Tufts

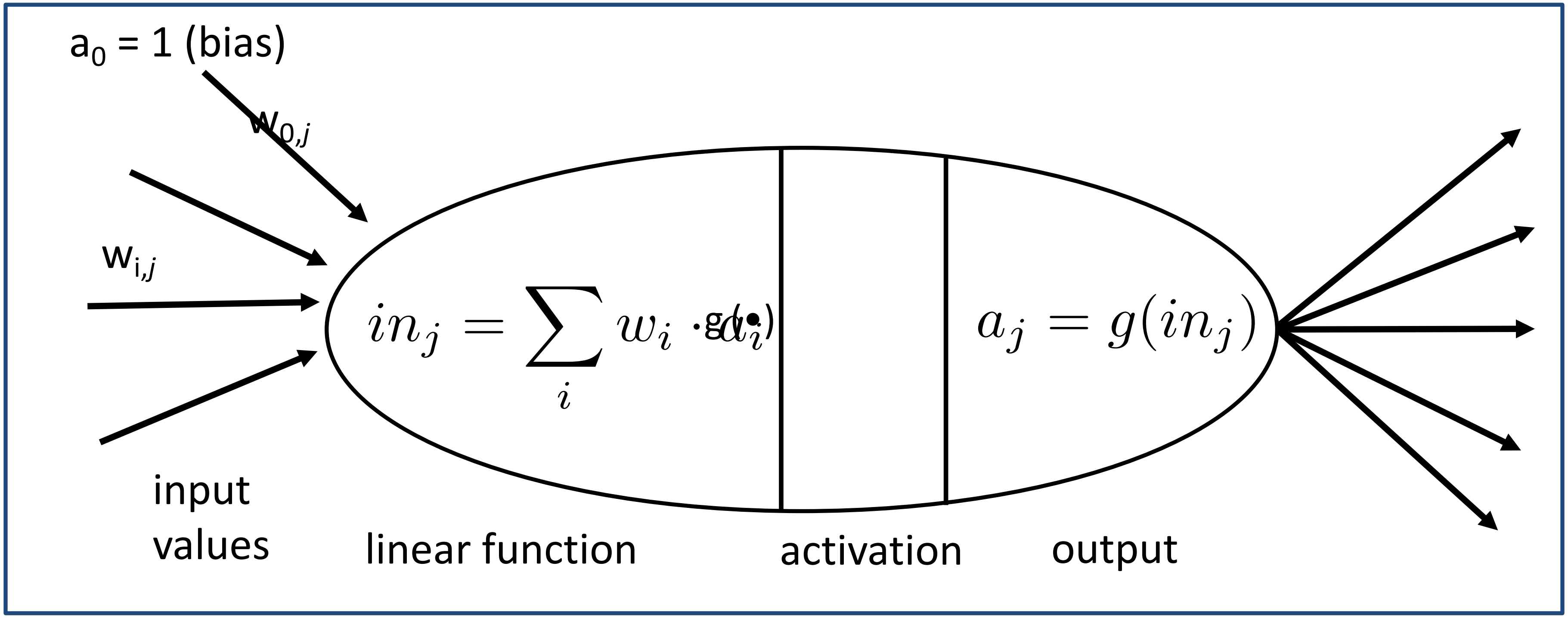
CS135
Introduction to Machine Learning

Lecture 11: Neural Networks

Neural Learning Methods

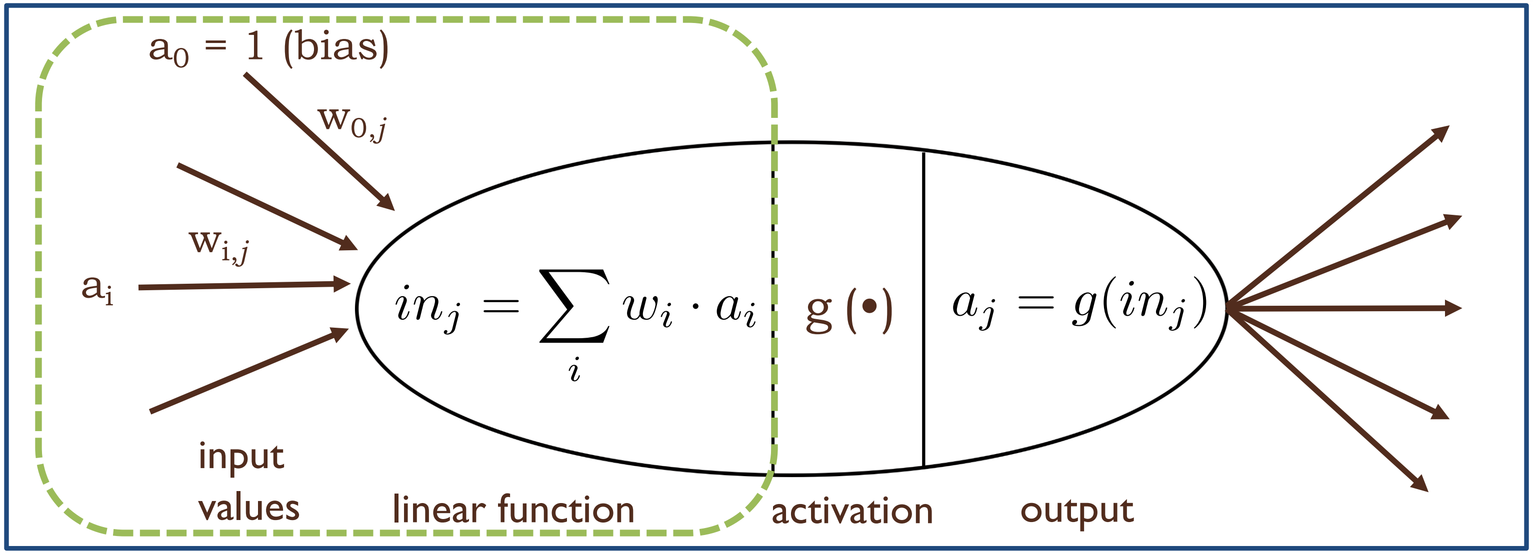
- An obvious source of biological inspiration for learning research: ***the brain***
- The work of McCulloch and Pitts on the perceptron (1943) started as research into how we could precisely model the ***neuron*** and the ***network of connections*** that allow animals (like us) to learn.
- These networks are used as ***classifiers***: given an input, they label that input with a classification or a distribution over possible classifications.

