1. Concept Learning

1. instance: x . labels: y
2. Concept : subset of objects from some space.
3. Concept space C: The set of unique concepts.
4. **Hypothesis space H**: a set of unique hypothesis
5. H != C
6. concept learning task: a hypothesis h in H such that h(x) = c(x) for all x in D.
7. consistent: iff h(x) = c(x) for each training example <x,c(x)> in D, hypothesis h is consistent with a set of training examples D of the target concept c.
8. **version space**: all the hypotheses that are **consistent** with the training examples.
9. how many hypotheses? 因为还有多两种 ?(don' care) Φ(no values allowed)所以全+2

- Given this encoding of hypotheses how many hypotheses are possible?

Number of Feet	Fur	Size	Has wings	Warm Blood
Integers 0 to 99	Yes,No	S,M,L,XL,XXL	Yes,No	Yes,No

(102) * 4 * 7 * 4 * 4 = 45,696 hypotheses

10. Concept learning can be thought of search through a hypotheses space and find the one or more that match the data.

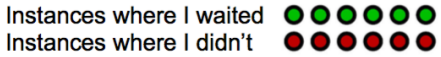
2. Decision Trees(supervised)

1. DTs can express any function of the input attributes.
2. DTs divide the feature space into axis-paralled rectangles and label each rectangle with one of the k classes.
3. recursive, greedy algorithm to build a tree
4. entropy:

$$H(\langle P_1, \ldots, P_n \rangle) = \sum_{i=1}^{n} - P_i \log_2 P_i$$

5. **ID3**: pick the attribute that creates the **lowest overall entropy.** Information gain = H(S) - Hp, the attribute with **largest information gain** is chosen.

Entropy prior to splitting



$P_1 = \text{probability I will wait for a table}$

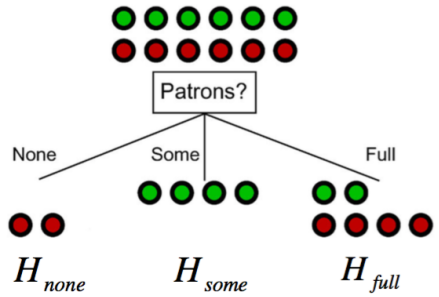
$P_2 = \text{probability I will NOT wait for a table}$

$$H_0\langle P_1, P_2\rangle = \sum_j -P_j \log_2 P_j$$

$$= -P_1 \log_2 P_1 - P_2 \log_2 P_2$$

$$= 1$$

If we split on Patrons



$$H_{Patrons} = W_{none} H_{none} + W_{some} H_{some} + W_{full} H_{full}$$

$$= \frac{2}{12}0 + \frac{4}{12}0 + \frac{6}{12}\left(-\frac{2}{6}\log_2\frac{2}{6} - \frac{4}{6}\log_2\frac{4}{6}\right) = .459$$

6. Avoid overfitting: 1. stop splitting when information gain is low pr the split is not statistically significant. 2. grow full tree and prune it.
7. C4.5: allowing continuous value attributes, allow missing attributes, prune trees after building.

