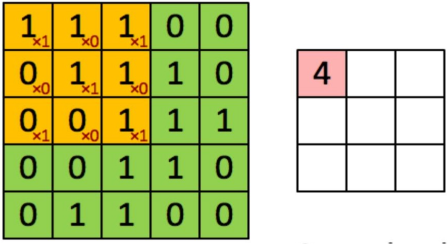
CNN, RNN & LSTM

Himanshu Singh

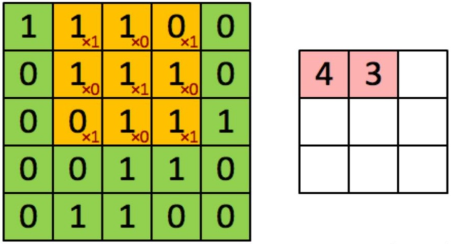
IIT Bombay

Convolution



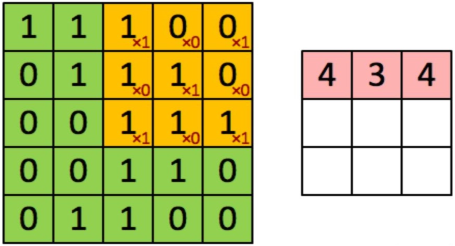
http://deeplearning.stanford.edu/wiki/index.php/Feature$_{e}xtraction_{u}sing_{c}onvolution$

Convolution



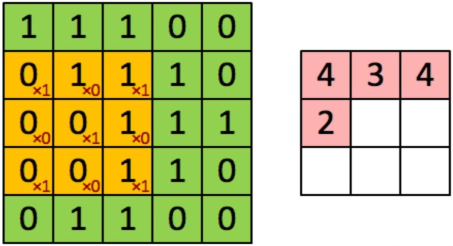
http://deeplearning.stanford.edu/wiki/index.php/Feature$_e$*xtraction*$_u$*sing*$_c$*onvolution*

Convolution



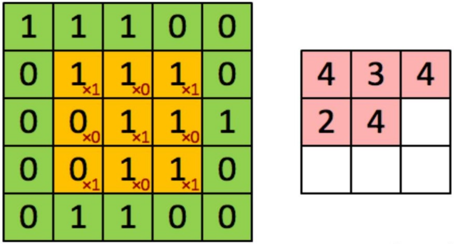
http://deeplearning.stanford.edu/wiki/index.php/Feature$_{e}xtraction_{u}sing_{c}onvolution$

Convolution



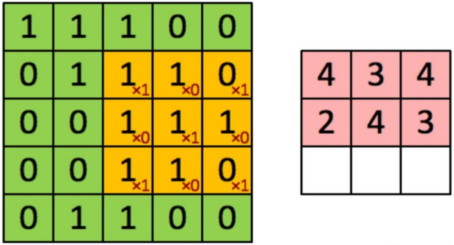
http://deeplearning.stanford.edu/wiki/index.php/Feature$_e$*xtraction*$_u$*sing*$_c$*onvolution*

Convolution



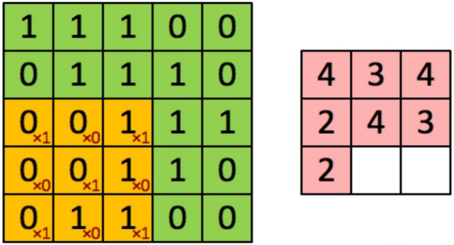
http://deeplearning.stanford.edu/wiki/index.php/Feature$_{e}$*xtraction*$_{u}$*sing*$_{c}$*onvolution*

Convolution



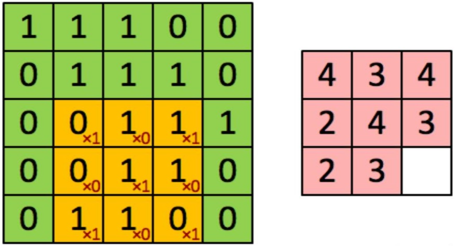
http://deeplearning.stanford.edu/wiki/index.php/Feature$_e$xtraction$_u$sing$_c$onvolution

Convolution



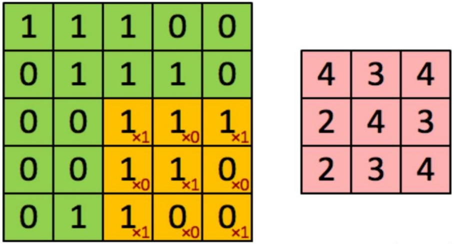
http://deeplearning.stanford.edu/wiki/index.php/Feature$_e xtraction_u sing_c onvolution$

Convolution



http://deeplearning.stanford.edu/wiki/index.php/Feature$_e$*xtraction*$_u$*sing*$_c$*onvolution*

Convolution



http://deeplearning.stanford.edu/wiki/index.php/Feature$_{e}xtraction_{u}sing_{c}onvolution$

Why Convolution?

