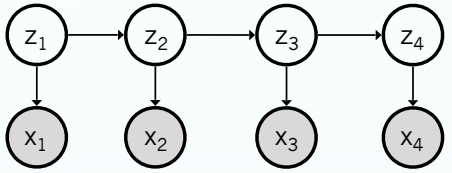
Deep Networks for Sequences: Recurrent Neural Networks

Mark Dredze

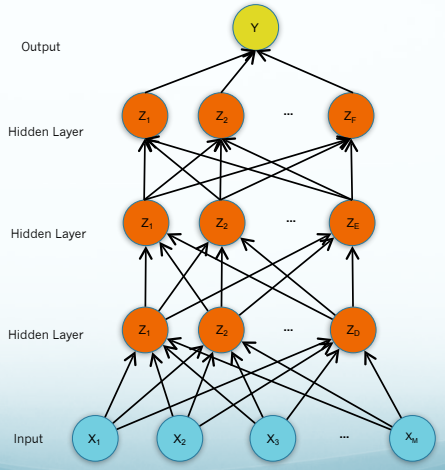
Machine Learning
CS 601.475

Slides from Chris Dyer, given at LxMLS 2016

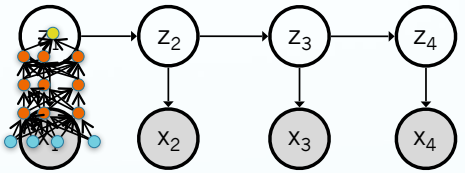
Sequential Events



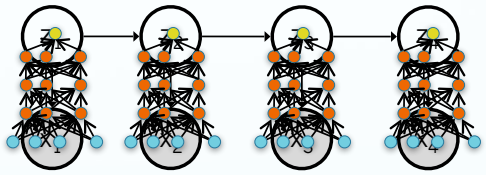
Deep Networks



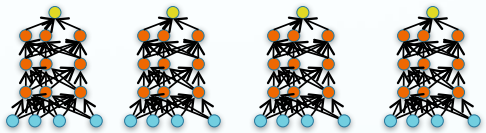
Deep Networks



Deep Networks



Deep Networks



- This assumes independence between the states
  - Akin to a 0th order markov model
- Goal: develop a neural network that connects each state in the sequence

Outline

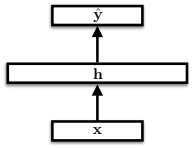
- Recurrent neural networks
  - Application: language models
- Learning challenges and solutions
  - Vanishing gradients
  - Long short-term memories
  - Gated recurrent units
- Bidirectional RNNs
  - Application: better word representations
- Sequence to Sequence transduction with RNNs
  - Applications: machine translation & image caption generation

Recurrent Neural Networks

Feed-forward NN

$$\mathbf{h} = g(\mathbf{V}\mathbf{x} + \mathbf{c})$$

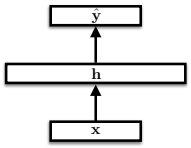
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}$$



Recurrent Neural Networks

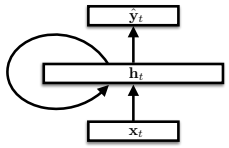
Feed-forward NN

$$\mathbf{h} = g(\mathbf{V}\mathbf{x} + \mathbf{c})$$
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}$$



Recurrent NN

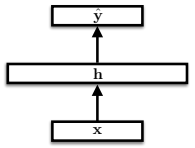
$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Recurrent Neural Networks

Feed-forward NN

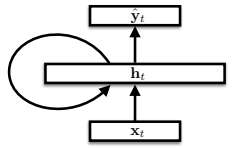
$$\mathbf{h} = g(\mathbf{V}\mathbf{x} + \mathbf{c})$$
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}$$



Recurrent NN

~~$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$~~

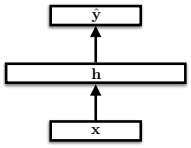
$$\mathbf{h}_t = g(\mathbf{V}[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{c})$$
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Recurrent Neural Networks

Feed-forward NN

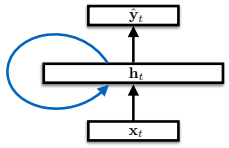
$$\mathbf{h} = g(\mathbf{V}\mathbf{x} + \mathbf{c})$$
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}$$



Recurrent NN

~~$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$~~

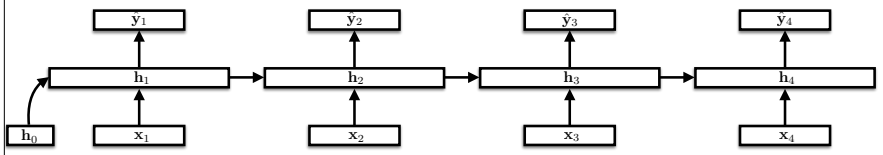
$$\mathbf{h}_t = g(\mathbf{V}[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{c})$$
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Recurrent Neural Networks

