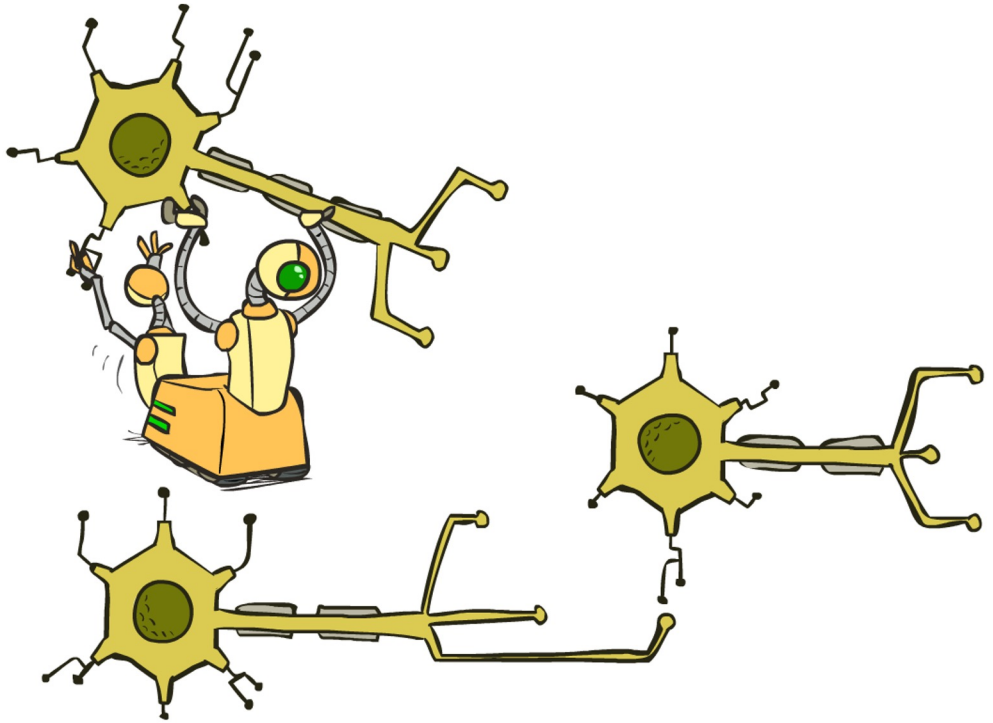
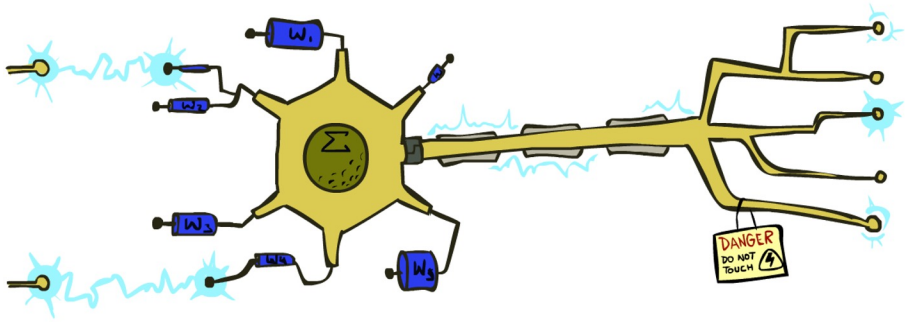
CS 188: Artificial Intelligence

Optimization and Neural Networks



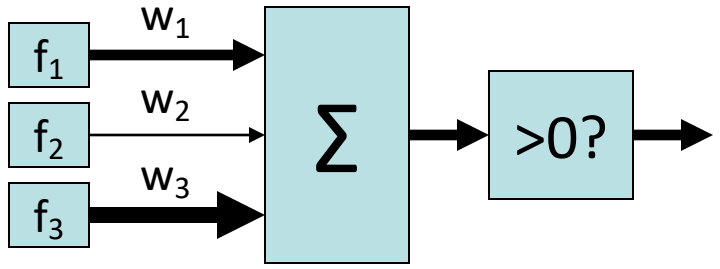
Reminder: Linear Classifiers

- 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?

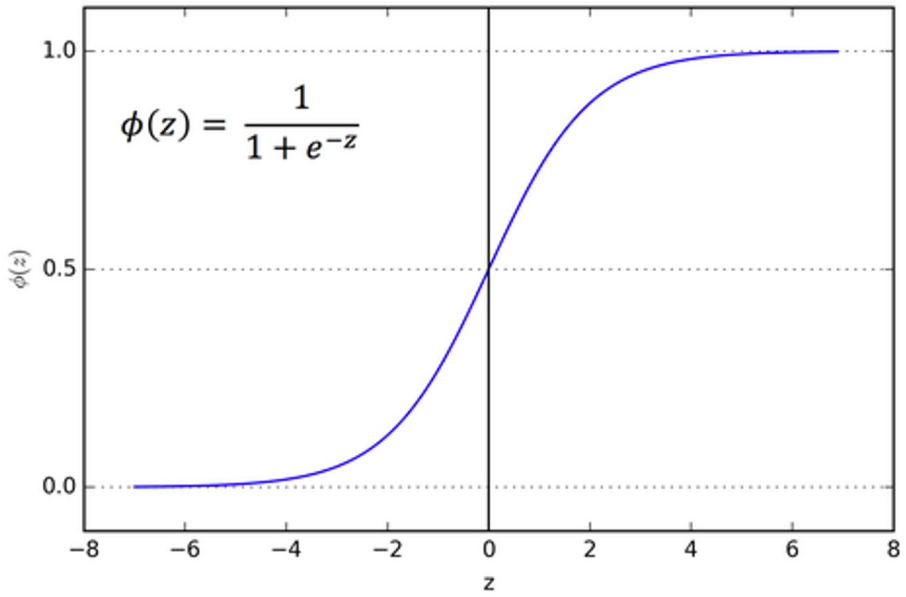
Activation: $z = w \cdot f(x)$

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

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

Sigmoid function

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



Best w?

Maximum likelihood estimation:

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

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

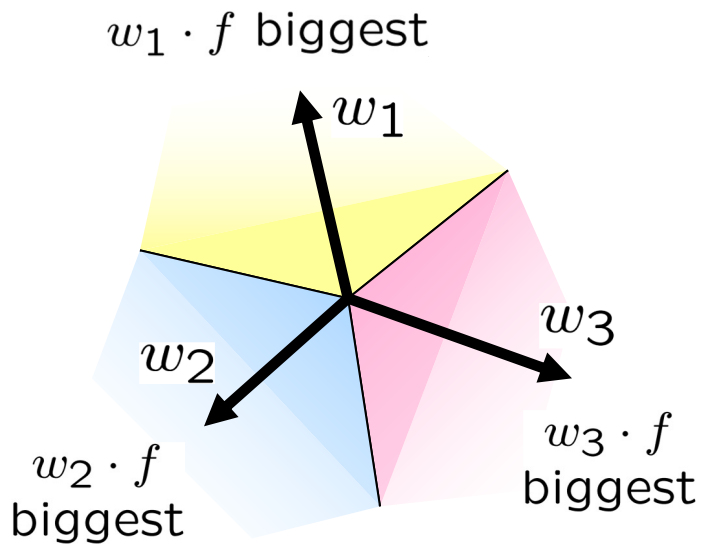
Multiclass Logistic Regression

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?

$$z_1, z_2, z_3 \rightarrow \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

original activations

softmax activations

Best w?

