Tips for Training Deep Neural Network

Hung-yi Lee
Announcement

• 分組
  • 請確認在 ceiba 上的分組是否正確
• HW1
  • 截止日期: 10/23 2:00 p.m. (下週五上課前)
• HW2
  • 公告日期: 10/23
  • 截止日期: 11/13 2:00 p.m.
    • 比第一堂課公告的提早一週截止
Outline

- Activation Function
- Cost Function
- Data Preprocessing
- Optimization
- Generalization
Outline

1. Activation Function
2. Cost Function
3. Data Preprocessing
4. Optimization
5. Generalization
ReLU

- Rectified Linear Unit (ReLU)

\[ \sigma(z) \]

\[ a = 0 \]

\[ a = z \]

\[ \sigma'(z) \]

\[ 0 \]

\[ 1 \]
ReLU

• Rectified Linear Unit (ReLU)

Reason:
1. Fast to compute
2. Biological reason
3. Infinite sigmoid with different biases
4. Vanishing gradient problem
Review: Backpropagation

Forward Pass

\[
\begin{align*}
    z^1 &= W^1 x + b^1 \\
    a^1 &= \sigma(z^1) \\
    \ldots \ldots \\
    z^{l-1} &= W^{l-1} a^{l-2} + b^{l-1} \\
    a^{l-1} &= \sigma(z^{l-1})
\end{align*}
\]

Backward Pass

\[
\begin{align*}
    \delta^L &= \sigma'(z^L) \cdot \nabla C_x(y) \\
    \delta^{L-1} &= \sigma'(z^{L-1}) \cdot (W^L)^T \delta^L \\
    \ldots \ldots \\
    \delta^l &= \sigma'(z^l) \cdot (W^{l+1})^T \delta^{l+1} \\
    \ldots \ldots 
\end{align*}
\]
Review: Backpropagation

\[
\frac{\partial C_x}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \frac{\partial C_x}{\partial z_i^l}
\]

Error signal

**Backward Pass**

\[
\delta^L = \sigma'(z^L) \cdot \nabla C_x(y)
\]

\[
\delta^{L-1} = \sigma'(z^{L-1}) \cdot (W^L)^T \delta^L
\]

\[
\delta^l = \sigma'(z^l) \cdot (W^{l+1})^T \delta^{l+1}
\]
Problem of Sigmoid

Derivative of Sigmoid Function is always smaller than 1
Vanishing Gradient Problem

Backward Pass:

For sigmoid function, \( \sigma'(z) \) always smaller than 1

Error signal is getting smaller and smaller

Gradient is smaller

\[
\frac{\partial C_x}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \frac{\partial C_x}{\partial z_i^l}
\]
Vanishing Gradient Problem

Learn very slowly
Still random

Learn faster
Already converge

The weights are converged based on random!?
ReLUs

Backward Pass:

\[
\frac{\partial C_x}{\partial y_1} \times \sigma'(z_1^{L-1}) \\
\frac{\partial C_x}{\partial y_2} \times \sigma'(z_2^{L-1}) \\
\vdots \\
\frac{\partial C_x}{\partial y_n} \times \sigma'(z_n^{L-1})
\]

where:
- \( z_i^{L-1} = \sigma(W_i^{L-1}x + \delta_i^{L-1}) \)
- \( \delta_i^{L-1} \)
- \( W_i^L \)
- \( \sigma'(z_i^L) \)
ReLU

Backward Pass:

\[ \begin{align*}
\nabla C_y(x) & \\
\frac{\partial C_x}{\partial y_1} & \\
\frac{\partial C_x}{\partial y_2} & \\
\frac{\partial C_x}{\partial y_n} & \\
\end{align*} \]
ReLU

Backward Pass:

Layer L-1
\( \delta^{L-1} \)

Layer L
\( \delta^L \)

\[ \nabla C_x(y) \]

\[ \frac{\partial C_x}{\partial y_1} \]
\[ \frac{\partial C_x}{\partial y_2} \]

A thinner network without any attenuation

\[ \sigma(z) \]
\[ a = z \]
\[ a = 0 \]

\[ \sigma'(z) \]
ReLU
ReLU

I have good influence to the output.

