Introduction to RNNs

Many materials are from Arun Mallya's

Outline

- Why Recurrent Neural Networks (RNNs)?
- The Vanilla RNN unit
- The RNN forward pass
- BackPropagation Through Time (BPTT)
- The RNN backward pass
- Issues with the Vanilla RNN
- The Long Short-Term Memory (LSTM) unit

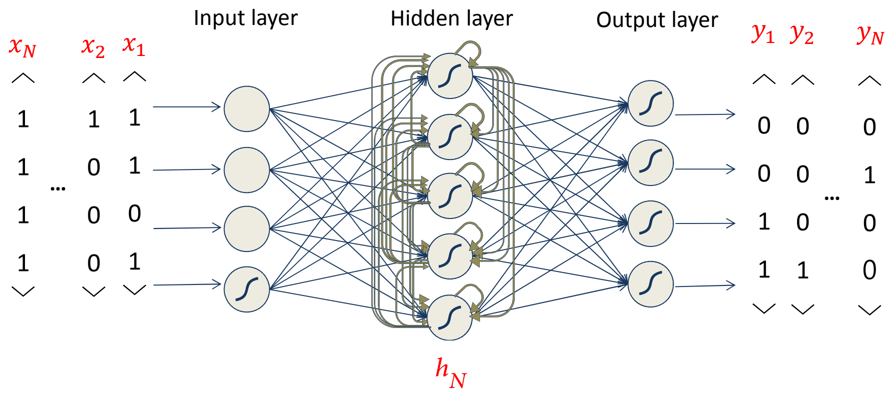
Motivation

- Not all problems can be converted into one with fixed-length inputs and outputs (function)
- Problems such as Speech Recognition or Time-series Prediction require a system to store and use context information
  - Simple case: Output YES if the number of 1s in a variable-length sequence is even, else NO
    1000010101 – YES, 100011 – NO, …
- Hard/Impossible to choose a fixed context window
  - There can always be a new sample longer than anything seen

Recurrent Neural Networks (RNNs)

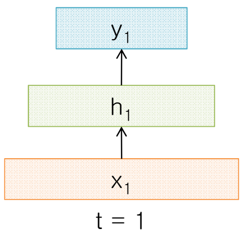
- Recurrent Neural Networks take the previous output of hidden layer as one of the next inputs of hidden layer. (like automata)
  → This input at time t has some historical information about the happenings at time T such that T < t
- RNN is useful as its intermediate state (the previous output of hidden layer) stores information about past inputs.

Revisted - Recurrent Neural Networks

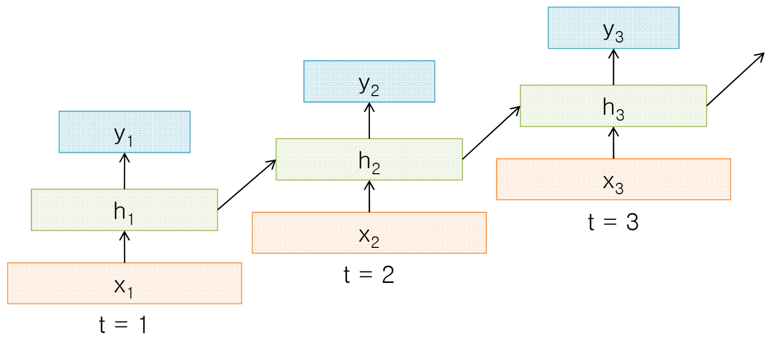


Time-series Training Data

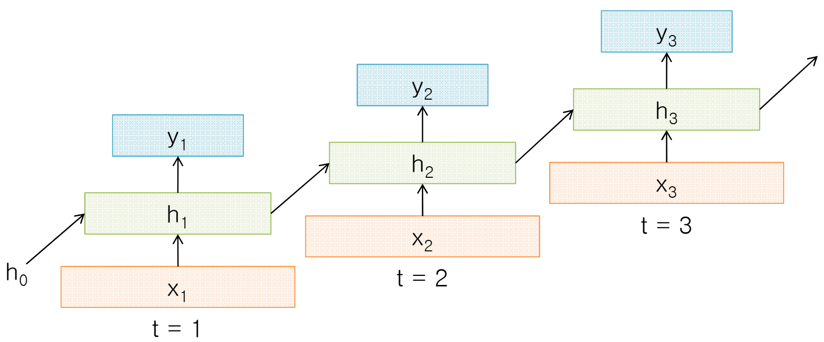
Sample Feed-forward Network



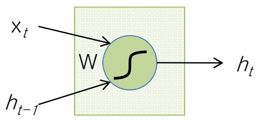
Sample RNN



Sample RNN

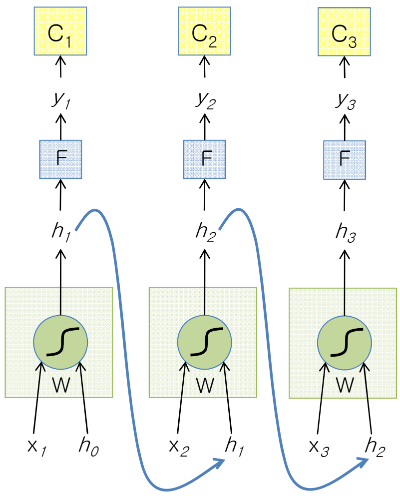


The Vanilla RNN Cell



$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

The Vanilla RNN Forward

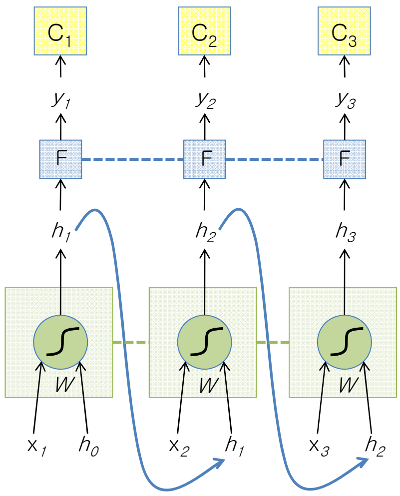


$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$y_t = \mathrm{F}(h_t)$$

$$C_t = \mathrm{Loss}(y_t, \mathrm{GT}_t)$$

The Vanilla RNN Forward



$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$y_t = \mathrm{F}(h_t)$$

$$C_t = \mathrm{Loss}(y_t, \mathrm{GT}_t)$$

------ indicates shared weights

Recurrent Neural Networks (RNNs)

- Note that the weight matrices W, F are shared over time
- Therefore, copies of the RNN cell are made over time (unrolling/unfolding), with different inputs at different time steps

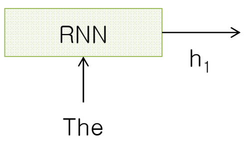
Sentiment Classification

- Classify a restaurant review from Yelp! or movie review from IMDB OR
  …
  as positive or negative

