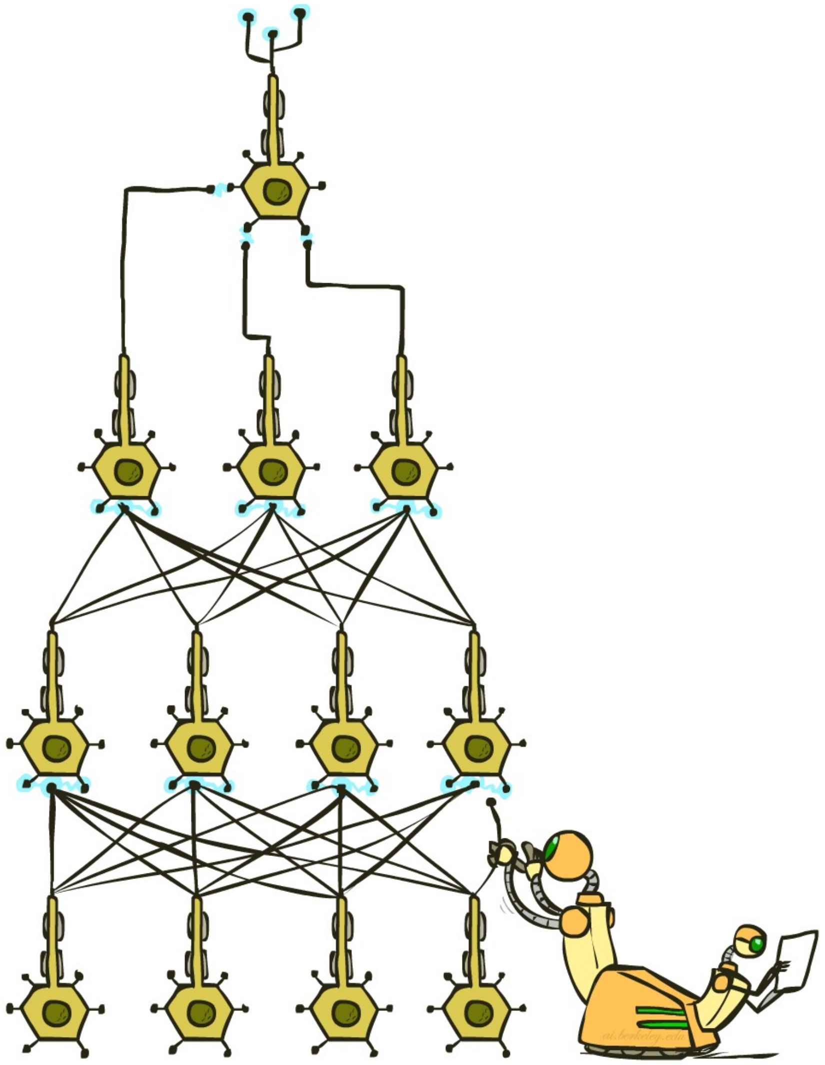
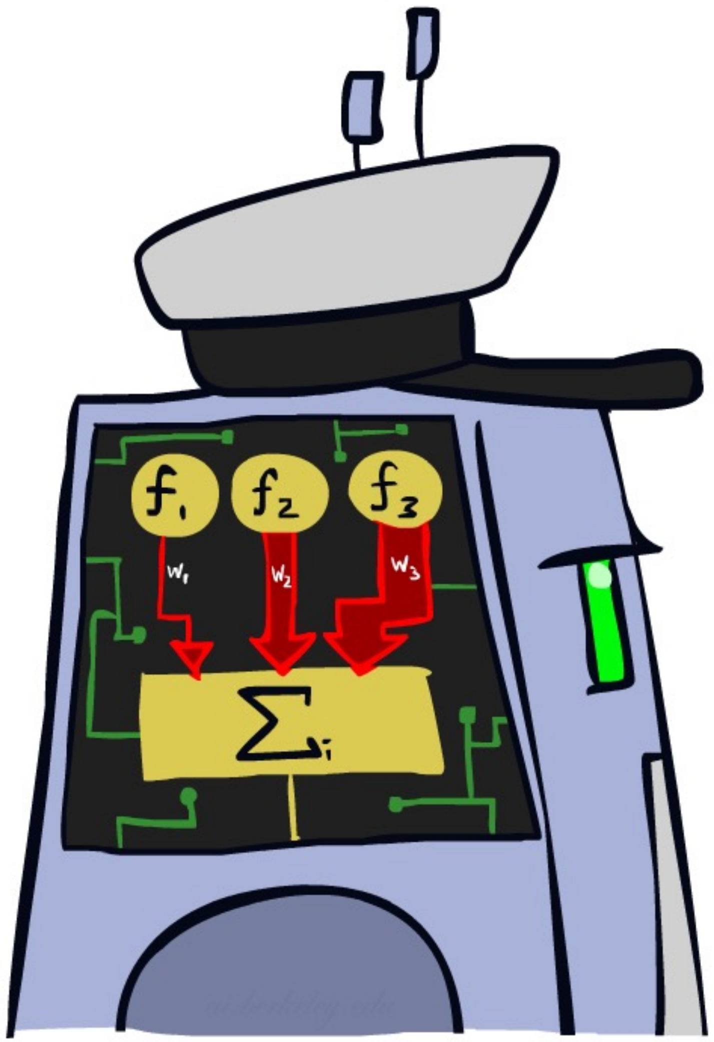


CS 343: Artificial Intelligence

Deep Learning

Prof. Yuke Zhu — The University of Texas at Austin

Review: Linear Classifiers



Feature Vectors

$x \quad\quad f(x) \quad\quad y$

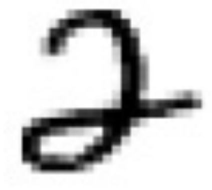
```
Hello,

Do you want free printr
cartriges?  Why pay more
when you can get them
ABSOLUTELY FREE!  Just
```

$\Rightarrow$

```
# free       : 2
YOUR_NAME    : 0
MISSPELLED   : 2
FROM_FRIEND  : 0
...
```

$\Rightarrow$

SPAM
or
+



$\Rightarrow$

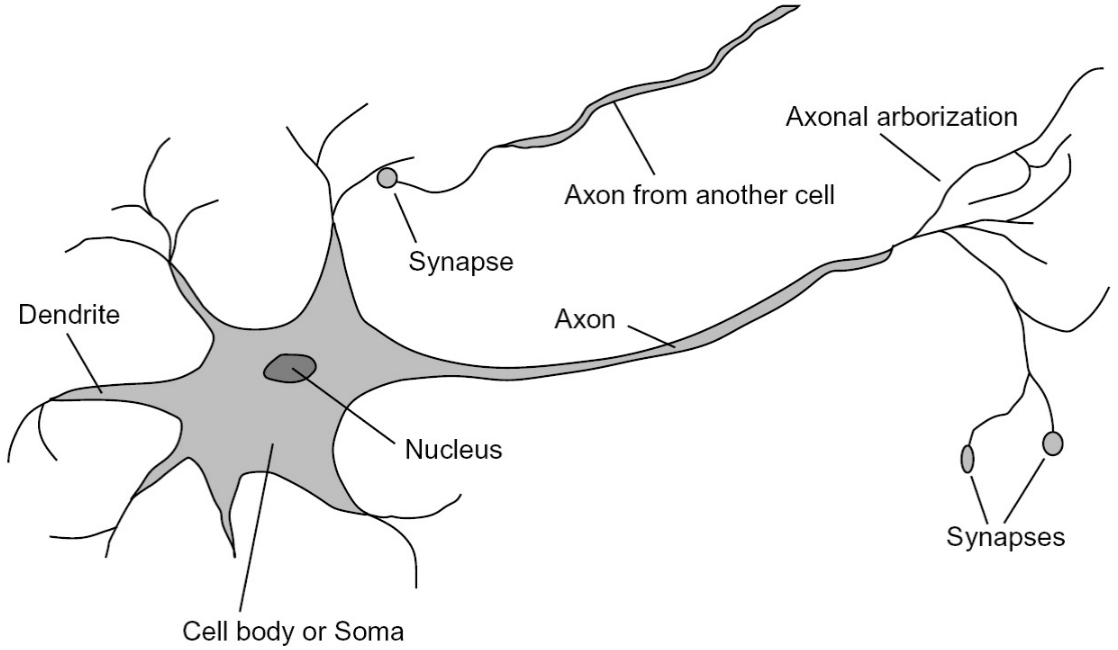
```
PIXEL-7,12   : 1
PIXEL-7,13   : 0
...
NUM_LOOPS    : 1
...
```

