Deep Learning - Lab 2 Exercise

Han Zhang hz5g21@soton.ac.uk

1 Exercise 1

Exercise 1.1:Implement gradient-based factorisation using PyTorch's AD.

```
from typing import Tuple
def gd_factorise_ad(A: torch.Tensor, rank: int,
    num_epochs: int = 1000, lr =0.01) ->
    Tuple[torch.Tensor, torch.Tensor]:
    (m, n) = A.shape
    U = torch.rand((m, rank), requires_grad=True,
        dtype=torch.double)
    V = torch.rand((n, rank), requires_grad=True,
        dtype=torch.double)
    for epoch in range(num_epochs):
        A_sgd_hat = U @ V.t()
        loss = torch.nn.functional.mse_loss
        (A_sgd_hat, A, reduction="sum")
        loss.backward(torch.ones(loss.shape))
        with torch.no_grad():
            U -= lr * U.grad
            V -= lr * V.grad
        U.grad.zero_()
        V.grad.zero_()
    return U.detach(), V.detach()
```

Exercise 1.2:Factorise and compute reconstruction error on real data.
The reconstruction loss of gradient-based factorisation is 15.2817. The reconstruction loss of SVD factorisation of the same data is 15.2288. This implies that those two methods have simliar results and SVD factorisation is slightly better.

Exercise 1.3:Compare against PCA.

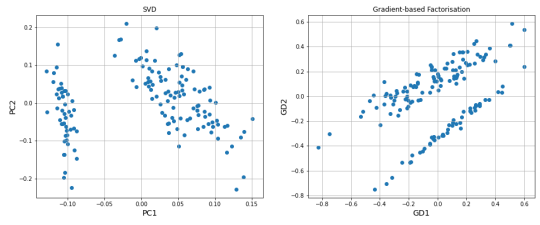

Figure 1: PCA and GD comparison

From Figure 1it can be observed that the results of PCA and gradient-based factorisation are very similar. Rotating and scaling one of the findings appears to generate the other. There are two types of PCA algorithms, one is to maximizes variance and the other is to maximizes reconstruction error. Both methods will give the same result for PCA. Meanwhile maximizing the reconstruction error is the loss function used by gradient-based factorisation. It can be inferred that through a mapping relationship, gradient-based factorisation and PCA can achieve the same effect.

2 Exercise 2

Exercise 2.1:Implement the MLP

```
def sgd_mlp(tr_data: Tensor, targets_tr: Tensor,
    num_epochs: int = 100, lr: float = 0.01
) -> Tuple[Tensor, Tensor, Tensor, Tensor]:
    W1 = torch.randn((4, 12), requires_grad=True)
    W2 = torch.randn((12, 3), requires_grad=True)
    b1 = torch.zeros(1, requires_grad=True)
    b2 = torch.zeros(1, requires_grad=True)

    for epoch in range(num_epochs):
        logits = mlp_func(tr_data, W1, W2, b1, b2)
        cross_entropy =
            torch.nn.functional.cross_entropy(
            logits, targets_tr, reduction="sum"
        )
        cross_entropy.backward()
        with torch.no_grad():
            W1 -= lr * W1.grad
            W2 -= lr * W2.grad
            b1 -= lr * b1.grad
            b2 -= lr * b2.grad
        for zv in [W1, W2, b1, b2]:
            zv.grad.zero_()

    return W1.detach(), W2.detach(), b1.detach(),
        b2.detach()
```

Exercise 2.2:Test the MLP

	Iris setosa	Iris versicolor	Iris virginica
Train	1.0000	0.7273	1.0000
validation	0.9375	0.7647	1.0000

The results show in the above table. The overall accurate rates are 0.9000 and 0.8800. The results show that the Iris virginica and Iris setosa have higher accuracy in the training and validation, compared to the Iris versicolor. This implies that Iris virginica and Iris setosa are highly separable while Iris versicolor is relatively inseparable.