ECE 884 Deep Learning

Lecture 10: Neural Networks

02/18/2021

Review of last lecture

- Training
  - GD
  - SGD
  - Momentum, Adam, AdamW

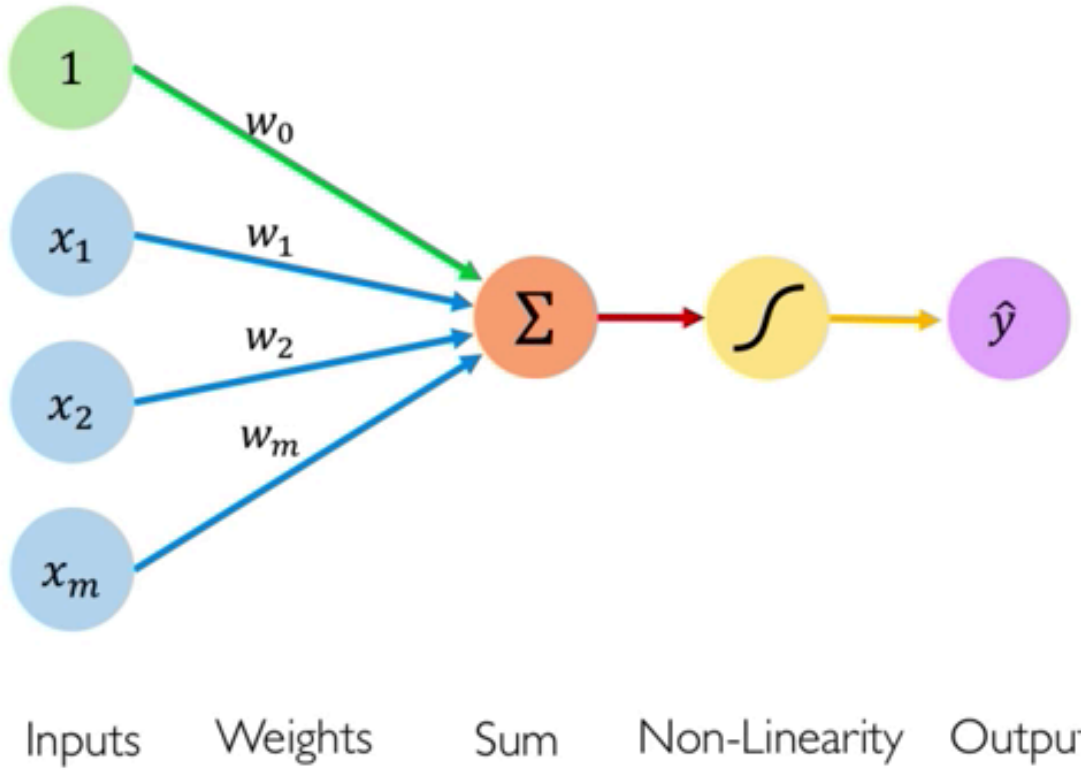
Today’s lecture

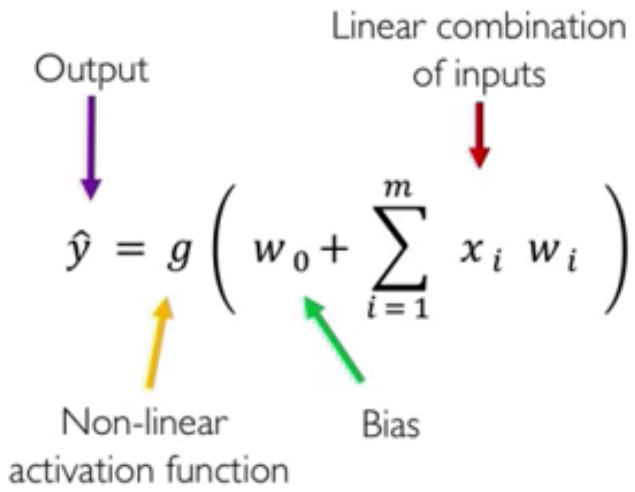
- (Deep) Neural Networks
  - Perceptron
  - Multilayer Perceptron (MLP)
  - Activation Functions
  - Why Neural Networks are powerful

Perceptron

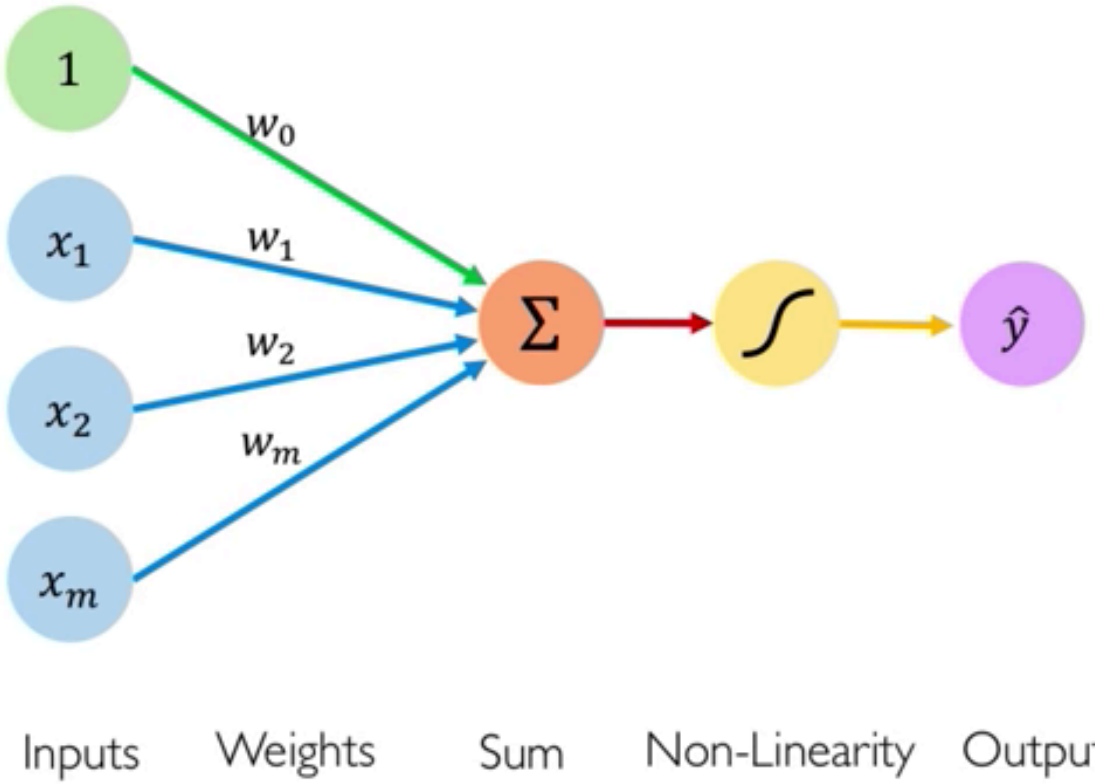
The structural building block of deep learning

Perceptron


1
x_1
x_2
x_m
w_0
w_1
w_2
w_m
Σ
$\int$
$\hat{y}$
Inputs
Weights
Sum
Non-Linearity
Output


Output
Linear combination of inputs
$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i\ w_i\right)$
Non-linear activation function
Bias