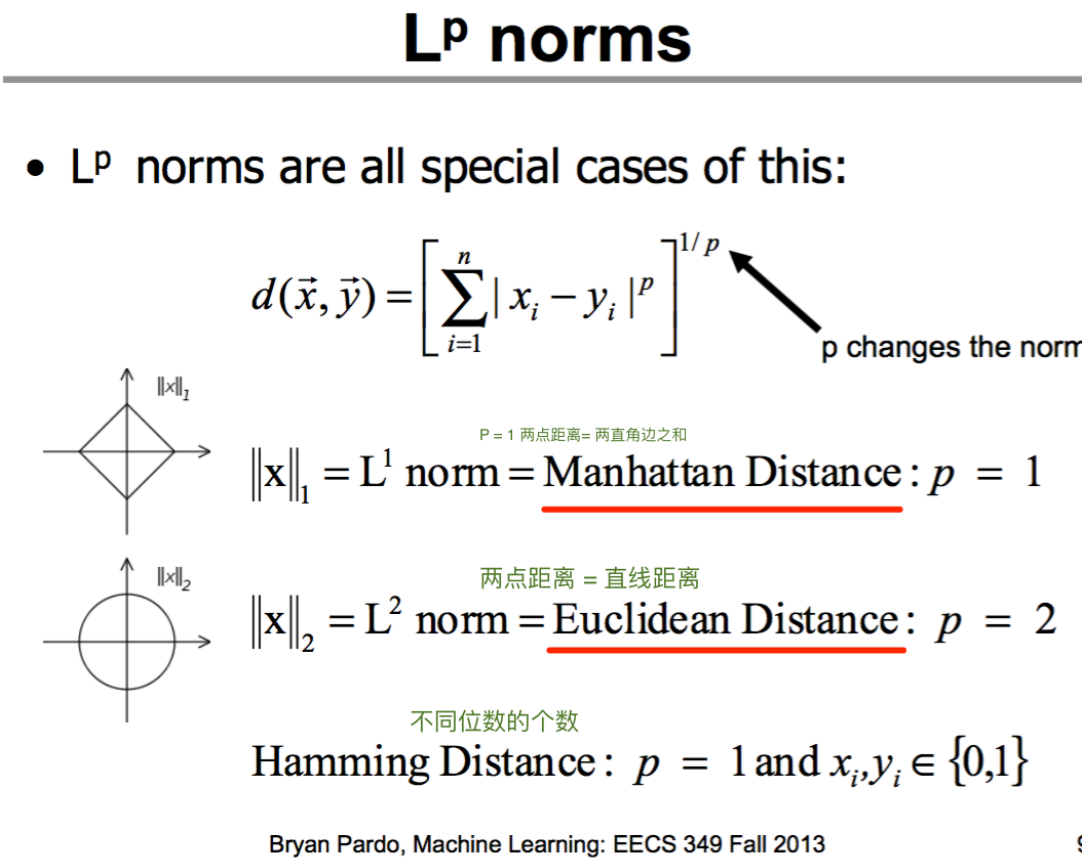
3. Measuring Distance

1. **metric**: a function of two values with these qualities:度量（metric）是对一个集合里面的两个元素而言的，是两点间距离的抽象。
   reflexivity(自反性): d(x,y) = 0 iff x = y
   non-negative: d(x,y) >= 0
   symmetry: d(x,y) = d(y,x)
   triangle inequality: d(x,y) + d(y,z) >= d(x,z)
2. **Norm**: a function that applies a **positive** value to all vectors except the 0 vector.范数（norm）是对一个元素而言的，是实数的绝对值或复数的模这样的长度的抽象。
   positive scalability: p(av) = |a|p(v)
   p(u) = 0 iff u is the zero vector
   triangle inequality: p(u) + p(v) >= p(u+v)

a norm vector: a function that assigns a strictly positive value to all vectors except 0 vector.

a normal vector: perpendicular to each other. 垂直向量

3. **every norm determines a metric.** d(x,y) = ||x-y||
4. some metrics determine a norm. ||x|| = d(x,0)
5. P-norms distances:

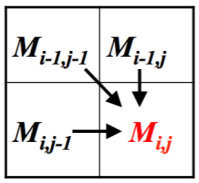


6. wighted norms: 只能表示轴平行于坐标轴的椭圆

$$d(\vec{x},\vec{y}) = \left[\sum_{i=1}^{n} w_i \mid x_i - y_i \mid^p\right]^{1/p}$$

7. levenshtein distance:

$$M_{i,j} = \min\begin{cases} M_{i-1,j} + \mu(-,q_j) & \text{Insert} \\ M_{i,j-1} + \mu(s_i,-) & \text{Delete} \\ M_{i-1,j-1} + \mu(s_i,q_j) & \text{Match} \end{cases}$$



4. memory(instance)-based learning

1. **eager vs lazy**(memory-based is lazy)

Eager vs. Lazy

Eager learning	Lazy learning
• Learn model ahead of time from training data	• Delay the learning process until a query example must be labeled
明确的 • Explicitly learn **h** from training data	• **h** is implicity learned from training data
• E.g. decision tree, linear regression, svm, neural nets, etc.	• E.g. Nearest neighbor, kNN, locally weighted regression, etc.

