Recurrent neural network

Hung-Hsuan Chen

Most slides are taken from Fei-Fei Li @Stanford

Sequential pattern mining

- The current output depends on the neighboring (e.g., previous or next) outputs
  - Stock market: today's price is dependent on yesterday's price
  - Grammar parser: the POS (Part-of-Speech) of a word depends on the POS of its neighboring words

Sequential pattern mining by SVM/logistic regression/decision trees/NN/…

- The "sequential" info has to be encoded into the feature manually
- E.g.,
  - Stock market: use the price of yesterday, last 3 days, last week, etc. as the features to predict today's price
  - Grammar parser: use the predicted POS of the next word and the predicted POS of the previous word as the features to predict the POS of the current word

“Vanilla” Neural Network

“Vanilla” neural network (based on the definition in “The Elements of Statistical Learning”)

- A single hidden layer back-propagation network
- a.k.a., single layer perceptron

Vanilla Neural Networks

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

e.g. **Machine Translation**
seq of words -> seq of words

Recurrent Neural Networks: Process Sequences

e.g. **Video classification on frame level**

Recurrent Neural Network

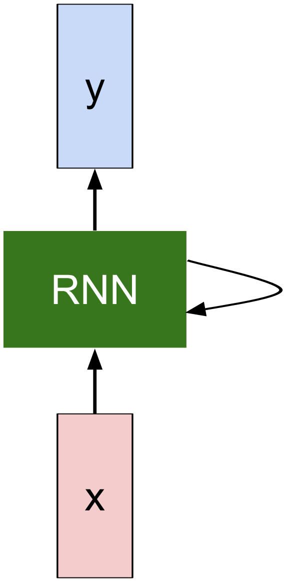
We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

