

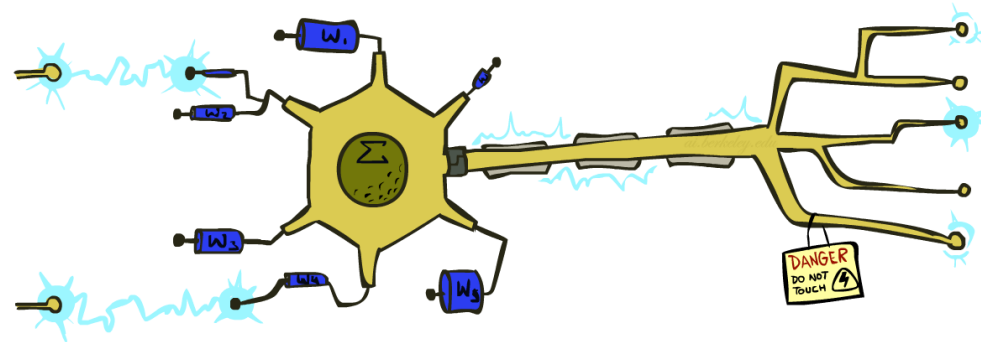
# CS 343H: Artificial Intelligence

## Deep Learning

Prof. Yuke Zhu — The University of Texas at Austin

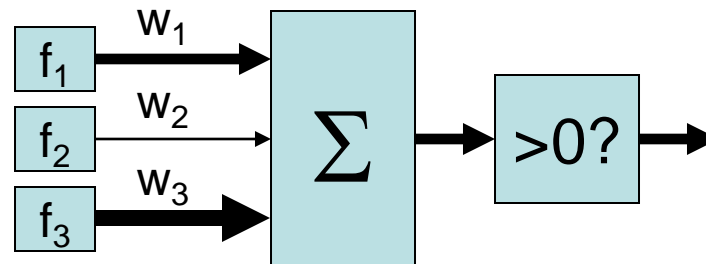
# Linear Classifiers

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



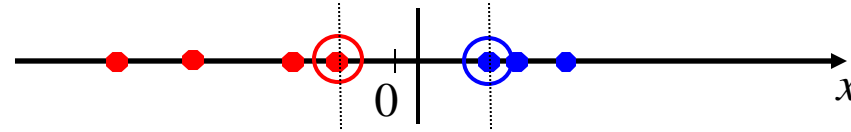
$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

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

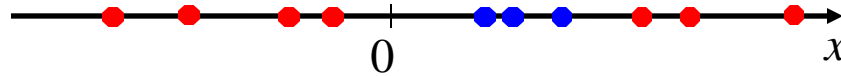


# Non-Linear Separators

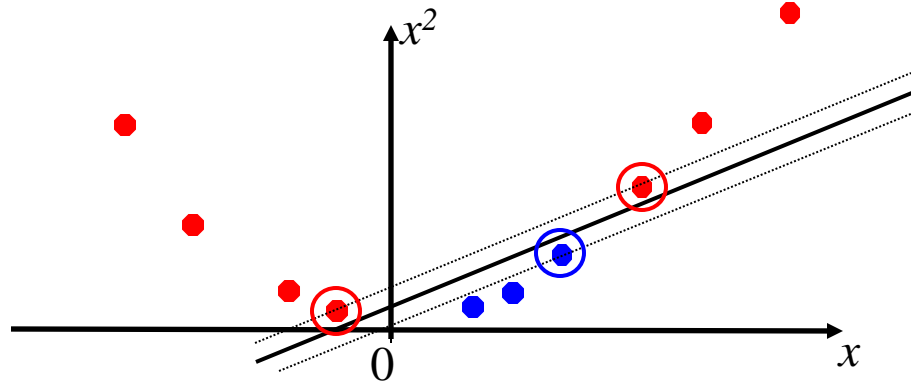
- Data that is linearly separable works out great for linear decision rules:



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

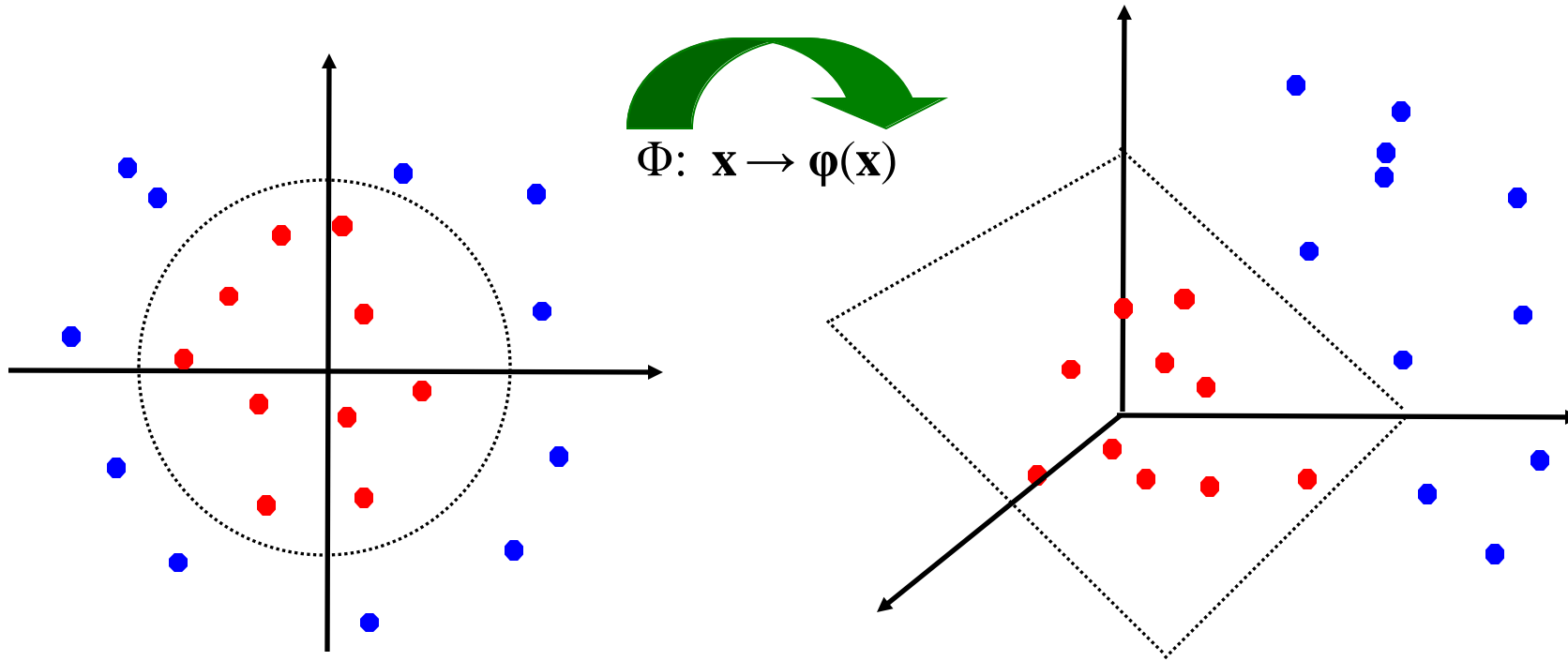


- How about... mapping data to a higher-dimensional space:



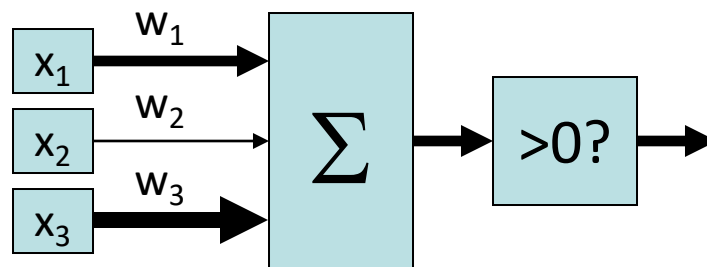
# Non-Linear Separators

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

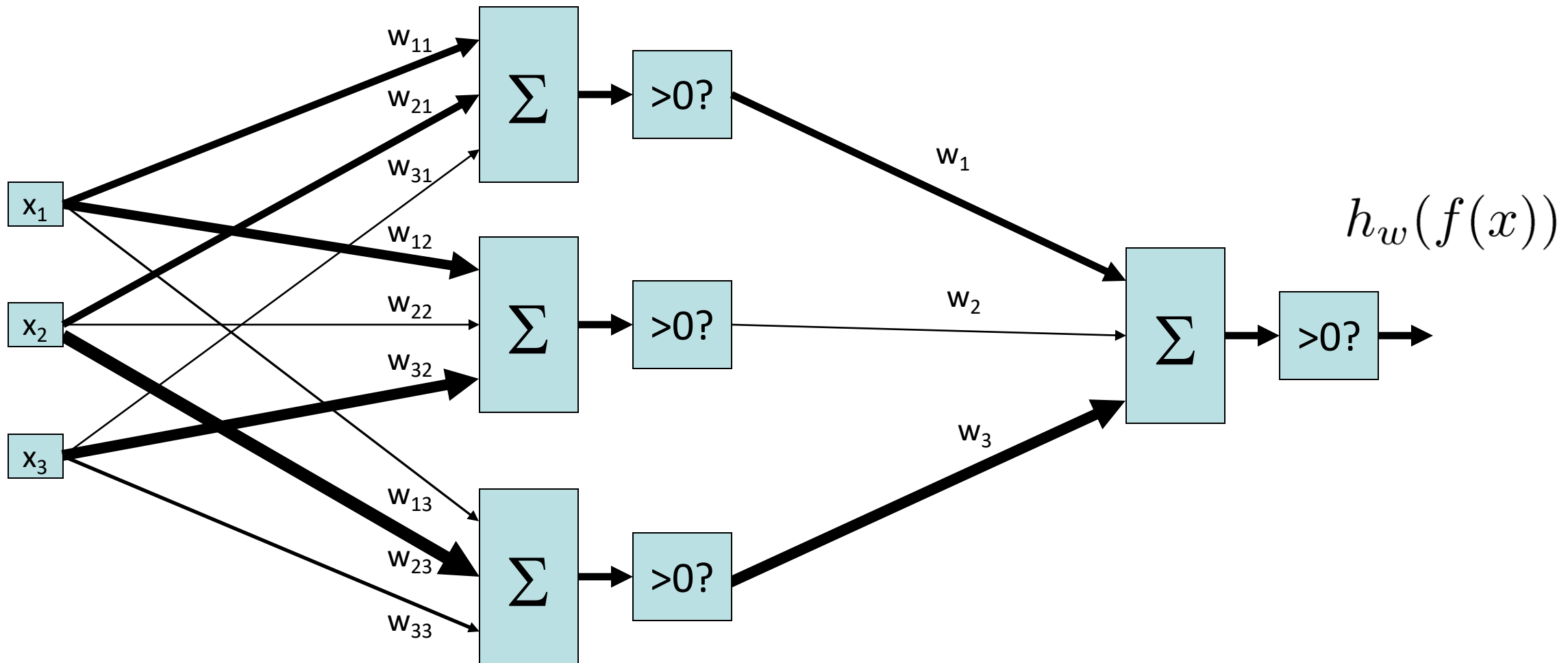


# Perceptron

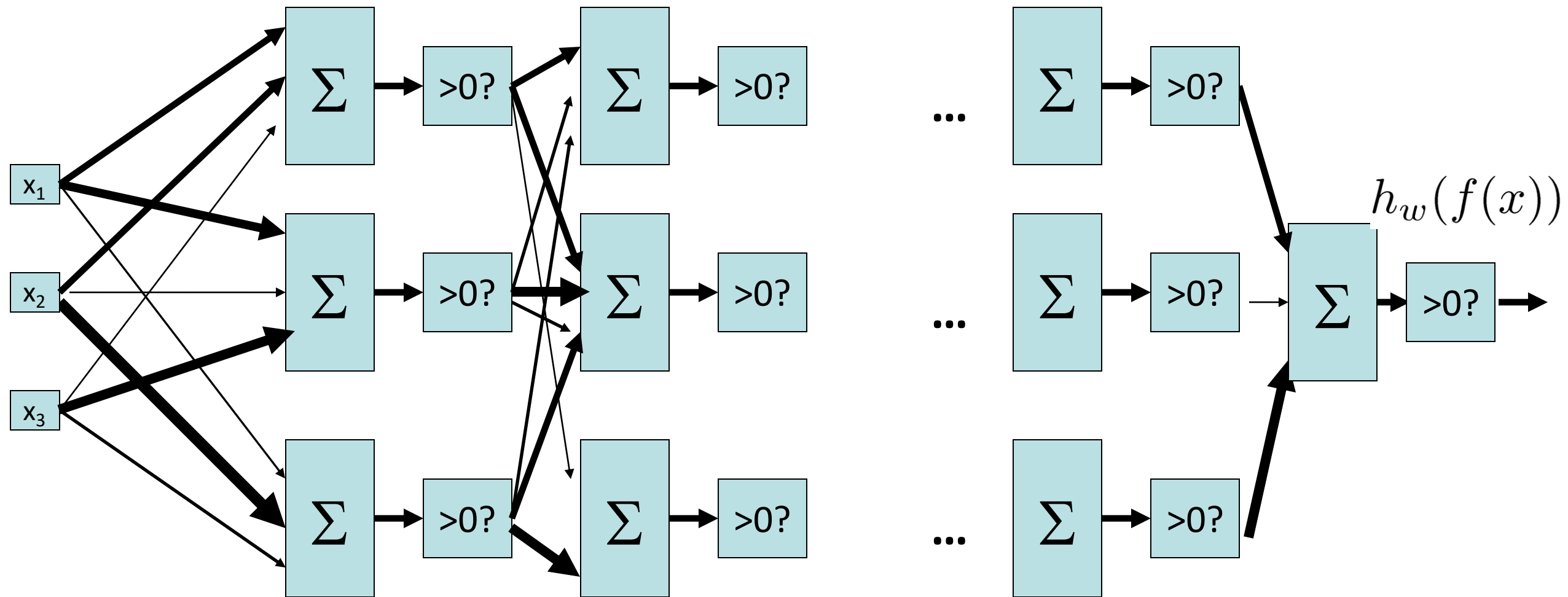
---



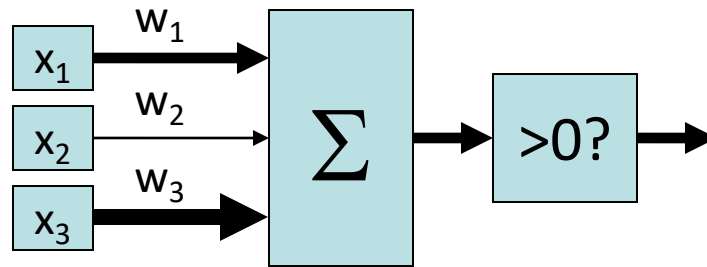
# Two-Layer Perceptron Network



# N-Layer Perceptron Network



# Perceptron



- Objective: Classification Accuracy

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

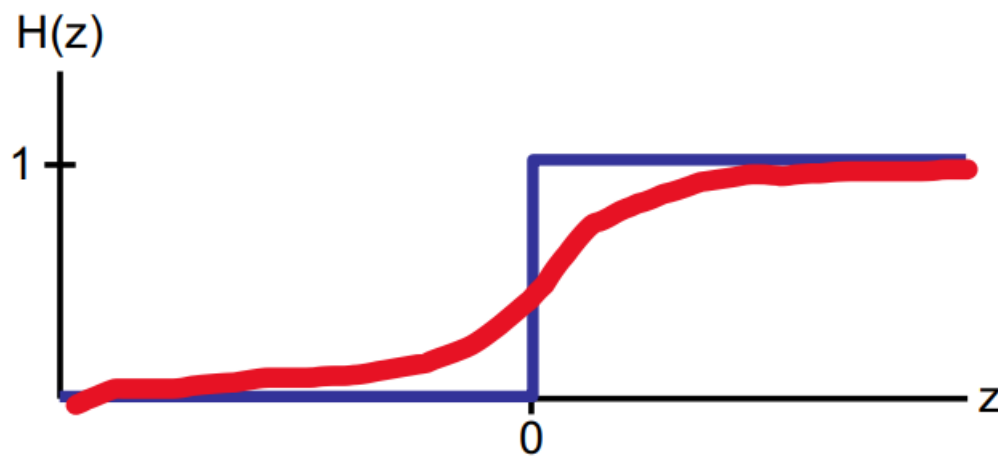


# Perceptrons give deterministic decisions

- Perceptron scoring:  $z = w \cdot f(x)$
- If  $z = w \cdot f(x)$  positive  $\rightarrow$  classifier says: 1.0 probability this is class +1
- If  $z = w \cdot f(x)$  negative  $\rightarrow$  classifier says: 0.0 probability this is class +1

- Step function

$$H(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$$



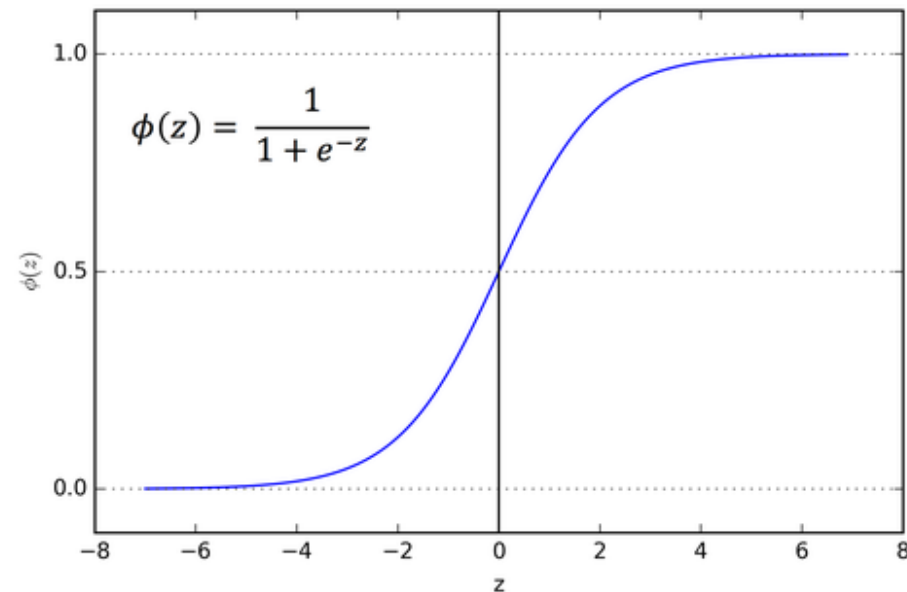
- $z$  = output of perceptron  
 $H(z)$  = probability the class is +1, according to the classifier

# How to get probabilistic decisions?

- Activation:  $z = w \cdot f(x)$
- If  $z = w \cdot f(x)$  very positive  $\rightarrow$  probability to approach 1
- If  $z = w \cdot f(x)$  very negative  $\rightarrow$  probability to approach 0

- Sigmoid function

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



# Find the best $w$ in logistic regression?

- Maximum likelihood estimation:

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

with:

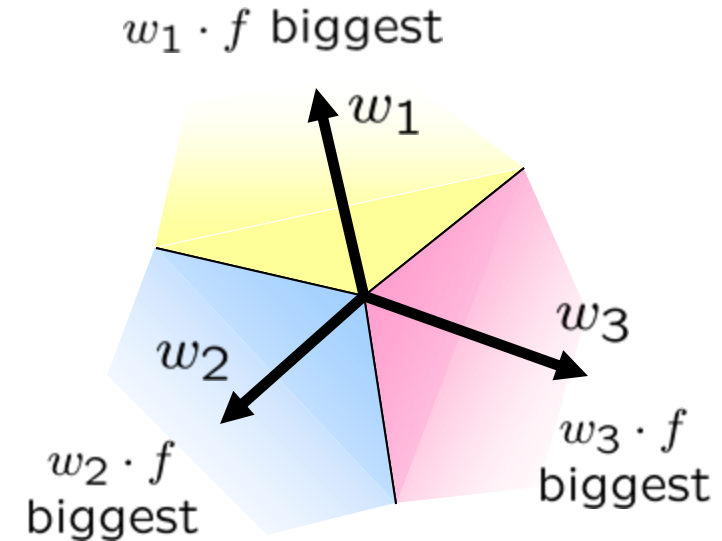
$$P(y^{(i)} = +1 | x^{(i)}; w) = \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$
$$P(y^{(i)} = -1 | x^{(i)}; w) = 1 - \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

**= That' Logistic Regression**

# Multiclass Logistic Regression

- Multi-class linear classification

- A weight vector for each class:  $w_y$
- Score (activation) of a class  $y$ :  $z_y = w_y \cdot f(x)$
- 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:

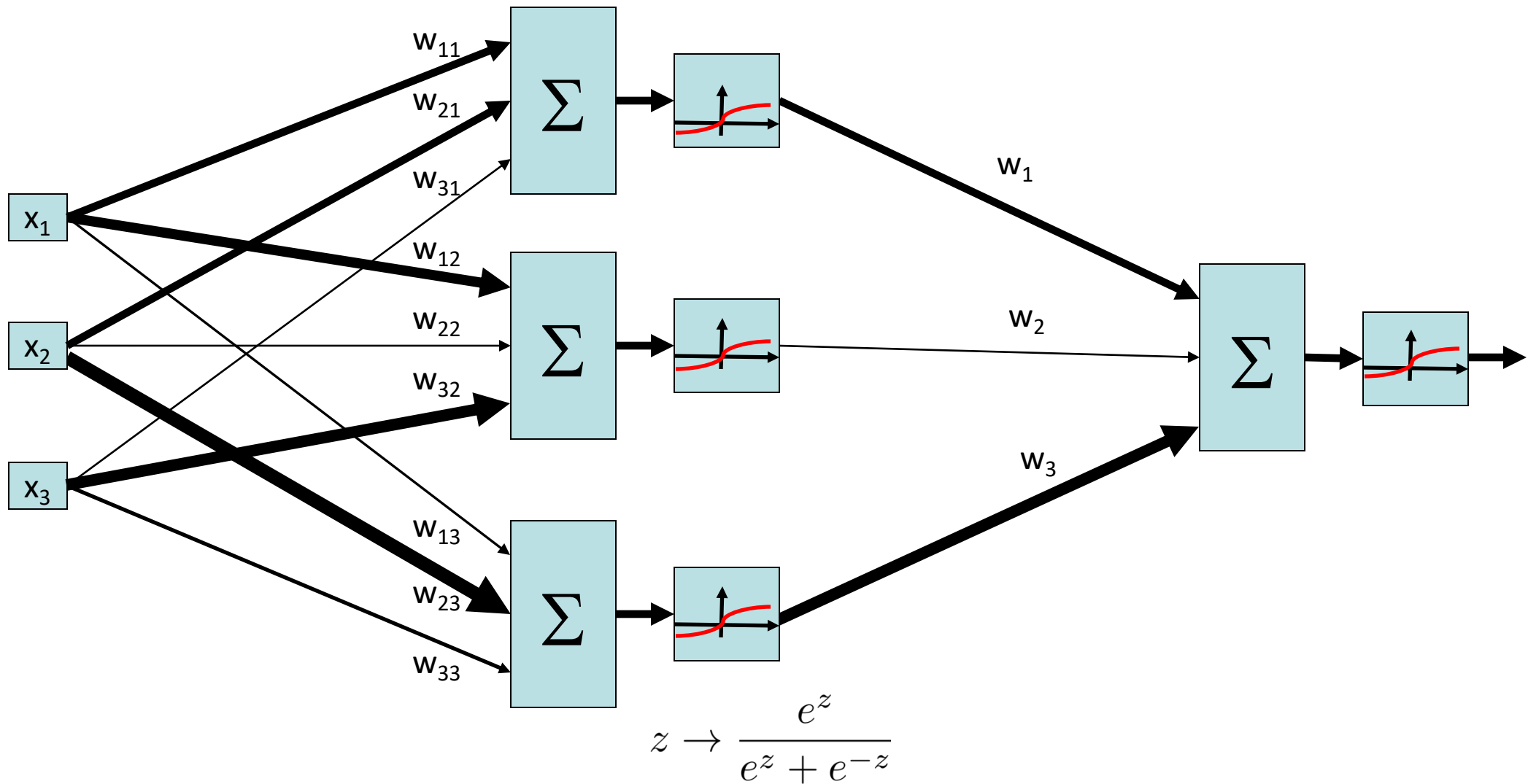
$$\max_w ll(w) = \max_w \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

with:

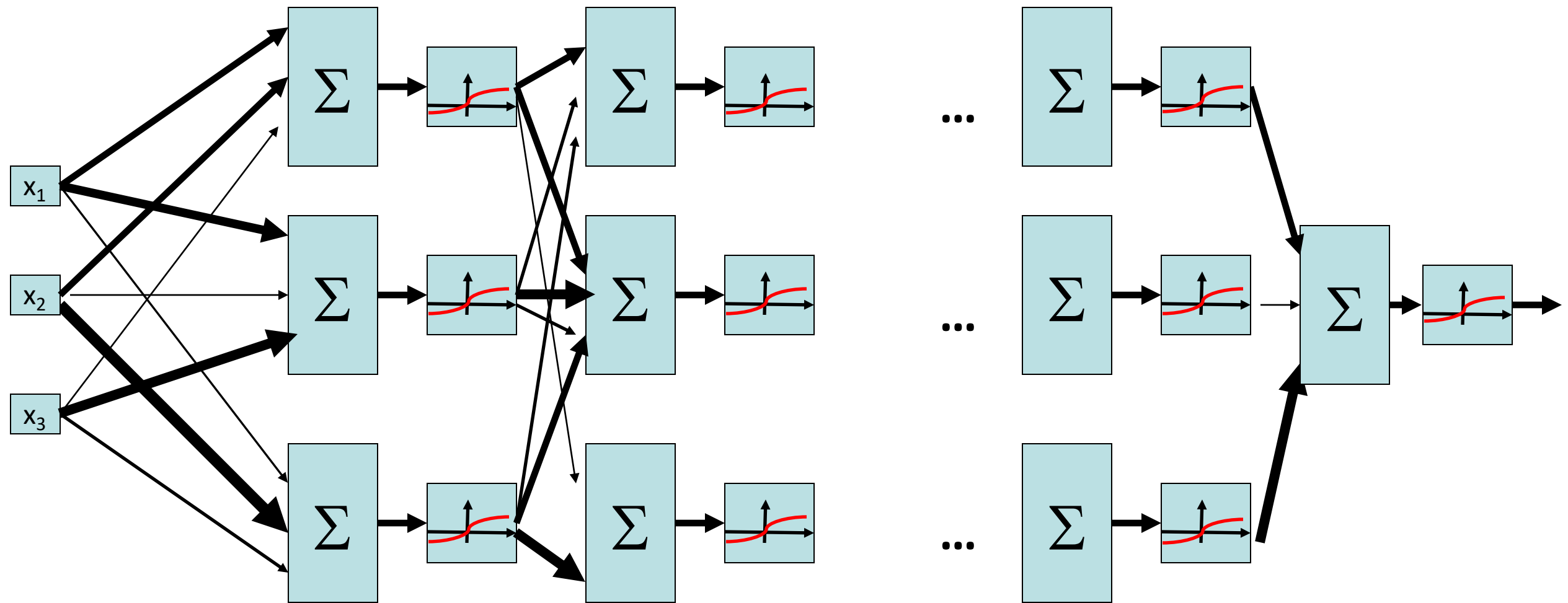
$$P(y^{(i)} | x^{(i)}; w) = \frac{e^{w_{y^{(i)}} \cdot f(x^{(i)})}}{\sum_y e^{w_y \cdot f(x^{(i)})}}$$

**= Multi-Class Logistic Regression**

# Two-Layer Neural Network



# N-Layer Neural Network



# Best $w$ ?

---

- Optimization

- i.e., how do we solve:

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

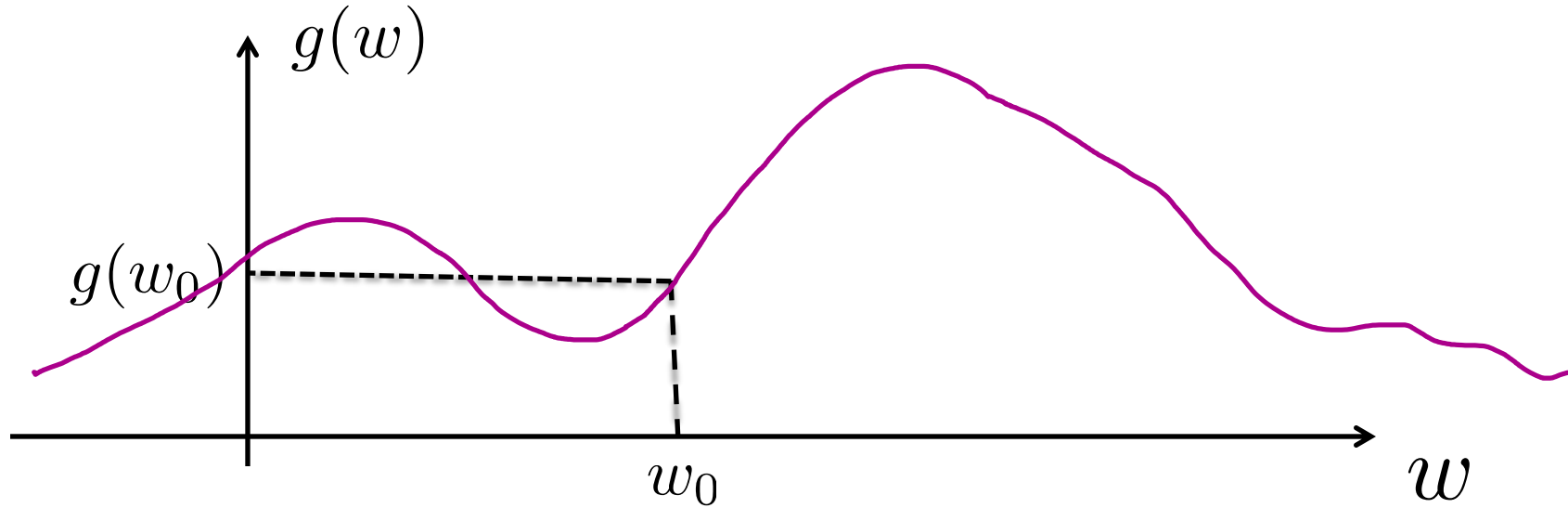


# Hill Climbing

- 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?



# 1-D Optimization



- Could evaluate  $g(w_0 + h)$  and  $g(w_0 - h)$

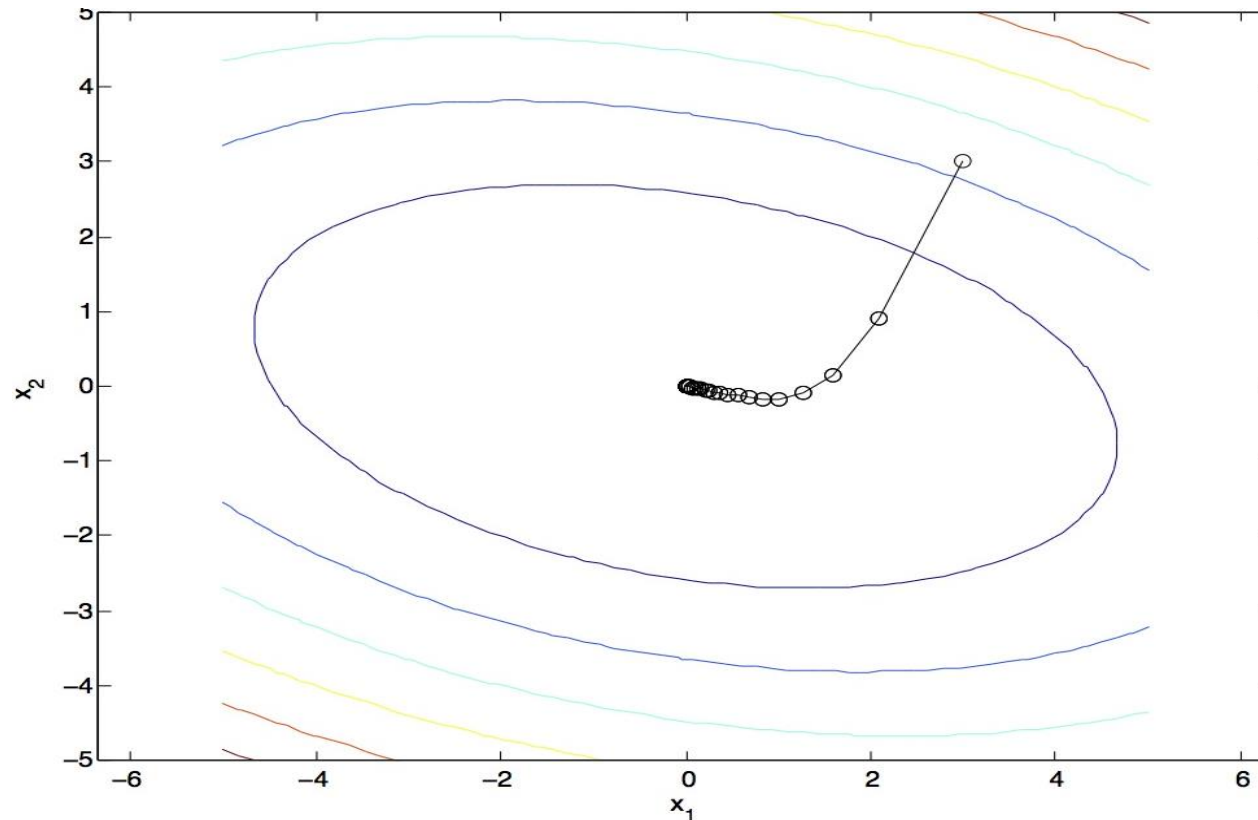
- Then step in best direction

- 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

# Gradient Ascent

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



# Gradient in n dimensions

---

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

# Deep Neural Network: Also Learn the Features!

- Training the deep neural network is just like logistic regression:

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

just  $w$  tends to be a much, much larger vector 😊

→ just run gradient ascent

+ stop when log likelihood of hold-out data starts to decrease

# How about computing all the derivatives?

- Derivatives tables:

$$\frac{d}{dx}(a) = 0$$

$$\frac{d}{dx}(x) = 1$$

$$\frac{d}{dx}(au) = a \frac{du}{dx}$$

$$\frac{d}{dx}(u + v - w) = \frac{du}{dx} + \frac{dv}{dx} - \frac{dw}{dx}$$

$$\frac{d}{dx}(uv) = u \frac{dv}{dx} + v \frac{du}{dx}$$

$$\frac{d}{dx}\left(\frac{u}{v}\right) = \frac{1}{v} \frac{du}{dx} - \frac{u}{v^2} \frac{dv}{dx}$$

$$\frac{d}{dx}(u^n) = nu^{n-1} \frac{du}{dx}$$

$$\frac{d}{dx}(\sqrt{u}) = \frac{1}{2\sqrt{u}} \frac{du}{dx}$$

$$\frac{d}{dx}\left(\frac{1}{u}\right) = -\frac{1}{u^2} \frac{du}{dx}$$

$$\frac{d}{dx}\left(\frac{1}{u^n}\right) = -\frac{n}{u^{n+1}} \frac{du}{dx}$$

$$\frac{d}{dx}[f(u)] = \frac{d}{du}[f(u)] \frac{du}{dx}$$

$$\frac{d}{dx}[\ln u] = \frac{d}{dx}[\log_e u] = \frac{1}{u} \frac{du}{dx}$$

$$\frac{d}{dx}[\log_a u] = \log_a e \frac{1}{u} \frac{du}{dx}$$

$$\frac{d}{dx}e^u = e^u \frac{du}{dx}$$

$$\frac{d}{dx}a^u = a^u \ln a \frac{du}{dx}$$

$$\frac{d}{dx}(u^v) = vu^{v-1} \frac{du}{dx} + \ln u \cdot u^v \frac{dv}{dx}$$

$$\frac{d}{dx} \sin u = \cos u \frac{du}{dx}$$

$$\frac{d}{dx} \cos u = -\sin u \frac{du}{dx}$$

$$\frac{d}{dx} \tan u = \sec^2 u \frac{du}{dx}$$

$$\frac{d}{dx} \cot u = -\csc^2 u \frac{du}{dx}$$

$$\frac{d}{dx} \sec u = \sec u \tan u \frac{du}{dx}$$

$$\frac{d}{dx} \csc u = -\csc u \cot u \frac{du}{dx}$$

# How about computing all the derivatives?

---

- But neural net  $f$  is never one of those?
  - No problem: CHAIN RULE:

If  $f(x) = g(h(x))$

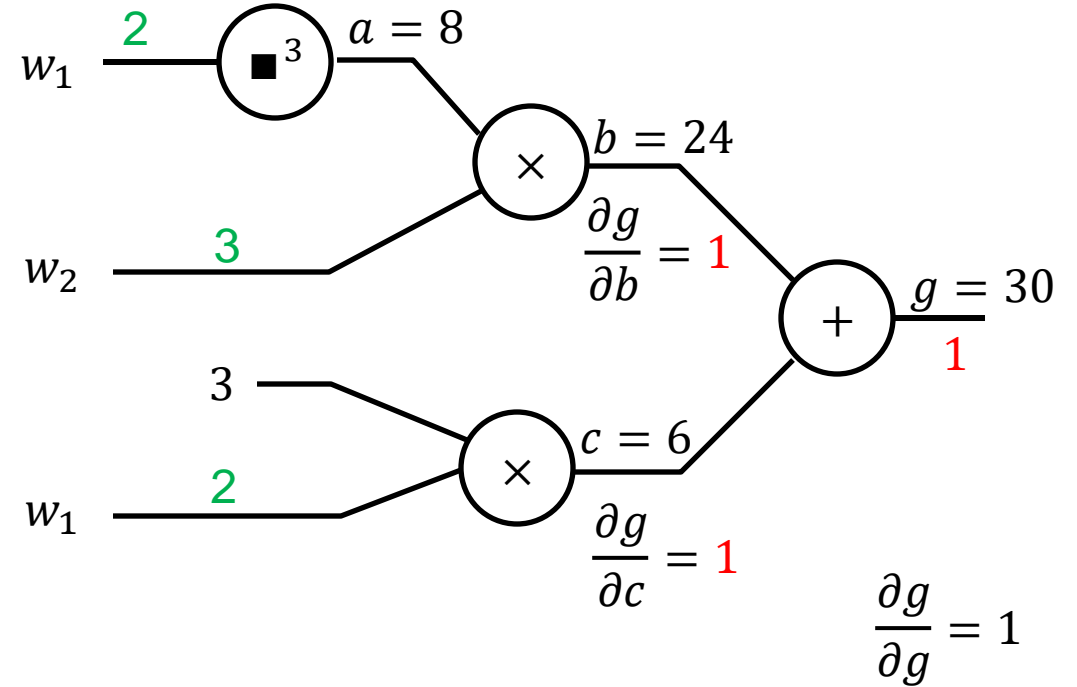
Then  $f'(x) = g'(h(x))h'(x)$

→ Derivatives can be computed by following well-defined procedures

# Back Propagation: $g(\mathbf{w}) = w_1^3 w_2 + 3w_1$

- Suppose we have  $g(\mathbf{w}) = w_1^3 w_2 + 3w_1$  and want the gradient at  $\mathbf{w} = [2, 3]$
- Think of the function as a composition of many functions, use chain rule.
  - Can use derivative chain rule to compute  $\partial g / \partial w_1$  and  $\partial g / \partial w_2$ .

