Stochastic Gradient Descent
+
Probabilistic Learning
(Binary Logistic Regression)
Reminders

• Practice for Exam 1
  – Mock Exam 1
    • Due: Wed, Feb. 16 at 11:59pm
    • See @683 for participation point details
  – Practice Problems 1 released on course website

• Exam 1: Thu, Feb. 17
  – Time: 6:30 – 8:30pm
  – Location: Your room/seat assignment will be announced on Piazza

• Homework 4: Logistic Regression
  – Out: Fri, Feb 18
  – Due: Sun, Feb. 27 at 11:59pm
OPTIMIZATION METHOD #3:
STOCHASTIC GRADIENT DESCENT
Algorithm 1 Gradient Descent

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

per-example objective:

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

original objective:

$$J(\theta) = \sum_{i=1}^{N} J^{(i)}(\theta)$$
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 \( i \in \) shuffle(\( \{1, 2, \ldots, N\} \) ) do
5: \( \theta \leftarrow \theta - \gamma \nabla \theta J^{(i)}(\theta) \)
6: return \( \theta \)

per-example objective:
\[
J^{(i)}(\theta)
\]
original objective:
\[
J(\theta) = \sum_{i=1}^{N} J^{(i)}(\theta)
\]

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. \( \text{Uniform}(\{1,2,\ldots,N\}) \)).
Why does SGD work?

Whiteboard

– Example of SGD on simple 2D function
Why does SGD work?

\[
\frac{dJ(\theta)}{d\theta_j} = \frac{1}{N} \sum_{i=1}^{N} \frac{d}{d\theta_j} (J_i(\theta))
\]

\[
\nabla J(\theta) = \begin{bmatrix}
\vdots \\
\vdots \\
\vdots
\end{bmatrix} = \frac{1}{N} \sum_{i=1}^{N} \nabla J_i(\theta)
\]

Recall: for any discrete r.v. \( X \)

\[
\mathbb{E}_X \sum f(x) I \overset{\Delta}{=} \sum_x P(x=x) f(x)
\]

Q: What is the expected value of a randomly chosen \( \nabla J_i(\theta) \)?

Let \( I \sim \text{Uniform}(1, \ldots, N) \)

\[
P(I=i) = \frac{1}{N} \quad \text{if} \; i \in \{1, \ldots, N\}
\]

\[
\mathbb{E}_I [\nabla J_I(\theta)] = \sum_{i=1}^{N} P(I=i) \nabla J_i(\theta)
\]

\[
= \frac{1}{N} \sum_{i=1}^{N} \nabla J_i(\theta)
\]

\[
= \nabla J(\theta)
\]
LINEAR REGRESSION: PRACTICALITIES
Empirical Convergence

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

**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.
Convergence of Optimizers

<table>
<thead>
<tr>
<th>Methods</th>
<th>Steps to Converge</th>
<th>Computation per iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton's Method</td>
<td>$O((\ln \ln \frac{1}{\epsilon})^2)$</td>
<td>$O(NM^2)$</td>
</tr>
<tr>
<td>GD</td>
<td>$O(\ln \frac{1}{\epsilon})$</td>
<td>$O(NM)$</td>
</tr>
<tr>
<td>SGD</td>
<td>$O(\frac{1}{\epsilon})$</td>
<td>$O(M)$</td>
</tr>
</tbody>
</table>

**Convergence Analysis:**

Def: Convergence is when $J(\theta) - J(\theta^*) < \epsilon$

"almost sure" convergence and conditions

*Takeaway:* SGD has much slower asymptotic convergence, but is often 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 $J^{(i)}(\theta)$:

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

[used by SGD]

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

$$
\nabla_{\theta} J^{(i)}(\theta) = \begin{bmatrix}
\frac{d}{d\theta_1} J^{(i)}(\theta) \\
\frac{d}{d\theta_2} J^{(i)}(\theta) \\
\vdots \\
\frac{d}{d\theta_M} J^{(i)}(\theta)
\end{bmatrix} = \begin{bmatrix}
(\theta^T x^{(i)} - y^{(i)}) x_1^{(i)} \\
(\theta^T x^{(i)} - y^{(i)}) x_2^{(i)} \\
\vdots \\
(\theta^T x^{(i)} - y^{(i)}) x_N^{(i)}
\end{bmatrix}
= (\theta^T x^{(i)} - y^{(i)}) x^{(i)}
$$

[used by SGD]

Derivative of $J(\theta)$:

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

[used by Gradient Descent]

Gradient of $J(\theta)$

$$
\nabla_{\theta} J(\theta) = \begin{bmatrix}
\frac{d}{d\theta_1} J(\theta) \\
\frac{d}{d\theta_2} J(\theta) \\
\vdots \\
\frac{d}{d\theta_M} J(\theta)
\end{bmatrix} = \begin{bmatrix}
\sum_{i=1}^{N} (\theta^T x^{(i)} - y^{(i)}) x_1^{(i)} \\
\sum_{i=1}^{N} (\theta^T x^{(i)} - y^{(i)}) x_2^{(i)} \\
\vdots \\
\sum_{i=1}^{N} (\theta^T x^{(i)} - y^{(i)}) x_N^{(i)}
\end{bmatrix}
= \sum_{i=1}^{N} (\theta^T x^{(i)} - y^{(i)}) x^{(i)}
$$

[used by Gradient Descent]
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 \text{Initialize parameters}
3: \textbf{while} not converged \textbf{do}
4: \hspace{1cm} \textbf{for} \( i \in \text{shuffle}([1, 2, \ldots, N]) \) \textbf{do}
5: \hspace{2cm} \( g \leftarrow (\theta^T x^{(i)} - y^{(i)}) x^{(i)} \) \hspace{1cm} \triangleright \text{Compute gradient}
6: \hspace{2cm} \theta \leftarrow \theta - \gamma g \hspace{1cm} \triangleright \text{Update parameters}
7: \hspace{1cm} \textbf{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: \(\theta \leftarrow \theta^{(0)}\) ▶ Initialize parameters
3: while not converged do
4: \(g \leftarrow \sum_{i=1}^{N}(\theta^T x^{(i)} - y^{(i)})x^{(i)}\) ▶ Compute gradient
5: \(\theta \leftarrow \theta - \gamma g\) ▶ Update parameters
6: return \(\theta\)
Linear Regression Objectives

You should be able to...

• Design k-NN Regression and Decision Tree Regression

• Implement learning for Linear Regression using three optimization techniques: (1) closed form, (2) gradient descent, (3) stochastic gradient descent

• Choose a Linear Regression optimization technique that is appropriate for a particular dataset by analyzing the tradeoff of computational complexity vs. convergence speed

• Distinguish the three sources of error identified by the bias-variance decomposition: bias, variance, and irreducible error.
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></th>
<th>Deterministic</th>
<th>Probabilistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classification</td>
<td>Is this a picture of a wheat kernel?</td>
<td>Is this plant drought resistant?</td>
</tr>
<tr>
<td>(binary output)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Regression</td>
<td>How many wheat kernels are in this picture?</td>
<td>What will the yield of this plant be?</td>
</tr>
<tr>
<td>(continuous output)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MAXIMUM LIKELIHOOD ESTIMATION
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)
MLE

What does maximizing likelihood accomplish?

• There is only a finite amount of probability mass (i.e. sum-to-one constraint)

• MLE tries to allocate as much probability mass as possible to the things we have observed...

...at the expense of the things we have not observed
Maximum Likelihood Estimation

The principle of maximum likelihood estimation (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:

$$
\hat{\theta}_{MLE} = \arg \max_{\theta \in \Theta} p(D|\theta)
$$

$$
= \arg \max_{\theta \in \Theta} \log p(D|\theta)
$$

$$
= \arg \max_{\theta \in \Theta} l(\theta)
$$

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

Since log is monotonic function:

$$
\log(a_1) < \log(a_2) \iff a_1 < a_2
$$

Treat as function of $\theta$ where $D$ is constant.

Usually a continuous optimization problem.
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)
- stunt (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)
  - Iberomesornis (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, gallinaceous (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)
            - airport, 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)
            - perris (0)
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.

The second convolutional layer takes as input the (response-normalized and pooled) output of the first convolutional layer and filters it with 256 kernels of size $5 \times 5 \times 48$.

The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3 \times 256$ connected to the (normalized, pooled) outputs of the second convolutional layer. The fourth convolutional layer has 384 kernels of size $3 \times 3 \times 192$, and the fifth convolutional layer has 256 kernels of size $3 \times 3 \times 192$.

The fully-connected layers have 4096 neurons each.

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.

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

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

1000-way softmax

Input image (pixels)

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

This “softmax” layer is Logistic Regression!
LOGISTIC REGRESSION
Logistic Regression

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

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

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

Directly modeling the hyperplane would use a decision function:

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

for:

$$ y \in \{-1, +1\} $$
**Background: Hyperplanes**

Hyperplane (Definition 1):
\[ \mathcal{H} = \{ \mathbf{x} : \mathbf{w}^T \mathbf{x} = b \} \]

Hyperplane (Definition 2):
\[ \mathcal{H} = \{ \mathbf{x}' : \theta^T \mathbf{x}' = 0 \text{ and } x'_1 = 1 \} \]

\[ \theta = [b, w_1, \ldots, w_M]^T \]
\[ \mathbf{x}' = [1, x_1, \ldots, x_M]^T \]

**Notation Trick:** fold the bias \( b \) and the weights \( \mathbf{w} \) into a single vector \( \theta \) by prepending a constant to \( \mathbf{x} \) and increasing dimensionality by one to get \( \mathbf{x}' \! \).
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...

$$h(x) = \text{“sign”}(\theta^Tx)$$

Use a differentiable function instead!

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

$$\text{logistic}(u) \equiv \frac{1}{1 + e^{-u}}$$
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.

$$\theta^* = \operatorname{argmin}_{\theta} J(\theta)$$

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

$$\hat{y} = \operatorname{argmax}_{y \in \{0, 1\}} p_\theta(y | x)$$
Logistic Regression

Whiteboard

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