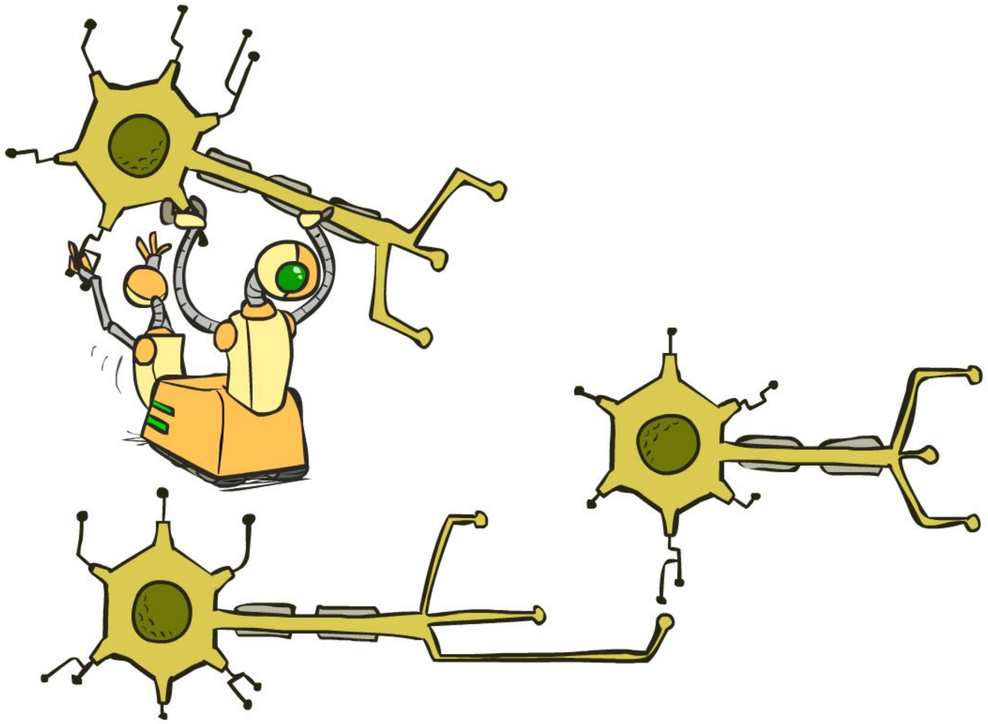
CS 188: Artificial Intelligence

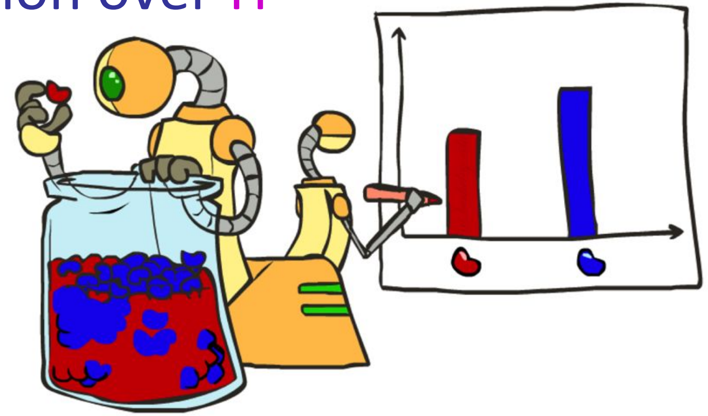
Neural Nets



Instructor: Stuart Russell and Dawn Song --- University of California, Berkeley

Recap: Maximum Likelihood Estimation

- Learning = Bayesian updating of a probability distribution over H
- Prior P(H), training data **X**



- Maximum likelihood estimation

$$\theta_{ML} = \arg\max_{\theta} P(\mathbf{X}|\theta)$$

$$= \arg\max_{\theta} \prod_i P_\theta(X_i)$$

- Maximum *conditional* likelihood estimation

$$\theta^\star = \arg\max_{\theta} P(\mathbf{Y}|\mathbf{X}, \theta)$$

$$= \arg\max_{\theta} \prod_i P_\theta(y_i|x_i)$$

- How to solve for θ_{ML}?

Recap: Naïve Bayes

- Naïve Bayes model:
  - Attributes conditionally independent of each other, given the class
  - Assuming there are 2 classes, n Boolean attributes X_i, how many parameters are in this Naïve Bayes model?

$$\theta = P(C = true), \theta_{i1} = P(X_i = true \mid C = true), \theta_{i2} = P(X_i = true \mid C = false)$$

  - Maximum likelihood parameter estimation
  - Inference: what's the class given attribute values ($x_1, \ldots, x_n$)?

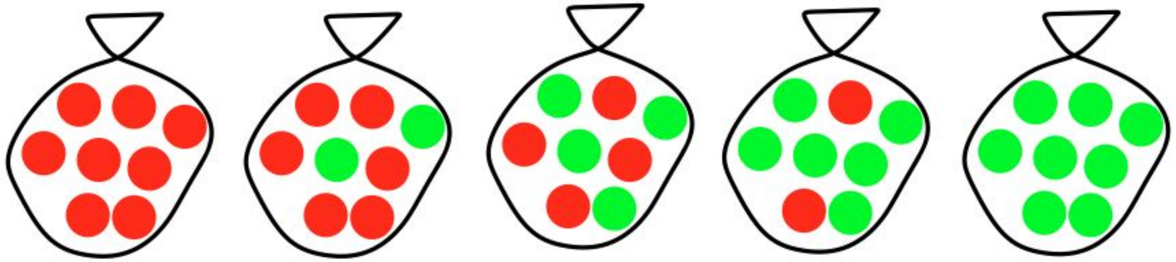
$$\mathbf{P}(C \mid x_1, \ldots, x_n) = \alpha\, \mathbf{P}(C) \prod_i \mathbf{P}(x_i \mid C)$$

Bayesian learning

- Learning = Bayesian updating of a probability distribution over H
- Prior P(H), training data $\mathbf{X}=\mathbf{x}_1,\ldots,\mathbf{x}_N$
- Given the data so far, each hypothesis has a posterior probability:
  - $P(h_k|\mathbf{X}) = \alpha\, P(\mathbf{X}|h_k)P(h_k)$ = α x Likelihood x Prior
- Predictions use a likelihood-weighted average over the hypotheses:
  - $P(\mathbf{x}_{N+1}|\mathbf{X}) = \Sigma_k\, P(\mathbf{x}_{N+1}|\mathbf{X},h_k)P(h_k|\mathbf{X}) = \Sigma_k\, P(\mathbf{x}_{N+1}|h_k)P(h_k|\mathbf{X})$
- No need to pick one best-guess hypothesis!
  - Drawback: Σ_k may be expensive/impossible for large/infinite H

Example: Surprise Candy Co.

- Suppose there are five kinds of bags of candies, no labels!!
  - 10% are h1: 100% cherry candies
  - 20% are h2: 75% cherry candies + 25% lime candies
  - 40% are h3: 50% cherry candies + 50% lime candies
  - 20% are h4: 25% cherry candies + 75% lime candies
  - 10% are h5: 100% lime candies



- Then we observe candies drawn from some bag:

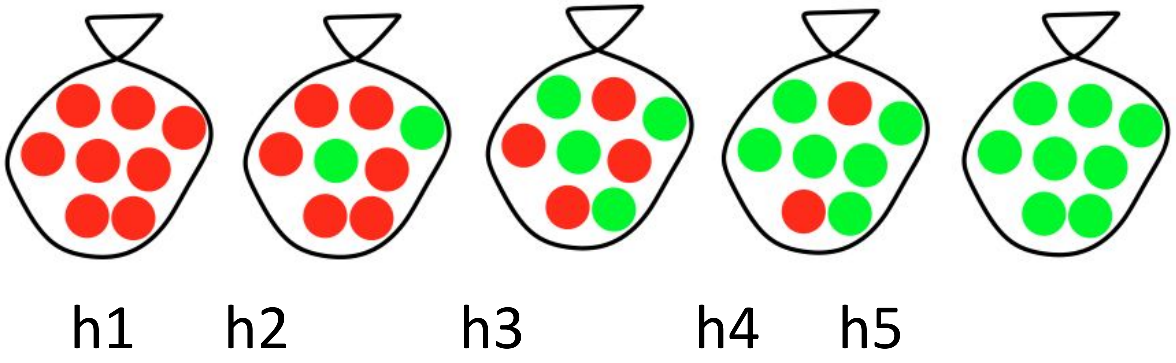
- What kind of bag is it?
- What flavour will the next candy be?

Posterior probability of hypotheses

$$P(\mathbf{d}|h_i) = \prod_j P(d_j|h_i)$$

- Prior over $h_1, \ldots, h_5$: <0.1, 0.2, 0.4, 0.2, 0.1>
- $P(h_k|\mathbf{X}) = \alpha P(\mathbf{X}|h_k)P(h_k)$
  - P(h1 | 5 limes) = αP(5 limes | h1) P(h1)= ____________________
  - P(h2 | 5 limes) = αP(5 limes | h2) P(h2) = _____________________
  - P(h3 | 5 limes) = αP(5 limes | h3) P(h3) = ______________________
  - P(h4 | 5 limes) = αP(5 limes | h4) P(h4) = _______________________
  - P(h5 | 5 limes) = αP(5 limes | h5) P(h5) = ______________________
- α = _________________________

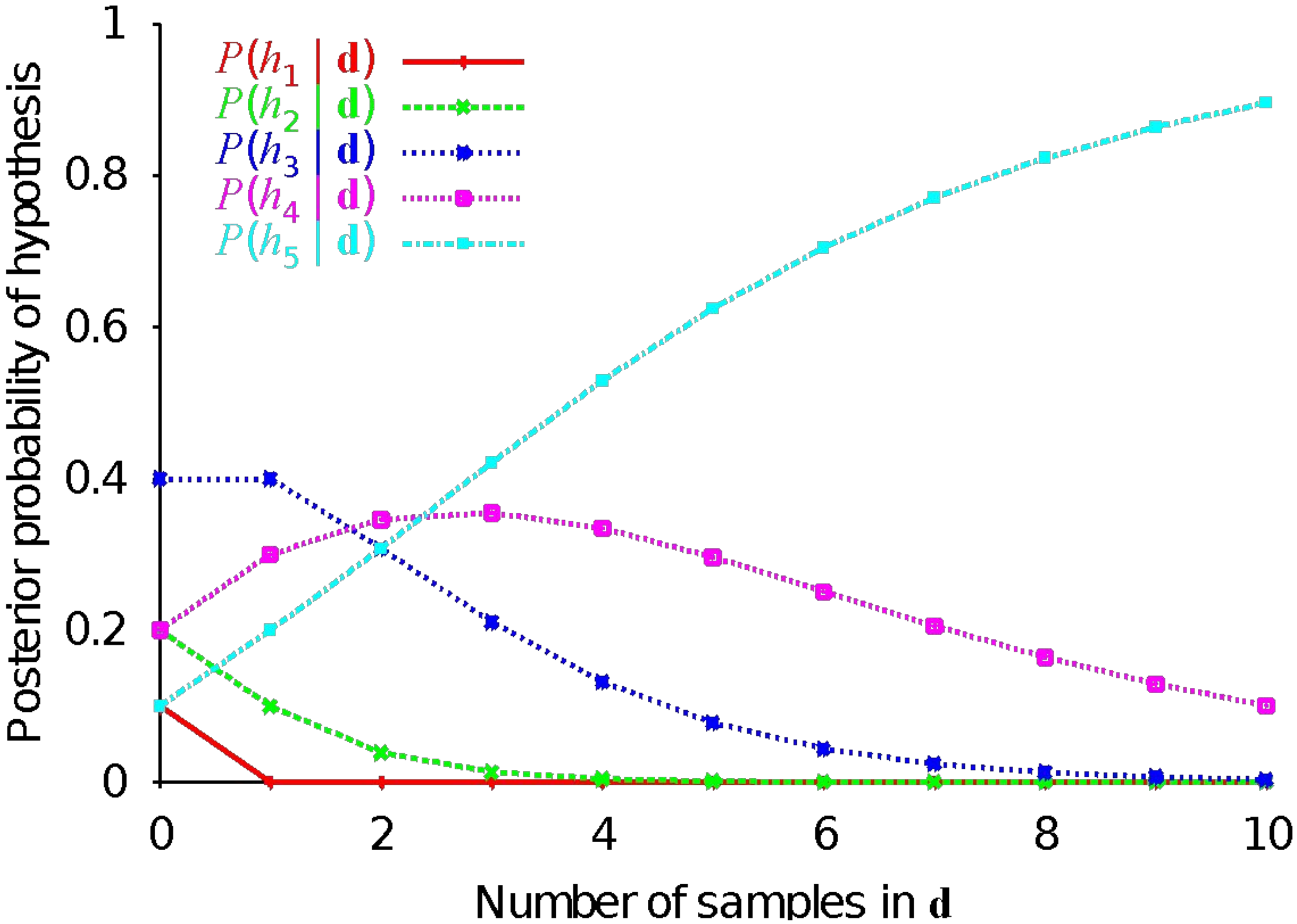
h_1: 100% cherry,
h_2: 75% cherry + 25% lime,
h_3: 50% cherry + 50% lime,
h_4: 25% cherry + 75% lime,
h_5: 100% lime.



Posterior probability of hypotheses

- $P(h_k|\mathbf{X}) = \alpha P(\mathbf{X}|h_k)P(h_k)$
  - P(h1 | 5 limes) = αP(5 limes | h1) P(h1)= $\alpha \cdot 0.0^5 \cdot 0.1$ = 0
  - P(h2 | 5 limes) = αP(5 limes | h2) P(h2) = $\alpha \cdot 0.25^5 \cdot 0.2$ = 0.000195 α
  - P(h3 | 5 limes) = αP(5 limes | h3) P(h3) = $\alpha \cdot 0.5^5 \cdot 0.4$ = 0.0125 α
  - P(h4 | 5 limes) = αP(5 limes | h4) P(h4) = $\alpha \cdot 0.75^5 \cdot 0.2$ = 0.0475 α
  - P(h5 | 5 limes) = αP(5 limes | h5) P(h5) = $\alpha \cdot 1.0^5 \cdot 0.1$ = 0.1 α
- α = 1/(0 + 0.000195 + 0.0125 + 0.0475 + 0.1) = 6.2424
- P(h1 | 5 limes) = 0
- P(h2 | 5 limes) = 0.00122
- P(h3 | 5 limes) = 0.07803
- P(h4 | 5 limes) = 0.29650
- P(h5 | 5 limes) = 0.62424

