Today

• Optimization with Gradient descent

• Linear Classifier
  • Two classes
  • Multiple classes
  • Perceptron Criterion Function
    • Batch perceptron rule
    • Single sample perceptron rule
  • Minimum Squared Error (MSE) rule
    • Pseudoinverse

• Generalized Linear Classifier

• Gradient Descent Based learning
Optimization

- How to minimize a function of a single variable
  \[ J(x) = (x-5)^2 \]

- From calculus, take derivative, set it to 0
  \[ \frac{d}{dx} J(x) = 0 \]

- Solve the resulting equation
  - maybe easy or hard to solve

- Example above is easy:
  \[ \frac{d}{dx} J(x) = 2(x - 5) = 0 \Rightarrow x = 5 \]
Optimization

- How to minimize a function of many variables
  \[ J(x) = J(x_1, \ldots, x_d) \]

- From calculus, take partial derivatives, set them to 0

\[
\begin{bmatrix}
\frac{\partial}{\partial x_1} J(x) \\
\vdots \\
\frac{\partial}{\partial x_d} J(x)
\end{bmatrix} = \nabla J(x) = 0
\]

- Solve the resulting system of \(d\) equations

- It may not be possible to solve the system of equations above analytically
Optimization: Gradient Direction

- Gradient $\nabla J(x)$ points in the direction of steepest increase of function $J(x)$
- $-\nabla J(x)$ points in the direction of steepest decrease
Gradient Direction in 2D

- $J(x_1, x_2) = (x_1 - 5)^2 + (x_2 - 10)^2$

- $\frac{\partial}{\partial x_1} J(x) = 2(x_1 - 5)$

- $\frac{\partial}{\partial x_2} J(x) = 2(x_2 - 10)$

- Let $a = [10, 5]$

- $- \frac{\partial}{\partial x_1} J(a) = -10$

- $- \frac{\partial}{\partial x_2} J(a) = 10$
Gradient Descent: Step Size

- \( J(x_1, x_2) = (x_1 - 5)^2 + (x_2 - 10)^2 \)
- Which step size to take?
- Controlled by parameter \( \alpha \)
  - called learning rate
- From previous example:
  - \( a = [10 \ 5] \)
  - \( -\nabla J(a) = [-10 \ 10] \)
- Let \( \alpha = 0.2 \)
- \( a - \alpha \nabla J(a) = [10 \ 5] + 0.2 [-10 \ 10] = [8 \ 7] \)
- \( J(10, 5) = 50 \)
- \( J(8, 7) = 18 \)
Gradient Descent Algorithm

\[ k = 1 \]
\[ x^{(1)} = \text{any initial guess} \]
choose \( \alpha, \varepsilon \)
while \( \alpha \| \nabla J(x^{(k)}) \| > \varepsilon \)
\[ x^{(k+1)} = x^{(k)} - \alpha \nabla J(x^{(k)}) \]
\[ k = k + 1 \]
Gradient Descent: Local Minimum

- Not guaranteed to find global minimum
  - gets stuck in local minimum

- Still gradient descent is very popular because it is simple and applicable to any differentiable function
How to Set Learning Rate $\alpha$?

- If $\alpha$ too small, too many iterations to converge.

- If $\alpha$ too large, may overshoot the local minimum and possibly never even converge.

- It helps to compute $J(x)$ as a function of iteration number, to make sure we are properly minimizing it.
How to Set Learning Rate $\alpha$?

- As we approach local minimum, often gradient gets smaller.
- Step size may get smaller automatically, even if $\alpha$ is fixed.
- So it may be unnecessary to decrease $\alpha$ over time in order not to overshoot a local minimum.

