Outline

Part I: Introduction of Deep Learning

Part II: Why Deep?

Part III: Tips for Training Deep Neural Network

Part IV: Neural Network with Memory
Part I: Introduction of Deep Learning

What people already knew in 1980s
Neural Network

Deep means many hidden layers
Single Neuron

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

\[ z = a_1 w_1 + a_2 w_2 + \cdots + a_K w_K + b \]
Example of Neural Network

Sigmoid Function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$
Example of Neural Network

\[ f : \mathbb{R}^2 \rightarrow \mathbb{R}^2 \]

\[ f \left( \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right) = \begin{bmatrix} 0.62 \\ 0.83 \end{bmatrix} \]
Example of Neural Network

$$f : \mathbb{R}^2 \to \mathbb{R}^2$$

$$f \left( \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right) = \begin{bmatrix} 0.62 \\ 0.83 \end{bmatrix} \quad f \left( \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 0.51 \\ 0.85 \end{bmatrix}$$

Different parameters define different function $$f$$
Matrix Operation

\[
\begin{bmatrix}
0.98 \\
0.12
\end{bmatrix} = \sigma( \begin{bmatrix}
1 & -2 \\
-1 & 1
\end{bmatrix} \begin{bmatrix}
1 \\
-1
\end{bmatrix} + \begin{bmatrix}
1 \\
0
\end{bmatrix})
\]

Using parallel computing techniques to speed up matrix operation
Fully Connected Feedforward Neural Network

\[ \sigma(W^1 x + b^1) = a^1 \]
\[ \sigma(W^2 a^1 + b^2) = a^2 \]
\[ \sigma(W^L a^{L-1} + b^L) = a^L = y \]
Handwriting Digit Recognition

"Hello World" for Deep Learning
Handwriting Digit Recognition

Input object should always be represented as a fixed-length vector.

Ink → 1
No ink → 1

Each output represents the confidence of a digit.
Handwriting Digit Recognition

16 x 16 = 256

Ink → 1
No ink → 1

Hopefully, the function network can ......

Input: 1 → y_1 has the maximum value
Input: 2 → y_2 has the maximum value
Handwriting Digit Recognition

In general, the output of network does not sum to one
May not be easy to interpret

Input is 1
Input is 2
Input is 0

Ink → 1
No ink → 1

16 x 16 = 256

$16 \times 16 = 256$

$\text{Ink} \rightarrow 1$
$\text{No ink} \rightarrow 1$

In general, the output of network does not sum to one
May not be easy to interpret
Softmax

• Softmax layer as the output layer

Ordinary Output layer

\[ y_1 = \sigma(z_1) \]

\[ y_2 = \sigma(z_2) \]

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

- Softmax layer as the output layer

**Softmax Layer**

\[
\begin{align*}
\sum_{j=1}^{3} e^{z_j} & \approx 0.88 \\
\sum_{j=1}^{3} e^{z_j} & = y_1 = e^{z_1} \\
\sum_{j=1}^{3} e^{z_j} & = y_2 = e^{z_2} \\
\sum_{j=1}^{3} e^{z_j} & = y_3 = e^{z_3}
\end{align*}
\]

*Probability:*
- \(1 > y_i > 0\)
- \(\sum_i y_i = 1\)
Handwriting Digit Recognition

16 x 16 = 256

Ink → 1
No ink → 1

1
x
2
x
256

……
……
……
……

…

0.1
0.7
0.2

Input is 1
Input is 2
Input is 0

Hopefuly, the function network can ......

How to let the neural network achieve this

Input: y_2 has the maximum value
Input: y_1 has the maximum value
Training Data

- Preparing training data: images and their labels

Find the network parameters from the training data
Cost

C: the difference between the output of network and its target

C can be Euclidean distance, cross entropy, etc.
Cost

For all training data ...

\[
\sum \quad x^1 \rightarrow \text{NN} \rightarrow y^1 \Leftarrow \hat{y}^1 \\
\quad x^2 \rightarrow \text{NN} \rightarrow y^2 \Leftarrow \hat{y}^2 \\
\quad x^3 \rightarrow \text{NN} \rightarrow y^3 \Leftarrow \hat{y}^3 \\
\quad \vdots \\
\quad x^R \rightarrow \text{NN} \rightarrow y^R \Leftarrow \hat{y}^R
\]

Total Cost:

\[
C = \sum_{r=1}^{R} C^r
\]

Find the network parameters that can minimize this value
Gradient Descent

Assume there are only two parameters \( w_1 \) and \( w_2 \) in a network.

