Recurrent Neural Networks

Jim Glass / MIT 6.806-6.864 / Spring 2021

1

Today's RNN Story

- RNN language models
- Training RNNs
- Gated RNNs
  - Gated recurrent units (GRUs)
  - Long short-term memories (LSTMs)
- Other architectures
  - Deep RNNs
  - Bidirectional RNNs

2

Language Modeling (Again)

- n-gram LMs represent history with the previous $n-1$ words
  - Number of n-gram parameters increases exponentially with n

$$p(w_t | w_{t-(n-1)}, \dots, w_{t-1})$$

- An alternative model represents word history with a latent variable

$$p(w_t | w_1, \dots, w_{t-1}) \approx p(w_t | h_t)$$

- Recurrent neural networks use hidden states to capture history
  - Latent variable h_t is computed based on input x_t and h_{t-1}

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

3

A Feed-Forward LM

- Softmax output

$$\boldsymbol{y} = softmax(\mathbf{U}\boldsymbol{h} + \boldsymbol{c}) \quad \mathbf{U} \in \mathbb{R}^{V \times h} \quad \boldsymbol{c} \in \mathbb{R}^{V}$$

- Non-linear hidden layer

$$\boldsymbol{h} = f(\mathbf{W}\boldsymbol{v} + \boldsymbol{b}) \quad \mathbf{W} \in \mathbb{R}^{h \times 4d} \quad \boldsymbol{b} \in \mathbb{R}^{\mathrm{h}}$$

- Truncated context vector

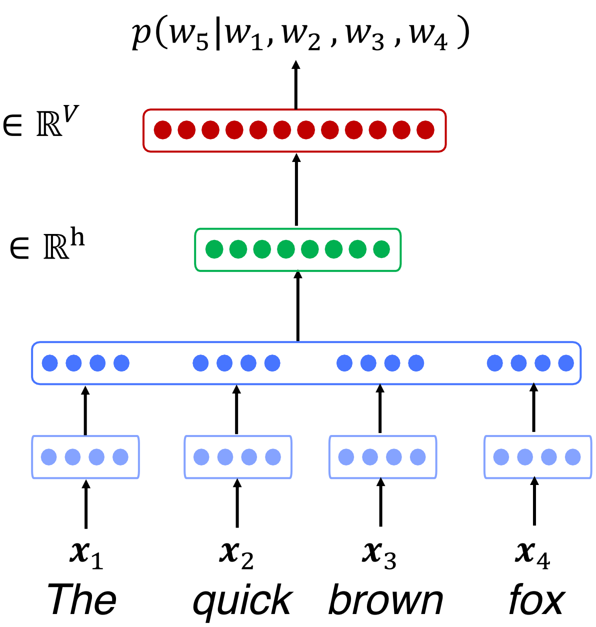
$$\boldsymbol{v} = [v_1; v_2; v_3; v_4] \quad \boldsymbol{v} \in \mathbb{R}^{4d}$$

- Word embedding vectors

$$v_i = \mathbf{E}\boldsymbol{x}_i \quad v_i \in \mathbb{R}^{d} \quad \mathbf{E} \in \mathbb{R}^{d \times V}$$

- One-hot input vectors

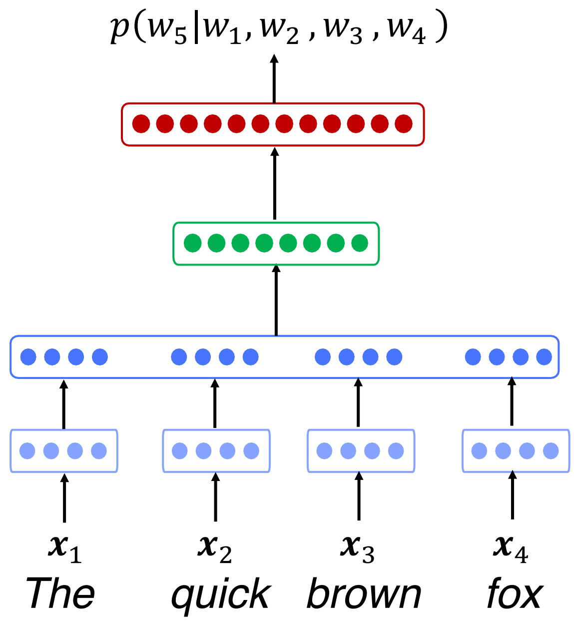
$$\boldsymbol{x}_i \in \{\boldsymbol{e}_i : 1 \le i \le V\}$$



4

Feed-Forward LMs

- Advantages
  - No n-gram sparsity issues
  - Better memory usage
- Disadvantages
  - Fixed context window
  - Limits ability to capture history
  - No parameter sharing in $\mathbf{W}$
- Need LM model to handle variable length sequential inputs



5

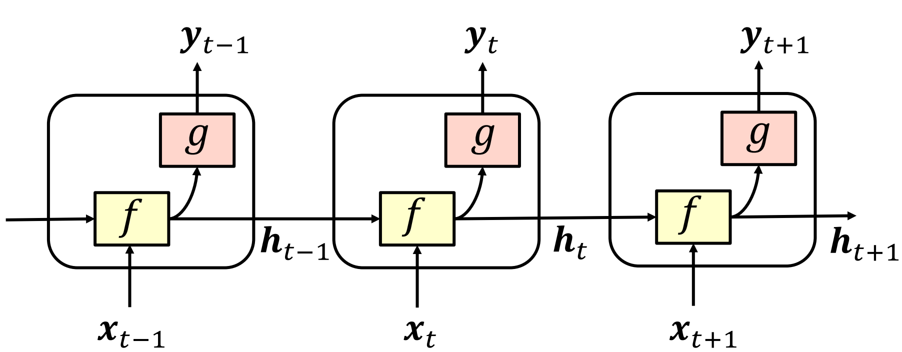
Recurrent Neural Networks (RNNs)

- RNNs accept variable length input sequences $\boldsymbol{x}_t$
- Use a hidden layer that incorporates current input and prior state

$$\boldsymbol{h}_t = f(\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \mathbf{W}_{xh}\boldsymbol{x}_t + \mathbf{b}_h)$$

- Optional outputs can be produced at every step

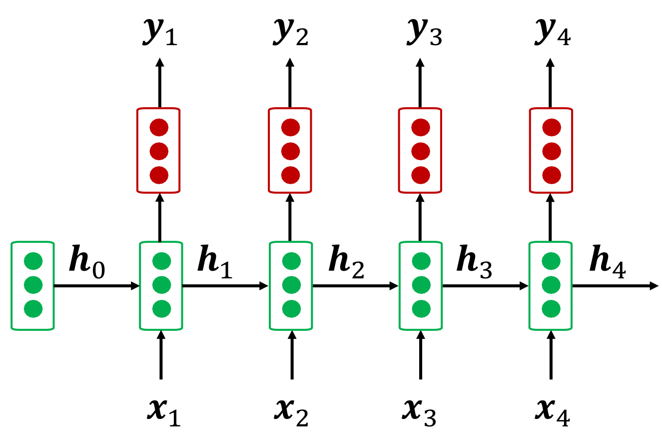
$$\boldsymbol{y}_t = g(\mathbf{W}_{hy}\boldsymbol{h}_t + \mathbf{b}_y)$$