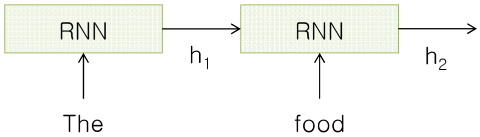
- Inputs: Multiple words, one (or more) sentences
- Outputs: Positive/Negative classification

- "The food was really good"
- "The chicken crossed the road because it was uncooked"

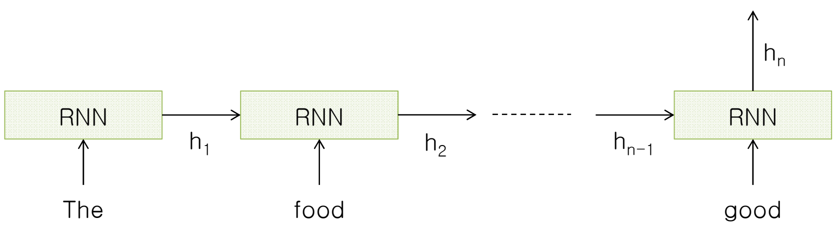
Sentiment Classification



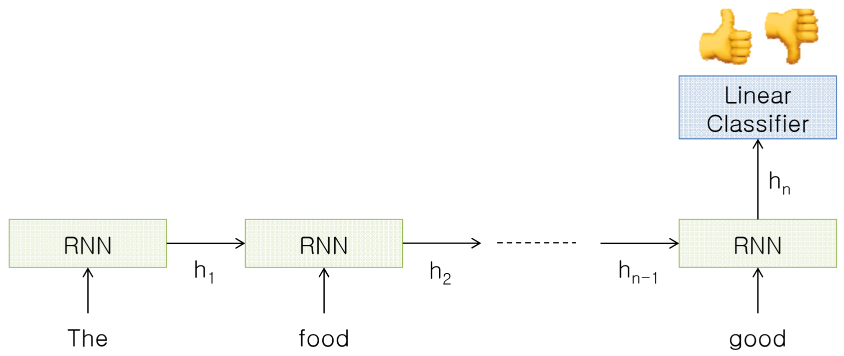
Sentiment Classification



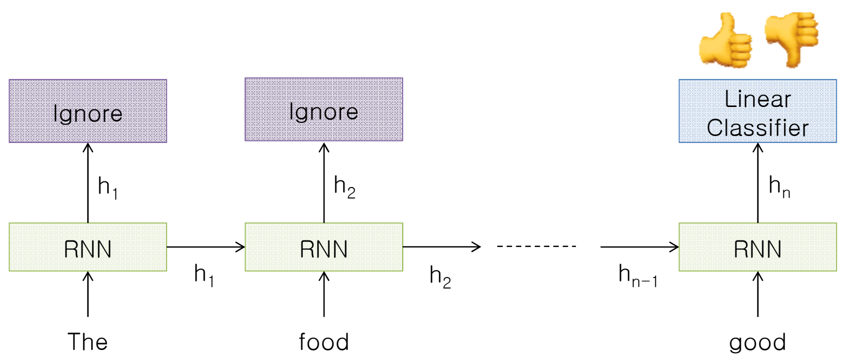
Sentiment Classification



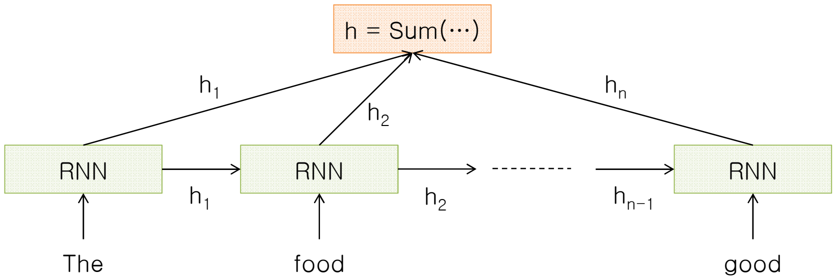
Sentiment Classification



Sentiment Classification



Sentiment Classification



Sentiment Classification

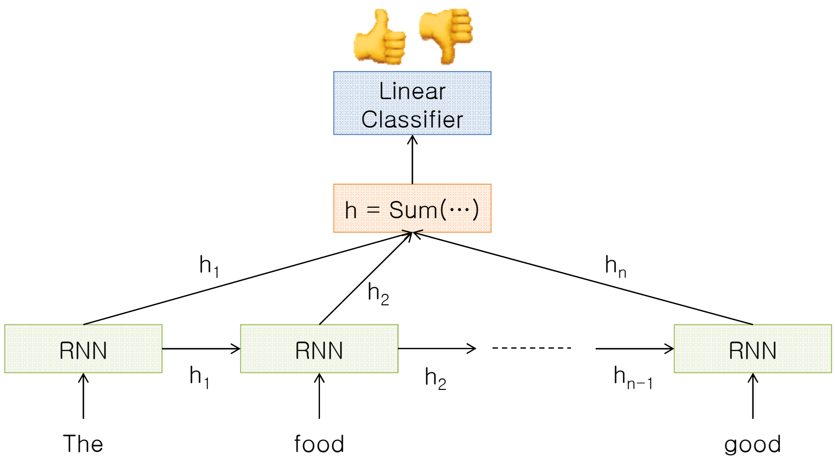


Image Captioning

- Given an image, produce a sentence describing its contents
- Inputs: Image feature (from a CNN)
- Outputs: Multiple words (one simple sentence)

