日常记录

冯哲*
计算机科学与技术学院

2020 年 11 月 25 日 更新

目录

*电子邮件：11945852710qq.com

第一部分　神经网络形式化

1　全连接神经网络

1.1　数学推导

1.1.1　作用过程示例

变量说明：

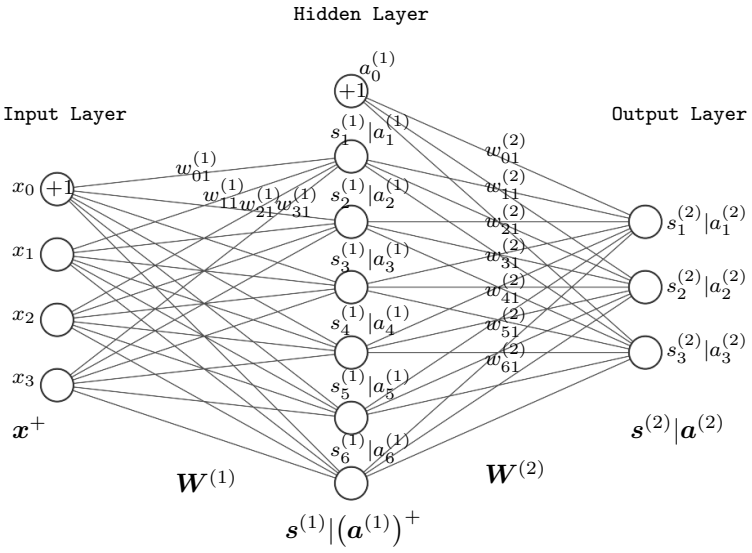
- 以零为下角标的变量为偏置（bias）。
- x 表示输入数据（一个训练样本为一个行向量），y 表示标签（one-hot 编码，为一个行向量），s 为求和值，a 为激活值。
- x_j^i 的上标表示第 i 行数据样本，下标表示该行数据样本的第 j 个值。
- x, s, a 若无下角标，则表示向量化。
- 多分类标签 y 采用 one-hot 编码。

定义 1．向量上标+ 运算符：为该向量添加*Bias*列，具体在第*0*列添加全*1*。
向量上标- 运算符：为该向量删去*Bias*列，具体删除第*0*列。

例 1．$$a=\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \quad a^+=\begin{pmatrix} 1 & a & b & c \\ 1 & d & e & f \\ 1 & g & h & i \end{pmatrix} \quad a^-=\begin{pmatrix} b & c \\ e & f \\ h & i \end{pmatrix}$$

定义 2．同维矩阵对应位置相乘运算：$A \circ B$ ．

输入层三个神经元，一个隐藏层六个神经元，输出层三个神经元(三分类)为例。激活函数以逻辑函数，损失函数以均方差损失为例。




```
function main ( int  RValue )
{
    frame(Nueron,Acti,userDefine,dataSet,fcnn,fclayer,adamP,loss,losstest,epoch,i,j,acc) and (
    MinstHWDataLoading();
    int Nueron[4]<==={784,512,256,10} and skip;
    int Acti[4]<==={0,3,3,5} and skip;                                                    I
    float loss<==0.0 and skip;
    float losstest<==0.0 and skip;
    int epoch<==1 and skip;
    int i<==0 and skip;

    Custom userDefine and skip;DataSet dataSet and skip;
    FCNN fcnn and skip;FCLayer fclayer and skip;AdamPara adamP and skip;
    InitCustom(&userDefine,RValue);InitDataSet(&dataSet,RValue);
    InitFCNN(&fcnn,RValue);InitFCLayer(&fclayer,RValue);initAdam(fcnn,&adamPara);

    userDefine.CompleteSampleNum:=70000;
    userDefine.TrainSampleNum:=60000;
    userDefine.TestSampleNum:=10000;
    userDefine.SampleDimensionNum:=784;                                                   II
    userDefine.HiddenLayerNum:=2;
    userDefine.ClassificationNum:=10;
    userDefine.LossFuncNum:=1;
    userDefine.WeightInitWayNum:=3;
    userDefine.BatchSize:=200;
    userDefine.XValArray:=Xval;
    userDefine.YValArray:=Yval;
    userDefine.NeuronNumArray:=Nueron;
    userDefine.ActiFuncNumArray:=Acti;
    DumpCustom(userDefine,RValue);
    LoadParaFromCustom(userDefine,&dataSet,&fcnn);

    DatasetConstruction(userDefine,&dataSet);                                             III

    CreateNNSpaceAndLoadinPara2FCLayer(&fcnn,userDefine,RValue);
    NNWeightinit(&fcnn,RValue);                                                           IV

    while( (i<epoch) )
    {                                                                                     VIII
        int j<==0 and skip;

        while ( (j<dataSet.BatchNum) )
        {                                                                                 VI
            loss:=NNforward(dataSet.BatchTrainFeature[j],dataSet.BatchTrainLabel[j],&fcnn,RValue);
            NNBackward(&fcnn,RValue);
            Adam(&fcnn,&adamP);                                                           V
            j:=j+1
        };
        losstest:=NNforward(dataSet.TestFeature,dataSet.TestLabelOneHot,&fcnn,RValue);
        float acc and skip;                                                               VII
        acc:=testAcc(fcnn,dataSet,RValue);
        i:=i+1
    }
    )
};
```

作用过程:

1. 向前传播计算误差:

$$\left(x^{(0)}\right)^+ w^{(1)} = s^{(1)}, \quad \sigma\left(s^{(1)}\right) = a^{(1)}$$

$$\left(a^{(1)}\right)^{+}w^{(2)}=s^{(2)},\quad \sigma(s^{(2)})=a^{(2)}$$

$$Loss\quad L=\frac{1}{2}\sum_{k=1}^{3}(a_k^{(2)}-y_k)^2$$

假设有 1 个训练样本:

