Summary

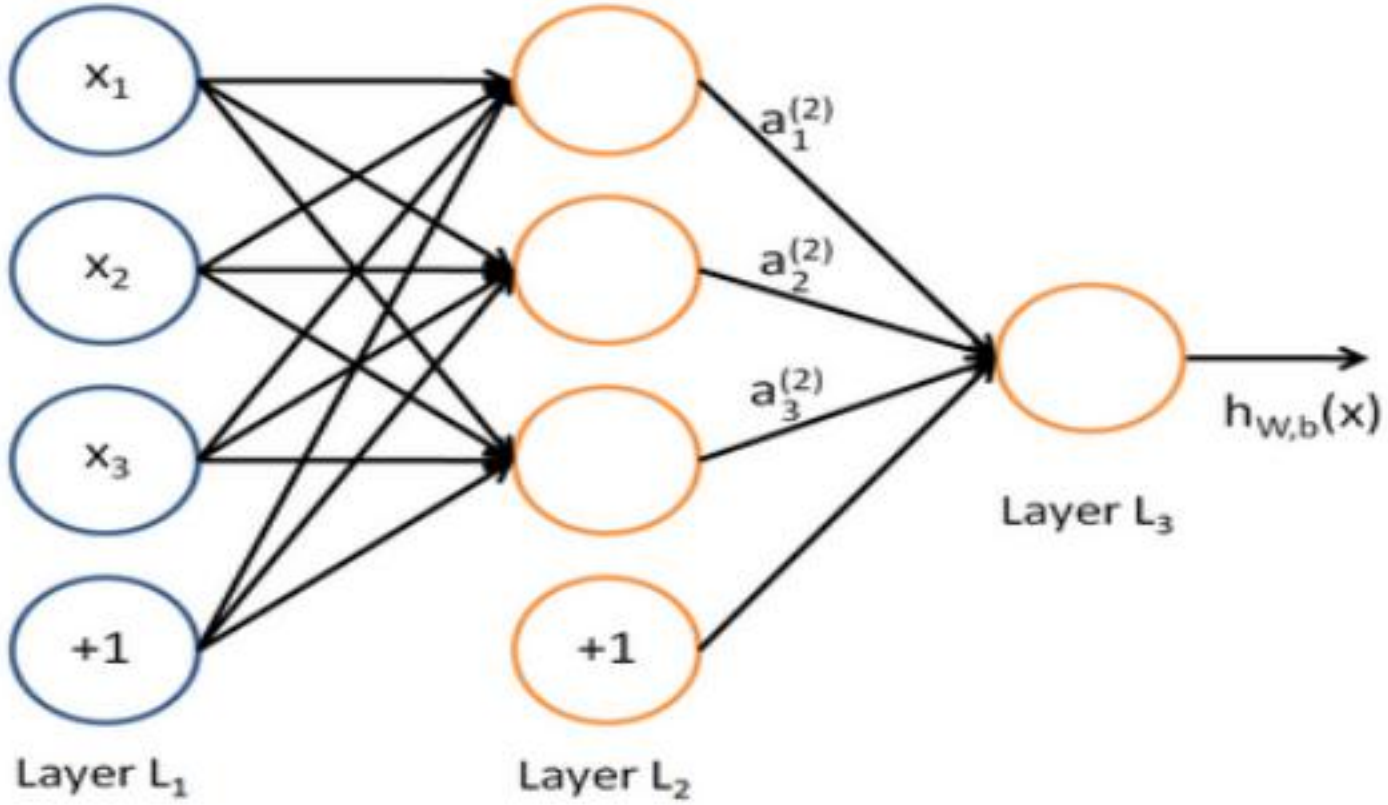
李娜

Deep Learning group

October 28.2016

CONTENTS

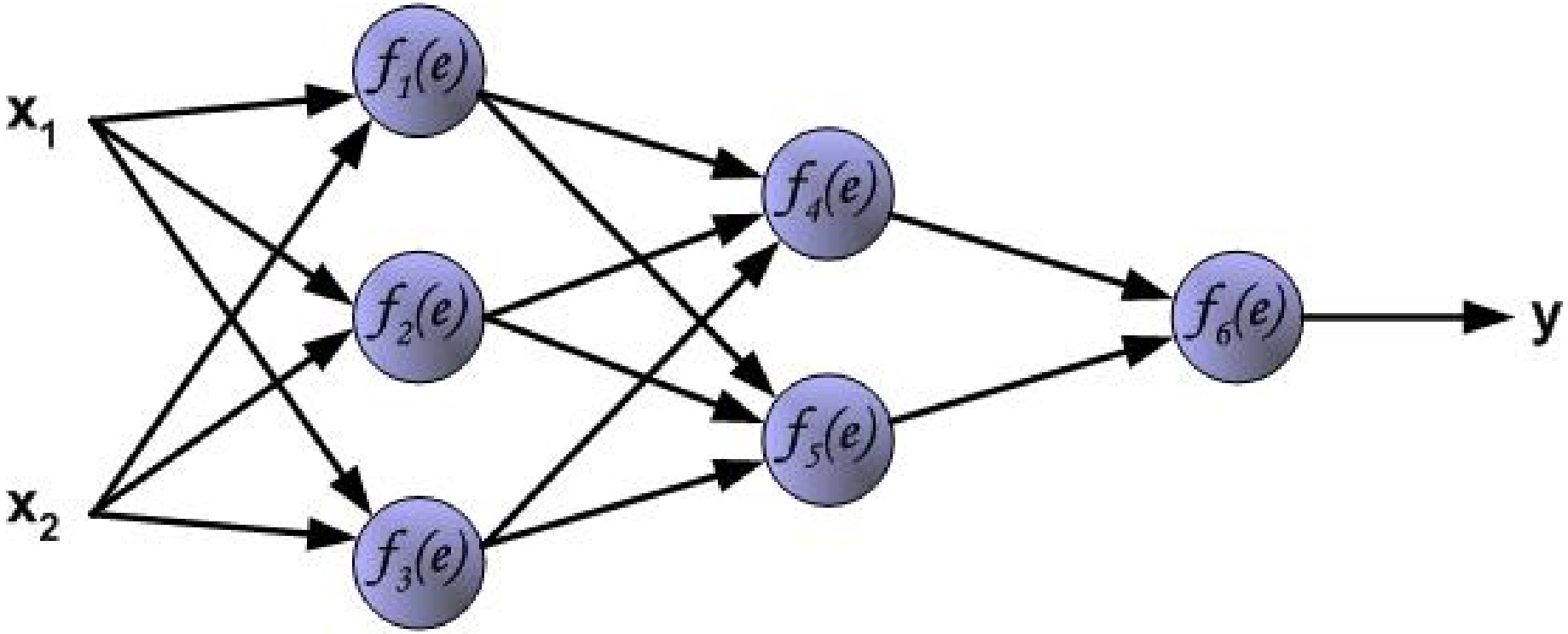
I. The knowledge of MLP



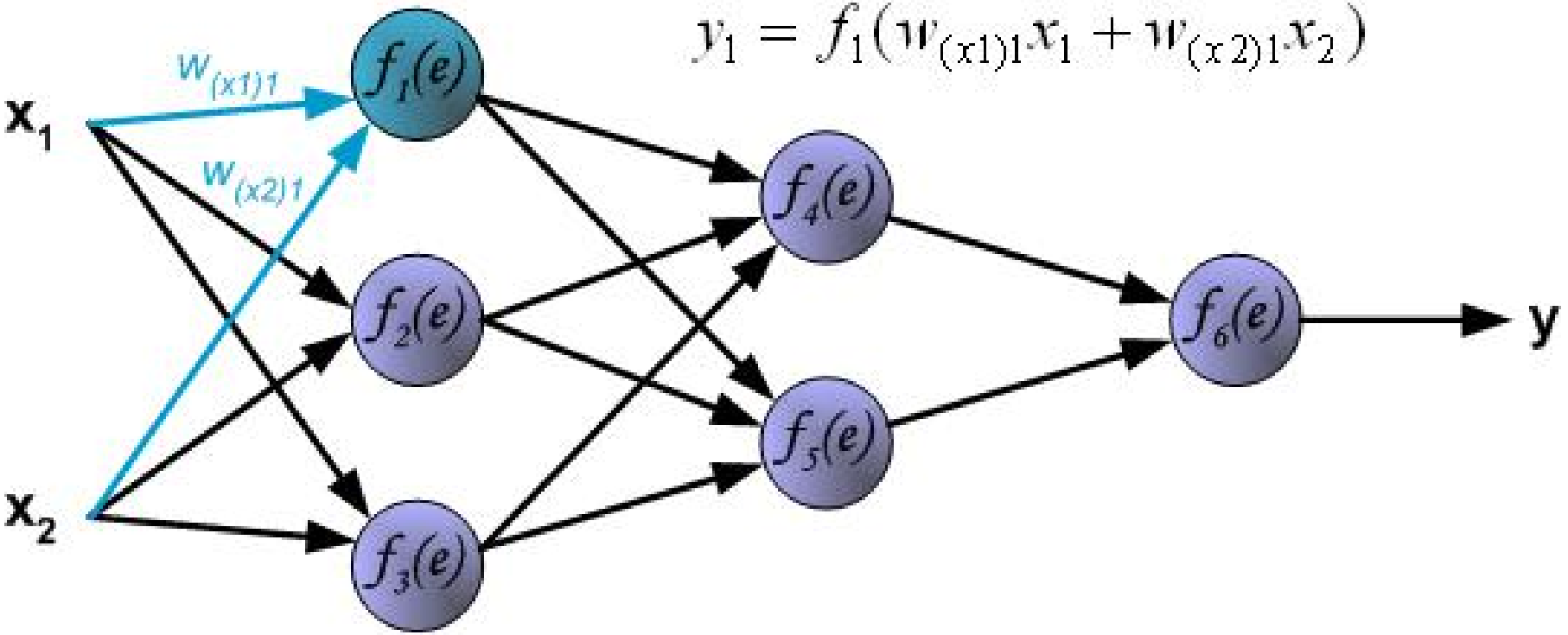
I. The knowledge of MLP

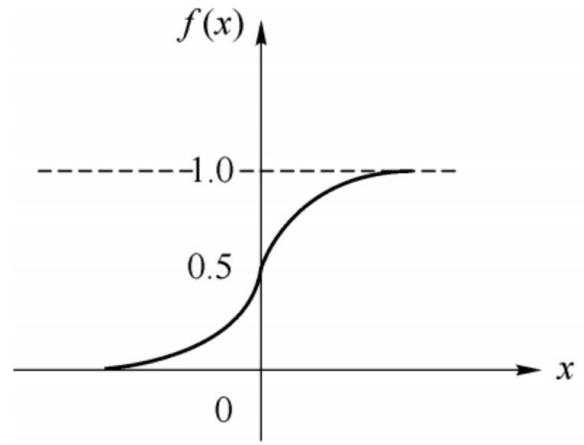
$$
\begin{aligned}
a_1^{(2)} &= f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \\
a_2^{(2)} &= f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \\
a_3^{(2)} &= f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}) \\
h_{W,b}(x) &= a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})
\end{aligned}
$$

