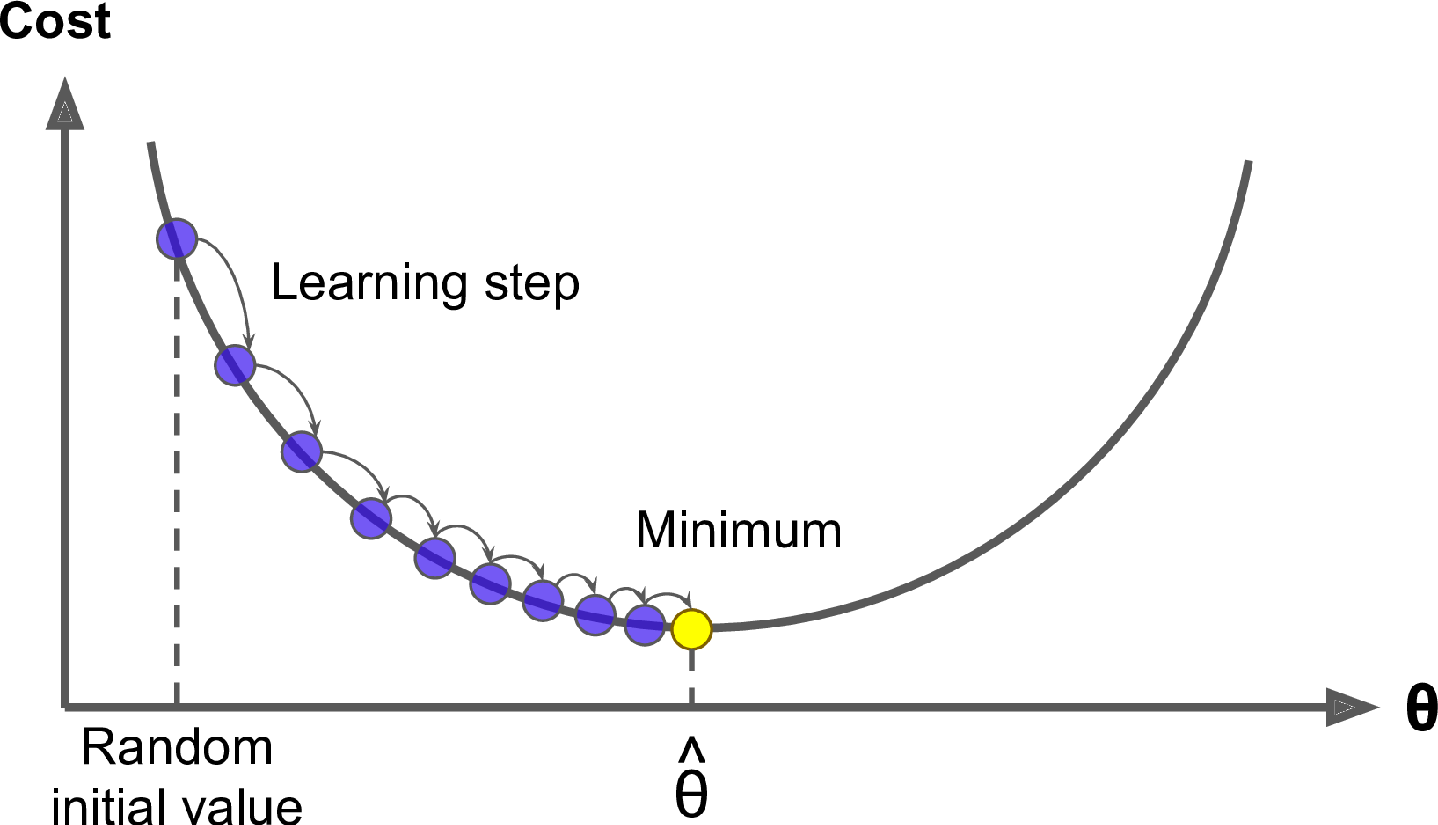
**1.Explain other types of Gradient Descent** .

Gradient Descent is a generic optimization algorithm capable of finding optimal solu‐ tions to a wide range of problems. The general idea of Gradient Descent is to tweak parameters iteratively in order to minimize a cost function. Suppose you are lost in the mountains in a dense fog, and you can only feel the slope of the ground below your feet. A good strategy to get to the bottom of the valley quickly is to go downhill in the direction of the steepest slope. This is exactly what Gradient Descent does: it measures the local gradient of the error function with regard to the parameter vector θ, and



it goes in the direction of descending gradient. Once the gradient is zero, you have reached a

minimum.