(1) x 的维度就是 1×3。

(2) x^{+} 的维度就是 1×4。

(3) $w^{(1)}$ 的维度就是 4×6。

(4) $s^{(1)}$ 的维度就是 1×6。

(5) $a^{(1)}$ 的维度就是 1×6。

(6) $\left(a^{(1)}\right)^{+}$ 的维度就是 1×7。

(7) $w^{(2)}$ 的维度就是 7×3。

(8) $s^{(2)}$ 的维度就是 1×3。

(9) $a^{(2)}$ 的维度就是 1×3。 (输出层)

2. 误差反向传播计算权值梯度:

(1) 输出层权值梯度:

$$Loss,\quad L=\frac{1}{2}\sum_{i=1}^{K}(a_i^{(2)}-y_i)^2$$

$$\begin{aligned}\frac{\partial L}{w_{ik}^{(2)}}&=\frac{\partial}{\partial w_{ik}^{(2)}}\frac{1}{2}\sum_{i=1}^{K}(a_i^{(2)}-y_i)^2\\&=\left(\sigma(s_k^{(2)})-y_k\right)\sigma(s_k^{(2)})\left(1-\sigma(s_k^{(2)})\right)\frac{\partial s_k^{(2)}}{\partial w_{ik}^{(2)}}\\&=\left(\sigma(s_k^{(2)})-y_k\right)\sigma(s_k^{(2)})\left(1-\sigma(s_k^{(2)})\right)\left(a_i^{(1)}\right)^{+}\end{aligned}$$

$$\begin{aligned}令,\quad \delta_k^{(2)}&=\left(\sigma(s_k^{(2)})-y_k\right)\sigma(s_k^{(2)})\left(1-\sigma(s_k^{(2)})\right)\\&=\left(a_k^{(2)}-y_k\right)a_k^{(2)}\left(1-a_k^{(2)}\right)\end{aligned}$$

$$则,\quad \frac{\partial L}{w_{ik}^{(2)}}=\delta_k^{(2)}\left(a_i^{(1)}\right)^{+}$$

(2) 隐藏层权值梯度:

$$\begin{aligned}\frac{\partial L}{\partial w_{ij}^{(1)}}&=\frac{\partial}{\partial w_{ij}^{(1)}}\frac{1}{2}\sum_{k\in K}(a_k^{(2)}-y_k)^2\\&=\sum_{k\in K}(a_k^{(2)}-y_k)\frac{\partial\sigma(s_k^{(2)})}{\partial w_{ij}^{(1)}}\\&=\sum_{k\in K}(a_k^{(2)}-y_k)\sigma(s_k^{(2)})\left(1-\sigma(s_k^{(2)})\frac{\partial s_k^{(2)}}{\partial w_{ij}^{(1)}}\right.\\&=\sum_{k\in K}(a_k^{(2)}-y_k)\sigma(s_k^{(2)})\left(1-\sigma(s_k^{(2)})\frac{\partial s_k^{(2)}}{\partial a_j^{(1)}}\frac{\partial a_j^{(1)}}{\partial w_{ij}^{(1)}}\right.\end{aligned}$$

$$
\begin{aligned}
&= \sum_{k\in K}(a_k^{(2)} - y_k)\sigma(s_k^{(2)})\big(1-\sigma(s_k^{(2)})w_{jk}^{(2)}\frac{\partial a_j^{(1)}}{\partial w_{ij}^{(1)}} \\
&= \frac{\partial a_j^{(1)}}{\partial w_{ij}^{(1)}}\sum_{k\in K}(a_k^{(2)} - y_k)\sigma(s_k^{(2)})\big(1-\sigma(s_k^{(2)})w_{jk}^{(2)} \\
&= a_j^{(1)}\big(1-a_j^{(1)}\big)\frac{\partial s_j^{(1)}}{w_{ij}^{(1)}}\sum_{k\in K}\delta_k^{(2)}w_{jk}^{(2)} \\
&= a_j^{(1)}\big(1-a_j^{(1)}\big)\big(x_i^{(0)}\big)^+\sum_{k\in K}\delta_k^{(2)}w_{jk}^{(2)}
\end{aligned}
$$

令，$\delta_j^{(1)} = a_j^{(1)}(1-a_j^{(1)})\sum_{k\in K}\delta_k^{(2)}w_{jk}^{(2)}$

则，$\frac{\partial L}{\partial w_{ij}^{(1)}} = \delta_j^{(1)}\big(x_i^{(0)}\big)^+$

3. 更新权值：

$$W = W - \eta\frac{\partial Loss}{\partial W}$$

1.1.2 作用过程推广

用户指定神经网络相关参数：

变量名称	符号	备注
训练样本维度	N_t	
训练样本数量	N_T	
隐藏层数	H	包含H个隐藏层
第i层神经元个数	$N^{(i)}$	$i=0,1,\cdots,H,H+1$ $i=H+1$，输出层，$N^i=K$ 这里定义不带 bias 神经元。
第i层的激活函数	$\sigma^{(i)}$	在常用的 sigmoid，tanh，relu，leaky relu 中选择
分类类别数	K	
损失函数	S_l	在常用的 MeanSquareError，CorssEntropyLoss 中选择

常见激活函数：

1. sigmoid

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

$$\sigma'(z) = \sigma(z)(1-\sigma(z))$$

2. tanh

$$\sigma(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = 1 - \frac{2e^{-z}}{e^z + e^{-z}}$$
$$\sigma'(z) = 1 - \sigma(z)^2$$

