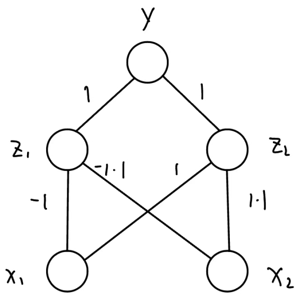
计算机视觉-作业2-彭程-2020011075

1.

1.1

满足要求的权重如下图所示：



1.2

网络计算如下：

$$
\begin{aligned}
& z_1' = -x_1 - x_2 \\
& z_2' = x_1 + x_2 \\
& z_1 = \operatorname{ReLU}\left(z_1'\right) \\
& z_2 = \operatorname{ReLU}\left(z_2'\right) \\
& y = z_1 + z_2 \\
& y_0 = \left|x_1 + 1.1 x_2\right| \\
& \Delta = \left(y - y_0\right)^2
\end{aligned}
$$

代入数据：

x_1	x_2	z_1	z_2	y	Δ
1	1	0	2	2	0.01
1	0	0	1	1	0
1	-1	0	0	0	0.01
0	1	0	1	1	0.01
0	0	0	0	0	0
0	-1	1	0	1	0.01
-1	1	0	0	0	0.01
-1	0	1	0	1	0
-1	-1	2	0	2	0.01

1.3

权重的梯度计算方式如下：

$$\frac{\partial E_n}{\partial w_{3i}} = \frac{\partial E_n}{\partial y} \cdot z_i = 2 \cdot z_i \cdot (y - y_0)$$
$$\frac{\partial E_n}{\partial w_{ij}} = x_j \frac{\partial E_n}{\partial z_i'} = x_j \cdot \frac{\partial E_n}{\partial y} \cdot \frac{\partial y}{\partial z_i'} = 2 \cdot x_j \cdot (y - y_0) \cdot w_{3i} \cdot h'(z_i')$$

如下编程求解可以计算到梯度：

```python
def compute_all(x,W12,W3):
    z_primes = np.dot(W12, x)
    z = np.maximum(z_primes, 0)
    y = np.dot(W3, z)
    y_0 = abs(x[0] + 1.1 * x[1])
    delta = (y - y_0) ** 2

    dW3 = 2 * z * (y - y_0)
    dW12 = np.ones_like(W12)
    dW12 = ((dW12 * x * 2 * (y - y_0)).T * W3 * (z_primes > 0)).T

    return dW3,dW12
```

用上述公式进行一次计算和迭代，得到如下结果（lr = 0.01）：

$$\text{loss}_{\text{before}} = 0.01$$

$$\mathbf{dw}_{12} = \begin{bmatrix} 0.02222222 & 0.04444444 \\ -0.02222222 & -0.04444444 \end{bmatrix}$$

$$\mathbf{dw}_3 = \begin{bmatrix} -0.06666667 & -0.06666667 \end{bmatrix}$$

$$\text{loss}_{\text{after}} = 0.0096$$

用梯度下降的方法迭代更新10000次，最后得到的权重如下：

$$\mathbf{W}_{12} = \begin{bmatrix} -0.96744489 & -1.06418938 \\ 0.96744489 & 1.06418938 \end{bmatrix}$$

$$\mathbf{W}_3 = \begin{bmatrix} 1.03365061 & 1.03365061 \end{bmatrix}$$

1.4

选取step size = 0.01，更新一次后，loss从$\text{loss}_{\text{before}} = 0.01$变为$\text{loss}_{\text{after}} = 0.00639$。更新1000次以后几乎趋于0。

1.5

如果初始化为图c的权重，重复上述的计算流程，计算结果如下：

$$\mathbf{W}_{12} = \begin{bmatrix} -99.99982504 & -100.00017501 \\ 99.99982504 & 100.00017501 \end{bmatrix}$$

$$\mathbf{W}_3 = \begin{bmatrix} 0.0105 & 0.0105 \end{bmatrix}$$

$$\text{loss}_{\text{before}} = 0.006666666666666678$$

$$\text{loss}_{\text{after}} = 0.0033330883596 20334$$

1.6

- 如1.5中所展示的，error大概减小了一半
- 对于迭代次数，可以发现，在这种初始化条件下，迭代次数增加，loss并没有显著增加，同时由于初始化值较大，还容易出现loss成为nan越界。所以此处我仅选择了10000迭代

