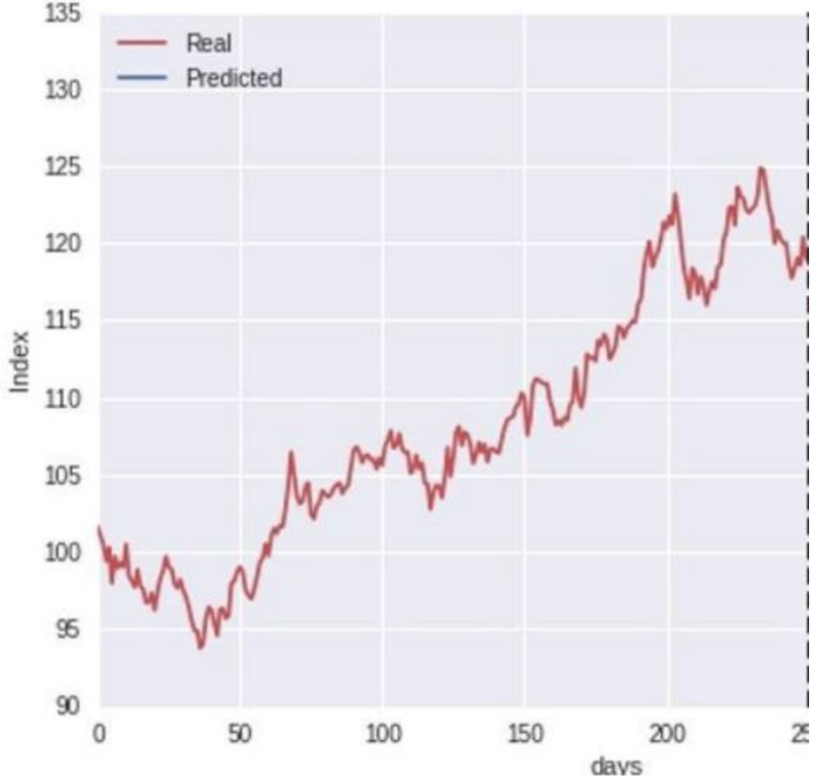
Sequence Handling

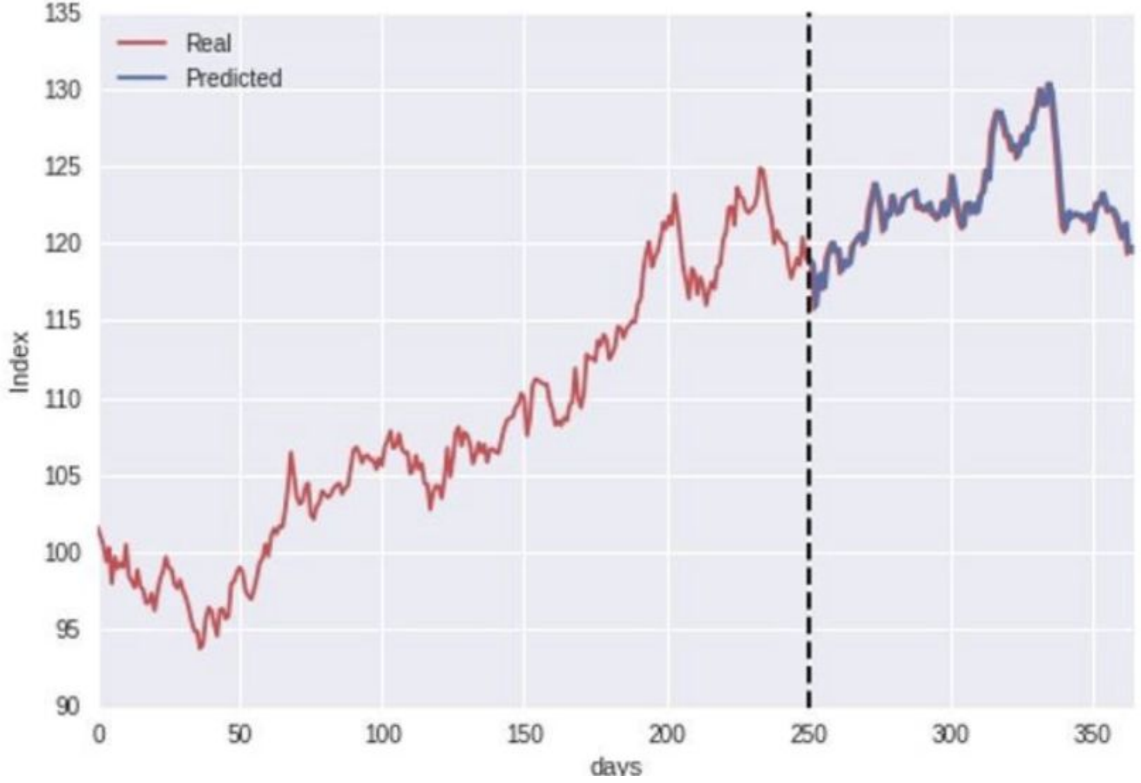
Deep Learning

Aziz Temirkhanov
Lambda, HSE

Time Series Forecasting



Time Series Forecasting



Why is She Scared?

The Knife! (was never in the script)

What is This Sequence Means?

g c g c g g g c t t g t c g t

What is This Sequence Means?

