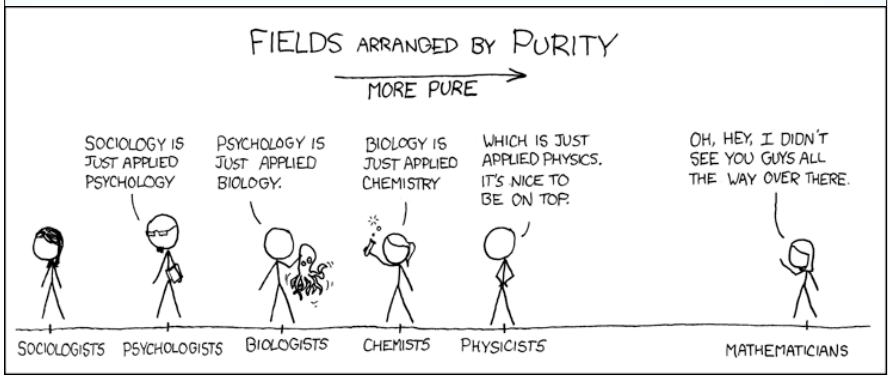
Welcome to the Wonderful World of Machine Learning

Mark Dredze
Machine Learning (CS 600.475)



Today's Topics

- Course Overview
  - Goals
  - Policies
- The Field of Machine Learning
  - What is it?
  - History
- Linear regression

Course Goals

- Learn the fundamentals of machine learning
- Learn to implement machine learning applications
- Learn how to apply machine learning to different settings

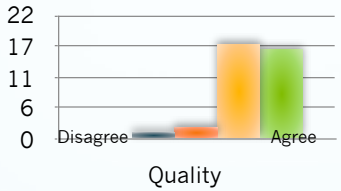
Course Policies

- Website: cs475.org
  - Bulletin Board
- Requirements
  - Programming: Python
  - Math
- Midterm
- Homeworks
  - Goals
  - Late Policy
- Final project
- Readings
- Grading
- Cheating

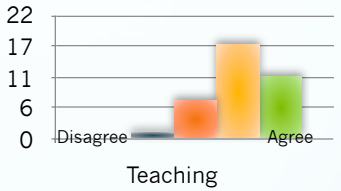
Todo List

- Get password for cs475.org account
- Sign up for an account on the message board
- Get a copy of the textbook
- Read about the course, including policies on the About section of the website
- Class registration
- Waitlist students: complete new poll

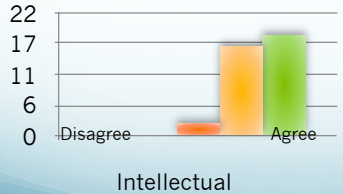
Course Ratings (Spring 2014)



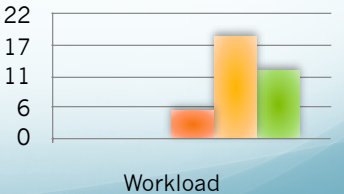
Quality



Teaching

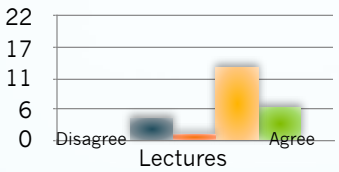


Intellectual

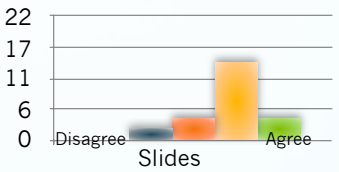


Workload

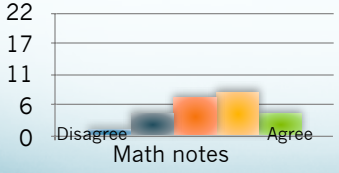
What Helped Learning?



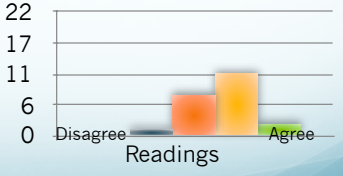
Lectures



Slides



Math notes



Readings

Machine Learning Foundations

Definition

- Machine learning allows computers to observe input and produce a desired output, either by example or through identifying latent patterns in the input.
- Data
  - What type of input?
  - What type of output?
- Patterns = algorithm
  - Intuition (empirical)
  - Objective (theoretical)

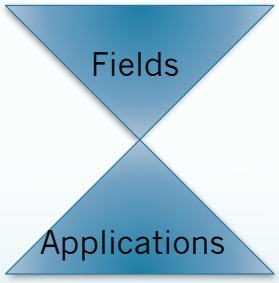
Different Definition

Fitting a function to data

- Fitting: Optimization, what parameters can we change?
- Function: Model, loss function
- Data: Data/model assumptions? How we use data?
- ML Algorithms: minimize a function on some data

A Diverse Topic

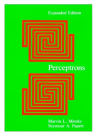
AI, Statistics, Linear algebra, Information Theory, Probability, Decision Theory



Vision, Robotics, NLP, Computational Biology, Bioinformatics, Speech, IR, Systems

Frank Rosenblatt (Cornell)
Linear classifier for character recognition
Based on neurons in the brain
By studying the Perceptron, "the fundamental laws of organization which are common to all information handling systems, machines and men included, may eventually be understood."

1959 1960 1969



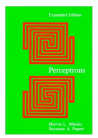
Marvin Minsky, Seymour Papert (MIT)
"Perceptrons: An Introduction to Computational Geometry"
Criticism of Perceptron demonstrating limitations
Research in Perceptrons stops for almost 20 years

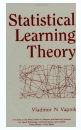
Arthur Samuel (IBM)
Checkers playing program
Rote learning- simulate many games to learn strategy
Early reinforcement learning

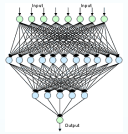
First workshop on Machine Learning
Second in 1983
Eventually became ICML in 1993

ICML

1959 1960 1969 1974 1980 1982



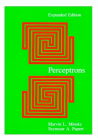


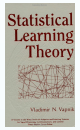
Multi-layer neural networks
1986· Backprop algorithm

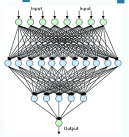
Vapnik and Chervonenkis
Statistical learning theory
Early idea behind SVMs
1990· kernels
1992· modern SVM

ICML

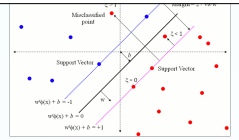
1959 1960 1969 1974 1980 1982 1985



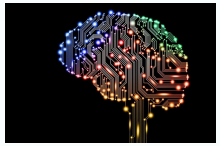


Rise of machine learning as a field
1985· AIStats
1985· UAI
1987·NIPS (90 papers, 2014: 1678 submissions, 414 papers)
1988· COLT



Rise of
the SVMs

3rd rise of
neural
networks
(now deep
learning)



1992 1993 2002 2007 2010 2016

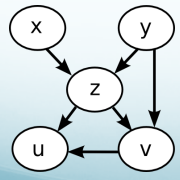
Second
death of
neural
networks

Rise of
the
graphical
models

Deep
learning
takes over

ML hits record size

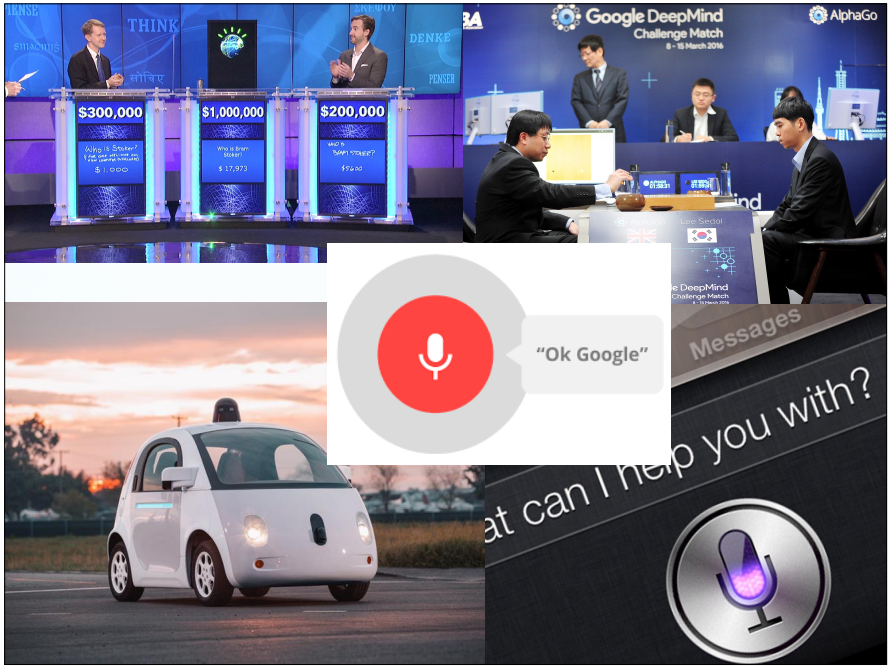
· 3000 people at NIPS 2015
· ICML 2016 doubles in size



Years are
approximate

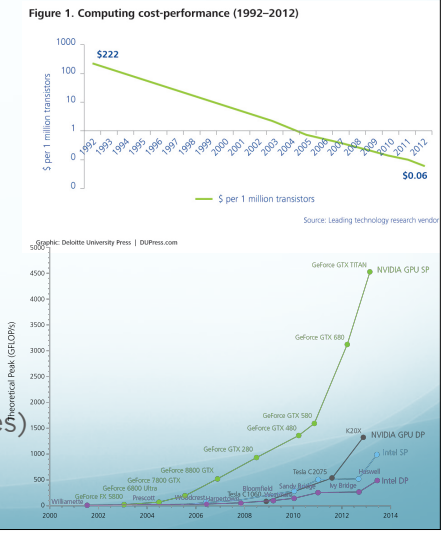
Machine Learning Today

- ~~Machine Learning~~
- Artificial intelligence
  - ML split from AI ~20 years ago, coming back
- Why?



Why success?

- (Somewhat) Better algorithms
  - Deep learning
- Better computers
- More data
- The rise of data science
  - More companies hire ML, more success (cycles)



ALLEN INSTITUTE *for* ARTIFICIAL INTELLIGENCE

Google DeepMind

OpenAI

Stanford | One Hundred Year Study on Artificial Intelligence (AI100)

Linear Regression
Our First Algorithm

If she loves you more each and every day,
by linear regression she hated you before you met.

Why Start Here?

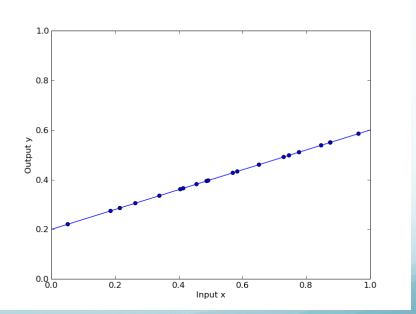
- Linear regression is well known
  - Not strictly a machine learning algorithm
- Learn about fundamentals of machine learning using a simple example

Example

- I have a large number of undergraduate applications for Johns Hopkins. I want to accept students who I think will get the highest GPA (0-4.0).
- Goal: Predict an applicant's GPA
- Data: Previous applications and resulting GPAs
- How do I do this?

Data Model

- Assume dependent variable (y) can be modeled by a *linear function* of the input variables (x)
- $y = \mathbf{w}\mathbf{x} + b$
- 2 dimensions
  - Compute w and b from two points
- Solution?
  - Given y and x, solve for w



Regression

- Data $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N} \quad \mathbf{x}_i \in \Re^M \quad y_i \in \Re$ ← Continuous Values
- Learn: a mapping from x to real valued y
  - f(x) = y
- Examples
  - GPAs
  - Stock price
  - Miles per gallon
  - Age of author

Try it at Home!

- Linear regression demo
- http://mste.illinois.edu/users/exner/java.f/leastsquares/
- http://onlinestatbook.com/2/regression/linear_fit_demo.html
- https://www.geogebra.org/m/FUe3HfRf

Recall Definition

Fitting a function to data

- Fitting: Solve for w given y and x
- Function: linear function
- Data: assume dependent variable linear combination of independent variables
- Minimize a function
  - What function are we minimizing?

What is the goal?

- You probably know linear regression from statistics
  - "In statistics, linear regression is an approach for modeling the relationship between a scalar dependent variable y and one or more explanatory variables (or independent variables) denoted X." (Wikipedia)
- Our goal: predict correctly the next example
  - Minimize: reduce prediction error
    - More to say about this later

Loss Functions

- Machine learning algorithms minimize loss functions
  - Or some substitute for a loss function
  - The best solution minimizes the loss function*
- Definition
  - A function that maps between (true label, prediction) -> non-negative number
  - 0 = perfect prediction

* Maybe

Loss: What We Minimize

- Loss measures the badness of our prediction
- What's a good loss function?
  - It depends on task and goals
- Regression loss function?
  - Proposal: How far are you from the correct answer

Sum of Squares Loss

$$f(x) - y$$

$$(f(x) - y)^2$$

$$\sum_{i=1}^{n} (f(x_i) - y_i)^2$$

GPA Example
$(3.5\text{-}3.0)^2 = .25$

Over all answers

Correct answer

$$\sum_{i=1}^{n} (w \cdot x_i - y_i)^2$$

Predicted answer

Recall Definition

Fitting a function data

- Fitting: Solve for w given y and x
- **Function: linear function**
- Data: assume dependent variable linear combination of independent variables
- Minimize a function
  - What function are we minimizing?

Hypothesis Class

- Learning algorithm selects hypothesis from hypothesis class
- Hypothesis class
  - A set of possible hypotheses (functions) that can be used to label the data
  - Can be finite or infinite
- Each learning algorithm has a hypothesis class
  - Fitting selects the best hypothesis using observed data

What is best hypothesis?

Choosing Hypothesis Class

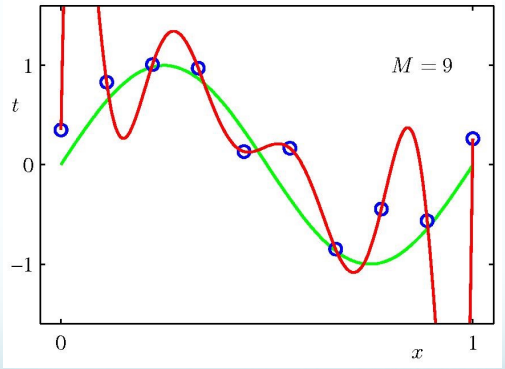
- What sort of hypothesis class do we want?
- The hypothesis class should contain the optimal hypothesis
- The hypothesis class for which our algorithm will find the best performing hypothesis

What is the difference?

Hypothesis Tradeoff

- Rich hypothesis class
  - Over-fitting
- Easier to search hypothesis class
  - Under-fitting
- Realization: we are unlikely to ever find the hypothesis that exactly explains the data
- Simplifying assumptions help find a reasonable hypothesis
- Select hypothesis class based on knowledge of data

Under/Over Fitting



Hypothesis Class

- What is the hypothesis class of linear regression?
- Linear functions
  - All possible linear functions encoded by parameters w
  - Hypothesis chosen by setting parameter values w
- How large is the hypothesis class?
  - Infinite

What is Learning? Another View

- Select a hypothesis from the hypothesis class
  - Model parameters correspond to hypotheses
  - Learn parameters of model based on data
- How do we write learning algorithms?
  - Theory: Objective driven
    - Write an objective that you want to minimize
    - Develop a procedure to minimize the objective
      - This is called the learning algorithm
  - Empirical: intuition driven
    - Many algorithms based on motivation and heuristics
    - Post hoc analysis of objective

Recall Definition

Fitting a function to data

- Fitting: Solve for w given y and x
- Function: linear function
- **Data: assume dependent variable linear combination of independent variables**
- Minimize a function
  - What function are we minimizing?

Learning Settings

- What information is available for learning?
  - What does the data look like?
  - How is it annotated?
- What output is desired?
  - What should the algorithm produce?
  - How will it be used?

Supervised Learning

- Learning with a teacher
  - Explicit feedback in the form of labeled examples
  - Goal: make prediction
  - Pros: Good performance
  - Cons: Labeled data is difficult to find
- Examples
  - Regression!
  - Classification
    - Sort documents by topic
  - Ranking
    - Sort web pages

Unsupervised Learning

- Learning by oneself
  - Only observed unlabeled examples
  - Goal: uncover structure in data
  - Pros: Easy to find lots of data
  - Cons: Finding patterns of interest
- Examples
  - Clustering
    - Group emails by topic
  - Manifold learning
    - Find a low dimensional data representation

Semi-Supervised Learning

- Labeled examples + unlabeled examples
- Lots of ways to do this
  - Use unlabeled to guide learning in classification
    - Some documents labeled by topic, lots of unsorted docs
  - Graph based models for labeling new data
    - Label propagation
  - Other weak forms of supervision
    - A list of names, learn to extract more

Reinforcement Learning

- Learn a behavior policy by interacting with the world
  - How to navigate in a world
  - Success measured by rewards received by actions
  - Maximize rewards - costs
- No examples
  - You don't know how you did till its over
    - Ex. Chess- was that a good move?
      Did you eventually win?
- Examples
  - Chess (and checkers) and game agents
  - Robot control
  - Piloting an airplane
  - Go

How Do We Represent Data?

- Data is complex
- How does a computer algorithm see data?

High Dimensional Vectors

- A learning example is a vector of length M x
- Examples a drawn from an underlying distribution

$$x_i \in \Re^M$$

- Each dimension represents a feature
  - Feature functions

$$x[j] = f_j(document)$$

- A collection of N examples

$$\{x_i\}_{i=1}^N$$

What Does Data Look Like?

Representations

$$x$$

- Designing feature functions is critical
  - Well designed representations greatly effect performance
- How should we design features?
  - Features are application specific
  - You need to know about biology/vision/speech/etc.
- Since this is domain specific we won't talk much about it
- More on this in last lecture

Recall

- Our goal: predict correctly the next example
  - Minimize: reduce prediction error

Goal of Learning

- "Reduce prediction error"
  - On what?
- True error

$$error_D(h) = P_{x \in D}[\ell(h(x_i), y_i) \neq 0]$$

  - We need infinite data to measure this!

Goal of Learning

- If we can't measure true error, how do we judge learning success?
- Should an algorithm maximize performance on observed data?
- Proposal: measure error on the given data
  - Call this the "training data"
  - Is this a good idea?

Measuring Error

- Very bad idea
- Recall: machine learning cares about prediction (the future)
  - How well will the system do once deployed?
- Memorizing the training data is easy
  - Most hypothesis classes are rich enough to exactly learn the training data

Difficult to Understand

- People routinely make this mistake!
- "The team from the eastern most state has always won the Superbowl when their quarterback is taller and the temperature is above 60°"
- True stories from my experiences
  - Project accepted to predict hot real estate markets
    - Promising because very high accuracy
    - Problem: measured error on training data
  - Paper submission to major conference claimed to have solved problem with 100% accuracy
    - Trained on the test data
  - Paper submission showed high accuracy on classification task
    - Data *written* by researchers with task in mind

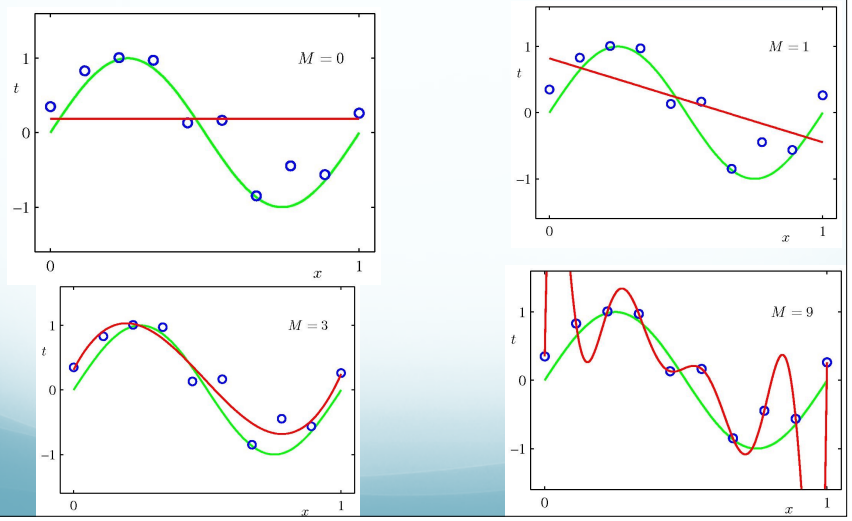
Generalization

- Generalization
  - The ability of an algorithm to generalize knowledge learned from observed data to new data
- Simple example: memory based classifier
  - Binary classification
  - Train: remember each example
  - Test: if we have seen an example before, report label
    - Otherwise, guess randomly
- Train error: 0% Test error: 50%
  - This is called over-fitting

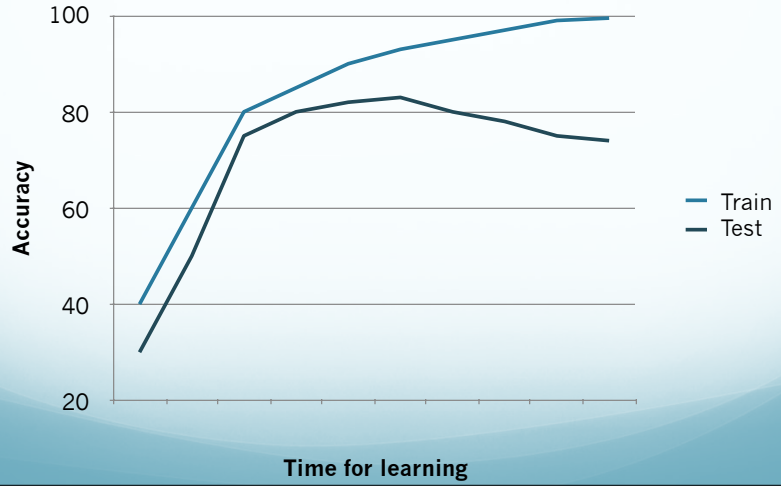
Bias vs. Variance

- How do we achieve good generalization?
- Tradeoff bias and variance
  - Bias- favor certain predictions
  - Variance- diversity of predictions
- Under-fit observed data
  - Favors bias- large changes to input have same output
- Over-fit observed data
  - Favors variance- small changes to input can dramatically change output
- Tradeoff key to generalization to new data

Under/Over Fitting



Typical Learning Curves



Measuring Error: The Right Way

- Collect two sets of data
  - Train data- use for training algorithm
  - Test data- only use for evaluation
    - Only good if you've never seen it before, not if you continuously tune on it
- How do we balance bias/variance?
  - Tune parameters on development/validation data

Two Common Methods

- Train/dev/test
  - Use held out sets for evaluation
  - Good when you have lots of data
- Cross validation
  - Create many train/test splits
  - Randomly sample train and test data splits
  - Divide data into folds, use each fold for testing once
  - Reasonable when you don't ave enough data

What Causes Error?

