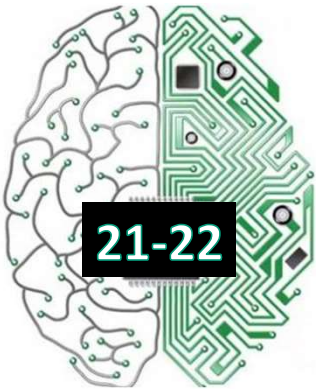
Open Elective Course [OE]
Course Code: CSO507
Winter 2023-24

Lecture#

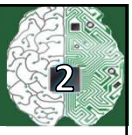
Deep Learning

Unit-5: Sequence Modeling with Recurrent Neural Network (RNN)_Part-II&III



Course Instructor:
Dr. Monidipa Das
Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Technology (Indian School of Mines) Dhanbad, Jharkhand 826004, India

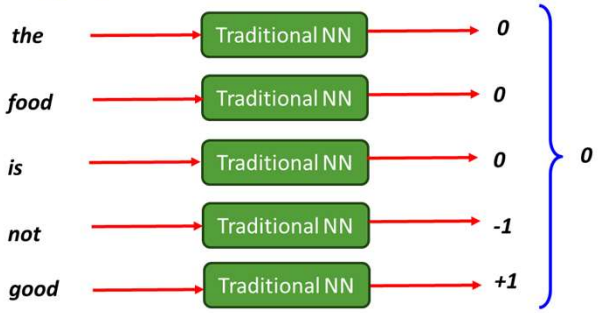
Sequential Data



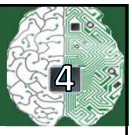
- Data where the order matters
  - The man went to the _____ to withdraw money
  - The boy went to the _____ to purchase some medicines for his mother
  - The food was not bad, great in fact
  - The food was not great, bad in fact

Challenges in Modelling Sequential Data

- Traditional NN cannot keep track of sequence
  - Categorizing a piece of text: "***the food is not good***"



Challenges in Modelling Sequential Data



- Dealing with variable size input using traditional NN

  The food was not good

  The food was not good, surprisingly instead of the ratings for the restaurant being high

  – Solution1: Can take a moving window with last k input items

    Mr. X who is an ex-employee of company Y is currently running his own business.

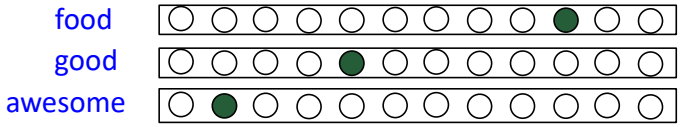
  – Solution2: Take a bag or set-based representation

    The food was not bad, great in fact

    The food was not great, bad in fact

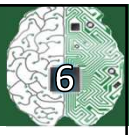
Challenges in Modelling Sequential Data

- Problems with representation
- Solution: Vector space representation



- Extremely high dimensional, sparse
- Problem is due to sparsity of **one-hot representation**

One-hot encoding

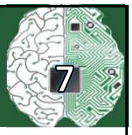


- Categorical data are usually represented by ***one-hot encoding vector***
- Here, a vector of length 10,000 is associated to each word of the vocabulary :

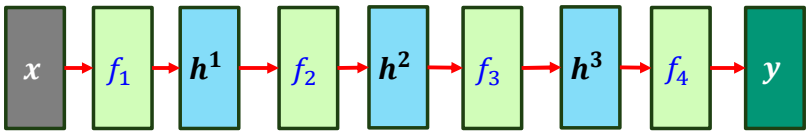
$$\text{word 1} = \begin{bmatrix}1\\0\\0\\0\\\vdots\\0\\0\end{bmatrix} \quad \text{word 2} = \begin{bmatrix}0\\1\\0\\0\\\vdots\\0\\0\end{bmatrix} \quad \text{word 3} = \begin{bmatrix}0\\0\\1\\0\\\vdots\\0\\0\end{bmatrix} \quad \ldots \quad \text{word 10,000} = \begin{bmatrix}0\\0\\0\\0\\\vdots\\0\\1\end{bmatrix}$$

Word 1: "the" Word 2: "and" Word 3: "a" Word 10,000: "<unk>"

Challenges in Modelling Sequential Data

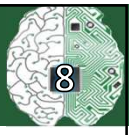


- No Shared Representation
- Feedforward network has different parameters for each layer