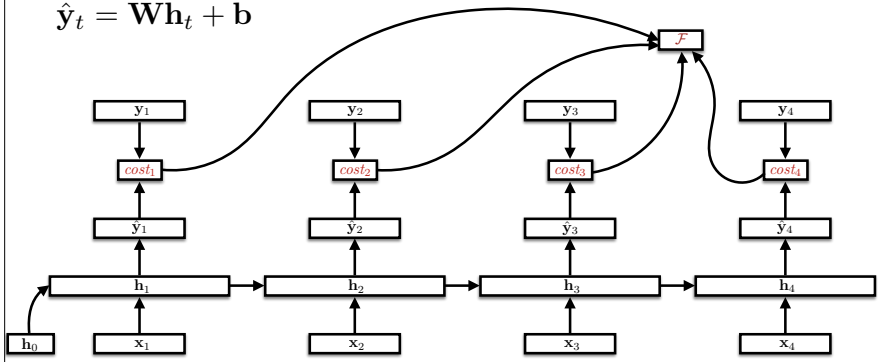
$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$

How do we train the RNN's parameters?

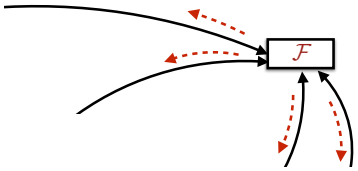


Recurrent Neural Networks

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



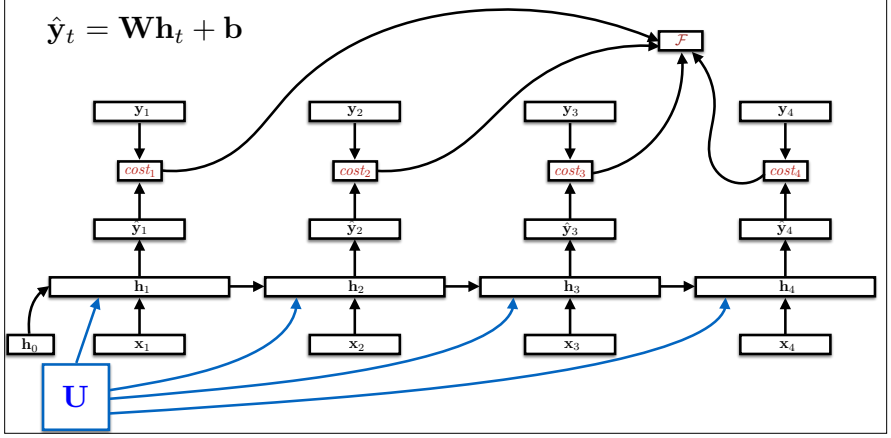
Recurrent Neural Networks



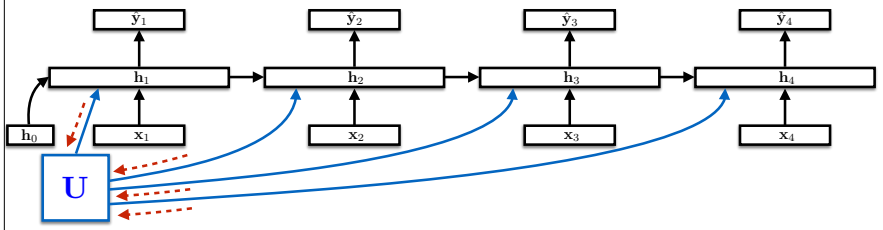
- The unrolled graph is a well-formed (DAG) computation graph—we can run backprop
  - Parameters are tied across time, derivatives are aggregated across all time steps
  - This is historically called “backpropagation through time” (BPTT)

Parameter Tying

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Parameter Tying



$$\frac{\partial \mathcal{F}}{\partial \mathbf{U}} = \sum_{t=1}^{4} \frac{\partial \mathbf{h}_t}{\partial \mathbf{U}} \frac{\partial \mathcal{F}}{\partial \mathbf{h}_t}$$

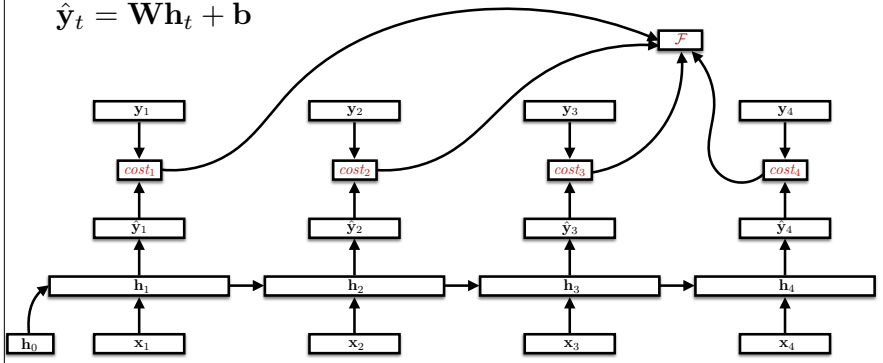
Parameter tying also came up when learning the transition matrices for HMMs!

Parameter Tying

- Why do we want to tie parameters?
  - Reduce the number of parameters to be learned
  - Deal with arbitrarily long sequences
- What if we always have short sequences?
  - Maybe you might untie parameters, then. But you wouldn't have an RNN anymore!

