Announcements

Assignments

- HW4
  - Due Wed, 10/14, 11:59 pm
- HW5
  - Plan: Out tomorrow
  - Due Mon, 10/26, 11:59 pm

Recitation

- No recitation the next two Fridays ☹
- We'll post a worksheet for neural nets and record a walk-through

Survey

Introduction to Machine Learning

Neural Networks

Instructor: Pat Virtue

Plan

Last Time

- Neural Networks
  - Perceptron
  - Multilayer perceptron

Today

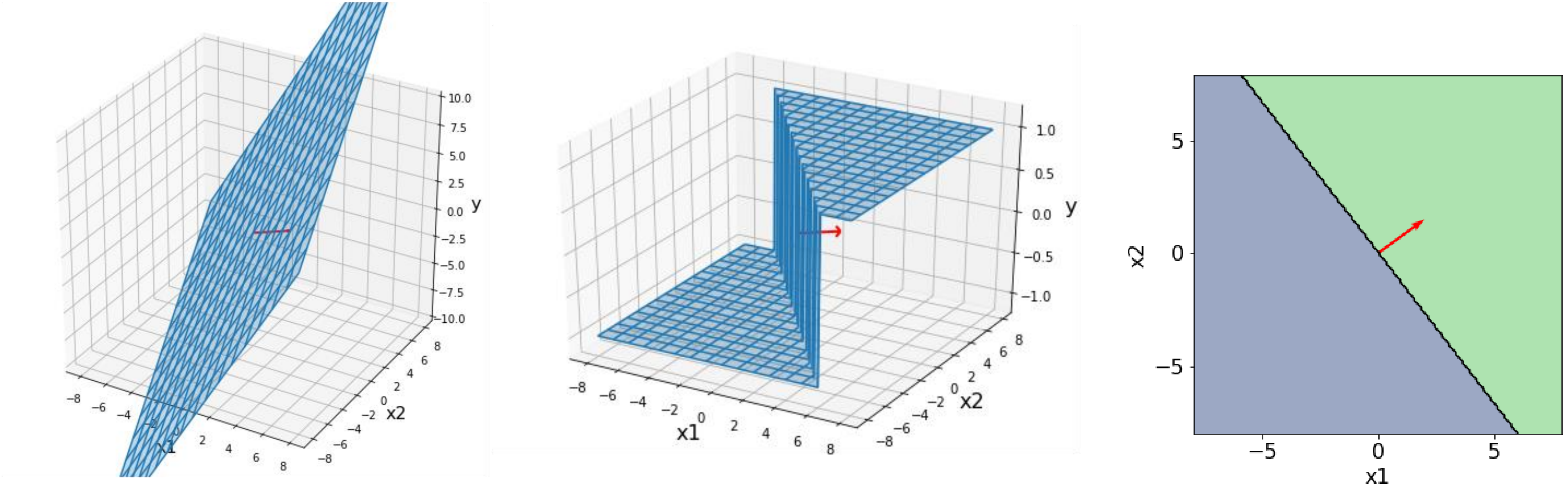
- Neural Networks
  - Building blocks
  - Optimization
    - Composite functions and chain rule
    - Forward-backward passes
    - Matrix calculus

Perceptron

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

Classification: Hard threshold on linear model

$$h(\boldsymbol{x}) = sign(\boldsymbol{w}^T \boldsymbol{x} + b)$$



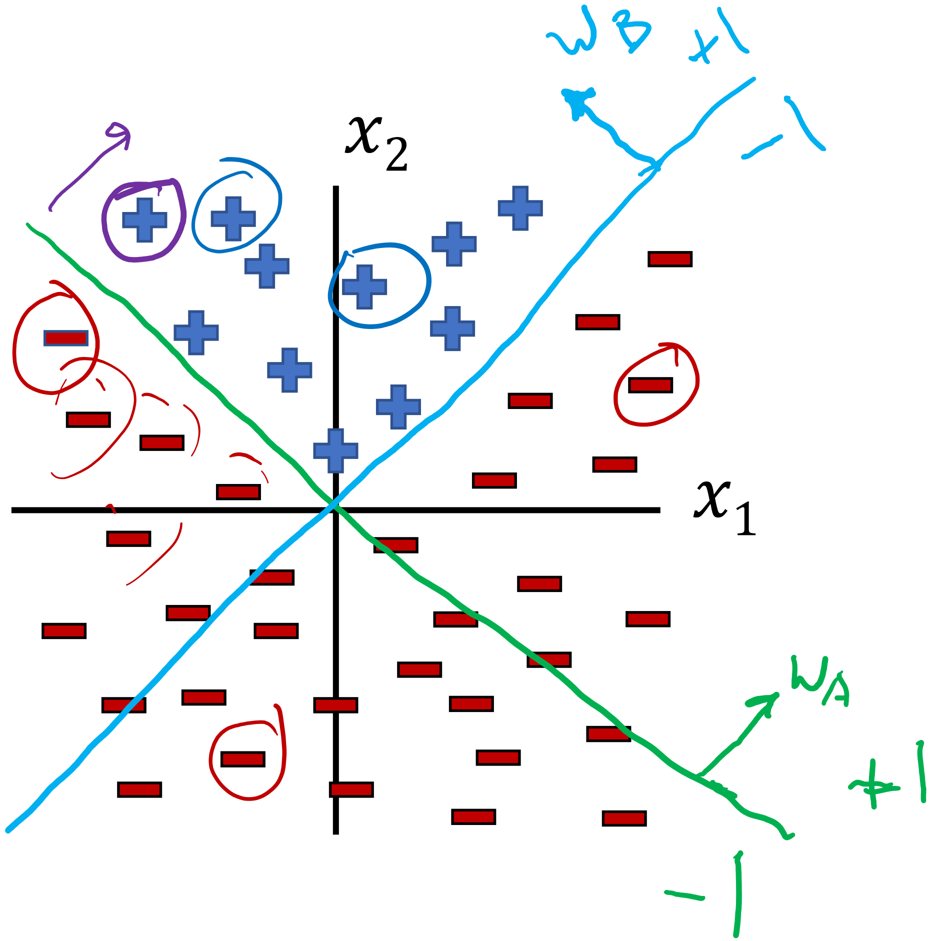
Classification Design Challenge

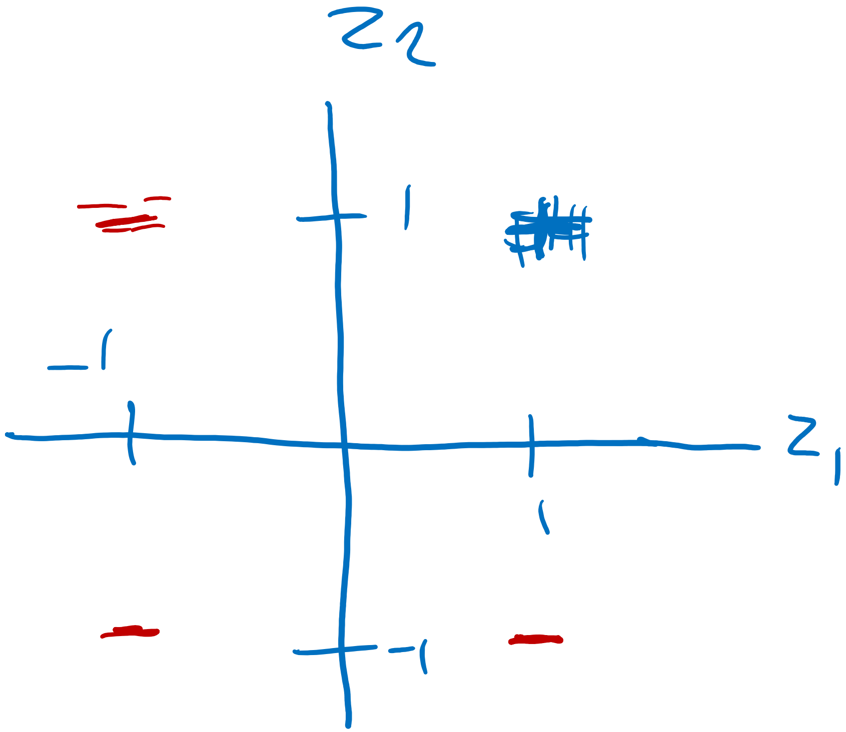
$z_1 = h_A(\mathbf{x}) = sign(\mathbf{w}_A^T \mathbf{x} + b_A)$