2. memory based learner: just store each data point in memory (or a database). Making a prediction about the output that will result from some input attributes based on the data is done by looking for similar points in memory

所需函数： distance measure, number of neighbors to consider(K), a weighting function(optional), how to fit with neighbors

how to fit with neighbors： regression function : 返回平均值 classification function: 哪个多返回哪个

K 太小 有noise(比如有的neighbor值特大),

k太大会导致分类选择(哪个多return哪个)时indistinct模糊的，应根据cross-validation经验选择

What makes a memory based learner?

- A distance measure
  Nearest neighbor: typically Euclidean
- Number of neighbors to consider
  Nearest neighbor: One
- A weighting function (optional)
  Nearest neighbor: unused (equal weights)
- How to fit with the neighbors
  Nearest neighbor: Same output as nearest neighbor

3. kernel regression

Kernel Regression

- A distance measure: ***Scaled Euclidean***
- Number of neighbors to consider: ***All of them***
- A weighting function:

$$w_i = \exp\left(\frac{-d(x_i, x_q)^2}{K_W^2}\right)$$

Nearby points to the query are weighted strongly, far points weakly. The K_w parameter is the Kernel Width.

- How to fit with the neighbors:

$$h(x_q) = \frac{\sum_i w_i \cdot f(x_i)}{\sum_i w_i}$$

A weighted average

4. Locally weighed linear regression: can sue linear or polynomial function to fit the data locally.
5. memory-based learner:

pros:easily adapts to new training examples(no retraining needed).
can handle complex decision boundaries and functions

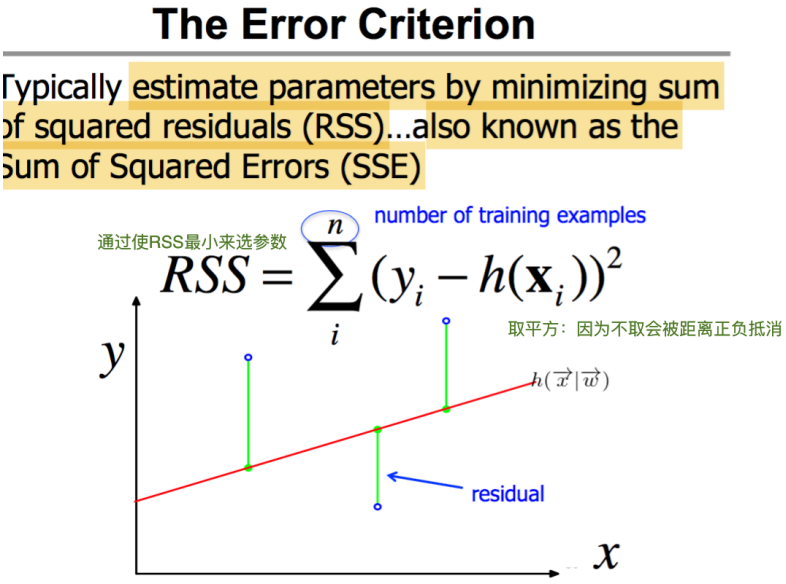
cons: requires retaining all training examples in memory.
slow to evaluate new queries, time grows with dataset size.

summary: 1. lazy
2. local:training data that around the query contributes more
3.non-parametric: 非参数的
4. robust to noise
5. curse of dimensionality

nearest neighbor: the nearest neighbor's label
kNN: the average of k NN's labels
kernel regression:weighted average of all training data's or kNN's labels.
locally wighted(linear) regression: fit a linear function locally.

5. Linear Regression Models

1. **assumption**: the expected value of the response variable y is a linear combination of the k independent attributes.
2. **hypothesis space** is the set of linear functions(hyperplanes):
   h(x) = w0 + w1x1 + w2x2 + … wkxk.
3. **goal**: is to learn a k+1 dimensional vector of weights that define a hyperplane **minimizing an error criterion**.
4. error criterion: rss(sse)



5. polynomial regression
6. bias and variance of an estimator

Bias and Variance of an Estimator

- Let X be a sample from a population specified by a true parameter θ
- Let d=d(X) be an estimator for θ

$$\mathbb{E}[(d-\theta)^2] = \mathbb{E}[(d-\mathbb{E}[d])^2] + (\mathbb{E}[d]-\theta)^2$$

mean square error *variance* *bias²*

increase complexity, bias decreases(a better fit) and variance increases(fit varies more with data)
bias: measures how much h(x) is wrong. **high bias->under fitting**
variance: how much h(x) fluctuate around the expected value as the sample varies. **high variance -> overfitting**

7. summary of linear regression:
    1. well studied
    2. **can handle non-linear situations if formulated properly.**

8. avoid overfitting: simple model, fewer features, dimensionality reduction, more data…

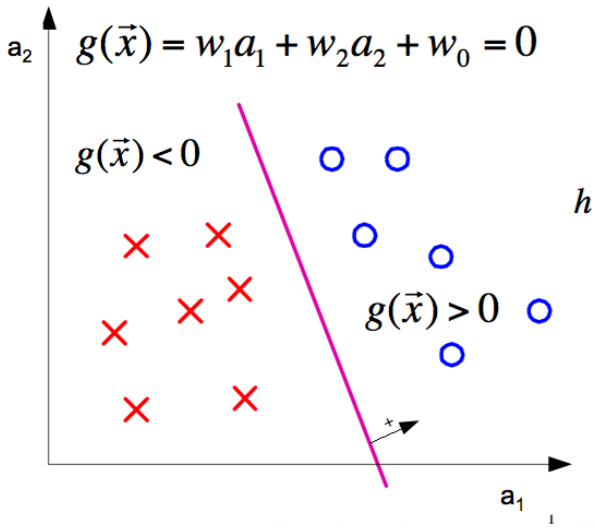
6. linear discriminants

1. linear discriminants: a linear combination of the attributes:

$$g(\vec{x} \mid \vec{w}, w_0) = w_0 + \vec{w}^T \vec{x} = w_0 + \sum_{i=1}^{k} w_i a_i$$

Two-Class Classification

$g(\vec{x}) = 0$ defines a decision boundary that splits the space in two



If a line exists that does this without error, the classes are *linearly separable*

$$h(\vec{x}) = \begin{cases} 1 & \text{if } g(\vec{x}) > 0 \\ -1 & \text{otherwise} \end{cases}$$

2. estimate model parameters

Estimating the model parameters

There are many ways to estimate the model parameters:

- Minimize least squares criteria (i.e. classification via regression – as shown earlier)
- Maximize Fisher linear discriminant criteria
- Minimize perceptron criteria (using numerical optimization techniques, e.g. gradient descent)
梯度
- Many other methods for solving for the inequalities using constrained optimization...

3. classification via regression:
    1. label each class by a number
    2. analytically derive a regression line
    3. round the regression output to the nearest label number
4. gradient descent: follow the gradient to a minimum
5. minimizing sum of squared residuals.
6. in **classification**, the line we learn is **INPUT** to our hypothesis function.

7. **perceptron criteria**:only converges when data is **linearly separable.** restricted to **2-class** discriminant.

感知条件

Perceptron Criteria

指导我们该怎么调整which way to shift

- Perceptron criteria:

$$J(\vec{w}) = -\sum_{n \in \mathcal{M}} \vec{w}^T \vec{x}_n t_n$$

NOTE: augmented to include the threshold w_0

where $t_n \in \{-1, +1\}$ so that we want $\forall n: \ \vec{w}^T \vec{x}_n t_n > 0$
and $\mathcal{M}$ is the set of misclassified training examples.