- Noise error
  - An example has an incorrect or inconsistent label
  - Our data representation fails to encode necessary information
- Model error
  - Hypothesis class is deficient
- Parameter estimation error
  - The model parameters are wrong
- Search error
  - We made a mistake in scoring a prediction
  - Common in tasks with complex output

Recall Definition

Fitting a function to data

- **Fitting: Solve for w given y and x**
- Function: linear function
- Data: assume dependent variable linear combination of independent variables
- Minimize a function
  - What function are we minimizing?

Linear Regression

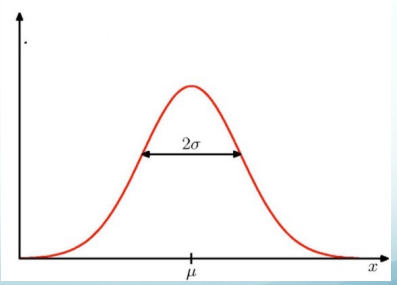
- We assumed output (y) linear combination of inputs
- This is wrong
  - Rarely is data actually linear
- But realistic assumption may be too complex
- A reasonable middle ground?

Noise from a Gaussian Distribution

$$N(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}$$

$$E[x] = \mu$$

$$\mathrm{var}[x] = \sigma^2$$



Noise

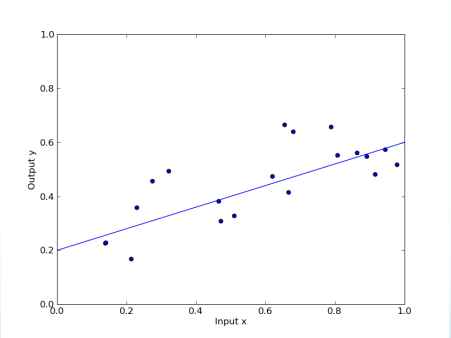
- Assume output permuted by Gaussian noise

$$y = h_w(x) + \varepsilon$$

$$\varepsilon \sim N(\mu, \sigma^2), \quad \mu = 0, \quad \sigma^2 = 1$$

- The data isn't really generated in this way
  - Assume that it is for sake of modeling

Linear Regression with Noise



Probability of Output

$$p(y \mid x, w, \sigma^2) = N(y, \sigma^2)$$
$$= N(h_w(x), \sigma^2)$$

Least Squares Regression

- Fitting: Solve for w given x and y
- Function: Linear function + Gaussian noise
  - Loss function: squared error
  - Assumes output (mostly) a linear combination of input
- Data: fit a model to training data, evaluate on held out data
- Minimize a function
  - What function are we minimizing?

Which Function are we Minimizing?

- Which is the best hypothesis?
  - Which setting of the parameters w is best?
- Select the hypothesis that *best explains* the observed data
- Select the hypothesis that minimizes the error

Explaining the Data

- What does it mean to explain the data?
  - Maximize the likelihood of the data
  - Likelihood = probability of observing data
- Writing likelihood
  - Assume data generated from our linear regression model

Maximum Likelihood For Gaussians

Maximum Likelihood for Regression

Sources of Error

- Noise error
  - An example has an incorrect or inconsistent label
  - Our data representation fails to encode necessary information
- Model error
  - Hypothesis class is deficient
- Parameter estimation error
  - The model parameters are wrong
- Search error
  - We made a mistake in scoring a prediction
  - Common in tasks with complex output

Bias?

- Gaussians: maximum likelihood estimate is biased
  - This is ok if we have infinite data
  - We never have infinite data!
- Over-fitting: avoid it by favoring certain solutions
- Regularization
  - Add term to objective to favor different considerations
  - What should we favor?
    - Occam's Razor: simpler is better
    - Favor small weights
      - Our parameters should be small

Regularized Least Squares

Regularized Least Squares

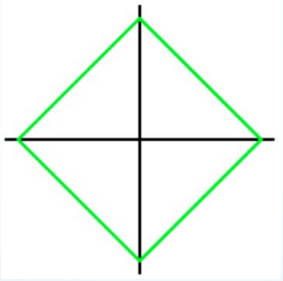
- Tradeoff: low error and small weights

$$E_D(w) = \frac{1}{2}\sum_{i=1}^{n}(y - w \cdot x)^2$$

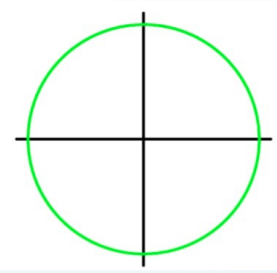
$$E_W(w) = \frac{1}{2}w^T w$$

$$E_D(w) + \lambda E_W(w) = \frac{1}{2}\sum_{i=1}^{n}(y - w \cdot x)^2 + \lambda\frac{1}{2}w^T w$$

Regularization Behavior



q=1 L1 (Lasso)



q=2 L2 (quadratic)

Bias vs. Variance

- Expected squared loss can be written as

$$\mathrm{E}[L] = \int \{f(x) - h(x)\}^2 p(x)dx + \int \{h(x) - y\}^2 p(x,y)dx\,dt$$

  - f(x)- prediction function
  - h(x)- true regression function
  - y- provided label (noisy)

  - First term- minimize loss
  - Second term- error from noise

Bias vs. Variance

- Imagine we can sample many datasets D from the underlying distribution
- Integrate the first term (which represented accuracy of the model)

$$(f(x,D) - h(x))^2$$

  - Depends on a particular sample of data D
- What is the expectation of this term over many samples of D?

Bias vs. Variance

$$\mathrm{E}_D[(f(x,D) - h(x))^2] =$$
$$(\mathrm{E}_D[f(x,D)] - h(x))^2 + \mathrm{E}_D[(f(x,D) - \mathrm{E}_D[f(x,D)])^2]$$

Bias Variance

- For learning we want to minimize this function
- The result is a tradeoff between bias and variance

Parameter Tradeoff

- The regularization parameter controls bias vs. variance
- Higher λ = more regularization
  - Favors bias
- Lower λ = less regularization
  - Favors variance

Problems

- Maximum likelihood under-estimates variance and over-fits
  - Try to fix using regularization
- How do we decide model complexity?
  - Parameter tuning on held out data
- Is there a better way?
  - Bayesian methods
  - more on this later

Summary: Machine Learning Fundamentals

Fitting a function to data

- Fitting: Optimization, what parameters can we change?
- Function: Model, loss function
- Data: Data/model assumptions? How we use data?
- ML Algorithms: minimize a function on some data

Summary: Machine Learning Fundamentals

- Data representation $\{(x_i, y_i)\}_{i=1}^{N} \quad x_i \in \Re^M \quad y_i \in \Re$
- Loss functions
- Hypothesis class and tradeoffs
- Generalization and bias/variance tradeoff
- Learning settings: Supervised and unsupervised
- Regularization
- Sources of error

Next Time: On to Classification!

From Regression to Classification with Logistic Regression

Mark Dredze

Machine Learning
CS 600.475

Classification

- Data $\{(x_i, y_i)\}_{i=1}^{N} \quad x_i \in \Re^M \quad y_i \in L$
- Learn: a mapping from x to discrete value y
  - f(x) = y
- Examples
  - Spam classification
  - Document topic classification
  - Identifying faces in images

Binary Classification

- We'll focus on binary classification
  - $y_i \in \{0,1\}$
- Usually easy to generalize to multi-class classification

Different Definition

Fitting a function to data

- Fitting: Optimization, what parameters can we change?
- **Function: Model, loss function**
- Data: Data/model assumptions? How we use data?
- ML Algorithms: minimize a function on some data

Evaluation

- Accuracy

$$\frac{\text{number of correct predictions}}{\text{total number of predictions}}$$

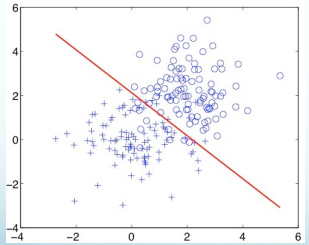
- Other measurements appropriate for some tasks
  - Ex. we care more about certain types of mistakes

Regression

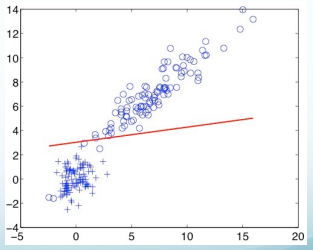
- Least squares regression
  - Outputs real number for each example
- It seems that classification should be easier!
- Let's use regression for classification
  - Learn least squares regression model $f_w(x)=y$
    - $f_w(x)=w^Tx$
  - If $y>0$, predict "True (1)"
  - If $y\leq 0$ predict "False (0)"

Regression for Classification

- $f_w(x)=0$ partitions the input space into two class specific regions
  - Linear decision boundary



Good



Bad

Figures by Tommi Jaakkola

Regression for Classification

- Mismatch between regression loss and classification
  - Classification: accuracy
  - We don't care about large vs. small values of output
- Outliers problematic
  - Prediction of 42 for example is fine for classification, bad for regression
- We need output to be either 1 or 0

Machine Learning

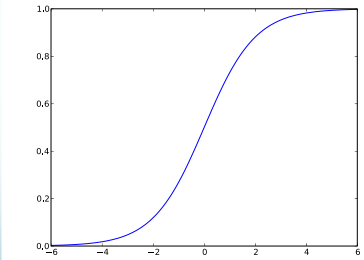
Fitting a function to data

- Fitting: Solve for w given y and x
- Function: Regression uses squared loss
  - Bad match for our task!
- Data: assume dependent variable linear combination of independent variables
- Our loss function doesn't match classification goals

Logistic Function

- Quick fix: apply a function to the output of regression that gives desired valued
- Logistic function
  - Outputs between 0 and 1
  - Scaling parameter α
  - Most outputs are close to 1 or 0

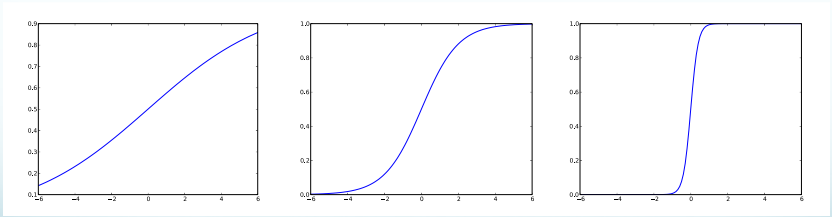
$$g_\alpha(x) = \frac{1}{1+e^{-\alpha x}}$$



α = 1

Logistic Function

$$g_\alpha(x) = \frac{1}{1+e^{-\alpha x}}$$



α = 0.3 α = 1 α = 5

Logistic Regression

- We can combine the logistic function and our regression model

$$g(w^T \cdot x_i) = \frac{1}{1+e^{-w^T \cdot x_i}}$$

- Notice: as $w^T \cdot x_i$ becomes:
  - Large- output closer to 1
  - Small- output closer to 0

Probabilistic View

- In regression we modeled probability of the output
- Probability of the example and classification?
  - p(x,y)?
  - p(x,y) = p(x|y) p(y)
  - Since we know x, we want to maximize y
  - p(x|y)p(y) = p(y|x)p(x)
  - Since p(x) is fixed:
  - $\arg\max_{y=0,1} p(x \mid y)\, p(y) = \arg\max_{y=0,1} p(y \mid x)$

Why?

- We can now write the distribution as
$$p_w(y=1 \mid x) = \frac{1}{1+e^{-w^T \cdot x}}$$
- Which implies that
$$p_w(y=0 \mid x) = \frac{e^{-w^T \cdot x}}{1+e^{-w^T \cdot x}}$$
- The odds of the event is then
$$\frac{p_w(y=1 \mid x)}{p_w(y=0 \mid x)} = \exp(-w^T \cdot x)$$
- And the log-odds are
$$\log \frac{p_w(y=1 \mid x)}{p_w(y=0 \mid x)} = -w^T \cdot x$$

Generalized Linear Models

- Decision boundary/surface
  - An n-1 dimensional hyper-plane that separates the data into two groups
  - These are linear functions of x, even though logistic is not linear
- Generalized linear models
  - A linear model whose output is passed through non-linear function
- Hypothesis class
  - Linear decision boundaries

Logistic Regression Decisions

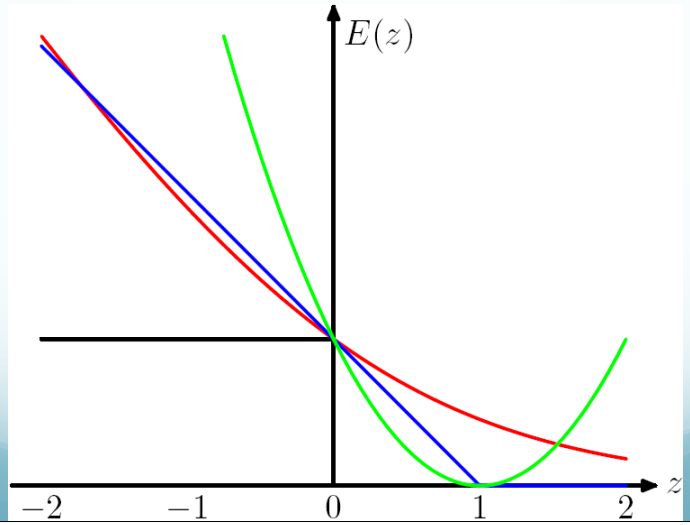
- Given parameters w, how do we make predictions?
$$p_w(y=1 \mid x) = \frac{1}{1+e^{-w^T \cdot x}}$$
- If output > .5, predict 1, else predict 0
- In addition to prediction, we have confidence in prediction
  - Confidence is the probability of the prediction

Logistic Regression

Fitting a function to data

- Fitting: Solve for w given y and x
- Function: Generalized linear function: logistic over regression
- Data: assume dependent variable linear combination of independent variables

Logistic Loss



Objective Function: Likelihood

- Conditional data likelihood

$$p(Y \mid X, w) = \prod_{i=1}^{n} p(y_i \mid x_i, w)$$

Conditional Log Likelihood

$$p(Y \mid X, w) = \prod_{i=1}^{n} p(y_i \mid x_i, w)$$

$$\ell(Y, X, w) = \log p(Y \mid X, w) = \sum_{i=1}^{n} \log p(y_i \mid x_i, w)$$

$$p(y = 1 \mid x, w) = \frac{1}{1 + e^{-w^T \cdot x}} \qquad p(y = 0 \mid x, w) = \frac{e^{-w^T \cdot x}}{1 + e^{-w^T \cdot x}}$$

Logistic Regression

Fitting a function to data

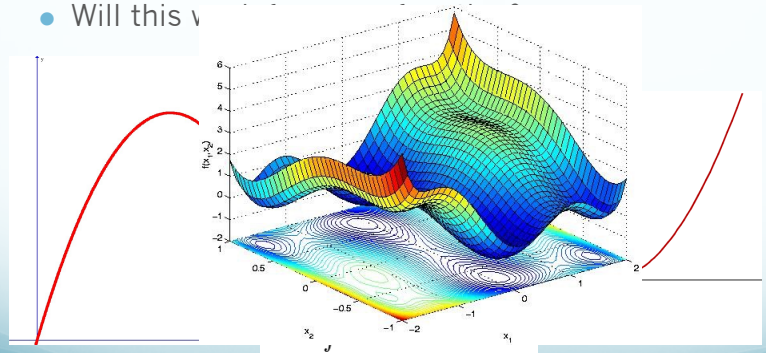
- **Fitting: Solve for w given y and x**
- Function: Generalized linear function: logistic over regression: conditional log likelihood
- Data: assume dependent variable linear combination of independent variables

Function Optimization

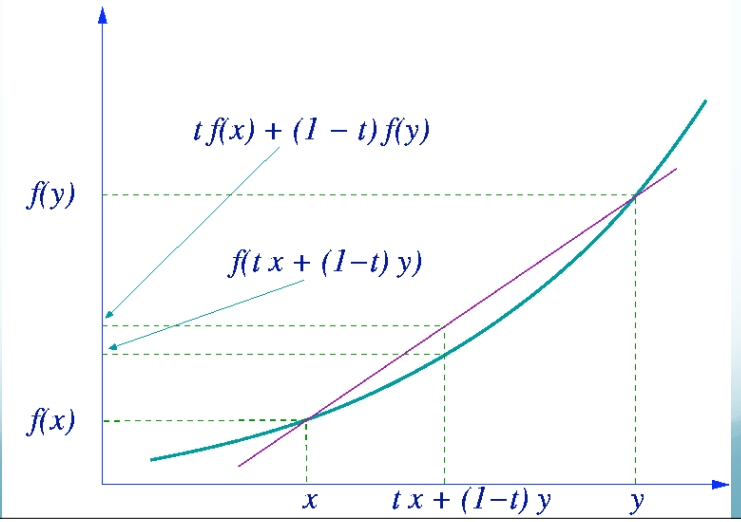
- We have a function and want to maximize/minimize it
- How do we find the point at which the function reaches its max/min?

Function Optimization

- Take the derivative, set it equal to 0, solve!
- Will this



Convex Functions



Maximum Likelihood Estimation

- MLE: Find the value at which the likelihood is maximized
  - We'll talk about other options later in the semester
- Given the conditional log likelihood
  - Take the derivatives for parameters w
  - Set each derivative to 0
  - M equations and M variables
  - Solve for w
- Problem
  - No closed form (analytical) solution for w

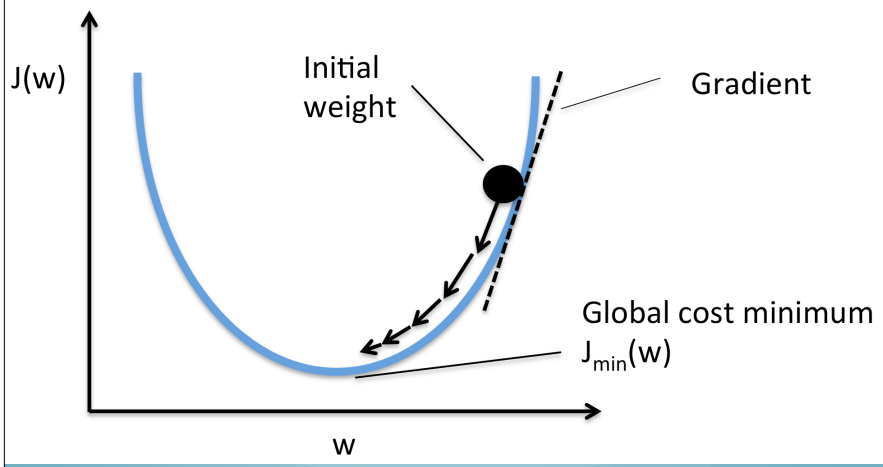
Convex Optimization

- The conditional maximum likelihood is concave
  - There is a single maximal solution
- We can maximize using convex optimization techniques
  - Its easy to optimize convex functions
  - There are **many** convex optimization algorithms

Gradient Descent

- First order method: needs first order derivatives
- Assuming F(x) is defined and differentiable, then F(x) decreases fastest if we go from x in the direction of the gradient of F
  - $-\nabla F(x)$· vector of partial derivatives of F
  - $x' = x - \gamma \nabla F(x)$ · Update
  - For sufficiently small values of γ, the value of the function will get smaller

Gradient Descent



Derivatives

$$\frac{\partial \ell(Y, X, w)}{\partial w} = \sum_{i=1}^{n} (y_i - p(y_i = 1 \mid x_i, w)) x_i = 0$$

- The derivative is 0 when $y_i = p(y_i=1 \mid x_i, w)$
  - Minimize the prediction error

Gradient Descent Solution

$$w^{(t+1)} = w^t + \gamma \frac{\partial \ell(Y, X, w)}{\partial w}$$

$$\frac{\partial \ell(Y, X, w)}{\partial w} = \sum_{i=1}^{n} (y_i - p(y_i = 1 \mid x_i, w)) x_i = 0$$

$$w^{(t+1)} = w^t + \gamma \sum_{i=1}^{n} (y_i - p(y_i = 1 \mid x_i, w)) x_i$$

Algorithm: Logistic Regression

- Train: given data X and Y
  - Initialize w to starting value
  - Repeat until convergence
    - Compute the value of the derivative for X,Y and w
    - Update w by taking a gradient step
- Predict: given an example x
  - Using the learned w, compute p(y|x,w)

$$p(y = 1 \mid x, w) = \frac{1}{1 + e^{-w^T \cdot x}}$$

- Note: many other optimization routines available

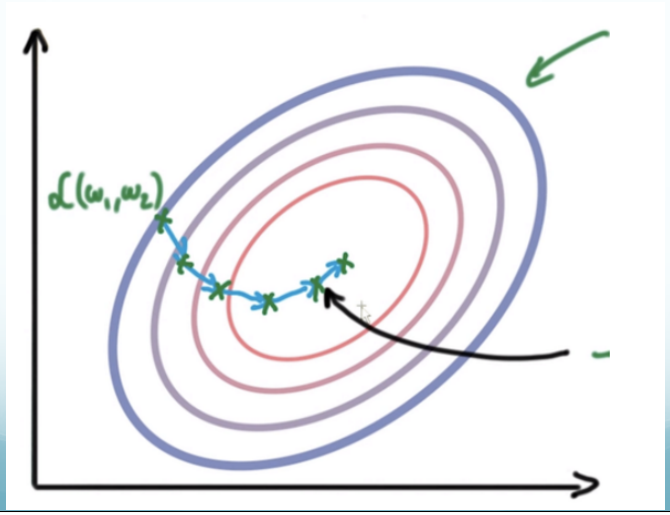
Gradient Based Optimization

- Multiple methods available for optimizing the same objective function
  - First order methods
  - Second order methods
  - Adaptive methods
  - ...

