5. Sequence Modeling: Recurrent and Recursive Nets

Lecture slides by
Prof. Sujata Pathak, IT, KJSCE

RNN

- Family of neural networks for processing sequential data.
- Recurrent neural network is a neural network that is specialized for processing a sequence of values $x(1), \ldots, x(\tau)$.

Motivation: Need for Sequential Modeling

Examples of Sequence data	Input Data	Output
Speech Recognition		*This is RNN*
Machine Translation	Hello, I am Pankaj.	Hallo, ich bin Pankaj. हैलो, मैं पंकज हूं।
Language Modeling 	Recurrent neural ? based ? model	network language
Named Entity Recognition	Pankaj lives in Munich	Pankaj lives in Munich *person* *location*
Sentiment Classification	There is nothing to like in this movie.	
Video Activity Analysis		Punching

Motivation: Need for Sequential Modeling

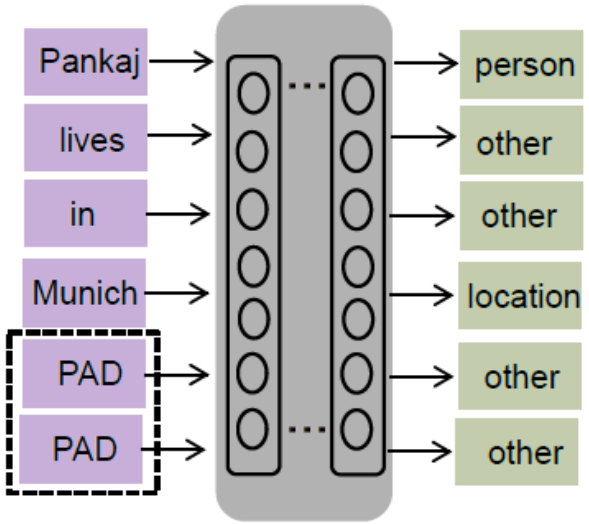
Inputs, Outputs can be different lengths in different examples

Example:

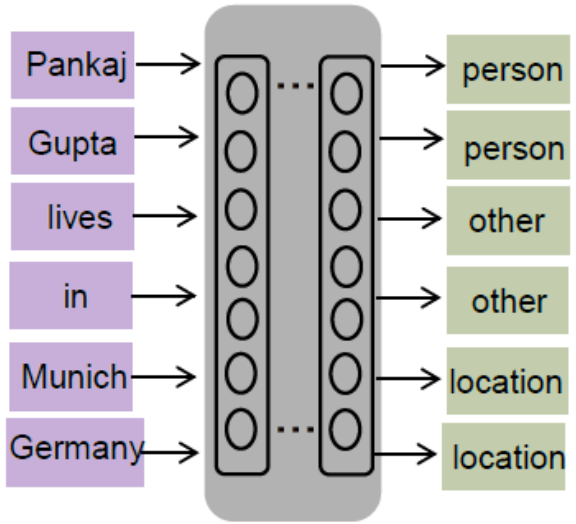
Sentence1: Pankaj lives in Munich

Sentence2: Pankaj Gupta lives in Munich DE

Additional word 'PAD' i.e., padding



FF-net / CNN



FF-net / CNN

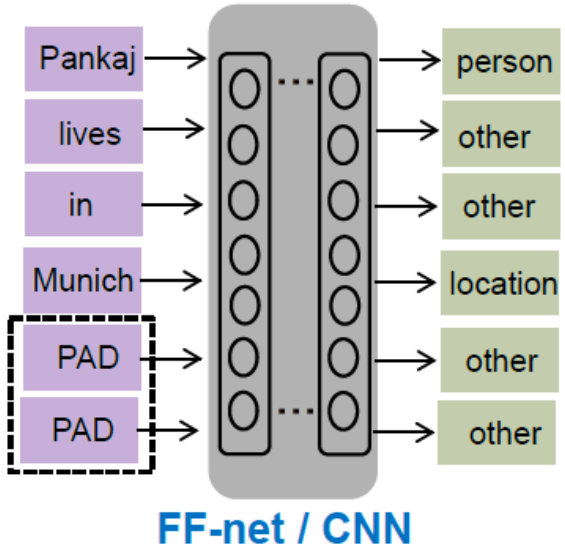
Motivation: Need for Sequential Modeling

Inputs, Outputs can be different lengths in different examples

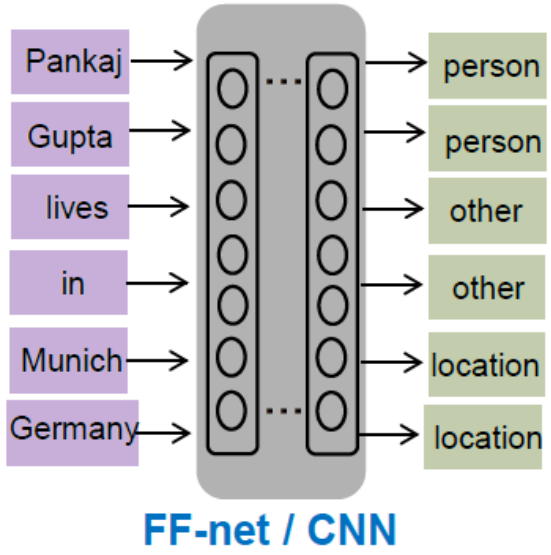
Example:

Sentence1: Pankaj lives in Munich

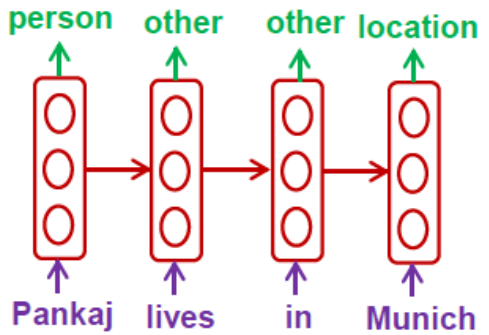
Sentence2: Pankaj Gupta lives in Munich DE



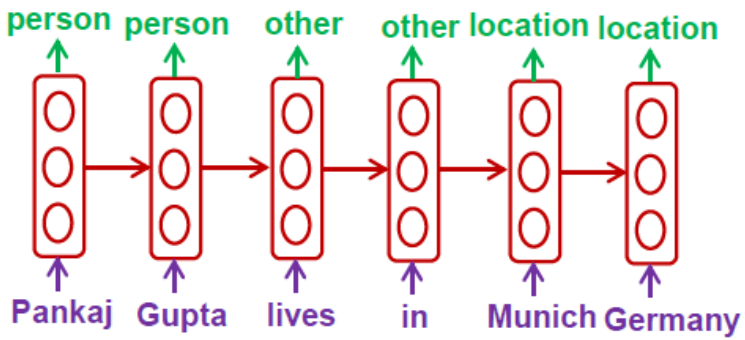
FF-net / CNN



FF-net / CNN



Models variable length sequences



Sequential model: RNN

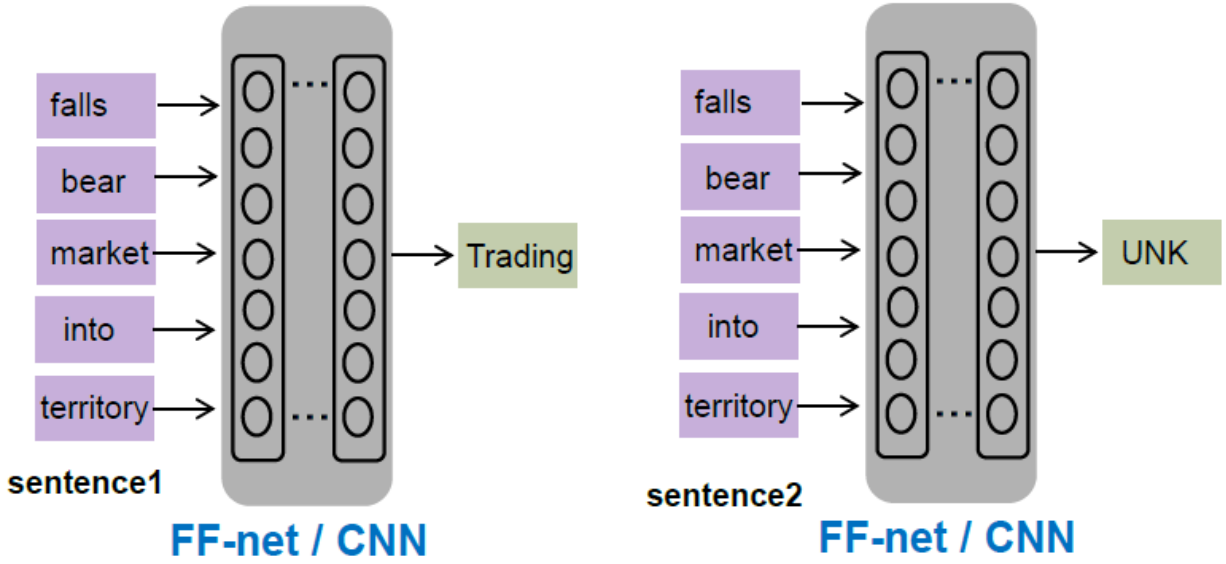
Motivation: Need for Sequential Modeling

Share Features learned across different positions or time steps

Example:

Sentence1: *Market falls into bear territory* → *Trading/Marketing*

Sentence2: Bear falls into market territory → *UNK*



No sequential or temporal modeling, i.e., order-less

Treats the two sentences the same

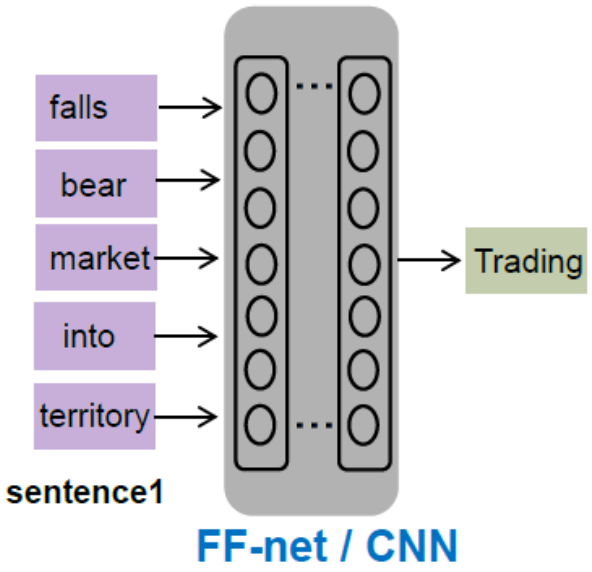
Motivation: Need for Sequential Modeling

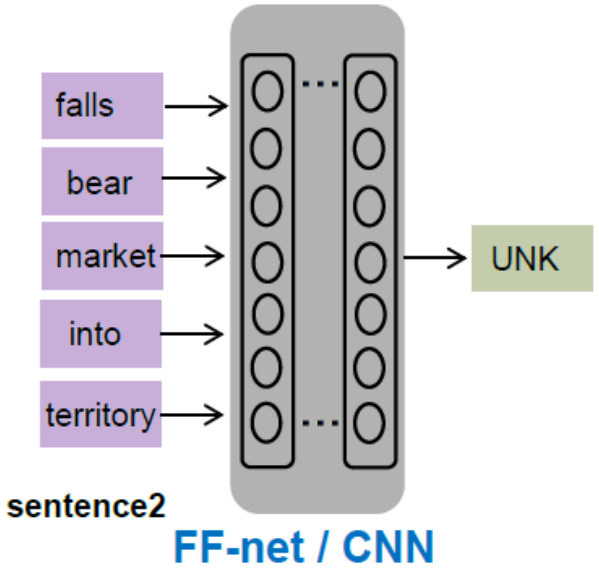
Share Features learned across different positions or time steps

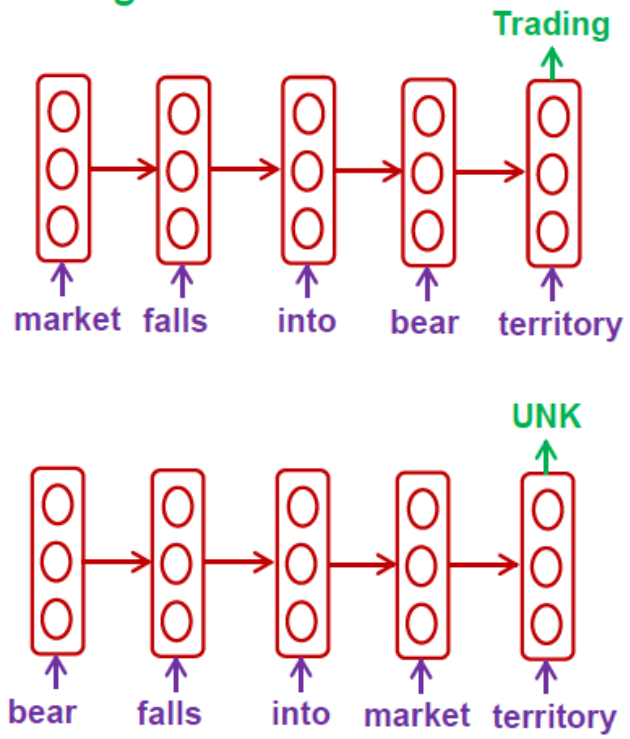
Example:

Sentence1: *Market falls into bear territory* → *Trading/Marketing*

Sentence2: Bear falls into market territory → *UNK*







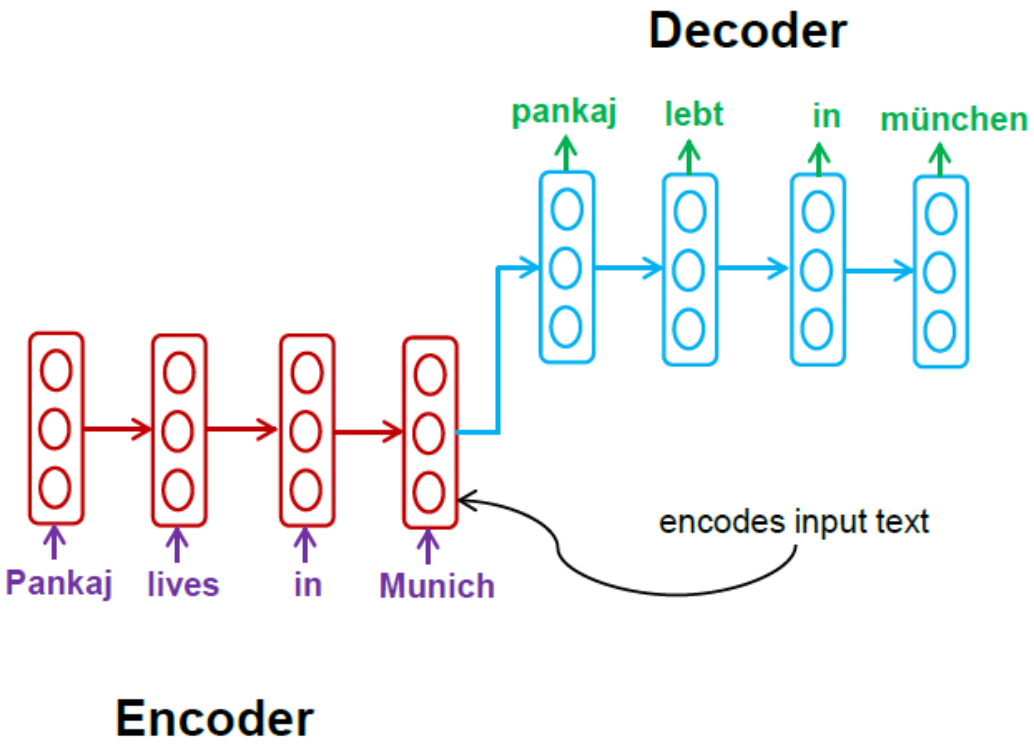
Language concepts,

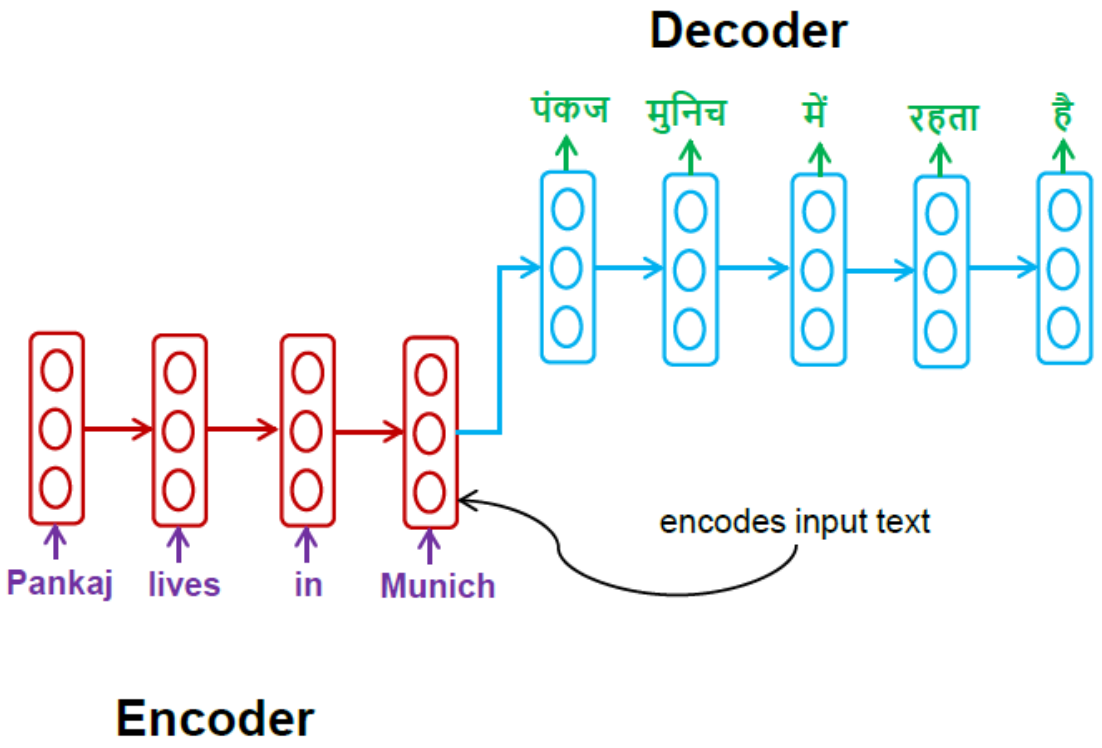
Word ordering,

Syntactic & semantic information

Motivation: Need for Sequential Modeling

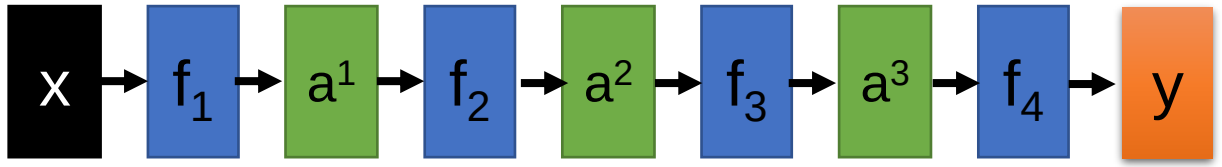
Machine Translation: Different Input and Output sizes, incurring sequential patterns





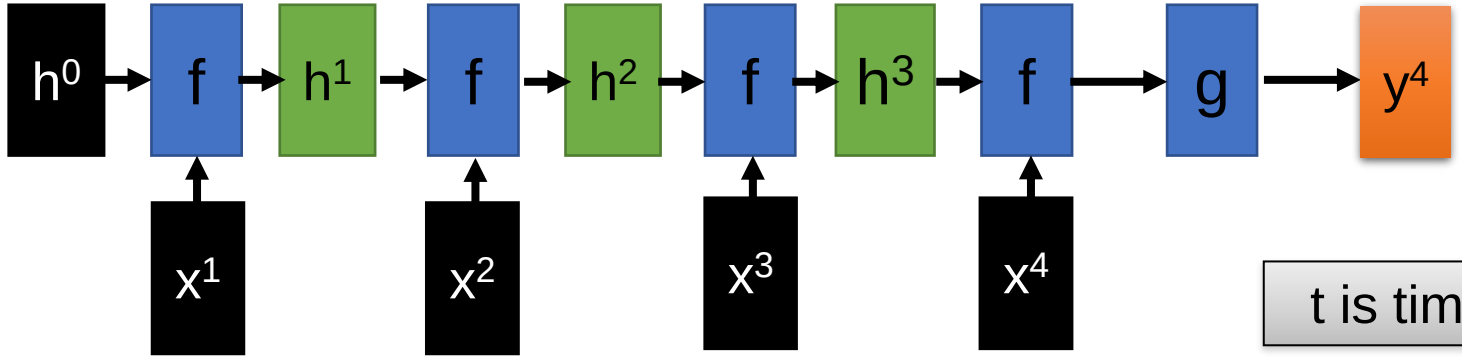
Feed-forward vs Recurrent Network

1. Feedforward network does not have input at each step
2. Feedforward network has different parameters for each layer



t is layer

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



t is time step

$a^t = f(a^{t-1}, x^t) = \sigma(W^h a^{t-1} + W^i x^t + b^i)$

We will turn the recurrent network 90 degrees.

Convolutional vs Recurrent Neural Networks

- RNN
  - Perform well when the input data is interdependent in a sequential pattern
  - Correlation between previous input to the next input
  - Introduce bias based on previous output
- CNN/FF-Nets
  - All the outputs are self-dependent
  - Feed-forward nets don’t remember historic input data at test time unlike recurrent networks.

Motivation: Need for Sequential Modeling

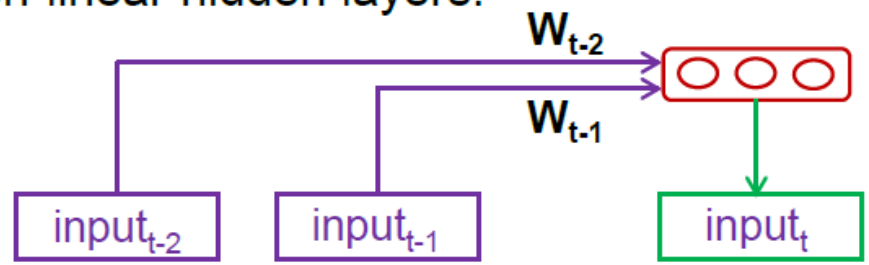
Memory-less Models

Autoregressive models:

Predict the next input in a sequence from a fixed number of previous inputs using "delay taps".

Feed-forward neural networks:

Generalize autoregressive models by using non-linear hidden layers.



Memory Networks

-possess a dynamic hidden state that can store long term information, e.g., RNNs.

Recurrent Neural Networks:

RNNs are very powerful, because they combine the following properties-

Distributed hidden state: can efficiently store a lot of information about the past.

Non-linear dynamics: can update their hidden state in complicated ways

Temporal and accumulative: can build semantics, e.g., word-by-word in sequence over time

Long Term and Short Dependencies

Short Term Dependencies

→ need recent information to perform the present task.

For example in a language model, predict the next word based on the previous ones.

"the clouds are in the ?" → *'sky'*

*"the clouds are in the **sky**"*

→ Easier to predict 'sky' given the context, i.e., *short term dependency*

Long Term Dependencies

→ Consider longer word sequence "I grew up in France…… I speak fluent ***French***."

→ Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back.

Foundation of Recurrent Neural Networks

Goal

- model long term dependencies
- connect previous information to the present task
- model sequence of events with loops, allowing information to persist

punching

Foundation of Recurrent Neural Networks Notations-

- $\boldsymbol{h}_t$: Hidden Unit
- $\boldsymbol{x}_t$: Input
- $\boldsymbol{o}_t$: Output
- $\boldsymbol{W}_{hh}$: Shared Weight Parameter
- $\boldsymbol{W}_{ho}$: Parameter weight between hidden layer and output
- θ: parameter in general
- g_θ : non linear function
- L_t :Loss between the RNN outputs and the true output
- E_t : cross entropy loss

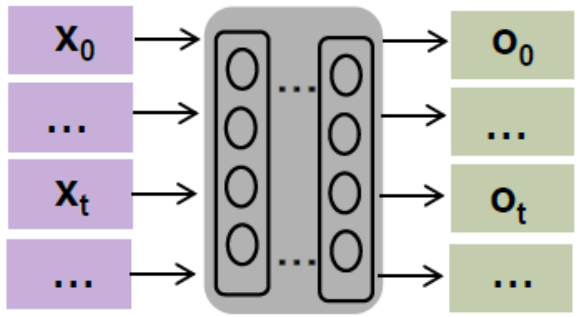
Foundation of Recurrent Neural Networks

Goal

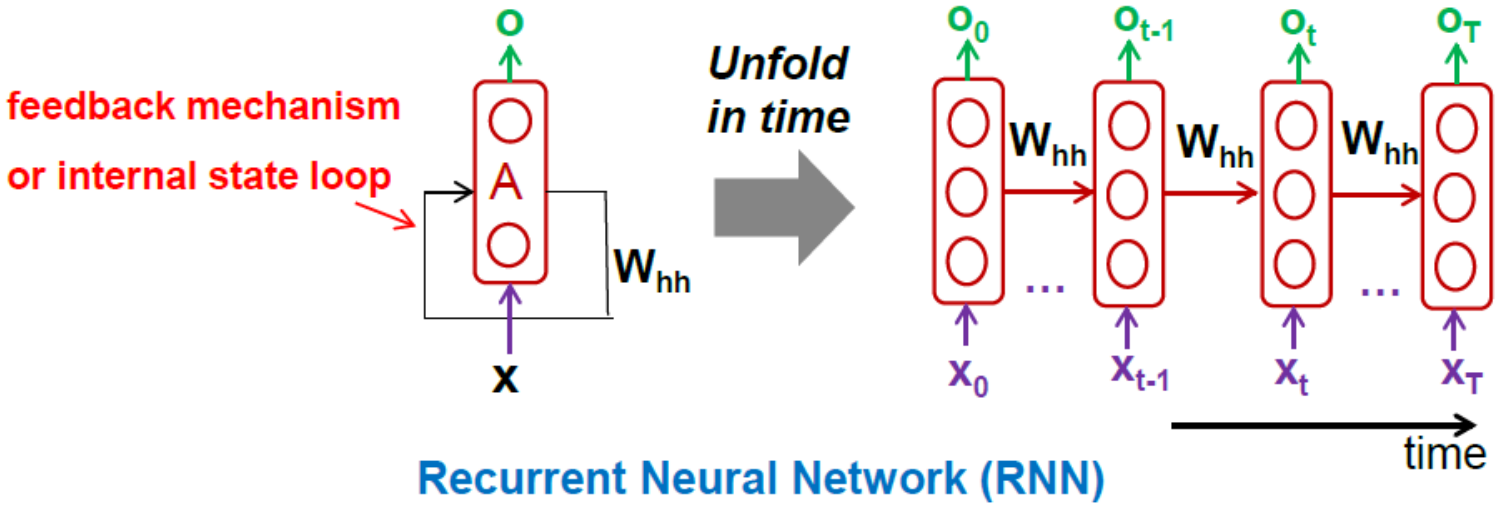
- model long term dependencies
- connect previous information to the present task
- model sequence of events with loops, allowing information to persist

Feed Forward NNets can **not** take **time dependencies** into account.

Sequential data needs a **Feedback Mechanism**.



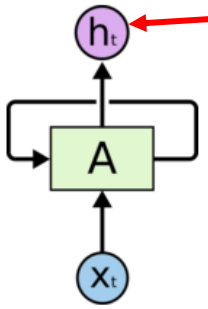
FF-net / CNN



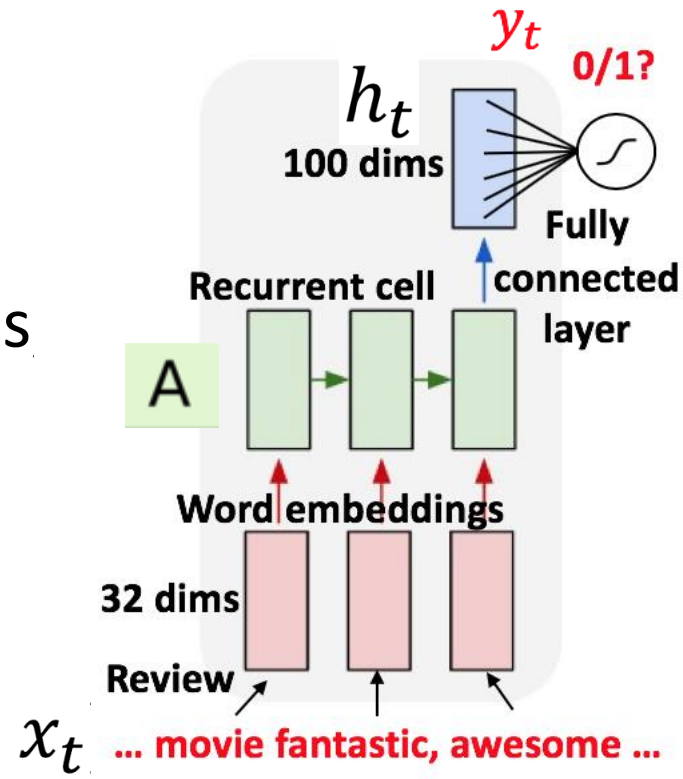
Recurrent Neural Network (RNN)

Recurrent Neural Networks

- Recurrent Neural Networks are networks with loops allowing information to persist.



Output is to predict a vector h_t, where $output\ y_t = \varphi(h_t)$ at some time steps (t)



Recurrent Neural Networks have loops.

In the above diagram, a chunk of neural network, $A = f_W$, looks at some input x_t and outputs a value h_t. A loop allows information to be passed from one step of the network to the next.

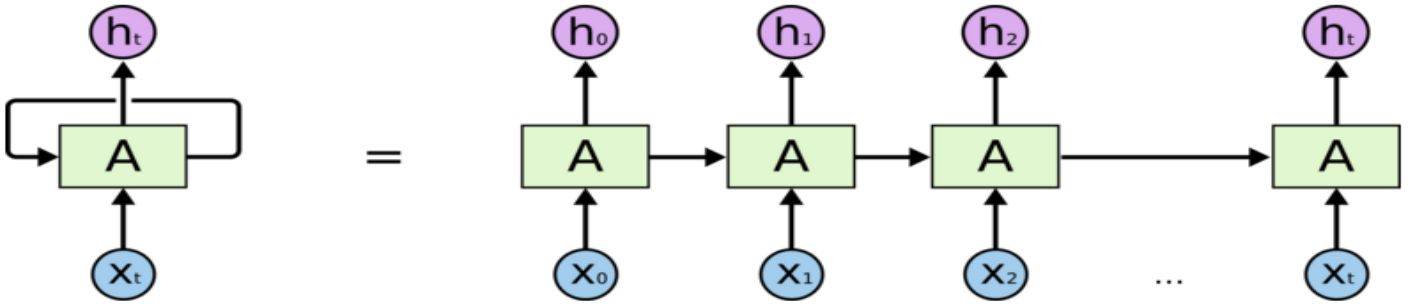
new state — old state

$$h_t = f_W(h_{t-1}, x_t)$$

function with parameter W — Input vector at some time step

Recurrent Neural Networks

- Unrolling RNN



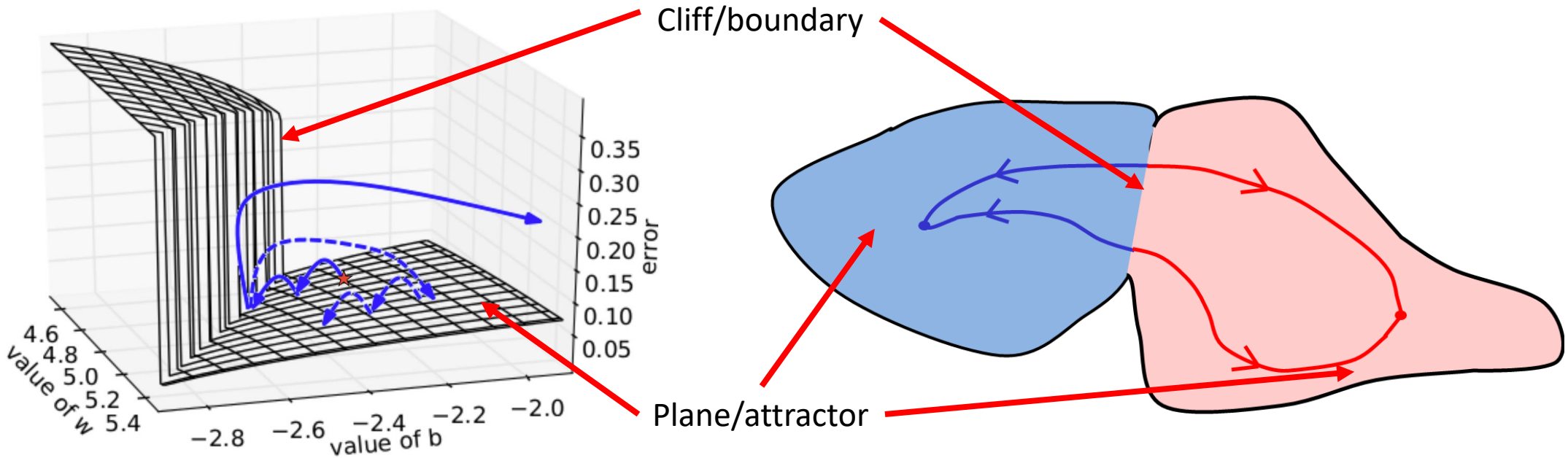
An unrolled recurrent neural network.

A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. The diagram above shows what happens if we **unroll the loop**.

Recurrent Neural Networks

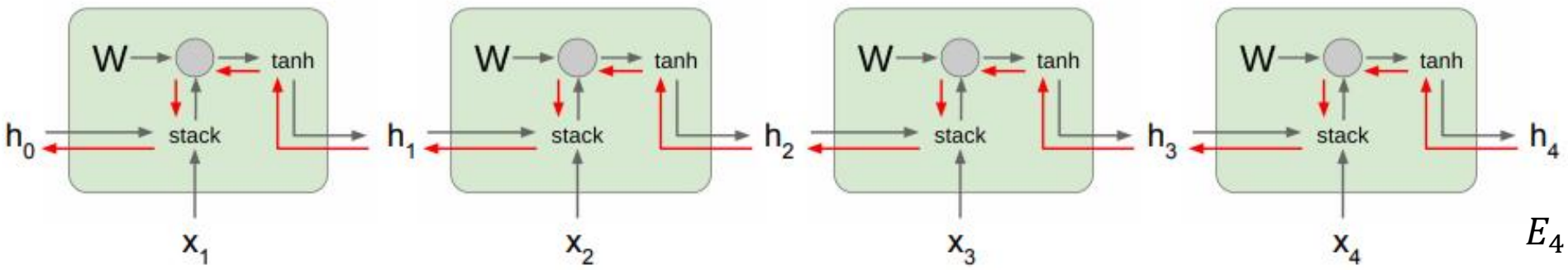
- The recurrent structure of RNNs enables the following characteristics:
  - Specialized for processing a sequence of values $x^{(1)}, \ldots, x^{(\tau)}$
    - Each value $x^{(i)}$ is processed with the **same network *A* that preserves past information**
  - Can scale to much **longer sequences** than would be practical for networks without a recurrent structure
    - Reusing network ***A*** reduces the required amount of parameters in the network
  - Can process **variable-length sequences**
    - The network complexity does not vary when the input length change
  - However, vanilla RNNs suffer from the training difficulty due to **exploding and vanishing gradients**.

Exploding and Vanishing Gradients



- **Exploding:** If we start almost exactly on the boundary (cliff), tiny changes can make a huge difference.
- **Vanishing:** If we start a trajectory within an attractor (plane, flat surface), small changes in where we start make no difference to where we end up.
- Both cases hinder the learning process.

Exploding and Vanishing Gradients



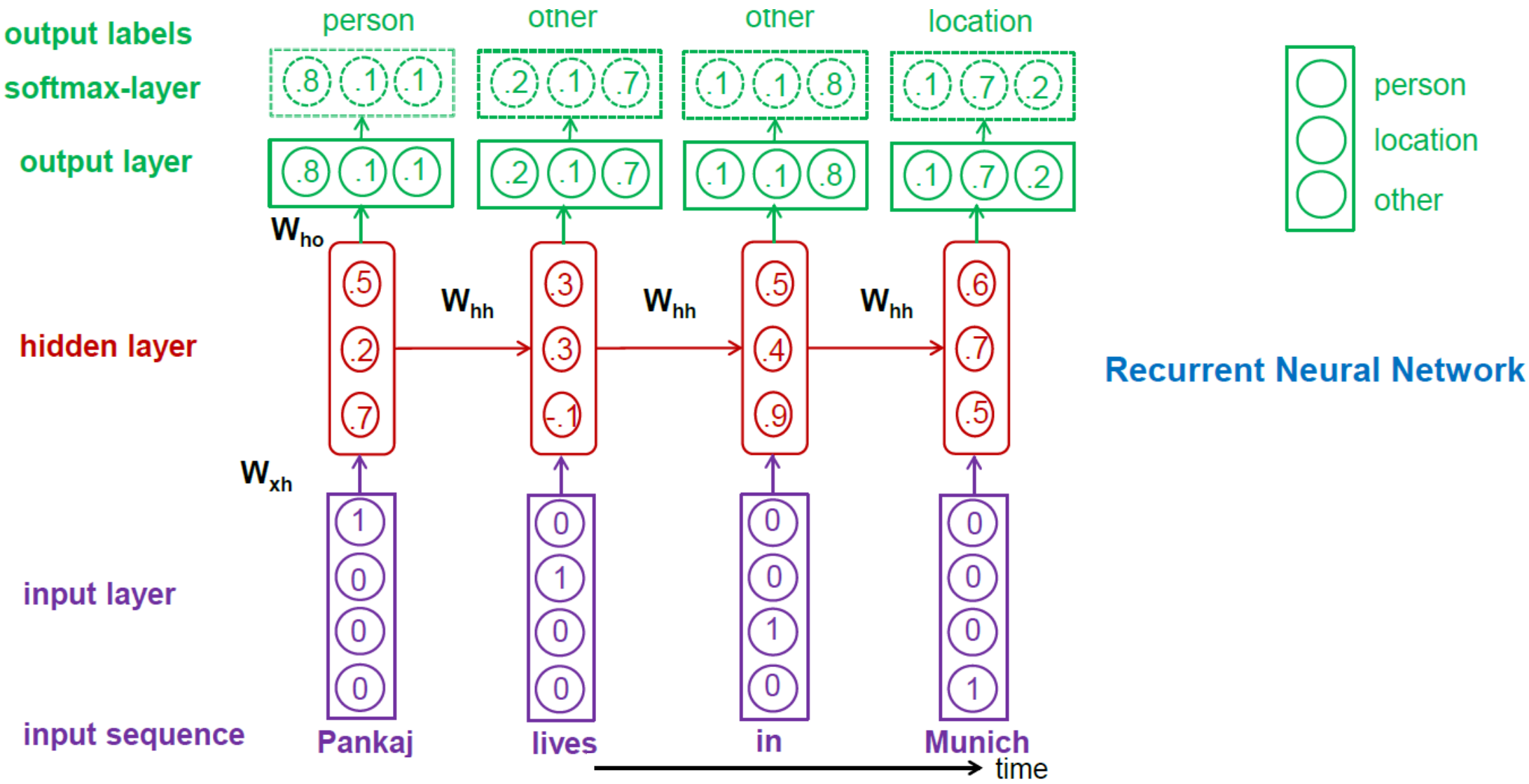
$$E_4 = J(y_4, \widehat{y_4})$$

In vanilla RNNs, computing this gradient involves many factors of W_{hh} (and repeated $\tanh$)*. If we decompose the singular values of the gradient multiplication matrix,

- Largest singular value > 1 → **Exploding gradients**
  - Slight error in the late time steps causes drastic updates in the early time steps → Unstable learning
- Largest singular value < 1 → **Vanishing gradients**
  - Gradients passed to the early time steps is close to 0. → Uninformed correction

* Refer to Bengio et al. (1994) or Goodfellow et al. (2016) for a complete derivation

Foundation of Recurrent Neural Networks



(Vanilla) Recurrent Neural Network

Process a sequence of vectors **x** by applying a recurrence at every time step:

$$h_t = g_\theta(h_{t-1}, x_t)$$

new hidden state at time step, t

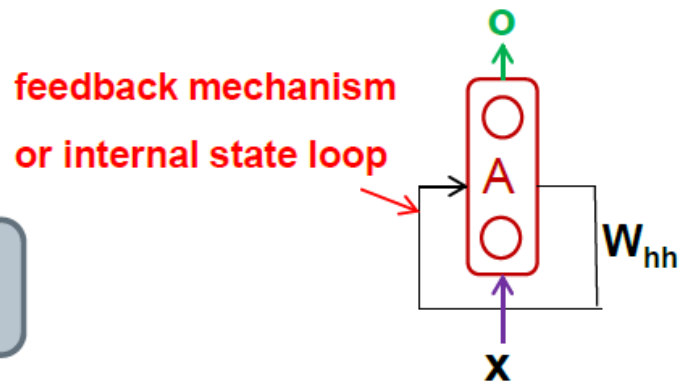
some function with parameters $\mathbf{W}_{hh}\,\mathbf{W}_{xh}$

old hidden state at time step, t-1

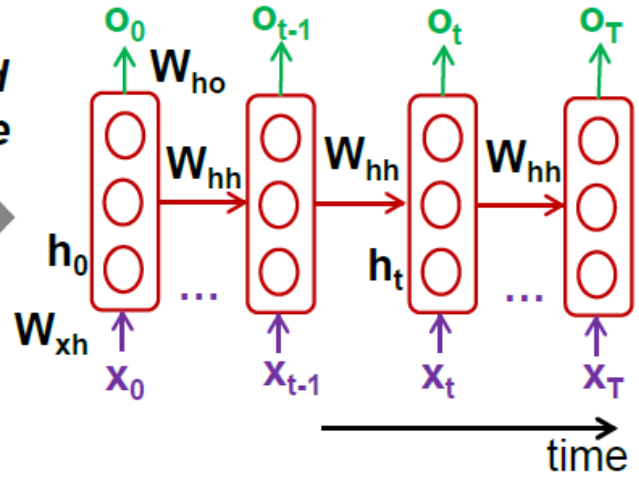
Input vector at time step, t

feedback mechanism

or internal state loop



Unfold in time



Vanilla Recurrent Neural Network (RNN)

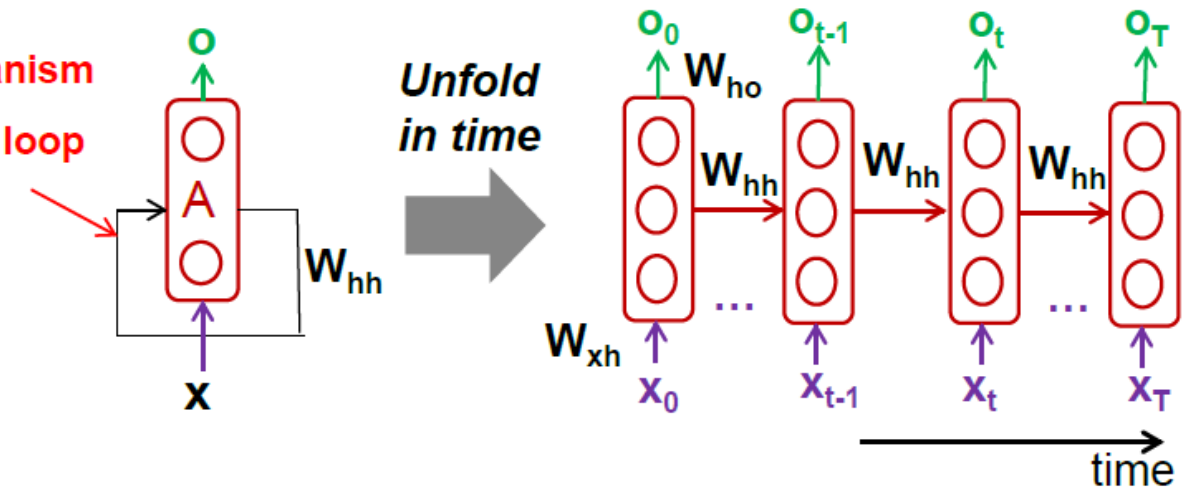
Remark: The same function *g* and same set of parameters **W** are used at every time step

(Vanilla) Recurrent Neural Network

Process a sequence of vectors **x** by applying a recurrence at every time step:

$$h_t = g_\theta(h_{t-1}, x_t)$$
$$\downarrow$$
$$h_t = \tanh(W_{hh}\, h_{t-1} + W_{xh} x_t)$$

$$o_t = softmax(W_{ho} h_t)$$



Vanilla Recurrent Neural Network (RNN)

Remark: RNN's can be seen as **selective summarization** of input sequence in a fixed-size state/hidden vector via a recursive update.

Recurrent Neural Network: Probabilistic Interpretation

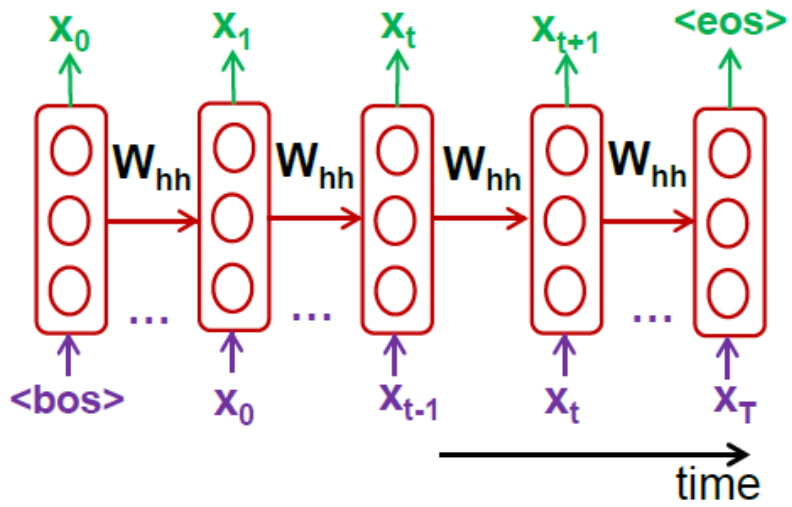
RNN as a generative model

- induces a set of procedures to model

the conditional distribution of $\mathbf{x}_{t+1}$ given $\mathbf{x}<=t$

for all t = 1, …,T

$$P(x) = P(x_1, \dots x_T) = \sum_{t=1}^{T} P(x_t | x_{t-1}, x_{t-2}, .. x_1)$$

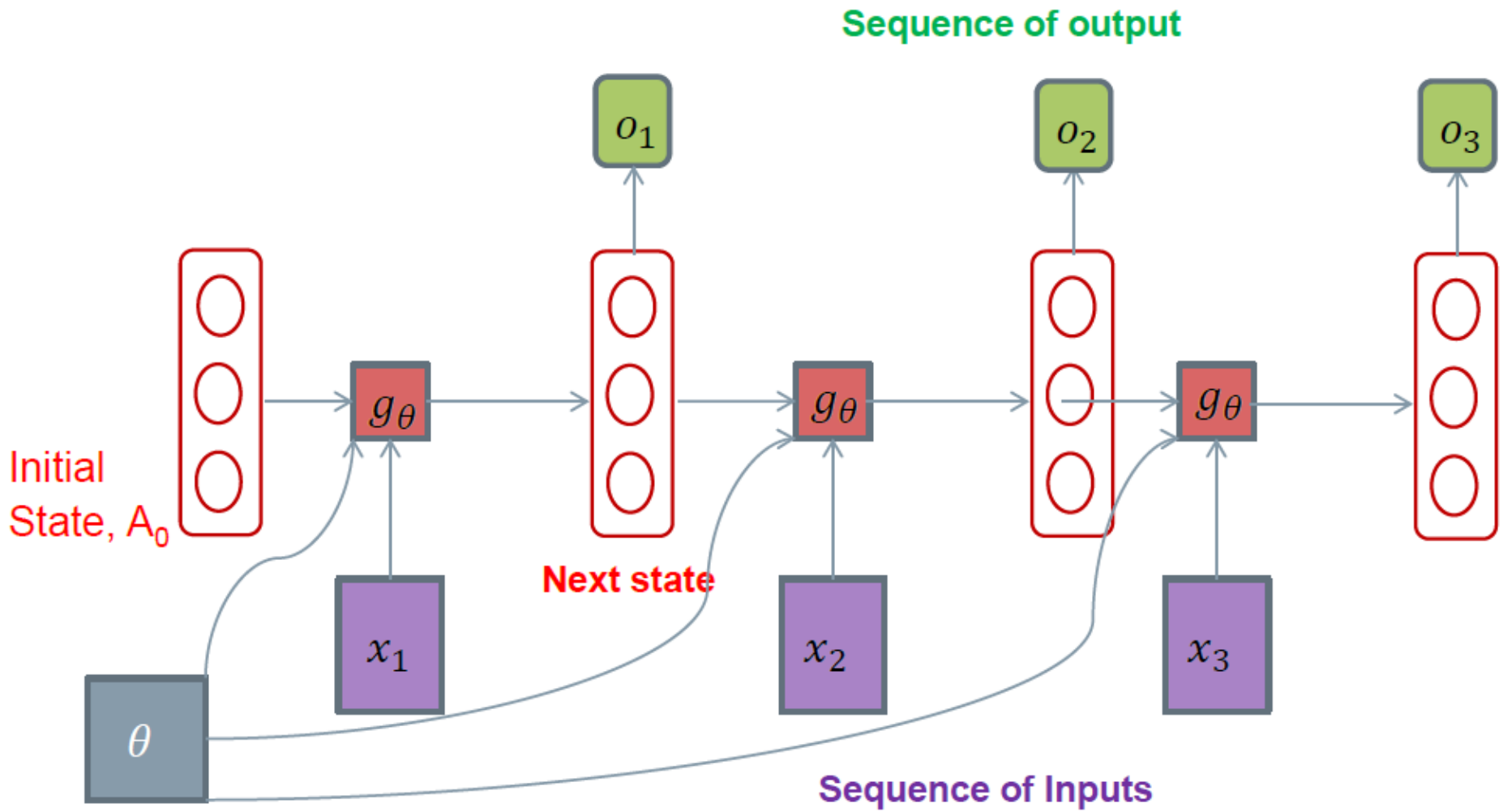


Generative Recurrent Neural Network (RNN)

- Think of the output as the probability distribution of the $\mathbf{x}_t$ given the previous ones in the sequence
- Training: Computing probability of the sequence and Maximum likelihood training

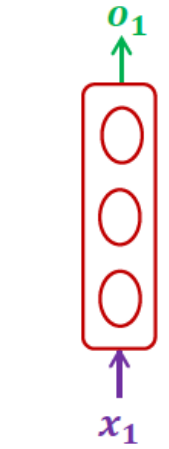
$$L_t = -\log P(x_t | x_{t-1}, x_{t-2}, .. x_1)$$

RNN: Computational Graphs



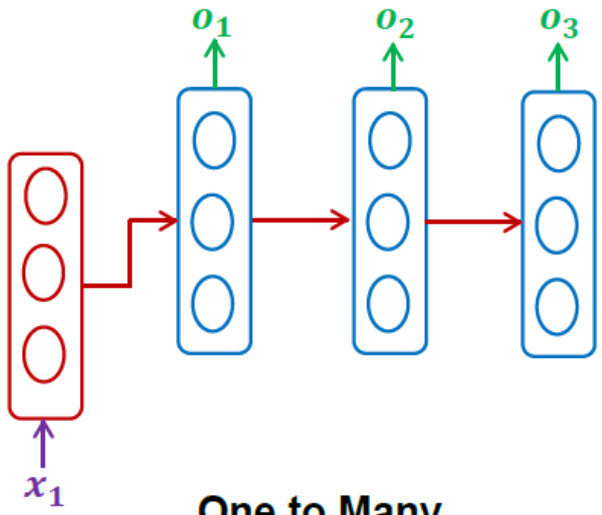
RNN: Different Computational Graphs

Image Classification.



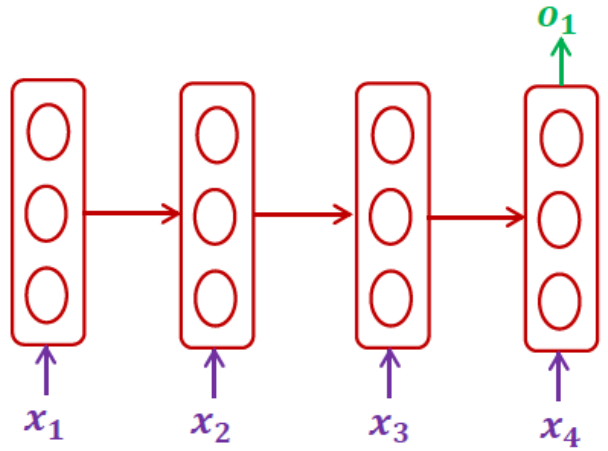
One to one

Music Generation and *Image Captioning*.

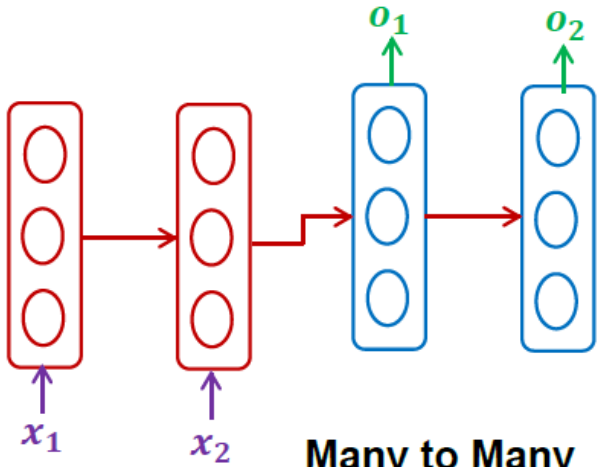


One to Many

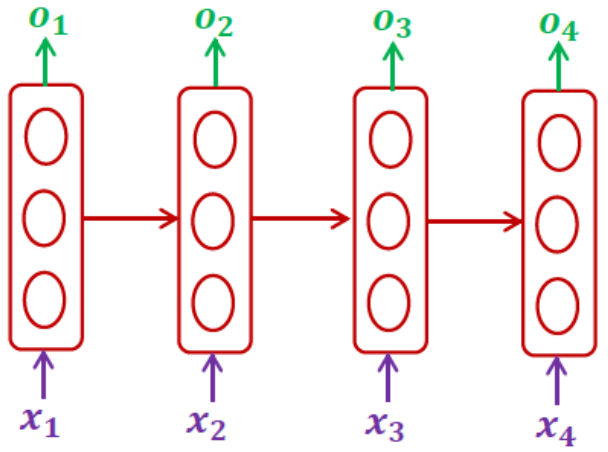
Sentiment Analysis



Many to One



Many to Many

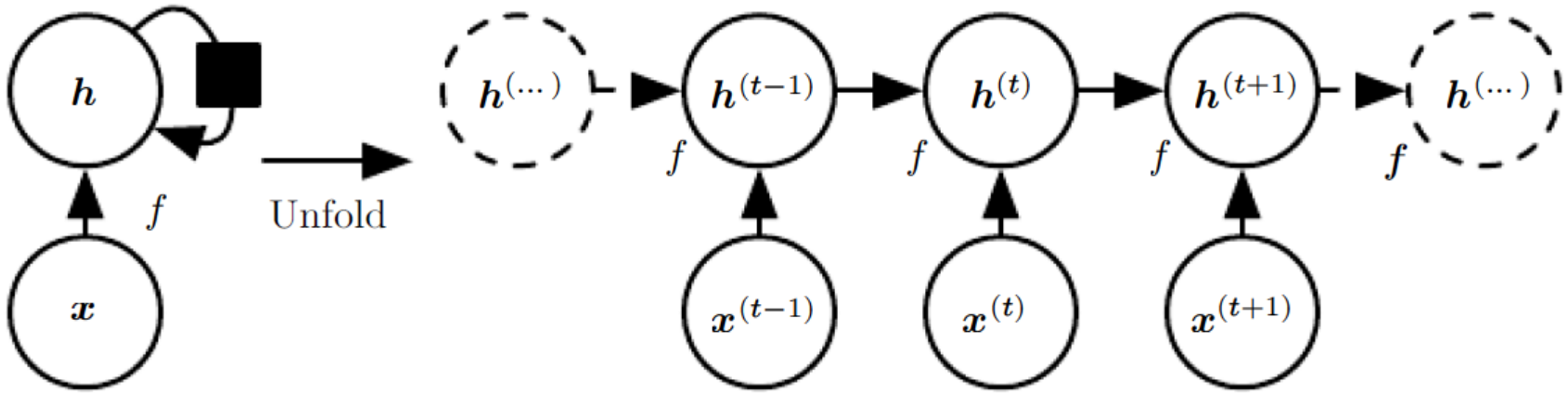


Many to Many

Name-Entity Recognition.

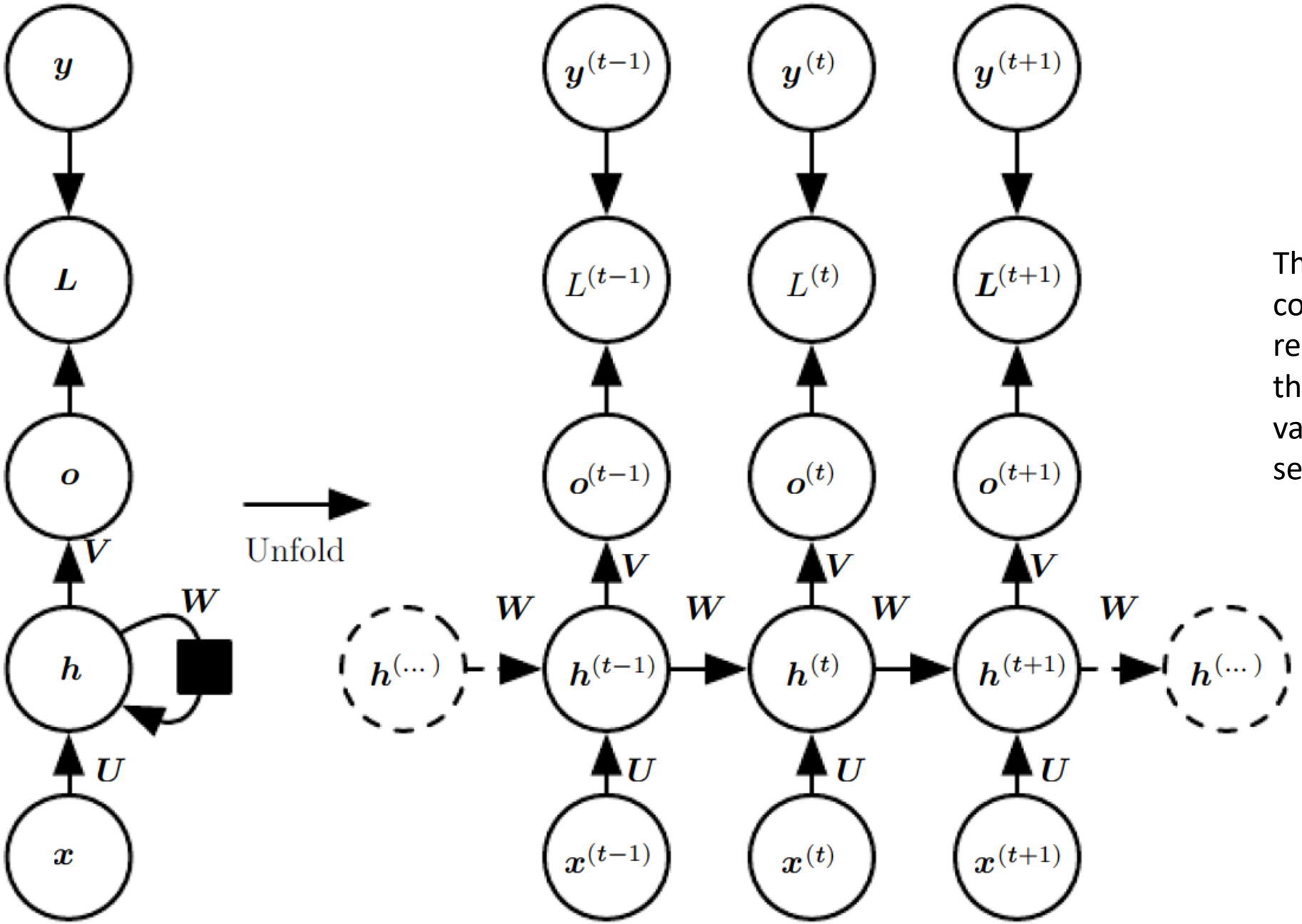
Machine Translation.

RNN-Design patterns



A recurrent network with no outputs

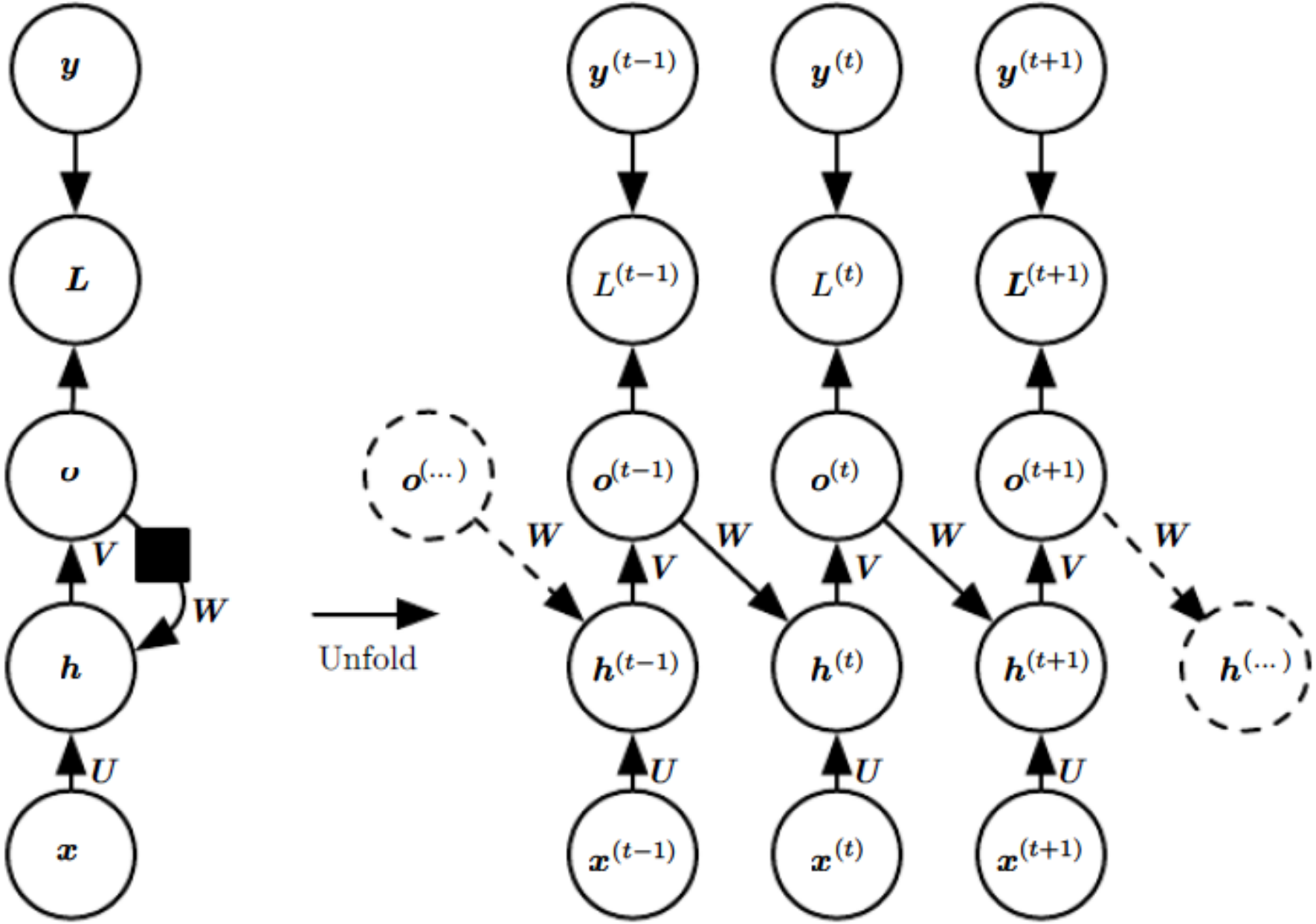
RNN-Design patterns



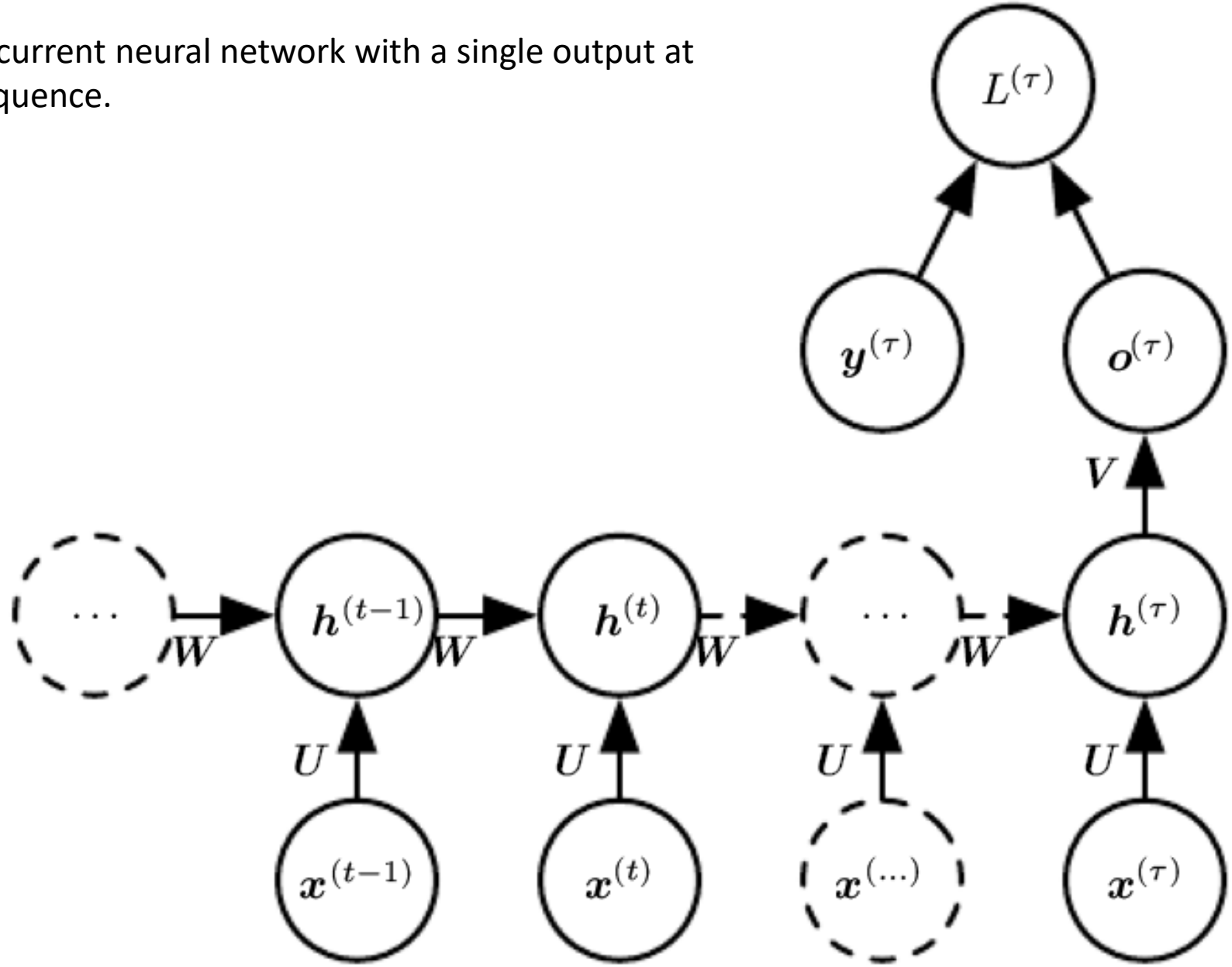
The computational graph to compute the training loss of a recurrent network
that maps an input sequence of x values to a corresponding sequence of output o values

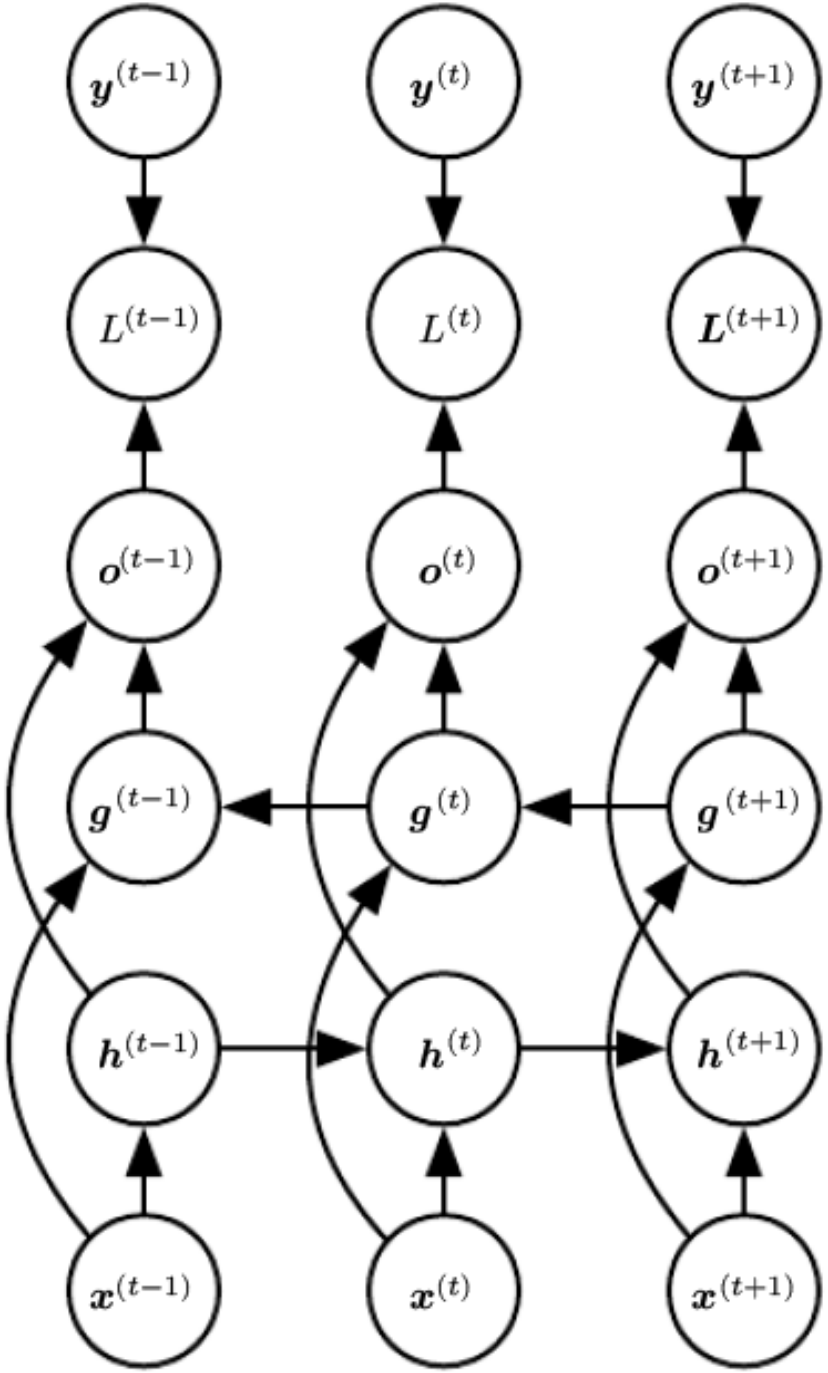
Recurrent networks that produce an output at each time step and have recurrent connections between hidden units

RNN-Design patterns



Time-unfolded recurrent neural network with a single output at the end of the sequence.

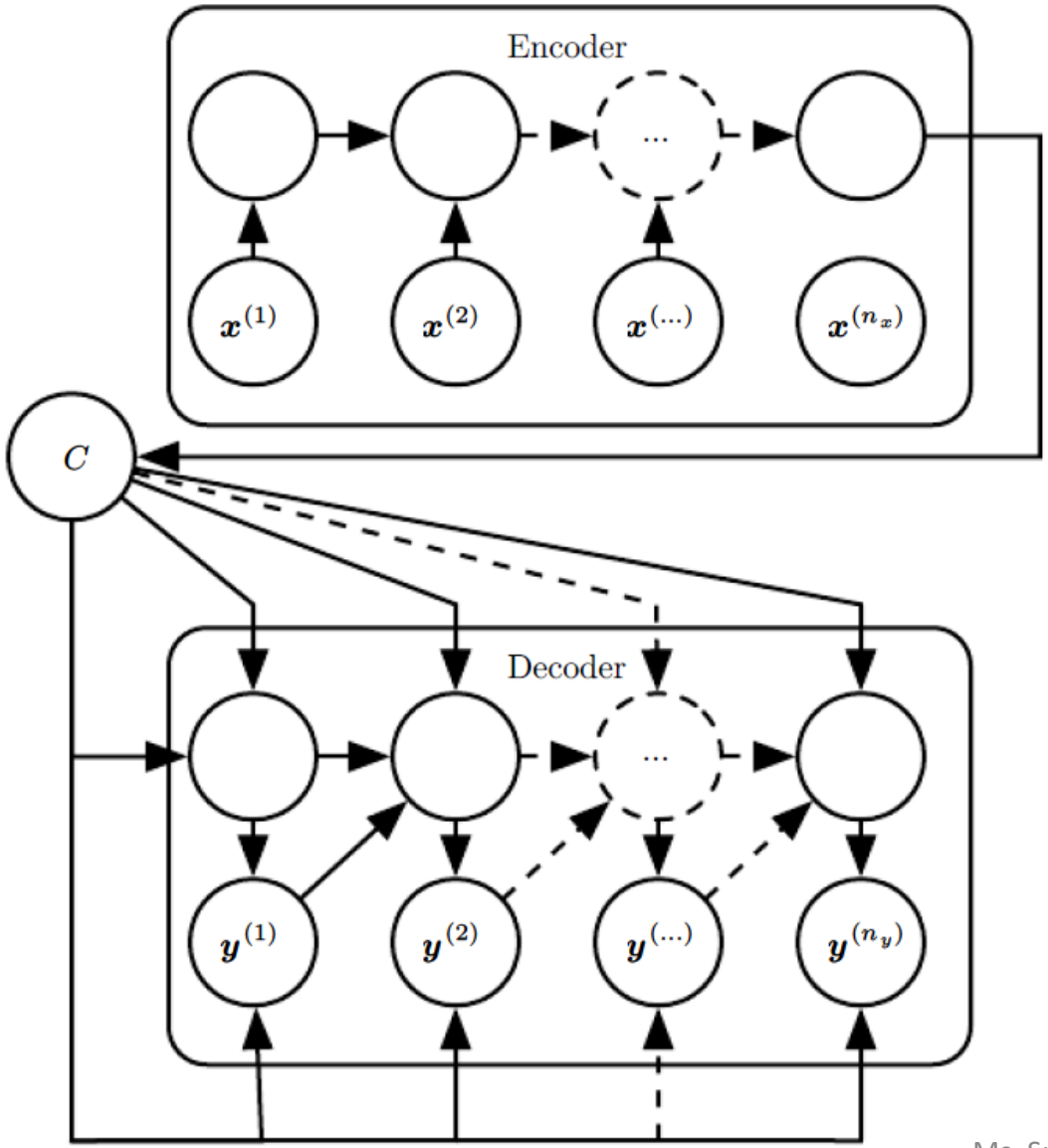




Bidirectional RNNs

Bidirectional RNNs combine an RNN that moves forward through time beginning from the start of the sequence with another RNN that moves backward through time beginning from the end of the sequence.

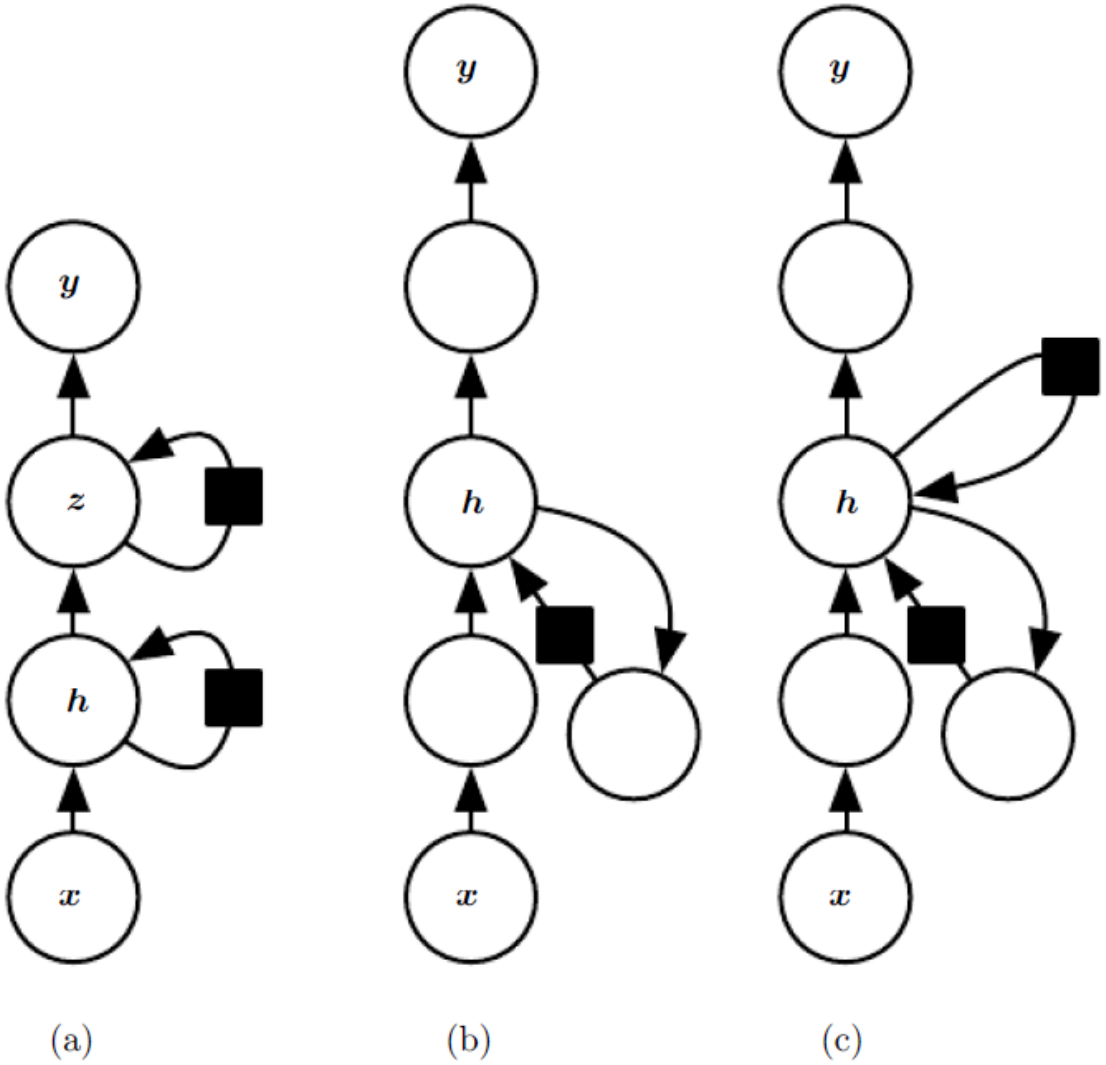
Example of an encoder-decoder or sequence-to-sequence RNN architecture



Deep Recurrent Networks

- The computation in most RNNs can be decomposed into three blocks of parameters and associated transformations:
  - from the input to the hidden state,
  - from the previous hidden state to the next hidden state
  - from the hidden state to the output.

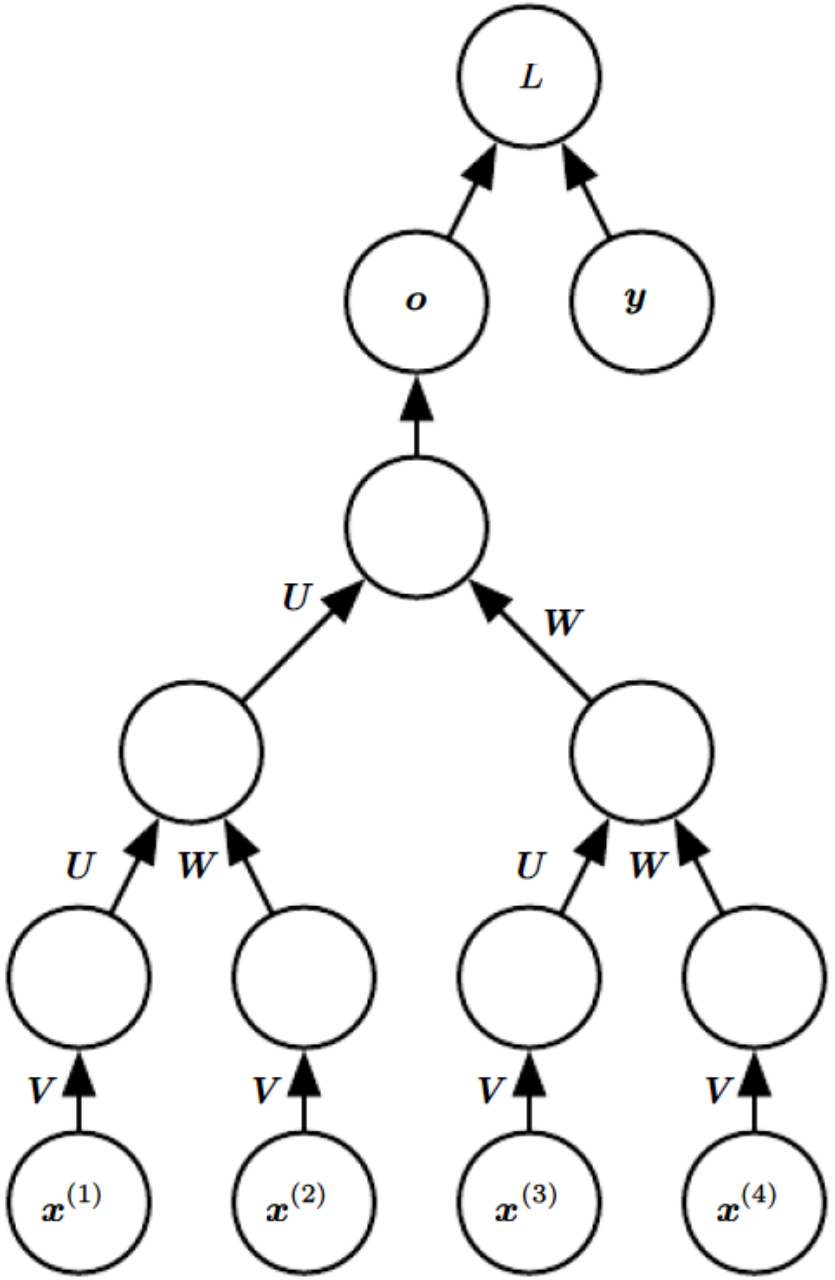
Deep Recurrent Networks



A recurrent neural network can be made deep in many ways

Recursive Neural Networks

- Generalization of recurrent networks, with a different kind of computational graph, structured as a deep tree, rather than the chain-like structure of RNNs.



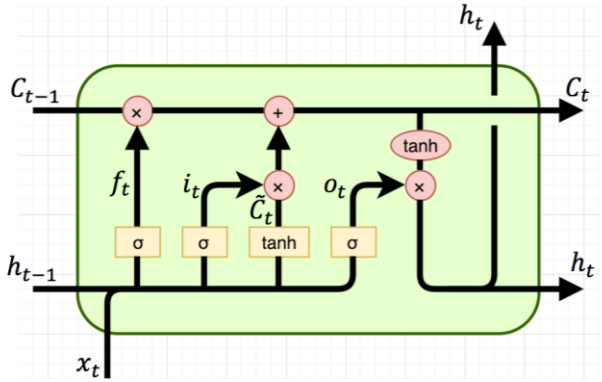
Networks with Memory

- Vanilla RNN operates in a “multiplicative” way (repeated tanh).

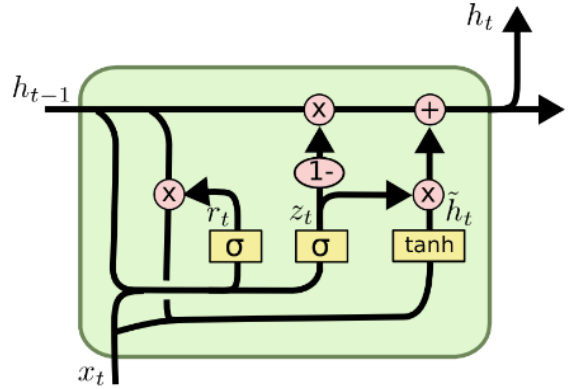
- Two recurrent cell designs were proposed and widely adopted:
  - **Long Short-Term Memory (LSTM)** (Hochreiter and Schmidhuber, 1997)
  - Gated Recurrent Unit (GRU) (Cho et al. 2014)

- Both designs process information in an “additive” way with gates to control information flow.
  - **Sigmoid gate outputs numbers between 0 and 1, describing how much of each component should be let through.**

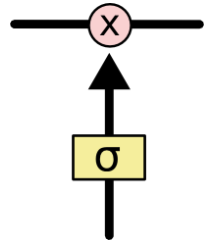
E.g. $f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right) = Sigmoid\left(W_f x_t + U_t h_{t\text{-}1} + b_f\right)$



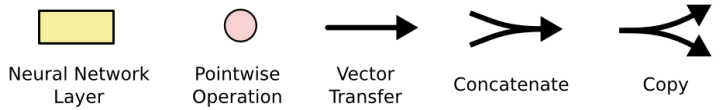
Standard LSTM Cell



GRU Cell



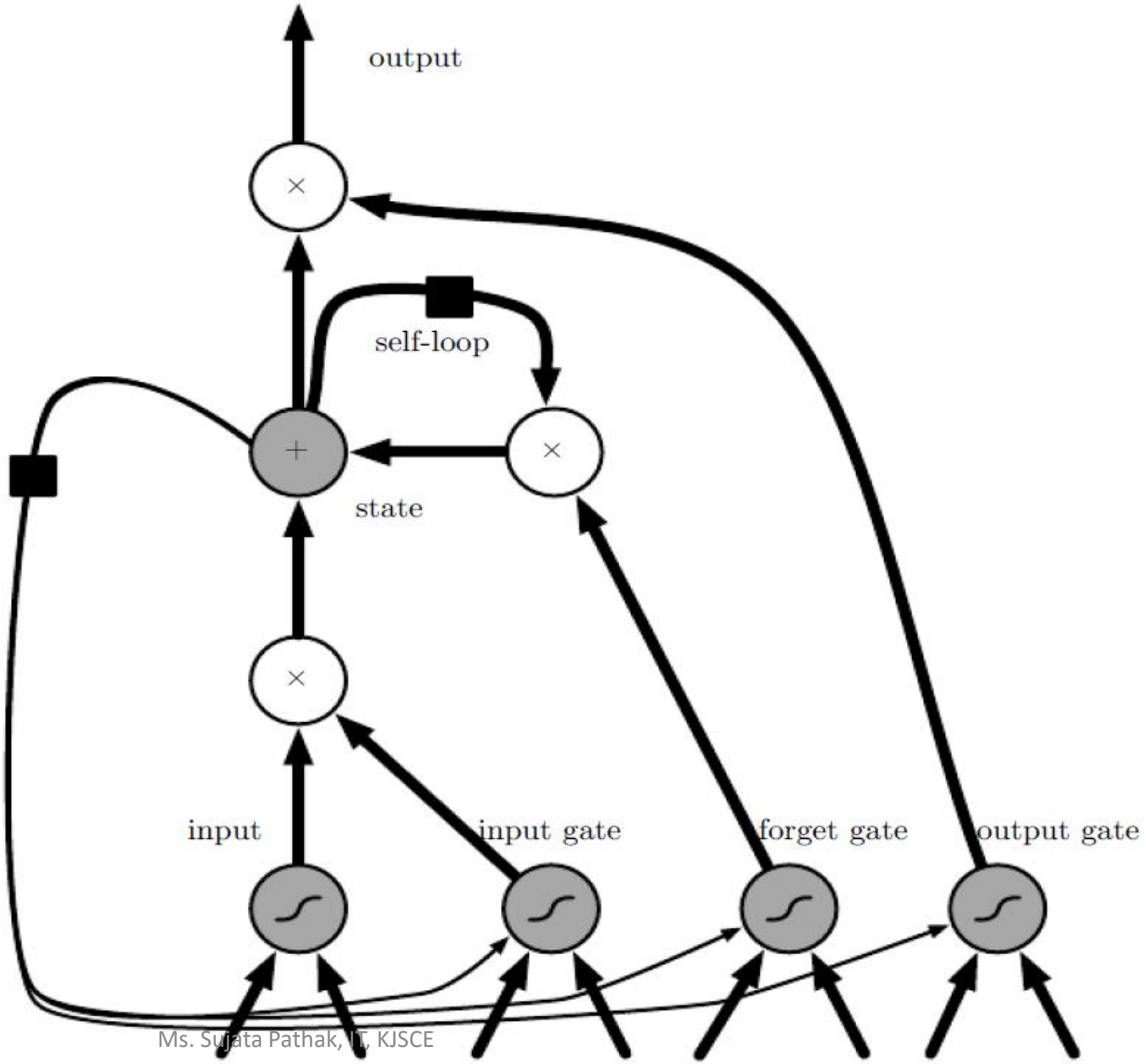
A Sigmoid Gate



The Long Short-Term Memory and Other Gated RNNs

- Most effective sequence models used in practical applications are called gated RNNs.
  - Long short-term memory
  - Networks based on the gated recurrent unit.
- Gated RNNs are based on the idea of creating paths through time that have derivatives that neither vanish nor explode
- Gated RNNs generalize this to connection weights that may change at each time step

Block diagram of the LSTM recurrent network “cell.”



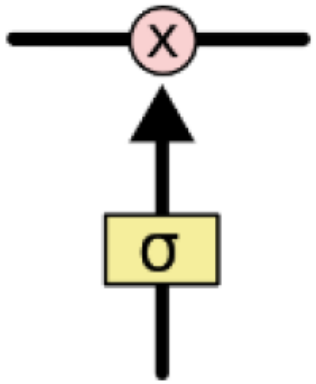
Long Short-Term Memory (LSTM): Gating Mechanism

Gates :

→ way to optionally let information through.

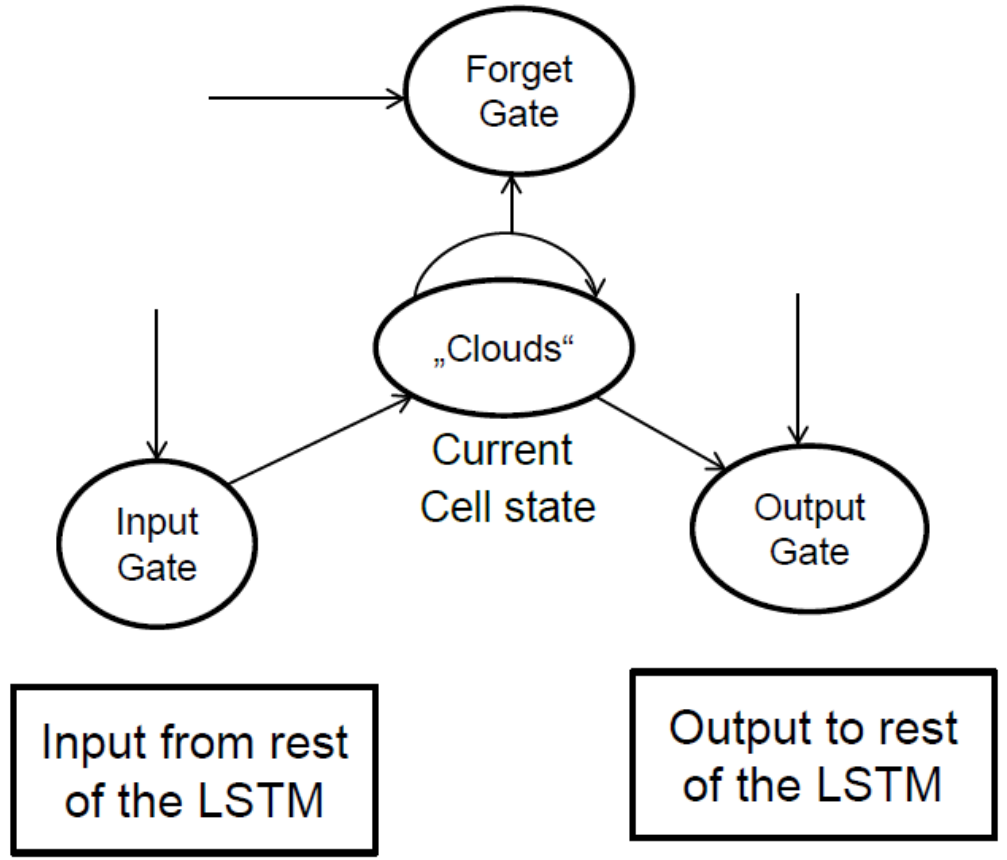
→ composed out of a sigmoid neural net layer and a pointwise multiplication operation.

→ remove or add information to the cell state



→ 3 gates in LSTM

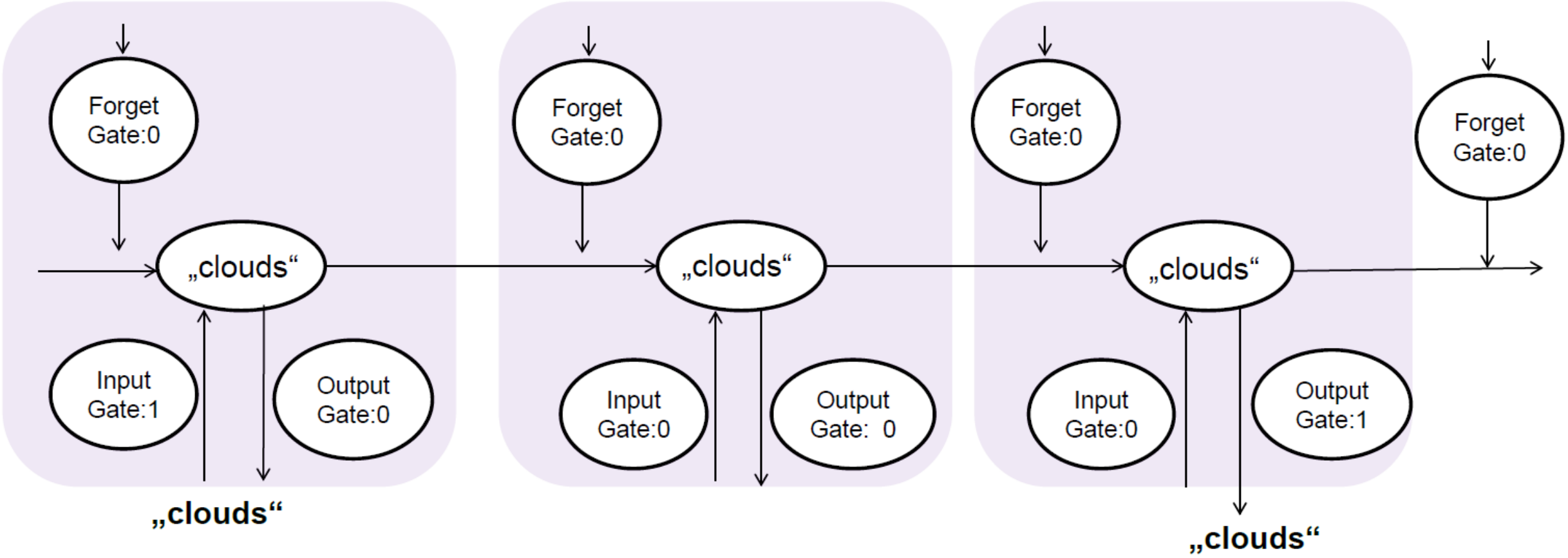
→ gates to protect and control the cell state.



Long Short-Term Memory (LSTM): Gating Mechanism

*"the clouds are in the**?**" → 'sky'*

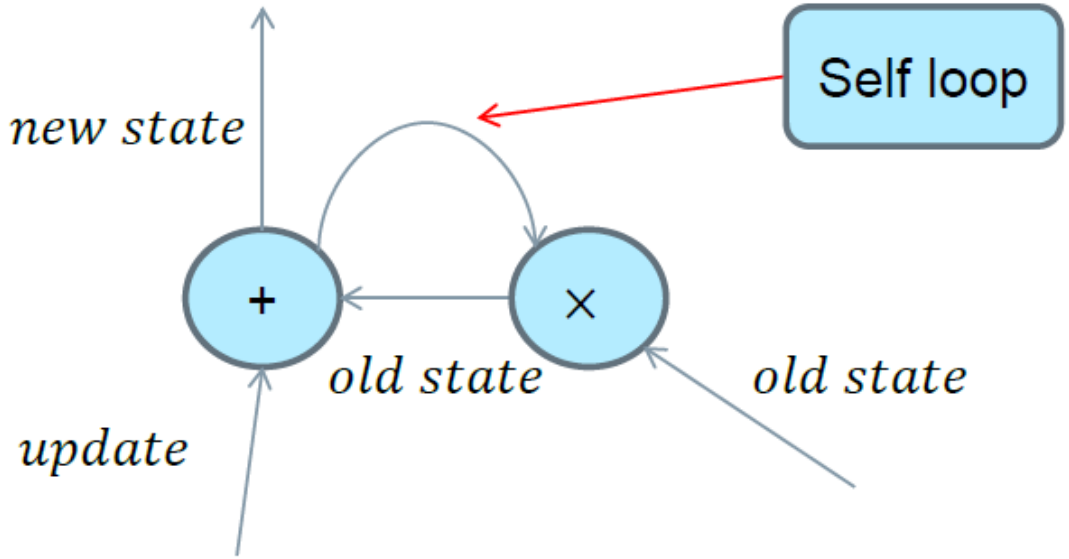
Remember the word „ **clouds**" over time….



Long Short-Term Memory (LSTM)

Motivation:

- **Create a self loop path from where gradient can flow**
- **self loop corresponds to an eigenvalue of Jacobian to be slightly less than 1**



$$new\ state = old\ state + update$$

$$\frac{\partial new\ state}{\partial old\ state} \cong Identity$$

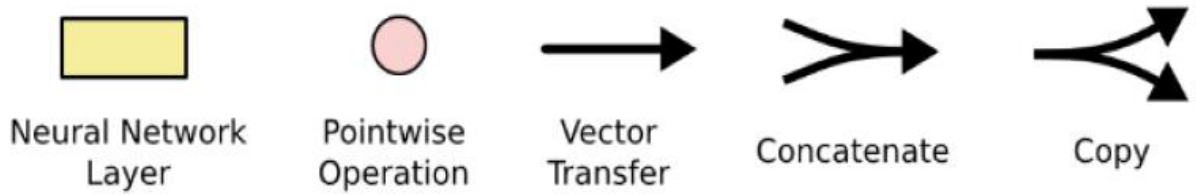
LONG SHORT-TERM MEMORY, Sepp Hochreiter and Jürgen Schmidhuber

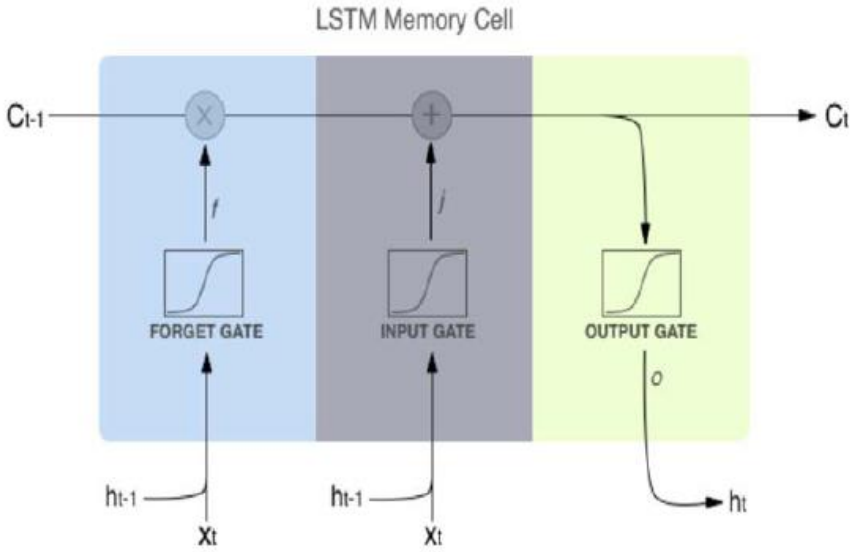
Long Short-Term Memory (LSTM)

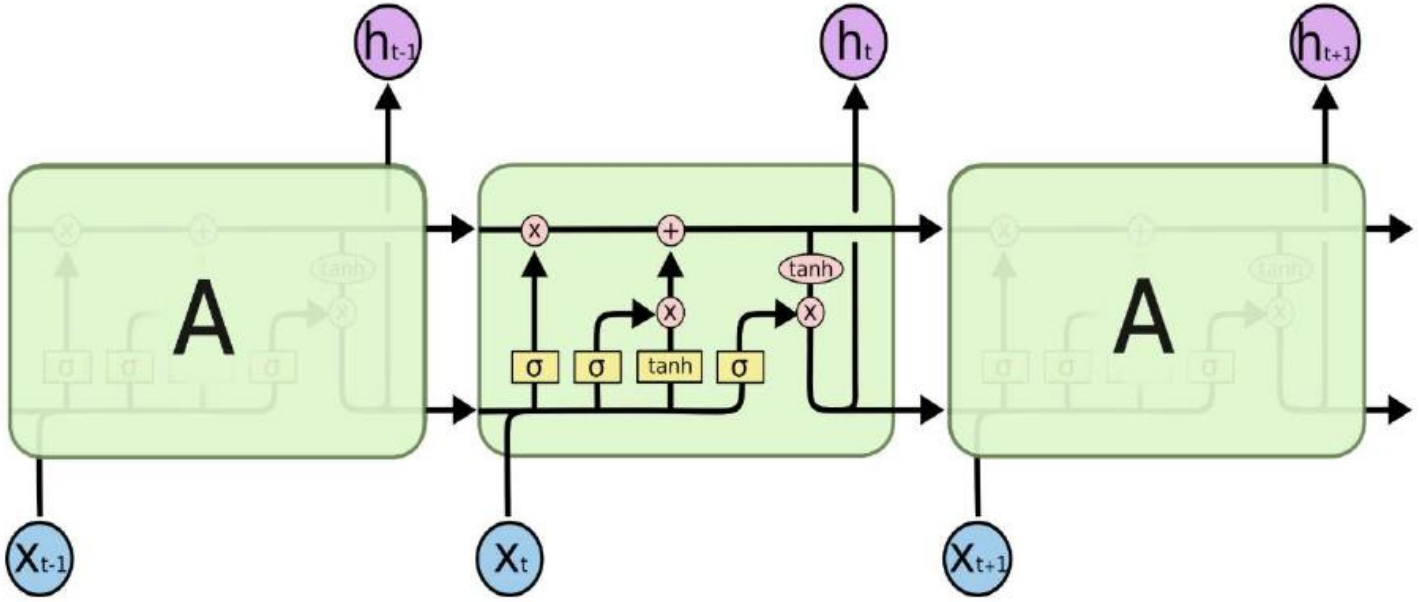
Key Ingredients

Cell state - transport the information through the units

Gates – optionally allow information passage







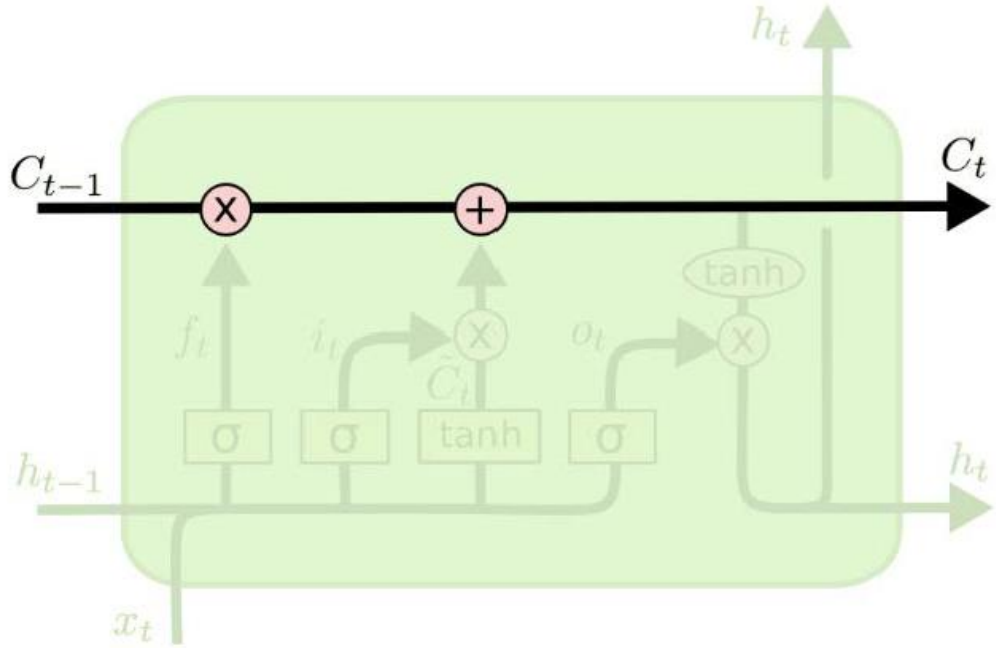
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

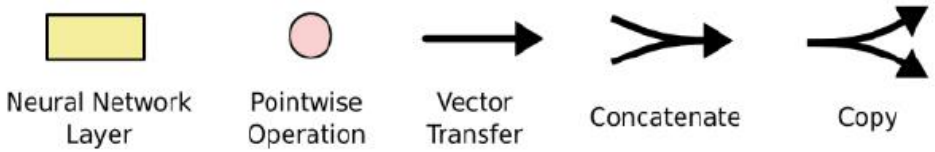
Long Short-Term Memory (LSTM): Step by Step

Cell: Transports information through the units (key idea)

→ the horizontal line running through the top

LSTM removes or adds information to the cell state using gates.





Long Short-Term Memory (LSTM): Step by Step

Forget Gate:

→ decides what information to throw away or remember from the previous cell state

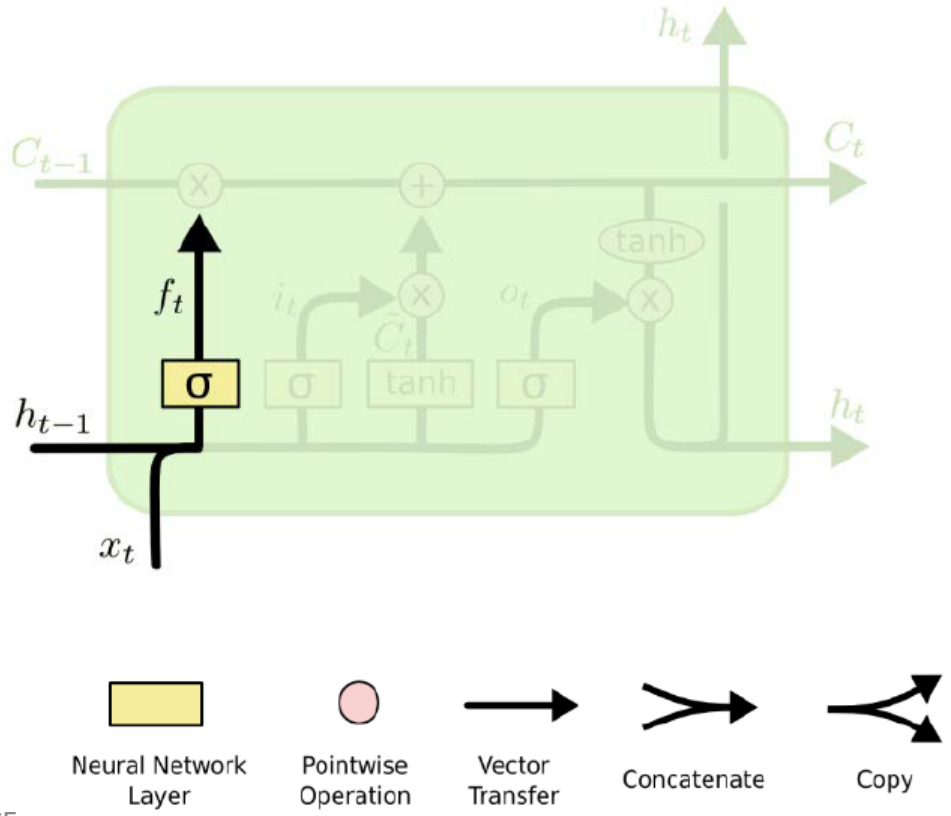
→ decision maker: sigmoid layer (*forget gate layer*)

The output of the sigmoid lies between 0 to 1,

→ 0 being forget, 1 being keep.

$$f_t = sigmoid(\theta_{xf} x_t + \theta_{hf} h_{t-1} + b_f)$$

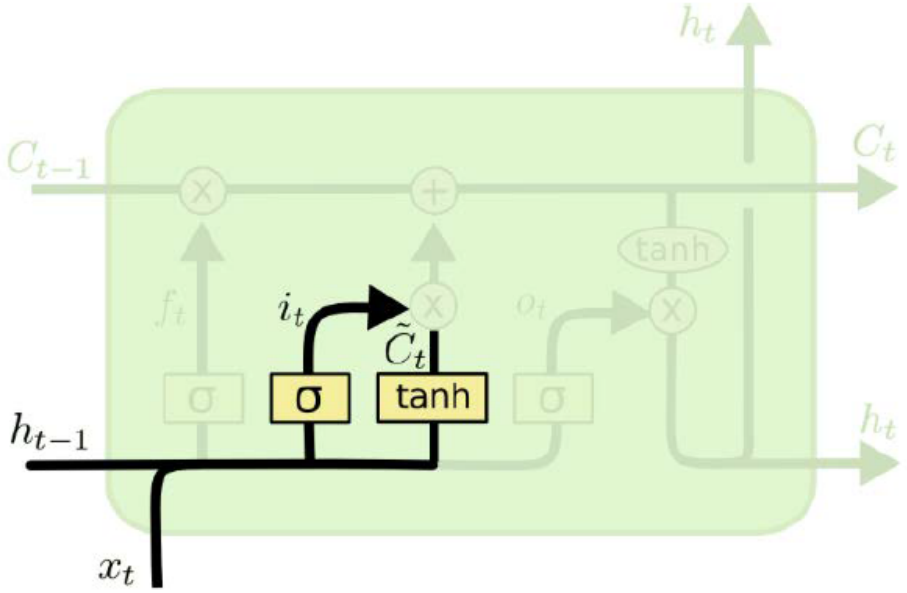
→ looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}



Long Short-Term Memory (LSTM): Step by Step

Input Gate: Selectively updates the cell state based on the new input.

A multiplicative input gate unit to protect the memory contents stored in j from perturbation by irrelevant inputs



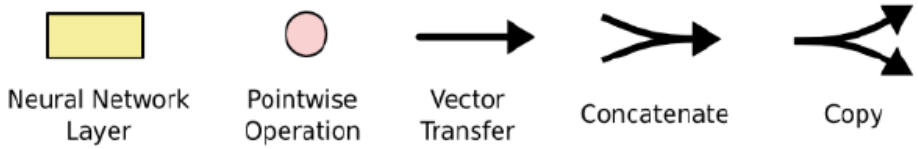
$$i_t = sigmoid(\theta_{xi}x_t + \theta_{hi}h_{t-1} + b_i)$$
$$\tilde{C}_t = Tanh(\theta_{xg}x_t + \theta_{hg}h_{t-1} + b_g)$$

The next step is to decide what new information we're going to store in the cell state. This has two parts:

1. A sigmoid layer called the "input gate layer" decides **which values we'll update**.

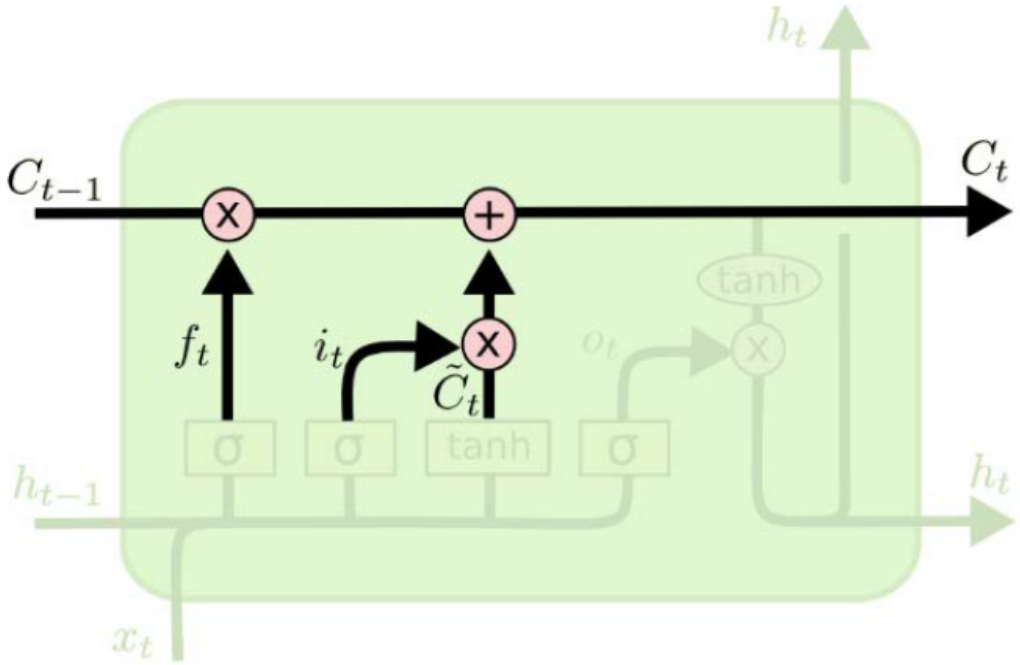
2. A tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that **could be added to the state**.

In the next step, we'll combine these two to **create an update** to the state.



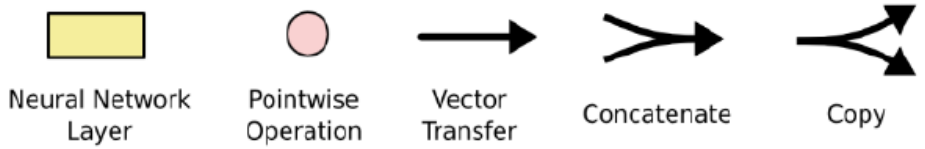
Long Short-Term Memory (LSTM): Step by Step

Cell Update



- update the old cell state, C_{t-1}, into the new cell state C_t
- multiply the old state by f_t, forgetting the things we decided to forget earlier
- add $i_t * \tilde{C}_t$ to get the new candidate values, scaled by how much we decided to update each state value.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



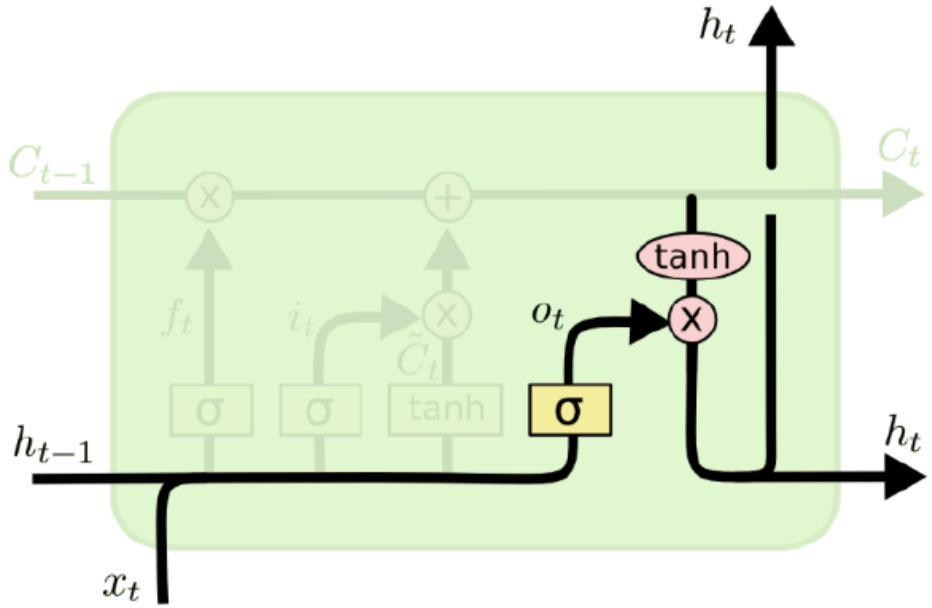
Long Short-Term Memory (LSTM): Step by Step

Output Gate: Output is the filtered version of the cell state

- Decides the part of the cell we want as our output in the form of new hidden state

- multiplicative output gate to protect other units from perturbation by currently irrelevant memory contents

- a sigmoid layer decides what parts of the cell state goes to output. Apply tanh to the cell state and multiply it by the output of the sigmoid gate → only output the parts decided



$$o_t = sigmoid(\theta_{xo} x_t + \theta_{ho} h_{t-1} + b_o)$$

$$h_t = o_t * tanh(C_t)$$

Gated Recurrent Unit (GRU)

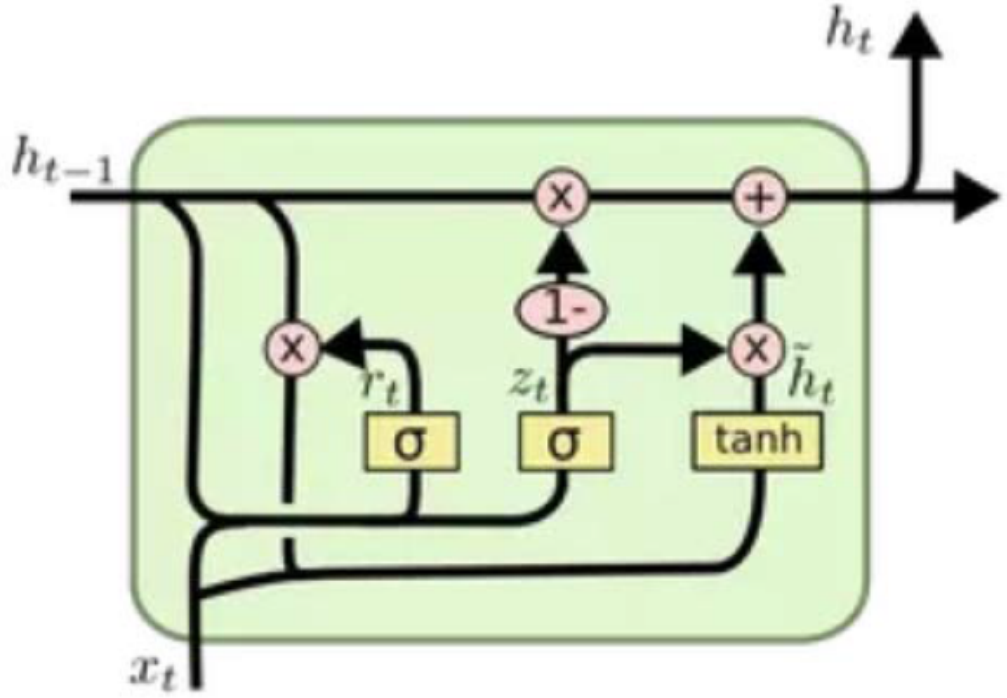
- GRU like LSTMs, attempts to solve the Vanishing gradient problem in RNN

Gates:

Update Gate

Reset Gate

These 2 vectors decide what information should be passed to output



- Units with short-term dependencies will have active reset gates r
- Units with long term dependencies have active update gates z

Gated Recurrent Unit (GRU)

Update Gate:

- to determine how much of the past information (from previous time steps) needs to be passed along to the future.
- to learn to copy information from the past such that gradient is not vanished.

Here, x_t is the input and h_{t-1} holds the information from the previous gate.

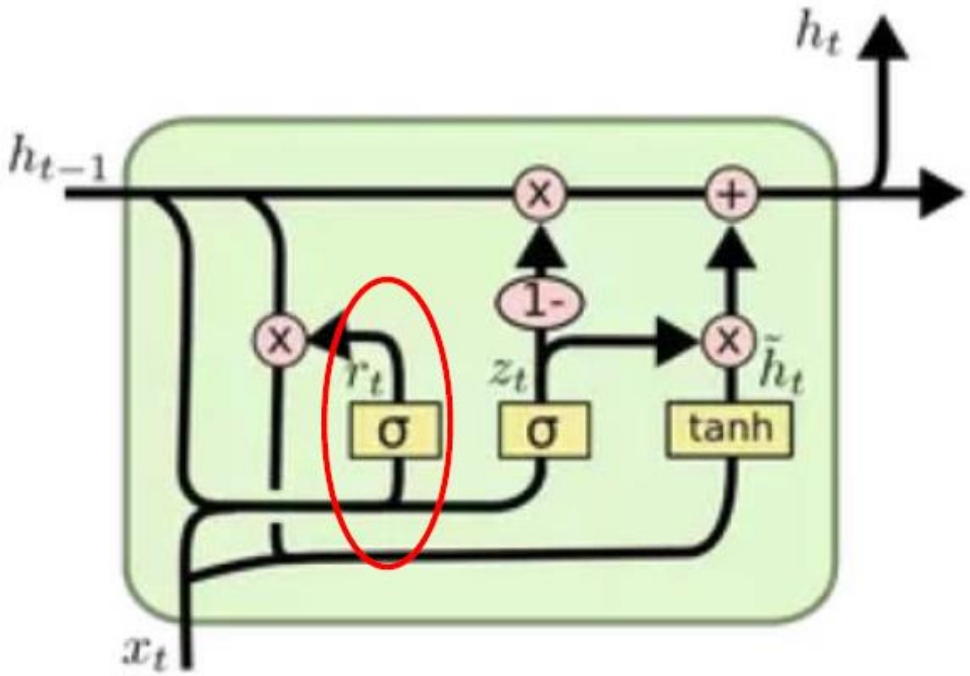
$$z_t = sigmoid(W^z x_t + U^z h_{t-1})$$

Gated Recurrent Unit (GRU)

Reset Gate

- model how much of information to forget by the unit

Here, x_t is the input and h_{t-1} holds the information from the previous gate.



$$r_t = sigmoid(W^{(r)}x_t + U^{(r)}h_{t-1})$$

Memory Content:

$$h'_t = tanh(Wx_t + r\_t \odot Uh_{t-1})$$

Final Memory at current time step

$$h_t = z_t \odot h_{(t-1)} + (1 - z_t) \odot h'_t$$

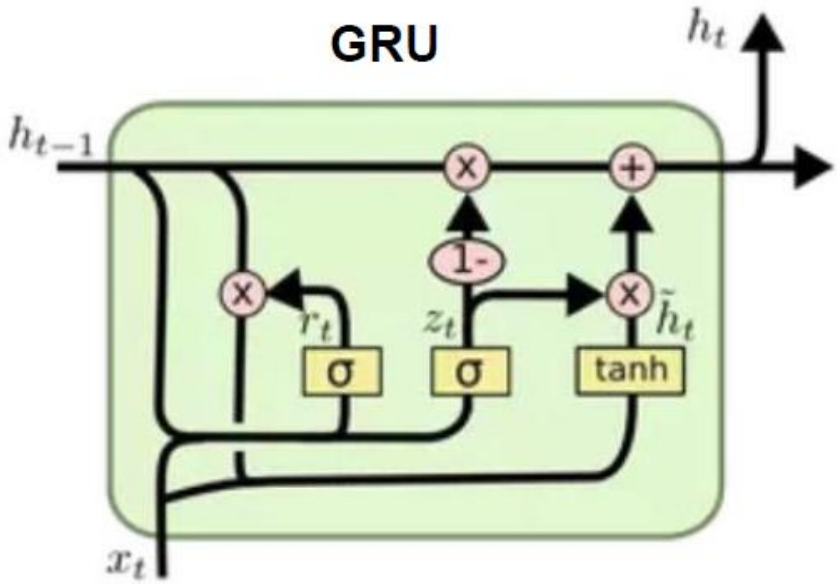
Comparing LSTM and GRU

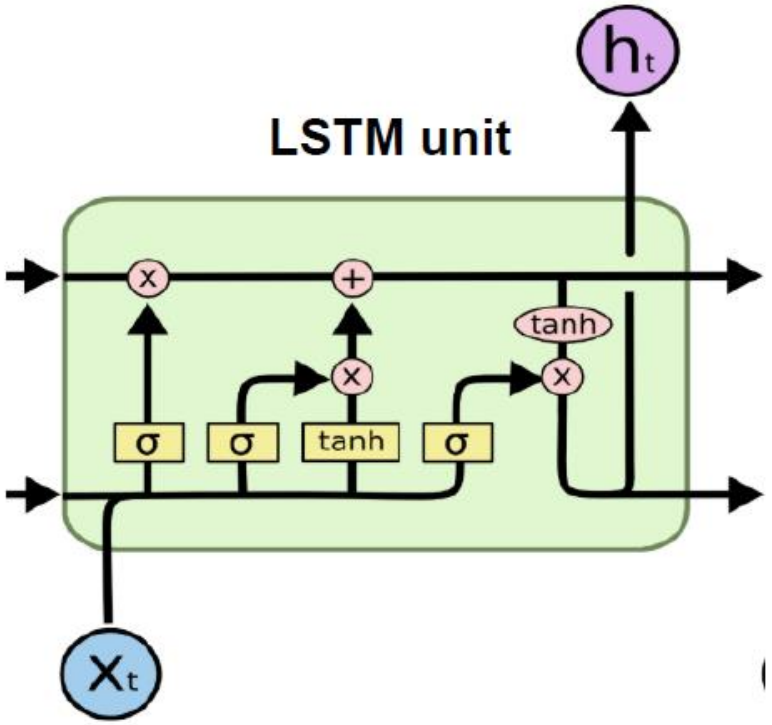
LSTM over GRU

One feature of the LSTM has: **controlled exposure of the memory content**, not in GRU.

In the LSTM unit, the amount of the memory content that is seen, or used by other units in the network is controlled by the output gate. On the other hand **the GRU exposes its full content without any control**.

- GRU performs comparably to LSTM





Chung et al, 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling

Dealing with Vanishing Gradientsin LSTM

As seen, the gradient vanishes due to the recurrent part of the RNN equations

$$h_t = W_{hh} \mathrm{h}_{t-1} + \text{some other terms}$$

How LSTM tackled vanishing gradient?

Answer: **forget gate**

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- The forget gate parameters takes care of the vanishing gradient problem
- Activation function becomes identity and therefore, the problem of vanishing gradient is addressed.
- The derivative of the identity function is, conveniently, always one. **So if f = 1**, information from the previous cell state can pass through this step unchanged

Bi-directional RNNs

Bidirectional Recurrent Neural Networks (BRNN)

- connects two hidden layers of opposite directions to the same output
- output layer can get information from past (backwards) and future (forward) states simultaneously
- learn representations from future time steps to better understand the context and eliminate ambiguity

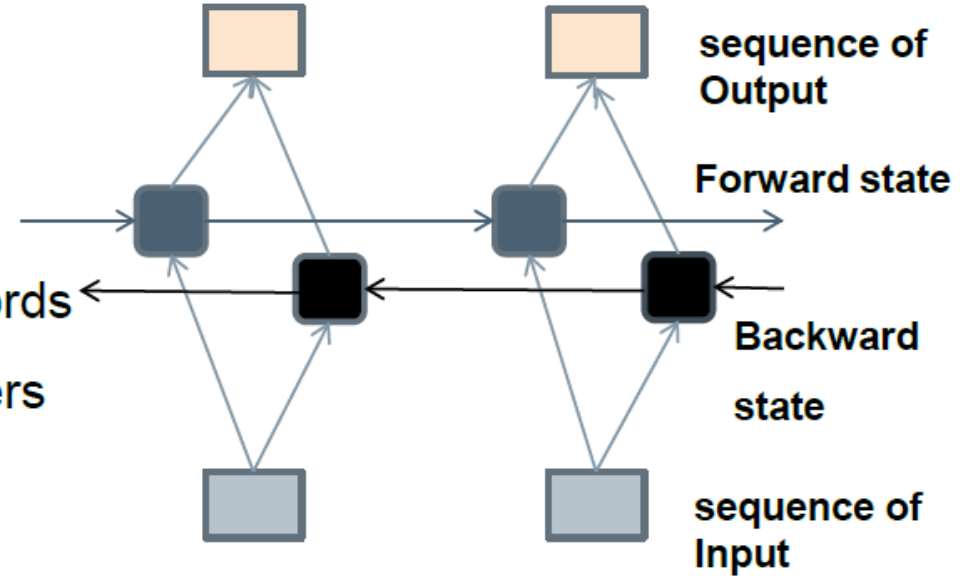
Example sentences:

Sentence1: "He said, ***Teddy*** bears are on sale"

Sentnce2: "He said, ***Teddy*** Roosevelt was a great President".

when we are looking at the word "Teddy" and the previous two words "He said", we might not be able to understand if the sentence refers to the President or Teddy bears.

Therefore, to resolve this ambiguity, we need to look ahead.



https://towardsdatascience.com/introduction-to-sequence-models-rnn-bidirectional-rnn-lstm-gru-73927ec9df15

Echo State Networks

- The recurrent weights mapping from h(t−1) to h(t) and the input weights mapping from x(t) to h(t) are some of the most difficult parameters to learn in a recurrent network.
- Solution → to set the recurrent weights such that the recurrent hidden units do a good job of capturing the history of past inputs and learn only the output weights.
- Used for echo state networks or ESNs

Leaky Units and Other Strategies for Multiple Time Scales

- Dealing with Long-term dependencies-
  - to design a model that operates at multiple time scales, so that some parts of the model operate at fine-grained time scales and can handle small details, while other parts operate at coarse time scales and transfer information from the distant past to the present more efficiently.
- Various strategies for building both fine and coarse time scales-
  - Addition of skip connections across time
  - “leaky units” that integrate signals with different time constants
  - removal of some of the connections used to model fine-grained time scales

Summary

- RNNs model sequential data
- Long term dependencies are a major problem in RNNs

  Solution:

  → careful weight initialization

  → LSTM/GRUs
- Gradients Explodes

  Solution: → Gradient norm clipping
- Regularization (Batch normalization and Dropout) and attention help
- Interesting direction to visualize and interpret RNN learning

References

- Ian Goodfellow, Yoshua Bengio, Aaron Courville, “Deep Learning”, MIT Press 2017
- Josh Patterson and Adam Gibson, “Deep Learning A Practitioner’s Approach”, O’Reilly Media, 2017
- RNN lecture (Ian Goodfellow) https://www.youtube.com/watch?v=ZVN14xYm7JA
- Andrew Ng lecture on RNN: https://www.coursera.org/lecture/nlp-sequence-models/why-sequence-models-0h7gT
- http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- https://towardsdatascience.com/introduction-to-sequence-models-rnn-bidirectional-rnn-lstm-gru-73927ec9df15