Deep Learning for Sequence Models

Anthony Faustine

PhD researcher machine learning
(IDLab, imec, research group at Ghent University)

20 August 2018

Learning goal

- Understand sequence models and limitation of MLP and CNN for these model.
- Learn the structure of RNN.
- Understand how to train RNN and limitation of RNN
- Learn Gated RNN (LSTM and GRU)
- Understand how to implement RNN, LSTM and GRU using Pytorch framework.

Outline

Introduction

Suppose that we are given an input sequence

$$\mathbf{x} = \{x_1, \dots x_T\}$$

And wish to predict some corresponding output

$$\mathbf{y} = \{y_1, \dots y_T\}$$

- At each time y_t is predicted using only those inputs that have been previously observed $x_1, \dots x_t$
- Thus a sequence model is a function $f_\theta : \mathbf{x}^{T+1} \rightarrow \mathbf{y}^{T+1}$ such that:

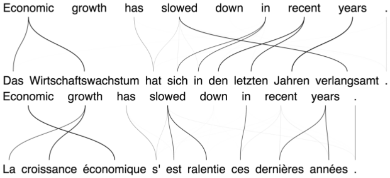
$$\hat{y}_1 \dots \hat{y}_T = f_\theta(x_1, \dots x_T)$$

- y_t depend only on $x_1, \dots x_t$ and not on any future inputs $x_{t+1}, \dots x_T$
- Each $\mathbf{x}$ and $\mathbf{y}$ consist of a pair of sequences that can have different lengths with variable time horizon.

Introduction

Many real world problems require processing a signal with sequence structure.

Machine translation



$$P(y_1 \dots y_M | x_1, \dots x_N)$$

Image Captioning:

"man in black shirt is playing guitar." "construction worker in orange safety vest is working on road." "two young girls are playing with lego toy."

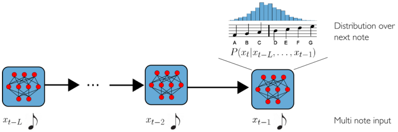
$$P(y_1 \dots y_M | I)$$

Sequence data

Speach recognition

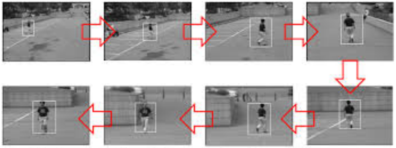
$$P(y_1 \dots y_M | x_1, \dots x_N)$$

Music generation:



$$P(y_1 \dots y_M | I)$$

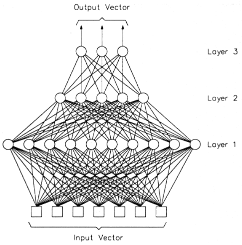
Object tracking



Moving object tracking results by using a PTZ camera

Time series signal

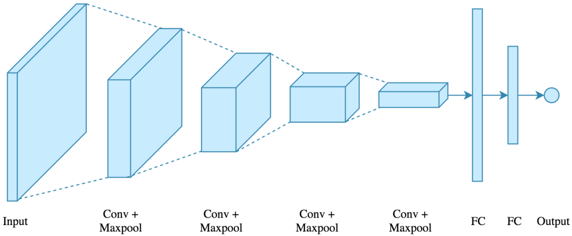
Introduction: Can we use MLP



MLP as universal function aproximator has limitation for such problems.

- Take fixed sized input and generates fixed sized output
- Does not share feature learned across different position of a sequence $\Rightarrow$ feature learned about the sequence won't transfer if they appear at different points in the sequence.
- It treat every example independently $\Rightarrow$ does not care about what happened at previous time step

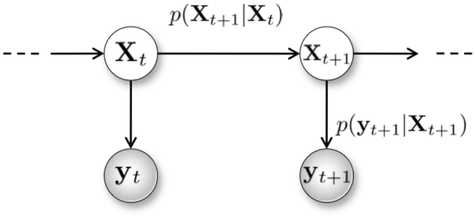
Sequential Modelling: What about CNN



CNN

- Does not handle arbitrary input/output lengths.
- Does not provide best way to handle long-term dependencies.

Sequential Modelling: Markov Models



Markov assumption:Each state depends only on the last state.

- Does not model long-term dependencies.

Sequential Modelling

To effectively model sequences we need to:

- Deal with variable-length sequences and maintain sequence order
- Keep track of long-term dependencies and
- Share parameters across the sequence

Outline

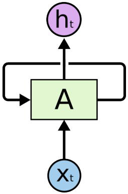
RNN

RNN: family of neural network for handling sequential data.

