Learning how to learn

Descending down steep curves

Amish Mittal

Gradient Descent

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

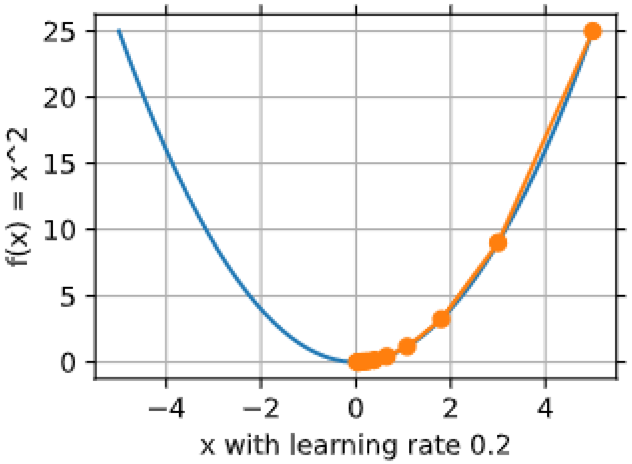
Goal: Approximate correct value of Y by fixing h and then estimating θ using loss function f

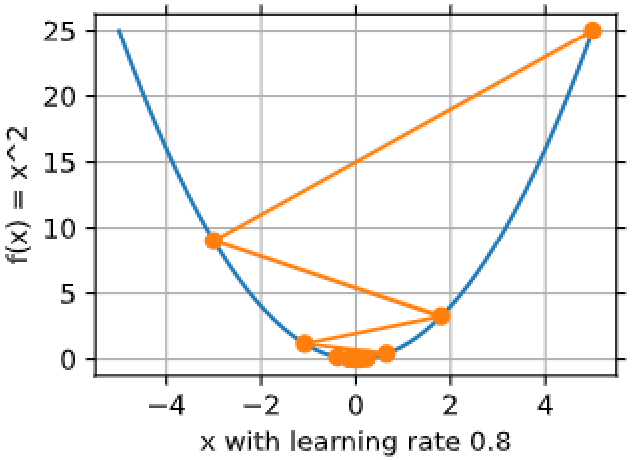
Ideal: $\text{argmin}_\theta \sum_{i=0}^{n} f(x_i, y_i)$

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: $\Delta\theta = -\eta\nabla f$

$f = 10x^2$

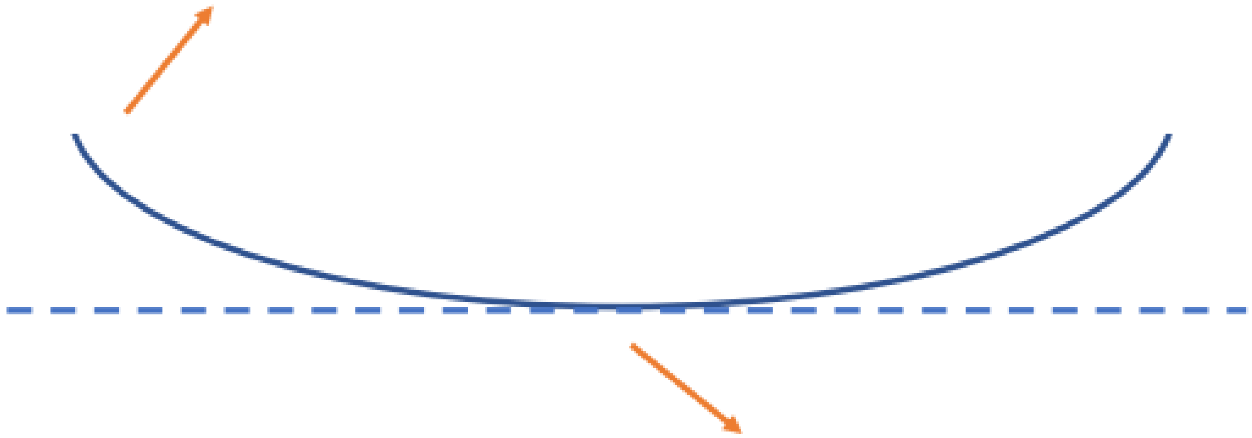
```
show_trace(gd(0.8, f_grad, 10), f, "10x^2", 0.8)
⊗ 0.4s
epoch 10, x: 2883251953125.000000
```

```
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.

Gradient Descent Convergence

Very steep so cannot
use big η



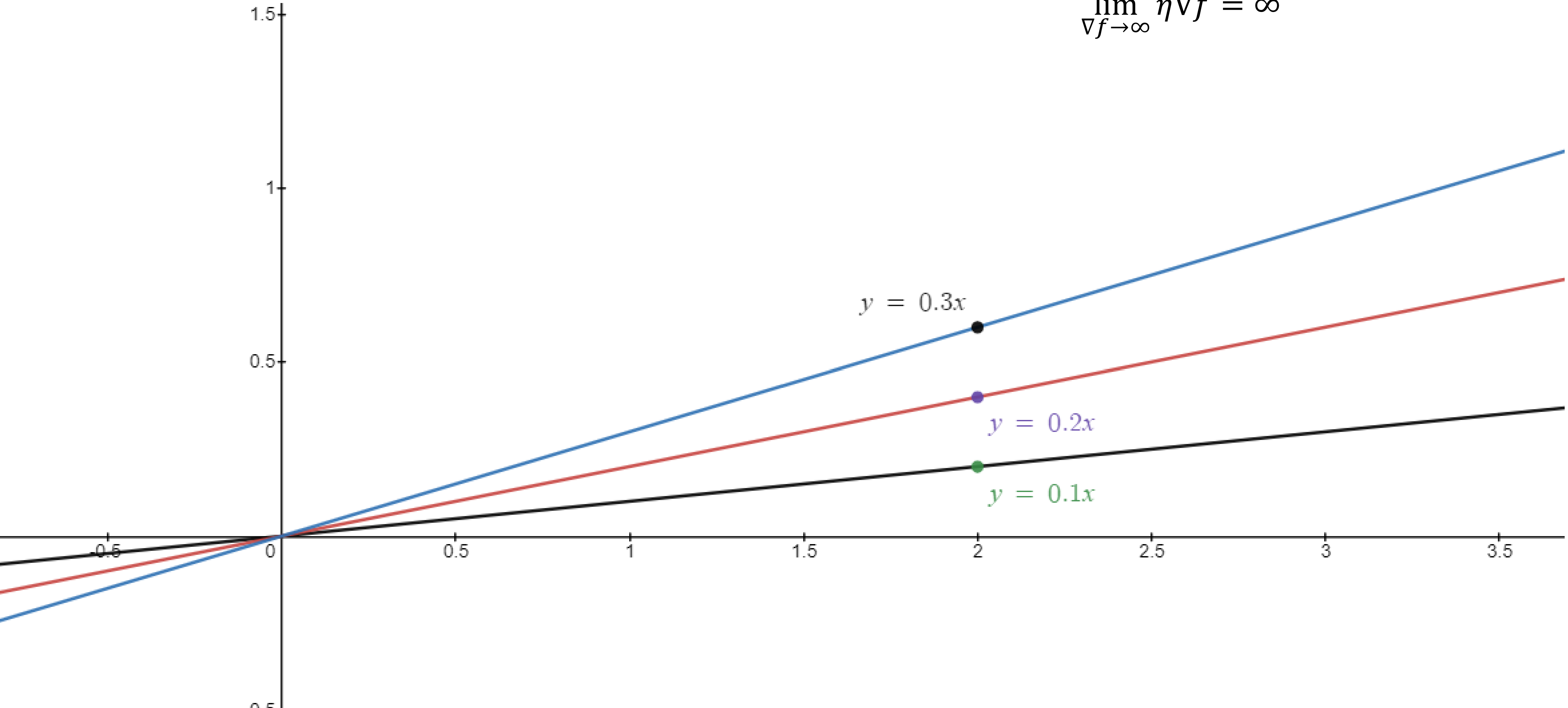
Very flat near x^* so small η
makes it slow

Source: https://praneethnetrapalli.org/

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

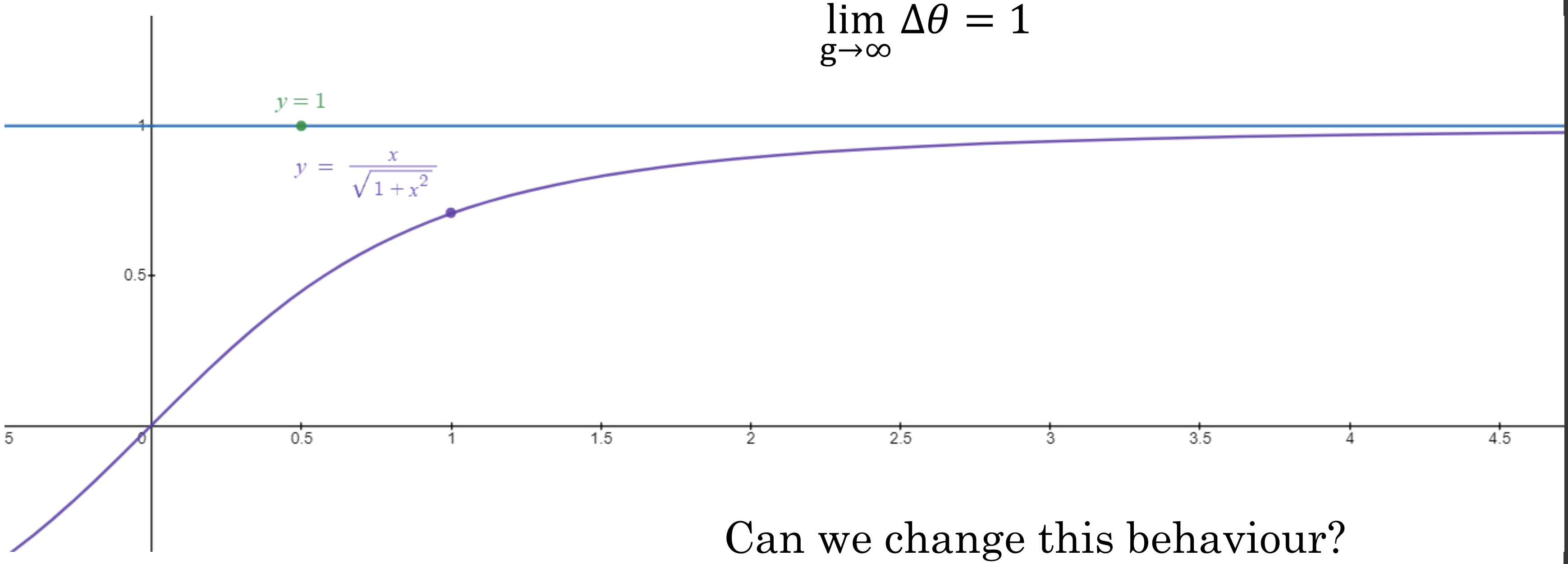
- Momentum
- Nesterov Momentum
- Adagrad
- Adadelta
- RMSprop
- Adam

etc…

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

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

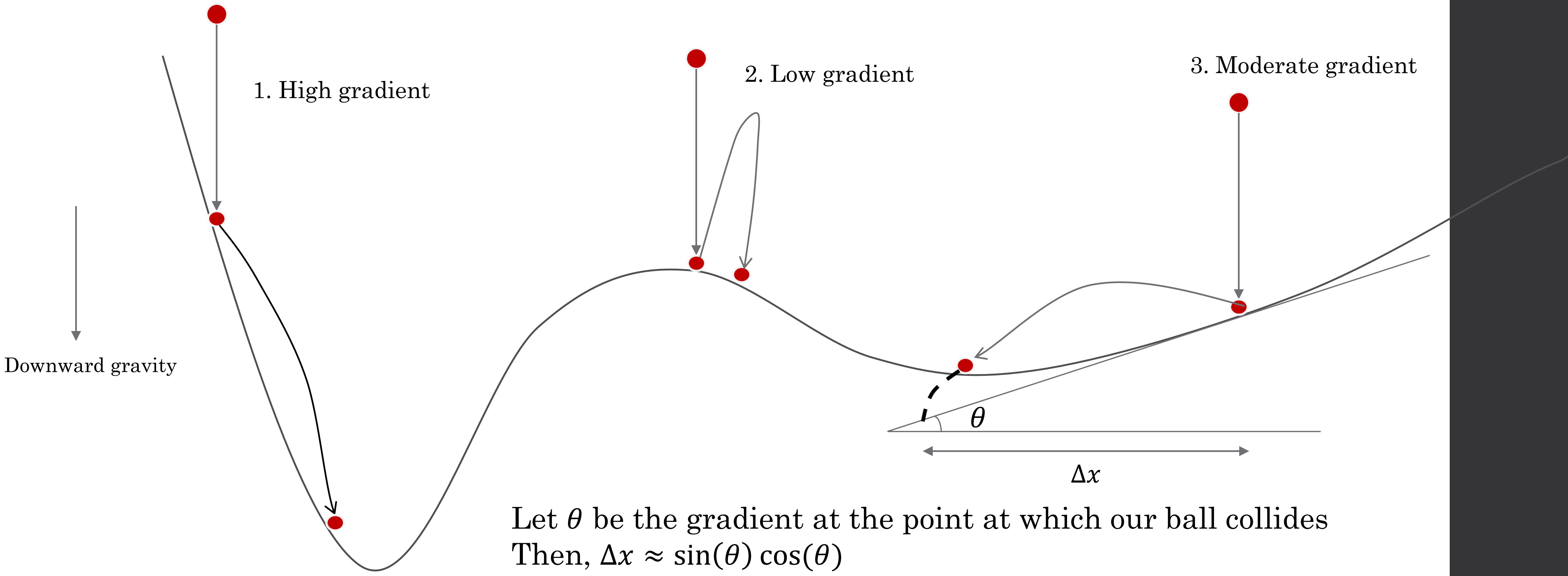