6

Recurrent Neural Networks (RNNs)

- Uses a hidden layer that incorporates current input and prior hidden state
- Accepts variable length sequences
- Optional outputs can be produced at every step
- Inherent parameter sharing



7

An RNN LM

- Softmax output computes $p(w_t|w_1, \ldots, w_{t-1})$

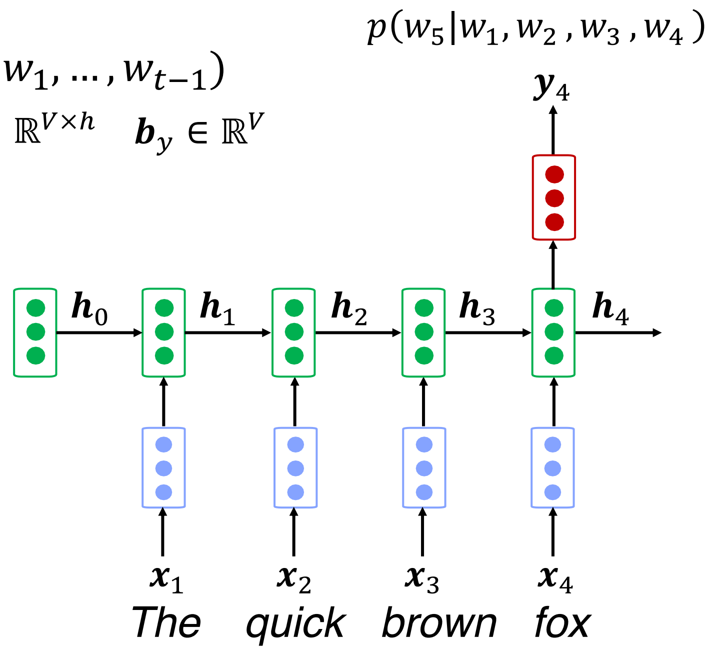
  $\boldsymbol{y}_t = softmax(\boldsymbol{W}_{hy}\boldsymbol{h}_t + \boldsymbol{b}_y)$ $\quad \boldsymbol{W}_{hy} \in \mathbb{R}^{V\times h}$ $\quad \boldsymbol{b}_y \in \mathbb{R}^{V}$
- Non-linear hidden layer

  $\boldsymbol{h}_t = f(\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh}\boldsymbol{v}_t + \boldsymbol{b}_h)$

  $\boldsymbol{W}_{hh} \in \mathbb{R}^{h\times h}$ $\quad \boldsymbol{W}_{xh} \in \mathbb{R}^{h\times d}$ $\quad \boldsymbol{b}_h \in \mathbb{R}^{\mathrm{h}}$
- Word embedding vectors

  $\boldsymbol{v}_t = \mathbf{E}\boldsymbol{x}_t$ $\quad \boldsymbol{v}_t \in \mathbb{R}^{d}$ $\quad \boldsymbol{E} \in \mathbb{R}^{d\times V}$
- One-hot input vectors

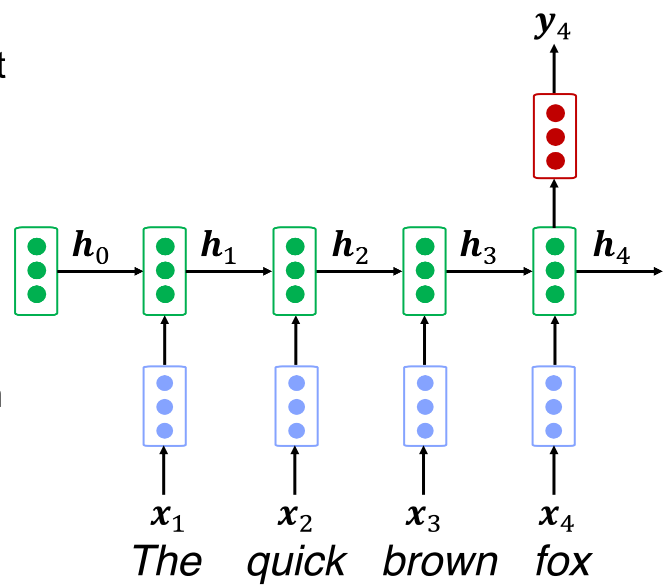
  $\boldsymbol{x}_t \in \{\boldsymbol{e}_i : 1 \le i \le V\}$



8

RNN LMs

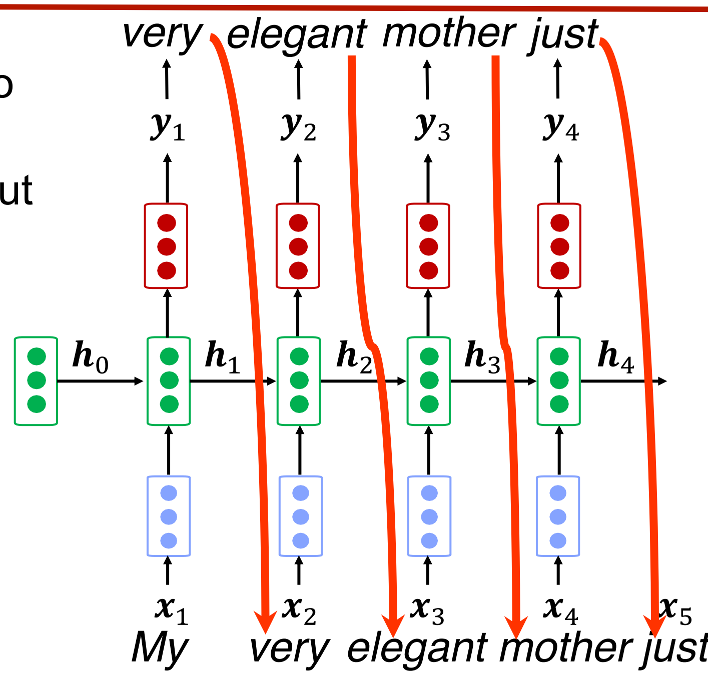
- Advantages
  - Can process variable length input
  - No truncated history
  - Model size context-independent
  - Sharing among weights
- Disadvantages
  - Recurrent computation is slow
  - Limitations on how far back it can incorporate context



9

RNN-based Language Generation

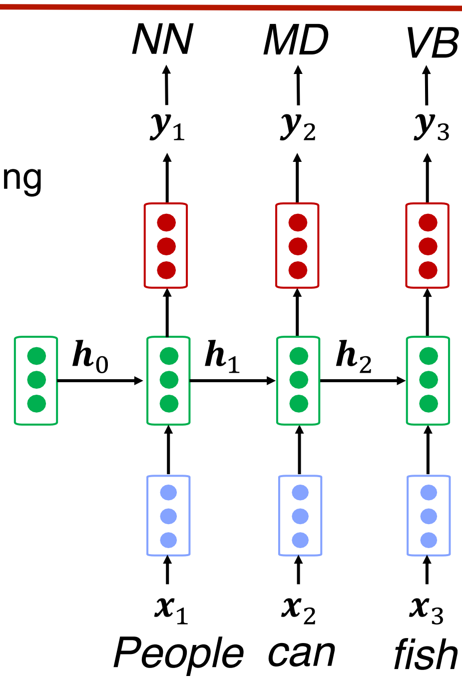
- An RNN LM can be used to *generate* text by sampling
- Sampled output is next input



