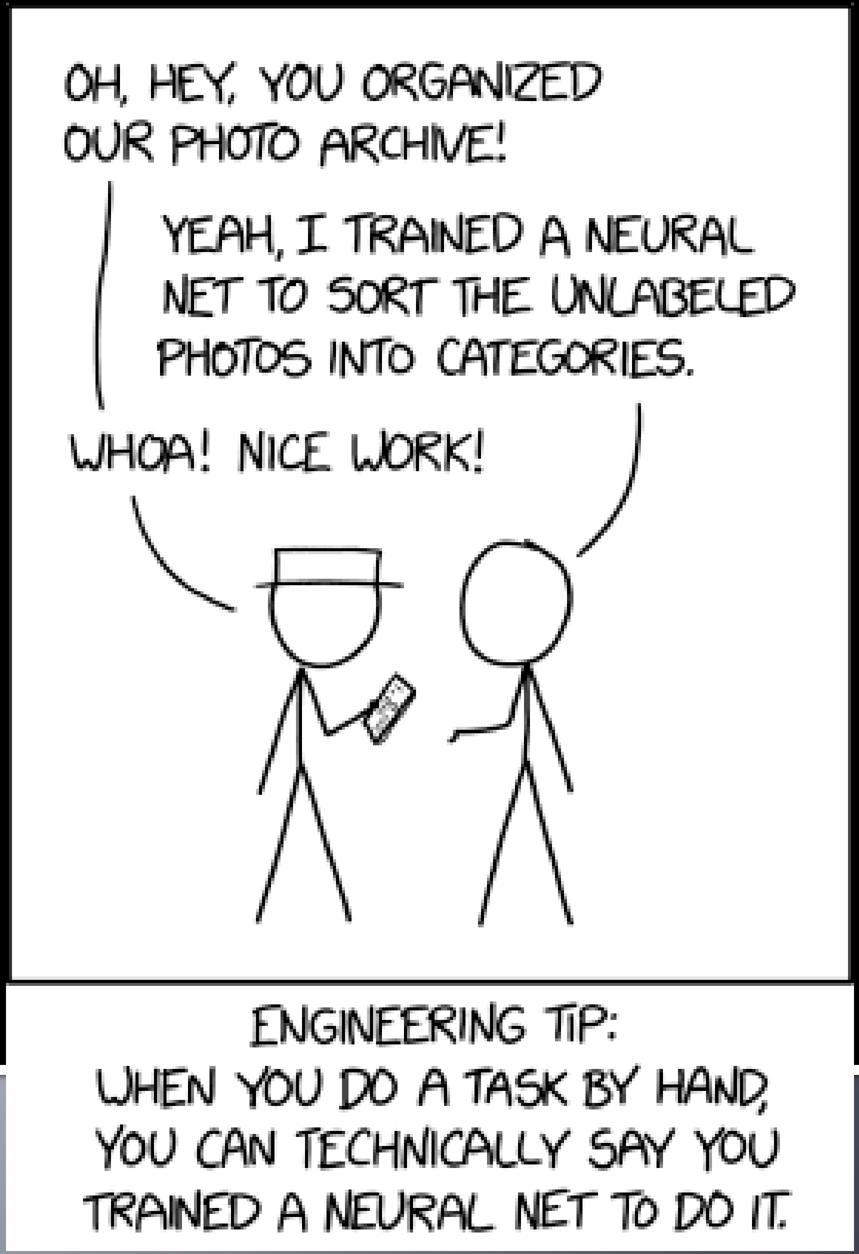
INFO 251: Applied Machine Learning

Foundations of Neural Networks



Course Outline

- Causal Inference and Research Design
  - Experimental methods
  - Non-experiment methods
- **Machine Learning**
  - Design of Machine Learning Experiments
  - Linear Models and Gradient Descent
  - Non-linear models
  - Fairness and Bias in ML
  - **Neural models**
  - Deep Learning
  - Practicalities
  - Unsupervised Learning
- Special topics

Key Concepts (Trees and Forests)

- Decision trees and regression trees
- Recursive tree algorithm
- Choosing splits
- Information gain
- Overfitting and pruning
- Regression trees
- Random forests
- AdaBoost
- Gradient boosting
- Feature Importance

Outline

- **Neural Networks: Motivation and Biology**
- The Perceptron
- Learning perceptron weights
- Multilayer networks
- Learning multilayer weights

Neural Networks: Common Applications

- Basic building block of many modern breakthroughs in AI
  - Computer vision: facial recognition, object detection
  - Natural language: chatbots, GPT, Siri, Google Assistant
  - Autonomous systems: self-driving cars, real-time decisions
  - Healthcare: Medical imaging analysis, drug discovery
  - Finance: Fraud detection, market forecasting
  - Etc., etc.