Alternate Methods

- Batch gradient descent
  - Utilize the gradient of all the data
  - Slow: need to consider all the data before making a single update

Gradient Descent



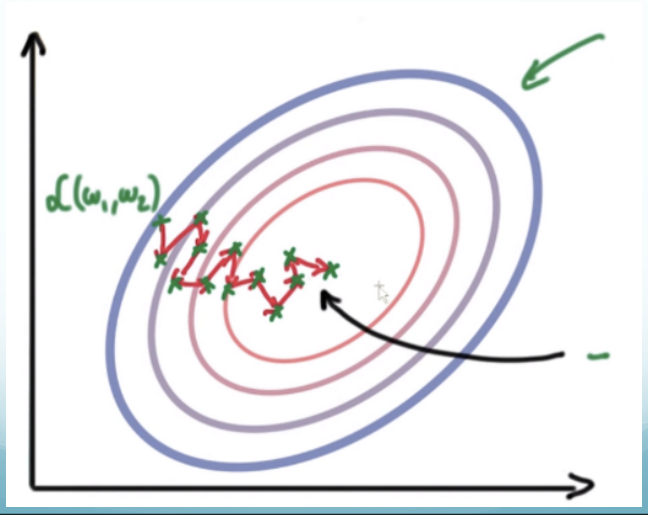
Stochastic Updates

- Compute the gradient on a single example at a time

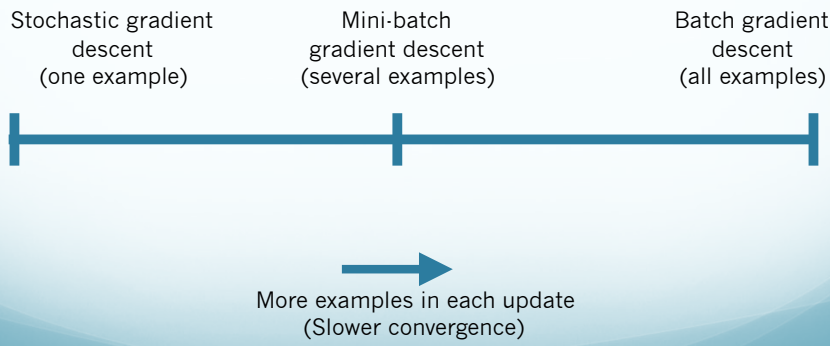
$$w^{(t+1)} = w^t + \gamma \sum_{i=1}^{n} \{y_i - p(y_i = 1|x_i, w)\} x_i$$

$$w^t + \gamma\{y_1 - p(y_i = 1|x_1, w)\}x_1 + \gamma\{y_2 - p(y_2 = 1|x_2, w)\}x_2 \ldots$$

Gradient Descent



Update Frequency



Regularization

- Same over-fitting problems as least squares
- Add regularization term to objective to favor different considerations
- Similar options
  - Quadratic regularization (L2)
  - L1 regularization (sparse solutions)
  - For each regularization optimize new objective function

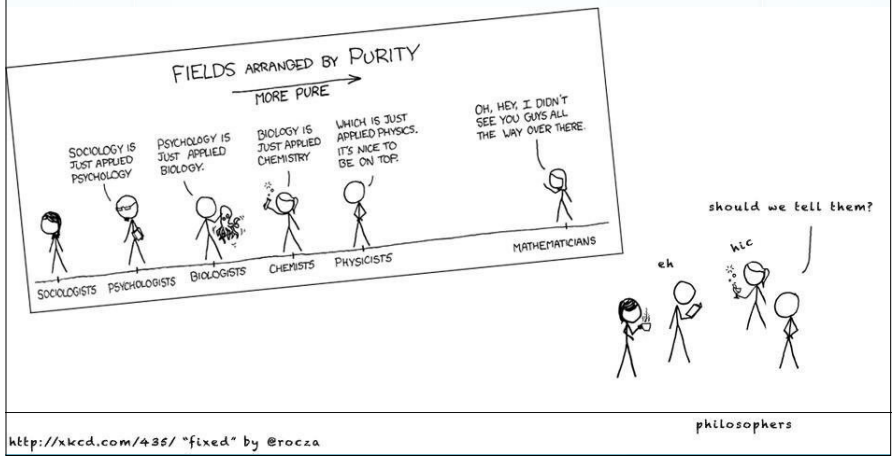
Summary

- Logistic regression
  - Learn p(y|x) directly with functional form of distribution
  - Maximize the data conditional log-likelihood
  - Equivalent to linear prediction
    - Decision rule is a hyper-plane
  - Regularization to prevent over-fitting

Perceptron

Mark Dredze

Machine Learning
CS 600.475



Different Definition

Fitting a function to data

- Fitting: Optimization, what parameters can we change?
- Function: Model, loss function
- Data: Data/model assumptions? How we use data?
- ML Algorithms: minimize a function on some data

Setting

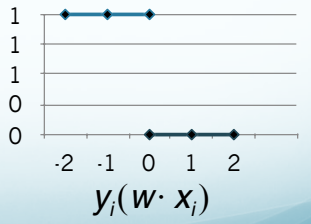
- Stochastic gradient descent
  - One example at a time
- Linear classifier

$$w \cdot x_i$$

- Let's take the simplest (and most direct) loss function we can think of

0/1 Loss function

- If we are wrong: loss of 1
- If we are correct: loss of 0
  - Pretty much the simplest loss function around
- This mean:
  - Only make a change when we make a mistake



Hard to Minimize

- Minimizing the 0/1 loss is difficult
  - Step function
- Replace with a similar form

$$L_w(y) = \sum_i^N max(0, -y_i w \cdot x_i)$$

- Single example

$$L_w(y_i) = max(0, -y_i w \cdot x_i)$$

Gradient

$$\partial L_w(y_i) = \begin{cases} 0 & y_i w \cdot x_i > 0 \\ -y_i x_i & y_i w \cdot x_i < 0 \end{cases}$$

- Ignore case when $w \cdot x_i = 0$ (tie)

Update Rule

$$w^{i+1} = w^i + \eta \partial L_w(y_i)$$

$$w^{i+1} = w^i + \eta(y_i - \hat{y}_i)x_i$$

$$\hat{y}_i = \text{sign}(w \cdot x_i)$$

Update

- Why is this a good update? $w^{j+1} = w^j + \eta\, y_i\, x_i$

$$\begin{aligned}(w^{j+1} \cdot x_i)y_i &= (w^j \cdot x_i)y_i + \eta(x_i y_i)(x_i y_i) \\ &= (w^j \cdot x_i)y_i + \eta y_i y_i (x_i x_i) \\ &= (w^j \cdot x_i)y_i + \eta \,\|\, x_i \,\|^2 \\ &> (w^j \cdot x_i)y_i\end{aligned}$$

- Our prediction has improved
  - This says nothing about seeing the example in the future
  - The prediction may still be incorrect
  - We are just moving in the right direction

Algorithm: Perceptron

- Initialize w and η
- On each round
  - Receive example x
  - Predict $\hat{y} = \text{sign}(w \cdot x)$
  - Receive correct label $y \in \{+1, -1\}$
  - Suffer loss $\ell_{0/1}(y, \hat{y})$
  - Update w: $w^{i+1} = w^i + \eta\, y_i\, x_i$

Perceptron

Fitting a function to data

- Fitting: Stochastic gradient descent
- Function: 0/1 loss with linear function
- Data: Update using a single example at a time

Perceptron Mark 1 (1960)

Why Perceptron?

- The first learning algorithm
- A way of thinking about data: Geometric interpretation of data
- A way of using data: online learning algorithms

A way of thinking about data

How We Represent Data

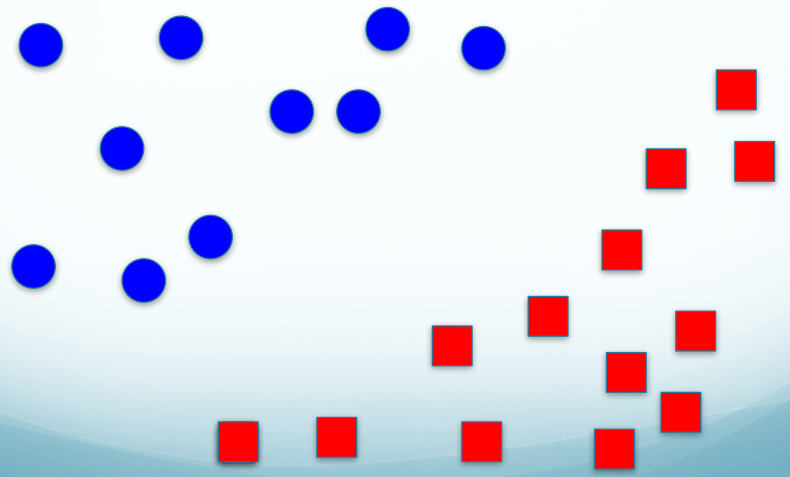
$$\{(x_i, y_i)\}_{i=1}^{N} \quad x_i \in \Re^M \quad y_i \in L$$

- A convenient way of representing data
- Also: a convenient way of *thinking about data*

Geometric Representations

- Each example $x_i \in \Re^M$ represents a point in an M dimensional space
- Classification: divides space into 2 parts
- Examples labeled according to where they are
- Linear classification: a linear decision boundary
  - A hyper-plane (flat high dimensional line)

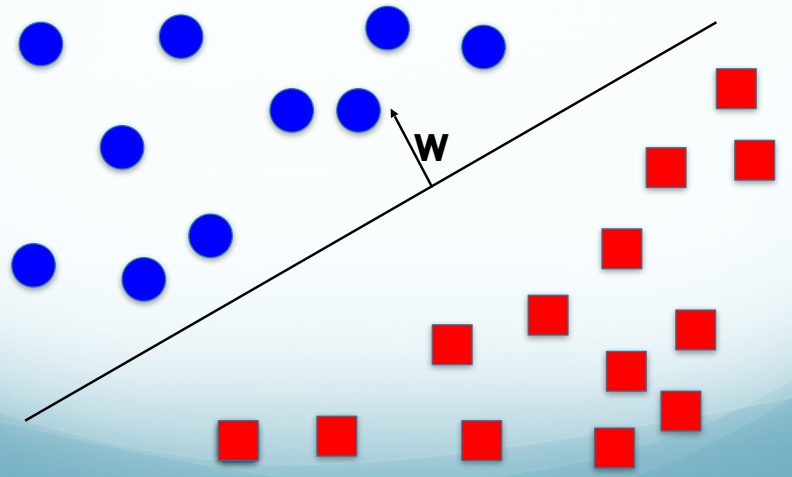
Geometric Representation



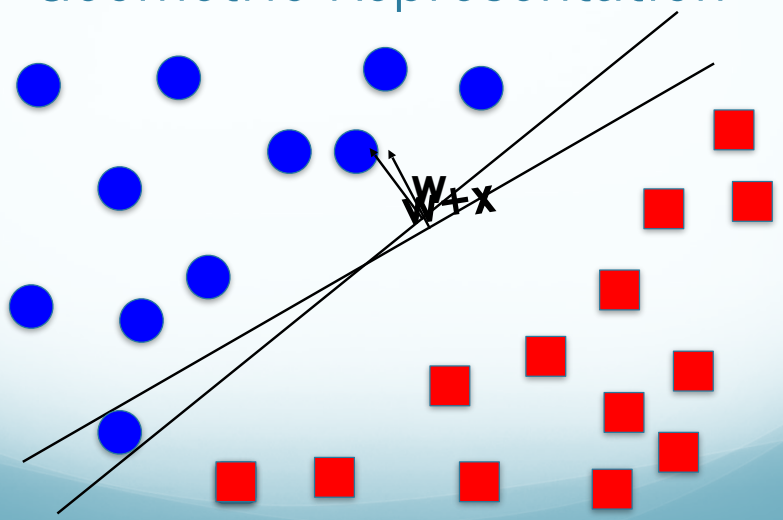
Discriminant Linear Classifiers

- Previously: we forced prediction by thresholding output
- Now: output either 0 or 1 directly
- Classification boundary represented by w
  - w is a vector that is orthogonal (normal) to the decision boundary $\hat{y} = \text{sign}(w \cdot x)$
- Prediction
  - The sign of the prediction indicates which side of the boundary
- Assume decision boundary passes through origin

Geometric Representation



Geometric Representation



Generalized Linear Function

- This is a linear function
  - $w \cdot x$
- Pass the output through a non-linear function
  - $\hat{y} = \text{sign}(w \cdot x)$
- Generalized linear function!
- The output is either +1 (positive) or -1 (negative)

Discriminant Linear Classifiers

- Magnitude of $w \cdot x$ does not matter
  - Relative magnitudes matter (we'll discuss soon)
- There are many representations of the same decision boundary
  - We can scale w without changing prediction

A way of using data

Batch Algorithms

- Train
  - Given training data {X,Y}
  - Learn the parameters of the model
- Test
  - Given an unseen unlabeled example x
  - Assign a label according to learned parameters
- Batch algorithms
  - Takes a batch of examples at once
- Idea: to improve speed, use a stochastic optimization algorithm

Assumptions Behind Batch

- The data is labeled with a consistent hypothesis
  - There is one optimal hypothesis that fits all of the data
    - This hypothesis is not necessarily in our hypothesis class
    - There may be some noise in the labels
- Examples are drawn from a common distribution IID (independent and identically distributed)
  - Identically- each draw is from the same distribution
  - Independent- each draw is independent
  - This allowed us to decompose the data likelihood in Logistic Regression

Removing Assumptions

- No train/test data
  - Instead access to a data stream
- Concept drift
  - No consistency in hypothesis used to label each example
- Examples not IID
  - Data distribution may change
  - Examples may be dependent
- Adversary model
  - An adversary controls the stream

Online Learning Algorithms

- Online learning handles all of these cases
- Make no assumptions about the data
  - Distribution
  - Hypothesis
- Online
  - They interact with the stream or a human
  - Predictions needed after every example
- Do the best you can on each single example

Why Online Learning?

- Few assumptions means widely applicable
- Strong theoretical foundation
- Scales to large amounts of data
  - Streaming metaphor processes examples one at a time
- Updates model without retraining
  - Spam filter: a single new spam email can update the model
- Handles a changing world
  - No assumptions about how data should behave

Online Learning Framework

- On each round
- Receive a single example x
- Predict $\hat{y}$ for x using stored hypothesis
- Receive correct label y
- Suffer loss $\ell(y, \hat{y})$
- Create new hypothesis using current hypothesis, x and y

How to Update?

- We know when to update
- How do we update?
- Change w so it is *less wrong* on the given example
  - Less wrong = has smaller loss
  - A gradient step in the right direction
- We need to answer:
  - What is our model (e.g. linear classifiers)
  - What is our loss function/objective function?

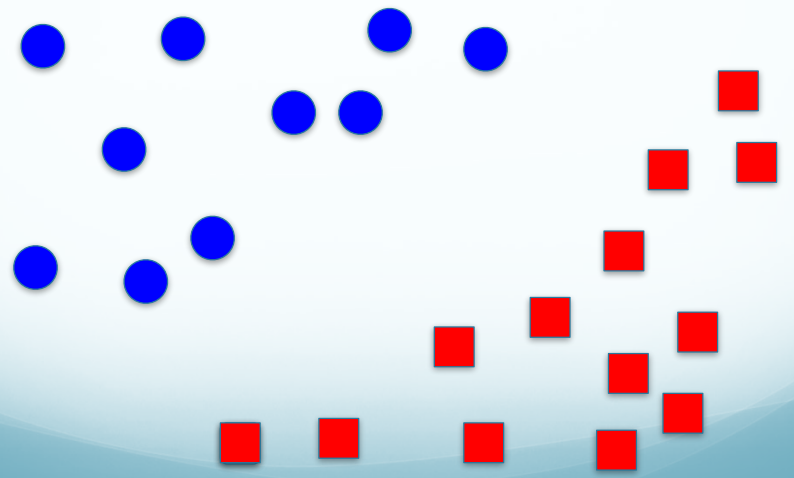
Expected Behavior

- We know that we improve with each update
- What can be said about the expected number of mistakes over a data stream?
  - Is it a lot?
  - A little?
  - Infinite?

Linearly Separable

- Question: Is there a linear boundary that correctly separates all of the examples?
  - Yes: the examples are linearly separable
  - No: the examples are not linearly separable
- This is a separate issue from a consistent or optimal hypothesis
  - Could be not linearly separable and consistent hypothesis

Linearly Separable



Not Linearly Separable

Convergence

- Assume the data are linearly separable
- Will the Perceptron converge?
  - ie. Will it stop making updates?
- Yes! The Perceptron is guaranteed to converge on linearly separable data
- If it converges, then updates stop
  - If updates stop, then it makes a finite number of mistakes

Convergence

- Theorem: if the provided data are linearly separable with margin γ, then Perceptron will terminate in iterations linear with respect to the number of examples
  - Different theoretical frameworks for analyzing online algorithms
    - E.g. mistake bounds
- There are many versions of this theorem and proof
- This is a general version

Not Linearly Separable

- Data not linearly separable then will not converge
- Trivial to make data linearly separable
  - Add a unique feature to each example
  - Not very useful in practice
  - We'll learn better ways to do this

Oscillating Models

- For non-separable data, w will oscillate
- For separable data, it will converge
  - But *which* decision boundary will it converge to?
- Both cases mean w is highly dependent on the order of the data in the stream

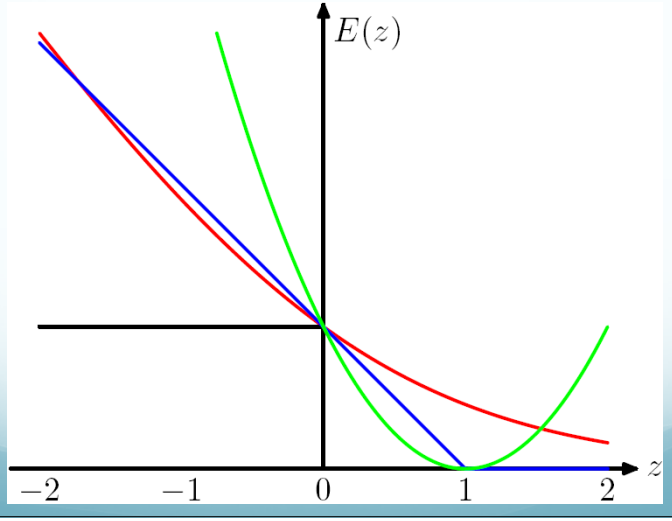
Model Combinations

- Solution: take the best model over time
  - A model that was good on many examples should be trusted more than the latest model
- Voting
  - Save w on each round and predict by voting
  - $\text{sign}([\sum_i \text{sign}(w_i x)])$
- Averaging
  - Take the average of w at each round and average
  - $\text{sign}([\sum_i w_i] x))$

Online Algorithms on the Rise

- Online algorithms are widely applied to large scale data problems
- In practice, many traditional algorithms now have stochastic optimizers
  - Faster for lots of data
  - Better for GPU hardware (mini-batch)

Rethinking Loss Functions



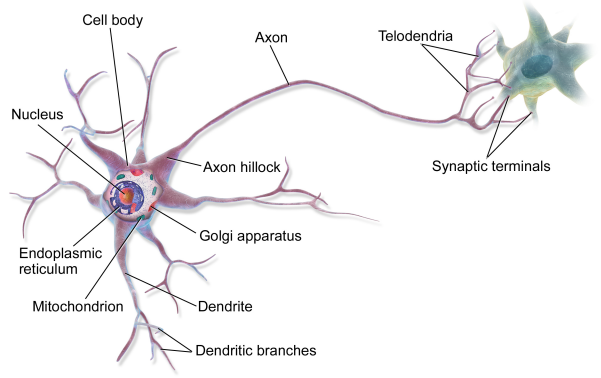
Lingering Questions

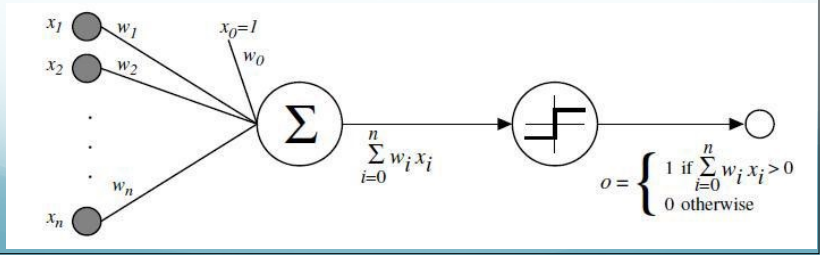
- We have discussed online training of discriminant functions for linear classifiers
  - What would we do if we saw all of the data (batch)?
- Perceptron converges to a separating hyperplane, but there are many
  - Which one is the best?
- Our solution for non-linear data was pretty silly
  - What can we do for non-linear data?

One More Thing: The Neural View

History

- Frank Rosenblatt: Psychologist at Cornell looking for a simple mathematical model of how neurons work in the brain to study how we learn
  - 1957: Starts work on Perceptron
  - 1960: Builds Perceptron Mark 1 machine
  - 1966: Minsky and Papert book show fundamental limitation of Perceptron
    - It's linear





Why a Neural View?

