Neural Network
(Basic Ideas)
Hung-yi Lee
Learning ≈ Looking for a Function

• Speech Recognition
  \[ f(\text{声波}) = \text{“你好”} \]

• Handwritten Recognition
  \[ f(2) = \text{“2”} \]

• Weather forecast
  \[ f(\text{今天天气}) = \text{“明天晴朗”} \]

• Play video games
  \[ f(\text{敌人位置和数量}) = \text{“发射”} \]
Framework

Hypothesis Function Set $f_1, f_2, \ldots$

Training: Pick the best Function $f^*$

Testing: $f^*(x') = y'$

$x$ : function input
$\hat{y}$ : function output

$\{(x^1, \hat{y}^1), (x^2, \hat{y}^2), \ldots\}$

$\hat{y}$ : “2” (label)
Outline

1. What is the model (function hypothesis set)?

2. What is the “best” function?

3. How to pick the “best” function?
Task Considered Today

- **Classification**

**Binary Classification**

- Only two classes

  - **input object**
    - Class A (yes)
    - Class B (no)

- **Spam filtering**
  - Is an e-mail spam or not?

- **Recommendation systems**
  - recommend the product to the customer or not?

- **Malware detection**
  - Is the software malicious or not?

- **Stock prediction**
  - Will the future value of a stock increase or not?
Task Considered Today

- Classification

**Binary Classification**
Only two classes

- Class A (yes)
- Class B (no)

**Multi-class Classification**
More than two classes

- Class A
- Class B
- Class C
- ...
Multi-class Classification

• Handwriting Digit Classification

  Input: 

  ![Image of a handwritten digit]
  
  Class: “1”, “2”, ....., “9”, “0”
  10 classes

• Image Recognition

  Input: 

  ![Image of a cat]
  
  Class: “dog”, “cat”, “book”, ....
  Thousands of classes
Multi-class Classification

• **Real** speech recognition is not multi-class classification

  Input:

  Classes: “hi”, “how are you”, “I am sorry” ........

  Cannot be enumerated

• The HW1 is multi-class classification

  The frame belongs to which **phoneme**.

  Classes are the phonemes.
1. What is the model?
What is the function we are looking for?

• **classification**

\[ y = f(x) \quad \quad f: R^N \rightarrow R^M \]

• x: input object to be classified
• y: class

• **Assume both x and y can be represented as fixed-size vector**
  • x is a vector with N dimensions, and y is a vector with M dimensions
What is the function we are looking for?

- **Handwriting Digit Classification**

\[ f : \mathbb{R}^N \rightarrow \mathbb{R}^M \]

**x:** image

16 x 16

Each pixel corresponds to an element in the vector

\[
\begin{bmatrix}
0 \\
1 \\
\vdots
\end{bmatrix}
\]

1: for ink, 0: otherwise

16 x 16 = 256 dimensions

**y:** class

10 dimensions for digit recognition

1: for ink, 0: otherwise

“1”, “2”, “3” or not

```
1
0
0
```

```
0
1
0
```

```
0
0
```

```
```

“1” “1” or not

“2” “2” or not

“3” “3” or not
1. What is the model?

A Layer of Neuron
Single Neuron

\[ f : \mathbb{R}^N \rightarrow \mathbb{R} \]

\[
\begin{align*}
    z &= w_1 x_1 + w_2 x_2 + \cdots + w_N x_N + b \\
    \sigma(z) &= \frac{1}{1 + e^{-z}}
\end{align*}
\]

Activation function

\[ \sigma(z) \]

\[ \sigma(z) \rightarrow y \]
Single Neuron

\[ f: \mathbb{R}^N \rightarrow \mathbb{R} \]

Activation function

\[ \sigma(z) = \frac{1}{1 + e^{-z}} \]
Single Neuron \( f: \mathbb{R}^N \rightarrow \mathbb{R} \)

- Single neuron can only do binary classification, cannot handle multi-class classification

\[ f: \mathbb{R}^N \rightarrow \mathbb{R} \]

\[
\begin{cases}
  \text{is } "2" & y \geq 0.5 \\
  \text{not } "2" & y < 0.5
\end{cases}
\]
A Layer of Neuron \( f: R^N \rightarrow R^M \)

- Handwriting digit classification
  - Classes: “1”, “2”, ..., “9”, “0”
  - 10 classes

If \( y_2 \) is the max, then the image is “2”.

