# 10-315
# Introduction to ML

# Neural Networks

Instructor: Pat Virtue

# Outline

## Last time: Neural Networks

- Three-neuron network + optimization
- Neural network structure (adding more neurons)
- Forward pass and Backpropagation (scalar version)

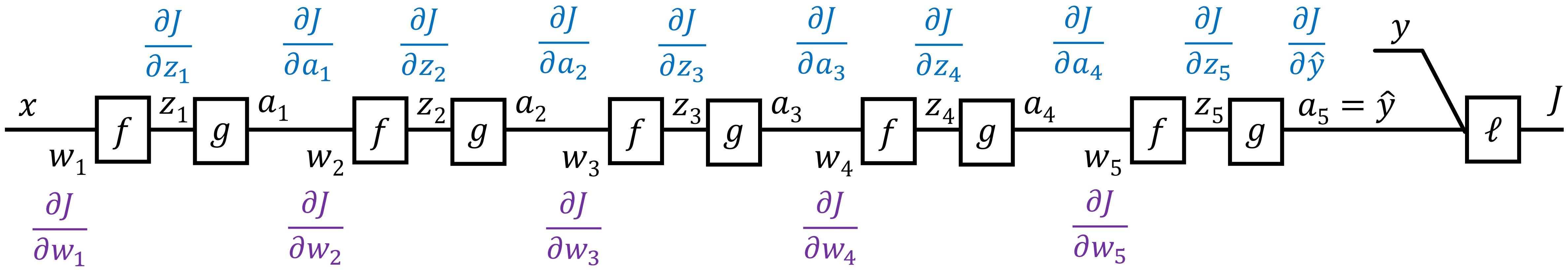
## Today: Neural Networks

- Calculus: multi-variate chain rule
- Backpropagation (vector version)
- Neural Network Properties and Intuition

# Backpropagation

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(w_5 \cdot g\left(w_4 \cdot g\left(w_3 \cdot g(w_2 \cdot g(w_1 \cdot x))\right)\right)\right)$$

## Width 1 deep network (no bias) (dumb but will help with calculus)



$\frac{\partial J}{\partial z_1}$
$\frac{\partial J}{\partial a_1}$
$\frac{\partial J}{\partial z_2}$
$\frac{\partial J}{\partial a_2}$
$\frac{\partial J}{\partial z_3}$
$\frac{\partial J}{\partial a_3}$
$\frac{\partial J}{\partial z_4}$
$\frac{\partial J}{\partial a_4}$
$\frac{\partial J}{\partial z_5}$
$\frac{\partial J}{\partial \hat{y}}$
$y$
$x$
$f$
$z_1$
$g$
$a_1$
$f$
$z_2$
$g$
$a_2$
$f$
$z_3$
$g$
$a_3$
$f$
$z_4$
$g$
$a_4$
$f$
$z_5$
$g$
$a_5 = \hat{y}$
$\ell$
$J$
$w_1$
$w_2$
$w_3$
$w_4$
$w_5$
$\frac{\partial J}{\partial w_1}$
$\frac{\partial J}{\partial w_2}$
$\frac{\partial J}{\partial w_3}$
$\frac{\partial J}{\partial w_4}$
$\frac{\partial J}{\partial w_5}$

To do gradient descent we need the partial derivative of the objective with respect to each parameter, $w_\ell \leftarrow w_\ell - \alpha\, \partial J/\partial w_\ell$

The backward pass propagates the change in the objective with respect to intermediate values ($\partial J/\partial z_\ell$ and $\partial J/\partial a_\ell$) back through the network to produce each $\partial J/\partial w_\ell$

# Generic Layer Implementation (so-far)

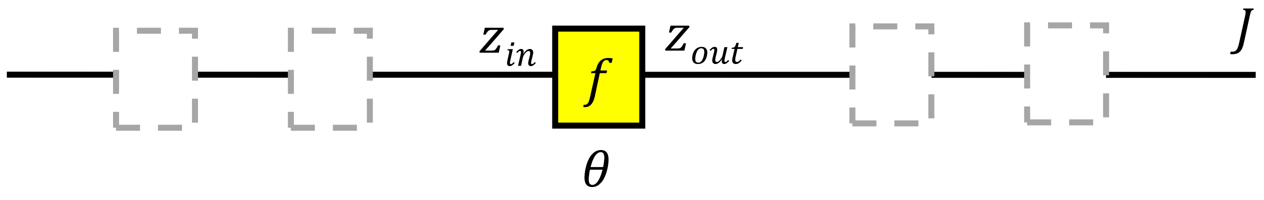
Compute derivatives per layer, utilizing previous derivatives

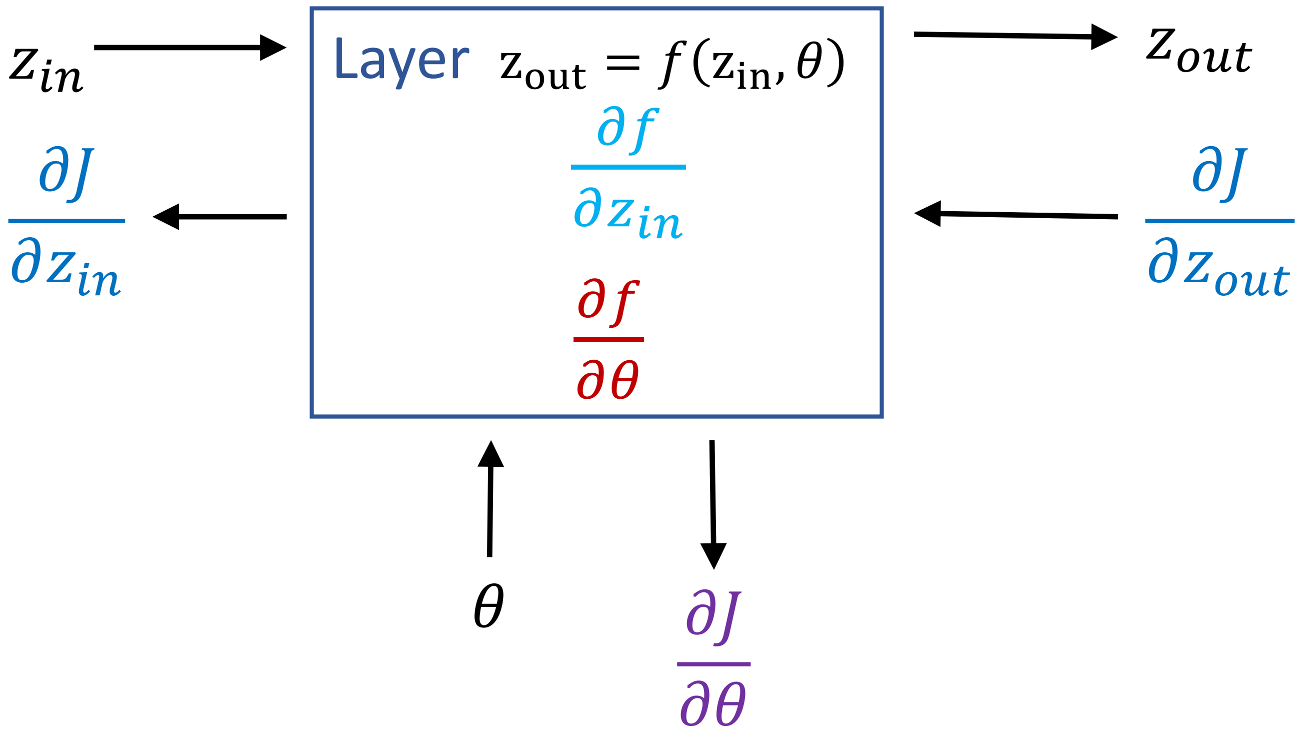
Objective: $J(\theta)$

Arbitrary layer: $\mathrm{z_{out}} = f(\mathrm{z_{in}}, \theta)$

Need:

- $\dfrac{\partial J}{\partial z_{in}} = \dfrac{\partial J}{\partial z_{out}} \dfrac{\partial f}{\partial z_{in}}$
- $\dfrac{\partial J}{\partial \theta} = \dfrac{\partial J}{\partial z_{out}} \dfrac{\partial f}{\partial \theta}$



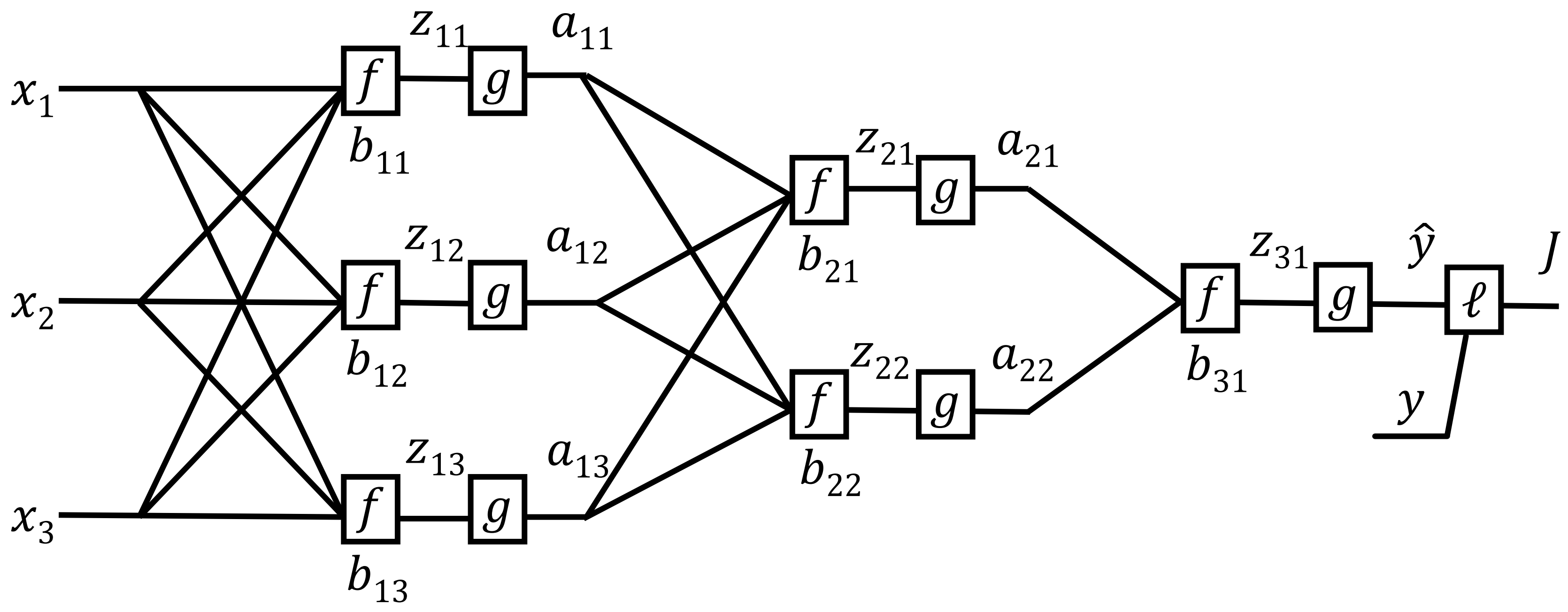


$z_{in}$
$\dfrac{\partial J}{\partial z_{in}}$
Layer $\mathrm{z_{out}} = f(\mathrm{z_{in}}, \theta)$
$\dfrac{\partial f}{\partial z_{in}}$
$\dfrac{\partial f}{\partial \theta}$
$z_{out}$
$\dfrac{\partial J}{\partial z_{out}}$
$\theta$
$\dfrac{\partial J}{\partial \theta}$

## Optimization

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(b_{3,1} + \sum_k w_{3,1,k}\ g\left(b_{2,k} + \sum_i w_{2,k,i}\ g\left(b_{1,i} + \sum_j w_{1,i,j}\ x_j\right)\right)\right)$$

Tons of repeated partial derivatives



$$\frac{\partial J}{\partial b_{3,1}} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial a_{3,1}}{\partial z_{3,1}} \frac{\partial z_{3,1}}{\partial b_{3,1}}$$

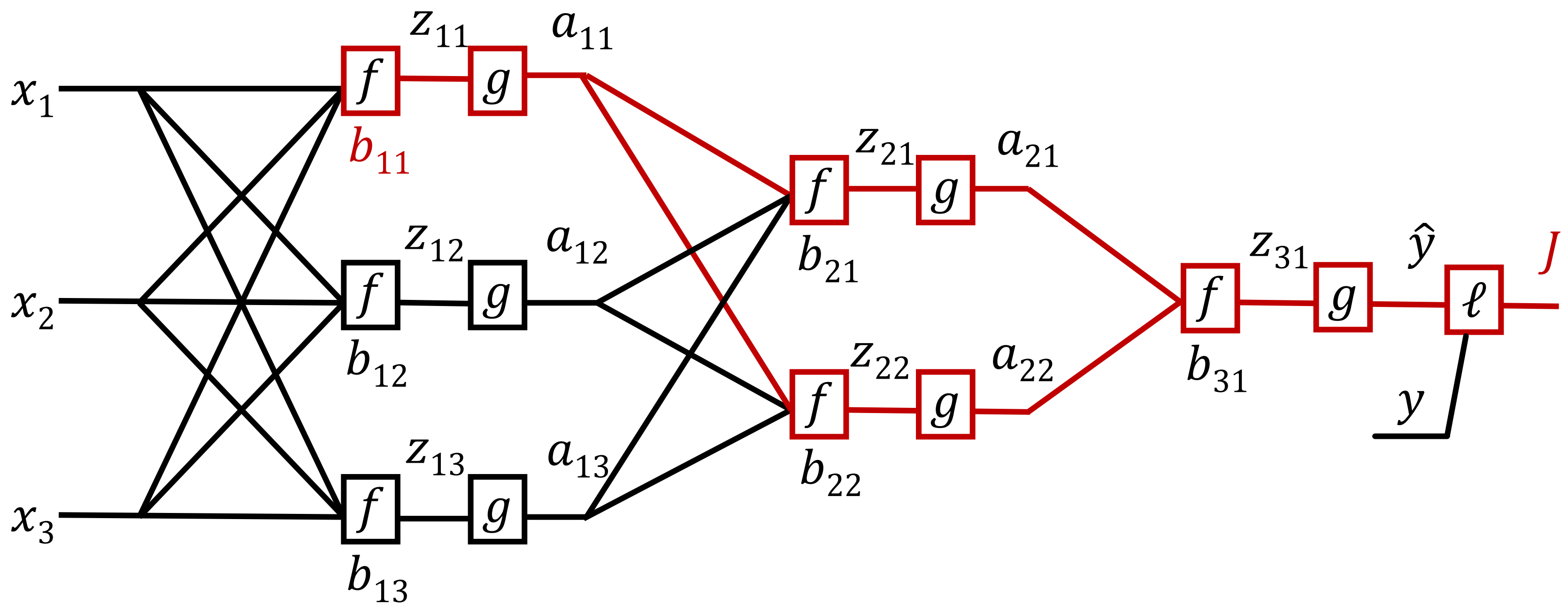
$$\frac{\partial J}{\partial b_{2,i}} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial a_{3,1}}{\partial z_{3,1}} \frac{\partial z_{3,1}}{\partial a_{2,i}} \frac{\partial a_{2,i}}{\partial z_{2,i}} \frac{\partial z_{2,i}}{\partial b_{2,i}}$$

$$\frac{\partial J}{\partial b_{1,i}} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial a_{3,1}}{\partial z_{3,1}} \frac{\partial z_{3,1}}{\partial \mathbf{a}_2} \frac{\partial \mathbf{a}_2}{\partial \mathbf{z}_2} \frac{\partial \mathbf{z}_2}{\partial a_{1,i}} \frac{\partial a_{1,i}}{\partial z_{1,i}} \frac{\partial z_{1,i}}{\partial b_{1,i}}$$

