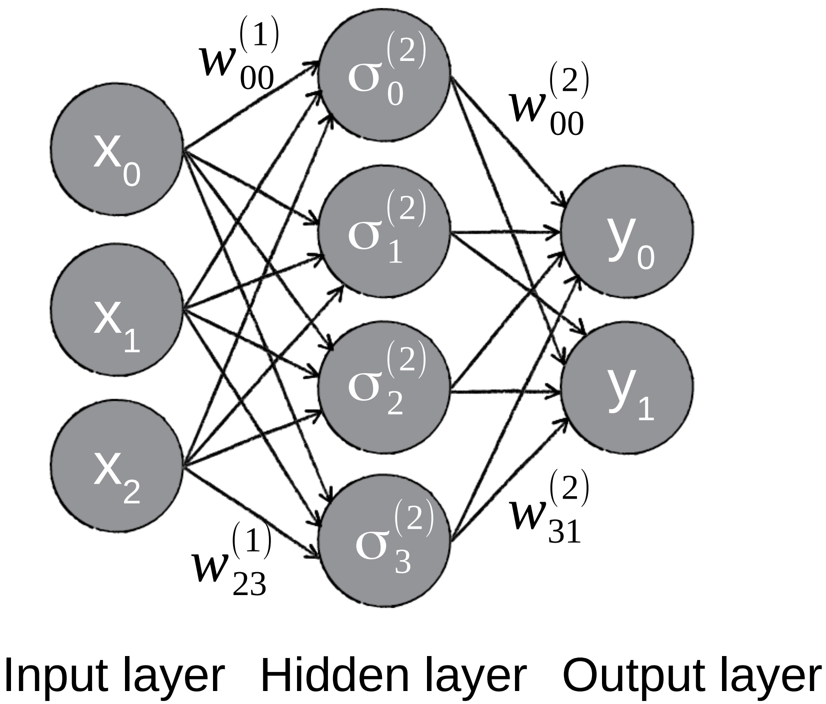


Input layer Hidden layer Output layer

MLP: Vectorization

- Note that: $w^{(j)}$ is the paramter matrix mapping from layer j to layer j+1
  - e.g.: $w^{(1)}_{ij}$ is the parameter from note i (layer 1) to node j (layer 2)

input $x=\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$ values at hidden layer $g=\begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix}$ output $y=\begin{bmatrix} y_0 \\ y_1 \end{bmatrix}$



Calculations of four hidden nodes (x is the input and $w^{(1)}$ are weights to this layer):

$$g=\sigma^{(2)}\left(w^{(1)}x\right)$$

Likewise, calculations of two output nodes (g is the input and $w^{(2)}$ are weights to this layer):

$$y=\sigma^{(3)}\left(w^{(2)}g\right)$$

A full forward pass (also called forward propagation):

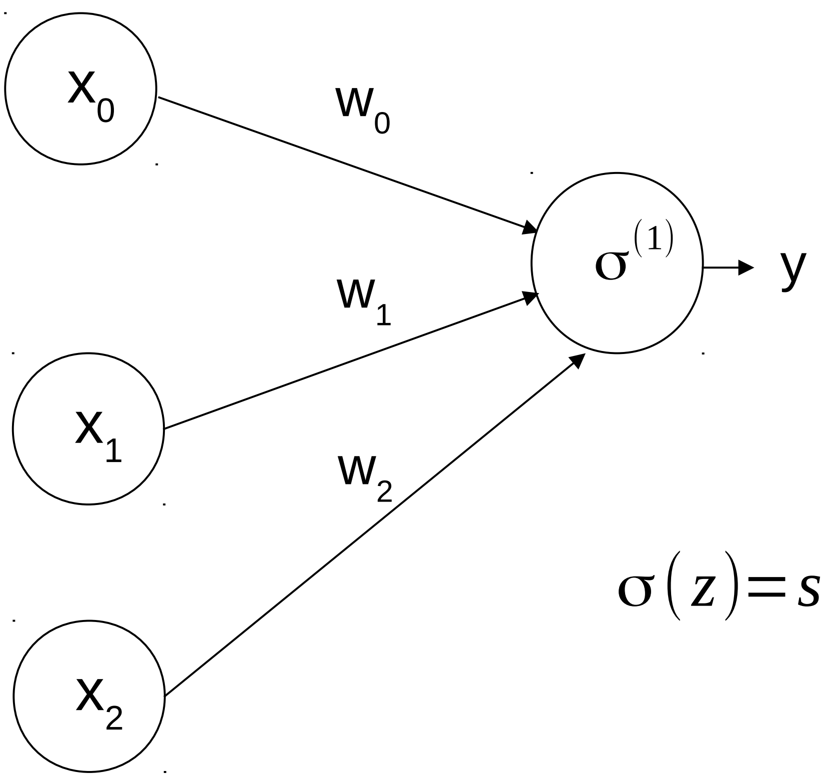
$$y=\sigma^{(3)}\left(w^{(2)}\sigma^{(2)}\left(w^{(1)}x\right)\right)$$

