深蓝学院
shenlanxueyuan.com

PointNet

主讲人 谢官麟

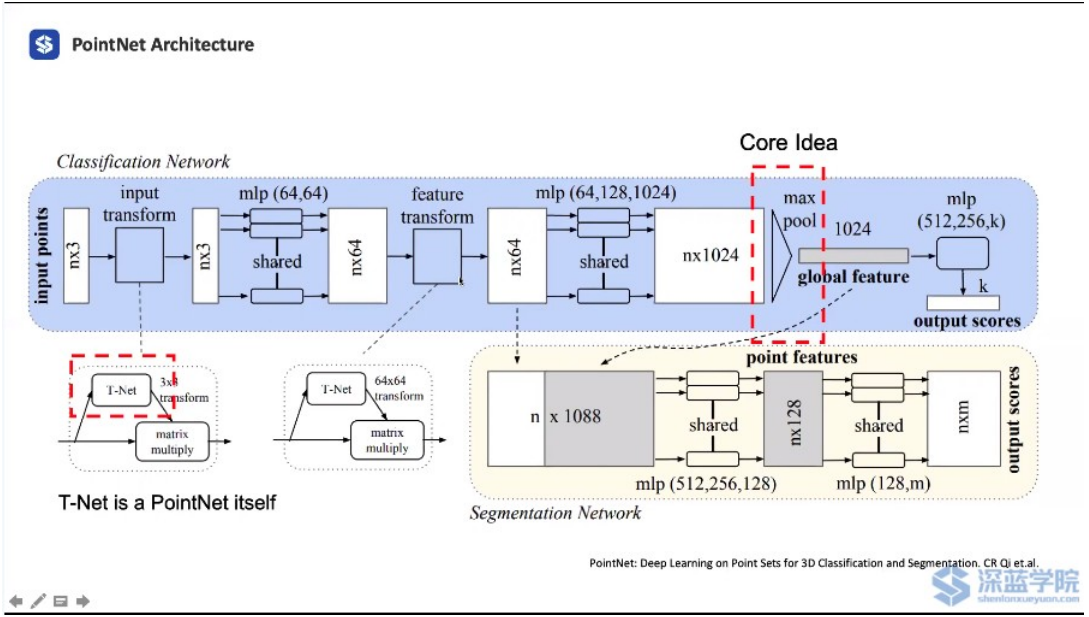
纲要

深蓝学院
shenlanxueyuan.com

- 第一部分：网络结构
- 第二部分：代码实现
- 第三部分：学习结果

网络结构

- 我实现的是课程 PPT 中所讲的最简单的 PointNet，由于课程中说 T-Net 作用有限所以我也就直接跳过了。
- 剩下的结构其实全都是 mlp 的堆叠，从图上 mlp 后的括号可以看出来维数的变化是 3->64->64->64->128->1024->512->256->k. 另外在到 1024 维的时候进行了一次 max pool 来消除点顺序的影响。