new state

some function with parameters W

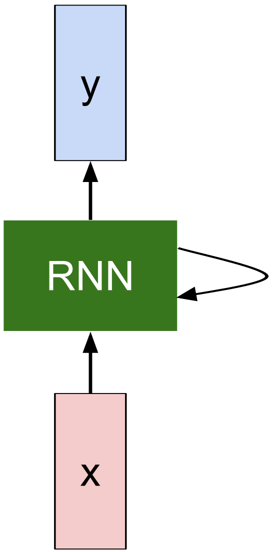
old state

input vector at some time step

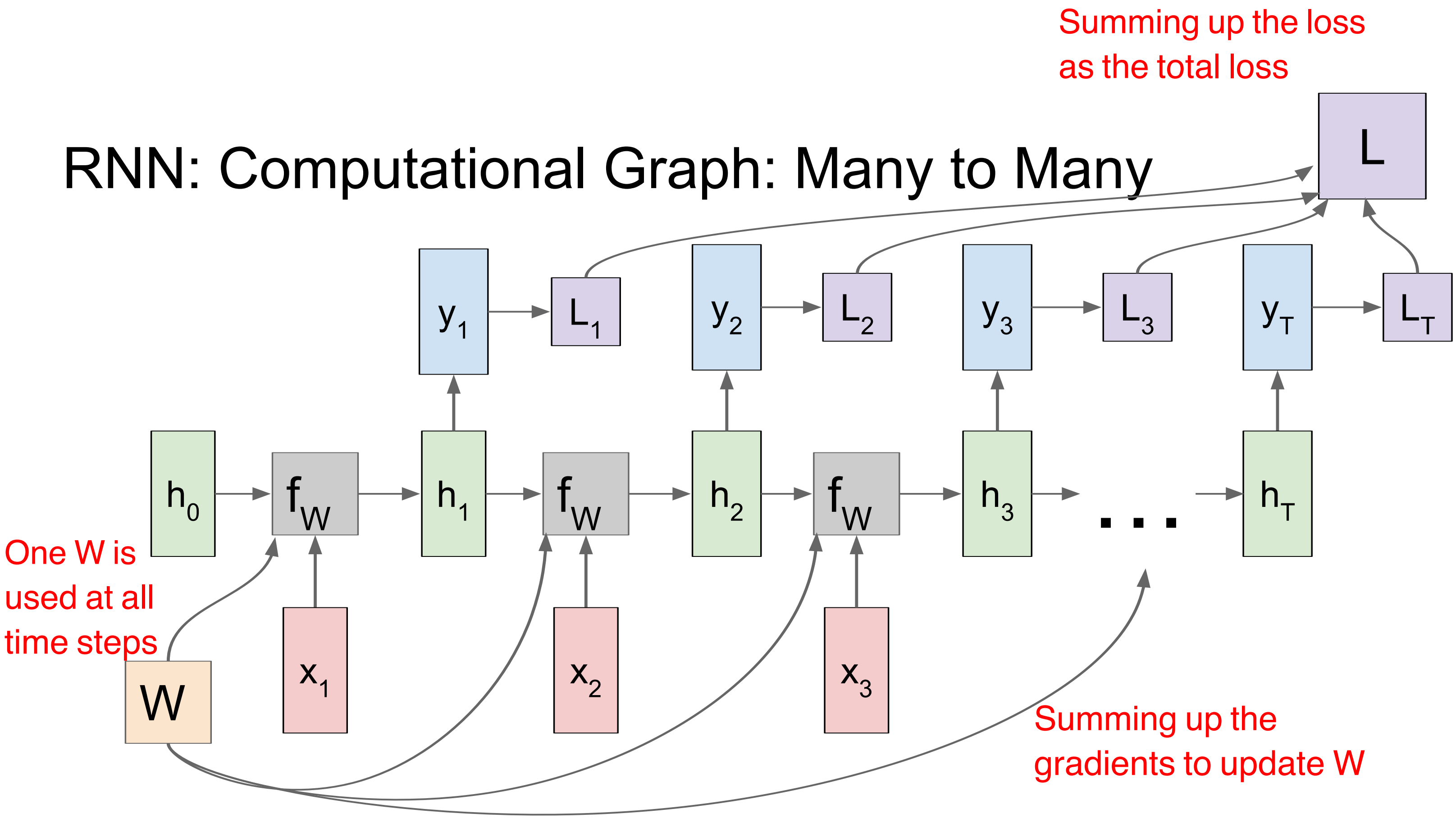


(Vanilla) Recurrent Neural Network

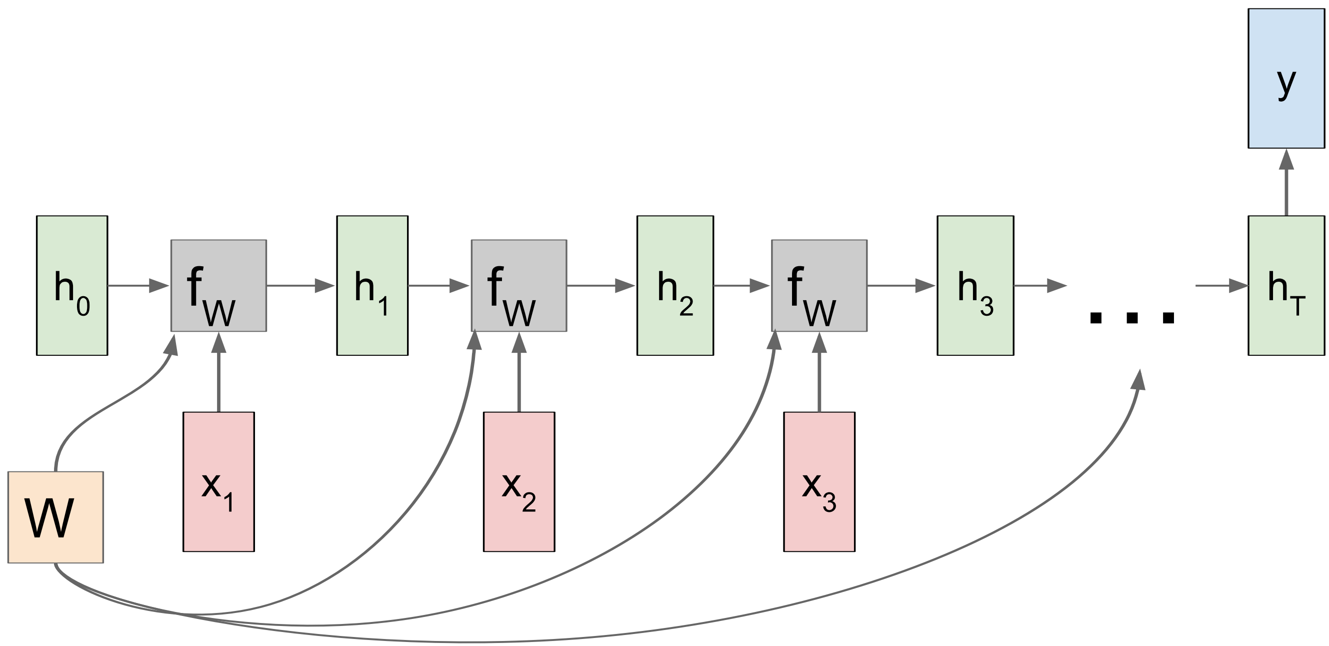
The state consists of a single *“hidden”* vector **h**:



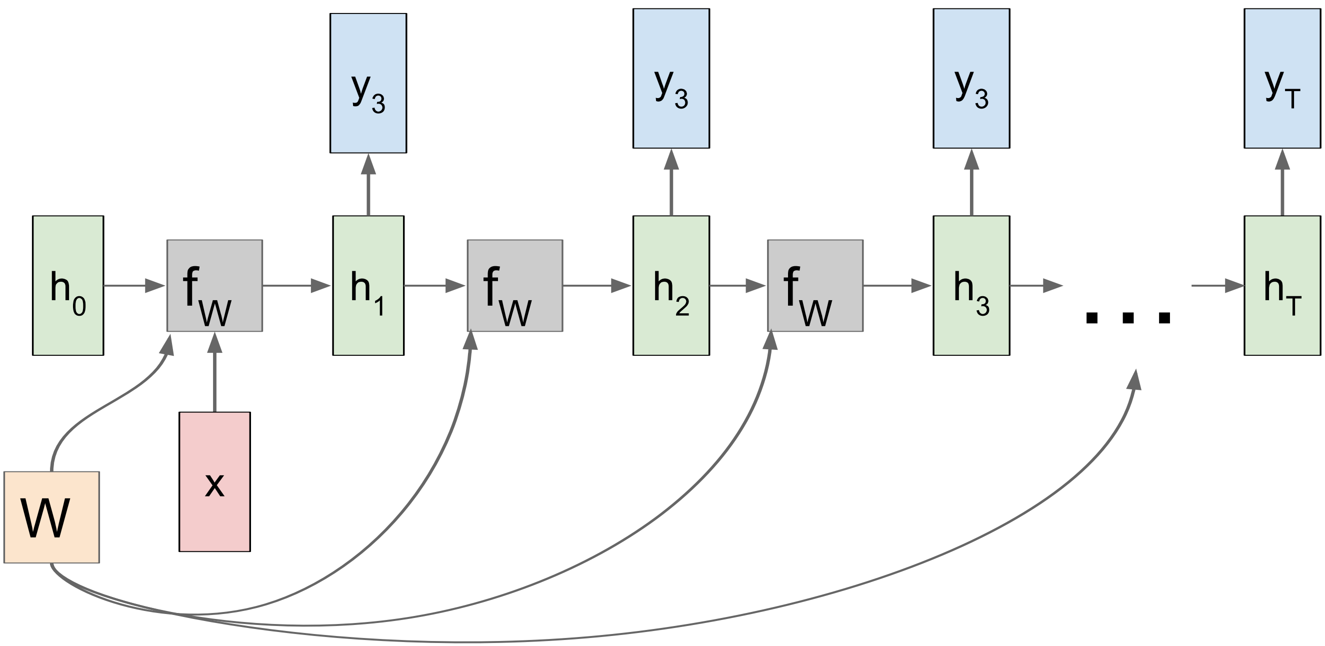
RNN: Computational Graph: Many to Many



RNN: Computational Graph: Many to One

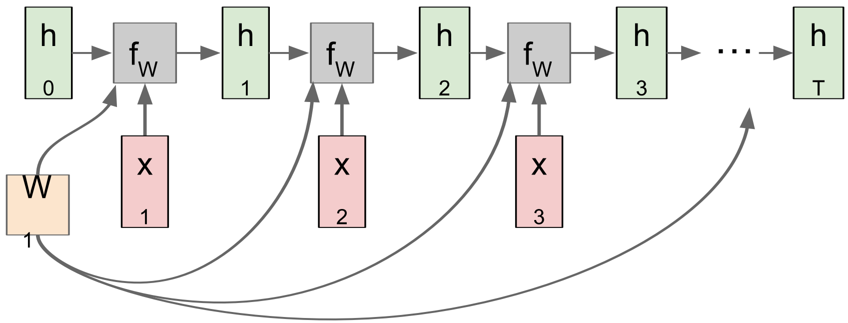


RNN: Computational Graph: One to Many

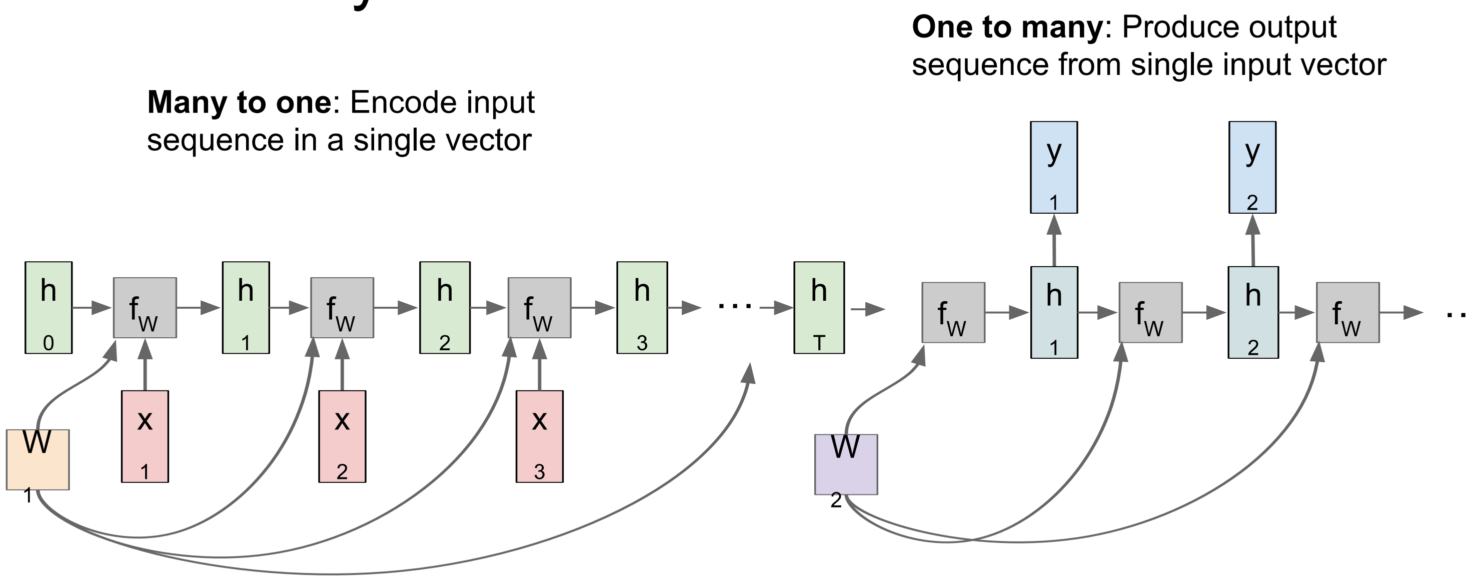


Sequence to Sequence: Many-to-one + one-to-many

Many to one: Encode input sequence in a single vector



