Lecture 5: Recurrent Neural Networks

Nima Mohajerin

University of Waterloo

WAVE Lab
nima.mohajerin@uwaterloo.ca

July 4, 2017

Overview

RNNs

- **RNN**s deal with sequential information.
- **RNN**s are dynamic systems. Frequently their dynamic is represented via state-space equations.

A simple RNN

- A simple RNN in discrete-time domain:

$$\mathbf{x}(k) = \mathbf{f}\big(\mathbf{A}\mathbf{x}(k-1) + \mathbf{B}\mathbf{u}(k) + \mathbf{b}_x\big)$$
$$\mathbf{y}(k) = \mathbf{g}\big(\mathbf{C}\mathbf{x}(k) + \mathbf{b}_y\big)$$

$\mathbf{x}(k) \in \mathbb{R}^s$: RNN state vector, no. of states = no. of hidden neuron

$\mathbf{y}(k) \in \mathbb{R}^n$: RNN output vector, no. of output neurons $= n$

$\mathbf{u}(k) \in \mathbb{R}^m$: Input vector to RNN (Independent input)

$\mathbf{A} \in \mathbb{R}^s \times \mathbb{R}^s$: State feedback weight matrix

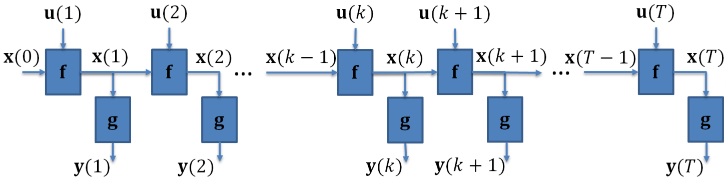
$\mathbf{B} \in \mathbb{R}^s \times \mathbb{R}^m$: Input weight matrix

$\mathbf{b}_x \in \mathbb{R}^s$: Bias term

$\mathbf{C} \in \mathbb{R}^n \times \mathbb{R}^s$: State to output weight matrix

$\mathbf{b}_y \in \mathbb{R}^n$: Output bias

Back Propagation Through Time



One data sample:

- Input: $\mathbf{U} = \begin{bmatrix} \mathbf{u}(k_0+1) & \mathbf{u}(k_0+2) & \dots & \mathbf{u}(k_0+T) \end{bmatrix}$.
- output: $\mathbf{Y}_t = \begin{bmatrix} \mathbf{y}_t(k_0+1) & \mathbf{y}_t(k_0+2) & \dots & \mathbf{y}_t(k_0+T) \end{bmatrix}$.
- SSE cost (per sample):

$$L = 0.5 \sum_{k=1}^{T} \mathbf{e}(k_0+k)^\top \mathbf{e}(k_0+k) = 0.5 \sum_{k=1}^{T} \sum_{i=1}^{n} \left(y_i(k_0+k) - y_{t,i}(k_0+k) \right)^2$$

- Batch cost (batch size = D):
$L = 0.5 \sum_{d=1}^{D} \sum_{k=1}^{T} \mathbf{e}_d(k_0+k)^\top \mathbf{e}_d(k_0+k)$

Gradients

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

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

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

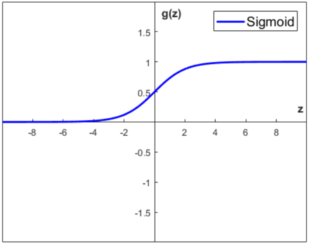
$$\frac{\partial \mathbf{x}(k)}{\partial a_{ij}} = \frac{\partial \mathbf{v}(k)}{\partial a_{ij}}\mathbf{f}'(\mathbf{v}(k)), \quad \mathbf{v}(k) = \mathbf{Ax}(k-1)+\mathbf{Bu}(k)+\mathbf{b}_x$$

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

Section 2

RNN Architectures for Learning Long Term Dependencies

Gated Architectures



$$\mathbf{g}(\mathbf{x}) = \mathbf{x} \odot \sigma(\mathbf{A}\mathbf{x} + \mathbf{b})$$