The Basic Neuron Model



- Its **input** is from other neurons or the data input layer; then, it computes function g
- **Output** a_j is either passed along to another set of neurons or used as the final output for the learning problem.

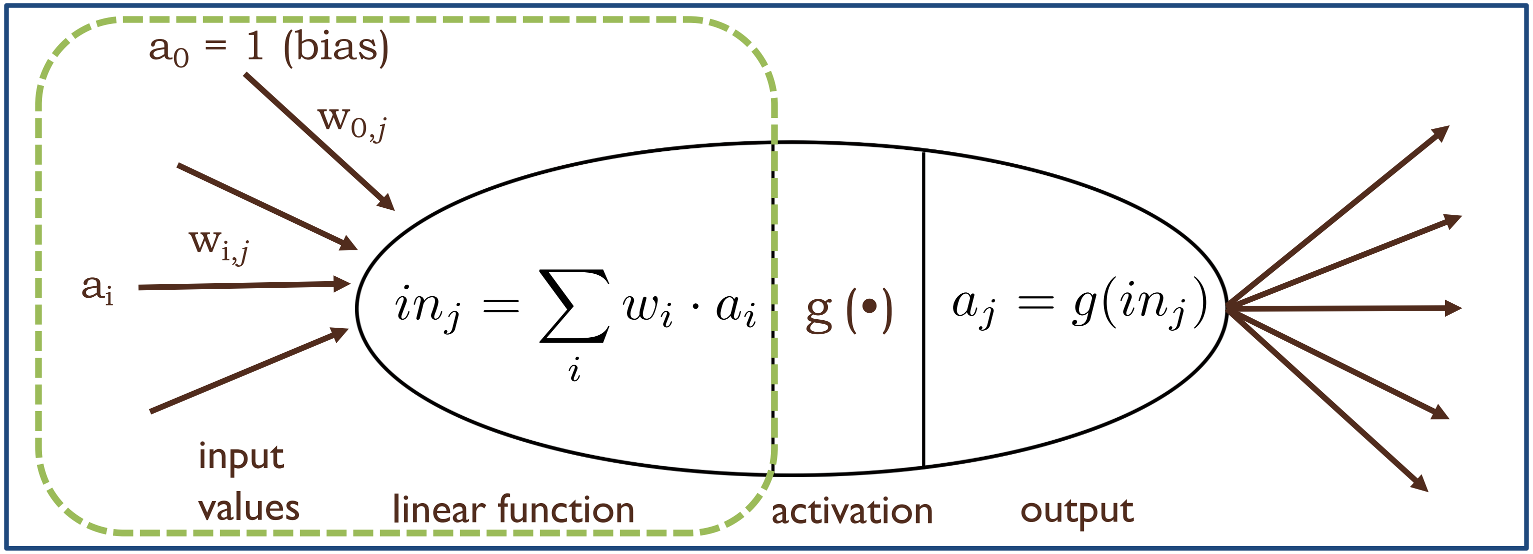
Input Bias Weights



- Each input a_i to neuron j is given a weight $w_{i,j}$
- Each neuron is treated as having a fixed dummy input, $a_0 = 1$
- The input function is then the weighted linear sum:

$$
\begin{aligned}
in_j &= \sum_{i=0}^{n} w_{i,j}\, a_i = w_{0,j}\, a_0 + w_{1,j}\, a_1 + w_{2,j}\, a_2 + \cdots + w_{n,j}\, a_n \\
&= w_{0,j} + w_{1,j}\, a_1 + w_{2,j}\, a_2 + \cdots + w_{n,j}\, a_n
\end{aligned}
$$

