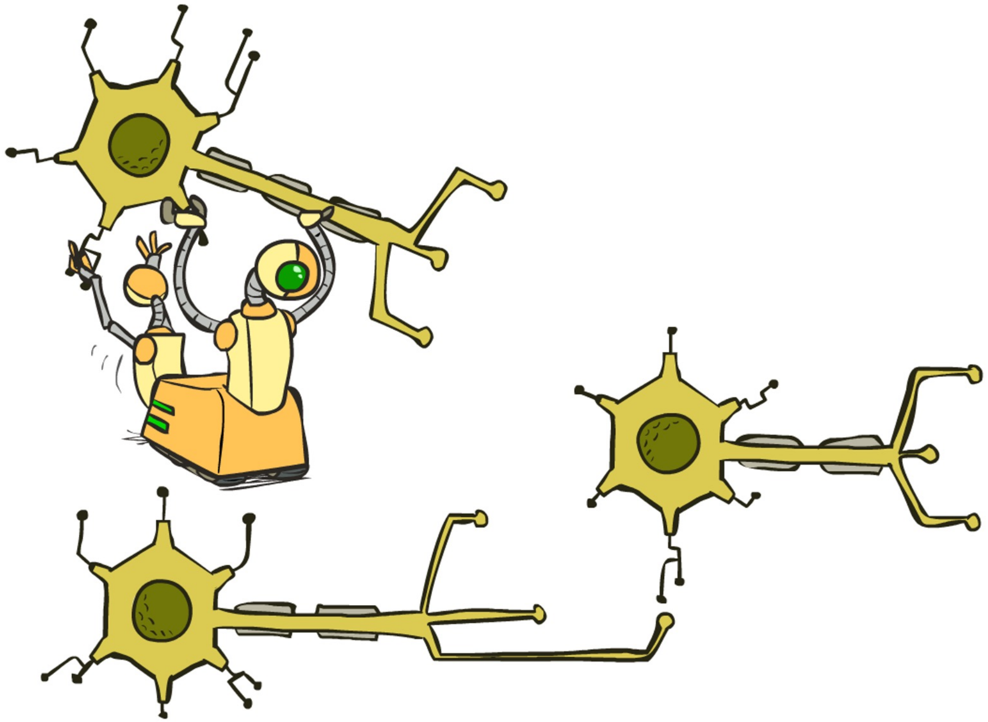
CS 188: Artificial Intelligence

Optimization and Neural Nets

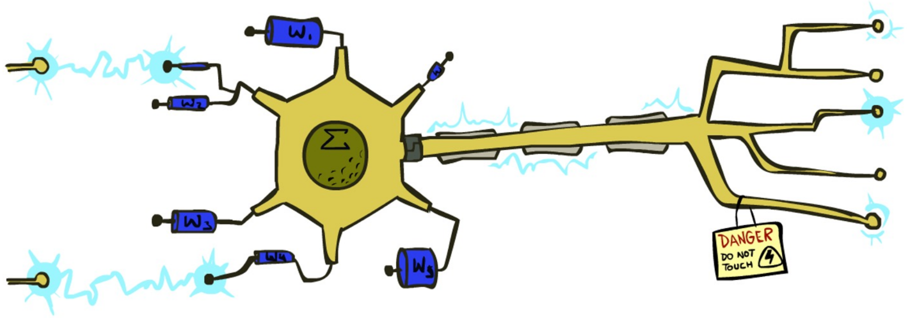


Summer 2024: Eve Fleisig & Evgeny Pobachienko

[Slides drawn from Nicholas Tomlin, Dan Klein, and Pieter Abbeel for CS188 at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