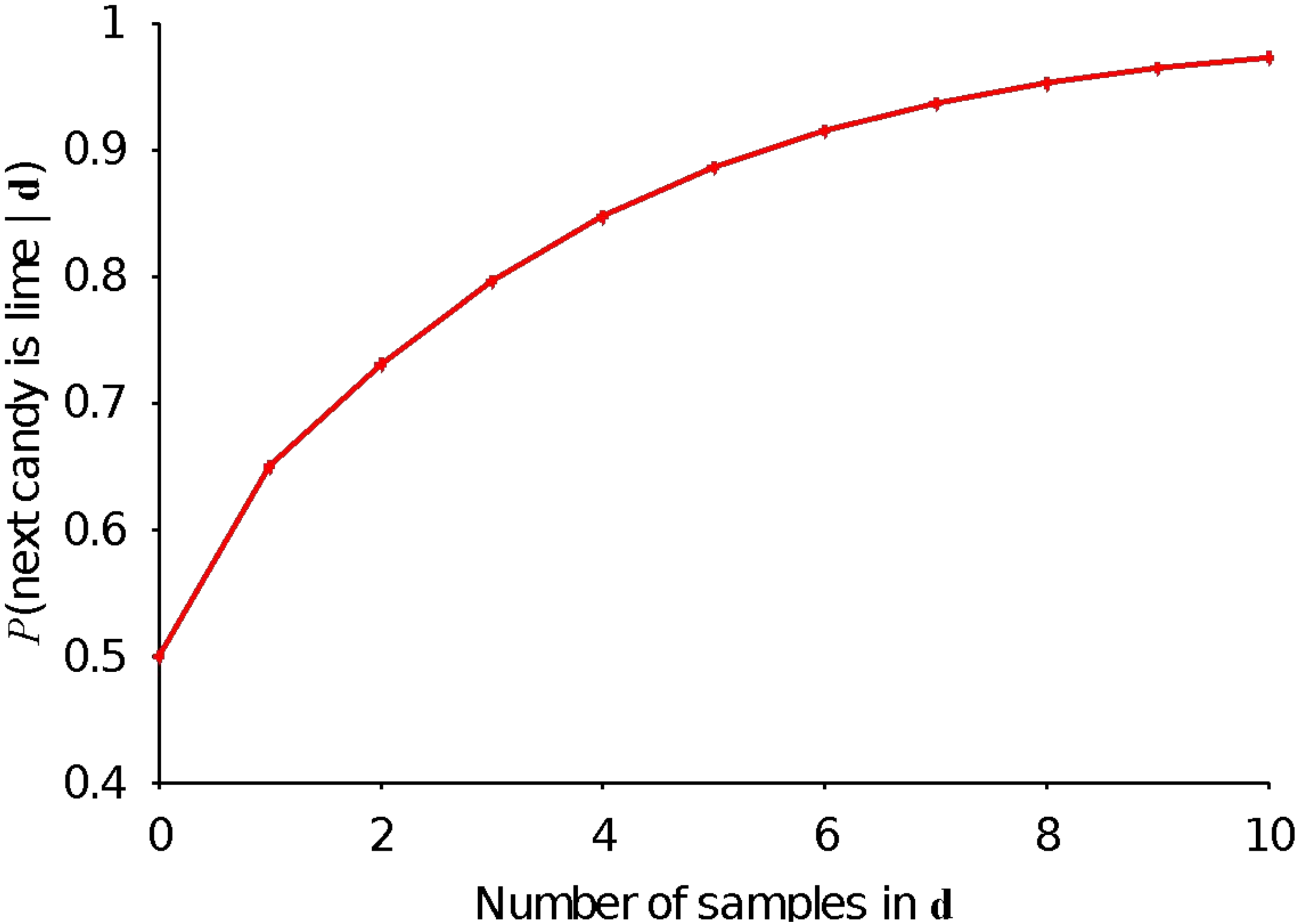
Posterior probability of hypotheses



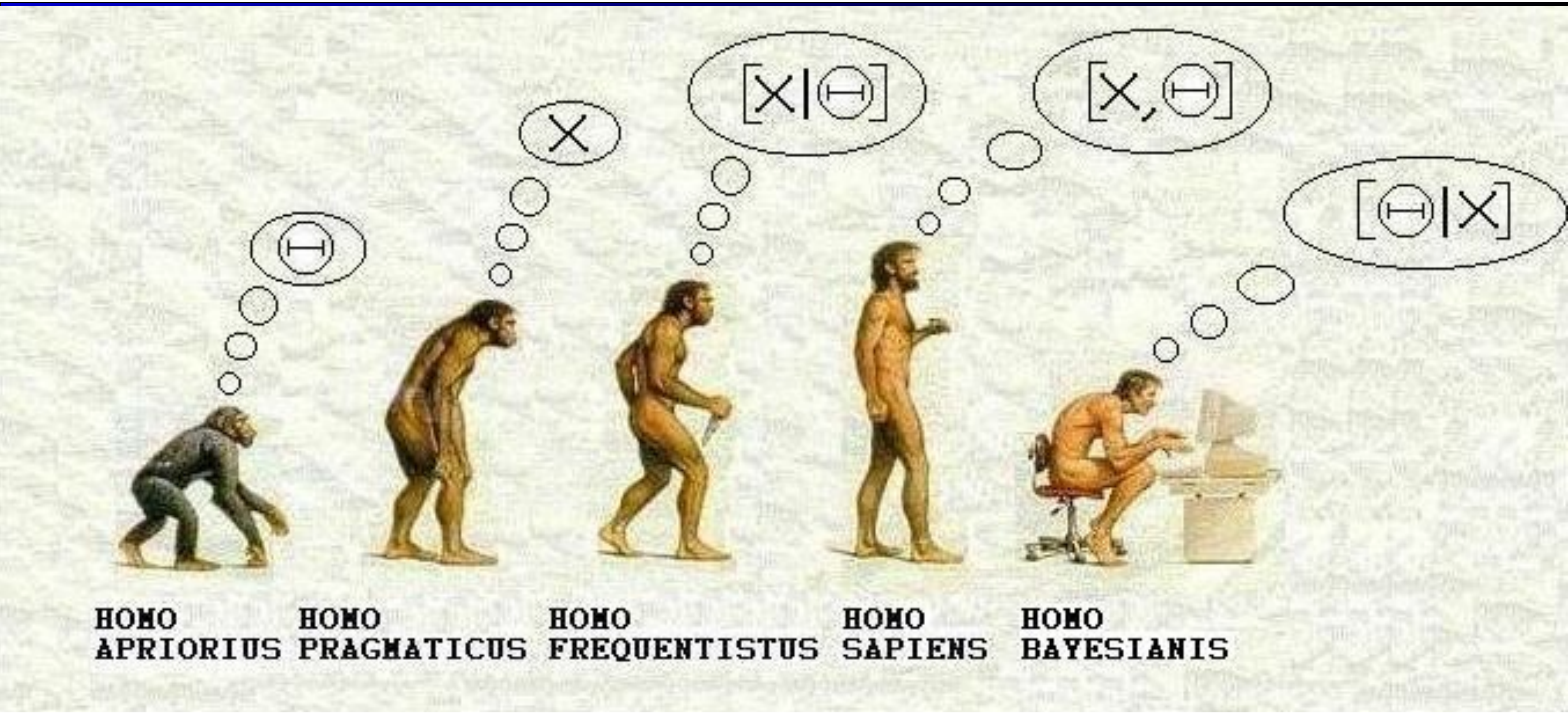
Prediction probability

- $P(\mathbf{x}_{N+1} | \mathbf{X}) = \Sigma_k P(\mathbf{x}_{N+1} | h_k) P(h_k | \mathbf{X})$
- P(lime on 6 | 5 limes)
  - = P(lime on 6 | h1)P(h1 | 5 limes)
  - + P(lime on 6 | h2)P(h2 | 5 limes)
  - + P(lime on 6 | h3)P(h3 | 5 limes)
  - + P(lime on 6 | h4)P(h4 | 5 limes)
  - + P(lime on 6 | h5)P(h5 | 5 limes)
  - $= 0\times 0 + 0.25\times 0.00122 + 0.5\times 0.07830 + 0.75\times 0.29650 + 1.0\times 0.62424$
  - $= 0.88607$

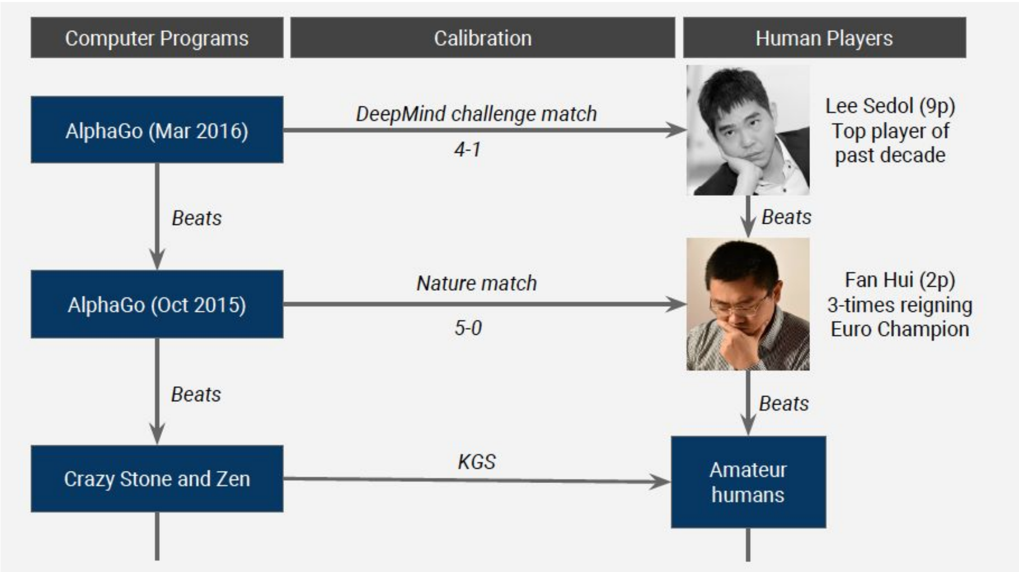
Prediction probability

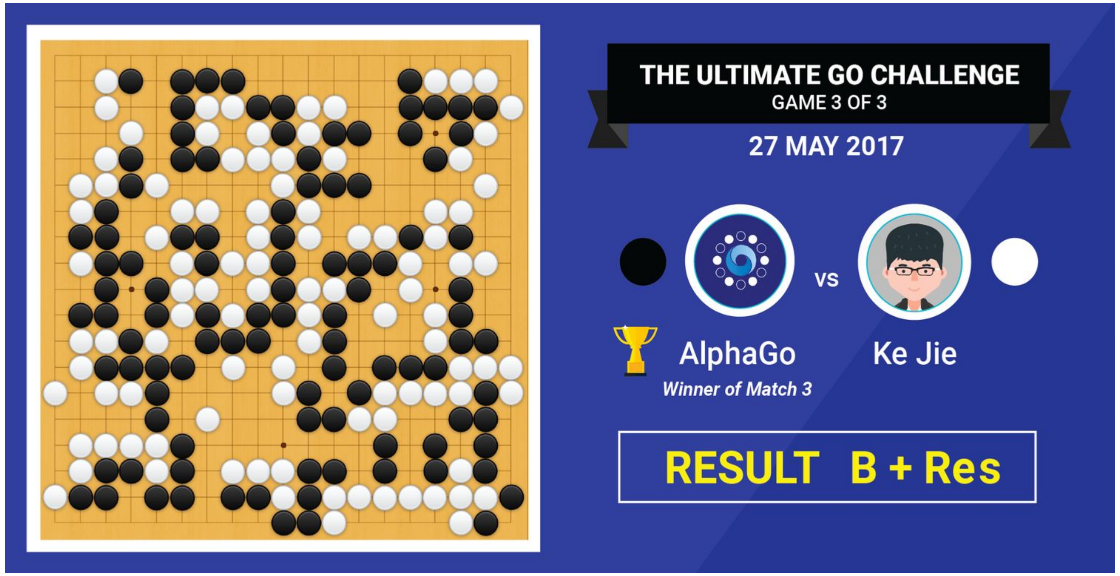


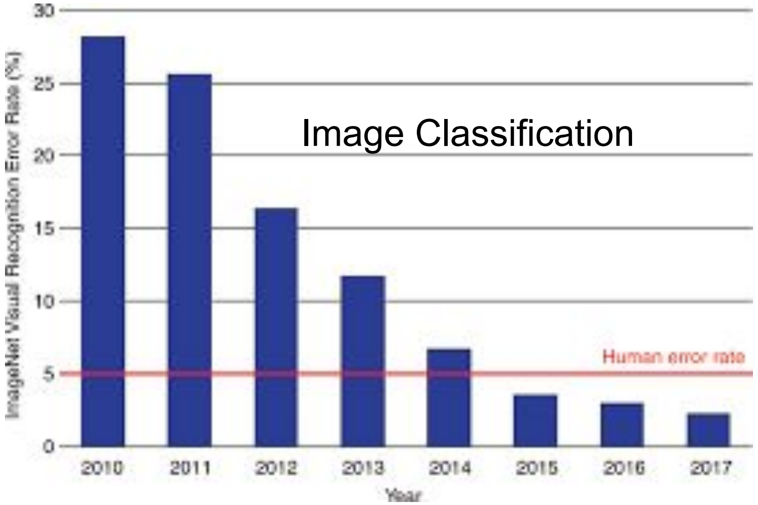
Bayesian learning

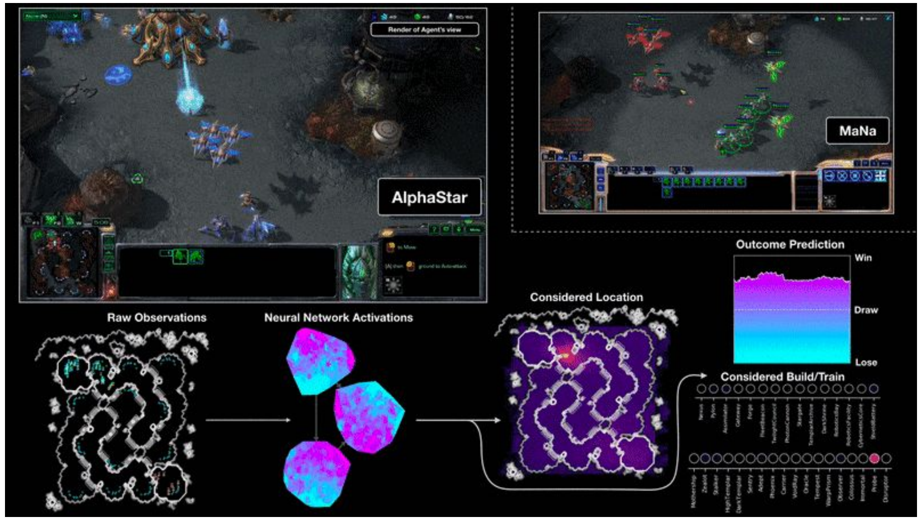


Deep Learning/Neural Network

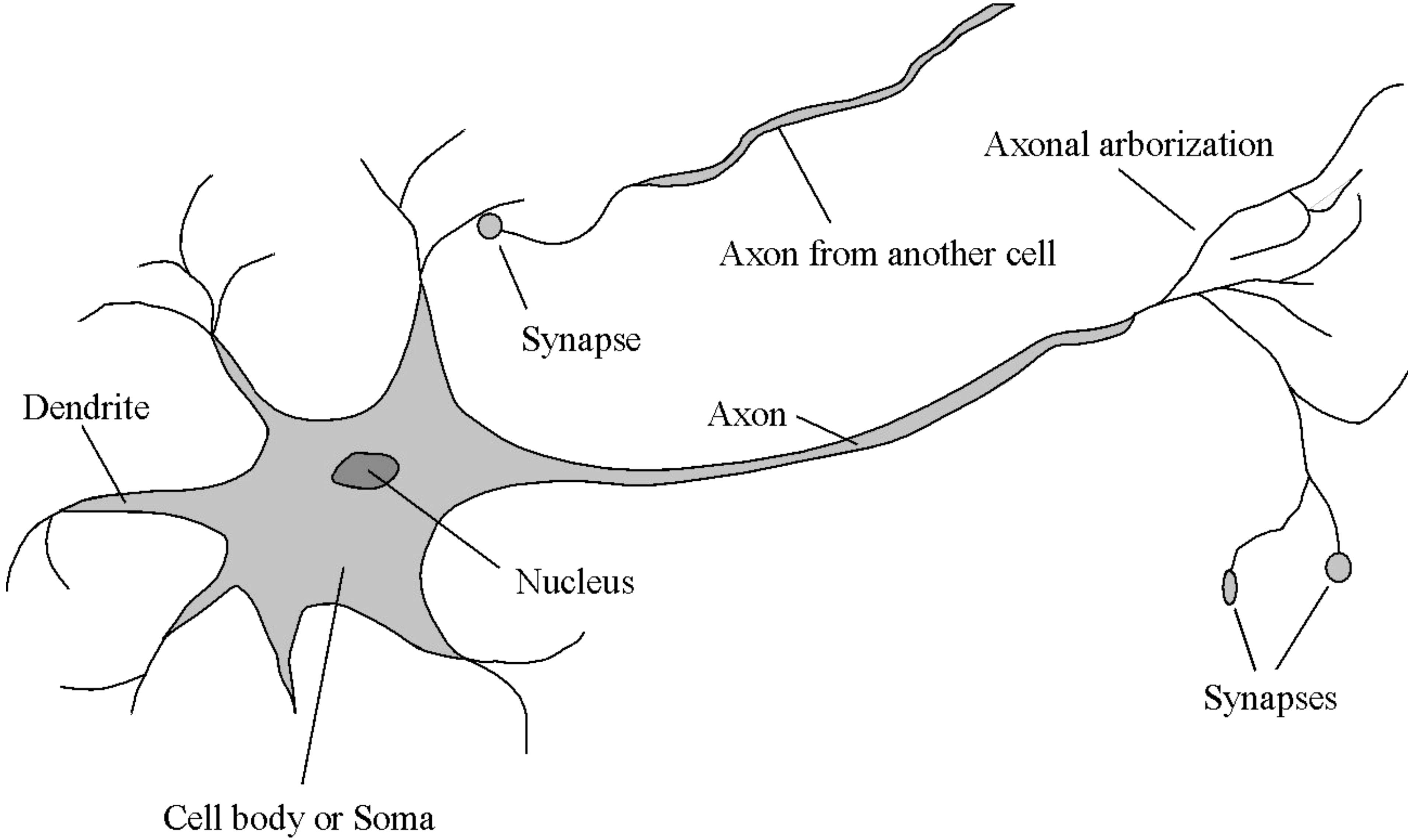




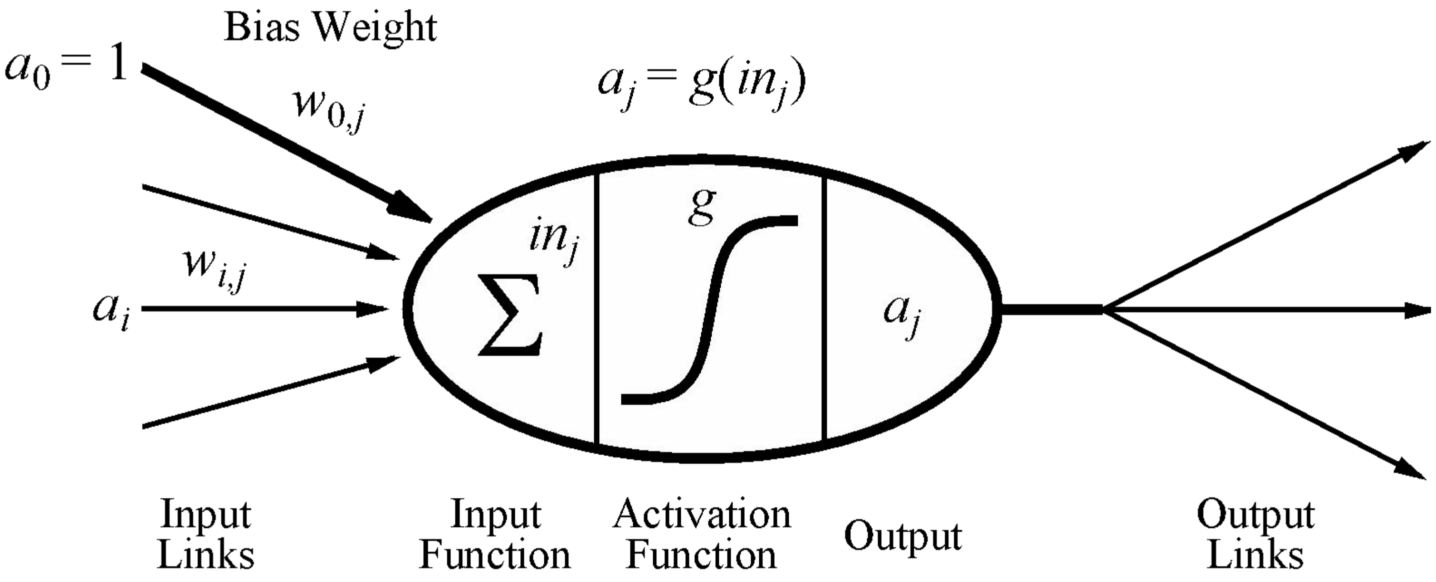




Very loose inspiration: Human neurons

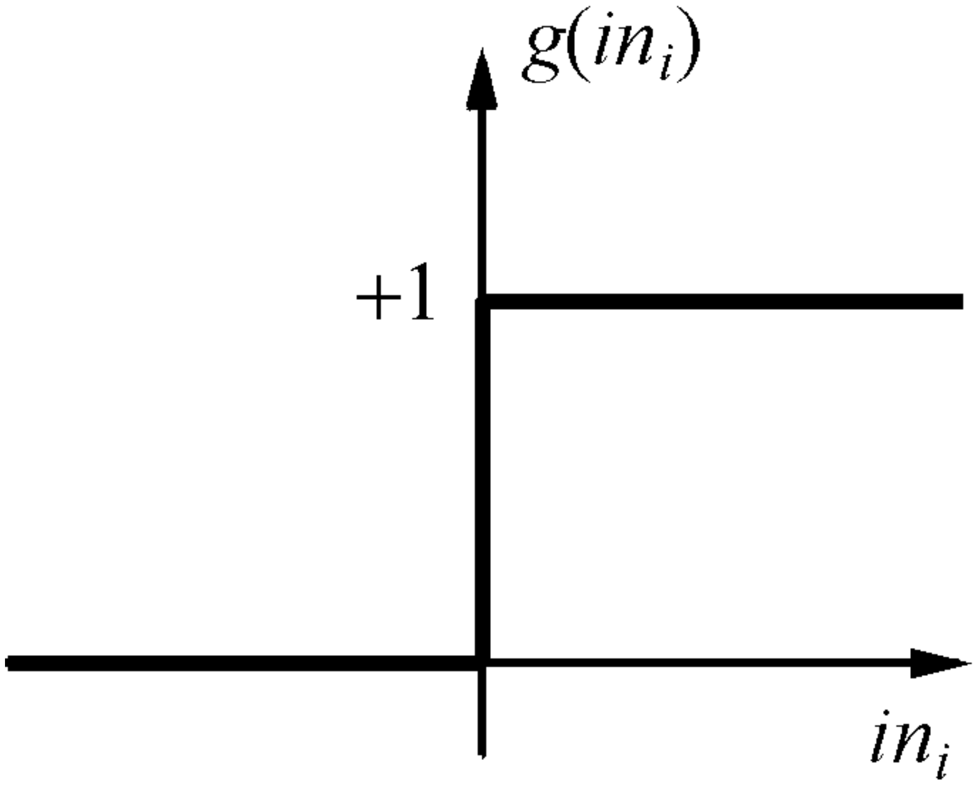


Simple model of a neuron (McCulloch & Pitts, 1943)

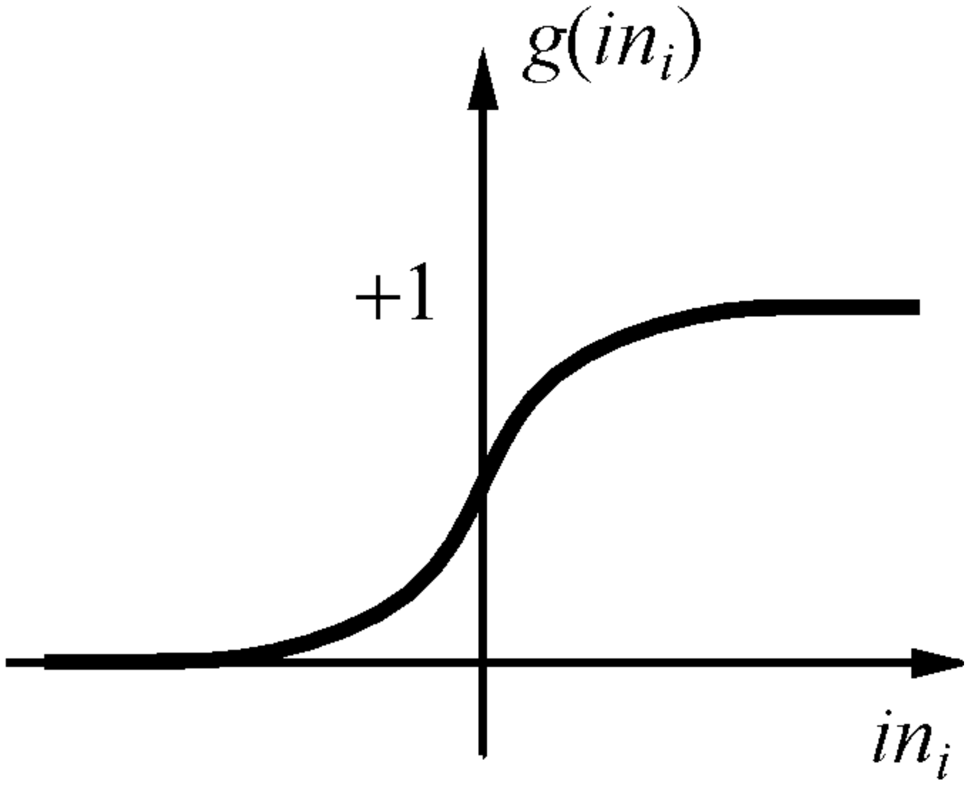


- Inputs a_i come from the output of node i to this node j (or from "outside")
