FNN

This is a Simple Fully-Connected-Neural-Network for beginners, unused Pytorch, only using Numpy.

构建一个全连接的神经网络（FNN）
[A Simple Fully-Connected-Neural-Network for beginners]

Pytorch 框架

- 别人已经搭建好了，易调用
- 问题：初学者不易理解其中较深刻的矩阵变换

仅用Numpy库，从零开始写一个FNN

- 从零开始构建一个全连接的神经网络（FNN）

Lable

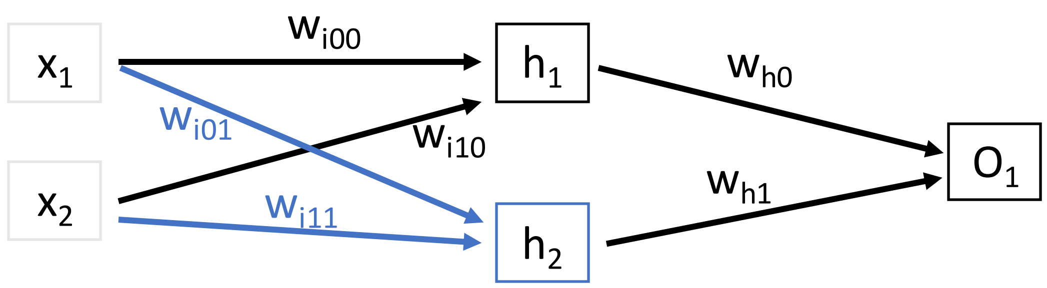
$$\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

$$\underset{4\times 2}{\overset{X}{\begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}}} \quad \underset{2\times 2}{\overset{W\_i}{\begin{bmatrix} w_{i00} & w_{i01} \\ w_{i10} & w_{i11} \end{bmatrix}}} + \underset{1\times 2}{\overset{b}{\begin{bmatrix} 0 & 0 \end{bmatrix}}} = \underset{4\times 2}{\overset{h}{\begin{bmatrix} 0 * w_{i00} + 0 * w_{i10} & 0 * w_{i01} + 0 * w_{i11} \\ 0 * w_{i00} + 1 * w_{i10} & 0 * w_{i01} + 1 * w_{i11} \\ 1 * w_{i00} + 0 * w_{i10} & 1 * w_{i01} + 0 * w_{i11} \\ 1 * w_{i00} + 1 * w_{i10} & 1 * w_{i01} + 1 * w_{i11} \end{bmatrix}}}$$

$$\underset{4\times 2}{\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}}$$

$$\underset{4\times 1}{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}$$

$$\underset{4\times 2}{\overset{h}{\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}}} \quad \underset{2\times 1}{\overset{W\_h}{\begin{bmatrix} w_{h0} \\ w_{h1} \end{bmatrix}}} + \underset{1\times 1}{\overset{b}{\begin{bmatrix} 0 \end{bmatrix}}} = \underset{4\times 1}{\overset{Output}{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}}$$



$$\begin{bmatrix} x_1 & x_2 \\ & \\ & \end{bmatrix} \quad \begin{bmatrix} w_{i00} & w_{i01} \\ w_{i10} & w_{i11} \end{bmatrix} \quad \begin{bmatrix} w_{h0} \\ w_{h1} \end{bmatrix}$$

$$
\underset{\text{Lable}}{\begin{bmatrix}0\\1\\1\\0\end{bmatrix}}
\qquad
\underset{4\times2}{\overset{X}{\begin{bmatrix}0&0\\0&1\\1&0\\1&1\end{bmatrix}}}
\qquad
\underset{2\times2}{\overset{W_i}{\begin{bmatrix}w_{i00}&w_{i01}\\w_{i10}&w_{i11}\end{bmatrix}}}
+
\underset{1\times2}{\overset{b}{\begin{bmatrix}0&0\end{bmatrix}}}
=
\underset{4\times2}{\overset{\text{Output}}{\begin{bmatrix}
0*w_{i00}+0*w_{i10} & 0*w_{i01}+0*w_{i11}\\
0*w_{i00}+1*w_{i10} & 0*w_{i01}+1*w_{i11}\\
1*w_{i00}+0*w_{i10} & 1*w_{i01}+0*w_{i11}\\
1*w_{i00}+1*w_{i10} & 1*w_{i01}+1*w_{i11}
\end{bmatrix}}}
$$

