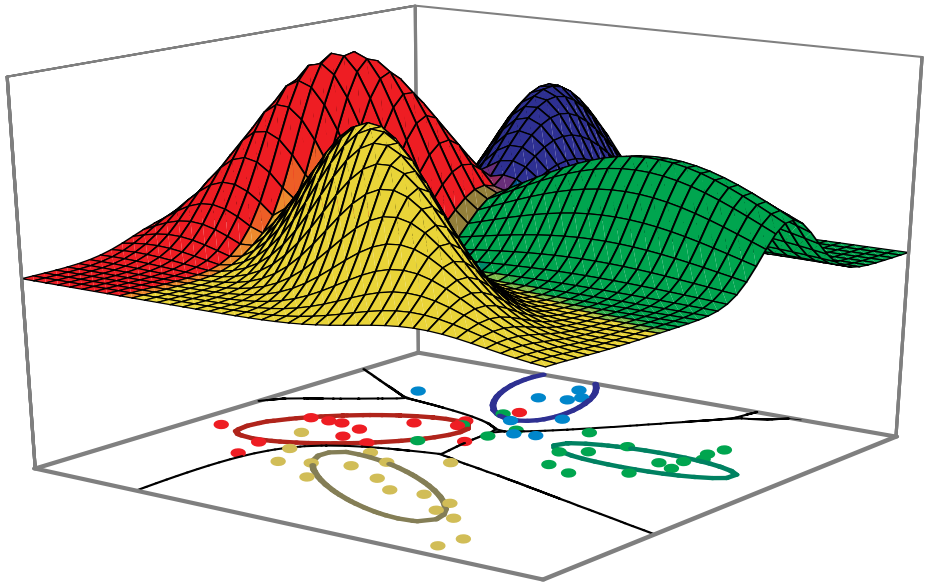
Introduction to Machine Learning
(and data center applications)



David Meyer

dmm@{brocade.com,uoregon.edu,1-4-5.net,…}

July, 2014

Agenda

- Goals for this Talk
- What is Machine Learning?
  - Kinds of Machine Learning
- Machine Learning Fundamentals
  - Shallow dive
  - Regression and Classification – Inductive Learning
  - Focus on Artificial Neural Networks (ANNs)
- A Bit on Unsupervised Learning
- Deep Learning
- Google *Power Usage Effectiveness* (PUE) Optimization Application
- Next Steps

Goals for this Talks

To give us a basic common understanding of machine learning so that we can understand and discuss examples/use cases such as those at the end of this deck

Before We Start

What is the State of the Art in Machine Learning?

- "Building High-level Features Using Large Scale Unsupervised Learning", Andrew Ng, et. al, 2012
  - http://arxiv.org/pdf/1112.6209.pdf
  - Training a *deep neural network*
  - Showed that it is possible to train neurons to be selective for high-level concepts using entirely *unlabeled* data
  - In particular, they trained a deep neural network that functions as detectors for faces, human bodies, and cat faces by training on random frames of YouTube videos (ImageNet[1]). These neurons naturally capture complex in-variances such as out-of-plane and scale invariances.

- Details of the Model
  - Sparse deep auto-encoder (we'll talk about what this is later in this deck)
  - $O(10^9)$ connections
  - $O(10^7)$ 200x200 pixel images, 10^3 machines, 16K cores
    - → Input data in R^{40000}
    - Three days to train
  - 15.8% accuracy categorizing 22K object classes
    - 70% improvement over current results
    - Random guess achieves less than 0.005% accuracy for this dataset

1 http://www.image-net.org/

Agenda

- ~~Goals for this Talk~~
- What is Machine Learning?
  - Kinds of Machine Learning
- Machine Learning Fundamentals
  - Shallow dive
  - Regression and Classification – Inductive Learning
  - Focus on Artificial Neural Networks (ANNs)
- A Bit on Unsupervised Learning
- Deep Learning
- Google PUE Optimization Machine Learning Application
- Next Steps

What is Machine Learning?

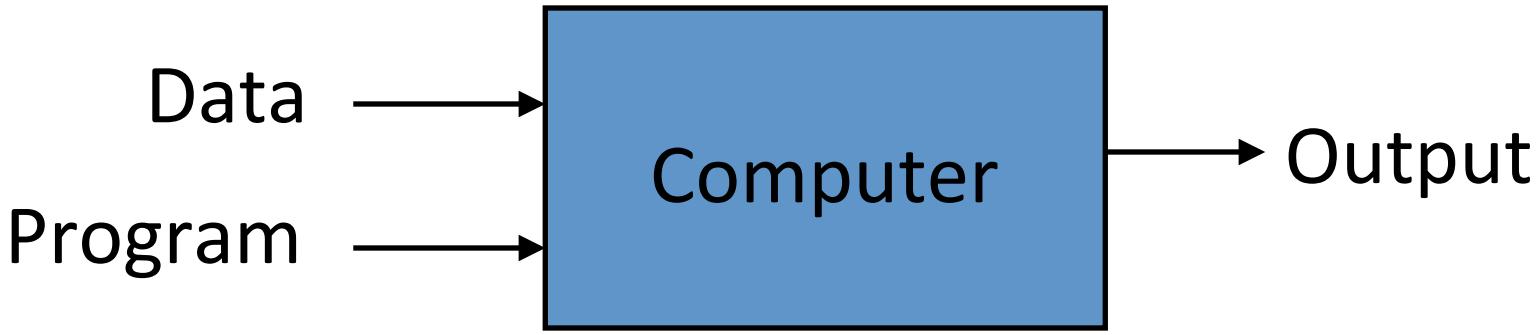
The complexity in traditional computer programming is in the code (programs that people write). In machine learning, algorithms (programs) are in principle simple and the complexity (structure) is in the data. Is there a way that we can automatically learn that structure? That is what is at the heart of machine learning.

-- Andrew Ng

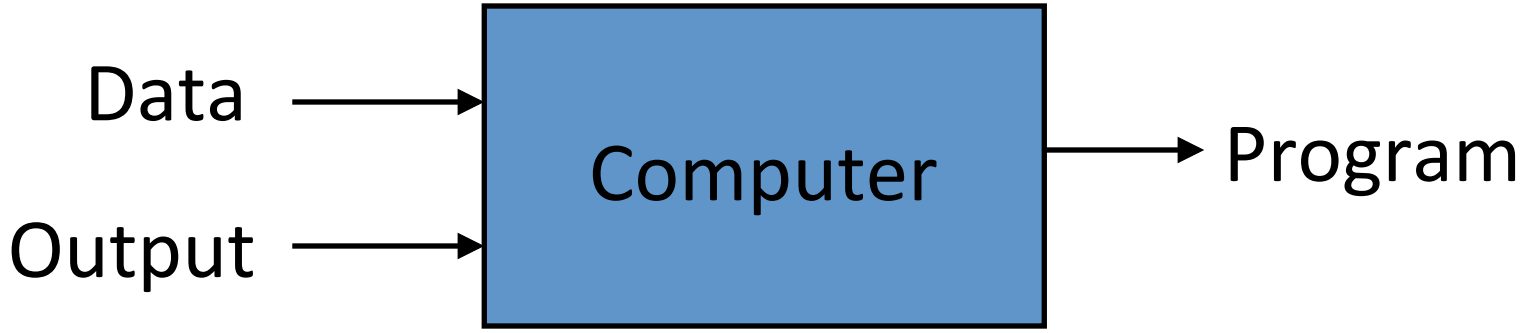
That is, machine learning is the about the construction and study of systems that can learn from data. This is very different than traditional computer programming.

The Same Thing Said in Cartoon Form

Traditional Programming



Machine Learning

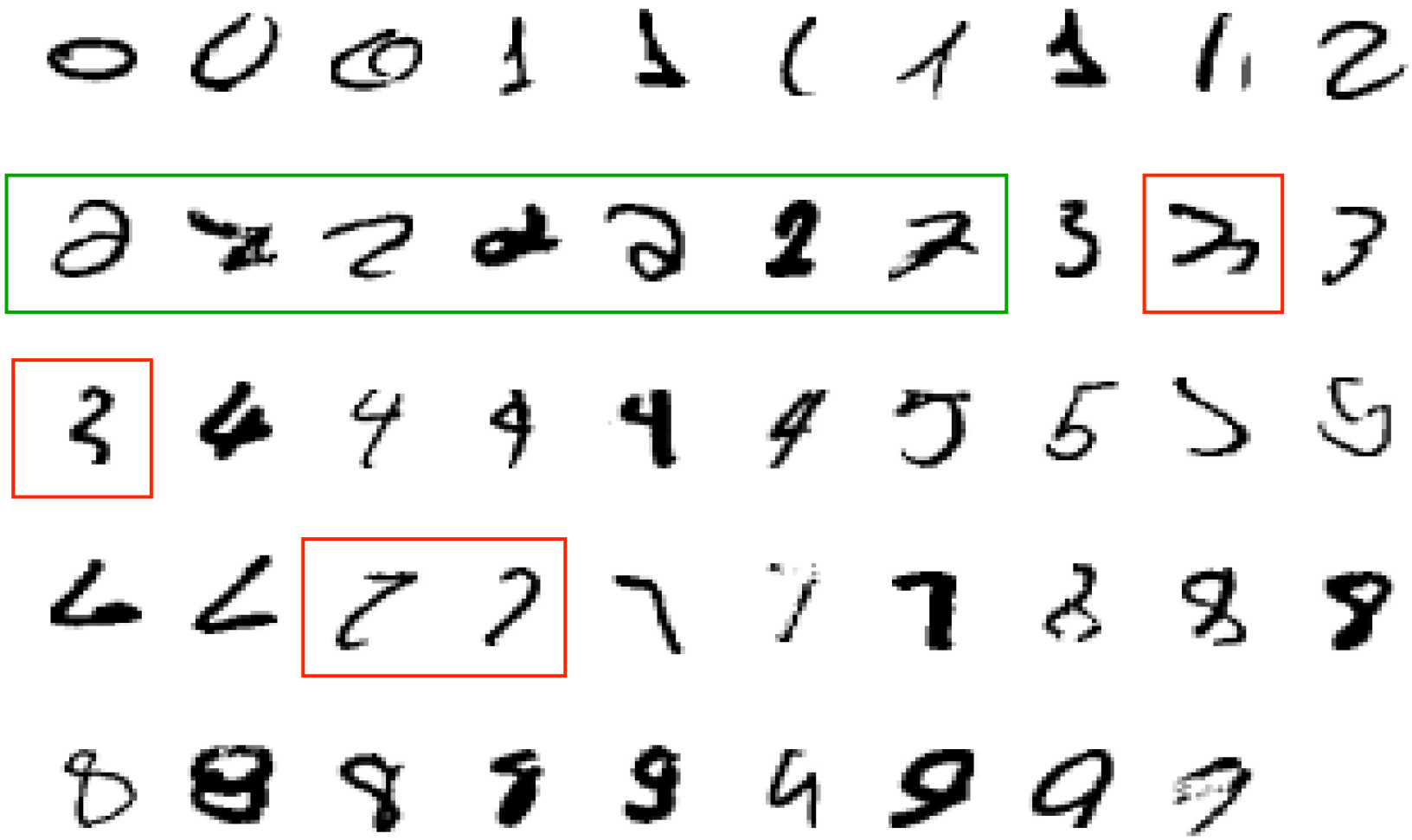


When Would We Use Machine Learning?

- When patterns exists in our data
  - Even if we don't know what they are
    - Or perhaps especially when we don't know what they are
- We can not pin down the functional relationships mathematically
  - Else we would just code up the algorithm
- When we have lots of (unlabeled) data
  - Labeled training sets harder to come by
  - Data is of high-dimension
    - High dimension "features"
    - For example, sensor data
  - Want to "discover" lower-dimension representations
    - Dimension reduction
- Aside: Machine Learning is heavily focused on implementability
  - Frequently using well know numerical optimization techniques
  - Lots of open source code available
    - See e.g., libsvm (Support Vector Machines): http://www.csie.ntu.edu.tw/~cjlin/libsvm/
    - Most of my code in Octave: http://en.wikibooks.org/wiki/Octave_Programming_Tutorial

Why Machine Learning Is Hard

What is a “2”?



Examples of Machine Learning Problems

- Pattern Recognition
  - Facial identities or facial expressions
  - Handwritten or spoken words (e.g., Siri)
  - Medical images
  - Sensor Data/IoT
- Optimization
  - Many parameters have “hidden” relationships that can be the basis of optimization
- Pattern Generation
  - Generating images or motion sequences
- Anomaly Detection
  - Unusual patterns in the telemetry from physical and/or virtual plants (e.g., data centers)
  - Unusual sequences of credit card transactions
  - Unusual patterns of sensor data from a nuclear power plant
    - or unusual sound in your car engine or …
- Prediction
  - Future stock prices or currency exchange rates

Spectrum of Machine Learning Tasks

Statistics----------Artificial Intelligence

- Low-dimensional data (e.g. less than 100 dimensions)
- Lots of noise in the data
- There is not much structure in the data, and what structure there is, can be represented by a fairly simple model.
- The main problem is distinguishing true structure from noise.

- High-dimensional data (e.g. more than 100 dimensions)
- The noise is not sufficient to obscure the structure in the data if we process it right.
- There is a huge amount of structure in the data, but the structure is too complicated to be represented by a simple model.
- The main problem is figuring out a way to represent the complicated structure that allows it to be learned.

Kinds of Learning

- **Supervised (inductive) learning**
  - Training data includes desired outputs
  - "Labeled" data
  - All kinds of "standard" training data sets available, e.g.,
    - http://archive.ics.uci.edu/ml/ (UCI Machine Learning Repository)
    - http://yann.lecun.com/exdb/mnist/ (subset of the MNIST database of handwritten digits)
    - http://deeplearning.net/datasets/
    - …

- **Unsupervised learning**
  - Training data does not include desired outputs
  - "Unlabeled" data

- **Semi-supervised learning**
  - Training data includes a few desired outputs

- **Reinforcement learning**
  - Rewards from sequence of actions

Agenda

- ~~Goals for this Talk~~
- ~~What is Machine Learning?~~
  - ~~Kinds of Machine Learning~~
- Machine Learning Fundamentals
  - Shallow dive
  - Inductive Learning: Regression and Classification
  - Focus on Artificial Neural Networks (ANNs)
- A Bit on Unsupervised Learning
- Deep Learning
- Google PUE Optimization Machine Learning Application
- Next Steps

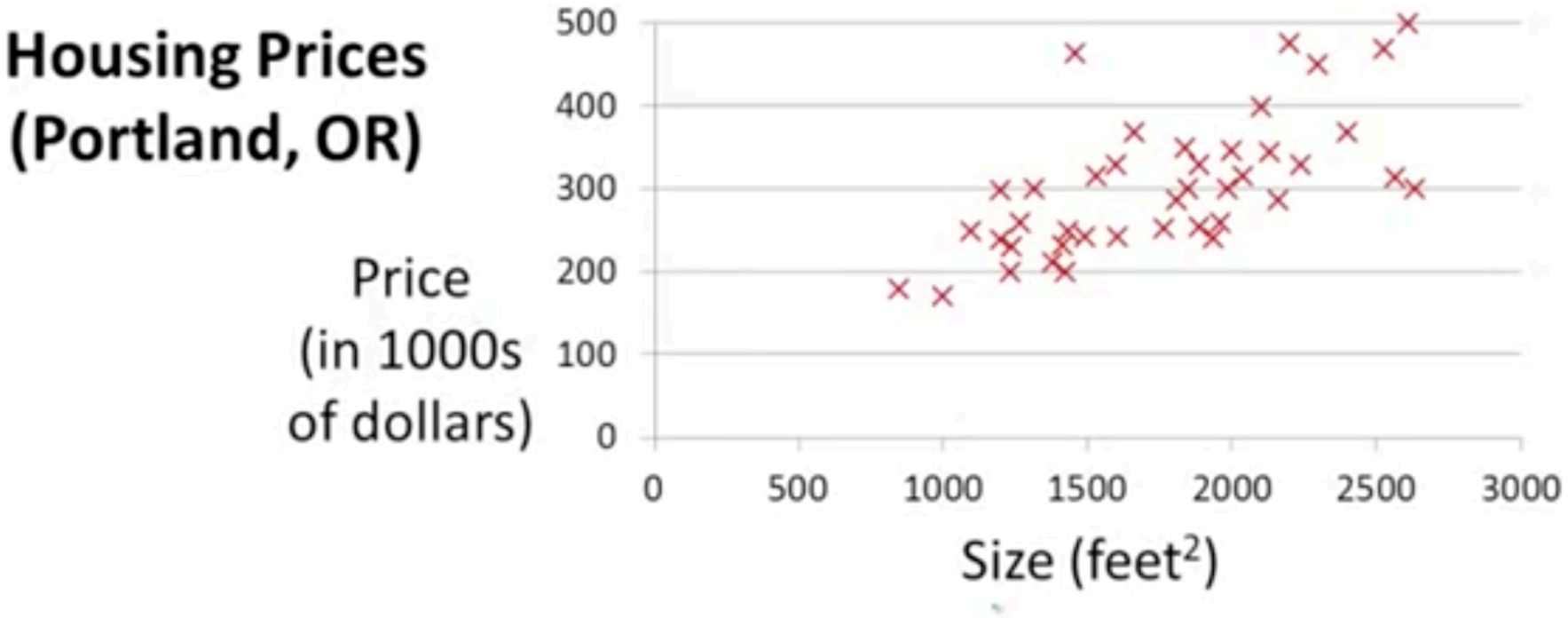
So What Is Inductive Learning?

- Basically

- Given examples of a function *(x, f(x))*
  - Supervised learning (because we're given *f(x)*)
  - Don't explicitly know *f* (rather, trying to fit a model to the data)
  - Labeled data set (i.e., the *f(x)*'s)
  - Training set may be noisy, e.g.*, (x, (f(x) + ε))*
  - Notation: $(x_i, f(x_i))$ denoted $(x^{(i)}, y^{(i)})$
  - $y^{(i)}$ sometimes called t_i (t for "target")

- Predict function *f(x)* for new examples *x*
  - Discrimination/Prediction (Regression): *f(x)* continuous
  - Classification*: f(x)* discrete
  - Estimation*: f(x)* = P(Y = c|*x*) for some class c

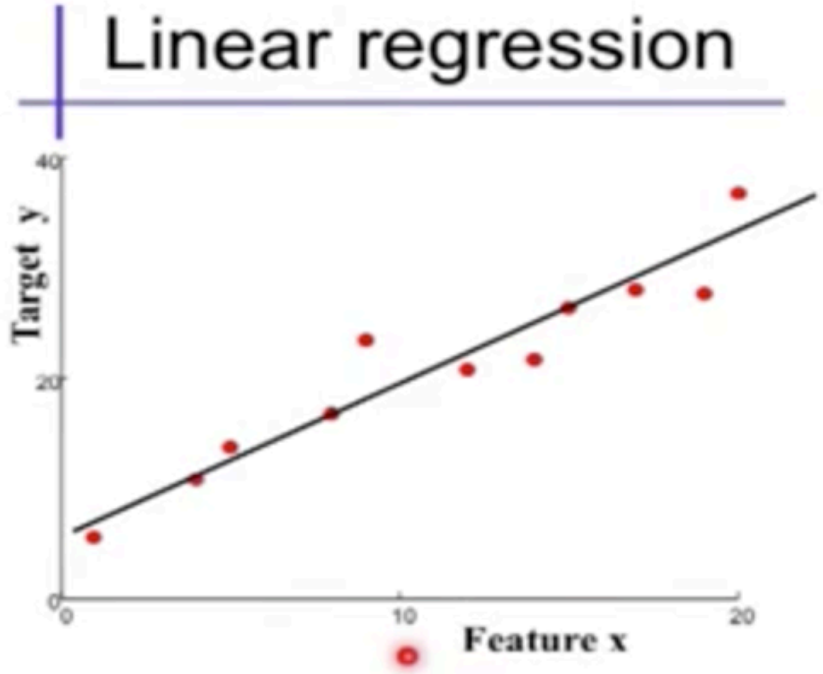
Introduction to Supervised Learning
Linear Regression



Training set is denoted $\{(x^{(0)},y^{(0)}), (x^{(1)},y^{(1)}), \ldots, (x^{(m)},y^{(m)})\}$
m is the number of training examples
x's are the "input variables" or "features"
y's are the "output variables" or "targets"
Predict the value of the output given new values of x

Linear Regression

(Prediction Problem with Linear Features, or Least Mean Squares (LMS) Learning)



"Predictor":
Evaluate line:

$$r = \theta_0 + \theta_1 x_1$$

return r

General form:
$\theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$
x is n-dimensional

The Θ_i are the "parameters"

- Define form of function f(x) explicitly
- Find a good f(x) within that family
- f(x) is typically called the *hypothesis* and denoted $h_\theta(x)$
- For linear regression: $h_\theta(\mathbf{x}) = \theta^T \mathbf{x}$

Linear Regression, cont

Training Set

Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

Linear Regression Summary

(Where is the machine learning?)

Hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Parameters:

$$\theta_0, \theta_1$$

Cost Function:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Learning as optimization (Empirical Risk Minimization)

More Formally
Empirical Risk Minimization

- Empirical risk minimization
  - framework to design learning algorithms

$$\arg\min_{\boldsymbol{\theta}} \frac{1}{T} \sum_{t} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) + \lambda \Omega(\boldsymbol{\theta})$$

  - $l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$ is a loss function (loss function also called "cost function" denoted $J(\theta)$)
  - $\Omega(\boldsymbol{\theta})$ is a regularizer (penalizes certain values of $\boldsymbol{\theta}$)
- Learning is cast as optimization
  - ideally, we'd optimize classification error, but it's not smooth
  - loss function is a surrogate for what we truly should optimize (e.g. upper bound)

Any interesting cost function is not differentiable and non-convex

Solving the Cost Minimization Problem
Gradient Descent – Basic Idea

Have some function $J(\theta_0, \theta_1)$

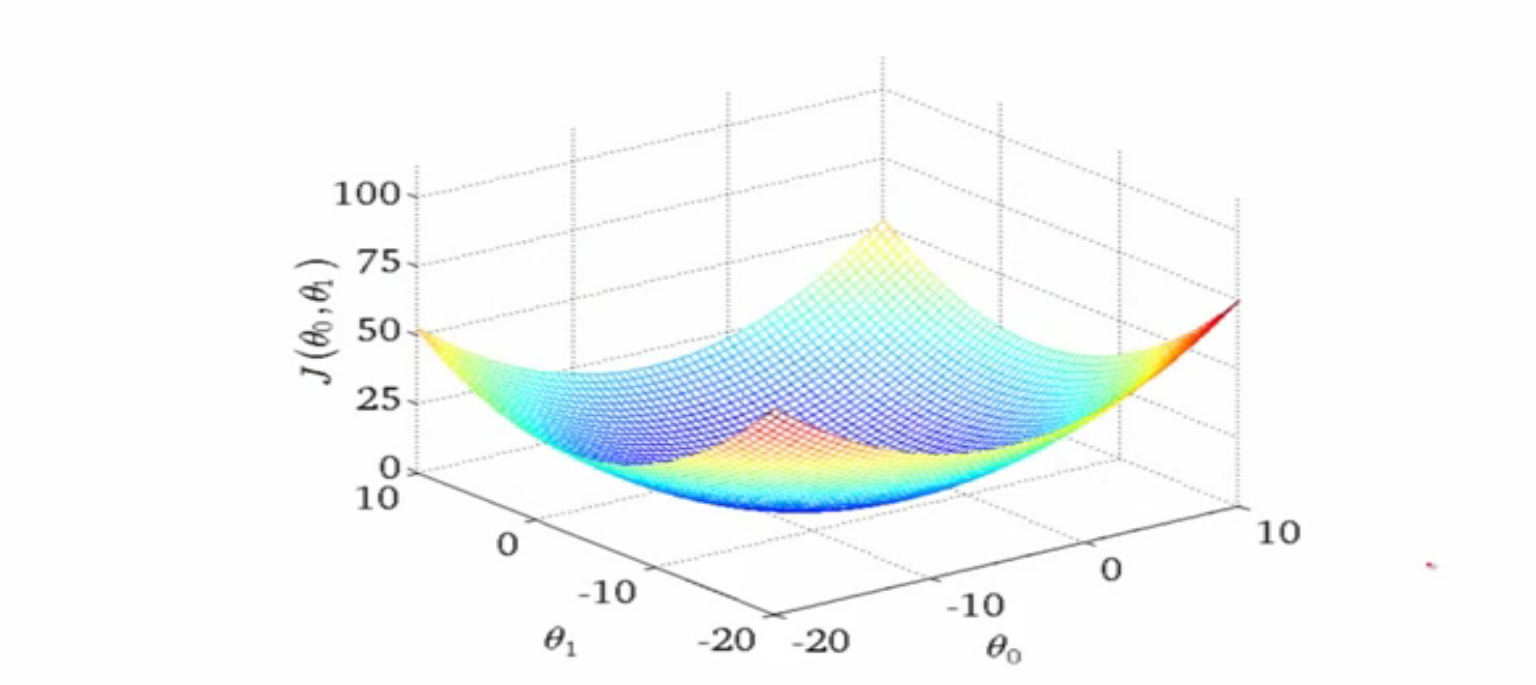
Want $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

Outline:

- Start with some θ_0, θ_1
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$

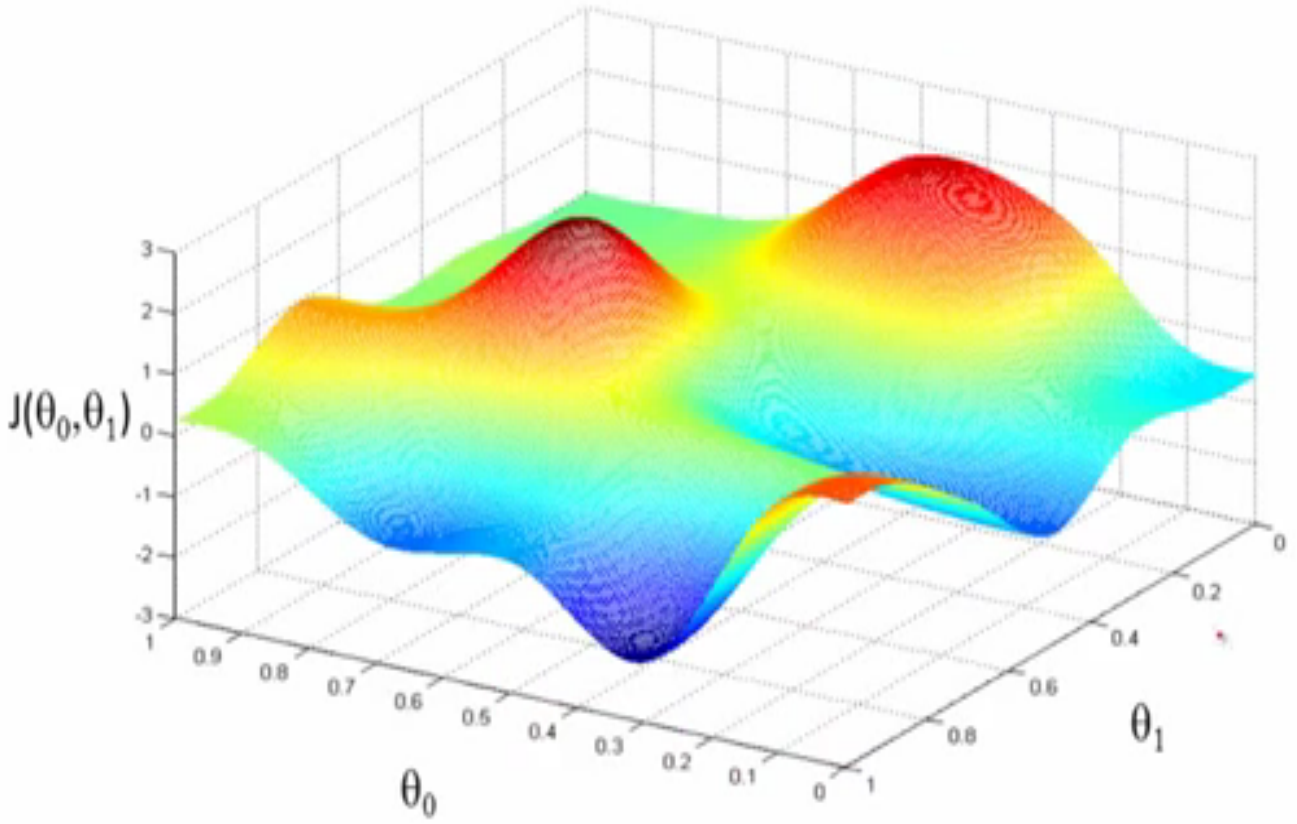
  until we hopefully end up at a minimum

Gradient Descent Intuition 1
Convex Cost Function



One of the many nice properties of convexity is that any local minimum is also a global minimum

Gradient Decent Intuition 2



Unfortunately, any interesting cost function is non-convex

Solving the Optimization Problem
Gradient Descent for Linear Regression

Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(for $j = 1$ and $j = 0$)

}

Linear Regression Model

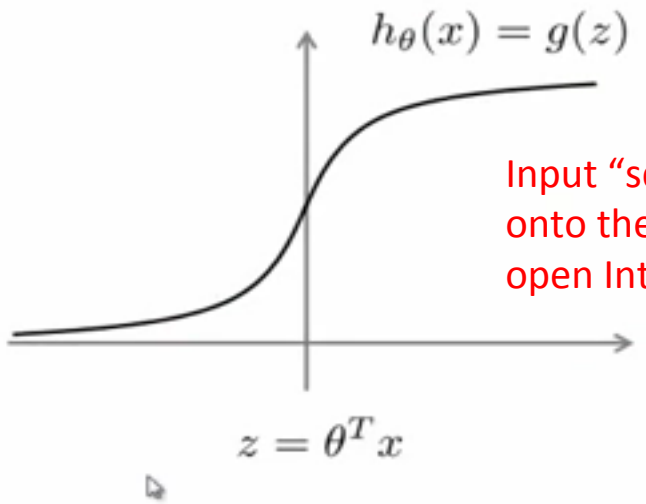
$$h_\theta(x) = \theta_0 + \theta_1 x$$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

Logistic Regression
Classification

- Recall that linear regression was about learning to predicting y_i given x_i
- Logistic regression is about modeling the conditional probability distribution $P(y|x;\theta)$
  - For $K \leq 2$ (number of classes), this is sufficient
    - For $K > 2$ need vectorized version
  - As such we need a hypothesis that gives us a real number in [0,1]
- Called "logistic regression" since $h_\theta(x) = g(\theta^T x)$ where g is the sigmoid/logistic function

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$



Input "squashed" onto the Real open Interval [0,1]

Binary vs. Multi-Class Classification

Binary classification

$y = 0$ or 1

1 output unit

Multi-class classification (K classes)

$y \in \mathbb{R}^K$ E.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

K output units

Logistic Regression Cost Function

Cost function

Logistic regression:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

Gradient Descent for Logistic Regression

Gradient Descent

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

Want $\min_\theta J(\theta)$:

Repeat $\{$

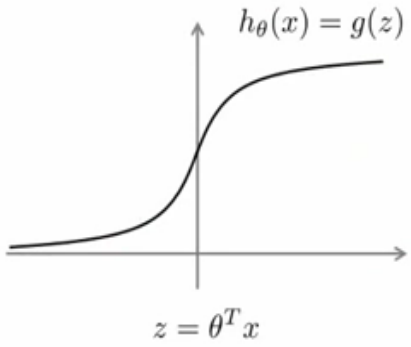
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

α is the *learning rate*

(simultaneously update all θ_j)

$\}$

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$



Derivative of the sigmoid function (quotient rule)

$$\begin{aligned}
\frac{d}{dx}\sigma(x) &= \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right) \\
&= \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \frac{(1+e^{-x})-1}{(1+e^{-x})^2} \\
&= \frac{1+e^{-x}}{(1+e^{-x})^2} - \left(\frac{1}{1+e^{-x}}\right)^2 \\
&= \sigma(x) - \sigma(x)^2 \\
\sigma' &= \sigma(1-\sigma)
\end{aligned}$$

Artificial Neural Networks

- A Bit of History
- Biological Inspiration
- Artificial Neurons (AN)
- Artificial Neural Networks (ANN)
- Computational Power of Single AN
- Computational Power of an ANN
- Training an ANN -- Learning