Perceptron

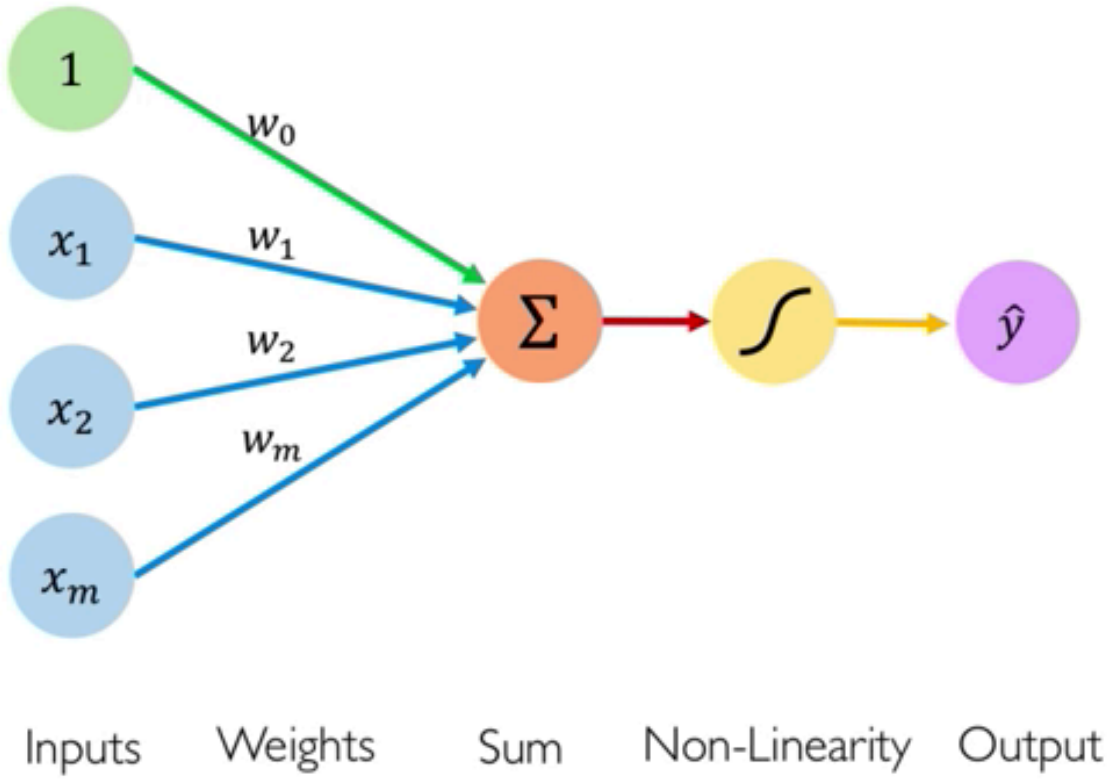


$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i \, w_i \right)$$

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$

where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

Perceptron

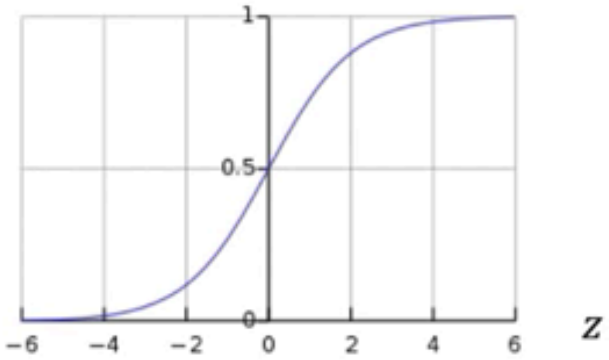


Activation Functions

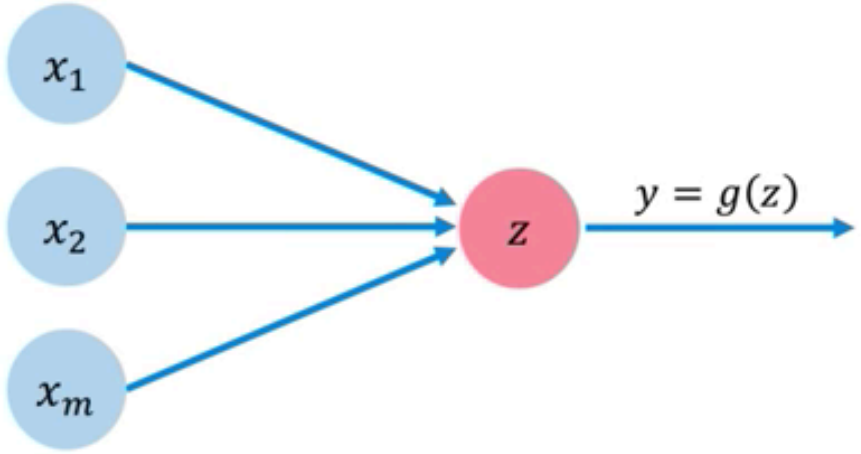
$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