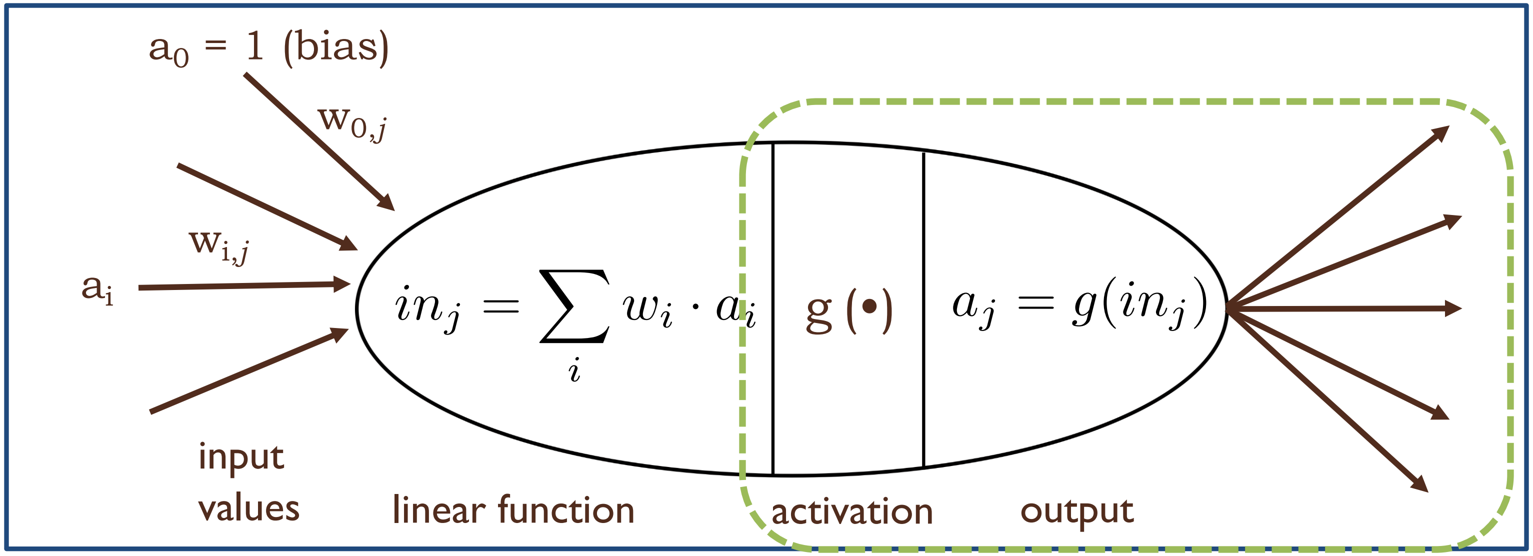
We've Seen This Before!



- The weighted linear sum of inputs, with a dummy, $a_0 = 1$, is just a form of the cross-product that our classifiers have been using all along
  - Remember that the "neuron" here is just another way of looking at the perceptron idea we already discussed

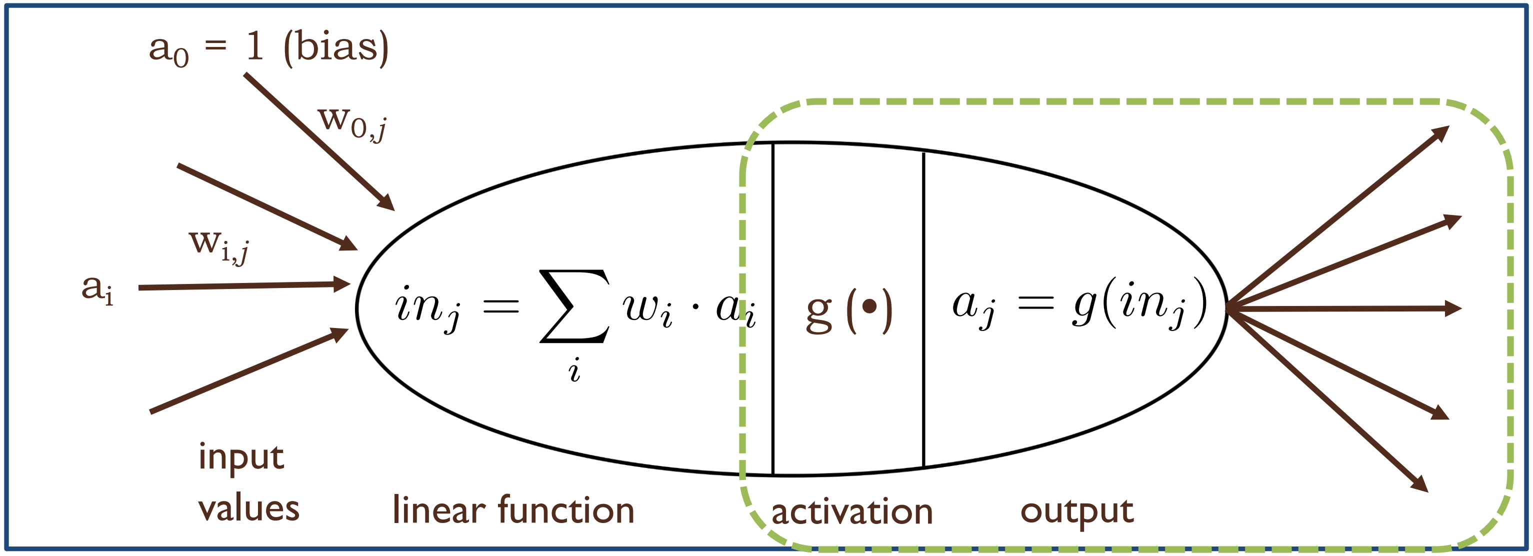
$$
\begin{aligned}
in_j &= \sum_{i=0}^{n} w_{i,j}\, a_i = w_{0,j} + w_{1,j}\, a_1, + w_{2,j}\, a_2 + \cdots + w_{n,j}\, a_n \\
&= \mathbf{w}_j \cdot \mathbf{a}
\end{aligned}
$$

Neuron Output Functions



- While the ***inputs*** to any neuron are treated linearly, the ***output*** function g ***need not*** be linear.
- The power of neural nets comes from the fact that we can combine large numbers of neurons to compute any function (linear or not) that we choose