What else can we do?

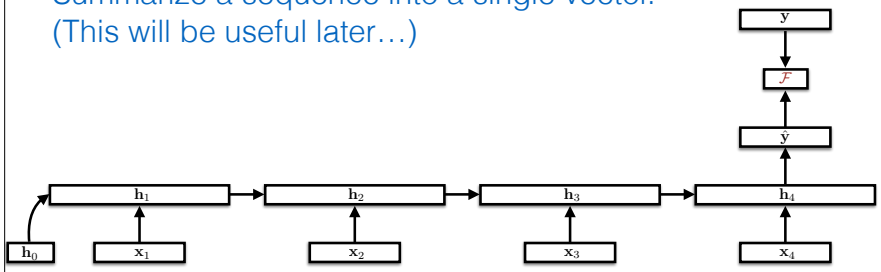
$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



"Read and summarize"

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$

Summarize a sequence into a single vector.
(This will be useful later…)



View 2: Recursive Definition

- Recall how to construct a list recursively:
  base case
  [] is a list (the empty list)

View 2: Recursive Definition

- Recall how to construct a list recursively:
  base case
    [] is a list (the empty list)

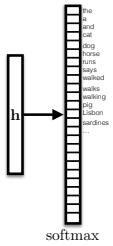
  induction
    [**t** | h] where **t** is a list and h is an atom is a list

View 2: Recursive Definition

- Recall how to construct a list recursively:
  base case
    [] is a list (the empty list)

  induction
    [**t** | h] where **t** is a list and h is an atom is a list
- RNNs define functions that compute representations recursively according to this definition of a list.
  - Define (learn) a representation of the base case
  - Learn a representation of the inductive step
- **Anything you can construct recursively, you can obtain an "embedding" of with neural networks using this general strategy**

Example: Language Model



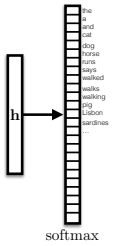
$$\mathbf{u} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

$$p_i = \frac{\exp u_i}{\sum_j \exp u_j}$$

$$\mathbf{h} \in \mathbb{R}^d$$

$$|V| = 100,000$$

Example: Language Model



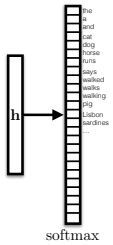
$$\mathbf{u} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

$$p_i = \frac{\exp u_i}{\sum_j \exp u_j}$$

$$\mathbf{h} \in \mathbb{R}^d$$

$$|V| = 100,000$$

Example: Language Model



$$\mathbf{u} = \mathbf{W}\mathbf{h} + \mathbf{b} \qquad \mathbf{h} \in \mathbb{R}^d$$

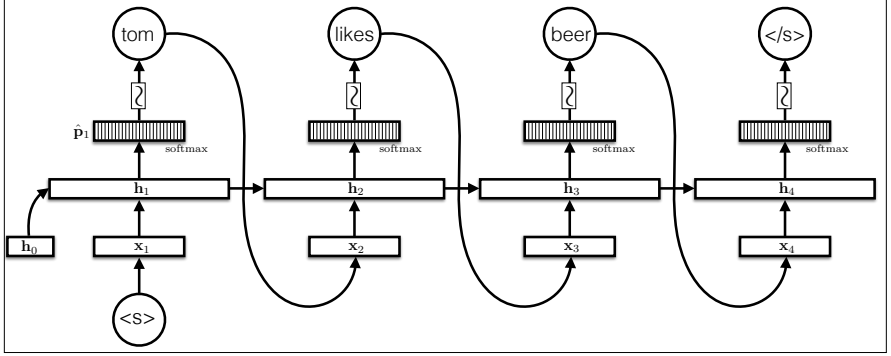
$$p_i = \frac{\exp u_i}{\sum_j \exp u_j} \qquad |V| = 100,000$$

$$p(\boldsymbol{e}) = p(e_1) \times$$
$$p(e_2 \mid e_1) \times$$
$$p(e_3 \mid e_1, e_2) \times$$
$$p(e_4 \mid e_1, e_2, e_3) \times$$
$$\ldots$$