- We can minimize this criteria to solve for our weight vector **w**
- Restricted to 2-class discriminant
- We can use stochastic gradient descent to solve for this
- *Only converges when data is linearly separable*

8. perceptron:

Perceptron Algorithm

- $\vec{w}$ are parameters
- η is set to 1 (this is fine in this case, since multiplying by a scalar 常量 doesn't affect the decision)
- $\nabla J_n(\vec{w}) = -\vec{x}_n t_n$

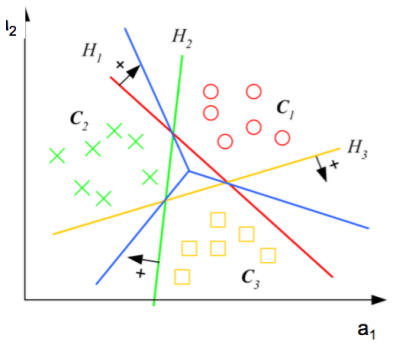
pick a weight, ...
separating hyper points..?

- Algorithm:

begin initialize $\vec{w}, i \leftarrow 0$
  do $i \leftarrow i + 1 \mod m$
    if $\vec{x}_i$ is misclassified by $\vec{w}$
    then $\vec{w} \leftarrow \vec{w} + \vec{x}_i t_i$
  until all examples are properly classified
  return $\vec{w}$
end

9. multi-**class** classification

Multi-class Classification



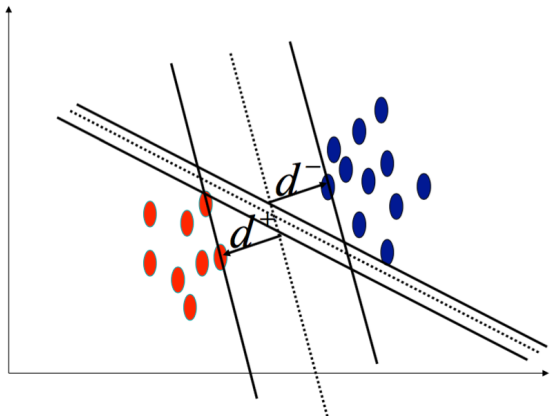
When there are N>2 classes:

- you can classify using N discriminant functions.
- Choose the class with the maximum output
- Geometrically divides feature space into N **convex** decision regions

Choose C_i if $g_i(\vec{x}) = \max_{j=1}^{N} g_j(\vec{x})$

7. support vector machine

1. SVMs are **binary** classifier.
   effective, train quickly on large data sets …
2. main idea:
   1. find optimal hyperplane the split the data into two sets: **maximize margin. m = (d+) + (d-),** maximize (2/||w||)



The margin

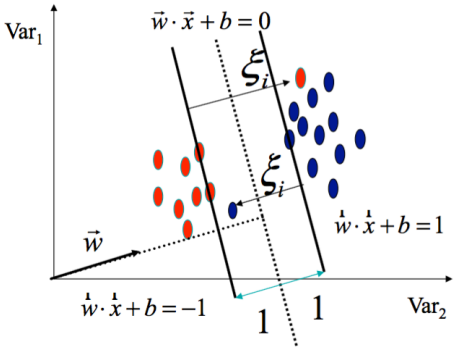
- There is some scale for **w** and for b where the following equation holds.

$$d^+ = d^- = \frac{1}{\| w \|}$$

- Those data points that lie within distance 1/||w|| of the hyperplane are called the **support vectors.**
- The support vectors define two planes parallel to the hyperplane separator

   2. give a guess of w, and search the space w and b's to find the widest margin hyperplane that correctly classifies the data.(math:use lagrangian multipliers to constraint all points are correctly classified)
   3. extend for **non-linearly** separable problems: have a penalty term for misclassifications.

