Layer init： 初始化參數 W, b (W1 代表第一層)

```
def init_layers(self, nn_architecture, seed = 99):
    np.random.seed(seed)
    params_values = {}

    for layer_idx, layer in enumerate(nn_architecture,1):

        layer_input_size = layer["input_dim"]
        layer_output_size = layer["output_dim"]
        params_values['W' + str(layer_idx)] = np.random.randn(layer_output_size, layer_input_size) * 0.1
        params_values['b' + str(layer_idx)] = np.random.randn(layer_output_size, 1) * 0.1

    return params_values
```

Forward： 分成 single layer, full forward。

- single layer 只做一層該做的事 ：input * W + b 並回傳 output(經 activate function), Z(未經 activate function)。
- full forward 儲存 single layer 輸出，memory{}紀錄每一層 output, Z 並回傳最後的 output 和 memory 給後續計算使用。

```
def single_layer_forward(self, A_prev, W_curr, b_curr, activation="relu"):

    Z_curr = W_curr@A_prev + b_curr

    if activation == "relu":
        activation_func = relu
    elif activation == "sigmoid":
        activation_func = sigmoid


    return activation_func(Z_curr), Z_curr
```

```
def forward(self, X, params_values, nn_architecture):
    memory = {}
    A_curr = X.T

    for idx, layer in enumerate(nn_architecture):
        layer_idx = idx + 1
        A_prev = A_curr

        activ_function_curr = layer["activation"]
        W_curr = params_values["W" + str(layer_idx)]
        b_curr = params_values["b" + str(layer_idx)]
        A_curr, Z_curr = self.single_layer_forward(A_prev, W_curr, b_curr, activ_function_curr)

        memory["A" + str(idx)] = A_prev
        memory["Z" + str(layer_idx)] = Z_curr

    return A_curr, memory
```

Backward: 分成 single layer, full backward。

- single layer 只做一層該做的事 ：
  1. 將 self.error*der_sigmoid(output) 得到 dz_dc
  2. 將 dz_dc @ pre_a(前一層 output) 得到 dW (Wwight 的更新量)
  3. Sum(dz_dc) 得到 db(bias 的更新量)

4. 將當前 W@ dz_dc 得到前一層的 dA(也就是前一層的 loss 數值)
5. 最後回傳 2, 3, 4

- Full backward： 儲存 single layer 輸出裡的 dW, db，用於接下來 update 使用

```python
def single_layer_backward(self, dA_curr, W_curr, Z_curr, A_prev, activation="relu"):

    if activation == "relu":
        backward_activation_func = relu_backward
    elif activation == "sigmoid":
        backward_activation_func = sigmoid_backward

    dZ_dC = backward_activation_func(dA_curr, Z_curr)
    dW_curr = np.dot(dZ_dC, A_prev.T)
    db_curr = np.sum(dZ_dC, axis=1, keepdims=True)
    dA_prev = np.dot(W_curr.T, dZ_dC)

    return dA_prev, dW_curr, db_curr
```

```python
def backward(self, output, label, memory, params_values, nn_architecture):
    grads = {}

    dA_prev = - (np.divide(label, output) - np.divide(1 - label, 1 - output))

    for layer_idx_prev, layer in reversed(list(enumerate(nn_architecture))):
        layer_idx_curr = layer_idx_prev + 1
        activ_function_curr = layer["activation"]

        dA_curr = dA_prev

        A_prev = memory["A" + str(layer_idx_prev)]
        Z_curr = memory["Z" + str(layer_idx_curr)]
        W_curr = params_values["W" + str(layer_idx_curr)]

        dA_prev, dW_curr, db_curr = self.single_layer_backward(
            dA_curr, W_curr, Z_curr, A_prev, activ_function_curr)

        grads["dW" + str(layer_idx_curr)] = dW_curr
        grads["db" + str(layer_idx_curr)] = db_curr

    return grads
```

Update :

New_W = (old W - dW) * lr

New_Bias = (old B - dB) *lr

```python
def update(self, params, grads, nn_architecture, learning_rate):
    for layer_idx, layer in enumerate(nn_architecture,1):
        params["W" + str(layer_idx)] -= learning_rate * grads["dW" + str(layer_idx)]
        params["b" + str(layer_idx)] -= learning_rate * grads["db" + str(layer_idx)]

    return params
```