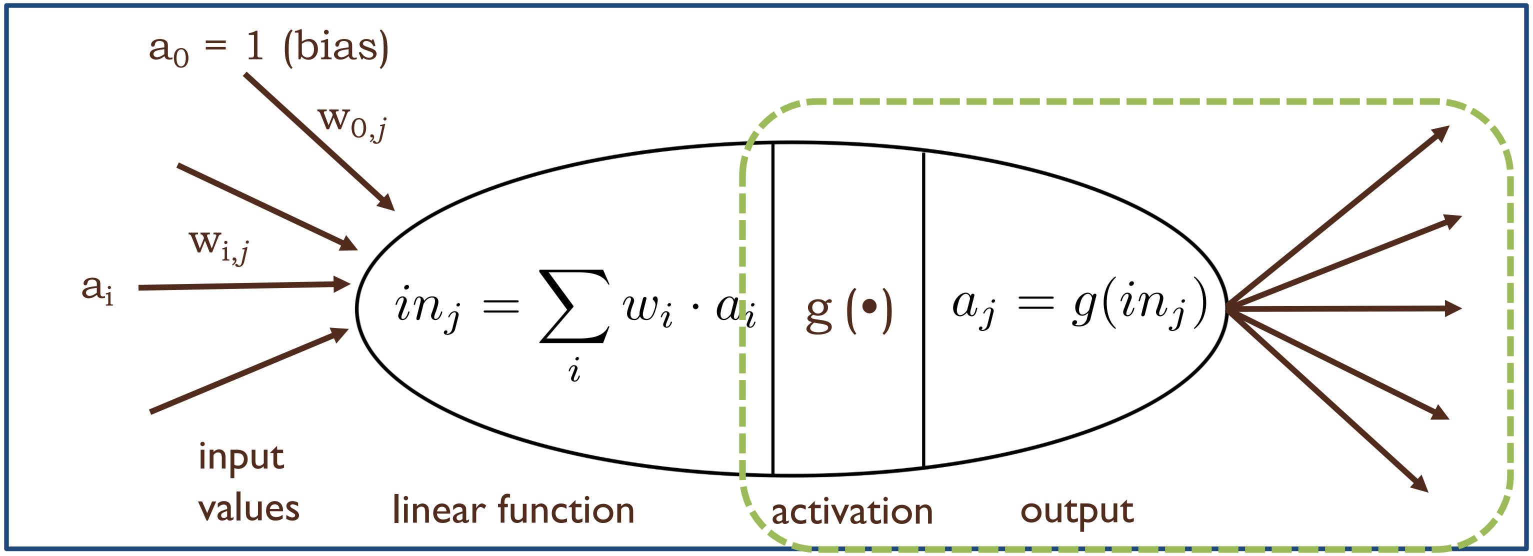
The Perceptron Threshold Function



- One possible function is the binary threshold, which is suitable for "firm" classification problems, and causes the neuron to activate based on a simple binary function:

$$g(in_j) = \begin{cases} 1 & \text{if } in_j \geq 0 \\ 0 & \text{else} \end{cases}$$

The Sigmoid Activation Function



- A function that has been more often used in neural networks is the logistic (also known as the Sigmoid)
- This gives us a "soft" value, which we can often interpret as the ***probability*** of belonging to some output class

$$g(in_j) = \frac{1}{1 + e^{-in_j}}$$

Power of Perceptron Networks

- A single-layer network combines a *linear* function of input weights with the *non-linear* output function
  - If we threshold output, we have a boolean (1/0) function
  - This is sufficient to compute numerous linear functions

x_1 OR x_2

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

x_1 AND x_2

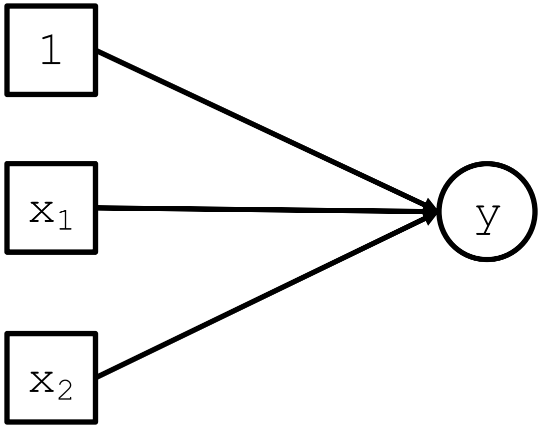
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

Power of Perceptron Networks

- A single-layer network with inputs for variables (x_1, x_2), and bias term (x_0 == 1), can compute the OR of its inputs
  - **Threshold**: (y == 1) if weighted sum (S >= 0); else (y == 0)

x_1 OR x_2

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

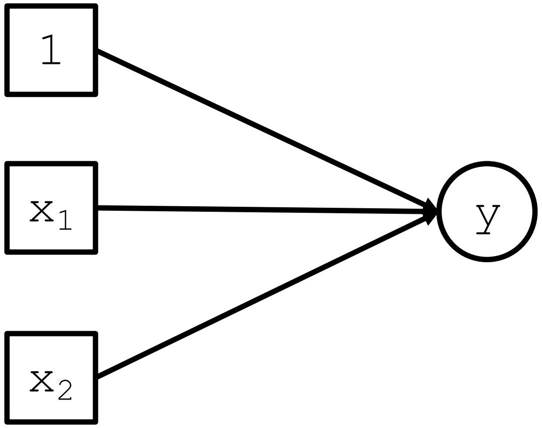


- *What* weights can we apply to the three inputs to produce OR?
  - One answer: $-0.5 + x_1 + x_2$

Power of Perceptron Networks

x_1 AND x_2

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



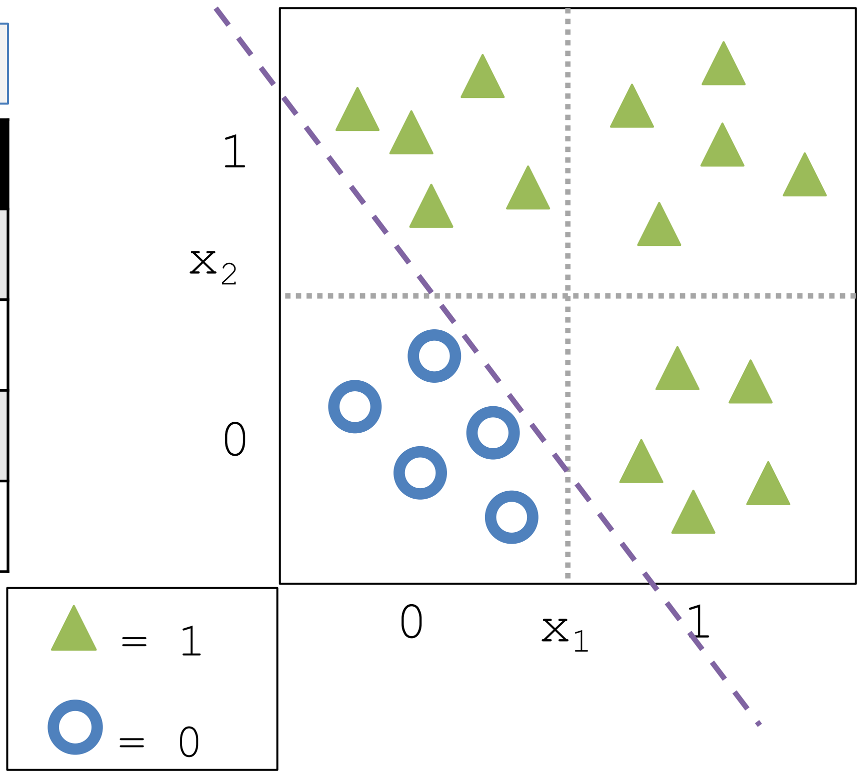
- What about the AND function instead?
  - One answer: $-1.5 + x_1 + x_2$

Linear Separation with Perceptron Networks

- We can think of binary functions as dividing (`x1, x2`) plane
- The ability to express such a function is *analogous* to the ability to linearly separate data in such regions

x_1 OR x_2

$\mathbf{x_1}$	$\mathbf{x_2}$	**y**
0	0	0
0	1	1
1	0	1
1	1	1

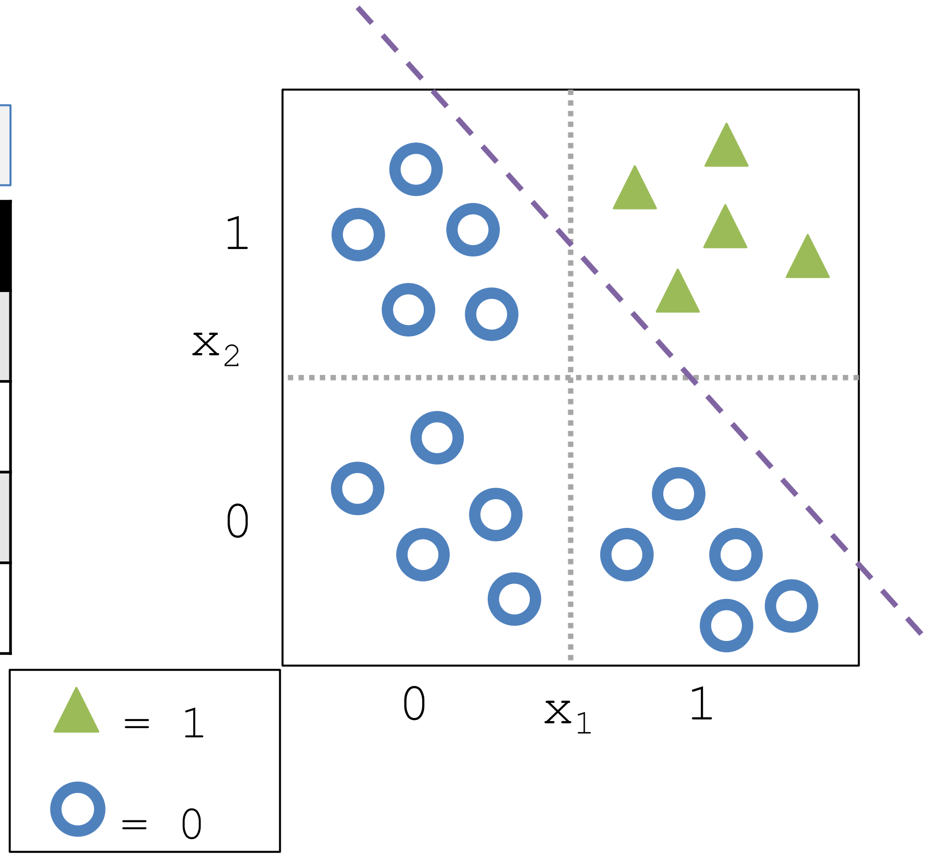


Linear Separation with Perceptron Networks

- We can think of binary functions as dividing (`x1, x2`) plane
- The ability to express such a function is *analogous* to the ability to linearly separate data in such regions

x_1 AND x_2

$\mathbf{x_1}$	$\mathbf{x_2}$	$\mathbf{y}$
0	0	0
0	1	0
1	0	0
1	1	1

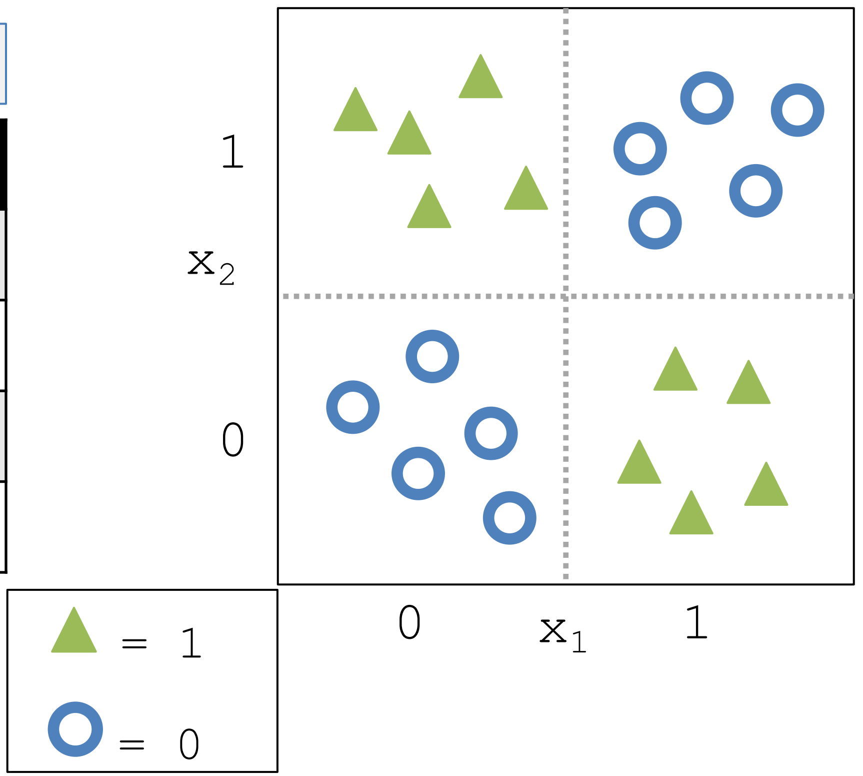


Functions with Non-Linear Boundaries

- There are some functions that cannot be expressed using a single layer of linear weighted inputs, and a non-linear output
- Again, this is analogous to the *inability* to linearly separate data in some cases

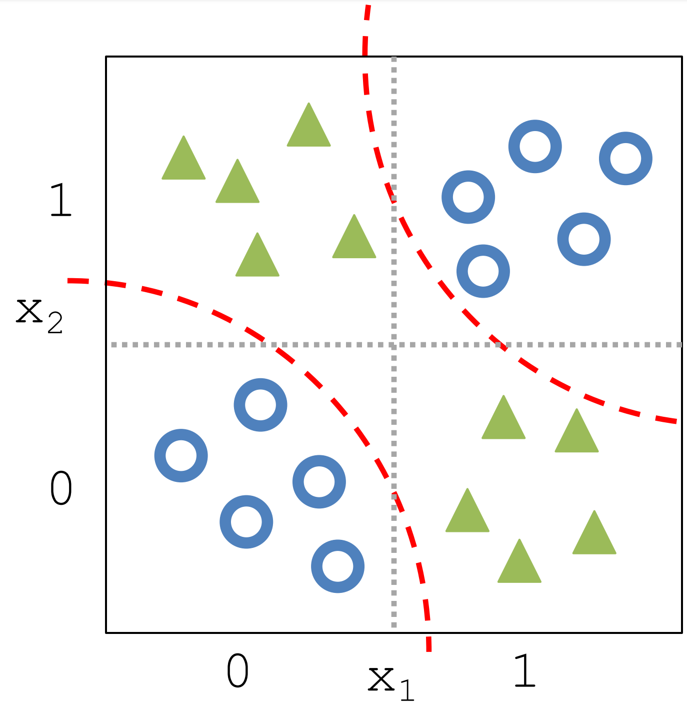
x_1 XOR x_2

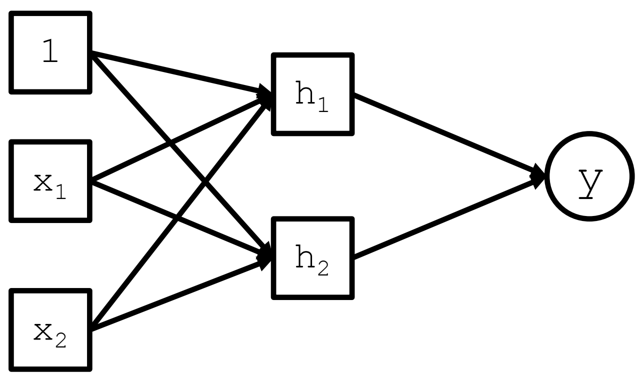
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



