# Why LSTM outperforms RNN?

**A TA lecture for EEE415**

**Xintao Huan**

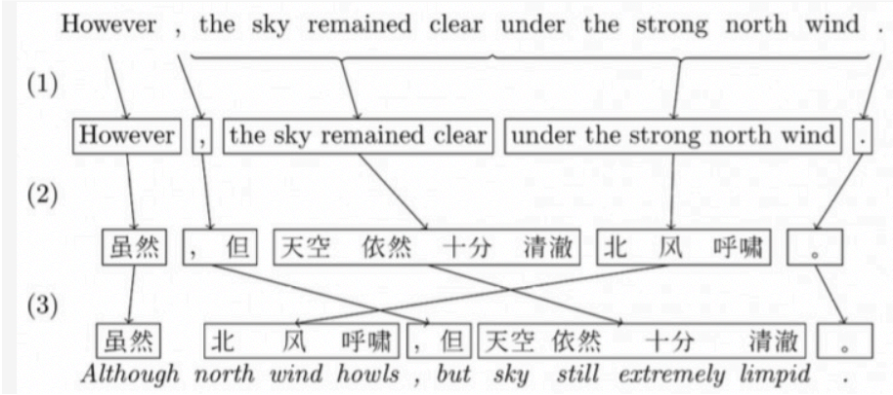
XJTLU | ELECTRICAL & ELECTRONIC ENGINEERING

Xi'an Jiaotong-Liverpool University 西交利物浦大學

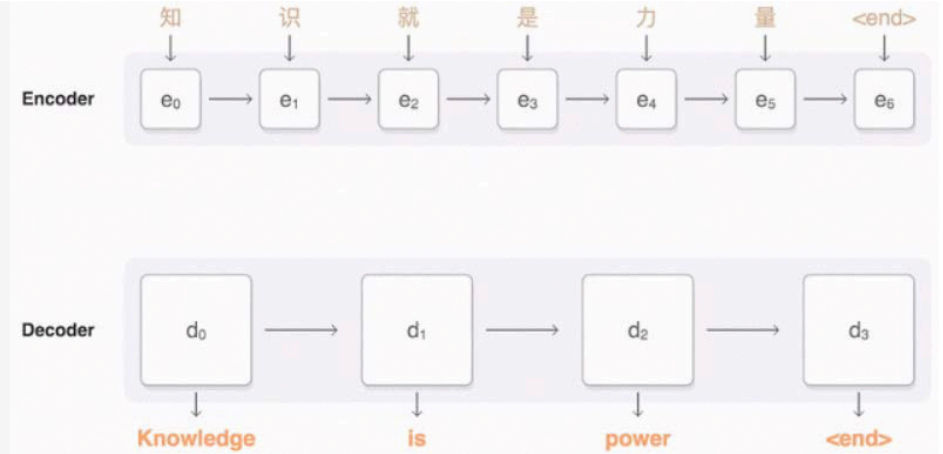
# Outline

- Review of RNN
- Achilles' Heel of RNN
- Solution - LSTM
- Summary

# Breakthrough brought by RNN – machine translation

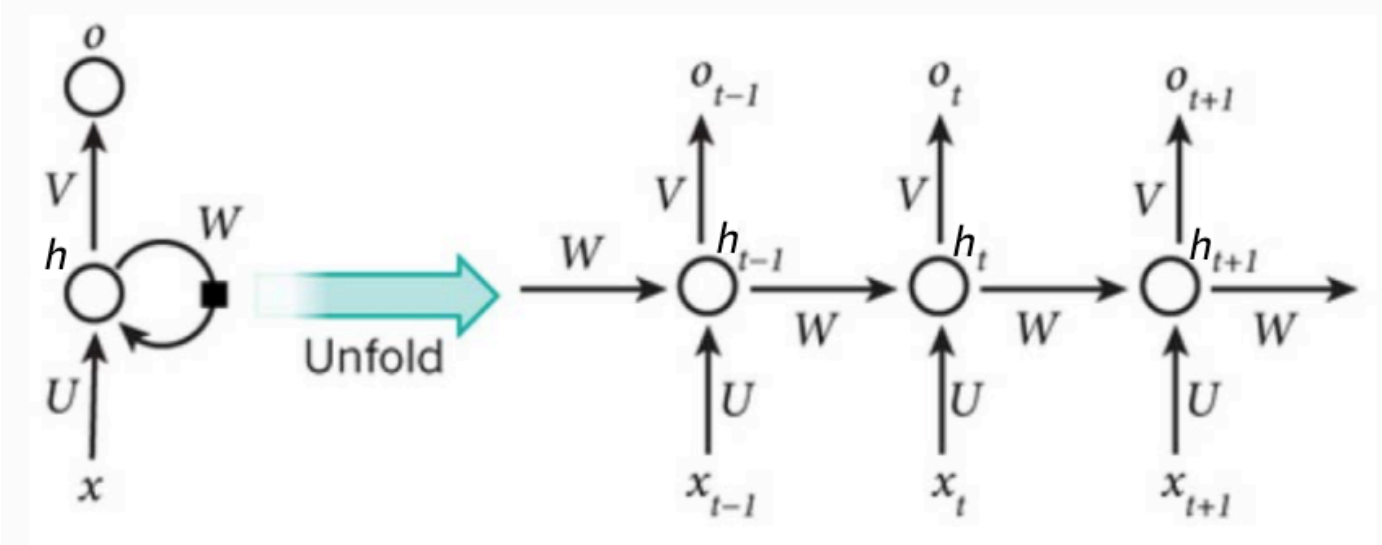



Conventional Method




Recurrent Neural Networks

# RNN – Network Structure




Recurrent Neural Network (RNN) and the unfolding in time of the computation involved in its forward computation.

# RNN – Network Structure

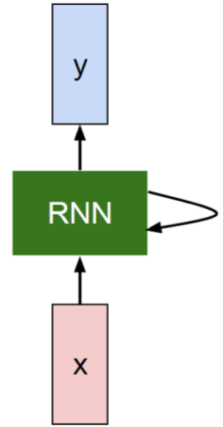
$$h_t = f_W(h_{t-1}, x_t)$$

new state

function with parameters W

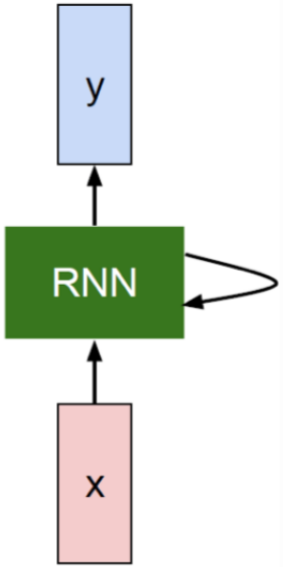
old state

input vector at time step t




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

# RNN – Network Structure




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

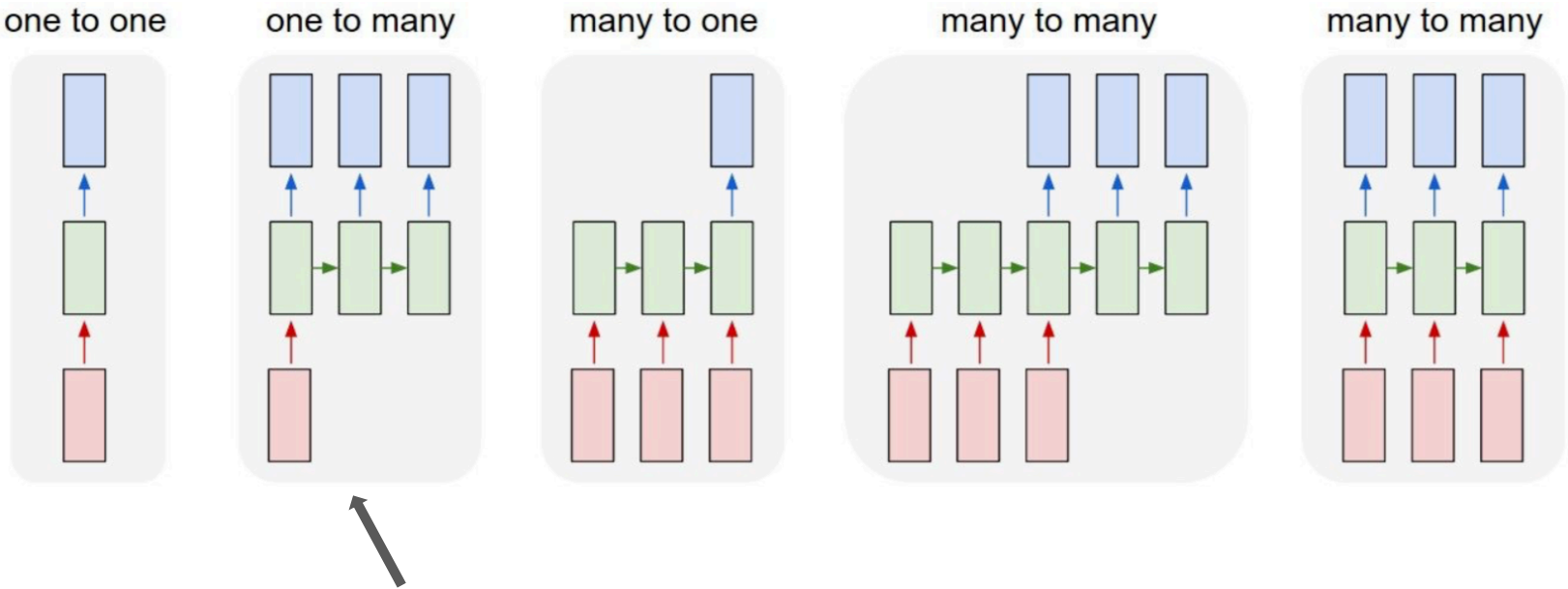
$$\downarrow$$

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

$$y_t = W_{hy}h_t$$

W*hh* : weight between hidden layers
W*xh*: weight between input layer and hidden layer
W*hy*: weight between hidden layer and output layer

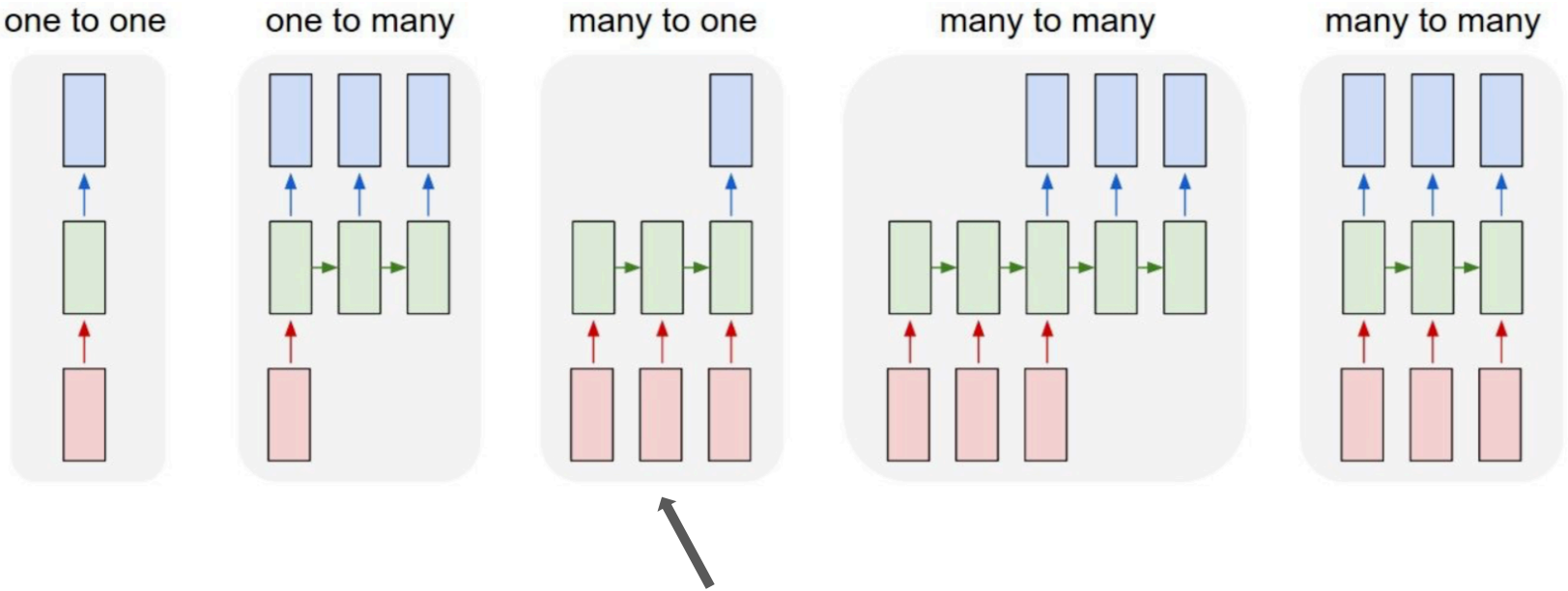
# RNN – Use Cases




e.g., image caption

image-> sequence of words

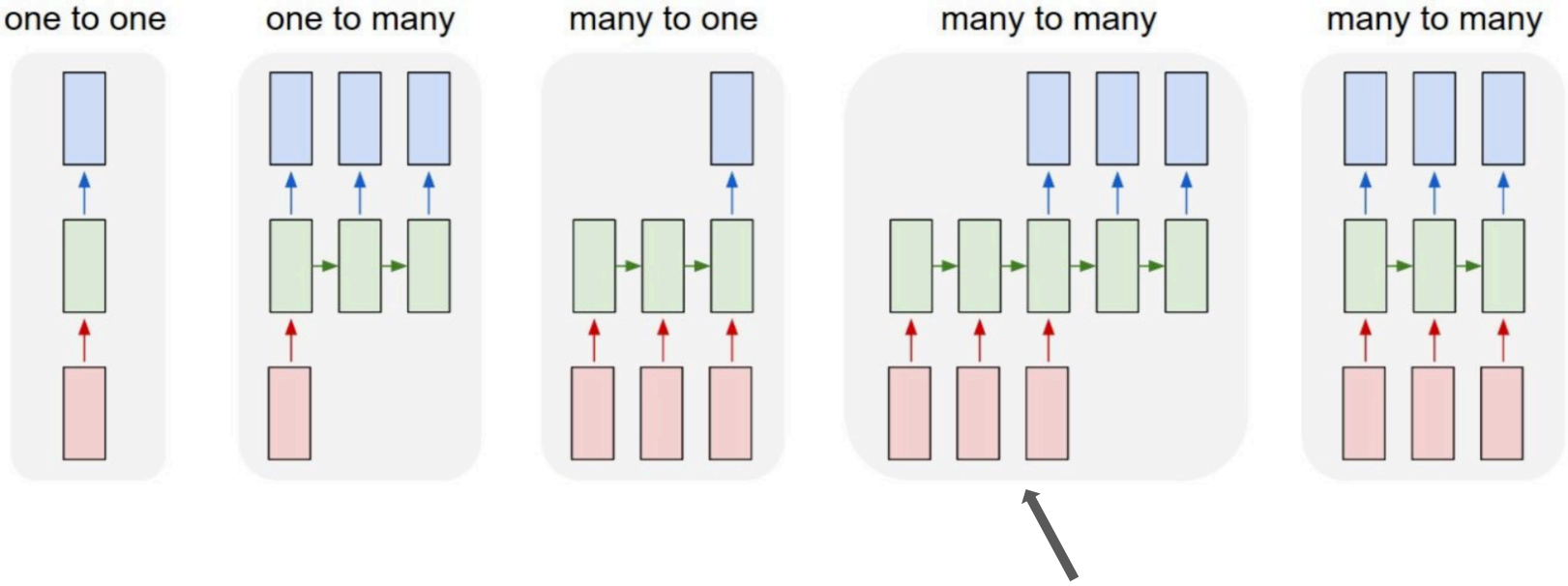
# RNN – Use Cases




e.g., sentiment classification

sequence of words -> sentiment

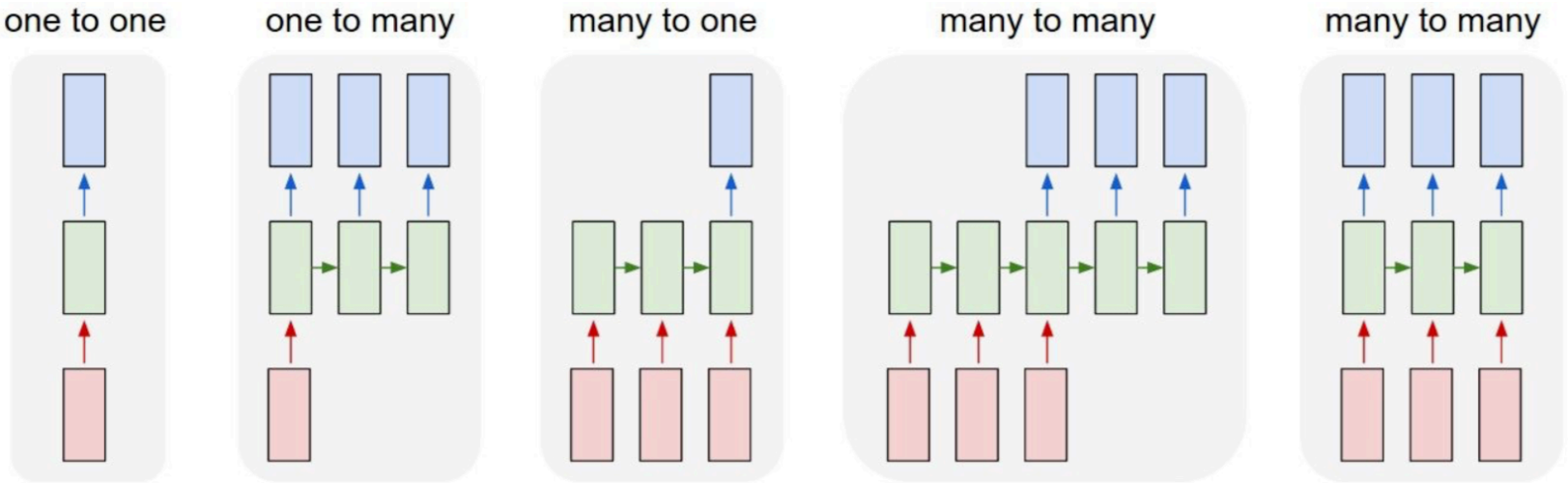
# RNN – Use Cases




e.g., machine translation

sequence of words -> sequence of words

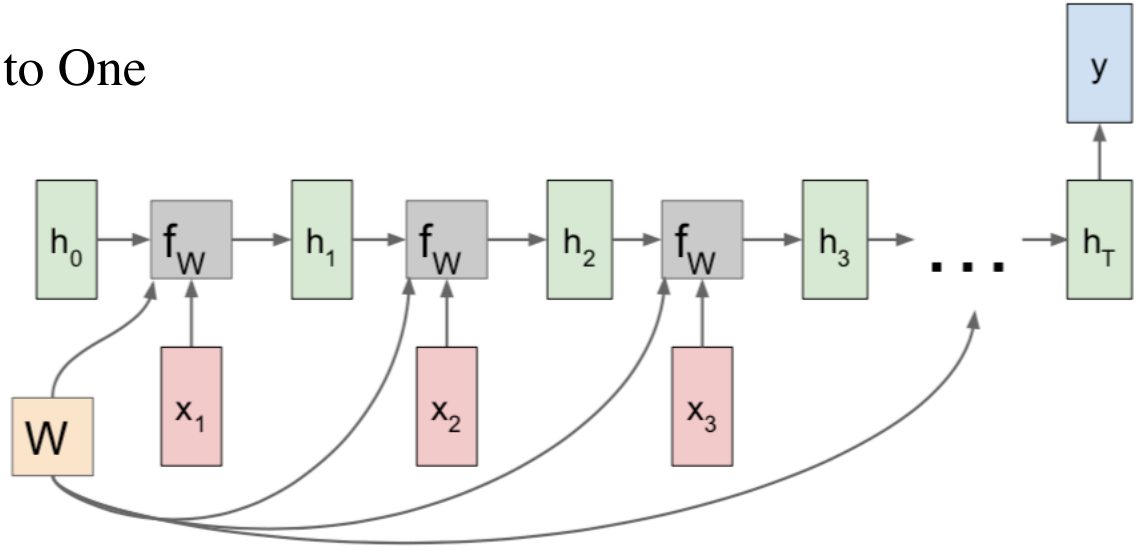
# RNN – Use Cases




e.g., video classification on frame level

# RNN – Computation Graph

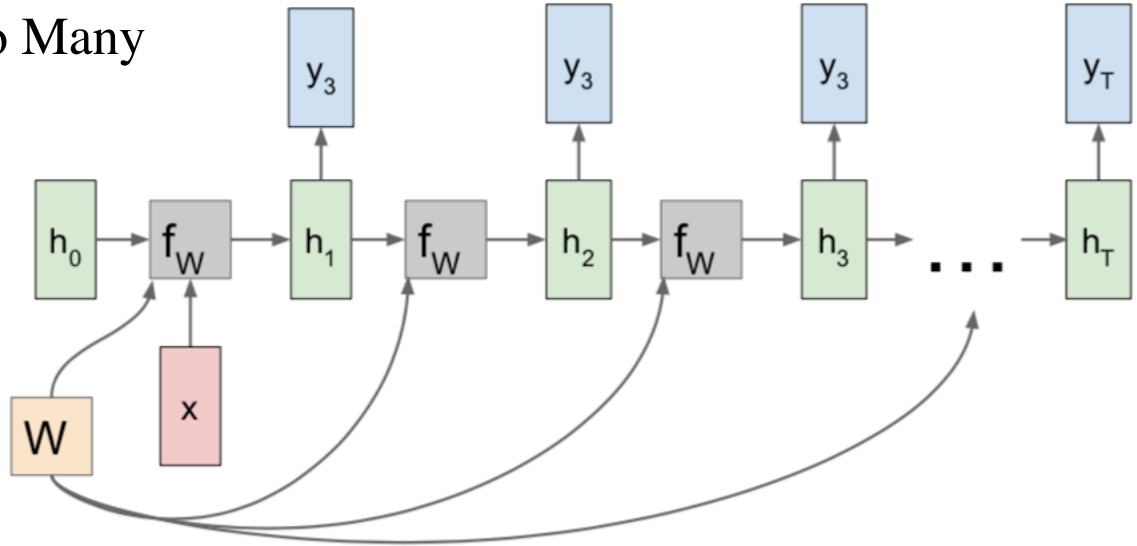
Many to One




Encode input sequence in a single vector

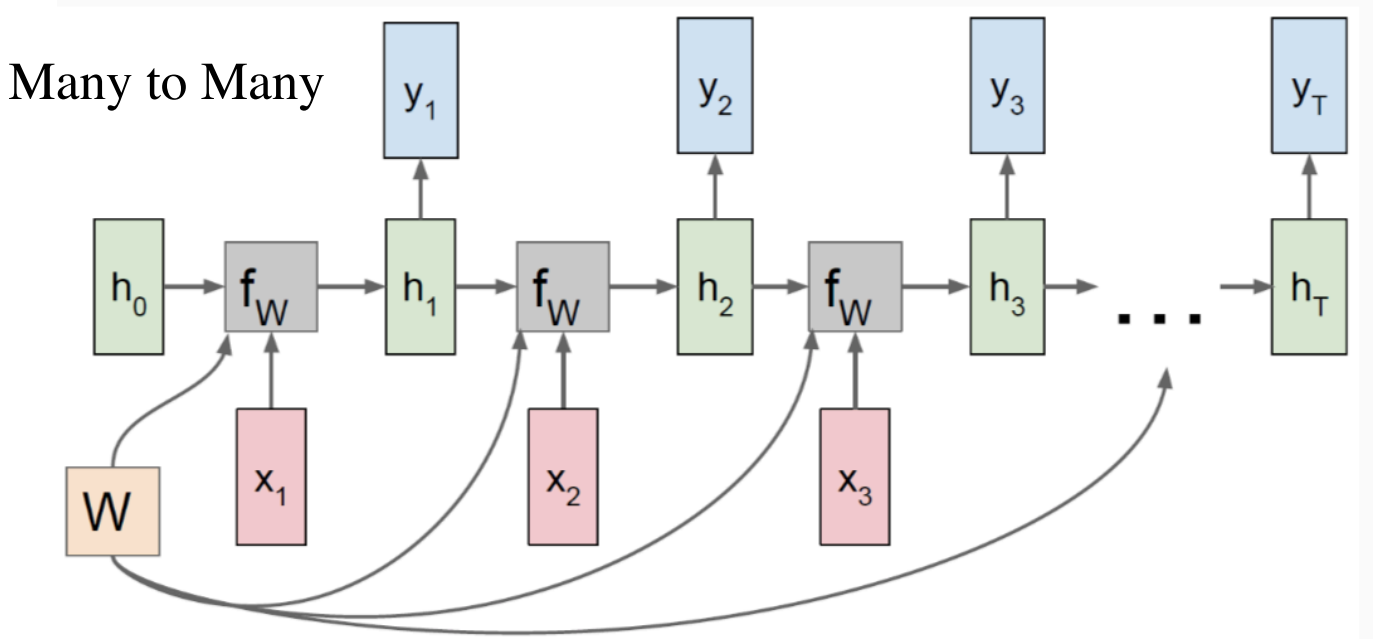
# RNN – Computation Graph

One to Many



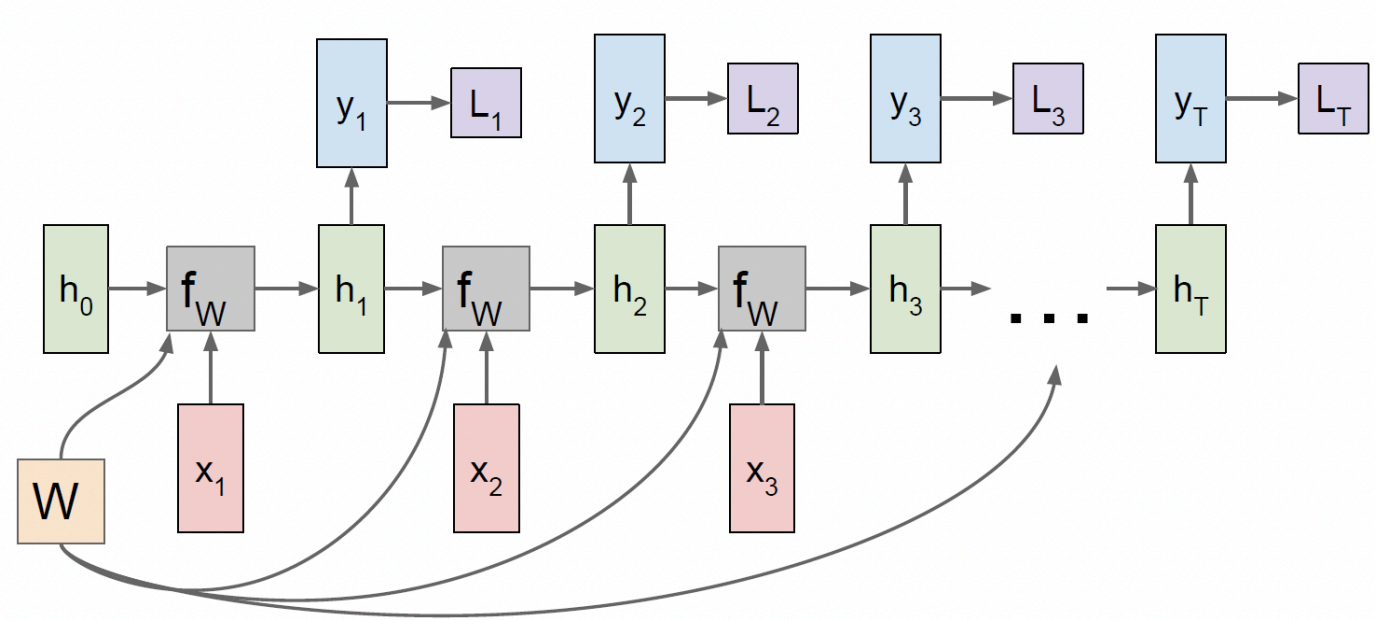

Produce output sequence from single input vector

# RNN – Computation Graph



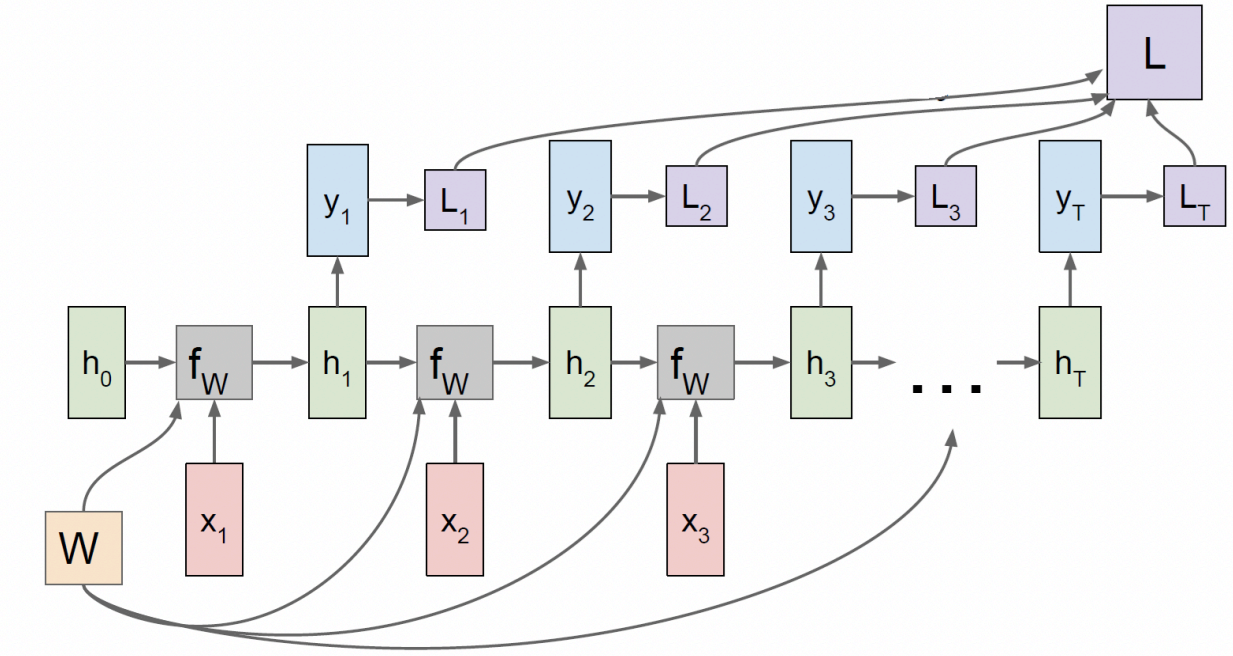

In the computation of the hidden layer, Weight Matrix **W** is **shared & used** in **all time-steps**!

# RNN – Computation Graph



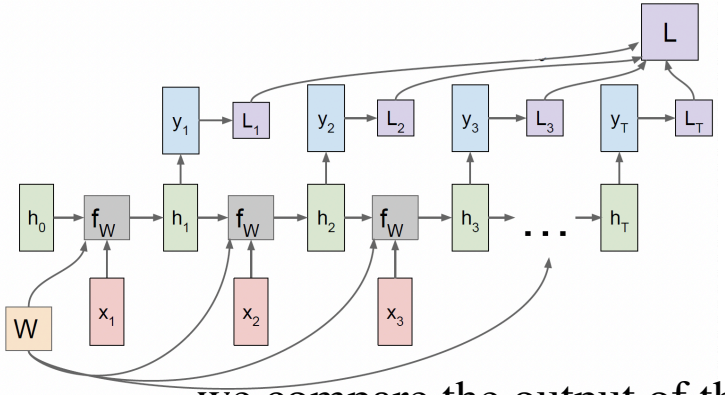

Every time-step computes the Loss

# RNN – Computation Graph




Eventually the loss matrix is computed

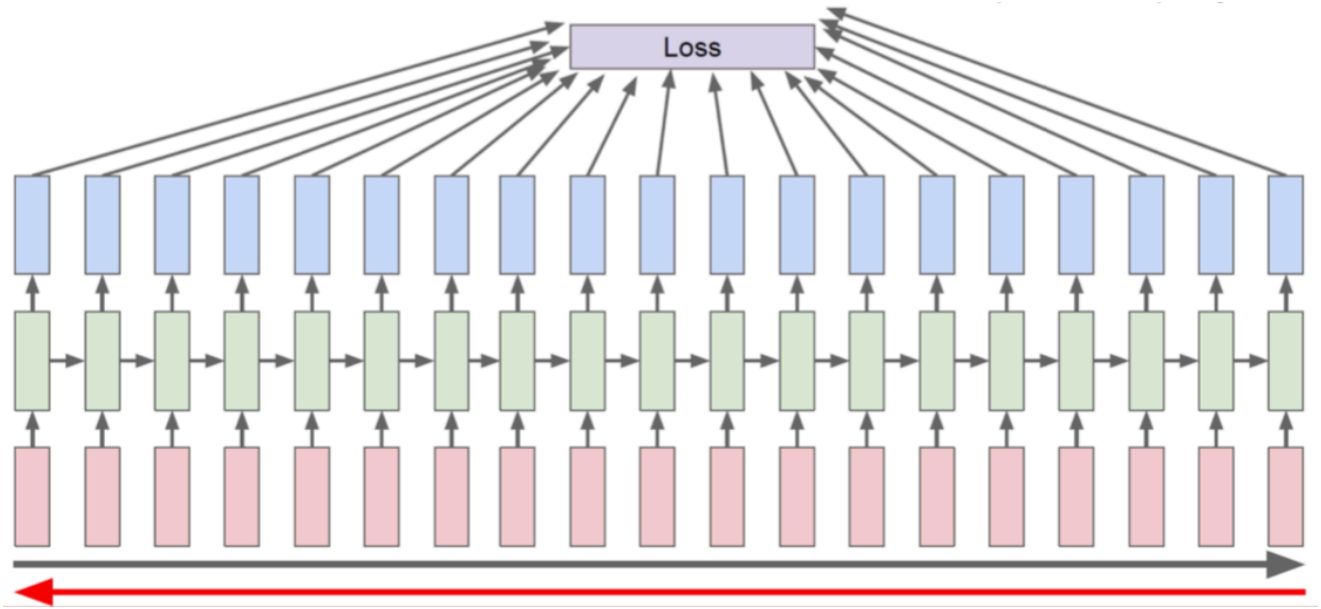
# RNN – Computation Graph




In TRAINING,
we compare the output of the time-step **yt** with the reference result, then the loss **Lt** is obtained,
and sequence loss **L** will be back propagated from the end time-step to the first time-step.
Next,
we employ **Stochastic Gradient Descent (SGD)** to minimize the loss and update the parameters in **W**.

# RNN – Back Propagation

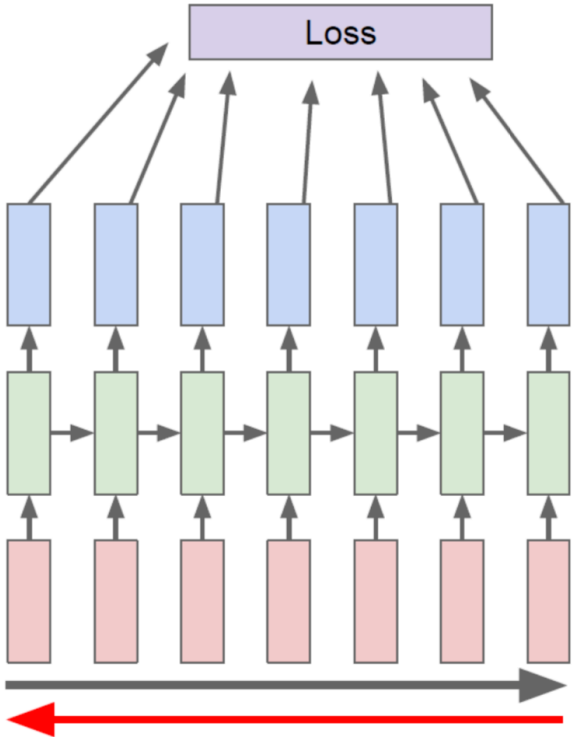
Back Propagation Through Time (BPTT)




Forward through entire sequence to compute the **Loss**.

Backward through entire sequence to **minimize the loss** and **update the parameters** in **W**.

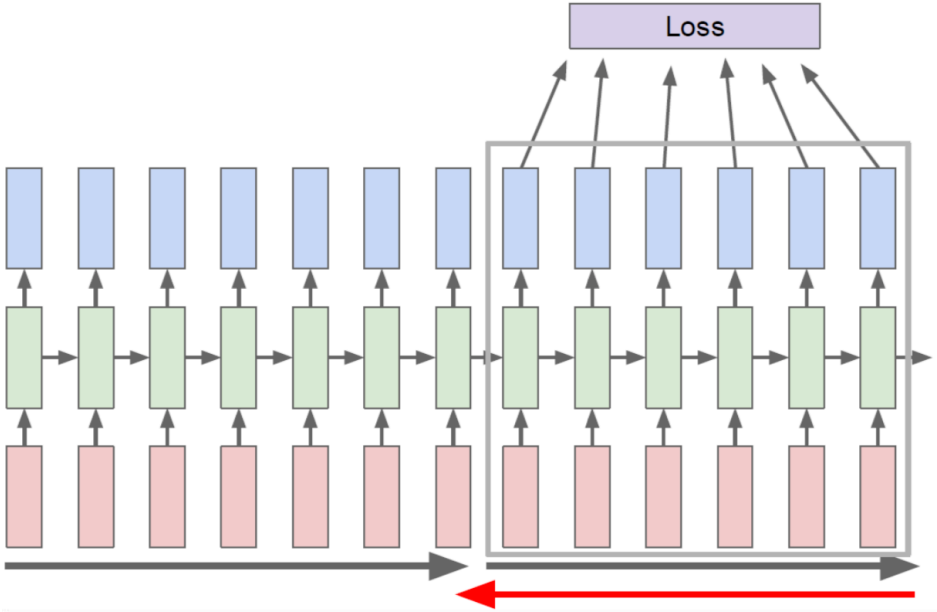
# RNN – Back Propagation




Truncated BPTT

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

# RNN – Back Propagation




Truncated BPTT

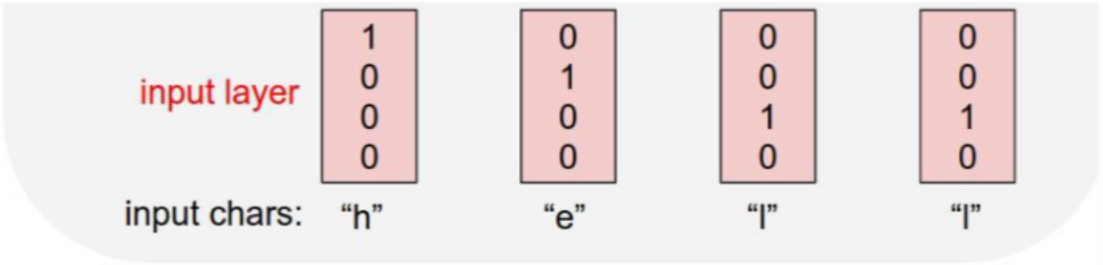
Carry hidden states forward in time forever (black lines), but only backpropagation for some smaller number of steps (red lines)

# RNN – Example

Example:
Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



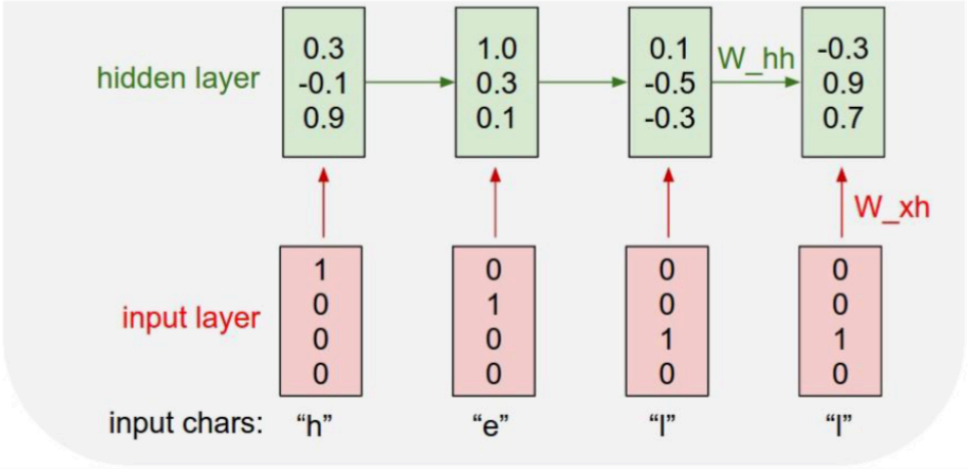
# RNN – Example

Example:
Character-level Language Model

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

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

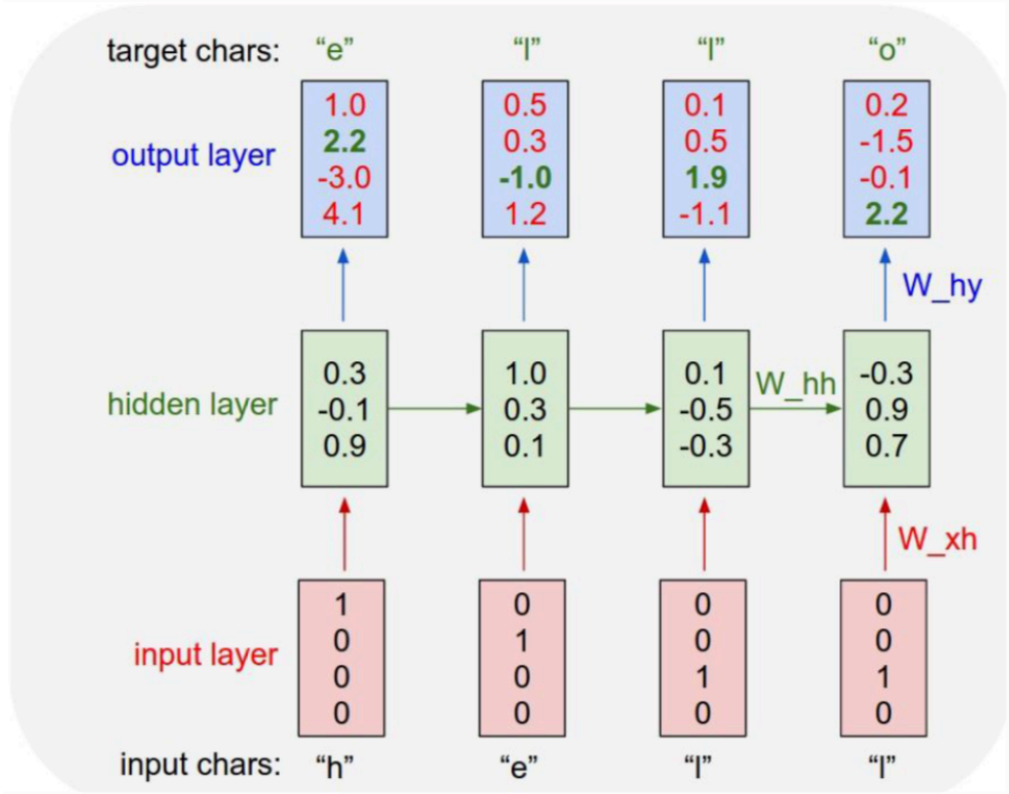


# RNN – Example

Example:
Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



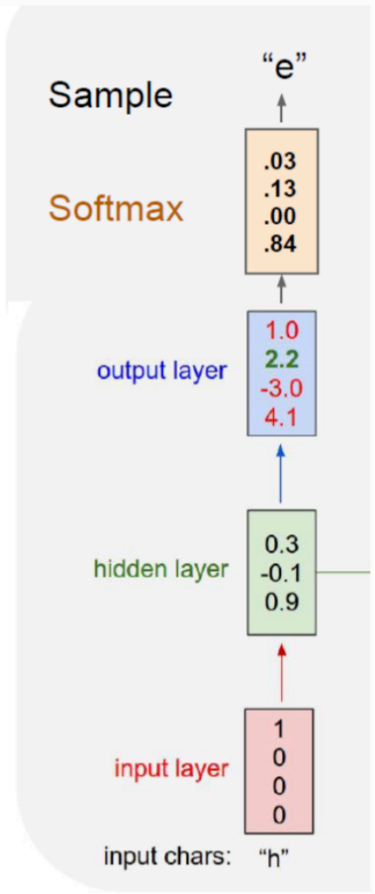
# RNN – Example

Example:
Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

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



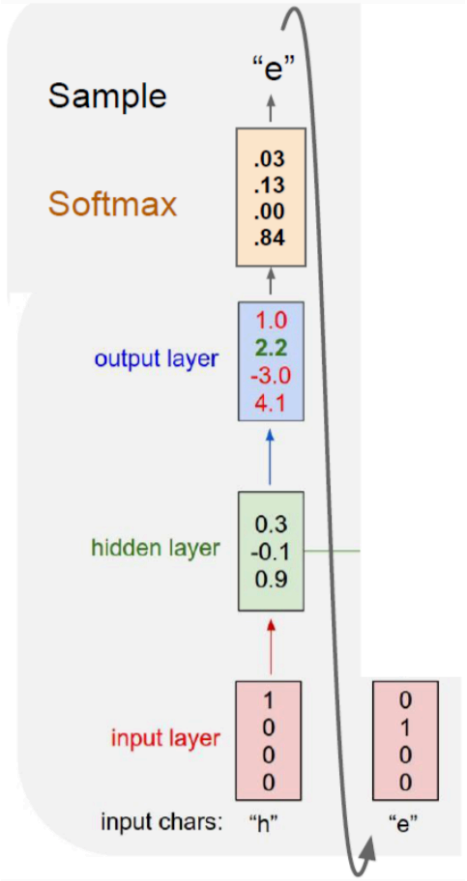
# RNN – Example

Example:
Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

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



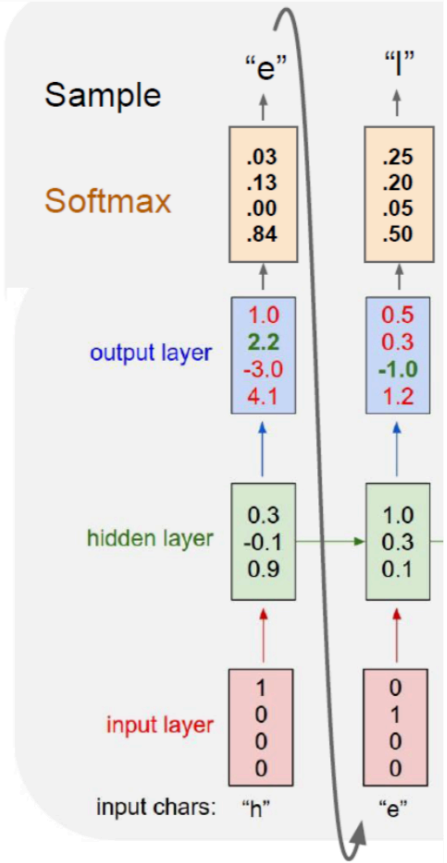
# RNN – Example

Example:
Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

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



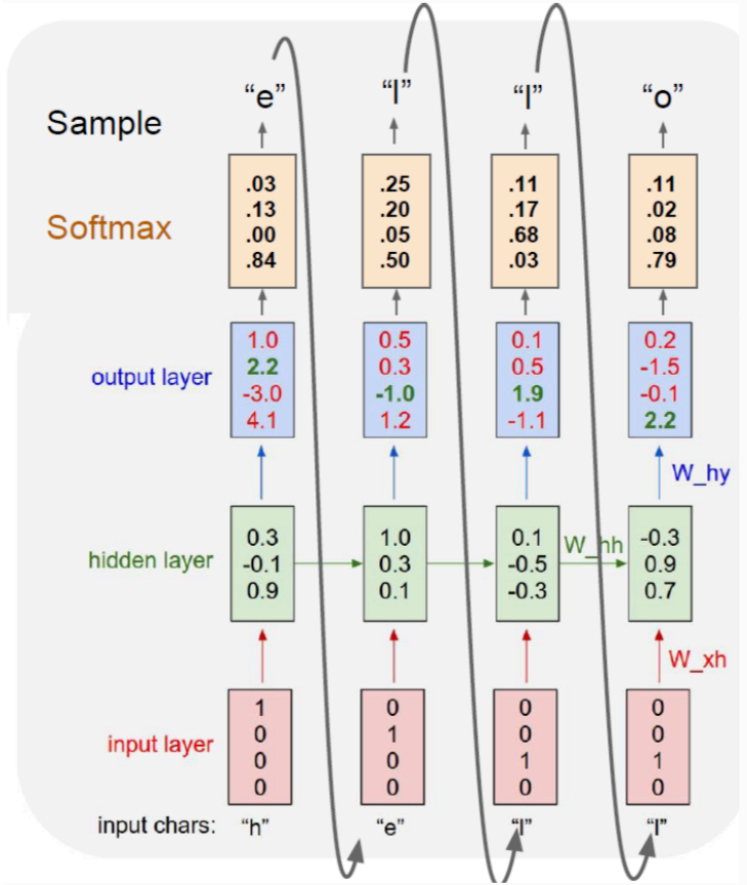
# RNN – Example

Example:
Character-level Language Model

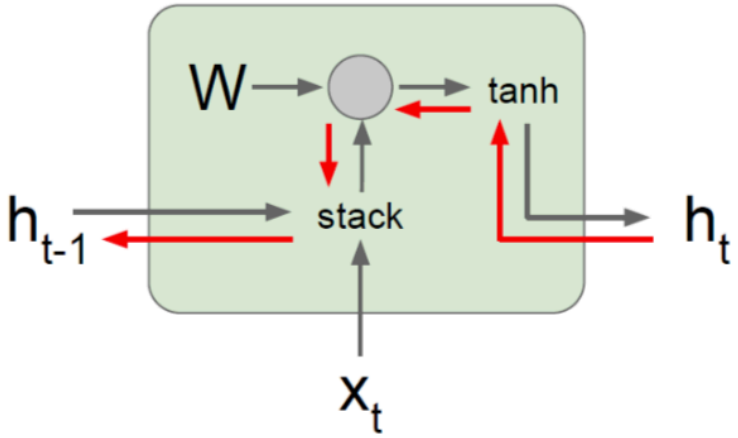
Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

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



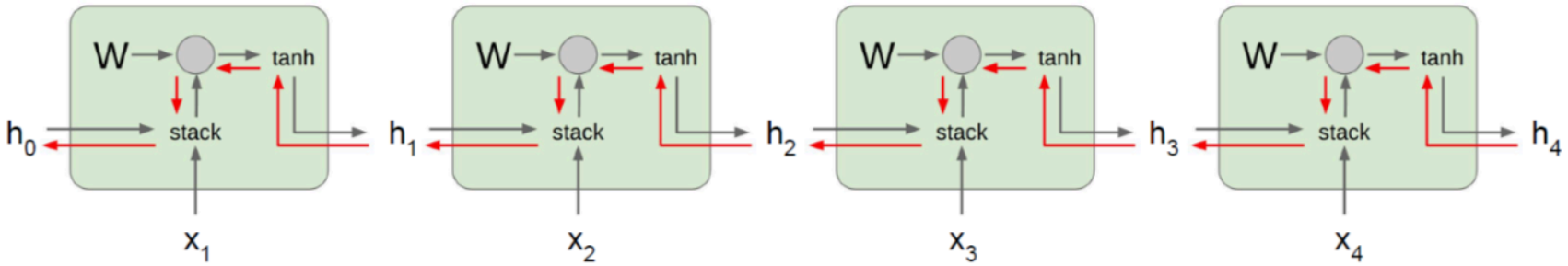
# RNN – Key Issue




$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left((W_{hh} \quad W_{hx})\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

**Backpropagation from $h_t$ to $h_{t-1}$ multiplies by W**

# RNN – Key Issue




Computing gradient of $h_0$ involves many factors of W !

If sequence is long enough and
W > 1, ***exploding gradients!***
W < 1, ***vanishing gradients!***

# RNN – Key Issue

*Exploding gradients:*

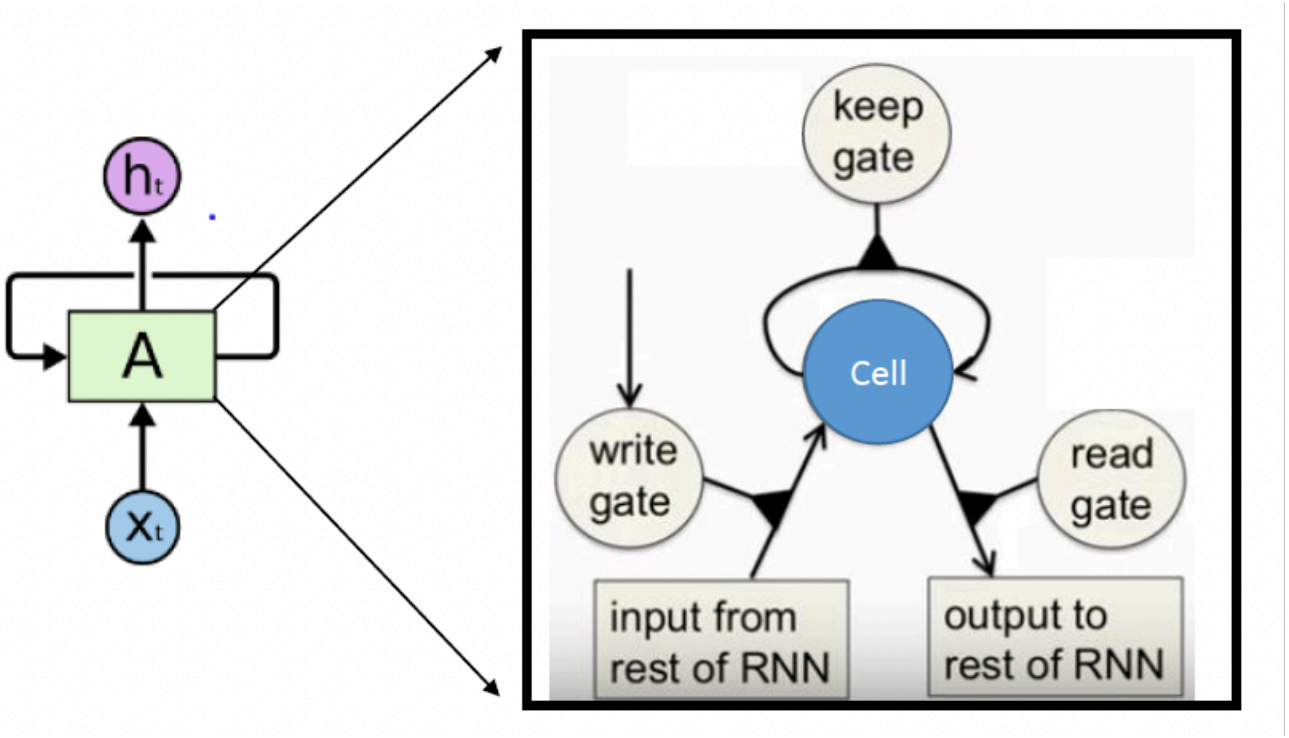
Employ *Gradient Clipping* to scale the gradient (e.g. cut the value).

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
    grad *= (threshold / grad_norm)
```

*Vanishing gradients:*

**Change RNN architecture !**

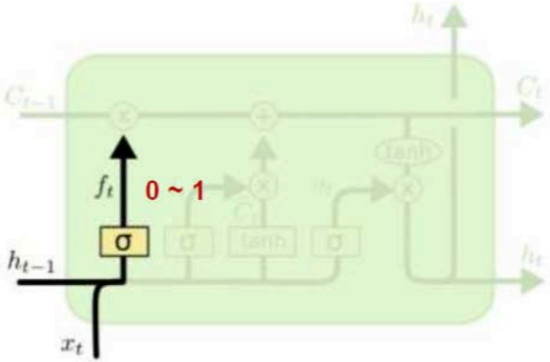
# From RNN to LSTM



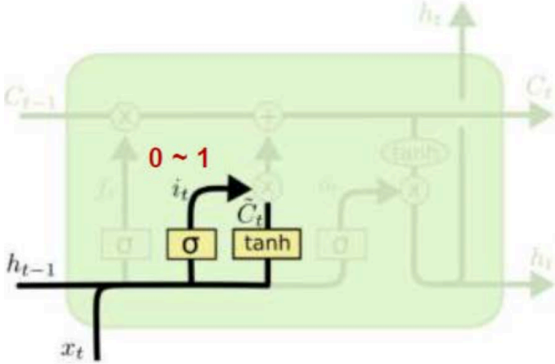
# Long Short Term Memory (LSTM) – Network Structure

[Hochreiter et al., 1997]

Keep Gate:

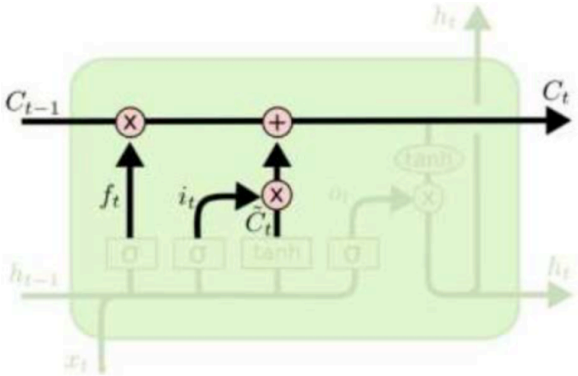



Write Gate:

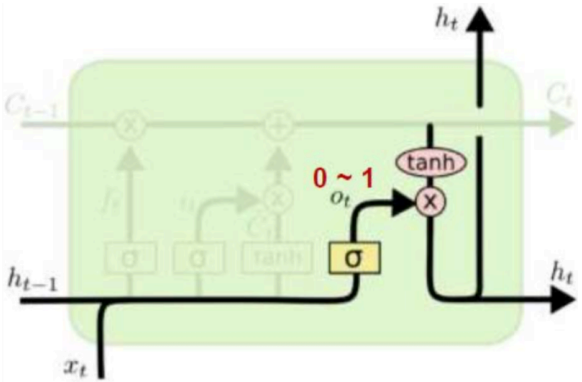


# LSTM – Network Structure

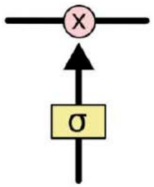
Update Gate:



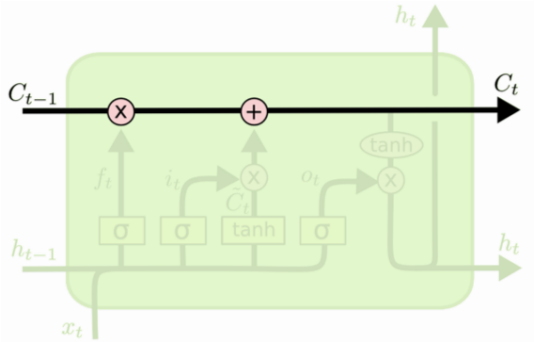

Read Gate:



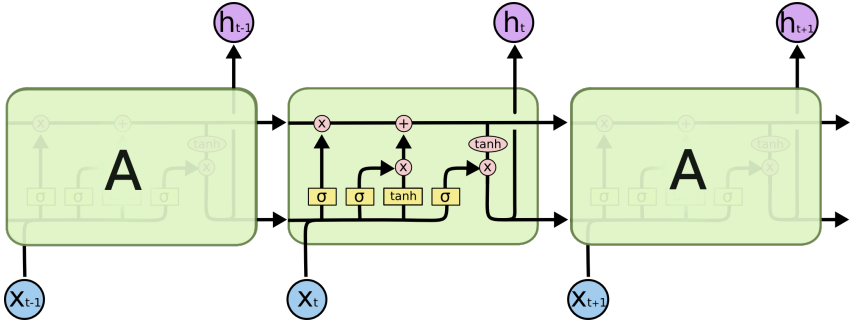
# LSTM – Network Structure




gate

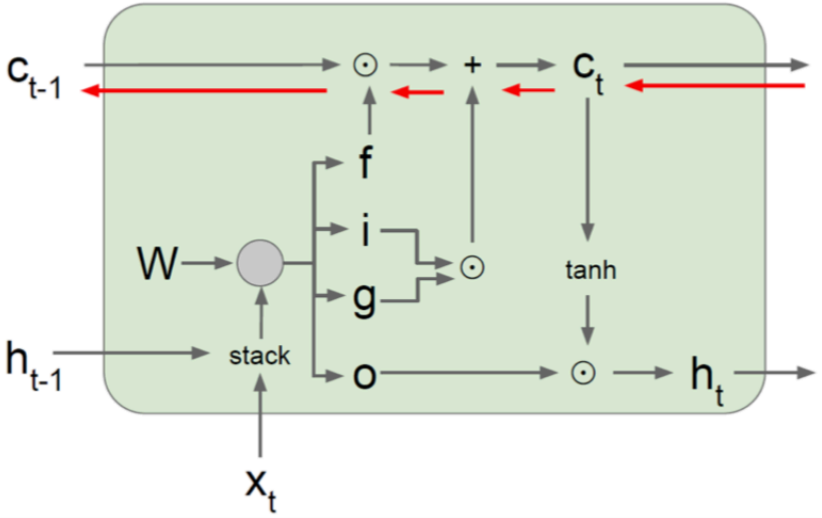



cell state




With the cell state, it runs straight down the entire chain, with only some **minor linear interactions (NOT matrix multiplication like in RNN)** It's very easy for information to just flow along it unchanged.

# LSTM – Network Structure



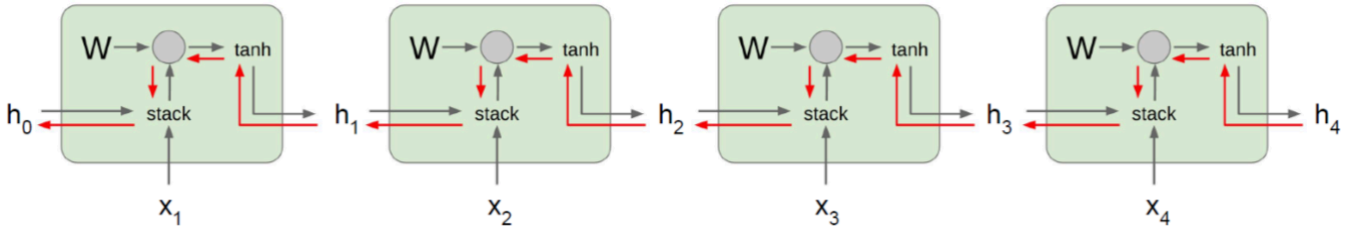

$$\begin{bmatrix} \text{input gate} \\ \text{forget gate} \\ \text{output gate} \\ \text{update gate} \end{bmatrix} \begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

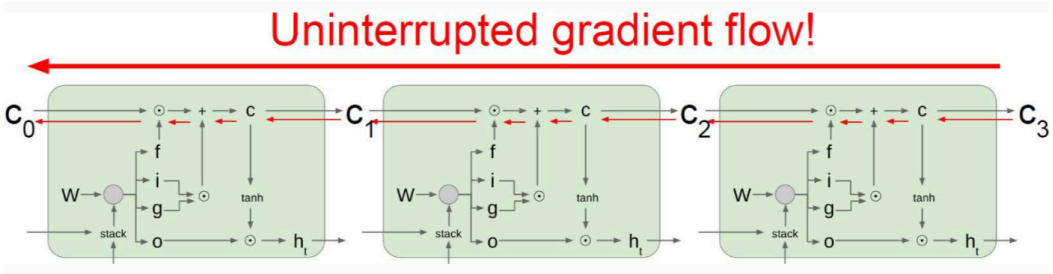
$$h_t = o \odot \tanh(c_t)$$

# LSTM – Network Structure

RNN




LSTM




***Vanishing gradients SOLVED!***

# Summary

- RNNs allow a lot of flexibility in architecture design
- Backward flow of gradients in RNN can explode or vanish.
- Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better/simpler architectures are a hot topic of current research
- Better understanding (both theoretical and empirical) is needed

# References


1. Understanding LSTM http://colah.github.io/posts/2015-08-Understanding-LSTMs/
2. The Unreasonable Effectiveness of Recurrent Neural Networks
   http://karpathy.github.io/2015/05/21/rnn-effectiveness/
3. Recurrent Neural Networks, Fei-Fei Li & Justin Johnson & Serena Yeung
4. An Empirical Exploration of Recurrent Network Architectures
   http://proceedings.mlr.press/v37/jozefowicz15.pdf
5. A Critical Review of Recurrent Neural Networks for Sequence Learning
   https://arxiv.org/pdf/1506.00019.pdf
6. https://www.zhihu.com/question/44895610
7. https://my.oschina.net/u/2719468/blog/662099
8. https://yq.aliyun.com/articles/574218
9. https://juejin.im/post/59ae29d36fb9a024966cac99

# Q&A