$$\lim_{g \to \infty} \Delta\theta = 1$$



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 $\rightarrow$ movement along parameter space assuming complete projectile motion without collision from 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,

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

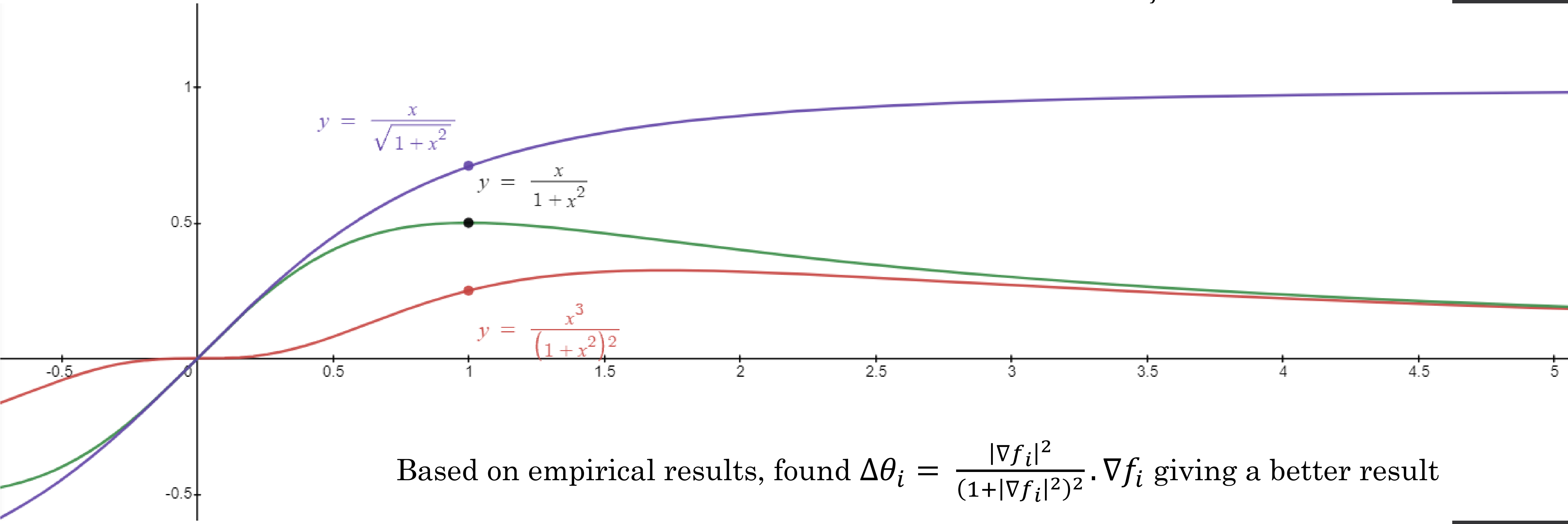
where,

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

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

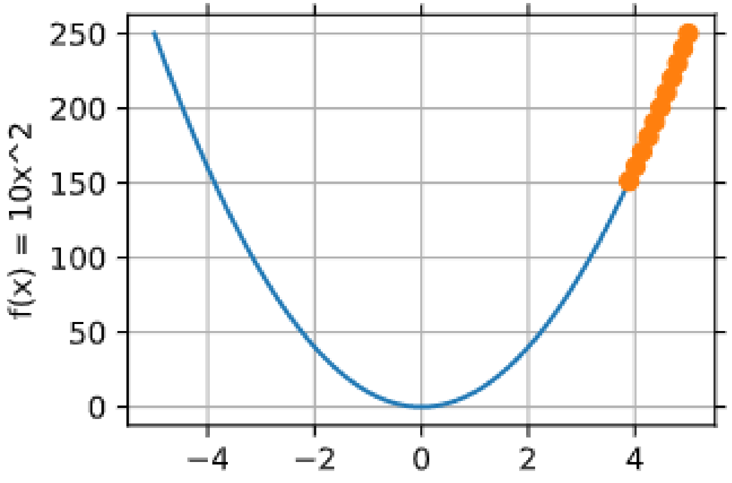
$$\lim_{\nabla f \to 0} \Delta\theta = 0$$

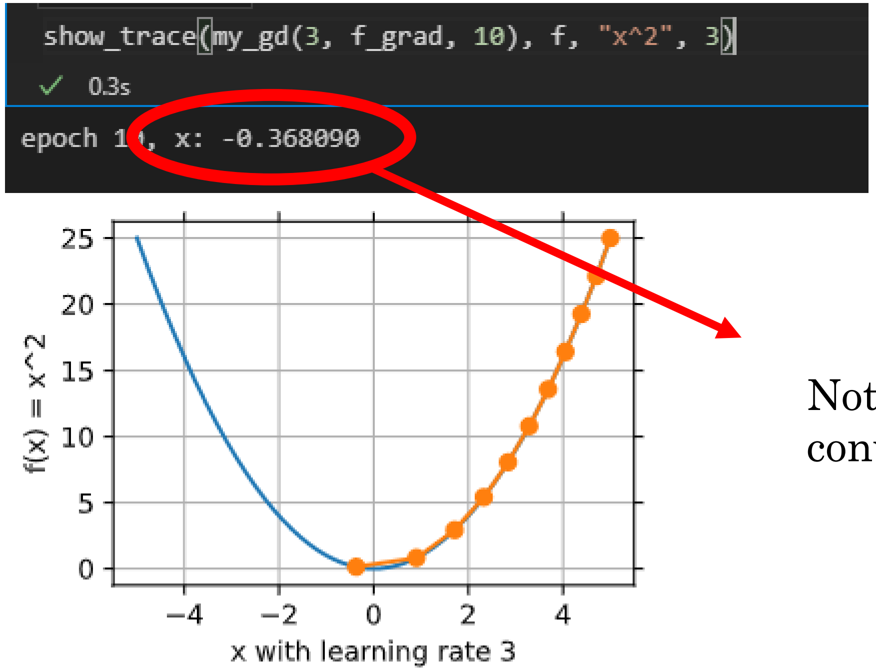
$$\lim_{\nabla f \to \infty} \Delta\theta = 0$$



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

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