1.7

对于有batch_norm的网络，有（参考：[笔记: Batch Normalization及其反向传播 - 知乎 (zhihu.com)]()）：

$$\frac{\partial L}{\partial x_i} = \frac{1}{N\cdot\sqrt{\sigma^2}}[N\frac{\partial L}{\partial \hat{x}_i} - \sum_{j=1}^{N}\frac{\partial L}{\partial \widehat{x}_j} - \widehat{x}_i\sum_{j=1}^{N}\frac{\partial L}{\partial \widehat{x}_j}\cdot\widehat{x}_j]$$

在本题中按照：线性-归一化-激活-线性依次得到：$x \to z' \to \hat{z}' \to z \to y$
所以有：

$$\frac{\partial E_n}{\partial w_{3i}} = \frac{\partial E_n}{\partial y}\cdot z_i = 2\cdot z_i\cdot(y-y_0)$$

$$\partial E_n \partial w_{ij} \quad = x_j\frac{\partial E_n}{\partial z_i'} = x_j\cdot\frac{\partial E_n}{\partial y}\cdot\frac{\partial y}{\partial z_i'} = 2\cdot x_j\cdot(y-y_0)\cdot w_{3i}\cdot\frac{1}{N\cdot\sqrt{\sigma^2}}[N\frac{\partial y}{\partial \hat{z}_i'} - \textstyle\sum_{j=1}^{N}\displaystyle\frac{\partial y}{\partial \hat{z}_j'} -$$

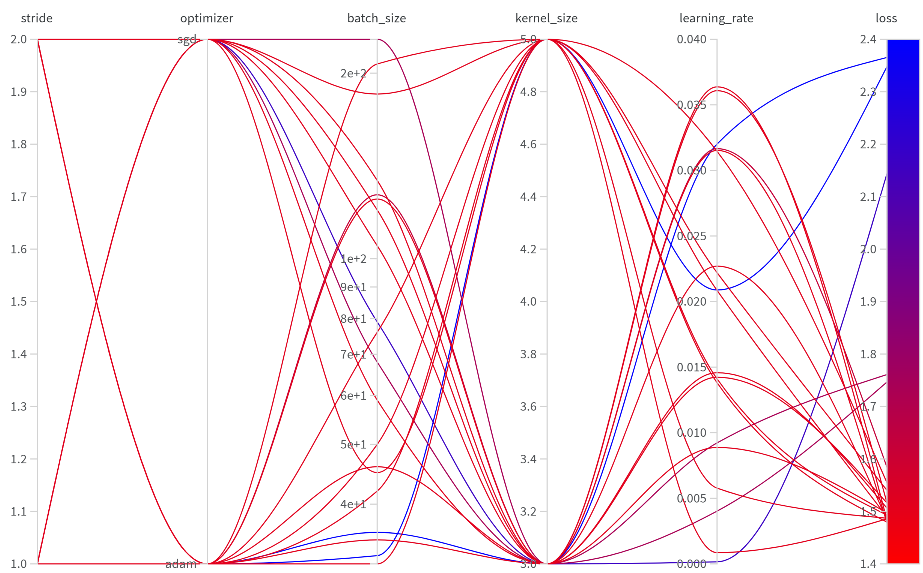
BN层的作用是把一个batch内的所有数据，从不规范的分布拉到正态分布。这样做的好处是使得数据能够分布在激活函数的敏感区域，敏感区域即为梯度较大的区域，因此在反向传播的时候能够较快反馈误差传播。