10 neurons
1. What is the model?

Limitation of Single Layer
Limitation of Single Layer

\[ z = w_1 x_1 + w_2 x_2 + b \]

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1)</td>
<td>(x_2)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ x_2 \]

\[ x_1 \]

\( \begin{cases} \text{yes} & a \geq \text{threshold} \\ \text{no} & a < \text{threshold} \end{cases} \)

Can we?
Limitation of Single Layer

- No, we can’t ......
Limitation of Single Layer

\[ a = \text{and}(x_1, x_2) + \text{and}(w_1, b) + \text{and}(w_2, z) \]

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 )</td>
<td>( x_2 )</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ z = x_1 + w_1 \]

\[ a = \text{or}(z, b) \]
Neural Network

Hidden Neurons

$1 \, a \, 1 \, + \, 1 \, z \, 1 \, + \, 1 \, x \, 1 \, 2 \, x \, 2 \, z \, 2 \, a \, 1 \, + \, z \, a \, 2 \, + \, 1 \, a$
\[ a_1 = 0.73 \]

\[ a_1 = 0.27 \]

\[ a_1 = 0.05 \]
\[ a_1 = 0.27 \quad \text{and} \quad a_2 = 0.73 \]

\[ a_1 = 0.73 \quad \text{and} \quad a_2 = 0.05 \]
1. What is the model?

Neural Network
Neural Network as Model

$$f : \mathbb{R}^N \to \mathbb{R}^M$$

- Fully connected feedforward network
- Deep Neural Network: many hidden layers
Notation

Layer $l - 1$
$N_{l-1}$ nodes

Layer $l$
$N_l$ nodes

Output of a neuron:
$\mathbf{a}_i^l$ Neuron $i$

Output of one layer:
$\mathbf{a}^l$ : a vector

Layer $l$
Notation

Layer $l-1$ to Layer $l$

$W_{ij}^{l}$

from neuron $j$ (Layer $l-1$) to neuron $i$ (Layer $l$)

$N_{l-1}$

$N_{l}$

$W^{l} = \begin{bmatrix}
    w_{11}^{l} & w_{12}^{l} & \cdots \\
    w_{21}^{l} & w_{22}^{l} & \cdots \\
    \vdots & \vdots & \ddots
\end{bmatrix}$
Notation

Layer $l - 1$ nodes

$N_{l-1}$

Layer $l$ nodes

$N_l$

bias for neuron $i$ at layer $l$

$b_i^l$

bias for all neurons in layer $l$

$b^l = \begin{bmatrix}
  b_1^l \\
  b_2^l \\
  \vdots \\
  b_i^l \\
  \vdots 
\end{bmatrix}$
Notation

