



PyTorch Tutorial

07. Multiple Dimension Input

Revision

x (hours)	y (points)
1	2
2	4
3	6
4	?

x (hours)	y (pass/fail)
1	0 (fail)
2	0 (fail)
3	1 (pass)
4	?

Diabetes Dataset

X1	X2	X3	X4	X5	X6	X7	X8	Y	Sample	
-0.29	0.49	0.18	-0.29	0.00	0.00	-0.53	-0.03	0		
-0.88	-0.15	0.08	-0.41	0.00	-0.21	-0.77	-0.67	1		
-0.06	0.84	0.05	0.00	0.00	-0.31	-0.49	-0.63	0		
-0.88	-0.11	0.08	-0.54	-0.78	-0.16	-0.92	0.00	1		
0.00	0.38	-0.34	-0.29	-0.60	0.28	0.89	-0.60	0		
-0.41	0.17	0.21	0.00	0.00	-0.24	-0.89	-0.70	1		
-0.65	-0.22	-0.18	-0.35	-0.79	-0.08	-0.85	-0.83	0		
0.18	0.16	0.00	0.00	0.00	0.05	-0.95	-0.73	1		
-0.76	0.98	0.15	-0.09	0.28	-0.09	-0.93	0.07	0		
-0.06	0.26	0.57	0.00	0.00	0.00	-0.87	0.10	0		

↑
feature

Diabetes Dataset

X1	X2	X3	X4	X5	X6	X7	X8	Y
-0.29	0.49	0.18	-0.29	0.00	0.00	-0.53	-0.03	0
-0.88	-0.15	0.08	-0.41	0.00	-0.21	-0.77	-0.67	1
-0.06	0.84	0.05	0.00	0.00	-0.31	-0.49	-0.63	0
-0.88	-0.11	0.08	-0.54	-0.78	-0.16	-0.92	0.00	1
0.00	0.38	-0.34	-0.29	-0.60	0.28	0.89	-0.60	0
-0.41	0.17	0.21	0.00	0.00	-0.24	-0.89	-0.70	1
-0.65	-0.22	-0.18	-0.35	-0.79	-0.08	-0.85	-0.83	0
0.18	0.16	0.00	0.00	0.00	0.05	-0.95	-0.73	1
-0.76	0.98	0.15	-0.09	0.28	-0.09	-0.93	0.07	0
-0.06	0.26	0.57	0.00	0.00	0.00	-0.87	0.10	0

Feature

Multiple Dimension Logistic Regression Model

Logistic Regression Model

$$\hat{y}^{(i)} = \sigma(x^{(i)} * \omega + b)$$



Logistic Regression Model

$$\hat{y}^{(i)} = \sigma\left(\sum_{n=1}^8 x_n^{(i)} \cdot \omega_n + b\right)$$

Multiple Dimension Logistic Regression Model

Logistic Regression Model

$$\hat{y}^{(i)} = \sigma(x^{(i)} * \omega + b)$$



Logistic Regression Model

$$\hat{y}^{(i)} = \sigma\left(\sum_{n=1}^8 x_n^{(i)} \cdot \omega_n + b\right)$$



$$\sum_{n=1}^8 x_n^{(i)} \cdot \omega_n = \begin{bmatrix} x_1^{(i)} & \dots & x_8^{(i)} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \vdots \\ \omega_8 \end{bmatrix}$$

Multiple Dimension Logistic Regression Model

Logistic Regression Model

$$\hat{y}^{(i)} = \sigma(x^{(i)} * \omega + b)$$



Logistic Regression Model

$$\hat{y}^{(i)} = \sigma\left(\sum_{n=1}^8 x_n^{(i)} \cdot \omega_n + b\right)$$



$$\sum_{n=1}^8 x_n^{(i)} \cdot \omega_n = \begin{bmatrix} x_1^{(i)} & \dots & x_N^{(i)} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \vdots \\ \omega_8 \end{bmatrix}$$



Logistic Regression Model

$$\begin{aligned} \hat{y}^{(i)} &= \sigma\left(\begin{bmatrix} x_1^{(i)} & \dots & x_8^{(i)} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \vdots \\ \omega_8 \end{bmatrix} + b\right) \\ &= \sigma(z^{(i)}) \end{aligned}$$