- Start off with activations of input unit
- Forward propagate and calculate the activation of each layer sequentially
- This is a vectorized version of this implementation

MLP Example: AND function

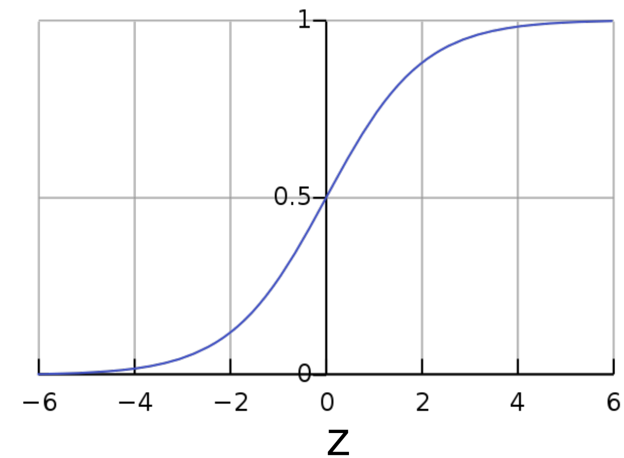
- A simple MLP network:
  - Two input nodes (with bias term at x_0): $x = [1, x_1, x_2]$
  - No hidden layer
  - One output node (**with sigmoid activations**): y
  - The weights of this network: $w^{(1)} = [w_0, w_1, w_2]$