$$\xrightarrow{\text{Sigmoid}}$$

$$
\underset{4\times2}{\overset{h}{\begin{bmatrix}
\text{Sigmoid}(0*w_{i00}+0*w_{i10}) & \text{Sigmoid}(0*w_{i01}+0*w_{i11})\\
\text{Sigmoid}(0*w_{i00}+1*w_{i10}) & \text{Sigmoid}(0*w_{i01}+1*w_{i11})\\
\text{Sigmoid}(1*w_{i00}+0*w_{i10}) & \text{Sigmoid}(1*w_{i01}+0*w_{i11})\\
\text{Sigmoid}(1*w_{i00}+1*w_{i10}) & \text{Sigmoid}(1*w_{i01}+1*w_{i11})
\end{bmatrix}}}
$$

(1)

$$\frac{\partial\ (\text{Output})}{\partial\ (w_i)} = ?$$

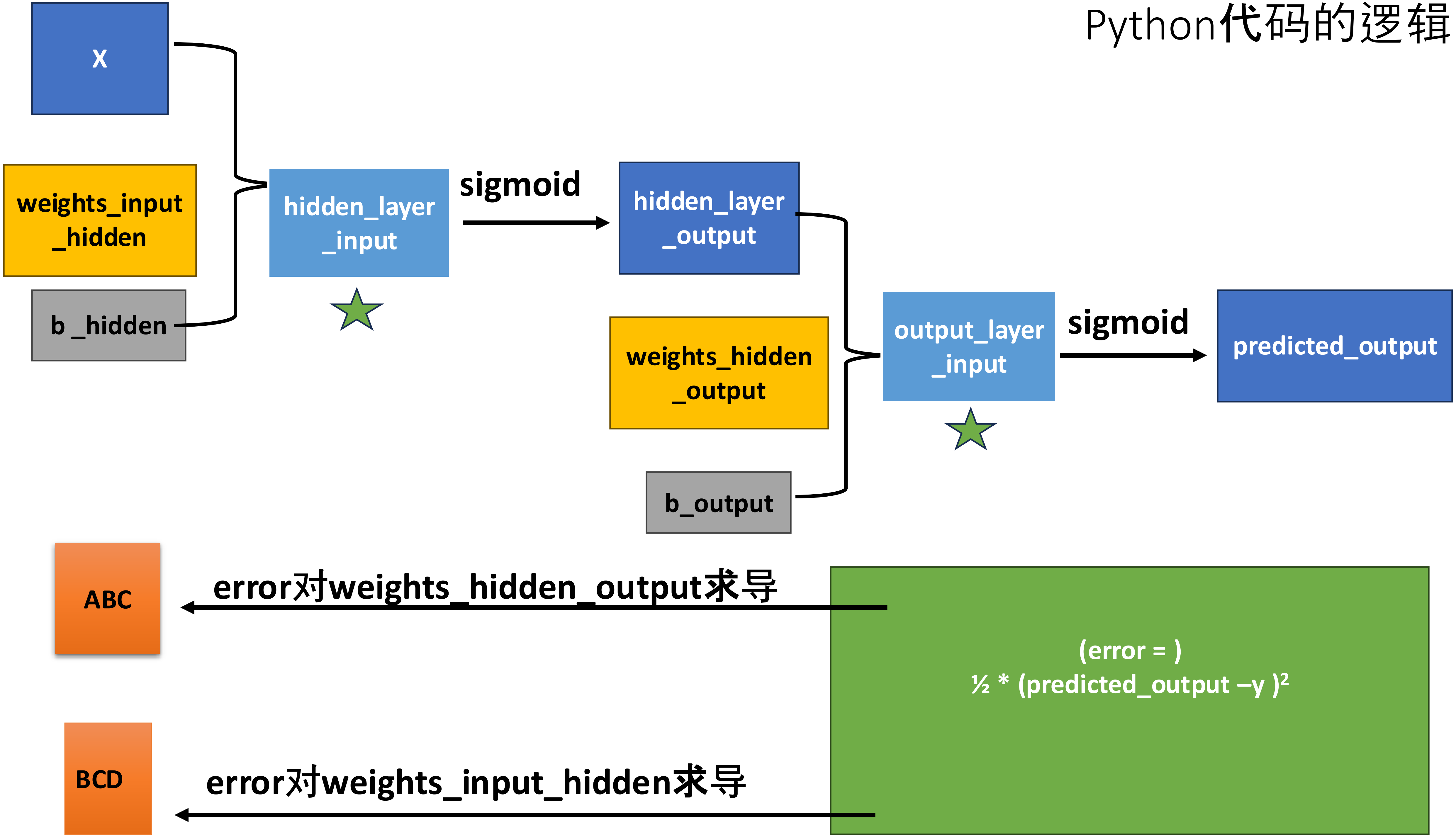
$$z=f(Y), Y=AX+B \rightarrow \frac{\partial z}{\partial X} = A^T\frac{\partial z}{\partial Y}$$