Averaging each pixel with its neighboring values blurs an image:

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

Taking the difference between a pixel and its neighbors detects edges:

	0	1	0	
	1	-4	1	
	0	1	0	

https://docs.gimp.org/en/plug-in-convmatrix.html

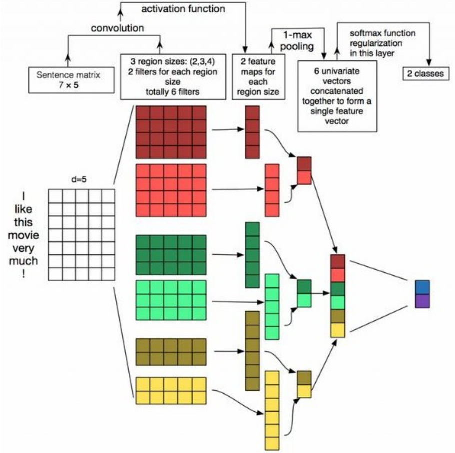
What is Convolutional Neural Network?

CNN

It is several layers of convolutions with nonlinear activation functions like ReLU or tanh applied to the results

- During the training phase, a CNN automatically learns the values of its filters based on the task you want to perform.
- **Location Invariance :** Let's say you want to classify whether or not there's an elephant in an image. Because you are sliding your filters over the whole image you don't really care where the elephant occurs.
- **Compositionality :** Each filter composes a local patch of lower-level features into higher-level representation.

What has CNN for NLP?

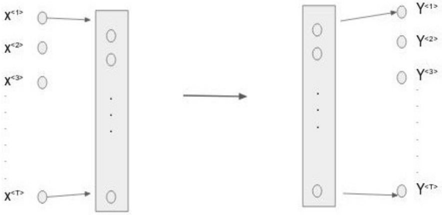


Zhang, Y., Wallace, B. (2015). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification.

Problems with CNN

- **Location Invariance** : You probably do care a lot where in the sentence a word appears unlike images.
- **Local Compositionality** : Pixels close to each other are likely to be semantically related (part of the same object), but the same isn't always true for words. In many languages, parts of phrases could be separated by several other words.
- **Compositional aspect is intuitive in Computer Vision** i.e. edges form shapes and shapes form objects. Clearly, words compose in some ways, like an adjective modifying a noun, but how exactly this works what higher level representations actually "mean" isn't as obvious as in the Computer Vision case.

Why not a traditional neural network for sequential task?



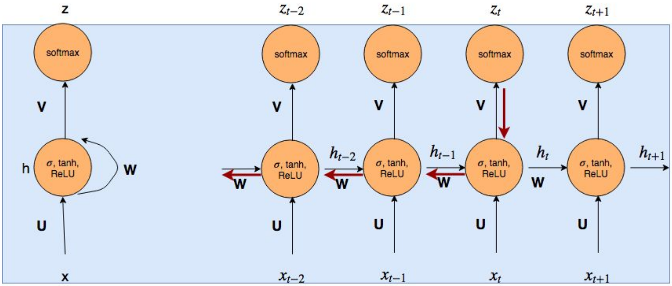
Problems:

- Inputs and ouputs can be of different lengths in different examples
- Traditional NN doesn't share features learned accross different positions of text

Recurrent Neural Network

RNN solves above two problems along with the problems posed by CNNs.

NOTE : Hidden state (h_t) tells us summary of the sequence till time t



Forward pass

$h_t = tanh(Wh_{t-1} + Ux_t + b_h)$

$z_t = softmax(Vh_t + b_z)$

Notation: E(x, y) = -$\sum_t y_t log z_t$
E : above objective function (i.e. sum of errors at all time stamps)
E(t) : to indicate the output at time t

We have, $h_t = tanh(Wh_{t-1} + Ux_t + b_h)$
$z_t = softmax(Vh_t + b_z)$

Gradient of E w.r.t V

let $\alpha_t = Vh_t + b_z$ then
$\frac{\partial E}{\partial V} = \sum_t \frac{\partial E}{\partial \alpha_t} \frac{\partial \alpha_t}{\partial V}$

$\frac{\partial E}{\partial \alpha_t}$ is derivative of softmax function w.r.t it's input α_t
$\frac{\partial E}{\partial \alpha_t} = z_t - y_t$ (cite) and $\frac{\partial \alpha_t}{\partial V} = h_t$