在本题中，由于使用的ReLU激活函数，同时数据的batch过小，所以BN层作用不大。

2

对于此题，我采用pytorch实现cnn进行mnist分类，代码见`code/mnist.py`。

利用wandb的sweep功能对各种超参数组合进行测试如下：

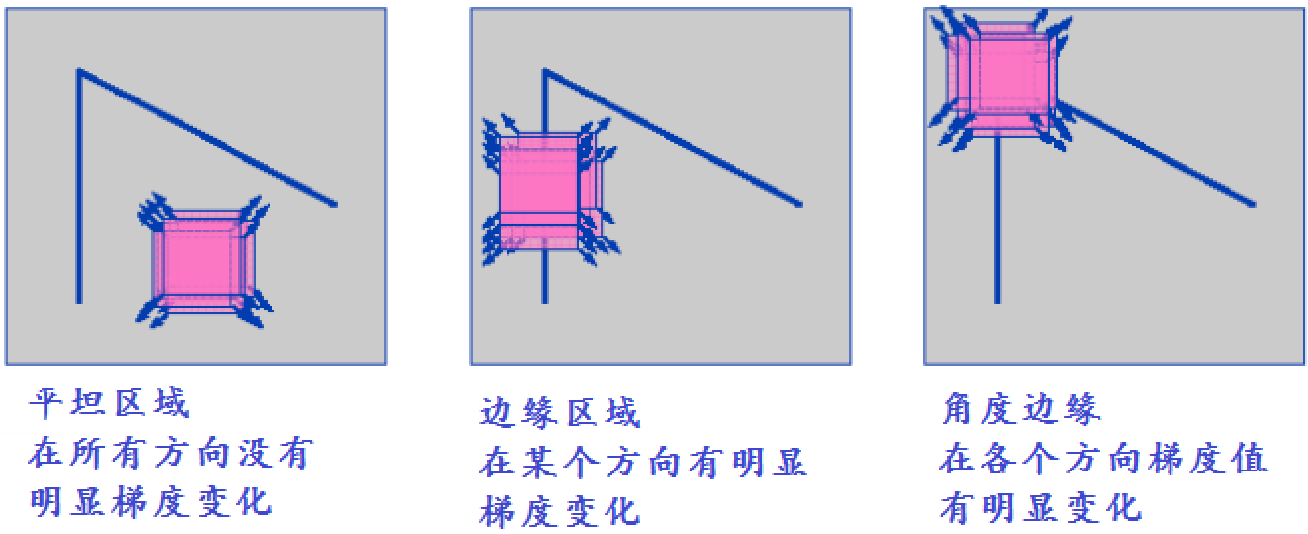


表现最好的参数组合为：

- stride: 1
- optimizer: sgd
- batch_size: 45
- kernel size: 5
- learning_rate: 0.01381
- loss: 1.48

根据实验结果可以看出，各参数对实验结果影响存在着极大的不确定性，并没有普遍的规律。

3.



计算公式为：

$$E(u,v)=\sum_{x,y}w(x,y)[I(x+u,y+v)-I(x,y)]^2$$

由泰勒展开：

$$f(x+u,y+v)\approx f(x,y)+uf_x(x,y)+vf_y(x,y)$$

代入公式有：

$$
\begin{aligned}
E(u,v) &= \sum_{(x,y)} w(x,y)[I(x+u,y+v)-I(x,y)]^2 \\
&\approx \sum_{(x,y)} w(x,y)\left[I(x,y)+\frac{\partial I}{\partial x}(x,y)u+\frac{\partial I}{\partial y}(x,y)v-I(x,y)\right]^2 \text{(一阶泰勒展开)} \\
&\approx \sum_{(x,y)} w(x,y)\left[\frac{\partial I}{\partial x}(x,y)u+\frac{\partial I}{\partial y}(x,y)v\right]^2 \text{(消除重复项)} \\
&= \sum_{x,y} w(x,y)\left(u^2 f_x^2(x,y)+2uvf_x(x,y)f_y(x,y)+v^2 f_y^2(x,y)\right) \text{(二阶泰勒展开)} \\
&= \sum_{x,y} w(x,y)\left(u^2 I_x^2+2uvI_xI_y+v^2I_y^2\right) \text{(简化)} \\
&= \begin{bmatrix} u & v \end{bmatrix}\left(\sum \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}\right)\begin{bmatrix} u \\ v \end{bmatrix} \\
&= \begin{bmatrix} u \\ v \end{bmatrix}^T H \begin{bmatrix} u \\ v \end{bmatrix}
\end{aligned}
$$

其中:

$$
H = \begin{bmatrix} \sum_{(x,y)} w(x,y)(\frac{\partial I}{\partial x}(x,y))^2 & \sum_{(x,y)} w(x,y)(\frac{\partial I}{\partial x}(x,y))\frac{\partial I}{\partial y}(x,y)) \\ \sum_{(x,y)} w(x,y)(\frac{\partial I}{\partial x}(x,y))\frac{\partial I}{\partial y}(x,y)) & \sum_{(x,y)} w(x,y)(\frac{\partial I}{\partial y}(x,y))^2 \end{bmatrix}
$$

为harris矩阵，由两个方向上的梯度构成。

对于H矩阵，可以分析其特征值;

- 两个特征值反映相互垂直方向上的变化情况，分别代表变化最快和最慢的方向，特征值大变化快，特征值小变化慢
- λ1 ≈ λ2 ≈ 0， 两个方向上变化都很小，兴趣点位于光滑区域
- λ1 > 0 , λ2 ≈ 0， 一个方向变化快，一个方向变化慢，兴趣点位于边缘区域
- λ1 , λ2 > 0， 两个方向变化都很快，兴趣点位于角点区域（容易判断）
  编写代码进行实现，见 `code/harris.py`
  检测结果如下图所示，其中蓝色为检测到的角点: