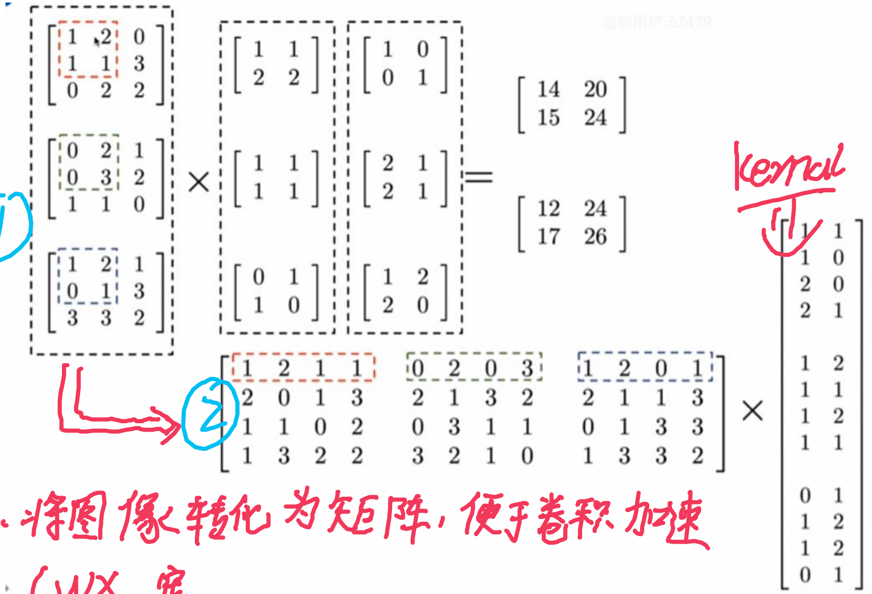
深蓝学院
shenlanxueyuan.com

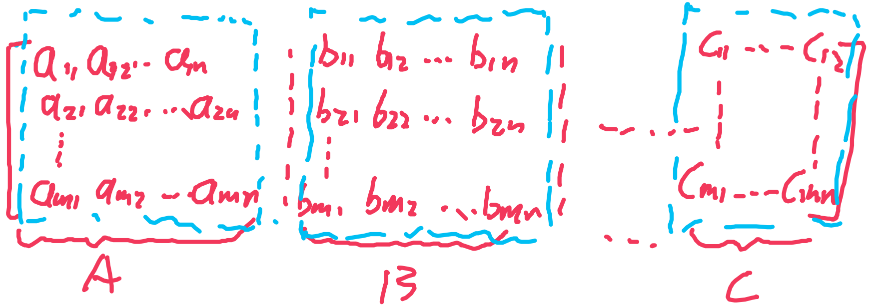
第三章作业分享

主讲人 潘光帅

代码分享

深蓝学院
shenlanxueyuan.com





1. 将图像转化为矩阵，便于卷积加速

图像 { wx 宽, hx 高, cx 深 }

卷积核：ksize, ksize ⇒ 特征图尺寸: feature_w = (wx - ksize)//step + 1

① 共有 cx 个矩阵块.

② 每个矩阵块 raw 个数为：feature map 中元素个数

③ 每个矩阵块 col 个数为：feature map 中单个元素所对应感受野中元数个数、(卷积核元素个数).

代码分享


```
for deep in range(cx):
    temp=np.zeros((feature_w * feature_w, ksize * ksize))
    num=0;
    for i in range(feature_w):
        for j in range(feature_w):
            temp[num]=x[i*step:i*step+ksize,j*step:j*step+ksize,deep].reshape(-1
            num=num+1
    image_col[:,deep*ksize * ksize:(deep+1)*ksize * ksize]=temp
return image_col
```