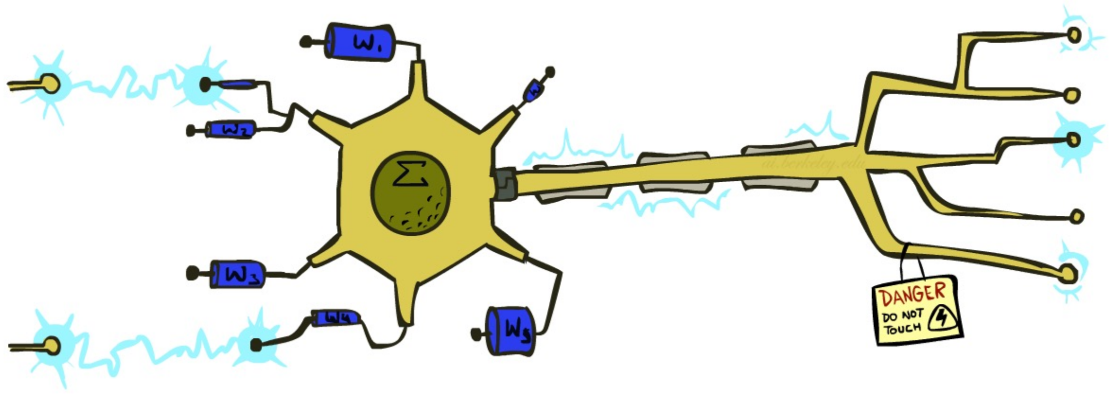
$\Rightarrow$

“2”

