國立高雄科技大學
National Kaohsiung University of Science and Technology

# Simple Recurrent Neural Network (RNN):

## An Example of Price Forecasting

Speaker: Shih-Shinh Huang

July 11, 2024

# Outline

- Introduction
  - About RNN
  - Model Concept
  - Price Forecasting
- RNN Architecture
  - Model State and Parameters
  - Recurrent Inference
  - Price Forecasting
- RNN Training
  - Training Process
  - Loss Definition

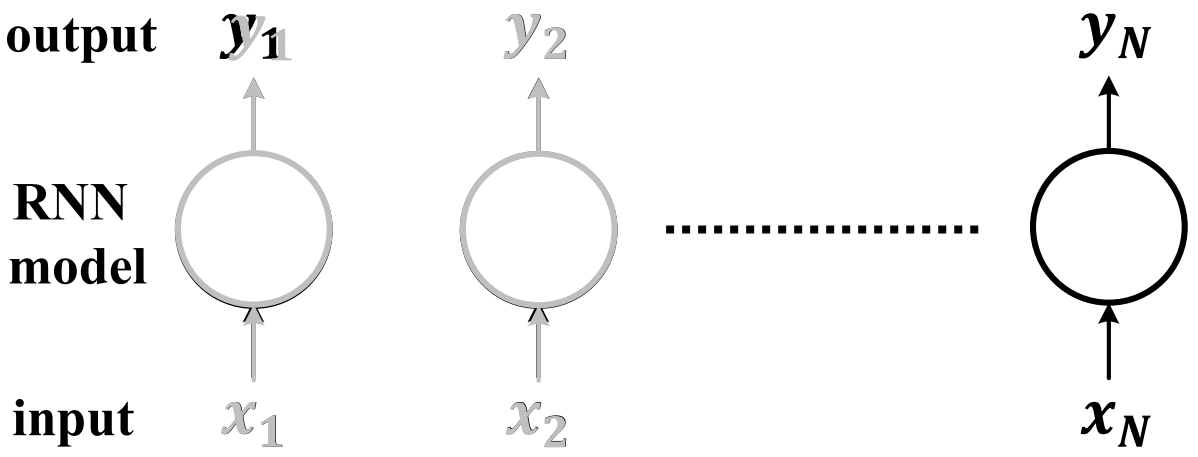
# Introduction

- About RNN
  - The RNN is a model for dealing with **sequential data**, $x_1, x_2, \ldots, x_N$, consisting of $N$ items.
    - The length of the sequence ($N$) changes over time.
    - The data items are mutually dependent.

# Introduction

- Model Concept
  - recurrence: repeatedly performs the same task for every data item in a sequence.

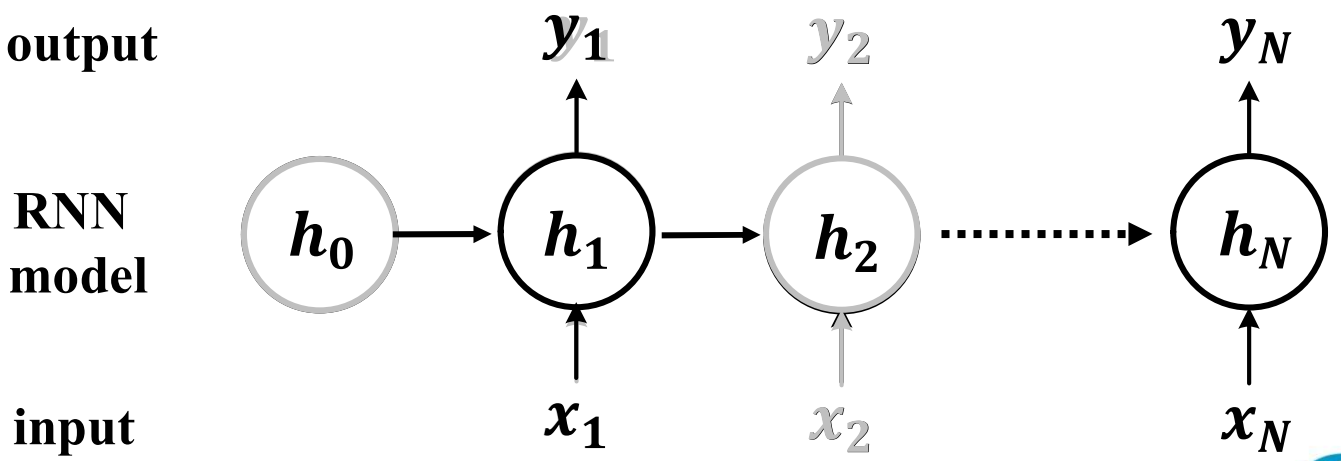
⇨ allow the input to be the various-length sequence.