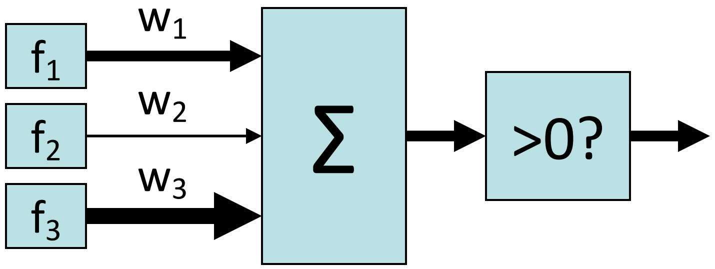
Reminder: Linear Classifiers

- 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

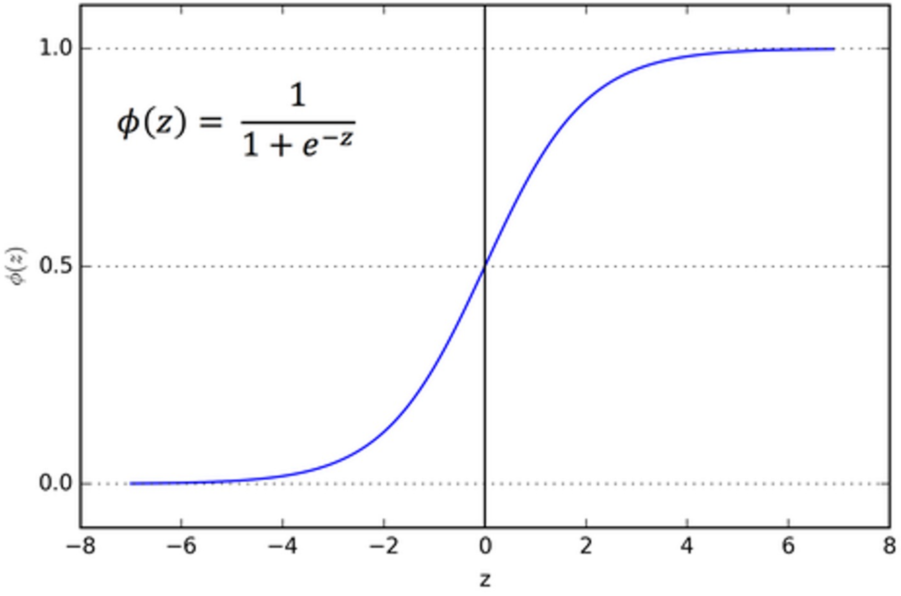


How to get probabilistic decisions?

- Activation: $z = w \cdot f(x)$
- If $z = w \cdot f(x)$ very positive: 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:

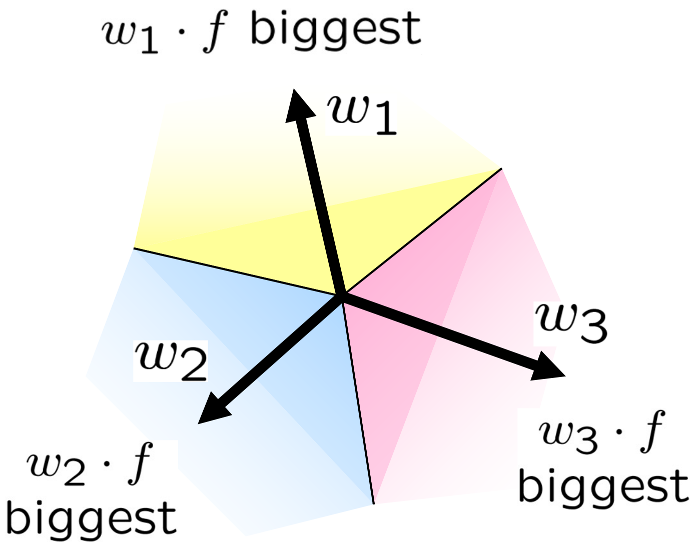
$$P(y^{(i)} = +1|x^{(i)}; w) = \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

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

= 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:

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

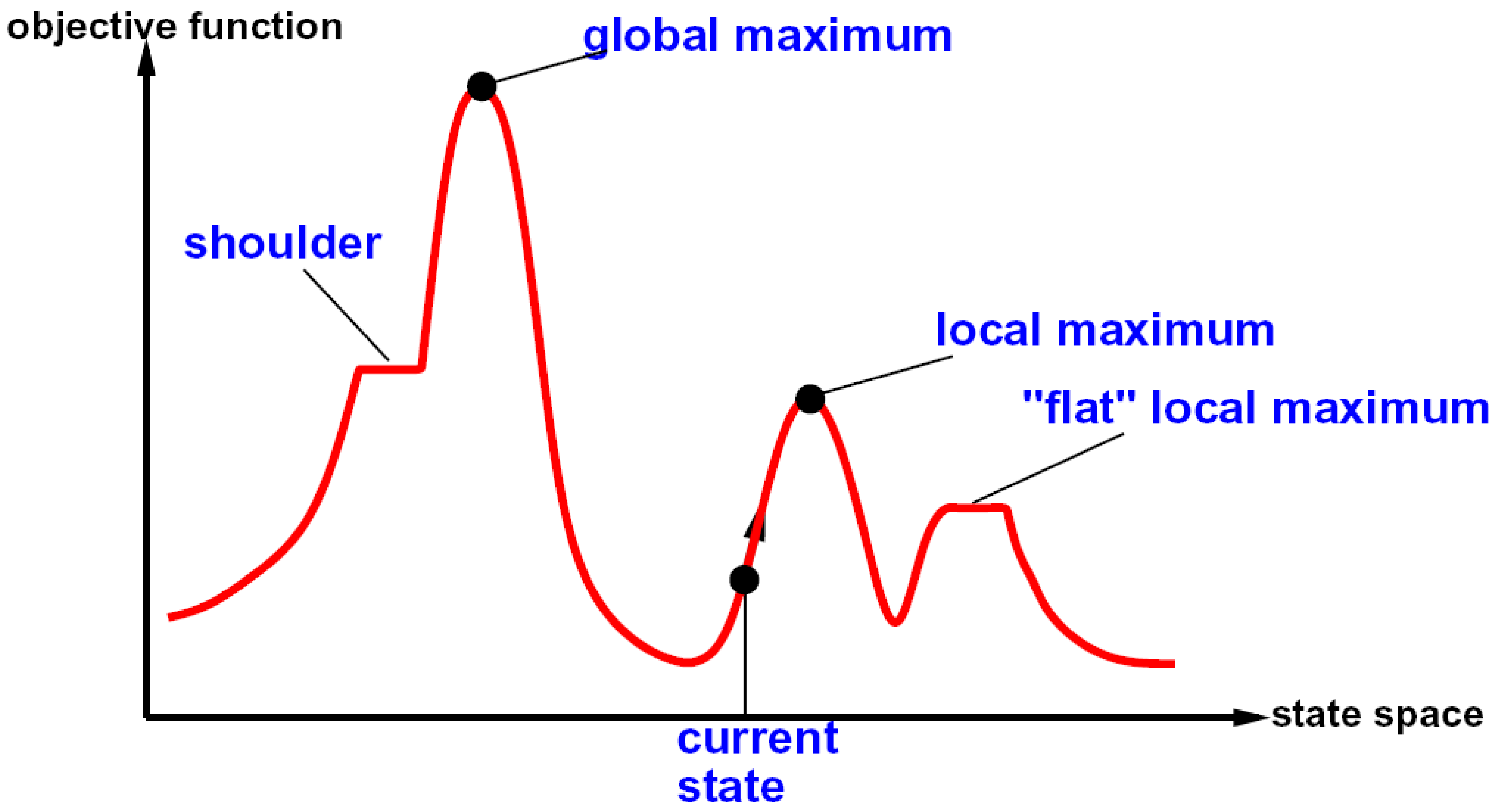
= Multi-Class Logistic Regression

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 Diagram



Required Mathematics Background

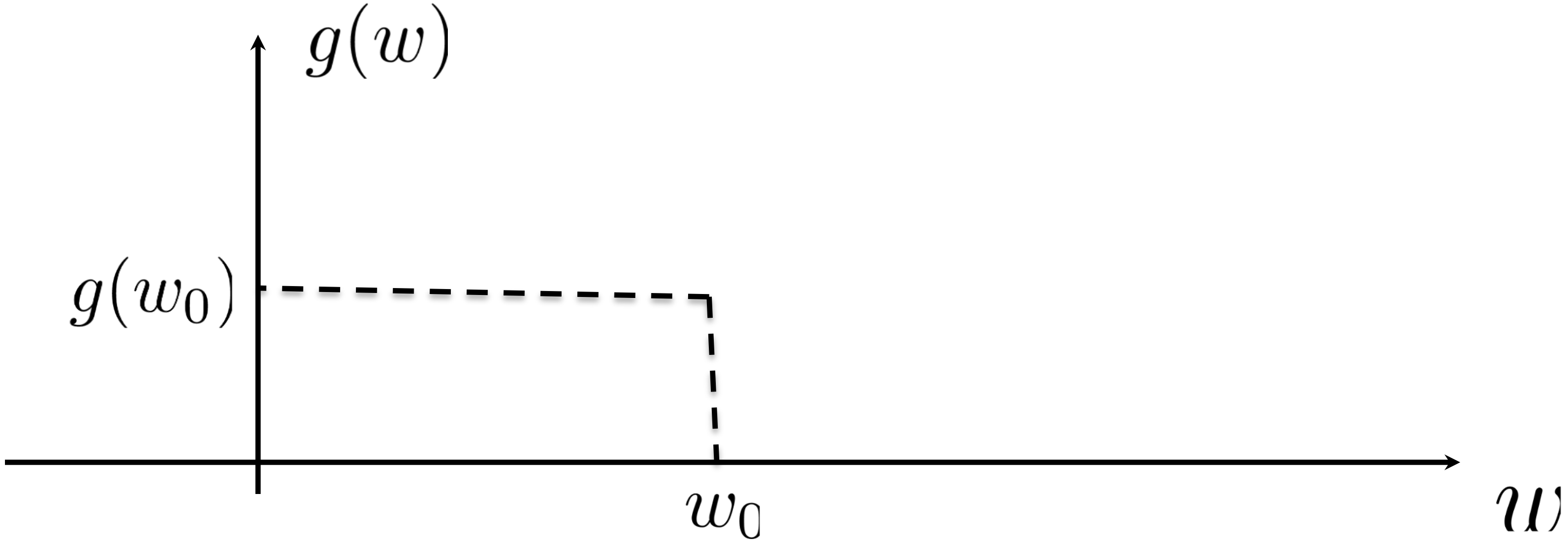
- Linear algebra:
  - Definition and properties of dot products
  - Composition of linear transformations is linear
- Vector calculus:
  - How to take partial derivatives (incl. chain rule, vector derivatives)
  - Solving optimization problems using derivatives (e.g., deriving MLE)
  - Taylor expansion (used in lecture; non-examinable)
- Probability: definition of a probability distribution, random variables, joint and marginal distributions, conditional probabilities, Bayes' rule, normalization

