Logistic Regression to Deep Networks and Transfer Learning

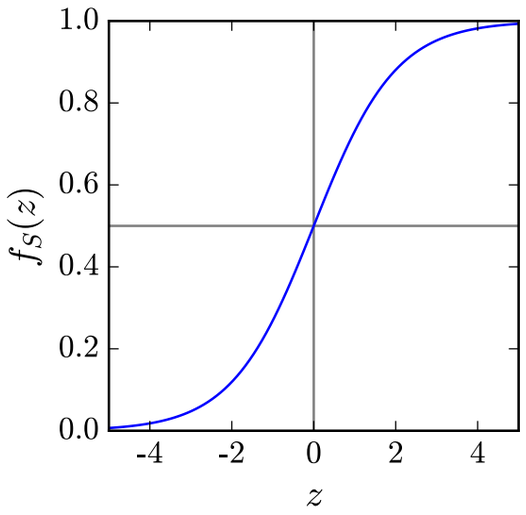
https://github.com/bostdiek/IntroToMachineLearning

Data for the lecture today can be found at

https://drive.google.com/file/d/1VjsNU6OwjHI-0vQqBAW517-0HFHj6PGx/view?usp=sharing

Logistic Regression

What if we are trying to predict a class, not a number?

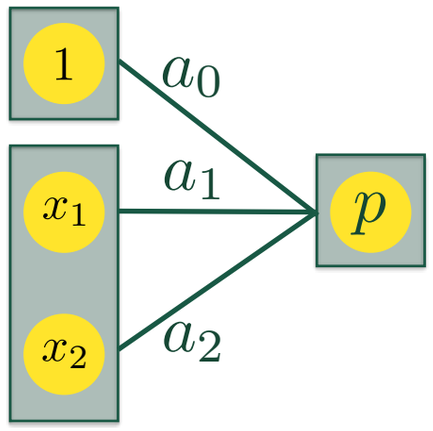


$$L(\vec{x},\vec{y},\vec{a}) = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log\left(f_S(p(x,a))\right) + (1-y_i)\log\left(1-f_S(p(x,a))\right)\right)$$

$$f_S(z) = \frac{1}{1+\mathrm{e}^{-z}} \qquad z = p(x,a)$$

What is $p(x,a)$?

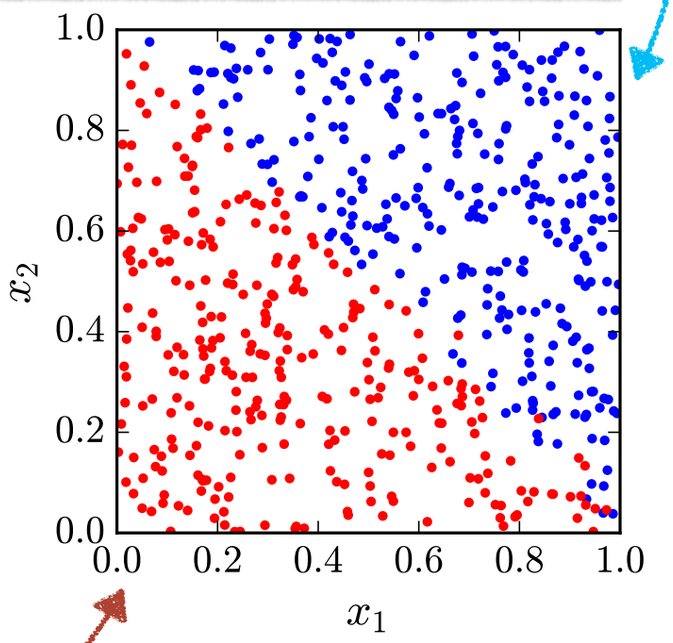
$$p(x,a) = a_0 + x_1 a_1 + x_2 a_2$$



Minimize the loss with respect to $\vec{a}$

Boundary at $p(x,a) = 0$

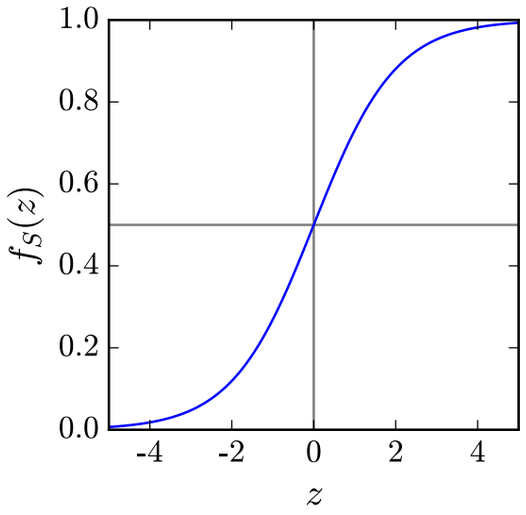
Large + values of p



Large - values of p

Logistic Regression

What if we are trying to predict a class, not a number?

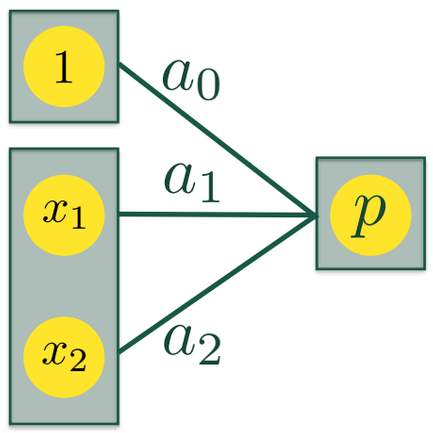


$$L(\vec{x},\vec{y},\vec{a}) = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log\left(f_S(p(x,a))\right) + (1-y_i)\log\left(1-f_S(p(x,a))\right)\right)$$

$$f_S(z) = \frac{1}{1+\mathrm{e}^{-z}} \qquad z = p(x,a)$$

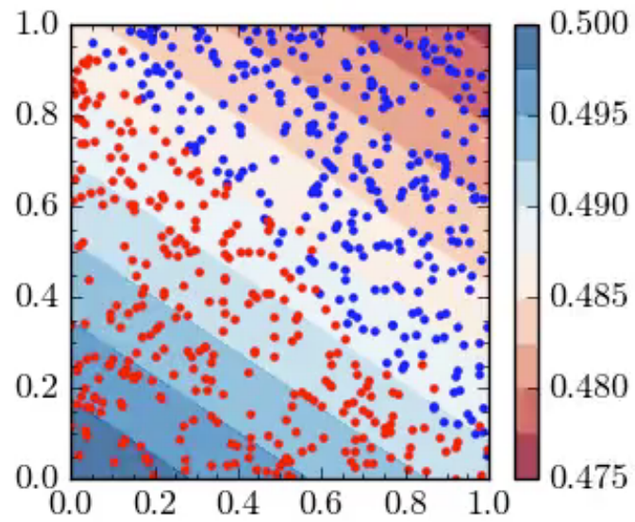
What is $p(x,a)$?

$$p(x,a) = a_0 + x_1 a_1 + x_2 a_2$$



Minimize the loss with respect to $\vec{a}$

Boundary at $p(x,a) = 0$

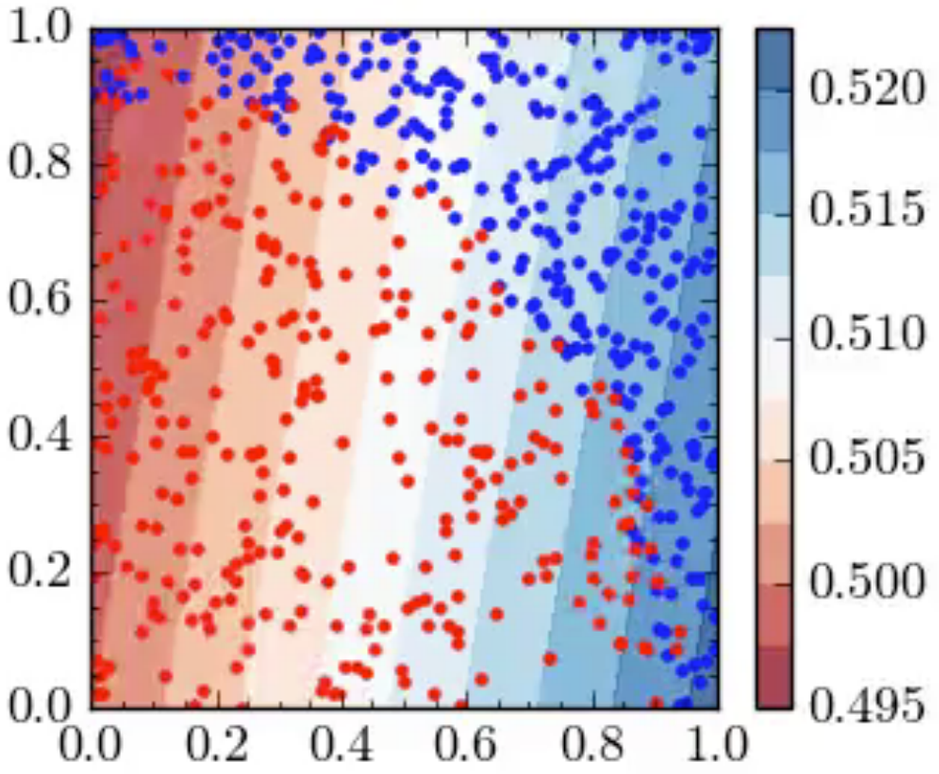


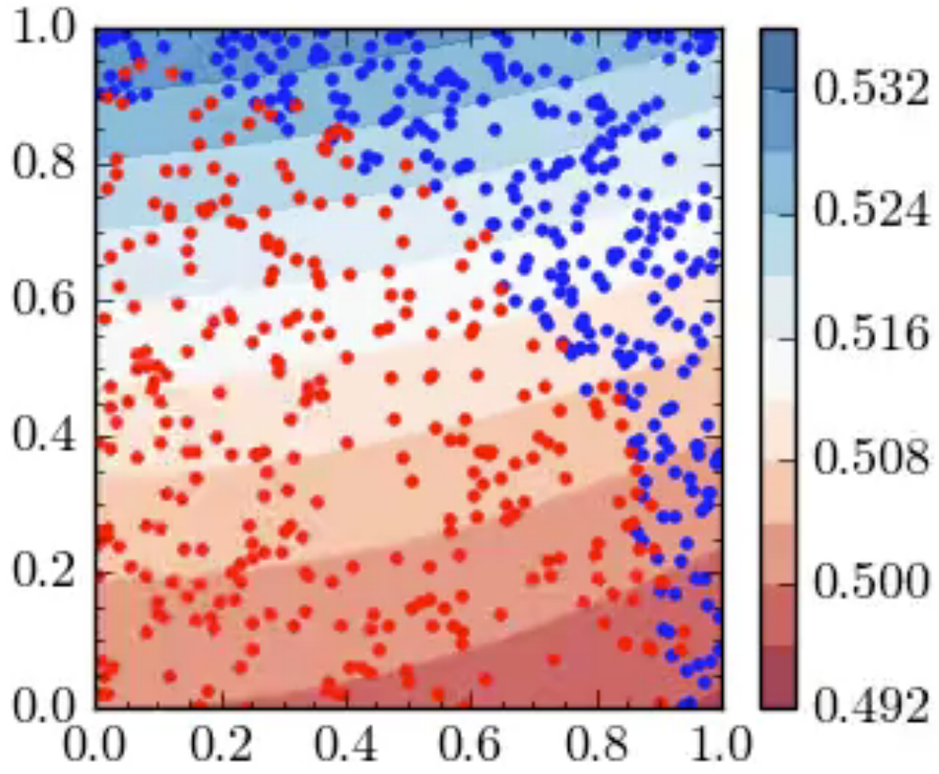
Logistic Regression

What if there is a shape in the data?

$$p(x, a) = a_0 + x_1 a_1 + x_2 a_2$$

$$p(x, a) = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_1^2 + a_4 x_2^2 + a_5 x_1 x_2$$



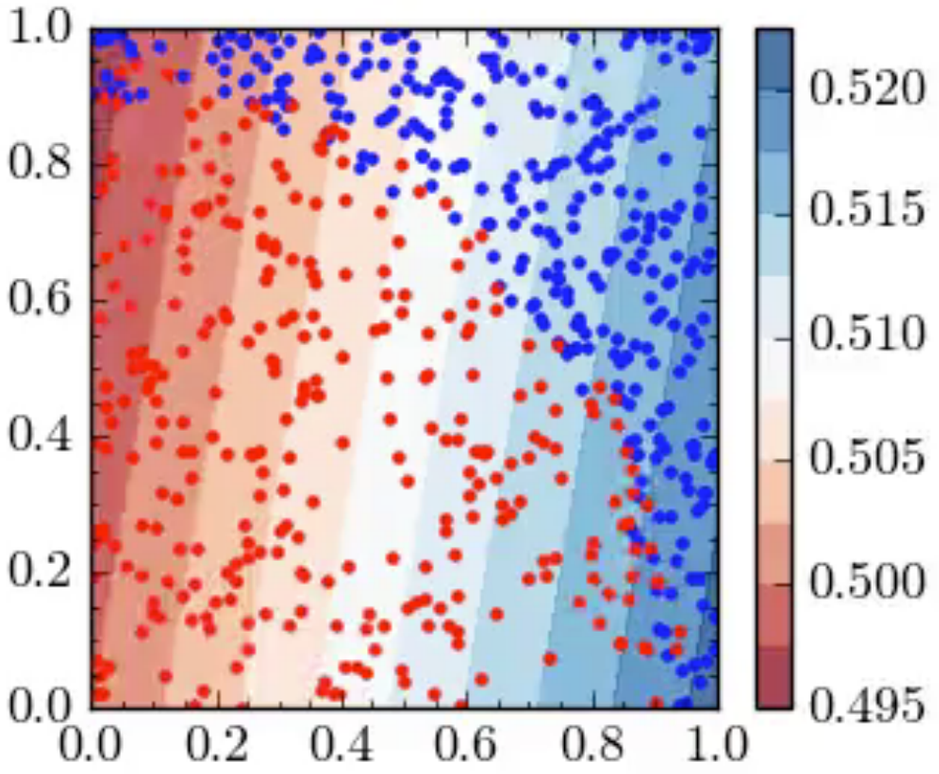


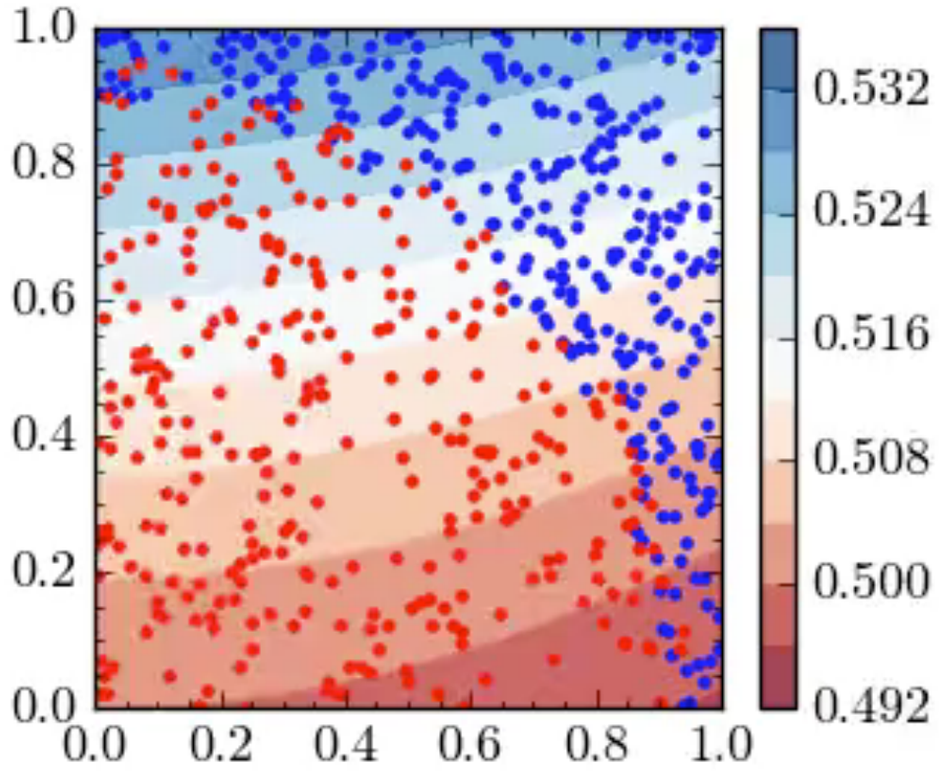
Logistic Regression

What if there is a shape in the data?

