

Feb 14th, 2018



Recurrent Neural Networks

Ansgar Rössig, Arne Schmidt

Agenda

Introduction

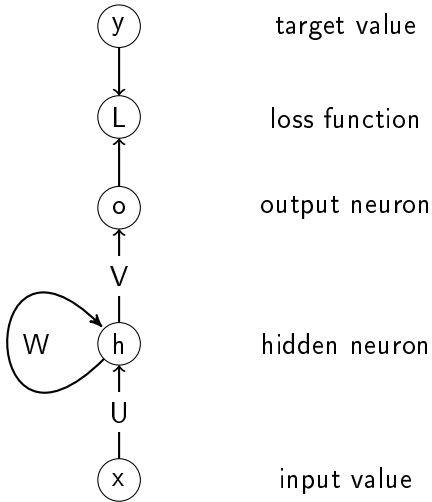
Computational Power

Variants of RNNs

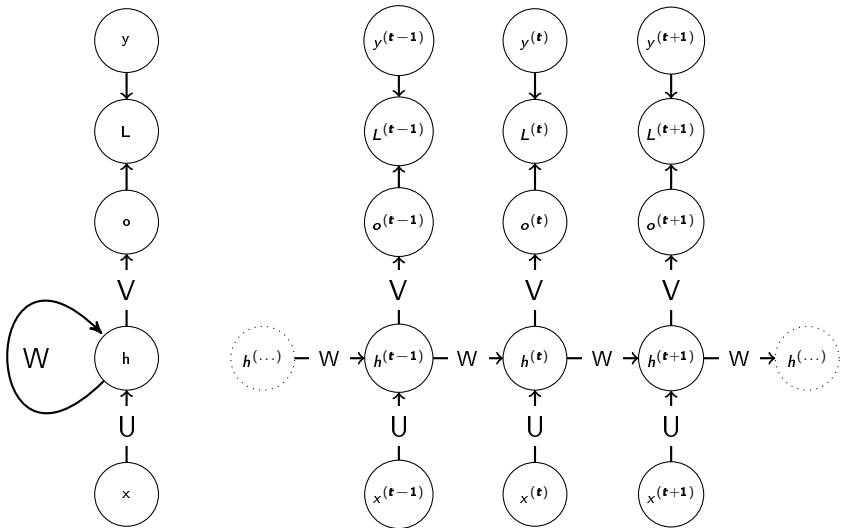
Long Term Dependencies

Hands on Programming

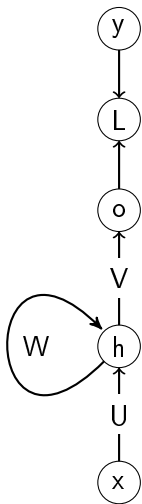
Recurrent Neural Network Example



Recurrent Neural Network Example



Forward Propagation



- ▶ start with initial state $h^{(0)}$
- ▶ then for $t = 1, \dots, \tau$:

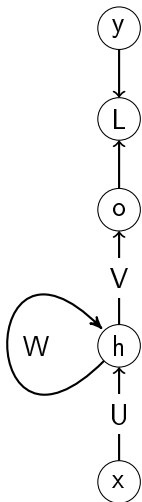
$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$$

$$h^{(t)} = \sigma(a^{(t)})$$

$$o^{(t)} = c + Vh^{(t)}$$

$$\hat{y}^{(t)} = \text{softmax}(o^{(t)})$$

Back-propagation and loss function

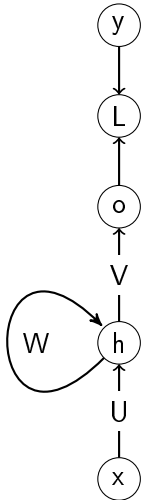


- ▶ let $L^{(t)}$ be the negative log-likelihood of $y^{(t)}$ given $x^{(1)}, \dots, x^{(t)}$
- ▶ sum the loss over all time steps

$$\begin{aligned} & L(\{x^{(1)}, \dots, x^{(\tau)}\}, \{y^{(1)}, \dots, y^{(\tau)}\}) \\ &= \sum_t L^{(t)} \\ &= - \sum_t \log p_{model}(y^{(t)} \mid \{x^{(1)}, \dots, x^{(t)}\}) \\ &= - \sum_t \log \hat{y}^{(t)} \end{aligned}$$

