Lab 2 Report — Automatic Differentiation

Jack Hilton-Jones (jhj1g23@soton.ac.uk)

I. Implement matrix factorisation using gradient descent

```
def gd_factorise_ad(A: torch.Tensor, rank: int,
    num_epochs=1000, lr=0.01) -> Tuple[torch.Tensor,
    torch.Tensor]:
  U = torch.rand(A.shape[0], rank, requires_grad=True)
  V = torch.rand(A.shape[1], rank, requires_grad=True)
  for epochs in range(num_epochs):
    U.grad = None
    V.grad = None
    reconstruction = U @ V.t()
    loss = torch.nn.functional.mse_loss(A, reconstruction
    , reduction = 'sum')
    loss.backward()
    U.data = U - U.grad * lr
    V.data = V - V.grad*lr
  return U,V
```

A. Factorise and compute reconstruction error on real data

The reconstruction loss of the gd_fractorise_ad is 15.2290, the reconstruction loss computed using a truncated Singular Value Decomposition of the same data is 15.2288. These two values are very similar suggesting that this may be the optimal factorisation for the data, if this is true then the gradient descent algorithm is performing well.

B. Compare against PCA

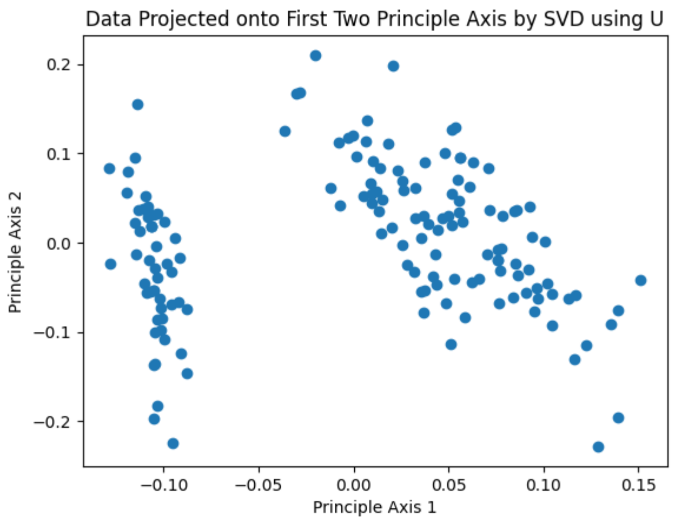

Fig. 1. Data Projected onto First Two Principle Axis by SVD using U

The output of this code is two scatter graphs. The U matrix aims to capture the maximum variance from the data and this outputs a bigger spread of the data. However $\hat{U}$ looks to minimise the reconstruction error which will simplify the data as it focuses on aligning the data along the direction of maximum variance. Using gradient descent looks to pronounce certain patterns whereas, SVD captures data variance.
PCA projects data onto a lower dimensional space spanned by the principal components and is the same as performing an orthogonal linear transformation. Minimising the reconstruction loss is related, as when this discards singular values it therefore finds a lower-dimensional representation of the data that loses as little information as possible. It will approximate the original data matrix with a lower rank matrix, relating to PCA. With a low reconstruction loss, it will closely resemble the original data points when reconstructed back into the original space. PCA minimises the reconstruction error indirectly by maximising the variance in a reduced dimension.

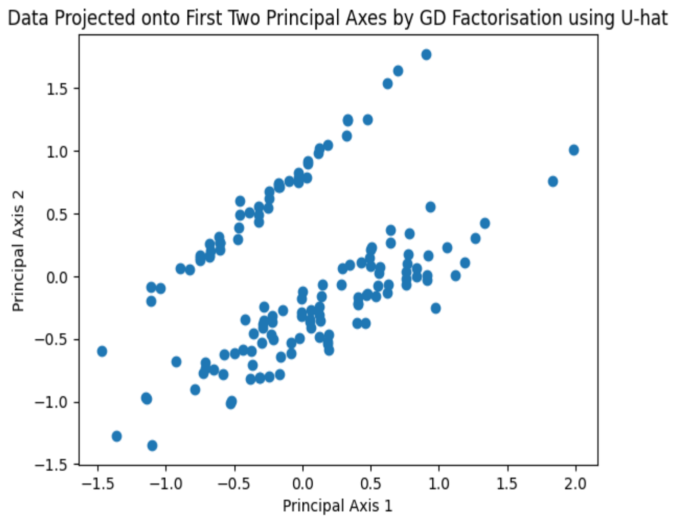

Fig. 2. Data Projected onto First Two Principle Axis by SVD using $\hat{U}$

II. A Simple MLP

```
def compute_accuracy(data, targets, W1, b1, W2, b2):
  z1 = torch.mm(data, W1) + b1
  a1 = torch.relu(z1)
  logits = torch.mm(a1, W2) + b2
  predictions = torch.argmax(logits, dim=1)
  accuracy = (predictions == targets).sum().item() /
    targets.size(0)
  return accuracy
```

The MLP is run on the training data five times and the loss, training accuracy and the validation accuracy is computed. The average loss is: 0.5787, the average training accuracy is: 78.8% and the average validation accuracy is: 76.0%. The MLP is showing a moderately good training accuracy, there was some variability in the validation accuracy. From this data, there is shown to be correlation between the loss values and the accuracy, as lower loss values tend to correspond to higher accuracies, however this is not strictly always true.