Learning how to learn

Descending down steep curves

Amish Mittal

Gradient Descent

ANN: $Y = h(\theta, X)$

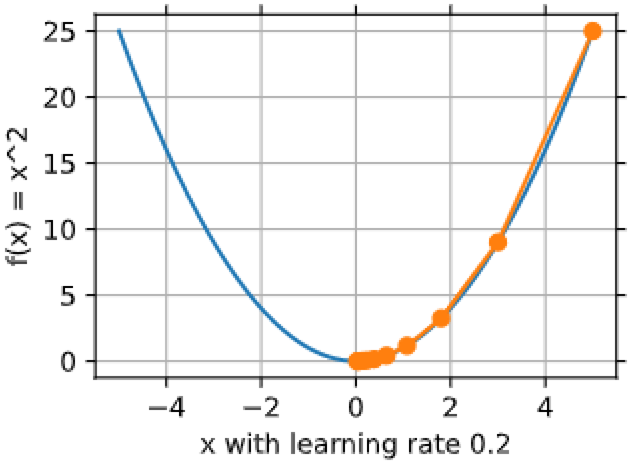
Goal: Approximate correct value of Y by fixing h and then estimating θ using loss function f

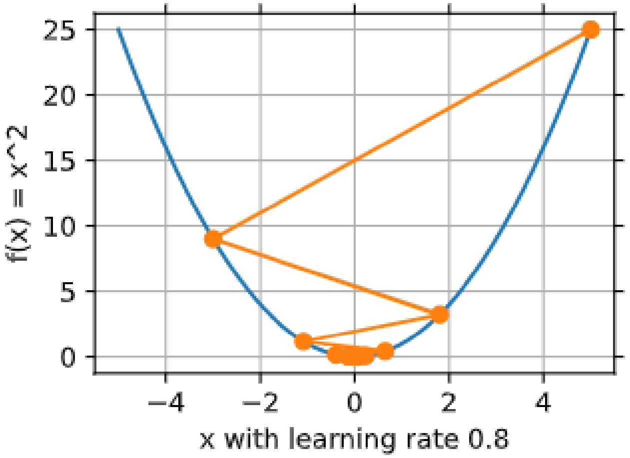
Ideal: $\text{argmin}_\theta \sum_{i=0}^{n} f(x_i, y_i)$

Gradient descent: $\theta_{t+1} = \theta_t - \eta \nabla f$

Vanilla Stochastic GD: $\Delta\theta = -\eta\nabla f$

$f = x^2$





Issue:

1. Highly susceptible to chosen learning rate. We don't know appropriate LR beforehand.
2. The pattern of descent changes. Can we make it more deterministic?

Vanilla Stochastic GD: $\Delta\theta = -\eta\nabla f$

$$f = 10x^2$$

```
show_trace(gd(0.8, f_grad, 10), f, "10x^2", 0.8)
0.4s
epoch 10, x: 2883251953125.000000
```