Mini-Batch (N samples)

$$\begin{bmatrix} \hat{y}^{(1)} \\ \vdots \\ \hat{y}^{(N)} \end{bmatrix} = \begin{bmatrix} \sigma(z^{(1)}) \\ \vdots \\ \sigma(z^{(N)}) \end{bmatrix} = \sigma \left(\begin{bmatrix} z^{(1)} \\ \vdots \\ z^{(N)} \end{bmatrix} \right)$$

Sigmoid function is in an element-wise fashion.

$$\text{torch.exp} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

\Downarrow

$$e^{x_1}$$

$$e^{x_2}$$

$$e^{x_3}$$

Mini-Batch (N samples)

$$\begin{bmatrix} \hat{y}^{(1)} \\ \vdots \\ \hat{y}^{(N)} \end{bmatrix} = \begin{bmatrix} \sigma(z^{(1)}) \\ \vdots \\ \sigma(z^{(N)}) \end{bmatrix} = \sigma \left(\begin{bmatrix} z^{(1)} \\ \vdots \\ z^{(N)} \end{bmatrix} \right)$$

Sigmoid function is in an element-wise fashion.

$$\begin{aligned} z^{(1)} &= \begin{bmatrix} x_1^{(1)} & \dots & x_8^{(1)} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \vdots \\ \omega_8 \end{bmatrix} + b \\ &\vdots \\ z^{(N)} &= \begin{bmatrix} x_1^{(N)} & \dots & x_8^{(N)} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \vdots \\ \omega_8 \end{bmatrix} + b \end{aligned}$$

Mini-Batch (N samples)

$$\begin{bmatrix} \hat{y}^{(1)} \\ \vdots \\ \hat{y}^{(N)} \end{bmatrix} = \begin{bmatrix} \sigma(z^{(1)}) \\ \vdots \\ \sigma(z^{(N)}) \end{bmatrix} = \sigma \left(\begin{bmatrix} z^{(1)} \\ \vdots \\ z^{(N)} \end{bmatrix} \right)$$

Sigmoid function is in an element-wise fashion.

$$\begin{aligned} z^{(1)} &= \begin{bmatrix} x_1^{(1)} & \dots & x_8^{(1)} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \vdots \\ \omega_8 \end{bmatrix} + b \\ &\vdots \\ z^{(N)} &= \begin{bmatrix} x_1^{(N)} & \dots & x_8^{(N)} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \vdots \\ \omega_8 \end{bmatrix} + b \end{aligned}$$



$$\begin{bmatrix} z^{(1)} \\ \vdots \\ z^{(N)} \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & \dots & x_8^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(N)} & \dots & x_8^{(N)} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \vdots \\ \omega_8 \end{bmatrix} + \begin{bmatrix} b \\ \vdots \\ b \end{bmatrix}$$

$N \times 1$ $N \times 8$ 8×1 $N \times 1$

Mini-Batch (N samples)

$$\begin{bmatrix} \hat{y}^{(1)} \\ \vdots \\ \hat{y}^{(N)} \end{bmatrix} = \begin{bmatrix} \sigma(z^{(1)}) \\ \vdots \\ \sigma(z^{(N)}) \end{bmatrix} = \sigma \left(\begin{bmatrix} z^{(1)} \\ \vdots \\ z^{(N)} \end{bmatrix} \right)$$

```
class Model(torch.nn.Module):  
    def __init__(self):  
        super(Model, self).__init__()  
        self.linear = torch.nn.Linear(8, 1)  
        self.sigmoid = torch.nn.Sigmoid()  
  
    def forward(self, x):  
        x = self.sigmoid(self.linear(x))  
        return x  
  
model = Model()
```

$$z^{(1)} = \begin{bmatrix} x_1^{(1)} & \dots & x_8^{(1)} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \vdots \\ \omega_8 \end{bmatrix} + b$$
$$\vdots$$
$$z^{(N)} = \begin{bmatrix} x_1^{(N)} & \dots & x_8^{(N)} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \vdots \\ \omega_8 \end{bmatrix} + b$$



$$\begin{bmatrix} z^{(1)} \\ \vdots \\ z^{(N)} \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & \dots & x_8^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(N)} & \dots & x_8^{(N)} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \vdots \\ \omega_8 \end{bmatrix} + \begin{bmatrix} b \\ \vdots \\ b \end{bmatrix}$$