- Randomly pick a starting point \( \theta^0 \)
- Compute the negative gradient at \( \theta^0 \)
- Times the learning rate

\[ -\nabla C(\theta^0) \]

\[ -\eta \nabla C(\theta^0) \]
Gradient Descent

Eventually, we would reach a minima ..... 

Randomly pick a starting point $\theta^0$

Compute the negative gradient at $\theta^0$

$-\nabla C(\theta^0)$

Times the learning rate $\eta$

$-\eta \nabla C(\theta^0)$
Local Minima

- Gradient descent never guarantee global minima

Different initial point $\theta^0$

Reach different minima, so different results
Besides local minima ......
In physical world ......

- Momentum

How about put this phenomenon in gradient descent?
Momentum

Still not guarantee reaching global minima, but give some hope ……

Real Movement = Negative of Gradient + Momentum

Negative of Gradient

Momentum

Real Movement

Gradient = 0

cost
Mini-batch

Randomly initialize $\theta^0$

Pick the 1$^{st}$ batch $C = C^1 + C^{31} + \cdots$ $\theta^1 \leftarrow \theta^0 - \eta \nabla C(\theta^0)$

Pick the 2$^{nd}$ batch $C = C^2 + C^{16} + \cdots$ $\theta^2 \leftarrow \theta^1 - \eta \nabla C(\theta^1)$

Until all mini-batches have been picked

Repeat above process

Faster
Better!
Backpropagation

• A network can have millions of parameters.
  • Backpropagation is the way to compute the gradients efficiently (not today)

• Many toolkits can compute the gradients automatically
  • Just a function
Part II: Why Deep?
Universality Theorem

Any continuous function \( f \)

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

Can be realized by a network with one hidden layer

(given enough hidden neurons)

Why “Deep” neural network not “Fat” neural network?

Reference for the reason:
Deeper is Better?

- Word error rate (WER)

<table>
<thead>
<tr>
<th>LxN</th>
<th>DBN-PT (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1×2k</td>
<td>24.2</td>
</tr>
<tr>
<td>2×2k</td>
<td>20.4</td>
</tr>
<tr>
<td>3×2k</td>
<td>18.4</td>
</tr>
<tr>
<td>4×2k</td>
<td>17.8</td>
</tr>
<tr>
<td>5×2k</td>
<td>17.2</td>
</tr>
<tr>
<td>7×2k</td>
<td>17.1</td>
</tr>
</tbody>
</table>

Multiple layers

Not surprised, more parameters, better performance

Fat + Short v.s. Thin + Tall

The same number of parameters

Which one is better?

Shallow

Deep

\[ x_1, x_2, \ldots, x_N \]
Fat + Short v.s. Thin + Tall

- Word error rate (WER)

<table>
<thead>
<tr>
<th>LxN</th>
<th>DBN-PT (%)</th>
<th>LxN</th>
<th>DBN-PT (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1×2k</td>
<td>24.2</td>
<td>1×3,772</td>
<td>22.5</td>
</tr>
<tr>
<td>2×2k</td>
<td>20.4</td>
<td>1×4,634</td>
<td>22.6</td>
</tr>
<tr>
<td>3×2k</td>
<td>18.4</td>
<td>1×16K</td>
<td>22.1</td>
</tr>
<tr>
<td>4×2k</td>
<td>17.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5×2k</td>
<td>17.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7×2k</td>
<td>17.1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Why Deep?

• Consider Logic Circuits
  • A two levels of basic logic gates can represent any Boolean function.
  • However, no one uses two levels of logic gates to build computers
  • Using multiple layers of logic gates to build some functions are much simpler (less gates needed).
Why Deep?

- Modulation

Image ➔ Classifier 1 ➔ Girls with glasses

Image ➔ Classifier 2 ➔ Boys with glasses

Image ➔ Classifier 3 ➔ Boys wearing blue shirt
Why Deep?