## Optimization

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(b_{3,1} + \sum_k w_{3,1,k}\; g\left(b_{2,k} + \sum_i w_{2,k,i}\; g\left(b_{1,i} + \sum_j w_{1,i,j}\; x_j\right)\right)\right)$$

### Tons of repeated partial derivatives



$$\frac{\partial J}{\partial b_{3,1}} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial a_{3,1}}{\partial z_{3,1}} \frac{\partial z_{3,1}}{\partial b_{3,1}}$$

$$\frac{\partial J}{\partial b_{2,i}} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial a_{3,1}}{\partial z_{3,1}} \frac{\partial z_{3,1}}{\partial a_{2,i}} \frac{\partial a_{2,i}}{\partial z_{2,i}} \frac{\partial z_{2,i}}{\partial b_{2,i}}$$

$$\frac{\partial J}{\partial b_{1,i}} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial a_{3,1}}{\partial z_{3,1}} \frac{\partial z_{3,1}}{\partial \mathbf{a}_2} \frac{\partial \mathbf{a}_2}{\partial \mathbf{z}_2} \frac{\partial \mathbf{z}_2}{\partial a_{1,i}} \frac{\partial a_{1,i}}{\partial z_{1,i}} \frac{\partial z_{1,i}}{\partial b_{1,i}}$$

## Calculus Time-out

## Multivariate-chain rule

## Multivariable Chain Rule

$g_1(x) = 3x$

$g_2(x) = 5x$

$f(z_1, z_2) = 2z_1 + 7z_2$

$y = f\big(g_1(x), g_2(x)\big)$

# Exercise: Multivariable Chain Rule

$$\boxed{\frac{df}{dx} = \frac{\partial f}{\partial g_1}\frac{\partial g_1}{\partial x} + \frac{\partial f}{\partial g_2}\frac{\partial g_2}{\partial x}}$$

$z_1 = g_1(x) = \sin(x)$

$z_2 = g_2(x) = x^3$

$y = f(z_1, z_2) = z_1^4 e^{z_2} + 5z_1 + 7z_2$

# Exercise: Multivariable Chain Rule

$$\boxed{\frac{df}{dx} = \frac{\partial f}{\partial g_1}\frac{\partial g_1}{\partial x} + \frac{\partial f}{\partial g_2}\frac{\partial g_2}{\partial x}}$$

$z_1 = g_1(x) = \sin(x)$

$z_2 = g_2(x) = x^3$

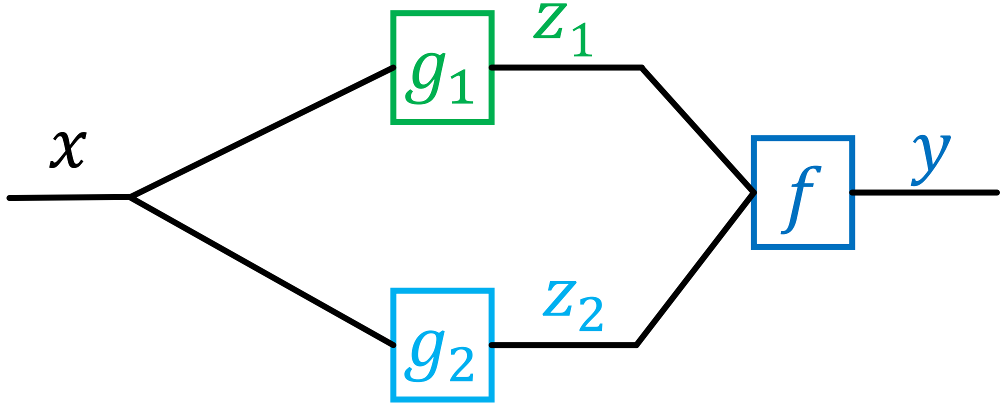
$y = f(z_1, z_2) = z_1^4 e^{z_2} + 5z_1 + 7z_2$

$dg_1/dx = \cos(x)$

$dg_2/dx = 3x^2$

$\partial f/\partial z_1 = 4z_1^3 e^{z_2} + 5$

$\partial f/\partial z_2 = z_1^4 e^{z_2} + 7$



$$\frac{df}{dx} = \frac{\partial f}{\partial g_1}\frac{\partial g_1}{\partial x} + \frac{\partial f}{\partial g_2}\frac{\partial g_2}{\partial x}$$

$$= \left(4z_1^3 e^{z_2} + 5\right)\cos(x) + \left(z_1^4 e^{z_2} + 7\right)3x^2$$

## Calculus Chain Rule

**Scalar:**

$y = f(z)$

$z = g(x)$

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

**Multivariate:**

$y = f(\mathbf{z})$

$\mathbf{z} = g(x)$

$$\frac{dy}{dx} = \sum_j \frac{\partial y}{\partial z_j}\frac{\partial z_j}{\partial x}$$

**Multivariate:**

$\mathbf{y} = f(\mathbf{z})$

$\mathbf{z} = g(\mathbf{x})$

$$\frac{dy_{\mathrm{i}}}{dx_{\mathrm{k}}} = \sum_j \frac{\partial y_i}{\partial z_j}\frac{\partial z_j}{\partial x_{\mathrm{k}}}$$

# Calculus Chain Rule

## Scalar:

$y = f(z)$

$z = g(x)$

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

## Multivariate:

$y = f(\mathbf{z})$

$\mathbf{z} = g(x)$

$$\frac{dy}{dx} = \sum_j \frac{\partial y}{\partial z_j}\frac{\partial z_j}{\partial x}$$

## Multivariate:

$\mathbf{y} = f(\mathbf{z})$

$\mathbf{z} = g(\mathbf{x})$

$$\frac{dy_{\mathrm{i}}}{dx_{\mathrm{k}}} = \sum_j \frac{\partial y_i}{\partial z_j}\frac{\partial z_j}{\partial x_{\mathrm{k}}}$$

# Multivariable Chain Rule

| | Numerator layout | Denominator layout | Notes |
|---|---|---|---|
| $\frac{d}{dt}\ f\left(g(t), h(t)\right)$ | $\frac{df}{dg}\frac{dg}{dt} + \frac{df}{dh}\frac{dh}{dt}$ | Same | $f : (\mathbb{R} \times \mathbb{R}) \to \mathbb{R},\ t \in \mathbb{R}$ <br> $g : \mathbb{R} \to \mathbb{R},\ h : \mathbb{R} \to \mathbb{R}$ |
| $\frac{d}{dt}\ f\left(g_1(t), \ldots, g_N(t)\right)$ | $\sum_{i=1}^{N} \frac{df}{dg_i}\frac{dg_i}{dt}$ | Same | $h : (\mathbb{R} \times \cdots \times \mathbb{R}) \to \mathbb{R},\ t \in \mathbb{R}$ <br> $f_i : \mathbb{R} \to \mathbb{R}\ \ \forall i \in \{1, \ldots, N\}$ |
| $\frac{d}{dt}\ f(\mathbf{g}(t))$ | $\frac{\partial f}{\partial \mathbf{g}}\frac{\partial \mathbf{g}}{\partial t}$ | $\frac{\partial \mathbf{g}}{\partial t}\frac{\partial f}{\partial \mathbf{g}}$ | $f : \mathbb{R}^N \to \mathbb{R},\ \mathbf{g} : \mathbb{R} \to \mathbb{R}^N$ <br> $t \in \mathbb{R}$ |
| $\frac{\partial}{\partial \mathbf{v}}\ f(\mathbf{g}(\mathbf{v}))$ | $\frac{\partial f}{\partial \mathbf{g}}\frac{\partial \mathbf{g}}{\partial \mathbf{v}}$ | $\frac{\partial \mathbf{g}}{\partial \mathbf{v}}\frac{\partial f}{\partial \mathbf{g}}$ | $f : \mathbb{R}^N \to \mathbb{R},\ \mathbf{g} : \mathbb{R}^M \to \mathbb{R}^N$ <br> $\mathbf{v} \in \mathbb{R}^M$ |
| $\frac{\partial}{\partial \mathbf{v}}\ f(\mathbf{g}(\mathbf{v}), \mathbf{h}(\mathbf{v}))$ | $\frac{\partial f}{\partial \mathbf{g}}\frac{\partial \mathbf{g}}{\partial \mathbf{v}} + \frac{\partial f}{\partial \mathbf{h}}\frac{\partial \mathbf{h}}{\partial \mathbf{v}}$ | $\frac{\partial \mathbf{g}}{\partial \mathbf{v}}\frac{\partial f}{\partial \mathbf{g}} + \frac{\partial \mathbf{h}}{\partial \mathbf{v}}\frac{\partial f}{\partial \mathbf{h}}$ | $h : \mathbb{R}^K \times \mathbb{R}^N \to \mathbb{R},\ \mathbf{v} \in \mathbb{R}^M$ <br> $f : \mathbb{R}^M \to \mathbb{R}^K,\ \mathbf{g} : \mathbb{R}^M \to \mathbb{R}^N$ |

## Backpropagation (vector version)

# Generic Layer Implementation (updated)

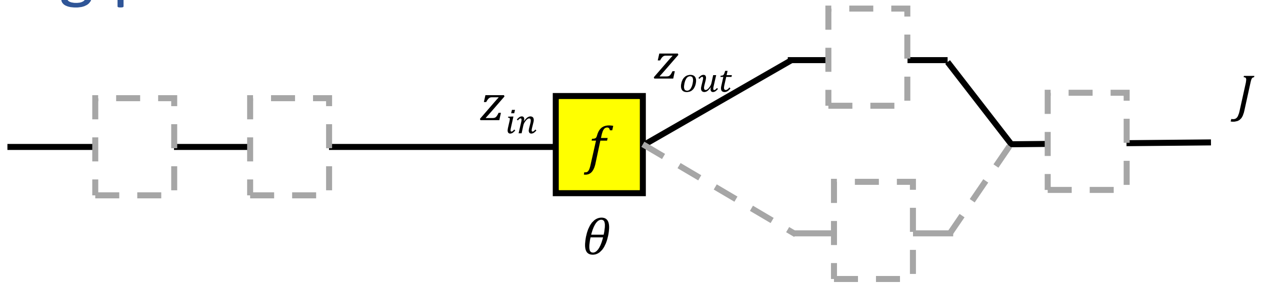
Compute derivatives per layer, utilizing previous derivatives

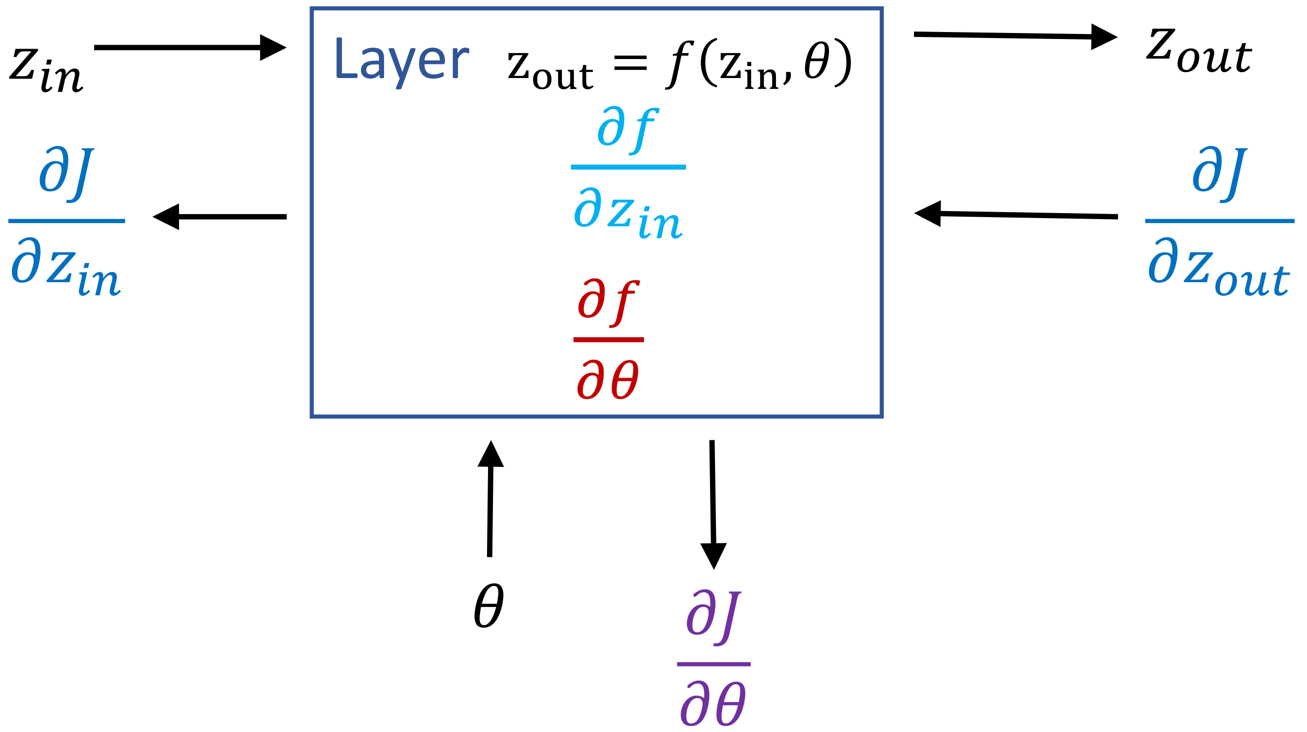
Objective: $J(\theta)$

Arbitrary layer: $\mathrm{z_{out}} = f(\mathrm{z_{in}}, \theta)$

Need:

- $\frac{\partial J}{\partial z_{in}} \mathrel{+}= \frac{\partial J}{\partial z_{out}} \frac{\partial f}{\partial z_{in}}$
- $\frac{\partial J}{\partial \theta} \mathrel{+}= \frac{\partial J}{\partial z_{out}} \frac{\partial f}{\partial \theta}$





