Report 4 Rebuild

2013012245 基科 31 白可

概述：

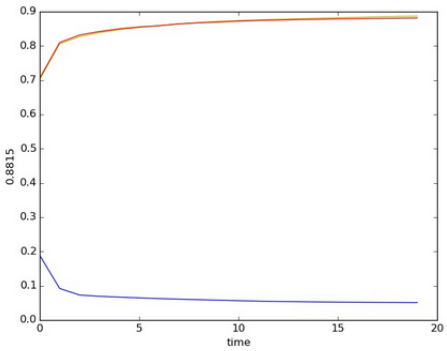
分为三个部分。
第一部分使用 MLP，比较 Relu 和 Sigmoid 函数区别（欧拉距离）
第二部分使用 MLP，比较单隐层/双隐层区别(softmax)
第三部分使用 CNN + Relu
大体上是模仿 Theano 教程中的代码，并对其中部分内容修改后方便适合自己的测试和比较。

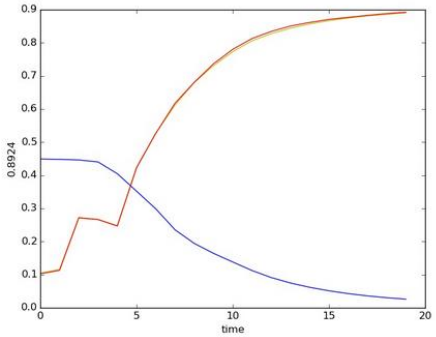
实验结果：

一、对比欧氏距离下的 Relu 与 sigmoid

将整个集合分割。以 100 为一个 bitch 进行训练。这里与之前实验有些不同。之前是先训练完，而后统一测试，这里每回测试都使用了“测试集”的数据。

·以下的图片，红色线表示“测试集正确率”，蓝色线表示训练集的“cost”，绿色线表示训练集的正确率





根据图像可以看出，Relu 函数依然保持的较好性质

似乎图中只有“两条线”但实际上是有三条线，绿线与红线几乎完全重合，表明训练集的正确率和测试集的正确率基本相同。

开始，Sigmoid 连续多次没有收敛。尝试了不同大小比例的权值。但是，几乎都不收敛。于是我改变了 Sigmoid 的权值矩阵的初值，Relu 为随机*0.01，大约在-0.01——0.01 之间浮动，我将其调整到-0.04——0.04 之间后，才逐渐收敛，且收敛速度较慢。

后来考虑到该模型在最开始初始化时已经将每个像素点进行了归一化处理，值的波动在 0——1 之间，所以权值矩阵权值的设置不必过小。

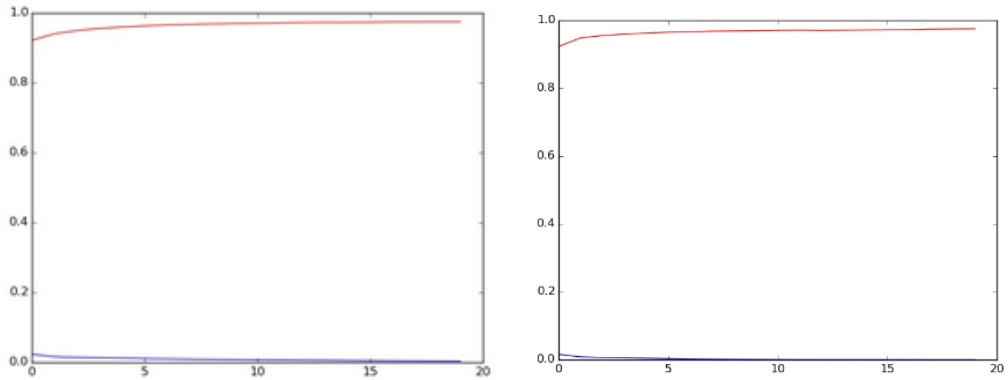
根据右侧图片可以看到，在正确率在不断增大。

类别	权值比	正确率（从前）	损失大小
Relu	– 0.05-0.02-0.3	97.62(98.01)	0.068
Relu	– 0.02-0.05-0.1	97.04(98.16)	0.0097
Relu	- 0.05-0.1-0.1	97.97(98.32)	0.0047
Sigmoid	-0.05-0.1-0.1	局部极小值	
Sigmoid	-0.02-0.05-0.1	局部极小值	
Sigmoid(0.04)	-0.1-0.2-0.3	91.48	0.010

这次实验让我意识到。测试两个不同集效率的时候，要先从比较弱的一方出发，因为它满足的条件要更强，这样才会有比较的可能性。

二、两层网络与三层网络的比较

两层网络



左侧是两层网络，右侧是三层网络
左侧步长 0.1-0.1 右侧步长 0.05-0.2

类别	权值比	正确率	损失大小
两层	0.1-0.1	97.49	0.003
两层	0.05-0.2	97.53	0.09
三层	0.05-0.1-0.2	97.54	0.05

两层与三层差别不大。具体已经在之前讨论过。这个正确率没有之前高，主要是因为在计算Δw时，没有像从前一样加入 momentum 与 weightdecay，这个之后会有讨论。