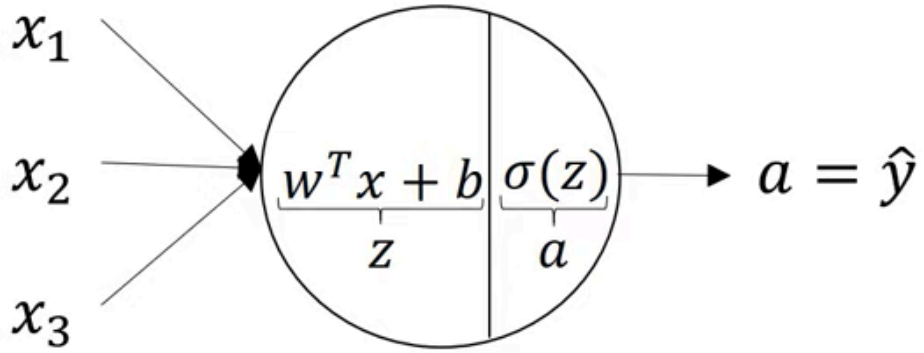
- Example: sigmoid function

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



Building Neural Networks with Perceptron



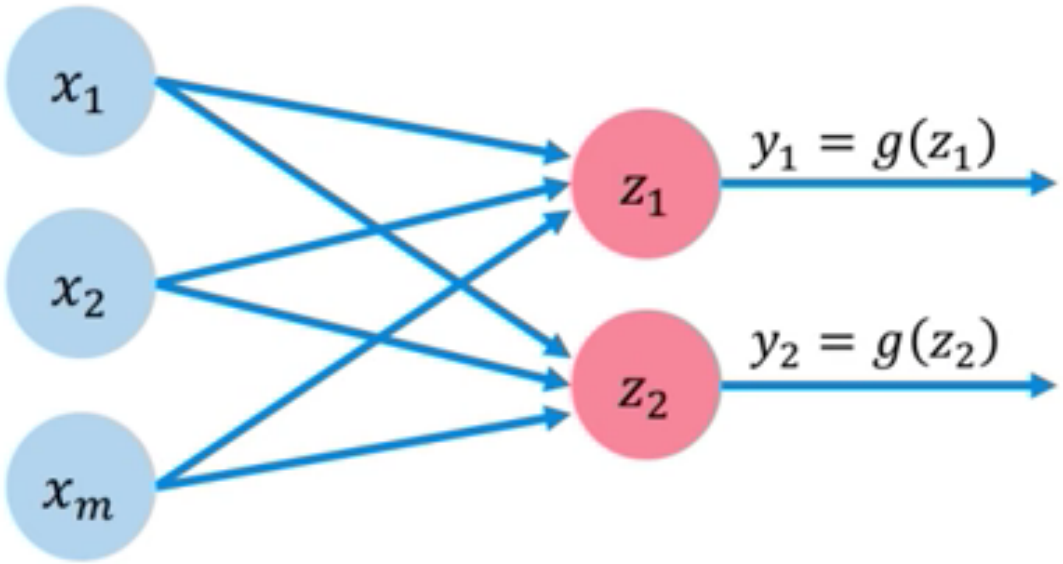


$$z = w_0 + \sum_{j=1}^{m} x_j w_j$$

$$z = w^T x + b$$

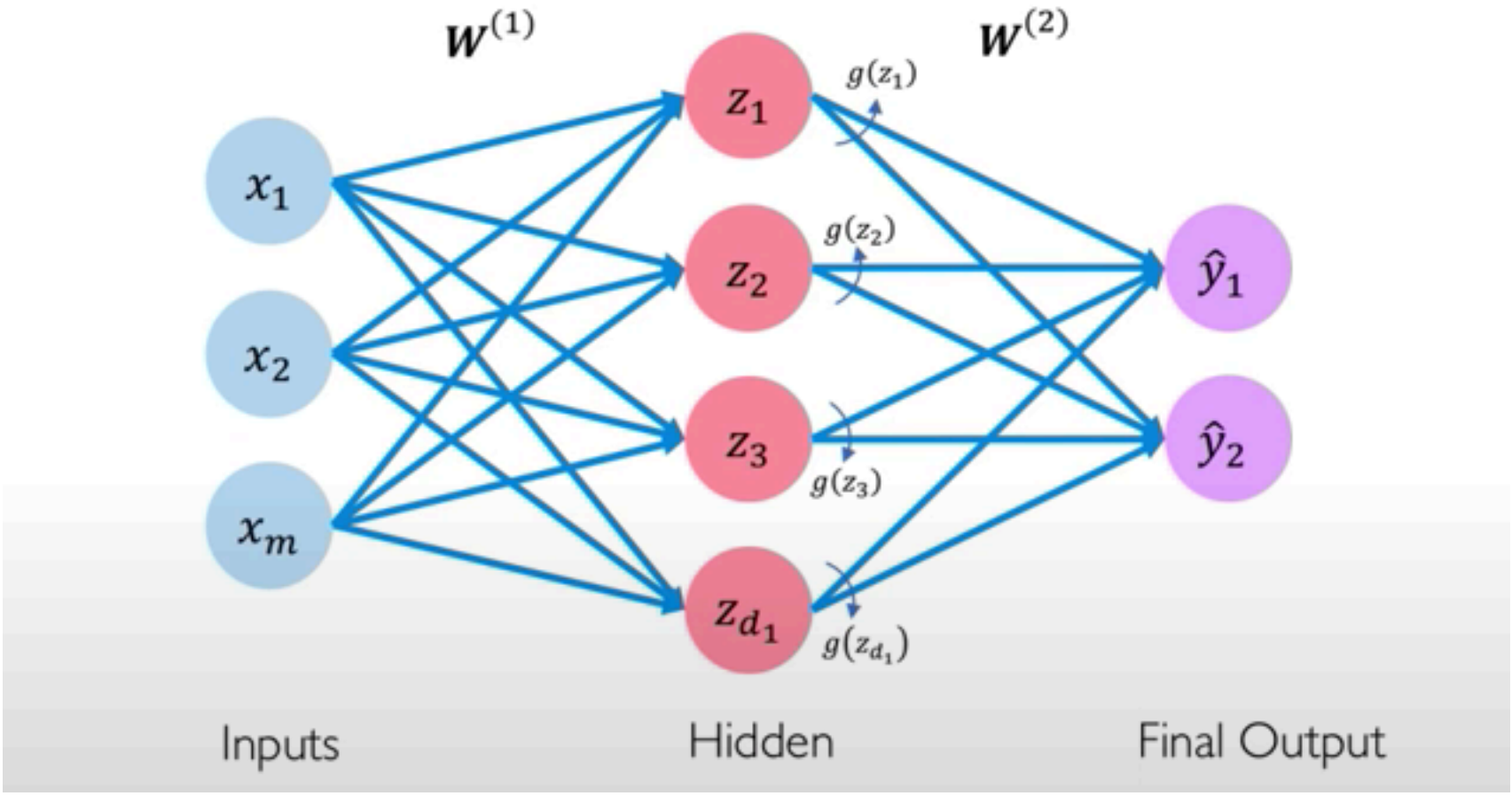
$$a = \sigma(z)$$

Building Neural Networks with Perceptron



$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

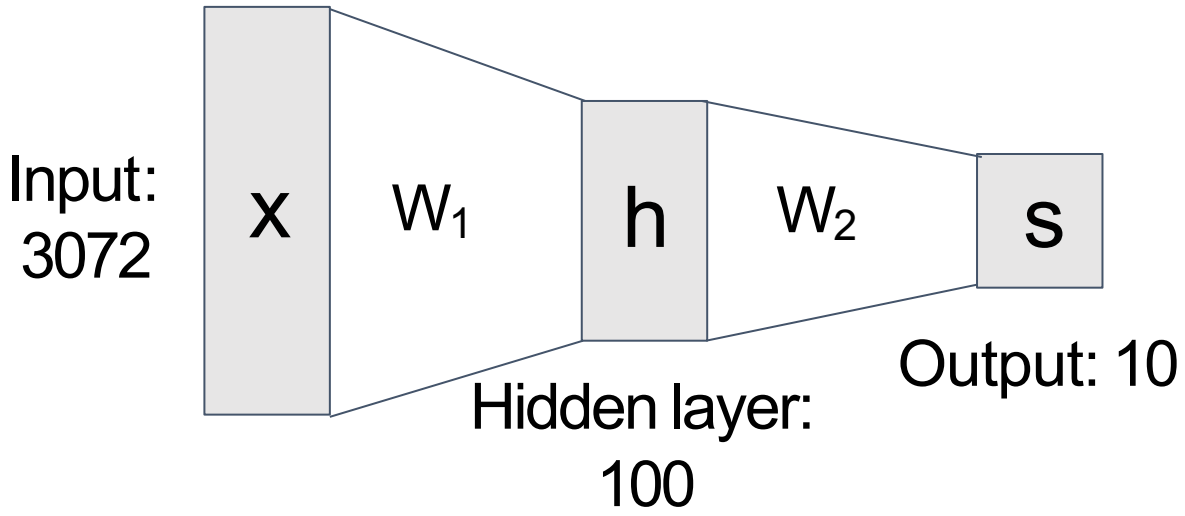
2-layer Neural Networks



2-layer Neural Networks

Before: Linear classifier $f(x) = Wx + b$

Now: 2-layer Neural Network $f(x) = W_2\, g\,(W_1 x + b_1) + b_2$



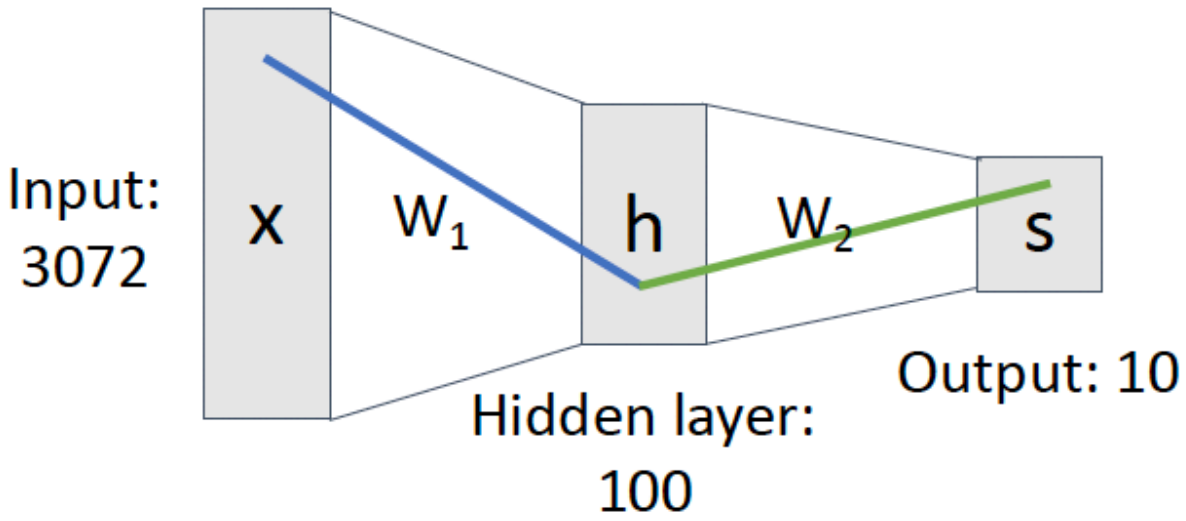
Credit: Justin

2-layer Neural Networks

Before: Linear classifier $f(x) = Wx + b$

Now: 2-layer Neural Network $f(x) = W_2\, g\,(W_1 x + b_1) + b_2$

Element (i, j) of W_1 gives the effect on h_i from x_j



Element (i, j) of W_2 gives the effect on s_i from h_j

Credit: Justin

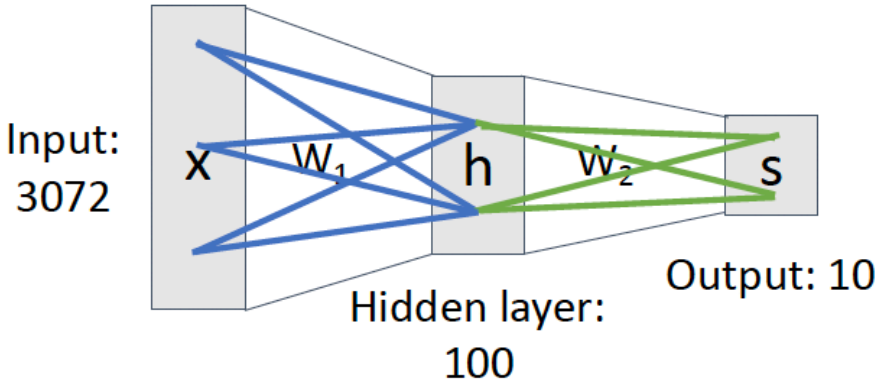
2-layer Neural Networks

Before: Linear classifier $f(x) = Wx + b$

Now: 2-layer Neural Network $f(x) = W_2\, g\,(W_1 x + b_1) + b_2$

Element (i, j) of W_1 gives the effect on h_i from x_j

All elements of x affect all elements of h



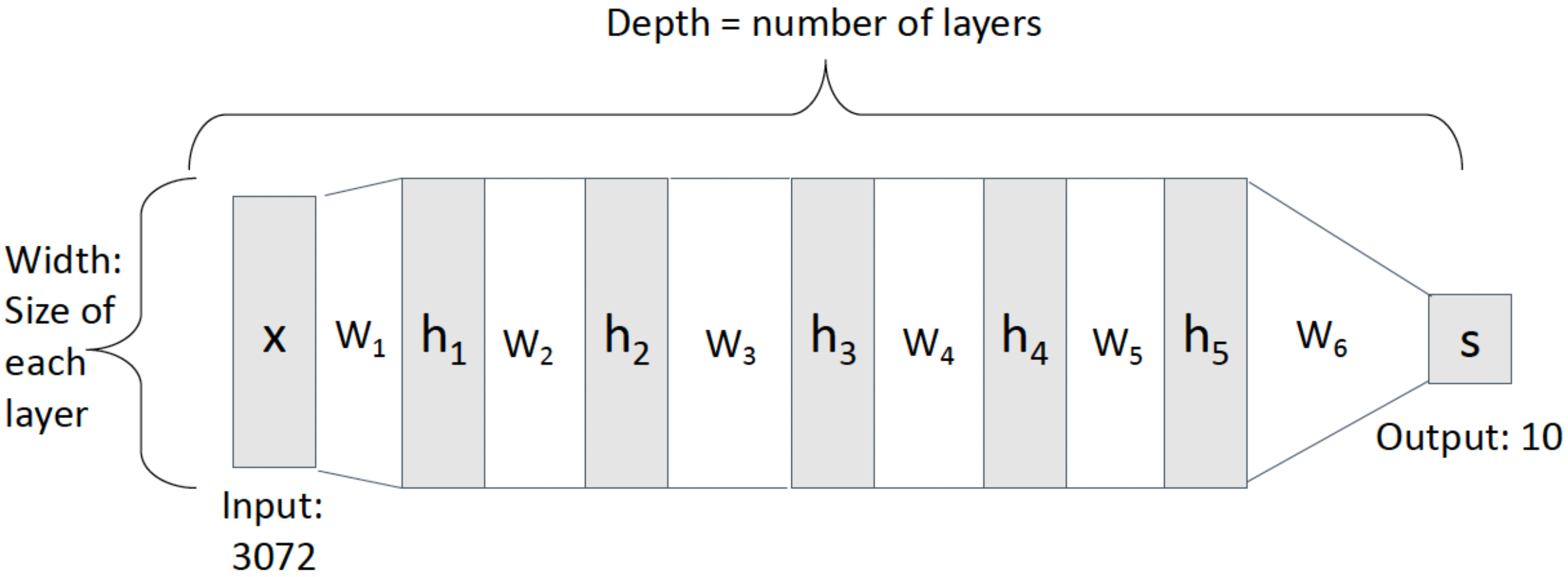
Element (i, j) of W_2 gives the effect on s_i from h_j

All elements of h affect all elements of s

Fully-connected neural network
Also "Multi-Layer Perceptron" (MLP)