$z_2 = h_B(\mathbf{x}) = sign(\mathbf{w}_B^T \mathbf{x} + b_B)$

$h_C(\mathbf{x}) = sign(\mathbf{w}_C^T \mathbf{x} + b_C)$

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





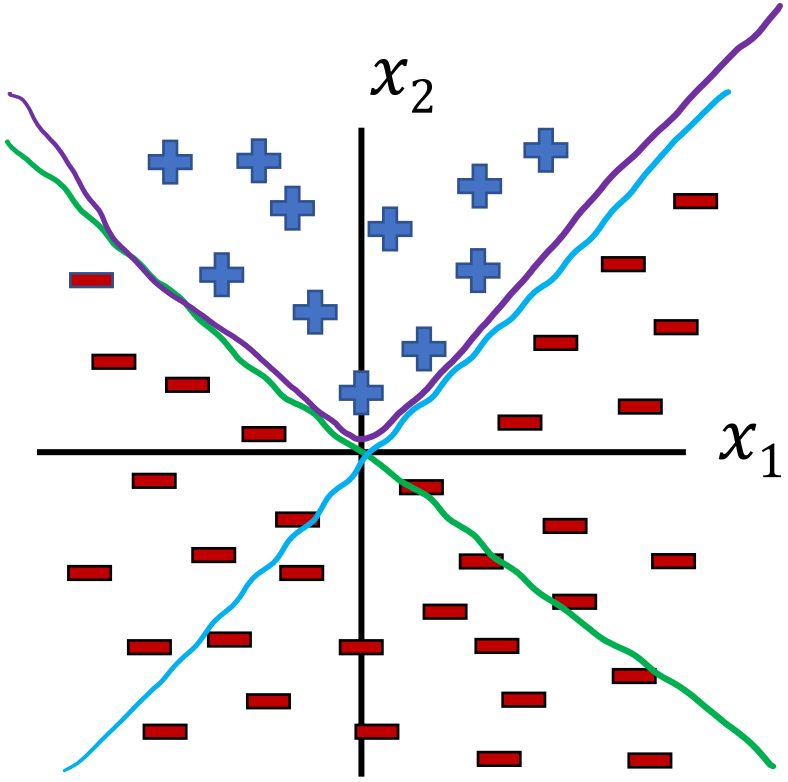
Classification Design Challenge

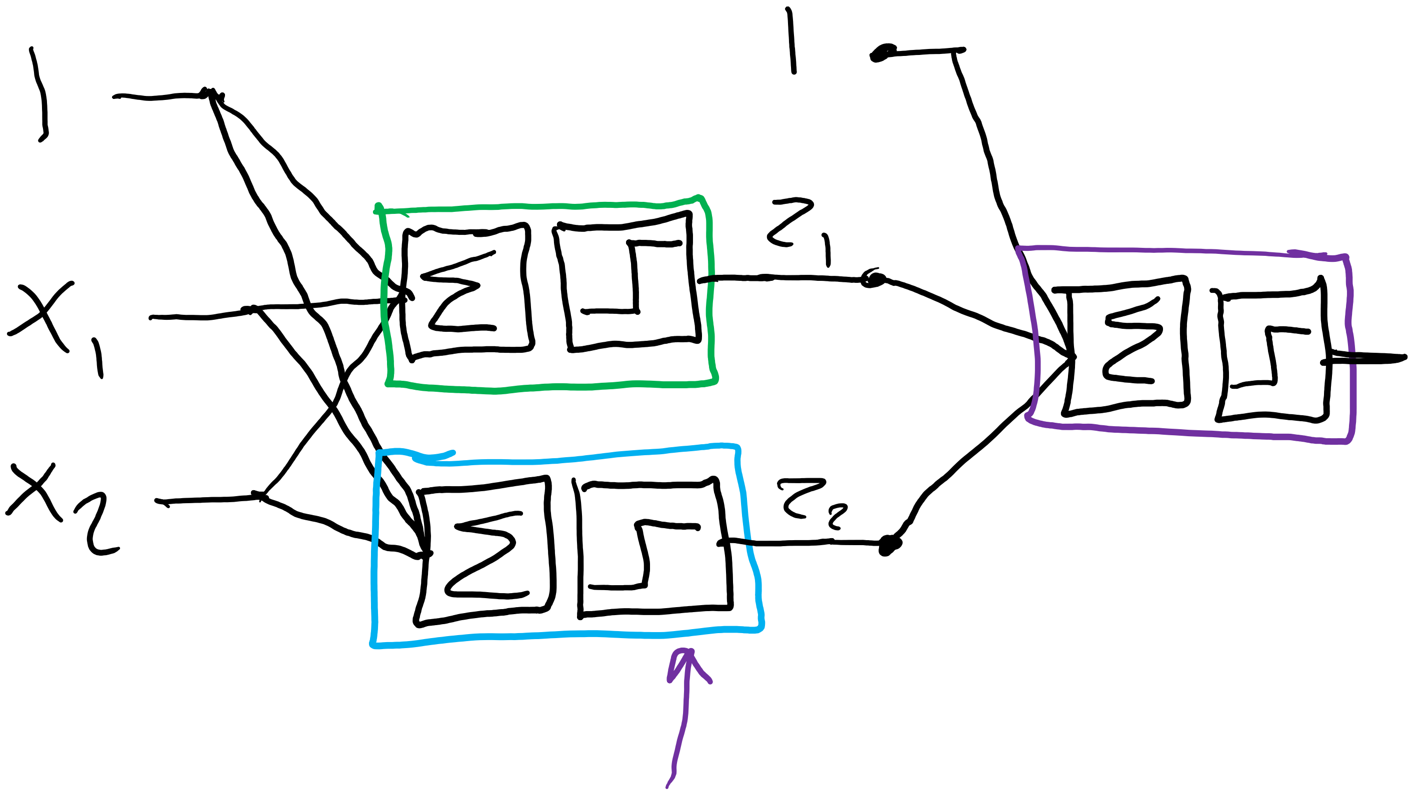
$$h_A(\boldsymbol{x}) = sign(\boldsymbol{w}_A^T\boldsymbol{x} + b_A)$$