# Introduction

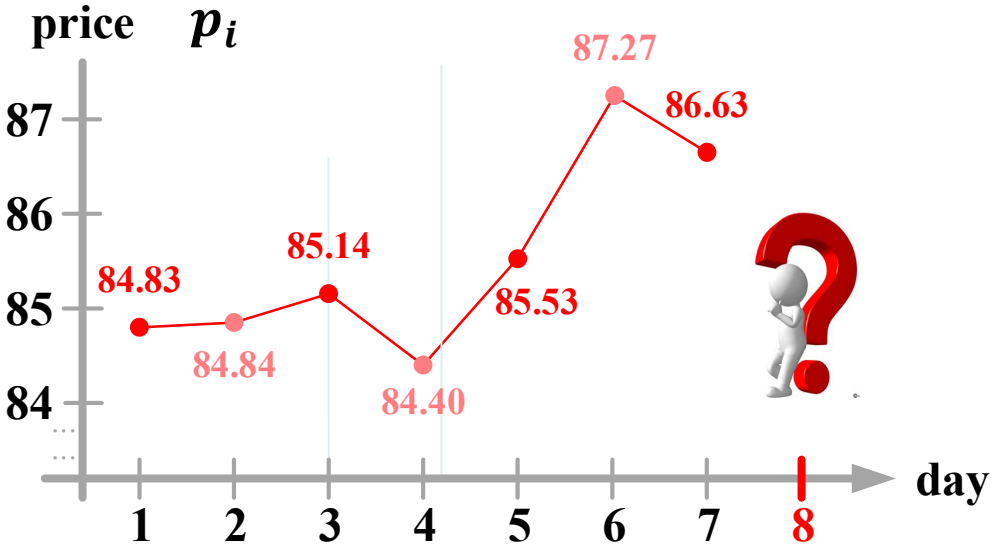
- Model Concept
  - state $(h)$: memorize the passed information.

⇨ model the dependency among data items



# Introduction

- Price Forecasting
  - input: a sequence of the past prices




  - output: the next price in the sequence

# Introduction

- Price Forecasting
  - **normalize** the raw prices $p_i$ to the values $q_i$

$$q_i = (p_i - \mu)/\sigma$$

$\mu$: mean

$\sigma$: standard deviation

| raw $p_i$ | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 84.83 | 84.84 | 85.14 | 85.40 | 85.53 | 87.27 | 86.63 |