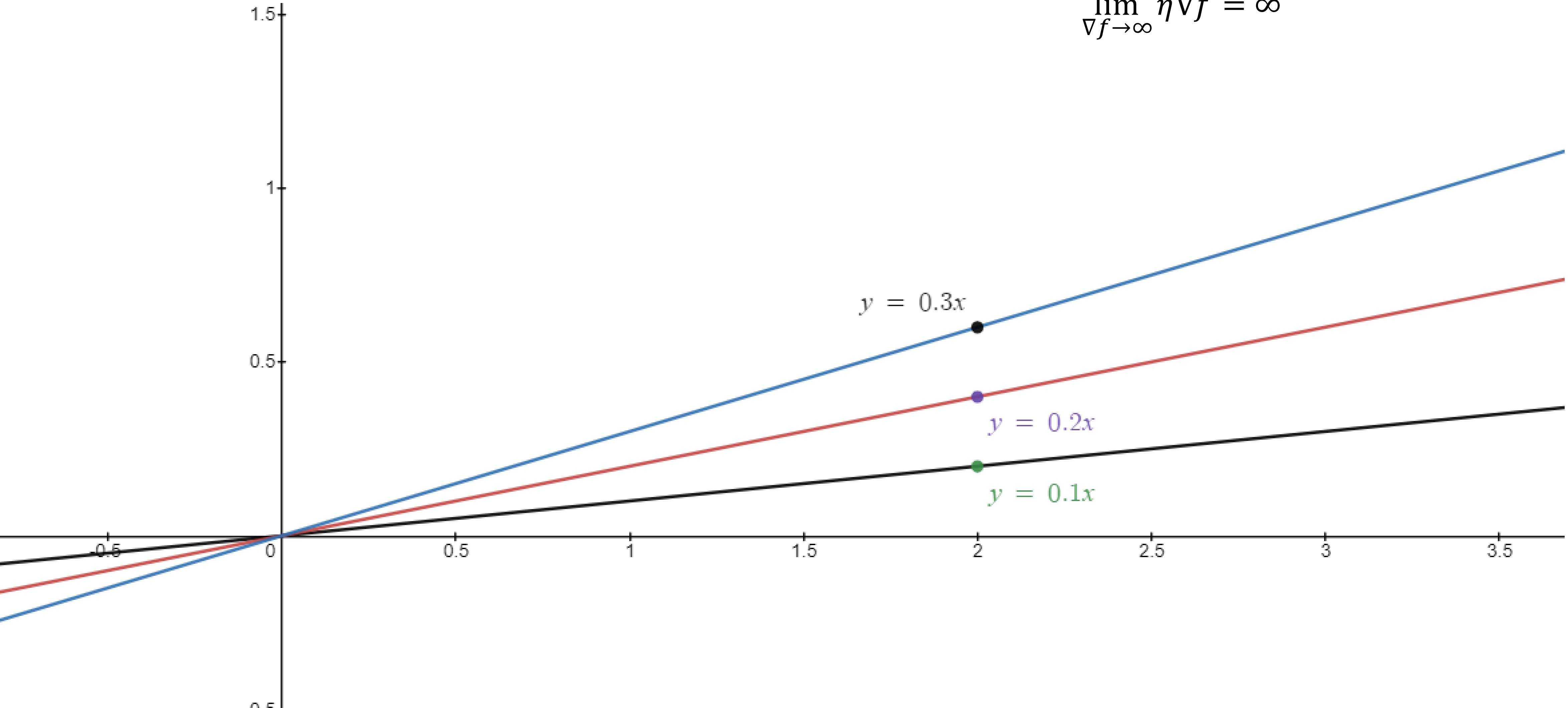
```
show_trace(gd(0.12, f_grad, 10), f, "10x^2", 0.12)
5.5s
epoch 10, x: 144.627327
```

Issue:
Gradient is way too high at our initialization point (89.4º). The optimization hence diverges even with low learning rate.

Nature of $\Delta\theta = \eta\nabla f$

$$\lim_{\nabla f \to 0} \eta\nabla f = 0$$

$$\lim_{\nabla f \to \infty} \eta\nabla f = \infty$$



Popular Optimizers

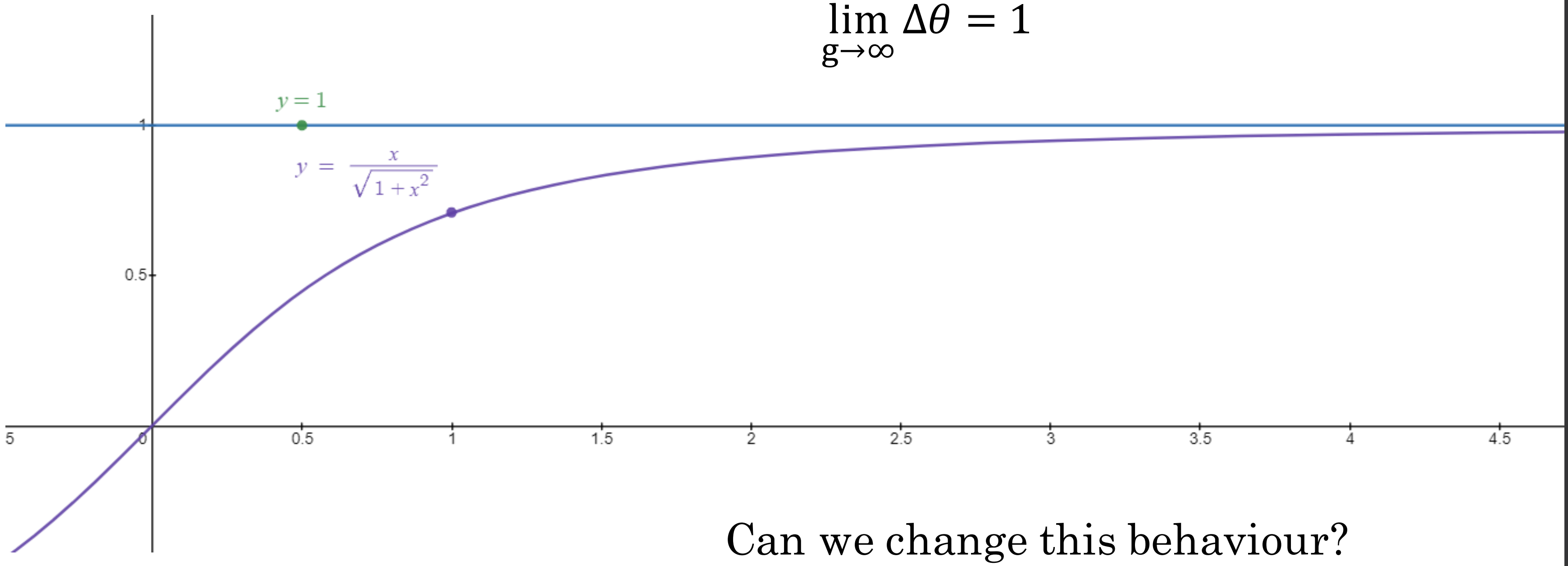
- Momentum
- Nesterov Momentum
- Adagrad
- Adadelta
- RMSprop
- Adam

etc…

$$\Delta\theta = -\frac{\eta}{\sqrt{E(g^2) + \epsilon}} \odot g$$

$$\lim_{g \to 0} \Delta\theta = 0$$

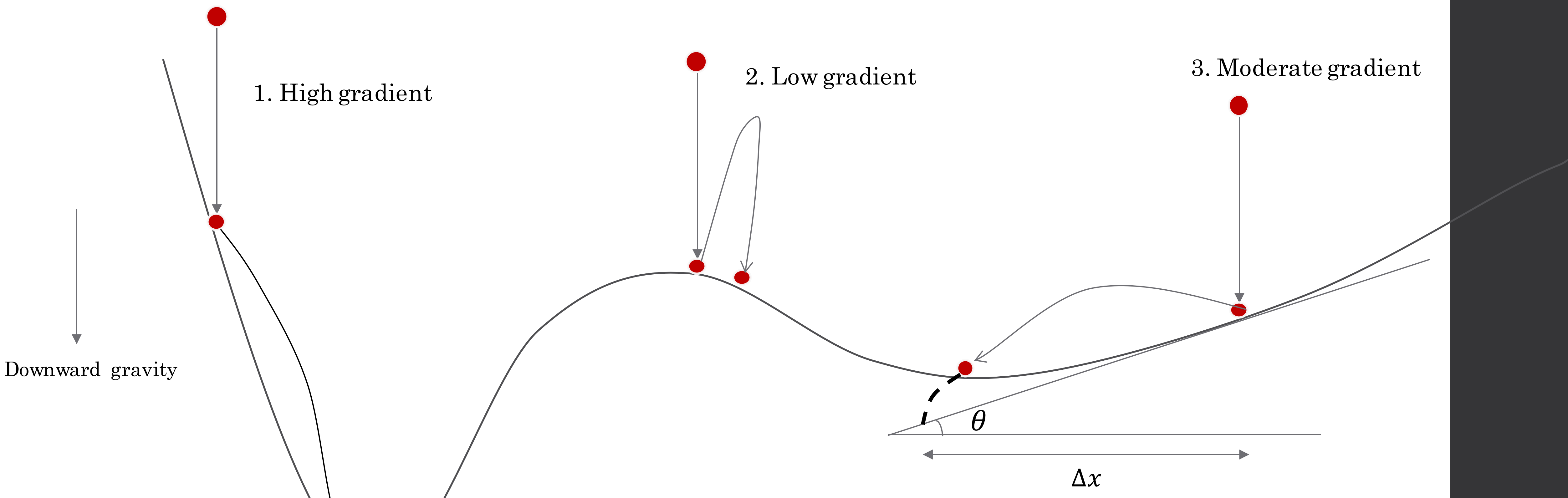
$$\lim_{g \to \infty} \Delta\theta = 1$$



Can we change this behaviour?

Taking inspiration from physics

Projectile motion on inclined plane



Let θ be the gradient at the point at which our ball collides

Then, $\Delta x \approx \sin(\theta)\cos(\theta)$

Δx → movement along parameter space assuming complete projectile motion without loss function

Trying to move over the loss function space instead of the parameter space

With known $\tan(\theta) = \nabla f$,

$$\therefore\ \Delta x \approx -\sin(\theta)\cos(\theta) \approx -\frac{\nabla f}{1 + |\nabla f|^2}$$