: The dog is hiding

Image Captioning

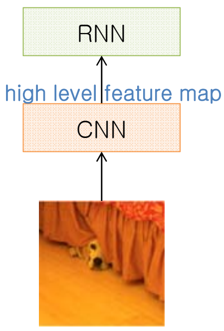


Image Captioning

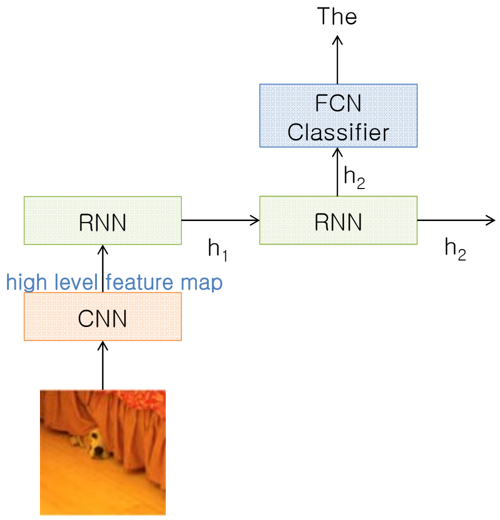
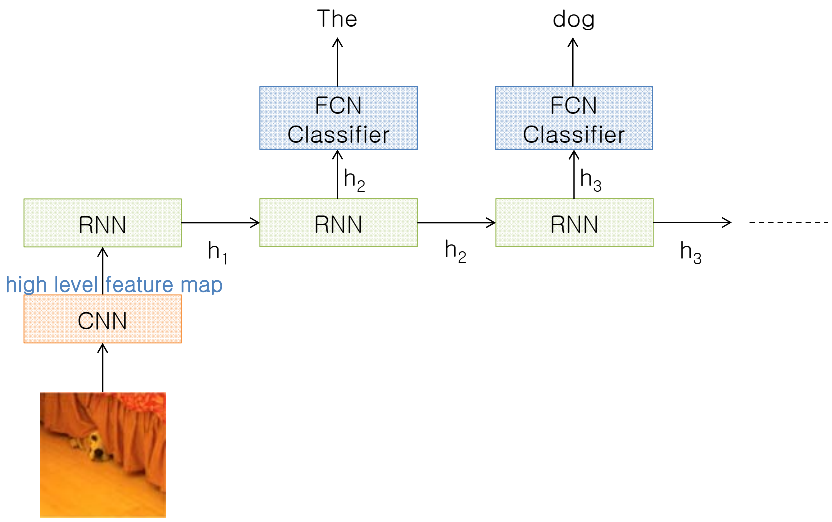
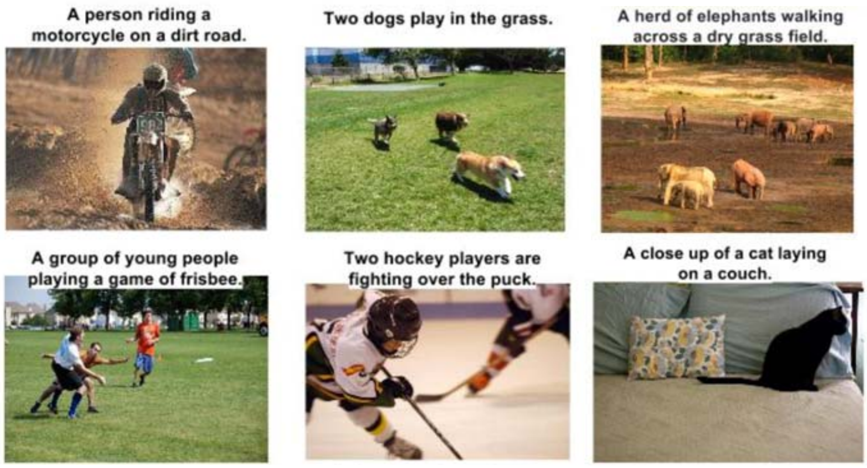


Image Captioning

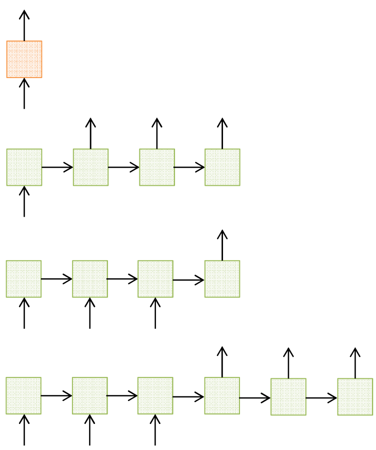


RNN Outputs: Image Captions



[Show and Tell: A Neural Image Caption Generator, CVPR 15]

Input – Output Scenarios



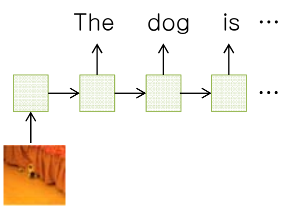
Single – Single one to one	Feed-forward Network
Single – Multiple one to many	Image Captioning
Multiple – Single many to one	Sentiment Classification
Multiple – Multiple many to many	Translation Video Captioning

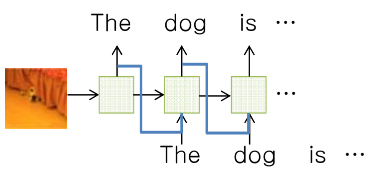
Input – Output Scenarios

Note: We might deliberately choose to frame our problem as a particular input-output scenario for ease of training or better performance.

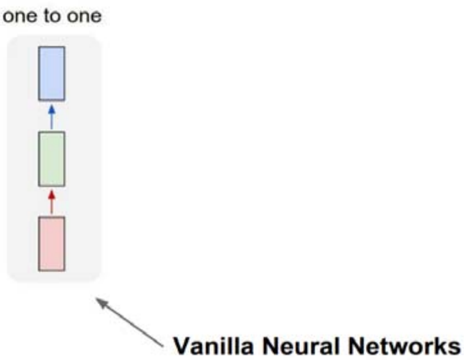
For the example of image captioning, at each time step, provide previous word as input.

→ (Single-Multiple to Multiple-Multiple).

