LAB 2 - AUTOMATIC DIFFERENTIATION

Supritha Konaje - 32864477
ssk1n21@soton.ac.uk

1. IMPLEMENT MATRIX FACTORISATION USING GRADIENT DESCENT (TAKE 2)

1.1. Implement gradient-based factorisation using PyTorch's AD

The PyTorch AD implementation for gradient based factorisation is given in the following snippet.

```
def gd_factorise_ad(A: torch.Tensor, rank: int,
    num_epochs = 1000, lr = 0.01) ->
    Tuple[torch.tensor, torch.tensor]:

  m, n = A.shape
  U = torch.rand(m , rank, requires_grad=True)
  V = torch.rand(n , rank, requires_grad=True)

  for epoch in range(num_epochs):
    norm_loss = torch.nn.functional.mse_loss(
    input = A, target = U @ V.T, reduction= 'sum')
    norm_loss.backward()

    U.data = U.data - U.grad.data * lr
    V.data = V.data - V.grad.data * lr

    U.grad.data.zero_()
    V.grad.data.zero_()

  return U, V
```

1.2. Factorise and compute reconstruction error on real data

The result for the reconstruction loss for the Gradient Descent Factorisation using AD and truncated SVD is given below:

```
GD Factorisation using AD Loss = 15.2290

Truncated SVD Loss = 15.2288
```

The result for both the losses are similar. SVD uses a low rank approximation in reducing the SVD output with rank 2.

2. COMPARE AGAINST PCA

The Principal Direction of the U from SVD is shown in Figure 1 and the direction for $\hat{U}$ from GD factorisation using AD is shown in Figure 2.

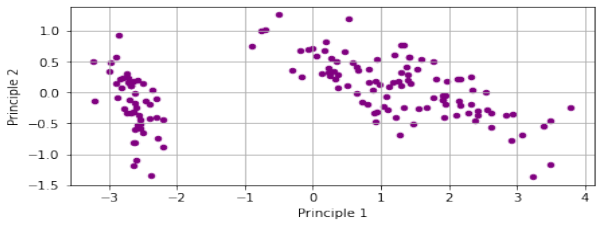

Fig. 1. SVD

When the two scatter plots are compared, it is seen that there has been some sort of rotation and the variance of the plot has also reduced in the Figure 2 which directly causes to increase the reconstruction loss. Hence, if the variance is less then the reconstruction loss is also comparatively more.

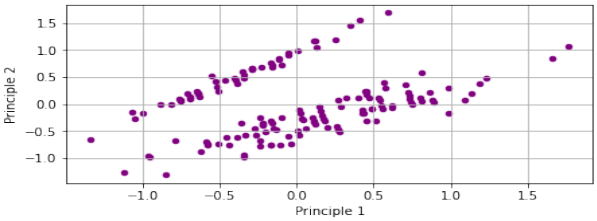

Fig. 2. Matrix Factorisation

3. A SIMPLE MLP

3.1. Implement the MLP

The implementation for the masked factorisation is given as:

```
def create_mlp(training_data,
    training_target_data, validation_data,
    validation_target_data, num_epochs = 100,
    lr_rate = 0.01):
  W1 = torch.rand(4,12, requires_grad=True)
  W2 = torch.rand(12,3, requires_grad=True)

  b1 = torch.tensor(0.0, requires_grad=True)
  b2 = torch.tensor(0.0, requires_grad=True)

  # Train the data
  for epoch in range(num_epochs):

    logits = torch.relu(training_data @
        W1 +b1) @ W2 + b2
    entropy = torch.nn.functional.cross_entropy(
    logits, training_target_data)
    entropy.backward()

    W1.data = W1.data + lr_rate * W1.grad.data
    W2.data = W2.data + lr_rate * W2.grad.data

    b1.data = b1.data + lr_rate * b1.grad.data
    b2.data = b2.data + lr_rate * b2.grad.data

    W1.grad.zero_()
    W2.grad.zero_()
    b1.grad.zero_()
    b2.grad.zero_()

  return W1, W2, b1, b2
```

3.2. Test the MLP

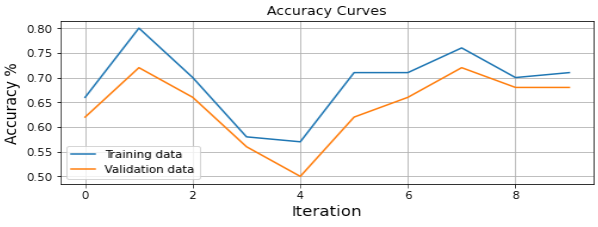

Fig. 3. MLP Accuracy Curve for Test and Validation Data

It can be observed from the graph that there are cases when the accuracy were not good enough. Random initialisation of the weights and biases in the start might be one of the reasons as in some cases initialising any variable randomly leads to improper results.