- $g = b + c$ 
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$

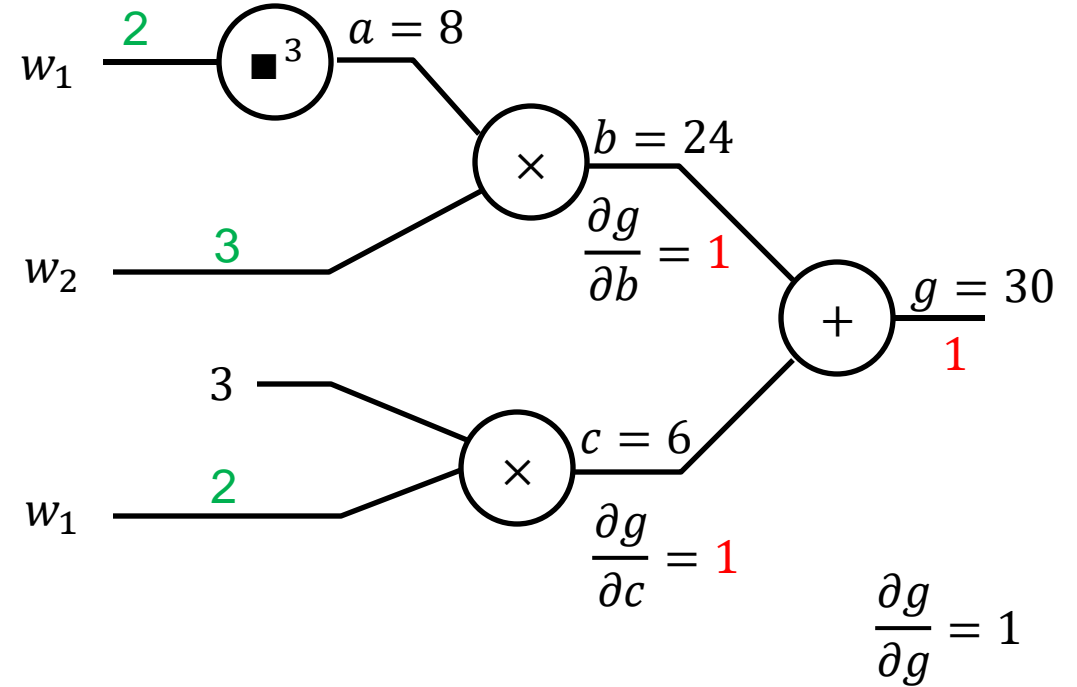




# Back Propagation: $g(\mathbf{w}) = w_1^3 w_2 + 3w_1$

- Suppose we have  $g(\mathbf{w}) = w_1^3 w_2 + 3w_1$  and want the gradient at  $\mathbf{w} = [2, 3]$
- Think of the function as a composition of many functions, use chain rule.
  - Can use derivative chain rule to compute  $\partial g / \partial w_1$  and  $\partial g / \partial w_2$ .

- $g = b + c$ 
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
- $b = a \times w_2$ 
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a}$

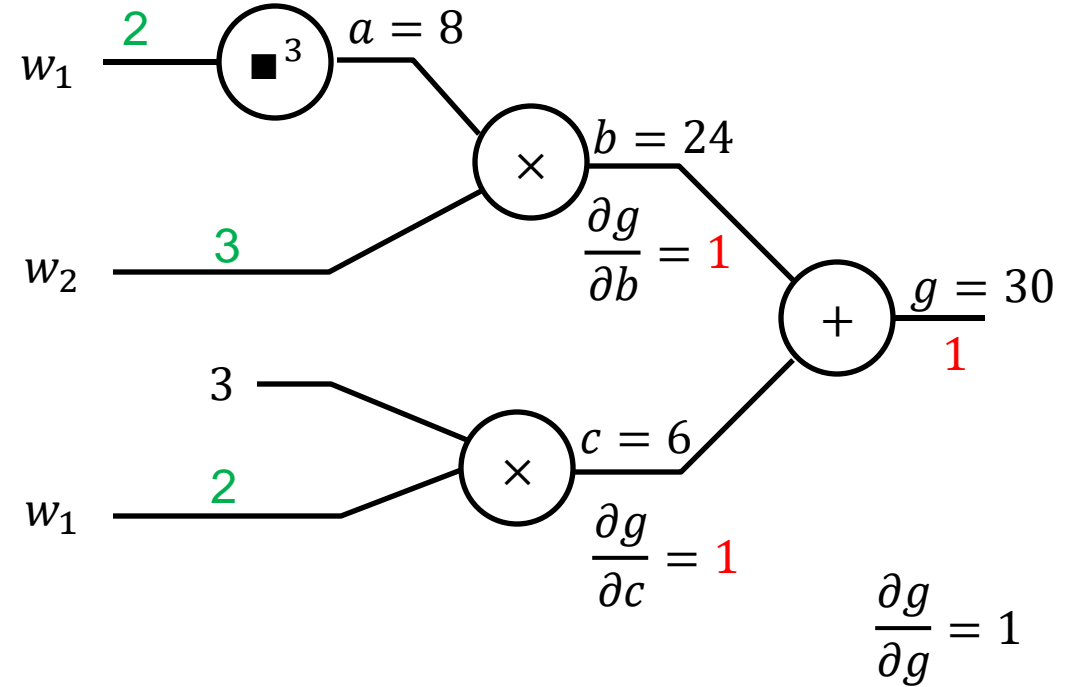


# Back Propagation: $g(\mathbf{w}) = w_1^3 w_2 + 3w_1$



- Suppose we have  $g(\mathbf{w}) = w_1^3 w_2 + 3w_1$  and want the gradient at  $\mathbf{w} = [2, 3]$
- Think of the function as a composition of many functions, use chain rule.
  - Can use derivative chain rule to compute  $\partial g / \partial w_1$  and  $\partial g / \partial w_2$ .

- $g = b + c$ 
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
- $b = a \times w_2$ 
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} = \text{?????}$



# Back Propagation: $g(\mathbf{w}) = w_1^3 w_2 + 3w_1$

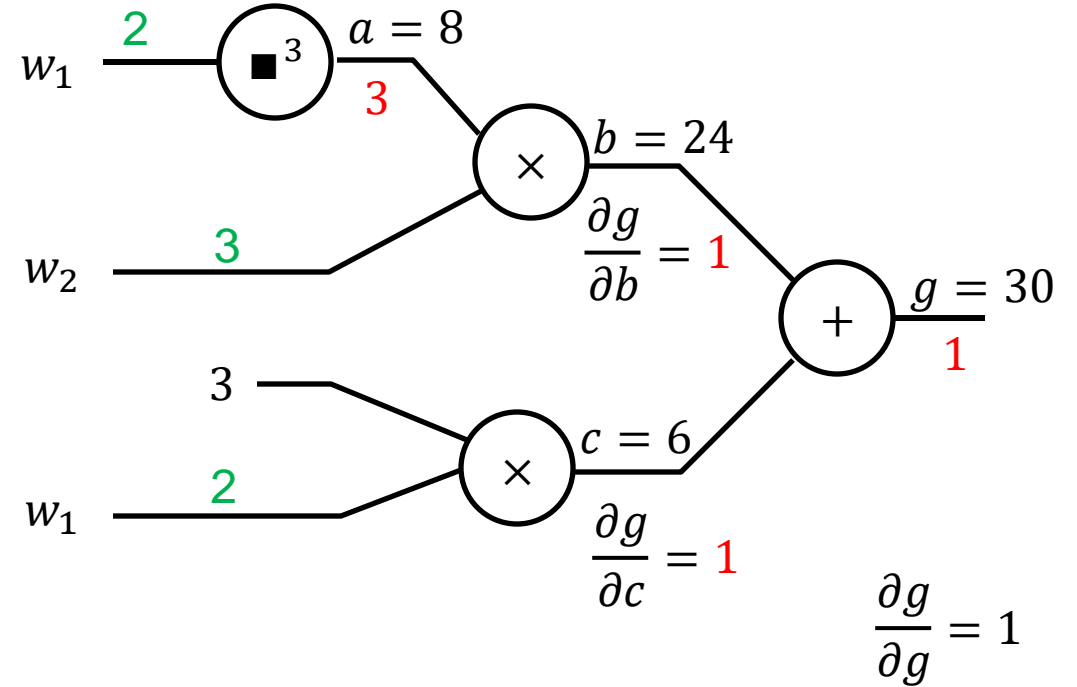
- Suppose we have  $g(\mathbf{w}) = w_1^3 w_2 + 3w_1$  and want the gradient at  $\mathbf{w} = [2, 3]$
- Think of the function as a composition of many functions, use chain rule.
  - Can use derivative chain rule to compute  $\partial g / \partial w_1$  and  $\partial g / \partial w_2$ .

- $g = b + c$

- $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$

- $b = a \times w_2$

- $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$



# Back Propagation: $g(\mathbf{w}) = w_1^3 w_2 + 3w_1$



- Suppose we have  $g(\mathbf{w}) = w_1^3 w_2 + 3w_1$  and want the gradient at  $\mathbf{w} = [2, 3]$
- Think of the function as a composition of many functions, use chain rule.
  - Can use derivative chain rule to compute  $\partial g / \partial w_1$  and  $\partial g / \partial w_2$ .

- $g = b + c$

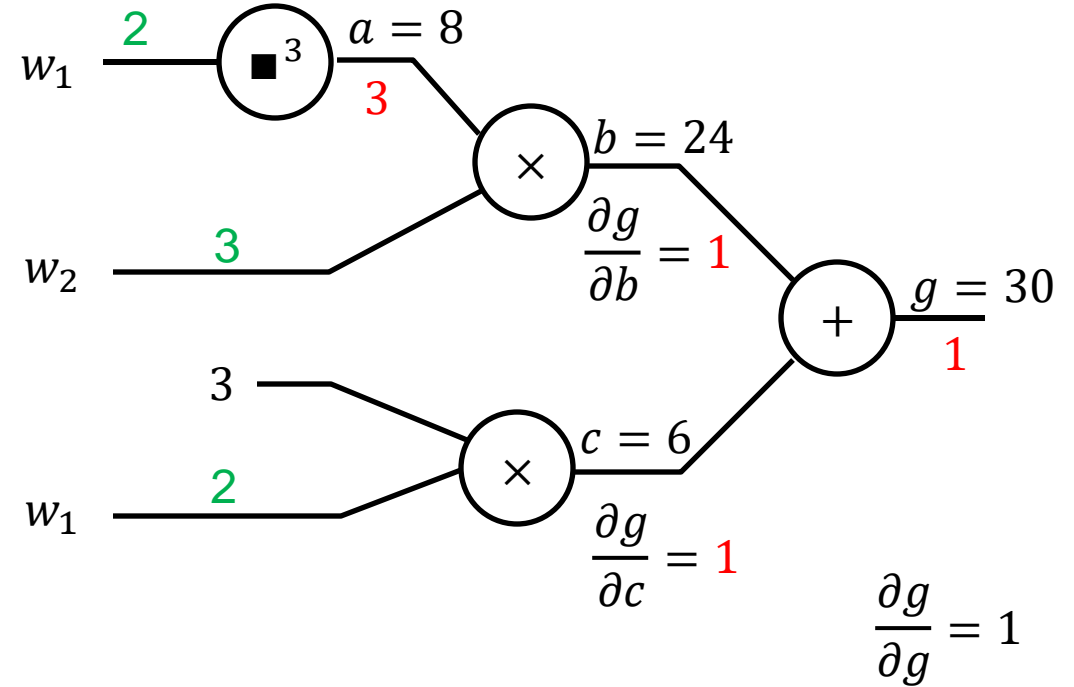
- $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$

- $b = a \times w_2$

- $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$

- $a = w_1^3$

- $\frac{\partial g}{\partial w_1} = \text{?????}$



# Back Propagation: $g(\mathbf{w}) = w_1^3 w_2 + 3w_1$

- Suppose we have  $g(\mathbf{w}) = w_1^3 w_2 + 3w_1$  and want the gradient at  $\mathbf{w} = [2, 3]$
- Think of the function as a composition of many functions, use chain rule.
  - Can use derivative chain rule to compute  $\partial g / \partial w_1$  and  $\partial g / \partial w_2$ .

