Announcements

Assignments:

- HW4
  - Due date Mon, 2/24, 11:59 pm
- HW5
  - Out tomorrow
  - Due date Thu, 2/27, 11:59 pm

Midterm Conflicts

- See Piazza post
- Due 11:59pm on Wednesday the 19th of February

Plan

Last time

- Decision Boundaries
- Gaussian Generative Models
- Neural Networks

Today

- Neural Networks
  - Universal Approximation
  - Optimization / Backpropagation
  - (Convolutional Neural Networks)

Introduction to Machine Learning

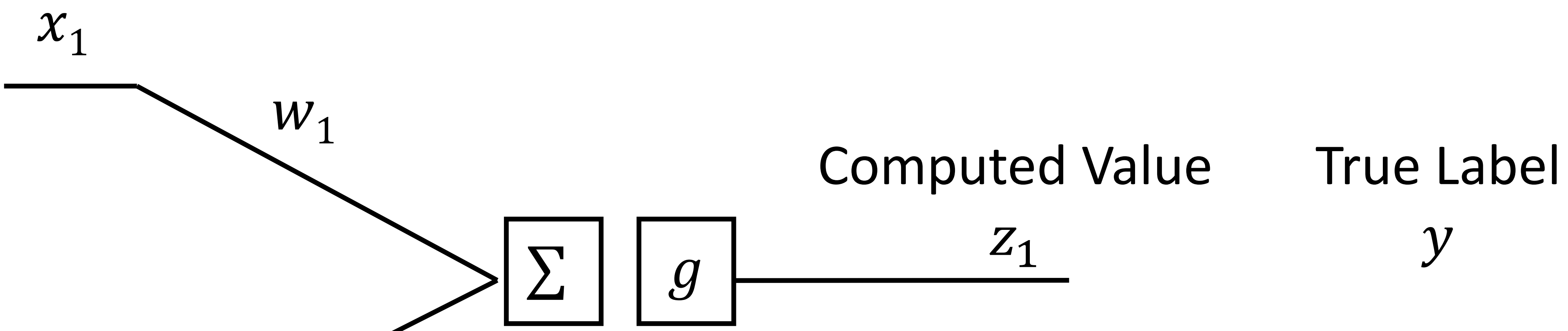
Neural Networks

Instructor: Pat Virtue

Single Neuron

Single neuron system

- Perceptron (if g is step function)
- Logistic regression (if g is sigmoid)



$$h_{\boldsymbol{w}}(\boldsymbol{x}) = z_1$$

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = g\left(\sum_i w_i x_i\right)$$

Optimizing

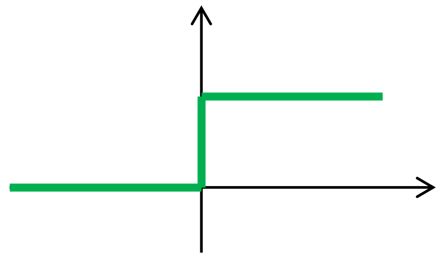
How do we find the "best" set of weights?

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = g\left(\sum_i w_i x_i\right)$$

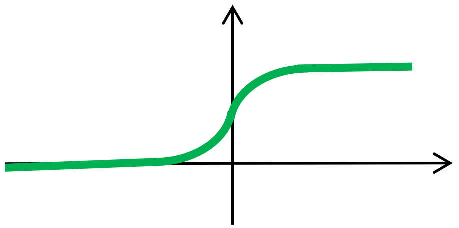
Activation Functions

It would be really helpful to have a g(z) that was nicely differentiable

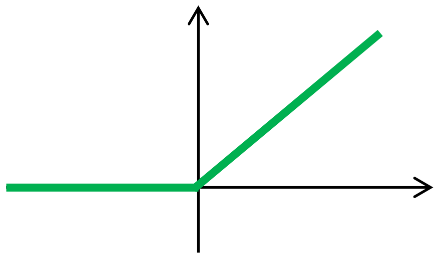
- Hard threshold: $g(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$ $\quad \frac{dg}{dz} = \begin{cases} 0 & z \geq 0 \\ 0 & z < 0 \end{cases}$