$$p(x, a) = a_0 + x_1 a_1 + x_2 a_2$$

$$p(x, a) = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_1^2 + a_4 x_2^2 + a_5 x_1 x_2$$





Regression Review

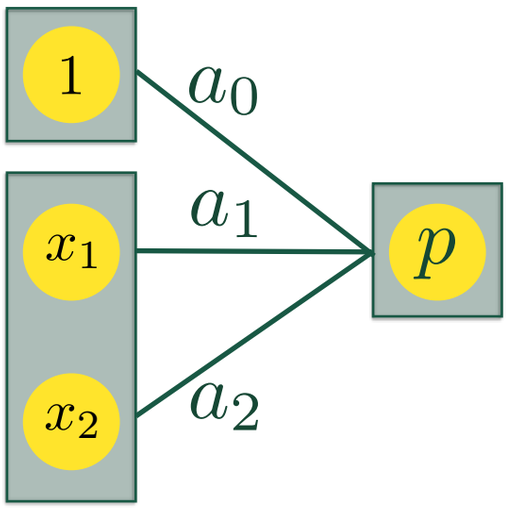
1. Can use nearly the same process for fitting a curve (predicting a number) or classification

2. Minimize a defined cost function

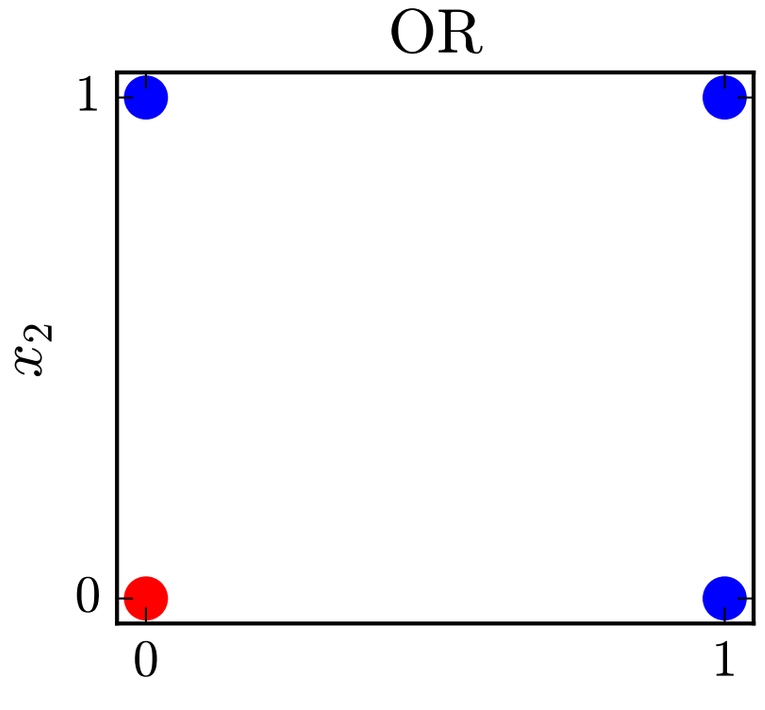
3. Easy to add parameters if shape is unknown — worry about over-fitting

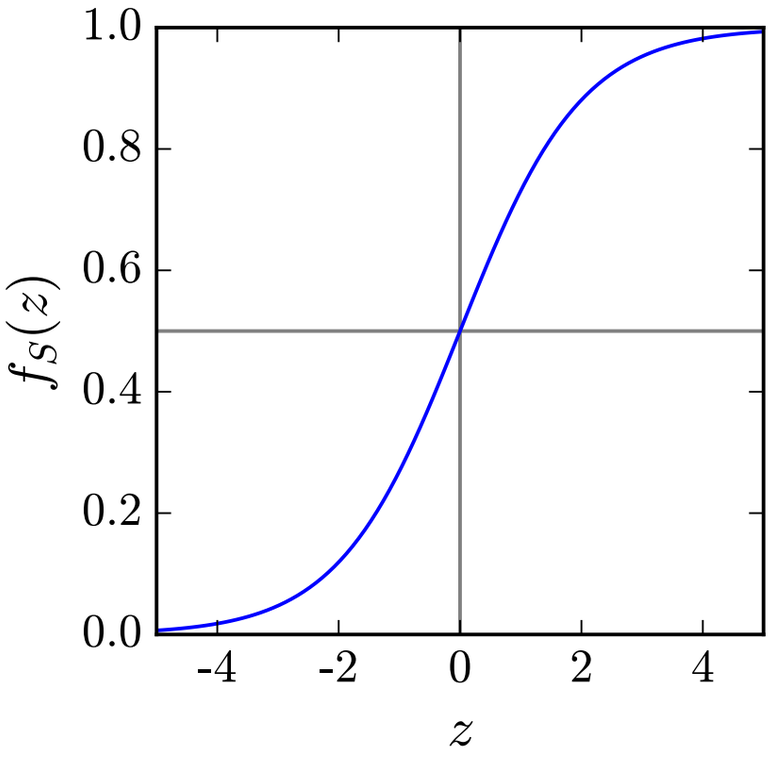
4. If many inputs and complicated shapes, number of parameters necessary grows very quickly

Neural Networks

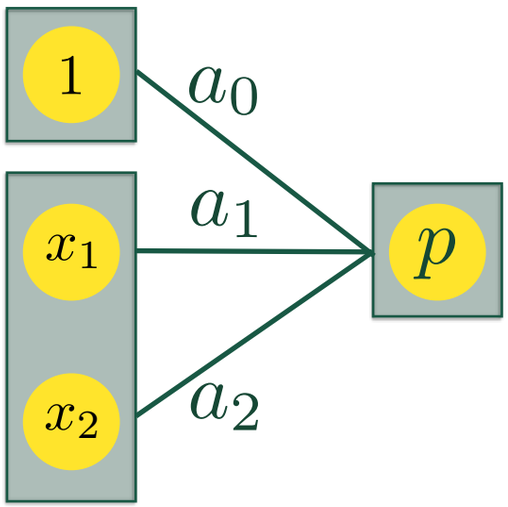


OR		
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1



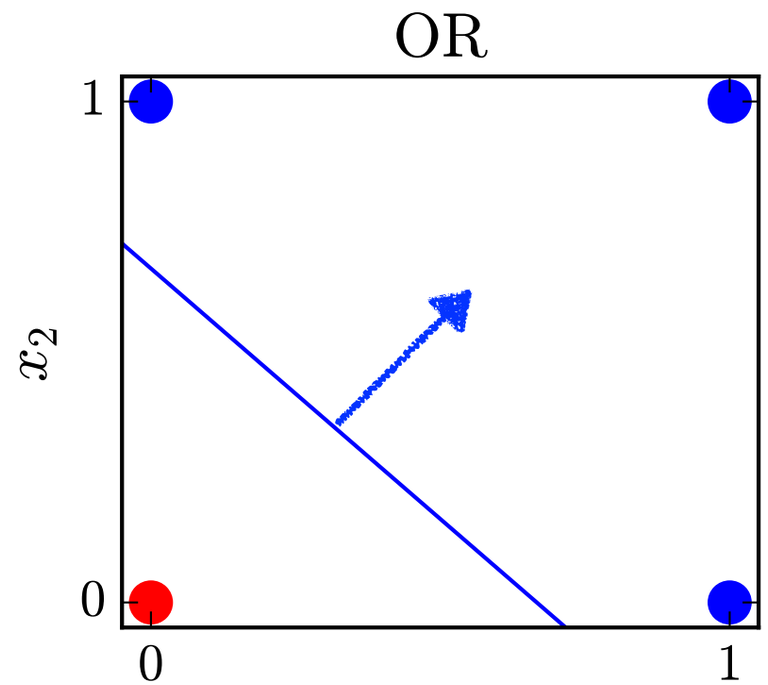


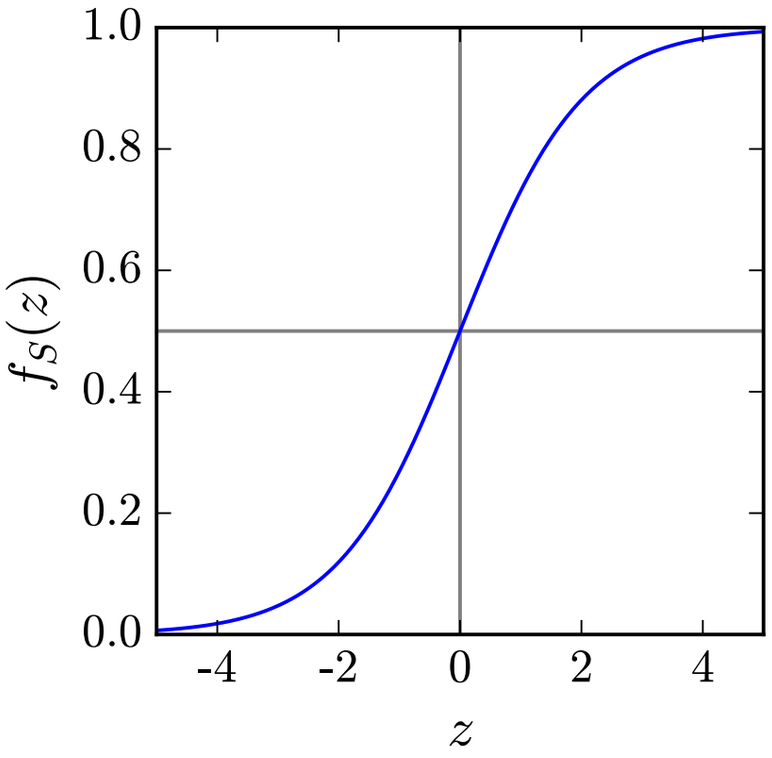
Neural Networks



OR

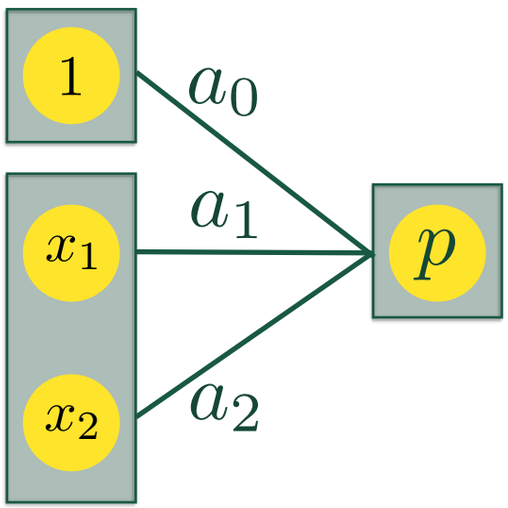
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

