HANDS-ON TUTORIALS, INTUITIVE DEEP LEARNING

Neural Network Optimizers Made Simple: Core algorithms and why they are needed

A Gentle Guide to fundamental techniques used by gradient descent optimizers like SGD, Momentum, RMSProp, Adam, and others, in plain English

Ketan Doshi · Follow

Published in Towards Data Science · 11 min read · Apr 8, 2021

Photo by George Stackpole on Unsplash

Optimizers are a critical component of a Neural Network architecture. During training, they play a key role in helping the network learn to make better and better predictions.

They do this by finding the optimal set of model parameters like weights and biases so that the model can produce the best outputs for the problem they're solving.

The most common optimization technique used by most neural networks is gradient descent.

Most popular deep learning libraries, such as Pytorch and Keras, have a plethora of built-in optimizers based on gradient descent eg. Stochastic Gradient Descent (SGD), Adadelta, Adagrad, RMSProp, Adam, and so on.

Why are there so many different optimization algorithms? How do we decide which one to choose?

If you read the docs for each one, they describe a formula for how it updates model parameters. What does each formula mean and what is its significance?

Before we are ready to dive into the maths, my goal with this article is to provide some overall context and get some intuition about how each algorithm fits in. In fact, I will not discuss the formulae themselves here but will leave that for another article.

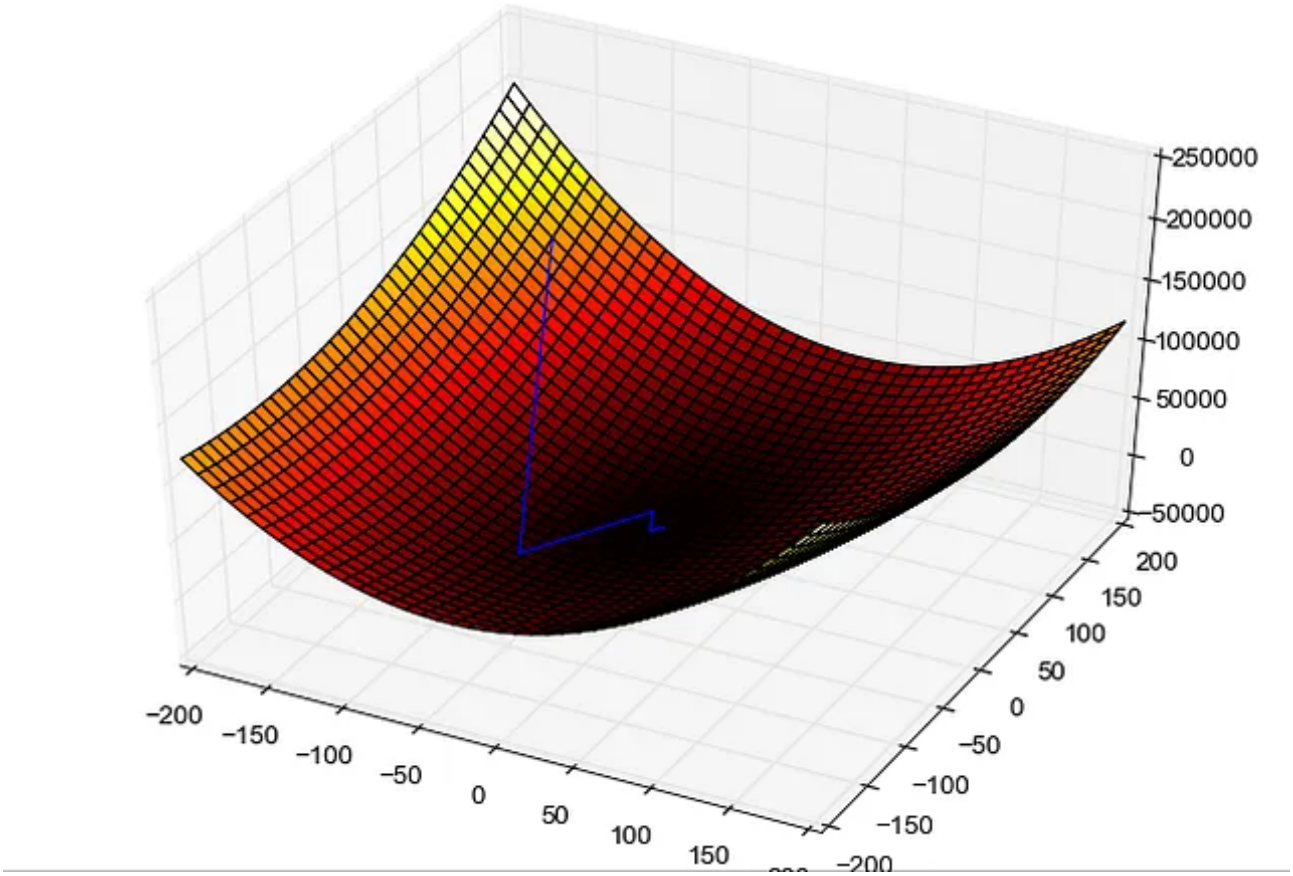
And if you're interested in Neural Network architectures in general, I have some other articles you might like.

1. Batch Norm — How it Works *(Essential deep learning layer that significantly stabilizes and speeds up model training)*
2. Differential and Adaptive Learning Rates (*How Optimizers and Schedulers can be used to boost model training and tune hyperparameters*)
3. Image Captions Architecture *(Multi-modal CNN and RNN architectures with Image Feature Encoders, Sequence Decoders, and Attention)*

Review of Optimization with Gradient Descent

Loss Curve

Let's start with a typical 3D picture of the gradient descent algorithm at work.