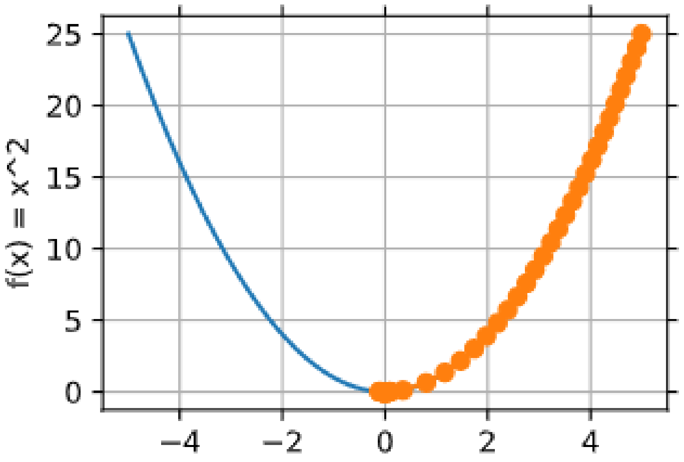


Not a good convergence

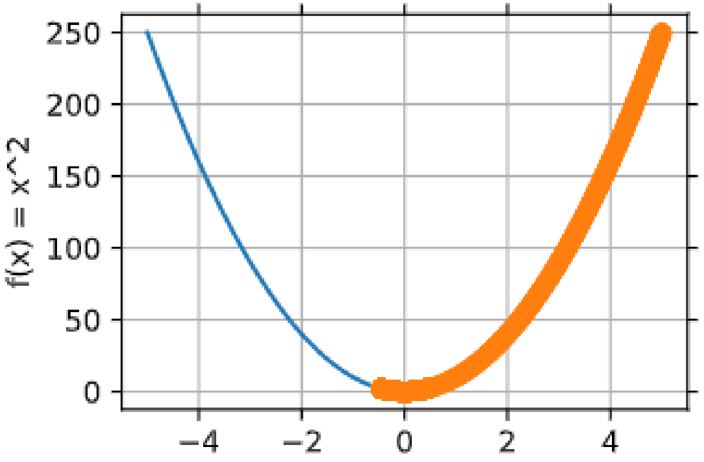
Issue:
For large value to gradient, the value becomes close to zero leading to very small iterations. So, we increase speed by increasing learning rate. However, for small value of gradient, the LR is quite large to get a good convergence.

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

$f = x^2$



$f = 10x^2$



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

Constraints of this exercise

- Have a GD optimizer which:
  1. Does not perform worse
  2. Convergence can be proved
  3. Has similar asymptotic time complexity
- Best case:
  - Gives a better convergence value
