Stochastic Gradient Descent

+ 

Probabilistic Learning

(Binary Logistic Regression)
**k-NN Regression**

**Algorithm 1: k=1 Nearest Neighbor Regression**
- **Train**: store all (x, y) pairs
- **Predict**: pick the nearest x in training data and return its y

**Algorithm 2: k=2 Nearest Neighbors Distance Weighted Regression**
- **Train**: store all (x, y) pairs
- **Predict**: pick the nearest two instances \(x^{(n1)}\) and \(x^{(n2)}\) in training data and return the weighted average of their y values

---

**Example**: Dataset with only one feature x and one scalar output y

This version is incorrect.
k-NN Regression

Algorithm 1: k=1 Nearest Neighbor Regression
- **Train**: store all \((x, y)\) pairs
- **Predict**: pick the nearest \(x\) in training data and return its \(y\)

Algorithm 2: k=2 Nearest Neighbors Distance Weighted Regression
- **Train**: store all \((x, y)\) pairs
- **Predict**: pick the nearest two instances \(x^{(n1)}\) and \(x^{(n2)}\) in training data and return the weighted average of their \(y\) values
**k-NN Regression**

**Algorithm 1: k=1 Nearest Neighbor Regression**
- **Train**: store all \((x, y)\) pairs
- **Predict**: pick the nearest \(x\) in training data and return its \(y\)

**Algorithm 2: k=2 Nearest Neighbors Distance Weighted Regression**
- **Train**: store all \((x, y)\) pairs
- **Predict**: pick the nearest two instances \(x^{(n1)}\) and \(x^{(n2)}\) in training data and return the weighted average of their \(y\) values
Reminders

• Practice Problems 1
  – released on course website

• Exam 1: Thu, Feb. 16
  – Time: 6:30 – 8:30pm
  – Location: Your room/seat assignment will be announced on Piazza

• Homework 4: Logistic Regression
  – Out: Fri, Feb 17
  – Due: Sun, Feb. 26 at 11:59pm
OPTIMIZATION METHOD #3: STOCHASTIC GRADIENT DESCENT
Algorithm 1 Gradient Descent

1: procedure GD(D, θ(0))
2: \[ \theta \leftarrow \theta^{(0)} \]
3: while not converged do
4: \[ \theta \leftarrow \theta - \gamma \nabla_\theta \hat{J}(\theta) \]
5: return \( \theta \)
Stochastic Gradient Descent (SGD)

Algorithm 2 Stochastic Gradient Descent (SGD)

1: procedure SGD(\(D, \theta^{(0)}\))
2: \(\theta \leftarrow \theta^{(0)}\)
3: while not converged do
4: \(i \sim \text{Uniform}\{1, 2, \ldots, N\}\)
5: \(\theta \leftarrow \theta - \gamma \nabla_{\theta} J^{(i)}(\theta)\)
6: return \(\theta\)

per-example objective:

\[ J^{(i)}(\theta) \]

original objective:

\[ J(\theta) = \frac{1}{N} \sum_{i=1}^{N} J^{(i)}(\theta) \]
Stochastic Gradient Descent (SGD)

**Algorithm 2** Stochastic Gradient Descent (SGD)

1: **procedure** SGD($\mathcal{D}, \theta^{(0)}$)
2: $\theta \leftarrow \theta^{(0)}$
3: **while** not converged **do**
4:     **for** $i \in \text{shuffle}(\{1, 2, \ldots, N\})$ **do**
5:         $\theta \leftarrow \theta - \gamma \nabla_{\theta} J^{(i)}(\theta)$
6: **return** $\theta$

In practice, it is common to implement SGD using sampling without replacement (i.e. \text{shuffle}(\{1,2,\ldots,N\})), even though most of the theory is for sampling with replacement (i.e. Uniform(\{1,2,\ldots,N\})).

**per-example objective:**

$$J^{(i)}(\theta)$$

**original objective:**

$$J(\theta) = \sum_{i=1}^{N} J^{(i)}(\theta)$$
Background: Probability

Expectation of a function of a random variable

• For any discrete random variable $X$

$$E_X[f(X)] = \sum_{x \in \mathcal{X}} P(X = x)f(x)$$
Why does SGD work?

