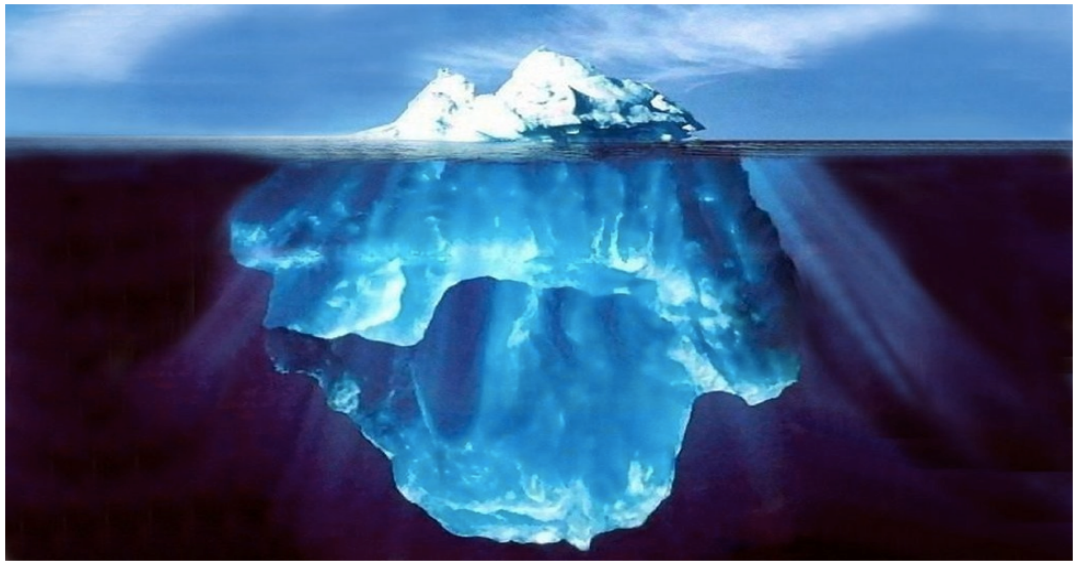
ML4NLP

Deep Structured Prediction – RNNs



Dan Goldwasser

Purdue University

dgoldwas@purdue.edu

10,000 feet view

- So far, we looked at simple classification problems.
  - Assume a word window, that provides fixed sized inputs.
    - In case you "run out of input" – zero padding.
- *What can you do if the size of the input is not fixed?*
  - Some notion of compositionality is needed
  - Simplest approach: sum up word vectors
    - Document structure is lost.. **Can we do better?**

Reminder: *Language Models*

- A Language model defines a probability distribution over a sequence of words:

$$P(w_1, \ldots, w_n)$$

- Simple, yet very useful idea!
  - Estimate using a large collection of text (no supervision!)
  - **P("I like NLP") > P("me like NLP")**

- Key assumption: **Markov model**

Reminder: *Language Models*

- Markov model: *Probability of word$_i$ is conditioned on previous **n** words*

$$P(w_1, ..., w_n) = \prod_{i=1}^{n} P(w_i|w_1, .., w_{i-1}) = \prod_{i-1}^{n} P(w_i|w_{i-n-1}, .., w_{i-1})$$

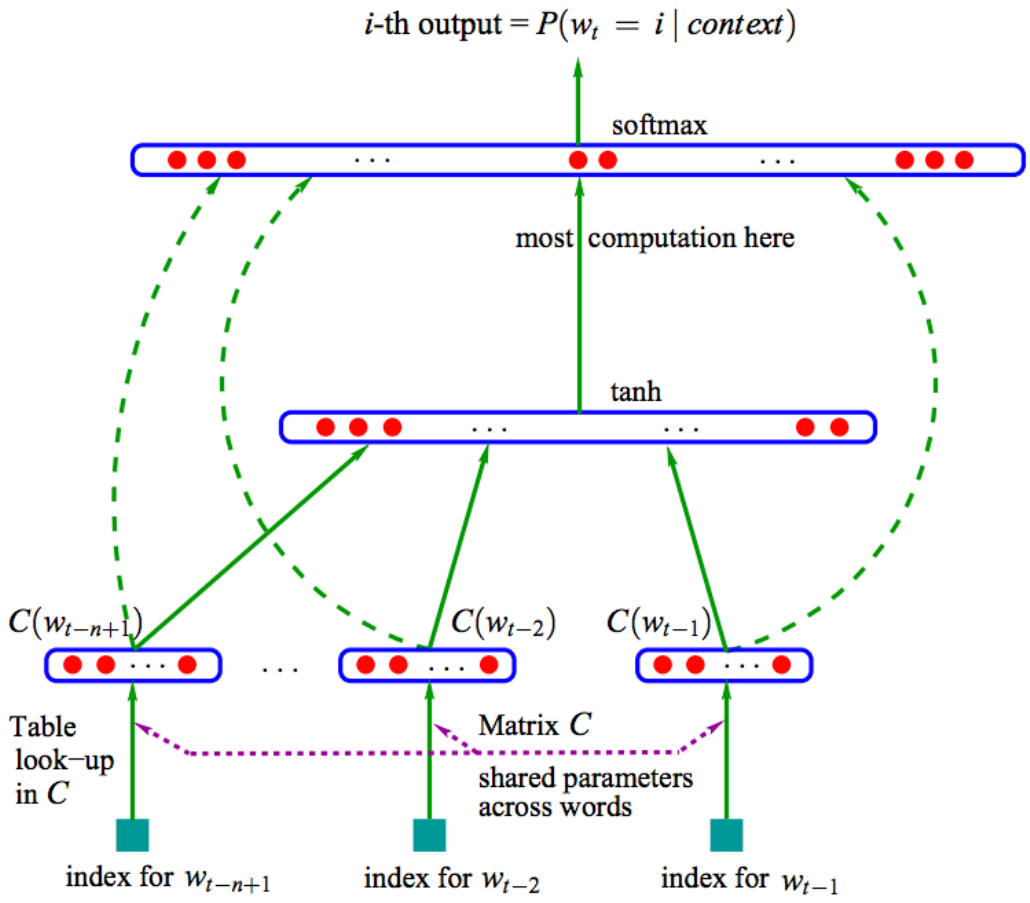
  - **Why is it needed?**
  - What will happen if we increase/decrease n?

