Lecture 4

Machine Learning

Linear Classifier

2 classes
Outline

• Optimization with gradient descent
• Linear Classifier
  • Two class case
    • Loss functions
    • Perceptron
      • Batch
      • Single sample
    • Logistic Regression
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 is just derivative in 1D

• Example: $J(x) = (x-5)^2$ and derivative is $\frac{d}{dx} J(x) = 2(x - 5)$

Let $x = 3$

$- \frac{d}{dx} J(3) = 4$

• derivative says increase $x$

Let $x = 8$

$- \frac{d}{dx} J(3) = -6$

• derivative says decrease $x$
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 \( a = \begin{bmatrix} 10 \\ 5 \end{bmatrix} \)
- \( \frac{\partial}{\partial x_1} J(a) = 10 \)
- \( \frac{\partial}{\partial x_2} J(a) = -10 \)
- \( \nabla J(a) = \begin{bmatrix} 10 \\ -10 \end{bmatrix} \)
- \( -\nabla J(a) = \begin{bmatrix} -10 \\ 10 \end{bmatrix} \)
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 slide
  - \( a = \begin{bmatrix} 10 \\ 5 \end{bmatrix}, \quad -\nabla J(a) = \begin{bmatrix} -10 \\ 10 \end{bmatrix} \)
  - \( \alpha = 0.2 \)
    - \( a - \alpha \nabla J(a) = \begin{bmatrix} 10 \\ 5 \end{bmatrix} + 0.2 \begin{bmatrix} -10 \\ 10 \end{bmatrix} = \begin{bmatrix} 8 \\ 7 \end{bmatrix} \)
- \( J(10, 5) = 50; \quad J(8,7) = 18 \)
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

\[ J(x) = x_1 x_2 (x_1 - 1)(x_2 - 2) \]

- 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
Variable Learning Rate

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

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

- Usually do not keep track of all intermediate solutions

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

- Monitor learning rate by looking at how fast the objective function decreases.

![Graph showing objective function vs. number of iterations for different learning rates]

- Very high learning rate
- Low learning rate
- High learning rate
- Good learning rate
Learning Rate: Loss Surface Illustration

\[ \alpha = 0.1 \]

\[ \sim 3k \text{ updates} \]

\[ \alpha = 0.001 \]

\[ \sim 3k \text{ updates} \]

\[ \alpha = 0.01 \]

\[ \sim 0.3k \text{ updates} \]
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 \texttt{fminunc} for various advanced optimization methods
Supervised Machine Learning (Recap)

• Chose type of $f(x, w)$
  • $w$ are tunable weights, $x$ is the input example
  • $f(x, w)$ should output the correct class of sample $x$
  • use labeled samples to tune weights $w$ so that $f(x, w)$ give the correct class $y$ for $x$
    • with help of loss function $L(f(x, w), y)$

• How to choose type of $f(x, w)$?
  • many choices
  • previous lecture: kNN classifier
  • this lecture: linear classifier
Linear Classifier

- Classifier is linear if it makes a decision based on linear combination of features
  \[ g(x, w) = w_0 + x_1w_1 + \ldots + x_dw_d \]
  - \( g(x, w) \) sometimes called *discriminant function*

- Encode 2 classes as
  - \( y = 1 \) for the first class
  - \( y = -1 \) for the second class

- One choice for linear classifier
  \[ f(x, w) = \text{sign}(g(x, w)) \]
  - 1 if \( g(x, w) \) is positive
  - -1 if \( g(x, w) \) is negative
Linear Classifier: Decision Boundary

- $f(x,w) = \text{sign}(g(x,w)) = \text{sign}(w_0 + x_1 w_1 + \ldots + x_d w_d)$
- Decision boundary is linear
- Find $w_0, w_1, \ldots, w_d$ that gives best separation of two classes with linear boundary
More on Linear Discriminant Function (LDF)

- LDF: \( g(x, w_0, w_1, \ldots, w_d) = w_0 + x_1 w_1 + \ldots + x_d w_d \)

\[ w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \]

The bias or threshold is represented by \( w_0 \).

The decision boundary is given by \( g(x) = 0 \).

The decision region for class 1 is where \( g(x) > 0 \).

The decision region for class 2 is where \( g(x) < 0 \).

Decision boundary
More on Linear Discriminant Function (LDF)

- Decision boundary: $g(x, w) = w_0 + x_1 w_1 + \ldots + x_d w_d = 0$
- This is a hyperplane, by definition
  - a point in 1D
  - a line in 2D
  - a plane in 3D
  - a hyperplane in higher dimensions
Vector Notation

• Linear discriminant function \( g(x, w, w_0) = w^t x + w_0 \)

• Example in 2D

\[
g(x, w, w_0) = 3x_1 + 2x_2 + 4
\]

\[
x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}
\]

\[
w = \begin{bmatrix} 3 \\ 2 \end{bmatrix}, \ w_0 = 4
\]

• Shorter notation if add extra feature of value 1 to \( x \)

\[
z = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}
\]

\[
a = \begin{bmatrix} 4 \\ 3 \\ 2 \end{bmatrix}
\]

\[
g(z, a) = z^t a = 4 + 3x_1 + 2x_2 = x^t w + w_0 = g(x, w, w_0)
\]

• Use \( a^t z \) instead of \( w^t x + w_0 \)
Fitting Parameters $w$

- Rewrite $g(x,w,w_0) = [w_0 \quad w^\top] \begin{bmatrix} 1 \\ x \end{bmatrix} = a^\top z = g(z,a)$

- $z$ is called augmented feature vector
- new problem equivalent to the old $g(z,a) = a^\top z$
**Augmented Feature Vector**

- Feature augmenting simplifies notation
- Assume augmented feature vectors for the rest of the lecture
  - given examples $x^1, ..., x^n$ convert them to augmented examples $z^1, ..., z^n$ by adding a new dimension of value 1
- $g(z,a) = a^t z$
- $f(z,a) = \text{sign}(g(z,a))$
• If there is weight vector $\mathbf{a}$ that classifies all examples correctly, it is called a \textit{separating} or \textit{solution} vector
  • then there are infinitely many solution vectors $\mathbf{a}$
  • then the original samples $\mathbf{x}^1, \ldots, \mathbf{x}^n$ are also linearly separable
Solution region: the set of all solution vectors \( \mathbf{a} \)
Loss Function

• How to find solution vector \( a \)?
  • or, if no separating \( a \) exists, a good approximate solution vector \( a \)?

