Lecture 5: Recurrent Neural Networks

Nima Mohajerin

University of Waterloo

WAVE Lab

nima.mohajerin@uwaterloo.ca

July 4, 2017
Overview

1. Recap
2. RNN Architectures for Learning Long Term Dependencies
3. Other RNN Architectures
4. System Identification with RNNs
RNNs deal with sequential information.

RNNs are dynamic systems. Frequently their dynamic is represented via state-space equations.
A simple RNN in discrete-time domain:

\[ x(k) = f(Ax(k-1) + Bu(k) + b_x) \]
\[ y(k) = g(Cx(k) + b_y) \]

- \( x(k) \in \mathbb{R}^s \): RNN state vector, no. of states = no. of hidden neurons
- \( y(k) \in \mathbb{R}^n \): RNN output vector, no. of output neurons = \( n \)
- \( u(k) \in \mathbb{R}^m \): Input vector to RNN (Independent input)
  - \( A \in \mathbb{R}^{s \times s} \): State feedback weight matrix
  - \( B \in \mathbb{R}^{s \times m} \): Input weight matrix
  - \( b_x \in \mathbb{R}^s \): Bias term
  - \( C \in \mathbb{R}^{n \times s} \): State to output weight matrix
  - \( b_y \in \mathbb{R}^n \): Output bias
Back Propagation Through Time

One data sample:

- **Input:** \( U = [u(k_0 + 1) \ u(k_0 + 2) \ldots u(k_0 + T)] \).
- **Output:** \( Y_t = [y_t(k_0 + 1) \ y_t(k_0 + 2) \ldots y_t(k_0 + T)] \).
- **SSE cost (per sample):**
  \[
  L = 0.5 \sum_{k=1}^{T} e(k_0+k)^\top e(k_0+k) = 0.5 \sum_{k=1}^{T} \sum_{i=1}^{n} (y_i(k_0+k) - y_{t,i}(k_0+k))^2
  \]
- **Batch cost (batch size = D):**
  \[
  L = 0.5 \sum_{d=1}^{D} \sum_{k=1}^{T} e_d(k_0 + k)^\top e_d(k_0 + k)
  \]
Gradients

To do a derivative-based optimization, we need the gradient of $L$:

$$
\frac{\partial L}{\partial a_{ij}} = \sum_{k=1}^{T} e^\top(k_0 + k) \frac{\partial e(k_0 + k)}{\partial a_{ij}} = \sum_{k=1}^{T} e^\top(k_0 + k) \frac{\partial y(k_0 + k)}{\partial a_{ij}}
$$

$$
\frac{\partial y(k)}{\partial a_{ij}} = \frac{\partial (Cx(k) + b_y)}{\partial a_{ij}} g'(Cx(k) + b_y) = C \frac{\partial x(k)}{\partial a_{ij}} g'(Cx(k) + b_y)
$$

$$
\frac{\partial x(k)}{\partial a_{ij}} = \frac{\partial v(k)}{\partial a_{ij}} f'(v(k)), \quad v(k) = Ax(k - 1) + Bu(k) + b_x
$$

$$
\frac{\partial v(k)}{\partial a_{ij}} = \frac{\partial A}{\partial a_{ij}} x(k - 1) + A \frac{\partial x(k - 1)}{\partial a_{ij}} = \begin{bmatrix} 0 \\ \vdots \\ x_j(k - 1) \\ 0 \\ \vdots \\ s \times 1 \end{bmatrix} + A \frac{\partial x(k - 1)}{\partial a_{ij}}
$$
Section 2

RNN Architectures for Learning Long Term Dependencies
Gated Architectures

\[ g(x) = x \odot \sigma(Ax + b) \]

\( \odot \): element-wise multiplication

\( x \in \mathbb{R}^n \)

\( g(x) \in \mathbb{R}^n \)

\( A \in \mathbb{R}^n \times \mathbb{R}^n \)
The idea is that if a neuron has a self-feedback with weight equal to one, the information will retain for an infinite amount of time when unfolded.

Some information should decay, some should not be stored.

With a gate the intention is to control the self-feedback weight.
Gated Recurrent Unit

\[
\begin{align*}
g_f(k) &= \sigma(W_f^i u(k) + W_f^o y_h(k - 1)) \\
g_i(k) &= \sigma(W_i^i u(k) + W_i^o y_h(k - 1)) \\
m(k) &= h(W_i^y u(k) + W_y^y (g_f \odot y_h(k - 1))) \\
y_h(k) &= g_i(k) \odot y_h(k - 1) + (1 \odot g_i(k)) \odot m(k)
\end{align*}
\]
Long Short Term Memory Cell

\[
\begin{align*}
    g_i(k) &= \sigma \left( W_i^i u(k) + W_i^o y_h(k - 1) + W_i^c c(k - 1) + b_i \right) \\
    g_f(k) &= \sigma \left( W_f^i u(k) + W_f^o y_h(k - 1) + W_f^c c(k - 1) + b_f \right) \\
    g_o(k) &= \sigma \left( W_o^i u(k) + W_o^o y_h(k - 1) + W_o^c c(k - 1) + b_o \right) \\
    c(k) &= g_i(k) \odot f \left( W_c^i u(k) + W_c^o y_h(k - 1) \right) + g_f(k) \odot c(k - 1) \\
    m(k) &= c(k) \odot g_o(k) \\
    y_h(k) &= h(W_y m(k - 1) + b_y)
\end{align*}
\]
One way to avoid gradient *exploding* is to **clip** the gradient:

\[
\text{if } ||g|| > \nu, \quad g \leftarrow \frac{g\nu}{||g||}
\]

One way to address vanishing gradient is to use a regularizer that **maintains the magnitude** of the gradient vector (Pascanu et al. 2013):

\[
\Omega = \sum_k \left( \frac{||\nabla x(k) L \frac{\partial x(k)}{\partial x(k-1)}||}{||\nabla x(k) L||} - 1 \right)^2
\]
Section 3

Other RNN Architectures
An RNN is a nonlinear chaotic system.

It can have many attractors in its phase space.

Hopfield (1985) model is the most popular one. It is a fully connected recurrent model where the feedback weight matrix is symmetric and has diagonal elements equal to zero.

Hopfield model is stable in a Lyapunov sense if the output neurons are updated one at a time. (Refer to Du KL, Swamy MNS (2006) Neural networks in a softcomputing framework doe further discussion)
RNNs as Associative Memories
RNNs as Associative Memories

Distorted input

Gravity

RNN state transitions

Final output
Deep RNNs
Deep RNNs

We can generalize this idea and create a *connection matrix*:

\[
C = \begin{bmatrix}
C_{11} & C_{12} & C_{13} & C_{14} \\
C_{21} & C_{22} & C_{23} & C_{24} \\
C_{31} & C_{32} & C_{33} & C_{34} \\
C_{41} & C_{42} & C_{43} & C_{44}
\end{bmatrix}
\]

- From input
- From layer 1
- From layer 2
- From layer 3

To layer 1
To layer 2
To layer 3
To output
**Deep RNNs**

Example:

\[
C = \begin{bmatrix}
(\text{to L1}) & (\text{to L2}) & (\text{to L3}) & (\text{to output}) \\
1 & 0 & 1 & 0 & (\text{from input}) \\
1 & 0 & 1 & 1 & (\text{from L1}) \\
1 & 0 & 1 & 0 & (\text{from L2}) \\
0 & 1 & 1 & 1 & (\text{from L3}) 
\end{bmatrix}
\]
Reservoir Computing

- One approach to cope with the difficulty of training RNNs.
- The idea is to use a very large RNN, as a reservoir and use it to transform the input.
- The transformed input by the RNN is then linearly combined to form the output.
- The linear weights are trained while the reservoir (RNN) is fixed.
- Echo State Networks (continuous output neurons), Liquid State Machines (spiking binary neurons)
How to set the reservoir weights?

Set weights in such a way that the RNN is at the edge of stability: set the eigenvalues of the state Jacobian close to one.

\[ J(k) = \frac{\partial x(k)}{\partial x(k-1)} \]

Echo State Networks (continuous output neurons), Liquid State Machines (spiking binary neurons)
Section 4

System Identification with RNNs
We have a set of observations, i.e., measurements of a dynamic system input and output (states).

We want to learn the system dynamics

RNNs are universal approximators for dynamic systems

(K. Funahashi and Y. Nakamura, 1993)

**Delay embedding theorem** (Taken’s theorem) States that a chaotic dynamical system can be reconstructed from a sequence of observations of the system.

It leads to Auto-Regressive with eXogenous (ARX) models.
Nonlinear Auto-Regressive with eXogenous Inputs

\[ y(k) = F(u(k), u(k - 1), \ldots, u(k - d_x), y(k - 1), \ldots, y(k - d_y)) \]

- \( F \) can be constructed using a neural network. Typically an MLP is used.
Teacher Forcing (Parallel Training)

- Is mainly used in NARX architectures.
- Substitute the past network predictions with the targets.

$$ y(k) = F(u(k), u(k-1), \ldots, u(k-d_x), y_t(k-1), \ldots, y_t(k-d_y)) $$

- Converts the RNN to a FFNN (Single-step prediction).