Non-Linearly Separable Data



Allow some instances to fall within the margin, but penalize them

Introduce slack variables ξ_i (one per data point)

The constraints then become…

$$y_i(w \cdot x + b) \geq 1 - \xi_i \quad \forall (x_i, y_i) \in D, \xi_i \geq 0$$

   4. can also linear, soft-margin SVMs.Notice: algorithm does not minimize the number of misclassifications, but the **sum of distances from the margin hyperplanes. soft margin always**

has a solution.soft margin is more robust to outliers.**hard margin requires no parameters.**

5. **map data to high dimensional space** where it is easier to classify with linear decision surfaces. Recall the SVM optimization problem with its **inner product** between data points.
6. **kernels** are functions that **return inner products** between the images of data points. since they are inner products, they can be thought of as **similarity functions.**

两个有多接近的..?

$$K(x_1, x_2) = \langle \Phi(x_1), \Phi(x_2) \rangle$$

7. **kernel trick**: only need to **calculate inner product in the original feature space, no need to explicit mapping Φ.**

3. SVM:
    1. strengths:
        1. well to high dimensional data
        2. non-traditional data can be used as input, like strings, trees, instead of feature vectors.
    2. weakness: tuning svms remains a black art.
    3. choices: what kernel? how much slack you will allow?

8. collaborative filtering

1. user based: You will like item A because users who are similar to you like item A.
2. Item based: You will like item A because you like items that are similar to item A.
3. feature selection: manhattan distance, cosine similarity(完全无关-1, 完全相关1)
4. K Nearest Neighbor Collaborative Filtering :定义similarity measure,z找到最近的k个neighbor，选择平均or最多。
5. cold start problem, missing value
6. choices: similarity measure, how many neighbors, how to weight neighbors chosen, user-based or item based, how to deal with missing data?

9. evaluating hypotheses

1. **IID**: independent and identically distributed
2. **true error**: misclassifications in data distribution D(no way to calculate true error)
3. **sample error**: proportion of examples in data sample set S that h misclassifies.
4. n-fold cross validation
5. student's t-test:
    1. **one sample t-test:** null hypothesis: no significant difference between the sample mean and the population mean
    2. **paired samples t-test:** Paired samples t-tests typically consist of a sample of matched pairs of similar units, or one group of units that has been tested twice
    3. **independent samples t-test:** Are the means of two normally distributed populations equal?
6. pitfalls:

Common t-test pitfalls

- Data is not normally distributed (can't use a t-test)
- Not enough sample points (degrees of freedom)
- Using a paired-samples t-test on data where the samples aren't paired (use independent samples t-test, instead)
- Using a Student's independent samples t-test when the variances of the two sets are different (use Welch's t-test in this case)

7. tests conditions:

Name of test	Samples IID	Gaussian	Paired	Both pops have same variance
Student's T-Test: Paired samples	X	X	X	X
Student's T-Test: Unpaired samples	X	X		X
Welch's T-test	X	X		
Wilcoxon signed-rank test	X		X	
Mann–Whitney *U* test	X			

8. precision vs recall

$$precision \quad p = \frac{tp}{tp + fp}$$

$$recall \quad r = \frac{tp}{tp + fn}$$

$$F - measure \quad F = 2\frac{p \cdot r}{p + r}$$

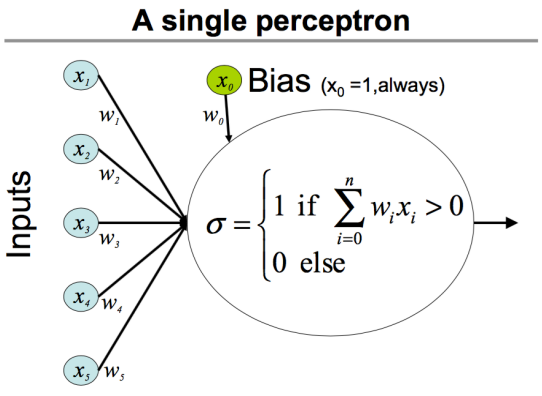
		True Classification	
Machine's Classification		True	False
	True	True positive (tp)	False positive (fp)
	False	False negative (fn)	True negative (tn)

