COMP6248 Lab 3 Exercise

Adrian Azzareli, aab1g18@soton.ac.uk

1 Exploring optimisation of analytic functions

By optimising the Rastrigin Function (global minimum at $[0,0]$) with SGD (with and without Momentum), Adagrad and Adam, the final position ater 100 epochs was captured in Table 1.

SGD	SGD with Momentum	Adagrad	Adam
[2.82, 2.82]	[-0.95, -0.95]	[4.48, 4.48]	[3.94, 3.94]

Table 1. Function output at epoch 100 for each optimiser

Figure 1 illustrates the loss sequence of each optimiser over the 100 epochs. Evidently, the SGD with Momentum optimiser converges quickest and results in a better solution after the final epoch. This is interesting considering Adagrad and Adam suggest improvement on SVD with momentum.

The expectation is that Adam proposes superior optimisation. It does however generalise poorly, as elaborated in Wilson et. al[1], which may be a reason for its poor performance. This also suggests that parameter tuning is almost necessary for Adam to perform well.

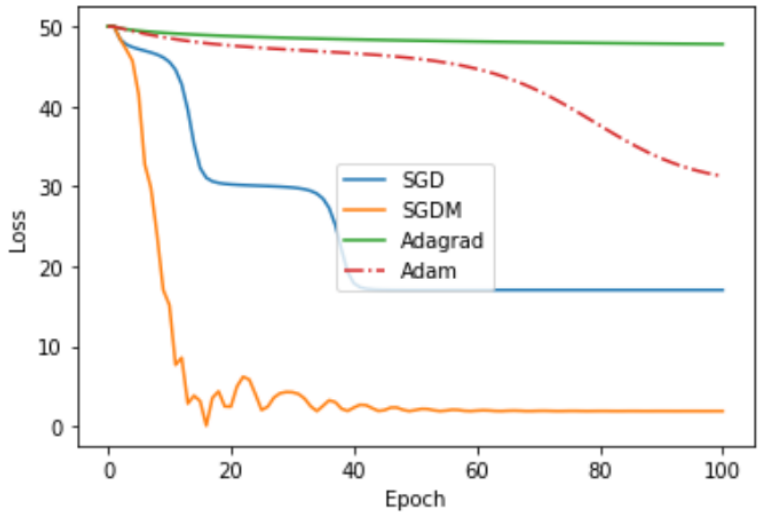

Fig. 1. History of Loss for each optimiser

2 Simple MLP

gd_train_mlp(.) was implemented to train the MLP using gradient descent. We also defined our objective function as *objective_function(.)*.

Accuracy was determined by finding the percentage of correctly estimated targets. The results in Table 2 indicate that a good training accuracy is *likely* to lead to a good validation accuracy (e.g. #3 and #6). It is also noticeable that a training accuracy of 0.69 repeatedly results in a validation of accuracy of 0.62. This could suggest these iterations were similarly initialised and perhaps indicates the existence of a local minimal which they get stuck in - hence the same validation accuracy. Further investigation is necessary to be certain.

[1] Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, Benjamin Recht. The Marginal Value of Adaptive Gradient Methods in Machine Learning. 2017. arXiv:1705.08292v2

```
def gd_train_mlp(data_tr:torch.Tensor,
                 target_tr:torch.Tensor,
                 num_epochs=100, lr=0.01):
  W1 = torch.rand((4, 12),requires_grad=True)
  W2 = torch.rand((12, 3),requires_grad=True)
  b1 = torch.zeros((1),requires_grad=True)
  b2 = torch.zeros((1),requires_grad=True)
  for e in range(num_epochs):
    for p in [W1,W2,b1,b2]:
      p.grad = None # zero param grads
    loss = objective_function(data_tr,
                    W1,W2,b1,b2,targets_tr)
    loss.backward()
    for p in [W1,W2,b1,b2]:
      p.data -= lr*p.grad # perform descent
  return W1, W2, b1, b2

def objective_function(data,W1,W2,b1,b2,
                                  targets):
    logits = torch.relu(data@W1+b1)@W2+b2
    return torch.nn.functional.cross_entropy
                                (logits, targets)
```

Table 2. Accuracy of MLP over 8 iterations

#	Training Accuracy	Validation Accuracy
0	0.69	0.62
1	0.71	0.64
2	0.69	0.62
3	0.79	0.80
4	0.65	0.56
5	0.65	0.64
6	0.69	0.62
7	0.62	0.54