Brief History of Neural Networks

- **1943:** McCulloch & Pitts show that neurons can be combined to construct a Turing machine (using ANDs, ORs, & NOTs)
- **1958:** Rosenblatt shows that perceptrons will converge if what they are trying to learn can be represented
- **1969:** Minsky & Papert showed the limitations of perceptrons, killing research for a decade
- **1985:** The backpropagation algorithm revitalizes the field
  - Geoff Hinton et al

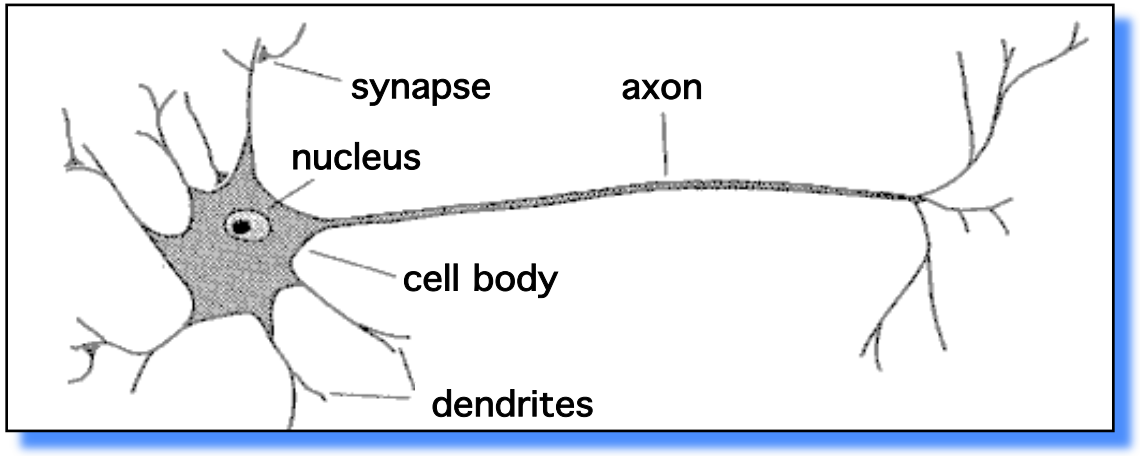
Biological Inspiration: Brains

- 200 billion neurons, 32 trillion synapses
- Element size: 10^{-6} m
- Energy use: 25W
- Processing speed: 100 Hz
- Parallel, Distributed
- Fault Tolerant
- Learns: Yes

- ~128 billion bytes RAM but trillions of bytes on disk
- Element size: 10^{-9} m
- Energy watt: 30-90W (CPU)
- Processing speed: 10^{9} Hz
- Serial, Centralized
- Generally not Fault Tolerant
- Learns: Some

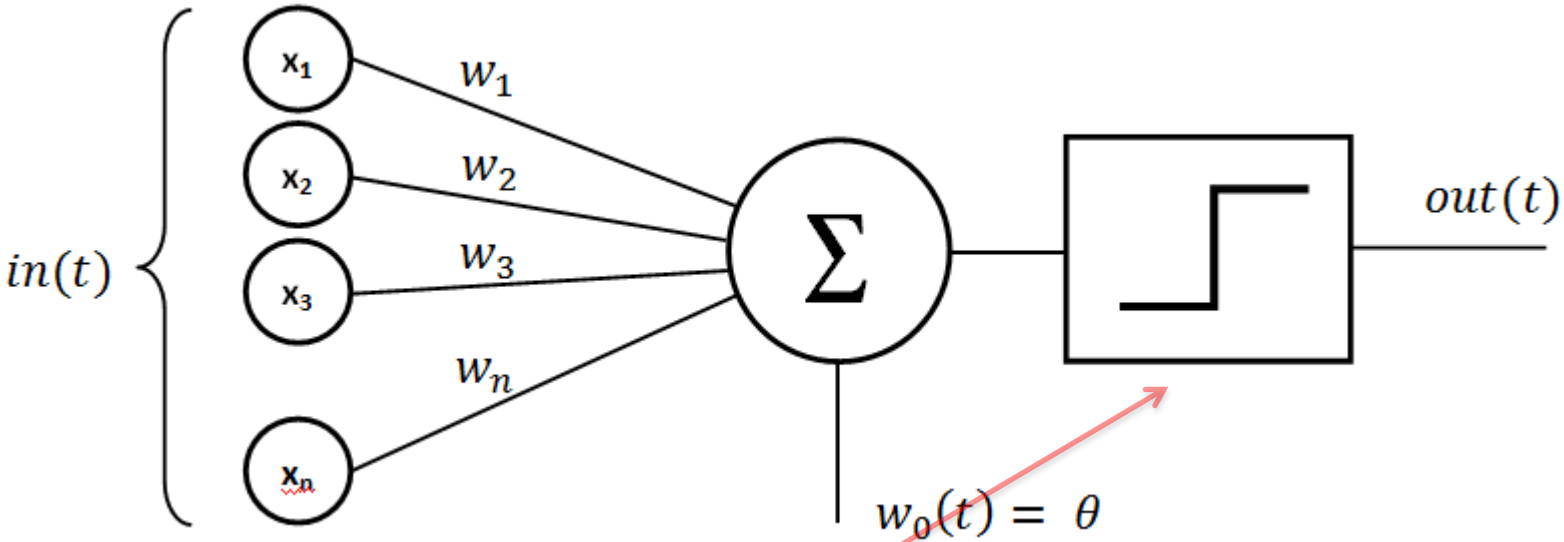
We will revisit the architecture of the brain when we talk about deep learning

Biological Inspiration: Neurons



- A neuron has
  - A branching input (dendrites)
  - A branching output (the axon)
- Information moves from the dendrites to the axon via the cell body
- Axon connects to dendrites via synapses
  - Synapses vary in strength
  - Synapses may be excitatory or inhibitory

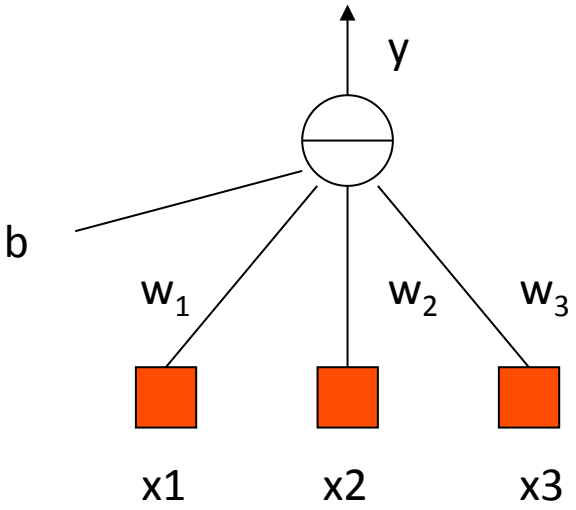
Basic Perceptron
(Rosenblatt, 1950s and early 60s)



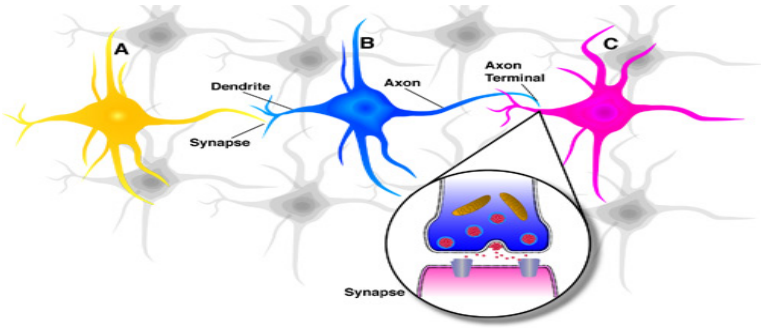
$$O = \begin{cases} 1 : \left(\sum_i w_i x_i\right) + b > 0 \\ 0 : otherwise \end{cases}$$

What is an Artificial Neuron?

- An Artificial Neuron (AN) is a non-linear parameterized function with restricted output range



$$y = f\left(b + \sum_{i=1}^{n-1} w_i x_i\right)$$



Neural Networks

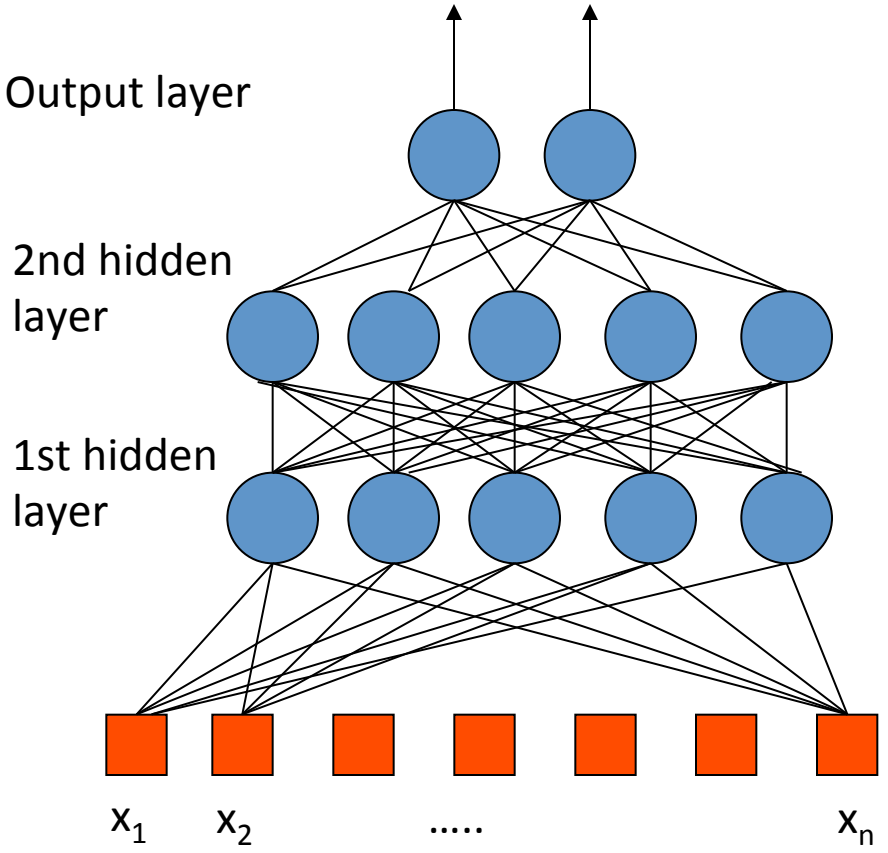
- An ANN is mathematical model designed to solve engineering problems
  - Group of highly connected neurons to realize compositions of non-linear functions
- Tasks
  - Classification
  - Discrimination
  - Estimation
- 2 types of networks
  - Feed forward Neural Networks
  - Recurrent Neural Networks
    - Can be generative

Discriminative vs. Generative Models

- Algorithms that try to learn p(y|x) directly are called *discriminative*
  - Here we try to fit a straight line (the decision boundary) to our data, say via logistic regression, then to classify a new x we look to see what side of the line the new x falls on. Perhaps the line separates elephants from dogs.
  - We will talk mostly about discriminative algorithms in this deck

- Algorithms that try to model p(y|x) are called *generative*
  - Learn p(x) rather than p(y|x)
  - Generative models can generate potential data points (the x's)
  - Here we would build a model of an elephant and a model of a dog
  - Then to classify a new animal, we match the new animal against the elephant model and match it against the dog model to see whether the new animal looks more like the elephants or more like the dogs we had seen in the training set

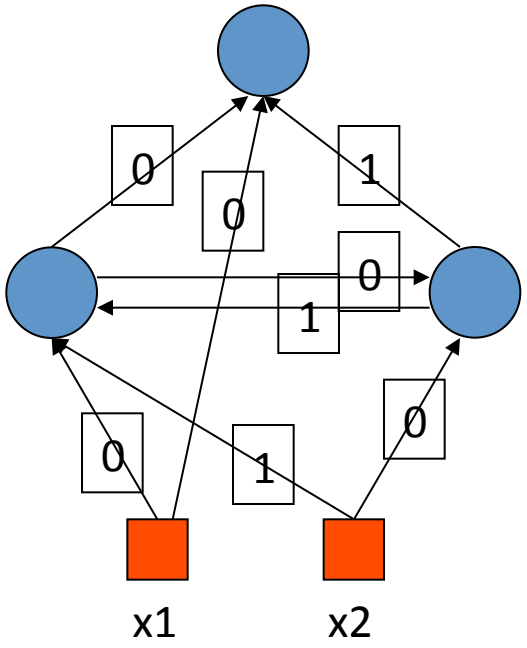
Feed Forward Neural Networks



- The information is propagated from the inputs to the outputs
  - Directed graph
- Computes one or more non-linear functions
  - Computation is carried out by composition of some number of algebraic functions implemented by the connections, weights and biases of the hidden and output layers
- Hidden layers compute intermediate representations
  - Dimension reduction
- Time has no role -- no cycles between outputs and inputs

We say that the input data, or features, are *n* dimensional

Recurrent Neural Networks



- Can have arbitrary topologies
  - Restricted Boltzmann Machines
  - Hopfield Network
  - …
- Can be generative
- Can model systems with internal states
- Delays are associated to a specific weight
- Training is more difficult
- Performance may be problematic
  - Stable outputs may be more difficult to evaluate
  - Unexpected behavior
    - oscillation, chaos, …

Learning

- The procedure that consists in estimating the parameters of neurons so that the whole network can perform a specific task
- 2 types of learning considered here
  - Supervised
  - Unsupervised
  - Semi-supervised learning
  - Reinforcement learning
- Supervised learning
  - Present the network a number of inputs and their corresponding outputs
  - See how closely the actual outputs match the desired ones
  - Modify the parameters to better approximate the desired outputs
- Unsupervised
  - Network learns internal representations and important features

Supervised learning

- The desired response of the neural network in function of particular inputs is well known
  - Training set mapping inputs to outputs
- Training set provides examples and teach the neural network how to fulfill a certain task
- Google example later in this deck uses supervised learning

Unsupervised learning

- Basic idea: Discover unknown structure in input data
- Data clustering and dimension reduction
  - More generally: find the relationships/structure in the data set
- No need for labeled data
  - The network itself finds the correlations in the data
- Learning algorithms include (there are many)
  - K-Means Clustering
  - Auto-encoders
  - Restricted Boltzmann Machines
    - Hopfield Networks
  - Sparse Encoders
  - ...

Well, How About Brains?

