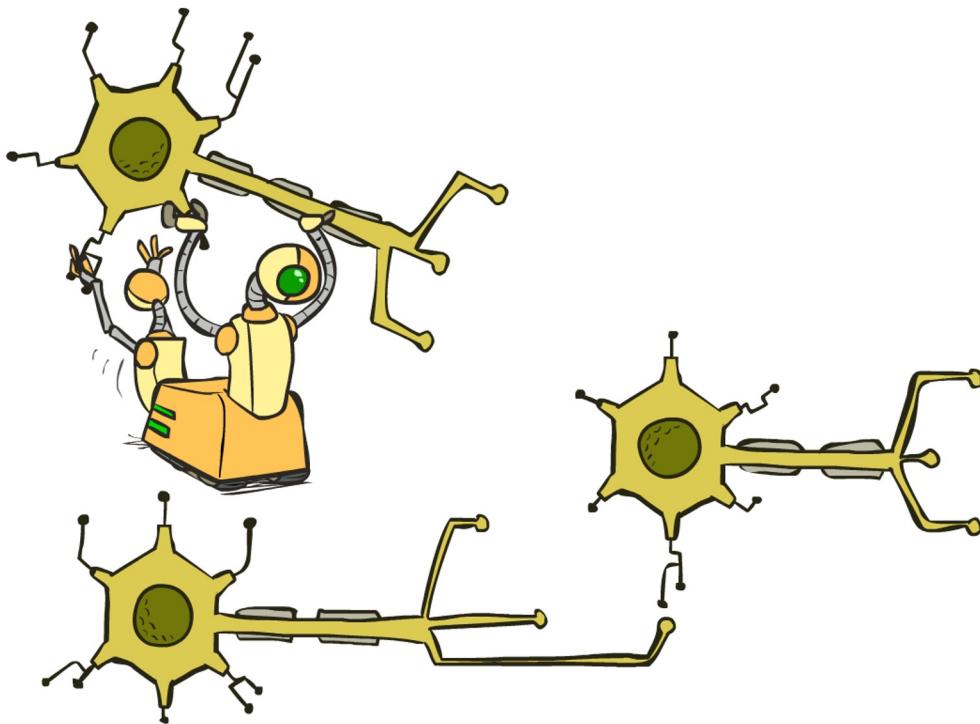


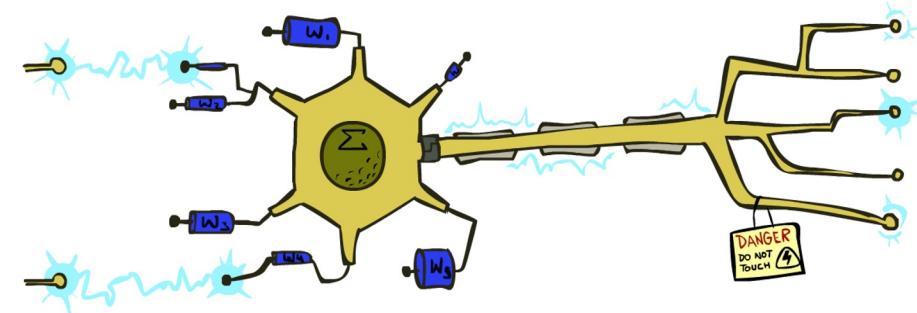
# CS 188: Artificial Intelligence

## Optimization and Neural Networks



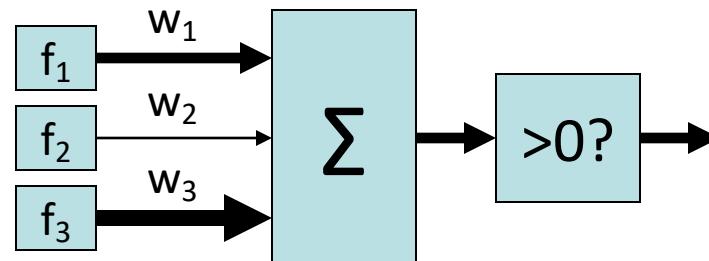
# Reminder: Linear Classifiers

- Inputs are **feature values**
- Each feature has a **weight**
- Sum is the **activation**



$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

- If the activation is:
  - Positive, output +1
  - Negative, output -1



# How to get probabilistic decisions?

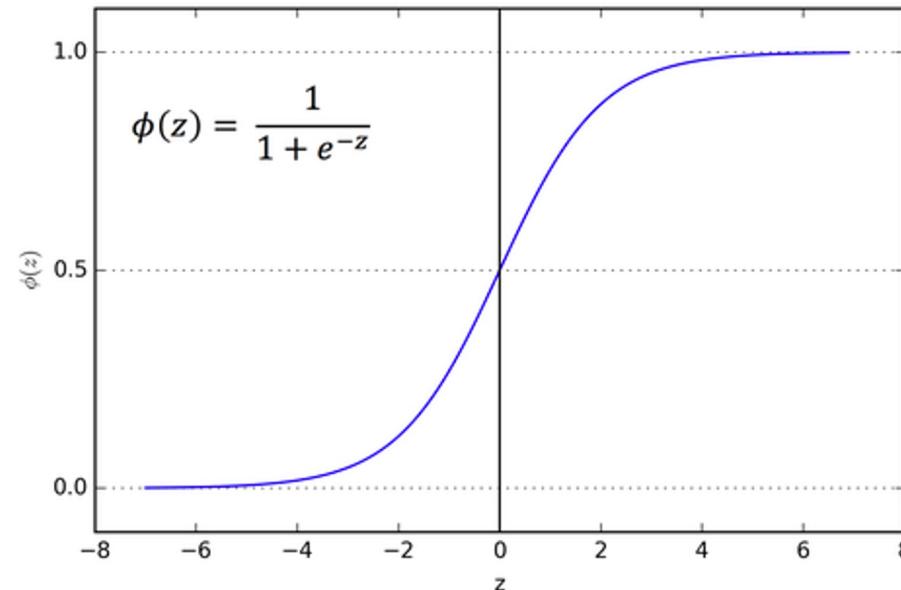
Activation:  $z = w \cdot f(x)$

If  $z = w \cdot f(x)$  very positive  $\rightarrow$  want probability going to 1

If  $z = w \cdot f(x)$  very negative  $\rightarrow$  want probability going to 0

Sigmoid function

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



# Best w?

---

Maximum likelihood estimation:

$$\max_w \text{ll}(w) = \max_w \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

with:

$$P(y^{(i)} = +1 | x^{(i)}; w) = \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

$$P(y^{(i)} = -1 | x^{(i)}; w) = 1 - \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

= Logistic Regression

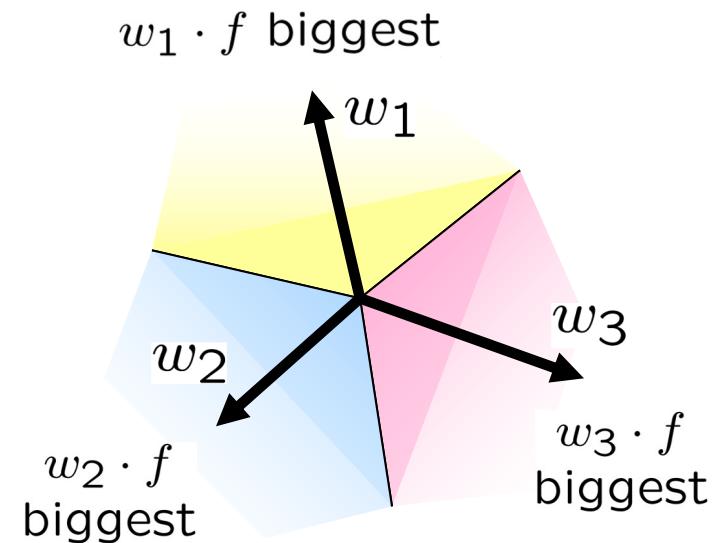
# Multiclass Logistic Regression

# Multi-class linear classification

A weight vector for each class:  $w_y$

Score (activation) of a class  $y$ :  $w_y \cdot f(x)$

Prediction w/highest score wins:  $y = \arg \max_y w_y \cdot f(x)$



# How to make the scores into probabilities?

$$z_1, z_2, z_3 \rightarrow \underbrace{\frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}}_{\begin{array}{l} \text{original activations} \\ \text{softmax activations} \end{array}}$$

# Best w?

---

Maximum likelihood estimation:

$$\max_w \text{ll}(w) = \max_w \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

with:  $P(y^{(i)} | x^{(i)}; w) = \frac{e^{w_{y^{(i)}} \cdot f(x^{(i)})}}{\sum_y e^{w_y \cdot f(x^{(i)})}}$

= Multi-Class Logistic Regression

# This Lecture

---

## Optimization

i.e., how do we solve:

$$\max_w \text{ll}(w) = \max_w \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

# Hill Climbing

- Recall from CSPs lecture: simple, general idea
  - Start wherever
  - Repeat: move to the best neighboring state
  - If no neighbors better than current, quit
- What's particularly tricky when hill-climbing for multiclass logistic regression?
  - Optimization over a continuous space
    - Infinitely many neighbors!
    - How to do this efficiently?



# Review: Derivatives and Gradients

---

- What is the derivative of the function  $g(x) = x^2 + 3$  ?

$$\frac{dg}{dx} = 2x$$

- What is the derivative of  $g(x)$  at  $x=5$ ?

$$\frac{dg}{dx}|_{x=5} = 10$$

# Review: Derivatives and Gradients

- What is the gradient of the function  $g(x, y) = x^2y$  ?
  - Recall: Gradient is a vector of partial derivatives with respect to each variable

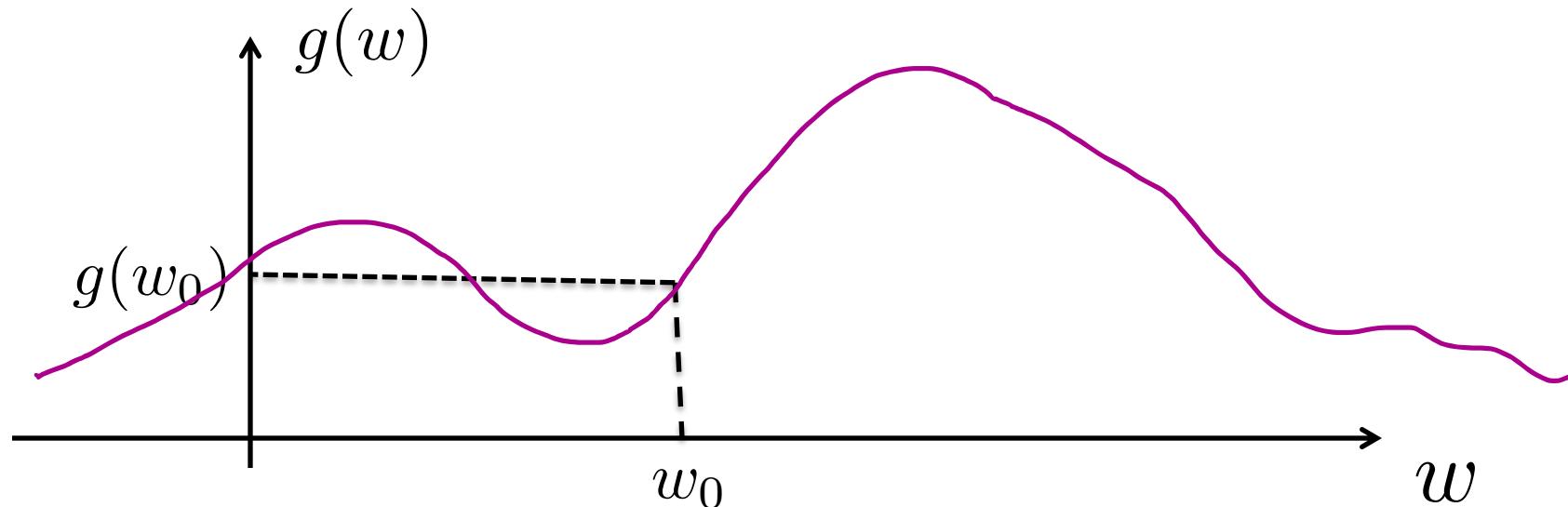
$$\nabla g = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2xy \\ x^2 \end{bmatrix}$$

$\begin{bmatrix} 2xy \\ x^2 \end{bmatrix}$

- What is the derivative of  $g(x, y)$  at  $x=0.5, y=0.5$ ?

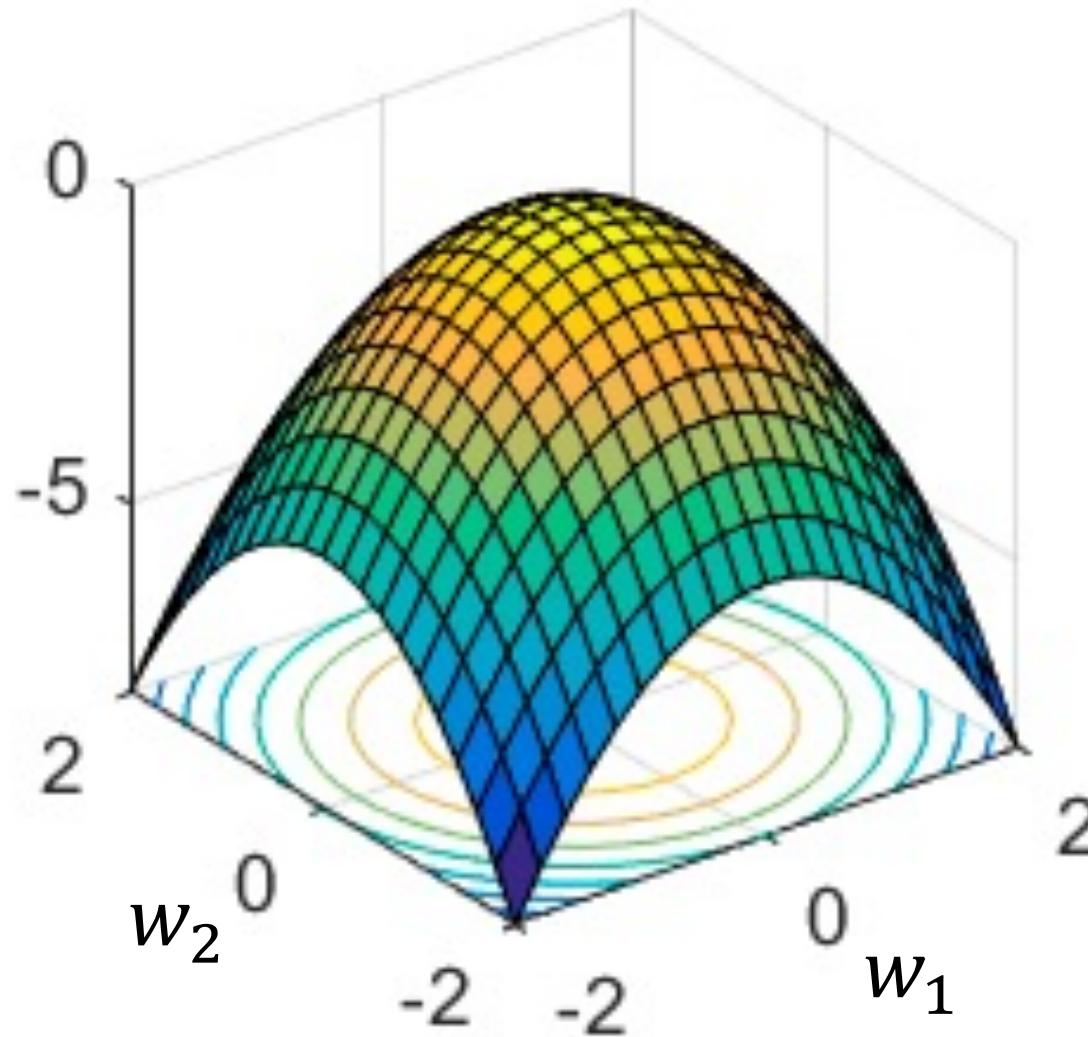
$$\nabla g|_{x=0.5, y=0.5} = \begin{bmatrix} 2(0.5)(0.5) \\ (0.5^2) \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.25 \end{bmatrix}$$