• Design a non-negative loss function \( L(a) \)
  • \( L(a) \) is small if \( a \) is good
  • \( L(a) \) is large if \( a \) is bad

• Minimize \( L(a) \) with gradient descent

• Usually design of \( L(a) \) has two steps
  1. design per-example loss \( L(f(z^i,a),y^i) \)
     • penalizes for deviations of \( f(z^i,a) \) from \( y^i \)
  2. total loss adds up per-sample loss over all training examples

\[
L(a) = \sum_i L(f(z^i,a), y^i)
\]
Loss Function, First Attempt

- Per-example loss function measures if error happens

\[
L(f(z^i, a), y^i) = \begin{cases} 
0 & \text{if } f(z^i, a) = y^i \\
1 & \text{otherwise}
\end{cases}
\]

- Example

\[
\begin{align*}
a &= \begin{bmatrix} 2 \\ -3 \end{bmatrix} \\
z^1 &= \begin{bmatrix} 1 \\ 2 \end{bmatrix} \\
f(z^1, a) &= \text{sign}(a^T z^1) \\
&= \text{sign}(1 \cdot 2 - 3 \cdot 2) \\
&= -1 \\
L(f(z^1, a), y^1) &= 1
\end{align*}
\]

\[
\begin{align*}
z^2 &= \begin{bmatrix} 1 \\ 4 \end{bmatrix} \\
f(z^2, a) &= \text{sign}(a^T z^2) \\
&= \text{sign}(1 \cdot 2 - 3 \cdot 4) \\
&= -1 \\
L(f(z^2, a), y^2) &= 0
\end{align*}
\]
Loss Function, First Attempt

- Per-example loss function measures if error happens
  \[
  L(f(z^i, a), y^i) = \begin{cases} 
  0 & \text{if } f(z^i, a) = y^i \\
  1 & \text{otherwise}
  \end{cases}
  \]

- Total loss function
  \[
  L(a) = \sum_{i} L(f(z^i, a), y^i)
  \]

- For previous example
  \[
  a = \begin{bmatrix} 2 \\ -3 \end{bmatrix} \quad L(f(z^1, a), y^1) = 1
  \]
  \[
  L(f(z^2, a), y^2) = 0 \quad L(a) = 1 + 0 = 1
  \]

- Thus this loss function just counts the number of errors
Loss Function: First Attempt

• Per-example loss
\[ L(f(z^i, a), y^i) = \begin{cases} 0 & \text{if } f(z^i, a) = y^i \\ 1 & \text{otherwise} \end{cases} \]

• Total loss
\[ L(a) = \sum_i L(f(z^i, a), y^i) \]

• Unfortunately, cannot minimize this loss function with gradient descent
  • piecewise constant, gradient zero or does not exist
Perceptron Loss Function

- Different Loss Function: Perceptron Loss

\[ L_p(f(z^i, a), y^i) = \begin{cases} 
0 & \text{if } f(z^i, a) = y^i \\
-y^i(a^T z^i) & \text{otherwise} 
\end{cases} \]

- \( L_p(a) \) is non-negative
  - positive misclassified example \( z^i \)
    - \( a^T z^i < 0 \)
    - \( y^i = 1 \)
    - \( y^i(a^T z^i) < 0 \)
  - negative misclassified example \( z^i \)
    - \( a^T z^i > 0 \)
    - \( y^i = -1 \)
    - \( y^i(a^T z^i) < 0 \)
  - if \( z^i \) is misclassified then \( y^i(a^T z^i) < 0 \)
  - if \( z^i \) is misclassified then \(-y^i(a^T z^i) > 0 \)

- \( L_p(a) \) proportional to distance of misclassified example to boundary
**Perceptron Loss Function**

\[
L_p(f(z^i, a), y^i) = \begin{cases} 
0 & \text{if } f(z^i, a) = y^i \\
-y^i(a^t z^i) & \text{otherwise}
\end{cases}
\]

- **Example**

  \[
a = \begin{bmatrix} 2 \\ -3 \end{bmatrix}
\]

  \[
z^1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad y^1 = 1
\]

  \[
f(z^1, a) = \text{sign}(a^t z^1)
  = \text{sign}(1 \cdot 2 - 3 \cdot 2)
  = \text{sign}(-4)
  = -1
\]

  \[
L_p(f(z^1, a), y^1) = 4
\]

  \[
z^2 = \begin{bmatrix} 1 \\ 4 \end{bmatrix} \quad y^2 = -1
\]

  \[
f(z^2, a) = \text{sign}(a^t z^2)
  = \text{sign}(1 \cdot 2 - 3 \cdot 4)
  = \text{sign}(-1)
  = -1
\]

  \[
L_p(f(z^2, a), y^2) = 0
\]

- **Total loss** \(L_p(a) = 4 + 0 = 4\)
Perceptron Loss Function

- **Per-example loss**
  \[ L_p(f(z^i, a), y^i) = \begin{cases} 
  0 & \text{if } f(z^i, a) = y^i \\ 
  -y^i(a^Tz^i) & \text{otherwise} 
\end{cases} \]

- **Total loss**
  \[ L_p(a) = \sum_i L(f(z^i, a), y^i) \]

- \( L_p(a) \) is piecewise linear and suitable for gradient descent
Optimizing with Gradient Descent

- Per-example loss
\[ L_p(f(z^i, a), y^i) = \begin{cases} 0 & \text{if } f(z^i, a) = y^i \\ -y^i(a^Tz^i) & \text{otherwise} \end{cases} \]

- Total loss
\[ L_p(a) = \sum_i L(f(z^i, a), y^i) \]

- Recall minimization with gradient descent, main step
\[ x = x - \alpha \nabla J(x) \]

- Gradient descent to minimize \( L_p(a) \), main step
\[ a = a - \alpha \nabla L_p(a) \]

- Need gradient vector \( \nabla L_p(a) \)
  - has as many dimensions as dimension of \( a \)
  - if \( a \) has 3 dimensions, gradient \( \nabla L_p(a) \) has 3 dimensions

\[ a = \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} \implies \nabla L_p(a) = \begin{bmatrix} \frac{\partial L_p}{\partial a_1} \\ \frac{\partial L_p}{\partial a_2} \\ \frac{\partial L_p}{\partial a_3} \end{bmatrix} \]
Optimizing with Gradient Descent