Sequence to Sequence: Many-to-one + one-to-many



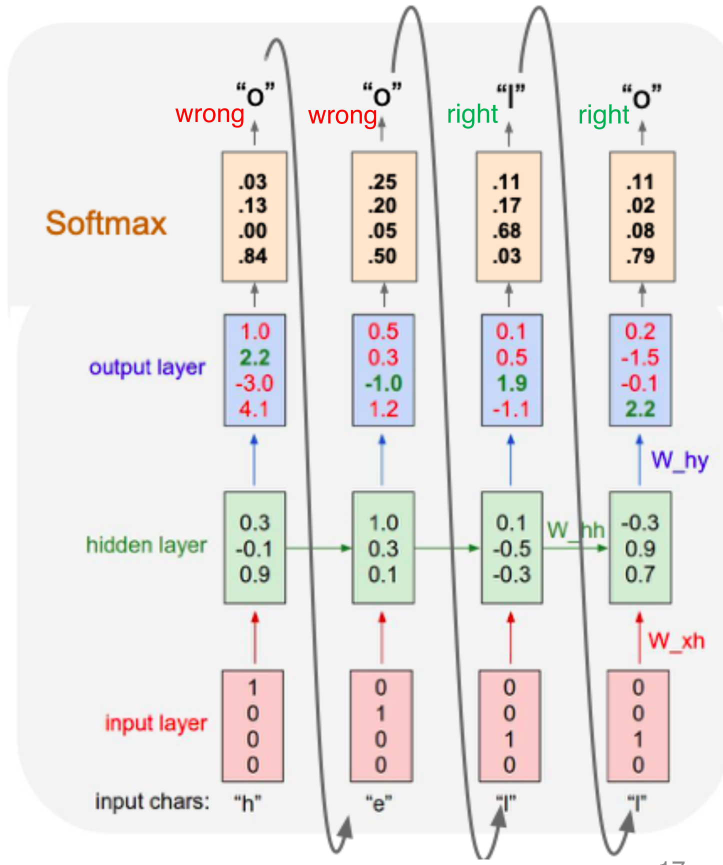
Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

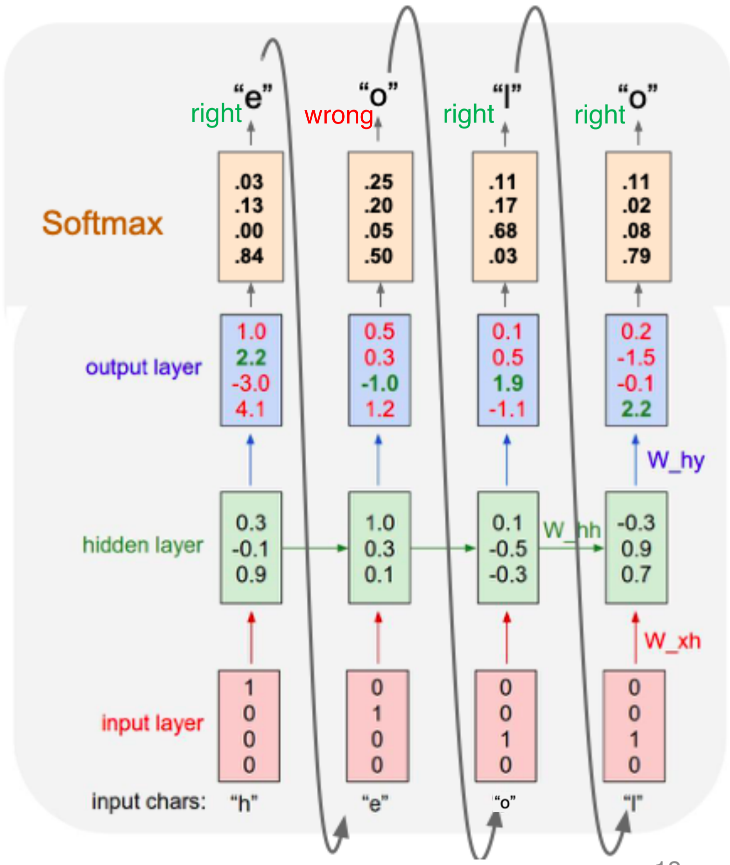
Example: character-level Language Model – training

- Vocabulary: [‘h’, ‘e’, ‘l’, ‘o’]
- Training sequence: “hello”
- At training-time, even if is incorrect, the correct is used as