Loss curve for Gradient Descent (Source)

This picture shows a network with two weight parameters:

- The horizontal plane has two axes, for weights w1 and w2 respectively.
- The vertical axis shows the value of the loss, for each combination of the weights

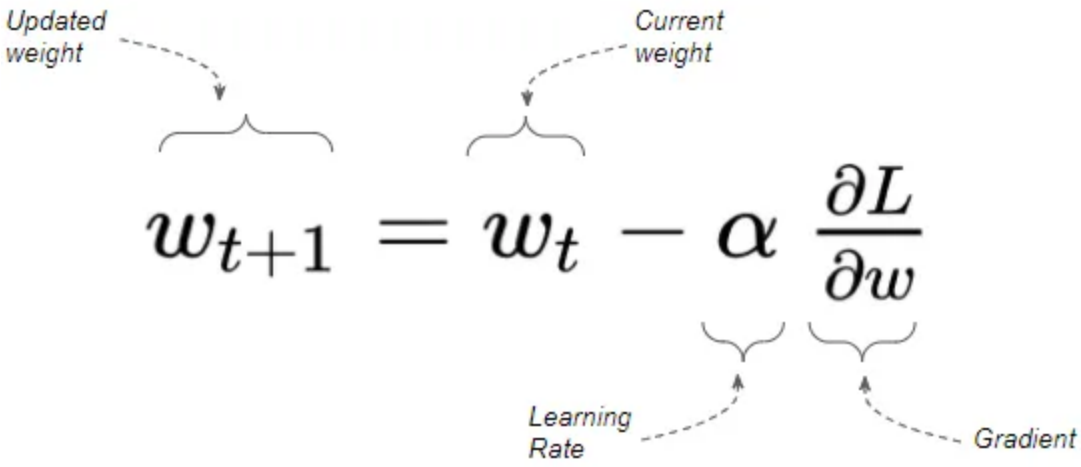
In other words, the shape of the curve shows the "Loss Landscape" for the neural network. It plots the loss for different values of the weights, while we keep the input dataset fixed.

The blue line plots the trajectory of the gradient descent algorithm during optimization:

- It starts off by choosing some random values for both weights and computes the loss value.
- At each iteration, as it updates its weight values, resulting in a lower loss (hopefully), it moves to a lower point along the curve
- Finally, it arrives at its objective which is the bottom of the curve where the loss is lowest.

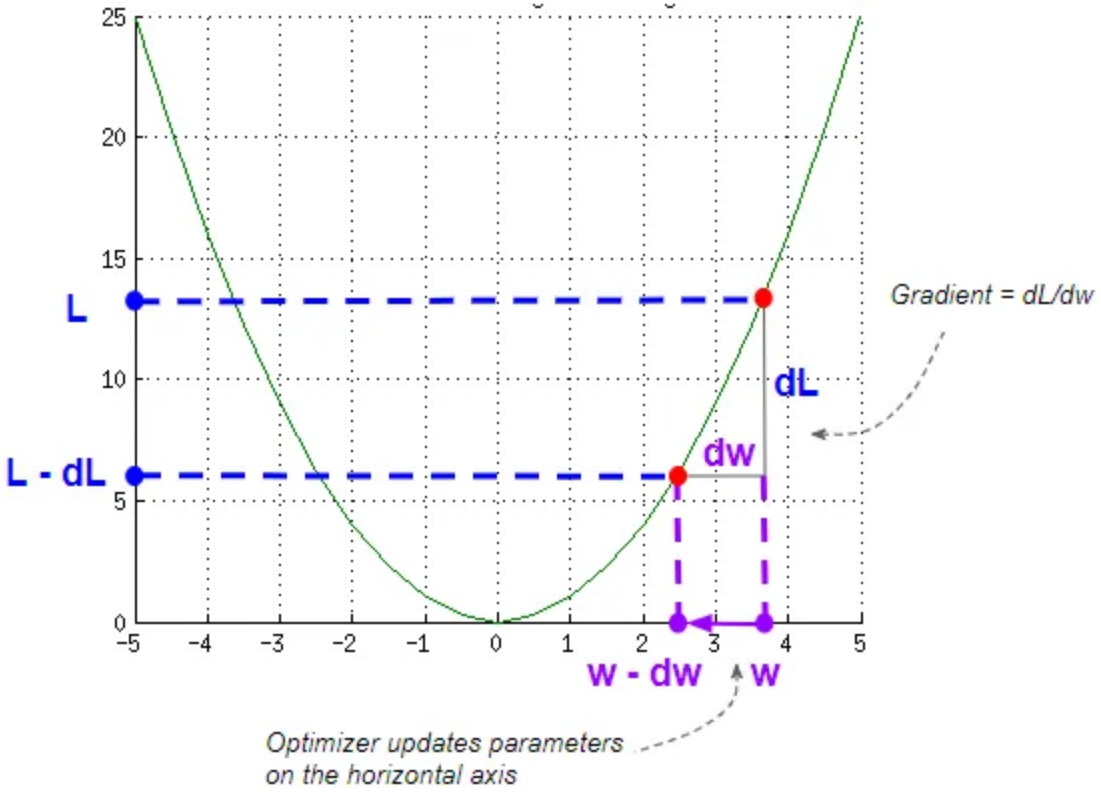
Computing the Gradient

The algorithm updates weights based on the gradient of the loss curve at that point, and a learning rate factor.



Gradient Descent parameter update (Image by Author)

The gradient measures the slope and is the change in the vertical direction (*dL*) divided by the change in the horizontal direction (*dW*). This means that the gradient is large for steep slopes and small for gentle slopes.