I. The knowledge of MLP------feedforward



I. The knowledge of MLP------feedforwa



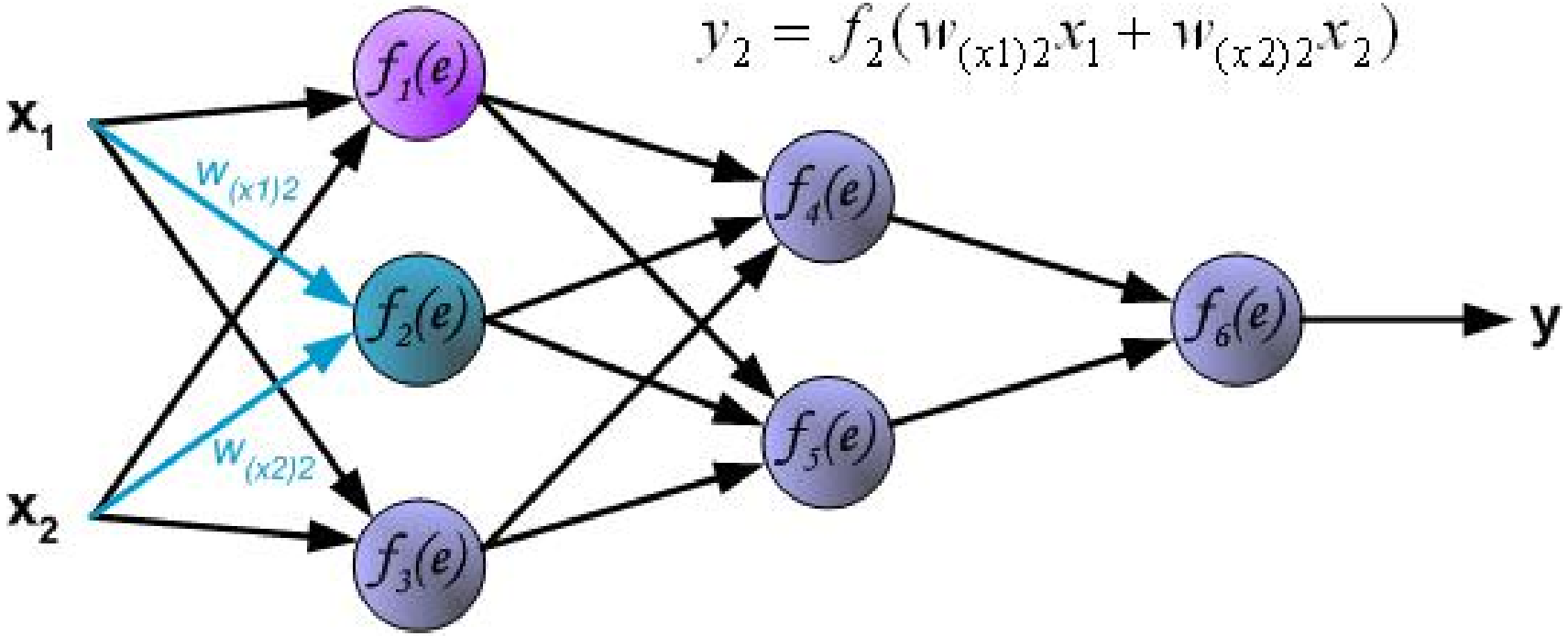


Sigmoid函数

$$f(\alpha) = \frac{1}{1+e^{-\alpha}}$$

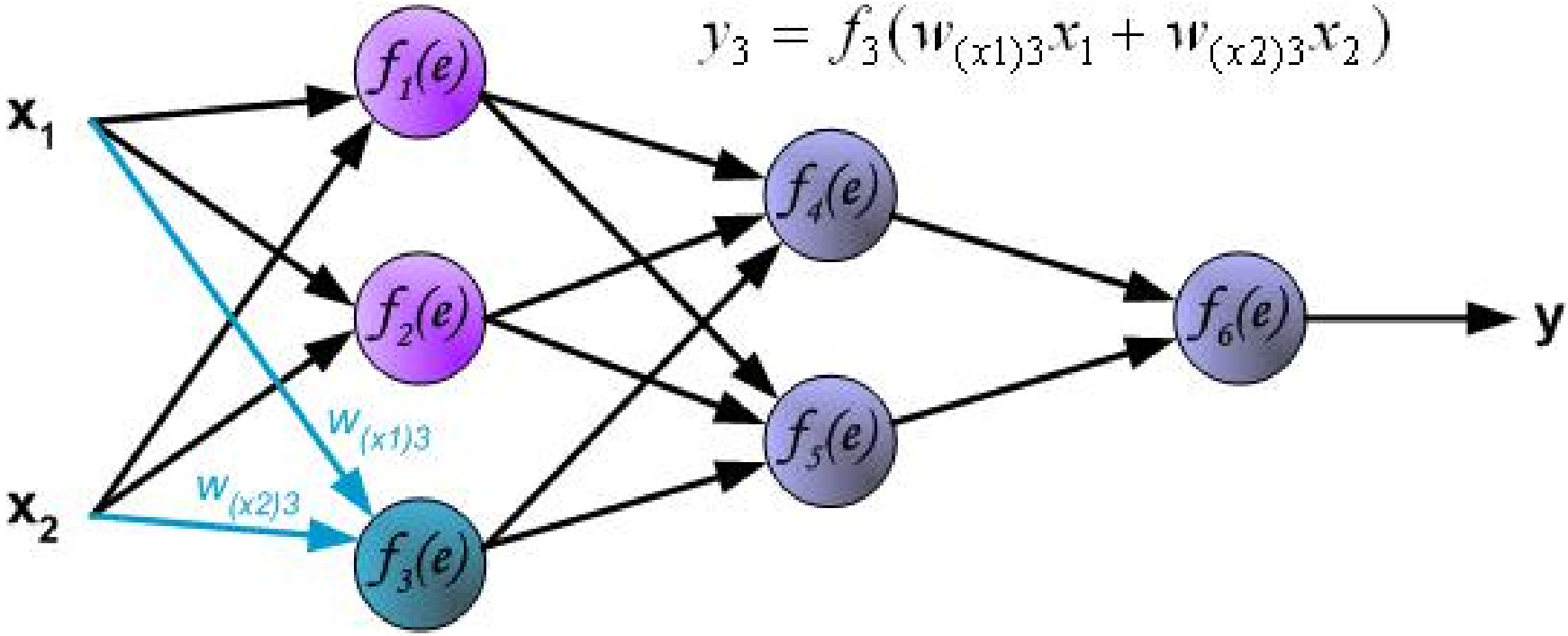
I. The knowledge of MLP------feedforward

