PyTorch Tutorial

07. Multiple Dimension Input

Revision

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

回归 $y \in R$

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

分类 $y \in \{\ \}$

Diabetes Dataset 糖尿病分类

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

Sample 一个样本

||

数据库：record

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 特征 = 数据库.字段

PS: 结构化数据 ~ 关系型数据库" 二维表形式

机器学习中
- 一部分作为输入 [X]
- 一部分作为输入 [Y]

(X, 10列) diabetes_data.csv.gz
(Y, 1列) diabetes_target.csv.gz

linux下流行的压缩文件格式

- excel 以逗号/tab分隔
- 记事本以空格分隔

Multiple Dimension Logistic Regression Model

Logistic Regression Model

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

$$\Downarrow$$

Logistic Regression Model

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

$$\left(\begin{bmatrix} w_1 & \cdots & w_8 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_8 \end{bmatrix} \right)^T \quad w^T x$$

$$\parallel$$

$$\begin{bmatrix} x_1^{(i)} & \cdots & x_8^{(i)} \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_8 \end{bmatrix} \quad x^T w$$

Multiple Dimension Logistic Regression Model

Logistic Regression Model

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

Logistic Regression Model

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

$x^{(i)T} \cdot w$

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

Logistic Regression Model

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

n个样本 n维特征

Logistic Regression Model

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

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

$\sigma = \frac{1}{1+e^{-z}}$

Logistic Regression Model

$$\hat{y}^{(i)} = \sigma(\begin{bmatrix} x_1^{(i)} & \cdots & x_8^{(i)} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \vdots \\ \omega_8 \end{bmatrix} + b)$$

$$= \sigma(z^{(i)})$$

Mini-Batch (N samples)

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

Sigmoid function is in an element-wise fashion.

Sigmoid函数是按向量计算的形式

$$\text{torch.exp}(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}) = \begin{bmatrix} e^{x_1} \\ e^{x_2} \\ e^{x_3} \end{bmatrix}$$

类似于numpy中的向量函数

Mini-Batch (N samples)

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

Sigmoid function is in an element-wise fashion.

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

Mini-Batch (N samples)

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

Sigmoid function is in an element-wise fashion.

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

$$\vdots$$

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

$\Rightarrow$

$$\underbrace{\begin{bmatrix} z^{(1)} \\ \vdots \\ z^{(N)} \end{bmatrix}}_{N \times 1} = \underbrace{\begin{bmatrix} x_1^{(1)} & \cdots & x_8^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(N)} & \cdots & x_8^{(N)} \end{bmatrix}}_{N \times 8} \underbrace{\begin{bmatrix} \omega_1 \\ \vdots \\ \omega_8 \end{bmatrix}}_{8 \times 1} + \underbrace{\begin{bmatrix} b \\ \vdots \\ b \end{bmatrix}}_{N \times 1}$$

Mini-Batch (N samples)

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

```python
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()
```

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

$$\vdots$$

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

$$\underbrace{\begin{bmatrix} z^{(1)} \\ \vdots \\ z^{(N)} \end{bmatrix}}_{N \times 1} = \underbrace{\begin{bmatrix} x_1^{(1)} & \cdots & x_8^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(N)} & \cdots & x_8^{(N)} \end{bmatrix}}_{N \times 8} \underbrace{\begin{bmatrix} \omega_1 \\ \vdots \\ \omega_8 \end{bmatrix}}_{8 \times 1} + \underbrace{\begin{bmatrix} b \\ \vdots \\ b \end{bmatrix}}_{N \times 1}$$