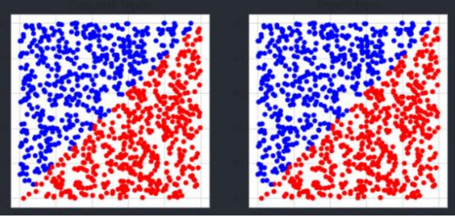
除了 sigmoid 我在最後一層前都使用 relu，並在 backward 用 der_relu:

```python
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def relu(x):
    return np.maximum(0,x)

def der_sigmoid(dA, Z):
    sig = sigmoid(Z)
    return dA * sig * (1 - sig)

def der_relu(dA, Z):
    dZ = np.array(dA, copy = True)
    dZ[Z <= 0] = 0
    return dZ
```

Finally output： 我'linear', 'XOR'都只分別讓他們 train 100,150 epochs 就達到 99%, 95%

```python
data, label = GenData.fetch_data('Linear', 1000)
model = SimpleNet(100)
params = model.train(data, label)
pred_result, _ = model.forward(data, params, model.nn_architecture)
pred_result = np.round(pred_result)
SimpleNet.plot_result(data, label, pred_result.T)
```

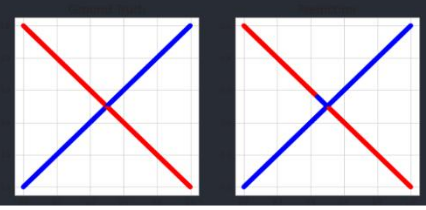
```
Epochs 99: loss : [[0.00017933]]accuracy: 99.06%Training finished
accuracy: 99.06%
```



```python
data, label = GenData.fetch_data('XOR', 1000)
model = SimpleNet(150)
params = model.train(data, label)
```

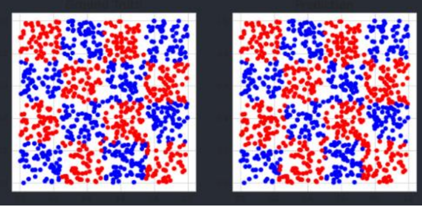
```
accuracy: 50.03% [[-0.50566948]]Training finished
accuracy: 95.47%
```

```python
pred_result, _ = model.forward(data, params, model.nn_architecture)
pred_result = np.round(pred_result)
SimpleNet.plot_result(data, label, pred_result.T)
```



```python
data, label = GenData.fetch_data('Chessboard', 1000)
model = SimpleNet(700)
params = model.train(data, label)
pred_result, _ = model.forward(data, params, model.nn_architecture)
pred_result = np.round(pred_result)
SimpleNet.plot_result(data, label, pred_result.T)
```

```
Epochs 699: loss : [[-0.00073978]]accuracy: 93.09%Training finished
accuracy: 93.09%
```



ablation experiment : use relu vs only use sigmoid

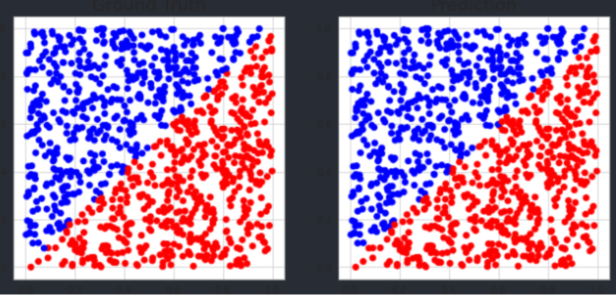
使用了 relu 後，在'linear'的 case 裡 train 15 個 epochs，acc 就到達 96.64，

而只用 sigmoid 必須 train 200 epochs，acc 才能到達 96.19%

use relu

```python
data, label = GenData.fetch_data('Linear', 1000)
model = SimpleNet(15)
params = model.train(data, label)
pred_result, _ = model.forward(data, params, model.nn_architecture)
pred_result = np.round(pred_result)
SimpleNet.plot_result(data, label, pred_result.T)
```

```
Epochs 14: loss : [[0.00741627]]accuracy: 96.64%Training finished
accuracy: 96.64%
```



only use sigmoid

```python
data, label = GenData.fetch_data('Linear', 1000)
model = SimpleNet(200)
params = model.train(data, label)
pred_result, _ = model.forward(data, params, model.nn_architecture)
pred_result = np.round(pred_result)
SimpleNet.plot_result(data, label, pred_result.T)
```

```
Epochs 199: loss : [[0.02305903]]accuracy: 96.19%Training finished
accuracy: 96.19%
```