Deep Learning for Applications - II

심스리얼리티 임직원을 위한 딥러닝 교육 과정

Dr. Kyong-Ha Lee (kyongha@kisti.re.kr)

KiSTi 한국과학기술정보연구원
Korea Institute of Science and Technology Information

Contents

Class2 : 뉴럴 네트워크에 대한 기초를 토대로 좀더 복잡한 모델 (RNN, Attention, Transformer, LLM)들을 이해합니다.

A Few Quates about Machine Learning

- **Motivation :**
  - **Why do we need 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 in different lengths in different examples**

Example:

Sentence1: Pankaj lives in Munich

Sentence2: Pankaj Gupta lives in Munich DE

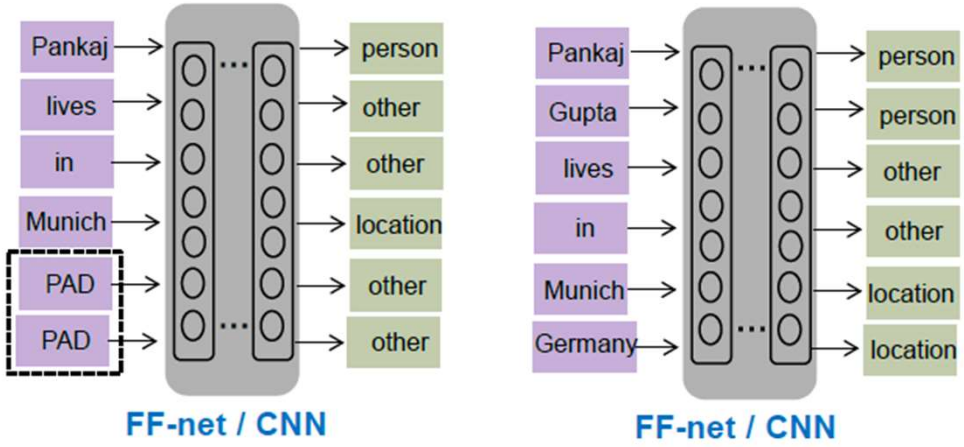
Motivation: Need for Sequential Modeling

- **Inputs, outputs can be in different lengths in different examples**

Example:

Sentence1: Pankaj lives in Munich

Sentence2: Pankaj Gupta lives in Munich DE



Additional word 'PAD' i.e., padding

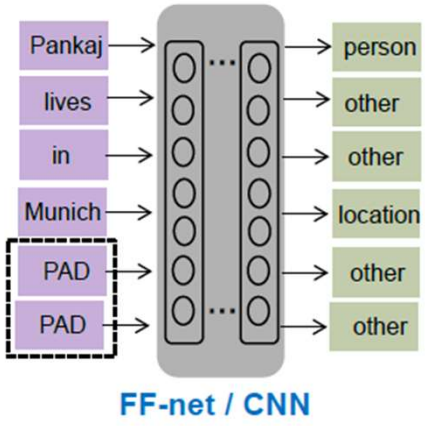
Motivation: Need for Sequential Modeling

- **Inputs, outputs can be in different lengths in different examples**

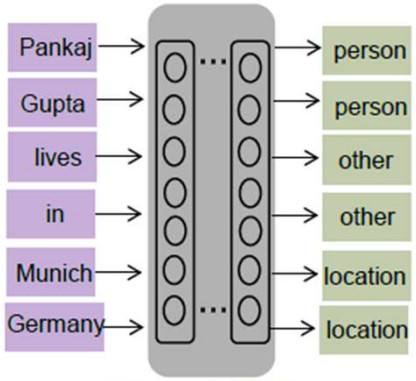
Example:

Sentence1: Pankaj lives in Munich

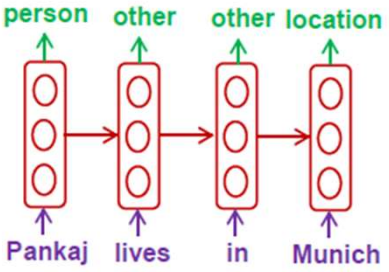
Sentence2: Pankaj Gupta lives in Munich DE



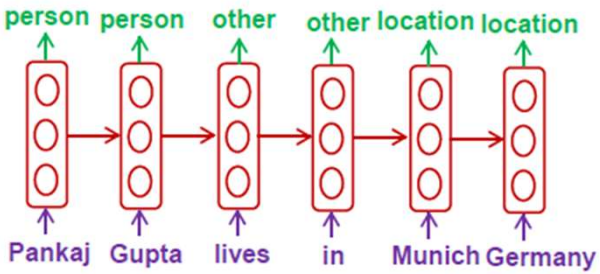
FF-net / CNN



FF-net / CNN



Models variable length sequences



Sequential model: RNN

Motivation: Need for Sequential Modeling

- **Shared features learned across different positions or time steps**

Example:

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

Sentence2: Bear falls into market territory → *UNK*

Same uni-gram statistics

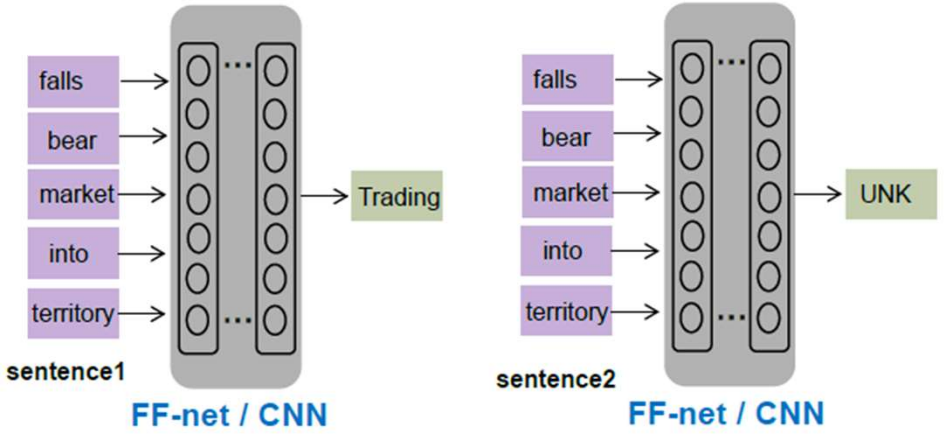
Motivation: Need for Sequential Modeling

- **Shared 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

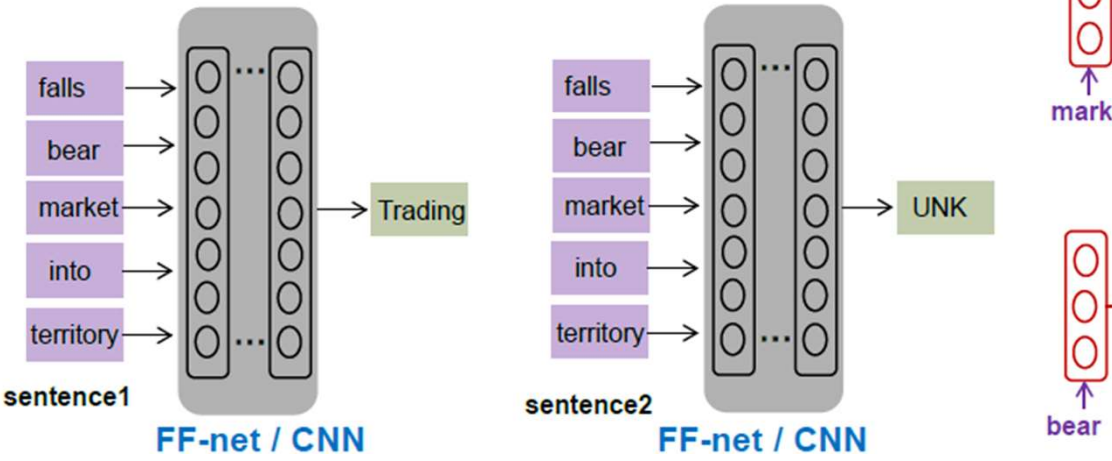
Motivation: Need for Sequential Modeling

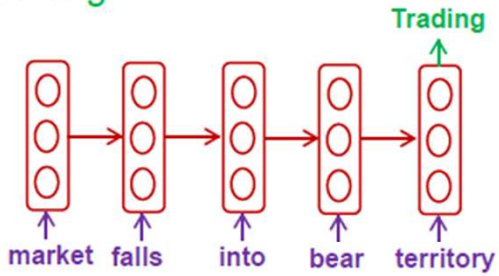
- **Shared features learned across different positions or time steps**

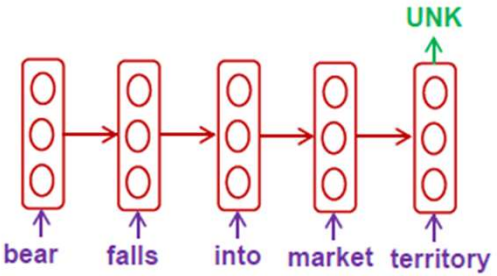
Example:

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

Sentence2: Bear falls into market territory → *UNK*







Language concepts,

Word ordering,

Syntactic & semantic information

Sequential model: RNN

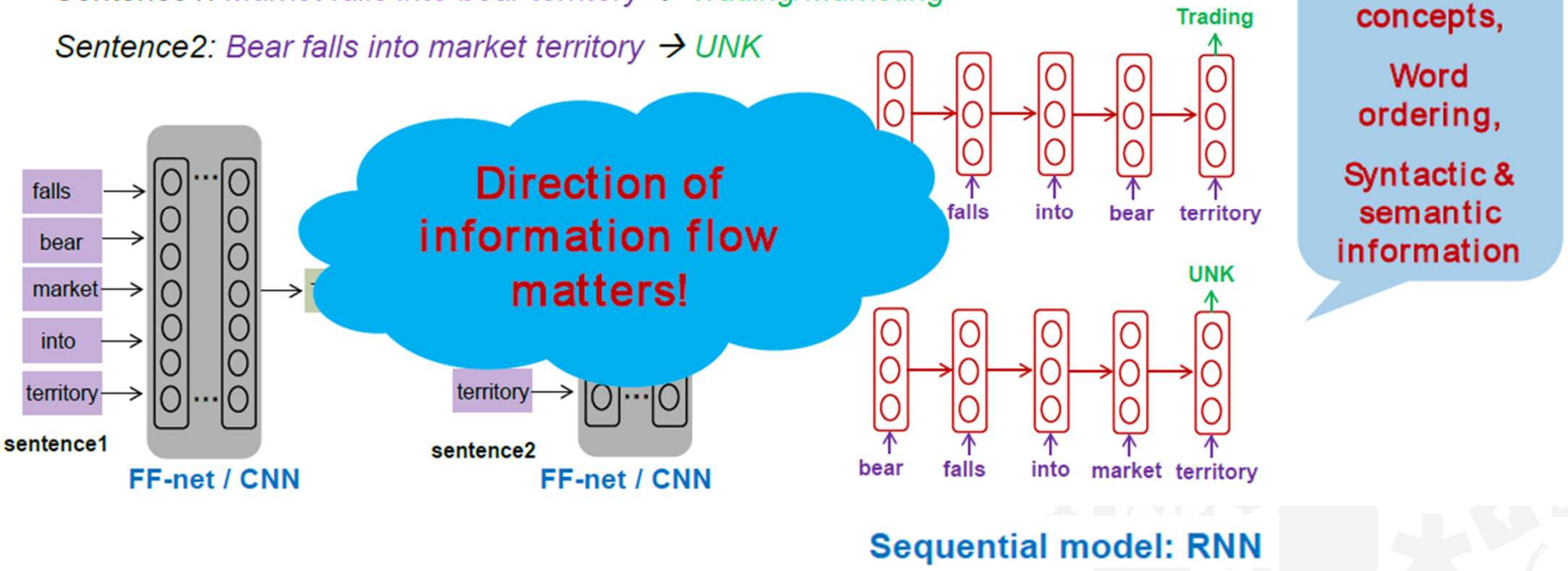
Motivation: Need for Sequential Modeling

- **Shared features learned across different positions or time steps**

Example:

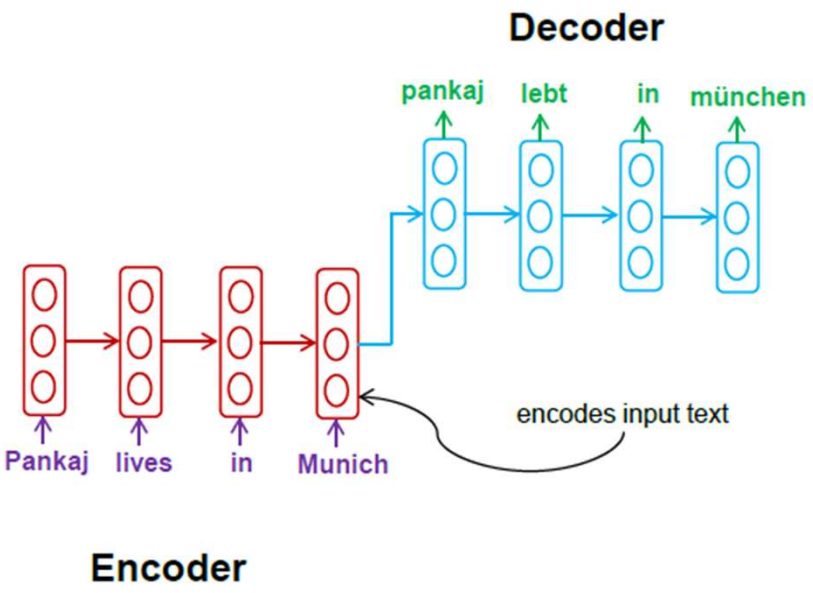
Sentence1: *Market falls into bear territory* → *Trading/Marketing*

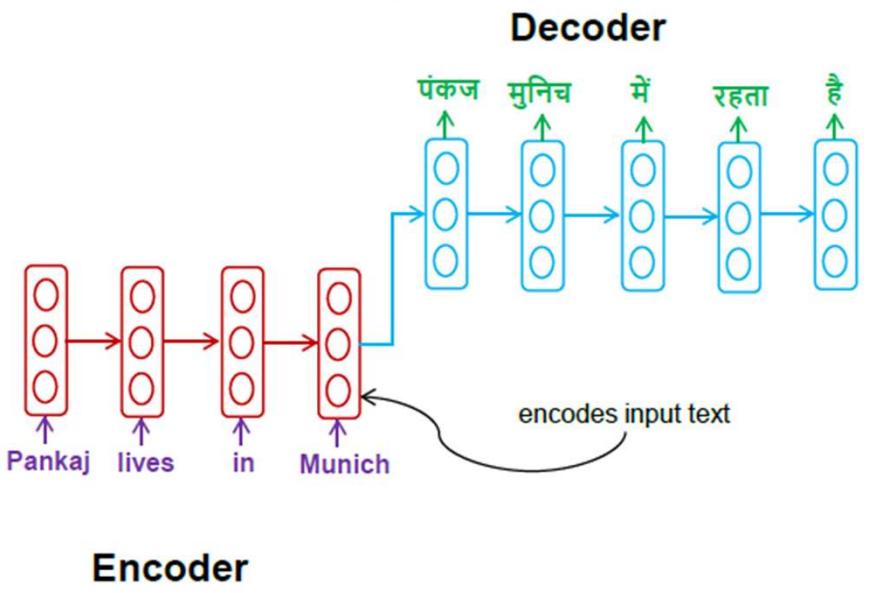
Sentence2: Bear falls into market territory → *UNK*



Motivation: Need for Sequential Modeling

- **Machine Translation:** different input & output sizes, incurring sequential patterns





Motivation: Need for Sequential Modeling

- **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 your 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

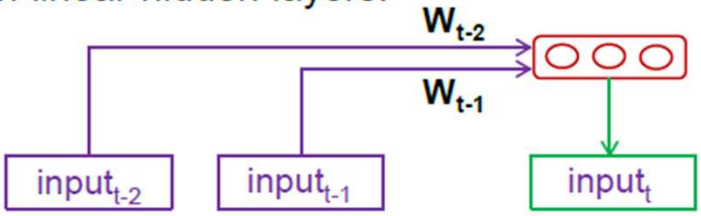
Memory-less Models

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

Term 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

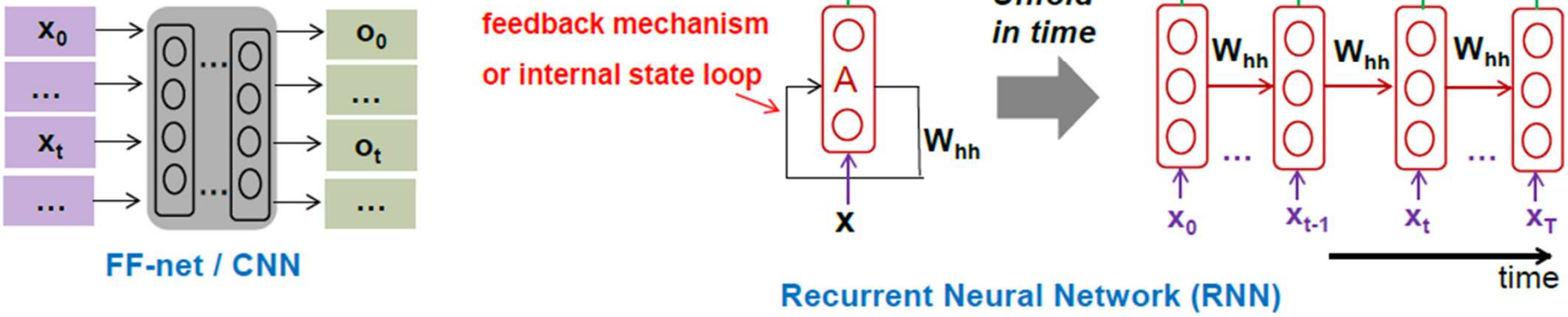
Foundation of Recurrent Neural Networks

Goal

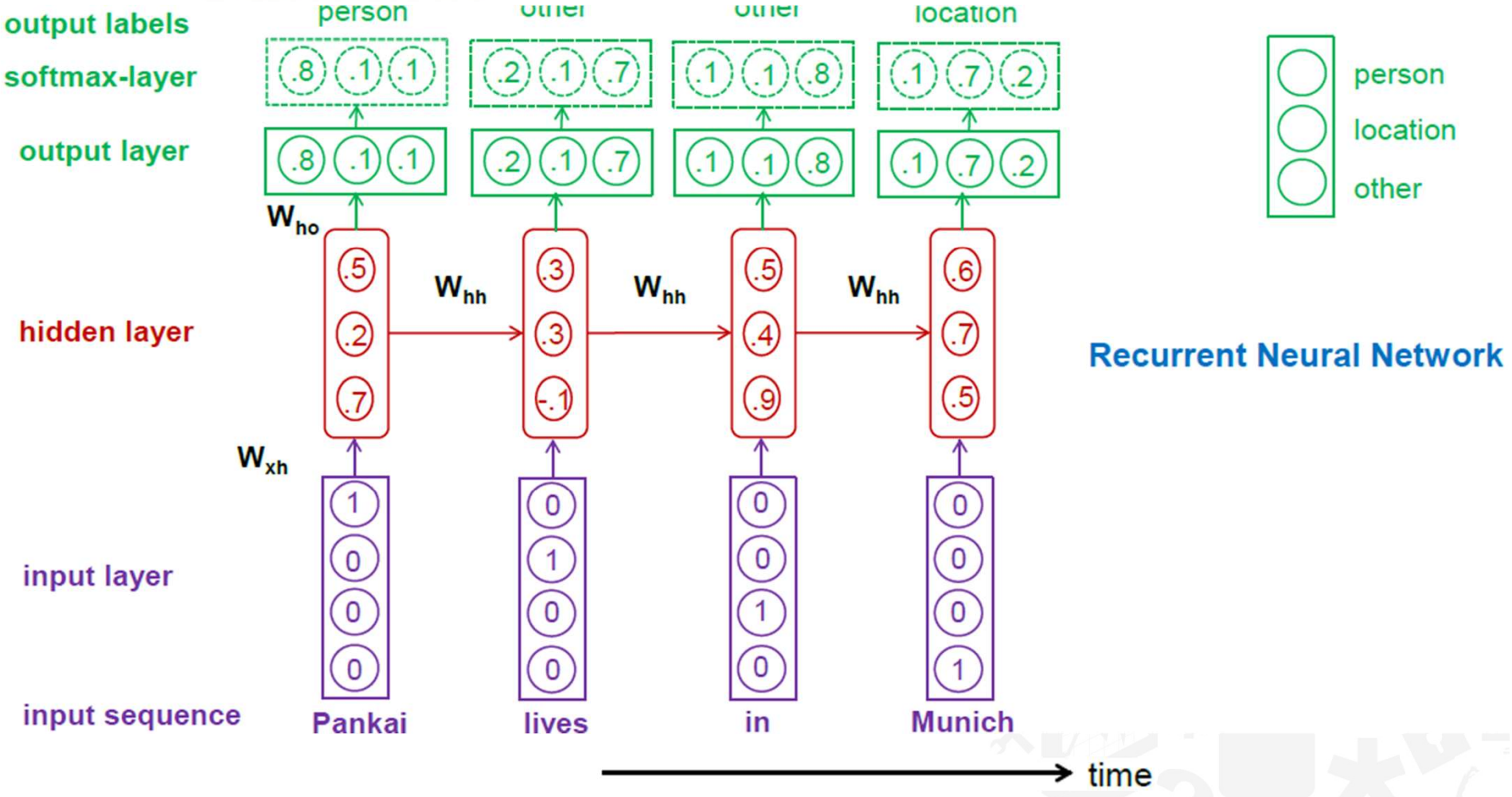
- model long term dependencies
- connect previous information to the present task
- model sequence of events with loops, allowing information to persist

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

Sequential data needs a **Feedback Mechanism**.



Foundation of Recurrent Neural Networks



(Vanilla) Recurrent Neural Network

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

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

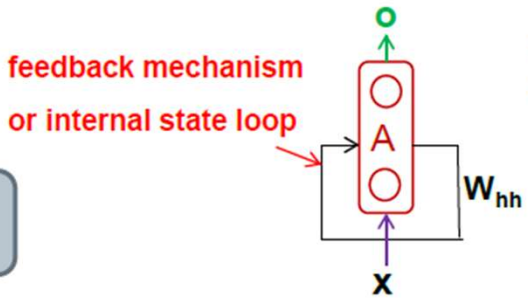
new hidden state at time step, t

some function with parameters W_{hh} W_{xh}

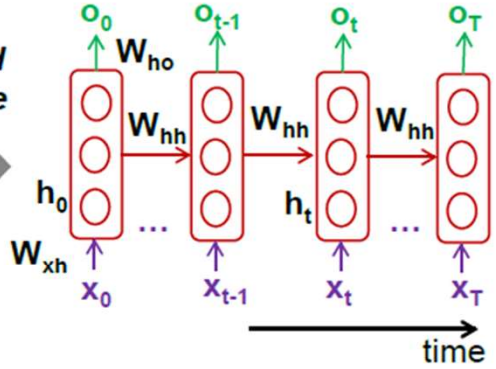
old hidden state at time step, t-1

Input vector at time step, t

feedback mechanism or internal state loop



Unfold in time



Vanilla Recurrent Neural Network (RNN)

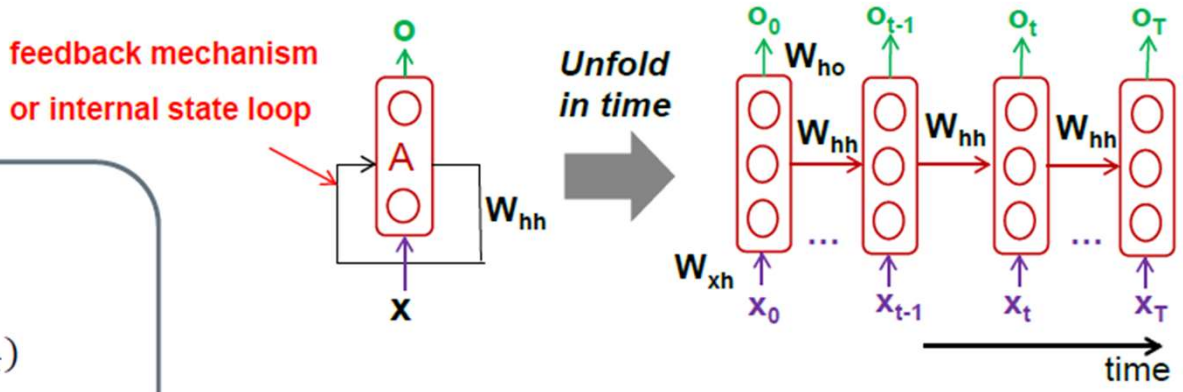
Remark: The same function *g* and same set of parameters **W** are used at every time step

(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.

RNN : Probablistic Interpretation

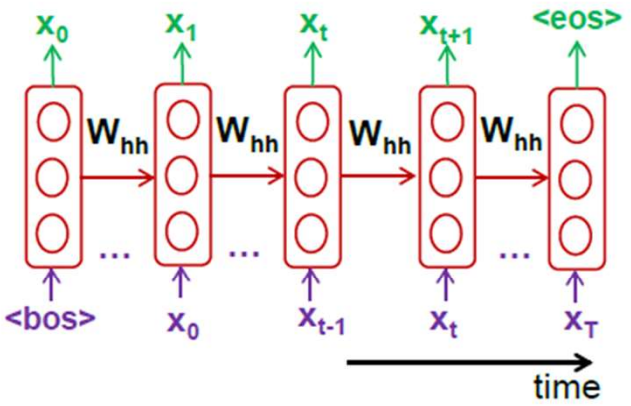
RNN as a generative model

- 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)

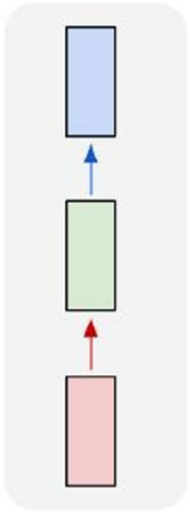
- Think of the output as the probability distribution of the $\mathbf{x}_t$ given the previous ones in the sequence
- Training: Computing probability of the sequence and Maximum likelihood training

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

Details: https://www.cs.cmu.edu/~epxing/Class/10708-17/project-reports/project10.pdf

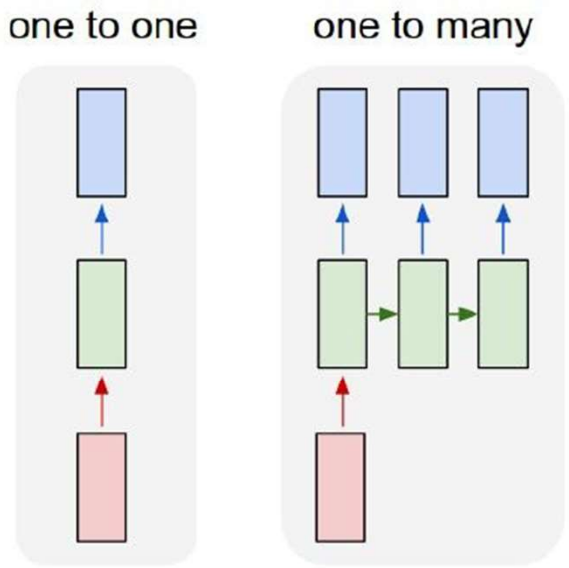
RNN : Computational Graphs

one to one



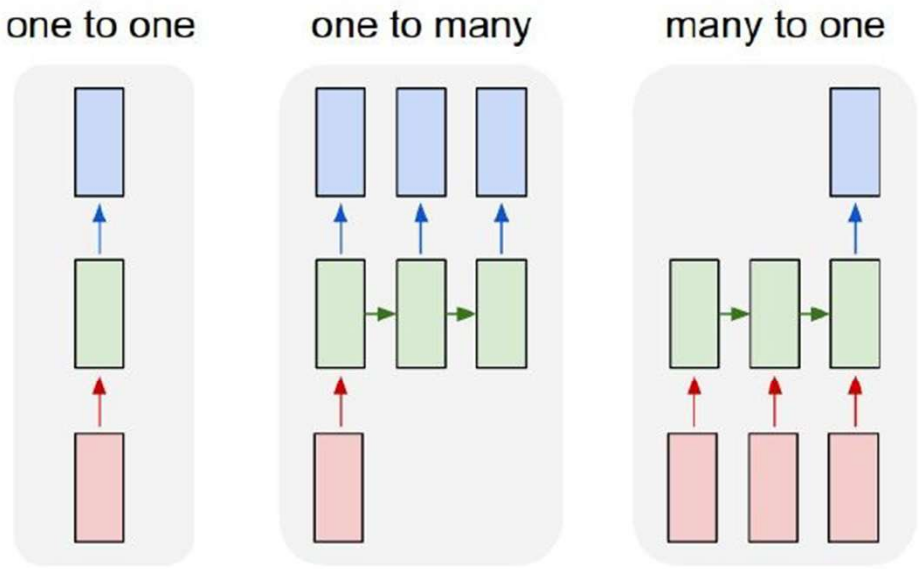
Vanilla Neural Networks

RNN : Computational Graphs



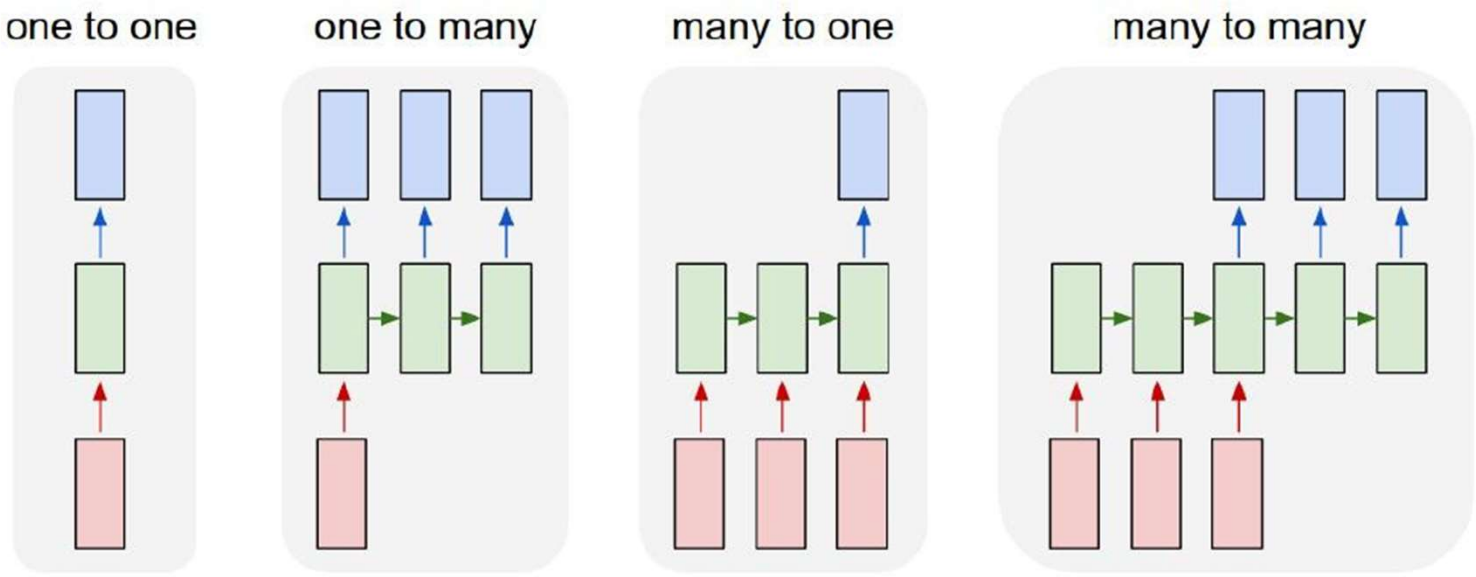
e.g. **Image Captioning**
image -> sequence of words

RNN : Computational Graphs



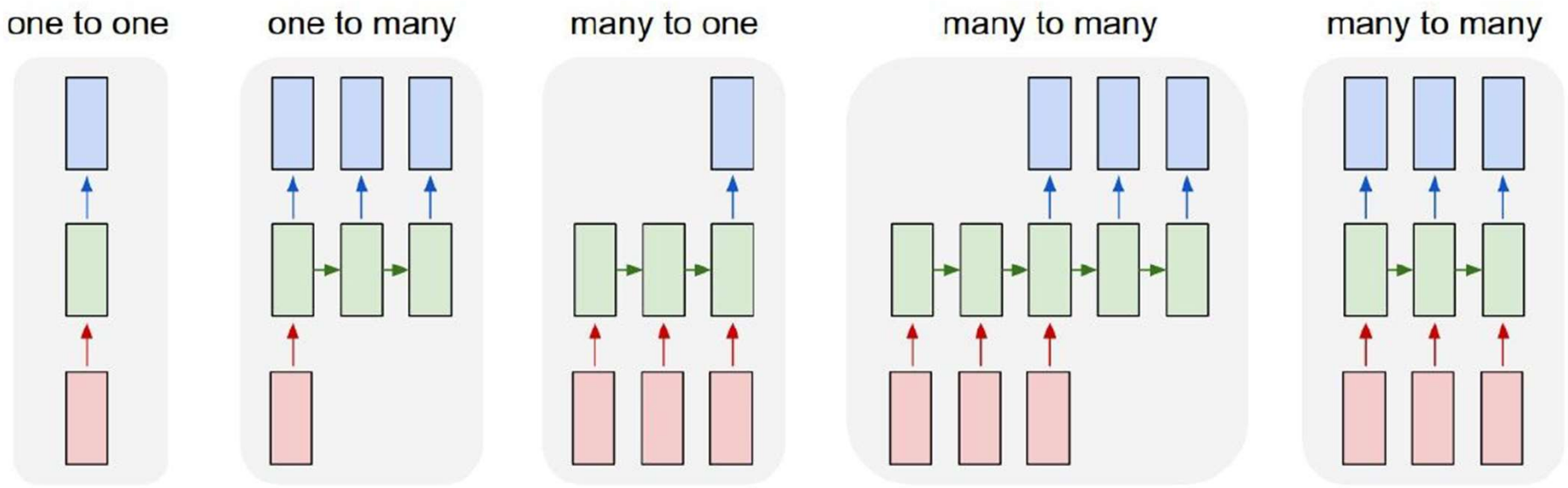
e.g. **action prediction**
sequence of video frames -> action class

RNN : Computational Graphs

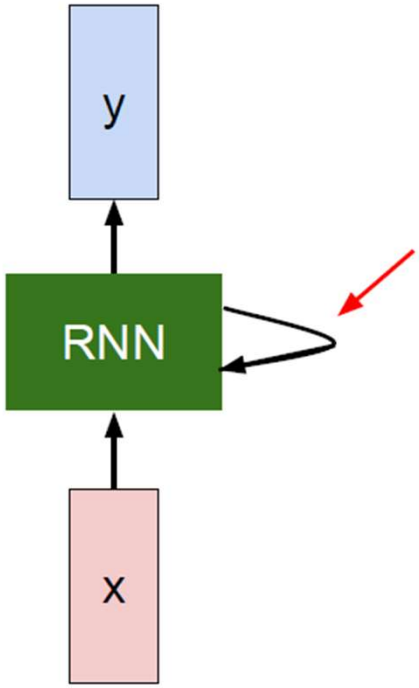


E.g. **Video Captioning**
Sequence of video frames -> caption

RNN : Computational Graphs

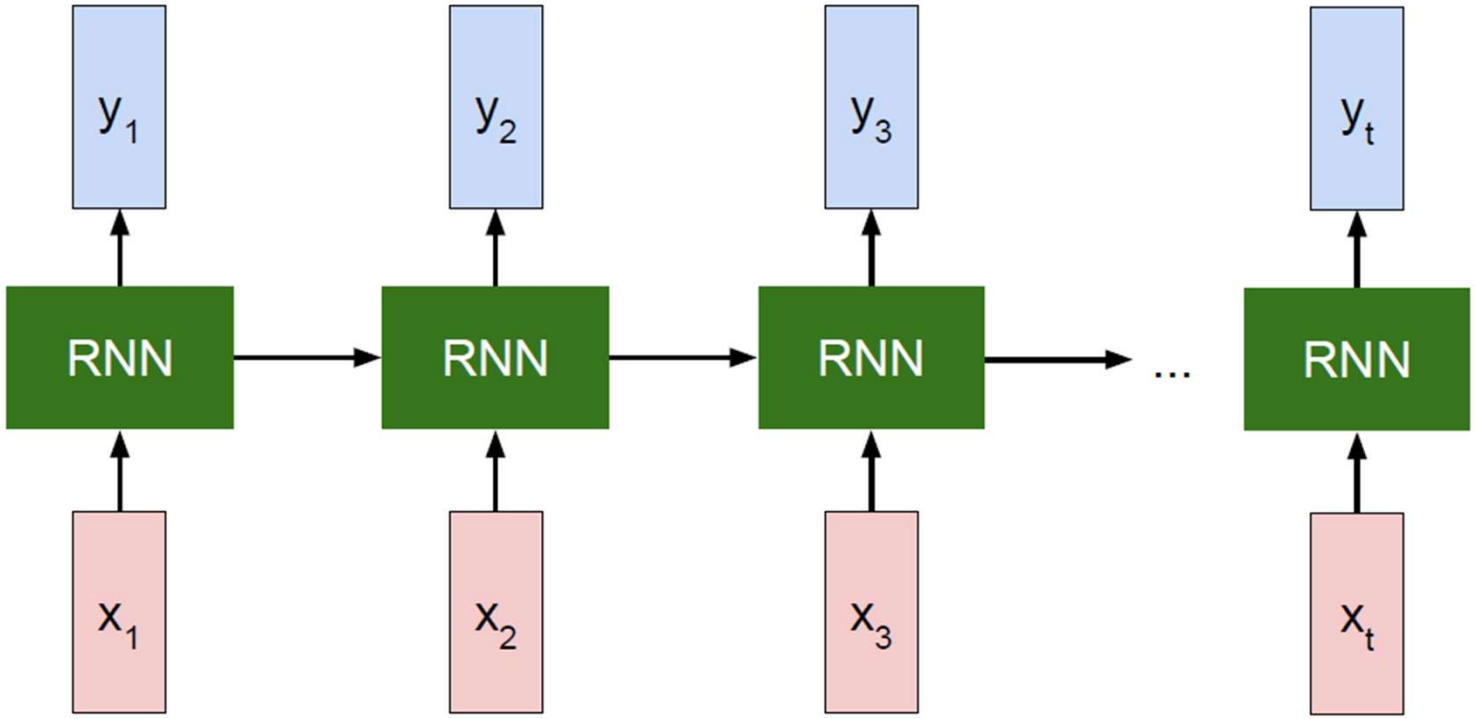


Recurrent Neural Neetwork



Key idea: RNNs have an "internal state" that is updated as a sequence is processed

RNN Unrolled



RNN hidden state update

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

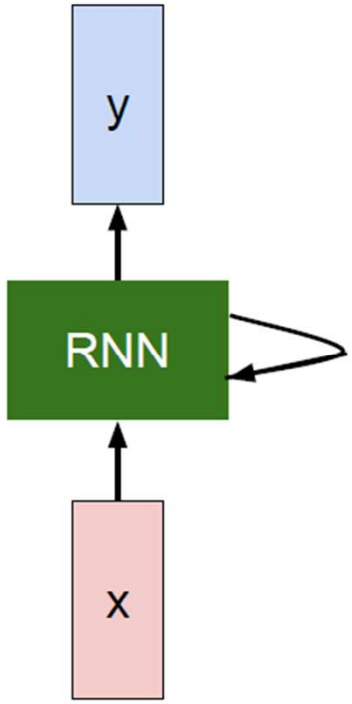
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

input vector at some time step



RNN output generation

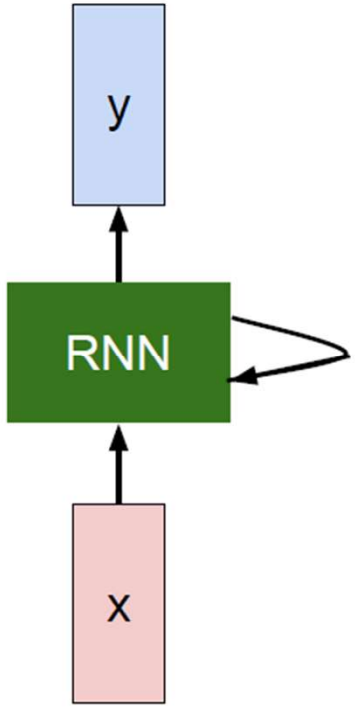
We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$y_t = f_{W_{hy}}(h_t)$$

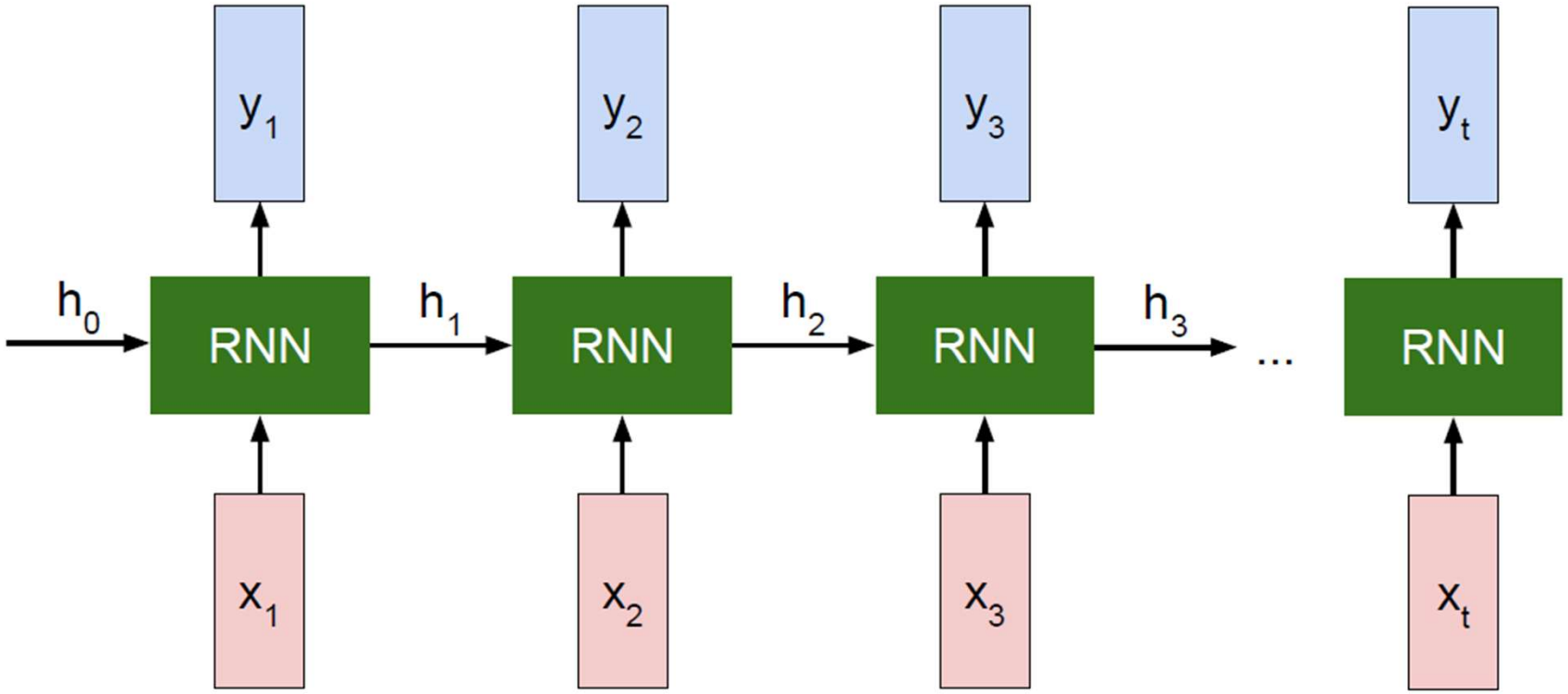
output

new state

another function with parameters W_o



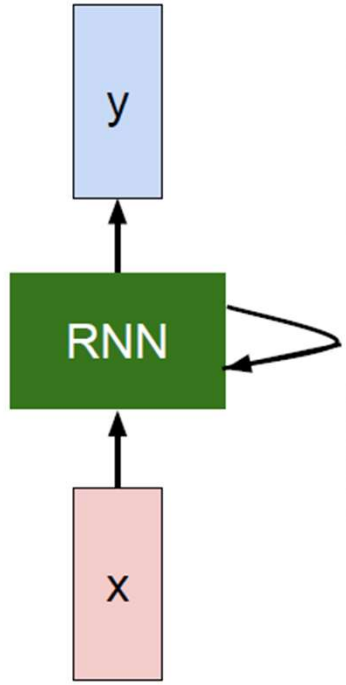
Recurrent Neural Network



Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

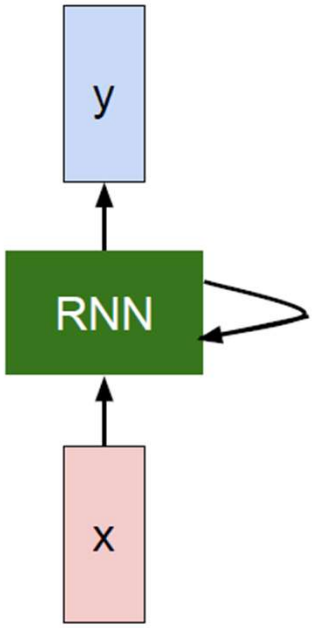
$$h_t = f_W(h_{t-1}, x_t)$$



Notice: the same function and the same set of parameters are used at every time step.

Vanilla RNN

The state consists of a single *"hidden"* vector **h**:



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

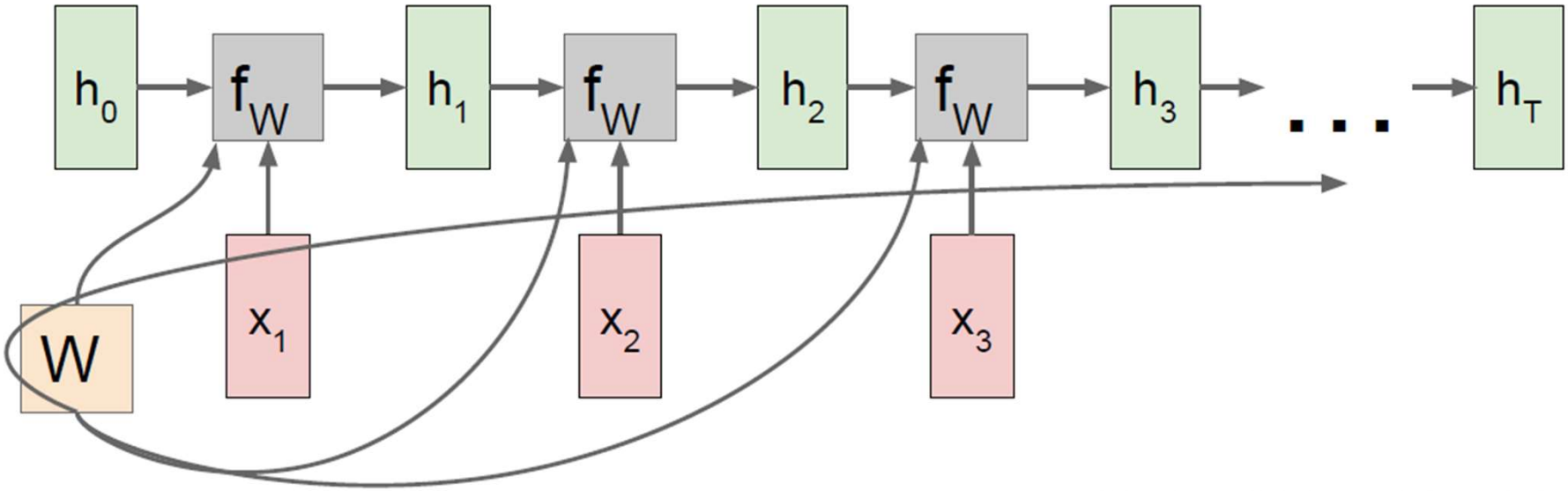
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

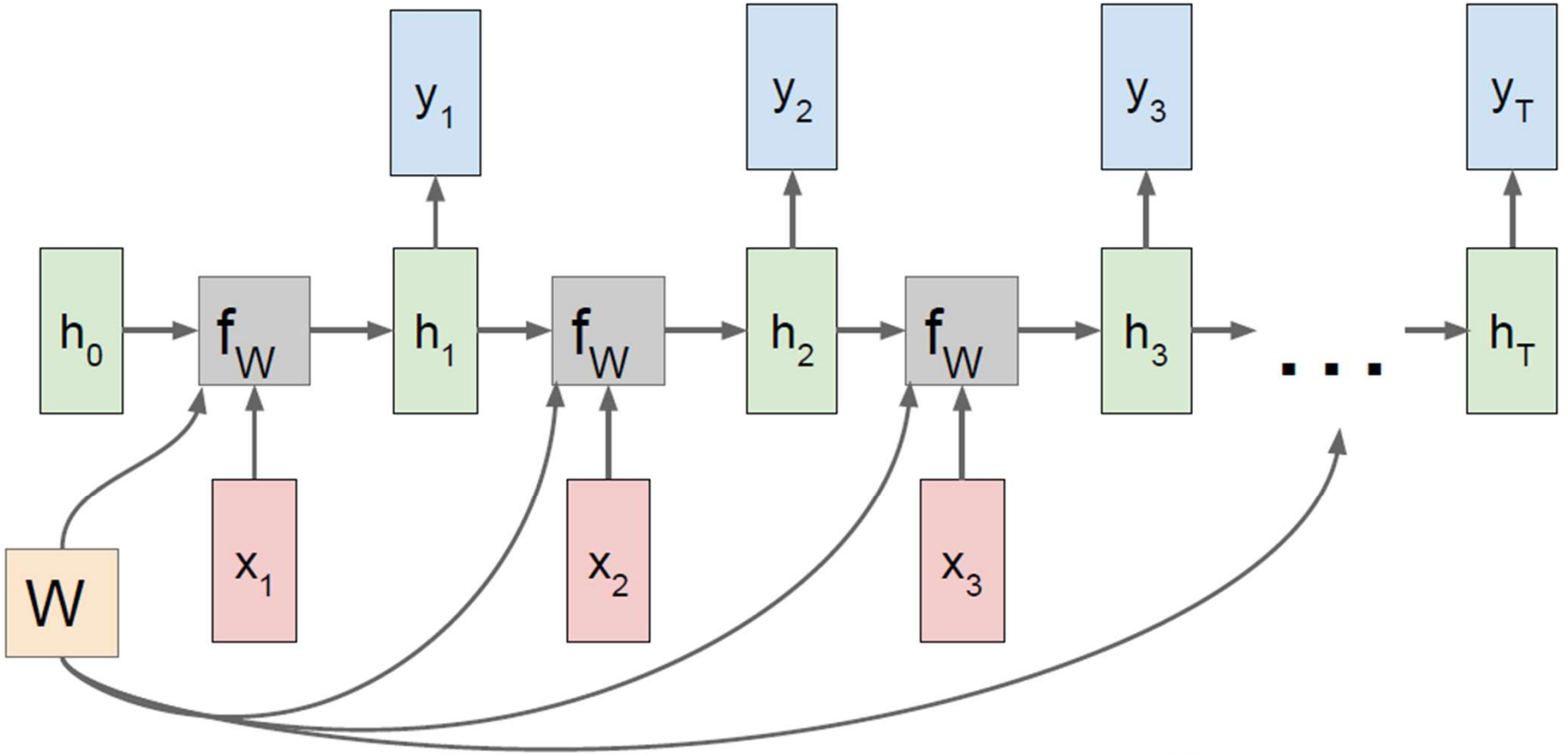
Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

RNN: Computational Graphs

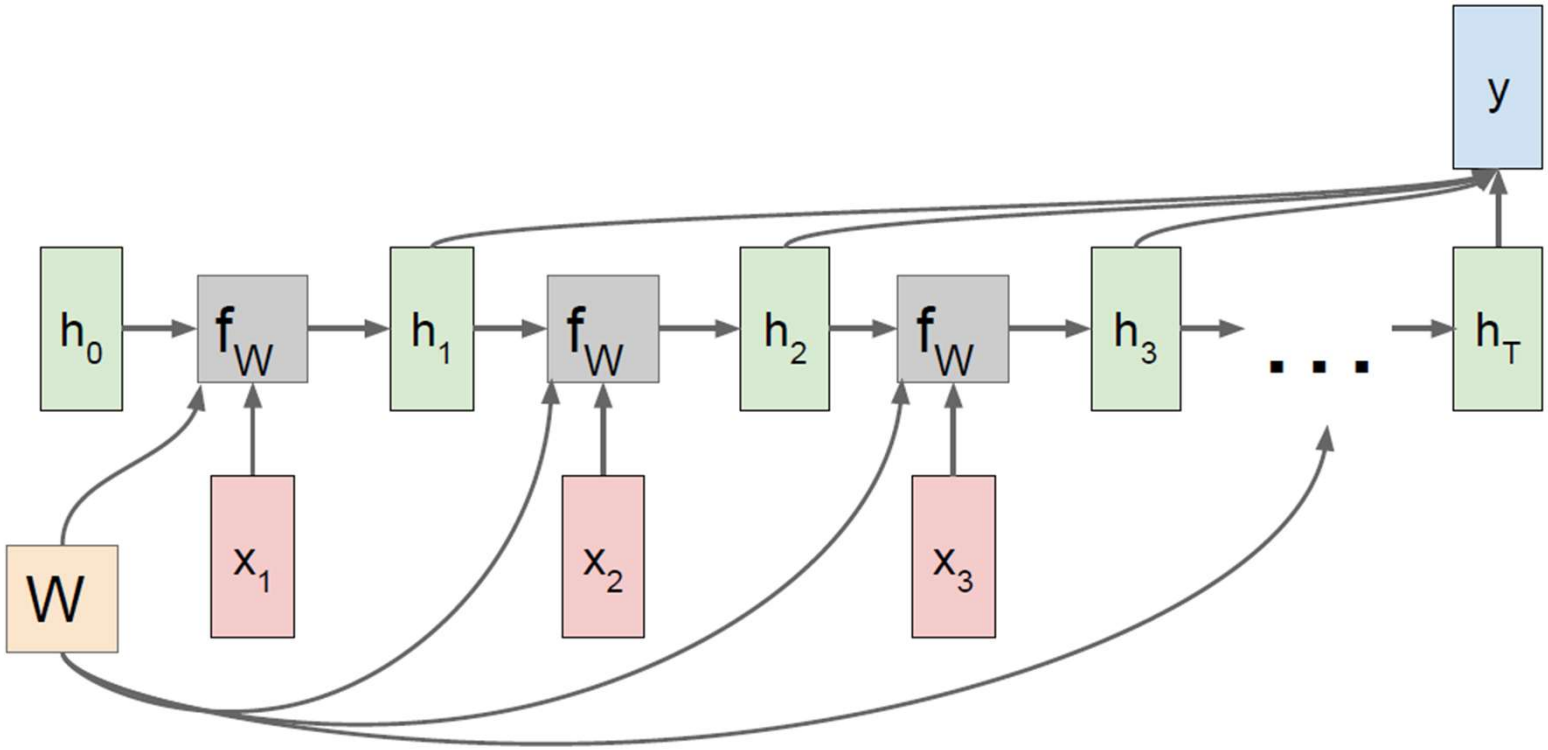
Re-use the same weight matrix at every time-step



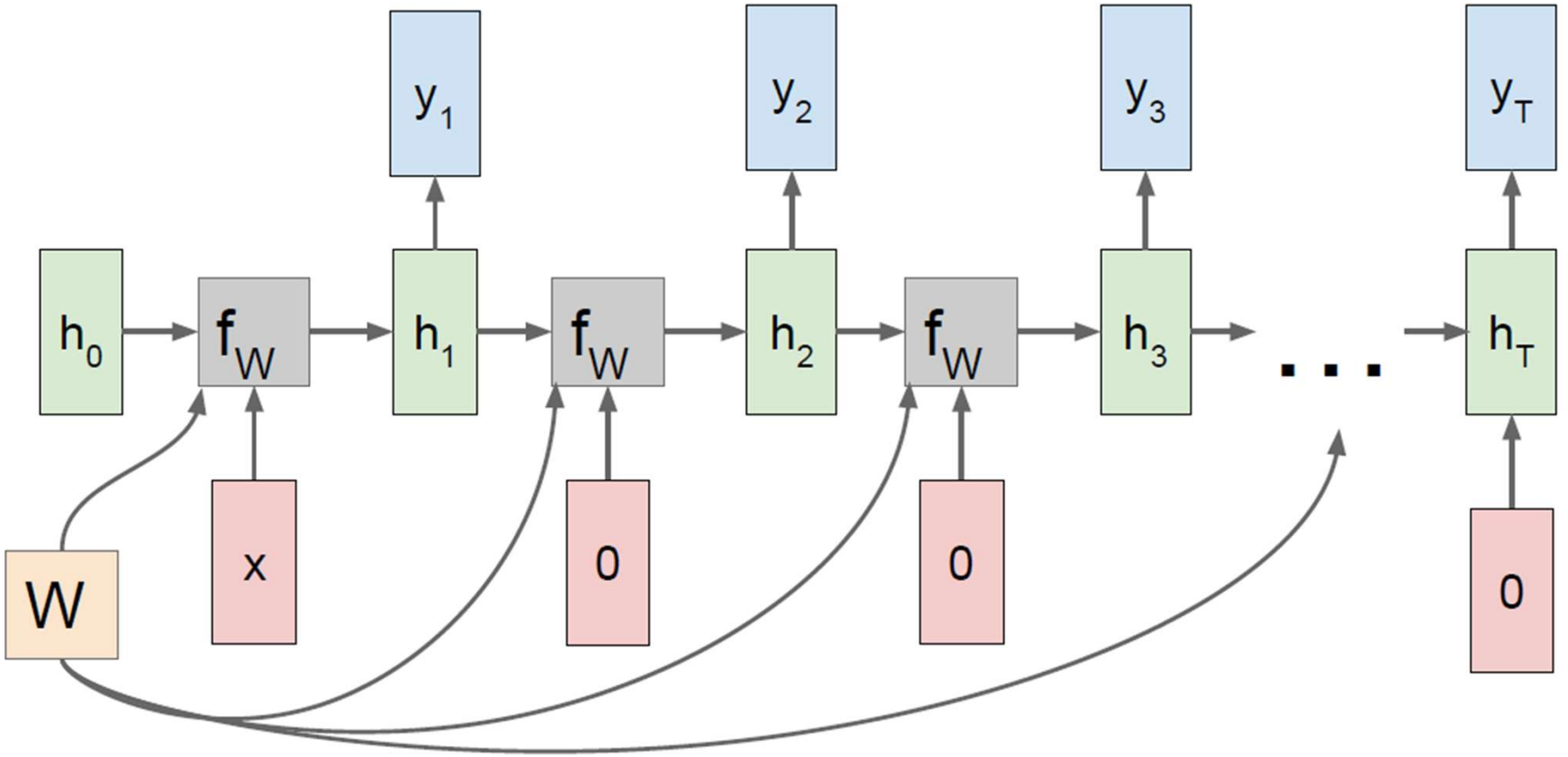
RNN: Computational Graphs: Many to Many



RNN: Computational Graphs: Many to One



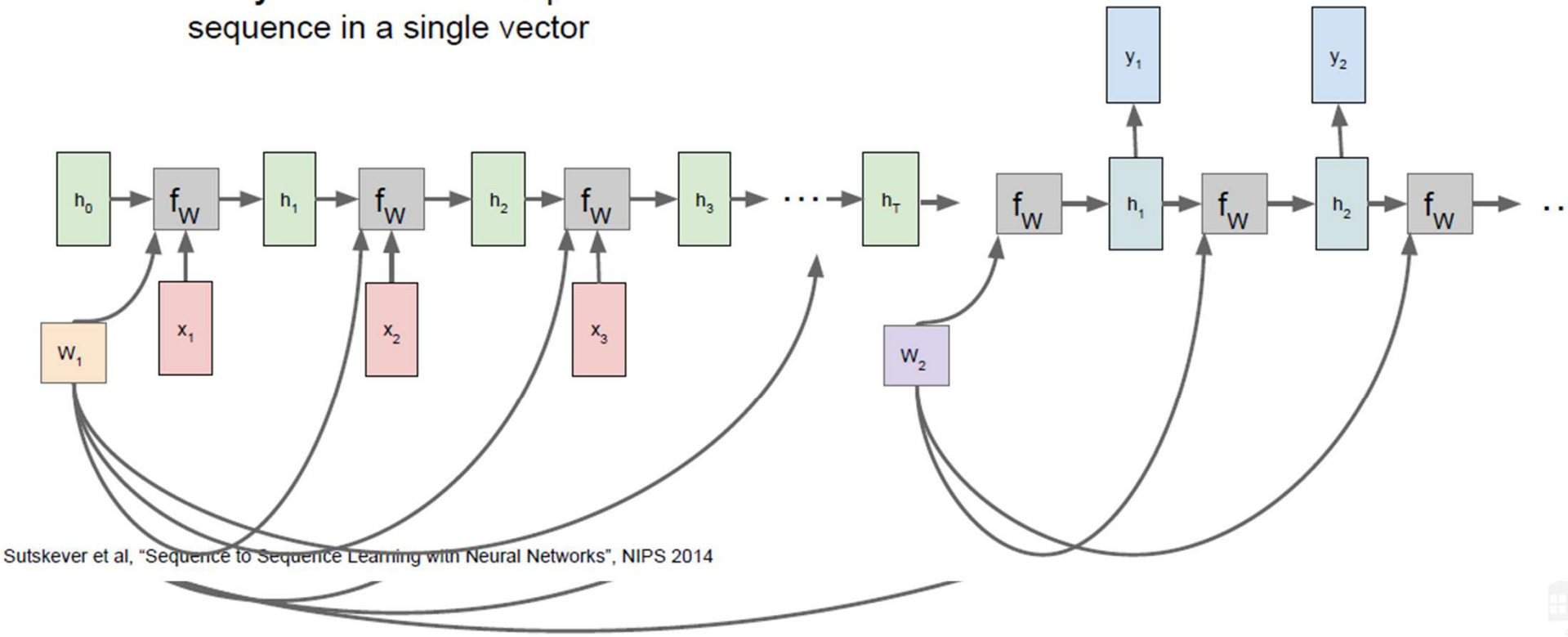
RNN: Computational Graphs: One to Many



RNN: Computational Graphs: One to Many

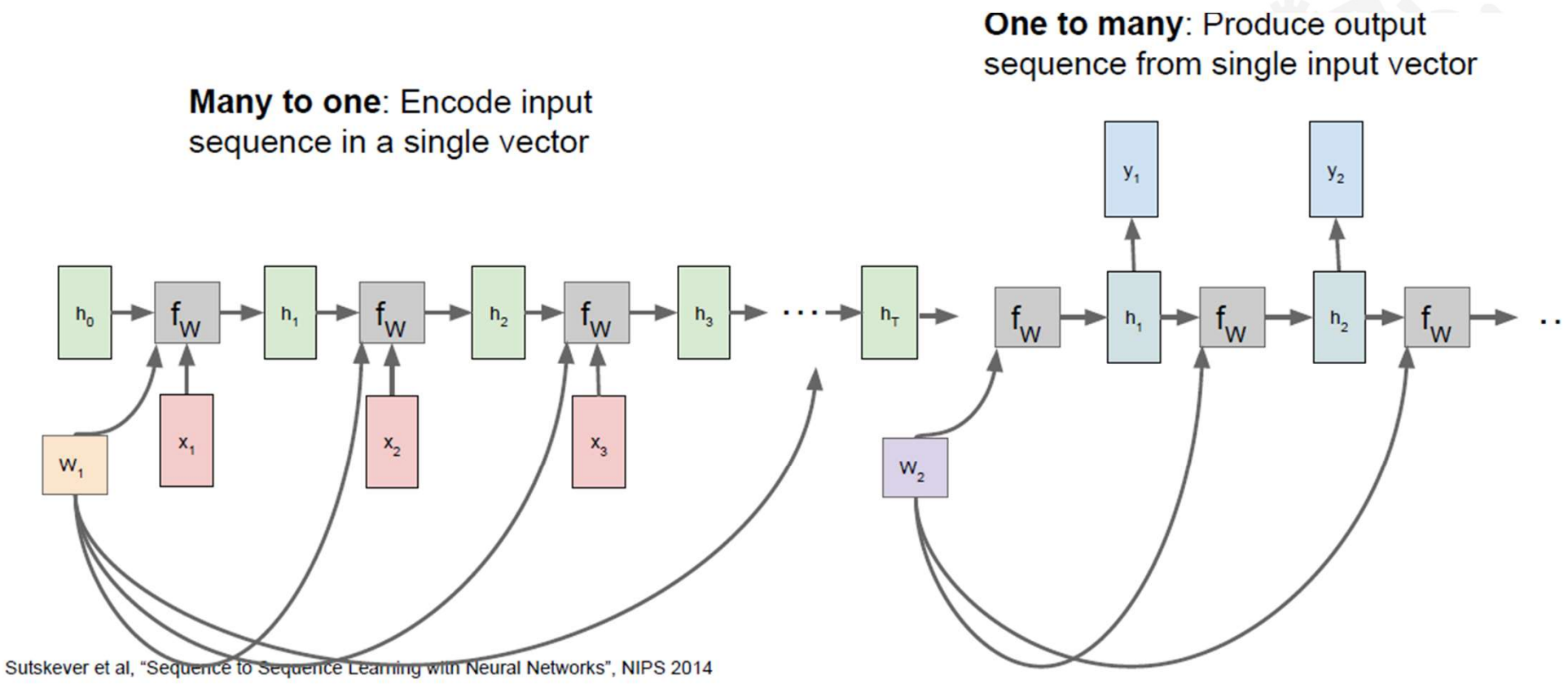
One to many: Produce output sequence from single input vector

Many to one: Encode input sequence in a single vector



Sutskever et al, "Sequence to Sequence Learning with Neural Networks", NIPS 2014

Seq. to Seq. : Many-to-one + one-to-many

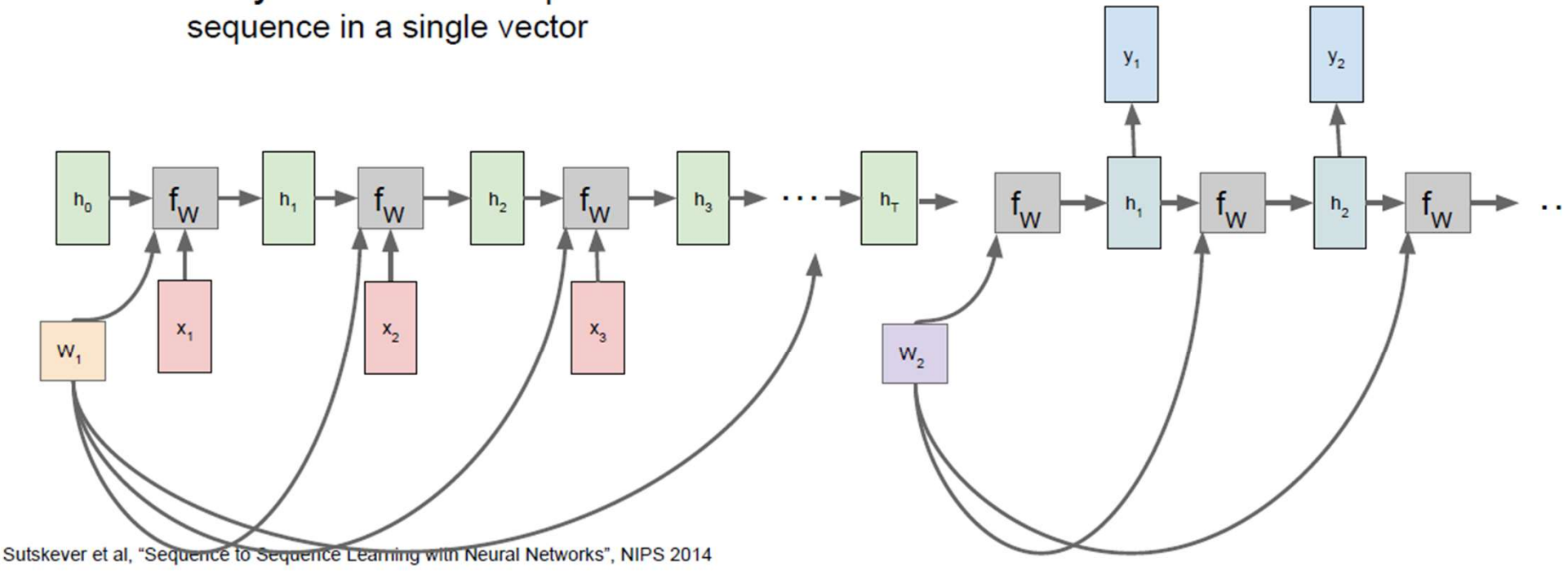


Sutskever et al, "Sequence to Sequence Learning with Neural Networks", NIPS 2014

Seq. to Seq. : Many-to-one + one-to-many

Many to one: Encode input sequence in a single vector

One to many: Produce output sequence from single input vector



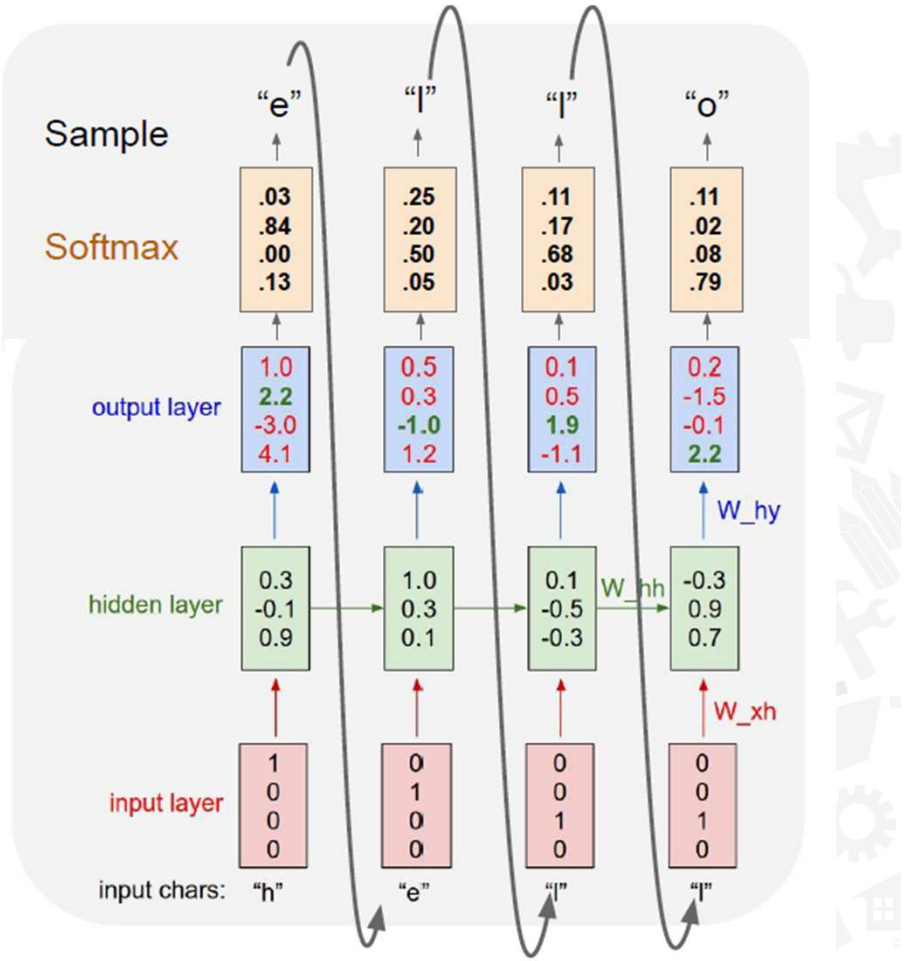
Sutskever et al, "Sequence to Sequence Learning with Neural Networks", NIPS 2014

RNN Example

Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model

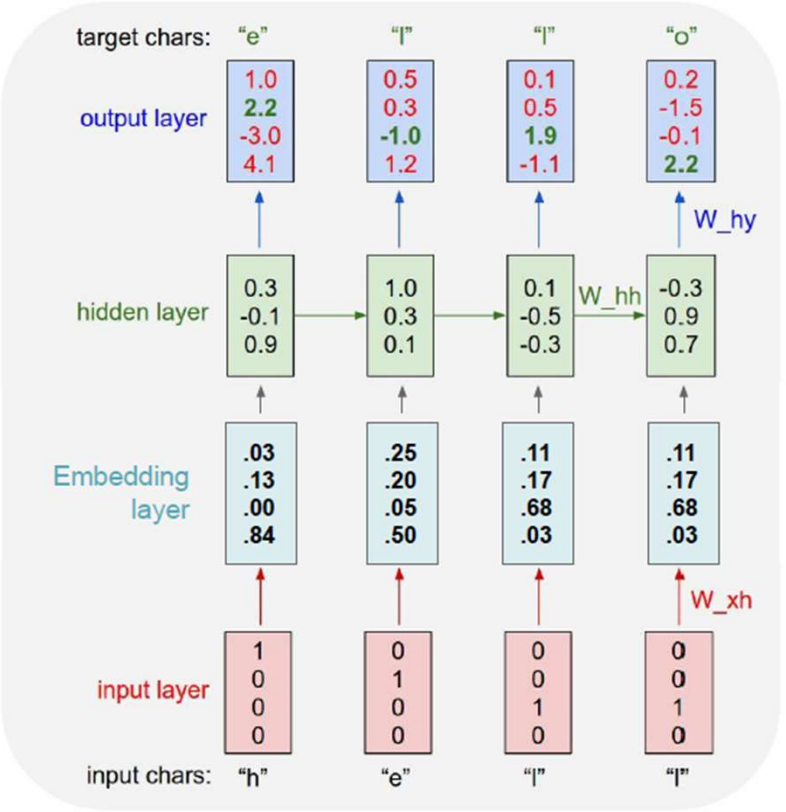


RNN Example

Example: Character-level Language Model Sampling

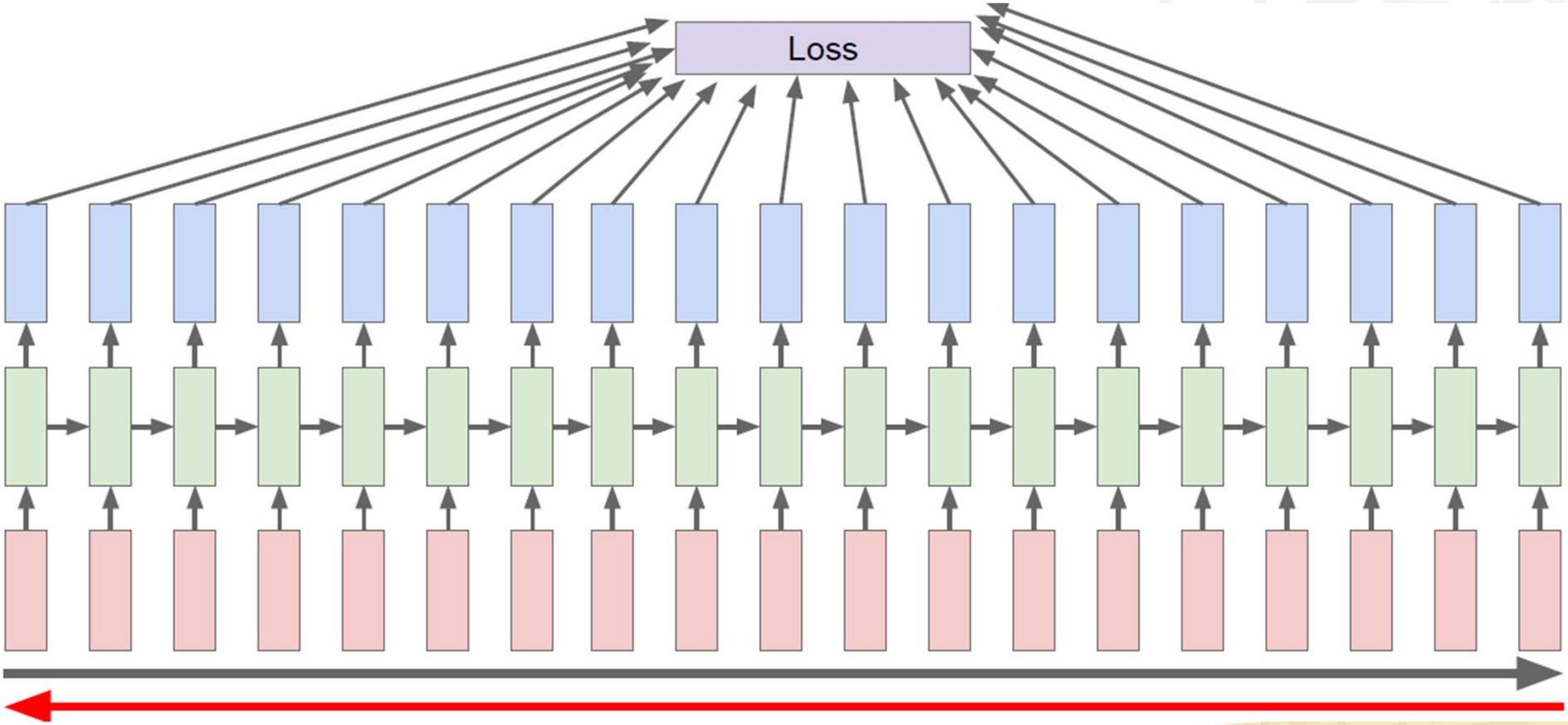
$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{14} \\ w_{31} & w_{32} & w_{33} & w_{14} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \end{bmatrix}$$

Matrix multiply with a one-hot vector just extracts a column from the weight matrix. We often put a separate **embedding** layer between input and hidden layers.

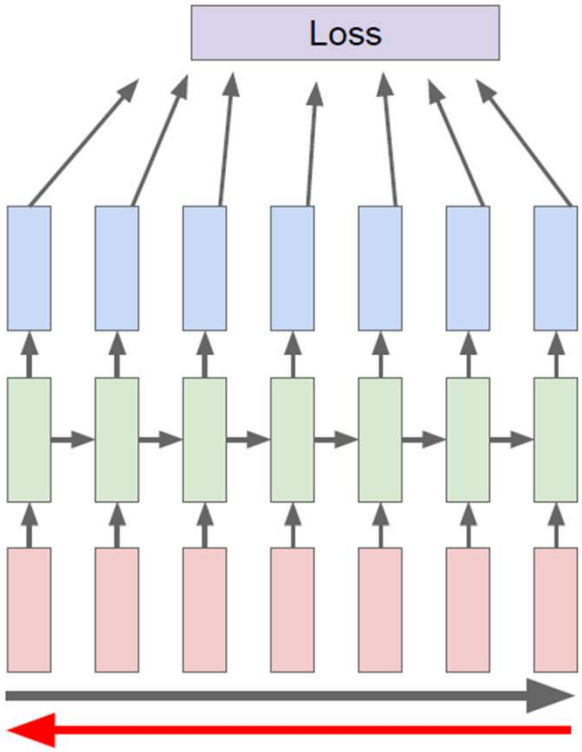


Backpropagation through time

Forward through entire sequence to compute loss,
then backward through entire sequence to compute gradient

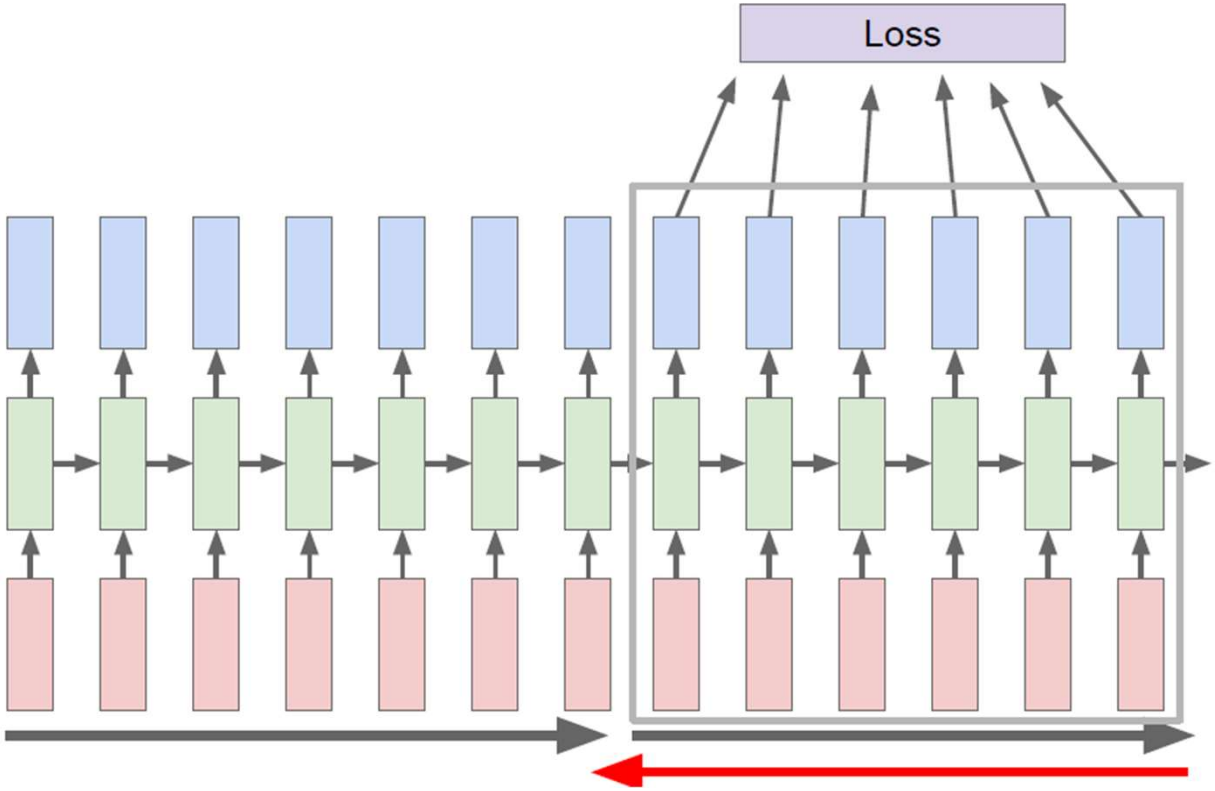


Truncated Backpropagation through time



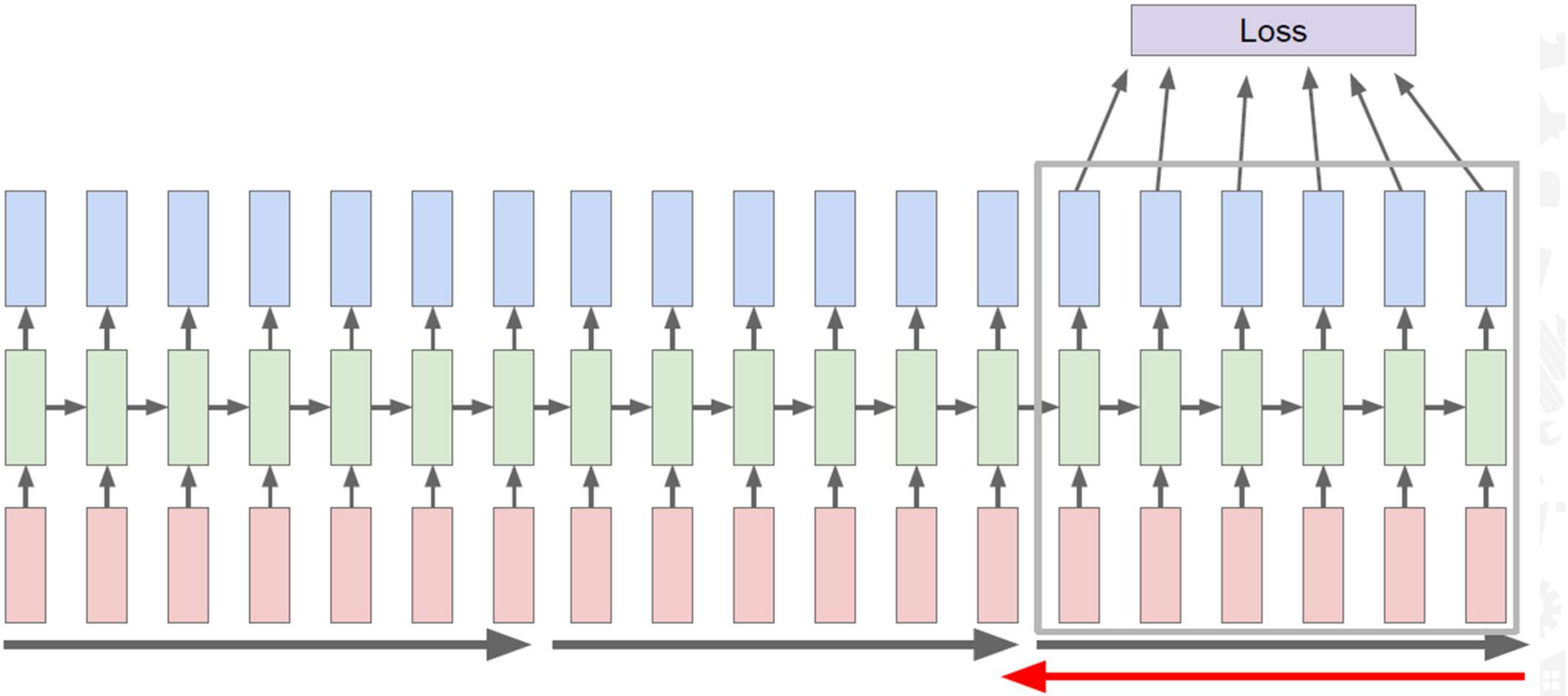
Run forward and backward through chunks of the sequence instead of whole sequence

Truncated Backpropagation through time



Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

Truncated Backpropagation through time



RNN trade-offs

RNN Advantages:

- Can process any length input
- Computation for step *t* can (in theory) use information from many steps back
- Model size doesn't increase for longer input
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

RNN Disadvantages:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back

Example: Image Captioning

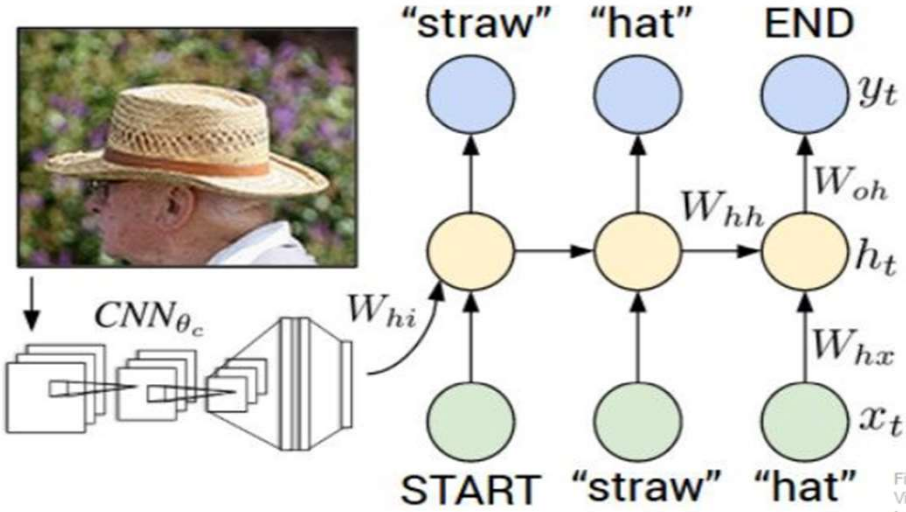


Figure from Karpathy et a, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015; figure copyright IEEE, 2015.
Reproduced for educational purposes.

Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei
Show and Tell: A Neural Image Caption Generator, Vinyals et al.
Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.
Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

Example: Image Captioning

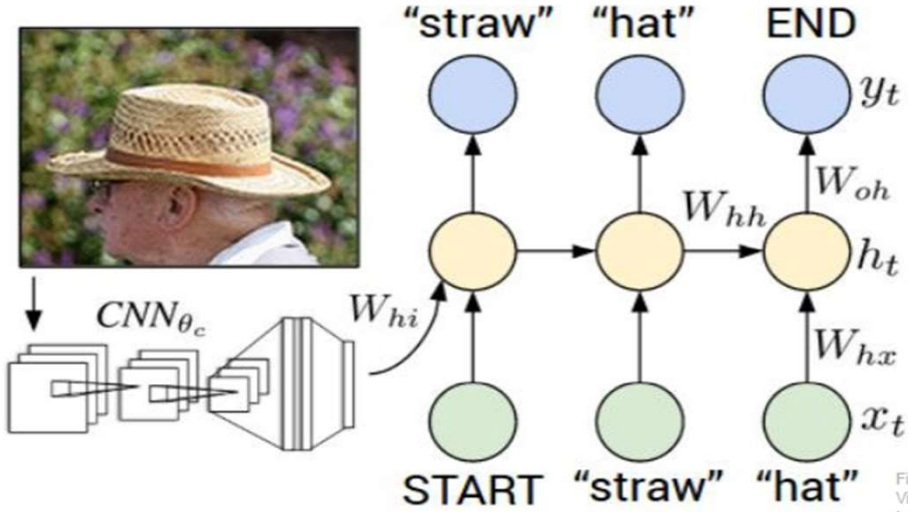
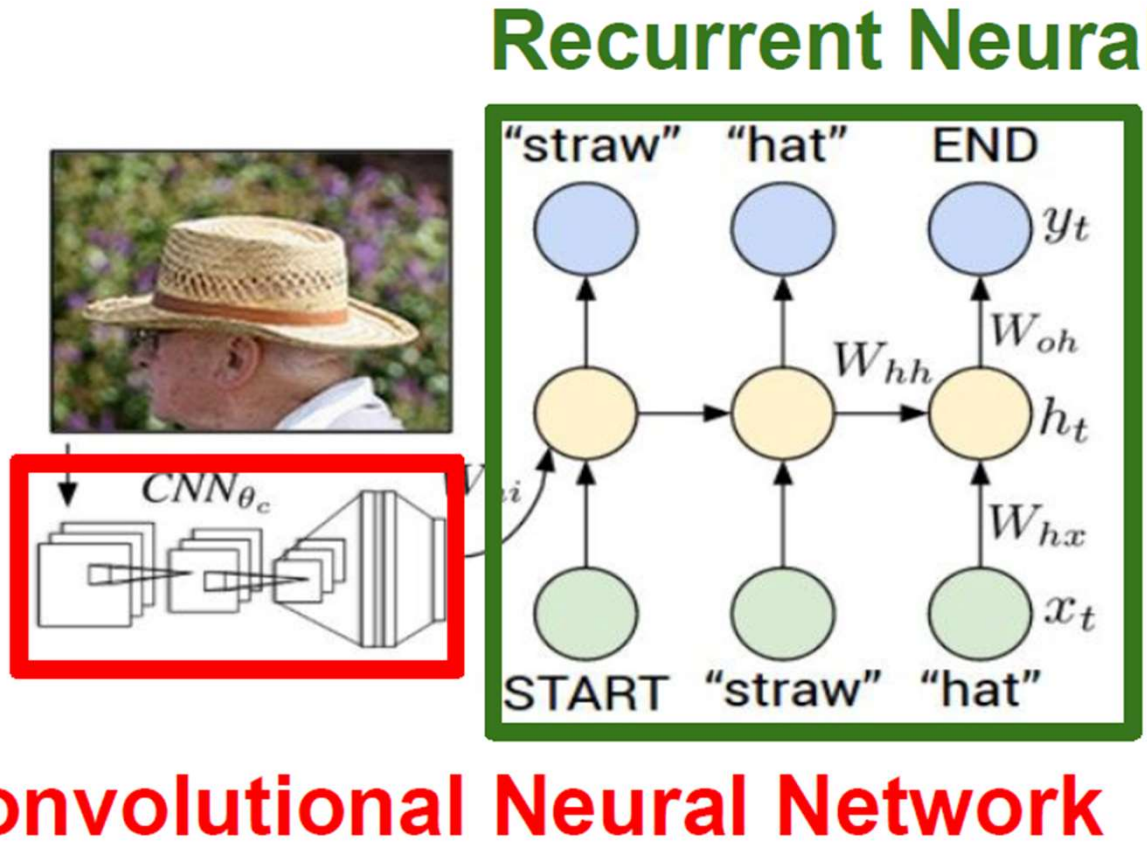


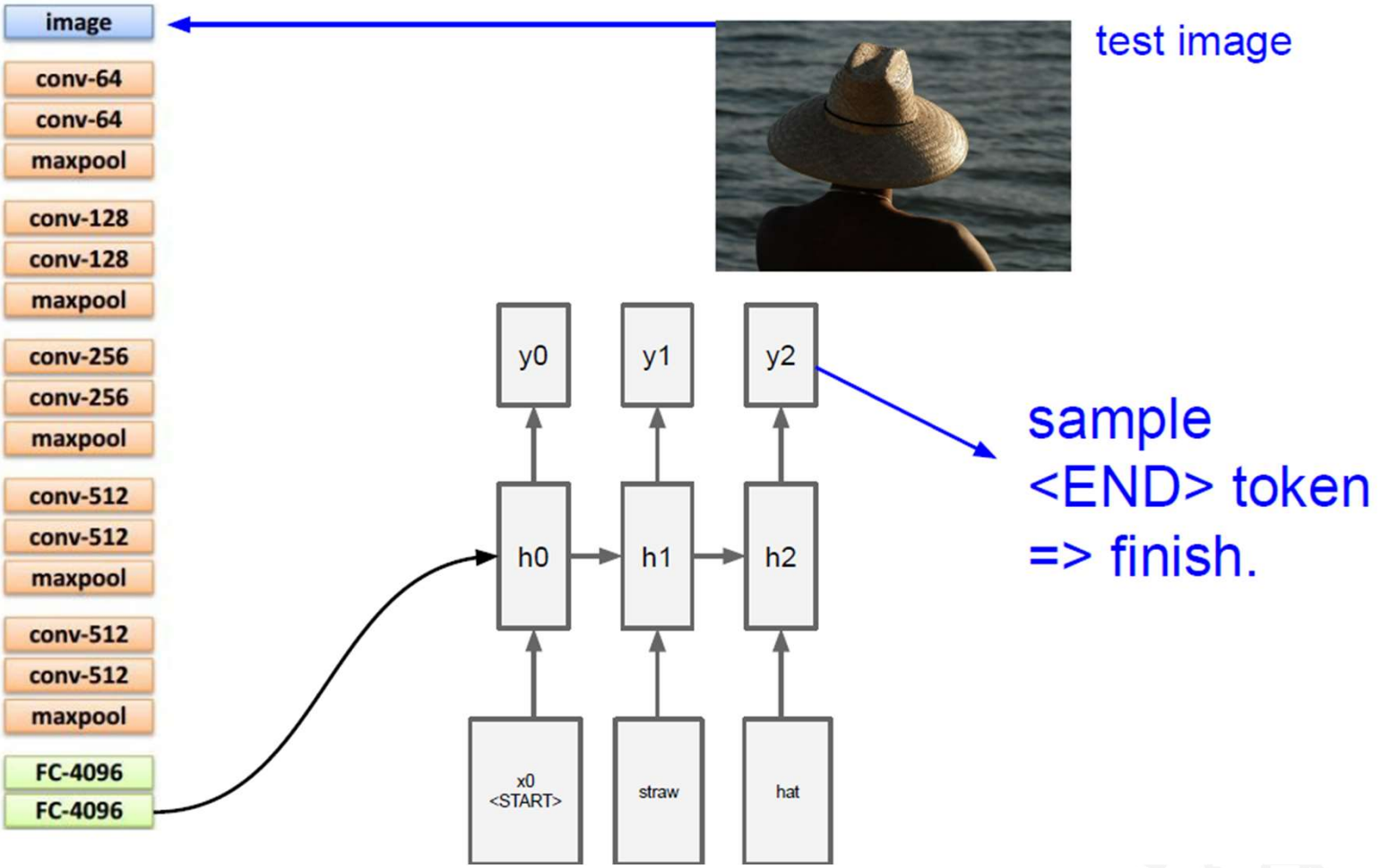
Figure from Karpathy et a, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015; figure copyright IEEE, 2015.
Reproduced for educational purposes.

Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei
Show and Tell: A Neural Image Caption Generator, Vinyals et al.
Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.
Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

Example: Image Captioning



Example: Image Captioning



Example: Image Captioning

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

A tennis player in action on the court

Two giraffes standing in a grassy field

A man riding a dirt bike on a dirt track

Example: Visual Question Answering(VQA)

Q: What endangered animal is featured on the truck?

A: **A bald eagle.**
A: A sparrow.
A: A humming bird.
A: A raven.

Q: Where will the driver go if turning right?

A: **Onto 24 ¾ Rd.**
A: Onto 25 ¾ Rd.
A: Onto 23 ¾ Rd.
A: Onto Main Street.

Q: When was the picture taken?

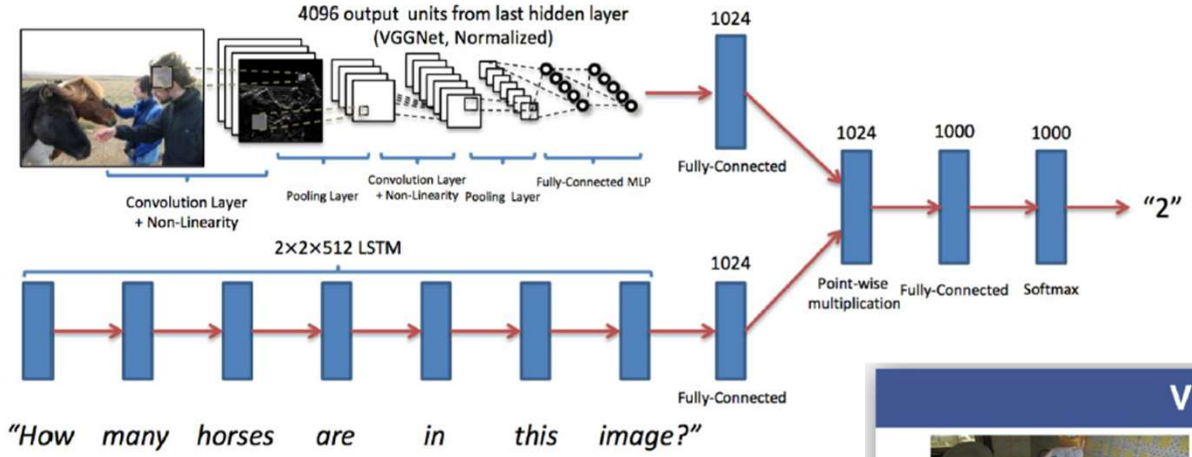
A: **During a wedding.**
A: During a bar mitzvah.
A: During a funeral.
A: During a Sunday church service

Q: Who is under the umbrella?

A: **Two women.**
A: A child.
A: An old man.
A: A husband and a wife.

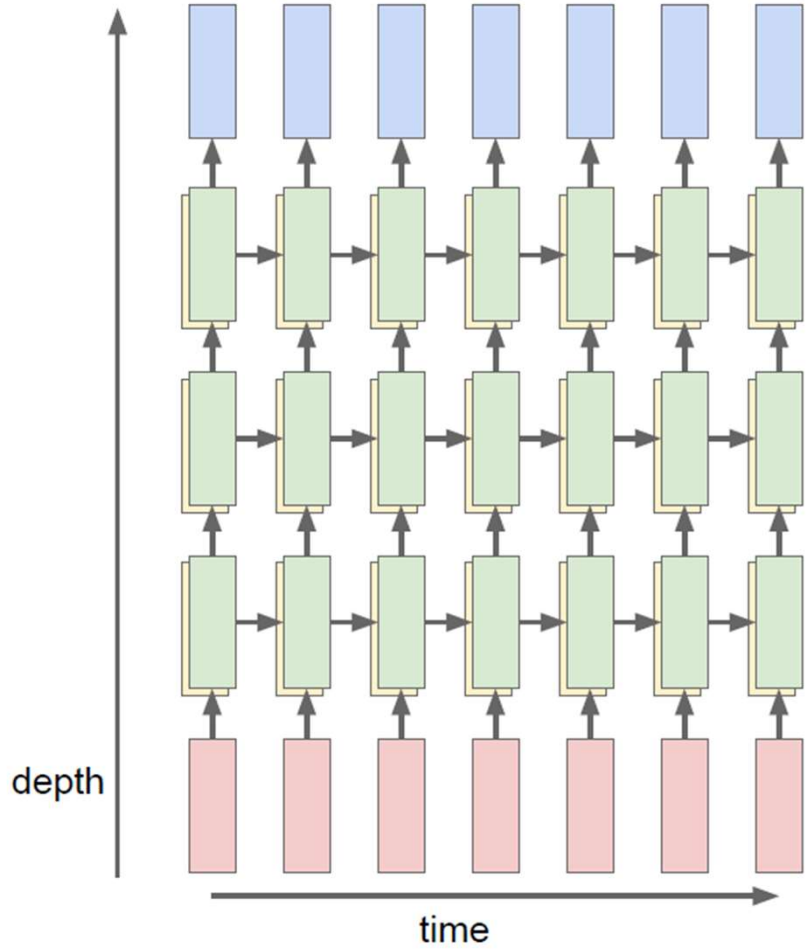
Agrawal et al, "VQA: Visual Question Answering", ICCV 2015
Zhu et al, "Visual 7W: Grounded Question Answering in Images", CVPR 2016

Example: Visual Question Answering(VQA)



Agrawal et al, "Visual 7W: Grounded Question Answering in Images", CVPR 2015
Figures from Agrawal et al, copyright IEEE 2015. Reproduced for educational purposes.

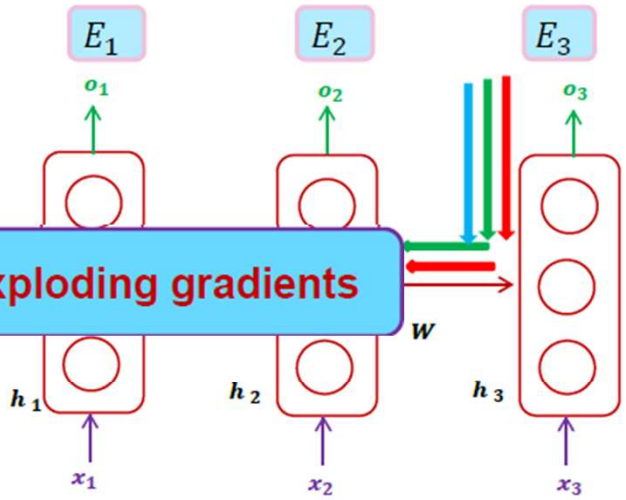
Multilayer RNNs



Vanishing vs Exploding Gradient

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W} + \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W} + \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial h_3}\frac{\partial h_3}{\partial W}$$

$$= \ll\ll 1 \quad + \quad \ll 1 \quad + \quad < 1$$



Repeated matrix multiplications leads to vanishing and exploding gradients

Total Gradient = **Gradient due to long term dependencies** + **Gradient due to short term dependencies**

Remark: *The gradients due to short term dependencies (just previous dependencies) dominates the gradients due to long-term dependencies.*

This means network will tend to focus on short term dependencies which is often not desired

Problem of Vanishing Gradient

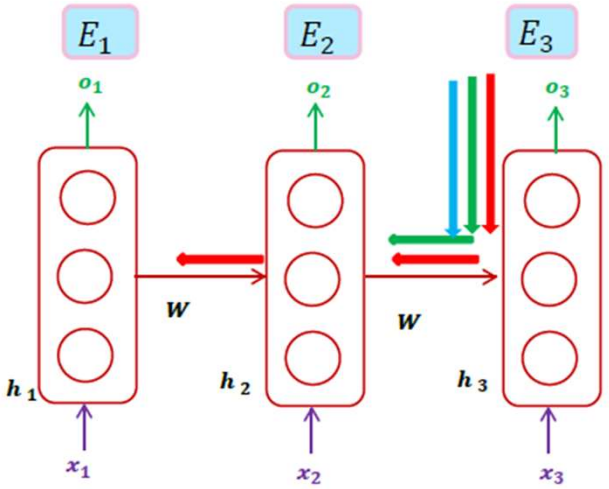
Vanishing vs Exploding Gradient

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W} + \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W} + \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial h_3}\frac{\partial h_3}{\partial W}$$

$$= \gg\gg 1 \quad + \quad \ggg 1 \quad + \quad \ggg 1$$

= Very large number, i.e., NaN

Problem of Exploding Gradient



Vanishing vs Exploding Gradient

$$\left\|\frac{\partial h_3}{\partial h_k}\right\| \leq (\gamma_w \gamma_g)^{t-k}$$

For tanh or linear activation

$\gamma_w \gamma_g$ **greater than** 1

Gradient *Expodes* !!!

$\gamma_w \gamma_g$ **less than** 1

Gradient *Vanishes* !!!

Remark: This problem of exploding/vanishing gradient occurs because the same number is multiplied in the gradient repeatedly.

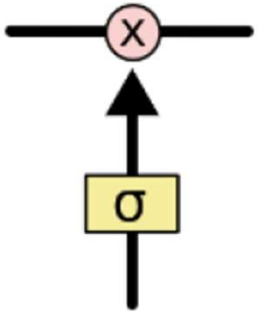
Long Short Term Memory (LSTM) : Gating

Gates :

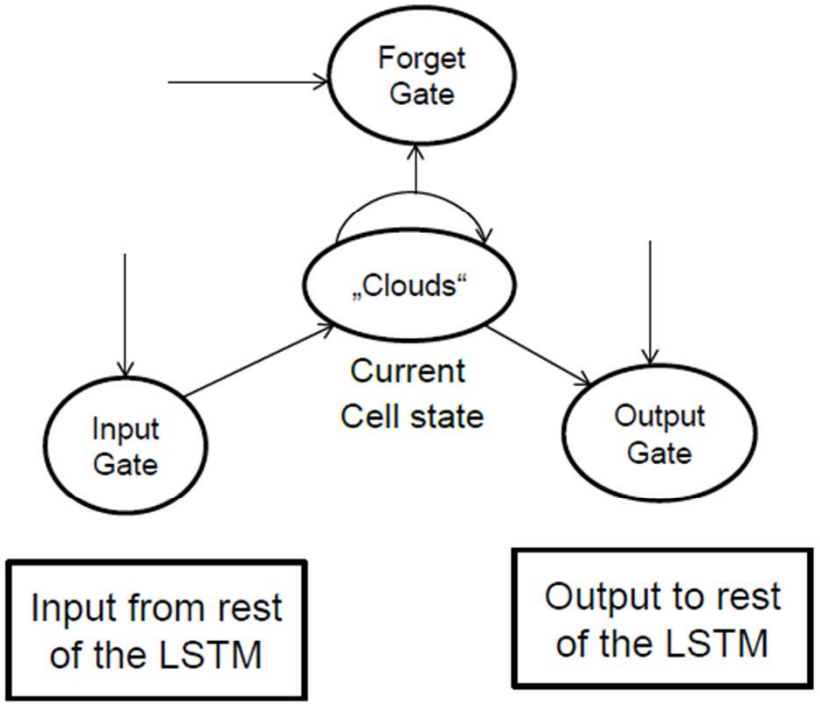
→ way to optionally let information through.

→ 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

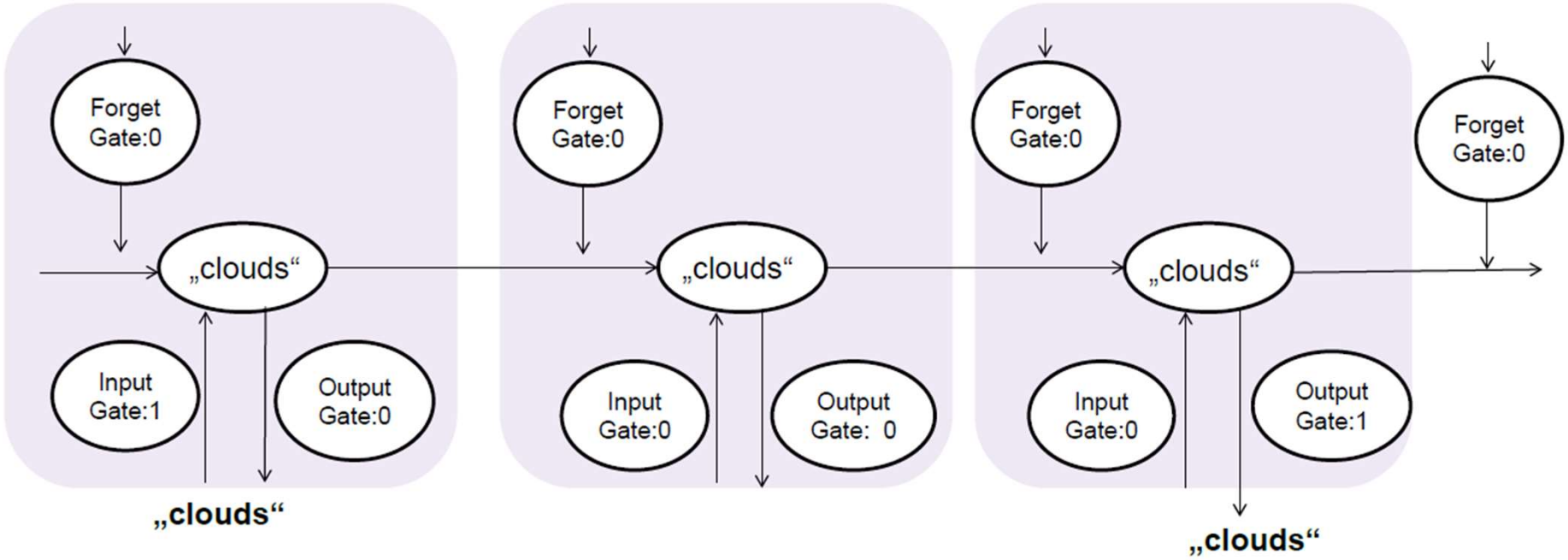


→ gates to protect and control the cell state.

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

Long Short Term Memory (LSTM) : Gating

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

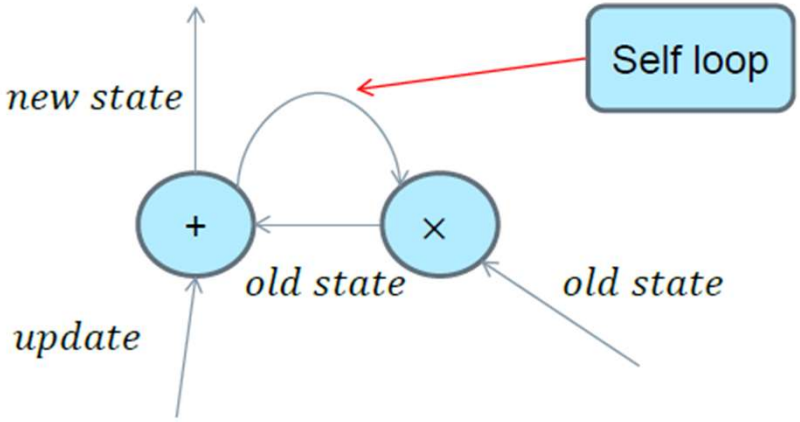


Lecture from the course Neural Networks for Machine Learning by Greff Hinton

Long Short Term Memory (LSTM) : Gating

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**



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

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

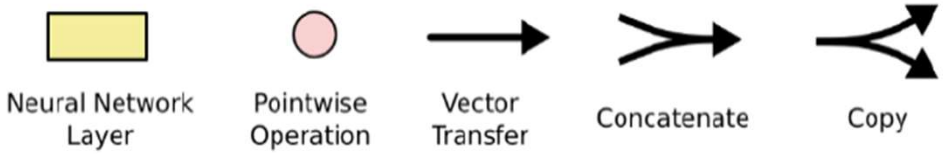
LONG SHORT-TERM MEMORY, Sepp Hochreiter and Jürgen Schmidhuber

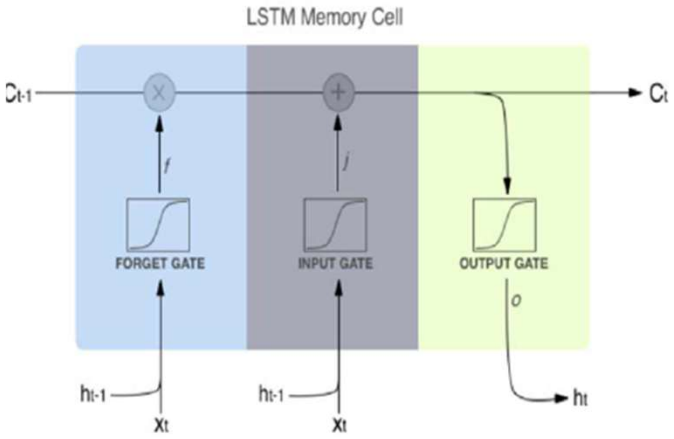
Long Short Term Memory (LSTM) : Step by Step

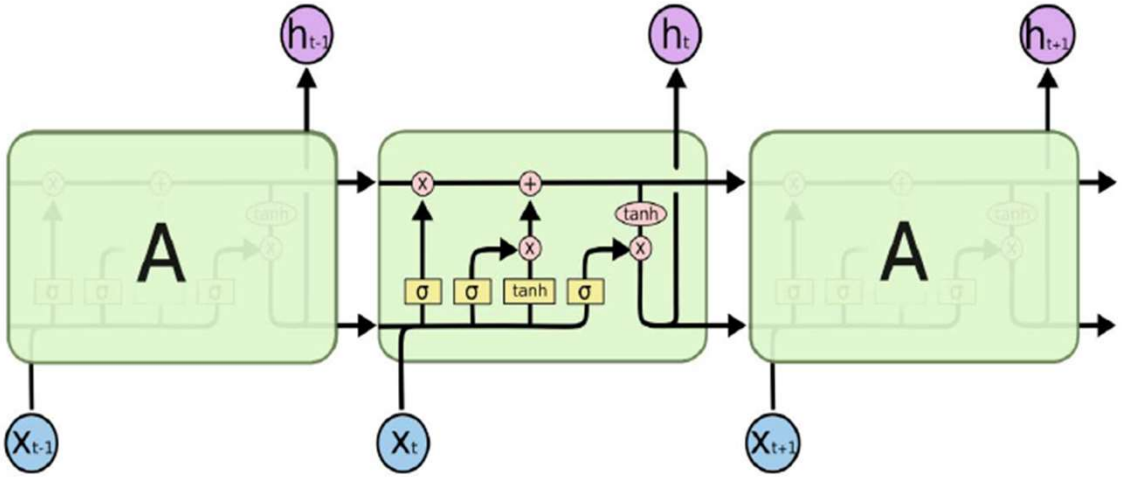
Key Ingredients

Cell state - transport the information through the units

Gates – optionally allow information passage







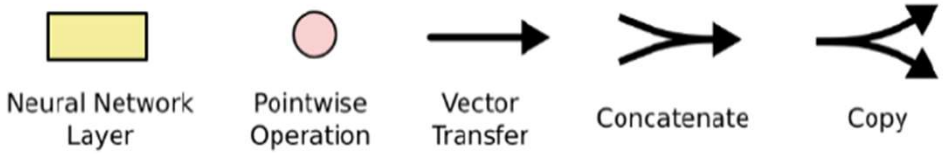
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

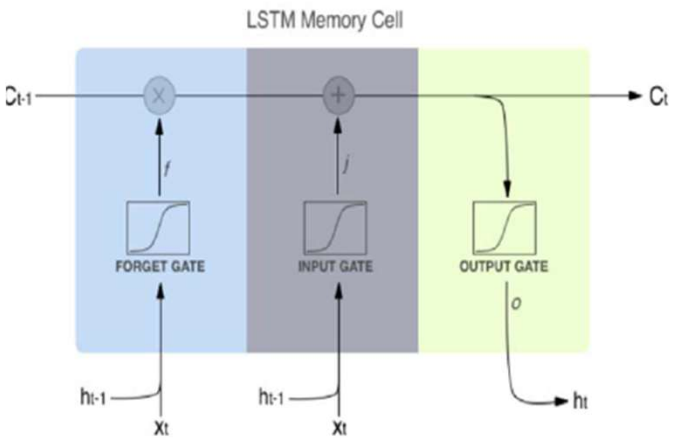
Long Short Term Memory (LSTM) : Step by Step

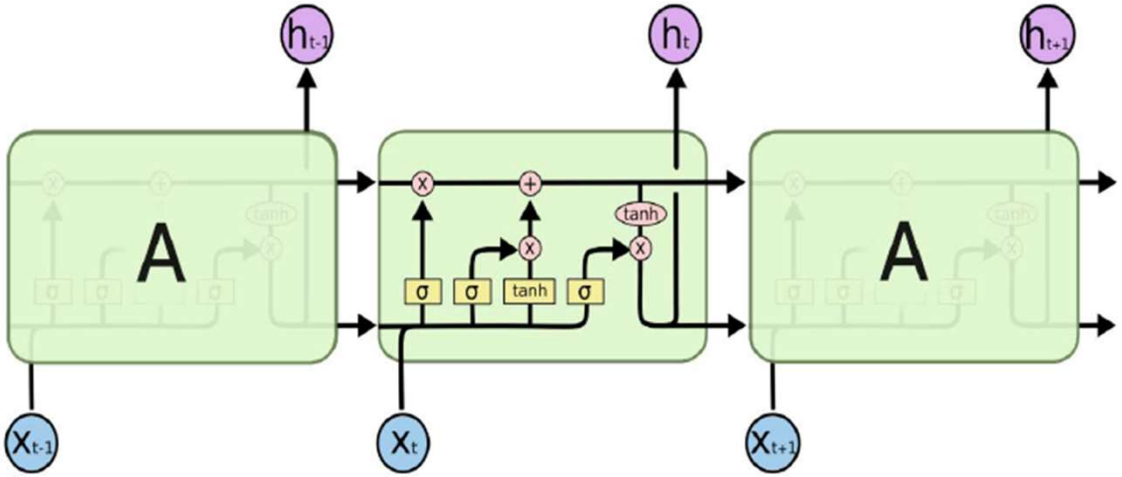
Key Ingredients

Cell state - transport the information through the units

Gates – optionally allow information passage







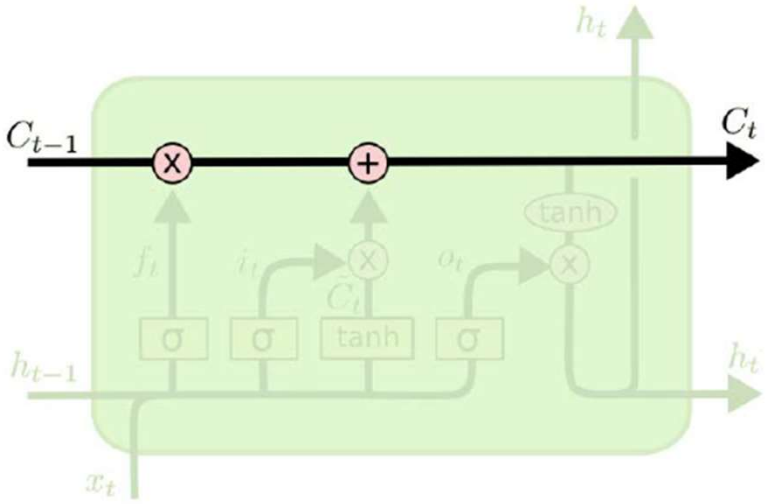
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

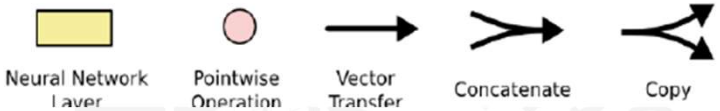
Long Short Term Memory (LSTM) : Step by Step

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

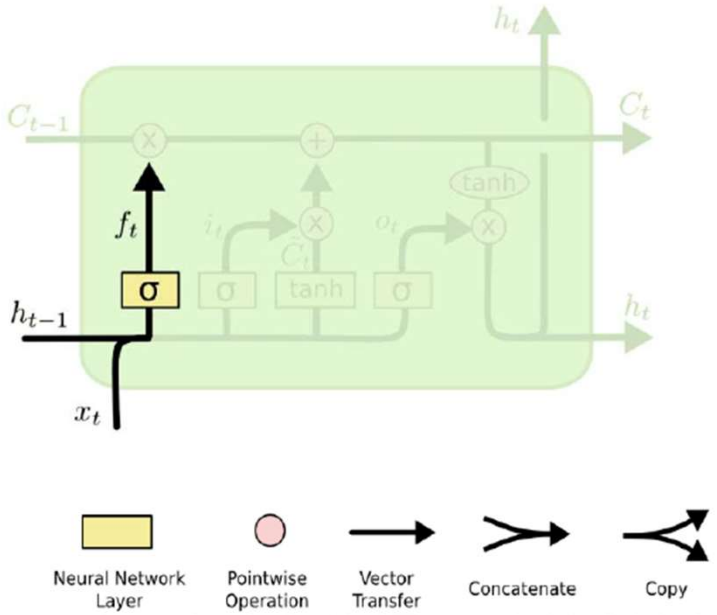
→ decision maker: sigmoid layer (*forget gate layer*)

The output of the sigmoid lies between 0 to 1,

→ 0 being forget, 1 being keep.

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

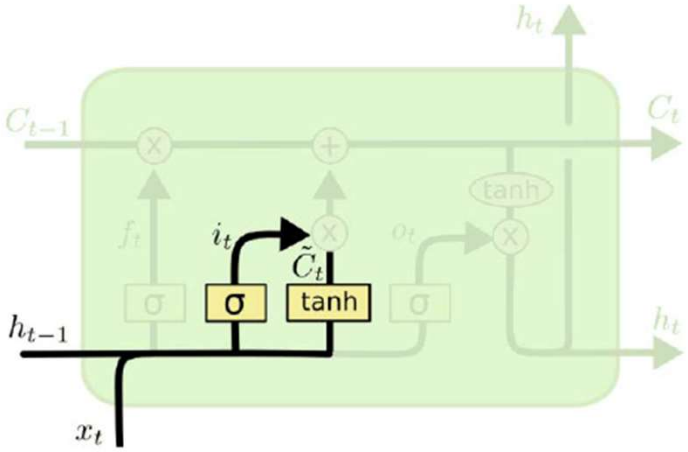
→ looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}



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



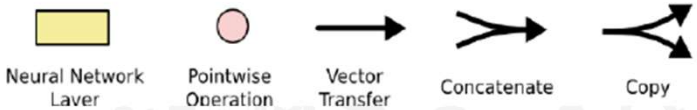
$$i_t = sigmoid(\theta_{xi}x_t + \theta_{hi}h_{t-1} + b_i)$$
$$\tilde{C}_t = Tanh(\theta_{xg}x_t + \theta_{hg}h_{t-1} + b_g)$$

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**.

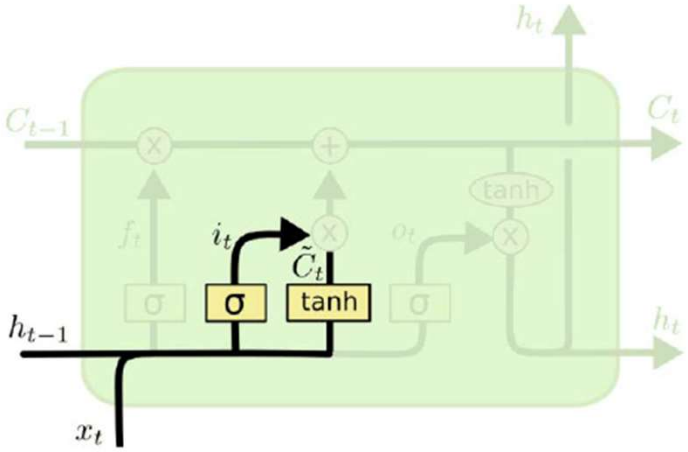
In the next step, we'll combine these two to **create an update** to the state.



Long Short Term Memory (LSTM) : Step by Step

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



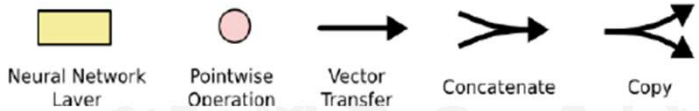
$$i_t = sigmoid(\theta_{xi}x_t + \theta_{hi}h_{t-1} + b_i)$$
$$\tilde{C}_t = Tanh(\theta_{xg}x_t + \theta_{hg}h_{t-1} + b_g)$$

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**.

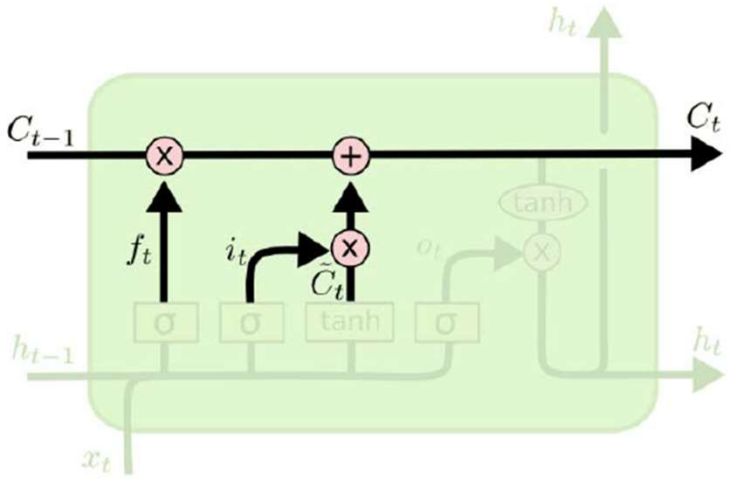
2. A tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that **could be added to the state**.

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



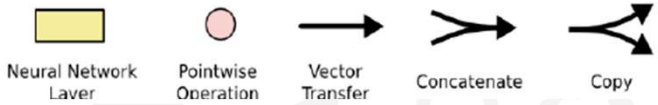
Long Short Term Memory (LSTM) : Step by Step

Cell Update



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

- 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.



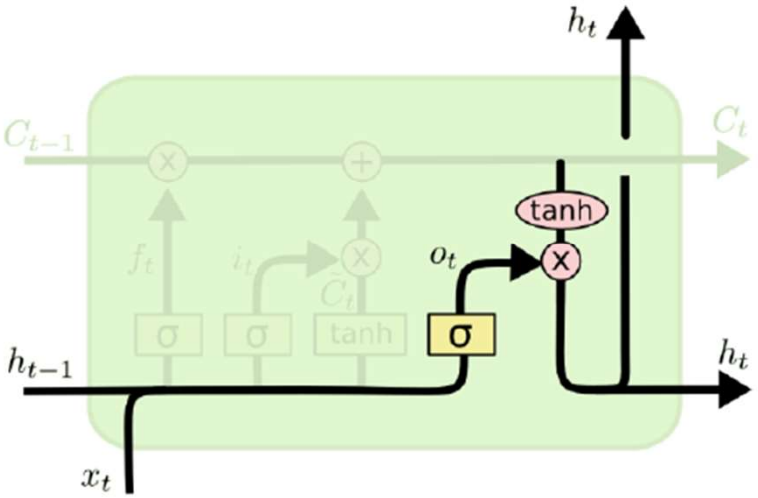
Long Short Term Memory (LSTM) : Step by Step

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



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

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

Long Short Term Memory (LSTM) : Step by Step

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

Gated Recurrent Unit(GRU)

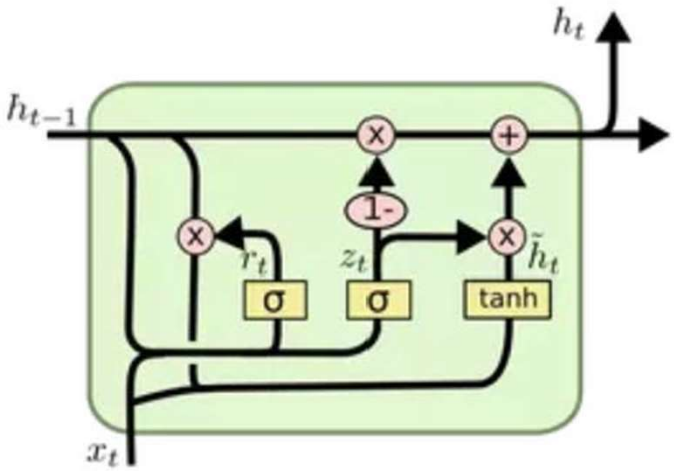
- GRU like LSTMs, attempts to solve the Vanishing gradient problem in RNN

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

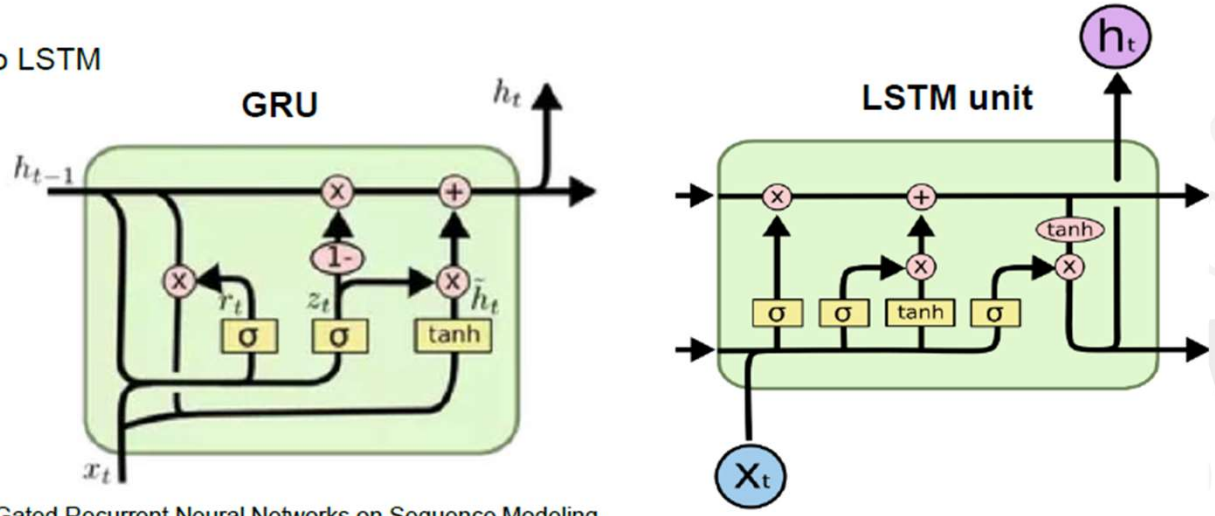
Gated Recurrent Unit(GRU)

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

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

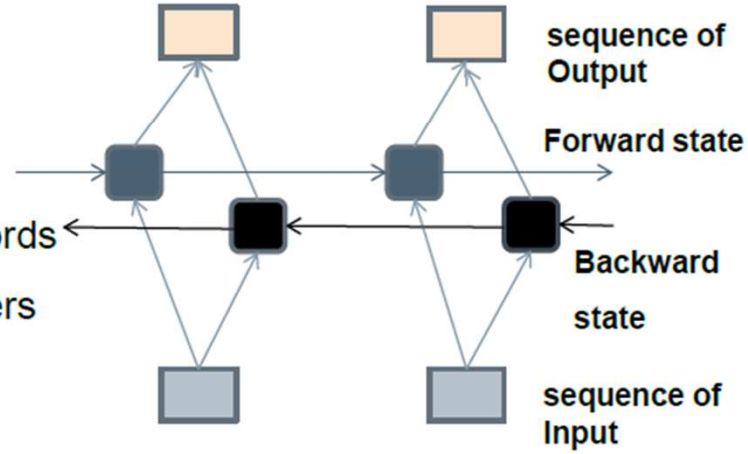
Example sentences:

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

Bi-directional RNNs

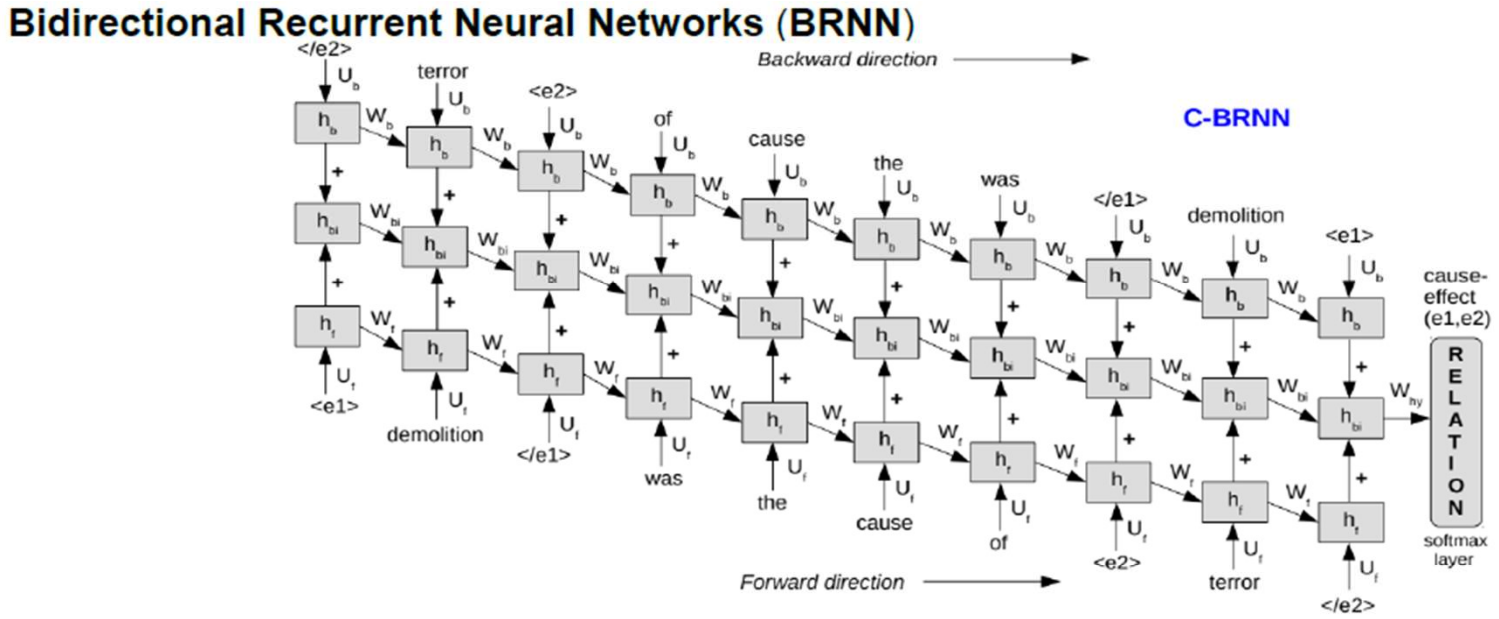


Figure 1: Connectionist Bi-directional Recurrent Neural Network (C-BRNN) (Vu et al., 2016a)

Gupta 2015. (Master Thesis). *Deep Learning Methods for the Extraction of Relations in Natural Language Text*

Gupta and Schütze. 2018. *LISA: Explaining Recurrent Neural Network Judgments via Layer-wIse Semantic Accumulation and Example to Pattern Transformation*

Vu et al., 2016. *Combining recurrent and convolutional neural networks for relation classification*

Thank You!