- Each input link has a ***weight*** $w_{i,j}$
- There is an additional fixed input a_0 with ***bias*** weight $w_{0,j}$
- The total input is $in_j = \Sigma_i \, w_{i,j} \, a_i$
- The output is $a_j = g(in_j) = g(\Sigma_i \, w_{i,j} \, a_i) = g(\mathbf{w.a})$

Activation functions g



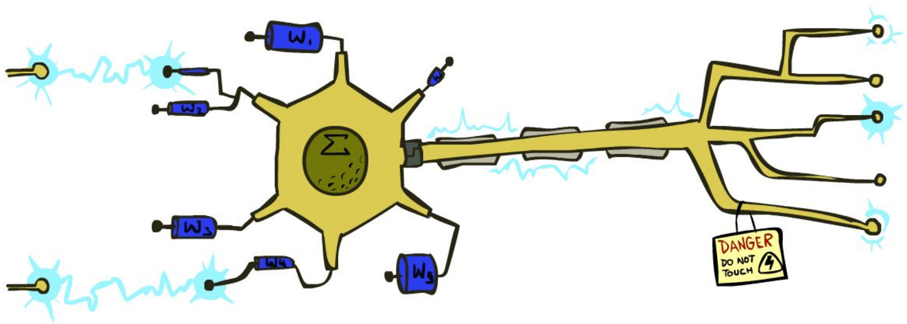
Threshold



Sigmoid $1/(1+e^{-x})$

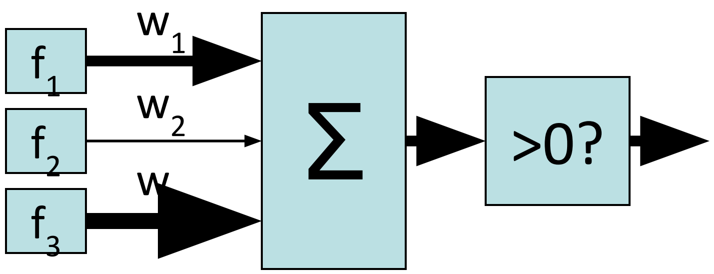
Reminder: Linear Classifiers

- 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



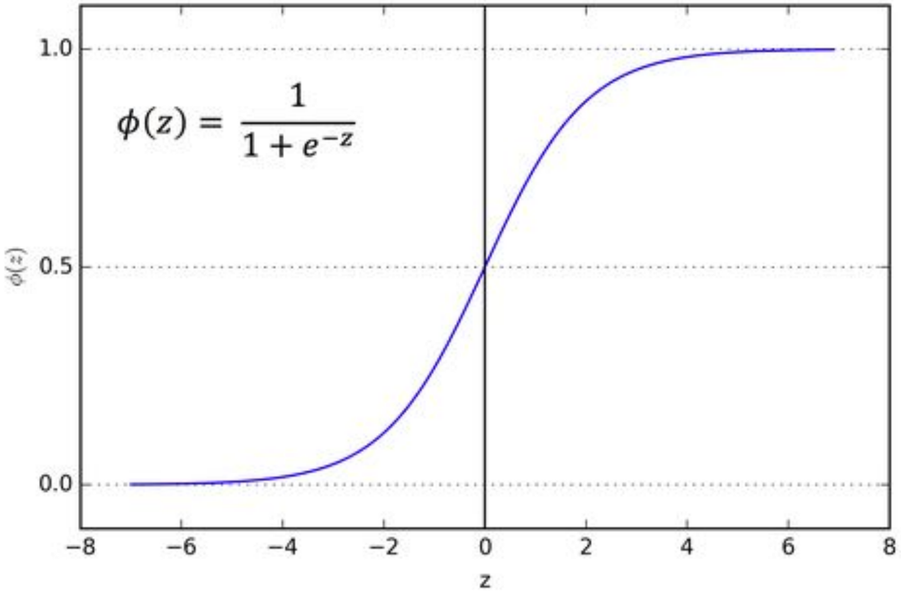
How to get probabilistic decisions?