Credit: Justin

Deep Neural Networks

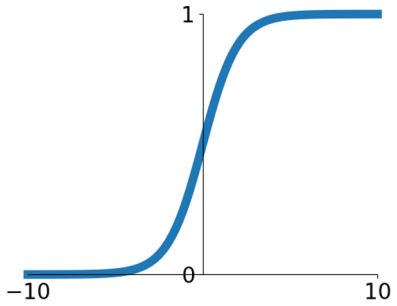


Credit: Justin

Activation Functions

Sigmoid

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

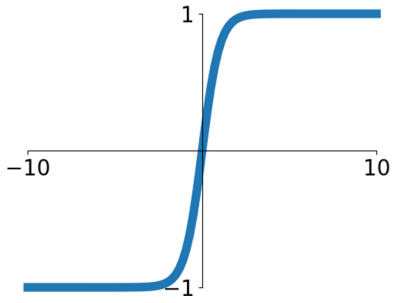


Probability interpretation

Activation Functions

tanh

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$



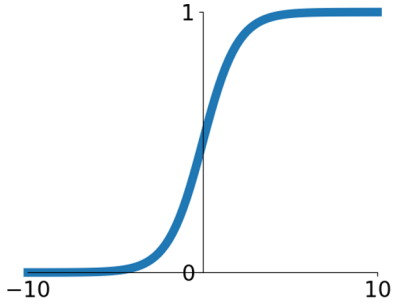
Zero mean / Zero centered

tanh is always superior than sigmoid

Activation Functions

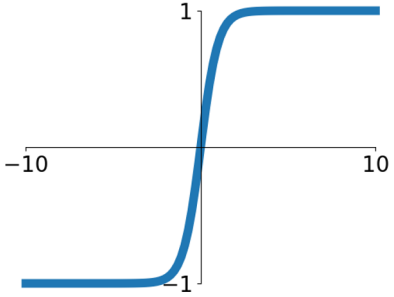
Sigmoid

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



tanh

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$



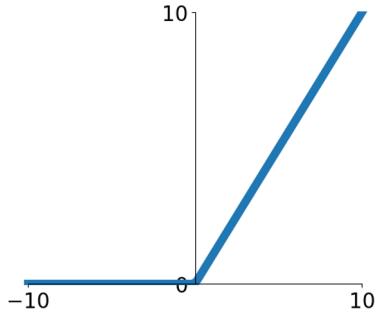
Drawbacks:

- gradient is small, and thus training (SGD) is slow
  - Exp () is computationally expensive

Activation Functions

ReLU (Rectified Linear Unit)

$\max(0, x)$



Default choice of activation function, ReLU is a good default choice for most problems

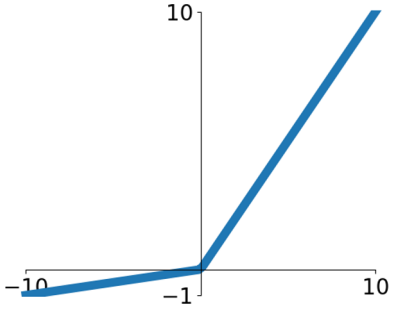
Drawbacks:

- Not zero centered.
- Dead ReLU when x < 0

Activation Functions

Leaky ReLU

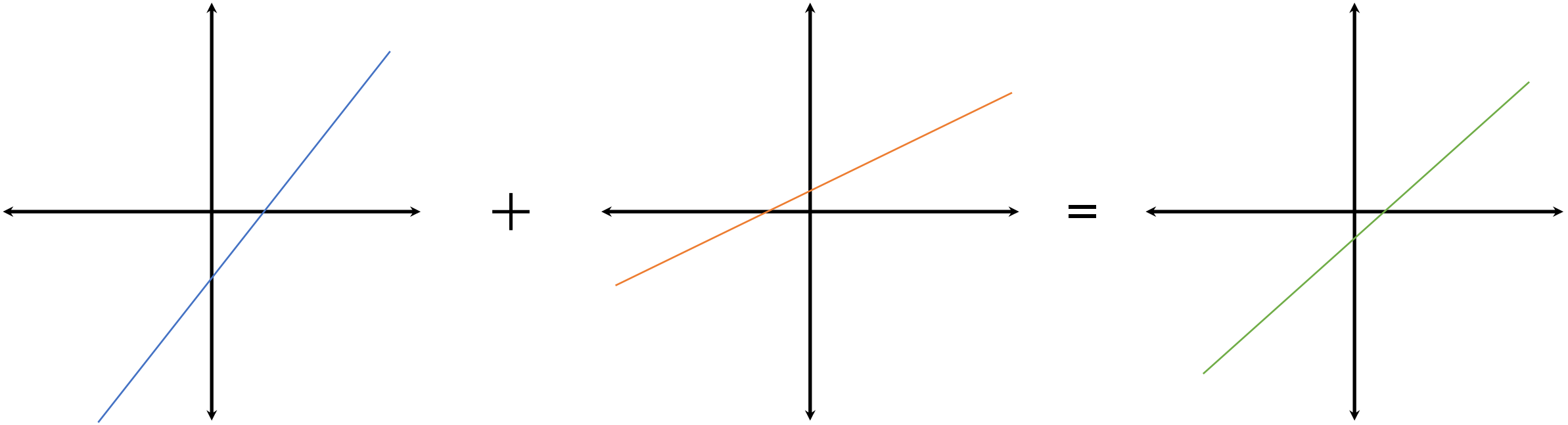
$\max(0.2x, x)$



Less used

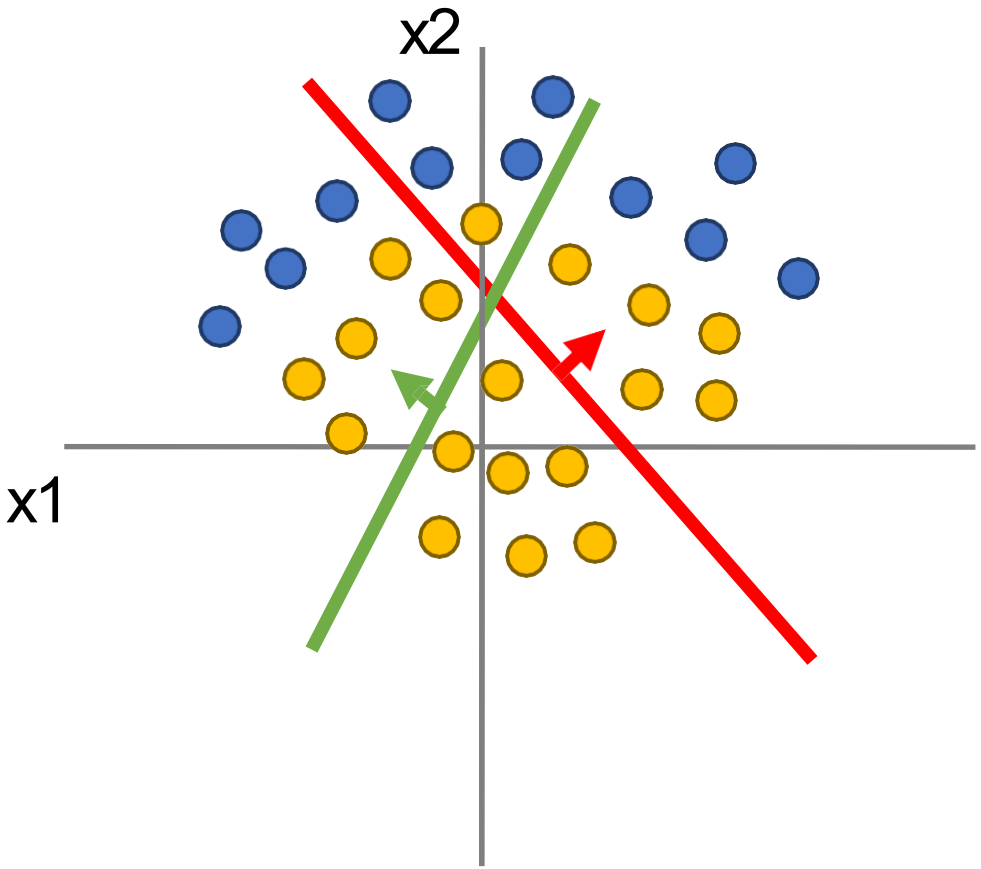
What happens if we build a neural network with no non-linear activation function?