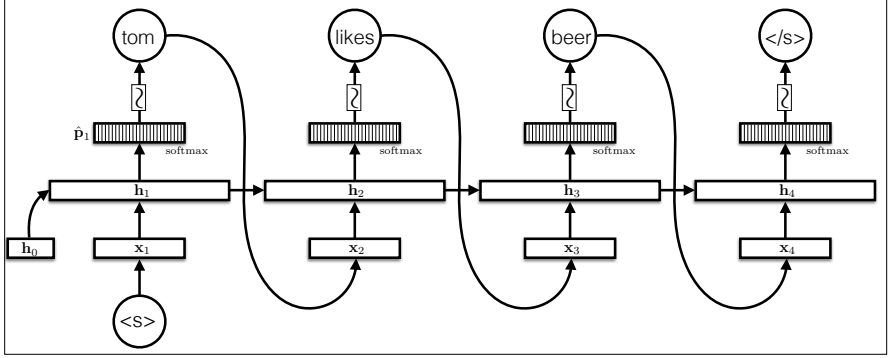
histories are sequences of words…

Example: Language Model

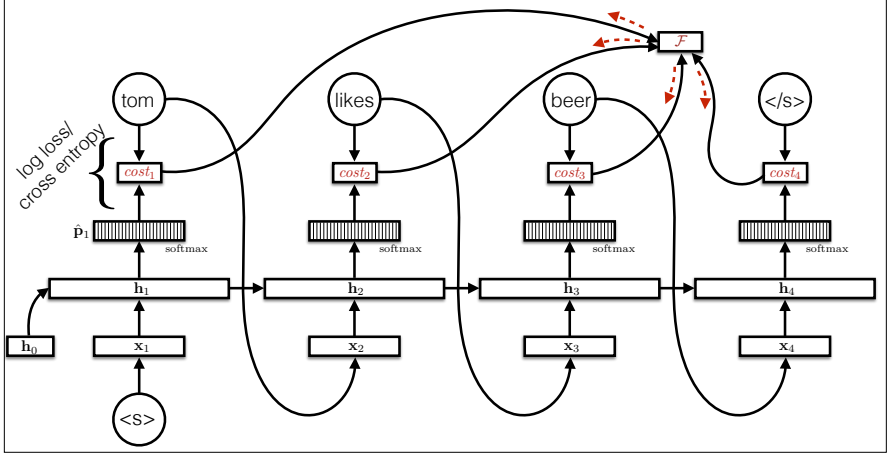
$$p(tom \mid \langle \mathtt{s} \rangle) \times p(likes \mid \langle \mathtt{s} \rangle, tom)$$
$$\times p(beer \mid \langle \mathtt{s} \rangle, tom, likes)$$
$$\times p(\langle / \mathtt{s} \rangle \mid \langle \mathtt{s} \rangle, tom, likes, beer)$$



Language Model Training



Language Model Training



RNN Language Models

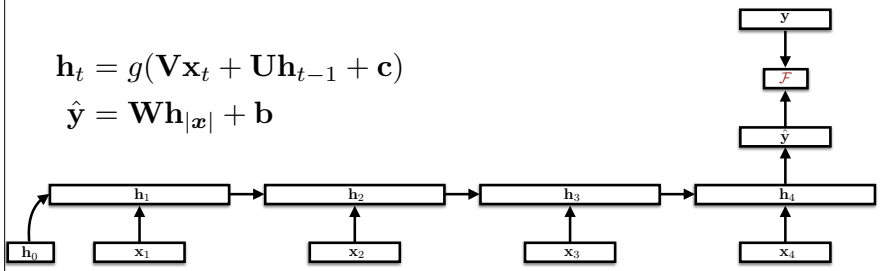
- Unlike Markov (*n*-gram) models, RNNs never forget
  - However we will see they might have trouble learning to use their memories (more soon…)

- Algorithms
  - Sample a sequence from the probability distribution defined by the RNN
  - Train the RNN to minimize cross entropy (aka MLE)

Learning Challenges

Training Challenges

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$

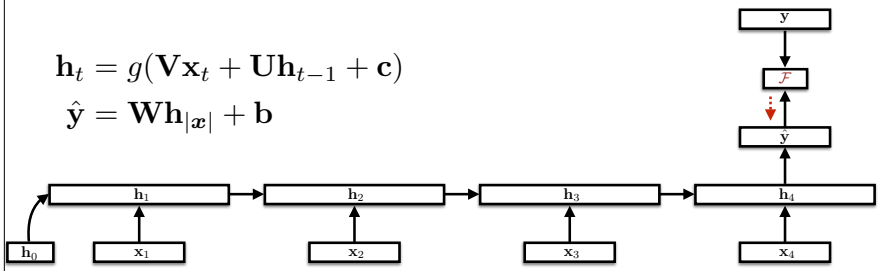


What happens to gradients as you go back in time?

$$\frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

Training Challenges

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$

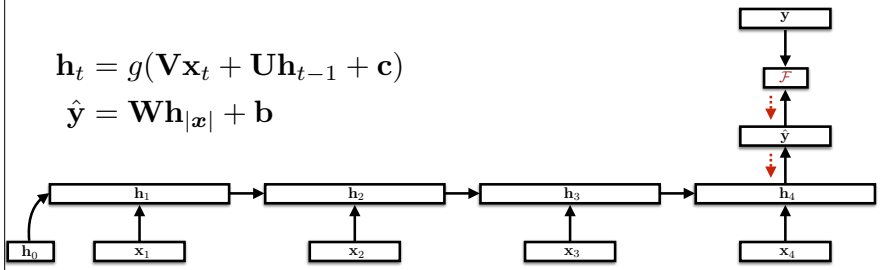


What happens to gradients as you go back in time?

$$\frac{\partial \mathcal{F}}{\partial \hat{\mathbf{y}}} \frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