$$z=f(Y), Y=XA+B \rightarrow \frac{\partial z}{\partial X} = \frac{\partial z}{\partial Y}A^T$$

(2) 当权重更新时:

$$W_{i\_new} = W_{i\_old} + \boxed{\quad} \times \alpha$$

Python代码的逻辑



Derivative

ABC

$$ABC = \frac{\partial\ (\text{error})}{\partial\ (\text{weights\_hidden\_output})} = \underbrace{\frac{\partial\ (\text{error})}{\partial\ (\textit{predicted\_output})} * \frac{\partial\ (\textit{predicted\_output})}{\partial\ (\textit{output\_layer\_input})}}_{\text{d\_predicted\_output}} * \underbrace{\frac{\partial\ (\textit{output\_layer\_input})}{\partial\ (\text{weights\_hidden\_output})}}_{\text{hidden\_layer\_output}^{T}} = \text{hidden\_layer\_output}^{T}\ \text{.dot(d\_predicted\_output)}$$

BCD

$$BCD = = \frac{\partial\ (\text{error})}{\partial\ (\text{weights\_input\_hidden})} = \underbrace{\underbrace{\underbrace{\frac{\partial\ (\text{error})}{\partial\ (\textit{predicted\_output})} * \frac{\partial\ (\textit{predicted\_output})}{\partial\ (\textit{output\_layer\_input})}}_{\text{d\_predicted\_output}} * \text{weights\_hidden\_output}^{T}}_{\text{error\_hidden\_layer}} * \frac{\partial\ (\textit{hidden\_layer\_output})}{\partial\ (\textit{hidden\_layer\_input})}}_{\text{d\_hidden\_layer}} * \frac{\partial\ (\textit{hidden\_layer\_input})}{\partial\ (\text{weights\_input\_hidden})}$$

$$= X^{T}\ \text{.dot(d\_hidden\_layer)}$$

Python中两个矩阵（A, B）相乘 [Multiplying two matrices (A, B) in Python]

A .dot (B)

1. 同线性代数中矩阵乘法的定义： np.dot()

np.dot(A, B)：对于二维矩阵，计算真正意义上的矩阵乘积，同线性代数中矩阵乘法的定义。对于一维矩阵，计算两者的内积。见如下Python代码：

```
import numpy as np

# 2-D array: 2 x 3
two_dim_matrix_one = np.array([[1, 2, 3], [4, 5, 6]])
# 2-D array: 3 x 2
two_dim_matrix_two = np.array([[1, 2], [3, 4], [5, 6]])

two_multi_res = np.dot(two_dim_matrix_one, two_dim_matrix_two)
print('two_multi_res: %s' %(two_multi_res))

# 1-D array
one_dim_vec_one = np.array([1, 2, 3])
one_dim_vec_two = np.array([4, 5, 6])
one_result_res = np.dot(one_dim_vec_one, one_dim_vec_two)
print('one_result_res: %s' %(one_result_res))
```