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

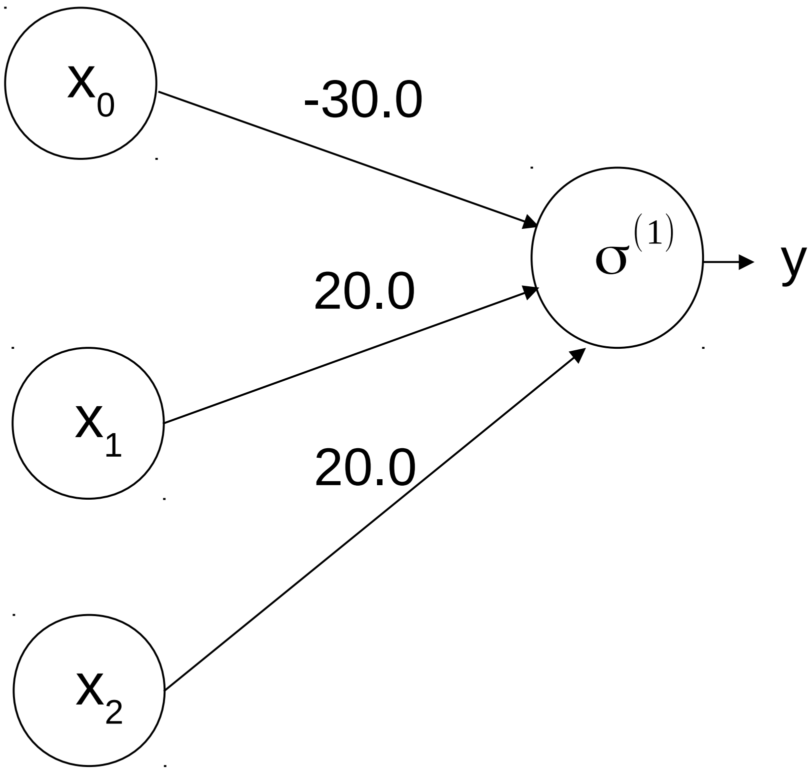


$$y = \sigma^{(1)}\left(w^{(1)} x\right) = \frac{1}{1 + e^{-w^{(1)} x}} = \frac{1}{1 + e^{-w_0 - w_1 x_1 - w_2 x_2}}$$

$$\sigma(z) = sigmoid(z) = \frac{1}{1 + e^{-z}}$$



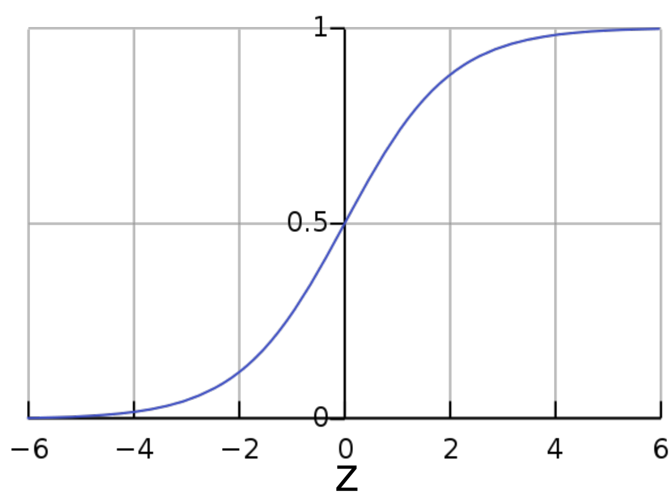
MLP Example: AND function



$$y=\sigma^{(1)}\left(w^{(1)}x\right)=\frac{1}{1+e^{-w^{(1)}x}}=\frac{1}{1+e^{-w_0-w_1x_1-w_2x_2}}$$

x_1	x_2	y (ground-truth)	predicted
0	0	0	Sigmoid(-30) = 0
0	1	0	Sigmoid(-10) = 0
1	0	0	Sigmoid(-10) = 0
1	1	1	Sigmoid(10) = 1

$$\sigma(z)=sigmoid(z)=\frac{1}{1+e^{-z}}$$



$$\bar{y}_1=\sigma^{(1)}\left(w^{(1)}[1,0,0]\right)=\frac{1}{1+e^{-30}}$$

$$\bar{y}_2=\sigma^{(1)}\left(w^{(1)}[1,0,1]\right)=\frac{1}{1+e^{-10}}$$

$$\bar{y}_3=\sigma^{(1)}\left(w^{(1)}[1,1,0]\right)=\frac{1}{1+e^{-10}}$$

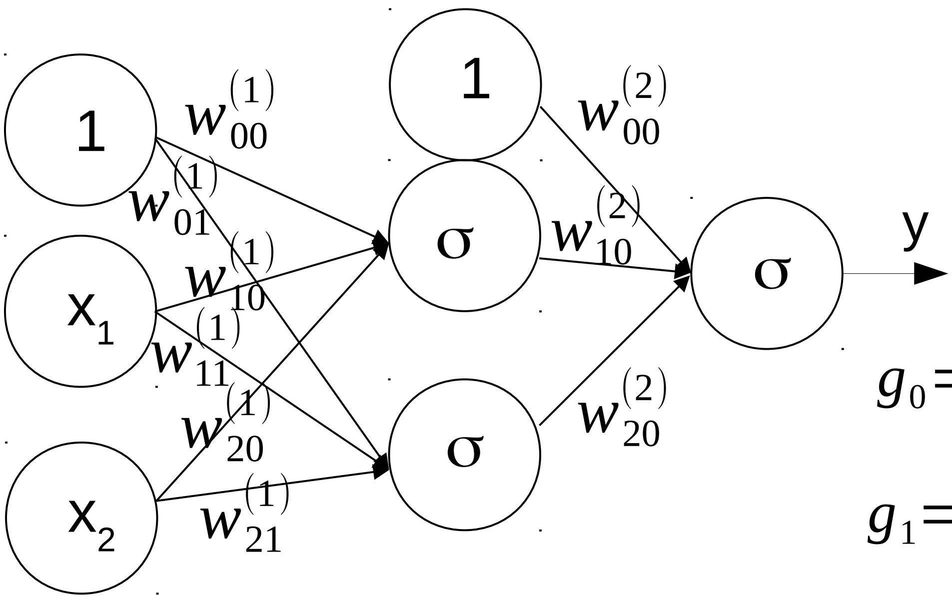
$$\bar{y}_4=\sigma^{(1)}\left(w^{(1)}[1,1,1]\right)=\frac{1}{1+e^{10}}$$