- Brains learn
  - Altering strength between neurons
  - Creating/deleting connections
  - Have a *deep* architecture
  - Use both supervised and unsupervised learning
- Hebb's Postulate (*Hebbian Learning*)
  - When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased
  - That is, *learning is about adjusting weights and biases*
- Long Term Potentiation (LTP)
  - Cellular basis for learning and memory
  - LTP is the long-lasting strengthening of the connection between two nerve cells in response to stimulation
  - Discovered in many regions of the cortex
- "One Learning Algorithm" Hypothesis

The "One Learning Algorithm" hypothesis

- There is some evidence that the human brain uses essentially *the same algorithm* to understand many different input modalities.
  - Example: Ferret experiments, in which the "input" for vision was plugged into auditory part of brain, and the auditory cortex learns to "see." [Roe et al., 1992]

(Roe et al., 1992. Hawkins & Blakeslee, 2004)

Artificial Neuron – Deeper Dive

- Neuron pre-activation (or input activation):

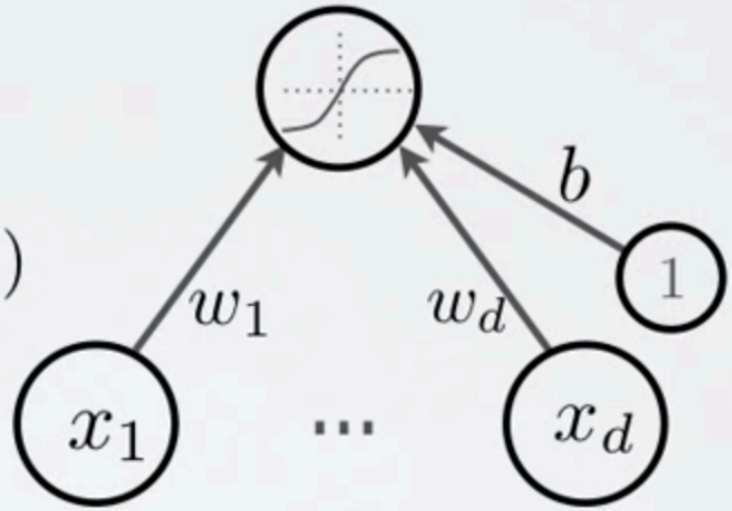
$$a(\mathbf{x}) = b + \sum_i w_i x_i = b + \mathbf{w}^\top \mathbf{x}$$

- Neuron (output) activation

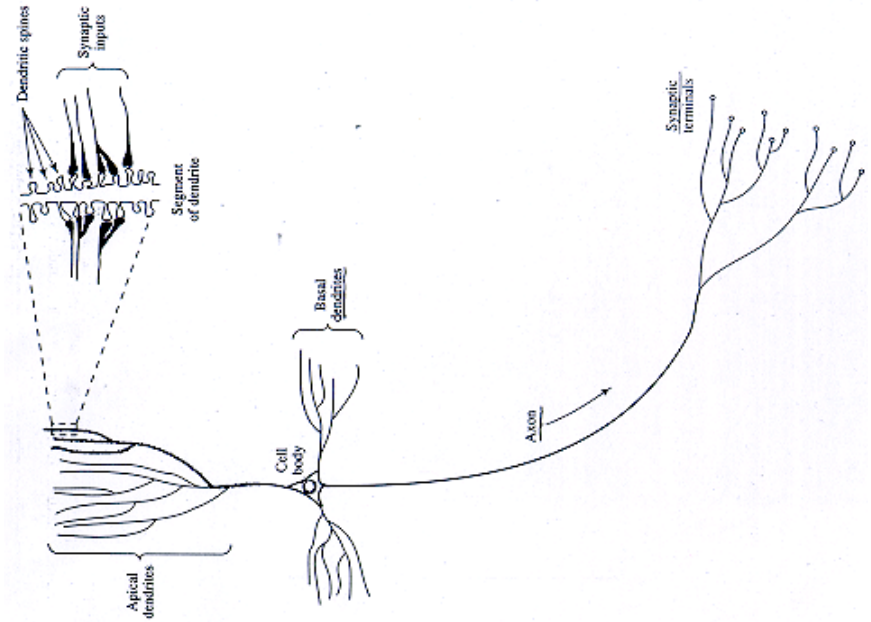
$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

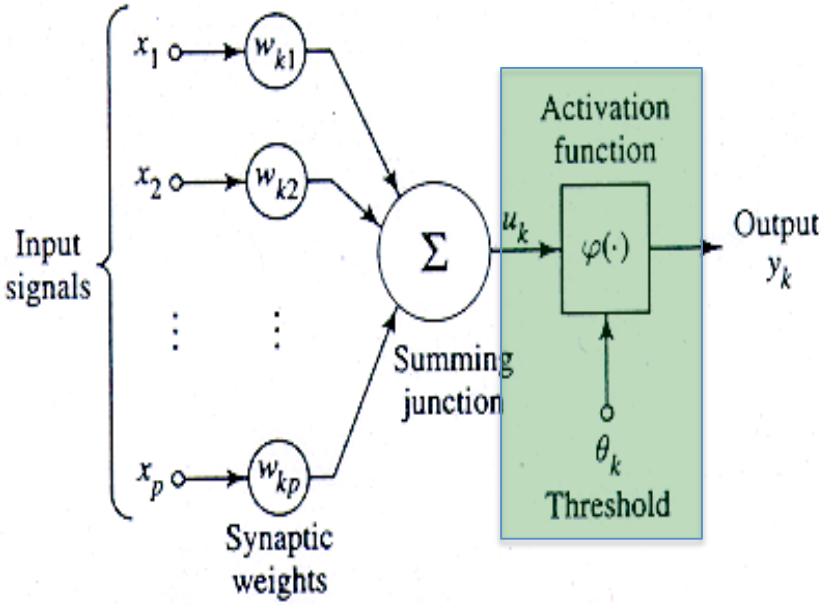
$h(x) \sim h_\theta(x)$

- $\mathbf{w}$ are the connection weights
- b is the neuron bias
- $g(\cdot)$ is called the activation function



Review: Mapping to Biological Neuron

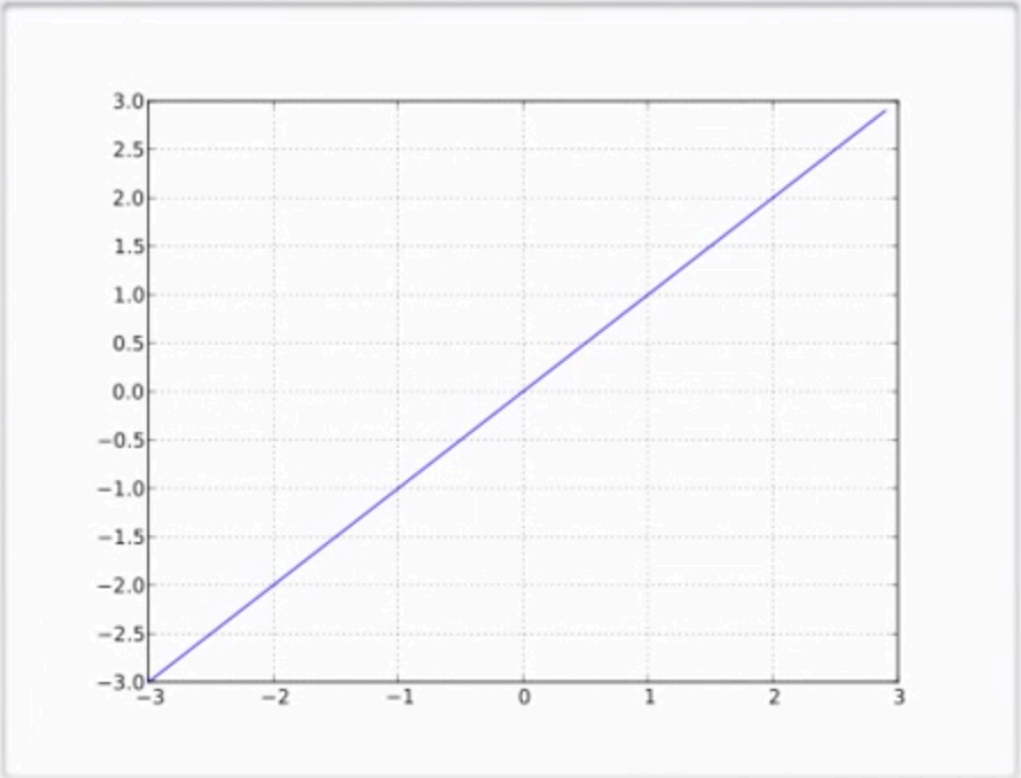




Dendrite Cell Body Axon

Activation Functions – Linear Function

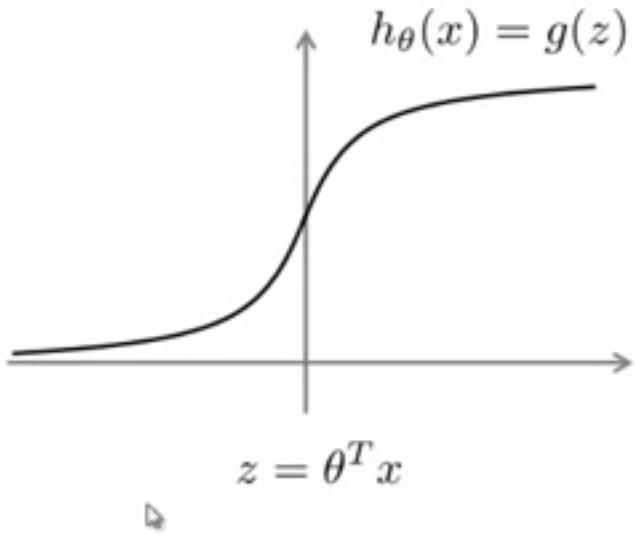
- Performs no input squashing
- Not very interesting...



$$g(a) = a$$

Activation Functions – Sigmoid Function

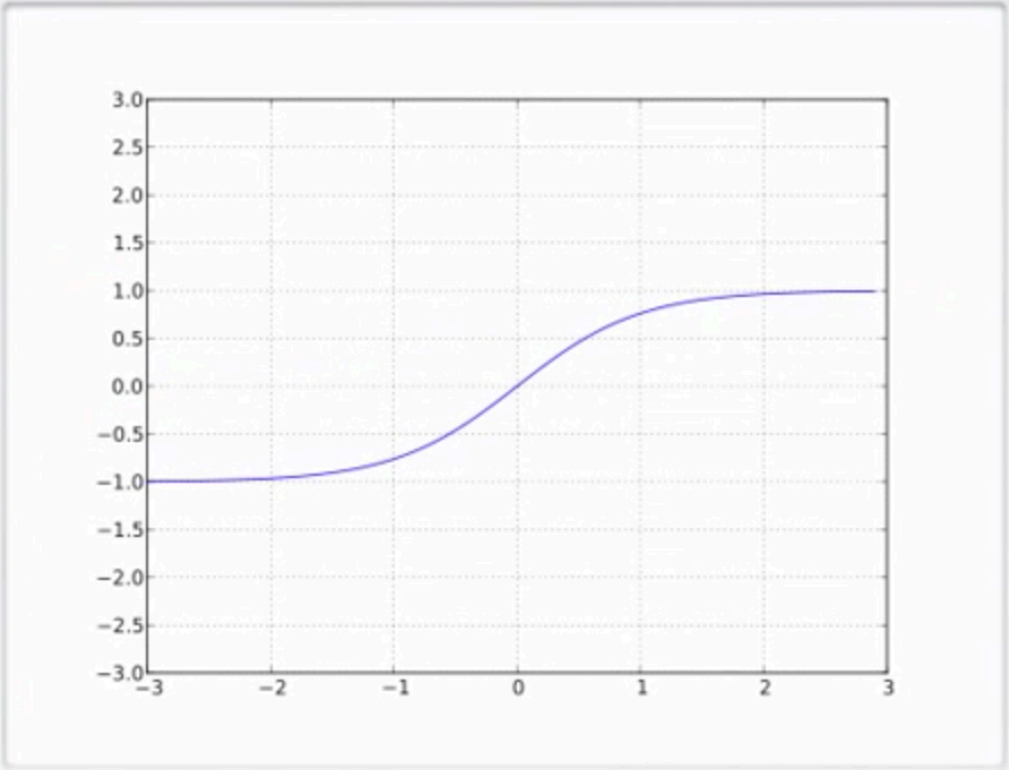
$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$



Squashes the input (x) onto the open interval [0,1]

Activation Functions – Hyperbolic Tangent

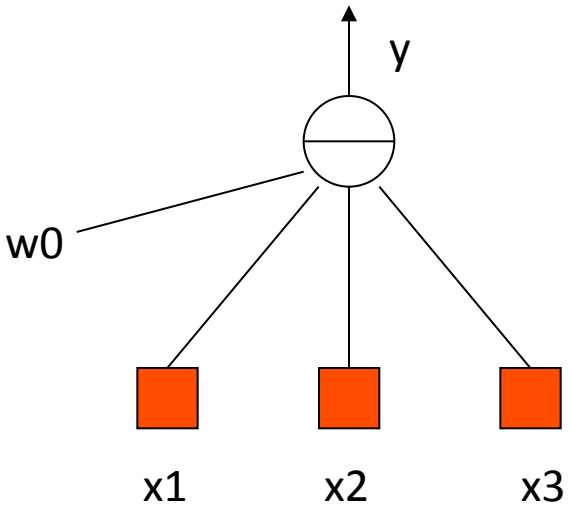
- Squashes the neuron's input between -1 and 1
- Can be positive or negative
- Bounded
- Strictly increasing



$$g(a) = \tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} = \frac{\exp(2a) - 1}{\exp(2a) + 1}$$

Summary: Artificial neurons

- An *Artificial Neuron* is a non-linear parameterized function with restricted output range



$$y = f\left(w_0 + \sum_{i=1}^{n-1} w_i x_i \right)$$

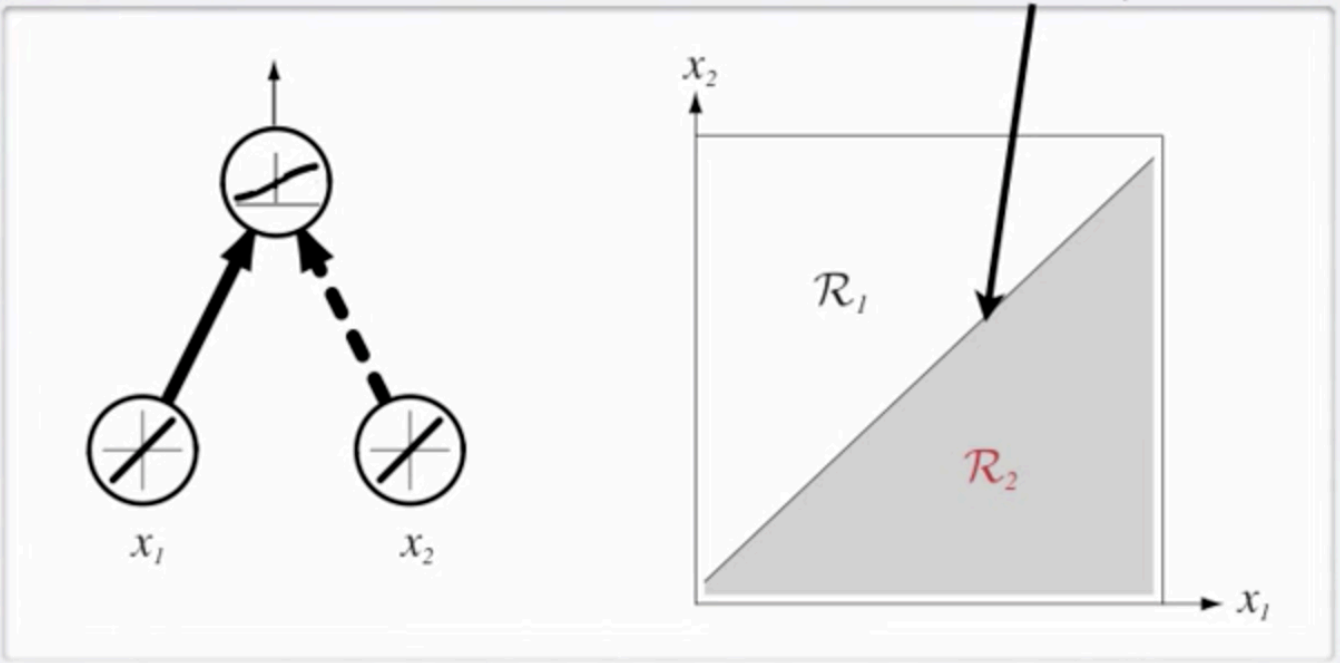
w_0 also called *a bias term (b_i)*

Computational Complexity
Single Artificial Neuron

- Could do binary classification:
  - with sigmoid, can interpret neuron as estimating $p(y = 1|\mathbf{x})$
  - also known as logistic regression classifier
  - if greater than 0.5, predict class 1
  - otherwise, predict class 0

(similar idea can apply with tanh)

