A Reprot for Deep Learning Lab Exercise 3

Ziyi Guo, MSc Data Science

zg2u21@sonton.ac.uk

1 Optimisation of Rastrign function

In the first part of the experiment, a series of optimisers in Pytorch are implemented and used to find the optimisation of the 2D Rastrigin Function. The loss surface of the function is visualized and the paths as well as loss changes of the optimisers are plotted as below.

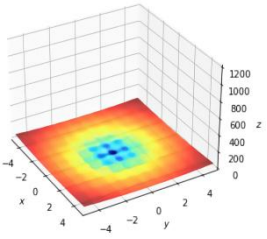

Fig 1. 2D Rastrigin Function Surface

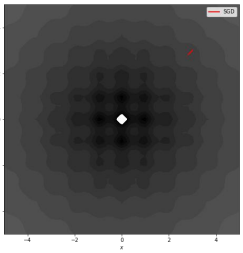

Fig 2. Path of SGD (lr=0.01)

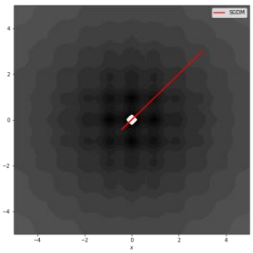

Fig 3. Path of SGDM (lr=0.01,m=0.9)

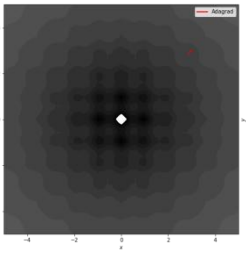

Fig 4. Path of Adagrad (lr=0.01)

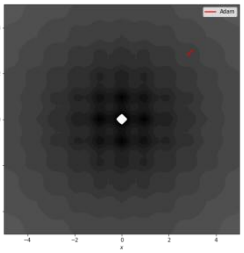

Fig 5. Path of Adam (lr=0.01)

It can be observed that with default values for unspecified parameters, only the ***SGDM*** has probably ever reached the global minima. Exploring the loss plot in Fig.6 and computing the points where the optimisers arrive, it is found that ***SGD***, ***Adagrad*** and ***Adam*** arrive at [2.8223,2.8223], [2.8685,2.8685], and [2.8216,2.8216] while ***SGDM*** arrives at [-0.001, -0.001]. It can be inferred that the ***SGDM*** approximates and nearly finds the global minima at [0,0], and thus working best in this circumstance.

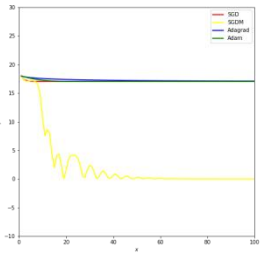

Fig 6. Optimisers' Loss Plot (lr=0.01)

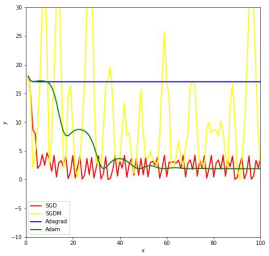

Fig 7. Optimisers' Loss Plot (lr=0.1)

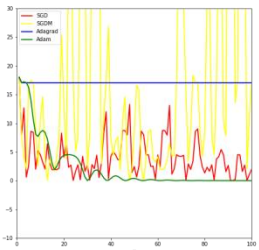

Fig 8. Optimisers' Loss Plot (lr=0.2)

In order to find out the reason, further experiments apply different learning rates to test the optimisers. It can be observed that with an increasing learning rate the ***Adam*** arrives at the global minima while the ***SGD*** as well as the ***SGDM*** are jumping around the global minima constantly, indicating that all the optimisers **except** the ***SGDM*** are stuck in local minima because of a low learning rate in the previous experiment while a high learning rate leads to excessive gradients for the ***SGD*** and the ***SGDM*** to find minima.

2 Optimisation of SVM on real data

In the second part of the experiment, a soft-margin SVM is trained and tested on the Iris dataset with different optimisers. The process of the algorithm (taking ***SGD*** programming for example) and the loss plot of both optimisers in one case are as below.

```
def svm(x, w, b):
    h = (w*x).sum(1) + b
    return h
def hinge_loss(y_pred, y_true):  # Define Loss Function
    N = y_true.size()[0]
    l = 1 - y_pred*y_true
    O = torch.zeros(N)
    loss = torch.max(l,O)
    e = torch.sum(loss)/N
    return e
w_SGD = torch.randn(1, 4, requires_grad=True)        # Parameter Initialization
b_SGD = torch.randn(1, requires_grad=True)
opt = optim.SGD([w_SGD,b_SGD], lr=0.01, weight_decay=0.0001)  # Define Optimiser
for epoch in range(100):                             # Gradient-based Optimisation
    for batch in dataloader:
        opt.zero_grad()
        H_SGD = svm(data_train, w_SGD, b_SGD)
        loss_SGD = hinge_loss(H_SGD, targets_train)
        loss_SGD.backward()
        opt.step()
output = svm(data_valid, w_SGD, b_SGD).data     # Computing Function Value
for i in range(output.size()[0]):               # Soft-margin Classification
    if output[i]>0:
        output[i] = 1
    else:
        output[i] =-1
Accuracy_SGD = accuracy_score(output,targets_valid)  # Computing Accuracy
```

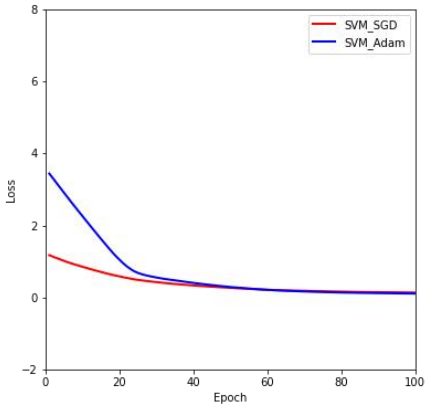

Fig 9. Loss Plot of SVM Optimisation with ***Adam*** and ***SGD***

In the abundant experiments, the validation accuracies of the SVM with ***SGD*** are in a range of [0.84, 0.96] while the SVM with ***Adam*** gets scores in [0.88, 0.92], indicating that the data are possibly not linear separable. Worth mentioning is that SVM with no optimiser but randomly initialized ***w*** and ***b*** (random classification) gets accuracy scores in a range of [0.12,0.40] in most cases and specifically in a comparatively best case where the ***SGD*** gets 0.96 while the ***Adam*** gets 0.92, it gets a score of 0.36, which are not beyond 50% as intuitive expectation. It could be due to the difference between data amounts in the classes.