MINI *chocolate chip* BATCH

Kuba Jerzmanowski

Tuesday 2nd July, 2024

1 Introduction to Gradient Descent Variants

We will first talk about BGD (Batch Gradient Descent) and SGD (Stochastic Gradient Descent) as we can think of gradient descent as a slider, with the two extremes being BGD and SGD, and mini-batches somewhere in the middle.

2 Gradient Descent Algorithm Review

Let's remember what the gradient descent algorithm does:

1. We get some output from the forward pass.
2. Calculate the loss by comparing the output from the forward pass to the expected/true value.
3. Get the gradient, which is the derivative of each of the weights and biases in the model with respect to the loss function (we are trying to minimize the loss).
4. Update the weights according to the gradient calculated for each one (the amount we change by can vary a lot depending on our hyperparameters).

3 Batch Gradient Descent (BGD)

Up until now (except for some parts of lesson 03), we always passed in the whole X_train (training set), got the average gradient for the whole thing, and then took a step in that direction. That was one epoch. For the next epoch, we pass in the whole X_train again, get the average gradient, and take a step in that direction again.

Example:

grad(X_{34}) (training sample 0) = +0.8
grad(X_{34}) (training sample 1) = +0.67
grad(X_{34}) (training sample 2) = -0.14
grad(X_{34}) (training sample 3) = +0.57

Sum: 0.8 + 0.67 - 0.14 + 0.57 = 1.9
Average = 1.9 / 4 = 0.475

So our gradient with respect to weight_{34} is now = 0.475

This is what we have been doing.

4 Stochastic Gradient Descent (SGD)

In SGD, we randomly select one training sample from X_train. We do the forward pass on just that one sample, then calculate the loss on that one sample and adjust the weights according to the gradient with respect to the loss of just that one example. That's one epoch. In the next epoch, we again get a single random sample from X_train with replacement and do a gradient descent step with just that one sample.

Example (given that we randomly select sample 1):

$\text{grad}(X_{34})$ (training sample 1) = +0.67
Sum: +0.67 = 0.67
Average = 0.67 / 1 = 0.67

So our gradient with respect to weight_{34} is now = 0.67

5 Comparing BGD and SGD

As mentioned in the beginning, the two extremes are: for each weight and bias adjustment (epoch), we do based on one randomly selected sample (SGD) or the average gradient over the whole training set (BGD). Now you can start to think of how there can be many different versions in the middle (Mini-Batch GD).

5.1 Practical Considerations

SGD requires less memory. For BGD, you have to load your dataset and store it in RAM (memory) for each training iteration. If your dataset is huge, this can obviously cause lots of problems. With SGD, you just need to store the sample you randomly selected for the training iteration, and during the next one, you store the new randomly selected sample while discarding the old one.

The other advantage of SGD is that it is plainly and simply just faster. If you need to calculate the gradient over 100,000 samples versus 1 sample, the 1 sample (SGD) will obviously be faster.

5.2 Learning Effects

Let's remember that there is a trade-off between variance and bias. Generally, this can be thought of as we want to balance how well we do on our training set to ensure we generalize to new data better. Remember, there is some true

function underlying the thing we are trying to model. If we had all the data in the world, then we would not care about overfitting as we could memorize the data since it represents the true underlying distribution. However, in reality, we don't know the true underlying distribution and don't have all the data in the world.

5.3 Loss Surface Visualization

In BGD, there is a loss surface and we want to find the minimum of it. It can be visualized as follows:

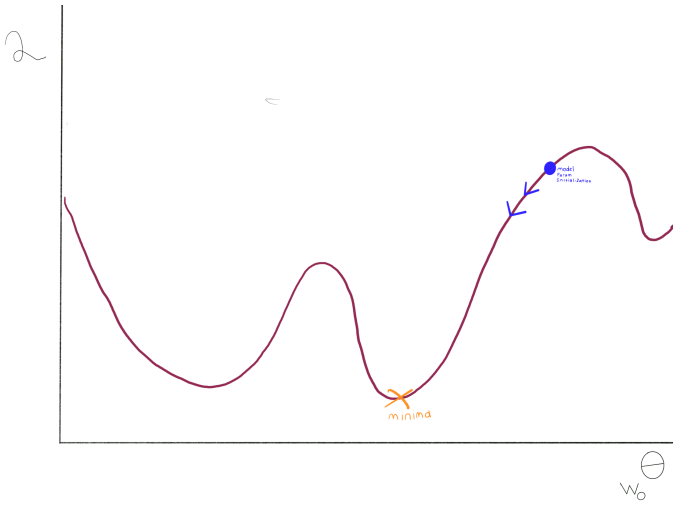

Figure 1: BGD Loss Surface

(a) Function of Loss w.r.t patameters in this case just w_0

We want to minimize L, the loss, by adjusting our parameters. In this case, it's just one parameter w_0. But one important thing to note is that this is a bit misleading as we don't walk around the loss surface. We don't really know the loss surface; we just know in which direction the gradient points. Thus, a more accurate representation is:

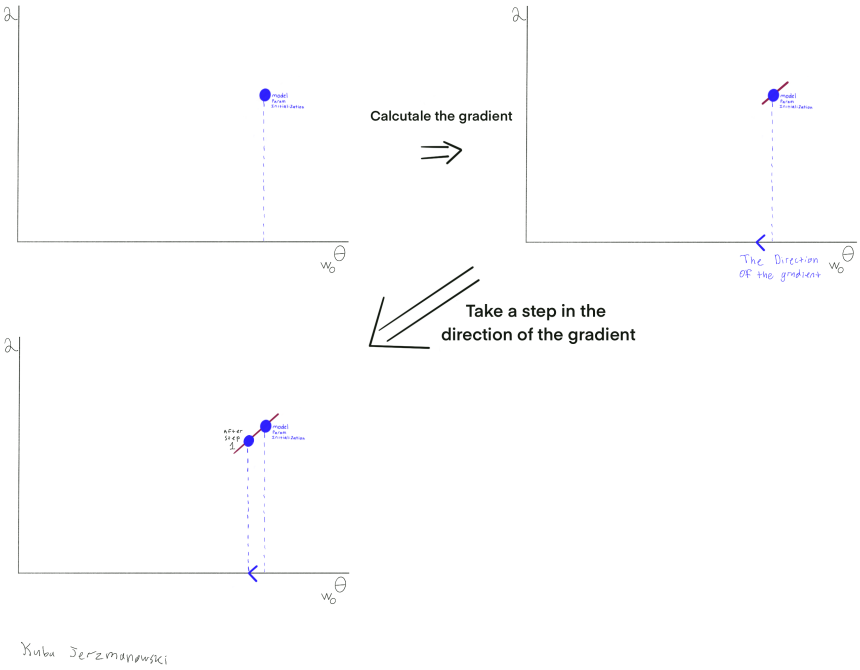

Figure 2: Gradient Direction Representation

But for our understanding throughout the rest of this, I will draw it as such:

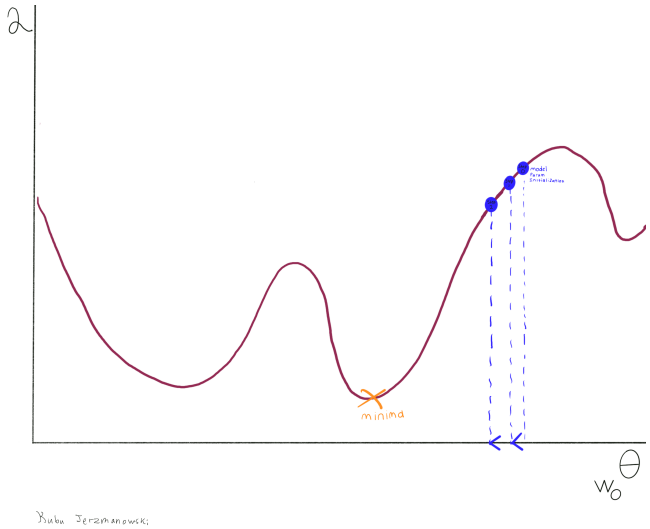

Figure 3: Visually Simplified Loss Surface Representation

It's vital to remember that the algorithms only know in what direction to move locally with very small changes of the parameters. That's the very definition of a derivative: the rate of change at a single instance.