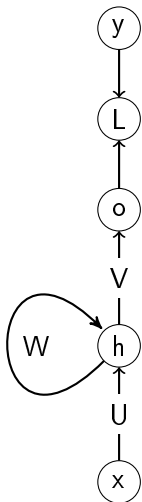
- ▶ use back-propagation through time (BPTT)

Back-propagation and loss function



- ▶ parameters are shared across all time steps
- ▶ therefore, the gradient is the sum over all time steps
- ▶ we need to introduce copies $W^{(t)}$ of W for each time step t

Back-propagation and loss function



- ▶ parameters are shared across all time steps
- ▶ therefore, the gradient is the sum over all time steps
- ▶ we need to introduce copies $W^{(t)}$ of W for each time step t

$$\nabla_W L = \sum_t \sum_i \left(\frac{\delta L}{\delta h_i^{(t)}} \right) \nabla_{W^{(t)}} h_i^{(t)}$$

Reminder

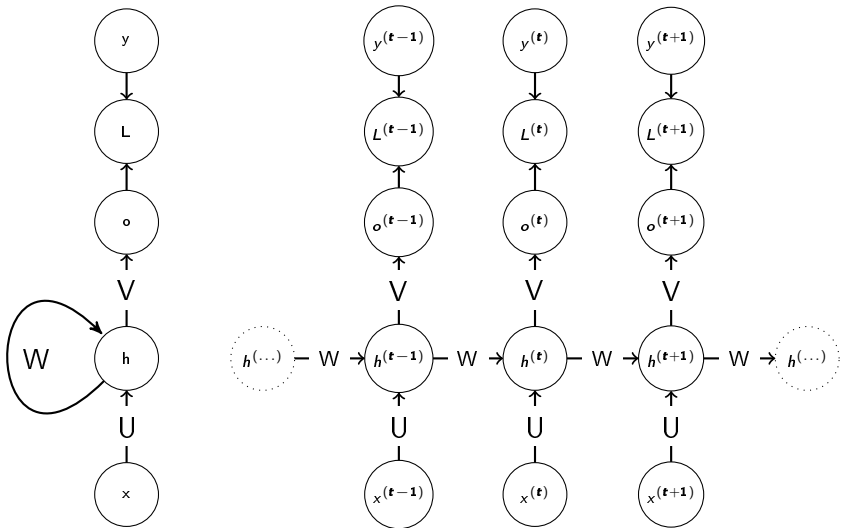
$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$$

$$h^{(t)} = \sigma(a^{(t)})$$

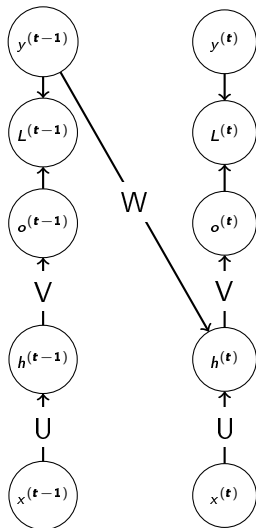
$$o^{(t)} = c + Vh^{(t)}$$

$$\hat{y}^{(t)} = \text{softmax}(o^{(t)})$$

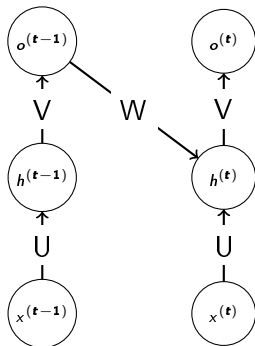
Back-propagation



Teacher forcing



Training Time vs. Test Time



Computational Power

Computational Power of RNN

Siegelmann and Sontag (1995) [?]

A universal Turing Machine can be simulated by a RNN of at most 886 neurons.

Church–Turing thesis

Turing machine

- ▶ infinite tape
- ▶ read / write head
- ▶ finite state register
- ▶ finite table of instructions (program)

Church-Turing thesis

The class of all Turing computable functions is the same as the class of all intuitively computable functions.

Church–Turing thesis

p-stack Turing machine

- ▶ p stacks
- ▶ p read / write heads
- ▶ push / pop / don't change stack
- ▶ state register
- ▶ table of instructions
- ▶ input and output: binary sequence on stack 1

Church-Turing thesis

The class of all Turing computable functions is the same as the class of all intuitively computable functions.

Key idea

- ▶ Three neurons per stack:
 - stack encoding
 - top element
 - `stack.isEmpty()`
- ▶ some other neurons which execute the “Turing machine”

Computational Power of RNN

$$\sigma(x) := \begin{cases} 0, & \text{if } 0 > x \\ x, & \text{if } 0 \leq x \leq 1 \\ 1, & \text{if } 1 < x \end{cases}$$

Lemma: Siegelmann and Sontag (1993) [?]

The computational power of a RNN which performs an update on neurons $x \in \mathbb{R}^n$ and inputs $u \in \mathbb{R}^m$ using a function $f = \psi \circ \phi$ is equivalent to a RNN which uses the activation function σ (up to polynomial differences in time).

$\phi : \mathbb{R}^{n+m} \rightarrow \mathbb{R}^n$ is a polynomial in $n + m$ variables

$\psi : \mathbb{R}^n \rightarrow \mathbb{R}^n$ has bounded range and is locally Lipschitz

Stack encoding

Represent a stack as binary string $\omega = \omega_1\omega_2 \dots \omega_n$. Encode this as

$$c(\omega) = \sum_{i=1}^n \frac{2\omega_i + 1}{4^i} \in [0, 1[.$$

- ▶ For the empty stack, set $c(\omega) = 0$.
- ▶ $\omega_1 = 1 \Leftrightarrow c(\omega) \geq \frac{3}{4}$
- ▶ $\omega_1 = 0 \Leftrightarrow c(\omega) \in [\frac{1}{4}, \frac{1}{2}[$
- ▶ e.g. for $\omega = 1011$:

$$c(1011) = 0.3133_4 = \frac{223}{256} \geq \frac{192}{256} = \frac{3}{4}$$

Stack operations

Let $q = c(\omega)$ be a stack encoding.

$$4q - 2 \begin{cases} \geq 1, & \text{if } q \geq \frac{3}{4} \\ \leq 0, & \text{if } q \leq \frac{1}{2} \end{cases}$$

Reading value of the top element:

$$\text{top}(q) = \sigma(4q - 2)$$

Stack operations

- ▶ Pushing 0 to stack $\omega = 1011$ gives $\tilde{\omega} = 01011$.
- ▶ $q = c(\omega) = 0.3133_4$
- ▶ $\tilde{q} = c(\tilde{\omega}) = 0.13133_4$

$$\tilde{q} = \frac{q}{4} + \frac{1}{4} = \sigma\left(\frac{q}{4} + \frac{1}{4}\right)$$

- ▶ Pushing 1 to stack $\omega = 1011$ gives $\tilde{\omega} = 11011$.
- ▶ $q = c(\omega) = 0.3133_4$
- ▶ $\tilde{q} = c(\tilde{\omega}) = 0.33133_4$

$$\tilde{q} = \frac{q}{4} + \frac{3}{4} = \sigma\left(\frac{q}{4} + \frac{3}{4}\right)$$

Stack operations

- ▶ Popping from stack $\omega = 1011$ gives $\tilde{\omega} = 011$.
- ▶ $q = c(\omega) = 0.3133_4$
- ▶ $\tilde{q} = c(\tilde{\omega}) = 0.133_4$

$$\tilde{q} = 4q - (2 \cdot \text{top}(q) + 1)$$

Stack operations

- ▶ Stack is empty $\Leftrightarrow q = 0$
- ▶ Stack is non-empty $\Leftrightarrow q \geq 0.1_4 = \frac{1}{4}$
- ▶ use

$$\text{empty}(q) = \sigma(4q) = \begin{cases} 0, & \text{if } q = 0 \\ 1, & \text{if } q \geq 0.1_4 \end{cases}$$

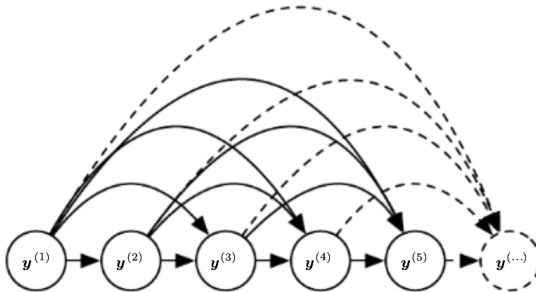
Universal Turing machine simulation

- ▶ create three neurons per stack to hold
 - q
 - $\text{top}(q)$
 - $\text{empty}(q)$
- ▶ some more neurons for states and computation
- ▶ can be used to simulate universal turing machine with at most 886 neurons
- ▶ in 2015, Carmantini *et al.* [?] constructed a RNN simulating a universal Turing machine with only 259 neurons

Variants of RNNs

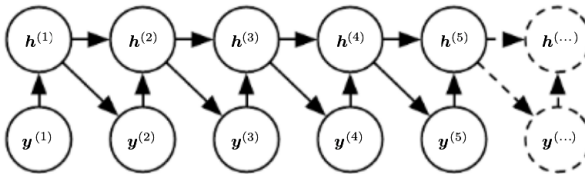
Directed graphical models

- ▶ directed acyclic graph
- ▶ nodes represent random variables
- ▶ edges show dependencies of variables

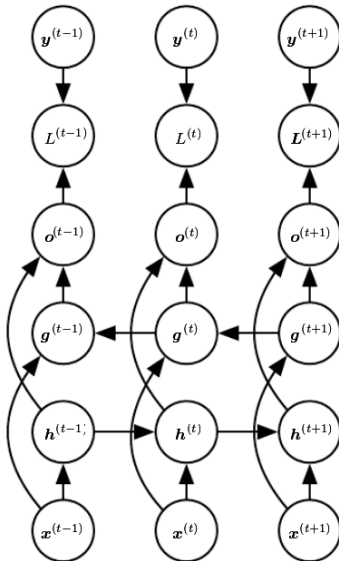


Directed graphical models with RNN

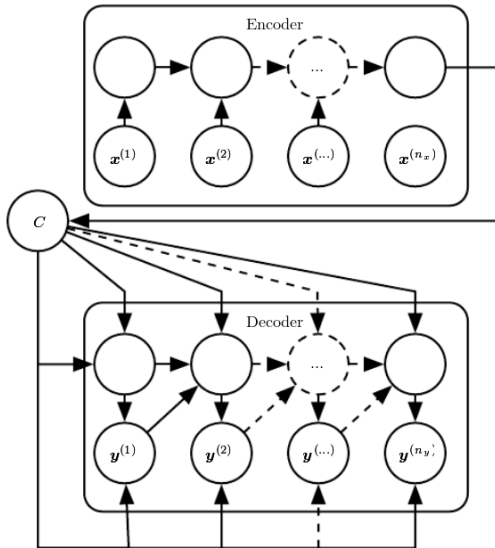
- ▶ RNN can represent the joint probability
- ▶ introduction of hidden units provides efficient parametrization
- ▶ number of parameters is independent of sequence length
- ▶ sparse representation, compared e.g. to table



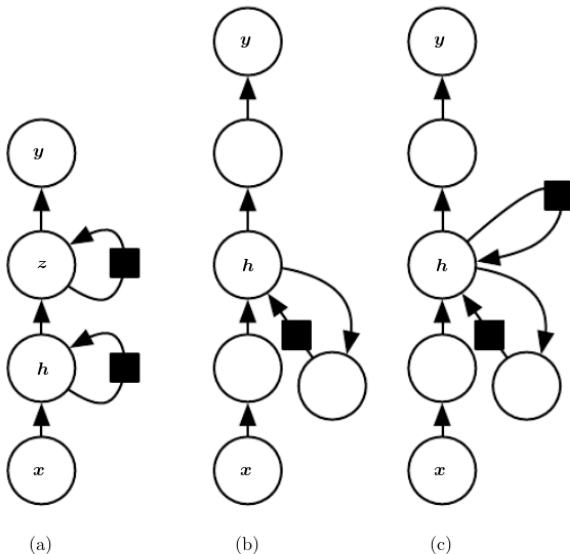
Bidirectional RNNs



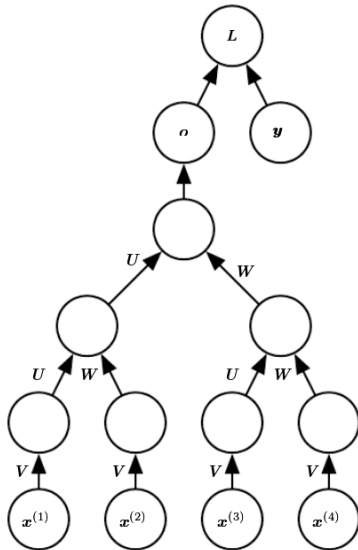
Encoder-Decoder Architectures



Deep Recurrent Networks



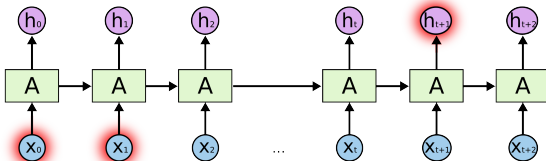
Recursive Neural Networks



Long Term Dependencies

Problem of Long Term Dependencies

- ▶ in RNN's gradients are propagated over many stages, which can lead to vanishing or exploding gradients
- ▶ the influence of an output vanishes over time
- ▶ challenge: How can we make information flow over long distances?



Techniques for Long Term Dependencies

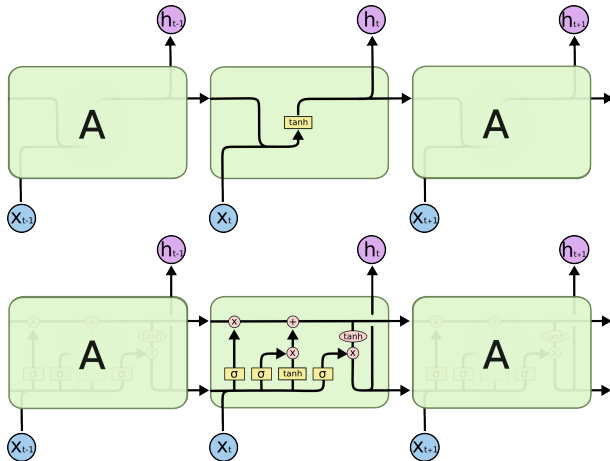
- ▶ Echo state networks
- ▶ Gated RNN's - LSTM

LSTM - Long Term Short Memory

Idea:

- ▶ have one state that flows over time and can be manipulated
- ▶ gates control the information that is passed to the state
- ▶ units are not connected through fixed weights but with dynamical connections

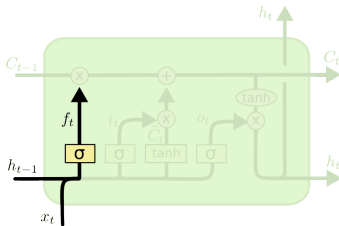
Difference between normal RNN and LSTM



Source: (Great Blog!)

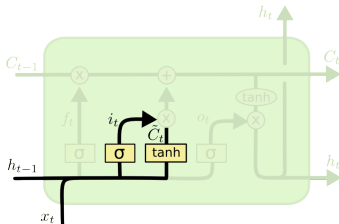
<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

The Forget-Gate



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

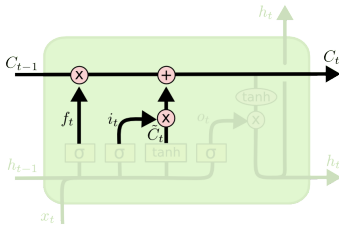
The Input-Gate



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

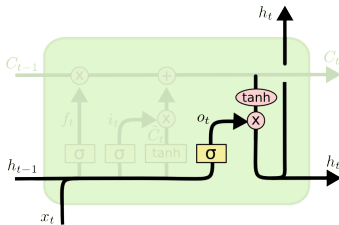
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

