CS 331
Computer Vision

10 / 27 / 2026

Instructor: Michael Eckmann

Topics

- Questions / Comments?
- Quick review on Regularization / Weight Decay
- Review on how Logistic Regression can learn high order polynomial boundaries
- Why Neural Networks?

Regularization

- What is regularization?

Regularization

- **Regularization** is a way for us to provide the learning algorithm a preference for some types of functions over others.
- The Deep Learning text book by Goodfellow et.al. defines regularization this way: any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.

Weight Decay

- What is weight decay?

Regularization

- One kind of regularization for linear regression is **weight decay**. The idea is we add in new terms to the loss function such that higher weights are more lossy (cause an increase in the loss function) --- i.e. we tell the loss function to prefer to keep weights small. Small weights result in less overfitting.
- We want to balance this penalty for large weights while also keeping the original loss low.

Regularization

- The loss function with weight decay becomes:

$$J(w) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(\hat{y}^{(i)} - y^{(i)}\right)^2 + \lambda w^T w\right]$$

excluding w_0.

$\hat{y} = w^T x$.

- The weight updates with the partial derivatives of the loss function with weight decay are:

$$w_0 = w_0 - \text{learnRate} * \frac{1}{m}\sum_{i=1}^{m}\left(\hat{y}^{(i)} - y^{(i)}\right) \cdot x_0^{(i)}$$

$$w_j = w_j - \text{learnRate} * \left[\frac{1}{m}\sum_{i=1}^{m}\left(\hat{y}^{(i)} - y^{(i)}\right) \cdot x_j^{(i)} + \frac{\lambda}{m} w_j\right]$$

Regularization for Logistic Regression

FOR LOGISTIC REGR.

COST function as before w/ $\lambda * w^T w$.

$$\hat{y} = \frac{1}{1 + e^{-w^T x}}$$

$$w_0 = w_0 - \text{learnRate} * \frac{1}{m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right) \cdot x_0^{(i)}$$

$$w_j = w_j - \text{learnRate} * \left[\frac{1}{m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right) x_j^{(i)} + \frac{\lambda}{m} w_j \right]$$

Logistic Regression

- Can Logistic Regression learn arbitrarily shaped decision boundaries?

Why Neural Networks?

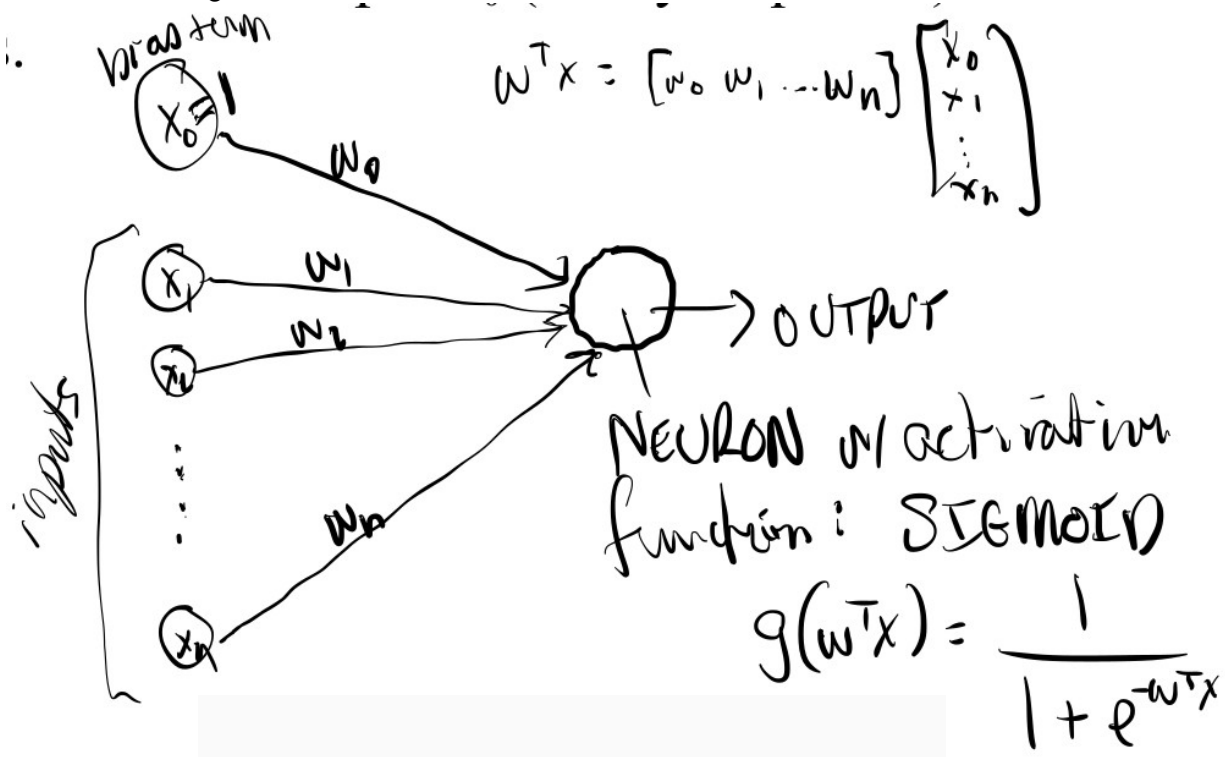
- Why use neural networks if logistic regression can do classification with arbitrary shaped decision boundaries?

