Sequence Modeling

CS 288 Spring 2026

UC Berkeley

cal-cs288.github.io/sp26

Berkeley EECS BAIR

Today's Question:
What is a Sequence Model?

- The classifier model we saw last lecture treats a sentence as a bag of words.
- But it ignores the ordering of words in a sentence

Sequence modeling

- Building models that understand inputs where order matters and where each element depends on what came before.
- Unlike a perceptron or an MLP, it can represent order, context, or dependencies across time.

Plans for this lecture:

- Survey sequence modeling tasks
- Architecture for sequence modeling: Recurrent neural networks (RNNs)!
    - In theory, infinite context
    - Motivates *attention (next week!)*

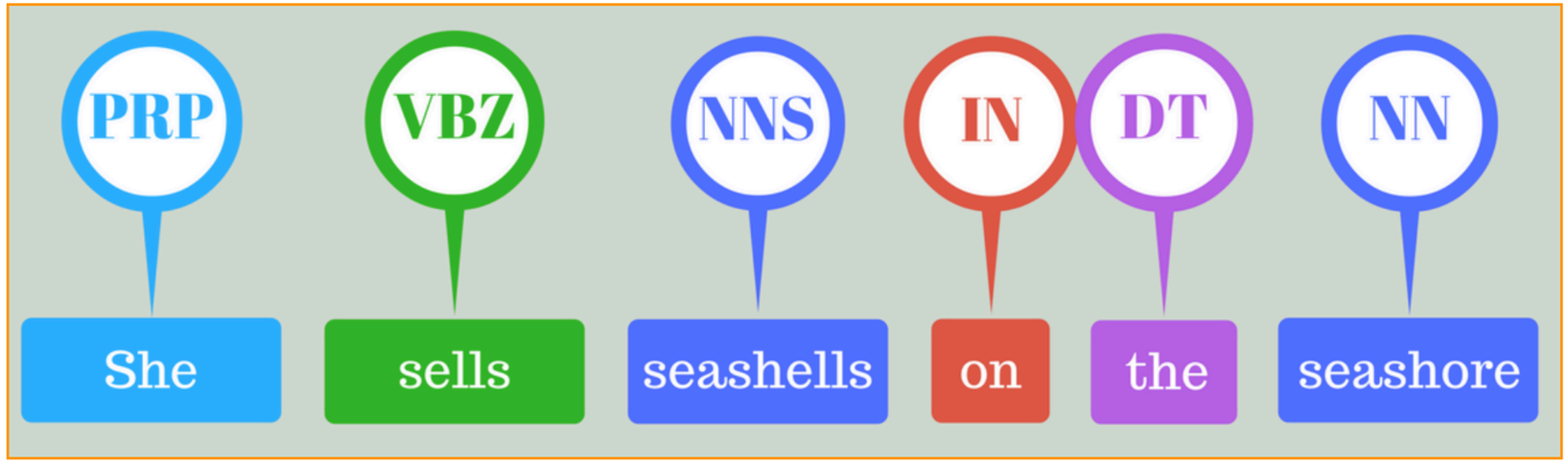
Sequence modeling tasks

Sequence modeling tasks

Technically, all NLP tasks benefit from sequence modeling!

- Classification: Text → Label
- Language modeling: Text → Next word
- Sequence-to-sequence: Text → Text (e.g., machine translation, question answering, … more in the next week's lecture)
- Sequence labeling: Text (a sequence of n words) → a sequence of n labels (one label per word)
    - What are some examples?

Part-of-speech (POS) tagging



PRP: Personal pronoun

VBZ: Verb, 3rd person singular present

NN: singular noun
NNS: plural noun

IN: preposition or subordinating conjunction
DT: determiner

Named Entity Recognition

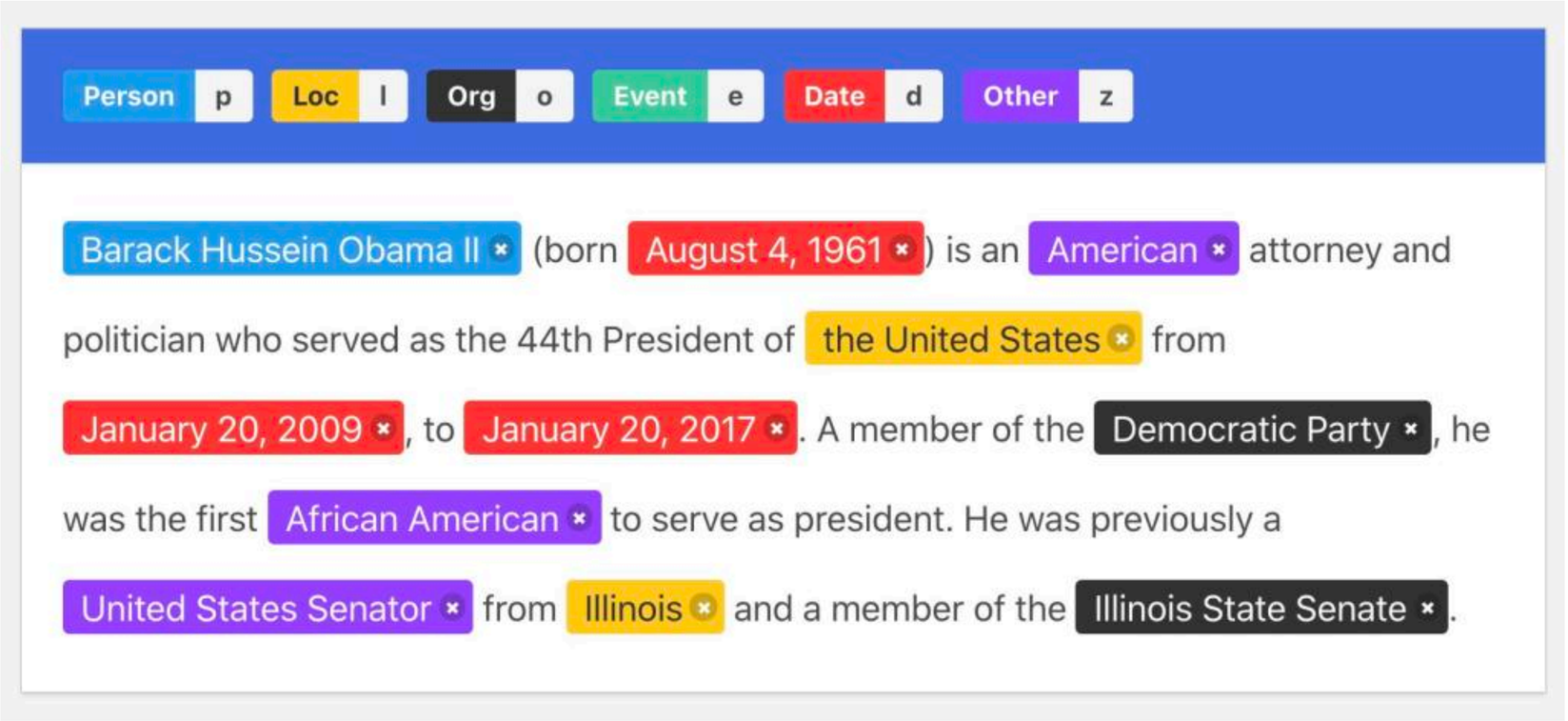
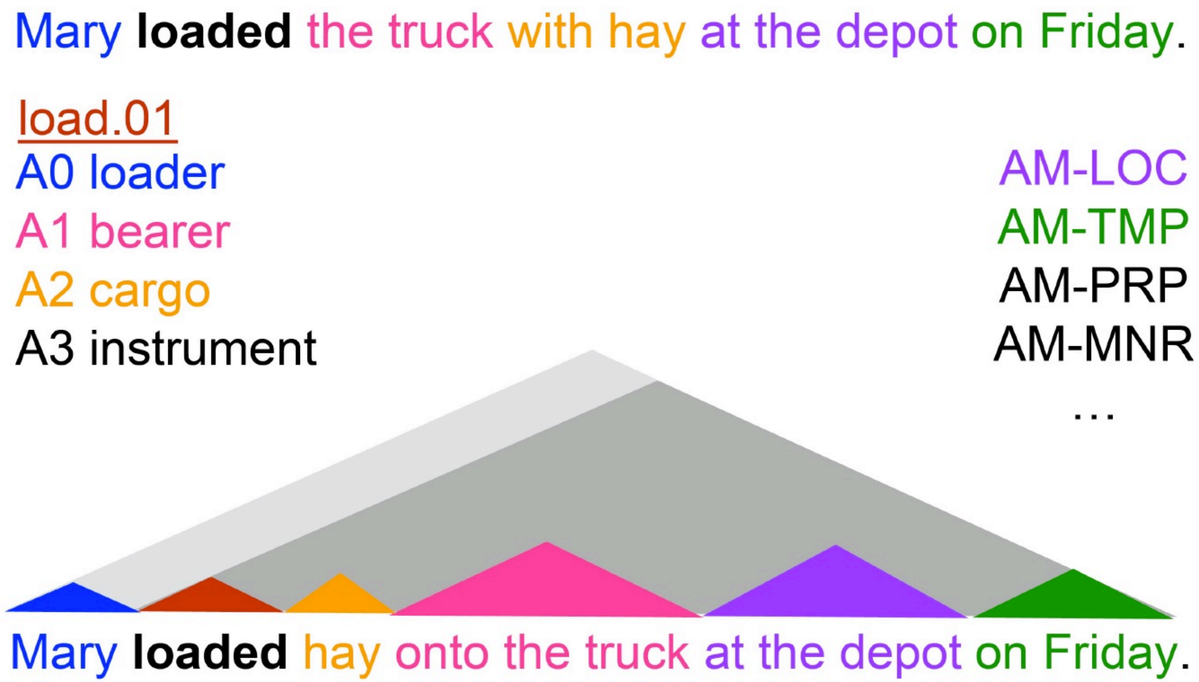