Reminder: *Language Models*

- Language models will become **more accurate**, yet **harder to estimate** as n grows.
  - Even for a small size of n, the number of parameters is too large!

- **Question**: *do we need to condition on more than 2-3 words?*

Neural Language Model – Take 1

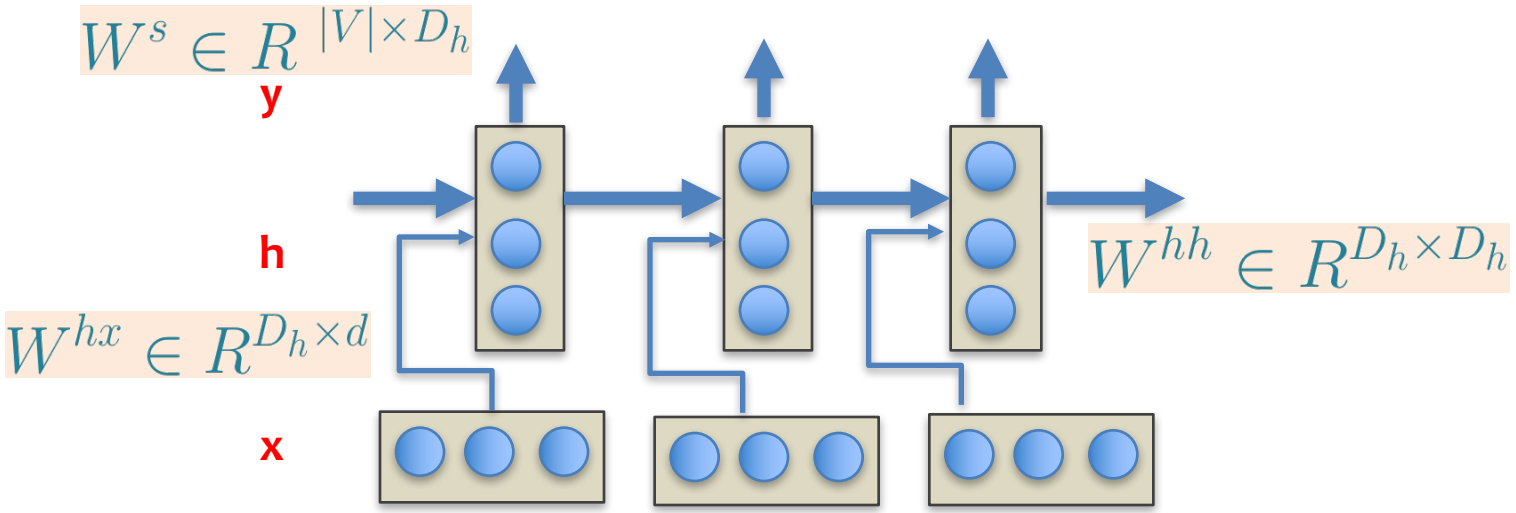


A Neural Probabilistic Language Model. Bengio et-al 2003

Recurrent Neural Networks

- A NN version of a language model.
  - More broadly: deal with **data over time.**
- Unlike N-gram models, an RNN conditions the current word on all previous words.
- **Efficient**, both in time and space

Recurrent Neural Networks



Input is a word (vectors) sequence: $x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n$

At any given time step *i* :

$$h_i = \sigma\left(W^{hh} h_{i-1} + W^{hx} x_i\right)$$

$$\hat{y} = \text{softmax}(W^s\ h_i)$$

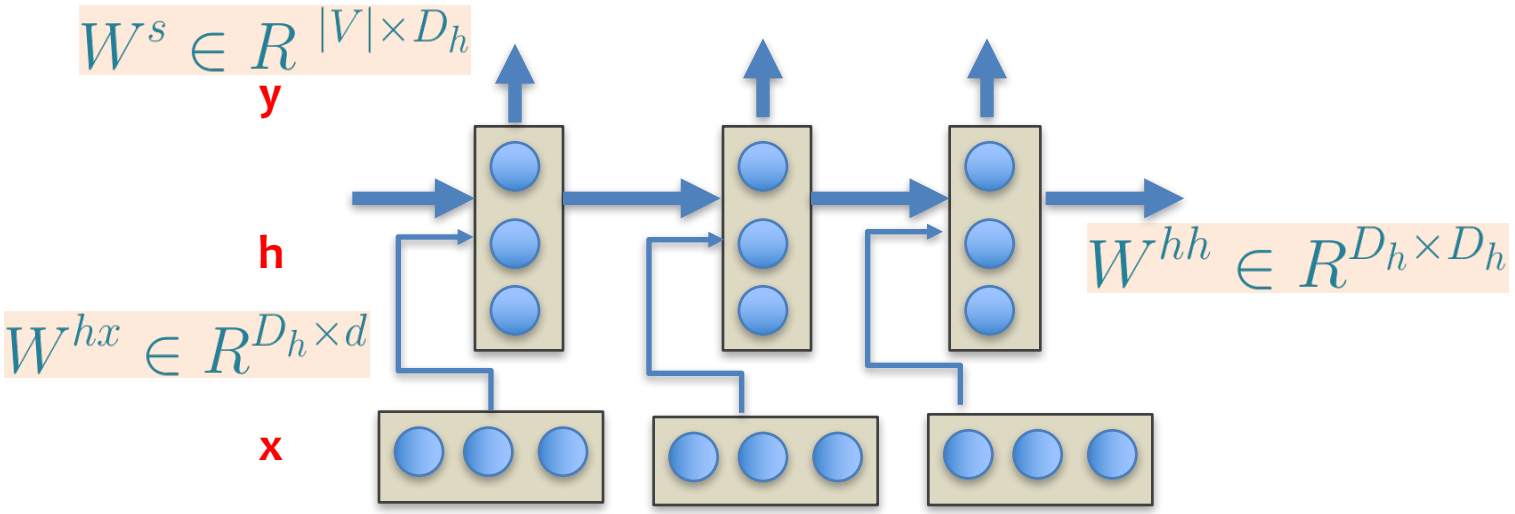
$$P(x_{i+1} = v_j | x_i, \ldots, x_1) = \hat{y}_{i,j}$$

Recurrent Neural Networks

- Similar to traditional LM, we define a probability distribution, estimated using an RNN
  - I.e., $\hat{y} \in R^{|V|}$ is a probability distribution over V
  - We can define a loss function (cross-entropy):

$$J^{(t)}(\theta) = -\sum_{j=1}^{|V|} y_{t,j} \log \hat{y}_{t,j}$$

RNN: Forward Propagation



The cat sat on the mat. Where has the cat sat?

