A Reprot for Deep Learning Lab Exercise 2

Ziyi Guo, MSc Data Science

zg2u21@sonton.ac.uk

1 Gradient-based matrix factorisation in PyTorch

In the first part of the experiment, a gradient-based factorisation using PyTorch's AD framework is implemented as below:

```
def gd_factorise_ad(A: torch.Tensor, rank:int, num_epochs=1000, lr=0.01) -> Tuple[torch.Tensor, torch.Tensor]:
    m,n = A.size()
    r = rank
    U = torch.randn((m,r))
    V = torch.randn((n,r))
    U = U.double()
    V = V.double()
    U = U.clone().detach().requires_grad_(True)
    V = V.clone().detach().requires_grad_(True)
    for epoch in range(num_epochs):
        Error = torch.nn.functional.mse_loss(U@V.t(),A,reduction ='sum')
        Error.backward()
        U.data -= lr*U.grad
        V.data -= lr*V.grad
        U.grad.zero_()
        V.grad.zero_()
        if epoch%10==0:
            print("Epoch:",epoch,"Error:",Error)
    return U, V
```

Illustrating a real dataset Iris of 150 instances and 4 features into a mean-centered tensor as the input matrix, and defining the factorisation to be rank-2, the algorithm outputs a reconstruction error of 15.2288. With the same input data, however, the rank-2 matrix reconstruction algorithm based on truncated SVD proposed in the last report outputs a reconstruction error of 3.5288, indicating a quite better performance in this real circumstance.

Further exploring the relationship between the two algorithms, the scatter plots of the data in matrix $\boldsymbol{U}_1$ output by matrix factorisation and the first two principle axes of the data in matrix $\boldsymbol{U}_2$ output by SVD are created as below.

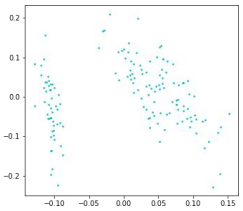

Fig 1. Data Distribution in Matrix $\boldsymbol{U}$ of Gradient-based Factorisation

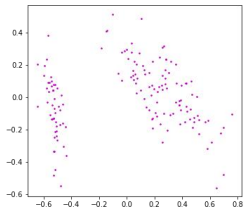

Fig 2. First Two Principle Data Distribution in Matrix $\boldsymbol{U}$ of SVD

It can be clearly observed that the data of the two matrices are in the same or really similar distributions, which means that when applying gradient-based matrix factorisation to minimise reconstruction error, the final factorisation result of matrix $\boldsymbol{U}$ that minimises the reconstruction error is approximating what SVD outputs. It can be inferred that the application of PCA on data based on SVD, maximising variance of data on orthogonal directions, actually gives the matrix factorisation that minimises reconstruction error.

2 Implementation and test for a simple MLP

In the second part of the experiment, a simple MLP trained with gradient descent using the AD framework is implemented as below:

```
def MLP(Data,Target,num_epochs = 100,lr = 0.01):
    W1 = torch.randn(4,12)     # Paras Initialization
    W2 = torch.randn(12,3)
    W1 = W1.float()
    W2 = W2.float()
    W1 = W1.clone().detach().requires_grad_(True)
    W2 = W2.clone().detach().requires_grad_(True)
    b1 = torch.tensor(0.0,requires_grad=True)
    b2 = torch.tensor(0.0,requires_grad=True)
    for epoch in range(num_epochs):
        logits = torch.relu(Data @ W1 + b1)@ W2 + b2              # MLP Function Definition
        Error = torch.nn.functional.cross_entropy(logits,Target)  # Error Function
        Error.backward()
        W1.data -= lr*W1.grad  # Gradient-based Updates
        W2.data -= lr*W2.grad
        b1.data -= lr*b1.grad
        b2.data -= lr*b2.grad
        if epoch%10==0 :
            print("Epoch:",epoch,"Error:",Error)
    W1e = W1.data
    W2e = W2.data
    b1e = b1.data
    b2e = b2.data
    output  = torch.relu(Data @ W1e + b1e)@ W2e + b2e             # Final Output
    pred = output.argmax(dim=1)                                   # Target Prediction
    Accuracy = accuracy_score(pred, Target)                       # Accuracy
    return Accuracy,Error
```

Illustrating the same Iris dataset and dividing the data into training and validation set of size 100 and 50, experiments are carried out on both sets for a number of times. The results are in the table as below.

Experiment	1		2		3		4		5		6		7		8		9		10	
	Acc	Loss	**Acc**	Loss	**Acc**	Loss	**Acc**	Loss	**Acc**	Loss	**Acc**	Loss	**Acc**	Loss	**Acc**	Loss	**Acc**	Loss	**Acc**	Loss
Training	**0.83**	0.31	**0.81**	0.66	**0.87**	0.52	**0.86**	0.54	**0.87**	0.46	**0.99**	0.24	**0.5**	1.21	**0.86**	0.55	**0.95**	0.40	**0.88**	1.28
Validation	**0.86**	0.33	**0.94**	0.27	**0.86**	0.49	**0.96**	0.37	**0.94**	0.37	**0.84**	0.46	**0.96**	0.51	**0.62**	0.72	**0.98**	0.26	**0.9**	0.31

Table 1. MLP Accuracy and Loss on Training and Validation Set

It can be observed that there is generally no obvious gap between training and validation accuracies in most cases except the extreme situations and both accuracies are generally high while the validation accuracy is more likely to be higher, indicating the efficiency and stability of the MLP algorithm on this data set.