Why Neural Networks?

- Why use neural networks if logistic regression can do classification with arbitrary shaped decision boundaries?
- Think of say small images of size 20x20 color pixels. If we treat each channel of each pixel as an input variable, we'll have 1200 input variables.
- Suppose we used logistic regression and wanted to learn (not even a very complex function) say an arbitrary quadratic function as a decision boundary on that data. What new augmented input terms would we need and how many?

Artificial Neuron

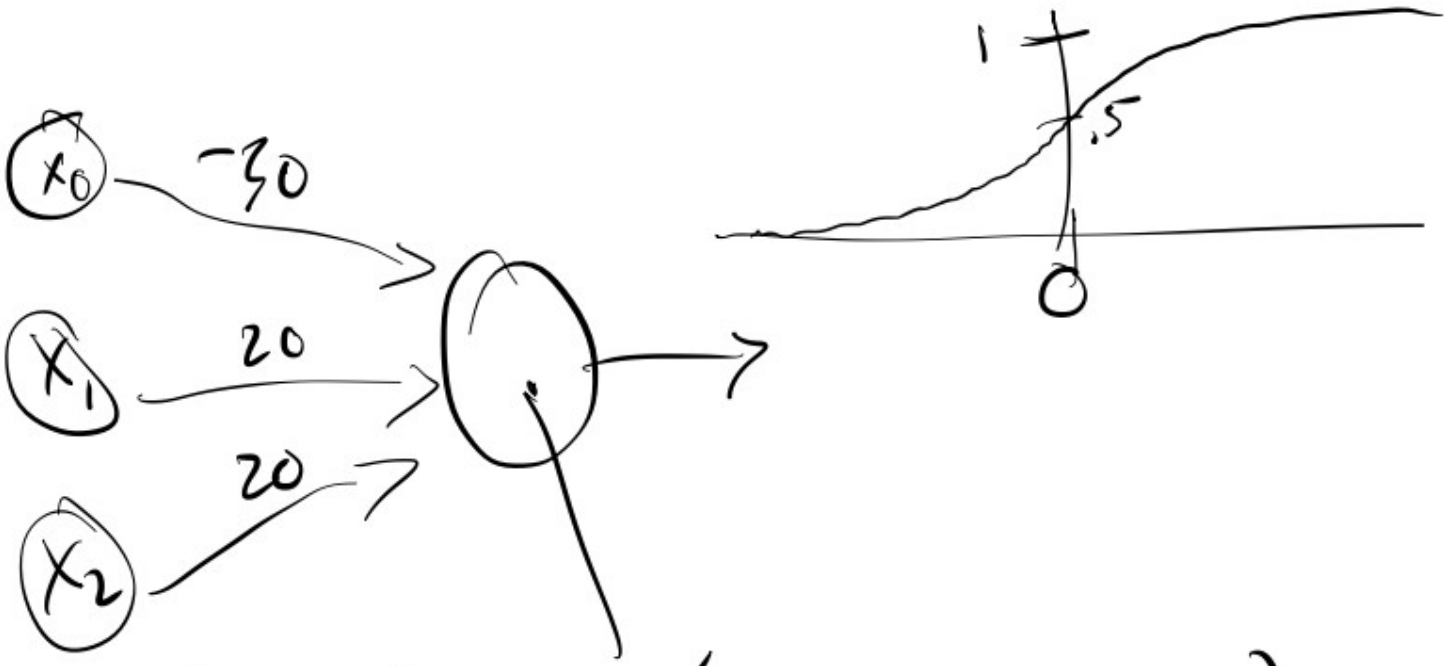
- **weights** on the links
- how the neuron uses the **inputs** and weights to compute an output via an ***activation function***.
- First input x_0 (always equals 1) is called the bias.