• If the example is sampled uniformly at random, the expected value of the pointwise gradient is the same as the full gradient!

\[ E[\nabla_{\theta} J^{(i)}(\theta)] = \sum_{i=1}^{N} \left( \text{probability of selecting } x^{(i)}, y^{(i)} \right) \nabla_{\theta} J^{(i)}(\theta) \]

\[ = \sum_{i=1}^{N} \left( \frac{1}{N} \right) \nabla_{\theta} J^{(i)}(\theta) \]

\[ = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} J^{(i)}(\theta) \]

\[ = \nabla_{\theta} J(\theta) \]

• In practice, the data set is randomly shuffled then looped through so that each data point is used equally often
SGD VS. GRADIENT DESCENT
SGD vs. Gradient Descent

Gradient Descent

Stochastic Gradient Descent
SGD vs. Gradient Descent

- Empirical comparison:

![Log-log scale plot](image)

- **Def:** an epoch is a single pass through the training data

1. For GD, only **one update** per epoch
2. For SGD, **N updates** per epoch
   \[ N = (\# \text{train examples}) \]

- SGD reduces MSE much more rapidly than GD
- For GD / SGD, training MSE is initially large due to uninformed initialization
SGD vs. Gradient Descent

• Theoretical comparison:

Define convergence to be when \( J(\theta^{(t)}) - J(\theta^*) < \epsilon \)

<table>
<thead>
<tr>
<th>Method</th>
<th>Steps to Convergence</th>
<th>Computation per Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient descent</td>
<td>( O(\log \frac{1}{\epsilon}) )</td>
<td>( O(NM) )</td>
</tr>
<tr>
<td>SGD</td>
<td>( O(\frac{1}{\epsilon}) )</td>
<td>( O(M) )</td>
</tr>
</tbody>
</table>

(with high probability under certain assumptions)

Main Takeaway: SGD has much slower asymptotic convergence (i.e. it’s slower in theory), but is often much faster in practice.
SGD FOR LINEAR REGRESSION
Linear Regression as Function Approximation

1. Assume $\mathcal{D}$ generated as:
   \[
   x^{(i)} \sim p^*(\cdot) \\
   y^{(i)} = h^*(x^{(i)})
   \]

2. Choose hypothesis space, $\mathcal{H}$:
   all linear functions in $M$-dimensional space
   \[
   \mathcal{H} = \{ h_\theta : h_\theta(x) = \theta^T x, \theta \in \mathbb{R}^M \}
   \]

3. Choose an objective function:
   mean squared error (MSE)
   \[
   J(\theta) = \frac{1}{N} \sum_{i=1}^{N} e_i^2 \\
   = \frac{1}{N} \sum_{i=1}^{N} \left( y^{(i)} - h_\theta(x^{(i)}) \right)^2 \\
   = \frac{1}{N} \sum_{i=1}^{N} \left( y^{(i)} - \theta^T x^{(i)} \right)^2
   \]

4. Solve the unconstrained optimization problem via favorite method:
   - gradient descent
   - closed form
   - stochastic gradient descent
   - ...
   \[
   \hat{\theta} = \arg\min_{\theta} J(\theta)
   \]

5. Test time: given a new $x$, make prediction $\hat{y}$
   \[
   \hat{y} = h_{\hat{\theta}}(x) = \hat{\theta}^T x
   \]
Gradient Calculation for Linear Regression

Derivative of $J^{(i)}(\theta)$:

$$
\frac{d}{d\theta_k} J^{(i)}(\theta) = \frac{1}{2} \frac{d}{d\theta_k} (\theta^T x^{(i)} - y^{(i)})^2 \\
= \frac{1}{2} \frac{d}{d\theta_k} (\theta^T x^{(i)} - y^{(i)}) \left( \sum_{j=1}^{K} \theta_j x_j^{(i)} - y^{(i)} \right) \\
= (\theta^T x^{(i)} - y^{(i)}) x_k^{(i)}
$$