$z_i^l$ : input of the activation function for neuron $i$ at layer $l$

$z_i^l$ : input of the activation function all the neurons in layer $l$

$$z_i^l = w_{i1}^l a_1^{l-1} + w_{i2}^l a_2^{l-1} \ldots + b_i^l$$

$$z_i^l = \sum_{j=1}^{N_{l-1}} w_{ij}^l a_j^{l-1} + b_i^l$$
Notation - Summary

\( a_i^l \): output of a neuron

\( W_{ij}^l \): a weight

\( a^l \): output of a layer

\( W^l \): a weight matrix

\( z_{ij}^l \): input of activation function

\( b_i^l \): a bias

\( z^l \): input of activation function for a layer

\( b^l \): a bias vector
Relations between Layer Outputs

Layer $l - 1$  
$N_{l-1}$ nodes

Layer $l$  
$N_l$ nodes
Relations between Layer Outputs

Layer $l - 1$

$N_{l-1}$ nodes

Layer $l$

$N_l$ nodes

$z_1^l = w_{11}^l a_1^{l-1} + w_{12}^l a_2^{l-1} + \cdots + b_1^l$

$z_2^l = w_{21}^l a_1^{l-1} + w_{22}^l a_2^{l-1} + \cdots + b_2^l$

$\vdots$

$z_i^l = w_{i1}^l a_1^{l-1} + w_{i2}^l a_2^{l-1} + \cdots + b_i^l$

$\vdots$

$$
\begin{bmatrix}
  z_1^l \\
  z_2^l \\
  \vdots \\
  z_i^l \\
  \vdots
\end{bmatrix}
= 
\begin{bmatrix}
  w_{11}^l & w_{12}^l & \cdots \\
  w_{21}^l & w_{22}^l & \cdots \\
  \vdots & \vdots & \ddots \\
  w_{i1}^l & w_{i2}^l & \cdots
\end{bmatrix}
\begin{bmatrix}
  a_1^{l-1} \\
  a_2^{l-1} \\
  \vdots \\
  a_i^{l-1} \\
  \vdots
\end{bmatrix}
+ 
\begin{bmatrix}
  b_1^l \\
  b_2^l \\
  \vdots \\
  b_i^l \\
  \vdots
\end{bmatrix}
$$

$z^l = W^l a^{l-1} + b^l$
Relations between Layer Outputs

Layer $l-1$ nodes

$N_{l-1}$ nodes

Layer $l$ nodes

$N_l$ nodes

$\mathbf{a}_i^l = \sigma(\mathbf{z}_i^l)$

$$
\begin{bmatrix}
a_1^l \\
a_2^l \\
\vdots \\
a_i^l \\
\vdots
\end{bmatrix} =
\begin{bmatrix}
\sigma(z_1^l) \\
\sigma(z_2^l) \\
\vdots \\
\sigma(z_i^l) \\
\vdots
\end{bmatrix}
$$

$\mathbf{a}_l = \sigma(\mathbf{z}_l)$
Relations between Layer Outputs

\[
z^l = W^l a^{l-1} + b^l
\]
\[
a^l = \sigma(z^l)
\]
\[
a^l = \sigma(W^l a^{l-1} + b^l)
\]
Function of Neural Network

Input | Layer 1 | Layer 2 | Layer L | Output

\[ x \]

\[ W^1, b^1 \]
\[ W^2, b^2 \]
\[ W^L, b^L \]

\[ a^1 \]
\[ a^2 \]
\[ a^L \]

\[ y_1 \]
\[ y_2 \]
\[ y_M \]

\[ \sigma(W^1 x + b^1) = a^1 \]
\[ \sigma(W^2 a^1 + b^2) = a^2 \]
\[ \sigma(W^L a^{L-1} + b^L) = a^L = y \]
Function of Neural Network

\[ y = f(x) = \sigma(W^L \ldots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \ldots + b^L) \]
2. What is the “best” function?
Best Function = Best Parameters

$$y = f(x) = \sigma(W^L \ldots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \ldots + b^L)$$

because different parameters $W$ and $b$ lead to different function

Formal way to define a function set:

$$f(x; \theta) \rightarrow \text{parameter set}$$

$$\theta = \{W^1, b^1, W^2, b^2 \ldots W^L, b^L\}$$

Pick the “best” function $f^*$

Pick the “best” parameter set $\theta^*$
Cost Function

• Define a function for parameter set $C(\theta)$
  • $C(\theta)$ evaluate how bad a parameter set is
  • The best parameter set $\theta^*$ is the one that minimizes $C(\theta)$
    \[
    \theta^* = \arg \min_{\theta} C(\theta)
    \]

• $C(\theta)$ is called \textit{cost/loss/error function}
  • If you define the goodness of the parameter set by another function $O(\theta)$
  • $O(\theta)$ is called objective function
Cost Function

Given training data:
\[ \{(x^1, \hat{y}^1)\cdots(x^r, \hat{y}^r)\cdots(x^R, \hat{y}^R)\} \]

- **Handwriting Digit Classification**

sum over all training examples

\[
C(\theta) = \frac{1}{R} \sum_r \left\| f(x^r; \theta) - \hat{y}^r \right\|
\]

Minimize distance
3. How to pick the “best” function?

Gradient Descent
Statement of Problems

• Statement of problems:
  • There is a function $C(\theta)$
    • $\theta$ represents parameter set
      • $\theta = \{\theta_1, \theta_2, \theta_3, \ldots\}$
    • Find $\theta^*$ that minimizes $C(\theta)$
  • Brute force?
    • Enumerate all possible $\theta$
  • Calculus?
    • Find $\theta^*$ such that
      \[
      \frac{\partial C(\theta)}{\partial \theta_1} \bigg|_{\theta=\theta^*} = 0, \quad \frac{\partial C(\theta)}{\partial \theta_2} \bigg|_{\theta=\theta^*} = 0, \ldots
      \]
Gradient Descent – Idea

• For simplification, first consider that $\theta$ has only one variable

Drop a ball somewhere ...
When the ball stops, we find the local minima
Gradient Descent – Idea

- For simplification, first consider that $\theta$ has only one variable

- Randomly start at $\theta^0$
- Compute $dC(\theta^0)/d\theta$
- $\theta^1 \leftarrow \theta^0 - \eta \frac{dC(\theta^0)}{d\theta}$
- Compute $dC(\theta^1)/d\theta$
- $\theta^2 \leftarrow \theta^1 - \eta \frac{dC(\theta^1)}{d\theta}$
- ......
Gradient Descent

• Suppose that $\theta$ has two variables $\{\theta_1, \theta_2\}$

- Randomly start at $\theta^0 = \begin{bmatrix} \theta_1^0 \\ \theta_2^0 \end{bmatrix}$
- Compute the gradients of $C(\theta)$ at $\theta^0$: $\nabla C(\theta^0) = \begin{bmatrix} \partial C(\theta_1^0)/\partial \theta_1 \\ \partial C(\theta_2^0)/\partial \theta_2 \end{bmatrix}$
- Update parameters

  $\begin{bmatrix} \theta_1^1 \\ \theta_2^1 \end{bmatrix} = \begin{bmatrix} \theta_1^0 \\ \theta_2^0 \end{bmatrix} - \eta \begin{bmatrix} \partial C(\theta_1^0)/\partial \theta_1 \\ \partial C(\theta_2^0)/\partial \theta_2 \end{bmatrix} \quad \Rightarrow \quad \theta^1 = \theta^0 - \eta \nabla C(\theta^0)$

- Compute the gradients of $C(\theta)$ at $\theta^1$: $\nabla C(\theta^1) = \begin{bmatrix} \partial C(\theta_1^1)/\partial \theta_1 \\ \partial C(\theta_2^1)/\partial \theta_2 \end{bmatrix}$
- ......
Gradient Descent

Start at position $\theta^0$

Compute gradient at $\theta^0$

Move to $\theta^1 = \theta^0 - \eta \nabla C(\theta^0)$

Compute gradient at $\theta^1$

Move to $\theta^2 = \theta^1 - \eta \nabla C(\theta^1)$

...
Formal Derivation of Gradient Descent

• Suppose that θ has two variables \{θ₁, θ₂\}

Given a point, we can easily find the point with the smallest value nearby. How?
Formal Derivation of Gradient Descent

• **Taylor series**: Let \( h(x) \) be infinitely differentiable around \( x = x_0 \).

\[
\begin{align*}
h(x) &= \sum_{k=0}^{\infty} \frac{h^{(k)}(x_0)}{k!} (x - x_0)^k \\
&= h(x_0) + h'(x_0)(x - x_0) + \frac{h''(x_0)}{2!} (x - x_0)^2 + \ldots
\end{align*}
\]

When \( x \) is close to \( x_0 \)

\[
h(x) \approx h(x_0) + h'(x_0)(x - x_0)
\]
E.g. Taylor series for $h(x) = \sin(x)$ around $x_0 = \pi/4$

$$
\sin(x) = \frac{1}{\sqrt{2}} + \frac{x - \frac{\pi}{4}}{\sqrt{2}} - \frac{(x - \frac{\pi}{4})^2}{2 \sqrt{2}} - \frac{(x - \frac{\pi}{4})^3}{6 \sqrt{2}} + \frac{(x - \frac{\pi}{4})^4}{24 \sqrt{2}} + \frac{(x - \frac{\pi}{4})^5}{120 \sqrt{2}} - \frac{(x - \frac{\pi}{4})^6}{720 \sqrt{2}} - \frac{(x - \frac{\pi}{4})^7}{5040 \sqrt{2}} + \frac{(x - \frac{\pi}{4})^8}{40320 \sqrt{2}} + \frac{(x - \frac{\pi}{4})^9}{362880 \sqrt{2}} - \frac{(x - \frac{\pi}{4})^{10}}{3628800 \sqrt{2}} + \ldots
$$

The approximation is good around $\pi/4$. 
Multivariable Taylor series

\[ h(x, y) = h(x_0, y_0) + \frac{\partial h(x_0, y_0)}{\partial x} (x - x_0) + \frac{\partial h(x_0, y_0)}{\partial y} (y - y_0) \]

\[ \approx h(x_0, y_0) + \frac{\partial h(x_0, y_0)}{\partial x} (x - x_0) + \frac{\partial h(x_0, y_0)}{\partial y} (y - y_0) \]

When \( x \) and \( y \) is close to \( x_0 \) and \( y_0 \)

+ something related to \((x-x_0)^2\) and \((y-y_0)^2\) + ......
Formal Derivation of Gradient Descent

Based on Taylor Series:
If the red circle is \textbf{small enough}, in the red circle

\[
C(\theta) \approx C(a, b) + \frac{\partial C(a, b)}{\partial \theta_1} (\theta_1 - a) + \frac{\partial C(a, b)}{\partial \theta_2} (\theta_2 - b)
\]

\[s = C(a, b)\]
\[u = \frac{\partial C(a, b)}{\partial \theta_1}, \quad v = \frac{\partial C(a, b)}{\partial \theta_2}\]

\[
C(\theta) \approx s + u(\theta_1 - a) + v(\theta_2 - b)
\]
Formal Derivation of Gradient Descent

\[ u = \frac{\partial C(a, b)}{\partial \theta_1}, \quad v = \frac{\partial C(a, b)}{\partial \theta_2} \]

Based on Taylor Series:

If the red circle is **small enough**, in the red circle

\[ C(\theta) \approx s + u(\theta_1 - a) + v(\theta_2 - b) \]

Find \( \theta_1 \) and \( \theta_2 \) yielding the smallest value of \( C(\theta) \) in the circle

\[
\begin{bmatrix}
\theta_1 \\
\theta_2 \\
\end{bmatrix} = 
\begin{bmatrix}
a \\
b \\
\end{bmatrix} - \eta 
\begin{bmatrix}
u \\
v \\
\end{bmatrix} = 
\begin{bmatrix}
a \\
b \\
\end{bmatrix} - \eta 
\begin{bmatrix}
\frac{\partial C(a, b)}{\partial \theta_1} \\
\frac{\partial C(a, b)}{\partial \theta_2} \\
\end{bmatrix} = 
\begin{bmatrix}
a \\
b \\
\end{bmatrix} - \eta \nabla C(a, b)
\]

Its value depending on the radius of the circle, \( u \) and \( v \).

This is how gradient descent updates parameters.
Gradient Descent for Neural Network

Starting Parameters

\[ \theta^0 \rightarrow \theta^1 \rightarrow \theta^2 \rightarrow \cdots \]

\[ \nabla C(\theta) \]

\[ \theta = \{ W^1, b^1, W^2, b^2, \ldots, W^l, b^l, \ldots, W^L, b^L \} \]

To compute the gradients efficiently, we use **backpropagation**.
Stuck at local minima?

- Who is Afraid of Non-Convex Loss Functions?
- http://videolectures.net/eml07_lecun_wia/
- Deep Learning: Theoretical Motivations
- http://videolectures.net/deeplearning2015_bengio_theoretical_motivations/
3. How to pick the “best” function?

Practical Issues for neural network
Practical Issues for neural network

• Parameter Initialization
• Learning Rate
• Stochastic gradient descent and Mini-batch
• Recipe for Learning
Parameter Initialization

• For gradient Descent, we need to pick an initialization parameter $\theta^0$.
• The initialization parameters have some influence to the training.
  • We will go back to this issue in the future.
• Suggestion today:
  • Do not set all the parameters $\theta^0$ equal
  • Set the parameters in $\theta^0$ randomly
Learning Rate

\[ \theta^i = \theta^{i-1} - \eta \nabla C(\theta^{i-1}) \]

- Set the learning rate \( \eta \) carefully

![Diagram showing the cost vs. number of parameters updates with different learning rates.](image)
Learning Rate

\[ \theta^i = \theta^{i-1} - \eta \nabla C(\theta^{i-1}) \]

- Set the learning rate \( \eta \) carefully

**Toy Example**

\[
\begin{align*}
    x \quad &\rightarrow w \\
    \quad &\rightarrow z \\
    \quad &\rightarrow y \\
\end{align*}
\]

\[ y = z \]

\[ \theta^* = \begin{bmatrix} w = 1 \\ b = 0 \end{bmatrix} \]

Training Data (20 examples)

\[
\begin{align*}
x &= [0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5, 9.0, 9.5] \\
y &= [0.1, 0.4, 0.9, 1.6, 2.2, 2.5, 2.8, 3.5, 3.9, 4.7, 5.1, 5.3, 6.3, 6.5, 6.7, 7.5, 8.1, 8.5, 8.9, 9.5]
\end{align*}
\]
Learning Rate

\[ \theta^i = \theta^{i-1} - \eta \nabla C(\theta^{i-1}) \]

• Toy Example

Error Surface: \( C(w,b) \)
Learning Rate

- **Toy Example**
  Different learning rate $\eta$

- $\eta = 0.1$
  ~ $3k$ updates

- $\eta = 0.001$
  ~ $0.3k$ updates

- $\eta = 0.01$
  ~ $0.3k$ updates
Stochastic Gradient Descent and Mini-batch

Gradient Descent

\[ C(\theta) = \frac{1}{R} \sum_{r} \| f(x^r; \theta) - \hat{y}^r \| \]

\[ = \frac{1}{R} \sum_{r} C^r(\theta) \]

\[ \theta^i = \theta^{i-1} - \eta \nabla C(\theta^{i-1}) \]

\[ \nabla C(\theta^{i-1}) = \frac{1}{R} \sum_{r} \nabla C^r(\theta^{i-1}) \]

Stochastic Gradient Descent

Faster! Better!

Pick an example \( x^r \)

\[ \theta^i = \theta^{i-1} - \eta \nabla C^r(\theta^{i-1}) \]

If all example \( x^r \) have equal probabilities to be picked

\[ E[\nabla C^r(\theta^{i-1})] = \frac{1}{R} \sum_{r} \nabla C^r(\theta^{i-1}) \]
Stochastic Gradient Descent and Mini-batch

What is epoch?

Training Data: \( \{(x^1, \hat{y}^1), (x^2, \hat{y}^2), \ldots (x^r, \hat{y}^r), \ldots (x^R, \hat{y}^R)\} \)

When using stochastic gradient descent

Starting at \( \theta_0 \)

- pick \( x^1 \) \[ \theta^1 = \theta^0 - \eta \nabla C^1(\theta^0) \]
- pick \( x^2 \) \[ \theta^2 = \theta^1 - \eta \nabla C^2(\theta^1) \]
- : \[ \vdots \]
- pick \( x^r \) \[ \theta^r = \theta^{r-1} - \eta \nabla C^r(\theta^{r-1}) \]
- : \[ \vdots \]
- pick \( x^R \) \[ \theta^R = \theta^{R-1} - \eta \nabla C^R(\theta^{R-1}) \]

Seen all the examples once

One epoch

- pick \( x^1 \) \[ \theta^{R+1} = \theta^R - \eta \nabla C^1(\theta^R) \]
Stochastic Gradient Descent and Mini-batch

- **Toy Example**

**Gradient Descent**
Update after seeing all examples

**Stochastic Gradient Descent**
If there are 20 examples, update 20 times in one epoch.
Stochastic Gradient Descent and Mini-batch