$\frac{\partial E}{\partial V} = \sum_t (z_t - y_t) h_t$

Backpropagation in RNN

We have

$h_t = tanh(Wh_{t-1} + Ux_t + b_h)$

$z_t = softmax(Vh_t + b_z)$

Gradient of E w.r.t W

$\frac{\partial E(t)}{\partial W} = \frac{\partial E(t)}{\partial z_t}\frac{\partial z_t}{\partial W} = \frac{\partial E(t)}{\partial z_t}\frac{\partial z_t}{\partial h_t}\frac{\partial h_t}{\partial W}$

from forward pass equations, h_t partially depends on h_{t-1}

$\frac{\partial E(t)}{\partial W} = \frac{\partial E(t)}{\partial z_t}\frac{\partial z_t}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial W}$

if we keep on substituting h_{t-1} in h_t eqn, we'll see that h_t indirectly depends on h_{t-2}, h_{t-3} ...

$\frac{\partial E(t)}{\partial W} = \sum_{k=1}^{t} \frac{\partial E(t)}{\partial z_t}\frac{\partial z_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W}$, and

$\frac{\partial E}{\partial W} = \sum_t \sum_{k=1}^{t} \frac{\partial E(t)}{\partial z_t}\frac{\partial z_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W}$

We have
$h_t = tanh(Wh_{t-1} + Ux_t + b_h)$
$z_t = softmax(Vh_t + b_z)$

Gradient of E w.r.t U
We can't consider h_{t-1} as constant when taking partial derivative of h_t w.r.t U because h_{t-1} depends on U i.e.
$h_{t-1} = tanh(Wh_{t-2} + Ux_{t-1} + b_h)$

Again, we get a similar form
$\frac{\partial E}{\partial U} = \sum_t \sum_{k=1}^{t} \frac{\partial E(t)}{\partial z_t} \frac{\partial z_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial U}$

Problem with RNN

Look closely to these equations:

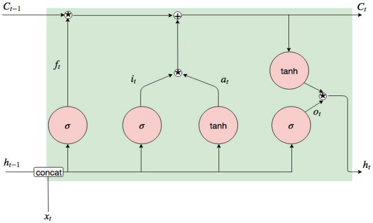
$$\frac{\partial E}{\partial W} = \sum_t \sum_{k=1}^{t} \frac{\partial E(t)}{\partial z_t} \frac{\partial z_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

$$\frac{\partial E}{\partial U} = \sum_t \sum_{k=1}^{t} \frac{\partial E(t)}{\partial z_t} \frac{\partial z_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial U}$$

We find out that $\frac{\partial h_t}{\partial h_k}$ is again a chain rule.

$$\frac{\partial h_t}{\partial h_k} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \ldots \frac{\partial h_{k+1}}{\partial h_k}$$

- If sequence length is large then there will be more number of terms in the product which will result in vanishing gradient problem or exploding gradient problem depending on whether each individual value is less/greater than 1.
- LSTM solves this problem to a large extent.



Forward Pass

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

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

$a_t = tanh(W_a[h_{t-1}; x_t] + b_a)$

$C_t = f_t * C_{t-1} + i_t * a_t$

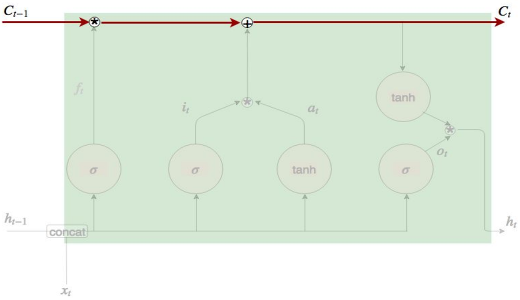
$o_t = \sigma(W_o[h_{t-1}; x_t] + b_o)$

$h_t = o_t * tanh(C_t)$

Cell state in LSTM

Vanishing Gradient Problem Addressed

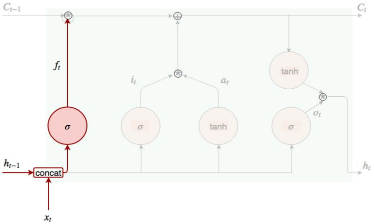
It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged. (Mathematical proof on later slides)



Gates in LSTM

Forget Gate

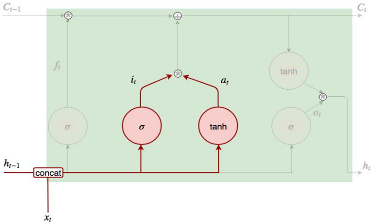
Decides what information should be thrown away from the cell state



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

Input Gate

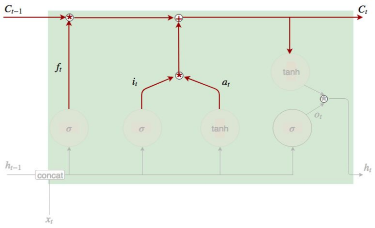
σ layer decides which values to update and a_t is a vector of new candidate values