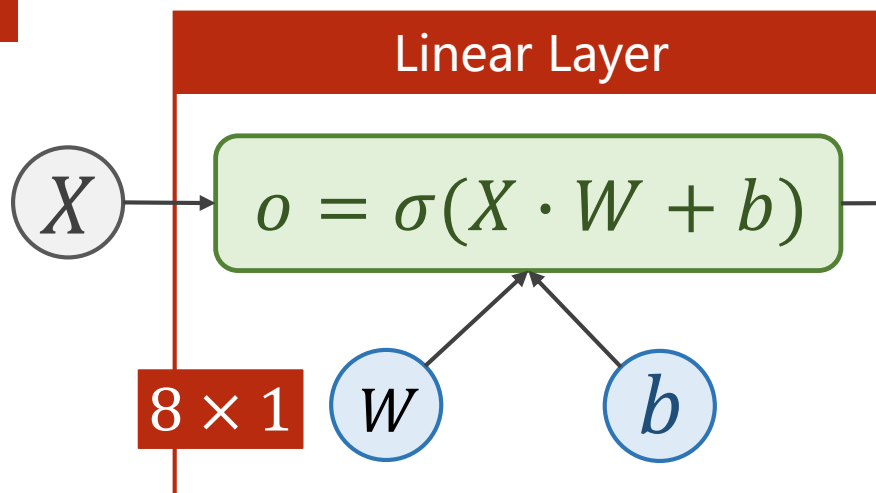
$N \times 1$ — $N \times 8$ — 8×1 — $N \times 1$

Linear Layer

```
self.linear = torch.nn.Linear(8, 1)
```

Size of each input sample

$$X = \begin{bmatrix} x_1^{(1)} & \dots & x_8^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(N)} & \dots & x_8^{(N)} \end{bmatrix}$$