Since the blue part is Pribnow box, a -10 promoter, we know that gtcgt is subject to transcription

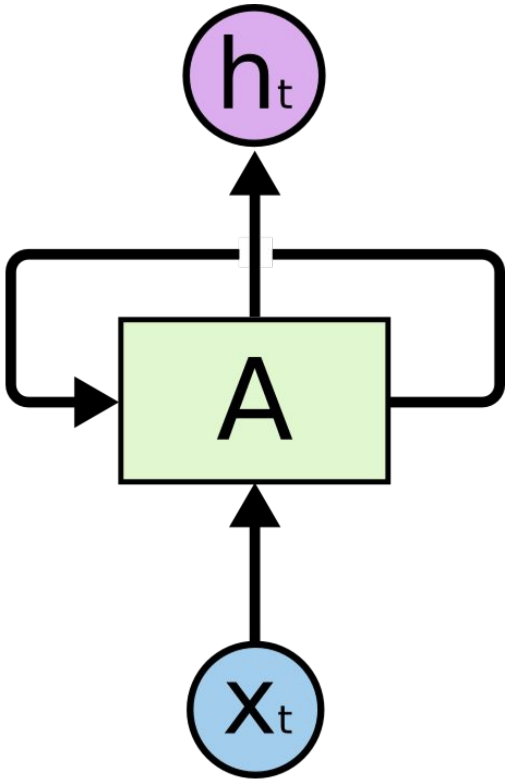
t a t a a t g c g c g g g c t t g t c g t

A Prior Knowledge

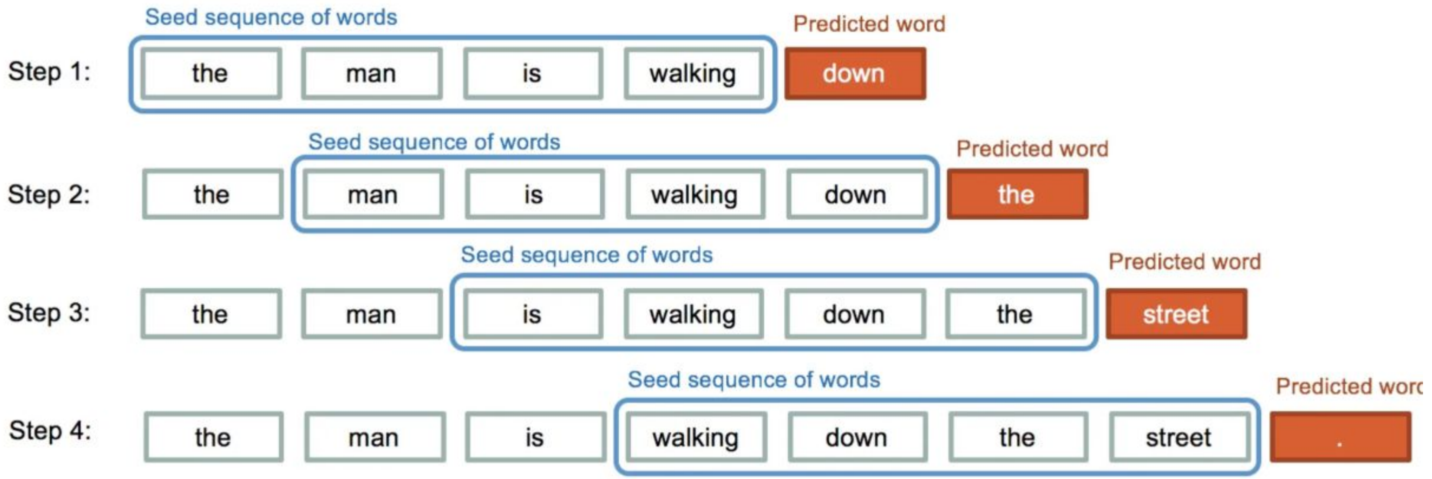
- Every time you think or analyse something, you rarely start from scratch
- You know the previous context in a movie, thus you can understand the facial expression more precisely
- In sequence analysis that often occurs in ML and NLP tasks the ability to take into account aforementioned context is crucial to better results
- But in a standard MLP and CNN that we discussed before you have no methods to do so

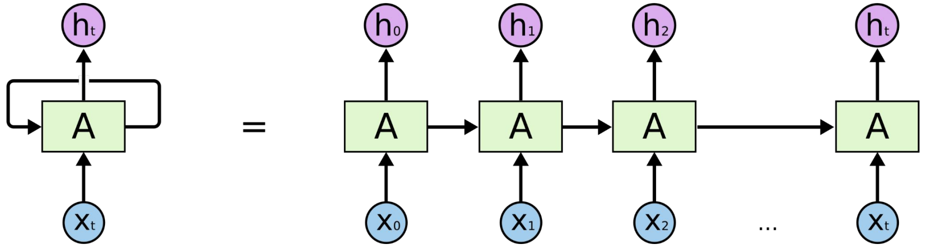
Let’s Fix It!

- Introducing a new way of tackling such information via feeding the output from previous step to the next one
- Alongside with this input we also have an input at the new step t
- This combined input is providing us with useful information about previous states!



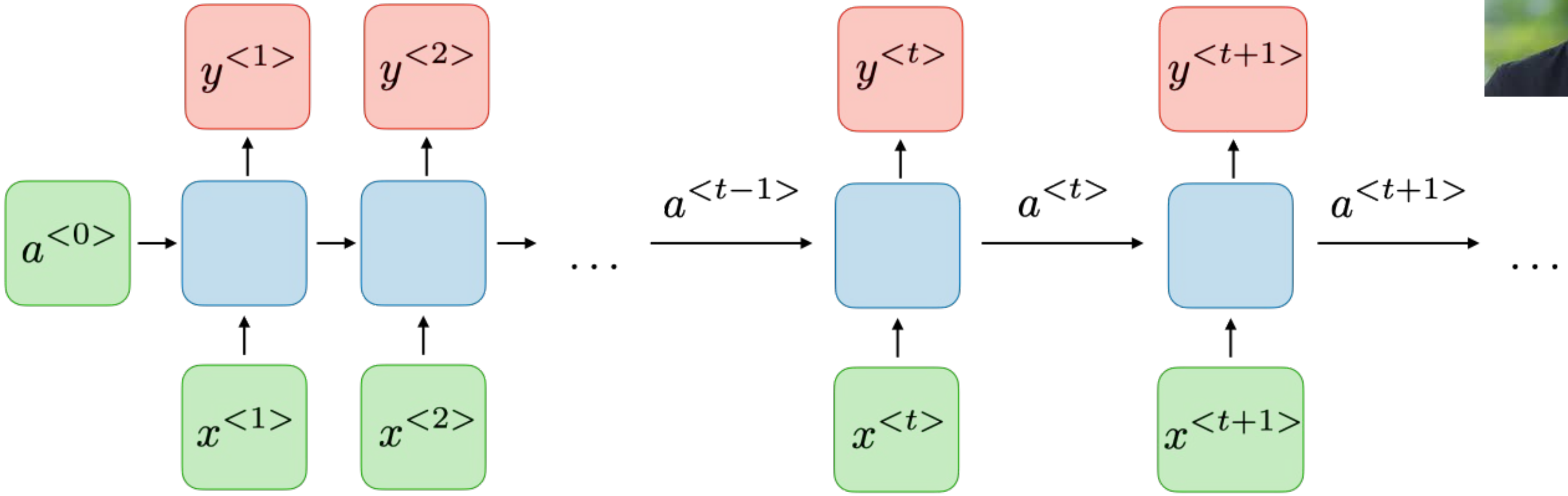
Example


Step 1:
Seed sequence of words
the
man
is
walking
Predicted word
down
Step 2:
the
Seed sequence of words
man
is
walking
down
Predicted word
the
Step 3:
the
man
Seed sequence of words
is
walking
down
the
Predicted word
street
Step 4:
the
man
is
Seed sequence of words
walking
down
the
street
Predicted worc
.


h_t
A
x_t
=
h_0
h_1
h_2
h_t
A
A
A
A
x_0
x_1
x_2
...
x_t

Recurrent Neural Network

- The model that we developed is called RNN



RNN

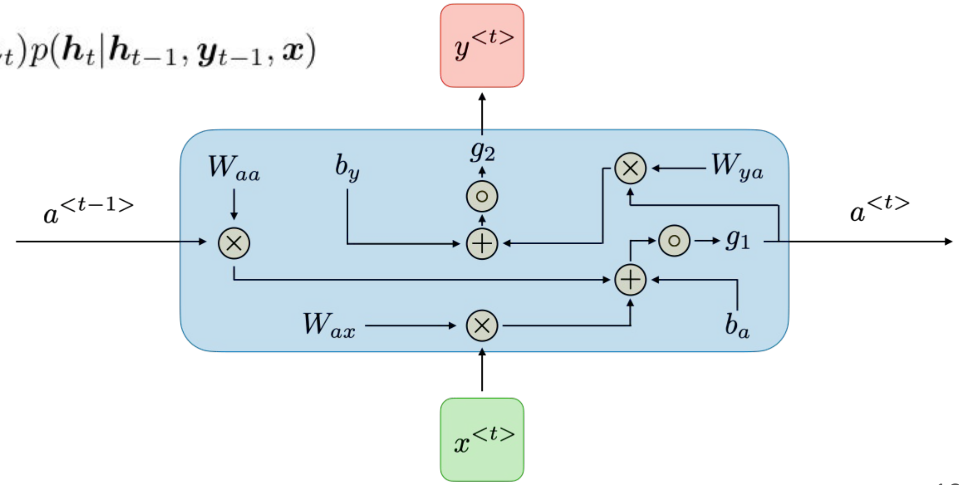
For each timestep t, the activation $a^{<t>}$ and the output $y^{<t>}$ are expressed as follows:

$$\boxed{a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)} \quad \text{and} \quad \boxed{y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)}$$