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

Updating Memory Cell

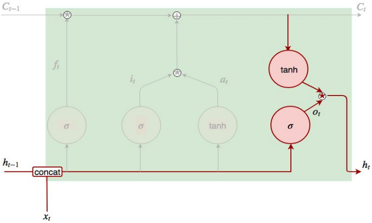
Multiply the old state by f_t, forgetting the things we decided to forget earlier.
Then we add $i_t * a_t$. This is the new candidate values, scaled by how much we decided to update each state value.



$$C_t = f_t * C_{t-1} + i_t * a_t$$

Output Gate

Output will be based on our cell state, but will be a filtered version. Cell state is put through tanh to push the output between -1 and 1

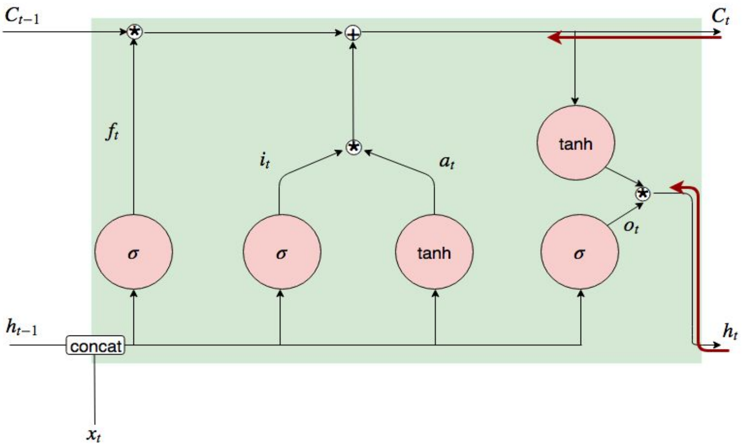


$$o_t = \sigma(W_o[h_{t-1}; x_t] + b_o)$$
$$h_t = o_t * tanh(C_t)$$

Backpropagation in LSTM

Error propagation

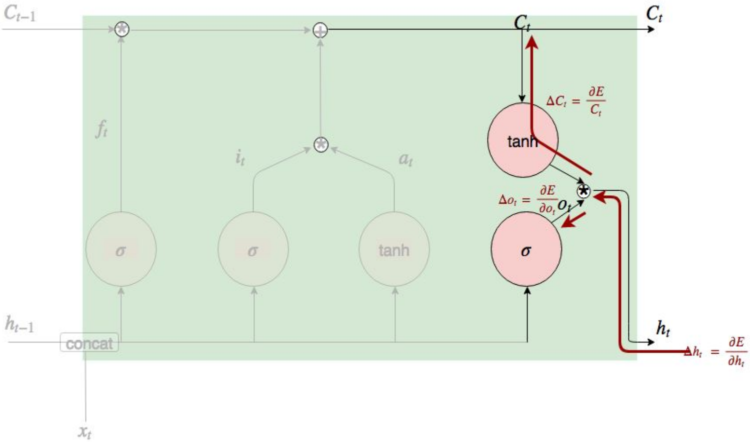
Error propagation happens through C_t and h_t