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

- 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:

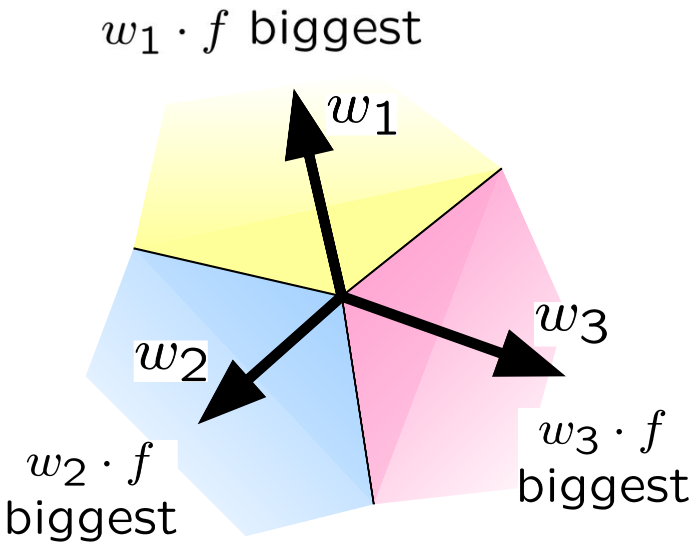
$$P(y^{(i)} = +1 | x^{(i)}; w) = \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

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

= 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

Optimization

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

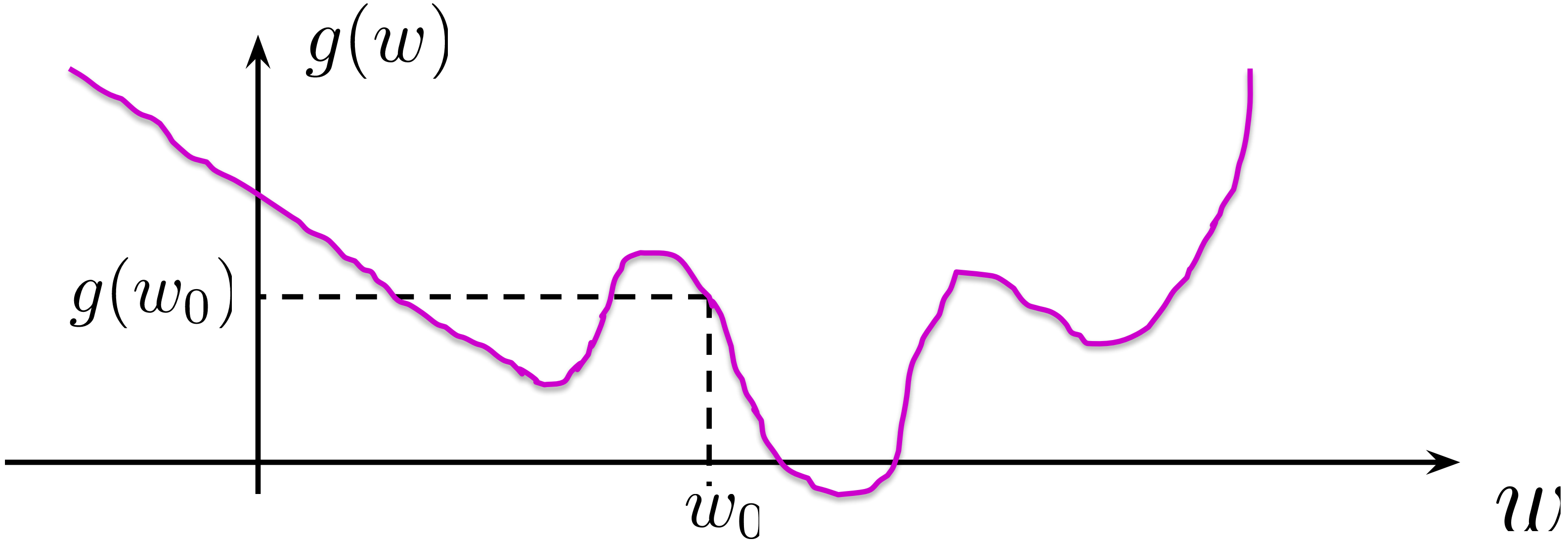
$$\max_{w} \quad ll(w) = \max_{w} \quad \sum_{i} \log P(y^{(i)}|x^{(i)}; w)$$

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?

