实验报告

莫书琪

17318086

模型架构

`cnn.py`中原来定义了一个三层的卷积神经网络，包含一个池化的卷积层和两个全连接层，其中第一个全连接层使用`relu`非线性变换，第二个全连接层使用`softmax`非线性变换计算`loss`作为网络输出。在训练best model时，我在以上网络结构的基础上进行改进，搭建了包含一个卷积层和三个全连接层的四层神经网络。

在卷积层中，我参考`AlexNet`卷积层的设计，定义了一个带batch normalization的池化卷积层。对新的卷积层，`layer_utils.py`中没有对应的前向和后向传播函数，所以要利用`layer.py`提供的组件定义新的前向和后向传播函数。

```
def conv_bn_relu_pool_forward(x, w, b, gamma, beta, conv_param, bn_param,
pool_param):
    a, conv_cache = conv_forward_fast(x, w, b, conv_param)
    an, bn_cache = spatial_batchnorm_forward(a, gamma, beta, bn_param)
    s, relu_cache = relu_forward(an)
    out, pool_cache = max_pool_forward_fast(s, pool_param)
    cache = (conv_cache, bn_cache, relu_cache, pool_cache)
    return out, cache

def conv_bn_relu_pool_backward(dout, cache):
    conv_cache, bn_cache, relu_cache, pool_cache = cache
    ds = max_pool_backward_fast(dout, pool_cache)
    dan = relu_backward(ds, relu_cache)
    da, dgamma, dbeta = spatial_batchnorm_backward(dan, bn_cache)
    dx, dw, db = conv_backward_fast(da, conv_cache)
    return dx, dw, db, dgamma, dbeta
```

在全连接层中，与原来定义的三层卷积神经网络相比，我首先在线性变换与非线性变换之间增加了batch normalization处理。因为`layer_utils.py`中有对应的前向和后向传播函数，所以直接调用就可以了。此外，为了减缓过拟合效应对训练结果的影响，我在中间两个全连接层输出的后面添加了`dropout`层处理，这里是调用`layer.py`中定义的`dropout_forward`和`dropout_backward`。

有一点需要注意的是，`dropout`层只在全连接层后面使用，不在卷积层后使用。这是因为在卷积层中图像中相邻的像素共享很多相同的信息，如果它们中的任何一个被删除，那么它们所包含的信息可能仍然会从活动的相邻像素传递，所以在卷积层后面加`dropout`作用不是很大。