10.Gaussian Mixture Models

1. **expectation maximization(EM):**
    1. after each iteration, the likelihood the model would generate the observed data increases(or at least decrease)
    2. EM always converges to a local optimum
2. **expectation maximization(EM) steps:**
    1. initialize the parameters
    2. E step: calculate the **likelihood** a model with these parameters **generated** the data
    3. M step: update parameters to **increase the likelihood** from E step.
    4. repeat E & M steps until convergence to a local optimum.
3. steps:
    1. choose number of Gaussian components K, K should be much less than the number of data points to avoid overfitting.
    2. expectation step:
4. GMM is a soft version of K-means
    1. K needs to be specified
    2. converges at some local optimal
    3. initialization matters final resutls

11. Neural Networks

1. perceptron



2. No weights can make XOR, linear combination doesn't allow this.
3. learning weights
    1. perceptron training rules
    2. gradient descent
4. perceptron rule vs perceptron rule
    1. learner converges on an answer only if data is linearly separable
    2. can't assign proper error to parent nodes
5. delta rule & linear units
    1. minimizes error even examples are not linear separable
    2. linear units only make linear decision surfaces, can't make XOR
6. multilayer perceptron:

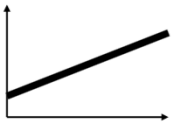
A compromise function

- Perceptron

$$output = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ 0 \text{ else} \end{cases}$$

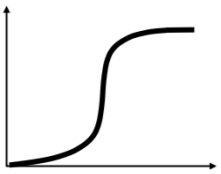
- Linear

$$output = net = \sum_{i=0}^{n} w_i x_i$$



- Sigmoid (Logistic)

$$output = \sigma(net) = \frac{1}{1+e^{-net}}$$



    1. sigmoid: non-linear, **easy to differentiate**.
7. backpropagation algorithm:

反向传播算法

The Backpropagation Algorithm

For each input training example, $\langle \vec{x}, \vec{t} \rangle$

1. Input instance $\vec{x}$ to the network and compute the output o_u for every unit u in the network
2. For each output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1-o_k)(t_k - o_k)$$

3. For each hidden unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1-o_h) \sum_{k \in outputs} w_{hk}\delta_k$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \eta \delta_k x_{ji}$$

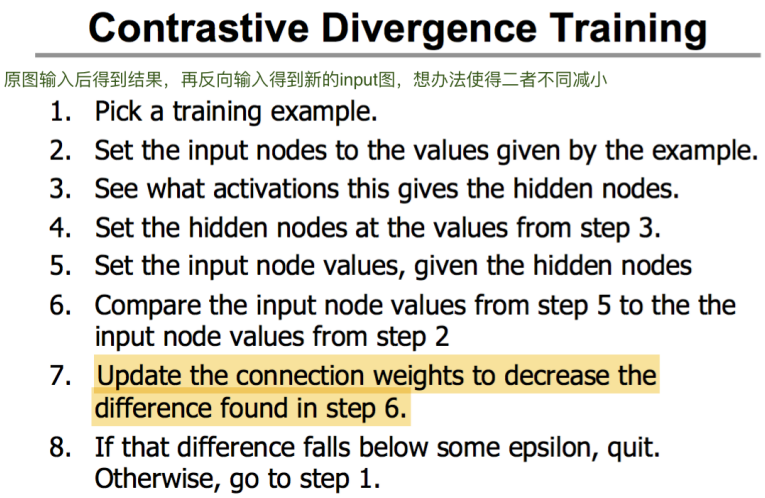
不确定

（1）将训练集数据输入
的输入层，经过隐藏层
后达到输出层并输出结
这是ANN的前向传播过

（2）由于ANN的输出
实际结果有误差，则计
计值与实际值之间的误
并将该误差从输出层向
层反向传播，直至传播
入层；

（3）在反向传播的过
根据误差调整各种参数
值；不断迭代上述过程
至收敛。

8. multi-layer perceptrons
    1. models that resemble nonlinear regression
    2. useful to model nonlinearly separable spaces
    3. can learn arbitrary boolean functions
    4. use **gradient descent(must be differentiable)**, susceptible to being stuck in local minima
9. restricted boltzman machine(RBM):
    1. 2 layers(hidden & input) of boolean nodes only connected to the other layer, and fully connected
    2. contrastive divergence training



