ONLY GOD

2022

mansoori@shirazu.ac.ir

Neural Network & Deep Learning

Recurrent NN

CSE & IT Department

School of ECE

Shiraz University

mansoori@shirazu.ac.ir

RNN with Hidden Layer

Sequential Data

mansoori@shirazu.ac.ir

- When reading text of a book, ideas you form and train of thoughts depend on what you have understood and retained up to a given point in book
- This persistence or ability to have some memory and pay attention helps to develop understanding of concepts, to think, to use knowledge, to solve problems and to innovate
- There is an inherent notion of progress with steps, with passage of time, for sequential data (datastream)
- Sequential memory is a mechanism that makes it easier for brain to recognize sequence patterns

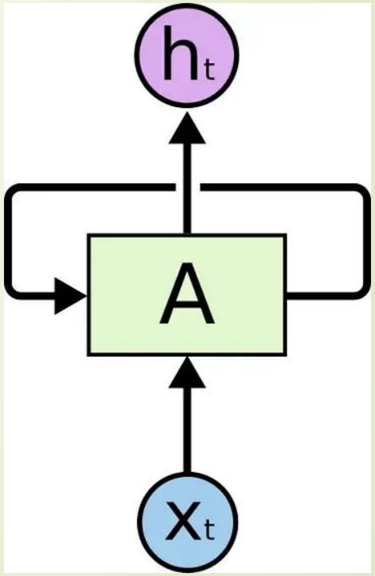
Why Recurrent NNs (RNNs)?

mansoori@shirazu.ac.ir

- How would a feed-forward NN (FFNN) read a sentence and pass on previously gathered information in order to completely understand and relate sequence of incoming data?

- FFNNs, despite how good being, lack this intuitive innate tendency for persistence, as their training for weights is not same as persistence of information for next step

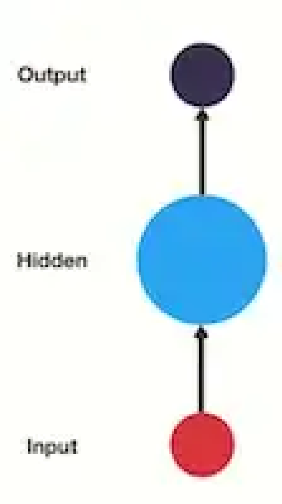
- RNNs address this drawback with a simple yet elegant mechanism and are great at modeling sequential data



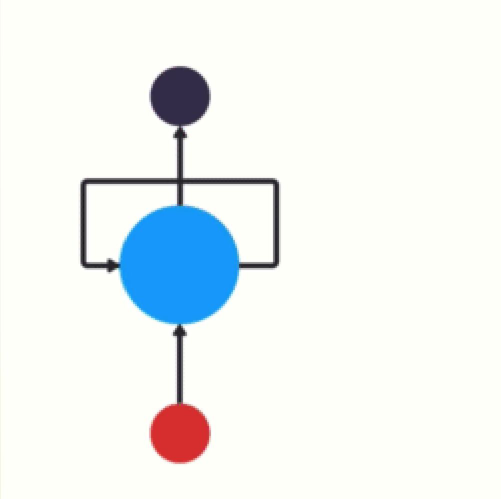
RNN vs FFNN

mansoori@shirazu.ac.ir

- A FFNN has input coming in, to hidden layer, which results in output



- An RNN feeds its output to itself at next time-step, forming a loop, passing down much needed information



RNN Features

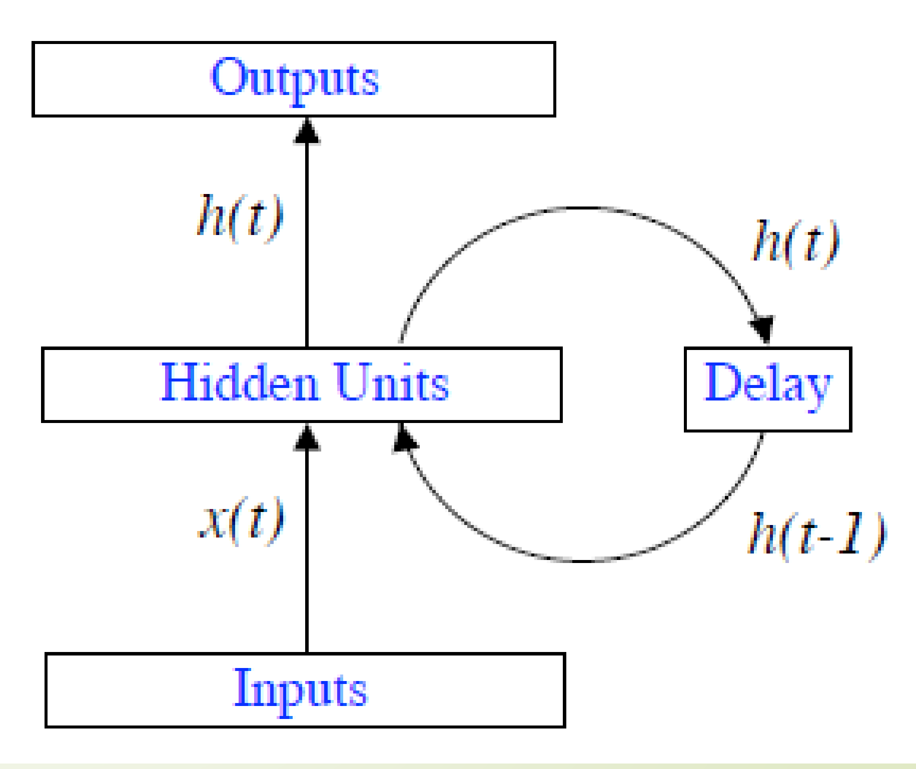
mansoori@shirazu.ac.ir

- Biological nervous systems show high levels of recurrency
- Hidden units have task of mapping both an external input and previous internal state to some desired output
- Can maintain a state vector (memory) that contains information about history of all past elements of sequence
- RNNs can do temporal processing and learn sequential data
  - speech recognition, handwritten recognition, language modeling, translation, image captioning