Gradient of $J^{(i)}(\theta)$ (used by SGD):

$$
\nabla_{\theta} J^{(i)}(\theta) = \left[ \frac{d}{d\theta_1} J^{(i)}(\theta), \frac{d}{d\theta_2} J^{(i)}(\theta), \ldots, \frac{d}{d\theta_M} J^{(i)}(\theta) \right] \\
= (\theta^T x^{(i)} - y^{(i)}) x^{(i)}
$$

Derivative of $J(\theta)$:

$$
\frac{d}{d\theta_k} J(\theta) = \sum_{i=1}^{N} \frac{d}{d\theta_k} J^{(i)}(\theta) \\
= \sum_{i=1}^{N} (\theta^T x^{(i)} - y^{(i)}) x_k^{(i)}
$$

Gradient of $J(\theta)$ (used by Gradient Descent):

$$
\nabla_{\theta} J(\theta) = \left[ \sum_{i=1}^{N} (\theta^T x^{(i)} - y^{(i)}) x_1^{(i)}, \sum_{i=1}^{N} (\theta^T x^{(i)} - y^{(i)}) x_2^{(i)}, \ldots, \sum_{i=1}^{N} (\theta^T x^{(i)} - y^{(i)}) x_N^{(i)} \right] \\
= \sum_{i=1}^{N} (\theta^T x^{(i)} - y^{(i)}) x^{(i)}$$
SGD for Linear Regression

SGD applied to Linear Regression is called the “Least Mean Squares” algorithm

Algorithm 1 Least Mean Squares (LMS)

1: procedure LMS($\mathcal{D}$, $\theta^{(0)}$)
2: $\theta \leftarrow \theta^{(0)}$ \hspace{1cm} $\triangleright$ Initialize parameters
3: while not converged do
4: for $i \in \text{shuffle}([1, 2, \ldots, N])$ do
5: $g \leftarrow (\theta^T x^{(i)} - y^{(i)}) x^{(i)}$ \hspace{1cm} $\triangleright$ Compute gradient
6: $\theta \leftarrow \theta - \gamma g$ \hspace{1cm} $\triangleright$ Update parameters
7: return $\theta$
GD for Linear Regression

Gradient Descent for Linear Regression repeatedly takes steps opposite the gradient of the objective function.

Algorithm 1 GD for Linear Regression

1: procedure GDLR(\(\mathcal{D}, \theta^{(0)}\))
2: \(\theta \leftarrow \theta^{(0)}\) \hspace{1cm} \(\triangleright\) Initialize parameters
3: while not converged do
4: \(g \leftarrow \sum_{i=1}^{N} (\theta^T x^{(i)} - y^{(i)}) x^{(i)}\) \hspace{1cm} \(\triangleright\) Compute gradient
5: \(\theta \leftarrow \theta - \gamma g\) \hspace{1cm} \(\triangleright\) Update parameters
6: return \(\theta\)
Solving Linear Regression

Question: Q1  A = toxic  B = True  C = False

True or False: If Mean Squared Error (i.e. \( \frac{1}{N} \sum_{i=1}^{N} (y^{(i)} - h(x^{(i)}))^2 \)) has a unique minimizer (i.e. argmin), then Mean Absolute Error (i.e. \( \frac{1}{N} \sum_{i=1}^{N} |y^{(i)} - h(x^{(i)})| \)) must also have a unique minimizer.

Answer:
Optimization Objectives

You should be able to...

• Apply gradient descent to optimize a function
• Apply stochastic gradient descent (SGD) to optimize a function
• Apply knowledge of zero derivatives to identify a closed-form solution (if one exists) to an optimization problem
• Distinguish between convex, concave, and nonconvex functions
• Obtain the gradient (and Hessian) of a (twice) differentiable function
PROBABILISTIC LEARNING
Probabilistic Learning

Function Approximation
Previously, we assumed that our output was generated using a deterministic target function:

\[
x^{(i)} \sim p^*(\cdot)
\]

\[
y^{(i)} = c^*(x^{(i)})
\]

Our goal was to learn a hypothesis \( h(x) \) that best approximates \( c^*(x) \)