- Coming up in a few weeks: Multi-layer Perceptrons
  - We can stack Perceptrons on top of each other
  - This creates a NON-linear function
  - Overcomes historical limitations
- Stacking of Perceptrons (neurons) creates a neural network

Next Time

Support Vector Machines

Support Vector Machines

Mark Dredze

Machine Learning
CS 600.475

Algorithm: Perceptron

- Initialize w and η
- On each round
  - Receive example x
  - Predict $\hat{y} = \text{sign}(w \cdot x)$
  - Receive correct label $y \in \{+1, -1\}$
  - Suffer loss $\ell_{0/1}(y, \hat{y})$
  - Update w: $w^{j+1} = w^j + \eta\, y_i\, x_i$

Perceptron

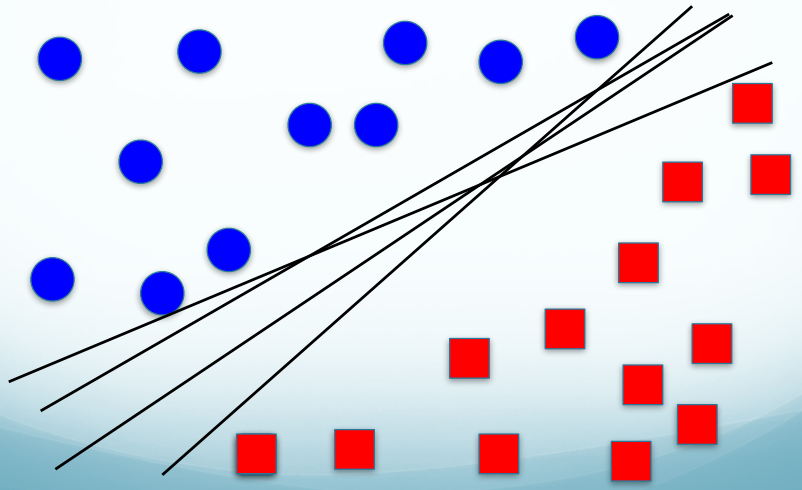
Fitting a function to data

- Fitting: Stochastic gradient descent
- Function: 0/1 loss with linear function
- Data: Update using a single example at a time

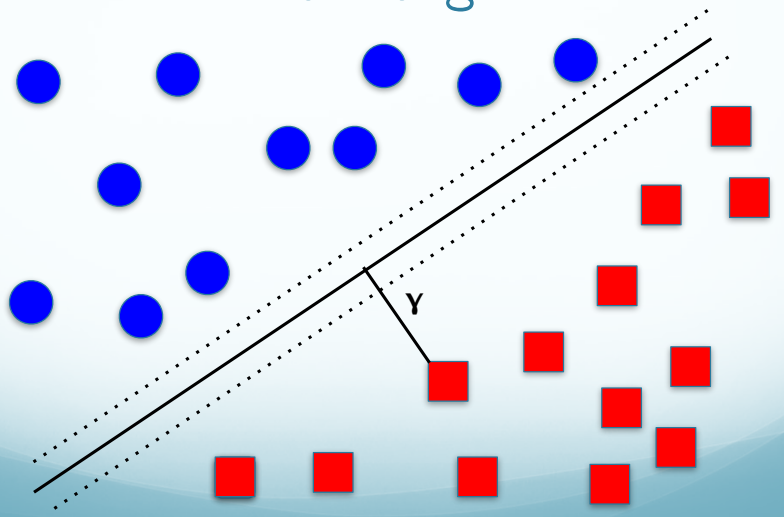
Questions

- Perceptron picks one separating hyperplane (of many)
  - What would we do if we saw all of the data (batch)?
  - We'd pick the best separating hyperplane!
- Which separating hyperplane is the best?
  - Let's look at the geometric model
- Better solutions for non-linear data?

Geometric Representation



The Margin



Functional Margin

- Prediction and y should agree to get large margin

$$\hat{\gamma}^i = y_i(w^T x + b)$$

- What if we double w?

$$\hat{\gamma}^i = y_i(2w^T x + 2b)$$

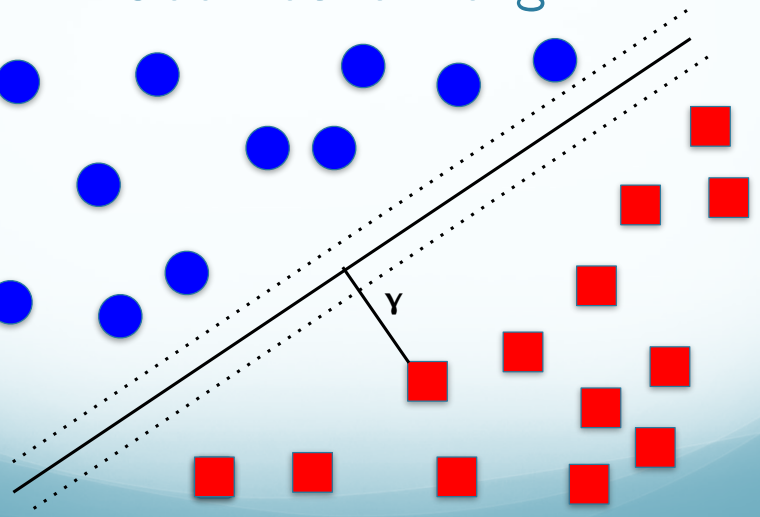
- Doubles margin, but no practical change
  - We will address this in a moment

Functional Margin of Data

- Given a training set of size N:
  - Smallest margin

$$\hat{\gamma} = \min_{i=1,\ldots,N} \hat{\gamma}^i$$

Geometric Margin



Geometric Margin

- Size of γ?
- $\frac{w}{||w||}$ is a unit length vector pointing in the direction of w
- γ intersects with the decision boundary at

$$x_i - \gamma^i \cdot \frac{w}{||w||}$$

and points on the boundary must give a prediction of 0

$$\gamma^i = y_i \left(\left(\frac{w}{||w||} \right)^T x_i + \frac{b}{||w||} \right)$$

if ||w|| = 1 then functional = geometric margin

Max-Margin Principle

- Assuming the observed data is linearly separable
- Select the hyperplane that separates the data with the maximal margin
- Why?
  - New examples are likely to be close to old examples
  - Gives the best generalization error on new data

Maximum Geometric Margin

$$\max_{\gamma,w,b} \quad \gamma$$

$$s.t. \quad y_i(w^T x_i + b) \geq \gamma, i = 1, \ldots, N$$

$$||w|| = 1$$

- Every training instance has margin at least γ
- ||w|| constraint means geometric = functional margin
- Problem: ||w|| constraint is non-convex!

Maximum Geometric Margin

- Functional and geometric related by $\gamma = \frac{\hat{\gamma}}{||w||}$

$$\max_{\hat{\gamma},w,b} \quad \frac{\hat{\gamma}}{||w||}$$

$$s.t. \quad y_i(w^T x_i + b) \geq \hat{\gamma}, i = 1, \ldots, N$$

Maximum Geometric Margin

- Recall: we can arbitrarily scale w!
  - Arbitrarily set $\gamma = 1$

$$\min_{w,b} \quad \frac{1}{2}||w||^2$$

$$\text{s.t.} \quad y_i(w^T x_i + b) \geq 1, i = 1, \ldots, N$$

  - min $||w||^2$ same as max $1/||w||$
- Quadratic program (QP): quadratic objective with linear constraints

Support Vector Machines

Fitting a function to data

- Fitting: Batch optimization method: QP solver
- Function: hyperplane with functional margin >= 1
  - New loss function?
- Data: Train in batch mode

SVM vs. Logistic Regression

- Both minimize the empirical loss with some regularization
  - SVM: $\frac{1}{n}\sum_{i=1}^{n}(1 - y_i[w \cdot x_i])^+ + \lambda\frac{1}{2}\|w\|^2$
  - Logistic: $\frac{1}{n}\sum_{i=1}^{n}\underbrace{-\log g(y_i[w \cdot x_i])}_{-P(y_i|x_i,w)} + \lambda\frac{1}{2}\|w\|^2$
    - $(z)^+$ indicates only positive values
    - $g(z) = (1+\exp(-z))^{-1}$ is the logistic function

Loss Function

- Both minimize

$$\frac{1}{n}\sum_{i=1}^{n}\ell(y_i[w\cdot x_i]) + \lambda\frac{1}{2}\|w\|^2$$

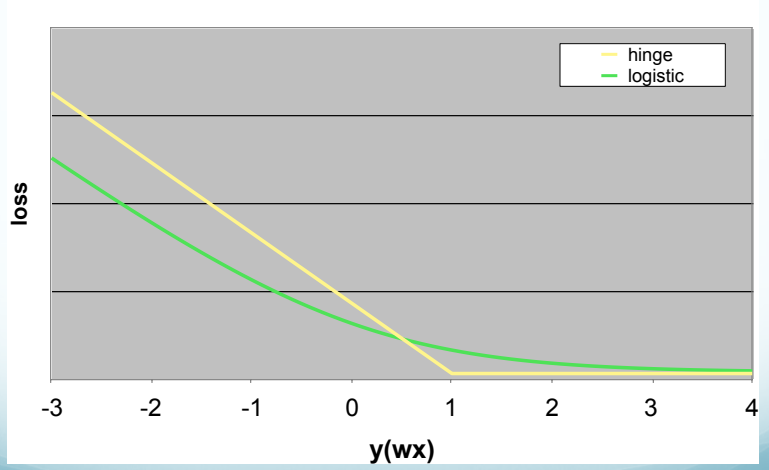
- Different loss functions
- SVM: Hinge Loss

$$\ell(w,x,y) = \max\ (0,\ 1 - y[w\cdot x])$$

- Logistic regression: Logistic loss

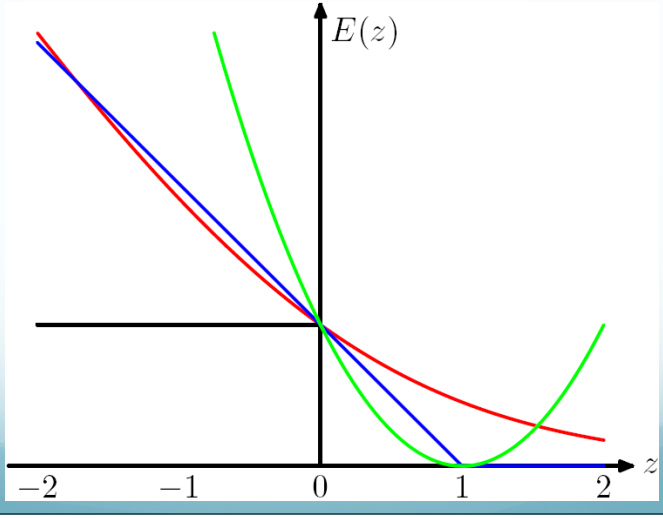
$$\ell(w,x,y) = \log\ (1+\exp\{-y[w\cdot x]\})$$

Loss Function



Thanks to Jason Eisner for figure

Rethinking Loss Functions



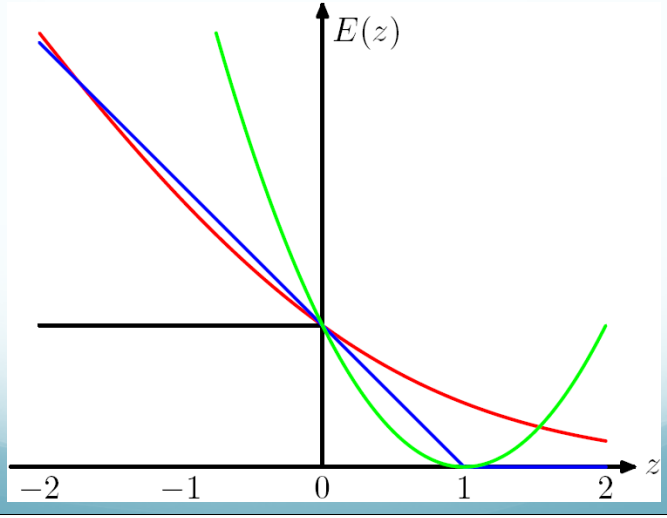
Perceptron: How to Update?

- How do we update w to improve our loss?
- Define an error function based on 0/1 loss

$$L_w(y) = \sum_{i}^{N} max(0, -y_i w \cdot x_i)$$

- What is the difference?

Rethinking Loss Functions



The Perceptron Connection

- SVM minimizes the Perceptron but goes further
- Perceptron gives local updates, SVM gives global updates
- SVM is more aggressive: max-margin principle
- Could we apply max-margin to online learning?
  - Yes! Perceptron with margin
  - Other methods as well

Support Vector Machines

Fitting a function to data

- Fitting: Batch optimization method
- Function: select hyperplane that ensures a fixed margin, L2 regularization
  - Loss: hinge loss
- Data: Train in batch mode

Another Formulation

Dual Formulation

- The primal and dual formulations are complimentary
  - Solving one will give the solution for the other

- Primal problem: objective function is a combination of the m variables
  - Minimize the objective function
  - Solution is a vector of m values that minimize function
- Dual problem: objective function is a combination of n variables
  - Maximize the objective function
  - Solution is a vector of n values called the dual variables

SVM Solution

- Select αs that maximize

$$\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j (x_i x_j^T)$$

  such that $\alpha_i \geq 0$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$

- Predictions for new examples

$$x^T \cdot w = x^T \cdot \sum_{i=1}^{n} [\alpha_i y_i x_i] = \sum_{i=1}^{n} \alpha_i y_i (x^T \cdot x_i)$$

New Approach

Fitting a function to data

- Fitting: Maximize objective in the dual using a QP solver
- Function: max margin linear classifier

$$\hat{y} = \text{sign}(x^T \cdot w) = \text{sign}(\sum_{i=1}^{n} \alpha_i y_i (x^T \cdot x_i))$$

- Data: Train in batch mode

Dual vs. Primal Formulation

- In the primal we have M variables to solve
  - Solve for the vector **w** (length of features)
- In the dual we have N variables to solve
  - Solve for the vector **α** (length of examples)
- When to use the primal?
  - Lots of examples without many features
- When to use the dual?
  - Lots of features without many examples
  - Some other reasons (we'll talk about later)

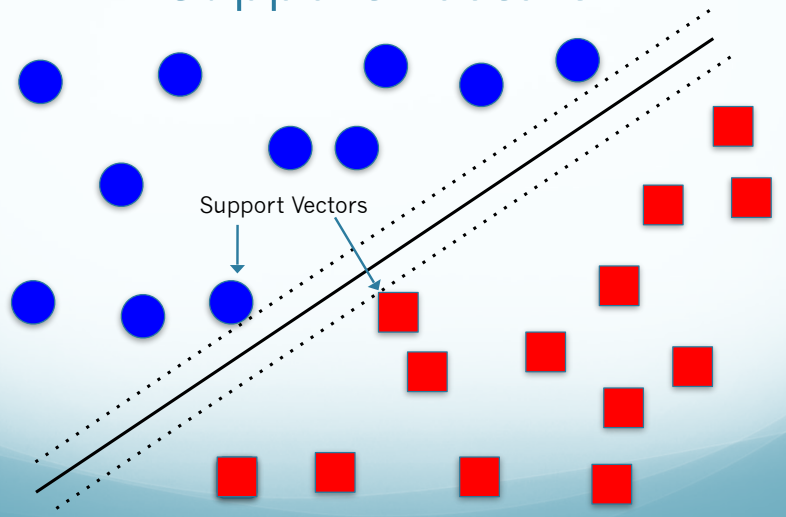
Support Vectors

- Why is it called support vector machine?
- Only some of the αs will be non-zero
  - All misclassified examples will be support vectors

$$\sum_{i=1}^{n} \alpha_i y_i (x^T \cdot x_i)$$

  - Only these vector support the hyperplane
  - These are the vectors closest to the hyperplane
- These are called "support vectors"

Support Vectors



By the Way

- We represented w in terms of the input X
- w is a *linear combination* of the inputs
  - Before: prediction was linear combination of w and x

$$w = \sum_{i=1}^{n} [\alpha_i y_i x_i]$$

- The same is true of Perceptron
  - If we store the support examples

Dual Perceptron

Non-Separable Data

- But not all data is linearly separable
  - Previous solution: add a unique feature to every example to make it separable
- What will SVMs do?
  - The regularization forces the weights to be small
  - But it must still find a max margin solution
  - Result: even with significant regularization, still leads to over-fitting

Slack Variables

$$\min_{w} \ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{such that} \ \ (wx_i)\,y_i + \xi_i \geq 1, \quad \forall i$$

$$\xi_i \geq 0, \quad \forall i$$

- We can always satisfy the margin using ξ
  - We want these ξs to be small
  - Trade off parameter C (similar to λ before)
- ξs are called slack variables
  - The cut the margin some "slack"

Non-Separable Solution

- Similar form to the separable solution
- Extra term added to objective

Bias vs. Variance

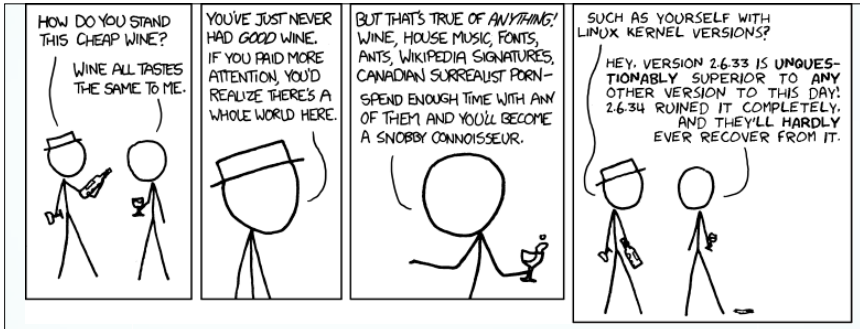
- Smaller C means more slack (larger ξ)
  - More training examples are wrong
  - More bias (less variance) in the output
- Larger C means less slack (smaller ξ)
  - Better fit to the data
  - Less bias (more variance) in the output
- For non-separable data we can't learn a perfect separator so we don't want to try too hard
  - Finding the right balance is a tradeoff

Lingering Questions

- What would we do if we saw all of the data (batch)?
  - We'd pick the best separating hyperplane!
- Which separating hyperplane is the best?
  - The maximum margin separator
  - Use a quadratic regularizer on the weights
- What can we do for non-linear data?
  - It's not separable, use slack variables
  - Can we do better?

Next Time

Kernel Methods and
Non-Linear Support Vector Machines



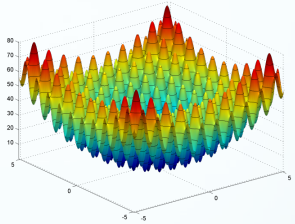
Kernel Methods

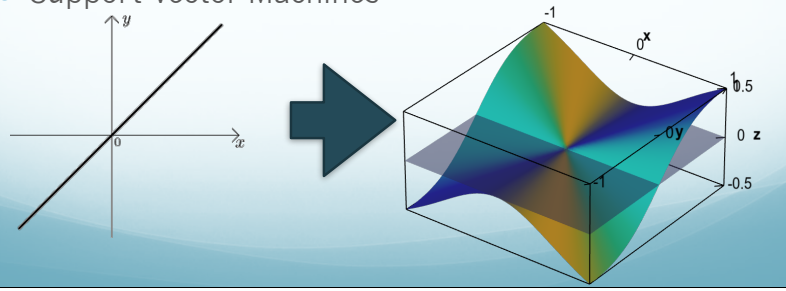
Mark Dredze

Machine Learning
CS 600.475

Linear Classification Methods

- Linear Regression $\hat{y} = \text{sign}(w \cdot x)$
- Logistic Regression
- Perceptron
- Support Vector Machines

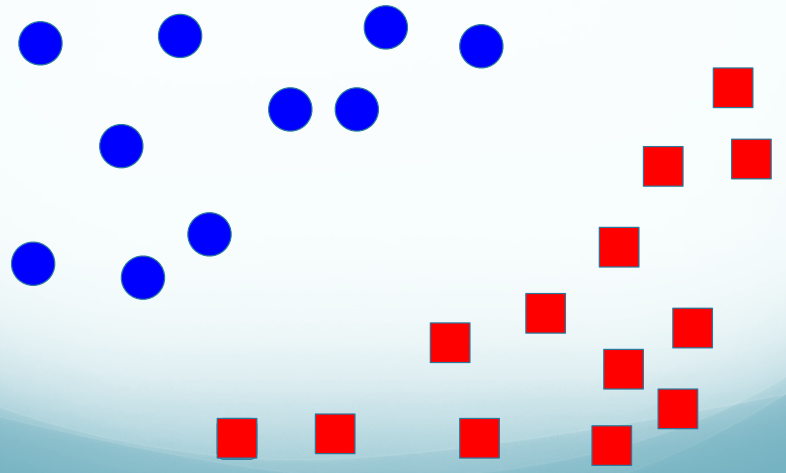




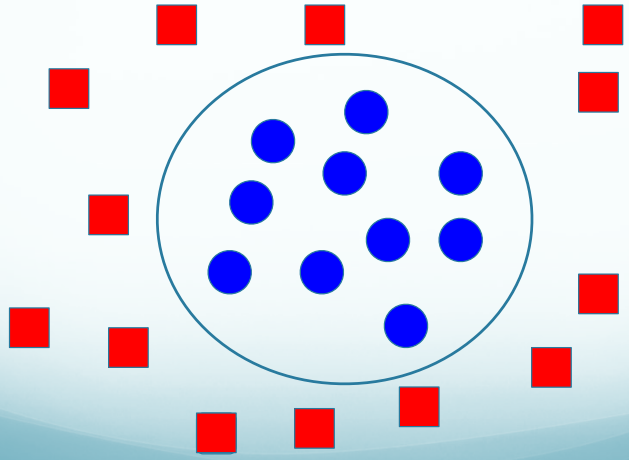
Review: Lingering Questions

- What would we do if we saw all of the data (batch)?
  - We'd pick the best separating hyperplane!
- Which separating hyperplane is the best?
  - The maximum margin separator
  - Use a quadratic regularizer on the weights
- What can we do for non-linear data?
  - It's not separable, use slack variables
  - Can we do better?

Linearly Separable



Not Linearly Separable



Handling Non-Linear Data

- Option 1: Add features by hand that make the data separable
  - Requires feature engineering
- Option 2: Learn a small number of additional features that will suffice
  - We'll see this eventually
- Option 3: Kernel trick
  - Today

Feature Mapping Functions

- Assuming a two dimensional vector x = [x(1),x(2)]
  - x(i) is the ith position of x
- Let's apply a feature mapping function
- Why is this useful?

$$\phi([x_{(1)}, x_{(2)}]) = (x_{(1)}^2, \sqrt{2} \cdot x_{(1)} x_{(2)}, x_{(2)}^2)$$

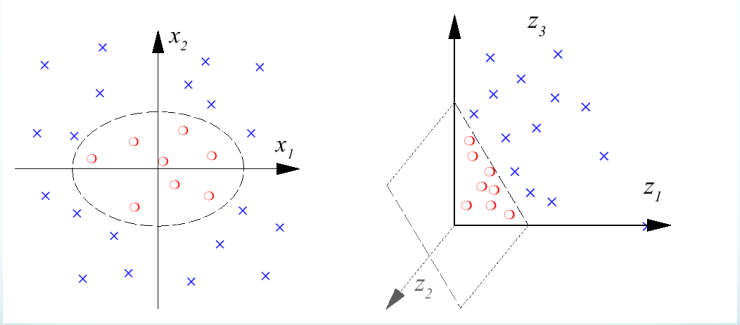
  - Elliptical decision boundary:
  - Not linear in x, but linear in Φ(x)

$$x_{(1)}^2 + 2x_{(2)}^2 < 3$$

  - Boundaries defined by linear combinations of x2, y2, xy, x, and y are ellipses, parabolas, and hyperbolas in the original space.