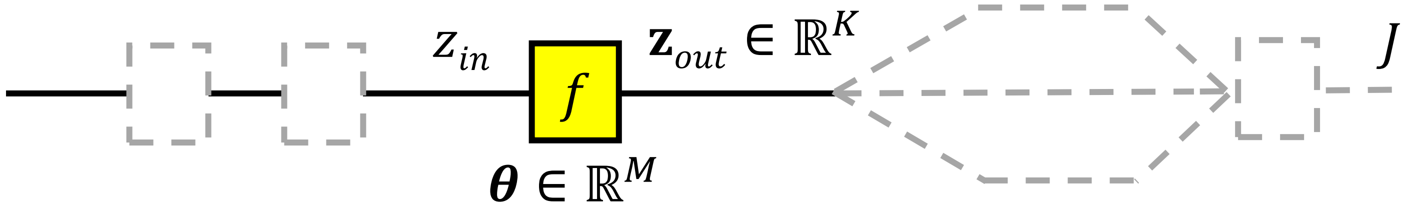
$z_{in}$
Layer $\mathrm{z_{out}} = f(\mathrm{z_{in}}, \theta)$
$\frac{\partial f}{\partial z_{in}}$
$\frac{\partial f}{\partial \theta}$
$z_{out}$
$\frac{\partial J}{\partial z_{in}}$
$\frac{\partial J}{\partial z_{out}}$
$\theta$
$\frac{\partial J}{\partial \theta}$

# Generic Layer Implementation (vector output version)

Compute derivatives per layer, utilizing previous derivatives

Objective: $J(\theta)$

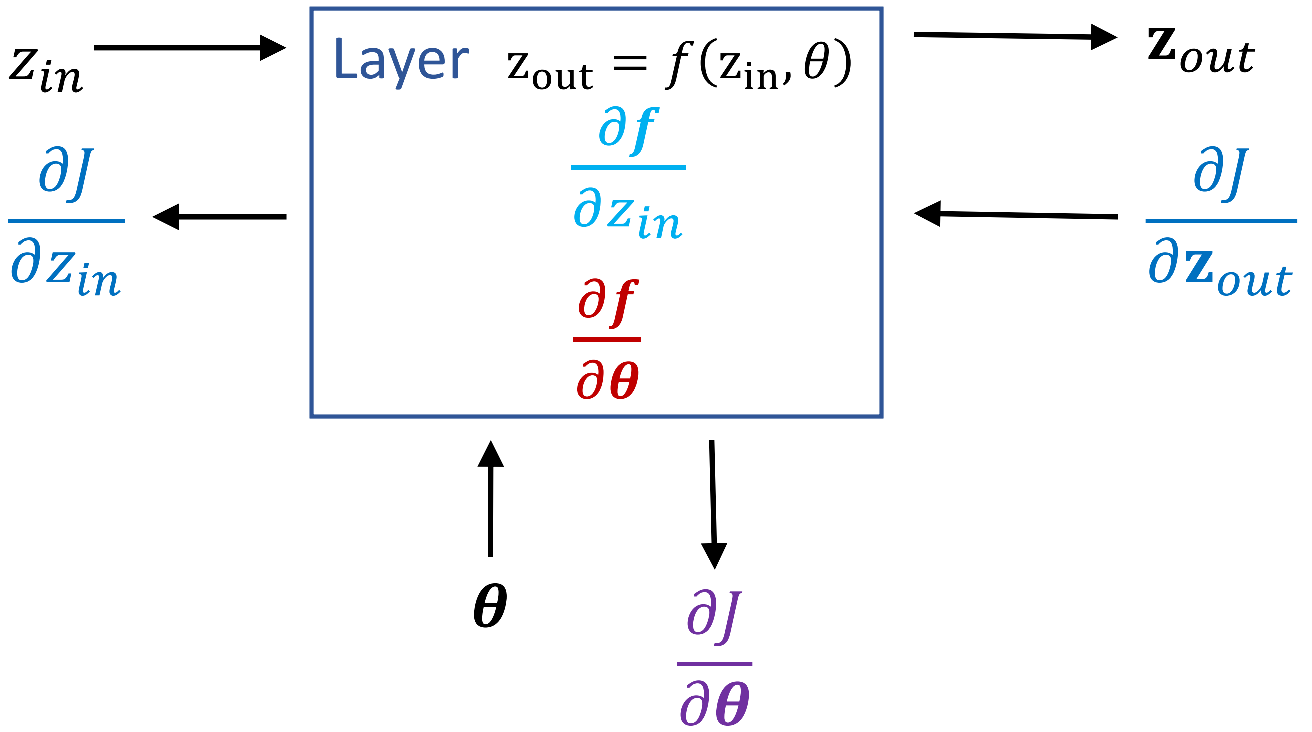
Arbitrary layer: $\mathrm{z}_{\mathrm{out}} = f(\mathrm{z}_{\mathrm{in}}, \theta)$



$z_{in}$
$f$
$\mathbf{z}_{out} \in \mathbb{R}^K$
$\boldsymbol{\theta} \in \mathbb{R}^M$
$J$

Need:

- $\dfrac{\partial J}{\partial z_{in}} =$
- $\dfrac{\partial J}{\partial \theta_j} =$



$z_{in}$
Layer $\mathrm{z}_{\mathrm{out}} = f(\mathrm{z}_{\mathrm{in}}, \theta)$
$\dfrac{\partial \boldsymbol{f}}{\partial z_{in}}$
$\dfrac{\partial \boldsymbol{f}}{\partial \boldsymbol{\theta}}$
$\mathbf{z}_{out}$
$\dfrac{\partial J}{\partial z_{in}}$
$\dfrac{\partial J}{\partial \mathbf{z}_{out}}$
$\boldsymbol{\theta}$
$\dfrac{\partial J}{\partial \boldsymbol{\theta}}$

# Generic Layer Implementation (vector output version)

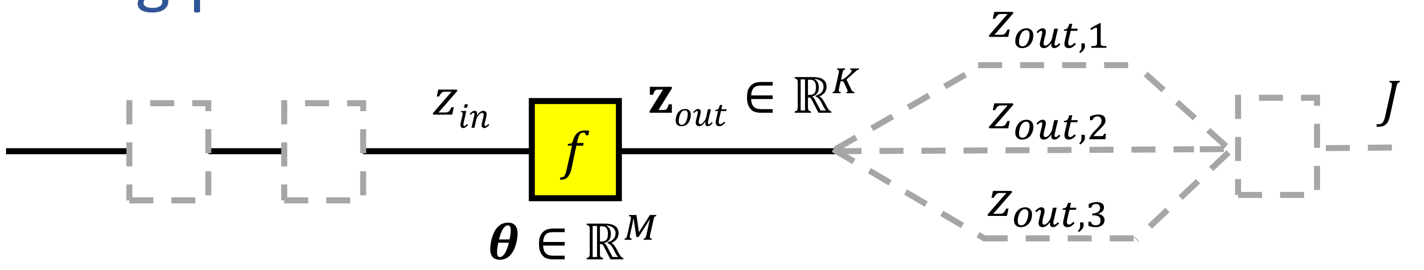
Compute derivatives per layer, utilizing previous derivatives

Objective: $J(\theta)$

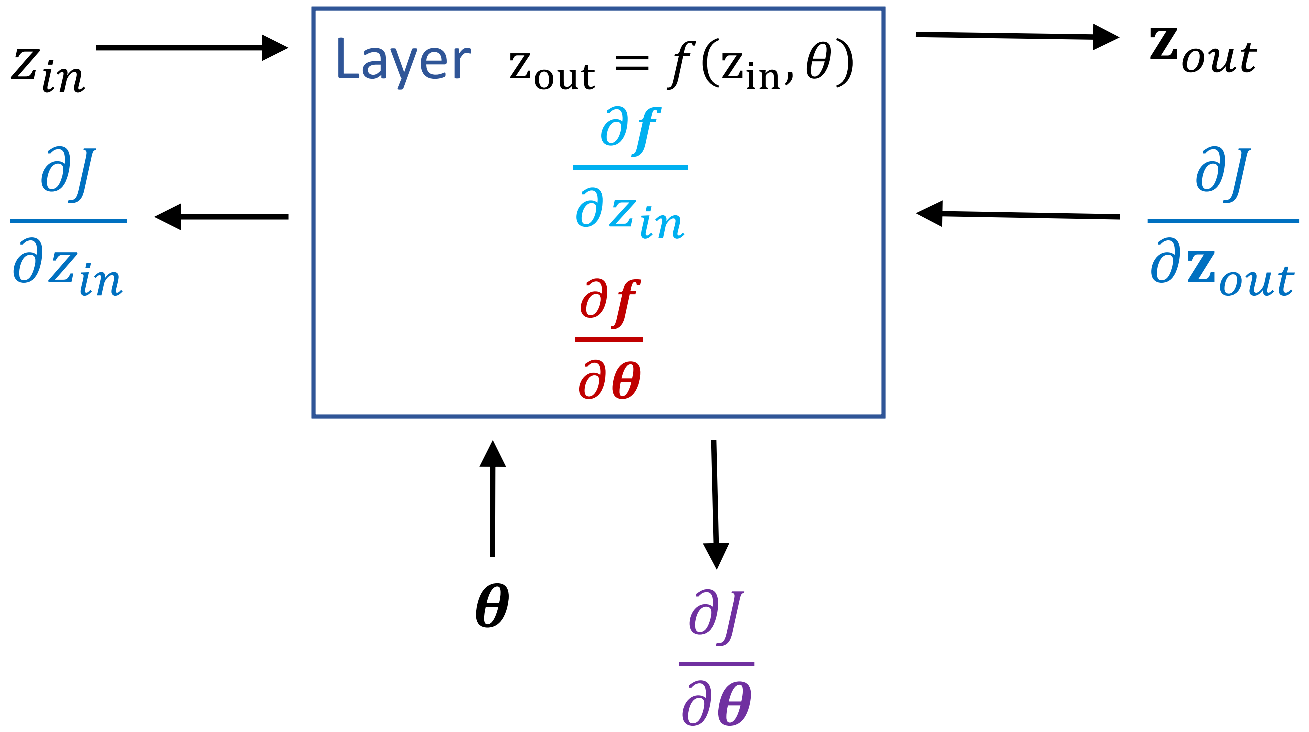
Arbitrary layer: $\mathrm{z_{out}} = f(\mathrm{z_{in}}, \theta)$

Need:

- $\frac{\partial J}{\partial z_{in}} = \sum_k \frac{\partial J}{\partial z_{out,k}} \frac{\partial z_{out,k}}{\partial z_{in}}$
- $\frac{\partial J}{\partial \theta_j} = \sum_k \frac{\partial J}{\partial z_{out,k}} \frac{\partial f}{\partial \theta_j}$

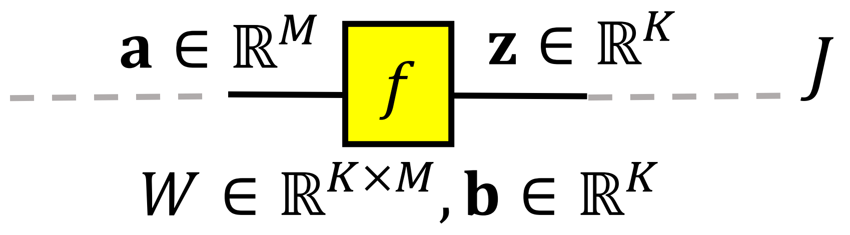


$z_{in}$
$f$
$\mathbf{z}_{out} \in \mathbb{R}^K$
$\boldsymbol{\theta} \in \mathbb{R}^M$
$z_{out,1}$
$z_{out,2}$
$z_{out,3}$
$J$



$z_{in}$
Layer
$\mathrm{z_{out}} = f(\mathrm{z_{in}}, \theta)$
$\frac{\partial \boldsymbol{f}}{\partial z_{in}}$
$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\theta}}$
$\mathbf{z}_{out}$
$\frac{\partial J}{\partial z_{in}}$
$\frac{\partial J}{\partial \mathbf{z}_{out}}$
$\boldsymbol{\theta}$
$\frac{\partial J}{\partial \boldsymbol{\theta}}$

# Linear Layer Implementation



$\mathbf{a} \in \mathbb{R}^M$
$f$
$\mathbf{z} \in \mathbb{R}^K$
$J$
$W \in \mathbb{R}^{K \times M}, \mathbf{b} \in \mathbb{R}^K$

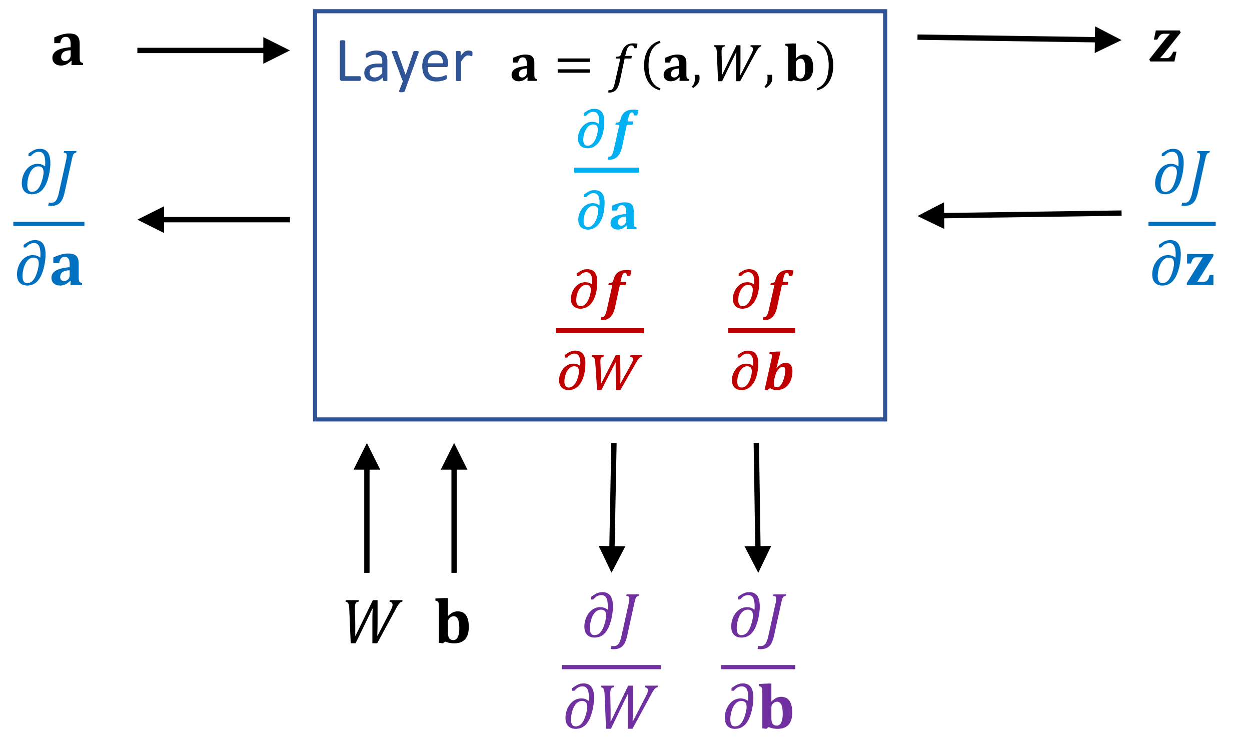
Compute derivatives per layer

Objective: $J(\theta)$

Layer: $\mathbf{z} = f(\mathbf{a}, W, \mathbf{b}) = W\mathbf{a} + \mathbf{b}$

Scalar version (no formatting)