3. relu

$$\sigma(z) = \begin{cases} 0, & \texttt{if } z < 0 \\ z, & \texttt{otherwise.} \end{cases}$$
$$\sigma'(z) = \begin{cases} 0, & \texttt{if } z < 0 \\ 1, & \texttt{otherwise.} \end{cases}$$

4. leaky relu

$$\sigma(z) = \begin{cases} z, & \texttt{if } z > 0 \\ az, & \texttt{otherwise.} \end{cases}$$
$$\sigma'(z) = \begin{cases} 1, & \texttt{if } z > 0 \\ a, & \texttt{otherwise.} \end{cases}$$

5. softmax

第 i 个类目的概率，$p_i = a_i^{(N+1)} = \sigma(s_i^{(N+1)}) = \dfrac{e^{s_i^{(N+1)}}}{\sum_{j=1}^{K} e^{s_j^{(N+1)}}}$

$$\frac{\partial p_i}{\partial s_j} = \frac{\partial \frac{e^{s_i}}{\sum_{k=1}^{N} e^{s_k}}}{\partial s_j}$$
$$f(x) = \frac{g(x)}{h(x)}, \qquad f'(x) = \frac{g'(x)h(x) - h'(x)g(x)}{h^2(x)}$$
$$g(x) = e^{s_i}, \qquad h(x) = \sum_{k=1}^{N} e^{s_k}$$

$i == j$

$$\frac{\partial \frac{e^{s_i}}{\sum_{k=1}^{N} e^{s_k}}}{\partial s_j} = \frac{e^{s_i}\sum_{k=1}^{N} e^{s_k} - e^{s_j}e^{s_i}}{\left(\sum_{k=1}^{N} e^{s_k}\right)^2} = \frac{e^{s_i}\left(\sum_{k=1}^{N} e^{s_k} - e^{s_j}\right)}{\left(\sum_{k=1}^{N} e^{s_k}\right)^2}$$

$i \neq j$

$$\frac{\partial \frac{e^{s_i}}{\sum_{k=1}^{N}}}{\partial s_j} = \frac{0 - e^{s_j}e^{s_i}}{\left(\sum_{k=1}^{N} e^{s_k}\right)^2} = -\frac{e^{s_i}}{\sum_{k=1}^{N} e^{s_k}} \frac{e^{s^j}}{\sum_{k=1}^{N} e^{s_k}}$$

$$= p_i(1 - p_j) \qquad\qquad = -p_i p_j$$

总结：

$$\delta_{ij} = \begin{cases} 1 & i == j \\ 0 & i \neq j \end{cases}$$

$$\Rightarrow \qquad \frac{\partial p_i}{\partial s_j} = p_i(\delta_{ij} - p_j)$$

常见损失函数：

1. `Mean Square Error`

$$Loss(z, y) = \frac{1}{2}\sum_{i=1}^{K}(z_i - y_i)^2$$

$$\frac{\partial Loss}{\partial z} = (z - y) \quad \text{向量相减}$$

2. `Cross Entropy Loss`

$$Loss(z, y) = -\sum_{i=1}^{K} y_i \log z_i = -y_k \log z_k (\texttt{one-hot 只有第k个分量为 1})$$

$$\frac{\partial Loss}{\partial z} = -\frac{y}{z} \quad \text{向量相除}$$

一般如果多分类问题输出层使用 `CrossEntropy` 损失，并用 `Softmax` 激活。
因为有：

$$\frac{\partial Loss(a^{(N+1)}, y)}{\partial s_j^{(N+1)}} = a_j^{(N+1)} - y_j$$

推导：

$$\frac{\partial Loss(a^{(N+1)}, y)}{\partial s_j^{(N+1)}} = \sum_{i=1}^{K} \frac{\partial Loss(a^{(N+1)}, y)}{\partial a_i} \frac{\partial a_i}{\partial s_j}$$

$$= -\sum_{i=1}^{K} \frac{y_i}{a_i} \frac{\partial a_i}{\partial s_j}$$

$$
\begin{aligned}
&= \big(-\frac{y_i}{a_i}\frac{\partial a_i}{\partial s_j}\big)_{i=j} - \sum_{i=1,i\neq j}^{K} \frac{y_i}{a_i}\frac{\partial a_i}{\partial s_j} \\
&= -\frac{y_j}{a_j} a_j(1-a_j) - \sum_{i=1,i\neq j}^{K} \frac{y_i}{a_i} \cdot -a_i a_j \\
&= -y_j + y_j a_j + \sum_{i=1,i\neq j}^{K} y_i a_j \\
&= -y_j + a_j \sum_{i=1}^{K} y_i \\
&= a_j^{(N+1)} - y_j
\end{aligned}
$$

变量说明：

变量	含义	备注
X	输入训练样本	维度 $N_T \times N_t$,相当于激活值矩阵$A^{(0)}$.
X^+	输入训练样本加偏置列	维度 $N_T \times (N_t+1)$,相当于激活值矩阵$\big(A^{(0)}\big)^+$.
$S^{(i)}$	第 i 层求和值矩阵	维度 $N_T \times N^{(i)}$, $i=1,\cdots,H,H+1$.
$A^{(i)}$	第 i 层激活值矩阵	维度 $N_T \times N^{(i)}$, $i=0,1,\cdots,H,H+1$.
$\big(A^{(i)}\big)^+$	第 i 层激活值矩阵加偏置列	维度 $N_T \times (N^{(i)}+1)$, $i=0,1,\cdots,H$.
$W^{(i)}$	第 i 层权值矩阵	维度 $N^{(i-1)} \times N^{(i)}$, $i=1,\cdots,H,H+1$.
$W^{(H+1)}$	输出层权值矩阵	维度 $N^{(N)} \times K$, 标签有 K 个类目。
$W_b^{(i)}$	第 i 层权值偏置矩阵	维度 $(N^{(i-1)}+1) \times N^{(i)}$, $i=1,\cdots,H,H+1$.
$W_b^{(H+1)}$	输出层权值偏置矩阵	维度 $(N^{(N)}+1) \times K$, 标签有 K 个类目。
$S^{(H+1)}$	输出层求和值矩阵	维度 $N_T \times K$.
$A^{(H+1)}$	输出层激活值矩阵	维度 $N_T \times K$.
Y	训练数据标签	维度 $N_T \times K$ (ONE-HOT).
$\Delta^{(i)}$	第 i 层的反向传播中间变量矩阵	维度 $N_T \times N^{(i)}$, $i=1,\cdots,H,H+1$.
$\Delta^{(H+1)}$	输出层反向传播中间变量矩阵	维度 $N_T \times K$.