# 1-D Optimization



- Could evaluate  $g(w_0 + h)$  and  $g(w_0 - h)$ 
  - Then step in best direction
- Or, evaluate derivative: 
$$\frac{\partial g(w_0)}{\partial w} = \lim_{h \rightarrow 0} \frac{g(w_0 + h) - g(w_0 - h)}{2h}$$
  - Tells which direction to step into

# 2-D Optimization



# Gradient Ascent

- Perform update in uphill direction for each coordinate
- The steeper the slope (i.e. the higher the derivative) the bigger the step for that coordinate
- E.g., consider:  $g(w_1, w_2)$

- Updates:

$$w_1 \leftarrow w_1 + \alpha * \frac{\partial g}{\partial w_1}(w_1, w_2)$$

$$w_2 \leftarrow w_2 + \alpha * \frac{\partial g}{\partial w_2}(w_1, w_2)$$

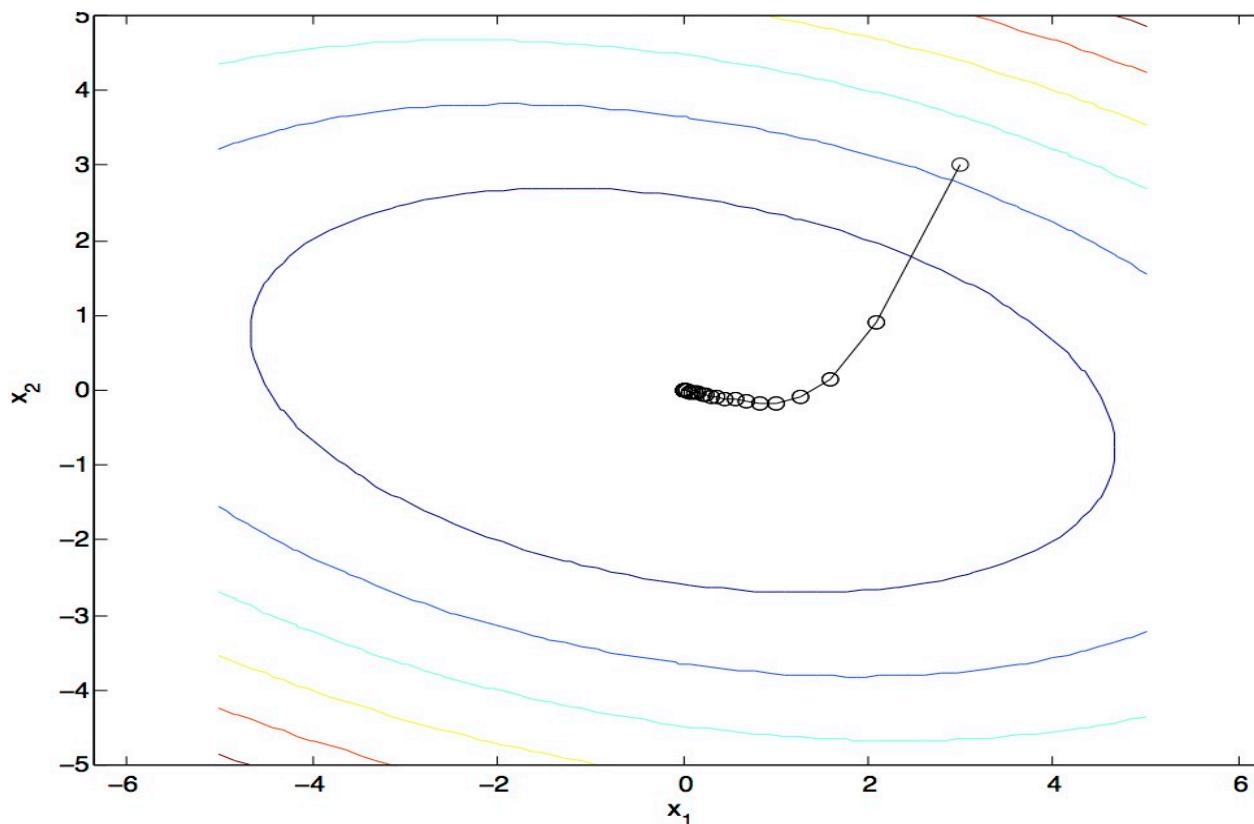
- Updates in vector notation:

$$w \leftarrow w + \alpha * \nabla_w g(w)$$

with:  $\nabla_w g(w) = \begin{bmatrix} \frac{\partial g}{\partial w_1}(w) \\ \frac{\partial g}{\partial w_2}(w) \end{bmatrix}$  = gradient

# Gradient Ascent

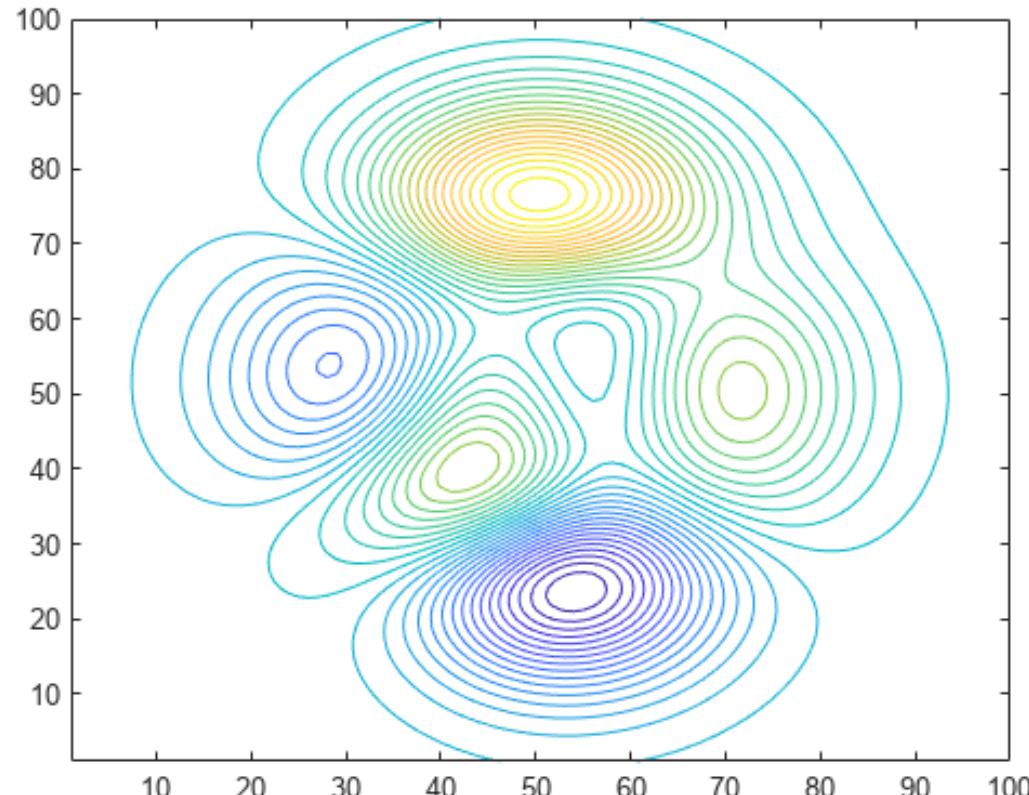
- Idea:
  - Start somewhere
  - Repeat: Take a step in the gradient direction



# Gradient Ascent

- Idea:
  - Start somewhere
  - Repeat: Take a step in the gradient direction

Not guaranteed to find  
global maximum:



# What is the Steepest Direction?\*

$$\max_{\Delta: \Delta_1^2 + \Delta_2^2 \leq \varepsilon} g(w + \Delta)$$



- First-Order Taylor Expansion:

$$g(w + \Delta) \approx g(w) + \frac{\partial g}{\partial w_1} \Delta_1 + \frac{\partial g}{\partial w_2} \Delta_2$$

- Steepest Descent Direction:

$$\max_{\Delta: \Delta_1^2 + \Delta_2^2 \leq \varepsilon} g(w) + \frac{\partial g}{\partial w_1} \Delta_1 + \frac{\partial g}{\partial w_2} \Delta_2$$

- Recall:

$$\max_{\Delta: \|\Delta\| \leq \varepsilon} \Delta^\top a \quad \rightarrow \quad \Delta = \varepsilon \frac{a}{\|a\|}$$

- Hence, solution:  $\Delta = \varepsilon \frac{\nabla g}{\|\nabla g\|}$

**Gradient direction = steepest direction!**

$$\nabla g = \begin{bmatrix} \frac{\partial g}{\partial w_1} \\ \frac{\partial g}{\partial w_2} \end{bmatrix}$$

# Gradient in n dimensions

---

$$\nabla g = \begin{bmatrix} \frac{\partial g}{\partial w_1} \\ \frac{\partial g}{\partial w_2} \\ \vdots \\ \frac{\partial g}{\partial w_n} \end{bmatrix}$$

# Optimization Procedure: Gradient Ascent

---

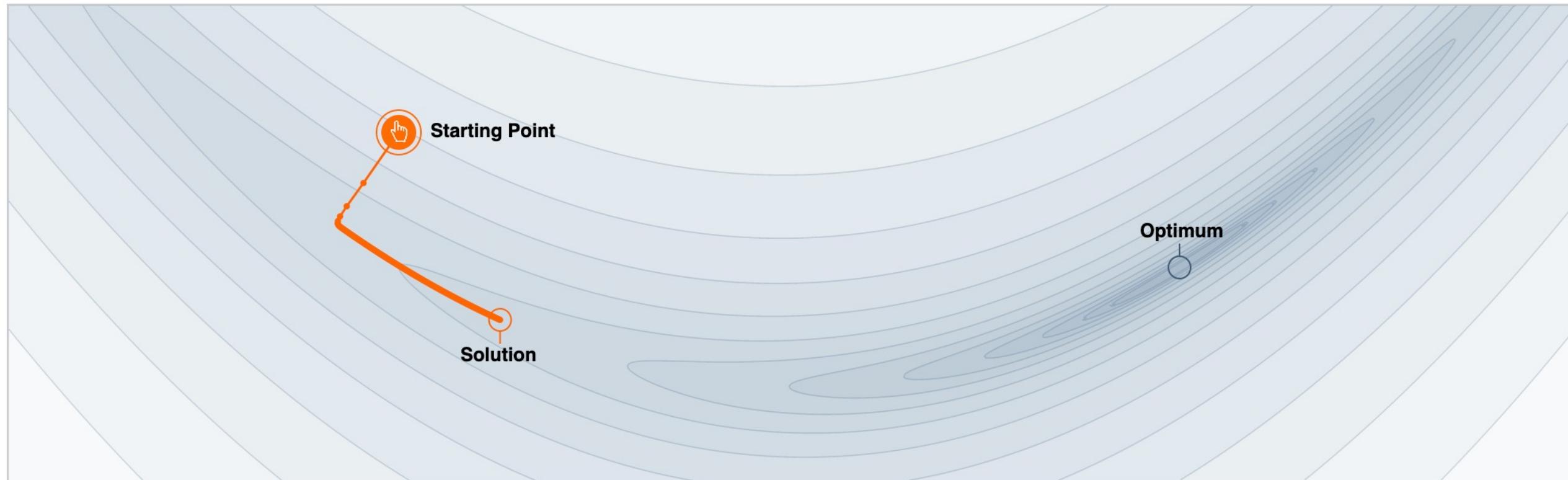
```
Init  $w$ 
for iter = 1, 2, ...
 $w \leftarrow w + \alpha \cdot \nabla g(w)$ 
```

- $\alpha$ : learning rate --- tweaking parameter that needs to be chosen carefully
- How? Try multiple choices
  - Crude rule of thumb: update changes  $w$  about 0.1 – 1 %

# Learning Rate

Choice of learning rate  $\alpha$  is a hyperparameter

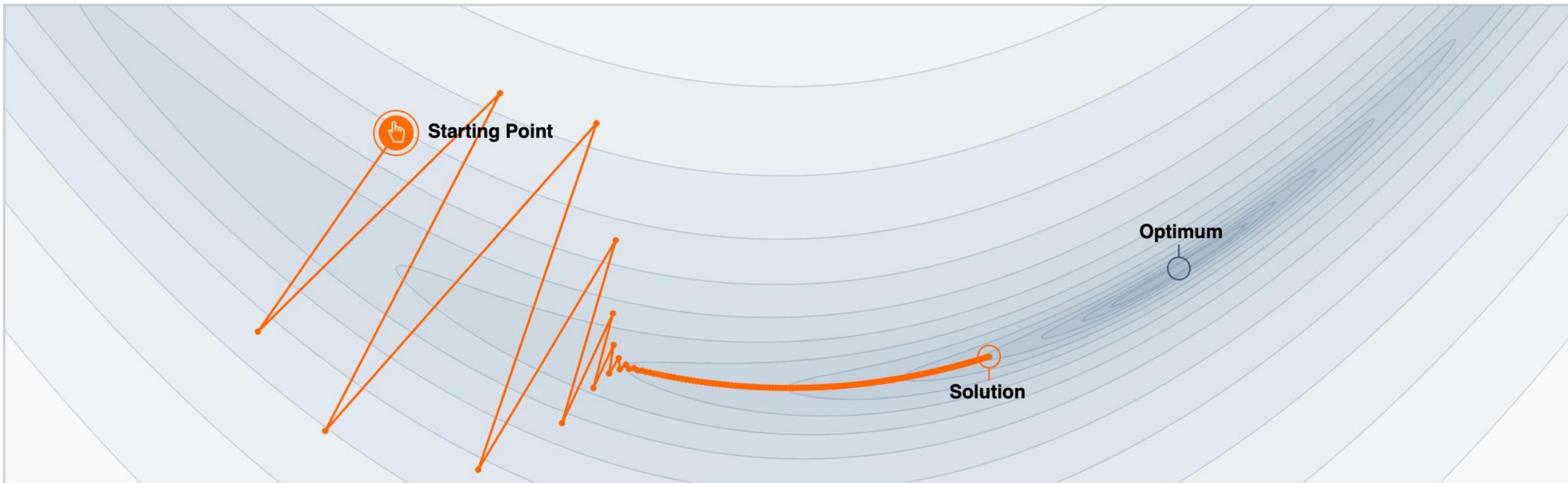
Example:  $\alpha=0.001$  (too small)



# Learning Rate

Choice of step size  $\alpha$  is a hyperparameter

Example:  $\alpha=0.004$  (too large)



# Gradient Ascent with Momentum\*

- Often use *momentum* to improve gradient ascent convergence

Gradient Ascent:

```
Init w  
for iter = 1, 2, ...  
     $w \leftarrow w + \alpha \cdot \nabla g(w)$ 
```

