# Lab 1 Exercise – Playing with gradients and matrices in PyTorch

Justin Ugwudike (jknu1g19@soton.ac.uk)

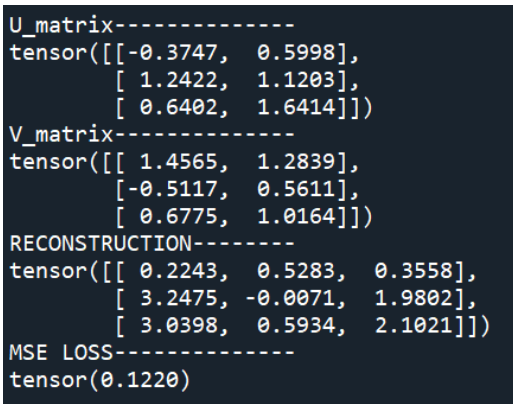
Feb 20, 2023

## 1.1 Question 1

$$\begin{bmatrix} 0.3374 & 0.6005 & 0.1735 \\ 3.3359 & 0.0492 & 1.8374 \\ 2.9407 & 0.5301 & 2.2620 \end{bmatrix}$$

Figure 1 – The target matrix

Using the code shown in Appendix A, the rank-2 factorisation of the matrix (shown in Figure 1) produced the results shown in Figure 2.



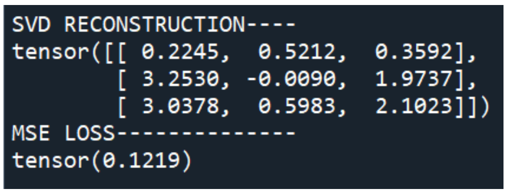
```
U_matrix--------------
tensor([[-0.3747,  0.5998],
        [ 1.2422,  1.1203],
        [ 0.6402,  1.6414]])
V_matrix--------------
tensor([[ 1.4565,  1.2839],
        [-0.5117,  0.5611],
        [ 0.6775,  1.0164]])
RECONSTRUCTION--------
tensor([[ 0.2243,  0.5283,  0.3558],
        [ 3.2475, -0.0071,  1.9802],
        [ 3.0398,  0.5934,  2.1021]])
MSE LOSS--------------
tensor(0.1220)
```

Figure 2 – The predicted U and V matrices as well as the reconstruction of the target matrix using U and V. The mean squared error loss is shown.

The code produces and approximation of the target matrix with the MSE loss being 0.1220.

## 1.2 Question 2

Using the function shown in Appendix B the Singular Value Decomposition of the matrix shown in Figure 1 was computed then the last singular value was set to zero after which the matrix is reconstructed using the equation for truncated SVD.



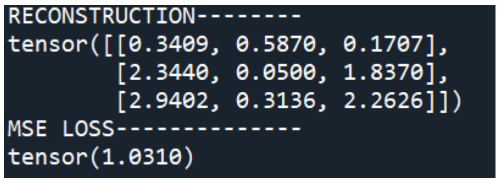
```
SVD RECONSTRUCTION----
tensor([[ 0.2245,  0.5212,  0.3592],
        [ 3.2530, -0.0090,  1.9737],
        [ 3.0378,  0.5983,  2.1023]])
MSE LOSS--------------
tensor(0.1219)
```

Figure 3 – results for question 2.

The reconstructed matrix has similar values to the matrix obtained using gradient descent this is reflected through the mean square error being 0.1219 which differs by -0.0001 from the error calculated in question 1. This is because this is a rank-2 approximation meaning that 2 rows of V transposed and 2 columns of U are being used, therefore the 3rd non-zero element of the singular value matrix is not used and setting it to zero should not affect the approximation.

## 1.3 Question 3

Modifying the function in Appendix A resulted in a new function shown in Appendix C. Using this function along with the mask shown in Appendix C produces the results in Figure 4:

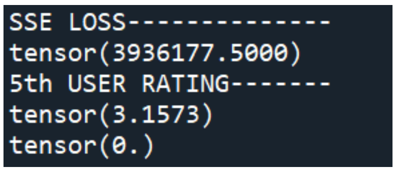


```
RECONSTRUCTION--------
tensor([[0.3409, 0.5870, 0.1707],
        [2.3440, 0.0500, 1.8370],
        [2.9402, 0.3136, 2.2626]])
MSE LOSS--------------
tensor(1.0310)
```

Figure 4 – the estimate of the completed matrix and MSE loss.