⇒单个代码块的尺寸

→水平移动总次数

→垂直移动总次数

⇒感受野中的元素重新排列

步长

行排列

代码分享

训练过程


```python
def train(batch=32, lr=0.01, epochs=10):
    # Mnist手写数字集
    dataset_path = "./datasets/mnist"
    train_data = torchvision.datasets.MNIST(root=dataset_path, train=True, download=False)
    train_data.data = train_data.data.numpy()  # [60000,28,28]
    train_data.targets = train_data.targets.numpy()  # [60000]
    train_data.data = train_data.data.reshape(60000, 28, 28, 1) / 255.  # 输入向量处理
    train_data.targets = onehot(train_data.targets, 60000)  # 标签one-hot处理 (60000, 10)

    # [28,28] 卷积 6x[5,5] -> 6x[24,24]
    conv1 = Conv(kernel_shape=(5, 5, 1, 6))
    relu1 = Relu()
    # 6x[24,24] -> 6x[12,12]
    pool1 = Pool()
    # 6x[12,12] 卷积 16x(6x[12,12]) -> 16x[8,8]
    conv2 = Conv(kernel_shape=(5, 5, 6, 16))  # 8x8x16
    relu2 = Relu()
    # 16x[8,8] -> 16x[4,4]
    pool2 = Pool()

    # 在这里可以尝试增加网络的深度，再实例化conv3和pool3，记得后面的前向传播过程
    # 和反向传播过程也要有对应的过程
    nn = Linear(256, 10)
    softmax = Softmax()
    for epoch in range(epochs):
        for i in range(0, 60000, batch):
            X = train_data.data[i:i + batch]
            Y = train_data.targets[i:i + batch]
            # 前向传播过程
            predict = conv1.forward(X)
            predict = relu1.forward(predict)
            predict = pool1.forward(predict)
            predict = conv2.forward(predict)
            predict = relu2.forward(predict)
            predict = pool2.forward(predict)
            predict = predict.reshape(batch, -1)
            predict = nn.forward(predict)
            # 误差计算
            loss, delta = softmax.cal_loss(predict, Y)
            # 反向传播过程
            delta = nn.backward(delta, lr)
            delta = delta.reshape(batch, 4, 4, 16)
            delta = pool2.backward(delta)
            delta = relu2.backward(delta)
            delta = conv2.backward(delta, lr)
            delta = pool1.backward(delta)
            delta = relu1.backward(delta)
            conv1.backward(delta, lr)
```

10回合.

前向

conv1 (5,5,1,6)
↓
Pool1
↓
conv2 (5,5,6,16)
↓
Pool2
↓
Linear

反向

conv1 backward
↑
Pool1.backward
↑
conv2.backward
↑
Pool2.backward
↑
Linear.backward

