Neural Networks for Computing

Overfitting

Hikmat Farhat

November 27, 2017

Introduction

- We have already seen convolution networks that are typically used for processing input represented as a grid (image)
- Recurrent Neural Networks (RNN) are a **family** of neural networks
- Typically used for processing **sequential** data.
- An important part of RNN is parameter sharing across different parts of the model
- Parameter sharing is important to learning features independently of their position in the sequence.

Input data

- We will refer to the input data as a sequence x^t
- The index t is called the time step even if it doesn't necessarily refer to physical time.
- Unlike feedforward networks, RNN have feedback loops
- These loops represent the influence of previous values on the current value.
- The loops typically are unfolded for the computation.

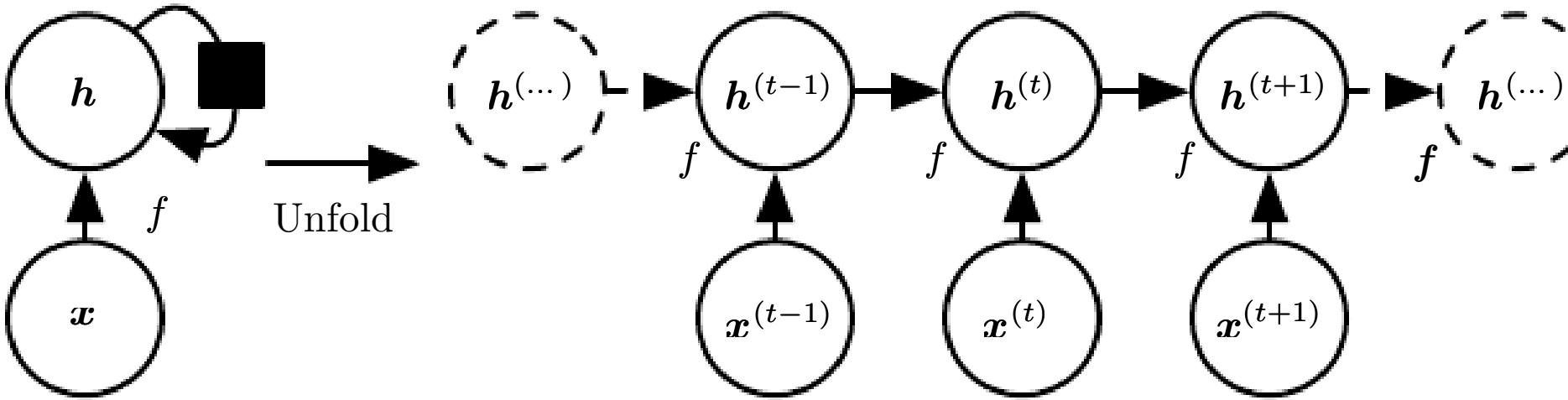

Figure: Example

Computation

- At each time step the RNN maintains a state.
- Given a time step t the state of the RNN at time t depends on the input at time t **and** the previous state
- For input The computation over a RNN can be write

$$h^t = f(h^{t-1}, x^t; \theta)$$

- Where θ represents the parameters of the system
- Note that f in the equation above is independent of t
- So it is applicable for every time step.

Unfolded computation

- By repeatedly applying the function f above we get

$$h^t = f(f(f(...f(h^0, x^0; \theta))$$

- Since f is the same at each time step we can "learn" f and generalized it to sequences not in the training set.
- When we do the computation we usually use the unfolded graph.
- The "folded" or recurrent version is usually more succinct.

Learning in RNN

- Below is a typically RNN used for learning

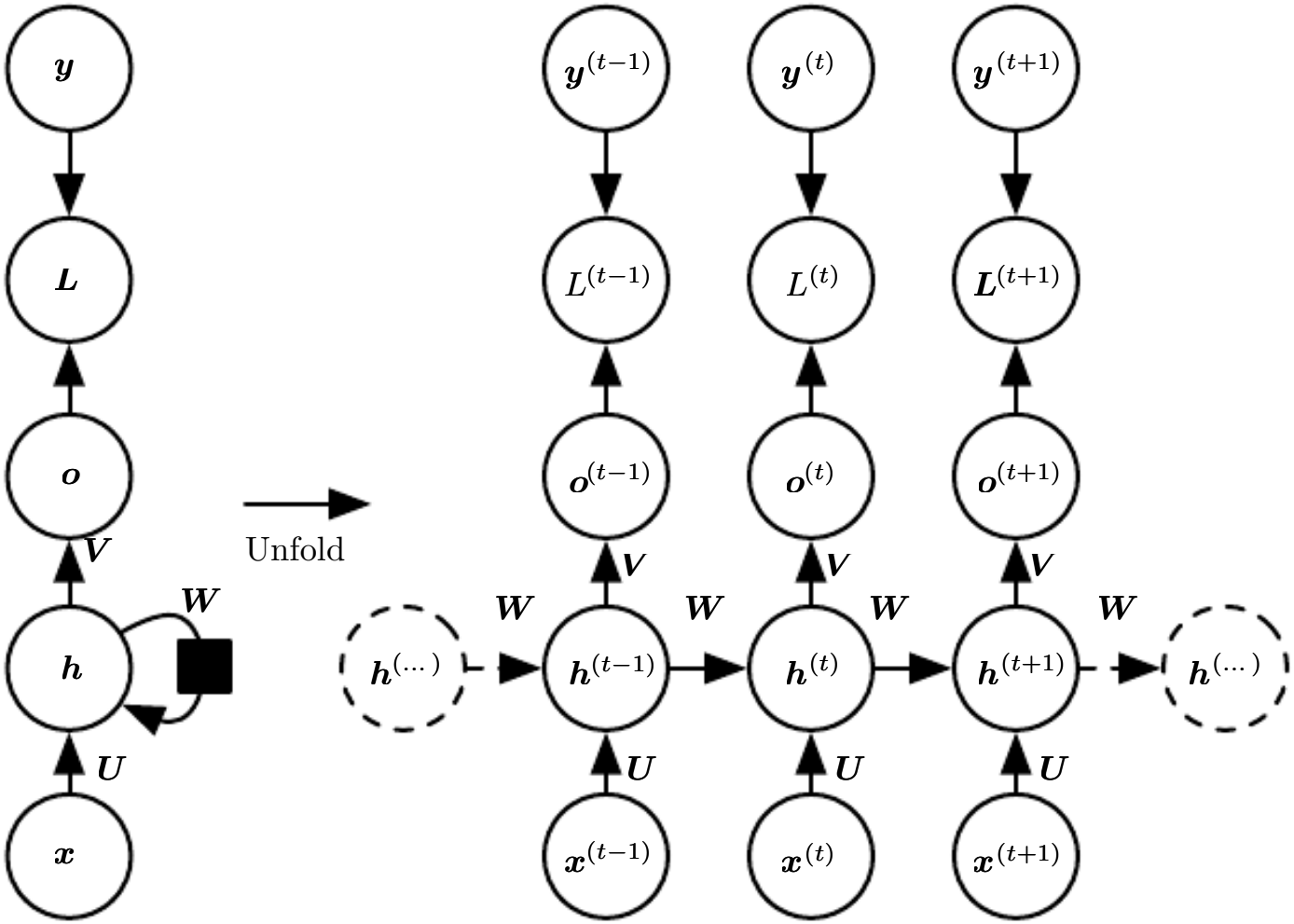

Figure: Learning RNN

Learning Equations

- Let x^t, h^t be the input and the state at time t.
- The computation steps in the figure above can be written as (we are stacking the data row-wise as in tensorflow)
- Once the outputs $\hat{y}^t$ is computed the cross-entropy is evaluated using that value and the "true" label of the data.

$$
\begin{aligned}
a^t &= h^{t-1} \cdot W + x^t \cdot U + b \\
h^t &= \tanh(a^t) \\
o^t &= h^t \cdot V + c \\
\hat{y}^t &= softmax(o^t)
\end{aligned}
$$

Example

- We will build a RNN to predict the values of a sequence that is dependent on another sequence
- The input, X, is a sequence of random 0's and 1's with 50% probability for each.
- The output or "labels" are a sequence Y of 0's and 1's with the following probability of occurence
  1. $Y(t)$ has probability of 50% of being a 1 if $X(t-3) = 0$ and $X(t-8) = 0$.
  2. $Y(t)$ has probability of 100% of being a 1 if $X(t-3) = 1$ and $X(t-8) = 0$.
  3. $Y(t)$ has probability of 25% of being a 1 if $X(t-3) = 0$ and $X(t-8) = 1$.
  4. $Y(t)$ has probability of 75% of being a 1 if $X(t-3) = 1$ and $X(t-8) = 1$.

Theoretical results

- We expect that if our RNN learns no dependencies between the input and output that the cross-entropy=0.66
- if our RNN learns the dependency on $X(-3)$ only then the cross-entropy=0.52
- if our RNN learns both dependencies then cross-entropy=0.45

Implementation details

- To implement our RNN model we need to unroll it.
- How far should we unroll?
- If the create less than 3 duplicates of our cell then gradient descent will not be able to learn the dependencies
- If we have less than 8 duplicates then again the gradient descent will learn only the first dependency (with some spill over)

Data batches

- The generated data goes through multiple transformations
- First a random X and the corresponding y are generated: a sequence of 0's and 1's of size N
- Then the data is divided into batches, of batch size B, and stacked together row-wise.

$$\begin{bmatrix} x_1 & \dots & x_n \end{bmatrix}$$

$$\begin{matrix} x_1 \dots & & x_{N/B} \\ \dots\dots & & \dots \end{matrix}$$