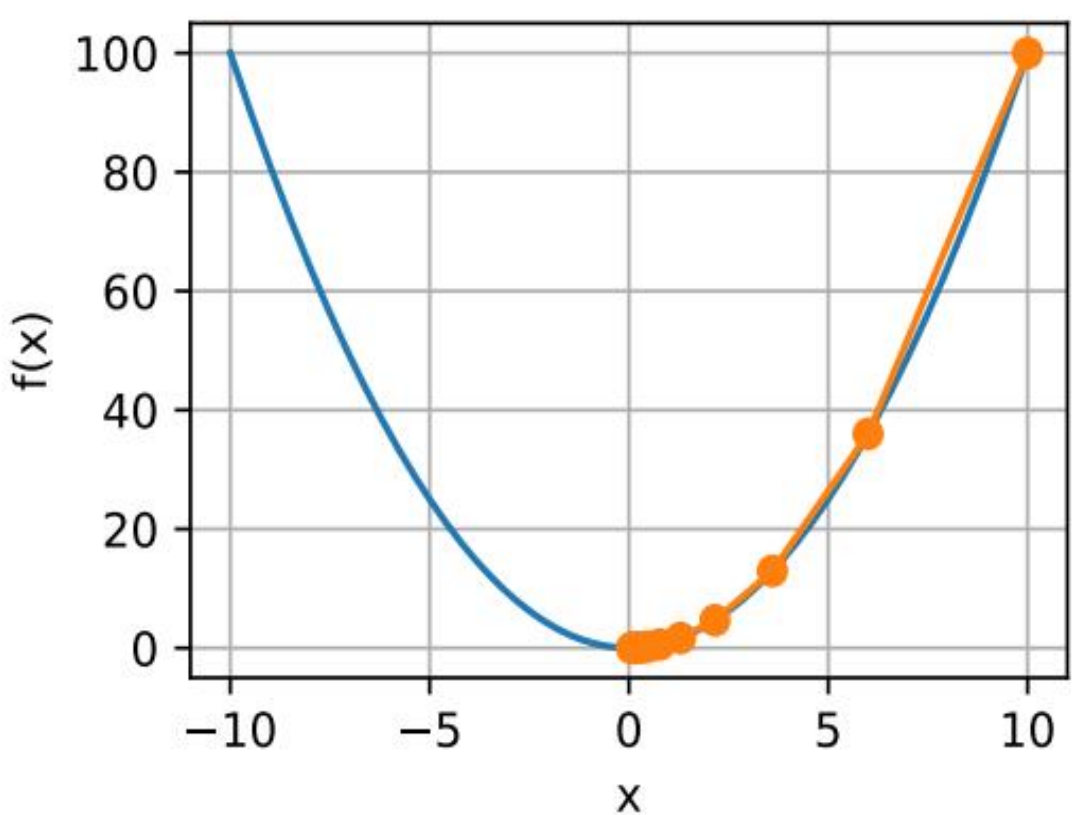
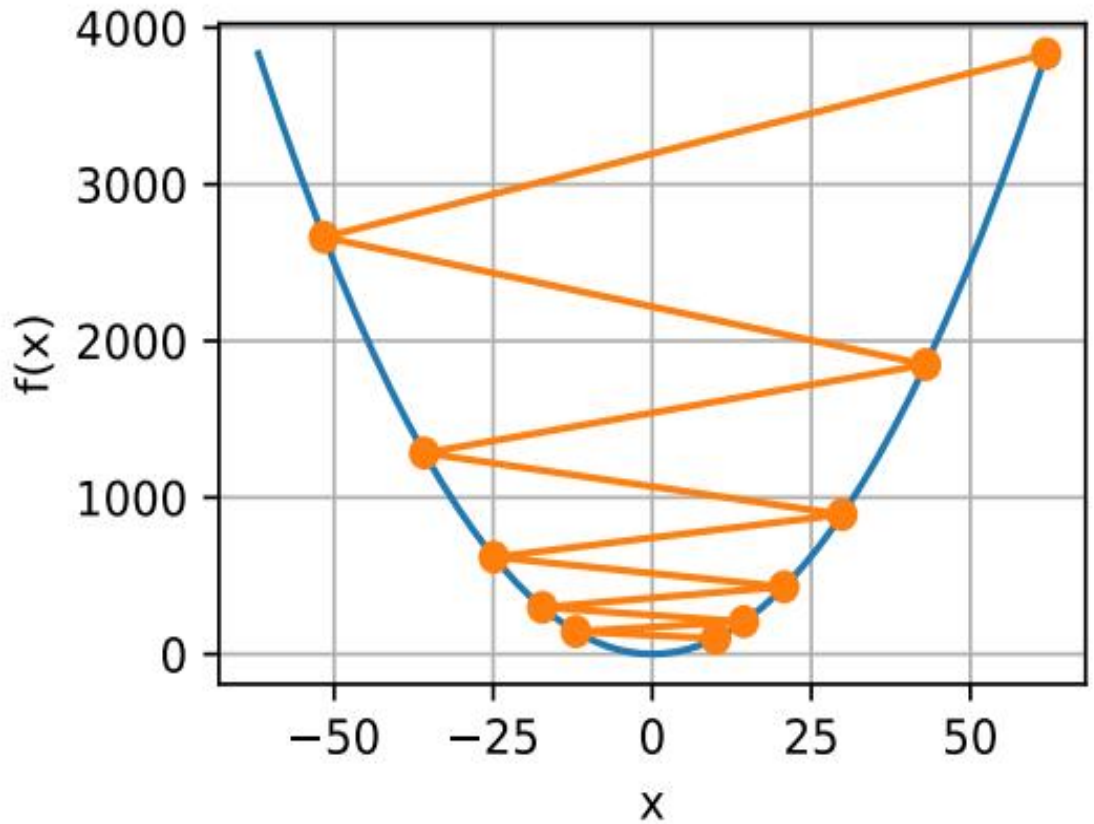