前向: $z^{(l)} = w^{(l)}a^{(l-1)} + b^{(l)}$

代码分享

链接层（前向传播）

深蓝学院 shenlanxueyuan.com

```
## 全连接层
class Linear(object):

    def __init__(self, inChannel, outChannel):
        scale = np.sqrt(inChannel / 2)
        self.W = np.random.standard_normal((inChannel, outChannel)) / sc
        self.b = np.random.standard_normal(outChannel) / scale
        self.W_gradient = np.zeros((inChannel, outChannel))
        self.b_gradient = np.zeros(outChannel)

    def forward(self, x):
        """前向过程"""
        ## 补全代码 ##  (32x256)   (256x10)
        self.x=x
        return np.dot(self.x,self.W)+self.b

    def backward(self, delta, learning_rate):
          """反向过程"""
            ## 梯度计算
          batch_size = self.x.shape[0]
          ## 补全代码 ##
          self.W_gradient =np.dot(self.x.transpose(),delta)
          self.b_gradient = np.sum(delta,axis=0)
          delta_backward = np.dot(delta,self.W.transpose())
          ## 反向传播
          self.W -= learning_rate/batch_size*self.W_gradient
          self.b -= learning_rate/batch_size*self.b_gradient
          return delta_backward
```

```
def train(batch=32, lr=0.01, epochs=10):
    # Mnist手写数字集
    dataset_path = "./datasets/mnist"
    train_data = torchvision.datasets.MNIST(root=dataset_path, train=Tru
    train_data.data = train_data.data.numpy()  # [60000,28,28]
    train_data.targets = train_data.targets.numpy()  # [60000]
    train_data.data = train_data.data.reshape(60000, 28, 28, 1) / 255.
    train_data.targets = onehot(train_data.targets, 60000)  # 标签one-hot

    # [28,28] 卷积 6x[5,5] -> 6x[24,24]
    conv1 = Conv(kernel_shape=(5, 5, 1, 6))
    relu1 = Relu()
    # 6x[24,24] -> 6x[12,12]
    pool1 = Pool()
    # 6x[12,12] 卷积 16x(6x[12,12]) -> 16x[8,8]
    conv2 = Conv(kernel_shape=(5, 5, 6, 16))  # 8x8x16
    relu2 = Relu()
    # 16x[8,8] -> 16x[4,4]
    pool2 = Pool()

    # 在这里可以尝试增加网络的深度，再实例化conv3和pool3，记得后面的前向传播过程
    # 和反向传播过程也要有对应的过程
    nn = Linear(256, 10)
    softmax = Softmax()
    for epoch in range(epochs):
        for i in range(0, 60000, batch):
            X = train_data.data[i:i + batch]
            Y = train_data.targets[i:i + batch]
            # 前向传播过程
            predict = conv1.forward(X)
            predict = relu1.forward(predict)
            predict = pool1.forward(predict)
            predict = conv2.forward(predict)
            predict = relu2.forward(predict)
            predict = pool2.forward(predict)
            predict = predict.reshape(batch, -1)  (32x256)
            predict = nn.forward(predict)
(32x10)
```

代码分享

$\delta_i^{(L)} = -(y_i - a_i^{(L)}) f'(z_i^{(L)})$ 全连接层(反向传播)

$\delta_i^{(l)} = \left(\sum_{j=1}^{n_{l+1}} \delta_j^{(l+1)} \theta_{ji}^{(l+1)}\right) f'(z_i^{(l)})$

深蓝学院
shenlanxueyuan.com

```python
## 全连接层
class Linear(object):

    def __init__(self, inChannel, outChannel):
        scale = np.sqrt(inChannel / 2)
        self.W = np.random.standard_normal((inChannel, outChannel)) / sc
        self.b = np.random.standard_normal(outChannel) / scale
        self.W_gradient = np.zeros((inChannel, outChannel))
        self.b_gradient = np.zeros(outChannel)

    def forward(self, x):
        """前向过程"""
        ## 补全代码 ##
        self.x=x
        return np.dot(self.x,self.W)+self.b

    def backward(self, delta, learning_rate):
        """反向过程"""
            ## 梯度计算
        batch_size = self.x.shape[0]
        ## 补全代码 ##
        self.W_gradient =np.dot(self.x.transpose(),delta)
        self.b_gradient = np.sum(delta,axis=0)
        delta_backward = np.dot(delta,self.W.transpose())
        ## 反向传播
        self.W -= learning_rate/batch_size*self.W_gradient
        self.b -= learning_rate/batch_size*self.b_gradient
        return delta_backward
```

256x10

10

batchs: 32

input: 256

output: 10

(32x10)

32x256

32

(256x32) (32x10)

(32x10) (10x256)

(32x256)

```python
## Softmax 函数
class Softmax(object):
    def cal_loss(self, predict, label):
        batchsize, classes = predict.shape
        self.predict(predict)
        loss = 0
        delta = np.zeros(predict.shape)
        for i in range(batchsize):
            delta[i] = self.softmax[i] - label[i]
            loss -= np.sum(np.log(self.softmax[i]) * label[i])
        loss /= batchsize
        return loss, delta

    def predict(self, predict):
        batchsize, classes = predict.shape
        self.softmax = np.zeros(predict.shape)
        for i in range(batchsize):
            predict_tmp = predict[i] - np.max(predict[i])
            predict_tmp = np.exp(predict_tmp)
            self.softmax[i] = predict_tmp / np.sum(predict_tmp)
        return self.softmax
```

(32x10)

(32x10)

$\sum \frac{\partial E}{\partial \theta_{ij}^{(l)}} = \sum \delta_i^{(l)} a_j^{(l-1)}$