神经网络向前传播

$$
\begin{aligned}
\big(A^{(0)}\big)^+ W_b^{(1)} = S^{(1)} \xrightarrow{\sigma+b} \big(A^{(1)}\big)^+ &\Rightarrow \cdots \Rightarrow \cdots \\
&\Rightarrow \big(A^{(H-1)}\big)^+ W_b^{(H)} = S^{(H)} \xrightarrow{\sigma+b} \big(A^{(H)}\big)^+ \\
&\Rightarrow \big(A^{(H)}\big)^+ W_b^{(H+1)} = S^{(H+1)} \xrightarrow{\sigma} A^{(H+1)} \\
&\Rightarrow \mathcal{L}(A^{(H+1)}, Y)
\end{aligned}
$$

神经网络反向传播

1．输出层权值求导：

$$\nabla_{W_b^{(H+1)}}\mathcal{L}(A^{(H+1)},Y)=\frac{1}{H_T}(A^{(H)^+})^T\Delta^{(H+1)}\qquad\text{（矩阵化）}$$

$$\Delta^{(H+1)}=\frac{\partial\mathcal{L}(A^{(H+1)},Y)}{\partial A^{(H+1)}}\circ\sigma'(S^{(H+1)})$$

当输出层使用 `Softmax+CrossEntropy` 时：

$$\Delta^{(H+1)}=A^{(H+1)}-Y$$

2．隐藏层权值求导：

$$\nabla_{W_b^{(H)}}\mathcal{L}(A^{(H+1)},Y)=\frac{1}{H_T}\left(A^{(H-1)^+}\right)^T\Delta^{(H)}$$

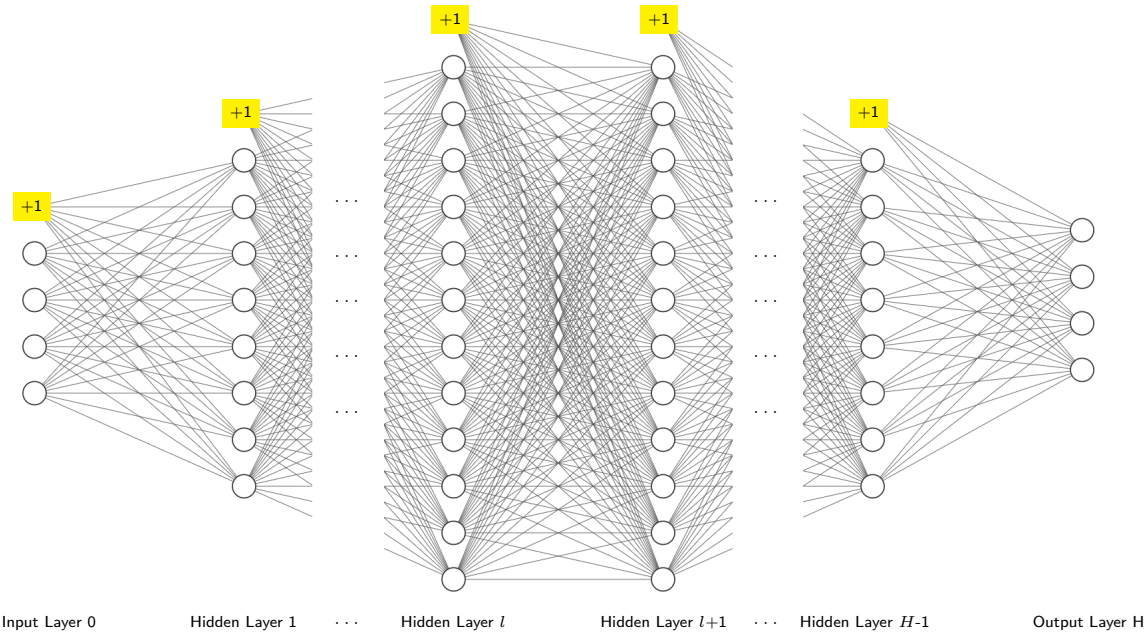
$$\text{令：}\ \Delta^{(H)}=\left(\Delta^{(H+1)}(W^{(H+1)})^T\right)\circ\sigma'^{(H)}(S^{(H)})$$

扩展： 对于第 i 个隐藏层的权值矩阵的梯度为($i=1,2,\cdots,H.$)：

$$\nabla_{W_b^{(i)}}\mathcal{L}(A^{(H+1)},Y)=\frac{1}{H_T}\left(A^{(i-1)^+}\right)^T\Delta^{(i)}$$

$$\text{令：}\ \Delta^{(i)}=\Delta^{(i+1)}(W^{(i+1)})^T\circ\sigma'^{(i)}(S^{(i)})$$