- Per-example loss
  \[ L_p(f(z^i, a), y^i) = \begin{cases} 
  0 & \text{if } f(z^i, a) = y^i \\ 
  -y^i(a^T z^i) & \text{otherwise} 
\end{cases} \]

- Total loss
  \[ L_p(a) = \sum_i L(f(z^i, a), y^i) \]

- Gradient descent to minimize \( L_p(a) \), main step
  \[ a = a - \alpha \nabla L_p(a) \]

- Need gradient vector \( \nabla L_p(a) \)
  \[ \nabla L_p(a) = \nabla \sum_i L_p(f(z^i, a), y^i) = \sum_i \nabla L_p(f(z^i, a), y^i) \]

- Compute and add up per example gradient vectors

\[ \begin{bmatrix} 
\frac{\partial L_p(f(z^i, a), y^i)}{\partial a_1} \\
\frac{\partial L_p(f(z^i, a), y^i)}{\partial a_2} \\
\frac{\partial L_p(f(z^i, a), y^i)}{\partial a_3} 
\end{bmatrix} \]

per example gradient
Per Example Loss Gradient

- Per-example loss has two cases

\[
    L_p(f(z^i, a), y^i) = \begin{cases} 
    0 & \text{if } f(z^i, a) = y^i \\
    -y^i(\mathbf{a}^Tz^i) & \text{otherwise}
    \end{cases}
\]

- First case, \( f(z^i, a) = y^i \)

\[
    \nabla L_p(f(z^i, a), y^i) = \begin{cases} 
    \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} & \text{if } f(z^i, a) = y^i \\
    ? & \text{otherwise}
    \end{cases}
\]

- To save space, rewrite

\[
    \nabla L_p(f(z^i, a), y^i) = \begin{cases} 
    0 & \text{if } f(z^i, a) = y^i \\
    ? & \text{otherwise}
    \end{cases}
\]
Per Example Loss Gradient

- Per-example loss has two cases

\[ L_p(f(z^i, a), y^i) = \begin{cases} 
0 & \text{if } f(z^i, a) = y^i \\
-y^i(a^t z^i) & \text{otherwise}
\end{cases} \]

- Second case, \( f(z^i, a) \neq y^i \)

\[ \nabla L_p(f(z^i, a), y^i) = \begin{bmatrix} 
\frac{\partial L}{\partial a_1} (-y^i(a^t z^i)) \\
\frac{\partial L}{\partial a_2} (-y^i(a^t z^i)) \\
\frac{\partial L}{\partial a_3} (-y^i(a^t z^i)) 
\end{bmatrix} = \begin{bmatrix} 
\frac{\partial L}{\partial a_1} (-y^i(a_1 z_1^i + a_2 z_2^i + a_3 z_3^i)) \\
\frac{\partial L}{\partial a_2} (-y^i(a_1 z_1^i + a_2 z_2^i + a_3 z_3^i)) \\
\frac{\partial L}{\partial a_3} (-y^i(a_1 z_1^i + a_2 z_2^i + a_3 z_3^i)) 
\end{bmatrix} = \begin{bmatrix} 
-y^i z_1^i \\
-y^i z_2^i \\
-y^i z_3^i 
\end{bmatrix} \]

\[ \nabla L_p(f(z^i, a), y^i) = -y^i z^i \]

- Combining both cases, gradient for per-example loss

\[ \nabla L_p(f(z^i, a), y^i) = \begin{cases} 
0 & \text{if } f(z^i, a) = y^i \\
-y^i z^i & \text{otherwise}
\end{cases} \]
Optimizing with Gradient Descent

- Gradient for per-example loss
  \[ \nabla L_p(f(z^i, a), y^i) = \begin{cases} 
  0 & \text{if } f(z^i, a) = y^i \\
  -y^iz^i & \text{otherwise}
\end{cases} \]

- Total gradient
  \[ \nabla L_p(a) = \sum_i \nabla L_p(f(z^i, a), y^i) \]

- Simpler formula
  \[ \nabla L_p(a) = \sum_{\text{misclassified examples } i} -y^iz^i \]

- Gradient decent update rule for \( L_p(a) \)
  \[ a = a + \alpha \sum_{\text{misclassified examples } i} y^iz^i \]

- called **batch** because it is based on all examples
- can be slow if number of examples is very large
Perceptron Loss Batch Example

- **Examples**

\[
\begin{align*}
x_1 &= \begin{bmatrix} 2 \\ 3 \end{bmatrix} & x_2 &= \begin{bmatrix} 4 \\ 3 \end{bmatrix} & x_3 &= \begin{bmatrix} 3 \\ 5 \end{bmatrix} \\
x_4 &= \begin{bmatrix} 1 \\ 3 \end{bmatrix} & x_5 &= \begin{bmatrix} 5 \\ 6 \end{bmatrix}
\end{align*}
\]

- **Labels**

\[
\begin{align*}
y_1 &= 1 & y_2 &= 1 & y_3 &= 1 & y_4 &= -1 & y_5 &= -1
\end{align*}
\]

- **Add extra feature**

\[
\begin{align*}
z_1 &= \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} & z_2 &= \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} & z_3 &= \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} & z_4 &= \begin{bmatrix} 1 \\ 3 \end{bmatrix} & z_5 &= \begin{bmatrix} 1 \\ 6 \end{bmatrix}
\end{align*}
\]

- **Pile all examples as rows in matrix** \( Z \)

- **Pile all labels into column vector** \( Y \)
Perceptron Loss Batch Example

- Examples in $Z$, labels in $Y$

\[
Z = \begin{bmatrix}
1 & 2 & 3 \\
1 & 4 & 3 \\
1 & 3 & 5 \\
1 & 1 & 3 \\
1 & 5 & 6
\end{bmatrix}, \quad
Y = \begin{bmatrix}
1 \\
1 \\
1 \\
-1 \\
-1
\end{bmatrix}
\]