  - Takes less number of iterations

Empirical Validation Parameters

1. Does it converge?

Defined as $|\theta_{t+1} - \theta_t| \leq k$ for all $t > c$,

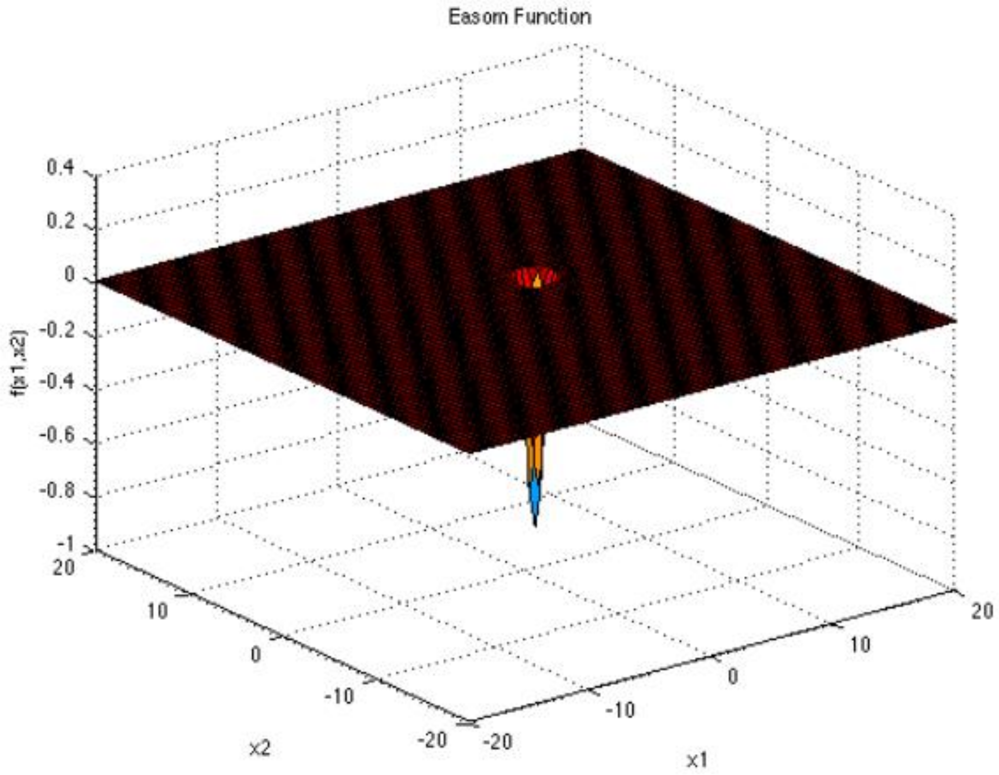
$k \rightarrow error\ parameter, \qquad c \rightarrow epochs\ required\ to\ converge$

2. Does it reach global minimum?

Defined as $|\theta_c - \theta^*| \leq k'$,

$k' \rightarrow error\ parameter, \qquad \theta^* \rightarrow global\ minima$

Easom functions



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

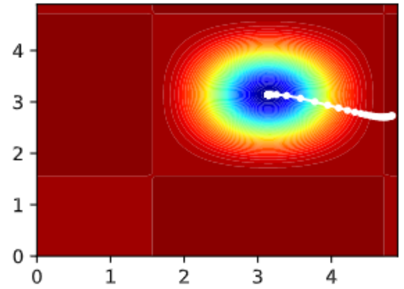
Global Minimum:

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

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

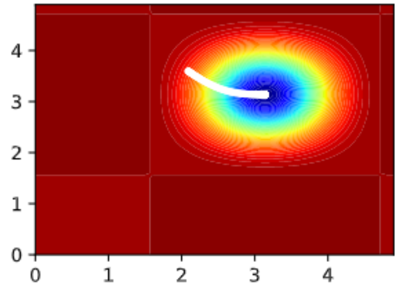
Adadelta