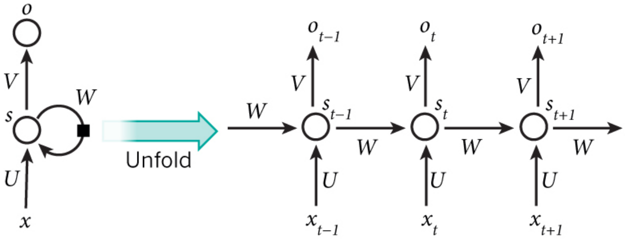
$\odot$: element-wise multiplicaiton

$\mathbf{x} \in \mathbb{R}^n$

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

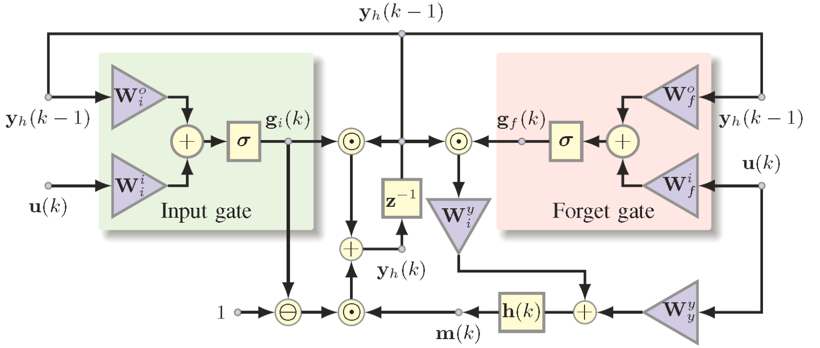
$\mathbf{A} \in \mathbb{R}^n \times \mathbb{R}^n$

Preserve Information with Gated Architectures



- 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



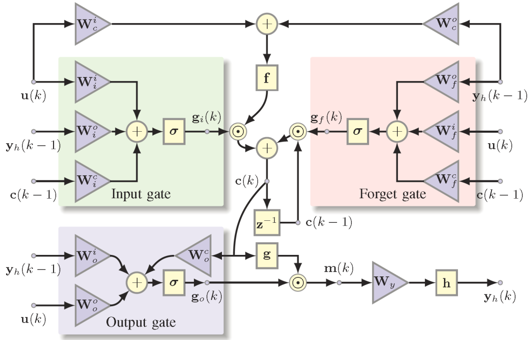
$$\mathbf{g}_f(k) = \boldsymbol{\sigma}\left(\mathbf{W}_f^i\mathbf{u}(k) + \mathbf{W}_f^o\mathbf{y_h}(k-1)\right)$$

$$\mathbf{g}_i(k) = \boldsymbol{\sigma}\left(\mathbf{W}_i^i\mathbf{u}(k) + \mathbf{W}_i^o\mathbf{y_h}(k-1)\right)$$

$$\mathbf{m}(k) = \mathbf{h}\left(\mathbf{W}_y^i\mathbf{u}(k) + \mathbf{W}_y^y(\mathbf{g}_f \odot \mathbf{y}_h(k-1))\right)$$

$$\mathbf{y}_h(k) = \mathbf{g}_i(k) \odot \mathbf{y_h}(k-1) + (1 \ominus \mathbf{g}_i(k)) \odot \mathbf{m}(k)$$