Neural Networks

- Computational models inspired by the brain
  - Early work dates back to 1940's (McCullough & Pitts, 1943)
  - Idea: mimic how the brain processes information
    - In the hopes that computers can reason as well as human brains
    - ...and perhaps even better!
- So, how do real neurons work? (cue expert)

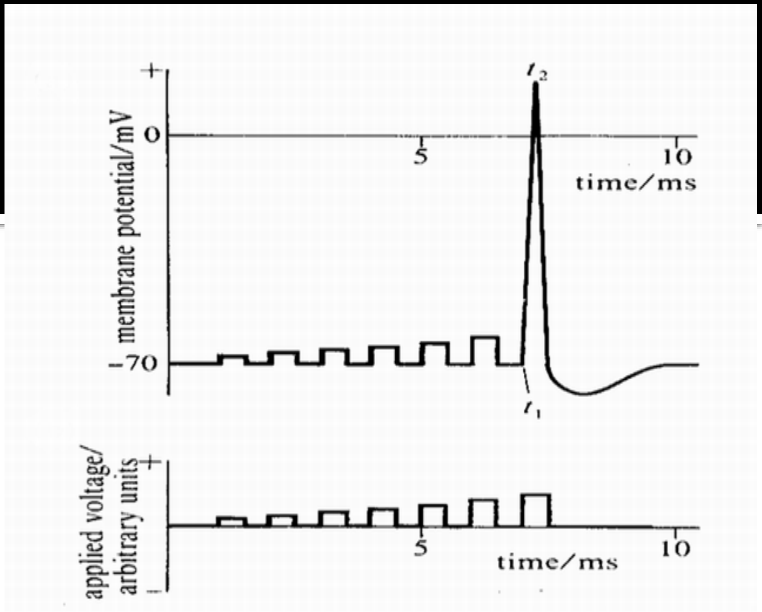
What's a Neuron?

What's a Neuron?

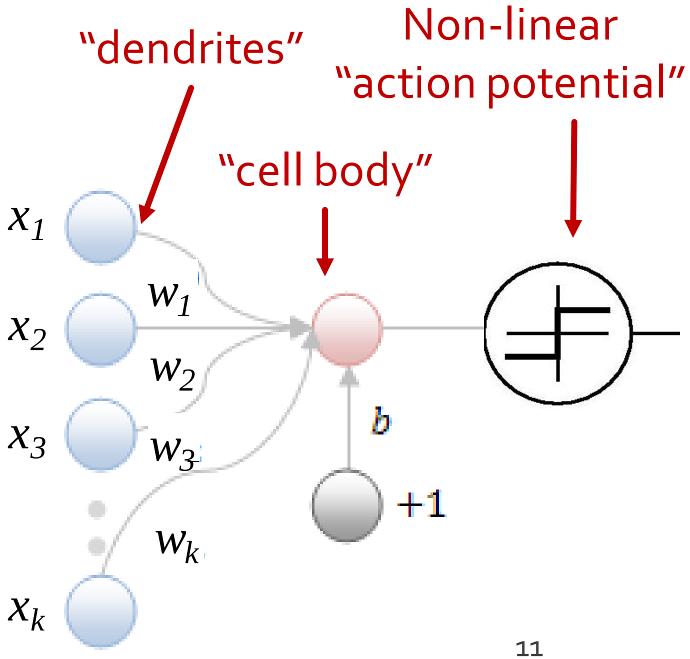
Outline

- Neural Networks: Motivation and Biology
- **The Perceptron**
- Learning perceptron weights
- Multilayer networks
- Learning multilayer weights: Intuition
- Generalizing logistic regression

Creating Artificial Neurons

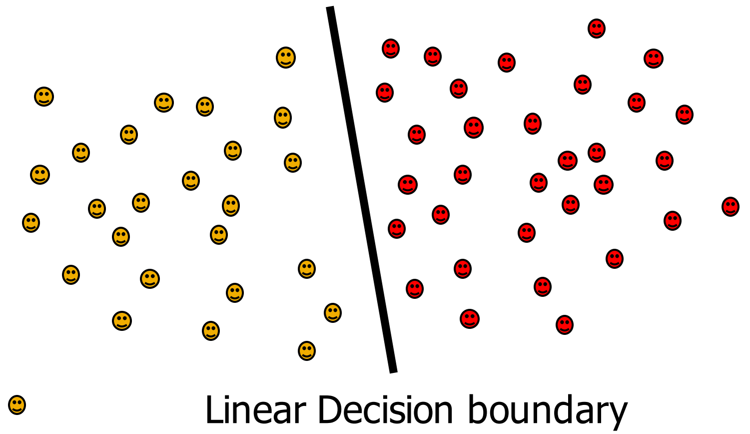


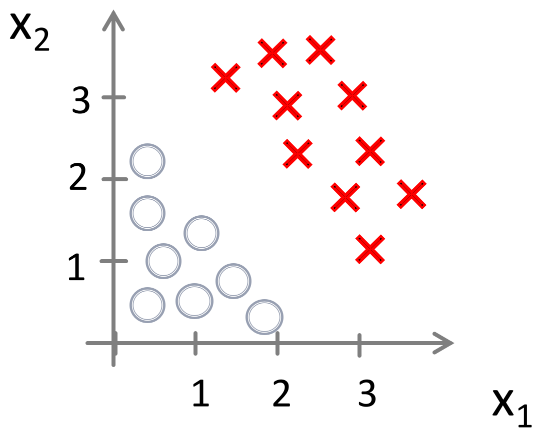
- How to model a neuron?
  - A real neuron fires when membrane potential exceeds a threshold
- The "Perceptron" (Rosenblatt, 1958)
  - Simple binary threshold function
  - Perceptron "fires" if weighted sum of inputs exceeds threshold
    - $h(x) = \text{Sign}(b + \Sigma_{d=1}^{k} w_d x_d)$
    - k weights indexed by w_d
    - Bias term b (or w_0) allows for non-zero threshold



Linearly separable data

- Simple model, but revolutionary at the time!
- Perceptron works with **linearly separable** data
  - i.e., boundary can be specified by hyperplane
  - E.g., $w_0 + w_1 x_1 + \cdots + w_k x_k = 0$
  - Example: what formula defines the separating hyperplane for these data?

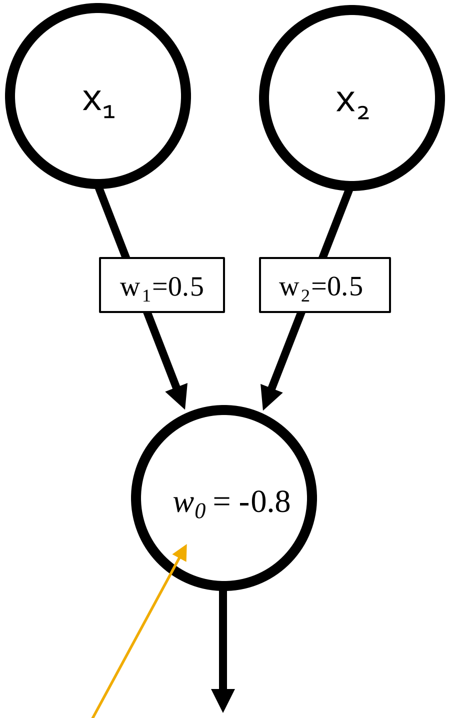




Perceptron: Examples

- A perceptron for AND:

x_1	x_2	y
1	1	T
1	0	F
0	1	F
0	0	F



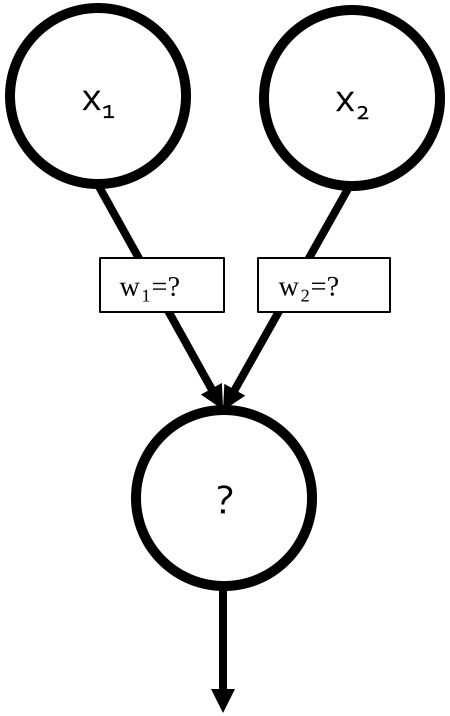
  - Two weights and intercept:
    - $h(x_i) = w_0 + w_1 x_{i1} + w_2 x_{i2}$
  - One solution:
    - *w_1=0.5, w_2=0.5 , w_0=-0.8*
  - *Can you find another one?*

Note: You'll sometimes see a threshold T used instead of the bias w_0, such that T = -w_0 (in this example, T=0.8)

Perceptron: Your turn

- A perceptron for OR:
  - Two weights and intercept:
    - $h(x_i) = w_0 + w_1 x_{i1} + w_2 x_{i2}$
  - Find possible weights w_0, w_1, w_2

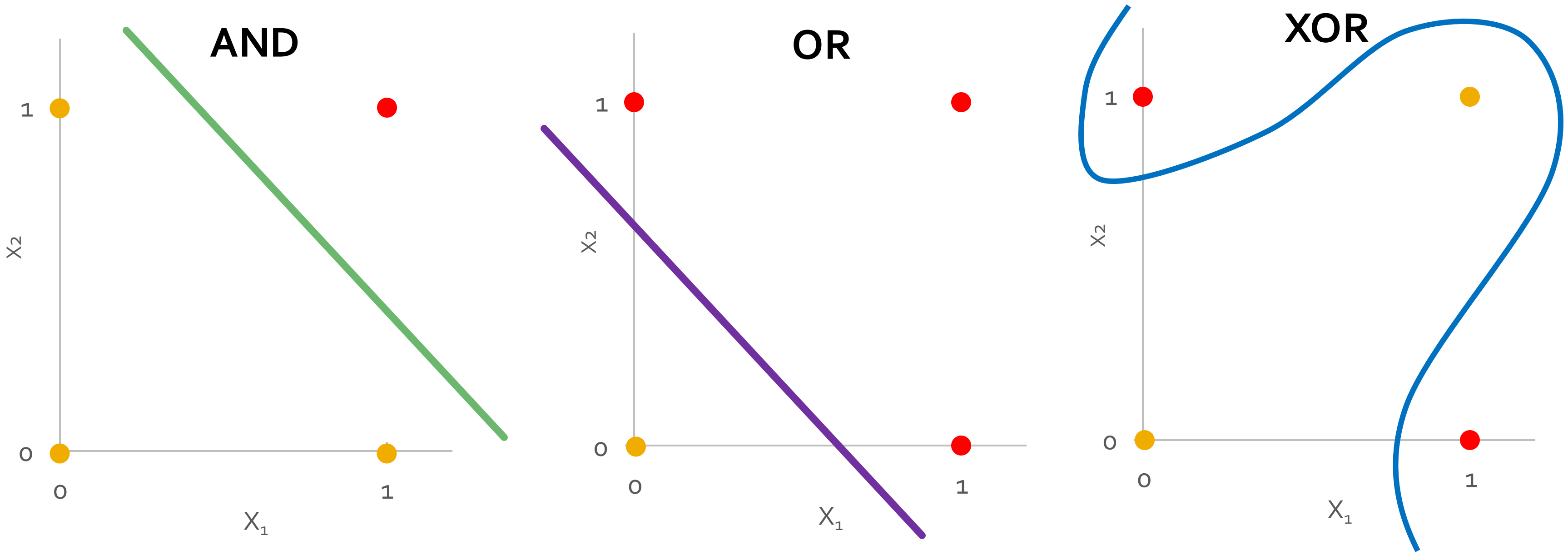
x_1	x_2	y
1	1	T
1	0	T
0	1	T
0	0	F



Perceptron: Examples

- You've seen AND and OR
- A perceptron for XOR?
  - Impossible! (Minsky & Papert 1969) → Why?
  - XOR is not **linearly separable**

