Numerical Optimization for ML&DL (NOFML&DL)

Information Technology Institute

Session 3

Feature Scaling (review and discussion (see previous lecture))

Batch Gradient Descent (Vanilla GD) Problems.

GD Variants (stochastic GD and mini-batch GD)**.**

GD Challenges (learning rate, local minima, vanishing and exploding gradient)**.**

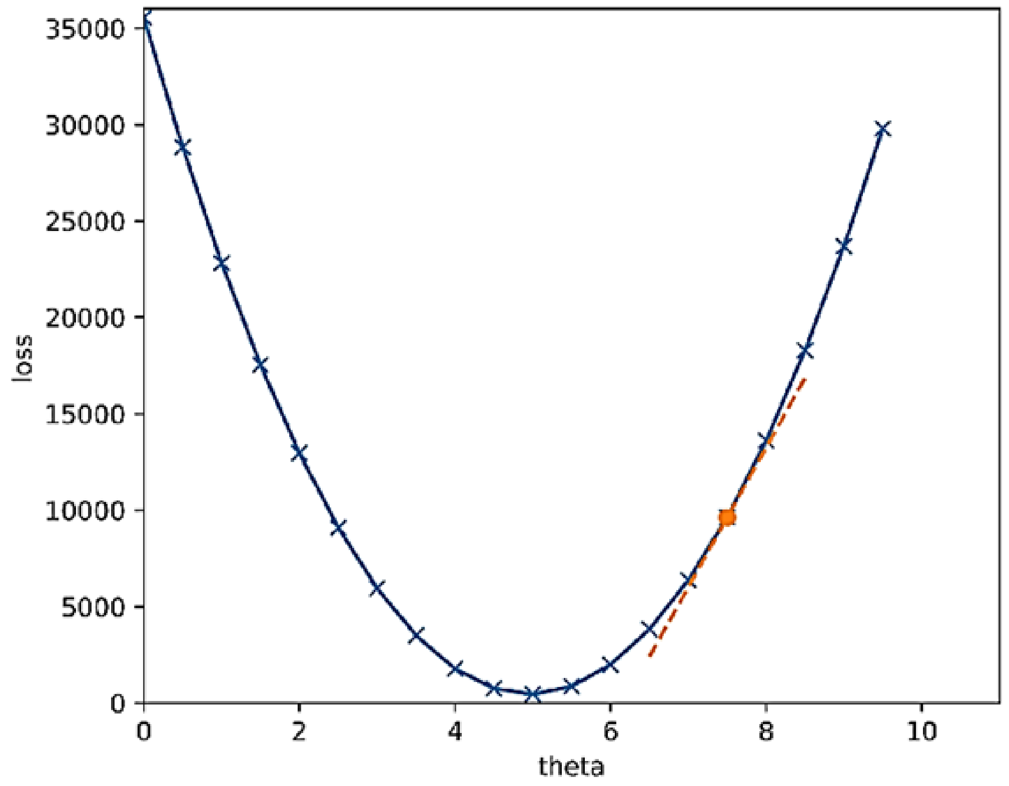
Momentum-based GD.

Nesterov accelerated GD (NAG.)

Batch/Vanilla GD Whole Picture

- Batch (vanilla) gradient descent, computes the gradient of the cost function w.r.t. the parameters **θ** for the entire training dataset:

$$\Theta = \Theta - \alpha \boldsymbol{\nabla}_{\Theta} \boldsymbol{J}(\Theta)$$



Gradient descent algorithm

```
repeat until convergence {
    $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$
        (for $j = 1$ and $j = 0$)
}
```

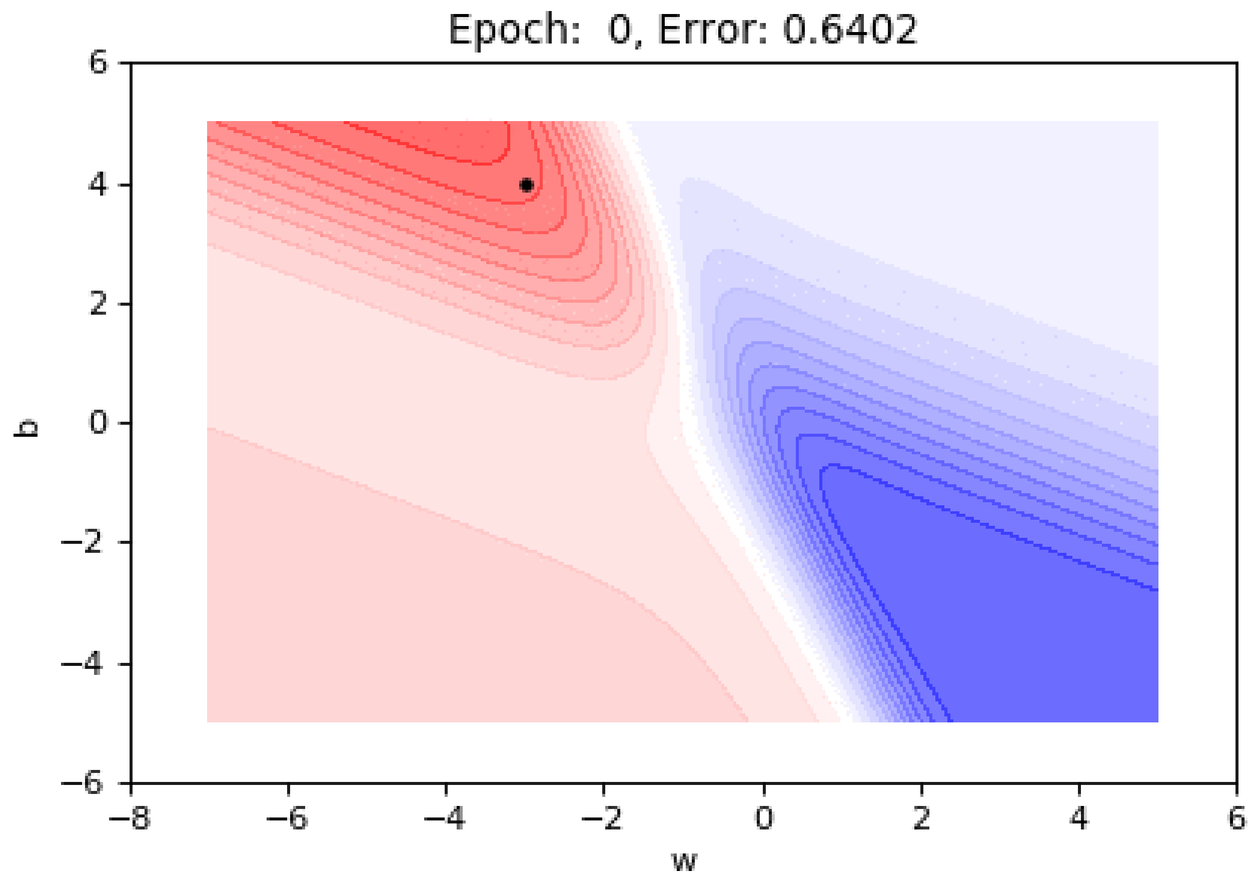
Linear Regression Model

$$h_\theta(x) = \theta_0 + \theta_1 x$$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Batch/Vanilla GD Whole Picture

- **GD and Backpropagation** algorithms are used to train artificial neural networks **(ANN)**. i.e., update **weights** and **biases**.
- Those are key algorithms in **Deep learning**.



$$w = w - \eta \nabla w \, ; \; b = b - \eta \nabla b$$

where $\nabla w = \frac{\partial L(w)}{\partial w}$ and $\nabla b = \frac{\partial L(b)}{\partial b}$

Batch GD Problems

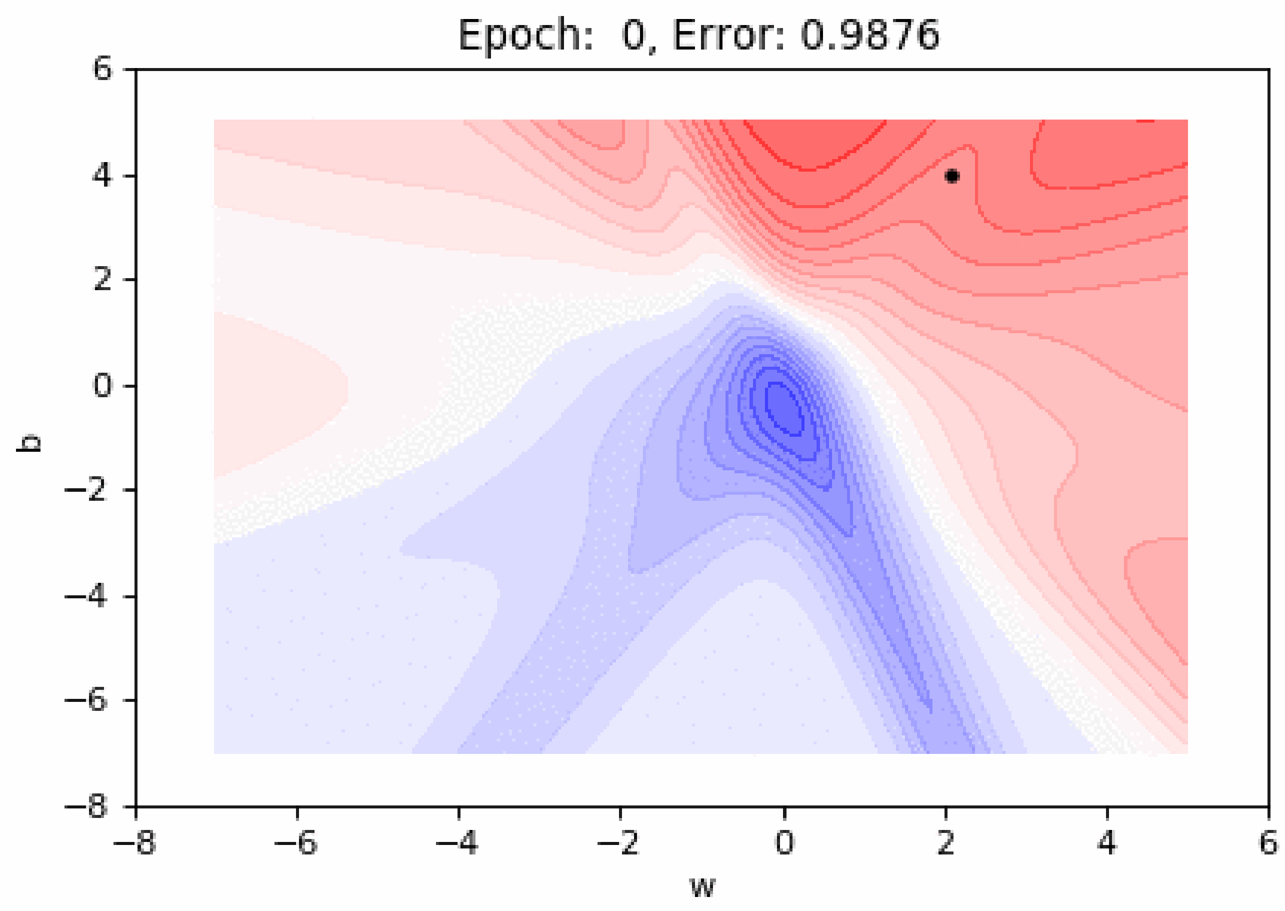
- ***Standard Gradient descent*** updates the *parameters* only after each epoch.
- I.e., after calculating the *derivatives* for all the observations, it updates the *parameters*.
- This may lead to the following ***problems:***
  - It can be very slow for very large datasets.
  - Large number of **epochs** is required to have a substantial number of updates.
  - For large datasets, the vectorization of data doesn’t fit into **memory**.
  - For non-convex surfaces, it may only find the **local minima.**
- **Different variants of GD can address these challenges.**

Stochastic GD (SGD)

- Updates the parameters for each observation which leads to a greater number of updates.

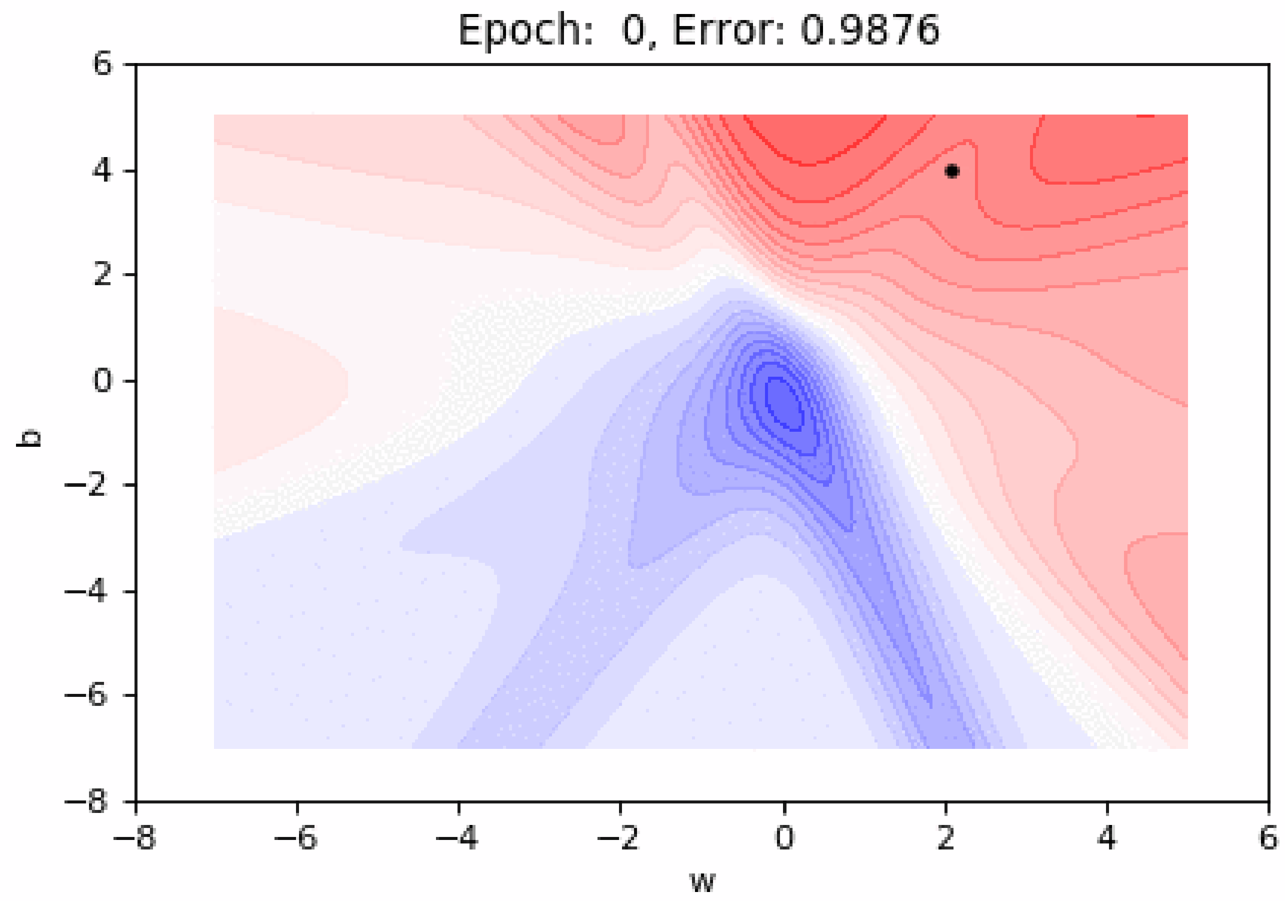
$$\Theta^{\text{new}} = \Theta^{\text{old}} - \alpha \boldsymbol{\nabla}_{\Theta} \boldsymbol{J}\left(\Theta^{\text{old}}\right)$$

$$J(\Theta) = \frac{1}{2}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$



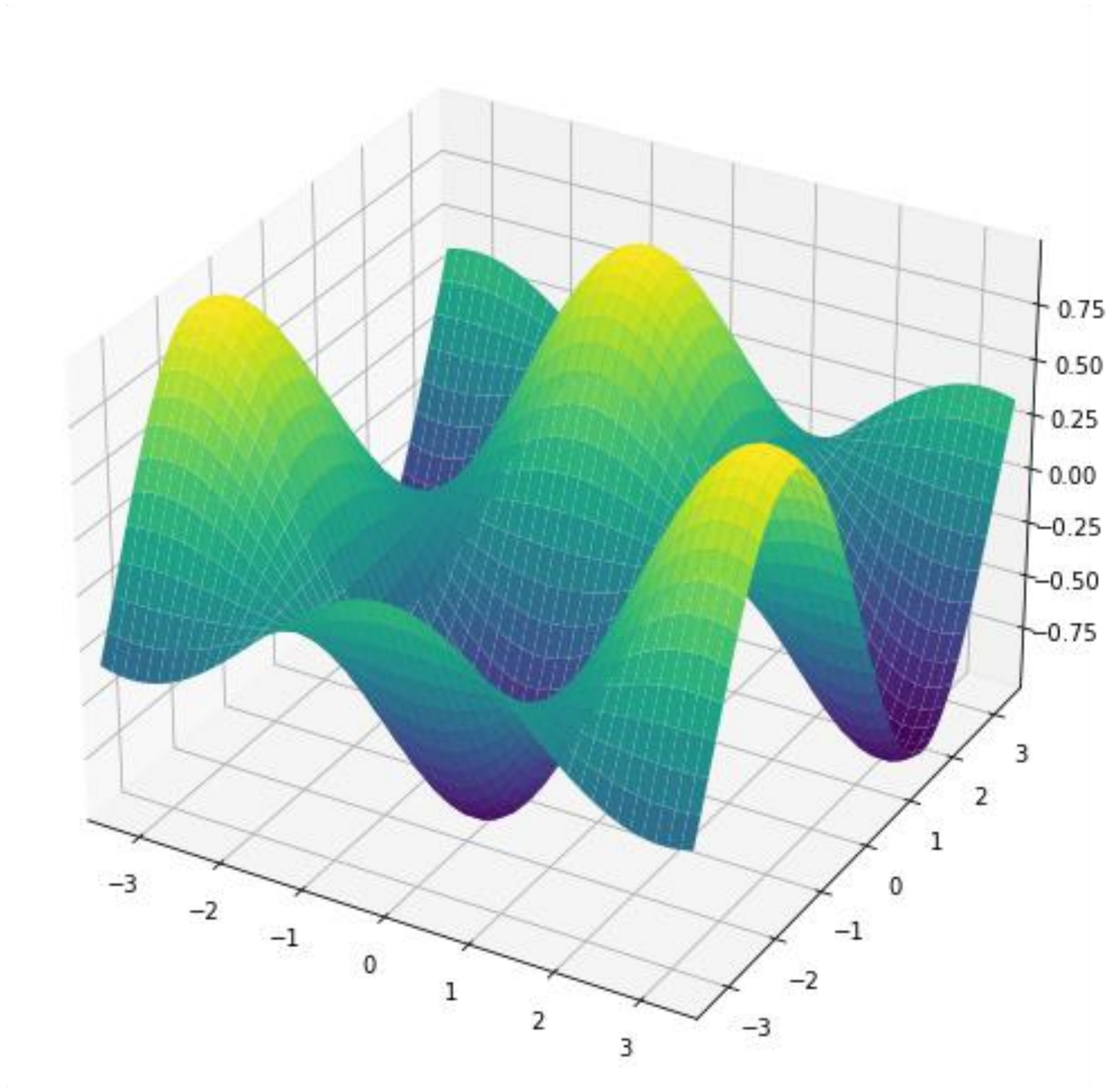
Disadvantages of SGD

- Due to frequent fluctuations, it will keep overshooting near to the desired minimum.
- Adds noise to the learning process i.e., the variance becomes large since we only use 1 example for each learning step.
- We can’t utilize vectorization over 1 example.



Another Advantage of SGD

- **SGD** works better than batch gradient descent for error manifolds that have lots of local maxima/minima.
- In this case, the somewhat noisier gradient calculated using the reduced number of samples tends to jerk the model out of local minima into a region that hopefully is more optimal.