Long Short Term Memory Cell



$$
\begin{aligned}
\mathbf{g}_i(k) =& \boldsymbol{\sigma}\left(\mathbf{W}_i^i\mathbf{u}(k) + \mathbf{W}_i^o\mathbf{y_h}(k-1) + \mathbf{W}_i^c\mathbf{c}(k-1) + \mathbf{b}_i\right)\\
\mathbf{g}_f(k) =& \boldsymbol{\sigma}\left(\mathbf{W}_f^i\mathbf{u}(k) + \mathbf{W}_f^o\mathbf{y_h}(k-1) + \mathbf{W}_f^c\mathbf{c}(k-1) + \mathbf{b}_f\right)\\
\mathbf{g}_o(k) =& \boldsymbol{\sigma}\left(\mathbf{W}_o^i\mathbf{u}(k) + \mathbf{W}_o^o\mathbf{y_h}(k-1) + \mathbf{W}_o^c\mathbf{c}(k-1) + \mathbf{b}_o\right)\\
\mathbf{c}(k) =& \mathbf{g}_i(k) \odot \mathbf{f}\left(\mathbf{W}_c^i\mathbf{u}(k) + \mathbf{W}_c^o\mathbf{y_h}(k-1)\right) + \mathbf{g}_f(k) \odot \mathbf{c}(k-1)\\
\mathbf{m}(k) =& \mathbf{c}(k) \odot \mathbf{g}_o(k)\\
\mathbf{y_h}(k) =& \mathbf{h}(\mathbf{W}_y\mathbf{m}(k-1) + \mathbf{b}_y).
\end{aligned}
$$

Learning Long-Term Dependencies

- One way to avoid gradient *exploding* is to **clip** the gradient:

$$\text{if } ||\mathbf{g}|| > v, \mathbf{g} \leftarrow \frac{\mathbf{g}v}{||\mathbf{g}||}$$

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

$$\Omega = \sum_k \left(\frac{||\nabla_{\mathbf{x}(k)} L \frac{\partial \mathbf{x}(k)}{\partial \mathbf{x}(k-1)}||}{||\nabla_{\mathbf{x}(k)} L||} - 1 \right)^2$$

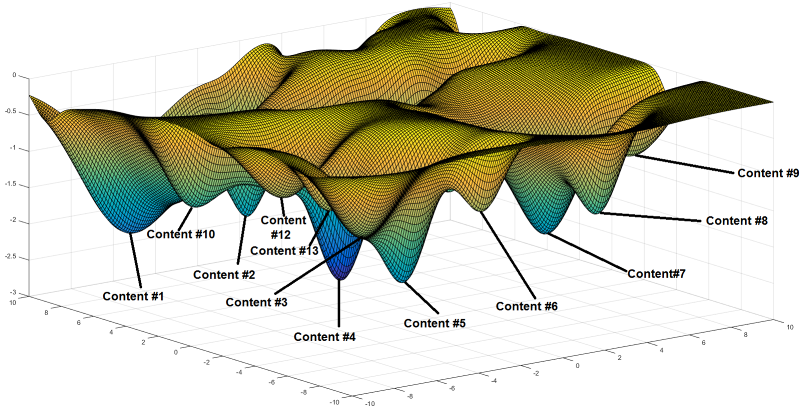
Section 3

Other RNN Architectures

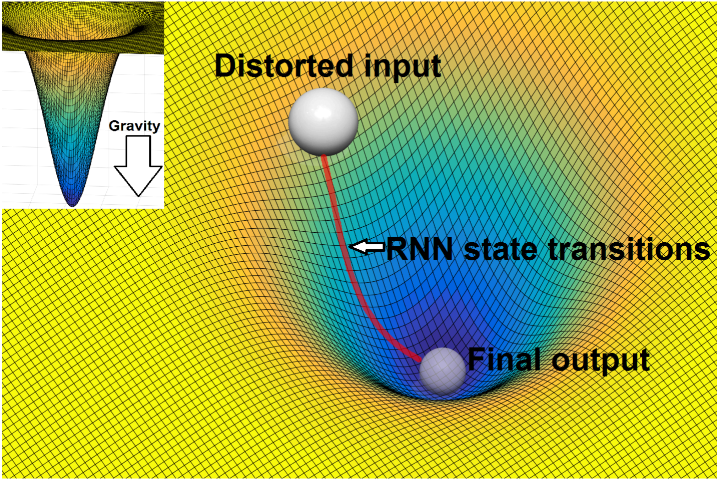
RNNs as Associative Memories

- 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)