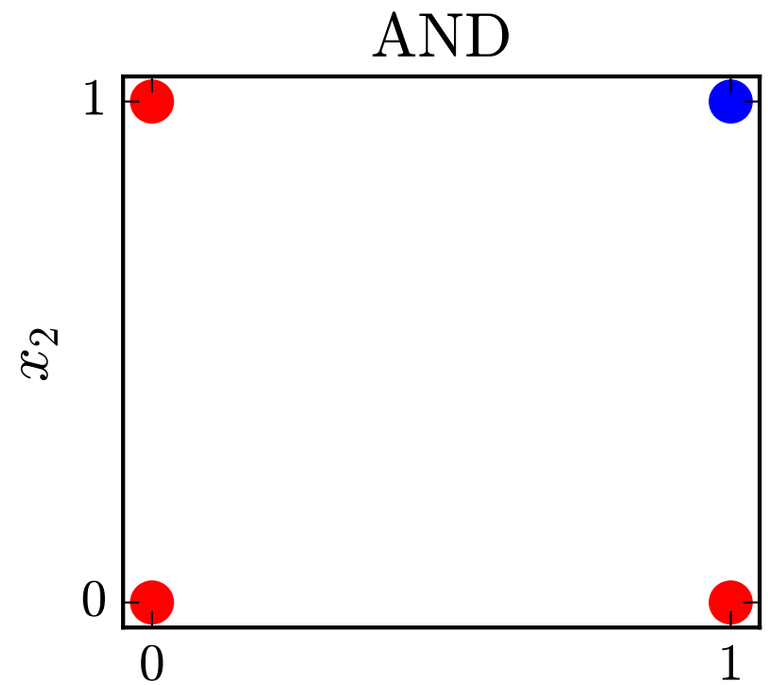




$$a_0 = -10, \quad a_1 = 15, \quad a_2 = 15$$

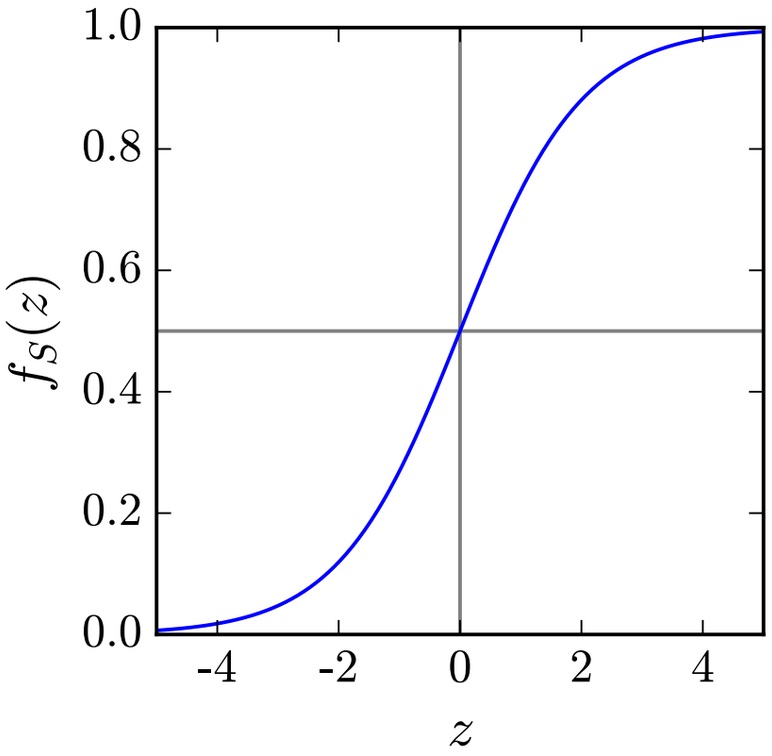
Neural Networks



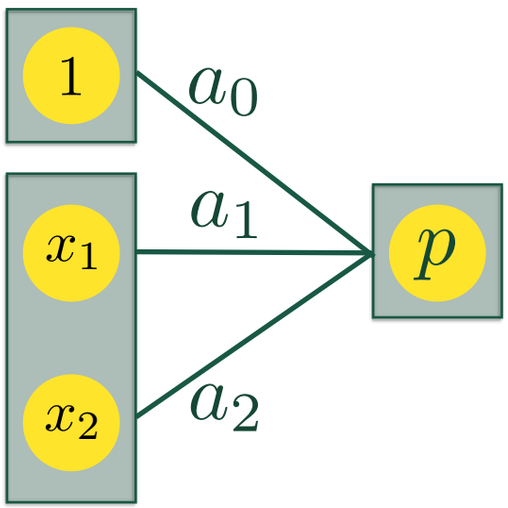


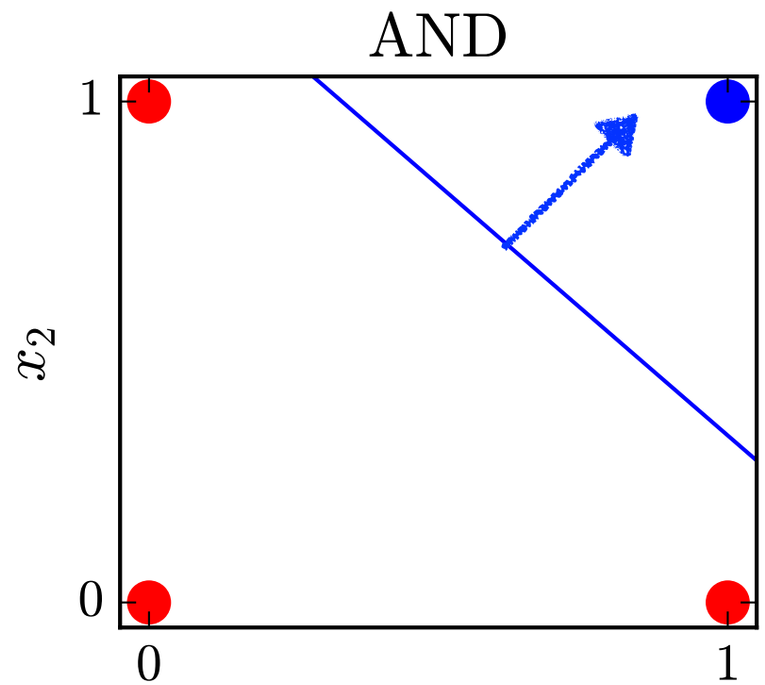
AND

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



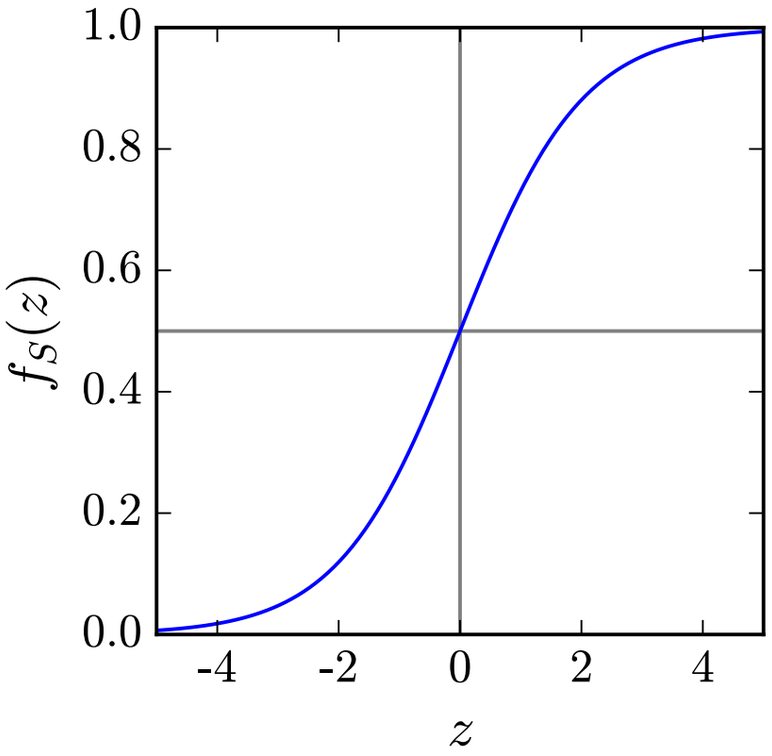
Neural Networks





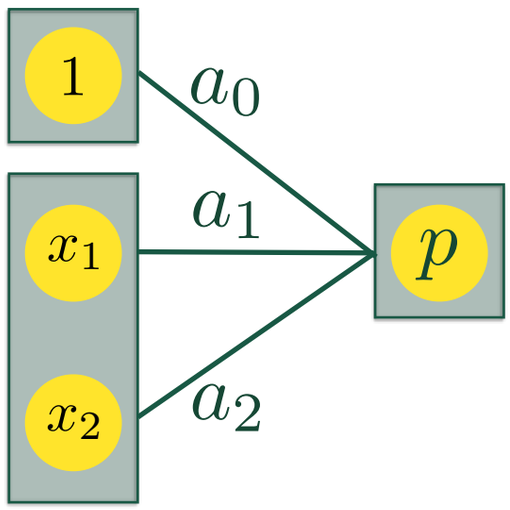
AND

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



$$a_0 = -20, \quad a_1 = 15, \quad a_2 = 15$$

Neural Networks



OR

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

AND

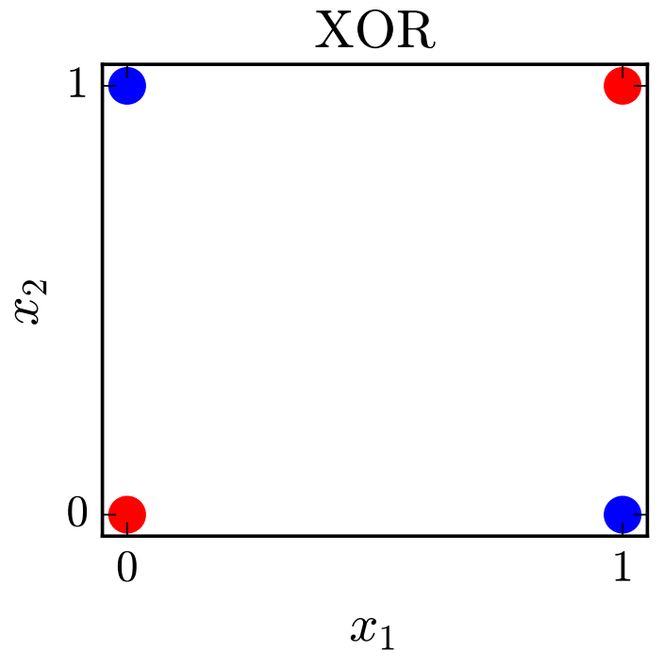
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

$a_0 = -10, \quad a_1 = 15, \quad a_2 = 15$

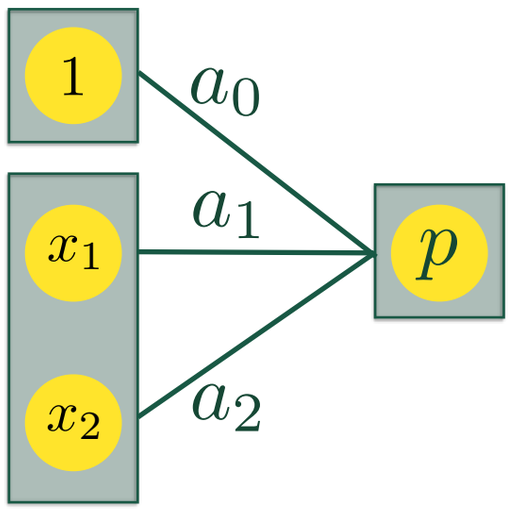
$a_0 = -20, \quad a_1 = 15, \quad a_2 = 15$

XOR

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



Neural Networks



OR

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

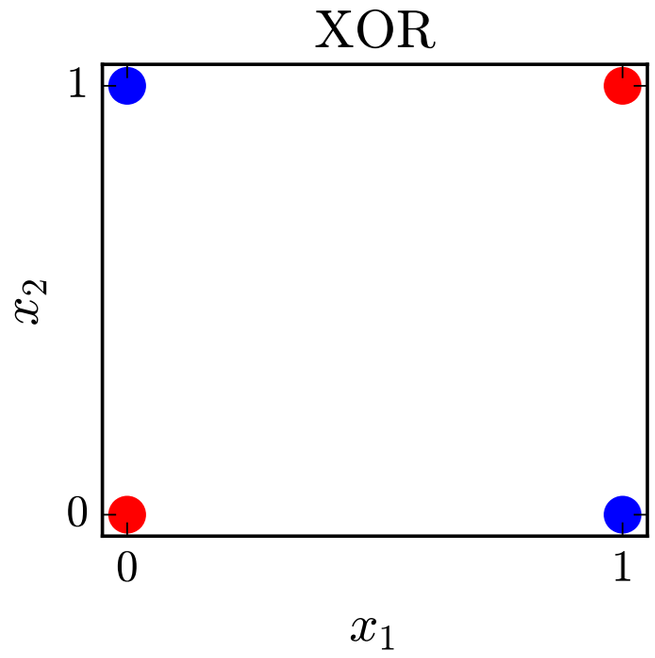
AND

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

$a_0 = -10, \quad a_1 = 15, \quad a_2 = 15$ $\qquad$ $a_0 = -20, \quad a_1 = 15, \quad a_2 = 15$

XOR

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



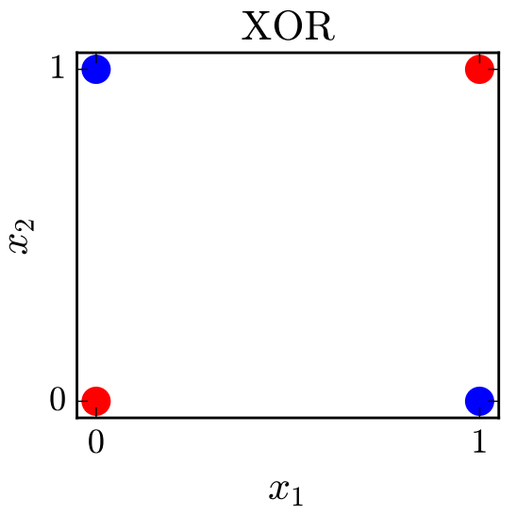
This system cannot produce XOR

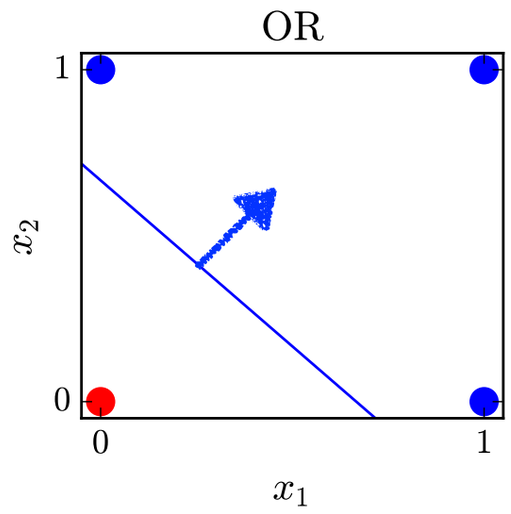
(cannot make a two sided cut)