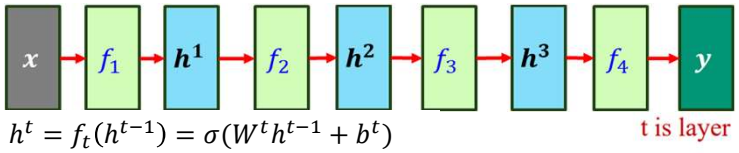


- **Solution:** Shared parameters for different parts of the input

Feed Forward Network vs Recurrent Network

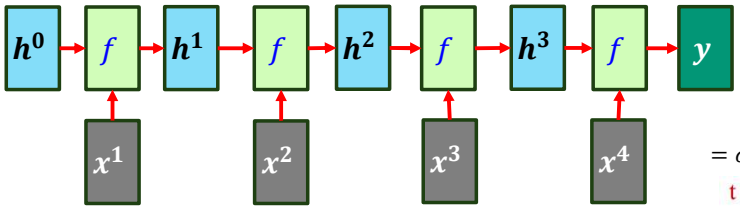


- Feed forward network does not have input at each step
- Feed forward network has different parameters for each layer



$h^t = f_t(h^{t-1}) = \sigma(W^t h^{t-1} + b^t)$

t is layer

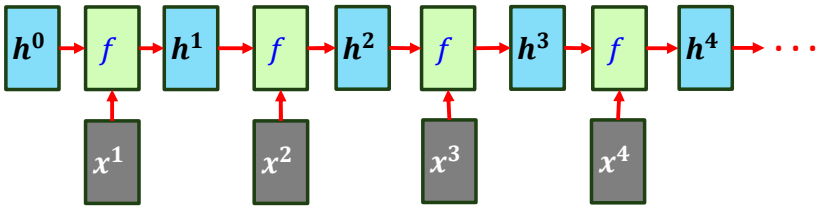


$h^t = f(h^{t-1}, x^t)$
$= \sigma(Wh^{t-1} + Ux^t + b)$

t is time step

Weight sharing in Recurrent Network

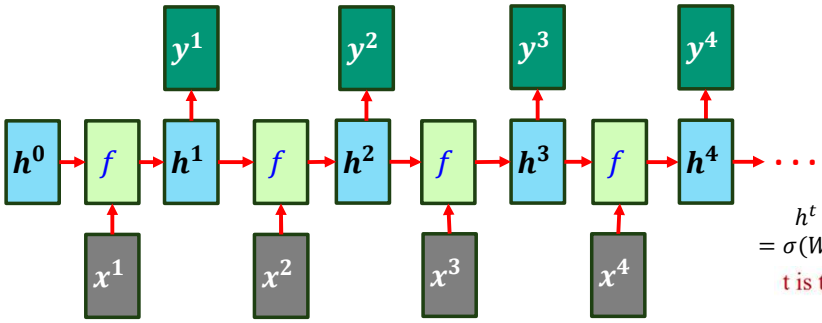
9



$$h^1 = \varphi(Wh^0 + Ux^1 + b)$$
$$h^2 = \varphi\big(Wh^1 + Ux^2 + b\big) = \varphi\big(W\varphi(Wh^0 + Ux^1 + b) + Ux^2 + b\big)$$
$$h^3 = \varphi\big(Wh^2 + Ux^3 + b\big) =$$
$$\varphi\big(W\varphi\big(W\varphi(Wh^0 + Ux^1 + b) + Ux^2 + b\big) + Ux^3 + b\big)$$
$$\cdots$$

Recurrent Neural Network (RNN)

10



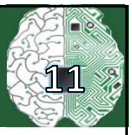
$$y^t = g(h^t)$$

$$h^t = f(h^{t-1}, x^t)$$
$$= \sigma(Wh^{t-1} + Ux^t + b)$$

t is time step

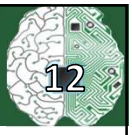
- No matter how long the input/output sequence is, we only need one function f
  - If f's are different then it becomes a feedforward NN
  - This may be treated as another compression from fully connected network

RNN



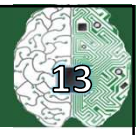
- RNNs are a family of neural networks for processing **sequential data**
- Recurrent networks can **scale to much longer sequences** than would be practical for networks without sequence-based specialization.
- Most recurrent networks can also process sequences of **variable length**.
- Based on **parameter sharing**
  - If we had separate parameters for each value of the time index,
    - we could not generalize to sequence lengths not seen during training,
    - nor share statistical strength across different sequence lengths,
    - and across different positions in time.