- $\dfrac{\partial J}{\partial a_j} =$
- $\dfrac{\partial J}{\partial W_{i,j}} =$
- $\dfrac{\partial J}{\partial b_i} =$

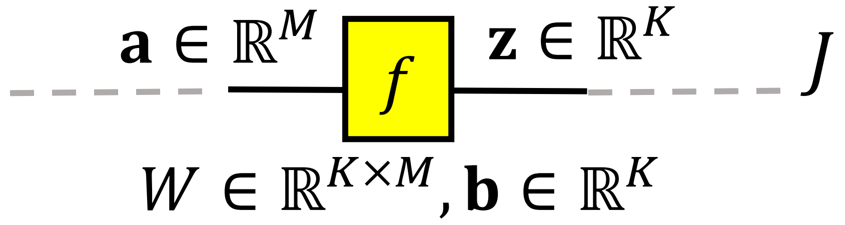


$\mathbf{a}$
$\frac{\partial J}{\partial \mathbf{a}}$
Layer $\mathbf{a} = f(\mathbf{a}, W, \mathbf{b})$
$\frac{\partial \boldsymbol{f}}{\partial \mathbf{a}}$
$\frac{\partial \boldsymbol{f}}{\partial W}$
$\frac{\partial \boldsymbol{f}}{\partial \mathbf{b}}$
$\mathbf{z}$
$\frac{\partial J}{\partial \mathbf{z}}$
$W$
$\mathbf{b}$
$\frac{\partial J}{\partial W}$
$\frac{\partial J}{\partial \mathbf{b}}$

$$\mathbf{z} = W\mathbf{a} + \mathbf{b}$$

$$z_k = b_k + \sum_j W_{k,j} a_j$$

# Linear Layer Implementation


$\mathbf{a} \in \mathbb{R}^M$
$f$
$\mathbf{z} \in \mathbb{R}^K$
$J$
$W \in \mathbb{R}^{K\times M}, \mathbf{b} \in \mathbb{R}^K$

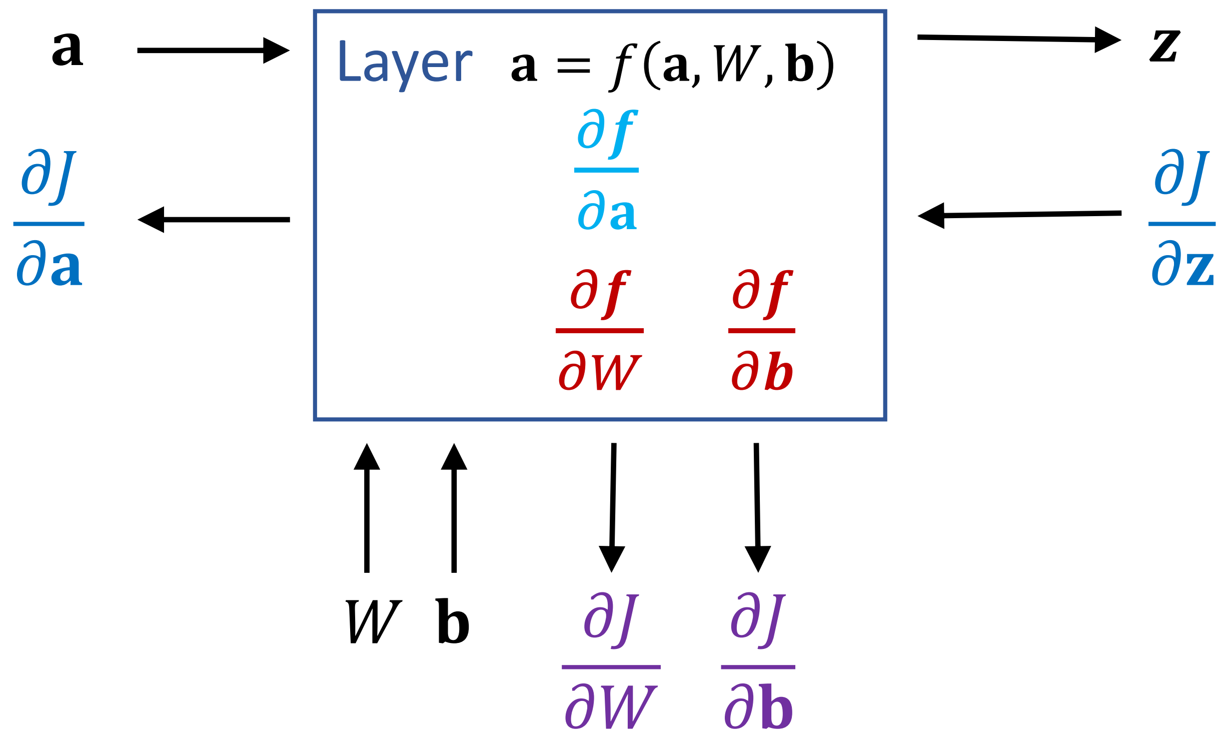
Compute derivatives per layer

Objective: $J(\theta)$

Layer: $\mathbf{z} = f(\mathbf{a}, W, \mathbf{b}) = W\mathbf{a} + \mathbf{b}$

Scalar version (no formatting)

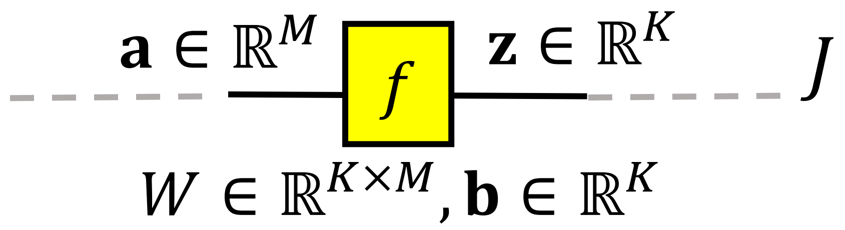
- $\frac{\partial J}{\partial a_j} = \sum_k \frac{\partial J}{\partial z_k} \frac{\partial z_k}{\partial a_j}$
- $\frac{\partial J}{\partial W_{i,j}} = \sum_k \frac{\partial J}{\partial z_k} \frac{\partial z_k}{\partial W_{i,j}}$
- $\frac{\partial J}{\partial b_i} = \sum_k \frac{\partial J}{\partial z_k} \frac{\partial z_k}{\partial b_i}$


$\mathbf{a}$
$\frac{\partial J}{\partial \mathbf{a}}$
Layer $\mathbf{a} = f(\mathbf{a}, W, \mathbf{b})$
$\frac{\partial \boldsymbol{f}}{\partial \mathbf{a}}$
$\frac{\partial \boldsymbol{f}}{\partial W}$
$\frac{\partial \boldsymbol{f}}{\partial \mathbf{b}}$
$\mathbf{z}$
$\frac{\partial J}{\partial \mathbf{z}}$
$W$
$\mathbf{b}$
$\frac{\partial J}{\partial W}$
$\frac{\partial J}{\partial \mathbf{b}}$

$$\mathbf{z} = W\mathbf{a} + \mathbf{b}$$

$$z_k = b_k + \sum_j W_{k,j} a_j$$

# Linear Layer Implementation



$\mathbf{a} \in \mathbb{R}^M$
$f$
$\mathbf{z} \in \mathbb{R}^K$
$J$
$W \in \mathbb{R}^{K\times M}, \mathbf{b} \in \mathbb{R}^K$

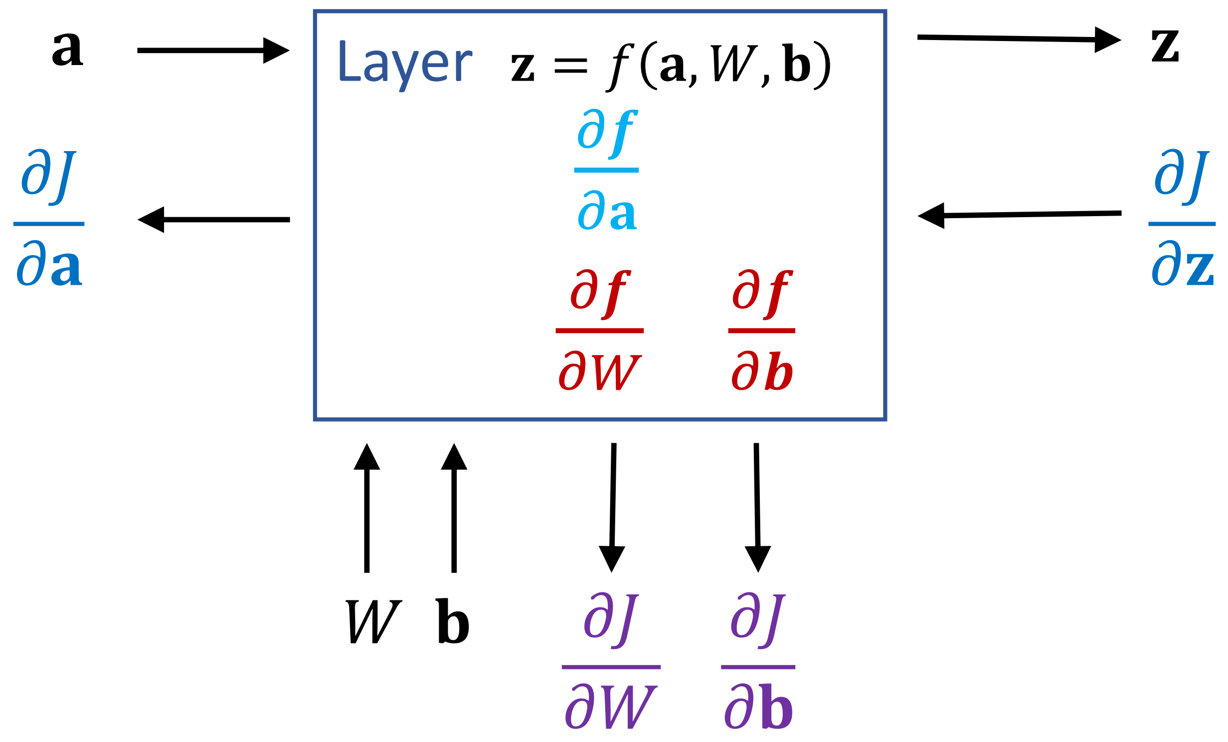
## Compute derivatives per layer

Objective: $J(\theta)$

Layer: $\mathbf{z} = f(\mathbf{a}, W, \mathbf{b}) = W\mathbf{a} + \mathbf{b}$

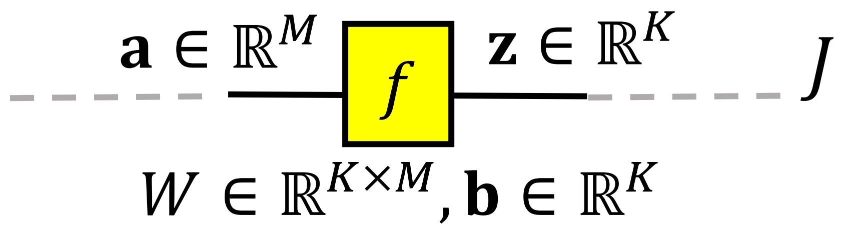
(Denominator format)

- $\frac{\partial J}{\partial \mathbf{a}}$
- $\frac{\partial J}{\partial W}$
- $\frac{\partial J}{\partial \mathbf{b}}$



$\mathbf{a}$
$\frac{\partial J}{\partial \mathbf{a}}$
Layer $\mathbf{z} = f(\mathbf{a}, W, \mathbf{b})$
$\frac{\partial \boldsymbol{f}}{\partial \mathbf{a}}$
$\frac{\partial \boldsymbol{f}}{\partial W}$
$\frac{\partial \boldsymbol{f}}{\partial \mathbf{b}}$
$\mathbf{z}$
$\frac{\partial J}{\partial \mathbf{z}}$
$W$
$\mathbf{b}$
$\frac{\partial J}{\partial W}$
$\frac{\partial J}{\partial \mathbf{b}}$

# Linear Layer Implementation

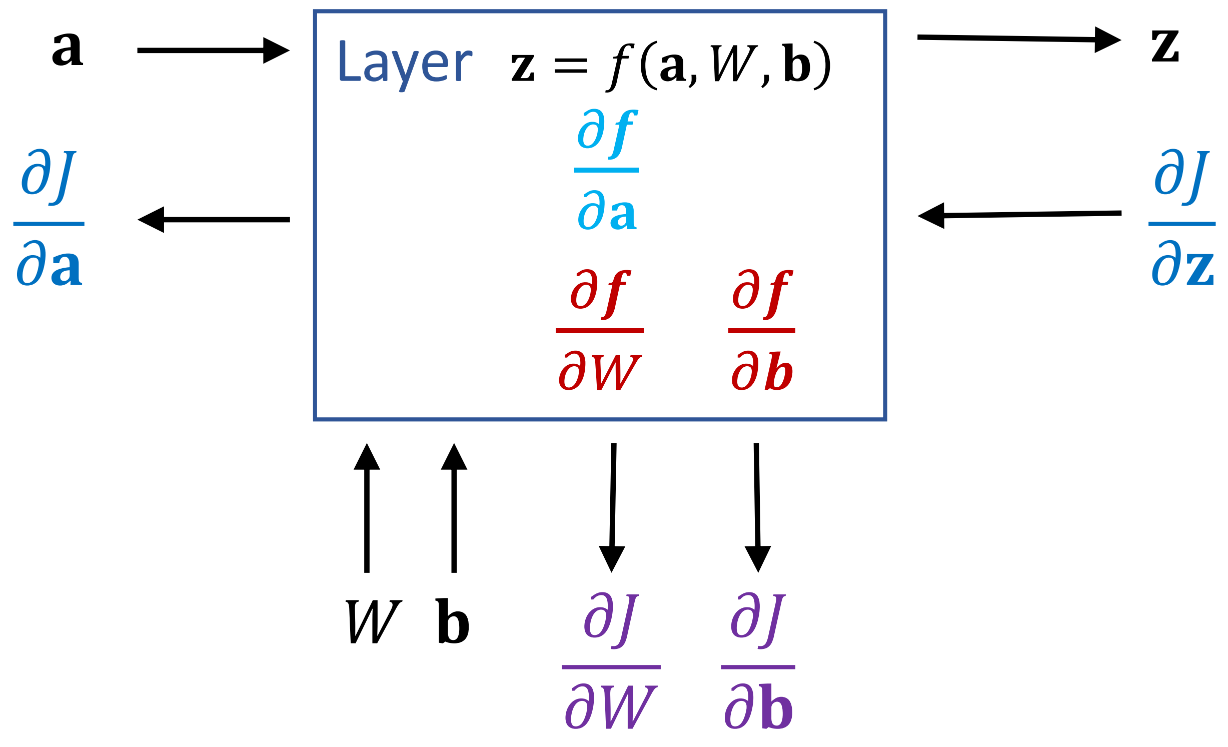

$\mathbf{a} \in \mathbb{R}^M$
$f$
$\mathbf{z} \in \mathbb{R}^K$
$J$
$W \in \mathbb{R}^{K \times M}, \mathbf{b} \in \mathbb{R}^K$