![Graph showing how to set learning rate](image)
Variable Learning Rate

- If desired, can change learning rate $\alpha$ at each iteration

\[ k = 1 \]
\[ x^{(1)} = \text{any initial guess} \]
\[ \text{choose } \alpha, \varepsilon \]
\[ \textbf{while } \alpha \| \nabla J(x^{(k)}) \| > \varepsilon \]
\[ x^{(k+1)} = x^{(k)} - \alpha \nabla J(x^{(k)}) \]
\[ k = k + 1 \]
Variable Learning Rate

- Usually don’t keep track of all intermediate solutions

\[ k = 1 \]
\[ x^{(1)} = \text{any initial guess} \]
choose \( \alpha, \varepsilon \)
while \( \alpha \| \nabla J(x^{(k)}) \| > \varepsilon \)
\[ x^{(k+1)} = x^{(k)} - \alpha \nabla J(x^{(k)}) \]
\[ k = k + 1 \]
Advanced Optimization Methods

- There are more advanced gradient-based optimization methods
- Such as conjugate gradient
  - automatically pick a good learning rate $\alpha$
  - usually converge faster
  - however more complex to understand and implement
- in Matlab, use `fminunc` for various advanced optimization methods
Last Time: Supervised Learning

- Training samples (or examples)
  \[ x^1, x^2, \ldots, x^n \]

- Each example is typically multi-dimensional
  - \( x^i = [x^i_1, x^i_2, \ldots, x^i_d] \)
  - \( x^i \) is often called a *feature vector*

- Know desired output for each example
  \[ y^1, y^2, \ldots, y^n \]

- regression:  continuous \( y \)
- classification: finite \( y \)
Last Time: Supervised Learning

• Wish to design a *machine* \( f(x,w) \) s.t.
  \[
  f(x,w) = y
  \]

• How do we choose \( f \)?
  • last lecture studied kNN classifier
  • this lecture in on liner classifier
  • many other choices

• \( W \) is typically multidimensional vector of weights (also called *parameters*)
  \[
  w = [w_1,w_2,...,w_k]
  \]

• By modifying \( w \), the machine “learns”
Training and Testing Phases

- Divide all labeled samples $x^1, x^2, ..., x^n$ into training and test sets

- Training phase
  - Uses training samples
  - goal is to “teach” the machine
  - find weights $w$ s.t. $f(x^i, w) = y^i$ “as much as possible”
    - “as much as possible” needs to be defined

- Testing phase
  - Uses only test samples
  - for evaluating how well our machine works on unseen examples
Loss Function

- How to quantify “\( f(x^i, w) = y^i \) as much as possible”?
- \( f(x, w) \) has to be “close” to the true output \( y \)
- Define Loss (or Error, or Criterion) function \( L \)
- Typically first define per-sample loss \( L(x^i, y^i, w) \)
  - for classification, \( L(x^i, y^i, w) = I[f(x^i, w) \neq y^i] \)
    - where \( I[true] = 1, I[false] = 0 \)
  - for regression, \( L(x^i, y^i, w) = \| f(x^i, w) - y^i \|^2 \)
    - how far is the estimated output from the correct one?
- Then loss function \( L = \sum_i L(x^i, y^i, w) \)
  - classification: counts number of misclassified examples
  - regression: sums distances to the correct output
Linear Machine: Regression

- \( f(x, w) = w_0 + \sum_{i=1,2,...,d} w_i x_i \)
- In vector notation
  - \( x = [x_1, x_2, ..., x_d] \)
  - \( f(x, w) = w_0 + w^T x \)
- 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 a system of linear equations
  \[
  w^* = [\sum x^i (x^i)^T]^{-1} \sum y^i x^i
  \]
Linear Machine: Classification

- First consider the two-class case
- We choose the following encoding:
  - \( y = 1 \) for the first class
  - \( y = -1 \) for the second class
- Linear classifier
  - \(-\infty \leq w_0 + x_1 w_1 + \ldots + x_d w_d \leq \infty\)
  - we need \( f(x, w) \) to be either +1 or -1
  - let \( g(x, w) = w_0 + x_1 w_1 + \ldots + x_d w_d = w_0 + w^t x \)
  - let \( f(x, w) = \text{sign}(g(x, w)) \)
    - 1 if \( g(x, w) \) is positive
    - -1 if \( g(x, w) \) is negative
  - other choices for \( g(x, w) \) are also used
  - \( g(x, w) \) is called the **discriminant function**
Linear Classifier: Decision Boundary