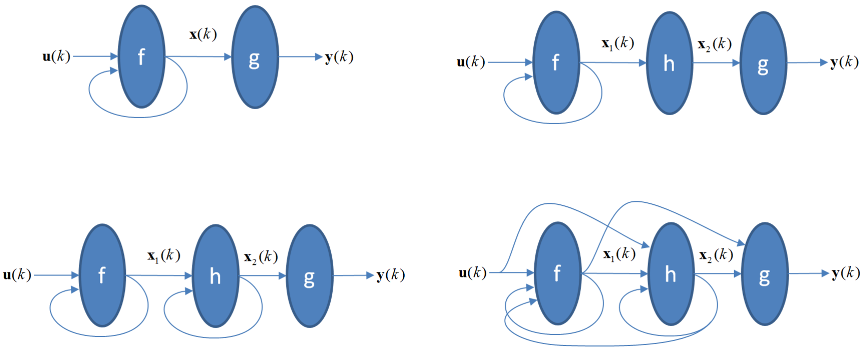
RNNs as Associative Memories



RNNs as Associative Memories



Deep RNNs



Deep RNNs

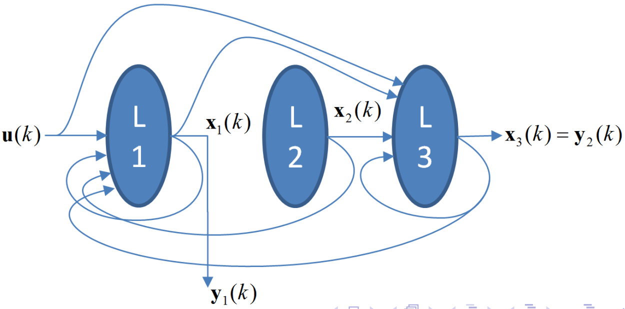
We can generalize this idea and create a *connection matrix*:

$$C = \begin{array}{cl} \begin{array}{cccc} \text{To layer 1} & \text{To layer 2} & \text{To layer 3} & \text{To output} \end{array} & \\ \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix} & \begin{array}{l} \text{From input} \\ \text{From layer 1} \\ \text{From layer 2} \\ \text{From layer 3} \end{array} \end{array}$$

Deep RNNs

Example:

$$C = \begin{array}{cc} \begin{array}{cccc} \text{(to L1)} & \text{(to L2)} & \text{(to L3)} & \text{(to output)} \end{array} & \\ \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} & \begin{array}{c} \text{(from input)} \\ \text{(from L1)} \\ \text{(from L2)} \\ \text{(from L3)} \end{array} \end{array}$$



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)

Reservoir Computing

- 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.

$$\mathbf{J}(k) = \frac{\partial \mathbf{x}(k)}{\partial \mathbf{x}(k-1)}$$

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

Section 4

System Identification with RNNs

Reconstructing the System States

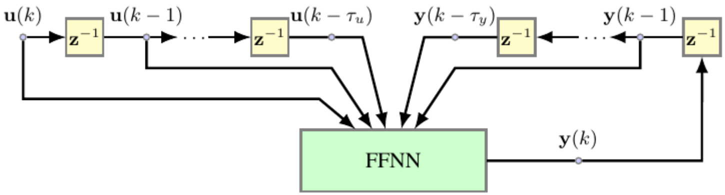
- We have a set of observations, i.e., measurements of a dynamic system input and output (states).
- We want to learn the system dynamics
- RNNs are universal approximators for dynamic systems

  (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

$$\mathbf{y}(k) = \mathbf{F}(\mathbf{u}(k), \mathbf{u}(k-1), \ldots, \mathbf{u}(k-d_x), \mathbf{y}(k-1), \ldots, \mathbf{y}(k-d_y)).$$

- 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.

$$\mathbf{y}(k) = \mathbf{F}(\mathbf{u}(k), \mathbf{u}(k-1), \ldots, \mathbf{u}(k-d_x), \mathbf{y}_t(k-1), \ldots, \mathbf{y}_t(k-d_y)).$$

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