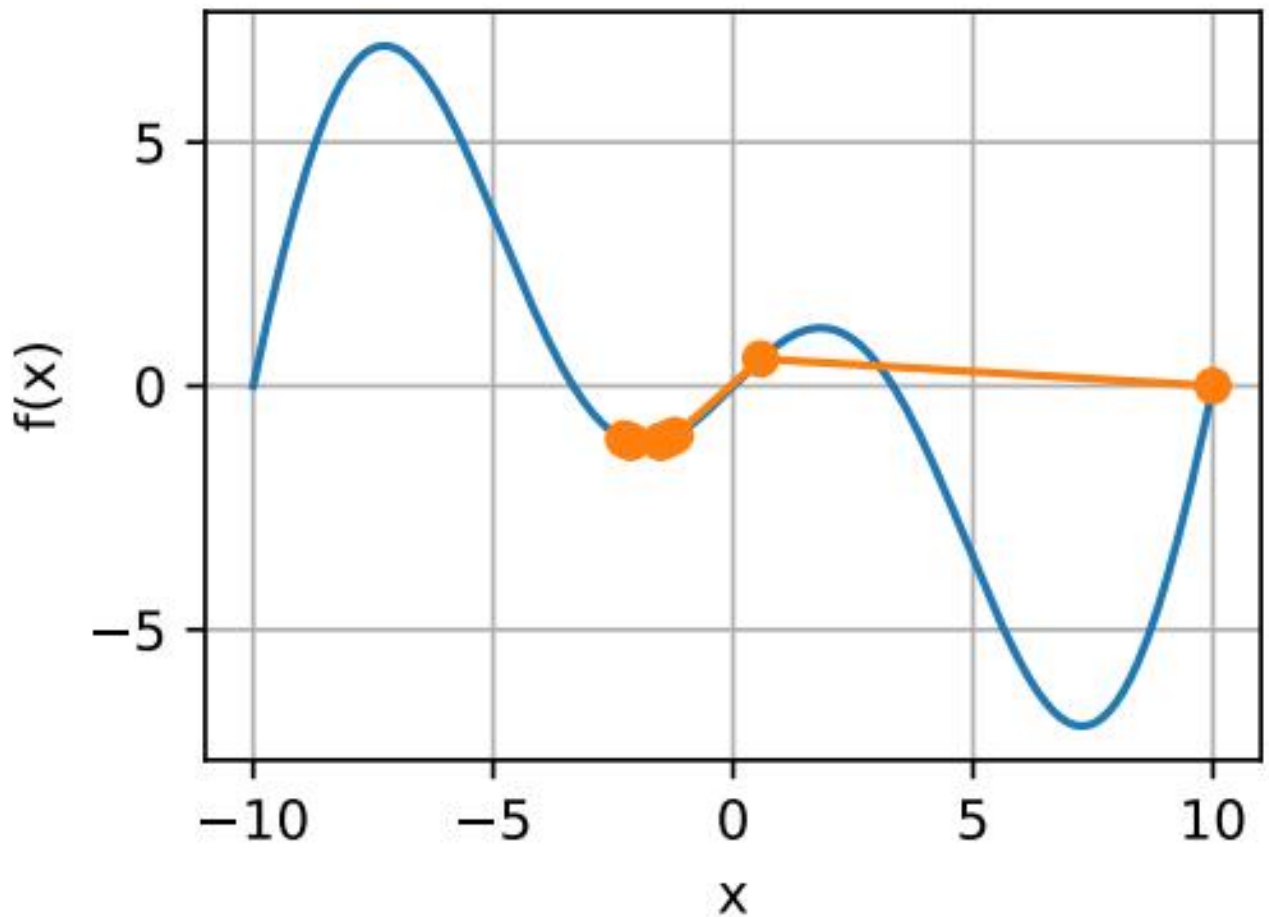
Optimizing Methods