Some (Simplified) Biology

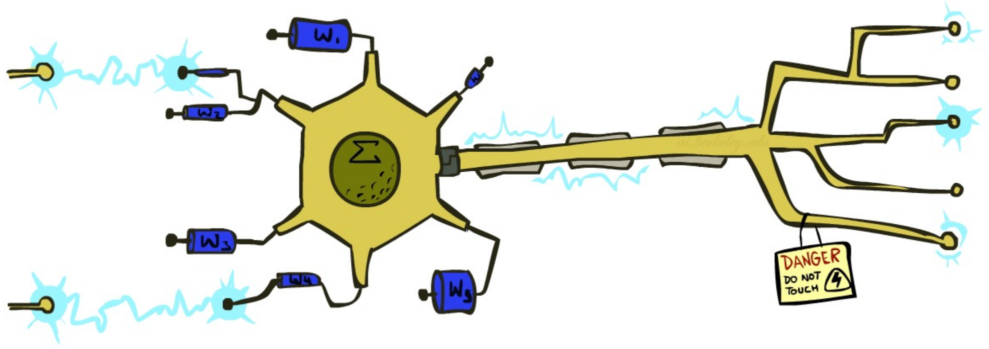
- Very loose inspiration: human neurons





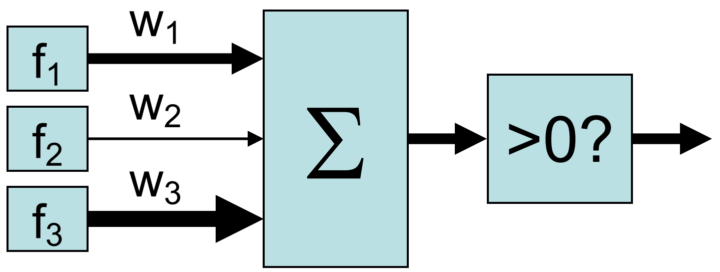
Linear Classifiers

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

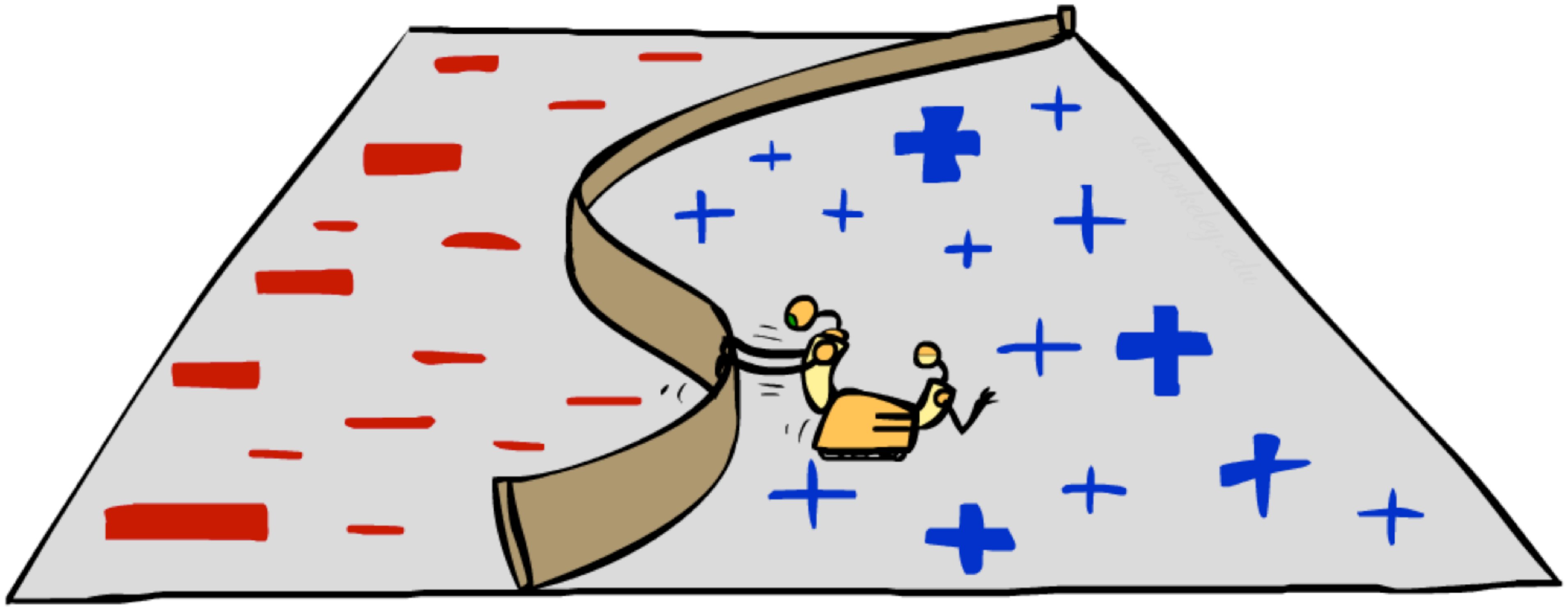


$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

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

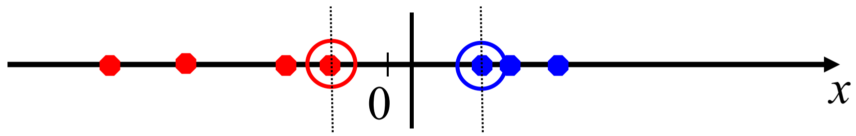


Non-Linearity

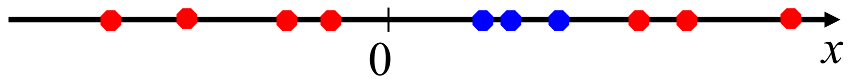


Non-Linear Separators

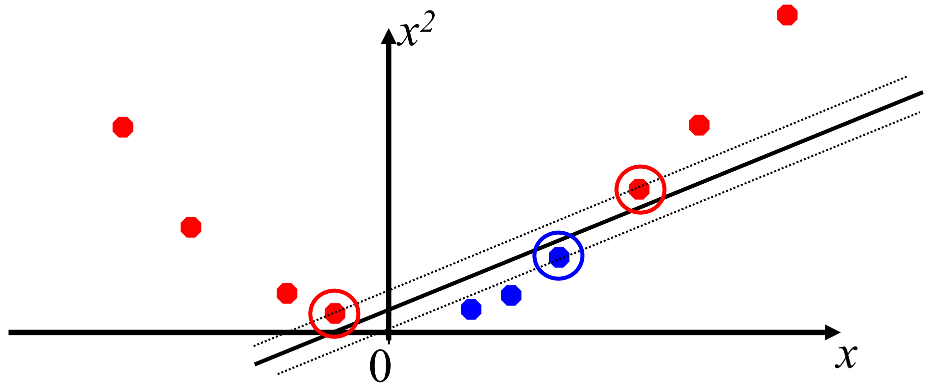
- Data that is linearly separable works out great for linear decision rules:



- But what are we going to do if the dataset is just too hard?



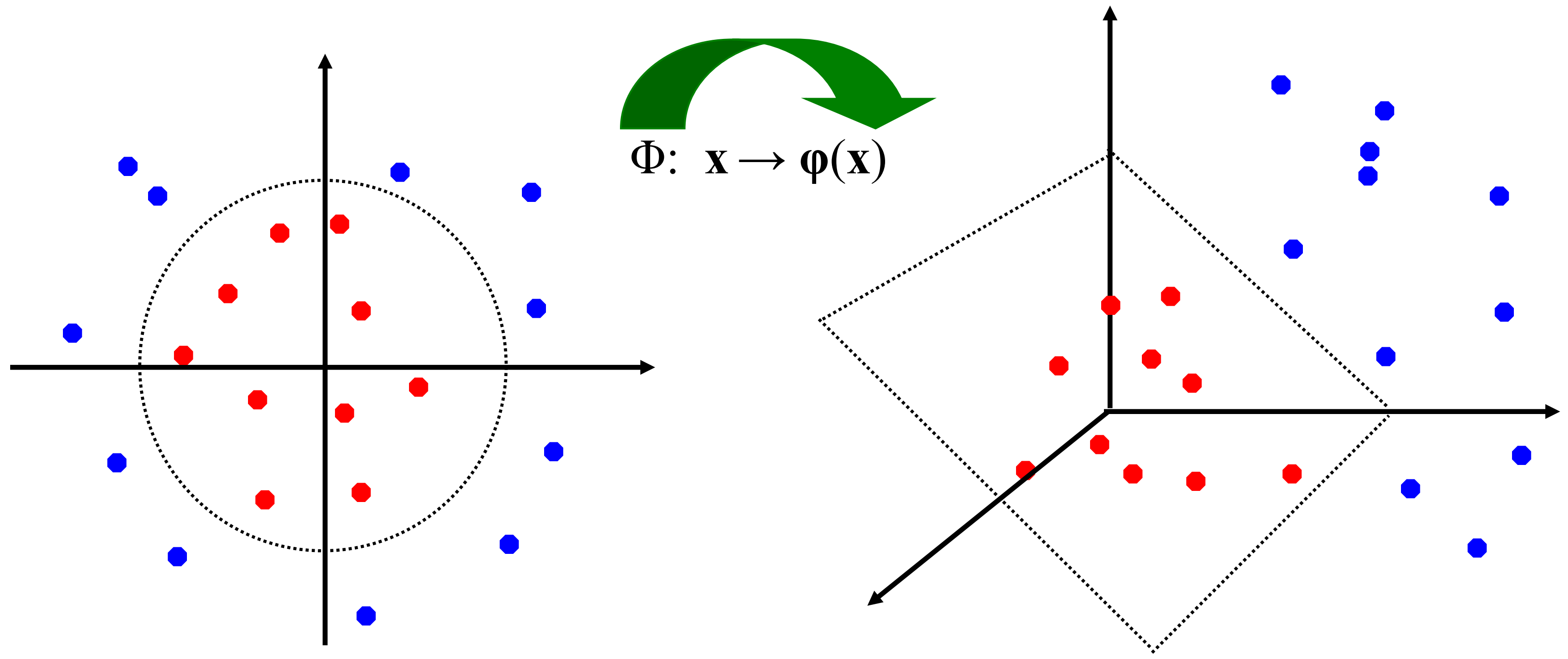
- How about… mapping data to a higher-dimensional space:



This and next slide adapted from Ray Mooney, UT

Non-Linear Separators

- General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:

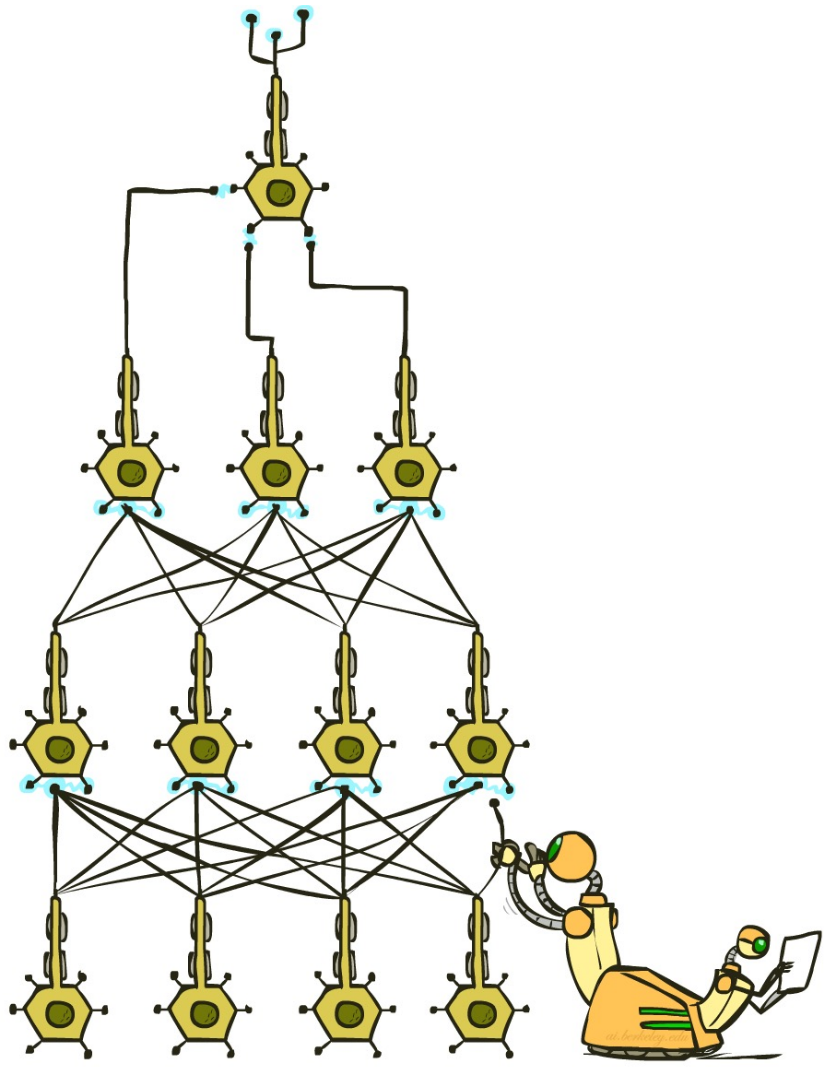


Feature Set Selection

- To choose between two feature sets:
  - For feature set 1: train perceptron on training data -> Classifier 1
  - For feature set 2: train perceptron on training data -> Classifier 2