Maximum likelihood estimation:

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

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:

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

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

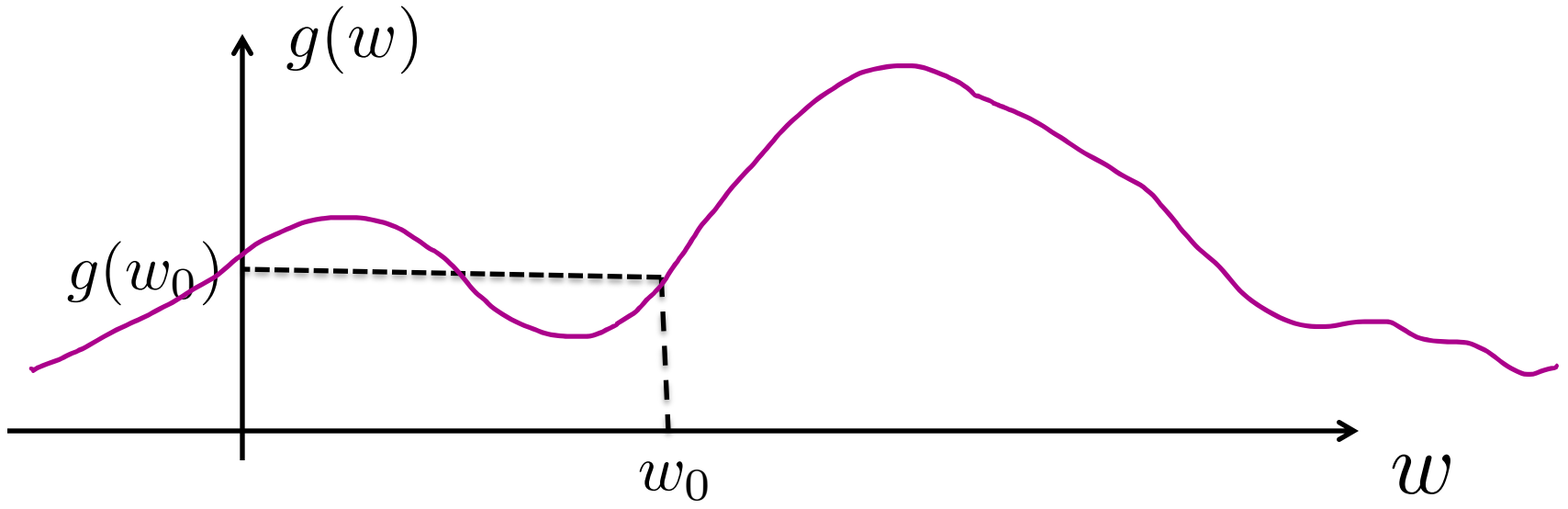
- What is the gradient of the function $g(x, y) = x^2 y$?
  - Recall: Gradient is a vector of partial derivatives with respect to each variable

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

- 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

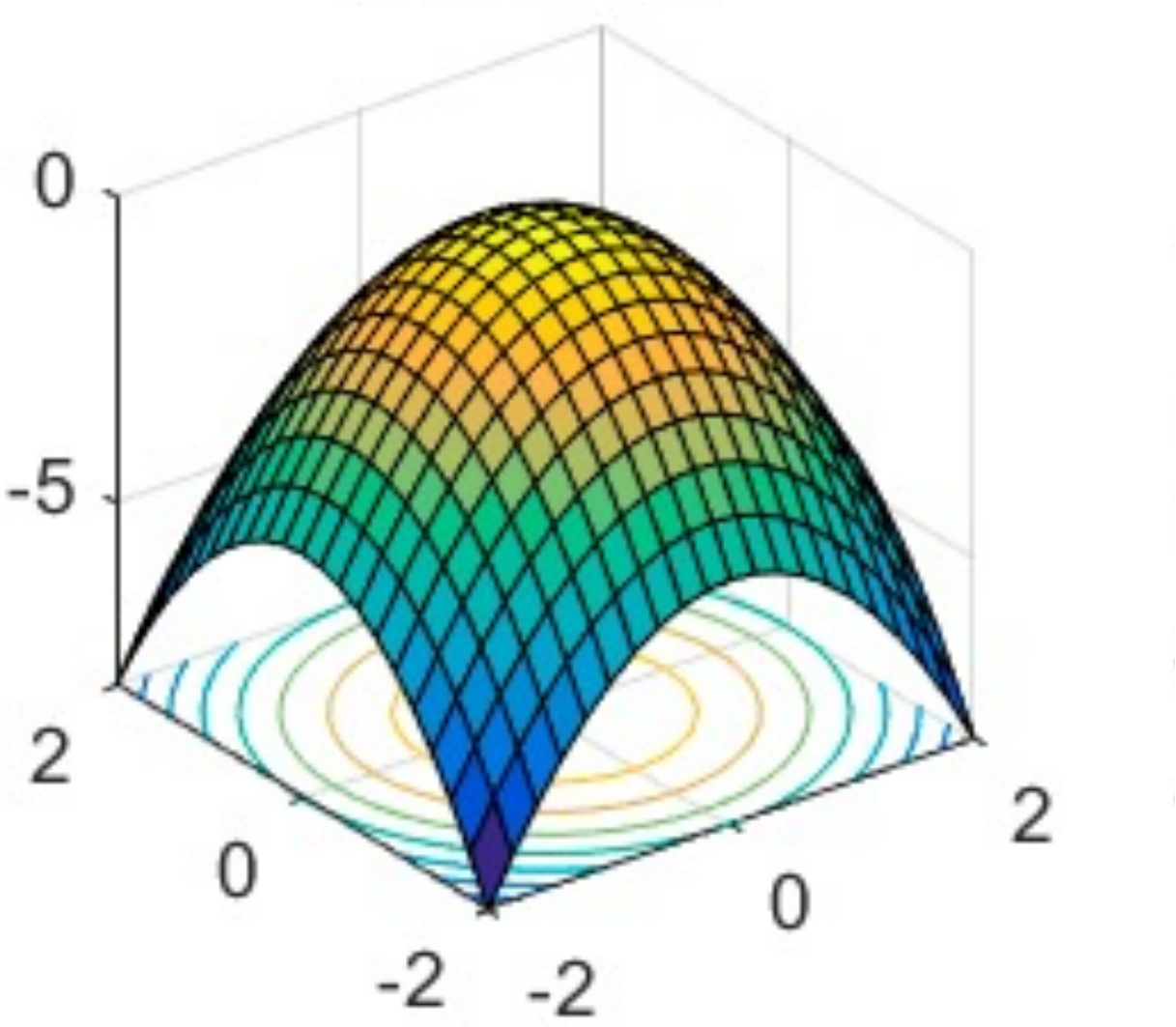


- Could evaluate $g(w_0 + h)$ and $g(w_0 - h)$
  - Then step in best direction
- Or, evaluate derivative:

$$\frac{\partial g(w_0)}{\partial w} = \lim_{h \to 0} \frac{g(w_0 + h) - g(w_0 - h)}{2h}$$