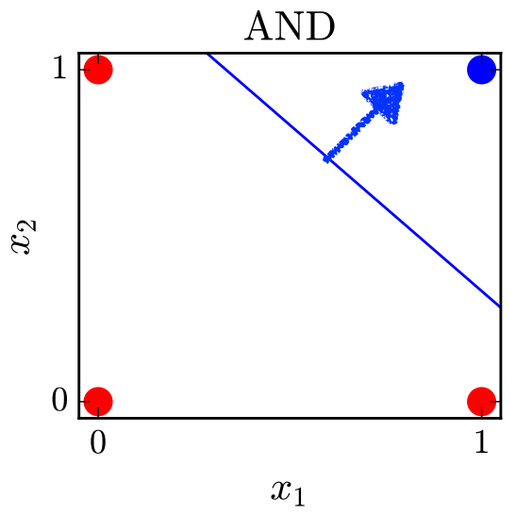
Neural Networks

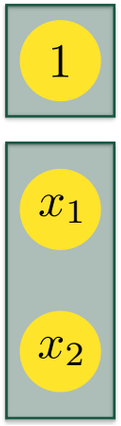
XOR

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





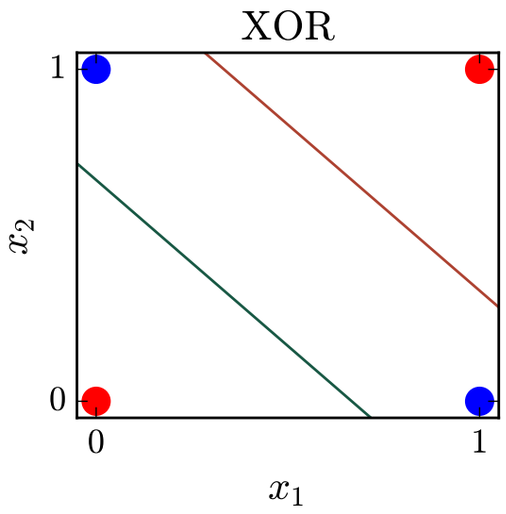


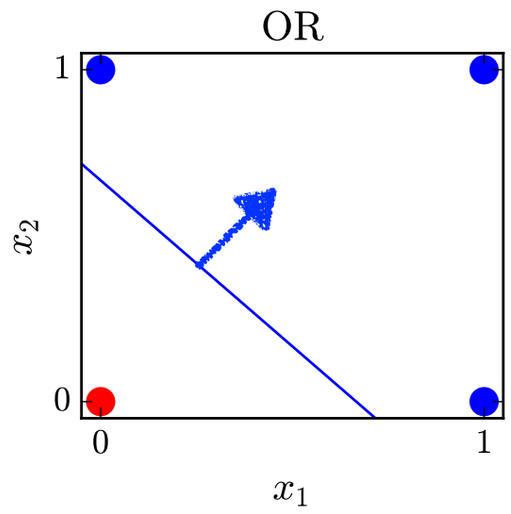


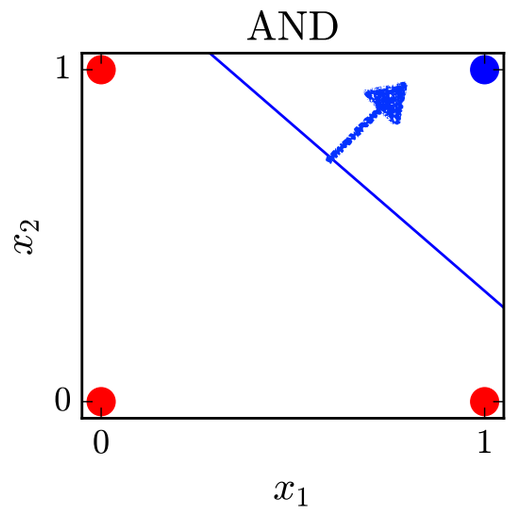
Neural Networks

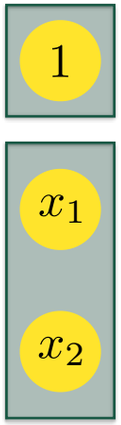
XOR

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





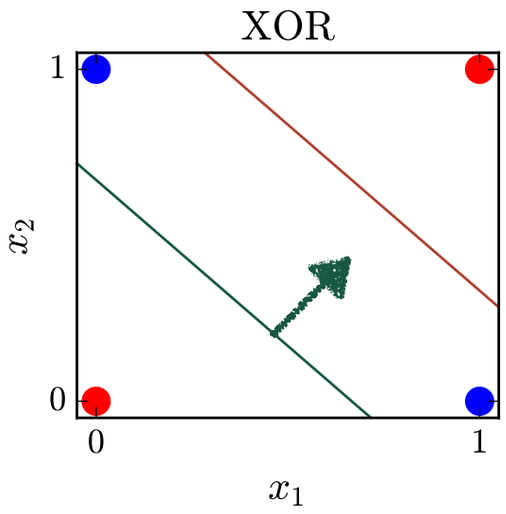


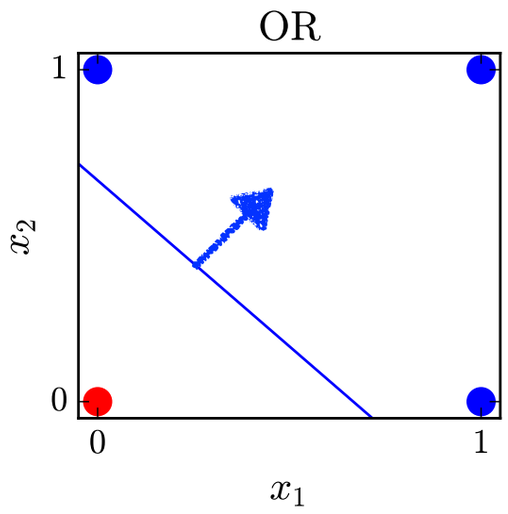


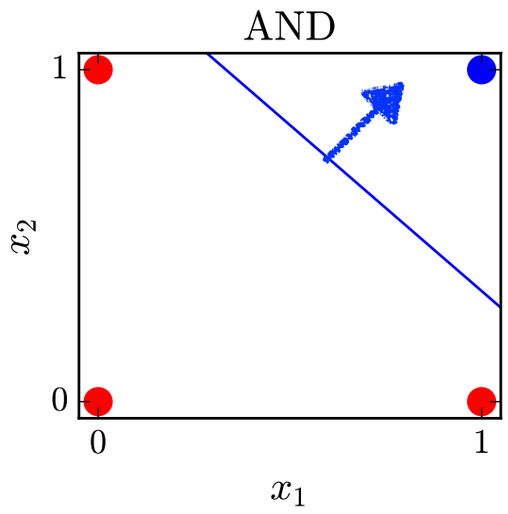
Neural Networks

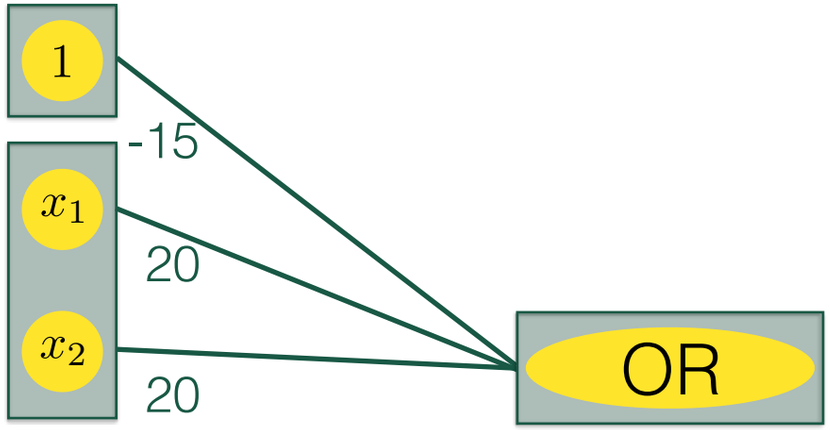
XOR

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





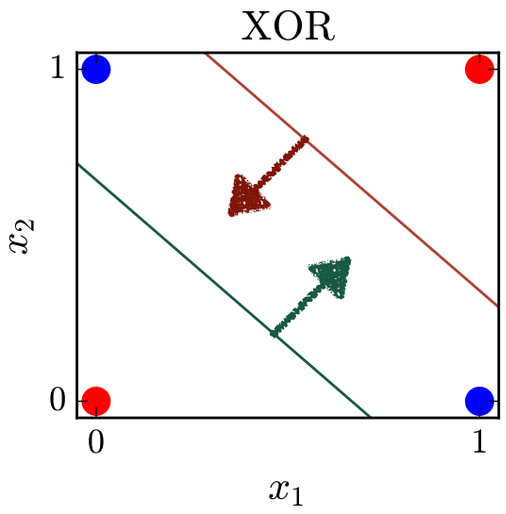


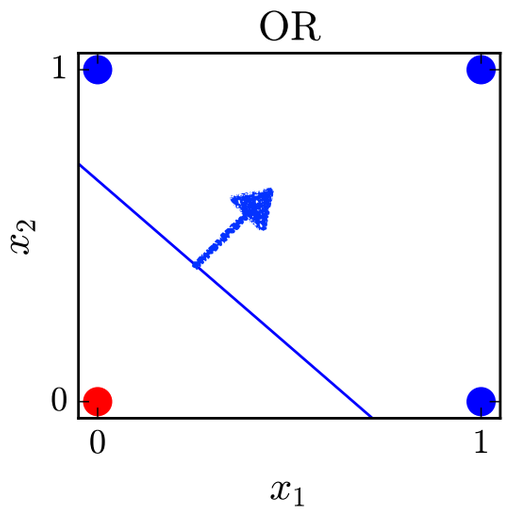


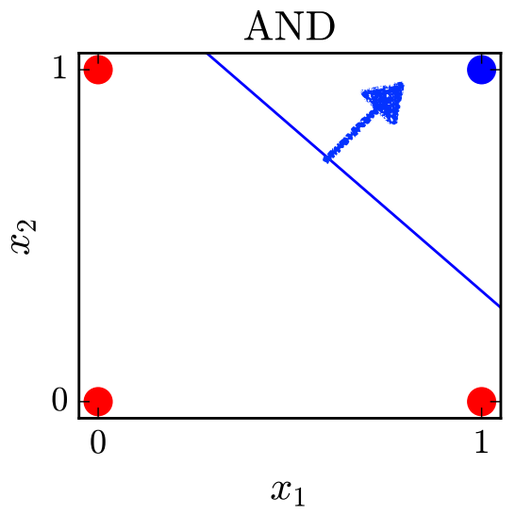
Neural Networks

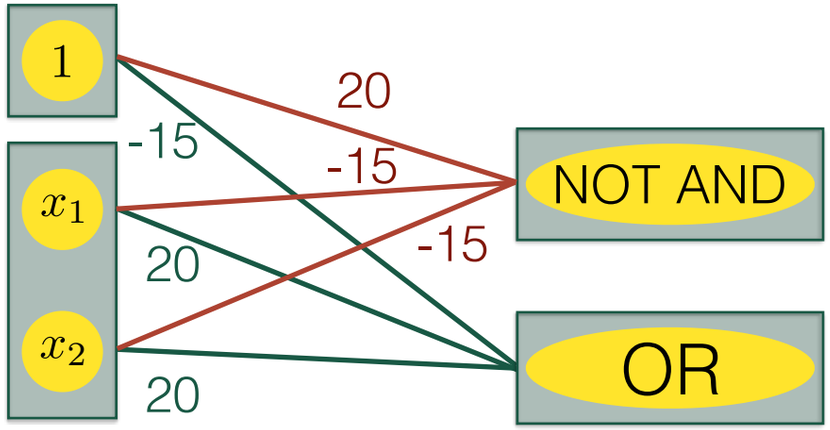
XOR

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





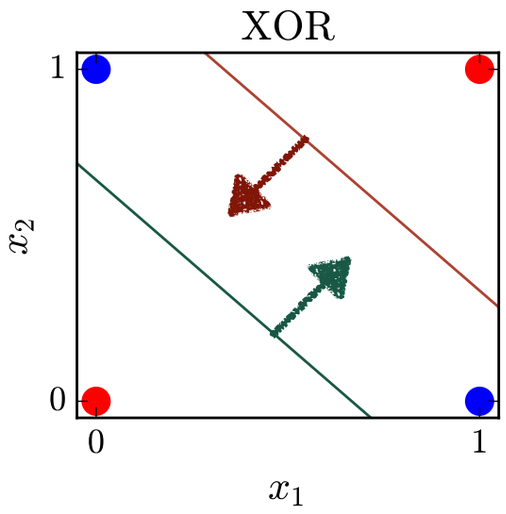


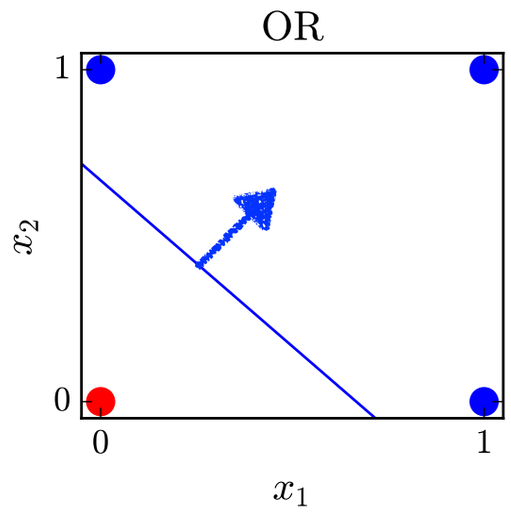


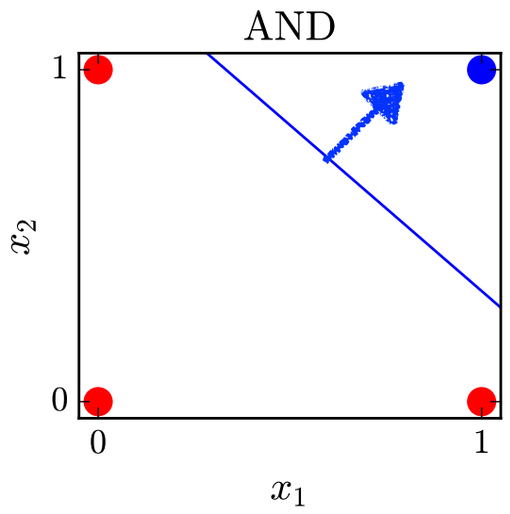
Neural Networks

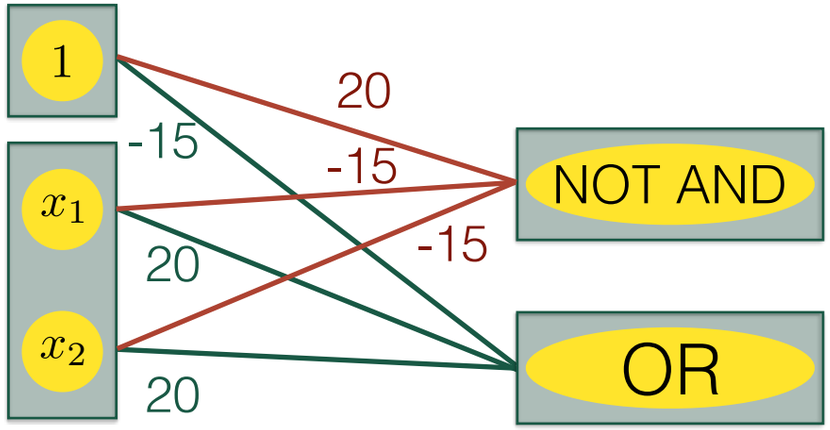
XOR

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







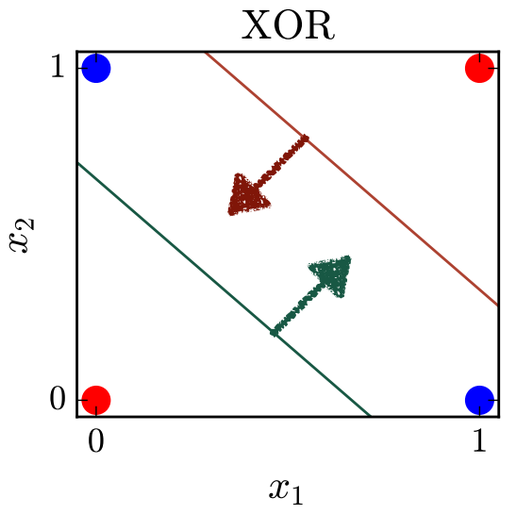


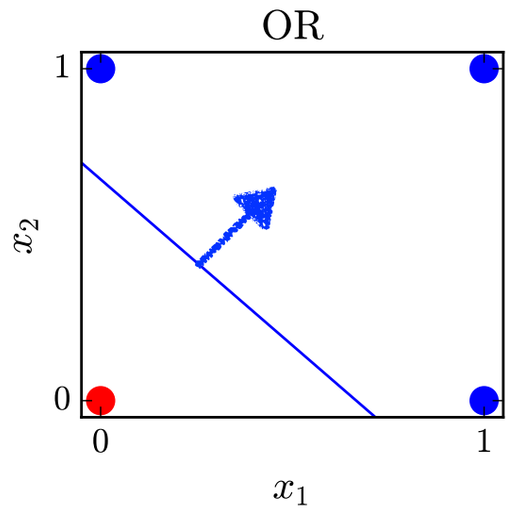
x_1	x_2	OR	NOT AND	XOR
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

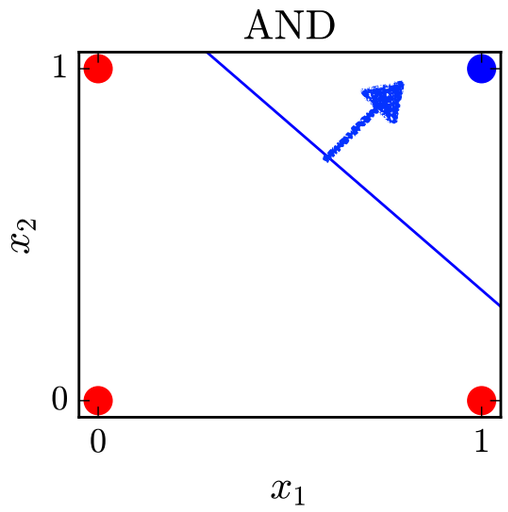
Neural Networks

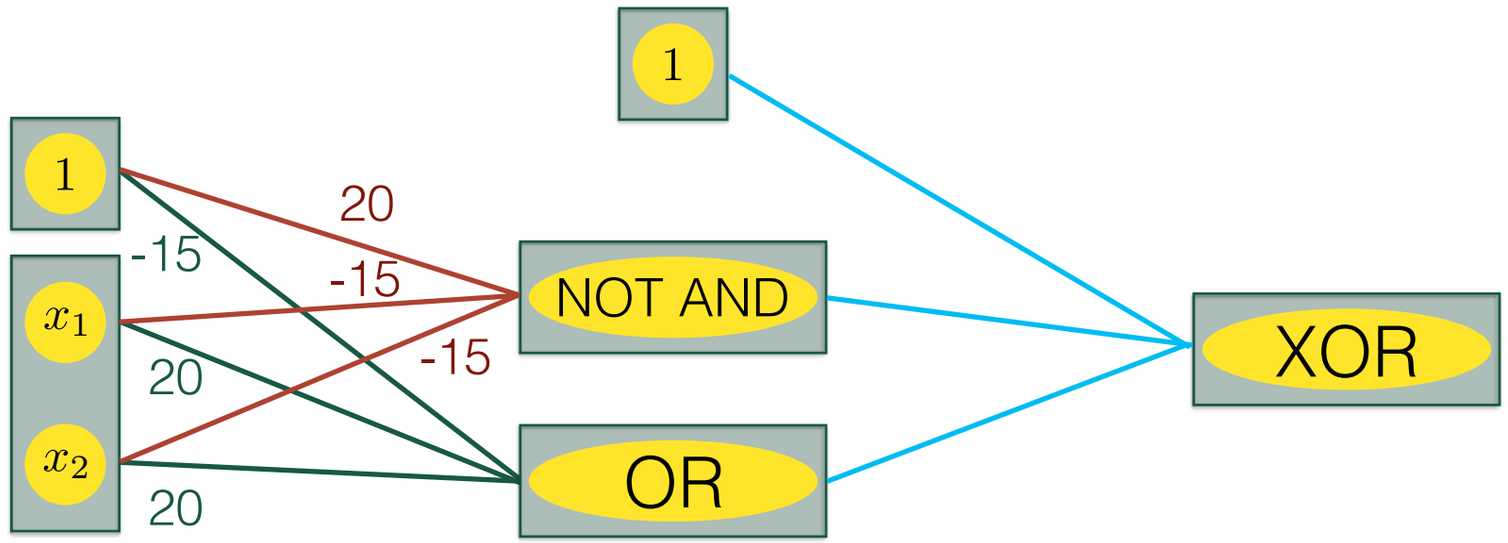
XOR

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





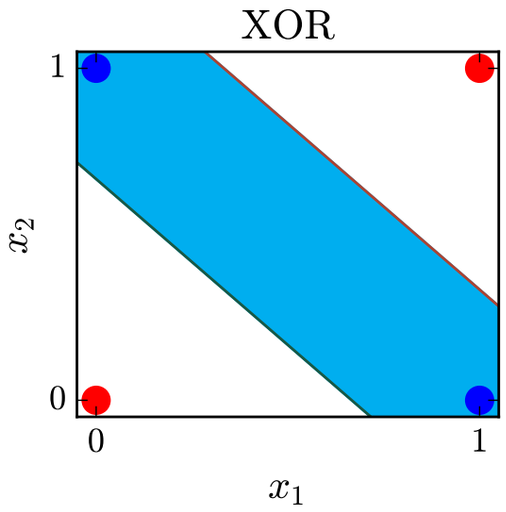


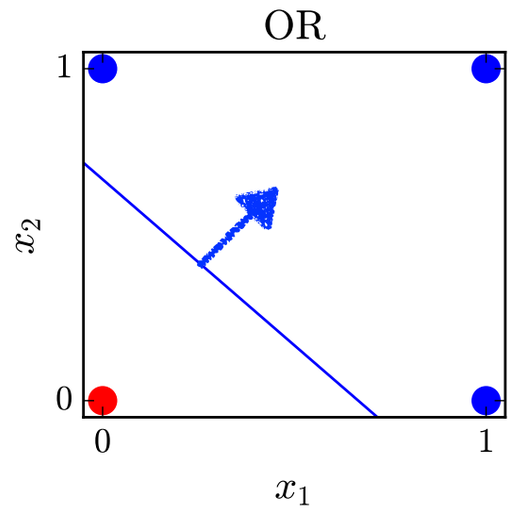


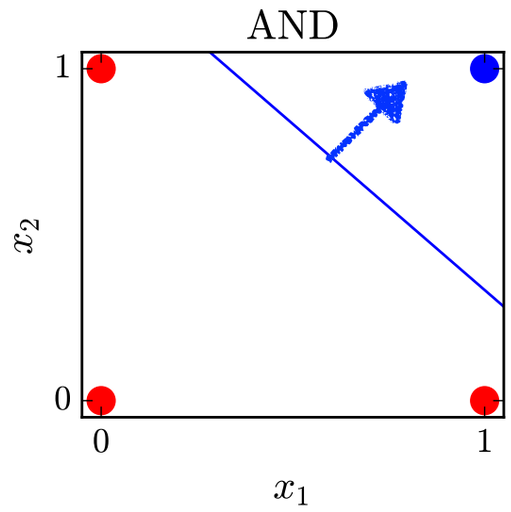
x_1	x_2	OR	NOT AND	XOR
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