## Compute derivatives per layer

Objective: $J(\theta)$

Layer: $\mathbf{z} = f(\mathbf{a}, W, \mathbf{b}) = W\mathbf{a} + \mathbf{b}$

(Numerator format)


$\mathbf{a}$
$\mathbf{z}$
$\frac{\partial J}{\partial \mathbf{a}}$
$\frac{\partial J}{\partial \mathbf{z}}$
Layer $\mathbf{z} = f(\mathbf{a}, W, \mathbf{b})$
$\frac{\partial \boldsymbol{f}}{\partial \mathbf{a}}$
$\frac{\partial \boldsymbol{f}}{\partial W}$
$\frac{\partial \boldsymbol{f}}{\partial \mathbf{b}}$
$W$
$\mathbf{b}$
$\frac{\partial J}{\partial W}$
$\frac{\partial J}{\partial \mathbf{b}}$

- $\frac{\partial J}{\partial \mathbf{a}}$
- $\frac{\partial J}{\partial W}$
- $\frac{\partial J}{\partial \mathbf{b}}$

## Outline

### Last time: Neural Networks

- Three-neuron network + optimization
- Neural network structure (adding more neurons)
- Forward pass and Backpropagation (scalar version)

### Today: Neural Networks

- Calculus: multi-variate chain rule
- Backpropagation (vector version)
- Neural Network Properties and Intuition

## Neural Network Properties

# Neural Networks Properties

## Practical considerations

- Large number of neurons
  - Danger for overfitting
- Modeling 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.

## Neural Networks Properties

Non-linearity

Fitting complex functions

Fitting any! function (universal approximation theorem)

Overfitting

## Neural Network Properties

Non-linearity

# Non-linearity

Neural network prediction function, $\hat{y} = h(x)$, is definitely not linear. The non-linear activation functions provide this non-linearity

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

Example

What happens with you add to linear functions together?

- Adding two lines? $h(x) = f_1(x, w_1, b_1) + f_2(x, w_2, b_2)$
- Adding two planes?

## Objective Function is Not Convex

| Objective function for… | Convex? | Closed-form solution? |
|---|---|---|
| Linear regression | | |
| Logistic regression | | |
| Neural networks | | |

# Optimization

## Convex function

If $f(\boldsymbol{x})$ is convex, then:

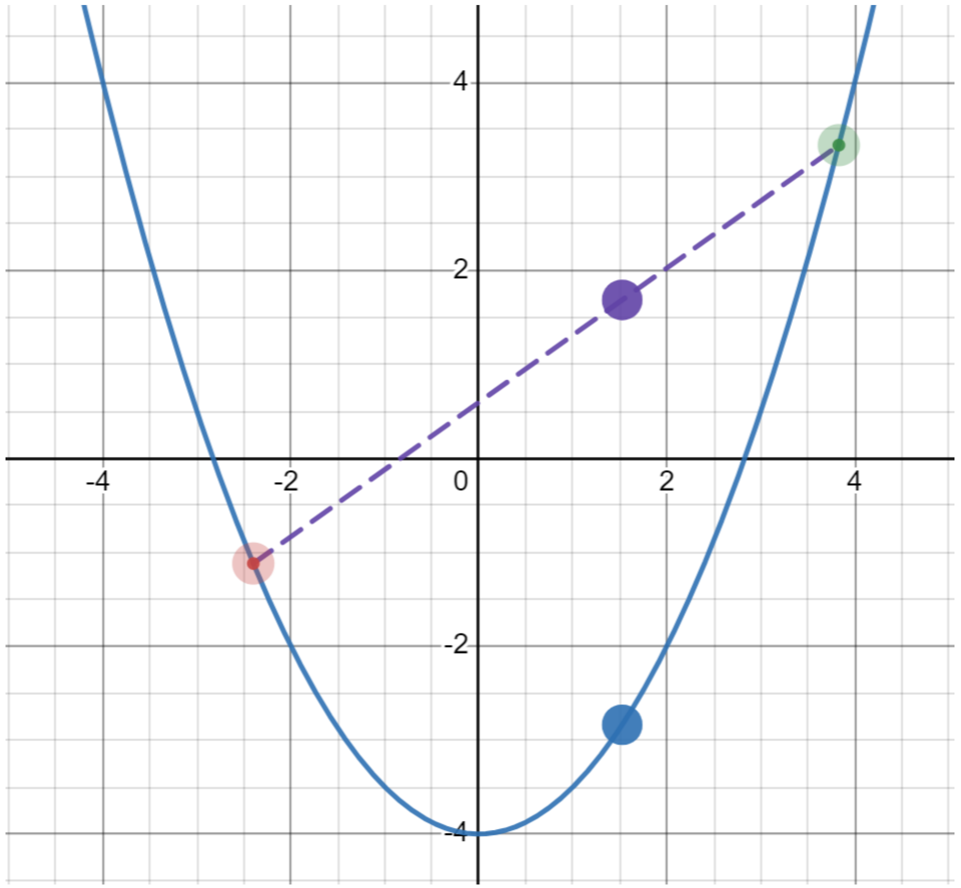
- $f(\alpha \boldsymbol{x} + (1-\alpha)\boldsymbol{z}) \leq \alpha f(\boldsymbol{x}) + (1-\alpha) f(\boldsymbol{z})$
  $\forall\ 0 \leq \alpha \leq 1$

## Convex optimization

If second derivative is $\geq 0$ everywhere then function is convex

If $f(\boldsymbol{x})$ is convex, then:

- Every local minimum is also a global minimum ☺



# Non-linearity

## Prediction function

Neural network prediction function, $\hat{y} = h(x)$, is definitely not linear. The non-linear activation functions provide this non-linearity

## Objective function

Neural network prediction function, $\hat{y} = h(x)$, is definitely not linear. The non-linear activation functions provide this non-linearity

## Just because it can fit any function will it?

- Objective function is non-convex

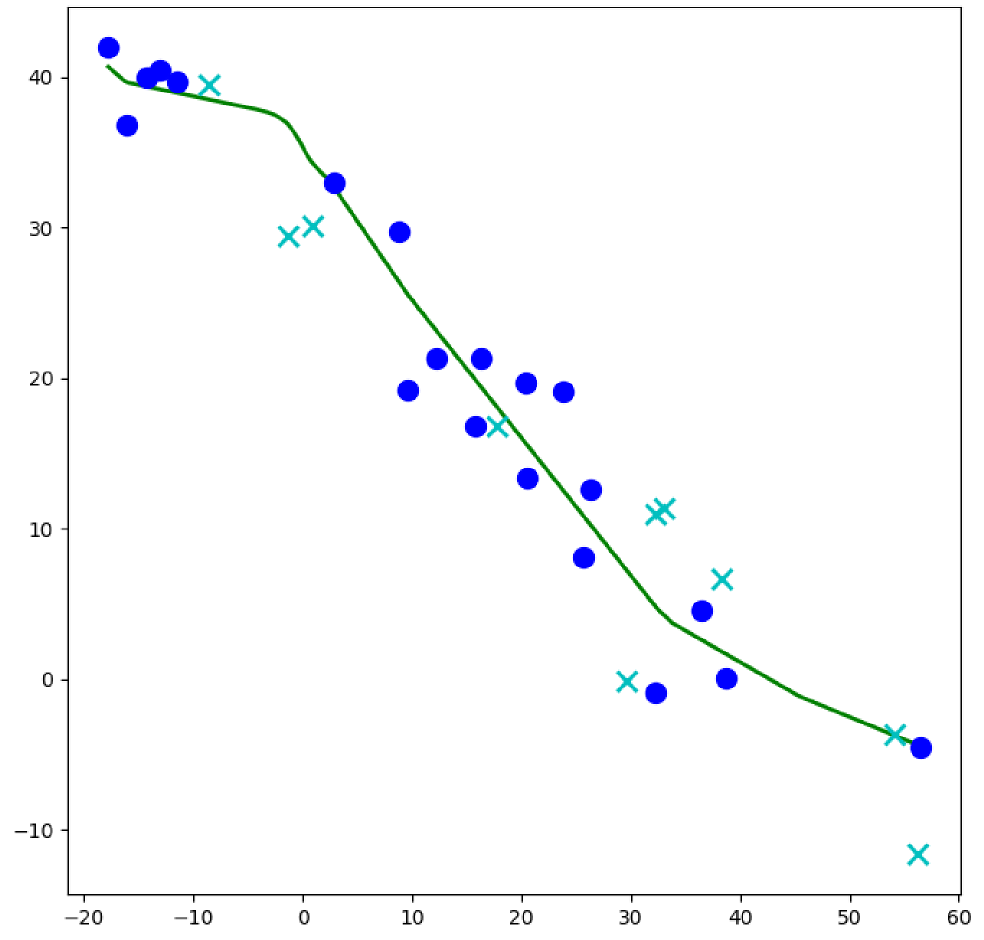
  → it will get stuck in local minima
- *Stochastic* gradient decent take pseudorandom steps

  → Helps pop out of local minima
- Is getting stuck at a local minima necessarily a bad thing??

# Neural Network Properties

Fitting complex functions (with 1-D input)