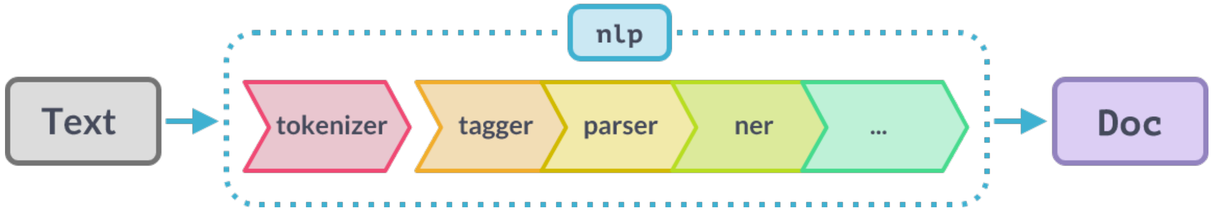
Image: https://www.analyticsvidhya.com/blog/2021/11/a-beginners-introduction-to-ner-named-entity-recognition/

Semantic role labeling

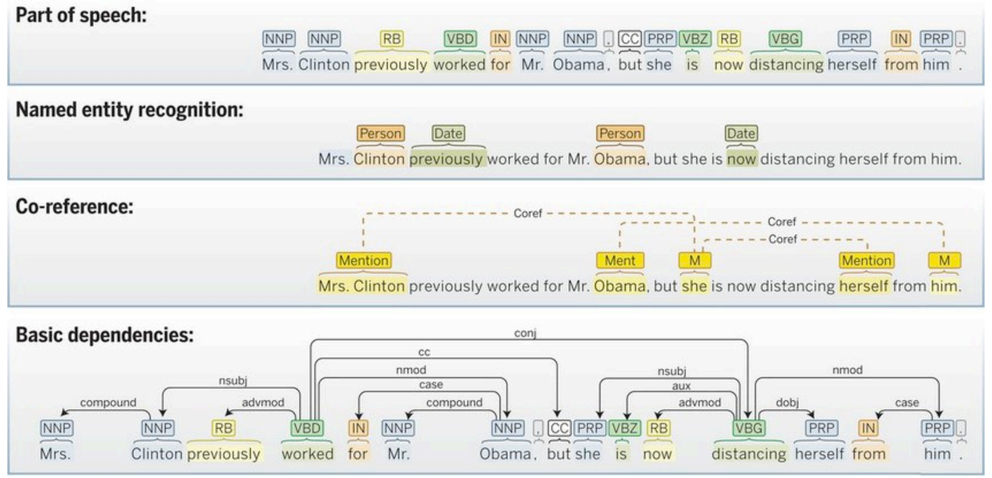


https://devopedia.org/semantic-role-labelling

NLP pipeline



NAME	COMPONENT	CREATES	DESCRIPTION
tokenizer	`Tokenizer`	`Doc`	Segment text into tokens.
PROCESSING PIPELINE			
tagger	`Tagger`	`Token.tag`	Assign part-of-speech tags.
parser	`DependencyParser`	`Token.head`, `Token.dep`, `Doc.sents`, `Doc.noun_chunks`	Assign dependency labels.
ner	`EntityRecognizer`	`Doc.ents`, `Token.ent_iob`, `Token.ent_type`	Detect and label named entities.



https://spacy.io/usage/processing-pipelines

https://stanfordnlp.github.io/CoreNLP/pipeline.html

Deeper dive I: Parts of speech

- Tag each word in a sentence with its part of speech
- Disambiguation task: each word might have different functions in different contexts
  - Consider a word “man” — what would be a POS tag?
  - The/DT man/NN bought/VBD a/DT boat/NN
  - The/DT old/NN man/VBP the/DT boat/NN

Same word, different tags

earnings growth took a **back/JJ** seat
a small building in the **back/NN**
a clear majority of senators **back/VBP** the bill
Dave began to **back/VB** toward the door
enable the country to buy **back/RP** about debt
I was twenty-one **back/RB** then

Some words have many functions!

JJ: adjective, NN: single or mass noun, VBP: Verb, non-3rd person singular present
VB: Verb, base form, RP: particle, RB: adverb

Deeper dive I: Parts of speech

How many part of speech tags do you think English has?

A) < 10

B) 10 - 20

C) 20 - 40

D) > 40

The answer is (D) - well, depends on definitions!

Penn treebank part-of-speech tagset

Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coordinating conjunction	*and, but, or*	PDT	predeterminer	*all, both*	VBP	verb non-3sg present	*eat*
CD	cardinal number	*one, two*	POS	possessive ending	*'s*	VBZ	verb 3sg pres	*eats*
DT	determiner	*a, the*	PRP	personal pronoun	*I, you, he*	WDT	wh-determ.	*which, that*
EX	existential 'there'	*there*	PRP$	possess. pronoun	*your, one's*	WP	wh-pronoun	*what, who*
FW	foreign word	*mea culpa*	RB	adverb	*quickly*	WP$	wh-possess.	*whose*
IN	preposition/ subordin-conj	*of, in, by*	RBR	comparative adverb	*faster*	WRB	wh-adverb	*how, where*
JJ	adjective	*yellow*	RBS	superlatv. adverb	*fastest*	$	dollar sign	*$*
JJR	comparative adj	*bigger*	RP	particle	*up, off*	#	pound sign	*#*
JJS	superlative adj	*wildest*	SYM	symbol	+,%, &	"	left quote	' or "
LS	list item marker	*1, 2, One*	TO	"to"	*to*	"	right quote	' or "
MD	modal	*can, should*	UH	interjection	*ah, oops*	(	left paren	[, (, {, <
NN	sing or mass noun	*llama*	VB	verb base form	*eat*	)	right paren	],), }, >
NNS	noun, plural	*llamas*	VBD	verb past tense	*ate*	,	comma	,
NNP	proper noun, sing.	*IBM*	VBG	verb gerund	*eating*	.	sent-end punc	. ! ?
NNPS	proper noun, plu.	*Carolinas*	VBN	verb past part.	*eaten*	:	sent-mid punc	: ; ... – -

45 tags

(Marcus et al., 1993)

based on Wall Street Journal (WSJ)

Other corpora: Brown, Switchboard

A simple baseline

- Most frequent class: Assign each word to the class it occurred most in the training set. (e.g. man/NN)
- How accurate do you think this baseline would be at tagging words?

(A) <50%
(B) 50-75%
(C) 75-90%
(D) >90%

The answer is (D)

- This baseline accurately tags 92.34% of word tokens on Wall Street Journal (WSJ)!
- State of the art ~ 97% (also human-level acc)
- Average English sentence ~14 words
  - Sentence level accuracies: with 0.9214 per word is 31% vs 0.9714 per word is 65%

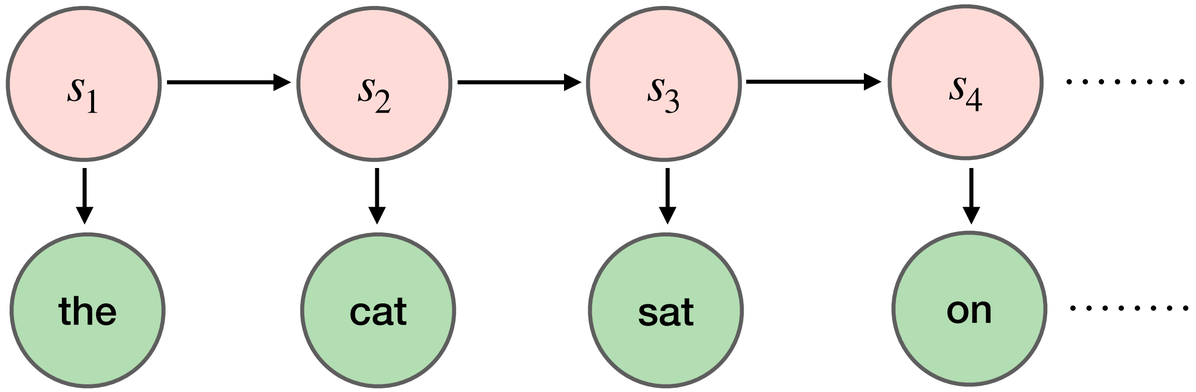
Model: Hidden Markov Model (HMM)

Tags

(hidden events)