结果如下：

```
two_multi_res: [[22 28]
 [49 64]]
one_result_res: 32
```

A * B

2. 对应元素相乘 element-wise product: np.multiply(), 或 *

在Python中，实现对应元素相乘，有2种方式，一个是np.multiply()，另外一个是*。见如下Python代码：

```
import numpy as np

# 2-D array: 2 x 3
two_dim_matrix_one = np.array([[1, 2, 3], [4, 5, 6]])
another_two_dim_matrix_one = np.array([[7, 8, 9], [4, 7, 1]])

# 对应元素相乘 element-wise product
element_wise = two_dim_matrix_one * another_two_dim_matrix_one
print('element wise product: %s' %(element_wise))

# 对应元素相乘 element-wise product
element_wise_2 = np.multiply(two_dim_matrix_one, another_two_dim_matrix_one)
print('element wise product: %s' % (element_wise_2))
```

结果如下：

```
element wise product: [[ 7 16 27]
 [16 35  6]]
element wise product: [[ 7 16 27]
 [16 35  6]]
```

• 矩阵的求导 (Derivative of a Matrix)

- 链式求导法则

矩阵求导也有链式法则，不过与标量求导中的链式法则形式不同。链式法则可以通过微分法推导出来，在某些情况下直接使用链式法则会比使用微分法更简洁一些。

标量对多向量链式求导，中间变量都是向量的情况：

$$\frac{\partial z}{\partial \mathbf{x}} = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)^T \frac{\partial z}{\partial \mathbf{y}}$$

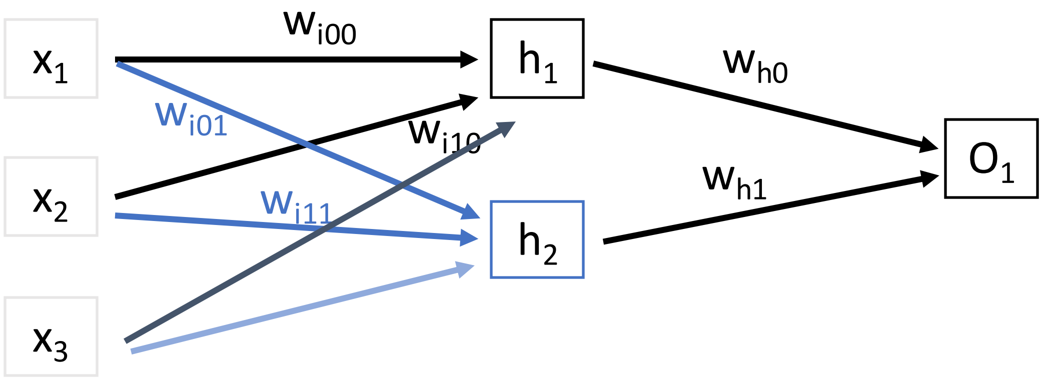
$$\frac{\partial z}{\partial \mathbf{y}_1} = \left(\frac{\partial \mathbf{y}_n}{\partial \mathbf{y}_{n-1}} \frac{\partial \mathbf{y}_{n-1}}{\partial \mathbf{y}_{n-2}} \cdots \frac{\partial \mathbf{y}_2}{\partial \mathbf{y}_1}\right)^T \frac{\partial z}{\partial \mathbf{y}_n}$$

标量对多矩阵链式求导，中间变量都是矩阵的情况：

$$\boxed{\begin{aligned} z = f(Y), Y = AX + B \rightarrow \frac{\partial z}{\partial X} = A^T \frac{\partial z}{\partial Y} \\ z = f(Y), Y = XA + B \rightarrow \frac{\partial z}{\partial X} = \frac{\partial z}{\partial Y} A^T \end{aligned}}$$

这里，Y可以是矩阵或向量。

Weight of 3x2



$$\begin{bmatrix} x_1 \; x_2 \; x_3 \\ \\ \\ \end{bmatrix} \quad \begin{bmatrix} w_{i00} & w_{i01} \\ w_{i10} & w_{i11} \\ w_{i20} & w_{i21} \end{bmatrix} \quad \begin{bmatrix} w_{h0} \\ w_{h1} \end{bmatrix}$$