Expanding this to $\mathbb{R}^n$ as $\theta \in \mathbb{R}^n$,

$$\Delta\theta_i = -\frac{1}{1 + \nabla f_i^2} . \nabla f_i$$

Vectorizing to improve time complexity,

$$\Delta\theta = -\frac{1}{1 + F} \odot \nabla f$$

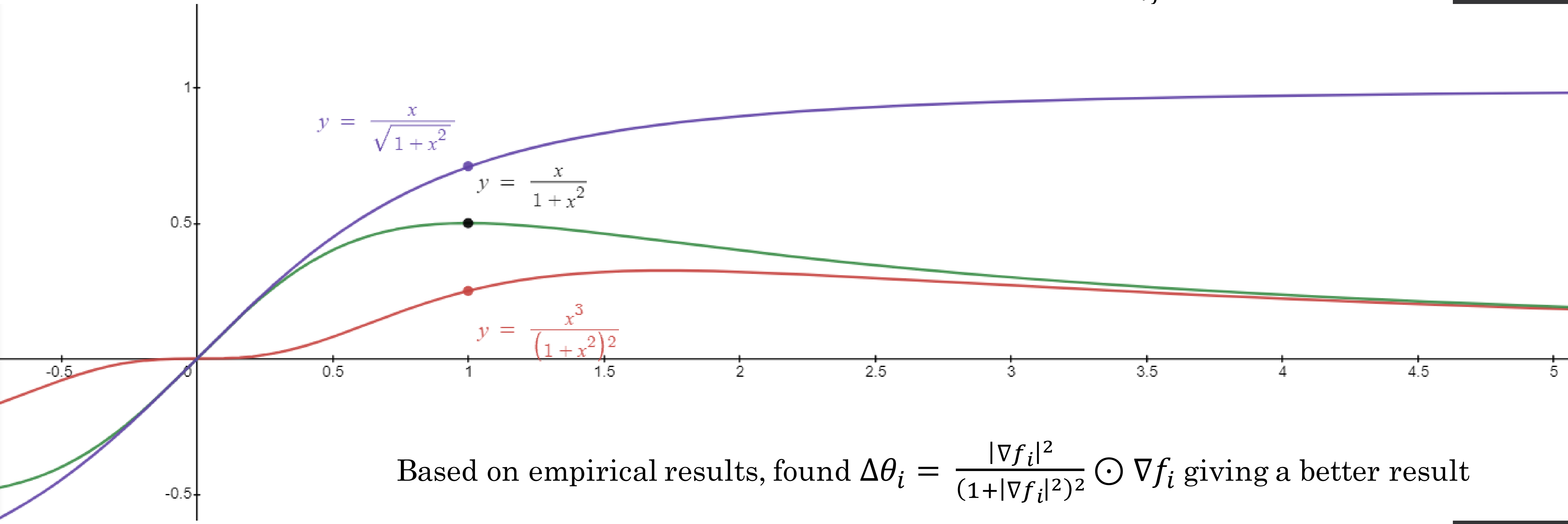
where,

$$F = \begin{bmatrix} \nabla f_0^2 & 0 & 0 & 0 \\ 0 & \nabla f_1^2 & 0 & 0 \\ 0 & 0 & \nabla f_2^2 & 0 \\ 0 & 0 & 0 & \nabla f_3^2 \end{bmatrix}$$

Nature of $\Delta\theta = \frac{\eta}{1+F} \odot \nabla f$

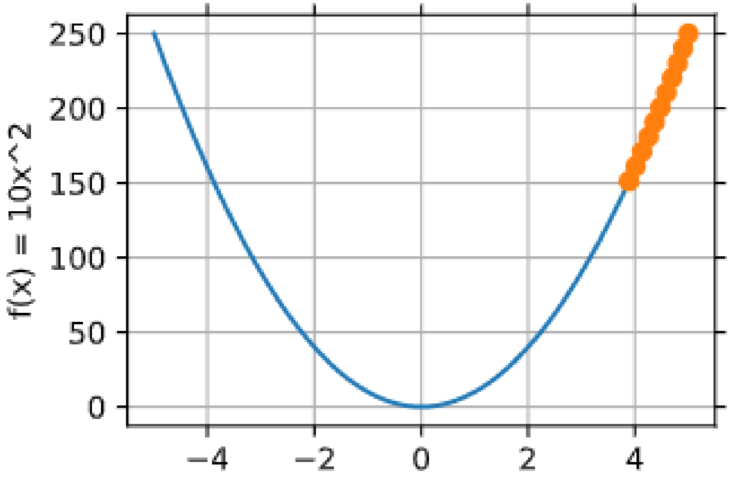
$$\lim_{\nabla f \to 0} \Delta\theta = 0$$

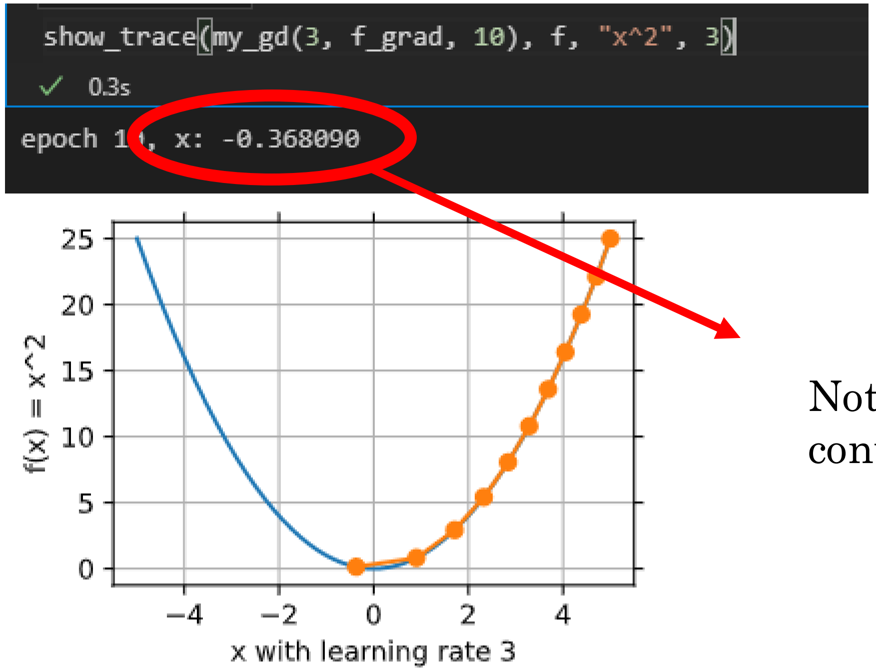
$$\lim_{\nabla f \to \infty} \Delta\theta = 0$$



Based on empirical results, found $\Delta\theta_i = \frac{|\nabla f_i|^2}{(1+|\nabla f_i|^2)^2} \odot \nabla f_i$ giving a better result

$$\Delta\theta = -\frac{\eta}{1+F} \odot \nabla f \text{ (added a constant LR)}$$



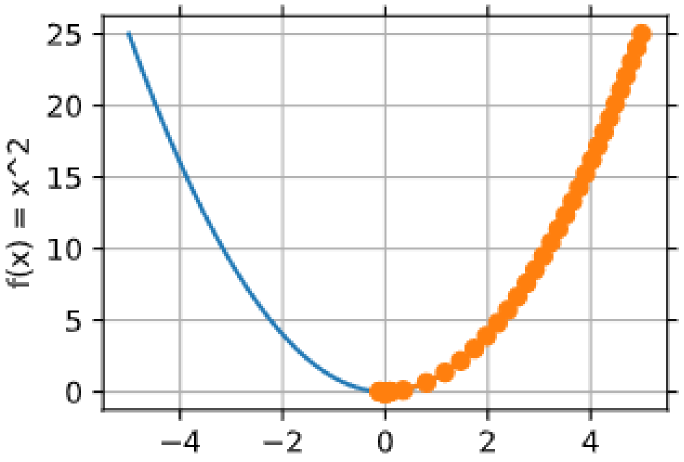


Not a good convergence

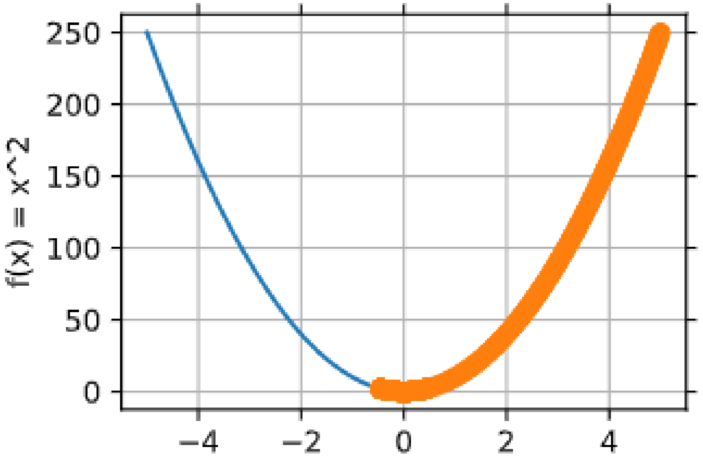
Issue:
For large value to gradient, the value becomes close to zero leading to very small iterations. So, we increase speed by increasing learning rate. However,
for small value of gradient, the LR is quite large to get a good convergence.

$$\Delta\theta_i = -\frac{|\nabla f_i|^2}{(1+|\nabla f_i|^2)^2} \odot \nabla f_i \text{ (no extra hyperparameter)}$$