MLP Example: XOR function

- The data is non-linearly separable, so a MLP network:
  - Two input nodes (with bias term at x_0): $x = [1, x_1, x_2]$
  - One hidden layer with two nodes (**with sigmoid activations**)
  - One output node (**with sigmoid activation**): y
  - The weights of this network: $w^{(2)}=[w^{(2)}_{00}, w^{(2)}_{10}, w^{(2)}_{20}]$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

$$w^{(1)}=\begin{bmatrix} w^{(1)}_{00} & w^{(1)}_{01} \\ w^{(1)}_{10} & w^{(1)}_{11} \\ w^{(1)}_{20} & w^{(1)}_{21} \end{bmatrix}$$



Hidden node 1:

$$g_0 = sigmoid\left(w^{(1)}_{00} + w^{(1)}_{10} x_1 + w^{(1)}_{20} x_2\right) = sigmoid\left(x^T w^{(1)}_0\right)$$

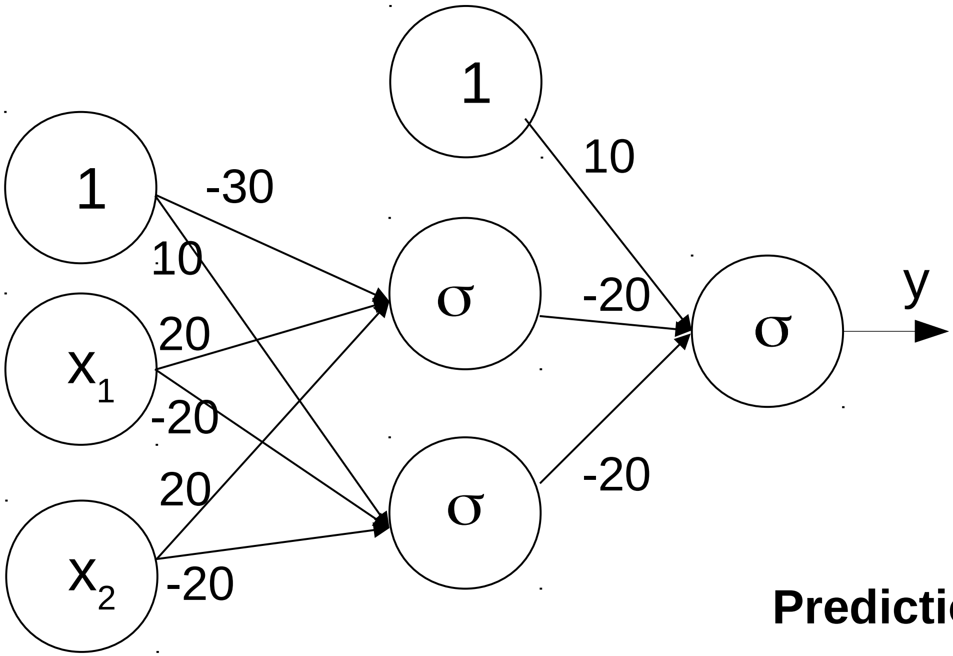
Hidden node 2:

$$g_1 = sigmoid\left(w^{(1)}_{01} + w^{(1)}_{11} x_1 + w^{(1)}_{21} x_2\right) = sigmoid\left(x^T w^{(1)}_1\right)$$