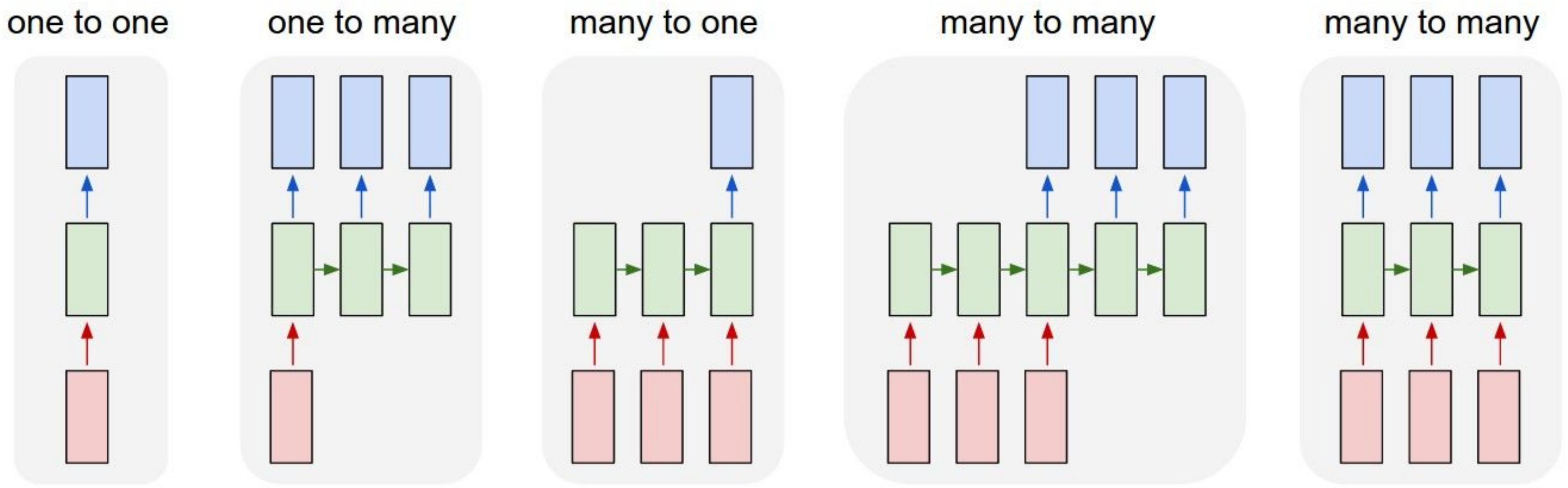
where $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ are coefficients that are shared temporally and g_1, g_2 activation functions.

$$p(\boldsymbol{y}_{1:T}|\boldsymbol{x}) = \sum_{\boldsymbol{h}_{1:T}} p(\boldsymbol{y}_{1:T}, \boldsymbol{h}_{1:T}|\boldsymbol{x}) = \sum_{\boldsymbol{h}_{1:T}} \prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{h}_t)p(\boldsymbol{h}_t|\boldsymbol{h}_{t-1}, \boldsymbol{y}_{t-1}, \boldsymbol{x})$$

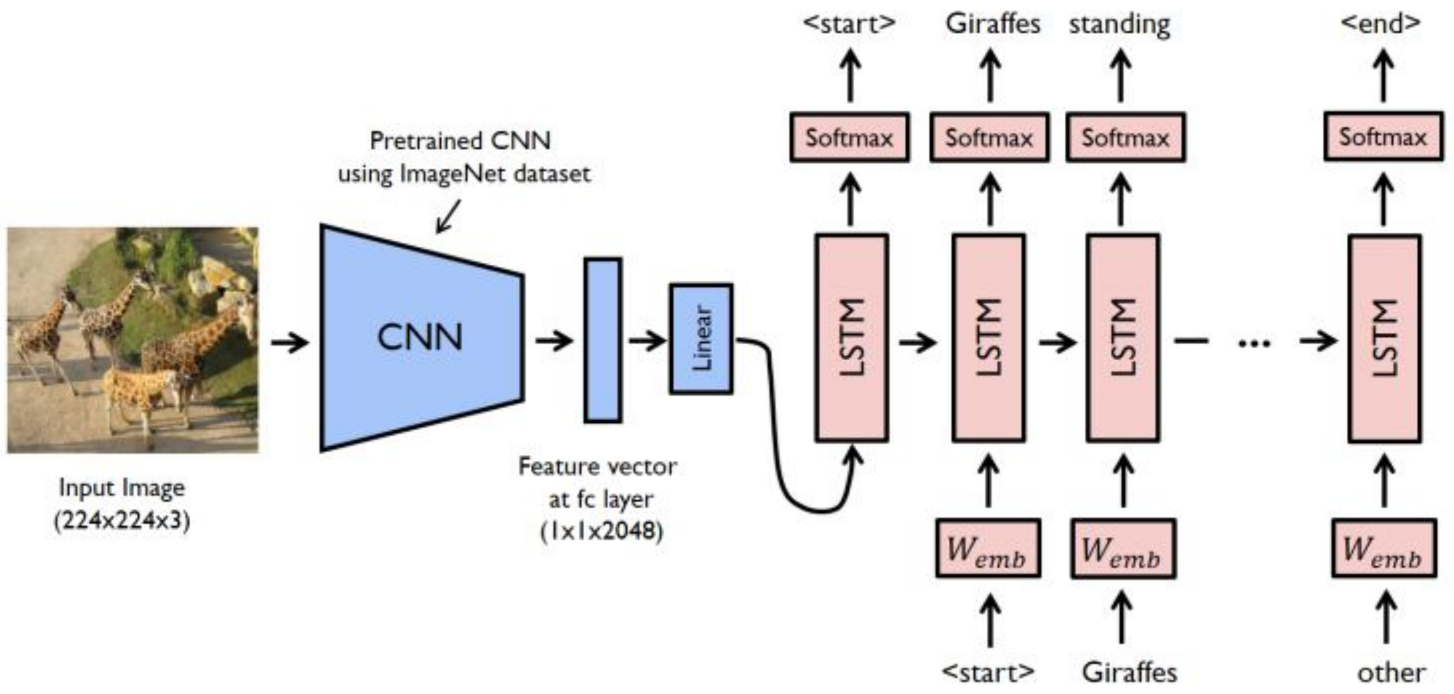
$$p(\boldsymbol{y}_t|\boldsymbol{h}_t) = \mathrm{Cat}(\boldsymbol{y}_t|\mathrm{softmax}(\mathbf{W}_{hy}\boldsymbol{h}_t + \boldsymbol{b}_y))$$



Seq2Seq Types



Example



Pretrained CNN using ImageNet dataset
CNN
Input Image (224x224x3)
Feature vector at fc layer (1x1x2048)
Linear
LSTM
Softmax
<start>
Giraffes
standing
<end>
W_{emb}
<start>
Giraffes
other

Backpropagation Through Time

We can compute the maximum likelihood estimate of the parameters for an RNN by solving $\boldsymbol{\theta}^* = \operatorname{argmax}_{\boldsymbol{\theta}} p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:T}, \boldsymbol{\theta})$, where we have assumed a single training sequence for notational simplicity. To compute the MLE, we have to compute gradients of the loss wrt the parameters.