Geometric Interpretation

$$\phi([x_{(1)}, x_{(2)}]) = (x_{(1)}^2, \sqrt{2} \cdot x_{(1)} x_{(2)}, x_{(2)}^2)$$



$$x_{(1)}^2 + 2x_{(2)}^2 < 3$$

Non-linear in x

$$\phi(x)_{(1)}^2 + 2\phi(x)_{(2)}^2 < 3$$

Linear in ϕ(x)

Why Feature Mapping Functions?

- Recall that to make something linearly separable I can just add a unique feature to every example
- Any dataset is linearly separable if we use enough dimensions
  - In an n-dimensional space, almost any set of up to n +1 labeled points is linearly separable!
- We can obtain linear separability by projecting data into higher dimensional spaces
  - Use smarter techniques to obtain generalizeable separability

Feature Functions + SVM

- Replace x with a feature mapping function

$$\arg\min_{w} ||w||_2^2$$
$$s.t. \quad y_i(w \cdot \phi(x_i)) \geq 1 \quad \forall i$$

- The dot product is now taken over a higher dimensional feature space
  - If ϕ is quadratic then the feature space is a quadratic space in terms of the inputs

Limitations

- We still have to learn w
  - w will grow in size of the feature space
  - e.g. quadratic kernel: |x| = 100 → |phi(x)| = 10000
- Feature functions just increase the feature space in a non-linear way
- Too limiting

SVMs and w

- Wait a minute, there is no w!

$$\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j (\phi(x_i)\phi(x_j^T))$$

- There is no modeling constraint that prevents us from making $\phi(x)$ very large
- αs do not grow in the size of $\phi(x)$
- Thank you dual!

Kernels

- Let's replace $\phi(x_i)\phi(x_j^T)$ with a kernel function K

$$\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j K(x_i, x_j)$$

- where

$$x^T \cdot w = x^T \cdot \sum_{i=1}^{n}[\alpha_i y_i x_i] = \sum_{i=1}^{n}\alpha_i y_i K(x, x_i)$$

$$K(x, x') = (\phi(x)\phi(x')^T)$$

Why?

- We have removed all dependencies in the SVM on the size of the feature space
  - The feature space $\phi(x)$ appears only in the kernel
- As long as the Kernel function does the work, we can handle any feature space

Intuition About Over-Fitting

- Wait a minute!
- Assuming we project features then even using the simple projection shown so far, we'd have way to many features!
- Didn't we learn that too many features means over-fitting?

Saved by the Dual

- We aren't free to choose a parameter for each feature
- w is a linear combination of the inputs
  - We can only choose the parameters for αs
  - There are only n αs, no matter how large our feature space projection
- The inputs x put a constraint on our flexibility in high dimensional space

The Kernel Trick

- Take a linear SVM
- Substitute a non-linear kernel
- Optimize objective in the dual
- We get non-linear classification!
- Without
  - Over-fitting
  - Learning too many parameters
  - Computing a large feature space

What is a Kernel?

- A kernel is a scalar product between two high dimensional feature vectors
  - $K(x,x') = (\phi(x)\phi(x')^T)$
- A proposed kernel function can be written in this form
- We can define any mapping function and then compute the kernel

Quadractic Kernel

- Let's take the cross product of all features (quadratic)
  - $K(x,x')=(x\ x')^2$
- Why is the quadratic a valid kernel?
  - It's actually just a scalar product of the two vectors

$$K(x,x') = (x \cdot x')^2$$
$$= (x_1 x'_1 + x_2 x'_2)^2$$
$$= (x_1^2 x'^2_1 + x_2^2 x'^2_2 + 2 x_1 x_2 x'_1 x'_2)$$
$$= (x_1^2, x_2^2, \sqrt{2} x_1 x_2) \cdot (x'^2_1, x'^2_2, \sqrt{2} x'_1 x'_2)$$
$$= \Phi(x) \cdot \Phi(x')$$

- $\phi(x)$ is the basis function used for the ellipse example
  - This is true for arbitrary dimensions of x

Polynomial Kernel

- In fact, this is true of any exponent p

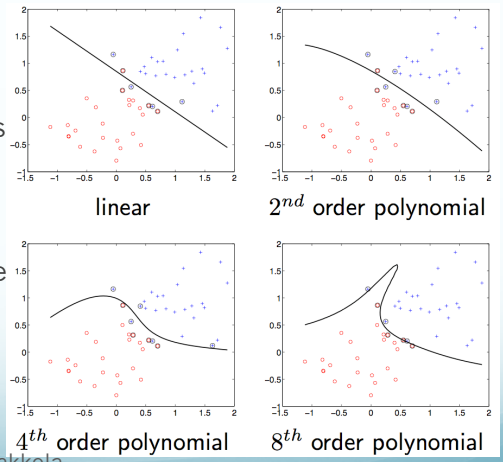
$$K(x,x') = (1 + (x^T x'))^P$$

- This is the polynomial kernel
  - To get the feature vectors we would concatenate all elements up to the *p*th order polynomial terms of the components of x (weighted appropriately)

- http://www.youtube.com/watch?v=3IiCbRZPrZA

Polynomial Kernel

- In the given feature space the separators are non-linear
- In the high dimensional space they are linear
- Support vectors are circled



Pictures by Tommi Jaakkola

Decision Boundary

- How does the kernel influence the decision boundary?
- Recall prediction given by

$$x^T \cdot w = \sum_{i=1}^{n} \alpha_i y_i K(x, x_i)$$

- The larger $K(x, x_i)$ the more x_i contributes to the decision for x
  - x receives a label based on those support vectors (examples with large α) with highest $K(x, x_i)$

Similarity Function?

- Does that mean $K(x, x_i)$ is a similarity function?
  - Give same label as most similar examples
- Sort of
  - Recall: $\cos\theta = \frac{x \cdot x'}{\|x\|\|x'\|}$
  - Therefore $x \cdot x' = \|x\|\|x'\|\cos\theta$
  - So $\alpha K(x, x') = \alpha\phi(x) \cdot \phi(x') = \alpha\|\phi(x)\|\|\phi(x')\|\cos\theta$

Similarity Function?

$$\alpha K(x, x') = \alpha\phi(x) \cdot \phi(x') = \alpha\|\phi(x)\|\|\phi(x')\|\cos\theta$$

- Note
  - $\phi(x)$: constant across all x' in the prediction
  - $\alpha\phi(x')$: α is scaled per x' so this just weighs importance
  - cos θ: the angle between the vectors
    - When θ is 0, this is 1 so larger values for more similar vectors

Kernel Definitions

- A scalar product of two vectors in high dimensional space
- OR
- Mercer's theorem

Mercer's Theorem

- Suppose K is a valid kernel
- Define Kernel matrix (Gram matrix) as

$$K_{ij} = K(x_i, x_j)$$

- K must be symmetric

$$K_{ij} = K(x_i, x_j) = \phi(x_i)^T\phi(x_j) = \phi(x_j)^T\phi(x_i) = K(x_j, x_i) = K_{ji}$$

Mercer's Theorem

- $\phi_k(x)$ kth position of vector

$$\begin{aligned} z^T K z &= \textstyle\sum_i \sum_j z_i K_{ij} z_j \\ &= \textstyle\sum_i \sum_j z_i \phi(x_i)^T \phi(x_j) z_j \\ &= \textstyle\sum_i \sum_j z_i \sum_k \phi_k(x_i)\phi_k(x_j) z_j \\ &= \textstyle\sum_k \sum_i \sum_j z_i \phi_k(x_i)\phi_k(x_j) z_j \\ &= \textstyle\sum_k \left(\sum_i z_i \phi_k(x_i)\right)^2 \\ &\geq 0 \end{aligned}$$

Mercer's Theorem

- Let K : R^N x R^N -> R be given. Then for K to be a valid (Mercer) kernel, it is necessary and sufficient that for any finite data set, the corresponding kernel matrix is symmetric positive semi-definite.

Kernel Definitions

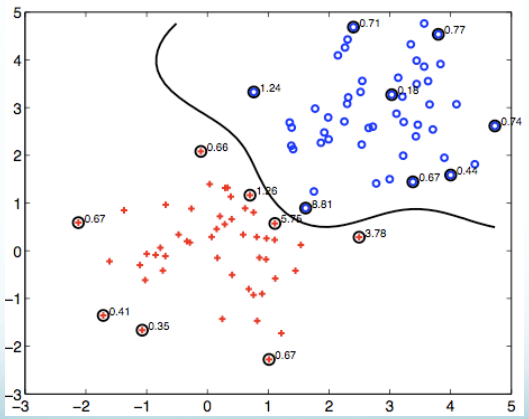
- A kernel is
  - A scalar product of two vectors in high dimensional space
  - Mercer's theorem
- How do we test a kernel without writing $\phi(x)$ explicitly?
- Equivalent definition
  - The Gram matrix **K** should be positive semidefinite for all x
    - Gram matrix **K** : $\mathbf{K}_{ij} = K(x_i, x_j)$
    - Positive semidefinite: $x^T \mathbf{M} x \geq 0$

Example of a Kernel

- Polynomial kernel $K(x, x') = (1 + (x^T x'))^P$
- Radial Basis Function (RBF) kernel
  - Gaussian version
  - Infinite dimensional function

$$K(x, x') = \exp\left(-\frac{1}{2}\|x - x'\|^2\right)$$

Radial Basis Function



Pictures by Tommi Jaakkola

Building Kernels

- How do we build a kernel?
  - Decide on a projection that is meaningful for data
- How do we know something is valid?
  - Show it's a scalar product
  - Show positive semidefinite kernel matrix
  - Best: compose new kernels from old kernels

Kernel Operations

- Many operations over kernels yield new kernels

$$K(x,x') = cK_1(x,x')$$
$$K(x,x') = f(x)K_1(x,x')f(x')$$
$$K(x,x') = \exp(K_1(x,x'))$$
$$K(x,x') = K_1(x,x') + K_2(x,x')$$
$$K(x,x') = K_1(x,x')K_2(x,x')$$

  - More examples in the book

Gaussian Kernel

- Use a Gaussian to define a kernel
  - Since this is not a probability drop the normalization

$$K(x,x') = \exp(-\|x-x'\|^2/2\sigma^2)$$

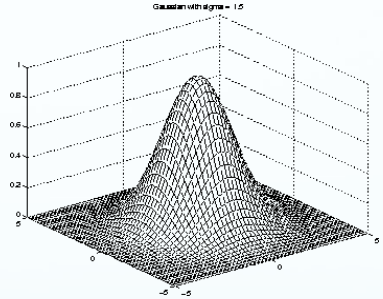
  - Why is this a valid kernel?
  - Expand the square

$$\|x-x'\|^2 = x^Tx + x'^Tx' - 2x^Tx'$$

  - Substitute

$$K(x,x') = \exp(-x^Tx/2\sigma^2)\exp(-x'^Tx'/2\sigma^2)\exp(2x^Tx'/2\sigma^2)$$
$$K(x,x') = f(x)K_1(x,x')f(x')$$
$$K(x,x') = \exp(K_1(x,x'))$$

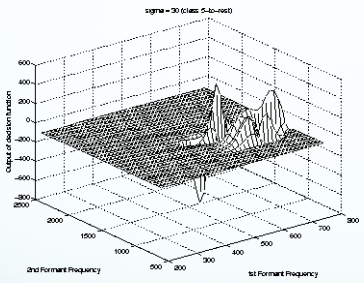
Gaussian Kernel



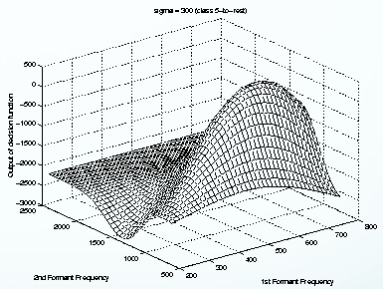
- Three dimensional Gaussian
  - σ determines the smoothness of the function
  - Large σ means the support vector has greater influence area
    - Less support vectors needed to cover boundaries

http://mi.eng.cam.ac.uk/~kai214/thesis_main/node31.html

Decision Boundary



Smaller σ



Larger σ

Kernels for Objects

- We've talked about kernels as operating over $x \in \Re^M$
- However, we can define x as anything
  - As long as we can compute $K(x, x')$
- Kernels for
  - Strings
  - Trees/Graphs
  - Images

Kernels for Strings

- Represent a document as a feature vector
  - Each feature corresponds to a word in the document
  - Classify document based on the words
- Even better: each feature corresponds to a sub-string in the document
  - Include non-contiguous sub-strings
  - Value of feature is dependent on frequency of where it appears
- For sub-strings of size > 4 cannot compute this feature space
  - Way too many features!

Kernels for Strings

- String Subsequence Kernel

$$K(x, x') = \sum_{u \in \Sigma^d} \sum_{i: u = x[i]} \sum_{j: u = x'[j]} \lambda^{|i|+|j|}$$

  - For all string u of length d
  - For all substrings of x
  - For all substrings of x'
  - λ to the power of the size of the combined lengths
- Computing features would take $O(|\Sigma|^d)$ time
- Can compute the kernel for this feature representation using dynamic programming

Lodhi, et al. 2002

Biology: Splice Site Recognition

- Find the boundary between exons and introns in eukaryotes (complex organism)
  - What part of DNA codes for genes
- Input is a sequence of DNA base pairs
- Normally each feature indicates a substring of base pairs appearing in the sequence

Biology: Splice Site Recognition

- Each possible substring of DNA is a new feature
- Use the kernel approach as for strings
- Problem for DNA: long substrings unlikely but still informative
- Solution: a kernel from many weighted spectrum kernels

$$K_\ell(x,x') = \sum_{d=1}^{\ell} \beta_d K_d^{spectrum}(x,x')$$

Other Kernel Methods

Everything Can be Non-Linear

Kernel Perceptron

- We showed a derivation for dual Perceptron

$$\hat{y} = sign(\sum_{i=1}^{n} \alpha_i y_i x_i x)$$

  - α_i=1 if we made a mistake on round i
- Replace the dot product with a kernel

$$\hat{y} = sign(\sum_{i=1}^{n} \alpha_i y_i K(x_i, x))$$

Kernel Linear Regression

- We can define linear regression with quadratic regularization using a linear combination of x

$$w = -\frac{1}{\lambda}\sum_{i=1}^{N}\{w^T\phi(x_i) - y_i\}\phi(x_i)$$

$$= \sum_{i=1}^{N}\alpha_i\phi(x_i)$$

- So we can get a kernel version

$$w^T\phi(x) = k(x)^T(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{Y}$$

$$k_i(x)^T = K(x_i, x)$$

Kernel Logistic Regression

- We can do the same trick with logistic regression
- Represent w in terms of x and α

$$w = \sum_{i=1}^{N}\alpha_i\phi(x_i)$$

- Insert a kernel in place of a dot product in the model

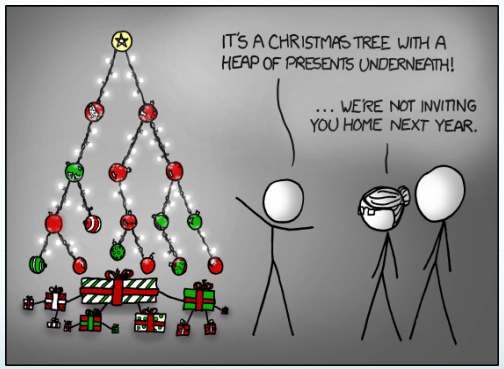
$$P(y=1 \mid x, w) = \frac{1}{1+\exp\{-(\sum_{i=1}^{n}\alpha_i K(x, x_i) + b)\}}$$

- Derive new gradient descent rule on α

Summary

- The good
  - Arbitrarily high dimensionality
  - Extensions to other data types
  - Non-linearity in a parametric linear framework
- The bad
  - What is a good kernel?
    - Whole field on designing kernels, learning kernels
  - Cannot handle large data
    - Kernel matrix grows quadratic in N

Decision Trees



Mark Dredze

Machine Learning
CS 600.475

Decision Trees

- Decision trees have a long history in ML
  - First popular algorithms 1979
- Very popular in many real world settings
- Intuitive to understand
- Easy to build

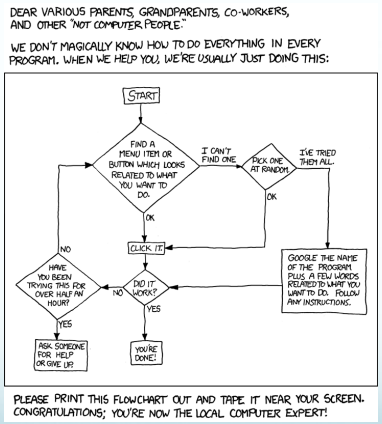
History

- EPAM- Elementary Perceiver and Memorizer
  - Feigenbaum 1961
  - Cognitive simulation model of human concept learning
- CLS- Early algorithm for decision tree construction
  - Hunt 1966
- ID3 based on information theory
  - Quinlan 1979
- C4.5 improved over ID3
  - Quinlan 1993
- Also has history in statistics as CART (Classification and regression tree)

Motivation

- How do people make decisions?
  - Consider a variety of factors
  - Follow a logical path of checks
- Should I eat at this restaurant?
  - If there is no wait
    - Yes
  - If there is short wait and I am hungry
    - Yes
  - Else
    - No

Decision Graph Example



http://www.xkcd.com/627/

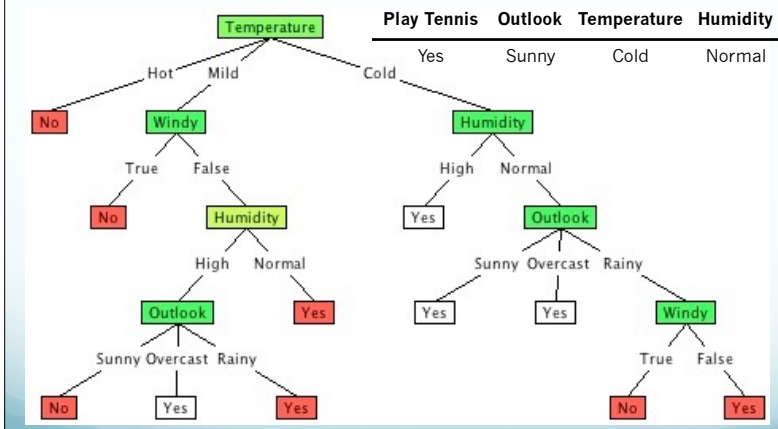
Example: Should We Play Tennis?

Play Tennis	Outlook	Temperature	Humidity	Windy
No	Sunny	Hot	High	No
No	Sunny	Hot	High	Yes
Yes	Overcast	Hot	High	No
Yes	Rainy	Mild	High	No
Yes	Rainy	Cold	Normal	No

y *x*

- If temperature is not hot
  - Play tennis
- If outlook is overcast
  - Play tennis
- Otherwise
  - Don't play tennis

Decision Tree



Play Tennis	Outlook	Temperature	Humidity	Windy
Yes	Sunny	Cold	Normal	No

Decision Trees

- A decision tree is formed of
  - Nodes
    - Attribute tests
  - Branches
    - Results of attribute tests
  - Leaves
    - Classifications

Hypothesis Class

- What functions can decision trees model?
  - Non-linear: very powerful hypothesis class
  - A decision tree can encode *any Boolean function*
  - Proof
    - Given a truth table for a function
    - Construct a path in the tree for each row of the table
    - Given a row as input, follow that path to the desired leaf (output)

Y	X_1	X_2	X_3
1	0	0	0
0	0	0	1
1	0	1	0

- Problem: exponentially large trees!

Smaller Trees

- Can we produce smaller decision trees for functions?
  - Yes (most of the time)
  - Counter examples
    - Parity function
      - Return 1 on even inputs, 0 on odd inputs
    - Majority function
      - Return 1 if more than half of inputs are 1
- Decision trees are good for some functions but bad for others
- Recall: tradeoff between hypothesis class expressiveness and learnability

Decision Trees

Fitting a function to data

- Fitting: ???
- Function: any boolean function
- Data: Batch: construct a tree using all the data

Building Decision Trees

What Makes a Good Tree?

- Small
  - Ockham's razor
    - Simpler is better
  - Avoids over-fitting
    - We'll discuss this again later
- A decision tree may be human readable, but not use human logic
  - The decision tree you would write for a problem may differ from computer

Small Trees

- How do we build small trees that accurately capture data?
- Optimal decision tree learning is NP-complete

- Constructing Optimal Binary Decision Trees is NP-complete. Laurent Hyafil, RL Rivest. Information Processing Letters, Vol. 5, No. 1. (1976), pp. 15-17.

Greedy Algorithms

- Like many NP-complete problems we can get pretty good solutions
- Most decision tree learning is by greedy algorithms
  - Adjustments are usually to fix greedy selection problems
- Top down decision tree learning
  - Recursive algorithms

ID3

- function BuildDecisionTree(data, labels):
  - if all labels are the same
    - return leaf node for that label
  - else
    - let f be the best feature for splitting
    - left = BuildDecisionTree(data with f=0, labels with f=0)
    - right= BuildDecisionTree(data with f=1, labels with f=1)
    - return Tree(f, left, right)


Does this always terminate?

Base Cases

- All data have same label
  - Return that label
- No examples
  - Return majority label of all data
- No further splits possible
  - Return majority label of passed data

ID3

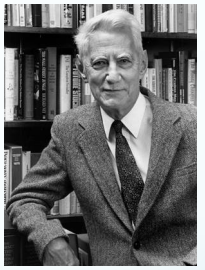
- function BuildDecisionTree(data, labels):
  - if all labels are the same
    - return leaf node for that label
  - else
    - let f be the best feature for splitting
    - left = BuildDecisionTree(data with f=0, labels with f=0)
    - right= BuildDecisionTree(data with f=1, labels with f=1)
    - return Tree(f, left, right)

Selecting Features

- The best feature for splitting
  - The most *informative feature*
  - Select the feature that is most informative about the labels
- Information theory

Information Theory

- The quantification of information
- Founded by Claude Shannon
  - Landmark paper in 1948
  - Noisy channel theorem



Information Theory

- A brief introduction...

