RECURRENT NEURAL NETWORKS (RNNs)

Neural networks Classification:

Static Networks : Output can be calculated directly from the input through feed-forward connections.

Dynamic Networks : Output depends not only on the current input to the network, but also on the current and/or previous inputs, outputs or states of the network.

***Dynamic Neural Networks* (DNNs)/RNNs have, in comparison to static neural networks, more complex architectures.**

What Are Recurrent Neural Networks?

✓Recurrent nets are a type of ANNs designed to recognize patterns in sequences of data, such as :

✓text, genomes, handwriting, the spoken word, or numerical times series data emanating from sensors, stock markets and government agencies.

✓These algorithms take time and sequence into account, have a **temporal dimension.**

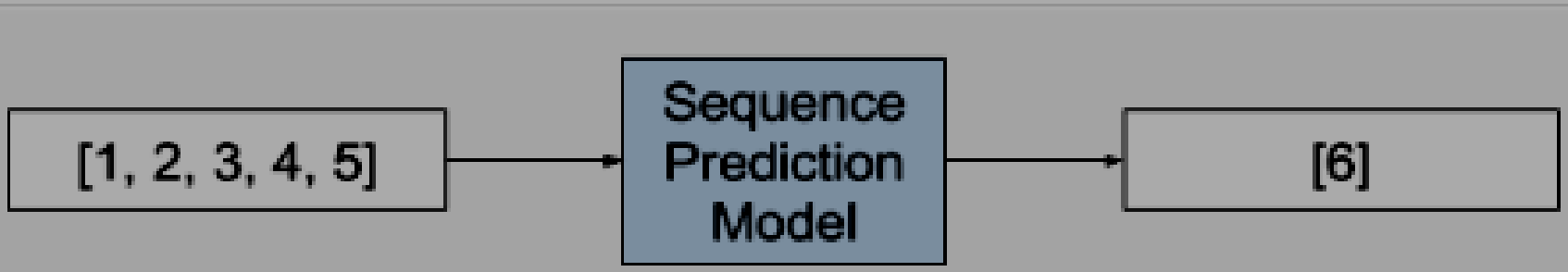
Applications of RNNs

- ✓Language translation and modeling
- ✓Handwriting recognition
- ✓Speech recognition, speech emotion recognition, speaker identification
- ✓Image captioning, image labeling, video description
- ✓Time series data such as stock prices to tell you when to buy or sell
- ✓Autonomous driving systems to anticipate car trajectories and help avoid accidents.

FOUR main types of sequence prediction problems:

1. Sequence Prediction

Given an input sequence, predict the next value in the sequence.



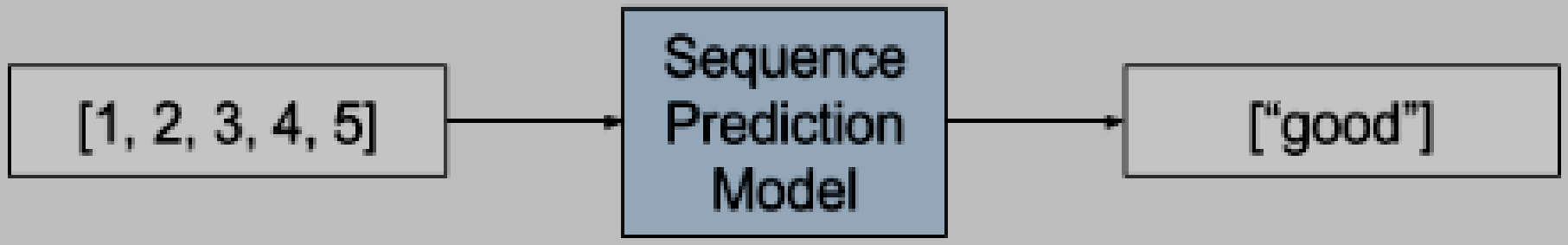
Example of a Sequence Prediction Problem

Examples include:

- Weather Forecasting
- Stock Market Prediction
- Product Recommendation

2. Sequence Classification

Given an input sequence, classify the sequence.



Example of a Sequence Classification Problem.

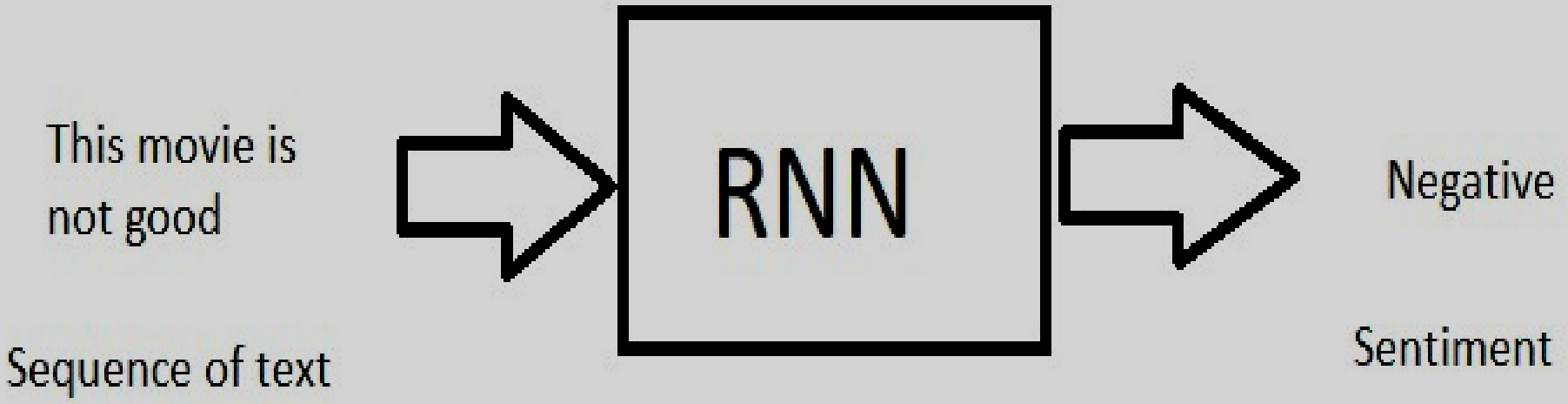
Examples include:

- DNA Sequence Classification
- Anomaly Detection
- Sentiment Analysis

2.Many to One

Falls under Many to One

RNN takes sequence of words as input and generates one output.



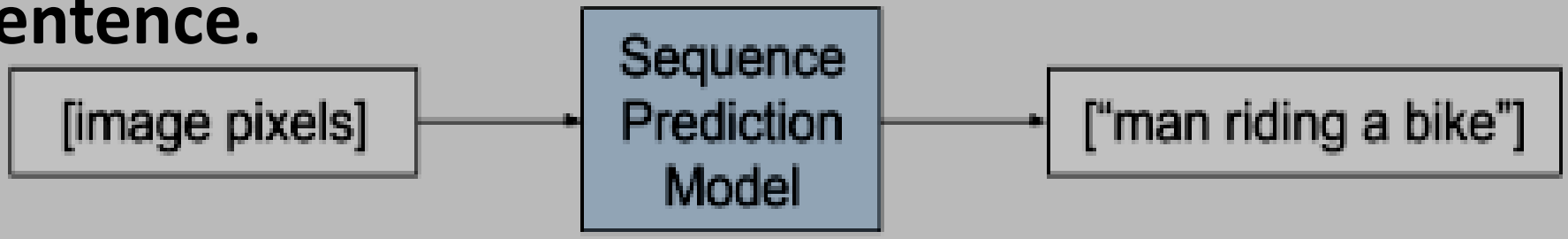
Many to one approach could be to

- ✓**handle a sequence of images (for example a video) and produce one word/sentence for it**

3. Sequence Generation

Given an observation, generate an output sequence.

Falls Under One to Many : label a image with a sentence.



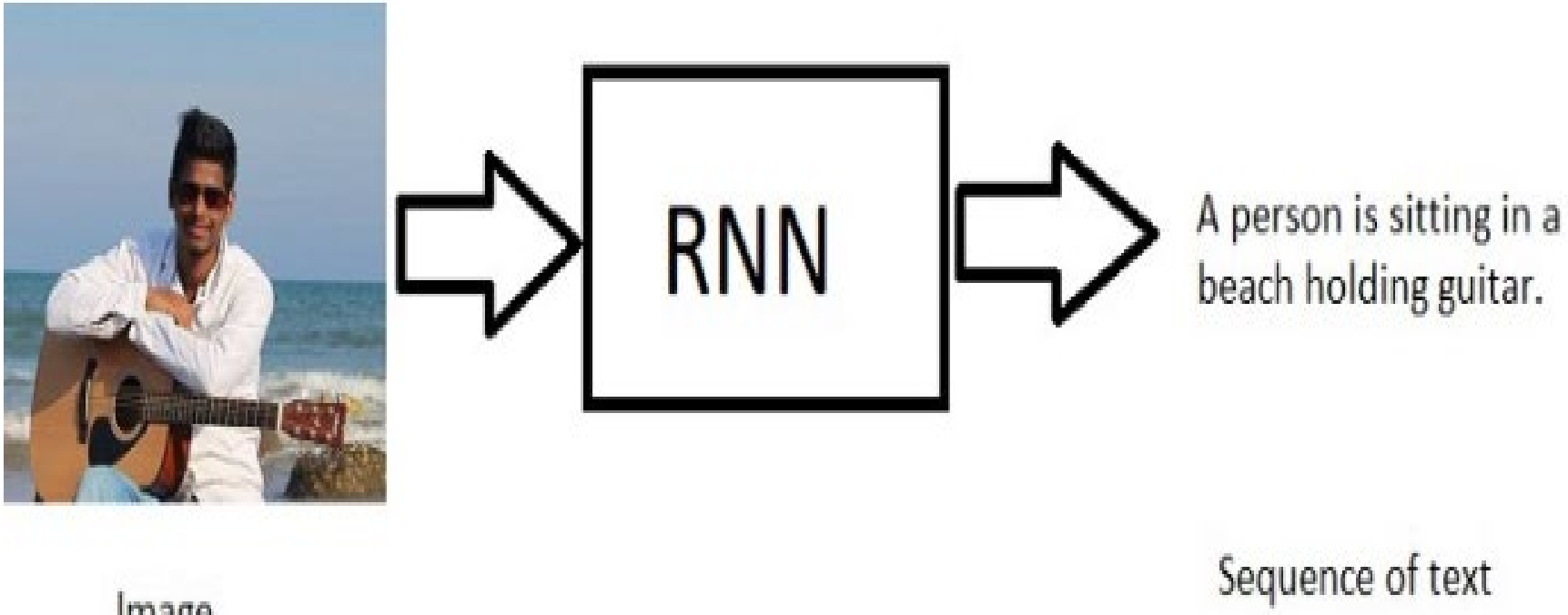
Example of a Sequence Generation Problem