- Initial weights $a = \begin{bmatrix}
1 \\
1 \\
1
\end{bmatrix}$

- This is line $x_1 + x_2 + 1 = 0$
Perceptron Loss Batch Example

\[
\mathbf{Z} = \begin{bmatrix}
1 & 2 & 3 \\
1 & 4 & 3 \\
1 & 3 & 5 \\
1 & 1 & 3 \\
1 & 5 & 6 \\
\end{bmatrix}
\]

\[
\mathbf{a} = \begin{bmatrix}
1 \\
1 \\
1 \\
\end{bmatrix}
\]

\[
\mathbf{Y} = \begin{bmatrix}
1 \\
1 \\
-1 \\
-1 \\
\end{bmatrix}
\]

- Perceptron Batch
  \[
a = a + \alpha \sum_{\text{misclassified examples } i} y^i z^i
\]
- Let us use learning rate \( \alpha = 0.2 \)
  \[
a = a + 0.2 \sum_{\text{misclassified examples } i} y^i z^i
\]
- Sample misclassified if \( y(a^t z) < 0 \)
Perceptron Loss Batch Example

\[ Z = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 5 \\ 1 & 1 & 3 \\ 1 & 5 & 6 \end{bmatrix} \]

\[ a = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \]

\[ Y = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \]

- Sample misclassified if \( y(a^Tz) < 0 \)
- Find all misclassified samples with one line in matlab
- Could have for loop to compute \( a^Tz \)
- For \( i = 1 \)
  \[ y^a^Tz = 1 \cdot \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = 6 > 0 \]
- Repeat for \( i = 2, 3, 4, 5 \)
Perceptron Loss Batch Example

\[ Z = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 5 \\ 1 & 1 & 3 \\ 1 & 5 & 6 \end{bmatrix} \]

\[ a = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \]

\[ Y = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \]

- Sample misclassified if \( y(a^t z) < 0 \)
- Find all misclassified samples with one line in matlab
- Can compute \( a^t z \) for all samples

\[
\begin{bmatrix}
 a^t z^1 \\
 a^t z^2 \\
 a^t z^3 \\
 a^t z^4 \\
 a^t z^5
\end{bmatrix} = Z * a = \begin{bmatrix} 6 \\ 8 \\ 9 \\ 5 \\ 12 \end{bmatrix}
\]
Perceptron Loss Batch Example

\[
Z = \begin{bmatrix}
1 & 2 & 3 \\
1 & 4 & 3 \\
1 & 3 & 5 \\
1 & 1 & 3 \\
1 & 5 & 6
\end{bmatrix}
\quad
a = \begin{bmatrix}
1 \\
1 \\
1
\end{bmatrix}
\quad
Y = \begin{bmatrix}
1 \\
1 \\
-1 \\
-1
\end{bmatrix}
\]

- Sample misclassified if \( y(a^t z) < 0 \)
- Can compute \( y(a^t z) \) for all samples in one line

\[
y^1(a^t z^1) = y^2(a^t z^2) = \cdots = y^5(a^t z^5) = y \cdot (Z \cdot a)
\]

\[
\begin{bmatrix}
1 \\
1 \\
1 \\
-1 \\
-1
\end{bmatrix} \cdot
\begin{bmatrix}
6 \\
8 \\
9 \\
-5 \\
-12
\end{bmatrix} = \begin{bmatrix}
6 \\
8 \\
9 \\
-5 \\
-12
\end{bmatrix}
\]

Total loss is \( L(a) = 5 + 12 = 17 \)

- Per example loss is

\[
L_p(f(z^i, a), y^i) = \begin{cases}
0 & \text{if } f(z^i, a) = y^i \\
-y^i(a^t z^i) & \text{otherwise}
\end{cases}
\]
Perceptron Loss Batch Example

\[ Z = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 5 \\ 1 & 1 & 3 \\ 1 & 5 & 6 \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \]

- Samples 4 and 5 misclassified
- Perceptron Batch rule update \( \mathbf{a} = \mathbf{a} + 0.2 \sum_{\text{misclassified examples } i} \mathbf{y}^i \mathbf{z}^i \)

\[
\mathbf{a} = \mathbf{a} + 0.2 \left( -1 \cdot \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix} -1 \cdot \begin{bmatrix} 3 \\ 5 \\ 6 \end{bmatrix} \right) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0.2 \\ 0.2 \\ 0.6 \end{bmatrix} - \begin{bmatrix} 0.2 \\ 1 \\ 1.2 \end{bmatrix} = \begin{bmatrix} 0.6 \\ -0.2 \\ -0.8 \end{bmatrix}
\]

- This is line \(-0.2x_1 -0.8 x_2 +0.6 = 0\)
Perceptron Loss Batch Example

\[ Z = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 5 \\ 1 & 1 & 3 \\ 1 & 5 & 6 \end{bmatrix} \quad a = \begin{bmatrix} 0.6 \\ -0.2 \\ -0.8 \end{bmatrix} \quad Y = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \]

- Sample misclassified if \( y(a^Tz) < 0 \)
- Find all misclassified samples \( (Z*a) \cdot Y = \begin{bmatrix} -2.2 \\ -2.6 \\ -4.0 \end{bmatrix} \)
- Total loss is \( L(a) = 2.2 + 2.6 + 4 = 8.8 \)
  - previous loss was 17 with 2 misclassified examples
Perceptron Loss Batch Example

\[ Z = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 5 \\ 1 & 1 & 3 \\ 1 & 5 & 6 \end{bmatrix} \quad a = \begin{bmatrix} 0.6 \\ -0.2 \\ -0.8 \end{bmatrix} \quad Y = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \]

\[ (Z \cdot a) \cdot Y = \begin{bmatrix} -2.2 \\ -2.6 \\ -4.0 \\ 2 \\ 5.2 \end{bmatrix} \]

- Perceptron Batch rule update

\[ a = a + 0.2 \sum_{\text{misclassified examples } i} y^i z^i \]

\[ a = a + 0.2 \left( 1 \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + 1 \cdot \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} + 1 \cdot \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \right) = \begin{bmatrix} 0.6 \\ -0.2 \\ -0.8 \end{bmatrix} + \begin{bmatrix} 0.2 \\ 0.4 \\ 0.6 \end{bmatrix} + \begin{bmatrix} 0.2 \\ 0.8 \\ 0.6 \end{bmatrix} + \begin{bmatrix} 0.2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.2 \\ 1.6 \\ 1.4 \end{bmatrix} \]