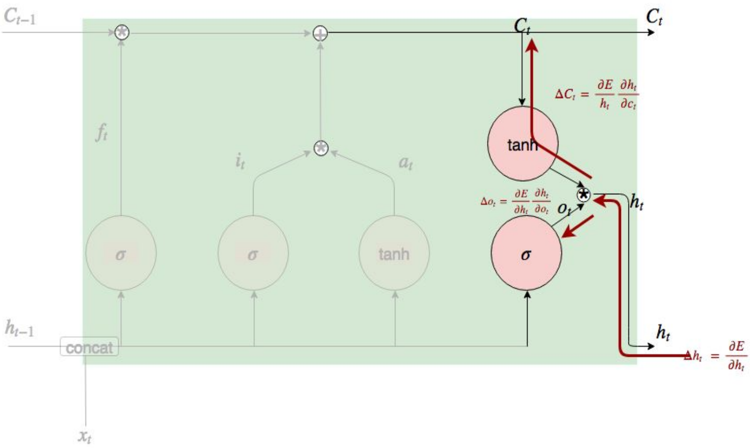
Backpropagation in LSTM

Error propagation

Error propagation through h_t

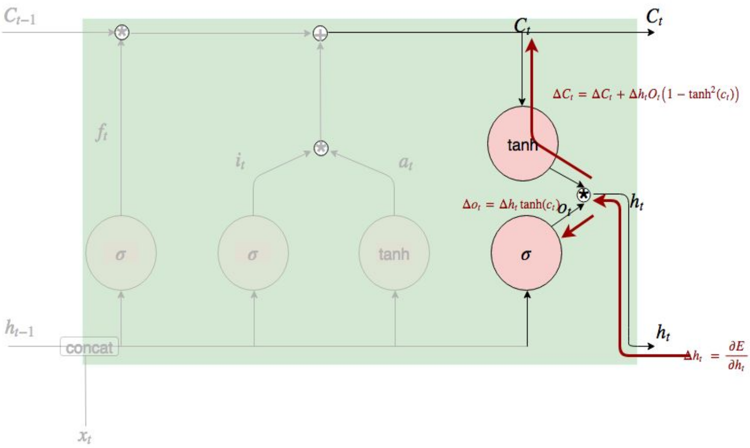


Backpropagation in LSTM



$$o_t = \sigma(W_o[h_{t-1}; x_t] + b_o)$$
$$h_t = o_t * tanh(C_t)$$

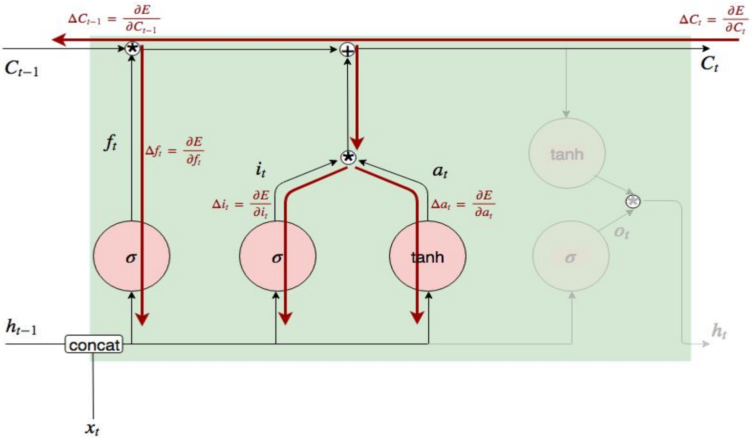
Backpropagation in LSTM

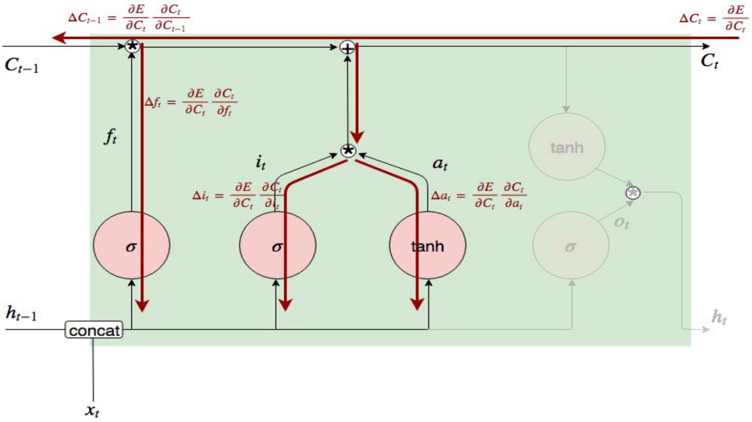


$$o_t = \sigma(W_o[h_{t-1}; x_t] + b_o)$$
$$h_t = o_t * tanh(C_t)$$

Error propagation

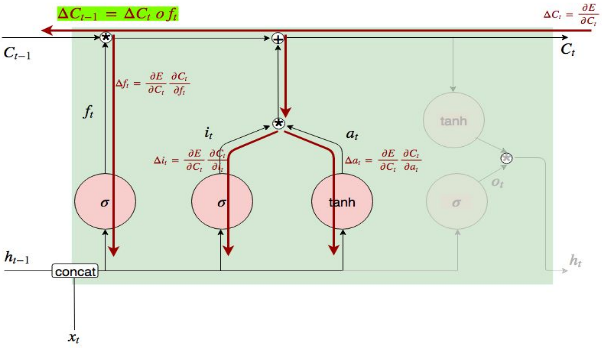
Error propagation through C_t





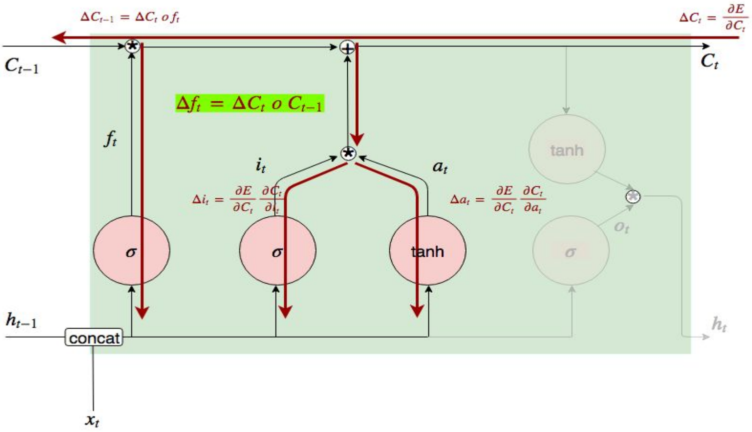
$$C_t = f_t * C_{t-1} + i_t * a_t$$

NOTE : This ΔC_t will be used at $(t-1)^{th}$ timestamp for further error propagation. If f is close to 1 then gradient from t^{th} timestamp is propagated perfectly to $(t-1)^{th}$ timestamp.