The values of the matrix where the corresponding values in the binary mask are 1, closely match the values of the target matrix shown in Figure 1. The predictions of the values in matrix which were masked with zeros are much less accurate. This implies that the less the parameters are altered according to different gradients during gradient descent, the more accurate the prediction of values which gradients were used to alter the parameters.

## 1.4 Question 4



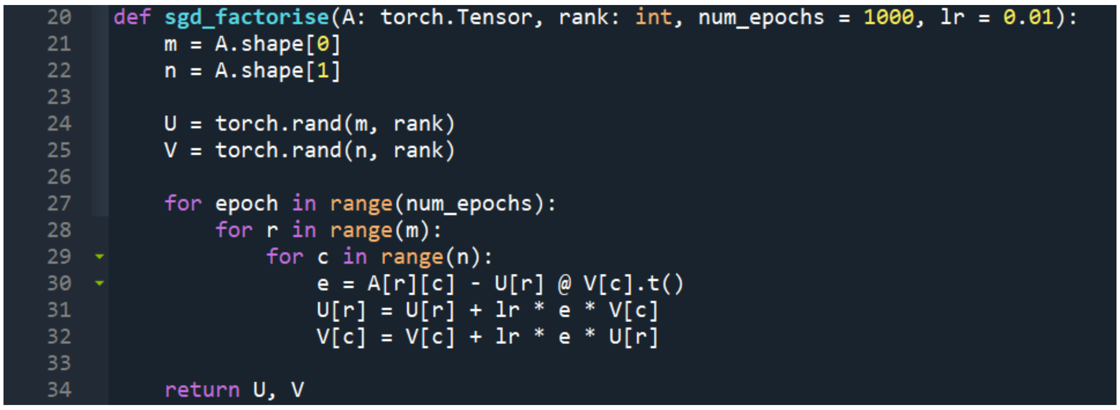
```
SSE LOSS--------------
tensor(3936177.5000)
5th USER RATING-------
tensor(3.1573)
tensor(0.)
```

Figure 5 – the MSE loss using masked gradient descent matrix factorisation and the prediction of the 5th user's rating of the 124th movie "A Beautiful Mind".

Running the code shown in Appendix D which utilises the provided gd_factorise_masked() function gives a prediction of 3.1573 out of 5 for the 5th user's rating of the film "A Beautiful Mind". The sum of squared errors (MSE) is a very large value: 3,936,177.

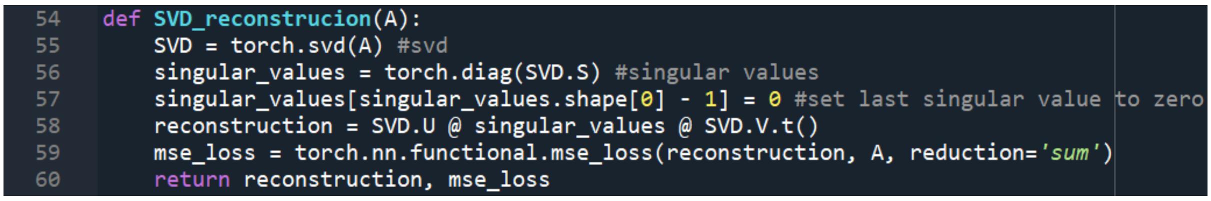
## 2 Appendix

Appendix A - Completed sgd_factorise() function.



