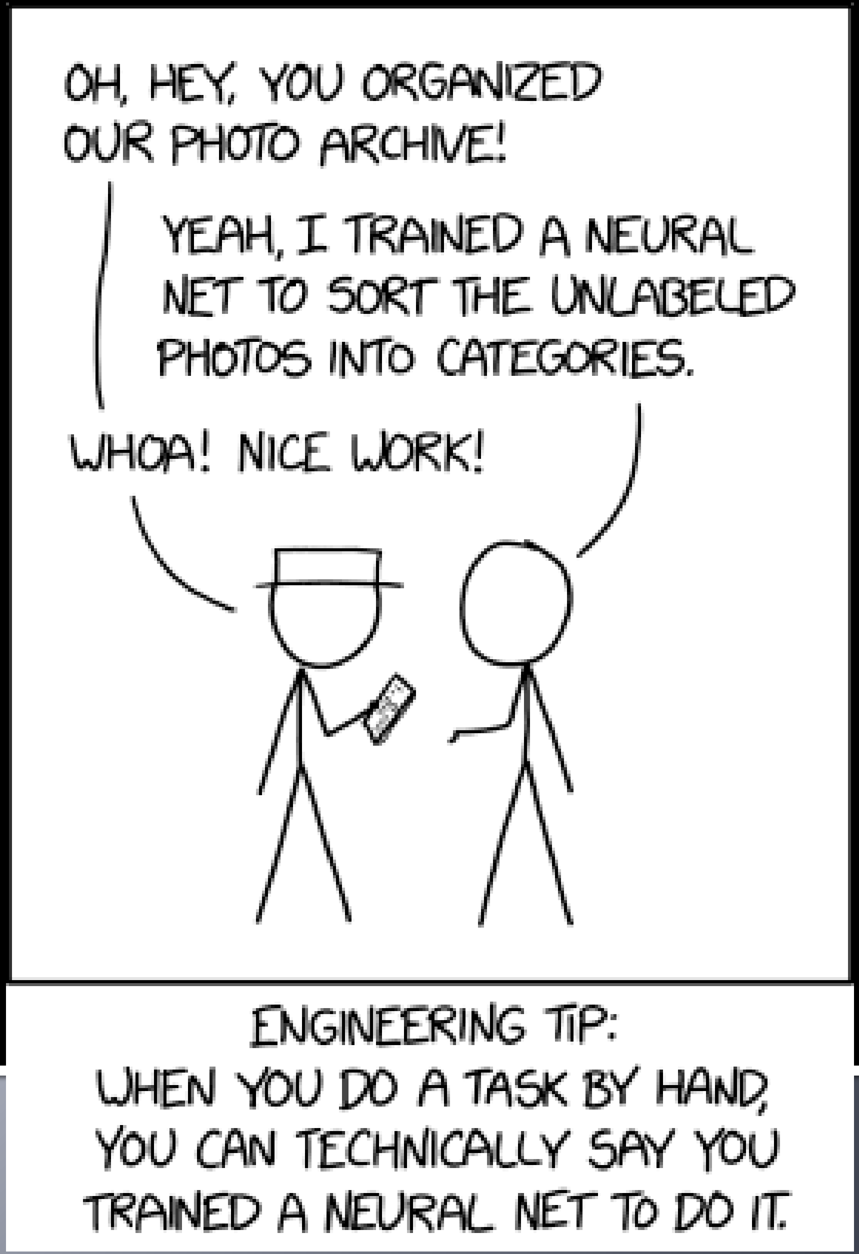
INFO 251: Applied Machine Learning

Neural Networks, part 1



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
  - **Neural models**
  - Unsupervised Learning
  - Practicalities, Fairness, Bias
- Special topics

Key Concepts (last two lectures)

- 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 weights
- Multilayer networks
- Backpropagation
- Summary

Neural Networks

- Computational models inspired by the brain
  - Mimic how the brain processes information
  - In the hopes that computers can reason as well as human brains
  - And perhaps even better
  - ➔ Build machine learning algorithms based on the most sophisticated learner out there!