Example

- A single neuron that takes binary inputs can be used to compute simple boolean functions like AND and OR.

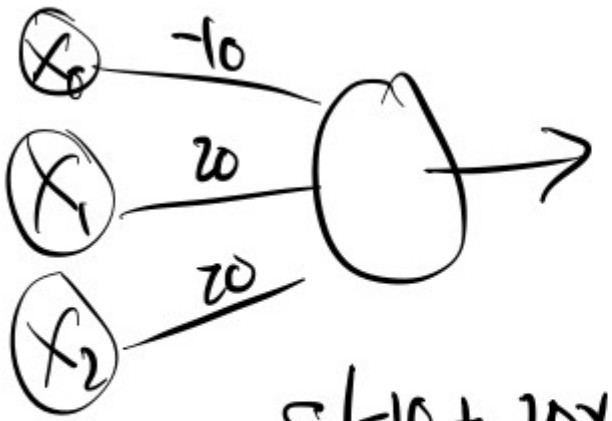
Example for AND



x_1	x_2	$w^T x$	$g(w^T x)$
0	0	-30	0
0	1	-10	0
1	0	-10	0
1	1	10	1

$g(-30 + 20x_1 + 20x_2)$

Example for OR

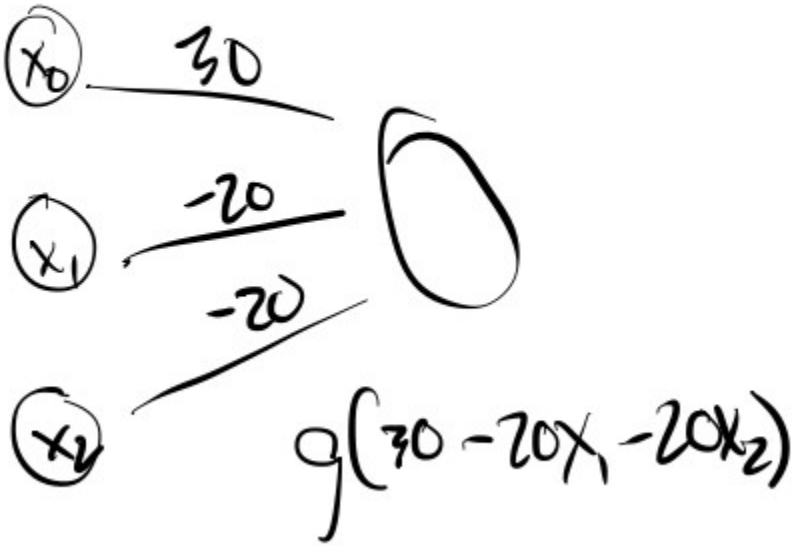


$$g(-10 + 20x_1 + 20x_2)$$

x_1 x_2	$w^T x$	$g(w^T x)$
0 0	-10	0
0 1	10	1
1 0	10	1
1 1	30	1

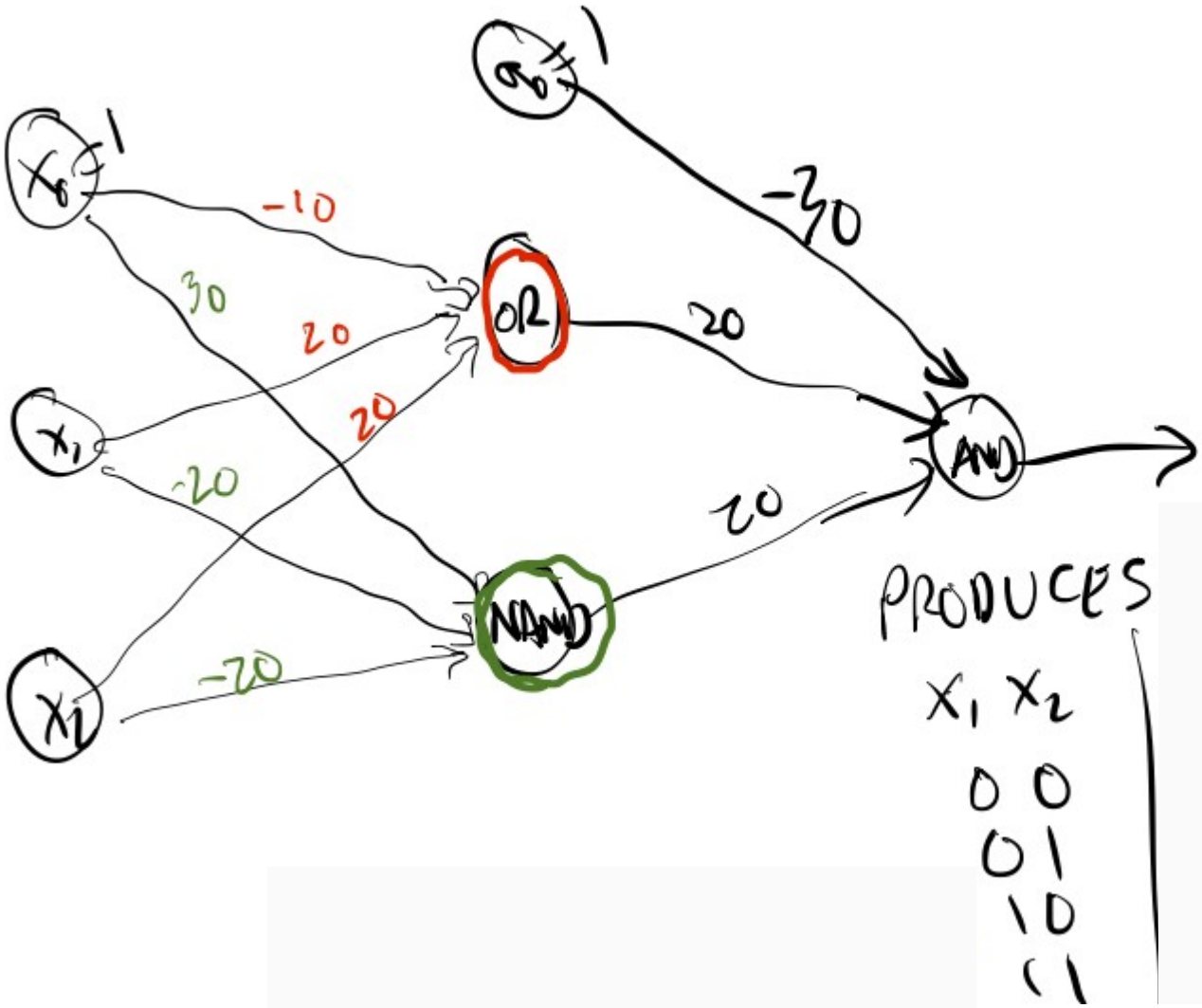
NAND

- x1 XOR x2 = (x1 OR x2) AND (NOT (x1 AND x2))
- (NOT (x1 AND x2)) is called NAND
- Let's show you a NAND neuron

