RNN vanishing/exploding gradient problem

Training Deep Networks is hard: Review

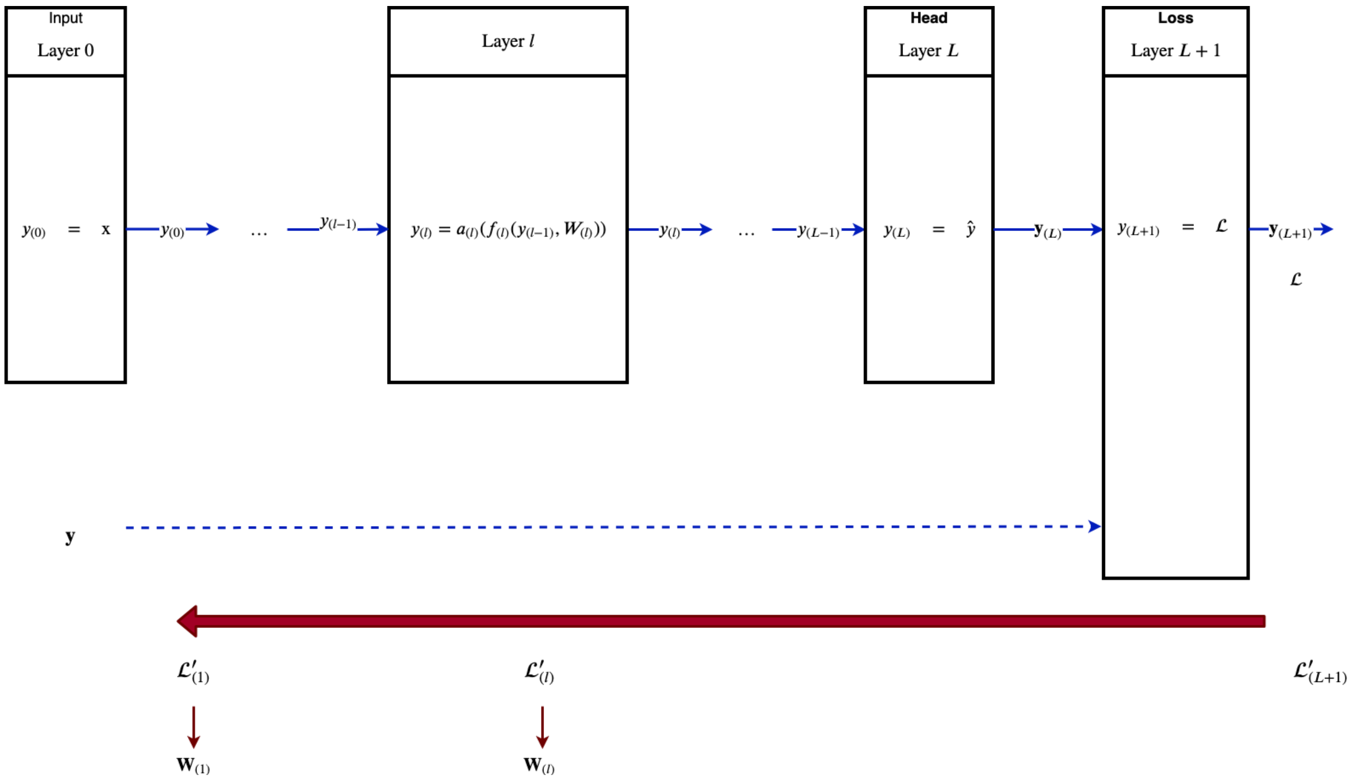
As we learned in the module on Vanishing and Exploding Gradients

- Training a very deep (many layer) network is difficult
- Because as the gradient flows backwards (from Loss layer to Input layer)
- The Loss Gradients successively either diminish or expand

Let's quickly review the issue of vanishing and exploding gradients.

Here is the picture of gradient flow during Back propagation:

Backward pass: Loss to Weights



The Loss Gradient of layer l

$$\mathcal{L}'_{(l)} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{(l)}}$$

flows backwards from Loss Layer $(L+1)$ inductively as:

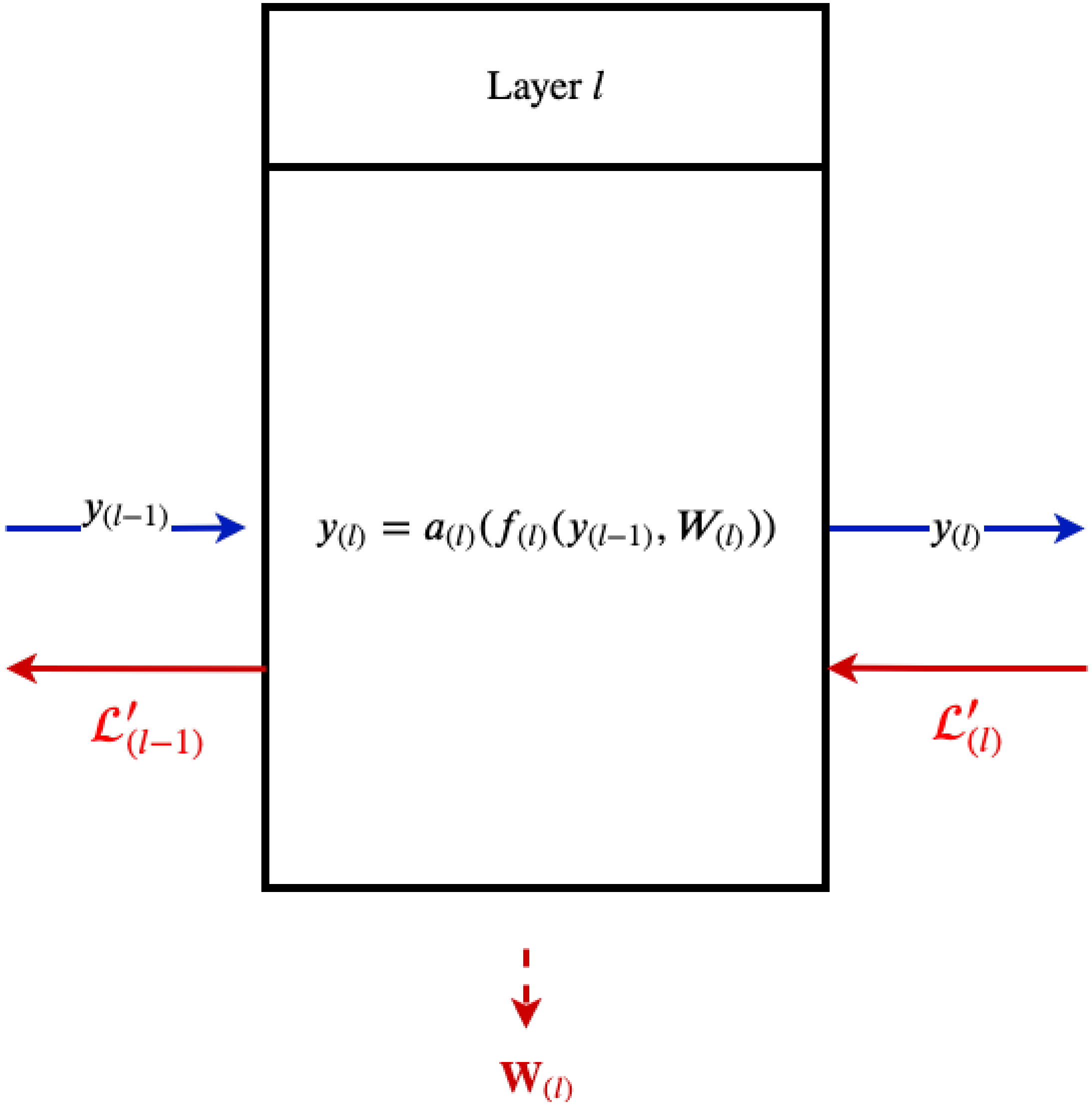
$$\begin{aligned} \mathcal{L}'_{(l-1)} &= \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{(l-1)}} \\ &= \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{(l)}} \frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{y}_{(l-1)}} \\ &= \mathcal{L}'_{(l)} \frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{y}_{(l-1)}} \end{aligned}$$

Moreover, from the Loss Gradient and a local gradient $\frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{W}_{(l)}}$ at layer l

- We can compute the derivative of the loss with respect to the layer's weights
- Which is used in the update equation for Gradient Descent
- To modify the estimate of the layer's weights
- In the direction of decreasing Loss

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{(l)}} \frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{W}_{(l)}} = \mathcal{L}'_{(l)} \frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{W}_{(l)}}$$

Forward and Backward pass: Detail



The issue arises in the second term $\frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{y}_{(l-1)}}$ of the inductive update of the Loss Gradient

$$\mathcal{L}'_{(l-1)} \quad = \quad \mathcal{L}'_{(l)} \frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{y}_{(l-1)}}$$

Since

$$\mathbf{y}_{(l)} = a_{(l)} \left(f_{(l)}(\mathbf{y}_{(l-1)}, \mathbf{W}_{(l)}) \right)$$