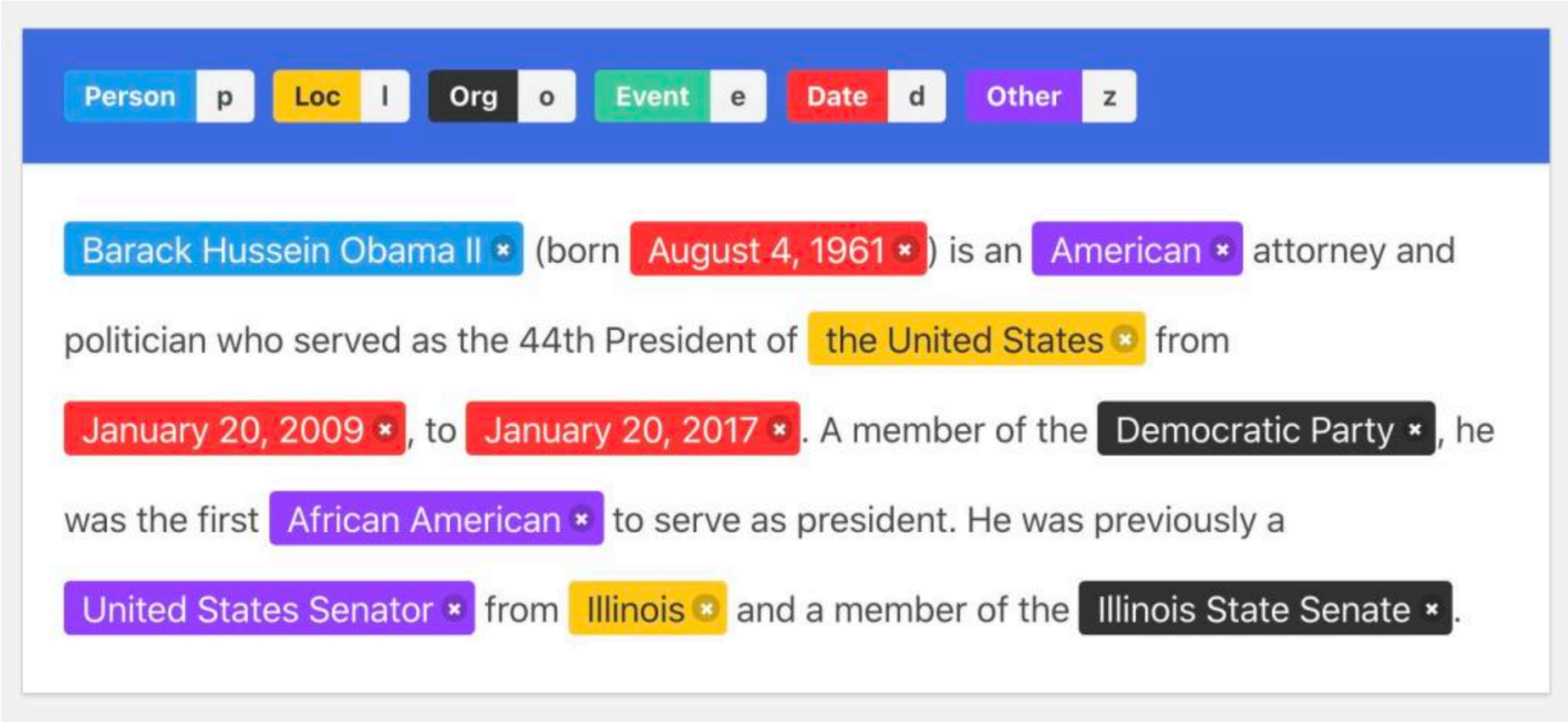
Words

(observed events)



- We don't normally see sequences of POS tags in text
- However, we do observe the words!
- The HMM allows us to *jointly reason* over both **hidden** and **observed** events.
  - Assume that each position has a tag that generates a word

Deeper dive II: Named entity recognition



Named entities

- Named entity, in its core usage, means anything that can be referred to with a proper name.
- NER is the task of 1) finding spans of text that constitute proper names; 2) tagging the type of the entity
- Most common 4 tags:
  - **PER** (Person): “Marie Curie”
  - **LOC** (Location): “New York City”
  - **ORG** (Organization): “Princeton University”
  - **MISC** (Miscellaneous): nationality, events, ..

Named entities

Steve Jobs founded Apple with Steve Wozniak .

PER PER O ORG O PER PER O

Only France and Britain backed Fischler ’s proposal .

O LOC O LOC O PER O O O

O = not an entity

If multiple words constitute a named entity, they will be labeled with the same tag.

Limitation: Can’t distinguish from having two named entities adjacent to each other?

NER: BIO Tagging

[PER Jane Villanueva] of [ORG United] , a unit of [ORG United Airlines Holding] , said the fare applies to the [LOC Chicago] route.

Words	BIO Label
Jane	B-PER
Villanueva	I-PER
of	O
United	B-ORG
Airlines	I-ORG
Holding	I-ORG
discussed	O
the	O
Chicago	B-LOC
route	O
.	O

B: token that begins a span

I: tokens that inside a span

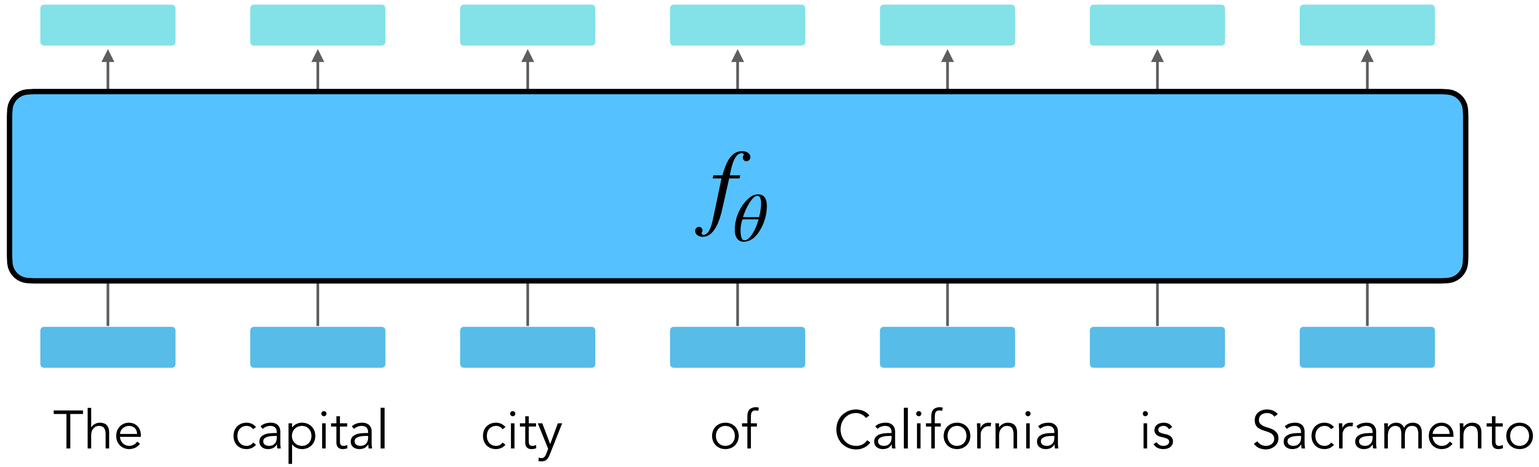
O: tokens outside of a span

Neural Sequence Modeling

Sequence Modeling

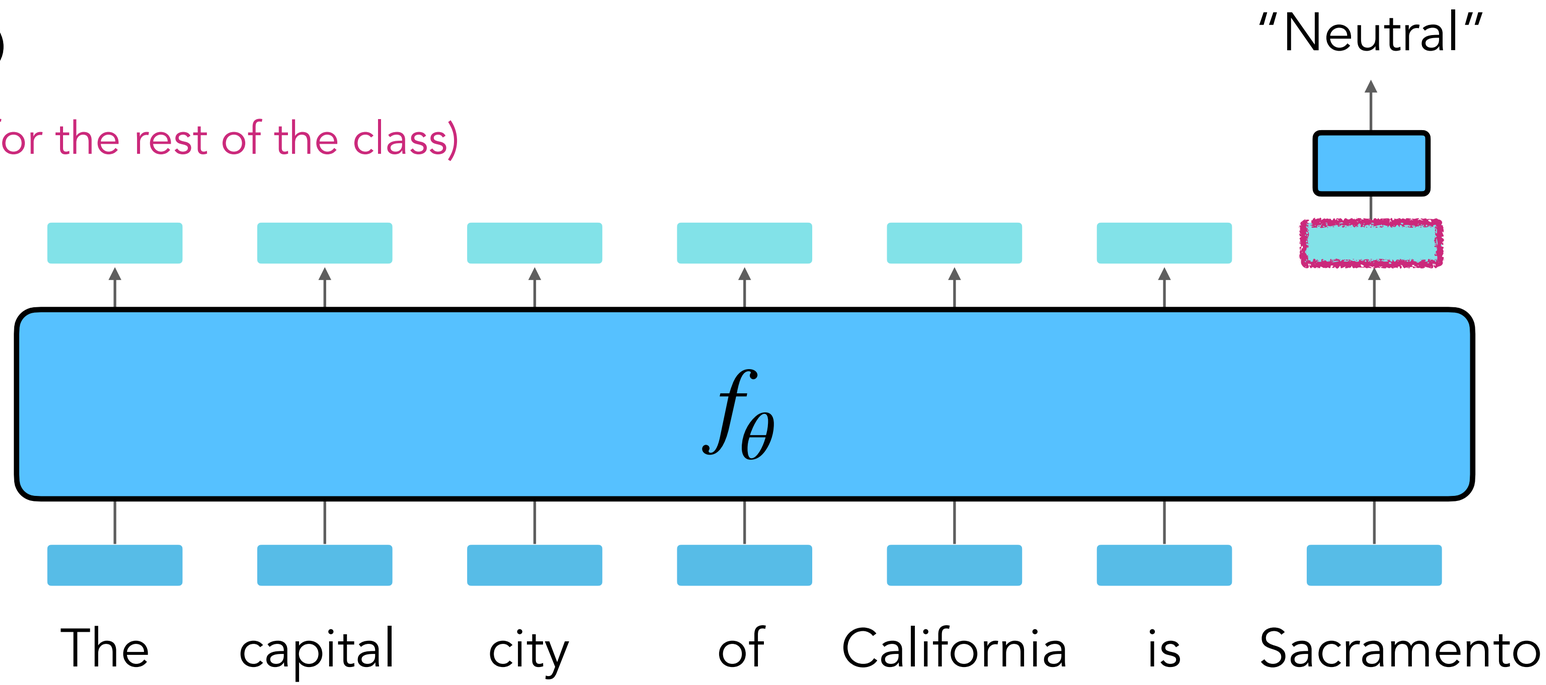
$f_\theta(x_1, \cdots, x_T) = \mathbf{h}_1, \cdots, \mathbf{h}_T$ where $\mathbf{h}_T \in \mathbb{R}^d$ is a hidden state

Why? How do we do an end task from here?
Let's look at three tasks: classification, sequence labeling (e.g., POS tagging), and language modeling!