Example: character-level Language Model – test

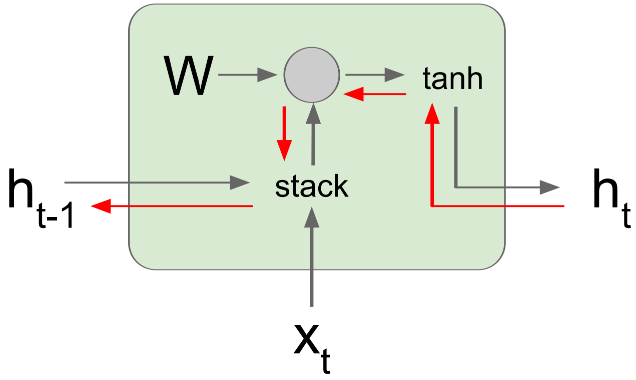
- Vocabulary: [‘h’, ‘e’, ‘l’, ‘o’]
- At test-time, sample and use is used as
  - E.g., given ‘h’ and ‘e’, the network predicts ‘o’ (incorrect), which is used as the next input



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)



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_0 h_1 h_2 h_3 h_4

x_1 x_2 x_3 x_4

Computing gradient of h_0 involves many factors of W (and repeated tanh)

If we ignore tanh function:

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_0 h_1 h_2 h_3 h_4

x_1 x_2 x_3 x_4

Computing gradient of h_0 involves many factors of W (and repeated tanh)

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

h_0 h_1 h_2 h_3 h_4

x_1 x_2 x_3 x_4

Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

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

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

h_0 h_1 h_2 h_3 h_4

x_1 x_2 x_3 x_4

Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

→ Change RNN architecture

LSTM:

Long Short Term Memory (LSTM)

Vanilla RNN

Vanilla RNN:

Maintain one hidden state for every time step

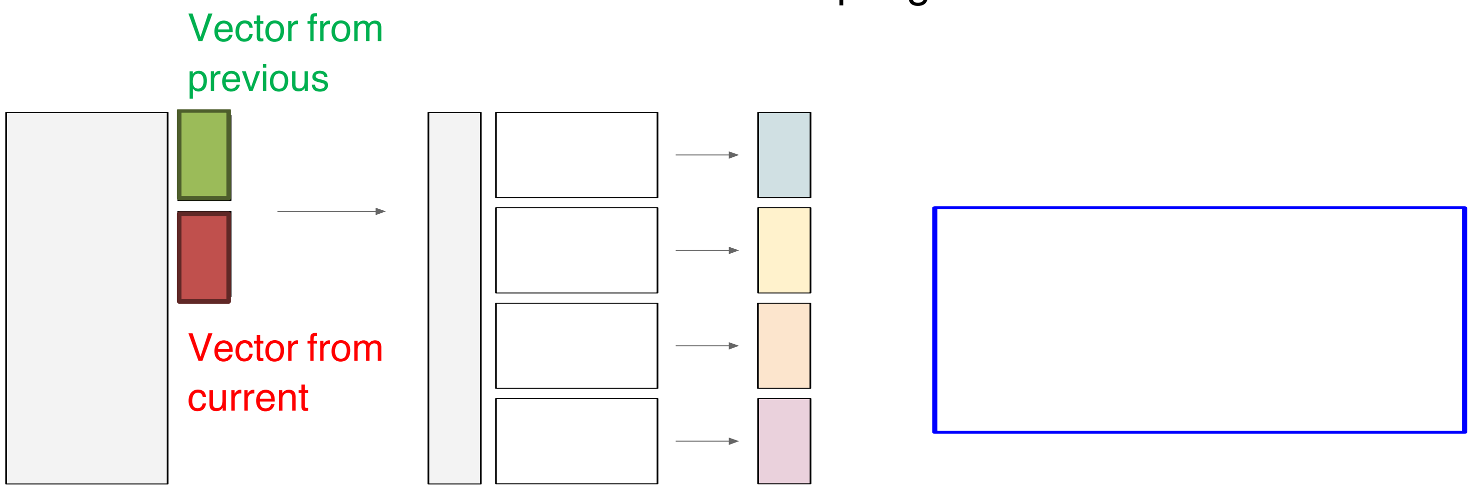
LSTM

Maintain one hidden state and one cell state for every time step:

- represents short-term state
- represents long-term state

Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997

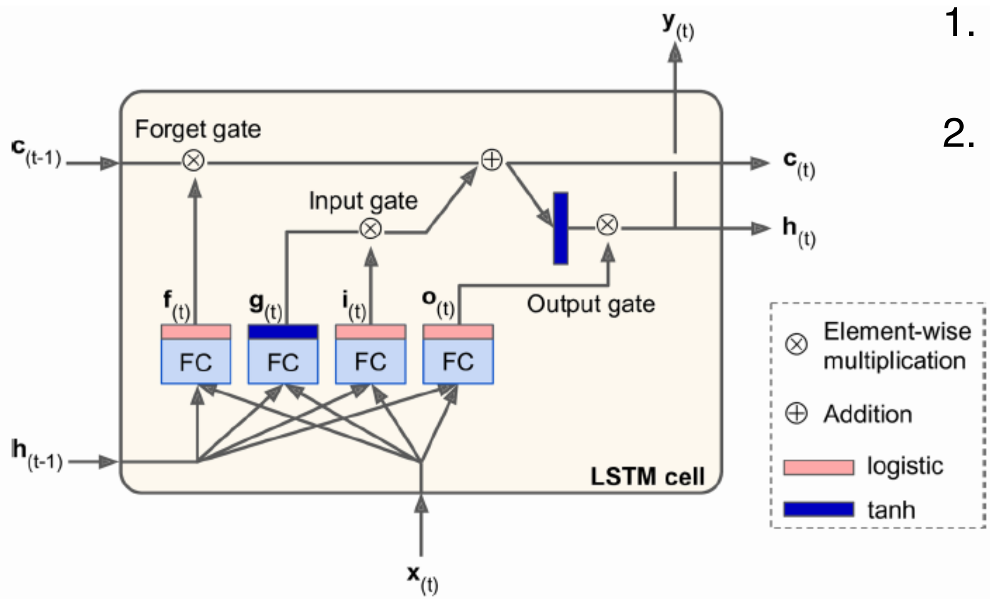
- : combining information from and
- : forget gate
- : input gate
- : output gate

