Gradient descent optimization algorithms

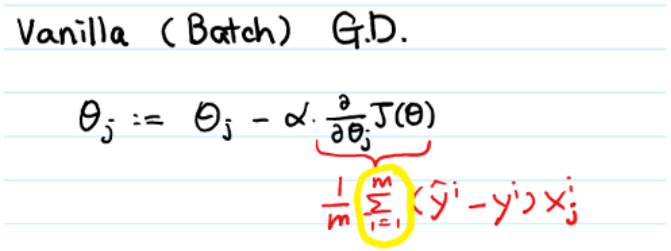
Gradient descent optimization algorithms

Gradient descent is one of the most popular algorithm to perform optimization and by far the most common way to optimize neural networks.

Gradient descent variants

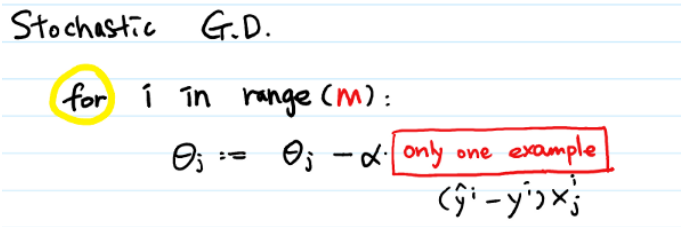
Batch Gradient descent

Batch gradient descent calculates the error for each example in the training dataset, but only updates the model after all training examples have been evaluated.



Mini-Batch Gradient Descent

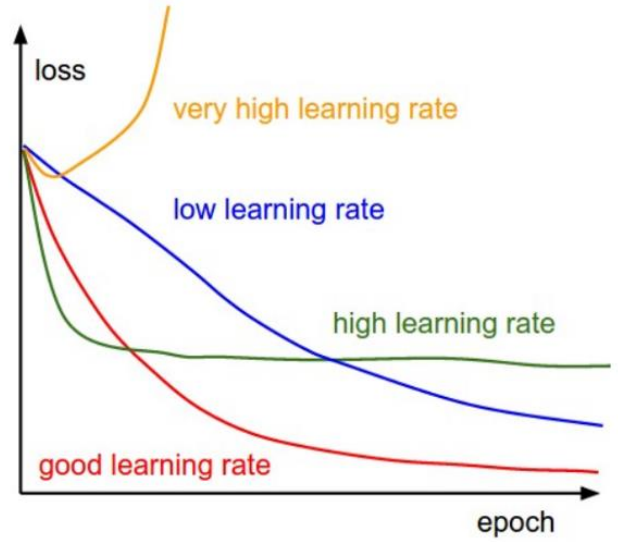
Mini-Batch gradient descent algorithm that splits the training dataset into small batches that are used to calculate model error and update model coefficients.



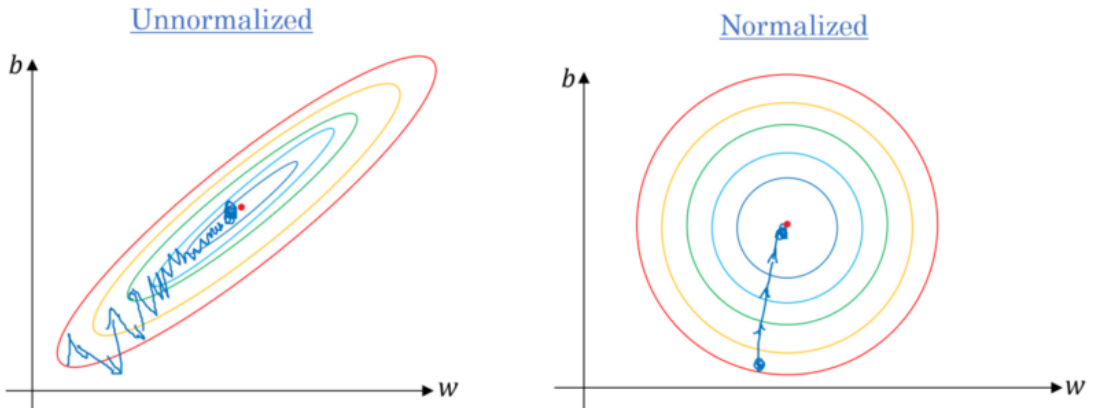
Stochastic Gradient Descent

Stochastic gradient descent, often abbreviated SGD, is a variation of the gradient descent algorithm that calculates the error and updates the model for each example in the training dataset.

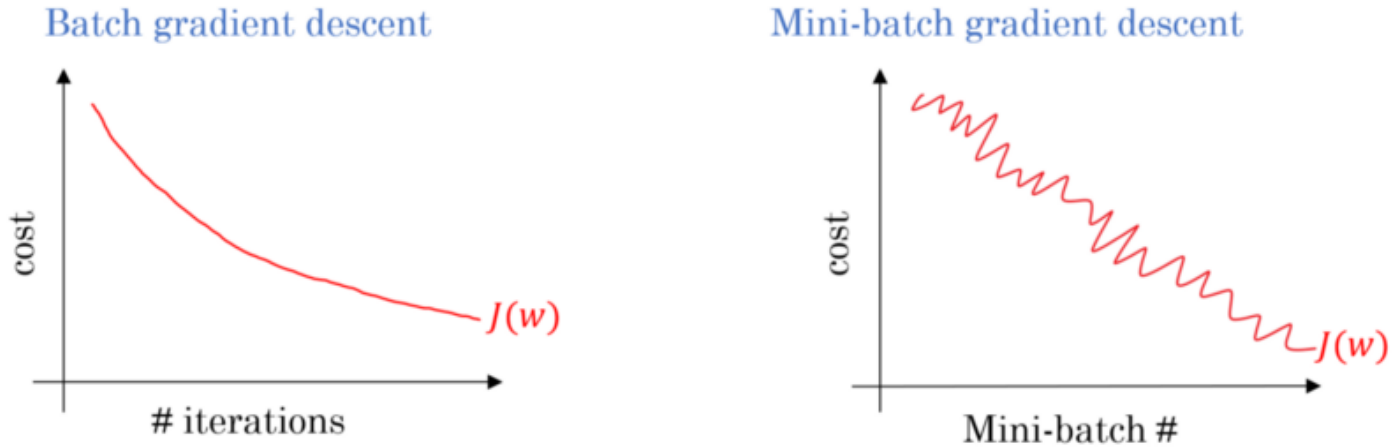
http://ruder.io/optimizing-gradient-descent/
https://towardsdatascience.com/difference-between-batch-gradient-descent-and-stochastic-gradient-descent-1187f1291aa1



Gradient descent with different learning rates.



Gradient descent: normalized versus unnormalized level curves.



Gradient descent: batch versus mini-batch loss function

https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3

Batch Gradient descent : Advantages

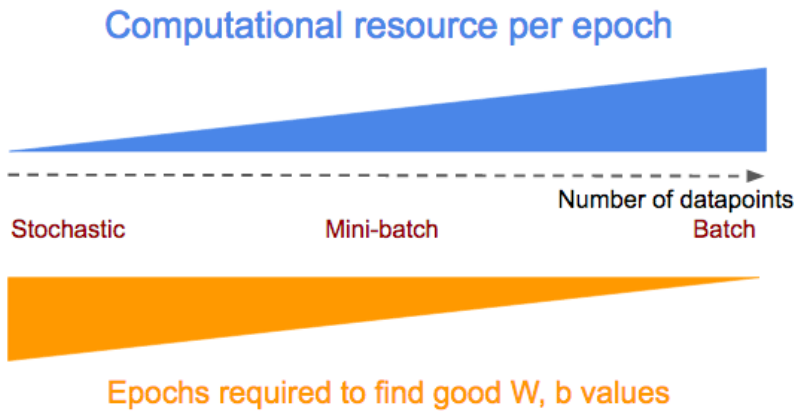
- It has straight trajectory towards the minimum and it is guaranteed to converge in theory to the global minimum if the loss function is convex and to a local minimum if the loss function is not convex.

- It has unbiased estimate of gradients. The more the examples, the lower the standard error.

Batch Gradient descent : Disadvantages

- Even though we can use vectorized implementation, it may still be slow to go over all examples especially when we have large datasets.

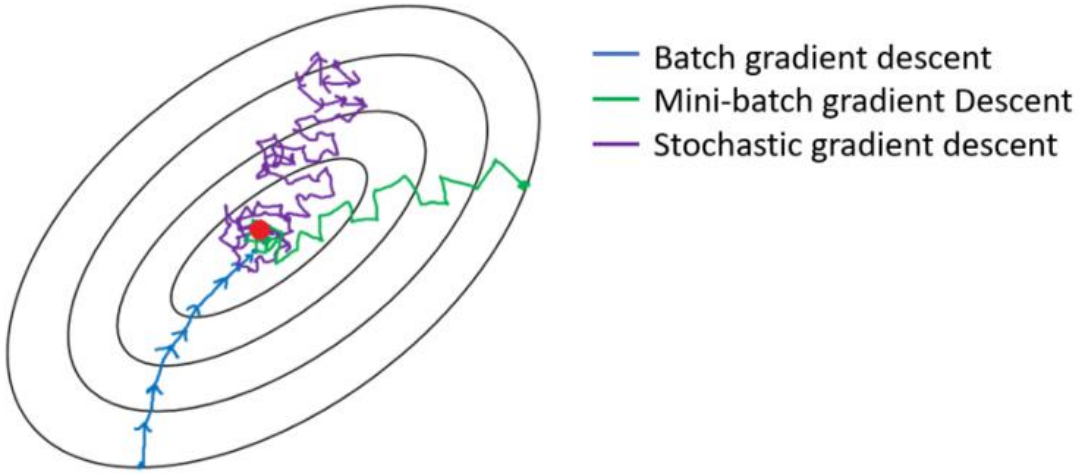
- Each step of learning happens after going over all examples where some examples may be redundant and don't contribute much to the update.



https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3
https://www.kdnuggets.com/2016/08/gentlest-introduction-tensorflow-part-2.html/2

Stochastic Gradient descent

- Shuffle the training data set to avoid pre-existing order of examples.
- Partition the training data set into *m* examples.
- It adds even more noise to the learning process than mini-batch that helps improving generalization error. However, this would increase the run time.
- We can't utilize vectorization over 1 example and becomes very slow. Also, the variance becomes large since we only use 1 example for each learning step.



Gradient descent variants' trajectory towards minimum

Mini Batch Gradient descent

- Shuffle the training data set to avoid pre-existing order of examples.
- Partition the training data set into *b* mini-batches based on the batch size. If the training set size is not divisible by batch size, the remaining will be its own batch.
- Usually chosen as power of 2 such as 32, 64, 128, 256, 512, etc. The reason behind it is because some hardware such as GPUs achieve better run time with common batch sizes such as power of 2.

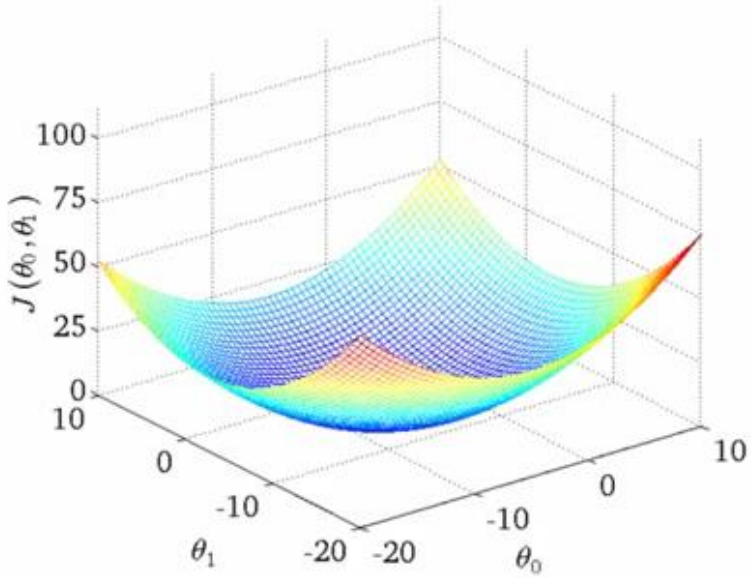
Advantages

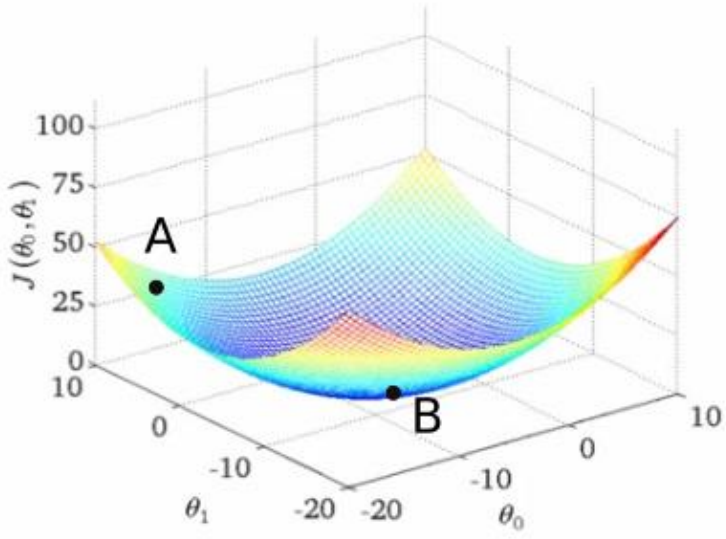
- Faster than Batch version because it goes through a lot less examples than Batch (all examples).
- Randomly selecting examples will help avoid redundant examples or examples that are very similar that don't contribute much to the learning.
- With batch size < size of training set, it adds noise to the learning process that helps improving generalization error.

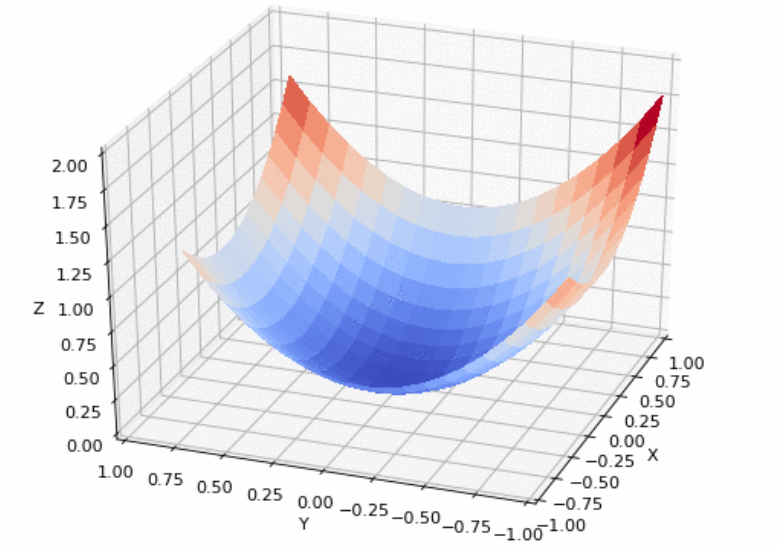
Disadvantages

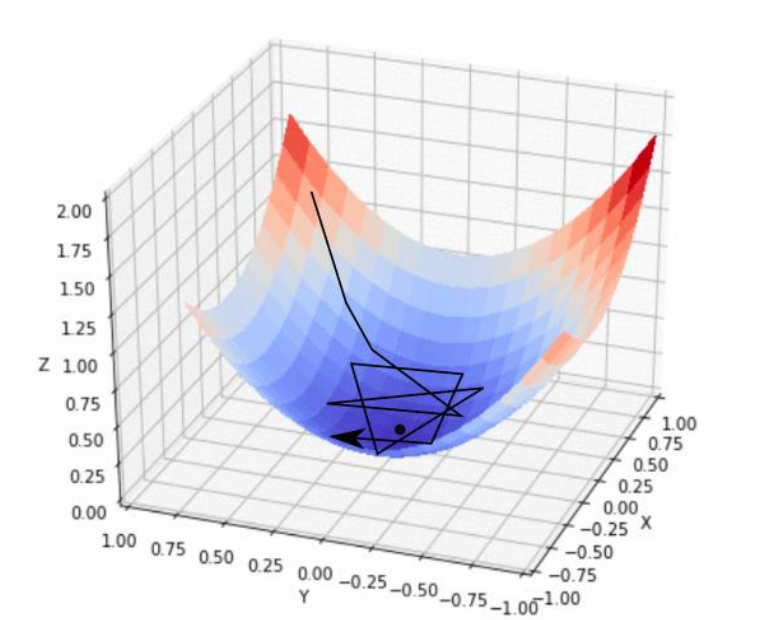
- It won't converge. On each iteration, the learning step may go back and forth due to the noise. Therefore, it wanders around the minimum region but never converges.
- Due to the noise, the learning steps have more oscillations and requires adding learning-decay to decrease the learning rate as we become closer to the minimum.

Challenges with Gradient Descent #1: Local Minima

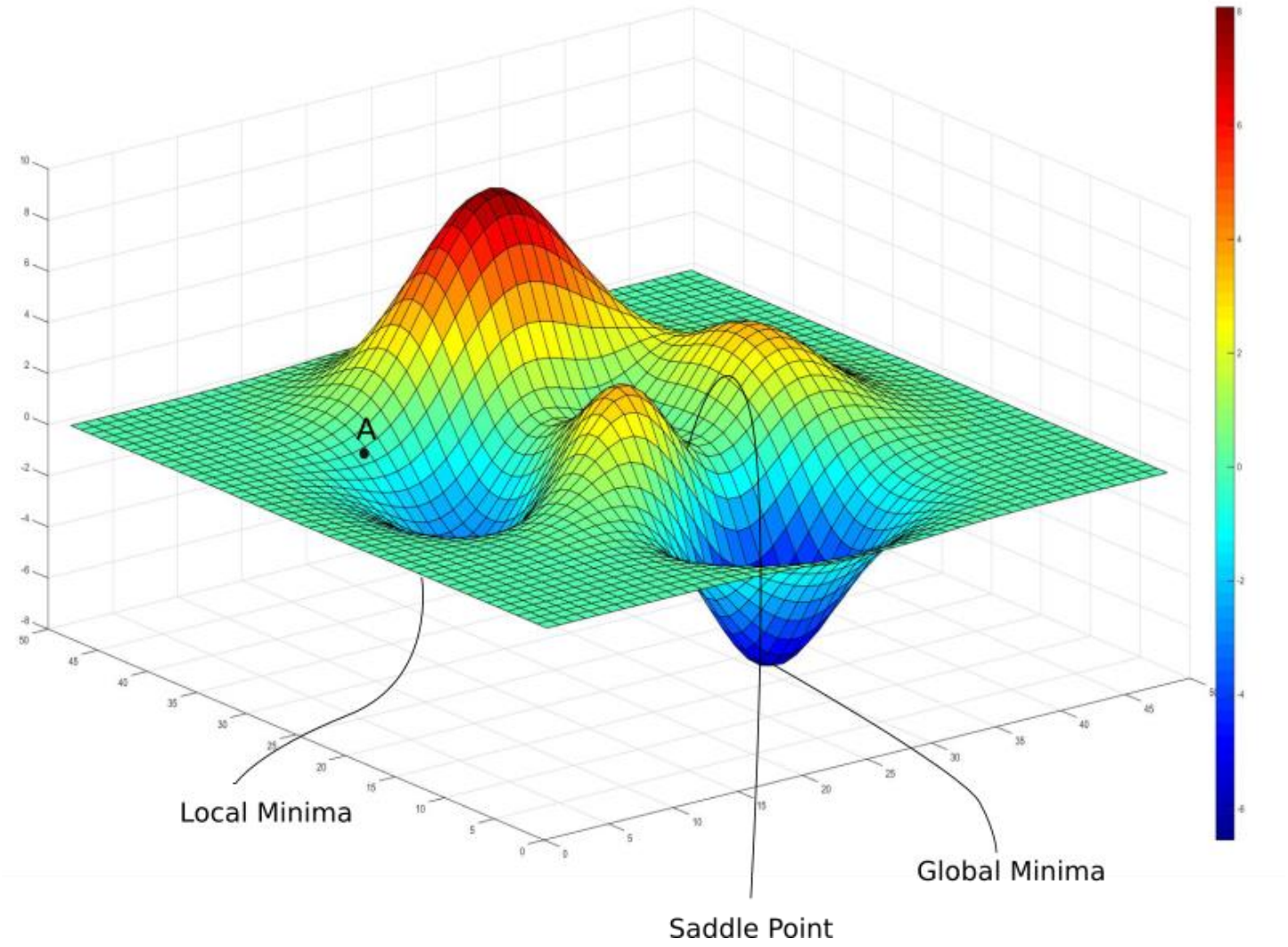








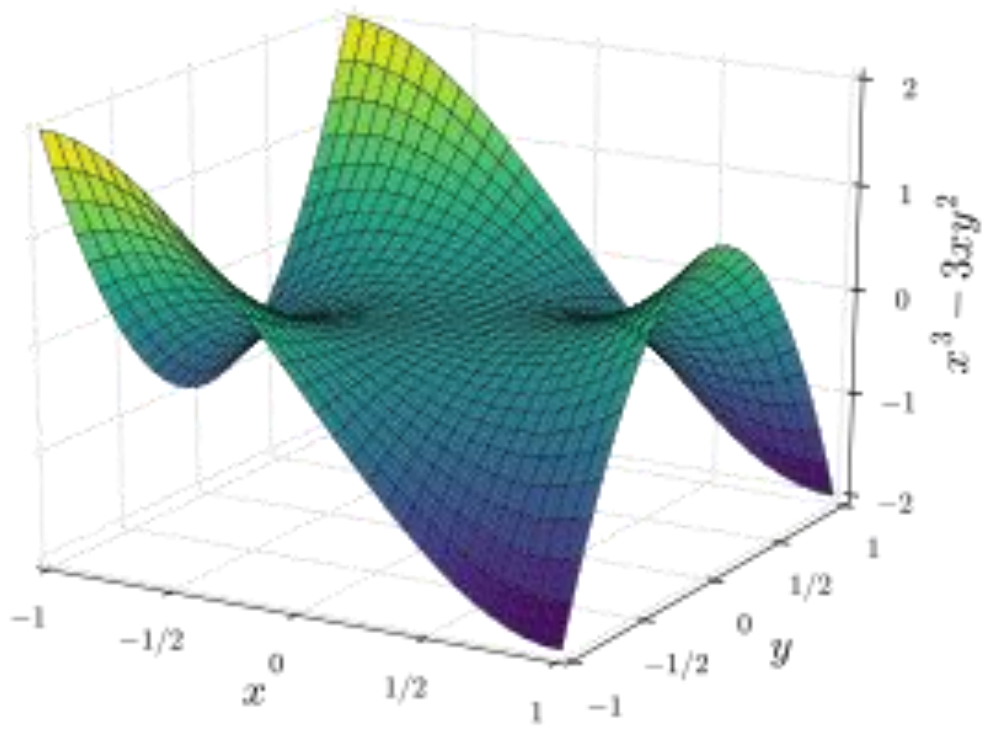
https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/

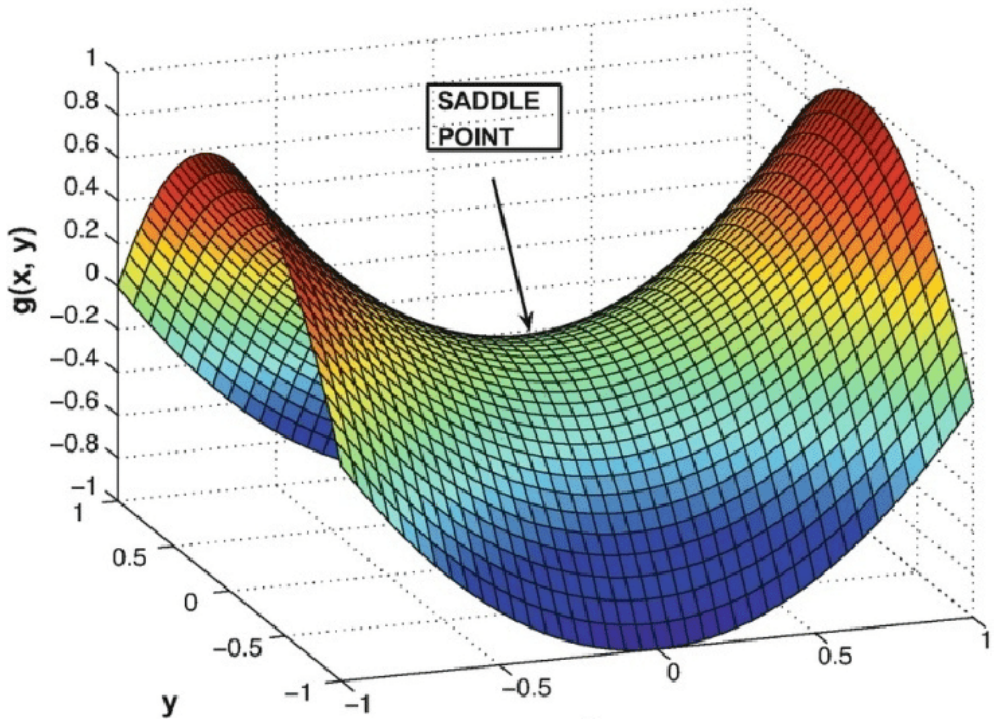


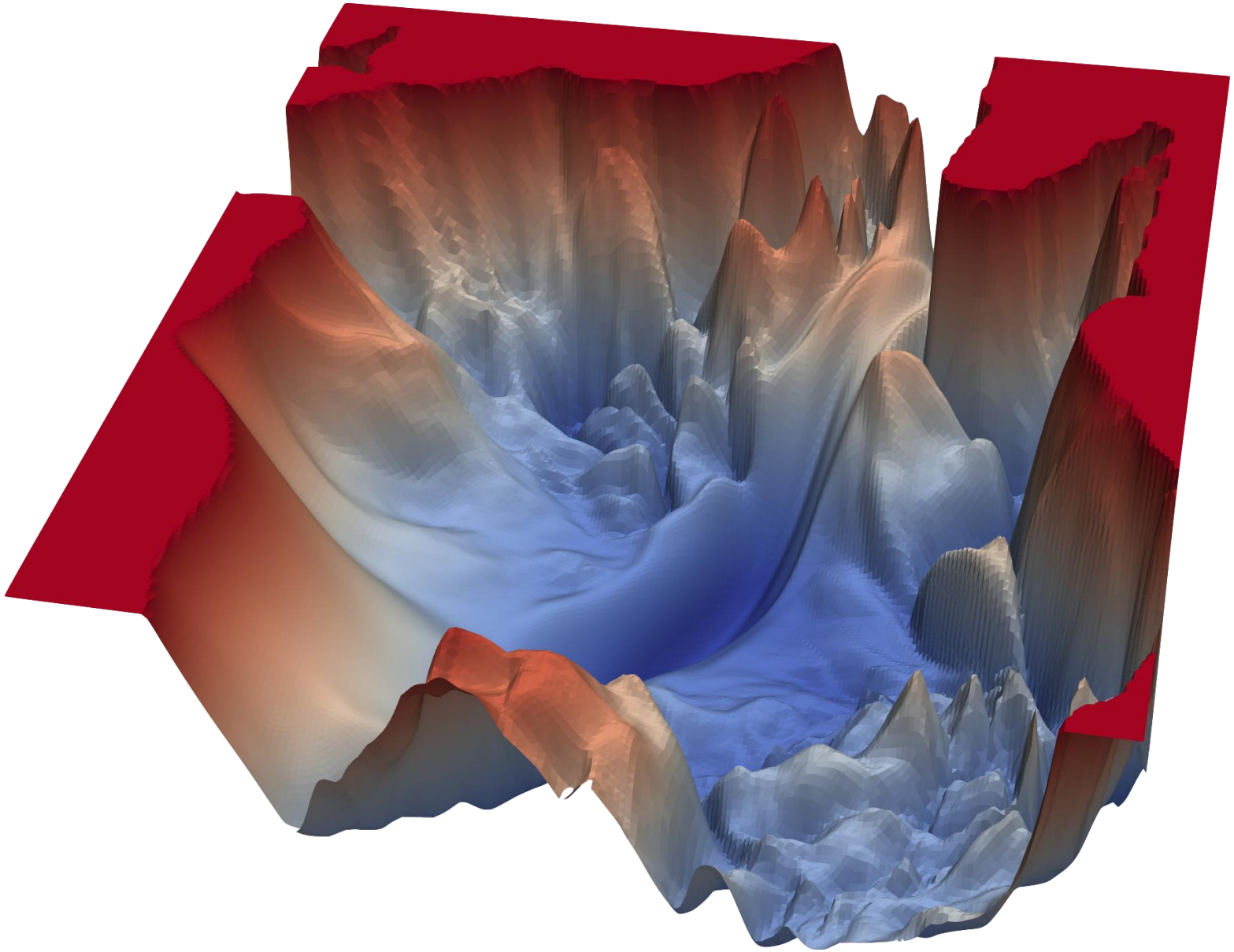
If we initialize your weights at point A, then we're going to converge to the local minima, and there's no way gradient descent will get you out of there, once you converge to the local minima.

Gradient descent is driven by the gradient, which will be zero at the base of any minima.

Challenges with Gradient Descent #2: Saddle Points

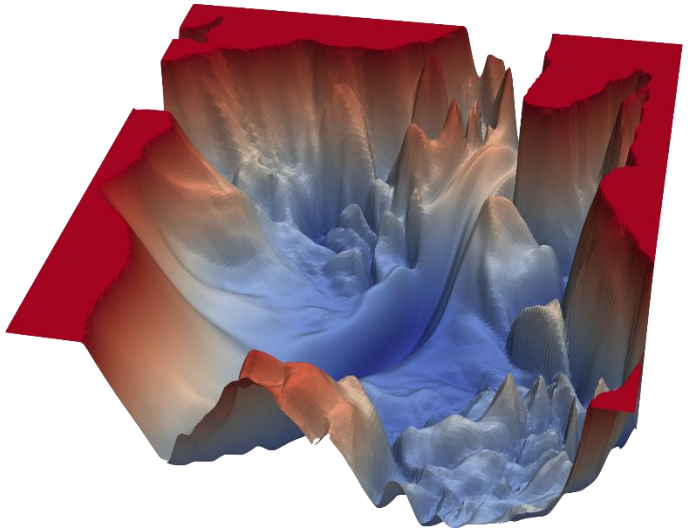




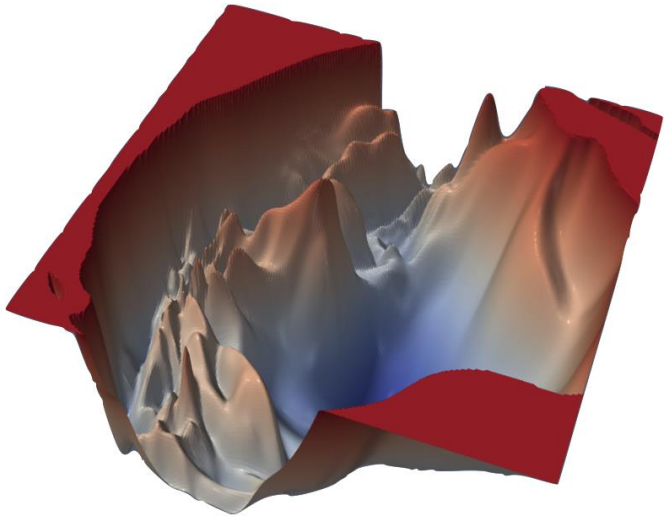


This contour is a constructed 3-D representation for loss contour of a VGG-56 deep network's loss function on the CIFAR-10 dataset.

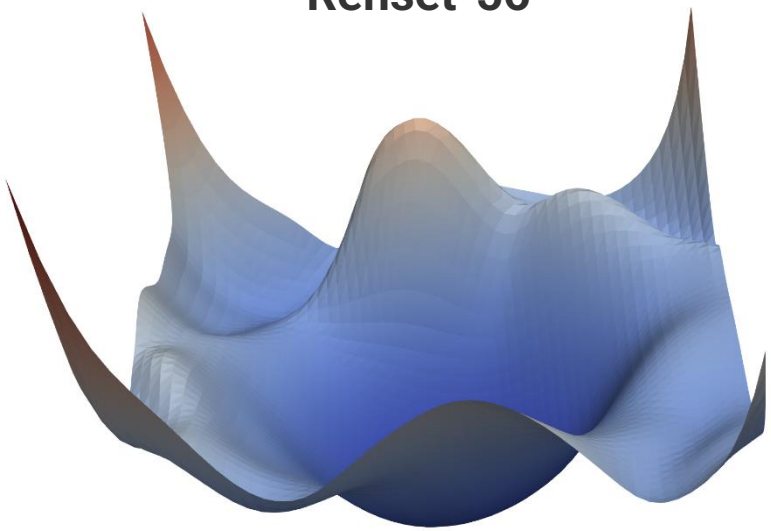
VGG-56



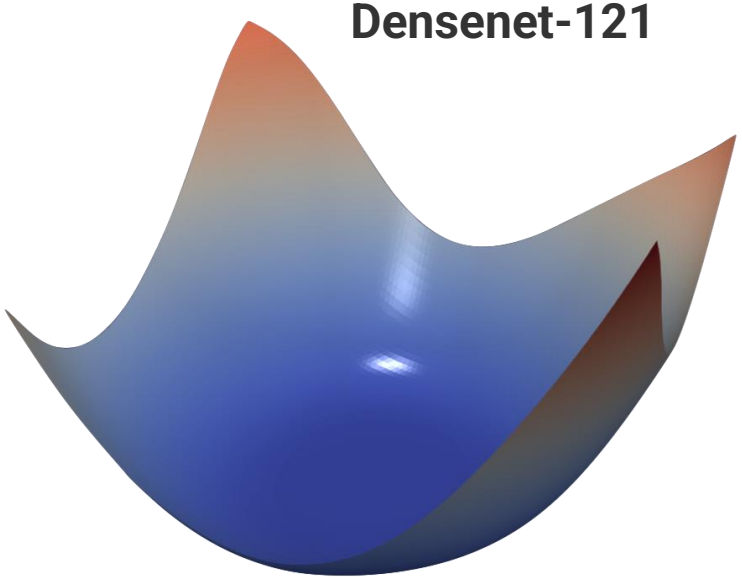
VGG-110



Renset-56



Densenet-121

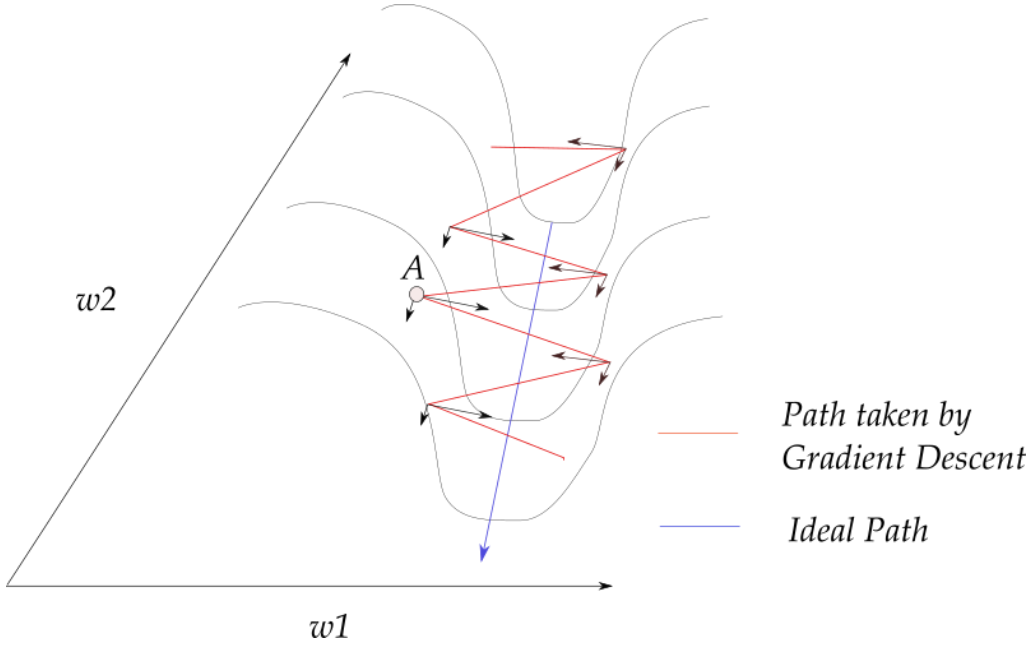


Neural loss functions with and without skip connections. The top row depicts the loss function of a 56-layer and 110-layer net using the CIFAR-10 dataset, without residual connections. The bottom row depicts two skip connection architectures. We have Resnet-56 (identical to VGG-56, except with residual connections), and Densenet (which has a very elaborate set of skip connections). Skip connections cause a dramatic "convexification" of the loss landscape.

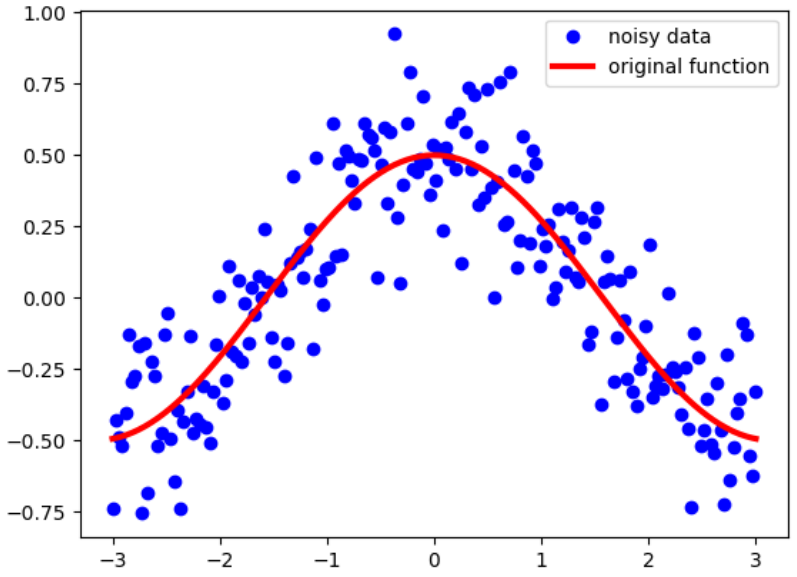
https://www.cs.umd.edu/~tomg/projects/landscapes/

Momentum

- SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another, which are common around local optima.
- Learning with SGD can sometimes be slow.
- The method of momentum is designed to accelerate learning, especially in the face of high curvature, small but consistent gradients, or noisy gradients.



Exponentially weighed averages

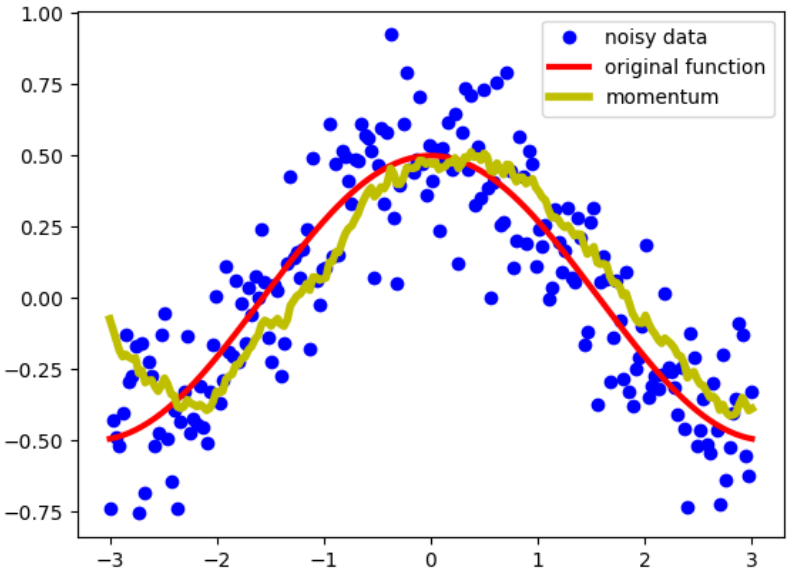


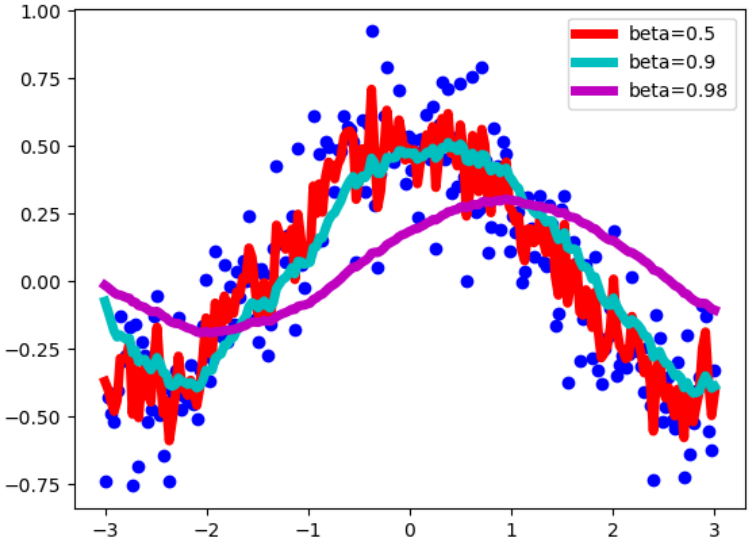
Cosine function with Gaussian noise

$$V_t = \beta v_{t-1} + (1-\beta) S_t$$

$$\beta \in [0,1)$$

$$V_t = \frac{1}{1-\beta^t}$$





https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d

$$W_1 = 0.9 \cdot W_0 + 0.1 \cdot \theta_1$$
$$W_2 = 0.9 \cdot W_1 + 0.1 \cdot \theta_2$$
$$\vdots$$
$$W_t = 0.9 \cdot W_{t-1} + 0.1 \cdot \theta_t$$

$$W_t = \begin{cases} 0 & t = 0 \\ \beta \cdot W_{t-1} + (1-\beta) \cdot \theta_t & t > 0 \end{cases}$$

$$W_t = \underbrace{\beta \cdot W_{t-1}}_{\text{trend}} + \overbrace{(1-\beta) \cdot \theta_t}^{\text{current value}}$$

$$W_3 = \underbrace{0.9 \cdot (0.9\underbrace{(0.9 \cdot 0 + 0.1 \cdot \theta_1)}_{W_1} + 0.1 \cdot \theta_2)}_{W_2} + 0.1 \cdot \theta_3$$

$$W_3 = 0.1(\theta_3 + 0.9\theta_2 + 0.9^2\theta_1)$$

Here it is quite clear what the roll of β = 0.9 parameter is in EWA, we can see older observations are given lower weights. The weights fall exponentially as the data point gets older hence the name exponentially weighted.

https://medium.com/mlearning-ai/exponentially-weighted-average-5eed00181a09

- The momentum algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction.

- Momentum proposes the following tweak to gradient descent. We give gradient descent a short-term memory:

$\theta_k = \theta_{k-1} - \alpha \nabla J(\theta_{k-1})$

$\theta_k = \theta_{k-1} - \alpha v_k$

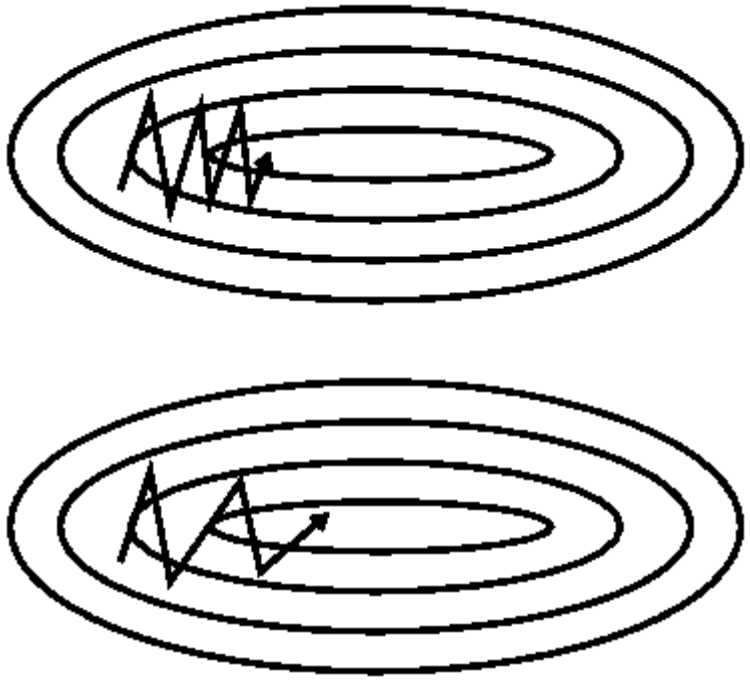
$v_k = \beta v_{k-1} + (1 - \beta) \nabla J(\theta_{k-1})$

- The momentum algorithm introduces a variable v that plays the role of velocity – it is the direction and speed at which the parameters moves through parameter space.

- A hypermeter $\beta \in [0, 1)$ determines how quickly the contributions of previous gradients exponentially decay.

Why momentum works ?

- With Stochastic Gradient Descent we don't compute the exact derivate of our loss function.
- We're estimating it on a small batch. Which means we're not always going in the optimal direction, because our derivatives are 'noisy'.
- Exponentially weighed averages can provide us a better estimate which is closer to the actual derivate than our noisy calculations.
- Ravines are common near local minimas in deep learning and SGD has troubles navigating them.
- SGD will tend to oscillate across the narrow ravine since the negative gradient will point down one of the steep sides rather than along the ravine towards the optimum.
- Momentum helps accelerate gradients in the right direction.



SGD without momentum, SGD with momentum.

AdaGrad

The idea behind Adagrad is to use different learning rates for each parameter base on iteration.

$$\text{SGD} \Rightarrow \; w_t = w_{t-1} - \eta \frac{\partial L}{\partial w_{t-1}}$$

$$\text{Adagrad} \; \Rightarrow w_t = w_{t-1} - \eta_t^{'} \frac{\partial L}{\partial w_{t-1}}$$

$$\text{where } \eta_t^{'} = \frac{\eta}{\sqrt{\alpha_t + \varepsilon}}$$

ε is a small $+$ve number to avoid divisibilty by 0

$$\alpha_t = \sum_{i=1}^{t} \left(\frac{\partial L}{\partial w_{t-1}}\right)^2 \quad \textit{summation of gradient square}$$

In the above Adagrad optimizer equation, the learning rate has been modified in such a way that it will automatically decrease because the summation of the previous gradient square will always keep on increasing after every time step.

https://towardsdatascience.com/deep-learning-optimizers-436171c9e23f

AdaDelta

The idea behind Adadelta is that instead of summing up all the past squared gradients from 1 to “t” time steps, what if we could restrict the window size.

This can be achieved using **Exponentially Weighted Averages** over Gradient**.**

$$\text{Adagrad} \Rightarrow \quad \eta_t^{'} = \frac{\eta}{\sqrt{\alpha_t + \varepsilon}} \quad \text{where} \quad \alpha_t = \sum_{i=1}^{t} \left(\frac{\partial \mathrm{L}}{\partial \mathrm{w}_{t-1}}\right)^2$$

$$\eta_t^{'} = \frac{\eta}{\sqrt{\mathrm{S}_{\mathrm{dw}_t} + \varepsilon}} \quad \text{where} \quad \mathrm{S}_{\mathrm{dw}_t} = \beta S_{\mathrm{dw}_{t-1}} + (1-\beta)\left(\frac{\partial \mathrm{L}}{\partial \mathrm{w}_{t-1}}\right)^2$$

ε is a small $+$ve number to avoid divisibilty by 0

https://towardsdatascience.com/deep-learning-optimizers-436171c9e23f

Adam – Algorithm with adaptive learning rate

- Adam is an adaptive learning rate method, which means, it computes individual learning rates for different parameters.

- Its name is derived from adaptive moment estimation, and the reason it's called that is because Adam uses estimations of first and second moments of gradient to adapt the learning rate for each weight of the neural network.

- To estimates the moments, Adam utilizes exponentially moving averages, computed on the gradient evaluated on a current mini-batch:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$w_t = w_{t-1} - \alpha \frac{m_t}{\sqrt{v_t} + \epsilon}$$

Moving averages of gradient and squared gradient.

- Where m and v are moving averages, g is gradient on current mini-batch, and betas — new introduced hyper-parameters of the algorithm.

https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c

AMSGrad – Algorithm with adaptive learning rate

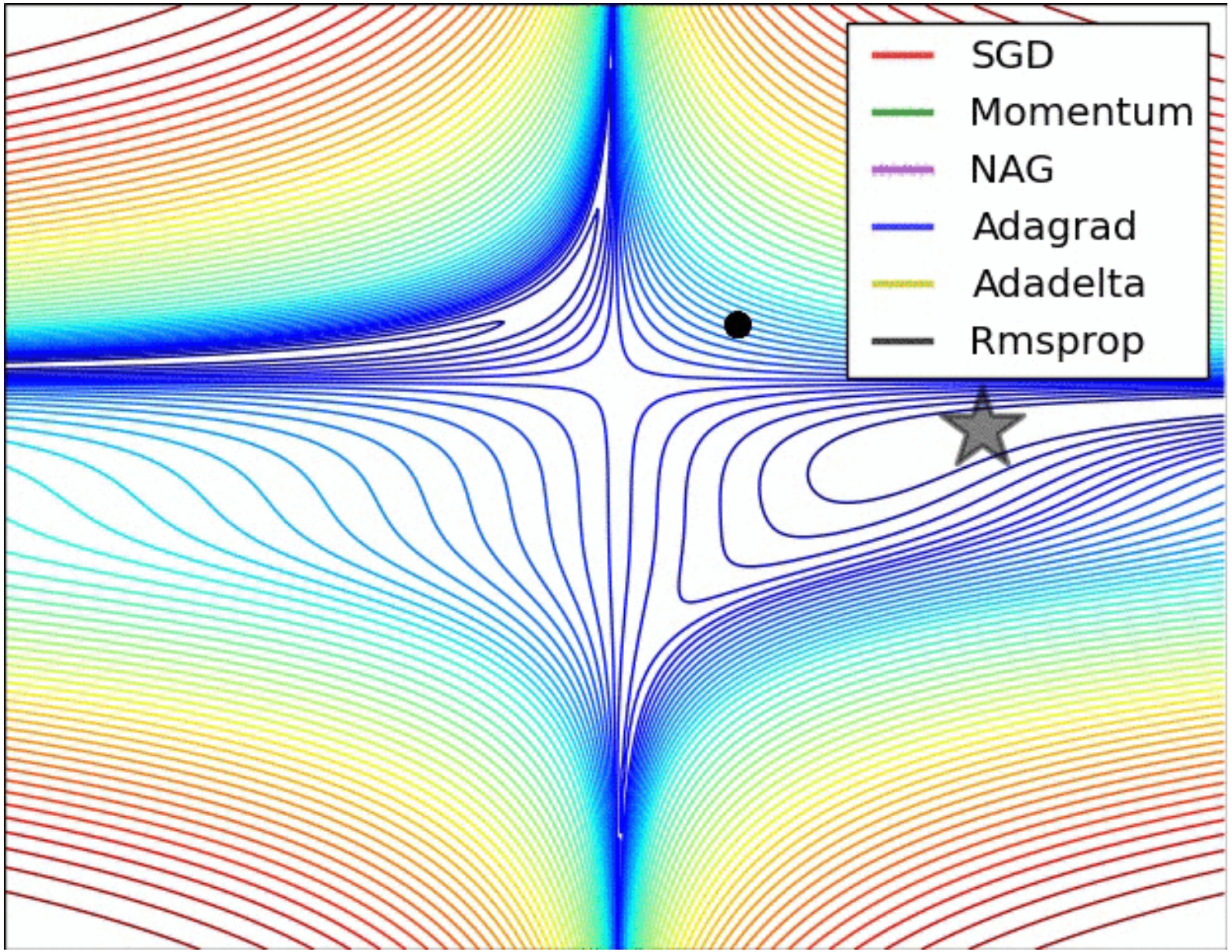
- In some cases, Adam fail to converge to an optimal solution and are outperformed by SGD with momentum.
- Reddi et al. (2018) formalize this issue and pinpoint the exponential moving average of past squared gradients as a reason for the poor generalization behavior of adaptive learning rate methods.

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

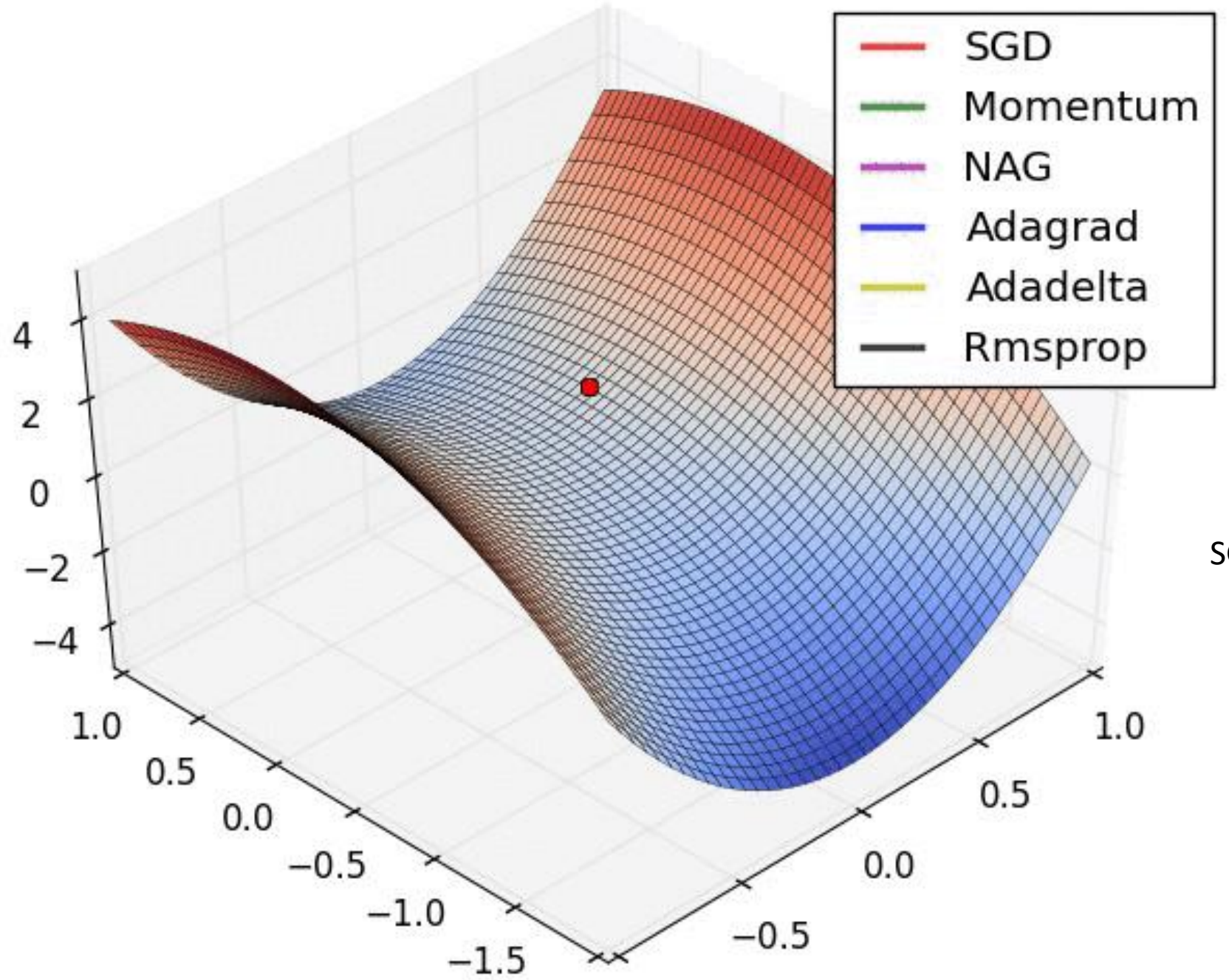
- Instead of using v_t (or its bias-corrected version $\widehat{v_t}$) directly, they employ the previous v_{t-1} if it is larger than the current one:

$$v_t = \max(v_{t\_1}, v_t)$$

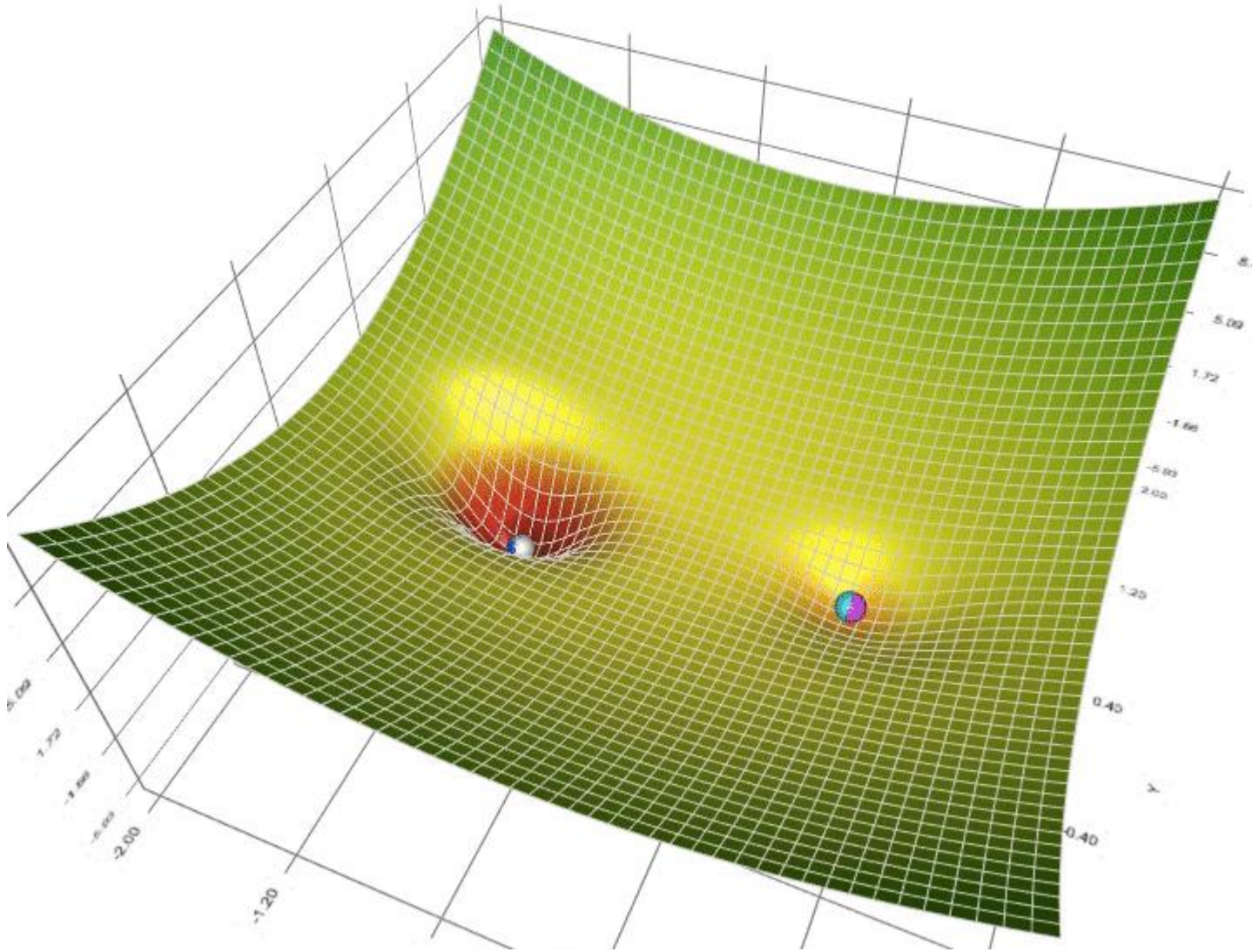
$$w_t = w_{t-1} - \alpha \frac{m_t}{\sqrt{v_t} + \epsilon}$$



optimization on loss surface contours



SGD optimization on saddle point



Animation of 5 gradient descent methods on a surface: gradient descent (cyan), momentum (magenta), AdaGrad (white), RMSProp (green), Adam (blue). Left well is the global minimum; right well is a local minimum.