- $f(x, w) = \text{sign}(g(x, w)) = \text{sign}(w_0 + x_1w_1 + \ldots + x_d w_d)$
- Decision boundary is linear
- Find the best linear boundary to separate two classes
- Search for best $w = [w_0, w_1, \ldots, w_d]$ to minimize training error
More on Linear Discriminant Function (LDF)

- LDF: \( g(x, w) = w_0 + x_1 w_1 + \ldots + x_d w_d \)
- Written using vector notation \( g(x) = w^T x + w_0 \)

- Decision boundary: \( g(x) = 0 \)
- Decision region for class 1: \( g(x) > 0 \)
- Decision region for class 2: \( g(x) < 0 \)

- Weight vector
- Bias or threshold
More on Linear Discriminant Function (LDF)

• Decision boundary: \( g(x,w) = w_0 + x_1w_1 + ... + x_dw_d = 0 \)
• This is a hyperplane, by definition
  • a point in 1D
  • a line in 2D
  • a plane in 3D
  • a hyperplane in higher dimensions
Multiple Classes

• We have \( m \) classes
• Define \( m \) linear discriminant functions
  \[ g_i(x) = w_i^t x + w_{i0} \] for \( i = 1, 2, \ldots, m \)
• Assign \( x \) to class \( i \) if
  \[ g_i(x) > g_j(x) \text{ for all } j \neq i \]
• Let \( R_i \) be the decision region for class \( i \)
  • That is all examples in \( R_i \) get assigned class \( i \)
Multiple Classes

• Can be shown that decision regions are convex
• In particular, they must be spatially contiguous
Failure Cases for Linear Classifier

- Thus applicability of linear classifiers is limited to mostly unimodal distributions, such as Gaussian
- Not unimodal data
- Need non-contiguous decision regions
- Linear classifier will fail
Linear Classifiers

• Linear classifiers give simple decision boundary
  • try simpler models first

• Linear classifiers are optimal for certain type of data
  • Gaussian distributions with equal covariance

• May not be optimal for other data distributions, but they are very simple to use
Fitting Parameters $w$

- Linear discriminant function $g(x) = w^tx + w_0$

- Can rewrite it $g(x) = \begin{bmatrix} w_0 & w^t \end{bmatrix} \begin{bmatrix} 1 \\ x \end{bmatrix} = a^tz = g(z)$

- $z$ is called augmented feature vector

- new problem equivalent to the old $g(z) = a^tz$

\[
\begin{bmatrix}
w_0 \\
w_1 \\
\vdots \\
w_d
\end{bmatrix}
\begin{bmatrix}
1 \\
x_1 \\
\vdots \\
x_d
\end{bmatrix}
\]
Augmented Feature Vector

- Feature augmenting is done to simplify notation.
- From now on we assume that we have augmented feature vectors.
  - given samples $x_1, \ldots, x_n$ convert them to augmented samples $z_1, \ldots, z_n$ by adding a new dimension of value 1.
- $g(z) = a^Tz$
Training Error

• For the rest of the lecture, assume we have 2 classes
• Samples $z^1, \ldots, z^n$ some in class 1, some in class 2
• Use these samples to determine weights $a$ in the discriminant function $g(z) = a^t z$
• Want to minimize number of misclassified samples
• Recall that
  \[
  \begin{cases}
  g(z^i) > 0 \Rightarrow \text{class 1} \\
  g(z^i) < 0 \Rightarrow \text{class 2}
  \end{cases}
  \]
• Thus training error is 0 if
  \[
  \begin{cases}
  g(z^i) > 0 \quad \forall z^i \text{ class 1} \\
  g(z^i) < 0 \quad \forall z^i \text{ class 2}
  \end{cases}
  \]
• Thus training error is 0 if\
\[
\begin{cases}
  a^t z^i > 0 & \forall z^i \text{ class 1} \\
  a^t z^i < 0 & \forall z^i \text{ class 2}
\end{cases}
\]
• Equivalently, training error is 0 if\
\[
\begin{cases}
  a^t z^i > 0 & \forall z^i \text{ class 1} \\
  a^t (-z^i) > 0 & \forall z^i \text{ class 2}
\end{cases}
\]
• Problem “normalization”:
  1. replace all examples $z^i$ from class 2 by $-z^i$
  2. seek weights $a$ s.t. $a^t z^i > 0$ for $\forall z^i$
• If exists, such $a$ is called a separating or solution vector
• Original samples $x^1, \ldots, x^n$ can also be linearly separated
Effect of Normalization

