

# CS6501: Deep Learning for Visual Recognition

## Recurrent Neural Networks (RNNs)



# Today's Class

- Recurrent Neural Network Cell
- Recurrent Neural Networks (RNNs)
- Bi-Directional Recurrent Neural Networks (Bi-RNNs)
- Multiple-layer / Stacked / Deep Bi-Direction Recurrent Neural Networks
- LSTMs and GRUs.
- Applications in Vision: Caption Generation.

# ACM Turing Award 2019

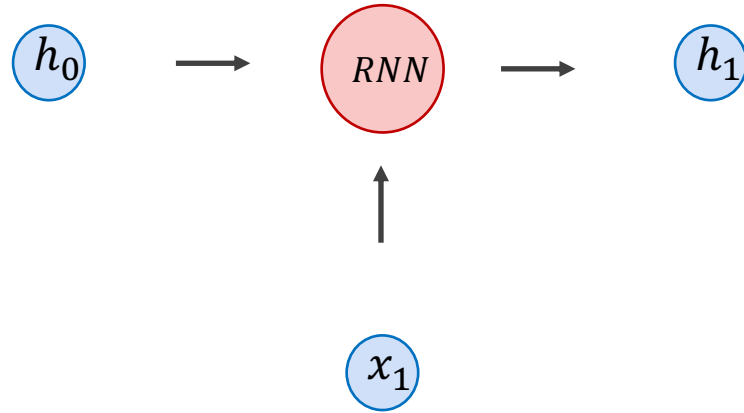


Yann LeCun  
CNNs

Geoff Hinton  
Backpropagation

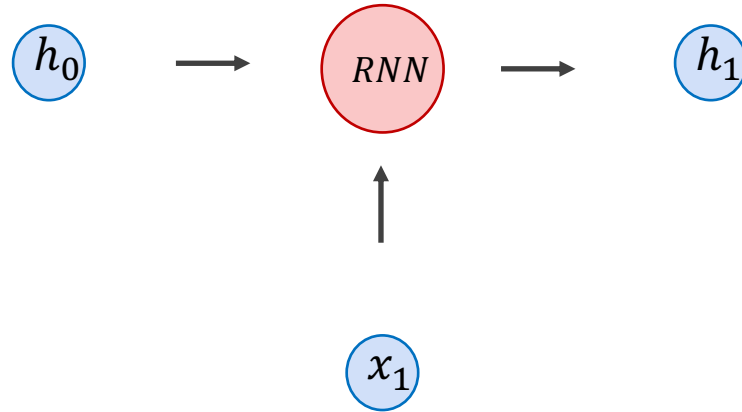
Yoshua Bengio  
GANs

# Recurrent Neural Network Cell

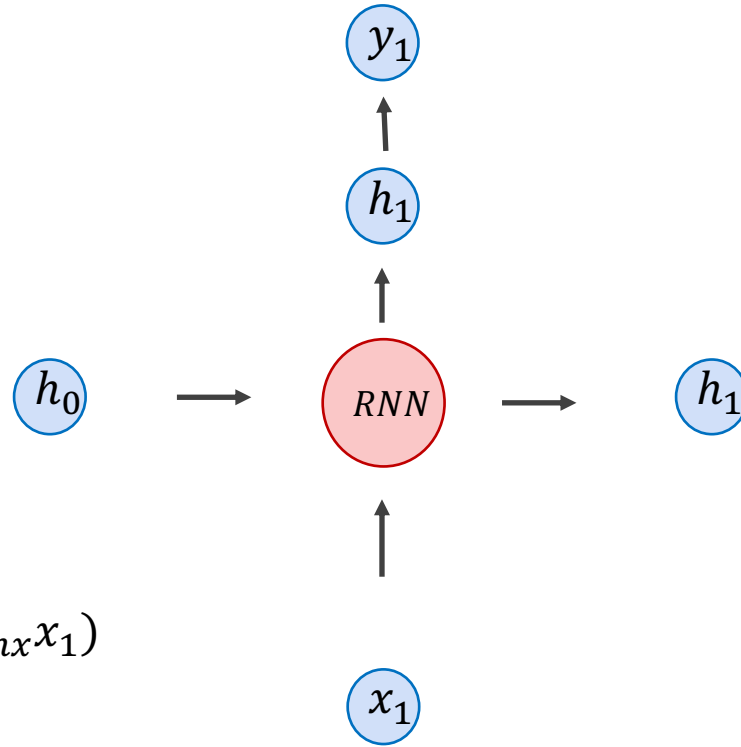


# Recurrent Neural Network Cell

$$h_1 = \tanh(W_{hh}h_0 + W_{hx}x_1)$$



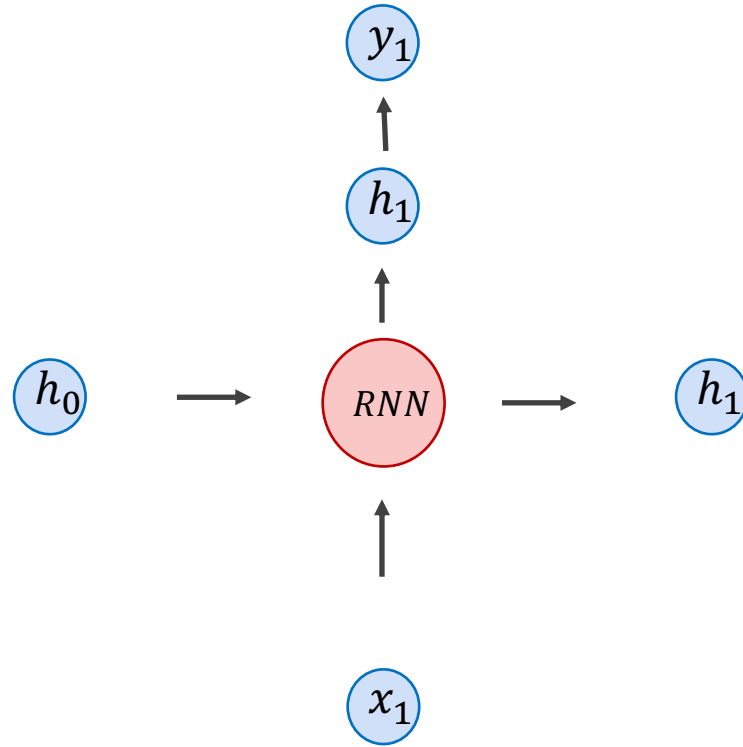
# Recurrent Neural Network Cell



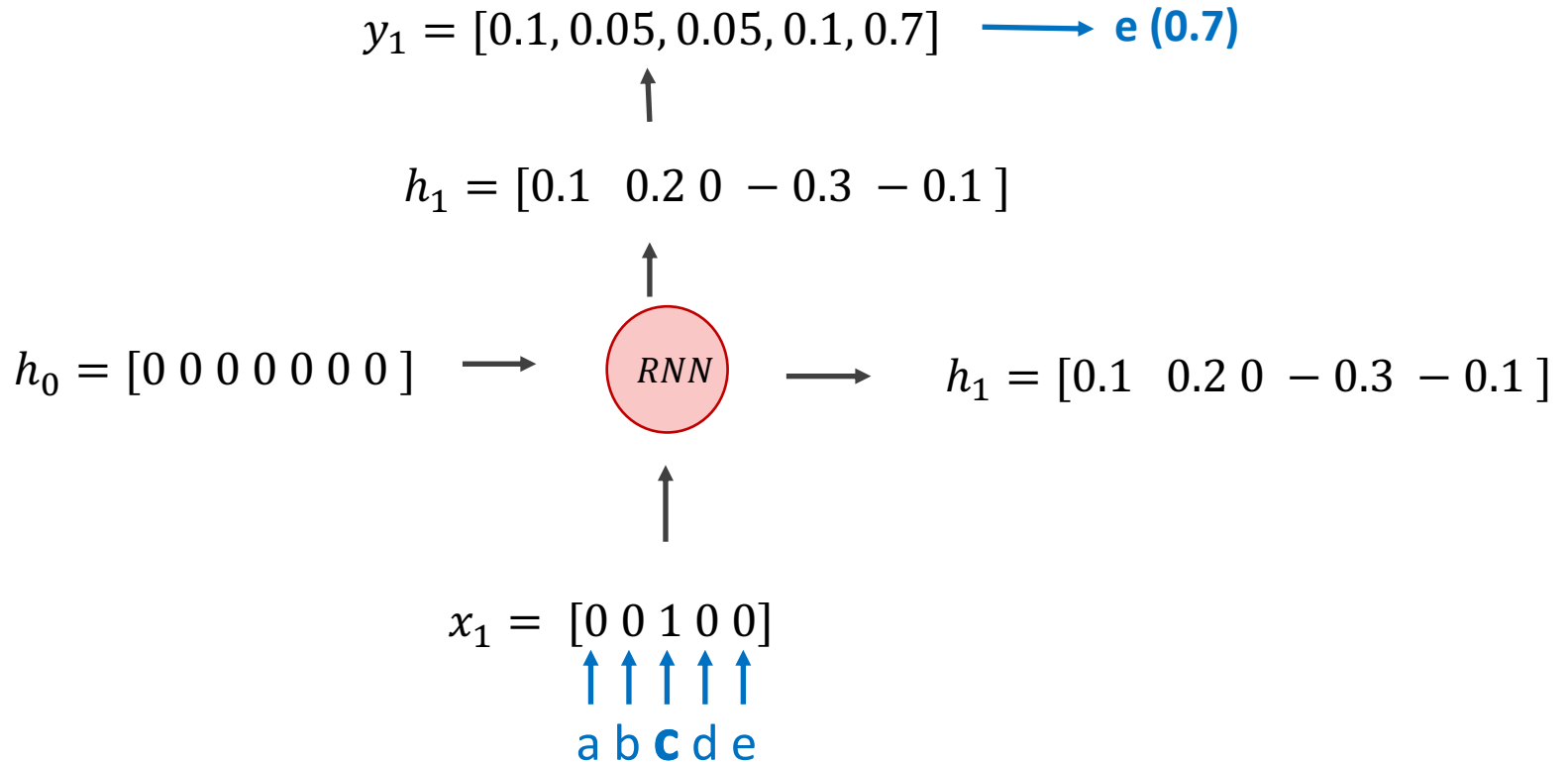
$$h_1 = \tanh(W_{hh}h_0 + W_{hx}x_1)$$

$$y_1 = \text{softmax}(W_{hy}h_1)$$

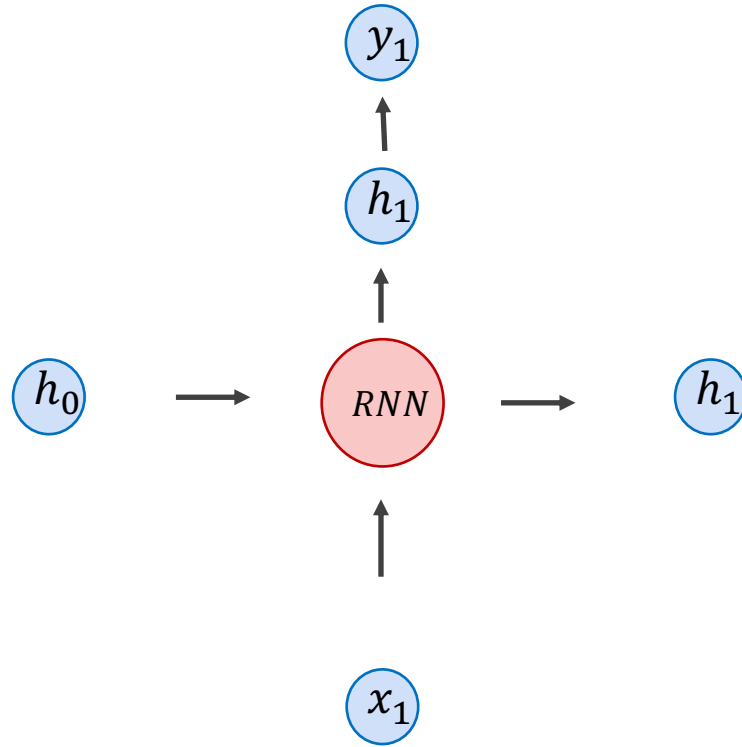
# Recurrent Neural Network Cell



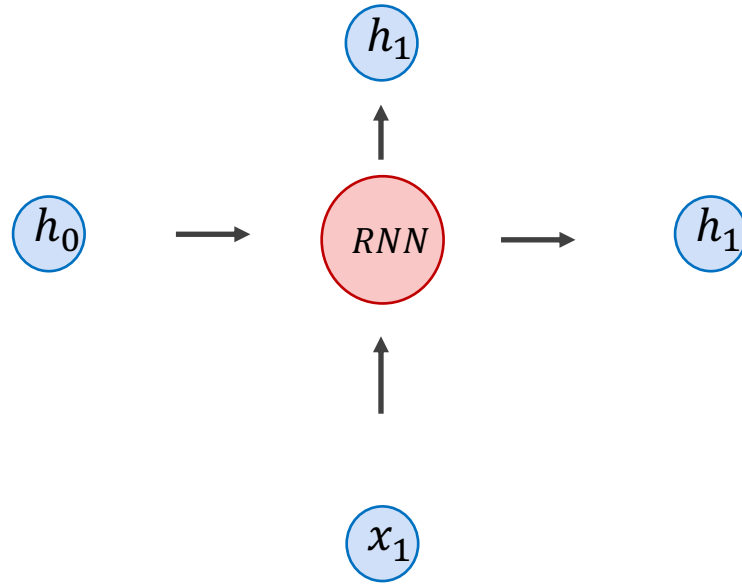
# Recurrent Neural Network Cell



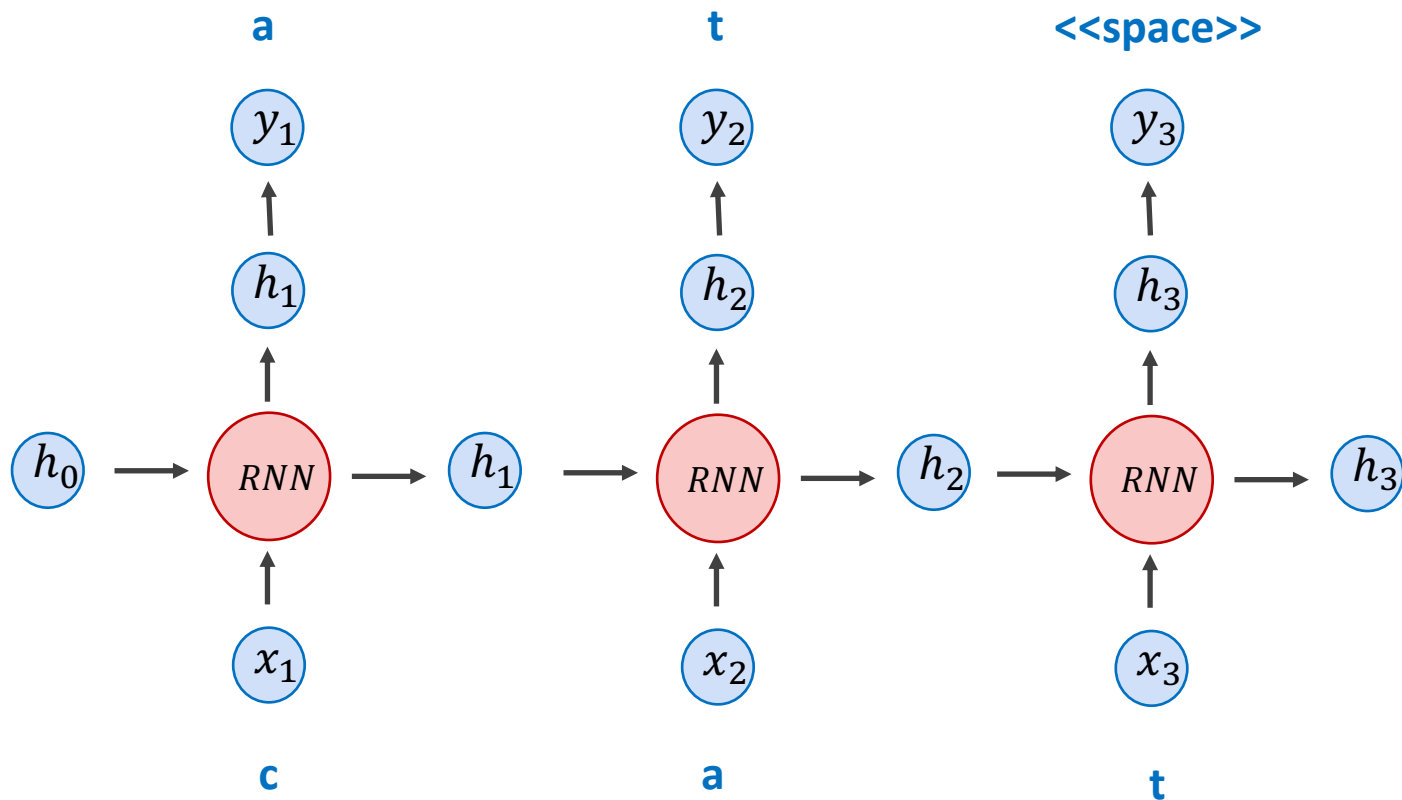
# Recurrent Neural Network Cell



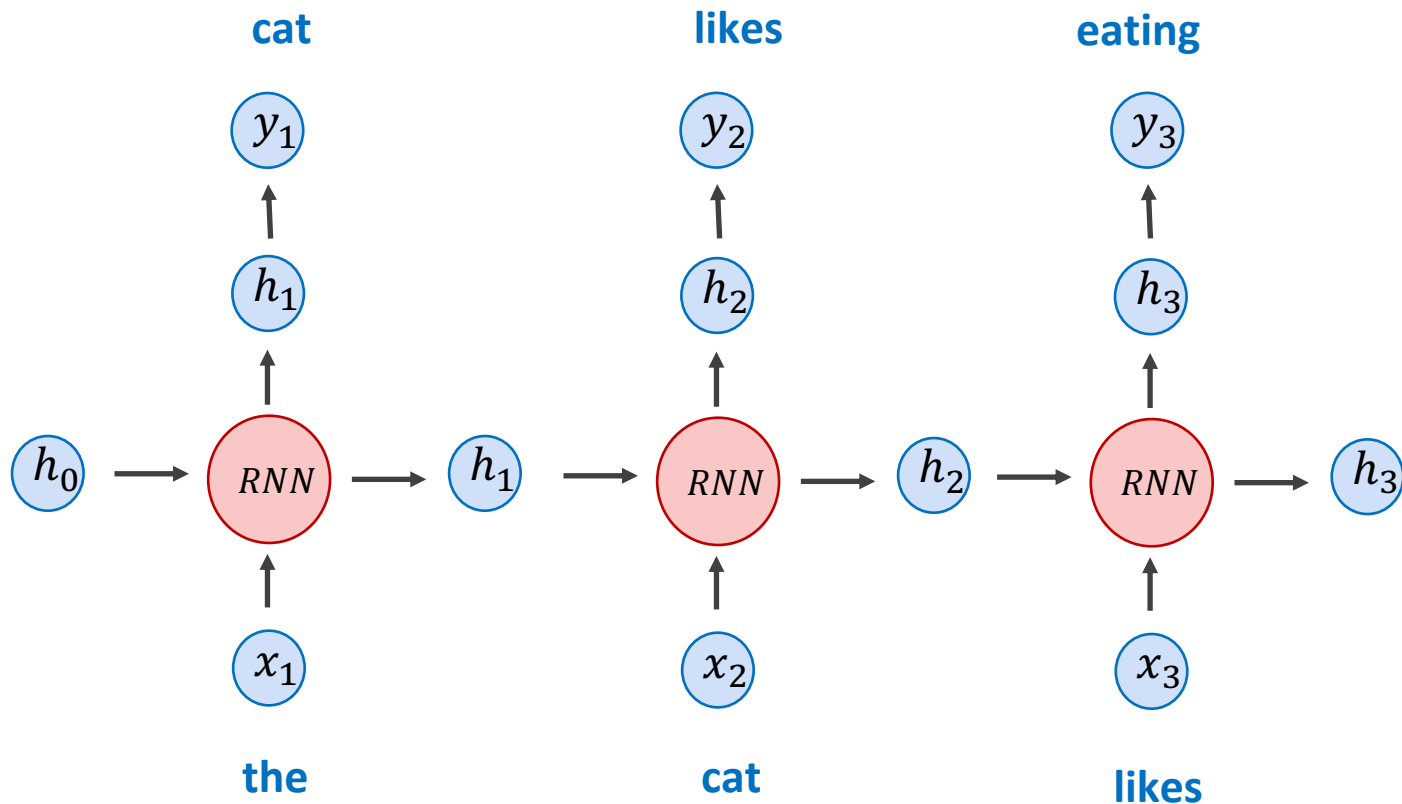
# Recurrent Neural Network Cell



# (Unrolled) Recurrent Neural Network

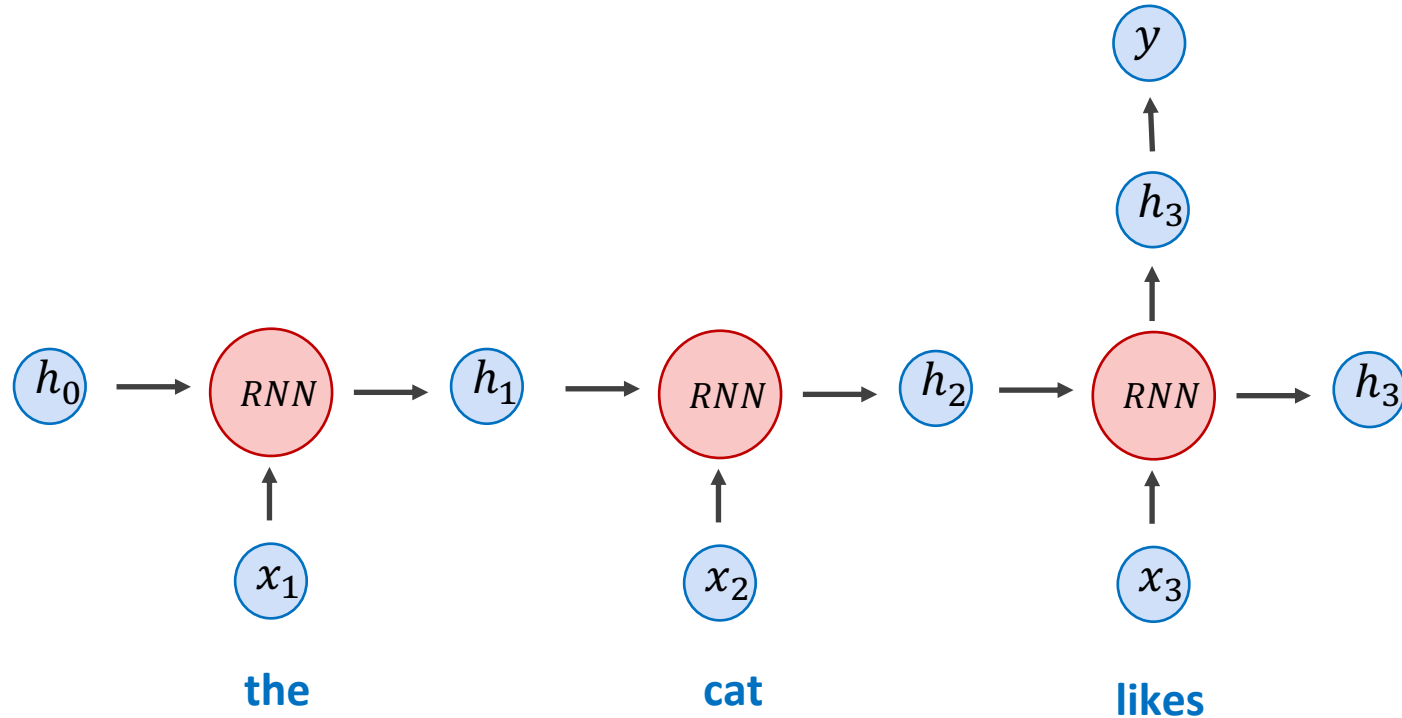


# (Unrolled) Recurrent Neural Network

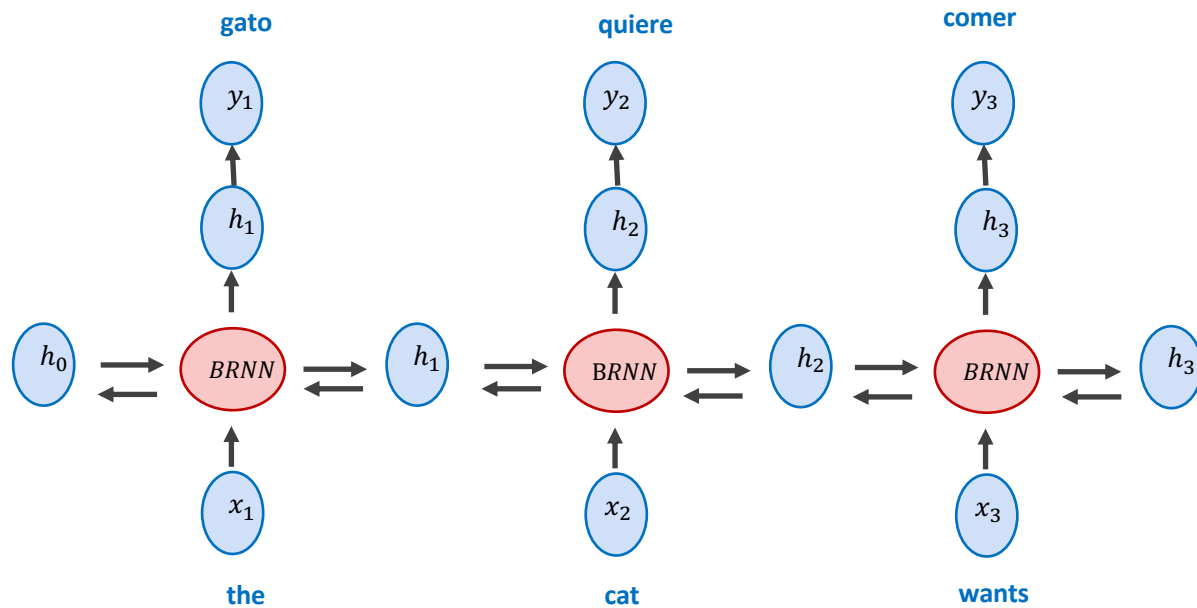


# (Unrolled) Recurrent Neural Network

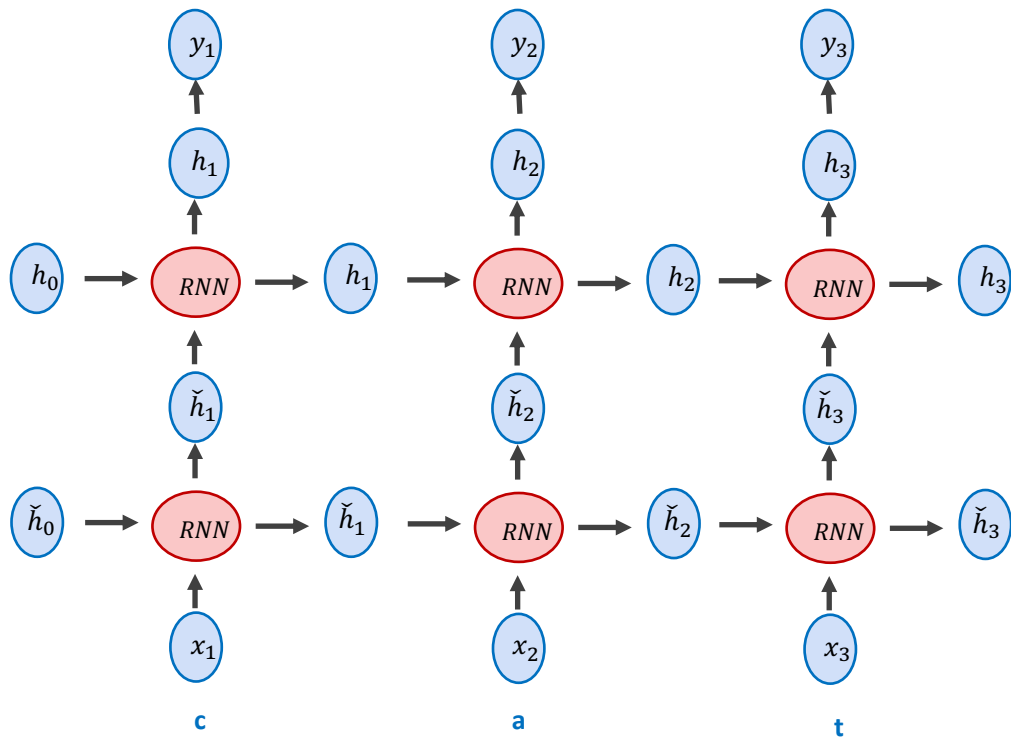
positive / negative sentiment rating



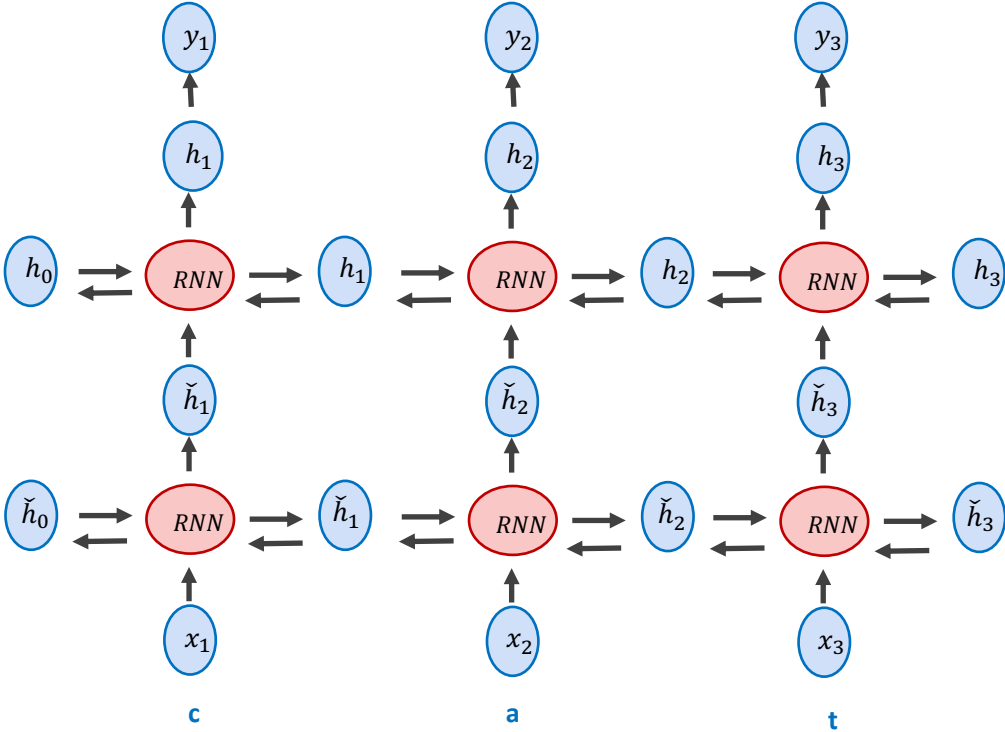
# Bidirectional Recurrent Neural Network



# Stacked Recurrent Neural Network



# Stacked Bidirectional Recurrent Neural Network



# RNN in Pytorch

## Recurrent layers

```
class torch.nn.RNN(*args, **kwargs) \[source\]
```

Applies a multi-layer Elman RNN with tanh or ReLU non-linearity to an input sequence.

For each element in the input sequence, each layer computes the following function:

$$h_t = \tanh(w_{ih} * x_t + b_{ih} + w_{hh} * h_{(t-1)} + b_{hh})$$

where  $h_t$  is the hidden state at time  $t$ , and  $x_t$  is the hidden state of the previous layer at time  $t$  or  $input_t$  for the first layer. If nonlinearity='relu', then *ReLU* is used instead of *tanh*.

- Parameters:
- **input\_size** – The number of expected features in the input  $x$
  - **hidden\_size** – The number of features in the hidden state  $h$
  - **num\_layers** – Number of recurrent layers.
  - **nonlinearity** – The non-linearity to use ['tanh'|'relu']. Default: 'tanh'
  - **bias** – If False, then the layer does not use bias weights  $b_{ih}$  and  $b_{hh}$ . Default: True
  - **batch\_first** – If True, then the input and output tensors are provided as (batch, seq, feature)
  - **dropout** – If non-zero, introduces a dropout layer on the outputs of each RNN layer except the last layer
  - **bidirectional** – If True, becomes a bidirectional RNN. Default: False

# LSTM Cell (Long Short-Term Memory)

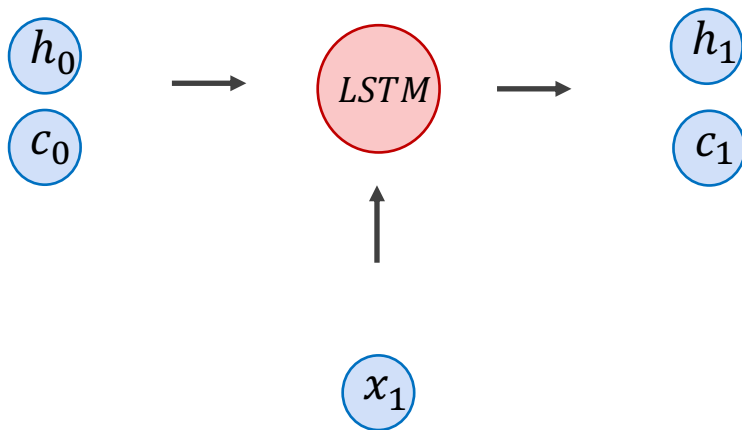
$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \quad (7)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \quad (8)$$

$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad (9)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \quad (10)$$

$$h_t = o_t \tanh(c_t) \quad (11)$$



# LSTM in Pytorch

```
class torch.nn.LSTM(*args, **kwargs) \[source\]
```

Applies a multi-layer long short-term memory (LSTM) RNN to an input sequence.

For each element in the input sequence, each layer computes the following function:

$$i_t = \text{sigmoid}(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi})$$

$$f_t = \text{sigmoid}(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf})$$

$$g_t = \text{tanh}(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg})$$

$$o_t = \text{sigmoid}(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho})$$

$$c_t = f_t * c_{(t-1)} + i_t * g_t$$

$$h_t = o_t * \text{tanh}(c_t)$$

where  $h_t$  is the hidden state at time  $t$ ,  $c_t$  is the cell state at time  $t$ ,  $x_t$  is the hidden state of the previous layer at time  $t$  or  $input_t$ , for the first layer, and  $i_t, f_t, g_t, o_t$  are the input, forget, cell, and out gates, respectively.

- Parameters:**
- **input\_size** - The number of expected features in the input x
  - **hidden\_size** - The number of features in the hidden state h
  - **num\_layers** - Number of recurrent layers.
  - **bias** - If False, then the layer does not use bias weights b\_ih and b\_hh. Default: True
  - **batch\_first** - If True, then the input and output tensors are provided as (batch, seq, feature)
  - **dropout** - If non-zero, introduces a dropout layer on the outputs of each RNN layer except the last layer
  - **bidirectional** - If True, becomes a bidirectional RNN. Default: False

# GRU in Pytorch

```
class torch.nn.GRU(*args, **kwargs) \[source\]
```

Applies a multi-layer gated recurrent unit (GRU) RNN to an input sequence.

For each element in the input sequence, each layer computes the following function:

$$\begin{aligned}r_t &= \text{sigmoid}(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \\z_t &= \text{sigmoid}(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz}) \\n_t &= \text{tanh}(W_{in}x_t + b_{in} + r_t * (W_{hn}h_{(t-1)} + b_{hn})) \\h_t &= (1 - z_t) * n_t + z_t * h_{(t-1)}\end{aligned}$$

where  $h_t$  is the hidden state at time  $t$ ,  $x_t$  is the hidden state of the previous layer at time  $t$  or  $input_t$  for the first layer, and  $r_t, z_t, n_t$  are the reset, input, and new gates, respectively.

- Parameters:**
- **input\_size** – The number of expected features in the input  $x$
  - **hidden\_size** – The number of features in the hidden state  $h$
  - **num\_layers** – Number of recurrent layers.
  - **bias** – If False, then the layer does not use bias weights  $b_{ih}$  and  $b_{hh}$ . Default: True
  - **batch\_first** – If True, then the input and output tensors are provided as (batch, seq, feature)
  - **dropout** – If non-zero, introduces a dropout layer on the outputs of each RNN layer except the last layer
  - **bidirectional** – If True, becomes a bidirectional RNN. Default: False

Questions?