- Sigmoid: $g(z) = \frac{1}{1+e^{-z}}$ $\quad \frac{dg}{dz} = g(z)\big(1 - g(z)\big)$
- (Softmax)



- ReLU: $g(z) = \ max(0, z)$ $\quad \frac{dg}{dz} = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$



Loss Functions

Regression

- MSE (SSE): $\ell(\boldsymbol{y}, \widehat{\boldsymbol{y}}) = \|\boldsymbol{y} - \widehat{\boldsymbol{y}}\|_2^2 = \sum_n (y_n - \widehat{y_n})^2$

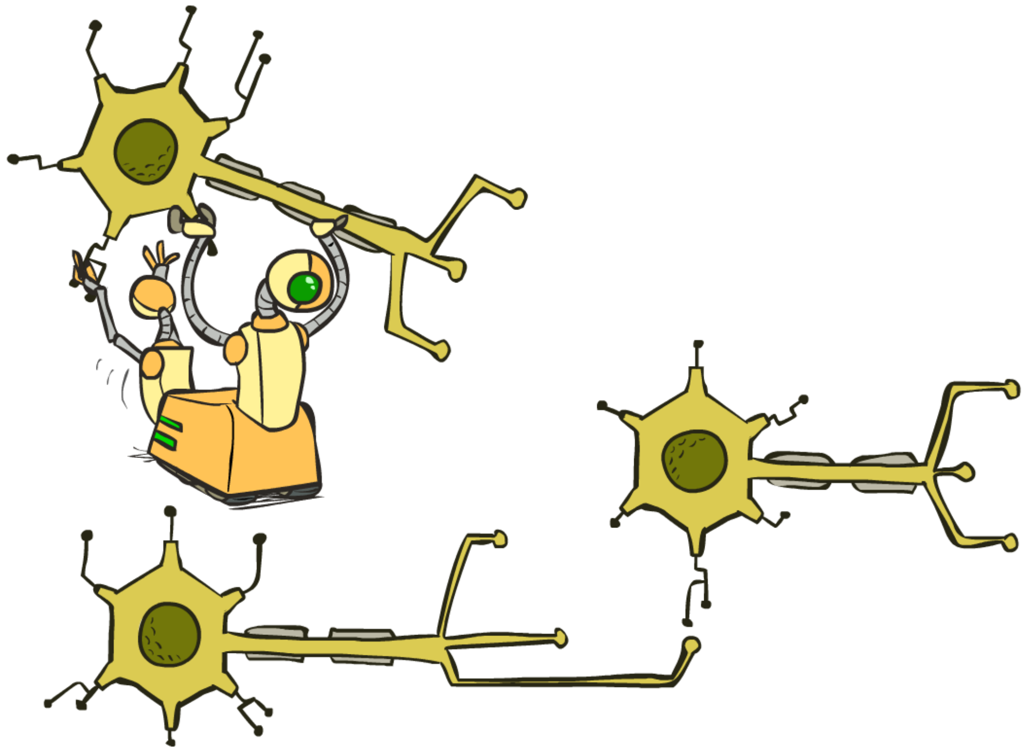
Classification

- Cross entropy: $\ell(\boldsymbol{y}, \widehat{\boldsymbol{y}}) = -\sum_n y_n \log \hat{y}_n$