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

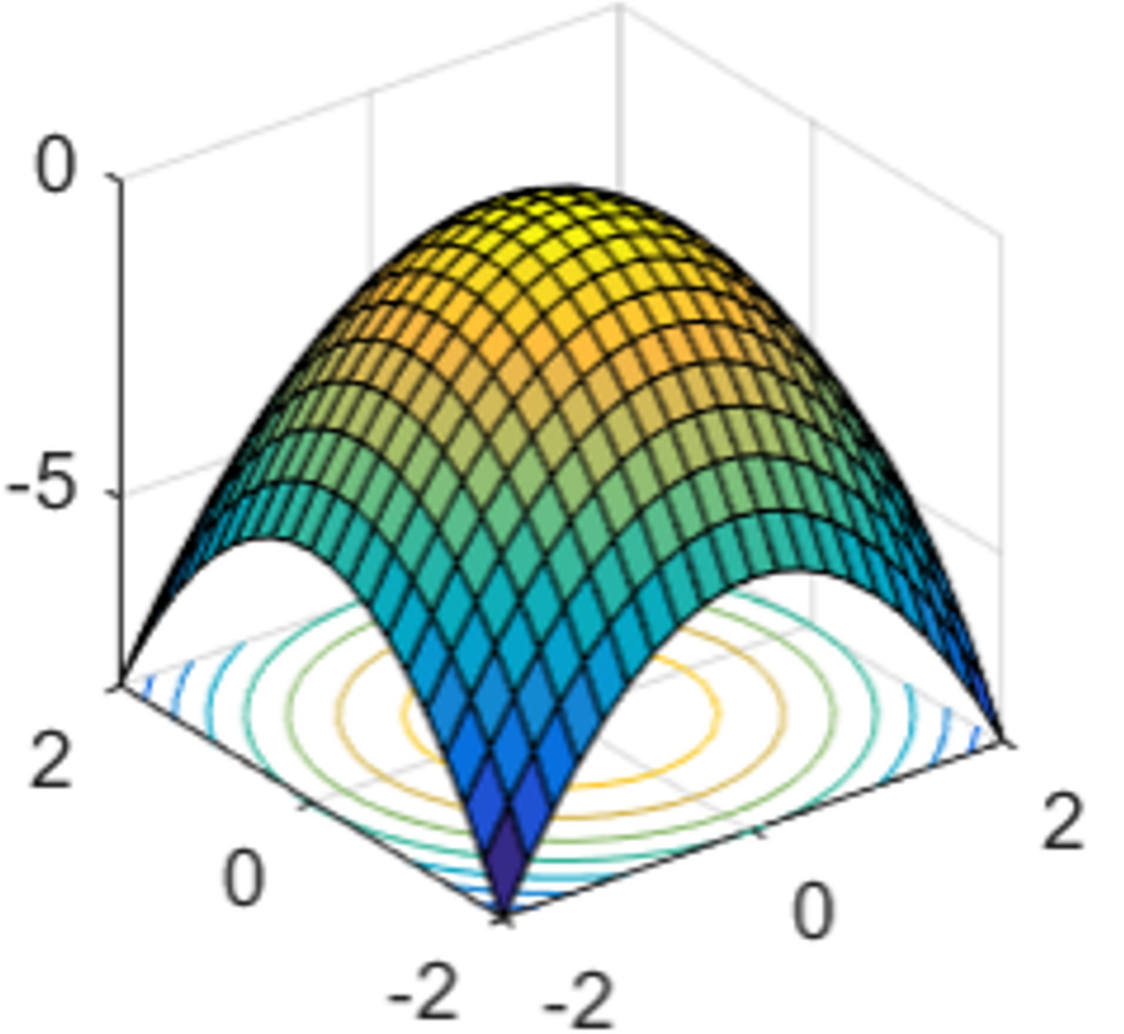
- What's particularly tricky when hill-climbing for multiclass logistic regression?
  - Optimization over a continuous space
    - Infinitely many neighbors!
    - How to do this efficiently?