100
80
60
40
20
0
f(x)
−10
−5
0
5
10
x


4000
3000
2000
1000
0
f(x)
−50
−25
0
25
50
x


5
0
−5
f(x)
−10
−5
0
5
10
x

Gradient Descent Challenges

Convergence Issues: Gradient descent may struggle to converge to the optimal solution, particularly if the learning rate is poorly chosen. The algorithm can oscillate around the minimum or take a long time to converge, especially for non-convex functions where there might be multiple local minima.

Choosing the Learning Rate: Selecting an appropriate learning rate (step size) is crucial for the performance of gradient descent. If the learning rate is too small, the algorithm may take a long time to converge; if it's too large, it may overshoot the minimum or fail to converge altogether.

Local Minima and Saddle Points: In high-dimensional spaces, gradient descent can get stuck in local minima or on saddle points, where the gradient is zero but the point is not an optimum. This can hinder progress towards finding the global optimum.

Ill-conditioned or Non-Convex: Gradient descent may struggle with ill-conditioned or highly non-convex functions. In such cases, the gradieznt direction may not reliably lead towards the minimum due to sharp curves or plateaus in the loss landscape.

Gradient Descent Challenges

Sensitivity to Initialization: The performance of gradient descent can be sensitive to the initial parameter values. Poor initialization may lead to slow convergence or convergence to suboptimal solutions.

Computational Cost: For large datasets or complex models, computing gradients for each iteration of gradient descent can be computationally expensive, especially if the model is deep or involves many parameters.

Overfitting: Gradient descent can potentially lead to overfitting if the model is trained for too many iterations or if the learning rate is too high. Overfitting occurs when the model learns noise from the training data instead of generalizing well to unseen data.

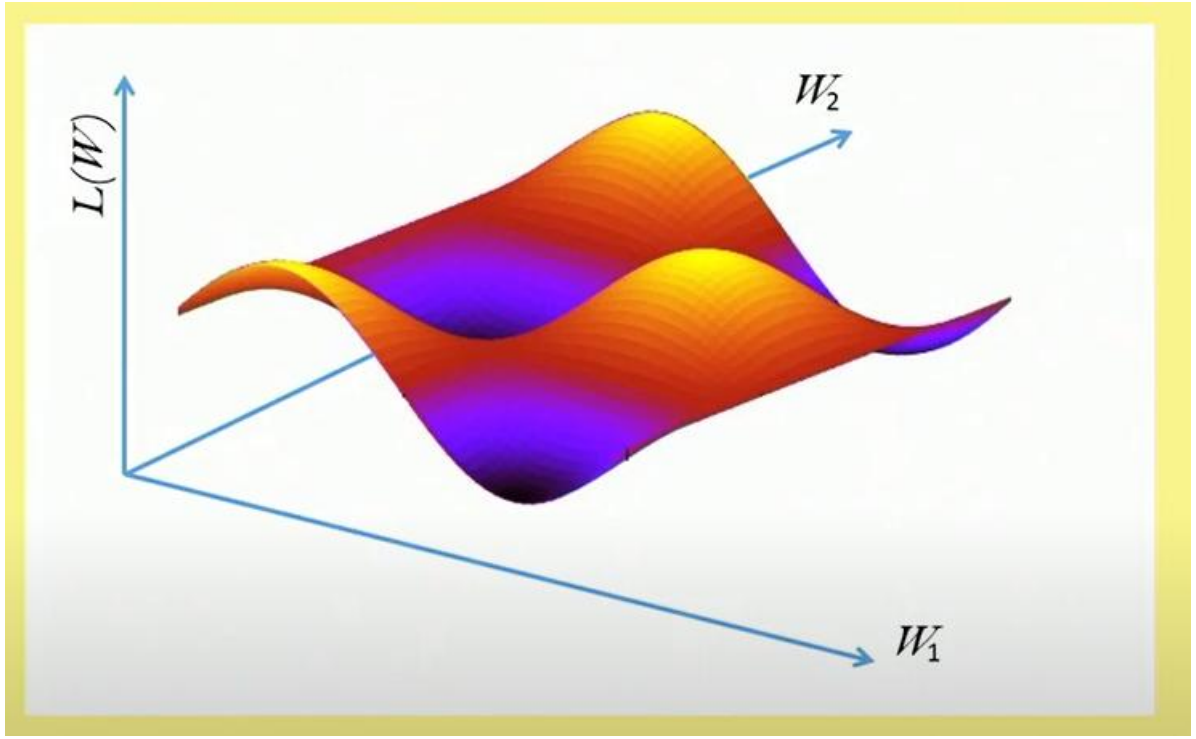
Choosing Optimization Variants: There are different variants of gradient descent (e.g., stochastic gradient descent, mini-batch gradient descent, adaptive learning rate methods like Adam), each with its own hyperparameters and trade-offs. Choosing the most suitable variant for a specific problem can be challenging.

Optimizing Gradient Descent

Momentom Optimizer

The problem with gradient descent is that the weight update at a moment (t) is governed by the learning rate and gradient at that moment only. It doesn't take into account the past steps taken while traversing the cost space.

$$\Delta w(t) = -\eta\delta(t)$$



U = mgh
F = - grad (U)

Types of Gradient Descent

Based on the error in various training models, the Gradient Descent learning algorithm can be divided into

Batch gradient descent
Stochastic gradient descent
Mini-batch gradient descent

1. Batch Gradient Descent:
Batch gradient descent (BGD) is used to find the error for each point in the training set and update the model after evaluating all training examples. This procedure is known as the training epoch. In simple words, it is a greedy approach where we have to sum over all examples for each update.

Advantages of Batch gradient descent:

It produces less noise in comparison to other gradient descent.
It produces stable gradient descent convergence.
It is Computationally efficient as all resources are used for all training samples.

Stochastic gradient descent (SGD) is a type of gradient descent that runs one training example per iteration. Or in other words, it processes a training epoch for each example within a dataset and updates each training example's parameters one at a time. As it requires only one training example at a time, hence it is easier to store in allocated memory. However, it shows some computational efficiency losses in comparison to batch gradient systems as it shows frequent updates that require more detail and speed. Further, due to frequent updates, it is also treated as a noisy gradient. However, sometimes it can be helpful in finding the global minimum and also escaping the local minimum.

Advantages of Stochastic gradient descent:

In Stochastic gradient descent (SGD), learning happens on every example, and it consists of a few advantages over other gradient descent.

It is easier to allocate in desired memory.
It is relatively fast to compute than batch gradient descent.
It is more efficient for large datasets.

3. MiniBatch Gradient Descent:
Mini Batch gradient descent is the combination of both batch gradient descent and stochastic gradient descent. It divides the training datasets into small batch sizes then performs the updates on those batches separately. Splitting training datasets into smaller batches make a balance to maintain the computational efficiency of batch gradient descent and speed of stochastic gradient descent. Hence, we can achieve a special type of gradient descent with higher computational efficiency and less noisy gradient descent.

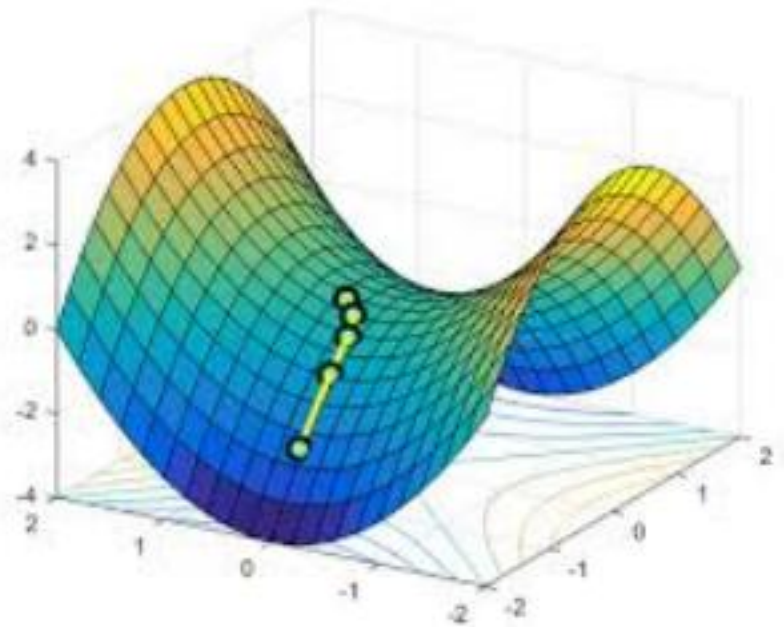
Advantages of Mini Batch gradient descent:

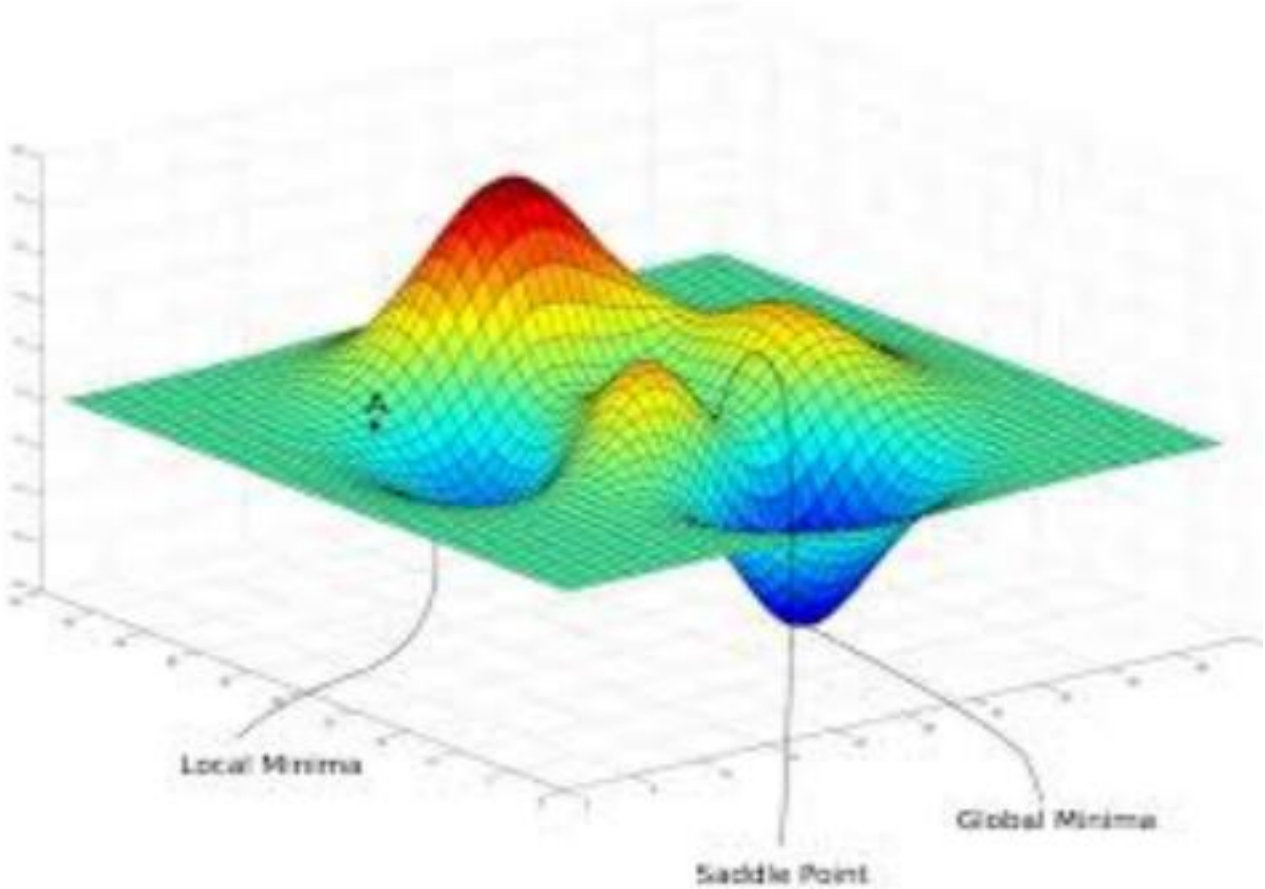
It is easier to fit in allocated memory.
It is computationally efficient.
It produces stable gradient descent convergence.

Challenges with the Gradient Descent

Although we know Gradient Descent is one of the most popular methods for optimization problems, it still also has some challenges. There are a few challenges as follows:

1. Local Minima and Saddle Point

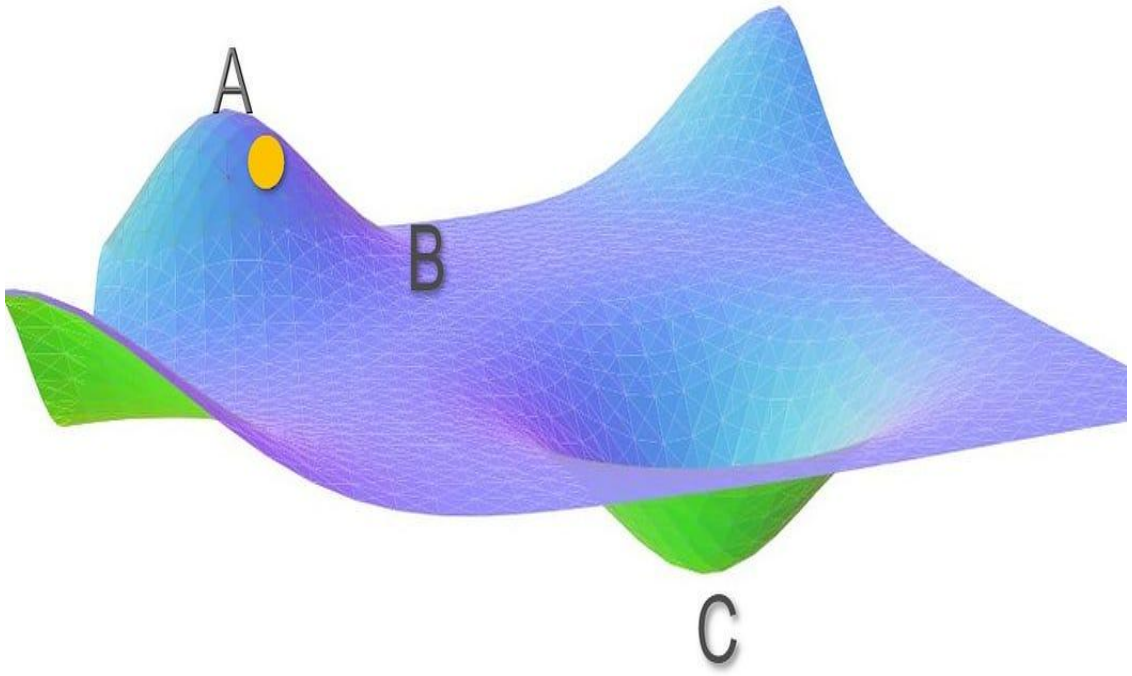




Adding Momentum

Let's assume the initial weights of the network under consideration correspond to point A. With gradient descent, the Loss function decreases rapidly along the slope AB as the gradient along this slope is high. But as soon as it reaches point B the gradient becomes very low. The weight updates around B is very small. Even after many iterations, the cost moves very slowly before getting stuck at a point where the gradient eventually becomes zero.

In this case, ideally, cost should have moved to the global minima point C, but because the gradient disappears at point B, we are stuck with a sub-optimal solution.



How can momentum fix this?

Now, Imagine you have a ball rolling from point A. The ball starts rolling down slowly and gathers some momentum across the slope AB. When the ball reaches point B, it has accumulated enough momentum to push itself across the plateau region B and finally following slope BC to land at the global minima C.

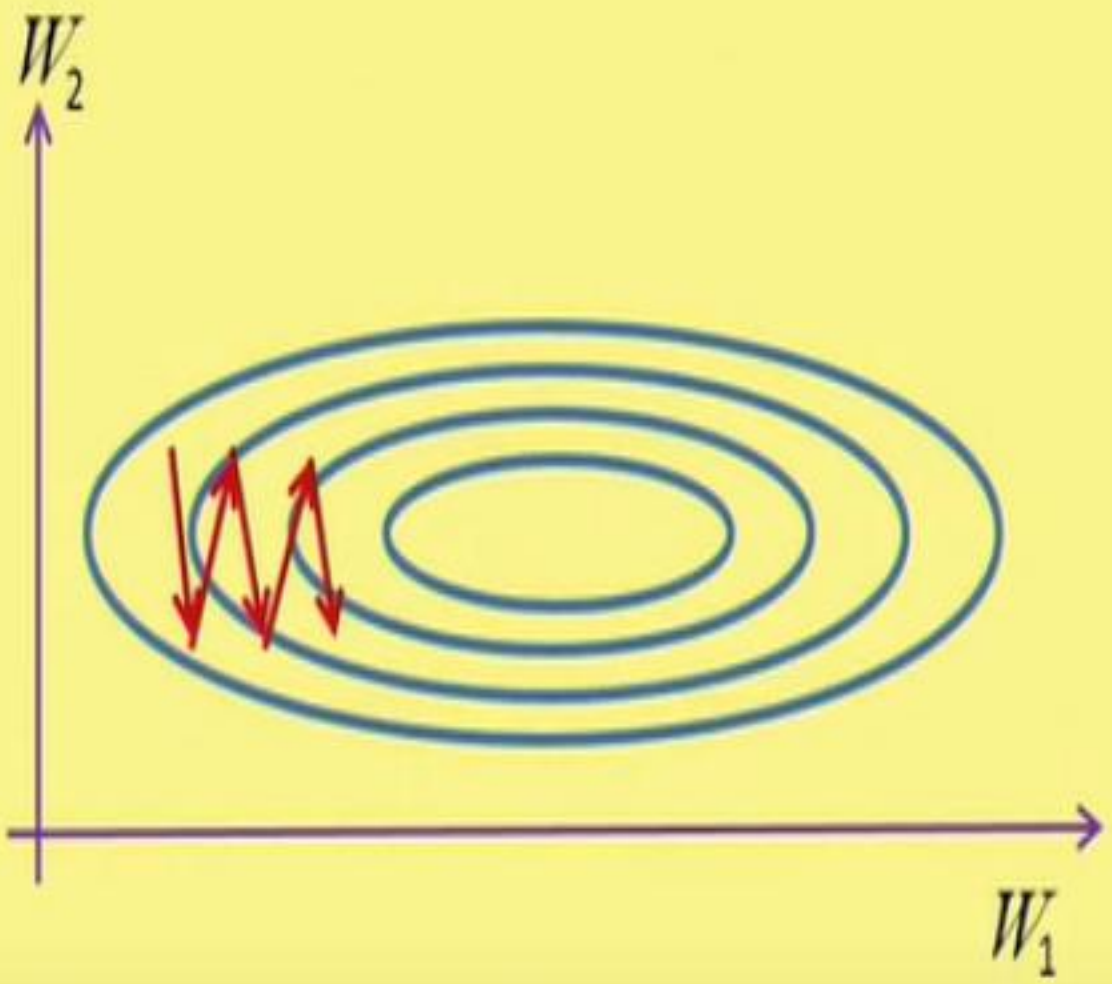
Vanilla SGD

$$w_{t+1} = w_t - \alpha \frac{\partial L}{\partial w_t}$$

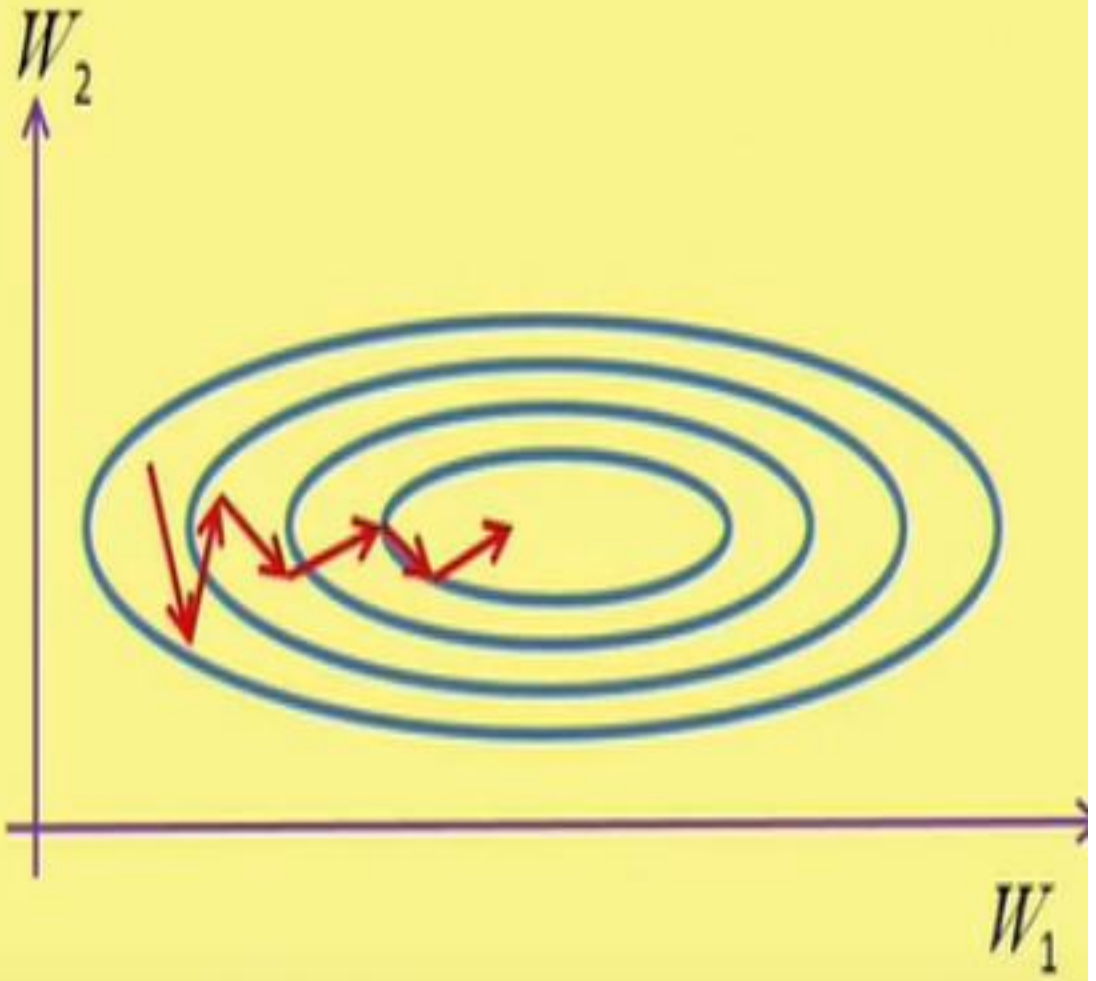
Momentum

$$w_{t+1} = w_t - \alpha V_t$$

$$V_t = \beta V_{t-1} + (1 - \beta) \frac{\partial L}{\partial w_t}$$



SGD



SGD with Momentum

Problem with Momentum Optimizer/NAG

- Both the algorithms require the hyper-parameters to be set manually.
- These hyper-parameters decide the learning rate.
- The algorithm uses same learning rate for all dimensions.
- The high dimensional (mostly) non-nonconvex nature of loss function may lead to different sensitivity on different dimension.
- We may require learning rate be small in some dimension and large in another dimension.

