COMP6248 Lab 1 Exercise

Adrian Azzareli, aab1g18@soton.ac.uk

Supplementary Note 1: Matrix Factorisation using Gradient Descent

The following code was implemented as per the algorithm provided in the exercise.

```
def sgdfactorise(A:torch.Tensor,rank:int,
  num_epochs=1000,lr=0.01) ->
  Tuple[torch.Tensor, torch.Tensor]:
  R,C = A.size()
  U = torch.rand((R, rank))
  V = torch.rand((C,rank))
  for i in range(num_epochs):
    for r in range(R):
      for c in range(C):
        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
```

A sample of $\hat{U}_0$ and $\hat{V}_0$ were computed as the following (these results are not fixed given the iterative scheme does not converge in the given epochs). The computed reconstruction loss, $\mathcal{L}_0$, is 0.1219. Over *eight* iterations the mean loss, $\mathcal{L}_\mu$, was determined as 0.1215.

$$\hat{U} = \begin{bmatrix} -0.4015, 0.5351 \\ 1.2626, 1.1777 \\ 0.6147, 1.6270 \end{bmatrix} \quad \textbf{(1)}$$

$$\hat{V} = \begin{bmatrix} 1.2858, 1.3812 \\ -0.5397, 0.5706 \\ 0.5531, 1.0850 \end{bmatrix} \quad \textbf{(2)}$$

Supplementary Note 2: Truncated Singular Value Decomposition

We use truncated SVD to reconstructed a matrix $\bar{A}$ by setting the last singular value to zero. The resulting error is 0.1219 (4 s.f.). This is similar to the loss from matrix factorisation using SGD. As the Eckart-Young-Mirsky theorem [1] suggests a solution for "approximating by one of lower rank". More directly, it suggests reductions in the number of non-zero singular values, hence the decision to bound the least significant singular value(s) to zero and reconstruct a matrix approximation.

[1] Golub, G.H., Hoffman, A. and Stewart, G.W., 1987. A generalization of the Eckart-Young-Mirsky matrix approximation theorem. Linear Algebra and its applications, 88, pp.317-327

Supplementary Note 3: Matrix Completion

The following code was implemented.

```
def sgdfactorise_mask(A:torch.Tensor,
 M:torch.Tensor,rank:int,num_epochs=1000,
 lr=0.01) -> Tuple[torch.Tensor, torch.Tensor]:
  R,C = A.size()
  U = torch.rand((R, rank))
  V = torch.rand((C,rank))
  for i in range(num_epochs):
    for r in range(R):
      for c in range(C):
        if M[r,c]:
          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
```

The estimated matrix $\hat{A} = \hat{U}\hat{V}^T$, where $\hat{U}_\mu = \begin{pmatrix} -0.4495, 0.7121 \\ 0.8957, 0.8445 \\ 1.0480, 1.1043 \end{pmatrix}$ and $\hat{V}_\mu = \begin{pmatrix} 1.3855, 1.3481 \\ -0.4640, 0.5504 \\ 1.1419, 0.9646 \end{pmatrix}$ (again, the iterative scheme tell us these values are not fixed - hence the average calculations). Therefore, the reconstruction matrix $\hat{A}_\mu = \begin{pmatrix} 0.3373, 0.6005, 0.1737 \\ 2.3795, 0.0492, 1.8374 \\ 2.9407, 0.1215, 2.2620 \end{pmatrix}$ (relative loss $\mathcal{L}_\mu = 1.101$).

The matrices A and $\hat{A}$ are similar (at known matrix indexes), though evidently there are large discrepancies in the indexes which were masked (unknown). This suggests *matrix completion* is not accurate.