Gradient Ascent with momentum:

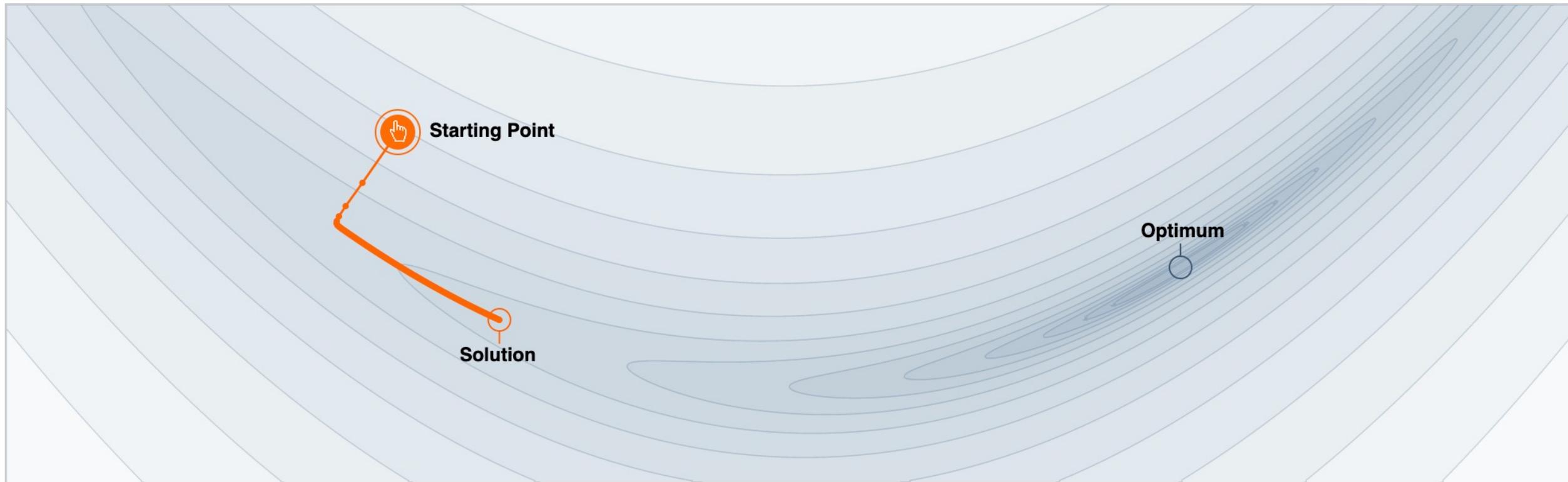
```
Init w  
for iter = 1, 2, ...  
     $z \leftarrow \beta \cdot z + \nabla g(w)$   
     $w \leftarrow w + \alpha \cdot z$ 
```

$$= w + \alpha (\beta z + \nabla g(w))$$

- One interpretation:  $w$  moves like a particle with mass
- Another: *exponential moving average* on gradient

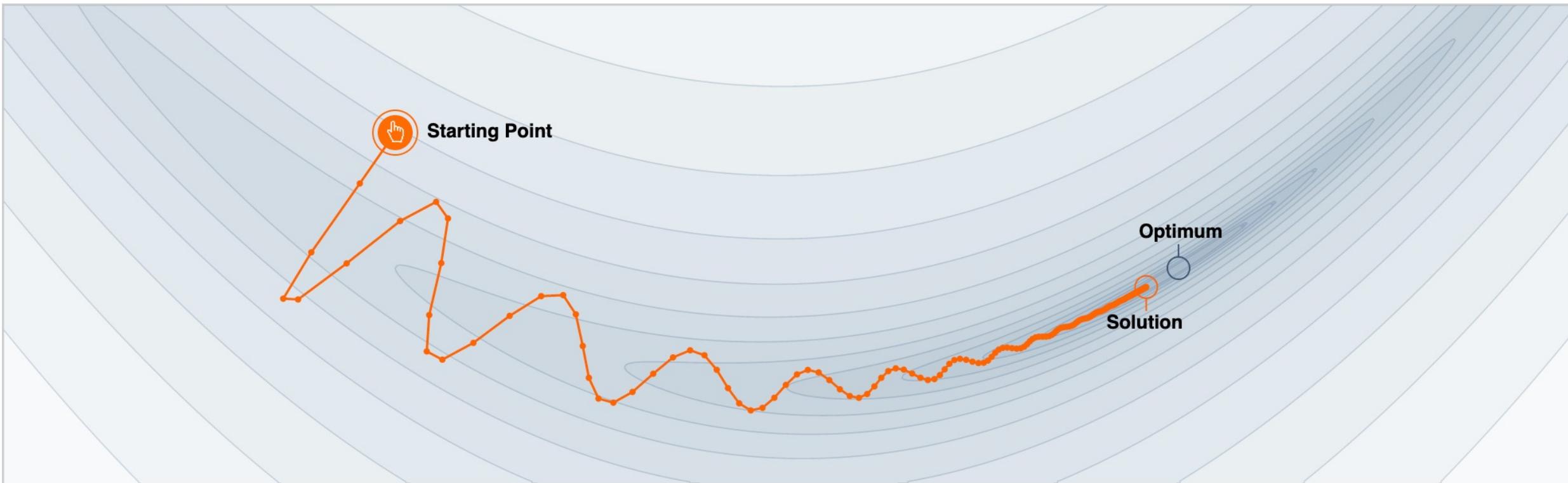
# Gradient Ascent with Momentum\*

Example:  $\alpha=0.001$  and  $\beta=0.0$



# Gradient Ascent with Momentum\*

Example:  $\alpha=0.001$  and  $\beta=0.9$



# Batch Gradient Ascent on the Log Likelihood Objective

$$\max_w \text{ll}(w) = \max_w \underbrace{\sum_i \log P(y^{(i)} | x^{(i)}; w)}_{g(w)}$$

- init  $w$
- for iter = 1, 2, ...

$$w \leftarrow w + \alpha * \sum_i \nabla \log P(y^{(i)} | x^{(i)}; w)$$

# Stochastic Gradient Ascent on the Log Likelihood Objective

$$\max_w \text{ll}(w) = \max_w \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

**Observation:** once gradient on one training example has been computed, might as well incorporate before computing next one

- init  $w$
- for iter = 1, 2, ...
  - pick random  $j$

$$w \leftarrow w + \alpha * \nabla \log P(y^{(j)} | x^{(j)}; w)$$

# Mini-Batch Gradient Ascent on the Log Likelihood Objective

$$\max_w \text{ll}(w) = \max_w \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

**Observation:** gradient over small set of training examples (=mini-batch) can be computed in parallel, might as well do that instead of a single one

- init  $w$
- for iter = 1, 2, ...
  - pick random subset of training examples  $J$

$$w \leftarrow w + \alpha * \sum_{j \in J} \nabla \log P(y^{(j)} | x^{(j)}; w)$$

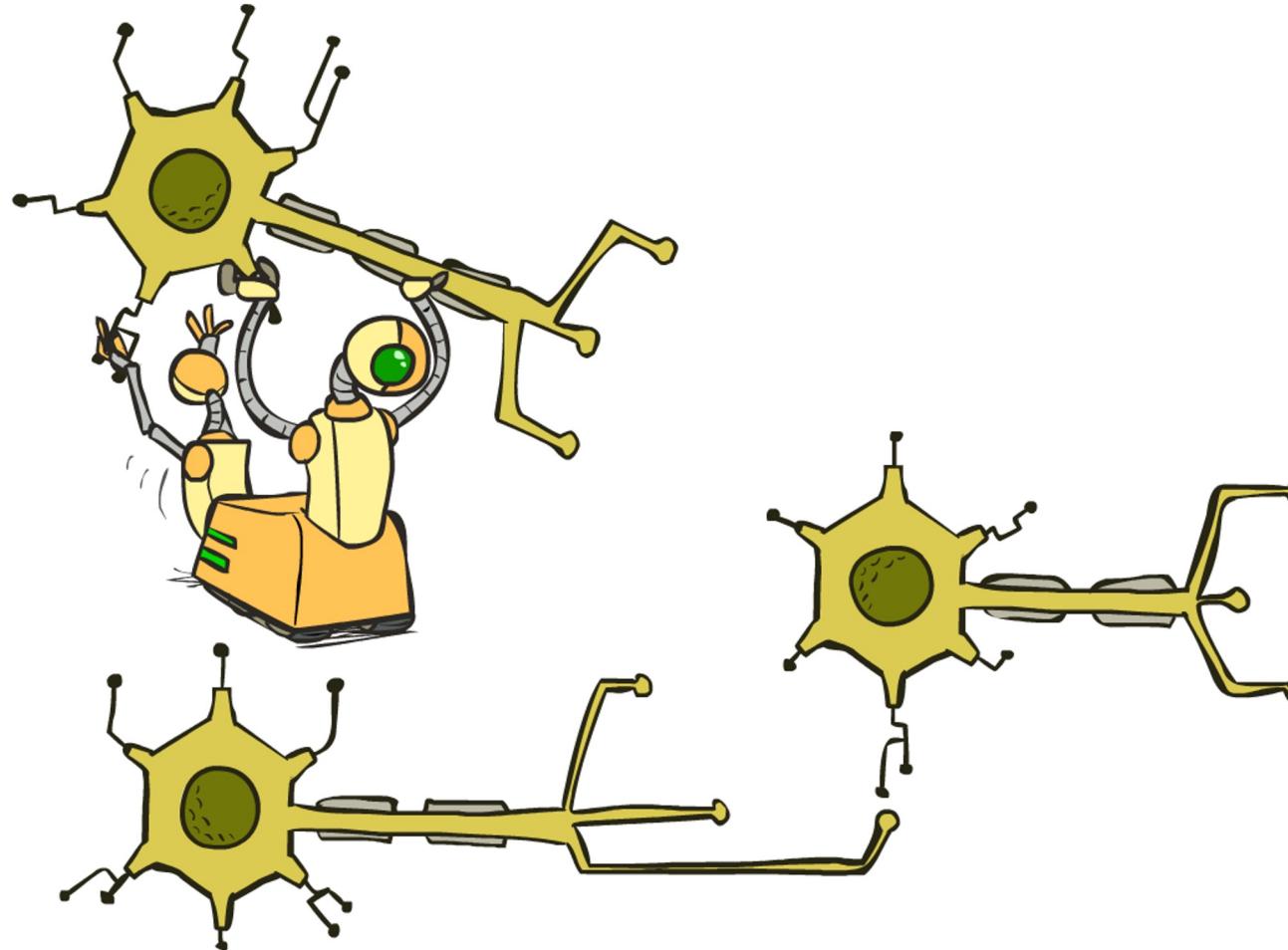
# How about computing all the derivatives?

---

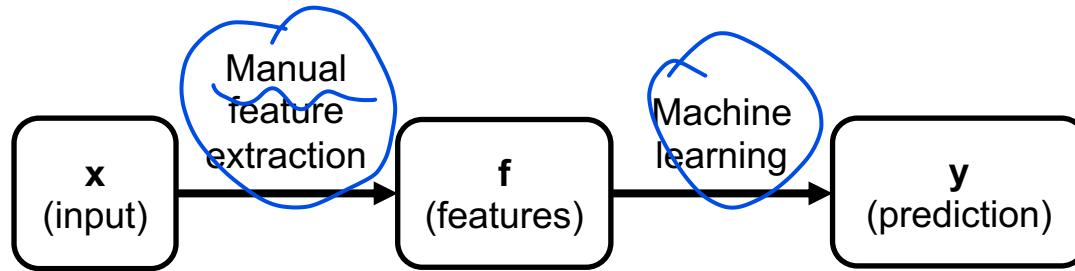
- We'll talk about that once we covered neural networks, which are a generalization of logistic regression

# Neural Networks

---

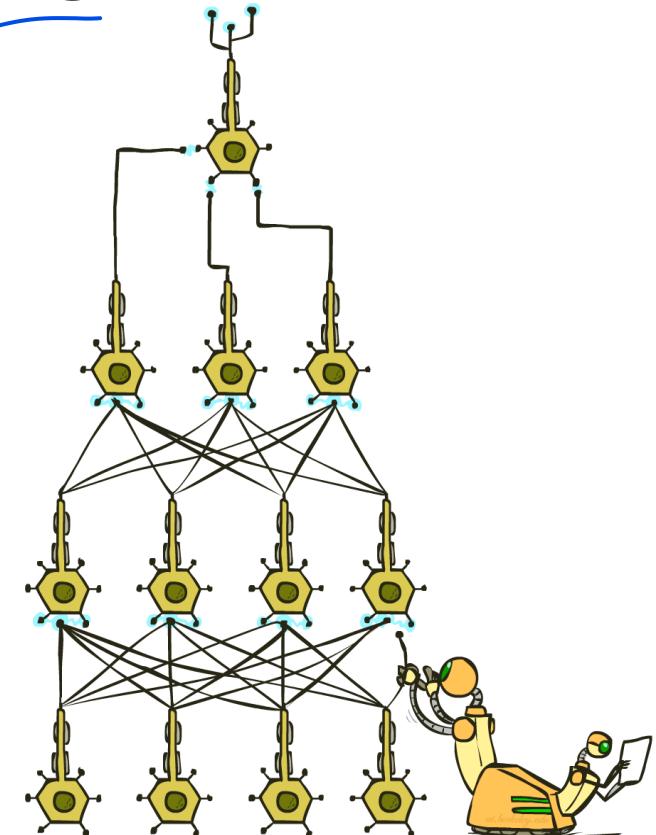


# Manual Feature Design vs. Deep Learning

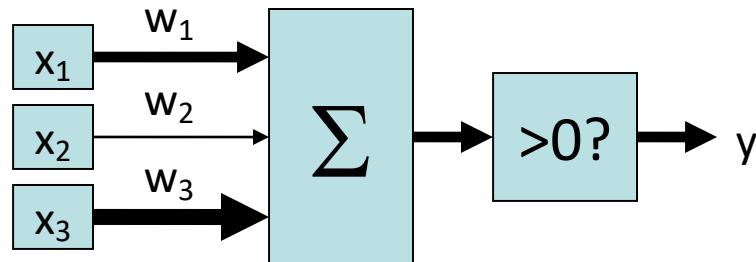


- Manual feature design requires:
  - Domain-specific expertise
  - Domain-specific effort

- What if we could learn the features, too?
- **Deep Learning**



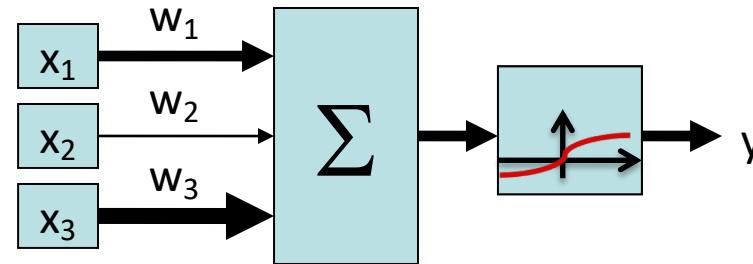
# Review: Perceptron



$$y = \begin{cases} 1 & w_1x_1 + w_2x_2 + w_3x_3 > 0 \\ 0 & \text{otherwise} \end{cases}$$