纲要

深蓝学院
shenlanxueyuan.com

- 第一部分：网络结构
- **第二部分：代码实现**
- 第三部分：学习结果

代码实现

- 网络中 mlp 对应的就是 nn.Linear ，括号中是输入与输出的维数，值得注意的是为了网络的非线性所以每次全连接层后面都要跟一个激活函数。由于 ReLu 没参数，所以整个网络共用一个就可以了。
- 在做 max pool 的时候我直接把表示数据点个数的维度放到后面，然后取完最大值就消掉了，因为算损失函数也不需要这个维度。

```
class PointNetCls(nn.Module):
    def __init__(self, k):
        super(PointNetCls, self).__init__()
        self.fc1 = nn.Linear(3, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, 64)
        self.fc4 = nn.Linear(64, 128)
        self.fc5 = nn.Linear(128, 1024)
        self.fc6 = nn.Linear(1024, 512)
        self.fc7 = nn.Linear(512, 256)
        self.fc8 = nn.Linear(256, k)
        # self.bn1 = nn.BatchNorm1d(65)
        # self.bn2 = nn.BatchNorm1d(64)
        # self.bn3 = nn.BatchNorm1d(64)
        # self.bn4 = nn.BatchNorm1d(128)
        # self.bn6 = nn.BatchNorm1d(512)
        # self.bn7 = nn.BatchNorm1d(256)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = x.transpose(2, 1).contiguous()
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.relu(self.fc3(x))
        x = self.relu(self.fc4(x))
        x = self.fc5(x)

        x = x.transpose(2, 1).contiguous()
        x = torch.max(x, dim=2, keepdim=False)[0]
        # x = x.transpose(2, 1).contiguous()

        x = self.relu(self.fc6(x))
        x = self.relu(self.fc7(x))
        x = F.log_softmax(self.fc8(x), dim=1)

        return x
```

代码实现

- 这里的 contiguous() 是因为换维度的时候数据的内存分布就不符合新的维度表达了，所以让它重新分配个连续的内存。
- 最后用的 log_softmax() 就是把 softmax 多算了个 log ，防止一些极端值在算对数的时候溢出，在加快运算速度的同时，可以保持数值的稳定性。要用 softmax 也行，就是算 loss 的时候注意 CrossEntropyLoss() 和 NLLLoss() 的区别。

```python
class PointNetCls(nn.Module):
    def __init__(self, k):
        super(PointNetCls, self).__init__()
        self.fc1 = nn.Linear(3, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, 64)
        self.fc4 = nn.Linear(64, 128)
        self.fc5 = nn.Linear(128, 1024)
        self.fc6 = nn.Linear(1024, 512)
        self.fc7 = nn.Linear(512, 256)
        self.fc8 = nn.Linear(256, k)
        # self.bn1 = nn.BatchNorm1d(65)
        # self.bn2 = nn.BatchNorm1d(64)
        # self.bn3 = nn.BatchNorm1d(64)
        # self.bn4 = nn.BatchNorm1d(128)
        # self.bn6 = nn.BatchNorm1d(512)
        # self.bn7 = nn.BatchNorm1d(256)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = x.transpose(2, 1).contiguous()
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.relu(self.fc3(x))
        x = self.relu(self.fc4(x))
        x = self.fc5(x)

        x = x.transpose(2, 1).contiguous()
        x = torch.max(x, dim=2, keepdim=False)[0]
        # x = x.transpose(2, 1).contiguous()

        x = self.relu(self.fc6(x))
        x = self.relu(self.fc7(x))
        x = F.log_softmax(self.fc8(x), dim=1)

        return x
```

代码实现

- 然后为什么我这里原本想用 bn 层后面注掉了呢?
- 因为数据是 (batch_size, num_points, values) 然后 BatchNorm1d 默认操作对象是第二个维度， Linear 默认操作对象是第三个维度，要用 bn 我最好改成 Conv1d ，否则就需要不停地做 transpose. 我感觉没什么必要而且 Linear 更符合 PPT 就没做，后续如果有谁比较感兴趣可以试一下。

```python
class PointNetCls(nn.Module):
    def __init__(self, k):
        super(PointNetCls, self).__init__()
        self.fc1 = nn.Linear(3, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, 64)
        self.fc4 = nn.Linear(64, 128)
        self.fc5 = nn.Linear(128, 1024)
        self.fc6 = nn.Linear(1024, 512)
        self.fc7 = nn.Linear(512, 256)
        self.fc8 = nn.Linear(256, k)
        # self.bn1 = nn.BatchNorm1d(65)
        # self.bn2 = nn.BatchNorm1d(64)
        # self.bn3 = nn.BatchNorm1d(64)
        # self.bn4 = nn.BatchNorm1d(128)
        # self.bn6 = nn.BatchNorm1d(512)
        # self.bn7 = nn.BatchNorm1d(256)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = x.transpose(2, 1).contiguous()
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.relu(self.fc3(x))
        x = self.relu(self.fc4(x))
        x = self.fc5(x)

        x = x.transpose(2, 1).contiguous()
        x = torch.max(x, dim=2, keepdim=False)[0]
        # x = x.transpose(2, 1).contiguous()

        x = self.relu(self.fc6(x))
        x = self.relu(self.fc7(x))
        x = F.log_softmax(self.fc8(x), dim=1)

        return x
```