Information Theory

- Entropy $H(X) = -\sum_x p(X = x) \log p(X = x)$
- Conditional Entropy $H(Y|X) = \sum_x p(X = x) H(Y|x = x)$
- Information Gain $IG(Y|X) = H(Y) - H(Y|X)$

Selecting Features

- The best feature for splitting
  - The most *informative feature*
  - The feature with the highest *information gain*

Notes for Decision Trees

- Since we compare H(Y|X) across all features, H(Y) is a constant
  - We can omit it for comparisons
- The base of the log doesn't matter as long as it is consistent

Example: Should We Play Tennis?

Play Tennis	Outlook	Temperature	Humidity	Windy
No	Sunny	Hot	High	No
No	Sunny	Hot	High	Yes
Yes	Overcast	Hot	High	No
Yes	Rainy	Mild	High	No
Yes	Rainy	Cold	Normal	No

- H(Tennis) = $-3/5 \log_2 3/5 - 2/5 \log_2 2/5 = 0.97$

Example: Should We Play Tennis?

Play Tennis	Outlook	Temperature	Humidity	Windy
No	Sunny	Hot	High	No
No	Sunny	Hot	High	Yes
Yes	Overcast	Hot	High	No
Yes	Rainy	Mild	High	No
Yes	Rainy	Cold	Normal	No

- H(Tennis|Outlook=Sunny) = $-2/2 \log_2 2/2 - 0/2 \log_2 0/2 = 0$
- H(Tennis|Outlook=Overcast) = $-0/1 \log_2 0/1 - 1/1 \log_2 1/1 = 0$
- H(Tennis|Outlook=Rainy) = $-0/2 \log_2 0/2 - 2/2 \log_2 2/2 = 0$
- H(Tennis|Outlook) = 2/5 * 0 + 1/5 * 0 + 2/5 * 0 = 0

Example: Should We Play Tennis?

Play Tennis	Outlook	Temperature	Humidity	Windy
No	Sunny	Hot	High	No
No	Sunny	Hot	High	Yes
Yes	Overcast	Hot	High	No
Yes	Rainy	Mild	High	No
Yes	Rainy	Cold	Normal	No

- IG(Tennis|Outlook) = 0.97 – 0 = 0.97
- If we knew the Outlook we'd be able to perfectly predict Tennis!
- Outlook is a great feature to pick for our decision tree

ID3

- function BuildDecisionTree(data, labels):
  - if basecase
    - return appropriate leaf node
  - Else
    - f = arg max IG(label|f)
    - left = BuildDecisionTree(data with f=0, labels with f=0)
    - right= BuildDecisionTree(data with f=1, labels with f=1)
    - return Tree(f, left, right)

Base Cases

- All data have same label
  - Return that label
- No examples
  - Return majority label of all data
- No further splits possible
  - Return majority label of passed data
- If max IG = 0?

IG=0 As a Base Case

- Consider the following

Y	X_1	X_2
0	0	0
1	0	1
1	1	0
0	1	1

- Both features give 0 IG
- Once we divide the data, perfect classification!

Training vs. Test Accuracy

- On the tree shown for tennis
- 100% training accuracy
- 30% testing accuracy
- Why?

Over-fitting

- X_5 perfectly predicts Y
- Let's randomly flip Y with probability ¼
- X_5 will be the first split
- But tree will keep going
  - Duplicate training data into train/test
  - Train accuracy will be 100%
  - Test accuracy will be 62.5% (5/8)
    - 1/16 examples are doubly corrupted
    - 9/16 are uncorrupted
    - 6/16 will be bad
  - Single node test accuracy: 75%

Y	X_1	X_2	X_3	X_4	X_5
0	0	0	0	0	0
1	0	0	0	0	1
0	0	0	0	1	0
1	0	0	0	1	1
0	0	0	1	0	0
1	0	0	1	0	1
0	0	0	1	1	0
1	0	0	1	1	1
…	32 examples total				

Bias/Variance Tradeoff

- Complete trees have no bias
  - But can over-fit badly
  - Lots of variance
- 0 depth trees (return most likely label) have no variance
  - Totally biased towards majority label
- A good tree balances between these two
  - How do we learn balanced trees?

Pruning: New Base Cases

- Stop when too few examples
- Stop when max depth reached
  - You will use this for HW
- Stop when my classification error is not much more than average of my children
- X^2 pruning- stop when remainder is no more likely than chance

Parameters

- All of these are parameters
- How do you select parameter values?
  - Train data?
  - Test data?
  - Development data!

Extensions

- Non-binary attributes
  - Categorical
  - Continuous (real valued)
    - In homework
    - Handle by thresholding- find the best single threshold to split the range
    - Regression trees
- Missing attributes
- Alternatives to information gain: Gini index, miss-classification rate
- Non-greedy algorithms?

Decision Trees

Fitting a function to data

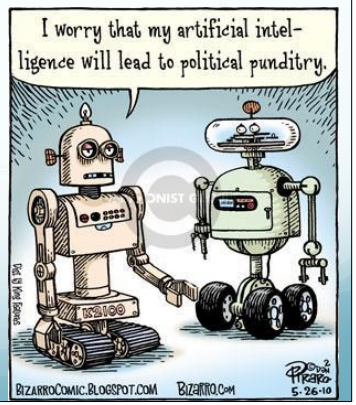
- Fitting: greedy algorithm to find a good tree
  - Extra heuristics to help with over-fitting
  - Optimal decision tree learning: NP-complete
- Function: any boolean function
- Data: Batch: construct a tree using all the data

Boosting

Mark Dredze

Machine Learning
CS 600.475

Based on tutorial by Freund and Schapire



Different Approach

- So far we've covered
  - Supervised Learning (Classification)
    - Linear and non-linear models
- Today's idea: combine the outputs of classifiers to get something better
  - Approach: build complex models from simpler models

Combinations

- Can I learn many different classifier using a supervised learning algorithm and combine them to get a better classifier?
  - Yes
- Ensemble learning
  - Combine an ensemble of classifiers

Learning

- weak learner
  - learns a predictor slightly better than random guessing
- strong learner
  - learns a predictor with arbitrary accuracy

Boosting

- Let's say I have a weak learner, can I take this weak learner and make it better?
  - Use sets of weak learners
- Question first proposed in 1988:
  - "Does weak learnability imply strong learnability?"
    - Asked about PAC learning, Kearns and Valiant, 1988
- Answer came in 1989 by Schapire
  - Yes! "boost" a weak learner to get a strong learner!

Some Boosting History

- Schapire 1989
  - First boosting algorithm
  - Show slight improvements in theory
- Freund 1990
  - An optimal algorithm that boosts by majority
- Drucker, Schapire and Simard 1992
  - First experiments using boosting
  - Limited by practical considerations
- Freund and Schapire 1996
  - AdaBoost- the first practical boosting algorithm
    - Cited 7000+ times on Google Scholar
  - The rest is history

General Boosting Approach

- Look at subset of data
  - Make simple rule to classify data (weak)
  - Repeat (make many rules)
- Boosting
  - Boost weak rules into strong predictors

A Formal View of Boosting

- Given training set $\{x_i, y_i\}_{i=1}^{N}$ and weak learner *f*
- Binary labels $y_i \in \{-1, +1\}$
- For each boosting iteration
  - Construct distribution D_t on N examples
  - Learn weak hypothesis h_t using *f* with error:

$$\varepsilon_t = P_{D_t}[h_t(x_i) \neq y_i]$$

- Output final hypothesis

Questions

- How do we choose subsets of the data?
- How do we combine all the rules into predictor?
- The answer: AdaBoost

AdaBoost

- Given: $\{x_i, y_i\}_{i=1}^{N}$ where: $y_i \in \{-1, +1\}$
  - Initialize $D_1(i)=1/N$

AdaBoost

- For each iteration t=1 to T:
  - Train weak learner using distribution D_t
  - Get weak hypothesis h_t with error

$$\varepsilon_t = P_{D_t}[h_t(x_i) \neq y_i]$$

  - Choose $\alpha_t = \frac{1}{2}\log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
  - Update

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases}$$

$$= \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

  - Z_t is a normalization constant

AdaBoost

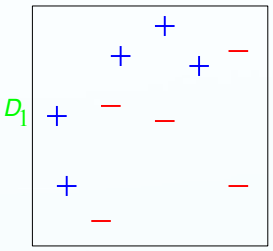
- Output the final hypothesis:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Notes

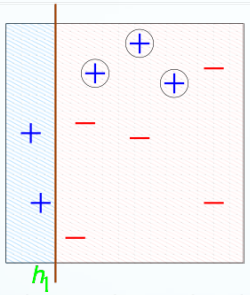
- α_t measures importance assigned to h_t
- As α gets larger, error gets smaller
- Distribution tends to concentrate on hard examples
  - Sound familiar?
    - SVMs!
    - Weight on examples close to the margin
    - We'll come back to this point

Example

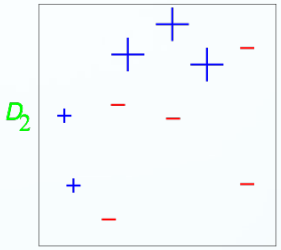


- A set of labeled points with a uniform distribution

Round 1

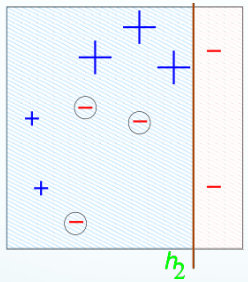


$\varepsilon_1=0.30$
$\alpha_1=0.42$

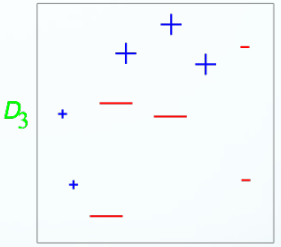


- Learn hypothesis, measure error, set α
- Recompute distribution placing more weight on incorrect examples

Round 2

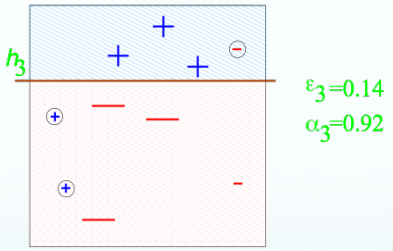


$\varepsilon_2=0.21$
$\alpha_2=0.65$

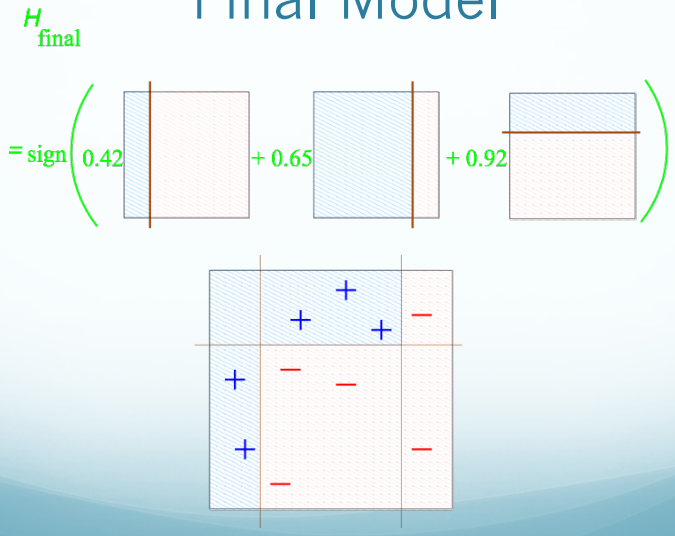


- Learn hypothesis, measure error, set α
- Recompute distribution placing more weight on incorrect examples

Round 3

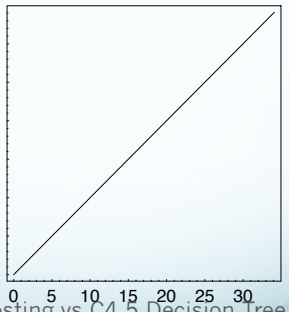


Final Model



Why is Boosting Good?

- Boosting achieves good empirical results
- Why?
- Many answers
  - Statistical View of Boosting
  - Boosting and Max Margin
  - PAC Learning (learning theory)
  - Game theory



Boosting vs C4.5 Decision Tree
Points above the line are better for boosting

Boosting

Fitting a function to data

- Fitting: Optimization, what parameters can we change?
- **Function: Model, loss function**
- Data: Weigh data based on previous prediction errors?

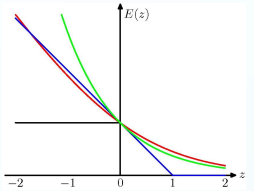
Statistical View of Boosting

- What is boosting doing?
  - We normally think of classifiers in terms of loss or likelihood
  - What is the objective function for boosting?

Exponential Loss

- Boosting can be viewed as an algorithm for minimizing exponential loss

$$\sum_i \exp(-y_i f(x_i))$$



- Choices of α_t and h_t minimize this loss
- A form of coordinate descent
- Is that it?

Synthetic Data Experiment

	% test error [# rounds]					
exp. loss	exhaustive AdaBoost		gradient descent		random AdaBoost	
10^{-10}	0.0	[94]	40.7	[5]	44.0	[24,464]
10^{-20}	0.0	[190]	40.8	[9]	41.6	[47,534]
10^{-40}	0.0	[382]	40.8	[21]	40.9	[94,479]
10^{-100}	0.0	[956]	40.8	[70]	40.3	[234,654]

Table 1 Results of the experiment described in Section 4. The numbers in brackets show the number of rounds required for each algorithm to reach specified values of the exponential loss. The unbracketed numbers show the percent test error achieved by each algorithm at the point in its run at which the exponential loss first dropped below the specified values. All results are averaged over ten random repetitions of the experiment. (Reprinted from [30] with permission of MIT Press.)

Regularization?

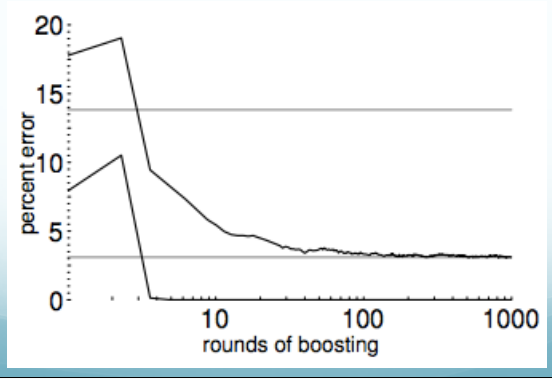
- No explicit form of regularization
- Observation: stopping (variant of) AdaBoost after (any) number of rounds provides an approximate solution to L_1 regularized minimization of exponential loss
  - Set α on each round to be small positive constant
  - Resulting classifiers approximate regularized minimization
- In practice, illustrative but doesn't apply to AdaBoost

Overfitting

- AdaBoost seems like it would overfit
  - Each round focuses more on incorrect examples
  - VC theory tells us it will overfit (details not important)
- It doesn't overfit!

Overfitting

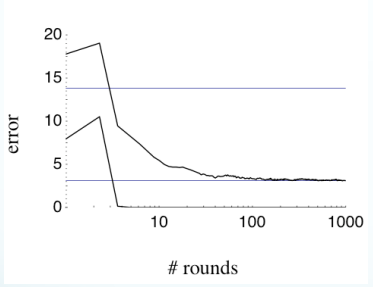
- Top: test error
- Bottom: train error



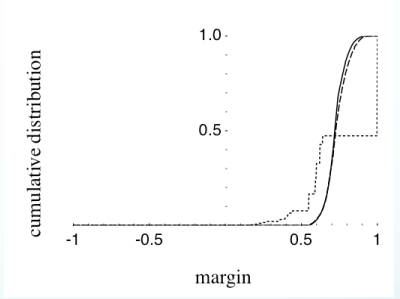
Margin Explanation

- Train error goes to 0 right away
- Do we get benefits in continued training?
- Yes! The margins improve
  - Better margins should give us better test error
  - Similar idea from SVMs

Margin Explanation



Learning curve for training (lower) and test (upper)



Cumulative distribution of margins of the training examples
5,100,1k iterations (dashed to solid)
Boosting increases the margin even after training error goes to 0

Margin Explanation

- Step 1: We can prove a generalization bound on AdaBoost that depends on margins of training examples, NOT number of rounds
- Over-fitting no longer a function of rounds
  - Depends on margins
  - Won't overfit as long as large margins can be achieved
- Step 2: AdaBoost generally increases margins of training examples
- Idea: design boosting that directly maximizes the margins!
  - Hasn't worked. Produces overly complex weak hypotheses, more over-fitting

SVM Connection

- Based on the generalization bound, and margin observations
- We can write Boosting as maximizing the minimum margin

$$\max_{\alpha} \min_{i} \frac{(\alpha \cdot h(x_i)) y_i}{\|\alpha\| \, \|h(x_i)\|}$$

Difference in Norms

$$\max_{\alpha} \min_{i} \frac{(\alpha \cdot h(x_i)) y_i}{\|\alpha\| \, \|h(x_i)\|}$$

- The norms for boosting are defined as

$$\|\alpha\|_1 = \sum_t |\alpha_t| \qquad \|h(x)\|_\infty = \max_t |h_t(x)|$$

- For SVMs, the norms are Euclidean

$$\|\alpha\|_2 = \sqrt{\sum_t \alpha_t^2} \qquad \|h(x)\|_2 = \sqrt{\sum_t h_t(x)^2}$$

- Boosting and SVMs are very similar

Differences

- The norms are different
  - In high dimensional space, the effective enforced margins may be very different
- SVM requires quadratic programming
  - Boosting requires linear programming
- Finding high dimensional separators
  - SVM uses kernels (inner products of examples)
  - Boosting uses weak learners to handle this
  - There is usually a big difference between the learning spaces of the weak learners and the kernels
- This isn't the whole story
  - Only part of the bound corresponds to maximizing the margin

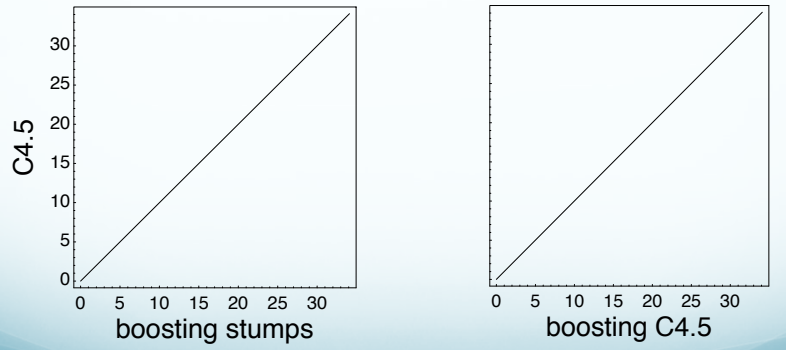
Story of Boosting

- These explanations illustrate the story of boosting
- Boosting works
  - But why?
- Different analyses yield different conclusions
  - Some not supported by empirical evidence
- Build new boosting algorithms based on these observations
  - Some work, others don't

What Should We Boost?

- If boosting improves a base classifier…
  - Boost the best classifier we can!
    - Boosted SVMs, KNN, Perceptrons, Logistic Regression, etc.
  - Problem: you have to train many of these, may not be efficient
- But boosting can boost a *weak* learner
  - For simplicity, let's use a weak learner
  - Decision stumps: very weak, can only look at 1 feature at a time
    - Decision trees with depth of 1

Boosting Results



Practical Advantages

- Easy to implement
- Very fast
- No parameters to tune
- Not specific to any weak learner
- Well motivated by learning theory
- Can identify outliers
- Extensions to:
  - Multi-class, Ranking, Regression

Boosting

Fitting a function to data

- Fitting: Specialized procedure for fitting to data
- Function: exponential loss, linear combination of underlying classifiers
  - The underlying classifiers determine hypothesis class
- Data: Weigh data based on previous prediction errors

Combining Classifiers

- Boosting uses weak learners
- What if I have multiple classifiers that I want to combine? Is there a general way?
  - Yes
  - Mixtures of experts

Mixtures of Experts

- Assume we have many "experts" giving us advice
  - Each expert examines an example and returns a label
- How can we combine this mixture of experts to create a single output?
- Intuition: some experts are better than others
- Solution: Learn which experts are the best and trust them the most

Weighted Majority

- Initialize the weight $w_k=1$ for expert k
  - Assume binary classification, $y_i=\{-1,+1\}$
- For each example x_i
  - Predict $\hat{y}_i = \text{sign}\left(\sum_k w_k f_k(x_i)\right)$
  - Update:
    - For each expert k
      - If $y_i \neq f_k(x_i)$
        - $w_k = w_k / 2$

Weighted Majority

- An online algorithm for learning mixtures of experts
- Bounded by regret to the best expert
  - Theorem states: the number of mistakes made by the weighted majority algorithm is never more than:

    $$2.41(m+\log k)$$

    where m is the number of mistakes made by the best expert so far
  - We will never do much worse than the best expert
  - Since we don't know the best expert, this is great
    - Only a log penalty for adding more experts

Weighted Majority

- First introduced by Littlestone and Warmuth (1994)
- No assumptions about data or expert quality
- Widely used in lots of settings
  - Stock portfolio balancing
  - Experts are individual stocks

Why Combine?

- We've talked about several ways to combine
- But why are combinations good?
- An example: you want to get advice about which stocks to invest in
  - What should you do?
  - Call the same stock broker 100 times and average?
  - Call 100 different stock brokers and average?

Diversity in Experts

- We can improve by combining multiple classifiers since they have a diversity of opinions
  - They won't all make the same mistakes
  - If they are all very good, then we can vote them to get even better
    - Reality: they do make some of the same mistakes
  - This is the idea behind boosting
    - If you can do a bit better than random (weak), then you can boost that to good performance (strong)

Creating Diversity

- We can create diversity by using K different classifiers
- We can also create diversity by creating K different *datasets*
- Bagging: create many different datasets by hiding some of the data
  - Instance bagging
  - Feature bagging

Instance Bagging

- Given N examples for training
  - Create K datasets
    - Select N examples with *replacement* from the training set
      - Probability of example being selected: 63.2%
    - Train a classifier on the dataset
- Final output: voting of the K classifiers
  - Or: weighted majority of the K classifiers!