Training Challenges

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$

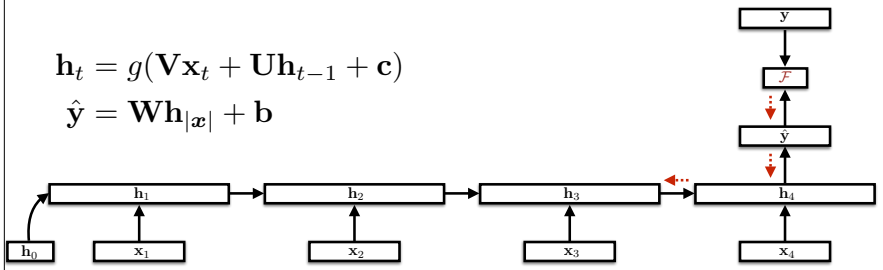


What happens to gradients as you go back in time?

$$\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}_4}\frac{\partial \mathcal{F}}{\partial \hat{\mathbf{y}}}\frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

Training Challenges

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$

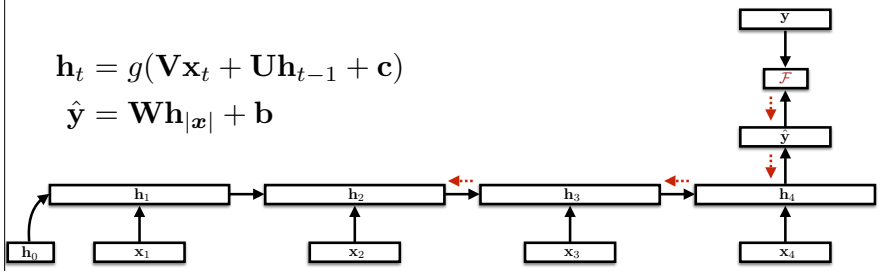


What happens to gradients as you go back in time?

$$\frac{\partial \mathbf{h}_4}{\partial \mathbf{h}_3}\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}_4}\frac{\partial \mathcal{F}}{\partial \hat{\mathbf{y}}}\frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

Training Challenges

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$



What happens to gradients as you go back in time?

$$\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2}\frac{\partial \mathbf{h}_4}{\partial \mathbf{h}_3}\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}_4}\frac{\partial \mathcal{F}}{\partial \hat{\mathbf{y}}}\frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

Vanishing Gradients

- This means that long-range dependencies are difficult to learn (although in theory they are learnable)
- Solutions
  - Better optimizers (second order methods, approximate second order methods)
  - Normalization to keep the gradient norms stable across time
  - Clever initialization so that you at least start with good spectra (e.g., start with random orthonormal matrices)
  - **Alternative parameterizations: LSTMs and GRUs**

Alternative RNNs

- Long short-term memories (LSTMs; Hochreiter and Schmidthuber, 1997)
- Gated recurrent units (GRUs; Cho et al., 2014)
- Intuition instead of **multiplying** across time (which leads to exponential growth), we want the error to be **constant**.

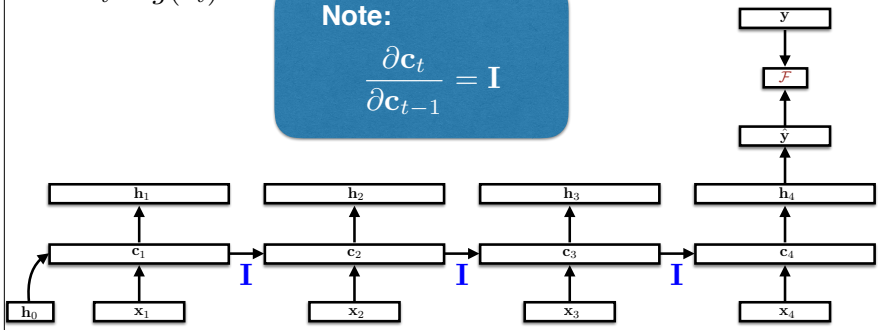
Memory cells

$$\mathbf{c}_t = \mathbf{c}_{t-1} + f(\mathbf{x}_t)$$
$$\mathbf{h}_t = g(\mathbf{c}_t)$$

$$f(\mathbf{v}) = \tanh(\mathbf{W}\mathbf{v} + \mathbf{b})$$

Note:

$$\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} = \mathbf{I}$$

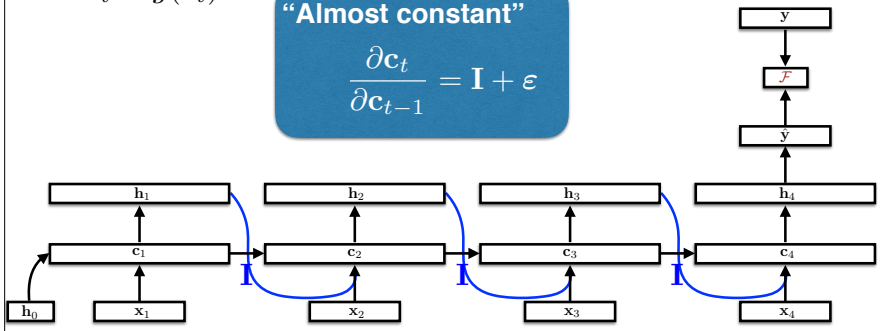


Memory cells

$$\mathbf{c}_t = \mathbf{c}_{t-1} + f([\mathbf{x}_t; \mathbf{h}_{t-1}])$$
$$\mathbf{h}_t = g(\mathbf{c}_t)$$