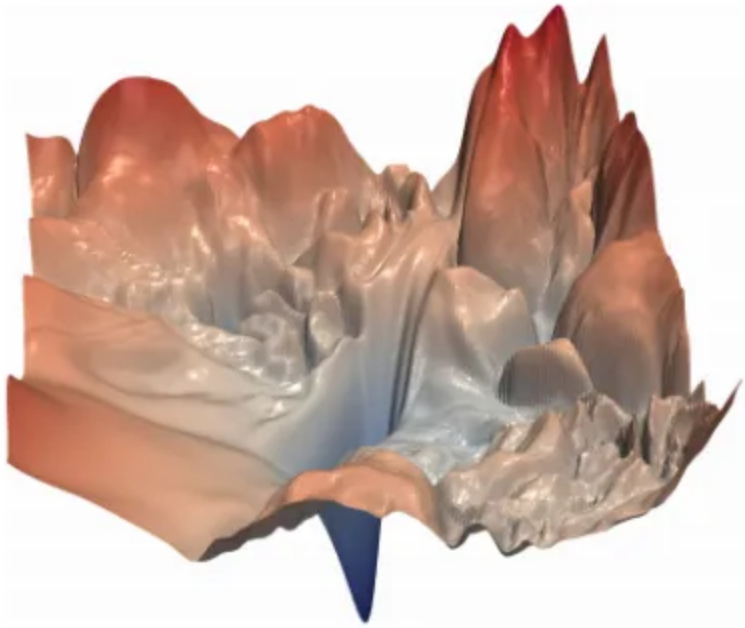


Computing the Gradient (Image by Author)

Gradient Descent in Practice

These loss curves are a useful visualization to understand the concept of gradient descent. However, we should realize that this is an idealized scenario, not a realistic one:

- The picture above shows a smooth convex-shaped curve. In reality, the curve is very bumpy.



A neural network loss landscape (Source, by permission of Hao Li)

- Secondly, we are not going to have just 2 parameters. There are often tens or hundreds of millions, and it is impossible to visualize or even imagine that in your head.

At each iteration, gradient descent works by "looking in all directions to find the best slope that it can go down". So what happens when the best slope (at that point) isn't the best direction to take?

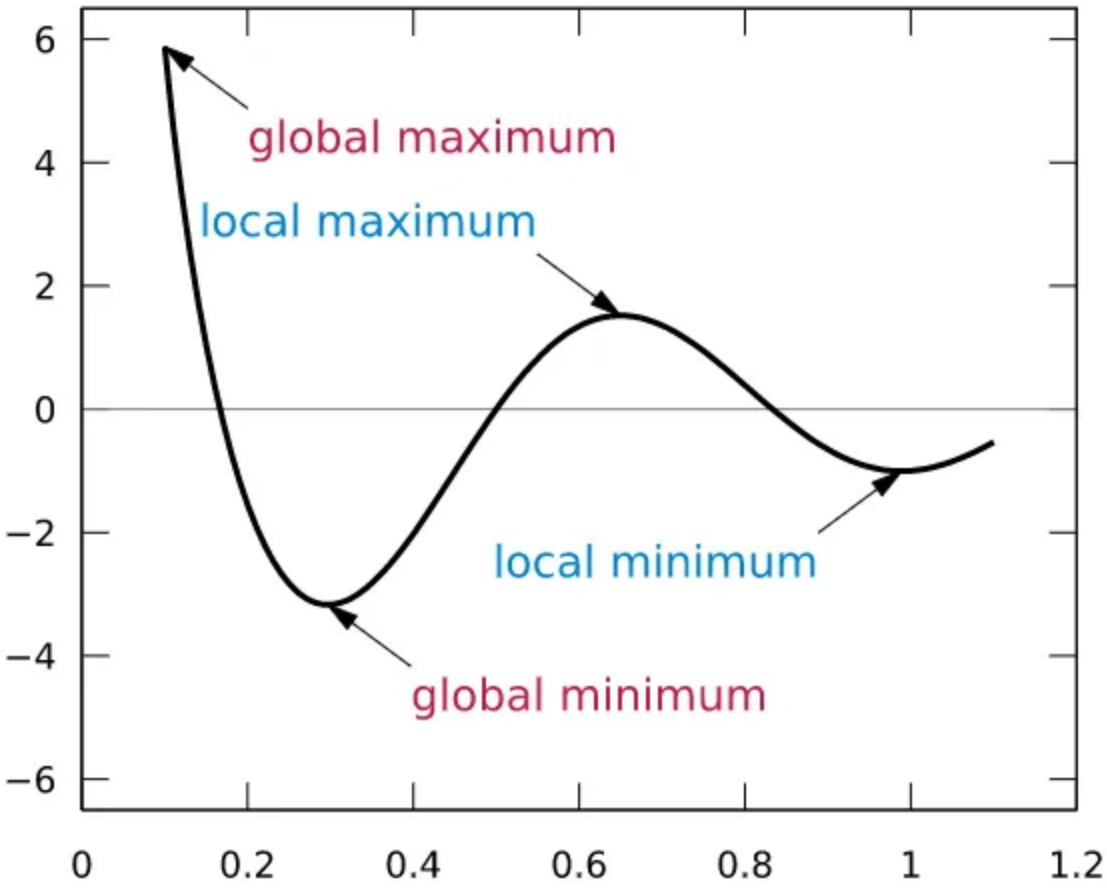
- What if the landscape slopes steeply in one direction but the lowest point is in the direction of the more gentle slope?
- Or what if the landscape all around is fairly flat?
- Or if it goes down a deep ditch how does it climb out of it?

These are some examples of curves that pose difficulties for it. Let's look at those next.