$$h_B(\boldsymbol{x}) = sign(\boldsymbol{w}_B^T\boldsymbol{x} + b_B)$$

$$h_C(\boldsymbol{x}) = sign(\boldsymbol{w}_C^T\boldsymbol{x} + b_C)$$

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

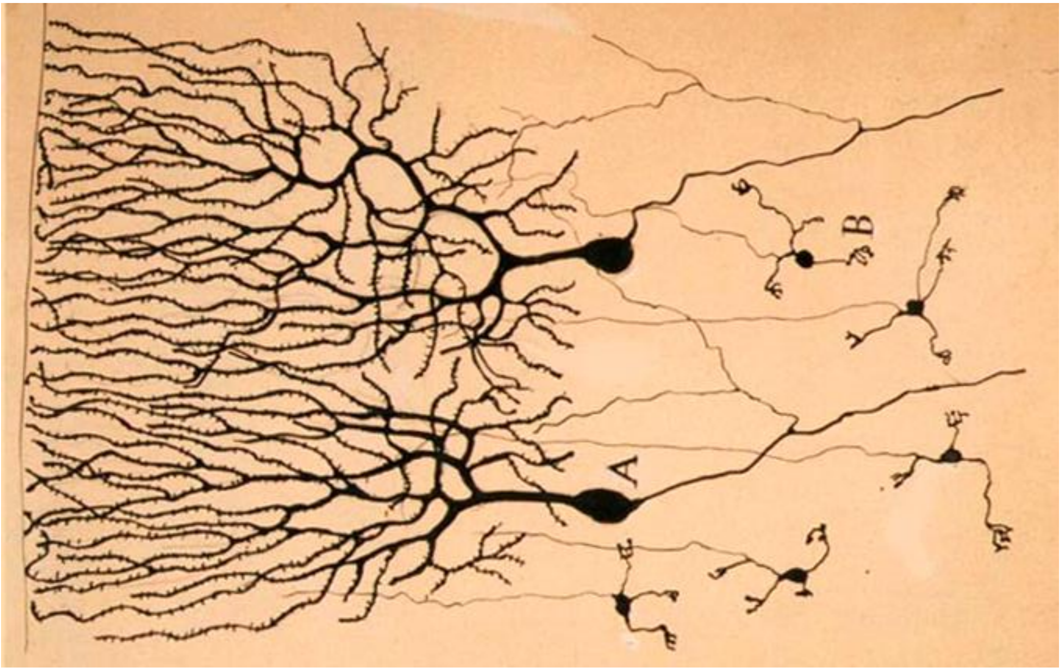




Neural Networks

Inspired by actual human brain

Input Signal



Output Signal

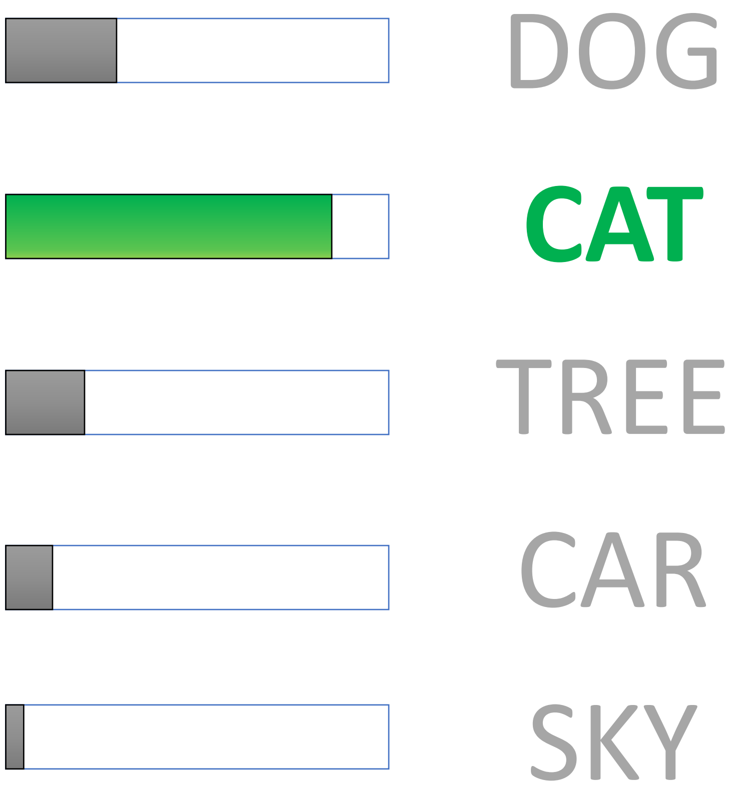
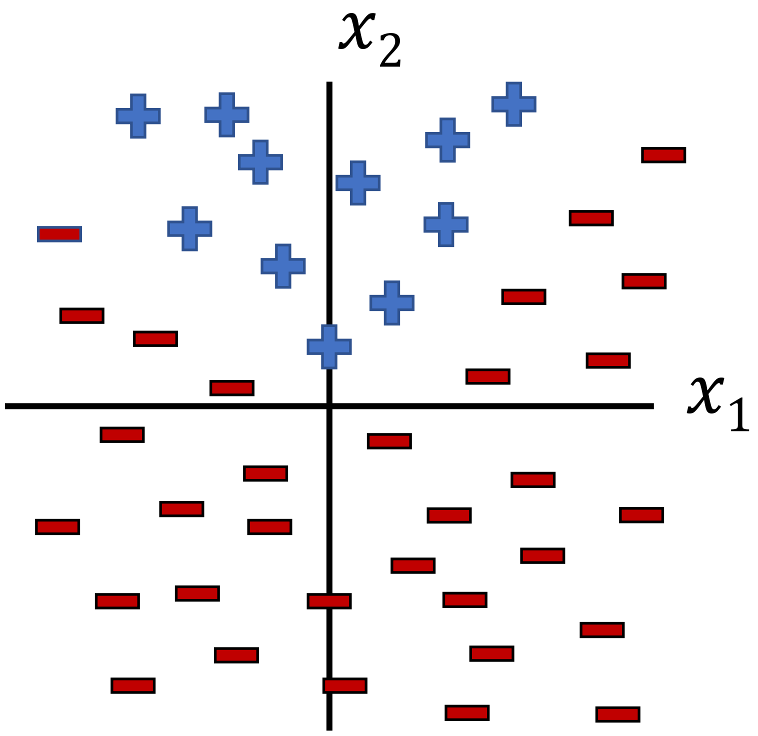


Image: https://en.wikipedia.org/wiki/Neuron

Classification Design Challenge

$$h_A(\boldsymbol{x}) = sign(\boldsymbol{w}_A^T\boldsymbol{x} + b_A)$$
$$h_B(\boldsymbol{x}) = sign(\boldsymbol{w}_B^T\boldsymbol{x} + b_B)$$
$$h_C(\boldsymbol{x}) = sign(\boldsymbol{w}_C^T\boldsymbol{x} + b_C)$$

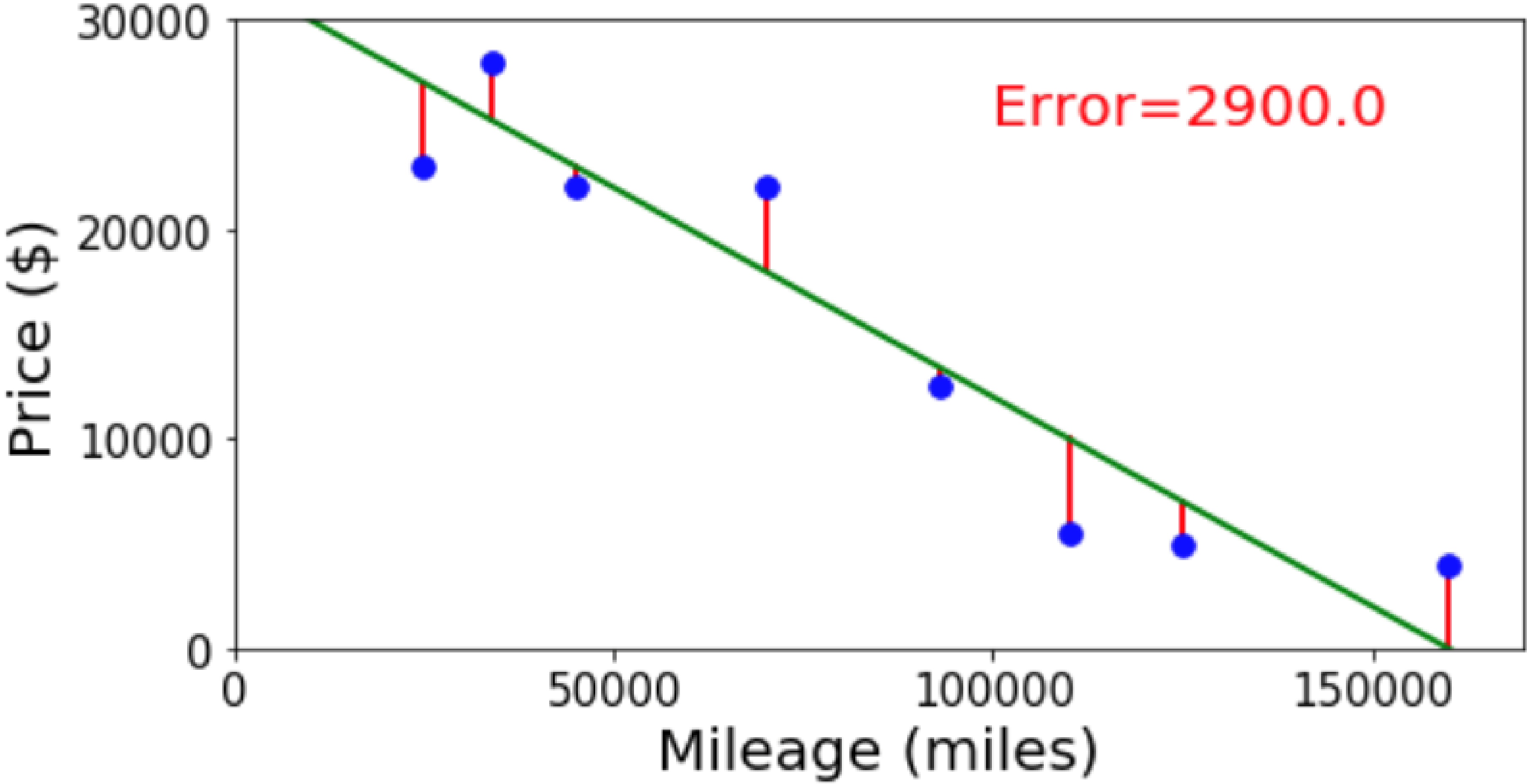
How could you configure three specific perceptrons to classify this data?



Neural Networks

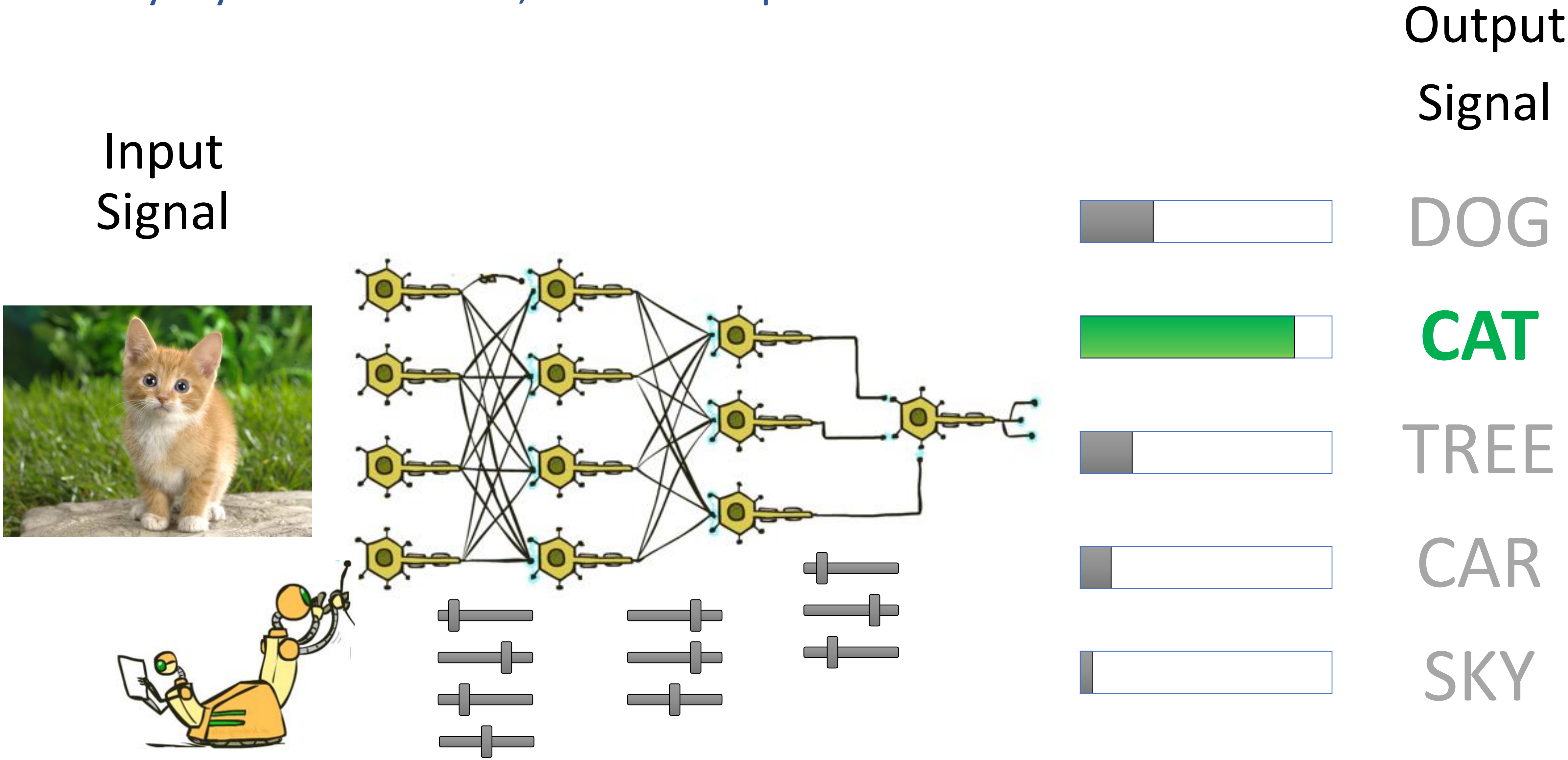
Simple single neuron example:

- Selling my car



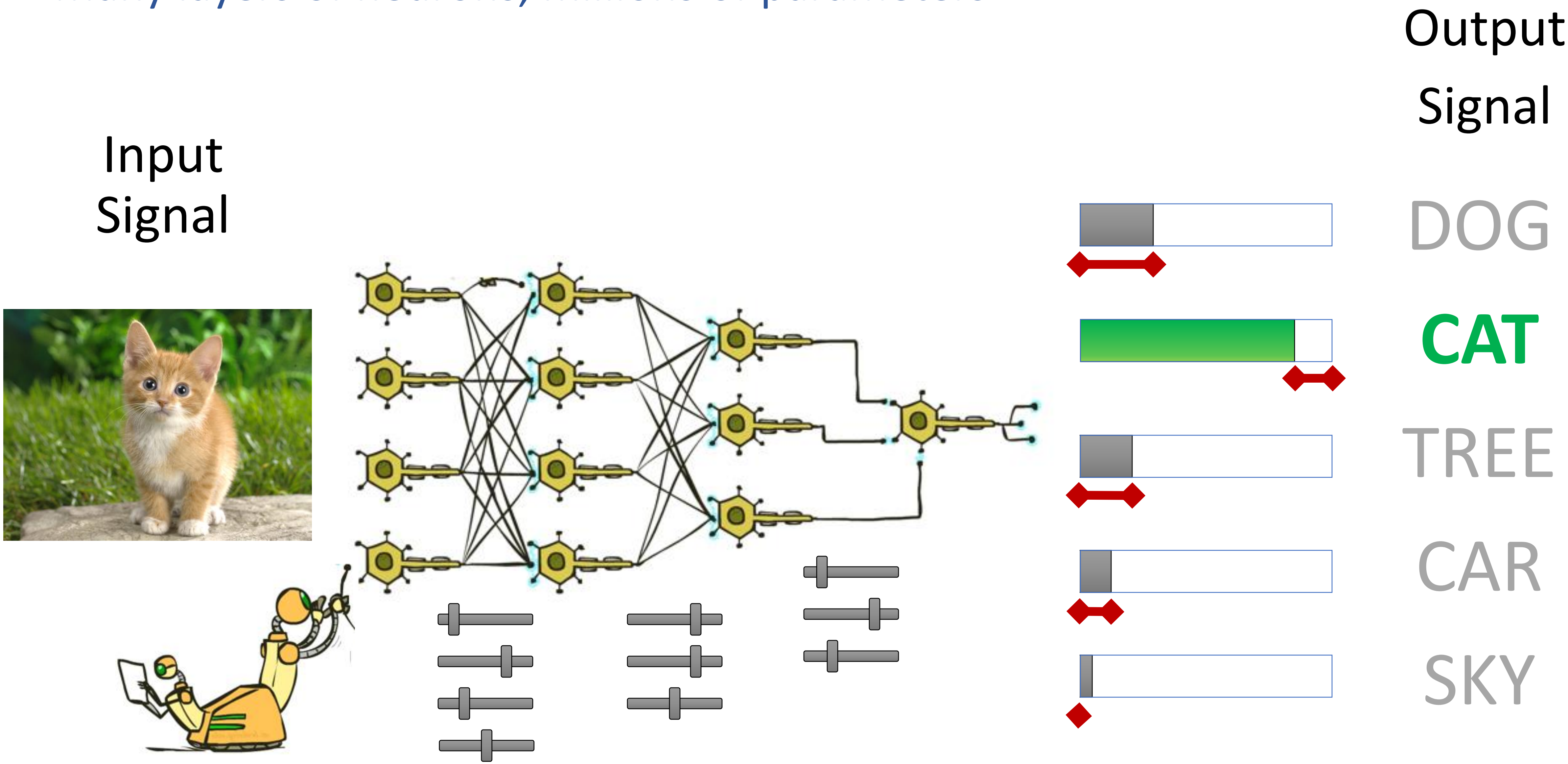
Neural Networks

Many layers of neurons, millions of parameters



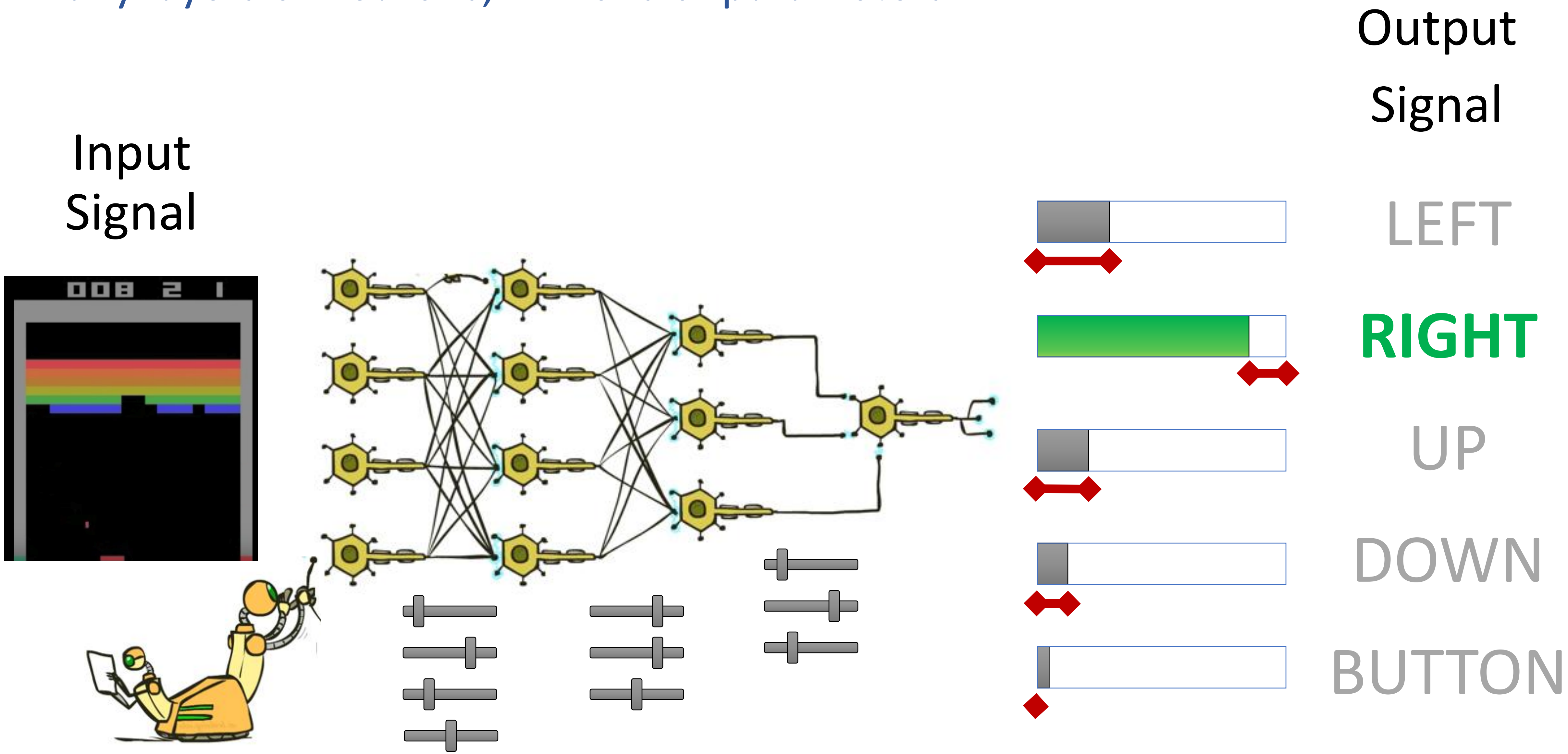
Neural Networks

Many layers of neurons, millions of parameters



Neural Networks

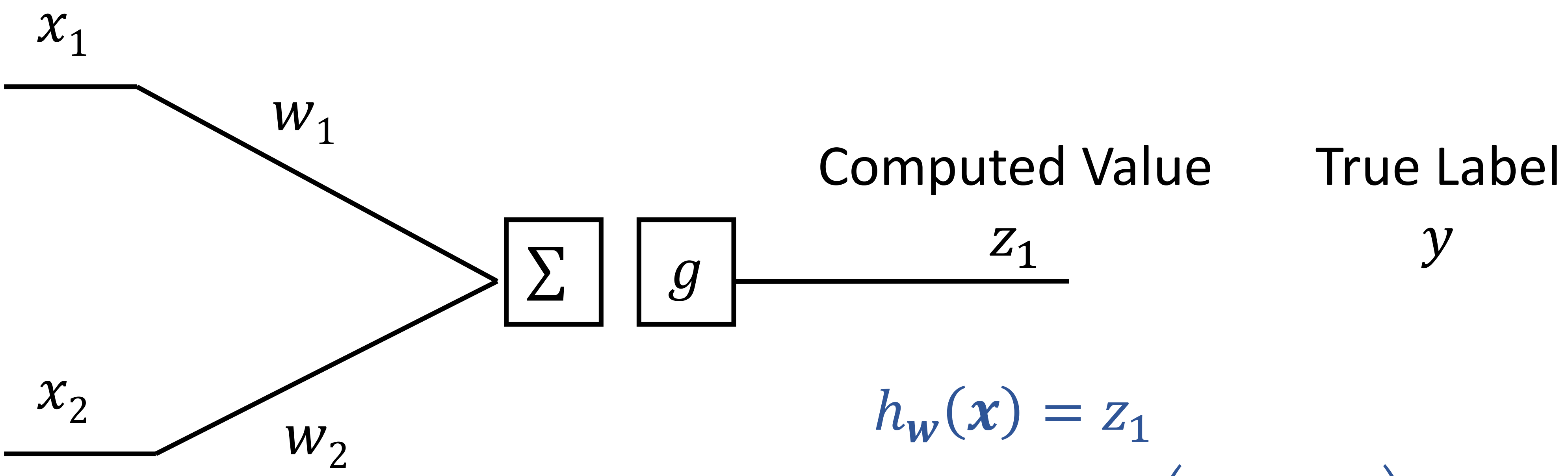
Many layers of neurons, millions of parameters



