A Reprot for Deep Learning Lab Exercise 5

Ziyi Guo, MSc Data Science

zg2u21@sonton.ac.uk

1 Simple CNN baseline for image regression

In the first part of the experiment, a simple CNN network is built to implement a regression task of predicting the best fitting line parameters for a target scatter plot. The simple CNN network is built and the predictions are visualized as below.

```
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN,self).__init__()
        self.conv1 = nn.Conv2d(in_channels = 1, out_channels = 48,
                               kernel_size = (3, 3), stride=1,padding=1)
        self.fc1 = nn.Linear(48 * 40**2, 128)
        self.fc2 = nn.Linear(128, 2)
    def forward(self, x):
        x = x.to(device)
        out = self.conv1(x)
        out = F.relu(out)
        out = out.view(out.shape[0], -1)
        out = self.fc1(out)
        out = F.relu(out)
        out = self.fc2(out)
        return out

model = SimpleCNN()
loss function = nn.MSELoss() # Define Loss Function
optimiser = optim.Adam(model.parameters())
device = "cuda:0" if torch.cuda.is_available() else "cpu"
trial = Trial(model, optimiser, loss_function, metrics=['loss'])
trial.with generators(trainloader, val generator=validloader,
                      test_generator=testloader).to(device)
trial.run(epochs=100)
results = trial.evaluate(data_key=torchbearer.TEST_DATA)
```

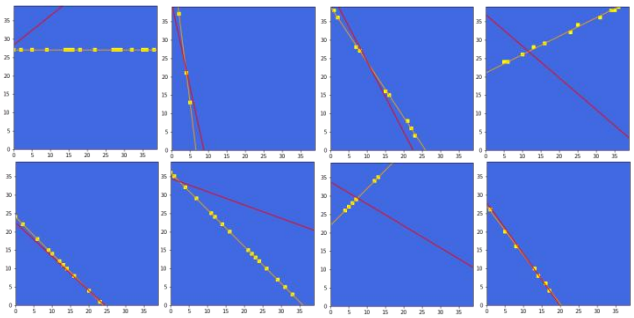

Fig 1. Predictions of Simple CNN

For a regression task, ***MSE*** is the proper choice of loss function to describe the difference between the real and the estimated parameters. As is computed above, the simple CNN outputs a prediction where the test ***MSE*** is 10.5459. According to the several visualization figures of the regression tasks above where the oranges line show the real parameters used to generate the yellow data points in each, quite biased estimations can be observed in most cases for both slopes and intercepts. Therefore, the model performs poorly.

2 Simple CNN with Global Pooling

In the second part of the experiment, a Global Max Pooling layer is added before the output layer to reduce the parameters as below.

```
class CNNPooling(nn.Module):
    def __init__(self):
        super(CNNPooling, self).__init__()
        self.conv1 = nn.Conv2d(in_channels = 1, out_channels = 48,
                               kernel_size = (3, 3), stride=1,padding=1)
        self.conv2 = nn.Conv2d(in_channels = 48, out_channels = 48,
                               kernel_size = (3, 3), stride=1,padding=1)
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(48, 128)
        self.fc2 = nn.Linear(128, 2)
    def forward(self, x):
        x = x.to(device)
        out = self.conv1(x)
        out = F.relu(out)
        out = self.conv2(out)
        out = F.relu(out)
        out = F.adaptive_max_pool2d(out,output_size=(1,1))
        out = self.flatten(out)
        out = self.fc1(out)
        out = F.relu(out)
        out = self.fc2(out)
        return out
```

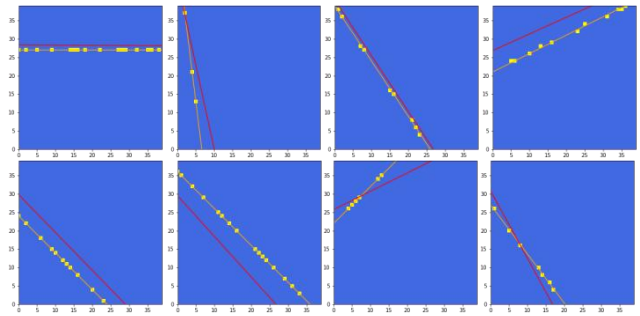

Fig 2. Predictions of Simple CNN with Global Max Pooling

For this time, the ***MSE*** becomes 16.5643 and the visualization figures actually shows greater bias in estimation as well. The pooling layer may reduce the learning degree of features in the training process. In all, both the first two models are hard to fit the real data and present issues in the regression task.

3 CoordConv solution to CNN fault

In the last part of the experiment, the approach known as CoordConv is taken to fix the model and the visualizations are as below.

```
class CNN3(nn.Module):
    def __init__(self):
        super(CNN3, self).__init__()
        self.conv1 = nn.Conv2d(in_channels = 3, out_channels = 48,
                               kernel_size = (3, 3), stride=1,padding=1)
        self.conv2 = nn.Conv2d(in_channels = 48, out_channels = 48,
                               kernel_size = (3, 3), stride=1,padding=1)
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(48, 128)
        self.fc2 = nn.Linear(128, 2)
    def forward(self, x):
        x = x.to(device)
        idxx = torch.repeat interleave(torch.arange(-20,20,dtype=torch.float).unsqueeze(0)/40.0,
                                       repeats = 40,dim = 0).to(x.device)
        idxy = idxx.clone().t()
        idx = torch.stack([idxx,idxy]).unsqueeze(0)
        idx = torch.repeat_interleave(idx,repeats=x.shape[0],dim=0)
        x = torch.cat([x,idx],dim=1)to [40,3,40,40]
        out = self.conv1(x)
        out = F.relu(out)
        out = self.conv2(out)
        out = F.relu(out)
        out = F.adaptive_max_pool2d(out,output_size=(1,1))
        out = self.flatten(out)
        out = self.fc1(out)
        out = F.relu(out)
        out = self.fc2(out)
        return out
```

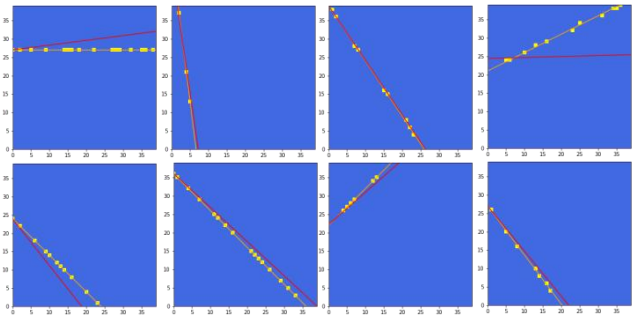

Fig 3. Predictions of Simple CNN with CoordConv

On implementing the correction, the model performs an ***MSE*** of 1.3076, which indicates a much better performance than the previous models. From the visualization figures, the third model generally makes more acceptable predictions in most cases in spite of existing bias and fits well with slopes and intercepts is close. This is due to the position which is made known to an applied convolutional kernel by CoordConv using data on other channels to hardcode data on the cartesian coordinate, and breaking the translation equivariance of normal convolution. Further, the operation process of CNN may not match normal linear regression tasks based on the MSE between data and the estimation.