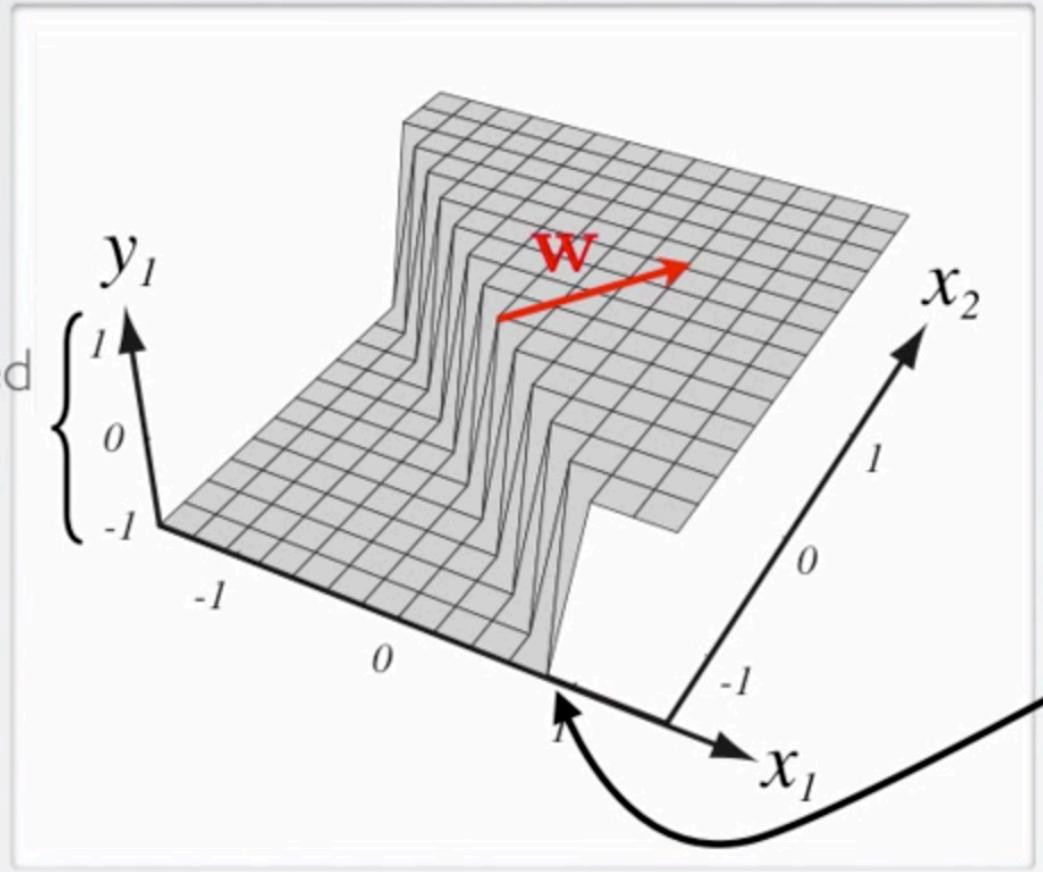
decision boundary is linear



(from Pascal Vincent's slides)

Linear Separability

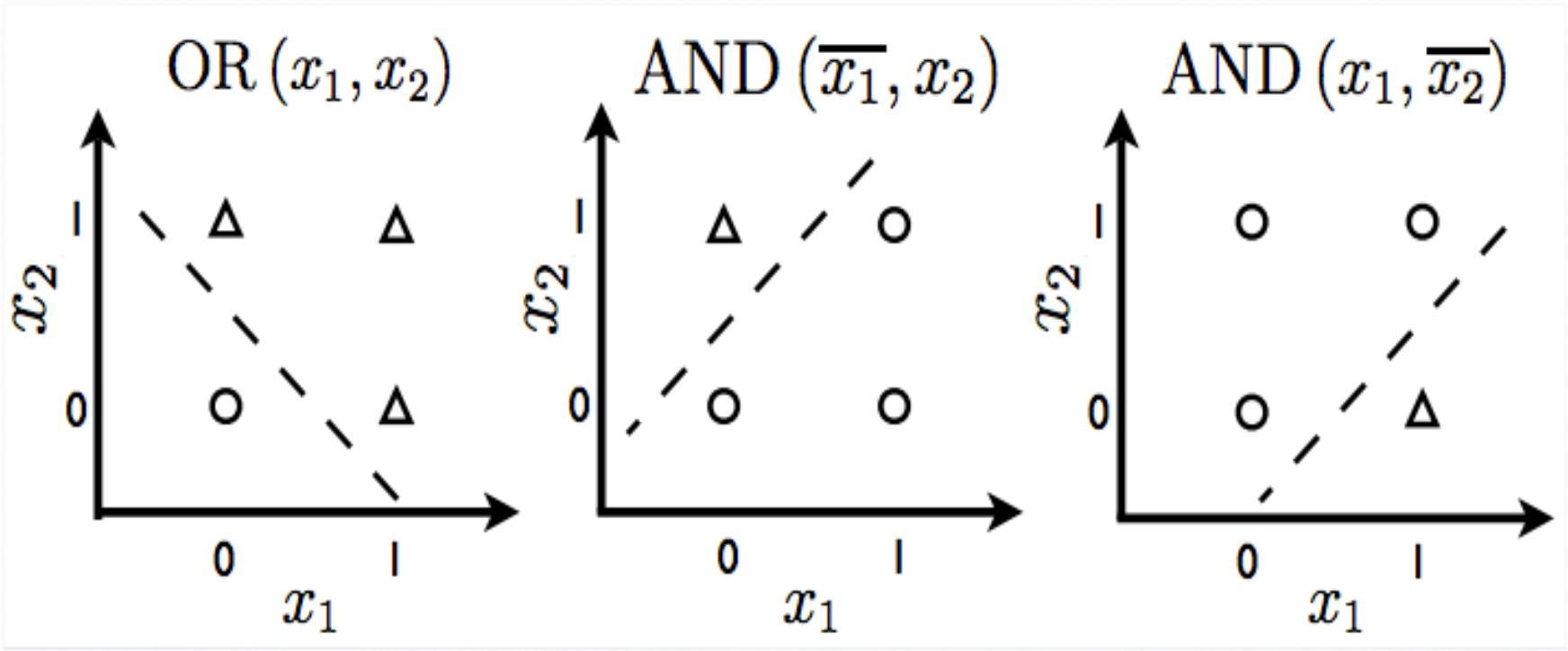
range determined by $g(\cdot)$



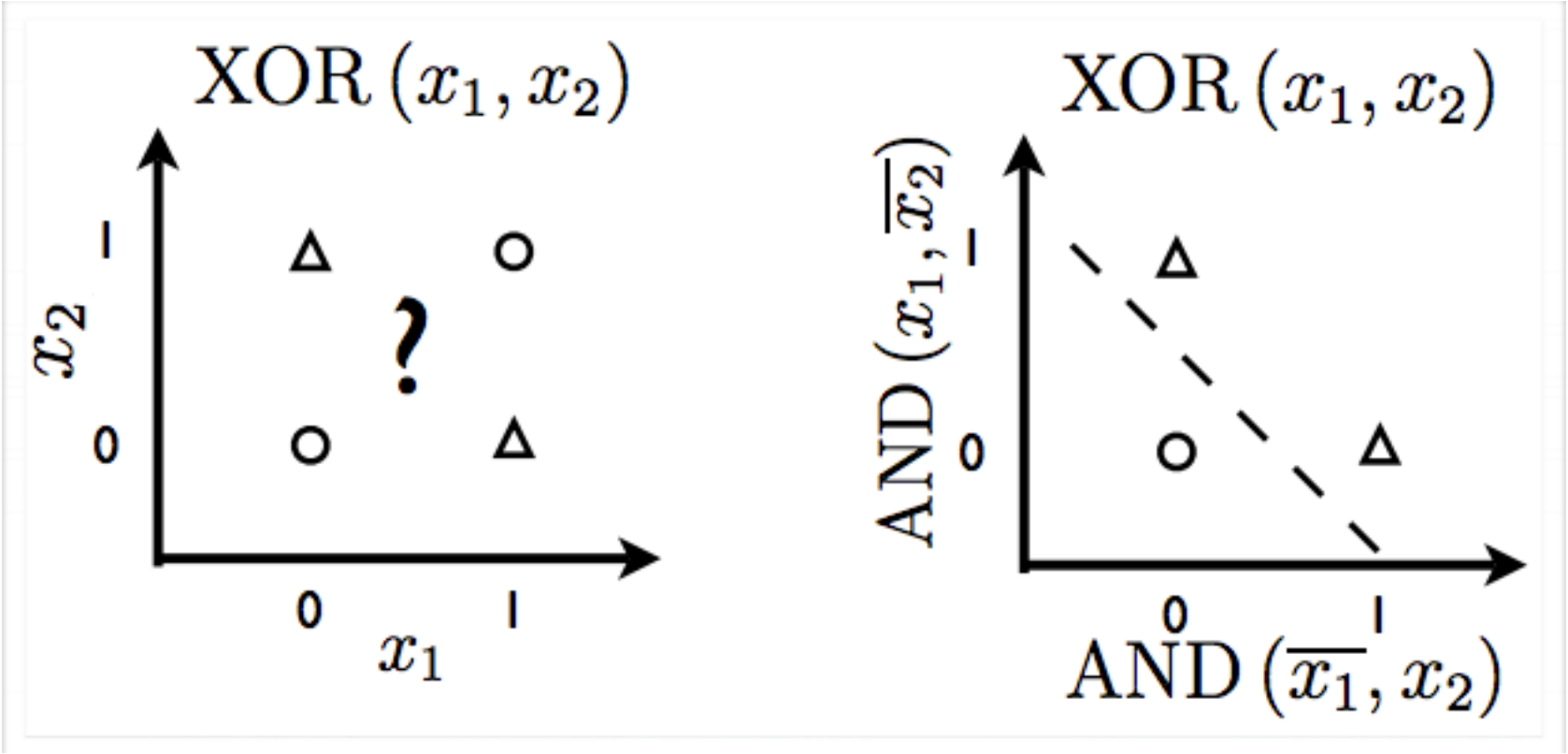
bias b only changes the position of the riff

(from Pascal Vincent's slides)

A Single Artificial Neuron Can Solve Linearly Separable Problems



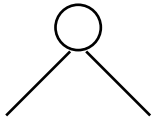
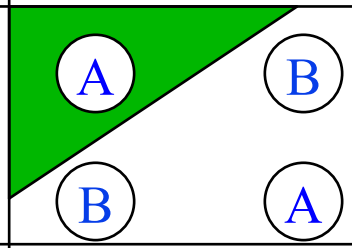
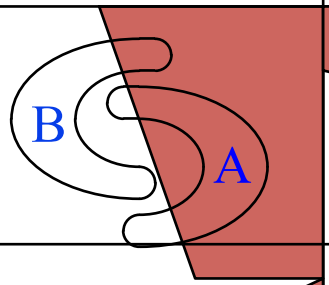
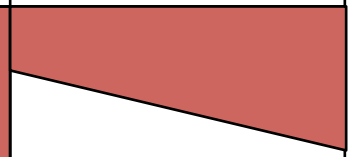
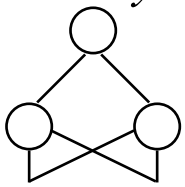
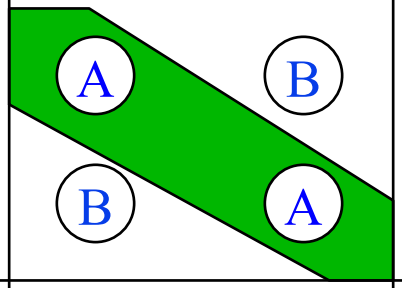
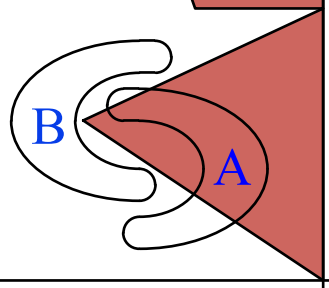
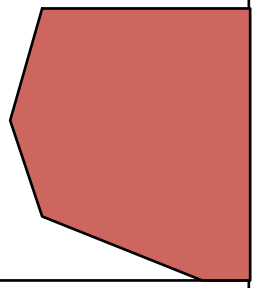
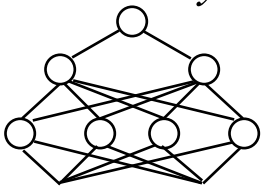
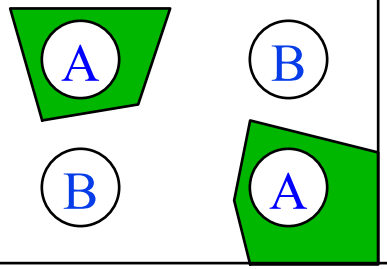
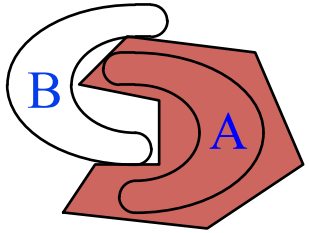
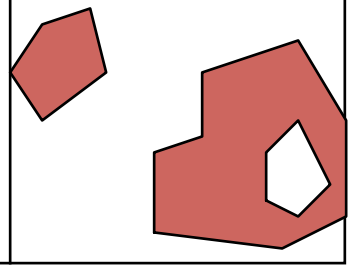
A Single Artificial Neuron Can't Solve Non-Linearly Separable Problems



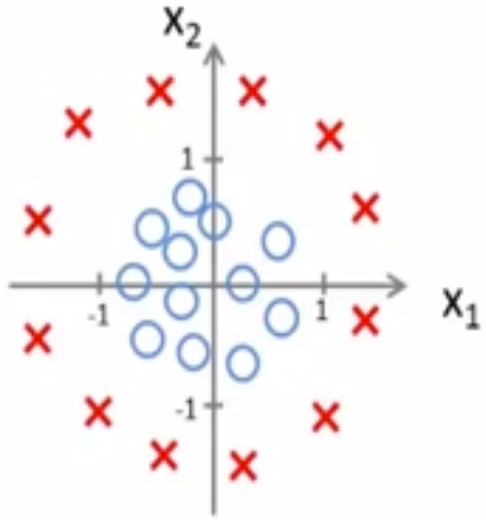
Unless the input is somehow transformed to into a linearly separable form…

- SVM + kernel trick
- Multi-layer Neural Networks
- RBMs
- …

Non-linearly separable problems

Structure	*Types of Decision Regions*	*Exclusive-OR Problem*	*Classes with Meshed regions*	*Most General Region Shapes*
Single-Layer 	*Half Plane Bounded By Hyperplane*			
Two-Layer 	*Convex Open Or Closed Regions*			
Three-Layer 	Arbitrary (Complexity Limited by No. of Nodes)			

Another Way to View Non-Linear Features



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

Note Potential for *Overfitting* → Model winds up emphasizing noise

Aside: Many Classical ANN Architectures

- Perceptron
- Multi-Layer Feed Forward Neural Network
- Support Vector Machines (SVMs)
- Conditional Random Fields (CRFs)
- Kohonen Features maps
- …

Single Hidden Layer Neural Network

Vectorized form

- Hidden layer pre-activation:

$$\mathbf{a}(\mathbf{x}) = \mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}$$

$$\left(a(\mathbf{x})_i = b_i^{(1)} + \sum_j W_{i,j}^{(1)} x_j\right)$$

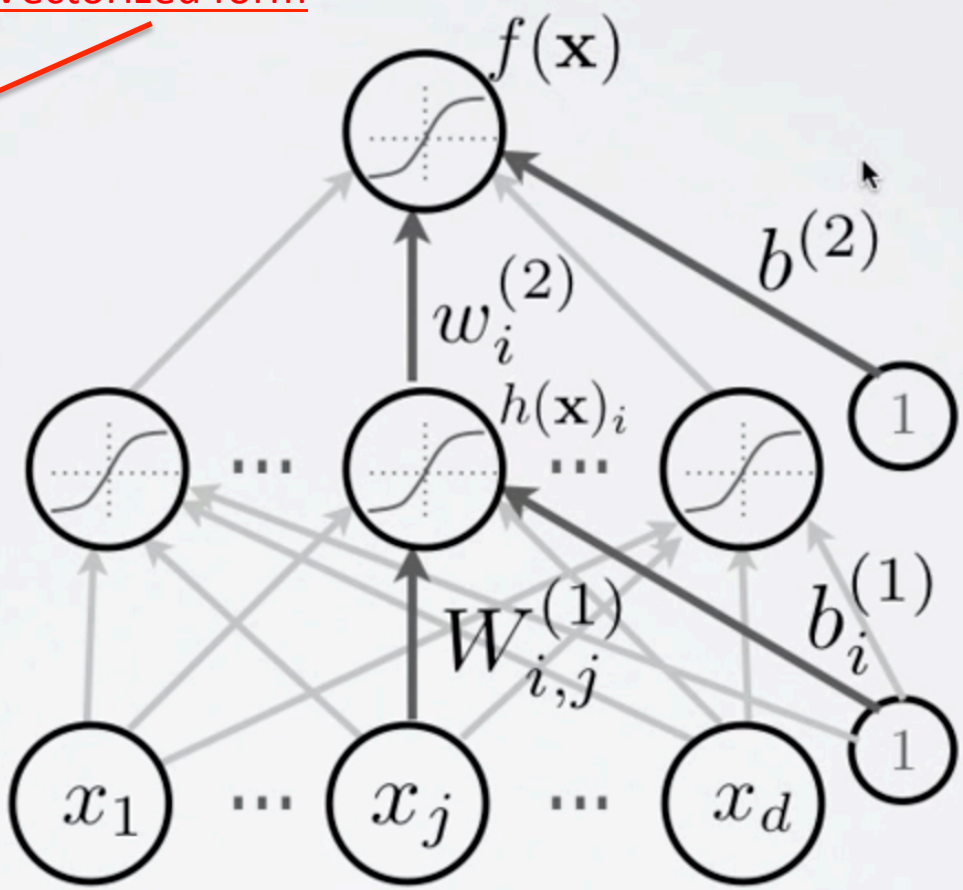
- Hidden layer activation:

$$\mathbf{h}(\mathbf{x}) = \mathbf{g}(\mathbf{a}(\mathbf{x}))$$

- Output layer activation:

$$f(\mathbf{x}) = o\left(b^{(2)} + \mathbf{w}^{(2)\top}\mathbf{h}^{(1)}\mathbf{x}\right)$$

output activation function



Note notation change: $W^{(l)}_{i,j}$ is the weight on the link beween the i^{th} and j^{th} unit in layer l

So what can a single hidden layer neural network compute?