  - Evaluate performance of Classifier 1 and Classifier 2 on hold-out data
    - Select the one performing best on the hold-out data

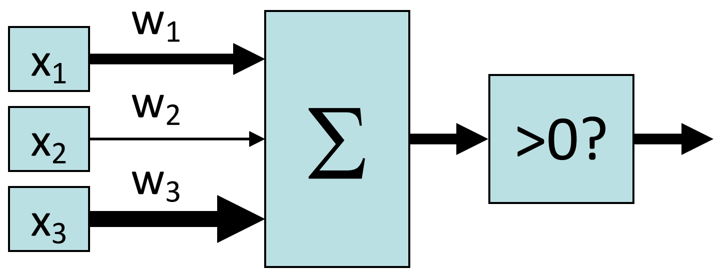
Manual Feature Design →Deep Learning



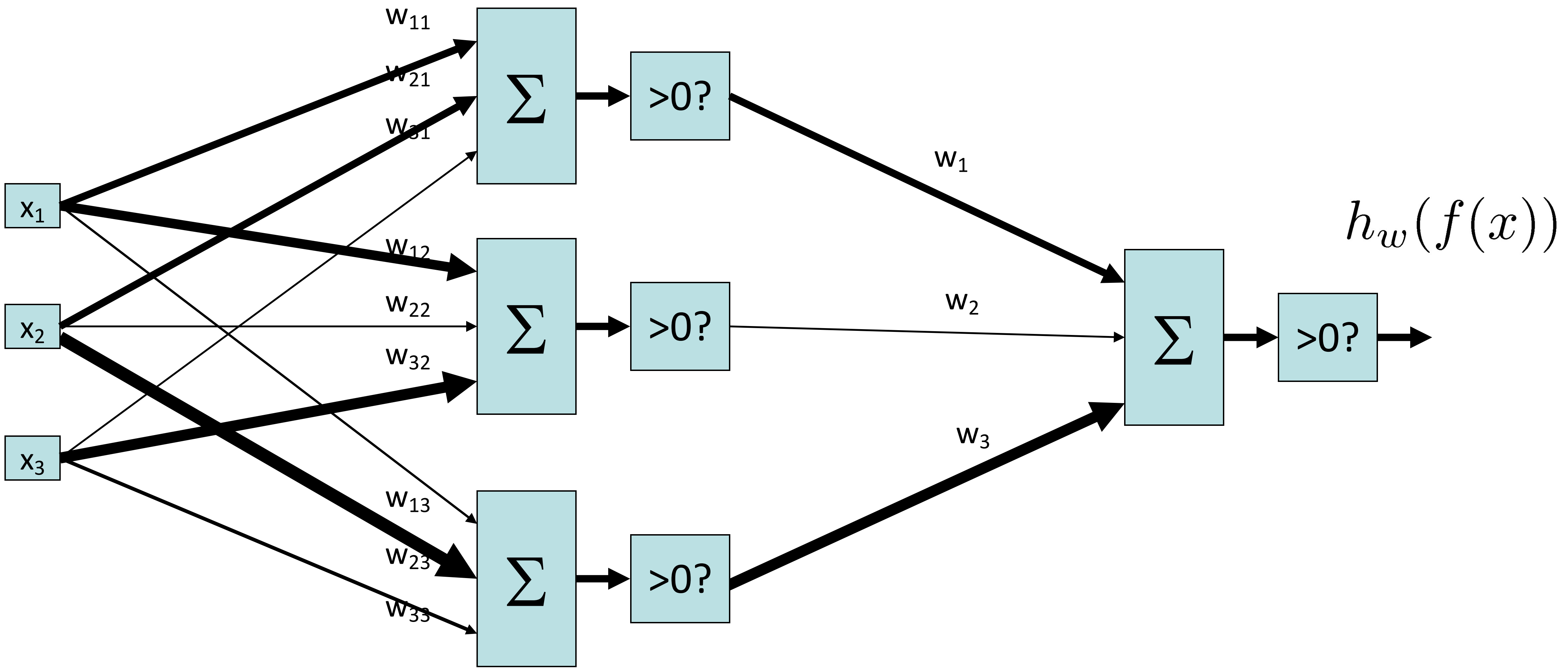
- Manual feature design requires:
  - Domain-specific expertise
  - Domain-specific effort