RNN Architectures

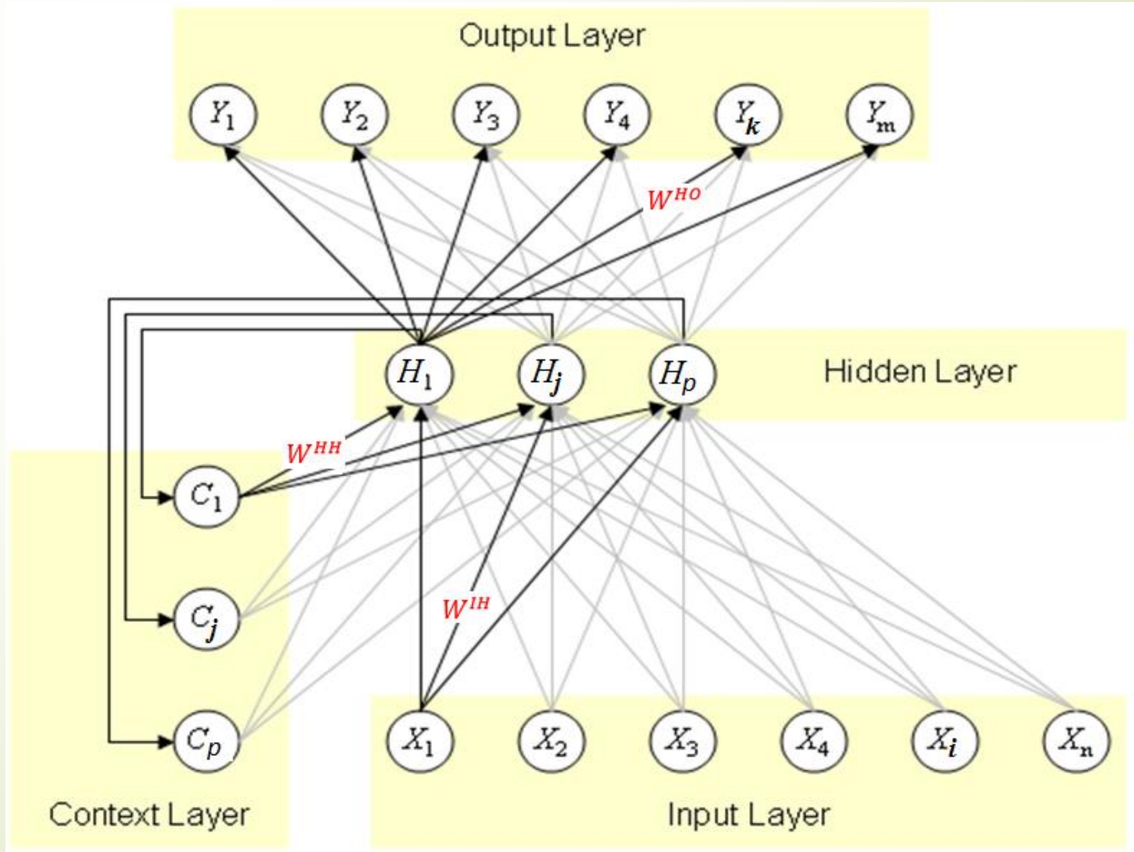
mansoori@shirazu.ac.ir

- A FFNN + added loops
- Nonlinear mapping capabilities of FFNN + some form of memory
- A fully RNN: a FFNN with hidden unit activations feeding back into net along with inputs
- A uniform structure (every neuron connected to all others + stochastic activation functions)



Elman Network

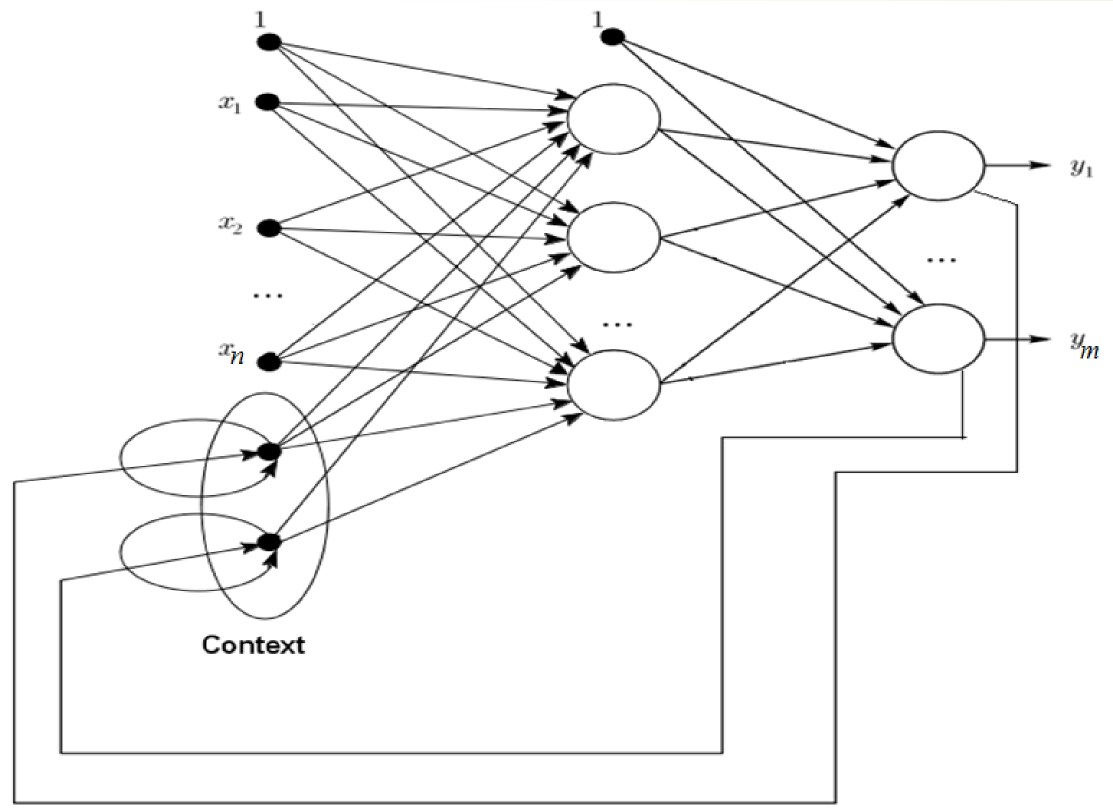
- A layer of context units (as internal states) + a FFNN
- States of hidden units could be fed-back into hidden units during next stage of input
- Designed to learn sequential or time-varying patterns
- Can recognize and predict learned series of values or events