x_1	x_2	y
1	1	F
1	0	T
0	1	T
0	0	F



Outline

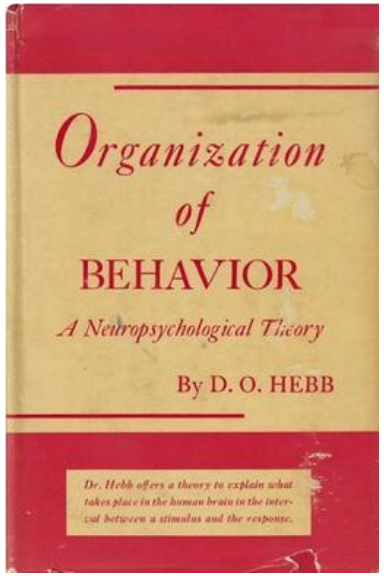
- Neural Networks: Motivation and Biology
- The Perceptron
- **Learning perceptron weights**
- Multilayer networks
- Learning multilayer weights: Intuition
- Generalizing logistic regression

Learning weights

- Given we have input and output (for instance, a truth table), how do we learn the weights?

- Early insight: Hebbian Learning and Synaptic Plasticity (Hebb, 1949)
  - "Neurons that fire together, wire together"

- Modern networks learn using a variety of ways
  - We'll start with Rosenblatt's algorithm (1958)



Learning weights (Rosenblatt)

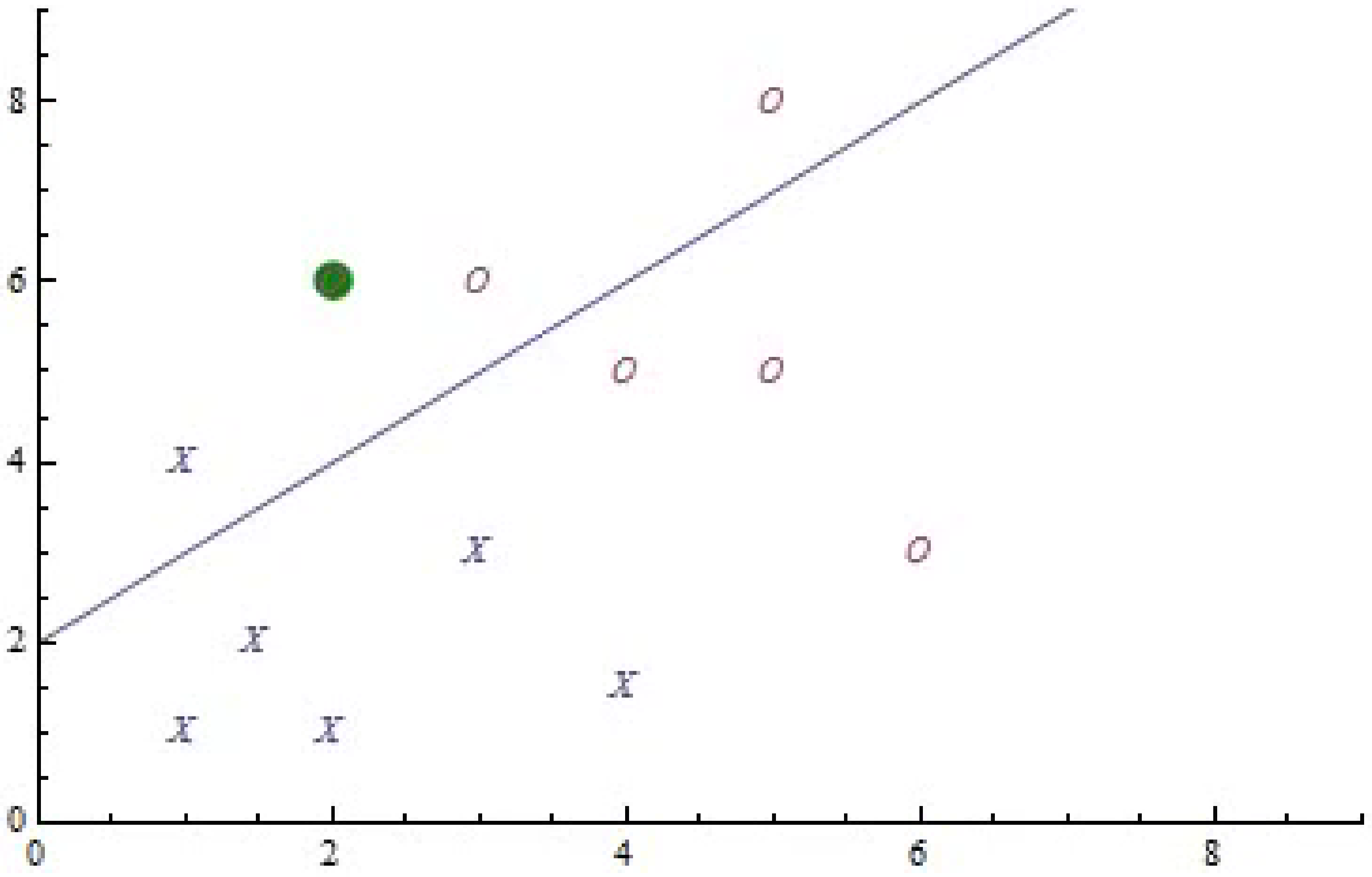
- Rosenblatt's Algorithm (perceptron):

```
initialize weights randomly

while termination condition is not met:
    initialize Δw_j=0

    for each training example (X_i,Y_i):
        compute predicted output Ŷ_i
        foreach weight w_j:
            Δw_j= Δw_j+η(Y_i-Ŷ_i)X_i

    for each weight w_j:
        w_j = w_j + Δw_j
```