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

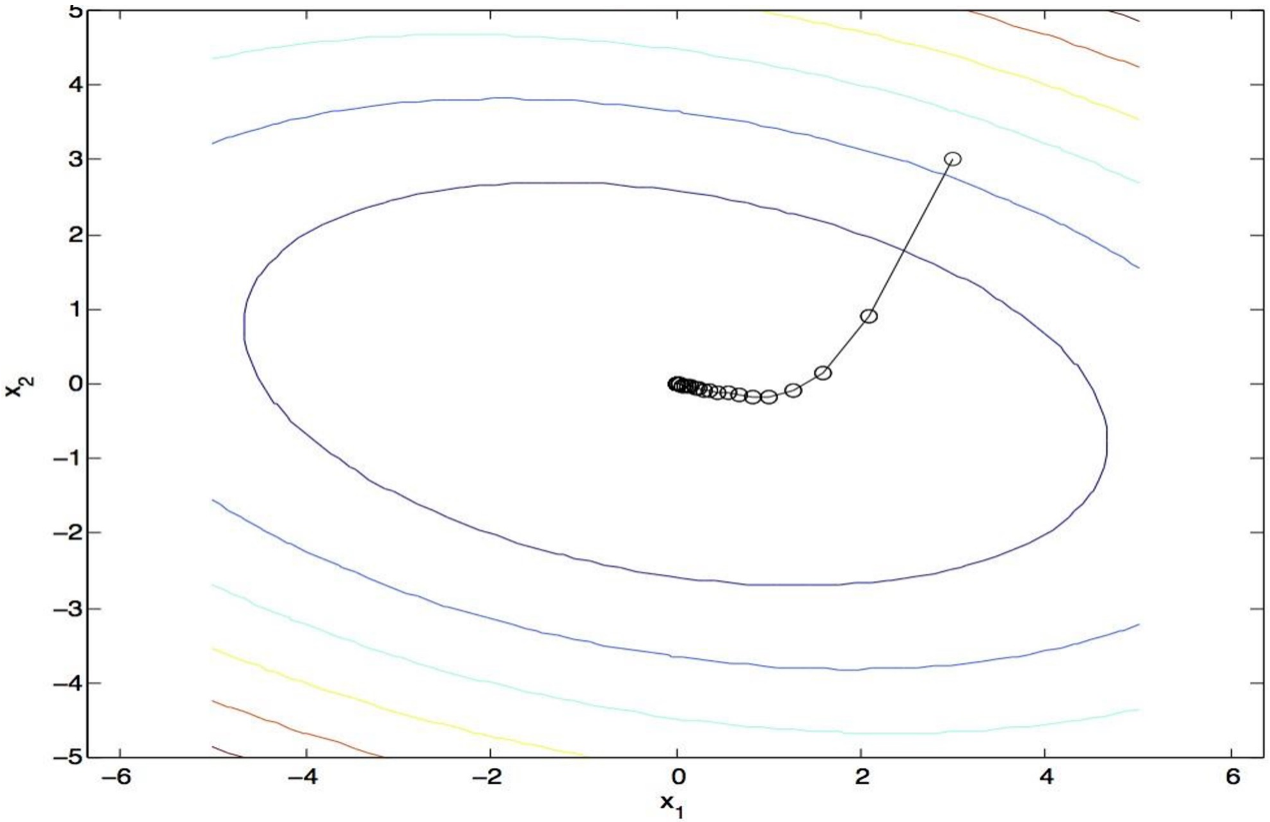


Figure source: Mathworks

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 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: $$\max_{\Delta:\|\Delta\|\leq\varepsilon} \Delta^\top a \qquad \Delta = \varepsilon\frac{a}{\|a\|}$$
- 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, …
```

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

- 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

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

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