$$y_2 = f_2(w_{(x1)2}x_1 + w_{(x2)2}x_2)$$

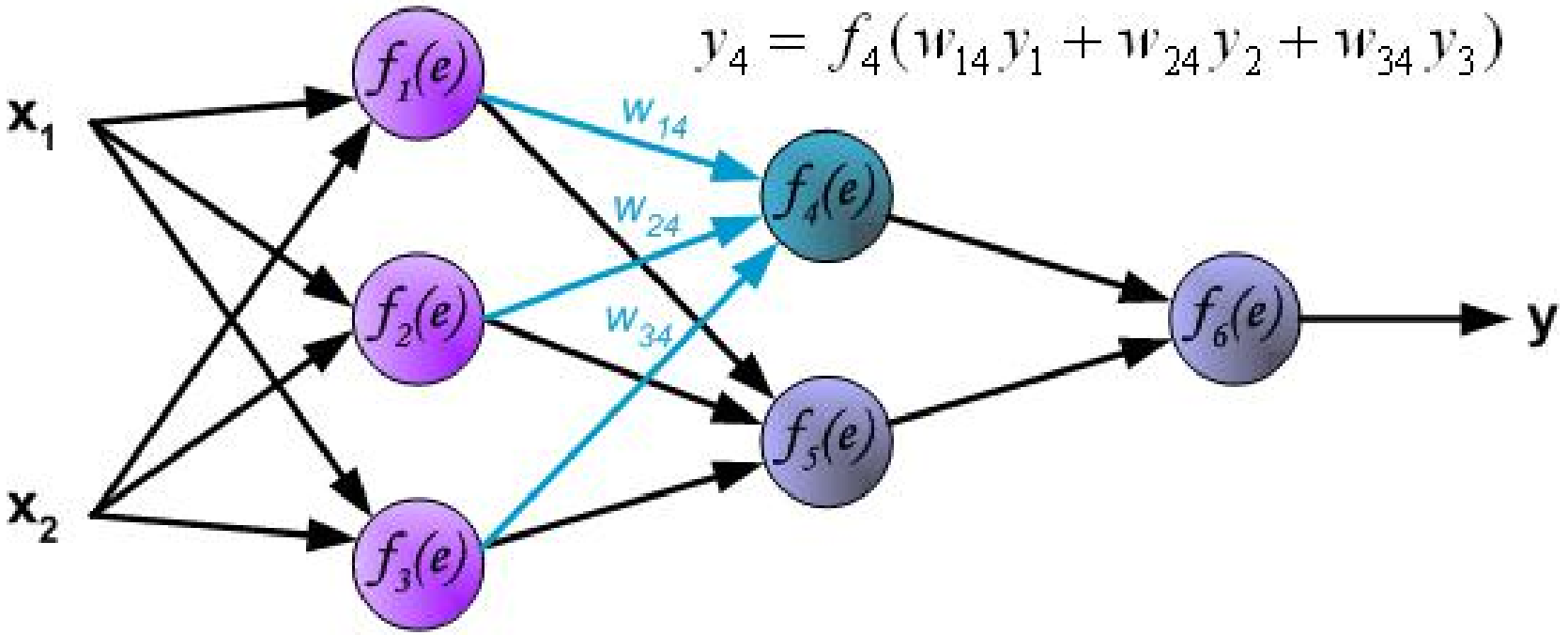


I. The knowledge of MLP------feedforward

$$y_3 = f_3(w_{(x1)3}x_1 + w_{(x2)3}x_2)$$

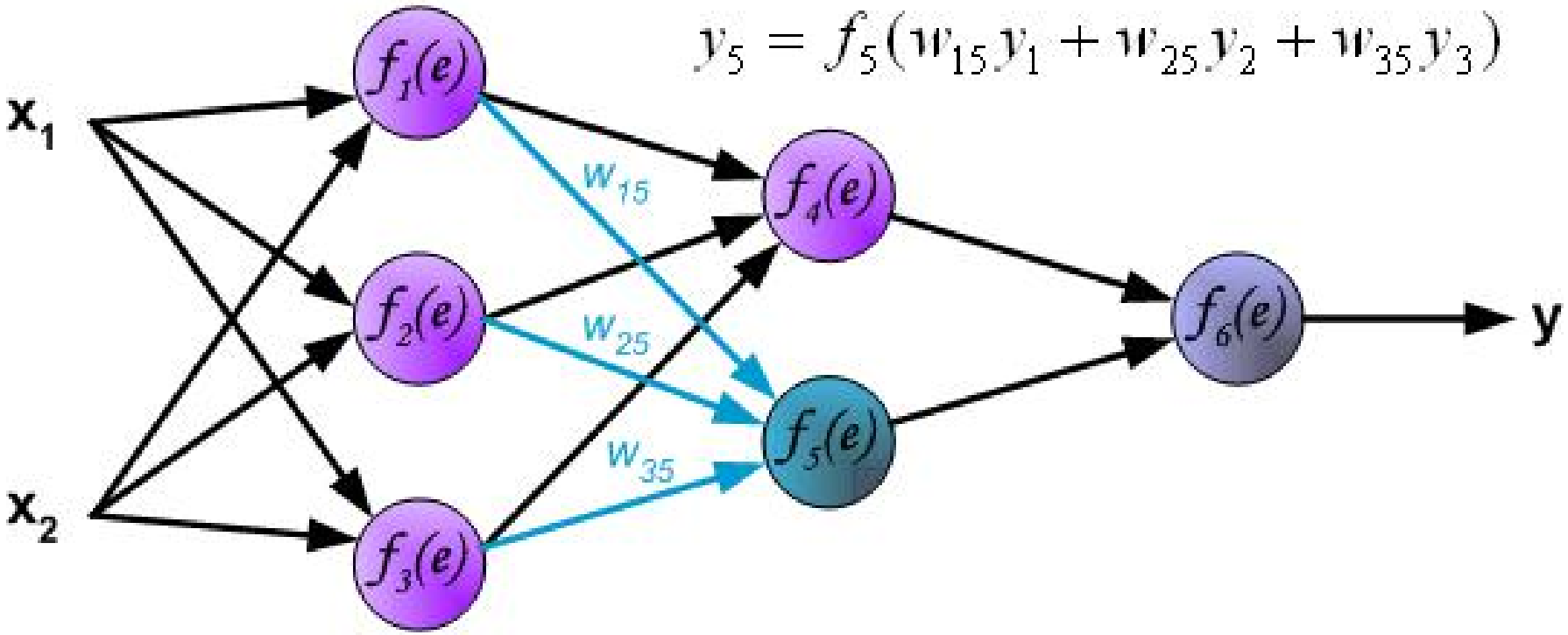


I. The knowledge of MLP------feedforward

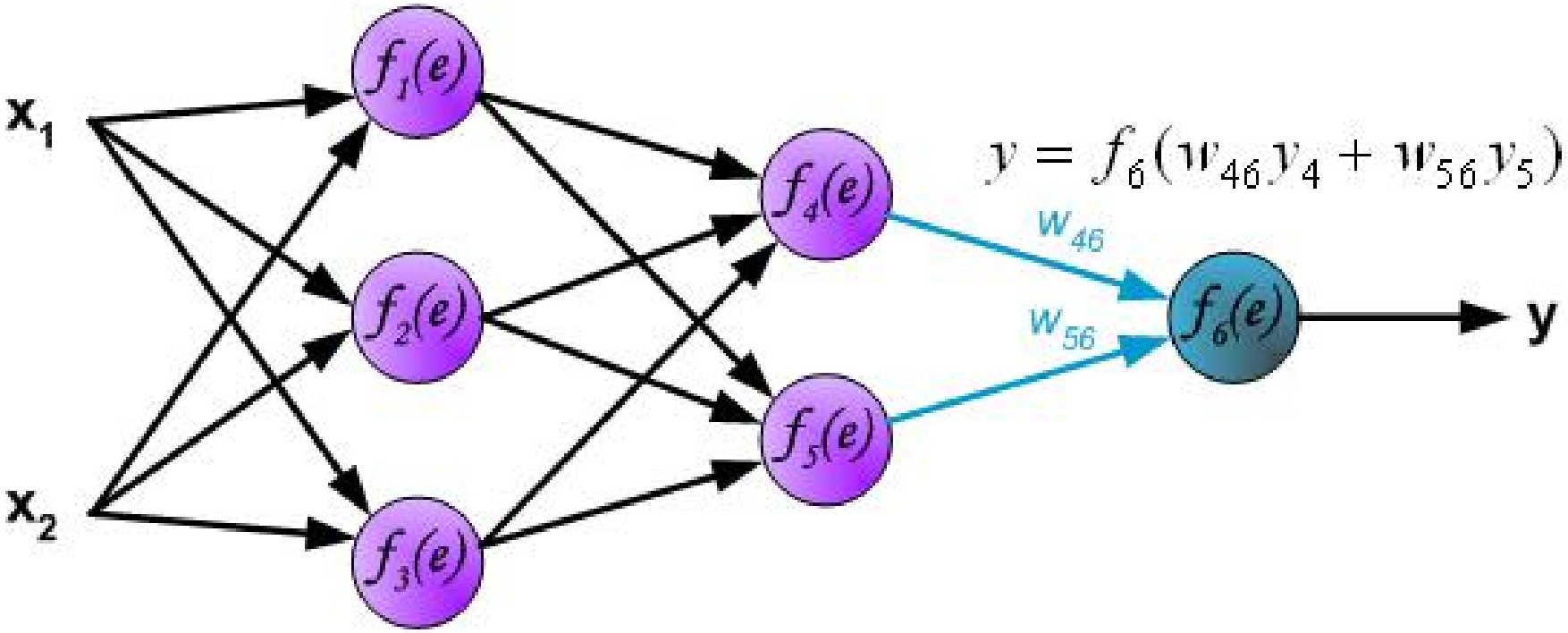


I. The knowledge of MLP------feedforward

$$y_5 = f_5(w_{15}y_1 + w_{25}y_2 + w_{35}y_3)$$

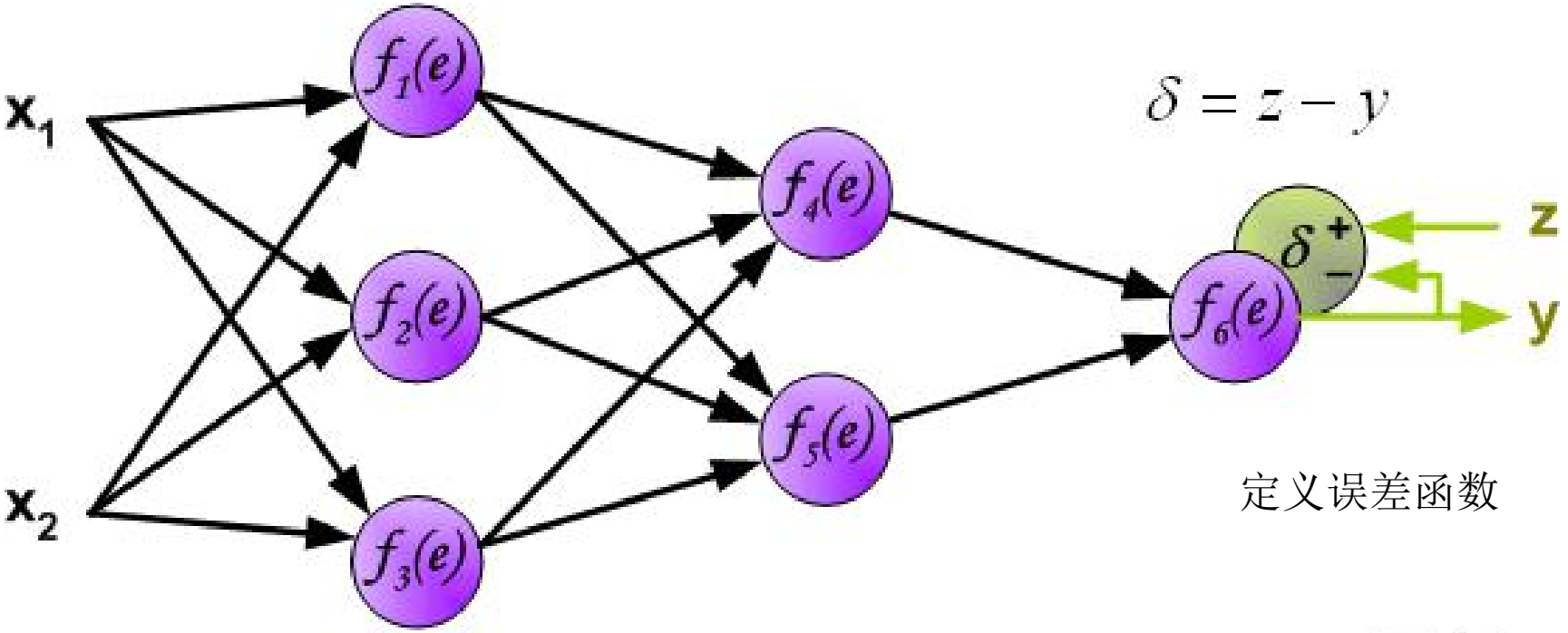


I. The knowledge of MLP------feedforward



I. The knowledge of MLP------feedback

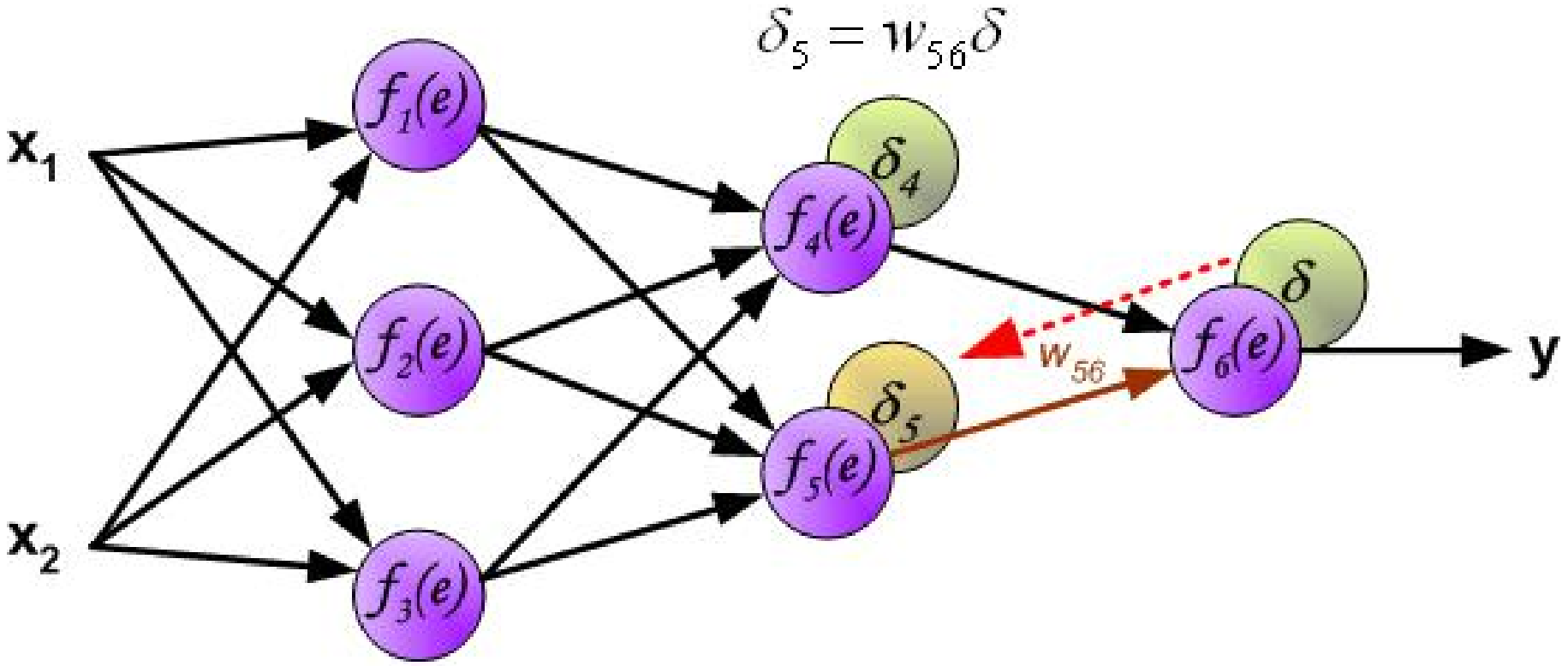
残差表明了该节点对最终输出值的残差产生了多少影响



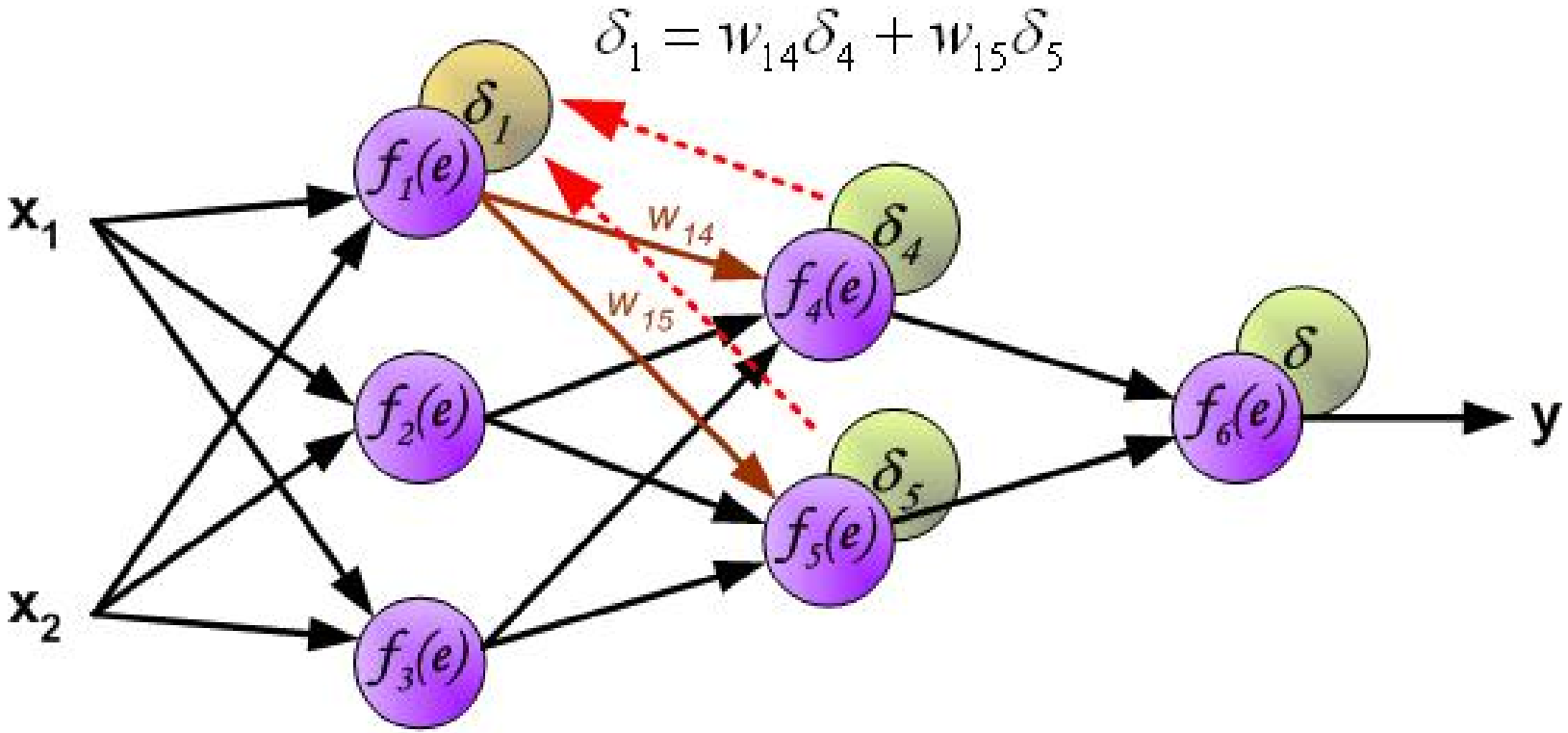
定义误差函数

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

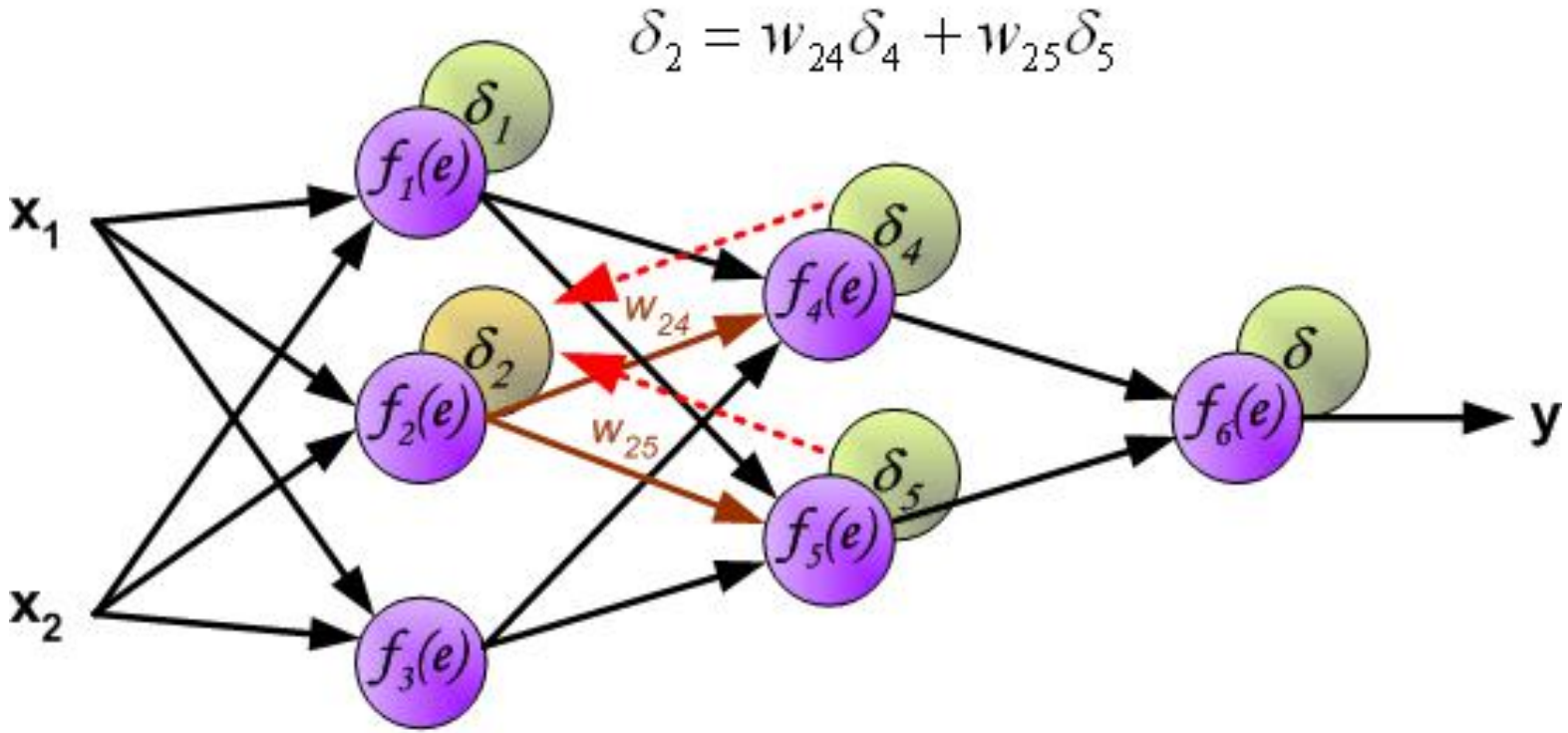