$\Rightarrow \delta_i^{(l)} \left(\sum_{j=1} \delta_j^{(l+1)} \theta_{ji}^{(l+1)}\right)$

$\sum \frac{\partial E}{\partial b_i^{(l)}} = \sum \delta_i^{(l)}$

代码分享

卷积层(前向传播)


```
conv
s Conv(object):
 def __init__(self, kernel_shape, step=1, pad=0):
     # [w, h, d]
     width, height, in_channel, out_channel = kernel_shape
     self.step = step
     self.pad = pad
     scale = np.sqrt(3 * in_channel * width * height / out_channel)
     self.k = np.random.standard_normal(kernel_shape) / scale
     self.b = np.random.standard_normal(out_channel) / scale
     self.k_gradient = np.zeros(kernel_shape)
     self.b_gradient = np.zeros(out_channel)

 def forward(self, x):
     self.x = x
     if self.pad != 0:
         self.x = np.pad(self.x, ((0, 0), (self.pad, self.pad), ...
                                  (self.pad, self.pad), (0, 0)), 'constant')
         #x batch,width,height,channel
     bx, wx, hx, cx = self.x.shape
     # kernel的宽、高、通道数、个数
     wk, hk, ck, nk = self.k.shape
     feature_w = (wx - wk) // self.step + 1  # 返回的特征图尺寸
     feature = np.zeros((bx, feature_w, feature_w, nk))

     self.image_col = []
     # kernal也进行了reshape, 便于卷积加速, 只保留通道维度, 是个二维的矩阵
     kernel = self.k.reshape(-1, nk)
     ## 补全代码 ##
     for i in range(bx):
         image_col=img2col(self.x[i],wk,self.step)
         feature[i] =(np.dot(image_col,kernel)+self.b ).reshape(feature_w, feature_w,nk)
         self.image_col.append(image_col)
     return feature
```

kernel重新排列

① 图像转化为矩阵

② $net^l = conv(w^l, a^{l-1}) + w_b$

代码分享 池化层（前向传播）

深蓝学院 shenlanxueyuan.com

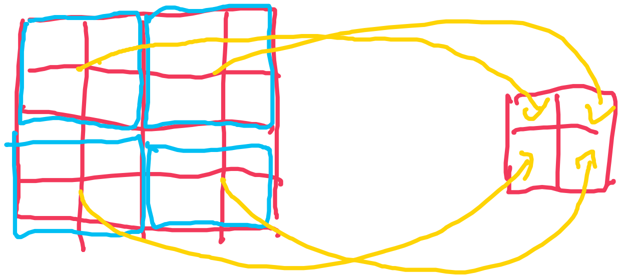
```
## Max Pooling层
class Pool(object):
    def forward(self, x):
        b, w, h, c = x.shape
        feature_w = w // 2
        feature = np.zeros((b, feature_w, feature_w, c))
        self.feature_mask = np.zeros((b, w, h, c))   # 记录最大池化时最大值的位置信息用于反向传播
        for bi in range(b):
            for ci in range(c):
                for i in range(feature_w):
                    for j in range(feature_w):
                        ## 补全代码
                        feature[bi, i, j, ci] = np.max(x[bi,i*2:i*2+2,j*2:j*2+2,ci])
                        index = np.argmax(x[bi, i * 2:i * 2 + 2, j * 2:j * 2 + 2, ci])
                        self.feature_mask[bi, i * 2 + index // 2, j * 2 + index % 2, ci] = 1
        return feature

    def backward(self, delta):
        return np.repeat(np.repeat(delta, 2, axis=1), 2, axis=2) * self.feature_mask
```

构建 feature map

记录最大值的位置

将最大值位置处置为1

```
>>> a
array([[-1.28095222, -1.78081359],
       [ 0.6110716 ,  0.33153531]])
>>> np.argmax(a)
2
```



代码分享. 卷积层(反向传播)


```python
def backward(self, delta, learning_rate):
    bx, wx, hx, cx = self.x.shape   # batch,14,14,inchannel
    wk, hk, ck, nk = self.k.shape   # 5,5,inChannel,outChannel
    bd, wd, hd, cd = delta.shape    # batch,10,10,outChannel
    # 计算self.k_gradient,self.b_gradient
    # 参数更新过程
    ## 补全代码 ##
    delta_col = delta.reshape(bd, -1, cd)
    for i in range(bx):
        self.k_gradient += np.dot(self.image_col[i].transpose(),...
                                  delta_col[i]).reshape(self.k .shape)
    self.k_gradient /= bx
    self.b_gradient += np.sum(delta_col,axis=(0,1))
    self.b_gradient /= bx

    # 计算delta_backward
    # 误差的反向传递
    delta_backward = np.zeros(self.x.shape)
    # numpy矩阵 (对应kernal) 旋转180度
    ## 补全代码 ##
    k_180 = self.k
    k_180 = k_180[::-1,::-1]
    k_180_col = k_180.reshape(-1,ck)
    if hd - hk + 1 != hx:
        pad = (hx - hd + hk - 1) // 2
        pad_delta = np.pad(delta, ((0, 0), ...
                                   (pad, pad), (pad, pad), (0, 0)), 'constant')
    else:
        pad_delta = delta

    for i in range(bx):
        pad_delta_col = img2col(pad_delta[i], wk, self.step)
        delta_backward[i] = np.dot(pad_delta_col, k_180_col).reshape(wx, hx, ck)

    # 反向传播
    self.k -= self.k_gradient * learning_rate
    self.b -= self.b_gradient * learning_rate

    return delta_backward
```

32, 12, 12, 6

5, 5, 6, 16

32, 8, 8, 16

(64, 150)

(64, 16)

(1, 16)

① k_180 = self.k

② k_180 = k_180[::-1,::-1]

保留深度通道

以 cov2 为例:

cov2(5,5,6,16)

k_shape: 5×6

inchannel: 6

outchannel: 16

step=1 Depth=6 Filter=16

$$\frac{\partial E_d}{\partial w_{ij}} = \sum^{wk,\,hk} \delta^{l}\, a^{l-1}$$

$$\frac{\partial E_d}{\partial w_b} = \sum_i \sum_j \delta^{l}_{ij}$$

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \xrightarrow{\text{flipped}} \begin{bmatrix} w_{22} & w_{21} \\ w_{12} & w_{11} \end{bmatrix}$$

```
>>> a
array([[0, 1],
       [2, 3]])
>>> a[::-1,::-1]
array([[3, 2],
       [1, 0]])
```

② (旋转180°)

代码分享. 池化层（反向传播）

```
## Max Pooling层
class Pool(object):
    def forward(self, x):
        b, w, h, c = x.shape
        feature_w = w // 2
        feature = np.zeros((b, feature_w, feature_w, c))
        self.feature_mask = np.zeros((b, w, h, c))  # 记录最大池化时最大值的位置信息用于反向传播
        for bi in range(b):
            for ci in range(c):
                for i in range(feature_w):
                    for j in range(feature_w):
                        ## 补全代码
                        feature[bi, i, j, ci] = np.max(x[bi,i*2:i*2+2,j*2:j*2+2,ci])
                        index = np.argmax(x[bi, i * 2:i * 2 + 2, j * 2:j * 2 + 2, ci])
                        self.feature_mask[bi, i * 2 + index // 2, j * 2 + index % 2, ci] = 1
        return feature

    def backward(self, delta):
        return np.repeat(np.repeat(delta, 2, axis=1), 2, axis=2) * self.feature_mask
```

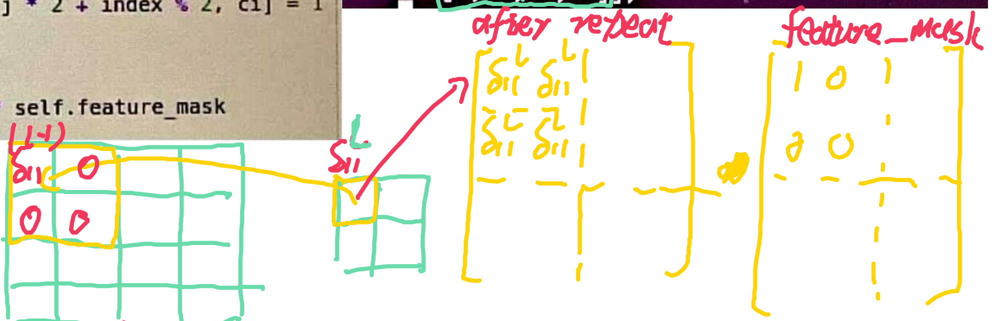
(32, 4, 4, 16)

axis=1 axis=2

以Pool2为例:

delta(32, 4, 4, 16)

```
>>> a
array([[0, 1],
       [2, 3]])
>>> np.repeat(np.repeat(a,2,axis=0),2,axis=1)
array([[0, 0, 1, 1],
       [0, 0, 1, 1],
       [2, 2, 3, 3],
       [2, 2, 3, 3]])
```



① 无学习的参数

② 将误差项传递到上一层，没有梯度计算

③ 对于 Max Pooling，下一层的误差项的值传递到上一层对应区中最大值对应的神经元，其它神经元误差项为0.

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

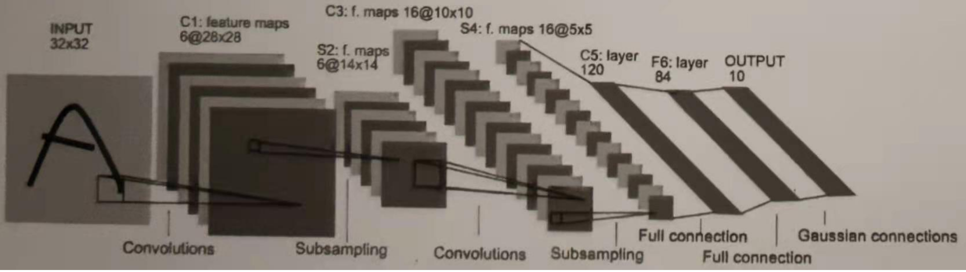
Thanks for Listening

作业讲解

1. 对比卷积神经网络与全连接神经网络，在图像分类任务中，原始图像大小的变化将会怎样影响模型可训练参数个数?

2. 卷积神经网络是通过什么方式来完成可训练参数的减少?

3. 如图是LeNet-5的示意图，试着写出每一层的参数个数。



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

作业讲解

Input：32×32

⇒ kernel_size：5×5

C_1：feature_map：6×(28×28)

C_1层参数：k：6×(1×25)=150　b：6×(1×1)=6　k+b=156

S_2：pooling_size：2×2

若採樣采用 max, mean 方法不需要参数训练

若采用一个训练参数，则需要：k：6×1=6　b：6×1=6　k+b=12

作业讲解

深蓝学院 shenlanxueyuan.com

C_3：根据右图的输入方式：共需要：

$6\times(\overset{\text{相邻3个神经元}}{3\times25}+1)+6\times(\overset{\text{相邻4个神经元}}{4\times25+1})+3\times(\overset{\text{间隔两个}}{2\times25+1})+3\times(\overset{\text{间隔一个}}{2\times25+1})$

$+1\times(\overset{\text{所有神经元}}{6\times25+1})=1516$

S_4层：
- 若采用 max，mean 则不需要参数
- 若采用可训练参数，共需要 $16\times(1+1)=32$

C_5层：共有120个特征图，每个特征图与S_4层16个单元相连。
由于S_4层的16个单元都为 5×5，特征图也为 5×5

故 C_5特征图大小为 $(5-5+1)=1$

作业讲解

深蓝学院
shenlanxueyuan.com

C_5层参数为： $120 \times (16 \times 5 \times 5 + 1) = 48120$

F_6层： 与C_5层全相连.

$$84 \times (120 \times \underbrace{(1 \times 1)}_{C_5\text{特征图}} + 1) = 10164$$