- Modulation

![Diagram showing the process of classification based on different characteristics of images.](image)

**Image** → **Boy or Girl?** → **With glasses?** → **wearing blue shirt?** → **Classifier 1** → **Girls with glasses**

**Classifier 2** → **Boys with glasses**

**Classifier 3** → **Boys wearing blue shirt**

*Little Data*"
Hand-crafted kernel function

SVM

Apply simple classifier

Input Space

Feature Space

Deep Learning

Learnable kernel

\[ \phi(x) \]

simple classifier

**Boosting**

Input $\mathbf{x}$

Weak classifier

Weak classifier

Weak classifier

Combine

**Deep Learning**

Weak classifier

Boosted weak classifier

Boosted weak classifier

$\mathbf{x}_1$

$\mathbf{x}_2$

$\vdots$

$\mathbf{x}_N$
Hard to get the power of Deep ...

Before 2006, deeper usually does not imply better.
Part III: Tips for Training DNN
Recipe for Learning

Recipe for Learning

Modify the Network
- New activation function, for example, ReLU or Maxout

Better optimization Strategy
- Adaptive learning rate

Prevent Overfitting
- Dropout

Only use this approach when you already obtained good results on the training data.
Part III:
Tips for Training DNN
New Activation Function
ReLU

- Rectified Linear Unit (ReLU)

**Reason:**

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

\[ a = z \]

\[ a = 0 \]

\[ \sigma(z) \]

[Xavier Glorot, AISTATS’11]
Vanishing Gradient Problem

- Smaller gradients: Learn very slow
- Larger gradients: Learn very fast
- Almost random
- Already converge

Based on random!?
ReLU

\[ y_1, y_2 \]

\[ x_1, x_2 \]
ReLU

A Thinner linear network

Do not have smaller gradients
ReLU - variant

**Leaky ReLU**
[Andrew L. Maas, ICML’13]

\[ a = 0.01z \]

**Parametric ReLU**
[Kaiming He, arXiv’15]

\[ a = \alpha z \]

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

• Learnable activation function [Ian J. Goodfellow, ICML’13]

All ReLU variants are just special cases of Maxout

You can have more than 2 elements in a group.
Maxout – ReLU is special case

\[ a = \max\{z_1, z_2\} \]

\[ z = wx + b \]

\[ z_1 = wx + b \]

\[ z_2 = 0 \]
Maxout – ReLU is special case

Input $x \quad 1$

ReLU $z = wx + b$

Learnable Activation Function

Input $x \quad 1$

Max $\max\{z_1, z_2\}$

$z_1 = wx + b$

$z_2 = w'x + b'$
Part III:
Tips for Training DNN
Adaptive Learning Rate
Learning Rate

If learning rate is too large, cost may not decrease after each update. Set the learning rate \( \eta \) carefully.
Learning Rate

If learning rate is too large
Cost may not decrease after each update
If learning rate is too small
Training would be too slow

Can we give different parameters different learning rate?
Adagrad

- Divide the learning rate of each parameter $w$ by a parameter dependent value $\sigma_w$

**Original Gradient descent**

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

$$g^t = \frac{\partial C(\theta^t)}{\partial w}$$

**Adagrad**

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sigma_w} g^t$$

$\sigma^t$ is parameter dependent

$$\sigma^t = \sqrt{\sum_{i=0}^{t} (g^i)^2}$$

Parameter dependent learning rate
Adagrad

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

<table>
<thead>
<tr>
<th>( g^0 )</th>
<th>( g^1 )</th>
<th>……</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.2</td>
<td>……</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( g^0 )</th>
<th>( g^1 )</th>
<th>……</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.0</td>
<td>10.0</td>
<td>……</td>
</tr>
</tbody>
</table>

\[
\frac{\eta}{\sqrt{0.1^2}} = \frac{\eta}{0.1}
\]

\[
\frac{\eta}{\sqrt{0.1^2 + 0.2^2}} = \frac{\eta}{0.22}
\]

\[
\frac{\eta}{\sqrt{20^2}} = \frac{\eta}{20}
\]

\[
\frac{\eta}{\sqrt{20^2 + 10^2}} = \frac{\eta}{22}
\]

**Observation:**

1. Learning rate is smaller and smaller for all parameters
2. Smaller gradient, larger learning rate, and vice versa
Reason of Adagrad

- Smaller gradient, larger learning rate, and vice versa
Not the whole story ......