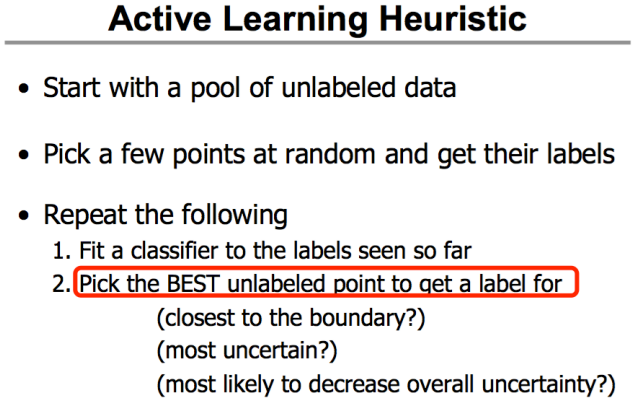
Contrastive Divergence Training

原图输入后得到结果，再反向输入得到新的input图，想办法使得二者不同减小

1. Pick a training example.
2. Set the input nodes to the values given by the example.
3. See what activations this gives the hidden nodes.
4. Set the hidden nodes at the values from step 3.
5. Set the input node values, given the hidden nodes
6. Compare the input node values from step 5 to the the input node values from step 2
7. Update the connection weights to decrease the difference found in step 6.
8. If that difference falls below some epsilon, quit. Otherwise, go to step 1.

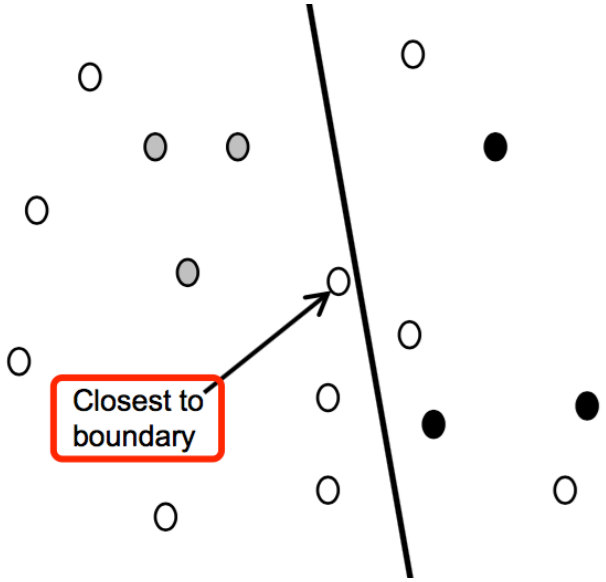
12. Active Learning

1. **concept learning:** defining a function that specifies which elements are in the concept set
2. Big idea: if we pick the right examples to label, we can learn the concept from only a few labeled examples
3. steps:



Active Learning Heuristic

- Start with a pool of unlabeled data
- Pick a few points at random and get their labels
- Repeat the following
    1. Fit a classifier to the labels seen so far
    2. Pick the BEST unlabeled point to get a label for
        (closest to the boundary?)
        (most uncertain?)
        (most likely to decrease overall uncertainty?)



4. hypothesis search
    1. **query selection strategies:**Pick the unlabeled instance that would cause the greatest change to the model, if we knew its label
    2. **density weighting selections**:

Density Weighting Selections

Pick instances that are both "informative" and "representative"

"informative" = score highly on one of the query evaluation measures discussed earlier

"representative" = inhabit dense regions of the input space

Example Density Weighting