Jordan Network

mansoori@shirazu.ac.ir

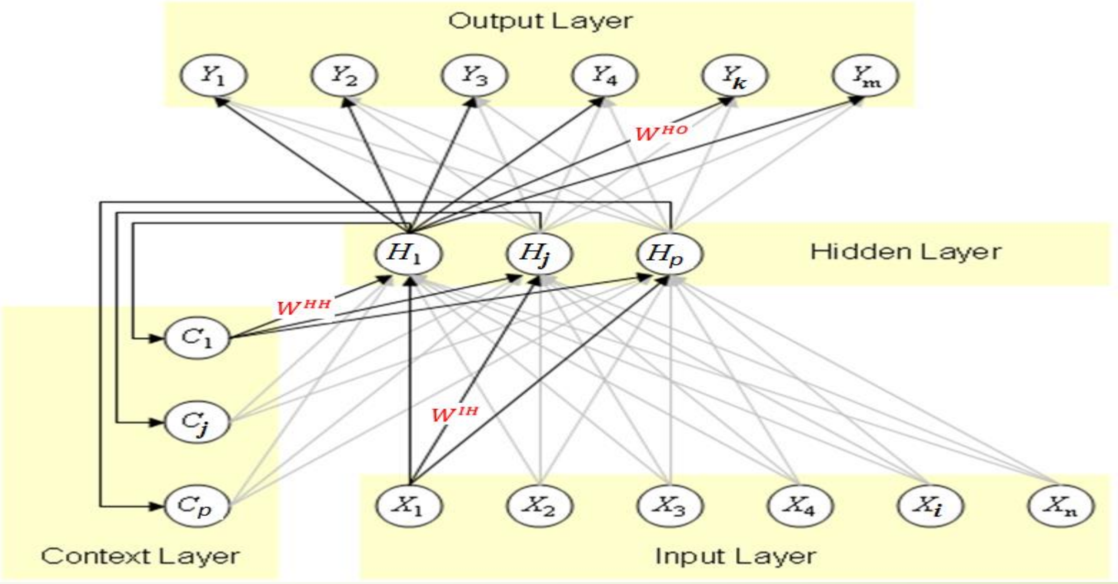
- Has connections that feed-back from output to input layer
- Also, some input layer units feed-back to themselves
- Has a form of short-term memory (STM)
- Useful for tasks that are dependent on sequence of successive states
- Can be trained by back-propagation



Forward Pass of RNN

- Activations at hidden layer arrive from both current external input and hidden layer activations from previous time step:
  $h\_in_J(t) = \sum_{i=1}^{n} w_{iJ}^{\mathrm{IH}}\, x_i(t) + \sum_{j=1}^{p} w_{jJ}^{\mathrm{HH}}\, h_j(t-1)$

- Initial values of $h_j(0)$ are set to zero



- Nonlinear and differentiable activation functions , $f^{\mathrm{H}}(.)$, are then applied: $h_J(t) = f^{\mathrm{H}}(h\_in_J(t))$
- So, $\vec{h}(t) = f^{\mathrm{H}}(W^{\mathrm{IH}}\, \vec{x}(t) + W^{\mathrm{HH}}\, \vec{h}(t-1))$

Forward Pass of RNN

mansoori@shirazu.ac.ir

- Complete sequence of hidden activations can be calculated by starting at $t = 1$ and incrementing t at each step

- Inputs to output units can be calculated at the same time as hidden activations: $y\_in_K(t) = \sum_{j=1}^{p} w_{jK}^{\mathrm{HO}}\, h_j(t)$

- Using output activation function, $f^{\mathrm{O}}(.)$, output of net is computed: $y_K(t) = f^{\mathrm{O}}(y\_in_K(t))$

- So, $\vec{y}(t) = f^{\mathrm{O}}(W^{\mathrm{HO}}\, \vec{h}(t))$

- For sequence classification tasks, $f^{\mathrm{O}}(.)$ uses sigmoid for two-class, and softmax for multi-class problems

Universal Approximation Theorem

m a n s o o r i @shirazu.ac.ir

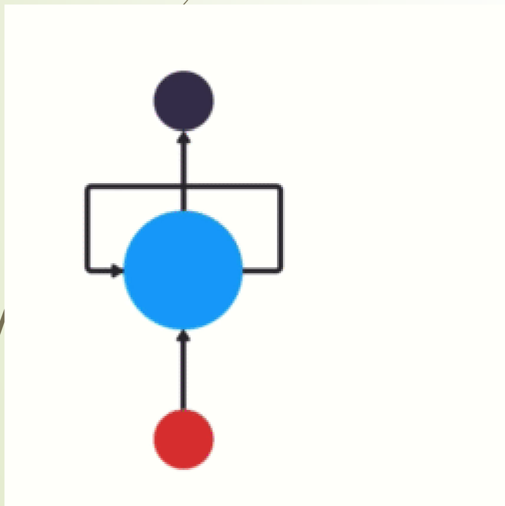
- Any nonlinear dynamical system can be approximated to any accuracy by an RNN, provided that network has enough sigmoidal hidden units