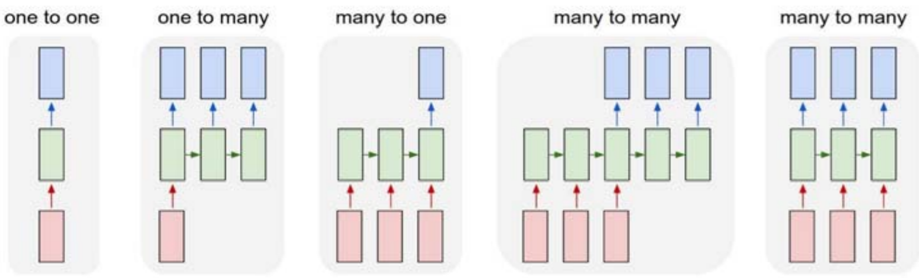




"Vanilla" Neural Network

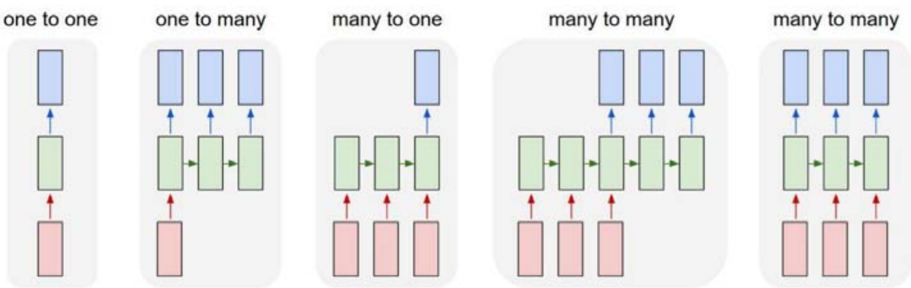


Recurrent Neural Networks: Process Sequences



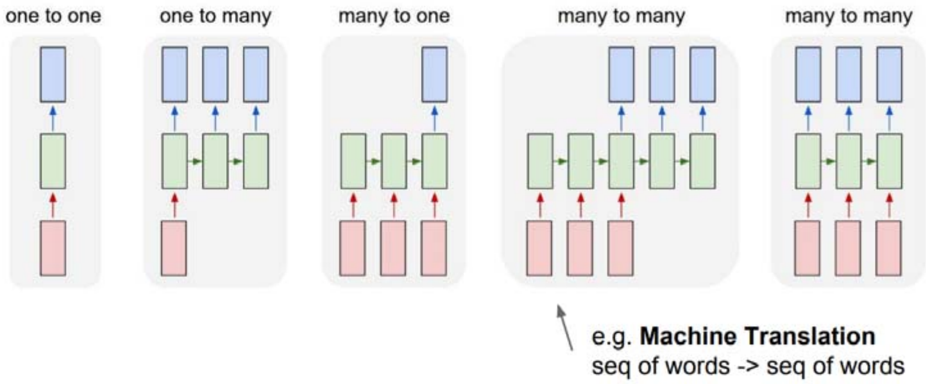
e.g. **Image Captioning**
image -> sequence of words

Recurrent Neural Networks: Process Sequences

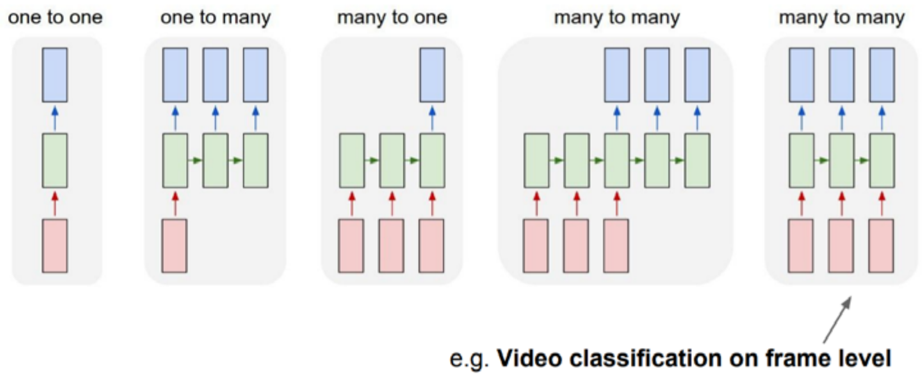


e.g. **Sentiment Classification**
sequence of words -> sentiment

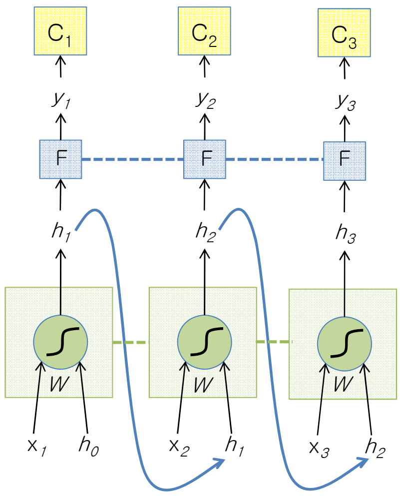
Recurrent Neural Networks: Process Sequences



Recurrent Neural Networks: Process Sequences



The Vanilla RNN Forward



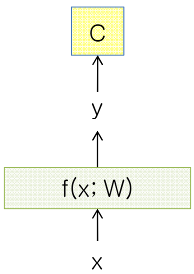
$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$y_t = \mathrm{F}(h_t)$$

$$C_t = \mathrm{Loss}(y_t, \mathrm{GT}_t)$$

"Unfold" network through time by making copies at each time-step

BackPropagation Revisted!



$$y = f(x; W)$$

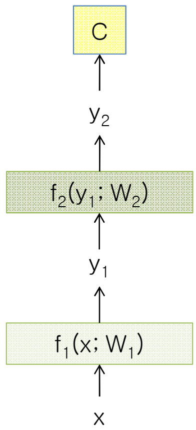
$$C = \mathrm{Loss}(y, y_{GT})$$

SGD Update

$$W \leftarrow W - \eta \frac{\partial C}{\partial W}$$

$$\frac{\partial C}{\partial W} = \left(\frac{\partial C}{\partial y}\right)\left(\frac{\partial y}{\partial W}\right)$$

Multiple Layers

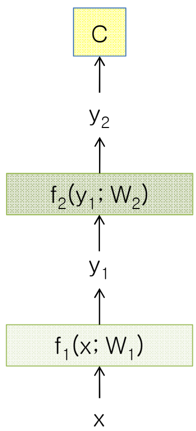


$$y_1 = f_1(x; W_1)$$
$$y_2 = f_2(y_1; W_2)$$
$$C = \text{Loss}(y_2, y_{GT})$$

SGD Update

$$W_2 \leftarrow W_2 - \eta \frac{\partial C}{\partial W_2}$$
$$W_1 \leftarrow W_1 - \eta \frac{\partial C}{\partial W_1}$$

Chain Rule for Gradient Computation Revisted!



$$y_1 = f_1(x; W_1)$$
$$y_2 = f_2(y_1; W_2)$$
$$C = \text{Loss}(y_2, y_{GT})$$

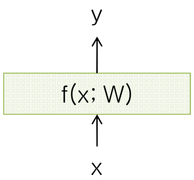
Find $\dfrac{\partial C}{\partial W_1}, \dfrac{\partial C}{\partial W_2}$

$$\frac{\partial C}{\partial W_2} = \left(\frac{\partial C}{\partial y_2}\right)\left(\frac{\partial y_2}{\partial W_2}\right)$$

$$\frac{\partial C}{\partial W_1} = \left(\frac{\partial C}{\partial y_1}\right)\left(\frac{\partial y_1}{\partial W_1}\right)$$
$$= \left(\frac{\partial C}{\partial y_2}\right)\left(\frac{\partial y_2}{\partial y_1}\right)\left(\frac{\partial y_1}{\partial W_1}\right)$$

Application of the Chain Rule

Chain Rule for Gradient Computation Revisted!



Given: $\left(\frac{\partial C}{\partial y}\right)$

We are interested in computing: $\left(\frac{\partial C}{\partial W}\right), \left(\frac{\partial C}{\partial x}\right)$

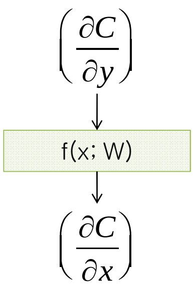
Intrinsic to the layer are:

$\left(\frac{\partial y}{\partial W}\right)$ – How does output change due to params

$\left(\frac{\partial y}{\partial x}\right)$ – How does output change due to inputs

$$\left(\frac{\partial C}{\partial W}\right) = \left(\frac{\partial C}{\partial y}\right)\left(\frac{\partial y}{\partial W}\right) \quad \left(\frac{\partial C}{\partial x}\right) = \left(\frac{\partial C}{\partial y}\right)\left(\frac{\partial y}{\partial x}\right)$$

Chain Rule for Gradient Computation Revisted!