$f = x^2$

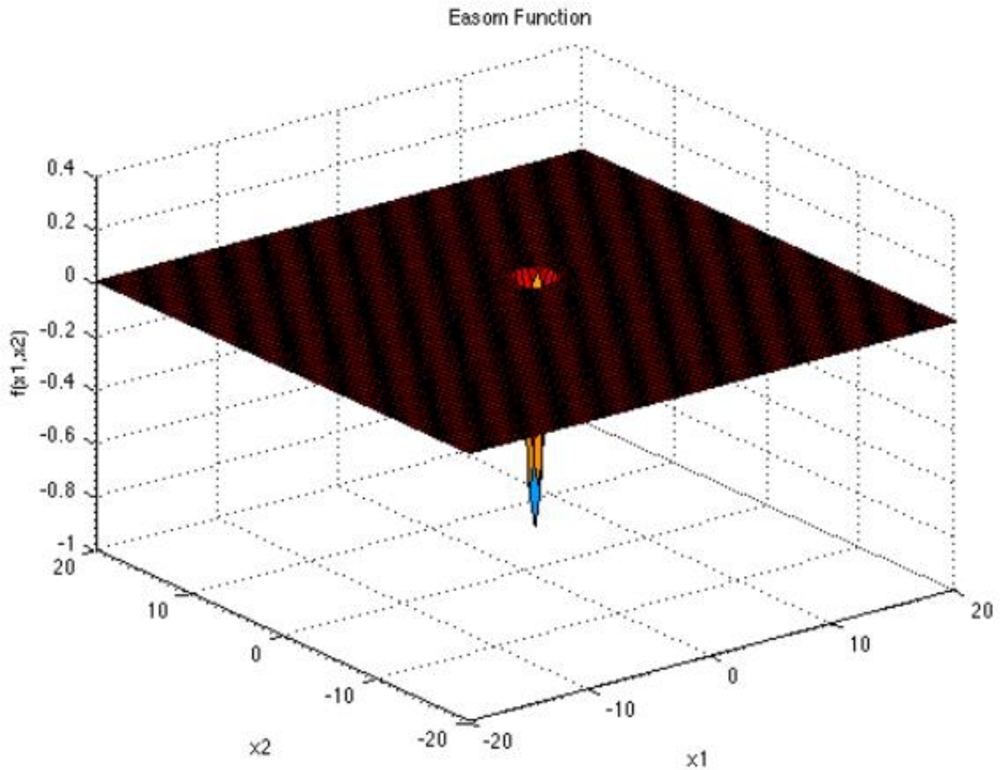


$f = 10x^2$



Issue:
The GD converges in a uniform manner, but the per epoch updates are extremely small → large number of iterations are needed

Easom functions



$$f(\mathbf{x}) = -\cos(x_1)\cos(x_2)\exp\left(-(x_1-\pi)^2-(x_2-\pi)^2\right)$$

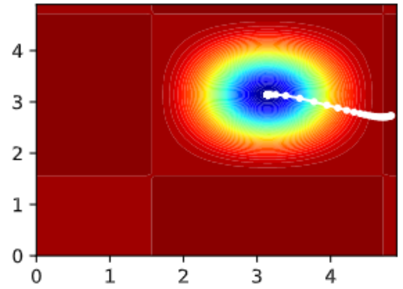
Global Minimum:

$$f(\mathbf{x}^*) = -1 \text{ , at } \mathbf{x}^* = (\pi, \pi)$$

Source: https://www.sfu.ca/~ssurjano/easom.html

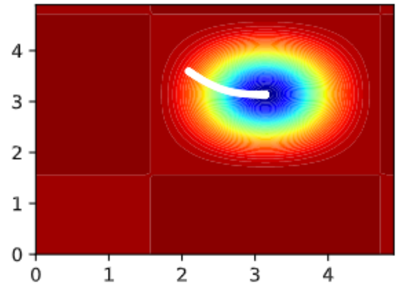
Adadelta

```
&gt;9997 f([3.14159265 3.14159265]) = -1.00000
&gt;9998 f([3.14159265 3.14159265]) = -1.00000
&gt;9999 f([3.14159265 3.14159265]) = -1.00000
```



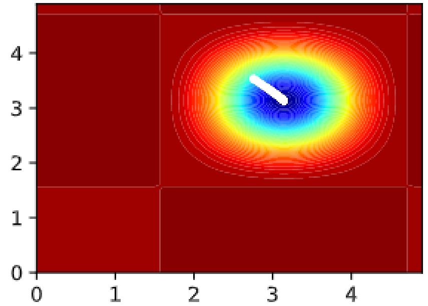
46 iterations

```
&gt;9997 f([3.14159265 3.14159265]) = -1.00000
&gt;9998 f([3.14159265 3.14159265]) = -1.00000
&gt;9999 f([3.14159265 3.14159265]) = -1.00000
```



