RNNs and LSTMs

*https://fr.slideshare.net/slideshow/recurrent-neural-networks-rnn-249991296/249991296
*https://karpathy.github.io/2015/05/21/rnn-effectiveness/
*https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Architecture for an RNN

- **Activation function:** tanh at hidden layers, softmax at output layer

$$a_t = W_H h_{t-1} + W_X X_t$$ Hidden Nodes

Activation Function

Output From Hidden State → $h_t = tanh(a_t)$ (Hidden State)

Prediction at time t → $y_t = softmax(W_Y h_t)$

2. Compute the loss using cross entropy: $L_t(y_t, \hat{y}_t) = -y_t log(\hat{y}_t)$

3. Back propagation: compute the gradient (error derivatives) at each time step.

- Update the weights to minimize the loss: $W_i := W_i - \eta \frac{\partial L_{total}(y, \hat{y})}{\partial W_i}$

Function dependencies with respect to W_Y

$$L_t = -y_t log(\hat{y}_t) \rightarrow \hat{y}_t = softmax(z_t) \rightarrow z_t = W_Y h_t$$

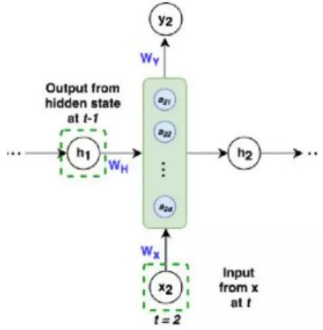
Chain rule with respect to W_Y

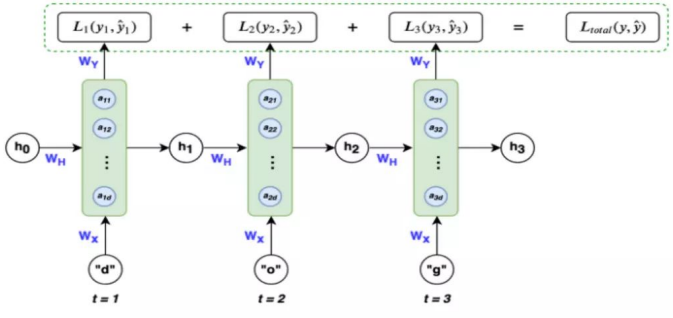
$$\frac{\partial L_t}{\partial W_Y} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial z_t} \frac{\partial z_t}{\partial W_Y}$$

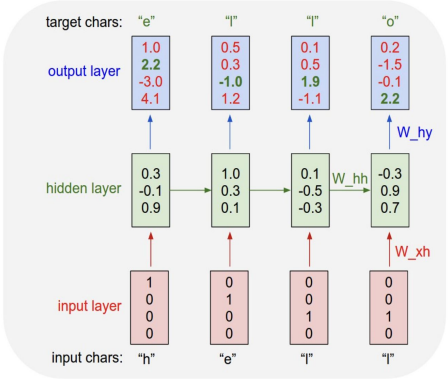
Gradient for W_Y

$$\frac{\partial L_{total}}{\partial W_Y} = \frac{\partial L_1}{\partial \hat{y}_1} \frac{\partial \hat{y}_1}{\partial z_1} \frac{\partial z_1}{\partial W_Y} + \frac{\partial L_2}{\partial \hat{y}_2} \frac{\partial \hat{y}_2}{\partial z_2} \frac{\partial z_2}{\partial W_Y} + \frac{\partial L_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial z_3} \frac{\partial z_3}{\partial W_Y} = \sum_{t=1}^{n} \frac{\partial L_t}{\partial W_Y}$$

For hidden state at t=2, input is the output from t-1 and x at t

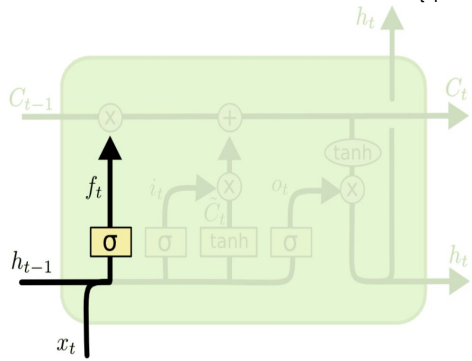






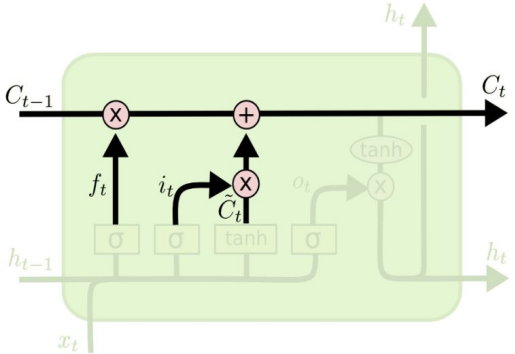
An example RNN with 4-dimensional input and output layers, and a hidden layer of 3 units (neurons). This diagram shows the activations in the forward pass when the RNN is fed the characters "hell" as input. The output layer contains confidences the RNN assigns for the next character (vocabulary is "h,e,l,o"); We want the green numbers to be high and red numbers to be low.

Forget layer: Get rid of some of C_{t-1} information based on h_{t-1} and x_t



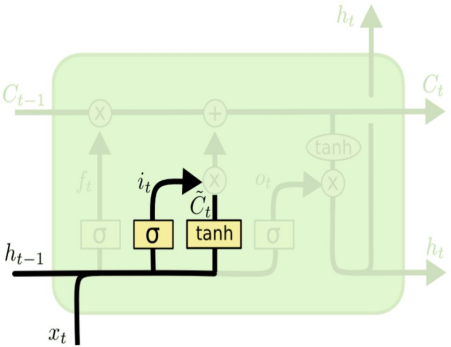
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Update the cell state



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

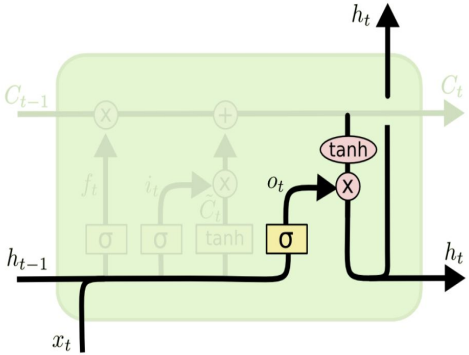
Input gate layer: What new do we want to add (tanh) and decide what do we want to get rid of from it (sigmoid)



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Output: Filtered version of the current cell state. Decide what do we keep (sigmoid) and combine (tanh and x)



$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Notes:

- LSTM were introduced to solve RNN's lack of long-term dependencies (vanishing/exploding gradient problem)
- C_0 is initialized with zeros or small values