A Thinner linear network
ReLU

Backward Pass:

All the weights connected to this neuron will not update.

\[ \delta_n^L = \frac{\partial C_x}{\partial z_n^L} = 0 \]

Possible solution:
1. softplus
2. Initialize with large bias
ReLU - variant

Leaky ReLU

\[ a = \begin{cases} 
z & \text{if } z > 0 \\
0.01z & \text{otherwise} 
\end{cases} \]

Parametric ReLU

\[ a = \begin{cases} 
\alpha z & \text{if } z \leq 0 \\
z & \text{otherwise} 
\end{cases} \]

\( \alpha \) also learned by gradient descent
Maxout

- All ReLU variants are just special cases of Maxout

\[ z^1 = W^1 x \]

\[ z^2 = W^2 a^1 \]
Maxout – ReLU is special case

\[ z = wx + b \]

\[ z_1 = wx + b \]

\[ z_2 = 0 \]

\[ \max\{z_1, z_2\} \]
Maxout – ReLU is special case

\[ z = wx + b \]

Learnable Activation Function

\[ z_1 = wx + b \]
\[ z_2 = w'x + b' \]
Maxout - Training

• Given a training data $x$, we know which $z$ would be the max.
Maxout - Training

- Given a training data $x$, we know which $z$ would be the max

- Train this thin and linear network

\[
\begin{align*}
\mathbf{x} & \rightarrow z^1_1, z^1_2, z^1_3, z^1_4 \\
\mathbf{x} & \rightarrow z^2_1, z^2_2, z^2_3, z^2_4 \\
\mathbf{a}^1 & \rightarrow \mathbf{a}^2
\end{align*}
\]
Outline

1. Activation Function
2. Cost Function
3. Data Preprocessing
4. Optimization
5. Generalization
Cost Function

Cost Function

\[ C = \frac{1}{2} \| y - \hat{y} \|^2 = \frac{1}{2} \sum_{n} (y_n - \hat{y}_n)^2 \]
Output Layer

Only one dimension is 1, and others are all 0

\[ \hat{y} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \end{bmatrix} \]

ReLU

\[ y = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1.2 \\ \vdots \end{bmatrix} \]

More similar?

\[ y = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 2 \\ \vdots \end{bmatrix} \]

Larger output means larger confidence

Better?

Classification Task:

It is better to let the output bounded.
Softmax

- Softmax layer as the output layer

**Ordinary Output layer**

\[ y_1 = \sigma(z_1^L) \]

\[ y_2 = \sigma(z_2^L) \]

\[ y_3 = \sigma(z_3^L) \]
Softmax

• Softmax layer as the output layer

**Softmax Layer**

\[
\begin{align*}
\hat{y}_1 &= \frac{e^{z_1}}{\sum_{j=1}^{3} e^{z_j}} \\
\hat{y}_2 &= \frac{e^{z_2}}{\sum_{j=1}^{3} e^{z_j}} \\
\hat{y}_3 &= \frac{e^{z_3}}{\sum_{j=1}^{3} e^{z_j}}
\end{align*}
\]

*Probability:*

- \(1 > y_i > 0\)
- \(\sum_i y_i = 1\)
Softmax

- What kind of cost function should we use for softmax layer output?

\[
\delta_i^L = \frac{\partial C}{\partial z_i^L}
\]

Large Error Signal
Softmax

\[ y_i = \frac{e^{z_i^L}}{\sum_{j} e^{z_j^L}} \]

Define cost:

\[ C = -\log y_r \]

Do we have to consider other dimensions?