Examples include:

- Text Generation
- Music Generation
- Image Captioning

1. One to Many

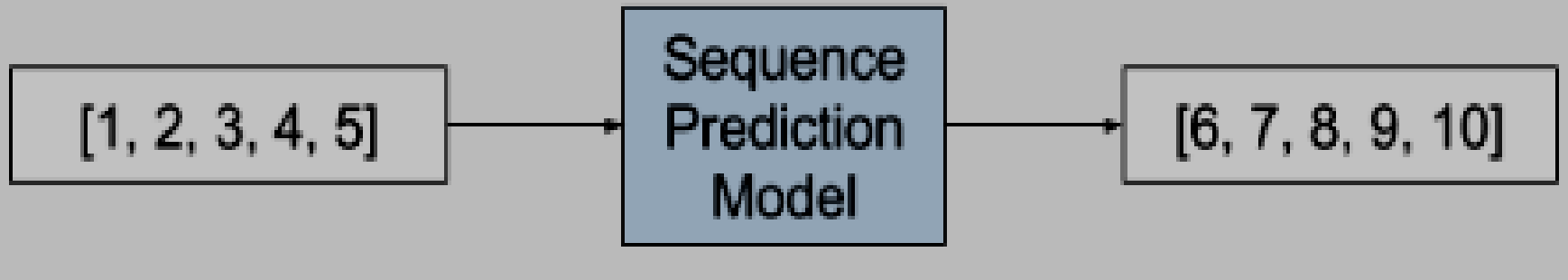
RNN takes one input lets say an image and generates a sequence of words.



4. Sequence-to-Sequence Prediction

Given an input sequence, generate an output sequence.

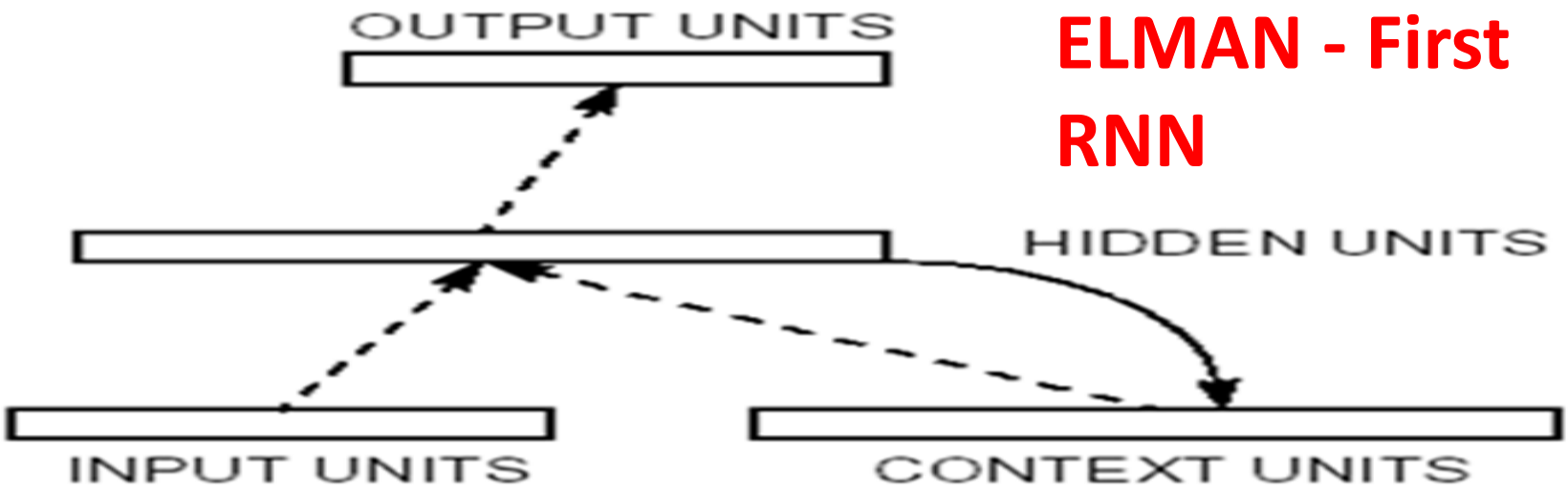
Falls under Many to Many



Example of a Sequence-to-Sequence Prediction Problem

Examples include:

- Multi-Step Time Series Forecasting
- Text Summarization
- Language Translation

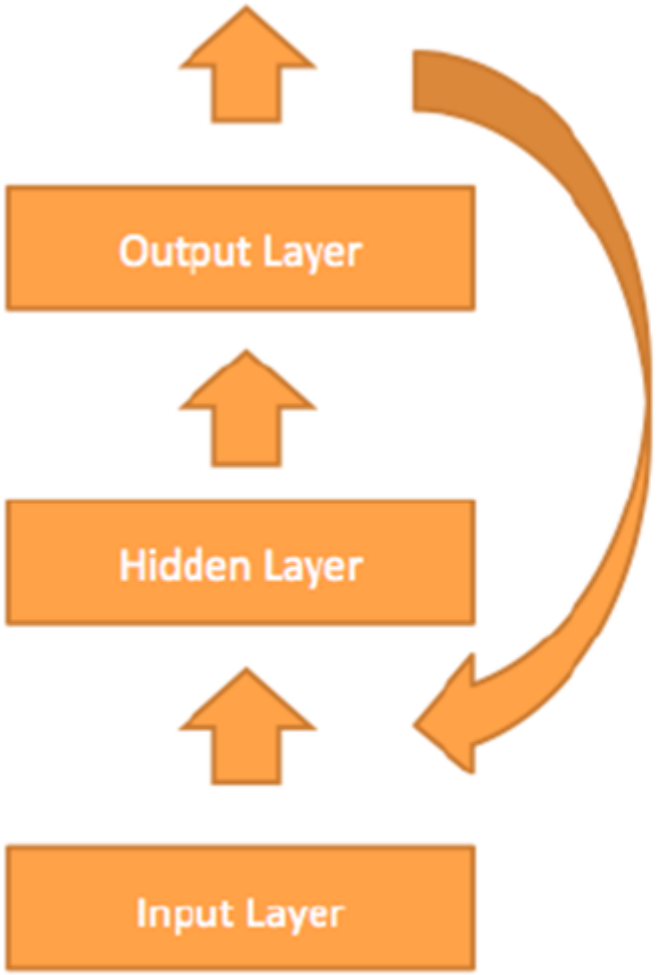


ELMAN - First RNN

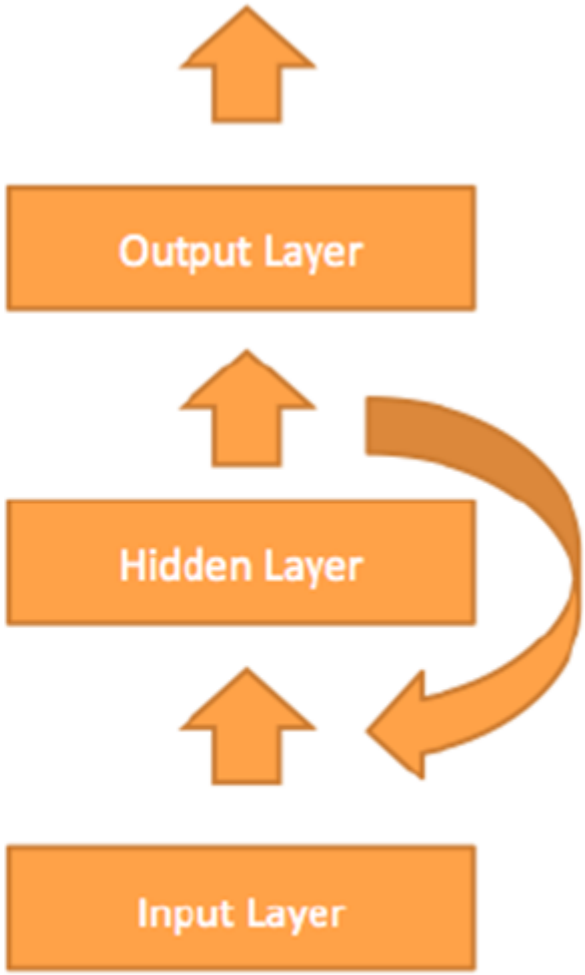
Activations are copied from hidden layer to context layer on a one-for-one basis, with FIXED WEIGHT OF 1
Dotted lines represt TRAINABLE connections

Context units remember the previous internal state.