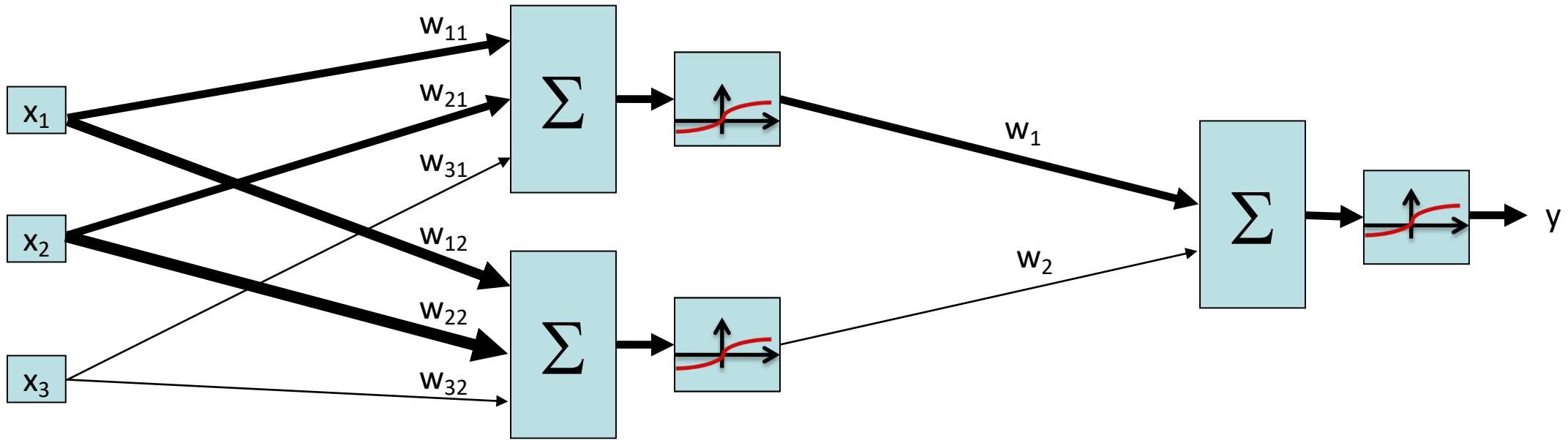
# Review: Perceptron with Sigmoid Activation

---

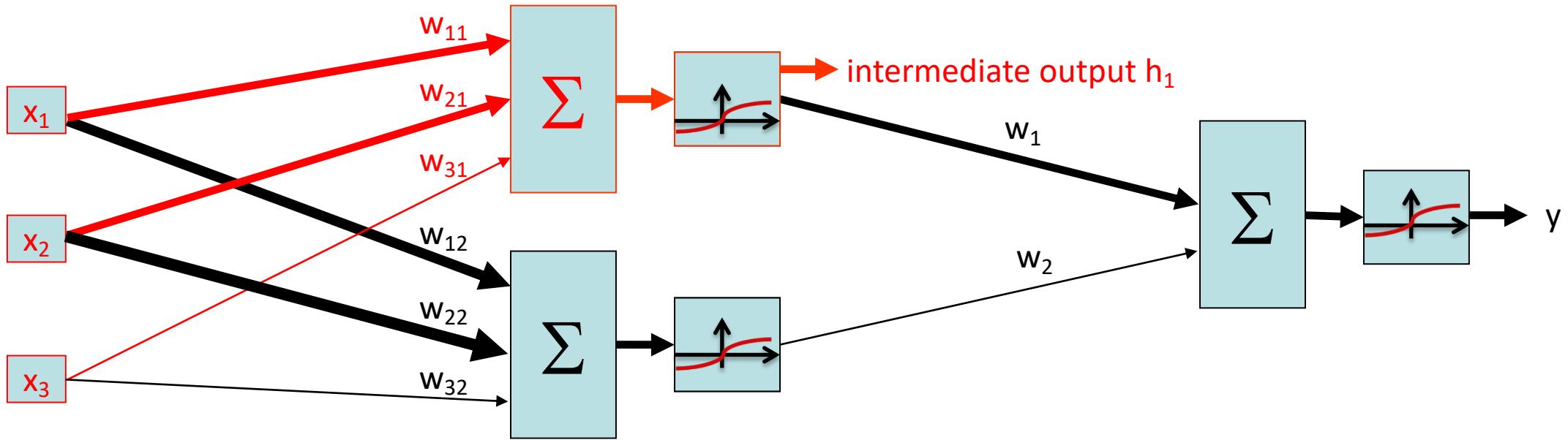


$$\begin{aligned}y &= \phi(w_1x_1 + w_2x_2 + w_3x_3) \\&= \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + w_3x_3)}}\end{aligned}$$

# 2-Layer, 2-Neuron Neural Network

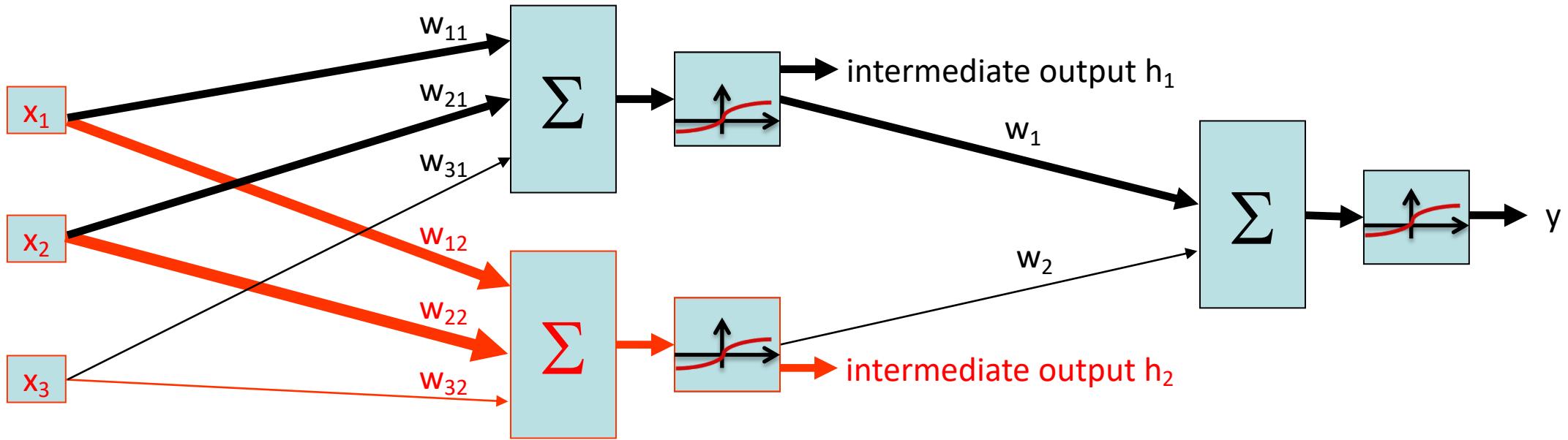


# 2-Layer, 2-Neuron Neural Network



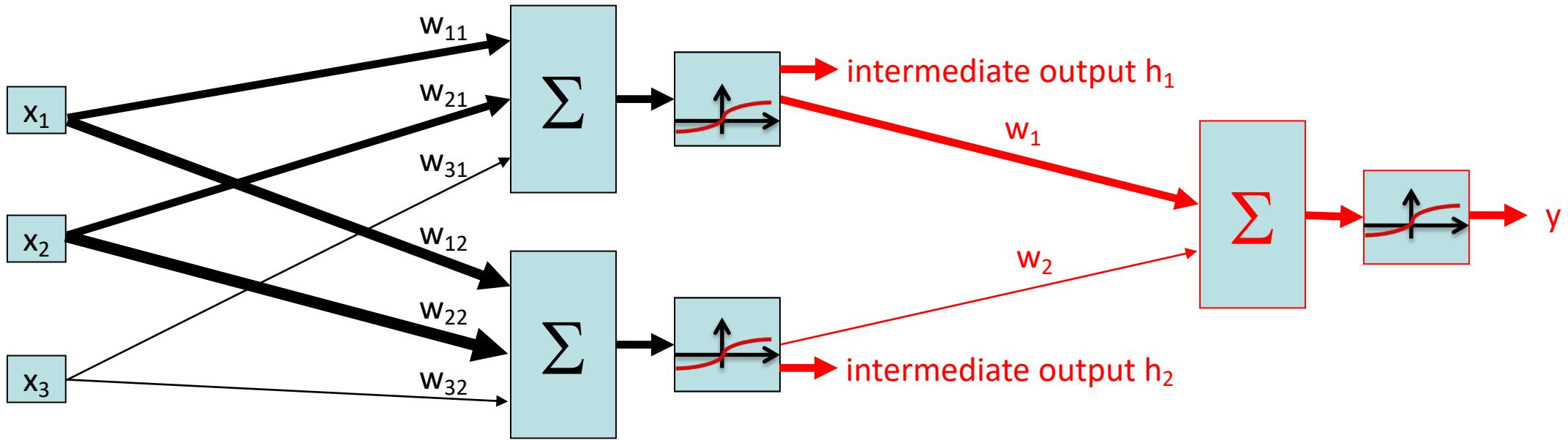
$$\begin{aligned}\text{intermediate output } h_1 &= \phi(w_{11}x_1 + w_{21}x_2 + w_{31}x_3) \\ &= \frac{1}{1 + e^{-(w_{11}x_1 + w_{21}x_2 + w_{31}x_3)}}\end{aligned}$$

# 2-Layer, 2-Neuron Neural Network



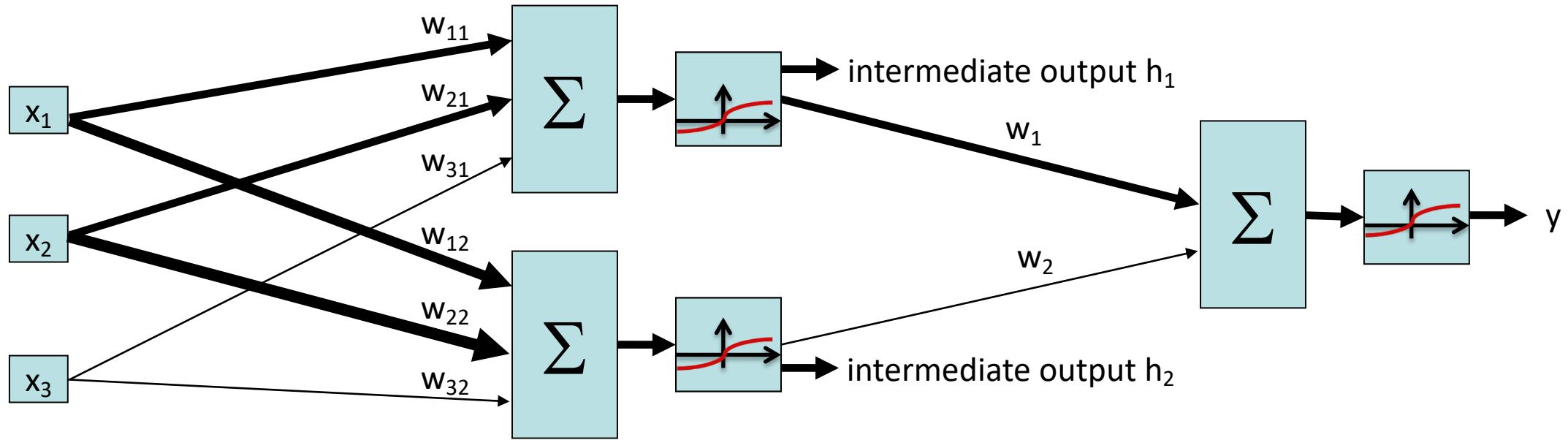
$$\begin{aligned} \text{intermediate output } h_2 &= \phi(w_{12}x_1 + w_{22}x_2 + w_{32}x_3) \\ &= \frac{1}{1 + e^{-(w_{12}x_1 + w_{22}x_2 + w_{32}x_3)}} \end{aligned}$$

# 2-Layer, 2-Neuron Neural Network



$$\begin{aligned}y &= \phi(w_1 h_1 + w_2 h_2) \\&= \frac{1}{1 + e^{-(w_1 h_1 + w_2 h_2)}}\end{aligned}$$

# 2-Layer, 2-Neuron Neural Network



$$\begin{aligned}y &= \phi(w_1 h_1 + w_2 h_2) \\&= \phi(w_1 \phi(w_{11} x_1 + w_{21} x_2 + w_{31} x_3) + w_2 \phi(w_{12} x_1 + w_{22} x_2 + w_{32} x_3))\end{aligned}$$

# 2-Layer, 2-Neuron Neural Network

---

$$\begin{aligned}y &= \phi(w_1 h_1 + w_2 h_2) \\&= \phi(w_1 \phi(w_{11}x_1 + w_{21}x_2 + w_{31}x_3) + w_2 \phi(w_{12}x_1 + w_{22}x_2 + w_{32}x_3))\end{aligned}$$

The same equation, formatted with matrices:

$$\begin{aligned}&\phi \left( \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \right) \\&= \phi \left( \begin{bmatrix} w_{11}x_1 + w_{21}x_2 + w_{31}x_3 & w_{12}x_1 + w_{22}x_2 + w_{32}x_3 \end{bmatrix} \right) \\&= \begin{bmatrix} h_1 & h_2 \end{bmatrix}\end{aligned}$$

$$\phi \left( \begin{bmatrix} h_1 & h_2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \right) = \phi(w_1 h_1 + w_2 h_2) = y$$

The same equation, formatted more compactly by introducing variables representing each matrix:

$$\phi(x \times W_{\text{layer 1}}) = h \quad \phi(h \times W_{\text{layer 2}}) = y$$

# 2-Layer, 2-Neuron Neural Network

$$\phi(x \times W_{\text{layer 1}}) = h$$

Shape (1, 3).  
Input feature vector.

Shape (3, 2).  
Weights to be learned.