$\mu = 85.52$

$\sigma = 0.97$

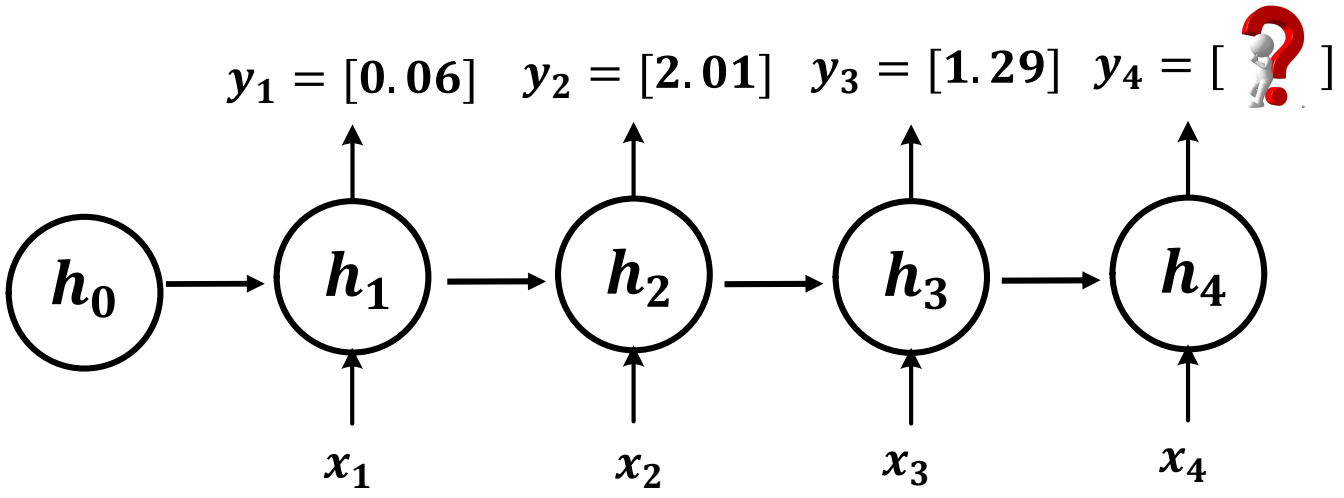
$$q_i = (p_i - 85.52)/\ 0.97$$

| normalized $q_i$ | | | | | | | |
|---|---|---|---|---|---|---|---|
| | -0.72 | -0.71 | -0.37 | -1.2 | 0.06 | 2.01 | 1.29 |

# Introduction

- Price Forecasting
  - take **four** values for predicting the next one

| $q_i$ | -0.72 | -0.71 | -0.37 | -1.2 | 0.06 | 2.01 | 1.29 | ? |
|---|---|---|---|---|---|---|---|---|




$[-0.72, -0.71, -0.37, -1.2, 0.06, 2.01, 1.29]$

# RNN Architecture

- Model State and Parameters
  - model state ($h$): memorizes past information
    - $h$ is **hidden** (cannot be directly observed)
    - $h$ is in the form of a $|h|$**-d vector**.

$$h = [h_1, h_2, \ldots, h_{|h|}]$$

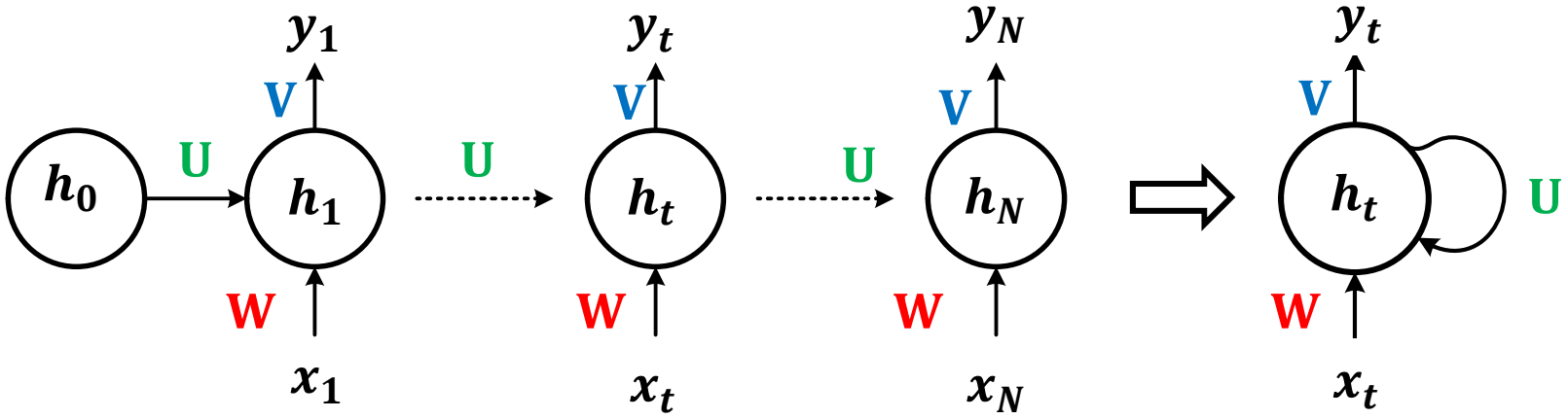
vector dimension is a constant
and specified by the user

# RNN Architecture

- Model State and Parameters
  - model parameters ($\theta$): weights **shared** across times