N linear classifiers = 1 linear classifier



The power of non-linear activation function

Points not linearly separable in original space

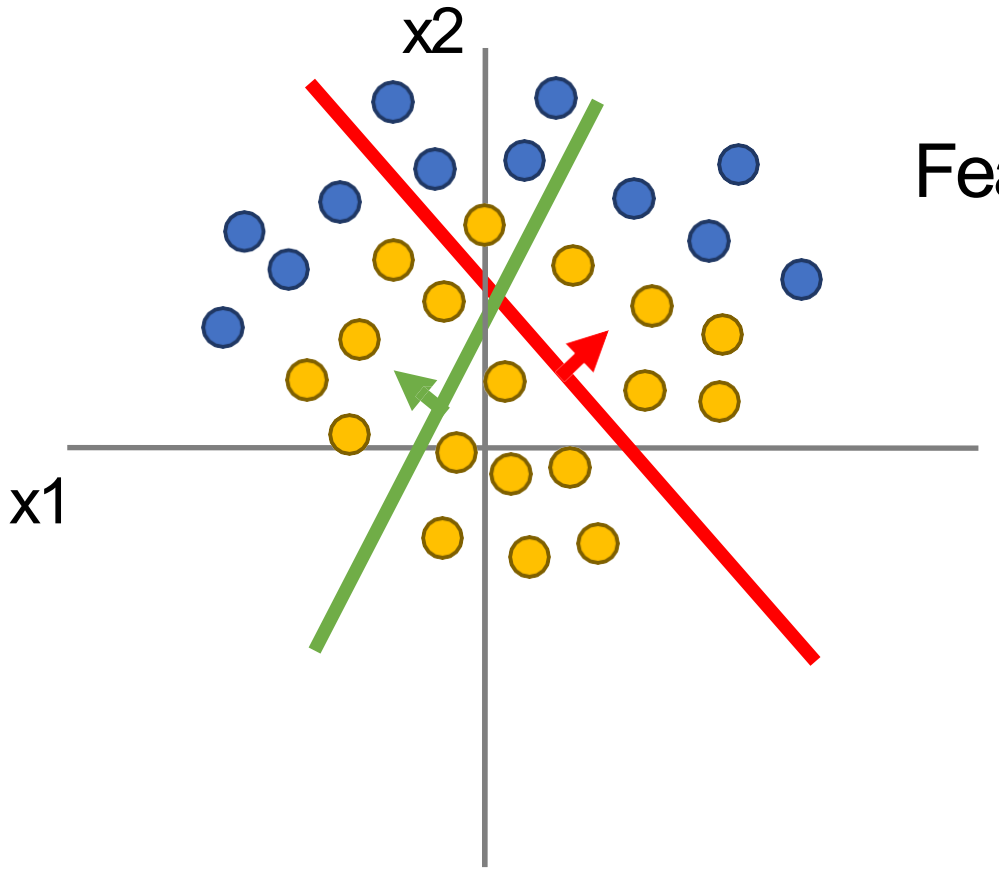


Consider a linear transform: $h = Wx$
Where x, h are both 2-dimensional

Credit: Justin

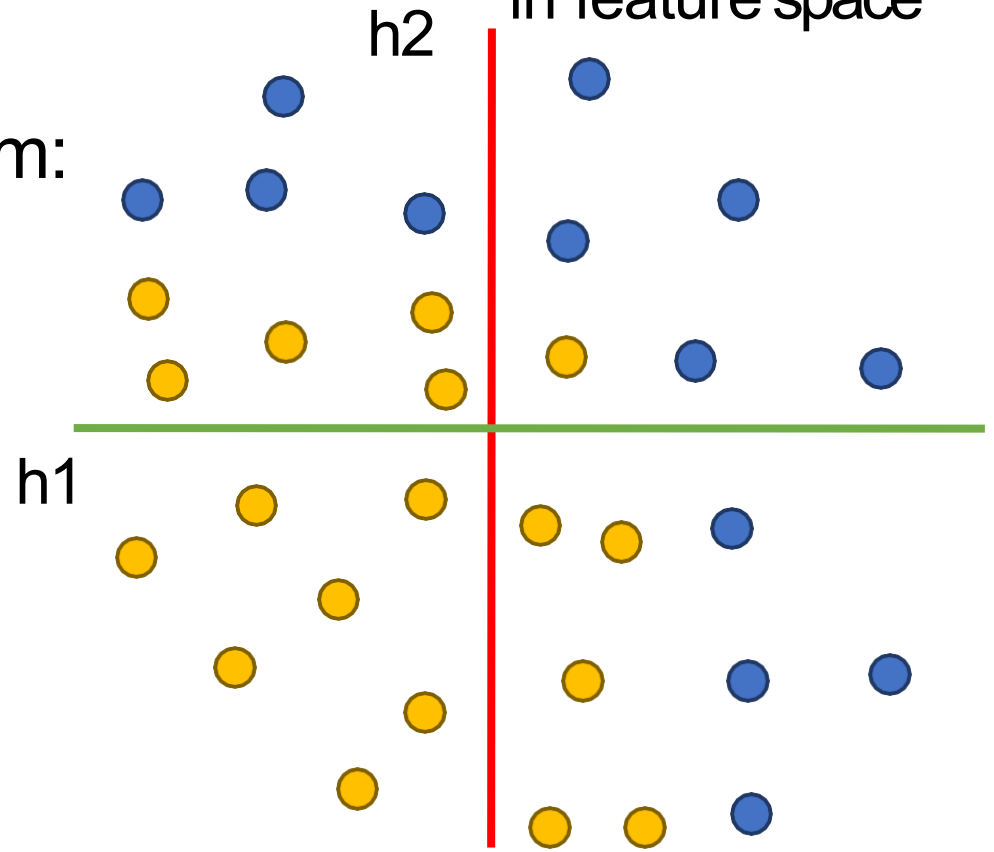
Consider a linear transform: h = Wx
Where x, h are both 2-dimensional

Points not linearly separable in original space



Feature transform:
h = Wx

Not linearly separable in feature space



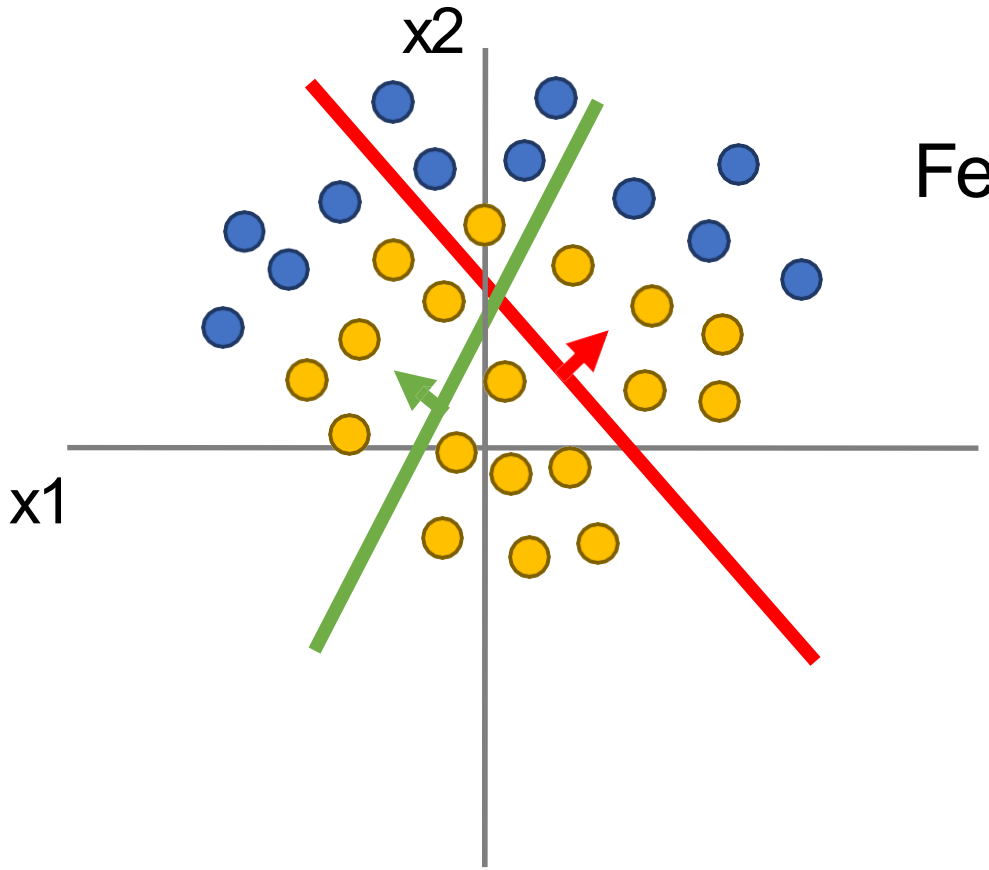
Credit: Justin

Consider a neural net hidden layer:

$h = \text{ReLU}(Wx) = \max(0, Wx)$

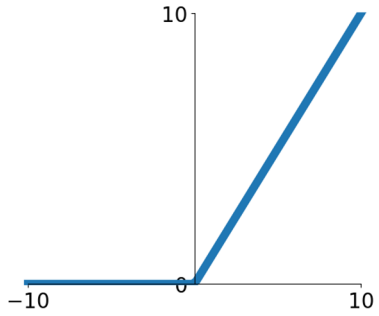
Where x, h are both 2-dimensional

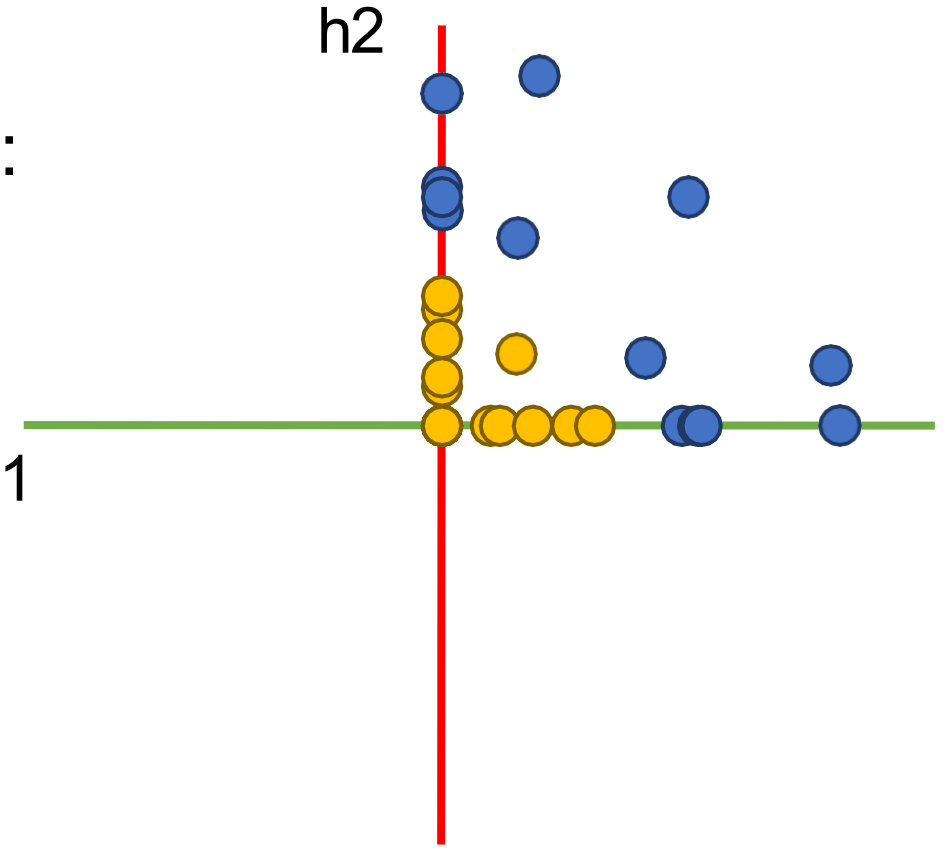
Points not linearly separable in original space



Feature transform:

$h = \text{ReLU}(Wx)$





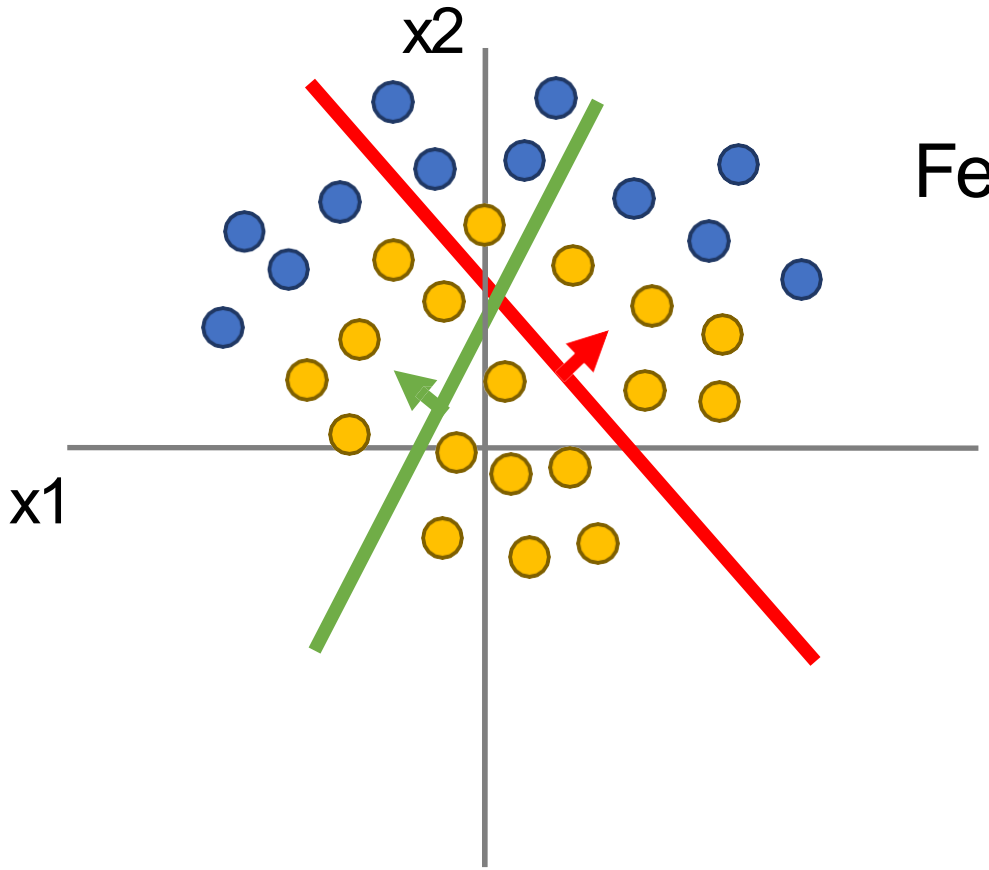
Credit: Justin

Consider a neural net hidden layer:

$h = \text{ReLU}(Wx) = \max(0, Wx)$

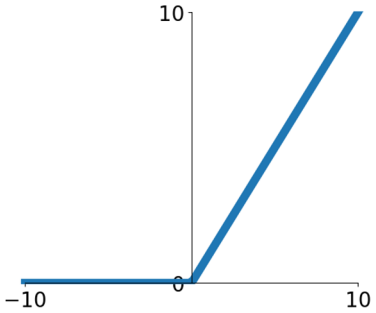
Where x, h are both 2-dimensional

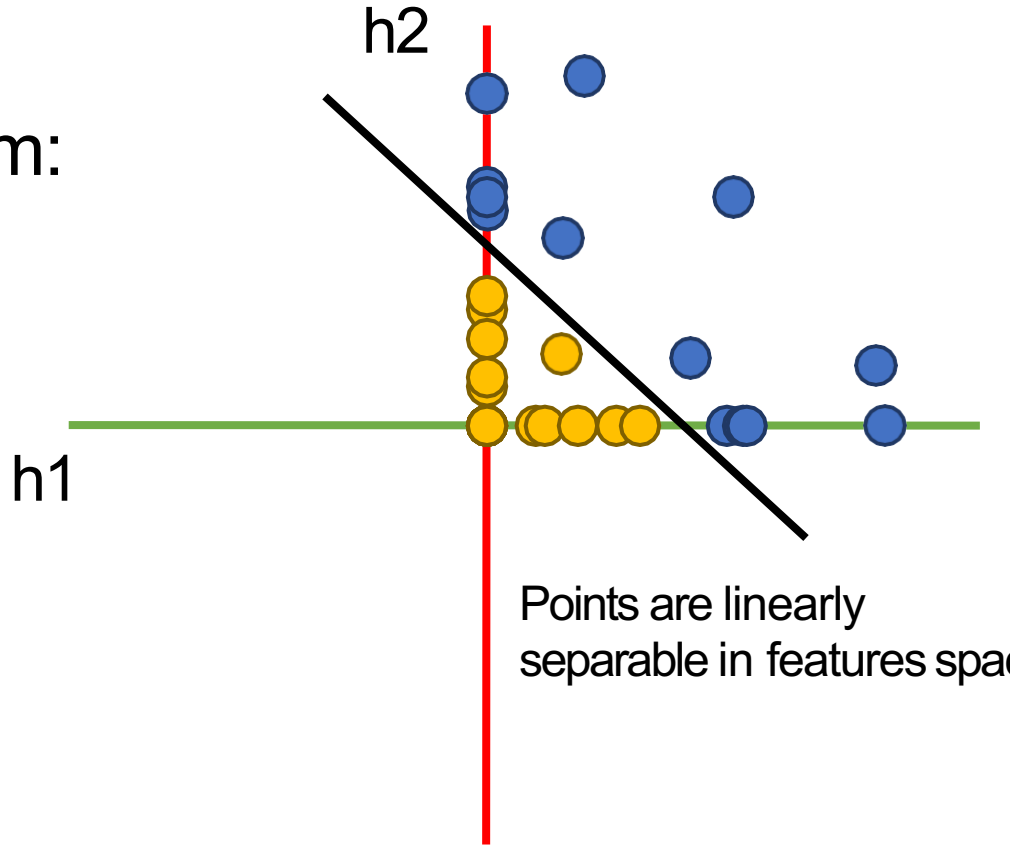
Points not linearly separable in original space



Feature transform:

$h = \text{ReLU}(Wx)$



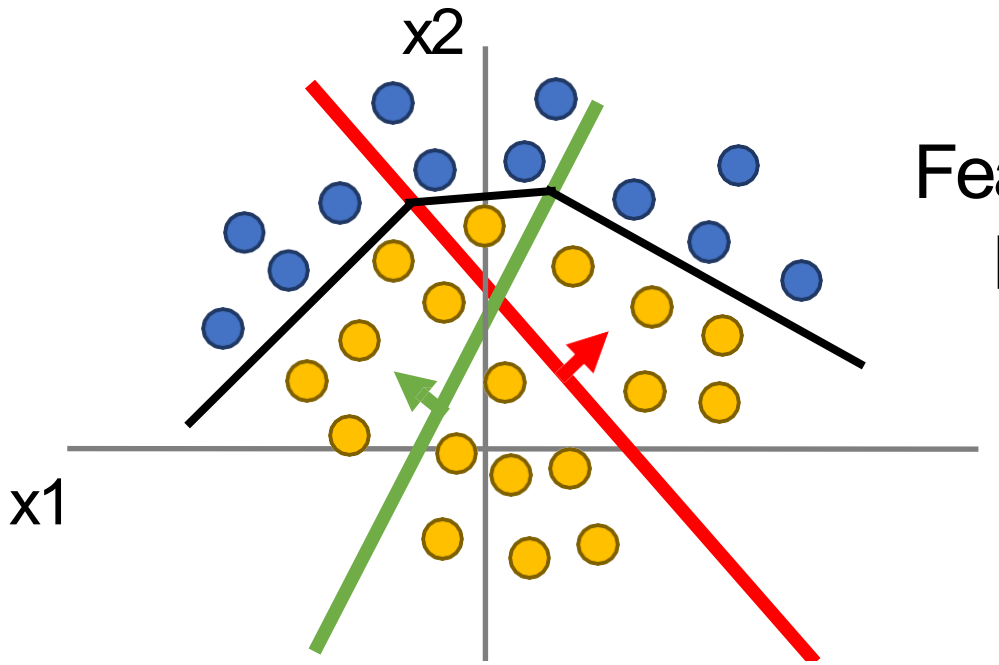


Points are linearly separable in features space!

Credit: Justin

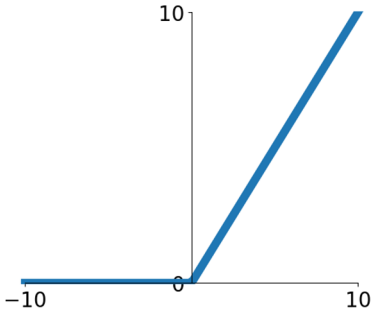
Consider a neural net hidden layer:
h = ReLU(Wx) = max(0, Wx)
Where x, h are both 2-dimensional