90 iterations

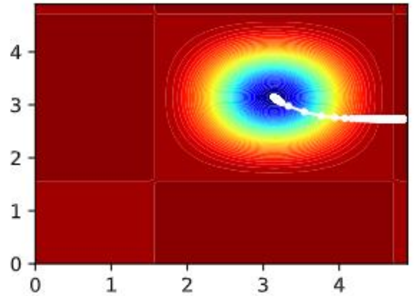
```
&gt;9997 f([3.14159265 3.14159265]) = -1.00000
&gt;9998 f([3.14159265 3.14159265]) = -1.00000
&gt;9999 f([3.14159265 3.14159265]) = -1.00000
```



70 iterations

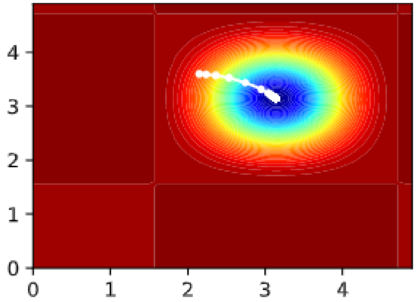
My Optimizer (biggest advantage – no worry about any hyperparameter – at least for now)

```
&gt;9997 f([3.14315725 3.14002806]) = -0.99999
&gt;9998 f([3.14315714 3.14002817]) = -0.99999
&gt;9999 f([3.14315704 3.14002827]) = -0.99999
```



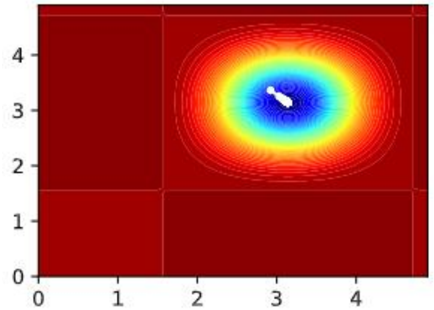
2400 iterations
(stuck in start)

```
&gt;9997 f([3.14023148 3.14295382]) = -0.99999
&gt;9998 f([3.14023155 3.14295376]) = -0.99999
&gt;9999 f([3.14023162 3.14295369]) = -0.99999
```



10 iterations

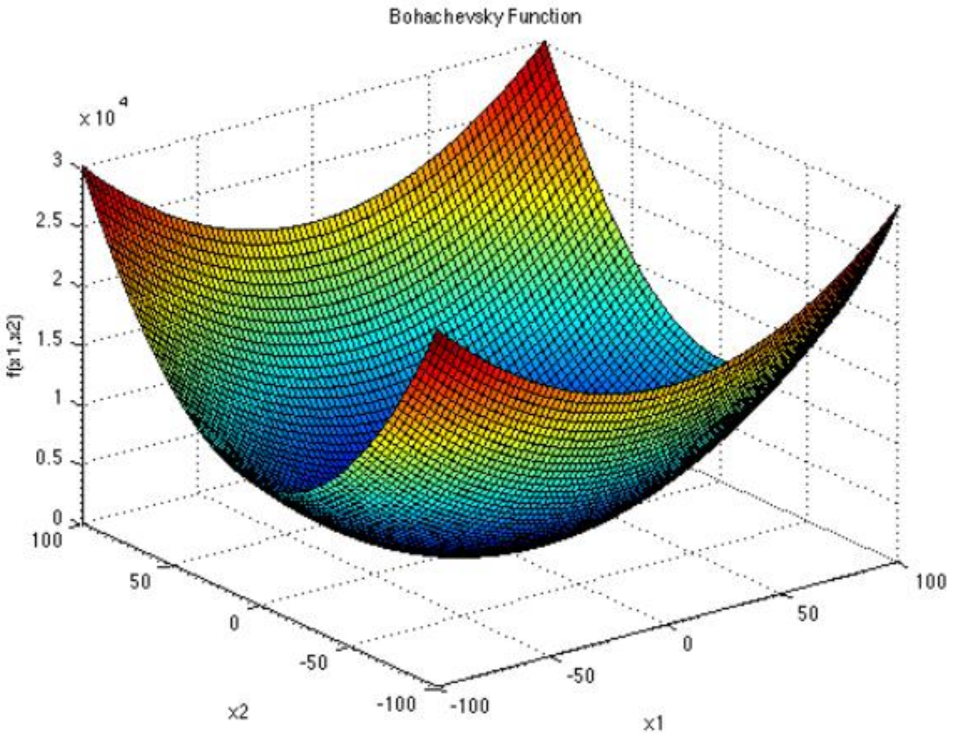
```
&gt;9997 f([3.14023181 3.1429535 ]) = -0.99999
&gt;9998 f([3.14023187 3.14295343]) = -0.99999
&gt;9999 f([3.14023194 3.14295337]) = -0.99999
```