代码实现

- 其它部分的代码我基本直接用的开源版本原来的，没什么改动。
- dataset 部分主要就注意 __getitem__ ，然后它的 data_augmentation 里面已经做了前两个维度 (x, y) 的旋转和随机的噪声。

```python
def __getitem__(self, index):
    fn = self.fns[index]
    cls = self.cat[fn.split('/')[0]]
    with open(os.path.join(self.root, fn), 'rb') as f:
        plydata = PlyData.read(f)
    pts = np.vstack([plydata['vertex']['x'], plydata['vertex']['y'], plydata['vertex']['z']]).T
    choice = np.random.choice(len(pts), self.npoints, replace=True)
    point_set = pts[choice, :]

    point_set = point_set - np.expand_dims(np.mean(point_set, axis=0), 0)  # center
    dist = np.max(np.sqrt(np.sum(point_set ** 2, axis=1)), 0)
    point_set = point_set / dist  # scale

    if self.data_augmentation:
        theta = np.random.uniform(0, np.pi * 2)
        rotation_matrix = np.array([[np.cos(theta), -np.sin(theta)], [np.sin(theta), np.cos(theta)]])
        point_set[:, [0, 2]] = point_set[:, [0, 2]].dot(rotation_matrix)  # random rotation
        point_set += np.random.normal(0, 0.02, size=point_set.shape)  # random jitter

    point_set = torch.from_numpy(point_set.astype(np.float32))
    cls = torch.from_numpy(np.array([cls]).astype(np.int64))
    return point_set, cls
```

代码实现


```
for epoch in range(opt.nepoch):
    scheduler.step()
    for i, data in enumerate(dataloader, 0):
        count += 1
        points, target = data
        target = target[:, 0]
        points = points.transpose(2, 1)
        points, target = points.cuda(), target.cuda()
        optimizer.zero_grad()
        classifier = classifier.train()
        pred = classifier(points)
        loss = F.nll_loss(pred, target)
        loss.backward()
        optimizer.step()
        pred_choice = pred.data.max(1)[1]
        correct = pred_choice.eq(target.data).cpu().sum()
        print('[%d: %d/%d] %s loss: %f accuracy: %f' % (epoch, i, num_batch, green('train'), loss.item(), correct.item() / float(opt.batchSize)))
        writer_train.add_scalar('loss', loss.item(), global_step=count)
        writer_train.add_scalar('accuracy', correct.item() / float(opt.batchSize), global_step=count)
```

