神经网络实验报告

191300061 吴玉阳

摘要

本次实验我选择了自己构建一个具有一个隐藏层的神经网络结构，并且采用了除了 baseline 方法外的 **MSRA 初始化权重和偏执、MSE 损失函数、周期更新学习率、dropout 正则化方法**。

最终得到的最理想的准确率为 98.7152%, 方法为 **MSRA 初始化权重和偏执、MSE 损失函数、固定学习率和不正则化**，epoch 为 25，更多的 epoch 会将准确率固定在 94%左右。

框架代码

由于这次是用的数据是经典的 MNIST 数据集, 所以直接从 MNIST 库中导入数据并读取为数组，此时图像为 28*28 的矩阵，再将其转化为一维数组，在加载数据时已经自动将标签转化为 onehot 编码。代码如下:

```
mndata = MNIST("./data")
tr_images, tr_labels = mndata.load_training()
test_images, test_labels = mndata.load_testing()

### 图片转换 ###
for i in range(0, len(test_images)):
  test_images[i] = np.array(test_images[i], dtype="float") / 255

for i in range(0, len(tr_images)):
  tr_images[i] = np.array(tr_images[i], dtype="float") / 255
```

神经网络的构建主要在 train(X,Y,learning_rate=0.5,epoch=10)函数当中，激活

函数为 Relu 函数，通过对输入值进行求和并通过激活函数然后求出最大值和标签比较，在得出损失函数反向传播更新权重和偏置，代码如下：

```
def train(X,Y,learning_rate=0.5,epoch=10):

  ### 随机产生权重和偏置 ###
w = (2 * np.random.rand(10, 784) - 1) / 10
b = (2 * np.random.rand(10) - 1) / 10

  ### 用MSRA方法初始化权重和配置 ###
#w = (2 * np.random.normal(0,math.sqrt(2/784),size=(10,784))
#b = (2 * np.random.normal(0,math.sqrt(2/784),size=10) - 1)

accu=[]
epo=[]

for e in range(epoch+1):
  for n in range(len(X)):
  #for n in random.sample(range(len(X)),k=54000):#伪dropout
    img = X[n]
    cls = Y[n]

    ### 显示每个图的概率向量 ###
    resp = np.zeros(10, dtype=np.float32)
    for i in range(0,10):
      r = w[i] * img
      r = ReLU(np.sum(r) + b[i])
      resp[i] = r
  '''
  找到最大概率值；我们把剩下的数组值和最大替换概率为1
  '''
  resp_cls = np.argmax(resp)
  resp = np.zeros(10, dtype=np.float32)
  resp[resp_cls] = 1.0

  ### 求出数字 ###
  true_resp = np.zeros(10, dtype=np.float32)
  true_resp[cls] = 1.0
  ########################################
  error=MSE(true_resp,resp)
  '''
  找出误差并计算新的权重系数值和新的位移系数值
  '''

  error = resp - true_resp
  delta = error * ((resp >= 0) * np.ones(10))

  for i in range(0,10):
    w[i] -= learning_rate*np.dot(img, delta[i])
    b[i] -= learning_rate*delta[i]

Ac=Predict(test_images,test_labels,w,b)
accu.append(Ac)
epo.append(e)
#learning_rate=LR_Adjuster(0.5,e) #学习率调整
```

主要代码框架中的其他函数有：（均在代码中标注）

- 预测函数 PredictImage：用于在测试测验集中得出输入 Image 的分类结果
- 测验函数 Predict：将训练好的模型测试测验集
- 激活函数 Relu：激活函数
- 绘图函数 Plot：绘制训练过程中的准确率曲线图