- How to approximate? Learning from a set of training data

Unfolding over Time

mansoori@shirazu.ac.ir

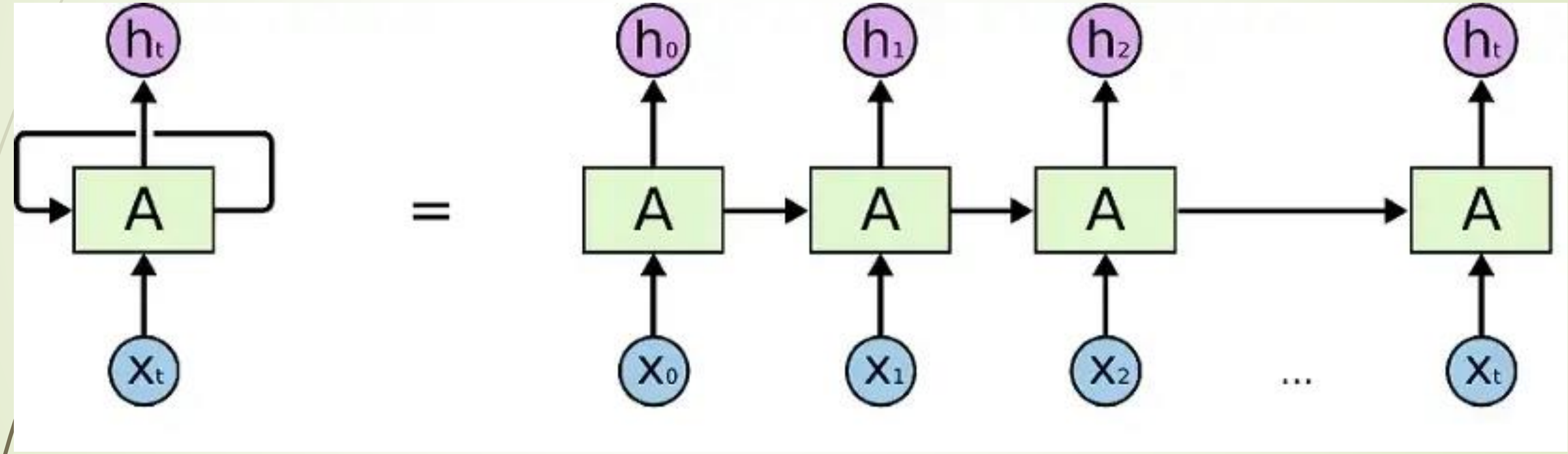
- RNN encodes incoming sequential data first before being utilized to determine the intent/action via another FFNN for decision



Unfolding over Time

mansoori@shirazu.ac.ir

- To better understand flow, in unrolled version of RNN, each RNN has different input (of sequence) and output at each time-step

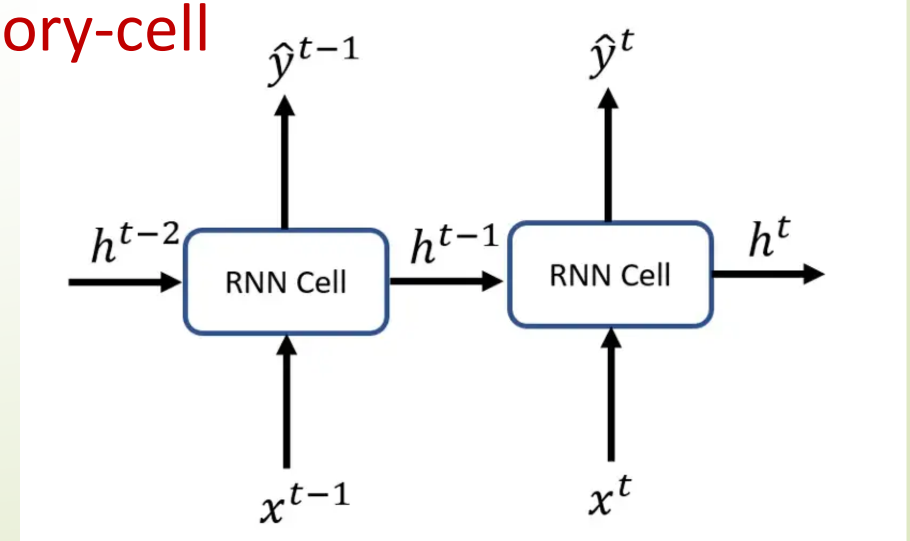


- NN A takes in input at each time step while giving output h and passing information to itself for next incoming input t+1 step

Memory in RNN

mansoori@shirazu.ac.ir

- Humans tend to retrieve information from memory, short or long, use current information and derive next action
- As output of a recurrent neuron in RNN, at time step t, is function of previous input with accumulated information till time step t-1, this mechanism is a form of memory
- Any part of NN which has notion of preserving state across time steps is a memory-cell



mansoori@shirazu.ac.ir

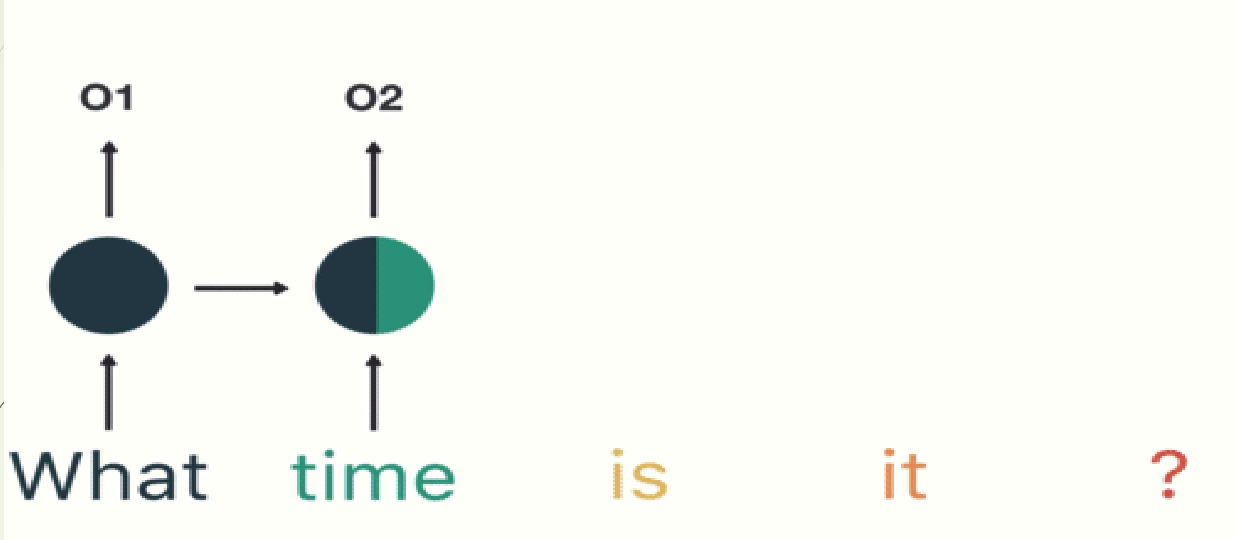
Decision in RNN

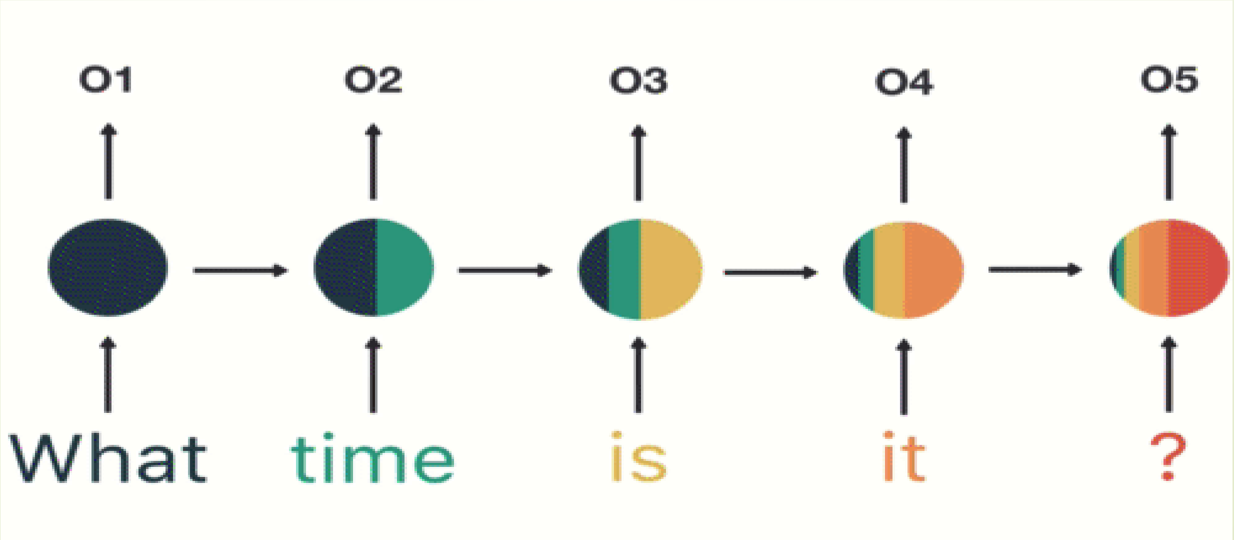
- How RNN analyzes sentence “what time is it?”
- we try to break the sequence and color code it

What time is it?