"error-driven" learning

Learning rate

Perceptron: In action



Who cares?

- Rosenblatt proved the algorithm is guaranteed to converge as long as:
  - Training data are linearly separable
  - Learning rate is sufficiently small
    - (In the proof, it has to be infinitesimally small)

Training Rule vs. Gradient Descent

- This looks a lot like gradient descent
- Are these approaches different?
  - Training Rule (Rosenblatt)
    - $\Delta w_j = \Delta w_j + \eta\,(Y_i - \hat{Y}_i)\,X_i$
  - Gradient Descent w/ Logistic Regression
    - $\beta \leftarrow \beta + R\left(Y_i - \hat{Y}_i\right)X_i$

- The key is the $\hat{Y}_i$
  - Perceptron: $\hat{Y}_i$ is a step function, either 0 or 1
    - G.D. requires convex surface, not a step function
  - Logit: $\hat{Y}_i$ is a smooth, continuous function

Training Rule vs. Gradient Descent

- Perceptron Training Rule
  - Guaranteed to work if data are linearly separable
  - Requires sufficiently small learning rate η

- Training with Gradient Descent
  - With convex loss…
  - Guaranteed to converge to minimum error
  - Works when data contains noise
  - Works when data are not linearly separable

Perceptron: Summary

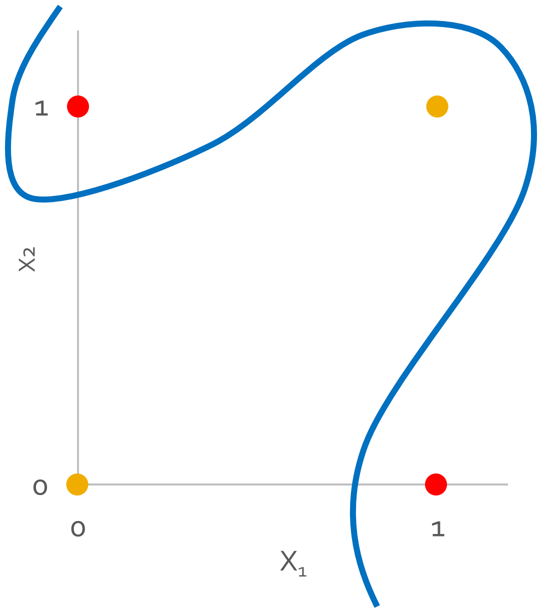
- **Online** algorithm: only considers one instance at a time
- **Error-driven**: Only updates on failure
- Guaranteed to converge if solution exists
- But boundary is **linear**

Outline

- Neural Networks: Motivation and Biology
- The Perceptron
- Learning perceptron weights
- **Multilayer networks**
- Learning multilayer weights: Intuition
- Generalizing logistic regression