**Batch gradient descent**

* Batch gradient descent sums the error for each point in a training set, updating the model only after all training examples have been evaluated. This process referred to as a training epoch.
* While this batching provides computation efficiency, it can still have a long processing time for large training datasets as it still needs to store all of the data into memory. Batch gradient descent also usually produces a stable error gradient and convergence, but sometimes that convergence point isn’t the most ideal, finding the local minimum versus the global one

Stochastic gradient descent

* Stochastic gradient descent (SGD) runs a training epoch for each example within the dataset and it updates each training example's parameters one at a time. Since you only need to hold one training example, they are easier to store in memory. While these frequent updates can offer more detail and speed, it can result in losses in computational efficiency when compared to batch gradient descent.
* Its frequent updates can result in noisy gradients, but this can also be helpful in escaping the local minimum and finding the global one.

**Mini-batch gradient descent**

Mini-batch gradient descent combines concepts from both batch gradient descent and stochastic gradient descent. It splits the training dataset into small batch sizes and performs updates on each of those batches. This approach strikes a balance between the computational efficiency of batch gradient descent and the speed of stochastic gradient descent.

**2. Problem with gradient descent**

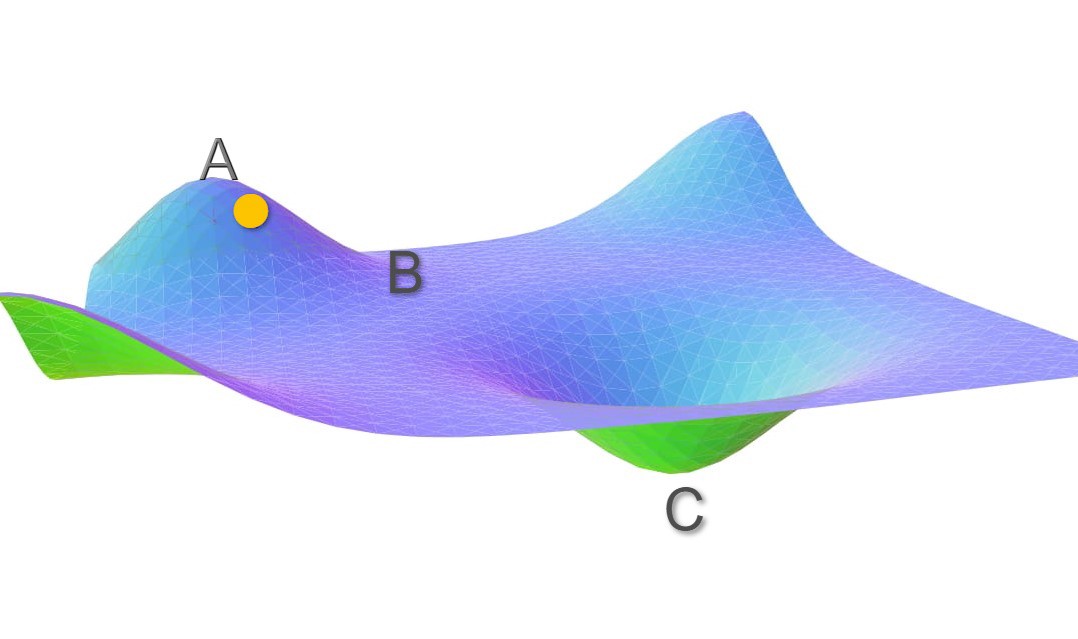
The problem with gradient descent is that the weight update at a moment (t) is governed by the learning rate and gradient at that moment only. It doesn’t take into account the past steps taken while traversing the cost space.

https://miro.medium.com/max/490/1*_2yTRTRx7Gl9ykYP4kVc6g.png

It leads to the following problems.

1. The gradient of the cost function at saddle points( plateau) is negligible or zero, which in turn leads to small or no weight updates. Hence, the network becomes stagnant, and learning stops
2. The path followed by Gradient Descent is very jittery even when operating with mini-batch mode

Consider the below cost surface.



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.

**3.how to handle these problems?**

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.

**How can this be used and applied to Gradient Descent?**

To account for the **momentum**, we can use a moving average over the past gradients. In regions where the gradient is high like AB, weight updates will be large. Thus, in a way we are gathering **momentum** by taking a moving average over these gradients. But there is a problem with this method, it considers all the gradients over iterations with equal weightage. The gradient at t=0 has equal weightage as that of the gradient at current iteration t. We need to use some sort of weighted average of the past gradients such that the recent gradients are given more weightage.

This can be done by using an Exponential Moving Average(EMA). An exponential moving average is a moving average that assigns a greater weight on the most recent values.

The EMA for a series Y may be calculated recursively

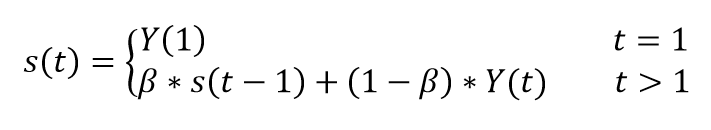


Image by author

where

* The coefficient β represents the degree of weighting increase, a constant smoothing factor between 0 and 1. A lower β discounts older observations faster.
* Y(t) is the value at a period t.
* S(t) is the value of the EMA at any period t.

In our case of a sequence of gradients, the new weight update equation at iteration t becomes

https://miro.medium.com/max/1390/1*pOIJGGuMtDcQDK3ZGFAauQ.png

Let's break it down.

𝓥**(t)**: is the new weight update done at iteration **t**

**β:**Momentum constant

**𝛿(t):**is the gradient at iteration **t**

Assume the weight update at the zeroth iteration t=0 is zero

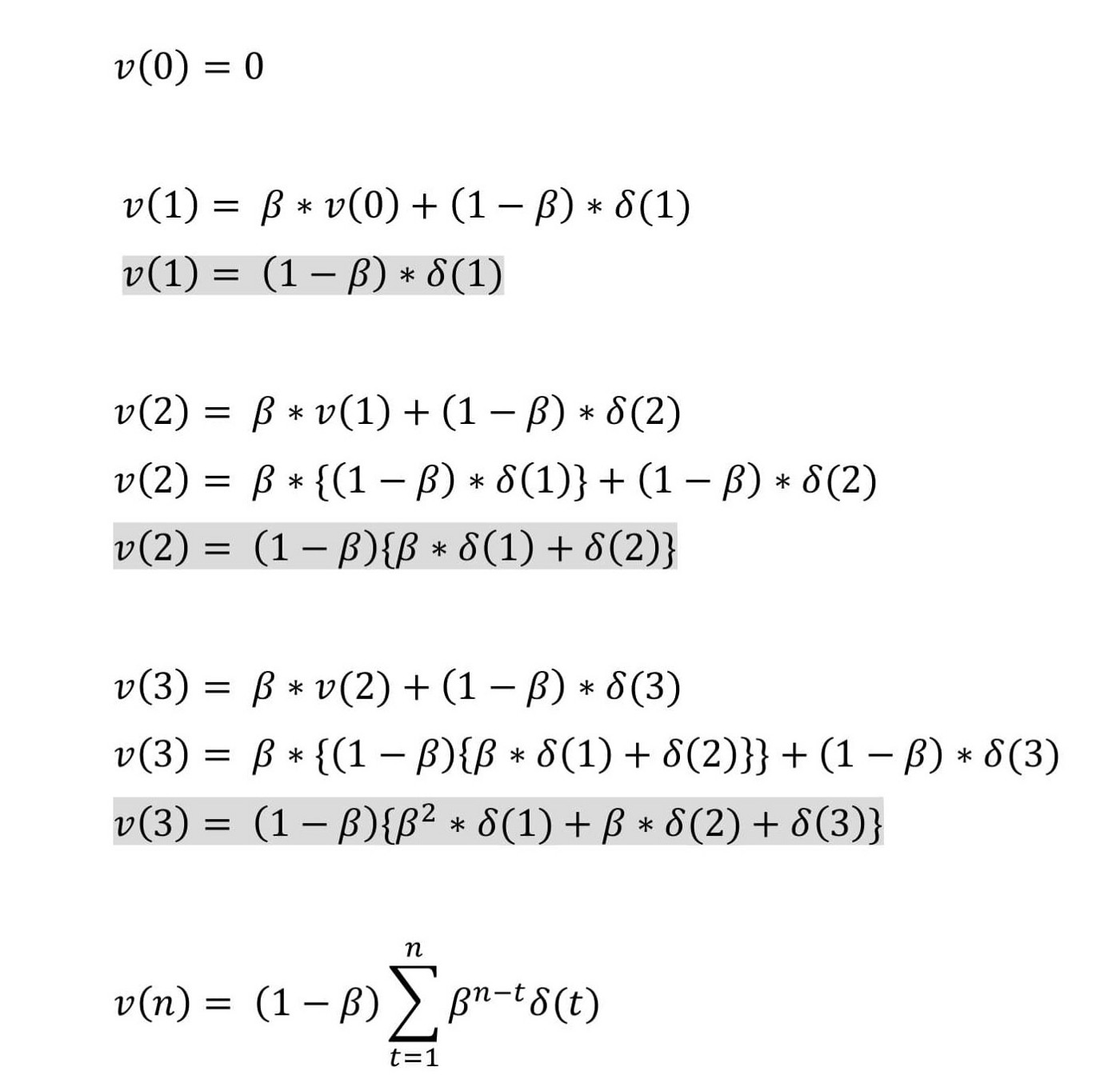


Image by author

Think about the constant β and ignore the term (1-β) in the above equation.