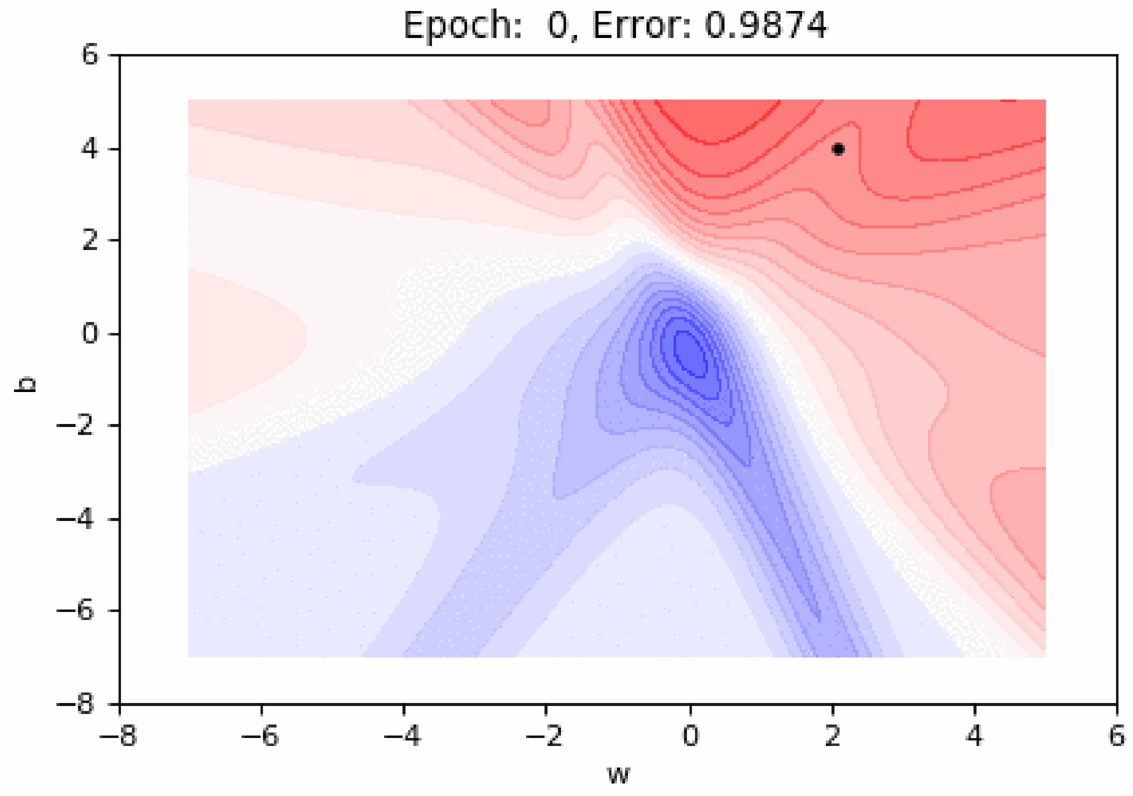


Mini-batch GD

- Instead of going over all examples, Mini-batch Gradient Descent sums up over lower number of examples.
- Therefore, learning happens on each mini-batch of **b** examples:

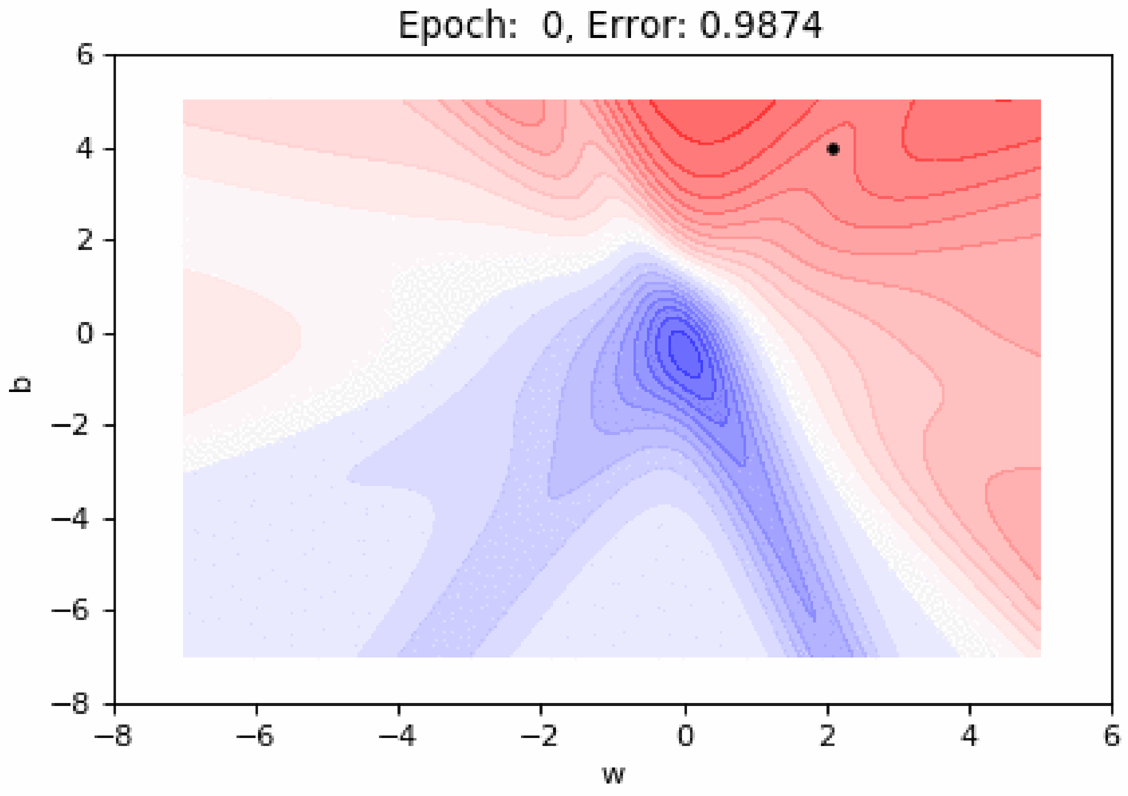
$$\Theta^{\text{new}} = \Theta^{\text{old}} - \alpha \boldsymbol{\nabla}_{\Theta} \boldsymbol{J}\left(\Theta^{\text{old}}\right)$$

$$\boldsymbol{J}(\Theta) = \frac{1}{2\text{b}} \sum_{\text{i}\in\text{b}} \left(\boldsymbol{h}_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(\boldsymbol{i})}\right) - \boldsymbol{y}^{(\boldsymbol{i})}\right)^2$$



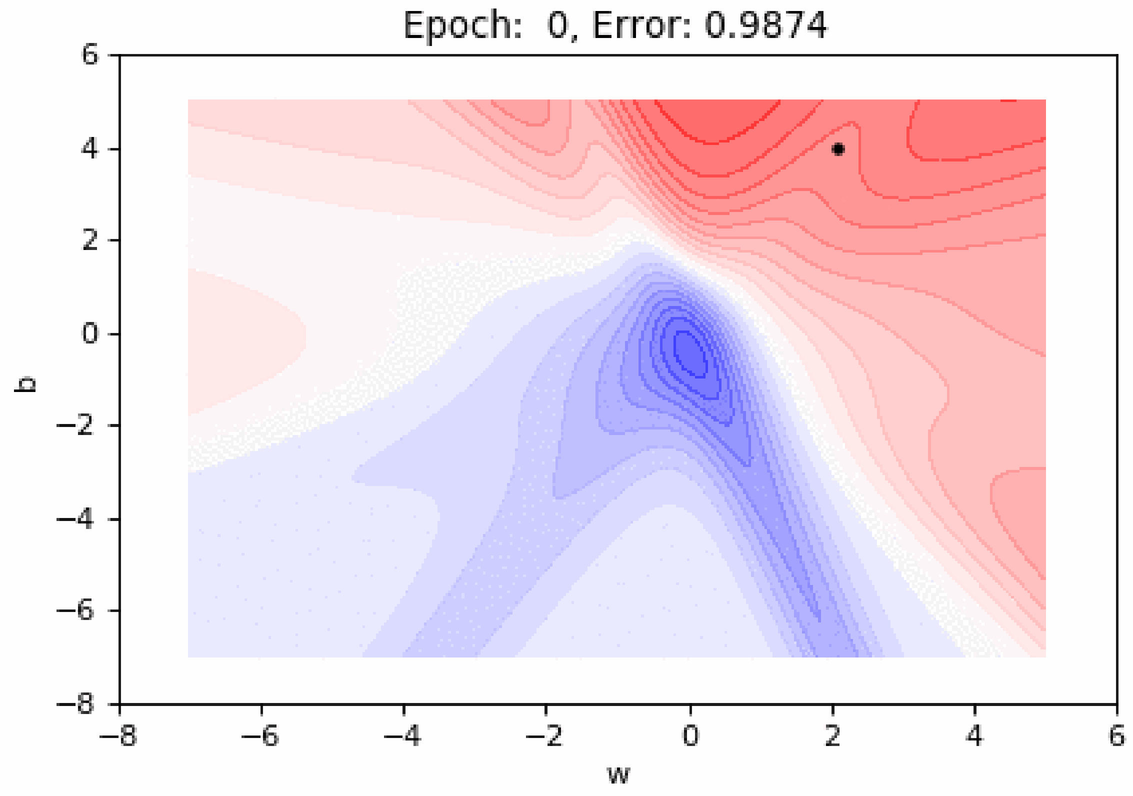
Advantages of Mini-batch GD

- Updates are less noisy compared to SGD which leads to better convergence.
- A high number of updates in a single epoch compared to GD so a smaller number of epochs are required for large datasets.
- Fits very well to the processor memory which makes computing faster.

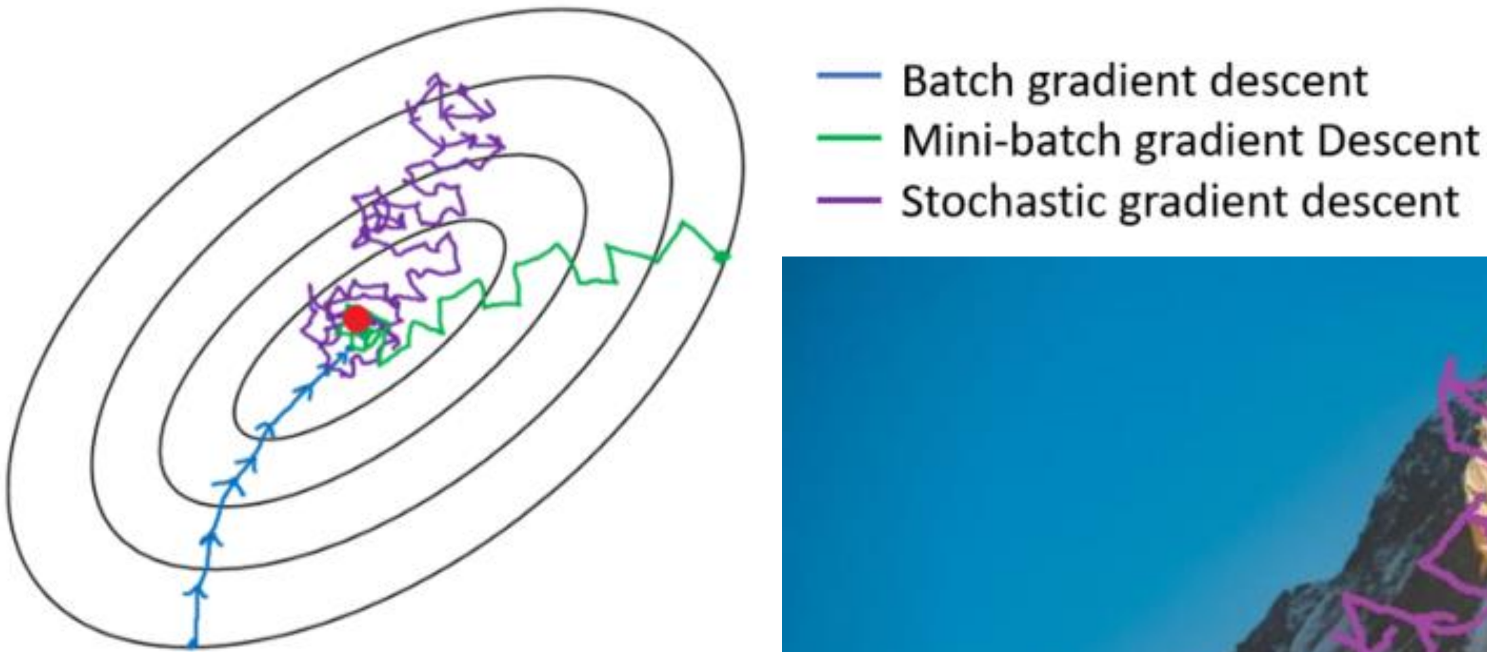


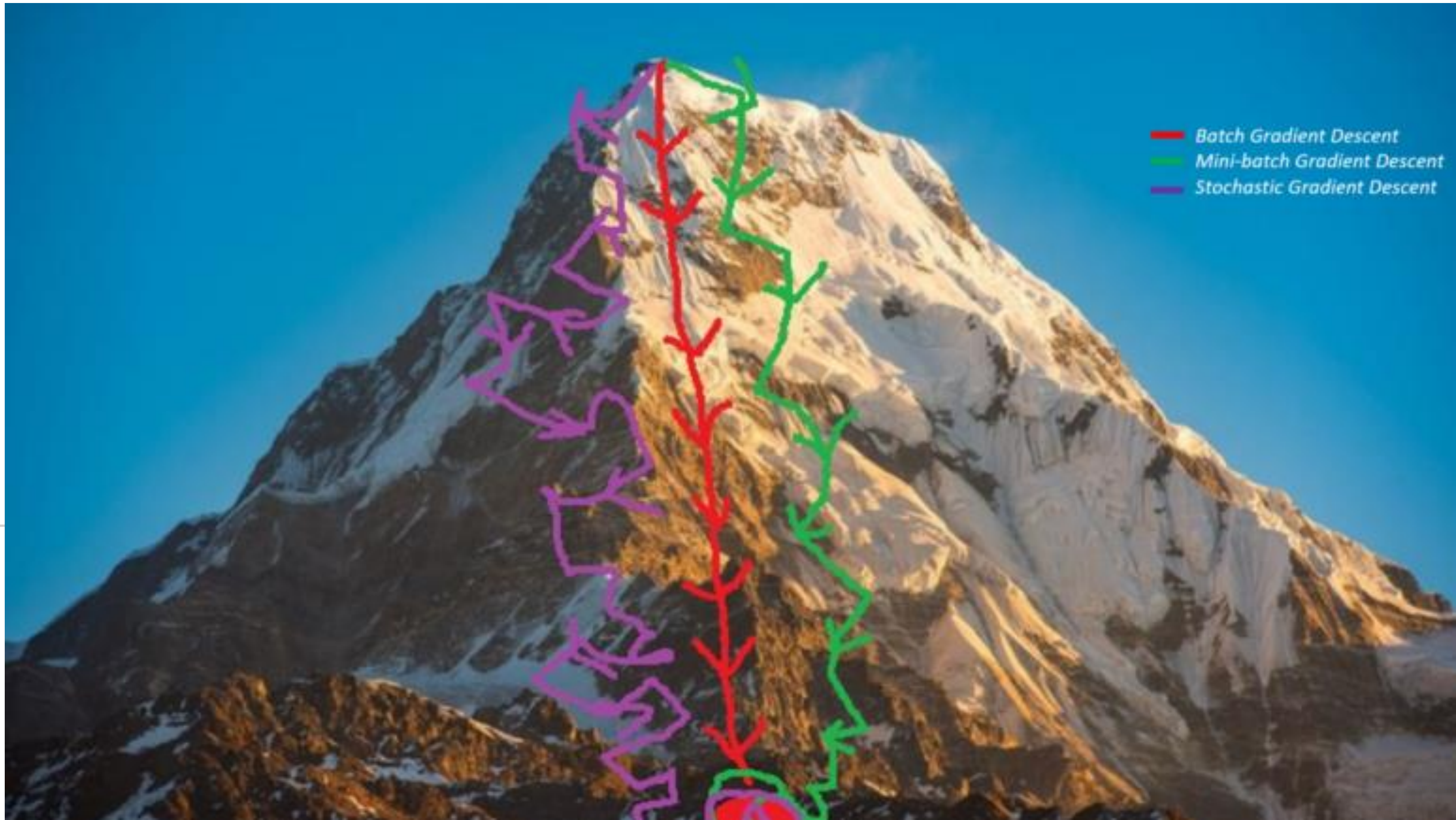
Mini Batch GD

- ***Note:*** The batch size is something we can tune. It is usually chosen as power of 2 such as 32, 64, 128, 256, 512, etc.
- ***Implementation Note: for both SGD and minibatch it is preferable to shuffle data before using it.***



GD Variants


Batch gradient descent
Mini-batch gradient Descent
Stochastic gradient descent


Batch Gradient Descent
Mini-batch Gradient Descent
Stochastic Gradient Descent

GD Challenges

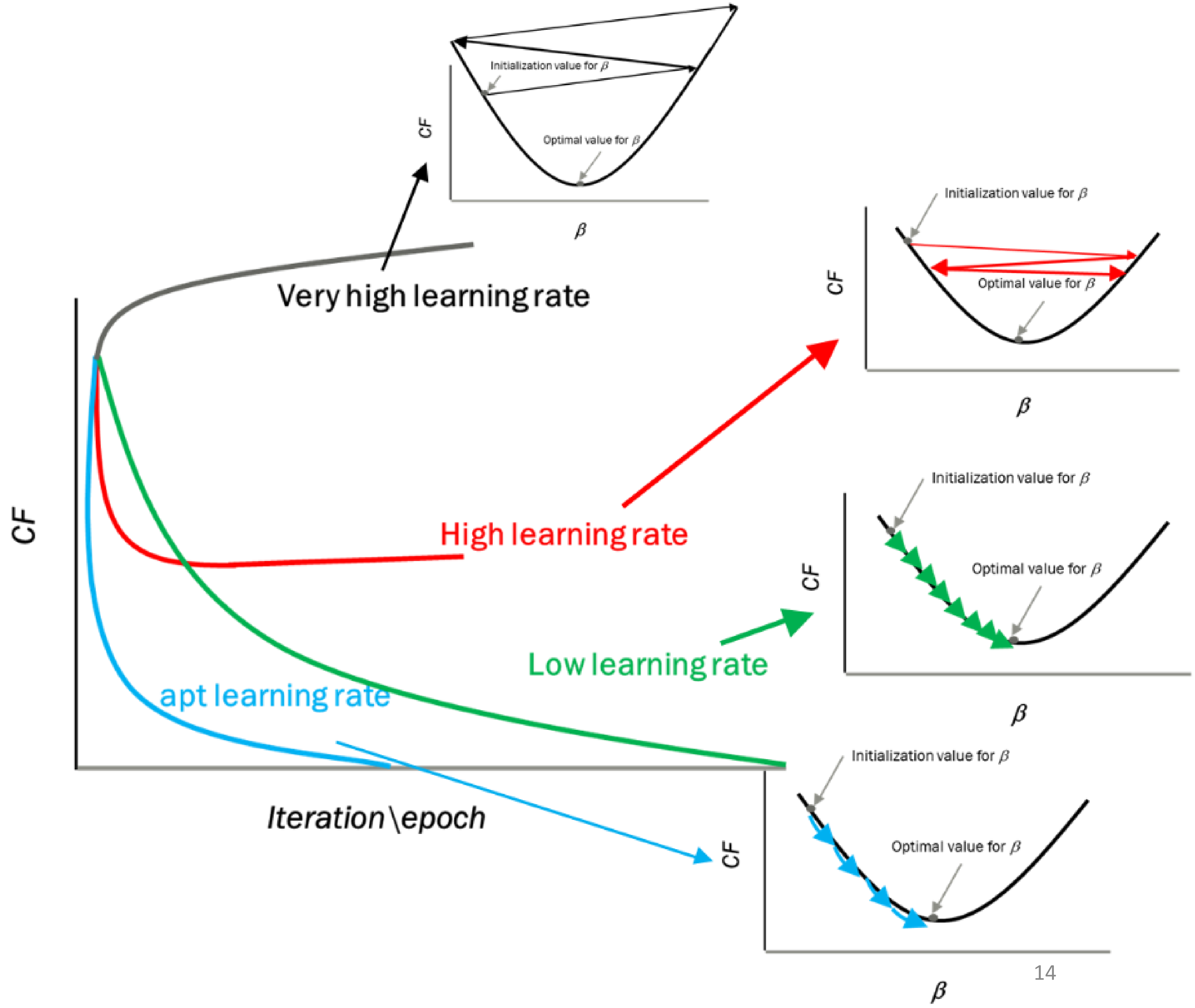
1. How to find a good value for the learning rate?

2. What to do in case of local minima?

3. How to solve the vanishing gradient problem?

1. Learning Rate

$$\theta_{t+1} = \theta_t - \alpha * gradient$$



Initialization value for β
Optimal value for β
CF
β
Very high learning rate
Initialization value for β
Optimal value for β
CF
β
High learning rate
Initialization value for β
Optimal value for β
CF
β
Low learning rate
apt learning rate
CF
Iteration\epoch
Initialization value for β
Optimal value for β
CF
β

How to find a good value for the learning rate?

Unfortunately, there is no magic bullet to find the perfect learning rate.

To find a good value, you have to test several values and pick the best.

Advices to choose the learning rate

- Plot cost function with epochs (iterations) and check if it is decreasing.