三、CNN

我在 CNN 上花的时间比较久。因此重点说一下 CNN 网络。

根据数据我们能够发现。实验一与实验二的效果都没有之前用 MATLAB 写的好。因此在第三个网络上，我重点测试了

一、不同的权值调整方法得到的最后对结果的影响。

二、不同的 dropout 比例对结果的影响。

但是实际上感觉这些得到的结论并没太大用，基于不同的模型，能够取到最好结果的这些参数是在不断发生变化的。

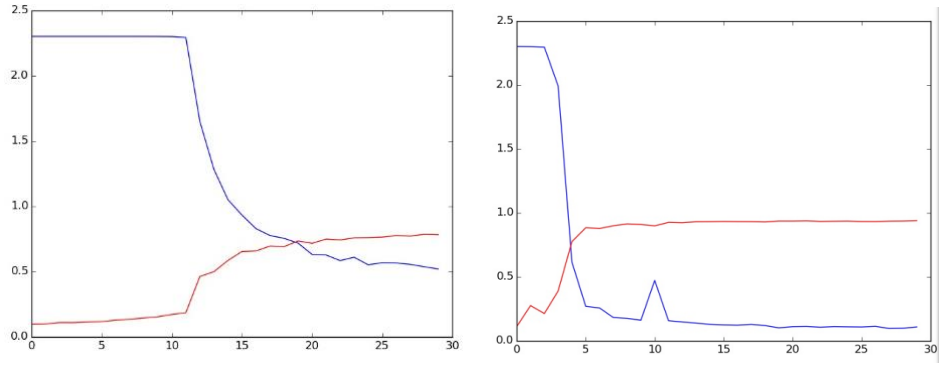
Part 1

Dropout 的影响：

以下数据的 weightchange 均加入了 momentum.

步长均取为 0.2，小于 0.1 时容易出现局部极小值现象

看网上的教程，推荐取 0.5，但是我取了 0.5 效果很不好



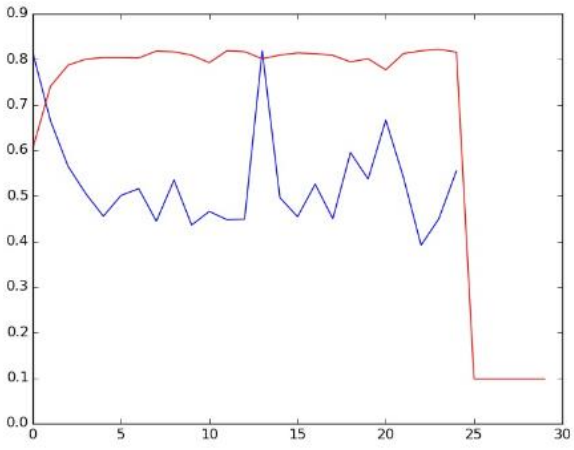
左图将 60%的元素置为零，可以看到，正确率较低，损失较大

右图将 5%元素置零，准确率较高，损失较小

挑取几组比较有典型意义的数据放在下面

Dropout	Rightrate
0.4	78.32%
0.95	94.01%
0.95	9.8%
1	87.25%

可以看到 drop 5% 时，出现了局部极小值。尽管步长和其他是相同的。我不知道为什么会出现这种结果——突然掉进了一个局部极小值。因此，对于 dropout 可能会出现的这种情况要做好心理准备。甚至会出现这样的结果：



正确率在最后突然掉到了“均值”，我猜测是因为每回减去的Δw使得权值矩阵W在某一时刻突然全都变成了 0，且按照我们的规定：越到最后变化越小。因此我们就很难跳出这个陷阱了。

Part2

三种不同的方法：

SGD

$$w(t) = w(t-1) - \epsilon * \triangle w(t)$$

Momentum

$$v(t) = \alpha v(t-1) - \epsilon \frac{\partial E}{\partial w}(t)$$

$$\triangle w(t) = v(t)$$

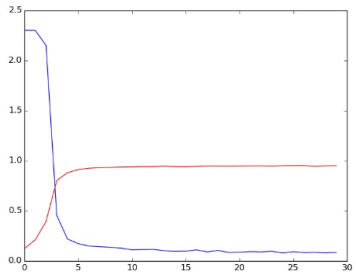
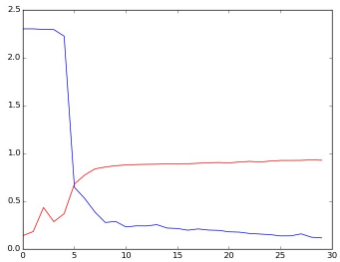
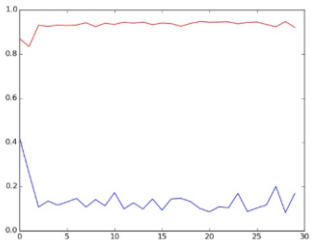
RMSPROP

$$MeanSquare(w,t) = 0.9MeansSquare(w,t-1) + 0.1\frac{\partial E}{\partial w}(t)^2$$

$$\triangle w(t) = \epsilon \frac{\partial E}{\partial w}(t) / (\sqrt{MeanSquare(w,t)} + \mu)$$

参考资料中提到，这三种方法，后者比前者好。但我在试验中证实了后面两个的关系，但是对于前面两个关系却不甚明显。我猜测有可能与 dropout 有关。

Method	dropout	rightrate
Sgd	0.9	95.15%
Sgd	0.5	85.85%
Momentum	0.5	84.28%
Momentum	0.9	92.82%
Rmsrop	0.5	9.8%
	(主要是因为步长原因，Rmsrop 对于步长长度设置要求较短)	
Rmsrop	0.8	92.09%

从左到右以此是这三种方法在同一种 dropout 下（0.9）的结果。其中前两者对于步长的要求比较大，大约需要>0.1 但是第三种必须在 0.1 以下。

可以看到，第三张图虽然看似“波动”较大，但是它的坐标轴取得参考坐标较小。

总结

之前本以为该作业做起来很快。但是自己基本功不扎实，先是解决了虚拟机的问题，然

后 Python 掌握的还不熟练，在开始数据的导入上花了一些时间，对于符号语言没有很好理解，在写 Momentum decay 时，对于 updates 的理解出现了偏差，绕了很大的弯路。

Anyway,学到了很多，还需很大的努力。