Single Neuron

Single neuron system

- Perceptron (if g is step function)
- Logistic regression (if g is sigmoid)
- Linear regression (if g is nothing)



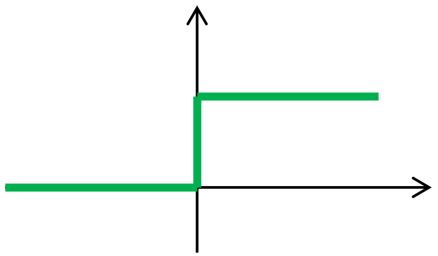
$$h_{\boldsymbol{w}}(\boldsymbol{x}) = z_1$$

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

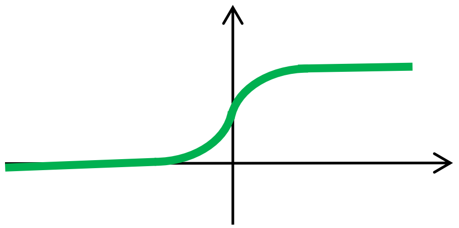
Activation Functions

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

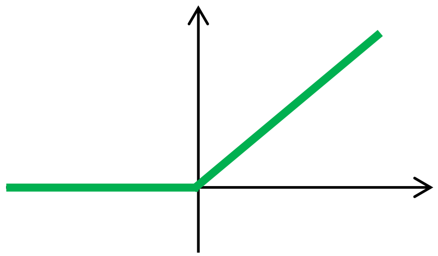
- Hard threshold: $g(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$ $\quad \frac{dg}{dz} = \begin{cases} 0 & z \geq 0 \\ 0 & z < 0 \end{cases}$



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



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



Optimizing

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

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = g\left(\sum_j w_j x_j\right)$$

Loss Functions

Regression

- Squared error: $\ell(y, \hat{y}) = (y - \hat{y})^2$

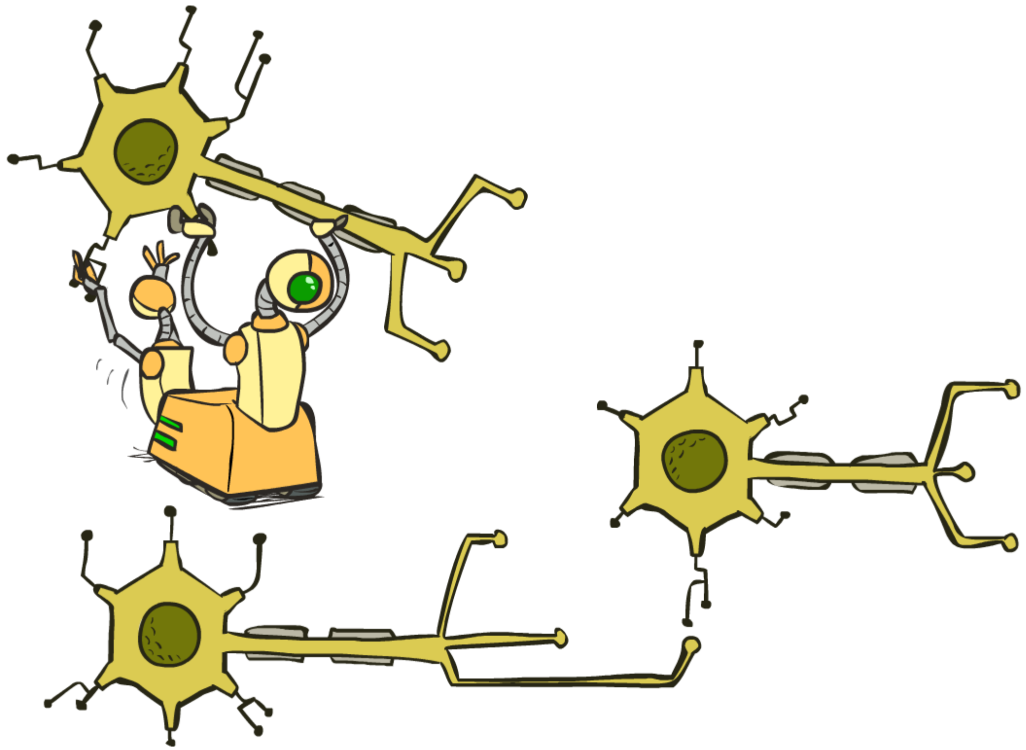
Classification

- Cross entropy: $\ell(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_k \, y_k \log \hat{y}_k$

Multilayer Perceptrons

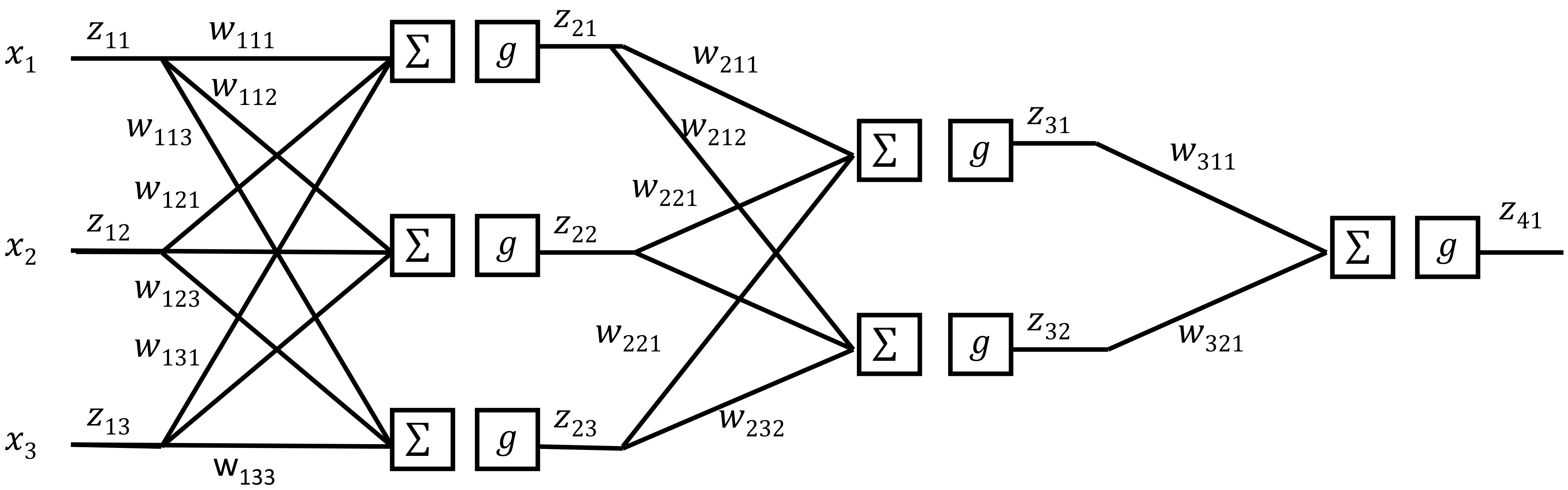
A ***multilayer perceptron*** is a feedforward neural network with at least one ***hidden layer*** (nodes that are neither inputs nor outputs)