以下为模型的完整代码：

```
class MyNet(object):
    """
    conv - bn - relu - 2x2 max pool - affine - bn - relu - dropout - affine - bn
- relu - dropout - affine - softmax
    """
    def __init__(
        self,
        input_dim=(3, 32, 32),
```

```
        num_filters=32,
        filter_size=5,
        hidden_dim=500,
        hidden_dim1=200,
        num_classes=10,
        weight_scale=1e-3,
        reg=0.0,
        dtype=np.float32,
    ):
        self.params = {}
        self.reg = reg
        self.dtype = dtype

        C, H, W = input_dim
        after_pooling_dim = int(H * W / 4.0)
        self.params["W1"] = np.random.normal(0.0, weight_scale,
(num_filters,C,filter_size,filter_size))
        self.params["b1"] = np.zeros(num_filters)
        self.params["W2"] = np.random.normal(0.0, weight_scale,
(after_pooling_dim*num_filters, hidden_dim))
        self.params["b2"] = np.zeros(hidden_dim, dtype=float)
        self.params["W3"] = np.random.normal(0.0, weight_scale, (hidden_dim,
hidden_dim1))
        self.params["b3"] = np.zeros(hidden_dim1, dtype=float)
        self.params["W4"] = np.random.normal(0.0, weight_scale,
(hidden_dim1,num_classes))
        self.params["b4"] = np.zeros(num_classes, dtype=float)
        self.params["gamma1"] = np.ones(num_filters)
        self.params["beta1"] = np.zeros(num_filters)
        self.params["gamma2"] = np.ones(hidden_dim)
        self.params["beta2"] = np.zeros(hidden_dim)
        self.params["gamma3"] = np.ones(hidden_dim1)
        self.params["beta3"] = np.zeros(hidden_dim1)


        for k, v in self.params.items():
            self.params[k] = v.astype(dtype)

    def loss(self, X, y=None):
        W1, b1 = self.params["W1"], self.params["b1"]
        W2, b2 = self.params["W2"], self.params["b2"]
        W3, b3 = self.params["W3"], self.params["b3"]
        W4, b4 = self.params["W4"], self.params["b4"]
        beta1 = self.params["beta1"]
        gamma1 = self.params["gamma1"]
        beta2 = self.params["beta2"]
        gamma2 = self.params["gamma2"]
        beta3 = self.params["beta3"]
        gamma3 = self.params["gamma3"]

        filter_size = W1.shape[2]
        conv_param = {"stride": 1, "pad": (filter_size - 1) // 2}
        pool_param = {"pool_height": 2, "pool_width": 2, "stride": 2}
        self.dropout_param = {'mode': 'train', 'p': 0.2}

        bn_param1 = {'mode': 'train', 'running_mean': np.zeros(beta1.shape[0]),
'running_var': np.zeros(beta1.shape[0])}
```

```
        out1, cache1 = conv_bn_relu_pool_forward(X, W1, b1, gamma1, beta1,
conv_param, bn_param1, pool_param)
        out1flat = out1.reshape(out1.shape[0], -1)
        bn_param2 = {'mode': 'train', 'running_mean': np.zeros(beta2.shape[0]),
'running_var': np.zeros(beta2.shape[0])}
        out2, cache2 = affine_bn_relu_forward(out1flat, W2, b2, gamma2, beta2,
bn_param2)
        out2, cache_do = dropout_forward(out2, self.dropout_param)
        cache2 = (cache2, cache_do)
        bn_param3 = {'mode': 'train', 'running_mean': np.zeros(beta3.shape[0]),
'running_var': np.zeros(beta3.shape[0])}
        out3, cache3 = affine_bn_relu_forward(out2, W3, b3, gamma3, beta3,
bn_param3)
        out3, cache_do = dropout_forward(out3, self.dropout_param)
        cache3 = (cache3, cache_do)
        out4, cache4 = affine_forward(out3, W4, b4)
        scores = out4

        if y is None:
            return scores

        loss, grads = 0, {}
        loss, dout4 = softmax_loss(scores, y)
        dout3, grads["W4"], grads["b4"] = affine_backward(dout4, cache4)
        cache3, cache_do = cache3
        dout3 = dropout_backward(dout3, cache_do)
        dout2, grads["W3"], grads["b3"], grads["gamma3"], grads["beta3"] =
affine_bn_relu_backward(dout3, cache3)
        cache2, cache_do = cache2
        dout2 = dropout_backward(dout2, cache_do)
        dout1flat, grads["W2"], grads["b2"], grads["gamma2"], grads["beta2"] =
affine_bn_relu_backward(dout2, cache2)
        dout1 = dout1flat.reshape(out1.shape)
        dx, grads["W1"], grads["b1"], grads["gamma1"], grads["beta1"] =
conv_bn_relu_pool_backward(dout1, cache1)

        for i in range(1,5):
            loss += 0.5 * self.reg * (self.params["W%d"%i] ** 2).sum()
            grads["W%d"%i] += self.reg * self.params["W%d"%i]
        return loss, grads
```

调参过程

使用原来的三层卷积神经网络训练集准确率是50.4%，测试集准确率是49.9%。我基于这一模型结构和结果进行了下面四个步骤的调参。

1.调整与卷积层相关的参数

首先希望可以调整与卷积层相关的参数，找到在这个任务下表现较好的参数组合。这一步放到最前面做考虑的是，如果后面模型结构比较复杂的时候，调参会比较花时间，而且这部分参数与具体的模型结构没有太大的关系，所以在简单结构下先调整好。根据jupyter notebook的指引，这一步可以调整`filter size`和`filter number`的值。