Multilayer Perceptrons

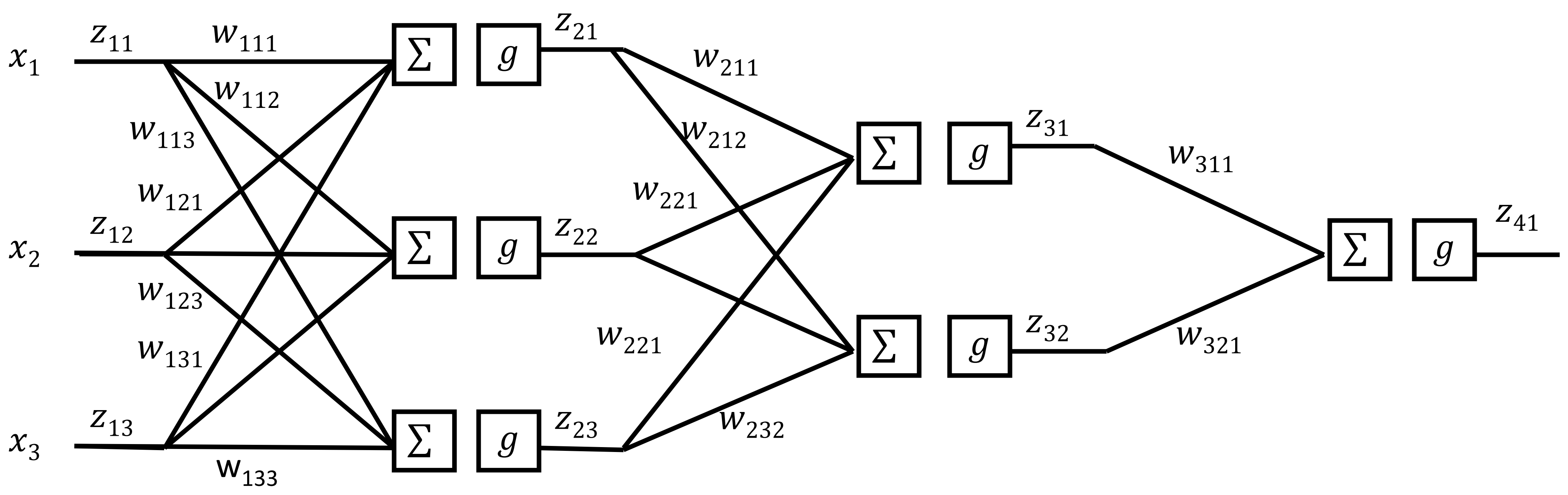
A ***multilayer perceptron*** is a feedforward neural network with at least one ***hidden layer*** (nodes that are neither inputs nor outputs)

MLPs with enough hidden nodes can represent any function



Slide from UC Berkeley AI

Neural Network Equations



$$h_w(\boldsymbol{x}) = z_{4,1}$$

$$z_{1,1} = x_1$$

$$z_{4,1} = g\left(\sum_i w_{3,i,1}\ z_{3,i}\right)$$

$$z_{3,1} = g\left(\sum_i w_{2,i,1}\ z_{2,i}\right)$$

$$z_{d,1} = g\left(\sum_i w_{d-1,i,1}\ z_{d-1,i}\right)$$

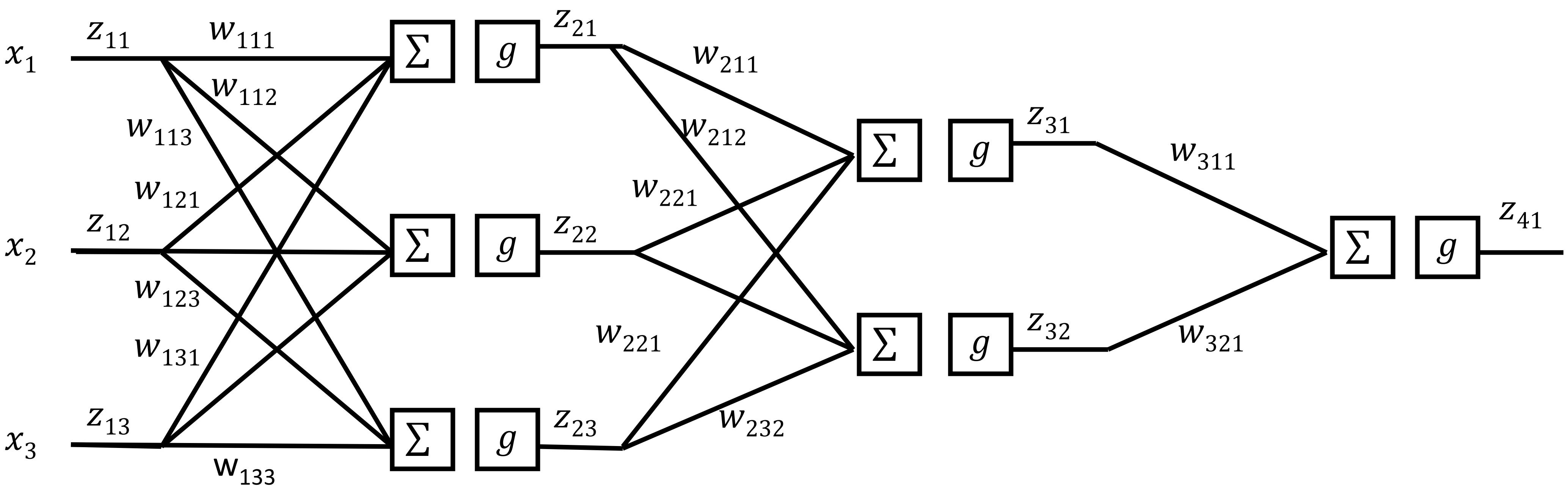
$$h_w(x) = g\left(\sum_k w_{3,k,1}\ g\left(\sum_j w_{2,j,k}\ g\left(\sum_i w_{1,i,j}\ x_i\right)\right)\right)$$

Optimizing

How do we find the "best" set of weights?

$$h_w(x) = g\left(\sum_k w_{3,k,1}\ g\left(\sum_j w_{2,j,k}\ g\left(\sum_i w_{1,i,j}\ x_i\right)\right)\right)$$

Neural Network Equations



How would you represent this specific network in PyTorch?

Neural Networks Properties

Practical considerations

- Large number of neurons
  - Danger for overfitting
- Modelling assumptions vs data assumptions trade-off
- Gradient descent can easily get stuck local optima

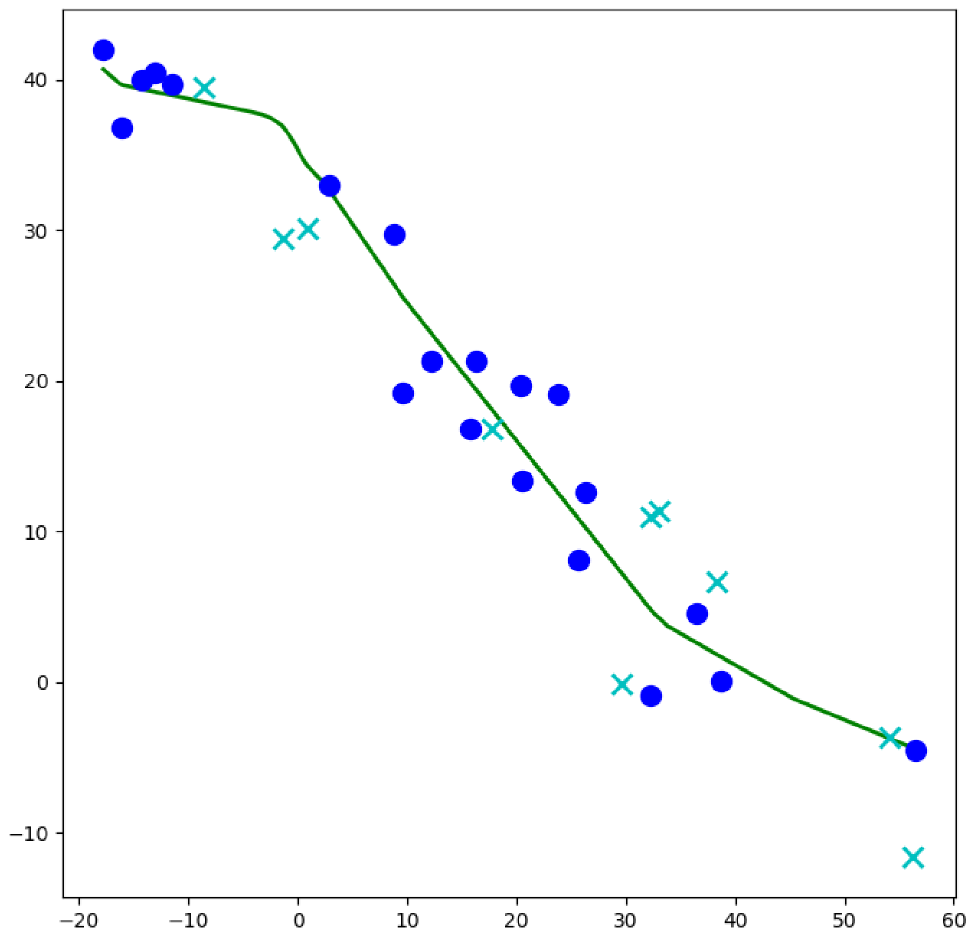
What if there are no non-linear activations?

