Introduction to Deep learning @22 Autumn, USTC

LAB1-实验报告

SA22011035 李钊佚

目录:

正文:

➢ 实验过程陈述:

本次实验研究使用简单前馈神经网络近似一个初等函数：$y = sinx + e^{-x},\ x \in [0, 4\pi]$，按照实验指导书的指引，我在 pytorch 环境下进行了本次实验。我的代码主要分为：数据集的产生与划分、模型的构建、模型训练部分代码、模型性能测试部分代码。经过调试，我的代码可以顺利运行，其执行逻辑流程如下：

【数据集产生和划分：随机且独立地从[0, 4π]中采样出若干个点，并将其随机划分为 training \ validation \ testing 集合】->

【模型的构建：使用 pytorch 构建简单的前馈神经网络，其中利用 nn.ModuleList()来动态定义前馈神经网络中的隐藏层数】->

【模型训练：把数据按照 batch 送入定义的前馈神经网络中，进行 forward 和 backward 过程，更新参数】->

【模型性能测试：每个 epoch 结束时，记录 epoch 中平均的 batch_loss，和当前模型在 validation set 上面的 loss，当模型训练

结束时，根据 validation loss 计算模型在 testing set 上的 loss】

➢ 关键代码展示：

【数据集产生和划分】

---

training set : development set : testing set = 7:1.5:1.5

```
data_size = 5000
x=np.linspace(0,4*np.pi,data_size)
print(x[1])
y=np.sin(x)+np.exp(-x)

X=np.expand_dims(x,axis=1)
Y=y.reshape(data_size,-1)

dataset=TensorDataset(torch.tensor(X,dtype=torch.float).cuda(),torch.tensor(
Y,dtype=torch.float).cuda())
train_ratio = 0.7
dev_ratio = 0.15
test_ratio = 0.15
train_size = int(train_ratio*data_size)
dev_size = int(dev_ratio*data_size)
test_size = data_size - train_size - dev_size
length_split = [train_size, dev_size, test_size]
train_set, dev_set, test_set = torch.utils.data.dataset.random_split(dataset,
length_split)
```

【模型的构建】

```
class Net(nn.Module):
def __init__(self):
super(Net, self).__init__()
self.inp_layer = nn.Linear(1, args.width)
self.hiddens = nn.ModuleList()
'''
to make parameter in the net.parameters()
'''
for i in range(args.depth-1):
self.hiddens.append(nn.Linear(args.width,
args.width).cuda())
```

```
self.out_layer = nn.Linear(args.width, 1)
if args.actfunc == 'relu':
self.activate = F.relu
elif args.actfunc == 'tanh':
self.activate = torch.tanh
elif args.actfunc == 'sigmoid':
self.activate = torch.sigmoid
else:
raise Exception("ERROR: parameter `actfunc` is invalid!")

def forward(self, inp):
x = self.activate(self.inp_layer(inp))
for hid_layer in self.hiddens:
x = self.activate(hid_layer(x))
x = self.out_layer(x)
return x
```

【模型训练】

```
epoch_num = 500
train_losses = list()
val_losses = list()
steps = list()

for epoch in range(epoch_num):
loss=None
loss_epoch = 0
cnt=0
for batch_x,batch_y in dataloader:
net.train()
#print('flag_')
y_predict=net(batch_x)
loss=Loss(y_predict,batch_y)
#print('flag___')
optim.zero_grad()
loss.backward()
```

```
optim.step()
loss_epoch += loss.item()
cnt += 1
loss_epoch /= cnt
```

【模型性能测试】

---

validation loss :

```
if (epoch+1)%10==0:
print("step: {0} , loss: {1}".format(epoch+1,loss.item()))
steps.append(epoch+1)
train_losses.append(loss_epoch)

net.eval()
with torch.no_grad():
y_val_pred =
net(torch.tensor(X_val,dtype=torch.float).cuda())
val_loss =
Loss(torch.tensor(Y_val,dtype=torch.float).cuda(),
y_val_pred)
print(val_loss.item())
val_losses.append(val_loss.item())
```

```
# testing loss
y_test_pred=net(torch.tensor(X_test,dtype=torch.float).cu
da())
test_loss =
Loss(torch.tensor(Y_test,dtype=torch.float).cuda(),
y_test_pred).item()
```

➢ 测试结果和超参数分析:

我们在如下空间搜索超参数：

【learning rate = 1e-2, 5e-3, 1e-3, 5e-4】

【activation function = relu, tanh, sigmoid】

关于网络宽度和网络深度的影响，为了公平比较，我们设置隐藏层神经元的数量一致：即隐藏层深度与隐藏层宽度的乘积约为 40

【width = 20, depth = 2;

width = 13, depth = 3;

width = 10, depth = 4】

测试结果：

@1: width = 20, depth = 2:

testing loss	1e-2	5e-3	1e-3	5e-4
relu	0.0011	0.0006	0.0003	0.0025
tanh	0.0006	0.0002	3e-5	0.0391
sigmoid	0.0001	3e-5	0.0428	0.0517

@2: width = 13, depth = 3:

testing loss	1e-2	5e-3	1e-3	5e-4
relu	0.0001	0.0002	0.0011	0.0021
tanh	0.0004	0.0002	0.0002	0.0002
sigmoid	0.0170	3e-5	0.0011	0.0177

@3: width = 10, depth = 4:

testing loss	1e-2	5e-3	1e-3	5e-4
relu	0.0005	0.0002	0.0004	0.0007
tanh	0.0039	0.0207	0.0002	9e-5
sigmoid	0.0173	0.0174	0.0162	0.0186

【超参数分析】

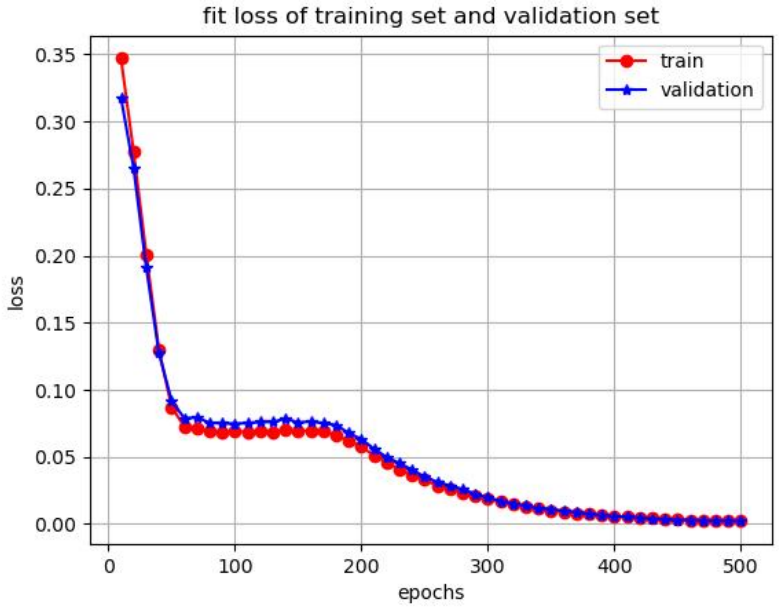
诚实地说，实验结果并没有表现出对某组超参数有明显的倾向规律。但是观察实验结果我们可以得到一些显然的结论：

@1:在网络结构固定时：学习率和激活函数都有可能对模型的训练产生影响。

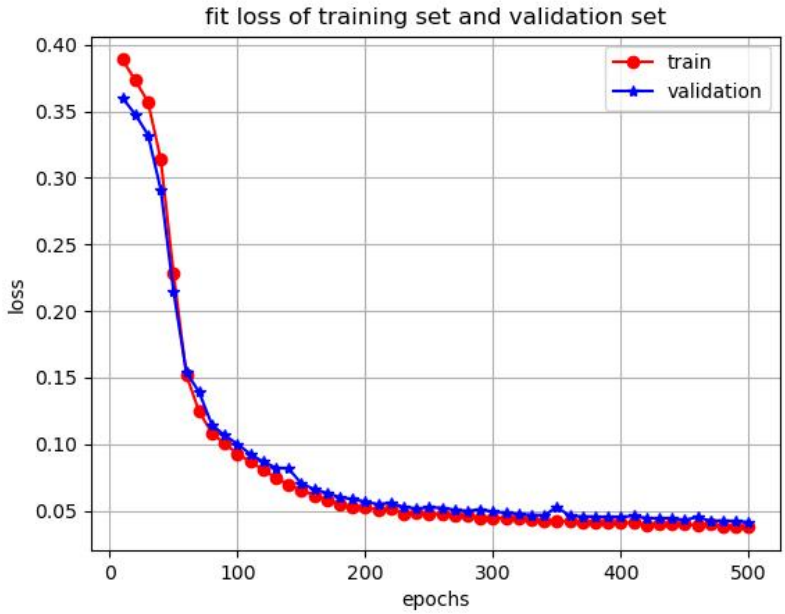
@2:综合来看，对于该回归任务，在几乎所有模型结构和学习率下，选用 relu 激活函数得到的在测试集合上的拟合性能（MSE loss）几乎是最为稳定 (一直处于 1e-4 级别的一个 loss 值) 的 (相比较于 tanh 和 sigmoid），tanh 激活函数的性能在绝大多数情况下也比较稳定，只有少数情况会收敛到局部最优, 但是 sigmoid 函数的性能不是很稳定，经常收敛到局部最优处，具体可以观察下面给出的【width=20, depth=2；lr=5e-4】情形下不同激活函数的训练曲线对比：

（relu 训练到最优，sigmoid 和 tanh 都收敛到局部最优）

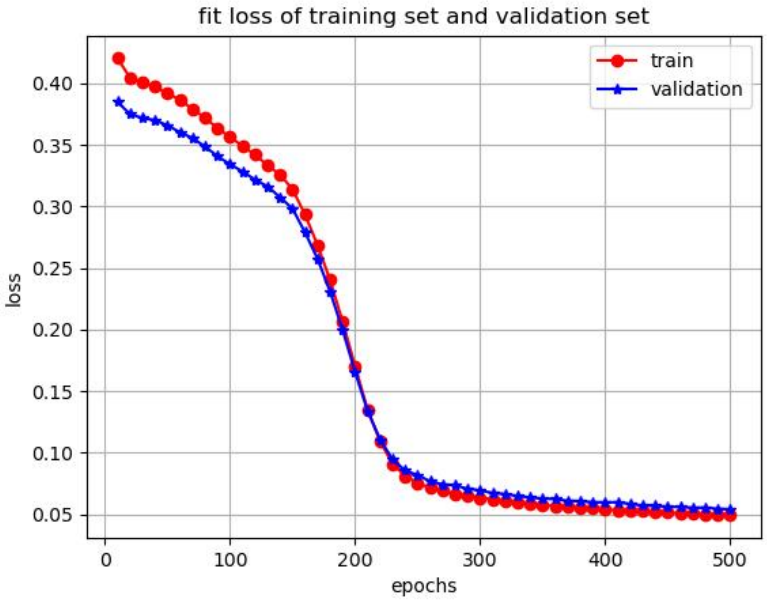
@1:relu



@2:tanh



@3:sigmoid


fit loss of training set and validation set
train
validation
loss
epochs
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0
100
200
300
400
500