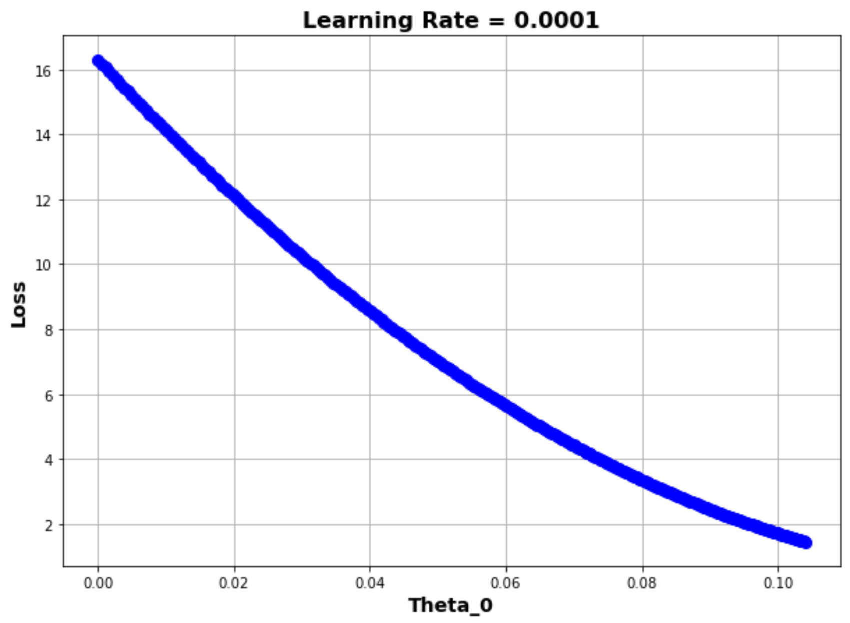
- Convergence check: $\boldsymbol{cost(i-1)}$ – $\boldsymbol{cost(i)} \ < \ \mathbf{0.001}$.

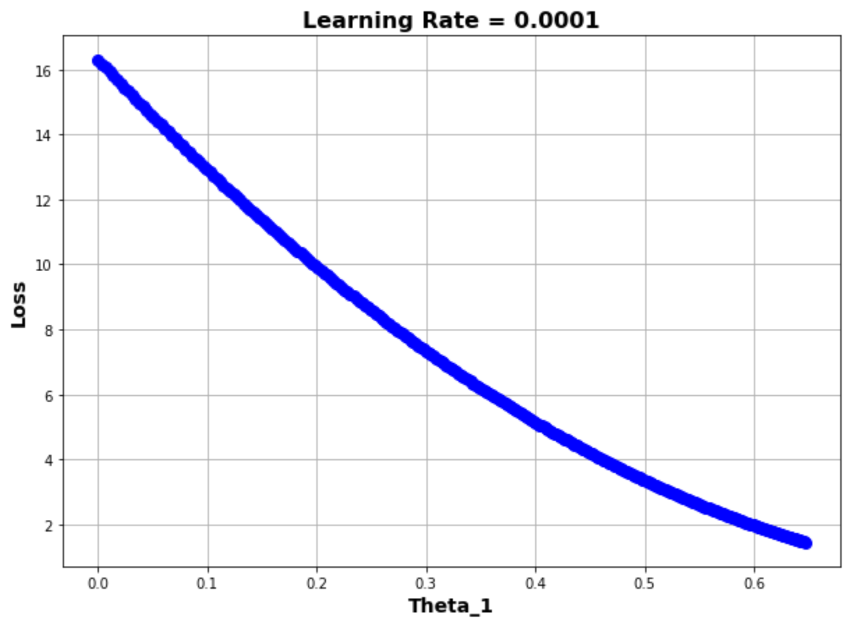
- Try range of **α** e.g., 0.001, 0.01,0.1,1 then **plot cost vs. epochs** and check for rapid and smooth conversion. Then you can select another **α** close the value in that range.
  e.g., if **0.001** is fine and **0.01** is bad you can try values in between such as **0.005**

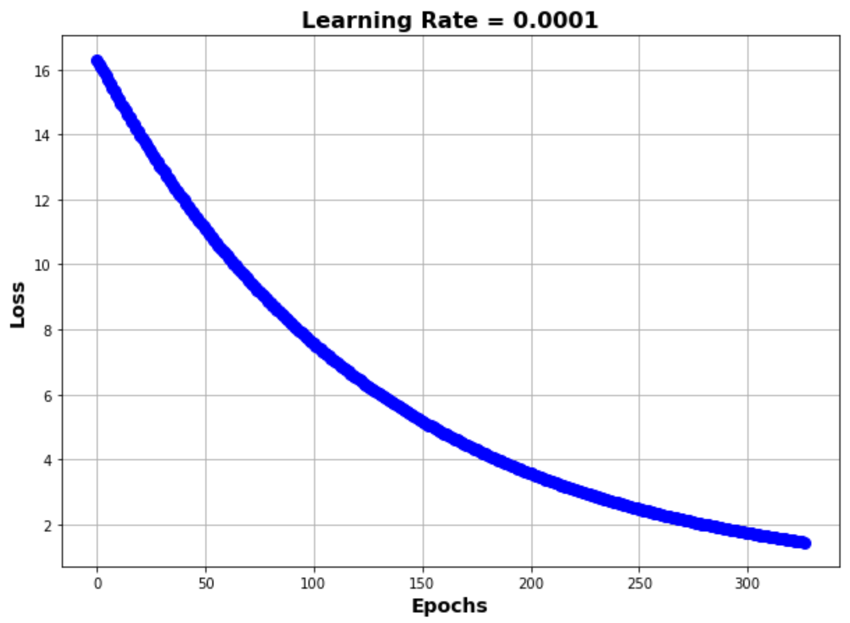
- **Plot parameters vs. cost.** Sometimes large learning rate implies many number of iterations due to parameter oscillation (cost function overshooting). Although, the decrease of cost function, overshooting increases number of iterations.

Plot parameters vs. cost

plot cost vs. epochs





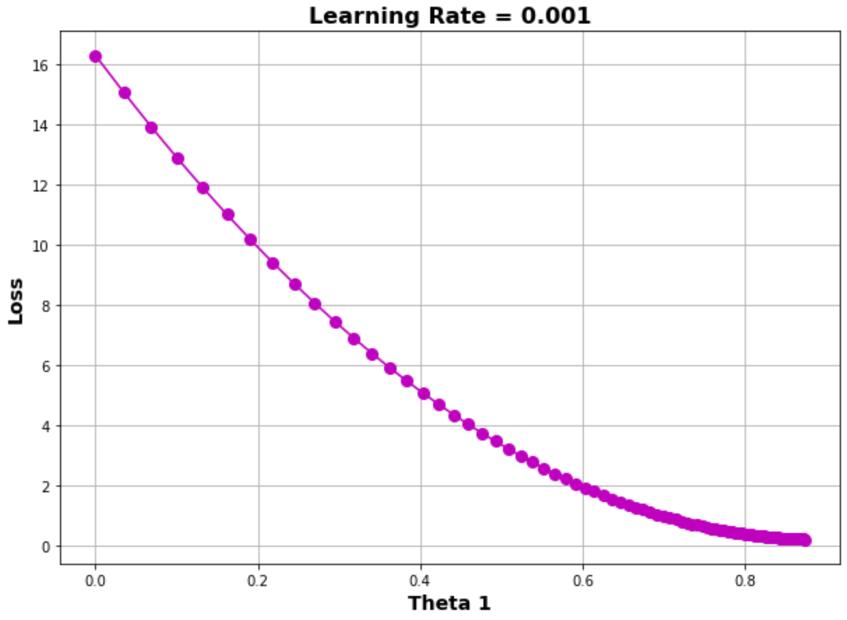


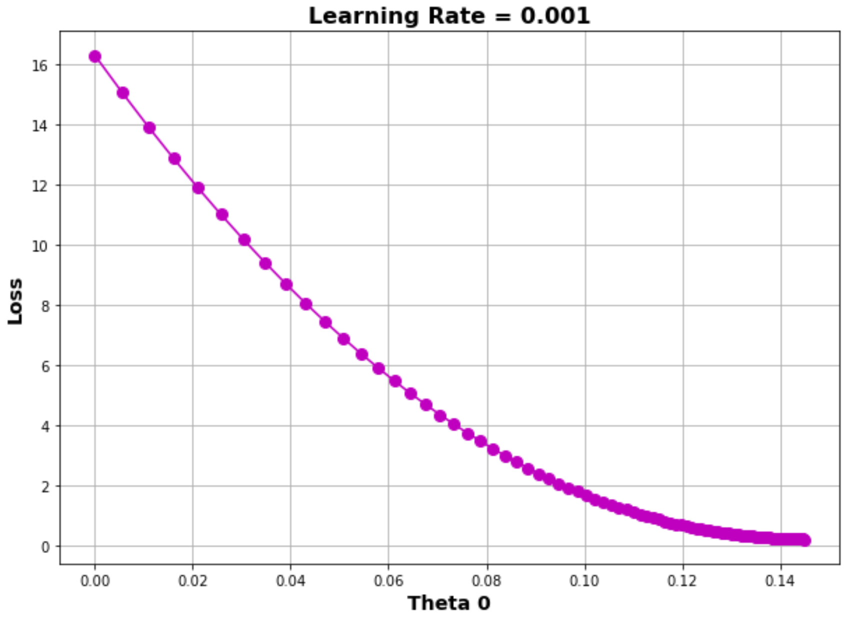
$$\alpha = 0.0001$$

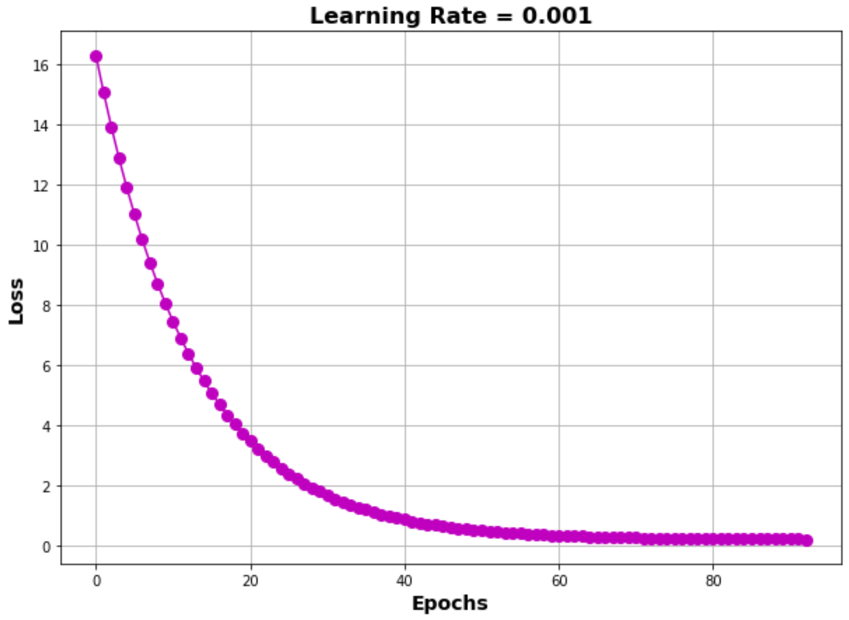
How to find a good value for the learning rate?

Plot parameters vs. cost

plot cost vs. epochs





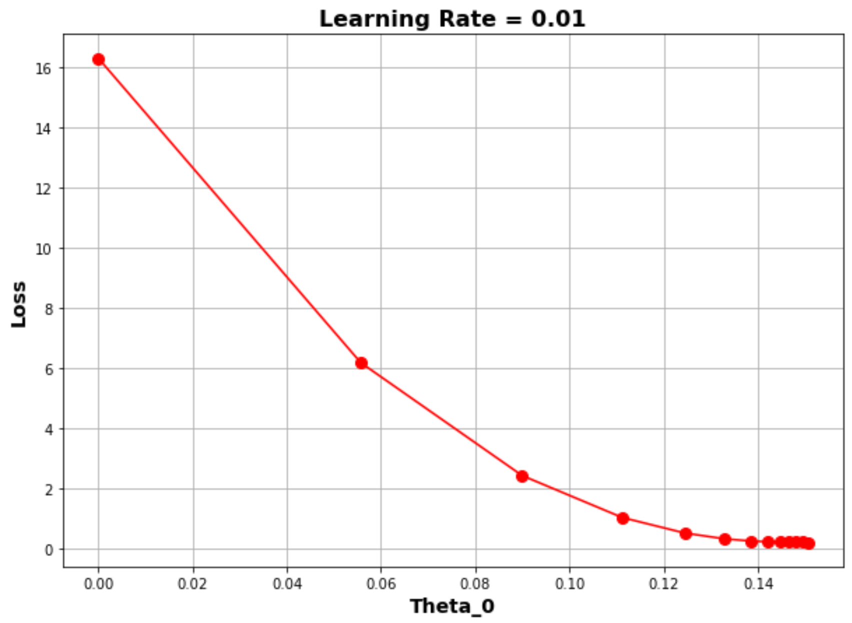


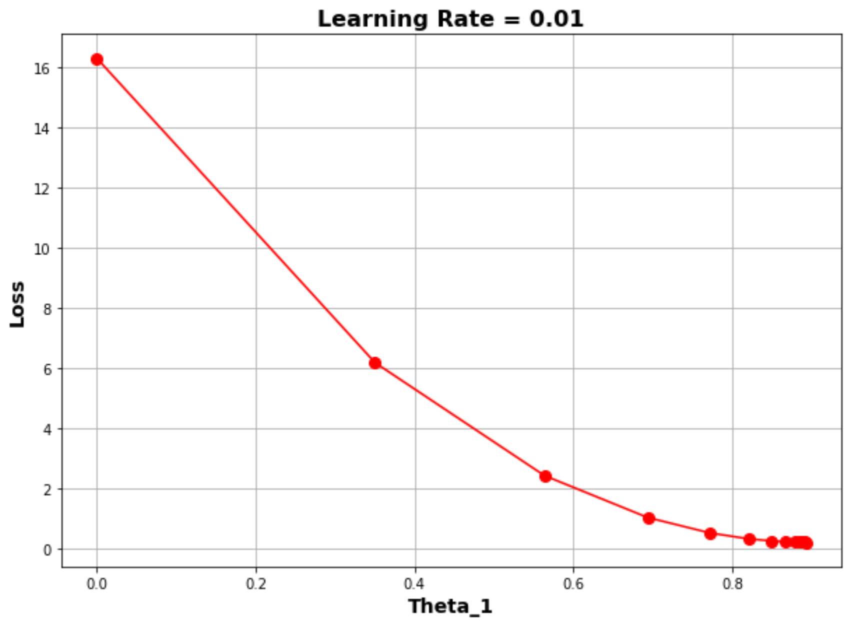
$\alpha = 0.001$

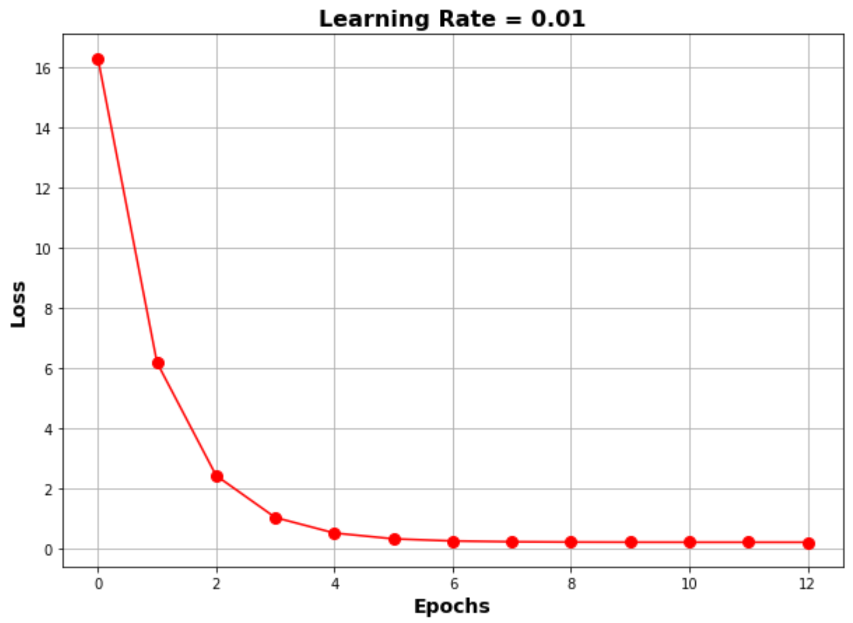
How to find a good value for the learning rate?

Plot parameters vs. cost

plot cost vs. epochs





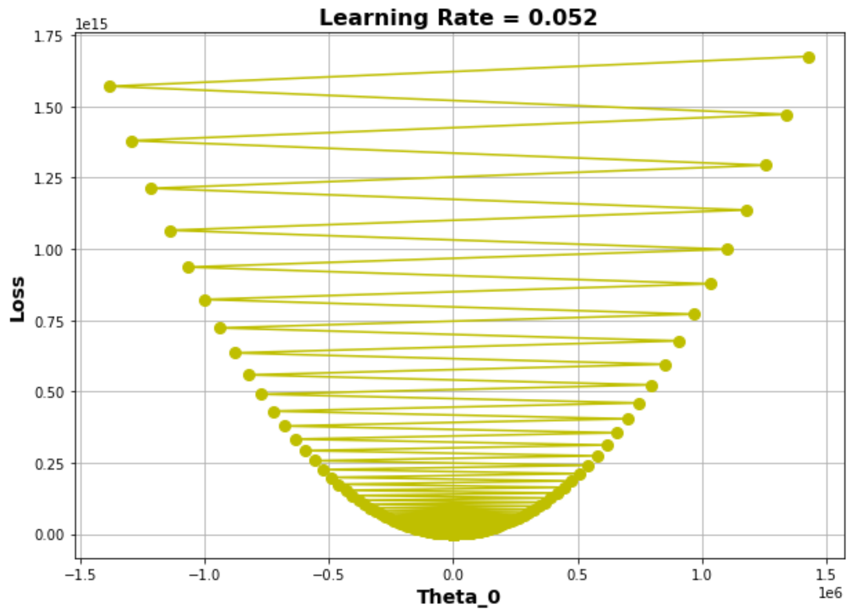


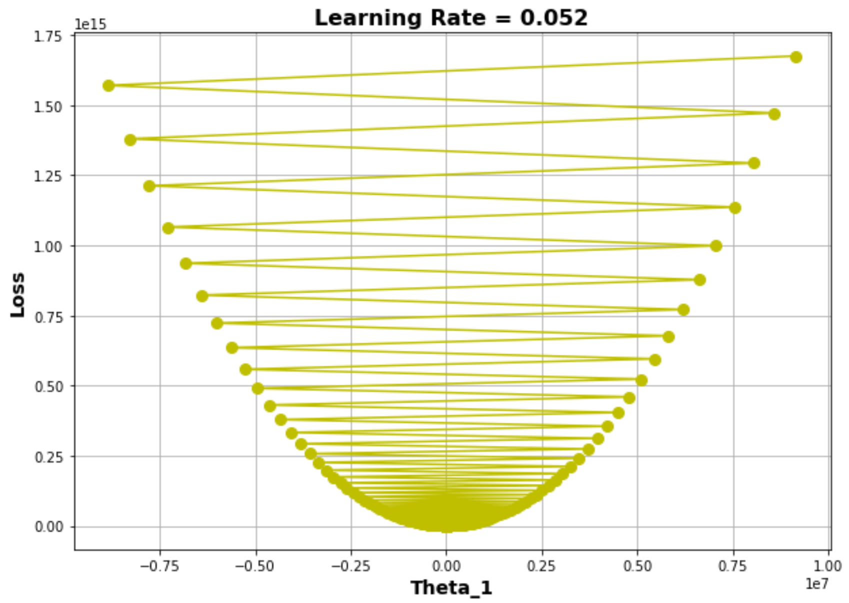
$\alpha = 0.01$

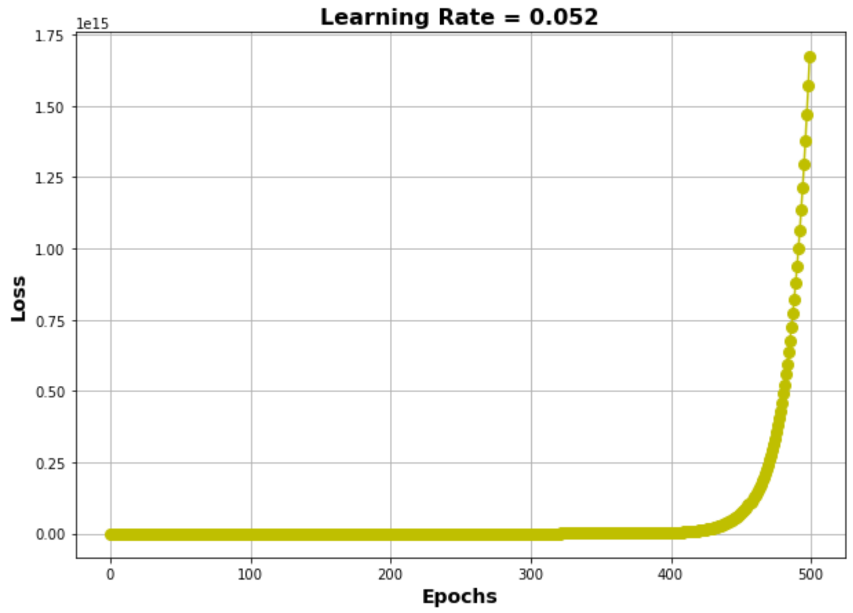
How to find a good value for the learning rate?