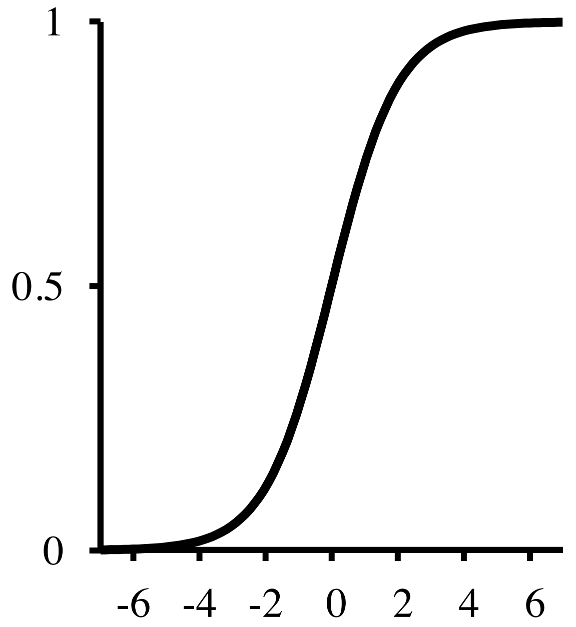
MLP's for Non-Linear Boundaries

- Neural networks gain expressive power because they can have *more than* one layer
- A multi-layer perceptron has one or more hidden layers between input and output
- Each hidden node applies a non-linear activation function, producing output that it sends along to the next layer
  - In such cases, much more complex functions are possible, corresponding to non-linear decision boundaries (as in current homework assignment)





Review: Properties of the Sigmoid Function



$$g(in_j) = \frac{1}{1 + e^{-in_j}}$$

- The Sigmoid takes its name from the shape of its plot
- It always has a value in range:

$$0 \leq x \leq 1$$

- The function is everywhere differentiable, and has a ***derivative*** that is easy to calculate, which turns out to be useful for learning:

$$g'(in_j) = g(in_j)(1 - g(in_j))$$

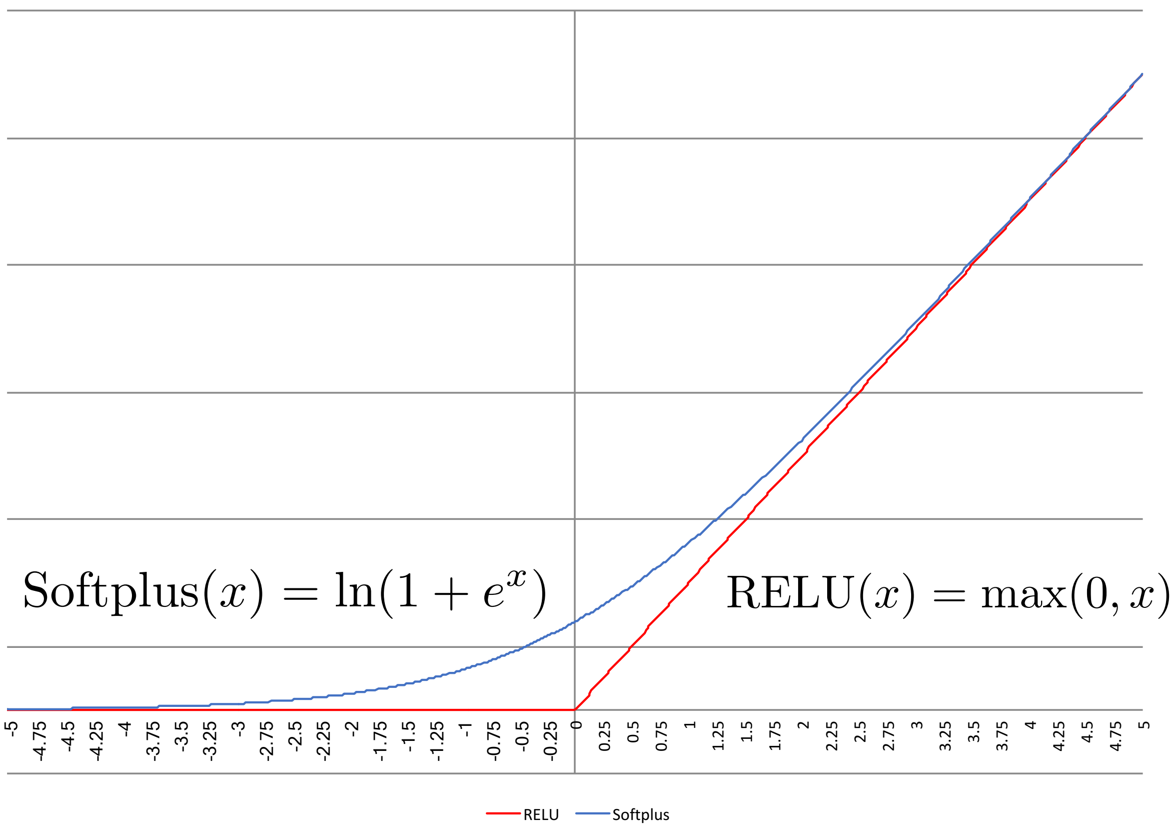
Do We Always Use the Logistic Sigmoid?