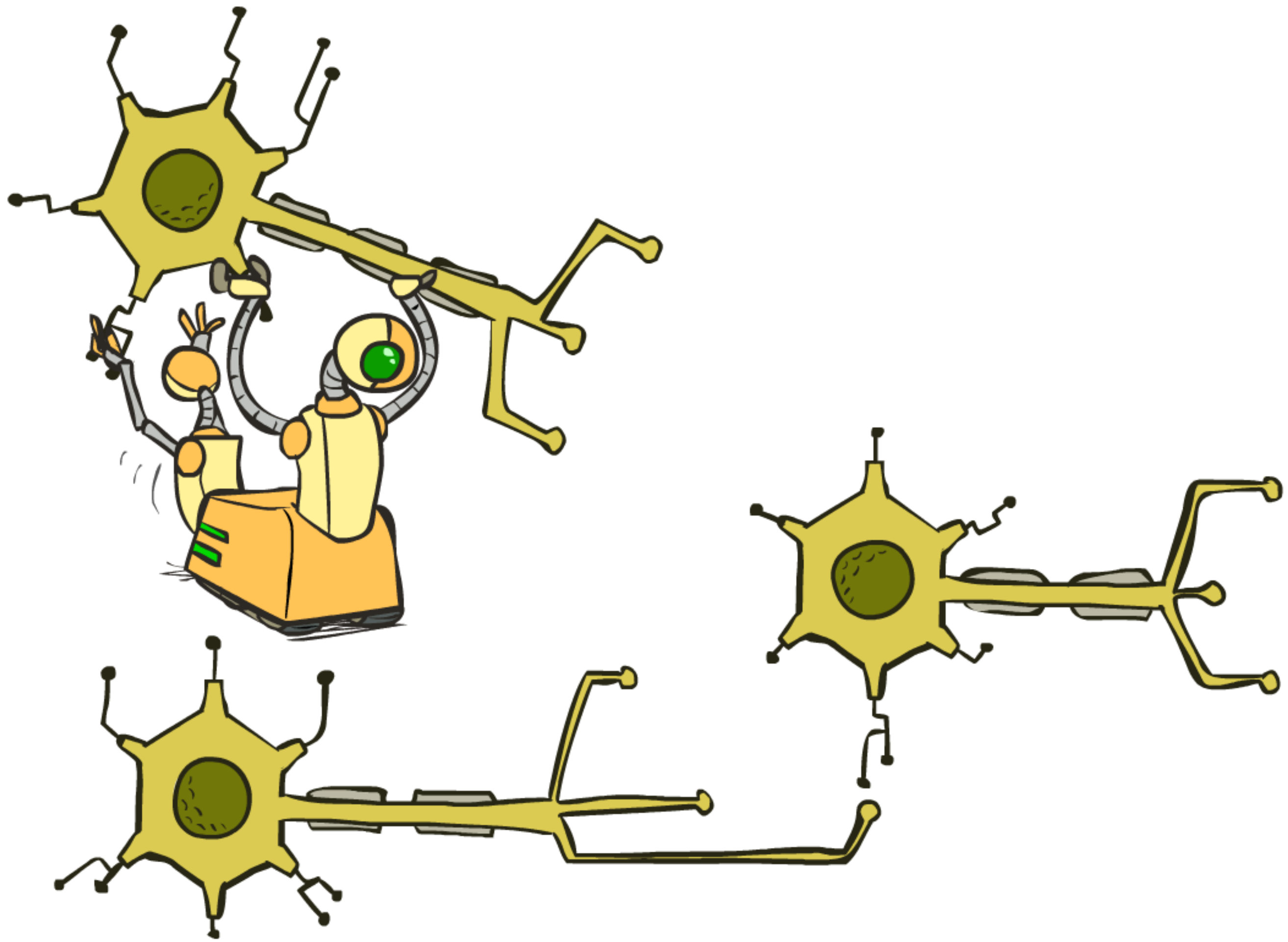
Mid-Semester Survey

- Add 2 points of extra credit to your midterm score!

Preview: Other Optimizers

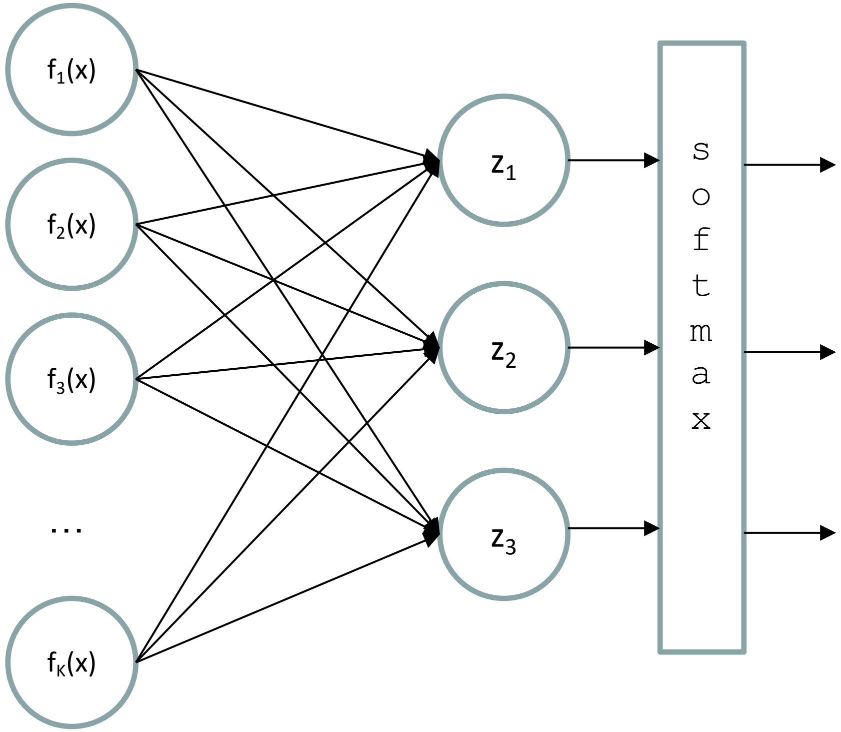
- Key ideas:
  - Second-order optimization methods
  - Momentum
  - Adaptive learning rates
- Example optimizers:
  - Newton’s method
  - Nesterov accelerated gradient
  - Adagrad, Adam, RMSProp, etc.

Neural Networks



Multi-class Logistic Regression

- = special case of neural network

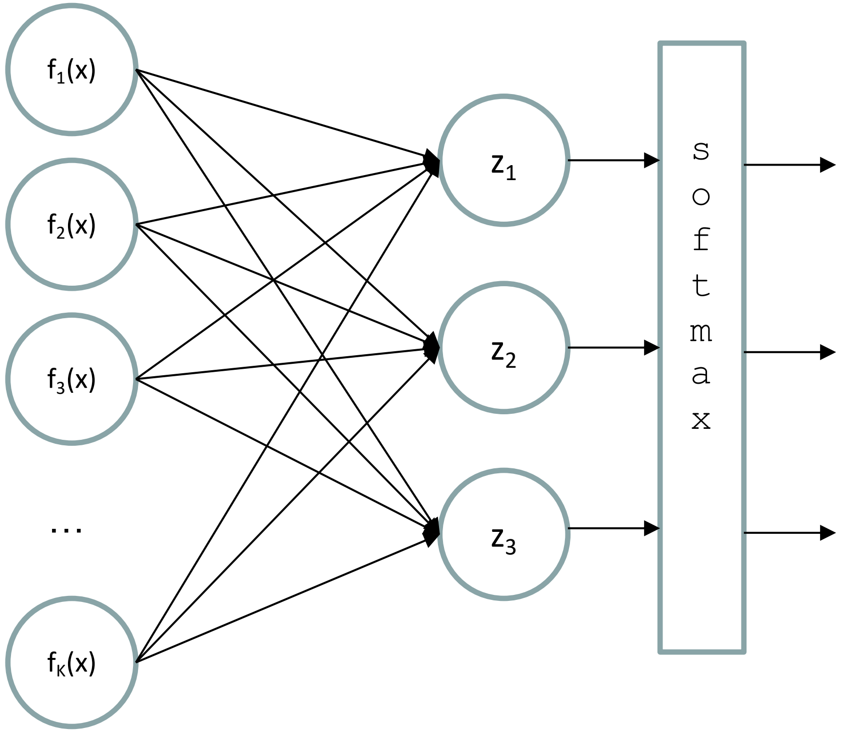