Feature Bagging

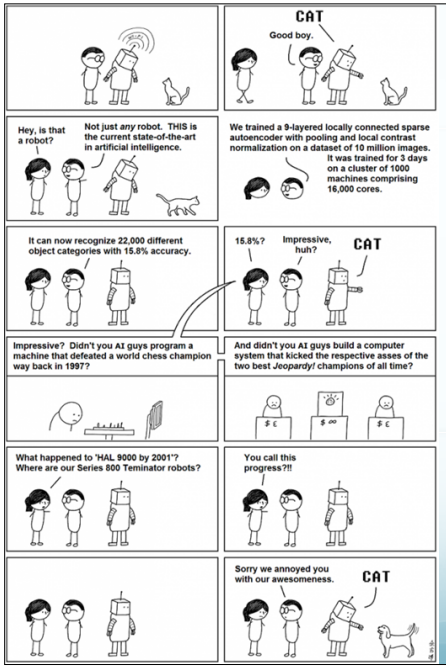
- Given N examples for training
  - Create K datasets
    - Select (K-1)/K of the features to use
      - Ignore the rest
    - Train a classifier on the dataset
- Final output: voting of the K classifiers
  - Or: weighted majority of the K classifiers!

Co-Training

- Feature bagging is closely connected to co-training
  - Blum and Mitchell 1998
- Co-training: A semi-supervised learning algorithm in which you have two representations of each example
  - Given: labeled (few) and unlabeled (many) data
  - Train a separate classifier on each representation
  - Label the unlabeled data
  - If one classifier is confident in its prediction, use it for training for both
- Uses diversity of representations to improve performance

Summary

- We can do a lot by combining a little
  - Boosting: turns weak learners into strong learners
- Mixtures of experts
  - Weighted majority uses the predictions of the best experts
- Diversity helps
  - Create artificial diversity
    - Instance bagging
    - Feature bagging
  - Diversity in representations for semi-supervised learning
    - Co-training



Artificial Neural Networks

Mark Dredze

Machine Learning
CS 600.475

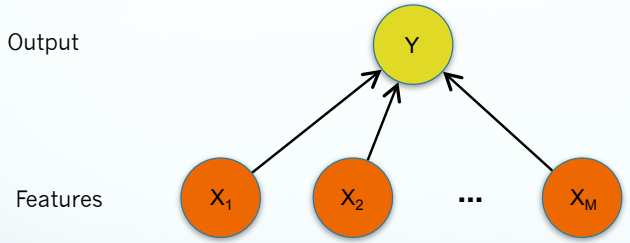
Handling Non-Linear Data

- Option 1: Add features by hand that make the data separable
  - Requires feature engineering
- Option 2: Learn a small number of additional features that will suffice
  - Today
- Option 3: Kernel trick

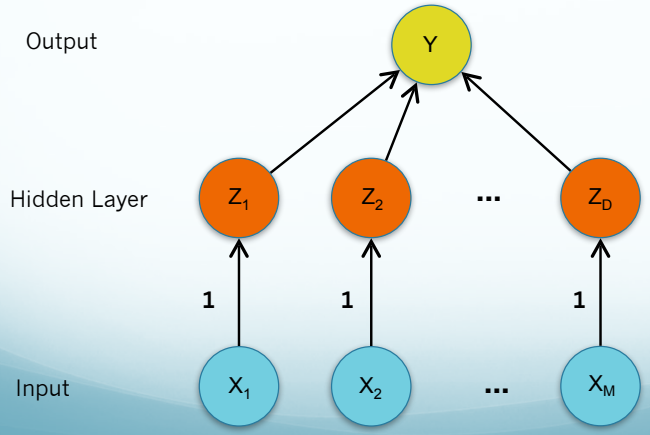
Motivation

- Where do features come from?
  - We build them by hand
- What if we wanted to *learn* features?
  - Goal: learn features that give linearly separable data
- After learning features apply usual linear classifier

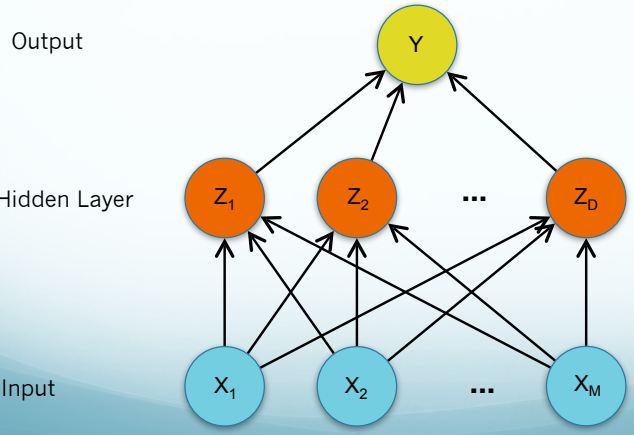
Perceptron: Graphical Representation



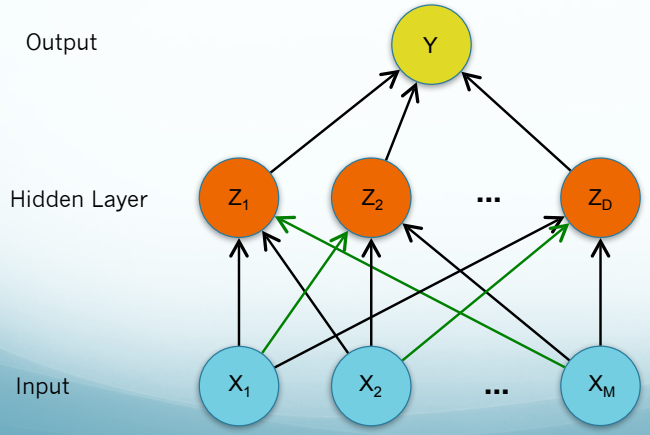
Perceptron: Graphical Representation



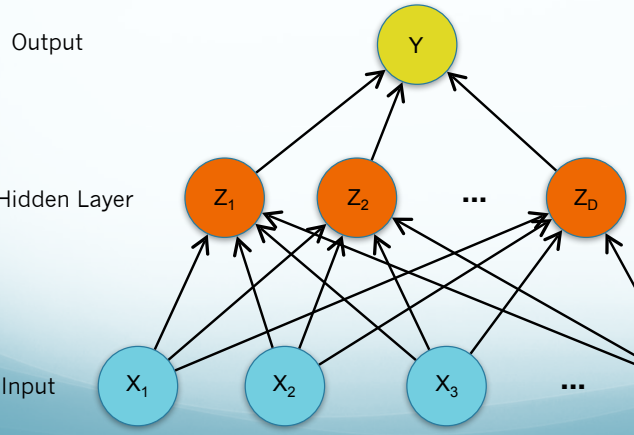
Perceptron: Graphical Representation



Perceptron: Graphical Representation



Perceptron: Graphical Representation



Why?

- Constraints from X
  - When D = M, likely to copy the features from X to Z
  - When D < M, cannot make an exact copy of X
    - Must come up with a representation that is more efficient
- Constraints from Y
  - Z should be a representation that helps learn Y
  - Forces the low-dimensional representation to capture properties of X useful in predicting Y

Why Non-Linear

- Generalized linear classifiers!
  - Start with linear function
    - $w \cdot x$
  - Pass the output through a non-linear function
    - $\hat{y} = h(w \cdot x)$
  - What is h?
    - Non-linear function
      - Logistic function
      - Sign function
- Each Z is the output of a non-linear function
  - Combinations of Z are now non-linear in X

Multi-Layer Perceptrons

Fitting a function to data

- **Fitting: what type of optimization algorithm?**
- Function: non-linear: linear combination of generalized linear functions
- Data: Data/model assumptions? How we use data?

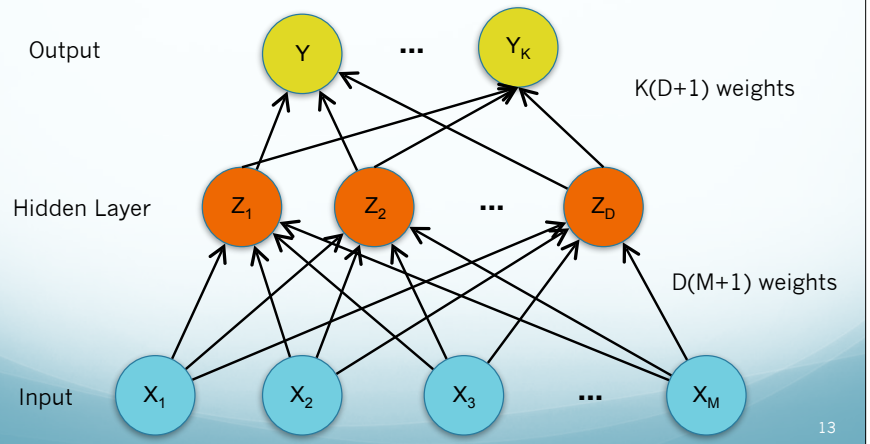
How Will We Learn?

- Perceptron: a training method for generalized linear classifiers
  - Training method for linear classifiers
  - Minimize the error of the training data
  - Chain multiple Perceptrons together
  - Update rule:

$$w^{j+1} = w^j + \nabla f(x, y)$$

  - The real work will be in computing the gradient

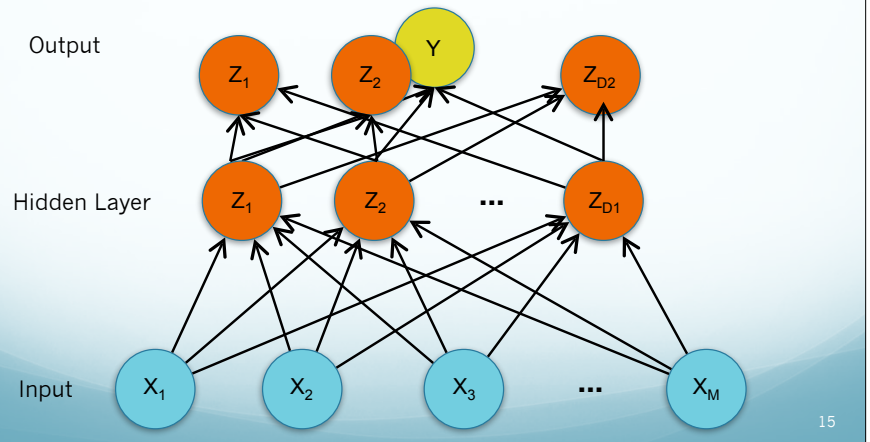
Example: Multi-Class



Network Terminology

- Input nodes: x
- Output node: y
- Hidden nodes: z
  - This network has 1 hidden layer
  - 2 layer network (two layers to learn)
- h for hidden nodes are called activation functions
- h for output depends on task
  - Identity for regression
  - Logistic for classification

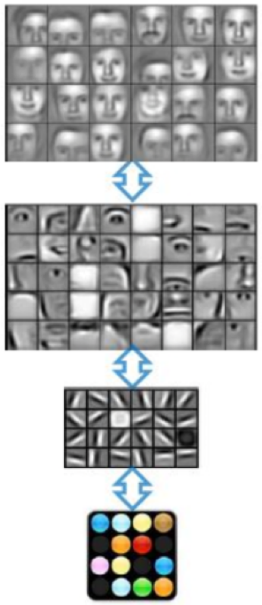
Deep Networks



Deep Networks

- Learn multiple levels of features at higher and higher abstractions
- Same learning techniques
  - Just more complex gradients

Feature representation



3rd layer
"Objects"

2nd layer
"Object parts"

1st layer
"Edges"

Pixels

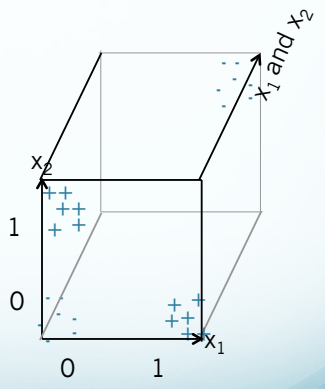
Example: Image Processing

Outline

- Lecture 1: Neural Networks
  - Nonlinearities
  - Objective Functions
  - Training
  - Gradient Computations
- Lecture 2: Deep Learning
  - Supervised and unsupervised training
  - Pre-training
  - Auto-encoders

An Non-linear Example

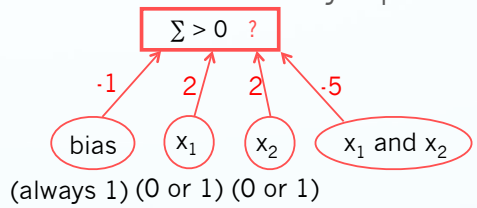
- Consider the xor function
  - y(x) = 1 iff x_1 xor x_2
- Clearly non-linear
  - No values for w will produce desired output
- We could solve this by adding a new feature
  - x_3= x_1 xor x_2



Example from Jason Eisner

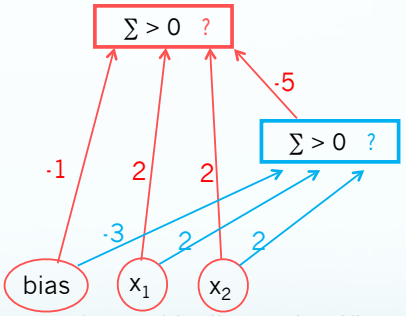
The Neural Network Solution

- Learn new features that are linearly separable



- We now have a linear classifier for XOR
- How do we learn these features?

The Neural Network Solution

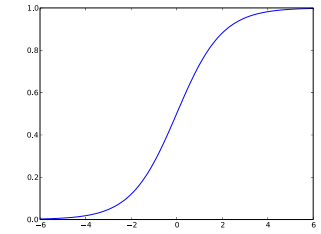


- The new features are learned by linear classifiers
  - All other hidden nodes (not shown) just replicate input
- The activation function makes the feature 1 or 0

Non-linear Activation Functions

- What non-linear function should we use for activation function h?
  - Typically use sigmoid functions
    - Logistic function

$$g_\alpha(x) = \frac{1}{1 + e^{-\alpha x}}$$



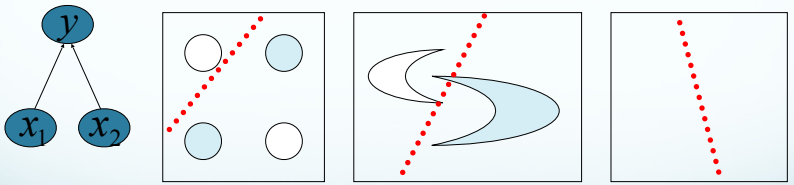
- Each hidden node has a threshold for activation
  - Will be 0 and then quickly transition to 1
- This is what we use when we stack Perceptron
- This is why we think of hidden nodes as features
  - They are off and then when enough input they turn on
  - Learning input weights turns on the feature!

Hypothesis Class

- What can a neural network learn?
  - Obviously highly non-linear outputs
- Universal approximators
  - With enough hidden layers and hidden nodes a neural network can model any continuous function on compact input domain (some number of inputs)
  - The power of the networks depends on its structure
    - General result independent of activation functions

Decision Boundary

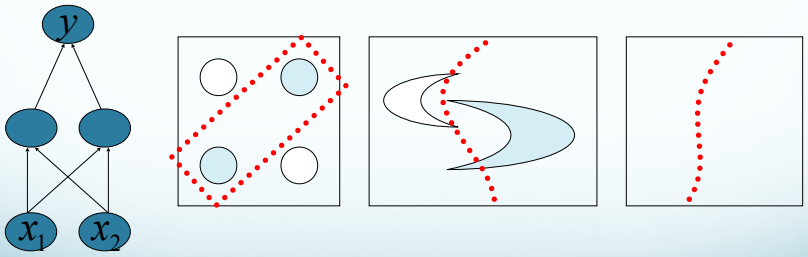
- 0 hidden layers: linear classifier
  - Hyperplanes



Example from to Eric Postma via Jason Eisner

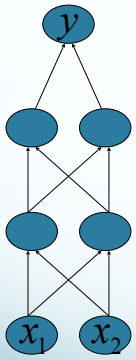
Decision Boundary

- 1 hidden layer
  - Boundary of convex region (open or closed)

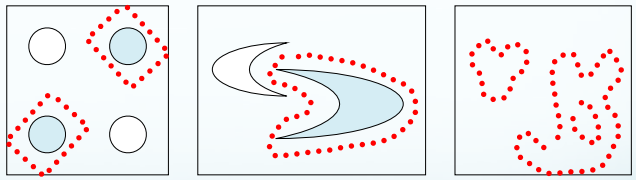


Example from to Eric Postma via Jason Eisner

Decision Boundary



- 2 hidden layers
  - Combinations of convex regions



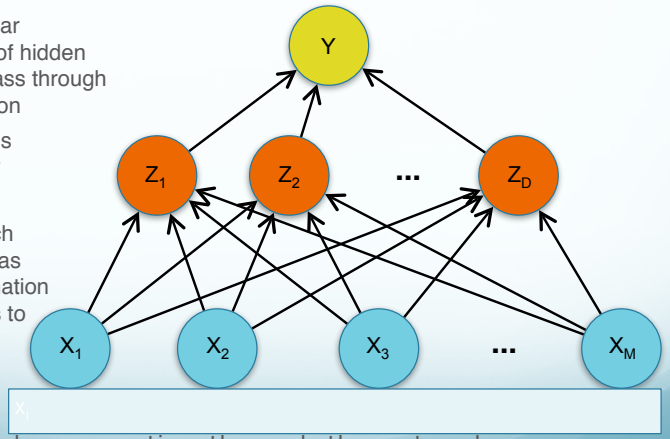
Example from to Eric Postma via Jason Eisner

Prediction

Compute linear combination of hidden nodes and pass through logistic function

Hidden nodes are now new features

Compute each hidden node as linear combination of x and pass to logistic



- Forward propagation through the network

Classification Objective

- Define an error function and minimize
- Cross entropy error function

$$E(w) = -\sum_{i=1}^{N} \{ y_i \ln \hat{y}_i + (1 - y_i) \ln(1 - \hat{y}_i) \}$$

- This arises naturally when we consider a logistic probability model and take the negative log likelihood
  - Details in the book

Regression Objective

- For regression we use the sum of squares error

$$E(w) = \frac{1}{2}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

- If we assume a Gaussian model for y, the error function arises from maximizing the likelihood function
  - We saw the same thing for linear regression

Combined Model

(See lecture notes for derivation)

Combined Model

- Writing the generalized linear classifier with generalized non-linear basis functions

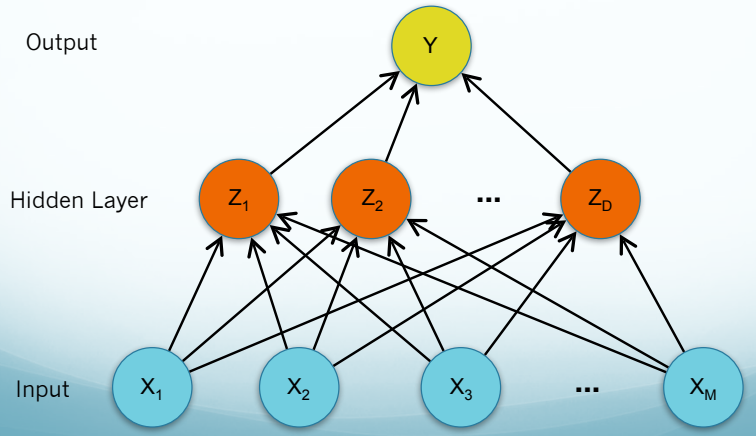
$$y(x;w) = h^{(2)}\left(\sum_{j=1}^{D} w_j^{(2)} h^{(1)}\left(\sum_{i=1}^{M} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_0^{(2)}\right)$$

  - $h^{(1)}$ is the non-linear function for the basis function
  - $h^{(2)}$ is the non-linear function for the output
  - $w^{(1)}$ are the parameters for the basis function
  - $w^{(2)}$ are the parameters for the linear model
  - w_0 are the bias parameters (shown here for clarity)

Training

- Prediction is relatively easy
- Learning is where the magic happens
- Strategy: compute the gradient of the objective function
  - Similar to perceptron
  - Gradient based update

Graphical Representation

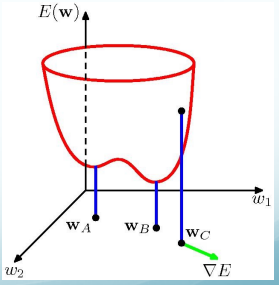


Training

- Prediction is relatively easy
- Learning is where the magic happens
- Strategy: compute the gradient of the objective function
  - Similar to perceptron
  - Gradient based update
- For the moment: assume black box computes gradient

Gradient Based Optimization

- The objective function is now non-convex
- Gradient based optimization NOT guaranteed to find global optimum
- For now: use gradient stochastic gradient and hope for the best
- Next time: tricks for non-convexity key to learning good networks



Computing the Gradient

- For arbitrary Neural Network architectures we can use Backpropagation!

