# 人工智能实验：Topic 5——神经网络

## Part 2：多层神经网络训练

**3190102060　黄嘉欣**

### 一、后向传输（back-propagation）

BP 算法的基本思想是，学习过程由信号的正向传播和误差的反向传播两个过程组成。输入由输入层输入，经隐藏层处理以后，传向输出层。如果输出层的实际输出和期望输出不符，就进入误差的反向传播阶段。误差反向传播是将输出误差以某种形式通过隐藏层向输入层反向传播，并将误差分摊给各层的所有单元，从而获得各层单元的误差信号，这个误差信号就作为修正单元权重的依据，直到输出的误差满足一定条件或者迭代达到一定次数。

如图 1.1，以含一层隐藏层的神经网络为例，其第一层输出为：$\begin{pmatrix} y_1^{(1)} \\ y_2^{(1)} \end{pmatrix} = \begin{pmatrix} \varphi(v_1^{(1)}) \\ \varphi(v_2^{(1)}) \end{pmatrix}$，其中：$\begin{pmatrix} v_1^{(1)} \\ v_2^{(1)} \end{pmatrix} = \begin{pmatrix} \omega_{11}^{(1)} & \omega_{12}^{(1)} \\ \omega_{21}^{(1)} & \omega_{22}^{(1)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \triangleq \boldsymbol{W}_1 \boldsymbol{x}$，第二层输出为：$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} \varphi(v_1) \\ \varphi(v_2) \end{pmatrix}$，其中$\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} \omega_{11}^{(2)} & \omega_{12}^{(2)} \\ \omega_{21}^{(2)} & \omega_{22}^{(2)} \end{pmatrix} \begin{pmatrix} y_1^{(1)} \\ y_2^{(1)} \end{pmatrix} \triangleq \boldsymbol{W}_2 \boldsymbol{y}^{(1)}$。得到输出后，可以计算两个输出节点各自的误差分别为：$\begin{cases} e_1 = d_1 - y_1 \\ \delta_1 = \varphi'(v_1) e_1 \end{cases}$，$\begin{cases} e_2 = d_2 - y_2 \\ \delta_2 = \varphi'(v_2) e_2 \end{cases}$；隐藏层两个节点的误差为：$\begin{cases} e_1^{(1)} = \omega_{11}^{(2)} \delta_1 + \omega_{21}^{(2)} \delta_2 \\ \delta_1^{(1)} = \varphi'(v_1^{(1)}) e_1^{(1)} \end{cases}$，$\begin{cases} e_2^{(1)} = \omega_{12}^{(2)} \delta_1 + \omega_{22}^{(2)} \delta_2 \\ \delta_2^{(1)} = \varphi'(v_2^{(1)}) e_2^{(1)} \end{cases}$，则：$\begin{pmatrix} e_1^{(1)} \\ e_2^{(1)} \end{pmatrix} = \begin{pmatrix} \omega_{11}^{(1)} & \omega_{21}^{(1)} \\ \omega_{12}^{(1)} & \omega_{22}^{(1)} \end{pmatrix} \begin{pmatrix} \delta_1 \\ \delta_2 \end{pmatrix} = \boldsymbol{W}_2^T \begin{pmatrix} \delta_1 \\ \delta_2 \end{pmatrix}$，故误差调整的方法为：$\omega_{ij}^{(2)} = \omega_{ij}^{(2)} + \alpha \delta_i y_j^{(1)}$，$\omega_{ij}^{(1)} = \omega_{ij}^{(1)} + \alpha \delta_i x_j$。

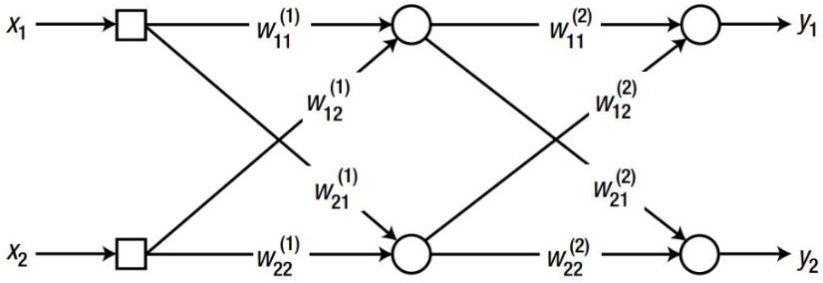


图 1.1

### 二、BP 训练 NN 步骤

根据一中所述原理，BP 训练 NN 的步骤如下：

① 初始化权重；

② 通过训练数据对（输入、准确解），计算 NN 的输出、误差及 Delta 值：$e = d - y$，$\delta = \varphi'(v)e$；

③ 计算下一级的误差及 Delta 值：$e^{(k)} = \boldsymbol{W}^T \delta$，$\delta^{(k)} = \varphi'\left(v^{(k)}\right)e^{(k)}$；

