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, $N \subseteq T$.

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

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.

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.

(a) (ii)

We downloaded a pair of functions which are reproduced in Figure 1. On line 6 is a python definition of $f_N(x)$, where N corresponds to `minibatch`. We generate T using lines 3 and 4 of Figure 1, and use the same T throughout the remainder of the discussion. A wireframe and a contour plot of $f_T(X)$ is presented in Figure 2, showing $x \in [-5, 5]^2$.

(a) (iii) Gradient Calculation

Implementations of both analytic calculation of gradients with sympy and finite difference are provided in `src/week6.py`. Our implementation of analytic calculation of gradients with sympy is unfortunately dirt-slow, so in these experiments we use the finite difference methods to estimate the mini-batch gradient according to

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

funcs.txt Thu Mar 21 11:56:17 2024 1

```
 1: import numpy as np
 2:
 3: def generate_trainingdata(m=25):
 4:     return np.array([0,0])+0.25*np.random.randn(m,2)
 5:
 6: def f(x, minibatch):
 7:     # loss function sum_{w in training data} f(x,w)
 8:     y=0; count=0
 9:     for w in minibatch:
10:         z=x-w-1
11:         y=y+min(10*(z[0]**2+z[1]**2), (z[0]+2)**2+(z[1]+4)**2)
12:         count=count+1
13:     return y/count
```

Figure 1: Functions downloaded for this assignment from `https://www.scss.tcd.ie/Doug.Leith/CS7DS2/week6.php`.

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

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

```
 1: import numpy as np
 2: import sympy as sp
 3:
 4: def gradient_function_fd(minibatch, epsilon=10**(-15)):
 5:     def gradient_fd(x):
 6:         dydx1 = (f(x + np.array([epsilon, 0]), minibatch) - f(x, minibatch)) / epsilon
 7:         dydx2 = (f(x + np.array([0, epsilon]), minibatch) - f(x, minibatch)) / epsilon
 8:         return np.array([dydx1, dydx2])
 9:     return gradient_fd
10:
11: def loss(x, w):
12:     z = x - w - 1
13:     left = 10 * (z[0]**2+z[1]**2)
14:     right = (z[0]+2)**2+(z[1]+4)**2
15:     return min(left, right)
16:
17: def f_clear(x, minibatch):
18:     return sum(loss(x, w) for w in minibatch) / len(minibatch)
19:
20: def generate_minibatches(T, n=5, seed=42, shuffle=True):
21:     if shuffle:
22:         T = T.copy()
23:         np.random.seed(seed)
24:         np.random.shuffle(T)
25:     num_rows = T.shape[0]
26:     i = 0
27:
28:     minibatch = np.zeros((n, T.shape[1]), T.dtype)
29:     while True:
30:         for j in range(n):
31:             minibatch[j] = T[i % num_rows]
32:             i += 1
33:             if shuffle and i >= num_rows:
34:                 # begin next epoch
35:                 np.random.shuffle(T)
36:                 i = 0
37:         current_minibatch = minibatch
38:         yield minibatch
39:
40: def generate_optimisation_functions(batch, minibatch_size=5, finite_difference=True, **kwargs):
41:     minibatch_generator = generate_minibatches(
42:         batch, n=minibatch_size, **kwargs)
43:     for minibatch in minibatch_generator:
44:         def optim_func(x):
45:             return f_clear(x, minibatch)
46:         gradf = gradient_function_fd(minibatch)
47:         yield (optim_func, gradf)
48:     yield "finished"
```

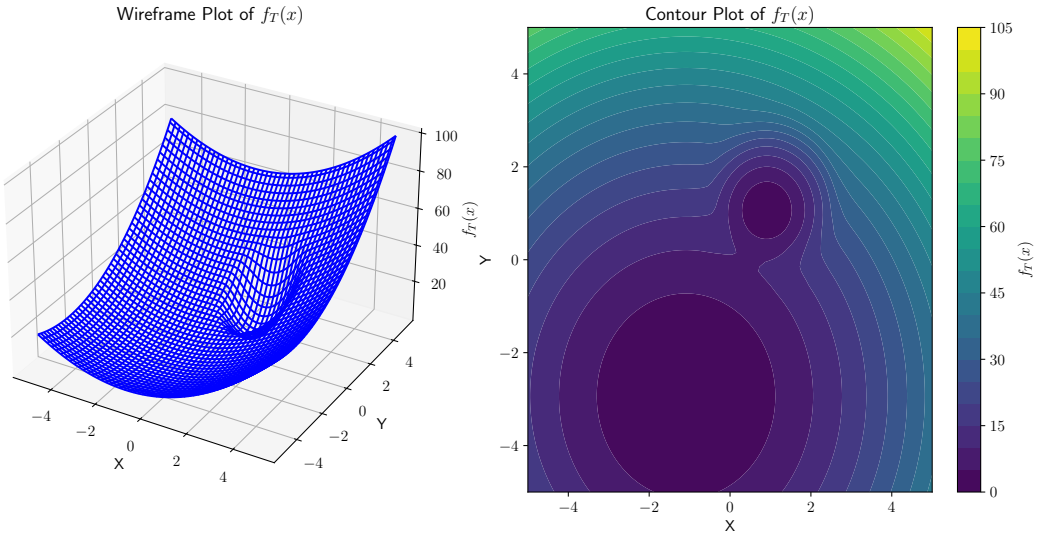

Figure 2: Wireframe (left) and contour (right) plots of $f_T(x)$ where T is a 'training set' generated by the code in lines 3 and 4 of Figure 1.

where we set $\epsilon = 10^{-15}$ for the remainder of this discussion. We also look at only at an example with $d = 2$, i.e. $x \in \mathbb{R}^2$ so the finite difference gradient function $\nabla f_N : \mathbb{R}^2 \to \mathbb{R}$ is:

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

This works by calculating the slope after a small perterbation to x_1 and then again a small perterbation to x_2. The code implementation of this is on line 4 in Algorithm 1.

(b) (i) Gradient Descent with constant step-size

Several runs of gradient descent with various constant step-sizes, α, are presented in Figure 3. The starting estimate is $x = [3, 3]$. The function $f_T(x)$ has a local minimum at about $x = [1, 1]$, but the global minimum is somewhere around $x = [-1, -3]$. A careless choice of α results in converging to the suboptimal local minimum, e.g. $\alpha = 0.01$. Either of $\alpha = 0.72$ or $\alpha = 0.5$ are reasonabl choices. For $\alpha > 0.72$ we see divergence. The lowest value of $F_T(x)$ for the $\alpha = 0.72$ run is marginally better than for the $\alpha = 0.5$ run, however, the $\alpha = 0.5$ converges marginally faster. For later experiments, where we are using Stochastic Gradient Descent, the noise caused by mini-batches will mean that we could diverge with lower values of α, so from here on we select $\alpha = 0.5$ rather than $\alpha = 0.72$ to mitigate the risk of divergence.

(b) (ii) Stochastic Gradient Descent with constant batch-size and step-size

4 runs of Stochastic Gradient Descent with constant batch-size, $n = 5$, and constant step size $\alpha = 0.5$ are presented in Figure 4. Each run is different due to the randomness introduced by sampling the batches. The first step for three of the runs is broadly in the direction of the global minimum, but the first run, run 0, has a first step which is about 70° off. Nonetheless, this run's overall trajectory is very similar to the those of the others. Even when a run achieves an x_t that minimizes $f_T(.)$ the run continues to 'explore' a region due to the noise introduced by mini-batches, i.e. the gradients are like a random variables drawn from $U(0, k)$. However, the further the current estimate is from the a local minimum, the more coherent are the gradients ∇f_{N_t}, $\nabla f_{N_{t+1}}$, etc., where N_t, N_{t+1} etc. are different mini-batches. This greater coherence at a distance is what allows the algorithm converge to similar values on most runs. It is possible for the algorithm to diverge with the same hyper parameters, $\alpha = 0.5$ and mini-batch size of 5, but that did occur for any of the 4 runs presented here. The gradient descent runs reported above in **(b) (ii)** have no randomness, so using $\alpha = 0.5$ gives the same results every time.

(b) (iii) Effect of Varying batch-size

We look at a broad range of batch sizes in Figure 5, from 1 up to 25 = $\#T$. In each case the SGD eventually hits a floor and then jumps around near that floor. The distance of jumps it takes from that floor is directly related to the

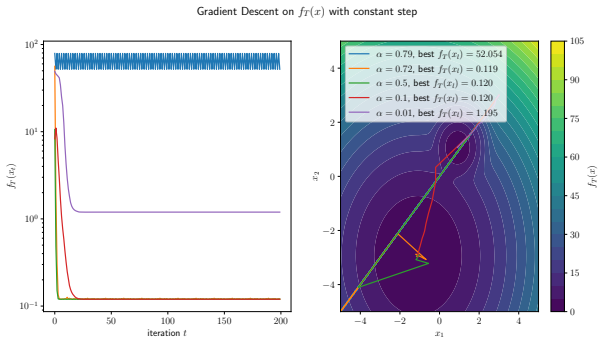

Figure 3: Visualizations of gradient descent on $f_T(x)$ using a constant step size α. On the left the function value is plotted against the iteration number. On the write the successive x_ts are plotted on a contour plot of $f_T(x)$.

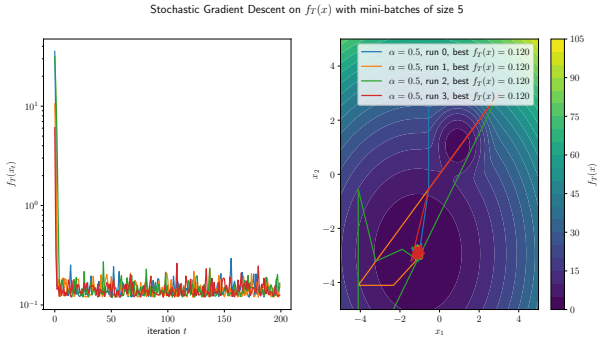

Figure 4: Visualizations of gradient descent on $f_{N_t}(x)$ using a constant step size α. N_t is drawn from T by first shuffling T and slicing T into batches of size 5. On the left the function value is plotted against the iteration number. On the write the successive x_ts are plotted on a contour plot of $f_T(x)$.

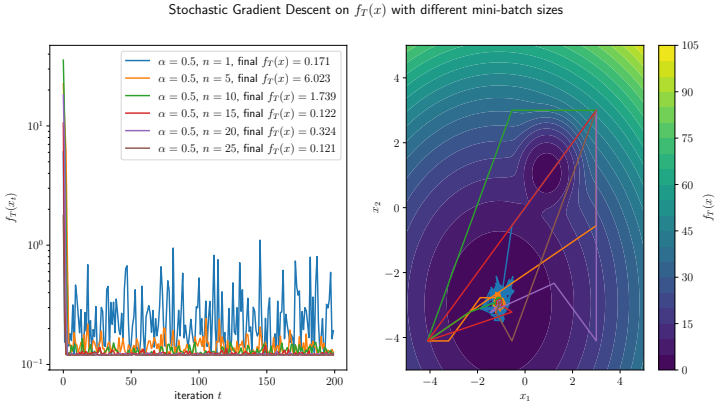

Figure 5: Visualizations of stochastic gradient descent on $f_T(x)$ with various batch-sizes.

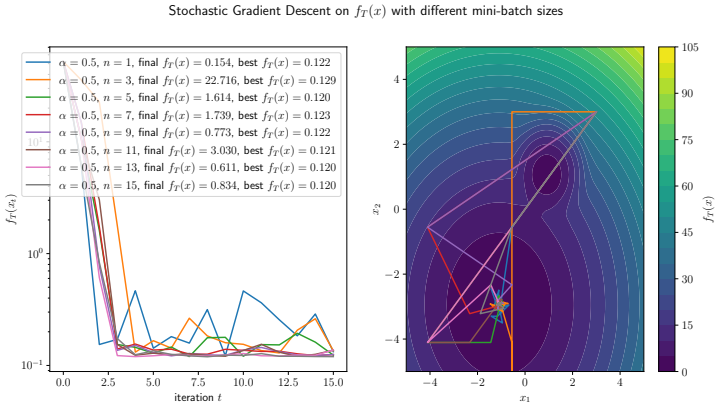

Figure 6: Visualizations of stochastic gradient descent on $f_T(x)$ with various batch-sizes and fixed α.

batch-size. With lower batch-size it takes larger jumps, i.e. there is greater noise for lower batch. When $n = 25 = \#T$, there is no noise, we calculate gradient with respect to the full set T each time. With a smaller value of n we see more noise in the gradient, meaning that the final value of x will be approx $\arg\min_x f_T(x) +$ noise, and the breadth of the noise will be inversely proportional to n. This is a regularising effect, because it means the final estimate of x is less tightly fit to the data T.

In Figure 6 we again vary n but in a narrower range (1-15) and for fewer iterations (15). There is no particular relationship between n and the speed with which $f_T(x)$ initially descends. In this experiment $f_T(x)$ is calculated at every iteration, which may be prohibitively expensive for optimisation problems with much larger datasets, so it is not necessarily practical to keep track of the best $f_T(x)$, but the data in Figure 6 do seem to indicate that a higher n increases the chances of a lower best $f_T(x)$, which could be construed as overfitting. The reason for this is that around the noise floor, using higher n means not jumping as far from the optimal region, and taking smaller jumps because the noise is less significant. This increases the chances of landing at a globally optimal value for x, because the interval effectively being searched is smaller.

(b) (iv) SGD with various step-sizes

In Figure 7 we present various runs of SGD, each with the same batch-size $n = 5$, but different step sizes α. The runs are set up with the same random seed such that the sequence of batches, N_t, N_{t+1}, is equivalent for each run, in order to isolate the effect of changing α. With $\alpha = 0.001$ the alg has similar behaviour to $\alpha = 0.1$ except taking many more iterations to follow the same trajectory. At $\alpha = 0.01$ the alg converges to the suboptimal local minimum. With $\alpha = 1$

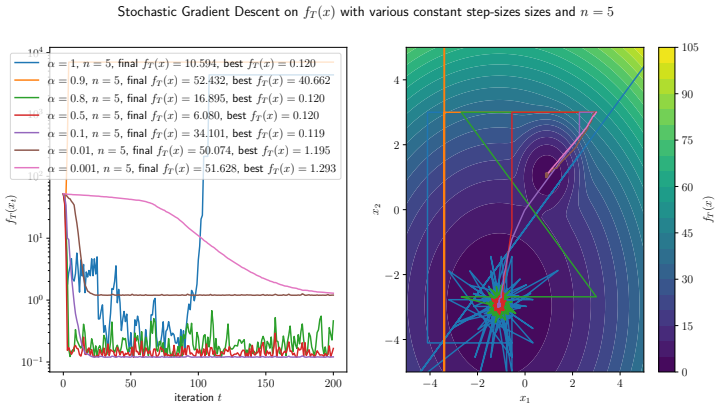

Figure 7: Visualizations of stochastic gradient descent on $f_T(x)$ with fixed batch-size $n = 5$, and varioust constant step-sizes, α.

the alg gets trapped for a time at a distance from the global minimum, but eventually get very close the global minimum, and then hops out again to fairly inaccurate estimates, diverging. The effect of the noise from stochastic batch sampling is seemingly amplified by a larger step size, such that for larger α we see a larger final search radius after approaching the noise floor, and a greater chance of diverging due to the noise, i.e. the chances of having a large unlucky step in the wrong direction. Where the batch-size n has a limited range of values, $[0, \#T]$, we can vary α arbitrarily in $\mathbb{R}^+$, and so if we want to tune just one hyperparameter rather than two, e.g. due to compute cost, then it may beneficial fix n and vary α.

By adjusting α we can achieve a noise level similar to the effect of decreasing n as we saw in section **(b) (iii)** above. This noise can be beneficial as seen in the run with $\alpha = 0.01$ in Figure 7. For this run the alg initially gets stuck in the suboptimal local minimum but eventually hops out and then converges to the global minimum.

(c) (i) SGD with Polyak step

Runs of SGD with Polyak step, $f^* = 0.119$, and $n = 1, 3, 5, 7$ are presented in Figure 8. If there is a relationship between n and the behaviour of the alg, the relationship is not strong. The experiment is run twice, with different random seeds, and there is no consistency between the two experiments. With seed=58 we see the alg converge to the suboptimal local minimum when $n = 1$, but with seed=59 the behaviour is different, it first explores the global minimum and then 'hops out' to the suboptimal local minimum. When we increase n to 7 we see the alg first explore the suboptimal local minimum and the 'hop out' to the global minimum. From these experiments there is no apparent stable relationship between n and the alg's behaviour.

When we don't use SGD, i.e. when $n = 25$, we see significantly smaller steps once the alg has approached the global minimum, so choosing $n << \#T$ does significantly change the behaivour compared to $n = 25$.

(c) (ii) SGD with RMSProp step

First we tune α and β on $f_T(x)$ using $n = \#T = 25$, see the top two plots in Figure 9. We find we always diverge when $\alpha = 0.01$ and sometimes when $\alpha = 0.5$. Of the options tested the alg converges most quickly with $\alpha = 0.1$ and $\beta = 0.99$, so these are the parameters used in the following discussion.

Keeping α and β fixed, see bottom two plots in Figure 9, we find that the relationship between n and the alg's behaviour is not strong, it can diverge when n is low ($n = 3, 5, 7$) and when n is high ($n = 25$). The runs with $n = 1$ and $n = 15$ behave very similarly, although the run with $n = 1$ is more noisy, i.e. has a slightly wider radius of exploration around the global minimum, and the run $n = 15$ has longer sequences of iterations where it takes very small steps.

(c) (iii) SGD with Heavy Ball step

After trying a wide array of hyperparameters $(\alpha, \beta) \in [0.005, 0.05, 0.075, 0.1, 1, 2] \times [0.1, 0.3, 0.5, 0.9, 0.99, 0.999]$, we find $(\alpha, \beta) = (0.1, 0.3)$ to give good results for Heavy Ball on $f_T(x)$ with $n = 25$. Four tuning runs are shown in the top two plots of Figure 10. To examine the effect of changing n we now fix $(\alpha, \beta) = (0.1, 0.3)$.

In the bottom two plots of Figure 10 we find that the Heavy Ball alg is liable to get stuck in the suboptimal local minimum. The batch-size n has a significant impact on the noise of the gradient once the alg, which is especially visible once the alg has approached a local minimum. Lower n results in more noise, in terms of both magnitude and direction

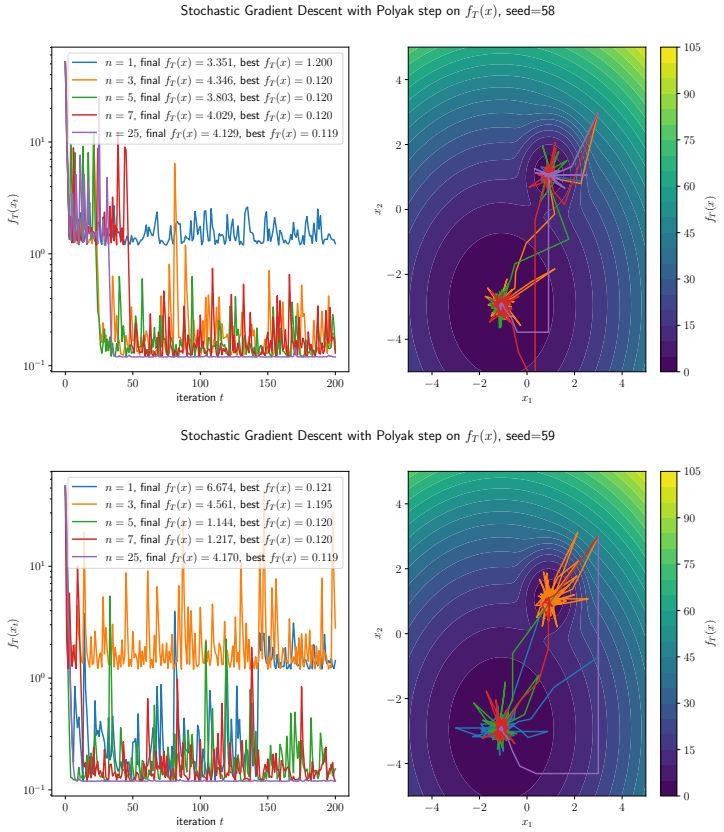

Figure 8: Visualizations of stochastic gradient descent on $f_T(x)$ with Polyak step and various n.

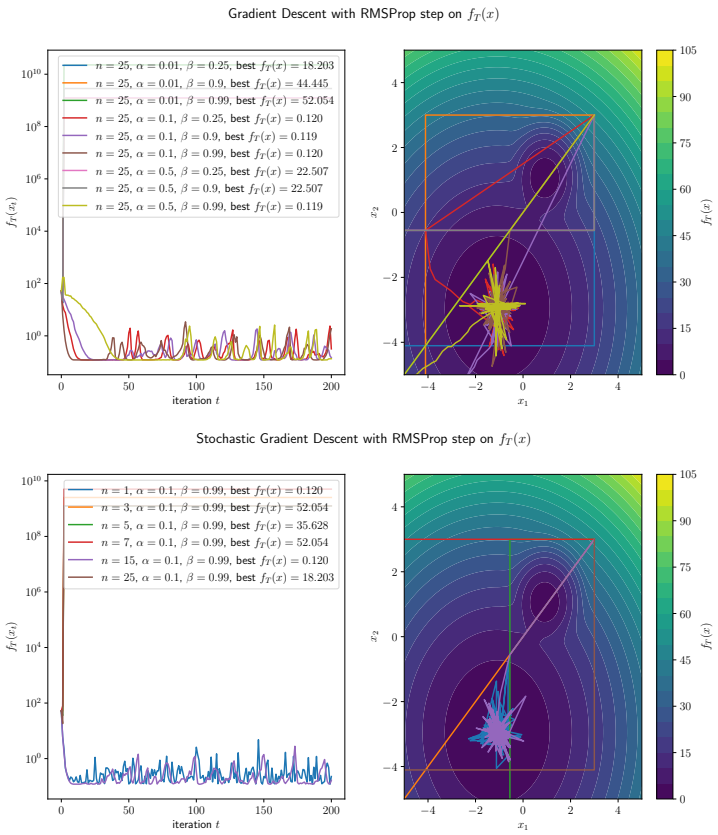

Figure 9: Visualizations of stochastic gradient descent on $f_T(x)$ with Polyak step and various n.

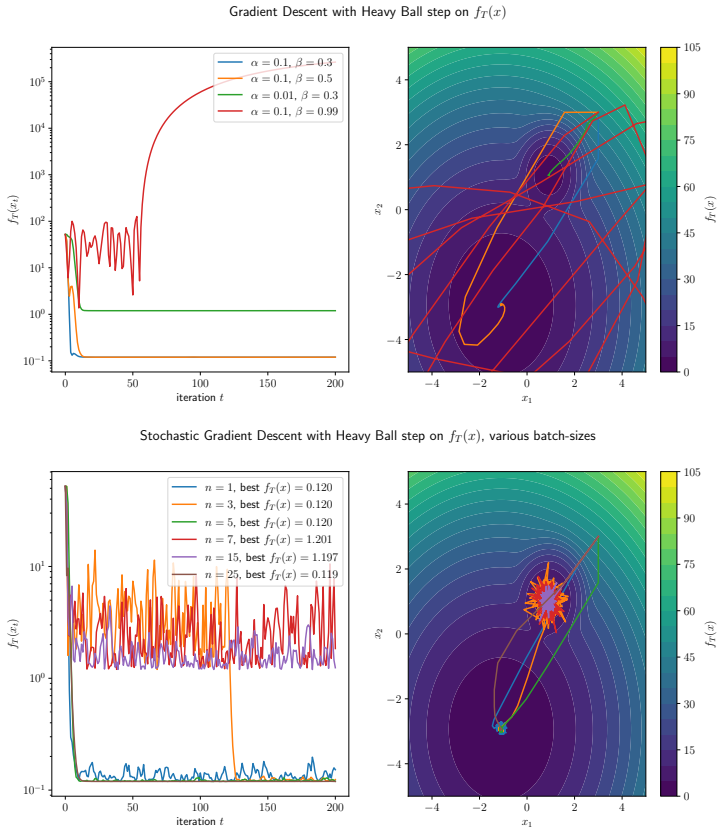

Figure 10: Visualizations of stochastic gradient descent on $f_T(x)$ with Heavy Ball step and various. Top two plots: tuning α and β. Bottom two plots: assessing impact of n.

of the gradient. The magnitude of the noise ends up being helpful for the run with $n = 3$, where it escapes the local minimum.