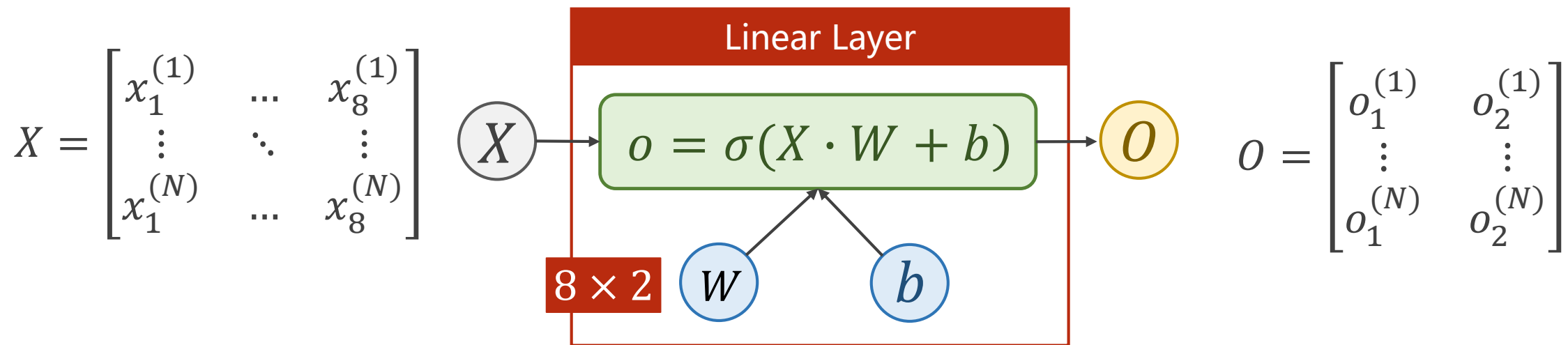
Size of each output sample

$$O = \begin{bmatrix} o^{(1)} \\ \vdots \\ o^{(N)} \end{bmatrix}$$

Linear Layer

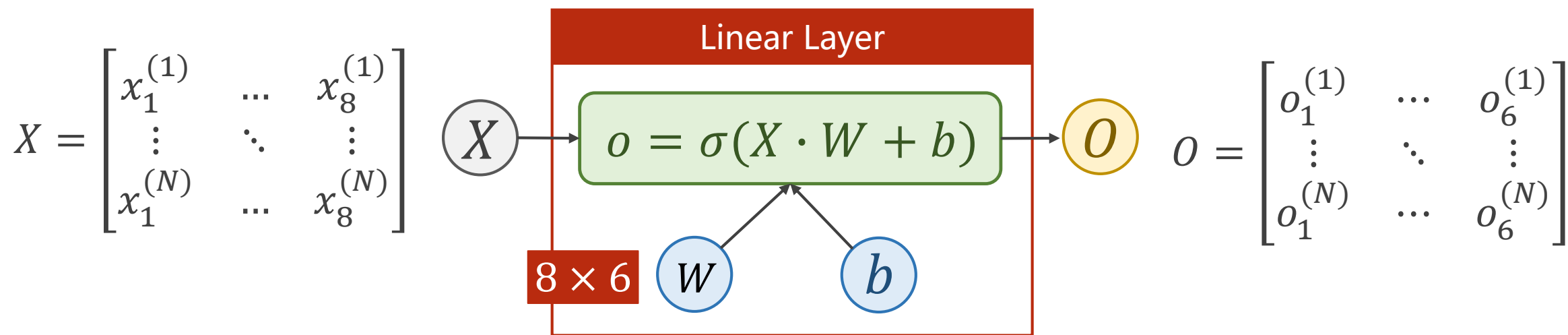
```
self.linear = torch.nn.Linear(8, 2)
```