  - Tells which direction to step into

2-D Optimization



Source: offconvex.org

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

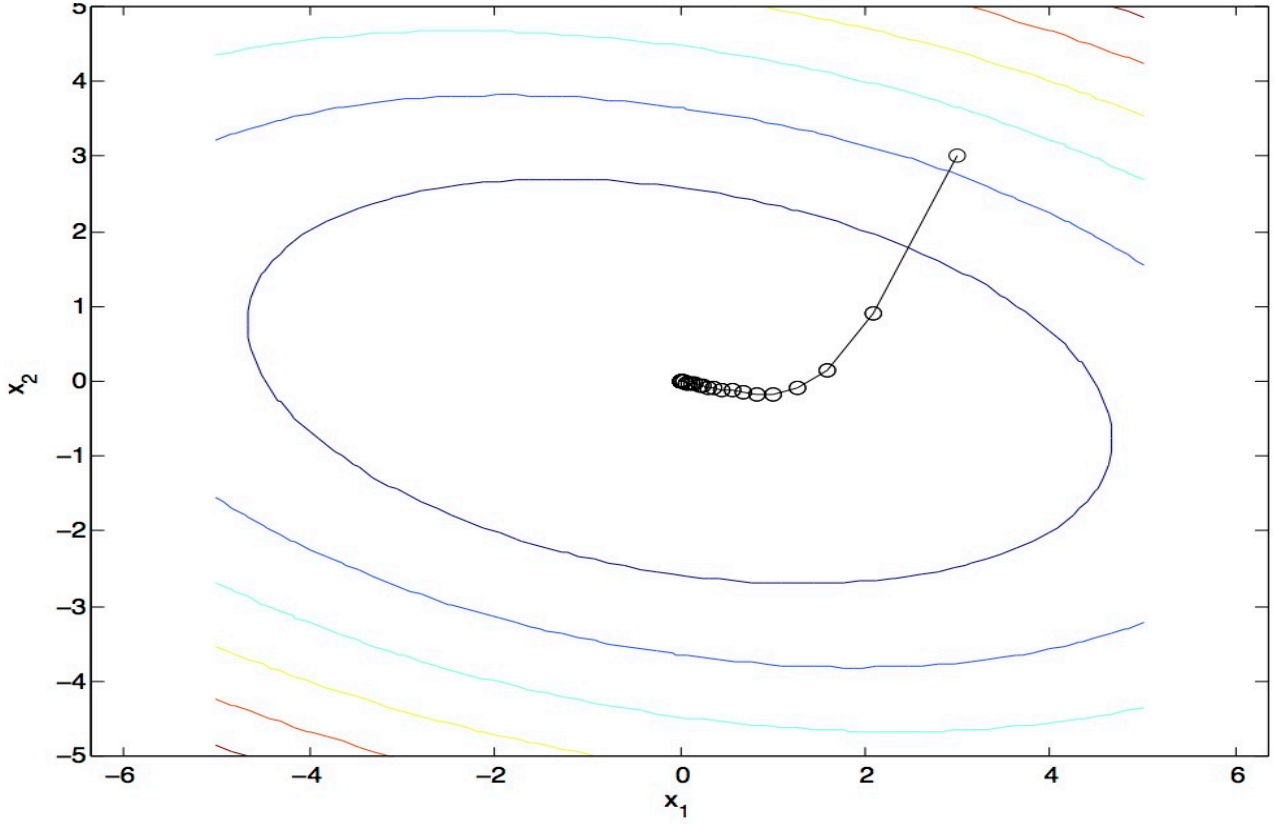


Figure source: Mathworks

Gradient Ascent

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

Not guaranteed to find global maximum:

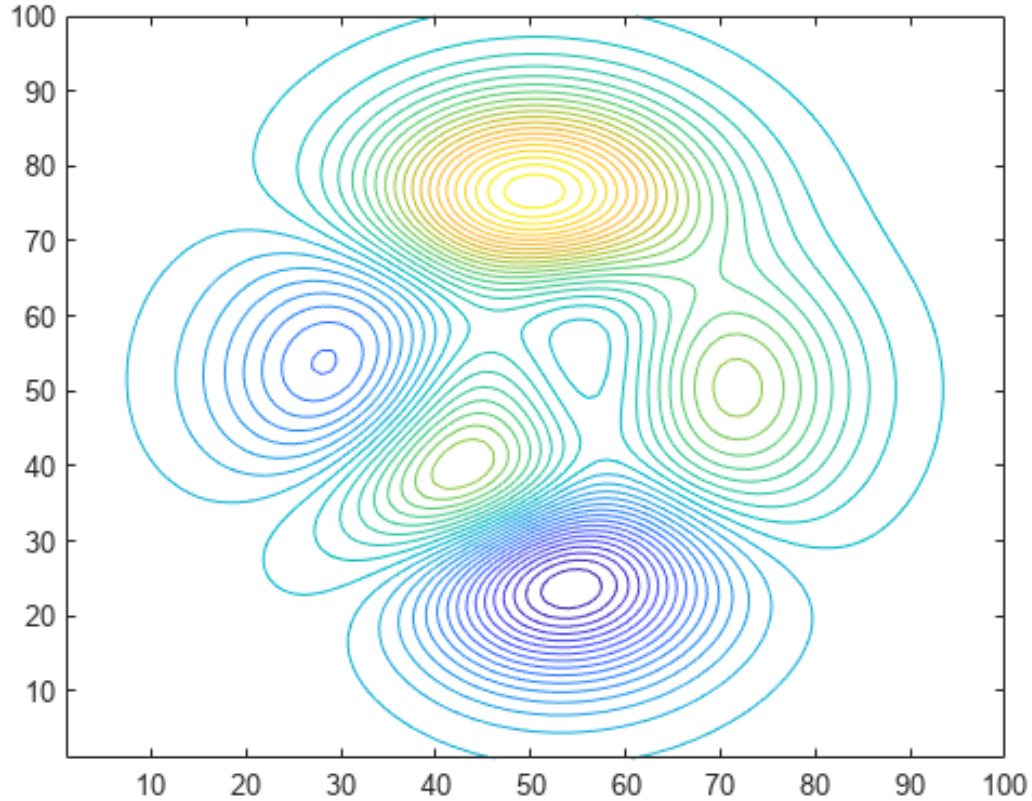
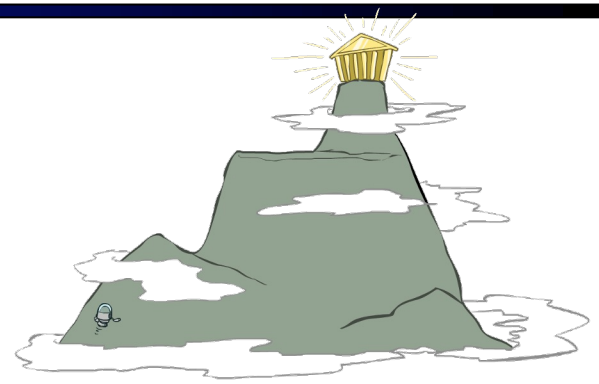


Figure source: Mathworks

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

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

Optimization Procedure: Gradient Ascent

Init w

for iter = 1, 2, …

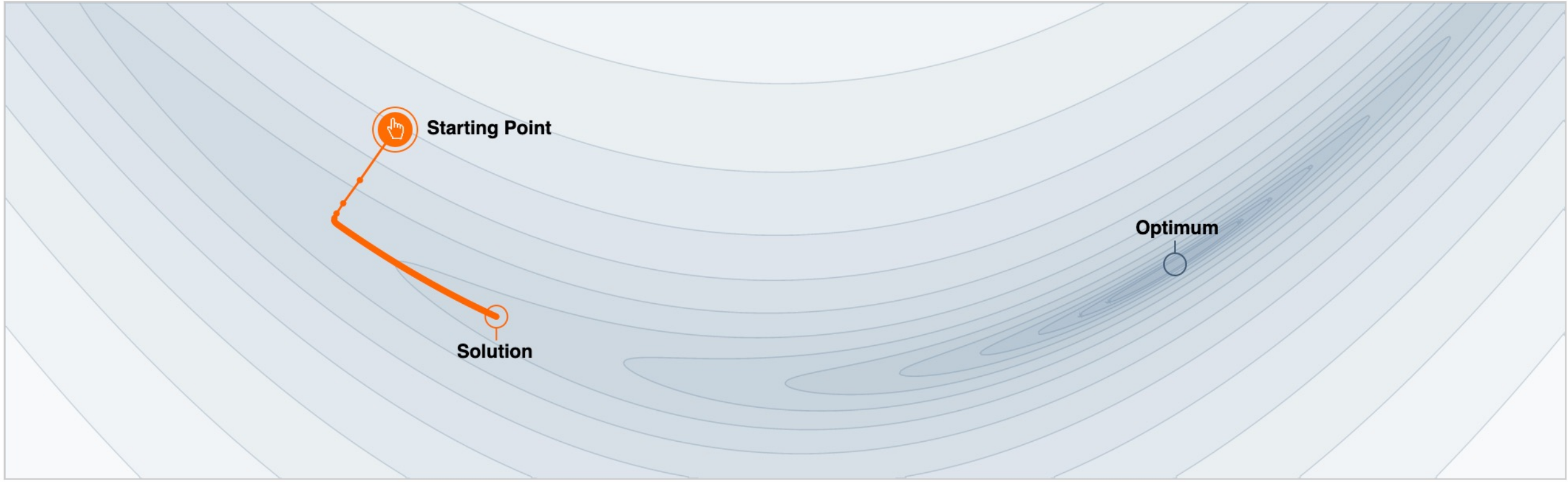
$$w \leftarrow w + \alpha \cdot \nabla g(w)$$