Given: $\left(\frac{\partial C}{\partial y}\right)$

We are interested in computing: $\left(\frac{\partial C}{\partial W}\right), \left(\frac{\partial C}{\partial x}\right)$

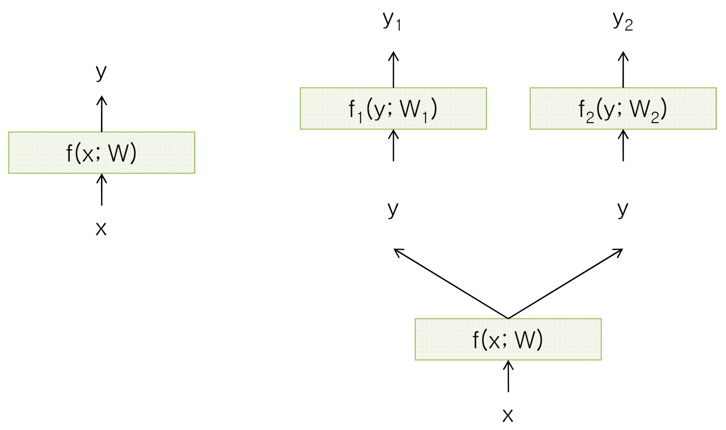
Intrinsic to the layer are:

$\left(\frac{\partial y}{\partial W}\right)$ – How does output change due to params

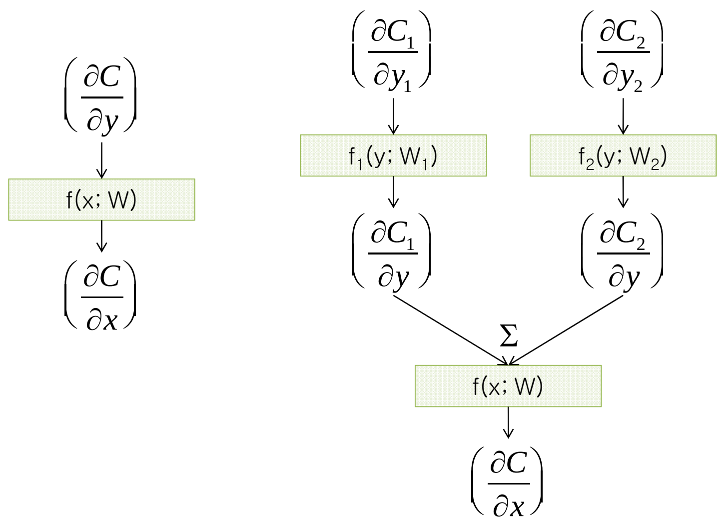
$\left(\frac{\partial y}{\partial x}\right)$ – How does output change due to inputs

$$\left(\frac{\partial C}{\partial W}\right) = \left(\frac{\partial C}{\partial y}\right)\left(\frac{\partial y}{\partial W}\right) \quad \left(\frac{\partial C}{\partial x}\right) = \left(\frac{\partial C}{\partial y}\right)\left(\frac{\partial y}{\partial x}\right)$$

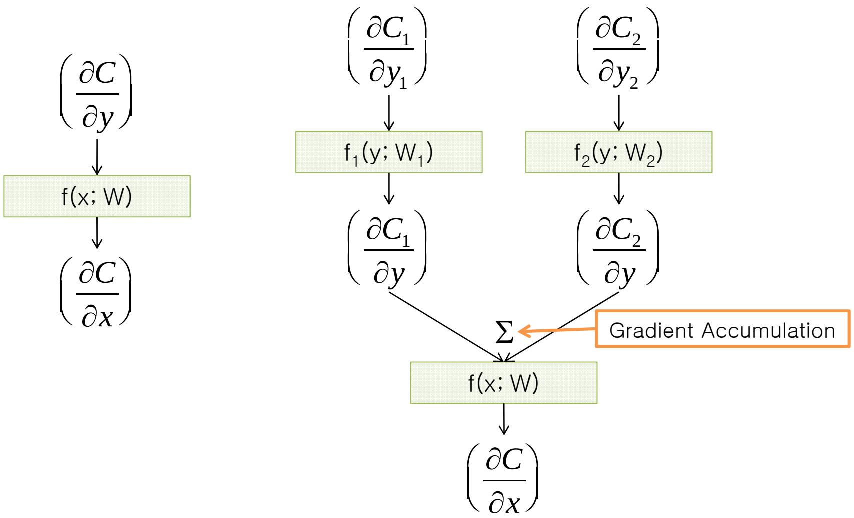
Extension to Computational Graphs



Extension to Computational Graphs



Extension to Computational Graphs

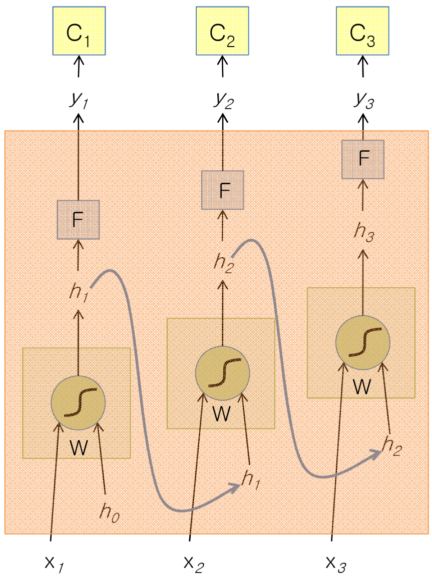


BackPropagation Through Time (BPTT)

- One of the methods used to train RNNs
- The unfolded network (used during forward pass) is treated as one big feed-forward network
- This unfolded network accepts the whole time series as input

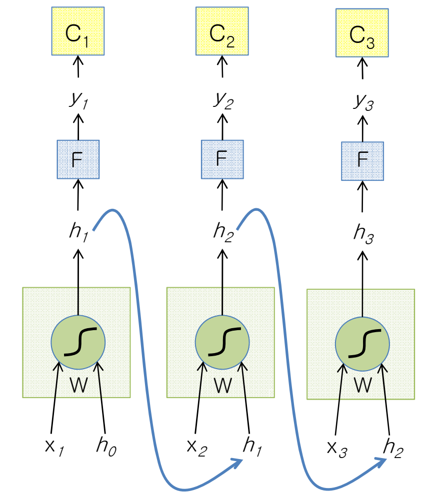
- The weight updates are computed for each copy in the unfolded network, then summed (or averaged) and then applied to the RNN weights

The Unfolded Vanilla RNN Forward



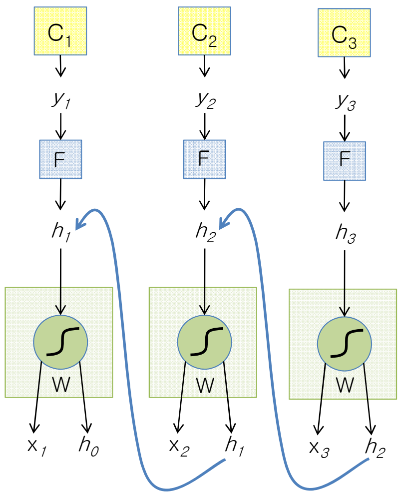
- Treat the unfolded network as one big feed-forward network!

