Introduction of Machine / Deep Learning

Hung-yi Lee 李宏毅
Machine Learning
≈ Looking for Function

• Speech Recognition
  
  \[ f(\text{声波图}) = \text{“How are you”} \]

• Image Recognition
  
  \[ f(\text{图片}) = \text{“Cat”} \]

• Playing Go
  
  \[ f(\text{棋盘}) = \text{“5-5” (next move)} \]
Different types of Functions

**Regression:** The function outputs a scalar.

Predict PM2.5

PM2.5 today temperature Concentration of O₃ → $f$ → PM2.5 of tomorrow

**Classification:** Given options (classes), the function outputs the correct one.

Spam filtering

EMAIL → $f$ → Yes/No
Different types of Functions

**Classification**: Given options (classes), the function outputs the correct one.

Each position is a class (19 x 19 classes)

Playing GO

Next move

a position on the board
Structured Learning

*create* something with structure (image, document)
How to find a function?

A Case Study
YouTube Channel

https://www.youtube.com/c/HungyiLeeNTU
The function we want to find ...

\[ y = f(\text{no. of views on 2/26}) \]
1. Function with Unknown Parameters

\[ y = f(x) \]

**Model** \[ y = b + wx_1 \]

Based on domain knowledge

- **Feature:**
  - \( y \): no. of views on 2/26
  - \( x_1 \): no. of views on 2/25

- \( w \) and \( b \) are unknown parameters (learned from data)

**Weight**  **Bias**
2. Define **Loss** from Training Data

- Loss is a function of parameters $L(b, w)$.
- Loss: how good a set of values is.

$L(0.5k, 1)$ $\quad y = b + wx_1 \quad \rightarrow \quad y = 0.5k + 1x_1$ How good it is?

Data from 2017/01/01 – 2020/12/31

- $y = 0.5k + 1x_1$ $\quad e_1 = |y - \hat{y}| = 0.4k$

$\hat{y}$ label
2. Define **Loss** from Training Data

Loss is a function of parameters $L(b, w)$

Loss: how good a set of values is.

$L(0.5k, 1)$ \( y = b + w x_1 \) \( \rightarrow \) \( y = 0.5k + 1x_1 \) How good it is?

Data from 2017/01/01 – 2020/12/31

\[
\begin{array}{ccccccc}
2017/01/01 & 01/02 & 01/03 & \cdots & 2020/12/30 & 12/31 \\
4.8k & 4.9k & 7.5k & & 3.4k & 9.8k \\
\end{array}
\]

\[0.5k + 1x_1 = y\]

5.4k

\[e_2 = |y - \hat{y}| = 2.1k\]

\[\hat{y}\]

\[\hat{y}\]

\[0.5k + 1x_1 = y\]

\[e_N\]

\[9.8k\]
2. Define Loss from Training Data

- Loss is a function of parameters $L(b, w)$
- Loss: how good a set of values is.

$$e = |y - \hat{y}|$$  $L$ is mean absolute error (MAE)

$$e = (y - \hat{y})^2$$  $L$ is mean square error (MSE)

If $y$ and $\hat{y}$ are both probability distributions  ➤ Cross-entropy
2. Define Loss from Training Data

Model $y = b + wx_1$

- Loss is a function of parameters $L(b, w)$
- Loss: how good a set of values is.

![Error Surface]
3. Optimization

\[ w^* = \arg \min_w L \]

**Gradient Descent**

- (Randomly) Pick an initial value \( w^0 \)
- Compute \( \frac{\partial L}{\partial w} \bigg|_{w=w^0} \)

**Negative** → Increase \( w \)
**Positive** → Decrease \( w \)

Source of image: http://chico386.pixnet.net/album/photo/171572850
3. Optimization

Gradient Descent

- (Randomly) Pick an initial value $w^0$
- Compute $\frac{\partial L}{\partial w} |_{w=w^0}$

$$w^1 \leftarrow w^0 - \eta \frac{\partial L}{\partial w} |_{w=w^0}$$

$\eta$: learning rate

Source of image: http://chico386.pixnet.net/album/photo/171572850
3. Optimization

$$w^* = \arg\min_w L$$

**Gradient Descent**

- (Randomly) Pick an initial value $w^0$
- Compute $\frac{\partial L}{\partial w}\big|_{w=w^0}$
- Update $w$ iteratively

Does local minima truly cause the problem?
3. Optimization

\[ w^*, b^* = \arg\min_{w,b} L \]

