MNIST 手写数字数据集的图像分类问题

本次课程报告我们采用选题 1:用 CNN 网络模型在 MNIST 上训练并测试。我们采用 ResNet 残差神经网络。我们采用 MNIST 数据集。MNIST 数据集为手写数字数据集，包含 0~9 的手写数字图片。数据集分为两部分，分别含有 60000 张训练图片和 10000 张测试图片。每一张图片包含大小为 28*28 像素点。MNIST 数据集把代表一张图片的二维数据转成一个向量，长度为 28*28=784。因此在 MNIST 的训练数据集中，MNIST.train.images 是 60000 个 28*28 的向量。我们使用多层卷积神经网络，然后再使用全连接层进行训练。使用多层卷积的原因是，一层卷积学到的特征往往是局部的，层数越高，高层的学到的特征的感受野就越大，学到的特征就越全局化。我们使用优化器提升模型的性能，提高准确率。然后作出不同优化器下 loss 图和 ACC 图形，得到混淆矩阵，并且将混淆矩阵进行可视化。

一、训练模型

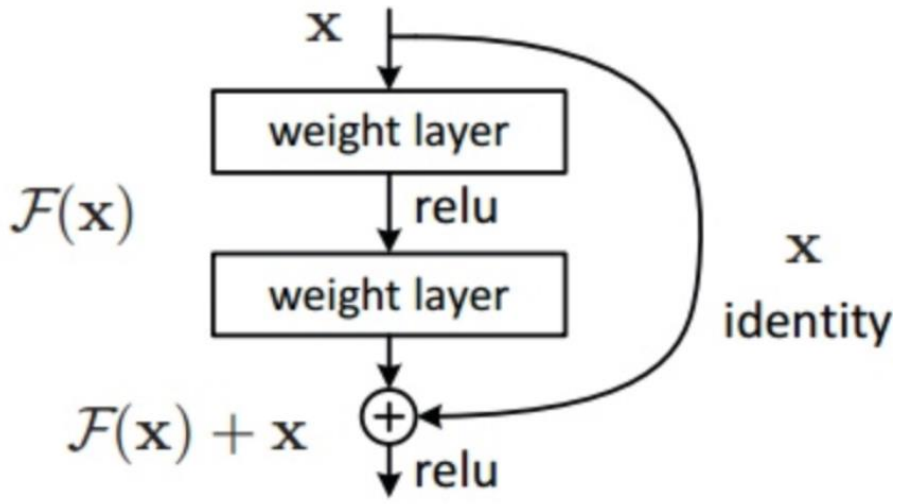
1.网络结构

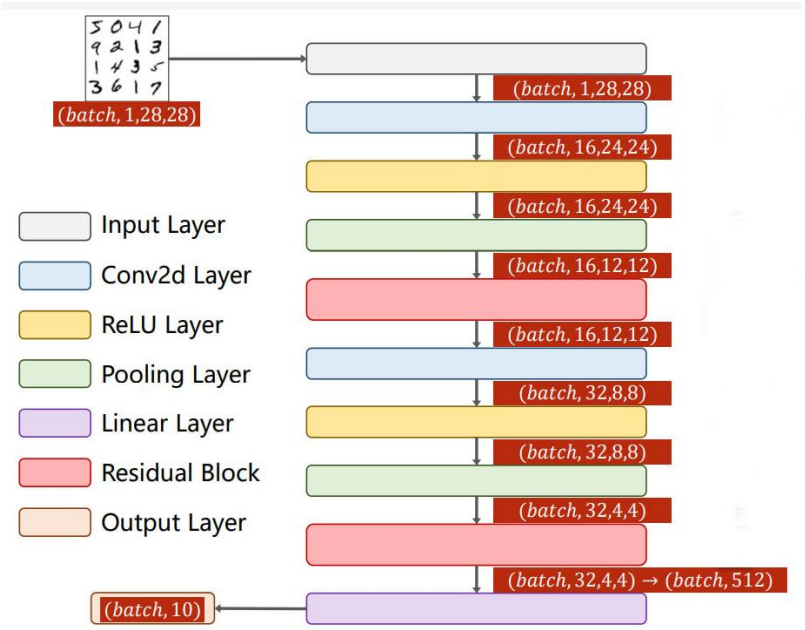
残差模块单元

初始化：我们设深度网络中某隐含层为

H（x）-x→F（x），

如果可以假设多个非线性层组合可以近似于一个复杂函数，那么也同样可以假设隐含层的残差近似于某个复杂函数。即那么我们可以将隐含层表示为 H（x）=F（x）+ x。这样一来，我们就可以得到一种全新的残差结构单元，如下图所示。可以看到，残差单元的输出由多个卷积层级联的，输出和输入元素间相加(保证卷积层输出和输入元素维度相同)，再经过 ReLU 激活后得到。将这种结构级联起来，就得到了残差模块单元。