before normalization

seek a hyperplane that separates samples from different categories

after normalization

seek hyperplane that puts normalized samples on the same (positive) side
Solution Region

- Find weight vector $a$ s.t. for all samples $z^1, ..., z^n$

$$a^t z^i = \sum_{k=0}^{d} a_k z^i_d > 0$$

- If there is one such $a$, then there are infinitely many $a$
Solution region: the set of all possible solutions for $a$
Minimum Squared Error Optimization (MSE)

• Linear Regression is a very well understood problem
• Problem is not regression, but let’s convert to regression!

\[ a^T z^i > 0 \text{ for all samples } z^i \]
solve system of linear inequalities

\[ a^T z^i = b_i \text{ for all samples } z^i \]
solve system of linear equations

• MSE procedure
  • choose positive constants \( b_1, b_2, ..., b_n \)
  • try to find weight vector \( a \) s.t. \( a^T z^i = b_i \) for all samples \( z^i \)
  • if succeed, then \( a \) is a solution because \( b_i \)’s are positive
  • consider all the samples (not just the misclassified ones)
By setting $\mathbf{a}^\top \mathbf{z}_i = \mathbf{b}_i$, we expect $\mathbf{z}_i$ to be at a relative distance $\mathbf{b}_i$ from the separating hyperplane.

Thus $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ are expected relative distances of examples from the separating hyperplane.

Should make $\mathbf{b}_i$ small if sample $\mathbf{i}$ is expected to be near separating hyperplane, and make $\mathbf{b}_i$ larger otherwise.

In the absence of any such information, there are good reasons to set

$$\mathbf{b}_1 = \mathbf{b}_2 = \ldots = \mathbf{b}_n = 1$$
MSE: Matrix Notation

• Solve system of $n$ equations

\[
\begin{align*}
\mathbf{a}^T \mathbf{z}^1 &= \mathbf{b}_1 \\
&\vdots \\
\mathbf{a}^T \mathbf{z}^n &= \mathbf{b}_n 
\end{align*}
\]

• Using matrix notation:

\[
\begin{bmatrix}
\mathbf{z}_0^1 & \mathbf{z}_1^1 & \ldots & \mathbf{z}_d^1 \\
\mathbf{z}_0^2 & \mathbf{z}_1^2 & \ldots & \mathbf{z}_d^2 \\
\vdots & \vdots & \ddots & \vdots \\
\mathbf{z}_0^n & \mathbf{z}_1^n & \ldots & \mathbf{z}_d^n 
\end{bmatrix}
\begin{bmatrix}
\mathbf{a}_0 \\
\mathbf{a}_1 \\
\vdots \\
\mathbf{a}_d 
\end{bmatrix}
= 
\begin{bmatrix}
\mathbf{b}_1 \\
\mathbf{b}_2 \\
\vdots \\
\mathbf{b}_n 
\end{bmatrix}
\]

• Solve a linear system $\mathbf{Z}\mathbf{a} = \mathbf{b}$
MSE: Approximate Solution

• Typically \( Z \) is overdetermined
  - more rows (examples) than columns (features)

\[
\begin{bmatrix}
Z \\
a
\end{bmatrix} = b
\]