(c) (iii) SGD with Adam step

We use two steps to tune α, β_1, and β_2. First we fix $\alpha = 1$ and search for decent parameters of β_1, and β_2. Results of this search are presented in the top two plots of Figure 11. We settle on $\beta_1 = 0.9$ and $\beta_2 = 0.9$, because this yields the fastest approach to the global minimum, of the parameters tested. The second step is to tune α based on the fixed β_1 and β_2. Results of a search for α are presented in the bottom two plots of Figure 11. For $1 \leq \alpha \leq 5$, the results are similar, with perhaps sligthly faster approach for $\alpha = 3$. Here on we fix $(\alpha, \beta_1, \beta_2) = (3, 0.9, 0.9)$.

Various runs of Adam with these parameters, but different n are presented in Figure 12. After 100 iterations all runs have approached the global minimum and are jumping around randomly. A lower n corresponds to bigger jumps around, and thus more noise in the final values of x and $f_T(x)$.

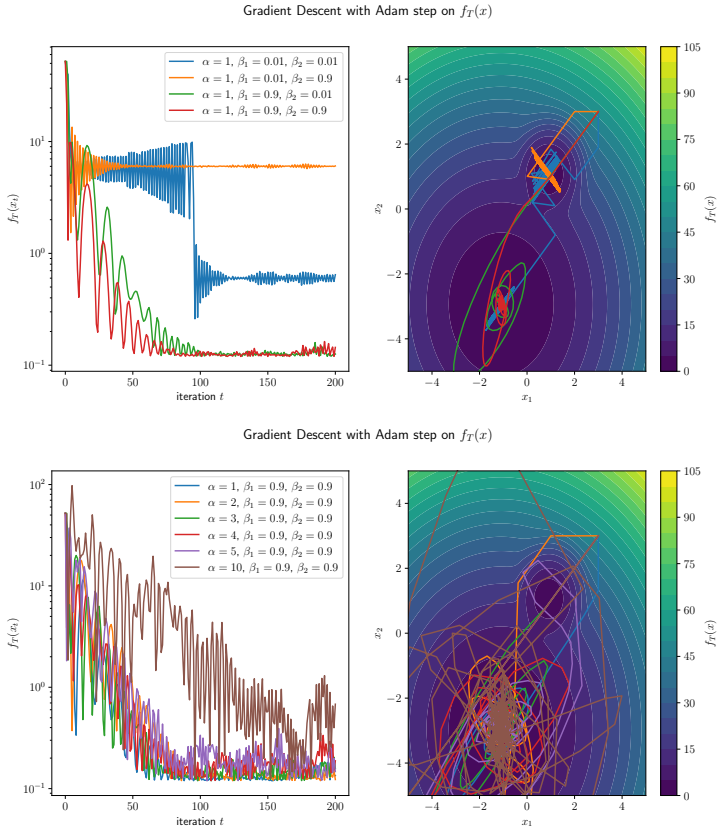

Figure 11: Visualizations of stochastic gradient descent on $f_T(x)$ with Adam step and various. Top two plots: tuning β_1, and β_2. Bottom two plots: tuning α given β_1 and β_2.

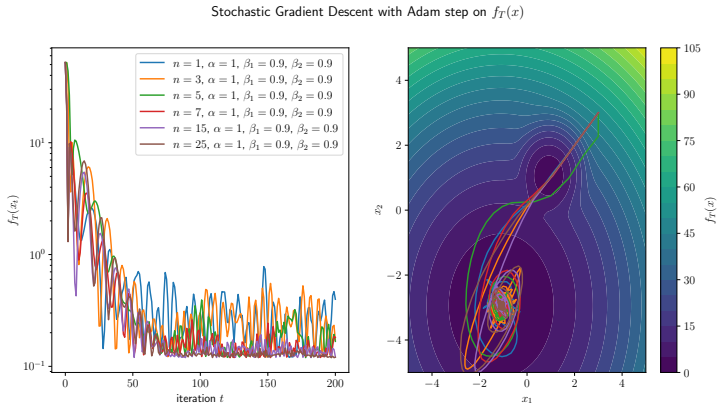

Figure 12: Visualizations of stochastic gradient descent on $f_T(x)$ with Adam step and various n.