$$P(y_1|x;w) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$P(y_2|x;w) = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$P(y_3|x;w) = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

Deep Neural Network = Also learn the features!

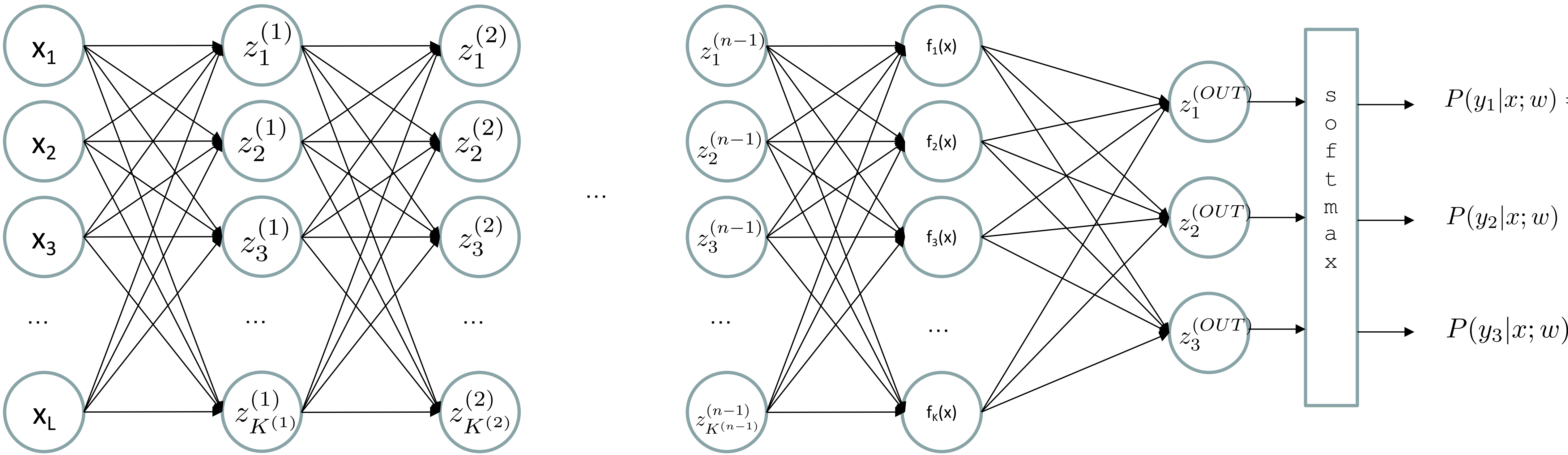


$$P(y_1|x;w) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$P(y_2|x;w) = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

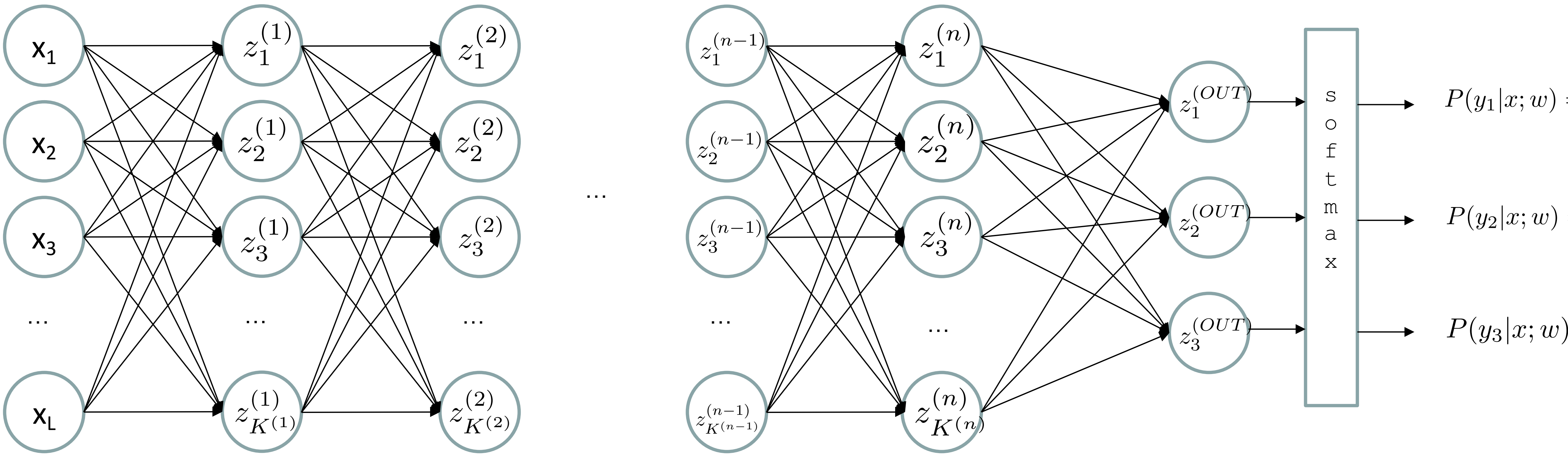
$$P(y_3|x;w) = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

Deep Neural Network = Also learn the features!



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