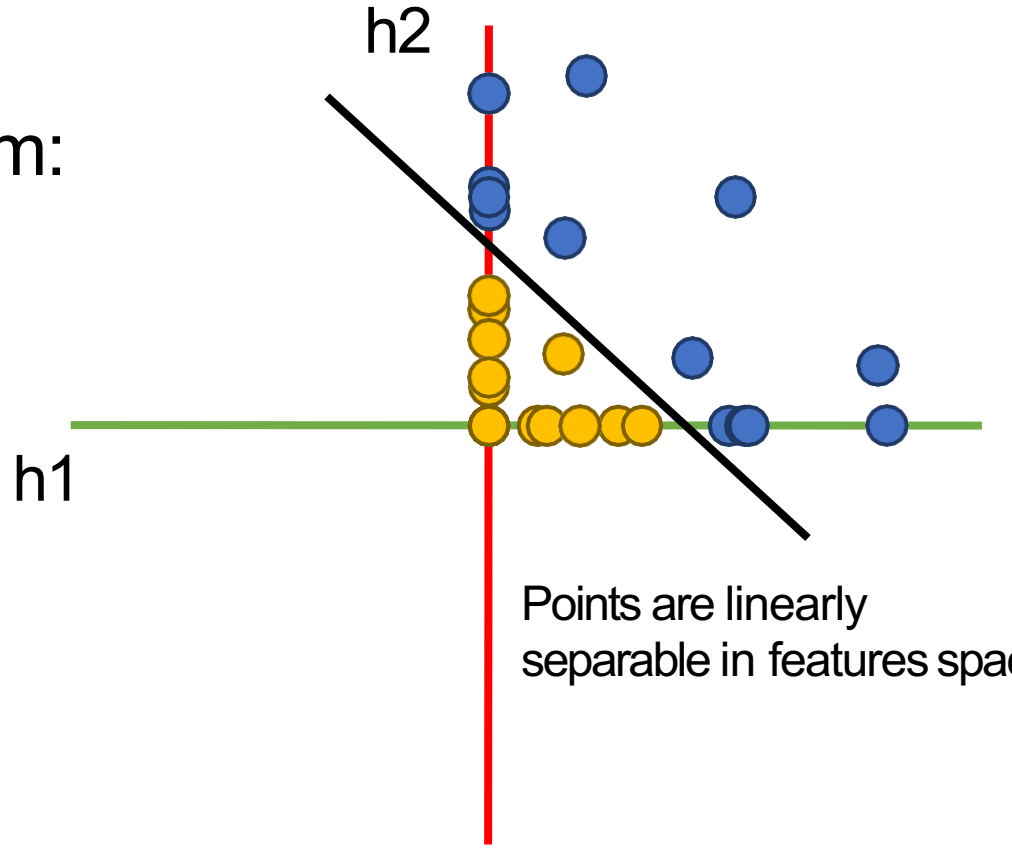
Points not linearly separable in original space



Linear classifier in feature space gives nonlinear classifier in original space

Feature transform:
h = ReLU(Wx)

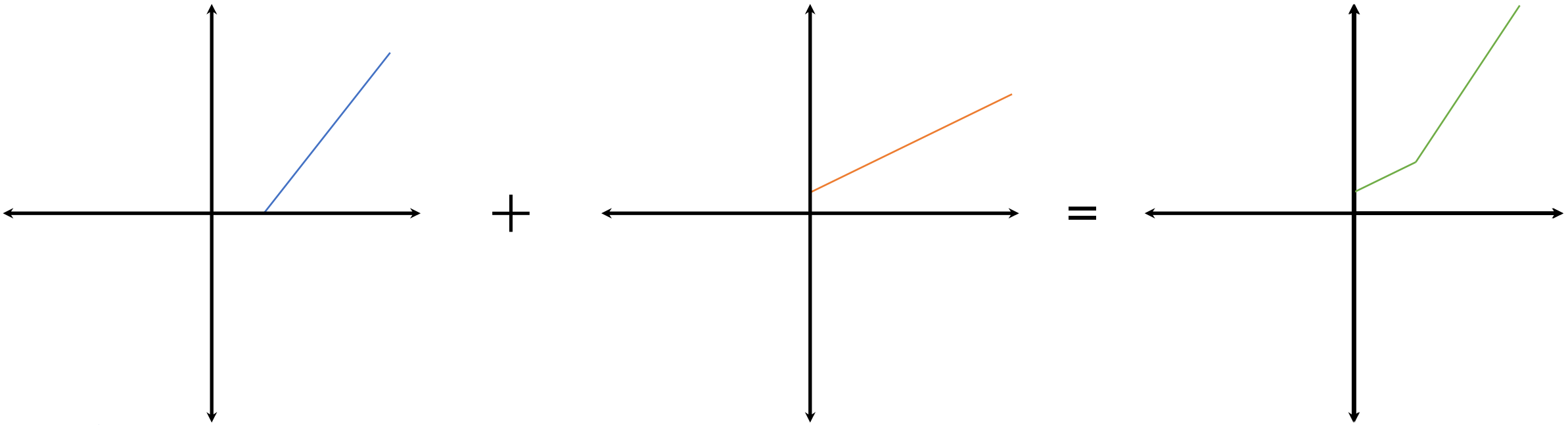




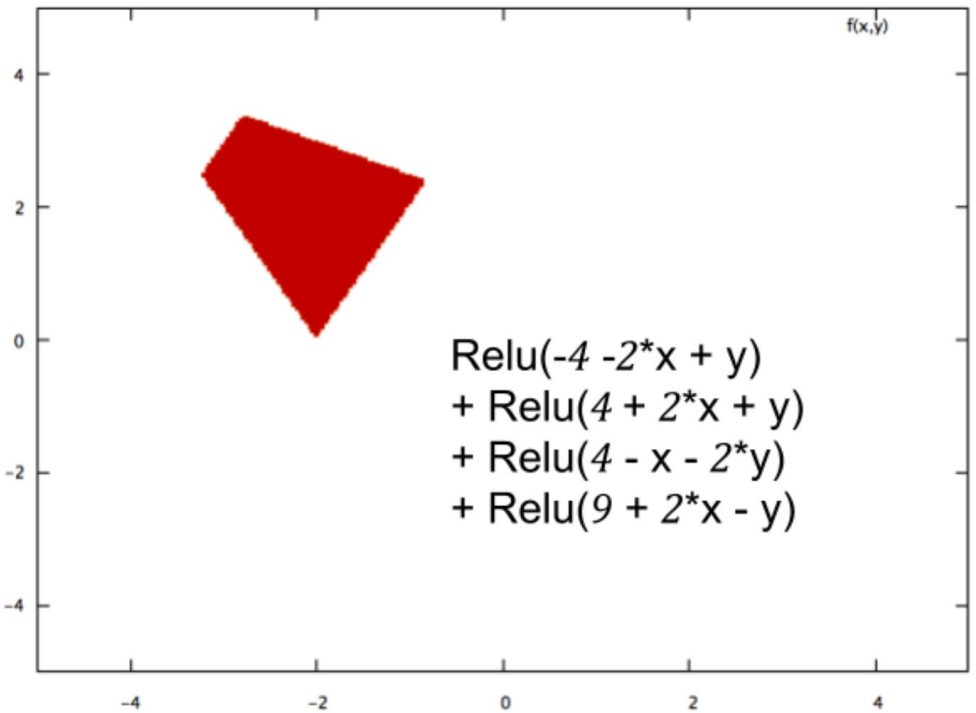
Points are linearly separable in features space!

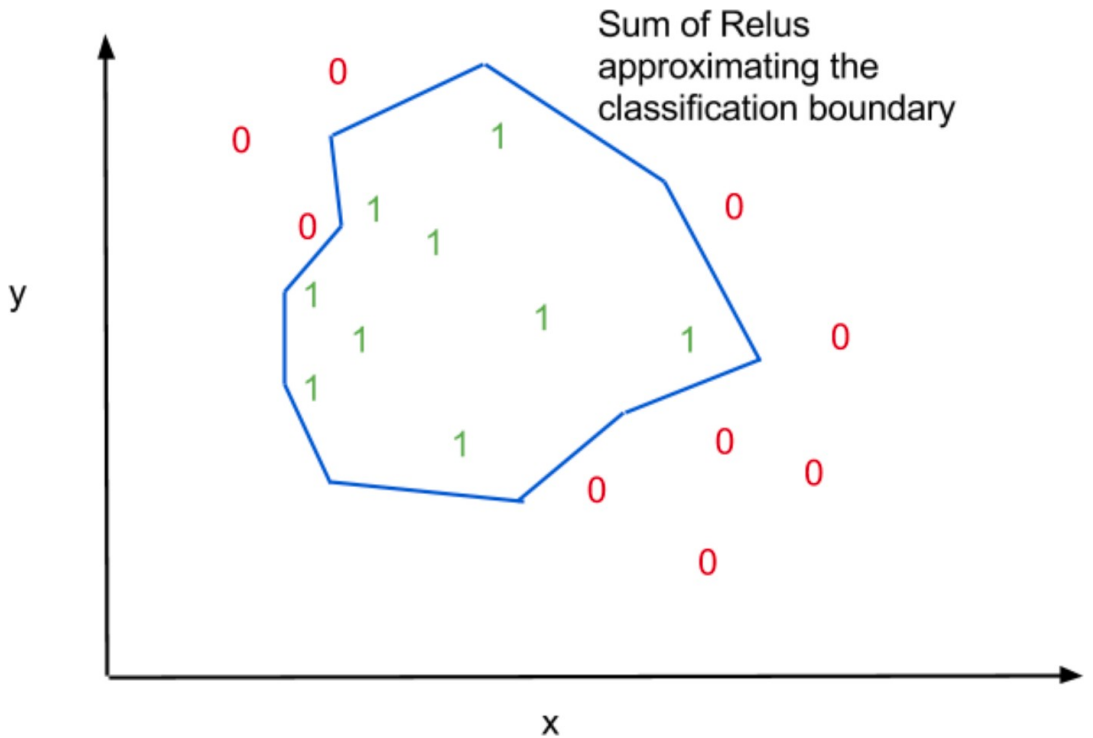
Credit: Justin

ReLU: Rectified Linear Units



ReLU, half-wave rectification, etc.


f(x,y)
4
2
0
-2
-4
-4
-2
0
2
4
Relu(-4 -2*x + y)
+ Relu(4 + 2*x + y)
+ Relu(4 - x - 2*y)
+ Relu(9 + 2*x - y)

