Lab 1 Exercise - Playing with gradients and matrices in PyTorch

1. A Low-rank Matrix Factorisation Using Gradient Descent

a. following the pseudocode I implemented the matrix factorisation algorithm which estimates a low-rank description of an input matrix. Then, with the provided 3×3 matrix sent to it, the function outputted $\widehat{U}$ and $\widehat{V}$. They were:

$$\begin{bmatrix} 0.6741 & -0.1215 \\ 0.2116 & 1.6963 \\ 0.9354 & 1.3231 \end{bmatrix} \ and \ \begin{bmatrix} 0.6635 & 1.8306 \\ 0.7727 & -0.0993 \\ 0.7184 & 1.0786 \end{bmatrix}$$

Plus, the mean square reconstruction error as computed by Pytorch built-in method was: 0.0136.

b. Also, using torch.svd(), we could obtain unitary matrices U, V and a diagonal matrix Σ. By simply cutting the last column of each of them their variations U_t, Σ_t, V_t can be obtained so that we rebuilt the $\tilde{A}$.

c. The MSE of truncated SVD in this case is 0.0135, quite close to the low-rank matrix factorisation. They are fair results where these two factorising methods redescribe a matrix with low errors equally. In other words, they both have found a pattern in data. More importantly, it can be characterised with data of lower verbosity and noise.

2. Matrix Completion

Low rank factorisation is also doing great job with imputing missing values in an incomplete matrix.
This matrix below is what I estimate from the incomplete one given (the original one on the right as a comparison) with MSE error of 0.4270.

$$\begin{bmatrix} 0.3425 & 0.5994 & 0.1677 \\ 2.1154 & 0.0494 & 1.8366 \\ 2.9396 & 1.3863 & 2.2634 \end{bmatrix} \ and \ \begin{bmatrix} 0.3347 & 0.6005 & 0.1735 \\ 3.3359 & 0.0492 & 1.8374 \\ 2.9407 & 0.5301 & 2.2620 \end{bmatrix}$$

Objectively, the estimate values are not much accurate but beneficial. They are from a latent variable model whose complexity is decided by the rank we make and there will be less noise.