The derivative

$$\frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{y}_{(l-1)}} = a'_{(l)} f'_{(l)}$$

where

$$
\begin{aligned}
a'_{(l)} &= \frac{\partial a_{(l)}(f_{(l)}(\mathbf{y}_{(l-1)}, \mathbf{W}_{(l)}))}{\partial f_{(l)}(\mathbf{y}_{(l-1)}, \mathbf{W}_{(l)})} && \text{derivative of } a_{(l)}(\ldots) \text{ wrt } f_{(l)}(\ldots) \\
f'_{(l)} &= \frac{\partial f_{(l)}(\mathbf{y}_{(l-1)}, W_{(l)})}{\partial \mathbf{y}_{(l-1)}} && \text{derivative of } f_{(l)}(\ldots) \text{ wrt } \mathbf{y}_{(l-1)}
\end{aligned}
$$

Substituting the value of the loss gradient into the backward update rule:

$$\begin{aligned}\mathcal{L}'_{(l-1)} &= \mathcal{L}'_{(l)} \frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{y}_{(l-1)}} \\ &= \mathcal{L}'_{(l)} a'_{(l)} f'_{(l)}\end{aligned}$$

We see that the backwards step from Loss Gradient of layer l to Loss Gradient of layer $(l-1)$ introduces $a'_{(l)}$ as a multiplicative term.

But as we continue backwards (expanding $\mathcal{L}'_{(l)}$ on the right hand side) we accumulate this multiplicative term

Starting from layer $(L+1)$ and proceeding backwards to layer l, the Loss Gradient term looks like

$$\mathcal{L}'_{(l)} = \mathcal{L}'_{(L+1)} \prod_{l'=l+1}^{L} a'_{(l')} f'_{(l')}$$

Specifically: it is the $a'_{(l)}$ term that is problematic

- If the activation functions $a_{(l)}$ is such that $a'_{(l)} < 1$:
    - The backwards pass attenuates the Loss Gradient
    - Eventually making it go to 0 (disappear)
- If the activation function $a_{(l)}$ is such that $a'_{(l)} > 1$:
    - The backwards pass amplifies the Loss Gradient
        - Eventually making it go to ∞ (explode)

Recall that

- For $a_{(l)} = \sigma$ (the sigmoid function)
- $\max_z a'_{(l)}(z) = 0.25$

so using the sigmoid as the default activation

- Made training of deep networks very difficult
- Which stifled progress in Deep Learning

An unrolled RNN is a Deep Network

If we unroll an RNN that has an input sequence of length T

$$\mathbf{x}_{(1)}, \ldots, \mathbf{x}_{(T)}$$

we wind up with a network of T layers (plus the Loss layer)

RNN Loss Gradient Flow

As the input sequence length T gets large

- It should be no surprise that training an RNN
- Is exposed to the problem of vanishing and exploding gradients
- Because of the derivative of the activation function (written as ϕ rather than $a_{(l)}$ in the RNN literature)

But it turns out that there is a *second* source of vanishing/exploding gradients for RNN's:

- The weight matrix $\mathbf{W}$ is shared at every step of the unrolled network

Let's see how this can lead to vanishing/exploding gradients.

Vanishing/Exploding gradients

Let's recall the RNN update equations:

$$\begin{aligned} \mathbf{h}_{(t)} &= \phi(\mathbf{W}_{xh}\mathbf{x}_{(t)} + \mathbf{W}_{hh}\mathbf{h}_{(t-1)} + \mathbf{b}_h) \\ \mathbf{y}_{(t)} &= \mathbf{W}_{hy}\mathbf{h}_{(t)} + \mathbf{b}_y \end{aligned}$$

For simplicity of presentation: we will assume activation function ϕ is the identity function in this section.

Returning to the equation that derives the derivative of the Loss with respect to weights $\mathbf{W}$:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{(l)}} \quad = \quad \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{(l)}} \frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{W}_{(l)}} \quad = \quad \mathcal{L}'_{(l)} \frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{W}_{(l)}}$$

Let's focus on the term

$$\frac{\partial \mathbf{y}_{(t)}^{(\mathbf{i})}}{\partial \mathbf{W}}$$

(replacing l as the index of the layer with t, the time step)

We will focus on the part of $\mathbf{W}$ that is $\mathbf{W}_{hh}$

$$\frac{\partial \mathbf{y}_{(t)}}{\partial \mathbf{W}_{hh}} = \frac{\partial \mathbf{y}_{(t)}}{\partial \mathbf{h}_{(t)}} \frac{\partial \mathbf{h}_{(t)}}{\partial \mathbf{W}_{hh}}$$