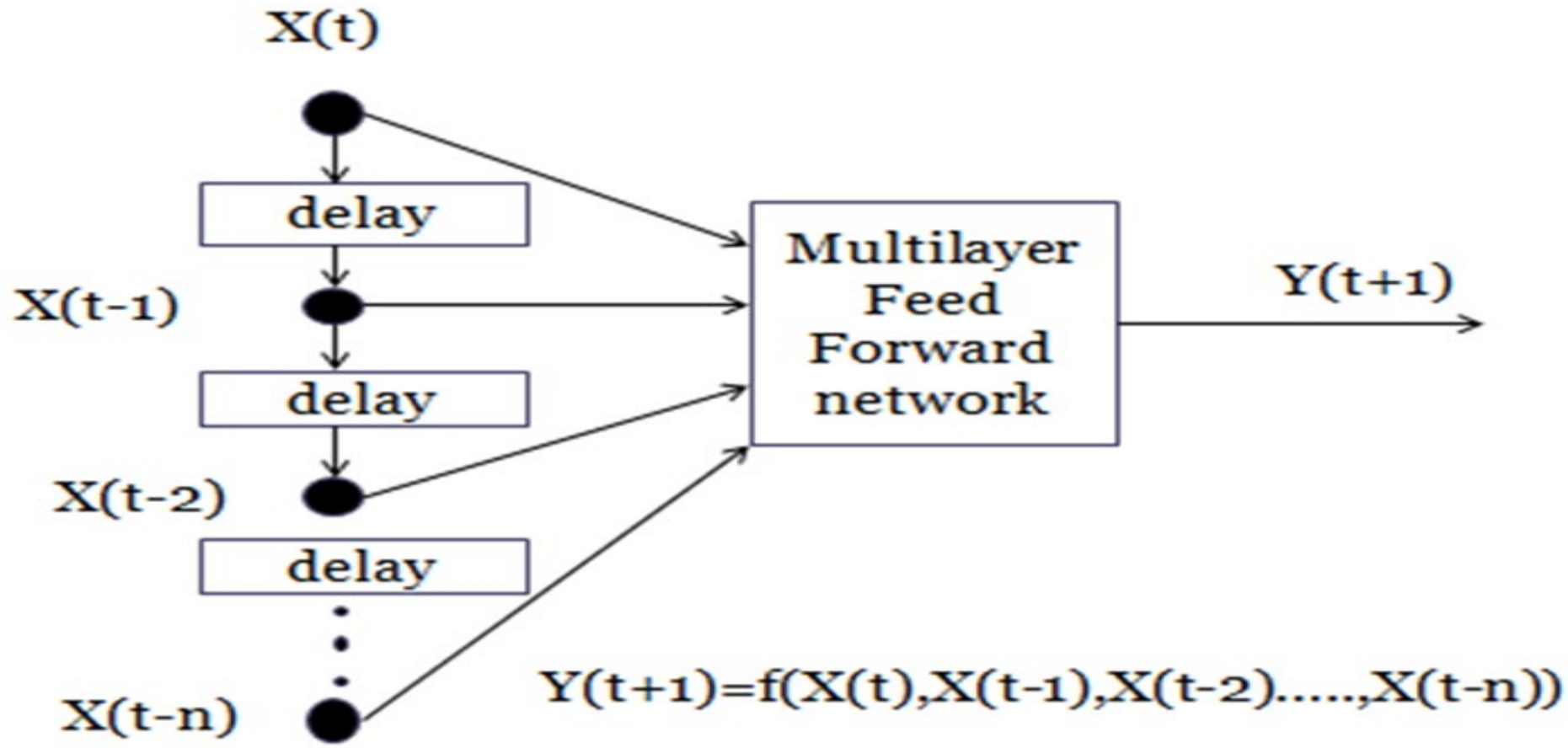
Hidden units have the task of mapping both an external input and also the previous internal state to some desired output



JORDAN network



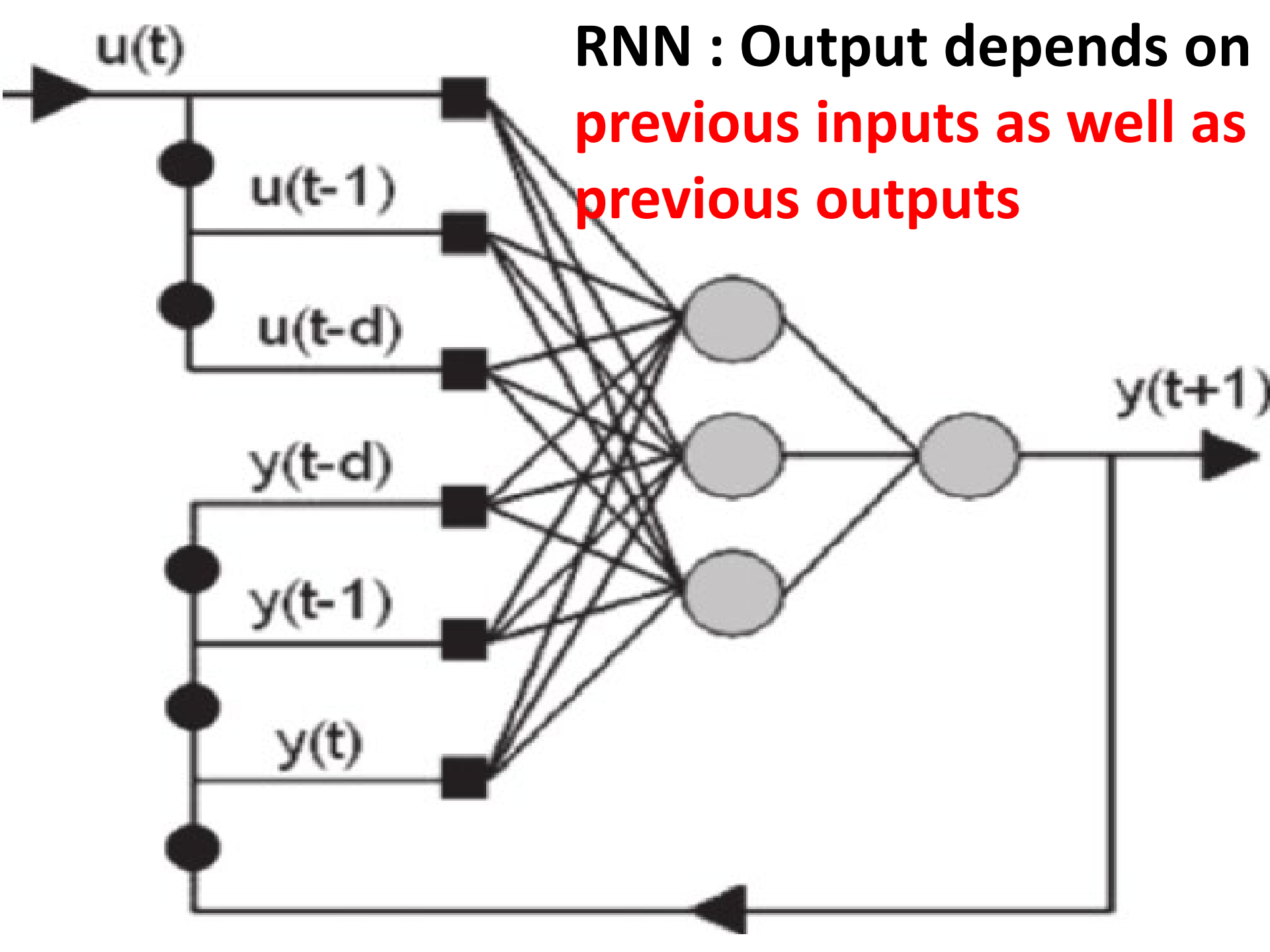
ELMAN Network

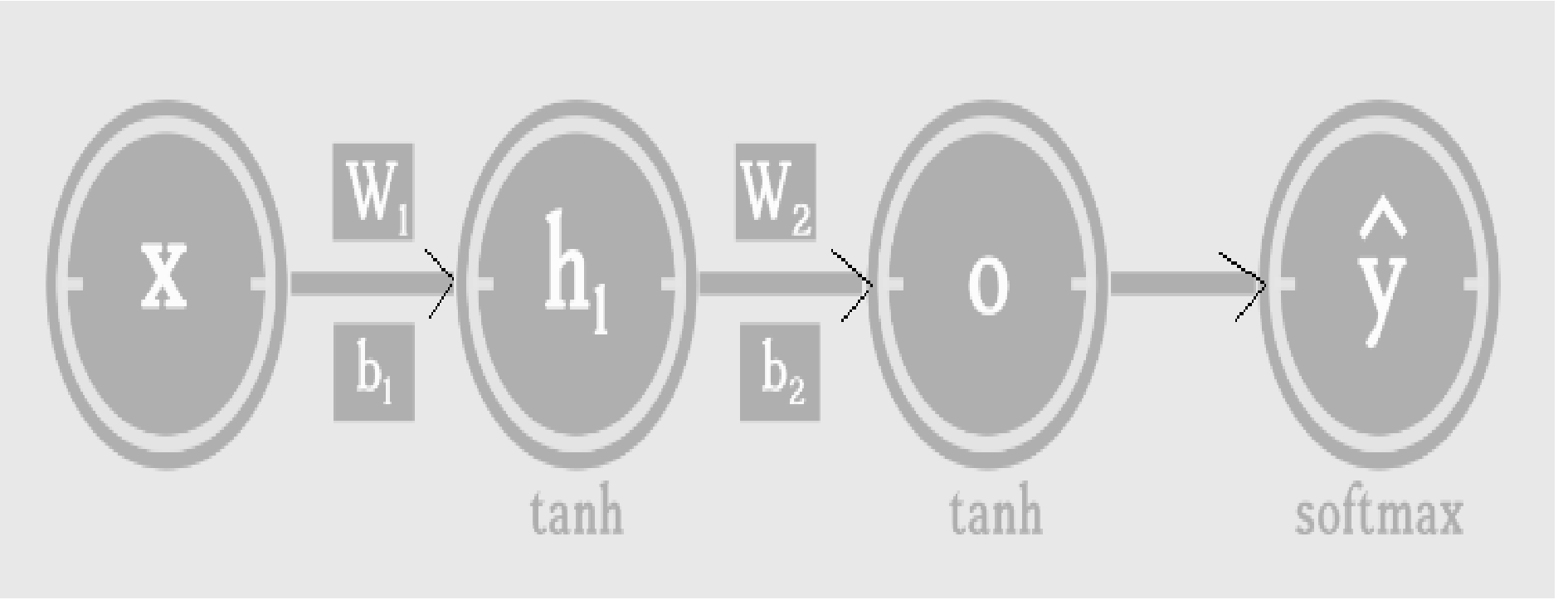


TDNN(Time Delay neural networks) - **particular case of feedforward neural network architecture to capture temporal dynamics**

The response at time t is based on the previous inputs .

RNN : Output depends on previous inputs as well as previous outputs

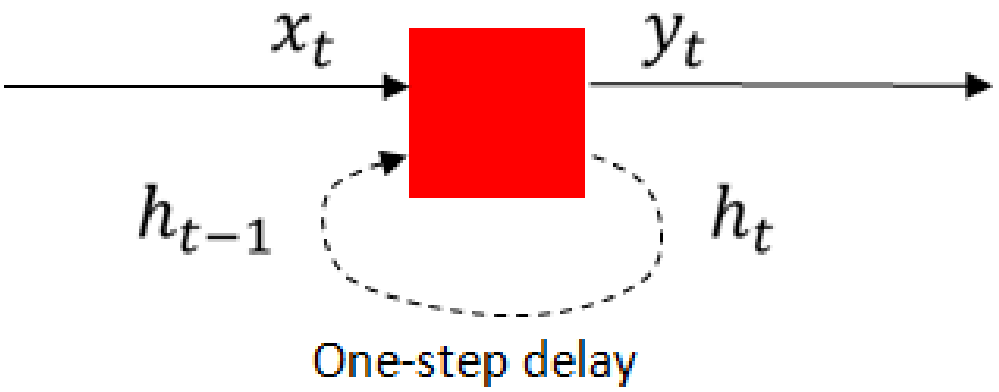




To understand the architecture and data flow, ANN can be expressed as a

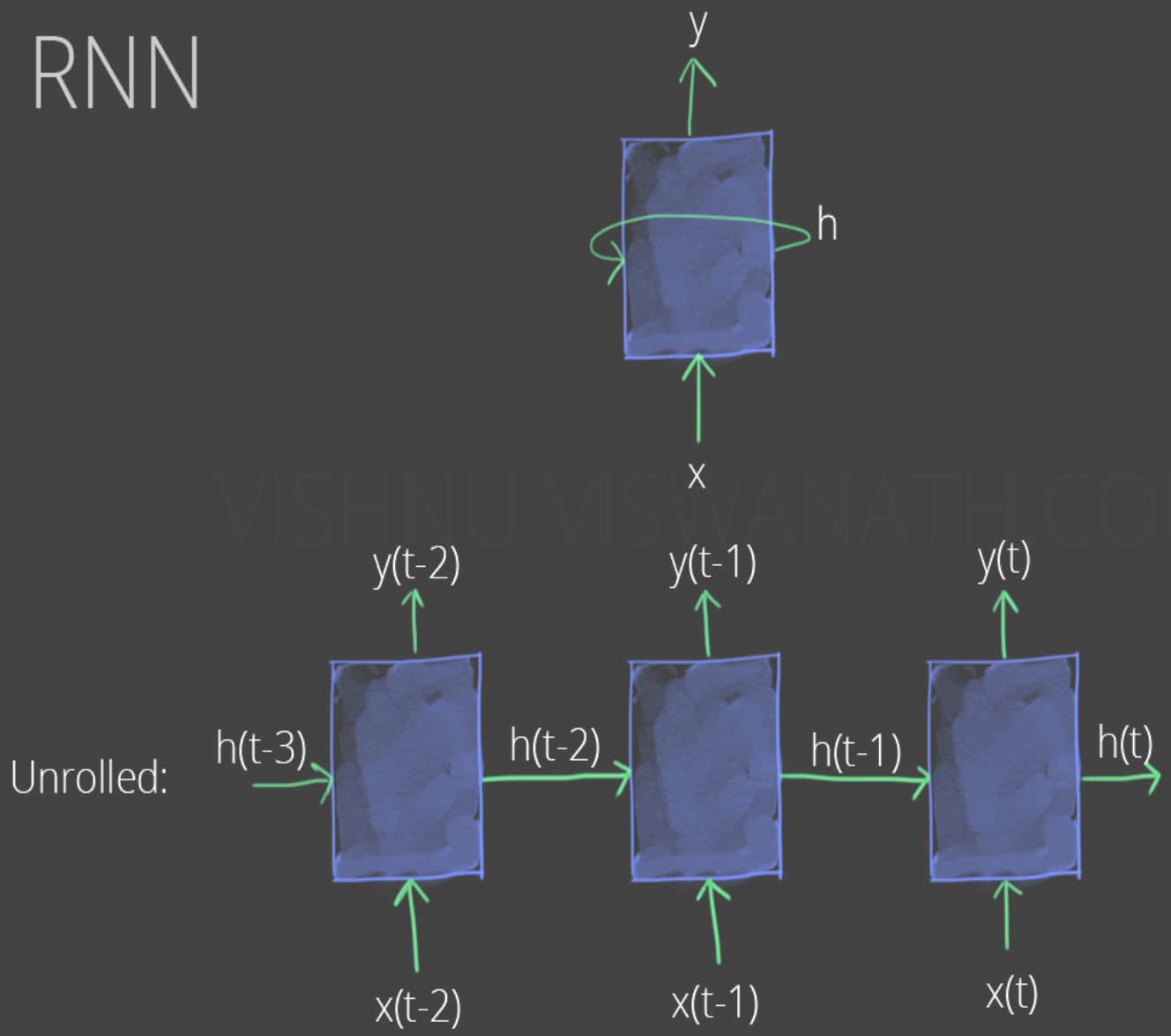
Directed Acyclic Graph(DAG) - a directed graph that contains no cycles.

Why can't we draw an RNN as a DAG?
Because RNNs have cycles in them.

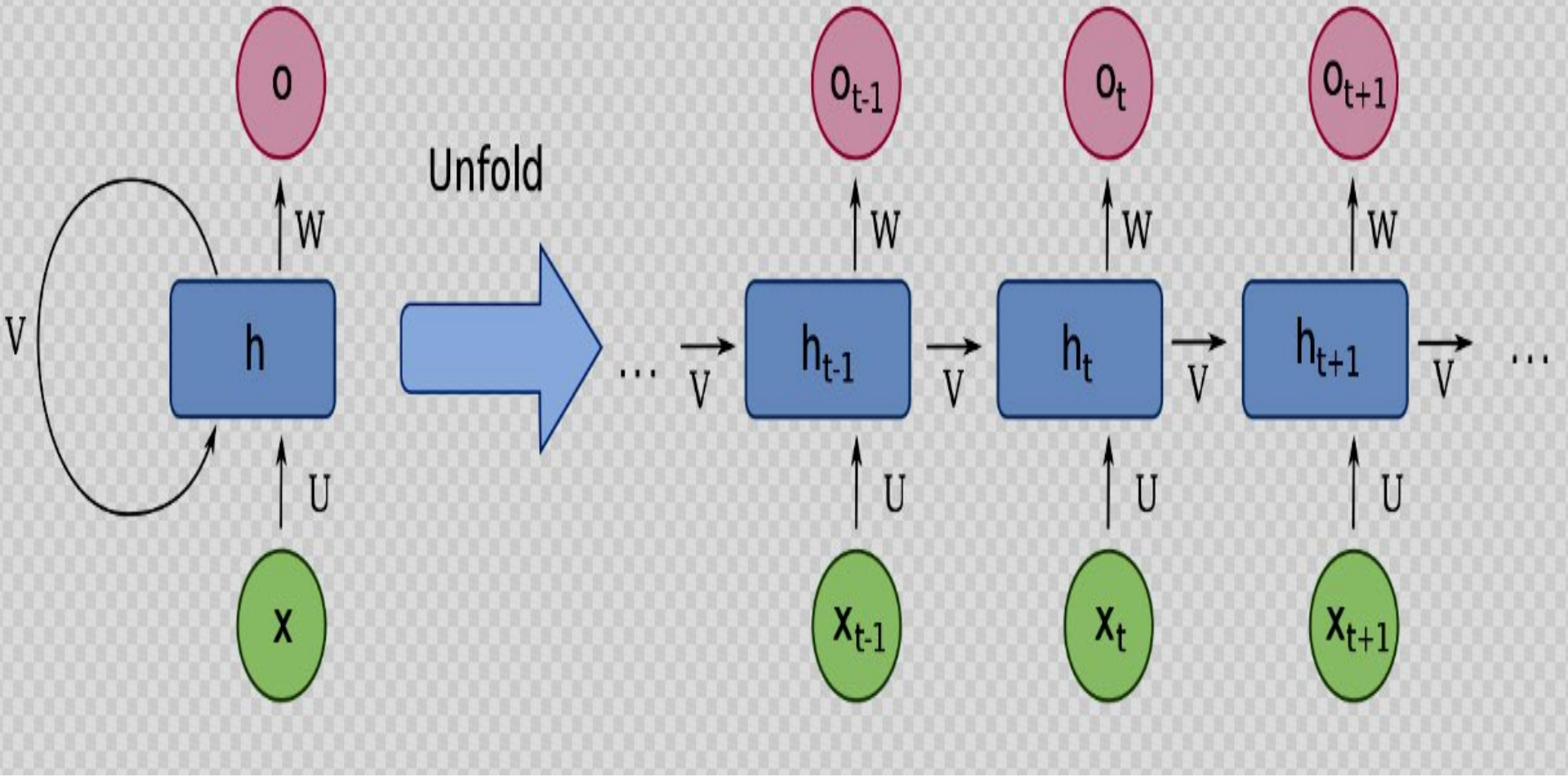


To understand RNNs, we need to remove these cycles and draw the network as a DAG [UNFOLDING]