可以注意到残差网络有这样几个特点：

1. 网络较瘦，控制了参数数量；
2. 存在明显层级，特征图个数逐层递进，保证输出特征表达能力
3. 使用了较少的池化层，大量使用下采样，提高传播效率；
4. 没有使用 Dropout，利用 BN 和全局平均池化进行正则化，加快了训练速度。

2.训练过程

下面我们开始定义训练过程。先设置一个初始值为 0 的 loss。batch_idx 表示进行 batch 迭代的次数。用列举的方法将 train_loader 的数据提取出来，使用 GPU 进行运算。

对于每一次的迭代，将输入 x 和相应的目标 y 从数据中拿出来，将输入和目标 y 都放到 GPU 上面，用优化模块进行梯度归零。然后我们将输入值代入模型，输出 y 值。求出输出值与目标值的损失，然后通过反向传播更新参数。每进行一次迭代，我们把损失值进行累加，求出总的损失值。

3.测试过程

下面我们定义 test 模块。正确的数量初始为 0，总数为 0。用 with 方式让 torch 不计算梯度，for 循环 data 在测试加载器中，输入和目标送到数据里面，将输入和目标 y 都放到 GPU 上面，将输入代入模型，求输出，用 torch 中的 max 函数，沿着输出的数据的维度为 1 方式，找到最大值和最大值的下标，_是最大值，predicted 是最大值的下标。总数等于标签的 size 取第 0 个元素相加求得总数，正确是预测和标签相等标量求和。

二:优化器

我们分别使用了四种不同的优化器：SGD，SGDM，adagrad，ADAM

1.SGD:基本的 mini-batch SGD 优化算法

基本的 mini-batch SGD 优化算法。其主要思想就是每次只拿总训练集的一小部分来训练，对于我们的 60000 个样本，每次拿的样本数量为 batch_size，来计算 loss，更新参数。完成整个样本集的训练，为一轮（epoch）。由于每次更新用了多个样本来计算 loss，就使得 loss 的计算和参数的更新更加具有代表性。不像原始 SGD 很容易被某一个样本给带偏 。loss 的下降更加稳定，同时小批量的计算，也减少了计算资源的占用。

参数的更新公式：

对于 mini-batch，其 loss-function 表示为：

$$loss_{batch} = \frac{1}{2k}\sum_{i=1}^{k}(y_{p,i} - y_i)^2$$

对于要更新的参数 a 与 b，分别计算其梯度：

$$\frac{\partial loss_{batch}}{\partial a} = \frac{1}{k}\sum_{i=1}^{k}(ax_i + b - y_i)x_i$$

$$\frac{\partial loss_{batch}}{\partial b} = \frac{1}{k}\sum_{i=1}^{k}(ax_i + b - y_i)$$

然后用梯度下降的方法更新 a 与 b

$$b_{new} = b - \alpha\nabla b$$

$$a_{new} = a - \alpha\nabla a$$

SGD 最大的缺点是下降速度慢，而且可能会在沟壑的两边持续震荡，停留在一个局部最优点。

2.SGDM:

SGD+ Momentum，加入了一个惯性，在 SGD 的基础上加入了一阶动量

为了抑制 SGD 的震荡，SGDM 认为梯度下降过程可以加入惯性。下坡的时候，如果发现是陡坡，那就利用惯性跑的快一些。SGDM 全称是 SGD with momentum，在 SGD 基础上引入了一阶动量：

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$

0.9*过去的动量+0.1*当前的动量

一阶动量是各个时刻梯度方向的指数移动平均值，约等于最近 1/(1-β1) 个时刻的梯度向量和的平均值。

也就是说，t 时刻的下降方向，不仅由当前点的梯度方向决定，而且由此前累积的下降方向决定。β1 的经验值为 0.9，这就意味着下降方向主要是此前累积的下降方向，并略微偏向当前时刻的下降方向。

3.Adagrad:自适应步长,历史梯度平方和的平方根

SGD 及其变种以同样的学习率更新每个参数，但深度神经网络往往包含大量的参数，这些参数并不是总会用得到。对于经常更新的参数，我们已经积累了大量关于它的知识，不希望被单个样本影响太大，希望学习速率慢一些；对于偶尔更新的参数，我们了解的信息太少，希望能从每个偶然出现的样本身上多学一些，即学习速率大一些。