• No exact solution for \( Za = b \) in this case

• Find an approximate solution \( a \), that is \( Za \approx b \)
  - approximate solution \( a \) does not necessarily give a separating hyperplane in the separable case
  - but hyperplane corresponding to an approximate \( a \) may still be a good solution

• Least Squares Solution: \( a = (Z^tZ)^{-1} Z^t b \)
### MSE: Example

- Class 1: (6, 9), (5, 7)
- Class 2: (5, 9), (0, 4)
- Add extra feature and "normalize"

\[
\begin{align*}
\mathbf{z}^1 &= \begin{bmatrix} 1 \\ 6 \\ 9 \end{bmatrix} & \mathbf{z}^2 &= \begin{bmatrix} 1 \\ 5 \\ 7 \end{bmatrix} & \mathbf{z}^3 &= \begin{bmatrix} -1 \\ -5 \\ -9 \end{bmatrix} & \mathbf{z}^4 &= \begin{bmatrix} -1 \\ 0 \\ -4 \end{bmatrix}
\end{align*}
\]

- \[\mathbf{Z} = \begin{bmatrix} 1 & 6 & 9 \\ 1 & 5 & 7 \\ -1 & -5 & -9 \\ -1 & 0 & -4 \end{bmatrix}\]
MSE: Example

- Choose \( b = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \)

- Use \( a = \text{Z}\backslash b \) to solve in Matlab

\[
a = \begin{bmatrix} 2.7 \\ 1.0 \\ -0.9 \end{bmatrix}
\]

- Note \( a \) is an approximation since \( \text{Z}a = \begin{bmatrix} 0.4 \\ 1.3 \\ 0.6 \end{bmatrix} \neq \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \)

- Gives a separating hyperplane since \( \text{Z}a > 0 \)
MSE: Example

- Class 1: (6 9), (5 7)
- Class 2: (5 9), (0 10)
- One example is far compared to others from separating hyperplane

\[
\begin{align*}
z_1^1 &= \begin{bmatrix} 1 \\ 6 \\ 9 \end{bmatrix} & z_2^1 &= \begin{bmatrix} 1 \\ 5 \\ 7 \end{bmatrix} & z_3 &= \begin{bmatrix} -1 \\ -5 \\ -9 \end{bmatrix} & z_4 &= \begin{bmatrix} -1 \\ 0 \\ -10 \end{bmatrix}
\end{align*}
\]

\[
Z = \begin{bmatrix}
1 & 6 & 9 \\
1 & 5 & 7 \\
-1 & -5 & -9 \\
-1 & 0 & -10 \\
\end{bmatrix}
\]
MSE: Example

- Choose \( \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \)
- Solve \( \mathbf{a} = \mathbf{Z} \backslash \mathbf{b} = \begin{bmatrix} 3.2 \\ 0.2 \\ -0.4 \end{bmatrix} \)
- \( \mathbf{Za} = \begin{bmatrix} 0.2 & 3.2 \\ 0.9 & 0.2 \\ -0.04 & 3.2 \\ 1.16 & 0.2 \end{bmatrix} \neq \begin{bmatrix} 1 \\ 1 \end{bmatrix} \)
- Does not give a separating hyperplane since \( \mathbf{a}^T \mathbf{z}^3 < 0 \)
MSE: Problems

- MSE wants all examples to be at the same distance from the separating hyperplane.
- Examples that are “too right”, i.e. too far from the boundary cause problems.
- No problems with convergence though, both in separable and non-separable cases.
- Can fix it in linearly separable case, i.e. find better $b$. 