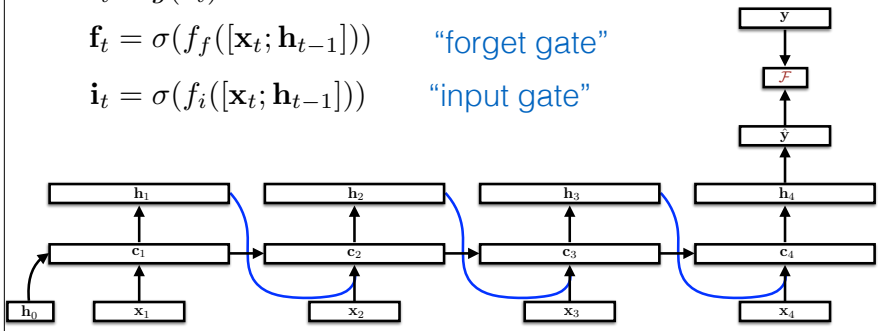
"Almost constant"

$$\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} = \mathbf{I} + \epsilon$$



Memory cells

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot f([\mathbf{x}_t; \mathbf{h}_{t-1}])$$
$$\mathbf{h}_t = g(\mathbf{c}_t)$$
$$\mathbf{f}_t = \sigma(f_f([\mathbf{x}_t; \mathbf{h}_{t-1}])) \quad \text{"forget gate"}$$
$$\mathbf{i}_t = \sigma(f_i([\mathbf{x}_t; \mathbf{h}_{t-1}])) \quad \text{"input gate"}$$



LSTM

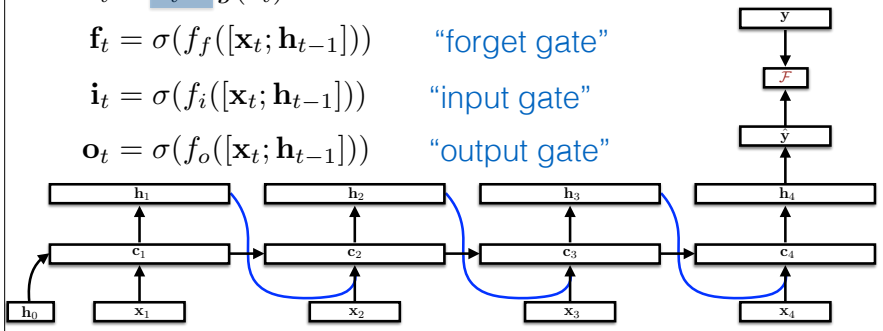
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot f([\mathbf{x}_t; \mathbf{h}_{t-1}])$$

$$\mathbf{h}_t = \mathbf{o}_t \odot g(\mathbf{c}_t)$$

$$\mathbf{f}_t = \sigma(f_f([\mathbf{x}_t; \mathbf{h}_{t-1}])) \quad \text{“forget gate”}$$

$$\mathbf{i}_t = \sigma(f_i([\mathbf{x}_t; \mathbf{h}_{t-1}])) \quad \text{“input gate”}$$

$$\mathbf{o}_t = \sigma(f_o([\mathbf{x}_t; \mathbf{h}_{t-1}])) \quad \text{“output gate”}$$



Another Visualization

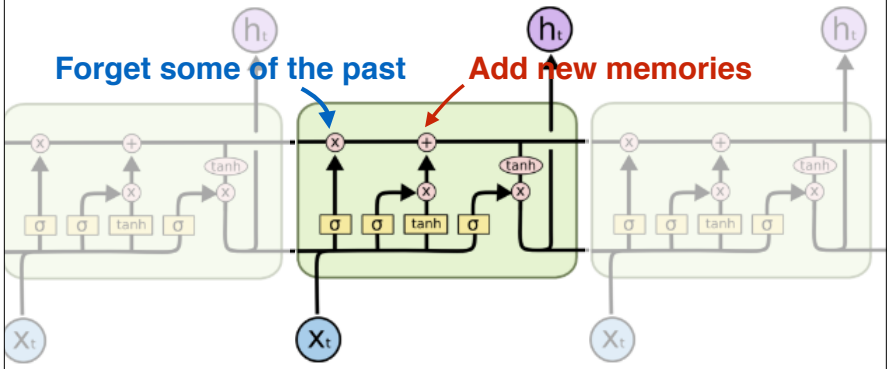


Figure credit: Christopher Olah

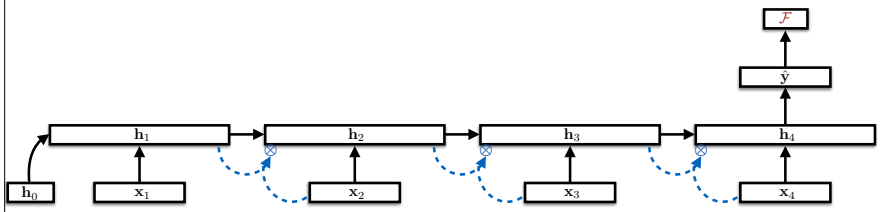
Gated Recurrent Units (GRUs)