8-D \Rightarrow 2-D

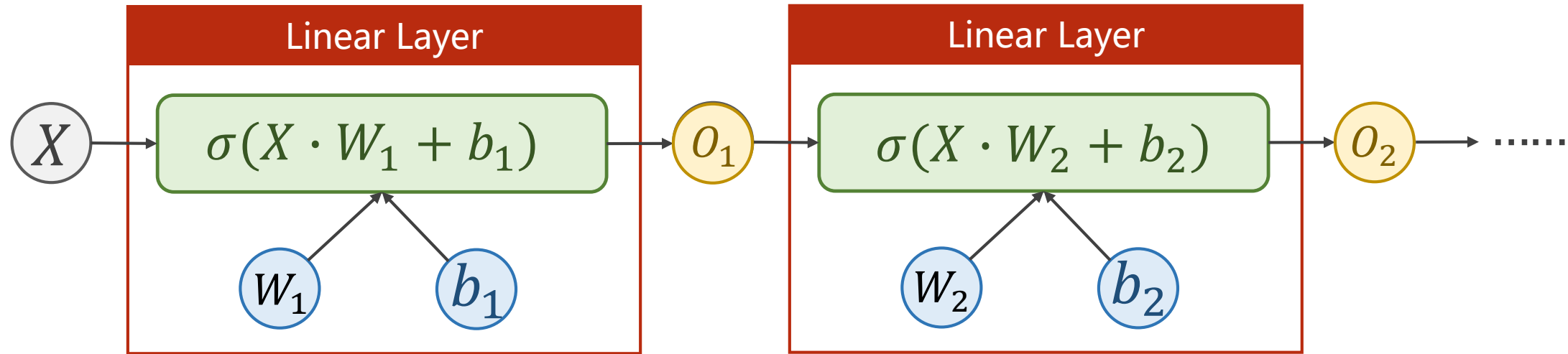


Linear Layer

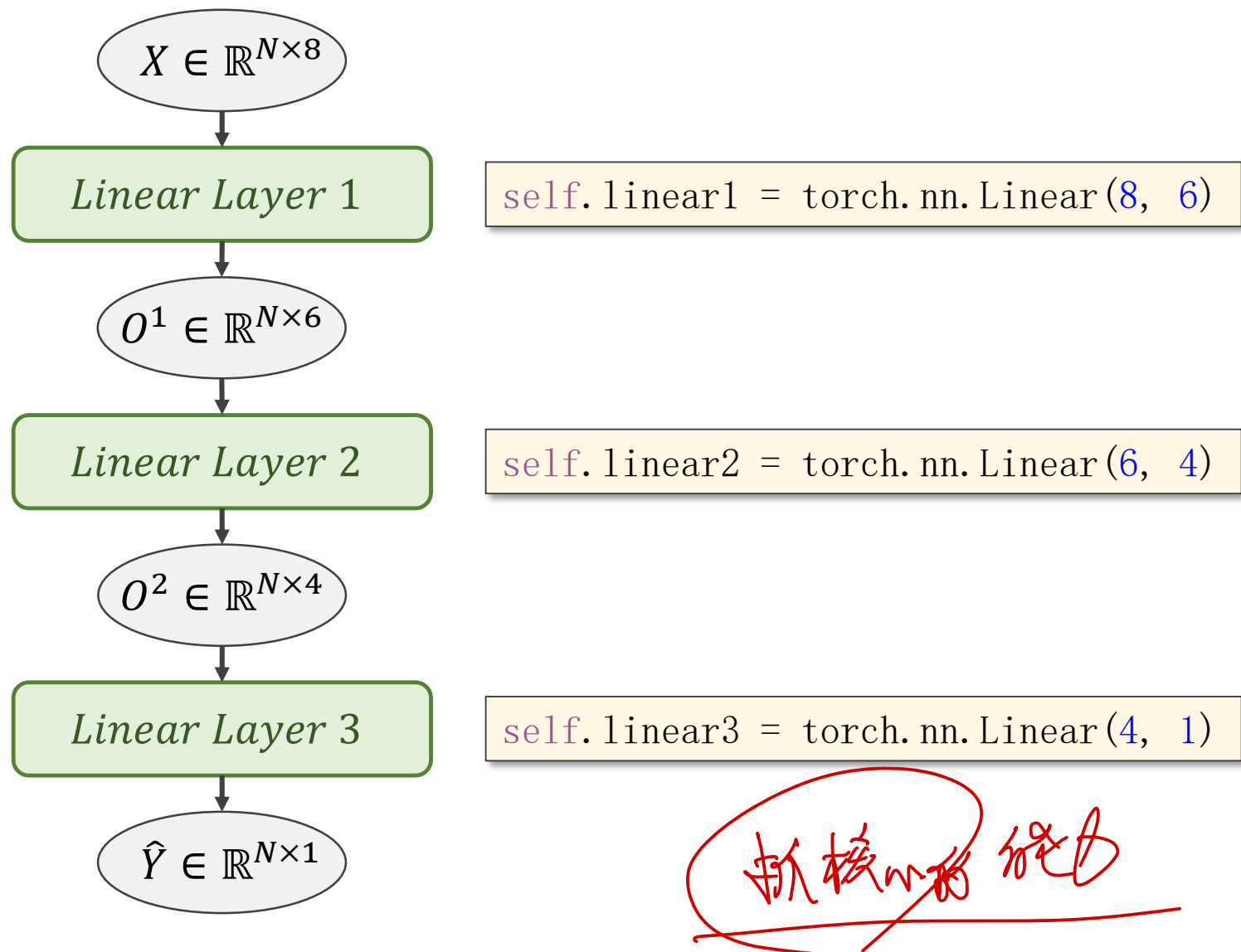
```
self.linear = torch.nn.Linear(8, 6)
```



Neural Network



Example: Artificial Neural Network



范化能力

读文档，
基本架构的
理解

抓核心能力

Example: Diabetes Prediction

X1	X2	X3	X4	X5	X6	X7	X8	Y
-0.29	0.49	0.18	-0.29	0.00	0.00	-0.53	-0.03	0
-0.88	-0.15	0.08	-0.41	0.00	-0.21	-0.77	-0.67	1
-0.06	0.84	0.05	0.00	0.00	-0.31	-0.49	-0.63	0
-0.88	-0.11	0.08	-0.54	-0.78	-0.16	-0.92	0.00	1
0.00	0.38	-0.34	-0.29	-0.60	0.28	0.89	-0.60	0
-0.41	0.17	0.21	0.00	0.00	-0.24	-0.89	-0.70	1
-0.65	-0.22	-0.18	-0.35	-0.79	-0.08	-0.85	-0.83	0
0.18	0.16	0.00	0.00	0.00	0.05	-0.95	-0.73	1
-0.76	0.98	0.15	-0.09	0.28	-0.09	-0.93	0.07	0
-0.06	0.26	0.57	0.00	0.00	0.00	-0.87	0.10	0