More precisely, consider the following model:

$$\boldsymbol{h}_t = \mathbf{W}_{hx}\boldsymbol{x}_t + \mathbf{W}_{hh}\boldsymbol{h}_{t-1}$$
$$\boldsymbol{o}_t = \mathbf{W}_{ho}\boldsymbol{h}_t$$

$$L = \frac{1}{T}\sum_{t=1}^{T} \ell(y_t, \boldsymbol{o}_t)$$

We need to compute the derivatives $\frac{\partial L}{\partial \mathbf{W}_{hx}}$, $\frac{\partial L}{\partial \mathbf{W}_{hh}}$, and $\frac{\partial L}{\partial \mathbf{W}_{ho}}$. The latter term is easy, since it is local to each time step. However, the first two terms depend on the hidden state, and thus require working backwards in time.

Backpropagation Through Time

We simplify the notation by defining

$$\boldsymbol{h}_t = f(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{w}_h)$$
$$\boldsymbol{o}_t = g(\boldsymbol{h}_t, \boldsymbol{w}_o)$$

where $\boldsymbol{w}_h$ is the flattened version of $\mathbf{W}_{hh}$ and $\mathbf{W}_{hx}$ stacked together. We focus on computing $\frac{\partial L}{\partial \boldsymbol{w}_h}$. By the chain rule, we have

$$\frac{\partial L}{\partial \boldsymbol{w}_h} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial \ell(y_t, \boldsymbol{o}_t)}{\partial \boldsymbol{w}_h} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial \ell(y_t, \boldsymbol{o}_t)}{\partial \boldsymbol{o}_t} \frac{\partial g(\boldsymbol{h}_t, \boldsymbol{w}_o)}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{w}_h}$$

We can expand the last term as follows:

$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{w}_h} = \frac{\partial f(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{w}_h)}{\partial \boldsymbol{w}_h} + \frac{\partial f(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{w}_h)}{\partial \boldsymbol{h}_{t-1}} \frac{\partial \boldsymbol{h}_{t-1}}{\partial \boldsymbol{w}_h}$$

Backpropagation Through Time

$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{w}_h} = \frac{\partial f(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{w}_h)}{\partial \boldsymbol{w}_h} + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} \frac{\partial f(\boldsymbol{x}_j, \boldsymbol{h}_{j-1}, \boldsymbol{w}_h)}{\partial \boldsymbol{h}_{j-1}} \right) \frac{\partial f(\boldsymbol{x}_i, \boldsymbol{h}_{i-1}, \boldsymbol{w}_h)}{\partial \boldsymbol{w}_h}$$

Unfortunately, this takes *O(T)* time to compute per time step, for a total of *O(T^2)* overall. It is therefore standard to truncate the sum to the most recent *K* terms. It is possible to adaptively pick a suitable truncation parameter *K*.

Vanishing and Exploding Gradient

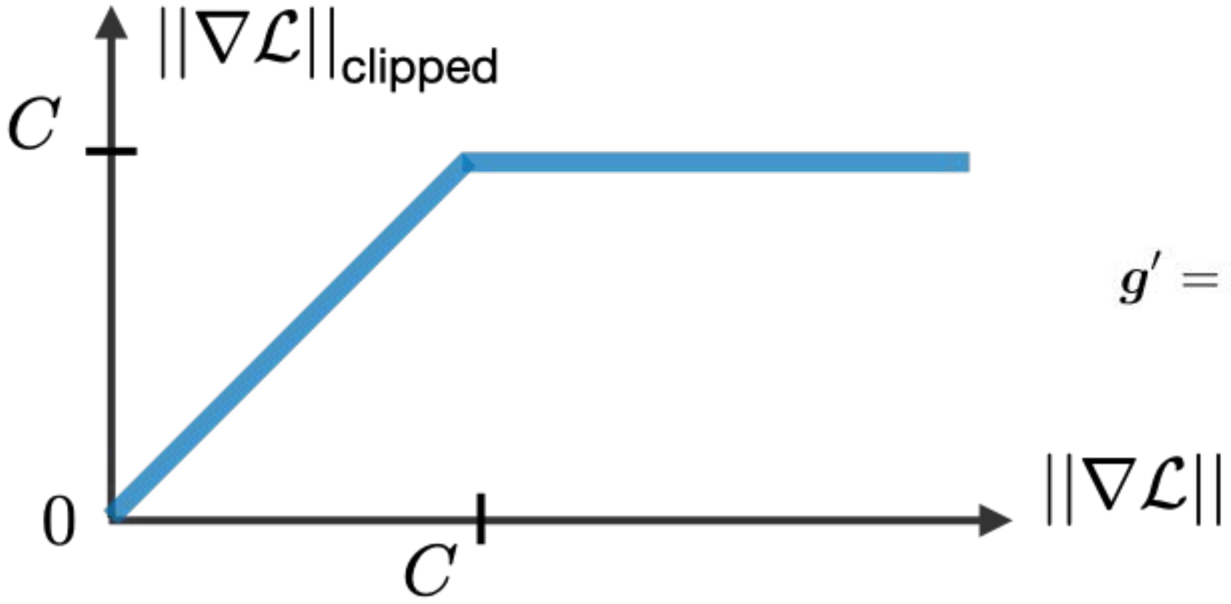
- Vanishing Gradients: we can see, that the tanh term will almost always be less than 1
- Removing tanh:
  - In $\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial h_t}(\prod_{t=2}^{T} W_{hh}^{T-1})\frac{\partial h_1}{\partial W}$ singular values of W_hh can be larger than 1. It will explode gradients
  - If SV of W_hh is lesser than 1 —> vanishing gradients