- While historically popular, the logistic function is ***not*** always used in modern neural network research
  - There are many other functions that can be, and are, used
  - Some models even use combinations of different functions on different layers of the network
  - Often, the logistic is used at the final layer only, where it is sometimes called a ***softmax*** (probability) function
  - In our presentation, we will assume the logistic, but the overall details of the key algorithm do not change if we use something else

- In general, we want a function that is

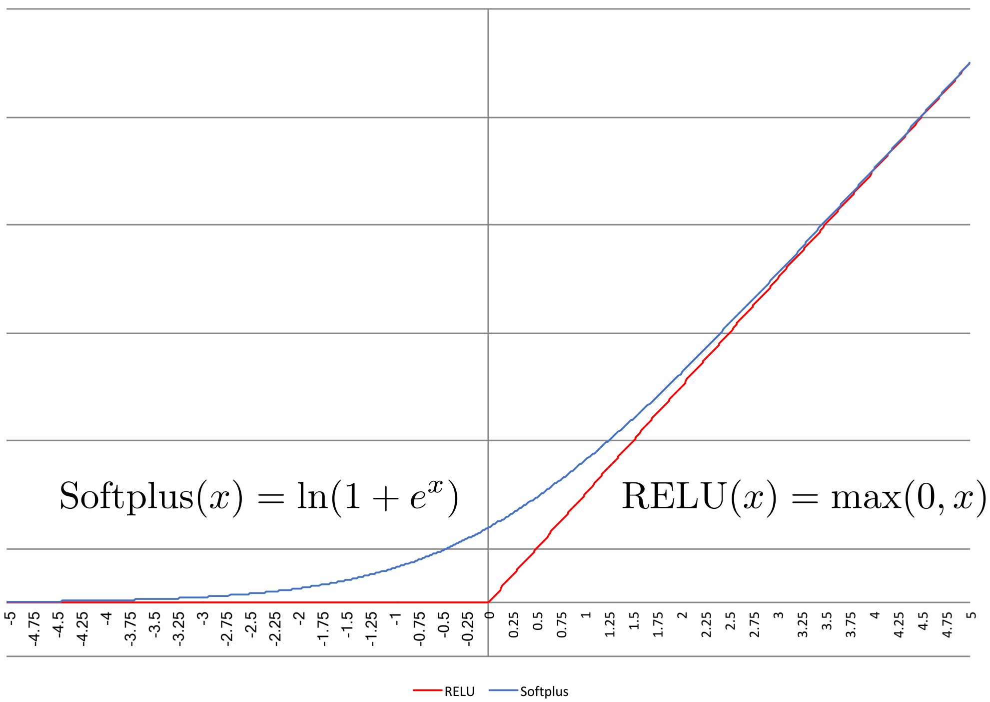
1. Non-linear: allowing for more complex outputs.
2. Differentiable: standard back-propagation algorithms for learning in the networks use gradient-based approaches, and require access to the derivative of the function

Other Popular Activation Functions



- The rectifier (or "ramp") function is popular for many modern applications
  - A network using the rectifier is known as a rectifier linear unit (ReLU)
- The Softplus function is a smooth approximation to the rectifier

Other Popular Activation Functions



- The ReLU function is ***partially*** differentiable:

$$\frac{\delta\,\text{ReLU}}{\delta x}(x) = \begin{cases} 0 & \text{if input } x < 0 \\ 1 & \text{if input } x > 0 \\ undef & \text{if input } x = 0 \end{cases}$$

- For many purposes, the undefined value of the derivative is simply set ***arbitrarily*** (say to 0.5)

- Alternatively, if using Softplus approximation, we have a well-defined derivative everywhere:

$$\frac{\delta\,\text{Softplus}}{\delta x}(x) = \frac{1}{1 + e^{-x}}$$

The derivative of Softplus is the Sigmoid Logistic!

Tufts

Activation Functions Everywhere!

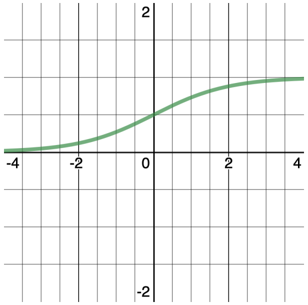
- Logistic $f(x) = \frac{1}{1+e^{-x}}$ $\frac{\delta f}{\delta x}(x) = f(x)(1 - f(x))$
- ReLU $f(x) = \max(0, x)$ $\frac{\delta f}{\delta x}(x) = \{0,\ undef,\ 1\}$
- Softplus $f(x) = \ln(1 + e^x)$ $\frac{\delta f}{\delta x}(x) = \frac{1}{1+e^{-x}}$
- Hyperboli c Tangent $f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$ $\frac{\delta f}{\delta x}(x) = 1 - f(x)^2$
- Gaussian $f(x) = e^{-\frac{x^2}{2}}$ $\frac{\delta f}{\delta x}(x) = -x\, f(x)$

Tufts

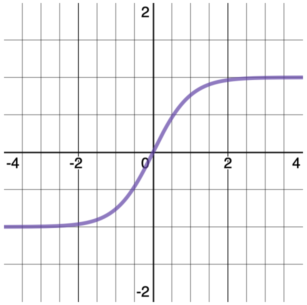
Choosing Activation Functions

- Functions have different pros and cons:

1. Sigmoid: historically popular, less so today
   - Susceptible to saturation: very large weights, tiny gradients
   - Not zero-centered, which is sometimes inconvenient
   - More popular as an *output probability* function (generally in a softmax manner, with values are normalized to sum to 1)



2. Hyperbolic tangent
   - Can saturate like the sigmoid, but *is* zero-centered



3. ReLU/Softplus: most popular function in modern uses
   - ReLU is susceptible to “dying” neurons (these do not contribute to output in any real way)
   - Sensitive to learning rate
   - Softplus sometimes preferred, due to its smoothness