(See lecture notes for derivation)

Algorithm: Neural Network

- Train: Given examples X and Y
  - Y can be multiple outputs
  - Define a network structure
  - ex. 2 layer feed forward, D nodes in hidden layer
  - Learn parameters w
- Predict: given example x
  - For 2 layer feed forward, compute output as

$$\hat{y} = h^{(2)}\left(\sum_{j=1}^{D} w_j^{(2)} h^{(1)}\left(\sum_{i=1}^{M} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_0^{(2)}\right)$$

Multi-Layer Perceptrons

Fitting a function to data

- Fitting: gradient based optimization with back-propogation
- Function: non-linear: linear combination of generalized linear functions
  - Universal approximations
  - can model any continuous function on compact input domain (some number of inputs)
- Data: Batch training using stochastic methods

Next Time

Deep(er) Networks

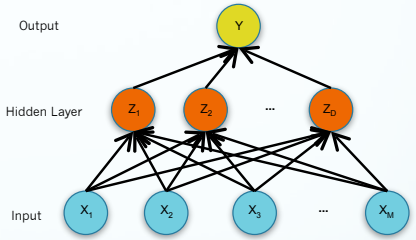
Deep Learning

Mark Dredze
Slides by Matt Gormley

Machine Learning
CS 600.475

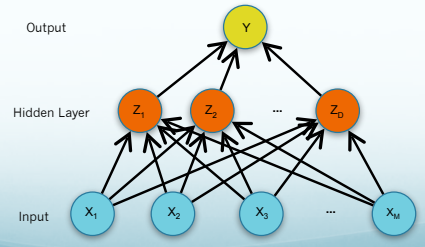
Recap of Neural Nets

- Motivation:
  - Learn features automatically
  - Capture non-linearities of data
- Two layers of binary logistic regression define a two-layer neural network
- Supervised training by SGD
  - Compute gradient with backpropagation
  - Take small steps in the direction of the gradient



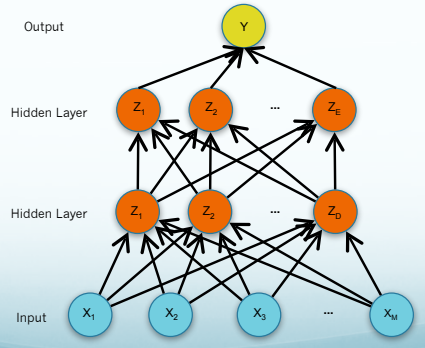
Deeper Networks

This lecture:



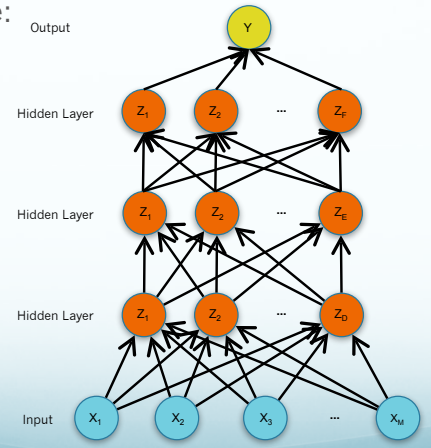
Deeper Networks

This lecture:



Deeper Networks

This lecture: Making the neural networks deeper



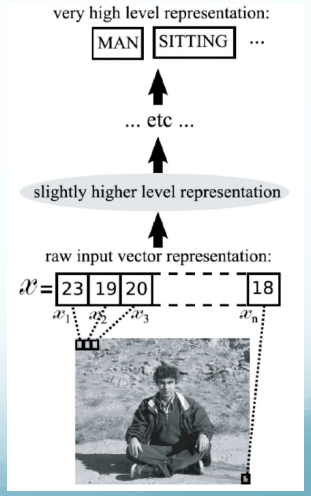
Motivation: Why go Deep?

- 2-layer Neural Nets are already universal function approximators...
  - But deep architectures can be representationally efficient
  - Fewer computational units for the same function
- 2-layer Neural Nets can represent non-linear combinations of the input features
  - But deep representations might allow for a hierarchy
  - Allows non-local generalizations
- Deep Nets: Multiple levels of latent variables allow combinatorial sharing of statistical strength
- 2-layer Neural Nets work well
  - But deep representations have been shown to work even better (vision, audio, NLP, etc.)!

Slide adapted from Honglak Lee (NIPS 2010)

The Promise of Deep Architectures

- Transform input image into higher levels of representation:
  - edges, local shapes, object parts, etc.
- We don't know the "right" levels of abstraction
- So let the model figure it out!

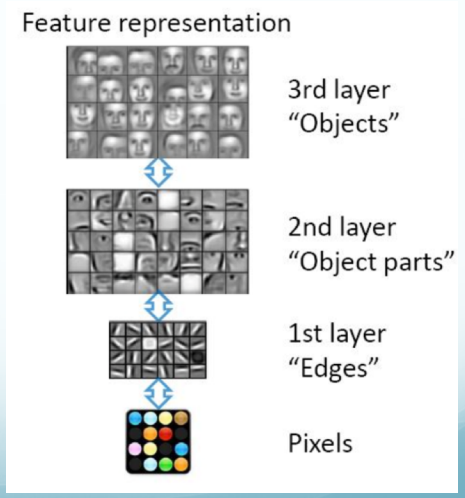


Example from Bengio (2009)

Different Levels of Abstraction

Face Recognition:

- Deep Network can build up increasingly higher levels of abstraction
- Lines, parts, regions



Example from Honglak Lee (NIPS 2010)

Deep Network Training

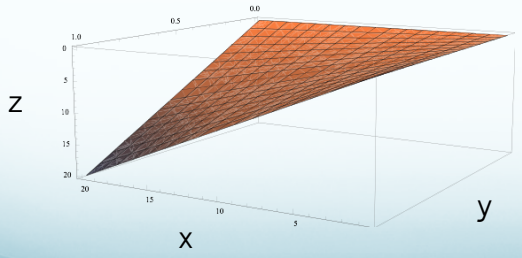
(that doesn't always work)

- **Idea #1: (Just like a shallow network)**
  - Compute the supervised gradient by backpropagation.
  - Take small steps in the direction of the gradient (SGD)

- Requires labeled data
  - Usually have gobs of unlabeled data
  - But can't learn from it
- What goes wrong?
  - A. Gets stuck in local optima
    - Nonconvex objective
    - Usually start at a random (bad) point in parameter space
  - B. Gradient is progressively getting more dilute
    - "Vanishing gradients"

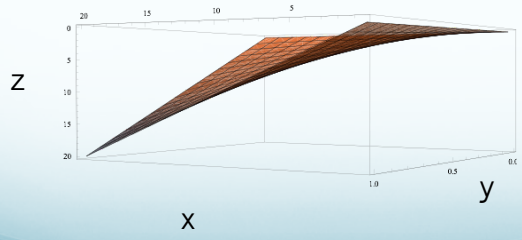
The problem: *Nonconvexity*

- Where does the nonconvexity come from?
- Even a simple quadratic z = xy objective is nonconvex:



The problem: *Nonconvexity*

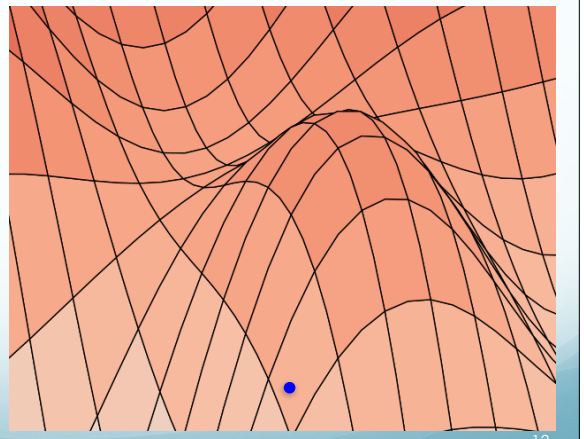
- Where does the nonconvexity come from?
- Even a simple quadratic z = xy objective is nonconvex:



The problem: *Nonconvexity*

Stochastic Gradient Descent...

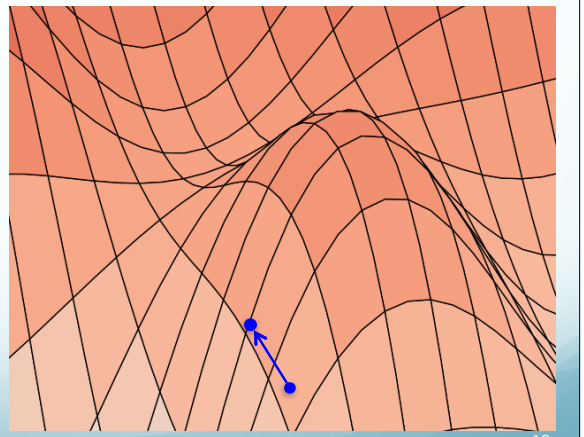
...climbs to the top of the nearest hill...



The problem: *Nonconvexity*

Stochastic Gradient Descent...

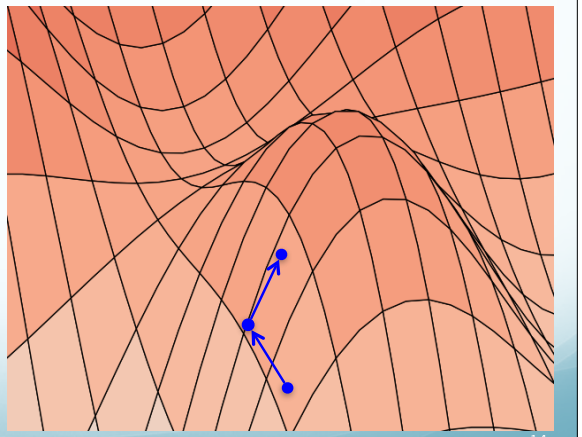
...climbs to the top of the nearest hill...



The problem: *Nonconvexity*

Stochastic Gradient Descent...

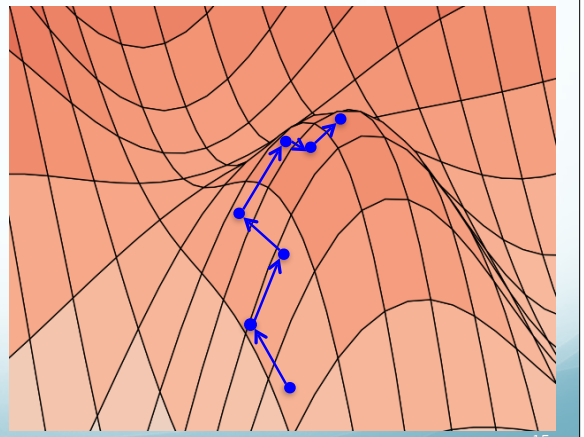
...climbs to the top of the nearest hill...



The problem: *Nonconvexity*

Stochastic Gradient Descent...

...climbs to the top of the nearest hill...

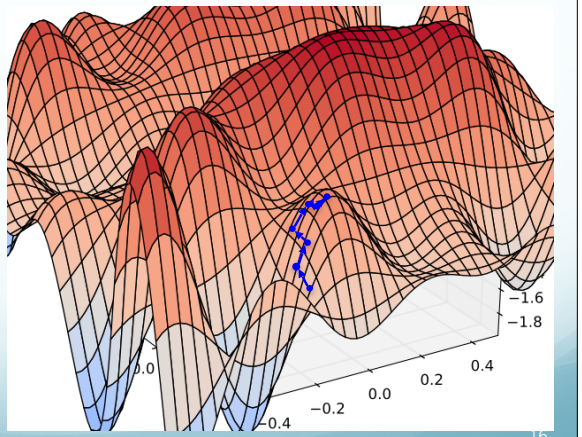


The problem: *Nonconvexity*

Stochastic Gradient Descent...

...climbs to the top of the nearest hill...

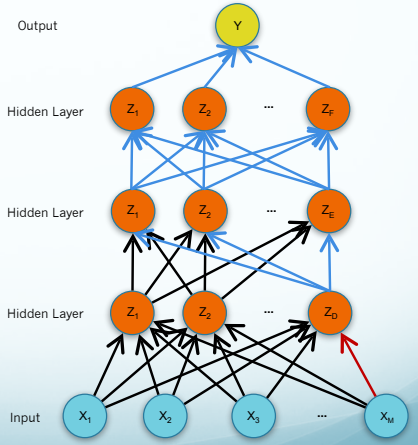
...which might not lead to the top of the mountain



The problem: *Vanishing Gradients*

The gradient for an edge at the base of the network depends on the gradients of many edges above it

The chain rule multiplies many of these gradients (adjoints) together



Deep Network Training

(that doesn't always work)

- **Idea #1: (Just like a shallow network)**
  - Compute the supervised gradient by backpropagation.
  - Take small steps in the direction of the gradient (SGD)

- Requires labeled data
  - Usually have gobs of unlabeled data
  - But can't learn from it
- What goes wrong?
  - A. Gets stuck in local optima
    - Nonconvex objective
    - Usually start at a random (bad) point in parameter space
  - B. Gradient is progressively getting more dilute
    - "Vanishing gradients"

Deep Network Training

(that still doesn't work)

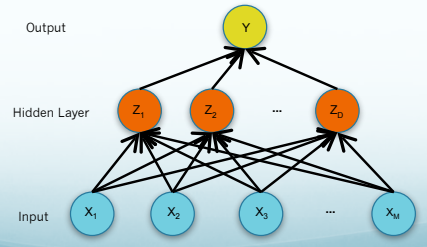
- **Idea #2: (Two Steps)**
  - **Train each level** of the model in a **greedy** way
  - Then use our **original idea**

1. Supervised Pre-training
   - Use **labeled** data
   - Work bottom-up
     - Train hidden layer 1. Then fix its parameters.
     - Train hidden layer 2. Then fix its parameters.
     - ...
     - Train hidden layer n. Then fix its parameters.
2. Supervised Fine-tuning
   - Use **labeled** data to train following "Idea #1"
   - Refine the features by backpropagation so that they become tuned to the end-task

Deep Network Training

(that still doesn't work)

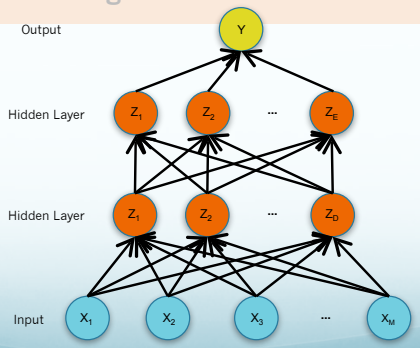
- **Idea #2: (Two Steps)**
  - **Train each level** of the model in a **greedy** way
  - Then use our **original idea**



Deep Network Training

(that still doesn't work)

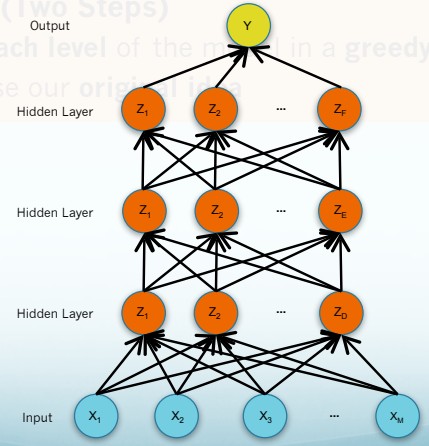
- **Idea #2: (Two Steps)**
  - **Train each level** of the model in a **greedy** way
  - Then use our **original idea**



Deep Network Training

(that still doesn't work)

- **Idea #2: (Two Steps)**
  - **Train each level** of the model in a **greedy** way
  - Then use our **original idea**



Deep Network Training

(that actually works!!)

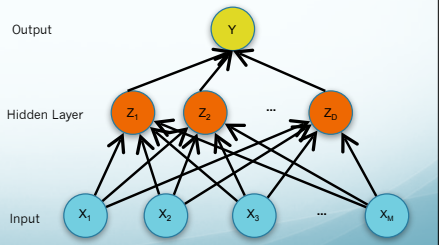
- **Idea #3: (Two Steps)**
  - Use our original idea, but **pick a better starting point**
  - **Train each level** of the model in a **greedy** way

1. Unsupervised Pre-training
   - Use **unlabeled** data
   - Work bottom-up
     - Train hidden layer 1. Then fix its parameters.
     - Train hidden layer 2. Then fix its parameters.
     - ...
     - Train hidden layer n. Then fix its parameters.
2. Supervised Fine-tuning
   - Use **labeled** data to train following "Idea #1"
   - Refine the features by backpropagation so that they become tuned to the end-task

The solution: *Unsupervised pre-training*

Unsupervised pre-training of the first layer:

- What should it predict?
- What else do we observe?
- **The input!**

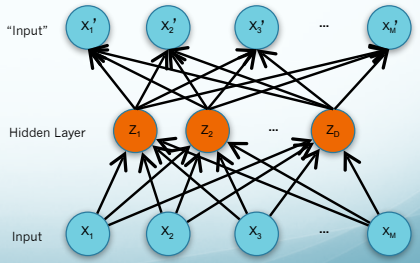


The solution: *Unsupervised pre-training*

Unsupervised pre-training of the first layer:

- What should it predict?
- What else do we observe?
- **The input!**

This topology defines an Auto-encoder.



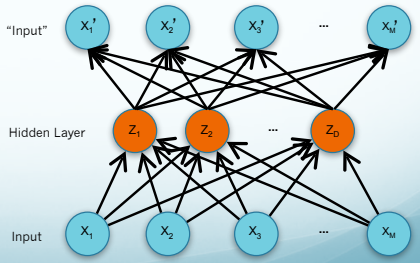
Auto-Encoders

Key idea: Encourage z to give small reconstruction error:

- x' is the *reconstruction* of x
- Loss = $|| x - \text{DECODER}(\text{ENCODER}(x)) ||^2$
- Train with the same backpropagation algorithm for 2-layer Neural Networks with x_m as both input and output.

DECODER: $x' = h(W'z)$

ENCODER: $z = h(Wx)$

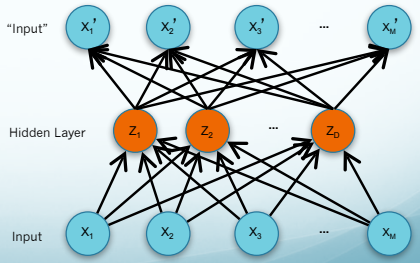


Slide adapted from Raman Arora

The solution: *Unsupervised pre-training*

Unsupervised pre-training

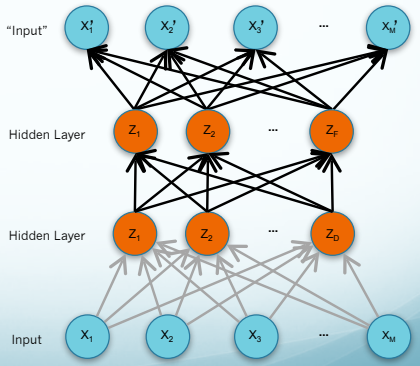
- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - ...
  - Train hidden layer n. Then fix its parameters.



The solution: *Unsupervised pre-training*

Unsupervised pre-training

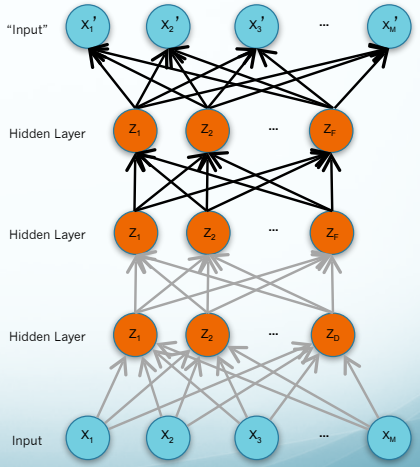
- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - ...
  - Train hidden layer n. Then fix its parameters.



The solution: *Unsupervised pre-training*

Unsupervised pre-training

- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - ...
  - Train hidden layer n. Then fix its parameters.

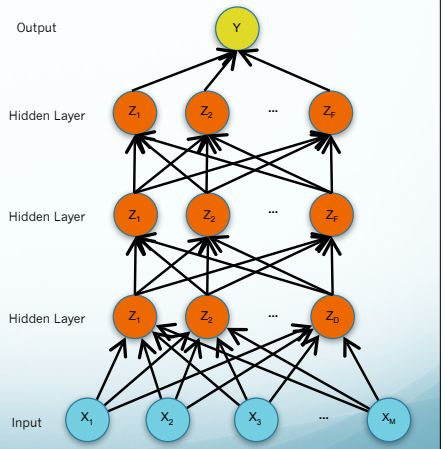


The solution: *Unsupervised pre-training*

Unsupervised pre-training

- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - ...
  - Train hidden layer n. Then fix its parameters.

Supervised fine-tuning
Backprop and update all parameters



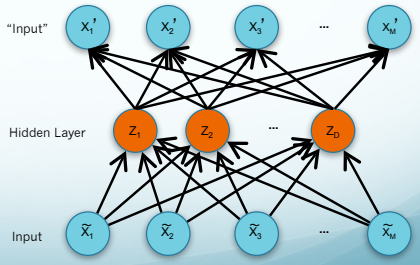
Denoising Auto-encoders

Key idea: Encourage z to give small reconstruction error

- x' is the *reconstruction* of $\tilde{x}$ = x+noise
- Loss = || x – DECODER(ENCODER(x + noise)) ||^2
- Train with the same backpropagation algorithm

DECODER: x' = h(W'z)

ENCODER: z = h(W$\tilde{x}$)
where $\tilde{x}$ = x+noise



Slide adapted from Raman Arora

Deep Network Training

- **Idea #1:**
  1. Supervised fine-tuning only
- **Idea #2:**
  1. Supervised layer-wise pre-training
  2. Supervised fine-tuning
- **Idea #3:**
  1. Unsupervised layer-wise pre-training
  2. Supervised fine-tuning

Deep Network Training

Results from Bengio et al. (2006)

Percent error (lower is better) on MNIST digit classification task

		Experiment 2			Experiment 3		
		train.	valid.	test	train.	valid.	test
Idea #3:	Deep net, auto-associator pre-training	0%	1.4%	1.4%	0%	1.4%	1.6%
Idea #2:	Deep net, supervised pre-training	0%	1.7%	2.0%	0%	1.8%	1.9%
Idea #1:	Deep net, no pre-training	.004%	2.1%	2.4%	.59%	2.1%	2.2%
	Shallow net, no pre-training	.004%	1.8%	1.9%	3.6%	4.7%	5.0%

Demo of Neural Network on MNIST digit classification:
http://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html

Is layer-wise pre-training always necessary?

Answer in 2006: Yes!
Answer in 2014: No!

1. If initialization is done well by design (e.g. sparse connections and convolutional nets), maybe won't have vanishing gradient problem
2. If you have an extremely large datasets, maybe won't overfit. (But maybe that also means you want an ever deeper net)
3. New architectures are emerging:
   - Stacked SVM's with random projections [Vinyals et al., 2012]
   - Sum-Product Networks [Poon and Domingos, 2011]

Slide adapted from Raman Arora

Deep Learning

- Goal: learn features at different levels of abstraction
- Training can be tricky due to...
  - Nonconvexity
  - Vanishing gradients
- Unsupervised layer-wise pre-training can help with both!