Engineering Brains

- What's a Brain?
  - Composed of 100 B neurons – we'll come back to this
  - Switching time: 0.001 seconds
  - *10,000 – 100,000 connections per neuron*
- Scene recognition
  - 0.1 seconds => Parallel computation
- Compare to transistor:
  - 100,000,000,000 transistors in modern chip (human x 10^3)
  - *Switching time: 0.000000001 seconds* (human x 10^7)
  - 10-100 connections per transistor

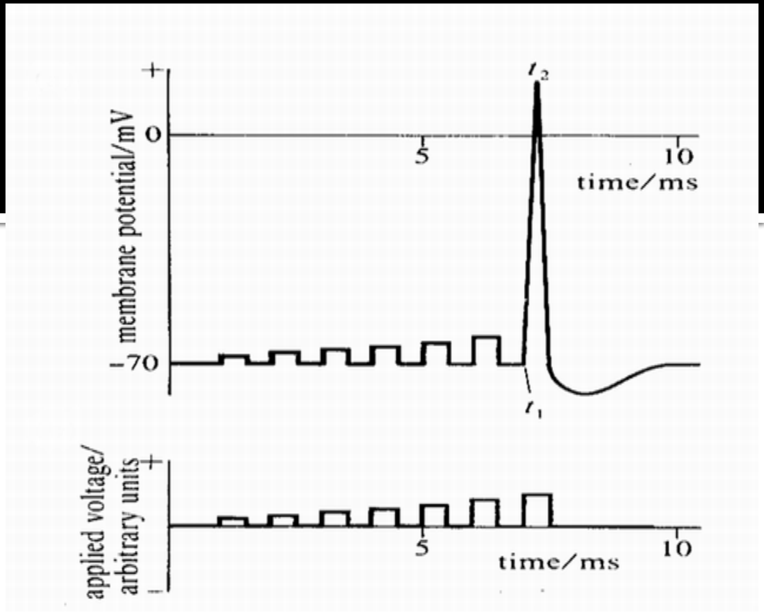
What's a Neuron?

What's a Neuron?

Outline

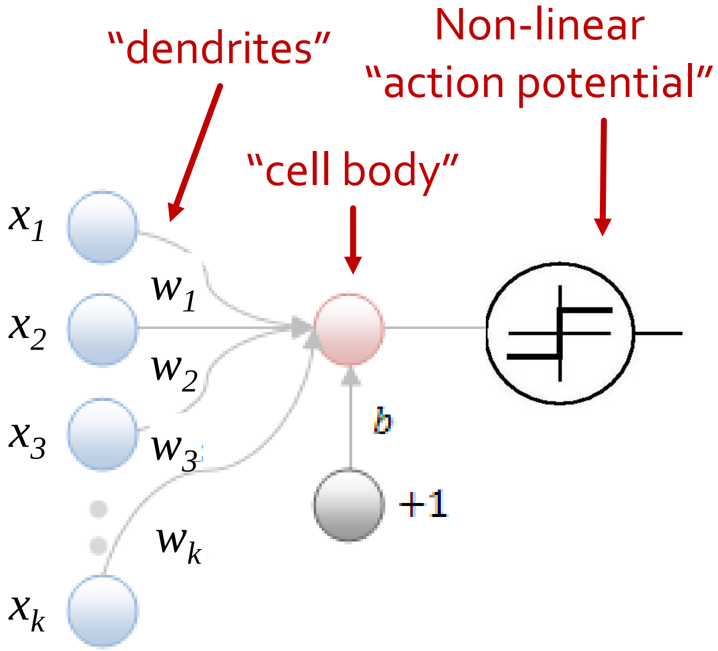
- Neural Networks: Motivation and Biology
- **The Perceptron**
- Learning weights
- Multilayer networks
- Backpropagation
- Summary