- This is line \( 1.6x_1 + 1.4x_2 + 1.2 = 0 \)
Perceptron Loss Batch Example

\[
Z = \begin{bmatrix}
1 & 2 & 3 \\
1 & 4 & 3 \\
1 & 3 & 5 \\
1 & 1 & 3 \\
1 & 5 & 6 \\
\end{bmatrix} \quad \mathbf{a} = \begin{bmatrix}
1.2 \\
1.6 \\
1.4 \\
\end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix}
1 \\
1 \\
-1 \\
-1 \\
\end{bmatrix}
\]

- Sample misclassified if \( y(a^T z) < 0 \)
- Find all misclassified samples

\[
(Z \ast \mathbf{a}) \ast \mathbf{Y} = \begin{bmatrix}
8.6 \\
11.8 \\
13.0 \\
-7 \\
-17.6 \\
\end{bmatrix}
\]

- Total loss is \( L(\mathbf{a}) = 7 + 17.6 = 24.6 \)
  - previous loss was 8.8 with 3 misclassified examples
  - loss went up, means learning rate of 0.2 is too high
**Perceptron Single Sample Gradient Descent**

- Batch Perceptron can be slow to converge if lots of examples.

**Single sample** optimization

- update weights $a$ as soon as possible, after seeing 1 example.

**One iteration (epoch)**

- go over all examples, as soon as find misclassified example, update

$$a = a + \alpha \cdot yz$$

- $z$ is misclassified example, $y$ is its label.

**Geometric intuition**

- $z$ misclassified by $a$ means

$$a^t yz \leq 0$$

- $z$ is on the wrong side of decision boundary.

- adding $\alpha \cdot yz$ moves decision boundary in the right direction.

- Illustration for positive example $z$.

- Best to go over examples in random order.
if $\alpha$ is too small, $z$ is still misclassified

if $\alpha$ is too large, previously correctly classified sample $z^i$ is now misclassified
Batch Gradient Descent, one iteration

Single sample gradient descent, one iteration
### Perceptron Single Sample Rule Example

<table>
<thead>
<tr>
<th>name</th>
<th>good attendance?</th>
<th>tall?</th>
<th>sleeps in class?</th>
<th>chews gum?</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>A</td>
</tr>
<tr>
<td>Steve</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>F</td>
</tr>
<tr>
<td>Mary</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>F</td>
</tr>
<tr>
<td>Peter</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>A</td>
</tr>
</tbody>
</table>

- **class 1**: students who get grade A
- **class 2**: students who get grade F
• Convert attributes to numerical values

<table>
<thead>
<tr>
<th>name</th>
<th>good attendance?</th>
<th>tall?</th>
<th>sleeps in class?</th>
<th>chews gum?</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>Steve</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>Mary</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>Peter</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Augment Feature Vector

<table>
<thead>
<tr>
<th>name</th>
<th>extra</th>
<th>good attendance?</th>
<th>tall?</th>
<th>sleeps in class?</th>
<th>chews gum?</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>Steve</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>Mary</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>Peter</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- convert samples $x^1, ..., x^n$ to augmented samples $z^1, ..., z^n$ by adding a new dimension of value 1
Set fixed learning rate to $\alpha = 1$

Gradient descent with single sample rule

- visit examples in random order
- example misclassified if $y(a^Tz) < 0$
- when misclassified example $z$ found, update $a^{(k+1)} = a^{(k)} + yz$