RNN vs. 1D Convolution



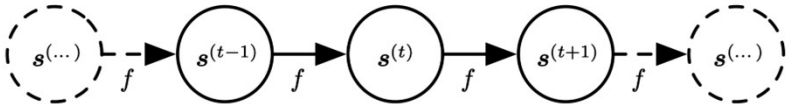
- The output of convolution is a sequence where each member of the output is a function of a small number of neighboring members of the input.
- Recurrent networks share parameters in a different way.
  - Each member of the output is a function of the previous members of the output
  - Each member of the output is produced using the same update rule applied to the previous outputs.
  - This recurrent formulation results in the sharing of parameters through a **very deep computational graph**.

Computational Graph for a Classical Dynamical System



$$s^{(t)} = f(s^{(t-1)}; \theta)$$
$$s^{(3)} = f(s^{(2)}; \theta) = f(f(s^{(1)}; \theta); \theta)$$

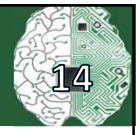
$s^{(t)}$: state of the system



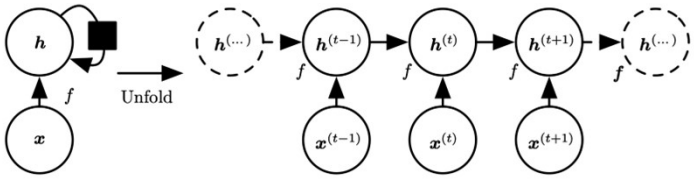
Other example: consider a dynamical system driven by an external signal $x^{(t)}$:

$$s^{(t)} = f(s^{(t-1)}, x^{(t)}; \theta)$$

Unfolding Computational Graphs



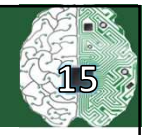
$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta)$$
$$= g^{(t)}(x^{(t)}, x^{(t-1)}, x^{(t-2)}, \dots, x^{(2)}, x^{(1)})$$

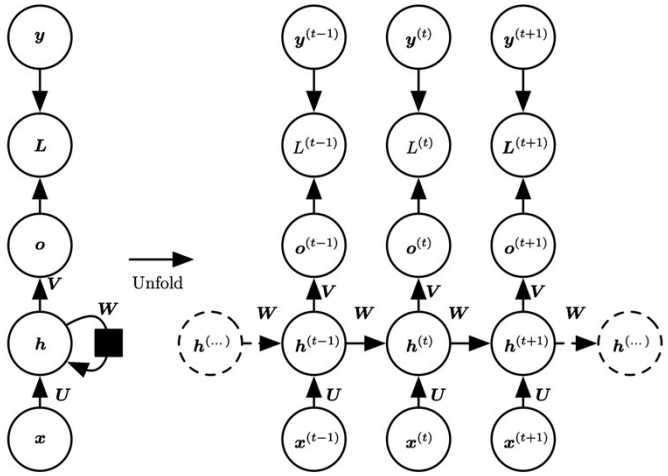


The network typically learns to use $h^{(t)}$ as a kind of lossy summary of the task-relevant aspects of the past sequence of inputs up to t

This summary is necessarily lossy, since it maps an arbitrary length sequence $x^t, x^{t-1}, x^{t-2}, \dots, x^2, x^1$ to a fixed length vector h^t

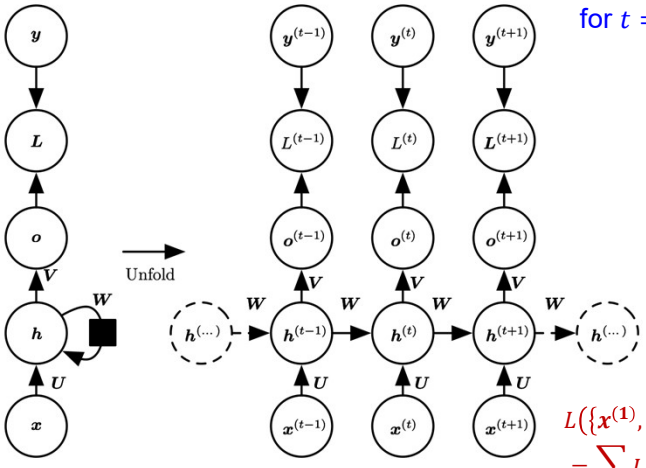
Recurrent Hidden Units





U: input-to-hidden
V: hidden-to-output
W: hidden-to-hidden

Forward Propagation



for $t = 1, \dots, \tau$:

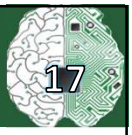
$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$$
$$h^{(t)} = \tanh\left(a^{(t)}\right)$$
$$o(t) = c + V\,h(t)$$
$$\hat{y}^{(t)} = \text{softmax}\left(o^{(t)}\right)$$

Need Backpropagation through time (BPTT)

Computationally expensive!