Plot parameters vs. cost

plot cost vs. epochs



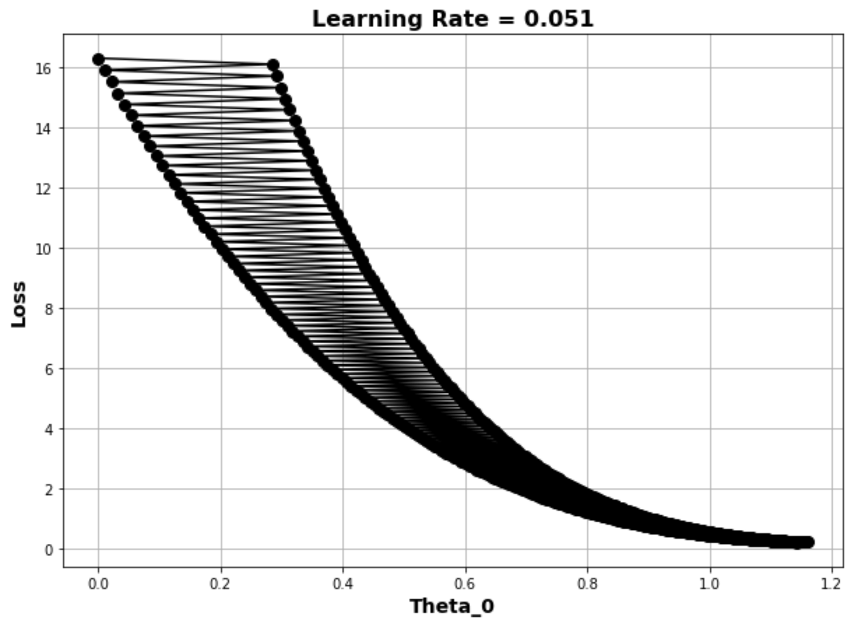


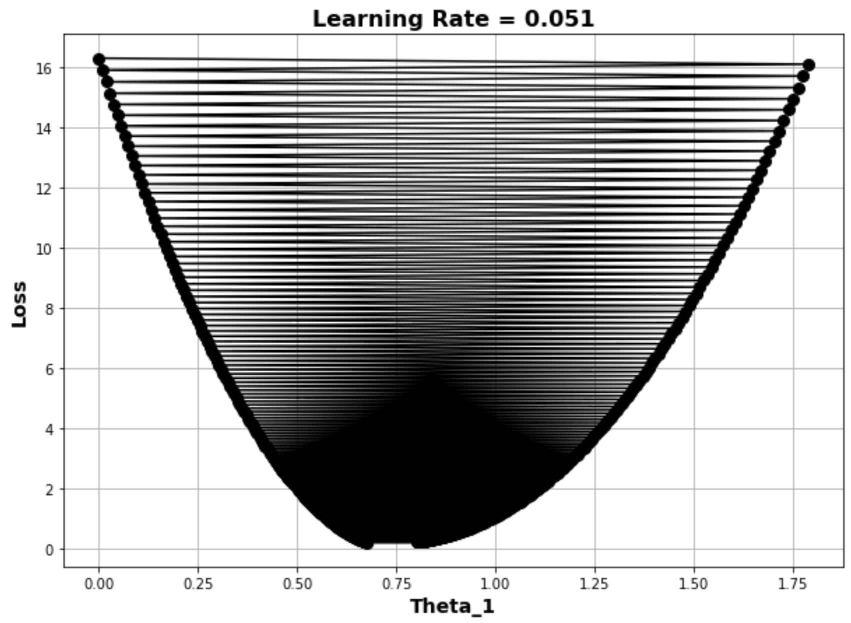


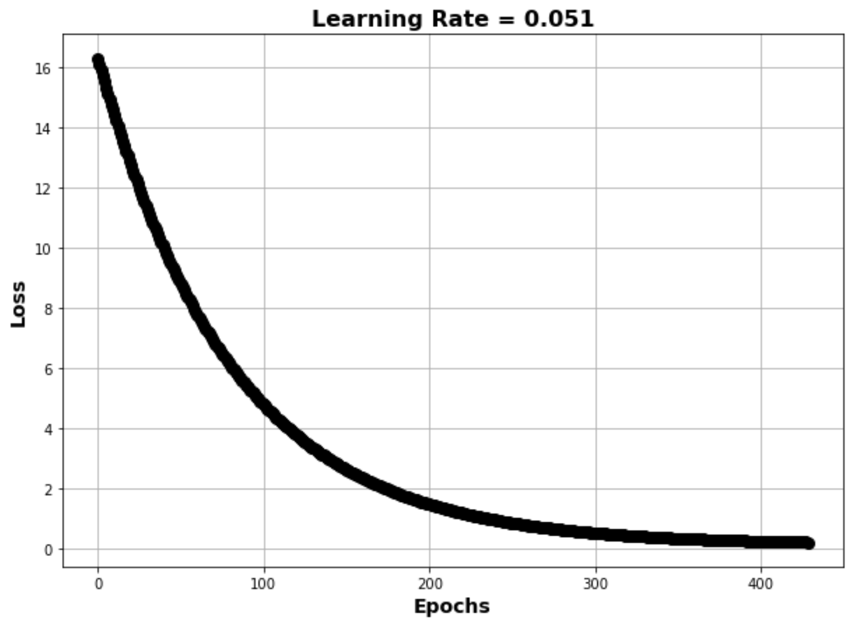
$\alpha = 0.052$

How to find a good value for the learning rate?

Plot parameters vs. cost **plot cost vs. epochs**







$\alpha = 0.051$

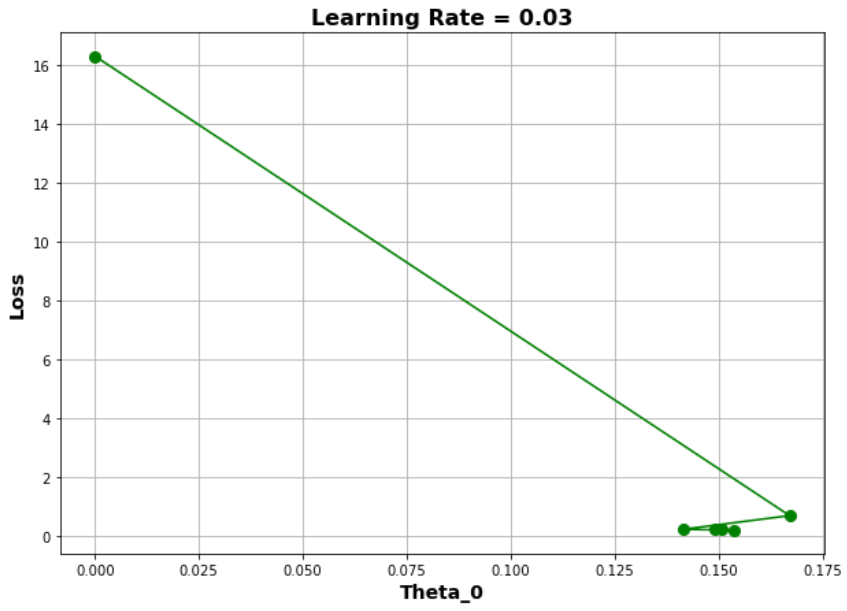
How to find a good value for the learning rate?

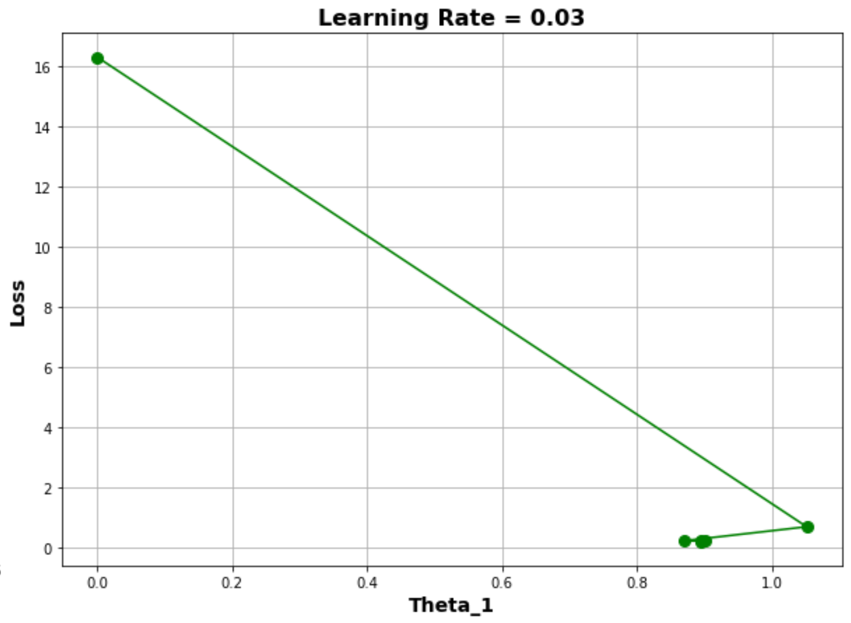
How to find a good value for the learning rate?

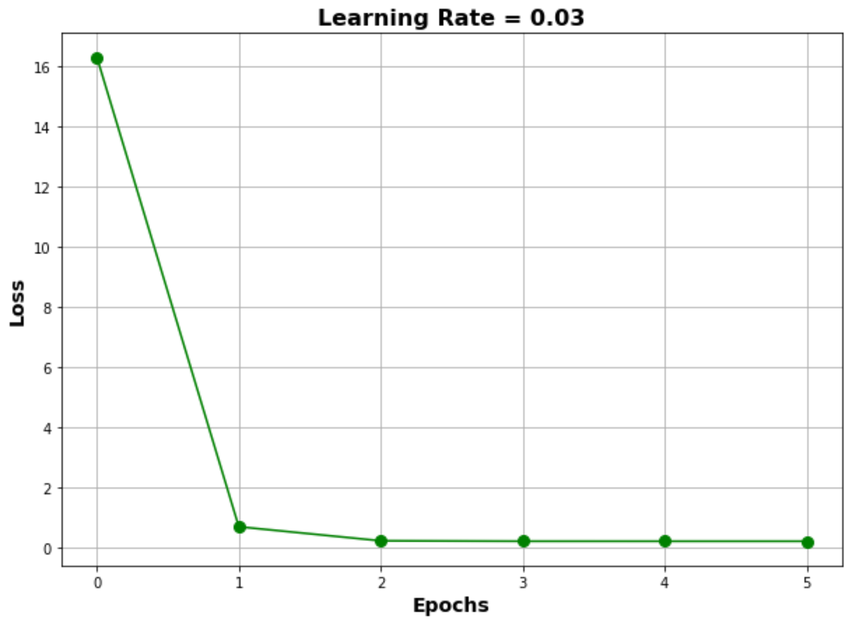
Can you guess now the suitable α?

Plot parameters vs. cost

plot cost vs. epochs





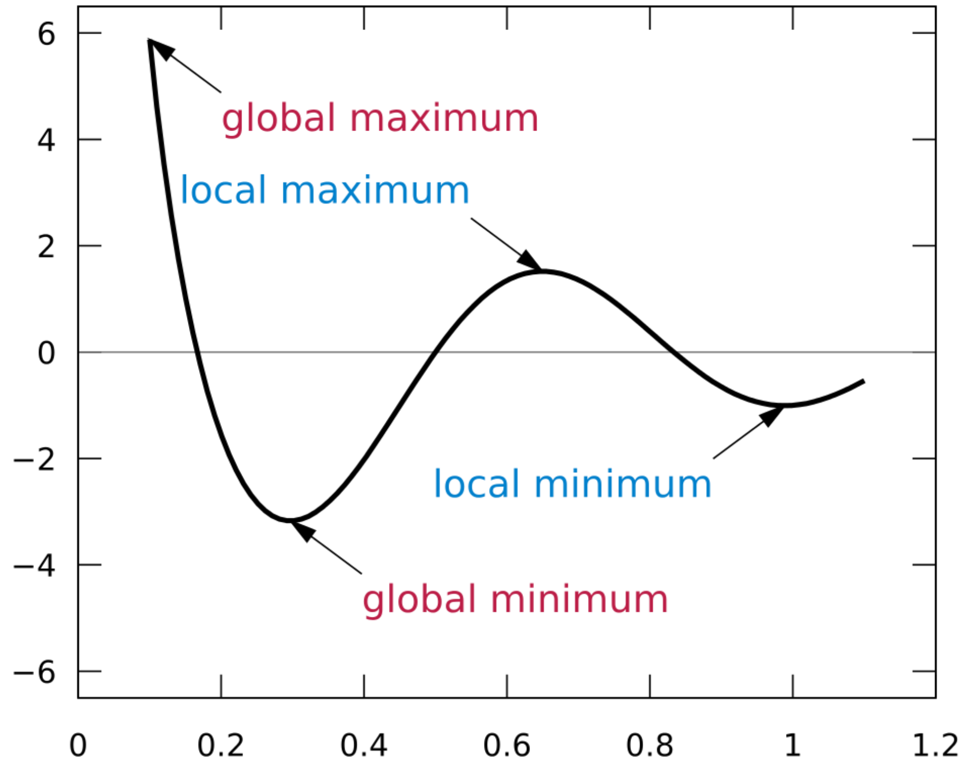


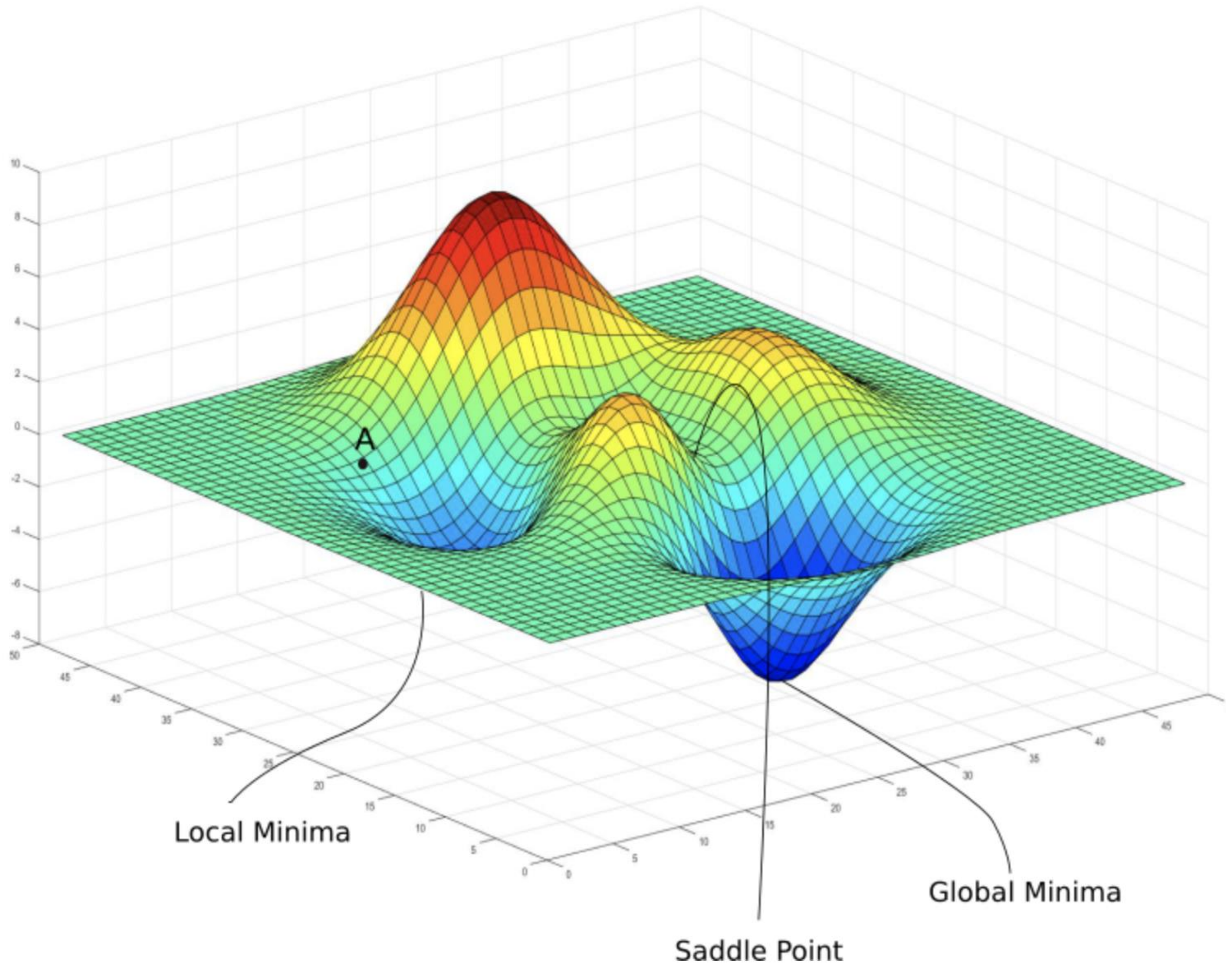
$\alpha = 0.03$

How to find a good value for the learning rate?

Of course, model accuracy need to be checked for these parameters (θ_0, θ_1)

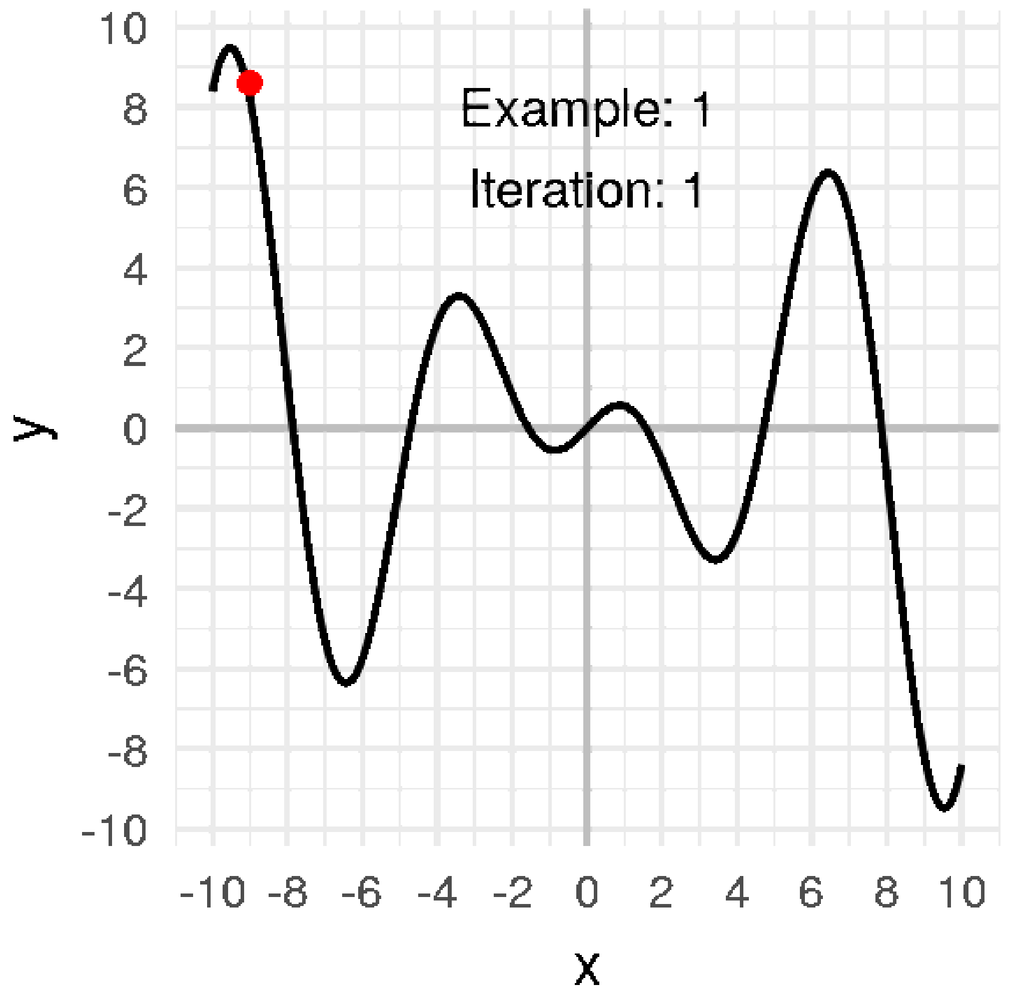
2. Local Minima





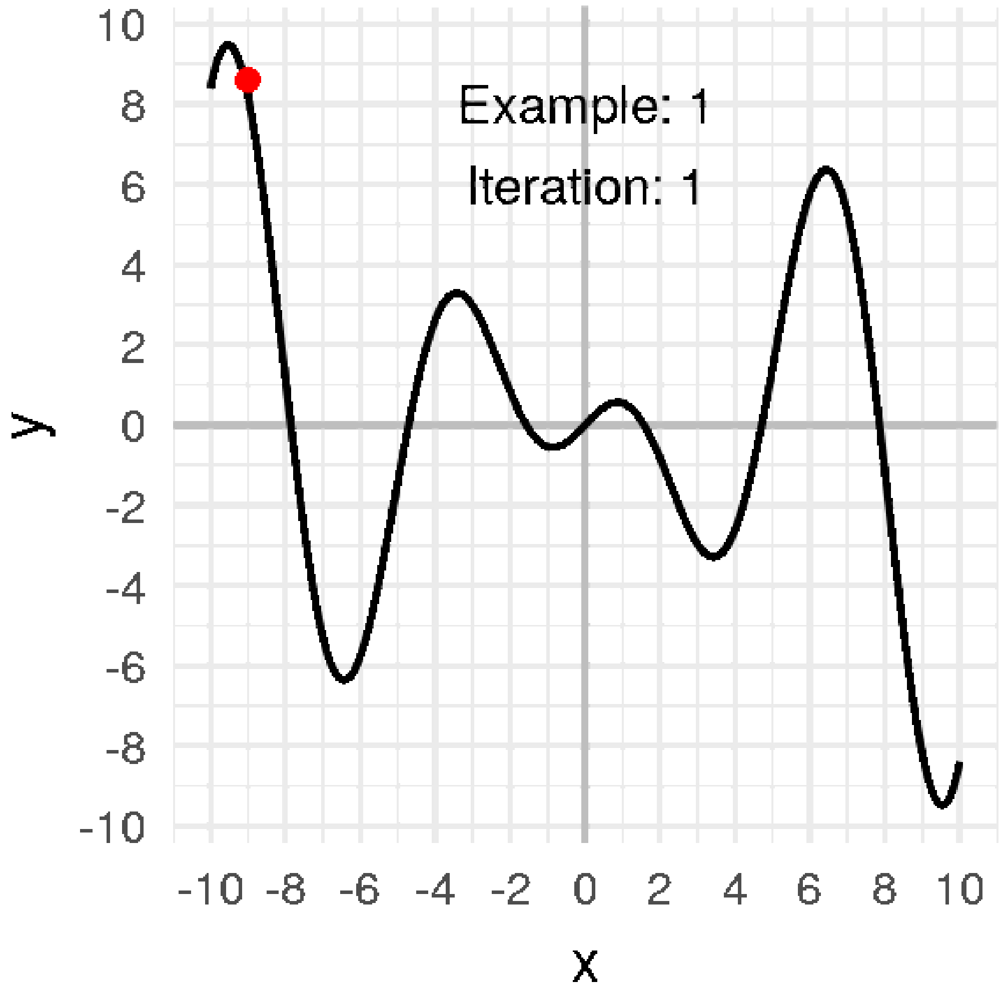
Local Minimum

- Notice the final convergence point depends a lot on the initial point.
- Sometimes it will find the global minimum. Other times not.
- To avoid this problem, the best way is to run the algorithms multiple times and keep the best minimum of all times. But, of course, it takes a long time to run.