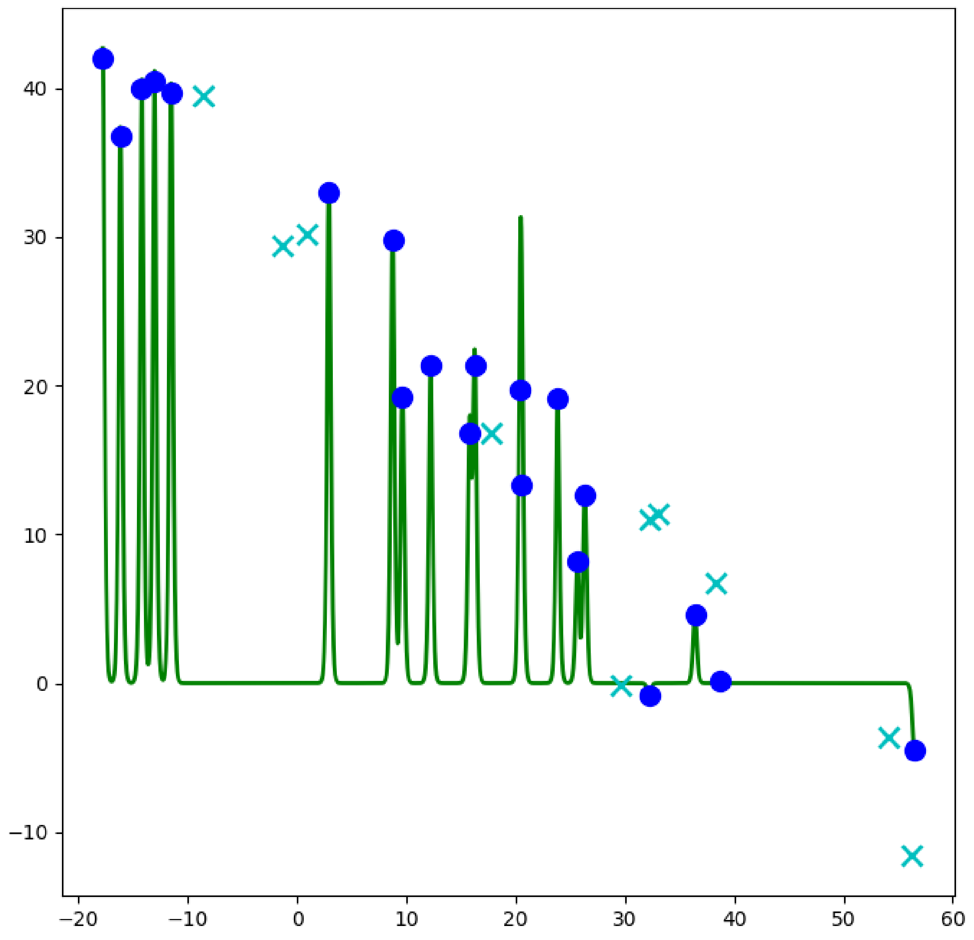
- A deep neural network with only linear layers can be reduced to an exactly equivalent single linear layer

Universal Approximation Theorem:

- A two-layer neural network with a sufficient number of neurons can approximate any continuous function to any desired accuracy.