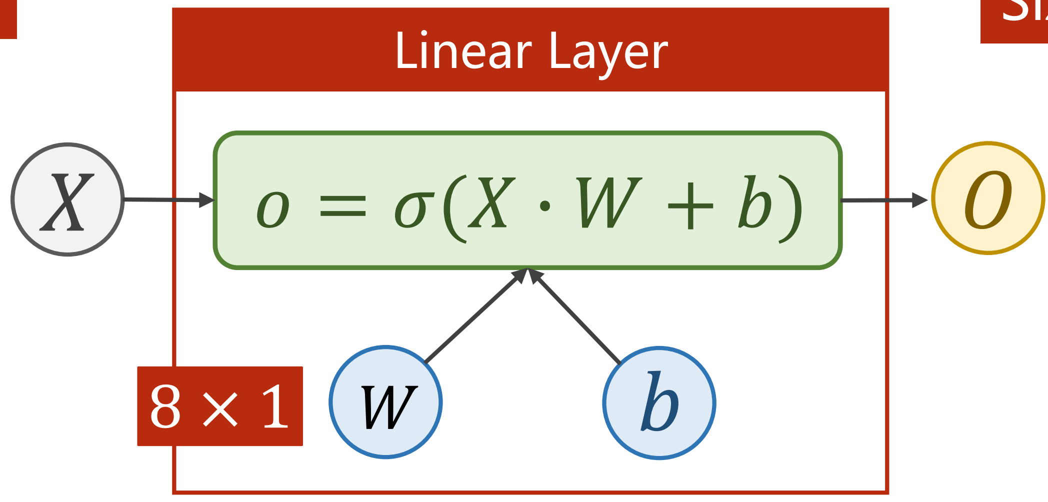
8D ⇒ 1D

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

Size of each input sample

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

feature = 8



Size of each output sample

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

feature = 1

$8D \Rightarrow 2D$

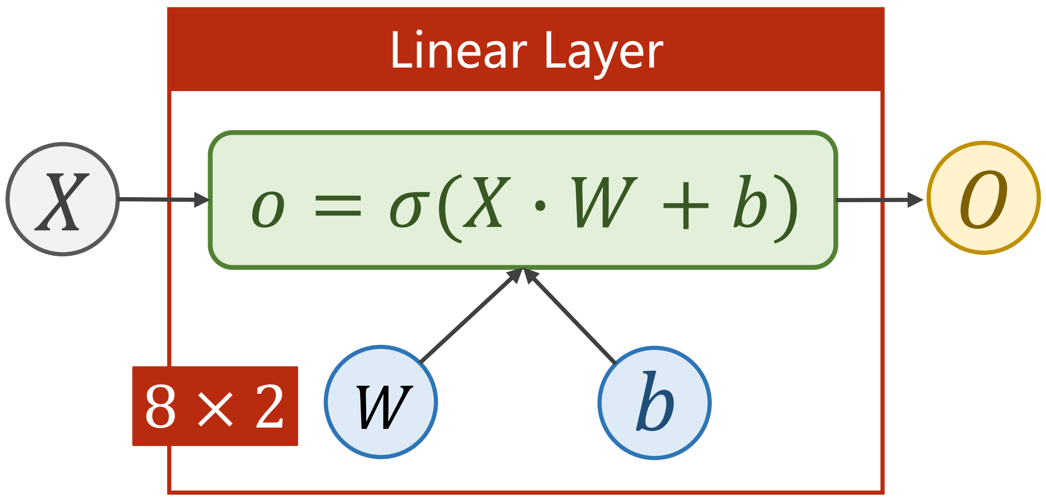
```
self.linear = torch.nn.Linear(8, 2)
```

$(2, 1)$

空间映射：

$$\underset{M\times 1}{y} = \underset{M\times N}{A}\,\underset{N\times 1}{x} \Rightarrow \begin{cases} x \in R^N \\ y \in R^M \\ A \in R^{M\times N} \end{cases}$$

$$X = \begin{array}{c} \quad 8 \\ \begin{bmatrix} x_1^{(1)} & \cdots & x_8^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(N)} & \cdots & x_8^{(N)} \end{bmatrix} \end{array}$$



