1 Vanishing and Exploding Gradients

Training neural networks using backpropagation requires us to compute the partial derivatives of the loss function wrt each weight and bias of the network. The weights and biases are then revised using these partial derivatives using the stochastic gradient descent formulation:

$$w^{\text{new}} = w^{\text{old}} - \eta \frac{\partial L}{\partial w}.$$

If the network is deep enough, then in certain cases it possible for the term $\partial L/\partial w$ to be very small or very large. If this term is very small, then the weight is not updated; if it is very large, there is a large update to the weight, which keeps on repeating at each updation step, with the result that the weights do not converge.

To illustrate this problem, consider a simple network with L layers, with two neurons in each of the first $L-1$ layers and a single neuron in the final layer L. This network is a map from $\boldsymbol{R}^2 \to \boldsymbol{R}^1$. Let $W^{(i)}$ be the weight matrix associated with layer i, where $W^{(1)}, \ldots, W^{(L-1)} \in \boldsymbol{R}^{2\times 2}$ and $W^{(L)} \in \boldsymbol{R}^{1\times 2}$. Ignore the biases and assume that there are no activation functions. Consider a single input-output pair $((x_1, x_2)^T, y)$. Denote the input to layer i as $\boldsymbol{x}^{(i-1)}$, so that input to the very first layer is $\boldsymbol{x}^{(0)} = (x_1, x_2)^T$. Then the output of layer i, ignoring the bias and the activation, is $W^{(i)}\boldsymbol{x}^{(i-1)}$. The output of the first layer is $W^{(1)}\boldsymbol{x}^{(0)}$, that of the second is $W^{(2)}W^{(1)}\boldsymbol{x}^{(0)}$ and so on. The final output $\hat{y}$ of the network is easily seen to be:

$$\hat{y} = W^{(L)} \cdot W^{(L-1)} \cdots W^{(1)} \cdot \boldsymbol{x}^{(0)}.$$

To write this final output more explicitly in terms of the individual weights, we need a little more notation. Let $W^{(i)}$, for $1 \le i \le L-1$, be:

$$\begin{pmatrix} w_{11}^{(i)} & 0 \\ 0 & w_{22}^{(i)} \end{pmatrix}.$$

Let $W^{(L)} = (w_{11}^{(L)}, w_{12}^{(L)})$. Then

$$\hat{y} = (w_{11}^{(L)}, w_{12}^{(L)}) \cdot \begin{pmatrix} w_{11}^{(L-1)} w_{11}^{(L-2)} \cdots w_{11}^{(2)} w_{11}^{(1)} x_1 \\ w_{22}^{(L-1)} w_{22}^{(L-2)} \cdots w_{11}^{(2)} w_{22}^{(1)} x_2 \end{pmatrix}.$$

This ultimately yields:

$$\hat{y} = w_{11}^{(L)} w_{11}^{(L-1)} w_{11}^{(L-2)} \cdots w_{11}^{(2)} w_{11}^{(1)} x_1 + w_{12}^{(L)} w_{22}^{(L-1)} w_{22}^{(L-2)} \cdots w_{11}^{(2)} w_{22}^{(1)} x_2.$$

Assume that the loss function is the standard squared loss: $L(y, \hat{y}) = 0.5 \cdot (y - \hat{y})^2$. Consider the derivative of L wrt one of the weights in the very first layer.

$$\begin{aligned} \frac{\partial L}{\partial w_{11}^{(1)}} &= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_{11}^{(1)}} \\ &= (\hat{y} - y) \cdot w_{11}^{(L)} w_{11}^{(L-1)} \cdots w_{11}^{(2)} x_1. \end{aligned}$$

Notice that the partial derivative is a product that tacks on weights, one from each layer, starting with the second layer. Now if L is large and these weights were all slightly less than 1, then resulting product would be very small; if these weights are slightly larger than 1, then the product would be a large quantity. The first case is the *vanishing gradient* problem; the second is the *exploding gradient* problem. Note that is does not matter which weight we use to differentiate the loss function. The partial derivative product term picks up weights form each layer (except the one wrt which we are differentiating).