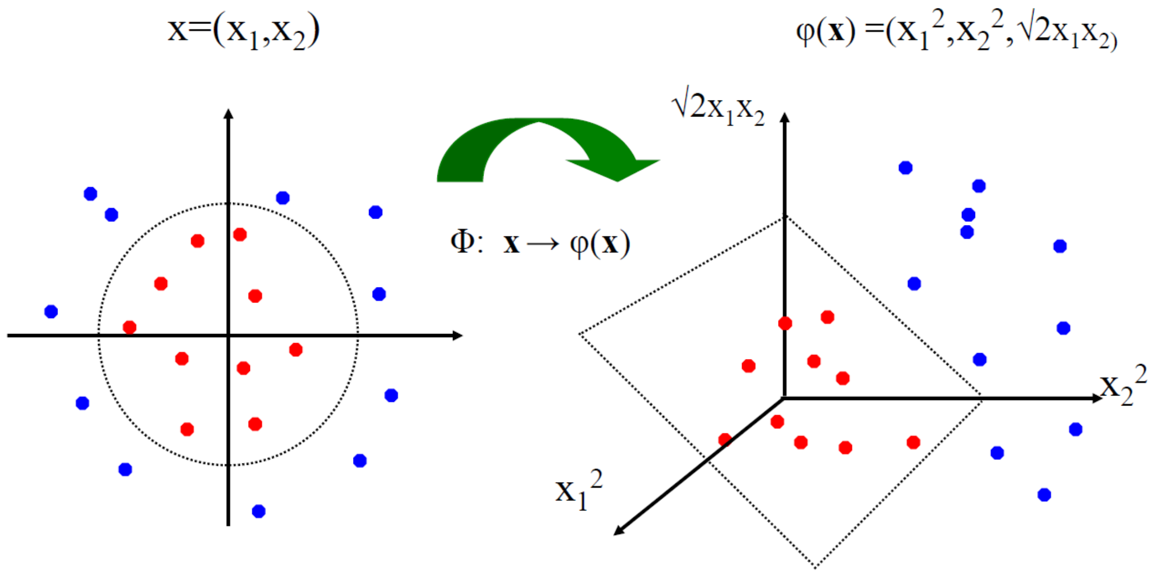
# Network to Approximate a 1-D Function



# Linear Classifiers for Nonlinear Data

Linear classifiers have linear decision boundaries

Feature mapping can convert nonlinear data to higher dimensions

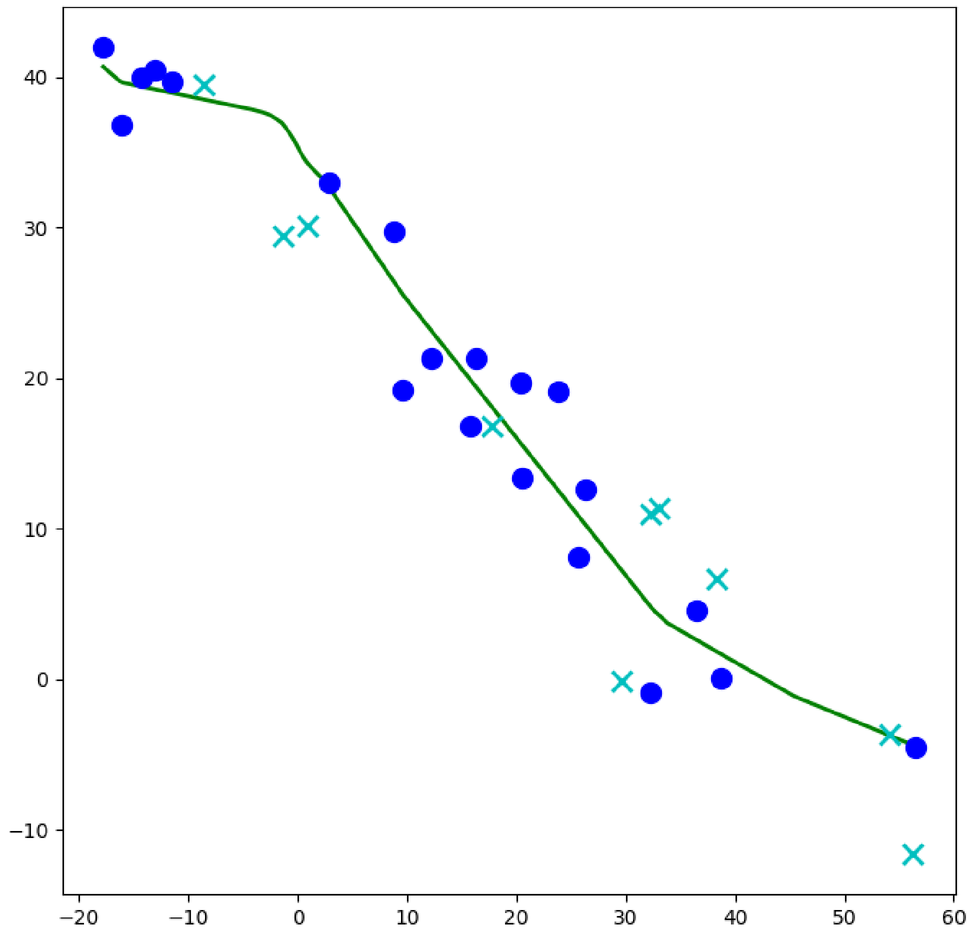


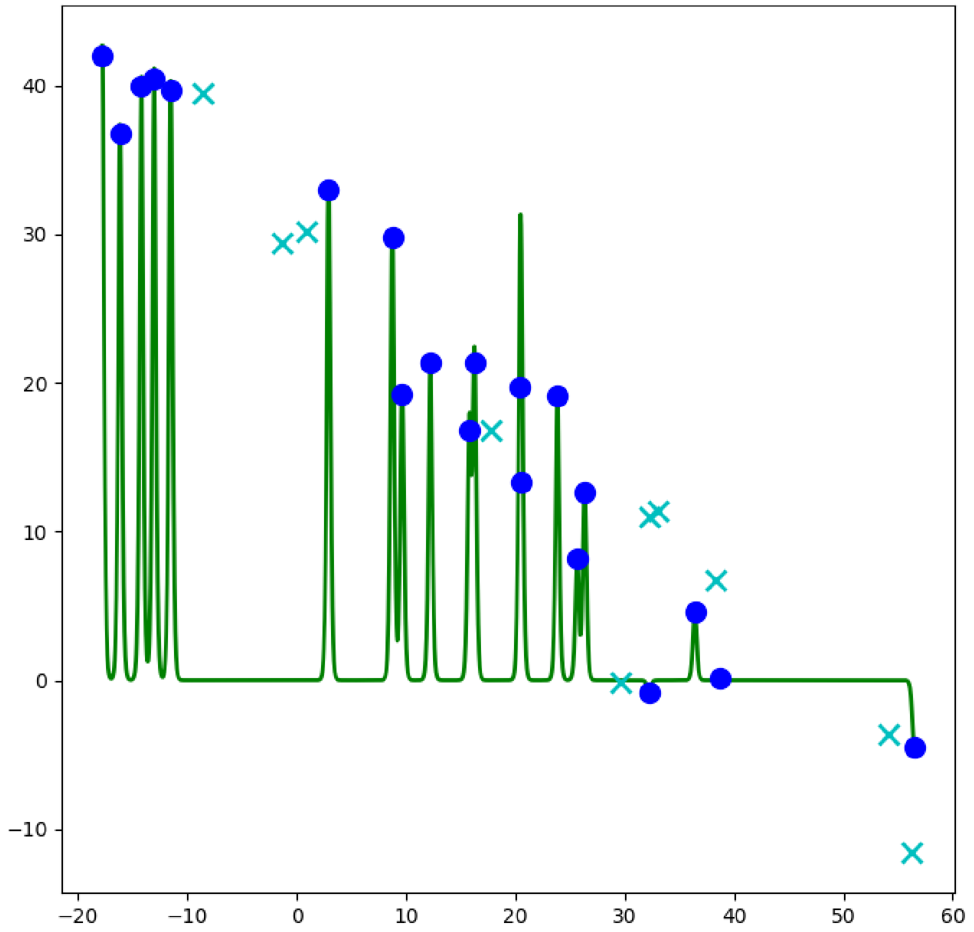
$x=(x_1,x_2)$
$\varphi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$
$\Phi: \mathbf{x} \rightarrow \varphi(\mathbf{x})$
$\sqrt{2}x_1x_2$
$x_2^2$
$x_1^2$

This slide is courtesy of *www.iro.umontreal.ca/~pift6080/documents/papers/**svm**_tutorial.**ppt***

Today, instead of choosing a feature mapping function, we'll use neural networks to learn nonlinear decision boundaries and regression functions

# Network to Approximate a 1-D Function



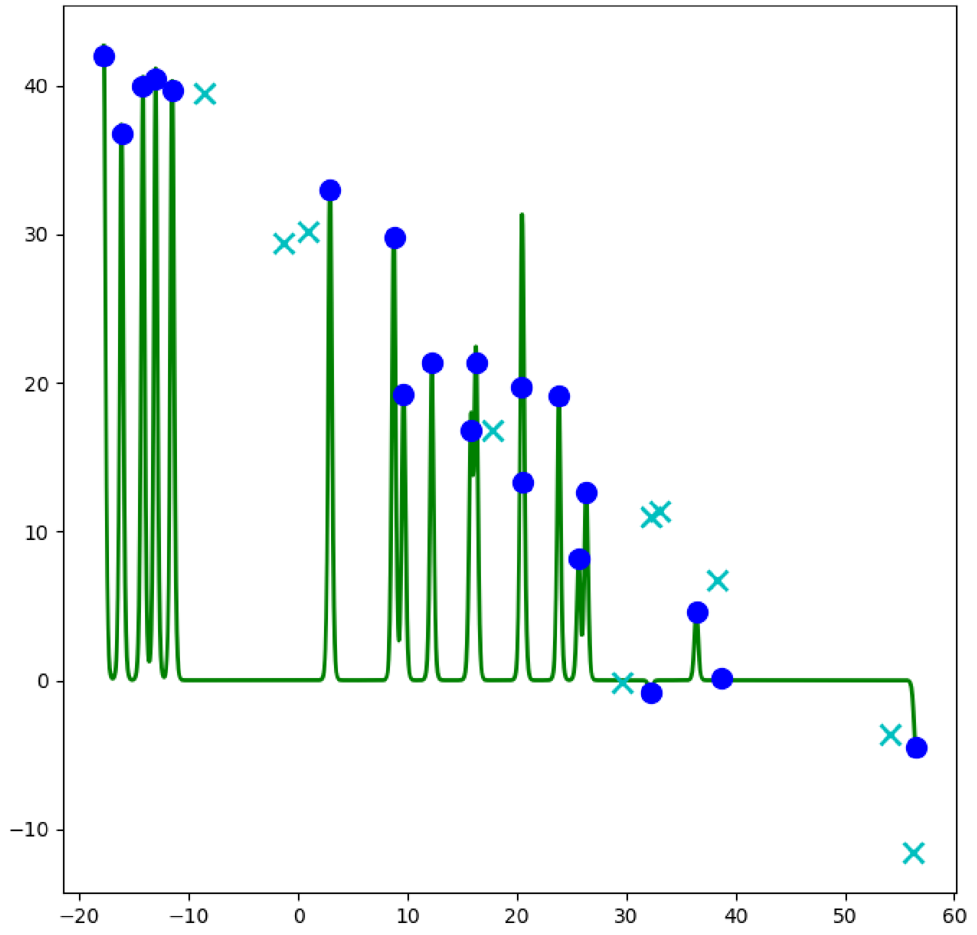


## Network to Approximate a 1-D Function

Design a network to approximate this function using:

Linear, Sigmoid, Step, or ReLU

# Network to Approximate a 1-D Function



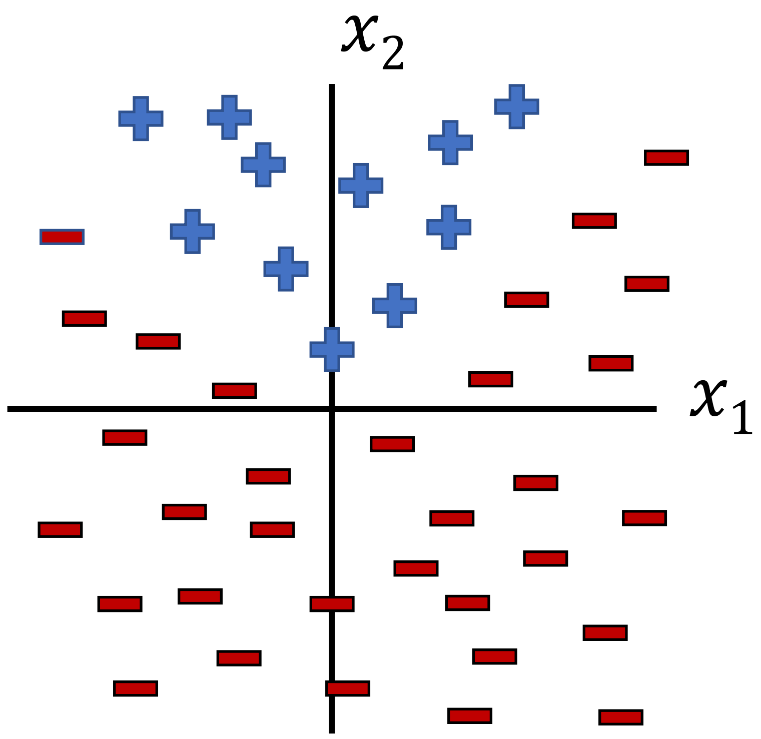
# Neural Network Properties

Fitting complicated functions (with 2-D input)

## Classification Design Challenge

$$h_A(\mathbf{x}) = sign\left(\mathbf{w}_A^\top \mathbf{x} + b_A\right)$$
$$h_B(\mathbf{x}) = sign\left(\mathbf{w}_B^\top \mathbf{x} + b_B\right)$$
$$h_C(\mathbf{z}) = sign\left(\mathbf{w}_C^\top \mathbf{z} + b_C\right)$$

How could you configure three specific perceptrons to classify this data?

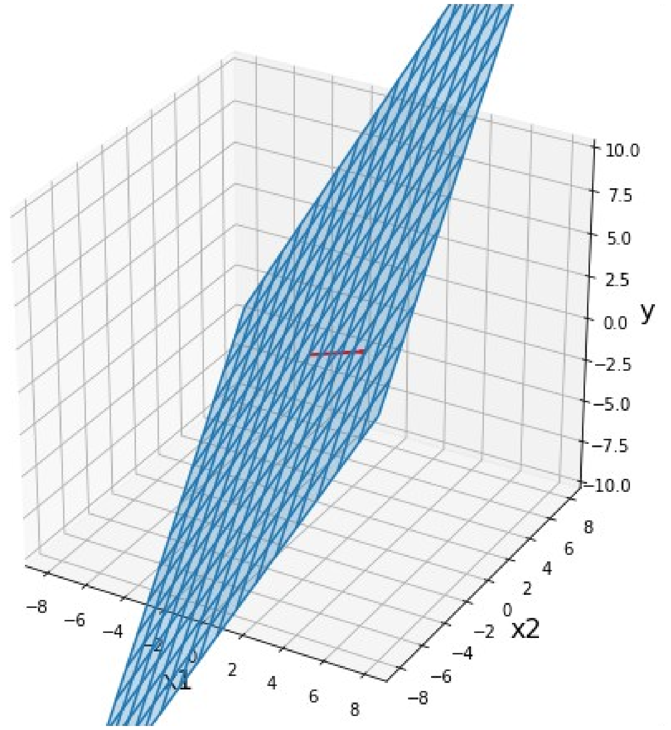


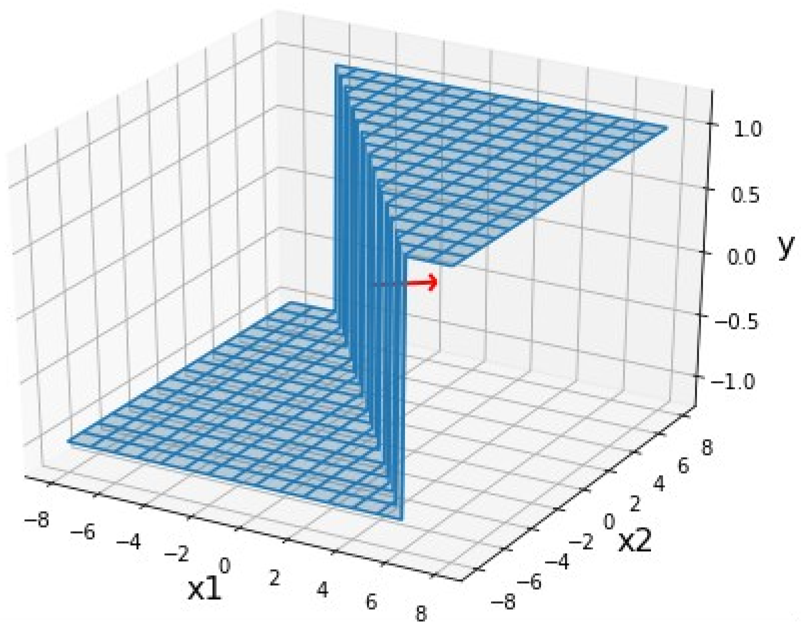
# Perceptron

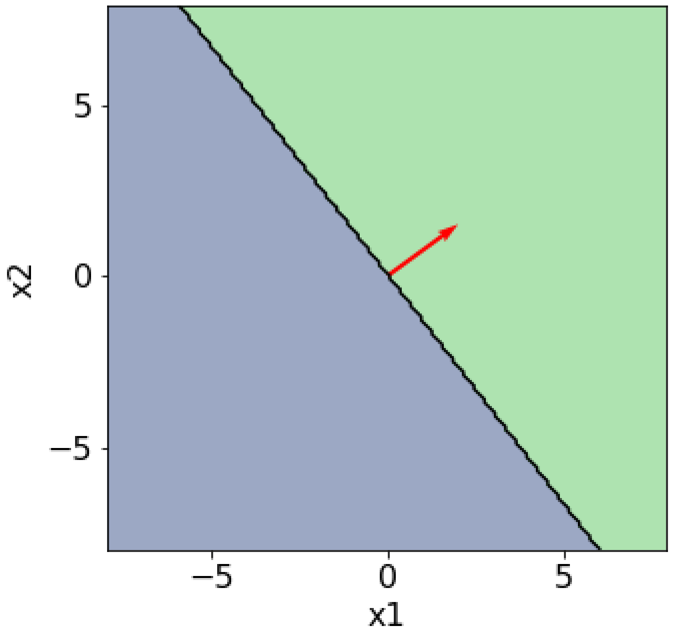
$$sign(\mathbf{z}) = \begin{cases} 1, & if\ z \geq 0 \\ -1, & if\ z < 0 \end{cases}$$

Classification: Hard threshold on linear model

$$h(\mathbf{x}) = sign(\mathbf{w}^\top \mathbf{x} + b)$$

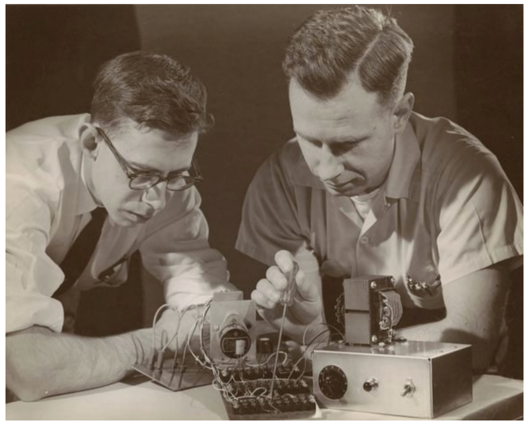


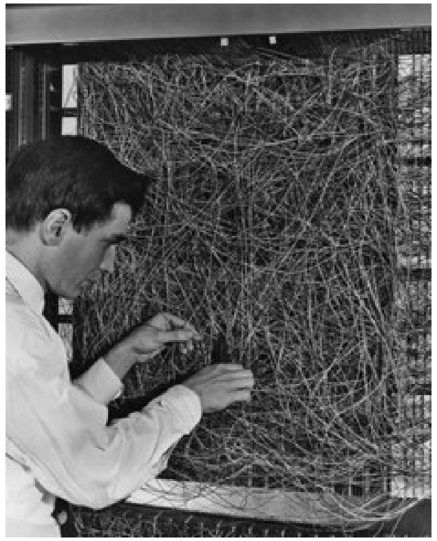




# Perceptron History

## Frank Rosenblatt, 1957





*The New Yorker*, December 6, 1958 P. 44

Talk story about the perceptron, a new electronic brain which hasn't been built, but which has been successfully simulated on the I.B.M. 704. Talk with Dr. Frank Rosenblatt, of the Cornell Aeronautical Laboratory, who is one of the two men who developed the prodigy; the other man is Dr. Marshall C. Yovits, of the Office of Naval Research, in Washington. Dr. Rosenblatt defined the perceptron as the first non-biological object which will achieve an organization o its external environment in a meaningful way. It interacts with its environment, forming concepts that have not been made ready for it by a human agent. If a triangle is held up, the perceptron's eye picks up the image & conveys it along a random succession of lines to the response units, where the image is registered. It can tell the difference betw. a cat and a dog, although it wouldn't be able to tell whether the dog was to theleft or right of the cat. Right now it is of no practical use, Dr. Rosenblatt conceded, but he said that one day it might be useful to send one into outer space to take in impressions for us.