10

RNN-based Sequence Labeling

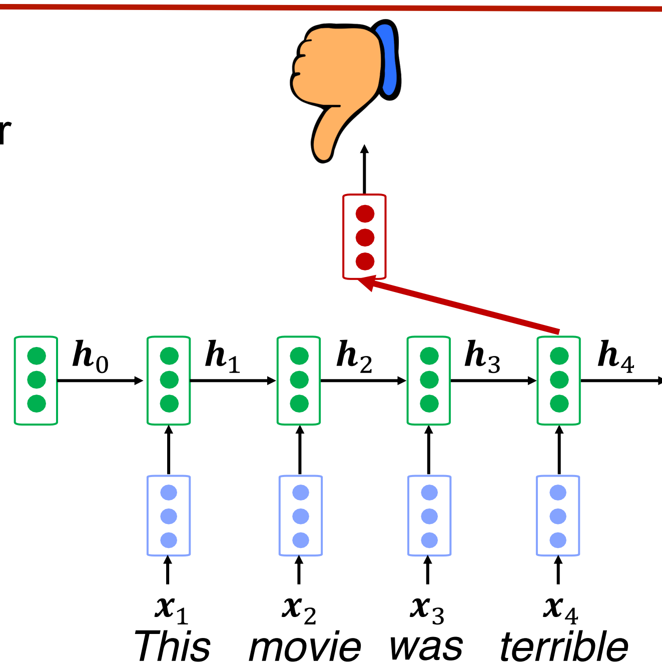
- An RNN can be trained to output tags for each word
  - Part-of-speech (POS) tagging
  - Named Entity Recognition



11

RNN-based Sentence Classification

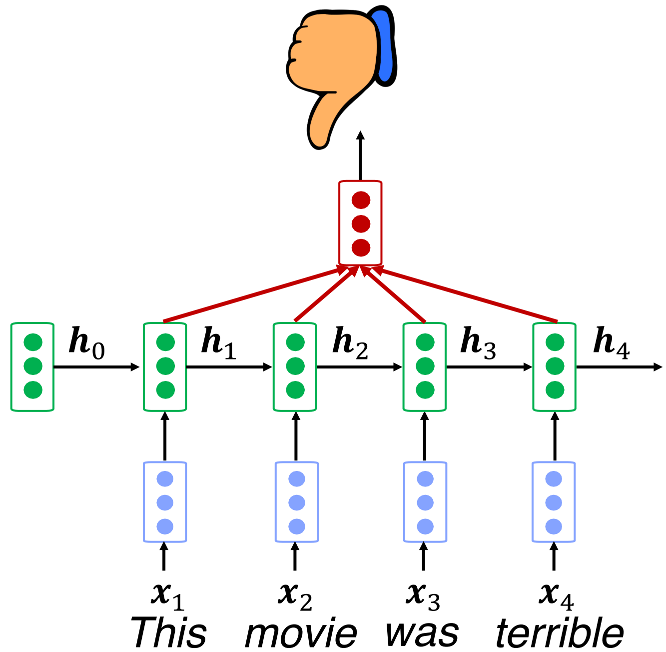
- Represent sentence as vector
- Use final hidden state as a representation of sentence
- Feed into penultimate layer for sentence classification



12

An Alternate RNN-based Sentence Classifier

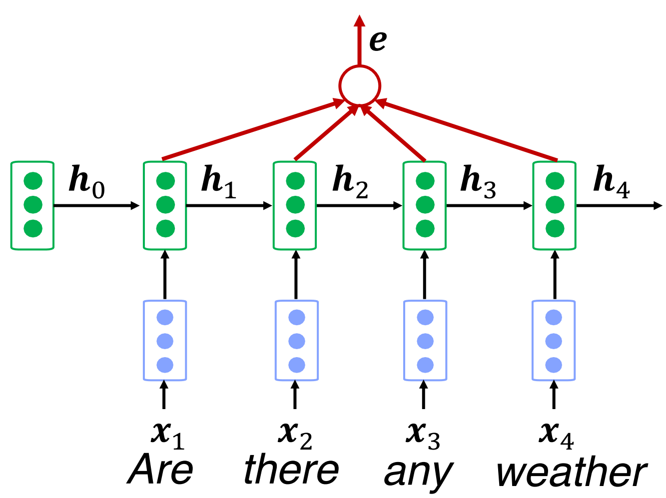
- Represent sentence as vector
- Take element-wise max or mean of all hidden states
  - A simple form of *attention*
- Feed into penultimate layer for sentence classification



13

RNN-based Encoder

- Represent sentence as vector
- Take element-wise max or mean of all hidden states
  - A simple form of *attention*
- Feed into subsequent layers for downstream processing
  - Question-answering
  - Machine translation
  - Etc.



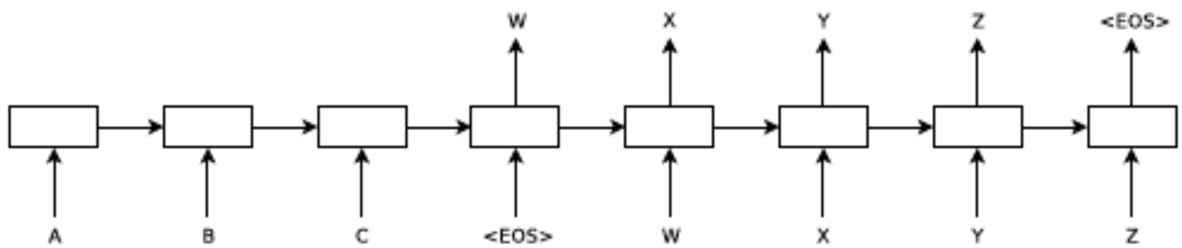
14

Sequence to Sequence Learning with Neural Networks

Ilya Sutskever Google ilyasu@google.com

Oriol Vinyals Google vinyals@google.com

Quoc V. Le Google qvl@google.com



NeurIPS 2014

15

Today's RNN Story

- RNN language models
- **Training RNNs**
- Gated RNNs
  - Gated recurrent units (GRUs)
  - Long short-term memories (LSTMs)
- Other architectures
  - Deep RNNs
  - Bidirectional RNNs

16

Training RNNs

- RNN parameters are learned on a training corpus
- Overall loss is average step-by-step cross-entropy loss

$$L(\boldsymbol{\theta}) = \frac{1}{T}\sum_{t=1}^{T} L_t$$

- For LMs, L_t equivalent to negative log likelihood of next true word
  - Computed by taking dot-product with next one-hot vector

$$L_t = -\log p(w_t | w_1, \cdots, w_{t-1}) = -\log(\boldsymbol{y}_i \cdot \boldsymbol{x}_{t+1})$$

