LSTM:

9 weight matrices and 5 bias vectors to learn

$x^t = (d_{in}, N)$

$h^t = (d_h, N)$

	Act as embedding on input token	Act on previous timestep
Remember:	W_r, B_r	U_r
New:	W_n, B_n	U_n
output:	W_o, B_o	U_o
Content:	$W_{c_{temp}}, B_{c_{temp}}$	$U_{c_{temp}}$
classify:	W_y, B_y	

Dimensions:

d_{in}: input dim of token

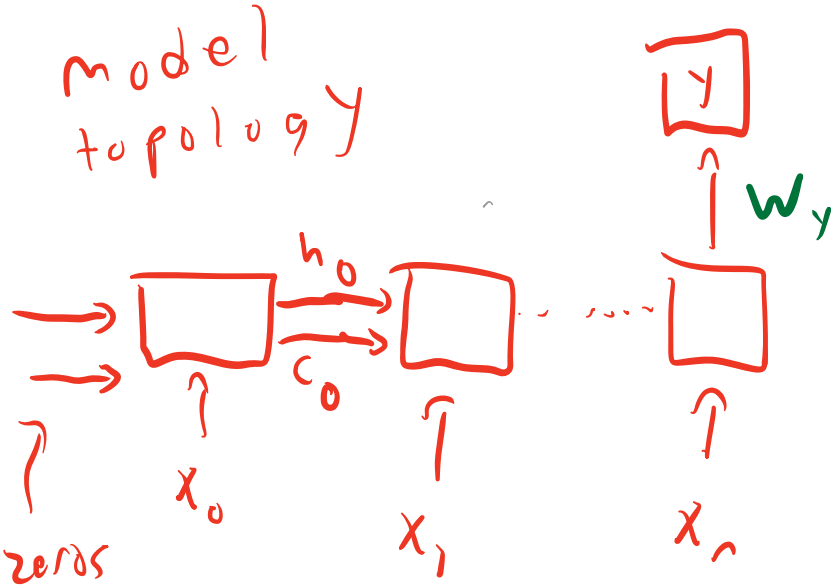
N: batch size

d_h: hidd dim

d_y: number output labels

d_s: seq length

model topology



Variables to learn:

$W_{r,n,o}$ dim $= (d_h, d_{in})$

$B_{r,n,o} = (d_h, 1)$

U dim $= (d_h, d_h)$

W_y dim: (d_y, d_h)

B_y dim $(d_y, 1)$

Forward Pass: computing units

Compute For $t=0$ to $t=d_s$ (whole sequence)

(d_h, N)

current Content: $c^t_{temp} = \tanh\left(W_{c_{temp}} x^t + U_{c_{temp}} h^{t-1} + B_{c_{temp}}\right)$

Remember: $r^t = \sigma\left(W_r x^t + U_r h^{t-1} + B_r\right)$

New: $n^t = \sigma\left(W_n x^t + U_n h^{t-1} + B_n\right)$

Content: $c^t = r^t \circ c^{t-1} + n^t \circ c^t_{temp}$

output: $o^t = \sigma\left(W_o x^t + U_o h^{t-1} + B_o\right)$

hidden: $h^t = o^t \circ \tanh\left(c^t\right)$

Get label after sequence (whole sequence)

linear output: $X_{out} = W_y h^{d_s - 1} + B_y$

label distribution: $\hat{y} = \text{Softmax}\left(X_{out}\right)$

Backwards Pass:

dim (d_y, N)

Let output be Y where Y_i is one-hot encoding of correct label of sample i. (samples are columns)

Let's use categorical cross-entropy loss. The avg loss for given batch is:

$$L = -\frac{1}{N} \sum_{i=0}^{N} \sum_{c=0}^{d_y} Y_{c,i} \cdot \log\left(\hat{Y}_{c,i}\right)$$

Now with softmax and cross-entropy loss we have that

Output layer:

$$\frac{dL}{dX_{out}} = \underbrace{\hat{Y} - Y}_{(d_y, N)} \cdot \underbrace{\frac{1}{N}}_{\text{think should average over batch}}$$

Label Layers

(d_y, d_h)