Challenges with Gradient Descent Optimization

Local Minima

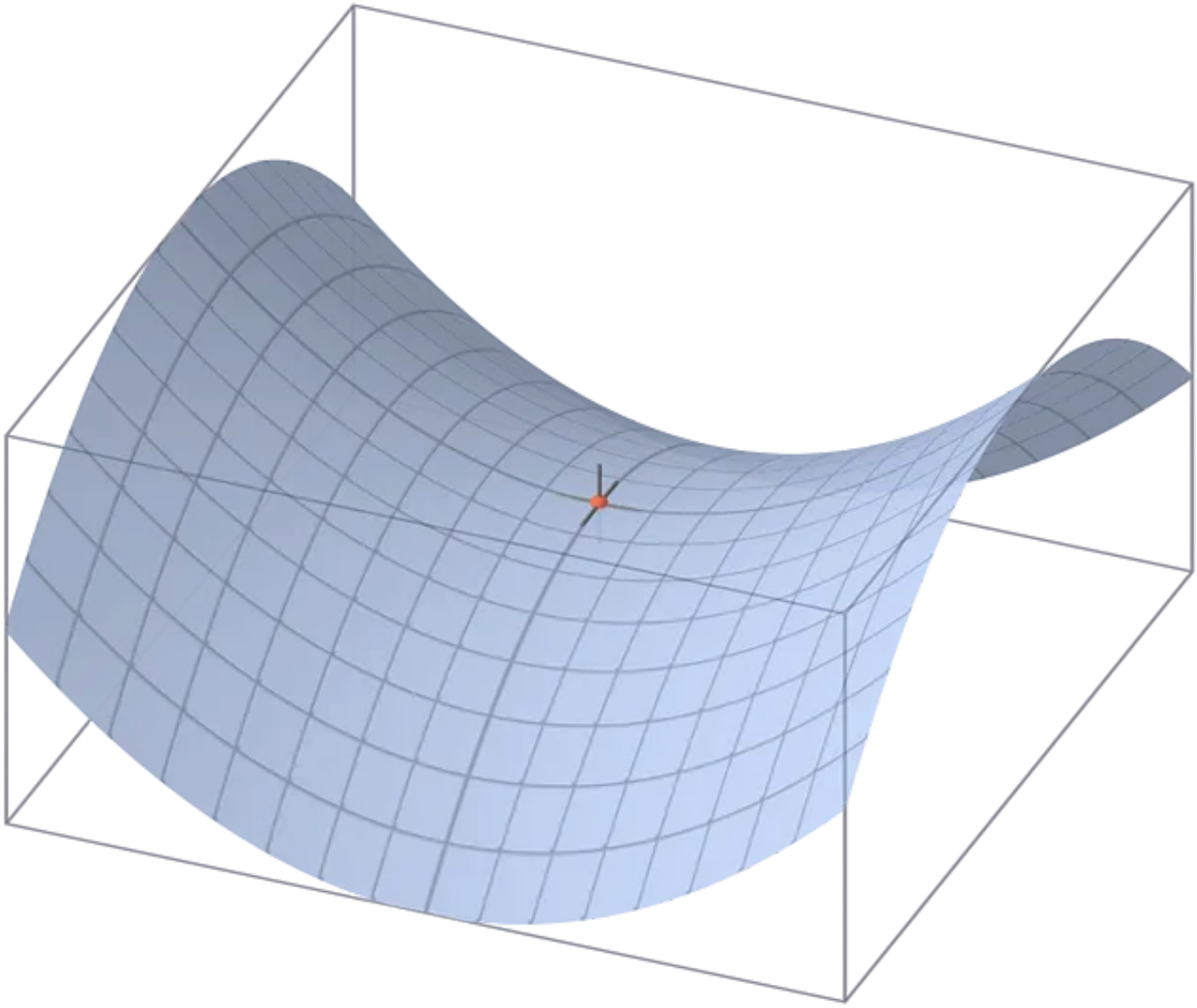
In a typical loss curve, you might have many local minima in addition to the global minimum. Because Gradient Descent is designed to keep going downwards, once it goes down a local minimum, it finds it very difficult to climb back up the slope. So it might get stuck there without reaching the global minimum.



Local minima and Global minimum (Source)

Saddle Points

Another key challenge is the occurrence of “saddle points”. This is a point where, in one direction corresponding to one parameter, the curve is at a local minimum. On the other hand, in a second direction corresponding to another parameter, the curve is at a local maximum.



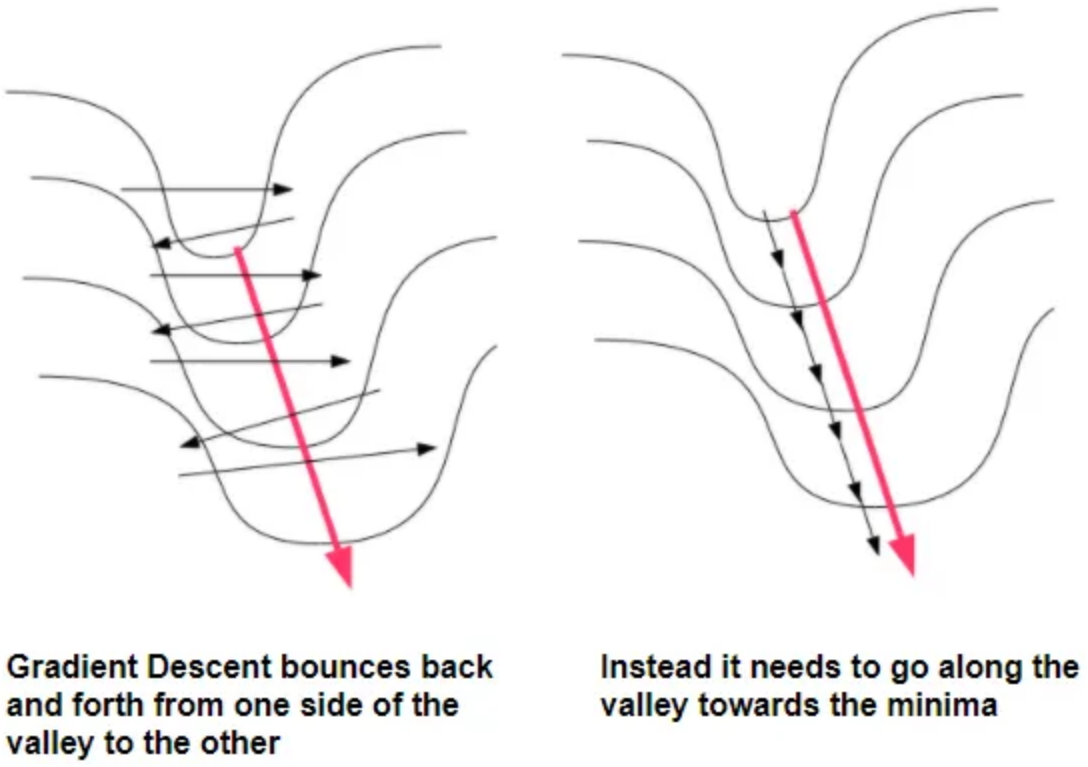
Saddle Point (Source)