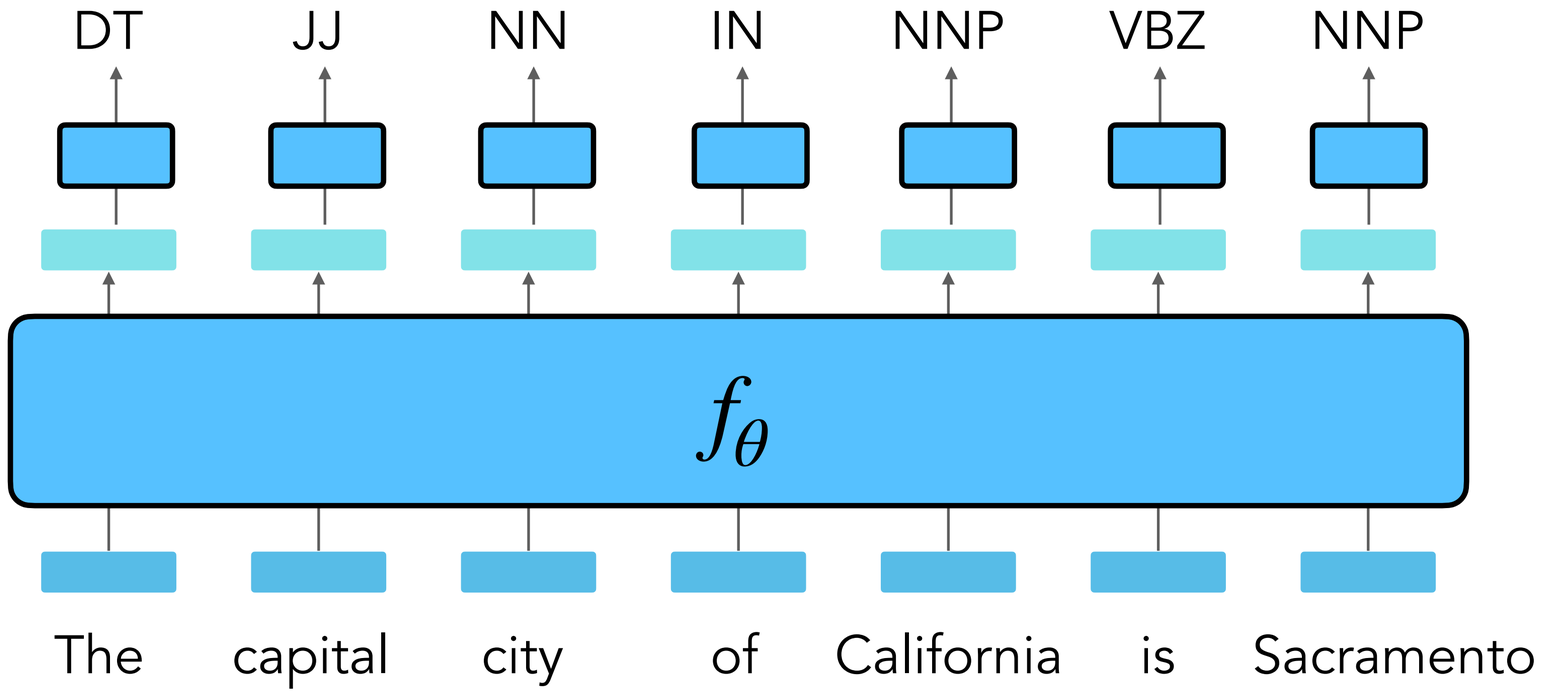
Sequence Modeling: (1) Text Classification

1. $f_\theta(x_1, \cdots, x_T) = \mathbf{h}_1, \cdots, \mathbf{h}_T$ where $\mathbf{h}_T \in \mathbb{R}^d$ is a hidden state
2. Summarize all hidden states into one vector
   - $\bar{\mathbf{h}} = \text{MeanPool}(\mathbf{h}_1, \cdots, \mathbf{h}_T)$
   - $\bar{\mathbf{h}} = \text{MaxPool}(\mathbf{h}_1, \cdots, \mathbf{h}_T)$
   - $\bar{\mathbf{h}} = \mathbf{h}_T$ (We'll assume this for the rest of the class)
3. $\hat{y} = \text{argmax}\mathbf{W}_o\bar{\mathbf{h}}^\intercal$



Sequence Modeling: (2) Part-of-Speech Tagging

1. $f_\theta(x_1, \cdots, x_T) = \mathbf{h}_1, \cdots, \mathbf{h}_T$ where $\mathbf{h}_T \in \mathbb{R}^d$ is a hidden state
2. $\hat{y}_t = \text{argmax}\mathbf{W}_o\mathbf{h}_t^\intercal$



Recurrent Neural Network

How do we actually model $f_\theta(x_1, \cdots, x_T) = \mathbf{h}_1, \cdots, \mathbf{h}_T$?

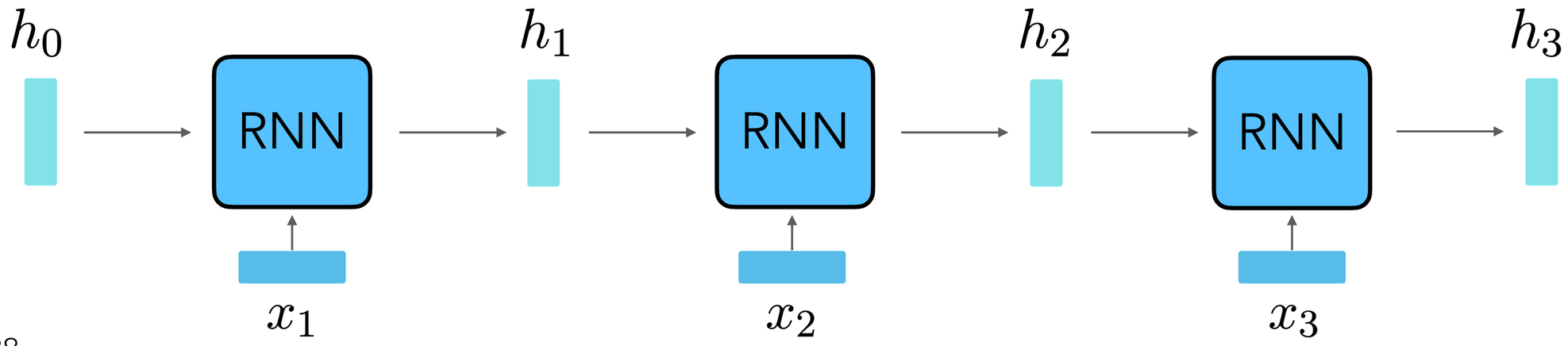
- $\mathbf{h}_0$: Initial hidden state (usually zeros)
- $\mathbf{h}_1 = \sigma(\mathbf{W}_h\mathbf{h}_0 + \mathbf{W}_x\mathbf{x}_1 + \mathbf{b})$, where $\mathbf{x}_1 = \mathbf{E}(x_1)$
- $\mathbf{h}_2 = \sigma(\mathbf{W}_h\mathbf{h}_1 + \mathbf{W}_x\mathbf{x}_2 + \mathbf{b})$, where $\mathbf{x}_2 = \mathbf{E}(x_2)$
- $\mathbf{h}_3 = \sigma(\mathbf{W}_h\mathbf{h}_2 + \mathbf{W}_x\mathbf{x}_3 + \mathbf{b})$, where $\mathbf{x}_3 = \mathbf{E}(x_3)$

$$\mathbf{h}_t = \sigma(\mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{W}_x\mathbf{x}_t + \mathbf{b})$$

Parameters:

$\mathbf{E} \in \mathbf{R}^{|V| \times d_{\text{in}}}$,

$\mathbf{W}_h \in \mathbf{R}^{d \times d}, \mathbf{W}_x \in \mathbf{R}^{d \times d_{\text{in}}}, \mathbf{b} \in \mathbf{R}^d$



Recurrent Neural Network: (1) Text Classification

1. $\mathbf{h}_0$: Initial hidden state (usually zeros)
2. $\mathbf{h}_1 = \sigma(\mathbf{W}_h\mathbf{h}_0 + \mathbf{W}_x\mathbf{x}_1 + \mathbf{b})$, where $\mathbf{x}_1 = \mathbf{E}(x_1)$
3. $\mathbf{h}_2 = \sigma(\mathbf{W}_h\mathbf{h}_1 + \mathbf{W}_x\mathbf{x}_2 + \mathbf{b})$, where $\mathbf{x}_2 = \mathbf{E}(x_2)$
4. $\mathbf{h}_3 = \sigma(\mathbf{W}_h\mathbf{h}_2 + \mathbf{W}_x\mathbf{x}_3 + \mathbf{b})$, where $\mathbf{x}_3 = \mathbf{E}(x_3)$
5. $\hat{y} = \text{argmax}\mathbf{W}_o\mathbf{h}_3^{\intercal}$

Parameters:

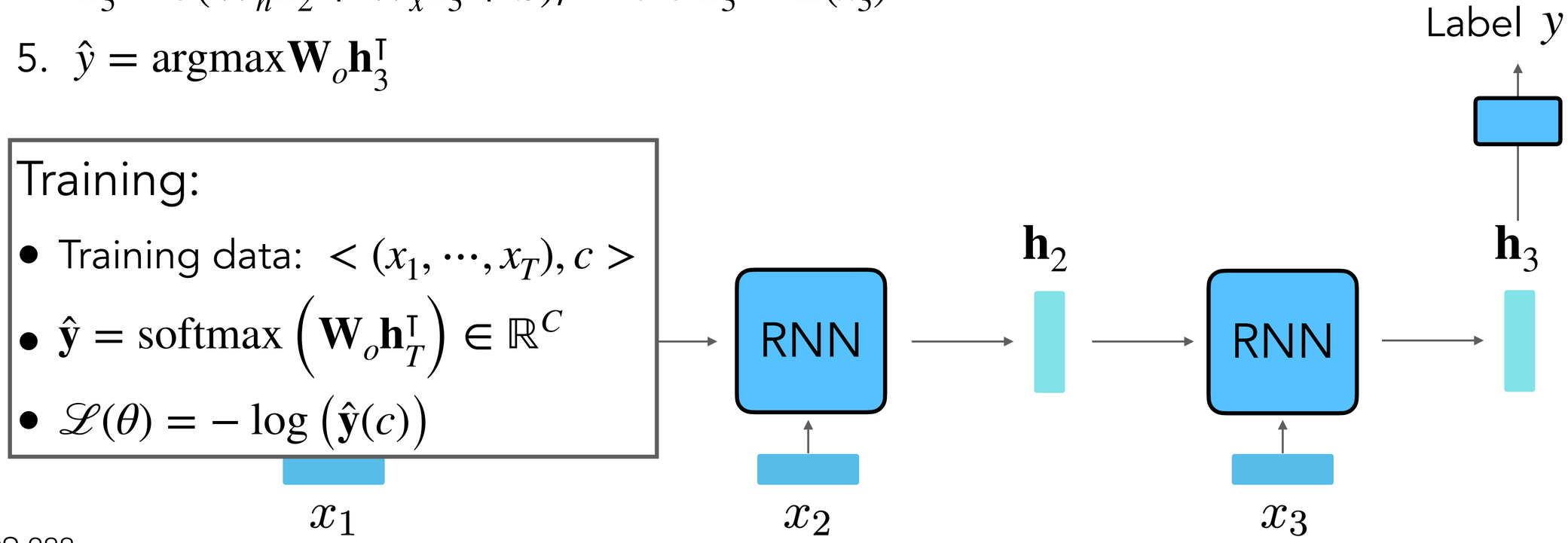
$\mathbf{E} \in \mathbf{R}^{|V| \times d_{\text{in}}}$,

$\mathbf{W}_h \in \mathbf{R}^{d \times d}, \mathbf{W}_x \in \mathbf{R}^{d \times d_{\text{in}}}$,

$\mathbf{b} \in \mathbf{R}^d, \mathbf{W}_o \in \mathbf{R}^{C \times d}$

Training:

- Training data: $< (x_1, \cdots, x_T), c >$
- $\hat{\mathbf{y}} = \text{softmax}\left(\mathbf{W}_o\mathbf{h}_T^{\intercal}\right) \in \mathbb{R}^C$
- $\mathscr{L}(\theta) = -\log\left(\hat{\mathbf{y}}(c)\right)$



Recurrent Neural Network: (2) Part-of-Speech Tagging

1. $\mathbf{h}_0$: Initial hidden state (usually zeros)
2. $\mathbf{h}_1 = \sigma(\mathbf{W}_h\mathbf{h}_0 + \mathbf{W}_x\mathbf{x}_1 + \mathbf{b}) \rightarrow \hat{y}_1 = \text{argmax}\mathbf{W}_o\mathbf{h}_1^\intercal$
3. $\mathbf{h}_2 = \sigma(\mathbf{W}_h\mathbf{h}_1 + \mathbf{W}_x\mathbf{x}_2 + \mathbf{b}) \rightarrow \hat{y}_2 = \text{argmax}\mathbf{W}_o\mathbf{h}_2^\intercal$
4. $\mathbf{h}_3 = \sigma(\mathbf{W}_h\mathbf{h}_2 + \mathbf{W}_x\mathbf{x}_3 + \mathbf{b}) \rightarrow \hat{y}_3 = \text{argmax}\mathbf{W}_o\mathbf{h}_3^\intercal$

Parameters:

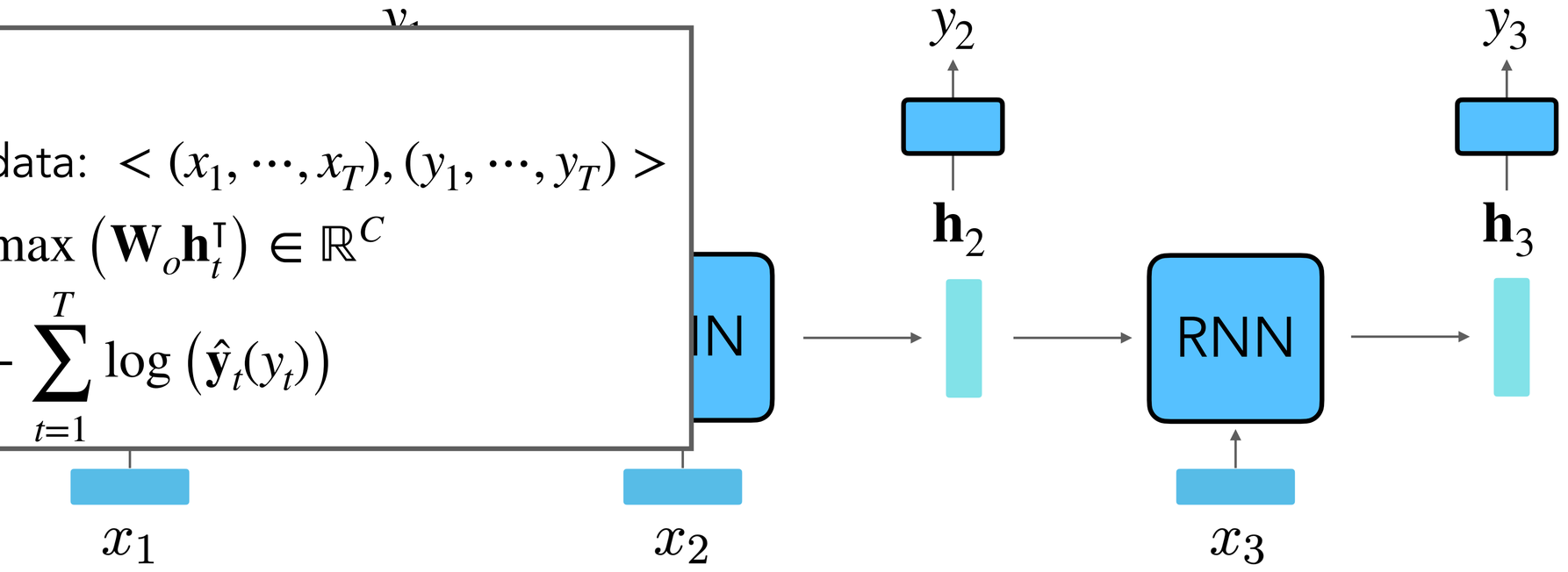
$\mathbf{E} \in \mathbf{R}^{|V| \times d_{\text{in}}}$,

$\mathbf{W}_h \in \mathbf{R}^{d \times d}, \mathbf{W}_x \in \mathbf{R}^{d \times d_{\text{in}}}$,

$\mathbf{b} \in \mathbf{R}^d, \mathbf{W}_o \in \mathbf{R}^{C \times d}$

Training:

- Training data: $< (x_1, \cdots, x_T), (y_1, \cdots, y_T) >$
- $\hat{\mathbf{y}}_t = \text{softmax}\left(\mathbf{W}_o\mathbf{h}_t^\intercal\right) \in \mathbb{R}^C$
- $\mathscr{L}(\theta) = -\sum_{t=1}^{T} \log\left(\hat{\mathbf{y}}_t(y_t)\right)$



Recurrent Neural Network: (3) Language modeling

1. $\mathbf{h}_0$: Initial hidden state (usually zeros)
2. $\mathbf{h}_1 = \sigma(\mathbf{W}_h\mathbf{h}_0 + \mathbf{W}_x\mathbf{x}_1 + \mathbf{b}) \rightarrow \hat{y}_1 = \text{argmax}\mathbf{W}_o\mathbf{h}_1^\intercal$
3. $\mathbf{h}_2 = \sigma(\mathbf{W}_h\mathbf{h}_1 + \mathbf{W}_x\mathbf{x}_2 + \mathbf{b}) \rightarrow \hat{y}_2 = \text{argmax}\mathbf{W}_o\mathbf{h}_2^\intercal$
4. $\mathbf{h}_3 = \sigma(\mathbf{W}_h\mathbf{h}_2 + \mathbf{W}_x\mathbf{x}_3 + \mathbf{b}) \rightarrow \hat{y}_3 = \text{argmax}\mathbf{W}_o\mathbf{h}_3^\intercal$

Parameters:

$\mathbf{E} \in \mathbf{R}^{|V| \times d_{\text{in}}}$,

$\mathbf{W}_h \in \mathbf{R}^{d \times d}, \mathbf{W}_x \in \mathbf{R}^{d \times d_{\text{in}}}$,

$\mathbf{b} \in \mathbf{R}^d, \mathbf{W}_o \in \mathbf{R}^{|V| \times d}$

Training:

- Training data: $x_1, x_2, \cdots, x_T$
- This is equivalent to having supervised training datasets
  $< (x_1), (x_2) > , < (x_1, x_2), (x_3) > , < (x_1, x_2, x_3), (x_4) > \cdots, < (x_1, \cdots, x_{T-1}), (x_T) >$
- $\hat{\mathbf{y}}_t = \text{softmax}\left(\mathbf{W}_o\mathbf{h}_t^\intercal\right) \in \mathbb{R}^{|V|}$
- $\mathscr{L}(\theta) = -\sum_{t=1}^{T-1} \log\left(\hat{\mathbf{y}}_t(x_{t+1})\right)$