Probabilistic Learning
Today, we assume that our output is sampled from a conditional probability distribution:

\[
x^{(i)} \sim p^*(\cdot)
\]

\[
y^{(i)} \sim p^*(\cdot | x^{(i)})
\]

Our goal is to learn a probability distribution \( p(y|x) \) that best approximates \( p^*(y|x) \)
## Robotic Farming

<table>
<thead>
<tr>
<th>Classification (binary output)</th>
<th>Deterministic</th>
<th>Probabilistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is this a picture of a wheat kernel?</td>
<td>Is this plant drought resistant?</td>
<td></td>
</tr>
</tbody>
</table>

| Regression (continuous output) | How many wheat kernels are in this picture? | What will the yield of this plant be? |
MAXIMUM LIKELIHOOD ESTIMATION
MLE

Suppose we have data $\mathcal{D} = \{x^{(i)}\}_{i=1}^{N}$

**Principle of Maximum Likelihood Estimation:** Choose the parameters that maximize the likelihood of the data.

$$\theta^{\text{MLE}} = \arg\max_{\theta} \prod_{i=1}^{N} p(x^{(i)} | \theta)$$

Maximum Likelihood Estimate (MLE)

\[
\begin{align*}
\theta^{\text{MLE}} &= \arg\max_{\theta} \prod_{i=1}^{N} p(x^{(i)} | \theta) \\
\theta^{\text{MLE}} &= \text{argmax} L(\theta) \\
\theta^{\text{MLE}} &= \text{argmax}_{\theta} L(\theta_1, \theta_2)
\end{align*}
\]
MLE

What does maximizing likelihood accomplish?

• There is only a finite amount of probability mass (i.e. sum-to-one constraint) \( \sum_{x \in X} P(X=x) = 1 \)

• MLE tries to allocate \textbf{as much} probability mass \textbf{as possible} to the things we have observed…

…\textbf{at the expense} of the things we have not observed
Maximum Likelihood Estimation

The principle is Maximum Likelihood estimator (MLE):

Choose parameters that make the data "most likely".

**Assumptions:** Data generated iid from distribution $p^*(X | \theta^*)$ and comes from a family of distributions parameterized

$\theta \in \Theta$ - set of possible parameters

**Formally:**

$$\hat{\theta}_{MLE} = \underset{\theta \in \Theta}{\text{argmax}} \ p(D | \theta)$$

$$= \underset{\theta \in \Theta}{\text{argmax}} \ \log p(D | \theta)$$

$$= \underset{\theta \in \Theta}{\text{argmax}} \ l(\theta)$$

where $l(\theta) = \log p(D | \theta)$ - log-likelihood

Usually a continuous optimization

$\log$ is monotonic

$treat$ as function $l(\theta)$

where $D$ is constant
MOTIVATION:
LOGISTIC REGRESSION
Example: Image Classification

- ImageNet LSVRC-2010 contest:
  - **Dataset**: 1.2 million labeled images, 1000 classes
  - **Task**: Given a new image, label it with the correct class
  - **Multiclass** classification problem
- Examples from http://image-net.org/
Bird
Warm-blooded egg-laying vertebrates characterized by feathers and forelimbs modified as wings

- marine animal, marine creature, sea animal, sea creature (1)
- scavenger (1)
- biped (0)
- predator, predatory animal (1)
- larva (49)
- acrodont (0)
- feeder (0)
- stilt (0)
- chordate (3087)
- tunicate, urochordate, urochord (6)
- cephalochordate (1)
- vertebrate, craniate (3077)
- mammal, mammalian (1169)
- bird (871)
- dickeybird, dicky-bird, dickybird, dicky-bird (0)
- cock (1)
- hen (0)
- nester (0)
- night bird (1)
- bird of passage (0)
- protoavis (0)
- archaeopteryx, archeopteryx, Archaeopteryx lithographi
- Sinornis (0)
- Iberomosornis (0)
- archaeornis (0)
- ratite, ratite bird, flightless bird (10)
- carinate, carinate bird, flying bird (0)
- passerine, passeriform bird (279)
- nonpasserine bird (0)
- bird of prey, raptor, raptorial bird (80)
- gallinaceous bird, gallinacean (114)
German iris, Iris kochii
Iris of northern Italy having deep blue-purple flowers; similar to but smaller than Iris germanica
Court, courtyard
An area wholly or partly surrounded by walls or buildings; "the house was built around an inner court"

Numbers in brackets: (the number of synsets in the subtree).