Limitations of the Perceptron

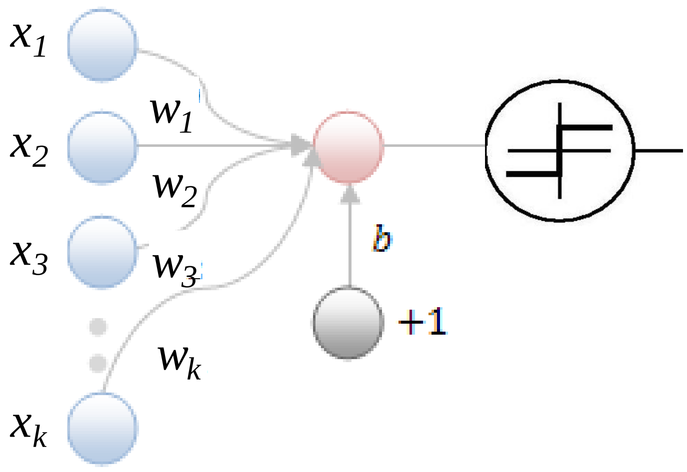
- Only works with linearly separable data
- Only works if learning rate is small enough (Rosenblatt's proof)
- These sort of problems led to "long winter" (1980's)



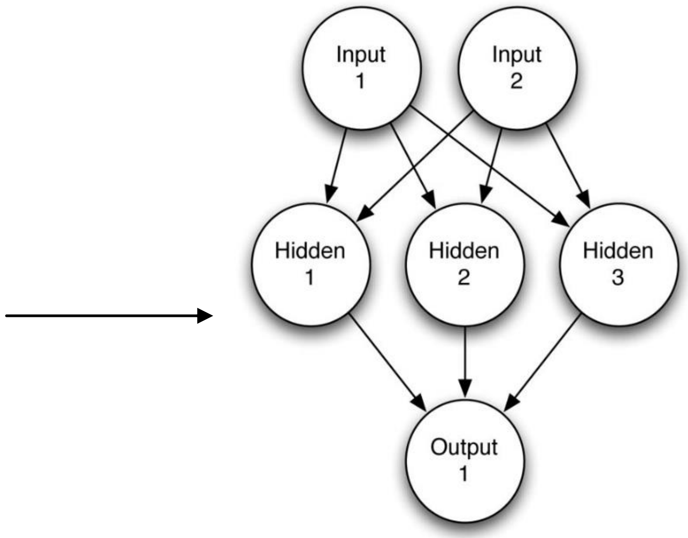
x_1	x_2	y
1	1	-1
1	0	1
0	1	1
0	0	-1

Multilayer Networks

- Single-layer networks are limited – they can only learn hyperplanes
  - Most real-world problems are more complex



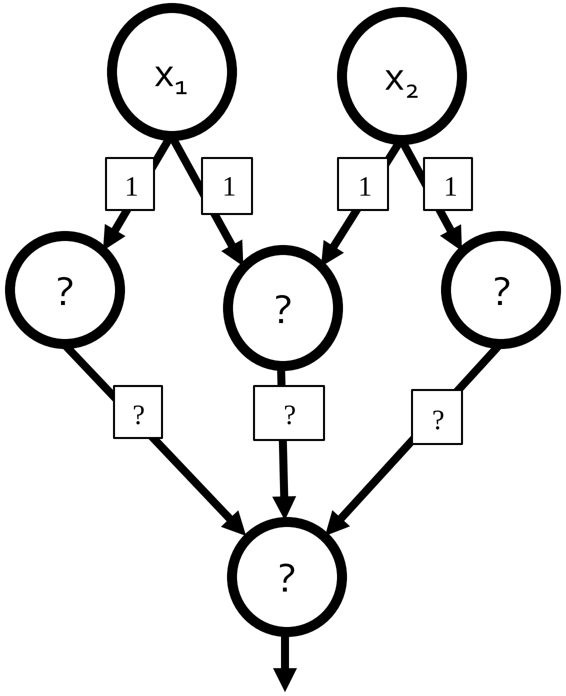
- What if we layer neurons?
  - **Multi-Layer Perceptron (MLP)**: an input layer, one or more hidden layers, and an output layer
  - E.g., two-layer network (two layers of weights)
  - This allows for *very* powerful and complex computation!