Deep Neural Network = Also learn the features!

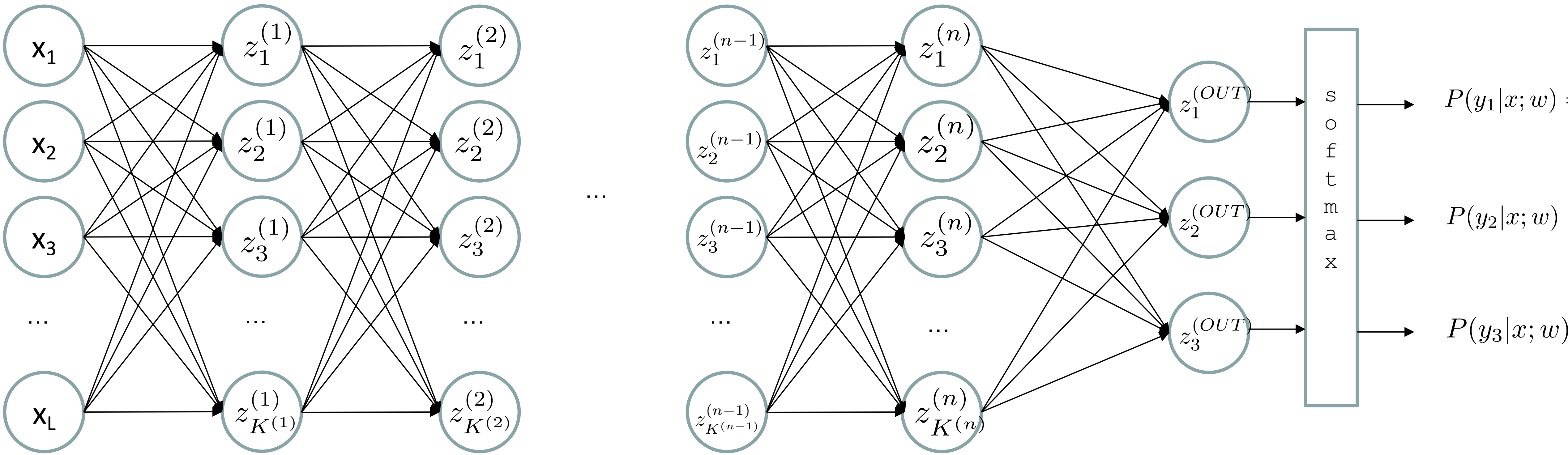


$$z_i^{(k)} = g\left(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)}\right)$$

Importance of Nonlinear Activation Functions

- What happens if we add more layers?
- $z_2 = W_2 (W_1 x + b_1) + b_2$
- $z_2 = W_2 (W_1 x + b_1) + b_2 = W_2 W_1 x + W_2 b_1 + b_2 = W_{new} x + b_{new}$
- No gain to adding more linear layers!
- Idea: add nonlinearities to capture more complex relationships

Deep Neural Network = Also learn the features!



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

g = nonlinear activation function