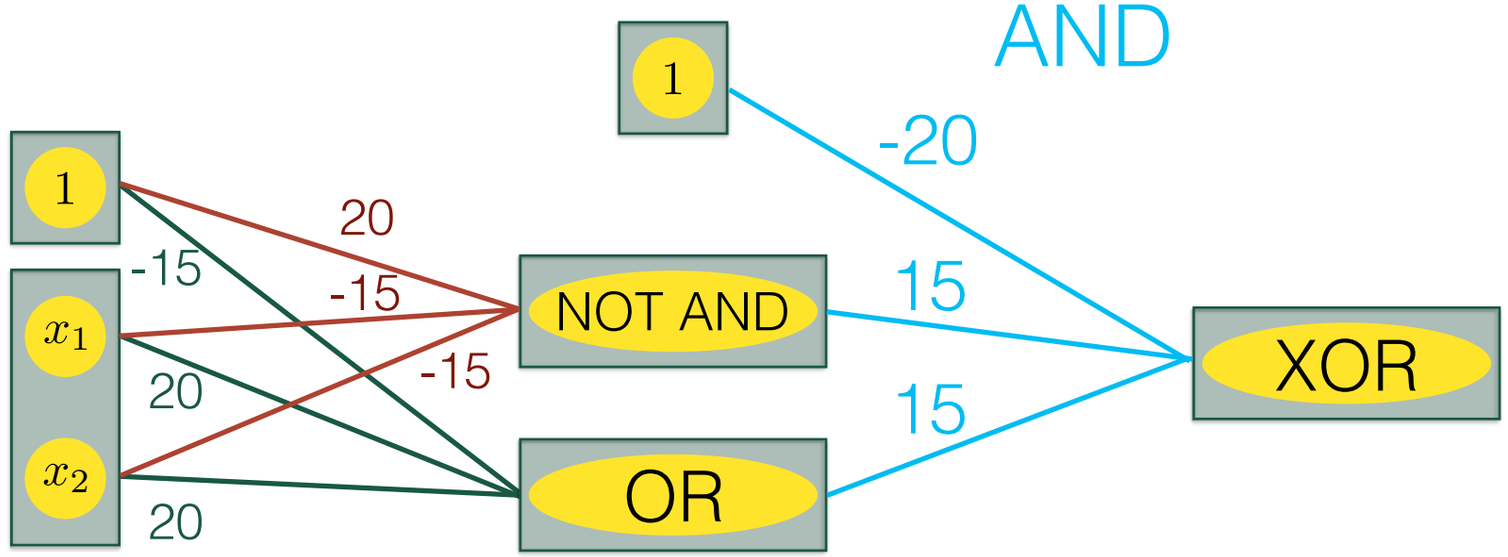
Neural Networks

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







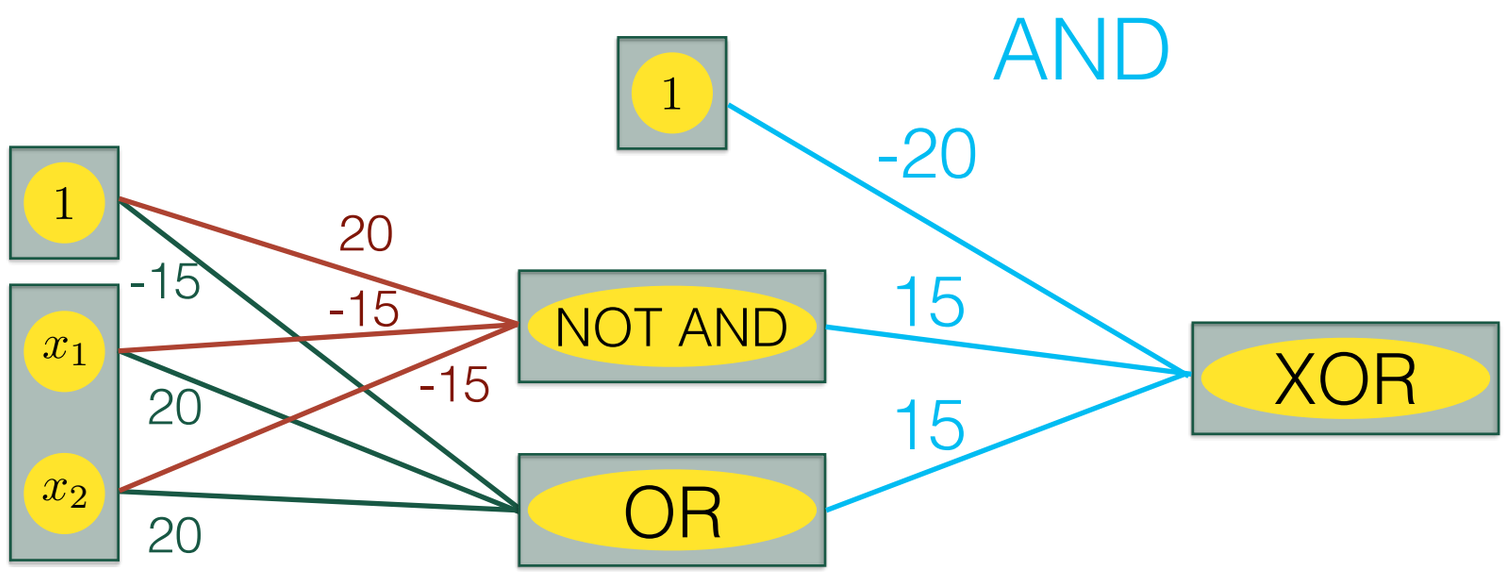


x_1	x_2	OR	NOT AND	XOR
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

Neural Networks

XOR

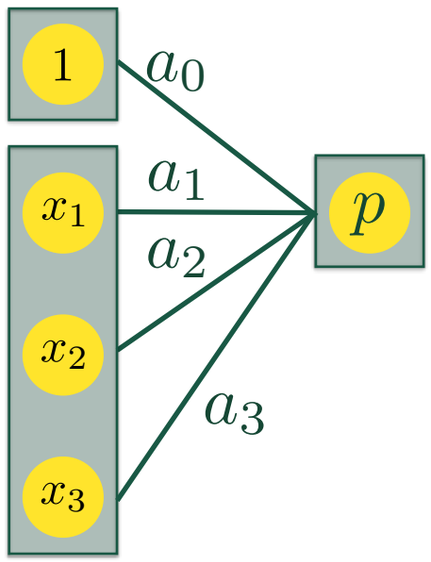
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

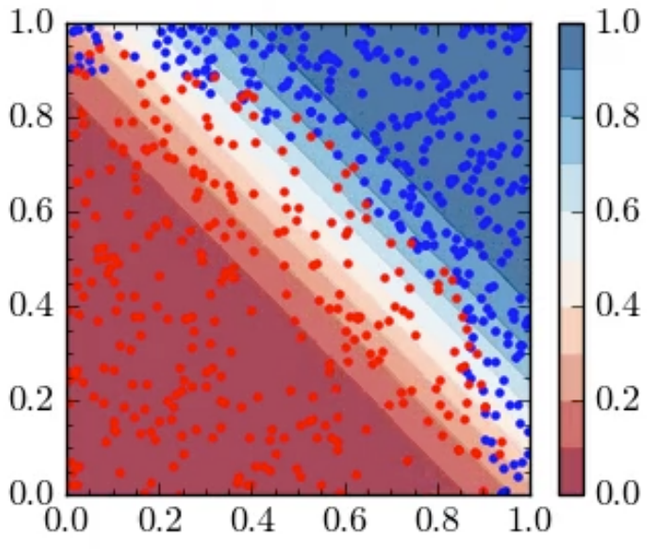


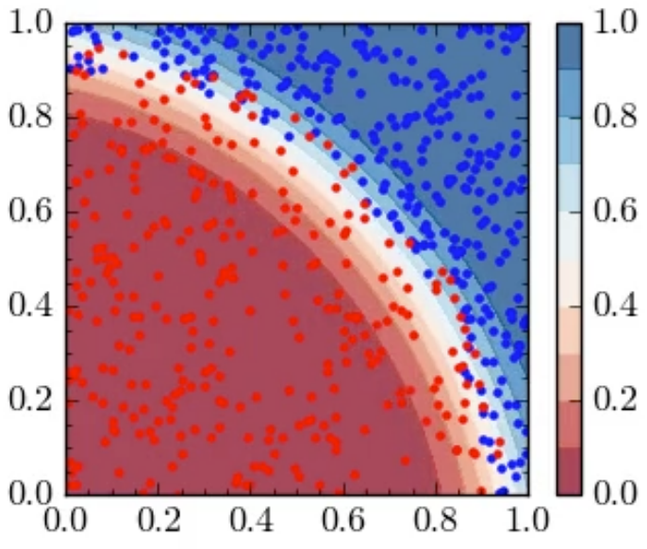
Simple example showing that neural network can access ‘high-level’ functions

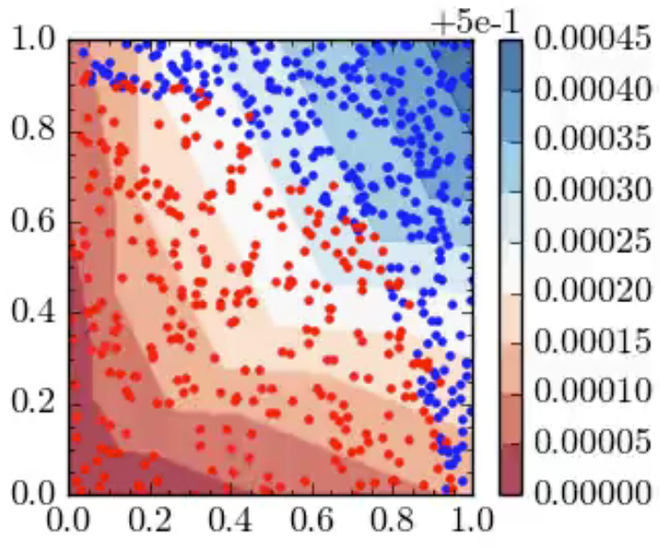
Neural Networks

- Don't add more inputs, let machine find own shape
- Ability to learn 'any' function
- More nodes/hidden layers allows for more complex features



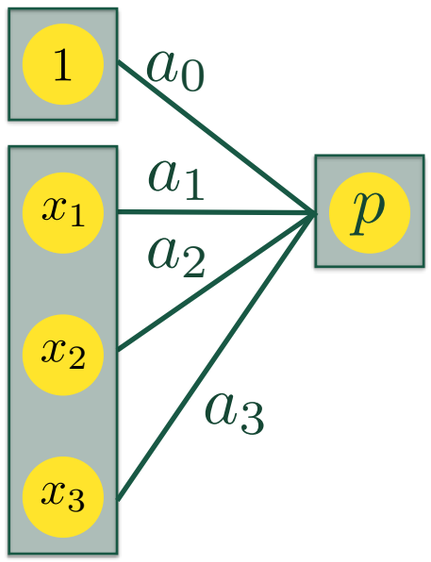


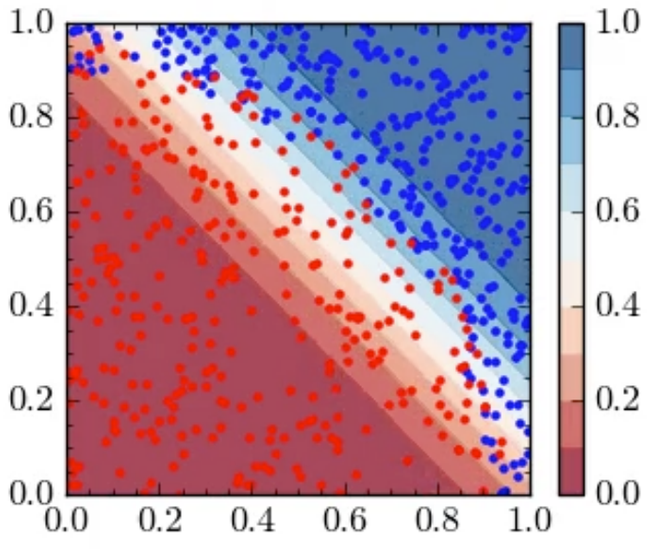


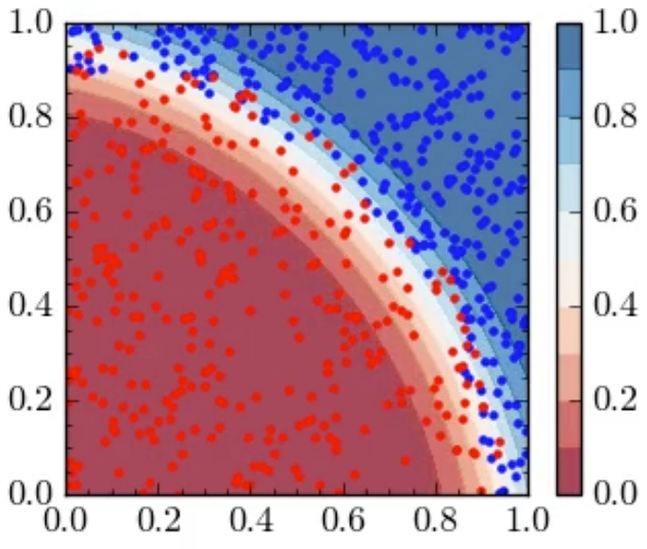


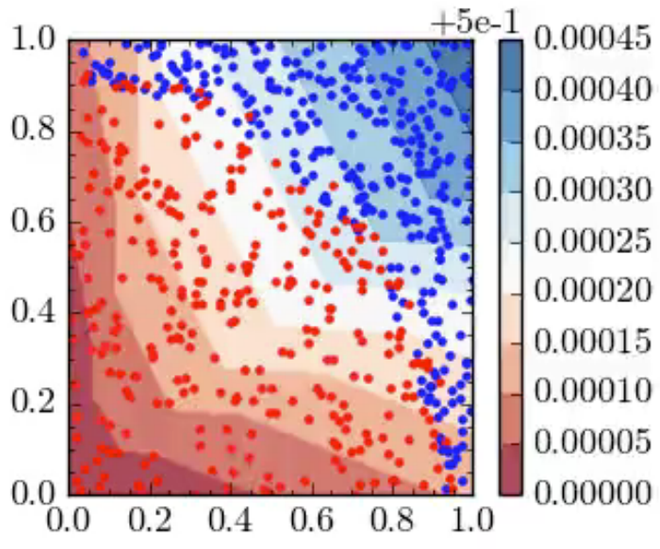
Neural Networks

- Don't add more inputs, let machine find own shape
- Ability to learn 'any' function
- More nodes/hidden layers allows for more complex features