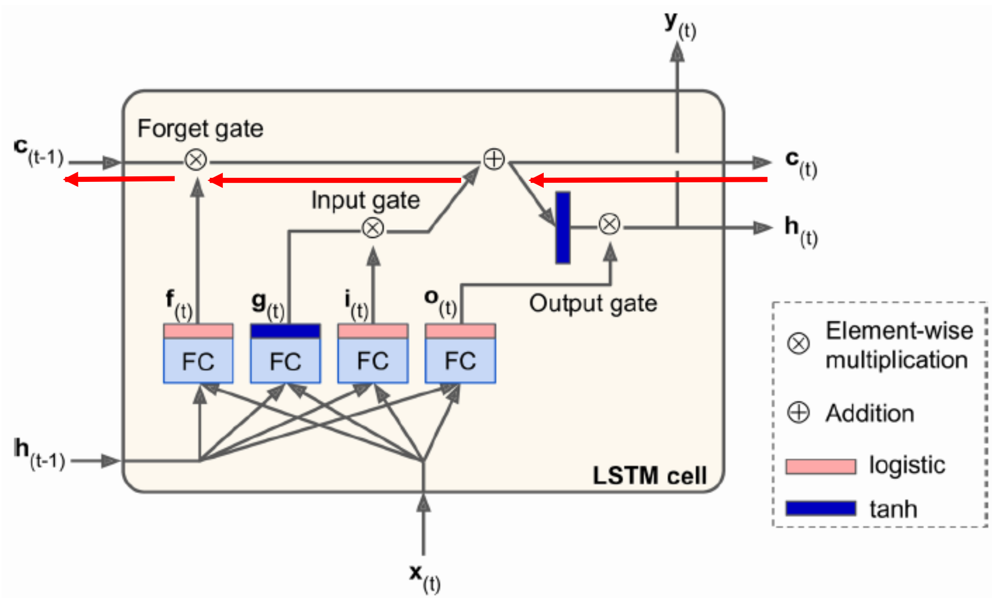


- : combining information from and
- : forget gate
- : input gate
- : output gate

- Long term state :
  1. Go through a *forget gate* to drop some memories
  2. Add some memory through addition operation, which adds memories selected by an *input gate*

- Short term state :
  1. Long term state is squashed by a *tanh* function
  2. Further filtered via an *output gate*