UNFOLDING RNN



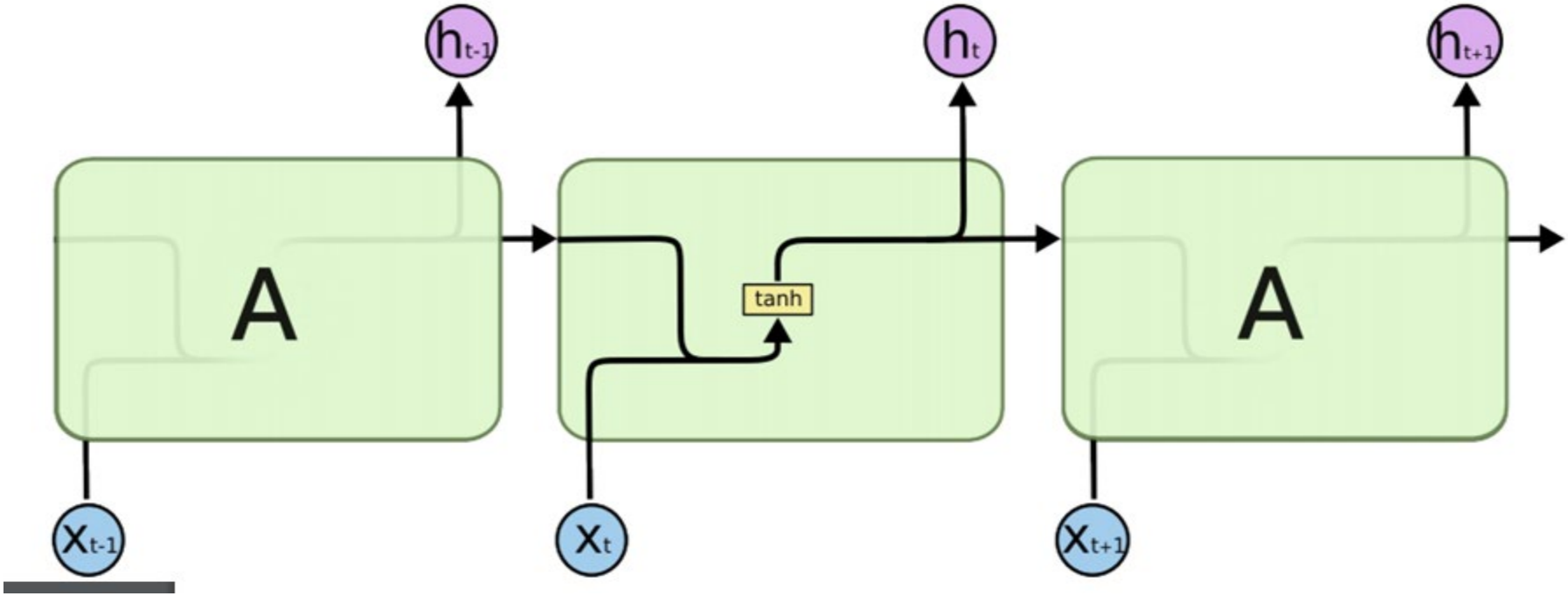
VANILLA RNN



$h_t = f(Ux_t + Wh_{t-1})$

$o_t = g(Vh_t)$ f &g are activation functions

Another way to represent --VANILLA RNN

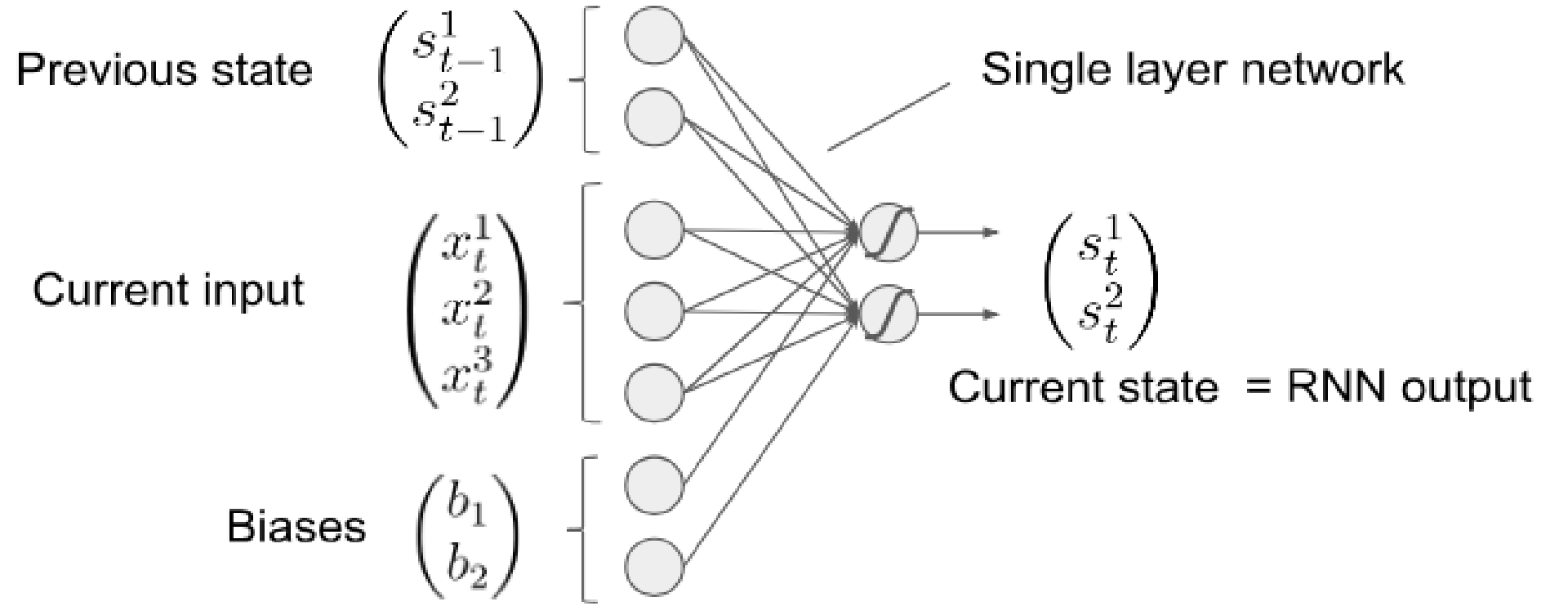


Vanilla RNNs trained with BPTT have difficulties learning long-term dependencies (e.g. dependencies between steps that are far apart) due to what is called the vanishing/exploding gradient problem.

Vanilla RNN

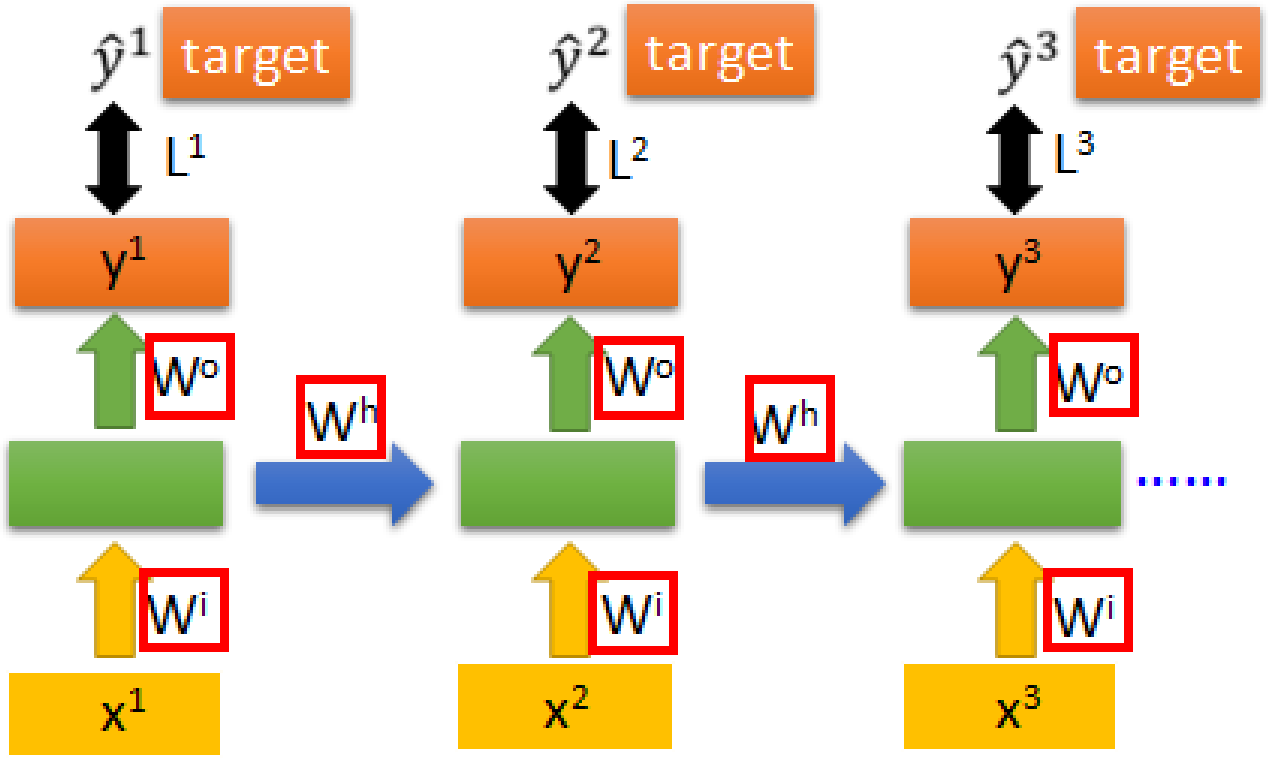
$$s_t = \varphi(W_b[s_{t-1}, x_t])$$

$$\begin{pmatrix} s_t^1 \\ s_t^2 \end{pmatrix} = \varphi\left(\begin{pmatrix} w_{11} & w_{12} & u_{11} & u_{12} & u_{13} & b_1 \\ w_{21} & w_{22} & u_{21} & u_{22} & u_{23} & b_2 \end{pmatrix} \begin{pmatrix} s_{t-1}^1 \\ s_{t-1}^2 \\ x_t^1 \\ x_t^2 \\ x_t^3 \\ 1 \end{pmatrix} \right)$$