- (Randomly) Pick initial values \( w^0, b^0 \)
- Compute

\[
\begin{align*}
\frac{\partial L}{\partial w} \bigg|_{w=w^0, b=b^0} & \quad w^1 \leftarrow w^0 - \eta \frac{\partial L}{\partial w} \bigg|_{w=w^0, b=b^0} \\
\frac{\partial L}{\partial b} \bigg|_{w=w^0, b=b^0} & \quad b^1 \leftarrow b^0 - \eta \frac{\partial L}{\partial b} \bigg|_{w=w^0, b=b^0}
\end{align*}
\]

Can be done in one line in most deep learning frameworks
- Update \( w \) and \( b \) interatively
3. Optimization

Model \( y = b + wx_1 \)

\[ w^*, b^* = \arg \min_{w,b} L \]

\[ w^* = 0.97, b^* = 0.1k \]

\[ L(w^*, b^*) = 0.48k \]
Machine Learning is so simple ....

\[ y = b + wx_1 \]

\[ w^* = 0.97, b^* = 0.1k \]

\[ L(w^*, b^*) = 0.48k \]
Machine Learning is so simple ……

\[ y = b + wx_1 \]

Step 1: function with unknown

Step 2: define loss from training data

Step 3: optimization

\[ w^* = 0.97, b^* = 0.1k \]
\[ L(w^*, b^*) = 0.48k \]

\[ y = 0.1k + 0.97x_1 \] achieves the smallest loss \( L = 0.48k \) on data of 2017 – 2020 (training data)

How about data of 2021 (unseen during training)?

\[ L' = 0.58k \]
$y = 0.1k + 0.97x_1$

Red: real no. of views
blue: estimated no. of views
\[ y = b + wx \]

\[ y = b + \sum_{j=1}^{7} w_j x_j \]

\[
\begin{array}{cccccccc}
 b & w_1^* & w_2^* & w_3^* & w_4^* & w_5^* & w_6^* & w_7^* \\
 0.05k & 0.79 & -0.31 & 0.12 & -0.01 & -0.10 & 0.30 & 0.18 \\
\end{array}
\]

\[ y = b + \sum_{j=1}^{28} w_j x_j \]

\[ y = b + \sum_{j=1}^{56} w_j x_j \]

2017 - 2020

2021

\[ L = 0.48k \]

\[ L' = 0.58k \]

\[ L = 0.38k \]

\[ L' = 0.49k \]

\[ L = 0.33k \]

\[ L' = 0.46k \]

\[ L = 0.32k \]

\[ L' = 0.46k \]

**Linear models**
Linear models are too simple ... we need more sophisticated modes.

Different $b$

Different $w$

Linear models have severe limitation.  \textbf{Model Bias}

We need a more flexible model!
red curve = constant + sum of a set of
All Piecewise Linear Curves

= constant + sum of a set of

More pieces require more
Beyond Piecewise Linear?

Approximate continuous curve by a piecewise linear curve.

To have good approximation, we need sufficient pieces.
red curve = constant + sum of a set of

How to represent this function?

**Sigmoid Function**

\[
y = c \frac{1}{1 + e^{-(b+wx_1)}}
\]

= \( c \text{ sigmoid}(b + wx_1) \)
Different $w$

Change slopes

Different $b$

Shift

Different $c$

Change height
red curve = sum of a set of \( \sum_{i} c_i \text{sigmoid}(b_i + w_i x_1) \) + constant

\[
y = b + \sum_{i} c_i \text{sigmoid}(b_i + w_i x_1)
\]
New Model: More Features

\[ y = b + w x_1 \]

\[ y = b + \sum_i c_i \text{sigmoid}(b_i + w_i x_1) \]

\[ y = b + \sum_j w_j x_j \]

\[ y = b + \sum_i c_i \text{sigmoid} \left( b_i + \sum_j w_{ij} x_j \right) \]
\[ y = b + \sum_i c_i \text{sigmoid} \left( b_i + \sum_j w_{ij} x_j \right) \]

\[ r_1 = b_1 + w_{11} x_1 + w_{12} x_2 + w_{13} x_3 \]

\[ w_{ij}: \text{weight for } x_j \text{ for } i\text{-th sigmoid} \]

\[ r_2 = b_2 + w_{21} x_1 + w_{22} x_2 + w_{23} x_3 \]