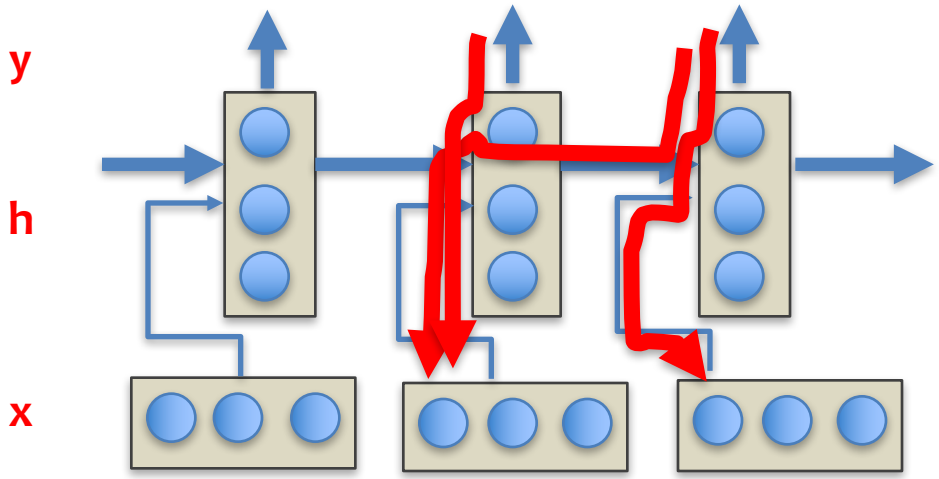
The cat sat on the hat. Where has the cat sat?

RNN-LM Results

Model	Perplexity
Kneser-Ney 5-gram	141
Random forest [Xu 2005]	132
Structured LM [Filimonov 2009]	125
Feedforward NN LM	116
Syntactic NN LM [Emami 2004]	110
RNN trained by BP	113
RNN trained by BPTT	106
4x RNN trained by BPTT	98

Extensions of recurrent neural network language model by Mikolov et al'11

Back-propagation for RNN



$$h_i = W^{hx} f(h_{i-1}) + W x_i$$

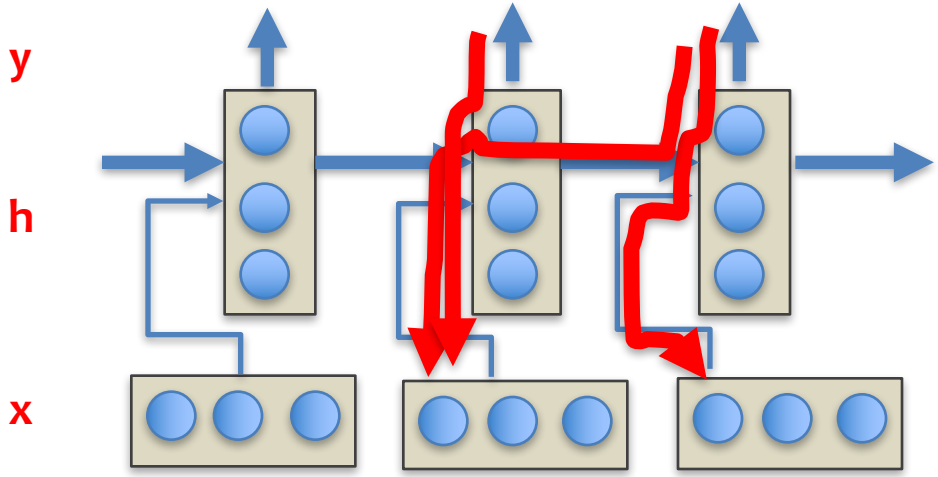
$$\hat{y}_i = W^{s} f(h_i)$$

$$\frac{\partial E}{\partial W} = \sum_{i=1}^{T} \frac{\partial E_i}{\partial W}$$

$$\frac{\partial E_i}{\partial W} = \sum_{k=1}^{i} \frac{\partial Ei}{\partial y_i} \frac{\partial y_i}{\partial h_i} \frac{\partial h_i}{\partial h_k} \frac{\partial h_k}{\partial W}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{i} \frac{\partial h_j}{\partial h_{j-1}}$$

Vanishing/Exploding Gradient



$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{i} \frac{\partial h_j}{\partial h_{j-1}}$$

$$\frac{d\mathbf{f}}{d\mathbf{x}} = \left[\frac{\partial \mathbf{f}}{\partial x_1} \quad \cdots \quad \frac{\partial \mathbf{f}}{\partial x_n}\right] = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=k+1}^{t} W^T \text{diag}[f'(h_{j-1})]$$

Vanishing/Exploding Gradient

- **Key issue**: *propagating the error over long word sequences*
- The gradient is a product of Jacobians, the product can become **very large** or **very small** **very quickly**

$$\frac{\partial E_i}{\partial W} = \sum_{k=1}^{t} \frac{\partial Ei}{\partial y_i} \frac{\partial y_i}{\partial h_i} \frac{\partial h_i}{\partial h_k} \frac{\partial h_k}{\partial W}$$

Why is this a problem?

RNN Extensions

- **Key issue**: *long range dependencies between inputs.*
  - "how can we know which word is important to keep around, when predicting the i+1 word?"
- **Solution idea**: *complex hidden units that implement a "memory"*
  - Maintain "old memories" representing relevant long range dependencies
  - Error updates can be back-propagated at different strengths.

