CS9840
Learning and Computer Vision
Prof. Olga Veksler

Lecture 3

Linear Machines

Information Theory (a little BIT)

Last Time: Supervised Learning

- Training samples (or examples) $X^1, X^2, \ldots X^n$
- Each example is typically multi-dimensional
  - $X^i_1, X^i_2, \ldots, X^i_d$ are typically called *features*, X^i is sometimes called a *feature vector*
  - **How many features and which features do we take?**
- Know desired output for each example (labeled samples) $Y^1, Y^2, \ldots Y^n$
  - This learning is supervised ("teacher" gives desired outputs).
  - Y^i are often one-dimensional, but can be multidimensional

Today

- Linear Classifier
- Mutual Information
- Next time:
  - paper: "Object Recognition with Informative Features and Linear Classification" by M. Naquet and S. Ullman
    - Ignore section of tree-augmented network

Last Time: Supervised Learning

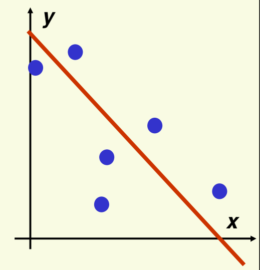
- Wish to design a *machine* f(X,W) s.t. f(X,W) = true output value at X
  - In classification want f(X,W) = label of X
  - **How do we choose f?**
    - when we choose a particular f, we are making implicit assumptions about our problem
  - W is typically multidimensional vector of weights (also called *parameters*) which enable the machine to "learn"
    - $W = [w_1, w_2, \ldots w_k]$

Training and Testing

- There are 2 phases, training and testing
  - Divide all labeled samples $X^1, X^2, \ldots X^n$ into 2 sets, *training* set and *testing* set
  - Training phase is for "teaching" our machine (finding optimal weights W)
  - Testing phase is for evaluating how well our machine works on unseen examples
- Training phase
  - Find the weights W s.t. $f(X^i,W) = Y^i$ "as much as possible" for the *training* samples X^i
  - "as much as possible" needs to be defined
  - Training can be quite complex and time-consuming

Linear Machine, Continuous Y

- $f(X,W) = w_0+\Sigma_{i=1,2,...d}\, w_i x_i$
  - w_0 is called bias
- In vector form, if we let $X = (1,x_1,x_2,\ldots,x_d)$, then $f(X,W) = W^T X$
  - notice abuse of notation, I made $X=[1\ X]$
- This is standard linear regression (line fitting)
  - assume $L(X^i,Y^i,W) = \|\, f(X^i,W) - Y^i \,\|^2$
  - optimal W can be found by solving linear system of equations $W^* = [\Sigma X^i (X^i)^T]^{-1}\, \Sigma Y^i X^i$

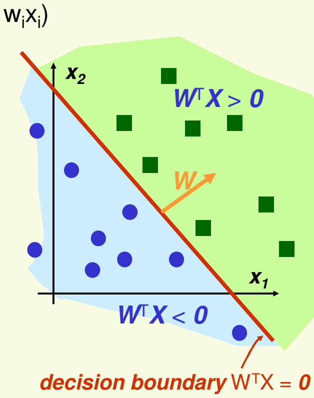


Loss Function

- How do we quantify what it means for the machine f(X,W) do well in the training and testing phases?
- f(X,W) has to be "close" to the true output on X
- Define Loss (or Error) function L
  - This is up to the designer (that is you)
- Typically first define per-sample loss $L(X^i,Y^i,W)$
  - Some examples:
    - for classification, $L(X^i,Y^i,W) = \mathbf{I}[f(X^i,W) \neq Y^i]$, where $\mathbf{I}[\text{true}] = 1$, $\mathbf{I}[\text{false}] = 0$
      - we just care if the sample has been classified correctly
    - For continuous Y, $L(X^i,Y^i,W) = \|\, f(X^i,W) - Y^i \,\|^2$,
      - how far is the estimated output from the correct one?
- Then loss function $L = \Sigma_i\, L(X^i,Y^i,W)$
  - Number of missclassified example for classification
  - Sum of distances from the estimated output to the correct output