- α: 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 α is a hyperparameter

Example: α=0.001 (too small)

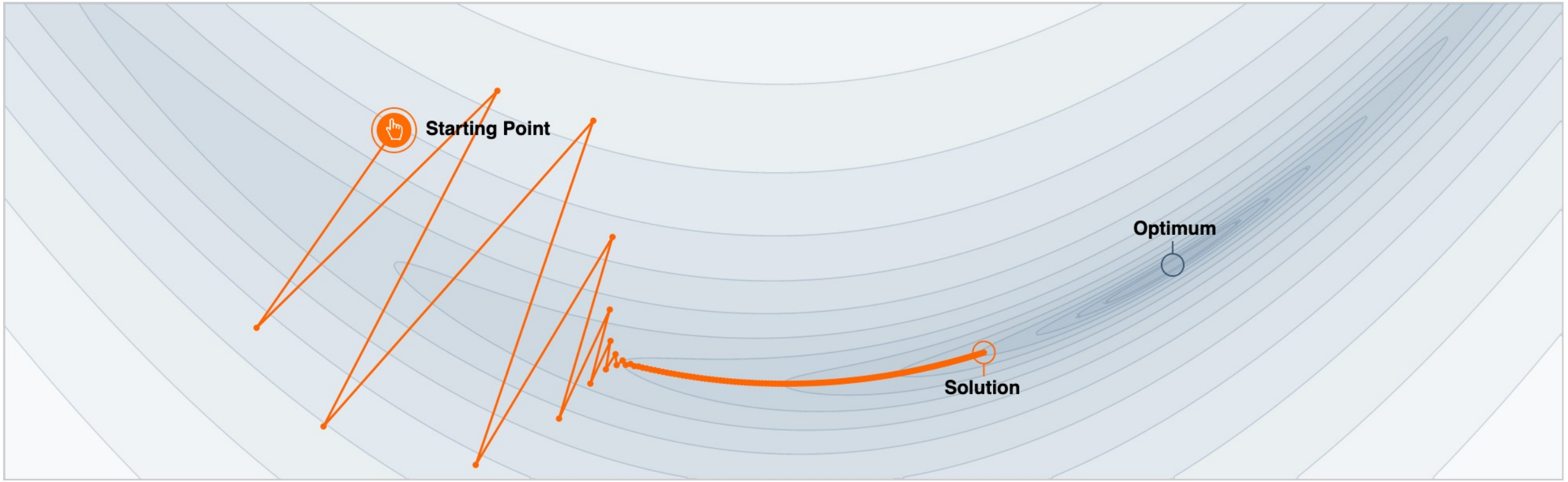


Source: https://distill.pub/2017/momentum/

Learning Rate

Choice of step size α is a hyperparameter

Example: α=0.004 (too large)