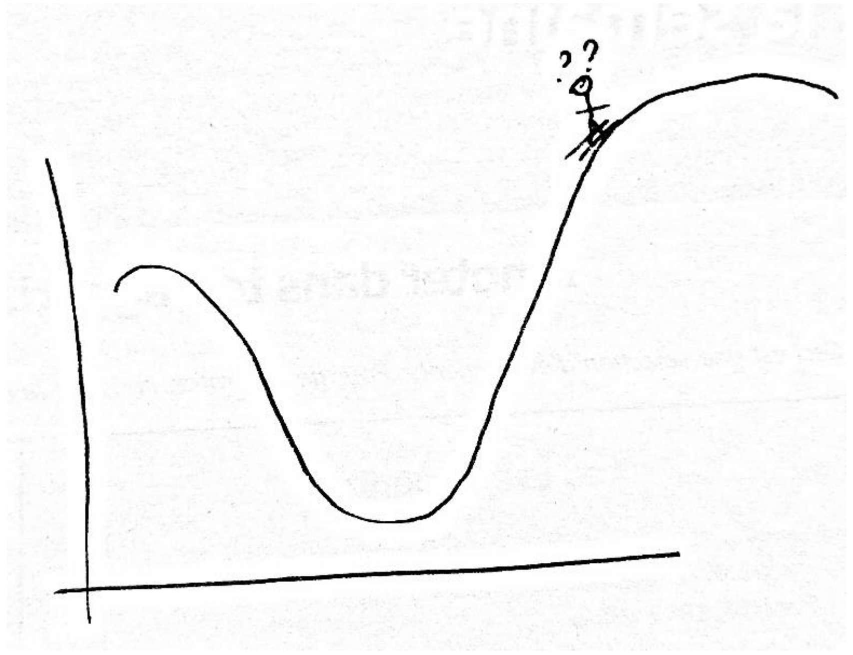
Local Minimum

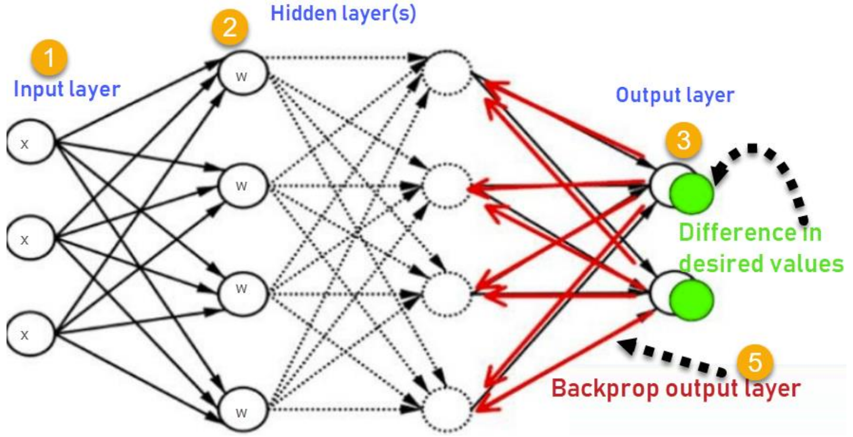
- Notice the final convergence point depends a lot on the initial point.
- Sometimes it will find the global minimum. Other times not.
- To avoid this problem, the best way is to run the algorithms multiple times and keep the best minimum of all times. But, of course, it takes a long time to run.



3. Vanishing Gradient and Exploding Gradient

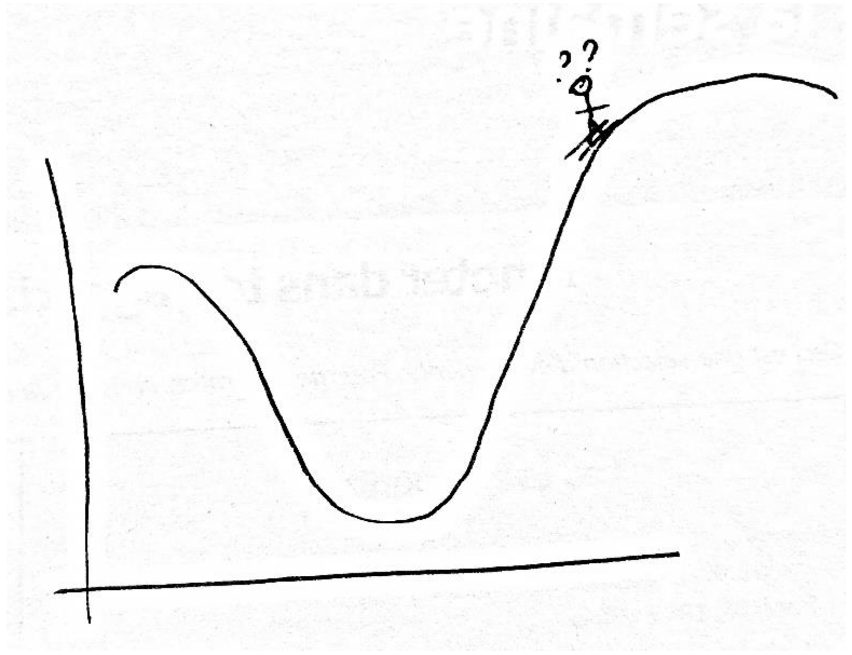
- There are two frequent problems in **<u>Deep Learning</u>:** ***exploding gradient*** **and** ***vanishing gradient***.
- In the first case, it's similar to having a too big learning rate. The algorithm is unstable and never converges.
- With Deep Learning, it can happen when your network is too deep. Since the gradients from each layer get multiplied with each other, you quickly obtain a gradient that explodes exponentially.

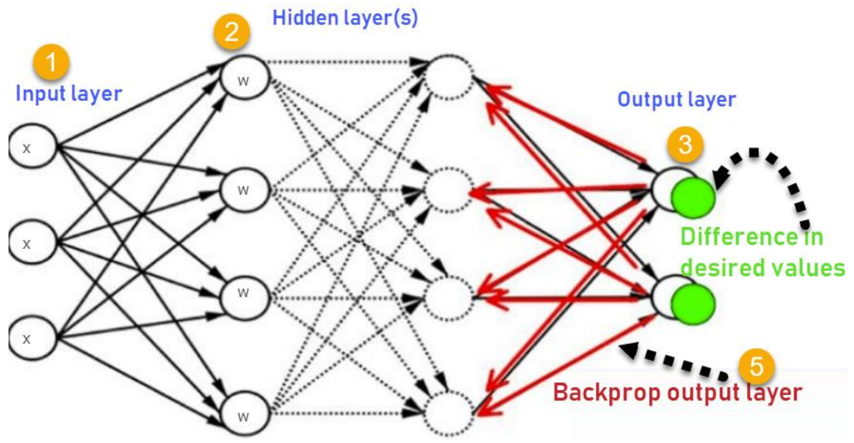




Vanishing Gradient

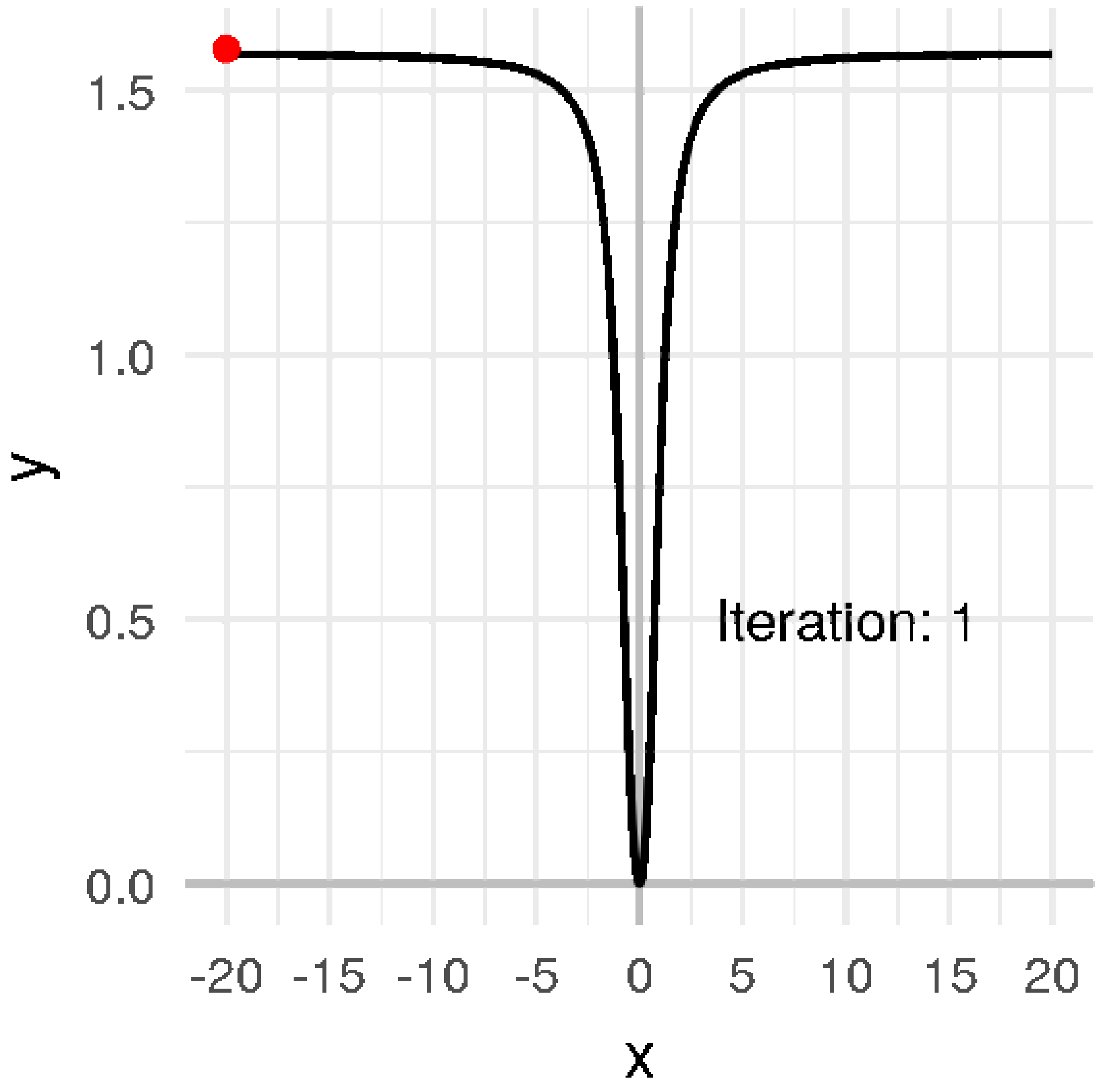
- For the vanishing gradient, it’s the opposite.
- The gradient becomes so small that the skier barely moves anymore.
- It can happen if the learning rate is too small.
- But it can also happen if the skier (the algorithm) is stuck on a flat line.





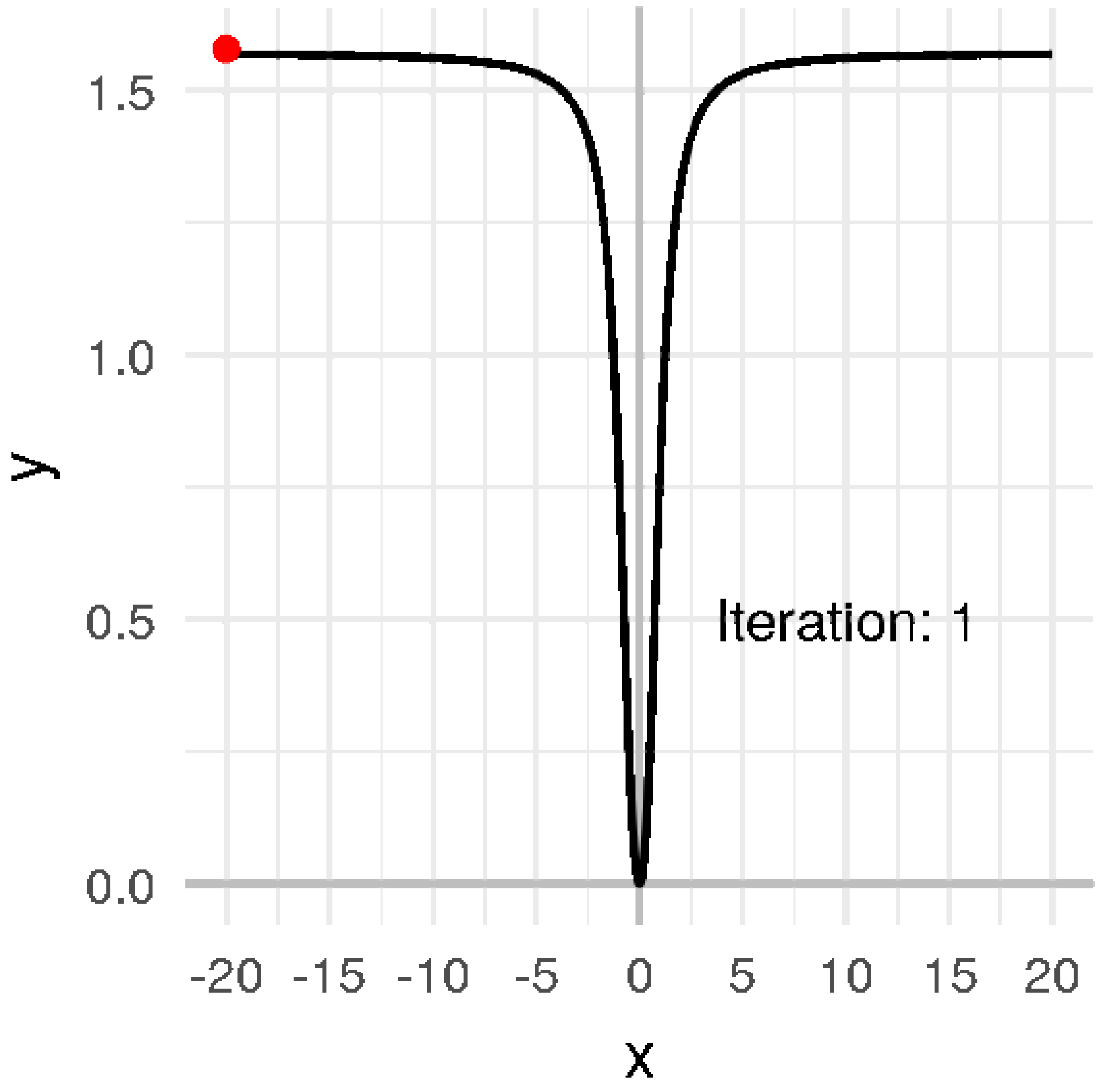
Vanishing Gradient

- It’s clear that the minimum is at x =0.
- But suppose the initial random point we pick is very far from 0, like −20.
- Even if we pick a big learning rate, the algorithm will take very long to converge.



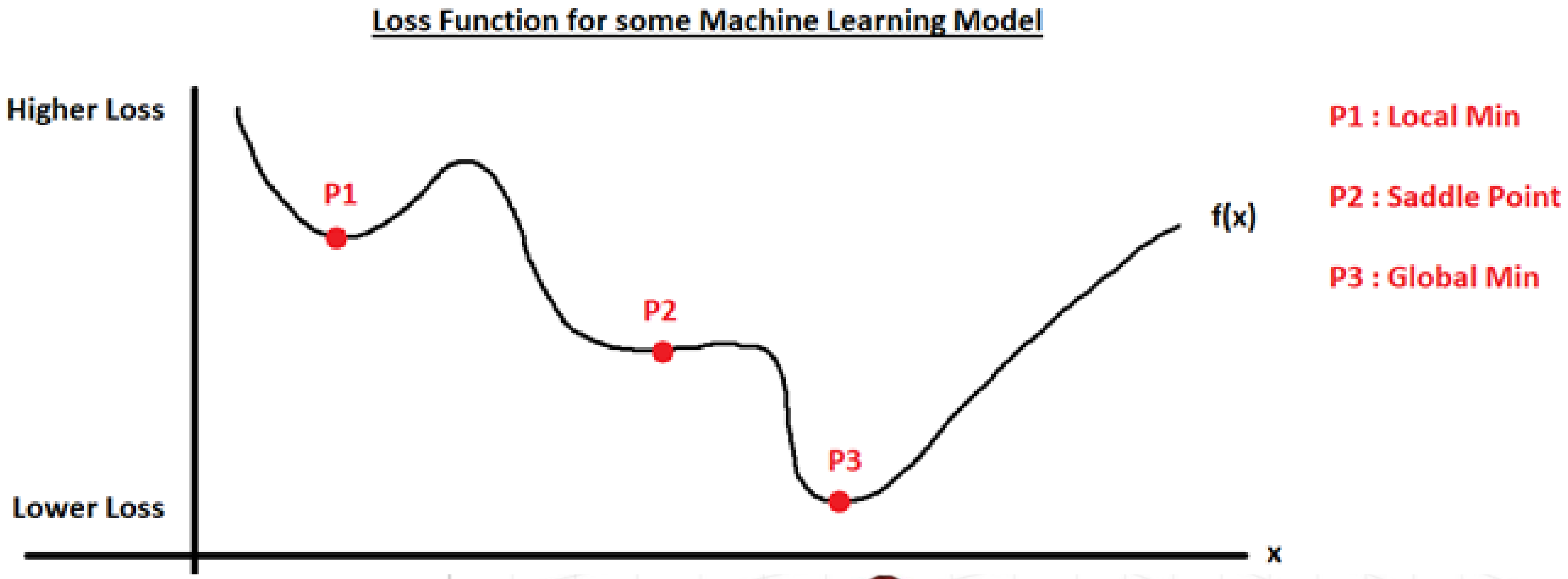
Vanishing Gradient

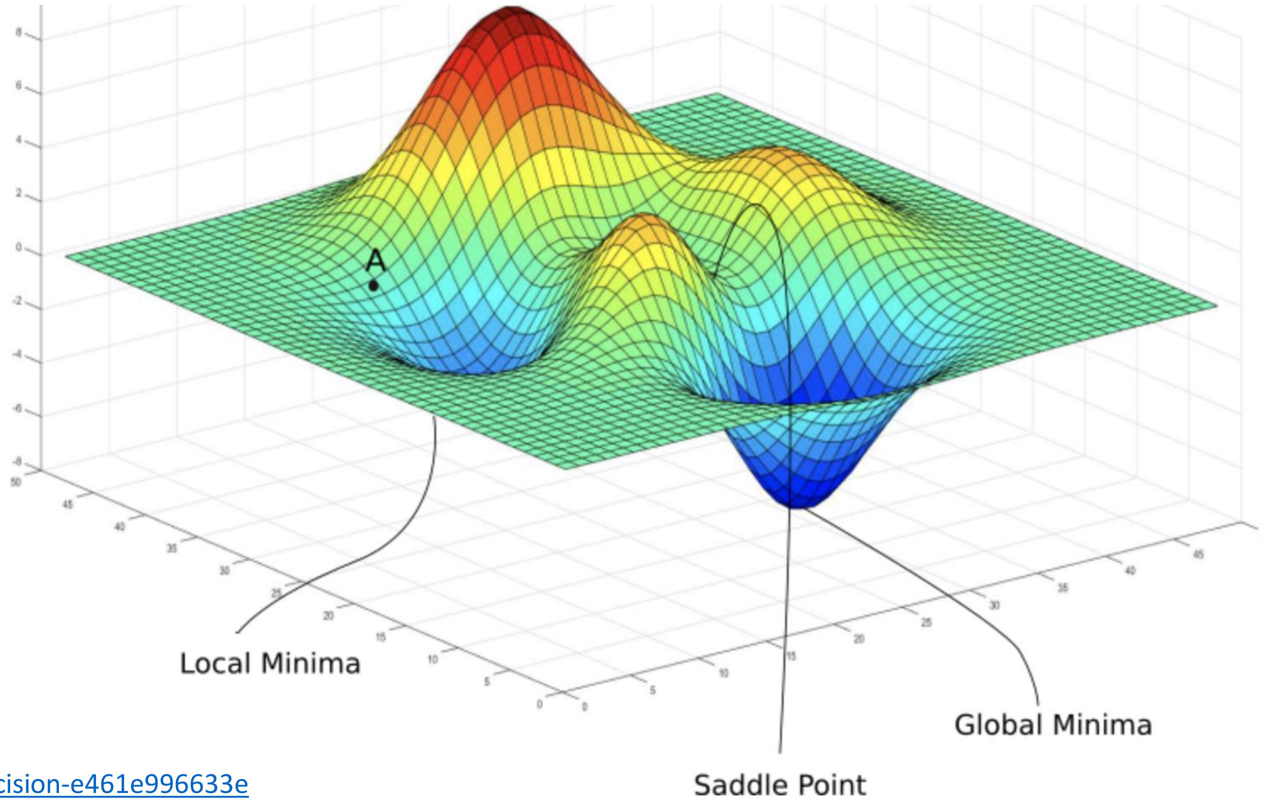
- In the shown example, we can still find the minimum. But with a more complex function that has flat lines in some places and is very curvy in other places it becomes a mess.
- In this case, whatever value you take for the learning rate, you'll be in trouble.
- Either a very slow convergence or an unstable algorithm.