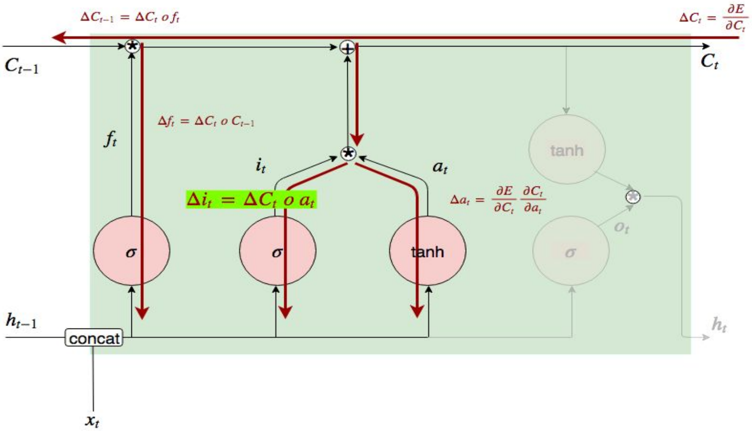
$$C_t = f_t * C_{t-1} + i_t * a_t$$

Backpropagation in LSTM

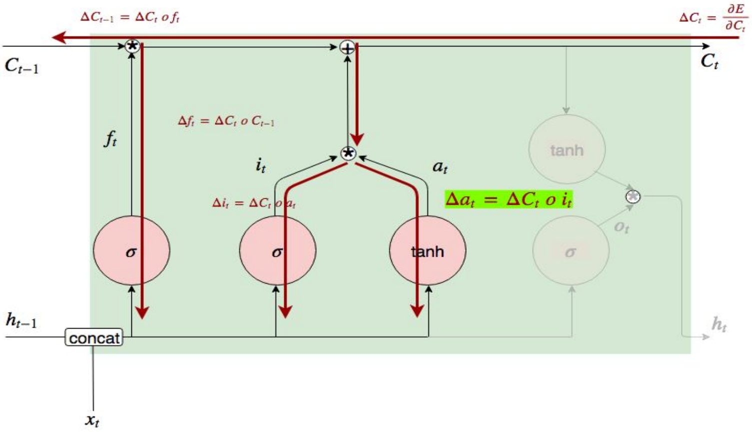


$$C_t = f_t * C_{t-1} + i_t * a_t$$

Backpropagation in LSTM



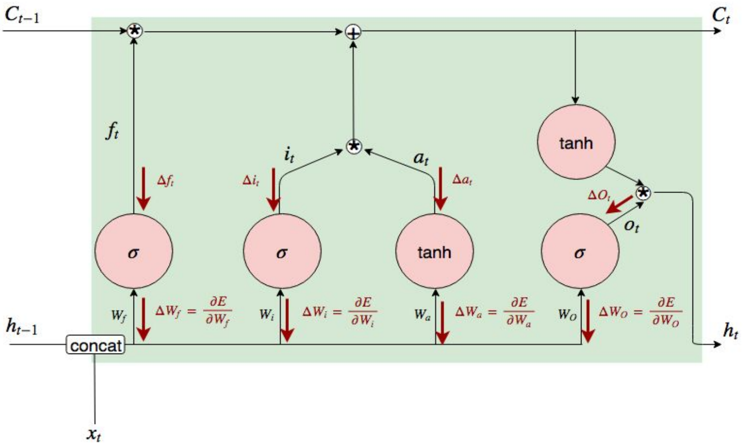
$$C_t = f_t * C_{t-1} + i_t * a_t$$

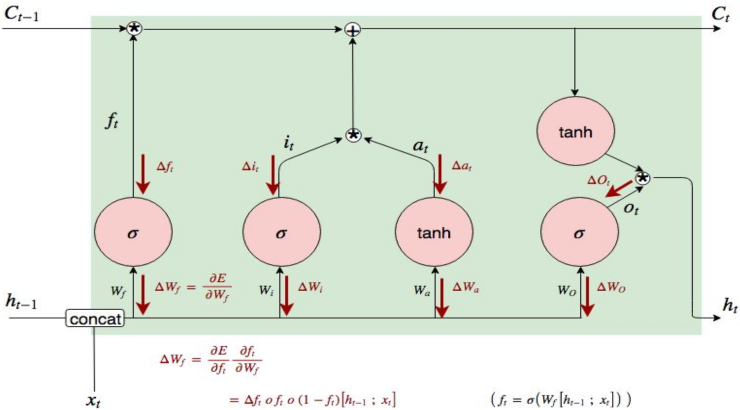


$$C_t = f_t * C_{t-1} + i_t * a_t$$

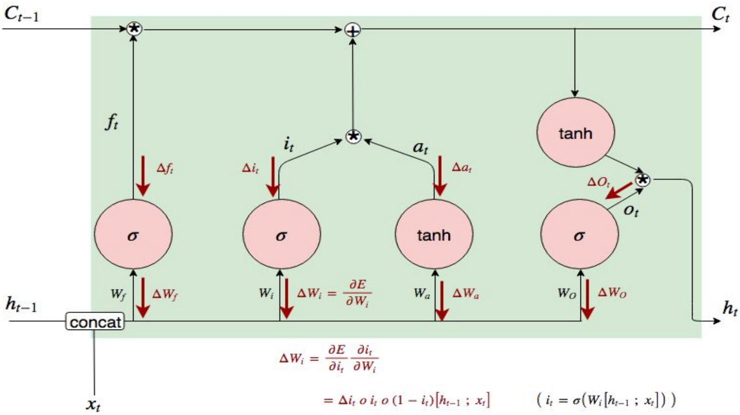
Combined Error

Error propagation from C_t and h_t both



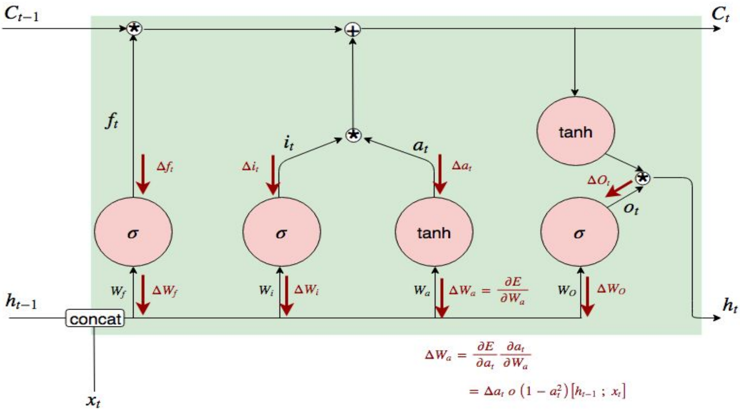

C_{t-1}
C_t
f_t
i_t
a_t
tanh
Δf_t
Δi_t
Δa_t
ΔO_t
o_t
σ
σ
tanh
σ
W_f
$\Delta W_f = \frac{\partial E}{\partial W_f}$