- Parameters optimized via back-propagation and SGD

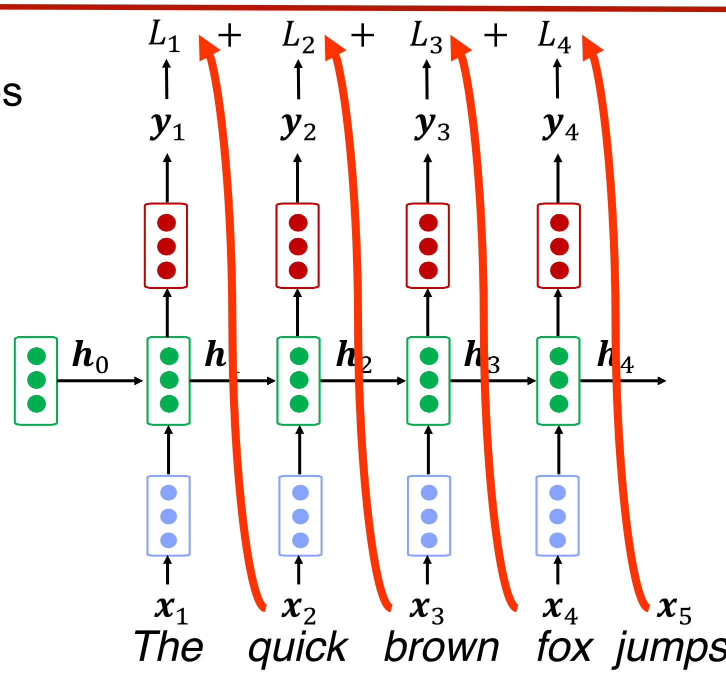
17

RNN LM Training

- The total loss is the sum of all word-by-word losses

$$L(\boldsymbol{\theta}) = \frac{1}{T}\sum_{t=1}^{T} L_t$$

- For SGD, losses are typically accumulated in batches of sentences



18

IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 5, NO. 2, MARCH 1994

Learning Long-Term Dependencies with Gradient Descent is Difficult

Yoshua Bengio, Patrice Simard, and Paolo Frasconi, *Student Member, IEEE*

On the difficulty of training recurrent neural networks

Razvan Pascanu PASCANUR@IRO.UMONTREAL.CA
Université de Montréal, 2920, chemin de la Tour, Montréal, Québec, Canada, H3T 1J8

Tomas Mikolov T.MIKOLOV@GMAIL.COM
Speech@FIT, Brno University of Technology, Brno, Czech Republic

Yoshua Bengio YOSHUA.BENGIO@UMONTREAL.CA
Université de Montréal, 2920, chemin de la Tour, Montréal, Québec, Canada, H3T 1J8

ICML 2013

19

Training Neural Networks via Backpropagation

- Gradients propagated backwards through network to minimize loss
  - Gradients are accumulated for each parameter in a training batch
- Numerical stability issues for many layered networks
  - Exploding gradients, vanishing gradients
  - Initialization, non-linearity choices affect results
  - Techniques developed for clipping exploding gradients
  - Residual and highway connections etc. help with vanishing gradients

20

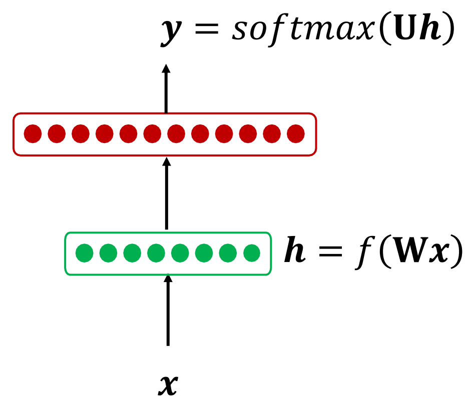
Computational Graphs

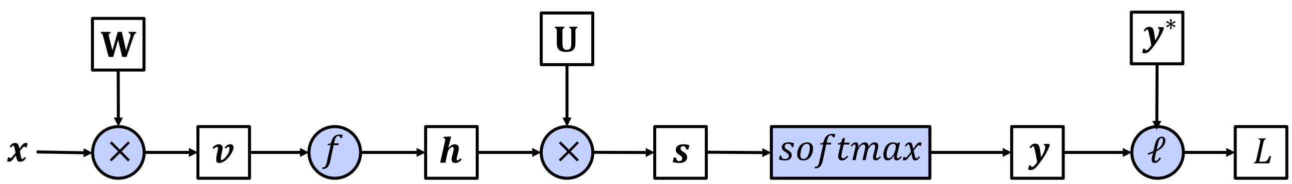
- Neural networks can be represented as a computational graph, e.g.,

$$\boldsymbol{v} = \boldsymbol{W}\boldsymbol{x} \qquad \boldsymbol{h} = f(\boldsymbol{v}) \qquad \boldsymbol{s} = \boldsymbol{U}\boldsymbol{h}$$

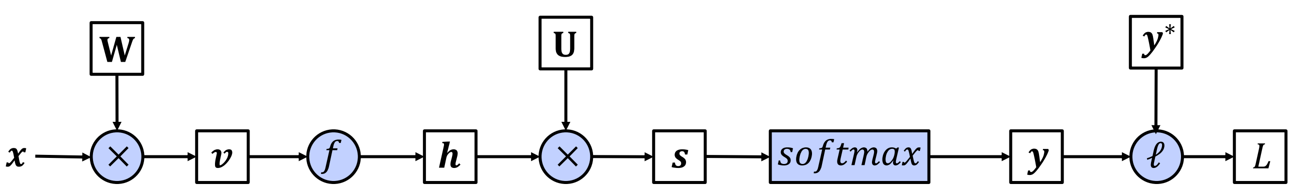
$$\boldsymbol{y} = softmax\ \boldsymbol{s} \qquad L = -\log(\boldsymbol{y} \cdot \boldsymbol{y}^*)$$

- For training, each data point takes a forward and backward pass through graph
  - Gradients are accumulated for each parameter





Computational Graph MLP Example



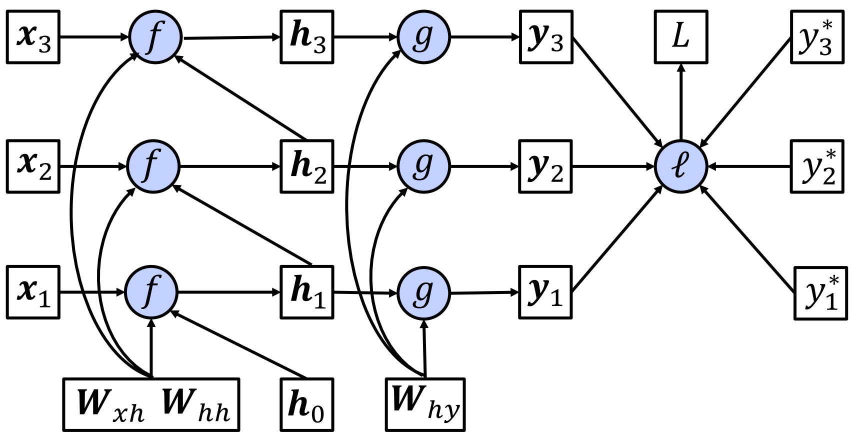
$$\boldsymbol{v} = \boldsymbol{W}\boldsymbol{x} \qquad \boldsymbol{h} = f(\boldsymbol{v}) \qquad \boldsymbol{s} = \boldsymbol{U}\boldsymbol{h} \qquad \boldsymbol{y} = softmax\ \boldsymbol{s} \qquad L = -\log(\boldsymbol{y} \cdot \boldsymbol{y}^*)$$

$$\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{W}} \qquad \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{v}} \qquad \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{h}} \quad \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{U}} \qquad \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{s}} \qquad \frac{\partial L}{\partial \boldsymbol{y}}$$

$$\frac{\partial L}{\partial \boldsymbol{W}} = \frac{\partial L}{\partial \boldsymbol{y}} \times \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{s}} \times \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{h}} \times \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{v}} \times \frac{\partial \boldsymbol{v}}{\partial \boldsymbol{W}} \qquad \frac{\partial L}{\partial \boldsymbol{U}} = \frac{\partial L}{\partial \boldsymbol{y}} \times \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{s}} \times \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{U}}$$