The Unfolded Vanilla RNN Forward



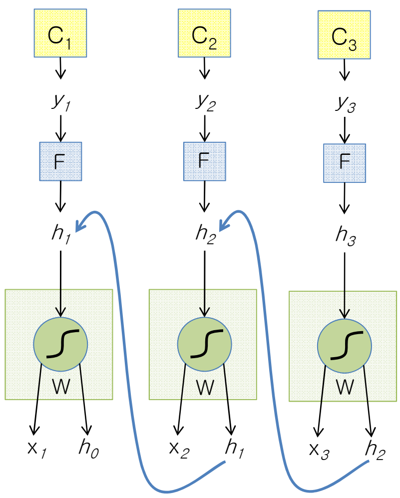
- This big network takes in entire sequence as an input

The Unfolded Vanilla RNN Backward



- Compute gradients through the usual backpropagation
- Update shared weights

The Unfolded Vanilla RNN Backward



$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$y_t = \mathrm{F}(h_t)$$

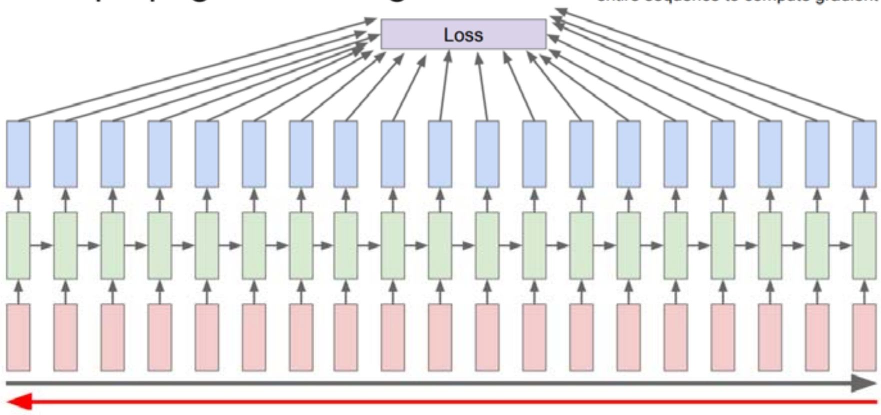
$$C_t = \mathrm{Loss}(y_t, \mathrm{GT}_t)$$

$$\frac{\partial C_t}{\partial h_1} = \left(\frac{\partial C_t}{\partial y_t}\right)\left(\frac{\partial y_t}{\partial h_1}\right)$$

$$= \left(\frac{\partial C_t}{\partial y_t}\right)\left(\frac{\partial y_t}{\partial h_t}\right)\underbrace{\left(\frac{\partial h_t}{\partial h_{t-1}}\right)\cdots\left(\frac{\partial h_2}{\partial h_1}\right)}$$

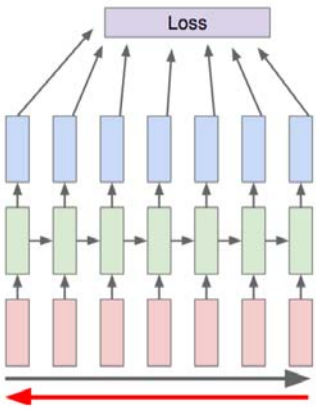
Backpropagation through time

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



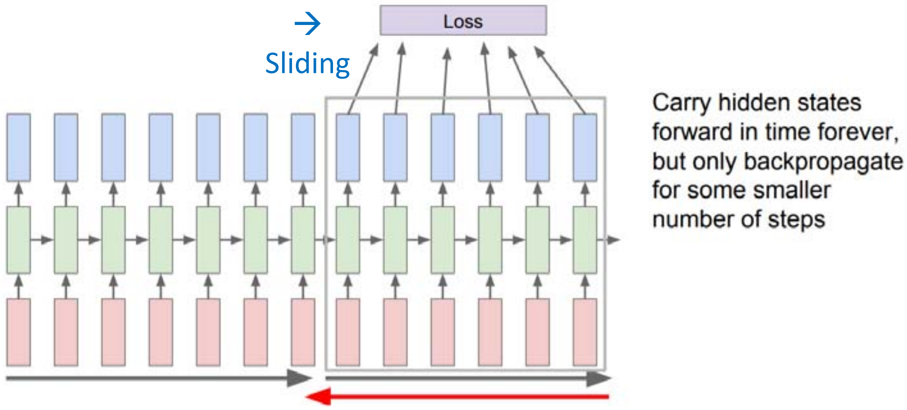
Vanishing and Exploding Gradients due to many multiplications of derivatives of activations and weights

Truncated Backpropagation through time

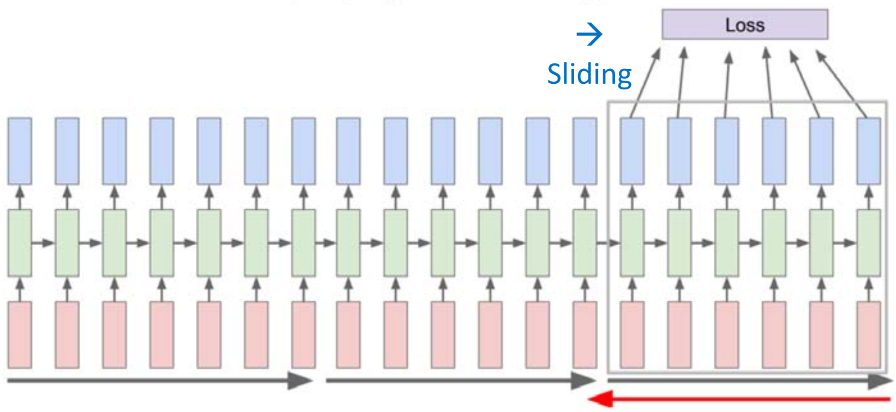


Run forward and backward through chunks of the sequence instead of whole sequence