Universal Approximation Theorem

- Universal approximation theorem (Hornik, 1991):
  - "a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units"
- The result applies for sigmoid, tanh and many other hidden layer activation functions
- This is a good result, but it doesn't mean there is a learning algorithm that can find the necessary parameter values!

Bad news: The single hidden layer neural network can be exponentially large

General Case
The Feed Forward Neural Network

Multi-class classification
(may use softmax activation function)

- Could have L hidden layers:
  - layer pre-activation for $k>0$ $(\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x})$

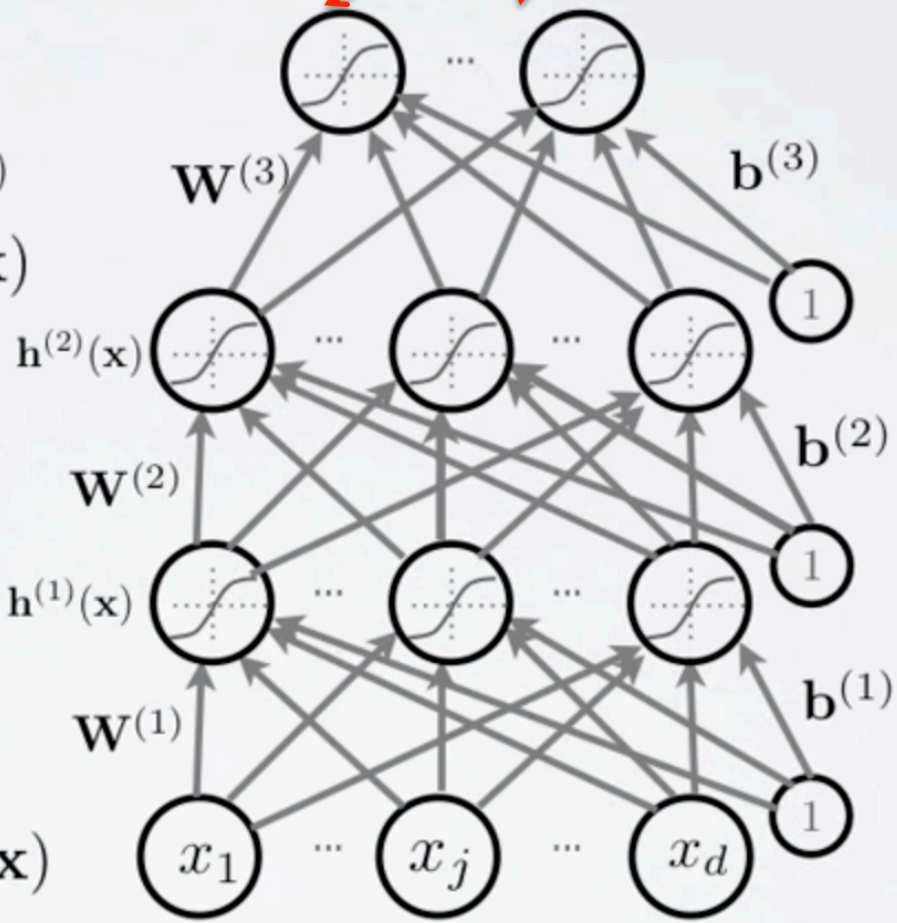
$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

  - hidden layer activation (k from 1 to L):

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$$

  - output layer activation ($k=L+1$):

$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$



Multi-class Classification
Softmax Activation Function

- For multi-class classification:
  - we need multiple outputs (1 output per class)
  - we would like to estimate the conditional probability $p(y = c|\mathbf{x})$

- We use the softmax activation function at the output:

$$\mathbf{o}(\mathbf{a}) = \text{softmax}(\mathbf{a}) = \left[\frac{\exp(a_1)}{\sum_c \exp(a_c)} \cdots \frac{\exp(a_C)}{\sum_c \exp(a_c)} \right]^\top$$

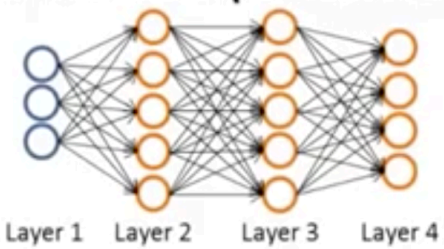
  - strictly positive
  - sums to one

- Predicted class is the one with highest estimated probability

Machine Learning with a ANN
Cost Function

- Recall that for classification we had

Neural Network (Classification)



$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$

$L =$ total no. of layers in network

$s_l =$ no. of units (not counting bias unit) in layer l

- Neural Network Cost Function

$$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$$

K classes

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

Review: Gradient Descent for Logistic Regression

Gradient Descent

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

Want $\min_\theta J(\theta)$:

Repeat $\{$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

α is the *learning rate*

(simultaneously update all θ_j)

$\}$

"log likelyhood"

$$h_\theta(x) = \frac{1}{1 - e^z}, where\ z = -\theta^T x$$

or

$$h_\theta(x) = \frac{1}{1 - e^{-\theta^T x}}$$

So to minimize the ANN Cost Function

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log h_\theta(x^{(i)})_k + (1 - y_k^{(i)}) \log(1 - h_\theta(x^{(i)})_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_j^{(l)})^2$$

$$\min_{\Theta} J(\Theta)$$

Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta_0, \theta_1)$$

(for $j = 1$ and $j = 0$)

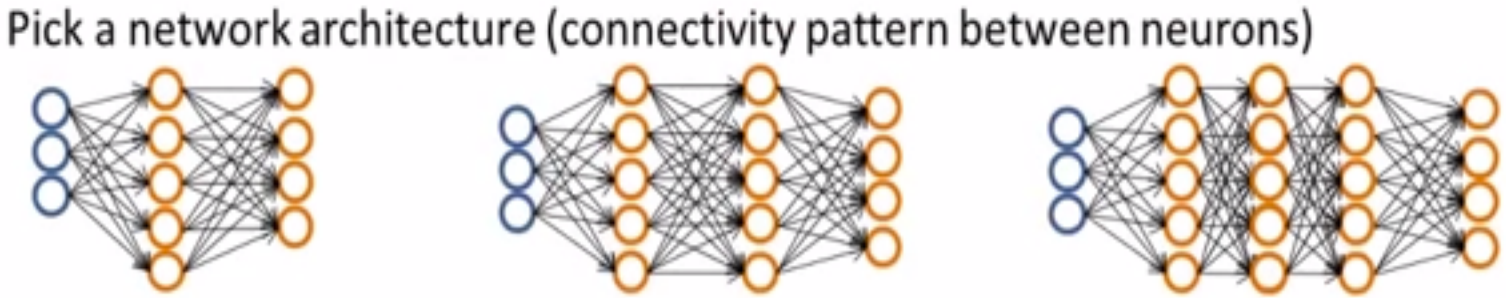
}

Need code to compute:

- $J(\Theta)$
- $\frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta)$

How to calculate these?
(we'll see this in a few slides)

Training an ANN(1)
First, Pick a Network Architecture



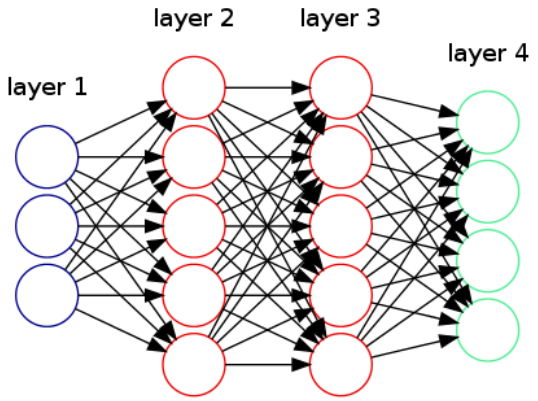
- Number of input units: Dimension of features $x^{(i)}$
- Number of output units: Number of classes (K)
- Number of hidden units: Reasonable default ~ 1 hidden layer
  - if > 1 same number of units in each layer (usually more is better)

Training an ANN (2)

1. Randomly initialize W (weights)
2. Until Converged (or fixed number of iterations)
   1. Implement forward propagation to get $h_\theta(x^{(i)})$ for all $x^{(i)}$
   2. Compute cost function $J(\theta)$
   3. Implement backpropagation to compute partial derivatives $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$
3. Use gradient checking to compare partial derivatives computed during backpropagation vs. numerical estimate of gradient of $J(\theta)$
4. Use gradient descent or other advanced optimization method with backprop to try to minimize $J(\theta)$ as a function of parameters θ

Forward Propagation: Technical Details

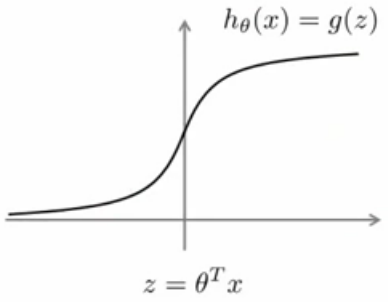
Example Network (L = 4, K=4)



Forward Propagation Algorithm (vectorized form)

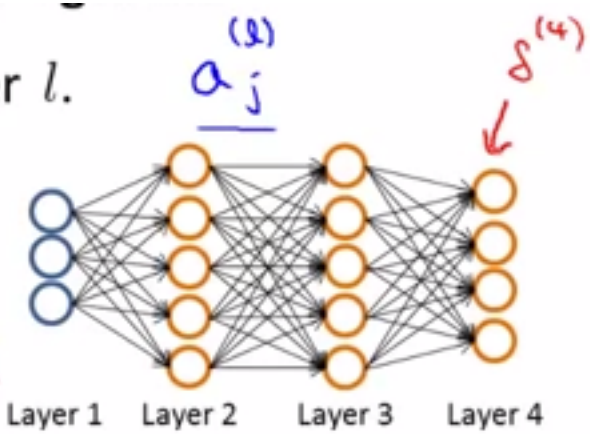
$$
\begin{aligned}
a^{(1)} &= x \\
z^{(2)} &= \Theta^{(1)} a^{(1)} \\
a^{(2)} &= g(z^{(2)}) \quad (\text{add } a_0^{(2)}) \\
z^{(3)} &= \Theta^{(2)} a^{(2)} \\
a^{(3)} &= g(z^{(3)}) \quad (\text{add } a_0^{(3)}) \\
z^{(4)} &= \Theta^{(3)} a^{(3)} \\
a^{(4)} &= h_\Theta(x) = g(z^{(4)})
\end{aligned}
$$

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$



Backward Propagation: Intuition
(computing the error $\delta_j^{(l)}$)

Intuition: $\delta_j^{(l)}$ = "error" of node j in layer l.



For each output unit (layer L = 4)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

Recall: $g'(z^{(i)}) = a^{(i)} .* (1 - a^{(i)})$

Note: no $\delta^{(1)}$ terms since there is no "error" in the input

Backward Propagation: Full Algorithm

Backpropagation algorithm

Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

Set $\Delta_{ij}^{(l)} = 0$ (for all l, i, j). (used to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$)

For $i = 1$ to m

Set $a^{(1)} = x^{(i)}$

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

$D^{(l)}_{i,j} := (1/m)\Delta^{(l)}_{i,j} + \lambda\Theta^{(l)}_{i,j}$ if $j \neq 0$

$D^{(l)}_{i,j} := (1/m)\Delta^{(l)}_{i,j}$ if $j = 0$ (bias terms)

Finally, what you can show is that $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$

Issues with Back Propagation

- It requires labeled training data
  - And almost all data is unlabeled
  - Supervised vs. unsupervised learning
  - Note auto-encoders

- Learning time does not scale well
  - It is very slow in networks with multiple hidden layers
    - Deep Learning
  - Gradient is progressively getting more dilute
    - below top few layers, correction signal is minima

- It can get stuck in poor local optima
  - True for any optimization approach that uses gradient descent on a non-convex objective function
  - Any non-trivial loss function will likely be non-convex
  - Local minima can often quite good, but for deep nets they are far from optimal

- Answers?
  - Deep Belief Nets stochastic binary units (Bernoulli variables)
  - Generative models
    - Can randomly generate observable data from probabilistic hidden parameters
  - Usually uses stochastic binary units (Bernoulli variables)

Agenda

- ~~Goals for this Talk~~
- ~~What is Machine Learning?~~
  - ~~Kinds of Machine Learning~~
- ~~Machine Learning Fundamentals~~
  - ~~Shallow dive~~
  - ~~Inductive Learning: Regression and Classification~~
  - ~~Focus on Artificial Neural Networks (ANNs)~~
- A Bit on Unsupervised Learning
- Deep Learning
- Google PUE Optimization Machine Learning Application
- Next Steps

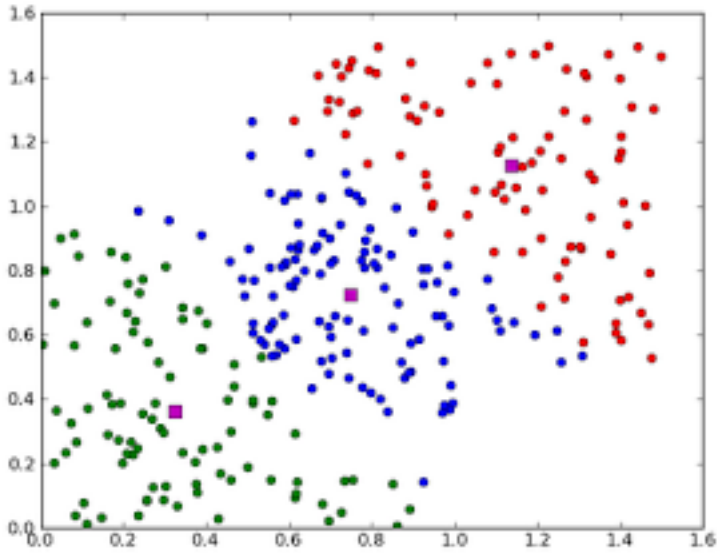
Unsupervised Learning

- Most data is unlabeled
  - Want to take advantage of this
  - Dimension reduction/structure discovery
  - Previously computationally impractical
    - 100x increase in computational power (GPUs, etc)
    - 300x increase in algorithmic efficiency

- Methods
  - K-Means (clustering)
  - {Stacked, Denoising} auto-encoders
  - Restricted Boltzmann Machines
    - Hopfield Networks
  - Sparse Encoding
  - PCA
  - …

Briefly: K-Means

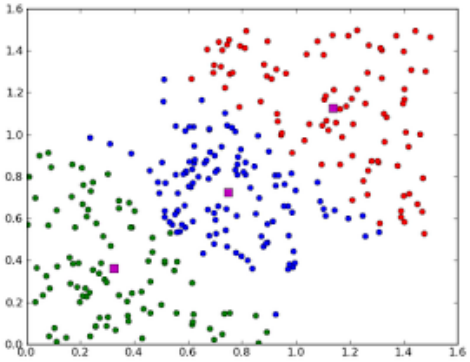
- Finds “clusters” in data set
- Consider the following data set



K-Means Algorithm

K-means algorithm

Randomly initialize K cluster centroids $\mu_1, \mu_2, \ldots, \mu_K \in \mathbb{R}^n$