④ 重复第三步，直到所有层都传递完成；

⑤ 根据下列公式调整权重：$\Delta\omega_{ij} = \alpha\delta_i x_j$，$\omega_{ij} = \omega_{ij} + \Delta\omega_{ij}$；

⑥ 重复②-⑤，直到遍历所有训练数据；

⑦ 重复②-⑥，直到网络达到训练要求。

## 三、学习率、损失函数、momentum

动量法可以综合利用前面步骤的方向调整权重更新方向，加入的动量项不仅与本次计算所得的梯度有关，还与上一次权值更新的方向和幅度有关，使得权重修正具有一定惯性。此过程用数学表达式可记为：$\Delta\omega = \alpha\delta x$，$m = \Delta\omega + \beta m^{-}$，$\omega = \omega + m$，$m^{-} = m$，只需在权值更新时不断迭代即可。

## 四、实验 5-2

① 实验题目

- 通过 SGD 训练方法、Sigmoid 激活函数及 BP 规则，对上述神经网络进行训练，并输出训练后的结果；
- 利用实验 5-1 的单层网络及 SGD 方法，用上述数据进行训练并测试，查看并对比两种网络输出结果，分析原因；

② 实验结果与分析

如图，经过 8000 次训练后，算法的输出结果为[0.042586，0.96905485，0.9689139 7，0.03456754]，与真值[0,1,1,0]十分接近，训练效果较好。最终两层网络的权重如图中所示，其中第一层网络权重维度为$4 \times 3$，第二层网络权重维度为$1 \times 4$。

```
Output of backward(without momentum):
 [0.042586   0.96905485 0.96891397 0.03456754]
Weight of backward(without momentum):
 [[ 1.36593771  1.36444828 -1.69412414]
 [ 1.36593771  1.36444828 -1.69412414]
 [ 1.36593771  1.36444828 -1.69412414]
 [ 7.95674575  7.94057607 -3.81953835]]
 [[-7.27278126 -7.27278126 -7.27278126 12.77514377]]
```

如图 4.1，为前向单层网络与后向双层网络输出误差的对比图。可以发现，后向双层神经网络的输出误差经历了两次陡峭下降，经过约 4000 次训练后，误差便已经接近为 0；而前向单层网络的误差在前期迅速下降后便稳定在了 0.25 左右（四个输入样本的输出结果都约为 0.5）。

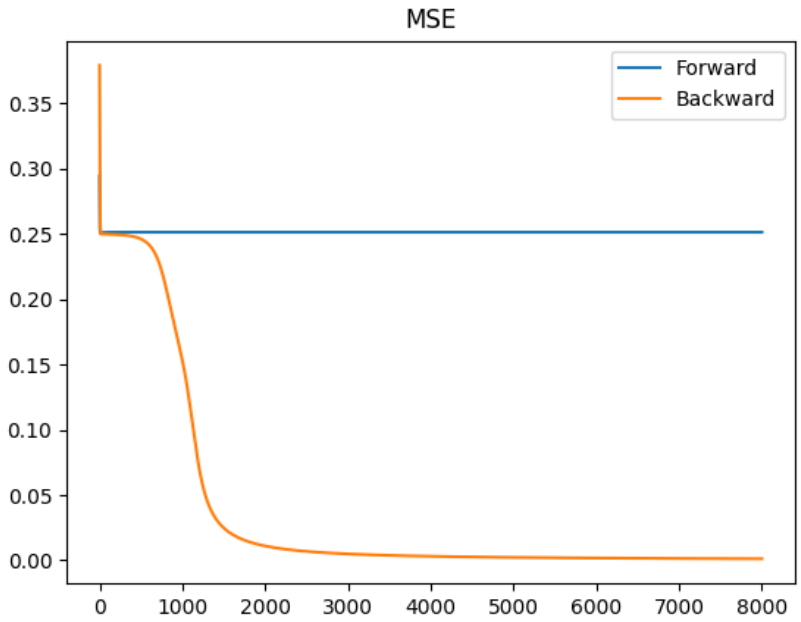


图 4.1 前向单层网络与后向双层网络对比

之所以会出现上述问题，是由于单层神经网络只能解决简单的线性分类问题，但无法解决“异或”，其只能采用多层神经网络进行处理。在输入的样本中，第三个数据都为 1，故前两位数据决定了输出，其真值表如下：