RNN

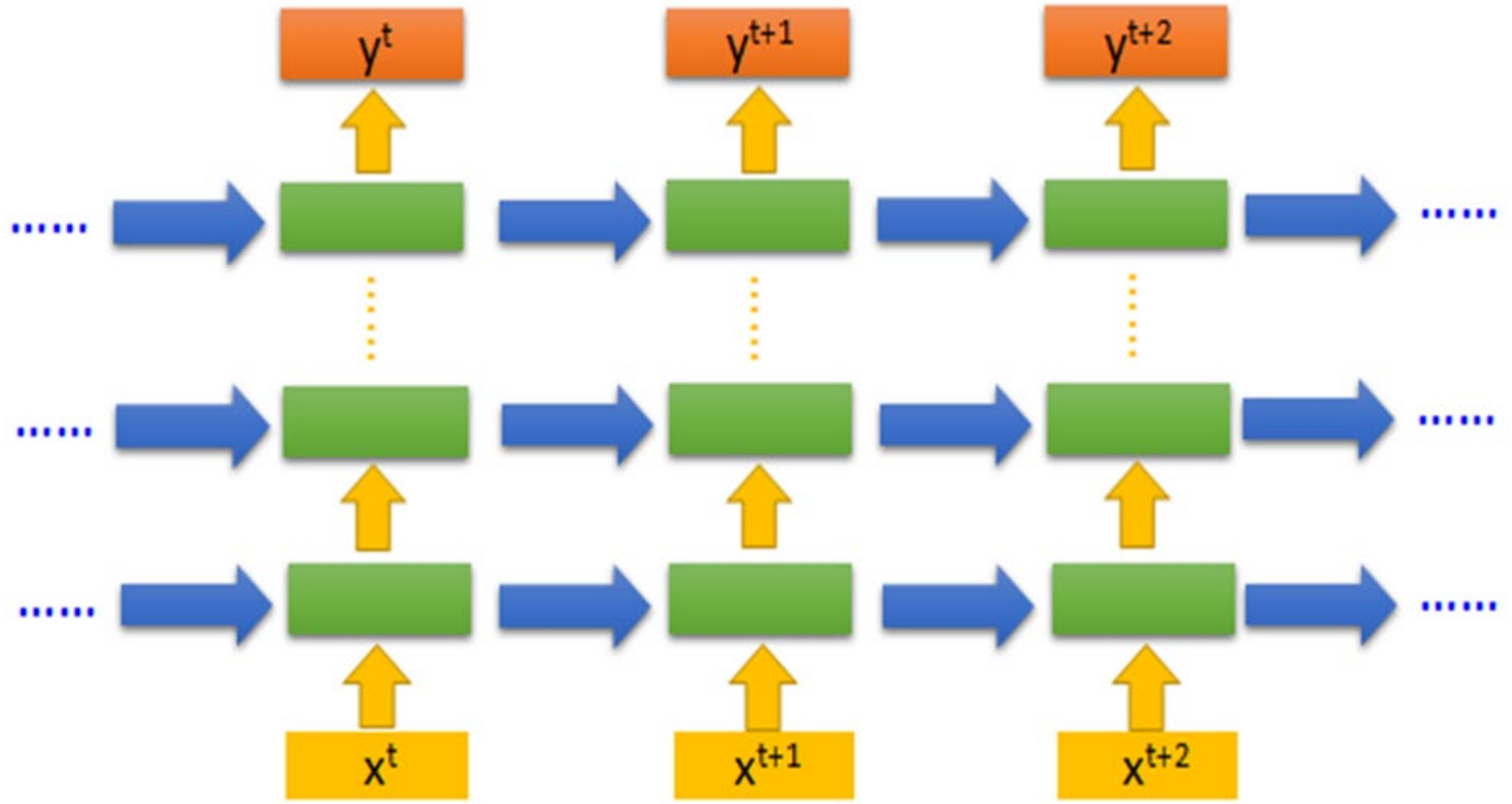
How to train?



Find the network parameters to minimize the total cost:

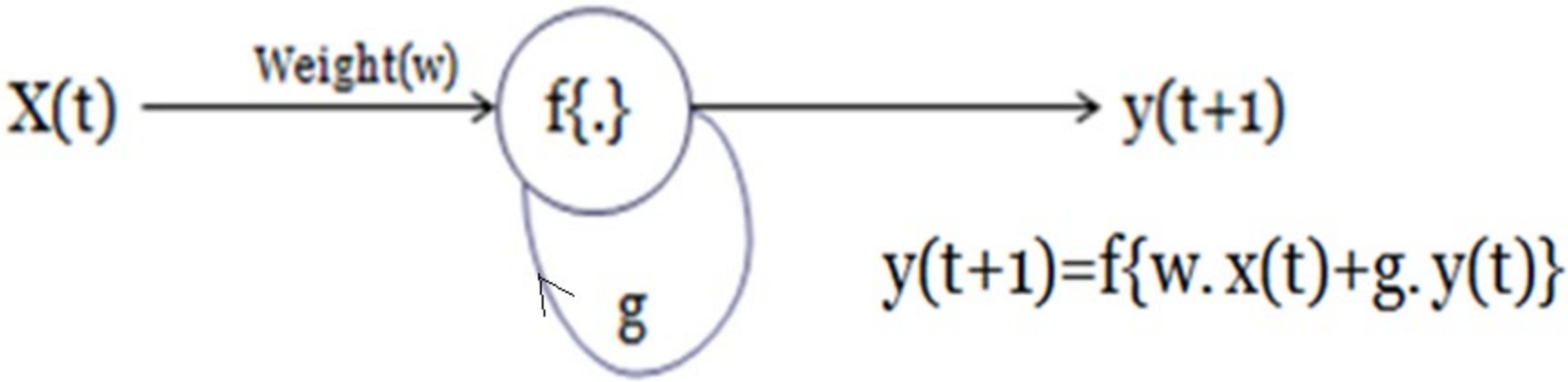
Backpropagation through time (BPTT)

DEEP Recurrent Neural Networks (RNN)



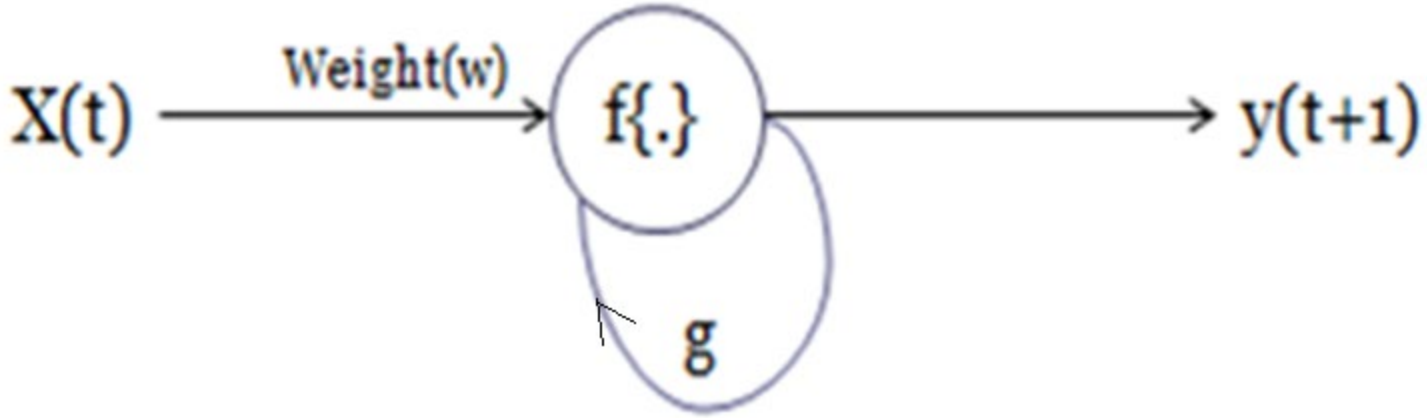
TRAINING A SIMPLE RECURRENT NETWORK
Back propagation through time(BPTT)

- ***Unfolding* an RNN in time and using the extended version of backpropagation.**



Input: x(t), output: y(t+1) PREDICTION, e.g. next word in sequence

If x(t), y(t) pair Translation



Input: x(t), output: y(t+1)
PREDICTION, e.g. next word in sequence

y(t+1)=f{w. x(t)+g. y(t)}

y(1)=f{w. x(0)+g. y(0)}

y(2)=f{w. x(1)+g. y(1)}

y(3)=f{w. x(2)+g. y(2)}

.

.

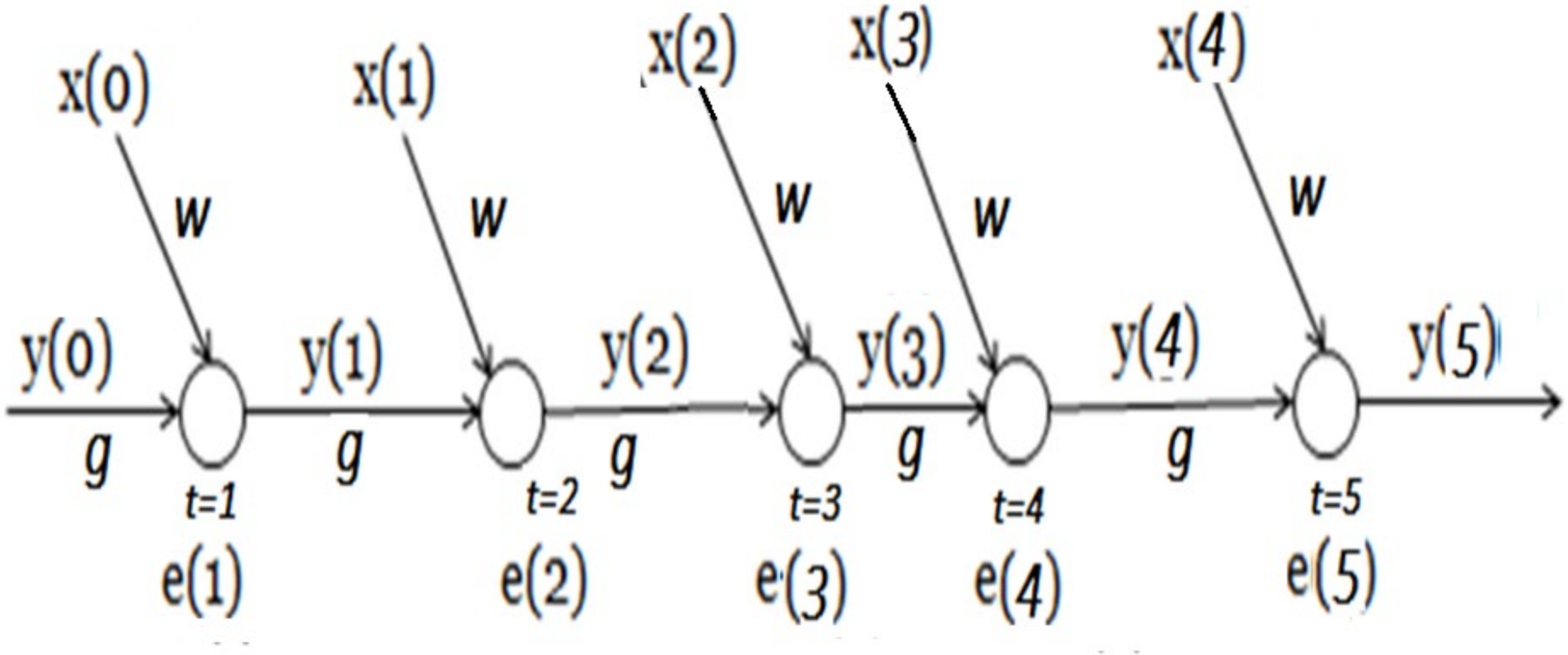
If x(t), y(t) pair Translation
y(t)=f{w. x(t)+g. y(t-1)}

y(1)=f{w. x(1)+g. y(0)}

y(2)=f{w. x(2)+g. y(1)}

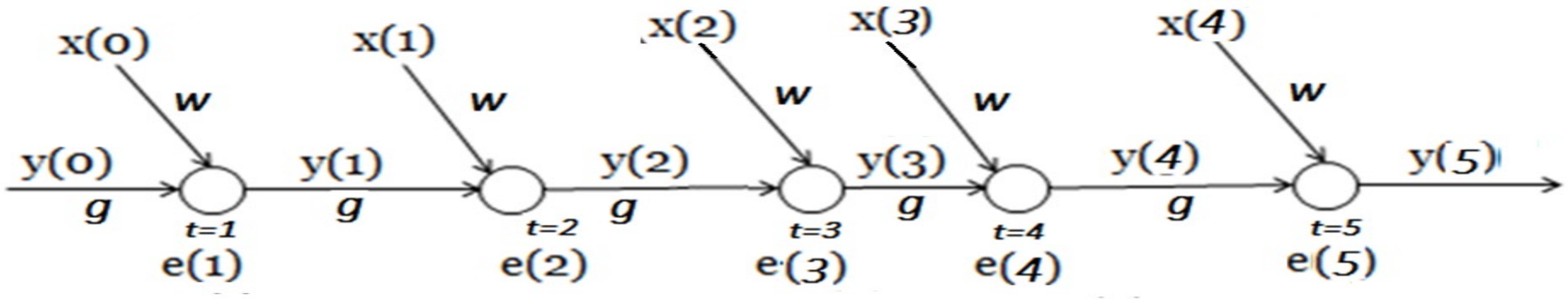
y(3)=f{w. x(2)+g. y(2)}

.



Activation Function : Logsigmoid

p 70 of Behra book ”Intelligent Systems and control”



1. **FORWARD PASS :** Compute the response of the sequence y(1) to y(5), given the sequence x(0) to x(4), and y(0)
2. **BACKWARD PASS**

2.1 compute the error e(5)

$e(5) = y^d(5) - y(5)$

$\delta_5 = (y(5)(1-y(5)))\, e(5)$

$\Delta w^5 = \eta \delta_5 x(4)$

$\Delta g^5 = \eta \delta_5 y(4)$

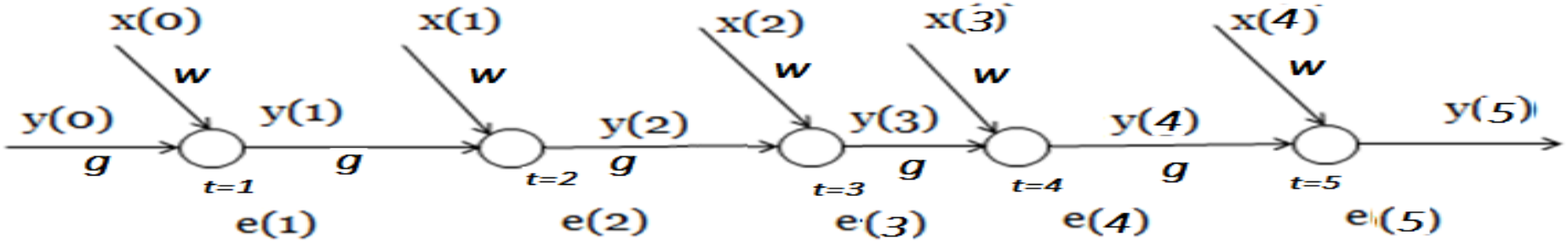
2.2 Compute e(4), e(3), e(2) , and e(1)

$e(4) = y^d(4) - y(4)$

$\delta_4 = (y(4)(1-y(4))) [\delta_5 g + e(4)]$

$\Delta w^4 = \eta \delta_4 x(3)$

$\Delta g^4 = \eta \delta_4 y(3)$



e(3)= $y^d(3) - y(3)$

$\delta_3 = (y(3)(1-y(3)) [\delta_4 g + e(3)]$

$\Delta w^3 = \eta \delta_3 x(2)$

$\Delta g^3 = \eta \delta_3 y(2)$

e(2)= $y^d(2) - y(2)$

$\delta_2 = (y(2)(1-y(2)) [\delta_3 g + e(2)]$

$\Delta w^2 = \eta \delta_2 x(1)$

$\Delta g^2 = \eta \delta_2 y(1)$

e(1)= $y^d(1) - y(1)$

$\delta_1 = y(1)(1-y(1)) [\delta_2 g + e(1)]$

$\Delta w^1 = \eta \delta_1 x(0)$

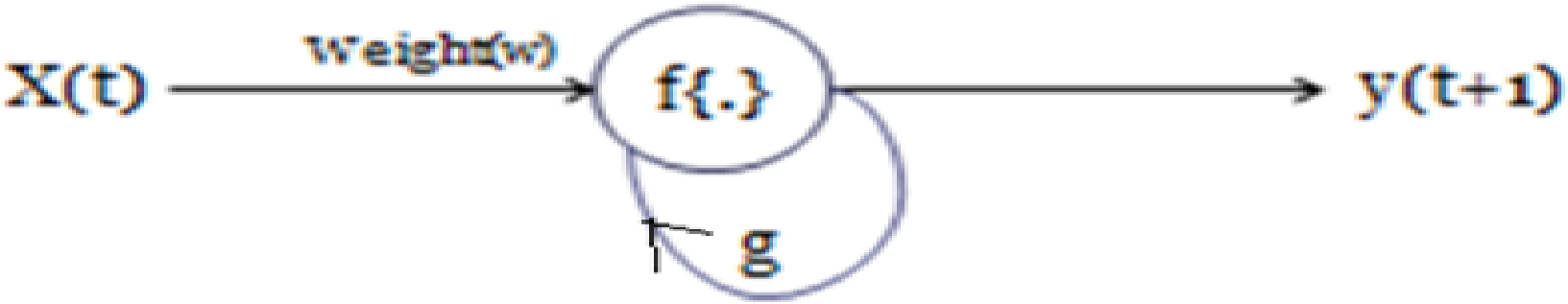
$\Delta g^1 = \eta \delta_1 y(0)$

$$w^{new} = w^{old} + \sum_{i=1}^{5} \Delta w^i$$

$$g^{new} = g^{old} + \sum_{i=1}^{5} \Delta g^i$$

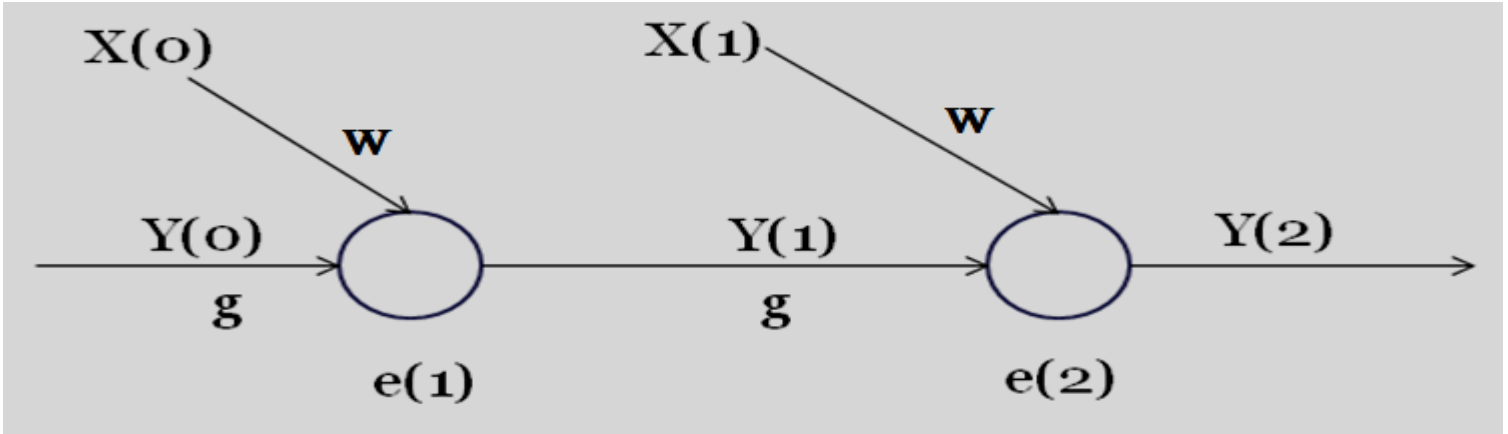
$X(0)=0.05, X(1)=0.1, w=0.3, g=0.4$

$Y(0)=0, Y_d(1)=0.6, Y_d(2)=0.75$

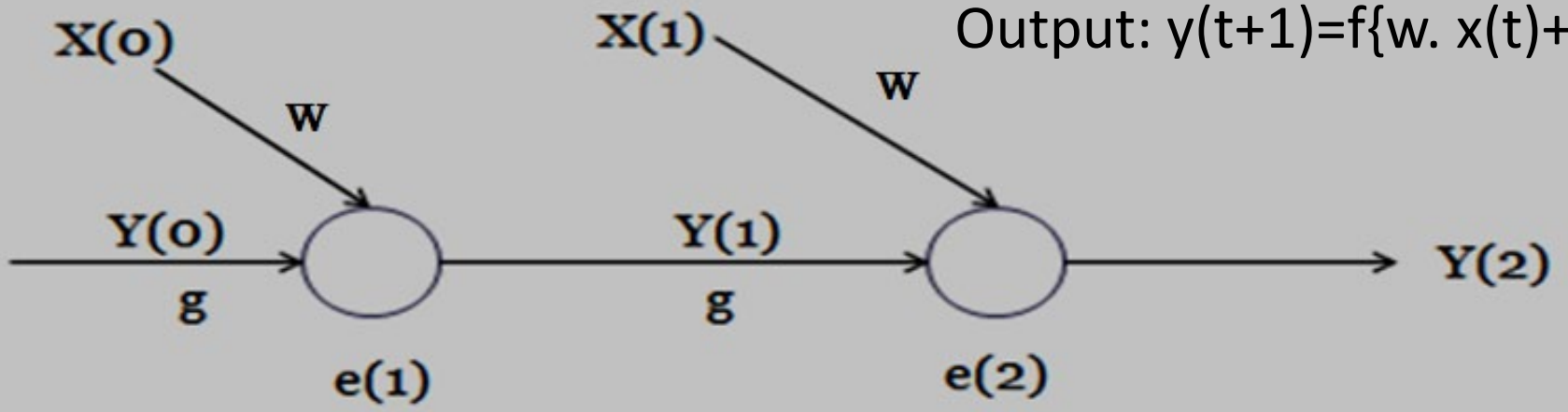


Learning rate(η)=0.5,
Logsigmoid transfer function

Unfolding of network:



X(0)=0.05, X(1)=0.1,w=0.3,g=0.4
$Y(0)=0, Y_d(1)=0.6, Y_d(2)=0.75$
Learning rate(η)=0.5, Logsigmoid transfer function



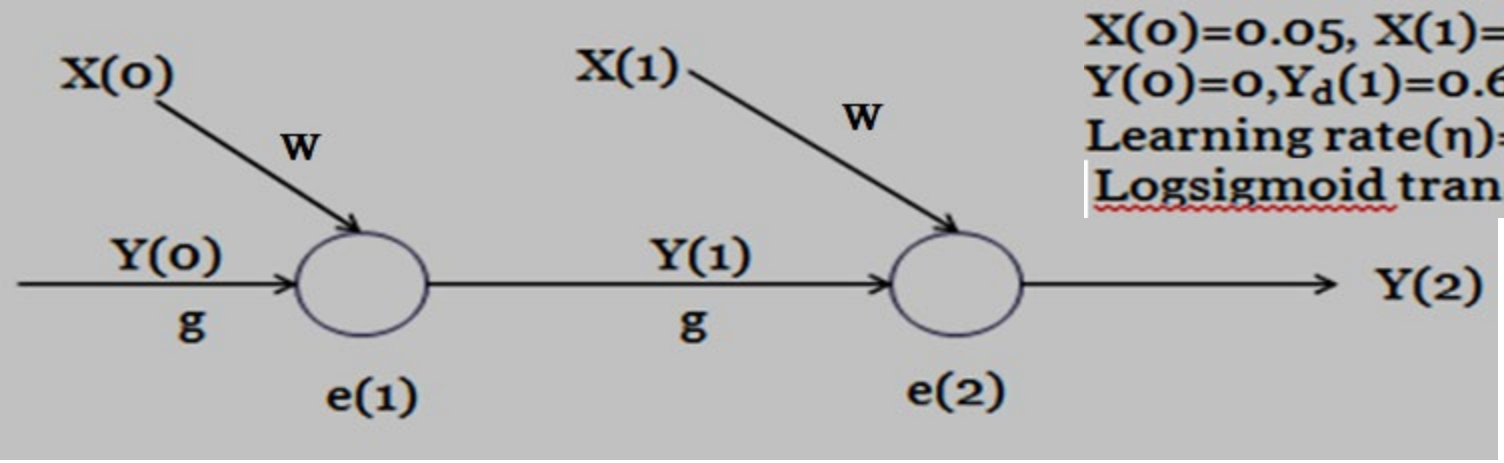
Output: y(t+1)=f{w. x(t)+g. y(t)}

Forward Pass

$$Y(0) = 0$$

$$Y(1) = f\{X(0)\times w + Y(0)\times g\}$$
$$= f\{0.05\times 0.3 + 0\times 0.4\}$$
$$= f(0.015)$$
$$= \frac{1}{1+e^{-0.015}} = 0.5037$$

$$Y(2) = f\{X(1)\times w + Y(1)\times g\}$$
$$= f\{0.10\times 0.3 + 0.5037\times 0.4\}$$
$$= f(0.2314)$$
$$= \frac{1}{1+e^{-0.2314}} = 0.5575$$



Backward Pass :

$e(2)= Y^d(2) - Y(2) = 0.75 - 0.5575 = 0.1924$

$\delta_2 = e(2) \times Y(2) \times (1-Y(2))$

$= 0.1924 \times 0.5575 \times 0.4425$

$= 0.0474$

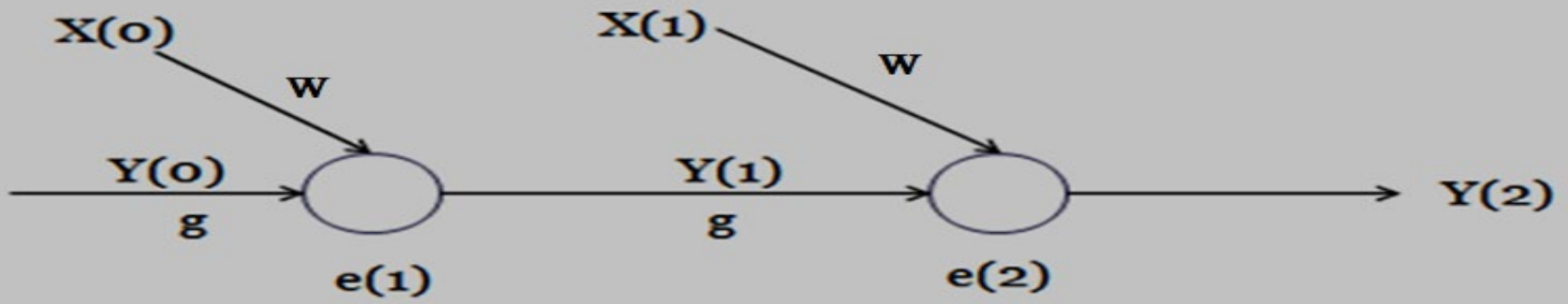
$e(1)= Y^d(1) - Y(1) = 0.6 - 0.5037 = 0.0963$

$\delta_1 = Y(1) \times (1-Y(1)) \times [e(1)+\delta_2 \times g]$

$= 0.5037 \times 0.4963 \times [0.0963+0.0474 \times 0.4]$

$= 0.0288$

X(0)=0.05, X(1)=0.1,w=0.3,g=0.4
Y(0)=0,$Y_d(1)$=0.6,$Y_d(2)$=0.75
Learning rate(η)=0.5, Logsigmoid transfer function



$\delta_2 = 0.0474$; $\delta_1 = 0.0288$

$\Delta w_2 = \eta \times \delta_2 \times X(1) = 0.5 \times 0.0474 \times 0.1 = 0.00237$

$\Delta g_2 = \eta \times \delta_2 \times Y(1) = 0.5 \times 0.0474 \times 0.5037 = 0.0119$

$\Delta w_1 = \eta \times \delta_1 \times X(0) = 0.5 \times 0.0288 \times 0.0 = 0.0007$

$\Delta g_1 = \eta \times \delta_1 \times Y(0) = 0.5 \times 0.0288 \times 0 = 0$

$$w_{new} = w_{old} + (\Delta w_2 + \Delta w_1) = 0.3 + (0.00237 + 0.0007) = 0.300307$$

$$g_{new} = g_{old} + (\Delta g_2 + \Delta g_1) = 0.4 + (0 + 0.0119) = 0.4199$$

Truncated BPTT is an approximation of full BPTT that is preferred for long sequences, since full BPTT's forward/backward cost per parameter update becomes very high over many time steps.

Truncated BPTT is the exact same thing but instead of propagating to the beginning of the sequence you only propagate backwards k steps.

END