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

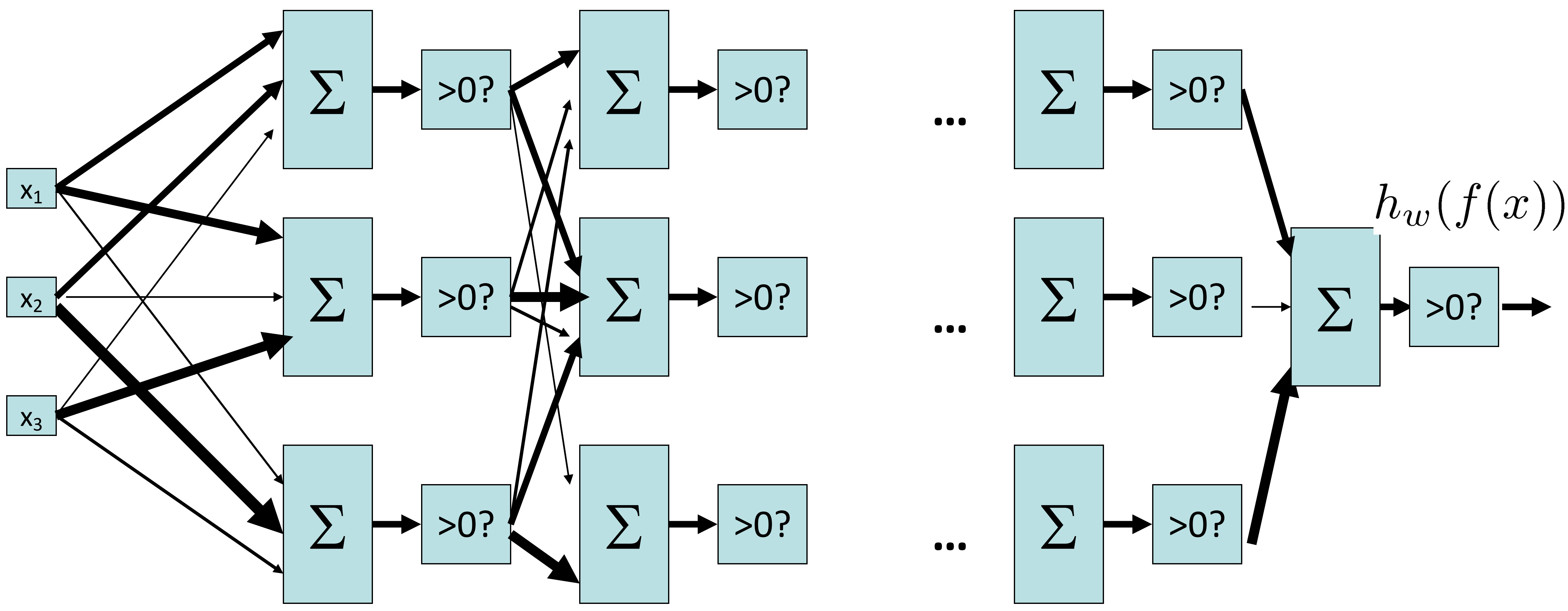
Perceptron



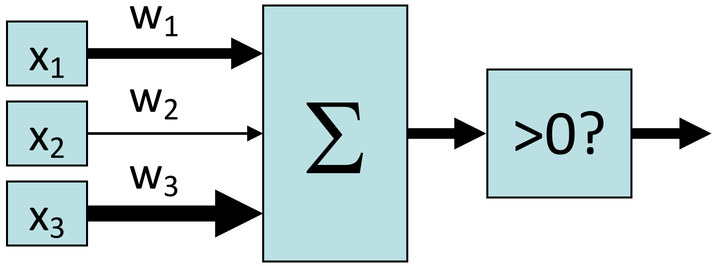
Two-Layer Perceptron Network



N-Layer Perceptron Network



Perceptron



- Objective: Classification Accuracy

$$l^{\text{acc}}(w) = \frac{1}{m}\sum_{i=1}^{m}\left(\text{sign}(w^\top f(x^{(i)})) == y^{(i)}\right)$$

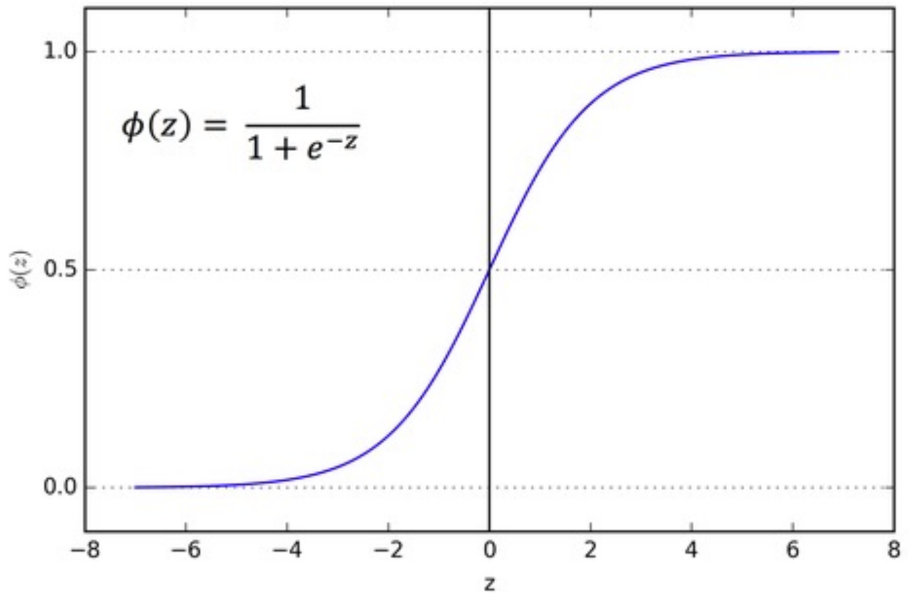
  - Issue: many plateaus → how to measure incremental progress toward a correct label?

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:

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

with:

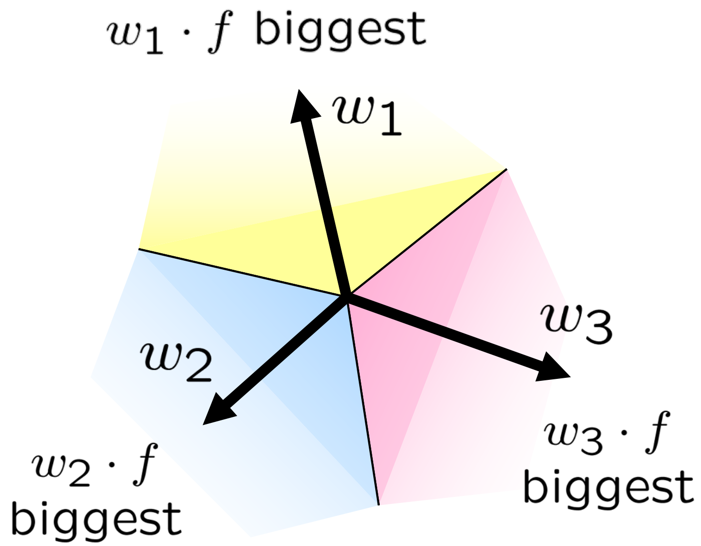
$$P(y^{(i)} = +1|x^{(i)};w) = \frac{1}{1+e^{-w\cdot f(x^{(i)})}}$$

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

= 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?

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

Best w?

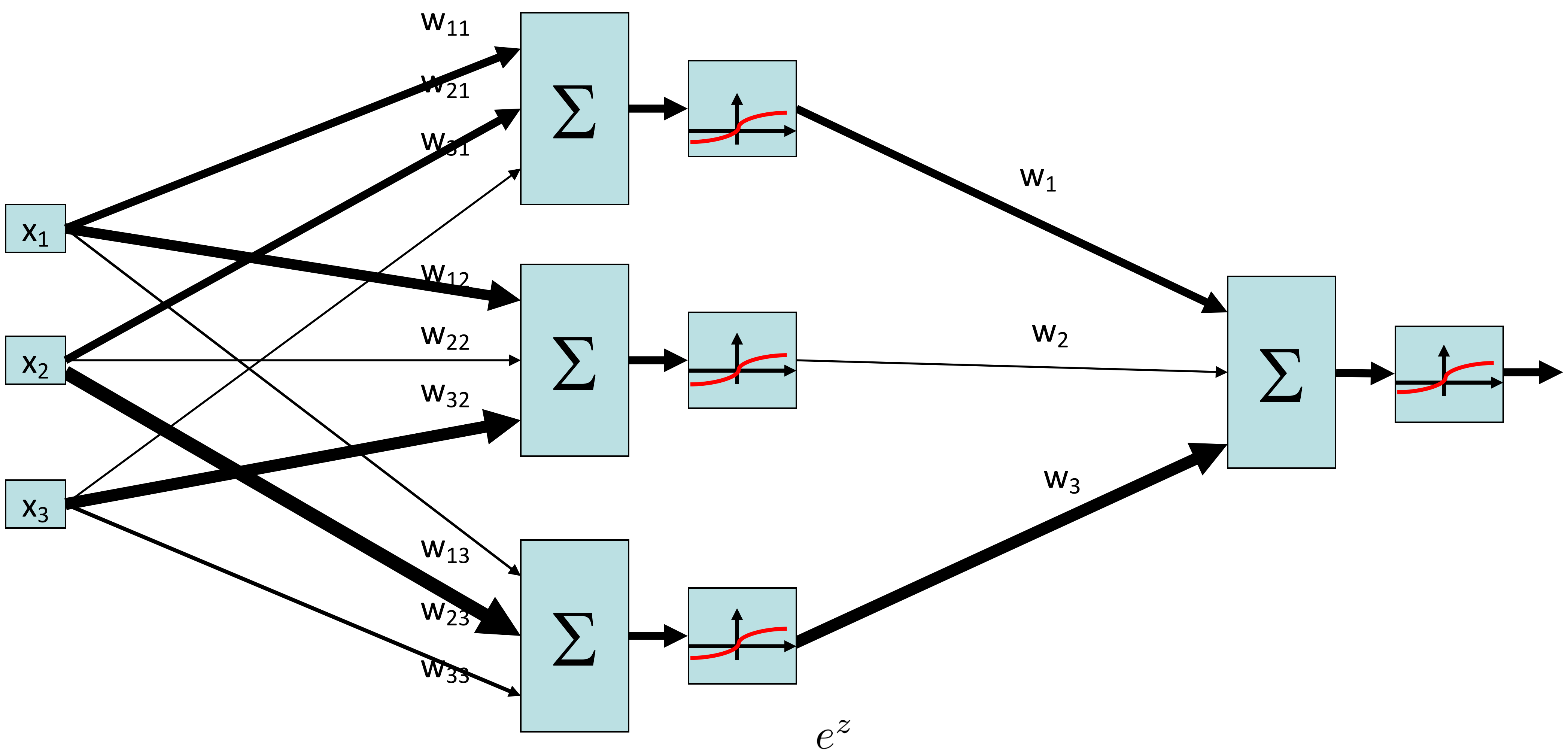
- Maximum likelihood estimation:

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

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

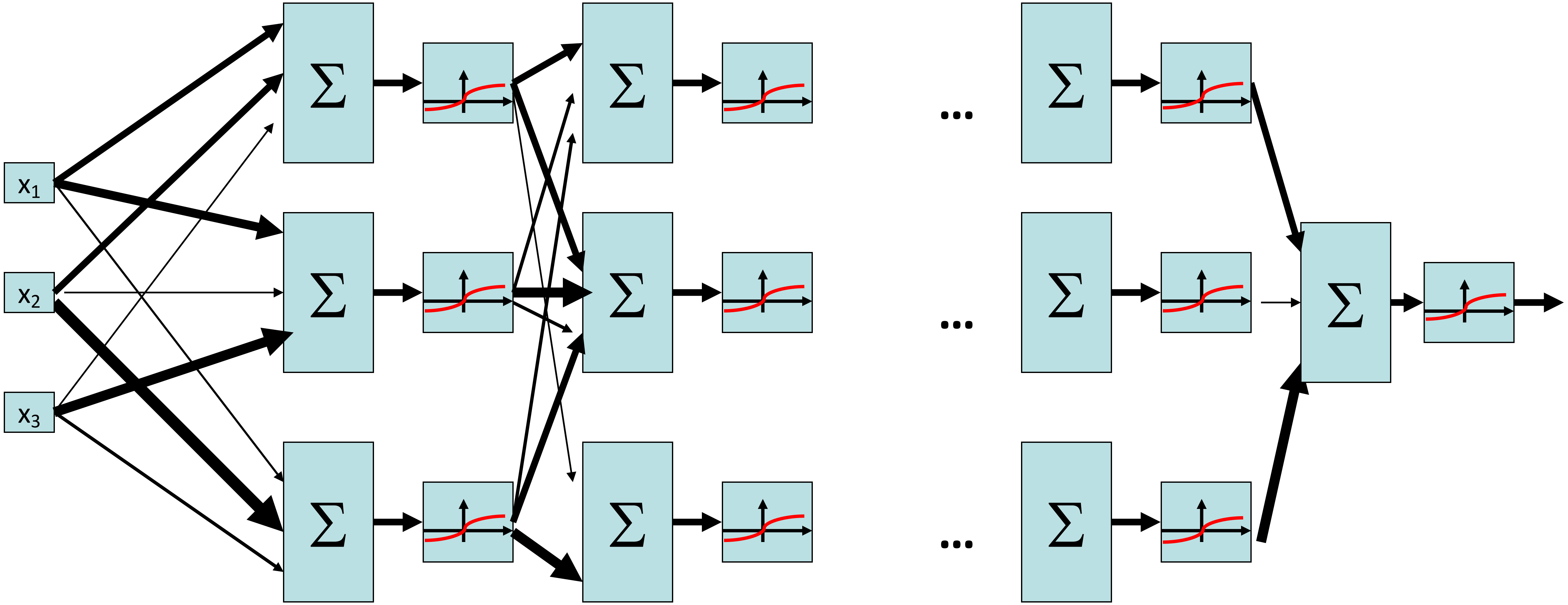
= Multi-Class Logistic Regression

Two-Layer Neural Network



$$z \to \frac{e^z}{e^z + e^{-z}}$$

N-Layer Neural Network



Best w?

- Optimization
  - i.e., how do we solve:

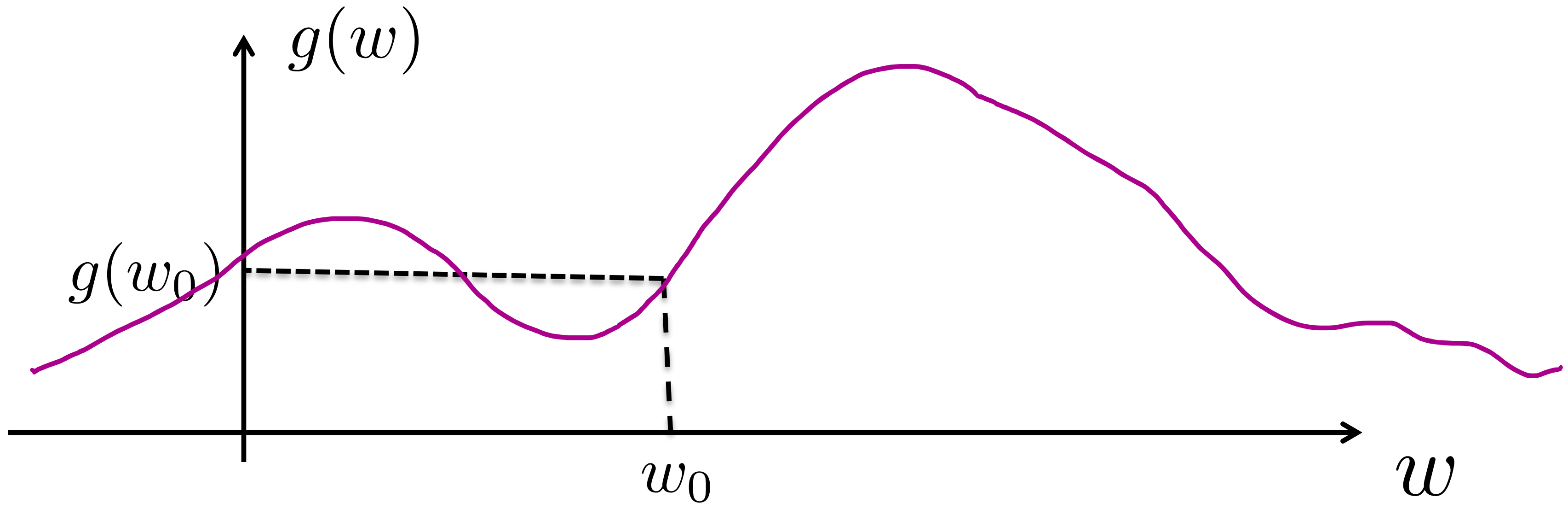
$$\max_{w} \quad ll(w) = \max_{w} \quad \sum_{i} \log P(y^{(i)}|x^{(i)}; w)$$

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?