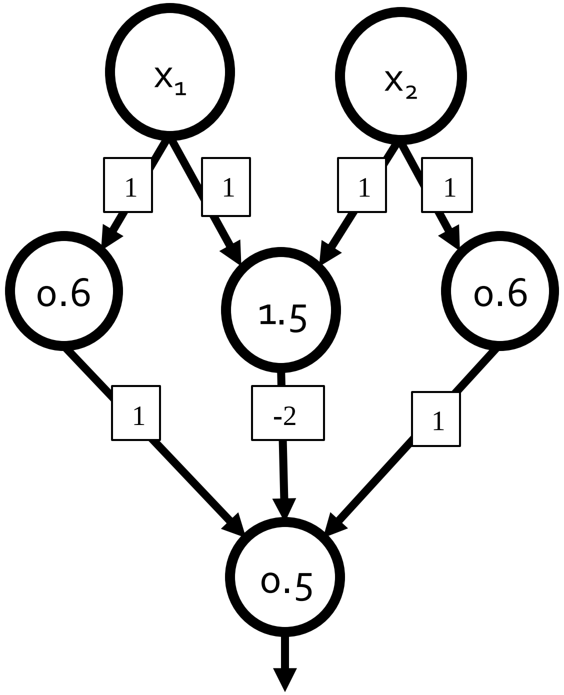


Nonlinearity

X1	X2	Z
1	1	-1
1	0	1
0	1	1
0	0	-1

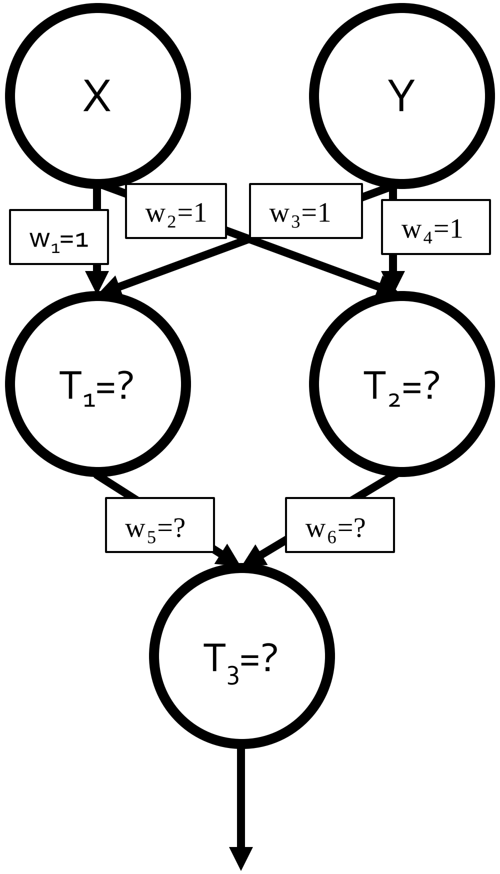
- We saw that XOR can't be solved with a single perceptron
- How about XOR with an MLP?
- Here's one example:





Your Turn: XOR

- Can you find weights that complete this XOR MLP?



Universal Approximation Theorem

- Two-Layer Networks are Universal Function Approximators)
  - Let F be a continuous function on a bounded subset of D-dimensional space. Then there exists a two-layer neural network F′ with a finite number of hidden units that can approximate F arbitrarily well. Namely, for all x in the domain of F,

$$|F(x) - F'(x)| < \varepsilon$$

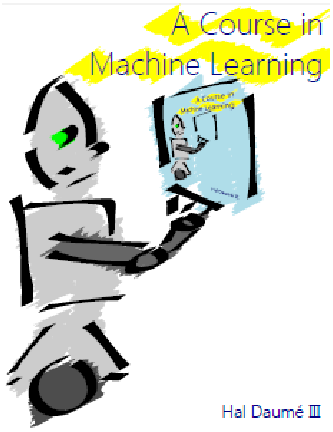
- i.e., "two-layer networks can approximate any function"
- But we still might want more than two layers
  - Fewer neurons, time to learn, time to compute, etc.

Universal Approximation Theorem

- This is a powerful theorem, but…
  - "Just because a function can be represented does not mean it can be learned"
- Learning may require:
  - Insane complexity
  - Insane amounts of data
  - Insane computational resources
- Even if learned, the resulting network can lead to overfitting

For Next Class:

- Read:
  - Daume, chapter 10



- Good luck finishing Problem Set 4!