$$\frac{\partial L_t}{\partial W_{hh}} = \frac{\partial L_t}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}} \dots \frac{\partial h_1}{\partial W_{hh}}$$

$$= \frac{\partial L_t}{\partial h_t}(\prod_{t=2}^{T}\frac{\partial h_t}{\partial h_{t-1}})\frac{\partial h_1}{\partial W_{hh}}$$

$$= \frac{\partial L_t}{\partial h_t}(\prod_{t=2}^{T} tanh^{'}(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}^{T-1})\frac{\partial h_1}{\partial W_{hh}}$$

Gradient Clipping



$$\boldsymbol{g}' = \min(1, \frac{c}{||\boldsymbol{g}||})\boldsymbol{g}$$

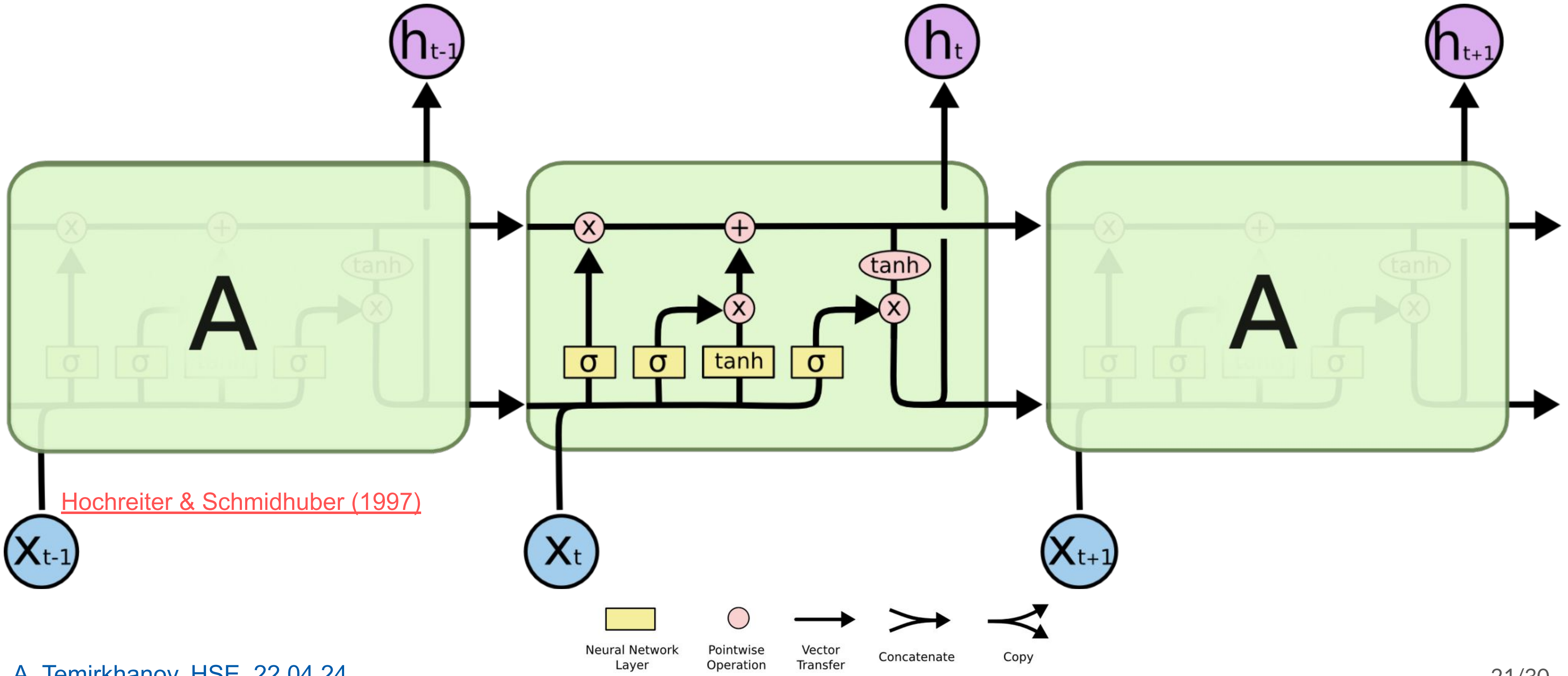
Gates

Introducing gates!

$$\boxed{\Gamma = \sigma(Wx^{<t>} + Ua^{<t-1>} + b)}$$

Type of gate	Role	Used in
Update gate Γ_u	How much past should matter now?	GRU, LSTM
Relevance gate Γ_r	Drop previous information?	GRU, LSTM
Forget gate Γ_f	Erase a cell or not?	LSTM
Output gate Γ_o	How much to reveal of a cell?	LSTM