## Exercise

$$h(\mathbf{x}) = sign(\mathbf{w}^\top \mathbf{x} + b)$$

Which of the following perceptron parameters will perfectly classify this data?

$$sign(\mathbf{z}) = \begin{cases} 1, & if\ z \geq 0 \\ -1, & if\ z < 0 \end{cases}$$

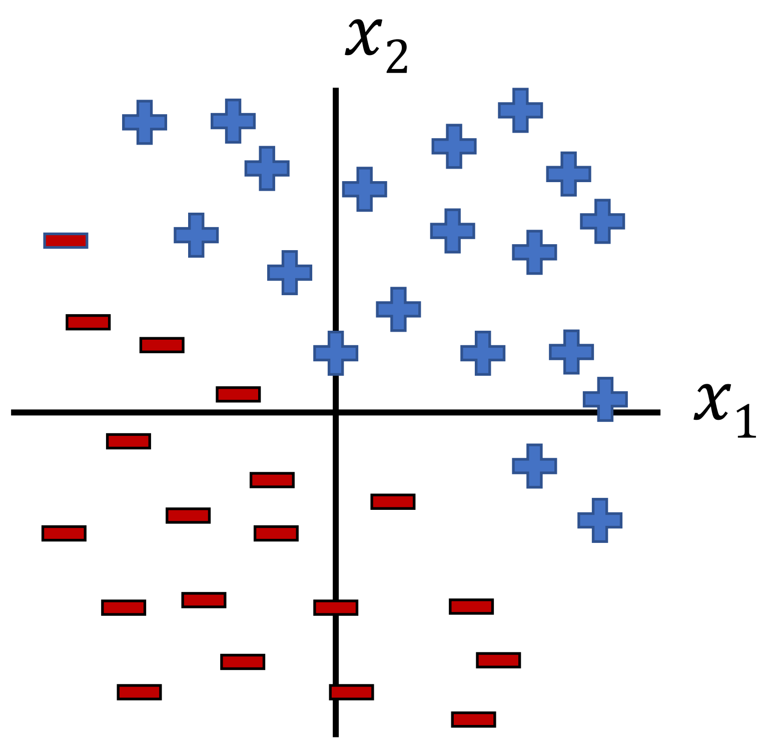
A. $\mathbf{w} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, b = 0$

B. $\mathbf{w} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, b = 0$

C. $\mathbf{w} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, b = 0$

D. $\mathbf{w} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, b = 0$

E. None of the above



# Poll

$$h(\mathbf{x}) = sign(\mathbf{w}^\top \mathbf{x} + b)$$

Which of the following perceptron parameters will perfectly classify this data?

$$sign(\mathbf{z}) = \begin{cases} 1, & if\ z \geq 0 \\ -1, & if\ z < 0 \end{cases}$$

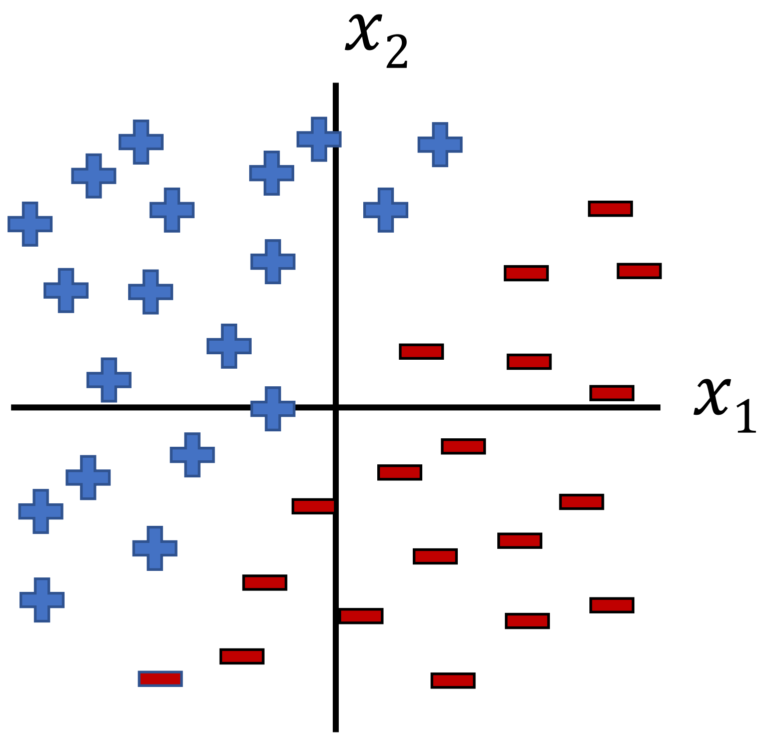
A. $\mathbf{w} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, b = 0$

B. $\mathbf{w} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, b = 0$

C. $\mathbf{w} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, b = 0$

D. $\mathbf{w} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, b = 0$

E. None of the above



# Poll

$$h_C(\mathbf{z}) = sign\left(\mathbf{w}_C^\top \mathbf{z} + b_C\right)$$

Which of the following parameters of $h_C(\mathbf{z})$ will perfectly classify this data?

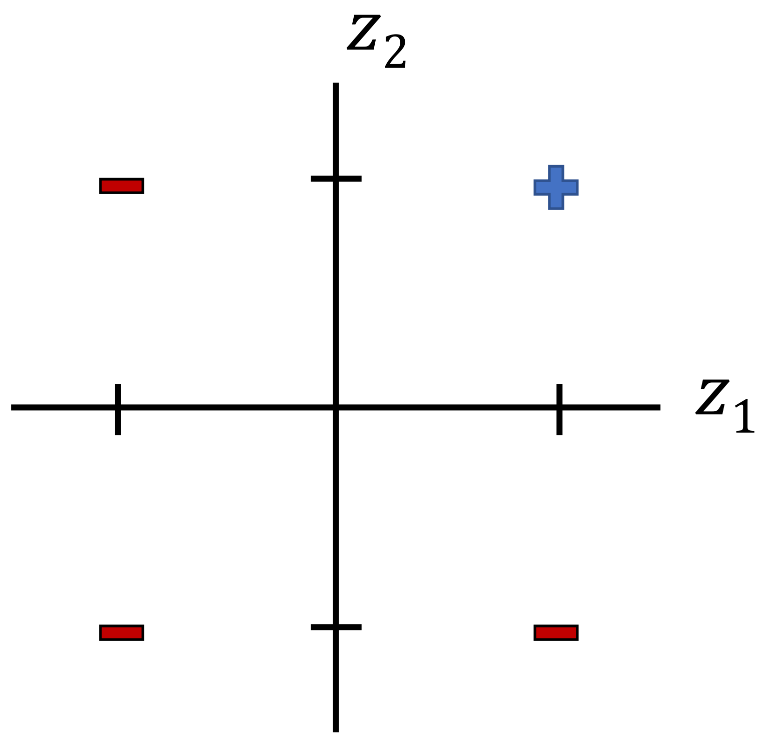
$$sign(\mathbf{x}) = \begin{cases} 1, & if\ x \geq 0 \\ -1, & if\ x < 0 \end{cases}$$

A. $\mathbf{w}_C = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, b_C = 0$

B. $\mathbf{w}_C = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, b_C = 1$

C. $\mathbf{w}_C = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, b_C = -1$

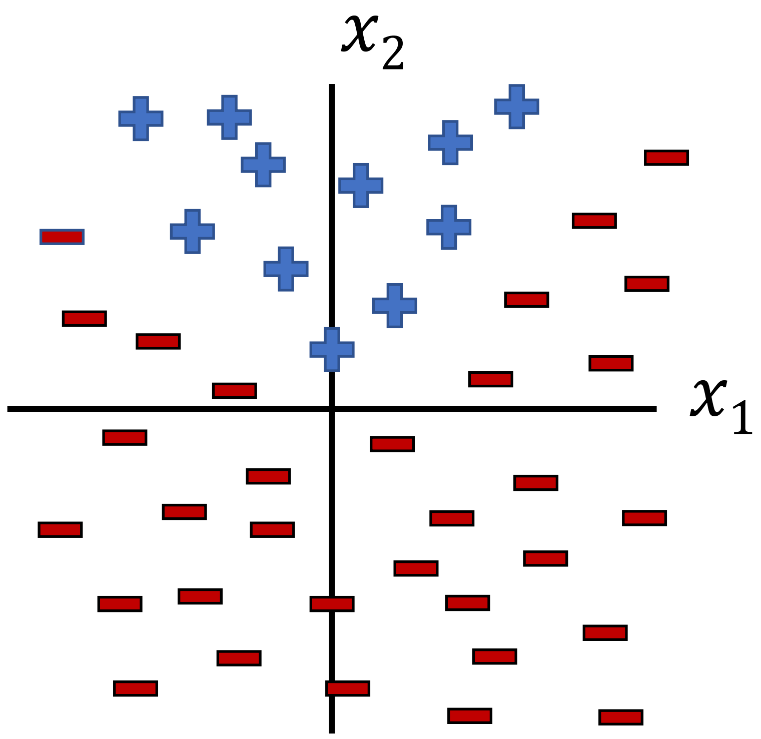
D. None of the above



## Classification Design Challenge

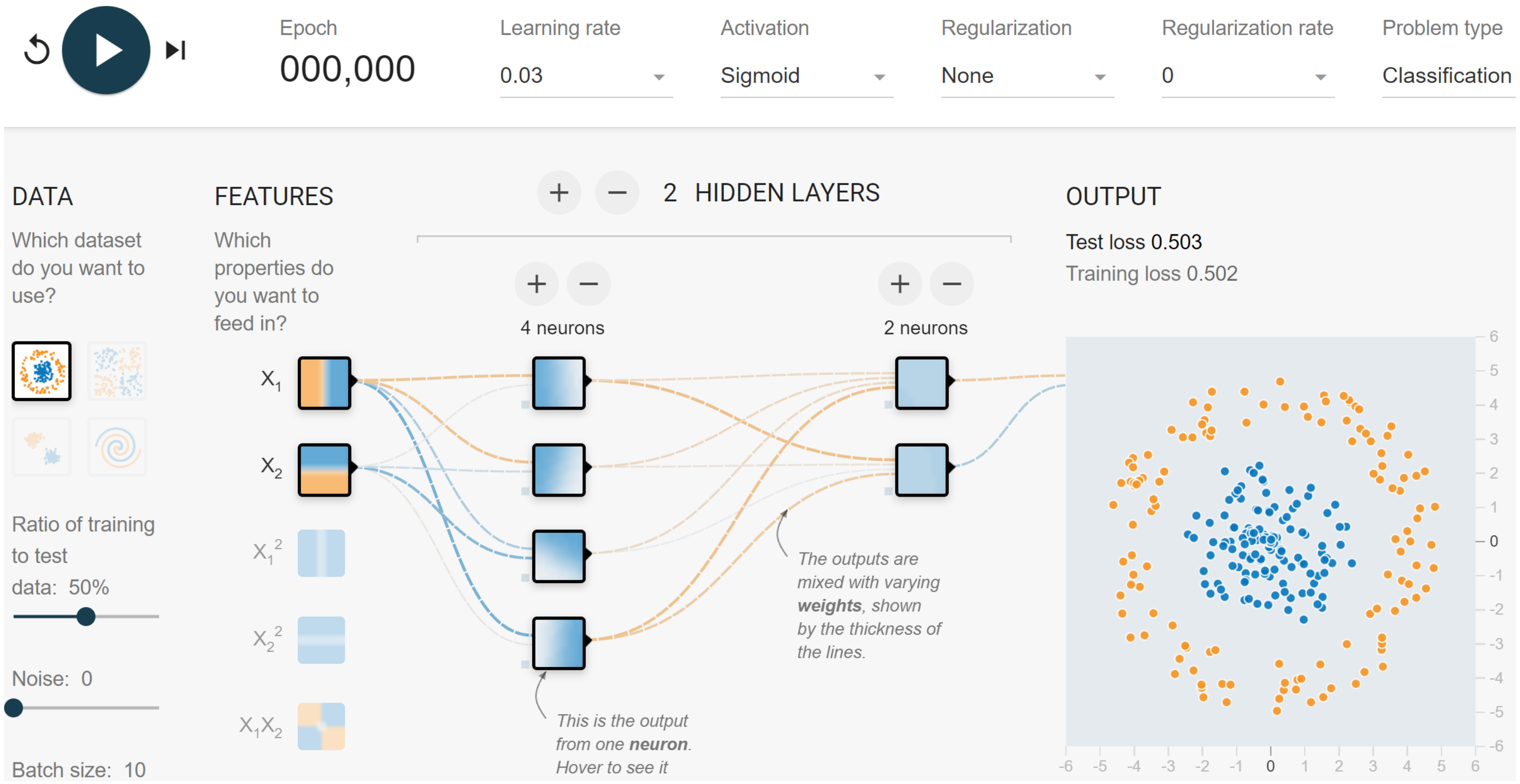
$$h_A(\mathbf{x}) = sign\left(\mathbf{w}_A^\top \mathbf{x} + b_A\right)$$
$$h_B(\mathbf{x}) = sign\left(\mathbf{w}_B^\top \mathbf{x} + b_B\right)$$
$$h_C(\mathbf{z}) = sign\left(\mathbf{w}_C^\top \mathbf{z} + b_C\right)$$

How could you configure three specific perceptrons to classify this data?