- $g = b + c$

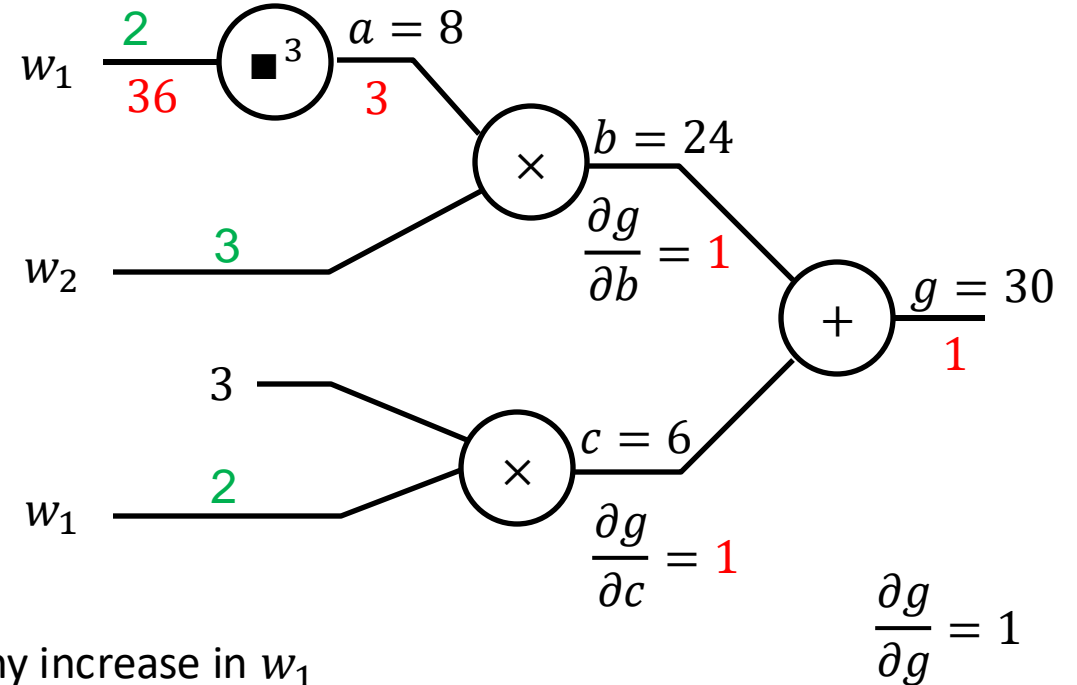
- $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$

- $b = a \times w_2$

- $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$

- $a = w_1^3$

- $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a} \frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36$



Interpretation: A tiny increase in  $w_1$  will result in an approximately  $36w_1$  increase in  $g$  due to this cube function.

# Back Propagation: $g(\mathbf{w}) = w_1^3 w_2 + 3w_1$



- Suppose we have  $g(\mathbf{w}) = w_1^3 w_2 + 3w_1$  and want the gradient at  $\mathbf{w} = [2, 3]$
- Think of the function as a composition of many functions, use chain rule.
  - Can use derivative chain rule to compute  $\partial g / \partial w_1$  and  $\partial g / \partial w_2$ .

- $g = b + c$

- $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$

- $b = a \times w_2$

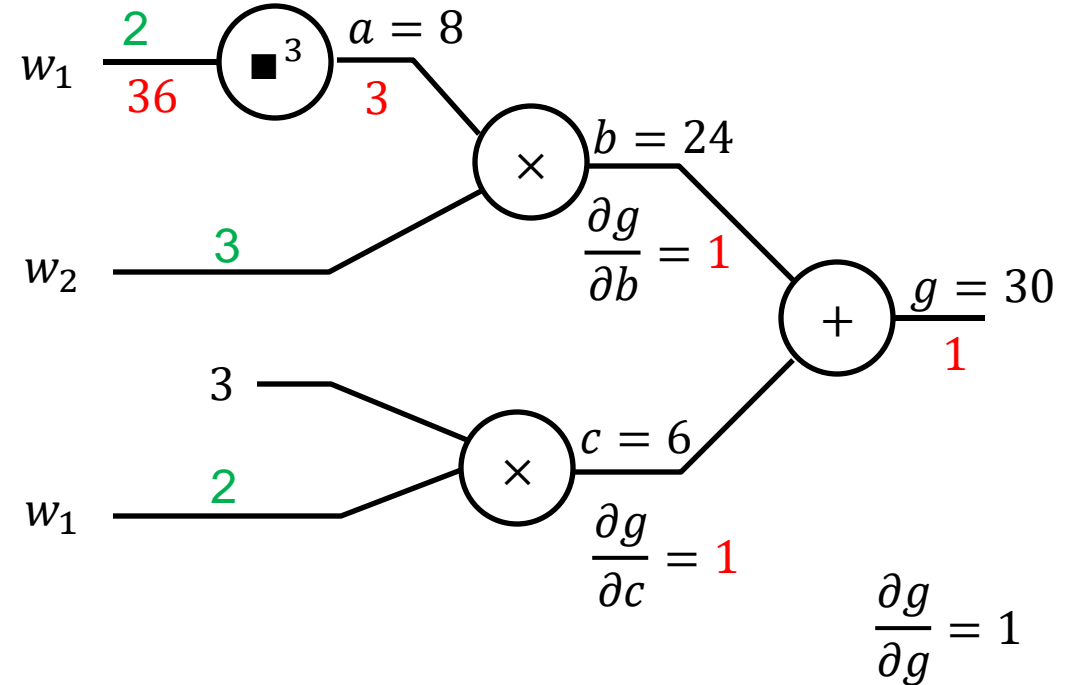
- $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$

- $a = w_1^3$

- $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a} \frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36$

- $\frac{\partial g}{\partial w_2} = ???$

Hint:  $b = a \times 3$  may be useful.



# Back Propagation: $g(\mathbf{w}) = w_1^3 w_2 + 3w_1$

- Suppose we have  $g(\mathbf{w}) = w_1^3 w_2 + 3w_1$  and want the gradient at  $\mathbf{w} = [2, 3]$
- Think of the function as a composition of many functions, use chain rule.
  - Can use derivative chain rule to compute  $\partial g / \partial w_1$  and  $\partial g / \partial w_2$ .

- $g = b + c$

- $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$

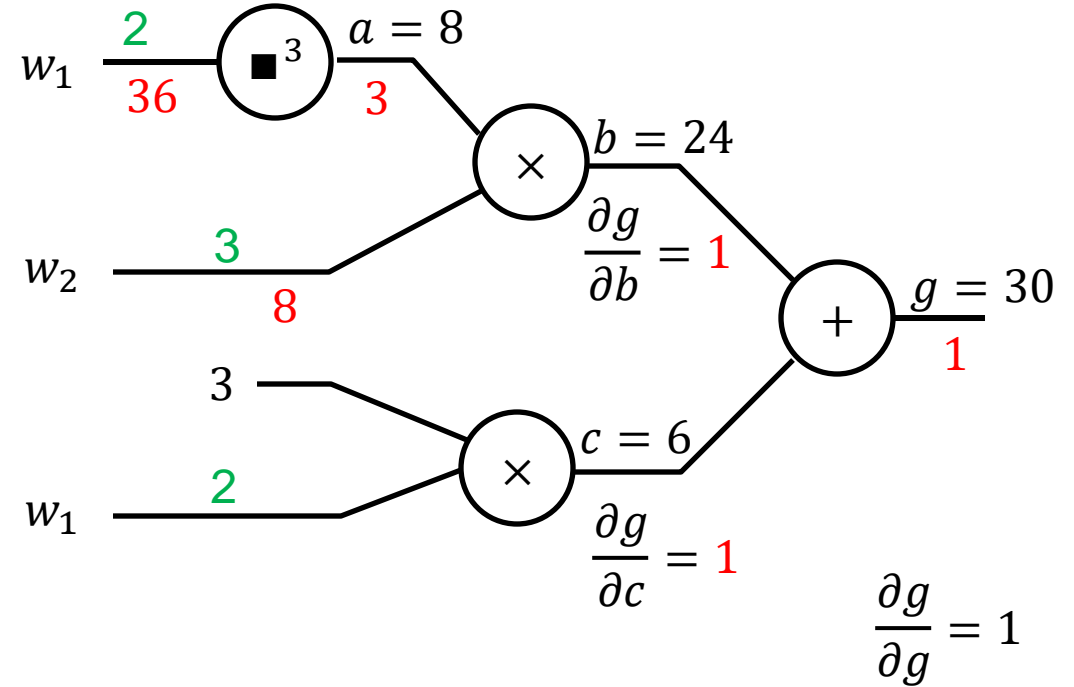
- $b = a \times w_2$

- $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$

- $\frac{\partial g}{\partial w_2} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial w_2} = 1 \frac{\partial b}{\partial w_2} = 1 \cdot 8 = 8$

- $a = w_1^3$

- $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a} \frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36$



# Back Propagation: $g(\mathbf{w}) = w_1^3 w_2 + 3w_1$

- Suppose we have  $g(\mathbf{w}) = w_1^3 w_2 + 3w_1$  and want the gradient at  $\mathbf{w} = [2, 3]$
- Think of the function as a composition of many functions, use chain rule.

- $g = b + c$

- $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$

- $b = a \times w_2$

- $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$

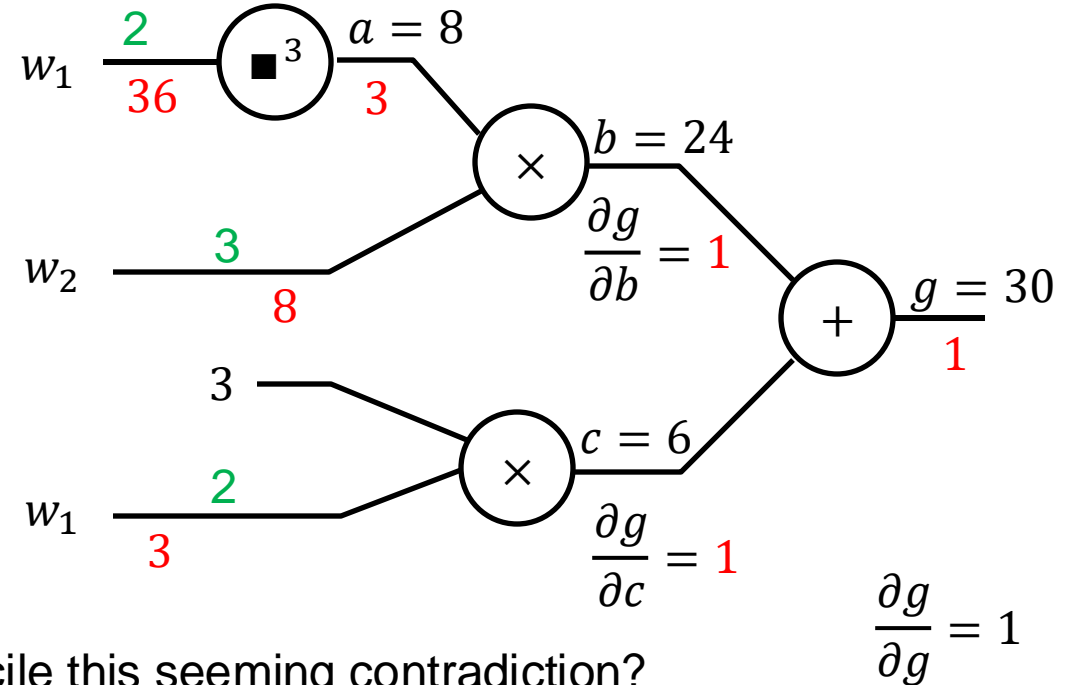
- $\frac{\partial g}{\partial w_2} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial w_2} = 1 \frac{\partial b}{\partial w_2} = 1 \cdot 8 = 8$

- $a = w_1^3$

- $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a} \frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36$

- $c = 3w_1$

- $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial c} \frac{\partial c}{\partial w_1} = 1 \cdot 3 = 3$



How do we reconcile this seeming contradiction?  
 Top partial derivative means cube function contributes  $36w_1$  and bottom p.d. means product contributes  $3w_1$  so add them.

$$\frac{\partial g}{\partial g} = 1$$



# Back Propagation: $g(\mathbf{w}) = w_1^3 w_2 + 3w_1$

- Suppose we have  $g(\mathbf{w}) = w_1^3 w_2 + 3w_1$  and want the gradient at  $\mathbf{w} = [2, 3]$
- Think of the function as a composition of many functions, use chain rule.