MLPs with enough hidden nodes can represent any function



Slide from UC Berkeley AI

Neural Network Equations



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

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

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

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

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

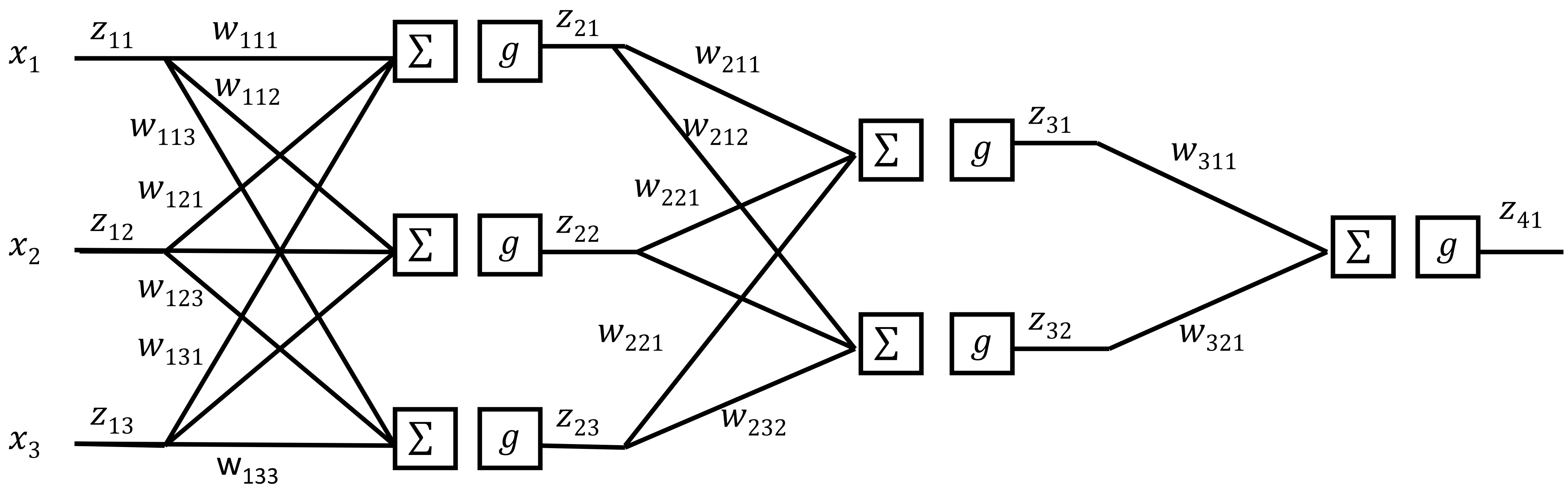
$$h_w(x) = g\left(\sum_k w_{3,k,1}\ g\left(\sum_j w_{2,j,k}\ g\left(\sum_i w_{1,i,j}\ x_i\right)\right)\right)$$

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 do we describe this network?

Network Optimization Details

Reminder: Calculus Chain Rule (scalar version)

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

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

Network Optimization

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

$$z_3 = f_3(w_3, z_2)$$

$$z_2 = f_2(w_2, z_1)$$

$$z_1 = f_1(w_1, x)$$

Network Optimization: Forward then Backwards

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

$z_3 = f_3(w_3, z_2)$

$z_2 = f_2(w_2, z_1)$

$z_1 = f_1(w_1, x)$

$$\frac{\partial J}{\partial w_3} = \frac{\partial J}{\partial z_3}\frac{\partial z_3}{\partial w_3}$$

$$\frac{\partial J}{\partial w_2} = \frac{\partial J}{\partial z_3}\frac{\partial z_3}{\partial z_2}\frac{\partial z_2}{\partial w_2}$$

$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial z_3}\frac{\partial z_3}{\partial z_2}\frac{\partial z_2}{\partial z_1}\frac{\partial z_1}{\partial w_1}$$

Lots of repeated calculations

Network Optimization: Layer Implementation

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

$$z_3 = f_3(w_3, z_2)$$

$$z_2 = f_2(w_2, z_1)$$

$$z_1 = f_1(w_1, x)$$

$$\frac{\partial J}{\partial w_3} = \frac{\partial J}{\partial z_3}\frac{\partial z_3}{\partial w_3}$$

$$\frac{\partial J}{\partial w_2} = \frac{\partial J}{\partial z_3}\frac{\partial z_3}{\partial z_2}\frac{\partial z_2}{\partial w_2}$$

$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial z_3}\frac{\partial z_3}{\partial z_2}\frac{\partial z_2}{\partial z_1}\frac{\partial z_1}{\partial w_1}$$

Lots of repeated calculations

Backpropagation (so-far)

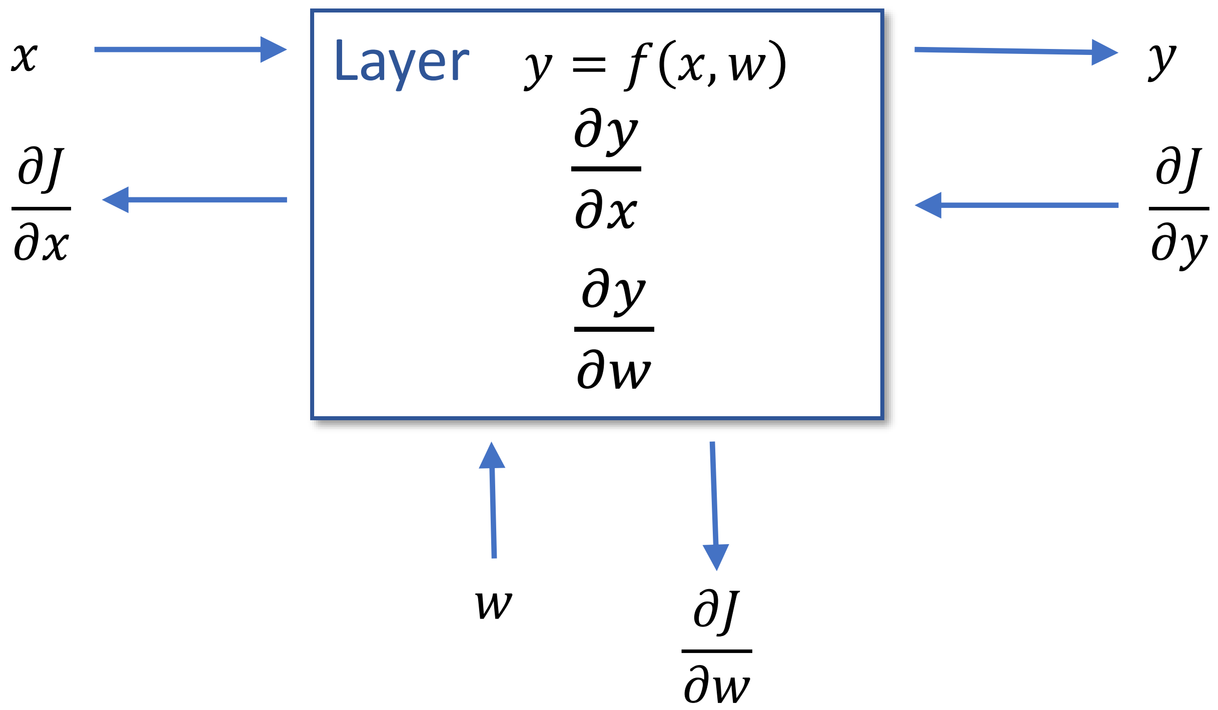
Compute derivatives per layer, utilizing previous derivatives

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

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

Need:

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



Matrix Calculus

One way to think of it: Bag of Derivatives

Matrix Calculus

One way to think of it: Bag of Derivatives

Matrix Calculus

Jacobian: Vector in, vector out

Numerator-layout

$$\boldsymbol{y} = f(\boldsymbol{x}) \qquad \boldsymbol{y} \in \mathbb{R}^N, \quad \boldsymbol{x} \in \mathbb{R}^M, \quad \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \in \mathbb{R}^{N \times M}$$