Only one dimension is 1, and others are all 0

\[ \hat{y} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \end{bmatrix} \]

Index of the dimension which is 1

Cross Entropy
\[ y_i = \frac{e^{z^L_i}}{\sum_j e^{z^L_j}} \]

\[ C = -\log y_r \]

\[ \delta^L_r = \frac{\partial C}{\partial z_r^L} = -\frac{1}{y_r} \frac{\partial y_r}{\partial z_r^L} = -\frac{1}{y_r} \left( y_r - y_r^2 \right) = y_r - 1 \]

\[ y_r = \frac{e^{z_r^L}}{\sum_j e^{z_j^L}} \]

\[ z_r^L \text{ appears in both numerator and denominator} \]

The absolute value of \( \delta^L_r \) is larger when \( y_r \) is far from 1.
\[ y_i = \frac{e^{z_i^L}}{\sum_j e^{z_j^L}} \]
\[ C = -\log y_r \]

\[ i \neq r \]

\[ \delta_i^L = \frac{\partial C}{\partial z_i^L} = -\frac{1}{y_r} \frac{\partial y_r}{\partial z_i^L} = -\frac{1}{y_r} \left( -y_r y_i \right) = y_i \]

\[ y_r = \frac{e^{z_r^L}}{\sum_j e^{z_j^L}} \]

\[ z_i^L \text{ appears only in denominator} \]

The absolute value of \( \delta_i^L \) is larger when \( y_i \) is larger.
Outline

1. Activation Function
2. Cost Function
3. Data Preprocessing
4. Optimization
5. Generalization
Normalizing Input

For each dimension $i$:
- mean: $m_i$
- standard deviation: $\sigma_i$

The means of all dimensions are 0, and the variances are all 1

$$x_i^r \leftarrow \frac{x_i^r - m_i}{\sigma_i}$$
Normalizing Input


Normalizing your training and testing data in the same way.
Vanilla 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 $\theta^2 = \theta^1 - \eta \nabla C(\theta^1)$

... 

- Stop until $\nabla C(\theta^t) \approx 0$

1. How to determine the learning rates
2. Stuck at local minima or saddle points
Outline

- Activation Function
- Cost Function
- Data Preprocessing
- Optimization
  - Learning Rate
  - Momentum
- Generalization
Learning Rates

Source:
Learning Rates

• Popular & Simple Idea: Reduce the learning rate by some factor every few epochs.
  • At the beginning, we are far from the destination, so we use larger learning rate
  • After several epochs, we are close to the destination, so we reduce the learning rate
  • E.g. 1/t decay: $\eta^t = \eta / \sqrt{t} + 1$

• Learning rate cannot be one-size-fits-all
  • Give different parameters different learning rates
Adagrad

\[ g^t = \frac{\partial C(\theta^t)}{\partial w} \quad \eta^t = \frac{\eta}{\sqrt{t + 1}} \]

- Divide the learning rate of each parameter by the root mean square of its previous derivatives

**Vanilla Gradient descent**

\[ w^{t+1} \leftarrow w^t - \eta^t g^t \]

**Adagrad**

\[ w^{t+1} \leftarrow w^t - \frac{\eta^t}{\sigma^t} g^t \]

\( \sigma^t \): root mean square of the previous derivatives of parameter w

Parameter dependent
Adagrad

\[ w^1 \leftarrow w^0 - \frac{\eta^0}{\sigma^0} g^0 \]
\[ w^2 \leftarrow w^1 - \frac{\eta^1}{\sigma^1} g^1 \]
\[ w^3 \leftarrow w^2 - \frac{\eta^2}{\sigma^2} g^2 \]
\[ \vdots \]
\[ w^{t+1} \leftarrow w^t - \frac{\eta^t}{\sigma^t} g^t \]

\[ \sigma^0 = g^0 \]
\[ \sigma^1 = \sqrt{\frac{1}{2} [(g^0)^2 + (g^1)^2]} \]
\[ \sigma^2 = \sqrt{\frac{1}{3} [(g^0)^2 + (g^1)^2 + (g^2)^2]} \]
\[ \sigma^t = \sqrt{\frac{1}{t+1} \sum_{i=0}^{t} (g^i)^2} \]