- $g = b + c$

- $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$

- $b = a \times w_2$

- $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$

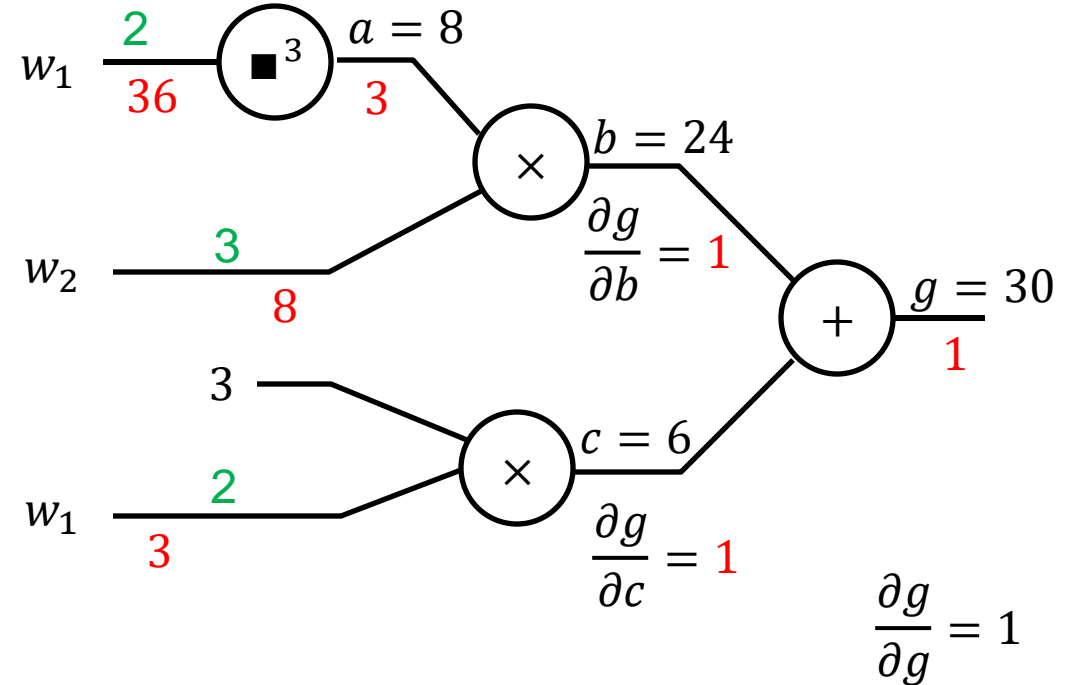
- $\frac{\partial g}{\partial w_2} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial w_2} = 1 \frac{\partial b}{\partial w_2} = 1 \cdot 8 = 8$

- $a = w_1^3$

- $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a} \frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36$

- $c = 3w_1$

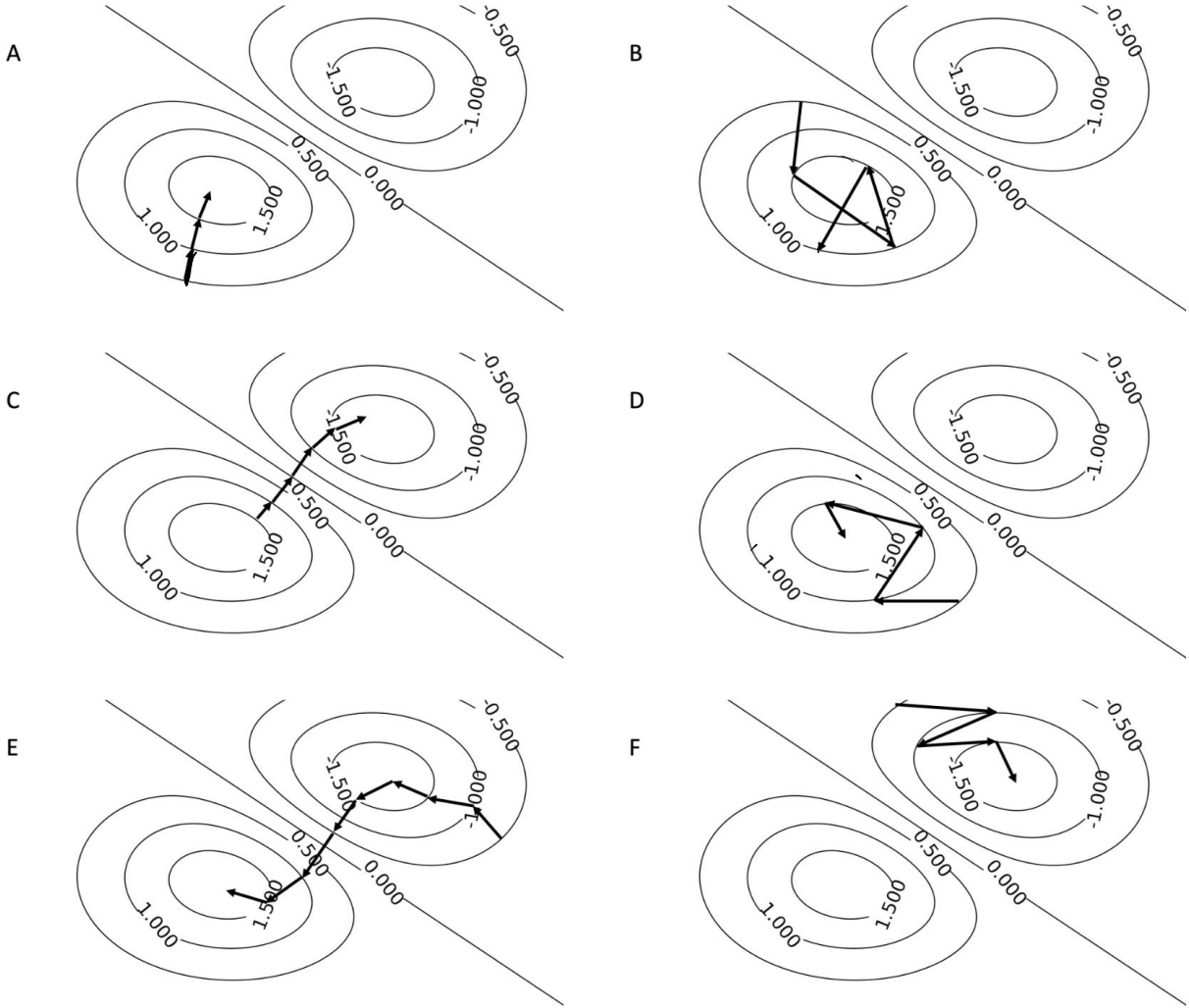
- $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial c} \frac{\partial c}{\partial w_1} = 1 \cdot 3 = 3$



$$\nabla g = \left[ \frac{\partial g}{\partial w_1}, \frac{\partial g}{\partial w_2} \right] = [39, 8]$$

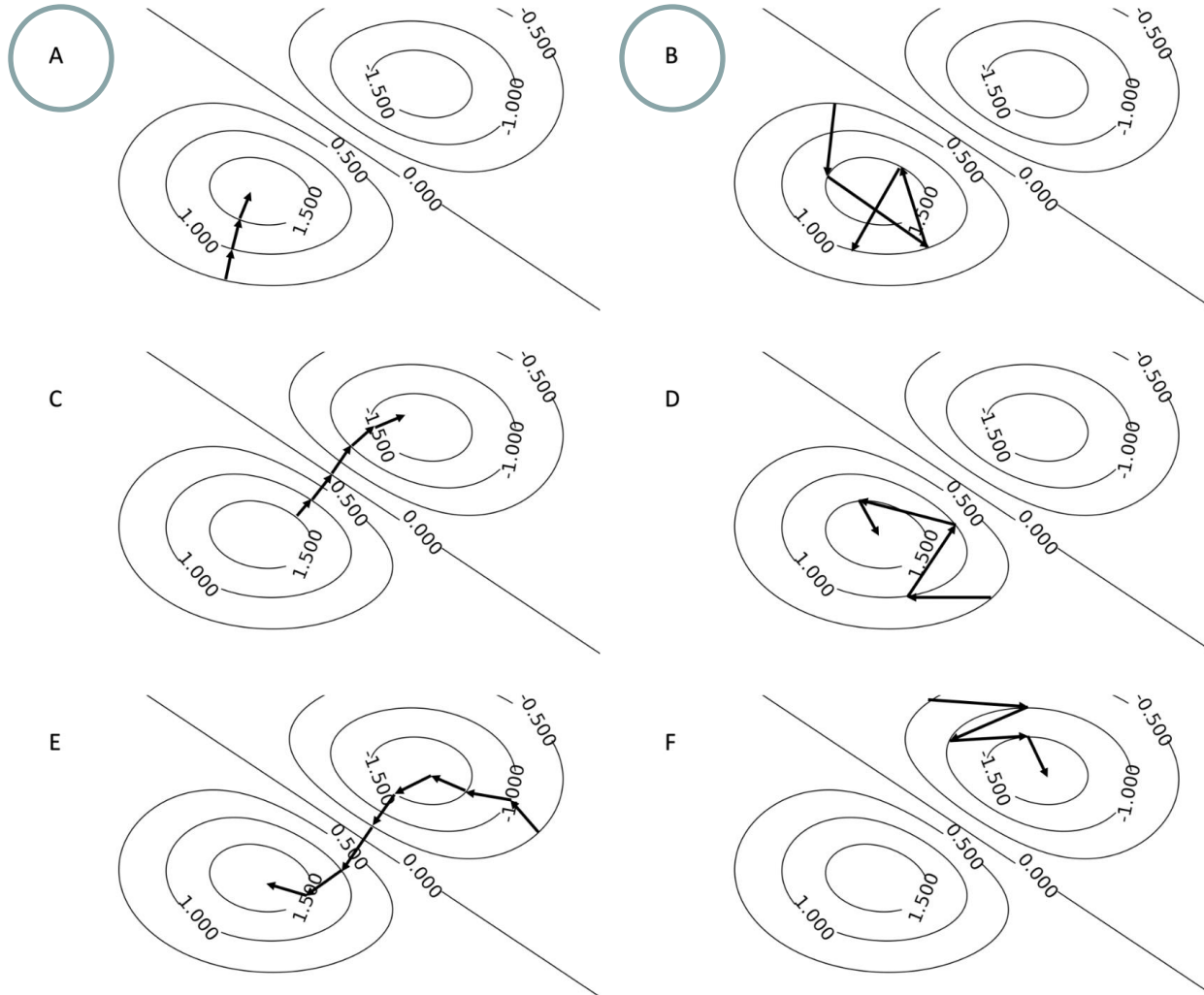
# Exercises: Gradient Ascent

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



# Exercises: Gradient Ascent

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



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.