10 iterations

Starting points randomly initialized

Bohachevsky functions



$$f_1(\mathbf{x}) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$$

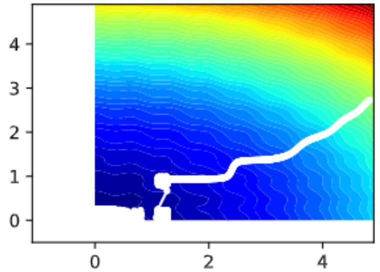
Global Minimum:

$f_j(\mathbf{x}^*) = 0$, at $\mathbf{x}^* = (0, 0)$, for all $j = 1, 2, 3$

Source: https://www.sfu.ca/~ssurjano/boha.html

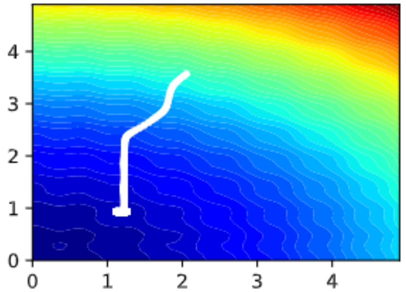
Adadelta (using hyperparameter tuning)

```
&gt;9996 f([-0.23646039 -0.23093087]) = 1.43455
&gt;9997 f([ 0.1988228  -0.09102323]) = 0.67999
&gt;9998 f([-0.29716366  0.233268  ]) = 1.57106
&gt;9999 f([-0.04848432  0.10222546]) = 0.34111
```



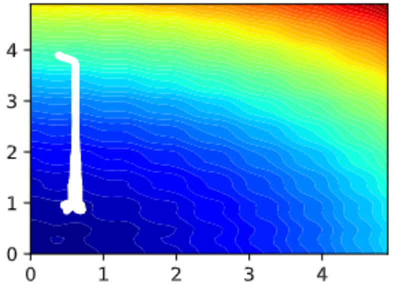
9999 iterations - diverging

```
&gt;9997 f([1.262151   0.93337914]) = 3.53261
&gt;9998 f([1.11961544 0.93337914]) = 3.55683
&gt;9999 f([1.17822396 0.93337914]) = 3.53017
```



9999 iterations

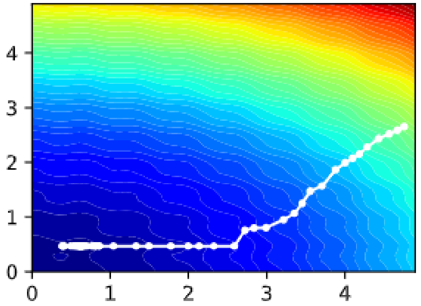
```
&gt;9997 f([0.66731963 0.8986926 ]) = 2.34328
&gt;9998 f([0.52969753 0.96759942]) = 2.40290
&gt;9999 f([0.69597394 0.86589144]) = 2.44097
```



9999 iterations

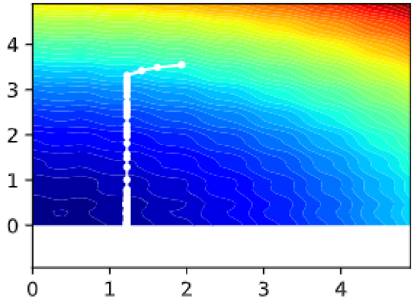
My Optimizer (biggest advantage – no worry about learning rate – at least for now)

```
&gt;9997 f([0.61855849 0.46951326]) = 0.88281
&gt;9998 f([0.61855849 0.46951326]) = 0.88281
&gt;9999 f([0.6185585  0.46951326]) = 0.88281
```



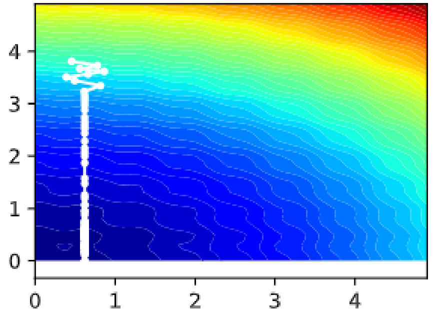
9999 iterations

```
&gt;9997 f([1.22261158 0.46954189]) = 2.11371
&gt;9998 f([1.22261158 0.46954189]) = 2.11371
&gt;9999 f([1.22261157 0.46954189]) = 2.11371
```



9999 iterations

```
&gt;9997 f([0.6185586  0.26705382]) = 1.34641
&gt;9998 f([0.6185586  0.26705382]) = 1.34641
&gt;9999 f([0.61855861 0.26705382]) = 1.34641
```



9999 iterations

Further Work

- Test on more non-convex optimization test cases
- Expand tests to more than 2 dimensions
- Add Nesterov momentum to the descent to accelerate it near high gradient values. Or use methods as outlined in Adam optimizer.
- Change implementation of GD in Tensorflow and test on deep neural networks.
- Benchmark against other optimizers and re-iterate after modifications.