5.4 BGD vs SGD Visualization

In BGD, it would look something like this:

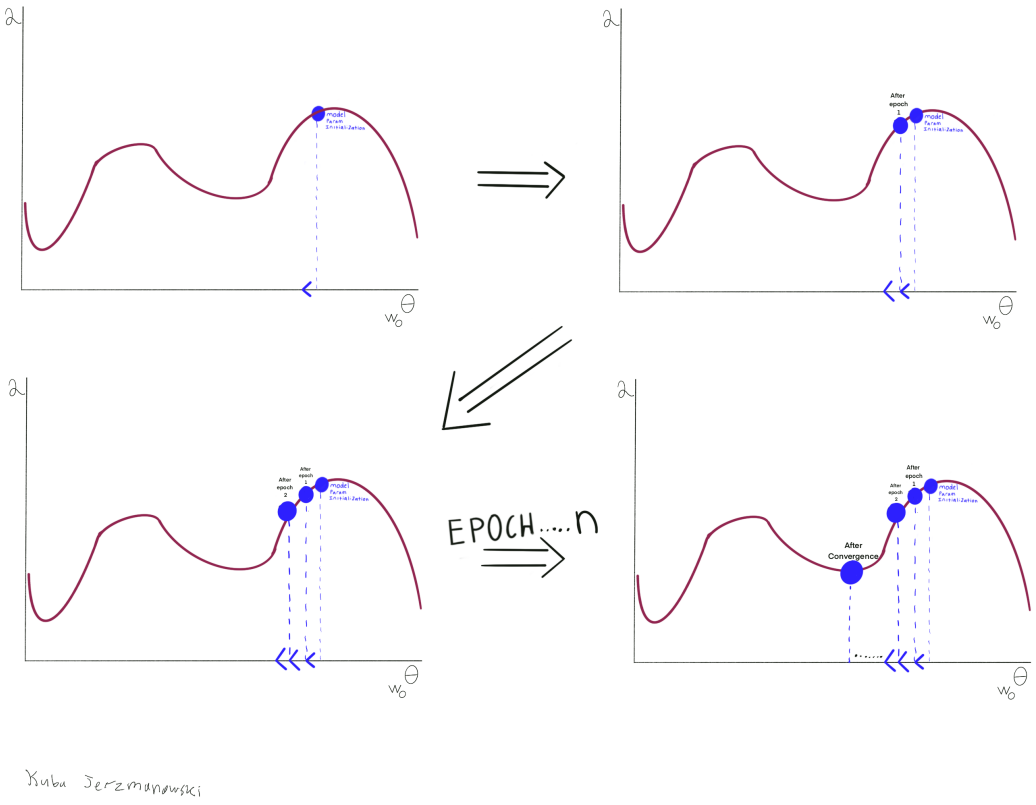

Figure 4: BGD Optimization Path

Our loss surface is always the same since it's based on the same training set. Each time we will move along our loss surface as the gradient points us, but the loss surface stays the same.

Here are visualizations of SGD. The red line is the distribution/loss surface of the whole training set and the light green line is the distribution/loss surface of the single sample that we randomly selected. Remember, the gradient is according to the light green line line, so we will see how that can lead us to new places:

note that the blue dot seems to be jumping from epoch to epoch up and down. That's one way to think of it. The parameters, denoted by θ (in our case just w_0) are adjusted in epoch 1 then they are the same till the end of epoch 2 where we will adjust them according to the new gradient however the "teleportation" happens in SGD since each sample has a slightly different loss function. This difference causes the same w_0 to result in a different loss