Source: https://distill.pub/2017/momentum/

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, …
```

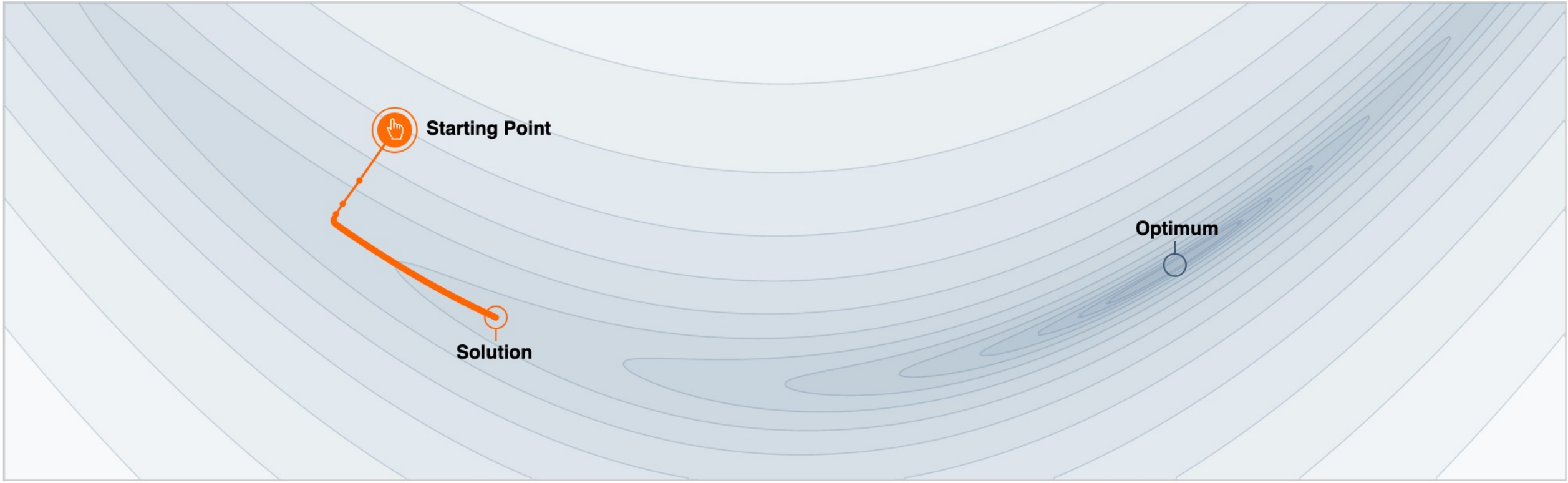
$$z \leftarrow \beta \cdot z + \nabla g(w)$$

$$w \leftarrow w + \alpha \cdot z$$

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

Gradient Ascent with Momentum*

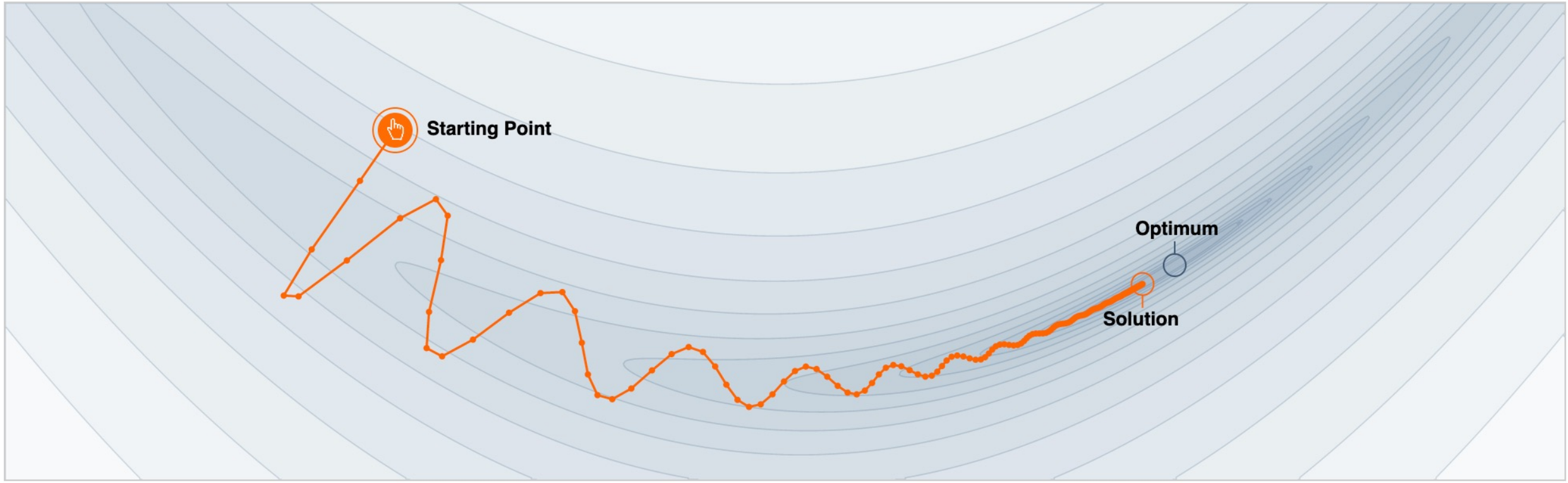
Example: α=0.001 and β=0.0



Source: https://distill.pub/2017/momentum/

Gradient Ascent with Momentum*

Example: α=0.001 and β=0.9



Source: https://distill.pub/2017/momentum/

Batch Gradient Ascent on the Log Likelihood Objective

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

- 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

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

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

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

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

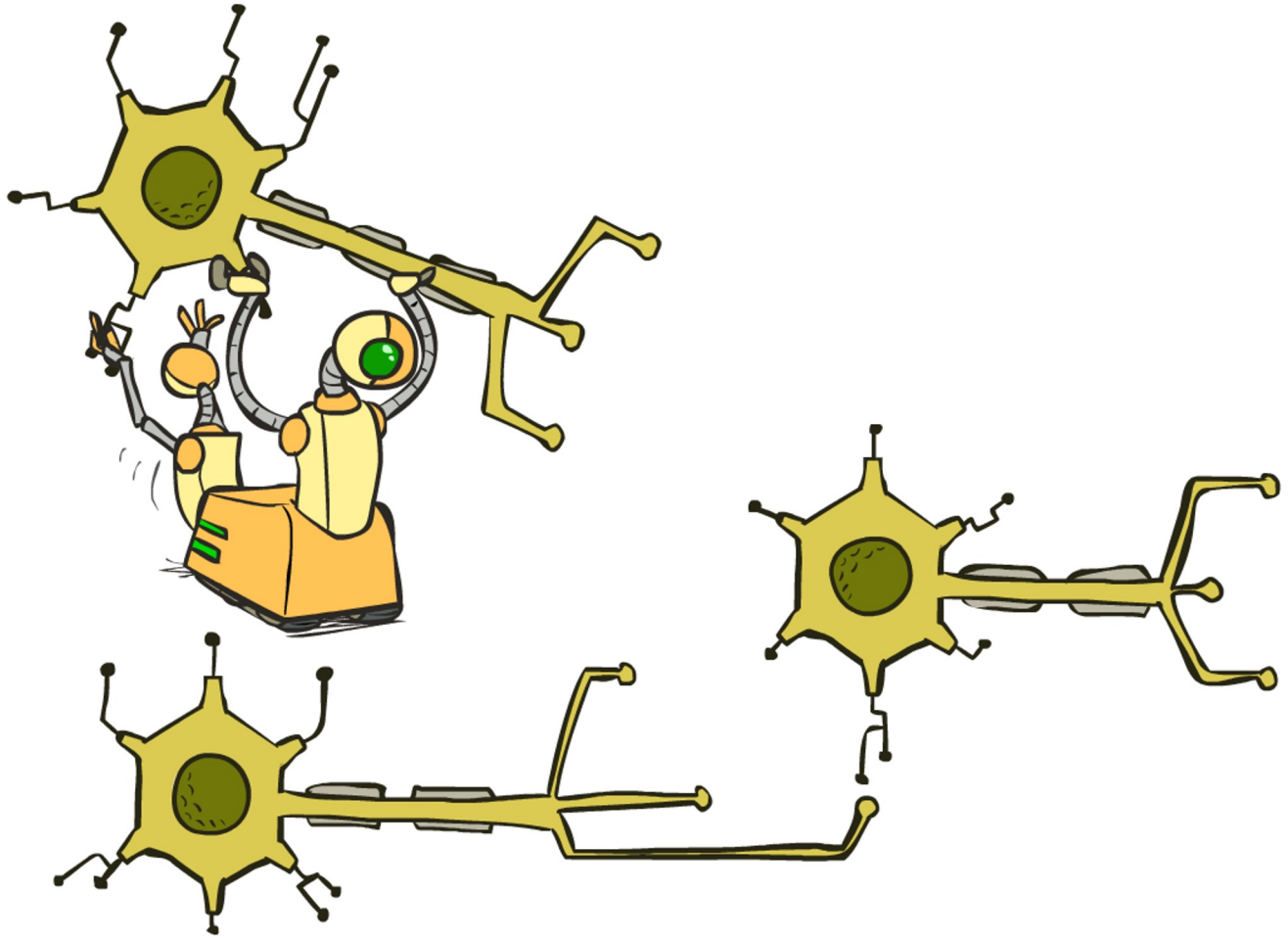