<table>
<thead>
<tr>
<th>name</th>
<th>extra</th>
<th>good attendance?</th>
<th>tall?</th>
<th>sleeps in class?</th>
<th>chews gum?</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>Steve</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>Mary</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>Peter</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Apply Single Sample Rule

```
<table>
<thead>
<tr>
<th>name</th>
<th>extra</th>
<th>good attendance?</th>
<th>tall?</th>
<th>sleeps in class?</th>
<th>chews gum?</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>Steve</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>Mary</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>Peter</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```
Apply Single Sample Rule

- initial weights $a^{(1)} = \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix}$
- for simplicity, we will visit all samples sequentially
- example misclassified if $y(a^t z) < 0$

<table>
<thead>
<tr>
<th>name</th>
<th>$y$</th>
<th>$y(a^t z)$</th>
<th>misclassified?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>1</td>
<td>$0.25<em>1+0.25</em>1+0.25<em>1+0.25</em>(-1)+0.25*(-1) &gt; 0$</td>
<td>no</td>
</tr>
<tr>
<td>Steve</td>
<td>-1</td>
<td>$-1 * (0.25<em>1+0.25</em>1+0.25<em>1+0.25</em>1+0.25*1) &lt; 0$</td>
<td>yes</td>
</tr>
</tbody>
</table>

- new weights $a^{(2)} = a^{(1)} + yz = \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.75 \\ 0.75 \\ 0.75 \\ 0.75 \end{bmatrix}$
Apply Single Sample Rule

\[ a^{(2)} = \begin{bmatrix} 0.75 \\ 0.75 \\ 0.75 \\ 0.75 \end{bmatrix} \]

<table>
<thead>
<tr>
<th>name</th>
<th>y</th>
<th>( y(a^t z) )</th>
<th>misclassified?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>-1</td>
<td>(-1\times(-0.75\times1 - 0.75 \times (-1) - 0.75 \times (-1) - 0.75 \times 1) &lt; 0)</td>
<td>yes</td>
</tr>
</tbody>
</table>

- new weights \( a^{(3)} = a^{(2)} + yz = \begin{bmatrix} 0.75 \\ 0.75 \\ 0.75 \\ 0.75 \end{bmatrix} - \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} -1.75 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix} \)
Apply Single Sample Rule

\[
a^{(3)} = \begin{bmatrix}
-1.75 \\
0.25 \\
0.25 \\
0.25 \\
-1.75
\end{bmatrix}
\]

<table>
<thead>
<tr>
<th>name</th>
<th>y</th>
<th>( y(a^t z) )</th>
<th>misclassified?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>1</td>
<td>-1.75 <em>1 +0.25</em> 1+0.25* (-1) +0.25 <em>(-1)-1.75</em>1 &lt; 0</td>
<td>yes</td>
</tr>
</tbody>
</table>

• new weights \( a^{(4)} = a^{(3)} + yz = \begin{bmatrix}
-1.75 \\
0.25 \\
0.25 \\
0.25 \\
-1.75
\end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix}
-0.75 \\
1.25 \\
-0.75 \\
-0.75 \\
-0.75
\end{bmatrix} \]
Single Sample Rule: Convergence

\[
a^{(4)} = \begin{bmatrix}
-0.75 \\
1.25 \\
-0.75 \\
-0.75 \\
-0.75 \\
\end{bmatrix}
\]

<table>
<thead>
<tr>
<th>name</th>
<th>y</th>
<th>( y(a^t z) )</th>
<th>misclassified?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>1</td>
<td>-0.75 <em>1 +1.25</em>1 -0.75*1 -0.75 *(-1) -0.75 *(-1)+0</td>
<td>no</td>
</tr>
<tr>
<td>Steve</td>
<td>-1</td>
<td>-1*(-0.75<em>1+1.25</em>1 -0.75<em>1 -0.75</em>1-0.75*1)&gt;0</td>
<td>no</td>
</tr>
<tr>
<td>Mary</td>
<td>-1</td>
<td>-1*(-0.75 <em>1+1.25</em>(-1)-0.75*(-1) -0.75 <em>(-1) –0.75</em>1 )&gt;0</td>
<td>no</td>
</tr>
<tr>
<td>Peter</td>
<td>1</td>
<td>-0.75 <em>1+ 1.25</em>1-0.75* (-1)-0.75* (-1) -0.75 *1 &gt;0</td>
<td>no</td>
</tr>
</tbody>
</table>
Discriminant function is

\[ g(z) = -0.75z_0 + 1.25z_1 - 0.75z_2 - 0.75z_3 - 0.75z_4 \]

Converting back to the original features \( x \)

\[ g(x) = 1.25x_1 - 0.75x_2 - 0.75x_3 - 0.75x_4 - 0.75 \]
Final Classifier

• Trained LDF: \( g(x) = 1.25x_1 - 0.75x_2 - 0.75x_3 - 0.75x_4 - 0.75 \)

• Leads to classifier:
  \[ 1.25x_1 - 0.75x_2 - 0.75x_3 - 0.75x_4 > 0.75 \Rightarrow \text{grade A} \]

• This is just one possible solution vector

• With \( a^{(1)} = [0, 0.5, 0.5, 0, 0] \), solution is \([-1, 1.5, -0.5, -1, -1]\)
  \[ 1.5x_1 - 0.5x_2 - x_3 - x_4 > 1 \Rightarrow \text{grade A} \]

  • in this solution, being tall is the least important feature
1. Classes are linearly separable
   • with fixed learning rate, both single sample and batch versions 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
   • Practical Issue: both single sample and batch algorithms converge faster if features are roughly on the same scale
     • see kNN lecture on feature normalization
## Batch vs. Single Sample Rules

<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, 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>
</tbody>
</table>

### Mini-Batch

- Update weights after seeing `batchSize` examples
- Faster convergence than the Batch rule
- Less susceptible to noisy examples than Single Sample Rule
Linear Classifier: Quadratic Loss

- Other loss functions are possible for our classifier
  \[ f(z^i, a) = \text{sign}(z^i a^i) \]

- Quadratic per-example loss
  \[ L_p(f(z^i, a), z^i) = \frac{1}{2}(y^i - a^i z^i)^2 \]

- Trying to fit labels +1 and -1 to function \( a^t z \)

- This is just standard line fitting in \( \mathbb{R}^n \) (linear regression)
  - note that even correctly classified examples can have a large loss

- Can find optimal weight \( a \) analytically with least squares
  - expensive for large problems

- Gradient descent more efficient for a larger problem
  \[ \nabla L_p(a) = -\sum_{i} (y^i - a^t z^i) z^i \]

- Batch update rule