W_i
ΔW_i
W_a
ΔW_a
W_O
ΔW_O
h_{t-1}
concat
h_t
x_t
$\Delta W_f = \frac{\partial E}{\partial f_t} \frac{\partial f_t}{\partial W_f}$
$= \Delta f_t \; o \; f_t \; o \; (1 - f_t)[h_{t-1} \; ; \; x_t]$
$(f_t = \sigma(W_f[h_{t-1} \; ; \; x_t]))$

Backpropagation in LSTM



$$\Delta W_i = \frac{\partial E}{\partial i_t} \frac{\partial i_t}{\partial W_i}$$

$$= \Delta i_t \circ i_t \circ (1 - i_t)[h_{t-1} ; x_t] \qquad (i_t = \sigma(W_i[h_{t-1} ; x_t]))$$

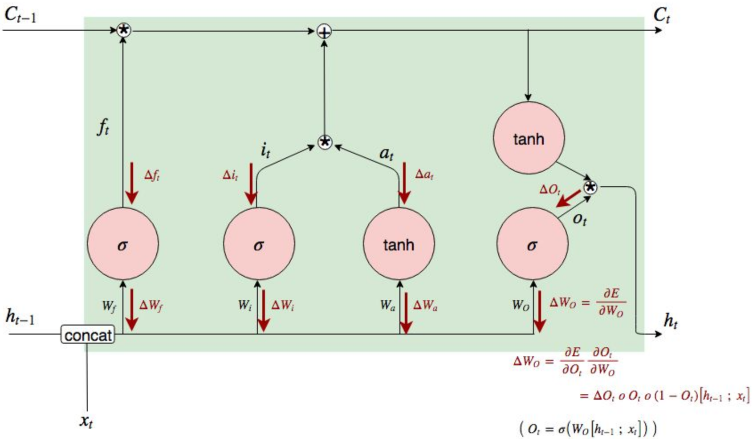


$$\Delta W_a = \frac{\partial E}{\partial a_t} \frac{\partial a_t}{\partial W_a}$$