主函数：获取数据并且进行模型训练，测验函数集成在训练函数中，这样可以在每进行一轮 epoch 就可以检验训练后的精度。

不同方法的性能对比

Baseline 方法

默认的方法为参数随机初始化 + 交叉熵 + 固定学习率 + 不正则化，这些

方法都很简单，下面直接给出对应的代码，在文件中，这些代码也给出了标注

参数随机初始化：

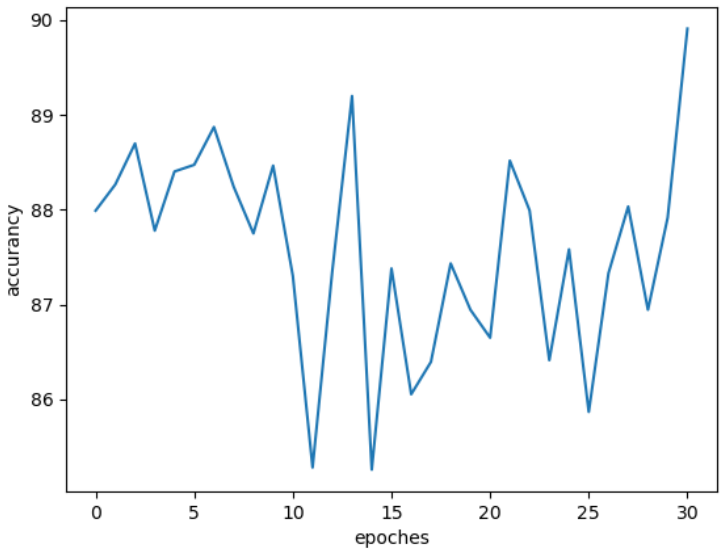
```
    ### 随机产生权重和偏置 ###
w = (2 * np.random.rand(10, 784) - 1) / 10
b = (2 * np.random.rand(10) - 1) / 10
```

交叉熵：（这里 delta 的作用是防止出现 np.log(0)导致后续计算无法进行而加上的微小值）

```
def CrossEntropy(y, t):
    delta = 1e-7
    return -np.sum(t * np.log(y + delta))
```

固定学习率设置为 0.6，epoch 为 30

最终准确率为 89.9093%，过程图如下所示。



MSRA 初始化：

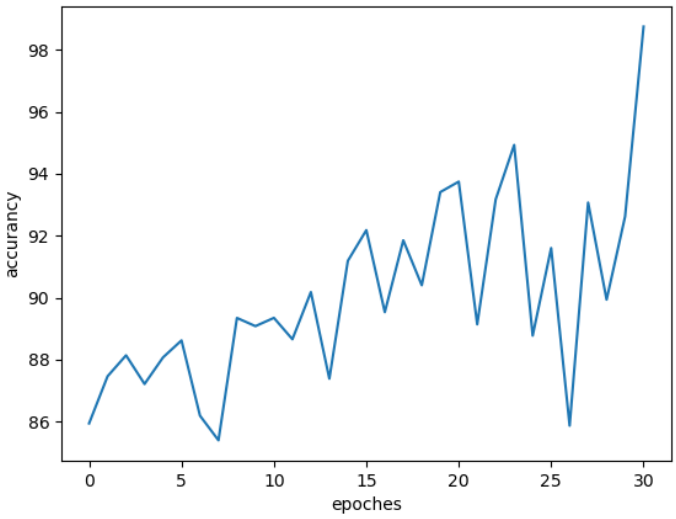
本次实验我选择了 MSRA 初始化方法，没有选择课上选择的 Xavier 方法是因为 Xavier 假设激活函数是线性的且均值为 0，而我选择的 Relu 激活函数显然

不适合这一点，因此选择了 MSRA 方法。在这个实验中，只需要考虑输入的个数，因此初始化是一个方差为$\sqrt{\frac{2}{F_{in}}}$的高斯分布。即：

$$\omega \sim G\left[0, \sqrt{\frac{2}{F_{in}}}\right]$$

代码和运算结果如下，最大的准确率为 98.7512%，但这个结论似乎具有偶然性，从图上可知一般准确率在 94%左右。但是这样的初始化方法收敛速度显著高于随机初始化方法

```
  ### 用MSRA方法初始化权重和配置 ###
#w = (2 * np.random.normal(0,math.sqrt(2/784),size=(10,784)) - 1) / 10
#b = (2 * np.random.normal(0,math.sqrt(2/784),size=10) - 1) / 10
```