Vanishing Gradient

- In the context of machine learning, a “saddle point” refers to a point in the optimization landscape of a cost function where the gradient is zero, but the point is neither a minimum nor a maximum.





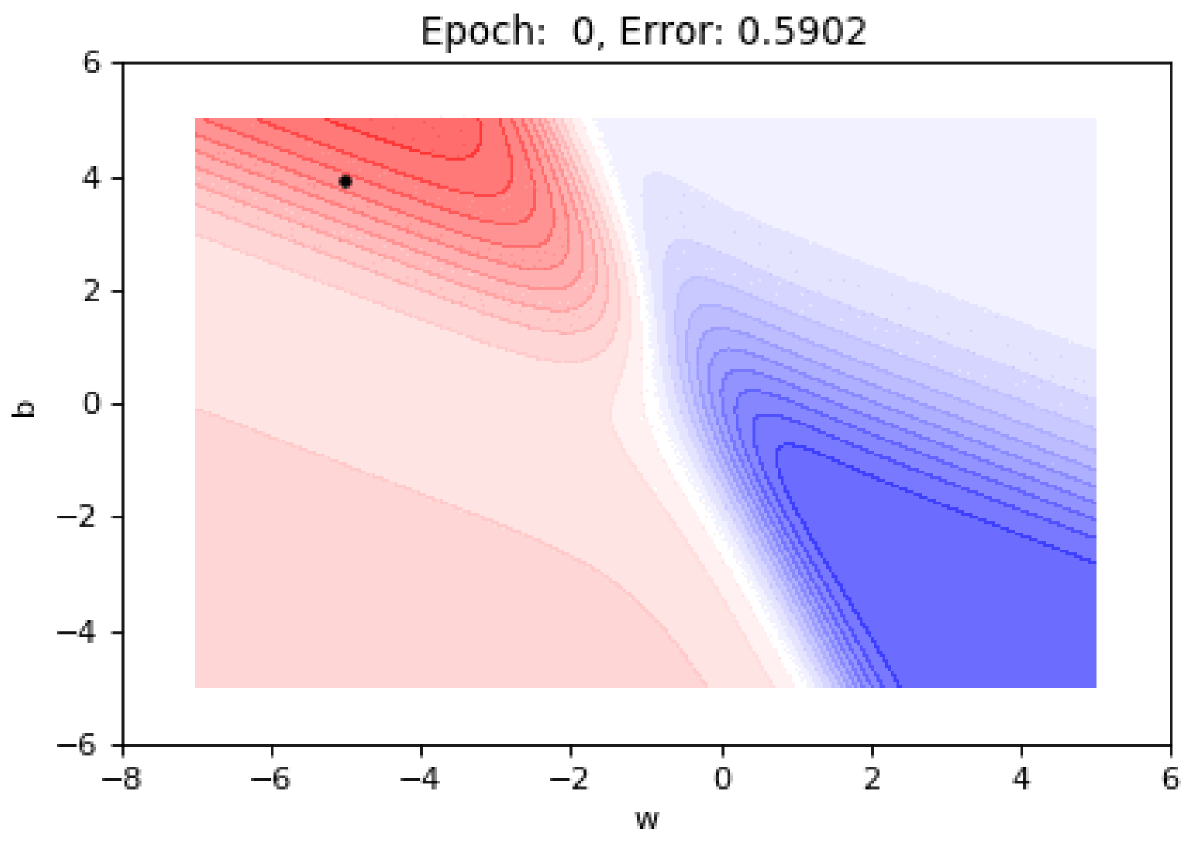
https://medium.com/@ompramod9921/mastering-gradient-descent-optimizing-neural-networks-with-precision-e461e996633e

GD Challenges Conclusion

GD Isn’t Enough

Better Optimization w.r.t. GD

- Consider a case with initialization in a flat surface where GD is used, and the error is not reducing when the gradient is in the flat surface.
- Even after a large number of epochs for e.g. 10000 the algorithm is not converging.
- The convergence is not achieved so easily, and the learning takes too much time.
- To overcome this problem **Momentum based gradient descent** is used.

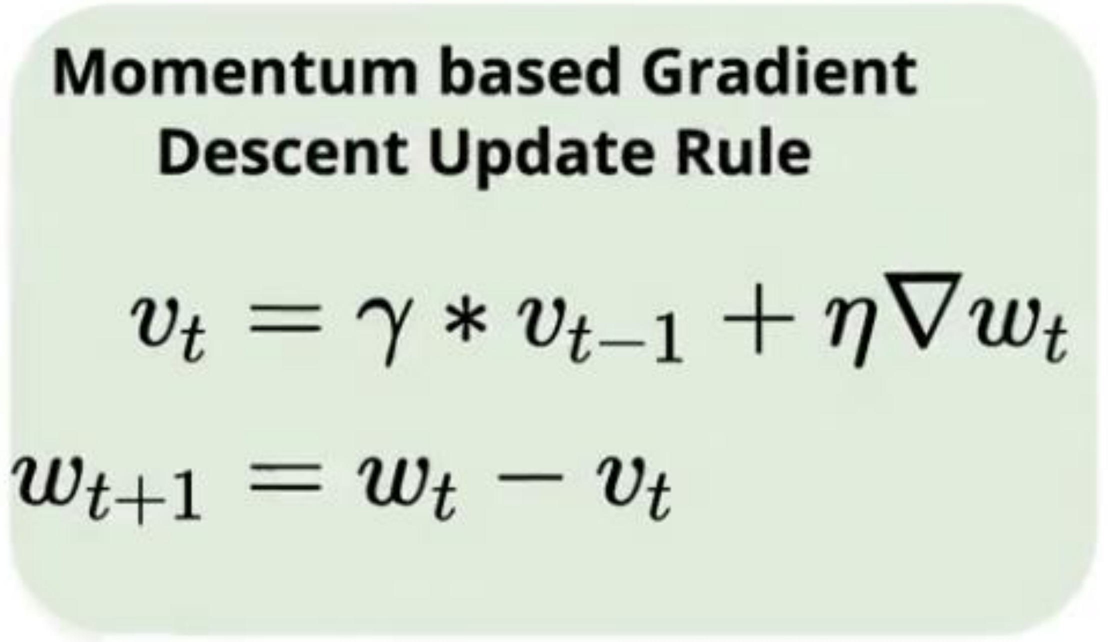


Momentum-based GD

- **Motivation:**
  - Consider a case where in order to reach to your desired destination you are continuously being asked to follow the same direction and once you become confident that you are following the right direction then you start taking ***bigger steps*** and you keep getting ***momentum*** in that same direction.
  - Similar to this if the ***gradient*** is in a ***flat surface*** for long term, then rather than taking constant steps it should take ***bigger steps*** and keep the momentum.
  - This approach is known as ***momentum based gradient descent***.

Momentum-based GD

- **Gamma parameter (γ)** is the momentum term which indicates how much acceleration you want.
- **γ** takes values between 0 and 1.
- Here along with the **current gradient (η∇w(t))**, the movement is also done according to history **(γ v(t−1))** so the **update** becomes larger which leads to faster movement and faster **convergence**.


Momentum based Gradient Descent Update Rule

$$v_t = \gamma * v_{t-1} + \eta \nabla w_t$$

$$w_{t+1} = w_t - v_t$$

Momentum-based GD

- **v(t)** is ***exponentially*** decaying weighted sum, as **t** increases **γ v(t−1)** becomes smaller and smaller i.e., this equation holds the farther updates by a small magnitude and recent updates by a large magnitude.

Momentum based Gradient Descent Update Rule

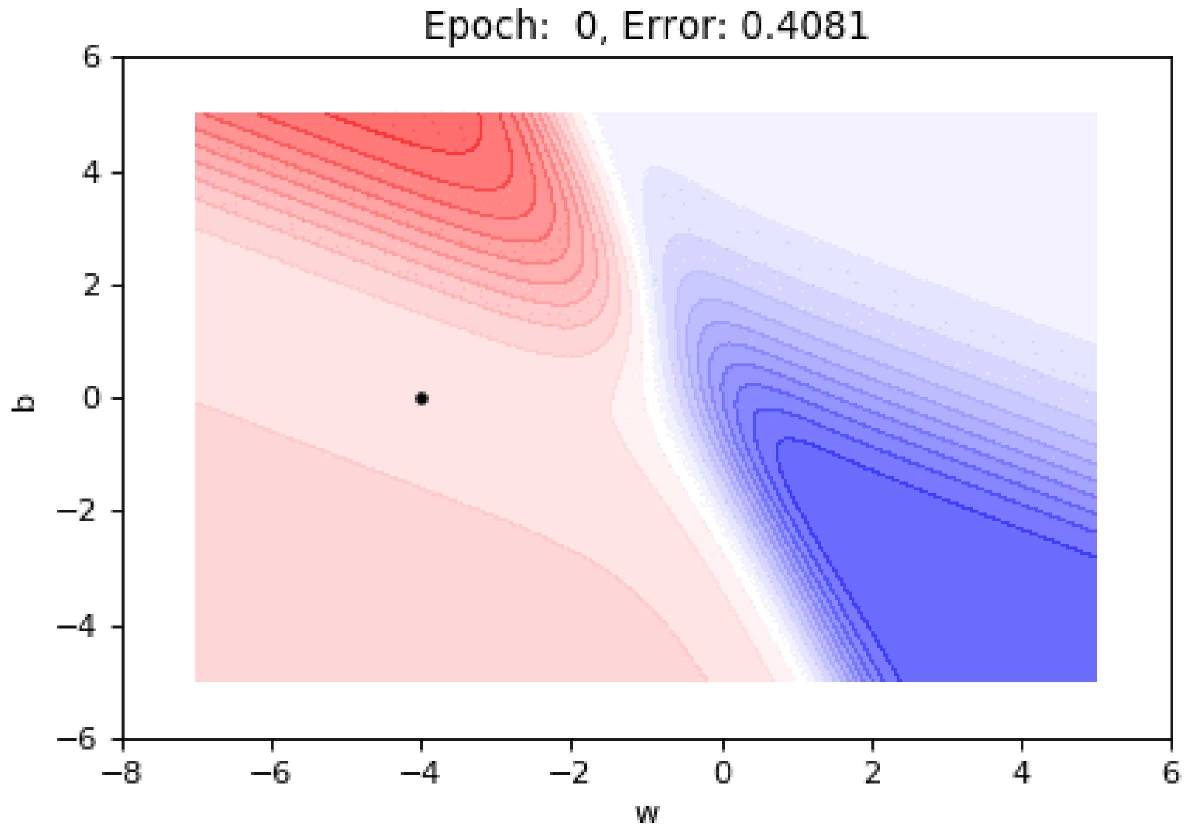
$$v_t = \gamma * v_{t-1} + \eta \nabla w_t$$

$$w_{t+1} = w_t - v_t$$

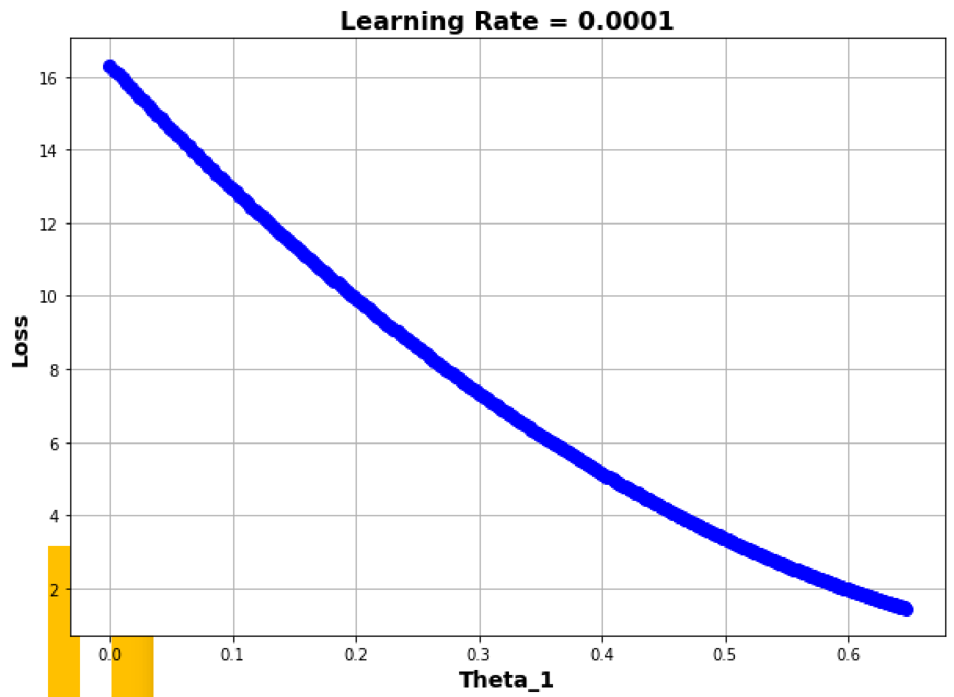
$$\begin{aligned}
v_0 &= 0 \\
v_1 &= \gamma \cdot v_0 + \eta \nabla w_1 = \eta \nabla w_1 \\
v_2 &= \gamma \cdot v_1 + \eta \nabla w_2 = \gamma \cdot \eta \nabla w_1 + \eta \nabla w_2 \\
v_3 &= \gamma \cdot v_2 + \eta \nabla w_3 = \gamma(\gamma \cdot \eta \nabla w_1 + \eta \nabla w_2) + \eta \nabla w_3 \\
&= \gamma \cdot v_2 + \eta \nabla w_3 = \gamma^2 \cdot \eta \nabla w_1 + \gamma \cdot \eta \nabla w_2 + \eta \nabla w_3 \\
v_4 &= \gamma \cdot v_3 + \eta \nabla w_4 = \gamma^3 \cdot \eta \nabla w_1 + \gamma^2 \cdot \eta \nabla w_2 + \gamma \cdot \eta \nabla w_3 + \eta \nabla w_4 \\
&\vdots \\
v_t &= \gamma \cdot v_{t-1} + \eta \nabla w_t = \gamma^{t-1} \cdot \eta \nabla w_1 + \gamma^{t-2} \cdot \eta \nabla w_1 + \ldots + \eta \nabla w_t
\end{aligned}$$

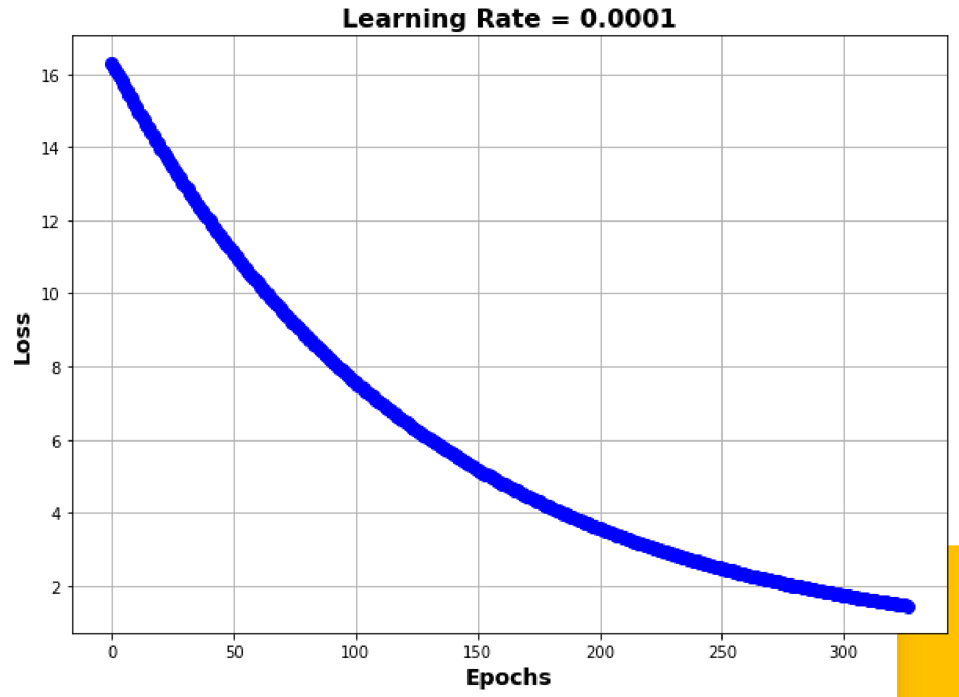
Momentum-based GD

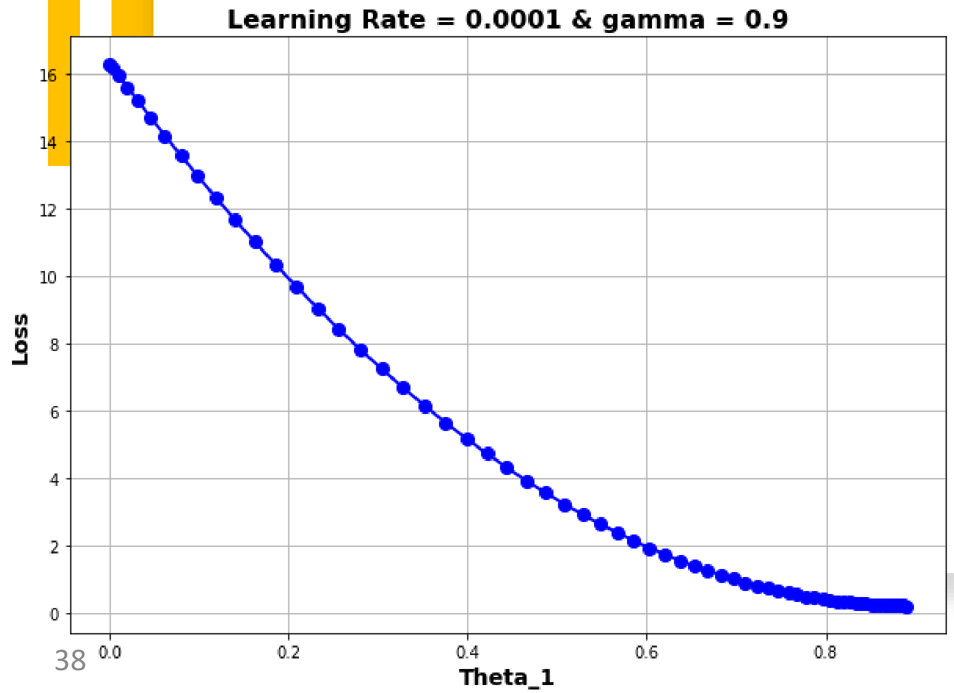
- This algorithm **adds momentum** in the direction of **consistent gradients** and **cancels the momentum** if the gradients are in **different directions.**