ADAGRAD

Adagrad

- Adagrad adaptively scales the learning rate for different dimensions.
- Scale factor of a parameter is inversely proportional to the square root of sum of historical squared values of the gradient.
- The parameters with the largest partial derivative of the loss will have rapid decrease in their learning rate.
- Parameters with small partial derivatives will have relatively small decrease in learning rate.

Adagrad

$$g_t = \frac{1}{n} \sum_{\forall X \in Minibatch} \nabla_W L(W_t, X) \qquad r_t = \sum_{\tau=1}^{t} g_\tau \circ g_\tau$$

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{\in I + r_t}} \circ g_t$$

$$\circ \rightarrow \text{element - wise product}$$

Adagrad

$$\begin{bmatrix} W_{t+1}^{(1)} \\ W_{t+1}^{(2)} \\ \vdots \\ W_{t+1}^{(d)} \end{bmatrix} = \begin{bmatrix} W_{t}^{(1)} \\ W_{t}^{(2)} \\ \vdots \\ W_{t}^{(d)} \end{bmatrix} - \begin{bmatrix} \frac{\eta}{\sqrt{\in + r_t^{(1)}}} \cdot g_t^{(1)} \\ \frac{\eta}{\sqrt{\in + r_t^{(2)}}} \cdot g_t^{(2)} \\ \vdots \\ \frac{\eta}{\sqrt{\in + r_t^{(d)}}} \cdot g_t^{(d)} \end{bmatrix}$$

Adagrad

Positive Side:

- Adagrad adaptively scales the learning rate for different dimensions by normalizing with respect to the gradient magnitude in the corresponding dimension.
- Adagrad eliminates the need to manually tune the learning rate.
- Reduces learning rate faster for parameters showing large slope and slower for parameters giving smaller slope.
- Adagrad converges rapidly when applied to convex functions.

Adagrad

Negative side:

- If the function is non-convex:- trajectory may pass through many complex terrains eventually arriving at a locally region.
- By then learning rate may become too small due to the accumulation of gradients from the beginning of training.
- So at some point the model may stop learning.

RMS Prop

Adagrad

$$g_t = \frac{1}{n} \sum_{\forall X \in Minibatch} \nabla_W L(W_t, X) \qquad r_t = \sum_{\tau=1}^{t} g_\tau \circ g_\tau$$

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{\in I + r_t}} \circ g_t$$

$\circ \rightarrow$ element - wise product

RMSProp

- RMSProp uses exponentially decaying average of squared gradient and discards history from the extreme past.
- Converges rapidly once it finds a locally convex bowl.
- Treats this as an instance of Adagrad algorithm initialized within that bowl.

RMSProp

$$r_t = \sum_{\tau=1}^{t} g_\tau \circ g_\tau$$

ADAGRAD

$$g_t = \frac{1}{n} \sum_{\forall X \in Minibatch} \nabla_W L(W_t, X)$$

$$r_t = \beta r_{t-1} + (1-\beta) g_t \circ g_t$$ ➡ Exponentially decaying average

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{\in I + r_t}} \circ g_t$$

ADAM (Adaptive Moment)

Adam

- Variant of the combination of RMSProp and Momentum.
- Incorporates first order moment (with exponential weighting) of the gradient (Momentum term).
- Momentum is incorporated in RMSProp by adding momentum to the rescaled gradients.
- Both first and second moments are corrected for bias to account for heir initialization to zero.

Adam

$$g_t = \frac{1}{n} \sum_{\forall X \in Minibatch} \nabla_W L(W_t, X)$$

Biased first and second moments

$$s_t = \beta_1 s_{t-1} + (1 - \beta_1) g_t$$

$$r_t = \beta_2 r_{t-1} + (1 - \beta_2) g_t \circ g_t$$

Adam

Bias corrected first and second moments

$$\hat{s}_t = \frac{s_t}{1-\beta_1} \qquad \hat{r}_t = \frac{r_t}{1-\beta_2}$$

$$W_{t+1} = W_t - \eta \frac{\hat{s}_t}{\sqrt{\in I + \hat{r}_t}}$$