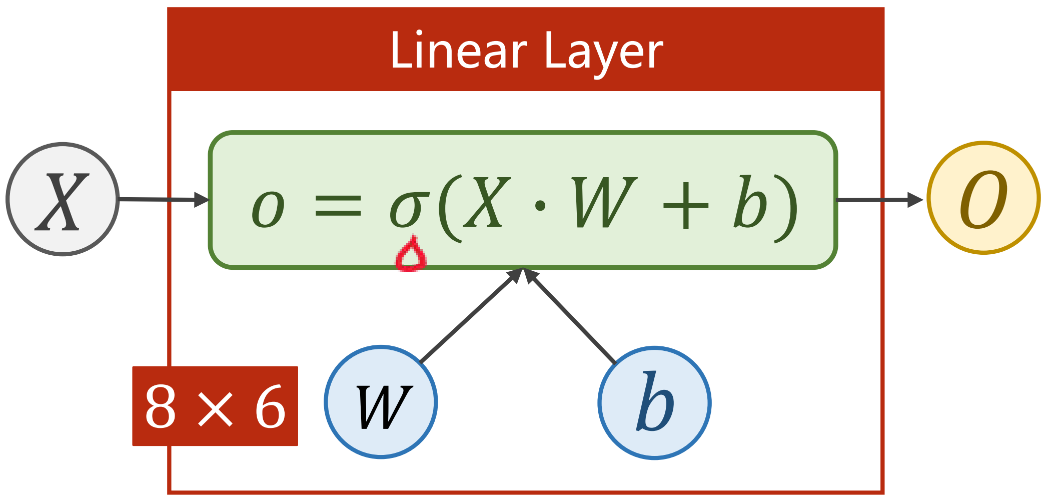
$$O = \begin{array}{c} \quad 2 \\ \begin{bmatrix} o_1^{(1)} & o_2^{(1)} \\ \vdots & \vdots \\ o_1^{(N)} & o_2^{(N)} \end{bmatrix} \end{array}$$

8D ⇒ 6D

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

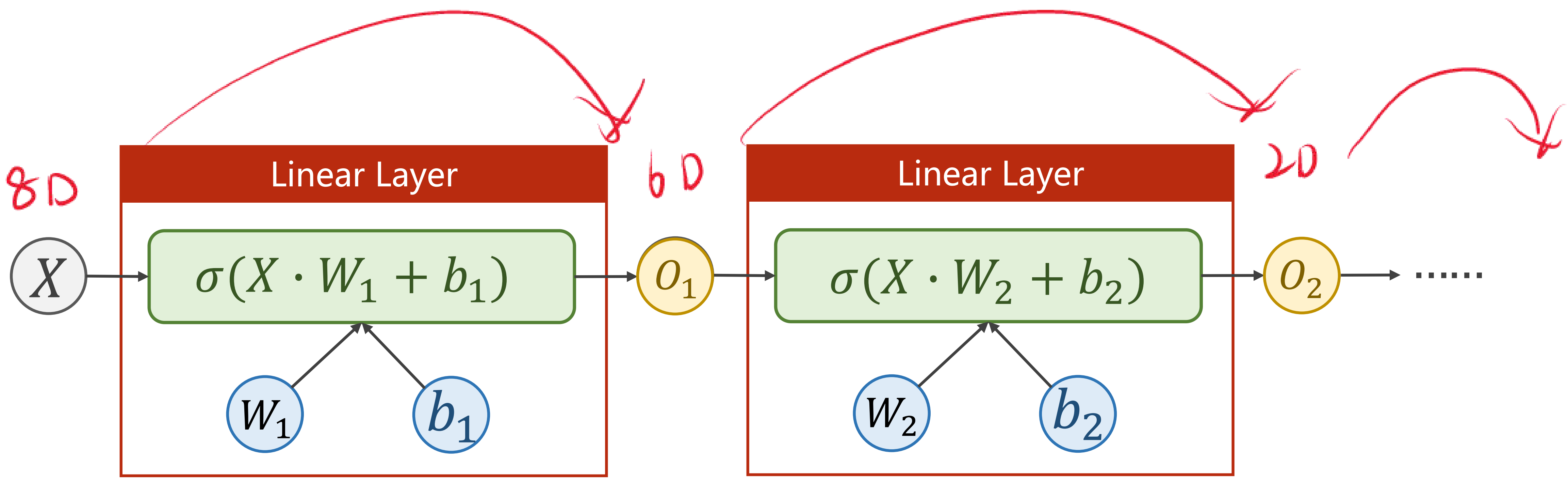
8D $\xrightarrow{\text{[illegible]降维}}$ 1D

$$X = \begin{array}{c} 8 \\ \begin{bmatrix} x_1^{(1)} & \cdots & x_8^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(N)} & \cdots & x_8^{(N)} \end{bmatrix} \end{array}$$

