Gradient-Based Learning Notes

Brian Fakhoury
September 19, 2017

1. Gradients

1.1 The univariate derivative

In single variable calculus, the derivative tells us the respective change of one variable to another. When computed, the derivative gives us a scalar value.

1.2 The multivariate derivative (the gradient)

In multivariable calculus, many variable are contributing to the output. So, a multivariable “derivative” takes into account the change of all those variables in respect to the dependent output. This results in a vector output, with each element being a scalar derivative in respect to that variable.

2. Cost functions

2.1 Definition

A cost function is an algebraic expression that calculates the distance between the hypothesis of a given model and the known correct answer.

2.2 Uses

Cost functions quantify the accuracy of a model. This accuracy calculation can be different depending on the model.

2.3 Optimizing the cost function

Because the cost function relates the distance the model is from an accurate output to the given parameters of the model, minimizing this loss will essentially improve the model. This is most commonly done by computing the gradient of the cost function in respect to the “weights” or parameters of the model.

2.4 Finding the minimum

Gradients allow one to minimize the cost function by finding the global minimum of the cost function. Once the minimum is found with the gradient, then the most effective parameters are known.

2.5 Non-convex scenarios

Cost functions more often produce complicated landscapes rather than simple “convex” ones. Convex landscapes have a desirable global minima that can be reached from any random starting point. Non-convex functions produce many local minima, which make optimization more difficult.

3. Gradient Descent

3.1 Definition

The gradient descent is the given method of iteratively taking steps downhill on a cost function by taking the gradient periodically, thereby effectively minimizing the outputted loss by finding the combination of n parameters where J(1,...,n) is closest to 0.

3.2 Effectivity

Gradient descent tracks a downhill slope in any dimension. Therefore, this method scales the optimization problem, and gives us a mechanism for updating the model.

3.3 Learning Rate

In and of itself, gradient descent provides undesirably large “steps” downhill. Therefore, a simple method of taming the descent is by scaling it with a scalar value. This value is known as the learning rate and allows gradient descent to more easily find the minima.

3.4 Batch Gradient Descent

Batch gradient descent is a simple method of applying an optimization algorithm over our entire dataset. In Batch GD, the entire (training) dataset is used to compute a loss for the model. This results in a smooth descent, as the model updates in as an average.

3.5 Stochastic Gradient Descent

Stochastic GD operates on one data point at a time. This means that the model's parameters will be updated for each data point in the data set. When the entire dataset has been iterated through, one epoch has passed. Since each point is reflecting it's needed parameters, the gradient descent path will be "noisier," meaning a less smooth line downhill. Stochastic GD can be useful in a "complex" landscape. In a "convex" manifold (or landscape), there is only one global minimum which is reachable from any starting point, as the slope downhill points towards the global minimum. In a complex landscape, many local minima exist, and can slow down or dampen the effects of larger descent methods like batch.

3.6 Mini-batch Gradient Descent

Mini-batch operates between stochastic and batch GD. Instead of taking one data point at a time like stochastic, mini-batch iterates over batches (4, 8, 16, etc...), and provides a "best of both worlds" as it updates the model in larger chunks like batch, yet retains the dynamic of stochastic GD.

3.7 Quick note on Saddle Points

Saddle points form when a points gradient calculation returns zero, yet there are many paths downhill around. This occurs because the derivative in all directions appears as zero, and therefore seems to be a minimum to an unassuming optimization method. Thankfully, saddle points can be overcome by adding noise, or using more advanced optimization methods.

4. Backpropagation

4.1 Definition

Backpropagation is the process of updating weights (parameters) in a computation graph by sending the error signal back from the output weight, through the model, and to the input weights. This allows for error contribution by specific parts of the model to be calculated.

4.2 Backprop Theory

Theoretically, since the final parameter contributes directly to the loss, then the parameters that feed into the final parameter affect the loss by affecting the final parameter. Then, the parameters before the second to last do the same, and so on. Thus, error contribution as a derivative can be propagated by the chain rule.

4.3 Computation Graph

A computation graph is a simple representation of a flow of calculations. One can compute the output by simply following the parameters and computing the desired function at each node. More usefully, we can use back propagation to update the parameters in a computation graph by first calculating the change contribution by each weight at each node. Then, this “local gradient” can be multiplied by the global gradient (from the GD) to compute the contribution of that parameter to the global loss.

4.4 Local Gradients

The local gradient only takes into consideration the change of one parameter in respect to the model’s loss. Each local gradient depends on the gradient of the local farther down the computation graph. Therefore, each new local gradient down the backward pass can be computed by multiplying the previous local gradient, which takes into account its error contribution, times the responsible change of the new local parameter on the previous one.

5. Backprop + GD

5.1 Letting the Model Learn

Finally, gradient descent can be paired with a backpropagation method to informatively update a model’s weights. For each parameter in the model, once the local gradient is calculated, it can be scaled by the learning rate, and added or subtracted to update the weight.

6. Conclusion

6.1 Other Methods

Gradient descent is only one optimization method of many. Another cool one is a genetic algorithm. Instead of informatively updating parameters in a model, the model will evolve randomly into many new variations. Each of these variations will be tested for loss, and eliminated upon poor performance like natural selection! This can be very powerful when we need AI to be creative and come up with a unique design, as an example.

6.2 References and Further Reading

1. Ruder, Sebastian. "An overview of gradient descent optimization algorithms." arXiv preprint arXiv:1609.04747 (2016).

2. Sun, Xu, et al. "meProp: Sparsified Back Propagation for Accelerated Deep Learning with Reduced Overfitting." arXiv preprint arXiv:1706.06197 (2017).
3. Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).
4. Zeiler, Matthew D. "ADADELTA: an adaptive learning rate method." arXiv preprint arXiv:1212.5701 (2012).
5. Du, Simon S., et al. "Gradient Descent Can Take Exponential Time to Escape Saddle Points." arXiv preprint arXiv:1705.10412 (2017).
6. Dean, Jeffrey, et al. "Large scale distributed deep networks." Advances in neural information processing systems. 2012.
7. https://www.youtube.com/watch?v=i94OvYb6noo
8. https://www.youtube.com/watch?v=5u4G23_OohI

6.4 Club Updates

MIC Paper signup: https://goo.gl/iAm6TL
BUMIC Projects signup: https://goo.gl/GmP9oK

Thank you!