



Images haven't loaded yet. Please exit printing, wait for images to load, and try to print again.

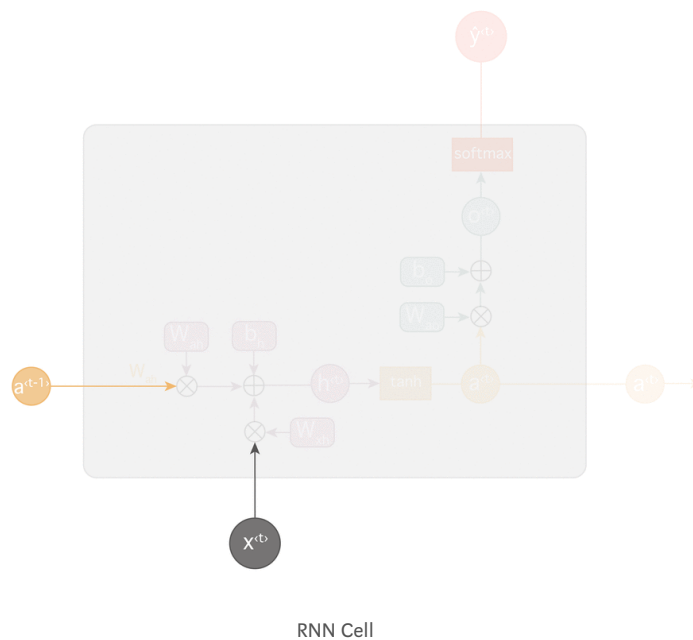
May 9 · 3 min read

RNN Training: Welcome to your Tape - Side A

Forward propagation in RNN

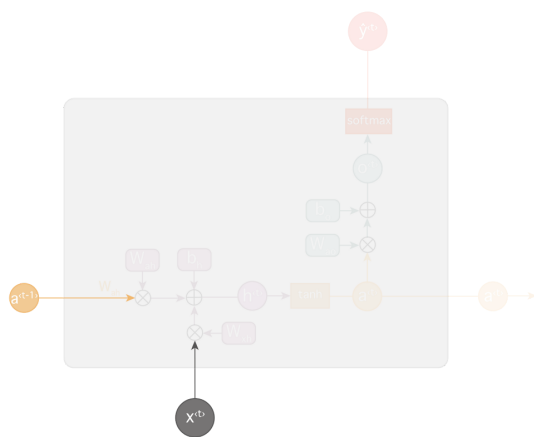
Forward propagation in single RNN Cell

In a previous [post](#), we looked at the basic RNN structure and its unrolling into repetitive chain of cells or time-steps, which we call RNN cells. Below, we would explore the internal structure and forward propagation computations that occur in a single time-step RNN cell.



Lets break down what happens into individual steps:

Step 1: The cell takes in two inputs: $x^{(t)}$ and $a^{(t-1)}$.



Forward Propagation Equations

1. $h^{(t)} = (W_{xh} * x^{(t)}) + (W_{ah} * a^{(t-1)}) + b_h$
2. $a^{(t)} = \tanh(h^{(t)})$
3. $o^{(t)} = (W_{ao} * a^{(t)}) + b_o$
5. $y^{(t)} = \text{softmax}(o^{(t)})$

Inputs:

- $a^{(t-1)}$ - hidden state activation at previous time-step $\langle t-1 \rangle$
- $x^{(t)}$ - input at time-step $\langle t \rangle$

Vector Operators:

- \otimes - matrix multiplication
- \oplus - matrix addition

Activation Functions:

- \tanh - hyperbolic tangent function
- softmax - softmax function

Weight Matrices & Bias Vectors:

- W_{hh} - hidden-to-hidden weight matrix
- W_{xh} - input-to-hidden weight matrix
- b_h - hidden state bias vector
- W_{ho} - hidden-to-output weight matrix
- b_o - output bias vector

Outputs:

- $h^{(t)}$ - hidden state at time-step $\langle t \rangle$
- $a^{(t)}$ - hidden state activation at time-step $\langle t \rangle$
- $o^{(t)}$ - Predicted output at time-step $\langle t \rangle$
- $y^{(t)}$ - Predicted output activation at time-step $\langle t \rangle$

Step 2: Next, it calculates the matrix multiplication \otimes between W_{xh} & $x(t)$ and between W_{ah} & $a(t-1)$. Afterwards, it computes $h(t)$ by adding the above together with the bias term b_h .