Prediction:

$$y = sigmoid\left(w^{(2)}_{00} + w^{(2)}_{10} g_0 + w^{(2)}_{20} g_1\right) = sigmoid\left([1, g_0, g_1] w^{(2)}\right)$$

MLP Example: XOR function



$$w^{(1)} = \begin{bmatrix} -30 & 10 \\ 20 & -20 \\ 20 & -20 \end{bmatrix}$$

$$w^{(2)} = [10, -20, -20]$$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Prediction for x=[1,0,0]

Hidden node 1:

$$g_0 = sigmoid\left([1,0,0] \begin{bmatrix} -30 \\ 20 \\ 20 \end{bmatrix}^T \right) = sigmoid(-30) \approx 0$$

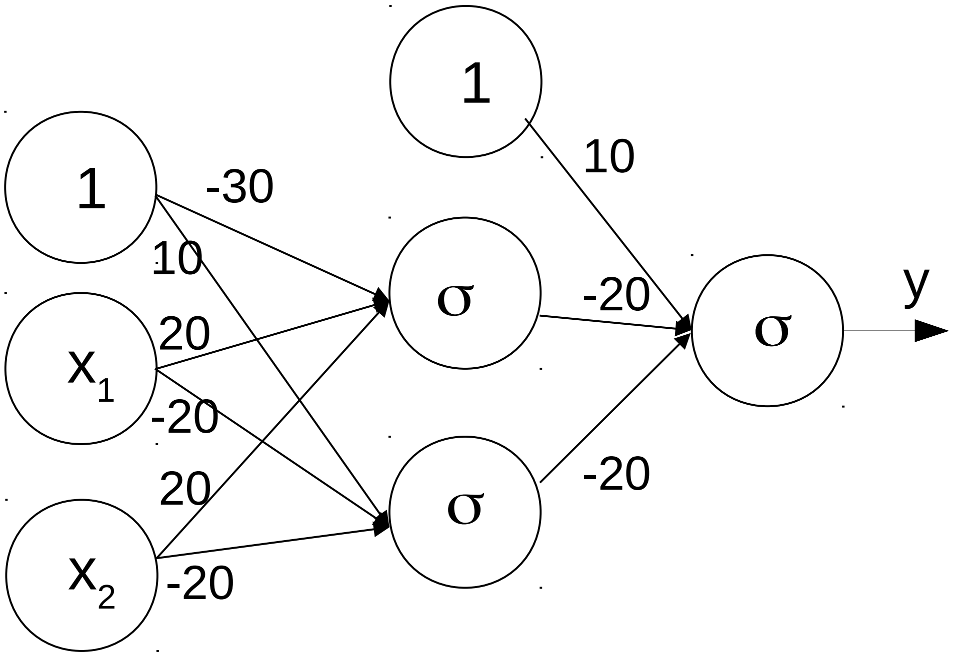
Hidden node 2:

$$g_1 = sigmoid\left([1,0,0] \begin{bmatrix} 10 \\ -20 \\ -20 \end{bmatrix}^T \right) = sigmoid(10) \approx 1$$

Prediction:

$$y = sigmoid\left(\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}^T \begin{bmatrix} 10 \\ -20 \\ -20 \end{bmatrix} \right) = sigmoid(-10) \approx 0$$

MLP Example: XOR function



$$w^{(1)} = \begin{bmatrix} -30 & 10 \\ 20 & -20 \\ 20 & -20 \end{bmatrix}$$

$$w^{(2)} = [10, -20, -20]$$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Prediction for x=[1,0,0] $\bar{y} = sigmoid\left(\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}^T \begin{bmatrix} 10 \\ -20 \\ -20 \end{bmatrix}\right) = sigmoid(-10) \approx 0$