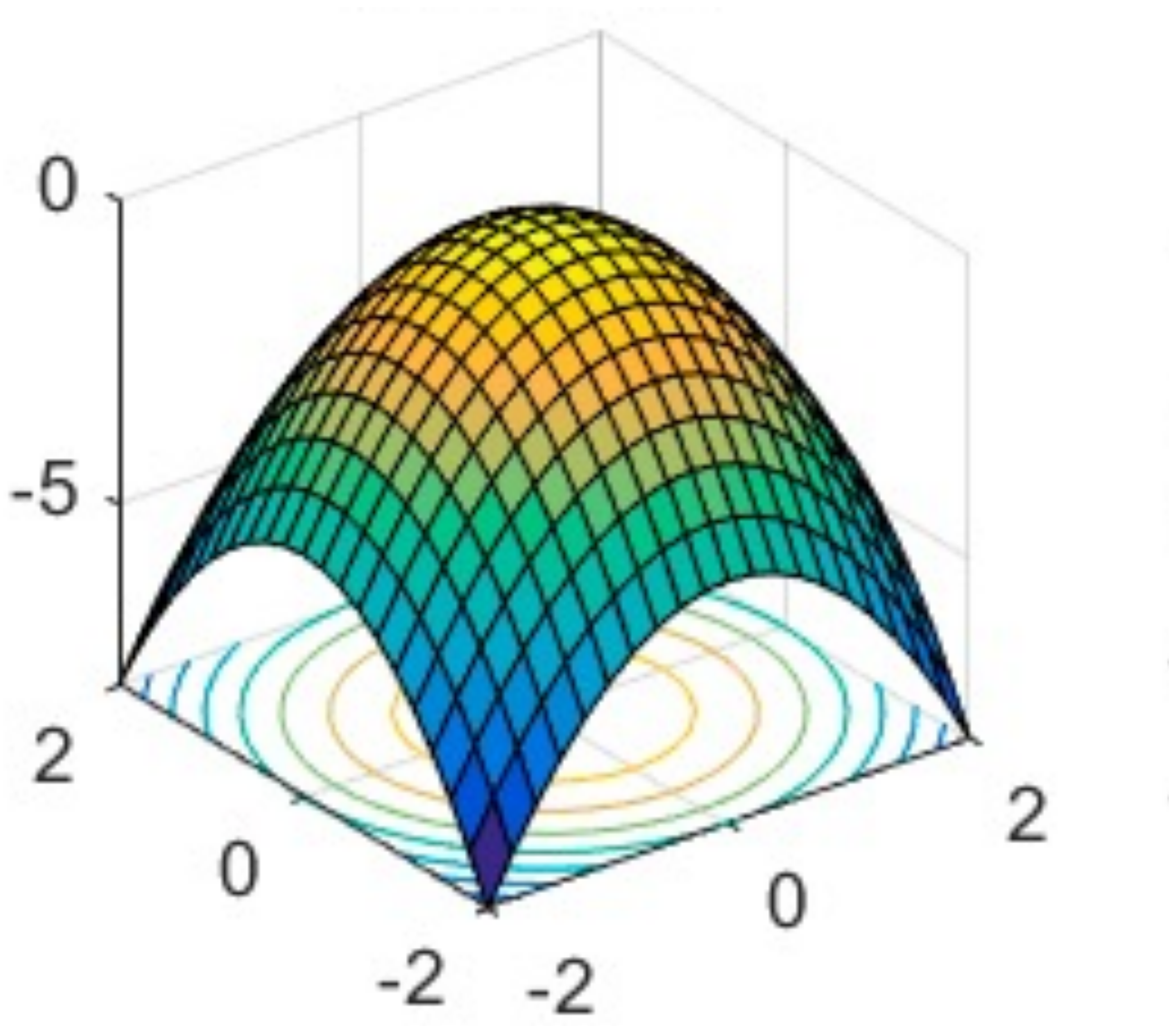
1-D Optimization



- 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

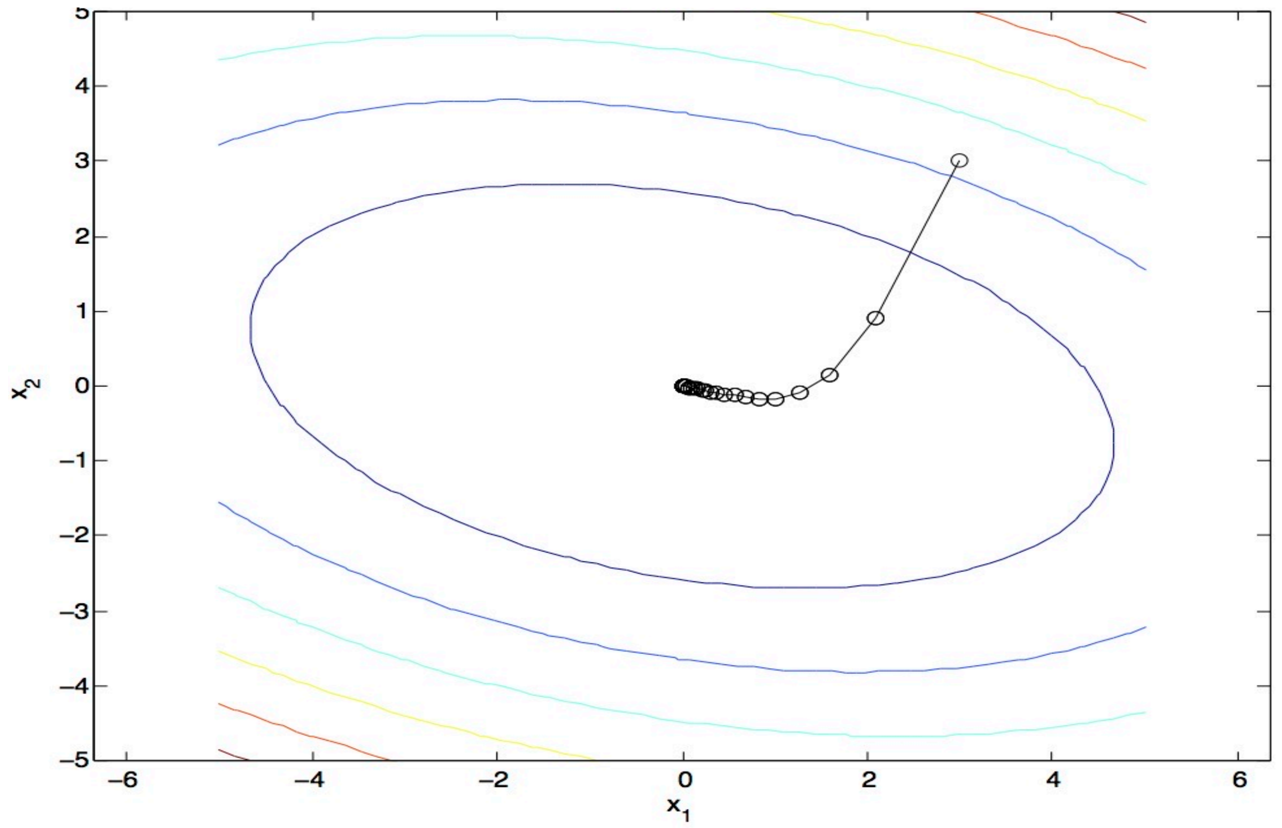


Figure source: Mathworks

Gradient 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, …
        $w \leftarrow w + \alpha * \nabla g(w)$
```

- 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 %

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

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

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

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

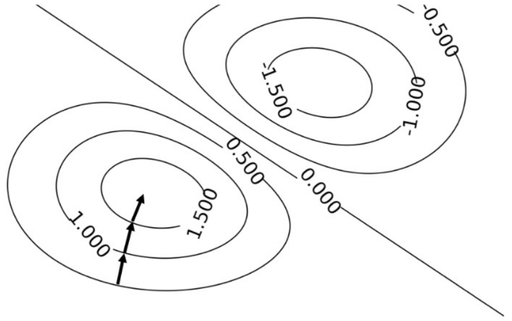
Observation: gradient over small set of training examples (=mini-batch) can be computed in parallel, might as well do that instead of a single one