Decision in RNN

mansoori@shirazu.ac.ir

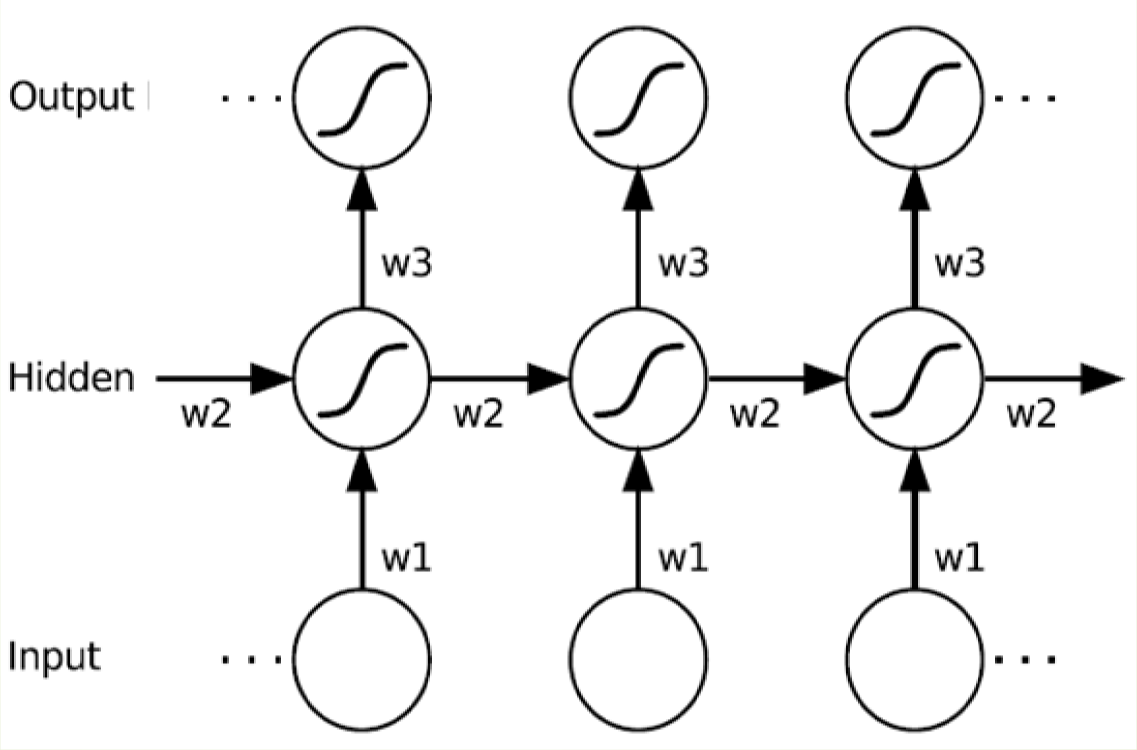




Unfolding over Time

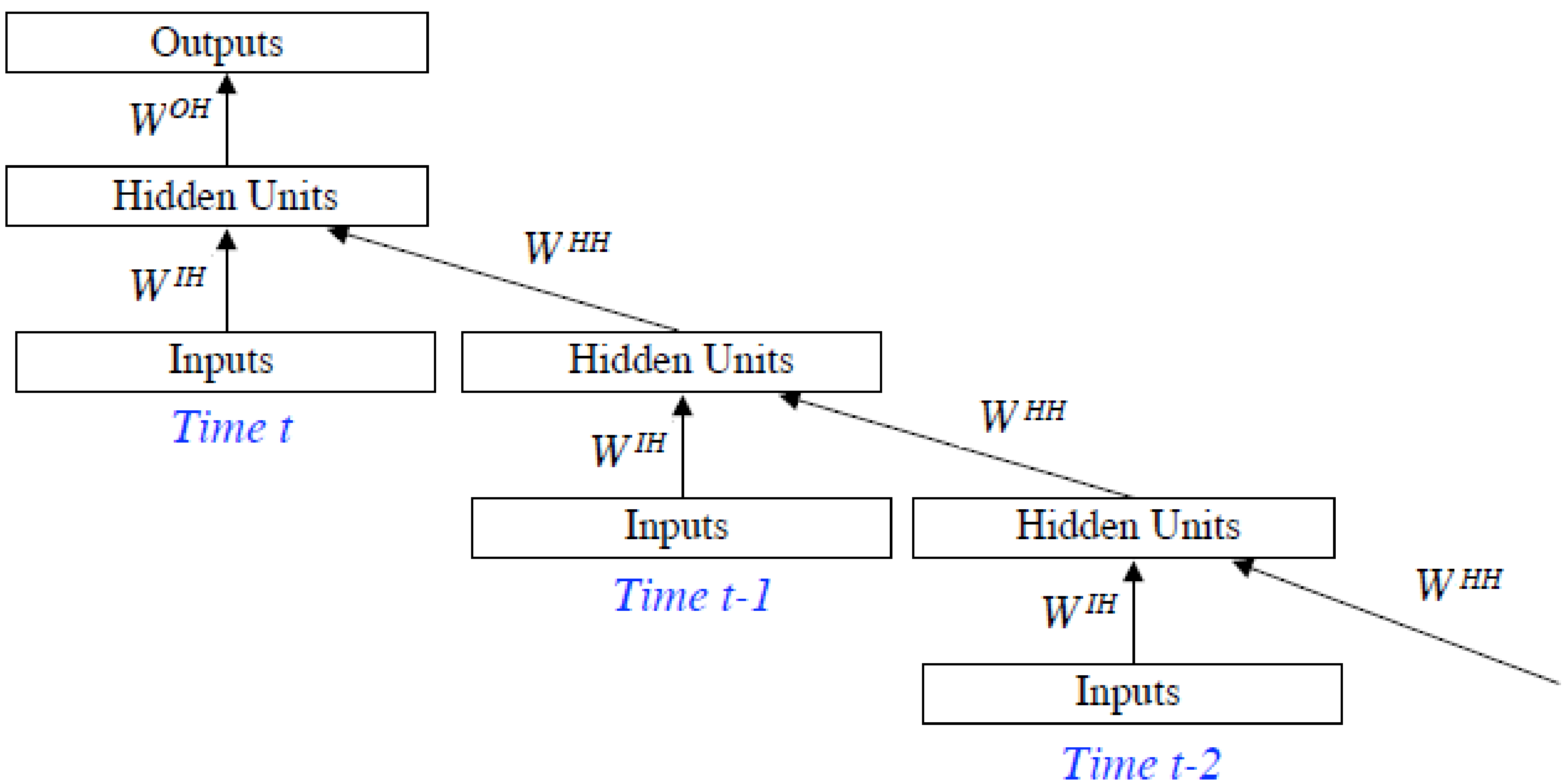
mansoori@shirazu.ac.ir

- An RNN can be converted into a FFNN by unfolding over time



Unfolding over Time

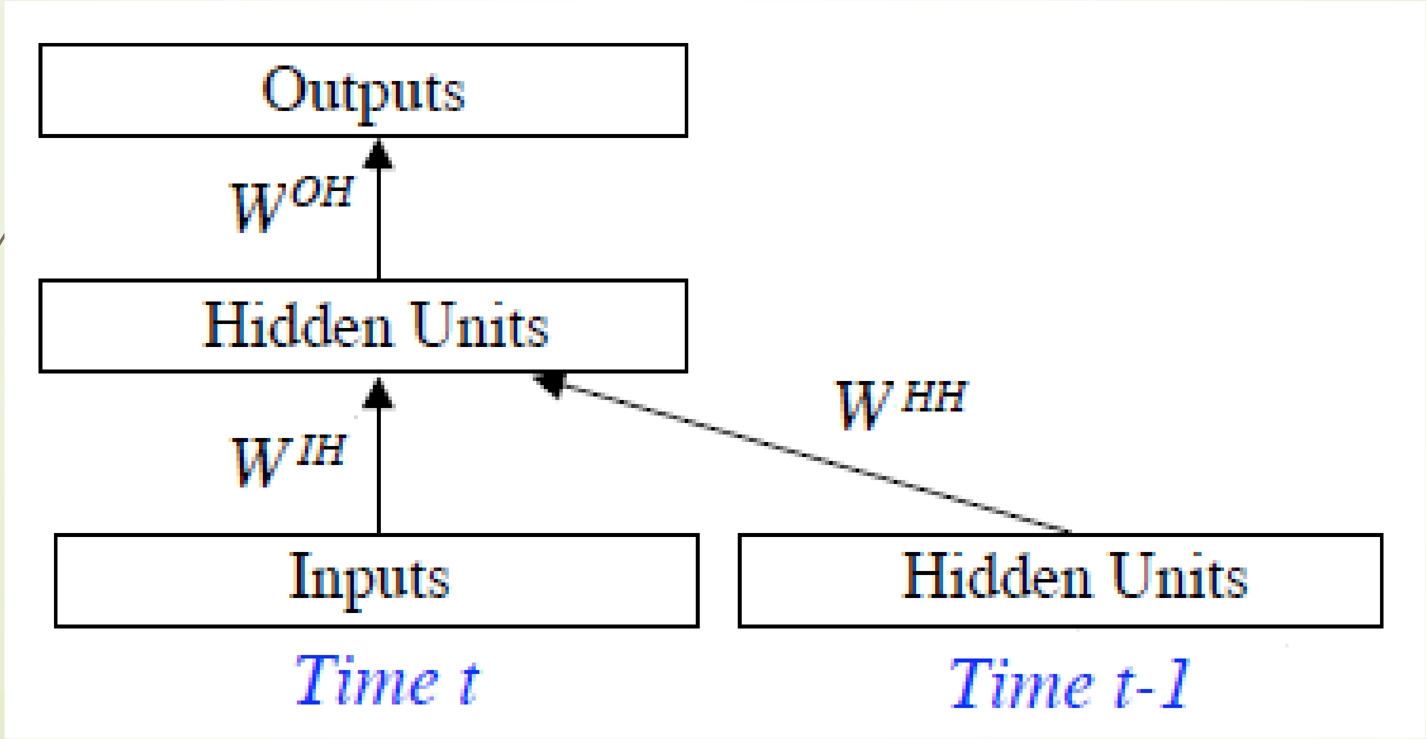
- So, all earlier theory about FFNN learning follows through



Unfolded Elman Network

mansoori@shirazu.ac.ir

Truncating unfolded network to just one time step reduces it to a simple RNN as Elman network



RNN Learning

mansoori@shirazu.ac.ir

- For simple architectures and deterministic activation functions, learning can be achieved using similar gradient descent procedures to those leading to back-propagation algorithm
- For stochastic activations functions, simulated annealing approaches may be more appropriate

1. Continuous training: network state is never reset during training
2. Epochwise training: network state reset at each epoch

Back-propagation Through Time

manoori@shirazu.ac.ir

- BPTT learning algorithm is a natural extension of standard back-propagation that performs gradient descent on a complete unfolded network

- If training sequence starts at time t_s and ends at time t_f, total cost function is sum over time of error, $E_{\text{ins}}(.)$, at each time-step while reusing same weights:

$$E_{\text{tot}}(t_s, t_f) = \sum_{t=t_s}^{t_f} E_{\text{ins}}(t)$$

- Gradient descent weight updates have contributions from each time-step:

$$\Delta w_{ij} = -\eta \frac{\partial E_{\text{tot}}(t_s, t_f)}{\partial w_{ij}} = -\eta \sum_{t=t_s}^{t_f} \frac{\partial E_{\text{ins}}(t)}{\partial w_{ij}} \quad \text{for } w_{ij} \in \{W^{\text{IH}}, W^{\text{HH}}\}$$

Backward Pass of BPTT

mansoori@shirazu.ac.ir

Like standard back-propagation, BPTT consists of a repeated application of chain rule (for $t = t_f, \dots, t_s$)

$$\Delta w_{jK}^{\mathrm{HO}}(t) = -\eta\, \delta_K^{\mathrm{O}}(t)\, h_j(t)\ ,$$

$$\delta_K^{\mathrm{O}}(t) = f^{\mathrm{O}'}\big(y\_in_K(t)\big)\{-(d_K - y_K(t))\}$$

$$\Delta w_{jJ}^{\mathrm{HH}}(t) = -\eta\, \delta_J^{\mathrm{H}}(t)\, h_j(t-1)\ ,$$

$$\delta_J^{\mathrm{H}}(t) = f^{\mathrm{H}'}(h\_in_J(t))\{\textstyle\sum_{k=1}^{m} \delta_k^{\mathrm{O}}(t) w_{Jk}^{\mathrm{HO}} + \sum_{j=1}^{p} \delta_j^{\mathrm{H}}(t+1) w_{Jj}^{\mathrm{HH}}\}$$

$$\Delta w_{iJ}^{\mathrm{IH}}(t) = -\eta\, \delta_J^{\mathrm{I}}(t)\, x_i(t)\ ,$$

$$\delta_J^{\mathrm{I}}(t) = f^{H'}(h\_in_J(t))\{\textstyle\sum_{k=1}^{m} \delta_k^{\mathrm{O}}(t) w_{Jk}^{\mathrm{HO}} + \sum_{j=1}^{p} \delta_j^{\mathrm{H}}(t+1) w_{Jj}^{\mathrm{HH}}\}$$

Practical Considerations for BPTT

mansoori@shirazu.ac.ir

- Unfolded network is quite complex because of need to keep track of all components at different time-steps
- Typically, weight updates are made in an online fashion (at each time-step)
- This requires history of inputs and past network states to be stored
- To be computationally feasible, truncation at a certain number of time-steps (e.g., ~30) is required
- Earlier information being ignored since contributions to weight updates are smaller

