Lab 3 Exercise - Optimise it!

Jonathon Hare (jsh2@ecs.soton.ac.uk)

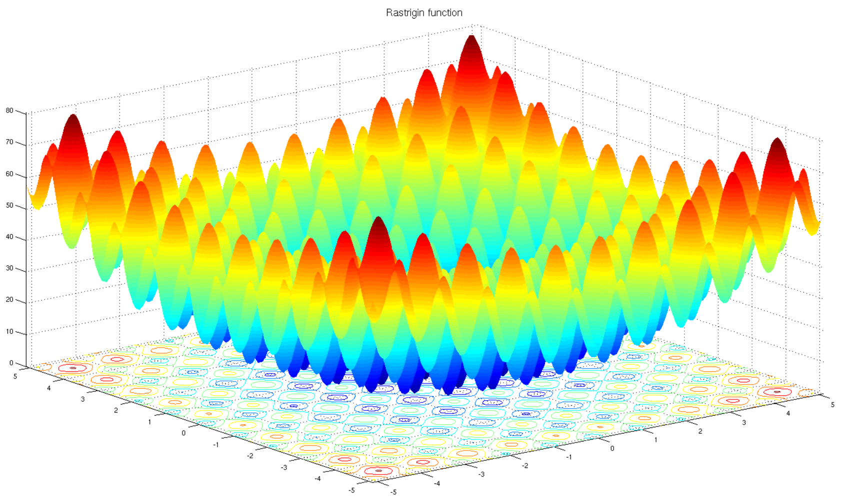
February 17, 2020

This is the exercise that you need to work through **on your own** after completing the third lab session. You'll need to write up your results/answers/findings and submit this to ECS handin as a PDF document along with the other lab exercises near the end of the module (1 pdf document per lab).

We expect that you should aim to use one side of A4 to cover your responses to *this* exercise. This exercise is worth 5% of your overall module grade.

1 Exploring optimisation of analytic functions

In the lab you looked at optimising Himmelblau's Function. Now we're going to explore something even more challenging. The Rastrigin Function is a fun optimisation challenge with many local minima and a single global minima:



1.1 Rastrigin (3 marks)

Consider the 2D Rastrigin function (`https://en.wikipedia.org/wiki/Rastrigin_function`). Starting at $[5, 5]$ compute the point where the following optimisers arrive at after 100 epochs:

- SGD (lr=0.01)
- SGD+Momentum (lr=0.01, momentum=0.9)
- Adagrad (lr=0.01)
- Adam (lr=0.001)

Create a loss plot showing the function value at each epoch for each of the different optimisers. Use the PyTorch implementations of the optimisers with the default values for unspecified parameters.

2 Optimisation of a SVM on real data

The second part of the lab saw you apply a soft-margin SVM to artificially generated data and optimise its parameters with gradient descent. Now we're going to do the same with real data. Just like last week we'll use the Iris dataset, and split it into training and validation subsets (and normalise), but this time, we'll only be using two of the classes:

```
import pandas as pd
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases'
        +'/iris/iris.data', header=None)
df = df.sample(frac=1) #shuffle

df = df[df[4].isin(['Iris-setosa', 'Iris-versicolor'])] #filter

# add label indices column
mapping = {k: v for v, k in enumerate(df[4].unique())}
df[5] = df[4].map(mapping)

# normalise data
alldata = torch.tensor(df.iloc[:, [0,1,2,3]].values, dtype=torch.float)
alldata = (alldata - alldata.mean(dim=0)) / alldata.var(dim=0)

# create datasets
targets_tr = torch.tensor(df.iloc[:100, 5].values, dtype=torch.long)
targets_va = torch.tensor(df.iloc[100:, 5].values, dtype=torch.long)
data_tr = alldata[:100]
data_va = alldata[100:]
```

2.1 Iris SVM (2 marks)

Use the following configurations to train Soft-margin Linear SVMs with three different amounts of weight decay $(0.1, 0.01, 0.001)$ on the training data, and compute accuracy on the validation set.

- SGD (lr=0.01)
- SGD+Momentum (lr=0.01, momentum=0.9)
- Adam (lr=0.001)

What is the accuracy for the nine different models? What is the best value for the weight decay parameter? What is the best optimiser configuration for this task & why?