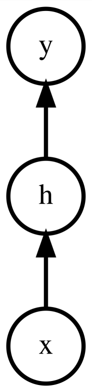
Unrolling Recurrent Neural Networks

Petru Rebeja

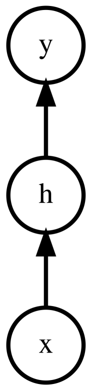
November 14, 2018

The shortcomings of a traditional Neural Network [Kar15b]



We all have seen the marvelous success achieved by neural networks (especially Convolutional Networks).

The shortcomings of a traditional Neural Network [Kar15b]



We all have seen the marvelous success achieved by neural networks (especially Convolutional Networks).

However, regardless of their success these networks have some shortcomings which deem them unfit for a wide range of tasks:

- Constrained to a **fixed size input** and produce **fixed size output**.
- The number of **computational steps** is **fixed**.

Working with data of variable size

- There are domains (Natural Language Processing for example) where the data samples don't have a fixed size; e.g. *documents* are a *sequence of tokens*.

Working with data of variable size

- There are domains (Natural Language Processing for example) where the data samples don't have a fixed size; e.g. *documents* are a *sequence of tokens*.
- Furthermore, some tasks require finding temporal patterns/dependencies or patterns/dependencies distributed across some projection of time.

Working with data of variable size

- There are domains (Natural Language Processing for example) where the data samples don't have a fixed size; e.g. *documents* are a *sequence of tokens*.
- Furthermore, some tasks require finding temporal patterns/dependencies or patterns/dependencies distributed across some projection of time.
- Detecting such patterns/dependencies in a computationally feasible way requires parameter sharing – a feature which Convolutional Networks are lacking.

Parameter sharing

- Allows generalization over samples of different lengths.
- Avoids overfitting the sample size from the training set.
- Reduces the number of parameters that the model has to learn.

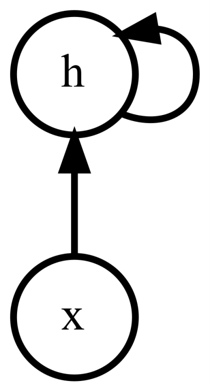
Using convolutions on sequential data [GBC16]

- Time-delay networks use convolutions across a temporal sequence by applying the same kernel at each time step.
- Although this approach allows for parameter sharing across time, *it is shallow*.
- The operation only captures a small number of neighboring members of the input.

Recurrent Neural Networks

- Are specialized for processing a sequence of values [GBC16]
- Offer a more powerful way of processing the data [Kar16] and are Turing-Complete [Sie95]
- Can process sequences of variable length [GBC16]
- Are a special case of **Recursive Neural Networks** which, in turn, are a separate topic (maybe for another presentation).

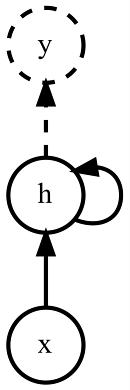
Computational Graph for Recurrent Networks



The theoretical computational graph of a Recurrent Network consists of two parts:

1. The *input* node which receives the current element of the sequence $x^{(t)}$
2. The *hidden* node which gets activated by both $x^{(t)}$ and hidden node state from previous step $h^{(t-1)}$

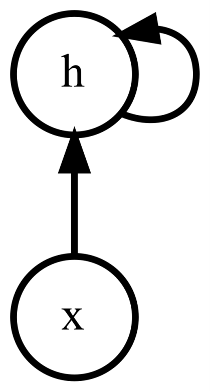
Computational Graph for Recurrent Networks



In practice we're also interested in outputting some values from the network (e.g. the probability of *w* being the next word in the generated sequence) thus we add extra nodes which compute:

- The output for the current element in the sequence
- The *loss* value of the output when in training etc.

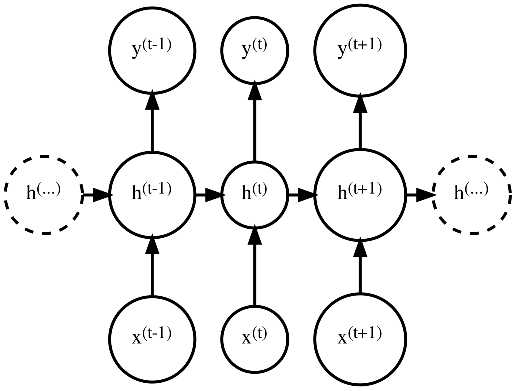
Unrolling computational graph



1. The problem with recurrent expressions is that they introduce cycles in the computational graph.
2. In order to avoid this, the computational graphs of Recurrent Networks are transformed through a process called **unrolling** or **unfolding**.

Unrolling computational graph

This transformation implies copying over a number of times the input, hidden and output nodes and connecting the hidden nodes between them.

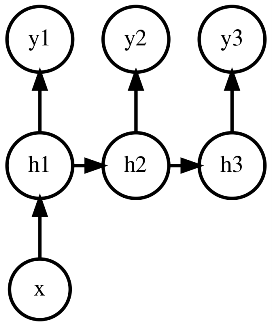


Unrolling computational graph

The unrolling process introduces two major advantages [GBC16]:

1. The input is interpreted as transitions from one fixed size state to another thus the sequence length does not influence neither the model nor the parameter size
2. *Same* transition function with same parameters can be applied at each step in the sequence

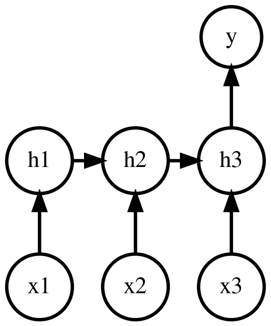
Also, unrolling introduces out-of-the box mini-batching.



One to many

- Given a fixed size input, the network outputs a sequence of values.
- e.g. *Image captioning* - The network takes an image and outputs a sequence of words.

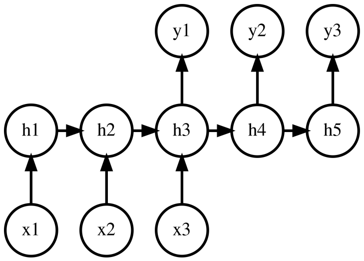
Flavors of Recurrent Networks [Kar15b] ii



Many to one

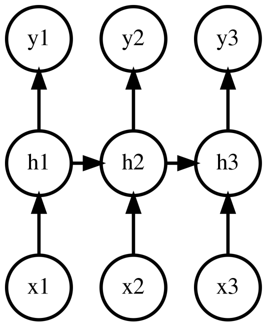
- Given a sequence of values, the network outputs a fixed size result.
- e.g. *Sentiment analysis* - The network takes a sentence (sequence of words) and classifies it as expressing positive, negative or neutral sentiment.

Flavors of Recurrent Networks [Kar15b] iii



Many to many (I)

- Given a sequence of values, the network outputs another sequence of values.
- e.g. *Machine translation* - The network takes a sentence in English and then outputs a sentence in Romanian.



Many to many (II)

- Given a sequence of values, the network outputs another sequence of values.
- e.g. *Video classification on frame level* - The network takes a sequence of video frames and classifies each one based on frame contents *and* the frames before.

An example i

Character-level language model with *tanh* activation and *softmax* normalization [Kar15a] [GBC16].

- **Forward pass**
  1. Start with some initial state h_0; usually initialized to zeroes
  2. For each $x^{(t)}$ in the mini-batch calculate:
     - Values for the hidden layer
     - (Normalized) Output probabilities
     - Loss

An example ii

- **Backward pass** – Start from the end of the sequence and back-propagate into each node
    1. Back-propagate into y
    2. Back-propagate into h
    3. Back-propagate through *tanh*

An example iii

- **Gradient clipping**
  - The recurrence formula is basically an exponentiation of W_{xh} matrix
  - Thus when its elements are small they risk *vanishing* and when they are large they may *explode*
  - A common practice to avoid this is to apply **clipping**

An example iv

- **Parameter update**
  - Update model parameters according to the optimization technique

Thank you!

References i

- Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep learning*, MIT Press, 2016, `http://www.deeplearningbook.org`.
- Andrej Karpathy, *Minimal character-level language model with a vanilla recurrent neural network*, `https://gist.github.com/karpathy/d4dee566867f8291f086/`, 2015, [Online; accessed November 12 2018].
- ———, *The unreasonable effectiveness of recurrent neural networks*, `https://karpathy.github.io/2015/05/21/rnn-effectiveness/`, 2015, [Online; accessed November 12 2018].

References ii

- _______, *Cs231n winter 2016 lecture 10 recurrent neural networks, image captioning, lstm*, `https://youtu.be/cO0a0QYmFm8`, 2016, [Online; accessed November 12 2018].
- Hava T Siegelmann, *Computation beyond the turing limit*, Science **268** (1995), no. 5210, 545–548.