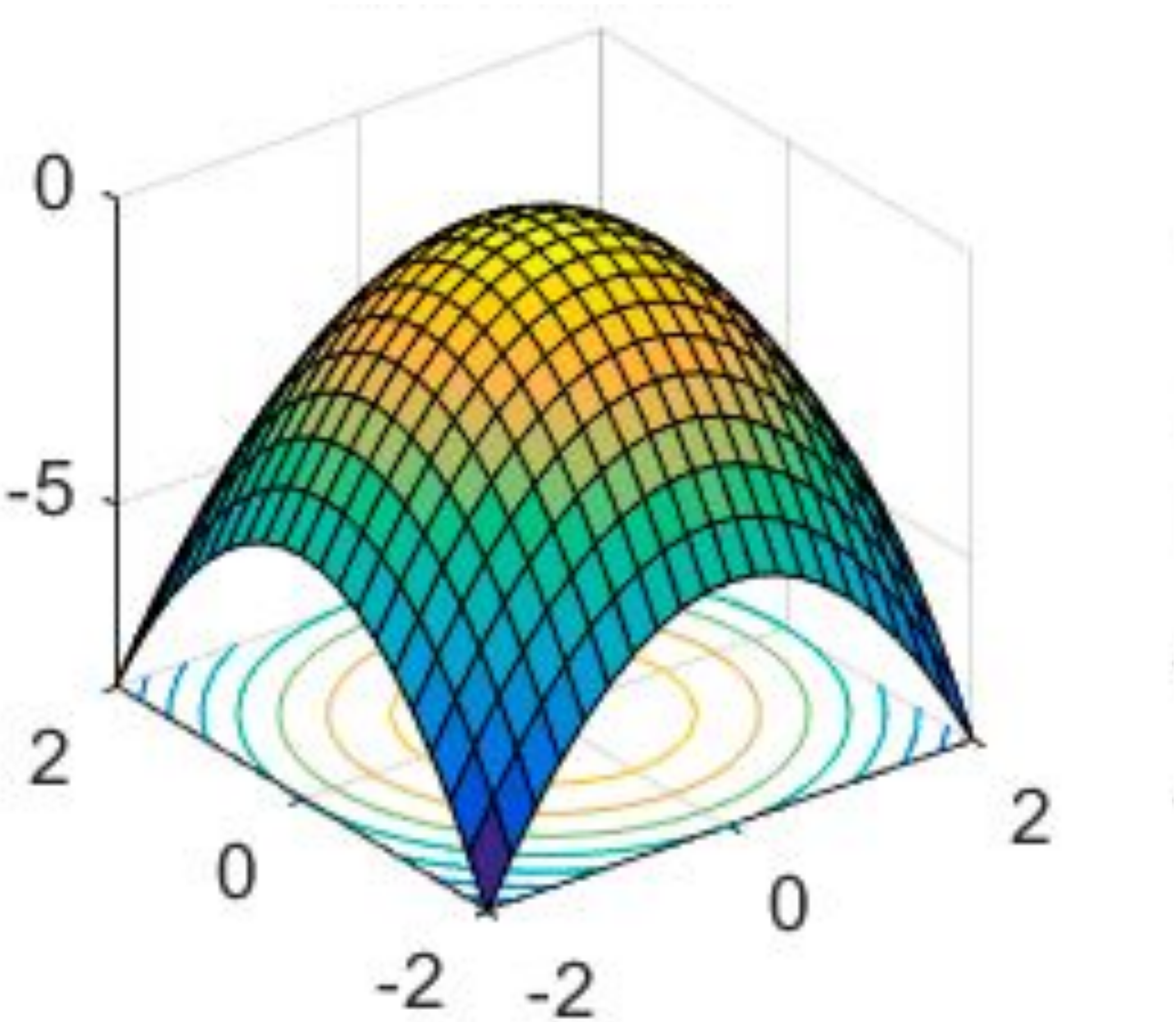
1-D Optimization



- 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**

Steepest Descent

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

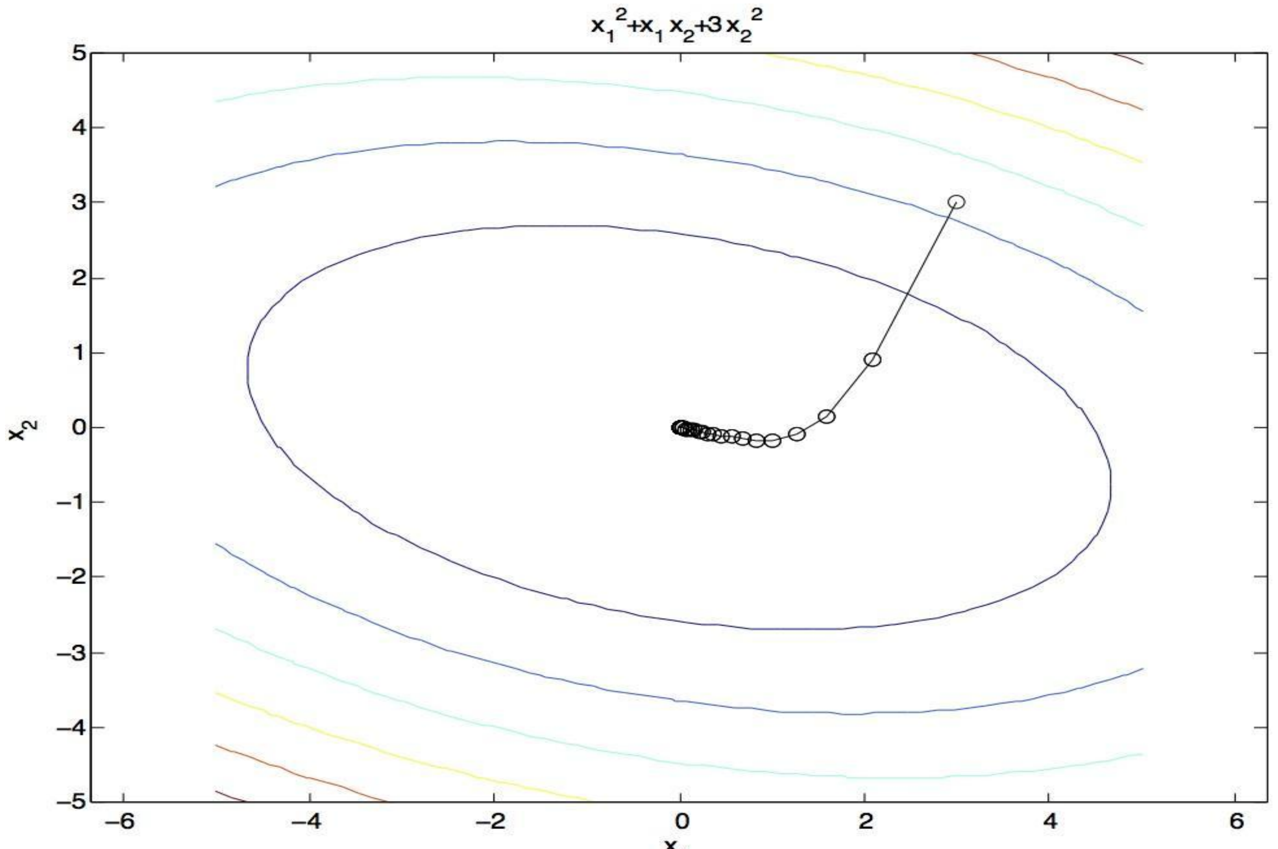


Figure source: Mathworks

Steepest Direction

- Steepest Direction = direction of the gradient

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

Optimization Procedure: Gradient Ascent

- `init` w
- `for iter = 1, 2, …`

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

- α: learning rate --- hyperparameter that needs to be chosen carefully

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

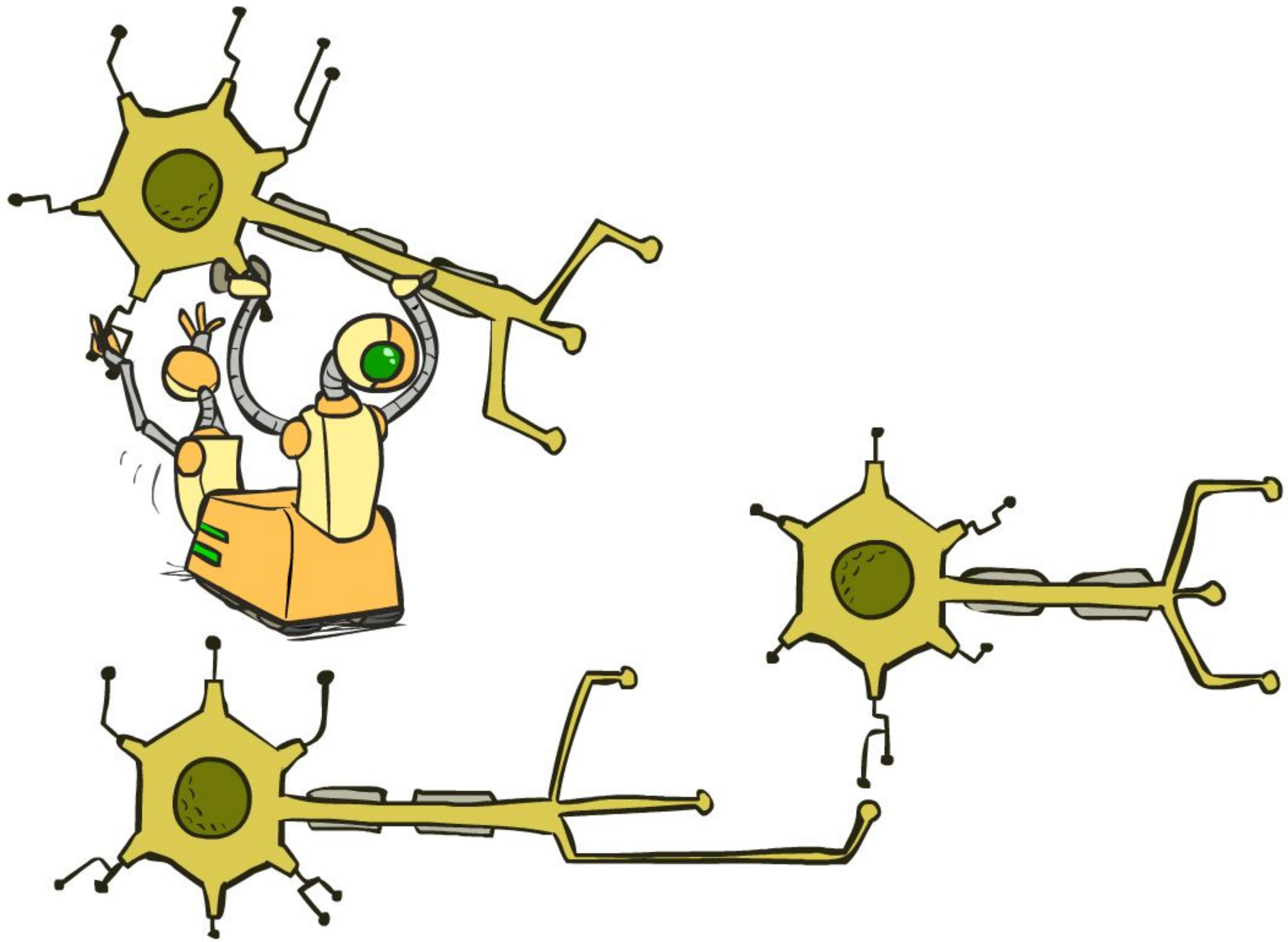
$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

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

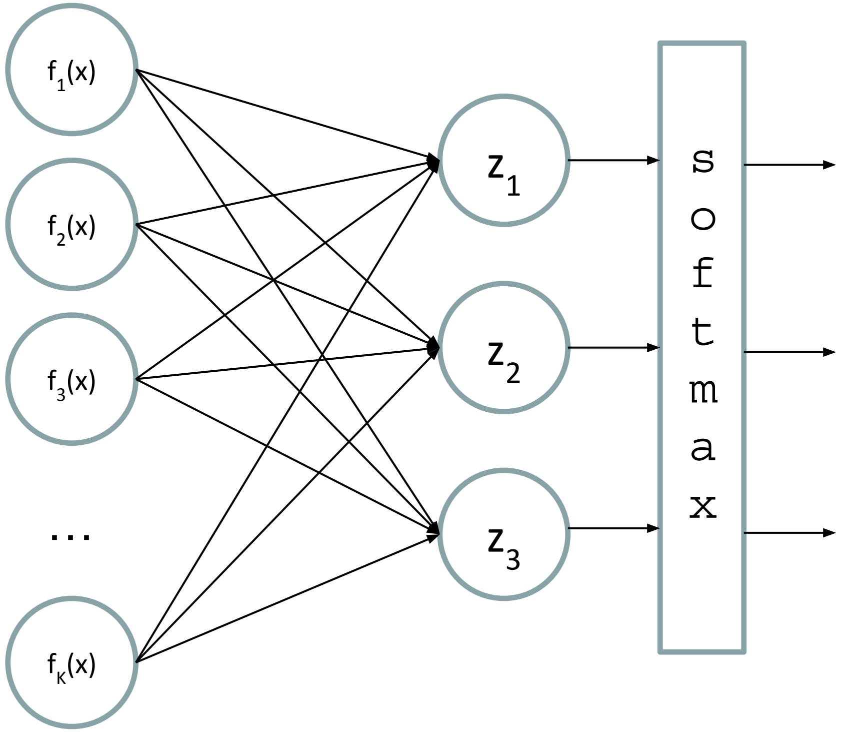
$$w \leftarrow w + \alpha * \sum_{j \in J} \nabla \log P(y^{(j)}|x^{(j)}; w)$$

Neural Networks



Multi-class Logistic Regression

- = special case of neural network

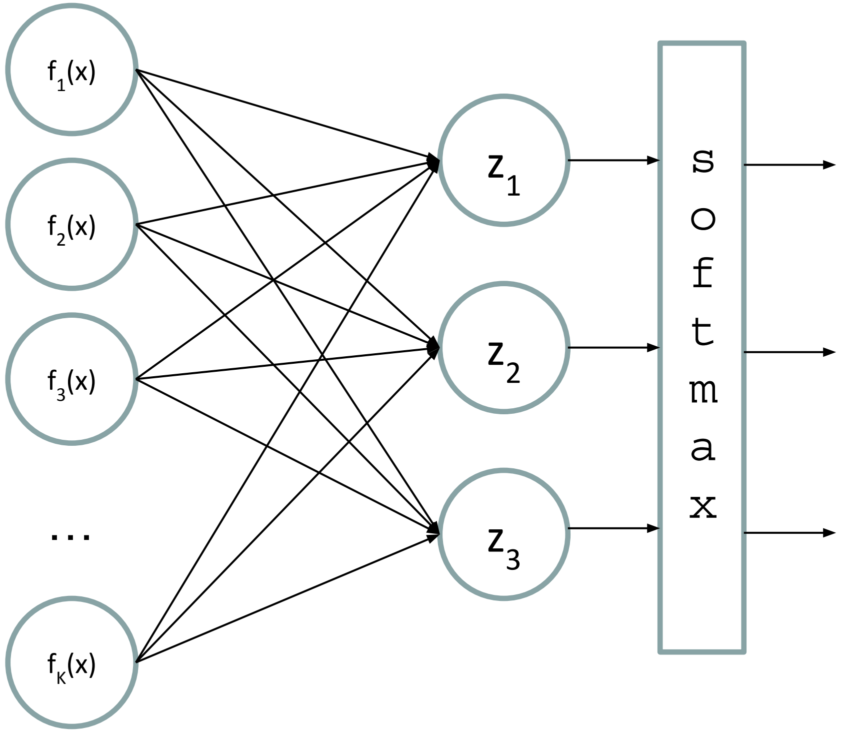


$$P(y_1|x;w) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

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

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

Deep Neural Network = Also learn the features!

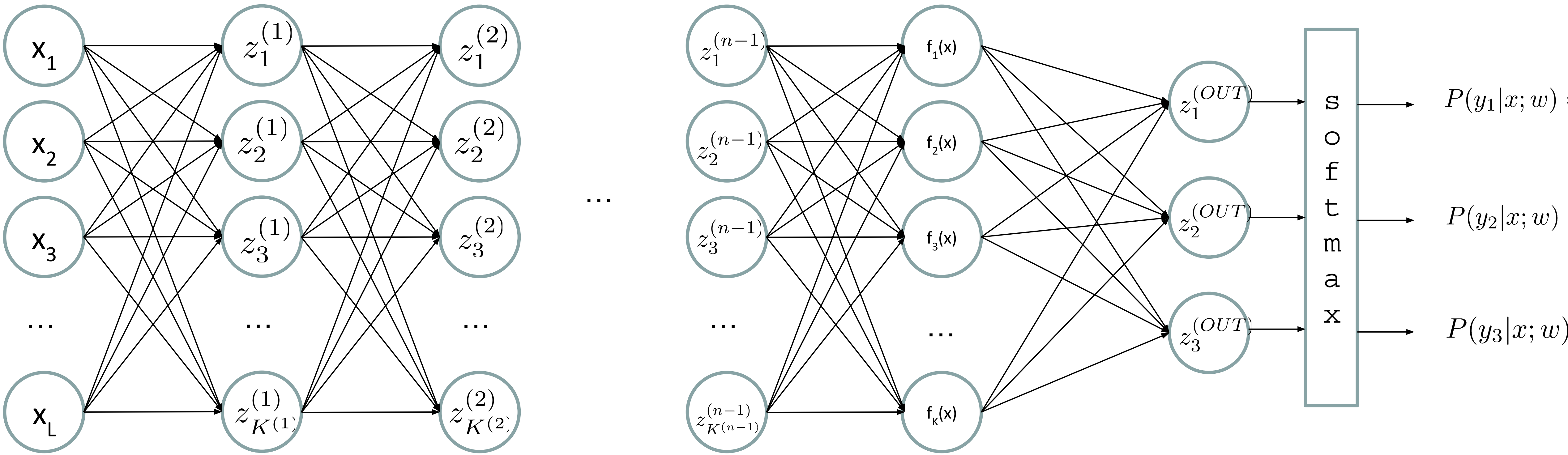


$$P(y_1|x;w) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

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

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

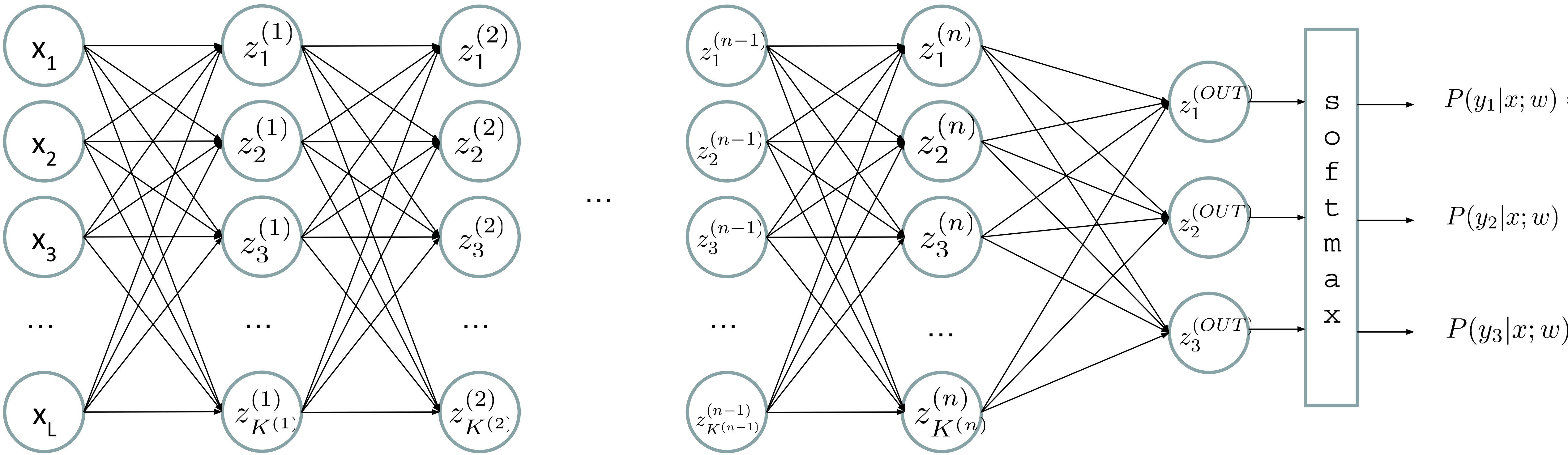
Deep Neural Network = Also learn the features!



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

g = nonlinear activation function

Deep Neural Network = Also learn the features!

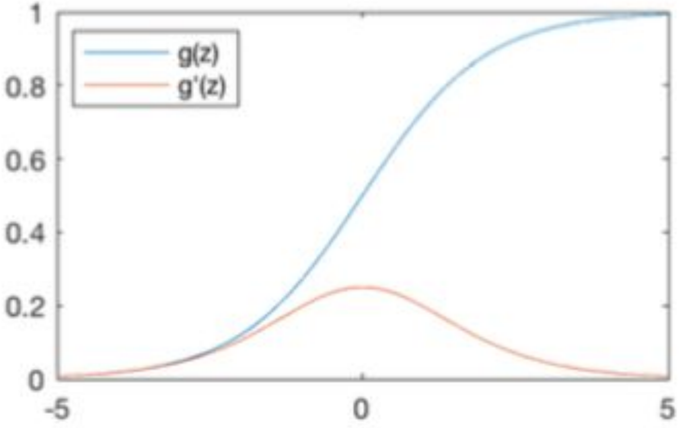


$$z_i^{(k)} = g\left(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)}\right)$$

g = nonlinear activation function

Common Activation Functions

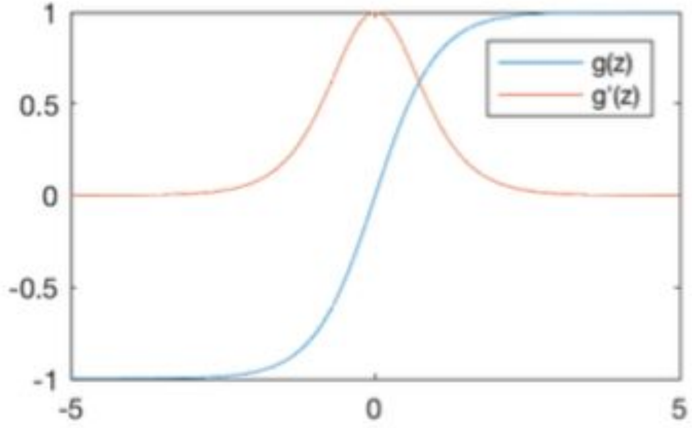
Sigmoid Function



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

$$g'(z) = g(z)(1 - g(z))$$

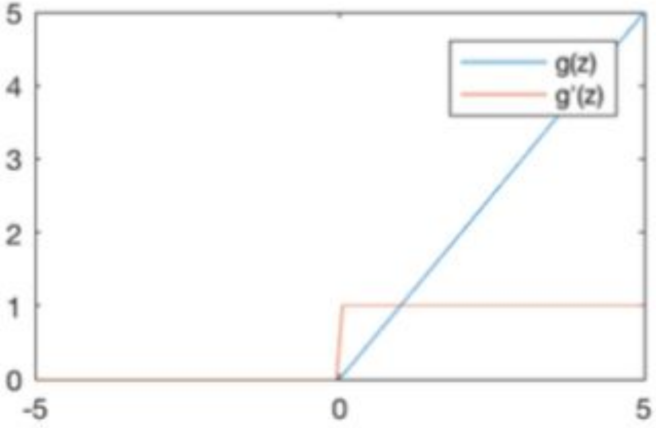
Hyperbolic Tangent



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

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

[source: MIT 6.S191 introtodeeplearning.com]

Deep Neural Network: Also Learn the Features!

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

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

just w tends to be a much, much larger vector ☺

just run gradient ascent