# Summary of Key Ideas

- Optimize probability of label given input  $\max_w ll(w) = \max_w \sum_i \log P(y^{(i)} | x^{(i)}; w)$
- Continuous optimization
  - Gradient ascent:
    - Compute steepest uphill direction = gradient (= just vector of partial derivatives)
    - Take step in the gradient direction
    - Repeat (until held-out data accuracy starts to drop = “early stopping”)
- Deep neural nets
  - Last layer = still logistic regression
  - Now also many more layers before this last layer
    - = computing the features
    - → the features are learned rather than hand-designed
  - Automatic differentiation gives the derivatives efficiently (how? = outside of scope of 343)

# Optimization Procedure: Gradient Ascent

```
■ init  $w$   
■ for  $iter = 1, 2, \dots$   
 $w \leftarrow w + \alpha * \nabla g(w)$ 
```

- $\alpha$ : 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 %

# Batch Gradient Ascent on the Log Likelihood Objective

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

- `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

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

**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

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

**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

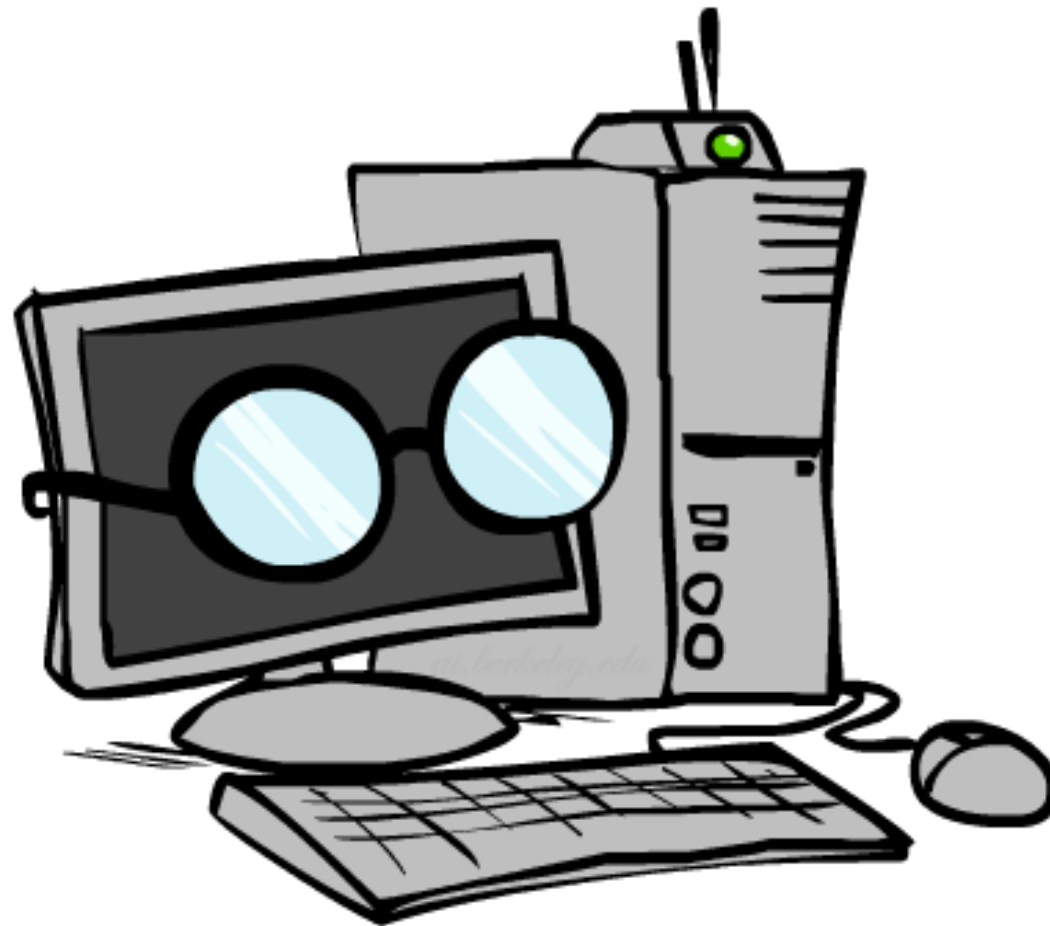
- `init w`
- `for iter = 1, 2, ...`
  - pick random subset of training examples  $J$

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

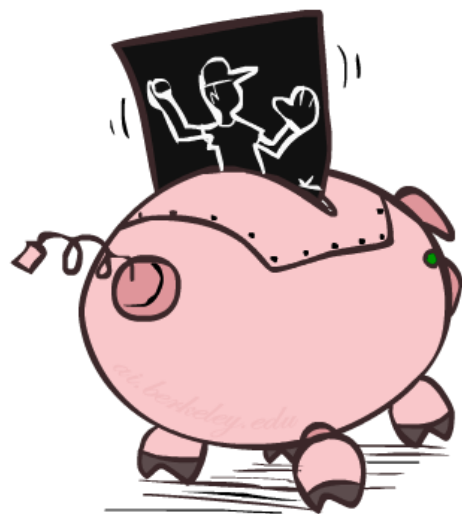
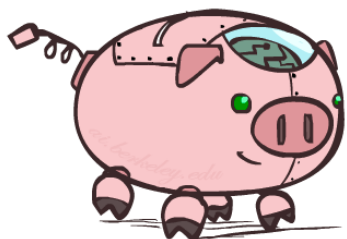