MSE 损失函数方法

Baseline 方法是使用交叉熵损失函数，这个方法有利于多分类问题，相比之下 MSE 损失函数更有利于回归问题，但为了比对性能，我也作了对比。

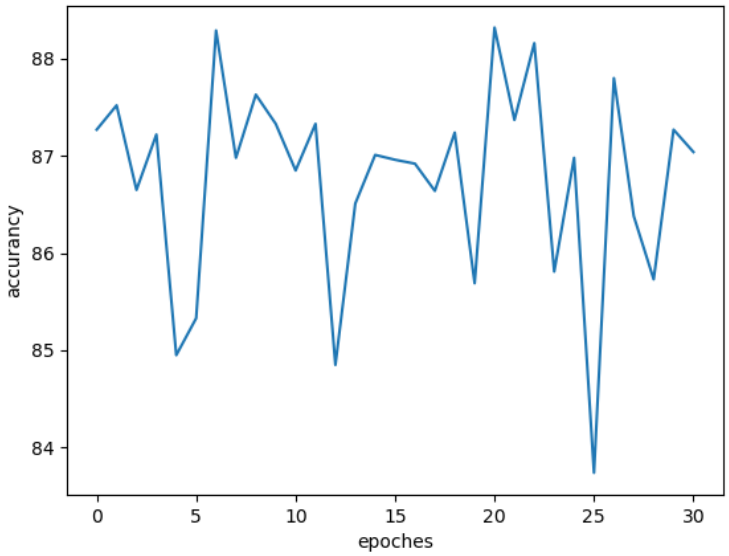
MSE 假设误差服从高斯分布下得到

$$l(y_{pred}) = \frac{1}{2}(y_{pred} - y)^2$$

一阶导数为$\frac{\partial l}{\partial y_{pred}} = y_{pred} - y$

代码和结果如下,可以看出收敛能力下降，最大准确率为 88.45%

```
def MSE(y_true, y_pred):
  return np.square(np.subtract(y_true, y_pred)).mean()
```

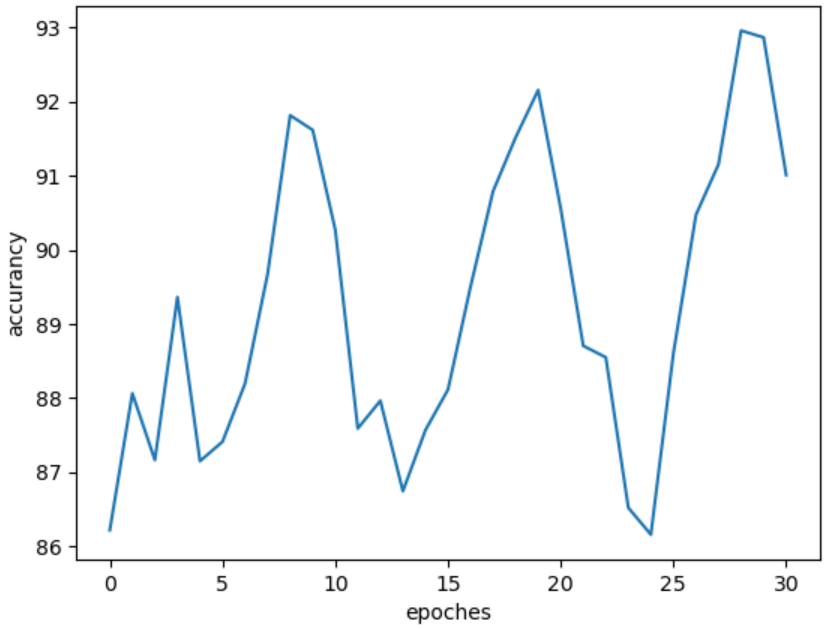


周期更新学习率

在前面的实验中，我发现部分情况下随着 epoch 增长，准确率有所下降，这可能是学习率设置过大导致，而有的时候又没有完全收敛，因此我决定使用周期更新学习率方法找出合适的学习率，学习率变化周期为 10epoch，从 0-1 按正弦函数变化，在每进行完一个 epoch 后更新，代码和结果如下:

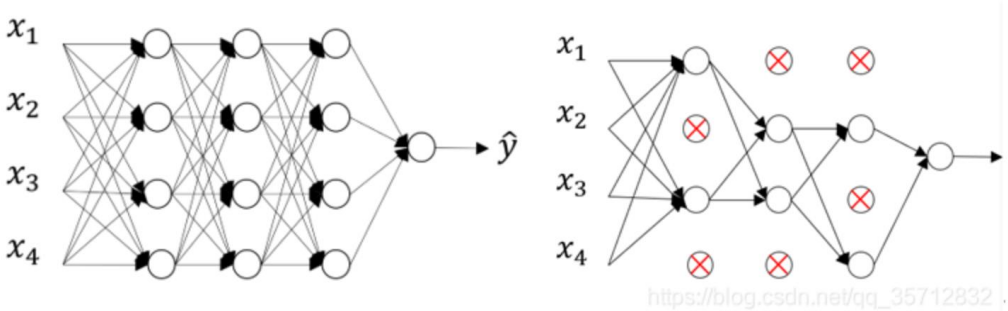
可以看出，代码准确率和学习率周期大致相同，最适合的学习率大致为 0.7 左右，最高的准确率为 92.9577%

```
def LR_Adjuster(l,e):
  return l+0.5*math.sin(0.2*e*math.pi)
```