Shape (1, 2).  
Outputs of layer 1,  
inputs to layer 2.

```
graph TD; x["Shape (1, 3).  
Input feature vector."] --> prod1["x × Wlayer 1"]; W1["Shape (3, 2).  
Weights to be learned."] --> prod1; prod1 --> h["Shape (1, 2).  
Outputs of layer 1,  
inputs to layer 2."];
```

$$\phi(h \times W_{\text{layer 2}}) = y$$

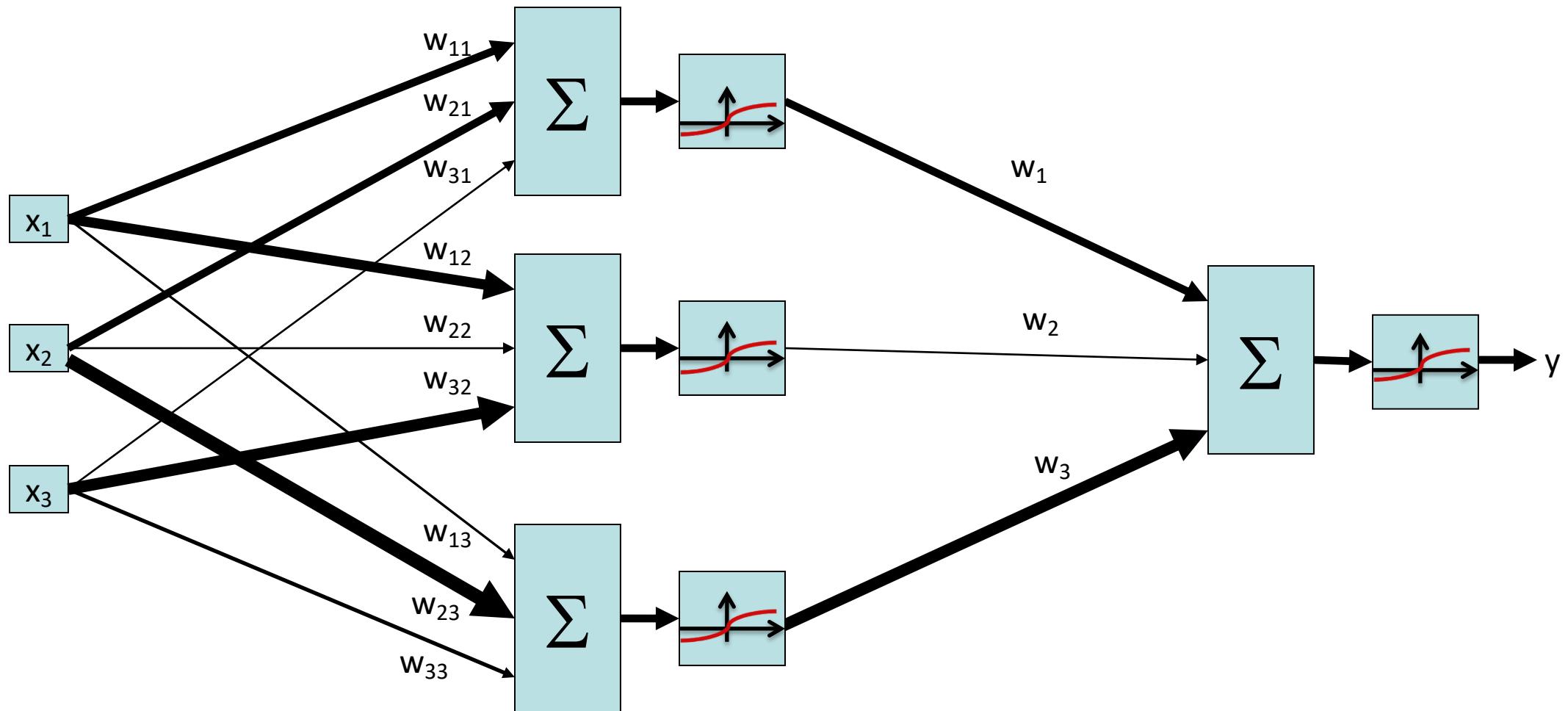
Shape (1, 2).  
Outputs of layer 1,  
inputs to layer 2.

Shape (2, 1).  
Weights to be learned.

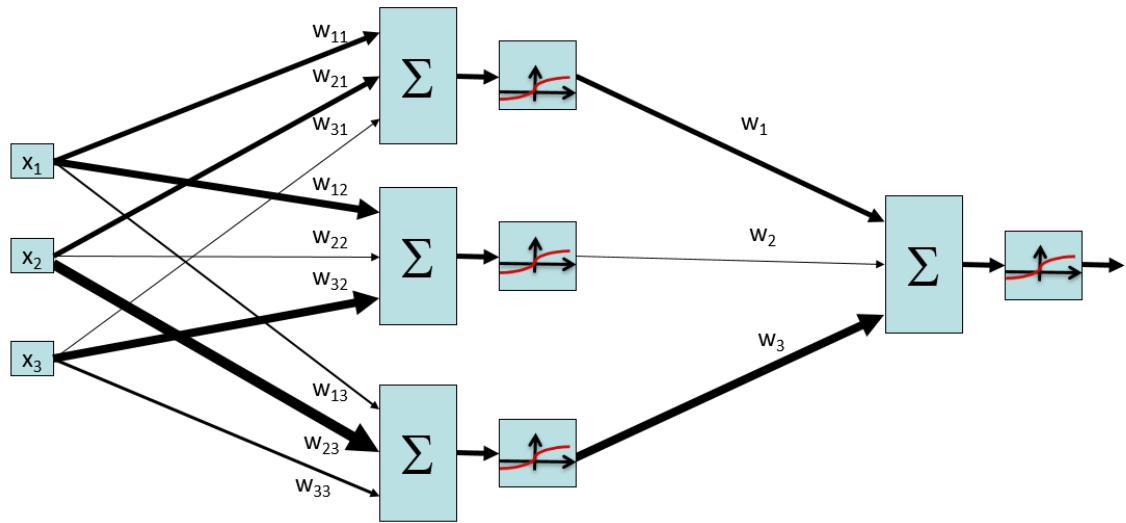
Shape (1, 1).  
Output of network.

```
graph TD; h["Shape (1, 2).  
Outputs of layer 1,  
inputs to layer 2."] --> prod2["h × Wlayer 2"]; W2["Shape (2, 1).  
Weights to be learned."] --> prod2; prod2 --> y["Shape (1, 1).  
Output of network."];
```

# 2-Layer, 3-Neuron Neural Network



# 2-Layer, 3-Neuron Neural Network



$$\begin{aligned}
 & \phi \left( \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \right) \\
 &= \phi \left( \begin{bmatrix} w_{11}x_1 + w_{21}x_2 + w_{31}x_3 & w_{12}x_1 + w_{22}x_2 + w_{32}x_3 & w_{13}x_1 + w_{23}x_2 + w_{33}x_3 \end{bmatrix} \right) \\
 &= \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix}
 \end{aligned}$$

$$\phi \left( \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} \right) = \phi(w_1h_1 + w_2h_2 + w_3h_3) = y$$

# 2-Layer, 3-Neuron Neural Network

$$\phi(x \times W_{\text{layer 1}}) = h$$

Shape (1, 3).  
Input feature vector.

Shape (3, 3).  
Weights to be learned

Shape (1, 3).  
Outputs of layer 1,  
inputs to layer 2.

```
graph TD; x["Shape (1, 3).  
Input feature vector."] --> prod1["phi(x * W_layer 1) = h"]; w1["Shape (3, 3).  
Weights to be learned"] --> prod1; h["Shape (1, 3).  
Outputs of layer 1,  
inputs to layer 2."] --> prod1;
```

$$\phi(h \times W_{\text{layer 2}}) = y$$

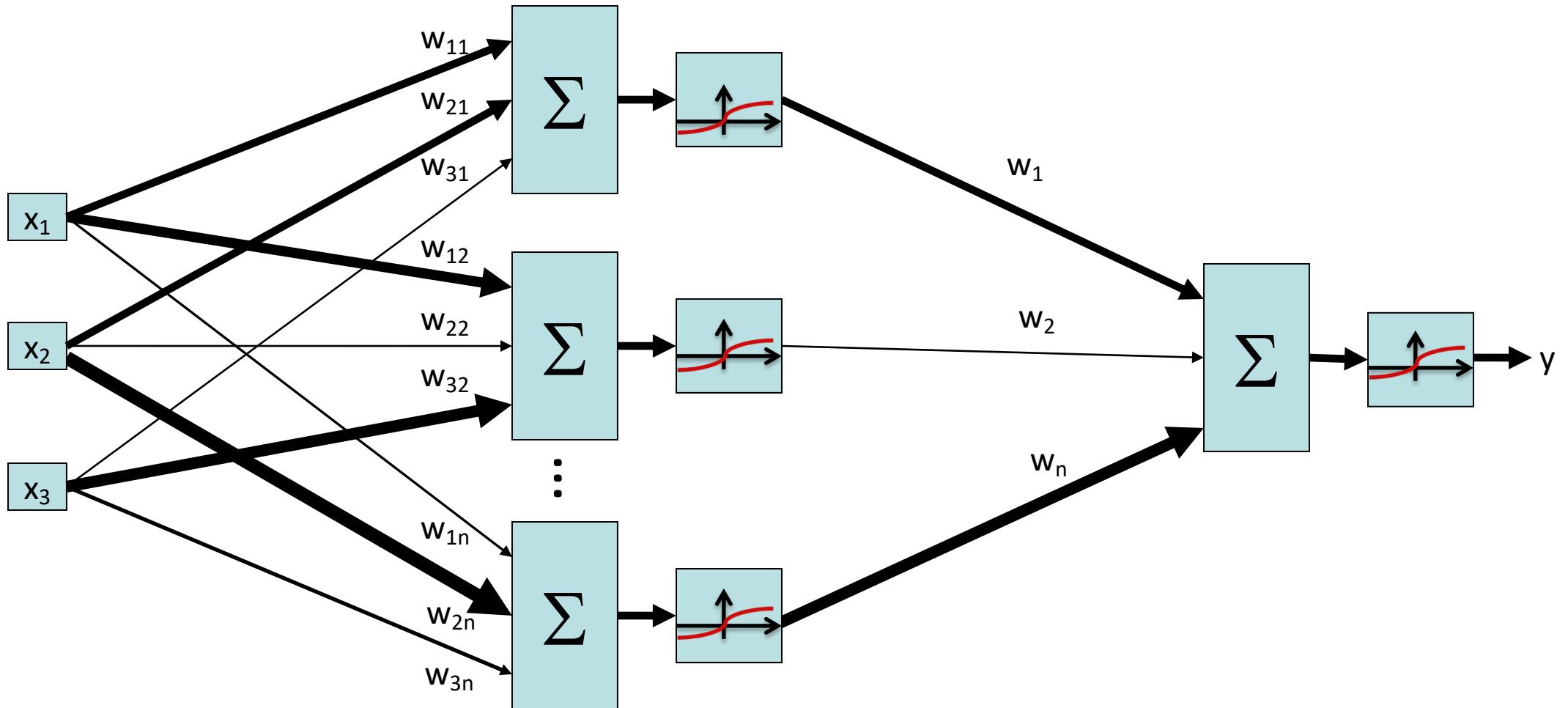
Shape (1, 3).  
Outputs of layer 1,  
inputs to layer 2.

Shape (3, 1).  
Weights to be learned.

Shape (1, 1).  
Output of network.

```
graph TD; h["Shape (1, 3).  
Outputs of layer 1,  
inputs to layer 2."] --> prod2["phi(h * W_layer 2) = y"]; w2["Shape (3, 1).  
Weights to be learned."] --> prod2; y["Shape (1, 1).  
Output of network."] --> prod2;
```

# Generalize: Number of hidden neurons



The hidden layer doesn't necessarily need to have 3 neurons; it could have any arbitrary number  $n$  neurons.

# Generalize: $n$ number of hidden neurons

$$\phi(x \times W_{\text{layer 1}}) = h$$

Shape (1, 3).  
Input feature vector.

Shape (3,  $n$ ).  
Weights to be learned

Shape (1,  $n$ ).  
Outputs of layer 1,  
inputs to layer 2.

$$\phi(h \times W_{\text{layer 2}}) = y$$

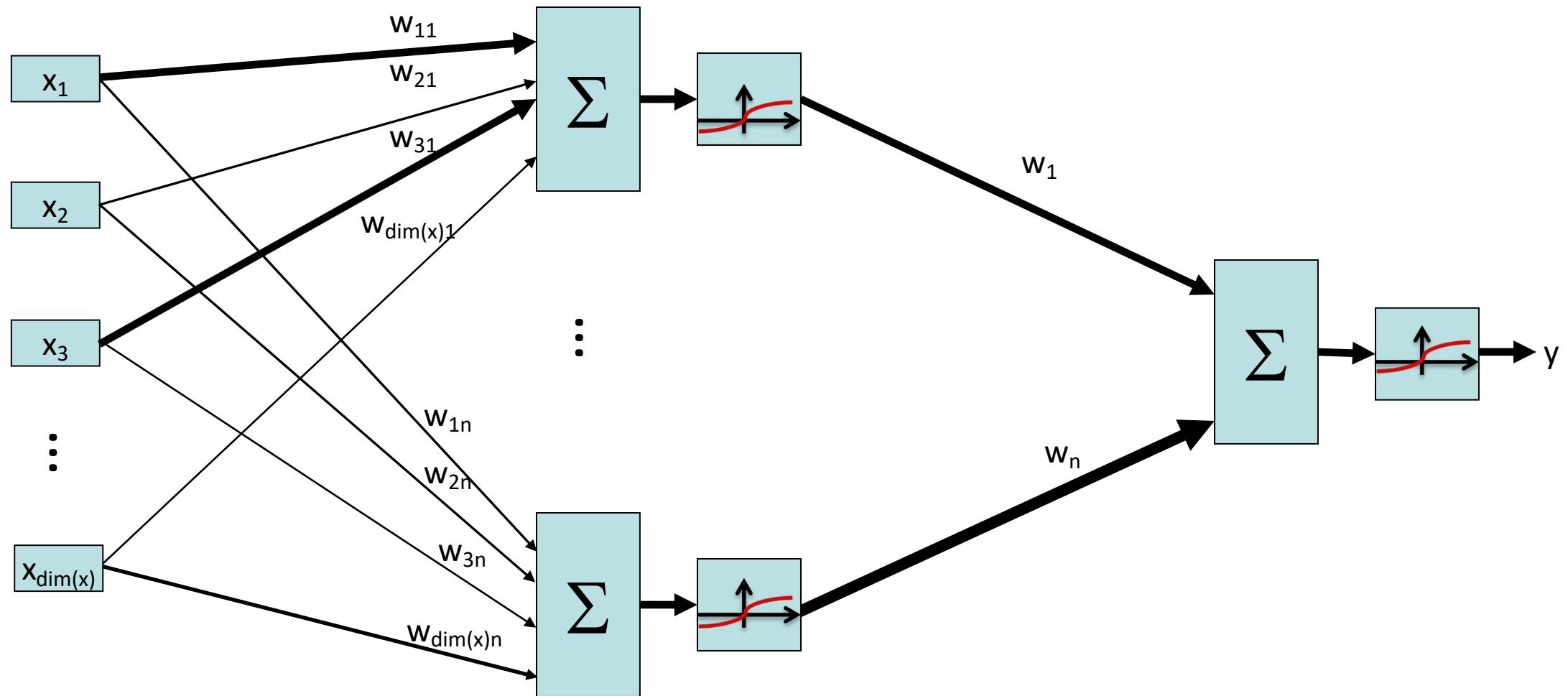
Shape (1,  $n$ ).  
Outputs of layer 1,  
inputs to layer 2.

Shape ( $n$ , 1).  
Weights to be learned.

Shape (1, 1).  
Output of network.

The hidden layer doesn't necessarily need to have 3 neurons; it could have any arbitrary number  $n$  neurons.

# Generalize: Number of input features



The input feature vector doesn't necessarily need to have 3 features; it could have some arbitrary number  $\text{dim}(x)$  of features.

# Generalize: Number of input features

$$\phi(x \times W_{\text{layer 1}}) = h$$

Shape (1,  $\text{dim}(x)$ ).  
Input feature vector.

Shape ( $\text{dim}(x)$ , n).  
Weights to be learned

Shape (1, n).  
Outputs of layer 1,  
inputs to layer 2.

```
graph TD; A["Shape (1, " <math>\text{dim}(x)</math>).  
Input feature vector."] --> C["Shape (1, n).  
Outputs of layer 1,  
inputs to layer 2."]; B["Shape (" <math>\text{dim}(x)</math>, n).  
Weights to be learned"] --> C; C["<math>\phi(x \times W_{\text{layer 1}}) = h</math>"]
```

$$\phi(h \times W_{\text{layer 2}}) = y$$

Shape (1, n).  
Outputs of layer 1,  
inputs to layer 2.