全连接神经网络结构图



神经网络张量流：

前向传播	维度	反向传播	维度
X	$N_T \times (N_t+1)$	LOSS	Scalar
$S^{(1)}$	$N_T \times N^{(1)}$	$\Delta^{(H+1)}$	$N_T \times K$
$(A^{(1)})^+$	$N_T \times (N^{(1)}+1)$	$\nabla_{W^{(H+1)}}\mathcal{L}$	$(N^{(H)}+1) \times K$
$S^{(2)}$	$N_T \times N^{(2)}$	$\Delta^{(H)}$	$N_T \times (N^{(H)}+1)$
$(A^{(2)})^+$	$N_T \times (N^{(2)}+1)$	$\nabla_{W^{(N)}}\mathcal{L}$	$(N^{(H-1)}+1) \times N^{(H)}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$S^{(H)}$	$N_T \times N^{(H)}$	$\Delta^{(i)}$	$N_T \times (N^{(i)}+1)$
$(A^{(H)})^+$	$N_T \times (N^{(H)}+1)$	$\nabla_{W^{(i)}}\mathcal{L}$	$(N^{(i-1)}+1) \times N^{(i)}$
$S^{(H+1)}$	$N_T \times K$	$\Delta^{(1)}$	$N_T \times (N^{(1)}+1)$
$A^{(H+1)}$	$N_T \times K$	$\nabla_{W^{(1)}}\mathcal{L}$	$(N_t+1) \times N^{(1)}$
comput Loss		update Weight	

优化算法

批量梯度下降（Batch Gradient Descent，BGD） **批**量梯度下降法是最原始的形式，它是指在每一次迭代时使用所有样本来进行梯度的更新。

1．对目标函数求导：

$$\frac{\Delta J(\theta_0,\theta_1)}{\Delta\theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)x_j^{(i)}$$

其中 $i=1,2,\cdots,m$ 表示样本数，$j=0,1$ 表示特征数，这里我们使用了偏置项 $x_0^{(i)}=1$ 。

2．每次迭代对参数进行更新：

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)x_j^{(i)}$$

伪码：

repeat{

$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta \left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$

(for $j = 0, 1$)

}

1．优点：

- 一次迭代是对所有样本进行计算，此时利用矩阵进行操作，实现了并行。
- 由全数据集确定的方向能够更好地代表样本总体，从而更准确地朝向极值所在的方向。当目标函数为凸函数时，BGD一定能够得到全局最优。

2．缺点：

- 当样本数目 m 很大时，每迭代一步都需要对所有样本计算，训练过程很慢。

随机梯度下降（Stochastic Gradient Descent，SGD） 随机梯度下降法不同于批量梯度下降，随机梯度下降是每次迭代使用一个样本来对参数进行更新。使得训练速度加快。

1．对一个样本的目标函数求偏导：

$$\frac{\Delta J^{(i)}\left(\theta_0, \theta_1\right)}{\theta_j} = \left(h_\theta \left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$$

2．参数更新：

$$\theta_j := \theta_j - \alpha \left(h_\theta \left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$$

伪码：

repeat{

for $i = 1, \cdots, m$ {

$\theta_j := \theta_j - \alpha \left(h_\theta \left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$

(for $j = 0, 1$)

}

}

1．优点：

- 由于不是在全部训练数据上的损失函数，而是在每轮迭代中，随机优化某一条训练数据上的损失函数，这样每一轮参数的更新速度大大加快。

2．缺点：

- 准确度下降。由于即使在目标函数为强凸函数的情况下，SGD仍旧无法做到线性收敛。
- 可能会收敛到局部最优，由于单个样本并不能代表全体样本的趋势。
- 不易于并行实现。

小批量梯度下降（Mini-Batch Gradient Descent, MBGD） 是对 BGD 以及 SGD 的一个折中办法。其思想是：每次迭代使用 batch_size 个样本来对参数进行更新。

这里我们假设 batch_size=10 ，样本数 m=1000 。

伪码：

repeat{

for i $= 1, 11, 21, 31, \ldots, 991$ {

$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{(i+9)} \left(h_\theta\left(x^{(k)}\right) - y^{(k)}\right) x_j^{(k)}$

(for $j = 0, 1$)

}

}

1．优点：

- 通过矩阵运算，每次在一个batch上优化神经网络参数并不会比单个数据慢太多。
- 每次使用一个batch可以大大减小收敛所需要的迭代次数，同时可以使收敛到的结果更加接近梯度下降的效果。(比如上例中的30W，设置batch_size=100时，需要迭代3000次，远小于SGD的30W次)
- 可实现并行化。

2．缺点：

- batch_size的不当选择可能会带来一些问题。

1.2 MSVL 语言框架

数据类型：单精度浮点数。

1.2.1 辅助函数

1. absolute
2. min
3. equal
4. F2S
5. sigmoid
6. tanh
7. relu
8. leakyRelu

1.2.2 矩阵操作函数

1. MatCreate
2. MatDelete
3. MatSetVal
4. MatShape
5. MatDump
6. MatAdd
7. MatSub
8. MatMul
9. MatProduct
10. MatNumMul
11. MatNumAdd
12. MatTrans
13. MatZeros
14. MatOnes
15. MatEye
16. MatRowSum
17. MatRowMax
18. MatExp
19. MatSquare
20. MatVectorSub
21. MatVectorDiv

22. MatCopy
23. MatPlusRow
24. MatPlusCol

1.2.3 矩阵激活函数

1. MatNoneActi
2. MatSoftmax
3. MatSigmoid
4. MatTanh
5. MatRelu
6. MatLeakyRelu

1.2.4 矩阵激活函数导数函数

1. MatDerivationNoneActi 0
2. MatDerivationSigmoid 1
3. MatDerivationTanh 2
4. MatDerivationRelu 3
5. MatDerivationLeakyRelu 4
6. MatDerivationSoftmax 5