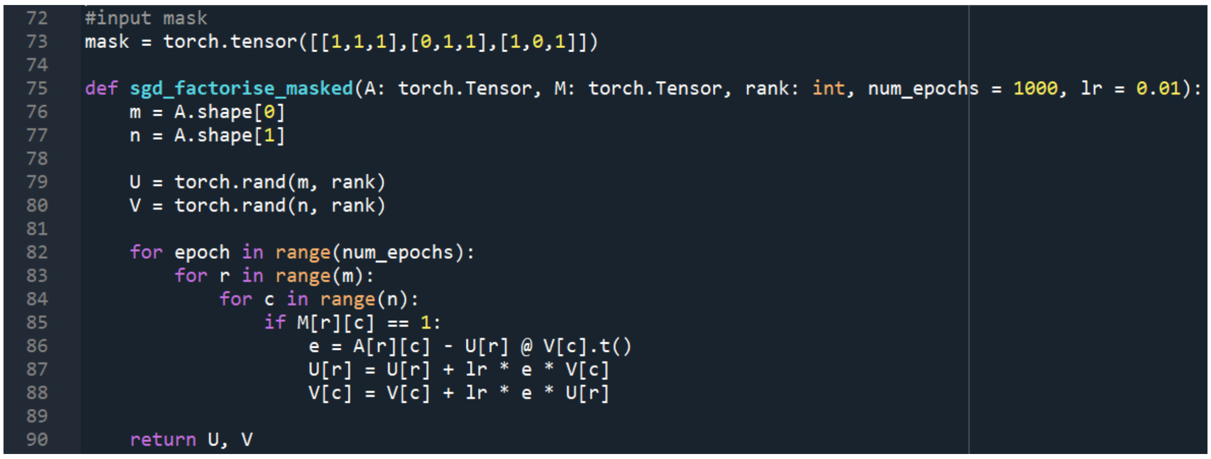
```python
def sgd_factorise(A: torch.Tensor, rank: int, num_epochs = 1000, lr = 0.01):
    m = A.shape[0]
    n = A.shape[1]

    U = torch.rand(m, rank)
    V = torch.rand(n, rank)

    for epoch in range(num_epochs):
        for r in range(m):
            for c in range(n):
                e = A[r][c] - U[r] @ V[c].t()
                U[r] = U[r] + lr * e * V[c]
                V[c] = V[c] + lr * e * U[r]

    return U, V
```

Appendix B – Function used to perform SVD, set the last singular value to zero and compute the reconstruction of an input matrix.



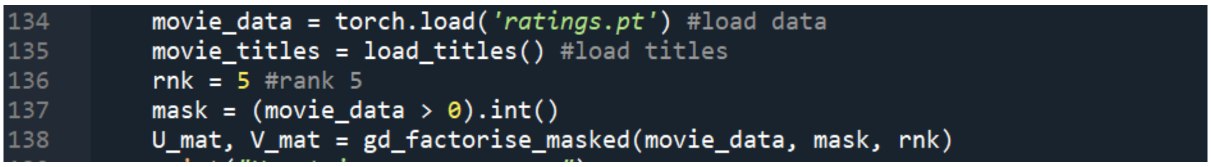
```python
def SVD_reconstrucion(A):
    SVD = torch.svd(A) #svd
    singular_values = torch.diag(SVD.S) #singular values
    singular_values[singular_values.shape[0] - 1] = 0 #set last singular value to zero
    reconstruction = SVD.U @ singular_values @ SVD.V.t()
    mse_loss = torch.nn.functional.mse_loss(reconstruction, A, reduction='sum')
    return reconstruction, mse_loss
```

Appendix C – Modified version of the sgd_factorise() function shown in Appendix A, this function now uses a binary mask. The mask used is also shown in the code below.



```python
#input mask
mask = torch.tensor([[1,1,1],[0,1,1],[1,0,1]])

def sgd_factorise_masked(A: torch.Tensor, M: torch.Tensor, rank: int, num_epochs = 1000, lr = 0.01):
    m = A.shape[0]
    n = A.shape[1]

    U = torch.rand(m, rank)
    V = torch.rand(n, rank)

    for epoch in range(num_epochs):
        for r in range(m):
            for c in range(n):
                if M[r][c] == 1:
                    e = A[r][c] - U[r] @ V[c].t()
                    U[r] = U[r] + lr * e * V[c]
                    V[c] = V[c] + lr * e * U[r]

    return U, V
```

Appendix D – Question 4 code. The binary masked used here omits the data points where the users' ratings are zero.



```python
    movie_data = torch.load('ratings.pt') #load data
    movie_titles = load_titles() #load titles
    rnk = 5 #rank 5
    mask = (movie_data > 0).int()
    U_mat, V_mat = gd_factorise_masked(movie_data, mask, rnk)
```