Truncated Backpropagation through time

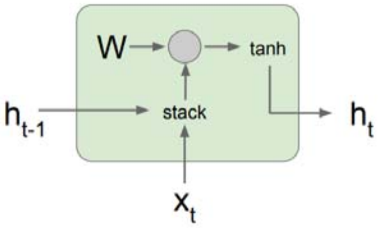


Truncated Backpropagation through time



Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

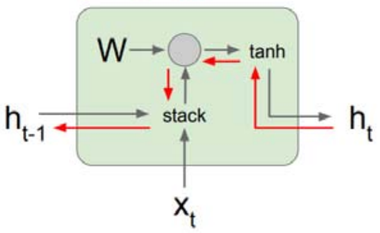


$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

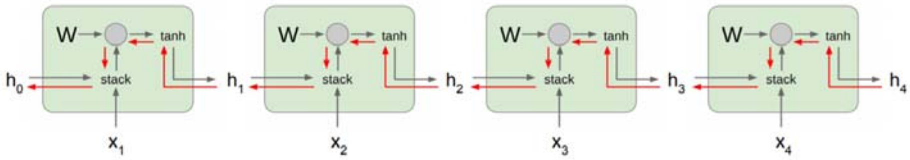
Backpropagation from h_t to h_{t-1} multiplies by W (actually W_{hh}^T)



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

Vanilla RNN Gradient Flow

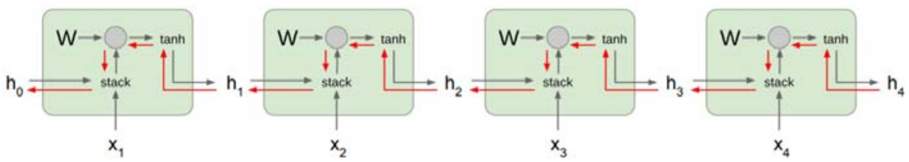
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



Computing gradient of h_0 involves many factors of W

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



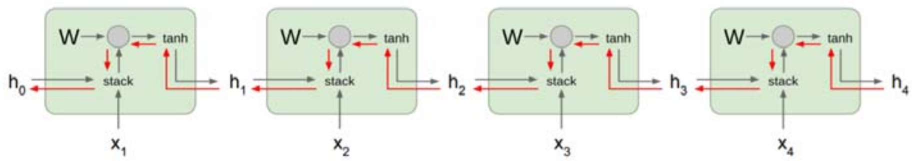
Computing gradient of h_0 involves many factors of W

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



Computing gradient of h_0 involves many factors of W

Largest singular value > 1:
Exploding gradients

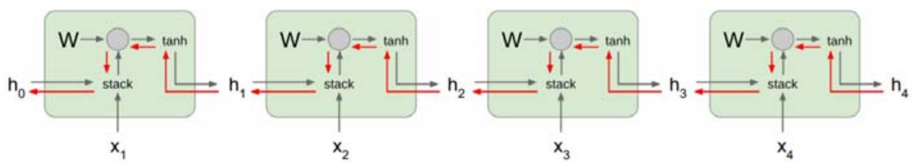
→ **Gradient clipping**: Scale gradient if its norm is too big

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

Largest singular value < 1:
Vanishing gradients

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



Computing gradient of h_0 involves many factors of W

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

→ Change RNN architecture

e.g. Identity relationship between the hidden states

The Identity Relationship

- Recall $\frac{\partial C_t}{\partial h_1} = \left(\frac{\partial C_t}{\partial y_t}\right)\left(\frac{\partial y_t}{\partial h_1}\right)$ $\qquad h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$

$$= \left(\frac{\partial C_t}{\partial y_t}\right)\left(\frac{\partial y_t}{\partial h_t}\right)\overbrace{\left(\frac{\partial h_t}{\partial h_{t-1}}\right)\cdots\left(\frac{\partial h_2}{\partial h_1}\right)}$$

$y_t = \mathrm{F}(h_t)$

$C_t = \mathrm{Loss}(y_t, \mathrm{GT}_t)$

- Suppose that instead of a matrix multiplication, we had an identity relationship between the hidden states

$$h_t = \mathrm{I}(h_{t-1}) + w(x_t) = h_{t-1} + w(x_t)$$

$$\boxed{\Rightarrow \left(\frac{\partial h_t}{\partial h_{t-1}}\right) = 1}$$

- The gradient does not decay neither explode as the error is propagated all the way back, also known as "Constant Error Flow"

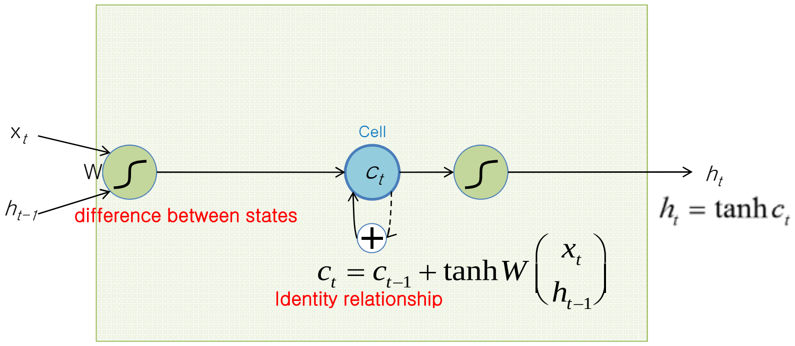
Long Short-Term Memory (LSTM)

- The LSTM is motivated from this idea of "Constant Error Flow" for RNNs, which ensures that gradients don't decay neither explode

- The key component is a memory cell that acts like an accumulator (contains the identity relationship) over time

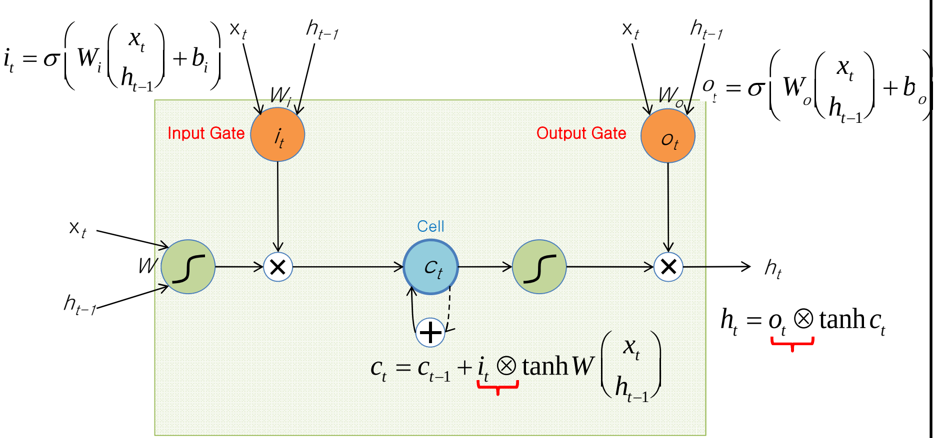
- Instead of computing new state as a matrix product with the old state, it rather computes the difference between them. Expressivity is the same, but gradients are better behaved