Repeat {

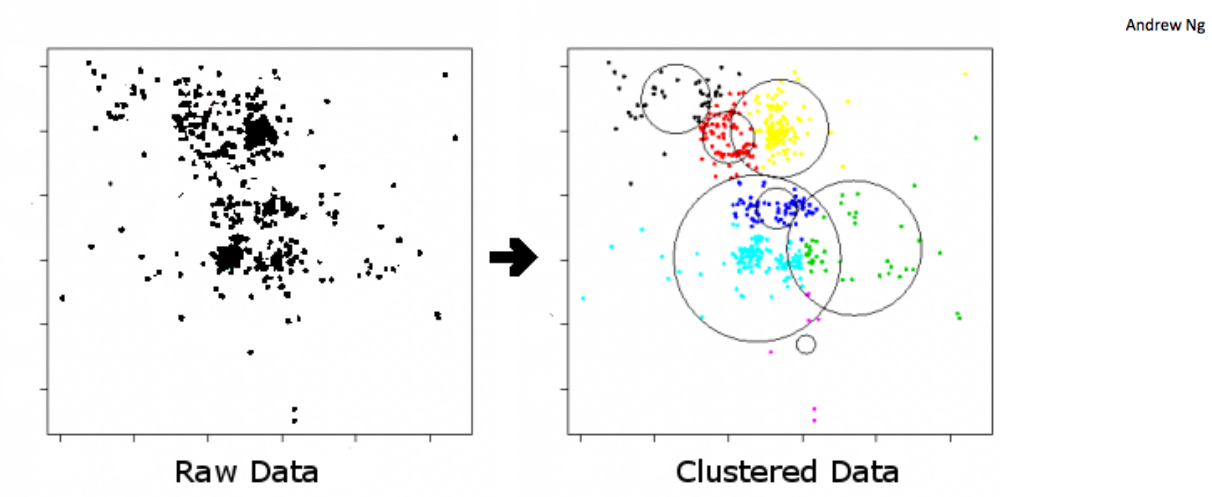
for i = 1 to m

$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$

for k = 1 to K

μ_k := average (mean) of points assigned to cluster k

}



Andrew Ng

Auto-encoders

- What would happen if you were to compute $h_\theta(\mathbf{x}) \approx \mathbf{x}$?
  - Rather than $h_\theta(\mathbf{x}) \approx \mathbf{y}$ (the supervised learning case)
  - Trying to learn the identity function and interesting lower dimensioned features
  - Note: *Curse of Dimensionality*

- How does this work? Suppose the inputs in **x** are pixel intensity and we have an image of size 10 x 10 (100 pixels). Suppose further that we have a network with one hidden layer and $s_2 = 50$ hidden units in L_2
  - Since there are 50 hidden units the network is forced to learn a *compressed* representation of **x**
  - But since $a^{(2)}$ is in R^{50} and y is in R^{100}, the network must also try to reconstruct the 100-pixel image
  - If the input were completely random, say x_i comes from and IID Gaussian, the compression task would be very hard. But if there is structure in the data, for example if there are correlations in the data, then this algorithm will discover those correlations
  - This argument relies on $|s_2|$ being small relative to the dimensionality of x and y
  - But the auto-encoder will still discover interesting structure even with a large number of hidden units if we impose a *sparsity* constraint on the hidden units
    - i.e., most neurons are inactive (0) most of the time

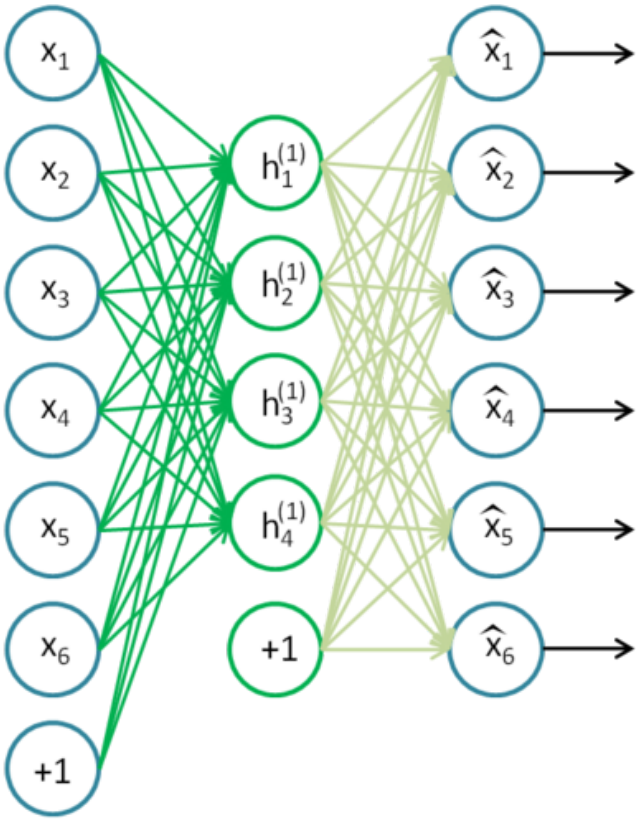
- Many kinds of auto-encoders
  - Stacked auto-encoder
  - Denoising auto-encoder
  - …

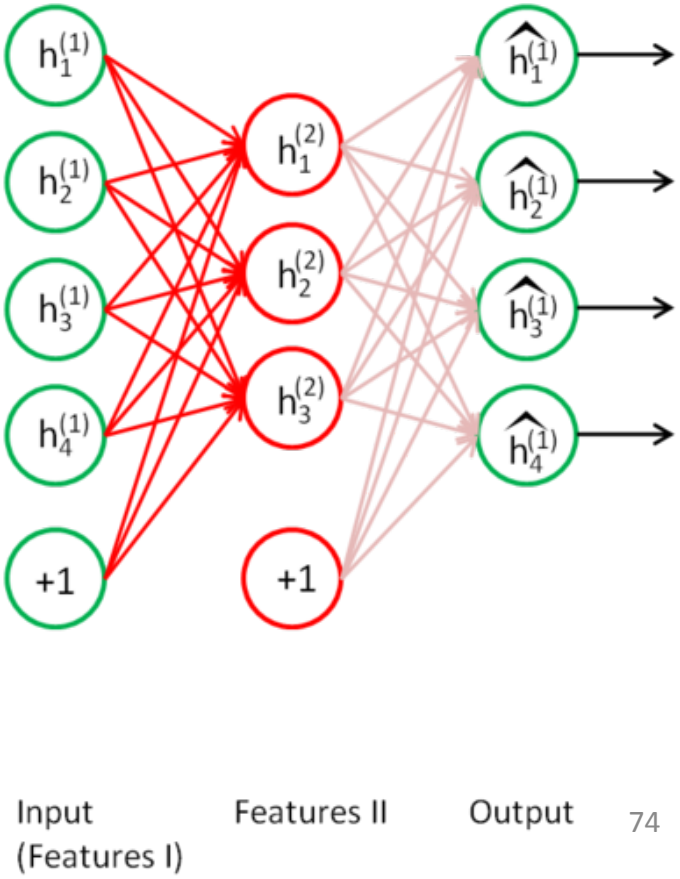
Stacked Auto-encoders

Bengio (2007)

Idea: Stack many (sparse) auto-encoders in succession

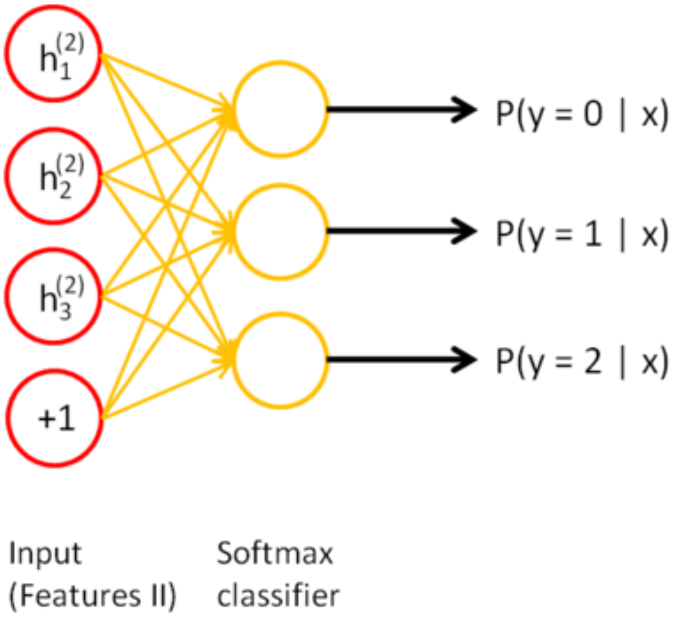
- Train them using greedy layer-wise training
- Drop the decode output layer each time

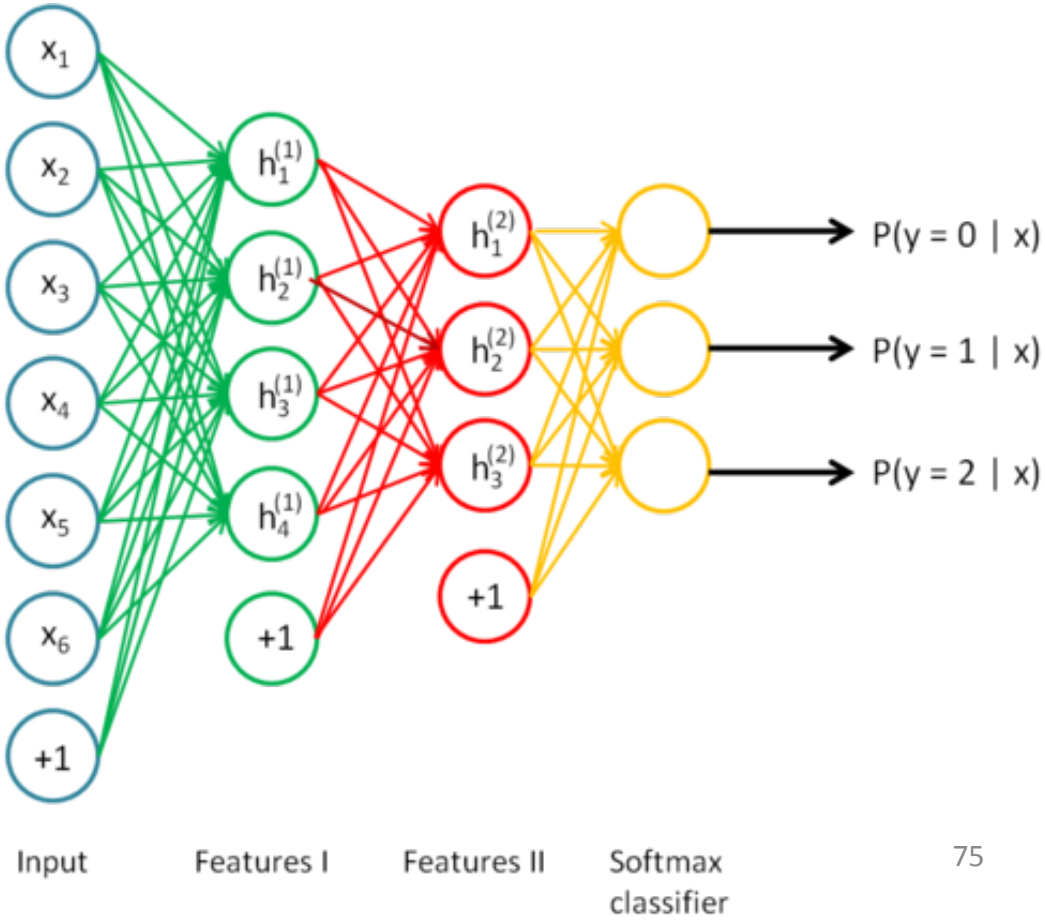




Stacked Auto-encoders

- Do supervised training on the last layer using final features
- Then do supervised training on the entire network to fine-tune **θ**





Agenda

- ~~Goals for this Talk~~
- ~~What is Machine Learning?~~
  - ~~Kinds of Machine Learning~~
- ~~Machine Learning Fundamentals~~
  - ~~Shallow dive~~
  - ~~Inductive Learning: Regression and Classification~~
  - ~~Focus on Artificial Neural Networks (ANNs)~~
- ~~A Bit on Unsupervised Learning~~
- Deep Learning
- Google PUE Optimization Machine Learning Application
- Next Steps

Deep Learning

- The brain has a deep architecture
  - Previously too hard to train deep architectures
  - Breakthrough: Hinton et. al circa 2006
    - "A Fast Learning Algorithm for Deep Belief Nets"
      - Hinton, et al., 2006
    - "Reducing the Dimensionality of Data with Neural Networks"
      - Hinton & Salakhutdinov, 2006

- All kinds of interesting work in this area
  - e.g., The Google Brain team created a deep neural network that learned to recognize higher-level concepts, such as cats, only from watching unlabeled images taken from YouTube videos

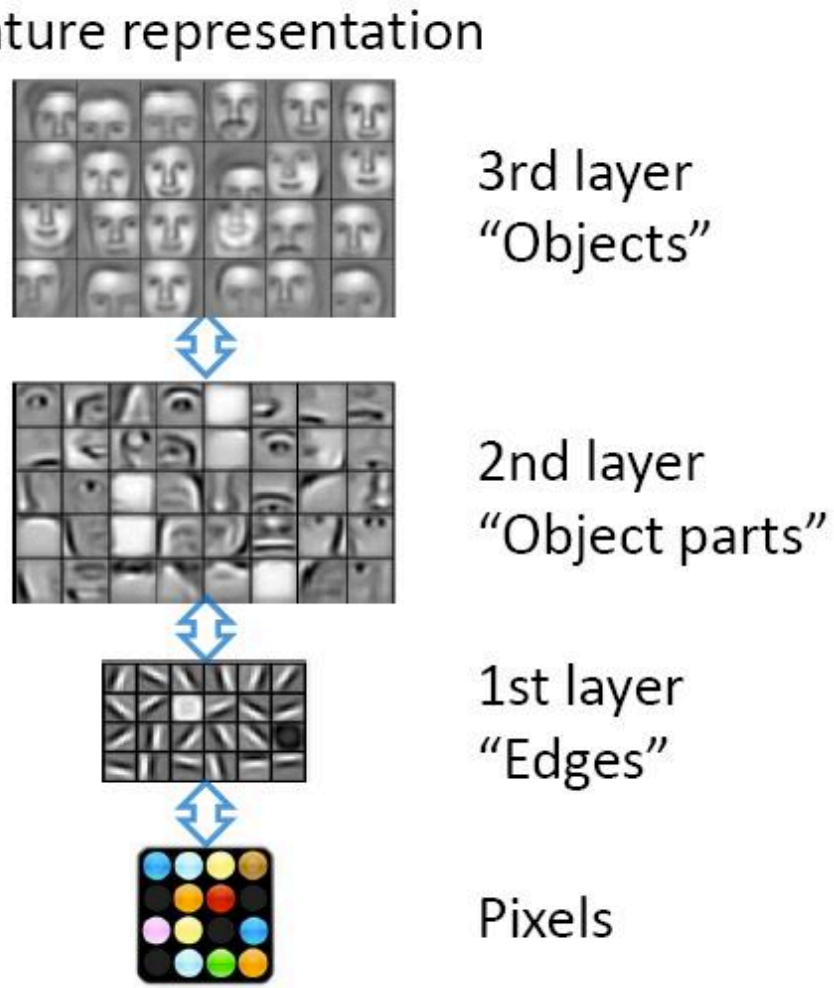
Why Deep Architecture?

- Can be representationally efficient
  - Fewer computational units for same function
- Might allow for representation hierarchy
  - Allows non-local generalization
  - Comprehensibility
- Multiple levels of latent variables allow combinatorial sharing of statistical strength
- Work very well
  - vision, audio, NLP, ….