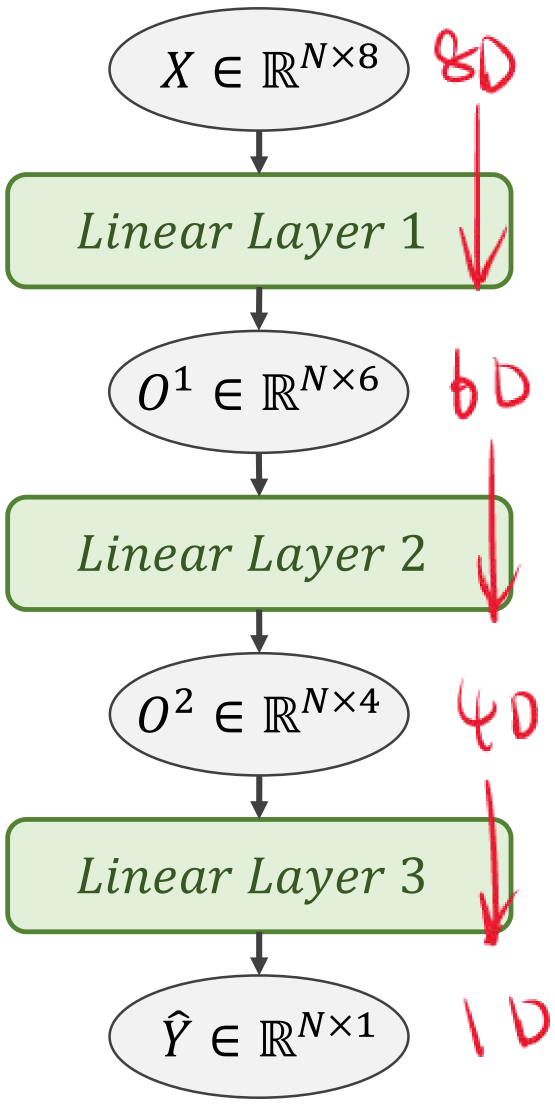


$$O = \begin{array}{c} 6 \\ \begin{bmatrix} o_1^{(1)} & \cdots & o_6^{(1)} \\ \vdots & \ddots & \vdots \\ o_1^{(N)} & \cdots & o_6^{(N)} \end{bmatrix} \end{array}$$

Neural Network



eg: 8D → 24D → 12D → ...




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

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

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

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

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

```python
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]])
```