What makes saddle points tricky, is that the area immediately around the saddle point is usually fairly flat, like a plateau. This means that the gradients are close to zero. This causes the optimizer to oscillate around the saddle point, in the direction of the first parameter, without being able to descend down the slope in the direction of the second parameter.

The gradient descent thus incorrectly assumes that it has found the minimum.

Ravines

Gradient Descent also finds it hard to traverse ravines. This is a long narrow valley that slopes steeply in one direction (ie. the sides of the valley) and gently (ie. along the valley) in the second direction. Often this ravine leads down to the minimum. Because it is difficult to navigate, this shape is also called Pathological Curvature.



Ravines (Modified from Source, by permission of James Martens)

Think of this like a narrow river valley that slopes down gently from the hills till it ends in a lake. What you want to do is move quickly downriver in the direction of the valley. However, it is very easy for gradient descent to bounce back and forth along the sides of the valley and move very slowly in the direction of the river.

Although they continue to use gradient descent at the core, optimization algorithms have developed a series of improvements on the vanilla gradient descent, to tackle these challenges.

First Improvement to Gradient Descent — Stochastic Gradient Descent (SGD)

Gradient Descent usually means “full-batch gradient descent”, where the loss and gradient are calculated using all the items in the dataset.

Instead, Mini-batch Stochastic Gradient Descent takes a randomly selected subset of the dataset for each training iteration.

That randomness helps us explore the loss landscape.

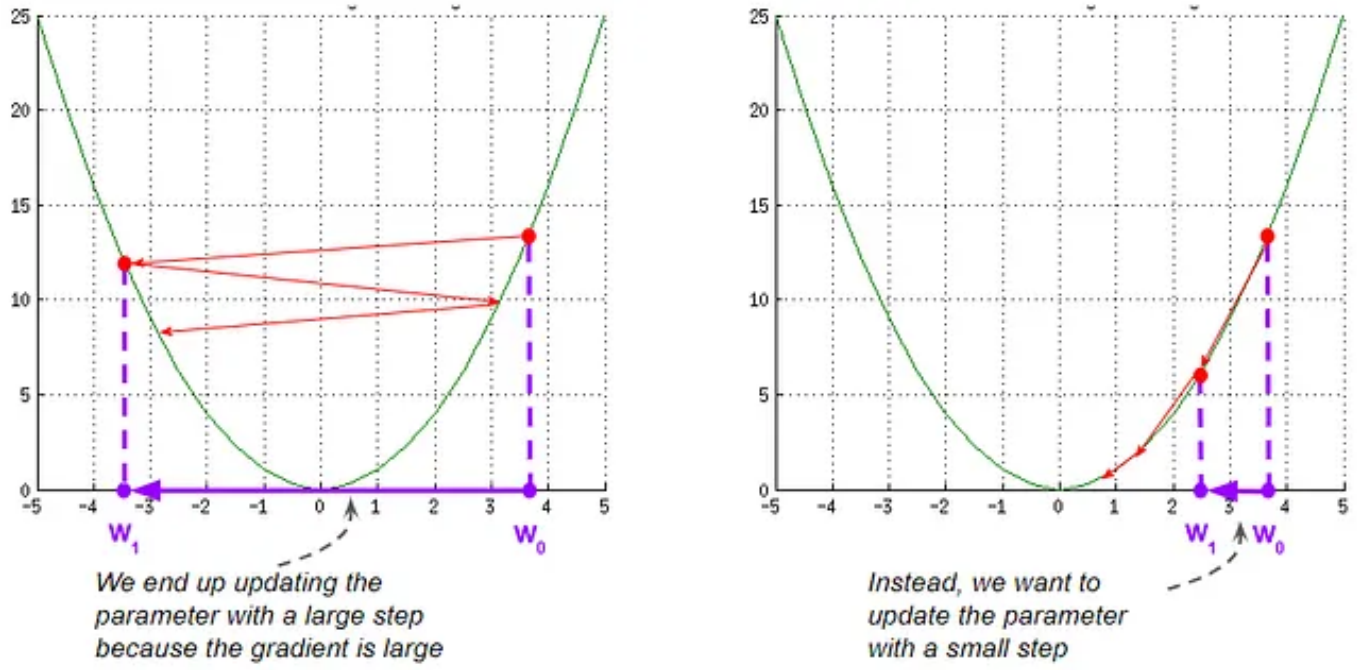
Earlier we had mentioned that the loss curve is obtained by varying the model parameters while keeping the input dataset fixed. However, if you vary the input by selecting different data samples in each mini-batch, the loss value and gradients will vary also. In other words, by varying the input dataset, you get a slightly different loss curve with each mini-batch.