Creating Artificial Neurons



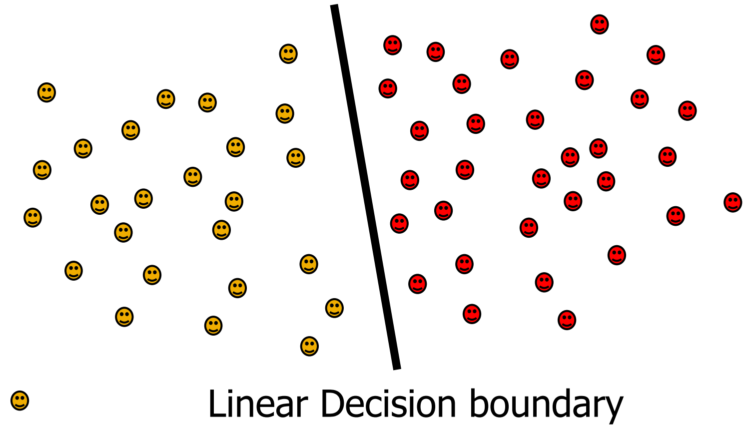
- How to model a neuron?
  - Neuron fires when membrane potential exceeds a threshold

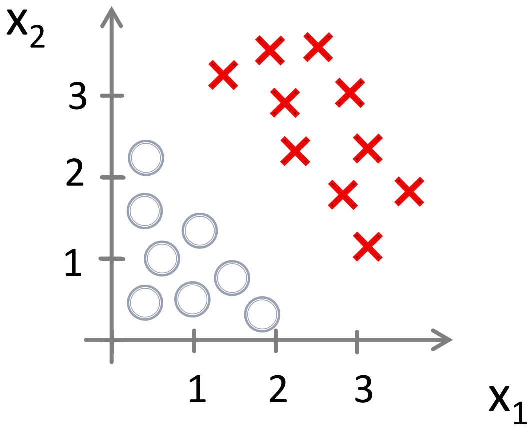
- The "Perceptron"
  - Perceptron "fires" if sum of inputs exceeds threshold
    - $h(x) = \text{Sign}(b + \Sigma_{d=1}^{k} w_d x_d)$
    - k weights indexed by w_d
    - Bias term b (or w_0) allows for non-zero threshold



Linearly separable data

- 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?

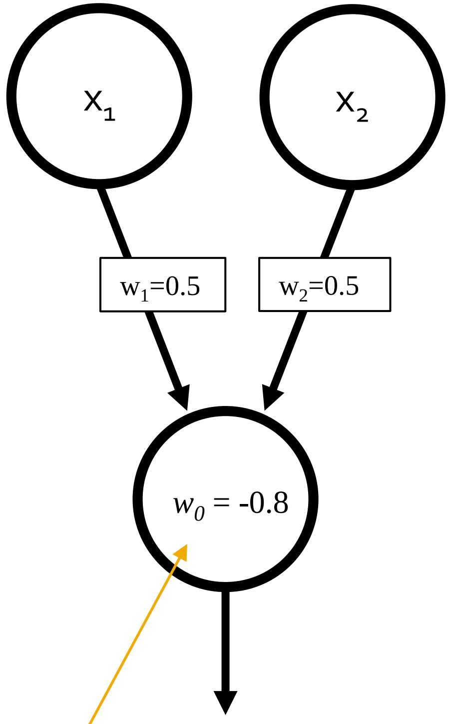




Perceptron: Examples

- A perceptron for AND:

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



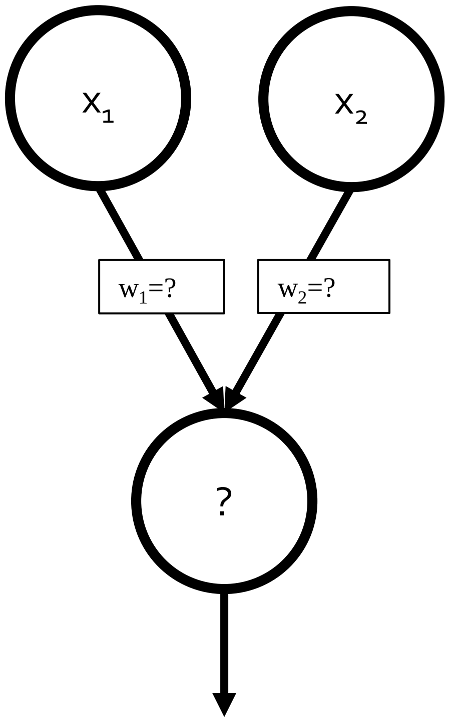
  - Two weights and intercept:
    - $h(x_i) = w_0 + w_1 x_{i1} + w_2 x_{i2}$
  - One solution:
    - w_1=0.5, w_2=0.5 , w_0=-0.8

Note: in drawing these diagrams, we sometimes indicate a threshold T instead of the bias w_0, such that T = $-w_0$

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

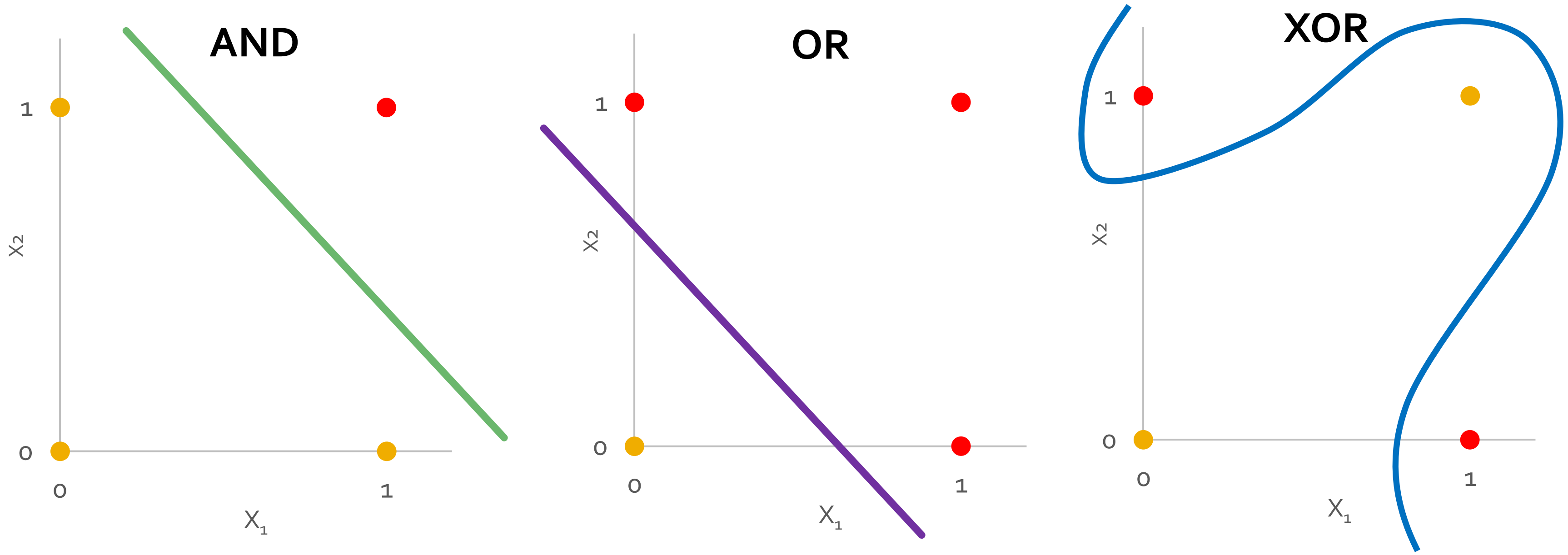
x_1	x_2	y
1	1	T
1	0	T
0	1	T
0	0	F



Perceptron: Examples

- You've seen AND and OR
- A perceptron for XOR?
- Impossible! → 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
- Backpropagation
- Summary

Learning weights

- Given we have input and output (for instance, a truth table), how do we learn the weights?
- In practice, there are several ways
  - We'll start with Rosenblatt's algorithm (circa 1950's)

Learning weights (Rosenblatt)