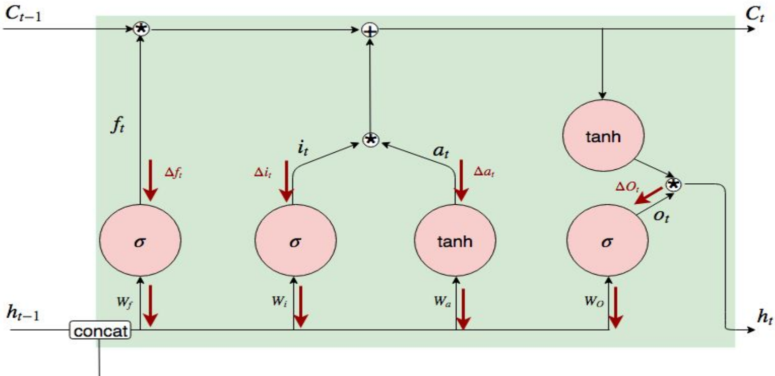
$$= \Delta a_t \, o \, (1 - a_t^2) [h_{t-1} \; ; \; x_t]$$

$$(\, a_t = \tanh(W_a [h_{t-1} \; ; \; x_t]) \,)$$

Backpropagation in LSTM

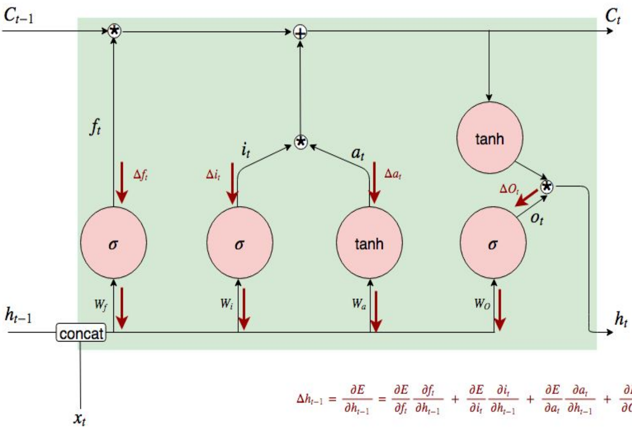


Backpropagation in LSTM



$$\Delta h_{t-1} = \frac{\partial E}{\partial h_{t-1}} = \frac{\partial E}{\partial f_t}\frac{\partial f_t}{\partial h_{t-1}} + \frac{\partial E}{\partial i_t}\frac{\partial i_t}{\partial h_{t-1}} + \frac{\partial E}{\partial a_t}\frac{\partial a_t}{\partial h_{t-1}} + \frac{\partial E}{\partial O_t}\frac{\partial O_t}{\partial h_{t-1}}$$

NOTE : Δh_{t-1} calculated here will be used by previous timestamp for further back propagation



$$\Delta h_{t-1} = \frac{\partial E}{\partial h_{t-1}} = \frac{\partial E}{\partial f_t}\frac{\partial f_t}{\partial h_{t-1}} + \frac{\partial E}{\partial i_t}\frac{\partial i_t}{\partial h_{t-1}} + \frac{\partial E}{\partial a_t}\frac{\partial a_t}{\partial h_{t-1}} + \frac{\partial E}{\partial O_t}\frac{\partial O_t}{\partial h_{t-1}}$$

$$\Delta h_{t-1} = \Delta f_t \, o \, f_t \, o \, (1-f_t) W_{f_h} + \Delta i_t \, o \, i_t \, o \, (1-i_t) W_{i_h} + \Delta a_t \, o \, \left(1-a_t^2\right) W_{a_h} + \Delta O_t \, o \, O_t \, o \, (1-O_t) W_{O_h}$$

$$\left(f_t = \sigma([\, W_{f_h} \, ; \, W_{f_x}][\, h_{t-1} \, ; \, x_t \,])\right)$$

Parameters update

We have calculated ΔW_f, ΔW_i, ΔW_a and ΔW_o.
Next step is to do gradient descent:
$W^* = W^*$ - $\alpha \Delta W^*$ where $* \in f, i, a, o$

References

- colah.github.io/posts/2015-08-Understanding-LSTMs
- www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp
- www.youtube.com/watch?v=KGOBB3wUbdc
- Zhang, Y., Wallace, B. (2015). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification.
- A Gentle Tutorial of Recurrent Neural Network with Error Backpropagation Gang Chen
- www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/