1.2.5 损失函数

1. OneHot
2. MSE
3. CrossEntropy

1.2.6 损失函数的导函数

1. MSEDerivative
2. CrossEntropyDerivative

1.2.7 神经网络的初始化

神经网络初始化介绍

1. AllZero initialization
2. Random initialization
3. Xavier initialization
4. He initialization

相关函数

1. gaussrand_NORMAL 标准高斯随机生成
2. gaussrand 高斯随机生成
3. MatInitZero 全零初始化
4. MatInitRandomNormalization 随机初始化
5. MatInitXavier Xavier初始化
6. MatInitHe 凯明初始化

1.2.8 神经网络用户自定义参数

用户自定义参数	参数表示	数据类型	备注
输入训练样本的数量	N_sample	int	NULL
单一训练样本的维度	D_sample	int	NULL
训练样本的真值	Xval	float *	数组长度为 N_sample * D_sample
训练样本的标签	Yval	float *	数组长度为 N_sample
权值初始化方式	Style_initWeight	int	0 ->全0初始化 1 ->随机初始化 2 ->Xavier初始化 3 ->何凯明初始化
神经网络隐藏层层数	N_hidden	int	注意：这里是隐藏层层数。
各层神经元个数	N_layerNeuron	int *	数组长度为 0(输入层),1,...,N_hidden,N_hidden+1(输出层).
各层激活函数	Nstr_ActiFsHidden	int *	数组长度为 0(输入层),1,...,N_hidden,N_hidden+1(输出层). 取值的映射关系： 0 ->no activation（input layer） 1 ->sigmoid 2 ->tanh 3 ->relu 4 ->leaky relu 5 ->softmax (output layer)
分类类别数	N_out	int	NULL
输出层损失函数	Nstr_LossF	int	取值映射关系： 0 ->MSE 1 ->CE

1.2.9 神经网络的运算所需空间变量及创建所需空间函数

神经网络的运算所需空间变量	参数表示	数据类型	备注
神经网络激活值矩阵	P_ActiMat	Mat *	列表索引i=0(输入层),1,...,N_hidden, N_hidden+1(输出层)
神经网络激活值矩阵加偏置列	P_ActiMatPlus	Mat *	列表索引i=0(输入层),1,...,N_hidden 加偏置列偏置列全部置1.输出层不需要Plus
神经网络求和矩阵	P_SumMat	Mat *	列表索引i=0(输入层),1,...,N_hidden, N_hidden+1(输出层) i=0 时输入层求和矩阵行列置零无实际含义
神经网络权值矩阵	P_WeightMat	Mat *	列表索引i=0(输入层),1,...,N_hidden, N_hidden+1(输出层) i=0 时输入层求和矩阵行列置零无实际含义
神经网络权值偏置矩阵	P_WeightBiasMat	Mat*	列表索引i = 0(输入层), 1, ..., N hidden, N hidden + 1(输出层) i = 0 时输入层求和矩阵行列置零无实际含义
训练数据标签	Mat_oneHot	Mat	row=N_sample col=N_out
反向传播中间变量矩阵	P_DeltaMat	Mat *	列表索引i=0(输入层),1,...,N_hidden, N_hidden+1(输出层) i=0 时输入层求和矩阵行列置零无实际含义
权值偏置矩阵导数变量矩阵	P_NablaWbMat	Mat*	列表索引i = 0(输入层), 1, ..., N hidden, N hidden + 1(输出层) i = 0 时输入层求和矩阵行列置零无实际含义
激活函数对求和值导数矩阵	P_ActiFunDerivation	Mat*	列表索引i = 0(输入层), 1, ..., N hidden, N hidden + 1(输出层) i = 0 时输入层求和矩阵行列置零无实际含义

创建所需空间函数：

1. SpaceCreateActi 创建激活值矩阵所需空间
2. SpaceCreateActiPlus 创建激活值Plus矩阵所需空间
3. SpaceCreateSum 创建求和值矩阵所需空间
4. SpaceCreateWeight 创建权值矩阵所需空间
5. SpaceCreateWeightBias 创建权值偏置矩阵所需空间
6. SpaceCreateDelta 创建反向传播中间变量矩阵所需空间

1.2.10 初始化神经网络

函数：

1. NNinit

说明：

1. 输入数据喂入激活值（plus）矩阵
2. 标签数据喂入矩阵并且进行独热编码
3. 选择常用的初始化方式进行权值初始化

1.2.11 神经网络前向传播

神经网络向前传播所需参数变量	参数表示	数据类型	备注
神经网络激活值矩阵	P_ActiMat	Mat *	列表索引i=0(输入层),1,...,N_hidden, N_hidden+1(输出层)
神经网络激活值矩阵加偏置列	P_ActiMatPlus	Mat *	列表索引i=0(输入层),1,...,N_hidden 加偏置列偏置列全部置1.输出层不需要Plus
神经网络求和矩阵	P_SumMat	Mat *	列表索引i=0(输入层),1,...,N_hidden, N_hidden+1(输出层) i=0 时输入层求和矩阵行列置零无实际含义
神经网络权值偏置矩阵	P_WeightBiasMat	Mat *	列表索引i = 0(输入层), 1, ..., N hidden, N hidden + 1(输出层) i = 0 时输入层求和矩阵行列置零无实际含义
训练数据标签	Mat_oneHot	Mat	row=N_sample col=N_out
神经网络隐藏层层数	N_hidden	int	注意：这里是隐藏层层数。
各层激活函数	NStr_ActiFsHidden	int *	数组长度为 0(输入层),1,...,N hidden,N hidden+1(输出层). 取值的映射关系： 0 ->no activation（input layer） 1 ->sigmoid 2 ->tanh 3 ->relu 4 ->leaky relu 5 ->softmax (output layer)
输出层损失函数	Nstr_LossF	int	取值映射关系： 0 ->MSE 1 ->CE