This term comes about due to the RNN update equation

$$\mathbf{y}_{(t)} = \mathbf{W}_{hy}\mathbf{h}_{(t)} + \mathbf{b}_y$$

- And $\mathbf{h}_{(t)}$ is a function of $\mathbf{W}_{hh}$

$$\frac{\partial \mathbf{y}^{(\mathbf{i})}_{(t)}}{\partial \mathbf{W}} = \frac{\partial \mathbf{y}^{(\mathbf{i})}_{(t)}}{\partial \mathbf{W}_{hy}} + \frac{\partial \mathbf{y}^{(\mathbf{i})}_{(t)}}{\partial \mathbf{h}_{(t)}} \frac{\partial \mathbf{h}_{(t)}}{\partial \mathbf{W}_{hh}}$$

Let's expand the term

$$\frac{\partial \mathbf{h}_{(t)}}{\partial \mathbf{W}_{hh}}$$

Since

$$\mathbf{h}_{(t)} = \mathbf{W}_{xh}\mathbf{x}_{(t)} + \mathbf{W}_{hh}\mathbf{h}_{(t-1)} + \mathbf{b}_h$$

$\mathbf{h}_{(t)}$ depends on $\mathbf{h}_{(t-1)}$, which by recursion depends on $\mathbf{h}_{(t-2)}$ which $\ldots$ depends on $\mathbf{h}_{(0)}$.

- and all $\mathbf{h}_{(t)}$ share the *same* $\mathbf{W}_{hh}$.

This means that $\mathbf{h}_{(t)}$ depends on $\mathbf{W}_{hh}$ through *each* $\mathbf{h}_{(t-k)}$ for $k = 1, \ldots, t$.

$$\frac{\partial \mathbf{h}_{(t)}}{\partial \mathbf{W}_{hh}} = \sum_{k=1}^{t} \frac{\partial \mathbf{h}_{(t)}}{\partial \mathbf{h}_{(t-k)}} \frac{\partial \mathbf{h}_{(t-k)}}{\partial \mathbf{W}_{hh}}$$

The problematic term for us is

$$\frac{\partial \mathbf{h}_{(t)}}{\partial \mathbf{h}_{(t-k)}}$$

It can be computed by k applications of the Chain Rule as

$$\begin{aligned}
\frac{\partial \mathbf{h}_{(t)}}{\partial \mathbf{h}_{(t-k)}} &= \frac{\partial \mathbf{h}_{(t)}}{\partial \mathbf{h}_{(t-1)}} \frac{\partial \mathbf{h}_{(t-1)}}{\partial \mathbf{h}_{(t-2)}} \cdots \frac{\partial \mathbf{h}_{(t-k+1)}}{\partial \mathbf{h}_{(t-k)}} \\
&= \prod_{u=0}^{k-1} \frac{\partial \mathbf{h}_{(t-u)}}{\partial \mathbf{h}_{(t-u-1)}}
\end{aligned}$$

Each term

$$\frac{\partial \mathbf{h}_{(t-u)}}{\partial \mathbf{h}_{(t-u-1)}}$$

results in a term $\mathbf{W}_{hh}$ because

$$\mathbf{h}_{(t)} = \mathbf{W}_{xh}\mathbf{x}_{(t)} + \mathbf{W}_{hh}\mathbf{h}_{(t-1)} + \mathbf{b}_h$$

So the **repeated product is equal to the matrix $\boldsymbol{W_{hh}}$ raised to the power $\boldsymbol{k}$**

For simplicity, suppose $\mathbf{W}_{hh}$ were a scalar (in general: use eigenvalues of matrices and matrix algebra)

Raising $\mathbf{W}_{hh}$ to the power of k

- Approaches 0 as k increases, when $\mathbf{W}_{hh} < 1$
- Approaches ∞ as k increases, when $\mathbf{W}_{hh} > 1$

In other words:

- As the distance k between time steps increases
- The Loss Gradient tends to either vanish or explode
- Inhibiting weight updates and learning

If updates *do* occur, they will either be

- Erratic (large loss gradients)
- Slow (small loss gradients)

Remember that this cause of vanishing/exploding gradients *is particular to* recurrent layers

- Because of the sharing of weights between time steps

Controlling exploding gradients by clipping

In theory, we can control the explosion by clipping the gradient $\frac{\partial \mathcal{L}}{\partial W_i}$.

We are still left with the vanishing gradient problem.

This means that "vanilla" RNN's have difficulty learning long-term dependencies (i.e., too many steps backward).

Conclusion

Recurrent layers are especially exposed to the problem of Vanishing and Exploding gradients

- As potentially very deep networks in the unrolled form
- Due to sharing weights $\mathbf{W}$ across time steps

We will introduce some architectural innovations in Recurrent layers to ameliorate this problem.

In [3]:

```python
print("Done")
```

```
Done
```