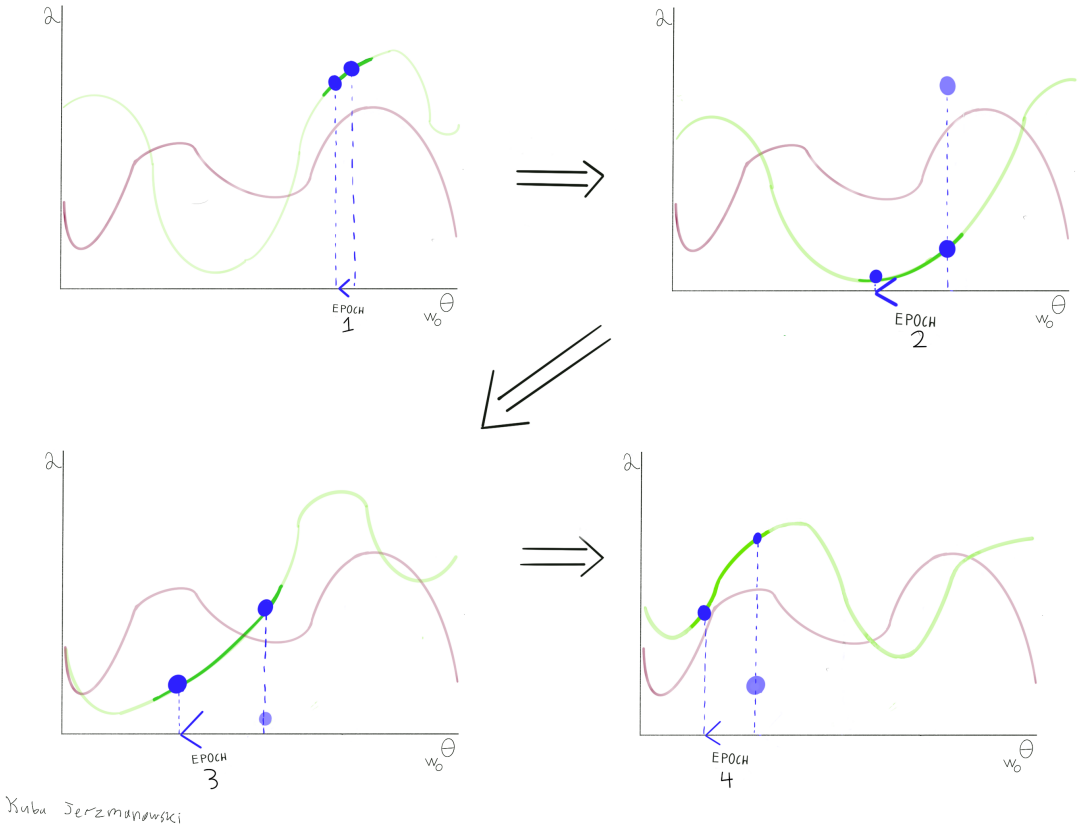

Figure 5: SGD Optimization Path - Scenario 1

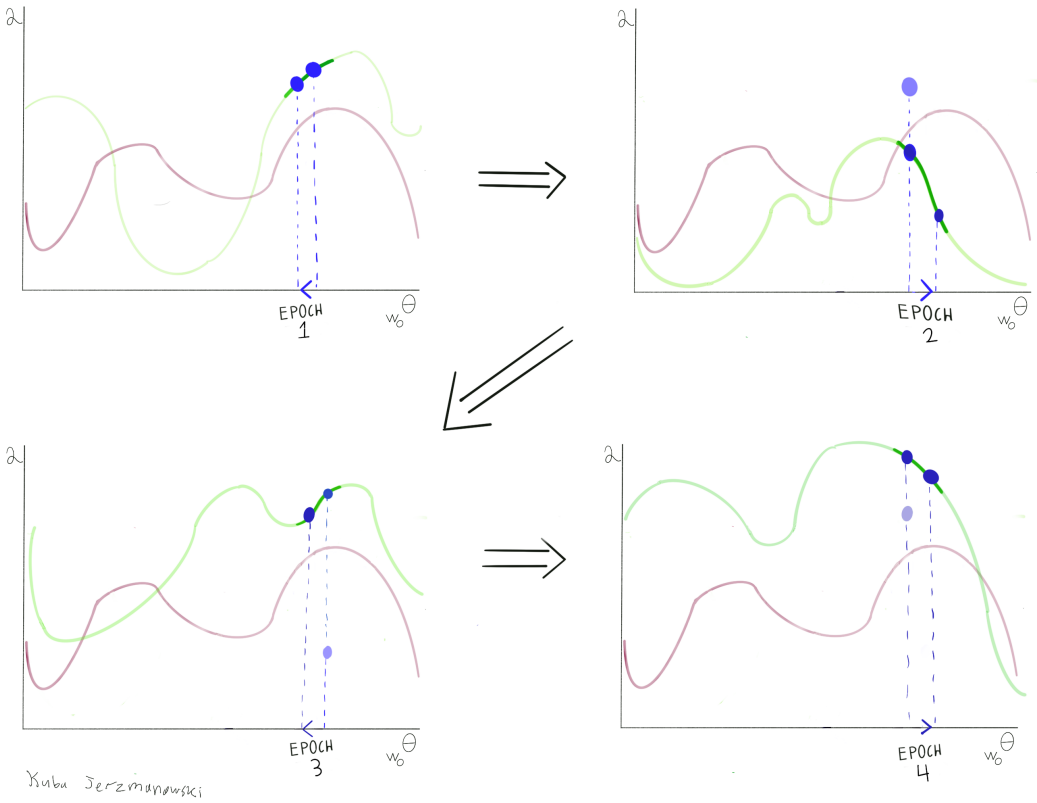

Figure 6: SGD Optimization Path - Scenario 2

In SGD, we can see something like this: each time we have a new loss surface. So we load a sample, calculate the gradient, and take a step in that direction. Then we load a new sample and take a step in the direction that the gradient points us for the new sample. You can see how this can help us find a better minimum as in Figure 5, but since we are sampling individual samples, it can cause us to jump around a lot, especially if we sometimes get a bad samples like in Figure 6.

5.5 BGD vs SGD Conclusion

We can see that there are advantages to not training on the whole dataset. Since the training set changes in SGD, it prevents us from having too high bias as we can't just memorize the training set. The training set changes all the time and we are "jumping around," leading us to better local minima. But we can also see how this can lead us to jump around too much since there is a lot of difference between single samples, which can lead us to have high variance.

Admittedly, the draw out examples were a bit contrived as we are adjusting

just one parameter W_0, but in reality, you would have hundreds to billions of parameters. But the idea is that SGD can help us explore places in the parameter space that we would have never seen given that we do BGD.

What if we could have the best of both worlds?

6 Mini-Batch Gradient Descent: The Best of Both Worlds

What if we could have the best of both worlds?

Mini-batch gradient descent offers a compromise between BGD and SGD. We want to have some noise to help us avoid getting stuck in local minima, but we don't want it to be so noisy that we don't learn effectively. That's where mini-batches come in.