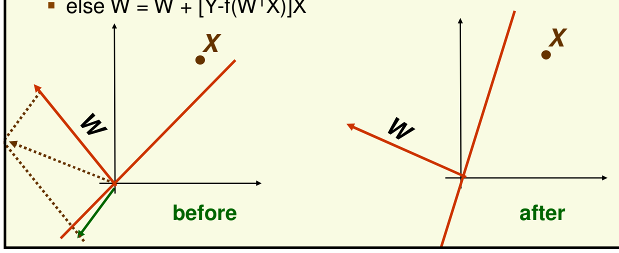
Linear Machine: binary Y

- $f(X,W) = \text{sign}(w_0+\Sigma_{i=1,2,...d}\, w_i x_i)$
  - sign(positive) = 1, sign(negative) = -1
  - w_0 is called bias
- In vector form, if we let $X = (1,x_1,x_2,\ldots,x_d)$ then $f(X,W) = \text{sign}(W^T X)$



Perceptron Learning Procedure (Rosenblatt 1957)

- $f(X,W) = \text{sign}(w_0+\Sigma_{i=1,2,...d}\, w_i x_i)$
- Let $L(X^i,Y^i,W) = \mathbf{I}[f(X^i,W) \neq Y^i]$. How do we learn W?
- A solution:
- Iterate over all training samples
  - if f(X,W)=Y (correct label), do nothing
  - else $W = W + [Y-f(W^TX)]X$



Optimization

- Need to minimize a function of many variables

$$J(\mathbf{x}) = J(x_1,\dots,x_d)$$

- We know how to minimize $J(\mathbf{x})$
  - Take partial derivatives and set them to zero

$$\begin{bmatrix} \frac{\partial}{\partial x_1} J(\mathbf{x}) \\ \vdots \\ \frac{\partial}{\partial x_d} J(\mathbf{x}) \end{bmatrix} = \underbrace{\nabla J(\mathbf{x})}_{\text{gradient}} = 0$$

- However solving analytically is not always easy
  - Would you like to solve this system of nonlinear equations?

$$\begin{cases} \sin(x_1^2 + x_2^3) + e^{x_4^2} = 0 \\ \cos(x_1^2 + x_2^3) + \log(x_5^3)^{x_4^2} = 0 \end{cases}$$

  - Sometimes it is not even possible to write down an analytical expression for the derivative, we will see an example later today

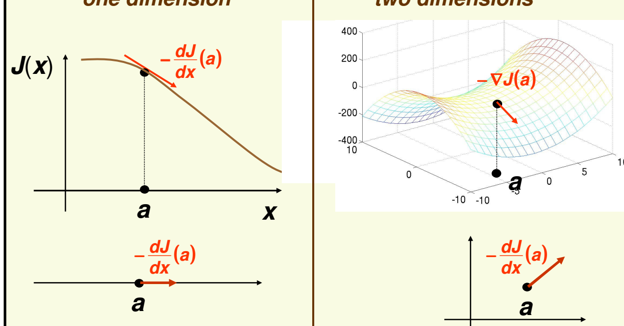
Perceptron Learning Procedure (Rosenblatt 1957)

- Amazing fact: If the samples are linearly separable, the perceptron learning procedure will converge to a solution (separating hyperplane) in a finite amount of time
- Bad news: If the samples are not linearly separable, the perceptron procedure will not terminate, it will go on looking for a solution which does not exist!
- For most interesting problems the samples are not linearly separable
- Is there a way to learn W in non-separable case?
  - Remember, it's ok to have training error, so we don't have to have "perfect" classification

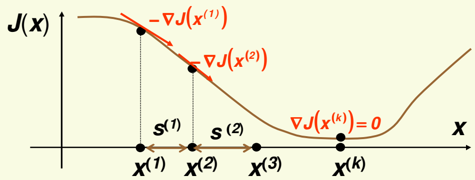
Optimization: Gradient Descent

- Gradient $\nabla J(\mathbf{x})$ points in direction of steepest increase of $J(\mathbf{x})$, and $-\nabla J(\mathbf{x})$ in direction of steepest decrease

one dimension *two dimensions*



Optimization: Gradient Descent



Gradient Descent for minimizing any function $J(x)$

set $k = 1$ and $x^{(1)}$ to some initial guess for the weight vector

while $\eta^{(k)} \left| \nabla J\left(x^{(k)}\right) \right| > \varepsilon$

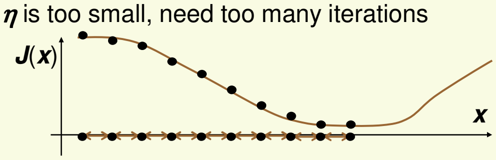
choose learning rate $\eta^{(k)}$

$x^{(k+1)} = x^{(k)} - \eta^{(k)} \nabla J(x)$ ***(update rule)***

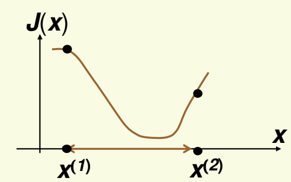
$k = k + 1$

Optimization: Gradient Descent

- Main issue: how to set parameter η (***learning rate***)
- If η is too small, need too many iterations