Network to Approximate a 1-D Function





Network to Approximate a 1-D Function

Design a network to approximate this function using:

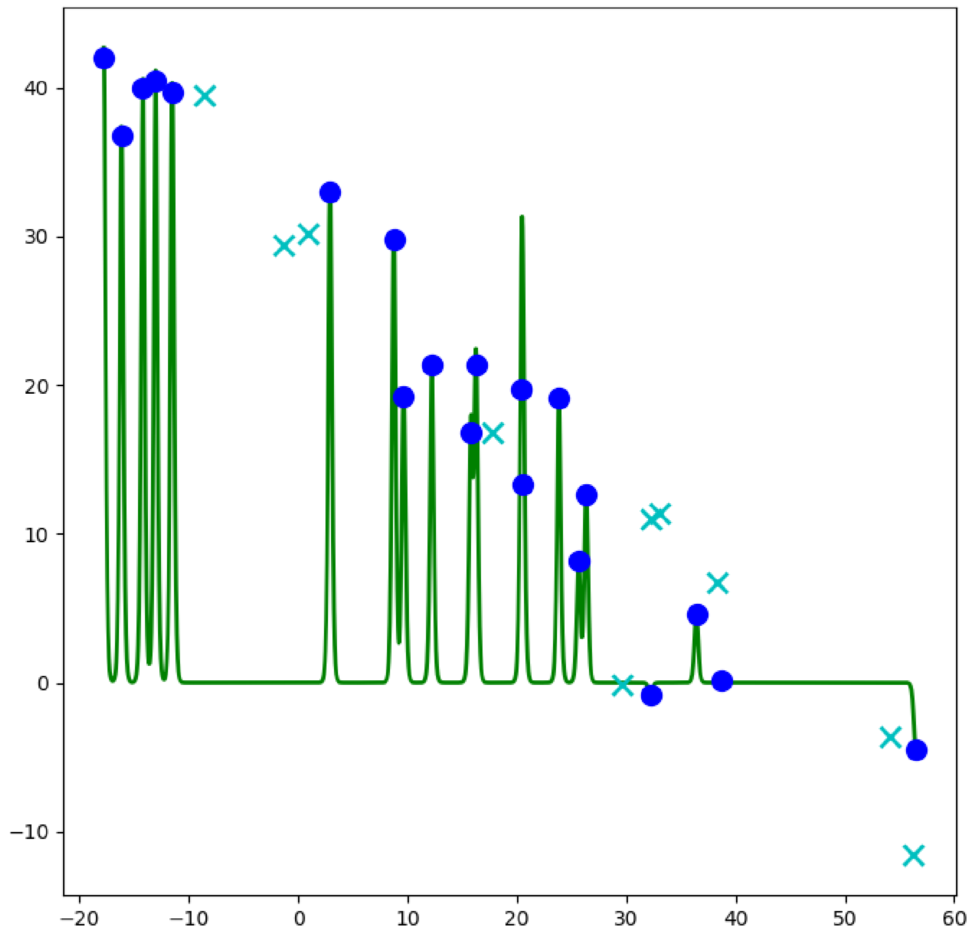
Linear, Sigmoid, Step, or ReLU

Network to Approximate a 1-D Function

Design a network to approximate this function using:

Linear, Sigmoid, Step, or ReLU

Network to Approximate a 1-D Function



Network to Approximate Binary Classification

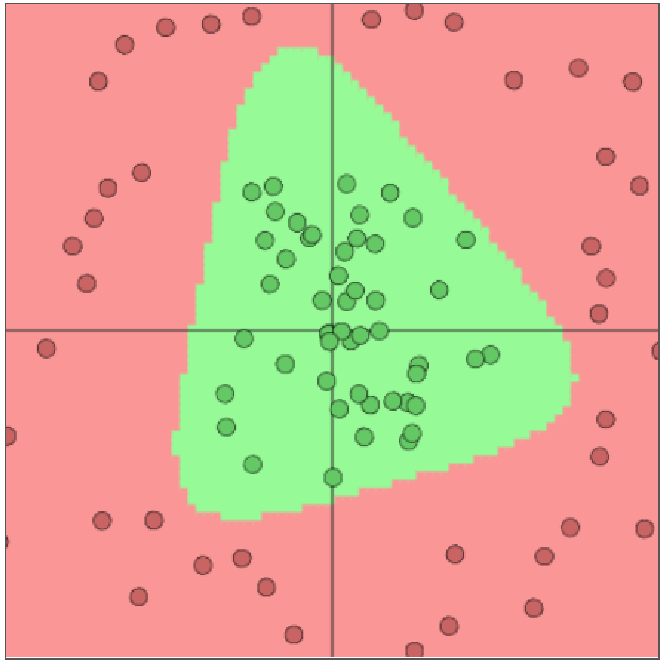
Approximate function $f(x_1, x_2) = x_1 \wedge x_2$