- `filter size`调参：这个参数的含义是卷积核的大小，卷积核如果尺寸过大容易丢失图像中的信息，尺寸过小也可能会捕捉不到图像关于轮廓这一类的关键信息。在原来的三层卷积神经网络中，这个参数被设定为7，我尝试设为9发现结果略差了一些，设为5的结果略好了一些，改成3又变差了

- `filter number`调参：这个参数的含义是卷积核的数量。在原来的三层卷积神经网络中，这个参数被设定为32，通过实验发现设为35的结果会略差一些，设为30的结果也会略差一些
- **最终参数设定**：`filter size`设定为5，`filter number`设定为32

2.调整模型结构

寻找到效果较好的卷积层参数组合后，下一步是调整模型结构。我在完成实验时经历了以下尝试：

- 尝试在第一层前面加多一个带bn的卷积层：准确度下降，而且有出现过拟合现象
- 直接在第一层卷积层中加bn处理：准确率提高到56.1%
- 在上一步的基础上把全连接层也改成带bn的：准确率提高到58.7%
- 在上一步的基础上加多一层200个参数的bn全连接层：准确率进一步提高，而且没有过拟合现象
- 尝试在上一步的基础上增加全连接层的参数值：发现模型效果变差了，所以不进行这一步处理
- 虽然一个Epoch训练的时候不太会出现过拟合效应，但为了之后多个Epoch训练方便调参，还是先把dropout层加到全连接层后面了，并且暂时把dropout参数值设为0，即没有进行dropout处理
- **最终模型结构**：带bn的池化卷积层 + 带bn和dropout的relu全连接层（500个参数）+ 带bn和dropout的relu全连接层（200个参数）+ softmax全连接层输出

3.调整影响整个训练过程的参数

确定好模型结构，接下来是对损失函数和全局参数进行玄学调参。我在完成实验时经历了以下尝试：

- 把`softmax loss`换成`svm loss`：这两个损失函数理论上都适用于多分类任务，并没有说哪个损失函数一定效果会比较好，还是要看具体任务。把`softmax loss`换成`svm loss`后，训练集准确率58.6%，测试集57.9%，感觉没有`softmax loss`的好
- `weight_scale`调参：这个参数代表模型参数初始值的方差。经过多次实验发现，这个参数被设定为0.005时，模型初始化准确率会比较好
- `reg`调参：这个参数是正则化项的参数，正则化项主要用于控制过拟合。多次实验尝试后没有找到更优的参数值
- `learning_rate`调参：这个参数是学习率，主要影响迭代时参数更新的过程。多次实验尝试后也是没有找到更优的参数值
- **最终参数设定**：把`weight_scale`设定为0.005，其余参数和损失函数不变

4.增加训练Epoch

有了一个结构较优、参数调整的差不多的模型后，就可以通过增加训练Epoch的方式来增加迭代次数，观察模型的准确率。在完成实验时，我把`batch_size`设置为100，`num_epoches`设置为10，大概需要花一个小时完成一次训练。