The State-Update



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

The Output-Gate



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

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

Hands on Programming

The Real World Problem

Semantic Segmentation



Source: University of Cambridge

My Toy-Example



..Greyscale Image



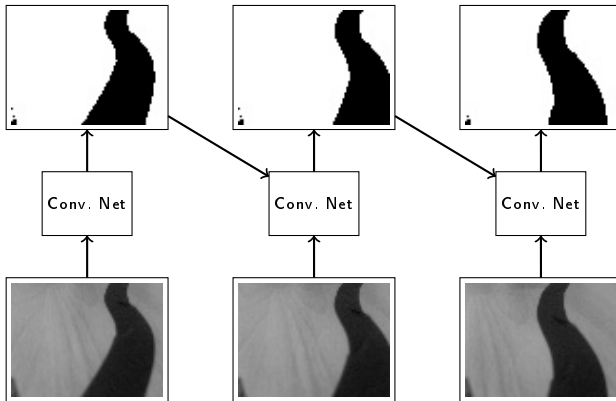
..Segmentation: Road- Not Road

The robot car

[IMAGE]

Light-sensors to follow the black line, webcam to take pictures

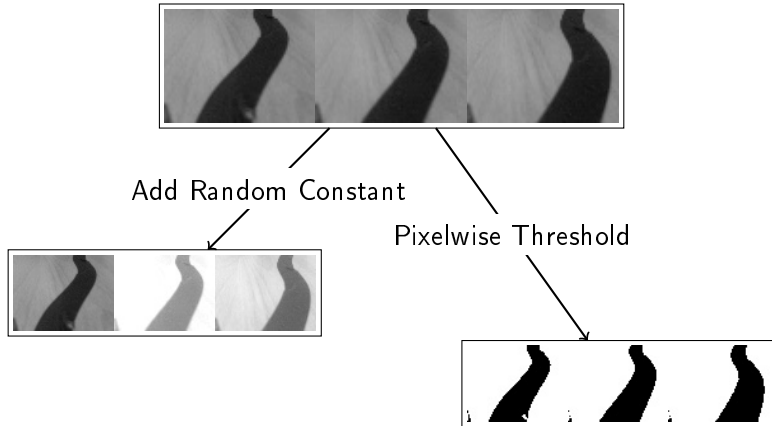
Idea



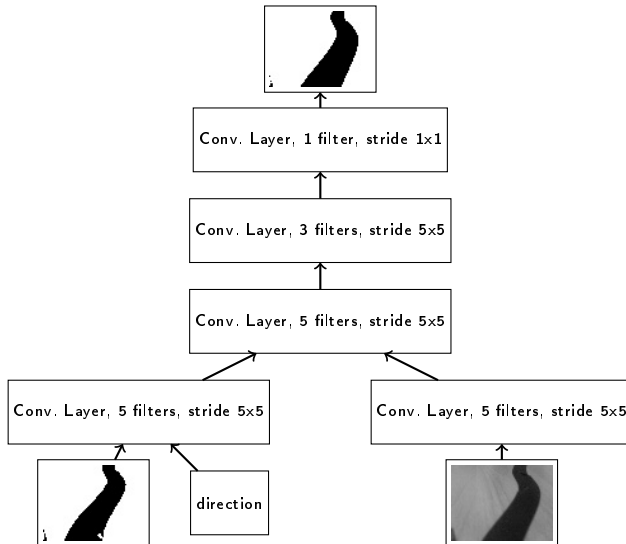
Challenges

- ▶ get a lot of training data
- ▶ generate the ground truth automatically
- ▶ make the task neither too complicated nor too easy

Generation of data and ground truth



Architecture of the RNN



Results

Pixelwise accuracy:

Image only: 92.45

Image+previous segmentation (RNN):98.64
(same architecture and training steps)

References