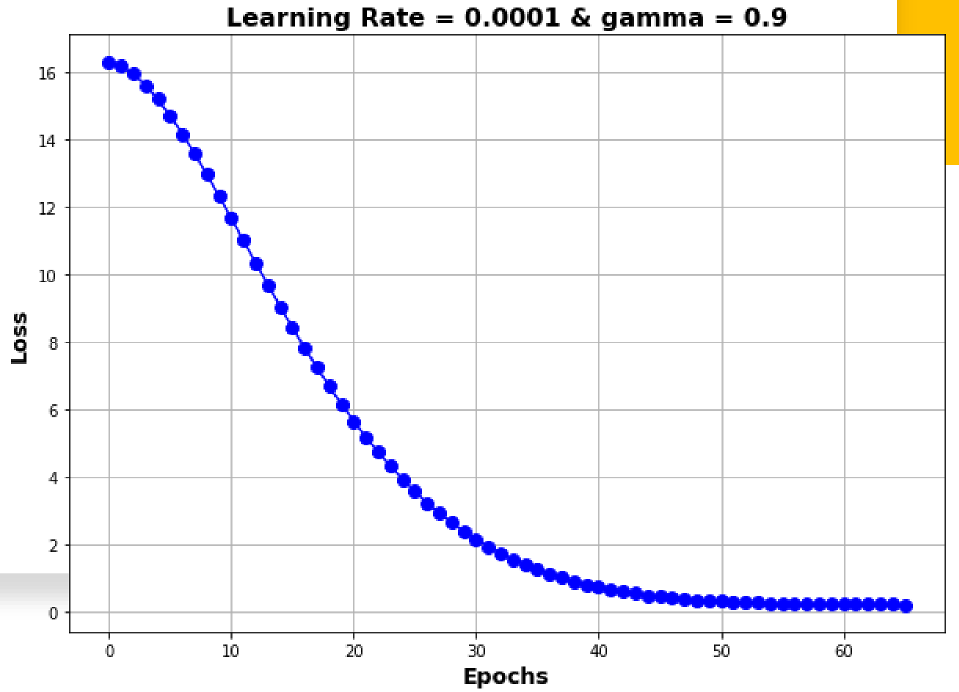


Momentum -based GD







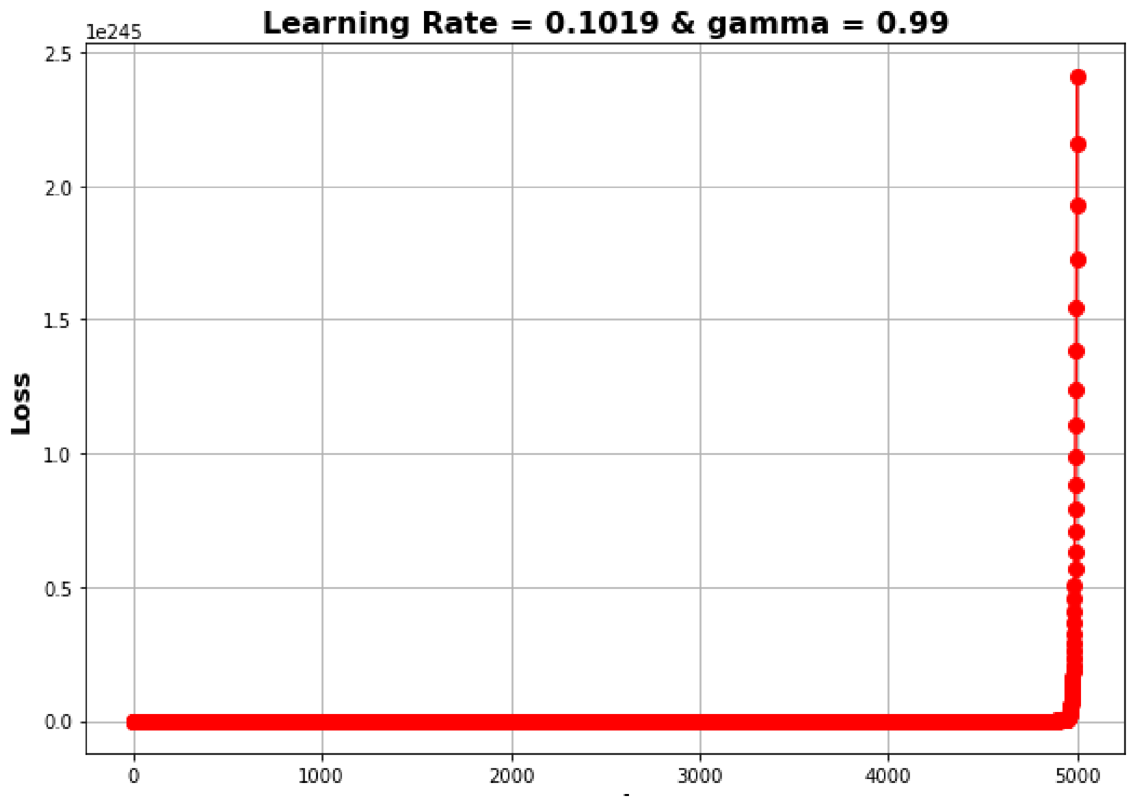


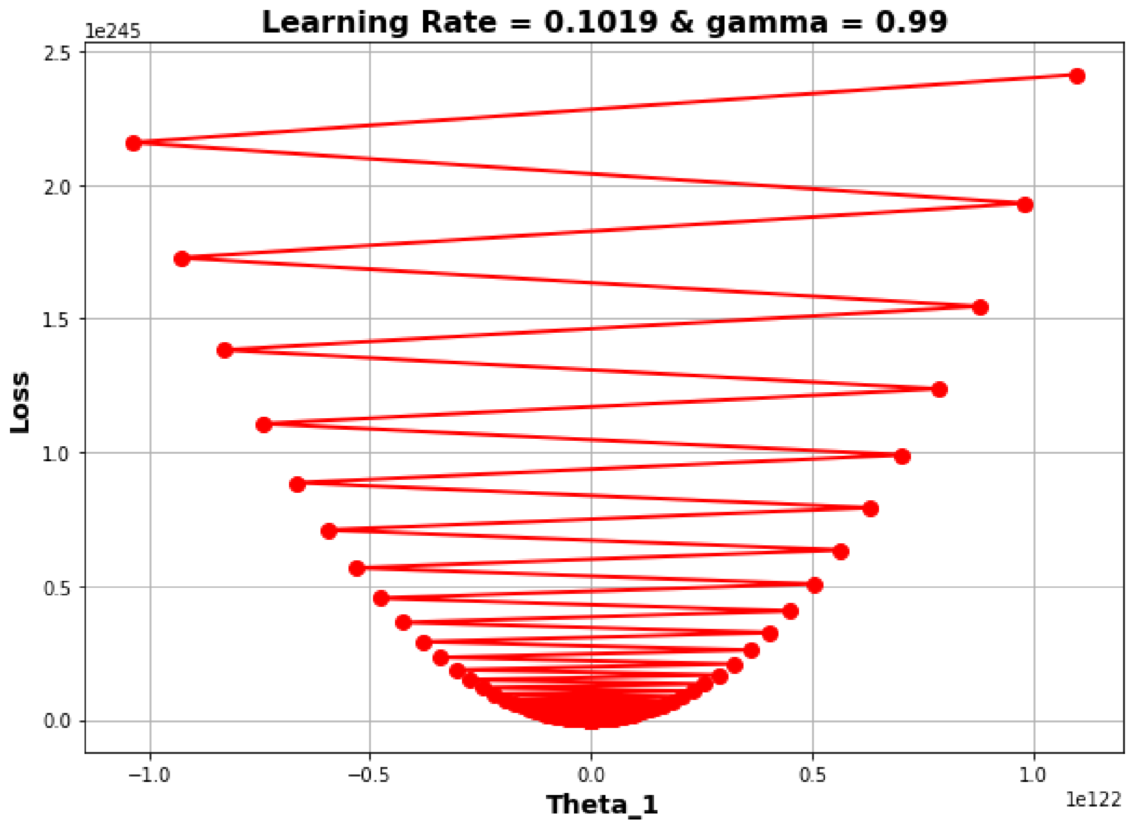
Momentum-based GD

- What will happen if we increase the value of the momentum term **(γ)**?

Momentum-based GD

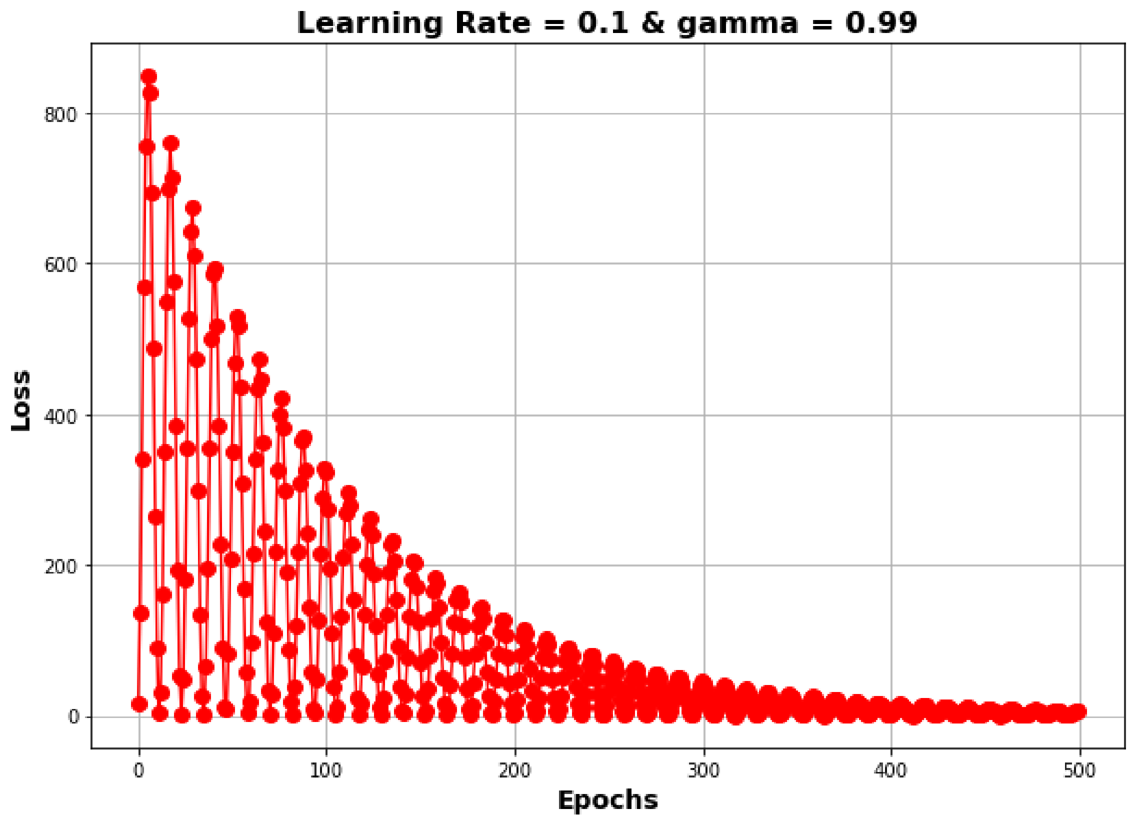
What will happen if we increase the value of the momentum term **(γ)**?

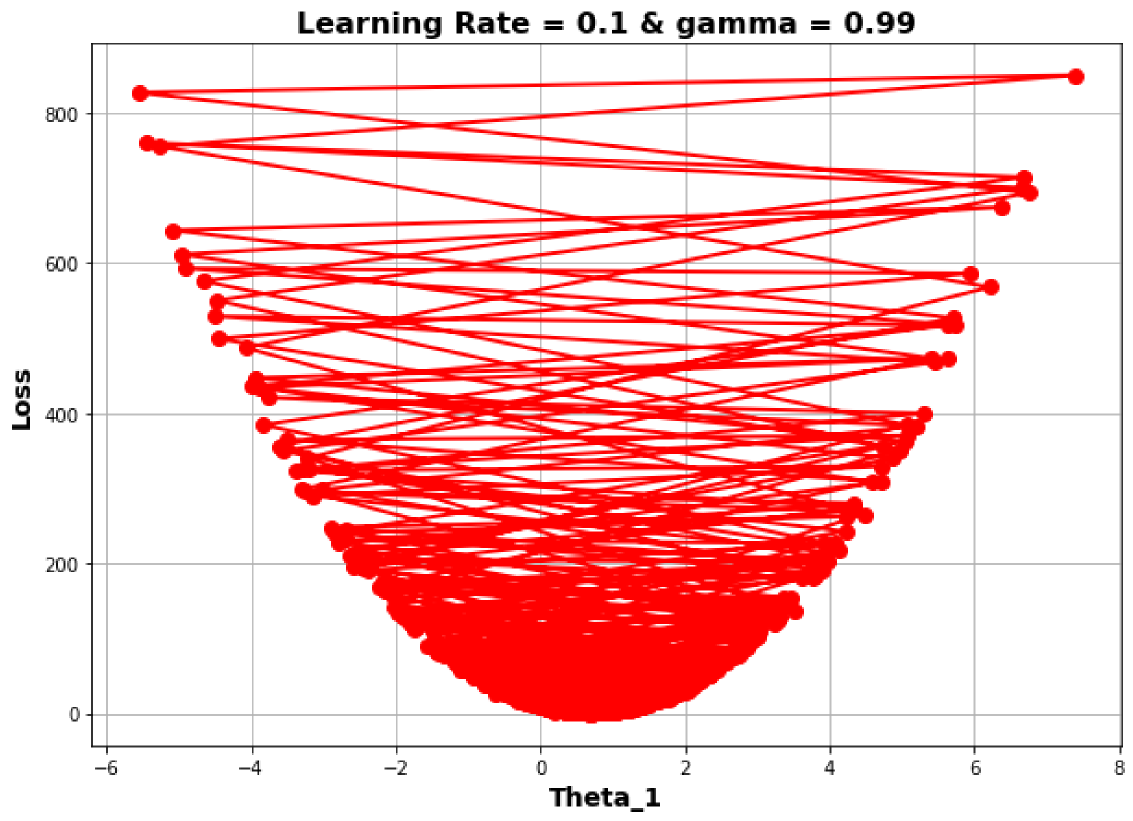




Momentum-based GD

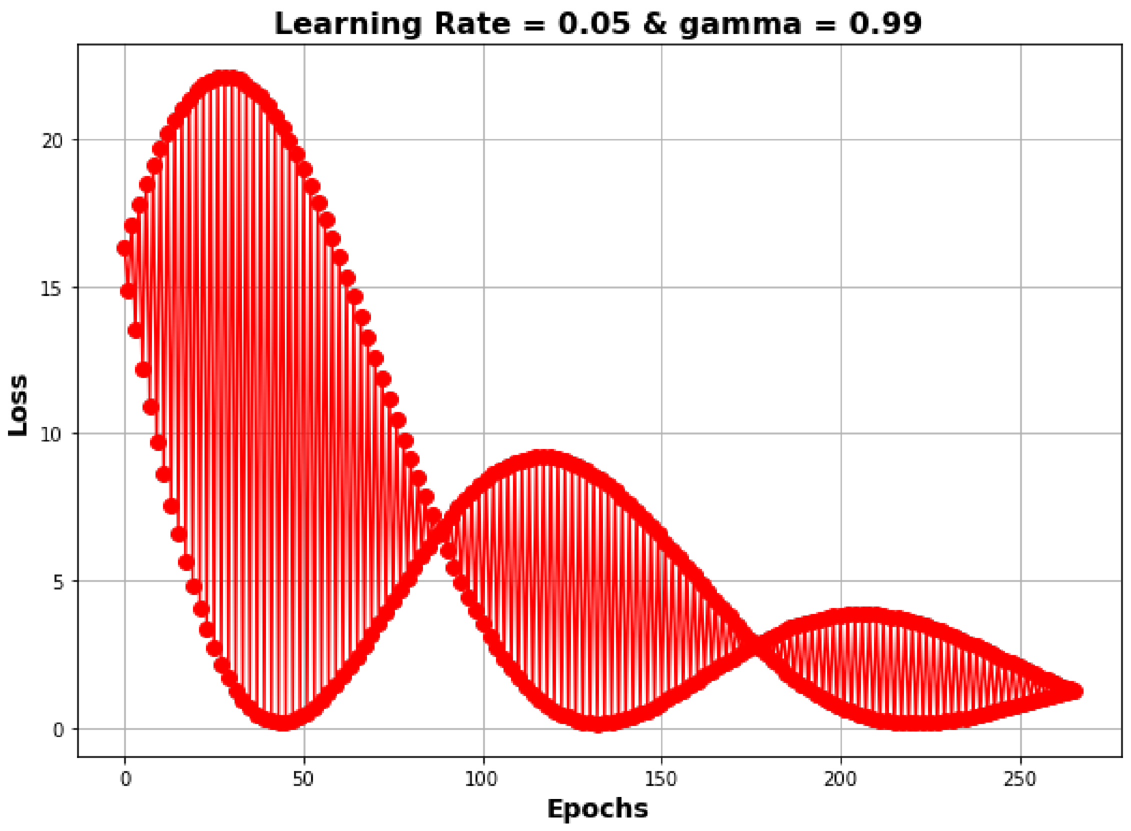
What will happen if we increase the value of the momentum term **(γ)**?

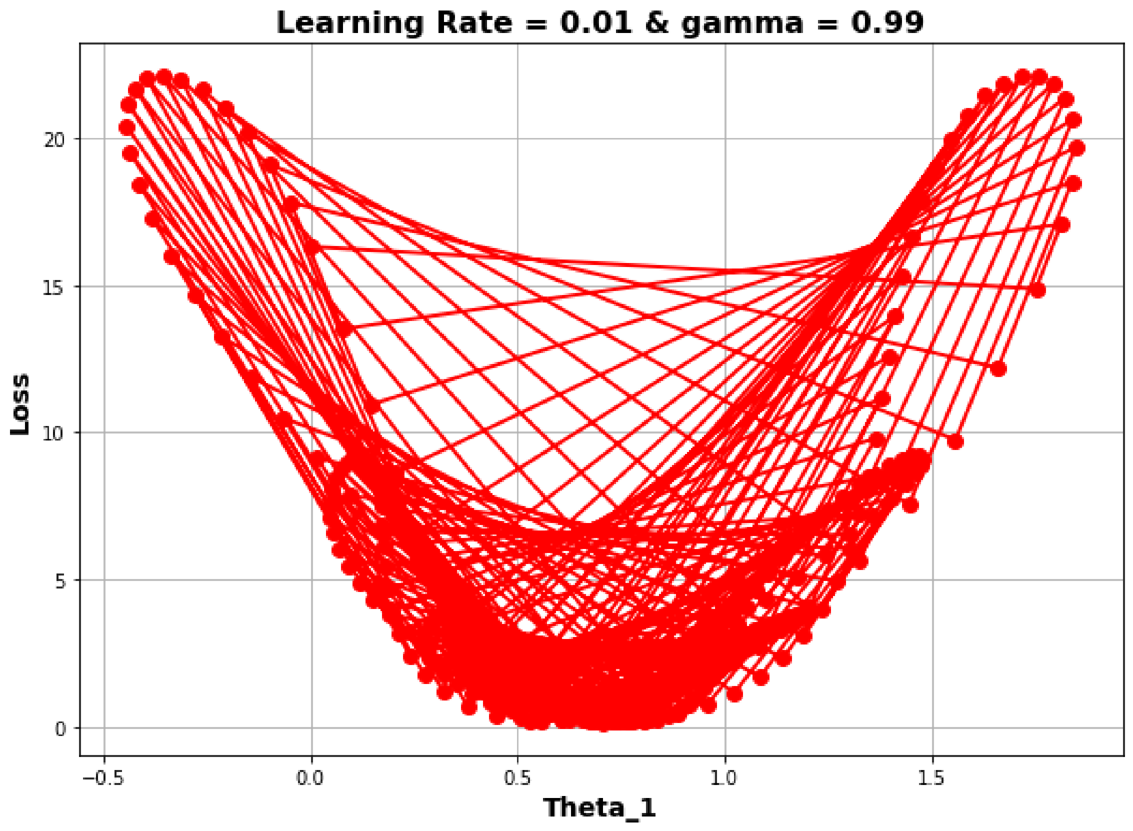




Momentum-based GD

What will happen if we increase the value of the momentum term **(γ)**?