| $x_1$ | $x_2$ | $y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

显然，$y = x_1 \oplus x_2$，满足异或关系，所以单层神经网络无法输出正确结果。当采用两层神经网络后，方能正确输出。

## 五、实验 5-3

① 实验题目

将神经网络更新规则改为 momentum，对上述神经网络进行训练，输出训练后的结果，比较不同$\beta$值 momentum 训练方法及不用 momentum 训练方法时，误差(真实结果与输出的 MSE)随 epoch 的变化趋势，并可视化结果。

② 实验结果与分析

如图 5.1 至 5.3，分别为$\beta$值等于 0.1,0.5,0.9 时 momentum 训练方法的输出误差与不使用 momentum 训练方法时误差的比较图。

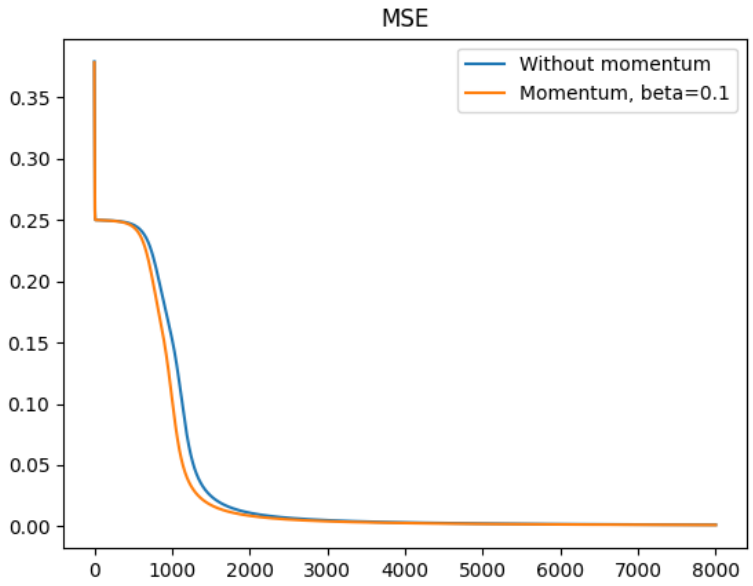


图 5.1 $\beta = 0.1$

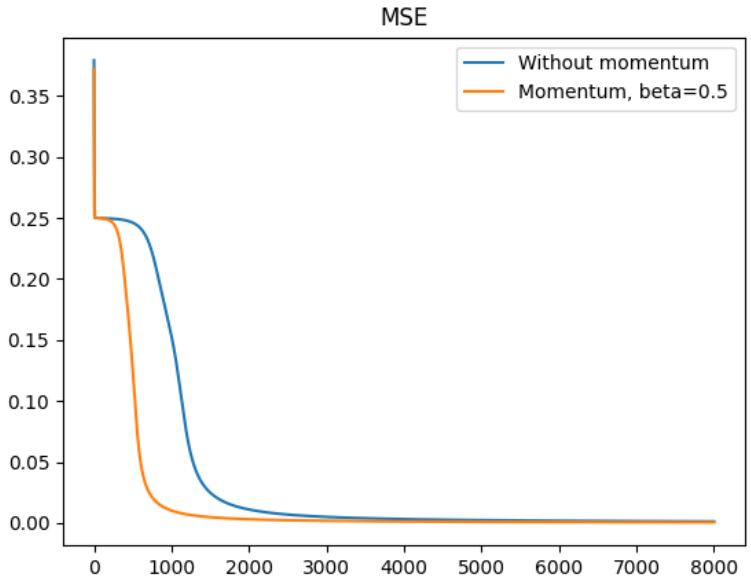


图 5.2 $\beta = 0.5$

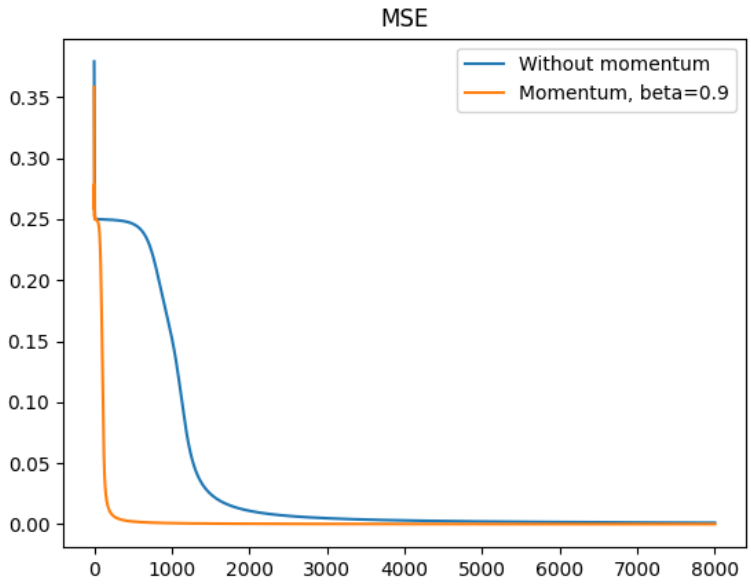


图 5.3 $\beta = 0.9$

可以发现，随着$\beta$值的增大$(0 < \beta < 1)$，神经网络输出摆脱局部最小值的速度不断加快。这是由于 momentum 方法考虑了历史时刻的梯度方向，当当前的权重更新方向和历史相同时，此次的下降幅度将会加大，从而加速收敛；否则将使当前时刻的梯度方向减弱。$\beta$值是对历史权重更新方向参考度的度量，当其越大时，表明历史权重更新方向对当前时刻的权重更新影响越大，进而加快网络的收敛速度。

**六、附录**：实验 Python 代码——*nn.py*

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import make_interp_spline


def sigmoid(x):
    """
    sigmoid 函数
    Param:
        x: 输入
    Return:
        y: 对应的 sigmoid 函数值
    """
    y = 1./(1+np.exp(-x))
    return y
```

```python
def mse(x, y):
    """
    计算均方误差
    Param:
        x, y: 输入
    Return:
        x 与 y 的均方误差向量
    """
    MSEs = []
    for i in range(x.shape[1]):