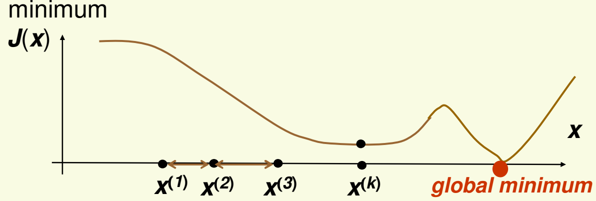


- If η is too large may overshoot the minimum and possibly never find it (if we keep overshooting)



Optimization: Gradient Descent

- Gradient descent is guaranteed to find only a local minimum



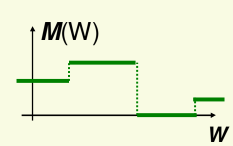
- Nevertheless gradient descent is very popular because it is simple and applicable to any differentiable function

"Optimal" W with Gradient Descent

- $f(X,W) = \text{sign}(w_0 + \Sigma_{i=1,2,\dots d}\, w_i x_i)$
- If we let $L(X^i,Y^i,W) = \mathbf{I}[f(X^i,W) \neq Y^i]$, then $L(W)$ is the number of missclassified examples
- Let M be the set of examples misclassified by W

$$M(W) = \left\{ \textit{sample } X^i \textit{ s.t. } W^T X^i \neq Y^i \right\}$$

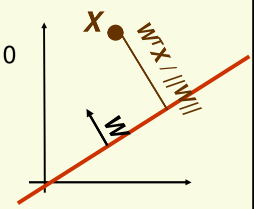
- Then $L(W) = |M(W)|$, the size of $M(W)$
- $L(W)$ is piecewise constant, gradient descent is useless



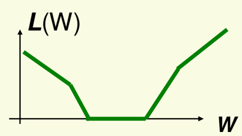
"Optimal" W with Gradient Descent

- Better choice:

$$L(W) = \sum_{X^i \in M} \left(-W^T X^i\right) Y^i$$



- If X^i is misclassified, $(W^T X^i) Y^i \leq 0$
- Thus $L(W,X^i,Y^i) \geq 0$
- $L(W,X^i,Y^i)$ is proportional to the distance of misclassified example to the decision boundary
- $L(W)=\Sigma L(W,X^i,Y^i)$ is piecewise linear and thus suitable for gradient decent



Single Sample Rule

- Thus *gradient decent single sample rule* for L(W) is:

$$W^{(k+1)} = W^{(k)} + \eta^{(k)}(XY)$$

  - apply for any sample X misclassified by $W^{(k)}$
  - must have a consistent way of visiting samples

Batch Rule

$$L(W, X^i, Y^i) = \sum_{X \in M} \left(-W^T X\right) Y$$

- Gradient of L is $\nabla L(W) = \sum_{X \in M} (-X) Y$
  - M are samples misclassified by W
  - It is not possible to solve $\nabla L(W) = 0$ analytically
- Update rule for gradient descent: $x^{(k+1)} = x^{(k)} - \eta^{(k)} \nabla J(x)$
- Thus *gradient decent batch update rule* for L(W) is:

$$W^{(k+1)} = W^{(k)} + \eta^{(k)} \sum_{Y \in M} XY$$

- It is called batch rule because it is based on all misclassified examples

Convergence

- If classes are linearly separable, and $\eta^{(k)}$ is fixed to a constant, i.e. $\eta^{(1)} = \eta^{(2)} = \ldots = \eta^{(k)} = c$ (*fixed learning rate*)
  - *both single sample and batch rules converge to a correct solution* (could be any W in the solution space)
- If classes are not linearly separable:
  - Single sample algorithm does not stop, it keeps looking for solution which does not exist
  - However by choosing appropriate learning rate, heuristically stop algorithm at hopefully good stopping point

$$\eta^{(k)} \to 0 \;\; as \;\; k \to \infty$$

  - for example, $\eta^{(k)} = \dfrac{\eta^{(1)}}{k}$
  - for this learning rate convergence in the linearly separable case can also be proven