- init w
- for iter = 1, 2, …
  - pick random subset of training examples J

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

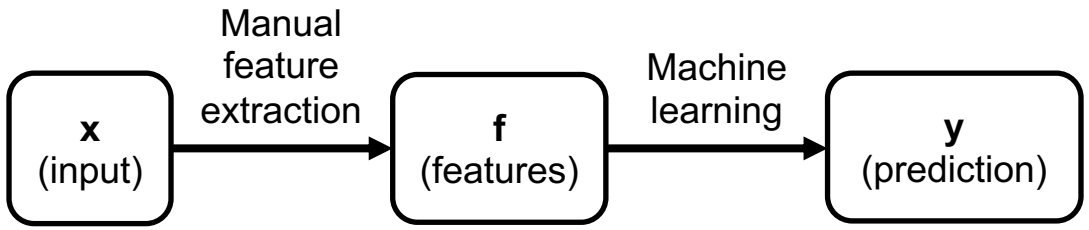
How about computing all the derivatives?

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

Neural Networks

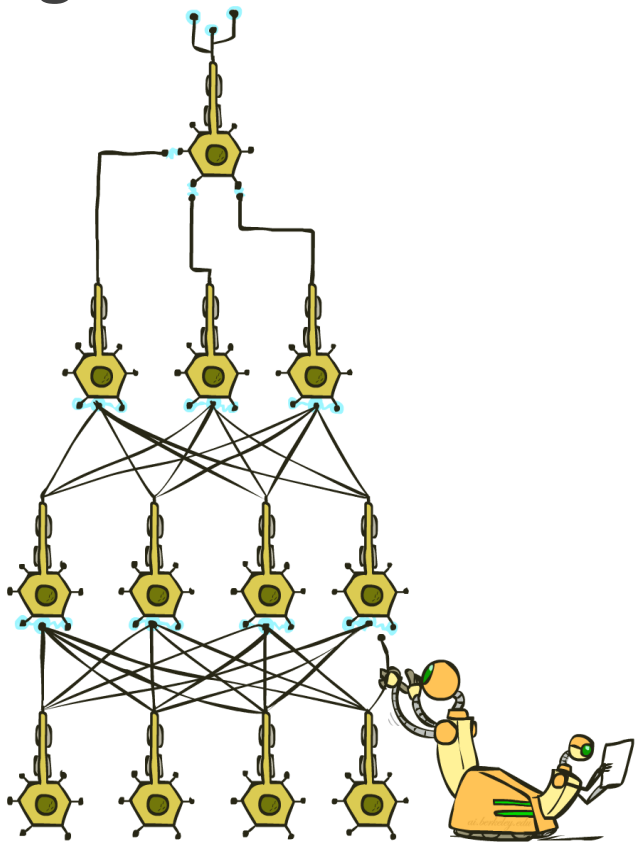


Manual Feature Design vs. Deep Learning

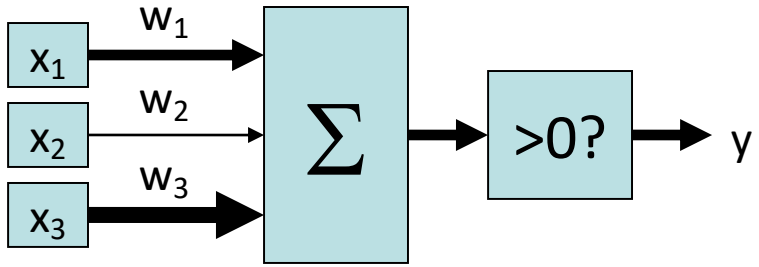


- Manual feature design requires:
  - Domain-specific expertise
  - Domain-specific effort

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

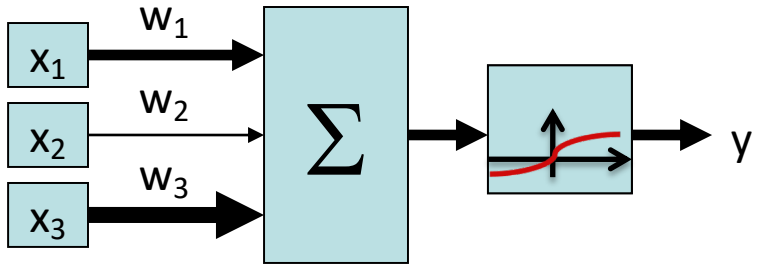


Review: Perceptron



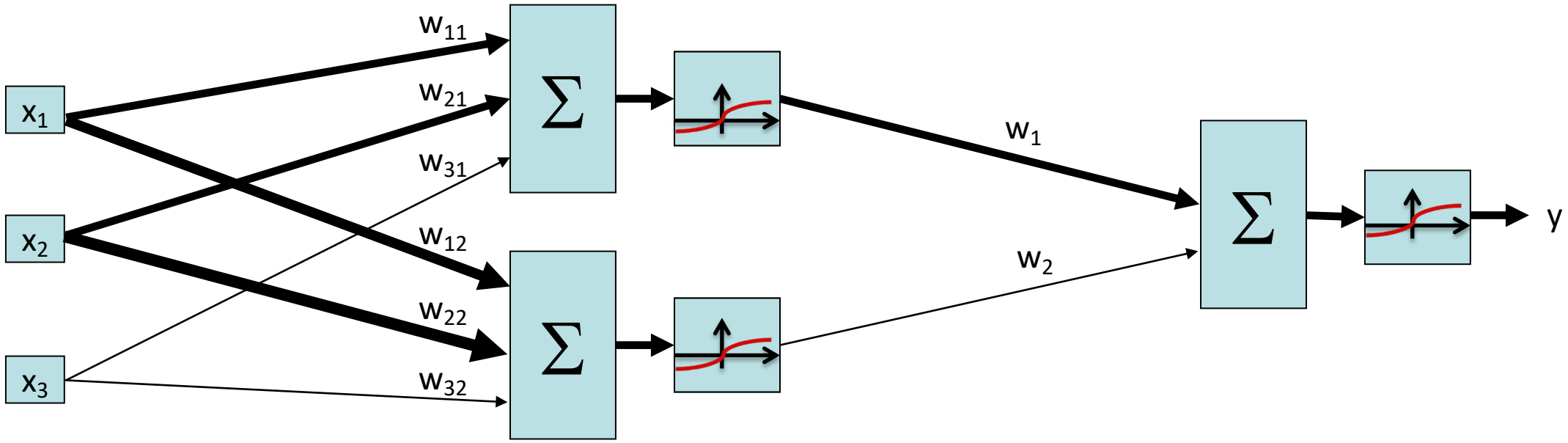
$$y = \begin{cases} 1 & w_1 x_1 + w_2 x_2 + w_3 x_3 > 0 \\ 0 & \text{otherwise} \end{cases}$$

Review: Perceptron with Sigmoid Activation

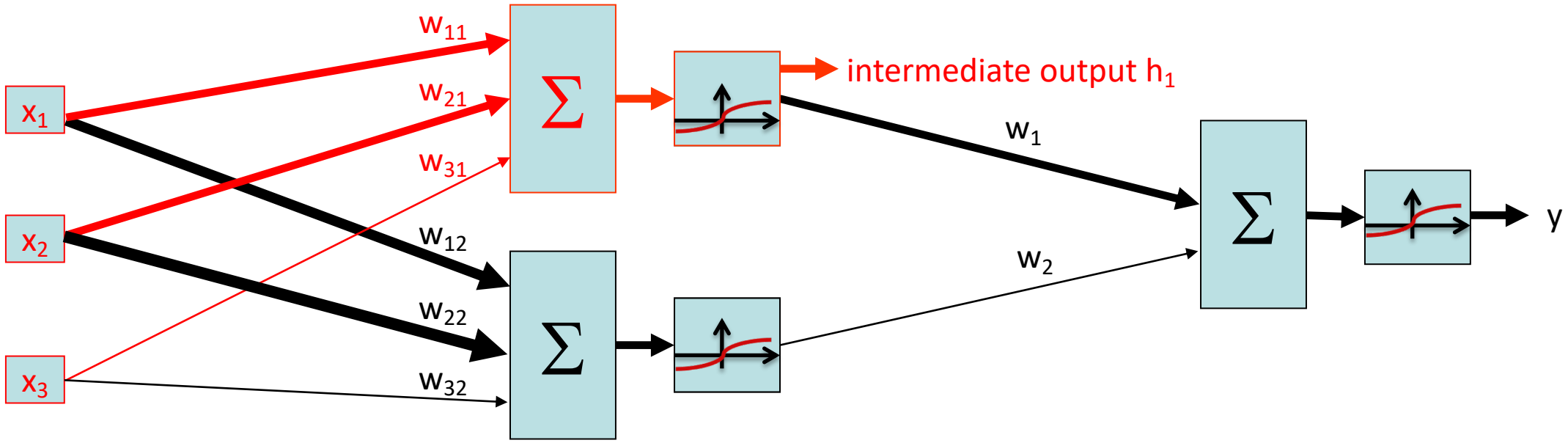


$$y = \phi(w_1 x_1 + w_2 x_2 + w_3 x_3)$$
$$= \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2 + w_3 x_3)}}$$