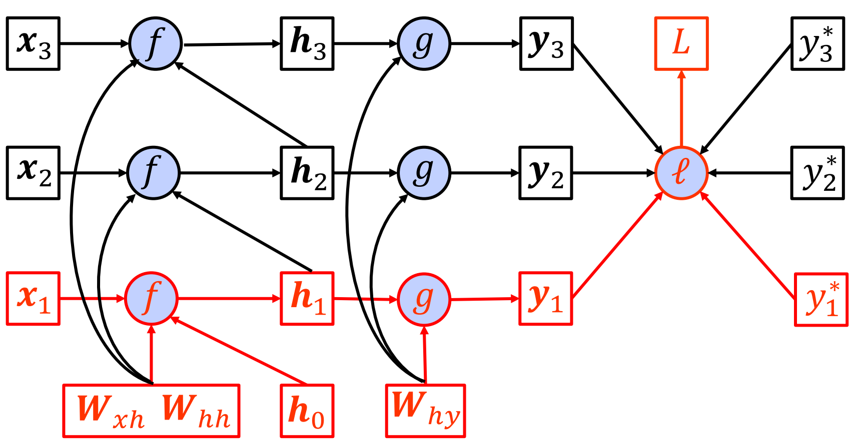
- Gradients are accumulated for each parameter over batch

An RNN Computational Graph



23

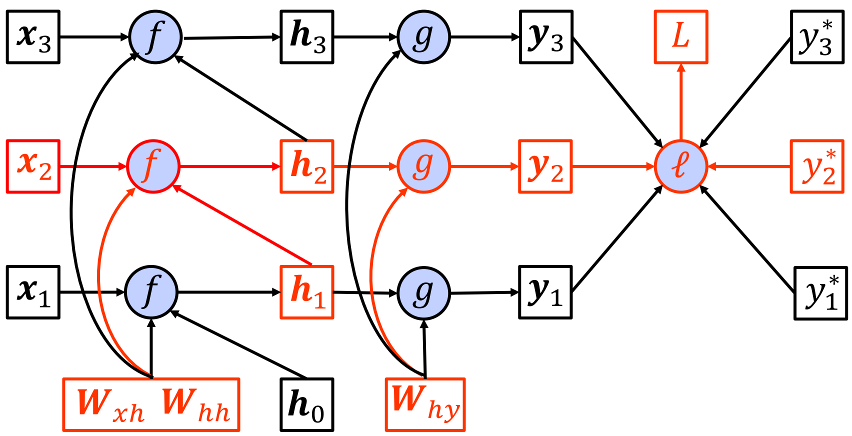
RNN Forward Pass Step 1



- Backpropagation for step 1 touches the same parameters

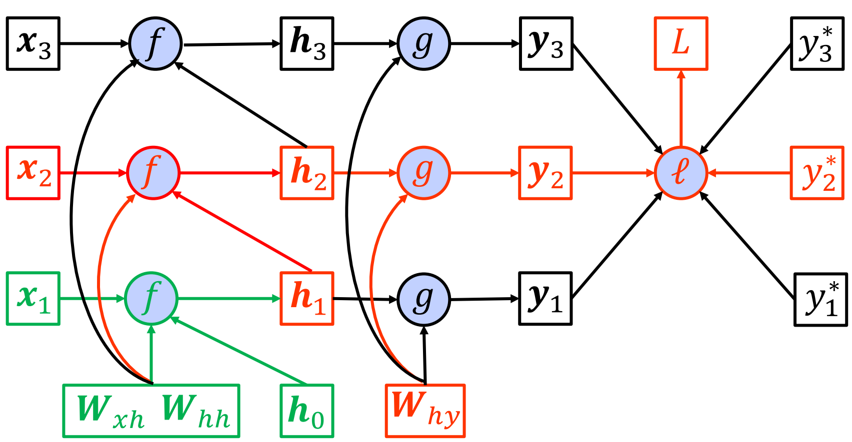
24

RNN Forward Pass Step 2



25

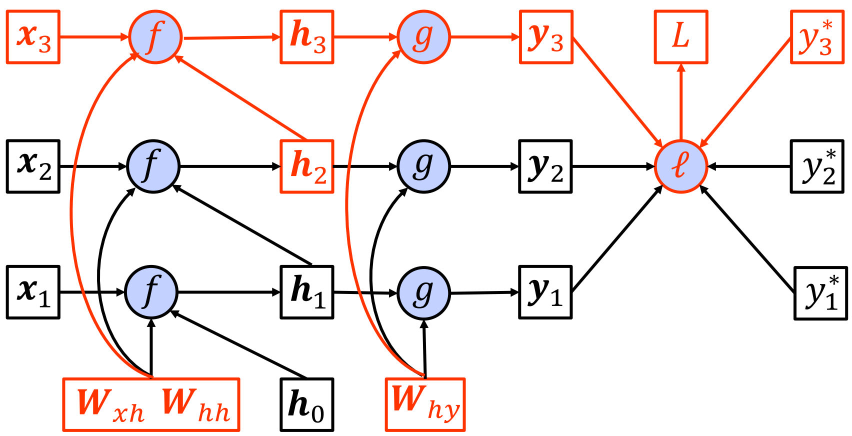
RNN Backpropagation Pass Step 2



- Backpropagation for step 2 must also consider gradients for $\boldsymbol{h}_1$

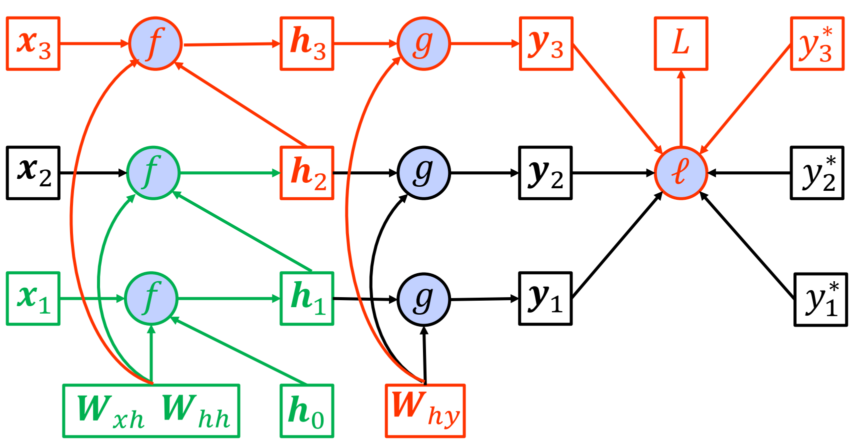
26

RNN Forward Pass Step 3



27

RNN Backpropagation Pass Step 3



- Backpropagation for step 3 must also consider partials for $\boldsymbol{h}_2$ and $\boldsymbol{h}_1$