        xi = x[:,i]                    # 列向量
        MSE = 0                        # 初始化
        for j in range(len(y)):
            square = (xi[j]-y[j])**2  # 平方和
            MSE += square
        MSE /= len(y)                  # 取平均
        MSEs.append(MSE)
    return np.array(MSEs)


def create_network(net_type, inputs, gts, epochs, beta=0):
    """
    构造神经网络，计算权重
    Params:
        net_type: 网络类型，前向或后向
        inputs: 数据输入
        gts: 正确值
        epochs: 训练轮数
        beta: momentum 参数
    Return:
        weight: 训练好的权重
        MSE: 训练结果与 ground truth 的均方误差
    """
    outputs_fp = []                    # 保存单次训练的结果
    outputs_bp = []
    if net_type == 'forward':          # 前向传输，SGD 方法，单层网络
        weight = np.array([[0.1,0.5,0.8]])  # 初始化权重
        alpha = 0.9                    # 学习率
        for i in range(epochs):
            for j in range(inputs.shape[0]):     # 每轮需要调整权重的次数
                input = inputs[j].reshape(1,3)            # 依次选取
                output = sigmoid(np.dot(weight, input.T)) # 计算输出，1*1
                e = gts[j] - output                       # 输出误差，1*1
                delta = alpha*output*(1-output)*e*input # 误差更新值，1*3
                weight = weight + delta                   # 更新权重
```