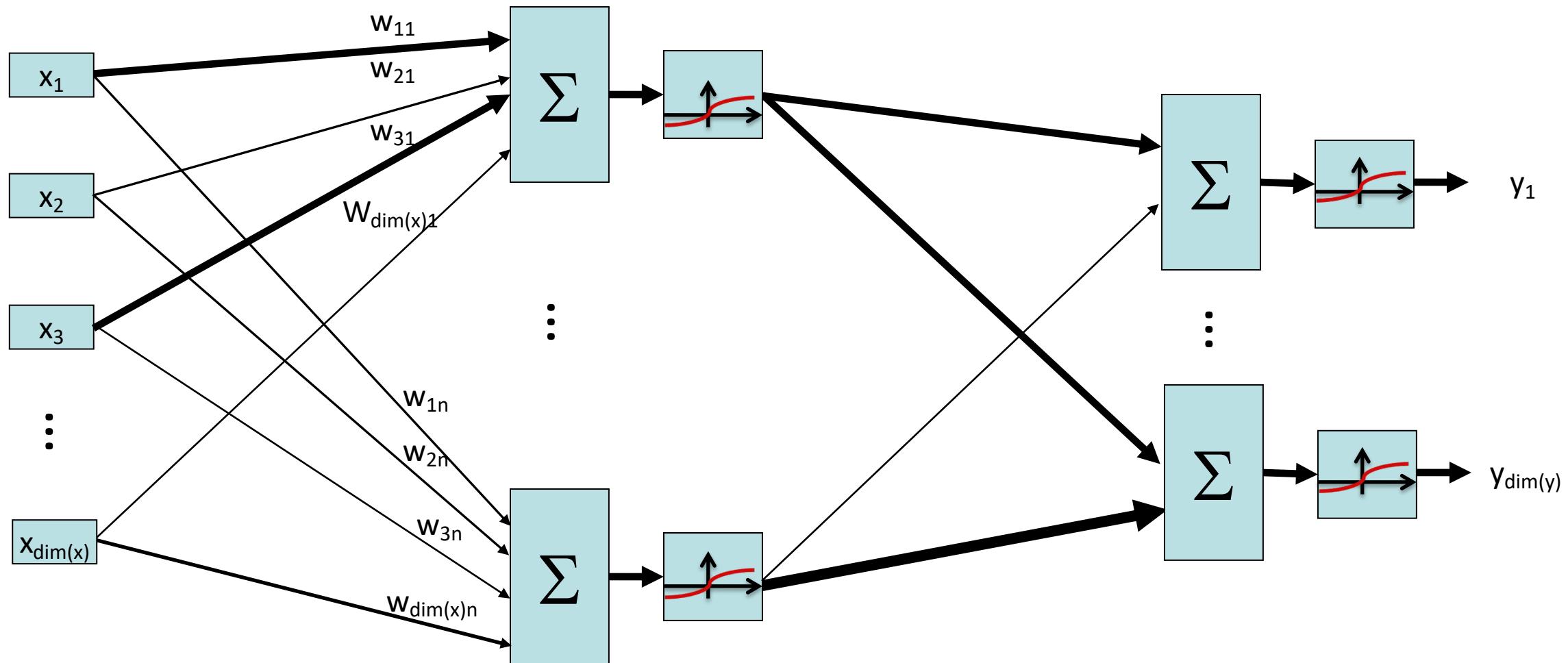
Shape (n, 1).  
Weights to be learned.

Shape (1, 1).  
Output of network.

```
graph TD; A["Shape (1, n).  
Outputs of layer 1,  
inputs to layer 2."] --> C["Shape (1, 1).  
Output of network."]; B["Shape (n, 1).  
Weights to be learned."] --> C; C["<math>\phi(h \times W_{\text{layer 2}}) = y</math>"]
```

The input feature vector doesn't necessarily need to have 3 features; it could have some arbitrary number  $\text{dim}(x)$  of features.

# Generalize: Number of outputs



The output doesn't necessarily need to be just one number; it could be some arbitrary  $\dim(y)$  length vector.

# Generalize: Number of input features

$$\phi(x \times W_{\text{layer 1}}) = h$$

Shape  $(1, \dim(x))$ .  
Input feature vector.

Shape  $(\dim(x), n)$ .  
Weights to be learned

Shape  $(1, n)$ .  
Outputs of layer 1,  
inputs to layer 2.

```
graph TD; A["Shape (1, dim(x)).  
Input feature vector."] --> C["Shape (1, n).  
Outputs of layer 1,  
inputs to layer 2."]; B["Shape (dim(x), n).  
Weights to be learned"] --> C;
```

$$\phi(h \times W_{\text{layer 2}}) = y$$

Shape  $(1, n)$ .  
Outputs of layer 1,  
inputs to layer 2.

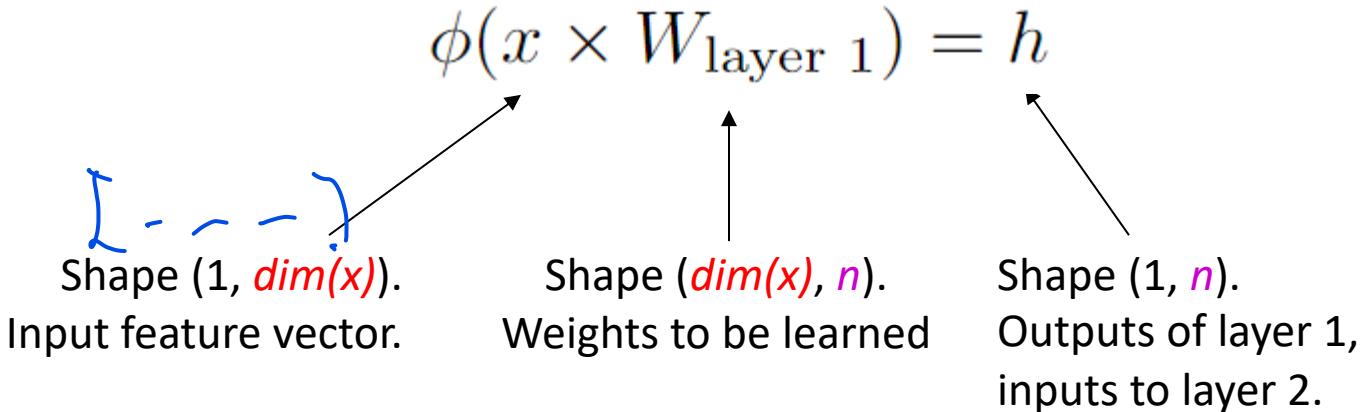
Shape  $(n, \dim(y))$ .  
Weights to be learned.

Shape  $(1, \dim(y))$ .  
Output of network.

```
graph TD; A["Shape (1, n).  
Outputs of layer 1,  
inputs to layer 2."] --> C["Shape (1, dim(y)).  
Output of network."]; B["Shape (n, dim(y)).  
Weights to be learned."] --> C;
```

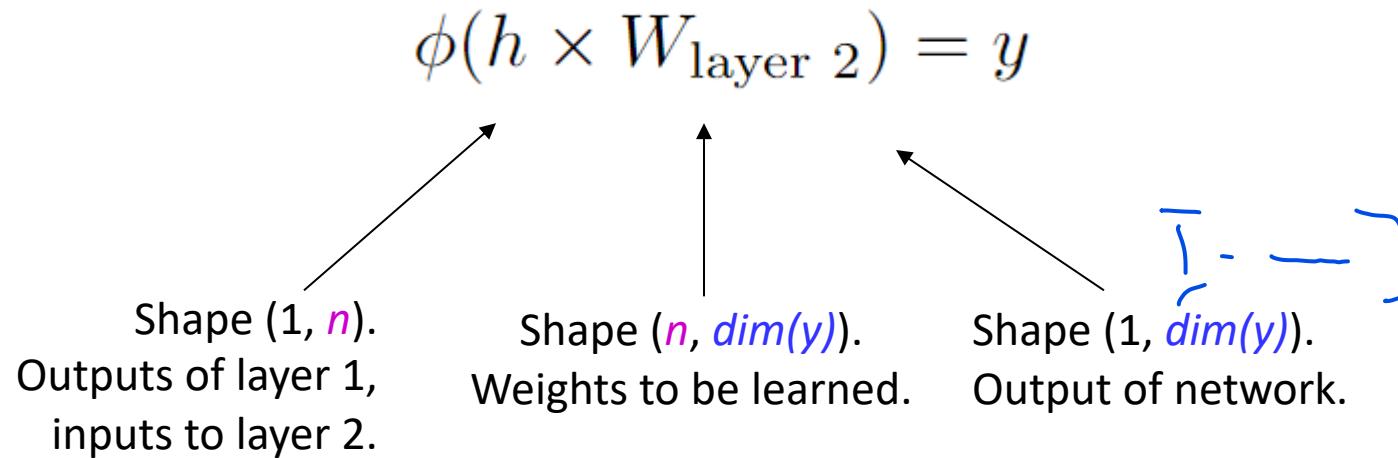
The output doesn't necessarily need to be just one number; it could be some arbitrary  $\dim(y)$  length vector.

# Generalized 2-Layer Neural Network



Layer 1 has weight matrix with shape  $(\dim(x), n)$ . These are the weights for  $n$  neurons, each taking  $\dim(x)$  features as input.

This transforms a  $\dim(x)$ -dimensional input vector into an  $n$ -dimensional output vector.

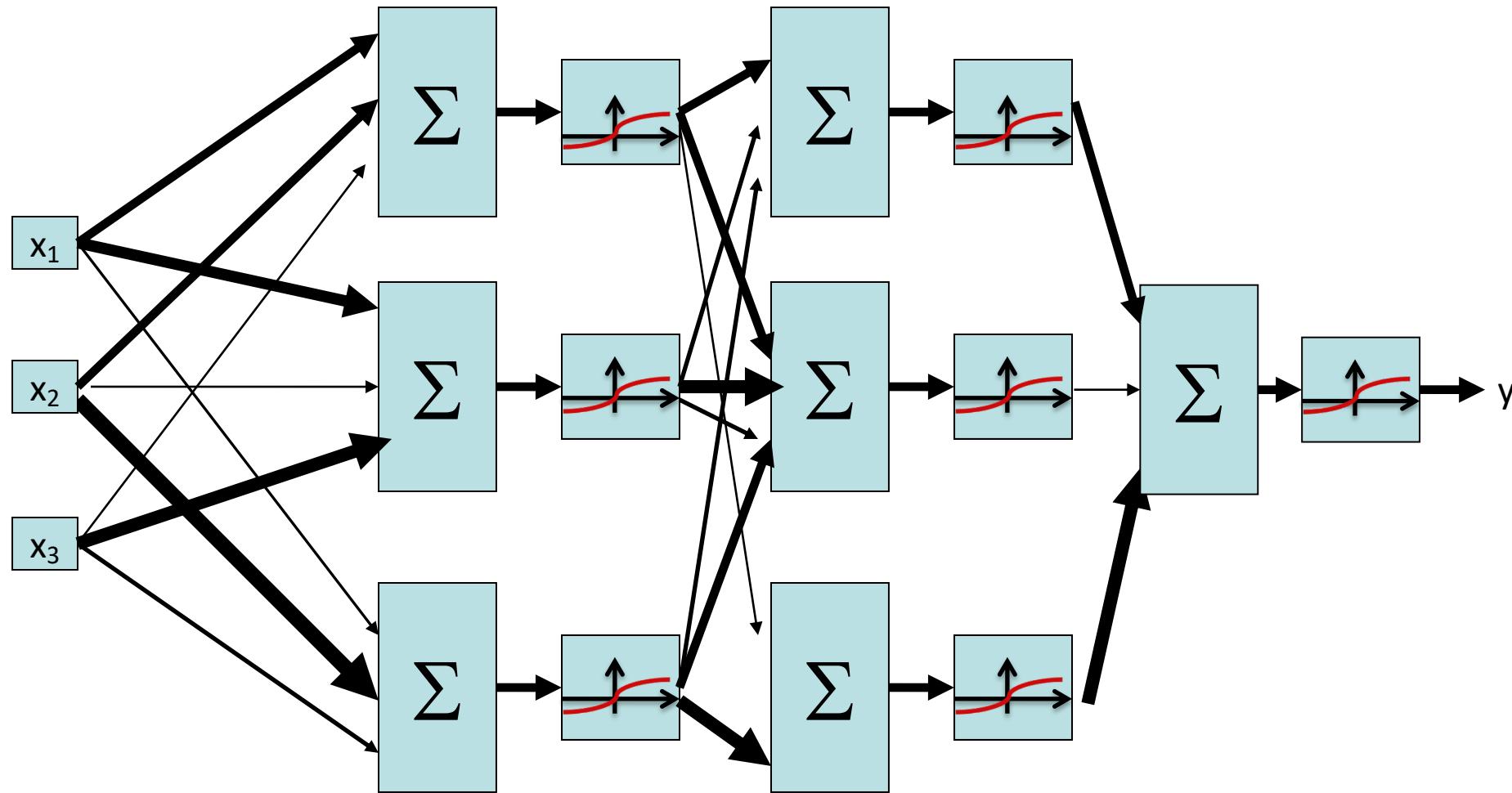


Layer 2 has weight matrix with shape  $(n, \dim(y))$ . These are the weights for  $\dim(y)$  neurons, each taking  $n$  features as input.

This transforms an  $n$ -dimensional input vector into a  $\dim(y)$ -dimensional output vector.

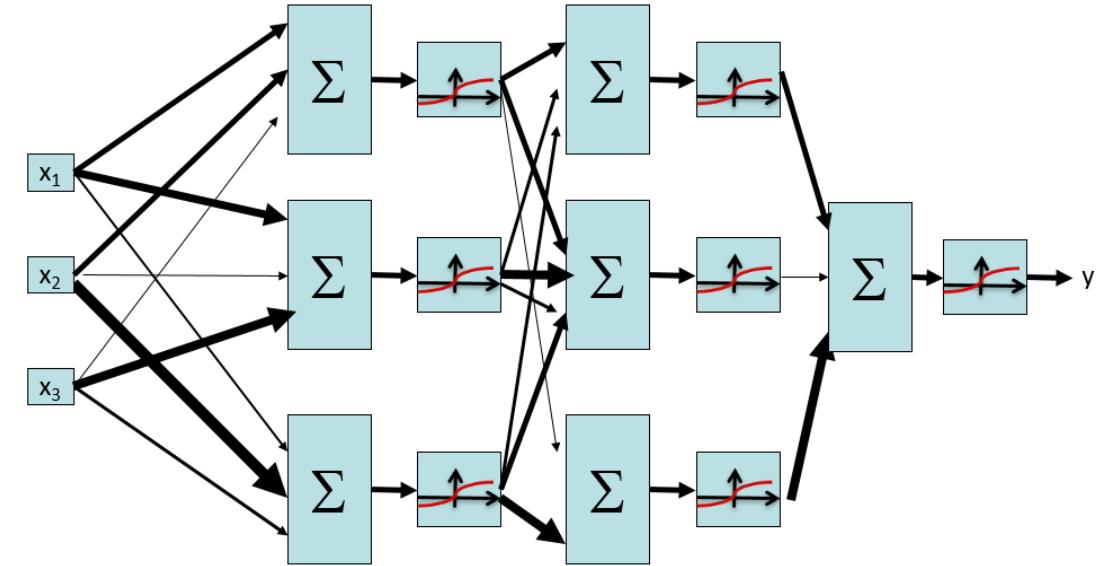
**Big idea:** The shape of a weight matrix is determined by the dimensions of the input and output of that layer.