Recurrent Neural Network: (3) Language modeling

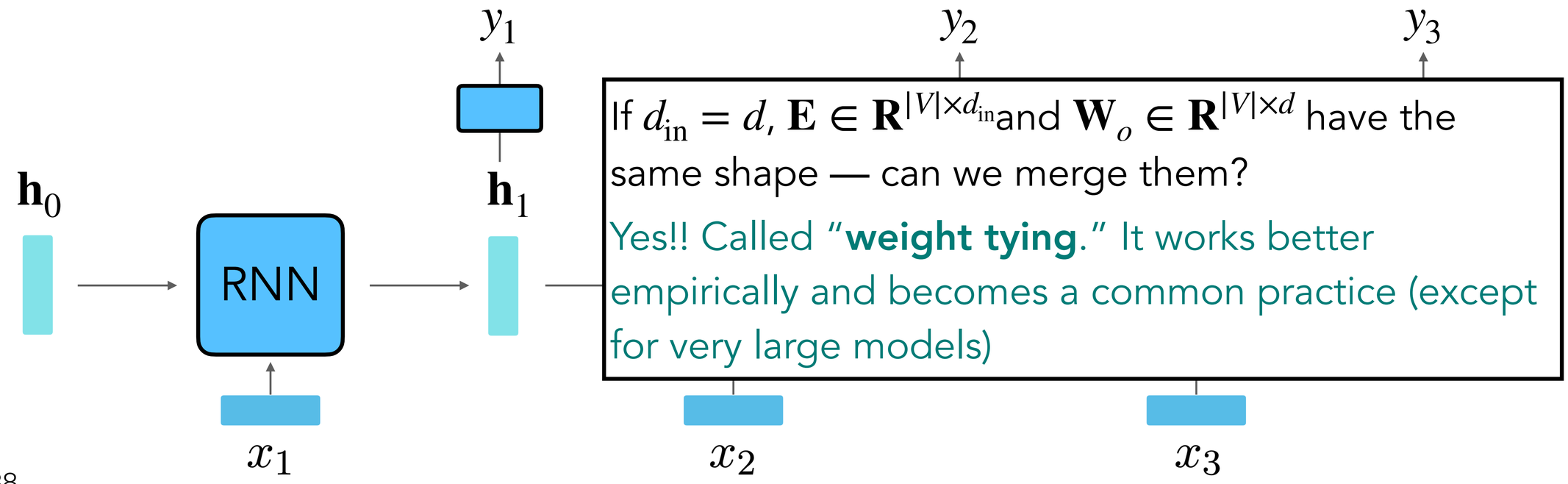
1. $\mathbf{h}_0$: Initial hidden state (usually zeros)
2. $\mathbf{h}_1 = \sigma(\mathbf{W}_h\mathbf{h}_0 + \mathbf{W}_x\mathbf{x}_1 + \mathbf{b}) \rightarrow \hat{y}_1 = \text{argmax}\mathbf{W}_o\mathbf{h}_1^\intercal$
3. $\mathbf{h}_2 = \sigma(\mathbf{W}_h\mathbf{h}_1 + \mathbf{W}_x\mathbf{x}_2 + \mathbf{b}) \rightarrow \hat{y}_2 = \text{argmax}\mathbf{W}_o\mathbf{h}_2^\intercal$
4. $\mathbf{h}_3 = \sigma(\mathbf{W}_h\mathbf{h}_2 + \mathbf{W}_x\mathbf{x}_3 + \mathbf{b}) \rightarrow \hat{y}_3 = \text{argmax}\mathbf{W}_o\mathbf{h}_3^\intercal$

Parameters:

$\mathbf{E} \in \mathbf{R}^{|V| \times d_{\text{in}}}$,

$\mathbf{W}_h \in \mathbf{R}^{d \times d}, \mathbf{W}_x \in \mathbf{R}^{d \times d_{\text{in}}}$,

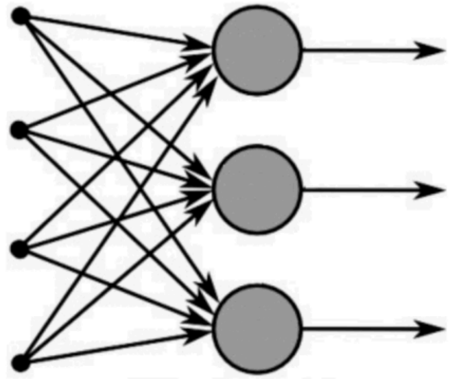
$\mathbf{b} \in \mathbf{R}^d, \mathbf{W}_o \in \mathbf{R}^{|V| \times d}$



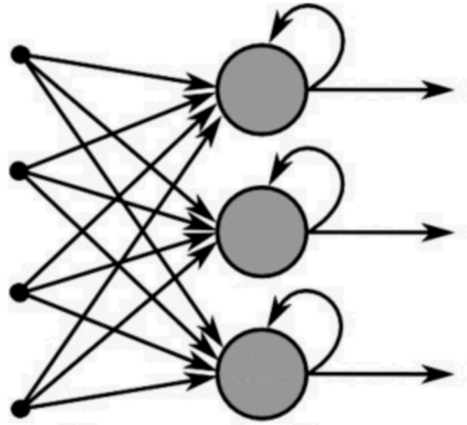
If $d_{\text{in}} = d$, $\mathbf{E} \in \mathbf{R}^{|V| \times d_{\text{in}}}$ and $\mathbf{W}_o \in \mathbf{R}^{|V| \times d}$ have the same shape — can we merge them?

Yes!! Called "**weight tying**." It works better empirically and becomes a common practice (except for very large models)

Feedforward NNs vs. RNNs



Feed-Forward Neural Network



Recurrent Neural Network

$$\mathbf{h}_1 = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \in \mathbb{R}^{h_1}$$

$$\mathbf{h}_2 = g(\mathbf{W}^{(2)}\mathbf{h}_1 + \mathbf{b}^{(2)}) \in \mathbb{R}^{h_2}$$

$$\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \in \mathbb{R}^{h}$$

RNNs: Results

On the Penn Treebank (PTB) dataset
Metric: perplexity

KN5: Kneser-Ney 5-gram

Model	Individual
KN5	141.2
KN5 + cache	125.7
Feedforward NNLM	140.2
Log-bilinear NNLM	144.5
Syntactical NNLM	131.3
Recurrent NNLM	124.7
RNN-LDA LM	113.7

(Mikolov and Zweig, 2012): Context dependent recurrent neural network language model

RNNs: Results

On the Penn Treebank (PTB) dataset
Metric: perplexity

Model	#Param	Validation	Test
Mikolov & Zweig (2012) – RNN-LDA + KN-5 + cache	9M‡	-	92.0
Zaremba et al. (2014) – LSTM	20M	86.2	82.7
Gal & Ghahramani (2016) – Variational LSTM (MC)	20M	-	78.6
Kim et al. (2016) – CharCNN	19M	-	78.9
Merity et al. (2016) – Pointer Sentinel-LSTM	21M	72.4	70.9
Grave et al. (2016) – LSTM + continuous cache pointer†	-	-	72.1
Inan et al. (2016) – Tied Variational LSTM + augmented loss	24M	75.7	73.2
Zilly et al. (2016) – Variational RHN	23M	67.9	65.4
Zoph & Le (2016) – NAS Cell	25M	-	64.0
Melis et al. (2017) – 2-layer skip connection LSTM	24M	60.9	58.3
Merity et al. (2017) – AWD-LSTM w/o finetune	24M	60.7	58.8
Merity et al. (2017) – AWD-LSTM	24M	60.0	57.3
Ours – AWD-LSTM-MoS w/o finetune	22M	58.08	55.97
Ours – AWD-LSTM-MoS	22M	**56.54**	**54.44**
Merity et al. (2017) – AWD-LSTM + continuous cache pointer†	24M	53.9	52.8
Krause et al. (2017) – AWD-LSTM + dynamic evaluation†	24M	51.6	51.1
Ours – AWD-LSTM-MoS + dynamic evaluation†	22M	**48.33**	**47.69**

(Yang et al, 2018): Breaking the Softmax Bottleneck: A High-Rank RNN Language Model

RNNs: Pros and cons

Advantages:

- Can process any length input
- Computation for step t can (in theory) use information from many steps back
- Model size doesn't increase for longer input context

Disadvantages:

- Recurrent computation is slow (can't parallelize) Transformers can!
- In practice, difficult to access information from many steps back (optimization issue)
  - We will see some advanced RNNs (e.g., LSTMs, GRUs, SSMs)

RNNs in Code (1/2)

```python
class RNNCell(torch.nn.Module):

    def __init__(self, d_input, d_hidden):
        super().__init__()
        self.d_input = d_input
        self.d_hidden = d_hidden
        self.Wh = torch.nn.Linear(d_hidden, d_hidden, bias=False)
        self.Wx = torch.nn.Linear(d_input, d_hidden, bias=True)
        self.activation = torch.nn.Tanh()

    def __call__(self, x, h):
        h = self.activation(self.Wh(h) + self.Wx(h))
        return h
```

RNNs in Code (2/2)

```
class RNN(torch.nn.Module):

    def __init__(self, vocab_size, d_input, d_hidden, n_labels):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, d_input)
        self.rnn = RNNCell(d_input, d_hidden)
        self.output = nn.Linear(d_hidden, n_labels)
        self.d_hidden = d_hidden

    def __call__(self, x, hidden=None):
        if hidden is None:
            hidden = self.init_hidden(x.size(0))

        x = self.embedding(x)

        outs = []
        for i in range(x.size(1)):
            hidden = self.rnn(x[:, i:i+1], hidden)
            out = self.output(hidden)
            outs.append(out)

        outs = torch.cat(outs, dim=1)
        return outs, hidden

    def init_hidden(self, batch_size):
        return torch.zeros(batch_size, 1, self.hidden_size)
```

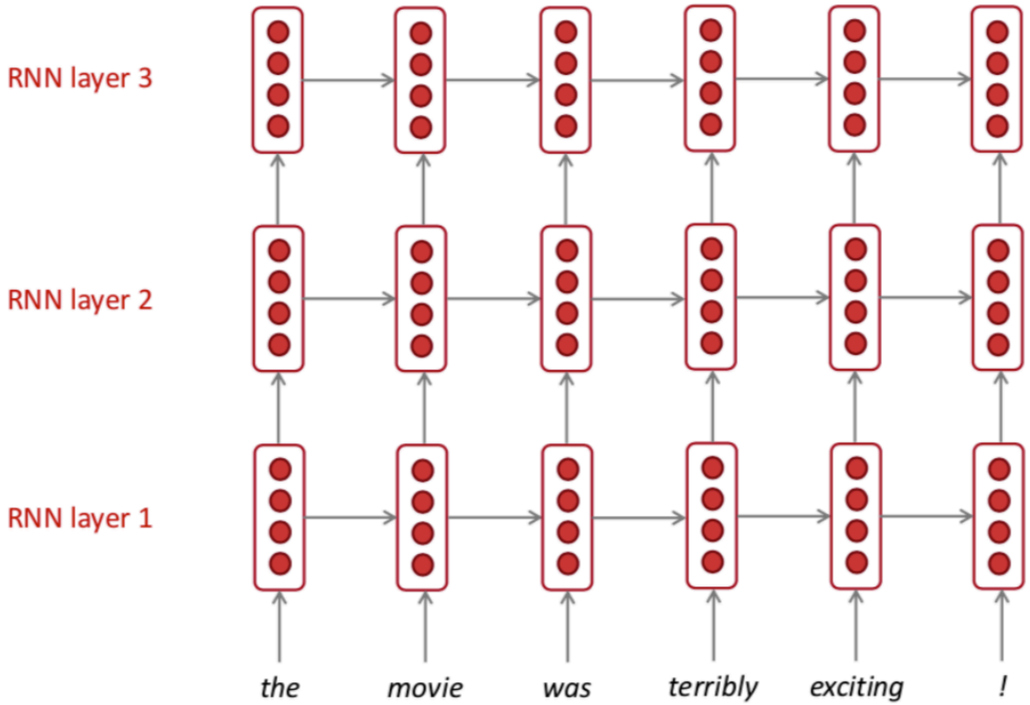
The Unreasonable Effectiveness of Recurrent Neural Networks

May 21, 2015

You can train an RNN-LM on any kind of text, then generate text in that style.

```
\begin{proof}
We may assume that $\mathcal{I}$ is an abelian sheaf on $\mathcal{C}$.
\item Given a morphism $\Delta : \mathcal{F} \to \mathcal{I}$
is an injective and let $\mathfrak q$ be an abelian sheaf on $X$.
Let $\mathcal{F}$ be a fibered complex. Let $\mathcal{F}$ be a category.
\begin{enumerate}
\item \hyperref[setain-construction-phantom]{Lemma}
\label{lemma-characterize-quasi-finite}
Let $\mathcal{F}$ be an abelian quasi-coherent sheaf on $\mathcal{C}$.
Let $\mathcal{F}$ be a coherent $\mathcal{O}_X$-module. Then
$\mathcal{F}$ is an abelian catenary over $\mathcal{C}$.
\item The following are equivalent
\begin{enumerate}
\item $\mathcal{F}$ is an $\mathcal{O}_X$-module.
\end{lemma}
```

Multi-layer RNNs



The hidden states from RNN layer i are the inputs to RNN layer i + 1

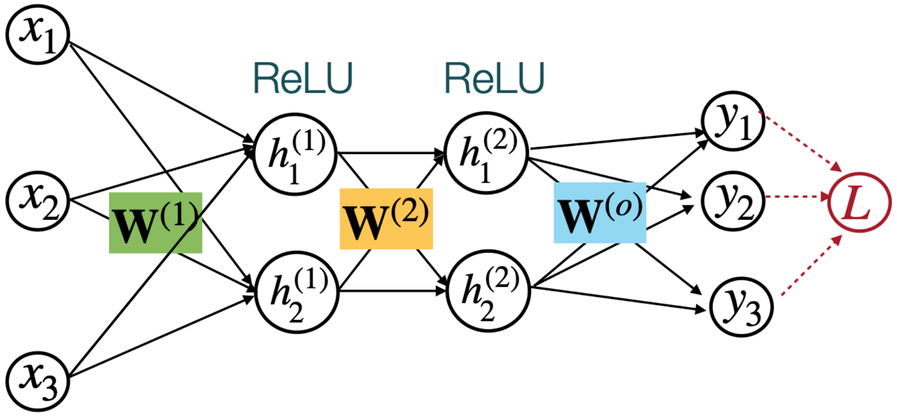
- In practice, using 2 to 4 layers is common (usually better than 1 layer)
- Transformer networks can be up to 24 layers with lots of skip-connections

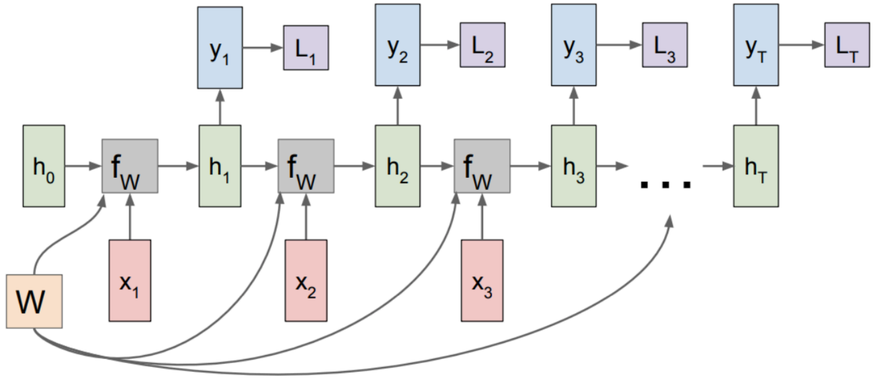
Issues of RNN Training

Training of RNNs

Backword pass:

- Backpropagation? Yes, but not that simple





The algorithm is called **Backpropagation Through Time (BPTT)**

Backpropagation through time

Recall:

$$\mathbf{h}_1 = \sigma(\mathbf{W}_h\mathbf{h}_0 + \mathbf{W}_x\mathbf{x}_1 + \mathbf{b}), \text{ where } \mathbf{x}_1 = \mathbf{E}(x_1)$$
$$\mathbf{h}_2 = \sigma(\mathbf{W}_h\mathbf{h}_1 + \mathbf{W}_x\mathbf{x}_2 + \mathbf{b}), \text{ where } \mathbf{x}_2 = \mathbf{E}(x_2)$$
$$\mathbf{h}_3 = \sigma(\mathbf{W}_h\mathbf{h}_2 + \mathbf{W}_x\mathbf{x}_3 + \mathbf{b}), \text{ where } \mathbf{x}_3 = \mathbf{E}(x_3)$$

$$\hat{\mathbf{y}} = \text{softmax}\left(\mathbf{W}_o\mathbf{h}_3^\intercal\right) \in \mathbb{R}^C$$
$$\mathscr{L}(\theta) = -\log\left(\hat{\mathbf{y}}(c)\right)$$

In order to do backpropagation, you need: $\frac{\partial \mathscr{L}}{\partial \mathbf{W}_h}$

$$\frac{\partial \mathscr{L}}{\partial \mathbf{W}_h} = \frac{\partial \mathscr{L}}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{W}_h} + \frac{\partial \mathscr{L}}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2}\frac{\partial \mathbf{h}_2}{\partial \mathbf{W}_h} + \frac{\partial \mathscr{L}}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2}\frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1}\frac{\partial \mathbf{h}_1}{\partial \mathbf{W}_h}$$

More generally:

$$\boxed{\frac{\partial \mathscr{L}}{\partial \mathbf{W}_h} = -\frac{1}{n}\sum_{t=1}^{T}\sum_{k=1}^{t}\frac{\partial \mathscr{L}}{\partial \mathbf{h}_t}\left(\prod_{j=k+1}^{t}\frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}}\right)}$$

If k and t are far away, the gradients can grow/shrink exponentially
(called the gradient exploding or gradient vanishing problem)

Backpropagation through time

Quiz: What will happen if the gradients become too large or too small?

(a) If too large, the model will become difficult to converge
(b) If too small, the model can’t capture long-term dependencies
(c) If too small, the model may capture a wrong recent dependency
(d) All of the above

Answer is (d). All of these are correct :)

Backpropagation through time: Solution

One solution for gradient exploding is called gradient clipping — if the norm of the gradient is greater than some threshold, scale it down before applying SGD update.