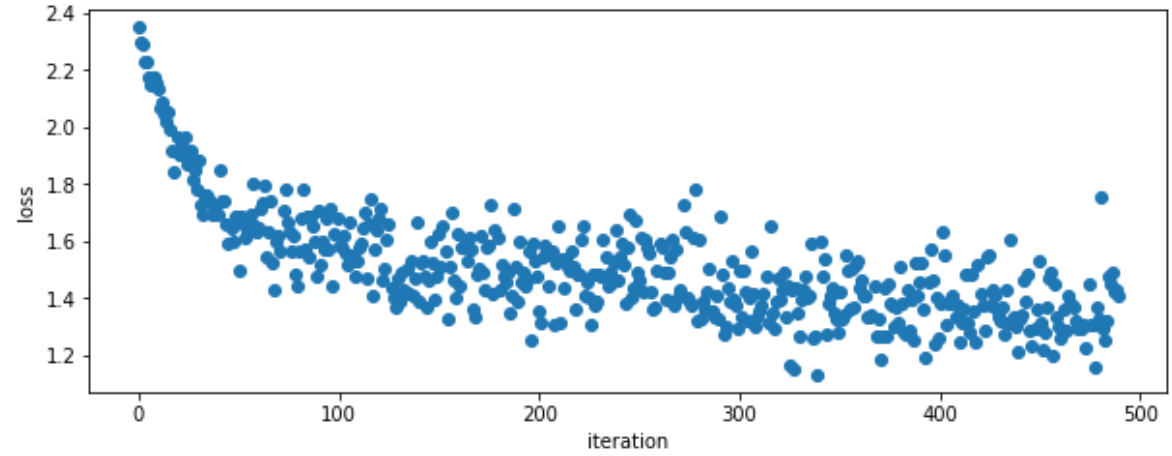
同时这里还需要对dropout比率`p`进行调参。经过多次尝试后，我发现`p`设置为0.2跑出来的效果会比较好，即全连接层在每次训练中都会随机选取20%的参数让其失效。

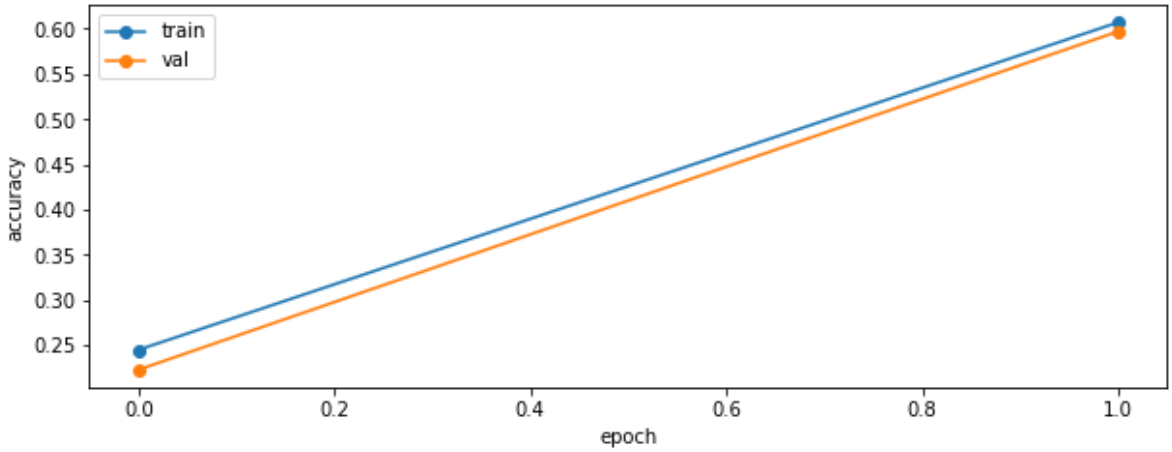
完成了这一步调参后，测试集最终可以达到**67%**以上的准确率。

实验结果

1个Epoch下的best model

在经历了前面三个调参步骤后，可以得到训练集60.7%准确率，测试集59.7%的结果。对训练中的`loss`可视化，可以看出随着迭代次数的增加，`loss`呈下降趋势，但是波动较大，还没有达到一个比较稳定的状态，说明还可以通过增加迭代次数进一步训练模型。