So even though you get stuck at someplace in the landscape in one mini-batch, you might see a different landscape for the next mini-batch, which lets you keep moving. This prevents the algorithm from getting stuck in a particular section of the landscape, especially in the early stages of training.

Second Improvement to Gradient Descent — Momentum

Adjust the Update Amount dynamically

One of the tricky aspects of Gradient Descent is dealing with steep slopes. Because the gradient is large there, you could take a large step when you actually want to go slowly and cautiously. This could result in bouncing back and forth, thus slowing down the training.



(Image by Author)

Ideally, you want to vary the magnitude of the update dynamically so you can respond to changes in the landscape around you. If the slope is very steep you want to slow down. If the slope is very flat you might want to speed up and so on.

With Gradient Descent you make an update to the weights at each step, based on the gradient and the learning rate. Therefore, to modify the size of the update, there are two things you can do:

- Adjust the gradient
- Adjust the learning rate

Momentum vs SGD

Momentum is a way to do the former above ie. adjust the gradient.

With SGD we look only at the current gradient and ignore all the past gradients along the path we took. This means that if there is a sudden

anomaly in the loss curve, your trajectory may get thrown off course.

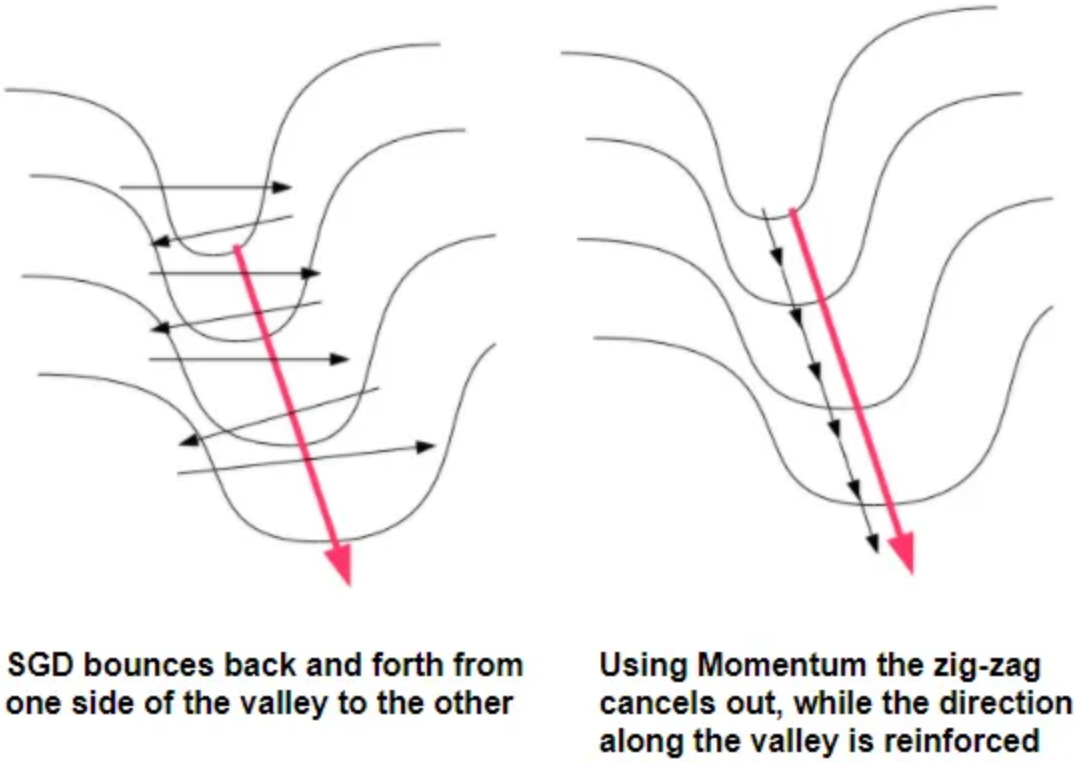
When using Momentum, on the other hand, you let the past gradients guide your overall direction so that you stay on course. This lets you use the knowledge of the surrounding landscape that you had seen up until that point, and helps dampen the effect of outliers in the loss curve.

1. The first question is how far in the past do you go? The further back you go, the less you will be affected by anomalies.
2. Secondly, does every gradient from the past count equally? It would make sense that things from the recent past should count more than things from the distant past. So if the change in the landscape is not a temporary anomaly, but a genuine structural change, then you do need to react to it and change your course gradually.

The Momentum algorithm uses the exponential moving average of the gradient, instead of the current gradient value.

Traverse Ravines using Momentum

Momentum helps you tackle the narrow ravine problem of pathological curvature, where the gradient is very high for one weight parameter but very low for another parameter.



Momentum helps you traverse ravines (Modified from Source, by permission of James Martens)

By using momentum, you dampen the zig-zag oscillations that would happen with SGD.

- For the first parameter with the steep slope, the large gradient causes a 'zig' from one side of the valley to the other. However, in the next step, this gets canceled out by the 'zag' in the reverse direction.
- On the other hand for the second parameter, the small updates from the first step are reinforced by the small updates for the second step because they are in the same direction. This is the direction along the valley in

formulae are:

- SGD with Momentum

- Nestorov Accelerated Gradient

Third Improvement to Gradient Descent — Modify Learning Rate (based on the gradient)

As mentioned above, the second way to modify the amount of the parameter update is by adjusting the learning rate.

So far, we have been keeping the learning rate constant from one iteration to the next. Secondly, the gradient updates are using the same learning rate for all parameters.

