May 10 · 4 min read

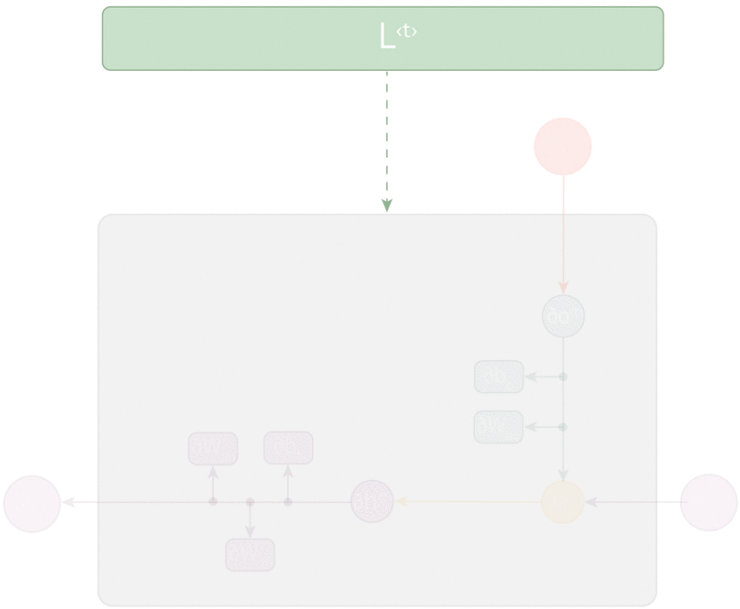
RNN Training: Welcome to your Tape - Side B

Deriving the gradients for Backward propagation in RNN

The first part of this post can be found here.

Back propagation in single RNN Cell

The goal of back propagation in the RNN is to compute the partial derivatives of the weight matrices (`W_xh` , `W_ah` , `W_ao`) and bias vectors (`b_h` , `b_o`) with respect to the final loss `L` .



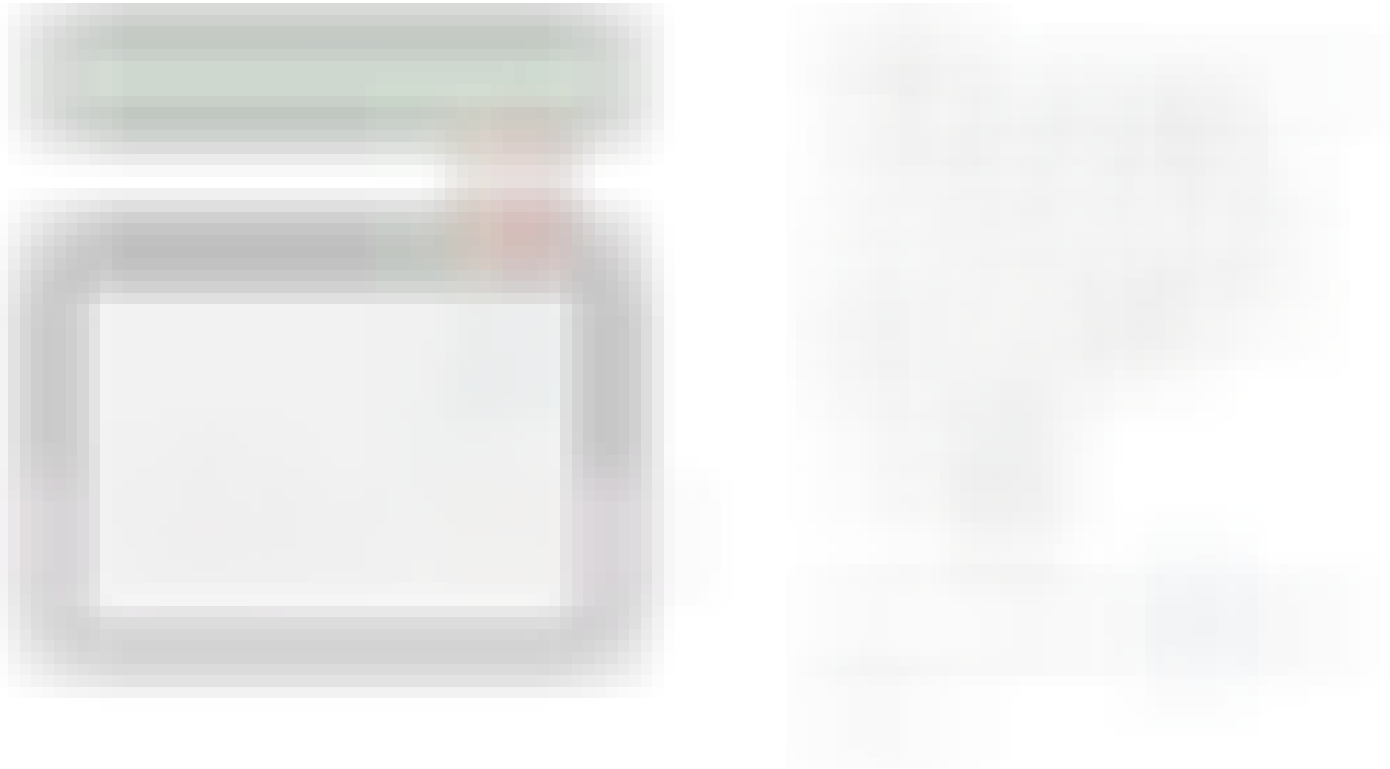
Backprop in single RNN Cell

Deriving the required derivatives is quite straightforward, we just simply compute them using the chain rule formula.

Step 1: The loss function is defined for cost computation. The choice of loss function is usually based on the task at hand, in this case we use the cross entropy loss function for multi-class output $L\langle t \rangle$ and is calculated as below:

Step 2: Next we work our way backwards by computing the partial derivative of the predicted output activation $\hat{y}\langle t \rangle$ with respect to loss $L\langle t \rangle,$ and because the softmax function takes in values from a multi-class output during forward propagation, $\partial\hat{y}\langle t \rangle$ is calculated for class i and other classes k as below:

Step 3: Next, compute the partial derivative of predicted output $o^{\langle t \rangle}$ with respect to loss $L^{\langle t \rangle}$ by using the partial derivative $\partial \hat{y}^{\langle t \rangle}$ for class i and other classes k as below:



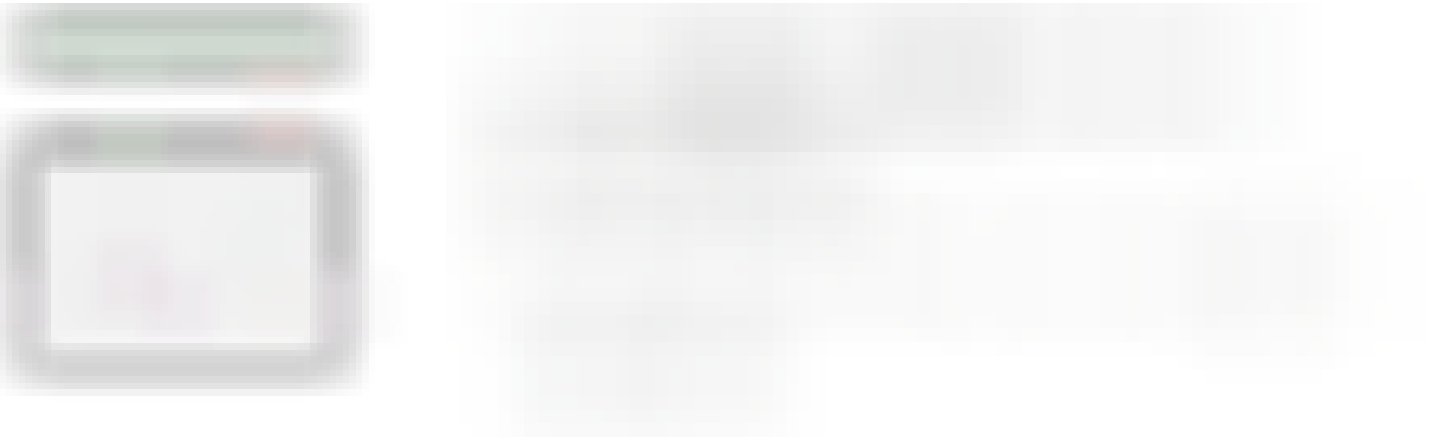
Step 4: Compute the partial derivative of output bias vector b_o with respect to loss $L^{\langle t \rangle}$ by using $\partial o^{\langle t \rangle}$ in the chain rule as below:

Step 5: Compute the partial derivative of hidden-to-output weight matrix `W_ao` with respect to loss `L⟨t⟩` by using `∂o⟨t⟩` in the chain rule as below:

Step 6: Compute the partial derivative of hidden state activation `a⟨t⟩` with respect to loss `L⟨t⟩` by using `∂o⟨t⟩ and ∂h⟨t+1⟩` in the chain rule as below:

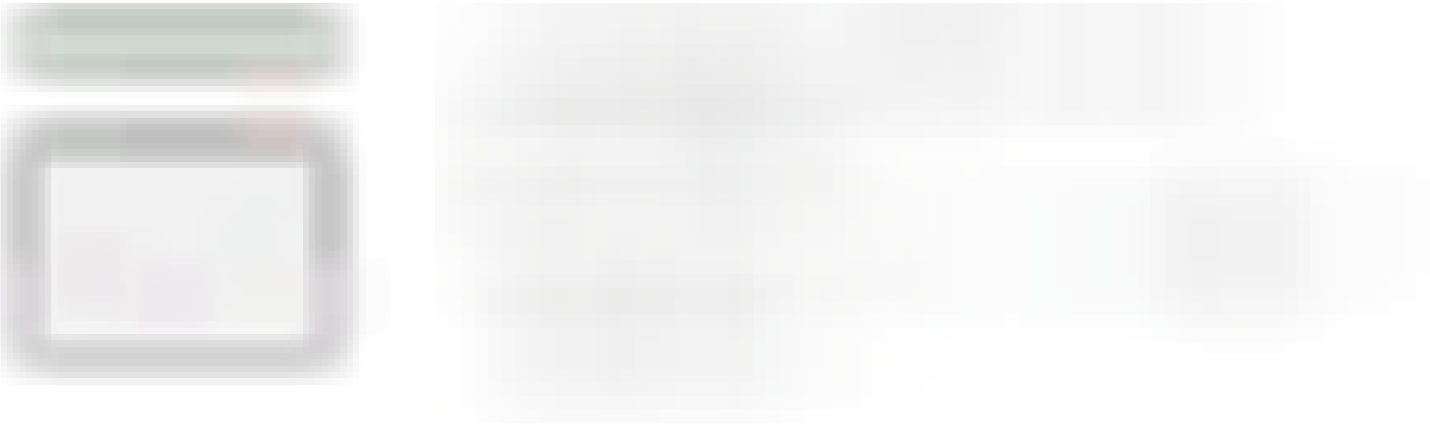
Step 7: Compute the partial derivative of hidden state `h⟨t⟩` with respect to loss `L⟨t⟩` by using `∂a⟨t⟩` in the chain rule as below:

Step 8: Compute the partial derivative of hidden state bias `b_h` with respect to loss `L⟨t⟩` by using `∂h⟨t⟩` in the chain rule as below:



Step 9: Compute the partial derivative of input-to-hidden bias `W_xh` with respect to loss `L⟨t⟩` by using `∂h⟨t⟩` in the chain rule as below:

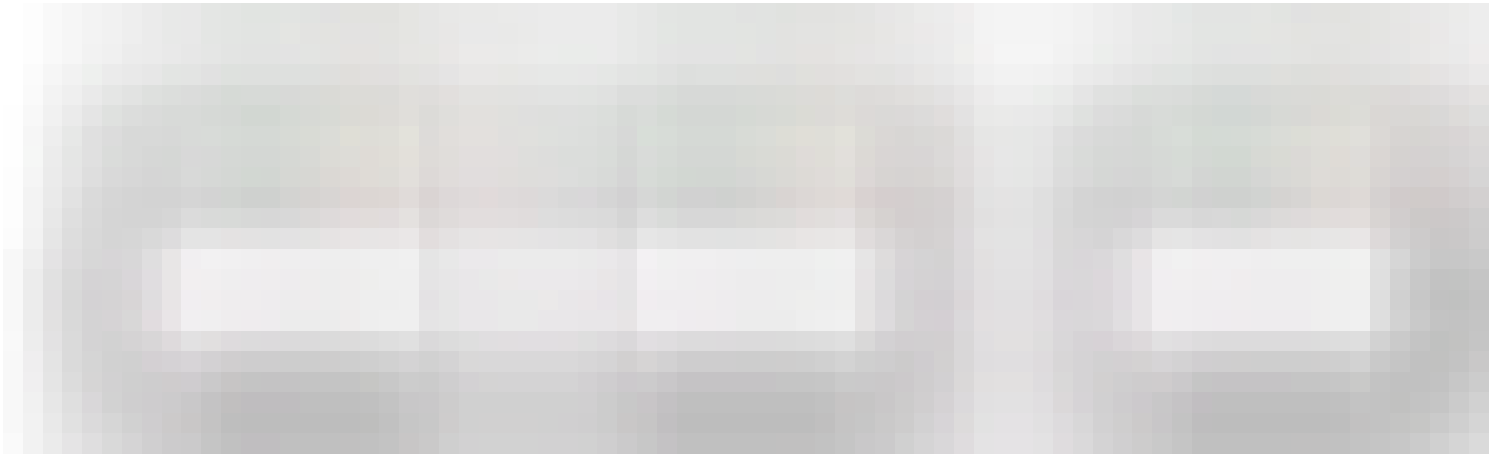
Step 10: Compute the partial derivative of input-to-hidden bias `W_ah` with respect to loss `L(t)` by using `∂h(t)` in the chain rule as below:



Step 11: Finally pass the partial derivative of hidden state `h⟨t⟩` with respect to loss `∂h⟨t⟩` to the previous RNN cell:

Back Propagation Through Time (BPTT)

Just like during forward propagation in the last post, BPTT is also just running the above steps backwards through the whole unrolled recurrent network.



The major difference here is that to update the weights and biases, we have to calculate the sum of each partial derivative $\partial W_{ao}, \partial b_o, \partial W_{ah}, \partial W_{xh}, \partial b_h,$ at every time step *t*, because these parameters are shared across during forward propagation.

Conclusion

In the part I & II of this post we looked at the forward and back propagation steps involved in training a recurrent neural network. Next, we would examine the exploding & vanishing gradient problem in RNNs and solutions developed to fix it.