```
&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
```



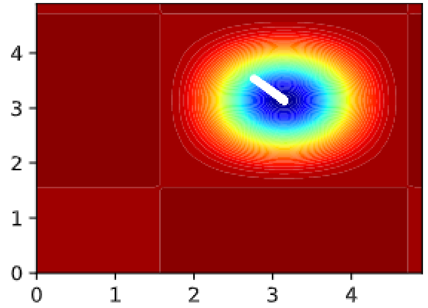
46 iterations

```
&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

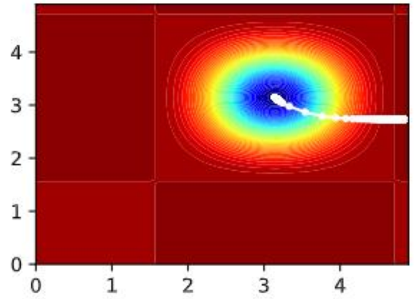
```
&gt;9997 f([3.14159265 3.14159265]) = -1.00000
&gt;9998 f([3.14159265 3.14159265]) = -1.00000
&gt;9999 f([3.14159265 3.14159265]) = -1.00000
```



70 iterations

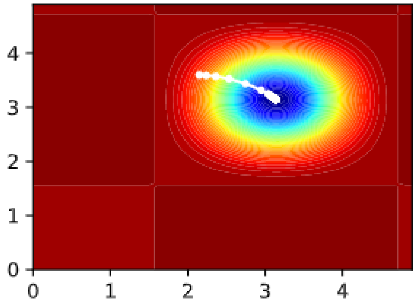
My Optimizer (biggest advantage – no worry about any hyperparameter – at least for now)

```
&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
```



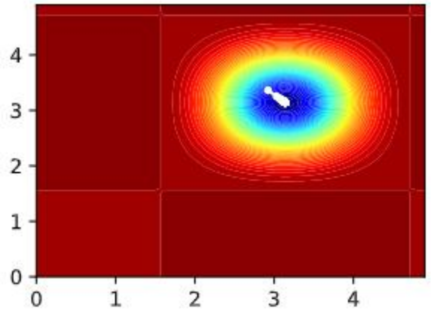
2400 iterations
(stuck in start)

```
&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

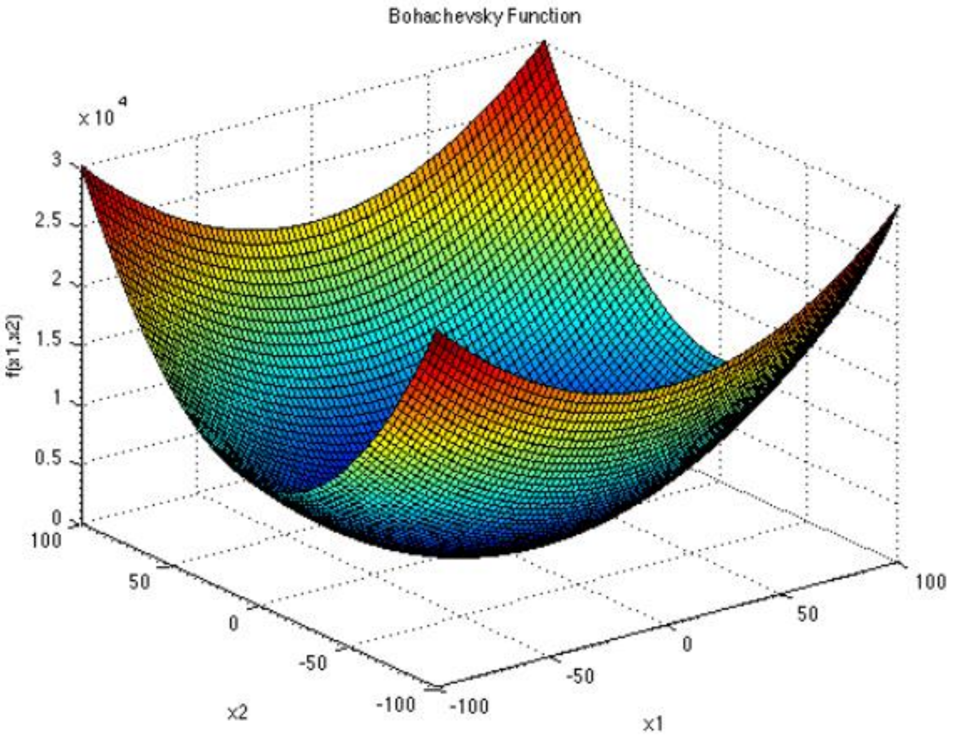
```
&gt;9997 f([3.14023181 3.1429535 ]) = -0.99999
&gt;9998 f([3.14023187 3.14295343]) = -0.99999
&gt;9999 f([3.14023194 3.14295337]) = -0.99999
```



10 iterations

Starting points randomly initialized

Bohachevsky functions



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

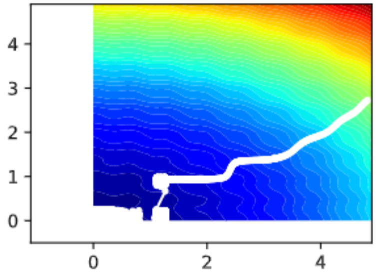
Global Minimum:

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

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

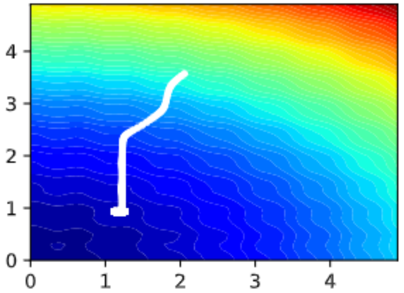
Adadelta (using hyperparameter tuning)

```
&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
```



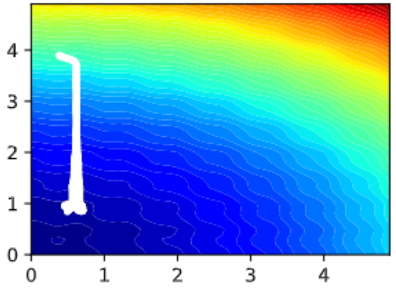
9999 iterations - diverging

```
&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

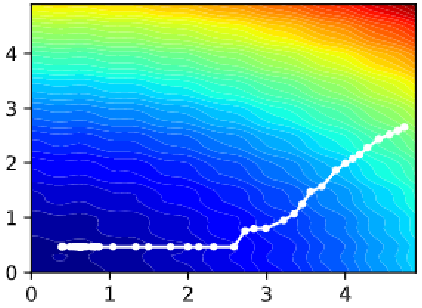
```
&gt;9997 f([0.66731963 0.8986926 ]) = 2.34328
&gt;9998 f([0.52969753 0.96759942]) = 2.40290
&gt;9999 f([0.69597394 0.86589144]) = 2.44097
```



9999 iterations

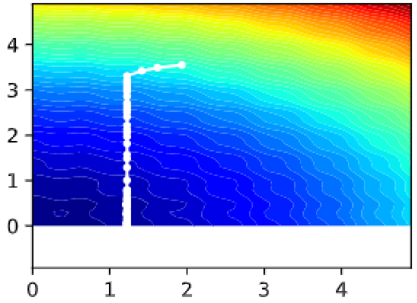
My Optimizer (biggest advantage – no worry about learning rate – at least for now)

```
&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
```