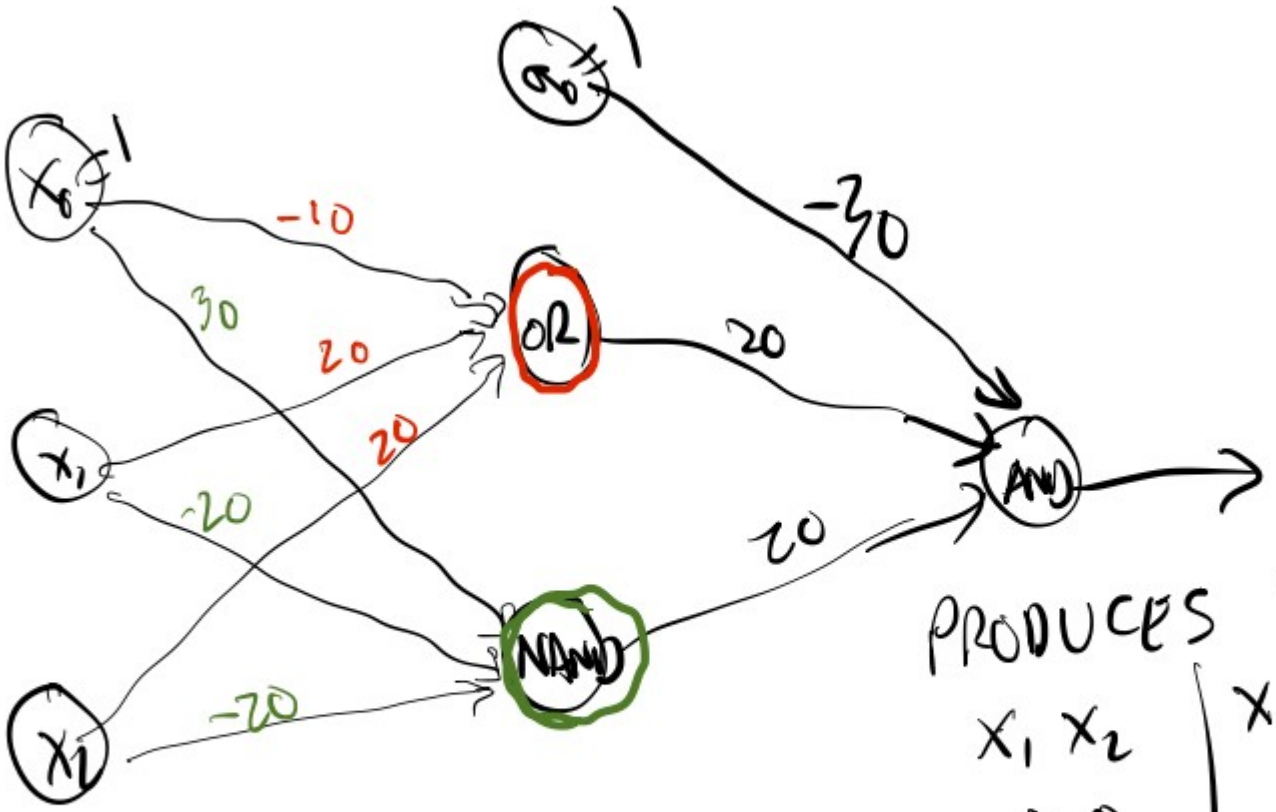


x_1	x_2	x_1 NAND x_2
0	0	1
0	1	1
1	0	1
1	1	0

Compute the results in class



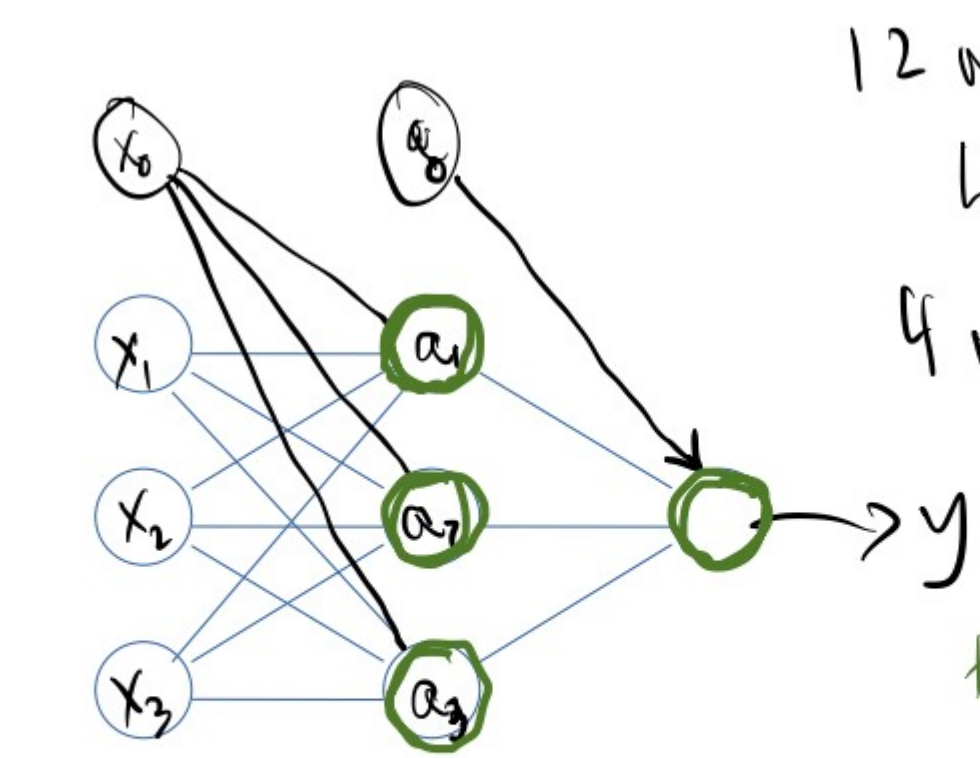
For XOR we need a network



PRODUCES XOR

x_1 x_2	x_1 XOR x_2
0 0	0
0 1	1
1 0	1
1 1	0

Small example



12 weights btwn Layers 1 & 2

4 weights btwn Layers 2 & 3.