Gated Recurrent Units (GRU)

- Until now, we assumed a simple hidden layer:
  - representing the previous steps and input word

$$h_i = \sigma\left(W^{hh} h_{i-1} + W^{hx} x_i\right)$$

- In GRU's the picture is more complex, it adds **gates,** that control how the hidden state is computed
- *Essentially, more layers that can be learned from data*
  - **Update Gate**
  - **Reset Gate**

Gated Recurrent Units (GRU)

$$h_i = \sigma\left(W^{hh} h_{i-1} + W^{hx} x_i\right)$$ ← **Original RNN**

GRU:

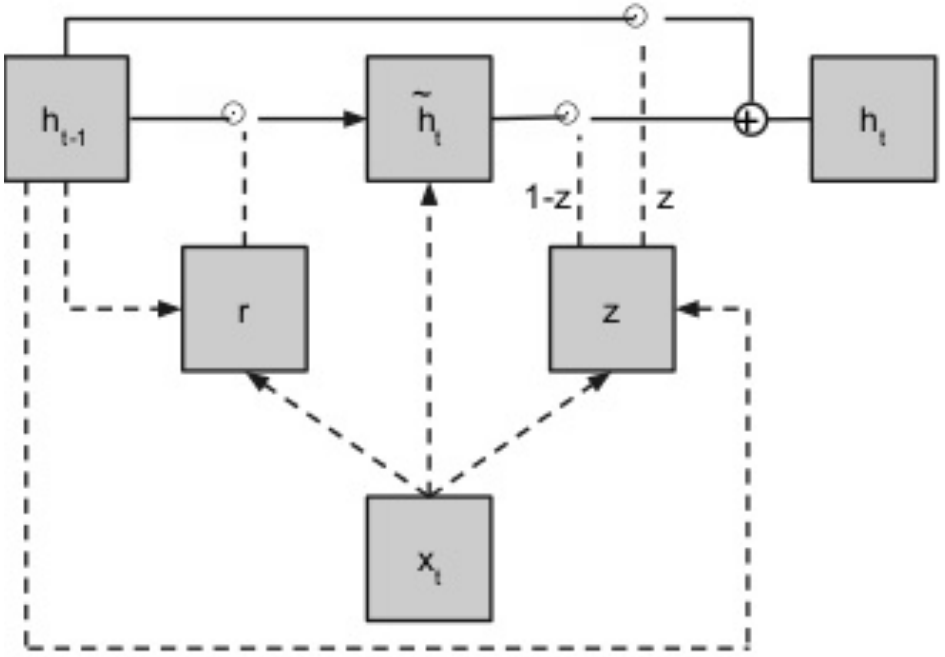
Update Gate: $$z_i = \sigma\left(W^z x_i + U^z h_{i-1}\right)$$

Reset Gate: $$r_i = \sigma\left(W^r x_i + U^r h_{i-1}\right)$$

New memory $$\tilde{h}_i = \tanh\left(W x_i + r_i \circ U h_{i-1}\right)$$

Final memory (aka *hidden Layer*) $$h_i = z_t \circ h_{i-1} + (1 - z_i) \circ \tilde{h}_i$$

Gated Recurrent Unit - GRU



⊕ Element wise addition

⊙ Element wise multiplication

Routes information can propagate along

Involved in modifying information flow and values

Why it works

- Learn a set of parameters for each one of the gates
  - **Recall**: *gates output a probability*
  - If **reset** gate is ~0: **ignore previous hidden state**
    - "forget" irrelevant information
    - **Short term dependencies**
  - If **update** gate ~1:
    copy past information
    - "*remember*" *past state*
    - **long term dependencies**

$$z_i = \sigma\left(W^z x_i + U^z h_{i-1}\right)$$

$$r_i = \sigma\left(W^r x_i + U^r h_{i-1}\right)$$

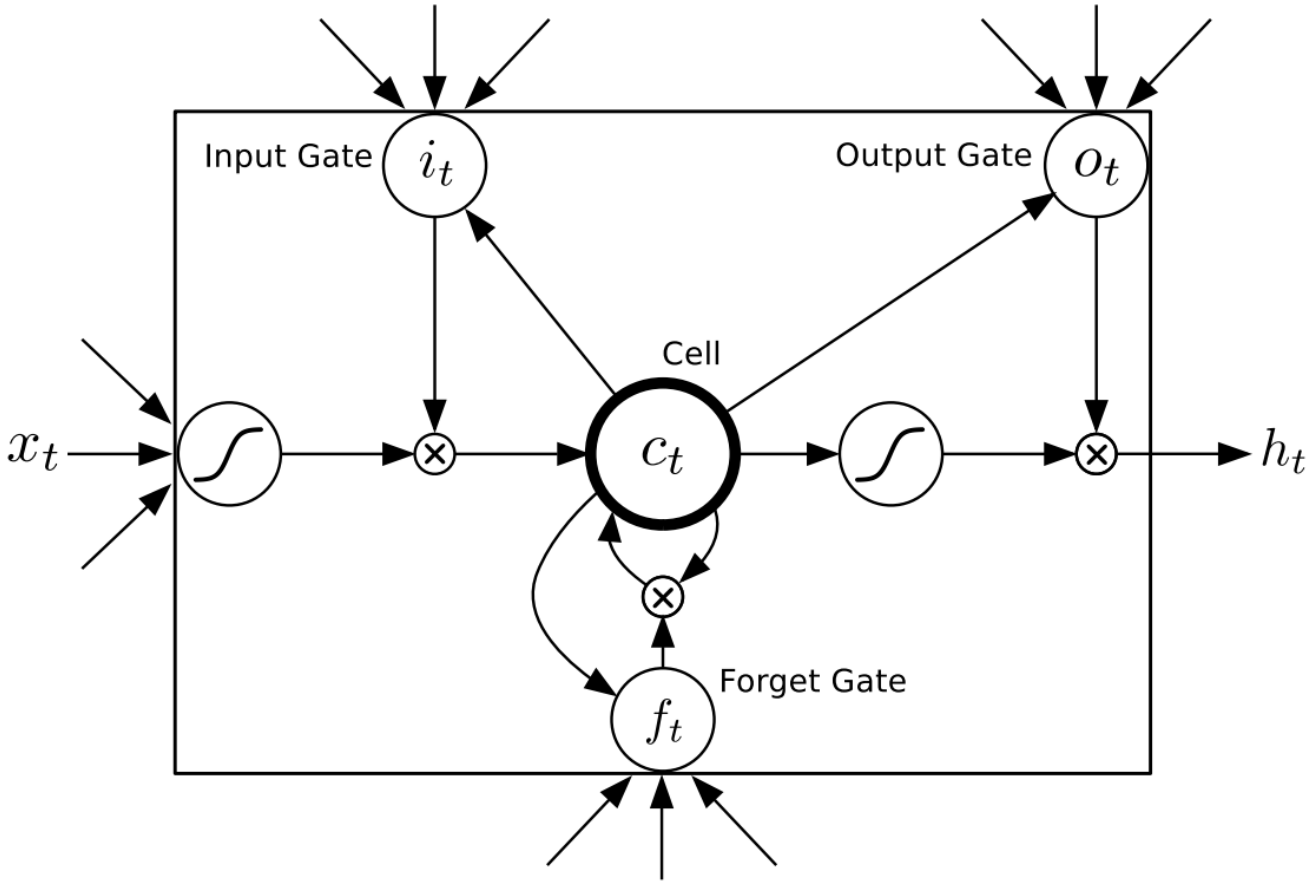
$$\tilde{h_i} = \tanh\left(W x_i + r_i \circ U h_{i-1}\right)$$

$$h_i = z_t \circ h_{i-1} + (1 - z_i) \circ \tilde{h_i}$$

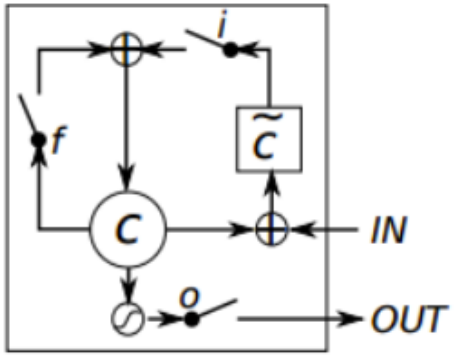
Long-Short-Term-Memories (LSTM)

- Similar (and older!) idea, though more complex
- Input gate $$i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$$
- Forget gate $$f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$$
- Output $$o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$$
- New memory $$\tilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$$
- Final Memory $$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$
- Final hidden state $$h_t = o_t \circ \tanh(c_t)$$

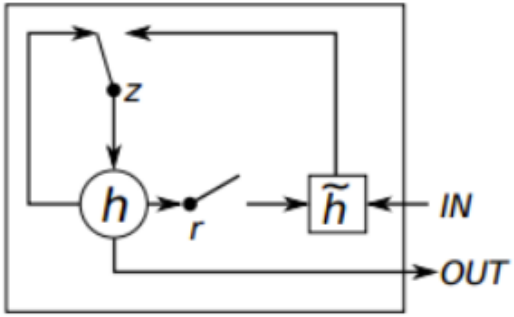
LSTM



RNN vs. LSTM vs. GRU



(a) Long Short-Term Memory

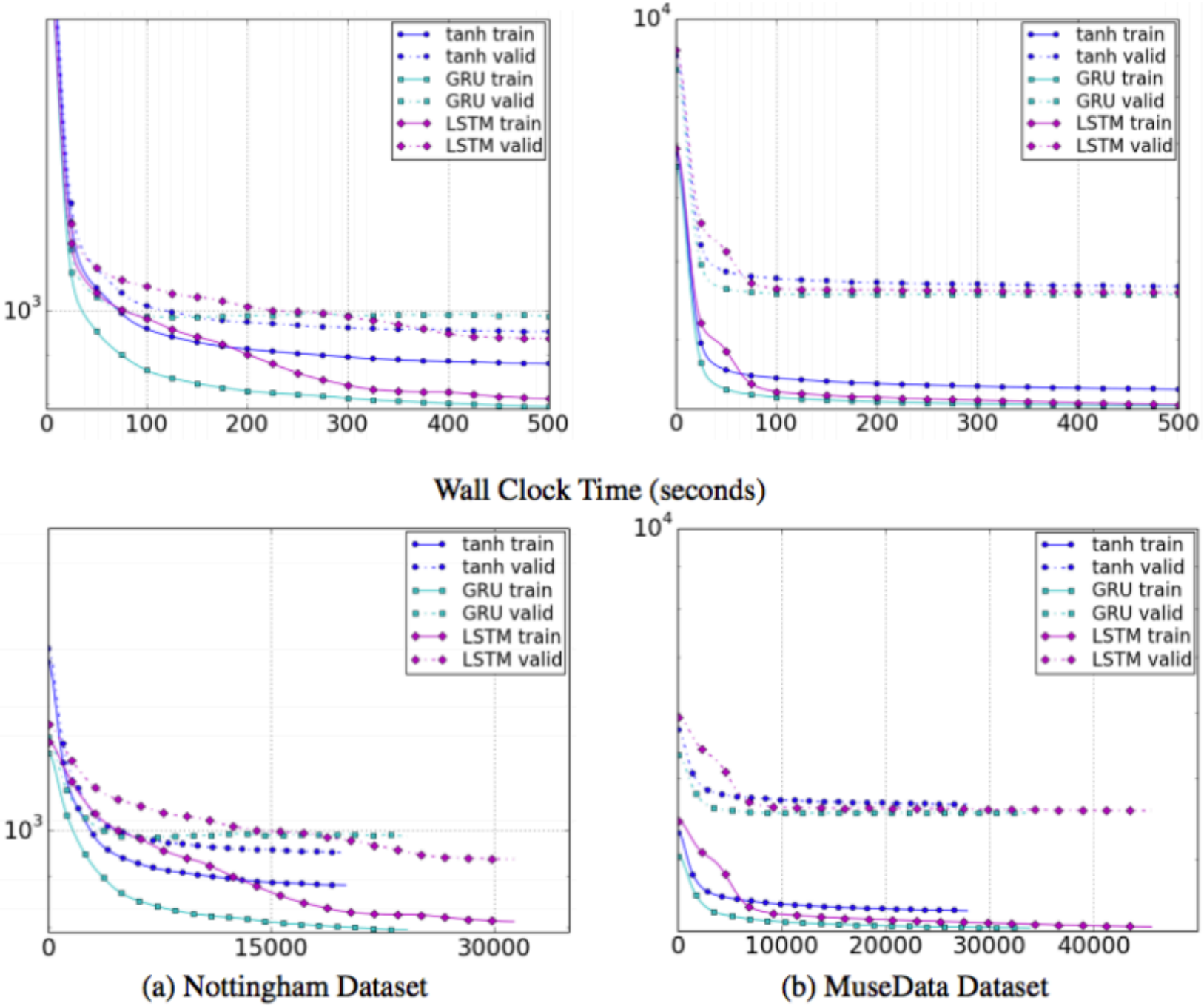


(b) Gated Recurrent Unit

			tanh	GRU	LSTM
Music Datasets	Nottingham	train	3.22	2.79	3.08
		test	**3.13**	3.23	3.20
	JSB Chorales	train	8.82	6.94	8.15
		test	9.10	**8.54**	8.67
	MuseData	train	5.64	5.06	5.18
		test	6.23	**5.99**	6.23
	Piano-midi	train	5.64	4.93	6.49
		test	9.03	**8.82**	9.03
Ubisoft Datasets	Ubisoft dataset A	train	6.29	2.31	1.44
		test	6.44	3.59	**2.70**
	Ubisoft dataset B	train	7.61	0.38	0.80
		test	7.62	**0.88**	1.26

Table 2: The average negative log-probabilities of the training and test sets.

Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. Chung et-al 2014



LSTM for Semantic Tasks

Learning Statistical Scripts with LSTM Recurrent Neural Networks.
Pichotta et-al AAAI-'16

Motivation

- *Following the Battle of Actium, Octavian invaded Egypt. As he approached Alexandria, Antony's armies deserted to Octavian on August 1, 30 BC.*
  - Did Octavian defeat Antony?

Motivation

- *Following the Battle of Actium, Octavian invaded Egypt. As he approached Alexandria,* ***Antony's armies deserted to Octavian*** *on August 1, 30 BC.*
  - Did Octavian defeat Antony?

Motivation

- *Antony's armies deserted to Octavian*

  ⇒

  Octavian defeated Antony

- Not simply a paraphrase rule!

- Need world knowledge.

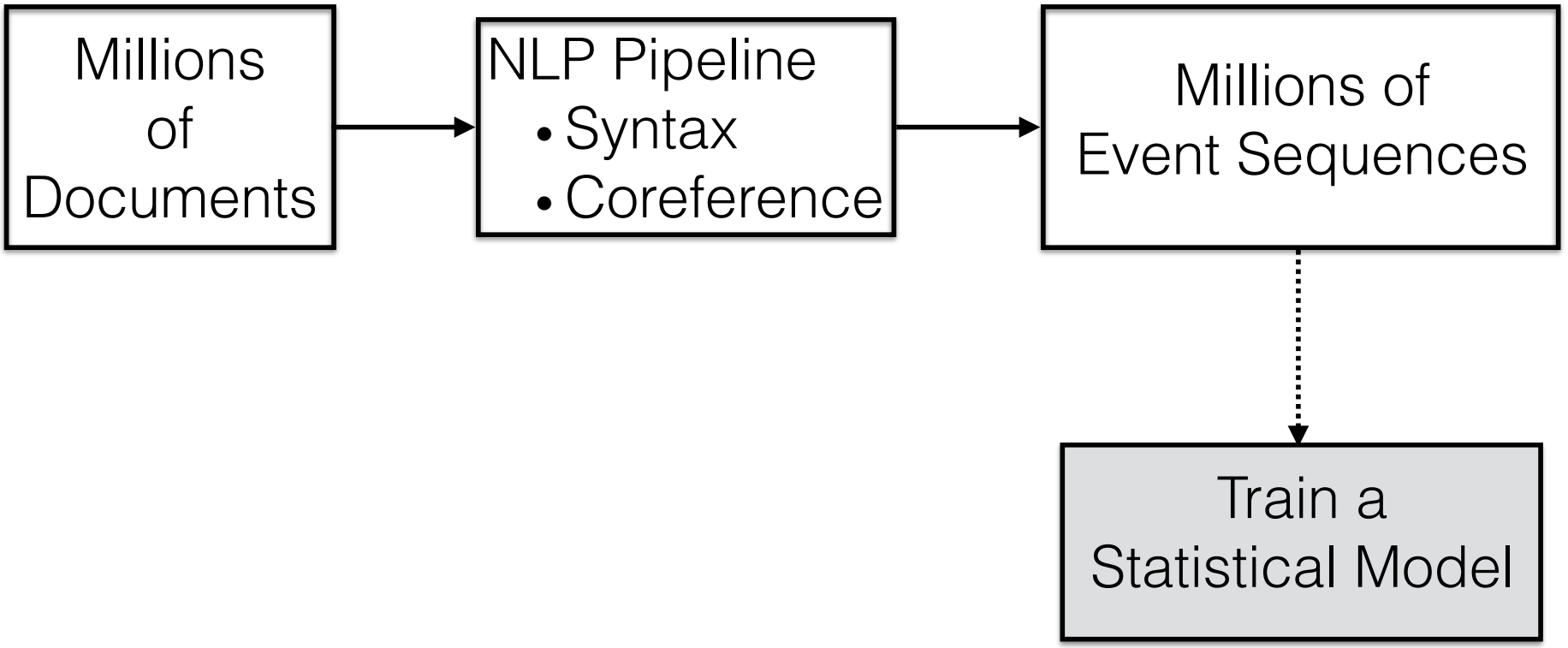
Scripts

- **Scripts**: models of events in sequence.
  - "**Event**": verb + arguments.
  - Events don't appear in text randomly, but according to world dynamics.
  - Scripts try to capture these dynamics.
  - Enable automatic inference of implicit events, given events in text (e.g. *Octavian defeated Antony*).