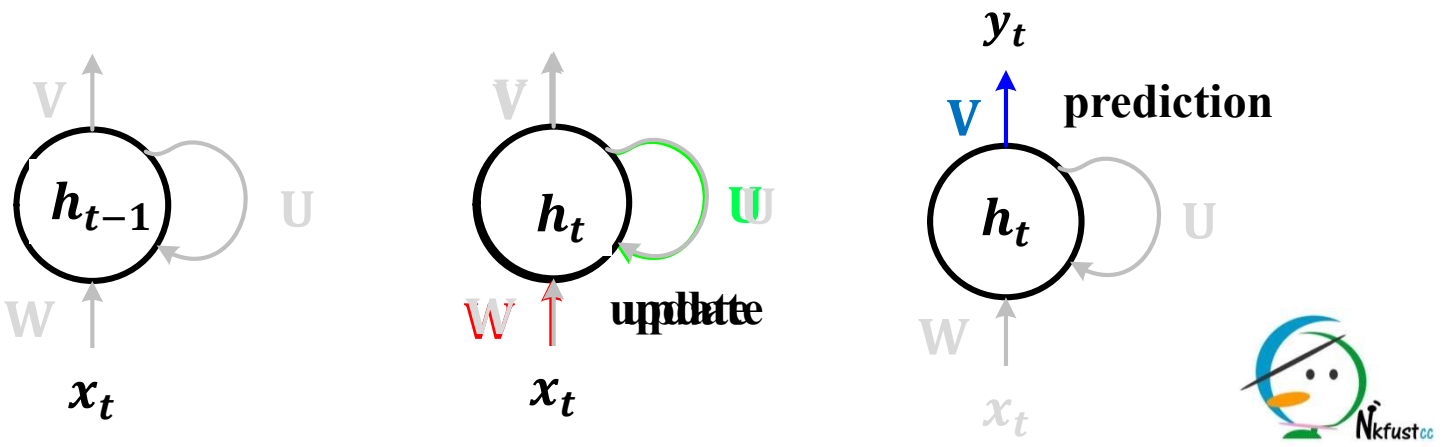
$$\theta = (\mathbf{W}, \mathbf{U}, \mathbf{V})$$

input weights  state transition weights  output weights

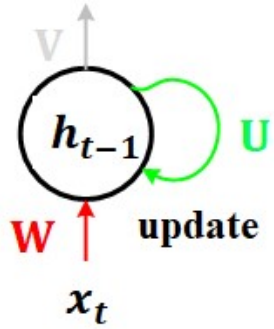


# RNN Architecture

- Recurrent Inference: Formulation
  - given: input $\boldsymbol{x_t}$ and the previous state $\boldsymbol{h_{t-1}}$
  - objective: output $\boldsymbol{y_t}$
    - **update stage:** evolve the state from $\boldsymbol{h_{t-1}}$ to $\boldsymbol{h_t}$
    - **prediction stage:** use $\boldsymbol{h_t}$ to produce $\boldsymbol{y_t}$



# RNN Architecture




- Recurrent Inference: update
  - Step 1: transform the input $x_t$ to the state $u_t$ by **W**.

transpose

$$(u_t)^T = \mathbf{W} \cdot (x_t)^T \Rightarrow$$ **W**: |**h**| × |**x**| matrix

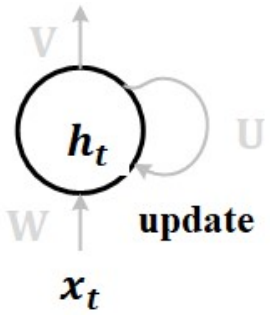
|**h**|-d state vector |**x**|-d input vector

  - Step 2: use **U** to evolve the state from $h_{t-1}$ to $h_t^-$

$$(h_t^-)^T = \mathbf{U} \cdot (h_{t-1})^T \Rightarrow$$ **U**: |**h**| × |**h**| matrix

|**h**|-d state vector

# RNN Architecture

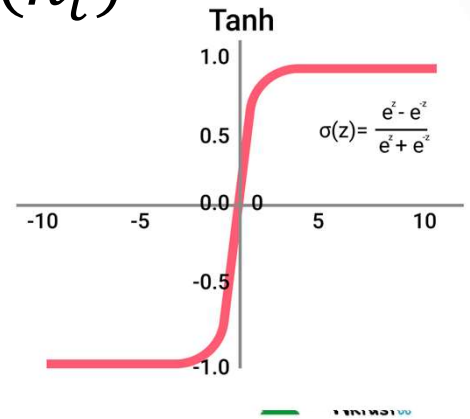



- Recurrent Inference: update
  - Step 3: fuse the data from the input and the past by **adding** $u_t$ and $h_t^-$
  - Step 4: apply **tanh**() to compute the state $h_t$

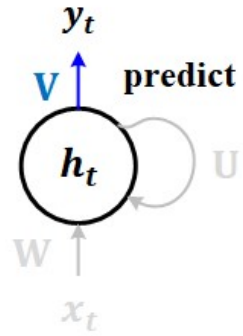
$$\tanh\left(\mathbf{W}\cdot(x_t)^T + \mathbf{U}\cdot(h_{t-1})^T\right) = (h_t)^T$$

$u_t$ $\qquad$ $h_t^-$

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$



# RNN Architecture




- Recurrent Inference: prediction
  - transform the state $h_t$ to output $y_t$ by **V**.

$$(y_t)^T = \mathbf{V} \cdot (h_t)^T \Rightarrow$$ **V**: |**y**| × |**h**| matrix

|**y**|-d output vector |**h**|-d state vector

# RNN Architecture

- Price Forecasting
  - take four values for predicting one price

    ⇨ $|\mathbf{x}| = 4 \quad |\mathbf{y}| = 1$

  - model hidden state $h$ as a 2-D vector and initialize to zero vector

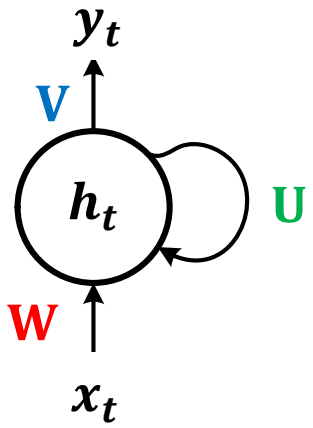
    $|\mathbf{h}| = 2$

    ⇨

    $h_0 = [0.0, 0.0]$

# RNN Architecture

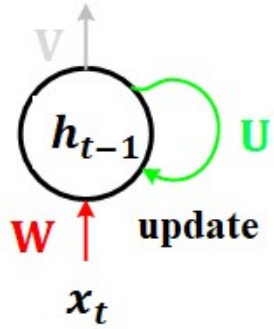
- Price Forecasting
  - model parameters
    - **W** (input → state) : $2 \times 4$ matrix
    - **U** (state → state) : $2 \times 2$ matrix
    - **V** (state → output): $1 \times 2$ matrix




$$\mathbf{W} = \begin{bmatrix} -0.14 & -0.09 & 0.39 & 0.00 \\ -0.48 & 0.06 & 0.30 & 1.09 \end{bmatrix}_{2\times4}$$

$$\mathbf{U} = \begin{bmatrix} -0.34 & 0.59 \\ -0.52 & 0.32 \end{bmatrix}_{2\times2} \qquad \mathbf{V} = \begin{bmatrix} 0.10 & 1.22 \end{bmatrix}_{1\times2}$$

# RNN Architecture




- Price Forecasting: update

| | | | | | | |
|---|---|---|---|---|---|---|
| -0.72 | -0.71 | -0.37 | -1.2 | 0.06 | 2.01 | 1.29 |

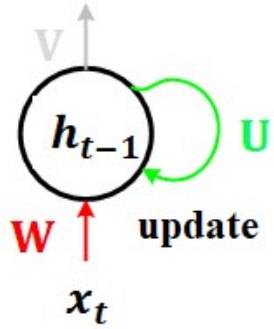
$$x_1 = [-0.72, -0.71, -0.37, -1.2]$$

- Step 1: transform the $x_1$ to $u_1$ by **W**.

$$(u_1)^T = \mathbf{W} \cdot (x_1)^T$$

$$\begin{bmatrix} 0.02 \\ -1.16 \end{bmatrix} = \begin{bmatrix} -0.14 & -0.09 & 0.39 & 0.00 \\ -0.48 & 0.06 & 0.30 & 1.09 \end{bmatrix} \times \begin{bmatrix} -0.72 \\ -0.71 \\ -0.37 \\ -1.2 \end{bmatrix}$$

# RNN Architecture




- Price Forecasting: update

| | | | | | | |
|---|---|---|---|---|---|---|
| -0.72 | -0.71 | -0.37 | -1.2 | 0.06 | 2.01 | 1.29 |

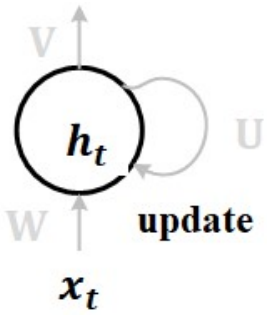
$h_0 = [0.0, 0.0]$

- Step 2: evolve the state from $h_0$ to $h_1^-$ by **U**

$$(h_1^-)^T = \mathbf{U} \cdot (h_0)^T$$

$$\begin{bmatrix} 0.0 \\ 0.0 \end{bmatrix} = \begin{bmatrix} -0.34 & 0.59 \\ -0.52 & 0.32 \end{bmatrix} \times \begin{bmatrix} 0.0 \\ 0.0 \end{bmatrix}$$

# RNN Architecture




- Price Forecasting: update

| | | | | | | |
|---|---|---|---|---|---|---|
| -0.72 | -0.71 | -0.37 | -1.2 | 0.06 | 2.01 | 1.29 |

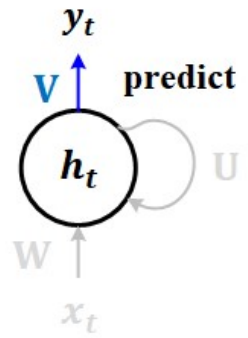
- Step 3: fuse $u_1$ and $h_1^-$ by **addition**

$$(u_1)^T + (h_1^-)^T = \begin{bmatrix} \mathbf{0.02} \\ \\ \mathbf{-1.16} \end{bmatrix} + \begin{bmatrix} \mathbf{0.0} \\ \\ \mathbf{0.0} \end{bmatrix} = \begin{bmatrix} \mathbf{0.02} \\ \\ \mathbf{-1.16} \end{bmatrix}$$

- Step 4: apply **tanh**() to compute the state $h_1$

$$\mathbf{tanh}\left(\begin{bmatrix} \mathbf{0.02} \\ \\ \mathbf{-1.16} \end{bmatrix}\right) = \begin{bmatrix} \mathbf{tanh(0.02)} \\ \\ \mathbf{tanh(-1.16)} \end{bmatrix} \Rightarrow (h_1)^T = \begin{bmatrix} \mathbf{0.02} \\ \\ \mathbf{-0.8} \end{bmatrix}$$

# RNN Architecture




- Price Forecasting: prediction

| | | | | | | |
|---|---|---|---|---|---|---|
| -0.72 | -0.71 | -0.37 | -1.2 | 0.06 | 2.01 | 1.29 |

- transform the $h_1$ to $y_1$ by $\mathbf{V}$

$$(y_1)^T = \mathbf{V} \cdot (h_1)^T$$

$$[-0.98] = [0.10 \quad 1.22] \times \begin{bmatrix} 0.02 \\ -0.8 \end{bmatrix}$$

- un-normalize $y_1$ to the real price

$$\text{price} = y_1 * 0.97 + 85.52 = 84.56$$

# RNN Training

- Training Overview
  - split data sequences into $k$ data segments

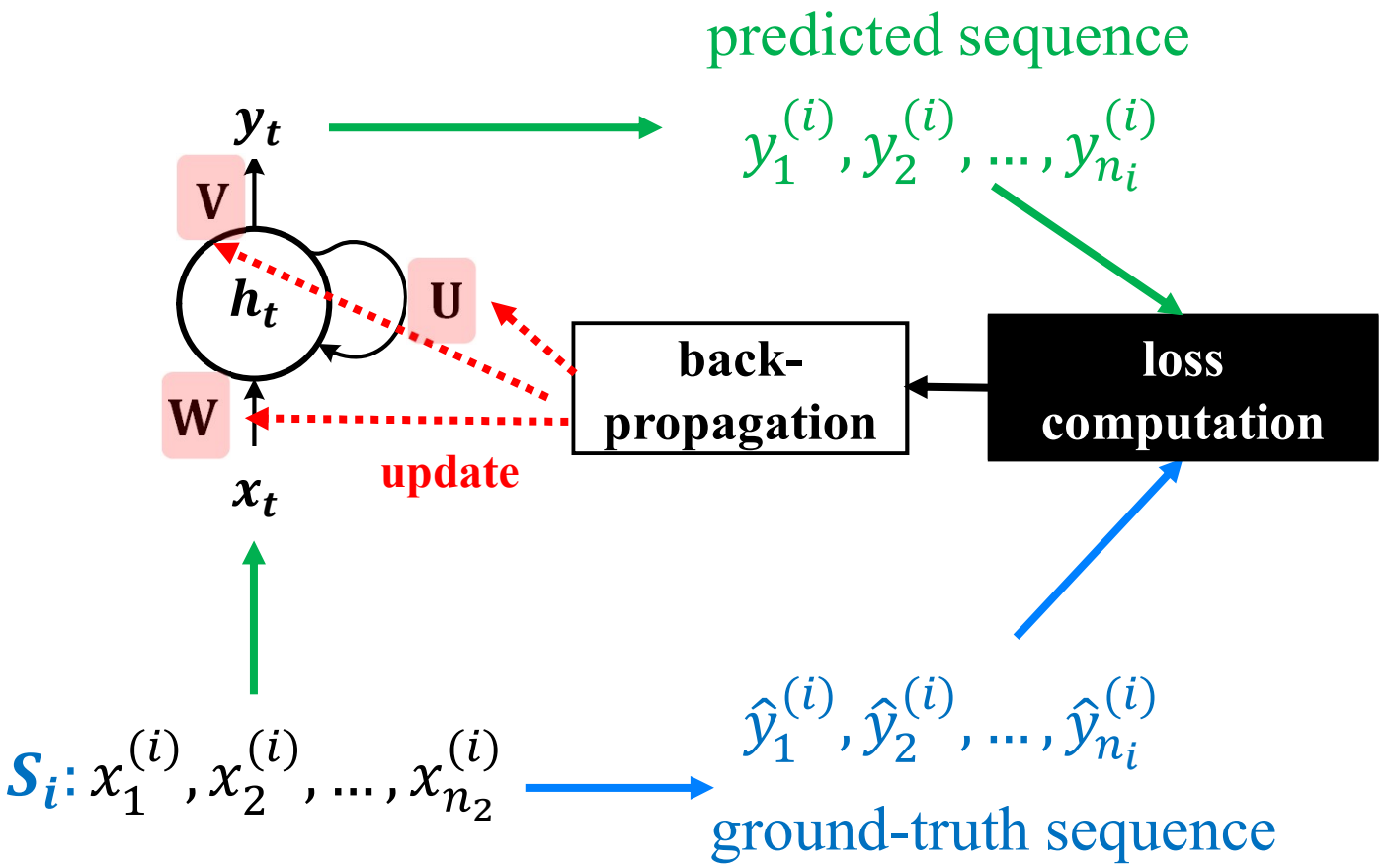
    $\boldsymbol{S} =< \boldsymbol{S_1}, \boldsymbol{S_2}, \ldots \boldsymbol{S_k} >$
  - randomly select a data segment $\boldsymbol{S_i}$ from $\boldsymbol{S}$

    $\boldsymbol{S_i} = x_1^{(i)}, x_2^{(i)}, \ldots, x_{n_i}^{(i)}$
  - flow through RNN to produce the output sequence

    $y_1^{(i)}, y_2^{(i)}, \ldots, y_{n_i}^{(i)}$
  - compute prediction loss for updating $\mathbf{W}, \mathbf{U}, \mathbf{V}$

$$x_1, x_2, x_3, x_4 \dots\dots\dots, x_N$$

$S_1$: $x_1^{(1)}, x_2^{(1)}, \dots, x_{n_1}^{(2)}$ ..... $S_i$: $x_1^{(i)}, x_2^{(i)}, \dots, x_{n_2}^{(i)}$ ..... $S_k$: $x_1^{(k)}, x_2^{(k)}, \dots, x_{n_2}^{(k)}$



# RNN Training

- Loss Definition
  - measure the error between the prediction $y_1, y_2, .. y_N$ and the ground truth $\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_N$
  - ✔ mean square error (MSE) (value prediction)

$$L_{MSE} = \frac{1}{N} \times \sum_i (y_i - \hat{y}_i)^2$$

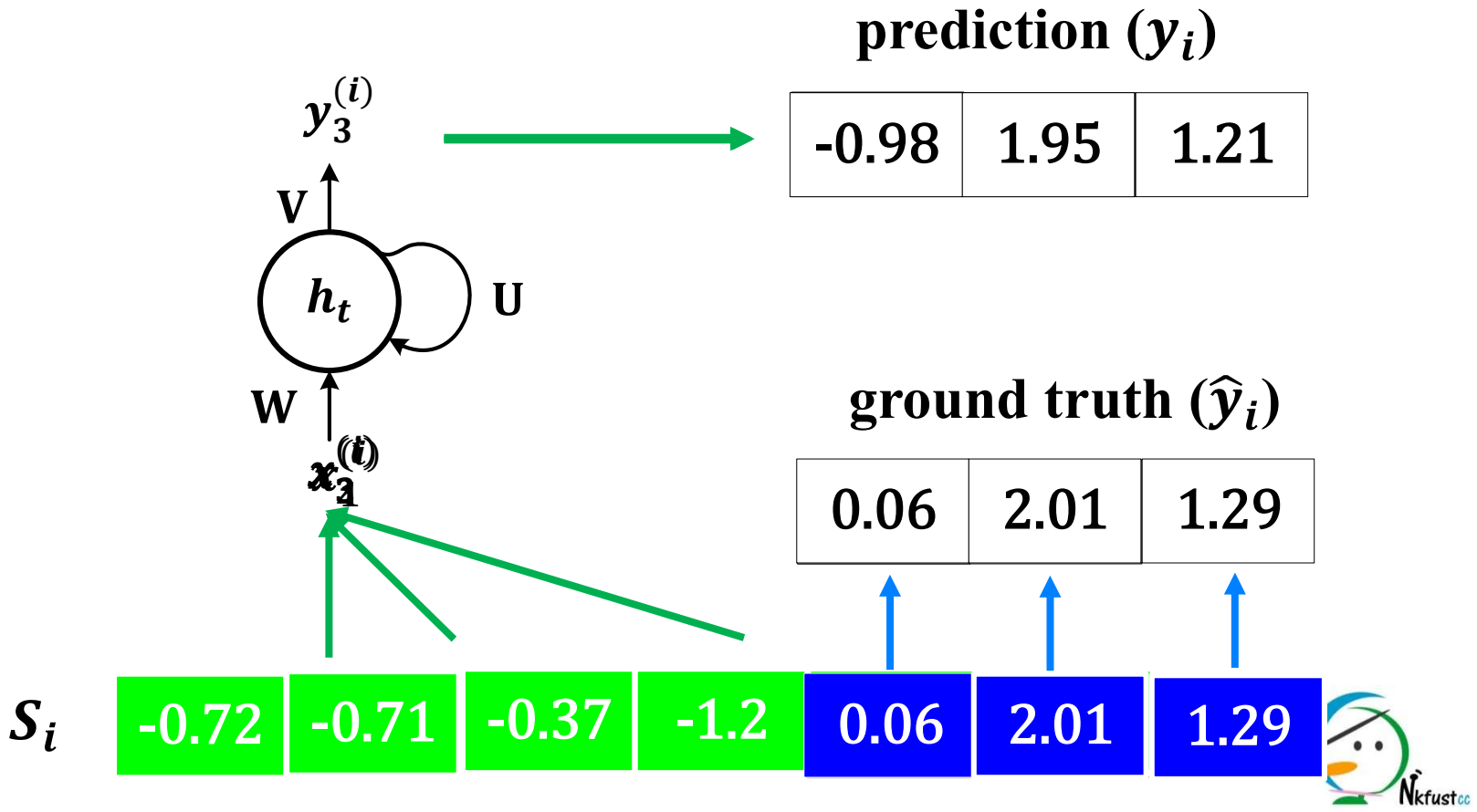
average — total square error

  - cross entropy (classification)

$$L_E = -\sum_i \hat{y}_i \times \log(y_i)$$

# RNN Training

- Loss Definition: Price Forecasting



# RNN Training

- Loss Definition: Price Forecasting

| prediction ($y_i$) | | | ground truth ($\hat{y}_i$) | | |
|---|---|---|---|---|---|
| -0.98 | 1.95 | 1.21 | 0.06 | 2.01 | 1.29 |

$$L_{MSE} = \frac{1}{3} \times \sum_i (y_i - \hat{y}_i)^2$$

average — total square error

$$= \frac{1}{3} \times \left( (-0.98 - 0.06)^2 + (1.95 - 2.01)^2 + (1.21 - 1.29)^2 \right)$$

$$= 1.09$$


thank you
© 1996, 2002 SANRIO CO., LTD.