- 主体部分所要做的也就是在每个 epoch 里读取数据，预测，计算误差（损失），反响传播更新梯度，更新学习率，非常简单。

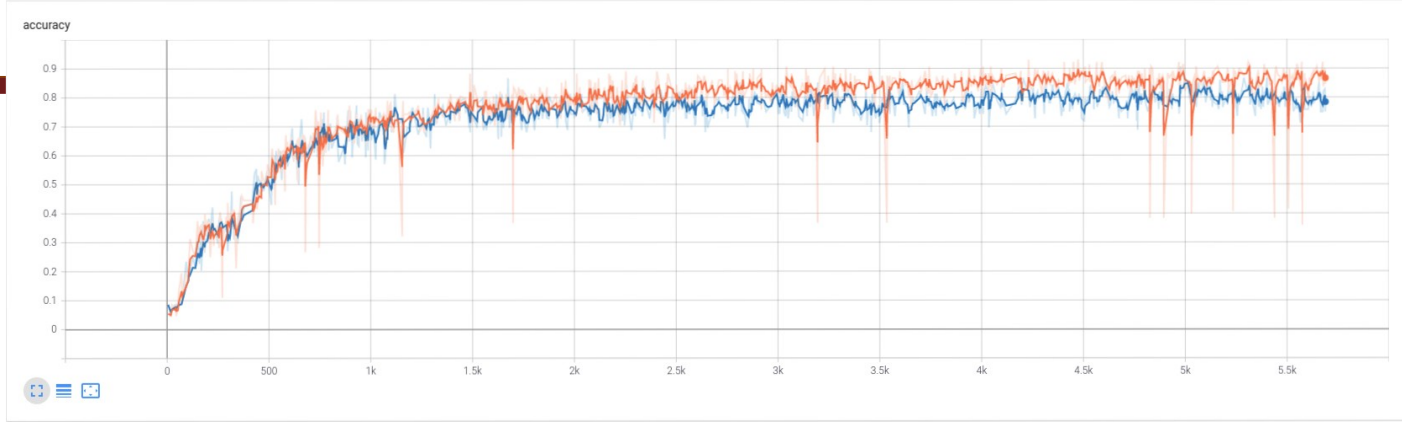
纲要

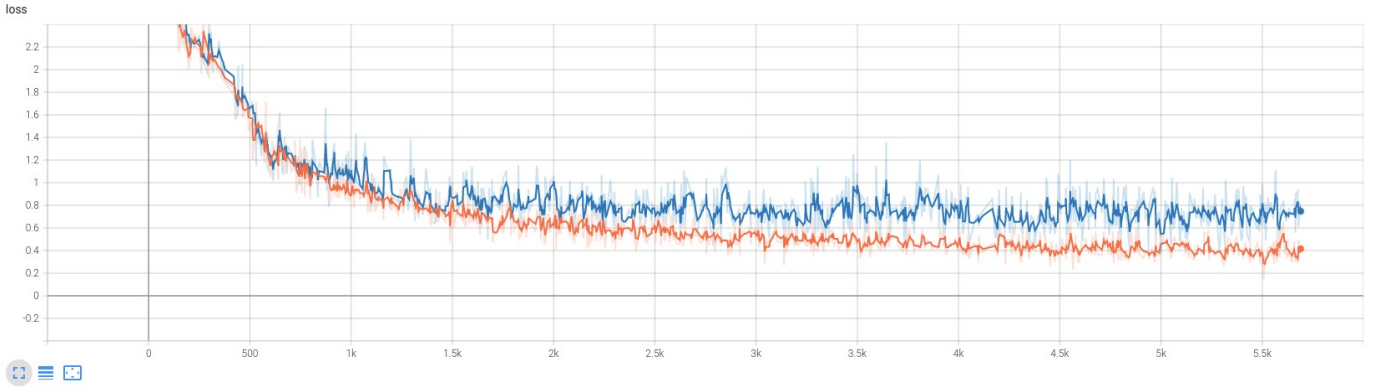
- 第一部分：网络结构
- 第二部分：代码实现
- **第三部分：学习结果**

学习结果

深蓝学院

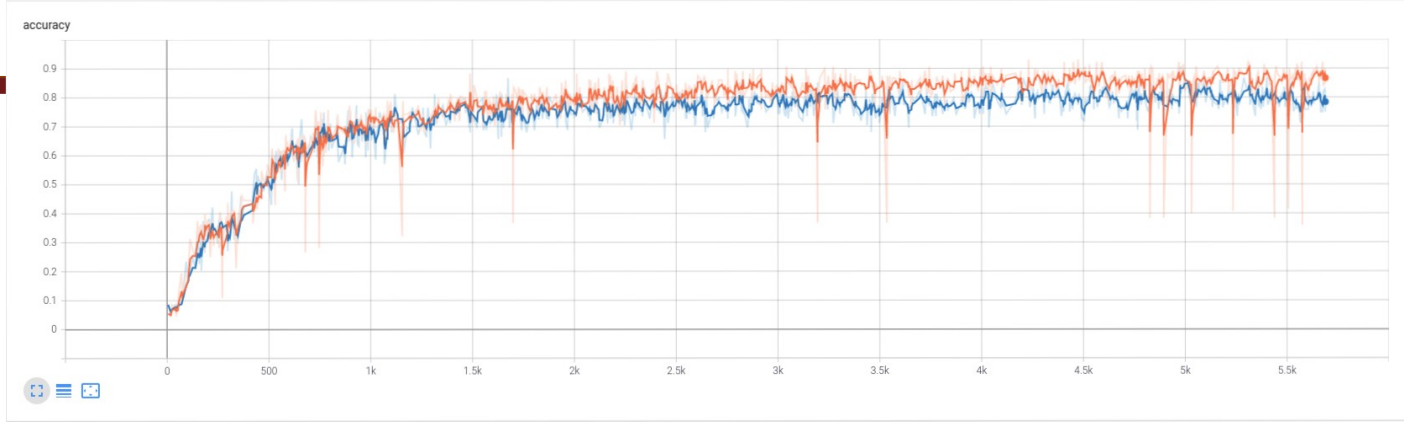
- 其中红线是训练集 (train) 过程，蓝线是验证集 (val) 过程，一开始的时候两个差不多，然后随着渐渐有点过拟合双方的差距就慢慢拉大了，总体上来说 train 的数据肯定是比 val 好看的，毕竟是针对其数据训练的。