\[ r_3 = b_3 + w_{31} x_1 + w_{32} x_2 + w_{33} x_3 \]
\[ y = b + \sum_{i} c_i \text{sigmoid} \left( b_i + \sum_{j} w_{ij} x_j \right) \quad i: 1,2,3 \\
\]
\[ r_1 = b_1 + w_{11} x_1 + w_{12} x_2 + w_{13} x_3 \\
r_2 = b_2 + w_{21} x_1 + w_{22} x_2 + w_{23} x_3 \\
r_3 = b_3 + w_{31} x_1 + w_{32} x_2 + w_{33} x_3 \\
\]
\[
\begin{bmatrix}
  r_1 \\
  r_2 \\
  r_3
\end{bmatrix} = \begin{bmatrix}
  b_1 \\
  b_2 \\
  b_3
\end{bmatrix} + \begin{bmatrix}
  w_{11} & w_{12} & w_{13} \\
  w_{21} & w_{22} & w_{23} \\
  w_{31} & w_{32} & w_{33}
\end{bmatrix} \begin{bmatrix}
  x_1 \\
  x_2 \\
  x_3
\end{bmatrix}
\]
\[ r = b + W x \]
\[ y = b + \sum_{i} c_i \text{sigmoid} \left( b_i + \sum_{j} w_{ij} x_j \right) \]

\[ i: 1,2,3 \]
\[ j: 1,2,3 \]

\[ r = b + W x \]
\[ y = b + \sum_{i} c_i \text{sigmoid} \left( b_i + \sum_{j} w_{ij} x_j \right) \]

\[ a_1 = \text{sigmoid}(r_1) = \frac{1}{1 + e^{-r_1}} \]

\[ a = \sigma(r) \]

\[ a_3 = \sigma(r) \]

\[ a_1 \rightarrow r_1 \rightarrow 1 \]

\[ a_2 \rightarrow r_2 \rightarrow 2 \]

\[ a_3 \rightarrow r_3 \rightarrow 3 \]

\[ x_1 \rightarrow w_{11} \rightarrow 1 \]

\[ x_2 \rightarrow w_{12} \rightarrow 2 \]

\[ x_3 \rightarrow w_{13} \rightarrow 3 \]
\[ y = b + \sum_i c_i \text{sigmoid} \left( b_i + \sum_j w_{ij} x_j \right) \]

\[ i: 1,2,3 \]

\[ j: 1,2,3 \]
\[ y = b + c^T a \]

\[ a = \sigma(r) \quad r = b + W x \]
\[ y = b + c^T \sigma (b + Wx) \]
Function with unknown parameters

\[
y = b + c^T \sigma(b + Wx)
\]
Step 1: function with unknown

Step 2: define loss from training data

Step 3: optimization

\[ y = b + c^T \sigma(b + Wx) \]
Loss

- Loss is a function of parameters \( L(\theta) \)
- Loss means how good a set of values is.

Given a set of values

\[
\begin{align*}
    y &= b + c^T \sigma(b + Wx) \\
    \hat{y} &= e
\end{align*}
\]

Loss:

\[
L = \frac{1}{N} \sum_{n} e_n
\]
Back to ML Framework

Step 1: function with unknown

Step 2: define loss from training data

Step 3: optimization

\[ y = b + c^T \sigma(b + Wx) \]
Optimization of New Model

$$\theta^* = \arg \min_{\theta} L$$

- (Randomly) Pick initial values $$\theta^0$$

$$g = \nabla L(\theta^0)$$

$$\theta^1 \leftarrow \theta^0 - \eta g$$
Optimization of New Model

\[ \theta^* = \arg \min_{\theta} L \]

- (Randomly) Pick initial values \( \theta^0 \)
- Compute gradient \( g = \nabla L(\theta^0) \)
  \[ \theta^1 \leftarrow \theta^0 - \eta g \]
- Compute gradient \( g = \nabla L(\theta^1) \)
  \[ \theta^2 \leftarrow \theta^1 - \eta g \]
- Compute gradient \( g = \nabla L(\theta^2) \)
  \[ \theta^3 \leftarrow \theta^2 - \eta g \]
Optimization of New Model

\[ \theta^* = \arg \min_{\theta} L \]

- (Randomly) Pick initial values \( \theta^0 \)
- Compute gradient \( g = \nabla L^1(\theta^0) \)
  \[ \text{update } \theta^1 \leftarrow \theta^0 - \eta g \]
- Compute gradient \( g = \nabla L^2(\theta^1) \)
  \[ \text{update } \theta^2 \leftarrow \theta^1 - \eta g \]
- Compute gradient \( g = \nabla L^3(\theta^2) \)
  \[ \text{update } \theta^3 \leftarrow \theta^2 - \eta g \]

1 epoch = see all the batches once
Optimization of New Model

**Example 1**

- 10,000 examples \((N = 10,000)\)
- Batch size is 10 \((B = 10)\)

How many update in 1 epoch?

1,000 updates

**Example 2**

- 1,000 examples \((N = 1,000)\)
- Batch size is 100 \((B = 100)\)

How many update in 1 epoch?