2-Layer, 2-Neuron Neural Network



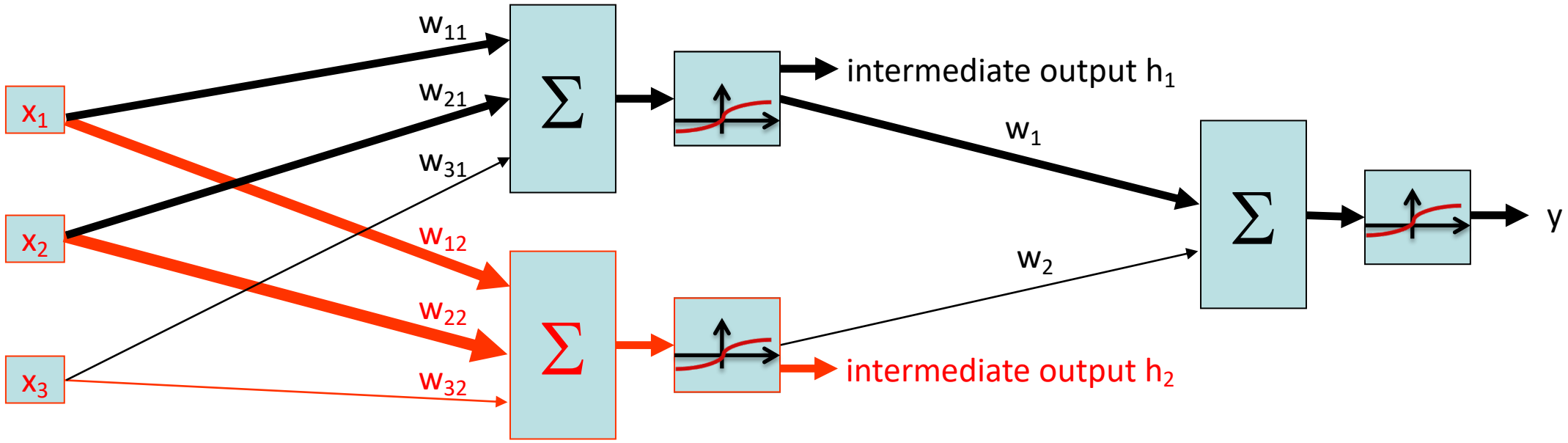
2-Layer, 2-Neuron Neural Network



$$\text{intermediate output } h_1 = \phi(w_{11}x_1 + w_{21}x_2 + w_{31}x_3)$$

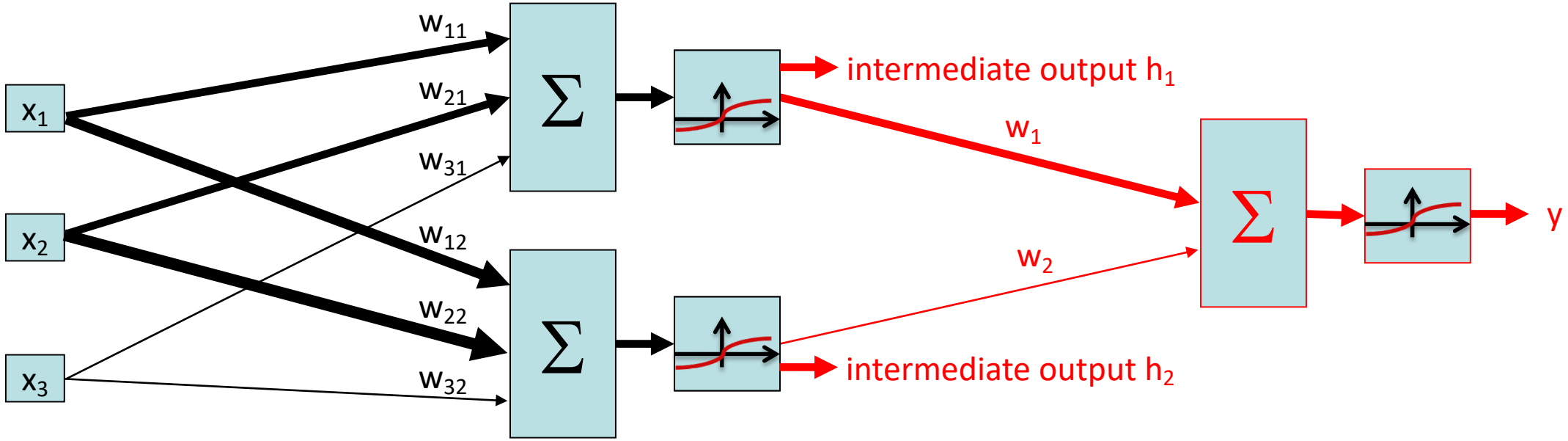
$$= \frac{1}{1 + e^{-(w_{11}x_1 + w_{21}x_2 + w_{31}x_3)}}$$

2-Layer, 2-Neuron Neural Network



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

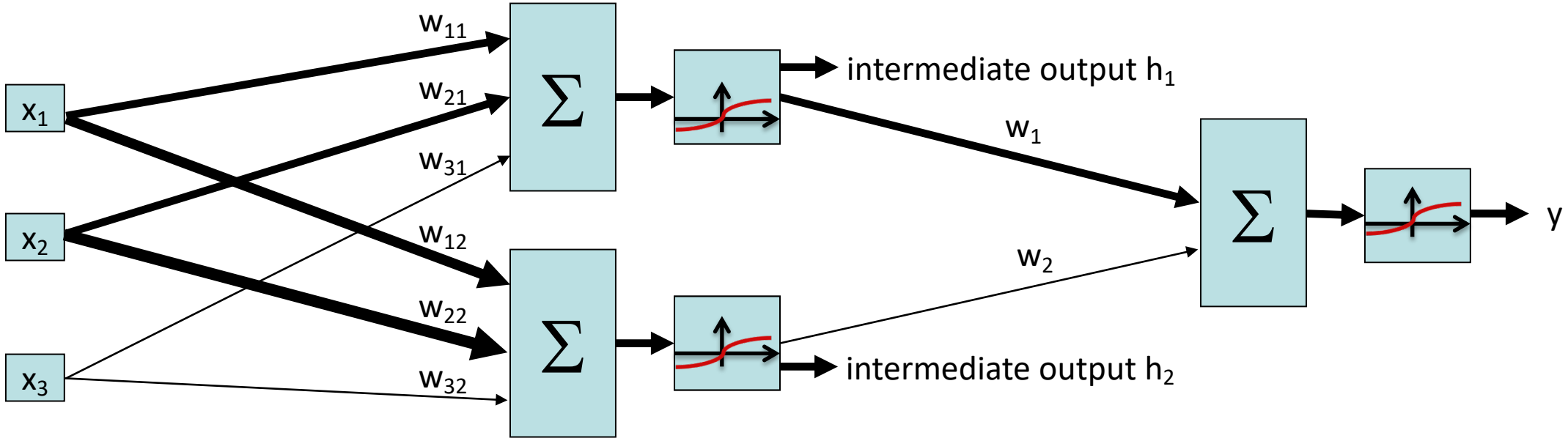
2-Layer, 2-Neuron Neural Network



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

$$= \frac{1}{1 + e^{-(w_1 h_1 + w_2 h_2)}}$$

2-Layer, 2-Neuron Neural Network



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

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:

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

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

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

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

