CS764: Assignment 7

Imputing 3D from 2D

Parth Shettiwar 170070021

1 Problem Statement

In this assignment, we had to predict 3D coordinates of the skeleton for a human given its 2D skeleton coordinates. The prediction had to be made using a deep learning network consisting of 2 residual blocks.

2 Dataset

The dataset consisted of 78047 sample points, each giving the 3D as well as 2D coordinates of sizes (15,3) and (15,2) respectively. A `data.py` was created to split the input dataset into train test. I used the conventional 80:20 split for train:test respectively (based on [1]). Finally dataset was saved in `data_trainN_lift.pkl` and `data_test_lift.pkl` which will be used in training and testing respectively. So given 30 input features (15*2), we should produce the (15*3) output 3D coordinates.

3 Model

The model used was the one mentioned in the document. Specifically following image shows the model summary:

```
LiftModel(
  (hidden1): Linear(in_features=30, out_features=1024, bias=True)
  (res-block_0): Sequential(
    (0): Linear(in_features=1024, out_features=1024, bias=True)
    (1): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=1024, out_features=1024, bias=True)
    (5): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU()
    (7): Dropout(p=0.5, inplace=False)
  )
  (res-block_1): Sequential(
    (0): Linear(in_features=1024, out_features=1024, bias=True)
    (1): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=1024, out_features=1024, bias=True)
    (5): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU()
    (7): Dropout(p=0.5, inplace=False)
  )
  (output): Linear(in_features=1024, out_features=45, bias=True)
)
```

Before the 1st residual block, a linear layer was added which converts 30 input features to 1024. Similarly after the second residual block, linear layer is added which converts 1024 to 45 output features. The class `LiftModel` was made for the same.

4 Training and Data loader

Firstly a train loader was made using the torch.Dataset library. Shuffle was kept to be true and batch size = 32. The class `SkeletonData` was specifically created for dataset to implement the len and getitem functions.
Adam optimizer is used with initial lr=.1 and betas=(0.5, 0.999). The StelLR is used as the scheduler with step size=200 and gamma=0.3. Above values are the standard used for training purposes. Dropout = 0.5 was used.
Training was done for 5 epochs on Google Colab GPU and is finished under a minute always. The best model weights are saved.
The loss function was implemented as mentioned in the assignment. Euclidean distance is taken and averaged out along both 1st and 2nd axis and final loss is printed as averaged over the length of data loader. Following is the training process happening.

```
loss for epoch 0  =  tensor(1.4696, device='cuda:0', grad_fn=<DivBackward0>)
100% 1951/1951 [00:07<00:00, 259.73it/s]

loss for epoch 1  =  tensor(0.0681, device='cuda:0', grad_fn=<DivBackward0>)
100% 1951/1951 [00:07<00:00, 256.48it/s]

loss for epoch 2  =  tensor(0.0681, device='cuda:0', grad_fn=<DivBackward0>)
100% 1951/1951 [00:07<00:00, 252.41it/s]

loss for epoch 3  =  tensor(0.0682, device='cuda:0', grad_fn=<DivBackward0>)
100% 1951/1951 [00:07<00:00, 258.24it/s]

loss for epoch 4  =  tensor(0.0682, device='cuda:0', grad_fn=<DivBackward0>)
```

The training loss is reported to be about 0.0682.

5 Testing

A seperate `eval.py` script was written. In this again similarly to the train loader, test loader object was created. The best model weights are loaded. The same loss function was used and output loss value is printed on screen on the dataset.

```
average test loss tensor(0.0648, device='cuda:0')
```

The test loss is reported to be about 0.0648.

6 Questions

Q1 Explain the purpose and location of the batch normalization later, and the dropout layer. What is the reason for duplicating the layer. What if we used three blocks?
The BatchNorm layer was added after every linear layer to ensure that the inputs are always unit/scaled gaussian normalised and the distribution of input is always same when feeded to the next linear layer. This is important since otherwise each layer would receive inputs from various distribution and it will be difficult for training as the gradients have to be back propagated.
The dropout layer is added in end to avoid over fitting as it randomly drops some neurons. It is usually added in the end as proposed by [4]. The intution we want to randomly clear out the effects of whole combined linear+activation process by randomly zero-ing out some values and hence effects of weights. However as mentioned by [5], in case of ReLu activation, it doesnt matter whether dropout

is applied in end or before activation function. Number of blocks dont matter much here, as I perofrm an experiment in next section. However if we use many blocks then overfitting possibilty is there.

Q2 **Provide details of your training. What experiments did you conduct? Show charts.** The details of training were mentioned in Training section before. The following experiments were conducted to know the effect of some hyper-parameters:

Change of Dropout

Dropout	loss (x 1e-4)
0	683
0.5	649
0.9999	38686

As we see, with very high and low dropouts, loses are more than optimal loss observed at dropout = 0.5.

Change of Learning Rate

Learning Rate	loss (x 1e-4)
0.001	1371
0.1	649
1	1270

Again very high and very low learning rate give higher test loses. 0.1 was optimal learing rate.

Change of Hidden neurons

Hidden Layer neurons	loss (x 1e-4)
256	663
512	641
1024	649

Almost similar loss values are observed with relatively better loss values in 1024 and 512 neurons.

Change of number of Residual Blocks

Number of Blocks	loss (x 1e-4)
1	657
2	649
3	653

Again similar values are noted irrespective if number of residual blocks used.

References

[1] https://en.wikipedia.org/wiki/Pareto_principle

[2] https://pytorch.org/tutorials/beginner/saving_loading_models.html

[3] https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html#sphx-glr-beginner-blitz-cifar10-tutorial-py

[4] https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

[5] https://sebastianraschka.com/faq/docs/dropout-activation.html