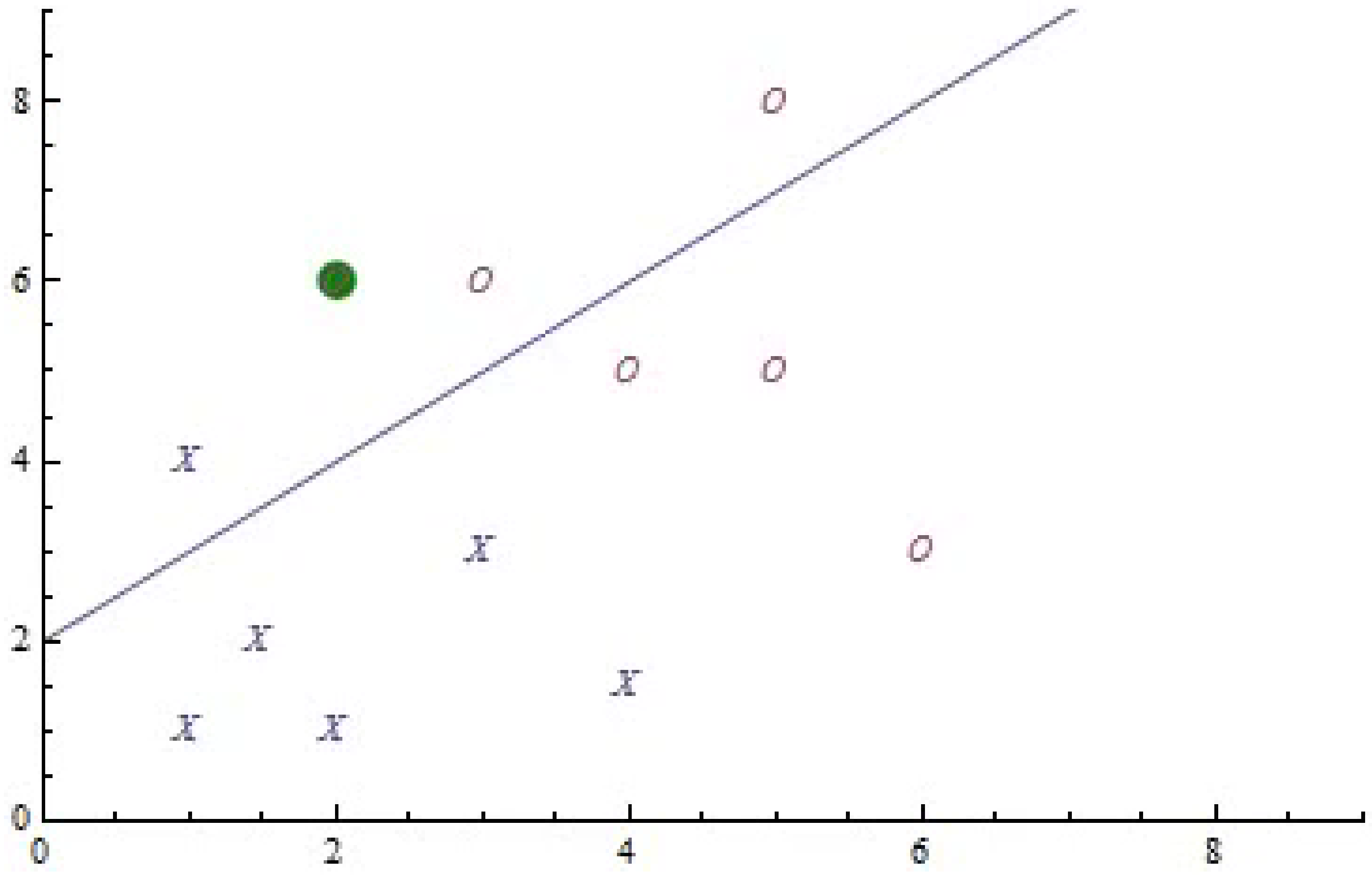
- Rosenblatt's Algorithm (perceptron):

```
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

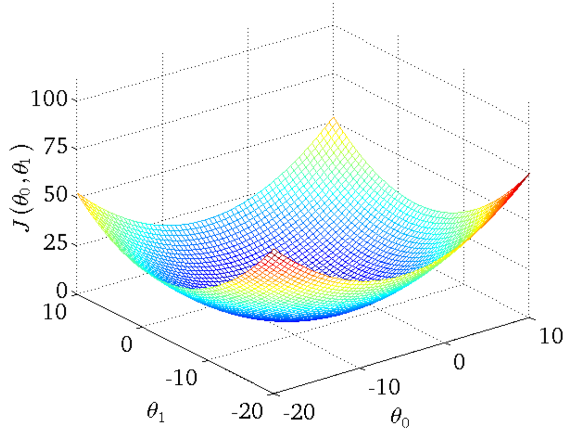


Who cares?

- 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)

Learning weights: Another Approach

- Output is linear function of weights:
  - (Forget about step function for a moment)
  - $\hat{Y}_i = w_0 + w_1 x_{i1} + \cdots + w_n x_{in}$
- Assume error is quadratic function of output
  - $J(\alpha, \beta) = \frac{1}{2N} \sum_{i=1}^{N} \left(Y_i - \hat{Y}_i\right)^2$
- What does this remind you of?
  - Gradient Descent!
  - β <– β – $\frac{R}{N}\left(Y_i - \hat{Y}_i\right) X_i$



Training Rule vs. 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