Prediction for x=[1,0,1] $\bar{y} = sigmoid(10) \approx 1$

Prediction for x=[1,1,0] $\bar{y} = sigmoid(10) \approx 1$

Prediction for x=[1,1,1] $\bar{y} = sigmoid(-10) \approx 0$

MLP for Multi-class Classification: 1 sv. All

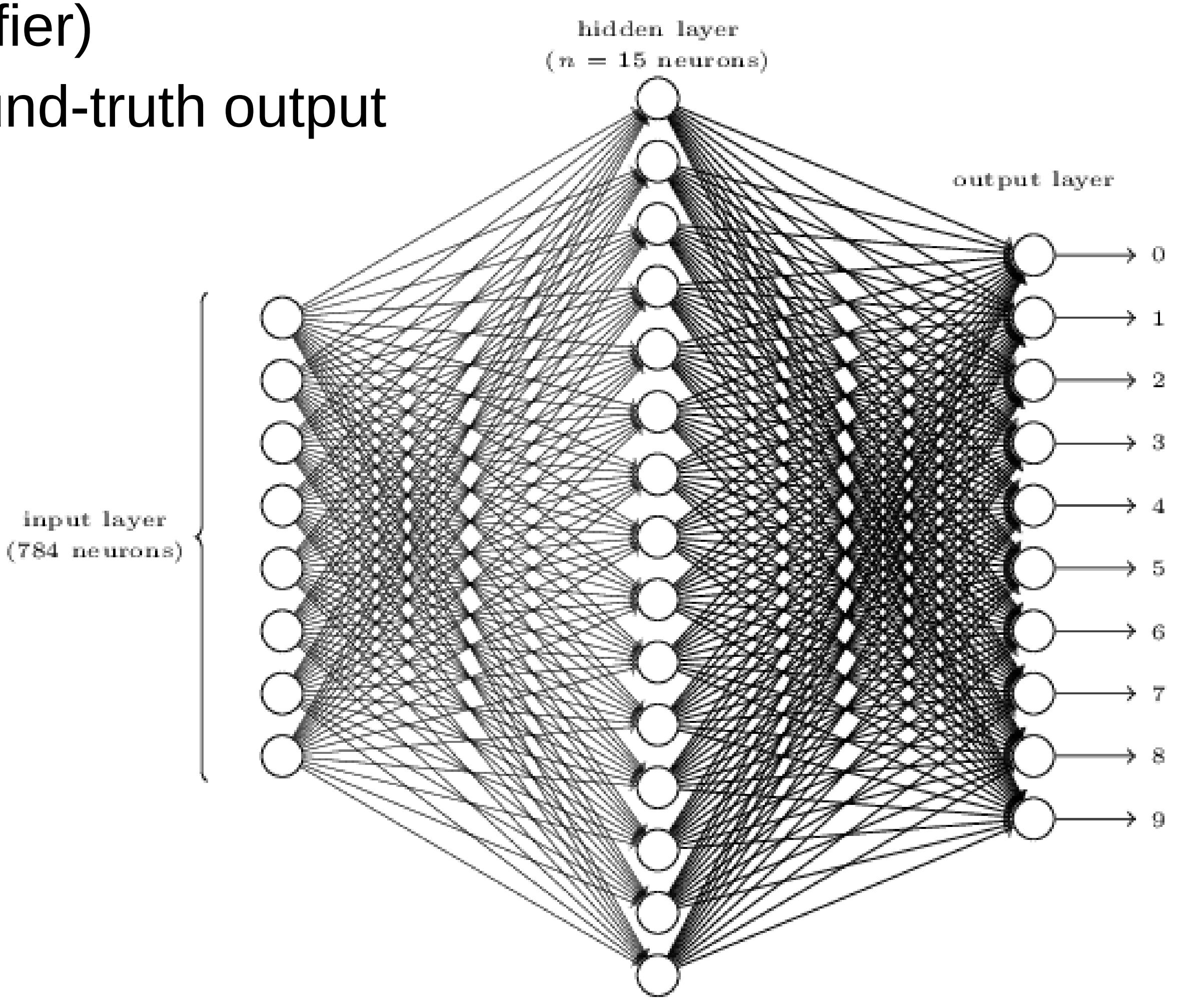
- Example: Hand-written digits dataset
  - Each image input is 28x28 = 784 dims
  - One or many hidden layers
  - Output layer: 10 neurons (i.e. each correspond to one logistic regression classifier)
  - Using one-hot vector for ground-truth output

digit **2** has output

$$y=\begin{bmatrix}0\\0\\1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}$$