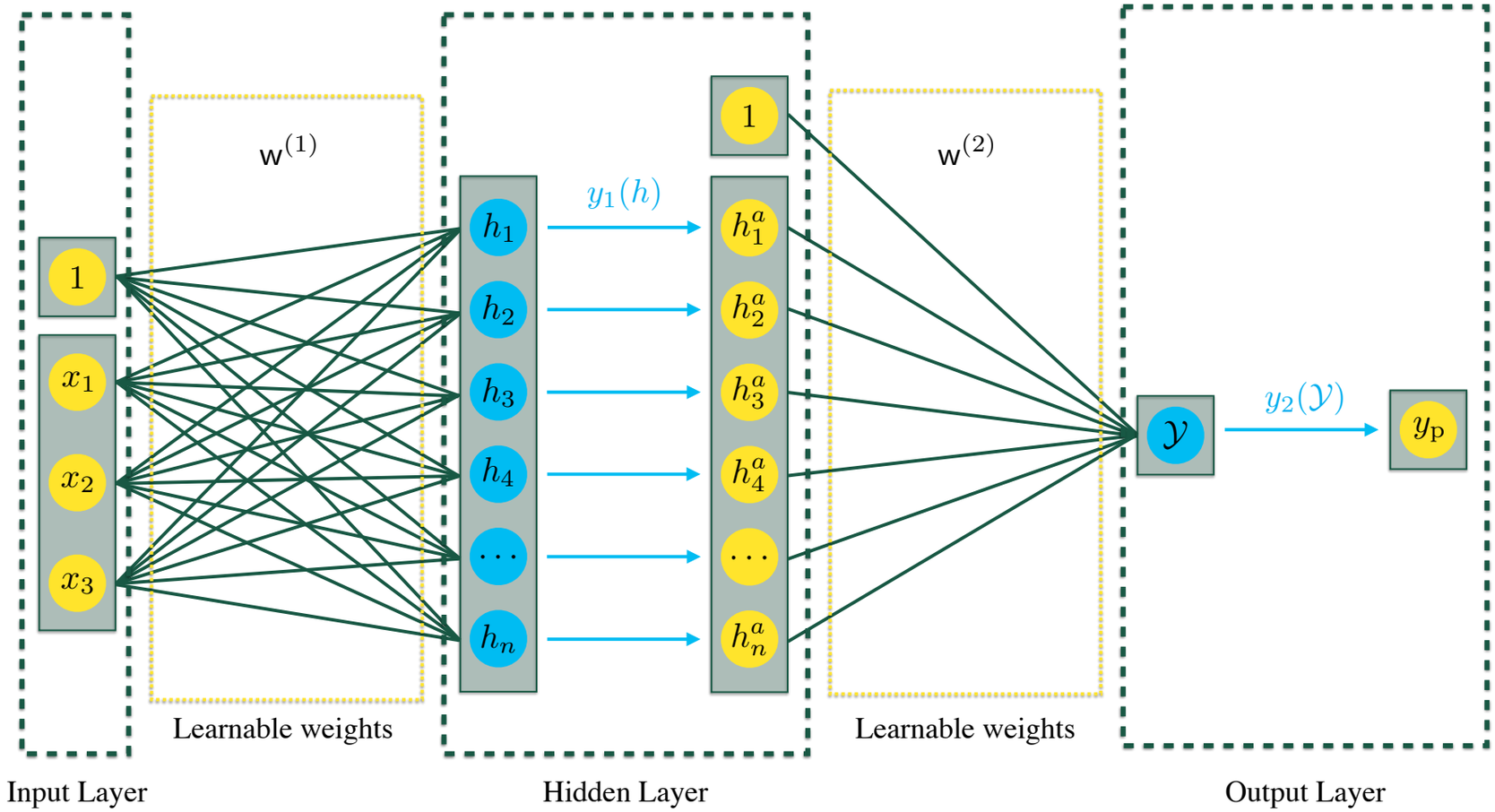


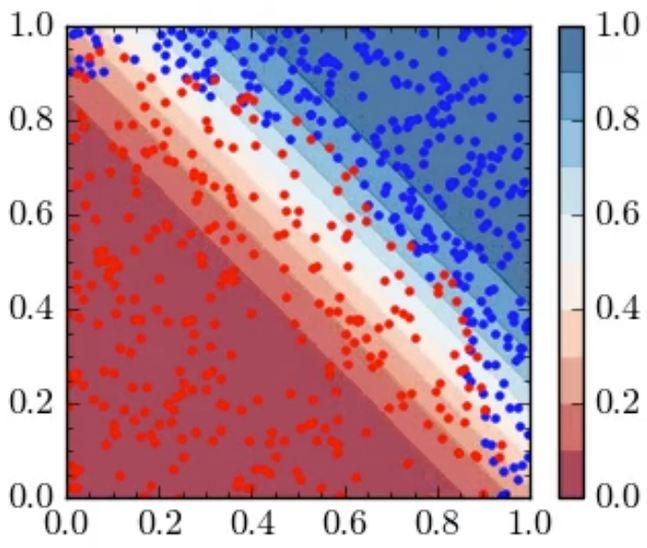


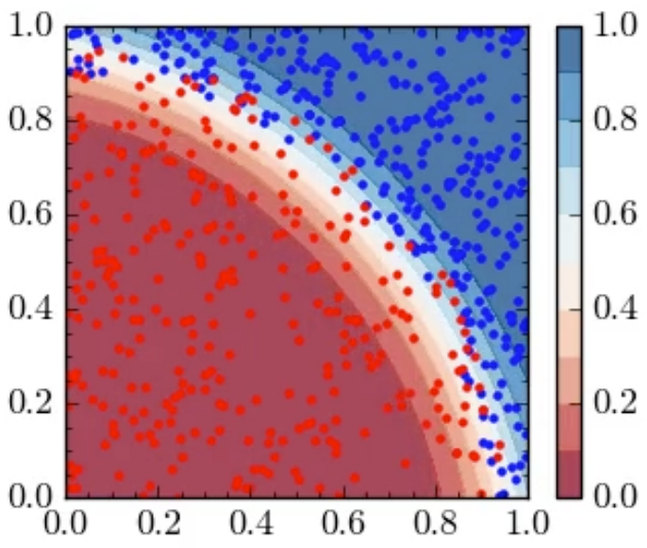


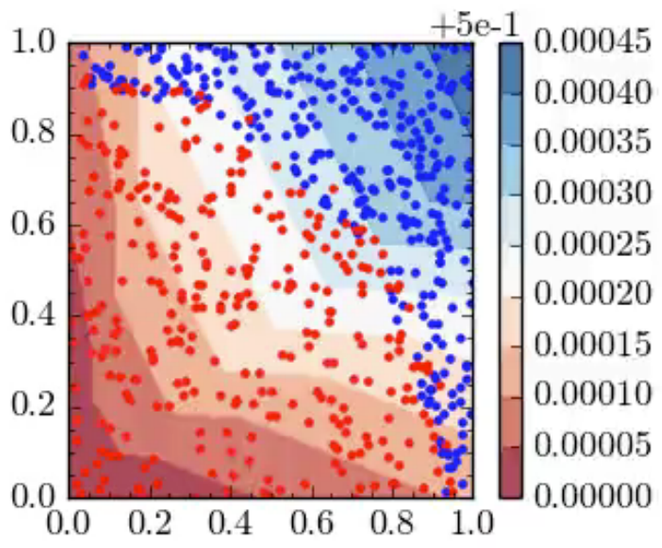
Neural Networks

- Don’t add more inputs, let machine find own shape
- Ability to learn ‘any’ function
- More nodes/hidden layers allows for more complex features



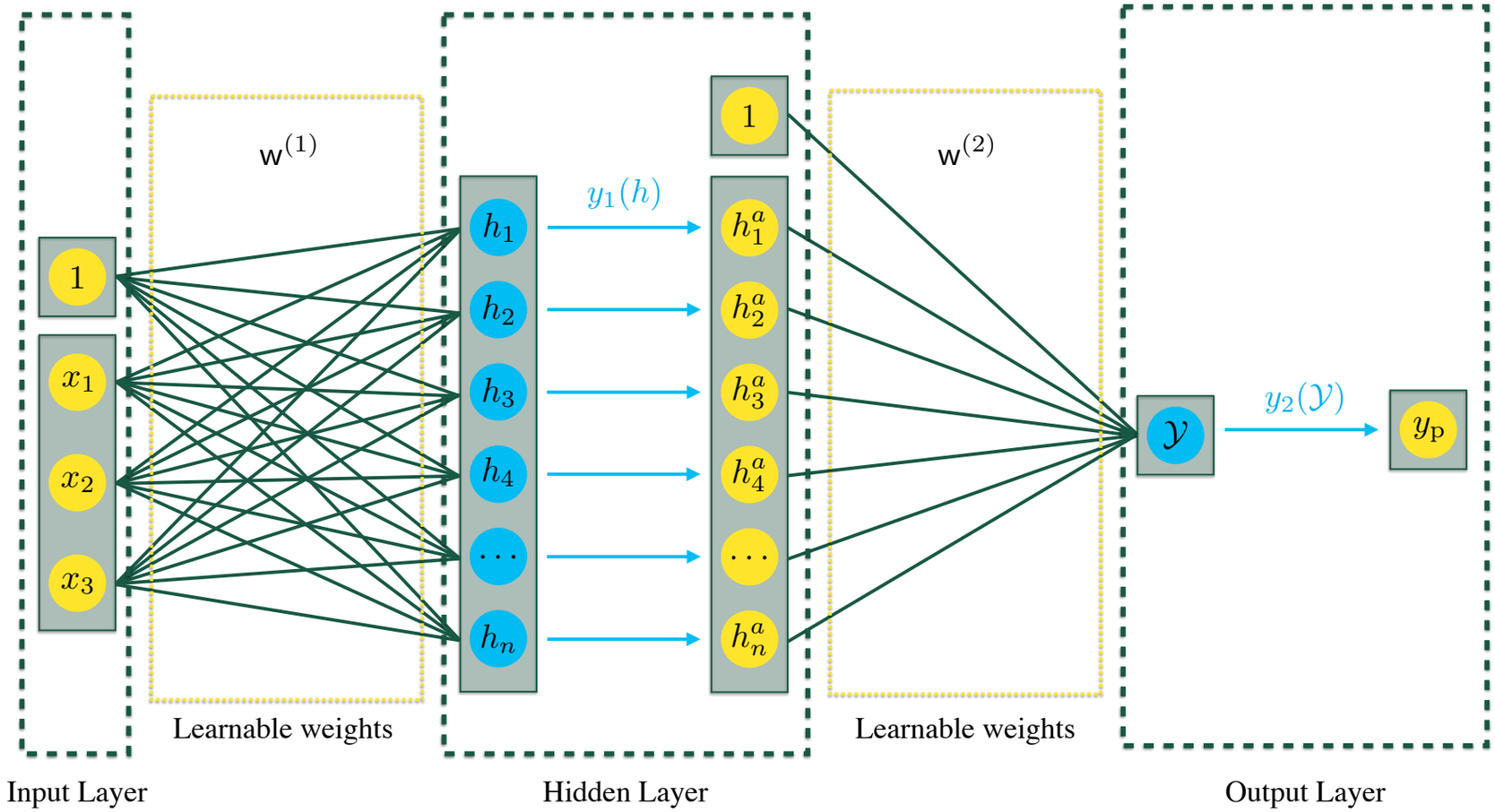


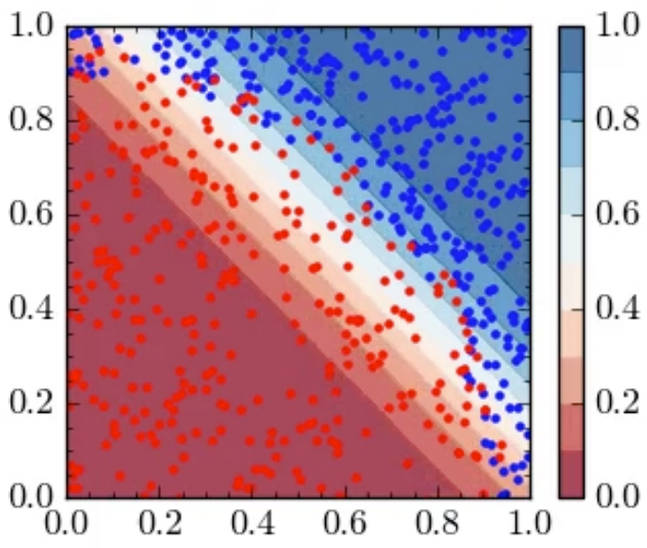


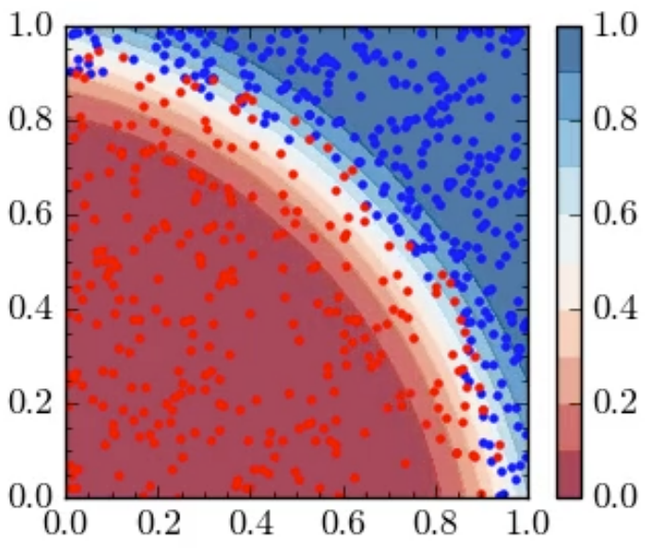


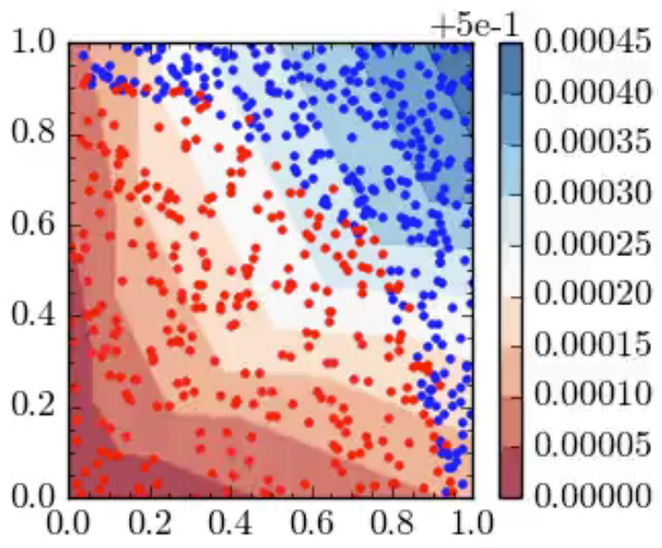
Neural Networks

- Don’t add more inputs, let machine find own shape
- Ability to learn ‘any’ function
- More nodes/hidden layers allows for more complex features