# 3-Layer, 3-Neuron Neural Network



# 3-Layer, 3-Neuron Neural Network

- Layer 1:
  - $x$  has shape  $(1, 3)$ . Input vector, 3-dimensional.
  - $W_{\text{layer } 1}$  has shape  $(3, 3)$ . Weights for 3 neurons, each taking in a 3-dimensional input vector.
  - $h_{\text{layer } 1}$  has shape  $(1, 3)$ . Outputs of the 3 neurons at this layer.
- Layer 2:
  - $h_{\text{layer } 1}$  has shape  $(1, 3)$ . Outputs of the 3 neurons from the previous layer.
  - $W_{\text{layer } 2}$  has shape  $(3, 3)$ . Weights for 3 new neurons, each taking in the 3 previous perceptron outputs.
  - $h_{\text{layer } 2}$  has shape  $(1, 3)$ . Outputs of the 3 new neurons at this layer.
- Layer 3:
  - $h_{\text{layer } 2}$  has shape  $(1, 3)$ . Outputs from the previous layer.
  - $W_{\text{layer } 3}$  has shape  $(3, 1)$ . Weights for 1 final neuron, taking in the 3 previous perceptron outputs.
  - $y$  has shape  $(1, 1)$ . Output of the final neuron.



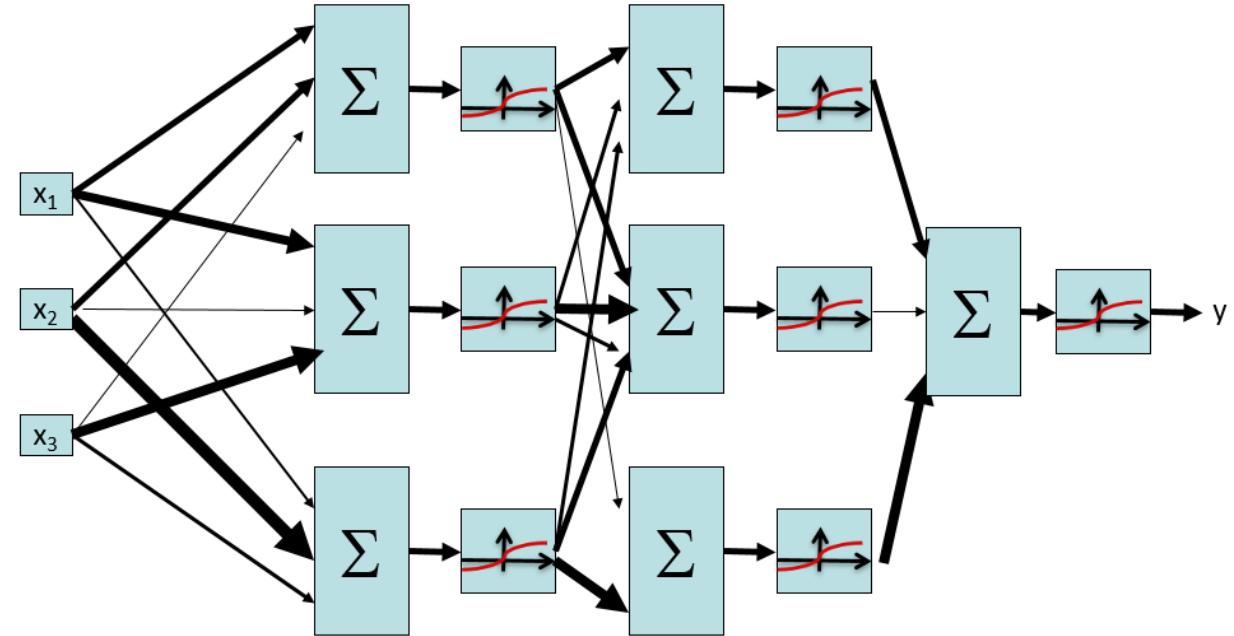
$$\phi(x \times W_{\text{layer } 1}) = h_{\text{layer } 1}$$

$$\phi(h_{\text{layer } 1} \times W_{\text{layer } 2}) = h_{\text{layer } 2}$$

$$\phi(h_{\text{layer } 2} \times W_{\text{layer } 3}) = y$$

# Generalized 3-Layer Neural Network

- Layer 1:
  - $x$  has shape  $(1, \dim(x))$
  - $W_{\text{layer } 1}$  has shape  $(\dim(x), \dim(L1))$
  - $h_{\text{layer } 1}$  has shape  $(1, \dim(L1))$
- Layer 2:
  - $h_{\text{layer } 1}$  has shape  $(1, \dim(L1))$
  - $W_{\text{layer } 2}$  has shape  $(\dim(L1), \dim(L2))$
  - $h_{\text{layer } 2}$  has shape  $(1, \dim(L2))$
- Layer 3:
  - $h_{\text{layer } 2}$  has shape  $(1, \dim(L2))$
  - $W_{\text{layer } 3}$  has shape  $(\dim(L2), \dim(y))$
  - $y$  has shape  $(1, \dim(y))$

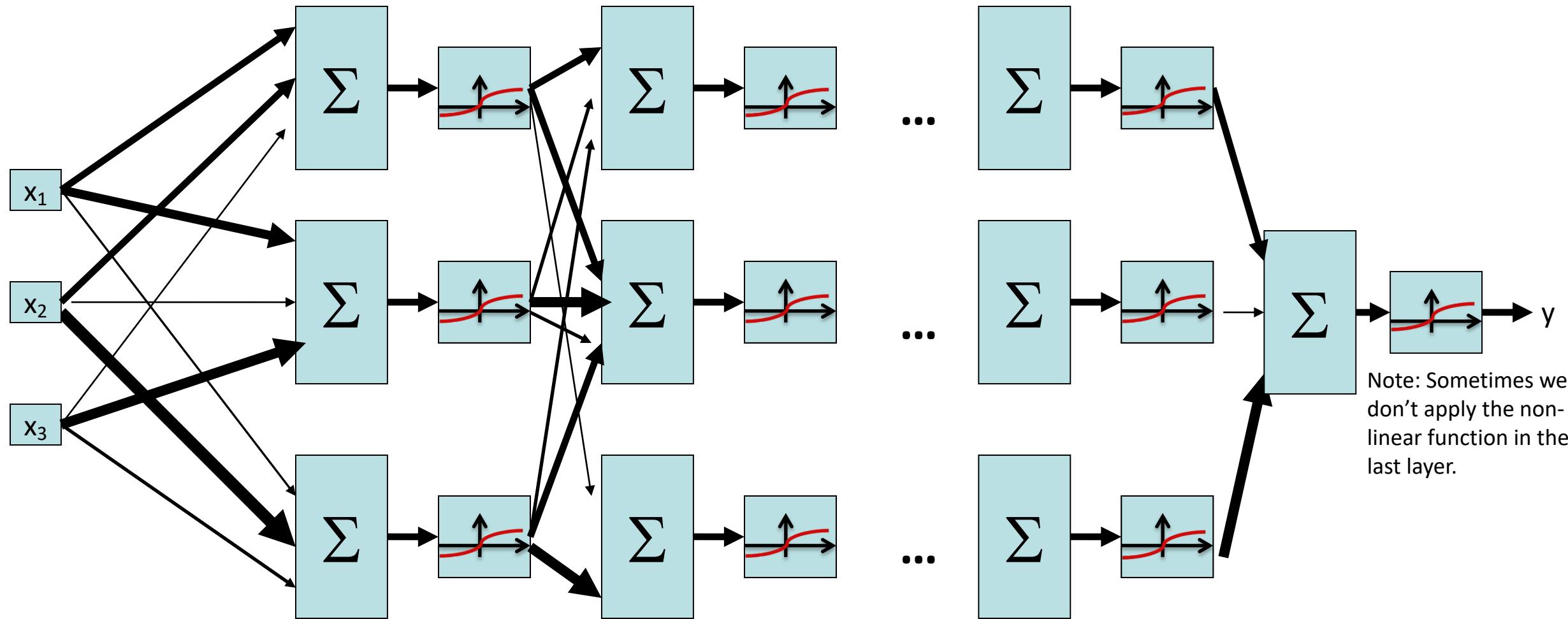


$$\phi(x \times W_{\text{layer } 1}) = h_{\text{layer } 1}$$

$$\phi(h_{\text{layer } 1} \times W_{\text{layer } 2}) = h_{\text{layer } 2}$$

$$\phi(h_{\text{layer } 2} \times W_{\text{layer } 3}) = y$$

# Multi-Layer Neural Network

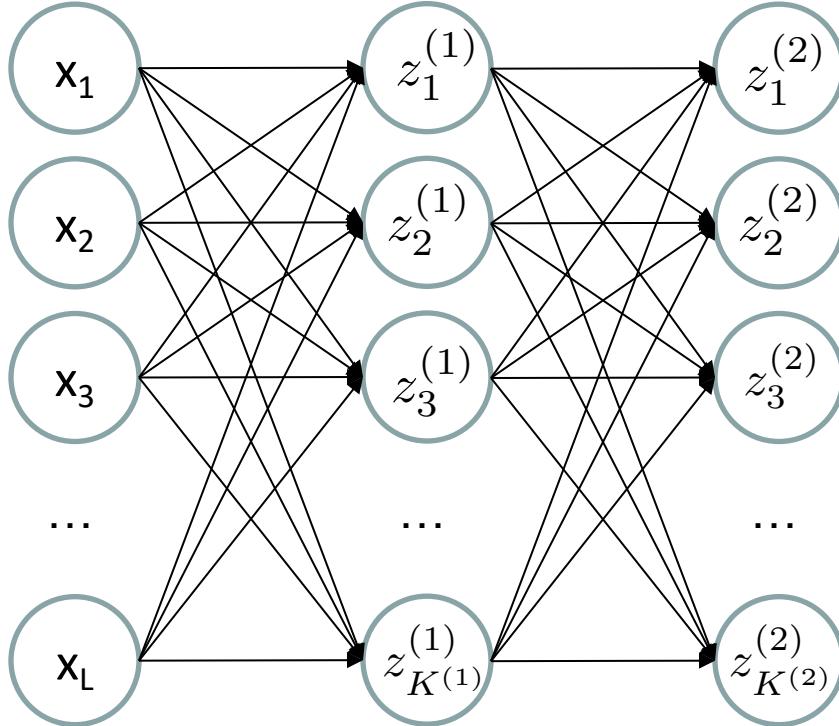


# Multi-Layer Neural Network

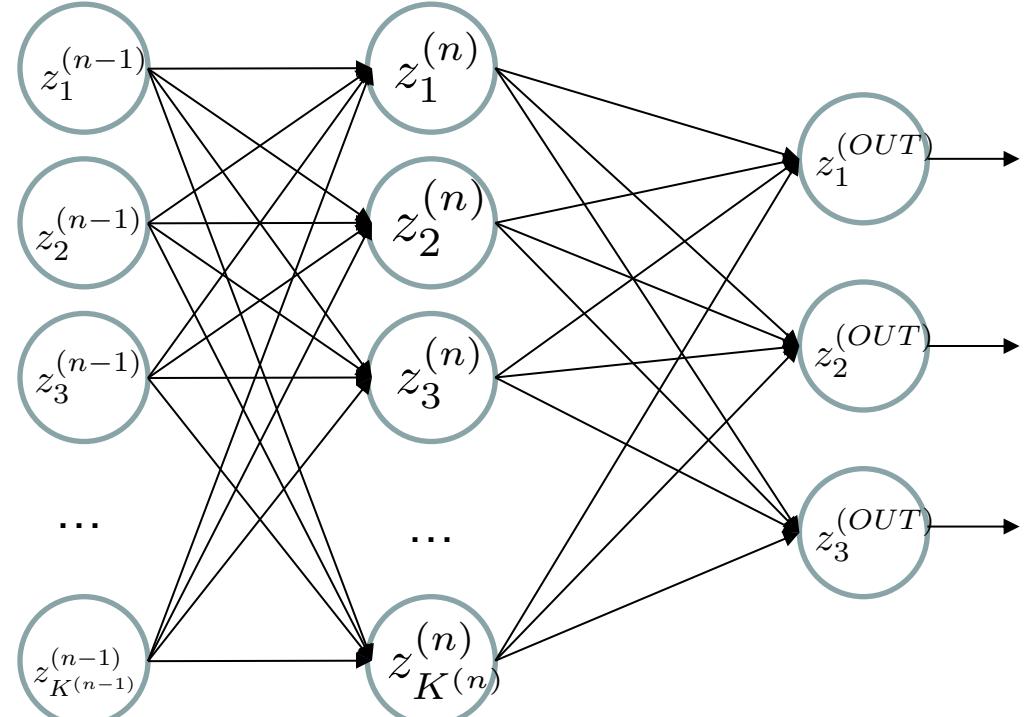
---

- Input to a layer: some  $\dim(x)$ -dimensional input vector
- Output of a layer: some  $\dim(y)$ -dimensional output vector
  - $\dim(y)$  is the number of neurons in the layer (1 output per neuron)
- Process of converting input to output:
  - Multiply the  $(1, \dim(x))$  input vector with a  $(\dim(x), \dim(y))$  weight vector.  
The result has shape  $(1, \dim(y))$ .
  - Apply some non-linear function (e.g. sigmoid) to the result.  
  
The result still has shape  $(1, \dim(y))$ .
- Big idea: Chain layers together
  - The input could come from a previous layer's output
  - The output could be used as the input to the next layer

# Deep Neural Network



...



...

...