Learning by Gradient Descent

- Suppose we suspect that the machine has to have functional form f(X,W), not necessarily linear
- Pick differentiable per-sample loss function $L(X^i,Y^i,W)$
- We need to find W that minimizes $L = \Sigma_i L(X^i,Y^i,W)$
- Use gradient-based minimization:
  - Batch rule: $W = W - \eta \nabla L(W)$
  - Or single sample rule: $W = W - \eta \nabla L(X^i,Y^i,W)$

Information theory

- Information Theory regards information as only those symbols that are uncertain to the receiver
  only infrmatn esentil to understnd mst b tranmitd
- Shannon made clear that uncertainty is the very commodity of communication
- The amount of information, or uncertainty, output by an information source is a measure of its entropy
- In turn, a source's entropy determines the amount of bits per symbol required to encode the source's information
- Messages are encoded with strings of 0 and 1 (bits)

Important Questions

- How do we choose the feature vector X?
- How do we split labeled samples into training/testing sets?
- How do we choose the machine f(X,W)?
- How do we choose the loss function $L(X^i,Y^i,W)$?
- How do we find the optimal weights W?

Information theory

- Suppose we toss a **fair** die with 8 sides
  - need 3 bits to transmit the results of each toss
  - 1000 throws will need 3000 bits to transmit
- Suppose the die is biased
  - side A occurs with probability 1/2, chances of throwing B are 1/4, C are 1/8, D are 1/16, E are 1/32, F 1/64, G and H are 1/128
  - Encode A= 0, B = 10, C = 110, D = 1110,…, so on until G = 1111110, H = 1111111
  - We need, on average, 1/2+2/4+3/8+4/16+5/32+6/64+7/128+7/128 = 1.984 bits to encode results of a toss
  - 1000 throws require 1984 bits to transmit
  - Less bits to send = less “information”
  - Biased die tosses contain less “information” than unbiased die tosses (know in advance biased sequence will have a lot of A’s)
  - What’s the number of bits in the best encoding?
- Extreme case: if a die always shows side A, a sequence of 1,000 tosses has no information, 0 bits to encode

Information theory

- if a die is fair (any side is equally likely, or uniform distribution), for any toss we need log(8) = 3 bits
- Suppose any of n events is equally likely (uniform distribution)
  - P(x) = 1/n, therefore -log P = -log(1/n) = log n
- In the “good” encoding strategy for our biased die example, every side x has -log p(x) bits in its code
- Expected number of bits is

$$-\sum_x p(x)\log p(x)$$

Conditional Entropy of X given Y

$$H[x\mid y]=\sum_{x,y} p(x,y)\log\frac{1}{p(x\mid y)}=-\sum_{x,y} p(x,y)\log p(x\mid y)$$

- Measures average uncertainty about x when y is known
- Property:
  - H[x] ≥ H[x|y], which means after seeing new data (y), the uncertainty about x is not increased, on average

Shannon’s Entropy

$$H[p(x)]=-\sum_x p(x)\,log\;p(x)=\sum_x p(x)\,log\frac{1}{p(x)}$$

- How much randomness (or uncertainty) is there in the value of signal x if it has distribution p(x)
  - For uniform distribution (every event is equally likely), H[x] is maximum
  - If p(x) = 1 for some event x, then H[x] = 0
  - Systems with one very common event have less entropy than systems with many equally probable events
- Gives the expected length of optimal encoding (in binary bits) of a message following distribution p(x)
  - doesn’t actually give this optimal encoding

Mutual Information of X and Y

$$I[x,y]=H(x)-H(x\mid y)$$

- Measures the average reduction in uncertainty about x after y is known
- or, equivalently, it **measures the amount of information that y conveys about x**
- Properties
  - I(x,y) = I(y,x)
  - I(x,y) ≥ 0
  - If x and y are independent, then I(x,y) = 0
  - I(x,x) = H(x)

MI for Feature Selection

$$I[x,c] = H(c) - H(c / x)$$

- Let x be a proposed feature and c be the class
- If I[x,c] is high, we can expect feature x be good at predicting class c