Handwritten red text: 数据集

Example: Diabetes Prediction

1

Prepare dataset
we shall talk about this later

2

Design model using Class
inherit from `nn.Module`

3

Construct loss and optimizer
using PyTorch API

4

Training cycle
forward, backward, update

Example: 1. Prepare Dataset

```
import numpy as np
xy = np.loadtxt('diabetes.csv.gz', delimiter=',', dtype=np.float32)
x_data = torch.from_numpy(xy[:, :-1])
y_data = torch.from_numpy(xy[:, [-1]])
```



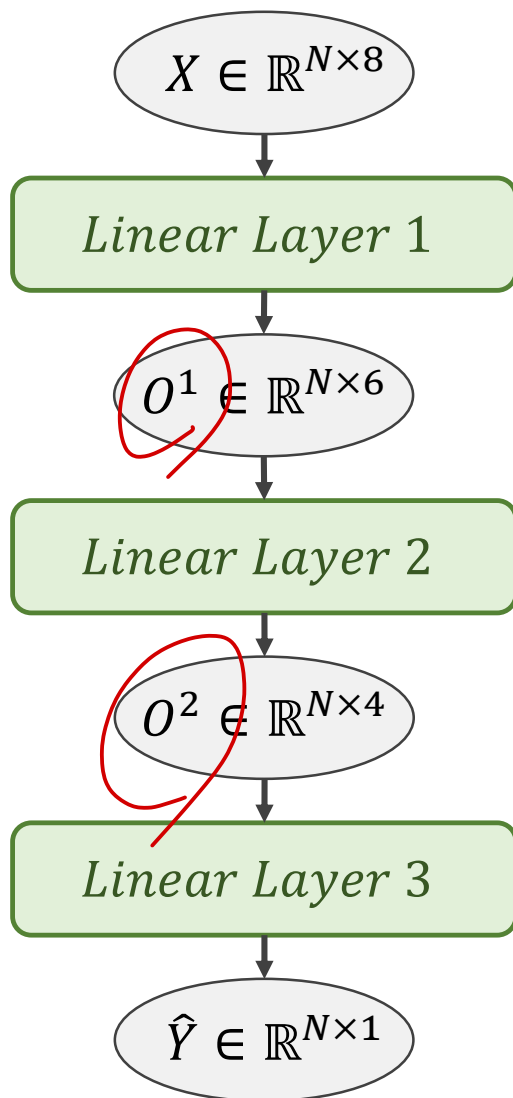
1080

2080

32 float.

```
diabetes.csv
1 -0.294118,0.487437,0.180328,-0.292929,0,0.00149028,-0.53117,-0.0333333,0
2 -0.882353,-0.145729,0.0819672,-0.414141,0,-0.207153,-0.766866,-0.666667,1
3 -0.0588235,0.839196,0.0491803,0,0,-0.305514,-0.492741,-0.633333,0
4 -0.882353,-0.105528,0.0819672,-0.535354,-0.777778,-0.162444,-0.923997,0,1
5 0,0.376884,-0.344262,-0.292929,-0.602837,0.28465,0.887276,-0.6,0
6 -0.411765,0.165829,0.213115,0,0,-0.23696,-0.894962,-0.7,1
7 -0.647059,-0.21608,-0.180328,-0.353535,-0.791962,-0.0760059,-0.854825,-0.833333,0
8 0.176471,0.155779,0,0,0,0.052161,-0.952178,-0.733333,1
9 -0.764706,0.979899,0.147541,-0.0909091,0.283688,-0.0909091,-0.931682,0.0666667,0
10 -0.0588235,0.256281,0.57377,0,0,0,-0.868488,0.1,0
11 -0.529412,0.105528,0.508197,0,0,0.120715,-0.903501,-0.7,1
12 0.176471,0.688442,0.213115,0,0,0.132638,-0.608027,-0.566667,0
13 0.176471,0.396985,0.311475,0,0,-0.19225,0.163962,0.2,1
14 -0.882353,0.899497,-0.0163934,-0.535354,1,-0.102832,-0.726729,0.266667,0
15 -0.176471,0.885823,0,0,0,0.185818,-0.553333,-0.633333,0
```