LSTM

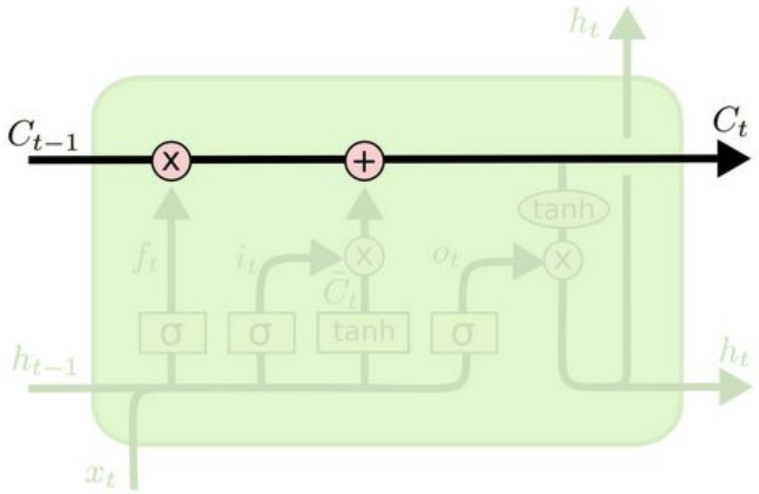


Hochreiter & Schmidhuber (1997)

LTSM Core

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.

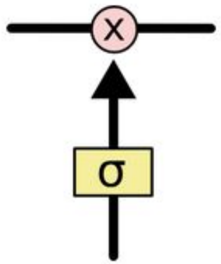
The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.



LSTM Core

The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

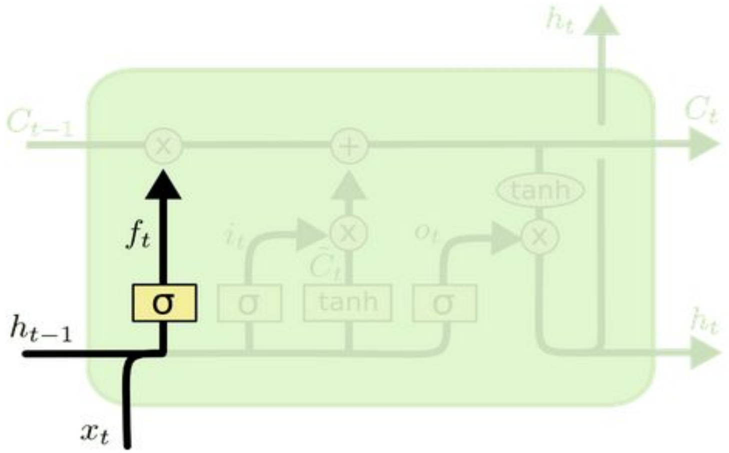
Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.



The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means “let nothing through,” while a value of one means “let everything through!”

LSTM Step-by-Step

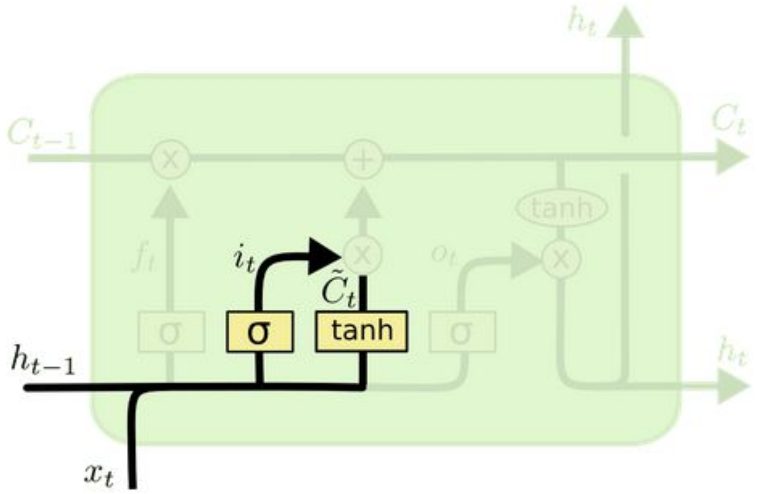
Let's go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.



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

LSTM Step-by-Step

In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.



input gate layer

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

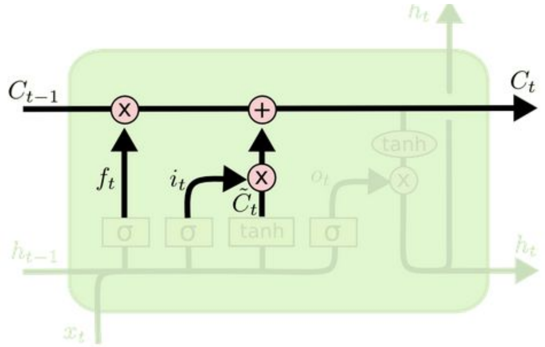
new candidates vector

LSTM Step-by-Step

It's now time to update the old cell state, C_{t-1}, into the new cell state C_t. The previous steps already decided what to do, we just need to actually do it.

We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.

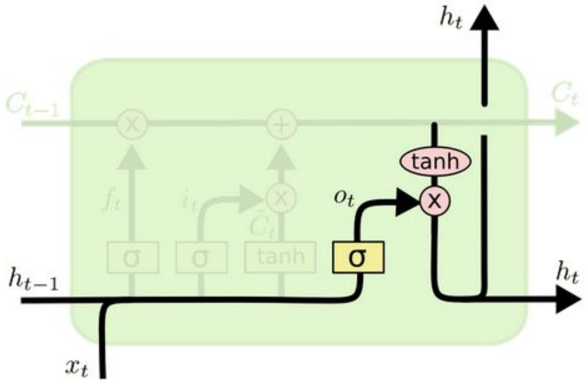
In the case of the language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in the previous steps.