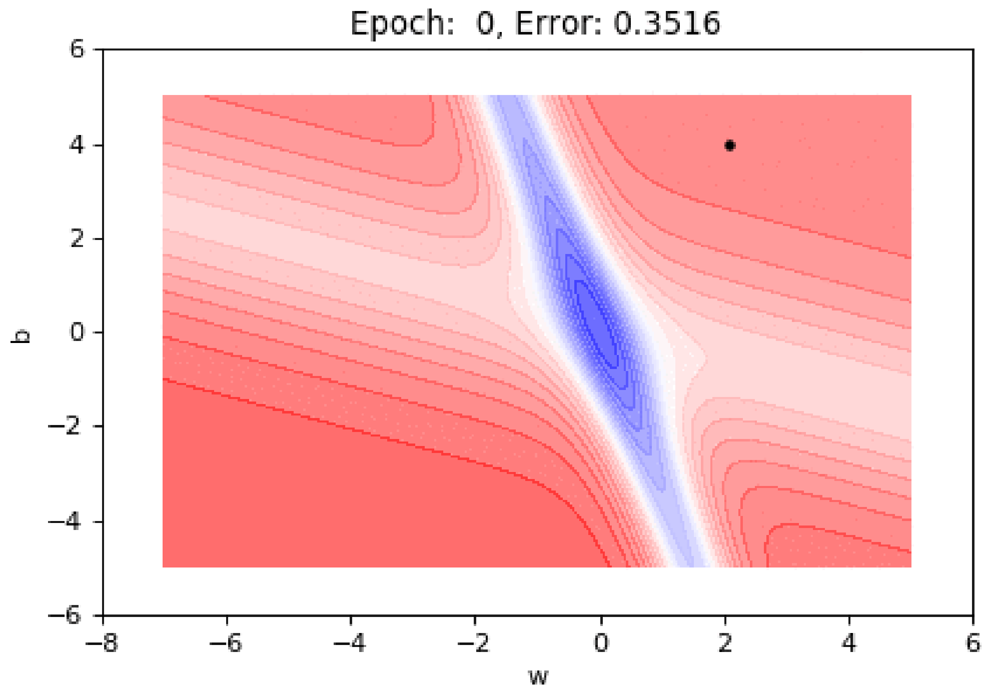


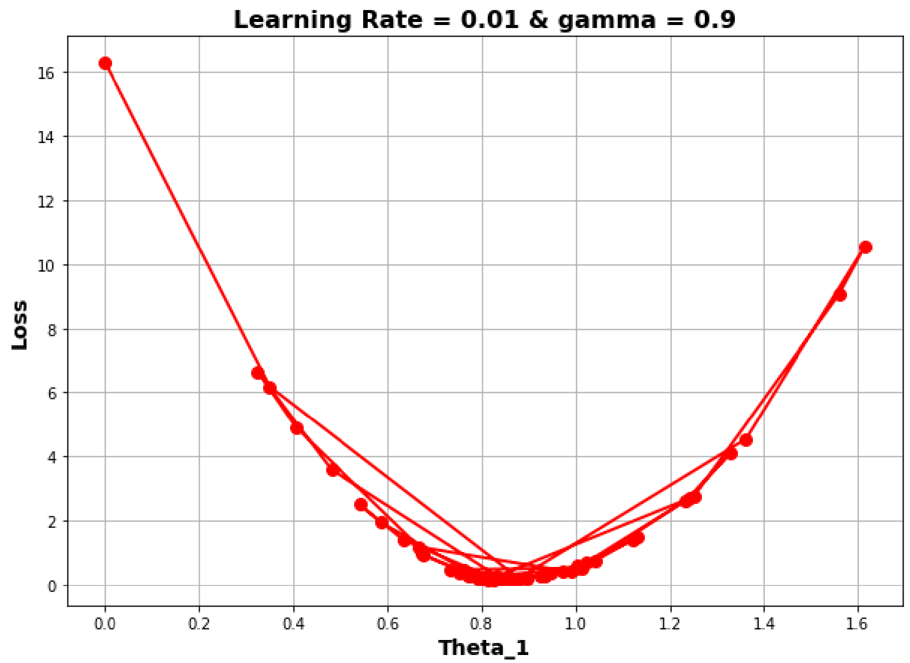


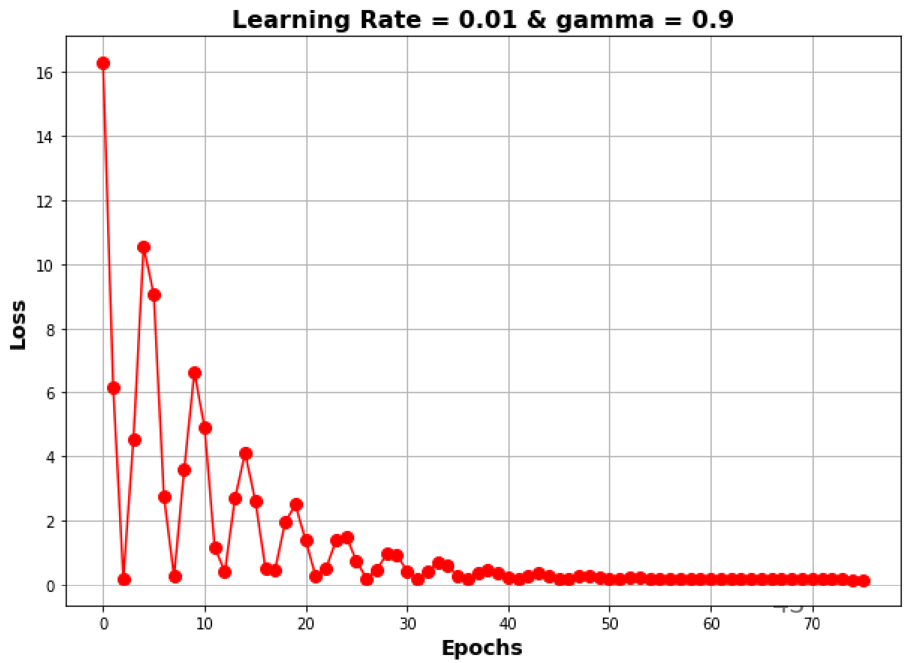
Momentum-based GD

Momentum based GD **oscillates** in and out of the minima valley i.e., making **U-turns** near the minima **(however it converges faster vanilla GD)**.

To overcome this issue **Nesterov accelerated GD (NAG)** is used.







Momentum vs. Nesterov Accelerated GD

Standard Momentum

- first compute the gradient at the current position;
- then take a big jump in the direction of the accumulated gradient.

Momentum based Gradient Descent Update Rule

$$v_t = \gamma * v_{t-1} + \eta \nabla w_t$$

$$w_{t+1} = w_t - v_t$$

NAG

- first make a big jump in the direction of the previous accumulated gradient; (looking ahead step)
- then measure the gradient where you end up and make a correction.

NAG Update Rule

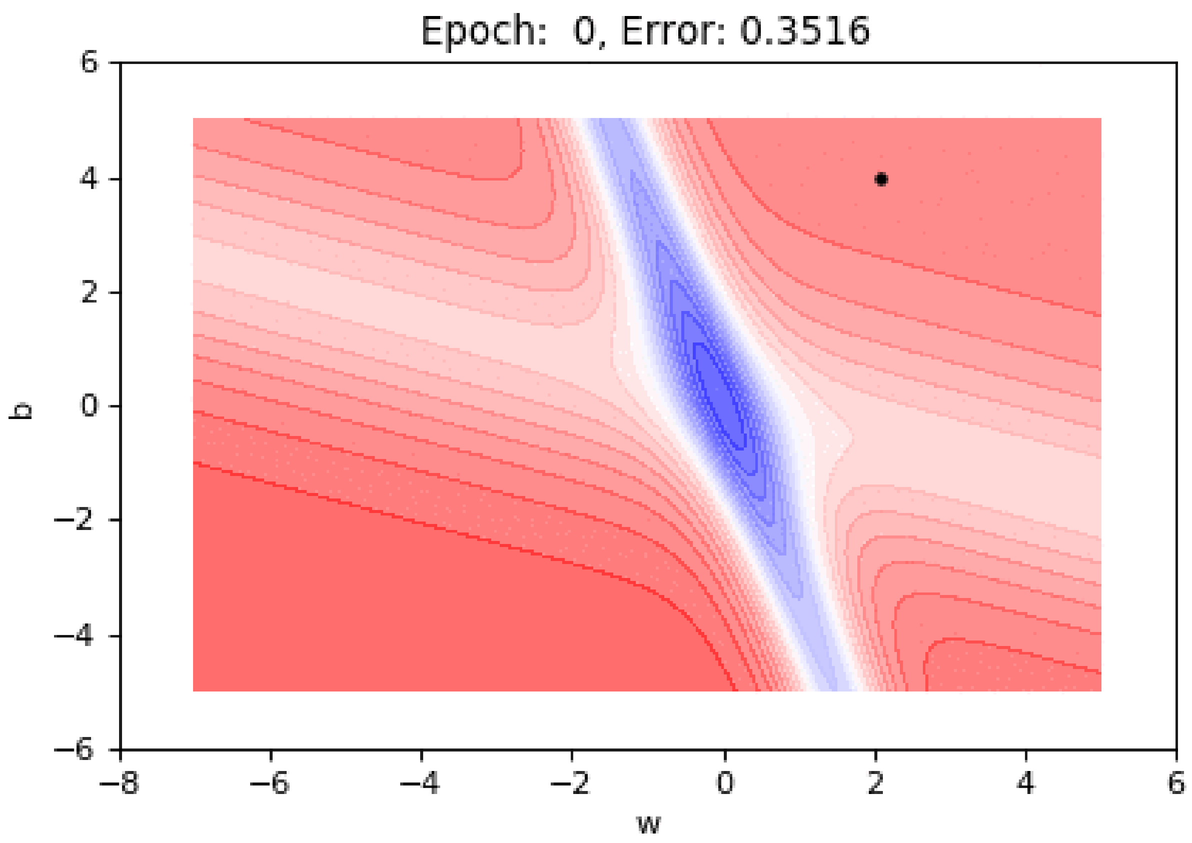
$$w_{temp} = w_t - \gamma * v_{t-1}$$

$$w_{t+1} = w_{temp} - \eta \nabla w_{temp}$$

$$v_t = \gamma * v_{t-1} + \eta \nabla w_{temp}$$

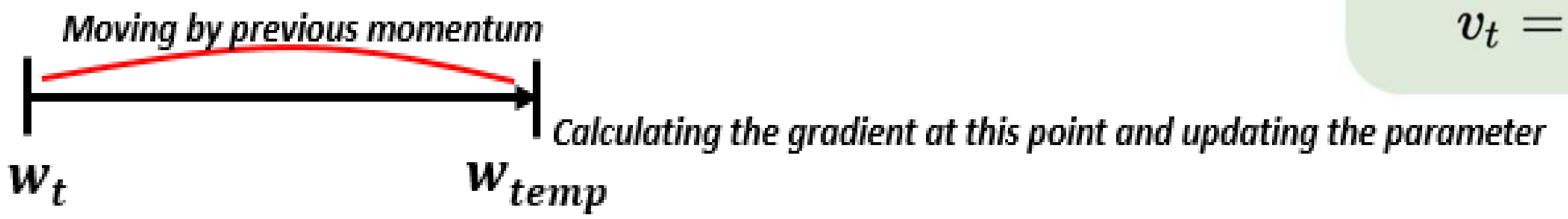
Nesterov Accelerated GD (NAG)

- This looking ahead helps **NAG** in finishing its job (finding the minima) quicker than **momentum-based GD**. Hence the **oscillations** are **less** compared to **momentum-based GD** and there are fewer chances of missing the **minima**.



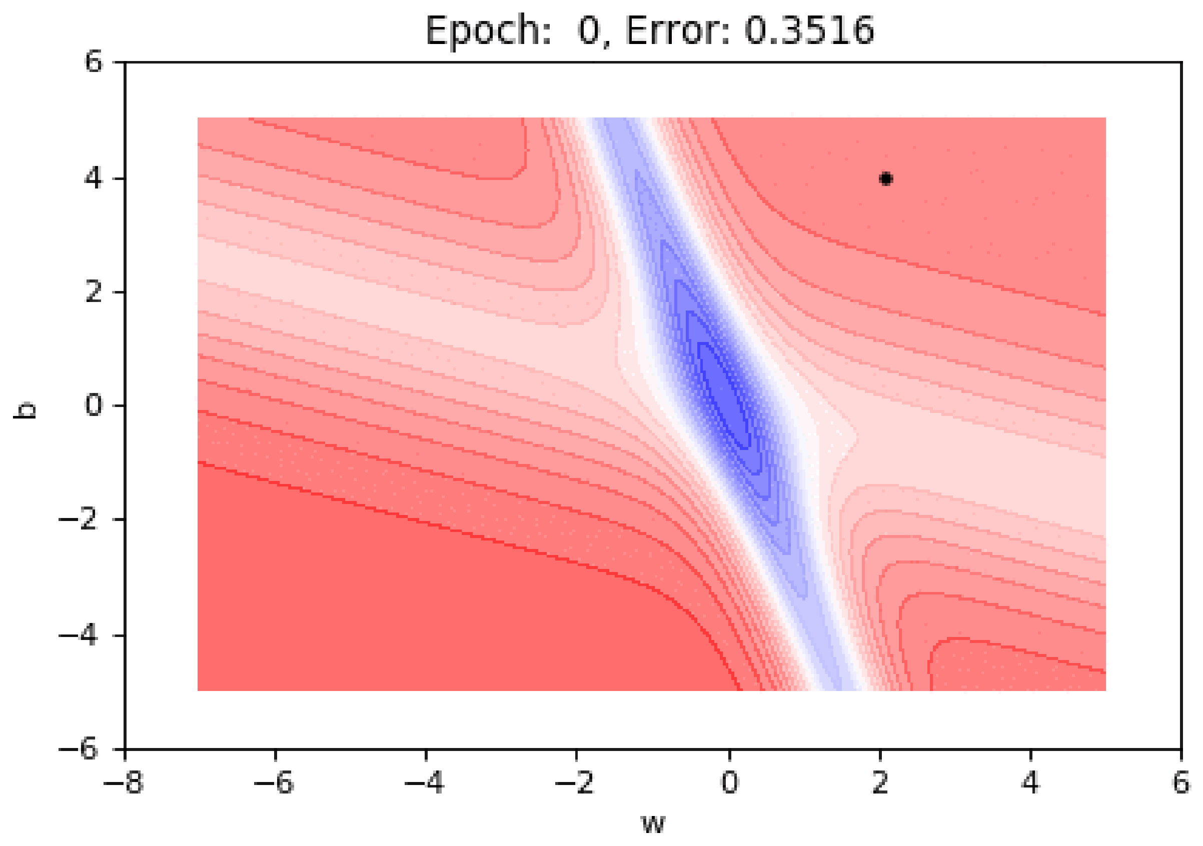
NAG Update Rule

$$w_{temp} = w_t - \gamma * v_{t-1}$$
$$w_{t+1} = w_{temp} - \eta \nabla w_{temp}$$
$$v_t = \gamma * v_{t-1} + \eta \nabla w_{temp}$$

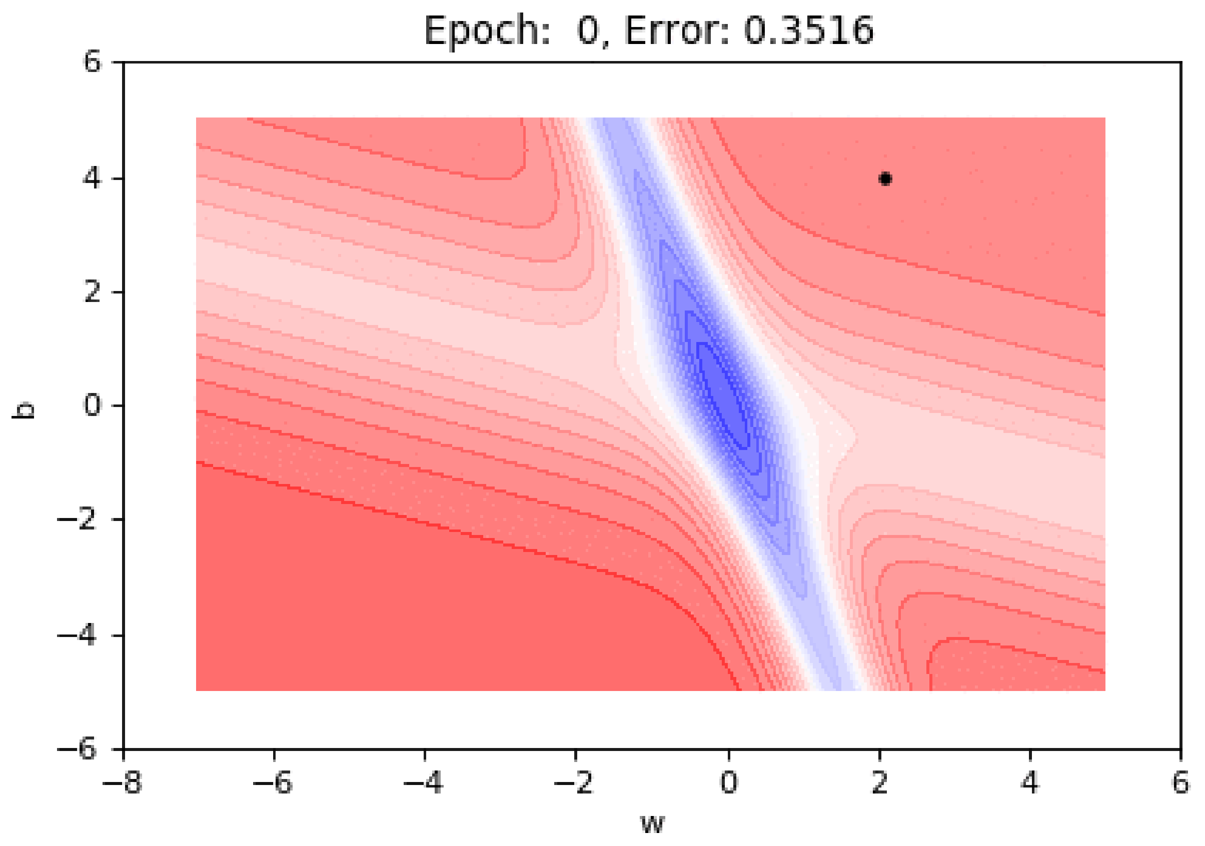

Moving by previous momentum
Calculating the gradient at this point and updating the parameter
w_t
w_{temp}

Momentum vs. Nesterov Accelerated GD

Standard Momentum



NAG



Resources

- Andrew Ng, Machine Learning, Stanford University, Coursera
- https://ruder.io/optimizing-gradient-descent/index.html#batchgradientdescent
- https://towardsdatascience.com/calculus-behind-linear-regression-1396cfd0b4a9
- https://towardsdatascience.com/why-gradient-descent-isnt-enough-a-comprehensive-introduction-to-optimization-algorithms-in-59670fd5c096
- https://towardsdatascience.com/why-gradient-descent-isnt-enough-a-comprehensive-introduction-to-optimization-algorithms-in-59670fd5c096
- https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3
- https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3
- https://www.charlesbordet.com/en/gradient-descent/#how-does-it-work
- https://www.charlesbordet.com/en/gradient-descent/#how-to-find-a-good-value-for-the-learning-rate
- https://www.charlesbordet.com/en/gradient-descent/#what-to-do-in-case-of-local-minima
- https://towardsdatascience.com/why-gradient-descent-isnt-enough-a-comprehensive-introduction-to-optimization-algorithms-in-59670fd5c096
- https://medium.com/hackernoon/implementing-different-variants-of-gradient-descent-optimization-algorithm-in-python-using-numpy-809e7ab3bab4
- https://ranasinghiitkgp.medium.com/optimization-algorithm-d62ac8f597b1
- http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

Resources

- https://towardsdatascience.com/why-gradient-descent-isnt-enough-a-comprehensive-introduction-to-optimization-algorithms-in-59670fd5c096
- https://ranasinghiitkgp.medium.com/optimization-algorithm-d62ac8f597b1
- https://www.youtube.com/watch?v=lAq96T8FkTw&ab_channel=DeepLearningAI
- https://www.youtube.com/watch?v=NxTFlzBjS-4&ab_channel=DeepLearningAI
- https://www.youtube.com/watch?v=lWzo8CajF5s&ab_channel=DeepLearningAIDeepLearningAI
- https://medium.datadriveninvestor.com/exponentially-weighted-average-for-deep-neural-networks-39873b8230e9
- https://towardsdatascience.com/understanding-gradient-descent-and-adam-optimization-472ae8a78c10
- DOI: 10.1177/0735633118757015
- https://www.asimovinstitute.org/author/fjodorvanveen/
- guru99.com
- https://towardsdatascience.com/an-introduction-to-reinforcement-learning-1e7825c60bbe