Example: 2. Define Model



```
import torch
```

```
class Model(torch.nn.Module):
```

```
    def __init__(self):
```

```
        super(Model, self).__init__()
```

```
        self.linear1 = torch.nn.Linear(8, 6)
```

```
        self.linear2 = torch.nn.Linear(6, 4)
```

```
        self.linear3 = torch.nn.Linear(4, 1)
```

```
        self.sigmoid = torch.nn.Sigmoid()
```

模块

```
    def forward(self, x):
```

```
        x = self.sigmoid(self.linear1(x))
```

```
        x = self.sigmoid(self.linear2(x))
```

```
        x = self.sigmoid(self.linear3(x))
```

```
        return x
```

```
model = Model()
```

Example: 3. Construct Loss and Optimizer

Mini-Batch Loss Function for Binary Classification

$$loss = -\frac{1}{N} \sum_{n=1}^N y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n)$$

Update

$$\omega = \omega - \alpha \frac{\partial cost}{\partial \omega}$$

```
criterion = torch.nn.BCELoss(size_average=True)
optimizer = torch.optim.SGD(model.parameters(), lr=0.1)
```

Example: 4. Training Cycle

```
for epoch in range(100):  
    # Forward  
    y_pred = model(x_data)  
    loss = criterion(y_pred, y_data)  
    print(epoch, loss.item())  
  
    # Backward  
    optimizer.zero_grad()  
    loss.backward()  
  
    # Update  
    optimizer.step()
```

NOTICE:

This program has not use **Mini-Batch** for training.

We shall talk about **DataLoader** later.

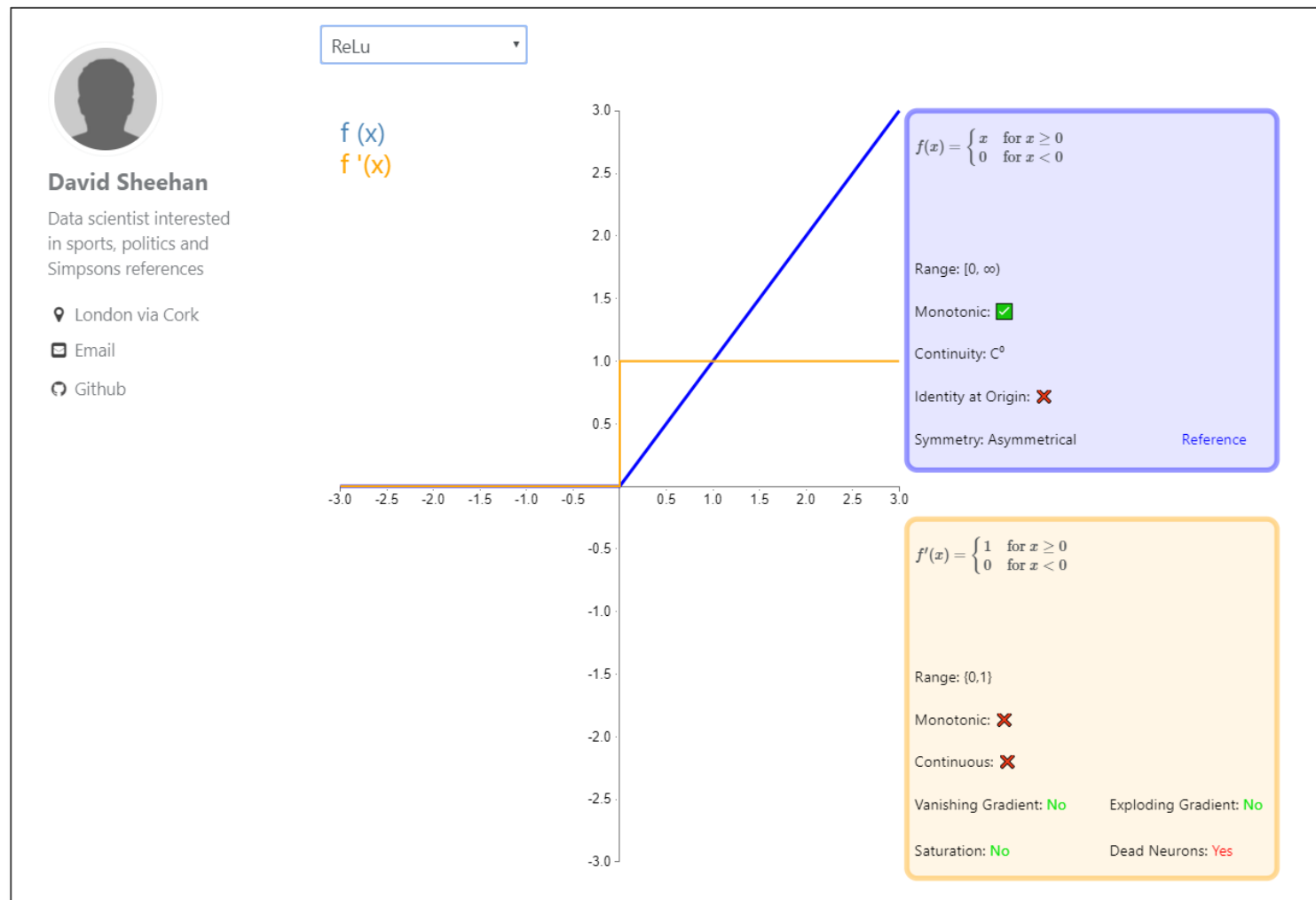
Exercise: Try different activate function