1. MatActivate 激活函数选择函数
2. LossFunction 损失函数选择函数
3. NNforward 神经网络前向传播函数-返回损失

1.2.12 神经网络反向传播

神经网络反向传播所需参数变量	参数表示	数据类型	备注
神经网络隐藏层层数	N_hidden	int	注意：这里是隐藏层层数
输入样本的数量	N_sample	int	样本数量（训练样本）
各层神经元个数	N_layerNeuron	int *	数组长度为 0(输入层),1,...,N hidden,N hidden+1(输出层)
神经网络求和矩阵	P_SumMat	Mat *	列表索引i=0(输入层),1,...,N_hidden, N_hidden+1(输出层)
i=0 时输入层求和矩阵行列置零无实际含义	P_ActiMat	Mat *	列表索引i=0(输入层),1,...,N_hidden, N_hidden+1(输出层)
神经网络激活值矩阵加偏置列	P_ActiMatPlus	Mat *	列表索引i=0(输入层),1,...,N_hidden 加偏置列偏置列全部置1.输出层不需要Plus
反向传播中间变量矩阵	P_DeltaMat	Mat *	列表索引i = 0(输入层), 1, ..., N hidden, N hidden + 1(输出层) i = 0 时输入层求和矩阵行列置零无实际含义
训练数据标签	Mat_oneHot	Mat	row = N_sample col = N_out
训练数据标签	Mat_oneHot	Mat	row=N_sample col=N_out
各层激活函数	NStr_ActiFsHidden	int *	数组长度为 0(输入层),1,...,N hidden,N hidden+1(输出层). 取值的映射关系： 0 ->no activation（input layer） 1 ->sigmoid 2 ->tanh 3 ->relu 4 ->leaky relu 5 ->softmax (output layer)
输出层损失函数	Nstr_LossF	int	取值映射关系： 0 ->MSE 1 ->CE
激活函数对求和值导数矩阵	P_ActiFunDerivation	Mat *	列表索引i = 0(输入层), 1, ..., N hidden, N hidden + 1(输出层) i = 0 时输入层求和矩阵行列置零无实际含义

1. ActiFunDerivation

2. LossFunDerivation
3. NNOuputLayerBackward
4. NNBackward

1.2.13 优化算法

1. BGD（批量梯度下降）
2. SGD（随机梯度下降）
3. MBGD（小批量梯度下降）
4. Adam

demo 成型，损失减小，可收敛。

1.3 MSVL 语言框架重构

1.3.1 对象结构体

1. 矩阵 结构体

```
typedef struct {
   int row, col;     // rowNum and columnNum [int]
   float** element; // element, two dimensions
}Mat;
```

2. BP单层 结构体

```
typedef struct{
   Mat ActiMat;              // active value Matrix without bias column
   Mat ActiMatPlus;          // active value Matrix with bias column
   Mat SumMat;               // sum value Matrix
   Mat WeightMat;            // weight value Matrix without bias row
   Mat WeightBiasMat;        // weight value Matrix with bias row
   Mat DeltaMat;             // backtrack temporary variable Matrix
   Mat NablaWbMat;           // gradient Matrix for weight with bias
   Mat ActiFunDerivationMat; // active Function Dervation Matrix

   int NeuronNum;            // number of neuron [int]
   int AcitFuncNum;          // active function [int]
}FCLayer;
```

3. BP神经网络 结构体

```
typedef struct{
   int CurrentSampleNum;          // number of current samples [int]
   int SampleDimensionNum; // dimensions(features) of sample [int]
   int HiddenLayerNum;       // number of hidden layer [int]
   int WeightInitWayNum;     // weight initialization mode [int]

   FCLayer *Layer;           // layers of FCNN
   Mat OnehotMat;            // onehot code Matrix

   int ClassificationNum;    // Number of categories classified [int]
   int LossFuncNum;          // loss function [int]
}FCNN;
```

4. 用户自定义参数 结构体

```
typedef struct{
    int CompleteSampleNum;  // number of all samples [int]
   int TrainSampleNum;      // number of training samples [int]
   int TestSampleNum;       // number of test samples [int]
   int ValidationNum;       // number of validation samples [int]
   int SampleDimensionNum; // dimensions(features) of sample [int]
   int HiddenLayerNum;      // number of hidden layer [int]
   int WeightInitWayNum;    // weight initialization mode [int]
   float *XValArray;        // samples features value [float*]
   float *YValArray;        // samples labels value [float*]
   int *NeuronNumArray;     // numbers of every layers Neuron [int*]
   int *ActiFuncNumArray;   // activate functions of every layers [int*]
   int ClassificationNum;   // Number of categories classified [int]
   int LossFuncNum;         // loss function [int]
}Custom;
```

5. 数据集 结构体

```
typedef struct{
   Mat CompleteFeatureDataSet; // complete featrue Mat for FCNN
   Mat CompleteLabelDataSet; // complete label Mat without onehot
   Mat CompleteTrainFeature; // complete featrue Mat for FCNN training
   Mat CompleteTrainLabel;  // complete label Mat without onehot
   Mat *BatchTrainFeature;  // batch featrue Mat for FCNN training [three
       dimensions]
   Mat *BatchTrainLabel;    // batch label Mat without onehot [three
       dimensions]
   Mat *BatchTrainLabelOneHot; // batch label Mat with onehot [three
       dimensions]
   Mat TestFeature;      // featrue Mat for FCNN test
   Mat TestLabel;        // label Mat without onehot
   Mat TestLabelOneHot;     // label Mat with onehot
   //Mat ValidationFeature; // featrue Mat for FCNN Validation
   //Mat ValidationLabel;  // label Mat withoutone


   int CompleteSampleNum;  // number of all samples [int]
   int TrainSampleNum;      // number of training samples [int]
   int TestSampleNum;       // number of test samples [int]
   //int ValidationNum;     // number of validation samples [int]
```

```
    int SampleDimensionNum; // dimensions(features) of sample [int]
    int ClassificationNum;  // Number of categories classified [int]
    int BatchSize;          // batch size for dataset
    int BatchNum;           // number of batch
    int remainder;          // the last batch size
}DataSet;
```

batch 机制的实现:建立 batch 结构体，分割数据集，复制到 FCNN 中。

其中 TrainSampleNum = BatchSize*BatchNum+remainder，当remainder = 0时，代表能被整除。

1.3.2 开发流程

1. 用户自定义参数及总数据录入;
2. 数据集构建;
3. 神经网络初始化;
4. 前向传播构建;
5. 反向传播构建;
6. 优化算法构建;
7. 神经网络训练构建;
8. 神经网络测试构建。