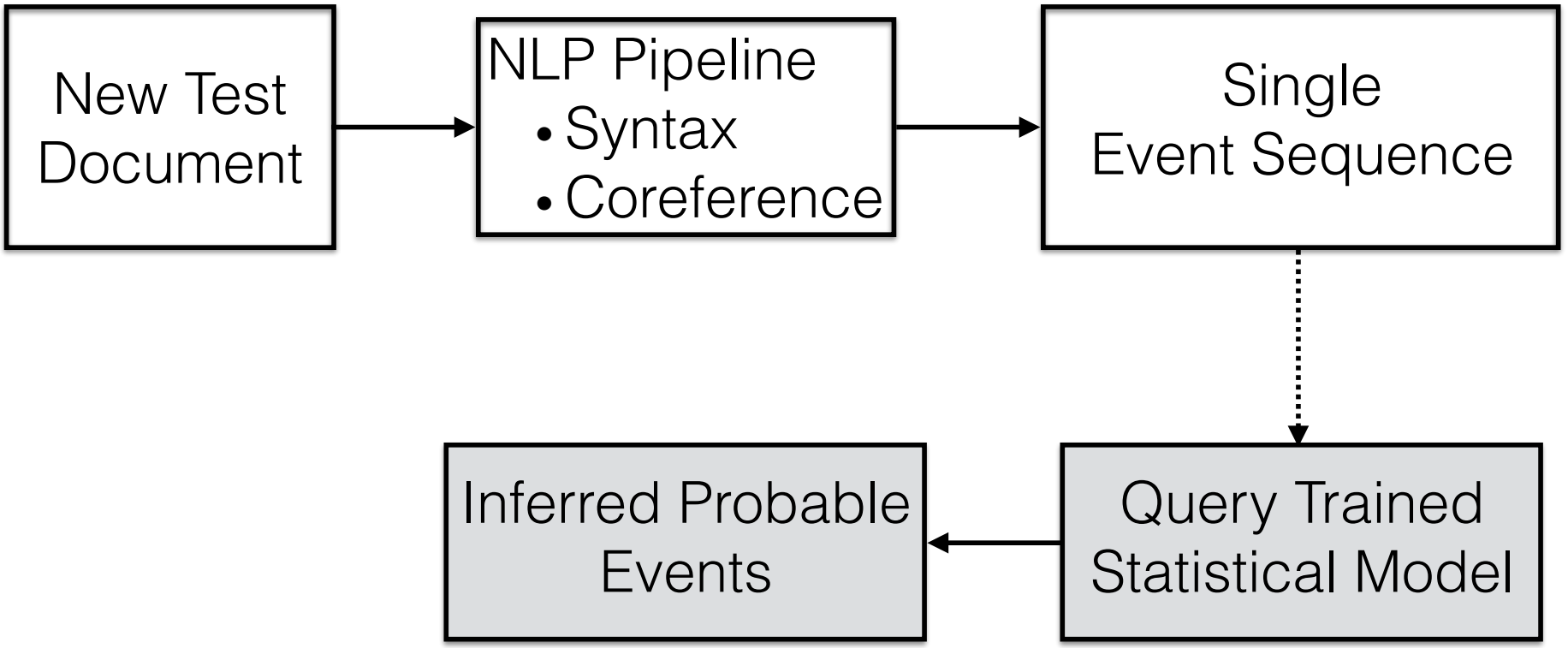
Background: Statistical Scripts

- **Statistical Scripts**: Statistical Models of Event Sequences.
- Non-statistical scripts date back to the 1970s [Schank & Abelson 1977].
- Statistical script learning is a small-but-growing subcommunity [e.g. Chambers & Jurafsky 2008].
- Model the probability of an event given prior events.

Background: Statistical Script Learning



Background: Statistical Script Inference



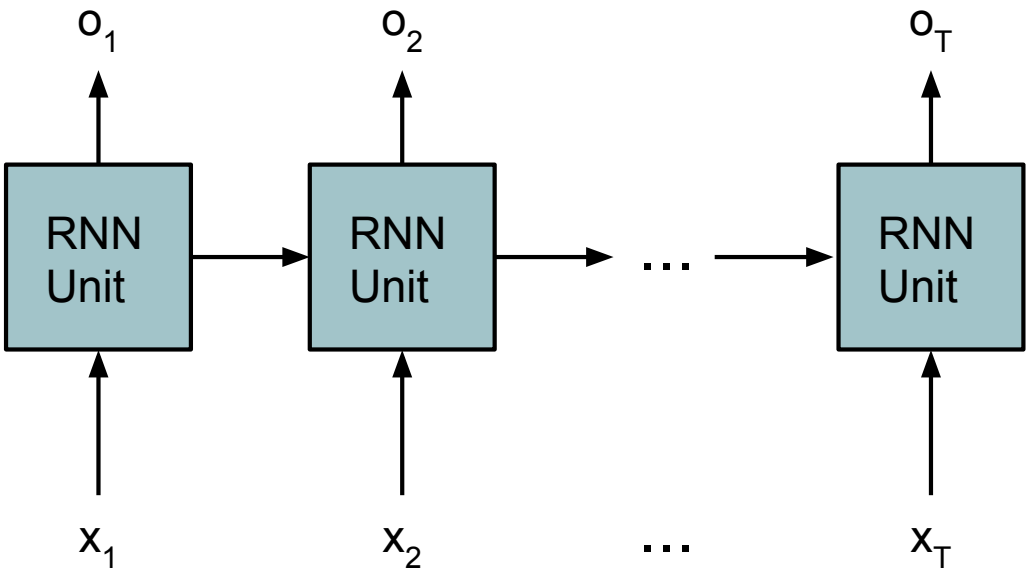
LSTM Script models

- Train LSTM sequence model on event sequences.
  - Events are (verbs + arguments).
  - Arguments can have noun info, coref info, or both.
- To infer events, the model generates likely events from sequence.

LSTM Script models

- *Mary's late husband Matthew, whom she married at 21 because she loved him, …*

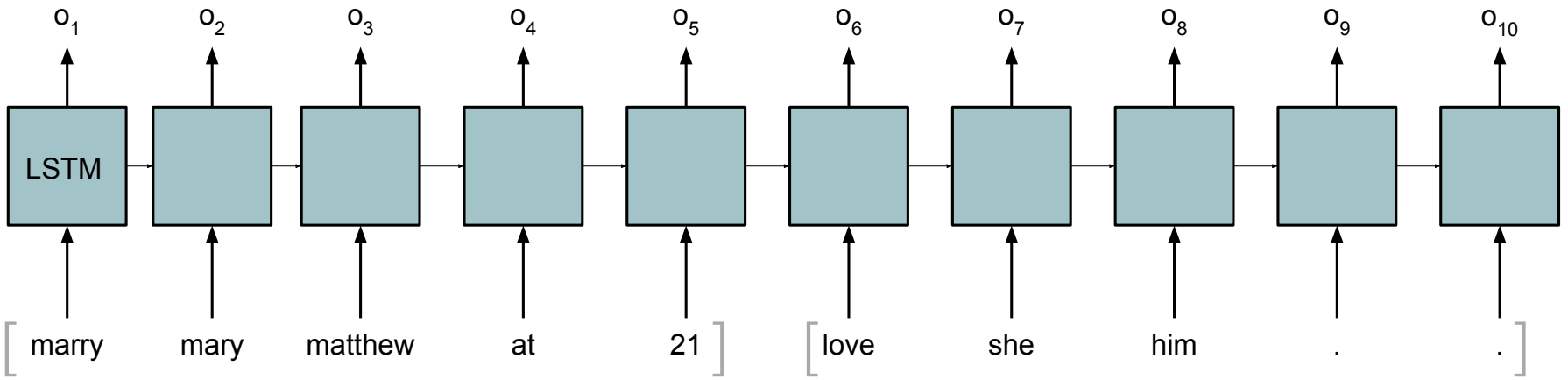
  [marry, mary, matthew, at, 21]; [love, she, him]



