COMP6248: Lab Exercise 2

David Jones (dsj1n15@soton.ac.uk)

Task: PyTorch Autograd

1 Exercise 1

Exercise 1.1: Implementation of matrix factorisation using gradient descent.

```python
def gd_factorise_ad(
    A: Tensor, rank: int, num_epochs: int = 1000, lr: float = 0.01
) -> Tuple[Tensor, 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()
```

Listing 1: Matrix factorisation using gradient descent with PyTorch's AD.

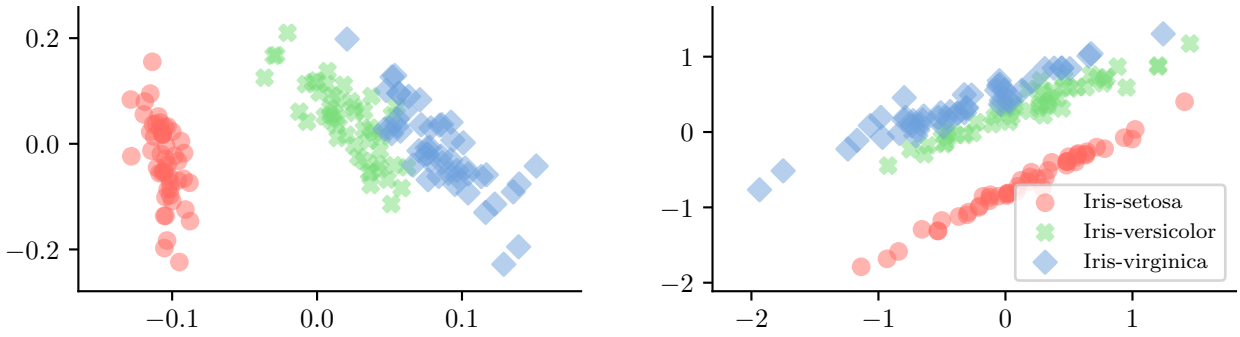
Exercise 1.2: Rank 2 reconstruction loss for Iris Dataset.

$$\mathbf{A} = \text{Iris Dataset} - \mu$$

$$\text{loss}_{\text{GD}} = \left\|\mathbf{A} - \hat{\mathbf{A}}_{\text{GD}}\right\|_{\text{F}}^2 = 15.232 \quad \text{loss}_{\text{SVD}} = \left\|\mathbf{A} - \hat{\mathbf{A}}_{\text{SVD}}\right\|_{\text{F}}^2 = 15.229$$

As expected loss_{SVD} is less than loss_{GD}, yet $\hat{\mathbf{A}}_{\text{GD}}$ is a close approximation of the optimum.

Exercise 1.3: PCA vs GD Rank Reduction.



(a) PCA

(b) GD

Figure 1: Rank 2 reduction on the Iris Dataset

The groupings of the data factorised by PCA and GD are very similar. It appears that a linear rotation and scaling transformation would map one to the other. A consequence of PCA maximising the variance

is that the reconstruction error will be minimised – this is the loss function being used by the GD. Although the mapped values are not identical, GD is not being applied to a multi-layer perceptron so it can at most learn a linear mapping of PCA.

2 Exercise 2

Exercise 2.1: Multi-Layer Perceptron (MLP).

```python
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()
```

Listing 2: PyTorch MLP Classifier for Iris Dataset

Exercise 2.2: MLP classification on Iris Dataset.
Below are the median train and validation results for classifying the Iris Dataset across 100 independent trainings. The order of class results are $\left[\text{Iris setosa}, \text{Iris versicolor}, \text{Iris virginica}\right]$.

$$\begin{aligned} \text{Acc}_{\text{train}} &= \begin{bmatrix} 1.000 & 0.806 & 0.969 \end{bmatrix} \quad | \quad \text{Overall} = 0.92 \\ \text{Acc}_{\text{validation}} &= \begin{bmatrix} 1.000 & 0.786 & 0.889 \end{bmatrix} \quad | \quad \text{Overall} = 0.90 \end{aligned}$$

This lines up with the rank reduction which shows Iris setosa is highly separable from the other classes. The results show that differentiating between Iris versicolor and Iris virginica is still not always possible even with the higher rank information. Validation accuracy is less for all groups, indicating either a lack of required diversity in the training set or overfitting.