分隔符是逗号

使用32位浮点数，一般显卡只支持32位float，贵的显卡才支持64位double

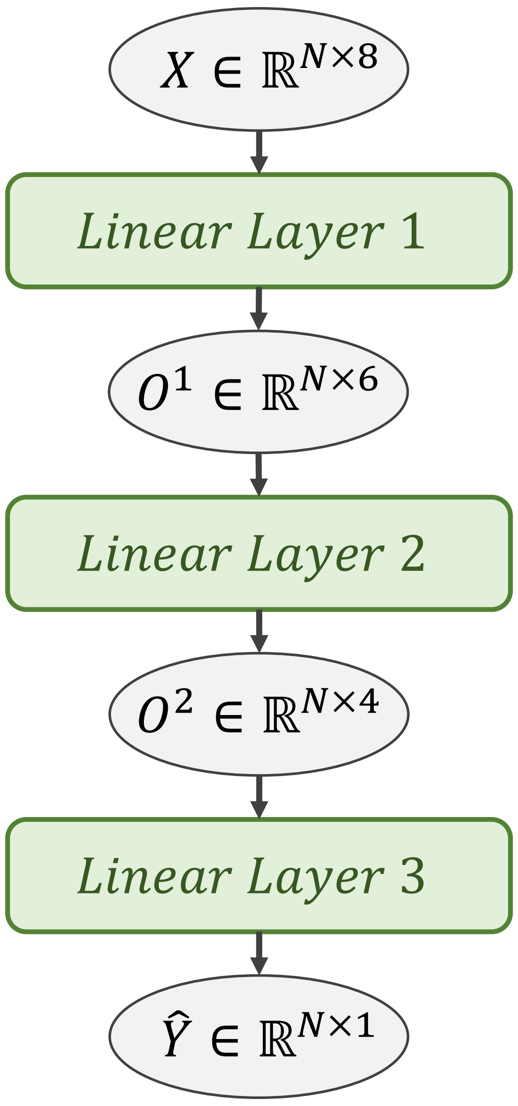
从torch的numpy中生成两个Tensor

diabetes.csv

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


x_data = torch.from_numpy(xy[:,:-1])
取前八列
第一个: 是指读取所有行
第二个: 是指从第一列开始，最后一列不要

y_data = torch.from_numpy(xy[:, [-1]]
取最后一列
[-1] 最后得到的是个矩阵
如果没有中括号，拿出来的是向量
https://blog.csdn.net/shenggedeqiang/article/details/84856051




```
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()

    def forward(self, x):
  O1    x = self.sigmoid(self.linear1(x))
  O2    x = self.sigmoid(self.linear2(x)) O1
  O3    x = self.sigmoid(self.linear3(x)) O2
        return x

model = Model()
```

$\sigma = \frac{1}{1+e^{-z}}$

是个运算模块
不包含任何参数
所以只定义一个，
但用于构造计算图

import torch.nn.Functional as F

F.sigmoid(self.linear(x))

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

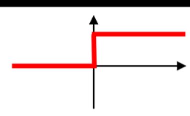
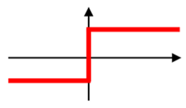
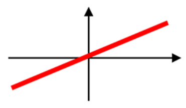
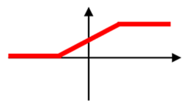
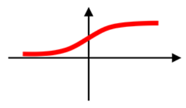
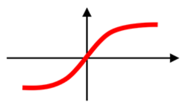
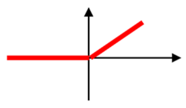
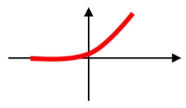
```
for epoch in range(100):
    # Forward
    y_pred = model(x_data)
    loss = criterion(y_pred, y_data)
    print(epoch, loss.item())

    # Backward
    optimizer.zero_grad()
    loss.backward()

    # Update
    optimizer.step()
```

NOTICE:

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

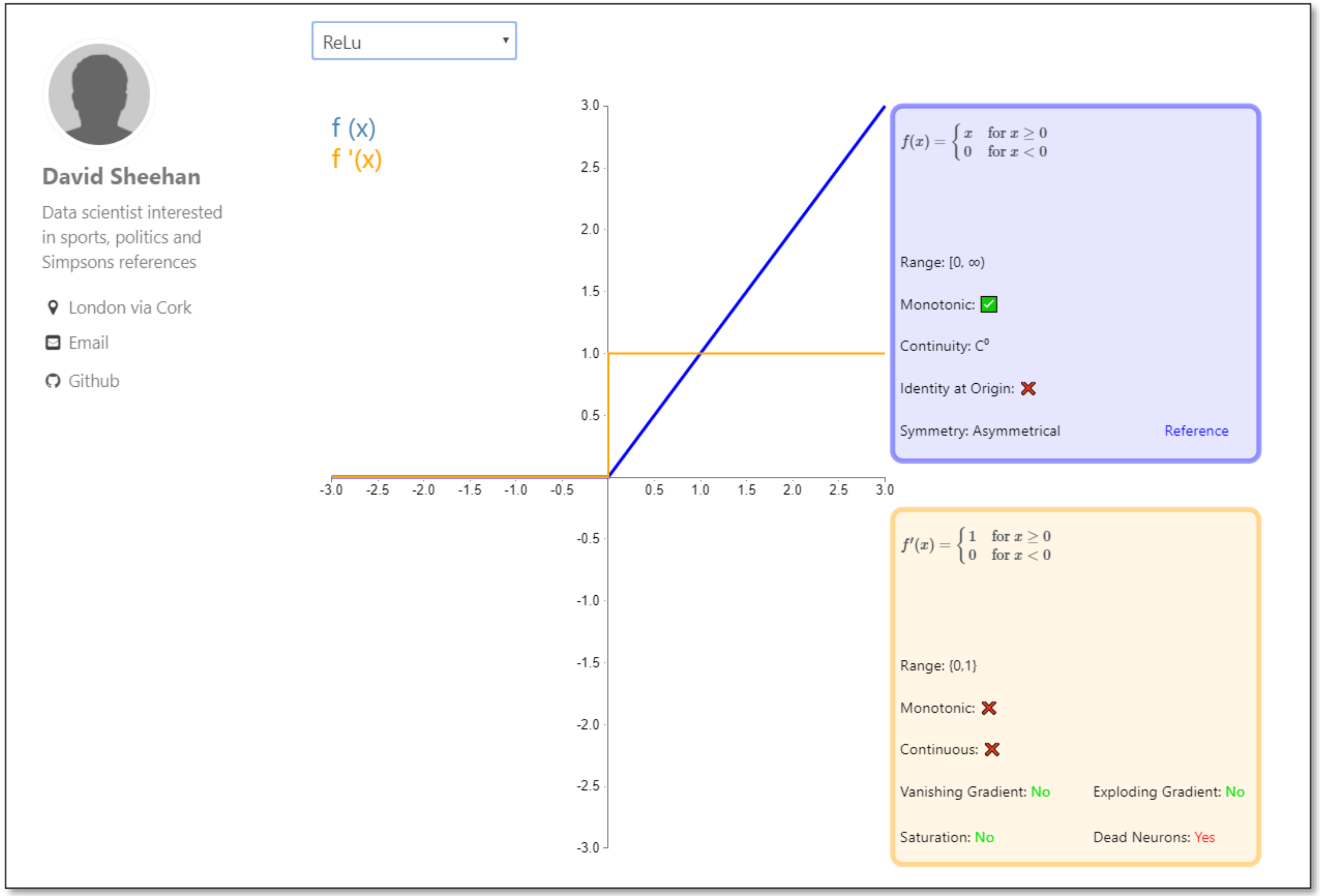
We shall talk about **DataLoader** later.

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, ReLU (Rectified Linear Unit)	$\phi(z)=max(0,z)$	Multi-layer Neural Networks	 [0, 1]
Rectifier, softplus 	$\phi(z)=\ln(1+e^{z})$	Multi-layer Neural Networks	

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

- 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)

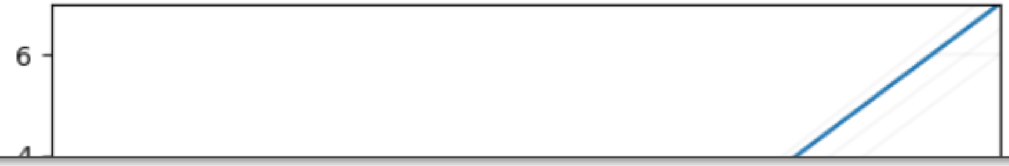
class `torch.nn.ELU`*(alpha=1.0, inplace=False)* [source]

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 其它：sigmoid

model = Model()
```

torch.nn
- Module：模型继承Module
- Linear：线性模型
- MSEloss：平均损失值
- BCEloss：交叉熵损失值
- functional：
- Sigmoid：非线性因素sigmoid函数调用

import torch.nn.functional as F

F.sigmoid(self.linear(x))

self.sigmoid = torch.nn.Sigmoid()

PyTorch Tutorial

torch.Tensor()：定义张量

torch.optim.SGD/Adam：优化器

07. Multiple Dimension Input

np.
- linspace(start, end, num) 等间隔num个点
- loadtxt('', delimiter=',', dtype=np.float32)

.item() 取出张量中的数值

torch.Tensor(x).view(-1,1)：数组numpy $\xrightarrow{\text{reshape}}$ Tensor

torch.from_numpy(xy[:, :-1])：数组numpy $\xrightarrow{\text{选择}}$ Tensor

y.data.numpy()：Tensor → 数组numpy

xy.shape[0]：得到矩阵列数