Deep Architectures Facilitate Different Levels of Abstraction

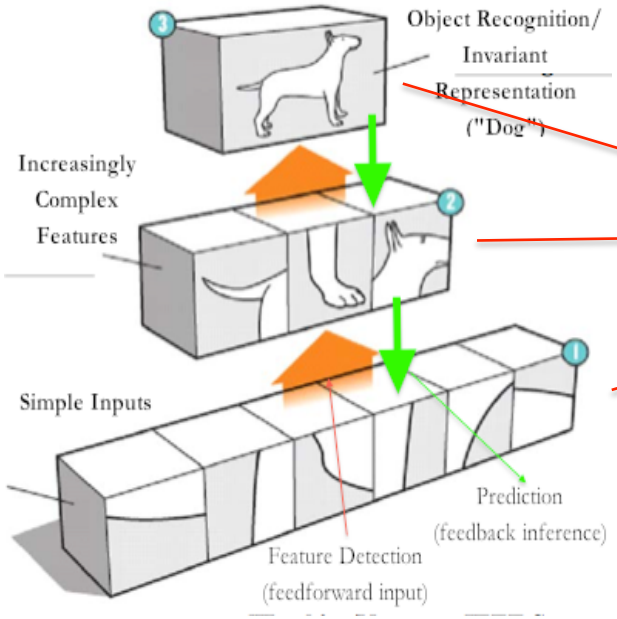
- **Hierarchical Learning**
  - Natural progression from low level to high level structure as seen in natural complexity
  - Easier to monitor what is being learnt and to guide the machine to better subspaces
  - A good lower level representation can be used for many distinct tasks

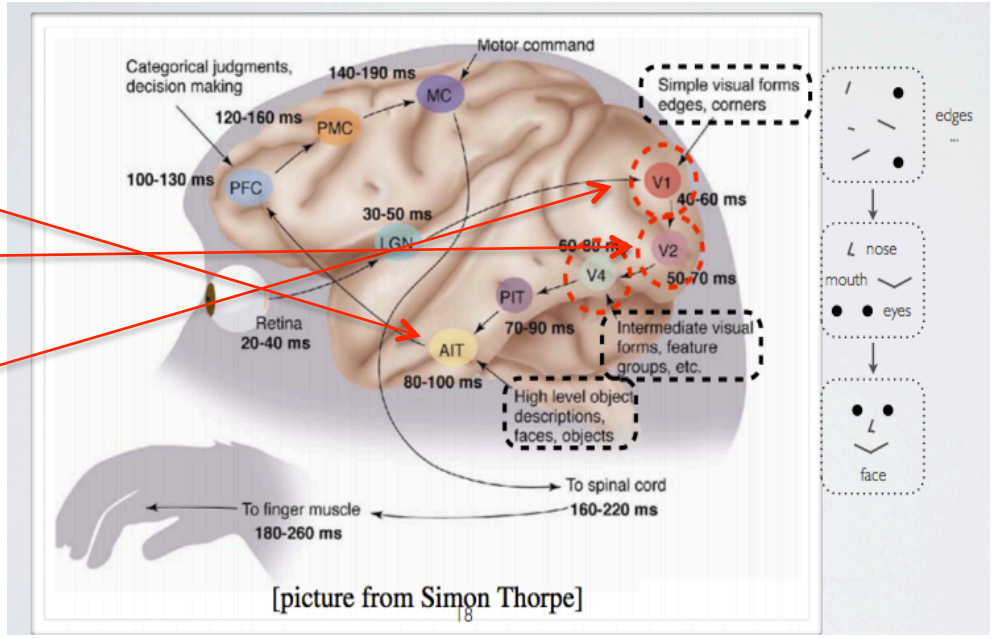
  Many parallels with the brain



Deep Learning
(Biological Inspiration)

- Deep Learning Networks have parallels with the brain's visual cortex
  - Both compute increasingly lower-dimension representations





Deep Learning algorithms

- Stack sparse coding algorithms
- Deep Belief Network (DBN) (Hinton)
- Deep sparse auto-encoders (Bengio)
- Many others: LeCun, Lee,Yuille, Ng,…
- Many authors at Google, FB, Baidu, …

- Unsupervised Deep Learning is where all the action is

Can Use Various Learning Algorithms with Deep Architectures

- Logistic regression
- Neural network
- Sparse auto-encoder
- Deep auto-encoder

Agenda

- ~~Goals for this Talk~~
- ~~What is Machine Learning?~~
  - ~~Kinds of Machine Learning~~
- ~~Machine Learning Fundamentals~~
  - ~~Shallow dive~~
  - ~~Inductive Learning: Regression and Classification~~
  - ~~Focus on Artificial Neural Networks (ANNs)~~
- ~~A Bit on Unsupervised Learning~~
- ~~Deep Learning~~
- Google PUE Optimization Machine Learning Application
- Next Steps

Google PUE Optimization Application[1]

- Straightforward application of ANN/supervised learning
  - Lots more happening at Google (and FB, Baidu, NFLX, MSFT,AMZN,…)
    - http://research.google.com/pubs/ArtificialIntelligenceandMachineLearning.html

- Use case: Predicting *Power Usage Effectiveness* (PUE)
  - Basically: They developed a neural network framework that learns from operational data and models plant performance
  - The model is able to predict PUE[2] within a range of 0.004 + 0.005 , or 0.4% error for a PUE of 1.1.

$$\text{PUE} = \frac{\text{Total Facility Energy}}{\text{IT Equipment Energy}}$$

- *"A simplified version of what the models do: take a bunch of data, find the hidden interactions, then provide recommendations that optimize for energy efficiency."*
  - *http://googleblog.blogspot.com/2014/05/better-data-centers-through-machine.html*

1 https://docs.google.com/a/google.com/viewer?url=www.google.com/about/datacenters/efficiency/internal/assets/machine-learning-applicationsfor-datacenter-optimization-finalv2.pdf
2 http://en.wikipedia.org/wiki/Power_usage_effectiveness

Google Use Case: Features

- Number of features relatively small (n = 19)

1. Total server IT load [kW]
2. Total Campus Core Network Room (CCNR) IT load [kW]
3. Total number of process water pumps (PWP) running
4. Mean PWP variable frequency drive (VFD) speed [%]
5. Total number of condenser water pumps (CWP) running
6. Mean CWP variable frequency drive (VFD) speed [%]
7. Total number of cooling towers running
8. Mean cooling tower leaving water temperature (LWT) setpoint [F]
9. Total number of chillers running
10. Total number of drycoolers running
11. Total number of chilled water injection pumps running
12. Mean chilled water injection pump setpoint temperature [F]
13. Mean heat exchanger approach temperature [F]
14. Outside air wet bulb (WB) temperature [F]
15. Outside air dry bulb (DB) temperature [F]
16. Outside air enthalpy [kJ/kg]
17. Outside air relative humidity (RH) [%]
18. Outdoor wind speed [mph]
19. Outdoor wind direction [deg]

Google Use Case: Algorithm

1. Randomly initialize the model parameters θ
2. Implement forward propagation
3. Compute the cost function $J(\theta)$
4. Implement the back propagation algorithm
5. Repeat steps 2-4 until convergence
   - or for the desired number of iterations

- Very standard…

Google Use Case: Details

- Neural Network
  - 5 hidden layers
  - 50 nodes per hidden layer
  - 0.001 as the regularization parameter (λ)

- Training Dataset
  - 19 normalized input parameters (features) per normalized output variable (the DC PUE)
    - Data normalized into the range [-1,-1]

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

  - 184,435 time samples at 5 minute resolution
    - *O*(2) years of data
  - 70% for training, 30% for cross validation

Google Use Case: PUE Predictive Accuracy

- Mean absolute error: 0.004
- Standard deviation: f 0.005

Increased error for PUE > 1.14 due to lack of training data

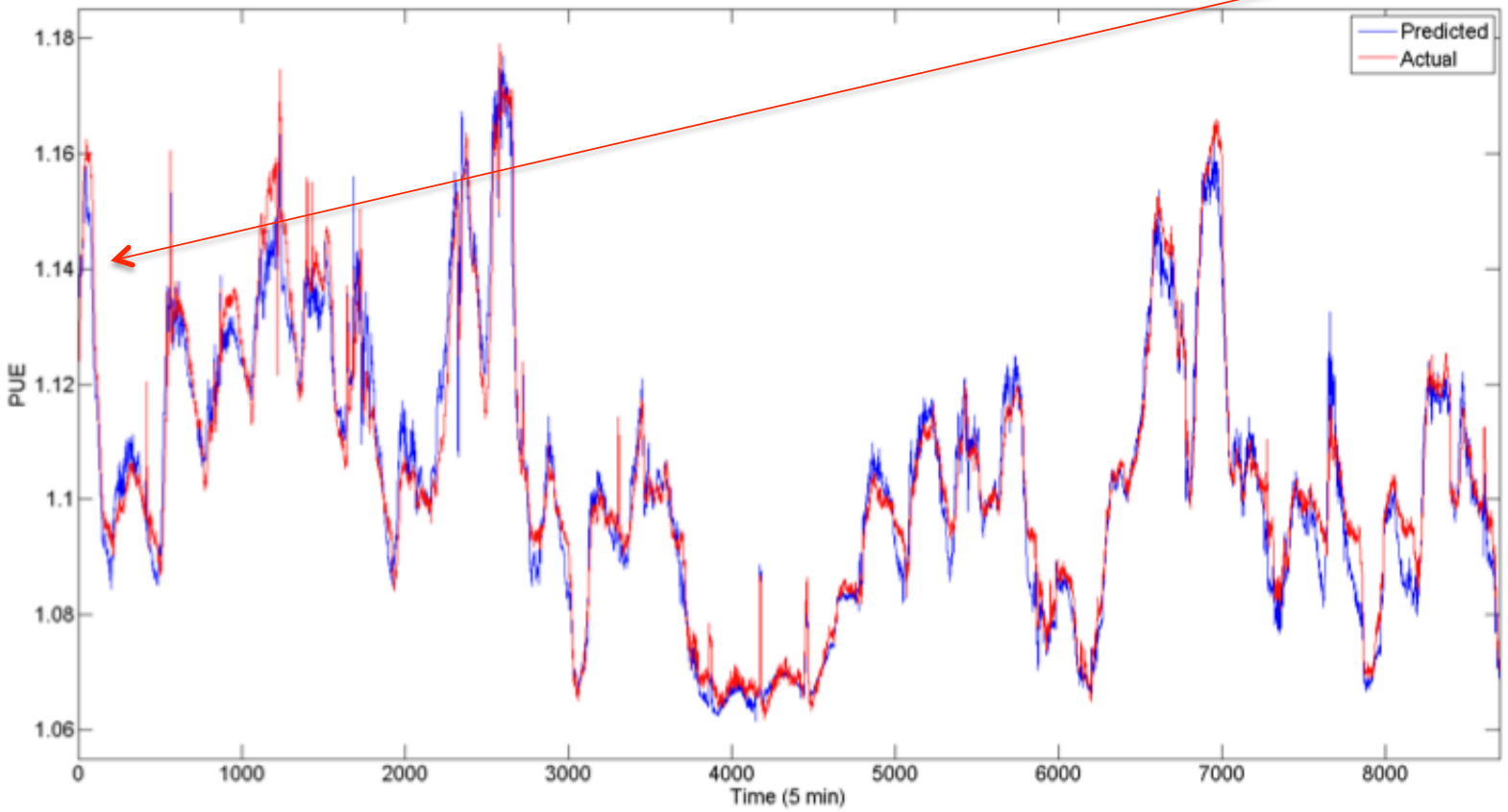
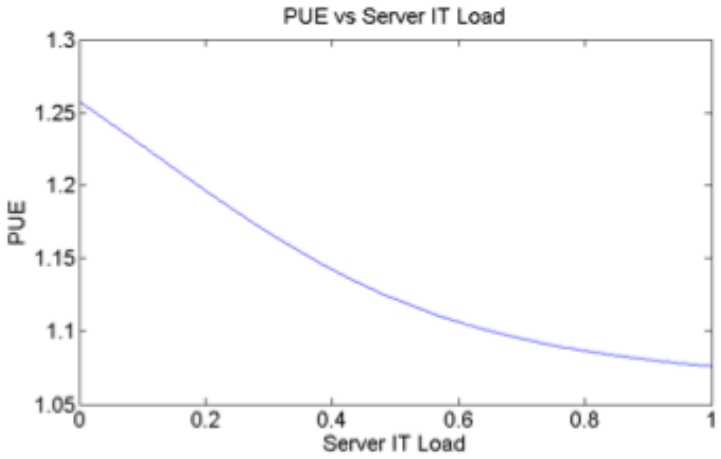


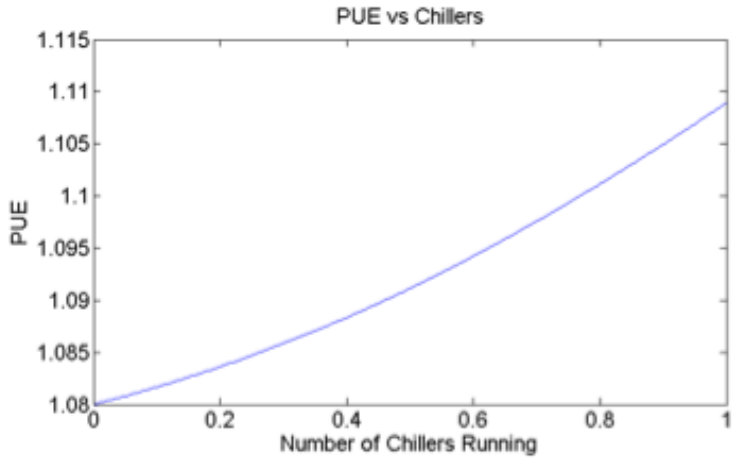
Fig. 3 Predicted vs actual PUE values at a major DC.

Google Use Case: Sensitivity Analysis

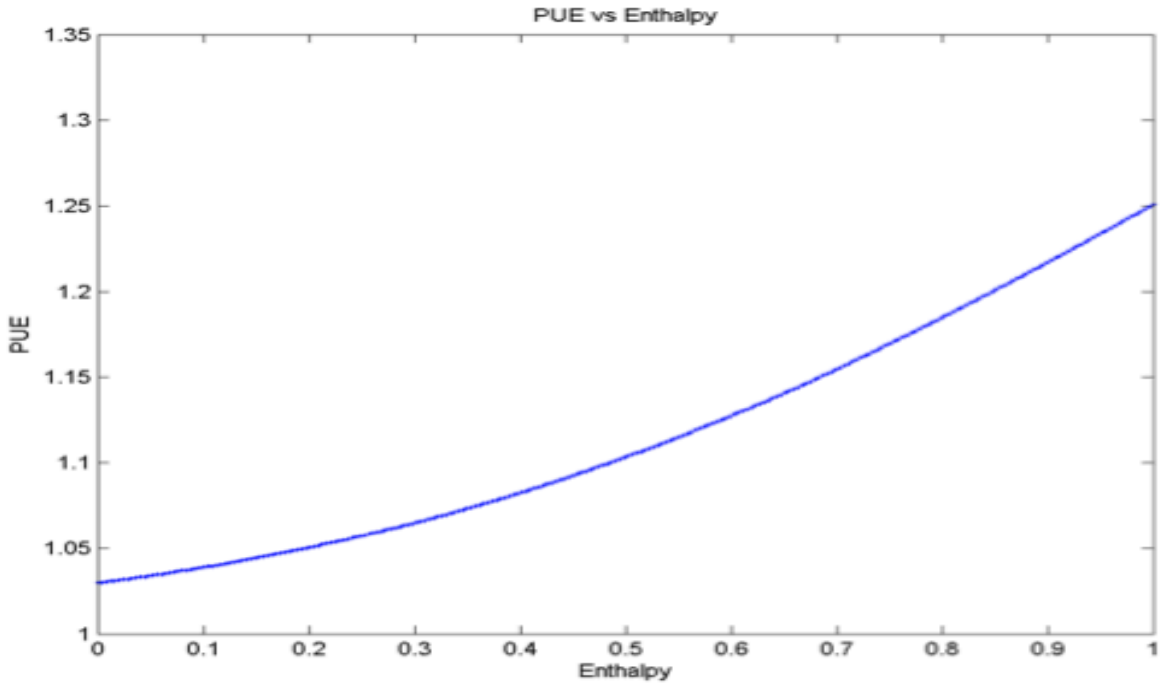
- After the model is trained, one can look at effect of individual parameters by varying one while holding the others constant

The relationship between PUE and the number of chillers running is nonlinear because chiller efficiency decreases exponentially with reduced load.





Google: Outside air enthalpy has largest impact on PUE



Relationship between PUE and outside air enthalpy, or total energy content of the ambient air. As the air enthalpy increases, the number of cooling towers, supplemental chillers, and associated loading rises as well, producing a nonlinear effect on the DC overhead. Note that enthalpy is a more comprehensive measure of outdoor weather conditions than the wet bulb temperature alone since it includes the moisture content and specific heat of ambient air.

Agenda

- ~~Goals for this Talk~~
- ~~What is Machine Learning?~~
  - ~~Kinds of Machine Learning~~
- ~~Machine Learning Fundamentals~~
  - ~~Shallow dive~~
  - ~~Inductive Learning: Regression and Classification~~
  - ~~Focus on Artificial Neural Networks (ANNs)~~
- ~~A Bit on Unsupervised Learning~~
- ~~Deep Learning~~
- ~~Google PUE Optimization Machine Learning Application~~
- Next Steps

Next Steps

Q & A

Thanks!