Matrix Calculus

Vector in, scalar out

Numerator-layout

$$y = f(\boldsymbol{x}) \qquad y \in \mathbb{R}, \quad \boldsymbol{x} \in \mathbb{R}^M, \quad \frac{\partial y}{\partial \boldsymbol{x}} \in \mathbb{R}^{1 \times M}$$

Matrix Calculus

Scalar in, vector out

Numerator-layout

$$\boldsymbol{y} = f(x) \qquad \boldsymbol{y} \in \mathbb{R}^N, \quad x \in \mathbb{R}, \quad \frac{\partial \boldsymbol{y}}{\partial x} \in \mathbb{R}^{N \times 1}$$

Matrix Calculus

Gradient: Vector in, scalar out

Transpose of numerator-layout

$$y = f(\boldsymbol{x}) \qquad y \in \mathbb{R}, \quad \boldsymbol{x} \in \mathbb{R}^M, \quad \frac{\partial y}{\partial \boldsymbol{x}} \in \mathbb{R}^{1\times M}, \quad \nabla_{\boldsymbol{x}} f \in \mathbb{R}^{M\times 1}$$

Matrix Calculus

Matrix in, scalar out

Keep same dimensions as matrix

$$y = f(\boldsymbol{X}) \quad y \in \mathbb{R}, \;\; \boldsymbol{X} \in \mathbb{R}^{N \times M}, \;\; \frac{\partial y}{\partial \boldsymbol{X}} \in \mathbb{R}^{N \times M}$$

Multivariate Chain Rule

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

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

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

Multivariate Chain Rule

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

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

$$y = f(z_1, z_2) = z_1 z_2$$

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:

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

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

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

Network Optimization

$J(\boldsymbol{w}) = z_4$

$z_4 = f_4(w_D, w_E, z_2, z_3)$

$z_3 = f_3(w_C, z_1)$

$z_2 = f_2(w_B, z_1)$

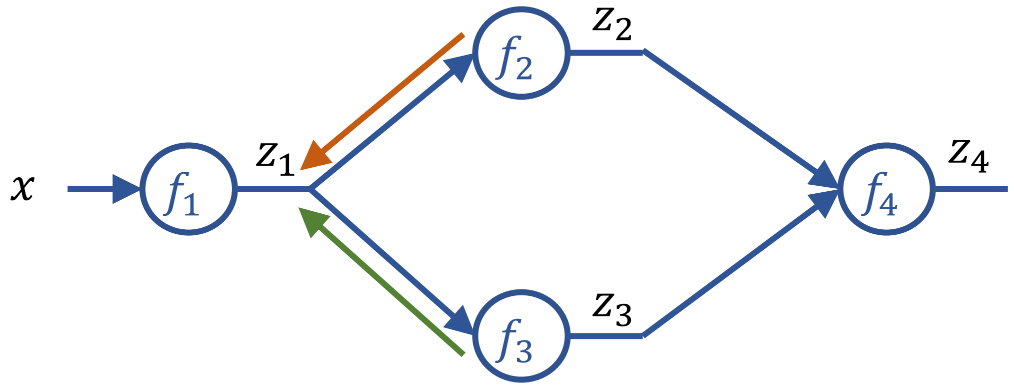
$z_1 = f_1(w_A, x)$

Need multivariate chain rule!

Network Optimization

$$J(\boldsymbol{w}) = z_4$$
$$z_4 = f_4(w_D, w_E, z_2, z_3)$$
$$z_3 = f_3(w_C, z_1)$$
$$z_2 = f_2(w_B, z_1)$$
$$z_1 = f_1(w_A, x)$$

Need multivariate chain rule!



$$\frac{\partial J}{\partial w_E} = \frac{\partial J}{\partial z_4}\frac{\partial z_4}{\partial w_E}$$

$$\frac{\partial J}{\partial w_D} = \frac{\partial J}{\partial z_4}\frac{\partial z_4}{\partial w_D}$$

$$\frac{\partial J}{\partial z_3} = \frac{\partial J}{\partial z_4}\frac{\partial z_4}{\partial z_3}$$
$$\frac{\partial J}{\partial z_2} = \frac{\partial J}{\partial z_4}\frac{\partial z_4}{\partial z_2}$$

$$\frac{\partial J}{\partial w_C} = \frac{\partial J}{\partial z_3}\frac{\partial z_3}{\partial w_C}$$
$$\frac{\partial J}{\partial w_B} = \frac{\partial J}{\partial z_2}\frac{\partial z_2}{\partial w_B}$$

$$\frac{\partial J}{\partial z_1} = \frac{\partial J}{\partial z_2}\frac{\partial z_2}{\partial z_1} + \frac{\partial J}{\partial z_3}\frac{\partial z_3}{\partial z_1}$$

$$\frac{\partial J}{\partial w_A} = \frac{\partial J}{\partial z_1}\frac{\partial z_1}{\partial w_A}$$

Backpropagation (updated)

Compute derivatives per layer, utilizing previous derivatives

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

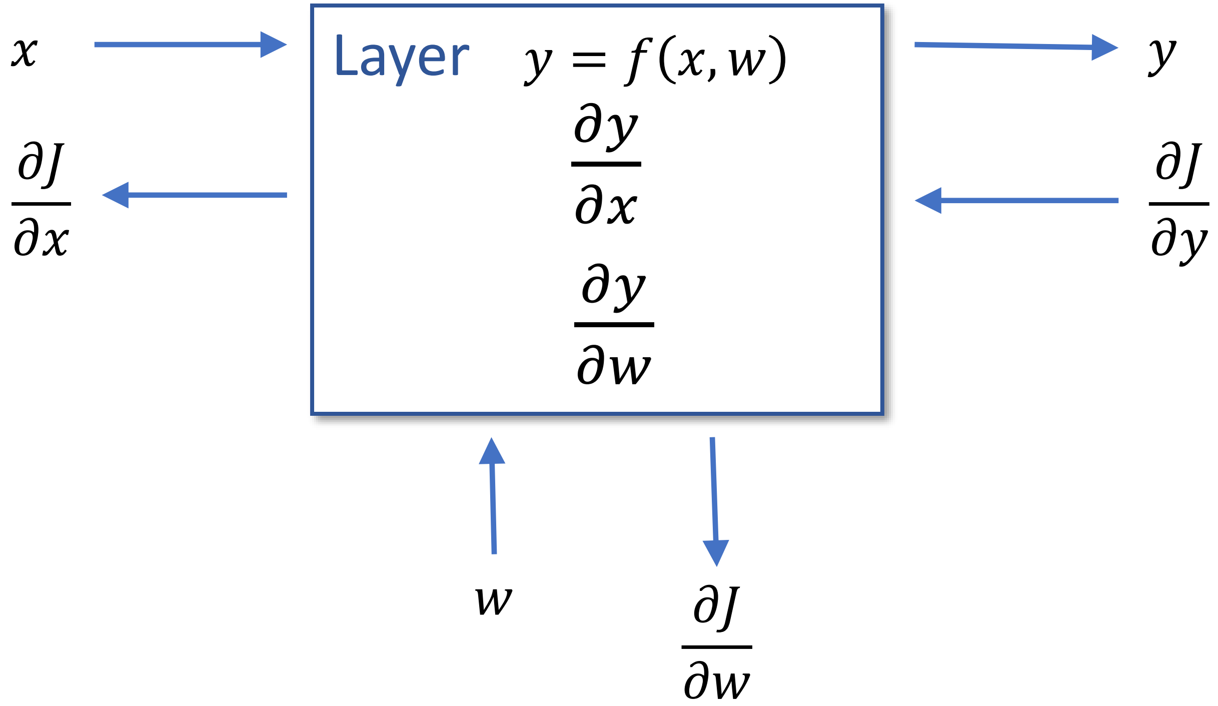
Arbitrary layer: $y = f(x, w)$

Init:

- $\frac{\partial J}{\partial x} = 0$
- $\frac{\partial J}{\partial w} = 0$

Compute:

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



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.