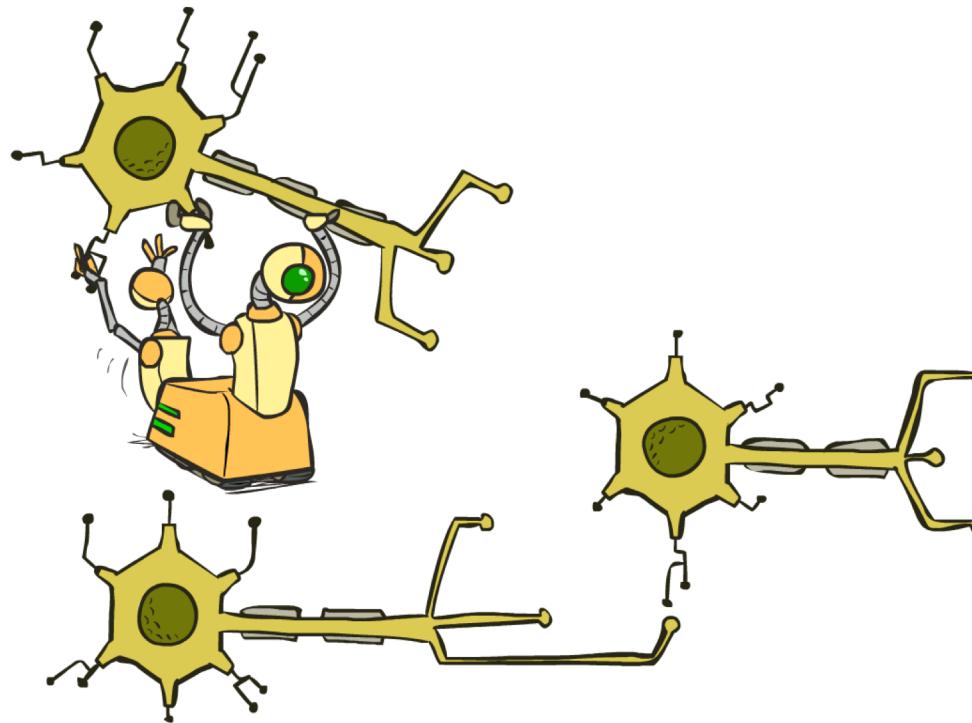


# CS 188: Artificial Intelligence

## Optimization and Neural Nets

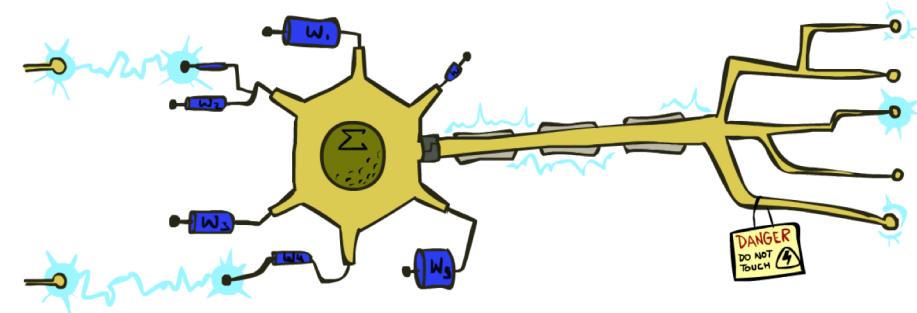


Instructors: Pieter Abbeel and Dan Klein --- University of California, Berkeley

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at <http://ai.berkeley.edu>.]