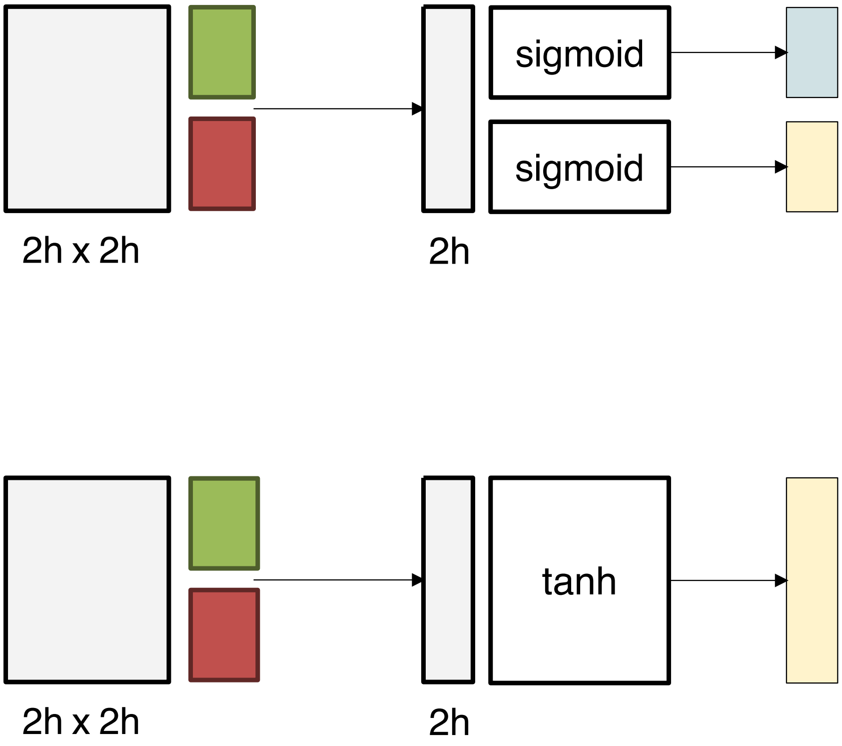




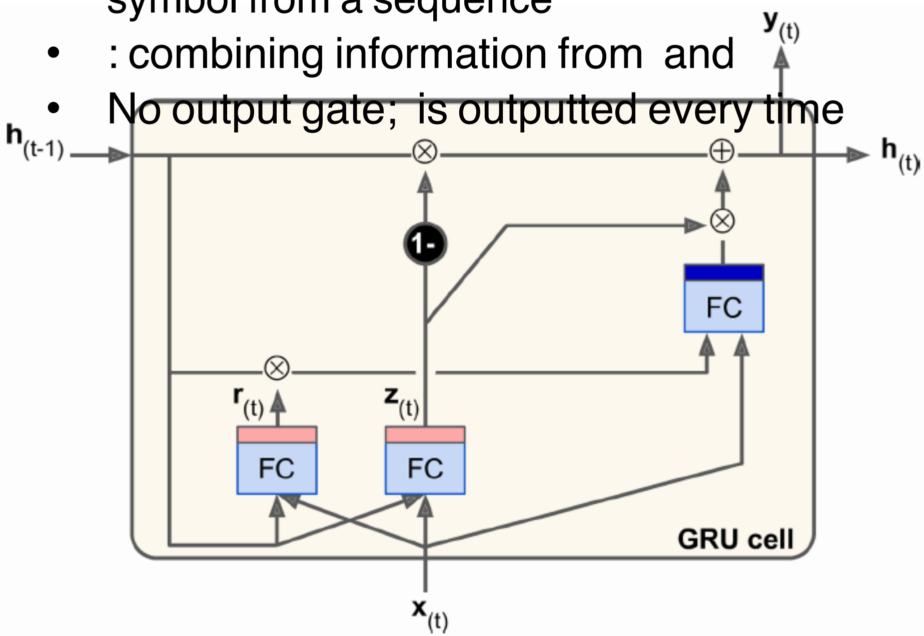
Backprop from to only elementwise multiplication by , which depends partially on and . So backprop contains no direct matrix multiplication by

□ **Uninterrupted gradient flow:**

Gated recurrent unit (GRU)



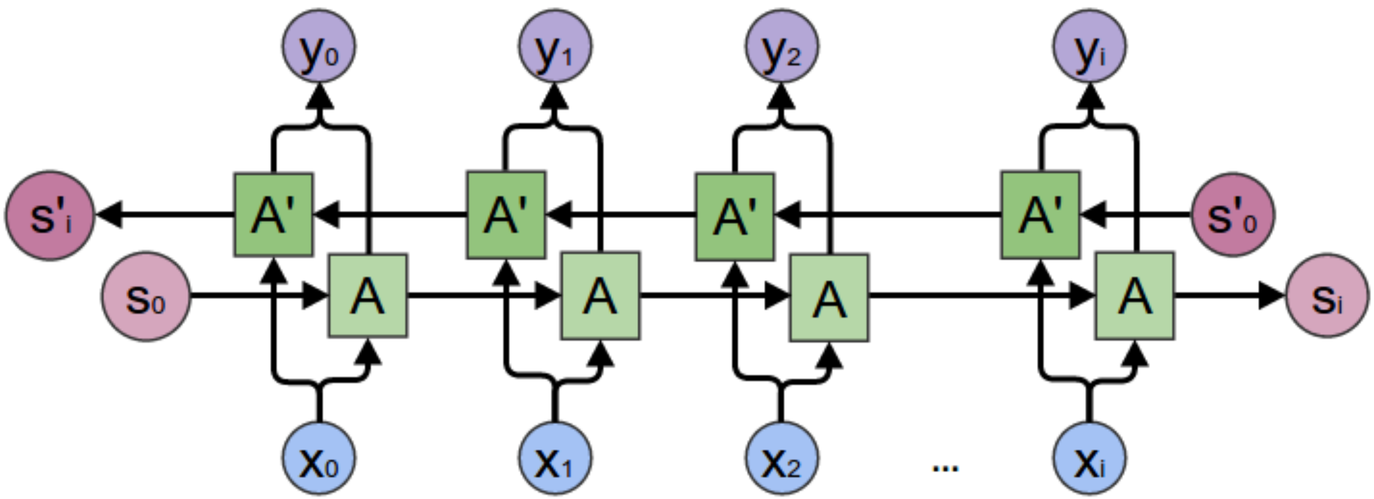
- Merge cell state and hidden state into one state
- : forget-and-input (update) gate
- : reset gate: when off (i.e., equals zero), as if it is reading the first symbol from a sequence
- : combining information from and
- No output gate; is outputted every time