- Adagrad
- RMSprop
- Adadelta
- Adam
- AdaSecant
- “No more pesky learning rates”
Part III: Tips for Training DNN

Dropout
Dropout

Training:

- Each time before computing the gradients
  - Each neuron has p% to dropout

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

Training:

Each time before computing the gradients

- Each neuron has p% to dropout

  The structure of the network is changed.

- Using the new network for training

For each mini-batch, we resample the dropout neurons
Dropout

Testing:

- No dropout

  - If the dropout rate at training is p%, all the weights times (1-p)%
  
  - Assume that the dropout rate is 50%.
    If a weight $w = 1$ by training, set $w = 0.5$ for testing.
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.

我的 partner 會擺爛，所以 我要好好做
Dropout - Intuitive Reason

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

*Training of Dropout*
Assume dropout rate is 50%

*Testing of Dropout*
No dropout

Weights from training
$z' \approx 2z$

Weights multiply (1-p)%
$z' \approx z$
Dropout is a kind of ensemble.

Train a bunch of networks with different structures.
Dropout is a kind of ensemble.

**Ensemble**

Testing data $x$

- Network 1
  - $Y_1$
- Network 2
  - $Y_2$
- Network 3
  - $Y_3$
- Network 4
  - $Y_4$

average
Dropout is a kind of ensemble.

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

\[ \text{Training of Dropout} \]

M neurons

\[ 2^M \text{ possible networks} \]
Dropout is a kind of ensemble.

**Testing of Dropout**

All the weights multiply \((1-p)\)%

\[
\begin{align*}
    y_1 & \rightarrow y_2 & \rightarrow y_3 \\
    \text{average} & & \approx y
\end{align*}
\]
More about dropout

• Dropout works better with Maxout [Ian J. Goodfellow, ICML’13]
• Dropconnect [Li Wan, ICML’13]
  • Dropout delete neurons
  • Dropconnect deletes the connection between neurons
• Annealed dropout [S.J. Rennie, SLT’14]
  • Dropout rate decreases by epochs
• Standout [J. Ba, NISP’13]
  • Each neural has different dropout rate
Part IV:
Neural Network with Memory
Neural Network needs Memory

• Input and output are sequences

\textit{E.g. Name Entity Recognition (NER)}

Detecting named entities like name of people, locations, organization, etc. in a sentence.
Neural Network needs Memory

• Input and output are sequences

\textit{E.g. Name Entity Recognition (NER)}

DNN needs memory!
Recurrent Neural Network (RNN)

The values are stored by RNN in the previous step.

Memory can be considered as another input.
RNN input: $x^1 \ x^2 \ x^3 \ \ldots \ \ldots \ x^N$

$$y^1 = \text{softmax}(W^o a^1)$$

$$a^1 = \sigma(W^i x^1 + W^h 0)$$
RNN

RNN input: \( x^1, x^2, x^3, \ldots, x^N \)

\[ y^1, y^2 \]

\[ y^2 = \text{softmax}(W^o a^2) \]

\[ a^2 = \sigma(W^i x^2 + W^h a^1) \]
RNN

RNN input: $x^1 \ x^2 \ x^3 \ \ldots \ x^N$

$y^3 = \text{softmax}(W^o a^3)$

$a^3 = \sigma(W^i x^3 + W^h a^2)$
The same network is used again and again.

Output $y^i$ depends on $x^1, x^2, \ldots, x^i$
The same network is used again and again.

Output $y^i$ depends on $x^1$, $x^2$, ..... $x^i$
Training

Backpropagation through time (BPTT)
Of course it can be deep ...
Bidirectional RNN

\[
\begin{align*}
\mathbf{x}_t & \quad \mathbf{y}_t & \quad \mathbf{y}_{t+1} & \quad \mathbf{y}_{t+2} \\
\mathbf{x}_{t+1} & \quad \mathbf{x}_{t+2} \\
\end{align*}
\]
Many to Many (Output is shorter)

• Both input and output are both sequences, but the output is shorter.
  • E.g. Speech Recognition

Problem?