Starting Point of the LSTM Idea

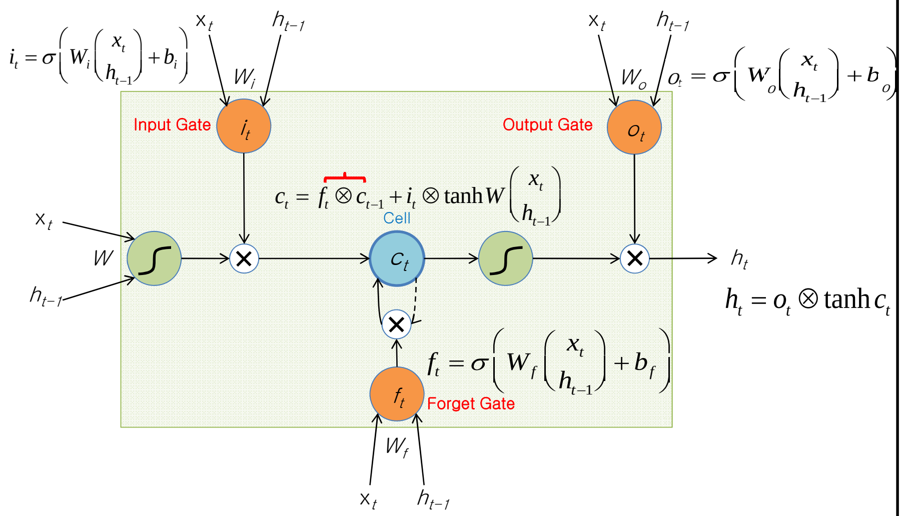


* Dashed line indicates time-lag

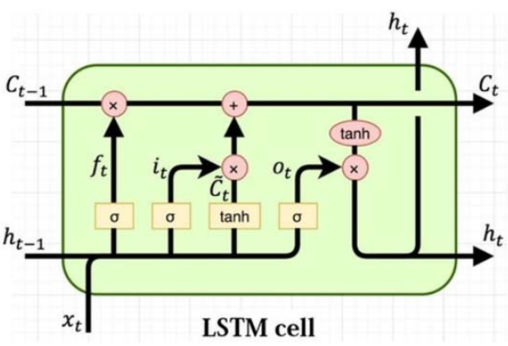
The Detail of LSTM Cell



The More Detail of LSTM Cell



LSTM Cell



$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$
$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$
$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$h_t = \tanh(C_t) * o_t$$

* *In previous slide,*

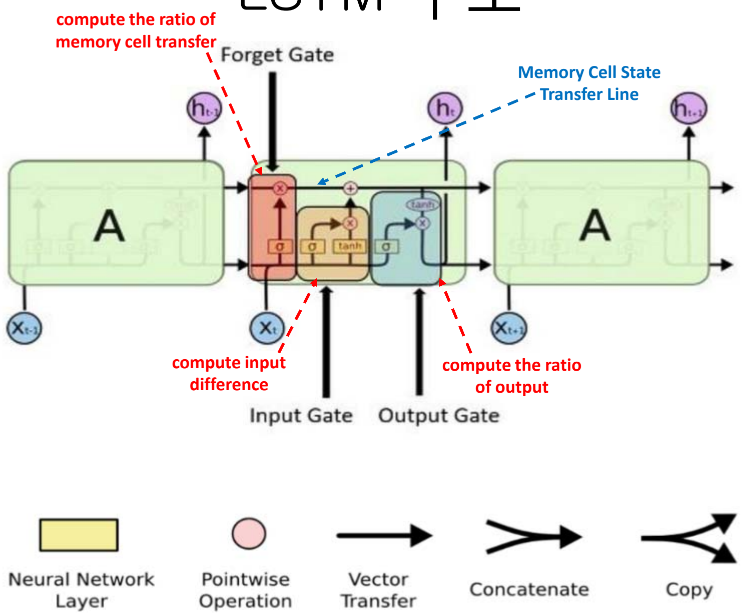
$W_i = [\, U^i \,;\, W^i \,]$

$W_f = [\, U^f \,;\, W^f \,]$

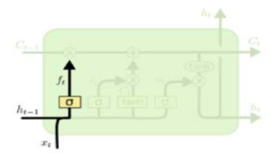
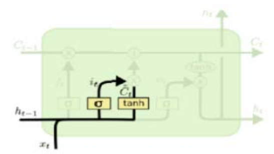
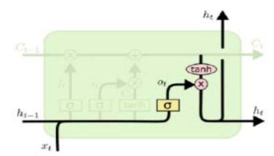
$W_o = [\, U^o \,;\, W^o \,]$

$W = [\, U^g \,;\, W^g \,]$

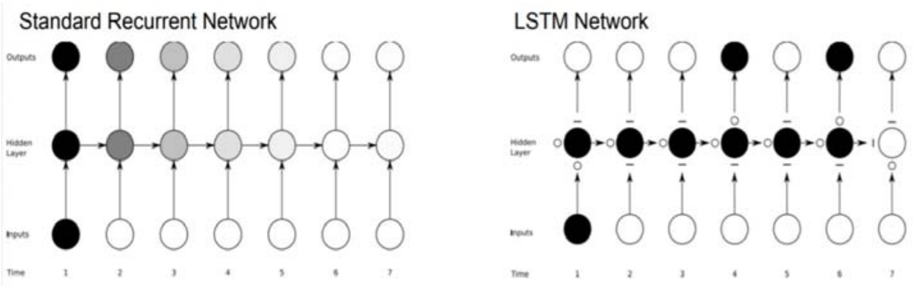
LSTM 구조



Role of Gates

Cell State 정보 선택 과정		– Forget Gate Layer – 시그모이드 레이어로 만들어짐
입력 정보 선택 과정		– Input Gate Layer – 업데이트 Data 결정 – Cell State에 더해짐
출력 정보 선택 과정		– Output Gate Layer – 출력 Data 결정 – 다음 노드로 전파

LSTM reduces vanishing or exploding gradient problem



Graves et al 2013

- The darker the shade, the greater the sensitivity
- The sensitivity decays exponentially over time as new inputs overwrite the activation of hidden unit and the network 'forgets' the first input

LSTM – Forward/Backward

For details, go to: Illustrated LSTM Forward and Backward Pass
http://arunmallya.github.io/writeups/nn/lstm/index.html#/1

Summary

- RNNs allow for processing of variable length inputs and outputs by maintaining state information across time steps
- Various Input-Output scenarios are possible (Single(One)/Multiple(Many))
- Vanilla RNNs are improved upon by LSTMs which address the vanishing or exploding gradient problem through the flow of hidden state difference
- Exploding gradients are also handled by gradient clipping