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

LSTM Step-by-Step

Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through **tanh** (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

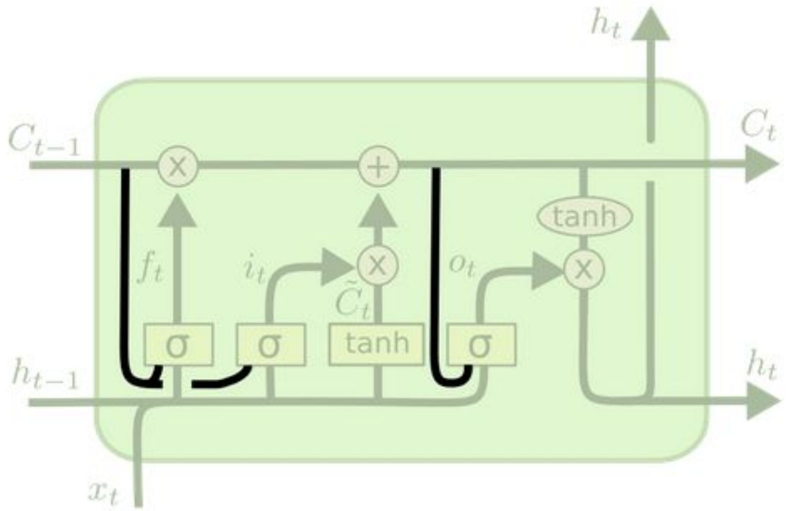


$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.

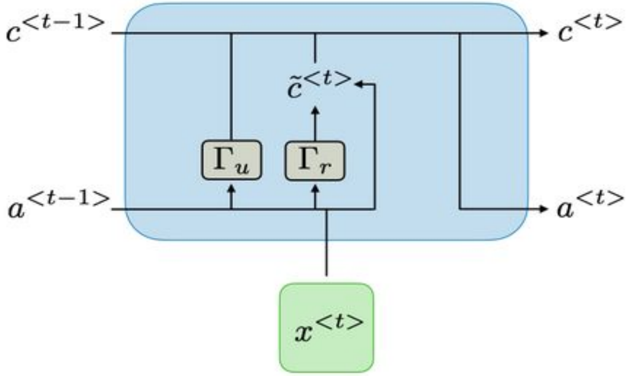
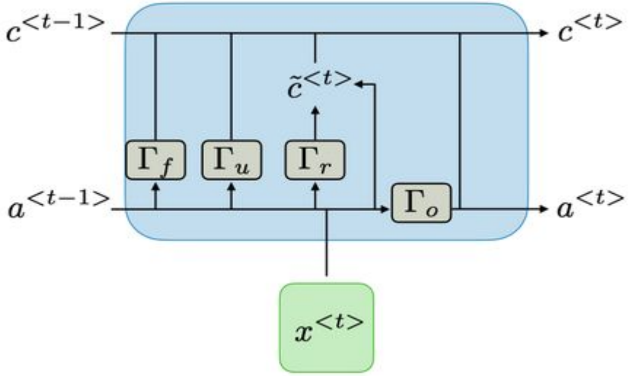
Schmidhuber Strikes Back!

One popular LSTM variant, introduced by Gers & Schmidhuber (2000), is adding “peephole connections.” This means that we let the gate layers look at the cell state.



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \ + \ b_o\right)$$

LSTM Summary

Characterization	**Gated Recurrent Unit** (GRU)	**Long Short-Term Memory** (LSTM)
$\tilde{c}^{<t>}$	$\tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$	$\tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$
$c^{<t>}$	$\Gamma_u \star \tilde{c}^{<t>} + (1 - \Gamma_u) \star c^{<t-1>}$	$\Gamma_u \star \tilde{c}^{<t>} + \Gamma_f \star c^{<t-1>}$
$a^{<t>}$	$c^{<t>}$	$\Gamma_o \star c^{<t>}$
Dependencies	 $c^{<t-1>}$ $c^{<t>}$ $\tilde{c}^{<t>}$ Γ_u Γ_r $a^{<t-1>}$ $a^{<t>}$ $x^{<t>}$	 $c^{<t-1>}$ $c^{<t>}$ $\tilde{c}^{<t>}$ Γ_f Γ_u Γ_r Γ_o $a^{<t-1>}$ $a^{<t>}$ $x^{<t>}$

Remark: the sign $\star$ denotes the element-wise multiplication between two vectors.

Perplexity

- In the best case scenario, the model always perfectly estimates the probability of the target token as 1. In this case the perplexity of the model is 1.
- In the worst case scenario, the model always predicts the probability of the target token as 0. In this situation, the perplexity is positive infinity.
- At the baseline, the model predicts a uniform distribution over all the available tokens of the vocabulary. In this case, the perplexity equals the number of unique tokens of the vocabulary. In fact, if we were to store the sequence without any compression, this would be the best we could do for encoding it. Hence, this provides a nontrivial upper bound that any useful model must beat.

$$\boxed{\mathrm{PP} = \prod_{t=1}^{T} \left(\frac{1}{\sum_{j=1}^{|V|} y_j^{(t)} \cdot \hat{y}_j^{(t)}} \right)^{\frac{1}{T}}}$$