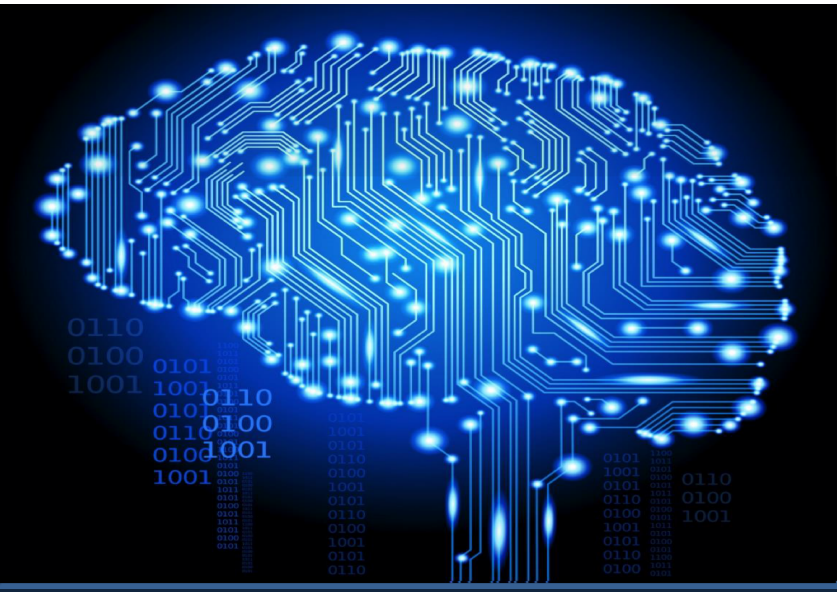
Prof. Saerom Park

Convergence Security Engineering
psr6275@sungshin.ac.kr
Sungshin Women's University



DL: Recurrent Neural Networks

Reference

- [Goodfellow et al. (2016)] I. Goodfellow, Y. Bengio, and A. Courville, Deep learning, The MIT Press, 2016.
- [Hinton. (2015)] G. Hinton, Online course on Neural Networks for Machine Learning.

  https://www.coursera.org/learn/neural-networks
- [Larochelle. (2014)] H. Larochelle, Online course on Neural Networks.

  http://info.usherbrooke.ca/hlarochelle/neural_networks/content.html
- [Gavves. (2017)] E. Gavves, UvA Deep Learning Course.

  http://uvadlc.github.io/
- Borrows slides (some modified) from Larochelle & Gavves

Source: http://uvadlc.github.io/

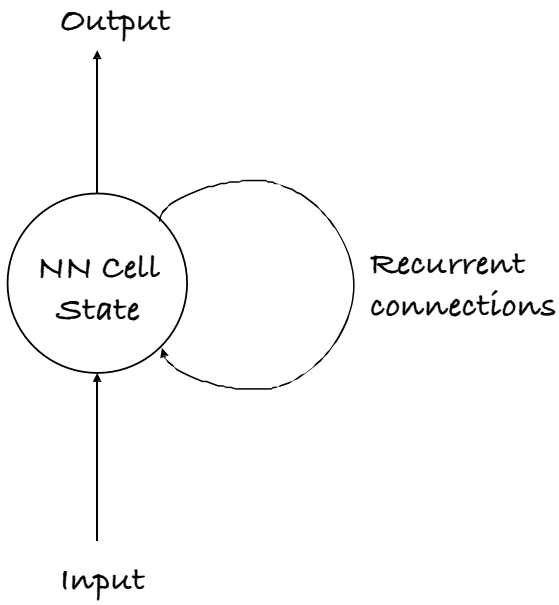
Lecture Overview

- Recurrent Neural Networks (RNN) for sequences
- Backpropagation Through Time (BPTT)
- Vanishing and Exploding Gradients and Remedies
- RNNs using Long Short-Term Memory (LSTM)
- Applications of Recurrent Neural Networks

Source: http://uvadlc.github.io/

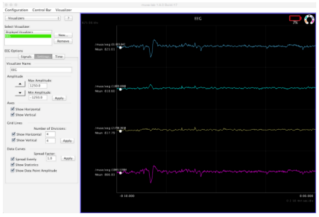
Recurrent Neural Networks

Prof. Saerom Park



Examples of Sequential Data and Time Series

- Videos
- Stock Exchange
- Biological Measurement
- Climate Measurement
- Market Analysis
- Speech /Music
- Use Behavior in Websites



Source: http://uvadlc.github.io/

Sequences

- Next data depend on previous data
  - Each data point on the previous ones and their distribution changes over time
  - How to model temporal dependencies, especially high-dim complex data?
- Sequences might be of arbitrary length
  - How to model dependencies at different temporal scales?
  - How to make a model that works for **both** 10 sec and 10 min sequences?
- Cycles in our computational graph
  - How to learn with cycles?
- Unclear how to assign credit to past time steps
  - Which of the infinite past time steps is responsible for this observed present?
- Chaotic behavior
  - How do we account for all possible effects of small changes in past sequence?

Source: http://uvadlc.github.io/

Solution!

- What makes sequences special?

Challenges	Inductive Bias
Non iid data	State models, Chain rule of Probabilities
Arbitrary length	Sharing weights
Credit assignment problem	Backpropagation through time
Chaotic behavior	LSTM, GRU

Source: http://uvadlc.github.io/

A sequence of probabilities

- Sequences are by definition non iid
  - p(He is a student) = p(He) p(is|He) p(a|He is) p(student|He is a)
  - p(She is also a student) = p(She) p(is|She) p(also|She is) p(a|She is also) p(student|She is also a)
- Compute likelihood by decomposing with chain rule of probabilities

$$\Pr(x) = \prod_i \Pr(x_i | x_{i-T}, \dots, x_{i-1})$$

- Model each term $\Pr(x_i | x_{i-T}, \dots, x_{i-1})$ separately

Source: http://uvadlc.github.io/

Memory

- To have the past influence present we must keep a summary of the past
  - A representation of the past
- At time step t project all information $1, \dots, t$ on a state (latent) space c_t with shared parameters θ
- Then, at time step $t+1$ re-use the parameters θ and the previous c_t

$$c_{t+1} = h(x_{t+1}, c_t; \theta)$$

- Recurrent parameters θ are the shared for all timesteps $t = 0, \dots$

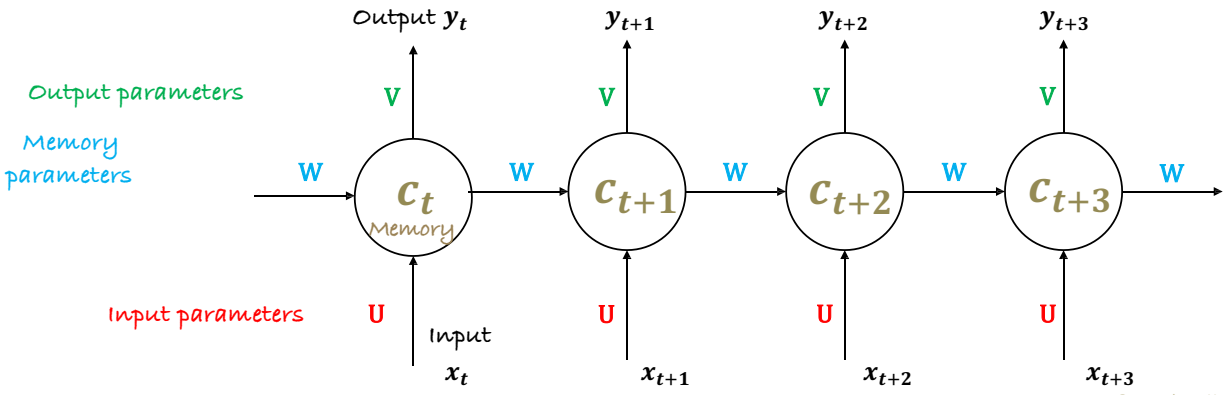
$$c_{t+1} = h(x_{t+1}, h(x_t, h(x_{t-1}, \dots, h(x_1, c_0; \theta); \theta); \theta); \theta)$$

- Sharing parameters through time enables sequences of arbitrary lengths

Source: http://uvadlc.github.io/

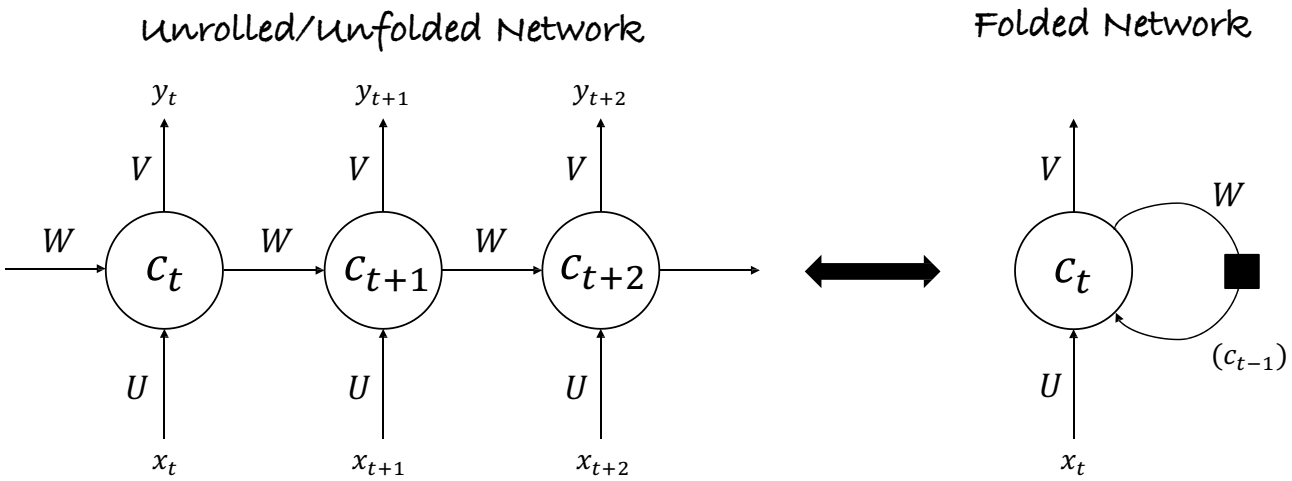
Memory as a Graph

- Simplest model
  - Input with parameters U
  - Memory embedding with parameters W
  - Output with parameters V



Source: http://uvadlc.github.io/

Folding the memory

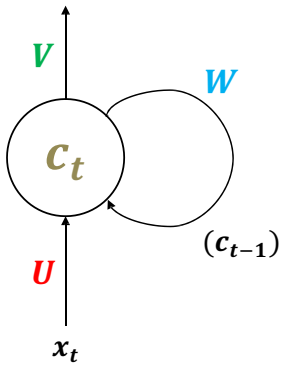


Source: http://uvadlc.github.io/

Recurrent Neural Network (RNN)

- Only two equations

$$c_t = \tanh(b + U\, x_t + W\, c_{t-1})$$
$$y_t = \mathrm{softmax}(a + V\, c_t)$$



Source: http://uvadlc.github.io/

Recurrent Neural Networks

- Putting things together!
- A memory of 3 units [Hyperparameter that we choose like layer size]
  - $c_t: [3\times1], W: [3\times3]$
- An input projection of 3 dimensions
  - $U: [3\times5]$
- An output projections of 10 dimensions
  - $V: [10\times3]$

$$b = 0,\ a = 0$$

$$c_t = \tanh(U\, x_t + W\, c_{t-1})$$

$$y_t = \text{softmax}(V\, c_t)$$

$$\mathcal{L} = \sum_t \mathcal{L}_t(y_t, l_t)$$

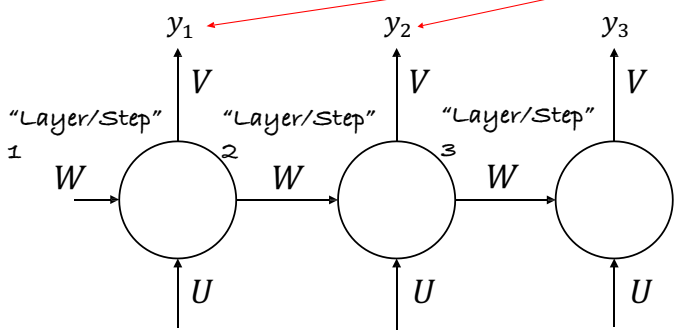
$$U \cdot x_{t=4} = \begin{bmatrix} 0.1 & -0.3 & 1.2 & 0.6 & -0.8 \\ -0.2 & 0.4 & 0.5 & 0.9 & -0.1 \\ -0.1 & 0.2 & -0.7 & -0.8 & 0.3 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.6 \\ 0.9 \\ -0.8 \end{bmatrix} = U^{(4)}$$

Source: http://uvadlc.github.io/

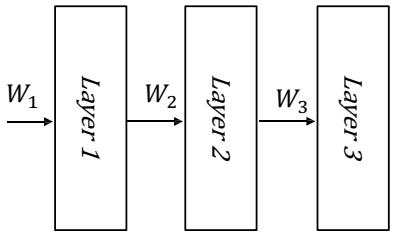
Rolled Network vs. Multi-layer Network?

- What is really different?
  - Steps instead of layers
  - Step parameters shared whereas in a Multi-Layer Network they are different

- Sometimes intermediate outputs are not even needed
- Removing them, we almost end up to a standard Neural Network



3-gram Unrolled Recurrent Network



3-layer Neural Network

Source: http://uvadlc.github.io/

Training Recurrent Networks

- Cross-entropy loss

$$P = \prod_{t,k} y_{tk}^{l_{tk}} \Longrightarrow \mathcal{L} = -\log P = \sum_t \mathcal{L}_t = -\frac{1}{T}\sum_t l_t \log y_t$$

- Backpropagation Through Time (BPTT)
  - Basically, chain rule again for $\frac{d\mathcal{L}}{dV}, \frac{d\mathcal{L}}{dU}, \frac{d\mathcal{L}}{dW}$ → the same algorithm
  - Shared computations complicate the chain rule!
  - Only difference: Gradients survive over time steps
    - Gradients flow not from a single path of previous layer like in MLP
    - The recurrence in the chain rule hides multiple dependencies

Source: http://uvadlc.github.io/

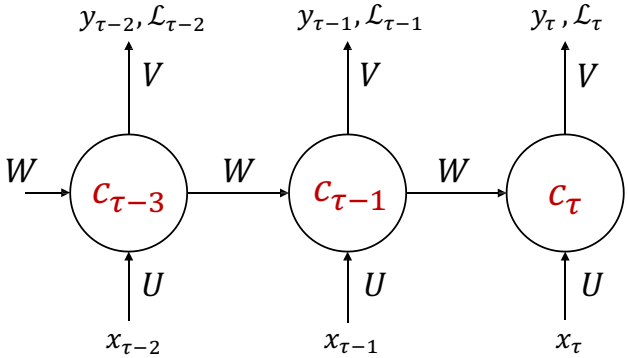
Backpropagation Through Time: An Example

- $\frac{\partial \mathcal{L}}{\partial V}, \frac{\partial \mathcal{L}}{\partial W}, \frac{\partial \mathcal{L}}{\partial U}$
- Let's focus on the final step τ

$$\frac{\partial \mathcal{L}_\tau}{\partial V}, \frac{\partial \mathcal{L}_\tau}{\partial W}, \frac{\partial \mathcal{L}_\tau}{\partial U}$$

Step by step explanation at:
http://www.wildml.com/2015/10/recurrent-neural-networks- tutorial-part-3-backpropagation-through-time-and-vanishing- gradients/

$$c_t = \tanh(b + Ux_t + Wc_{t-1})$$
$$y_t = \text{softmax}(a + Vc_t)$$
$$\mathcal{L} = -\sum_t l_t \log y_t = \sum_t \mathcal{L}_t$$



Source: http://uvadlc.github.io/

Backpropagation Through Time

$$\frac{\partial \text{softmax}(\boldsymbol{z})_j}{\partial z_i} = \text{softmax}(\boldsymbol{z})_i(\delta_{ij} - \text{softmax}(\boldsymbol{z})_j)$$

$$\frac{\partial}{\partial \alpha^{(L)}(x)_c} - \ln f(x)_y = \frac{-1}{f(x)_y}\frac{\partial f(x)_y}{\partial \alpha^{(L)}(x)_c}$$
$$= \frac{-1}{f(x)_y} f(x)_y(\delta_{yc} - f(x)_c) = -(1_{(y=c)} - f(x)_c)$$

$$\frac{\partial \mathcal{L}_\tau}{\partial c_\tau} = \left(\frac{\partial \alpha_\tau}{\partial c_\tau}\right)^T \frac{\partial \mathcal{L}_\tau}{\partial \alpha_\tau} = \left(\frac{\partial (V c_\tau)}{\partial c_\tau}\right)^T \cdot [-(l_\tau - y_\tau)] = V^T(y_\tau - l_\tau)$$

$$\frac{\partial \mathcal{L}_\tau}{\partial V} = \frac{\partial \mathcal{L}_\tau}{\partial \alpha_\tau}\frac{\partial \alpha_\tau}{\partial V} = (y_\tau - l_\tau) \cdot \frac{\partial (V c_\tau)}{\partial V} = (y_\tau - l_\tau) \cdot c_\tau^T$$

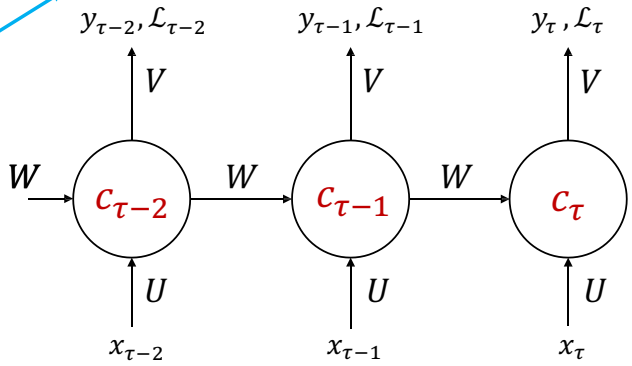
$$\frac{\partial \mathcal{L}}{\partial \alpha_t} = \frac{\partial \mathcal{L}_t}{\partial \alpha_t}\frac{\partial \mathcal{L}}{\partial \mathcal{L}_t} = y_t - l_t$$

$$\frac{\partial L}{\partial a_l} = \left(\frac{\partial a_{l+1}}{\partial a_l}\right)^T \cdot \frac{\partial L}{\partial a_{l+1}}$$
$$\frac{\partial L}{\partial \theta_l} = \frac{\partial a_l}{\partial \theta_l} \cdot \left(\frac{\partial L}{\partial a_l}\right)^T$$
$$\frac{\partial L}{\partial \theta_l^T} = \frac{\partial a_l}{\partial \theta_l^T} \cdot \left(\frac{\partial L}{\partial a_l}\right)$$

$$\mathcal{L} = \sum_t \mathcal{L}_t \rightarrow \frac{\partial \mathcal{L}}{\partial \mathcal{L}_t} = 1$$

$$\frac{\partial}{\partial \boldsymbol{x}}(\boldsymbol{c}^T\boldsymbol{x}) = \boldsymbol{c}$$
$$\frac{\partial}{\partial \boldsymbol{x}}(\mathbf{A}\boldsymbol{x}) = \mathbf{A}$$
$$\frac{\partial}{\partial \mathbf{A}}(\mathbf{A}^T\boldsymbol{x}) = \boldsymbol{x}$$
$$\frac{\partial}{\partial \mathbf{A}}(\mathbf{A}\boldsymbol{x}) = \boldsymbol{x}^T$$

$$c_t = \tanh(b + Ux_t + Wc_{t-1})$$
$$\alpha_t = a + Vc_t$$
$$y_t = \text{softmax}(\alpha_t)$$
$$\mathcal{L} = -\sum_t l_t \log y_t = \sum_t \mathcal{L}_t$$



Backpropagation Through Time

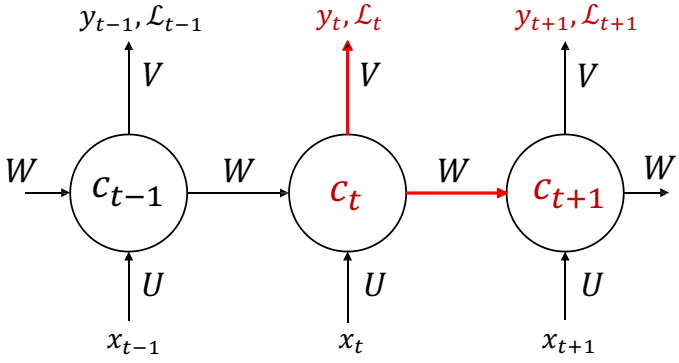
- c_t has as descendants both c_{t+1} and y_t
  - $\frac{\partial f(\varphi(x),\psi(x))}{\partial x} = \frac{\partial f}{\partial \varphi}\frac{\partial \varphi}{\partial x} + \frac{\partial f}{\partial \psi}\frac{\partial \psi}{\partial x}$

$$\frac{\partial \mathcal{L}}{\partial c_t} = \left(\frac{\partial c_{t+1}}{\partial c_t}\right)^T \frac{\partial \mathcal{L}}{\partial c_{t+1}} + \left(\frac{\partial \alpha_t}{\partial c_t}\right)^T \frac{\partial \mathcal{L}}{\partial \alpha_t} = \left(\frac{\partial c_{t+1}}{\partial c_t}\right)^T \frac{\partial \mathcal{L}}{\partial c_{t+1}} + \left(\frac{\partial \alpha_t}{\partial c_t}\right)^T \frac{\partial \mathcal{L}_t}{\partial \alpha_t} = W^T \text{diag}(1 - c_{t+1}^2)\frac{\partial \mathcal{L}}{\partial c_{t+1}} + V^T(y_t - l_t)$$

In $\mathcal{L}$, only $\mathcal{L}_t$ and $\mathcal{L}_{t+1}$ depends on c_t

$$a = \tanh(x) \rightarrow a' = 1 - a^2$$

$$c_t = \tanh(b + Ux_t + Wc_{t-1})$$
$$\alpha_t = a + Vc_t$$
$$y_t = \text{softmax}(\alpha_t)$$
$$\mathcal{L} = -\sum_t l_t \log y_t = \sum_t \mathcal{L}_t$$

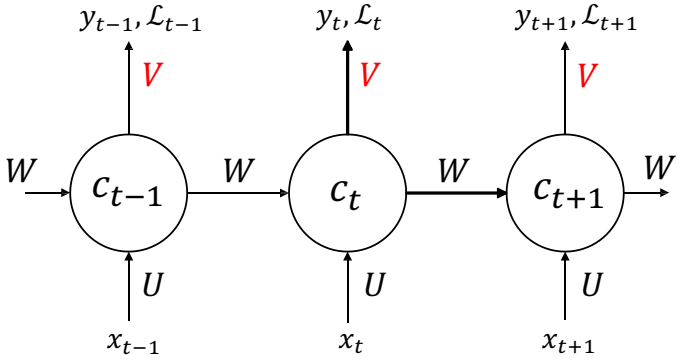


Backpropagation Through Time

$$\frac{\partial \mathcal{L}}{\partial V} = \sum_{t=1}^{\tau} \frac{\partial \mathcal{L}}{\partial \mathcal{L}_t} \frac{\partial \mathcal{L}_t}{\partial \alpha_t} \frac{\partial \alpha_t}{\partial V} = (l_t - y_t) \cdot \frac{\partial (V c_t)}{\partial V} = \sum_{t=1}^{\tau} (y_t - l_t) \cdot c_t^T$$

$$\frac{\partial \mathcal{L}}{\partial a} = \sum_{t=1}^{\tau} \frac{\partial \mathcal{L}}{\partial \mathcal{L}_t} \left(\frac{\partial \alpha_t}{\partial a}\right)^T \frac{\partial \mathcal{L}_t}{\partial \alpha_t} = \sum_{t=1}^{\tau} \boldsymbol{I} \cdot (y_t - l_t) = \sum_{t=1}^{\tau} (y_t - l_t)$$

$$c_t = \tanh(b + U x_t + W c_{t-1})$$
$$\alpha_t = a + V c_t$$
$$y_t = \mathrm{softmax}(\alpha_t)$$
$$\mathcal{L} = -\sum_t l_t \log y_t = \sum_t \mathcal{L}_t$$



Backpropagation Through Time

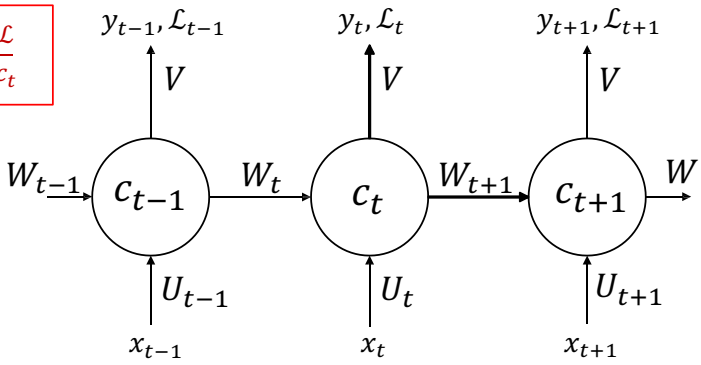
$$\frac{\partial \mathcal{L}}{\partial U} = \sum_{t=1}^{\tau} \frac{\partial \mathcal{L}}{\partial U_t} = \sum_{t=1}^{\tau} \frac{\partial \mathcal{L}}{\partial c_t} \left(\frac{\partial c_t}{\partial U_t}\right)^T = \sum_{t=1}^{\tau} \frac{\partial \mathcal{L}}{\partial c_t} \cdot diag(1 - c_t^2) \cdot x_t^T$$

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_{t=1}^{\tau} \frac{\partial \mathcal{L}}{\partial W_t} = \sum_{t=1}^{\tau} \frac{\partial \mathcal{L}}{\partial c_t} \left(\frac{\partial c_t}{\partial W_t}\right)^T = \sum_{t=1}^{\tau} \frac{\partial \mathcal{L}}{\partial c_t} \cdot \mathrm{diag}(1 - c_t^2) \cdot c_{t-1}^T$$

$$\frac{\partial \mathcal{L}}{\partial b} = \sum_{t=1}^{\tau} \frac{\partial \mathcal{L}}{\partial b_t} = \sum_{t=1}^{\tau} \left(\frac{\partial c_t}{\partial b_t}\right)^T \frac{\partial \mathcal{L}}{\partial c_t} = \sum_{t=1}^{\tau} \mathrm{diag}(1 - c_t^2) \cdot \frac{\partial \mathcal{L}}{\partial c_t}$$

The $\frac{\partial \mathcal{L}}{\partial W}$ operator used in calculus takes into account the contribution of W to the value of $\mathcal{L}$ due to **all** edges in the computational graph. To resolve this ambiguity, we introduce dummy variables W_t that are defined to be copies of W but with each W_t used only at time step t. We may then use $\frac{\partial \mathcal{L}}{\partial W_t}$ to denote the contribution of the weights at time step t to the gradient.

$$c_t = \tanh(b_t + U_t x_t + W_t c_{t-1})$$
$$\alpha_t = a + V c_t$$
$$y_t = \mathrm{softmax}(\alpha_t)$$
$$\mathcal{L} = -\sum_t l_t \log y_t = \sum_t \mathcal{L}_t$$



Challenge of long-term dependencies

- Vanishing gradients
  - After a few time steps the gradients become almost 0
- Exploding gradients
  - After a few time steps the gradients become huge
- Can't capture long-term dependencies

$$\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{W}} = \sum_{i=0}^{t} \frac{\partial \mathcal{L}_t}{\partial y_t} \frac{\partial y_t}{\partial \boldsymbol{s}_t} \frac{\partial \boldsymbol{s}_t}{\partial \boldsymbol{s}_i} \frac{\partial \boldsymbol{s}_i}{\partial \boldsymbol{W}} = \sum_{i=0}^{t} \frac{\partial \mathcal{L}_t}{\partial y_t} \frac{\partial y_t}{\partial \boldsymbol{s}_t} \left(\prod_{j=i+1}^{t} \frac{\partial \boldsymbol{s}_j}{\partial \boldsymbol{s}_{j-1}} \right) \frac{\partial \boldsymbol{s}_i}{\partial \boldsymbol{W}}$$

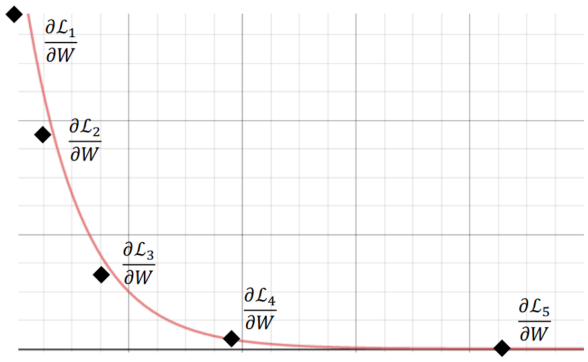
  - To make it simpler, assume that $c_t = W \cdot c_{t-1}$ and W admits an eigendecomposition of the form $W = V \Lambda V^{-1}$, then

$$c_t = W \cdot c_{t-1} = W^t \cdot c_0 = V \Lambda^{\mathrm{t}} V^{-1} \cdot c_0$$

    - The eigenvalues are raised to the power of t causing eigenvalues with magnitude less than one to decay to zero and eigenvalues with magnitude greater than one to explode. Any component of c_0 that is not aligned with the largest eigenvector will eventually be discarded.

Vanishing gradients

- Exponentially smaller contribution of longer-term terms
  - Model emphasizes on shorter-term terms as they have larger gradients
- Can be undesirable if the 'distant past' is a key factor



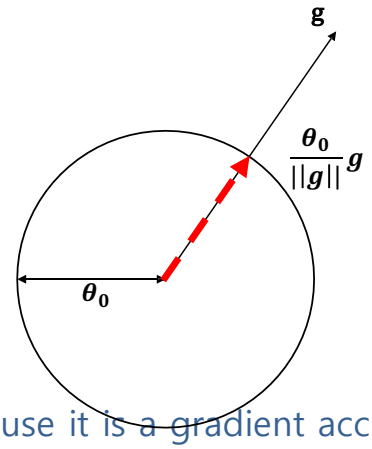
Source: http://uvadlc.github.io/

Gradient clipping for exploding gradients

- Scale the gradients to a threshold

Pseudocode

1. $g \leftarrow \frac{\partial \mathcal{L}}{\partial W}$
2. if $||g|| > \theta_0$:
   $g \leftarrow \frac{\theta_0}{||g||} g$
   else:
   print("Do nothing")

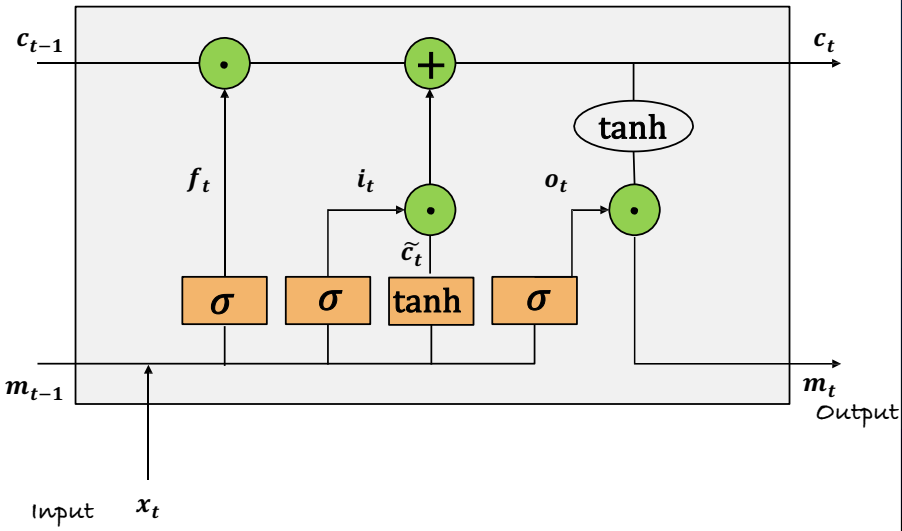


- Simple, but works!
- We cannot rescale vanishing gradients because it is a gradient accuracy issue
  - A vanishing gradient does not count as a gradient in the first place

Source: http://uvadlc.github.io/

Advanced RNNs

Prof. Saerom Park



How to fix the vanishing gradients?

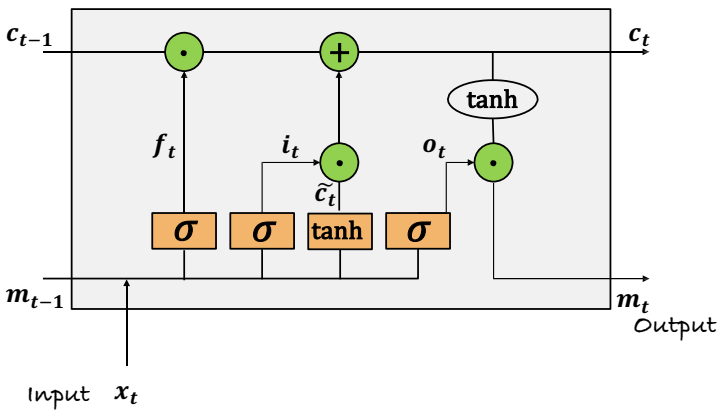
- Error signal over time must have not too large, not too small norm
- Solution: have an activation function with derivative equal to 1
  - $\frac{\partial c_t}{\partial c_{t-1}} = 1$ (Identify function)
- Remove immediate nonlinear relation between c_t and c_{t-1}
  - Replace tanh between c_t and c_{t-1} with identity
  - By doing so, gradients do not become too small not too large
- Also, avoid continuous overwriting of state
  - Modulate the importance of new input by a gate
  - Modulate the importance of new output by a gate
  - Modulate the importance of past memories by a gate

$$c_t = \tanh(b + U\, x_t + W\, c_{t-1})$$
$$y_t = \text{softmax}(a + V\, c_t)$$

Source: http://uvadlc.github.io/

Long Short-Term Memory (LSTM: Beefed up RNN)

$$i_t = \sigma(b^{(i)} + U^{(i)} x_t + W^{(i)} m_{t-1})$$
$$f_t = \sigma(b^{(f)} + U^{(f)} x_t + W^{(f)} m_{t-1})$$
$$o_t = \sigma(b^{(o)} + U^{(o)} x_t + W^{(o)} m_{t-1})$$
$$\tilde{c}_t = \tanh(b^{(g)} + U^{(g)} x_t + W^{(g)} m_{t-1})$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$
$$m_t = \tanh(c_t) \odot o_t$$

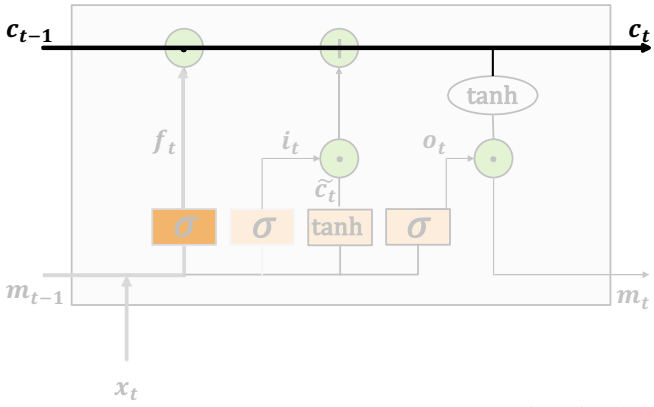


Long Short-Term Memory (LSTM: Beefed up RNN)

- The cell state carries the essential information over time

Cell state line

$$i_t = \sigma(b^{(i)} + U^{(i)}x_t + W^{(i)}m_{t-1})$$
$$f_t = \sigma(b^{(f)} + U^{(f)}x_t + W^{(f)}m_{t-1})$$
$$o_t = \sigma(b^{(o)} + U^{(o)}x_t + W^{(o)}m_{t-1})$$
$$\tilde{c}_t = \tanh(b^{(g)} + U^{(g)}x_t + W^{(g)}m_{t-1})$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$
$$m_t = \tanh(c_t) \odot o_t$$

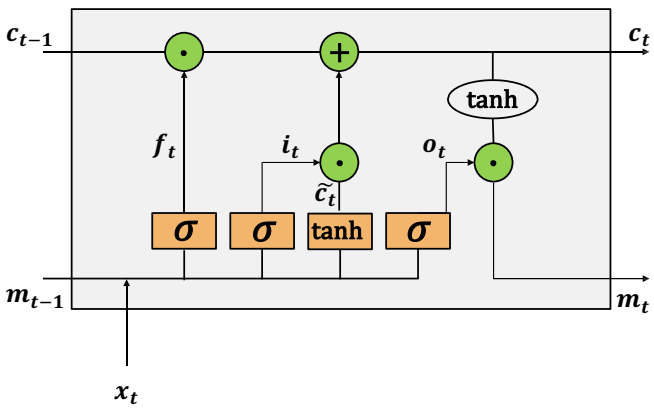


Source: http://uvadlc.github.io/

LSTM nonlinearities

- $\sigma \in (0,1)$ control gate – something like a switch
- $\tanh \in (-1,1)$: recurrent nonlinearity

$$i_t = \sigma(b^{(i)} + U^{(i)}x_t + W^{(i)}m_{t-1})$$
$$f_t = \sigma(b^{(f)} + U^{(f)}x_t + W^{(f)}m_{t-1})$$
$$o_t = \sigma(b^{(o)} + U^{(o)}x_t + W^{(o)}m_{t-1})$$
$$\tilde{c}_t = \tanh(b^{(g)} + U^{(g)}x_t + W^{(g)}m_{t-1})$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$
$$m_t = \tanh(c_t) \odot o_t$$



Source: http://uvadlc.github.io/

LSTM Step (1): Forget gate

- E.g. Model the sentence "Yesterday she slapped me. Today she loves me."
- Decide what to forget and what to remember for the new memory
  - Sigmoid 1 → Remember everything
  - Sigmoid 0 → Forget everything

$$i_t = \sigma(b^{(i)} + U^{(i)}x_t + W^{(i)}m_{t-1})$$

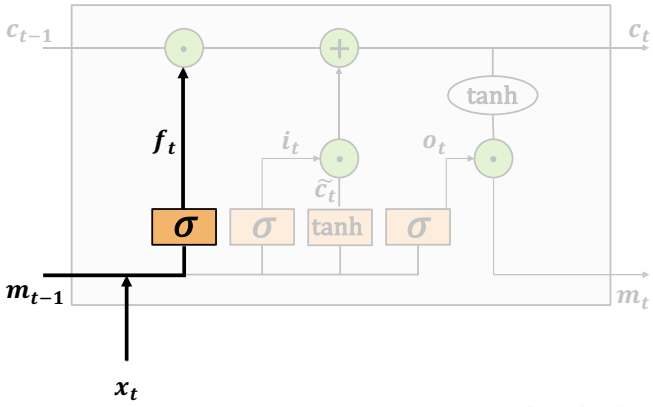
$$f_t = \sigma(b^{(f)} + U^{(f)}x_t + W^{(f)}m_{t-1})$$

$$o_t = \sigma(b^{(o)} + U^{(o)}x_t + W^{(o)}m_{t-1})$$

$$\tilde{c}_t = \tanh\left(b^{(g)} + U^{(g)}x_t + W^{(g)}m_{t-1}\right)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$$m_t = \tanh(c_t) \odot o_t$$



Source: http://uvadlc.github.io/

LSTM Step (2): Input gate

- Decide what new information should you add to the new memory
  - Modulate the input i_t
  - Generate candidate memories $\tilde{c}_t$

$$i_t = \sigma(b^{(i)} + U^{(i)}x_t + W^{(i)}m_{t-1})$$

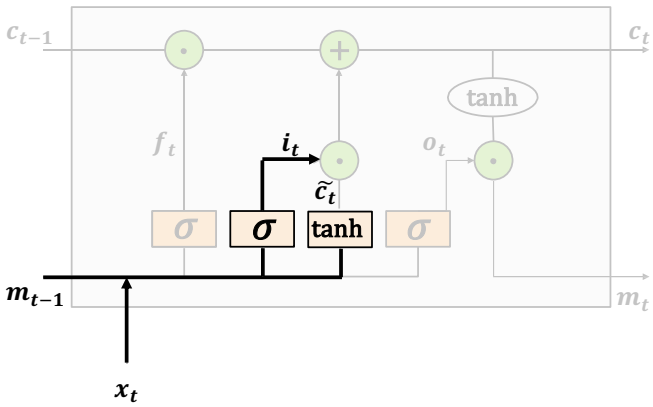
$$f_t = \sigma(b^{(f)} + U^{(f)}x_t + W^{(f)}m_{t-1})$$

$$o_t = \sigma(b^{(o)} + U^{(o)}x_t + W^{(o)}m_{t-1})$$

$$\tilde{c}_t = \tanh\left(b^{(g)} + U^{(g)}x_t + W^{(g)}m_{t-1}\right)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$$m_t = \tanh(c_t) \odot o_t$$

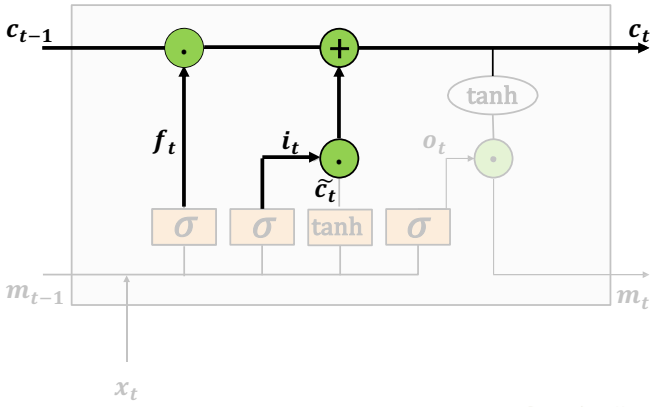


Source: http://uvadlc.github.io/

LSTM Step (3): Cell gate

- Compute and update the current cell state c_t
  - Depends on the previous cell state
  - What we decide to forget
  - What inputs we allow
  - The candidate memories

$$i_t = \sigma(b^{(i)} + U^{(i)}x_t + W^{(i)}m_{t-1})$$
$$f_t = \sigma(b^{(f)} + U^{(f)}x_t + W^{(f)}m_{t-1})$$
$$o_t = \sigma(b^{(o)} + U^{(o)}x_t + W^{(o)}m_{t-1})$$
$$\tilde{c}_t = \tanh(b^{(g)} + U^{(g)}x_t + W^{(g)}m_{t-1})$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$
$$m_t = \tanh(c_t) \odot o_t$$

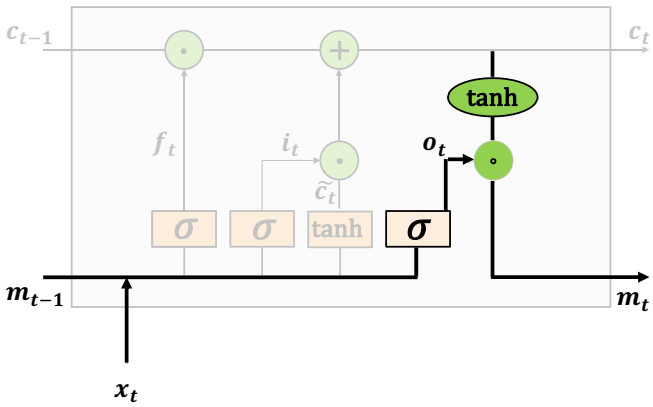


Source: http://uvadlc.github.io/

LSTM Step (4): Output gate

- Modulate the output
  - Does the cell state contain something relevant? → Sigmoid 1
- Generate the new memory

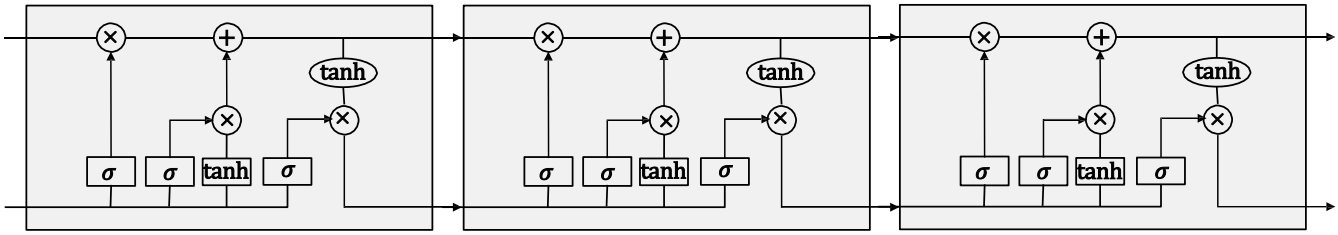
$$i_t = \sigma(b^{(i)} + U^{(i)}x_t + W^{(i)}m_{t-1})$$
$$f_t = \sigma(b^{(f)} + U^{(f)}x_t + W^{(f)}m_{t-1})$$
$$o_t = \sigma(b^{(o)} + U^{(o)}x_t + W^{(o)}m_{t-1})$$
$$\tilde{c}_t = \tanh(b^{(g)} + U^{(g)}x_t + W^{(g)}m_{t-1})$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$
$$m_t = \tanh(c_t) \odot o_t$$



Source: http://uvadlc.github.io/

LSTM Unrolled Network

- Macroscopically very similar to standard RNNs
- The engine is a bit different (more complicated)
  - Because of their gates LSTMs capture long and short term dependencies

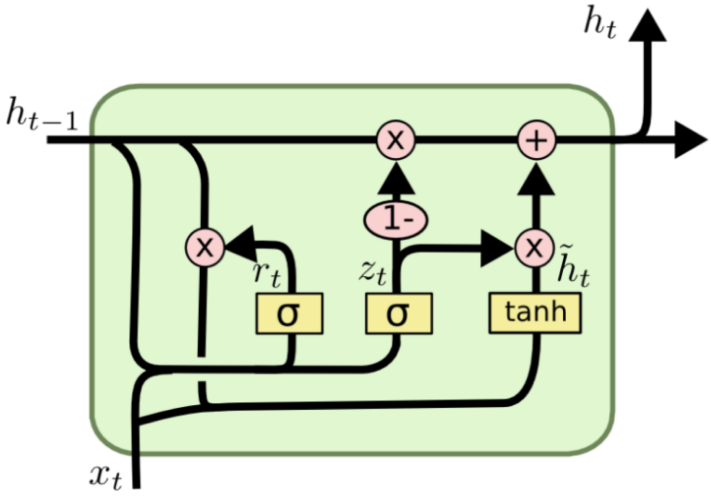


Source: http://uvadlc.github.io/

Gated Recurrent Units (GRU)

- Gated Recurrent Units (GRU)

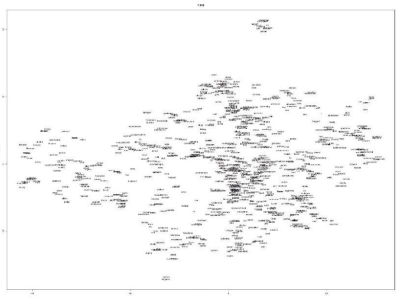
$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$
$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

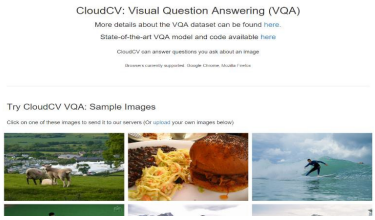


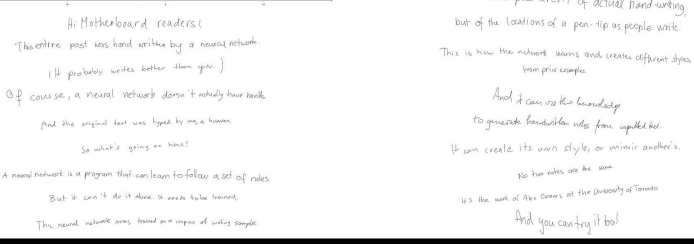
Applications of Recurrent Networks

Prof. Saerom Park

NeuralTalk and Walk, recognition, text description of the image while walking

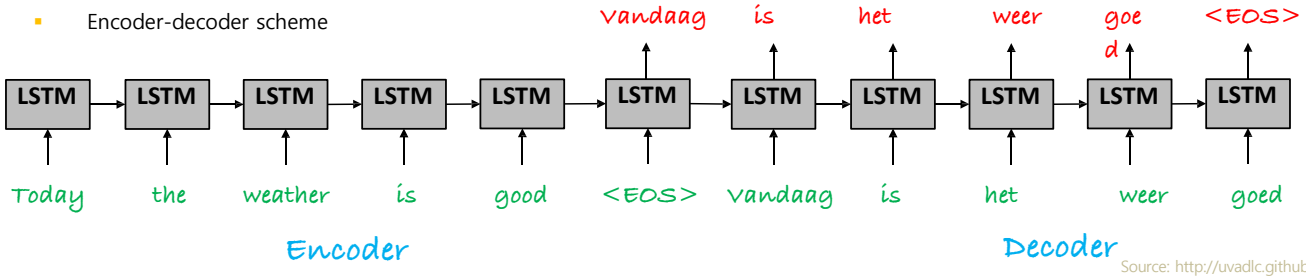






Machine Translation

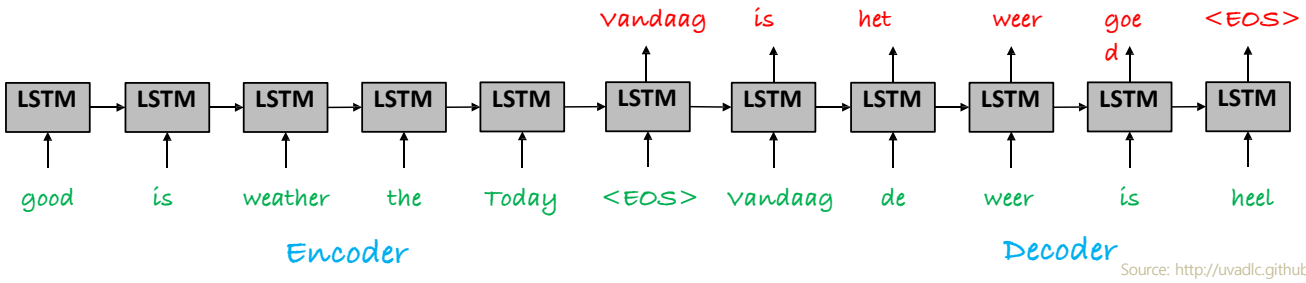
- The phrase in the source language is one sequence
  - "Today the weather is good"
- The phrase in the target language is also a sequence
  - "Vandaag is het weer goed"
- Problems
  - no perfect word alignment, sentence length might differ
- Solution
  - Encoder-decoder scheme



Source: http://uvadlc.github.io/

Better Machine Translation

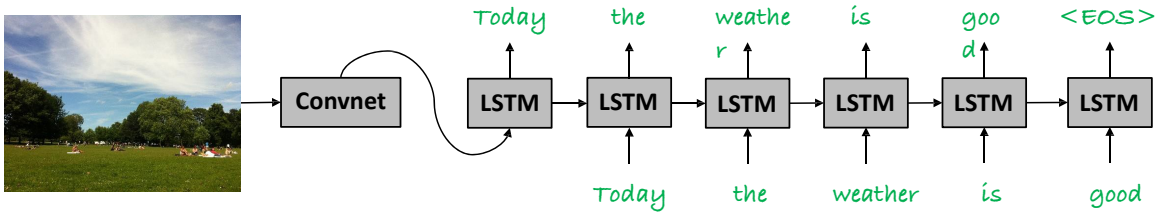
- It might even pay off reversing the source sentence
  - The first target words will be closer to their respective source words
- The encoder and decoder parts can be modelled with different LSTMs
- Deep LSTM



Source: http://uvadlc.github.io/

Image captioning

- An image is a thousand words, literally!
- Pretty much the same as machine transation
- Replace the encoder part with the output of a Convnet
  - E.g. use Alexnet or a VGG16 network
- Keep the decoder part to operate as a translator



Source: http://uvadlc.github.io/

Image captioning demo

- https://milhidaka.github.io/chainer-image-caption/

Generating image caption demo

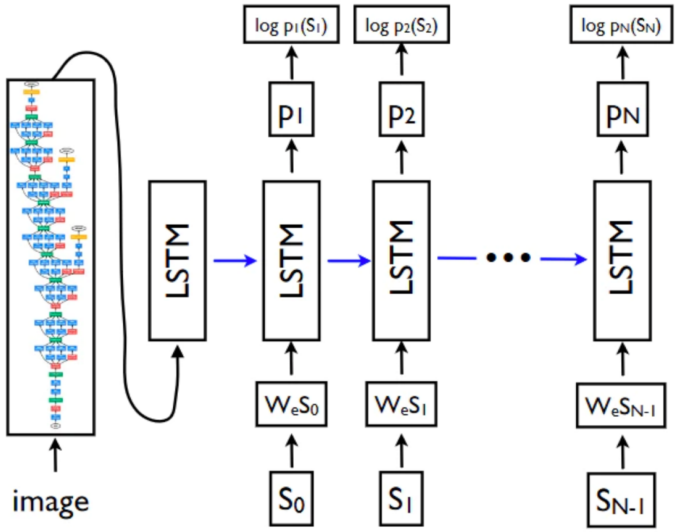
Input image (can drag-drop image file):

파일 선택 선택된 파일 없음

Generate caption

Load models > Analyze image > Generate text

```
Generated caption will be shown here.
```



Source: http://uvadlc.github.io/

Reading material & references

- http://www.deeplearningbook.org/
  - Part II: Chapter 13,14

Source: http://uvadlc.github.io/