Practical Considerations for BPTT

mansoori@shirazu.ac.ir

- In a stable network, contributions to weight updates should become smaller the further back in time they come from
- This is because they depend on higher powers of small feedback strengths (corresponding to sigmoid derivatives multiplied by feedback weights)
- In Elman network, each set of weights appears only once, so standard back-propagation rather than full BPTT can be used
    - Error signal will not get propagated back very far

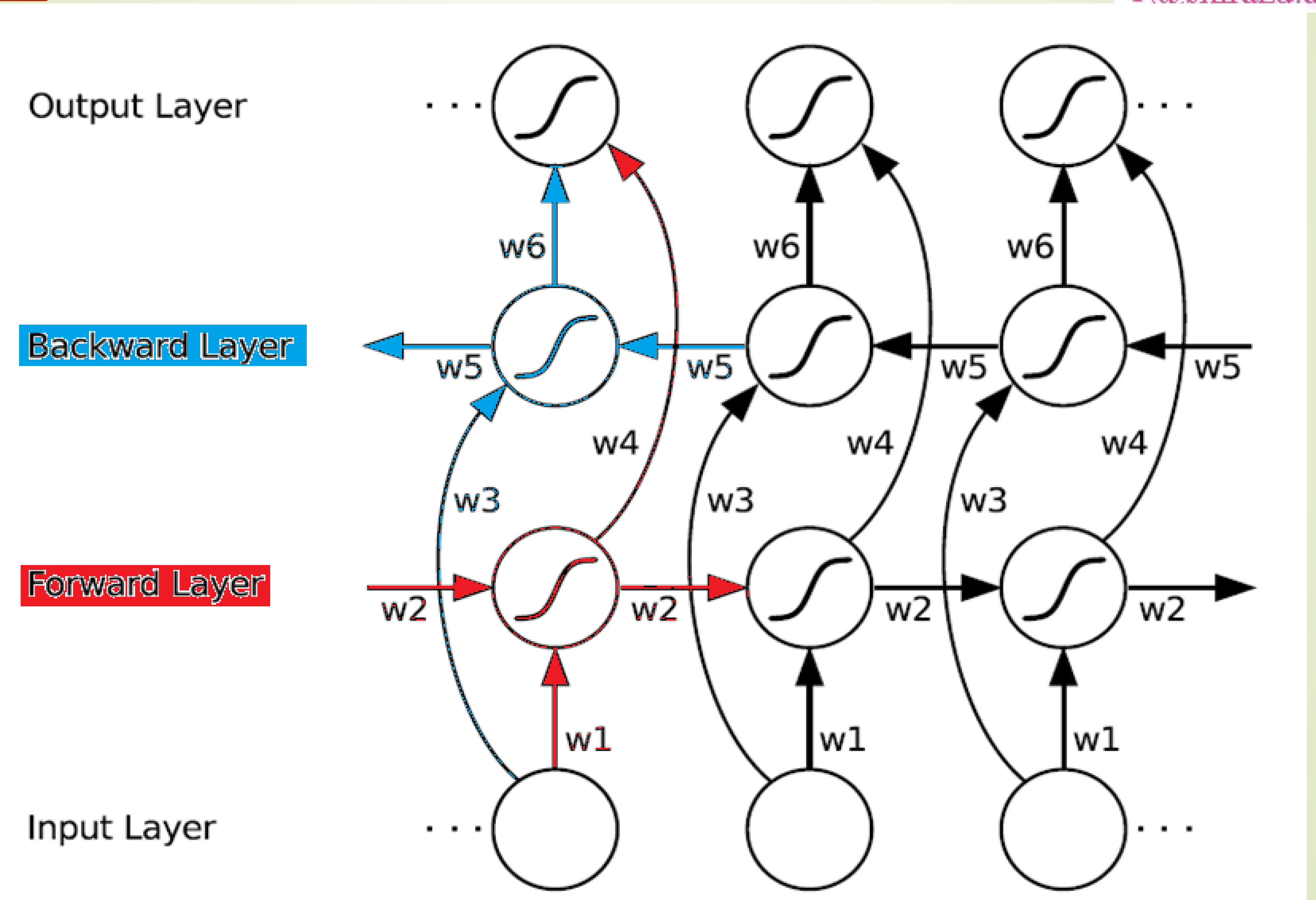
Bidirectional RNN (BRNN)

m a n s o o r i @shirazu.ac.ir

- It is beneficial to have access to future as well as past context
- Since standard RNNs process sequences in temporal order, they ignore future context
- Solution: present each training sequence forwards and backwards to two separate recurrent hidden layers, both of which are connected to the same output layer
- Provides output layer with complete past and future context for every point in input sequence

Unfolded BRNN

mansoori@shirazu.ac.ir



Forward Pass of BRNN

mansoori@shirazu.ac.ir

- Input sequence is presented in opposite directions to two hidden layers
- Output layer is not updated until both hidden layers have processed entire input sequence:

for $t = t_s$ to t_f **do**

Forward pass for forward hidden layer, storing activations at each time-step

for $t = t_f$ to t_s **do**

Forward pass for backward hidden layer, storing activations at each time-step

for all t, in any order, **do**

Forward pass for output layer, using stored activations from both hidden layers

Backward Pass of BRNN

mansoori@shirazu.ac.ir

- All output layer δ terms are calculated first, then fed back to two hidden layers in opposite directions

for all t, in any order, **do**

Backward pass for output layer, storing δ terms at each time-step

for $t = t_f$ to t_s **do**

BPTT backward pass for forward hidden layer, using stored δ terms from output layer

for $t = t_s$ to t_f **do**

BPTT backward pass for backward hidden layer, using stored δ terms from output layer