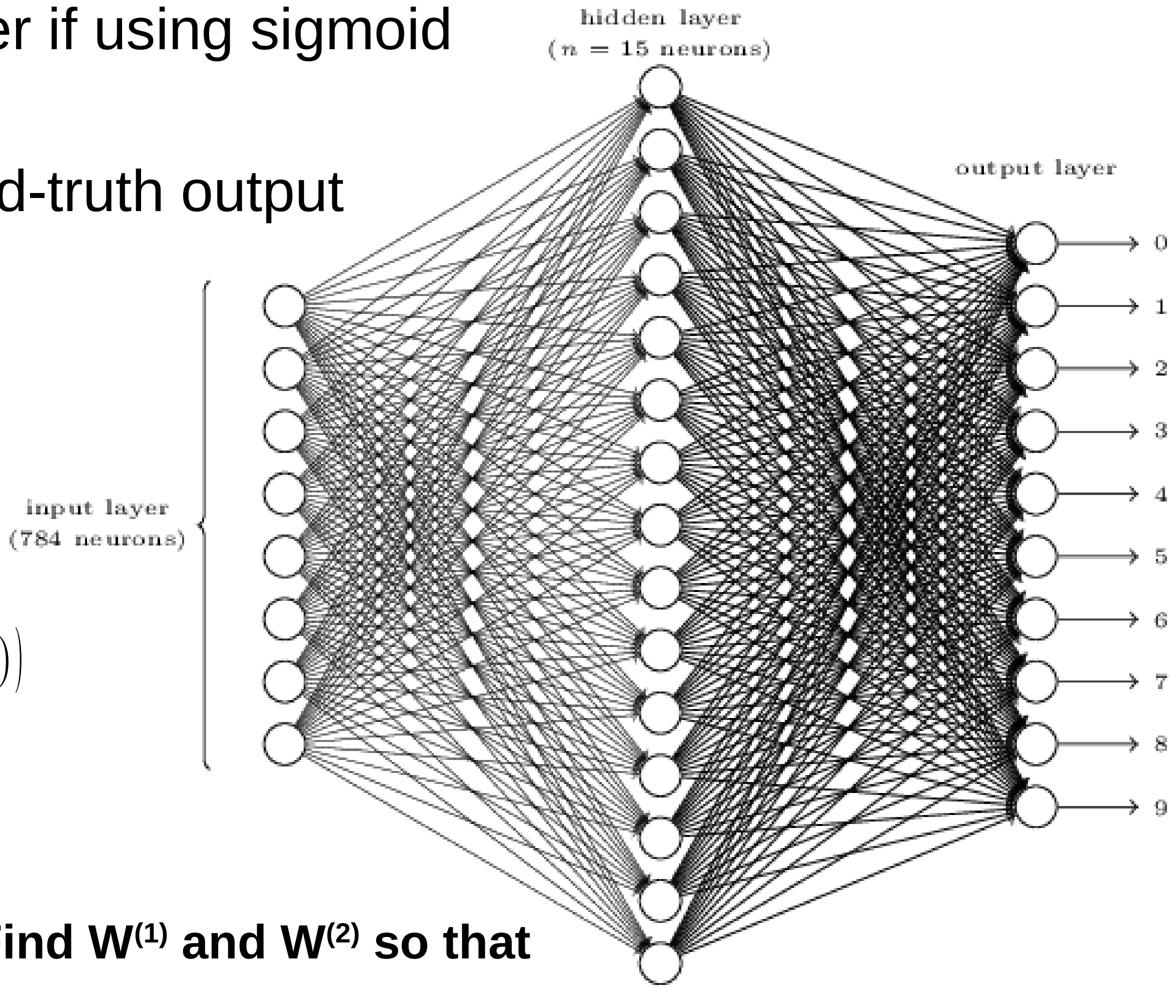
digit 8 has output

$$y=\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\1\\0\end{bmatrix}$$



MLP for Multi-class Classification: 1 sv. All

- Example: Hand-written digits dataset
  - Each image input is 28x28 = 784 dims
  - One or many hidden layers
  - Output layer: 10 neurons (i.e. each correspond to one logistic regression classifier if using sigmoid activation function)
  - Using one-hot vector for ground-truth output