# Computer Vision

---



# Manual Feature Design

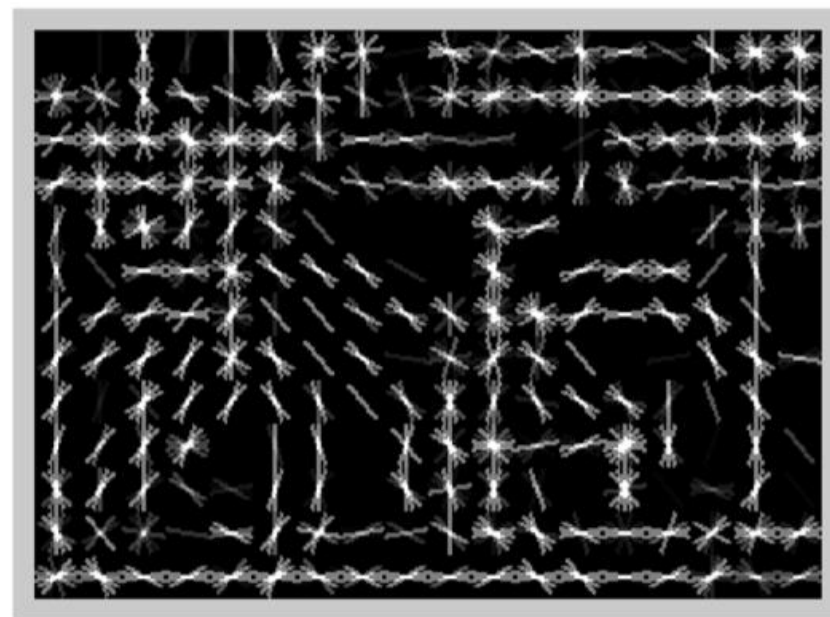


# Features and Generalization

---



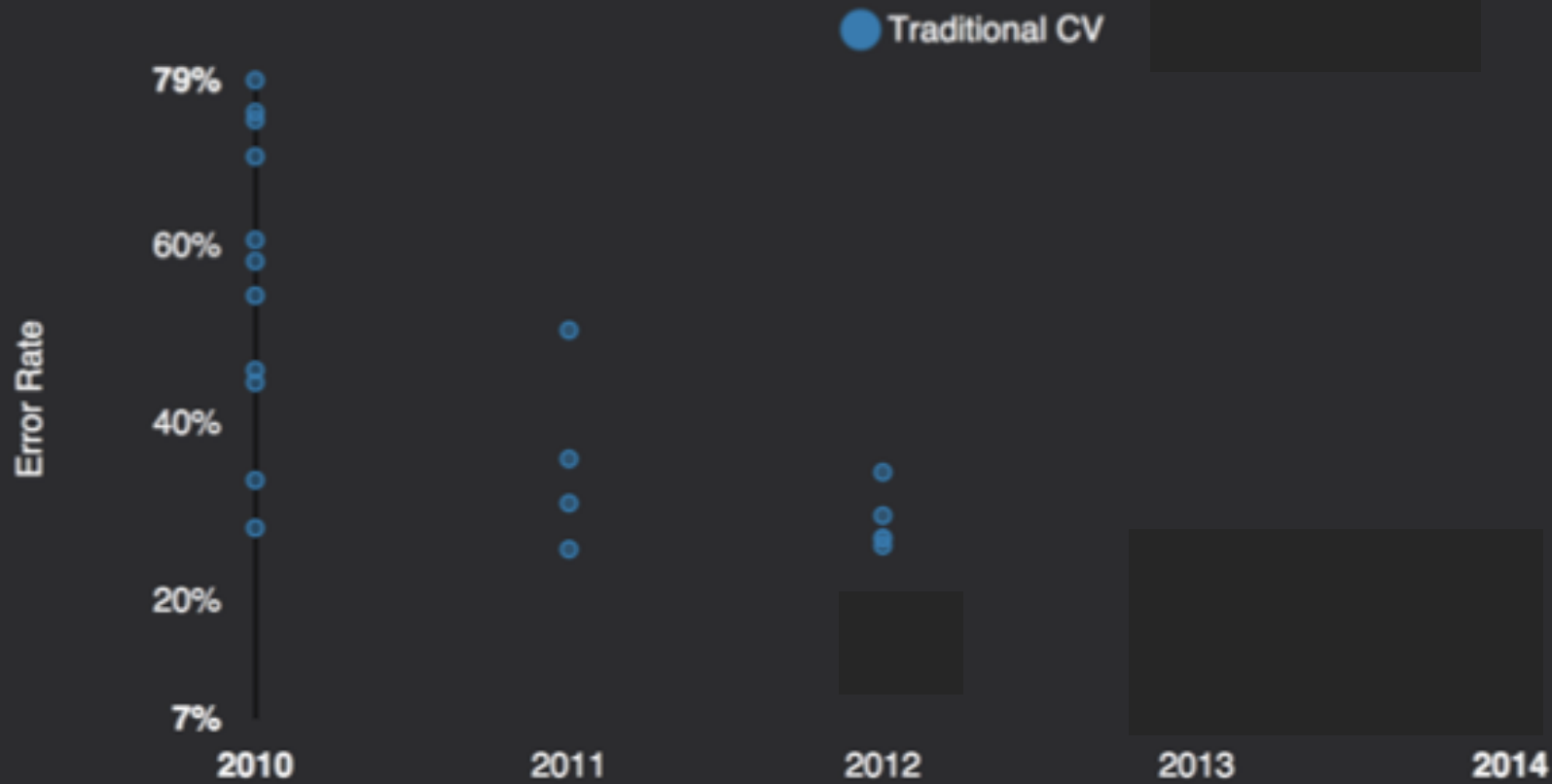
Image



HoG

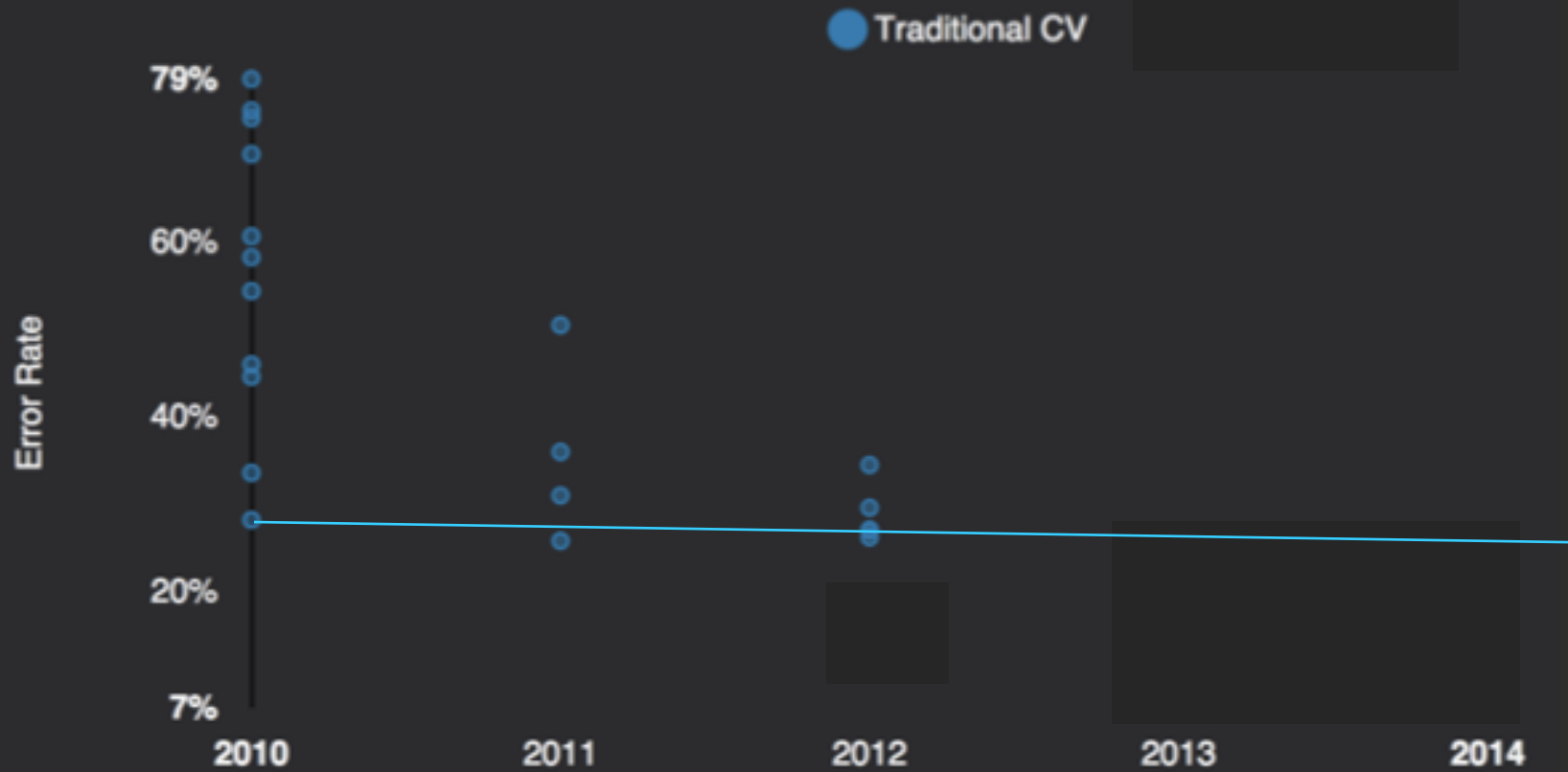
# Performance

## ImageNet Error Rate 2010-2014



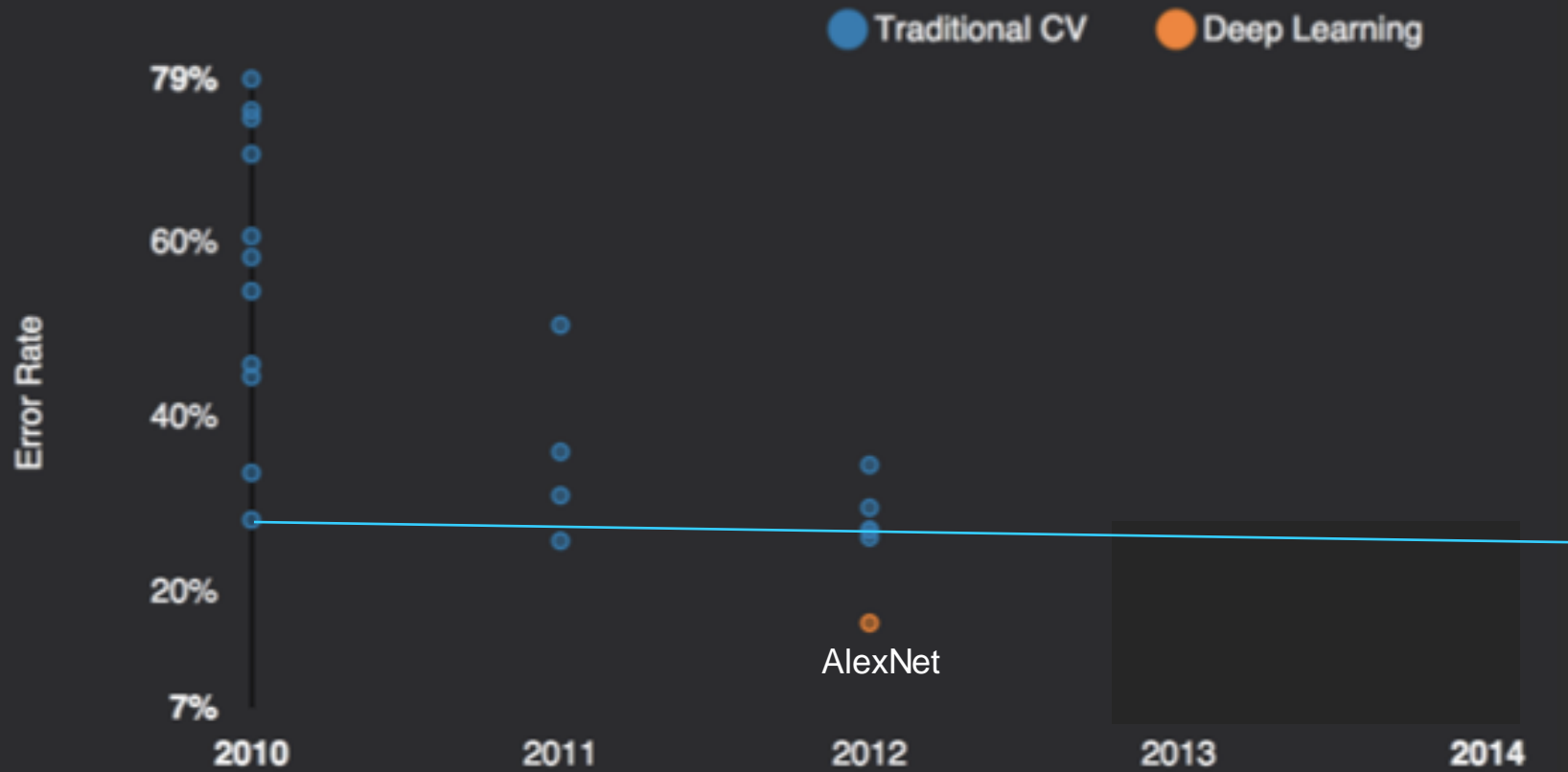
# Performance

## ImageNet Error Rate 2010-2014



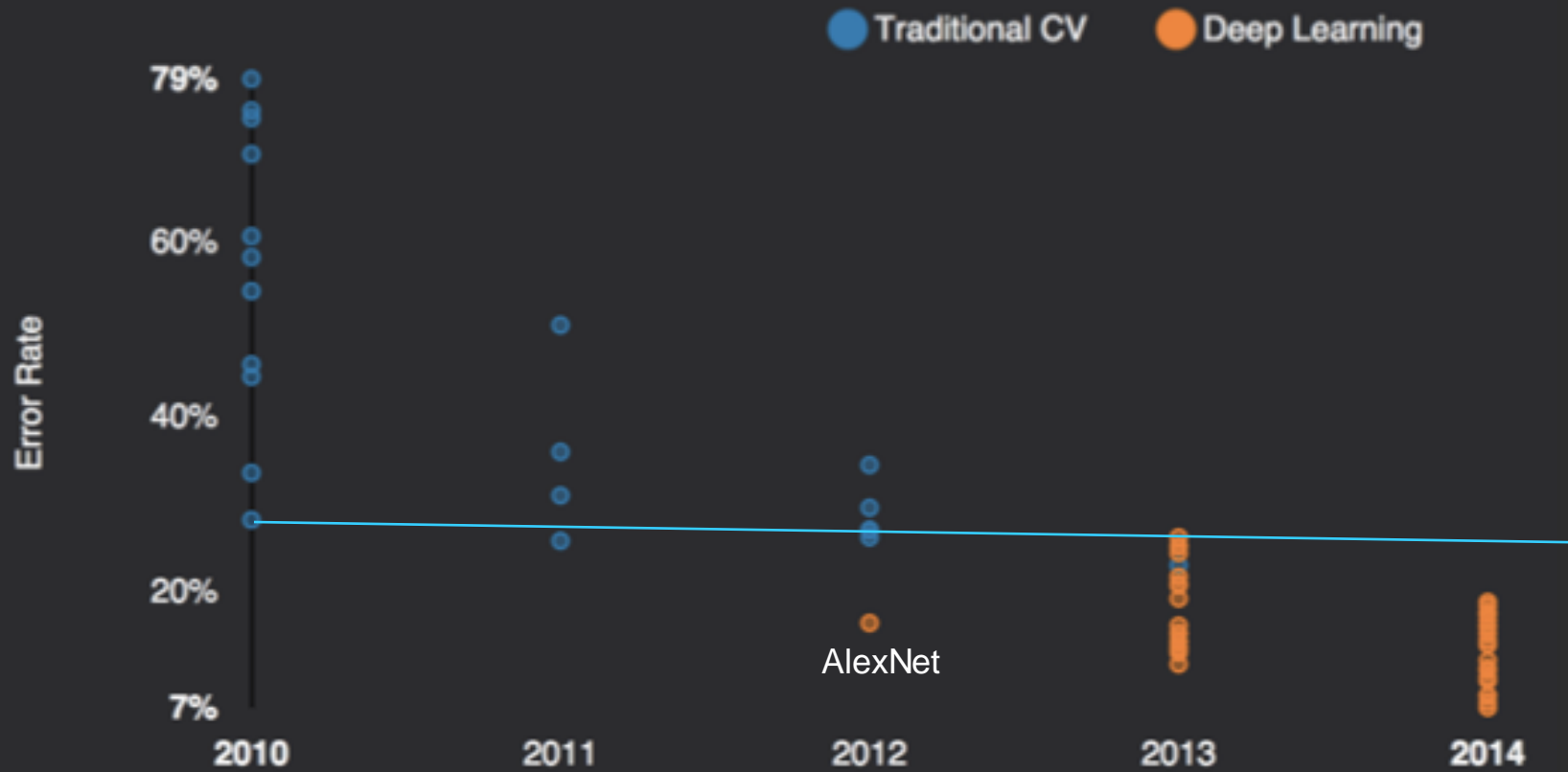
# Performance

## ImageNet Error Rate 2010-2014



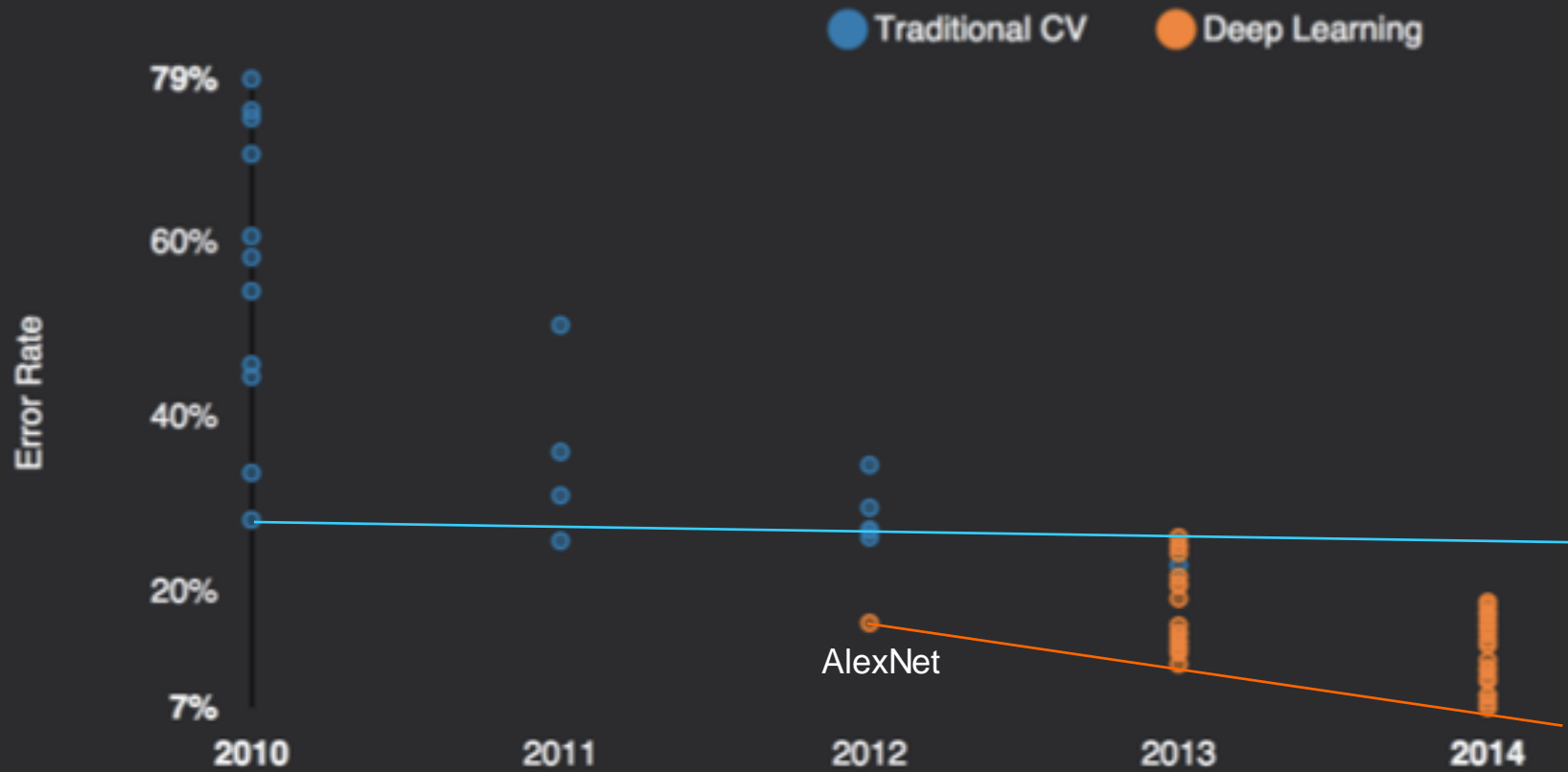
# Performance

## ImageNet Error Rate 2010-2014



# Performance

## ImageNet Error Rate 2010-2014





# Machine Translation

Google Neural Machine Translation (in production)