Why can’t it be “好棒棒”

Input: (vector sequence)

Output: “好棒” (character sequence)

Trimming

(vector sequence)
Many to Many (Output is shorter)

• Both input and output are both sequences, **but the output is shorter.**

• Connectionist Temporal Classification (CTC) [Alex Graves, ICML’06][Alex Graves, ICML’14][Haşim Sak, Interspeech’15][Jie Li, Interspeech’15][Andrew Senior, ASRU’15]

“好棒”

Add an extra symbol “φ” representing “null”

“好棒棒”
Many to Many (No Limitation)

• Both input and output are both sequences with **different lengths**. → **Sequence to sequence learning**
  • E.g. **Machine Translation** (machine learning → 機器學習)

![Diagram showing the process of machine learning containing all information about the input sequence.]
Many to Many (No Limitation)

• Both input and output are both sequences with different lengths. → Sequence to sequence learning
  • E.g. Machine Translation (machine learning→機器學習)

Don’t know when to stop
Many to Many (No Limitation)

Ref: http://zh.pttpedia.wikia.com/wiki/%E6%8E%A5%E9%BE%8D%E6%8E%A8%E6%96%87 (鄉民百科)
Many to Many (No Limitation)

• Both input and output are both sequences with different lengths. → Sequence to sequence learning
  • E.g. Machine Translation (machine learning → 機器學習)

Add a symbol “===“ (斷)

[Ilya Sutskever, NIPS’14][Dzmitry Bahdanau, arXiv’15]
The error surface is rough.

The error surface is either very flat or very steep.

[Clipping] [Razvan Pascanu, ICML’13]
Toy Example

- \( w = 1 \) \( \Rightarrow \ y^{1000} = 1 \)
- \( w = 1.01 \) \( \Rightarrow \ y^{1000} \approx 20000 \)
- \( w = 0.99 \) \( \Rightarrow \ y^{1000} \approx 0 \)
- \( w = 0.01 \) \( \Rightarrow \ y^{1000} \approx 0 \)

Large gradient \( \Rightarrow \) Small Learning rate?

Small gradient \( \Rightarrow \) Large Learning rate?
Helpful Techniques

• NAG:
  • Advance momentum method

• RMS Prop
  • Advanced Adagrad
  • Second derivative change

• Long Short-term Memory (LSTM)
  • Can deal with the gradient vanishing problem
Long Short-term Memory (LSTM)

Signal control the output gate
(Other part of the network)

Signal control the input gate
(Other part of the network)

Special Neuron:
4 inputs, 1 output

Other part of the network

LSTM

Other part of the network
Activation function $f$ is usually a sigmoid function between 0 and 1. Mimic open and close gate.

$$c' = g(z)f(z_i) + cf(z_f)$$
Original Network:

» Simply replace the neurons with LSTM
4 times of parameters

\[ a_1, a_2 \]

Input

\[ x_1, x_2 \]
LSTM

Extension: “peephole”

\[ c^{t-1} \]

\[ c^t \]

\[ c^{t+1} \]

\[ z_f^t \]

\[ z_i^t \]

\[ z^t \]

\[ z_o^t \]

\[ x^t \]

\[ y^t \]

\[ y^{t+1} \]

\[ h^t \]

\[ n^{t+1} \]
Other Simpler Alternatives

Gated Recurrent Unit (GRU)  Structurally Constrained Recurrent Network (SCRN)

Vanilla RNN Initialized with Identity matrix + ReLU activation function [Quoc V. Le, arXiv’15]

➢ Outperform or be comparable with LSTM in four different tasks

[Cho. EMNLP’14]  [Tomas Mikolov, ICLR’15]
What is the next wave?

- Attention-based Model
Concluding Remarks
Concluding Remarks

• Introduction of deep learning
• Discussing some reasons using deep learning
• New techniques for deep learning
  • ReLU, Maxout
  • Giving all the parameters different learning rates
  • Dropout
• Network with memory
  • Recurrent neural network
  • Long short-term memory (LSTM)
Thank you for your attention!