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0 \\ 0.5, & z = 0 \\ 1, & z > 0 \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0 \\ 0, & z = 0 \\ 1, & z > 0 \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2} \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2} \\ 0, & z \leq -\frac{1}{2} \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, <u>ReLU</u> (Rectified Linear Unit)	$\phi(z) = \max(0, z)$	Multi-layer Neural Networks	
Rectifier, <u>softplus</u>	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

Copyright © Sebastian Raschka 2016
(<http://sebastianraschka.com>)

http://rasbt.github.io/mlxtend/user_guide/general_concepts/activation-functions/#activation-functions-for-artificial-neural-networks

Exercise: Try different activate function



<https://dashee87.github.io/data%20science/deep%20learning/visualising-activation-functions-in-neural-networks/>

Exercise: Try different activate function

Handwritten red mark: a stylized signature or symbol.

Non-linear activations (weighted sum, nonlinearity)

- ELU
- Hardshrink
- Hardtanh
- LeakyReLU
- LogSigmoid
- PReLU
- ReLU
- ReLU6
- RReLU
- SELU
- Sigmoid
- Softplus
- Softshrink
- Softsign
- Tanh
- Tanhshrink
- Threshold

Non-linear activations (weighted sum, nonlinearity)

```
class torch.nn.ELU(alpha=1.0, inplace=False) [source]
```

Applies element-wise, $\text{ELU}(x) = \max(0, x) + \min(0, \alpha * (\exp(x) - 1))$

Parameters:

- **alpha** – the α value for the ELU formulation. Default: 1.0
- **inplace** – can optionally do the operation in-place. Default: `False`

Shape:

- Input: $(N, *)$ where $*$ means, any number of additional dimensions
- Output: $(N, *)$, same shape as the input

ELU activation function

<https://pytorch.org/docs/stable/nn.html#non-linear-activations-weighted-sum-nonlinearity>

Exercise: Try different activate function

```
import torch

class Model(torch.nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.linear1 = torch.nn.Linear(8, 6)
        self.linear2 = torch.nn.Linear(6, 4)
        self.linear3 = torch.nn.Linear(4, 1)
        self.activate = torch.nn.ReLU()

    def forward(self, x):
        x = self.activate(self.linear1(x))
        x = self.activate(self.linear2(x))
        x = self.activate(self.linear3(x))
        return x

model = Model()
```

Sign: 01



PyTorch Tutorial

07. Multiple Dimension Input