No convergence problems though, both in separable and non-separable cases.
Another Approach: Design a Loss Function

• Find weight vector $a$ s.t. $\forall z^1, \ldots, z^n, a^t z^i > 0$

• Design a loss function $J(a)$, which is minimum when $a$ is a solution vector

• Let $Z(a)$ be the set of examples misclassified by $a$

\[ Z(a) = \{ z^i \mid a^t z^i < 0 \} \]

• Natural choice: number of misclassified examples

\[ J(a) = |Z(a)| \]

• Unfortunately, can’t be minimized with gradient descent

  • piecewise constant, gradient zero or does not exist
Perceptron Loss Function

- Better choice: Perceptron loss function

\[ J_p(a) = \sum_{z \in Z(a)} (-a^t z) \]

- If \( z \) is misclassified, \( a^t z < 0 \)
- Thus \( J(a) \geq 0 \)
- \( J_p(a) \) is proportional to the sum of distances of misclassified examples to decision boundary
- \( J_p(a) \) is piecewise linear and suitable for gradient descent
Optimizing with Gradient Descent

\[ J_p(a) = \sum_{z \in Z(a)} (-a^t z) \]

- Gradient of \( J_p(a) \) is \( \nabla J_p(a) = \sum_{z \in Z(a)} (-z) \)
  - cannot solve \( \nabla J_p(a) = 0 \) analytically because of \( Z(a) \)

- Recall update rule for gradient descent
  \[ x^{(k+1)} = x^{(k)} - \alpha \nabla J(x^{(k)}) \]

- Gradient decent update rule for \( J_p(a) \) is:
  \[ a^{(k+1)} = a^{(k)} + \alpha \sum_{z \in Z(a)} z \]
  - called batch rule because it is based on all examples
  - true gradient descent
Perceptron Single Sample Rule

• Gradient decent single sample rule for $J_p(a)$ is
  
  $$a^{(k+1)} = a^{(k)} + \alpha \cdot z_M$$

• $z_M$ is one sample misclassified by $a^{(k)}$

• must have a consistent way to visit samples

• Geometric Interpretation:

• $z_M$ misclassified by $a^{(k)}$
  
  $$\left(a^{(k)}\right)^t z_M \leq 0$$

• $z_M$ is on the wrong side of decision boundary

• adding $\alpha \cdot z_M$ to a moves decision boundary in the right direction
if $\alpha$ is too large, previously correctly classified sample $z^i$ is now misclassified

if $\alpha$ is too small, $z_M$ is still misclassified
Non-Linearily Separable Case

- Suppose we have examples:
  - class 1: [2,1], [4,3], [3,5]
  - class 2: [1,3], [5,6]
  - not linearly separable
- Still would like to get approximate separation
- Good line choice is shown in green
- Let us run gradient descent
  - Add extra feature and “normalize”

\[
\begin{align*}
\mathbf{z}^1 &= \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} & \mathbf{z}^2 &= \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} & \mathbf{z}^3 &= \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} & \mathbf{z}^4 &= \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} & \mathbf{z}^5 &= \begin{bmatrix} -1 \\ -5 \\ -6 \end{bmatrix}
\end{align*}
\]
Non-Linearily Separable Case

- single sample perceptron rule
- Initial weights $a^{(1)} = [1 \ 1 \ 1]$
- This is line $x_1 + x_2 + 1 = 0$
- Use fixed learning rate $\alpha = 1$
- Rule is: $a^{(k+1)} = a^{(k)} + z_M$

\[
\begin{bmatrix}
z^1 \\
z^2 \\
z^3 \\
z^4 \\
z^5 \\
\end{bmatrix} =
\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} +
\begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} +
\begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} +
\begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} +
\begin{bmatrix} -1 \\ -5 \\ -6 \end{bmatrix}
\]

- $a^t z^1 = [1 \ 1 \ 1] \cdot [1 \ 2 \ 1]^t > 0$
- $a^t z^2 = [1 \ 1 \ 1] \cdot [1 \ 4 \ 3]^t > 0$
- $a^t z^3 = [1 \ 1 \ 1] \cdot [1 \ 3 \ 5]^t > 0$
Non-Linearily Separable Case

- \( a^{(1)} = [1 \ 1 \ 1] \)
- rule is: \( a^{(k+1)} = a^{(k)} + z_M \)

\[
\begin{bmatrix}
1 \\
2 \\
1
\end{bmatrix} \quad \begin{bmatrix}
1 \\
4 \\
3
\end{bmatrix} \quad \begin{bmatrix}
1 \\
3 \\
5
\end{bmatrix} \quad \begin{bmatrix}
-1 \\
-1 \\
-3
\end{bmatrix} \quad \begin{bmatrix}
-1 \\
-5 \\
-6
\end{bmatrix}
\]