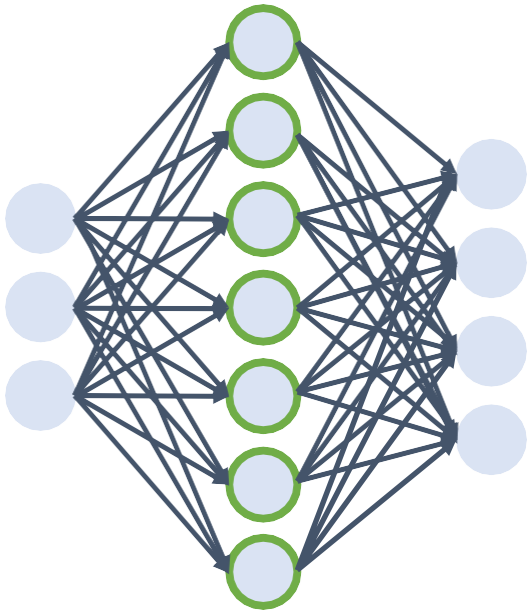

Sum of Relus
approximating the
classification boundary
y
x
0
0
1
0
1
1
0
1
1
1
1
1
0
0
0
0
0
0

Credit: blog

Universal Approximation

- A simple FC-Non-Linearity-FC network can approximate any function on a finite support, under:
  - mild assumptions
  - mild restrictions on non-linearity
- First proven for Sigmoid by Cybenko 1989
  - Later extended for ReLU and other non-linearities

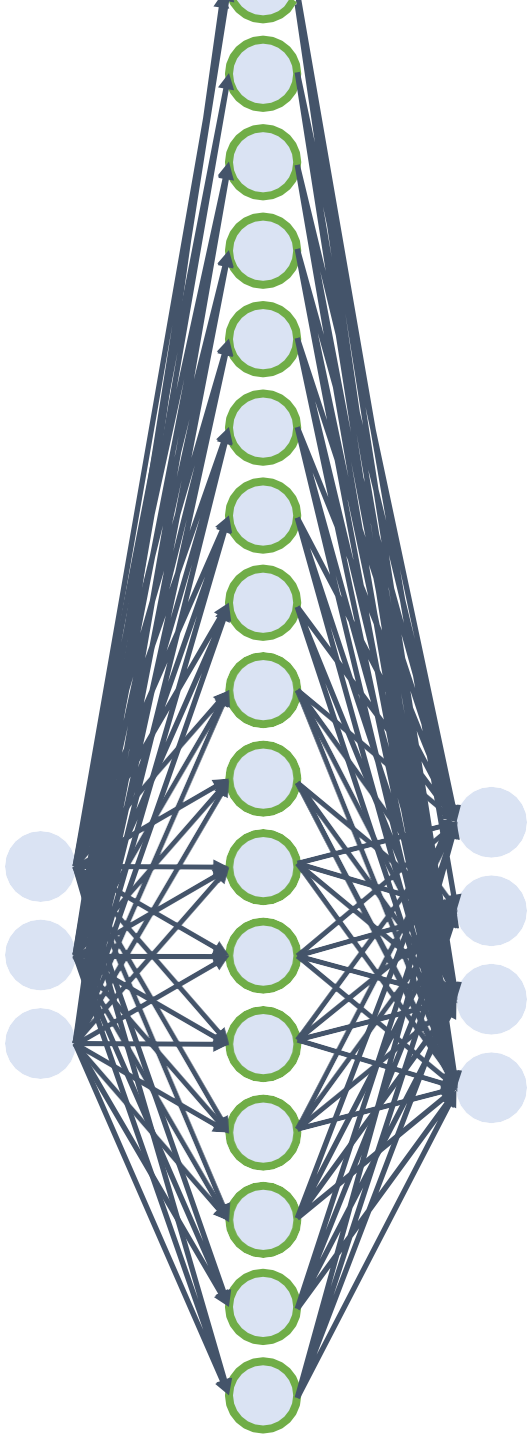
$$\hat{y} = w_2^T f(w_1^T x + b_1) + b_2$$



Credit: Krähenbühl.

Universal Approximation

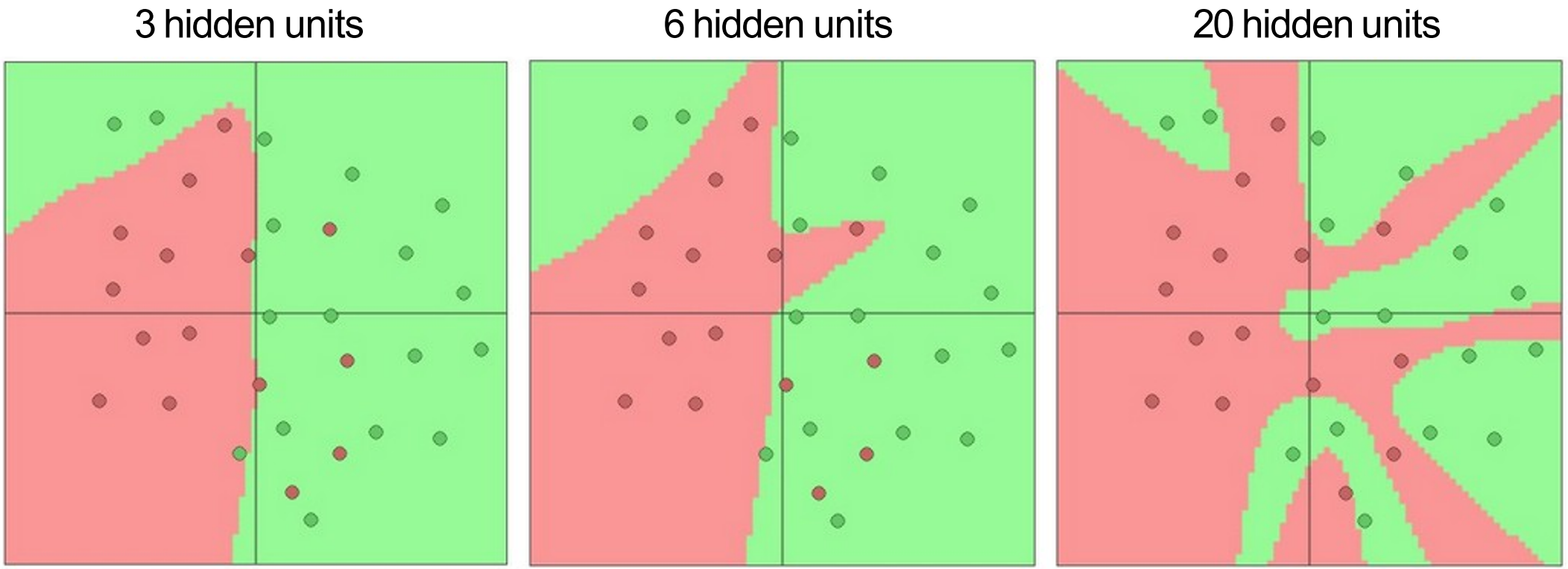
- Asimple FC-Non-Linearity-FC network can approximate any function on a finite support, under:
  - mild assumptions
  - mild restrictions on non-linearity
- Requires extremely large number of neurons in the FC layer
  - Hard to train
  - Computational infeasible
  - Theoretically possible



Credit: Krähenbühl.

Setting the number of layers and their sizes

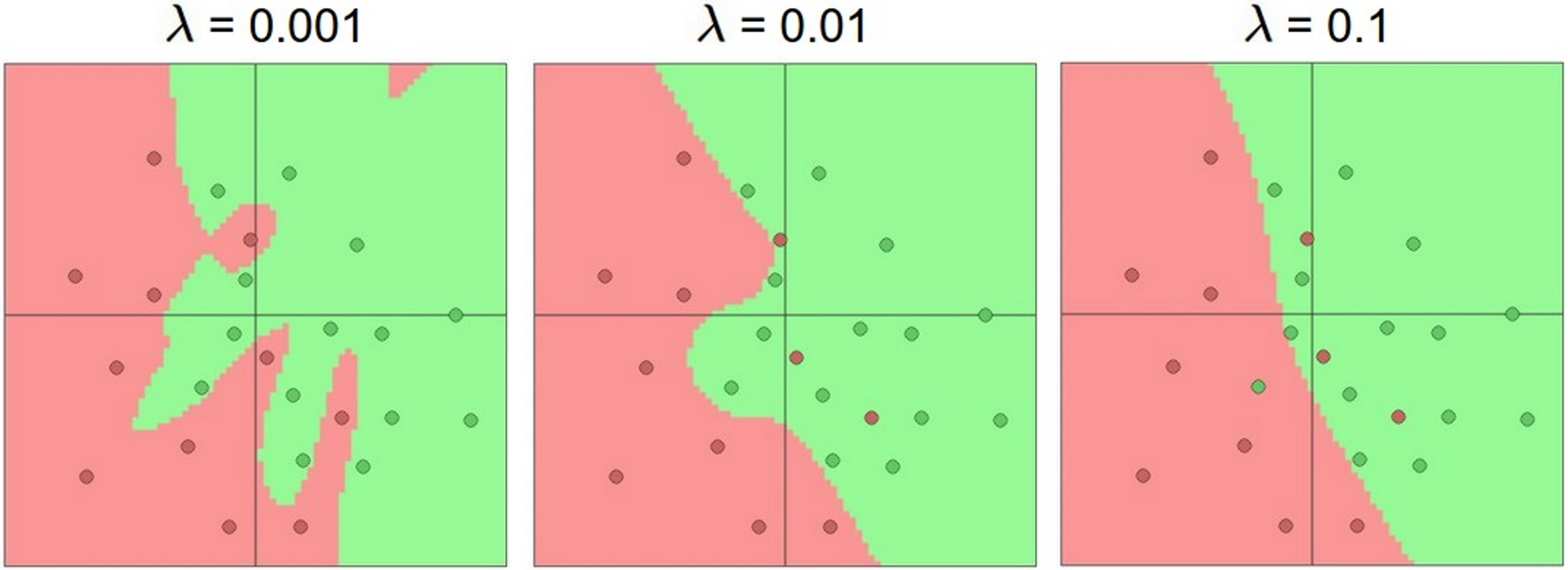
Setting the number of layers and their sizes



More hidden units = more capacity

Credit: Justin

Don’t regularize with size; instead use stronger L2



Credit: Justin

Any Question ?