Network to Approximate Binary Classification

Approximate function $f(x_1, x_2) = x_1 \oplus x_2$

Network to Approximate Binary Classification

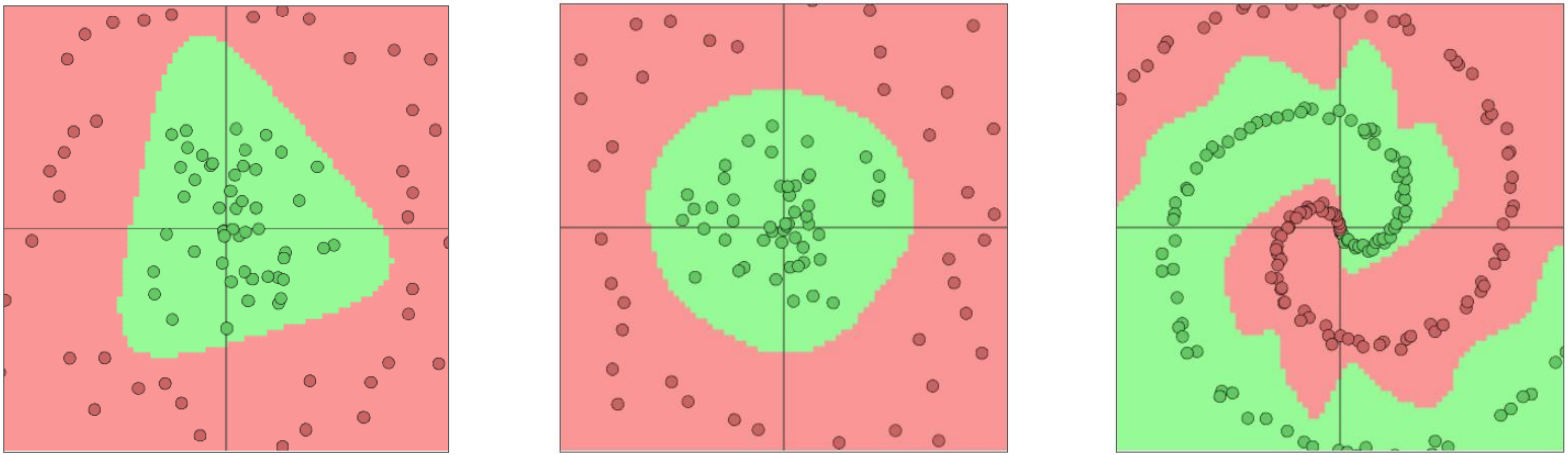
Approximate arbitrary decision boundary



https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

Network to Approximate Binary Classification

Approximate arbitrary decision boundary



https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

Network Optimization

Reminder: Calculus Chain Rule (scalar version)

$$y = f(z)$$
$$z = g(x)$$

$$\frac{dy}{dx} = \frac{dy}{dz}\frac{dz}{dx}$$

Network Optimization

$J(\mathbf{w}) = z_3$

$z_3 = f_3(w_3, z_2)$

$z_2 = f_2(w_2, z_1)$

$z_1 = f_1(w_1, x)$

Backpropagation (so-far)

Compute derivatives per layer, utilizing previous derivatives

Objective: $J(\boldsymbol{w})$

Arbitrary layer: $y = f(x, w)$

Need:

- $\frac{\partial J}{\partial x} = \frac{\partial J}{\partial y} \frac{\partial y}{\partial x}$
- $\frac{\partial J}{\partial w} = \frac{\partial J}{\partial y} \frac{\partial y}{\partial w}$