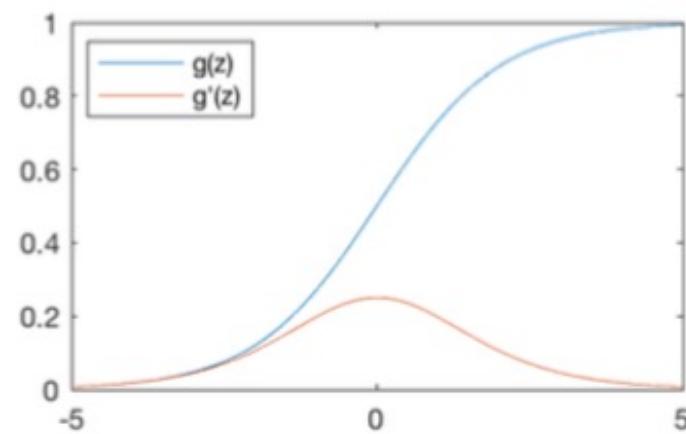
...

$$z_i^{(k)} = g\left(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)}\right)$$

**g = nonlinear activation function**

# Common Activation Functions

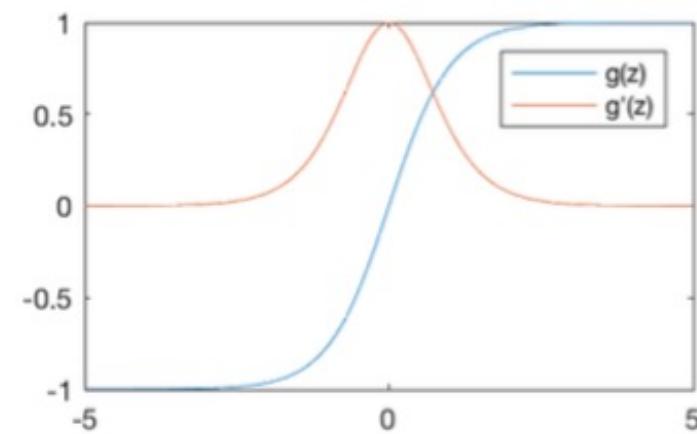
Sigmoid Function



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

$$g'(z) = g(z)(1 - g(z))$$

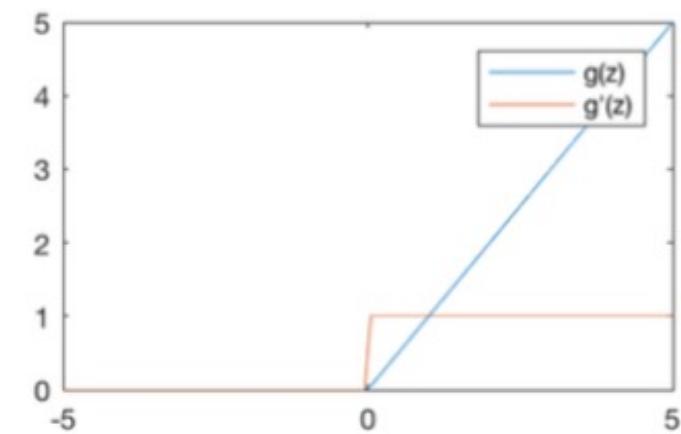
Hyperbolic Tangent



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

$$g'(z) = 1 - g(z)^2$$

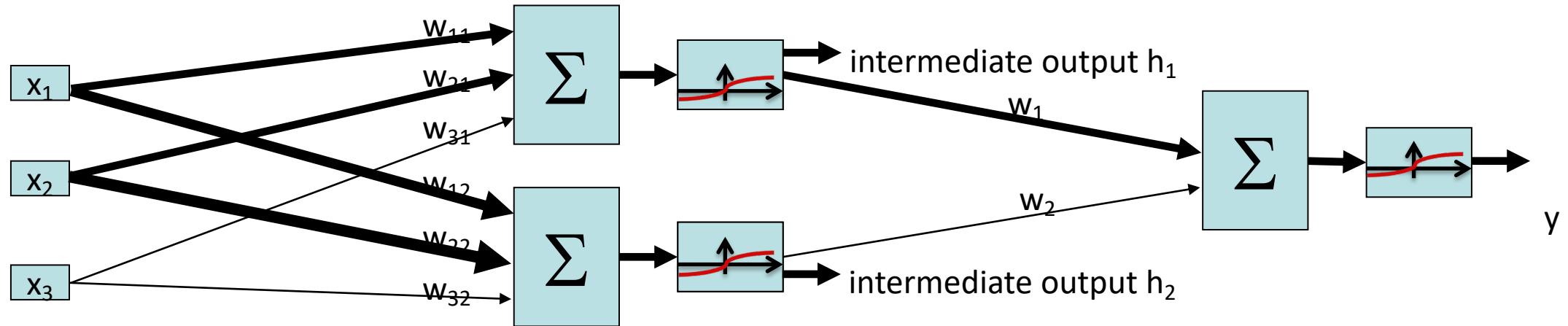
Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

# Important to use non-linear activation functions



- With non-linear activation  $\phi$  for intermediate output:

$$y = \phi(w_1 h_1 + w_2 h_2)$$

$$= \phi(w_1 \phi(w_{11} x_1 + w_{21} x_2 + w_{31} x_3) + w_2 \phi(w_{12} x_1 + w_{22} x_2 + w_{32} x_3))$$

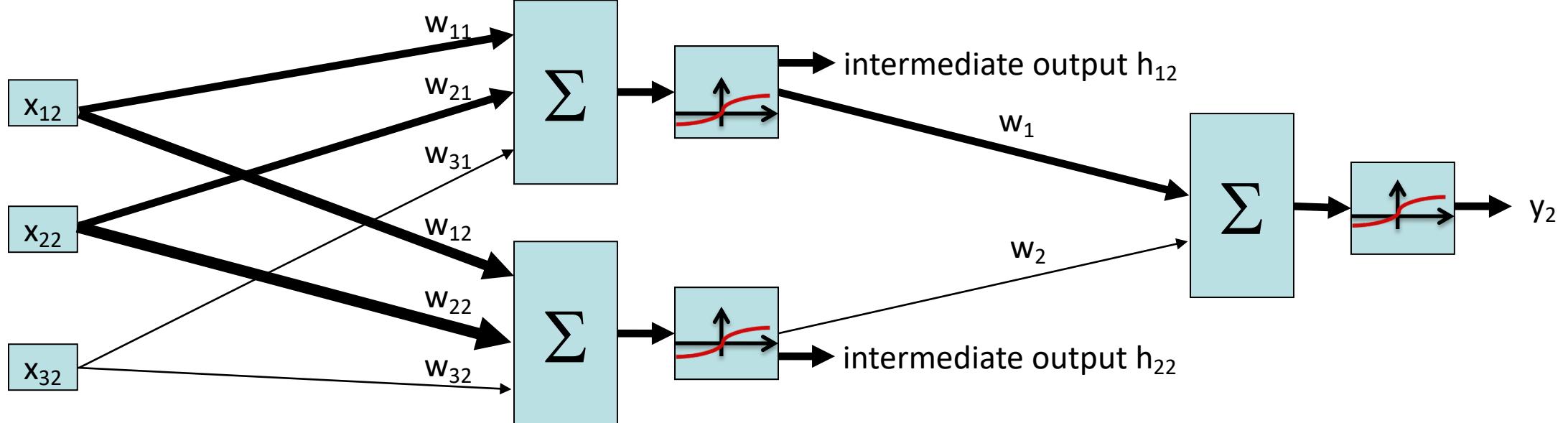
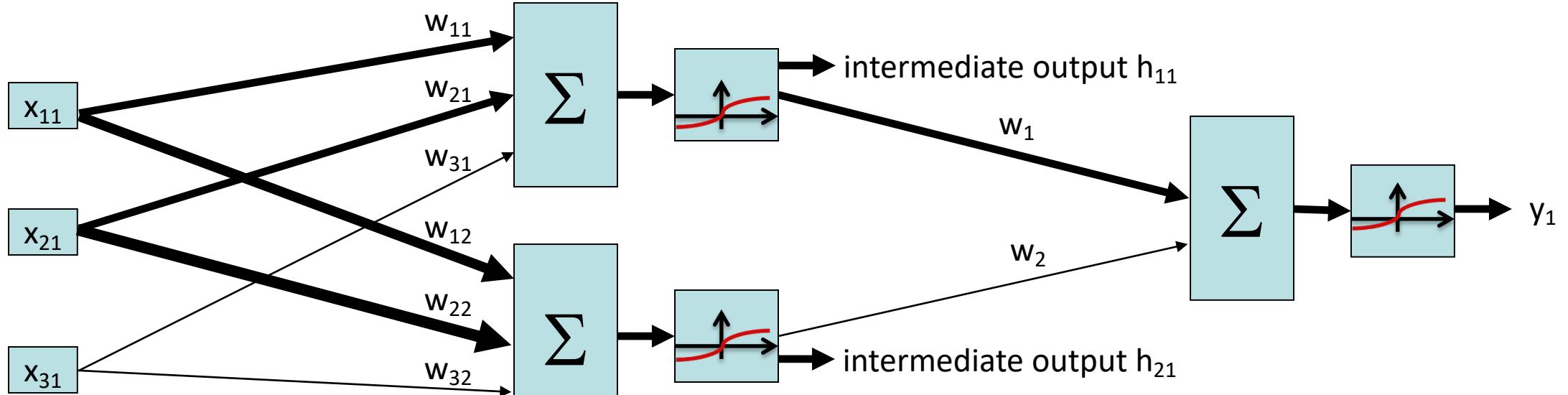
- Without intermediate activations  $\phi$ :

$$y = \phi(w_1(w_{11} x_1 + w_{21} x_2 + w_{31} x_3) + w_2(w_{12} x_1 + w_{22} x_2 + w_{32} x_3))$$

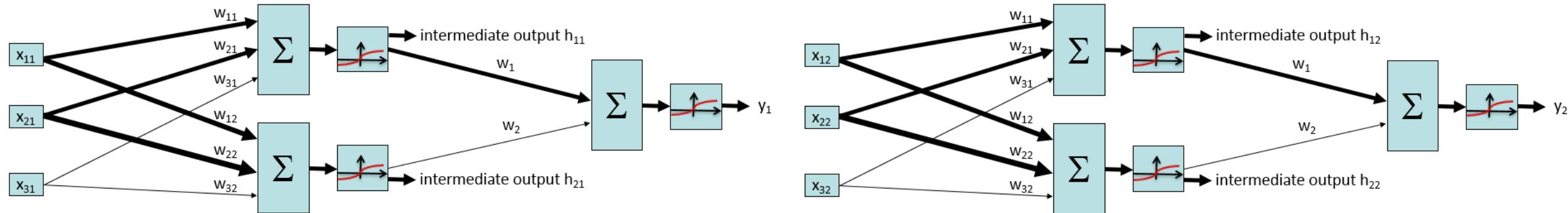
$$= \phi((w_1 w_{11} + w_2 w_{12})x_1 + (w_1 w_{21} + w_2 w_{22})x_2 + (w_1 w_{31} + w_2 w_{32})x_3)$$

$$= \phi(ax_1 + bx_2 + cx_3) \leftarrow \text{same as not including a hidden layer!}$$

# Batch Sizes



# Batch Sizes



$$\begin{aligned} y_1 &= \phi(w_1 h_{11} + w_2 h_{12}) \\ &= \phi(w_1 \phi(w_{11} x_{11} + w_{21} x_{12} + w_{31} x_{13}) + w_2 \phi(w_{12} x_{11} + w_{22} x_{12} + w_{32} x_{13})) \\ y_2 &= \phi(w_1 h_{21} + w_2 h_{22}) \\ &= \phi(w_1 \phi(w_{11} x_{21} + w_{21} x_{22} + w_{31} x_{23}) + w_2 \phi(w_{12} x_{21} + w_{22} x_{22} + w_{32} x_{23})) \end{aligned}$$

We're not changing the architecture; we're just running the 2-neuron, 2-layer network twice to classify 2 inputs.

# Batch Sizes

$$\begin{aligned}y_1 &= \phi(w_1 h_{11} + w_2 h_{12}) \\&= \phi(w_1 \phi(w_{11}x_{11} + w_{21}x_{12} + w_{31}x_{13}) + w_2 \phi(w_{12}x_{11} + w_{22}x_{12} + w_{32}x_{13})) \\y_2 &= \phi(w_1 h_{21} + w_2 h_{22}) \\&= \phi(w_1 \phi(w_{11}x_{21} + w_{21}x_{22} + w_{31}x_{23}) + w_2 \phi(w_{12}x_{21} + w_{22}x_{22} + w_{32}x_{23}))\end{aligned}$$

Rewriting in matrix form:

$$\begin{aligned}&\phi \left( \begin{bmatrix} x_{11} & x_{21} & x_{31} \\ x_{12} & x_{22} & x_{32} \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \right) \\&= \phi \left( \begin{bmatrix} w_{11}x_{11} + w_{21}x_{21} + w_{31}x_{31} & w_{12}x_{11} + w_{22}x_{21} + w_{32}x_{31} \\ w_{11}x_{12} + w_{21}x_{22} + w_{31}x_{32} & w_{12}x_{12} + w_{22}x_{22} + w_{32}x_{32} \end{bmatrix} \right) \\&= \begin{bmatrix} h_{11} & h_{21} \\ h_{12} & h_{22} \end{bmatrix}\end{aligned}$$

$$\phi \left( \begin{bmatrix} h_{11} & h_{21} \\ h_{12} & h_{22} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \right) = \phi \left( \begin{bmatrix} w_1 h_{11} + w_2 h_{21} \\ w_1 h_{12} + w_2 h_{22} \end{bmatrix} \right) = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

# Batch Sizes

$$\phi(x \times W_{\text{layer 1}}) = h$$

Shape (**batch**,  $\text{dim}(x)$ ).  
Input feature vector.

Shape ( $\text{dim}(x)$ ,  $n$ ).  
Weights to be learned

Shape (**batch**,  $n$ ).  
Outputs of layer 1,  
inputs to layer 2.

$$\phi(h \times W_{\text{layer 2}}) = y$$

Shape (**batch**,  $n$ ).  
Outputs of layer 1,  
inputs to layer 2.

Shape ( $n$ ,  $\text{dim}(y)$ ).  
Weights to be learned.

Shape (**batch**,  $\text{dim}(y)$ ).  
Output of network.

Big idea: We can “stack” inputs together to classify multiple inputs at once. The result is multiple outputs “stacked” together.

# Multi-Layer Network, with Batches

---

- Input to a layer: *batch* different  $\dim(x)$ -dimensional input vectors
- Output of a layer: *batch* different  $\dim(y)$ -dimensional output vectors
  - $\dim(y)$  is the number of neurons in the layer (1 output per neuron)
- Process of converting input to output:
  - Multiply the (*batch*,  $\dim(x)$ ) input matrix with a ( $\dim(x)$ ,  $\dim(y)$ ) weight vector.  
The result has shape (*batch*,  $\dim(y)$ ).
  - Apply some non-linear function (e.g. sigmoid) to the result.  
The result still has shape (*batch*,  $\dim(y)$ ).
- Big idea: Stack inputs/outputs to batch them
  - The multiplication by weights and non-linear function will be applied to each row (data point in the batch) separately.

# Quiz: Sizes of neural networks

2 layer - 2 neuron

$$\phi(\begin{pmatrix} 4 \times 5 \\ x \end{pmatrix} \times \begin{pmatrix} 2 \times 2 \\ W_{layer\ 1} \end{pmatrix}) = \begin{pmatrix} 4 \times 2 \\ h \end{pmatrix}$$

$$\phi(\begin{pmatrix} 4 \times 2 \\ h \end{pmatrix} \times \begin{pmatrix} 2 \times 1 \\ W_{layer\ 2} \end{pmatrix}) = \begin{pmatrix} 4 \times 1 \\ y \end{pmatrix}$$

①  $\boxed{4} \times \boxed{2} = \boxed{2}$

We have a neural network with the matrices drawn.

②  $\boxed{2} \times \boxed{1} = \boxed{1}$

1. How many layers are in the network? 2
2. How many input dimensions  $\dim(x)$ ? 3
3. How many hidden neurons  $n$ ? 2
4. How many output dimensions  $\dim(y)$ ? 1
5. What is the batch size? 4

# Quiz: Sizes of neural networks

$$\phi(\begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline & & \\ \hline & & \\ \hline\end{array}) \times \begin{array}{|c|c|}\hline & \\ \hline & \\ \hline\end{array} = \begin{array}{|c|c|}\hline & \\ \hline & \\ \hline & \\ \hline & \\ \hline\end{array}$$

$x$                                    $W_{layer\ 1}$                                    $h$

$$\phi(\begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline & & \\ \hline & & \\ \hline\end{array}) \times \begin{array}{|c|}\hline & \\ \hline & \\ \hline\end{array} = \begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline & & \\ \hline & & \\ \hline\end{array}$$

$h$                                        $W_{layer\ 2}$                                $y$

We have a neural network with the matrices drawn.

1. How many layers are in the network?  
**2**
2. How many input dimensions  $\dim(x)$ ?  
**3**
3. How many hidden neurons  $n$ ?  
**2**
4. How many output dimensions  $\dim(y)$ ?  
**1**
5. What is the batch size?  
**4**

# Next Time: Training Neural Networks & Applications

---