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t$$

$$\mathbf{z}_t = \sigma(f_z([\mathbf{h}_{t-1}; \mathbf{x}_t]))$$

$$\mathbf{r}_t = \sigma(f_r([\mathbf{h}_{t-1}; \mathbf{x}_t]))$$

$$\tilde{\mathbf{h}}_t = f([r_t \odot \mathbf{h}_{t-1}; \mathbf{x}_t]))$$



Summary

- Better gradient propagation is possible when you use additive rather than multiplicative/highly non-linear recurrent dynamics

RNN $\mathbf{h}_t = f([\mathbf{x}_t; \mathbf{h}_{t-1}])$

LSTM $\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot f([\mathbf{x}_t; \mathbf{h}_{t-1}])$

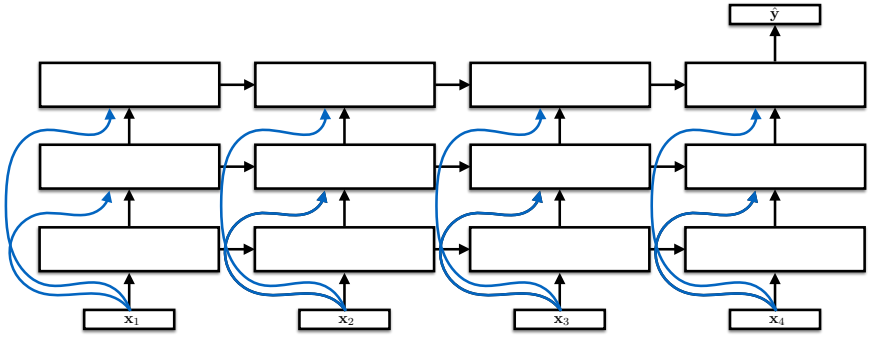
GRU $\mathbf{h}_t = (\mathbf{1} - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot f([\mathbf{x}_t; \mathbf{r}_t \odot \mathbf{h}_{t-1}])$

A Few Tricks of the Trade

- Depth
- Dropout

"Deep" LSTMs

- This term has been defined several times, but the following is the most standard convention

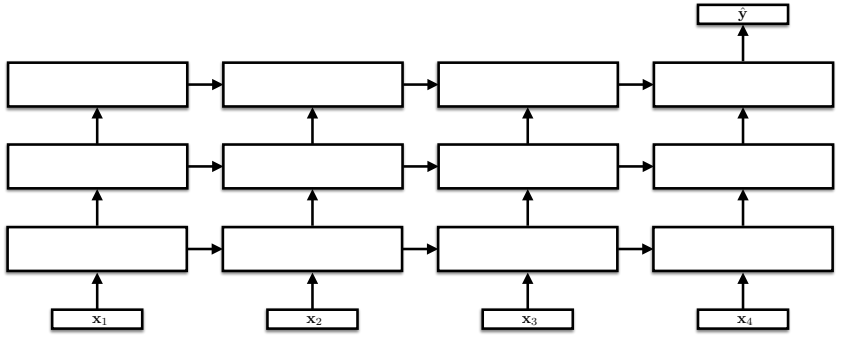


Does Depth Matter?

- Yes, it helps
- It seems to play a less significant role in text than in audio/visual processing
  - H1: More transformation of the input is required for ASR, image recognition, etc., than for common text applications (word vectors become customized to be "good inputs" to RNNs whereas you're stuck with what nature gives you for speech/vision)
  - H2: less effort has been made to find good architectures (RNNs are expensive to train; have been widely used for less long)
  - H3: back prop through time + depth is hard and we need better optimizers
  - Many other possibilities…
- 2-8 layers seems to be standard
- Input "skip" connections are used often but by no means universally

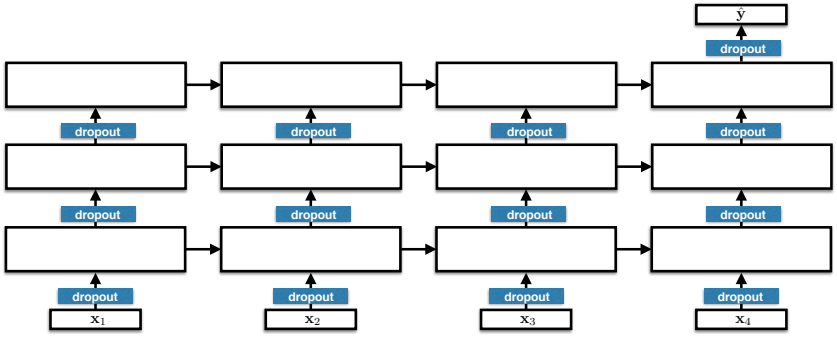
Dropout and Deep LSTMs

- Applying dropout layers requires some care



Dropout and Deep LSTMs

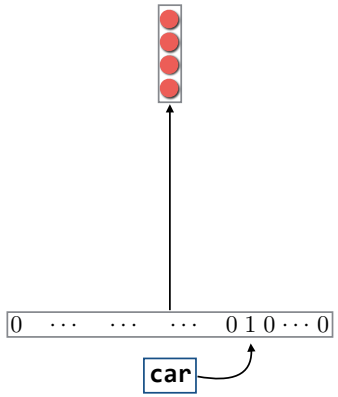
- Apply dropout between layers, but not on the recurrent connections