```
▪ init $w$
▪ for iter = 1, 2, …
    ▪ pick random subset of training examples J
        $w \leftarrow w + \alpha * \sum_{j \in J} \nabla \log P(y^{(j)}|x^{(j)}; w)$
```

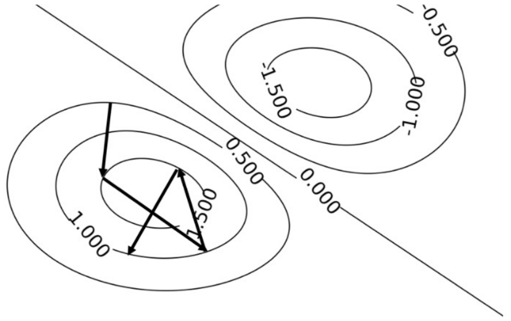
Exercises: Gradient Ascent

Which of the following paths is a feasible trajectory for the gradient ascent algorithm?

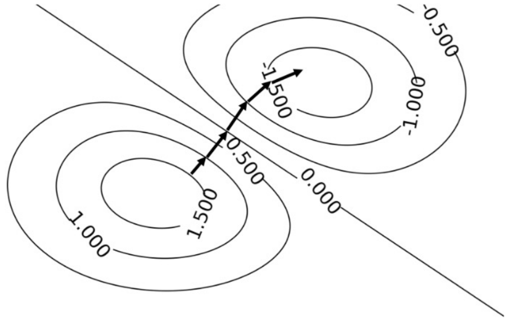
A



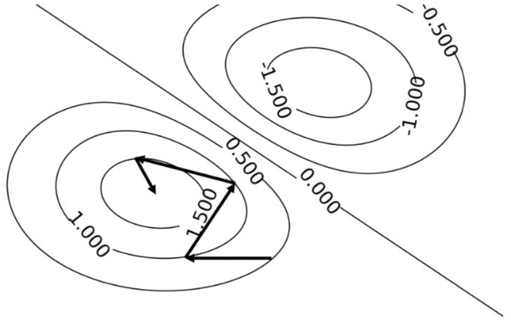
B



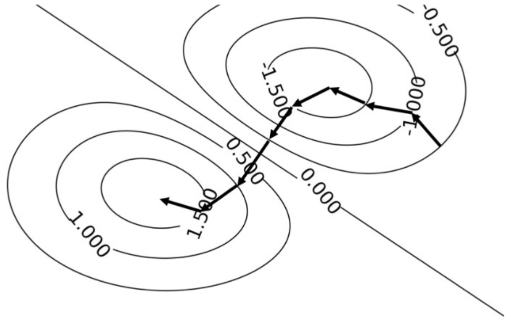
C



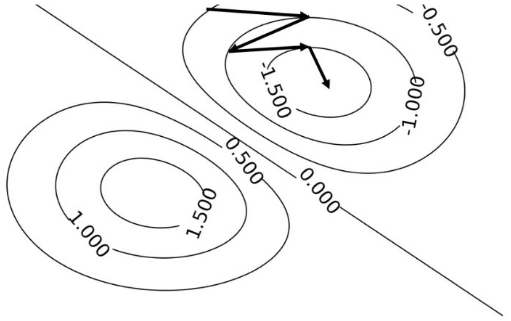
D



E



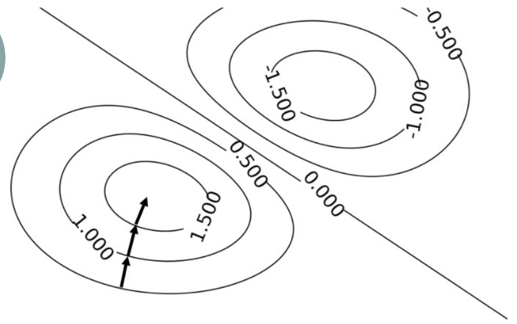
F



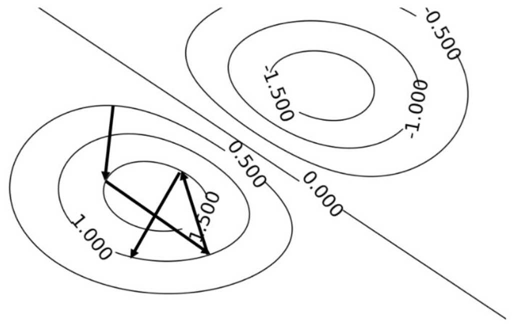
Exercises: Gradient Ascent

Which of the following paths is a feasible trajectory for the gradient ascent algorithm?

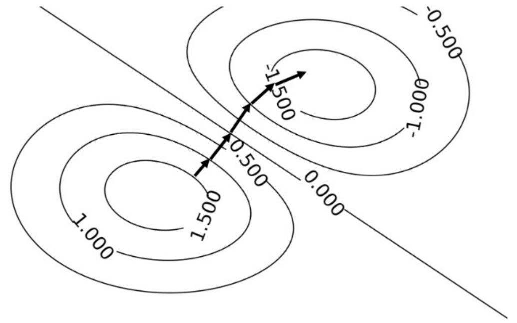
[✓ A]



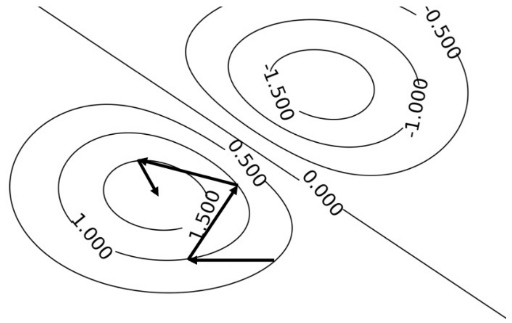
[✓ B]



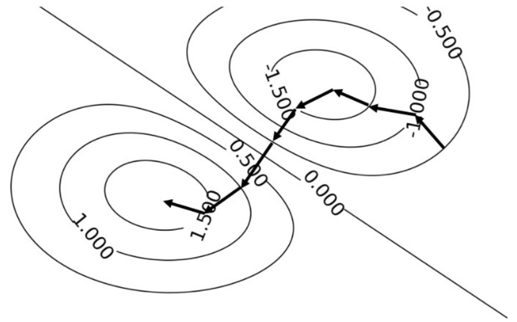
C



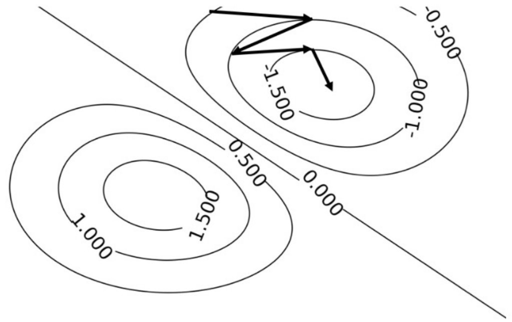
D



E



F



🟥 A 🟥 B ☐ C ☐ D ☐ E ☐ F

A is a gradient ascent path since the gradient lines are orthogonal to the contours and the point towards the maximum. B is also a gradient ascent path with a high learning rate. C is not because the path is going towards the minimum instead of the maximum. D is not a gradient ascent path since the gradient is not orthogonal to the contour lines. E is not a gradient ascent path since it starts going towards the minimum. F is not since it goes towards the minimum and the gradients are not orthogonal to the contour lines.

How about computing all the derivatives?

- Maximum likelihood estimation:

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

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

Try it on data point $(\hat{x}, \hat{y})$?

= Multi-Class Logistic Regression