- 当然这比官方实现差了点，毕竟只是一个最简单的实现。

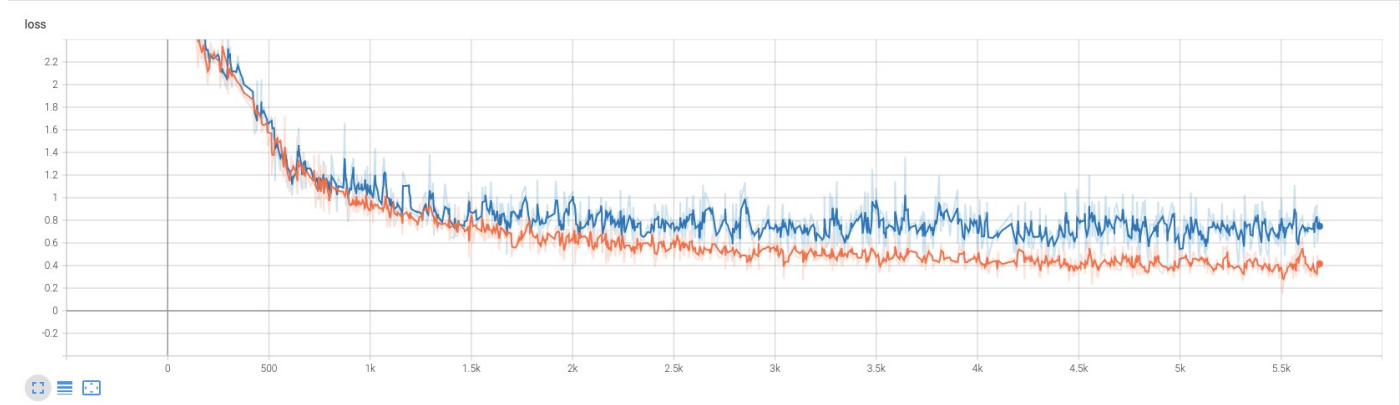




学习结果

- 最后拿测试集试了下结果是80.264%，应该说结构这么简单的网络能有这样的结果已经达到预期了。
- 另外其实在学习过程中验证集的accuracy连续下降或者loss连续上升的时候也应该提早终止训练的，我懒癌发作又没写。感兴趣的可以试一下，训练的循环里加一句就可以了。





在线问答

深蓝学院
shenlanxueyuan.com

Q&A

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening