LAB 5 - A LITTLE LINEAR REGRESSION

Supritha Konaje - 32864477
ssk1n21@soton.ac.uk

1. AN INITIAL ATTEMPT

1.1. A simple CNN baseline

The implementation for Simple CNN model is given in the following code snippet.

```
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 48, (3, 3), stride = 1, padding=1)
        self.fc1 = nn.Linear(48 * 40**2, 128)
        self.fc2 = nn.Linear(128, 2)

    def forward(self, x):
        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
```

The loss function used was:

```
loss_function = torch.nn.CrossEntropyLoss()
```

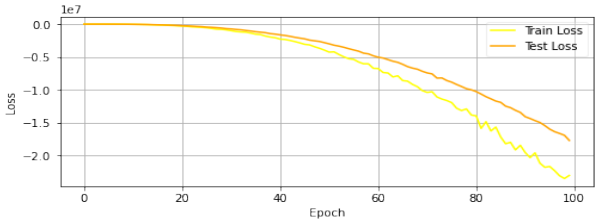

Fig. 1. Loss vs Epoch curve for Simple CNN

The test loss and train loss vary with some divergence after few epochs. Hence, this model is a good model but cannot be considered as a stable model. Also, the number of parameters in this hidden layers are more in this model.

2. A SECOND ATTEMPT

2.1. A simple CNN with global pooling

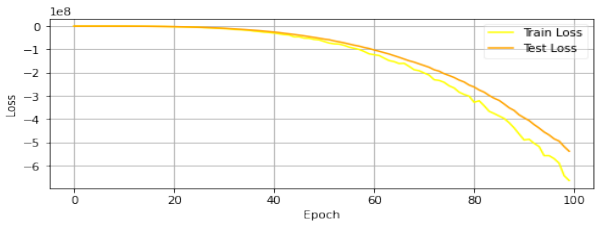

Fig. 2. Loss vs Epoch curve for Simple CNN with Global Pooling

The number of parameters in the hidden layer was reduced. The training loss and test loss had less divergence but the performance was a bit unstable.

The code snippet for the 2rd model is given below:

```
class SimpleCNNWithGlobalPooling(nn.Module):
    def __init__(self):
        super(SimpleCNNWithGlobalPooling, self).__init__()
        self.conv1 = nn.Conv2d(1, 48, (3, 3), stride = 1, padding=1)
        self.conv2 = nn.Conv2d(48, 48, (3, 3), stride = 1, padding=1)
        self.globalmaxpool1 = nn.AdaptiveMaxPool2d((1, 1))
        self.fc1 = nn.Linear(48 * 1**2, 128)
        self.fc2 = nn.Linear(128, 2)

    def forward(self, x):
        out = self.conv1(x)
        out = F.relu(out)
        out = self.conv2(out)
        out = F.relu(out)
        out = self.globalmaxpool1(out)
        out = out.view(out.shape[0], -1)
        out = self.fc1(out)
        out = F.relu(out)
        out = self.fc2(out)
        return out
```

3. SOMETHING THAT ACTUALLY WORKS?

3.1. Let's regress

This model has performed better as compared to the other two models. Hence, this model is definitely a better version. The channel data is high in variance and this has taken a transpose of the original data channel. The interleaved channel helps to learn better when a data with high variance is given. And hence, the performance of this model is better when compared to the other models.

The code snippet for the 3rd model is given below:

```
class RegressCNN(nn.Module):
    def __init__(self):
        super(RegressCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 48, (3, 3), stride = 1, padding=1)
        self.conv2 = nn.Conv2d(48, 48, (3, 3), stride = 1, padding=1)
        self.globalmaxpool1 = nn.AdaptiveMaxPool2d((1, 1))
        self.fc1 = nn.Linear(48 * 1**2, 128)
        self.fc2 = nn.Linear(128, 2)

    def forward(self, x):
        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)

        out = self.conv1(x)
        out = F.relu(out)
        out = self.conv2(out)
        out = F.relu(out)
        out = self.globalmaxpool1(out)
        out = out.view(out.shape[0], -1)
        out = self.fc1(out)
        out = F.relu(out)
        out = self.fc2(out)
        return out
```

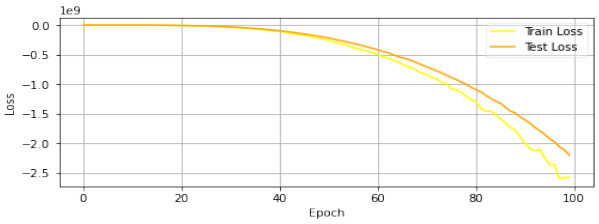

Fig. 3. Loss vs Epoch curve for Regress CNN