- **Gradient Descent**
  \[
  \theta^i = \theta^{i-1} - \eta \nabla C(\theta^{i-1})
  \]
  \[
  \nabla C(\theta^{i-1}) = \frac{1}{R} \sum_r \nabla C^r(\theta^{i-1})
  \]

- **Stochastic Gradient Descent**
  Pick an example \( x_r \)
  \[
  \theta^i = \theta^{i-1} - \eta \nabla C^r(\theta^{i-1})
  \]

- **Mini Batch Gradient Descent**
  Pick \( B \) examples as a batch \( b \)
  \[
  \theta^i = \theta^{i-1} - \eta \frac{1}{B} \sum_{x_r \in b} \nabla C^r(\theta^{i-1})
  \]
  Average the gradient of the examples in the batch \( b \)

Shuffle your data
Stochastic Gradient Descent and Mini-batch

- **Handwriting Digit Classification**

![Graph showing the relationship between batch size and accuracy/training time. The x-axis represents batch size with labels: stochastic, 10, 100, 1000, 10000, full. The y-axis represents accuracy and training time percentages. The graph shows a decrease in accuracy and increase in training time with larger batch sizes.]
Stochastic Gradient Descent and Mini-batch

• Why mini-batch is faster than stochastic gradient descent?

**Stochastic Gradient Descent**

\[
\begin{align*}
\mathbf{z}_1^1 &= \mathbf{W}^1 \mathbf{x} \\
\mathbf{z}_2^1 &= \mathbf{W}^1 \mathbf{x} \\
&\vdots \\
\mathbf{z}_m^1 &= \mathbf{W}^1 \mathbf{x}
\end{align*}
\]

**Mini-batch**

\[
\begin{align*}
\begin{bmatrix} \mathbf{z}_1^1 & \mathbf{z}_2^1 \end{bmatrix} &= \mathbf{W}^1 \\
\mathbf{x} &\quad \mathbf{x}
\end{align*}
\]

Practically, which one is faster?
Recipe for Learning

- Data provided in homework

Training Data

Validation

Real Testing

"Best" Function $f^*$
Recipe for Learning

- Data provided in homework

Training Data

\[ x \] \[ \hat{y} \]

Testing Data

Validation

\[ x \] \[ y \]

Do not know the accuracy until the deadline

(what really count)

Real Testing

\[ x \] \[ y \]

Immediately know the accuracy
Recipe for Learning

- **Do I get good results on training set?**
  - no
  - Modify your training process

- Your code has bug.
- Can not find a good function
  - Stuck at local minima, saddle points ....
  - Change the training strategy
- Bad model
  - There is no good function in the hypothesis function set.
  - Probably you need bigger network
Recipe for Learning

Do I get good results on training set?

Modify your training process

Do I get good results on validation set?

Preventing Overfitting

Your code usually do not have bug at this situation.
Recipe for Learning - Overfitting

• You pick a “best” parameter set $\theta^*$

  Training Data: $\{\ldots(x^r, \hat{y}^r)\ldots\} \rightarrow \forall r: f(x^r; \theta^*) = \hat{y}^r$

However,

  Testing Data: $\{\ldots x^u \ldots\} \rightarrow f(x^u; \theta^*) \neq \hat{y}^u$

Training data and testing data have different distribution.
Recipe for Learning - Overfitting

• Panacea: Have more training data
  • You can do that in real application, but you can’t do that in homework.
• We will go back to this issue in the future.
Concluding Remarks

1. What is the model (function hypothesis set)?
   - Neural Network

2. What is the “best” function?
   - Cost Function

3. How to pick the “best” function?
   - Parameter Initialization
   - Learning Rate
   - Stochastic gradient descent, Mini-batch
   - Recipe for Learning
Acknowledgement

• 感謝 余朗祺 同學於上課時糾正投影片上的拼字錯誤
• 感謝 吳柏瑜 同學糾正投影片上的 notation 錯誤
• 感謝 Yes Huang 糾正投影片上的打字錯誤