28

Backpropagation Through Time

- SGD for RNNs must consider the impact of past inputs and states
  - This process is known as *Backpropagation Through Time* (*BPTT*)
- The gradients for longer time spans are exponential, e.g.,

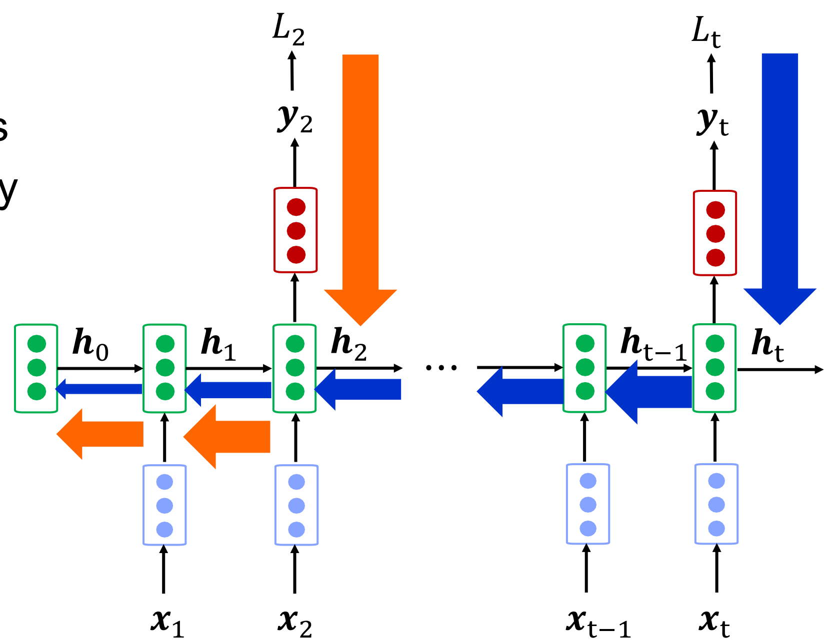
$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{W}_{hh}} = \sum_{i=1}^{t} \left(\boldsymbol{W}_{hh}^T\right)^{t-i} \boldsymbol{h}_i \qquad \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{W}_{xh}} = \sum_{i=1}^{t} \left(\boldsymbol{W}_{hh}^T\right)^{t-i} \boldsymbol{v}_i$$

  - Potential for exploding gradients or vanishing gradients
- Since BPTT is computationally intensive for long sequences, sometimes truncated BPTT is used to save computation

29

Impact of Vanishing Gradients

- Long distance gradients are weaker and have less impact than local gradients
- Model parameters primarily learn local dependencies
- This motivated the search for RNNs that could better model long distance dependencies by some internal memory state



30

Today's RNN Story

- RNN language models
- Training RNNs
- Gated RNNs
  - Gated recurrent units (GRUs)
  - Long short-term memories (LSTMs)
- Other architectures
  - Deep RNNs
  - Bidirectional RNNs

31

Gated RNNs

- Conventional RNNs can be challenging to train
  - Long products of matrices lead to vanishing or divergent gradients
  - Effect of BPTT focuses attention on recent history
- Desiderata:
  - A *memory* mechanism to store important information over long distances
  - A *forgetting* mechanism to erase unimportant information from the model
  - A mechanism to *reset* the internal state representation
- A number of alternative RNNs attempt to address these issues
  - Gated RNNs are far more commonly used for sequence labeling
  - One of the earliest is *Long Short-Term Memory* (LSTM) RNNs
  - *Gated Recurrent Unit* (GRU) RNNs are more streamlined and faster

32

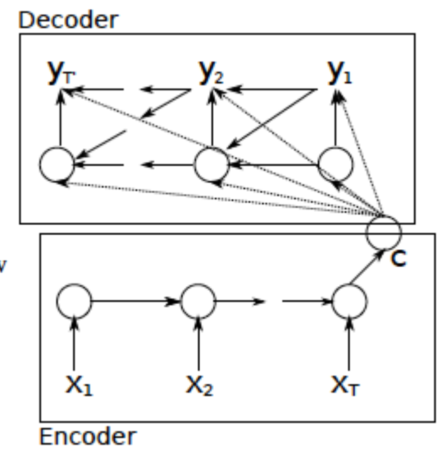
Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation

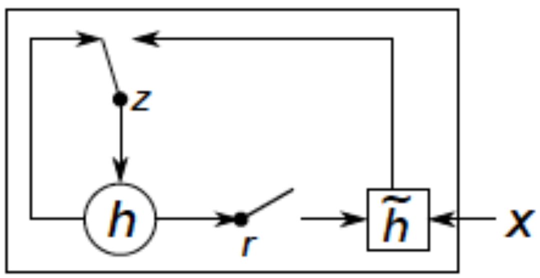
Kyunghyun Cho
Bart van Merriënboer Caglar Gulcehre
Université de Montréal
firstname.lastname@umontreal.ca

Dzmitry Bahdanau
Jacobs University, Germany
d.bahdanau@jacobs-university.de

Fethi Bougares Holger Schwenk
Université du Maine, France
firstname.lastname@lium.univ-lemans.fr

Yoshua Bengio
Université de Montréal, CIFAR Senior Fellow
find.me@on.the.web

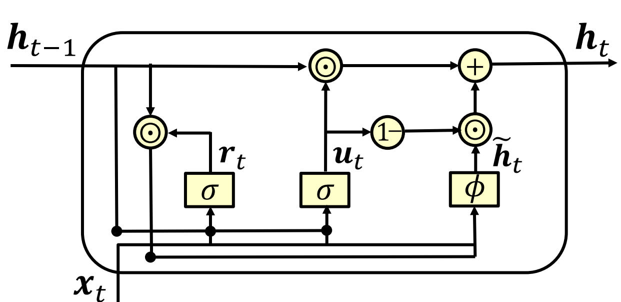




Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1724–1734,
October 25-29, 2014, Doha, Qatar. ©2014 Association for Computational Linguistics

Gated Recurrent Units (GRUs)

- Key distinction between regular RNNs and GRUs is gating
- Dedicated mechanisms for updating and resetting hidden state
  - *Reset* gate controls how much prior state information to remember
  - *Update* gate controls how much new state retains of old state
- Gating mechanisms are a function of current input and prior state



⊙ Element-wise multiplication

GRU Reset Gates

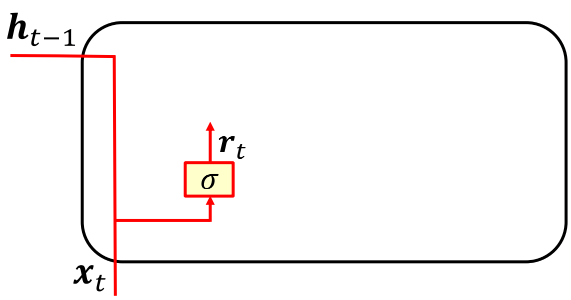
- Reset gate is used to reduce influence of $\boldsymbol{h}_{t-1}$ (i.e., reset past)

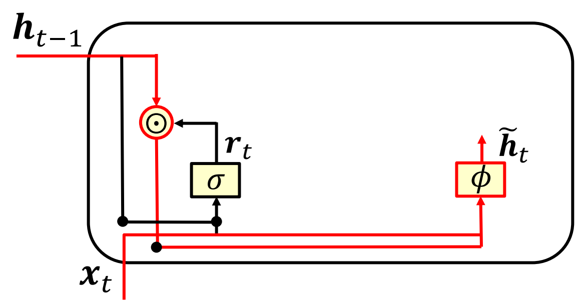
$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}_{hr}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xr}\boldsymbol{x}_t + \boldsymbol{b}_r) \qquad \boldsymbol{r}_t \in \mathbb{R}^h$$

- Produces a candidate hidden state by de-weighting prior state

$$\widetilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W}_{hh}(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}) + \mathbf{W}_{xh}\boldsymbol{x}_t + \mathbf{b}_h)$$

- For $\boldsymbol{r}_t \sim \mathbf{1}$ GRU behaves as RNN; for $\boldsymbol{r}_t \sim \mathbf{0}$ GRU behaves as MLP





35

GRU Update Gates

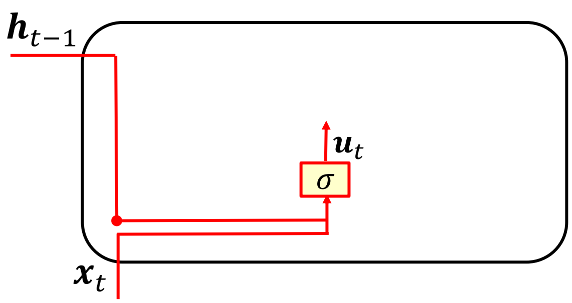
- Determines extent new state is old state vs new candidate state

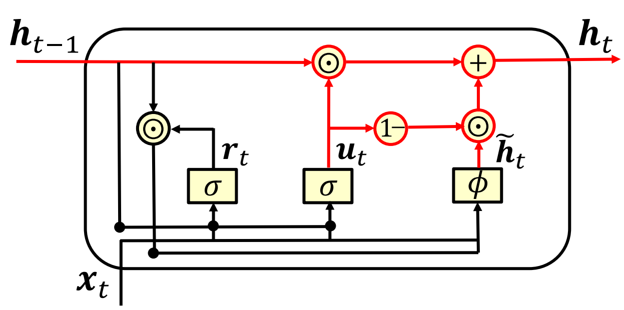
$$\boldsymbol{u}_t = \sigma(\boldsymbol{W}_{xu}\boldsymbol{x}_t + \boldsymbol{W}_{hu}\boldsymbol{h}_{t-1} + \boldsymbol{b}_u) \qquad \boldsymbol{u}_t \in \mathbb{R}^h$$

- Update gate $\boldsymbol{u}_t$ applied in convex combination with $\boldsymbol{h}_{t-1}$ and $\widetilde{\boldsymbol{h}}_t$

$$\boldsymbol{h}_t = \boldsymbol{u}_t \odot \boldsymbol{h}_{t-1} + (\mathbf{1} - \boldsymbol{u}_t) \odot \widetilde{\boldsymbol{h}}_t$$

- When $\boldsymbol{u}_t \sim \mathbf{1}$ we essentially skip time step t and remember prior state





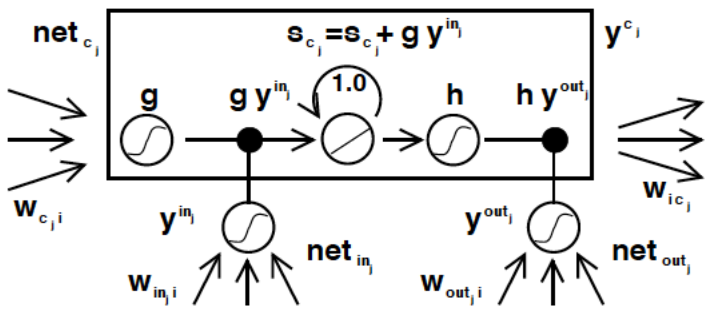
36

LONG SHORT-TERM MEMORY

NEURAL COMPUTATION 9(8):1735–1780, 1997

Sepp Hochreiter
Fakultät für Informatik
Technische Universität München
80290 München, Germany
hochreit@informatik.tu-muenchen.de
http://www7.informatik.tu-muenchen.de/~hochreit

Jürgen Schmidhuber
IDSIA
Corso Elvezia 36
6900 Lugano, Switzerland
juergen@idsia.ch
http://www.idsia.ch/~juergen



37

Long Short-Term Memories (LSTMs)

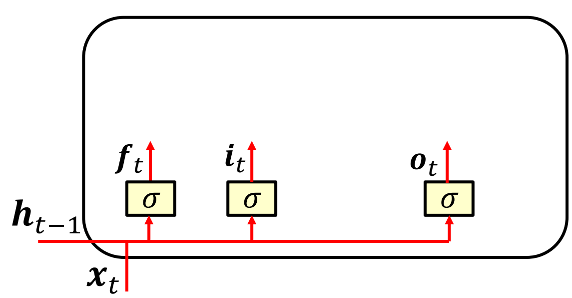
- LSTMs pre-date GRUs, but are slightly more complex (3 gates)
- Inspired by logic gates to control a *memory cell*
  - An *output* gate reads out entries from the cell
  - An input gate is used to read data into a cell
  - A forget gate is used to reset cell contents
- All three gates are a function of current input and prior state

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_{hi}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xi}\boldsymbol{x}_t + +\boldsymbol{b}_i)$$

$$\boldsymbol{f}_t = \sigma\left(\boldsymbol{W}_{hf}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xf}\boldsymbol{x}_t + +\boldsymbol{b}_f\right)$$

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_{ho}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xo}\boldsymbol{x}_t + +\boldsymbol{b}_o)$$

$$\boldsymbol{i}_t, \boldsymbol{f}_t, \boldsymbol{o}_t \in \mathbb{R}^h$$



38

LSTM Memory Cell

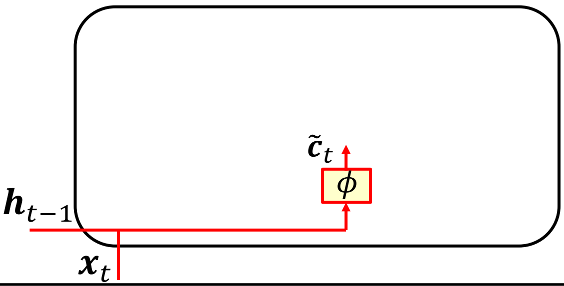
- A *candidate* memory cell is based on a regular RNN hidden state

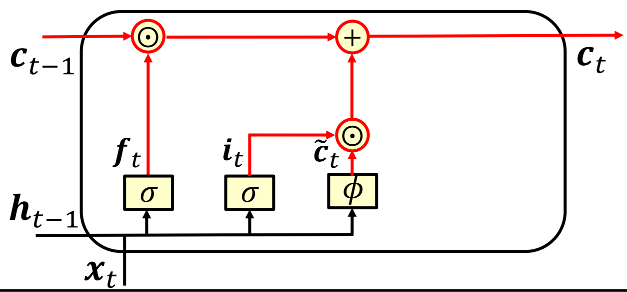
$$\tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \mathbf{W}_{xh}\boldsymbol{x}_t + \mathbf{b}_h)$$