Dropout 正则化

Dropout 提供了正则化一大类模型的方法，计算方便但功能强大。简单来说 Dropout 可以理解为在概率意义上随机删除神经网络中的节点，以此简化神经网络模型来防止过拟合的一种正则化方法，下图说明了 Dropout 的处理过程。



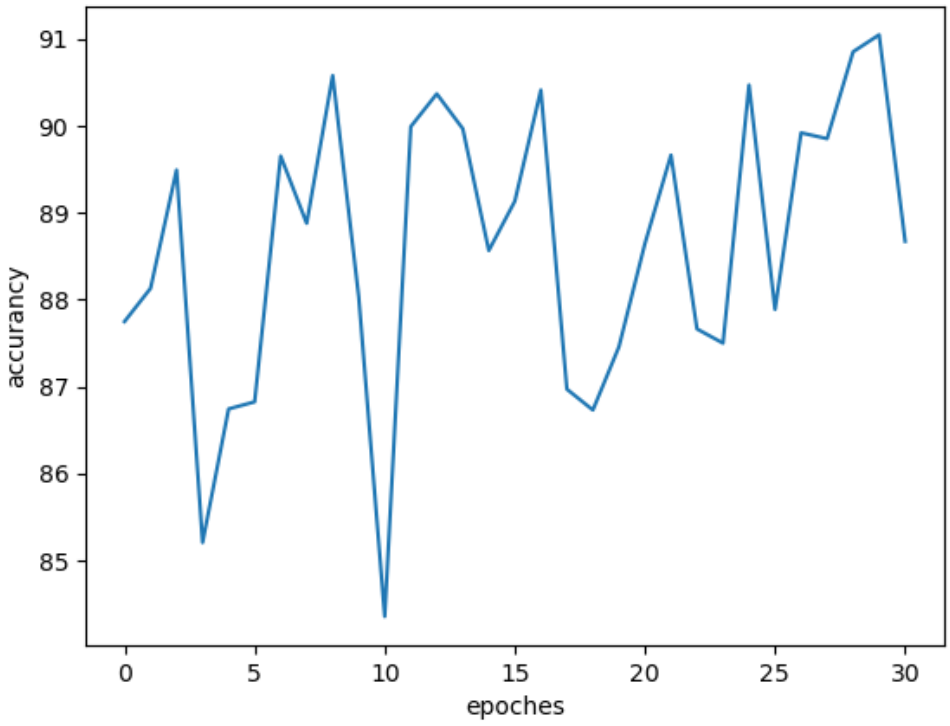
（资料参考自 https://blog.csdn.net/qq_35712832/article/details/114218032）

Dropout 会在每次迭代中随机关闭一些神经元,其中涉及到正则化参数 keep_prob,keep_prob 是一个随机数，在正向传播时，我们给每一个神经元赋予一个概率，通过比较 keep_prob 与神经元的随机概率大小来给出决策。当该概率小

于 keep_prob 时，我们关闭该神经元。做法是通过应用掩码来关闭正向传播过程中的某些神经元。对于反向传播，将用相同的掩码重新来关闭相同的神经元，进而进行导数的计算。

但由于能力问题我没能够实现这一功能，但是为了模拟该正则化的方法，我使用了每次随机抽取出部分输入值的方法来减少过拟合，代码和结果如下，最终结果为 91.0476%，如果能够实现 dropout 应该会更理想：

```
#for n in random.sample(range(len(X)),k=54000):#伪dropout正则化
```



总结

这次试验我感受到了从头到尾自己构建神经网络的过程，同时也感觉到了自己能力和知识上的缺乏，对最后的结果我自己感到还有提升空间。同时感谢助教老师的耐心阅读！