Bidirectional RNN

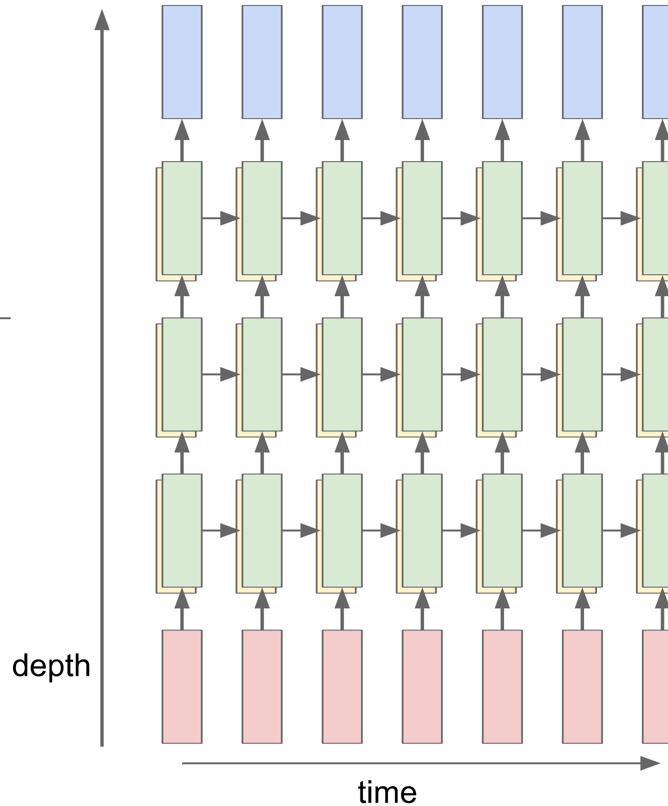
- For certain tasks, seeing “future” inputs is reasonable
  - E.g., when translating from English to Chinese, we don’t translate word-by-word; we read the entire sentence (or paragraph) and translate
    - “Previous” words and “future words” together influence the selection of the current word
- Bidirectional RNN
  - Run a RNN from left to right
  - Run another RNN from right to left
  - Combine (e.g., concatenate) their outputs at each time step

Bidirectional RNN

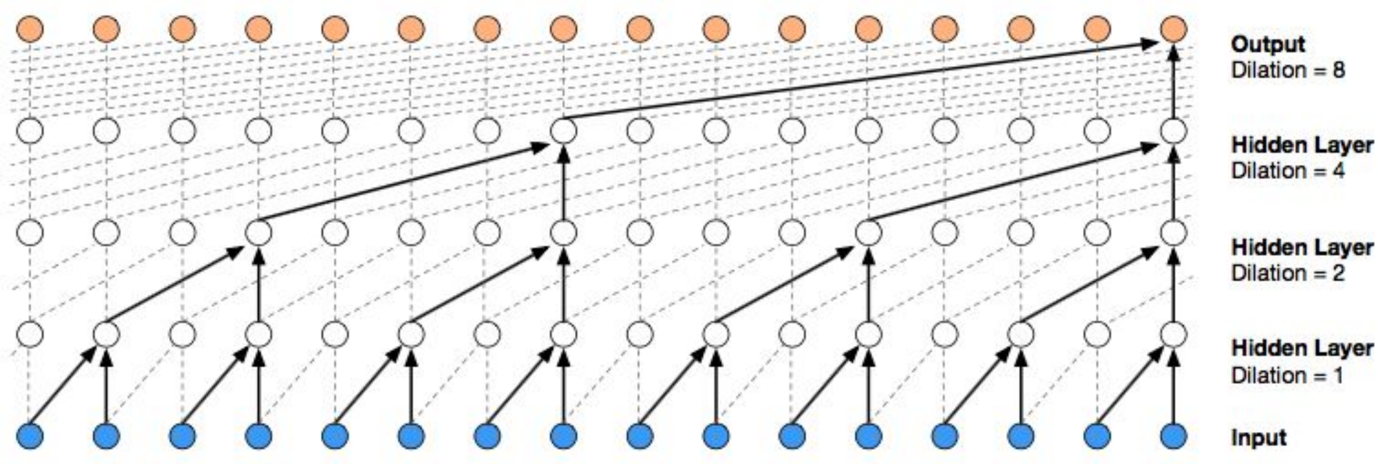


Multilayer RNNs

LSTM:



WaveNet



Output
Dilation = 8
Hidden Layer
Dilation = 4
Hidden Layer
Dilation = 2
Hidden Layer
Dilation = 1
Input

WaveNet structure

- Stacked 1D convolution layers
- Doubling the dilation rate (how spread apart each neuron's inputs are) at each layer
  - 1st convolutional layer gets 2 time steps
  - 2nd convolutional layer gets 4 time steps
  - 3rd convolutional layer gets 8 time steps
- Overall, lower layers learn short-term patterns; higher layers learn long-term patterns
- Efficiently process large sequences

Beam search

- Greedily output the most likely word at every time step may not be an optimal result
- Beam search keeps track of a short list of the most promising candidates (is called the “beam width”)
  - Extend each candidate by one token and keep the most promising candidate for each extension
  - Keep only the most promising candidates out of the candidates
  - Repeat the above two steps

Example: French to English translation

- Jane visite l'Afrique en septembre
- Steps
  1. Out of 10,000 possible English words as the beginning, select the top 3
     - [In, Jane, September]
  2. Extend each candidate's top 3 candidates
     - [In [September, the, this]]
     - [Jane [is, wants, will]]
     - [September [is, seems, will]]
  3. Select top 3 among them
     - [In September], [Jane is], [Jane will]
  4. …

The example is taken from https://medium.com/@dhartidhami/beam-search-in-seq2seq-model-7606d55b21a5

Summary

- RNNs allow a lot of flexibility in architecture design
- Vanilla RNNs are simple but don’t work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- 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.