$$V_t = \sum_{\tau=1}^{t} g_\tau^2$$

$$\eta_t = \alpha \cdot m_t / \sqrt{V_t}$$

即学习率为α/根号下 vt

4.ADAM: 结合了 AdaGrad 和 RMSProp 算法最优的性能

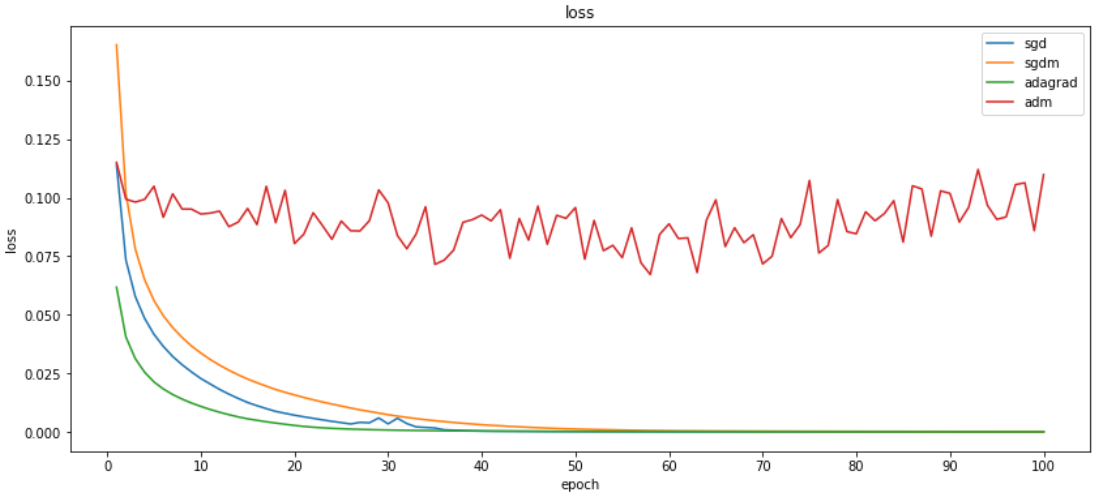
谈到这里，Adam 和 Nadam 的出现就很自然而然了——它们是前述方法的集大成者。我们看到，SGD-M 在 SGD 基础上增加了一阶动量，AdaGrad 和 AdaDelta 在 SGD 基础上增加了二阶动量。把一阶动量和二阶动量都用起来，就是 Adam 了——Adaptive + Momentum。

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$

这里是 Adadelta

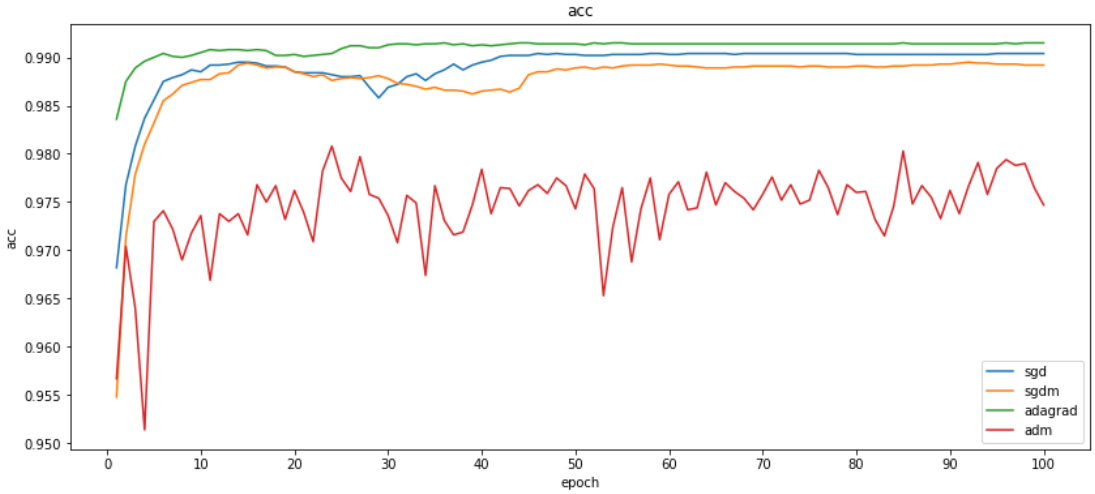
$$V_t = \beta_2 * V_{t-1} + (1 - \beta_2) g_t^2$$

三:实验结果



看出收敛最快的是 adagrad，其次是 SGD，然后是 SGDMM，而 ADAM 一直没有有效的收敛。

四种不同的优化器有四种不同的 ACC 图



由上图可以看出，精确度最高的是 adagrad，其次是 SGD，其次是 SGDM，其次是 ADAM。后期 Adam 的学习率太低，影响了有效的收敛。

混淆矩阵也称误差矩阵，是表示精度评价的一种标准格式，用 n 行 n 列的矩阵形式来表示。我们的混淆矩阵是一个 10*10 的矩阵，其中的每个元素 k(i,j)表示属于 i 的元素被分入 j 类的个数。# 是吗

我们输出了混淆矩阵，可以看出 ADAM 的分类效果非常好

1.结果

Adam 的准确率，召回率，f1，support

support

	precision	recall	f1-score	support
1	0.97	0.99	0.98	980
2	0.98	0.99	0.98	1135
3	0.98	0.97	0.97	1032
4	0.98	0.97	0.97	1010
5	0.98	0.98	0.98	982
6	0.94	0.98	0.96	892
7	0.99	0.97	0.98	958
8	0.98	0.97	0.97	1028
9	0.97	0.97	0.97	974
10	0.98	0.95	0.97	1009
accuracy			0.97	10000
macro avg	0.97	0.97	0.97	10000
weighted avg	0.97	0.97	0.97	10000