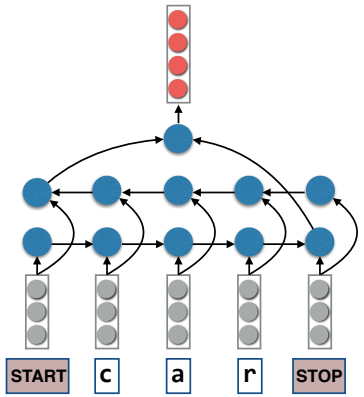
Bidirectional RNNs

- We can read a sequence from left to right to obtain a representation
- Or we can read it from right to left
- Or we can read it from both and combine the representations

Word Embedding Models



Memorize



Generalize

Language modeling

CharLSTM > Word Lookup

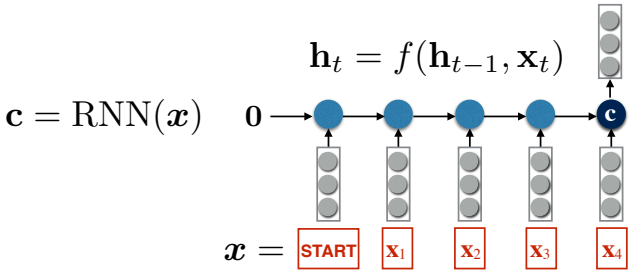
		ppl Words	ppl Chars	Δ
Analytic	English	59.4	57.4	-2.0

Language modeling
Word similarities

increased	John
reduced	Richard
improved	George
expected	James
decreased	Robert
targeted	Edward

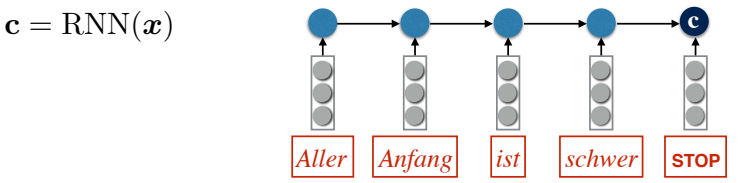
Sequence to Sequence transduction with RNNs

Recurrent Neural Networks (RNNs)



What is a vector representation of a sequence $\boldsymbol{x}$?

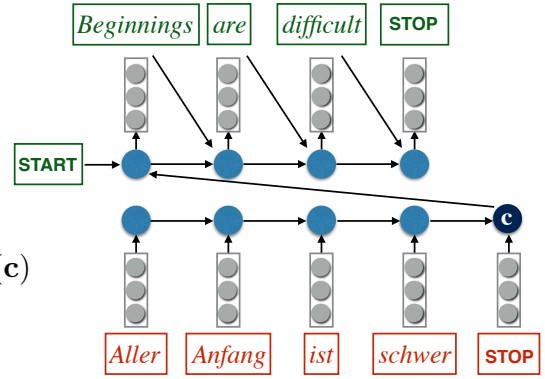
RNN Encoder-Decoders



What is the probability of a sequence $\boldsymbol{y} \mid \boldsymbol{x}$?

Cho et al. (2014); Sutskever et al. (2014)

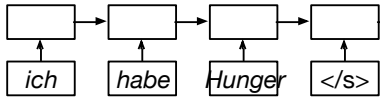
RNN Encoder-Decoders

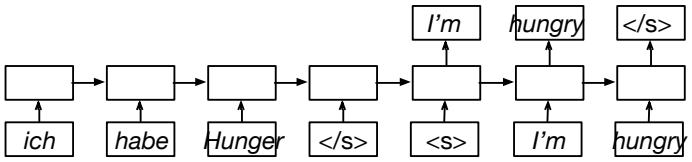


$$\mathbf{c} = \text{RNN}(\boldsymbol{x})$$
$$\boldsymbol{y} \mid \mathbf{c} \sim \text{RNNLM}(\mathbf{c})$$

What is the probability of a sequence $\boldsymbol{y} \mid \boldsymbol{x}$?

Cho et al. (2014); Sutskever et al. (2014)





Sutskever et al. (2014)

Method	test BLEU score (ntst14)
Bahdanau et al. [2]	28.45
Baseline System [29]	33.30
Single forward LSTM, beam size 12	26.17
Single reversed LSTM, beam size 12	30.59
Ensemble of 5 reversed LSTMs, beam size 1	33.00
Ensemble of 2 reversed LSTMs, beam size 12	33.27
Ensemble of 5 reversed LSTMs, beam size 2	34.50
Ensemble of 5 reversed LSTMs, beam size 12	**34.81**

Ensembles of NNs

- Sutskever noticed that their single models did not work well
- But by combining N independently trained models and obtaining a "consensus", the performance could be improved a lot
- This is called **ensembling**.

Encode anything as a vector!