    - $\beta <- \beta + R(Y_i - \hat{Y}_i)X_i$
- 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

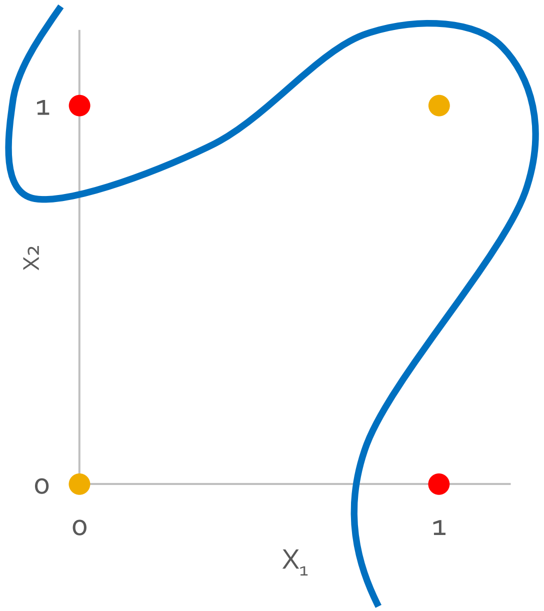
- **Online** algorithm: only considers one instance at a time
- **Error-driven**: Only updates on failure
- Guaranteed to converge if solution exists
- But boundary is **linear**

Outline

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

Limitations of the Perceptron

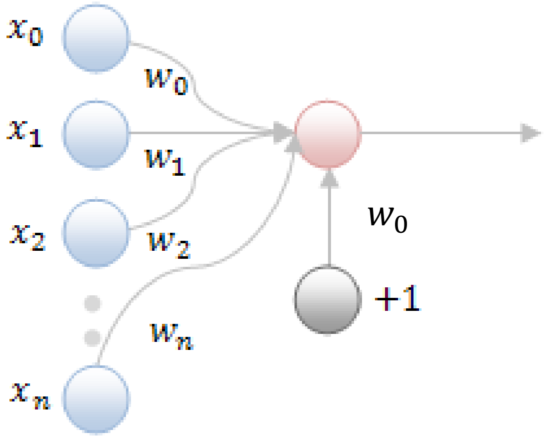
- Only works with linearly separable data
- Only works if learning rate is small enough (Rosenblatt's proof)
- These sort of problems led to "long winter" (1980's)



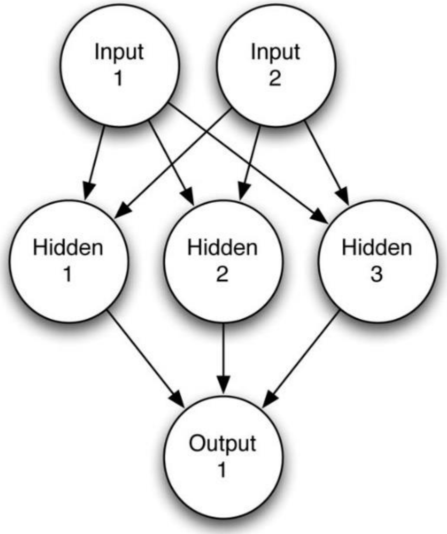
x_1	x_2	y
1	1	-1
1	0	1