- $\frac{dL}{dW_y} = \frac{dL}{dX_{out}} \cdot \left(h^{d_s-1}\right)^T$
- $\frac{dL}{dB_y} = \frac{dL}{dX_{out}} \cdot \mathbb{1}_{(d_h, 1)}$
- $\frac{dL}{dh^{d_s-1}} = (W_y)^T \cdot \frac{dL}{dX_{out}}$ → link to recursive structure (first $\frac{dL}{dh^t}$)

(d_h, N) (d_h, d_y) (d_y, N)

Recursive layers

output:

- $\frac{dL}{do^t} = \frac{dL}{dh^t} \circ \tanh(c^t)$

★ All W, B, V needed to be added each iteration backwards

Content:

- $\frac{dL}{dc^t} = \frac{dL}{dh^t} \circ o^t \circ \left(1 - \tanh(c_t)^2\right)$

- $\frac{dL}{dr^t} = \frac{dL}{dc^t} \circ c^{t-1}$

- $\frac{dL}{dn^t} = \frac{dL}{dc^t} \circ c^t_{temp}$

- $\frac{dL}{dc^t_{temp}} = \frac{dL}{dc^t} \circ n^t$

General Form for W, U, B

For component of type $k \in K$

From $K = \{C_{temp}, \Gamma, \cap, \theta\}$

- $\frac{dL}{dW_k^t} = \left[\underbrace{\frac{dL}{dk^t}}_{\text{upstream}} \circ \underbrace{\left[k^t \circ (1 - k^t)\right]}_{\text{sigmoid activ deriv}}\right] \cdot (x^t)^T$ $\quad (d_h, N) \quad (N, d_{in})$

- $\frac{dL}{dB_k^t} = \left[\frac{dL}{dk^t} \circ \left[k^t \circ (1 - k^t)\right]\right] \cdot \underbrace{1}_{(N,1)}$

- $\frac{dL}{dU_k} = \left[\frac{dL}{dk^t} \circ \left[k^t \circ (1 - k^t)\right]\right] \cdot \underbrace{(h^{t-1})^T}_{(N, d_h)}$

- $\frac{dk^t}{dh^{t-1}} = \left[\underbrace{U_k}_{(d_h, d_h)} \cdot \underbrace{\left[k^t \circ (1 - k^t)\right]}_{(d_h, N)}\right]$ will be used to compute overall $\frac{dL}{dh^{t-1}}$ (below)

Gradient to prior cell

$$\cdot \frac{dL}{dC^{t-1}} = \frac{dL}{dC^{t}} \circ f^{t}$$

→ will need to add to the next passes $\frac{dL}{dC^{t}}$

$$\cdot \frac{dL}{dh^{t-1}} = \frac{dL}{do^{t}} \cdot \frac{do^{t}}{dh^{t-1}} + \frac{dL}{dC^{t}_{temp}} \cdot \frac{dC^{t}_{temp}}{dh_{t-1}} + \frac{dL}{df^{t}} \cdot \frac{df^{t}}{dh_{t-1}} + \frac{dL}{di^{t}} \cdot \frac{di^{t}}{dh_{t-1}}$$

Will serve as starting point for next pass back's $\frac{dL}{dh^{t}}$

What data structs are updated through each pass back through cell?

- Each cell backwards add

$$\frac{dL}{dW_k}, \frac{dL}{dB_k}, \frac{dL}{dU_k} \quad \forall k$$

(add to 12 "global" variables)

- Pass $\frac{dL}{dh^{t-1}}, \frac{dL}{dC^{t-1}}$, but these are over-written each iteration backwards

How to efficiently store all temporary computations which are local to a single iteration through time?

- If we care about memory can just globally store x^t, h^t and recompute k^t on the way back,

LSTM Intuition

Idea is that we start with hidden state and output of previous cell. Then based on previous hidden state and current input token we compute vector of percentages (0, 1) of how much to remember (analogously, forget) of previous output. [Remember] We also create vector of percentages for how much

to incorporate value from current iteration, [New]

We set current iteration using [content]

We apply these percentages and sum the two get the output of the cell.

Moreover we have another vector of percentages of how much we want the current output to contribute to the next hidden state [output]