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.

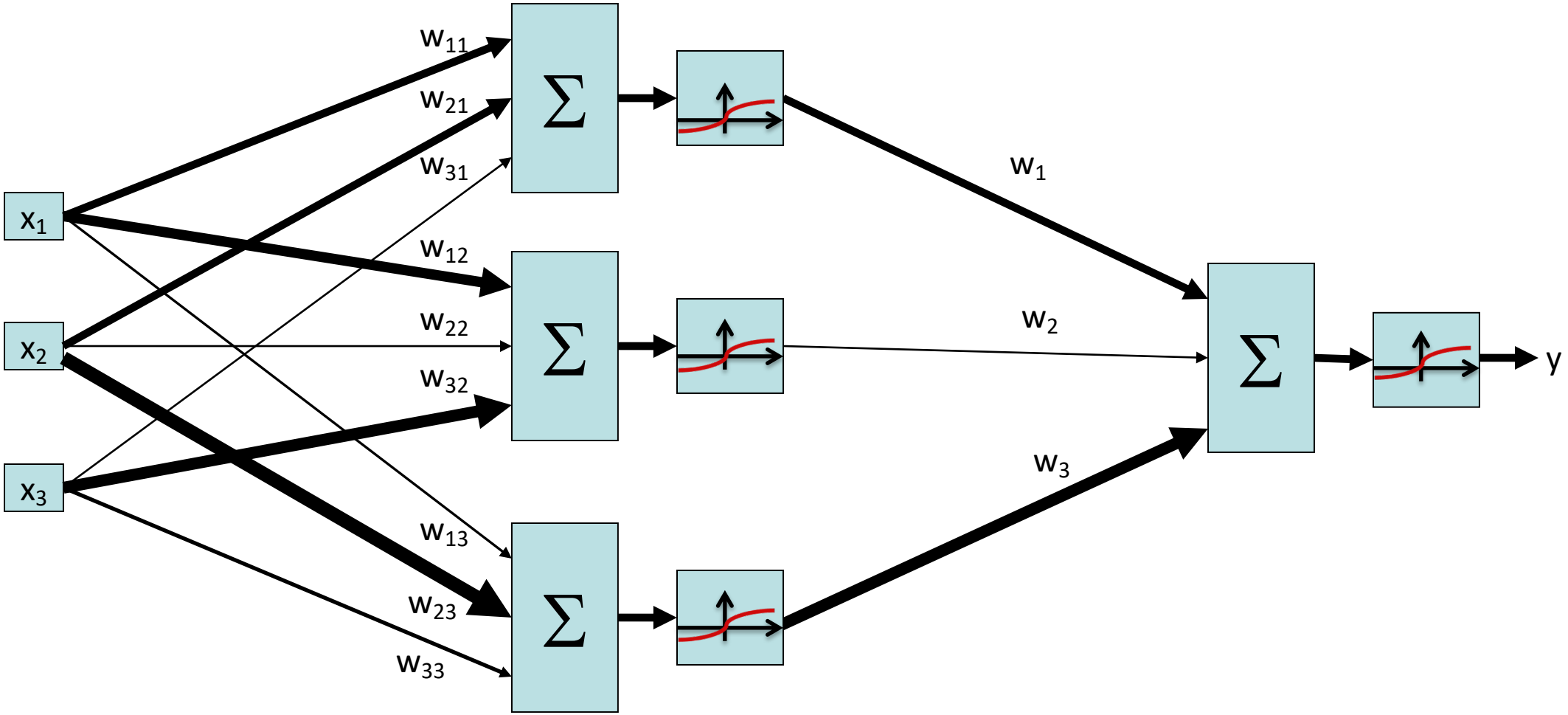
$$\phi(h \times W_{\text{layer } 2}) = y$$

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

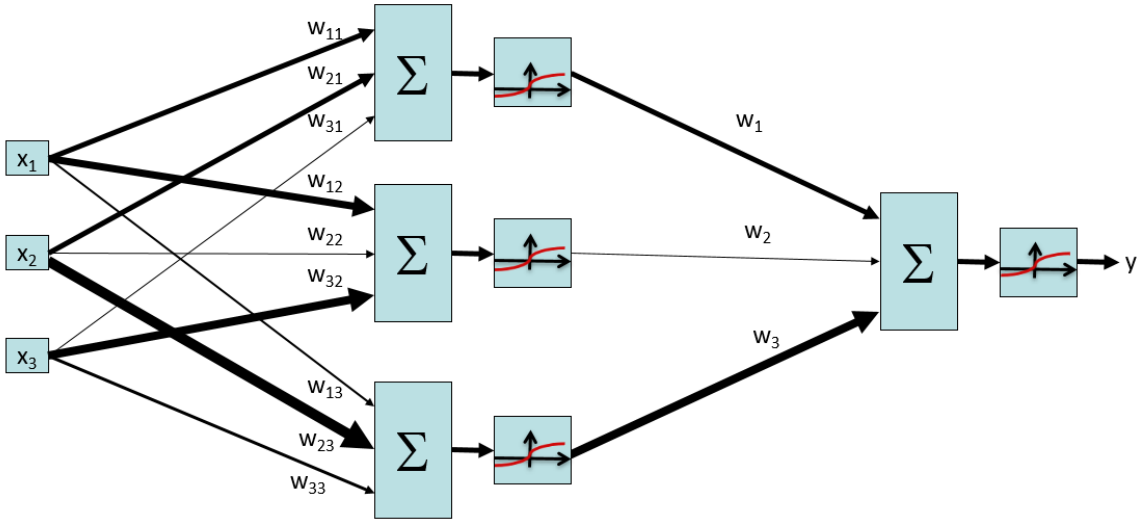
Shape (2, 1).
Weights to be learned.

Shape (1, 1).
Output of network.

2-Layer, 3-Neuron Neural Network



2-Layer, 3-Neuron Neural Network



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

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

$$= \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix}$$

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

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.

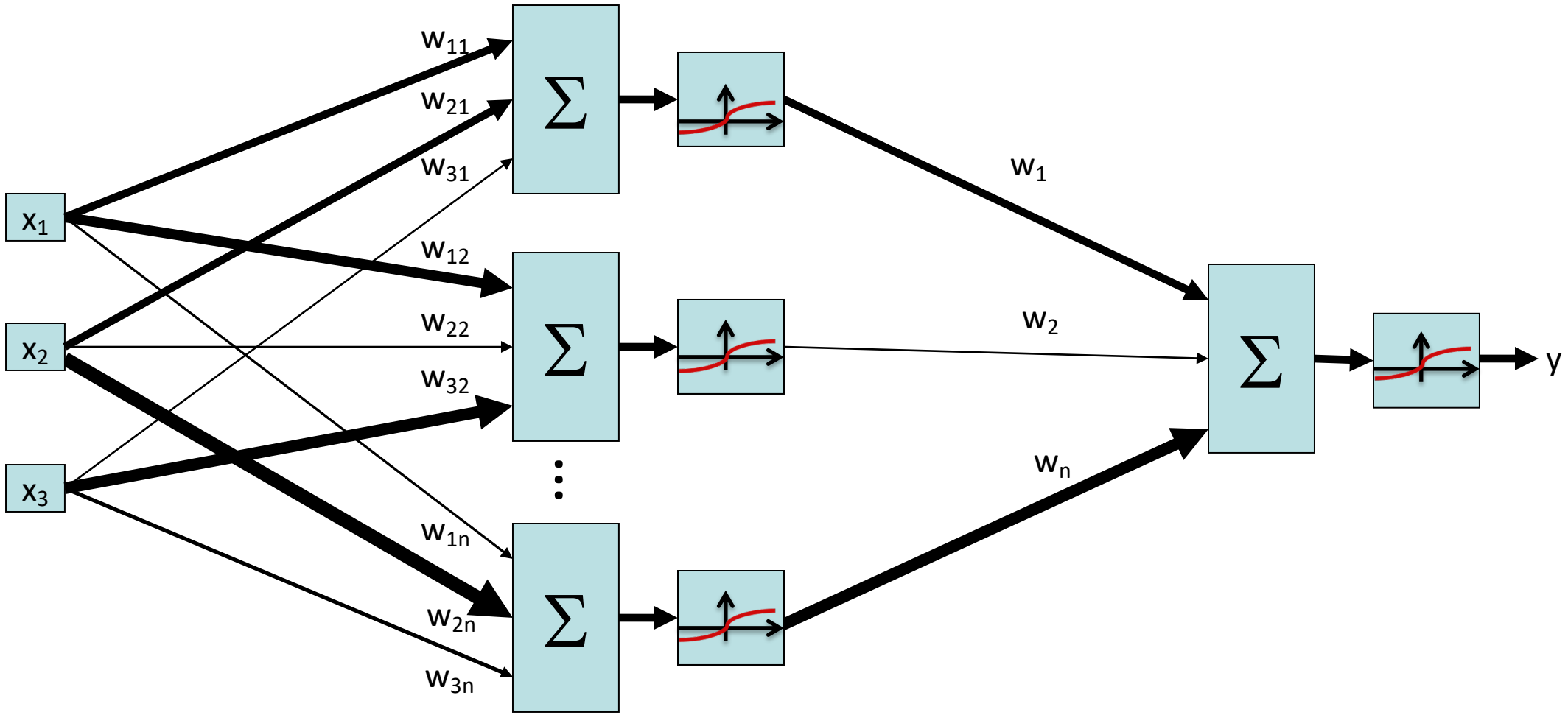
$$\phi(h \times W_{\text{layer 2}}) = y$$

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

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

Shape (1, 1).
Output of network.

Generalize: Number of hidden neurons



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