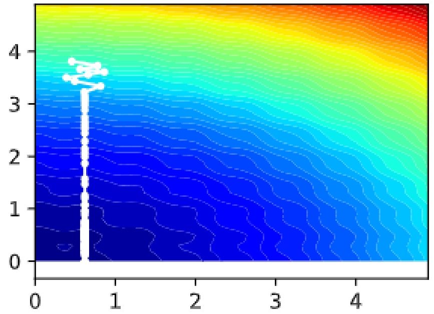
9999 iterations

```
&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

Similarly, we test for 28 more such test cases.

First observation: Convergence is quicker and often closer to global minima.

TODO: Score the optimizers on these parameters by creating a metric

Generalizing the optimizer

Proposing a new update rules, of the family,

$$\Delta\theta_i = -\frac{|\nabla f_i|^h}{\left|1 + \nabla f_i^{\,2}\right|^h} . \nabla f_i$$

This hyperparameter h has theoretical significance

Proving the convergence

Theorem:

A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is convex and differentiable, and that its gradient is Lipschitz continuous with constant $L < 2^{\mathrm{h}+1}$, i.e. we have that $\left|\left|\nabla f_x - \nabla f_y\right|\right|_2 \leq L\left|\left|x\ - y\right|\right|_2$ for any x, y and some $h \epsilon \mathbb{R}$. Then, if we run gradient descent for k iterations with update rule as $\Delta\theta_i = -\ \frac{|\nabla f_i|^h}{\left|1+\nabla f_i^2\right|^h}.\nabla f_i$, it will always lead us to a solution which satisfies

Assumption

∇f is Lipschitz continuous with constant L, i.e. $\left|\left|\nabla f_x - \nabla f_y\right|\right|_2 \leq L\left|\left|x - y\right|\right|_2$

This intuitively means that the gradient does not abruptly change, it is in a sense bounded by a real number L. Every function that has bounded first derivatives is Lipschitz continuous.

Intuitively, a Lipschitz continuous function is limited in how fast it can change.

Source: https://en.wikipedia.org/wiki/Lipschitz_continuity

Proof

For the loss function f, we can expand it around some x using Taylor's series:

$$f(y) \leq f(x) + \nabla f(x)^T (y-x) + \frac{1}{2} \nabla^2 f(x) \|y-x\|_2^2$$

$$\leq f(x) + \nabla f(x)^T (y-x) + \frac{1}{2} L \|y-x\|_2^2$$

Lipschitz constant

For gradient descent

$$y = x - \frac{(\|\nabla f\|)^h}{(1 + (\|\nabla f\|)^2)^h} \nabla f = x^+$$

Let $t = \frac{(\nabla f)^h}{(1 + (\nabla f)^2)^h}$

$$f(x^+) \leq f(x) + \nabla f_x^T (x^+ - x) + \frac{1}{2} L \|x^+ - x\|_2^2$$

$$\leq f(x) - \nabla f_x^T t \nabla f_x + \frac{1}{2} L \|t \nabla f_x\|_2^2$$

$$\leq f(x) - t \|\nabla f_x\|_2^2 + \frac{1}{2} L t^2 \|\nabla f_x\|_2^2$$

$$\leq f(x) - \left(1 - \frac{1}{2} L t\right) t \|\nabla f_x\|_2^2$$

Now, $t \|\nabla f(x)\|_2^2$ will always be +ve.

as all terms are positive.

$$t = \frac{|\nabla f|^h}{(1 + |\nabla f|^2)^h} \rightarrow \text{always +ve}$$

For GD to converge,

$\left(1 - \frac{1}{2}Lt\right)$ should be positive always.

$$1 - \frac{1}{2}Lt > 0$$

$$\frac{Lt}{2} < 1$$

$$L < \frac{2}{t}$$

$$\frac{Lt}{2} < 1$$

$$L < \frac{2}{t}$$

t can be thought as a function of type $\frac{x^n}{(1+x^2)^n}$

$$\frac{dt}{dx} = \frac{(1+x^2)^n \cdot n x^{n-1} - x^n \cdot n(1+x^2)^{n-1} \cdot 2x}{(1+x^2)^{2n}} = 0$$

Thus, this would be max at $x = 1$.

Maximum value of $t = \frac{1}{2^n}$

as $L < \frac{2}{t}$

$\therefore L < 2^{h+1}$

$\therefore$ GD would converge for all functions having $L < 2^{h+1}$ if the update rule is used with hyperparameter h.

In practice, $h = 2$ should be more than enough.

Hence, we can start iterating from $h = 2$ and go on if worst case to get a very good guaranteed convergence.

Further Work

- 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.