- \( a^t z^4 = [1 \ 1 \ 1] \cdot [-1 \ -1 \ -3]^t = -5 < 0 \)
- **Update:** \( a^{(2)} = a^{(1)} + z_M = [1 \ 1 \ 1] + [-1 \ -1 \ -3] = [0 \ 0 \ -2] \)

- \( a^t z^5 = [0 \ 0 \ -2] \cdot [-1 \ -5 \ -6]^t = 12 > 0 \)
- \( a^t z^1 = [0 \ 0 \ -2] \cdot [1 \ 2 \ 1]^t < 0 \)
- **Update:** \( a^{(3)} = a^{(2)} + z_M = [0 \ 0 \ -2] + [1 \ 2 \ 1] = [1 \ 2 \ -1] \)
Non-Linearily Separable Case

- $a^{(3)} = [1 \ 2 \ -1]$
- rule is: $a^{(k+1)} = a^{(k)} + z_M$

\[
z^1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}, \quad z^2 = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix}, \quad z^3 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}, \quad z^4 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix}, \quad z^5 = \begin{bmatrix} -1 \\ -5 \\ -6 \end{bmatrix}
\]

- $a^t z^2 = [1 \ 4 \ 3] \cdot [1 \ 2 \ -1]^t = 6 > 0$
- $a^t z^3 = [1 \ 3 \ 5] \cdot [1 \ 2 \ -1]^t = 2 > 0$
- $a^t z^4 = [-1 \ -1 \ -3] \cdot [1 \ 2 \ -1]^t = 0$
- $Update: \ a^{(4)} = a^{(3)} + z_M = [1 \ 2 \ -1] + [-1 \ -1 \ -3] = [0 \ 1 \ -4]$
Non-Linearily Separable Case

• We can continue this forever
  • there is no solution vector $a$ satisfying for all $a^t z_i > 0$ for all $i$

• Need to stop at a good point

• Solutions at iterations 900 through 915

• Some are good some are not

• How do we stop at a good solution?
Convergence of Perceptron Rules

1. Classes are linearly separable:
   • with fixed learning rate, both single sample and batch rules converge to a correct solution \( a \)
   • can be any \( a \) in the solution space

2. Classes are not linearly separable:
   • with fixed learning rate, both single sample and batch do not converge
   • can ensure convergence with appropriate variable learning rate
     • \( \alpha \to 0 \) as \( k \to \infty \)
     • example, inverse linear: \( \alpha = c/k \), where \( c \) is any constant
       • also converges in the linearly separable case
       • no guarantee that we stop at a good point, but there are good reasons to choose inverse linear learning rate
   • Practical Issue: both single sample and batch algorithms converge faster if features are roughly on the same scale
     • see kNN lecture on feature normalization
<table>
<thead>
<tr>
<th>Batch</th>
<th>Single Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>True gradient descent, full gradient computed</td>
<td>Only partial gradient is computed</td>
</tr>
<tr>
<td>Smoother gradient because all samples are used</td>
<td>Noisier gradient, therefore may concentrates more than necessary on any isolated training examples (those could be noise)</td>
</tr>
<tr>
<td>Takes longer to converge</td>
<td>Converges faster</td>
</tr>
<tr>
<td></td>
<td>Easier to analyze</td>
</tr>
</tbody>
</table>
Linear Machine: Logistic Regression