**Note**: In many texts, you might find (1-β) replaced with η the learning rate.

what if β is 0.1?

At n=3; the gradient at t =3 will contribute 100% of its value, the gradient at t=2 will contribute 10% of its value, and gradient at t=1 will only contribute 1% of its value.

here contribution from earlier gradients decreases rapidly.

what if β is 0.9?

At n=3; the gradient at t =3 will contribute 100% of its value, t=2 will contribute 90% of its value, and gradient at t=1 will contribute 81% of its value.

From above, we can deduce that higher β will accommodate more gradients from the past. Hence, generally, β is kept around 0.9 in most of the cases.

**Note**: The actual contribution of each gradient in the weight update will be further subjected to the learning rate.

This addresses our first point where we said when the gradient at the current moment is negligible or zero the learning becomes zero. Using **momentum** with gradient descent, gradients from the past will push the cost further to move around a saddle point.

In the cost surface shown earlier let's zoom into point C.

With gradient descent, if the learning rate is too small, the weights will be updated very slowly hence convergence takes a lot of time even when the gradient is high. This is shown in the left side image below. If the learning rate is too high cost oscillates around the minima as shown in the right side image below.

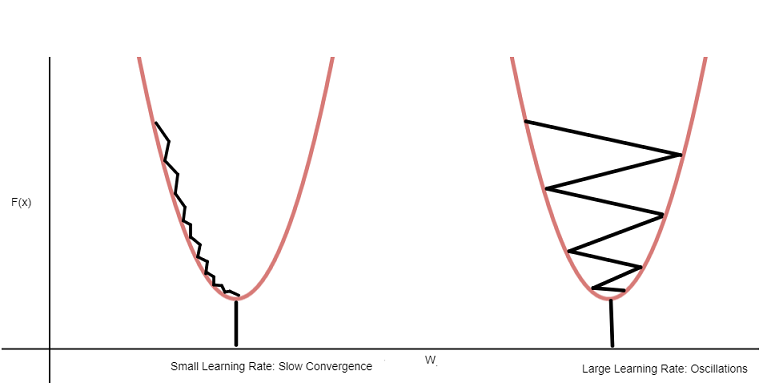


Image by author

**How does Momentum fix this?**

Let's look at the last summation equation of the **momentum** again.

**Case 1**: When all the past gradients have the same sign

The summation term will become large and we will take large steps while updating the weights. Along the curve BC, even if the learning rate is low, all the gradients along the curve will have the same direction(sign) thus increasing the **momentum** and accelerating the descent.

**Case 2**: when some of the gradients have +ve sign whereas others have -ve

The summation term will become small and weight updates will be small. If the learning rate is high, the gradient at each iteration around the valley C will alter its sign between +ve and -ve and after few oscillations, the sum of past gradients will become small. Thus, making small updates in the weights from there on and damping the oscillations.

This to some amount addresses our second problem. **Gradient Descent with Momentum** takes small steps in directions where the gradients oscillate and take large steps along the direction where the past gradients have the same direction(same sign).

# **Conclusion**

By adding a **momentum** term in the gradient descent, gradients accumulated from past iterations will push the cost further to move around a saddle point even when the current gradient is negligible or zero.

Even though **momentum**with gradient descent converges better and faster, it still doesn’t resolve all the problems. First, the hyperparameter η (learning rate) has to be tuned manually. Second, in some cases, where, even if the learning rate is low, the momentum term and the current gradient can alone drive and cause oscillations.

First, the Learning rate problem can be further resolved by using other variations of Gradient Descent like **AdaptiveGradient**and **RMSprop.**Second, a large **momentum** problem can be further resolved by using a variation of **momentum-based gradient descent**called **Nesterov Accelerated Gradient Descent**.

**4. Limitation of Ridge and lasso Regression**

**Limitation of Ridge Regression**

* Limitation of Ridge Regression
* It includes all the predictors in the final model.
* It is not capable of performing feature selection.
* It shrinks coefficients towards zero.
* It trades variance for bias

**Limitation of lasso Regression**

* Due to feature restriction/ignorance, the technique can ignore important features resulting in a useless model
* Setting alpha too low may remove the effect of regularization and result in underfitting
* Elastic Net regularization may work better since it combines the penalties of both the Lasso and Ridge
* Lasso can behave unpredictably, especially when the number of features is more than the number of training instances or when features are deeply correlated.

**5. What is the elastic net?**

The **elastic net** algorithm uses a weighted combination of L1 and L2 regularization. As you can probably see, the same function is used for LASSO and Ridge regression with only the L1\_wt argument changing. This argument determines how much weight goes to the L1-norm of the partial slopes. If the regularization is pure L2 (Ridge) and if L1\_wt = 1.0 the regularization is pure L1 (LASSO).