- ImageNet 2011 Fall Release (32326)
  - plant, flora, plant life (4486)
  - geological formation, formation (175)
  - natural object (1112)
  - sport, athletics (176)
  - artifact, artefact (10504)
  - instrumentality, instrumentation (5494)
    - structure, construction (1405)
      - airdock, hangar, repair shed (0)
      - altar (1)
      - arcade, colonnade (1)
      - arch (31)
    - area (344)
      - aisle (0)
      - auditorium (1)
      - baggage claim (0)
      - box (1)
      - breakfast area, breakfast nook (0)
      - bullpen (0)
      - chancel, sanctuary, bema (0)
      - choir (0)
      - corner, nook (2)
    - court, courtyard (6)
      - atrium (0)
      - bailey (0)
      - cloister (0)
      - food court (0)
      - forecourt (0)
      - narvis (0)
Example: Image Classification

CNN for Image Classification
(Krizhevsky, Sutskever & Hinton, 2011)
17.5% error on ImageNet LSVRC-2010 contest

Input image (pixels)

• Five convolutional layers (w/max-pooling)
• Three fully connected layers

1000-way softmax

Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network’s input is 150,528-dimensional, and the number of neurons in the network’s remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Reducing Overfitting

Our neural network architecture has 60 million parameters. Although the 1000 classes of ILSVRC make each training example impose 10 bits of constraint on the mapping from image to label, this turns out to be insufficient to learn so many parameters without considerable overfitting. Below, we describe the two primary ways in which we combat overfitting.

4.1 Data Augmentation

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations (e.g., [25, 4, 5]). We employ two distinct forms of data augmentation, both of which allow transformed images to be produced from the original images with very little computation, so the transformed images do not need to be stored on disk. In our implementation, the transformed images are generated in Python code on the CPU while the GPU is training on the previous batch of images. So these data augmentation schemes are, in effect, computationally free.

The first form of data augmentation consists of generating image translations and horizontal reflections. We do this by extracting random \( 224 \times 224 \) patches (and their horizontal reflections) from the \( 256 \times 256 \) images and training our network on these extracted patches. This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly interdependent. Without this scheme, our network suffers from substantial overfitting, which would have forced us to use much smaller networks. At test time, the network makes a prediction by extracting five \( 224 \times 224 \) patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network’s softmax layer on the ten patches.

The second form of data augmentation consists of altering the intensities of the RGB channels in training images. Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components.

This is the reason why the input images in Figure 2 are \( 224 \times 224 \times 3 \)-dimensional.
Example: Image Classification

CNN for Image Classification
(Krizhevsky, Sutskever & Hinton, 2011)
17.5% error on ImageNet LSVRC-2010 contest

Input image (pixels)

• Five convolutional layers (w/max-pooling)
• Three fully connected layers

The rest is just some fancy feature extraction (discussed later in the course)

This “softmax” layer is Logistic Regression!

1000-way softmax

4 Reducing Overfitting

Our neural network architecture has 60 million parameters. Although the 1000 classes of ILSVRC make each training example impose 10 bits of constraint on the mapping from image to label, this turns out to be insufficient to learn so many parameters without considerable overfitting. Below, we describe the two primary ways in which we combat overfitting.

4.1 Data Augmentation

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations (e.g., [25, 4, 5]). We employ two distinct forms of data augmentation, both of which allow transformed images to be produced from the original images with very little computation, so the transformed images do not need to be stored on disk. In our implementation, the transformed images are generated in Python code on the CPU while the GPU is training on the previous batch of images. So these data augmentation schemes are, in effect, computationally free.

The first form of data augmentation consists of generating image translations and horizontal reflections. We do this by extracting random $224 \times 224$ patches (and their horizontal reflections) from the $256 \times 256$ images and training our network on these extracted patches. This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly interdependent. Without this scheme, our network suffers from substantial overfitting, which would have forced us to use much smaller networks. At test time, the network makes a prediction by extracting five $224 \times 224$ patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network’s softmax layer on the ten patches.

