Week 6 Assignment

Neimhin Robinson Gunning, 16321701

28th March 2024

(a) (i) Implementing mini-batch Stochastic Gradient Descent
Our global loss function is

$$f_T(x) = \sum_{w \in T} \frac{loss(x, w)}{\#W}$$

which is just the average of $loss(w, x)$ ranging over the entire dataset, T. We can also calculate an approximation of the loss using a subset (minibatch) of T.

$$f_N(x) = \sum_{w \in N} \frac{loss(x, w)}{\#N}$$

This is implemented on line 17 of Algorithm 1. We can also approximate the gradient w.r.t. to the minibatch rather than the full training dataset. In these experiments we use the finite difference methods to estimate the mini-batch gradient according to

$$\frac{df_N}{dx_i} \approx \frac{f_N([x_1, \ldots, x_i + \epsilon, \ldots, x_d]) - f_N(x)}{\epsilon}$$

where we set $\epsilon = 10^{-15}$ for the remainder of this discussion. We also look at only at an example with $d = 2$ so the finite difference gradient is:

$$\nabla f_N = [\frac{f_N([x_1 + \epsilon, x_2]) - f_N(x)}{\epsilon}, \frac{f_N([x_1, x_2 + \epsilon]) - f_N(x)}{\epsilon}]$$

The code implementation of this is on line 4 in Algorithm 1.

To generate mini-batches we first shuffle the rows data set and then take successive slices with n rows, where n is the mini-batch size. The first mini-batch consists of the 1st to the nth data items, the second consists of the $(n+1)th$ to the $(n+n)th$, etc. If we reach the end of the dataset before filling the minibatch we shuffle the dataset and start again from index 1. This is implemented on line 10 of Algorithm 1.

The implementation of mini-batch SGD here relies on generating successive f_{N_t} and ∇f_{N_t}, where N_t is the mini-batch for iteration t. This is implemented on line 40 of Algorithm 1.

At each iteration the step size can be calculated with respect to f_{N_t} and ∇f_{N_t} using of the Polyak, RMSProp, Heavy Ball, and Adam methods. Each of the step types are implemented in `src/sgd.py` which is included in the appendix.

Algorithm 1 Generating mini-batches, N, and associated f_N and ∇f_N.

src/ai.py   Sun Mar 24 17:42:18 2024   1

```python
import numpy as np
import sympy as sp

def gradient_function_fd(minibatch, epsilon=10**(-15)):
    def gradient_fd(x):
        dydx1 = (f(x + np.array([epsilon, 0]), minibatch) - f(x, minibatch)) / epsilon
        dydx2 = (f(x + np.array([0, epsilon]), minibatch) - f(x, minibatch)) / epsilon
        return np.array([dydx1, dydx2])
    return gradient_fd

def loss(x, w):
    z = x - w - 1
    left = 10 * (z[0]**2+z[1]**2)
    right = (z[0]+2)**2+(z[1]+4)**2
    return min(left, right)

def f_clear(x, minibatch):
    return sum(loss(x, w) for w in minibatch) / len(minibatch)

def generate_minibatches(T, n=5, seed=42, shuffle=True):
    if shuffle:
        T = T.copy()
        np.random.seed(seed)
        np.random.shuffle(T)
    num_rows = T.shape[0]
    i = 0

    minibatch = np.zeros((n, T.shape[1]), T.dtype)
    while True:
        for j in range(n):
            minibatch[j] = T[i % num_rows]
            i += 1
            if shuffle and i >= num_rows:
                # begin next epoch
                np.random.shuffle(T)
                i = 0
        current_minibatch = minibatch
        yield minibatch

def generate_optimisation_functions(batch, minibatch_size=5, finite_difference=True, **kwargs):
    minibatch_generator = generate_minibatches(
        batch, n=minibatch_size, **kwargs)
    for minibatch in minibatch_generator:
        def optim_func(x):
            return f_clear(x, minibatch)
        gradf = gradient_function_fd(minibatch)
        yield (optim_func, gradf)
    yield "finished"
```