I. The knowledge of MLP------feedback



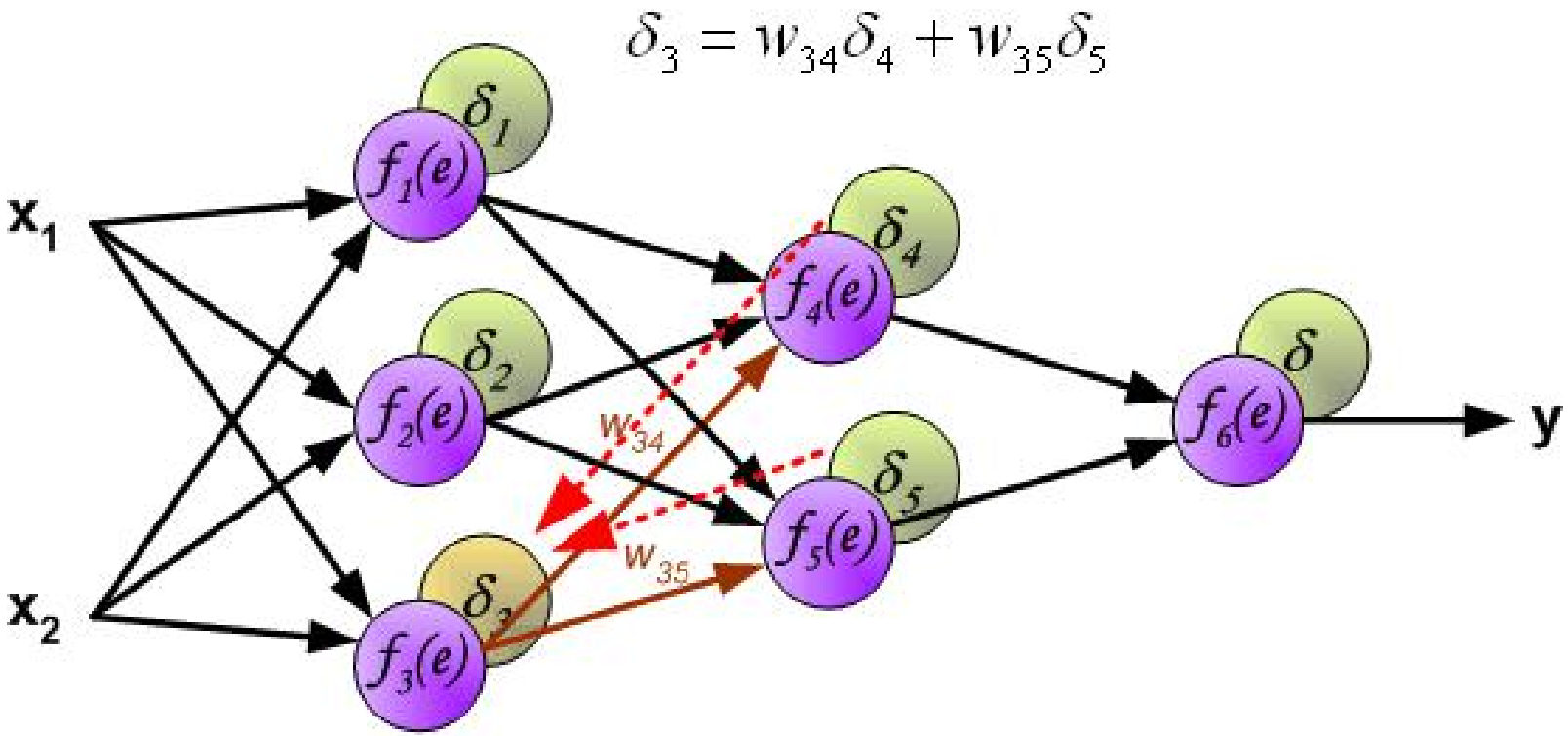
I. The knowledge of MLP------feedback



I. The knowledge of MLP------feedback



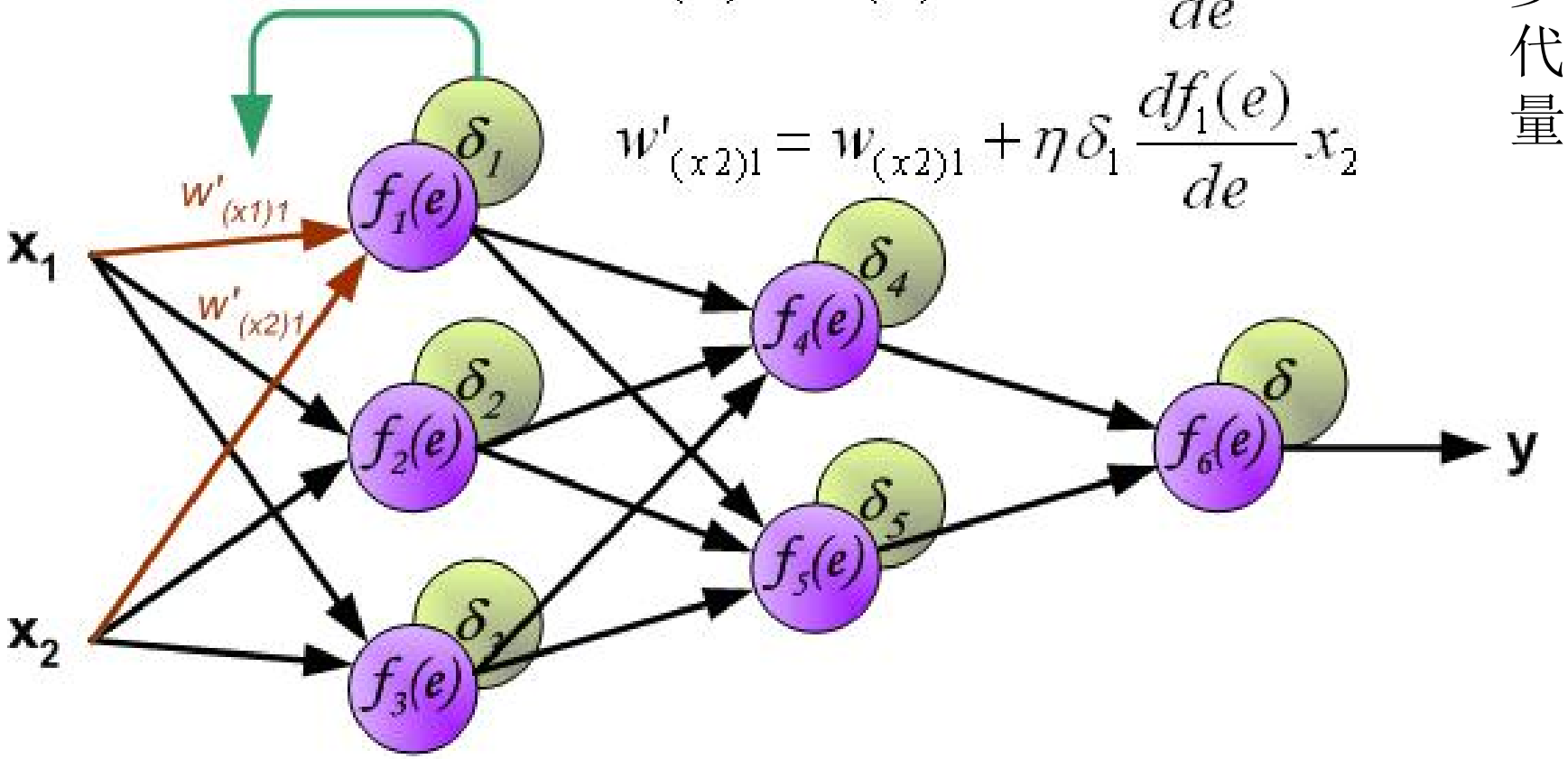
I. The knowledge of MLP------feedback



I. The knowledge of MLP------feedback

$$w'_{(x1)1} = w_{(x1)1} + \eta\delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta\delta_1 \frac{df_1(e)}{de} x_2$$

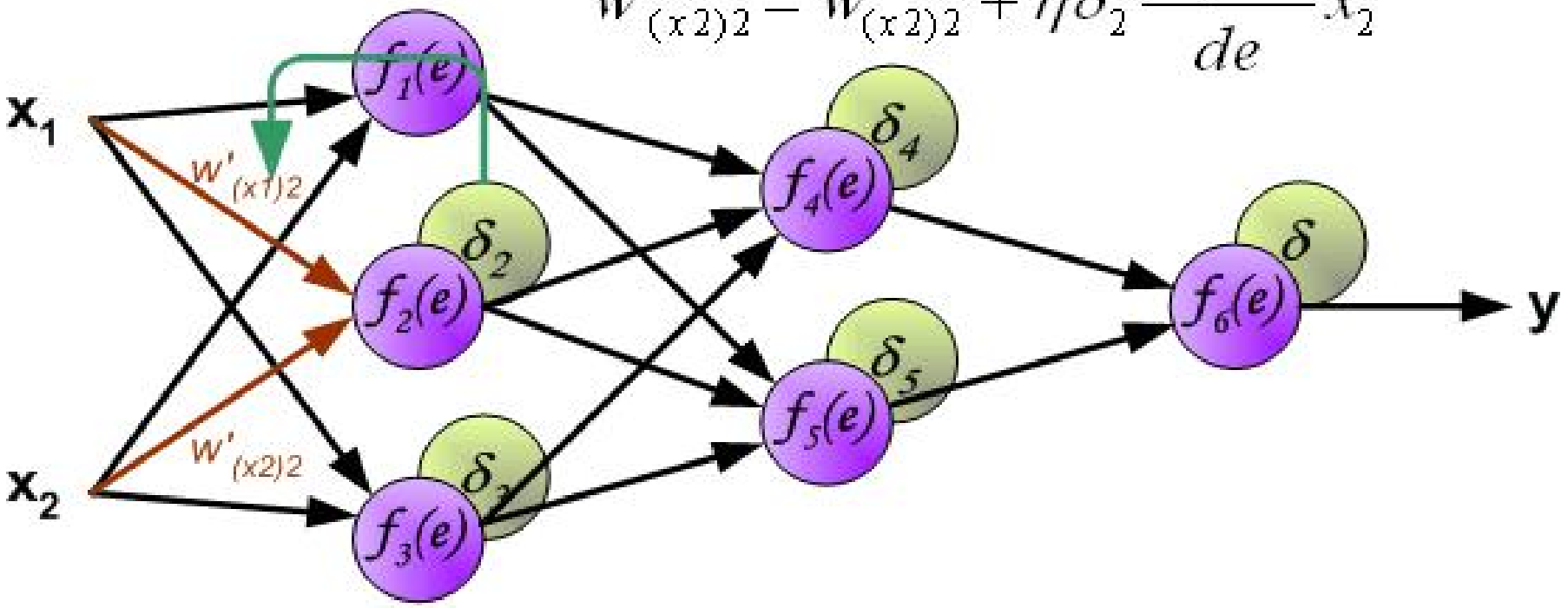


学习率η:某方向的移动步长，使得代价函数尽量取小

I. The knowledge of MLP------feedback

$$w'_{(x1)2} = w_{(x1)2} + \eta \delta_2 \frac{df_2(e)}{de} x_1$$

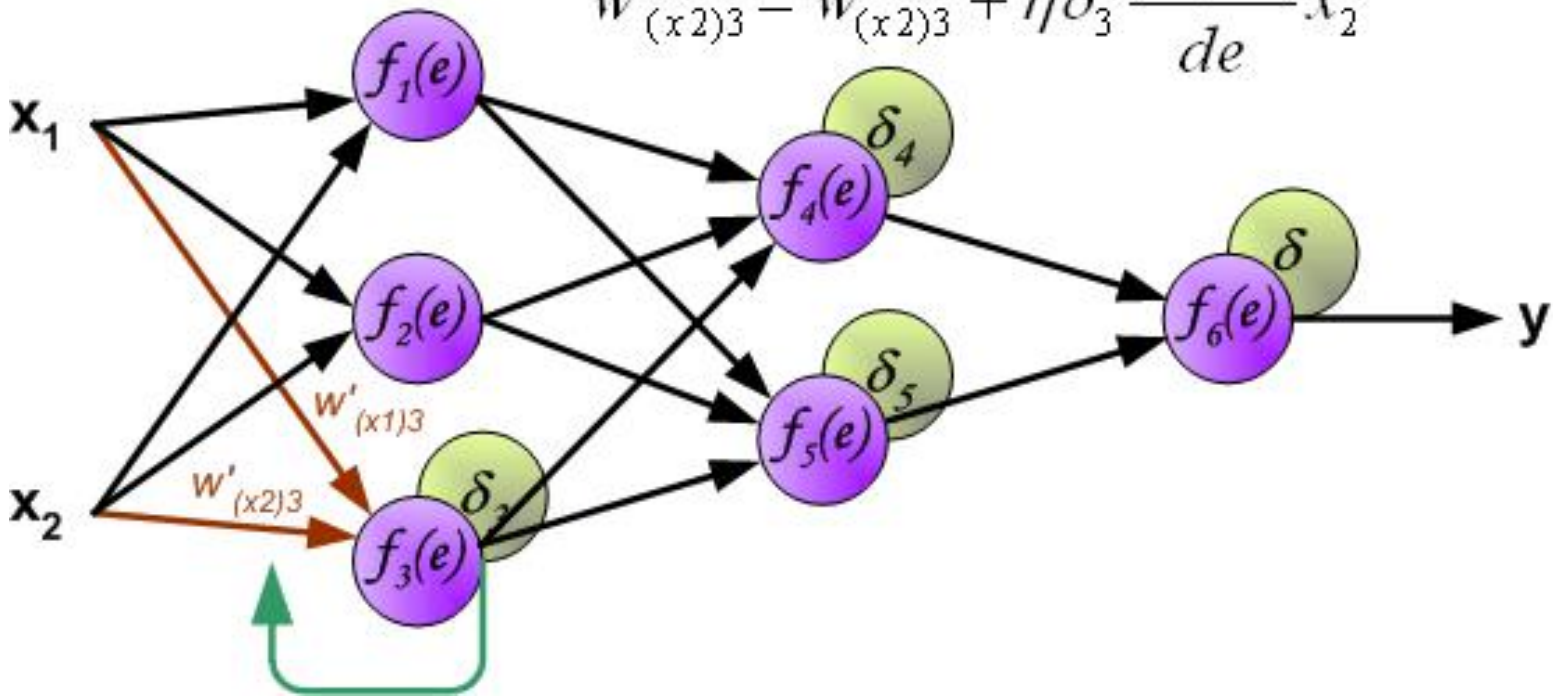
$$w'_{(x2)2} = w_{(x2)2} + \eta \delta_2 \frac{df_2(e)}{de} x_2$$



I. The knowledge of MLP

$$w'_{(x1)3} = w_{(x1)3} + \eta\delta_3 \frac{df_3(e)}{de} x_1$$

$$w'_{(x2)3} = w_{(x2)3} + \eta\delta_3 \frac{df_3(e)}{de} x_2$$

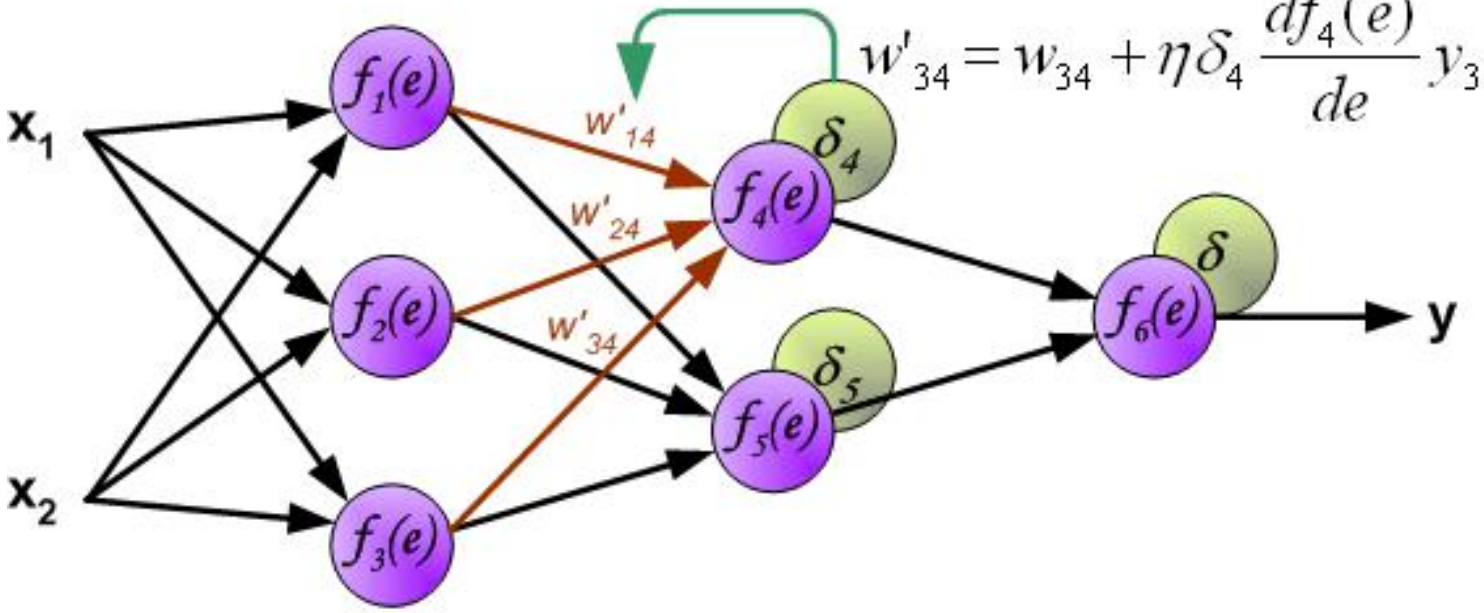


I. The knowledge of MLP------feedback

$$w'_{14} = w_{14} + \eta\delta_4 \frac{df_4(e)}{de} y_1$$

$$w'_{24} = w_{24} + \eta\delta_4 \frac{df_4(e)}{de} y_2$$

$$w'_{34} = w_{34} + \eta\delta_4 \frac{df_4(e)}{de} y_3$$

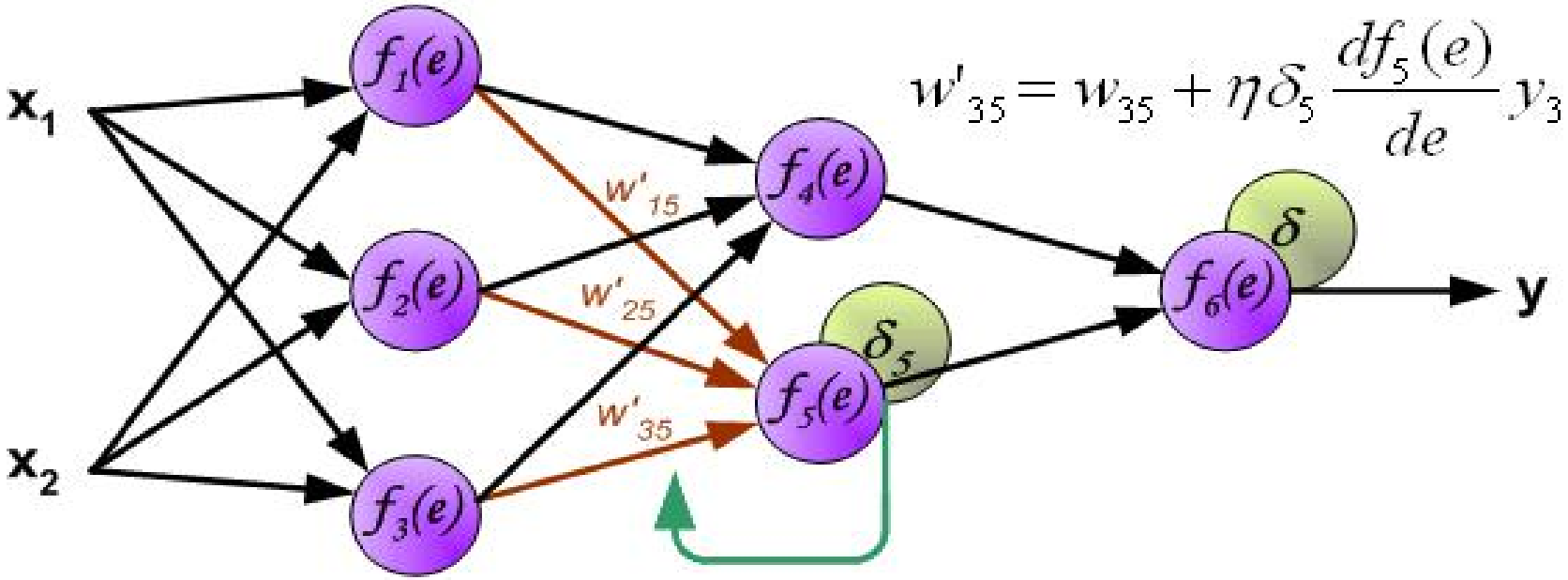


I. The knowledge of MLP------feedback

$$w'_{15} = w_{15} + \eta\delta_5 \frac{df_5(e)}{de} y_1$$

$$w'_{25} = w_{25} + \eta\delta_5 \frac{df_5(e)}{de} y_2$$

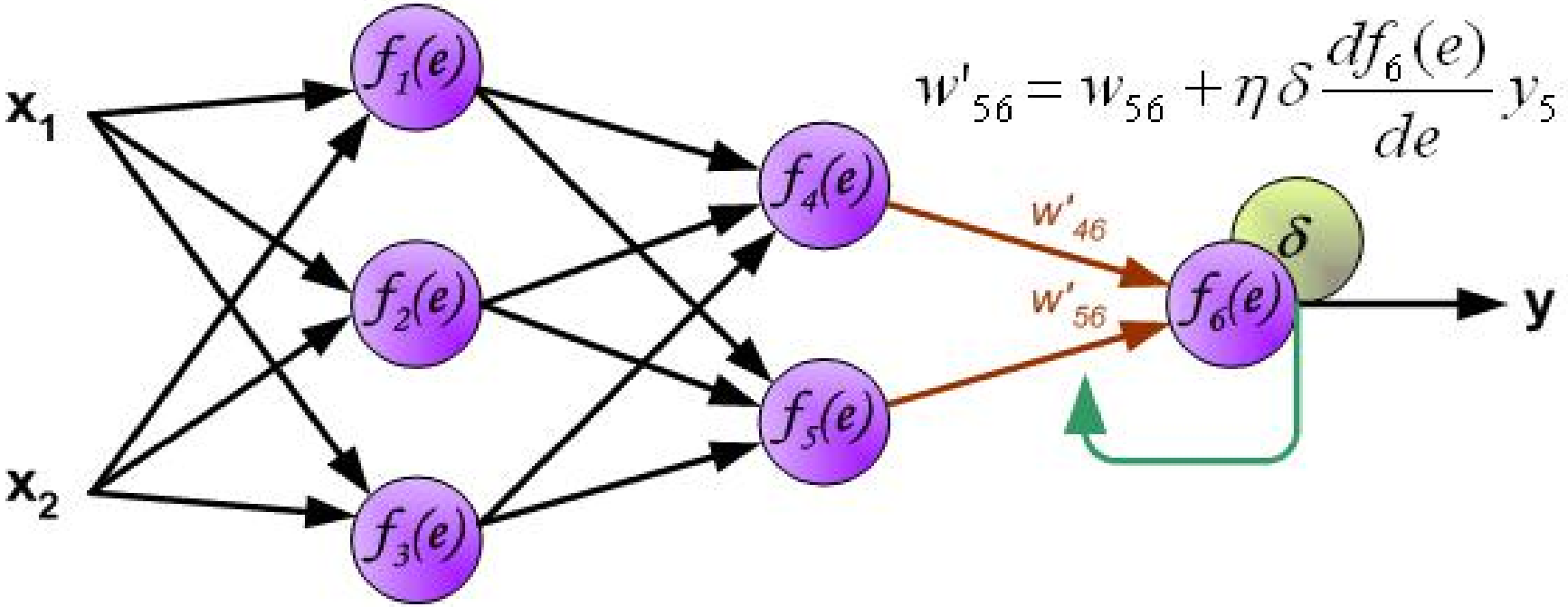
$$w'_{35} = w_{35} + \eta\delta_5 \frac{df_5(e)}{de} y_3$$



I. The knowledge of MLP------feedback

$$w'_{46} = w_{46} + \eta\,\delta\,\frac{df_6(e)}{de}\,y_4$$

$$w'_{56} = w_{56} + \eta\,\delta\,\frac{df_6(e)}{de}\,y_5$$



Ⅱ. The code of MLP

- 算法基本流程就是：
- 1、初始化网络权值和神经元的偏置结点值（随机初始化）
- 2、前向传播：按照公式一层一层的计算隐层神经元和输出层神经元的输入和输出。
- 3、反向传播：根据公式修正权值和偏置值

Ⅱ. The code of MLP------detail

- 1.规律的定义变量：节点值、参数（权值、偏置）、函数、参数梯度分别放在不同的类中；

```
using namespace std;
#define e 2.718281
#define dataNumber 300
#define dataDimension 30
#define inputLayerNodeNumber 30
#define firstHiddenLayerNodeNumber 40
#define secondHiddenLayerNodeNumber 40
#define outputLayerNodeNumber 2
#define learningRate 0.5
#define epochNum 500
#define testDataNumber 369
class inputLayer{
public:
    float inputData[dataNumber][dataDimension];
    float label[dataNumber];
    float inputTestData[dataNumber][dataDimension];
};
class nodeValue{
    public:
    float firstHiddenLayerNodeValue[dataNumber][firstHiddenLayerNodeNumber];
    float secondHiddenLayerNodeValue[dataNumber][secondHiddenLayerNodeNumber];
    float outputLayerNodeValue[dataNumber][outputLayerNodeNumber];
    float firstHiddenLayerNodeRoundValue[firstHiddenLayerNodeNumber];
    float secondHiddenLayerNodeRoundValue[secondHiddenLayerNodeNumber];
    float tempResultFirst[dataNumber][firstHiddenLayerNodeNumber];
    float tempResultSecond[dataNumber][secondHiddenLayerNodeNumber];
    float tempResultOut[dataNumber][outputLayerNodeNumber];
```

Ⅱ. The code of MLP------detail

```
};
class parameter{
public:
    float weightInputHiddenLayer[inputLayerNodeNumber][firstHiddenLayerNodeNumber];
    float weightHiddenHiddenLayer[firstHiddenLayerNodeNumber][secondHiddenLayerNodeNumber];
    float weightHiddenOutputLayer[secondHiddenLayerNodeNumber][outputLayerNodeNumber];
    float biasFirstHiddenLayer[firstHiddenLayerNodeNumber];
    float biasSecondHiddenLayer[secondHiddenLayerNodeNumber];
    float biasOutputLayer[outputLayerNodeNumber];

};
class function{
public:
    float randomGaussian(float miu,float sigma);
    float sigmoid(float x);
    float dsigmoid(float x);
    float error();
};
class roundparameter{
public:
    float roundweightInputHiddenLayer[inputLayerNodeNumber][firstHiddenLayerNodeNumber];
    float roundweightHiddenHiddenLayer[firstHiddenLayerNodeNumber][secondHiddenLayerNodeNumber];
    float roundweightHiddenOutputLayer[secondHiddenLayerNodeNumber][outputLayerNodeNumber];
    float roundbiasFirstHiddenLayer[firstHiddenLayerNodeNumber];
    float roundbiasSecondHiddenLayer[secondHiddenLayerNodeNumber];
```

Ⅱ. The code of MLP------detail

- 给所有的权值和偏置，随机赋初值，并定义函数。

- 赋值初始值不能为0，也不应该相同，采用高斯函数获得随机数来赋值，这样更容易收敛。

```
//gaussian function
float function::randomGaussian(float miu,float sigma){//lei wai ding yi (you tou wen jian)
    static float V1, V2, S;
    static int phase = 0;
    float X;
    float gaussianOutput;
    if (phase == 0){
        do{
            float U1 = (float)rand() / RAND_MAX;
            float U2 = (float)rand() / RAND_MAX;
            V1 = 2 * U1 - 1;
            V2 = 2 * U2 - 1;
            S = V1 * V1 + V2 * V2;
        } while (S >= 1 || S == 0);
        X = V1 * sqrt(-2 * log(S) / S);
    }
    else
        X = V2 * sqrt(-2 * log(S) / S);
    phase = 1 - phase;
    gaussianOutput = X * sigma + miu;
    return gaussianOutput;
}
//sigmoid
float function::sigmoid(float x){//lei wai ding yi (you tou wen jian)
float y;
y =1/(1+exp(-x));
return y;
}
//dsigmoid
float function::dsigmoid(float x){
float z;
z=x*(1-x);
return z;
}
```