The second form of data augmentation consists of altering the intensities of the RGB channels in training images. Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components.
LOGISTIC REGRESSION
Logistic Regression

**Data:** Inputs are continuous vectors of length M. Outputs are discrete.

\[
\mathcal{D} = \{x^{(i)}, y^{(i)}\}_{i=1}^{N} \text{ where } x \in \mathbb{R}^M \text{ and } y \in \{0, 1\}
\]

We are back to classification.

Despite the name logistic regression.
Linear Models for Classification

Key idea: Try to learn this hyperplane directly

Directly modeling the hyperplane would use a decision function:

\[ h(x) = \text{sign}(\theta^T x) \]

for:

\[ y \in \{-1, +1\} \]

Looking ahead:

- We’ll see a number of commonly used Linear Classifiers
- These include:
  - Perceptron
  - Logistic Regression
  - Naïve Bayes (under certain conditions)
  - Support Vector Machines

Recall...
Background: Hyperplanes

Notation Trick: fold the bias $b$ and the weights $w$ into a single vector $\theta$ by prepending a constant to $x$ and increasing dimensionality by one to get $x'$!

Hyperplane (Definition 1):
$$\mathcal{H} = \{x : w^T x + b = 0\}$$

Hyperplane (Definition 2):
$$\mathcal{H} = \{x' : \theta^T x' = 0 \text{ and } x'_0 = 1\}$$
$$\theta = [b, w_1, \ldots, w_M]^T$$
$$x' = [1, x_1, \ldots, x_M]^T$$

Half-spaces:
$$\mathcal{H}^+ = \{x : \theta^T x > 0 \text{ and } x'_0 = 1\}$$
$$\mathcal{H}^- = \{x : \theta^T x < 0 \text{ and } x'_0 = 1\}$$
Using gradient ascent for linear classifiers

Key idea behind today’s lecture:

1. Define a linear classifier (logistic regression)
2. Define an objective function (likelihood)
3. Optimize it with gradient descent to learn parameters
4. Predict the class with highest probability under the model
Optimization for Linear Classifiers

Whiteboard

– Strawman: Mean squared error for Perceptron!
– What does $\theta^T x$ tell us about $x$?
Using gradient ascent for linear classifiers

Suppose we wanted to learn a linear classifier, but instead of predicting \( y \in \{-1, +1\} \) we wanted to predict \( y \in \{0, 1\} \)

\[
h(x) = \text{sign}(\theta^T x)
\]
Using gradient ascent for linear classifiers

Suppose we wanted to learn a linear classifier, but instead of predicting \( y \in \{-1,+1\} \) we wanted to predict \( y \in \{0,1\} \)

\[
h(x) = \text{"sign"}(\theta^T x)
\]

**Goal:** Learn a linear classifier with Gradient Descent
Using gradient ascent for linear classifiers

But this decision function isn’t differentiable...

\[ h(x) = \text{“sign”}(\theta^T x) \]

Use a differentiable function instead!

\[
p_\theta(y = 1 | x) = \frac{1}{1 + \exp(-\theta^T x)}
\]
Logistic Regression

**Data:** Inputs are continuous vectors of length $M$. Outputs are discrete.

$$\mathcal{D} = \{x^{(i)}, y^{(i)}\}_{i=1}^{N} \text{ where } x \in \mathbb{R}^{M} \text{ and } y \in \{0, 1\}$$

**Model:** Logistic function applied to dot product of parameters with input vector.

$$p_{\theta}(y = 1|x) = \frac{1}{1 + \exp(-\theta^T x)}$$

**Learning:** finds the parameters that minimize some objective function.

$$\theta^* = \arg\min_{\theta} J(\theta)$$

**Prediction:** Output is the most probable class.

$$\hat{y} = \arg\max_{y \in \{0, 1\}} p_{\theta}(y|x)$$
Logistic Regression

Whiteboard

- Logistic Regression Model
- Partial derivative for logistic regression
- Gradient for logistic regression
- Decision boundary