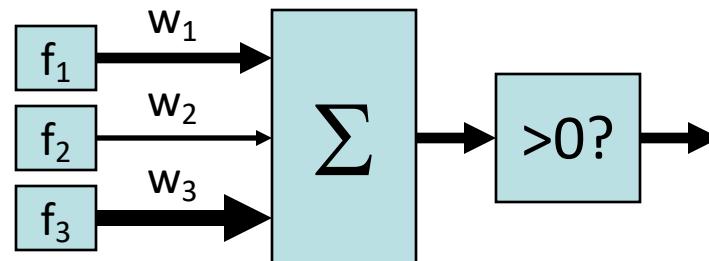
# Reminder: Linear Classifiers

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



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

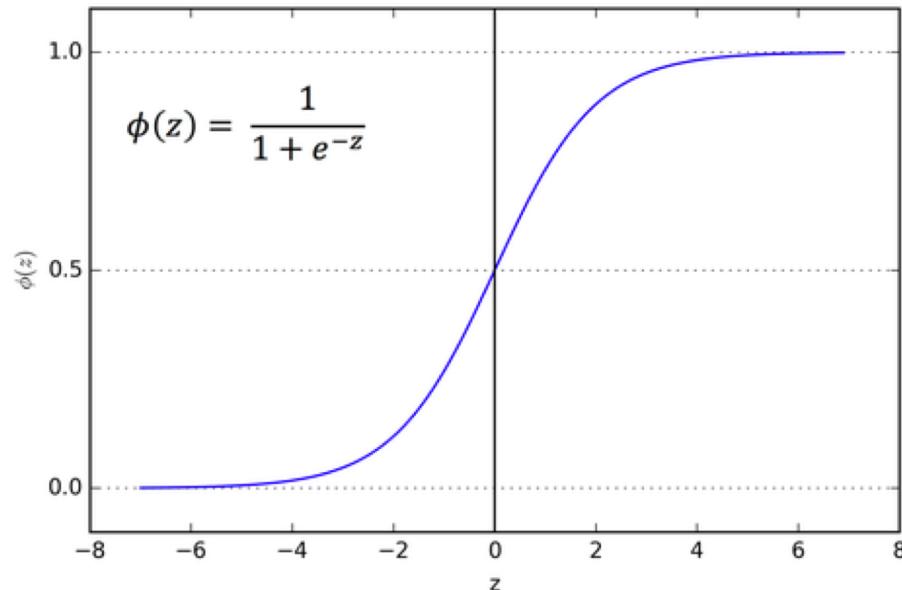
- If the activation is:
  - Positive, output +1
  - Negative, output -1



# 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:

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

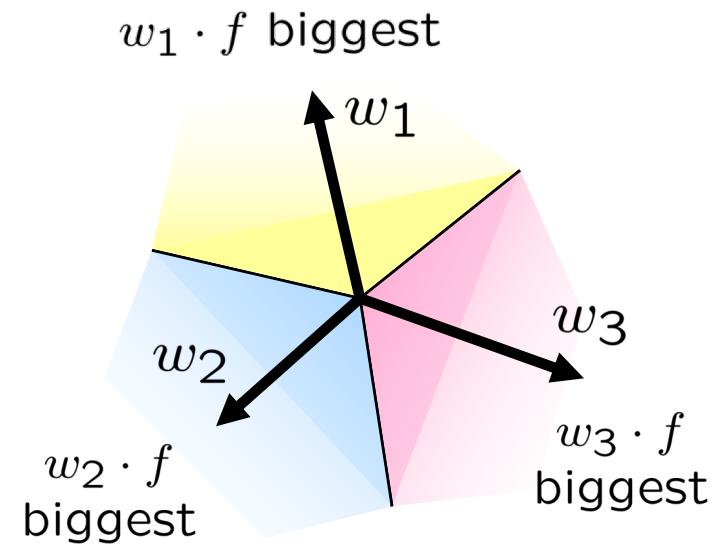
$$P(y^{(i)} = -1 | x^{(i)}; w) = 1 - \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

= 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?

# Best w?

---

- Maximum likelihood estimation:

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

with:

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

= Multi-Class Logistic Regression

# This Lecture

---

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

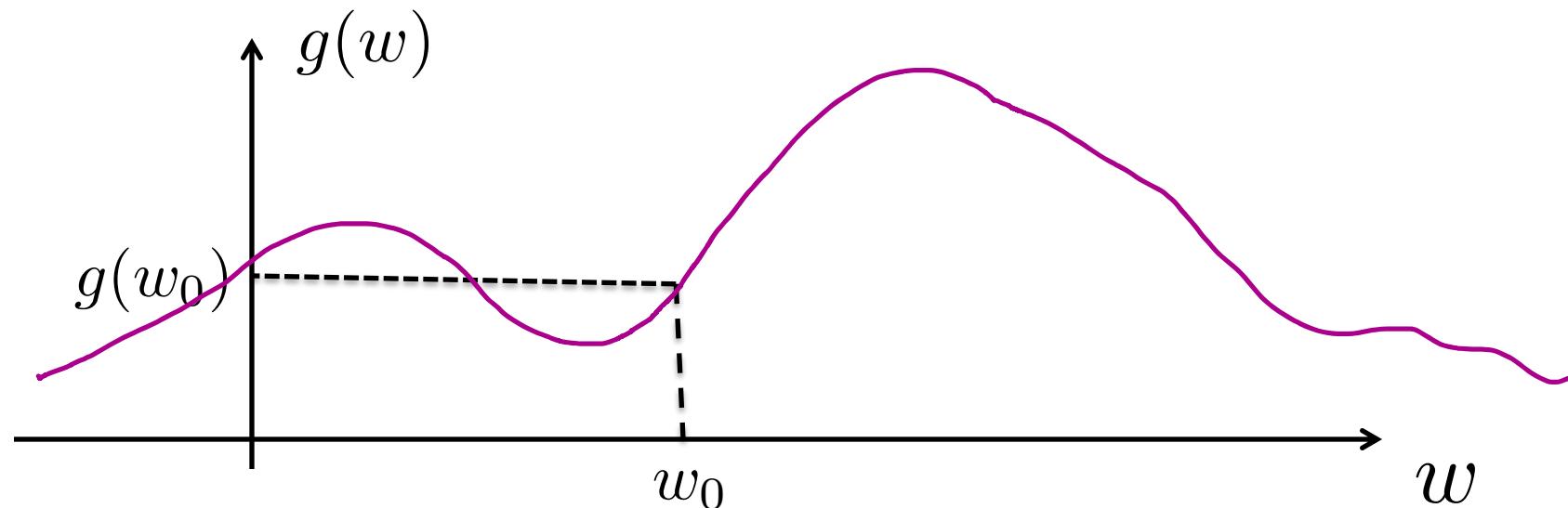
$$\max_w \text{ll}(w) = \max_w \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