- The input and forget gates are used to create the new memory cell

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t$$

  - $\boldsymbol{i}_t$ controls how much new information to take into account
  - $\boldsymbol{f}_t$ controls how much old information to retain
  - If $\boldsymbol{f}_t \sim \mathbf{1}$ and $\boldsymbol{i}_t \sim \mathbf{0}$ then old information will be retained



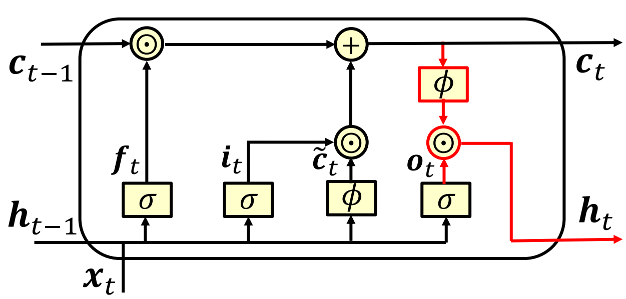


LSTM Hidden State

- The new hidden state is partially read from the new memory cell

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)$$

  - Amount retained in $\boldsymbol{h}_t$ is controlled by output gate
  - $\tanh$ ensures $\boldsymbol{h}_t$ spans interval (-1,1)
- If $\boldsymbol{o}_t \sim \mathbf{1}$ pass all information through to prediction for next time step



GRUs vs LSTMs

- Both gated RNNs are much better able to maintain information over many timesteps compared to a vanilla RNN
- Both models have been very effective on many NLP tasks
  - LSTMs attained state-of-the-art results in the 2013-2015 time frame
  - GRUs are newer models, but have also achieved good results
- The LSTM memory cell is not bounded like the hidden state, and has demonstrated an excellent ability to count etc.
- The GRU has fewer parameters and is faster than the LSTM

41

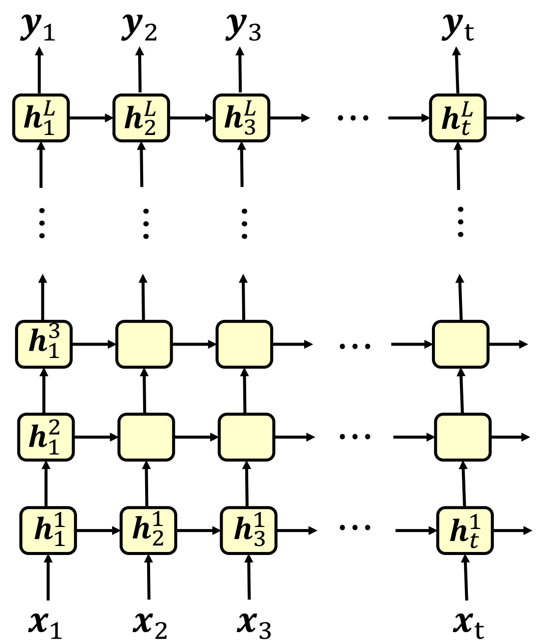
Today's RNN Story

- RNN language models
- Training RNNs
- Gated RNNs
  - Gated recurrent units (GRUs)
  - Long short-term memories (LSTMs)
- Other architectures
  - Deep RNNs
  - Bidirectional RNNs

42

Deep RNNs

- The conventional DNN is a feed-forward network
- RNNs can be stacked in multiple layers on top of each other
  - The hidden state $\boldsymbol{h}_t$ from lower RNN becomes input to the upper RNN
  - The topmost layer is responsible for generating any outputs $\boldsymbol{y}_t$
- Deep RNN layers can potentially focus on different information
  - Much more flexibility than HMMs



43

Bidirectional RNNs

- Bidirectional RNNs consist of two RNNs
  - $\overrightarrow{RNN}$ runs in a forward direction starting from the beginning

$$\overrightarrow{\boldsymbol{h}}_t = \phi\left(\overrightarrow{\boldsymbol{W}}_{hh}\overrightarrow{\boldsymbol{h}}_{t-1} + \overrightarrow{\mathbf{W}}_{xh}\boldsymbol{x}_t + \vec{\mathbf{b}}_h\right)$$

  - $\overleftarrow{RNN}$ runs in a backward direction starting from the end

$$\overleftarrow{\boldsymbol{h}}_t = \phi\left(\overleftarrow{\boldsymbol{W}}_{hh}\overleftarrow{\boldsymbol{h}}_{t+1} + \overleftarrow{\mathbf{W}}_{xh}\boldsymbol{x}_t + \overleftarrow{\mathbf{b}}_h\right)$$

- At time step t, the hidden state is the concatenation of $\overrightarrow{\boldsymbol{h}}_t$ and $\overleftarrow{\boldsymbol{h}}_t$

$$\boldsymbol{h}_t = \left[\overrightarrow{\boldsymbol{h}}_t; \overleftarrow{\boldsymbol{h}}_t\right] \qquad \boldsymbol{h}_t \in \mathbb{R}^{2h}$$

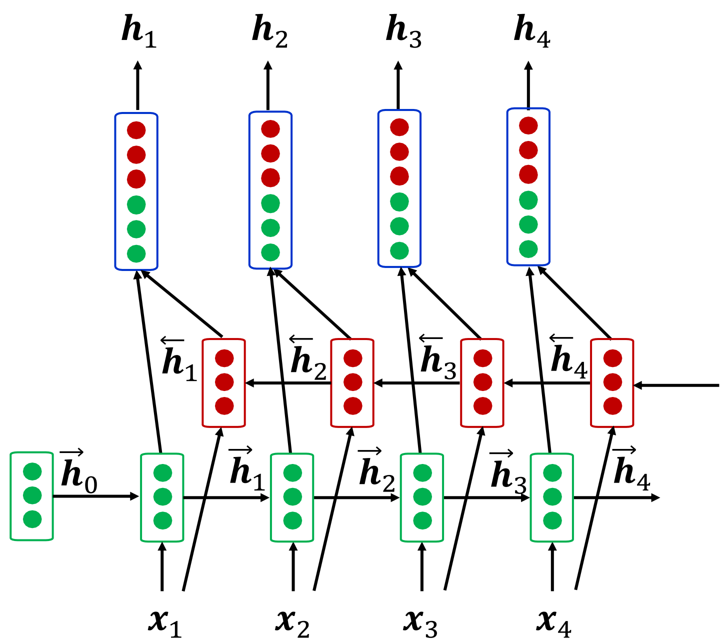
- The bidirectional RNN output is computed like vanilla RNNs

$$\boldsymbol{y}_t = \phi\left(\mathbf{W}_{hq}\boldsymbol{h}_t + \mathbf{b}_q\right) \qquad \mathbf{W}_{hq} \in \mathbb{R}^{q\times 2h}$$

44

Bidirectional RNNs

- Provides fuller context for each input token
- Bidirectional RNNs have achieved very good performance
- Requires access to entire input label sequence



45

References

- Readings:
  - Jurafsky & Martin, “Speech and Language Processing,” 2020 (RNNs 9.2-9.3)

46