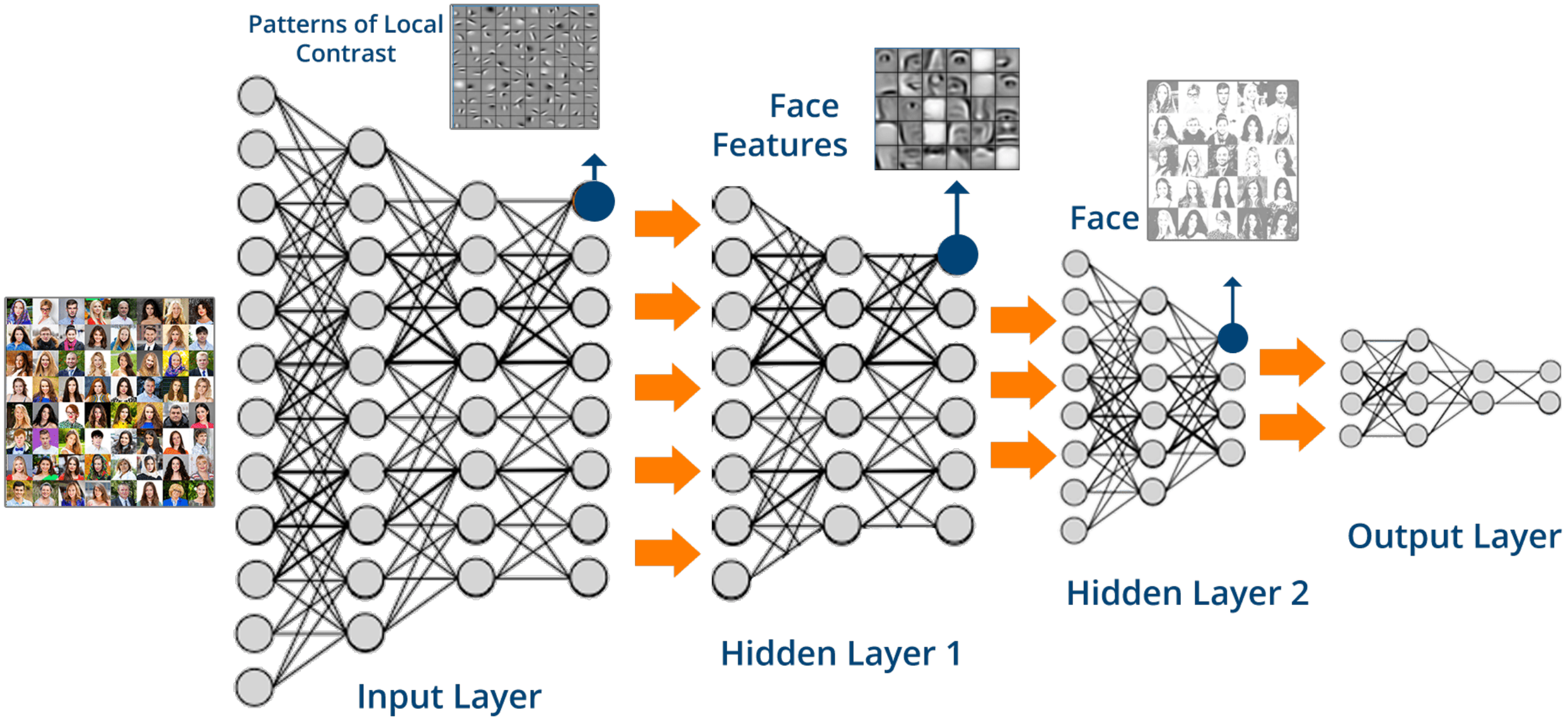




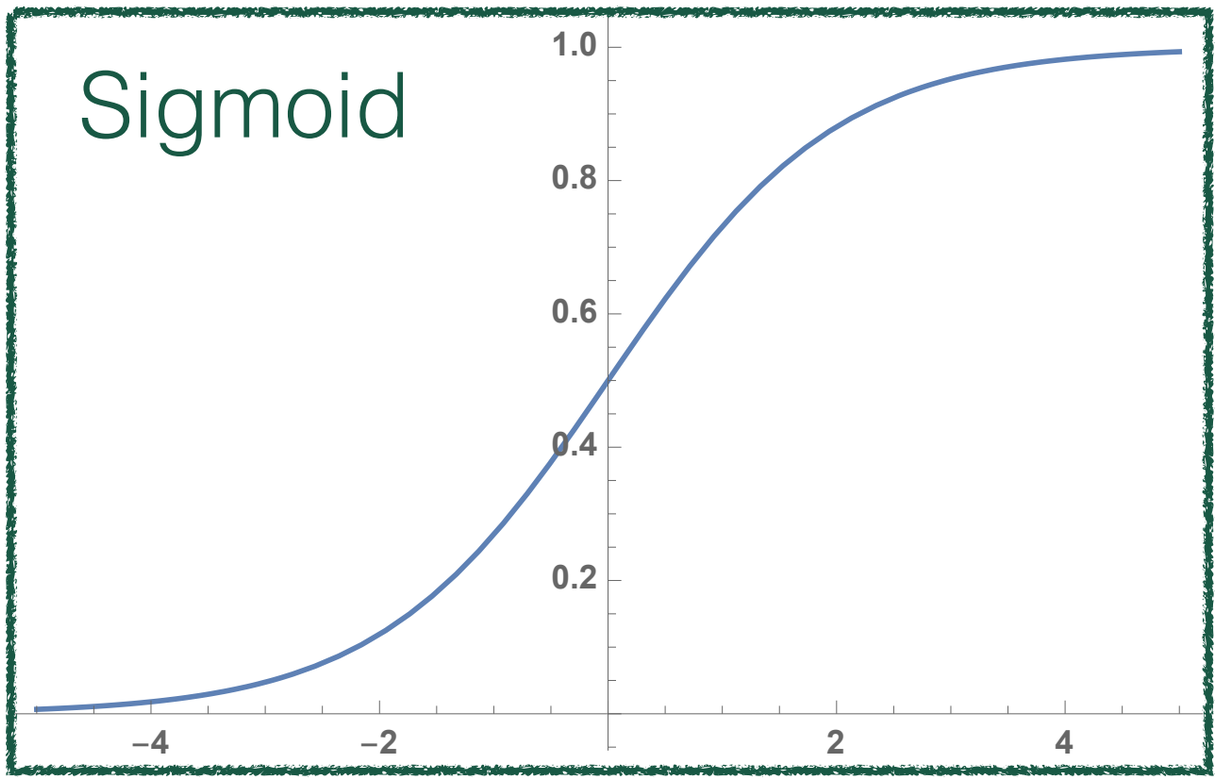


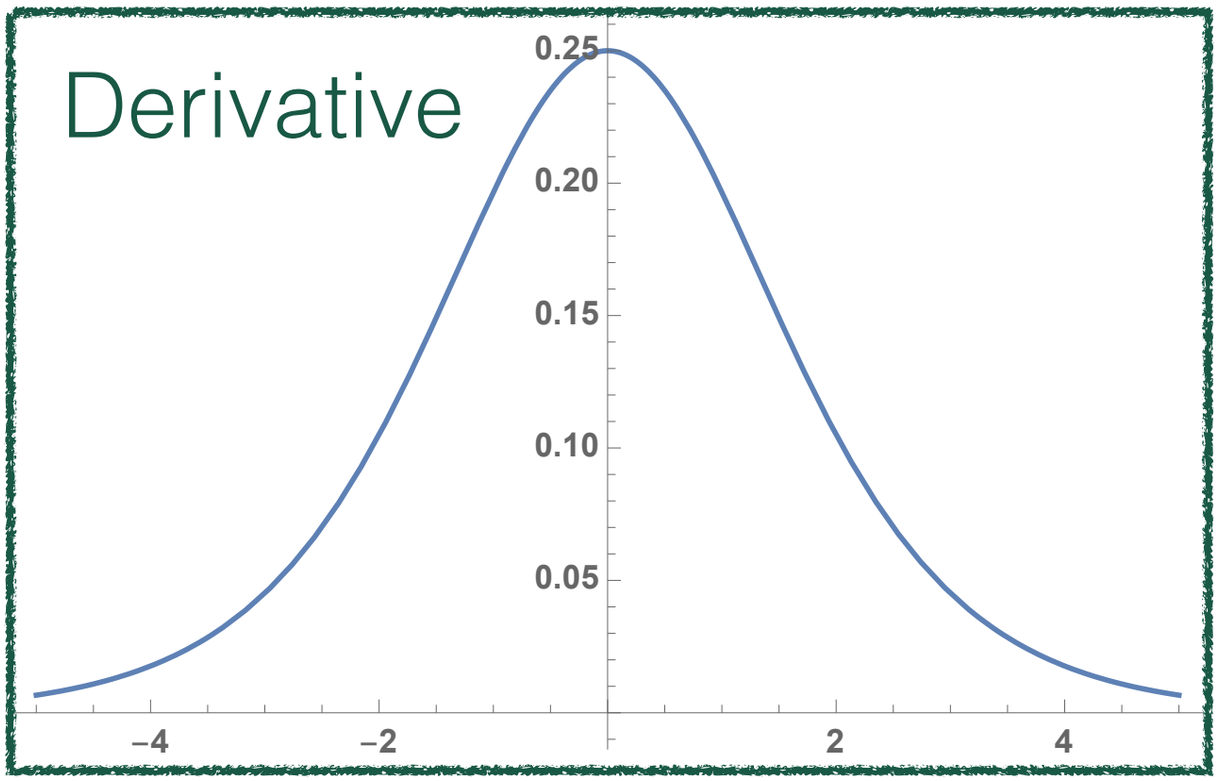
Deep learning uses raw inputs, with many hidden layers to let the machine come up with its own features.

How deep can we go before running into problems?


Patterns of Local Contrast
Face Features
Face
Input Layer
Hidden Layer 1
Hidden Layer 2
Output Layer

Disappearing Gradient





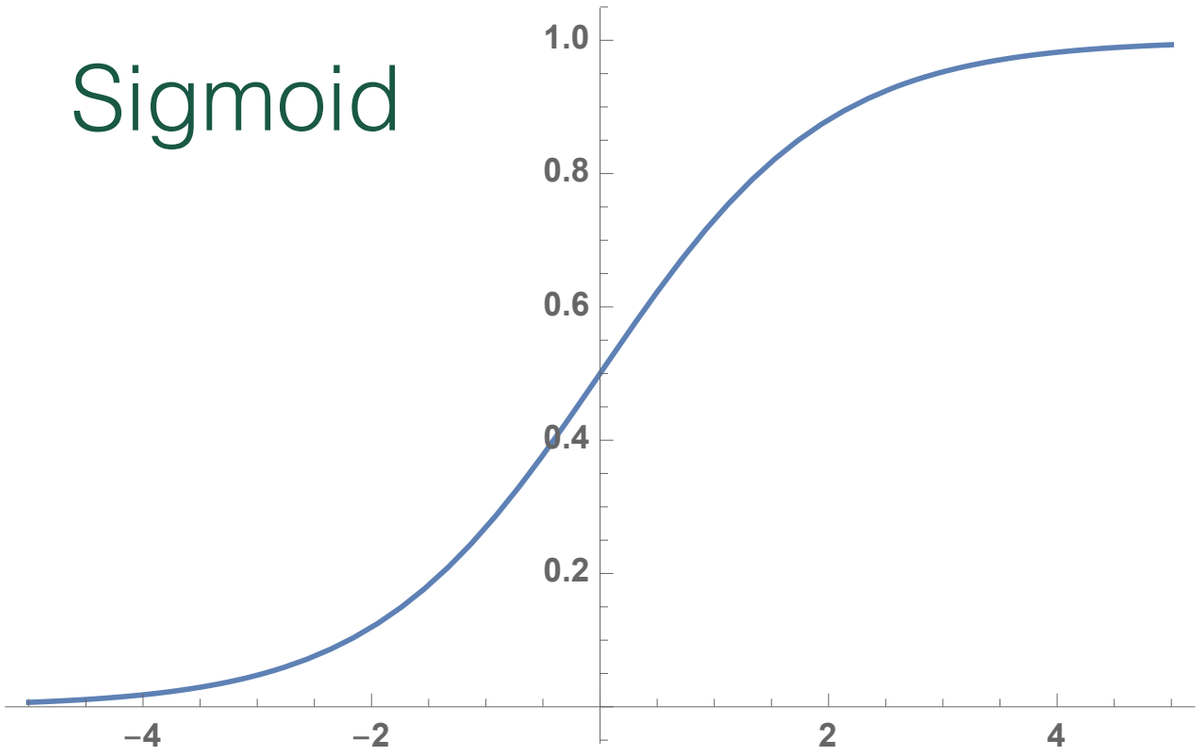
Chain rule for gradient of network involves multiple factors of the derivative multiplied together

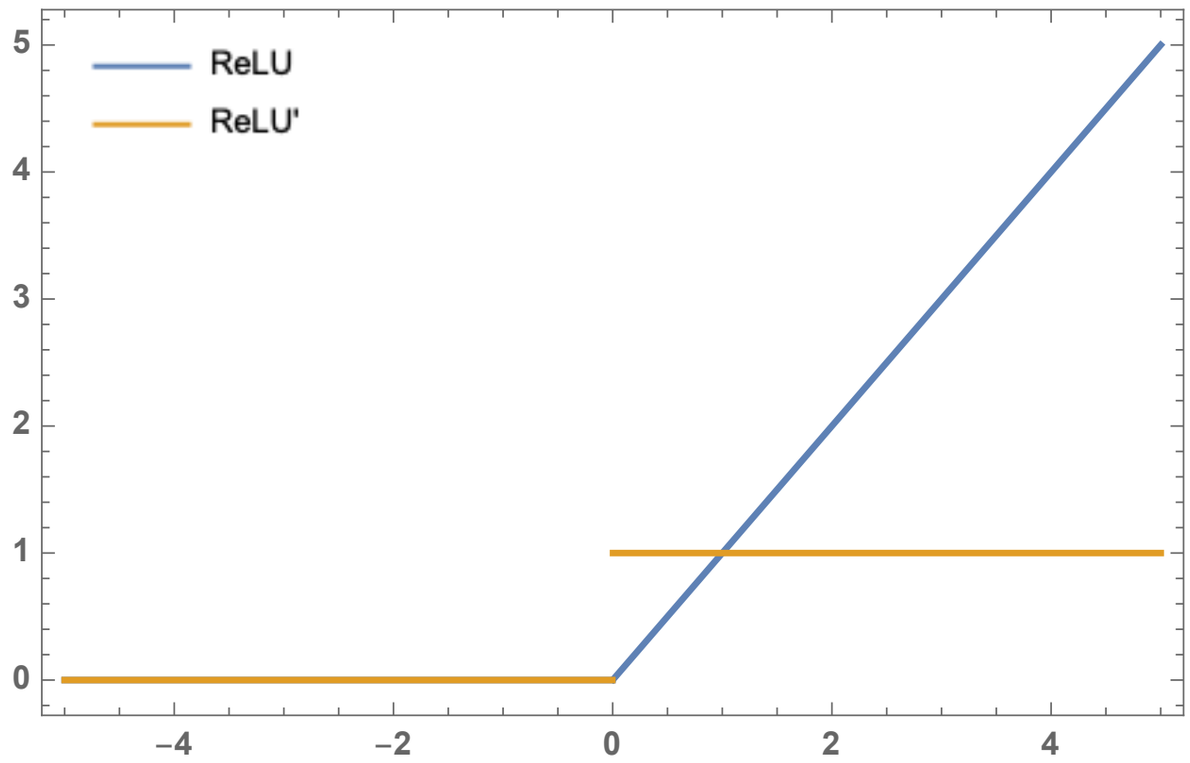
$$(0.25)^4 = 0.0039$$

Deep networks with Sigmoid activations have exponentially hard time training early layers

Disappearing Gradient

Sigmoid





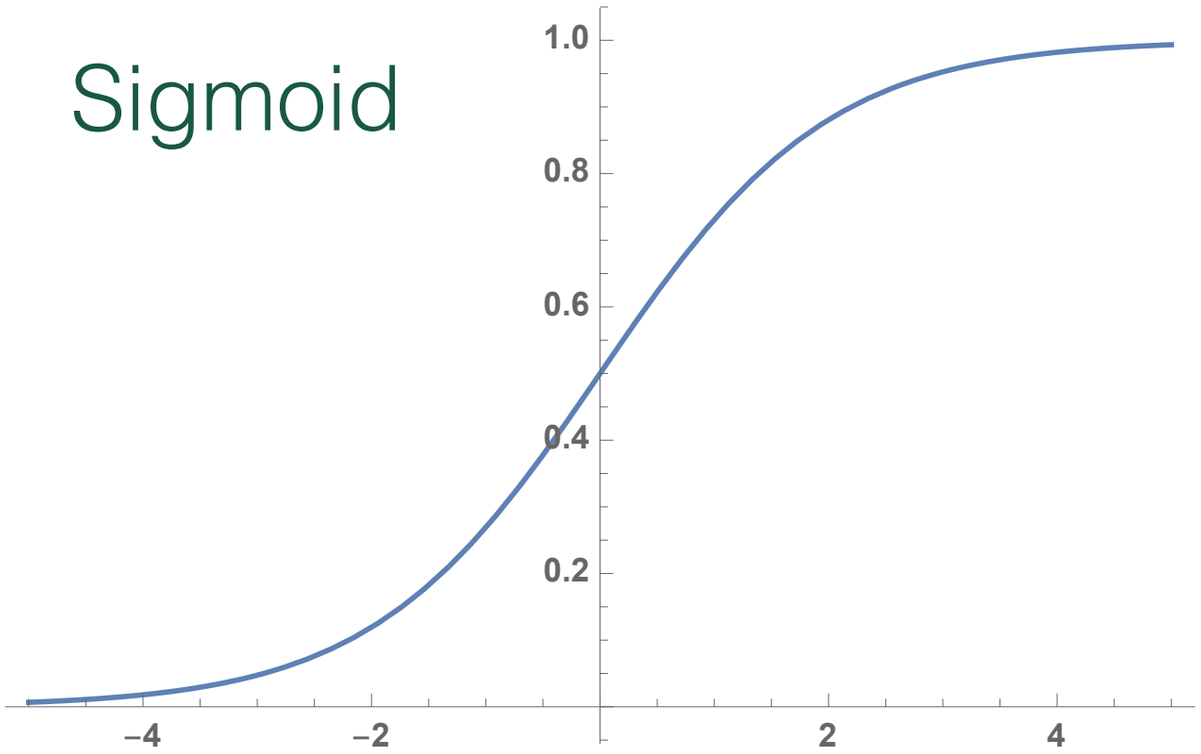
Using the Rectified Linear Unit (ReLU) solves this problem.
ReLU(x) = {0 if x <=0, x if x >0}