• Despite the name, used for classification, not regression
• Instead of putting $g(x)$ through a sign function, can put it through a smooth function
  • smooth function is better for gradient descent
• Logistic sigmoid function
  • $g(x,w) = w_0 + x_1 w_1 + \ldots + x_d w_d$
  • let $f(x,w) = \sigma(g(x,w))$

$$
\sigma(a) = \frac{1}{1 + \exp(-a)}
$$
Linear Machine: Logistic Regression

- \( f(x,w) = \sigma(g(x,w)) \)
  - bigger 0.5 if \( g(x,w) \) is positive
    - decide class 1
  - less 0.5 if \( g(x,w) \) is negative
    - decide class 2
- Has an interesting probabilistic interpretation
- \( P(\text{class 1}|x) = \sigma(g(x,w)) \)
- Under a certain loss function, can be optimized exactly with gradient decent

\[
\sigma(a) = \frac{1}{1 + \exp(-a)}
\]
Generalized Linear Classifier

- Can use other discriminant functions, like quadratics
  \[ g(x) = w_0 + w_1 x_1 + w_2 x_2 + w_{12} x_1 x_2 + w_{11} x_1^2 + w_{22} x_2^2 \]

- Methodology is almost the same as in the linear case:
  - \[ f(x) = \text{sign}(w_0 + w_1 x_1 + w_2 x_2 + w_{12} x_1 x_2 + w_{11} x_1^2 + w_{22} x_2^2) \]
  - \[ z = [ 1 \ x_1 \ x_2 \ x_1 x_2 \ x_1^2 \ x_2^2 ] \]
  - \[ a = [ w_0 \ w_1 \ w_2 \ w_{12} \ w_{11} \ w_{22} ] \]
  - “normalization”: multiply negative class samples by -1
  - all the other procedures remain the same, i.e. gradient descent to minimize Perceptron loss function, or MSE procedure, etc.
Generalized Linear Classifier

• In general, to the linear function:
  \[ g(x, w) = w_0 + \sum_{i=1}^{d} w_i x_i \]

• can add quadratic terms:
  \[ g(x, w) = w_0 + \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d} \sum_{j=1}^{d} w_{ij} x_i x_j \]

• This is still a linear function in its parameters \( w \)

• \( g(y, v) = v_0 + v^t y \)
  \[ v_0 = w_0 \]
  \[ y = [x_1 \ x_2 \ldots \ x_d \ x_1x_1 \ x_1x_2 \ldots \ x_dx_d] \]
  \[ v = [w_1 \ w_2 \ldots \ w_d \ w_{11} \ w_{12} \ldots \ w_{dd}] \]

• Can use all the same training methods as before
Generalized Linear Classifier

- Generalized linear classifier

\[ g(x,w) = w_0 + \sum_{i=1}^{m} w_i h_i(x) \]

- \( h(x) \) are called basis function, can be arbitrary functions
  - in strictly linear case, \( h_i(x) = x_i \)

- Linear function in its parameters \( w \)

\[ g(x,w) = w_0 + w^t h \]

\[ h = [h_1(x) \ h_2(x) \ ... \ h_m(x)] \]

\[ [w_1 \ ... \ w_m] \]

- Can use all the same training methods as before
Generalized Linear Classifier

- Usually face severe overfitting
  - too many degrees of freedom
  - Boundary can “curve” to fit to the noise in the data
- Helps to regularize by keeping $w$ small
  - small $w$ means the boundary is not as curvy
- Usually add $\lambda ||w||^2$ to the loss function
- Recall quadratic loss function
  \[ L(x^i, y^i, w) = ||f(x^i, w) - y^i||^2 \]
- Regularized version
  \[ L(x^i, y^i, w) = ||f(x^i, w) - y^i||^2 + \lambda ||w||^2 \]
- How to set $\lambda$?
- With cross-validation
Learning by Gradient Descent

- Can have classifiers even more general
- More general than generalized linear 😊
- Suppose we suspect that the machine has to have functional form $f(x, w)$, not necessarily linear
- Pick differentiable per-sample loss function $L(x^i, y^i, w)$
- Need to find $w$ that minimizes $L = \Sigma_i L(x^i, y^i, w)$
- Use gradient-based minimization:
  - Batch rule: $w = w - \alpha \nabla L(w)$
  - Or single sample rule: $W = W - \alpha \nabla L(x^i, y^i, w)$