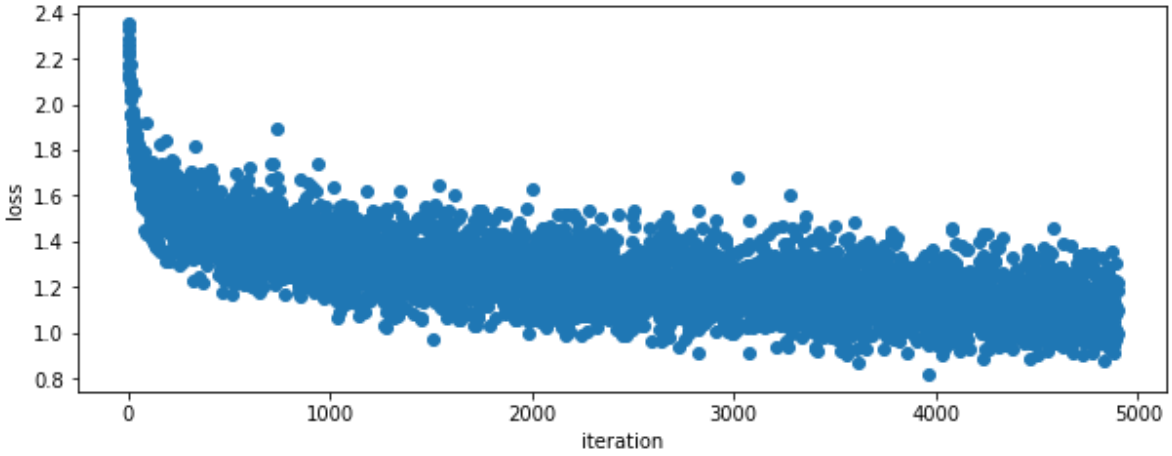


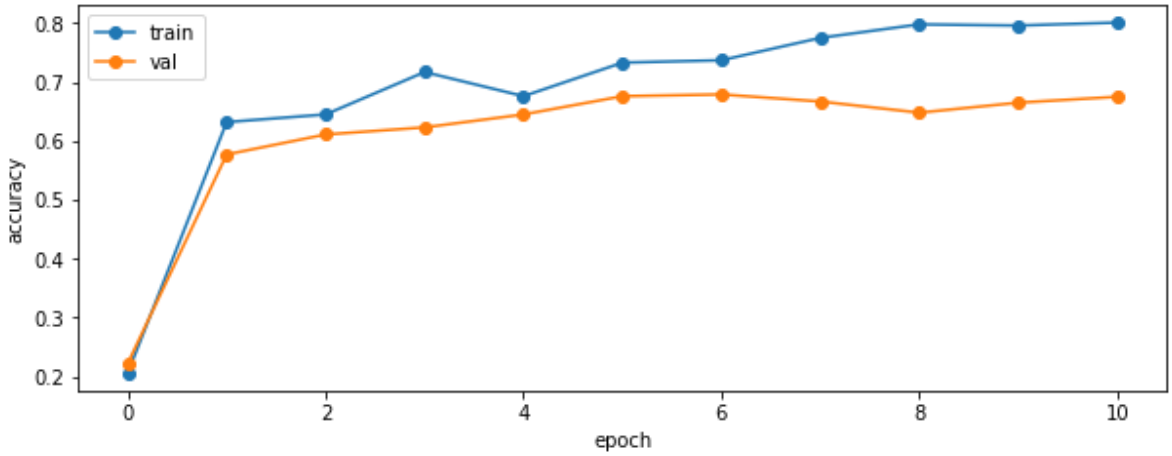


10个Epoch下的best model

把dropout的`p`设置为0.2，然后迭代10个epoch，如果以测试集准确率为评价标准，**得到的最好模型训练集准确率是73.7%，测试集准确率是67.9%（Epoch 6）**。

从图中可以看出，`loss`迭代到后面的epoch下降斜率已经比较接近0了，而且波动的宽度基本一致，说明已经达到了比较稳定的状态。对比训练集和测试集在不同epoch下的准确率，发现在第5个epoch后测试集准确率基本稳定在了67.5%左右的水平，训练集准确率持续提升到第8个epoch后也基本稳定在了80%的水平。这里我尝试过调高dropout的`p`值参数，但是并没有改善过拟合效果。因此推测67.5%左右是这种网络结构下测试集能达到的理想准确率水平。


loss
iteration
2.4
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0
1000
2000
3000
4000
5000


train
val
accuracy
epoch
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
2
4
6
8
10