神经网络用户自定义参数及总数据录入 目的是将相关参数及总的数据赋值给用户自定义结构体中（其中注意对数组数据的赋值，传引用即可），再将各个参数分发给各个结构体。

数据集构建 根据用户输入的数据相关参数将训练测试数据集构建完成。构建路线:

1. 根据训练集数据量以及 Batch_Size 大小计算并导入相关参数。
2. 根据整体数据集的各个参数，开辟所需的全部空间。
3. 将整体数据集数组根据训练测试集数目划分成两个数组，包括训练特征数组和训练标签数组，测试特征数组和测试标签数组。
4. 再将训练特征数组和训练标签数组划分成若干个块大小的块训练特征数组和块训练标签数组。
5. 最后将数据喂入到相应的矩阵结构体当中。

神经网络初始化

1. 开辟神经网络所需空间并导入每层相关参数注意创建顺序不可变化。
   (1) 创建神经网络层空间
      <1> FCLayer
   (2) 导入每层参数
      <1> NeuronNum
      <2> AcitFuncNum
   (3) 创建神经网络运算空间
      <1> ActiMat
      <2> ActiMatPlus
      <3> SumMat
      <4> ActiFunDerivationMat
      <5> DeltaMat
      <6> OnehotMat（FCNN）
   (4) 创建神经网络参数空间(参数空间的大小不随输入神经网络样本大小而改变)
      <1> WeightMat
      <2> WeightBiasMat
      <3> NablaWbMat
2. 进行权值初始化
   (a) WeightMat
   (b) WeightBiasMat

神经网络前向传播求损失 神经网络向前传播将小批量输入样本导入神经网络输入层激活值矩阵，并将对应的标签数据进行独热编码，然后进行矩阵运算损失处理得到标量损失值。其中要注意小批量输入样本的数量不一定和已经开辟的神经网络有关矩阵空间的数量相一致，要做检查处理。

1. 小批量余数处理
2. 输入特征数据拷贝到激活值矩阵
3. 输入标签独热编码数据拷贝到神经网络独热编码矩阵中
4. 前向传播矩阵相乘，激活，扩展激活值矩阵
5. 预测值矩阵与真实值矩阵求损失，返回

神经网络反向传播求梯度 神经网络先经过一次正向传播得到损失之后，再求其损失对所有权值偏置的导数得到梯度，用到了反向传播算法。注意，每一次反向传播是建立

在一次正向传播求得损失的基础之上的。

1. 输出层反向传播，求得输出层的中间变量矩阵以及权值导数
2. 隐藏层反向传播，求得各隐藏层的中间变量矩阵以及权值导数

Adam算法构建 Adam算法使用了动量变量 v_t 和RMSProp算法中小批量随机梯度按元素平方的指数加权移动平均变量 s_t，并在时间步0将它们中每个元素初始化为0。
默认参数变量：$\beta_1 = 0.9, \beta_2 = 0.999, \eta = 10^{-3}, \epsilon = 10^{-8}$。

$$v_t = \beta_1 v_{t-1} + (1-\beta_1) g_t \tag{1}$$

$$s_t = \beta_2 s_{t-1} + (1-\beta_2) g_t \odot g_t \tag{2}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_1^t} \tag{3}$$

$$\hat{s}_t = \frac{s_t}{1-\beta_2^t} \tag{4}$$

$$g'_t = \frac{\eta \hat{v}_t}{\sqrt{\hat{s}_t} + \epsilon} \tag{5}$$

$$x_t = x_{t-1} - g'_t \tag{6}$$

1. 创建Adam参数结构体。
2. 初始化结构体参数。
   (a) 初始化参变量为默认值。
   (b) 创建变量所需空间。
   (c) 时间步设置为0。
3. 根据六个公式更新梯度。

1.4 FCNN应用

1.4.1 Minst 手写字体识别

1. 数据集构建。

   minst 手写字体识别数据集包括一共 70000 张灰度图片，每张图片 28×28 个像素，图片数据集包括 0∼9 十类。全体数据集是经过出最大值归一化后的数据集。

2. 神经网络搭建。

```
CompleteSampleNum: 70000
TrainSampleNum:    60000
TestSampleNum:  10000
SampleDimensionNum: 784
HiddenLayerNum:    2
WeightInitWayNum: 3
ClassificationNum: 10
LossFuncNum:    1
BatchSize:   100
XValArray:(length = 54880000)
YValArray:(length = 70000)
NeuronNumArray:
784   512   256   10
ActiFuncNumArray:(include output layer Acti-Function)
0  3  3  5
OptimizationMethod: Mini-Batch Stochastic gradient descent
```

1. BSGD 收敛性不佳，指的是时间以及准确度上都达不到要求；
2. 开发Adam优化算法，提高准确度提高了五个百分点，以及收敛时间都缩短了1.5倍。

3. 对稀疏矩阵的运算做优化，收敛时间有缩短了一倍。