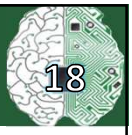
$$L\left(\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(\tau)}\}, \{y^{(1)}, y^{(2)}, \dots, y^{(\tau)}\}\right)$$
$$= \sum_t L(t) = -\sum_t \log p_{model}(y^{(t)} | \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(t)}\})$$

Computing the gradient



- $L^{(t)} = -\sum p_i \log \quad softmax\left(o_i^{(t)}\right) = -\sum p_i \log \hat{y}_i^{(t)}$
- $\frac{\partial L^{(t)}}{\partial o_i^{(t)}} = -\frac{\partial}{\partial o_i^{(t)}}\left(\sum_{j \neq i} p_j \log \hat{y}_j^{(t)} + p_i \log \hat{y}_i^{(t)}\right)$
  - $\frac{\partial}{\partial o_i^{(t)}} \log \hat{y}_i^{(t)} = 1 - \hat{y}_i^{(t)}$
  - $\frac{\partial}{\partial o_i^{(t)}} \sum_{j \neq i} p_j \log \hat{y}_j^{(t)} = \frac{\partial}{\partial o_i^{(t)}}\left(\sum_{j \neq i} p_j o_j^{(t)} - \sum_{j \neq i} p_j \log \sum_k \exp\left(o_k^{(t)}\right)\right) =$
    $-\sum_{j \neq i} p_j \, \hat{y}_i^{(t)} = -\hat{y}_i^{(t)} \sum_{j \neq i} p_j = -\hat{y}_i^{(t)}(1 - p_i)$
- $\frac{\partial L^{(t)}}{\partial o_i^{(t)}} = -p_i\left(1 - \hat{y}_i^{(t)}\right) + (1 - p_i)\hat{y}_i^{(t)} = -p_i + \hat{y}_i^{(t)}$
- $\frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - \mathbf{1}_{i,y^{(t)}}$ $\qquad \nabla_{\boldsymbol{o}^{(t)}} L^{(t)} = \hat{\boldsymbol{y}}^{(t)} - \mathbf{1}_{y^{(t)}}$

Computing the gradient

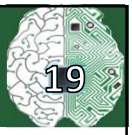


for $t = 1, \ldots, \tau$:

$a^{(t)} = b + W h^{(t-1)} + U x^{(t)}$
$h^{(t)} = \tanh\left(a^{(t)}\right)$
$o^{(t)} = c + V h^{(t)}$
$\hat{y}^{(t)} = \text{softmax}\left(o^{(t)}\right)$

- $\nabla_{\boldsymbol{h}^{(\tau)}} L = V^T \nabla_{\boldsymbol{o}^{(\tau)}} L$
- $\nabla_{\boldsymbol{a}^{(\tau)}} L = \left(\frac{\partial \boldsymbol{h}^{(\tau)}}{\partial \boldsymbol{a}^{(\tau)}}\right)^T \nabla_{\boldsymbol{h}^{(\tau)}} L = diag\left(1 - h^{(\tau)^2}\right) \nabla_{\boldsymbol{h}^{(\tau)}} L$
- $\nabla_{\boldsymbol{h}^{(\tau-1)}} L = W^T \nabla_{\boldsymbol{a}^{(\tau)}} L + V^T \nabla_{\boldsymbol{o}^{(\tau-1)}} L$
  $\nabla_{\boldsymbol{h}^{(\tau-1)}} L = W^T diag\left(1 - h^{(\tau)^2}\right) \nabla_{\boldsymbol{h}^{(\tau)}} L + V^T \nabla_{\boldsymbol{o}^{(\tau-1)}} L$
- Valid for any $t < \tau$
  $\nabla_{\boldsymbol{h}^{(t)}} L = W^T diag\left(1 - h^{(t+1)^2}\right) \nabla_{\boldsymbol{h}^{(t+1)}} L + V^T \nabla_{\boldsymbol{o}^{(t)}} L$

Gradients on the parameter nodes

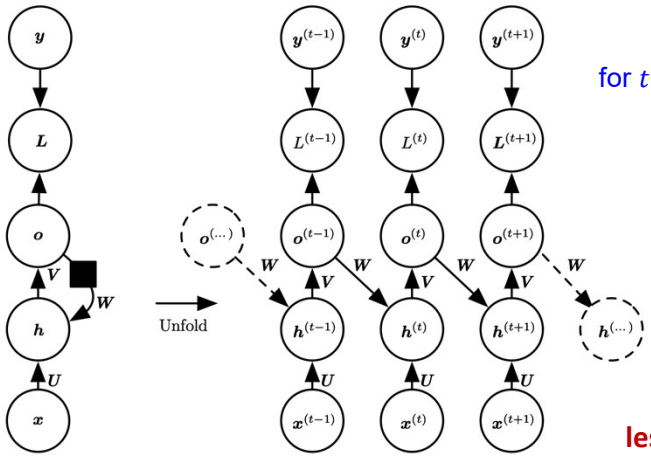


- $\nabla_c L = \sum_t \nabla_{o^{(t)}} L$
- $\nabla_V L = \sum_t \left(\nabla_{o^{(t)}} L\right) h^{(t)^T}$
- $\nabla_b L = \sum_t \nabla_{a^{(t)}} L = \sum_t diag\left(1 - h^{(t)^2}\right) \nabla_{\boldsymbol{h}^{(t)}} L$
- $\nabla_W L = \sum_t \left(\nabla_{a^{(t)}} L\right) h^{(t-1)^T} = \sum_t \left(diag\left(1 - h^{(t)^2}\right) \nabla_{\boldsymbol{h}^{(t)}} L\right) h^{(t-1)^T}$
- $\nabla_U L = \sum_t \left(\nabla_{a^{(t)}} L\right) x^{(t)^T} = \sum_t \left(diag\left(1 - h^{(t)^2}\right) \nabla_{\boldsymbol{h}^{(t)}} L\right) x^{(t)^T}$

for $t = 1, \ldots, \tau$:

$$a^{(t)} = b + W h^{(t-1)} + U x^{(t)}$$
$$h^{(t)} = \tanh\left(a^{(t)}\right)$$
$$o^{(t)} = c + V h^{(t)}$$
$$\hat{y}^{(t)} = \text{softmax}\left(o^{(t)}\right)$$

Recurrence through only the Output

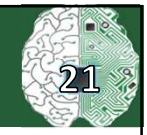


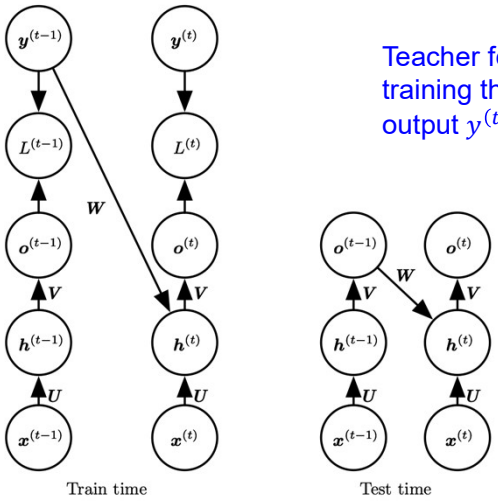
for $t = 1, \ldots, \tau$:

$$a^{(t)} = b + W o^{(t-1)} + U x^{(t)}$$
$$h^{(t)} = \tanh\left(a^{(t)}\right)$$
$$o(t) = c + V h(t)$$
$$\hat{y}^{(t)} = \text{softmax}\left(o^{(t)}\right)$$

less powerful, but easier to train!

Teacher Forcing



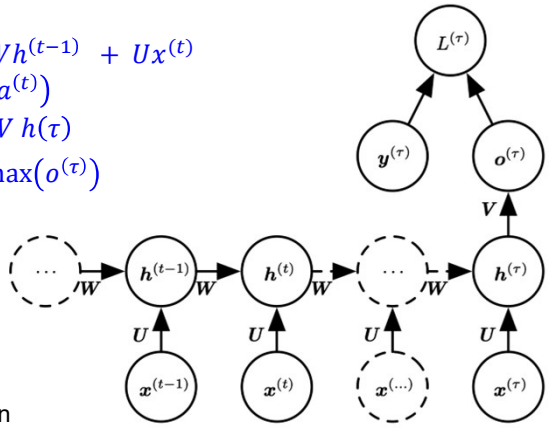


Teacher forcing is a procedure in which during training the model receives the ground truth output $y^{(t)}$ as input at time $t + 1$.

Sequence Input, Single Output

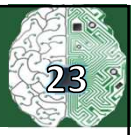
for $t = 1, \dots, \tau$:

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$$
$$h^{(t)} = \tanh\left(a^{(t)}\right)$$
$$o(\tau) = c + V\,h(\tau)$$
$$\hat{y}^{(\tau)} = \text{softmax}\left(o^{(\tau)}\right)$$

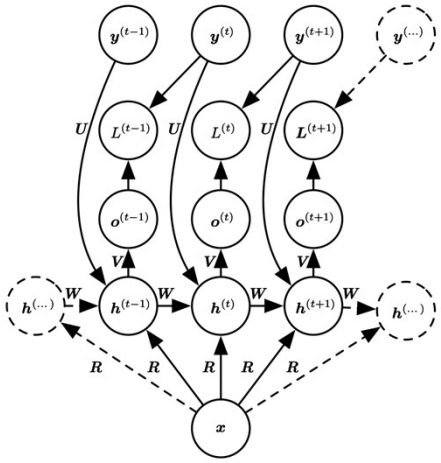


U: input-to-hidden
V: hidden-to-output
W: hidden-to-hidden

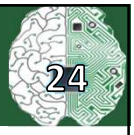
Vector to Sequence



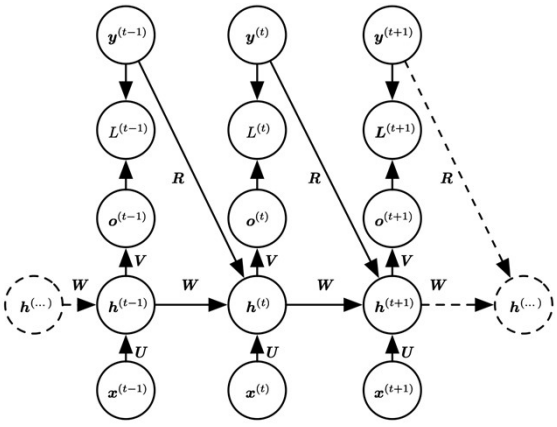
- An RNN that maps a fixed-length vector x into a distribution over sequences Y.
- Each element $y^{(t)}$ of the observed output sequence serves both as input (for the current time step) and, during training, as target (for the previous time step).



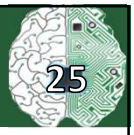
Hidden and Output Recurrence



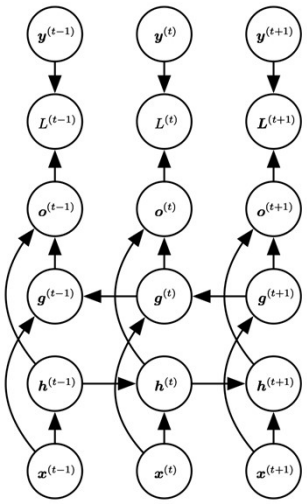
- A conditional recurrent neural network mapping a variable-length sequence of x values into a distribution over sequences of y values of the same length.
- This RNN contains connections from the previous output to the current state.
- These connections allow this RNN to model an arbitrary distribution over sequences of y given sequences of x of the same length.



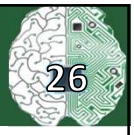
Bidirectional RNN



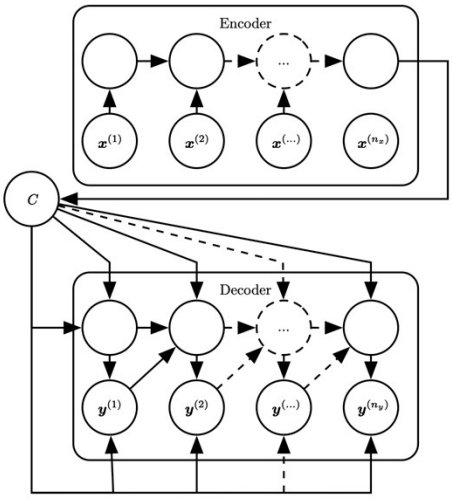
- Computation of a typical bidirectional recurrent neural network, meant to learn to map input sequences x to target sequences y, with loss $L^{(t)}$ at each step t.
- Thus at each point t, the output units $o^{(t)}$ can benefit from a relevant summary of the past in its $h^{(t)}$ input and from a relevant summary of the future in its $g^{(t)}$ input.



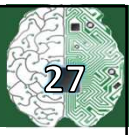
Sequence to Sequence Architecture



- RNN can be trained to map an input sequence to an output sequence which is not necessarily of the same length.
- This comes up in many applications

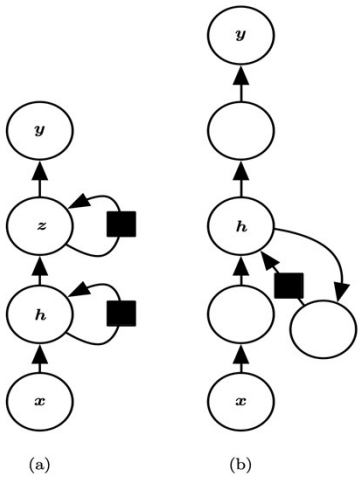


Deep RNNs

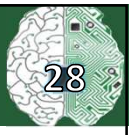


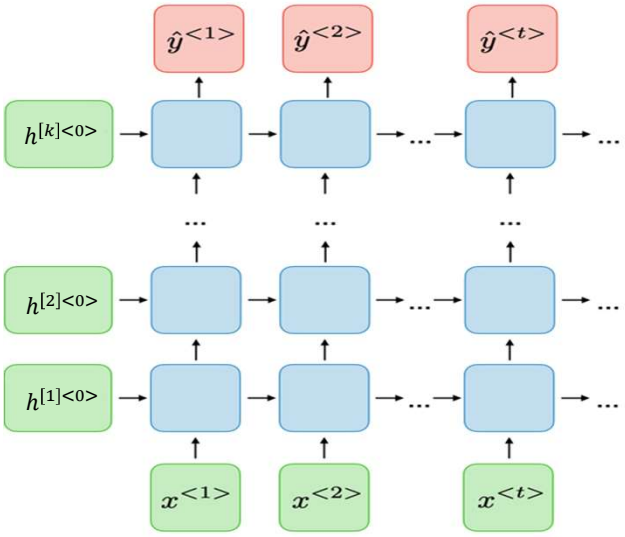
A recurrent neural network can be made deep in many ways.

- (a) The hidden recurrent state can be broken down into groups organized hierarchically.
- (b) Deeper computation (e.g., an MLP) can be introduced in the input-to-hidden, hidden-to-hidden and hidden-to-output parts. This may lengthen the shortest path linking different time steps.



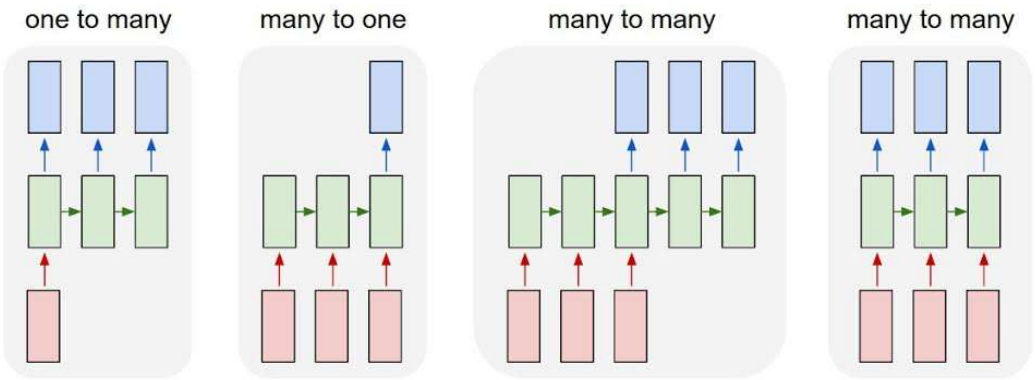
A Typical Architecture (unfolded CG) of Deep RNN





Applications of RNN Architectures: Summary

29



- Image Captioning
- Video Activity Recognition
- Text Classification
- Video Captioning
- Machine Translation
- POS Tagging
- Language Modeling

30

Questions?