2 LSTMs

These notes are based on [1]. LSTMs were developed in order to circumvent the vanishing gradient problem that plagues multi-layered RNNs. LSTMs are equipped with a long-term memory and a short-term working memory.

Let $x_t \in \boldsymbol{R}^p$ denote the input at time t; let $\text{ltm}_t \in \boldsymbol{R}^d$ and $\text{wm}_t \in \boldsymbol{R}^d$ denote, respectively, the long-term memory and the working memory available to the LSTM cell at time t.

Updating the long-term memory. In order to update the long-term memory at step t, the LSTM first figures out what to remember from the long-term memory of the last step $t-1$.

$$\text{rem}_t = \sigma\left(W_r \cdot x_t + U_r \cdot \text{wm}_{t-1} + b_r\right). \tag{1}$$

This is accomplished using a one-layer neural network with a sigmoid activation function that estimates the weight matrices $W_r \in \boldsymbol{R}^{p\times d}$, $U_r \in \boldsymbol{R}^{d\times d}$ and the bias vector $b_r \in \boldsymbol{R}^d$. Since the activation function is sigmoid, the components of rem_t are between 0 and 1. If a component is closer to 1, we would want to remember it; if it is close to 0, then we want to forget it.

It next calculates a "candidate" vector to add to its long-term memory. This is done using a single-layer neural network with a tanh activation function. Denote this candidate by ltm'_t.

$$\text{ltm}'_t = \tanh\left(W_l \cdot x_t + U_l \cdot \text{wm}_{t-1} + b_l\right). \tag{2}$$

As usual, $W_l \in \boldsymbol{R}^{p\times d}$, $U_l \in \boldsymbol{R}^{d\times d}$ and $b_l \in \boldsymbol{R}^d$.

Not all parts of this candidate vector may be worth remembering. As such, a save_t vector is created using another single-layer neural network with a sigmoid activation function.

$$\text{save}_t = \sigma\left(W_s \cdot x_t + U_s \cdot \text{wm}_{t-1} + b_s\right). \tag{3}$$

The dimensions of the weight matrices W_s, U_s and the bias vector b_s are such that $\text{save}_t \in \boldsymbol{R}^d$. Now the long-term component of the cell is computed using:

$$\text{ltm}_t = \text{rem}_t \odot \text{ltm}_{t-1} + \text{save}_t \odot \text{ltm}'_t, \tag{4}$$

where $\odot$ represents component-wise multiplication of the d-dimensional vectors.

Updating the working memory. To do this, the LSTM first calculates what parts of the long-term memory it currently wants to focus on. It uses another single-layer neural network with a sigmoid activation to calculate $\text{focus}_t \in \boldsymbol{R}^d$.

$$\text{focus}_t = \sigma\left(W_f \cdot x_t + U_f \cdot \text{wm}_{t-1} + b_f\right). \tag{5}$$

It then updates its working memory using:

$$\text{wm}_t = \text{focus}_t \odot \tanh(\text{ltm}_t). \tag{6}$$

$$\begin{aligned}
\text{rem}_t &= \sigma\left(W_r \cdot x_t + U_r \cdot \text{wm}_{t-1} + b_r\right) \\
\text{save}_t &= \sigma\left(W_s \cdot x_t + U_s \cdot \text{wm}_{t-1} + b_s\right) \\
\text{focus}_t &= \sigma\left(W_f \cdot x_t + U_f \cdot \text{wm}_{t-1} + b_f\right) \\
\text{ltm}'_t &= \tanh\left(W_l \cdot x_t + U_l \cdot \text{wm}_{t-1} + b_l\right) \\
\text{ltm}_t &= \text{rem}_t \odot \text{ltm}_{t-1} + \text{save}_t \odot \text{ltm}'_t \\
\text{wm}_t &= \text{focus}_t \odot \tanh(\text{ltm}_t).
\end{aligned}$$

Figure 1: All the LSTM equations at once.

References

[1] Edwin Chen. Blog post at `http://blog.echen.me/2017/05/30/exploring-lstms/`.