COMP6248: Lab Exercise 1

David Jones (dsj1n15@soton.ac.uk)

Task: Gradients and matrices in PyTorch

1 Exercise 1

Exercise 1.1: Implementation of matrix factorisation using gradient descent.

```python
def sgd_factorise(
    A: Tensor, rank: int, num_epochs: int = 1000, lr: float = 0.01
) -> Tuple[Tensor, Tensor]:
    (m, n) = A.shape
    U = torch.randn((m, rank))
    V = torch.randn((n, rank))
    for epoch in range(num_epochs):
        e = A - U @ V.t()
        for r in range(m):
            for c in range(n):
                e = A[r, c] - U[r] @ V[c].t()
                U[r] += lr * e * V[c]
                V[c] += lr * e * U[r]
    return U, V
```

Listing 1: Matrix factorisation using stochastic gradient descent (SGD).

Exercise 1.2: Rank 2 reconstruction loss for input **A**.

$$\mathbf{A} = \begin{bmatrix} 0.337 & 0.601 & 0.174 \\ 3.336 & 0.049 & 1.837 \\ 2.941 & 0.530 & 2.262 \end{bmatrix} \quad \hat{\mathbf{U}} = \begin{bmatrix} -0.063 & -0.906 \\ -2.387 & 0.683 \\ -2.059 & -0.707 \end{bmatrix} \quad \hat{\mathbf{V}} = \begin{bmatrix} -1.414 & -0.149 \\ -0.131 & -0.500 \\ -0.916 & -0.346 \end{bmatrix}$$

$$\hat{\mathbf{A}} = \hat{\mathbf{U}}\hat{\mathbf{V}}^\top = \begin{bmatrix} 0.224 & 0.462 & 0.371 \\ 3.271 & -0.0301 & 1.950 \\ 3.016 & 0.623 & 2.131 \end{bmatrix} \quad \boxed{\text{loss} = \left\|\mathbf{A} - \hat{\mathbf{A}}\right\|_\text{F}^2 = 0.1257}$$

2 Exercise 2

Exercise 2.1: Matrix factorisation using Singular Value Decomposition (SVD).

```python
def svd_factorise(A: Tensor, rank: int) -> Tuple[Tensor, Tensor, Tensor]:
    U, S, V = torch.svd(A)
    S[rank:] = 0
    return (U, torch.diag(S), V)
```

Listing 2: Matrix factorisation using SVD.

$$\hat{\mathbf{U}} = \begin{bmatrix} -0.080 & -0.745 & 0.663 \\ -0.710 & 0.509 & 0.486 \\ -0.699 & -0.432 & -0.570 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 5.334 & 0.000 & 0.000 \\ 0.000 & 0.696 & 0.000 \\ 0.000 & 0.000 & \cancel{0.349}\nearrow{}^{0} \end{bmatrix} \quad \hat{\mathbf{V}} = \begin{bmatrix} -0.835 & 0.255 & 0.488 \\ -0.085 & -0.936 & 0.343 \\ -0.544 & -0.245 & -0.803 \end{bmatrix}$$

$$\hat{\mathbf{A}} = \hat{\mathbf{U}}\mathbf{\Sigma}\hat{\mathbf{V}}^\top = \begin{bmatrix} 0.225 & 0.521 & 0.359 \\ 3.253 & -0.0090 & 1.974 \\ 3.038 & 0.598 & 2.102 \end{bmatrix} \quad \boxed{\text{loss} = \left\|\mathbf{A} - \hat{\mathbf{A}}\right\|_\text{F}^2 = 0.1219}$$

The reconstruction loss of the SVD approach is less than that of the SGD approach. This is explained by the Eckart-Young theorem, which states that a low-rank approximation created using SVD is optimum for that rank.

3 Exercise 3

Exercise 3.1: Masked matrix factorisation.

```python
def sgd_factorise_masked(
    A: Tensor, M: Tensor, rank: int, num_epochs: int = 1000, lr: float = 0.01
) -> Tuple[Tensor, Tensor]:
    (m, n) = A.shape
    U = torch.randn((m, rank))
    V = torch.randn((n, rank))
    for epoch in range(num_epochs):
        e = A - U @ V.t()
        for r in range(m):
            for c in range(n):
                if M[r, c]:
                    e = A[r, c] - U[r] @ V[c].t()
                    U[r] += lr * e * V[c]
                    V[c] += lr * e * U[r]
    return U, V
```

Listing 3: Masked matrix factorisation using stochastic gradient descent (SGD).

Exercise 3.2: Reconstructing a matrix using masked factorisation.

$$\mathbf{A} = \begin{bmatrix} 0.337 & 0.601 & 0.174 \\ \cancel{3.336} & 0.049 & 1.837 \\ 2.941 & \cancel{0.530} & 2.262 \end{bmatrix} \qquad \hat{\mathbf{U}} = \begin{bmatrix} -0.216 & 0.2940 \\ -1.544 & 0.4287 \\ -1.769 & 0.7095 \end{bmatrix} \qquad \hat{\mathbf{V}} = \begin{bmatrix} -2.025 & -0.882 \\ 0.463 & 1.815 \\ -0.993 & 0.680 \end{bmatrix}$$

$$\hat{\mathbf{A}} = \hat{\mathbf{U}}\hat{\mathbf{V}}^\top = \begin{bmatrix} 0.178 & 0.433 & 0.414 \\ \boxed{2.749} & 0.064 & 1.825 \\ 2.955 & \boxed{0.470} & 2.239 \end{bmatrix} \qquad \text{loss} = \left\|\mathbf{A} - \hat{\mathbf{A}}\right\|_F^2 = 0.4610$$

Both of the reconstructed values in $\hat{\mathbf{A}}$ are close to their original values in $\mathbf{A}$. This indicates that information defining their value was carried in the other values of the matrix. Some of this information is captured in the reduced rank approximation and projected back to create the masked values.