10 updates
Step 1: function with unknown
Step 2: define loss from training data
Step 3: optimization

\[ y = b + c^T \sigma(b + Wx) \]

More variety of models ...
Sigmoid $\rightarrow$ ReLU

How to represent this function?

Rectified Linear Unit (ReLU)

$c \max(0, b + wx_1)$

$c' \max(0, b' + w'x_1)$
Sigmoid $\rightarrow$ ReLU

$$y = b + \sum_i c_i \text{sigmoid} \left( b_i + \sum_j w_{ij} x_j \right)$$

**Activation function**

$$y = b + \sum_i c_i \text{max} \left( 0, b_i + \sum_j w_{ij} x_j \right)$$

Which one is better?
Experimental Results

\[ y = b + \sum_{2i} c_i \max \left( 0, b_i + \sum_j w_{ij} x_j \right) \]

<table>
<thead>
<tr>
<th></th>
<th>linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017 – 2020</td>
<td>0.32k</td>
</tr>
<tr>
<td>2021</td>
<td>0.46k</td>
</tr>
</tbody>
</table>
Back to ML Framework

Step 1: function with unknown

Step 2: define loss from training data

Step 3: optimization

\[ y = b + c^T \sigma(b + Wx) \]

Even more variety of models ...
\[
\begin{align*}
\mathbf{a}' &= \sigma(\mathbf{b}' + \mathbf{W}' \mathbf{a}) \\
\mathbf{a} &= \sigma(\mathbf{b} + \mathbf{W} \mathbf{x})
\end{align*}
\]
Experimental Results

• Loss for multiple hidden layers
  • 100 ReLU for each layer
  • input features are the no. of views in the past 56 days

<table>
<thead>
<tr>
<th></th>
<th>1 layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017 – 2020</td>
<td>0.28k</td>
</tr>
<tr>
<td>2021</td>
<td>0.43k</td>
</tr>
</tbody>
</table>
3 layers

Red: real no. of views
blue: estimated no. of views

Views (k)

2021/01/01
2021/02/14
Step 1: function with unknown

Step 2: define loss from training data

Step 3: optimization

\[ y = b + c^T \sigma(b + WX) \]

It is not \textit{fancy} enough.

Let’s give it a \textit{fancy} name!
This mimics human brains … (???)

Neural Network

Many layers means Deep → Deep Learning
Deep = Many hidden layers


Deep = Many hidden layers

Why we want “Deep” network, not “Fat” network?

AlexNet (2012) 16.4%
VGG (2014) 7.3%
GoogleNet (2014) 6.7%
Residual Net (2015) 3.57%
Taipei 101
Why don’t we go deeper?

- Loss for multiple hidden layers
  - 100 ReLU for each layer
  - Input features are the no. of views in the past 56 days

<table>
<thead>
<tr>
<th></th>
<th>1 layer</th>
<th>2 layer</th>
<th>3 layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017 – 2020</td>
<td>0.28k</td>
<td>0.18k</td>
<td>0.14k</td>
</tr>
<tr>
<td>2021</td>
<td>0.43k</td>
<td>0.39k</td>
<td>0.38k</td>
</tr>
</tbody>
</table>
Why don’t we go deeper?

- Loss for multiple hidden layers
  - 100 ReLU for each layer
  - Input features are the no. of views in the past 56 days

<table>
<thead>
<tr>
<th></th>
<th>1 layer</th>
<th>2 layer</th>
<th>3 layer</th>
<th>4 layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017 – 2020</td>
<td>0.28k</td>
<td>0.18k</td>
<td>0.14k</td>
<td>0.10k</td>
</tr>
<tr>
<td>2021</td>
<td>0.43k</td>
<td>0.39k</td>
<td>0.38k</td>
<td>0.44k</td>
</tr>
</tbody>
</table>

Better on training data, worse on unseen data

Overfitting
Let’s predict no. of views today!

- If we want to select a model for predicting no. of views today, which one will you use?

<table>
<thead>
<tr>
<th>Year</th>
<th>1 layer</th>
<th>2 layer</th>
<th>3 layer</th>
<th>4 layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017 – 2020</td>
<td>0.28k</td>
<td>0.18k</td>
<td>0.14k</td>
<td>0.10k</td>
</tr>
<tr>
<td>2021</td>
<td>0.43k</td>
<td>0.39k</td>
<td>0.38k</td>
<td>0.44k</td>
</tr>
</tbody>
</table>

We will talk about model selection next time. 😊
To learn more ......

Basic Introduction

https://youtu.be/Dr-WRIEFefw

Backpropagation
Computing gradients in an efficient way

https://youtu.be/ibJpTrp5mcE