# Network to Approximate Binary Classification

https://playground.tensorflow.org/#activation=sigmoid



Epoch 000,000

Learning rate 0.03

Activation Sigmoid

Regularization None

Regularization rate 0

Problem type Classification

DATA

Which dataset do you want to use?

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

FEATURES

Which properties do you want to feed in?

2 HIDDEN LAYERS

4 neurons

2 neurons

OUTPUT

Test loss 0.503

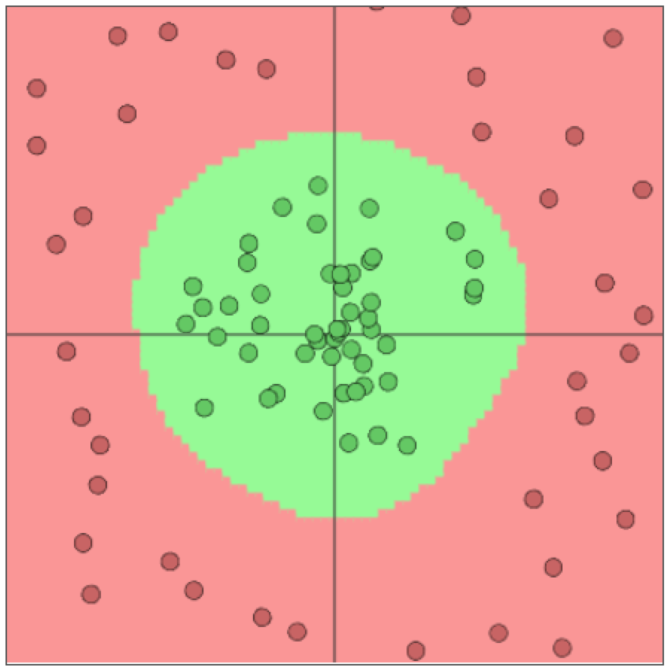
Training loss 0.502

*The outputs are mixed with varying **weights**, shown by the thickness of the lines.*

*This is the output from one **neuron**. Hover to see it*

## Network to Approximate Binary Classification

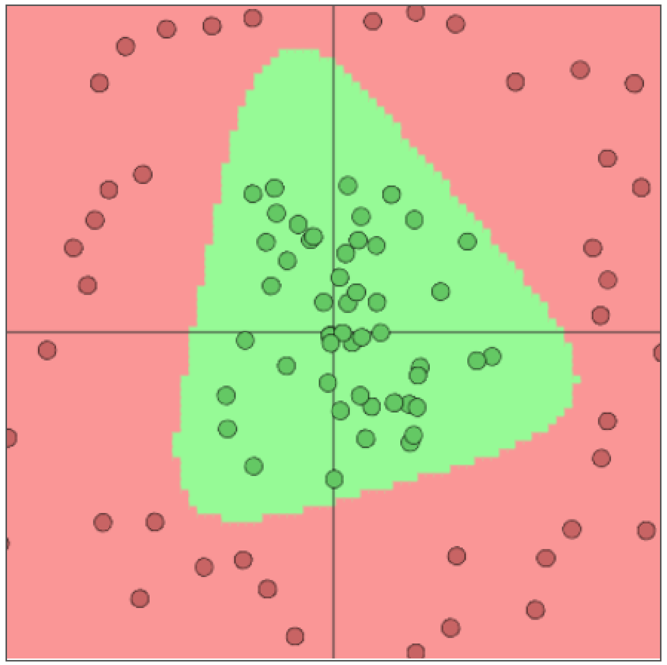
### Approximate arbitrary decision boundary

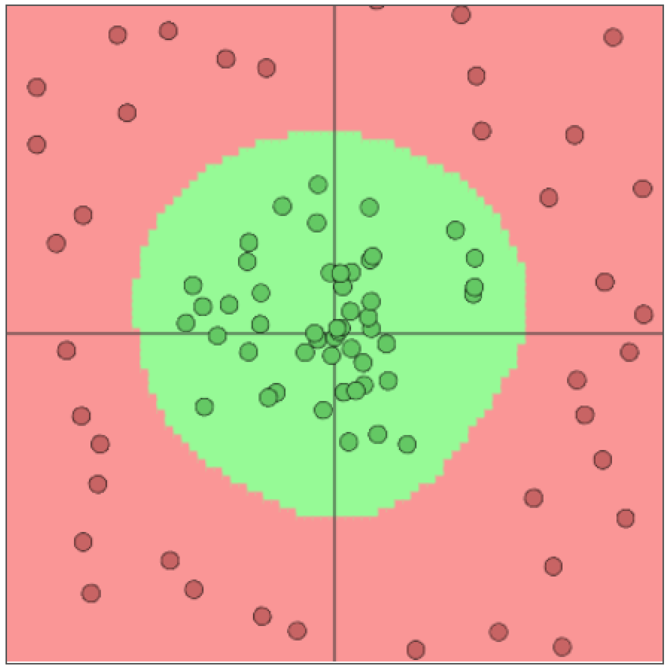


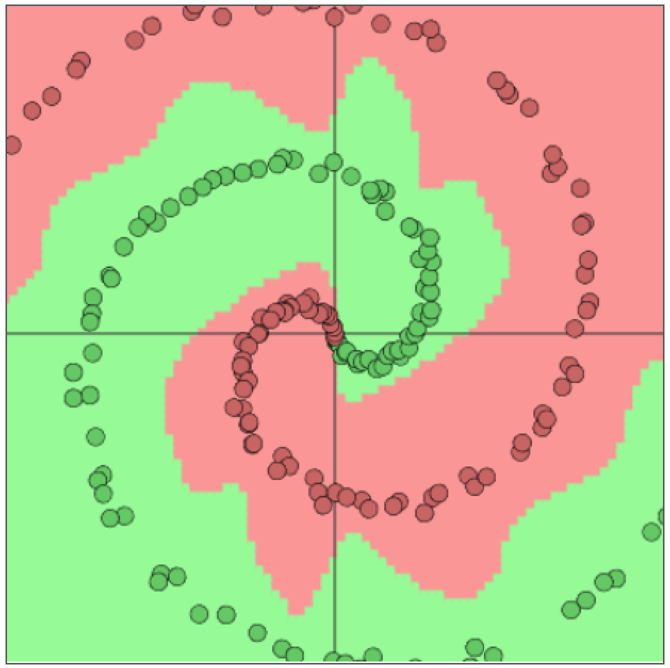
https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

# Network to Approximate Binary Classification

## Approximate arbitrary decision boundary



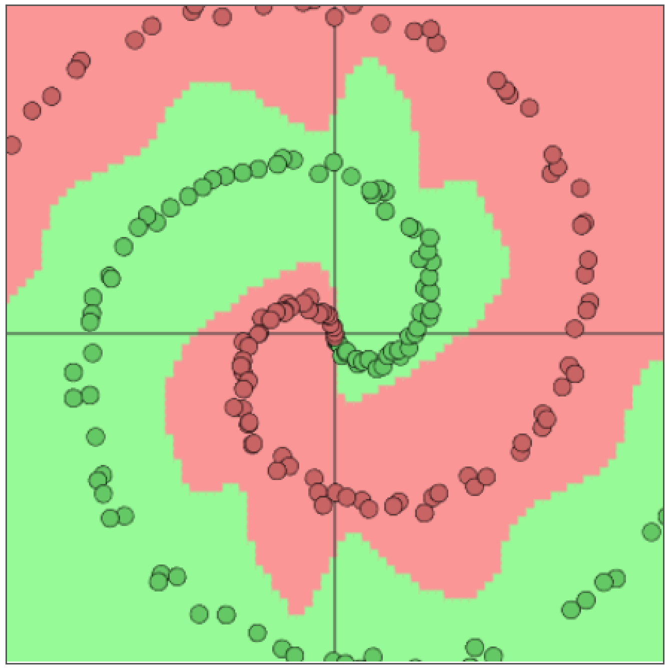




https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

# Network to Approximate Binary Classification

## Approximate arbitrary decision boundary



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

## Neural Network Properties

(Over?) Fitting any function

# Neural Networks Properties

## Universal Approximation Theorem

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

## Reminder

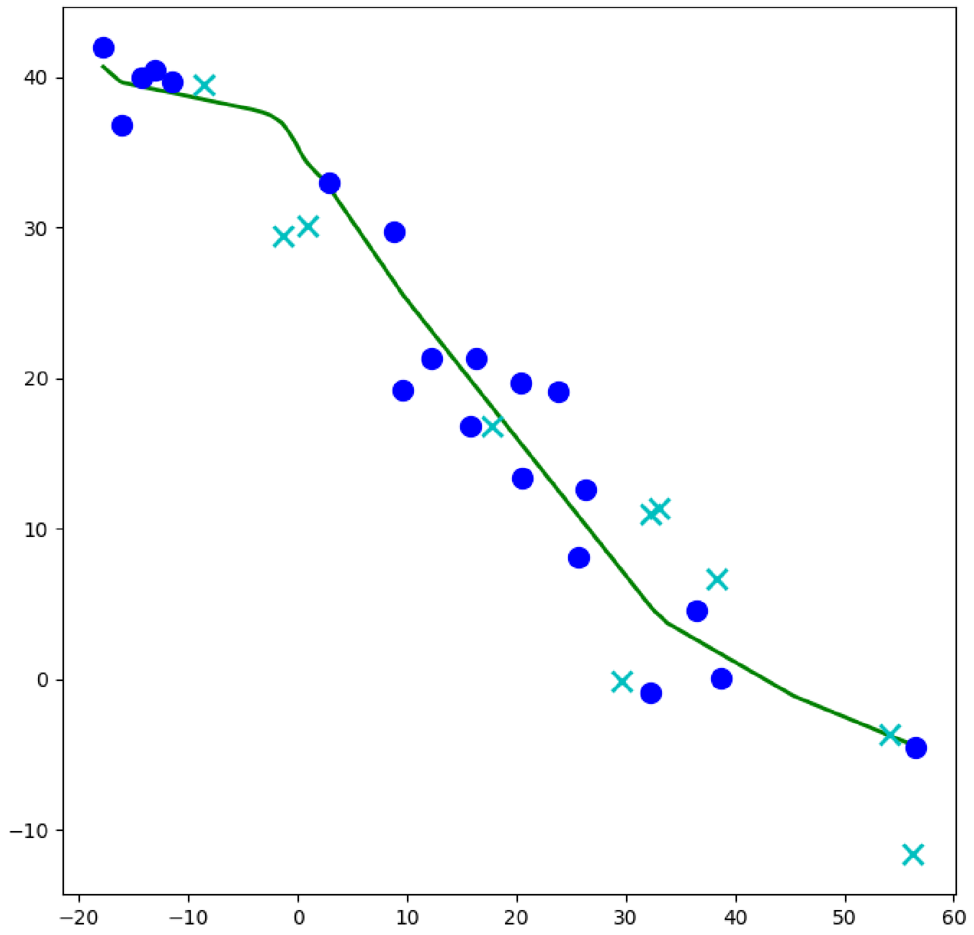
## Just because it can fit any function will it?

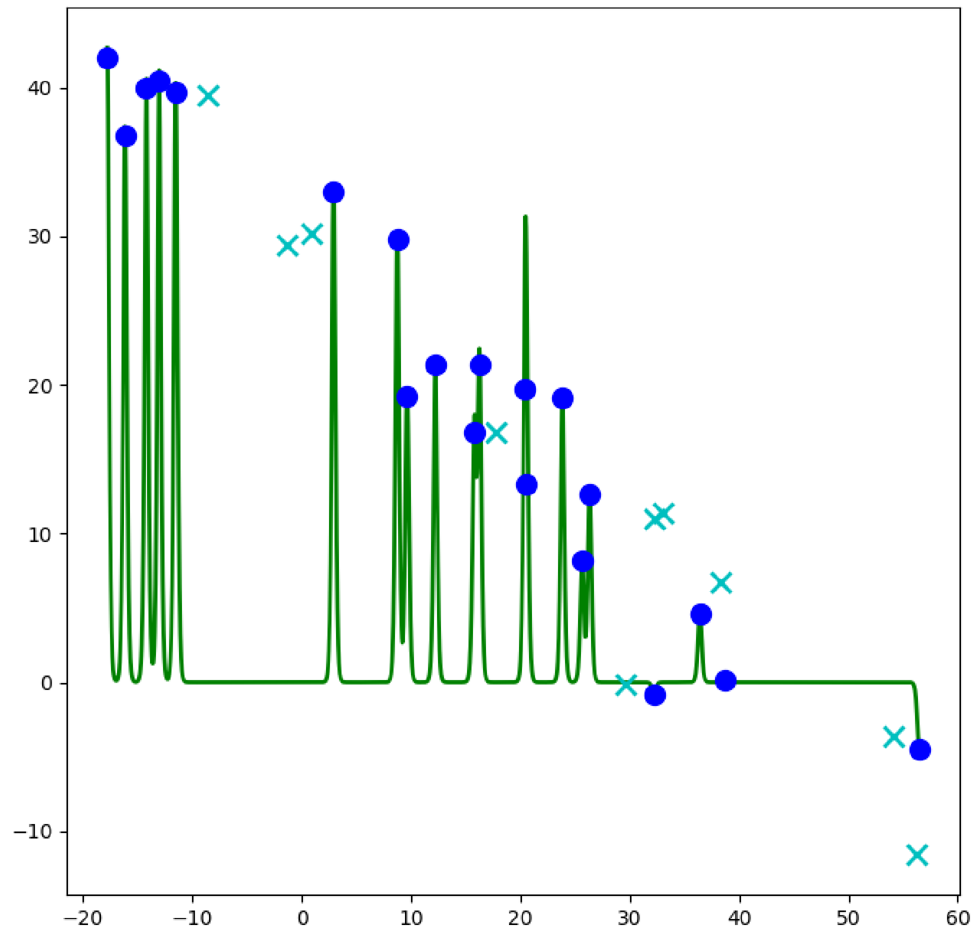
- Objective function is non-convex

  → it will get stuck in local minima
- *Stochastic* gradient decent take pseudorandom steps

  → Helps pop out of local minima
- Is getting stuck at a local minima necessarily a bad thing??

# Network to Approximate a 1-D Function





## Debugging Overfitting and Underfitting

### Underfitting (check this first!)

- Evidence: poor training loss (and poor validation loss)
  - Note: Compare with human performance as a baseline, i.e., make sure the task isn't impossible for a human (with unlimited resources)
- Try: Adding more capacity to network (wider or deeper)
  - But how do we choose a new network structure?

### Overfitting

- Evidence: good training loss, but poor validation loss
- Evidence: really large parameter values
- Try: Regularization (we'll learn about this soon!)
- Try: Adding more data

# Summary: Neural Networks Properties

## Practical considerations

- Large number of neurons
  - Danger for overfitting
- Modeling 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.