Jun 6 · 4 min read

The curious case of the vanishing & exploding gradient

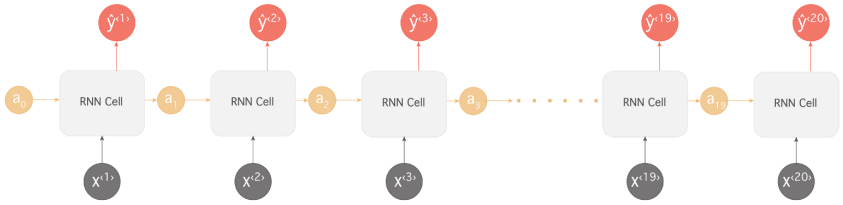
Understanding why gradients explode or vanish and methods for dealing with the problem.

Photo by SpaceX on Unsplash

In the last post, we introduced a step by step walkthrough of RNN training and how to derive the gradients of the network weights using back propagation and the chain rule. But it turns out that during this training the RNN can suffer greatly from two problems: 1. Vanishing gradients or 2. Exploding gradients.

Why Gradients Explode or Vanish

Recall the many-to-many architecture for text generation shown below and in the introduction to RNN post, lets assume the input sequence to the network is a 20 word sentence: *"I grew up in France,…….. I speak French fluently.*

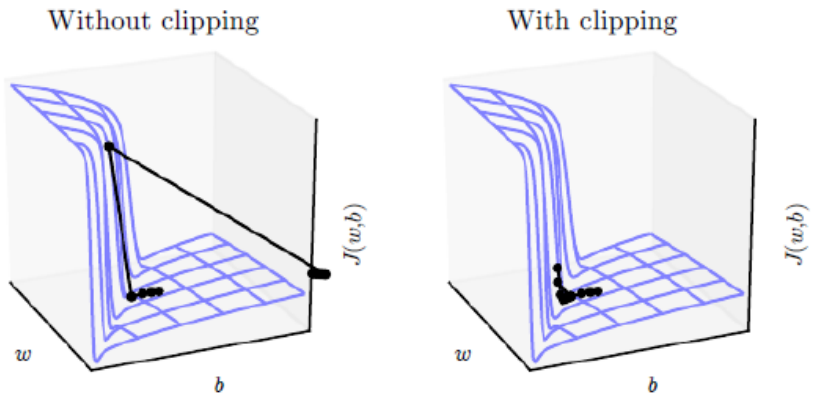


We can see from the example above that for the RNN to predict the word ***"French"*** which comes at the end of the sequence, it would need information from the word ***"France"***, which occurs further back at the beginning of the sentence. This kind of dependence between sequence data is called long-term dependencies because the distance between the relevant information "***France"*** and the point where it is needed to make a prediction "***French"*** is very wide. Unfortunately, in practice as this distance becomes wider, RNNs have a hard time learning these dependencies because it encounters either a vanishing or exploding gradient problem.

These problems arise during training of a deep network when the gradients are being propagated back in time all the way to the initial layer. The gradients coming from the deeper layers have to go through continuous matrix multiplications because of the the chain rule, and as they approach the earlier layers, if they have small values (<1), they shrink exponentially until they vanish and make it impossible for the model to learn , this is the *vanishing gradient problem*. While on the other hand if they have large values (>1) they get larger and eventually blow up and crash the model, this is the *exploding gradient problem*

Dealing with Exploding Gradients

When gradients explode, the gradients could become `NaN` because of the numerical overflow or we might see irregular oscillations in training cost when we plot the learning curve. A solution to fix this is to apply **gradient clipping**; which places a predefined threshold on the gradients to prevent it from getting too large, and by doing this it doesn't change the direction of the gradients it only change its length.



Reference: Ian Goodfellow et. al, "Deep Learning", MIT press, 2016

Dealing with Vanishing Gradients

One simple solution for dealing with vanishing gradient is the **identity RNN** architecture; where the network weights are initialized to the identity matrix and the activation functions are all set to ReLU and this ends up encouraging the network computations to stay close to the identity function. This works well because when the error derivatives are being propagated backwards through time, they remain constants of either 0 or 1, hence aren't likely to suffer from vanishing gradients.

An even more popular and widely used solution is the **Long Short-Term Memory** architecture (LSTM); a variant of the regular recurrent network which was designed to make it easy to capture long-term dependencies in sequence data. The standard RNN operates in such a way that the hidden state activation are influenced by the other local activations closest to them, which corresponds to a *"short-term memory"*, while the network weights are influenced by the computations that take place over entire long sequences, which corresponds to a *"long-term memory"*. Hence the RNN was redesigned so that it has an activation state that can also act like weights and preserve information over long distances, hence the name *"Long Short-Term Memory"*.

LSTM Unit

Conclusion

In this post we explored the reasons why gradients explode or vanish and examined some methods for dealing with the problem. In the next post we would look further into the LSTM architecture and learn how to derive its gradients for back propagation.