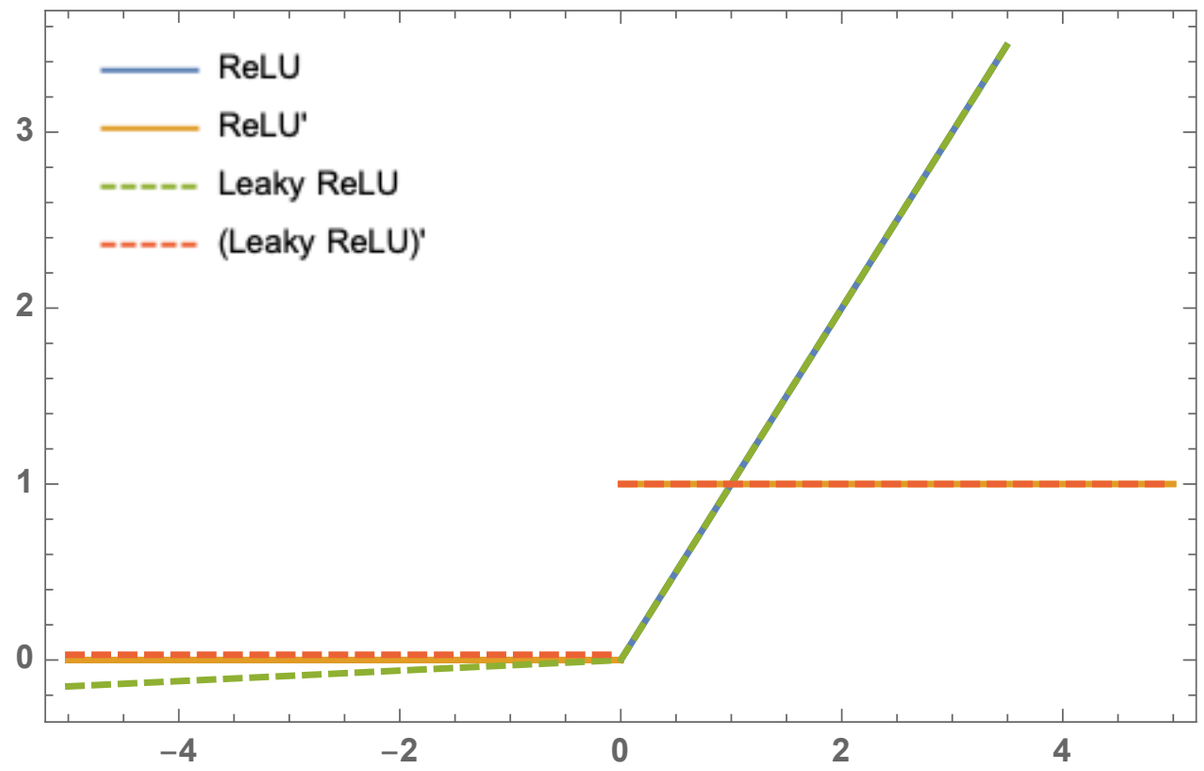
Still has nonlinearity which allows network to learn complicated patterns

Nodes can die (derivative always 0 so cannot update)

Disappearing Gradient

Sigmoid





Leaky Rectified Linear Unit (LeakyReLU) solves this problem.

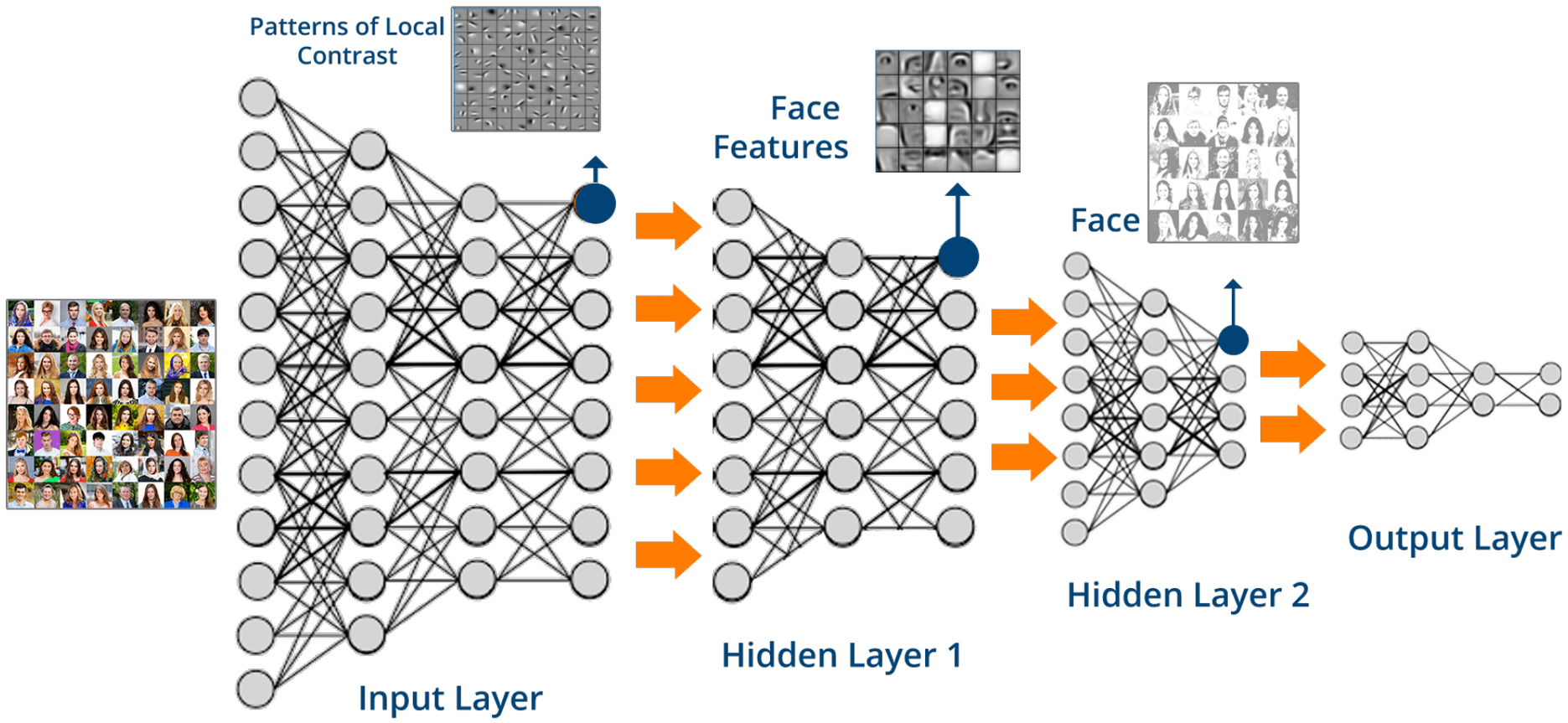
LeakyReLU(x) = {alpha*x if x <=0, x if x >0}

I have never had to use this in practice

Transfer Learning

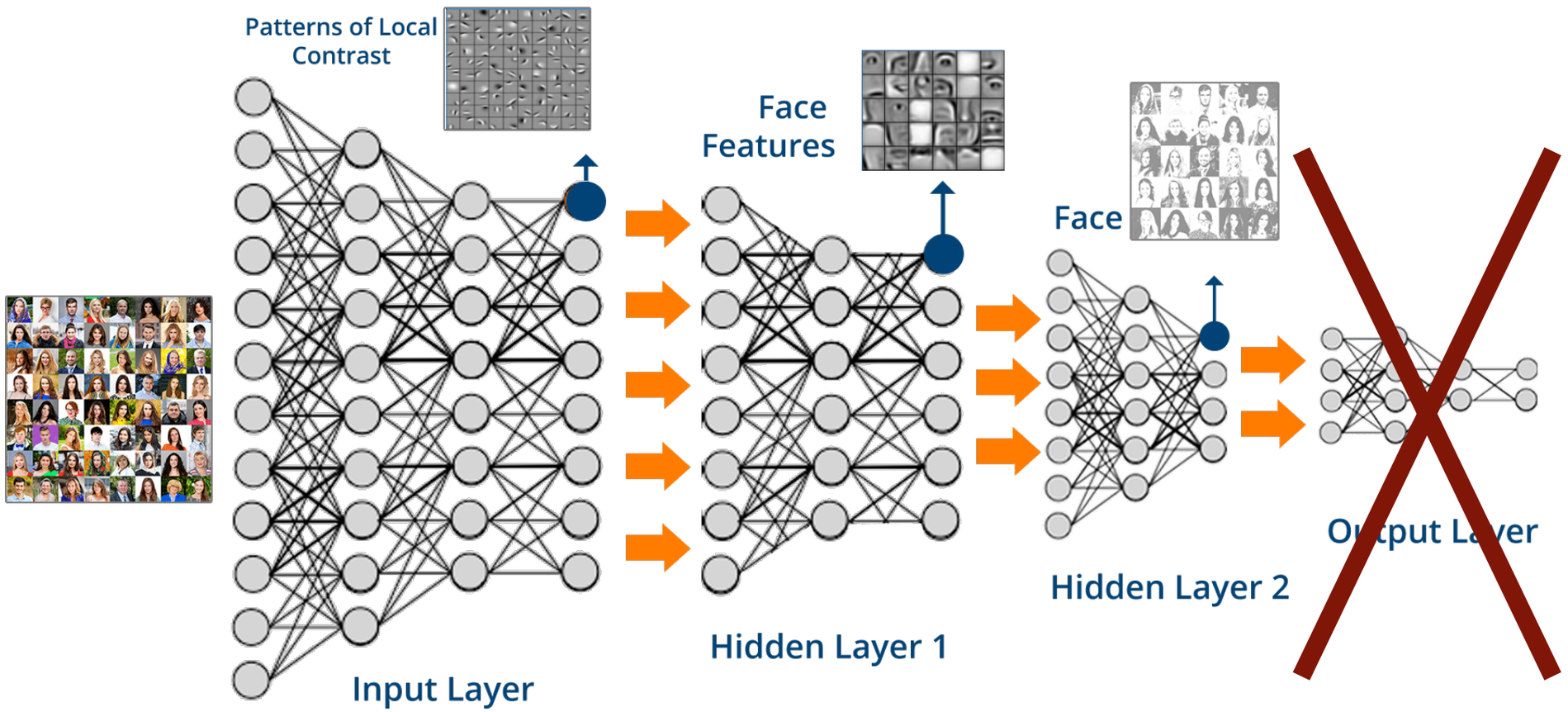
Pre-trained networks for image classification are readily available

https://keras.io/applications/



What if we want to do something with images, but not classification (using the same categories)?

Transfer Learning

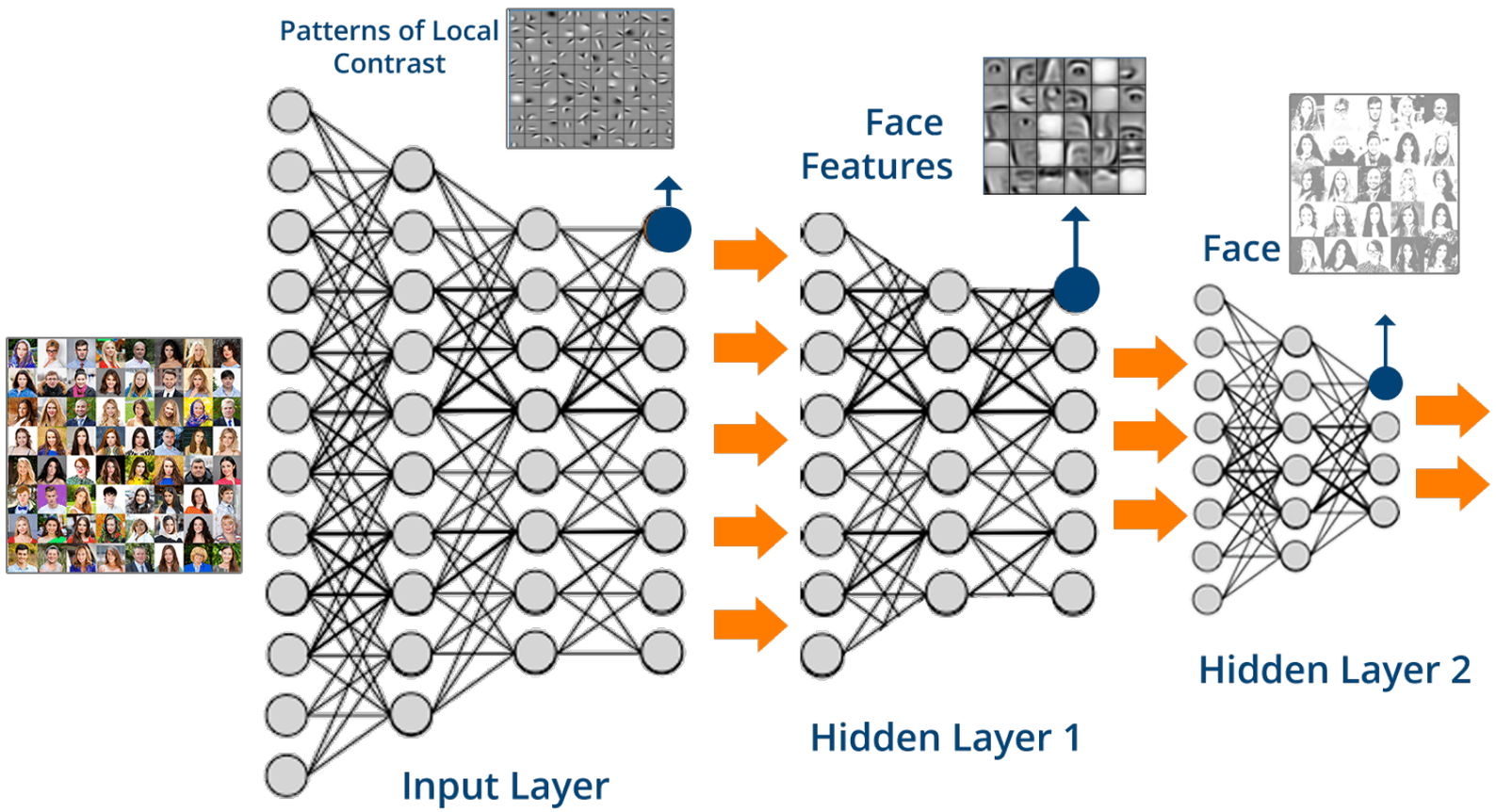


Keep the features

Get rid of the "classification part"

Freeze the layers so that the features don't change

Transfer Learning



New network

"Preprocess" our data

New network has small number of trainable weights, can train with limited resources

Transfer Learning

1. Deep learning lets the network learn features, rather than human engineering
2. Adding more layers allows for more complex features
3. Deeper networks use different activations
4. Transfer learning uses features learned from one task on a related task

Example

https://github.com/AlexEMG/DeepLabCut

Exercise Notebook

Train a network on simulated data, but want to apply it to real data