# 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?



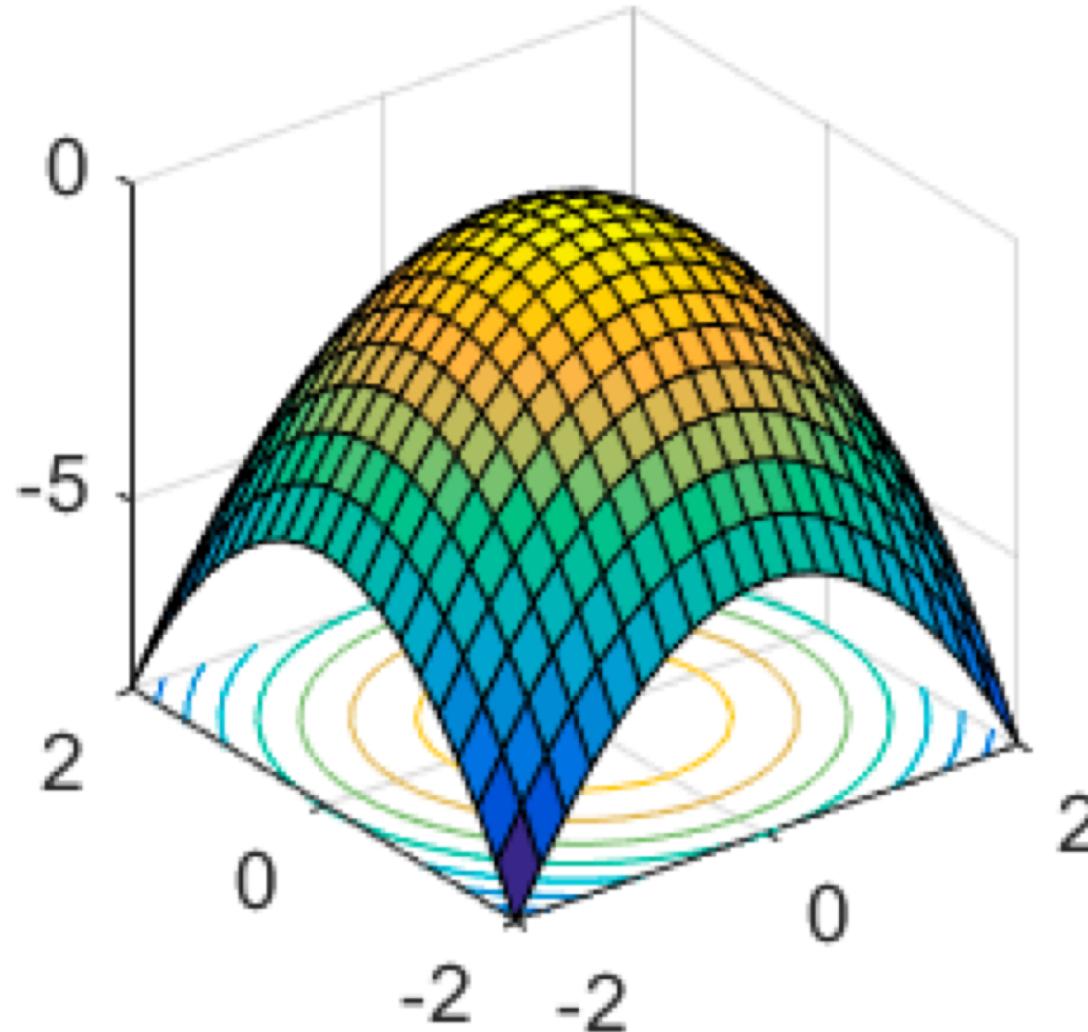
# 1-D Optimization



- 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:

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

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

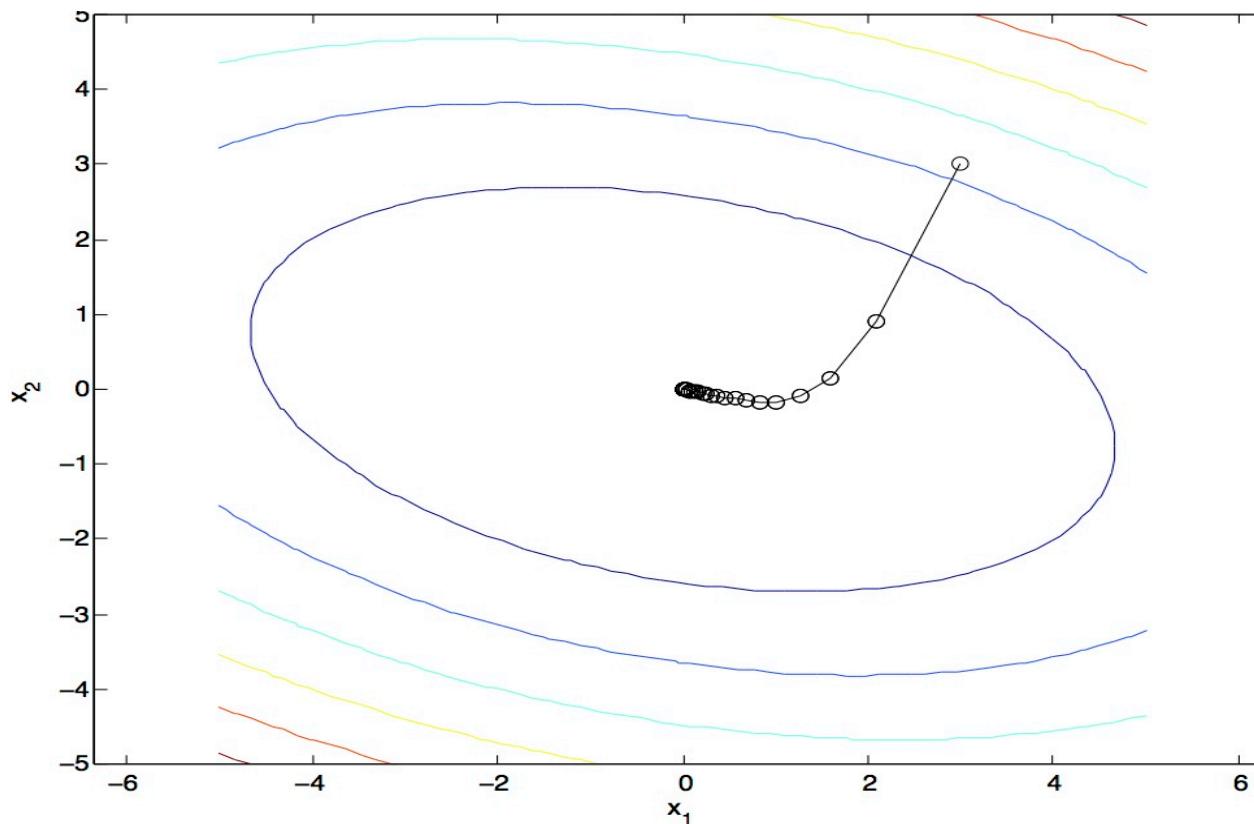
- Updates in vector notation:

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

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



# 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, ...`

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

- $\alpha$ : 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

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

- `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  $J$

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