```
            # 前向传输的结果
            output_fp = sigmoid(np.dot(weight, inputs.T))
            outputs_fp.append(output_fp.reshape(4,))    # 保存结果
        outputs_fp = np.array(outputs_fp)
        MSE_fp = mse(outputs_fp.T, gts)                 # 均方误差
        # 输出训练后的结果和权重
        print('Output of forward:\n', output_fp.reshape(4,))
        print('Weight of forward:\n', weight, '\n')
        return weight, MSE_fp
    elif net_type == 'backward':                # 后向传输
        weight_1 = np.array([[0.5,0.5,1],
                             [0.5,0.5,1],
                             [0.5,0.5,1],
                             [0.5,0.5,1]])      # 第一层权重
        weight_2 = np.array([[0.5,0.5,0.5,1]])  # 第二层权重
        alpha = 0.45                            # 学习率
        m_2_ = 0                                # momentum 参数
        m_1_ = 0
        for i in range(epochs):
            for j in range(inputs.shape[0]):
                input = inputs[j].reshape(1,3)   # 依次选取，1*3
                # 第一层输出，4*1
                output_1=sigmoid(np.dot(weight_1,input.T)).reshape(4,1)
                # 第二层输出，1*1
                output_2 = sigmoid(np.dot(weight_2, output_1))
                e_2 = gts[j] - output_2             # 第二层输出误差，1*1
                delta_2 = output_2*(1-output_2)*e_2  # 第二层 delta，1*1
                # 后向传输，第一层误差，4*1
                e_1 = np.dot(weight_2.T, delta_2)
                delta_1 = np.zeros((4,1))           # 第一层 delta，4*1
                for k in range(4):
                    delta_1[k] = output_1[k]*(1-output_1[k])*e_1[k]
                delta_w_2 = alpha*delta_2*output_1.T  # 第二层更新值，1*4
                # 第一层更新值，4*3
                delta_w_1 = alpha*np.dot(delta_1, input)
                # 根据 momentum 调整方向，当不使用 momentum 时，beta 默认为 0
                m_2 = delta_w_2 + beta*m_2_   # momentum，1*4
                m_1 = delta_w_1 + beta*m_1_   # momentum，4*3
                weight_2 = weight_2 + m_2     # 第二层权重，1*4
                weight_1 = weight_1 + m_1     # 第一层权重，4*3
                m_2_ = m_2                   # 保存第二层的 momentum 更新值
                m_1_ = m_1                   # 保存第一层的 momentum 更新值
            # 后向传输第一层的结果，4*1
            output_bp_1 = sigmoid(np.dot(weight_1, inputs.T))
```

```
            # 后向传输的结果，1*1
            output_bp = sigmoid(np.dot(weight_2, output_bp_1))
            outputs_bp.append(output_bp.reshape(4,))
        outputs_bp = np.array(outputs_bp)
        MSE_bp = mse(outputs_bp.T, gts)
        if beta == 0:
            print('Output of backward(without momentum):\n',
                  output_bp.reshape(4,))
            print('Weight of backward(without momentum):\n', weight_1,
                  '\n', weight_2, '\n')
        else:
            print('Output of backward(momentum):\n',
                  output_bp.reshape(4,))
            print('Weight of backward(momentum):\n', weight_1, '\n',
                  weight_2, '\n')
        return weight_1, weight_2, MSE_bp


def main():
    dataset = np.array([[0,0,1,0],[0,1,1,1],[1,0,1,1],[1,1,1,0]])
    inputs = dataset[:,0:3]       # 输入
    gts = dataset[:,3:4]          # ground truth
    epochs = 8000                 # 训练轮数
    beta = 0.1                    # momentum 参数
    # beta = 0.5
    # beta = 0.9

    # 前向传输的权重和均方误差
    weight_fp, MSE_fp = create_network('forward', inputs, gts, epochs)

    # 后向传输的权重和均方误差(不使用 momentum)
    weight_bp_1, weight_bp_2, MSE_bp = create_network('backward',
                                       inputs, gts, epochs)

    # 后向传输的权重和均方误差(使用 momentum)
    weight_bp_m_1, weight_bp_m_2, MSE_bp_m = create_network('backward',
                                             inputs, gts, epochs, beta)

    epoch = np.linspace(0, epochs, epochs)
    # 绘制平滑曲线
    model_fp = make_interp_spline(epoch, MSE_fp.reshape(epochs,))
    model_bp = make_interp_spline(epoch, MSE_bp.reshape(epochs,))
    model_bp_m = make_interp_spline(epoch, MSE_bp_m.reshape(epochs,))
    mse_fp = model_fp(epoch)
    mse_bp = model_bp(epoch)
```

```
    mse_bp_m = model_bp_m(epoch)

    plt.figure(1)                       # 实验 5-2，可视化
    l1, = plt.plot(epoch, mse_fp)
    l2, = plt.plot(epoch, mse_bp)
    plt.legend(handles=[l1,l2],labels=["Forward","Backward"],loc='best')
    plt.title('MSE')
    plt.draw()

    plt.figure(2)                       # 实验 5-3，可视化
    l2, = plt.plot(epoch, mse_bp)
    l3, = plt.plot(epoch, mse_bp_m)
    plt.legend(handles=[l2,l3],labels=["Without momentum",
               "Momentum, beta=%3f" % beta],loc='best')
    plt.title('MSE')
    plt.show()


if __name__ == '__main__':
    main()
```