LSTM Script models

- *Mary's late husband Matthew, whom she married at 21 because she loved him, …*

[marry, mary, matthew, at, 21]; [love, she, him]



Experimental Setup

- Train on English Wikipedia.
- Use Stanford CoreNLP to extract event sequences.
- Train LSTM using Batch Stochastic Gradient Descent with Momentum.
- To infer next events, have the LSTM generate additional events with highest probability.

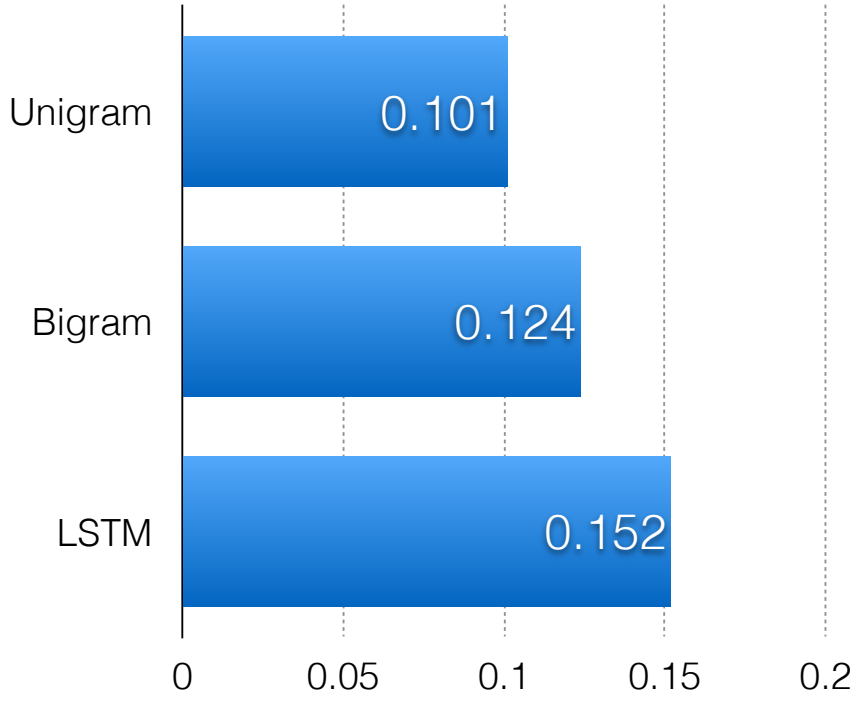
Evaluation

- "Narrative Cloze" (Chambers & Jurafsky, 2008): from an unseen document, hold one event out, try to infer it given remaining document.
- "Recall at *k*" (Jans et al., 2012): make *k* top inferences, calculate recall of held-out events.
- (More metrics in the paper.)

Evaluation

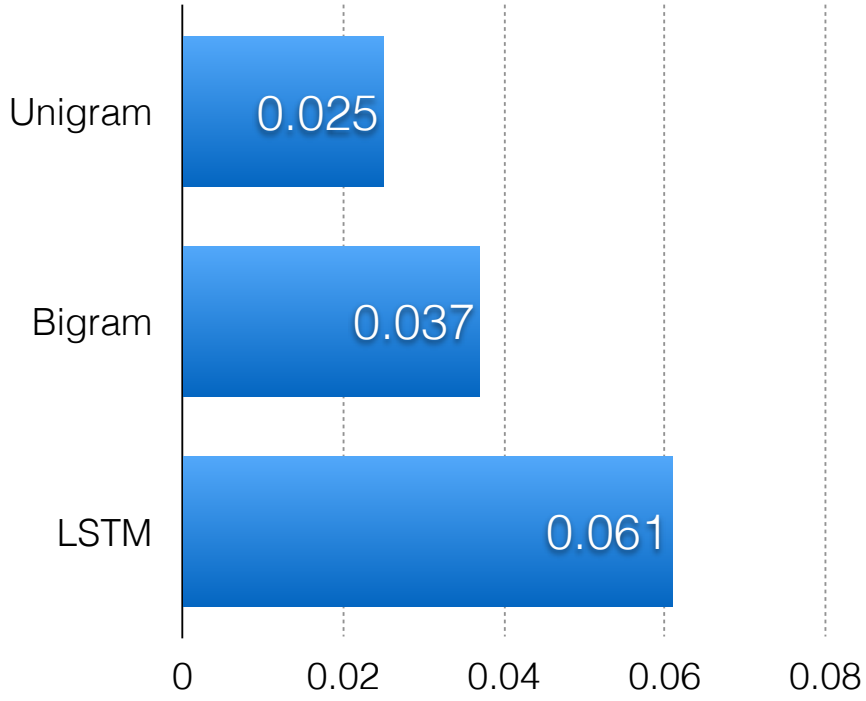
- Three Systems:
  - **Unigram:** Always guess most common events.
  - **Bigram:** Variations of Pichotta & Mooney (2014)
    - Uses event co-occurrence counts.
    - Best-published system on task.
  - **LSTM:** LSTM script system (this work).

Results: Predicting Verbs & Coreference Info



Recall at 25 for inferring Verbs & Coref info

Results: Predicting Verbs & Nouns



Recall at 25 for inferring Verbs & Nouns

Human Evaluations

- Solicit judgments on individual inferences on Amazon Mechanical Turk.
  - Have annotators rate inferences from 1-5 (or mark “Nonsense,” scored 0).
  - More interpretable.

Results: Crowdsourced Eval