With a mini-batch approach, we:

1. Sample a batch (set of examples) from the training set
2. Calculate the average gradient of this batch
3. Perform the parameter update for the epoch
4. Sample a new batch for the next epoch

That's mini-batches in a nutshell.

6.1 Tips for Mini-Batches

- Batch sizes between M=2 to M=32 typically work best for generalization [Masters and Luschi, 2018]
- For larger datasets and with batch normalization (normalizing each batch before the epoch), batch sizes up to M=64 can be useful

6.2 A Note on Learning Rates

Now, there is the whole discussion of learning rates. We will leave this discussion for a further date when I have further looked into learning rates and recent research on scaling the learning rate.

For now, as a general guideline, try something between 2×10^{-8} to 2×10^{-4} for your learning rate.

6.3 Limitations and Ongoing Research

It's important to note that while we are talking about mini-batches as a great solution, they share a limitation with SGD that we didn't mention before. Neither mini-batches nor SGD can fully take advantage of the benefits of computational parallelism (processing things in parallel/at the same time).

With larger batches, you can start to get the benefits of parallelization, but you start to lose the generalization benefit. This trade-off between parallelization and generalization is an ongoing area of research [Keskar et al., 2016].

Last Modified: Tuesday 2nd July, 2024

References

Keskar, Nitish Shirish et al. (2016). "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima". In: *CoRR* abs/1609.04836. arXiv: 1609.04836. URL: http://arxiv.org/abs/1609.04836.

Masters, Dominic and Carlo Luschi (2018). "Revisiting Small Batch Training for Deep Neural Networks". In: *CoRR* abs/1804.07612. arXiv: 1804.07612. URL: http://arxiv.org/abs/1804.07612.