  \[ a = a + \alpha \sum_{i} (y^i - a^t z^i) z^i \]
**Linear Classifier: Quadratic Loss**

- Quadratic loss is an inferior choice for classification

- Optimal classifier under quadratic loss
  - smallest squared errors
  - one sample misclassified

- Classifier found with Perceptron loss
  - huge squared errors
  - all samples classified correctly

- Idea: instead of trying to get $a^T z$ close to $y$, use some differentiable function $f(a^T z)$ with “squished range”, and try to get $f(a^T z)$ close to $y$
Linear Classifier: Logistic Regression

- Denote classes with 1 and 0 now
  - $y^i = 1$ for positive class, $y^i = 0$ for negative
- Use logistic sigmoid function $\sigma(t)$ for “squishing” $a^t z$

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$

- Despite “regression” in the name, logistic regression is used for classification, not regression
Logistic Regression vs. Regression

\[ \hat{\sigma}(a^Tz) \]

quadratic loss

logistic regression loss
Logistic Regression: Loss Function

• Could use \((y^i - \sigma(a^Tz))^2\) as per-example loss function

• Instead use a different loss
  • if example \(z\) has label 1, want \(\sigma(a^Tz)\) close to 1, define loss as
    \[-\log [\sigma(a^Tz)]\]
  • if example \(z\) has label 0, want \(\sigma(a^Tz)\) close to 0, define loss as
    \[-\log [1-\sigma(a^Tz)]\]
Logistic Regression: Loss Function

- Per-example loss function
  - If example $x$ has label 1, loss is $-\log [\sigma(a^Tz)]$
  - If example $x$ has label 0, loss is $-\log [1-\sigma(a^Tz)]$
- Total loss is sum over per-example losses
- Convex, can be optimized exactly with gradient descent
- Gradient descent batch update rule

$$a = a + \alpha \sum_i (y^i - \sigma(a^Tz^i)) z^i$$

- Logistic Regression has interesting probabilistic interpretation
  - $P(\text{class 1}) = \sigma(a^Tz)$
  - $P(\text{class 0}) = 1 - P(\text{class 1})$
  - Therefore loss function is $-\log P(y)$ (negative log-likelihood)
  - Standard objective in statistics
Logistic Regression vs. Perceptron

- Green example classified correctly, but close to decision boundary
  - Suppose $w^T x = 0.8$ for green example
  - classified correctly, no loss under Perceptron
  - loss of $-\log(\sigma(0.8)) = 0.37$ under logistic regression
  - Logistic Regression (LR) encourages decision boundary move away from any training sample
  - may work better for new samples (better generalization)

- zero Perceptron loss
- smaller LR loss
- zero Perceptron loss
- larger LR loss
- red classifier works better for new data
Linear Classifier: Logistic Regression

- Examples in $Z$, labels in $Y$

\[
Z = \begin{bmatrix}
1 & 2 & 3 \\
1 & 4 & 3 \\
1 & 3 & 5 \\
1 & 1 & 3 \\
1 & 5 & 6
\end{bmatrix}, \quad Y = \begin{bmatrix}
1 \\
1 \\
1 \\
0 \\
0
\end{bmatrix}
\]

- Batch Logistic Regression with learning rate $\alpha=1$

\[
a = \begin{bmatrix}
1 \\
1
\end{bmatrix}
\]

- Initial weights

- This is line $x_1 + x_2 + 1 = 0$
Linear Classifier: Logistic Regression

\[ Z = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 5 \\ 1 & 1 & 3 \\ 1 & 5 & 6 \end{bmatrix} \quad a = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad Y = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \]

- Logistic Regression Batch rule update with \( \alpha = 1 \)

\[ a = a + \sum_i \left( y^i - \sigma(a^t z^i) \right) z^i \]

- Can compute each \( (y^i - \sigma(a^t z^i)) z^i \) with for loop, and add them up

- For \( i = 1 \),

\[ (y^1 - \sigma(a^t z^1)) z^1 = \begin{bmatrix} 1 - \sigma \left( \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \right) \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = (1 - \sigma(6)) \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = 0.0025 \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.0025 \\ 0.005 \\ 0.0075 \end{bmatrix} \]
Linear Classifier: Logistic Regression

\[
Z = \begin{bmatrix}
1 & 2 & 3 \\
1 & 4 & 3 \\
1 & 3 & 5 \\
1 & 1 & 3 \\
1 & 5 & 6 \\
\end{bmatrix} \quad a = \begin{bmatrix}
1 \\
1 \\
\end{bmatrix} \quad Y = \begin{bmatrix}
1 \\
1 \\
0 \\
0 \\
\end{bmatrix}
\]

- Logistic Regression  Batch rule update with \( \alpha = 1 \)

\[
a = a + \sum_i (y^i - \sigma(a^i z^i)) z^i
\]

- But also can compute update with a few lines in Matlab, no need for a loop
- First compute \( a^i z^i \) for all examples

\[
\begin{bmatrix}
 a^1 z^1 \\
 a^2 z^2 \\
 a^3 z^3 \\
 a^4 z^4 \\
 a^5 z^5 \\
\end{bmatrix} = Z \ast a = \begin{bmatrix}
 6 \\
 8 \\
 9 \\
 5 \\
 12 \\
\end{bmatrix}
\]
**Linear Classifier: Logistic Regression**

\[
Z = \begin{bmatrix}
1 & 2 & 3 \\
1 & 4 & 3 \\
1 & 3 & 5 \\
1 & 1 & 3 \\
1 & 5 & 6
\end{bmatrix}
\]

\[
a = \begin{bmatrix}
1 \\
1 \\
-1
\end{bmatrix}
\]

\[
Y = \begin{bmatrix}
1 \\
1 \\
-1 \\
-1
\end{bmatrix}
\]

- **Batch update rule**
  \[
a = a + \sum_i (y^i - \sigma(a^i z^i)) z^i
  \]

- **Apply sigmoid to each row**

\[
\begin{bmatrix}
\sigma(a^1 z^1) \\
\sigma(a^2 z^2) \\
\sigma(a^3 z^3) \\
\sigma(a^4 z^4) \\
\sigma(a^5 z^5)
\end{bmatrix} = \begin{bmatrix}
\sigma(6) \\
\sigma(8) \\
\sigma(9) \\
\sigma(5) \\
\sigma(12)
\end{bmatrix} = \begin{bmatrix}
0.9975 \\
0.9997 \\
0.9999 \\
0.9933 \\
1.000
\end{bmatrix}
\]
Linear Classifier: Logistic Regression

• Assume you have sigmoid function $\sigma(t)$ implemented
  • takes scalar $t$ as an input, outputs $\sigma(t)$
• To apply sigmoid to each element of column vector with one line, use `arrayfun(functionPtr, A)` in matlab

\[
\begin{bmatrix}
\sigma(6) \\
\sigma(8) \\
\sigma(9) \\
