Deep Learning : Assignment 1

Problem 2: Training 2-Layer Linear Neural Networks with Stochastic Gradient Descent

In the problem stated of training an age regressor using a 2-layer NN with SGD, we trained the model using a combination of three profound hyperparameters namely learning_rate, batch_size and number_of_epochs and tuned these hyperparameters so as to minimize the loss on the validation dataset.

Hyperparameters tested:

```
learning_rates = [1e-5,1e-4,1e-3]
mini_batch_sizes = [32, 64,128]
num_epochs_testing = [50, 100,150]
```

The model training was smooth for learning rates mentioned above. We also tried higher learning rates of **1e-1 and 1e0**,however, the MSE values were overshooting resulting in multiple “nan” and “inf” values in the dataset which affected the training output. Hence, we decided to opt for lower learning rates as shown above.

We found the optimal hyperparameters for the given dataset,thereby, giving the min MSE, to be:

```
{'num_epochs': 150, 'learning_rate': 0.0001, 'mini_batch': 64}
```

Training_MSE = 0.7645158065620998

On testing the model using the optimized weights and bias, tuned on the best hyperparameter combination, we obtained **Testing_MSE = 0.7690748379706631**

```
: y_test_pred = np.dot(X_te, test_weights) + test_bias


  test_mse = np.mean((y_test_pred - y_te) ** 2)
  print(f"Test MSE: {test_mse}")

  Test MSE: 0.7690748379706631
```

Output shown below indicates the MSE calculated at each possible combination of hyperparameters.

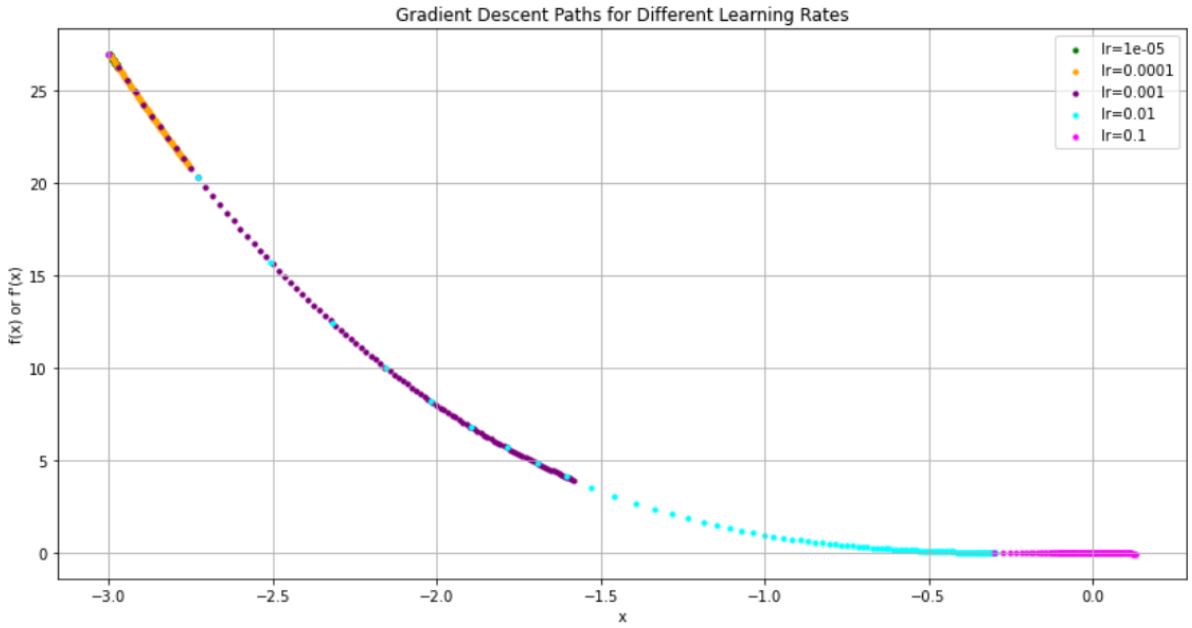
```
In [9]: best_hyp,best_weights,best_bias,best_mse= validation(X_train,y_train,X_val,y_val)

Num_Epoch 50, Batch_size 32, Learning_rate 1e-05, MSE: 0.7980858284148201
Num_Epoch 100, Batch_size 32, Learning_rate 1e-05, MSE: 0.7817600759463128
Num_Epoch 150, Batch_size 32, Learning_rate 1e-05, MSE: 0.7764923913809932
Num_Epoch 50, Batch_size 64, Learning_rate 1e-05, MSE: 0.825059311597272
Num_Epoch 100, Batch_size 64, Learning_rate 1e-05, MSE: 0.7982439602620542
Num_Epoch 150, Batch_size 64, Learning_rate 1e-05, MSE: 0.7871683315511885
Num_Epoch 50, Batch_size 128, Learning_rate 1e-05, MSE: 0.858354726422686
Num_Epoch 100, Batch_size 128, Learning_rate 1e-05, MSE: 0.8248488083114183
Num_Epoch 150, Batch_size 128, Learning_rate 1e-05, MSE: 0.8079652447471504
Num_Epoch 50, Batch_size 32, Learning_rate 0.0001, MSE: 0.768266541636217
Num_Epoch 100, Batch_size 32, Learning_rate 0.0001, MSE: 0.7662357760753514
Num_Epoch 150, Batch_size 32, Learning_rate 0.0001, MSE: 0.7726601133474943
Num_Epoch 50, Batch_size 64, Learning_rate 0.0001, MSE: 0.7692581662422355
Num_Epoch 100, Batch_size 64, Learning_rate 0.0001, MSE: 0.7649789122920176
Num_Epoch 150, Batch_size 64, Learning_rate 0.0001, MSE: 0.7645158065620998
Num_Epoch 50, Batch_size 128, Learning_rate 0.0001, MSE: 0.7777352539608878
Num_Epoch 100, Batch_size 128, Learning_rate 0.0001, MSE: 0.7715885337195111
Num_Epoch 150, Batch_size 128, Learning_rate 0.0001, MSE: 0.7677323043511067
Num_Epoch 50, Batch_size 32, Learning_rate 0.001, MSE: 0.8111897687858998
```

Training cost values for the last 10 iterations of gradient descent:

```
Epoch 150, Batch 45, MSE: 0.6511987188825773
Epoch 150, Batch 46, MSE: 0.4899284830150784
Epoch 150, Batch 47, MSE: 0.708024839257939
Epoch 150, Batch 48, MSE: 0.731853745717124
Epoch 150, Batch 49, MSE: 0.7272323539909239
Epoch 150, Batch 50, MSE: 0.6402859924099382
Epoch 150, Batch 51, MSE: 0.8287744928610086
Epoch 150, Batch 52, MSE: 0.8019119919145636
Epoch 150, Batch 53, MSE: 0.7266157982315757
Epoch 150, Batch 54, MSE: 0.6209848137819514
Epoch 150, Batch 55, MSE: 0.7748640852595792
Epoch 150, Batch 56, MSE: 0.7107980899023356
Epoch 150, Batch 57, MSE: 0.7245459889655109
Epoch 150, Batch 58, MSE: 0.8235122189069581
Epoch 150, Batch 59, MSE: 0.9188852404405099
Epoch 150, Batch 60, MSE: 0.7963911855056597
Epoch 150, Batch 61, MSE: 0.7220965522215743
Epoch 150, Batch 62, MSE: 0.896496156966533
Epoch 150, Batch 63, MSE: 0.831652339847603
```

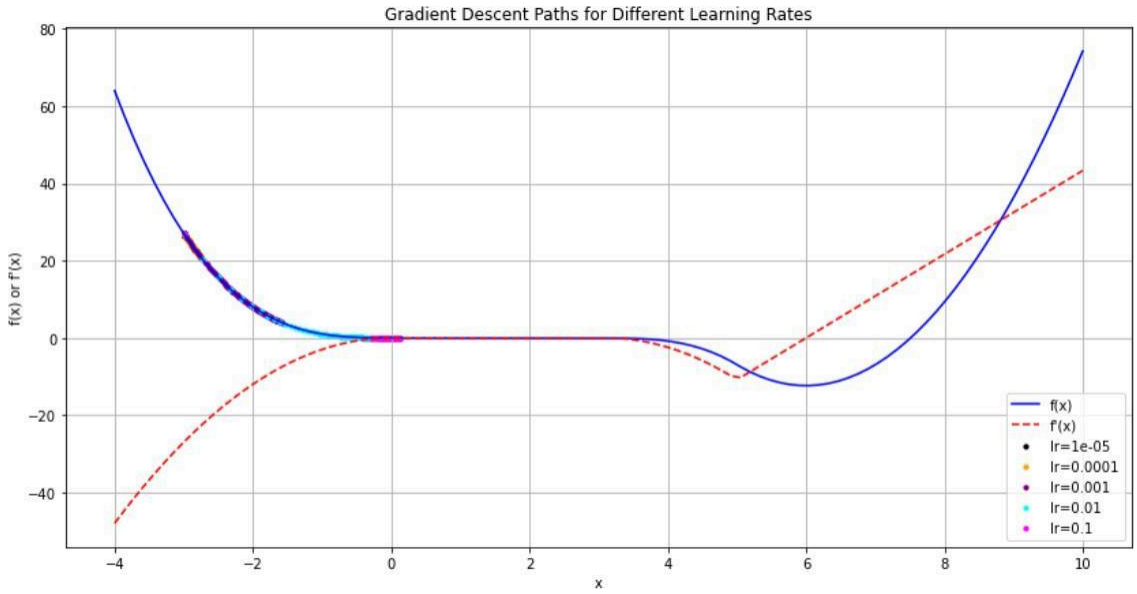
Problem 3: Gradient descent: what can go wrong?

a)



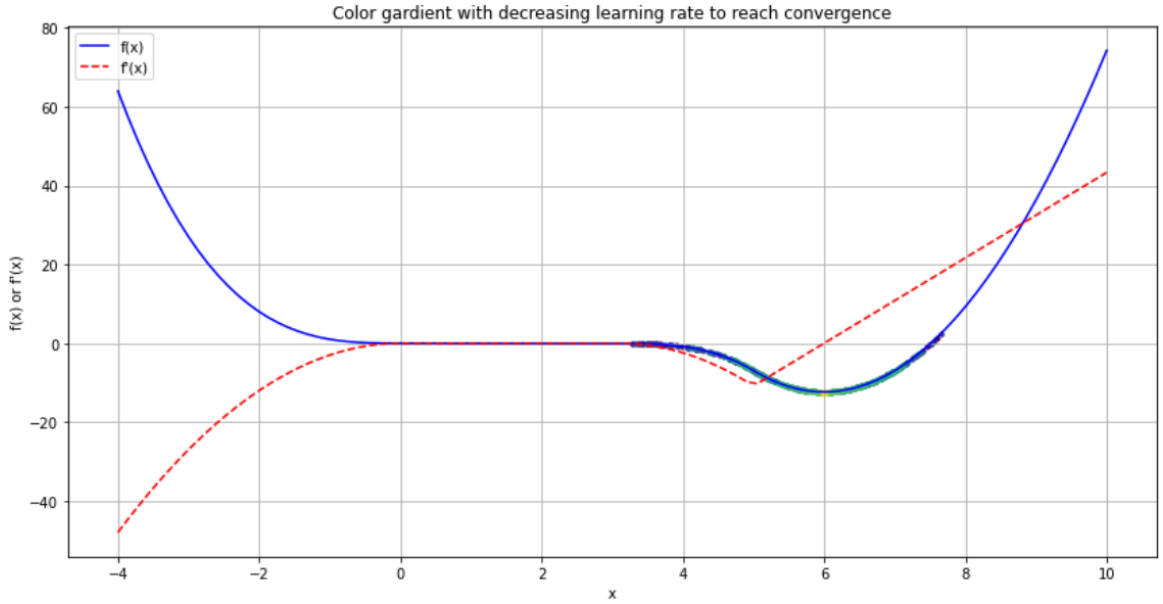
For learning rates **1e-5,1e-4,1e-3,1e-2 and 1e-1** , we observe gradual descent towards the **global minimum**. The lower learning rates take a very long time to converge at T>> 100 iterations, hence the delay compared to other learning rates. As we increase the learning rate, the convergence is faster towards the minimum. However, there are 2 key observations:

1. For all these mentioned learning rates, **the descent lies in the plateau region** near the **y=0 line**, suggesting an i**ncrease in learning rate** to successfully **cross the plateau** and reach the minimum.
2. For learning rate **1e0 and 1e1**, the loss functions overshoots and we observe a very **high** amount of **divergence** closing infinity, hence were not suitable for the experiment.



To reach the global minimum, we found that a learning rate between **0.22 and 0.25** allows us to pass through the minimum of f(x), but it does not converge and instead oscillates around the minimum. By gradually **decaying the learning rate** by a small fraction each time it gets close to the minimum, we are able to achieve convergence. Therefore, we started with a learning rate of 0.245, **decaying it by 10% each time** over 1000 iterations.

Instead of relying on the gradient to reach zero at the minimum, we set a tolerance of 1e-6 to stop the iteration once it is very close to the minimum. With this approach, we eventually converged to the minimum with a **learning rate of 0.16**, starting from 0.245.



The color intensity increases with each point and is maximum at the global minimum as shown above.

b) Speculate how the SGD trajectory would look if the learning rate were made to be very small (e.g., 100x smaller than in the figure above).

Part (i) : In the given scatter plot, we have a zig-zag pattern about the **vertical axis**.

1. We observe that the optimization process has a **high variance** in its model parameter updates, **oscillating** about an optimum value. If we see from a **learning rate** pov, the value for the given plot is on the **higher end** causing it to **overshoot** from the optimum values and oscillate back and forth.

2. The oscillation is only about the **x1-axis** implying the **gradients are uneven** and that the **curvature** of the surface **varies** along different directions.

If we **reduce the learning rate** a lot more (100x) compared to the learning rate as shown in the plot, then

1. We would likely observe a **dense cluster of points slowly converging** in the direction of gradient towards the **minimum**, however with a much shorter distance between two consecutive points.
2. Since the points will be a lot more finely spaced, the path would be **gradual and smooth** with **minimum oscillations** about the optimum value as the small learning rate will **reduce the tendency to overshoot.**
3. Path will be **stable** , however the **time to converge** will be much **longer** compared to the previous learning rate.

Part (ii) : Why is the zig-zag more stronger in one direction compared to the other?

The loss func has different curvature along diff directions as mentioned in part (i). The Curvature along x1 might be steeper than x2, resulting in longer gradient descent steps in that direction, resulting in higher oscillations and the resulting zig-zag pattern more dominant in one direction than the other. For any 2D function, the double derivative f''(x) helps us understand the curvature in each of the directions, namely the constants a1 and a2.

Since, the curvature along x1 is more steeper and its gradient is more dominant resulting in a zig-zag pattern, we conclude f''(x1) > f''(x2), implying a1>>a2 (significantly observed on a scatter plot).

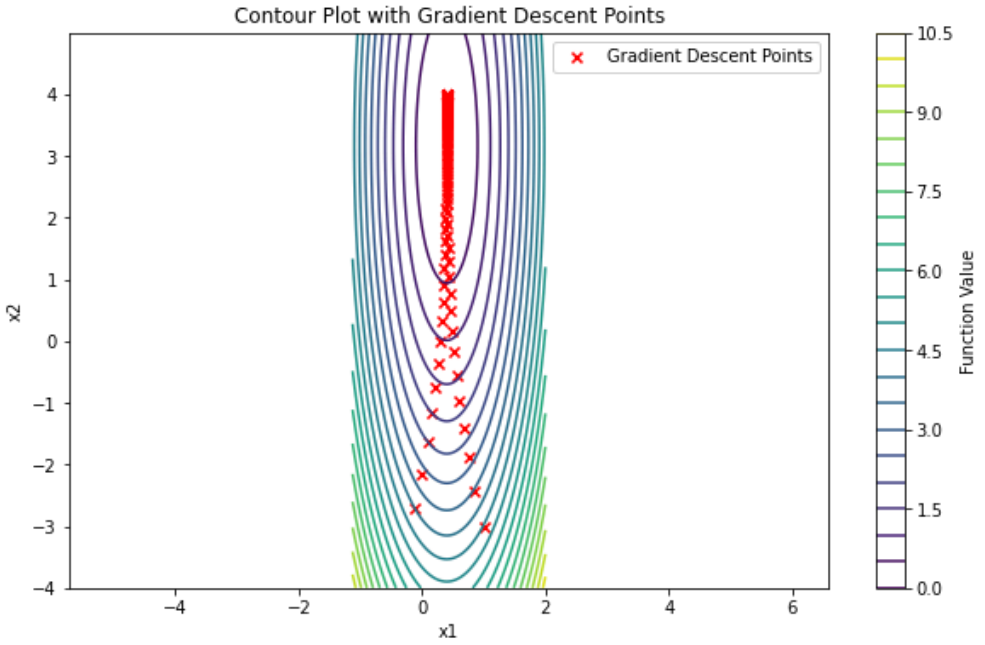
We chose c1 and c2 constants defining the origins of the function plot as mean(x1) and mean(x2) for the following reasons:

1. They represent the central location of the data in the respective dimensions. Using these means as the starting point for contour plotting or gradient descent visualization starting from a point that is representative of the overall distribution of the data.
2. Faster convergence with minimum number of iterations.

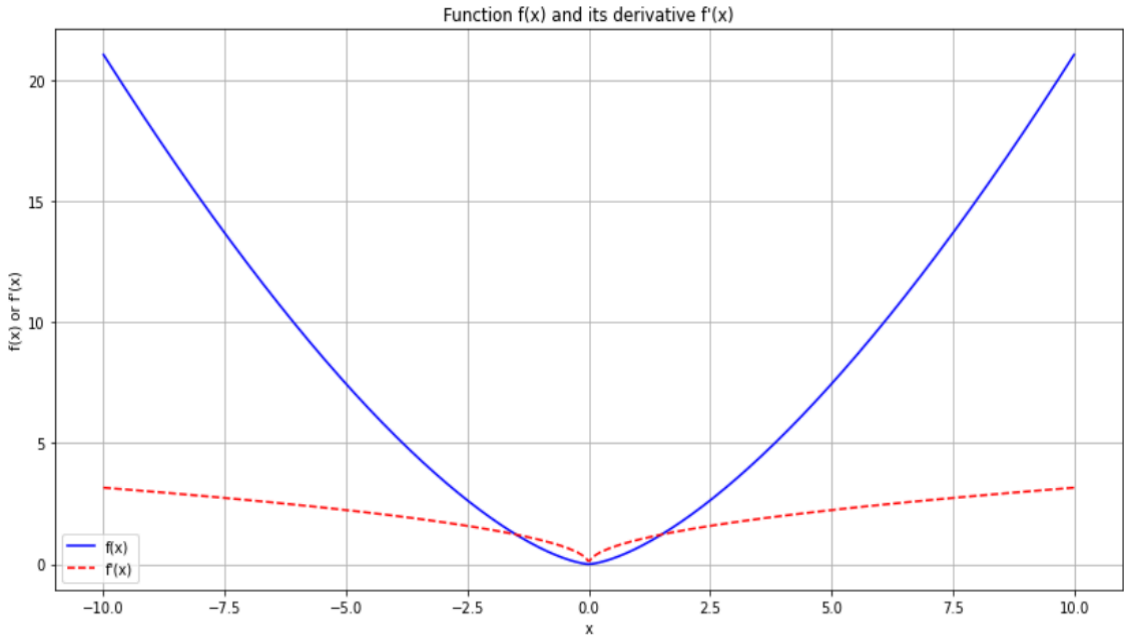
Based on the drawn conclusions, we chose a1,a2,c1,c2 as follows:

```
a1 = 2.0
a2 = 0.1
c1 = np.mean(x1)
c2 = np.mean(x2)
```

Based on the given gradient descent sequence and the chosen function constants, we obtained the contour graph and the superimposed scatter plot as shown below.



c) Testing the function behavior f(x) = 2/3 |x|3/2 for different learning rates and starting points.



Based on the paper, the given function has only one equilibrium at the origin. There are two categories of initialization of starting points for convergence.

1. X0 belongs to a set S , such that,

$$\mathcal{S} = \left\{ 0, \delta^2, -\delta^2, \frac{3+\sqrt{5}}{2}\delta^2, -\frac{3+\sqrt{5}}{2}\delta^2, \dots \right\}$$

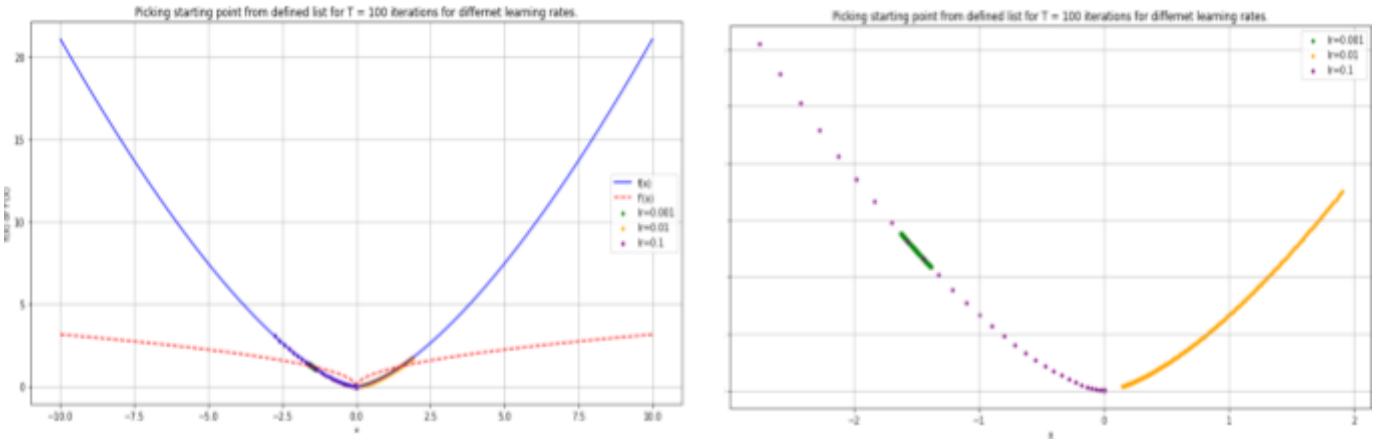
   Where, delta is the learning rate. In this set of initialization, the gradient descent converges to the global minimum,i.e at the origin.

2. When x[0] is drawn randomly from a continuous distribution, it does not converge at origin,yet, converges to an oscillation between:

$$\delta^2/4 \text{ and } -\delta^2/4$$

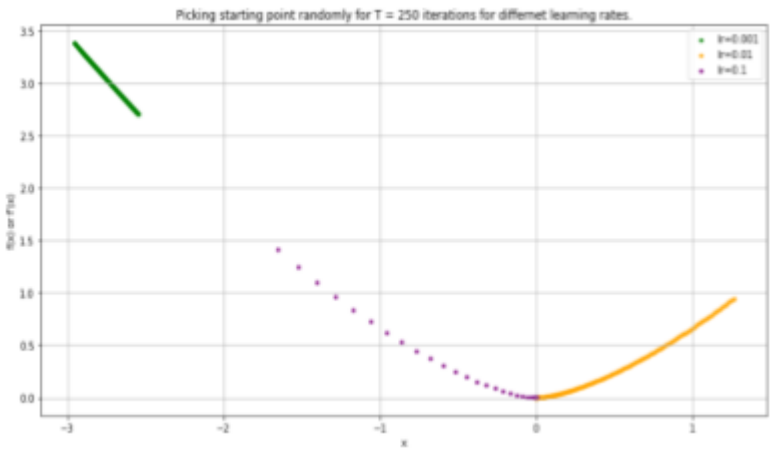
For visualization and analysis, we defined a set as mentioned above and also a collection of random values from a normal distribution. We pick starting points from either way and test them at different learning rates and number of iterations.

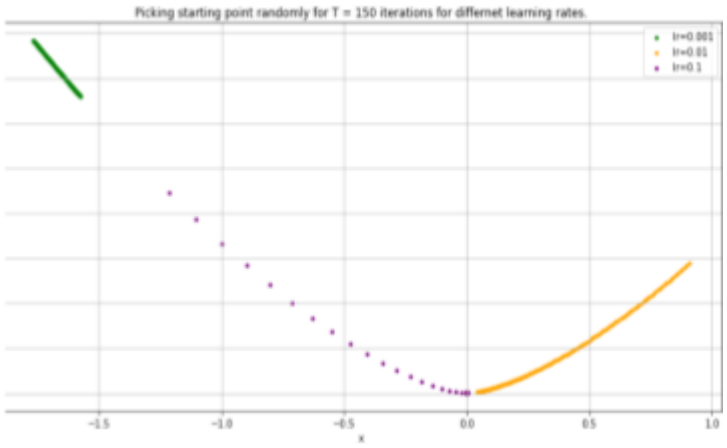
Example 1.



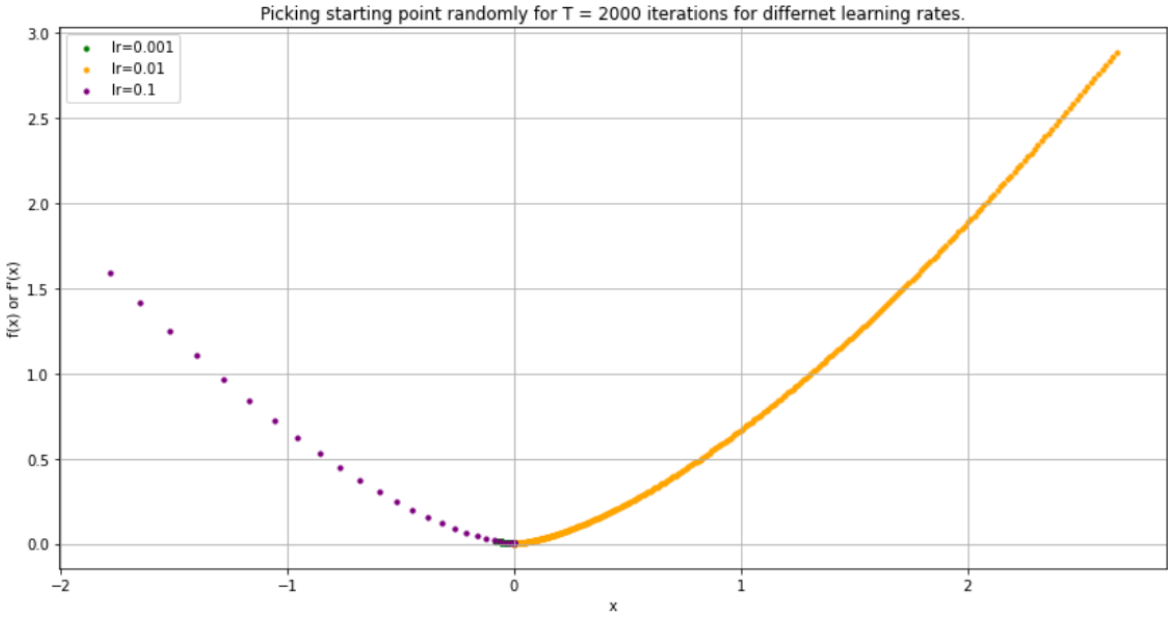
Here, for each learning rate we choose a different starting point from the set S(very close to the origin) and see how it converges to the minimum. Here for T=100 iterations, learning rate 1e-2 and 1e-1 fell short. So we can cover it by increasing the number of iterations.

Example 2:



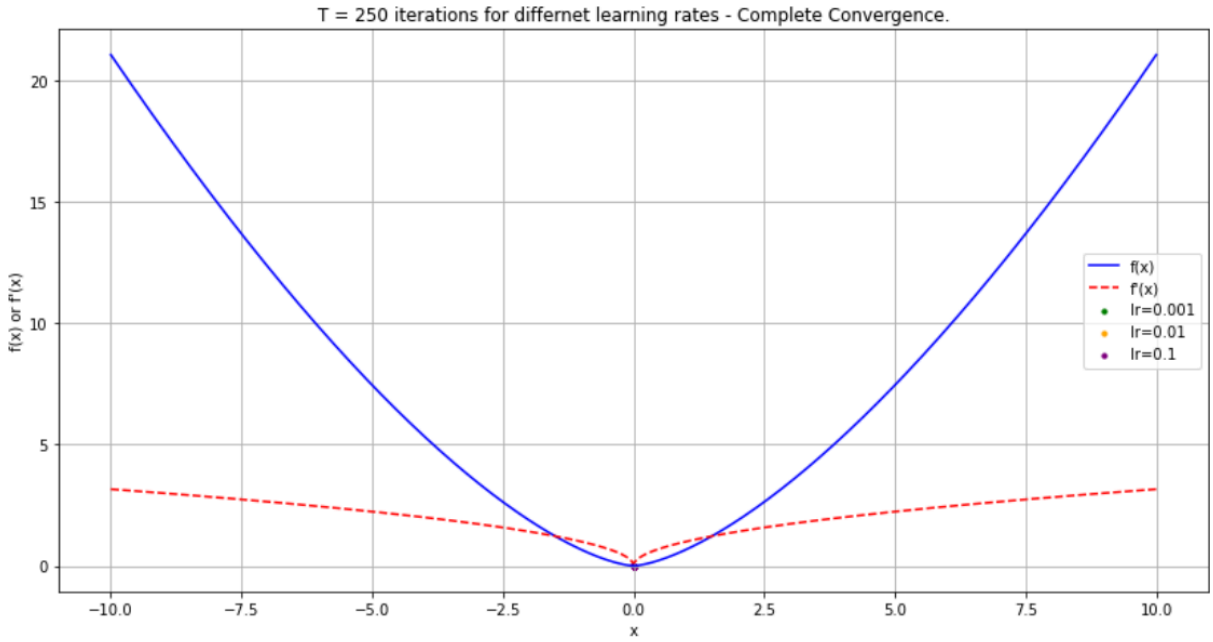


Here, for each learning rate we choose a different starting point randomly from a gaussian distribution and see how it converges to the minimum. In this case it has been observed to oscillate around the origin in a close region of convergence. Here for T=150 and T = 250 iterations, learning rate 1e-3 fell short.but 1e-1 and 1e-2 oscillated around the origin. On increasing T to a very high value we observe,



On printing the history of the gradient descent, we can observe the oscillation for selecting the starting point randomly outside the defined set of convergence.

Example 3: On increasing the number of iterations compared to Example 1, we observe complete convergence to the origin.



On printing the history of the gradient descent, we can observe complete convergence at the origin.

```
0.          0.          0.          0.          0.          0.
0.          0.          0.          0.          0.          0.
0.          0.          0.          0.          0.          0.
0.          0.          0.          0.          0.          0.
0.          0.          0.          0.          0.          0.
0.          0.          0.          0.          0.          0.
0.          0.          0.          0.          0.          0.
0.          0.          0.          0.          0.          0.
0.          0.          0.          0.          0.          0.
0.          0.          0.          0.          0.          0.
0.          0.          0.          0.          0.          0.
0.          0.          0.          0.          ]
```

d) Achieving Local Maximum after one descent iteration.

For **f(x) = x^4 - 4*x^2** on differentiating we obtain **x=0, sqrt(2),-sqrt(2)**. We observe **x=0 to be the point of local maximum**.

Now for one descent iteration from x=x_initial to x=0, we need to find the x_initial and the corresponding learning rate.

On calculating, we obtained **x_initial = sqrt(3)** and **learning_rate = 0.25**, to be one such combination. To validate we check the first decent value for the calculated starting point and learning rate and we obtain x=0,thereby, proving the required result.