Ⅱ. The code of MLP------detail

- 2.两个for循环可以实现两层之间公式计算，但如果公式中有三个以上变量则难以表示
- 解决方案：需要设置一个值来暂存某两个层之间的值，再另外两层之间调用这个值，从而化简成两个参数的表达式；
- 3.每迭代一次（输入全部数据，训练出一组网络权值和偏置之后），权重和偏置的值要保留，每层结点的值要清零，进行重新训练。

```
//forward
float feedforward(inputLayer &inp,nodeValue &nod,parameter &par){//dingyihanshu feedforward ,xingcan inp shi yinyongbianliang
float tempFirst=0,tempSecond=0,tempOut=0,sum=0;
function fun;//han shu shi li hua


for(int n=0;n<dataNumber;n++){
    for(int i=0;i<firstHiddenLayerNodeNumber;i++){
        tempFirst=0;
        nod.tempResultFirst[n][i]=0;
        for(int j=0;j<inputLayerNodeNumber;j++){
            tempFirst=tempFirst+par.weightInputHiddenLayer[j][i]*inp.inputData[n][j];
            nod.tempResultFirst[n][i]=tempFirst+par.biasFirstHiddenLayer[i];
        }
        nod.firstHiddenLayerNodeValue[n][i]=fun.sigmoid(nod.tempResultFirst[n][i]);
    }
    for(int i=0;i<secondHiddenLayerNodeNumber;i++){
        tempSecond=0;
        nod.tempResultSecond[n][i]=0;
        for(int j=0;j<firstHiddenLayerNodeNumber;j++){
            tempSecond=tempSecond+par.weightHiddenHiddenLayer[j][i]*nod.firstHiddenLayerNodeValue[n][j];
            nod.tempResultSecond[n][i]=tempSecond+par.biasSecondHiddenLayer[i];
        }
        nod.secondHiddenLayerNodeValue[n][i]=fun.sigmoid(nod.tempResultSecond[n][i]);
    }
    for(int i=0;i<outputLayerNodeNumber;i++){
        tempOut=0;
        nod.tempResultOut[n][i]=0;
            for(int j=0;j<secondHiddenLayerNodeNumber;j++){
                tempOut=tempOut+par.weightHiddenOutputLayer[j][i]*nod.outputLayerNodeValue[n][i];
                nod.tempResultOut[n][i]=tempOut+par.biasOutputLayer[i];

            }
            sum=sum+exp(nod.tempResultOut[n][i]);
        }
        for(int i=0;i<outputLayerNodeNumber;i++){
            tempOut=0;
            for(int j=0;j<secondHiddenLayerNodeNumber;j++){
                tempOut=tempOut+par.weightHiddenOutputLayer[j][i]*nod.outputLayerNodeValue[n][i];
            }
            nod.outputLayerNodeValue[n][i]=exp(tempOut)/sum;
        }
}
    return 0;
```

```
float feedback(inputLayer &inp,nodeValue &nod,parameter &par,int epoch){
    roundparameter rou;
    function fun;
    float error=0;
    float acc=0;
    error/=2;

 for(int n=0;n<dataNumber;n++){
    for(int i=0;i<firstHiddenLayerNodeNumber;i++){
        nod.firstHiddenLayerNodeRoundValue[i]=0;
        rou.roundbiasFirstHiddenLayer[i]=0;
        nod.tempResultFirst[n][i]=0;
        for(int j=0;j<secondHiddenLayerNodeNumber;j++){
            rou.roundweightHiddenHiddenLayer[i][j]=0;
        }
    }
    for(int i=0;i<secondHiddenLayerNodeNumber;i++){
        nod.secondHiddenLayerNodeRoundValue[i]=0;
        rou.roundbiasSecondHiddenLayer[i]=0;
        nod.tempResultSecond[n][i]=0;
        for(int j=0;j<outputLayerNodeNumber;j++){
            rou.roundweightHiddenOutputLayer[i][j]=0;
        }
    }
    for(int i=0;i<outputLayerNodeNumber;i++){
        rou.roundbiasOutputLayer[i]=0;
        //nod.tempResultOut[n][i]=0;
    }
    for(int i=0;i<inputLayerNodeNumber;i++){
        for(int j=0;j<firstHiddenLayerNodeNumber;j++){
            rou.roundweightInputHiddenLayer[i][j]=0;
        }
    }
}
```

将变量初始化

隐藏层2---输出层

```
for(int n=0;n<dataNumber;n++){
    //ohCENG
    int maxId=0;
    float tempMax=0;
    for(int i=0;i<outputLayerNodeNumber;i++){
        if(i==inp.label[n]){
            rou.roundbiasOutputLayer[i]+=(nod.outputLayerNodeValue[n][i]-1);//tidu
            error=error+(nod.outputLayerNodeValue[n][i]-1)*(nod.outputLayerNodeValue[n][i]-1)/outputLayerNodeNumber;
        }
        else{
            rou.roundbiasOutputLayer[i]+=(nod.outputLayerNodeValue[n][i]-0);//tidu
            error=error+(nod.outputLayerNodeValue[n][i]-0)*(nod.outputLayerNodeValue[n][i]-0)/outputLayerNodeNumber;
        }


        if(tempMax<nod.outputLayerNodeValue[n][i]){
            tempMax=nod.outputLayerNodeValue[n][i];
            maxId=i;
        }
    if(maxId==inp.label[n])
     acc=acc+1;//yi lun yi lun de

    for(int i=0;i<secondHiddenLayerNodeNumber;i++){
              for(int j=0;j<outputLayerNodeNumber;j++){
                  if(j==inp.label[n])
                      rou.roundweightHiddenOutputLayer[i][j]=rou.roundweightHiddenOutputLayer[i][j]+(nod.outputLayerNodeValue[n][j]-1)*nod.tempResult
                  else{
                      rou.roundweightHiddenOutputLayer[i][j]=rou.roundweightHiddenOutputLayer[i][j]+(nod.outputLayerNodeValue[n][j]-0)*nod.tempResult
                  }
              }
          }
    for(int i=0;i<secondHiddenLayerNodeNumber;i++){
    nod.secondHiddenLayerNodeRoundValue[i]=0;
    for(int j=0;j<outputLayerNodeNumber;j++){
        if(j==inp.label[n])
            nod.secondHiddenLayerNodeRoundValue[i]+=(nod.outputLayerNodeValue[n][j]-1)*(1-fun.sigmoid(nod.tempResultOut[n][j]))*fun.sigmoid(nod.tempRe
        else{
            nod.secondHiddenLayerNodeRoundValue[i]+=(nod.outputLayerNodeValue[n][j]-0)*(1-fun.sigmoid(nod.tempResultOut[n][j]))*fun.sigmoid(nod.tempRe
        }
```

反馈过程使用前馈的值

```
        //
        //fh-shCENG
        //
        for(int i=0;i<firstHiddenLayerNodeNumber;i++){
            for(int j;j<secondHiddenLayerNodeNumber;j++){
            rou.roundbiasSecondHiddenLayer[j]+=nod.secondHiddenLayerNodeRoundValue[j]*(1-fun.sigmoid(nod.tempResultSecond[n][j]))*fun.sigmoid(nod.tempResultSec
        }
    }
        for(int i=0;i<firstHiddenLayerNodeNumber;i++){
            for(int j;j<secondHiddenLayerNodeNumber;j++){
                rou.roundweightHiddenHiddenLayer[i][j]+=nod.secondHiddenLayerNodeRoundValue[j]*(1-fun.sigmoid(nod.tempResultSecond[n][j]))*fun.sigmoid(nod.temp
            }
        }
        for(int i=0;i<firstHiddenLayerNodeNumber;i++){
            for(int j=0;j<secondHiddenLayerNodeNumber;j++){
                nod.firstHiddenLayerNodeRoundValue[i]+=nod.secondHiddenLayerNodeRoundValue[j]*(1-fun.sigmoid(nod.tempResultSecond[n][j]))*fun.sigmoid(nod.tempR
[i][j];
            }
        }
        for(int i=0;i<firstHiddenLayerNodeNumber;i++){
            for(int j=0;j<secondHiddenLayerNodeNumber;j++){
                par.weightHiddenHiddenLayer[i][j]-=learningRate*rou.roundweightHiddenHiddenLayer[i][j];
            }
        }
        for(int i=0;i<secondHiddenLayerNodeNumber;i++){
            par.biasSecondHiddenLayer[i]-=learningRate*rou.roundbiasSecondHiddenLayer[i];
        }

        //
```

隐藏层1---隐藏层2

偏置的梯度

权值的梯度

输入层---隐藏层1需要使用的值（在此求出）

权值更新

偏置更新

输入层---隐藏层1

```
//in-fgCENG
//
for(int i=0;i<inputLayerNodeNumber;i++){
    for(int j=0;j<firstHiddenLayerNodeNumber;j++){
        rou.roundbiasFirstHiddenLayer[j]+=nod.firstHiddenLayerNodeRoundValue[j]*(1-fun.sigmoid(nod.tempResultFirst[n][j]))*fun.sigmoid(nod.tempR
    }
}
for(int i=0;i<inputLayerNodeNumber;i++){
    for(int j=0;j<firstHiddenLayerNodeNumber;j++){
        rou.roundweightInputHiddenLayer[i][j]+=nod.firstHiddenLayerNodeRoundValue[j]*(1-fun.sigmoid(nod.tempResultFirst[n][j]))*fun.sigmoid(nod.
    }
}
for(int i=0;i<inputLayerNodeNumber;i++){
    for(int j=0;j<firstHiddenLayerNodeNumber;j++){
        par.weightInputHiddenLayer[i][j]-=learningRate*rou.roundweightInputHiddenLayer[i][j];
    }
}
for(int i=0;i<firstHiddenLayerNodeNumber;i++){
    par.biasFirstHiddenLayer[i]-=learningRate*rou.roundbiasFirstHiddenLayer[i];
}
```

```
for(int i=0;i<outputLayerNodeNumber;i++){
    rou.roundbiasOutputLayer[i]=0;
}
for(int i=0;i<secondHiddenLayerNodeNumber;i++){
    nod.secondHiddenLayerNodeRoundValue[i]=0;
    rou.roundbiasSecondHiddenLayer[i]=0;
    for(int j=0;j<outputLayerNodeNumber;j++){
        rou.roundweightHiddenOutputLayer[i][j];

    }
}
for(int i=0;i<firstHiddenLayerNodeNumber;i++){
    nod.firstHiddenLayerNodeRoundValue[i]=0;
    rou.roundbiasFirstHiddenLayer[i]=0;
    for(int j=0;j<secondHiddenLayerNodeNumber;j++){
        rou.roundweightHiddenHiddenLayer[i][j];
    }
}
for(int i=0;i<inputLayerNodeNumber;i++){
```

更新权值和偏置之后，将每层节点值、权值梯度、偏置梯度统一清零

```
int main(){
    inputLayer inp;
    parameter par;
    function fun;
    nodeValue nod;
    ifstream dataFile;
    dataFile.open("/home/lina/programme/MLP12345678/MLPtest4/new");
    for(int i=0;i<dataNumber;i++){
        for(int j=0;j<dataDimension;j++){
            dataFile>>inp.inputData[i][j];
        }
    }
    dataFile.close();

    ifstream labelFile;
    labelFile.open("/home/lina/programme/MLP12345678/MLPtest4/datalabel.txt");
    for(int i=0;i<dataNumber;i++){
        labelFile>>inp.label[i];
    }
    labelFile.close();

    for(int i=0;i<inputLayerNodeNumber;i++){
        for(int j=0;j<firstHiddenLayerNodeNumber;j++){
        par.weightInputHiddenLayer[i][j]=fun.randomGaussian(0,0.01);
        }
    }
    for(int i=0;i<firstHiddenLayerNodeNumber;i++){
        for(int j=0;j<secondHiddenLayerNodeNumber;j++){
            par.weightHiddenHiddenLayer[i][j]=fun.randomGaussian(0,0.01);
        }
    }
    for(int i=0;i<secondHiddenLayerNodeNumber;i++){
        for(int j=0;j<outputLayerNodeNumber;j++){
            par.weightHiddenOutputLayer[i][j]=fun.randomGaussian(0,0.01);
        }
    }
    for(int i=0;i<outputLayerNodeNumber;i++){
        par.biasOutputLayer[i]=fun.randomGaussian(0,0.01);
    }
    for(int i=0;i<secondHiddenLayerNodeNumber;i++){
        par.biasSecondHiddenLayer[i]=fun.randomGaussian(0,0.01);
    }
    for(int i=0;i<firstHiddenLayerNodeNumber;i++){
        par.biasFirstHiddenLayer[i]=fun.randomGaussian(0,0.01);
    }
```

读数据

读标签

给权值和偏置随机赋值（高斯函数）

前馈反馈过程

```
//die dai
for(int i=0;i<epochNum;i++){
    feedforward(inp,nod,par);
    feedback(inp,nod,par,i);
}

FILE* fp;
fp=fopen("/home/lina/programme/MLP12345678/MLPtest4/weight.txt", "w");
for(int i=0;i<inputLayerNodeNumber;i++){
    for(int j=0;j<firstHiddenLayerNodeNumber;j++){
        cout<<par.weightInputHiddenLayer[i][j]<<"\t";
    }
    cout<<endl;
}

for(int i=0;i<firstHiddenLayerNodeNumber;i++){
    for(int j=0;j<secondHiddenLayerNodeNumber;j++){
        cout<<par.weightHiddenHiddenLayer[i][j]<<"\t";
    }
    cout<<endl;
}
for(int i=0;i<secondHiddenLayerNodeNumber;i++){
    for(int j=0;j<outputLayerNodeNumber;j++){
        cout<<par.weightHiddenOutputLayer[i][j]<<"\t";
    }
    cout<<endl;
}
for(int i=0;i<outputLayerNodeNumber;i++){
    cout<<par.biasOutputLayer[i]<<"\t";
}
 cout<<endl;
for(int i=0;i<secondHiddenLayerNodeNumber;i++){
    cout<<par.biasSecondHiddenLayer[i]<<"\t";
}
 cout<<endl;
for(int i=0;i<firstHiddenLayerNodeNumber;i++){
    cout<<par.biasFirstHiddenLayer[i]<<"\t";
}
 cout<<endl;
 fclose(fp);
 return 0;
}
```

权值和偏置输出到文本中

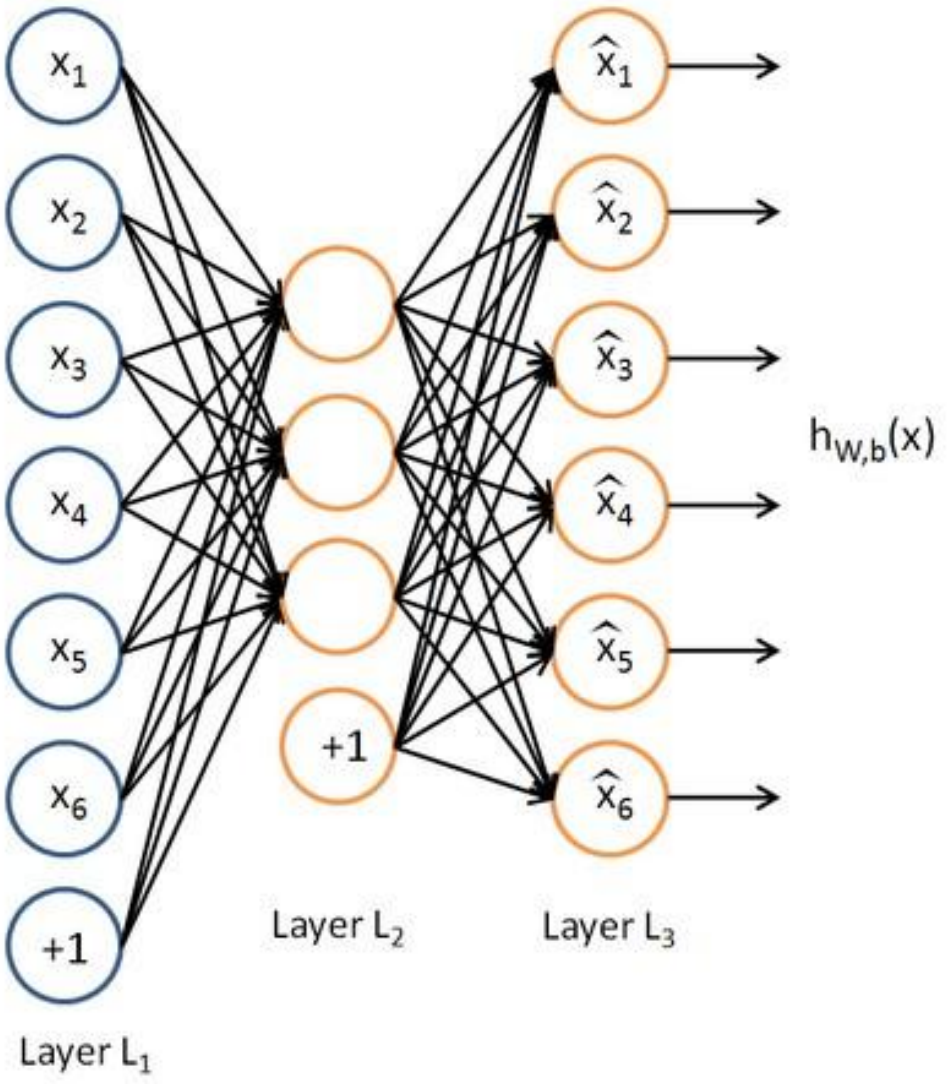
Ⅱ. The code of MLP

- 影响BP算法的最终收敛结果的参数有以下几个：

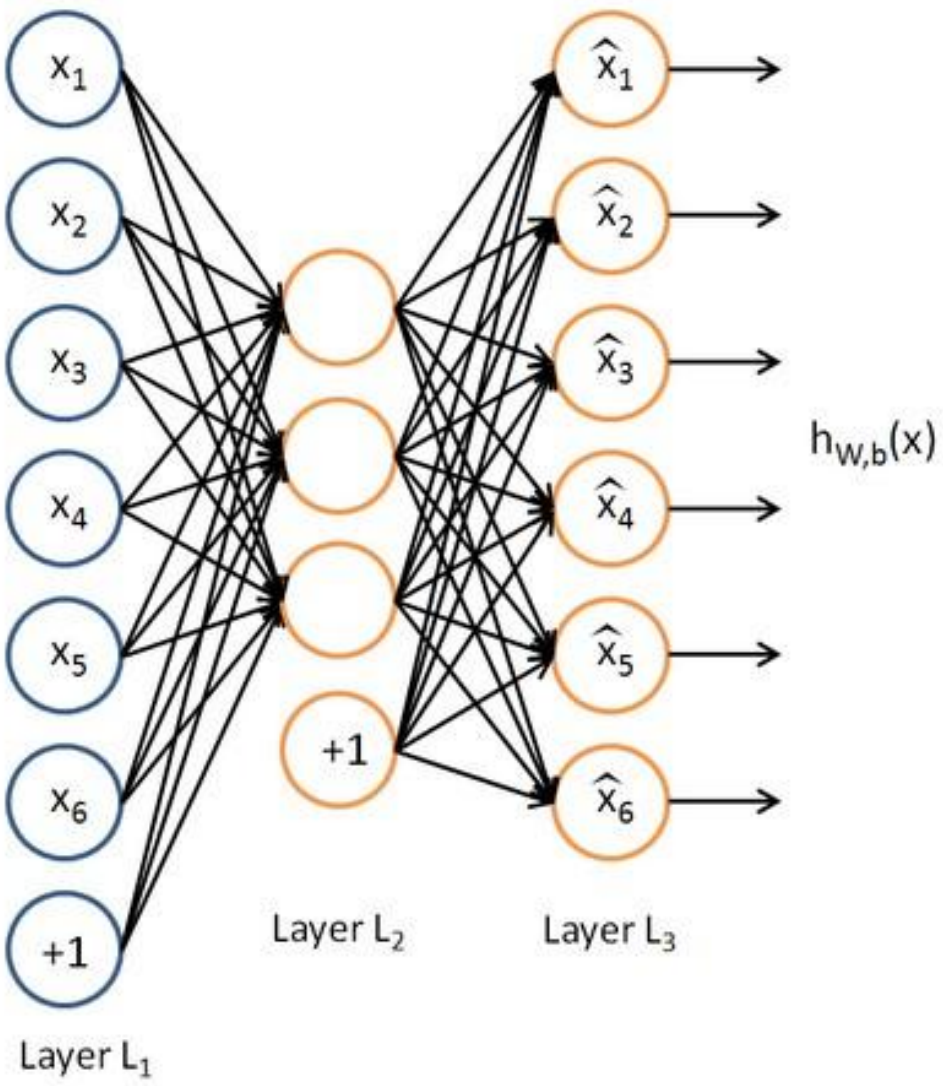
- 1）初始权值的影响，用高斯函数进行随机赋值；
- 2）学习率：
- 学习率太大，刚开始看起来收敛速度会比较快，但是很容易出现算法震荡而无法收敛或收敛很慢；
- 学习率太小，权值调整非常的慢，收敛速度也会非常慢，且一旦陷入局部最优，就容易停在那里不动。
- 学习率可选择在0.1~0.01之间。

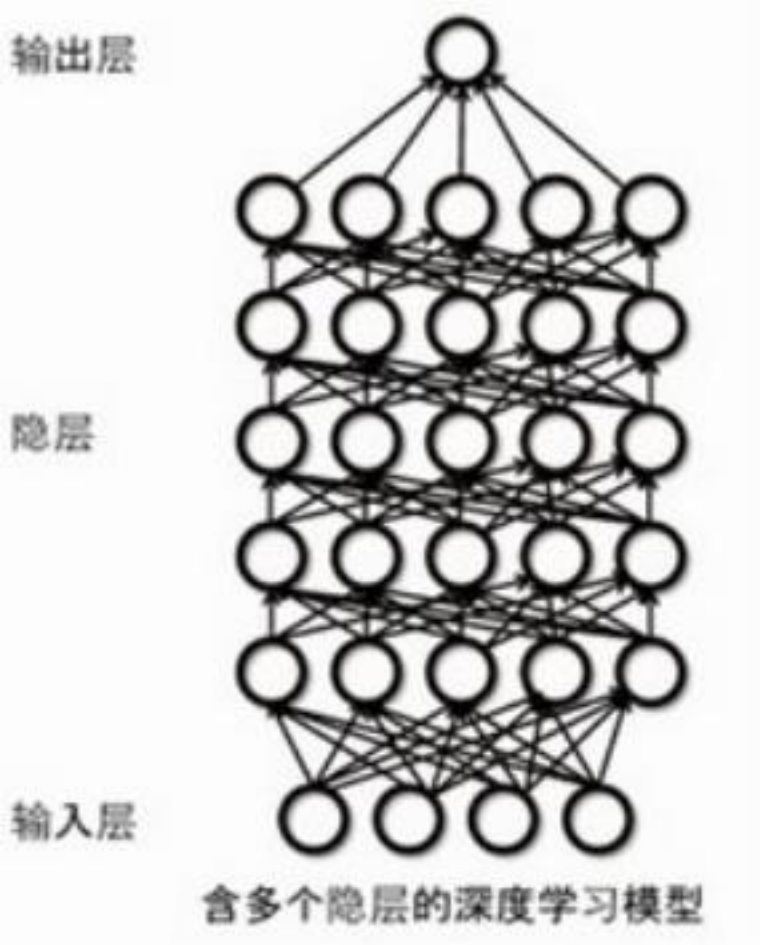
Ⅲ. The knowledge of Auto-encoder

- Auto-encoder（自编码算法）：
- 无监督学习算法，
- 尽可能复现输入信号的神经网络
- 它使用了反向传播算法，
- 目的：目标值等于输入值 hW,b(x)≈x
- 实现非线性降维
- （相比PCA实现线性降维）



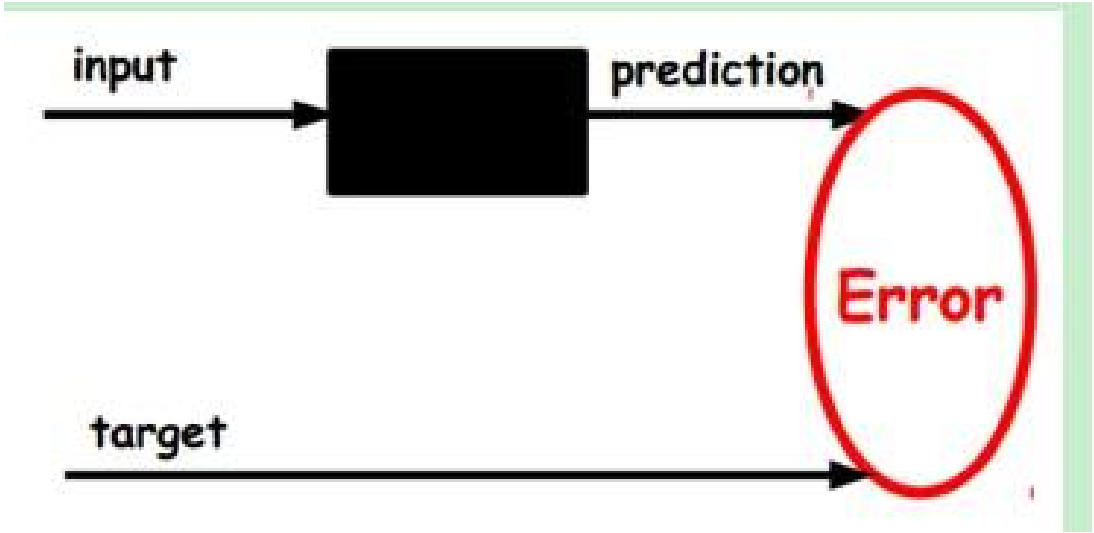
Ⅲ. The knowledge of Auto-encoder

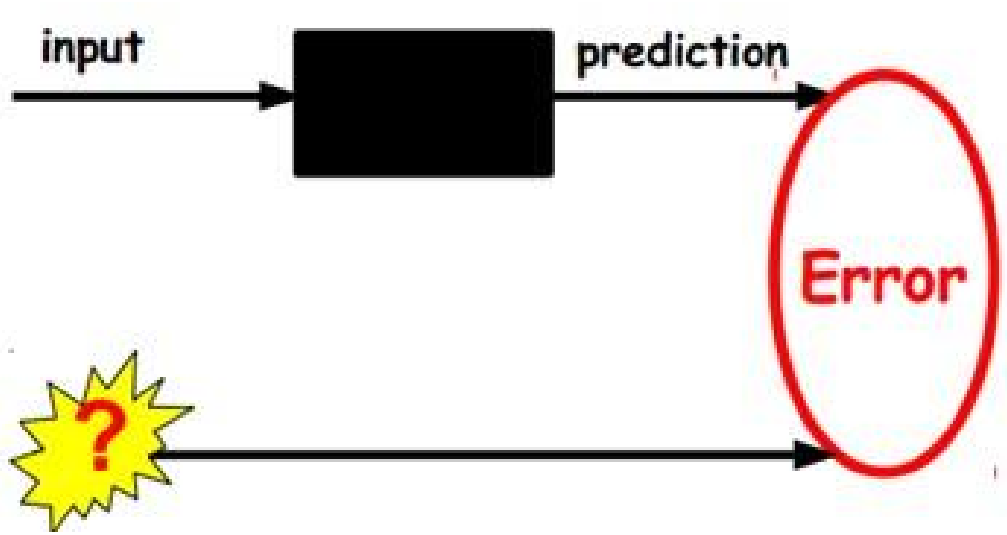




Ⅲ. The knowledge of Auto-encoder

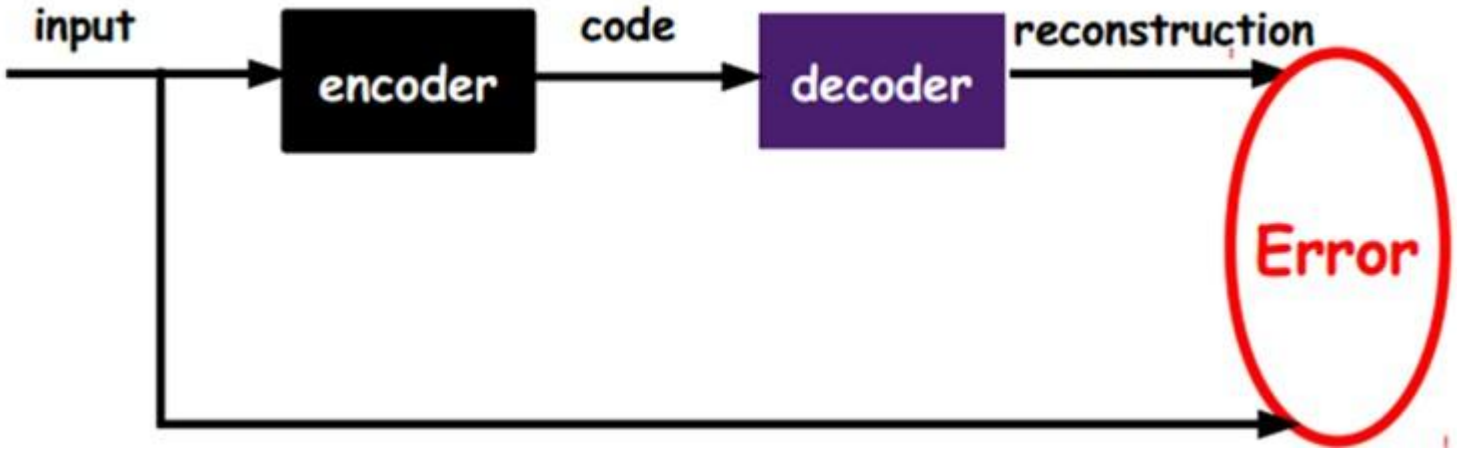
1）给定无标签数据，用非监督学习学习特征



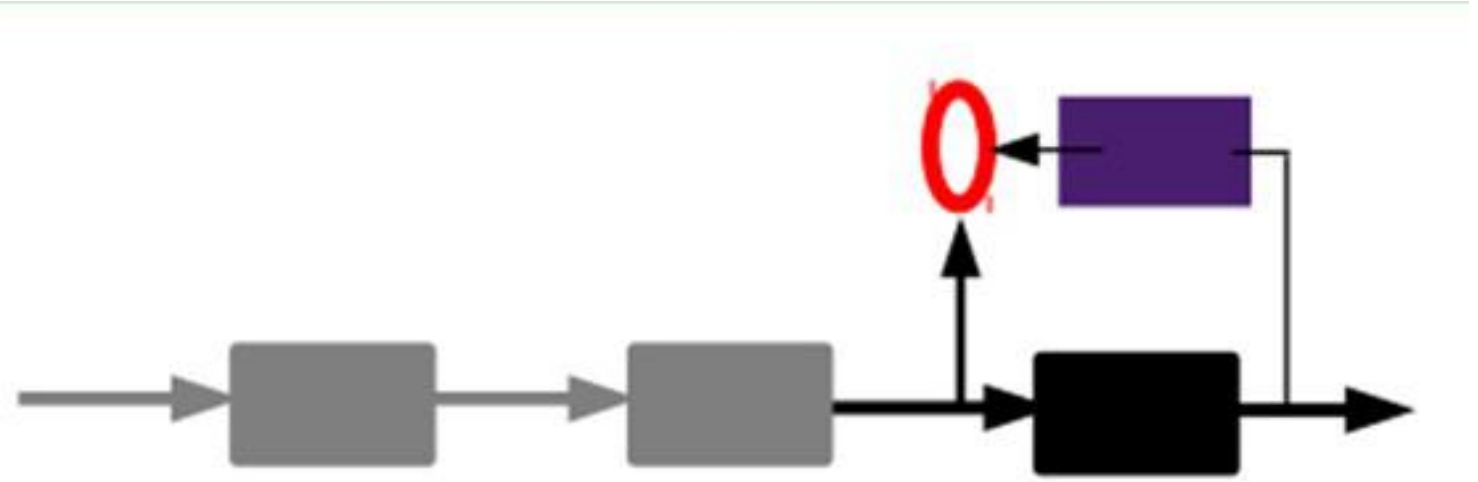


Ⅲ. The knowledge of Auto-encoder

2）通过编码器产生特征，然后训练下一层。这样逐层训练：



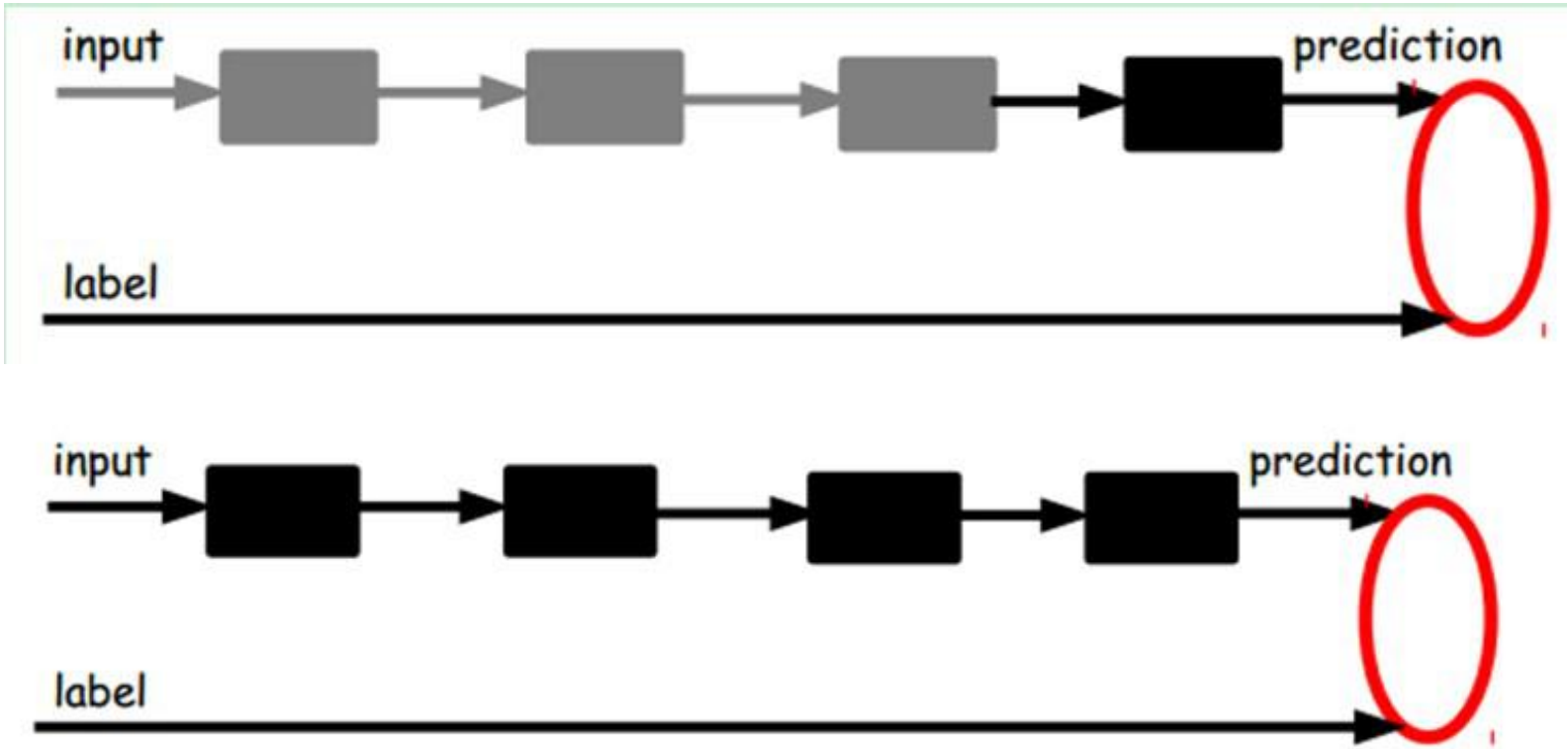
Ⅲ. The knowledge of Auto-encoder



第二层与第一层训练方式相同，以此类推

Ⅲ. The knowledge of Auto-encoder

3）有监督微调：



Ⅲ. The knowledge of Auto-encoder

- 1）求各神经层的激活值
- 对每层的神经元进行前向传导计算（前馈传导），利用前向传导公式，得到各层的激活值
- 2）计算W和B的残差，用梯度下降法更新W和B，使输出不断接近输入
- 使用后向传导算法（BP）求出最终输出层与各层神经元的“残差”，利用梯度下降法不断更新W和B，使输出越来越接近输入（使用梯度下降法更新权重参数W和B）

- PCA（主成分分析）:Principal Component Analysis
- 数据降维：将n个特征降维到k个，可以用来进行数据压缩，例如100维的向量最后可以用10维来表示，那么压缩率为90%。（线性降维）

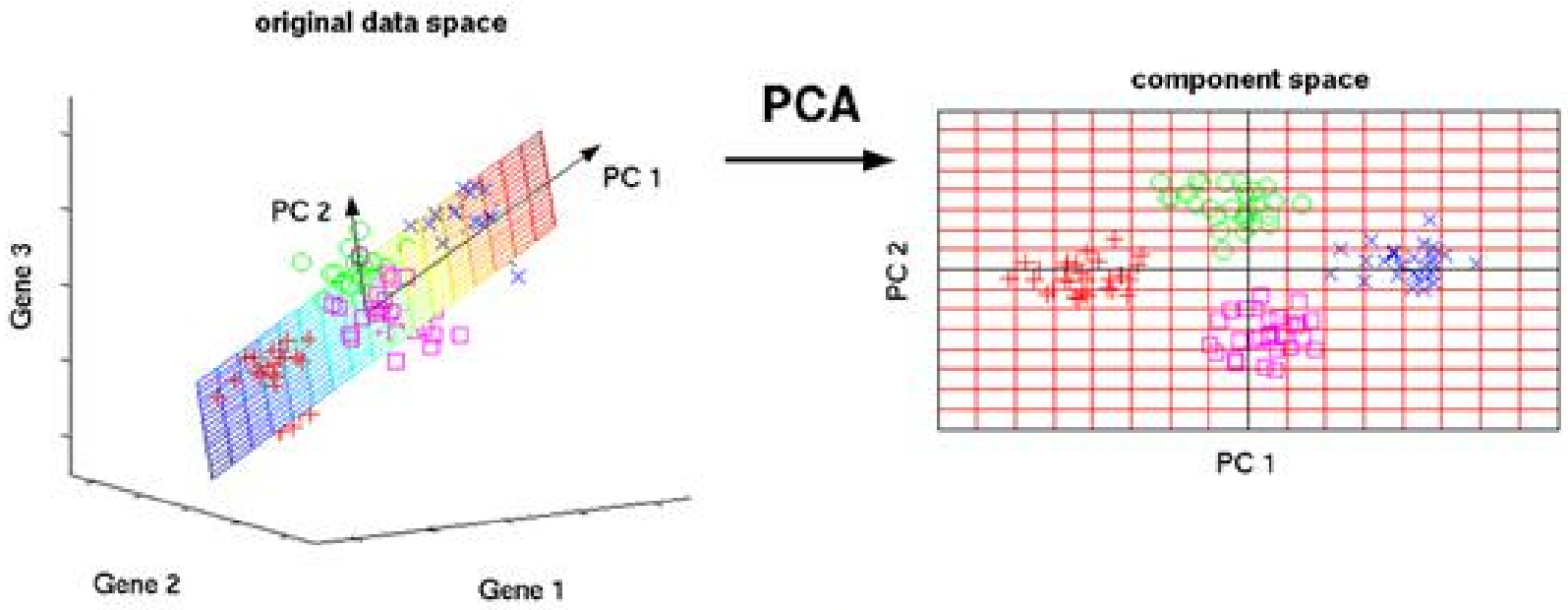
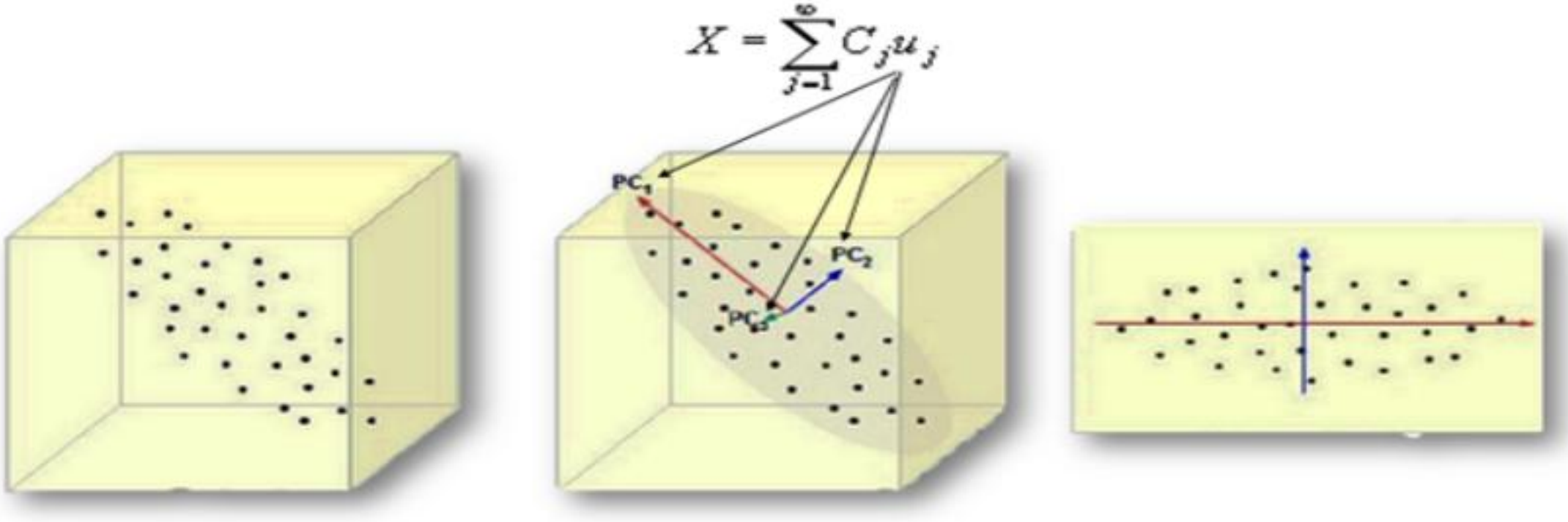


图 1

1）将原来的数据在新坐标系下表示
2）把某些坐标轴去掉（把区分数据不明显的坐标轴去掉）
3）数据降维



Ⅴ. Future Work

1.斯坦福的Andrew Ng机器学习视频（coursera）

2.完成高级程序语言设计课程和编程练习（截止11.16）

3.尽快学习C++面向对象

4.每周练习英语听力（CCTV---dialogue）

	星期一	星期二	星期三	星期四	星期五	星期六
上午	C编程	DL学习	C编程	英语写作	DL学习	C编程
	C编程	DL学习	C编程	英语口语	DL学习	C编程
下午	汇报	政治	模式识别	数字图像	傅里叶光谱	中国文化
	汇报	政治	模式识别	数字图像	傅里叶光谱	中国文化
晚上	汇报总结	英语dialogue	VALSE	课堂作业	课堂作业	课程整理

THANKS

中国海洋大学