- It use the same idea like CNN for parameter sharing across different part of the model $\Rightarrow$ allow sharing of statistical features and generalize to unseen sequence during training.
- Each output is a function of the previous output with the same update rule applied.

RNN: Recurrence

RNN maintain a recurrent state updated at each time step t.

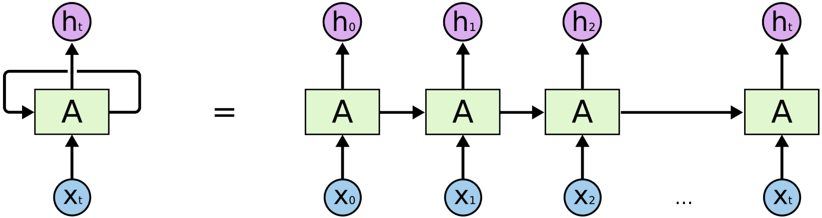


$$h^{(t)} = f_\theta(h^{(t-1)}, x_t)$$

This is recurrence because the definition of h at time t refer back to to the same definition at time $t-1$

The state $h^{(t)}$ is called hidden state $\Rightarrow$contain information about the whole past sequence.

RNN



Vanilla RNN

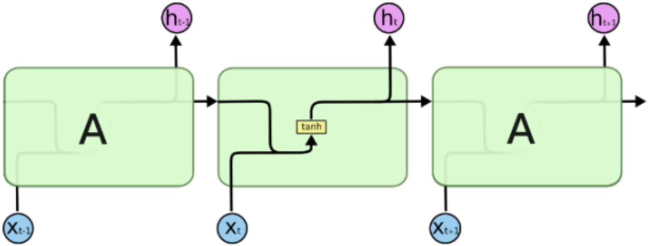


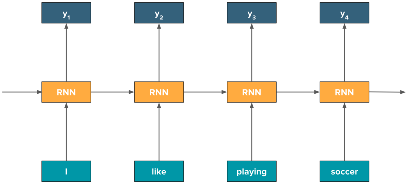
Figure 1: RNN cell

$$h_{(t)} = \tanh(W_{xh} \cdot x_t + W_{hh} \cdot h_{(t-1)} + b_h)$$
$$y_{(t)} = \sigma(W_{hy} \cdot h_{(t)} + b_y)$$

where $\sigma(.)$ is output activation function

- The out layer can read information from $h^{(t)}$ to make prediction.
- Such architecture is called vanilla RNN $\Rightarrow$ produce output at each time steps.

Vanilla RNN



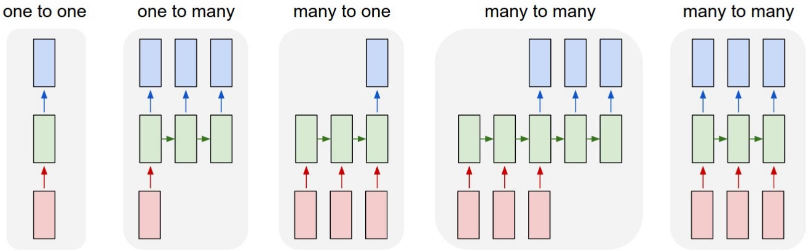
Consider machine translation problem from English to Swahili

- I like playing soccer → Napenda cheza mpira wa miguu.

Is the architecture above suitable for this problem?

Different types of RNNs

RNN variants



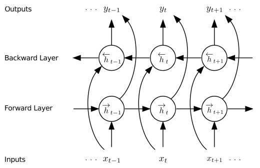
RNN Variants

From left to right:

1. Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification).
2. Sequence output (e.g. image captioning takes an image and outputs a sentence of words).
3. Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment).
4. Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French).
5. Synced sequence input and output (e.g. video classification where we wish to label each frame of the video).

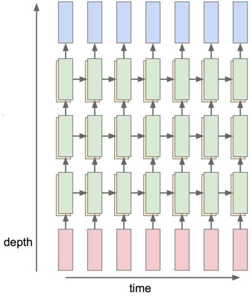
RNN offer lot of flexibility

Bidirectional RNN



- Process input sequence in forward and in reverse direction.
- Popular in speech recognition and machine transaltion.

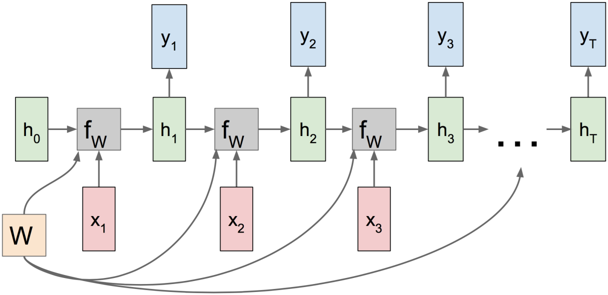
Deep RNN



Outline

Training RNN: BTT

Consider RNN graph



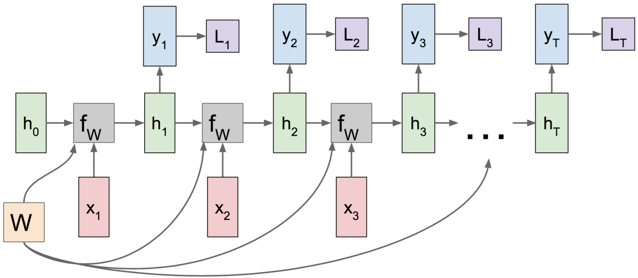
where

$$h_{(t)} = \tanh(W_{xh} \cdot x_t + W_{hh} \cdot h_{(t-1)} + b_h)$$
$$y_{(t)} = \sigma(W_{hy} \cdot h_{(t)} + b_y)$$

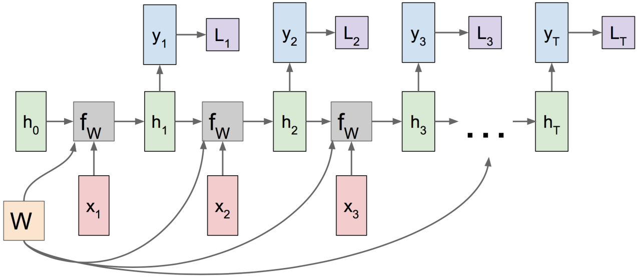
Training RNN: BTT

The loss is the sum of losses over time steps.

$$\mathcal{L}(\{x_1, \ldots x_t\}, \{y_1, \ldots y_t\}) = \sum_t L_t = -\sum_t \log p(y_t | x_1, \ldots x_t)$$



Training RNN: BTT



Need to find derivative for each parameters θ

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_t \frac{\partial \mathcal{L}_t}{\partial \theta_t}$$

Training RNN: BTT

It can be shown that: for each parameters θ

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_{t=0}^{T} \frac{\partial \mathcal{L}}{\partial y_T} \cdot \frac{\partial y_T}{\partial h_T} \cdot \frac{\partial h_T}{\partial h_t} \cdot \frac{\partial h_t}{\partial \theta}$$

where:

$$\frac{\partial h_T}{\partial h_t} = \frac{\partial h_T}{\partial h_{T-1}} \cdot \frac{\partial h_{T-1}}{\partial h_{T-2}} \cdots \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial h_0}$$

For larger sequence length T this product $\frac{\partial h_T}{\partial h_t}$ gets longer and longer!

Training RNN: BTT

Each $\frac{\partial h_T}{\partial h_{T-1}}$ term is equal to:

$$\frac{\partial h_T}{\partial h_{T-1}} = \mathbf{W}^T(diag)f^{'}(W_{hh} \cdot h_{(t-1)} + W_{xh} \cdot x_t)$$

where: $f(.)$ is activation function which can be tanh() or sigmoid

Two problems

1. Vanishing gradient: In most cases $f^{'} < 1$ and $\mathbf{W} < \mathbf{1} \Rightarrow$ sampled from normal distribution.
   - multiplying a lot of small gradients together $\rightarrow$ gradient shrink and vanish.
2. Exploding gradient: In rarely cases $\frac{\partial h_T}{\partial h_{T-1}} > 1$.
   - multiplying a lot of large gradients together $\rightarrow$ gradient explode.

Training RNN: Long-Term Dependencies

Problem:Gradients propagated over many stages tend to vanish (most of the time) or explode (relatively rarely)

- Parameters become biased to capture shorter-term dependencies $\Rightarrow$ miss long term dependencies.
- The network can not learn over long input sequences.

One approach to address vanishing gradient is to use gated cell.

- Add gates to control what information is passed through.
- Two common gated cell:
  1. Long Short-Term Memory (LSTM)
  2. Gated Recurrent Unit (GRU)

Outline

Long Short-Term Memory (LSTM)

LSTM is an extension of RNN designed to address RNN's vanishing gradient problem.

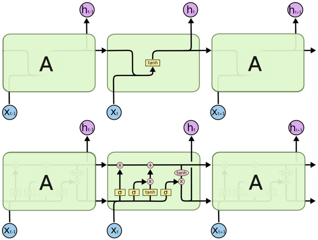


Figure 2: RNN vs LSTM

It uses three specialized gates;

1. forget gate modulate what information to forget from a cell state,
2. input/update gate decides what information is going to stored in the cell state.
3. output gate control weather to output the state

Long Short-Term Memory (LSTM)

The recurrent state is composed of a cell state c_t and an hidden state h_t.

- cell state: stores contextual and longer term information
- hidden state: stores immediately necessary information

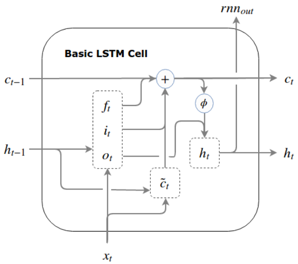


Figure 3: LSTM cell

$$f_t = \sigma\left(W_{xf} \cdot x_t + W_{hf} \cdot h_{t-1} + b_f\right)$$
$$i_t = \sigma\left(W_{xi} \cdot x_t + W_{hi} \cdot h_{t-1} + b_i\right)$$
$$o_t = \sigma\left(W_{xo} \cdot x_t + W_{ho} h_{t-1} + b_o\right)$$

Long Short-Term Memory (LSTM)

- The recurrent state c_t has an update with gating i_t and full update $\tilde{c}_t$ such that:

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

where

$$\tilde{c}_t = \tanh\left(W_{xc} \cdot x_t + W_{hc} \cdot h_{t-1} + b_c\right)$$

- This assures that the derivatives of the loss wrt c_t does not vanish

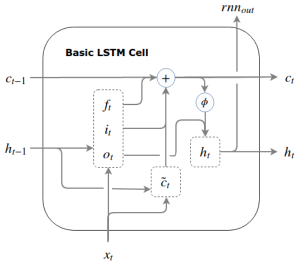


Figure 4: LSTM cell

Long Short-Term Memory (LSTM)

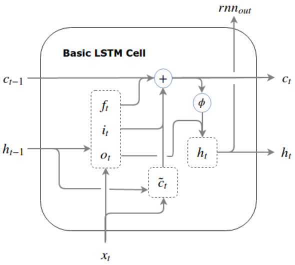


Figure 5: LSTM cell

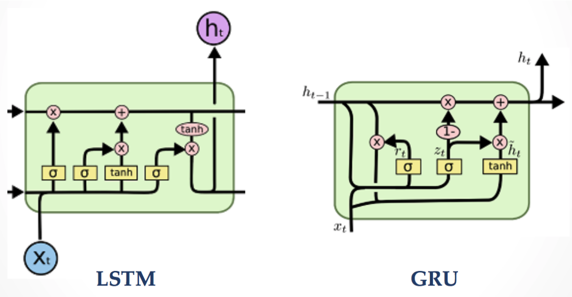
- The new state update h_t has an update with gating o_t and c_t such that

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

  where $\odot$ is the Hadamard (element-wise) multiplication
- At any time t the LSTM output is equal to h_t

Gated Recurrent Unit (GRU)

GRU: a very simplified version of LSTM



- Merges forget and input gate into a single update gate
- It also merge cell and hidden state.
- It has reset gate r_t and update gate z_t

GRU

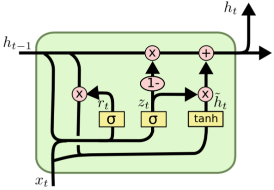


Figure 6: gru cell

$$r_t = \sigma\left(W_{xr} \cdot x_t + W_{hr} \cdot h_{t-1} + b_r\right)$$
$$z_t = \sigma\left(W_{xz} \cdot x_t + W_{hz} \cdot h_{t-1} + b_z\right)$$

The recurrent state h_t has an update with gating z_t and full update $\tilde{h}_t$ such that:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t \text{ where}$$
$$\tilde{h}_t = \tanh\left(W_{xh} \cdot x_t + W_{hh} \cdot [r_t \odot h_{t-1}] + b_h\right)$$

GRU

- If $r_t \sim 0$ it ignore previous hidden state and only store the new input.
  - Allows model to drop information that is irrelevant in the future.
- z_t control how much of past state should matter now
- The final memory h_t combine both current and previous time-steps.

There isn't a universal superior between LSTM and GRU.

- One of the advantages of GRU is that it's simpler and can be used to build much bigger network but the LSTM is more powerful and general.