digit **2** has output

$$y=\begin{bmatrix}0\\0\\1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}$$

prediction

$$\bar{y}=\begin{bmatrix}y_0\\y_1\\y_2\\y_3\\y_4\\y_5\\y_6\\y_7\\y_8\\y_9\end{bmatrix}=\sigma\left(W^{(2)}\sigma\left(W^{(1)}x\right)\right)$$

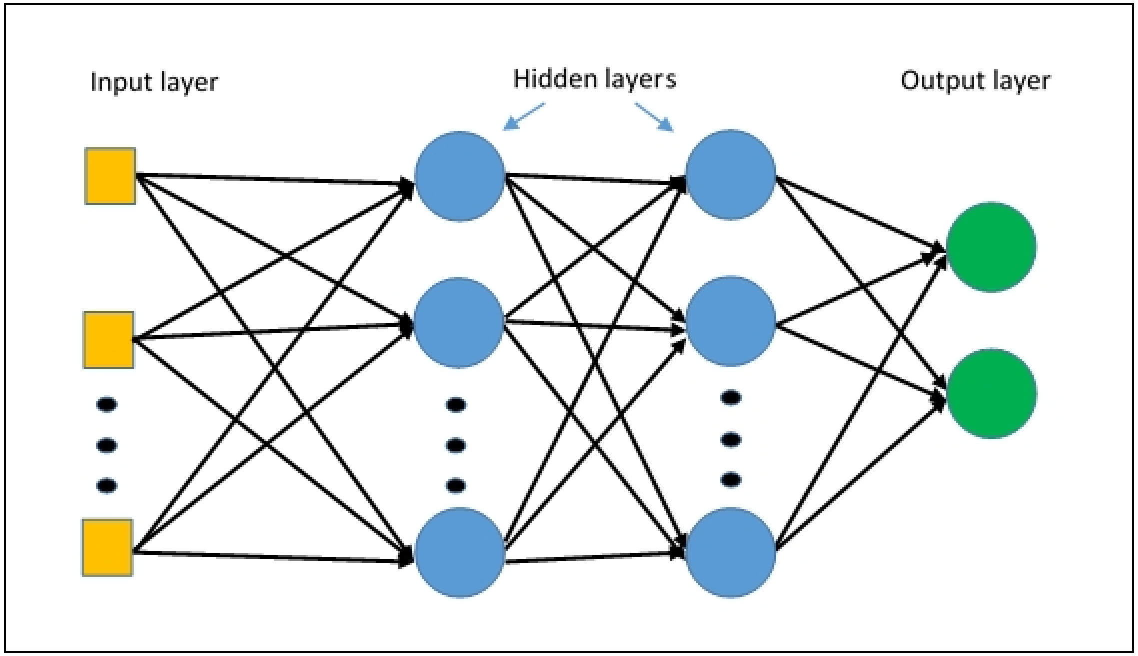
Training = Find $W^{(1)}$ and $W^{(2)}$ so that

$y \approx \bar{y}$ at all training instances x

MLP General Cases

- A multi-layer perceptron with K-1 hidden layers (deep neural network)
  - Activation function at layer j is $\sigma^{(j)}$
  - Matrix $W^{(j)}$ is the parameter matrix mapping from layer j to layer j+1
- Prediction is

$$y=\sigma^{(K)}\left(W^{(K-1)}\sigma^{(K-1)}\left(\ldots\sigma^{(2)}\left(W^{(1)}x\right)\ldots\right)\right)$$



Next lecture: Backpropagation method to train a neural network