0	1	1
0	0	-1

Multilayer Networks

- Single-layer networks are limited – they can only learn hyperplanes



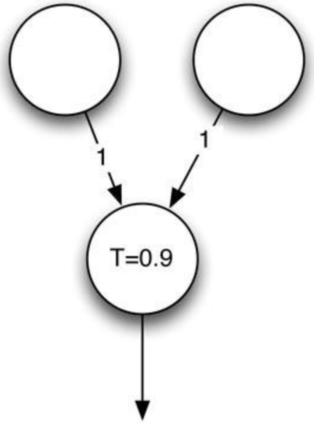
- What if we layer neurons?
  - Two-layer network
  - (two layers of weights)



Nonlinearity

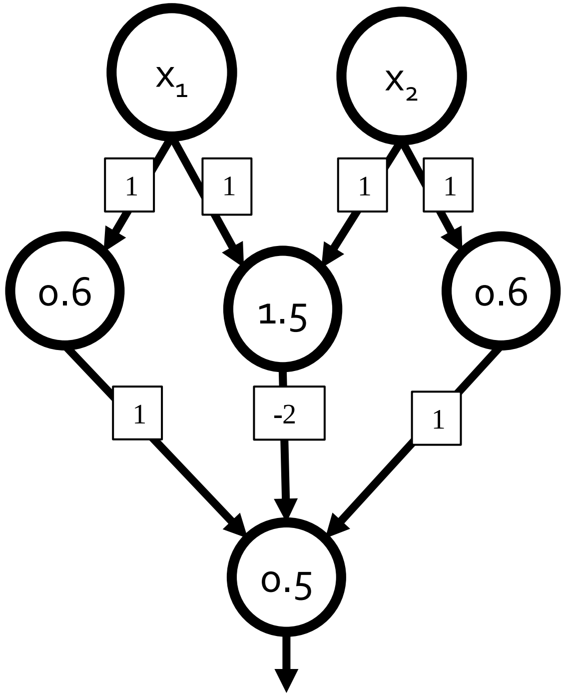
- OR perceptron:
  - $w_1=1, w_2=1, b=-0.9$

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



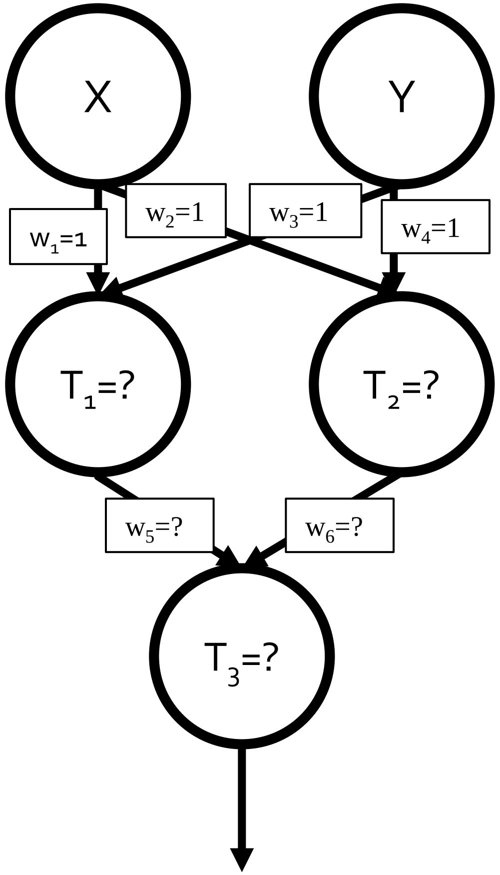
- Two-layer XOR:

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



Your Turn: XOR

- What weights complete the 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 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

Outline

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