the 4 green ☐ contain activation function

LAYER 1 aka input

LAYER 2 hidden layer

LAYER 3 output layer

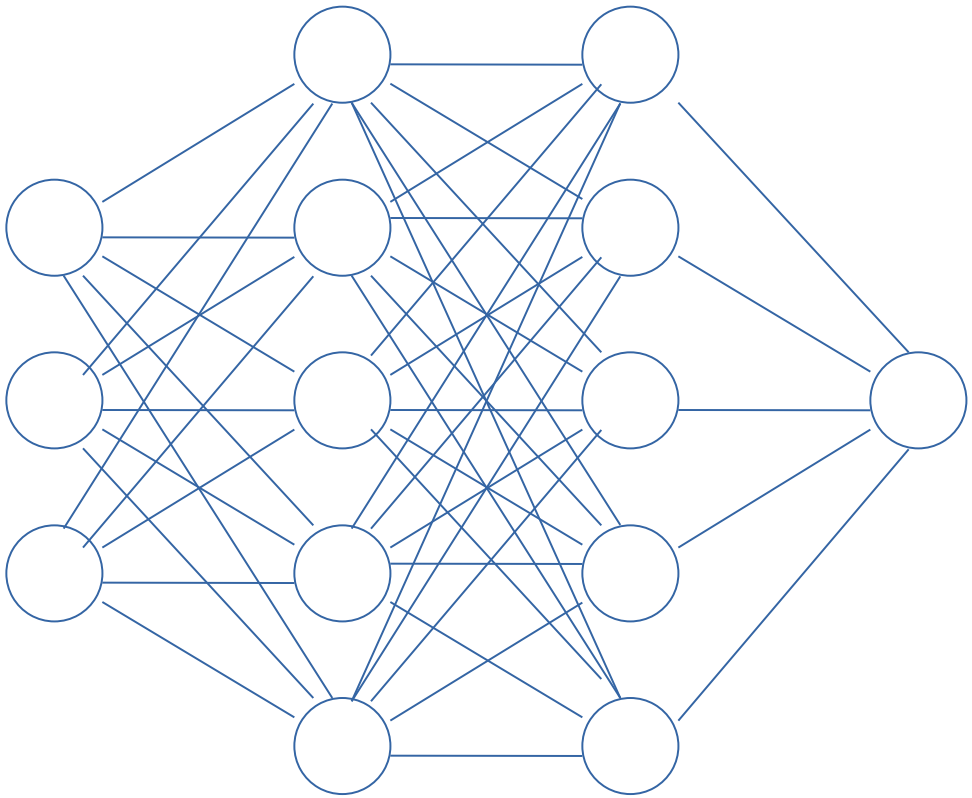
1 -> # of layers

Topics for Neural Networks

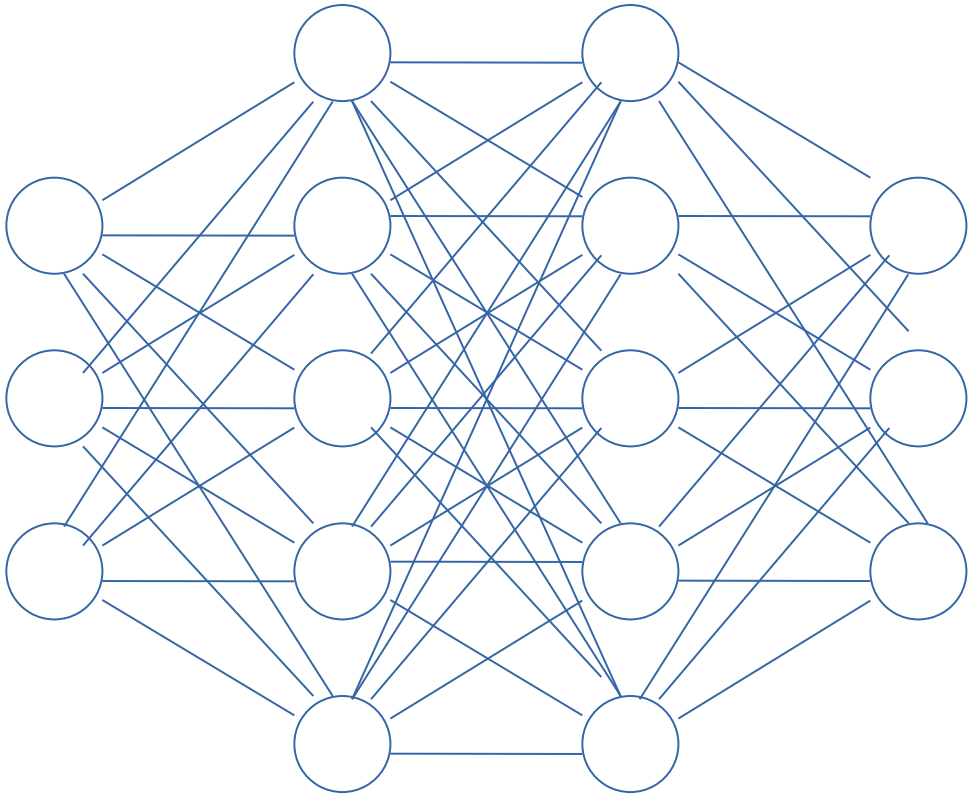
Neural networks

- Input layer, hidden layer(s), output layer
- Sizes of weight matrices
- Introduce another activation function: ReLU
- Forward propagation and an example
- What happens if we create a Neural Network without using activation functions?
- Show how a Neural Network with ReLUs can approximate arbitrary functions

2-class classification



Multiclass classification



Layers and subscripts

- Our neural network will be an input layer followed by some number of hidden layers, followed by an output layer
- We have a weight (aka learned parameter) on each connection between units in consecutive layers
- Each of the last two slides has an input layer (with 3 inputs), 2 hidden layers (each of size 5) and an output layer. The 2-class classifier has one unit in the output layer. The multiclass classifier example has 3 outputs.
- Number of layers in both: L = 4.
- Number of units per layer is s_l (s sub l)
  - The subscript is the letter l, standing for layer number
- $s_1 = 3$, $s_2 = 5$, $s_3 = 5$, $s_4 = 1$ or 3 respectively in our last 2 examples

Layers and subscripts

- $W^{(j)}$ are the weights from layer j to layer j+1
- Layer numbers (j) will be superscripts
- Number of a unit within a layer will be subscript
- The bias term at each layer will not generally be shown and will be numbered with a subscript of 0
- In the last example of the multiclass classifier we will have 3 weight matrices $W^{(1)}$ of size 5x4, $W^{(2)}$ of size 5x6 and $W^{(3)}$ is 1x6. Note: the reason the number of columns is 1 more than the number of units in the layer is due to the bias unit.