However, as we have seen there might be large variations between the gradients of different parameters. One parameter might have a steep slope while another has a gentle slope.

We can make use of this to adapt the learning rate to each parameter. We can make use of past gradients (for each parameter separately) to choose the learning rate for that parameter.

There are a few Optimizer algorithms that do this, using slightly different techniques eg. Adagrad, Adadelta, RMS Prop.

For instance, Adagrad squares the past gradients and adds them up, weighting all of them equally. RMSProp also squares the past gradients but uses their exponential moving average, thus giving more importance to recent gradients.

Now, by squaring the gradients, they all become positive ie. have the same direction. This negates the canceling out effect that we talked about for Momentum, with gradients in opposite directions.

This means that for a parameter that has a steep slope, the gradients are large and the squares of the gradients are really large and always positive, so they accumulate fast. To dampen this, the algorithm calculates the learning rate by dividing the accumulated squared gradients by a larger factor. This allows it to slow down on steep slopes.

Similarly, for shallower slopes, the accumulation is small and so the algorithm divides the accumulated squares by a smaller factor to compute the learning rate. This boosts the learning rate for gentle slopes.

Some Optimizer algorithms combine both approaches — modify the learning rate as above as well as use Momentum to modify the gradient. eg. Adam and its many variants, LAMB.

Fourth Improvement to Gradient Descent — Modify Learning Rate (based on your training progress)

In the previous section, the learning rate was modified based on the gradients of the parameters. In addition, we can adjust the learning rate based on the progression of the training process. The learning rate is set based on the training epoch and is independent of the model's parameters at that point.

This is actually not done by Optimizers at all. It is, in fact, a separate component of the neural network known as a Scheduler. I am mentioning this for completeness and to show the relationship with the Optimization techniques we've discussed, but will not cover them further here.

Conclusion

We now understand the basic techniques used by Optimizers based on Gradient Descent, why they are used, and how they relate to one another.

This puts us in a good position to go deeper into many of the specific optimization algorithms and understand how they work in detail. I hope to cover that in another article soon...

And finally, if you liked this article, you might also enjoy my other series on Transformers, Audio Deep Learning, and Geolocation Machine Learning.

Transformers Explained Visually (Part 1): Overview of Functionality

A Gentle Guide to Transformers for NLP, and why they are better than RNNs, in Plain English. How Attention helps...

towardsdatascience.com

Audio Deep Learning Made Simple (Part 1): State-of-the-Art Techniques

A Gentle Guide to the world of disruptive deep learning audio applications and architectures. And why we all need to...

towardsdatascience.com

Leveraging Geolocation Data for Machine Learning: Essential Techniques

A Gentle Guide to Feature Engineering and Visualization with Geospatial data, in Plain English

towardsdatascience.com

Let's keep learning!

Neural Networks | Deep Learning | Data Science | Editors Pick | Hands On Tutorials

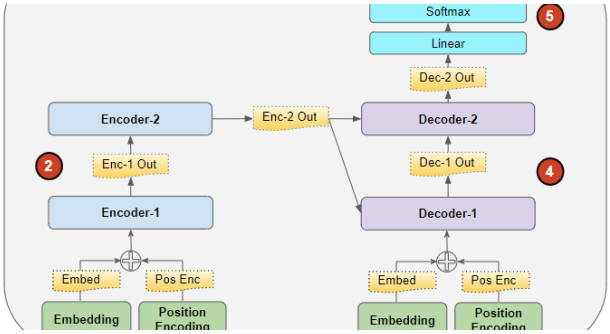
Written by Ketan Doshi

4.8K Followers · Writer for Towards Data Science

Machine Learning and Big Data

Follow

More from Ketan Doshi and Towards Data Science



Ketan Doshi in Towards Data Science

Transformers Explained Visually (Part 1): Overview of Functionality

A Gentle Guide to Transformers for NLP, and why they are better than RNNs, in Plain...

10 min read · Dec 13, 2020

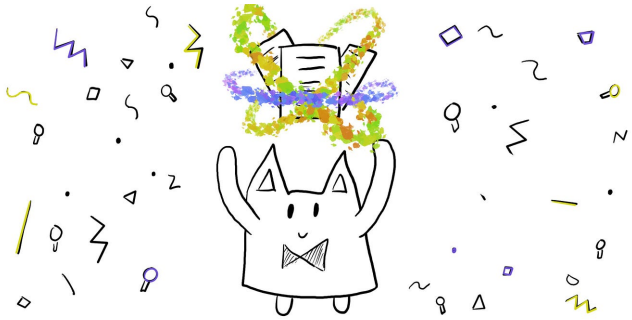
Marco Peixeiro in Towards Data Science

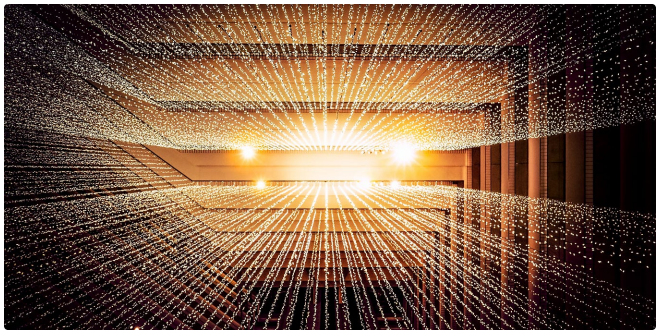
TimeGPT: The First Foundation Model for Time Series Forecasting

Explore the first generative pre-trained forecasting model and apply it in a project...

· 12 min read · Oct 24

2.9K 15 2.2K 19





Adrian H. Raudaschl in Towards Data Science

Forget RAG, the Future is RAG-Fusion

The Next Frontier of Search: Retrieval Augmented Generation meets Reciprocal...