Step 3: Following the above, it calculates $a(t)$ by passing $h(t)$ through an activation function such as tanh or relu. In this case we use the \tanh function.

Step 4: The cell outputs $a(t)$ and passes it on to the next time-step cell for computation.

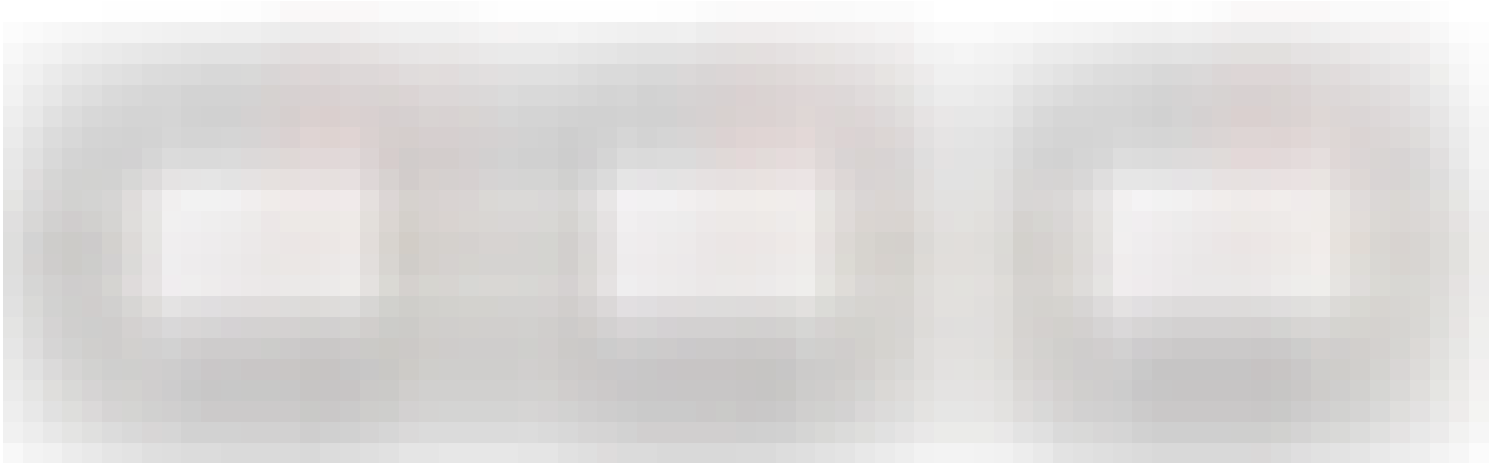
Step 5: Next, it computes $o(t)$; which is the unnormalized log probability of each possible value of the true output value. It does this, by calculating the matrix multiplication \otimes between w_{ao} & $a(t)$ and adding it together with b_o .

Step 6: Finally, it obtains a vector of normalized probabilities $\hat{y}(t)$ over the true output, by passing $o(t)$ through an activation function such as sigmoid or softmax. The choice of output activation function usually depends on the expected output (sigmoid for binary class outputs, softmax for multi-class outputs).

Forward Propagation Through Time

Forward propagation through time is simply running these steps in the whole recurrent network rather than in just a single time-step RNN cell. It begins with the initialization of a hidden state $a(0)$, sharing of the weights and bias vectors w_{xh} , w_{ah} , w_{ao} , b_h , b_o across all time steps $t = 1$ to T , and repeating the steps above in each time step.

For example, if we have an 8 sequence input, $x(1), x(2), \dots, x(8)$ then forward propagation for this network would be steps 1–6 repeated in a loop 8 times.



The above is the first step in training RNNs, in the second part of this post we would look at the back propagation through time (BPTT) algorithms and how to derive its gradients.