\( \sigma^t \): Root mean square of the previous derivatives of parameter \( w \).
Adagrad

• Divide the learning rate of each parameter by the root mean square of its previous derivatives

\[ \eta^t = \frac{\eta}{\sqrt{t + 1}} \quad 1/t \text{ decay} \]

\[ \sigma^t = \sqrt{\frac{1}{t + 1} \sum_{i=0}^{t} (g^i)^2} \]

\[ w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\sum_{i=0}^{t} (g^i)^2}} g^t \]

\[ w^{t+1} \leftarrow w^t - \eta^t g^t \]
Contradiction? 

\[ g^t = \frac{\partial C(\theta^t)}{\partial w} \quad \eta^t = \frac{\eta}{\sqrt{t + 1}} \]

**Vanilla Gradient descent**

\[ w^{t+1} \leftarrow w^t - \eta^t g^t \]

Larger gradient, larger step

**Adagrad**

\[ w^{t+1} \leftarrow w^t - \frac{\eta g^t}{\sqrt{\sum_{i=0}^{t} (g^i)^2}} \]

Larger gradient, smaller step
Intuitive Reason

- 反差

<table>
<thead>
<tr>
<th>$g^0$</th>
<th>$g^1$</th>
<th>$g^2$</th>
<th>$g^3$</th>
<th>$g^4$</th>
<th>......</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.001</td>
<td>0.001</td>
<td>0.003</td>
<td>0.002</td>
<td>0.1</td>
<td>......</td>
</tr>
<tr>
<td>$g^0$</td>
<td>$g^1$</td>
<td>$g^2$</td>
<td>$g^3$</td>
<td>$g^4$</td>
<td>......</td>
</tr>
<tr>
<td>10.8</td>
<td>20.9</td>
<td>31.7</td>
<td>12.1</td>
<td>0.1</td>
<td>......</td>
</tr>
</tbody>
</table>

\[ w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\sum_{i=0}^{t}(g^i)^2}} g^t \]

造成反差的效果
Larger gradient, larger steps?

\[ y = ax^2 + bx + c \]

\[ \frac{\partial y}{\partial x} = |2ax + b| \]

\[ \frac{\partial y}{\partial x} = \left| \frac{b}{2a} \right| \]

Best step:

\[ \left| x_0 + \frac{b}{2a} \right| \]

\[ \frac{|2ax_0 + b|}{2a} \]
Second Derivative

\[ y = ax^2 + bx + c \]

\[ \frac{\partial y}{\partial x} = |2ax + b| \]

\[ \frac{\partial^2 y}{\partial x^2} = 2a \]

The best step is

**Second derivative**

\[ |2ax_0 + b| \]
More than one parameters

The best step is

<table>
<thead>
<tr>
<th>First derivative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Second derivative</td>
</tr>
</tbody>
</table>

Smaller Learning Rate

Larger Learning Rate

Learning Rate

Learning Rate

Larger second derivative

Smaller second derivative
What to do with Adagrad?

The best step is

<table>
<thead>
<tr>
<th>First derivative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Second derivative</td>
</tr>
</tbody>
</table>

Use *first derivative* to estimate *second derivative*

\[ \sqrt{\text{(first derivative)}^2} \]

**Diagram:**
- For \( w_1 \), the function is a decreasing curve, indicating a smaller second derivative.
- For \( w_2 \), the function is an increasing curve, indicating a larger second derivative.
Easy to stuck

Very slow at the plateau

Stuck at local minima

Stuck at saddle point

\[ \nabla C(\theta^t) \approx 0 \]

\[ \nabla C(\theta^t) = 0 \]

cost

parameter space
In physical world ......

• Momentum

How about put this phenomenon in gradient descent?
Momentum

Movement: movement of last step minus gradient at present

\[ \nabla C(\theta^0) \]
\[ \nabla C(\theta^1) \]
\[ \nabla C(\theta^2) \]
\[ \nabla C(\theta^3) \]

Start at point \( \theta^0 \)
Movement \( v^0 = 0 \)
Compute gradient at \( \theta^0 \)
Movement \( v^1 = \lambda v^0 - \eta \nabla C(\theta^0) \)
Move to \( \theta^1 = \theta^0 + v^1 \)
Compute gradient at \( \theta^1 \)
Movement \( v^2 = \lambda v^1 - \eta \nabla C(\theta^1) \)
Move to \( \theta^2 = \theta^1 + v^2 \)

Movement not just based on gradient, but previous movement.
Momentum

Movement: movement of last step minus gradient at present

\( v^i \) is actually the weighted sum of all the previous gradient:
\[ \nabla C(\theta^0), \nabla C(\theta^1), \ldots \nabla C(\theta^{i-1}) \]

\( v^0 = 0 \)

\( v^1 = -\eta \nabla C(\theta^0) \)

\( v^2 = -\lambda \eta \nabla C(\theta^0) - \eta \nabla C(\theta^1) \)

\ldots

Start at point \( \theta^0 \)
Movement \( v^0=0 \)
Compute gradient at \( \theta^0 \)
Movement \( v^1 = \lambda v^0 - \eta \nabla C(\theta^0) \)
Move to \( \theta^1 = \theta^0 + v^1 \)
Compute gradient at \( \theta^1 \)
Movement \( v^2 = \lambda v^1 - \eta \nabla C(\theta^1) \)
Move to \( \theta^2 = \theta^1 + v^2 \)

Movement not just based on gradient, but previous movement
Momentum

- **cost**
- **parameter space**

**Movement**
- **Negative of Gradient**
- **Movement of last step**

**Gradient = 0**
http://www.reddit.com/r/MachineLearning/comments/2gopfa/visualizing_gradient_optimization_techniques/cklhott (By Alec Radford)
Outline

- Activation Function
- Cost Function
- Data Preprocessing
- Optimization
- Generalization
Panacea

- Have more training data
- *Create* more training data (?)

Handwriting recognition:

Original Training Data: 📒

Created Training Data: 📒

Shift 15°
Outline

 Activation Function

 Cost Function

 Data Preprocessing

 Optimization

 Generalization

 Early Stopping  Weight Decay  Dropout
Early Stopping

How many parameter updates do we need?

Value of Cost Function

Number of parameter updates
Outline

Activation Function

Cost Function

Data Preprocessing

Optimization

Generalization

Early Stopping  Weight Decay  Dropout
Weight Decay

• The parameters closer to zero is preferred.

Training data:
\( \{(x, \hat{y}), \ldots\} \)

Testing data:
\( \{(x', \hat{y}), \ldots\} \)

\( x' = x + \varepsilon \)

\( z' = w \cdot (x + \varepsilon) = w \cdot x + w \cdot \varepsilon = z + w \cdot \varepsilon \)

To minimize the effect of noise, we want \( w \) close to zero.
Weight Decay

• New cost function to be minimized
  • Find a set of weight not only minimizing original cost but also close to zero

\[ C'(\theta) = C(\theta) + \lambda \frac{1}{2} \| \theta \|^2 \]

Original cost
(e.g. minimize square error, cross entropy ...)

Regularization term:
\[ \theta = \{ W^1, W^2, \ldots \} \]
\[ \| \theta \|^2 = (w_{11}^1)^2 + (w_{12}^1)^2 + \ldots \]
\[ + (w_{11}^2)^2 + (w_{12}^2)^2 + \ldots \]
(not consider biases. why?)
Weight Decay

- New cost function to be minimized

\[ C'(\theta) = C(\theta) + \lambda \frac{1}{2} \|\theta\|^2 \]

Gradient:
\[ \frac{\partial C'}{\partial w} = \frac{\partial C}{\partial w} + \lambda w \]

Update:
\[ w^{t+1} \rightarrow w^t - \eta \frac{\partial C'}{\partial w} = w^t - \eta \left( \frac{\partial C}{\partial w} + \lambda w^t \right) \]
\[ = (1 - \eta \lambda)w^t - \eta \frac{\partial C}{\partial w} \]

Smaller and smaller
Weight Decay

• Our Brain
Dropout

Training:

\[ \theta^t \leftarrow \theta^{t-1} - \eta \nabla C_x(\theta^{t-1}) \]

- Each neuron has p% to dropout

In each *iteration*
**Dropout**

**Training:**

- \( x \) (stochastic gradient descent)

- For each iteration, we resample the dropout neurons

\[
\theta^t \leftarrow \theta^{t-1} - \eta \nabla C_x(\theta^{t-1})
\]

- In each **iteration**
  - Each neuron has p% to dropout
  - The structured of the network is changed.
  - Using the new network for training

**For each iteration, we resample the dropout neurons**
Dropout

No dropout

- If the dropout rate at training is p%, all the weights times (1-p)%
- Assume that the dropout rate is 50%. If $w_{ij}^l = 1$ from training, set $w_{ij}^l = 0.5$ for testing.
Dropout - Intuitive Reason

**Training**
Dropout (腳上綁重物)

**Testing**
No dropout
(拿下重物後就變很強)
Dropout - Intuitive Reason

- When teams up, if everyone expect the partner will do the work, nothing will be done finally.
- However, if you know your partner will dropout, you will do better.
- When testing, no one dropout actually, so obtaining good results eventually.
Dropout
- Intuitive Reason

• Why the weights should multiply (1-p)% (dropout rate) when testing?

Training of Dropout
Assume dropout rate is 50%

\[
\begin{align*}
  w_1 & \rightarrow z' \approx 2z \\
  w_2 & \rightarrow \\
  w_3 & \rightarrow \\
  w_4 & \rightarrow
\end{align*}
\]

Testing of Dropout
No dropout

\[
\begin{align*}
  0.5 \times w_1 & \rightarrow z' \approx z \\
  0.5 \times w_2 & \rightarrow \\
  0.5 \times w_3 & \rightarrow \\
  0.5 \times w_4 & \rightarrow
\end{align*}
\]
Train a bunch of networks with different structures
Dropout
- Ensemble

\textit{Ensemble}

Testing data $x$

\begin{align*}
\text{Network } 1 & \rightarrow y_1 \\
\text{Network } 2 & \rightarrow y_2 \\
\text{Network } 3 & \rightarrow y_3 \\
\text{Network } 4 & \rightarrow y_4
\end{align*}

average
Dropout - Ensemble

Training of Dropout

- Using one data to train one network
- Some parameters in the network are shared

- Dropout ≈ Ensemble.

2^M possible networks
Dropout - Ensemble

Testing of Dropout

\[ \text{testing data } x \]

\[ y_1 \quad y_2 \quad y_3 \]

average

\[ ? \approx y \]

\[ \text{All the weights multiply } (1-p)\% \]

\[ \text{Dropout } \approx \text{Ensemble.} \]
Dropout - Ensemble

- Experiments on handwriting digital classification

Practical Suggestion for Dropout

• Larger network
  • If you know your task need \( n \) neurons, for dropout rate \( p \), your network need \( \frac{n}{1-p} \) neurons.

• Longer training time
• Higher learning rate
• Larger momentum
Concluding Remarks

**Activation Function**
- ReLU, Maxout

**Cost Function**
- Softmax + Cross Entropy
- Mean=0, Variance=1

**Data Preprocessing**

**Optimization**
- Adagrad, Momentum

**Generalization**
- Early Stopping, Weight decay, Dropout

Not covered today:
- Parameters Initialization
  http://neuralnetworksanddeeplearning.com/chap3.html#weight_initialization
Acknowledgement

• 感謝 李朋軒 同學糾正投影片上的錯誤
  • 很多地方 p 應該改為 1-p
• 感謝 Ryan Sun 來信指出投影片上的錯誤