\sigma(5) \\
\sigma(12)
\end{bmatrix} =
\begin{bmatrix}
0.9975 \\
0.9997 \\
0.9999 \\
0.9933 \\
1.000
\end{bmatrix}
\]
Linear Classifier: Logistic Regression

\[
Z = \begin{bmatrix}
1 & 2 & 3 \\
1 & 4 & 3 \\
1 & 3 & 5 \\
1 & 1 & 3 \\
1 & 5 & 6
\end{bmatrix}
\]

\[
a = \begin{bmatrix}
1 \\
1 \\
1
\end{bmatrix}
\]

\[
Y = \begin{bmatrix}
1 \\
1 \\
0 \\
0
\end{bmatrix}
\]

- Batch rule update

\[
a = a + \sum_i \left( y^i - \sigma(a^t z^i) \right) z^i
\]

\[
\begin{bmatrix}
\sigma(a^t z^1) \\
\sigma(a^t z^2) \\
\sigma(a^t z^3) \\
\sigma(a^t z^4) \\
\sigma(a^t z^5)
\end{bmatrix} = \begin{bmatrix}
0.9975 \\
0.9997 \\
0.9999 \\
0.9933 \\
1.000
\end{bmatrix}
\]

- Subtract from labels \( Y \)

\[
\begin{bmatrix}
y^1 - \sigma(a^t z^1) \\
y^2 - \sigma(a^t z^2) \\
y^3 - \sigma(a^t z^3) \\
y^4 - \sigma(a^t z^4) \\
y^5 - \sigma(a^t z^5)
\end{bmatrix} = Y - \begin{bmatrix}
0.9975 \\
0.9997 \\
0.9999 \\
0.9933 \\
1.000
\end{bmatrix} = \begin{bmatrix}
0.0025 \\
0.0003 \\
0.0001 \\
-0.9933 \\
-1.000
\end{bmatrix}
\]
Linear Classifier: Logistic Regression

\[
Z = \begin{bmatrix}
1 & 2 & 3 \\
1 & 4 & 3 \\
1 & 3 & 5 \\
1 & 1 & 3 \\
1 & 5 & 6 \\
\end{bmatrix}
\]

\[
a = \begin{bmatrix}
1 \\
1 \\
\end{bmatrix}
\]

\[
Y = \begin{bmatrix}
1 \\
1 \\
0 \\
0 \\
\end{bmatrix}
\]

- Batch rule update

\[
a = a + \sum_i \left( y^i - \sigma(a^t z^i) \right) z^i
\]

\[
v = \begin{bmatrix}
y^1 - \sigma(a^t z^1) \\
y^2 - \sigma(a^t z^2) \\
y^3 - \sigma(a^t z^3) \\
y^4 - \sigma(a^t z^4) \\
y^5 - \sigma(a^t z^5)
\end{bmatrix} = \begin{bmatrix} 0.0025 \\ 0.0003 \\ 0.0001 \\ -0.9933 \\ -1.000 \end{bmatrix}
\]

- Multiply by corresponding example

\[
\begin{bmatrix}
y^1 - \sigma(a^t z^1) \\
y^2 - \sigma(a^t z^2) \\
y^3 - \sigma(a^t z^3) \\
y^4 - \sigma(a^t z^4) \\
y^5 - \sigma(a^t z^5)
\end{bmatrix} = \text{repmat}(v,1,3) \cdot Z
\]

\[
\begin{bmatrix}
0.0025 & 0.0025 & 0.0025 \\
0.0003 & 0.0003 & 0.0003 \\
0.0001 & 0.0001 & 0.0001 \\
-0.99 & -0.99 & -0.99 \\
-1.00 & -1.00 & -1.00
\end{bmatrix}
\]
Linear Classifier: Logistic Regression

\[ Z = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 5 \\ 1 & 1 & 3 \\ 1 & 5 & 6 \end{bmatrix} \]

\[ a = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \]

\[ Y = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \]

- Multiply by corresponding example continued

- Batch rule update

\[
a = a + \sum_i \left( y^i - \sigma(a^t z^i) \right) z^i
\]

\[
v = \begin{bmatrix} y^1 - \sigma(a^t z^1) \\ y^2 - \sigma(a^t z^2) \\ y^3 - \sigma(a^t z^3) \\ y^4 - \sigma(a^t z^4) \\ y^5 - \sigma(a^t z^5) \end{bmatrix} = \begin{bmatrix} 0.0025 \\ 0.0003 \\ 0.0001 \\ -0.9933 \\ -1.000 \end{bmatrix}
\]

\[
\begin{bmatrix} 0.0025 & 0.0025 & 0.0025 \\ 0.0003 & 0.0003 & 0.0003 \\ 0.0001 & 0.0001 & 0.0001 \\ -0.99 & -0.99 & -0.99 \\ -1.00 & -1.00 & -1.00 \end{bmatrix} \times Z = \begin{bmatrix} 0.0025 & 0.0049 & 0.0074 \\ 0.0003 & 0.0013 & 0.001 \\ 0.0001 & 0.0004 & 0.0006 \\ -0.99 & -0.99 & -2.98 \\ -1.00 & -5.0 & -6.0 \end{bmatrix}
\]
**Linear Classifier: Logistic Regression**

- Batch rule update: \[ a = a + \sum_i (y^i - \sigma(a^t z^i)) z^i \]

\[
\begin{bmatrix}
    y^1 - \sigma(a^t z^1)
z^1 \\
    y^2 - \sigma(a^t z^2)
z^2 \\
    y^3 - \sigma(a^t z^3)
z^3 \\
    y^4 - \sigma(a^t z^4)
z^4 \\
    y^5 - \sigma(a^t z^5)
z^5
\end{bmatrix} =
\begin{bmatrix}
    0.0025 & 0.0049 & 0.0074 \\
    0.0003 & 0.0013 & 0.001 \\
    0.0001 & 0.0004 & 0.0006 \\
    -0.99 & -0.99 & -2.98 \\
    -1.00 & -5.0 & -6.0
\end{bmatrix}
= A
\]

- Add up all rows

\[ \text{sum}(A,1) = [-1.99 \ -5.99 \ -8.97] \]

- Transpose to get the needed update

\[
[-1.99 \ -5.99 \ -8.97]^t =
\begin{bmatrix}
    -1.99 \\
    -5.99 \\
    -8.97
\end{bmatrix}
= \sum_i (y^i - \sigma(a^t z^i)) z^i
\]
Linear Classifier: Logistic Regression

- Batch rule update

\[ a = a + \sum_i (y^i - \sigma(a^i z^i)) z^i \]

\[
\begin{bmatrix}
1 \\
1 \\
1
\end{bmatrix}
\begin{bmatrix}
-1.99 \\
-5.99 \\
-8.97
\end{bmatrix}
\]

- Finally update \( a = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} -1.99 \\ -5.99 \\ -8.97 \end{bmatrix} = \begin{bmatrix} -0.99 \\ -4.99 \\ -7.97 \end{bmatrix} \)

- This is line \(-4.99x_1 - 7.97x_2 - 0.99 = 0\)
Logistic Regression vs. Regression vs. Perceptron

- Assuming labels are +1 and -1

- Logistic regression loss
- Perceptron loss
- Quadratic loss

- Misclassified
- Classified correctly but close to decision boundary
- Classified correctly and not too close to the decision boundary
More General Discriminant Functions

- Linear discriminant functions
  - simple decision boundary
  - should try simpler models first to avoid overfitting
  - optimal for certain type of data
  - Gaussian distributions with equal covariance
  - May not be optimal for other data distributions

- Discriminant functions can be more general than linear
  - For example, polynomial discriminant functions
  - Decision boundaries more complex than linear
  - Later will look more at non-linear discriminant functions
Summary

• Linear classifier works well when examples are linearly separable, or almost separable

• Two Linear Classifiers
  • Perceptron
    • find a separating hyperplane in the linearly separable case
    • uses gradient descent for optimization
    • does not converge in the non-separable case
    • can force convergence by using a decreasing learning rate
  • Logistic Regression
    • has probabilistic interpretation
    • can be optimized exactly with gradient descent