· 10 min read · Oct 6

2.4K 24

Ketan Doshi in Towards Data Science

Transformers Explained Visually (Part 2): How it works, step-by-step

A Gentle Guide to the Transformer under the hood, and its end-to-end operation.

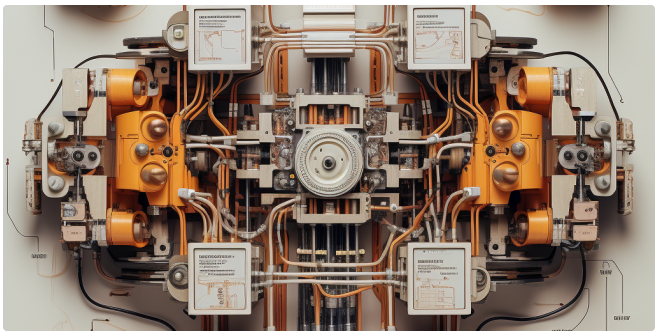
11 min read · Jan 2, 2021

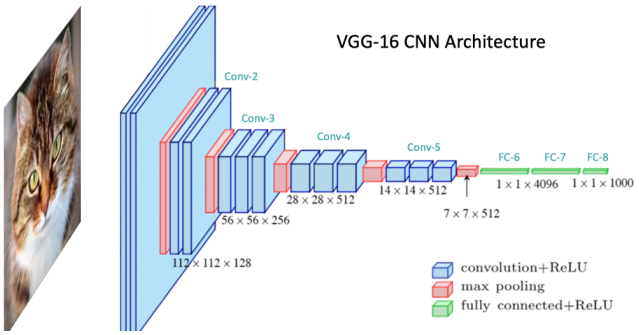
2K 21

See all from Ketan Doshi

See all from Towards Data Science

Recommended from Medium





Daniel Warfield in Towards Data Science

Transformers—Intuitively and Exhaustively Explained

Exploring the modern wave of machine learning: taking apart the transformer step b...

14 min read · Sep 20

1.4K 10

Luís Fernando Torres in LatinXinAI

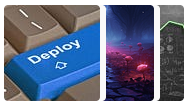
Convolutional Neural Network From Scratch

The most effective way of working with image data

21 min read · Oct 16

405 4

Lists



Predictive Modeling w/ Python

20 stories · 594 saves

New_Reading_List

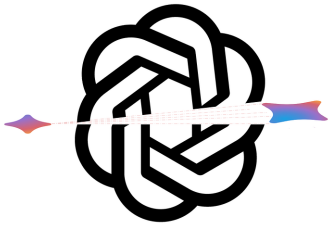
174 stories · 192 saves

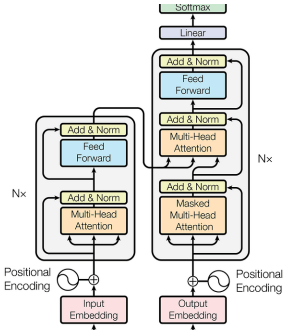
Practical Guides to Machine Learning

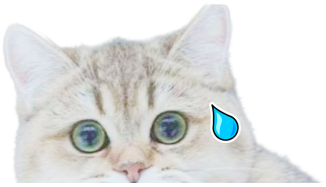
10 stories · 672 saves

Natural Language Processing

847 stories · 395 saves







AL Anany

The ChatGPT Hype Is Over—Now Watch How Google Will Kill...

It never happens instantly. The business game is longer than you know.

✦ · 6 min read · Sep 1

18.7K 571

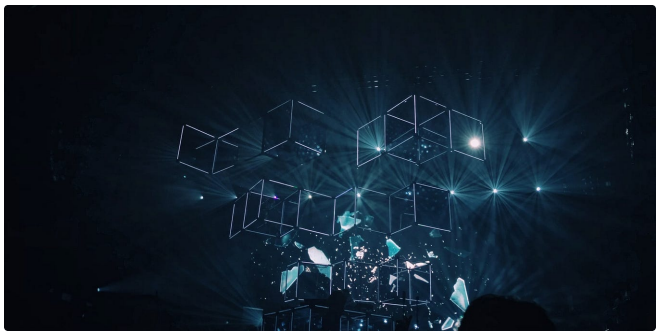
Fareed Khan in GoPenAI

Understanding Transformers: A Step-by-Step Math Example—Pa...

I understand that the transformer architecture may seem scary, and you might have...

6 min read · Jun 5

2K 47



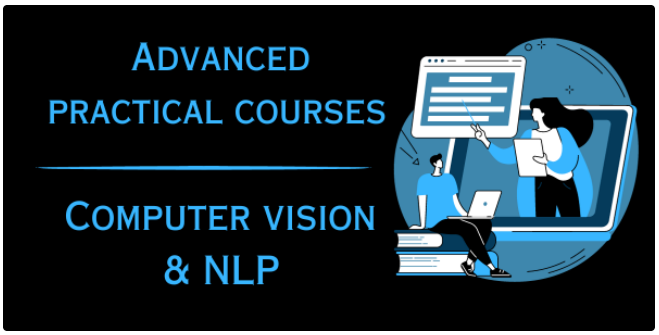
Virat Patel

I applied to 230 Data science jobs during last 2 months and this is...

A little bit about myself: I have been working as a Data Analyst for a little over 2 years....

✦ · 3 min read · Aug 11

2.4K 55



Youssef Hosni in Level Up Coding

Advanced Practical Computer Vision & NLP Courses from Top...

With the rise of powerful large language models and the latest advances in computer...

✦ · 5 min read · Nov 7

270 1

See more recommendations