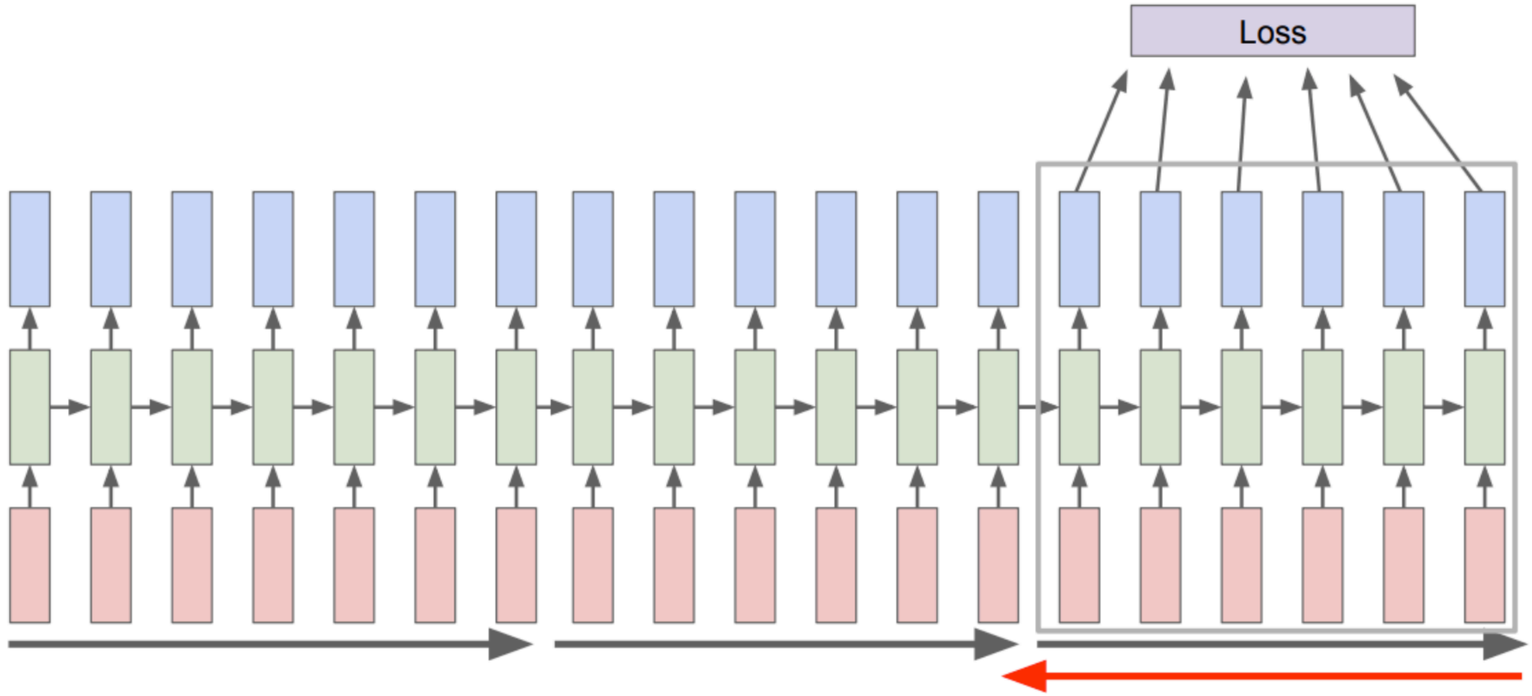
Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if

Intuition: take a step in the same direction but a smaller step!

Gradient vanishing is a harder problem to solve.

Truncated backpropagation through time



- Run forward and backward through chunks of the sequence instead of whole sequence
- Carry hidden states forward in time forever, but only back-propagate for some smaller number of steps

Advanced RNN variants
(“Gated” architectures)

Advanced RNN variants: Overview

Vanilla RNN

$$\mathbf{h}_t = \sigma(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t + \mathbf{b})$$

Long Short-Term Memory (LSTMs)

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \mathbf{h}_{t-1} + \mathbf{U}_i \mathbf{x}_t + \mathbf{b}_i) \in \mathbb{R}^d$$
$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{h}_{t-1} + \mathbf{U}_f \mathbf{x}_t + \mathbf{b}_f) \in \mathbb{R}^d$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{h}_{t-1} + \mathbf{U}_o \mathbf{x}_t + \mathbf{b}_o) \in \mathbb{R}^d$$
$$\mathbf{g}_t = \sigma(\mathbf{W}_g \mathbf{h}_{t-1} + \mathbf{U}_g \mathbf{x}_t + \mathbf{b}_g) \in \mathbb{R}^d$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh\left(\mathbf{c}_t\right)$$

Gated Recurrent Unit (GRU)

$$\mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{h}_{t-1} + \mathbf{U}_r \mathbf{x}_t + \mathbf{b}_r) \in \mathbb{R}^d$$
$$\mathbf{z}_t = \sigma(\mathbf{W}_z \mathbf{h}_{t-1} + \mathbf{U}_z \mathbf{x}_t + \mathbf{b}_z) \in \mathbb{R}^d$$

$$\tilde{\mathbf{h}}_t = \tanh\left(\mathbf{W}\left(\mathbf{r}_t \odot \mathbf{h}_{t-1}\right) + \mathbf{U}\mathbf{x}_t + \mathbf{b}\right)$$
$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t$$

Long Short-Term Memory RNNs (LSTMs)

A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the **vanishing gradients problem**.

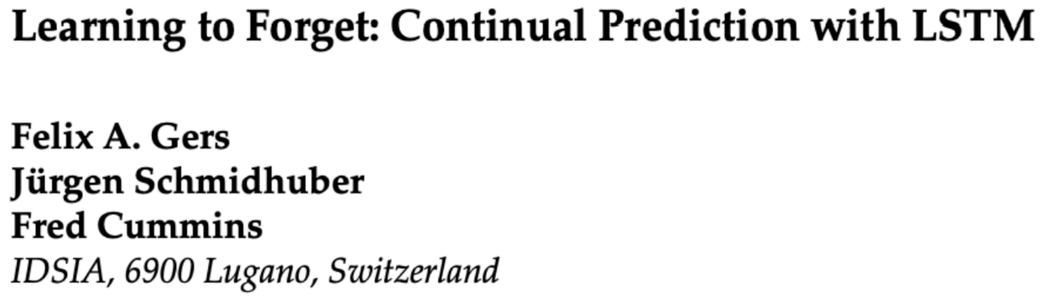
- Everyone cites that paper but really a crucial part of the modern LSTM is from Gers et al. (2000)

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



Learning to Forget: Continual Prediction with LSTM

Felix A. Gers
Jürgen Schmidhuber
Fred Cummins
IDSIA, 6900 Lugano, Switzerland

Recap: Vanishing gradients

Recall:

$$\mathbf{h}_1 = \sigma(\mathbf{W}_h\mathbf{h}_0 + \mathbf{W}_x\mathbf{x}_1 + \mathbf{b}), \text{ where } \mathbf{x}_1 = \mathbf{E}(x_1)$$
$$\mathbf{h}_2 = \sigma(\mathbf{W}_h\mathbf{h}_1 + \mathbf{W}_x\mathbf{x}_2 + \mathbf{b}), \text{ where } \mathbf{x}_2 = \mathbf{E}(x_2)$$
$$\mathbf{h}_3 = \sigma(\mathbf{W}_h\mathbf{h}_2 + \mathbf{W}_x\mathbf{x}_3 + \mathbf{b}), \text{ where } \mathbf{x}_3 = \mathbf{E}(x_3)$$

$$\hat{\mathbf{y}} = \text{softmax}\left(\mathbf{W}_o\mathbf{h}_3^\intercal\right) \in \mathbb{R}^C$$
$$\mathscr{L}(\theta) = -\log\left(\hat{\mathbf{y}}(c)\right)$$

When these are small, the gradient signals get smaller and smaller as it back propagates further

In order to do backpropagation, you need:

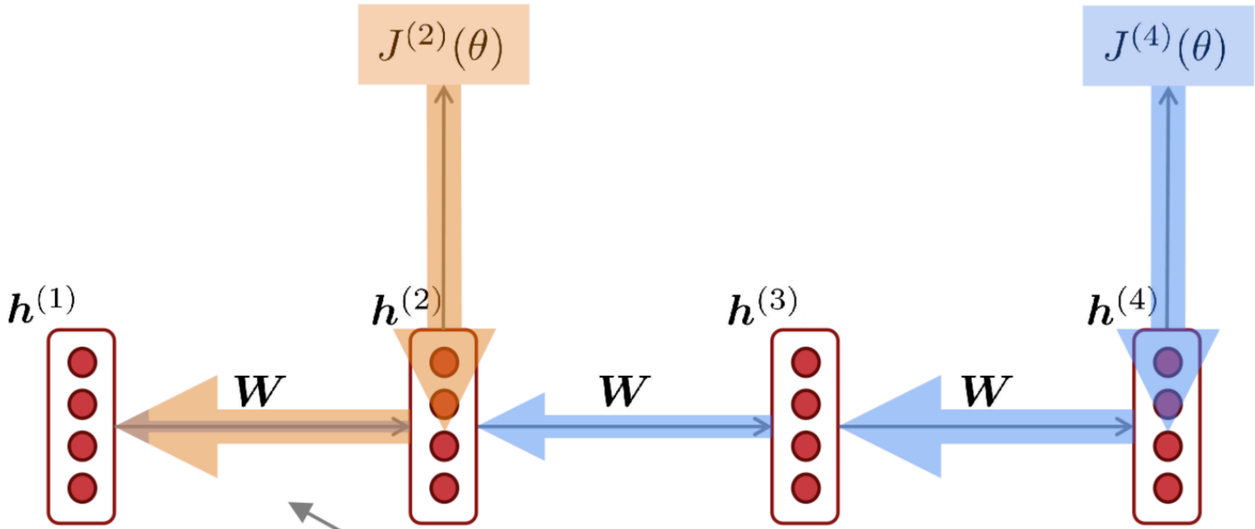
$$\frac{\partial \mathscr{L}}{\partial \mathbf{W}} = \frac{\partial \mathscr{L}}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{W}} + \frac{\partial \mathscr{L}}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2}\frac{\partial \mathbf{h}_2}{\partial \mathbf{W}} + \frac{\partial \mathscr{L}}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2}\frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1}\frac{\partial \mathbf{h}_1}{\partial \mathbf{W}}$$

More generally:

$$\boxed{\frac{\partial \mathscr{L}}{\partial \mathbf{W}} = -\frac{1}{n}\sum_{t=1}^{T}\sum_{k=1}^{t}\frac{\partial \mathscr{L}}{\partial \mathbf{h}_t}\left(\prod_{j=k+1}^{t}\frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}}\right)}$$

If k and t are far away, the gradients can grow/shrink exponentially (called the gradient exploding or gradient vanishing problem)

Recap: Vanishing gradients



Gradient signal from far away is lost because it's much smaller than gradient signal from close-by.

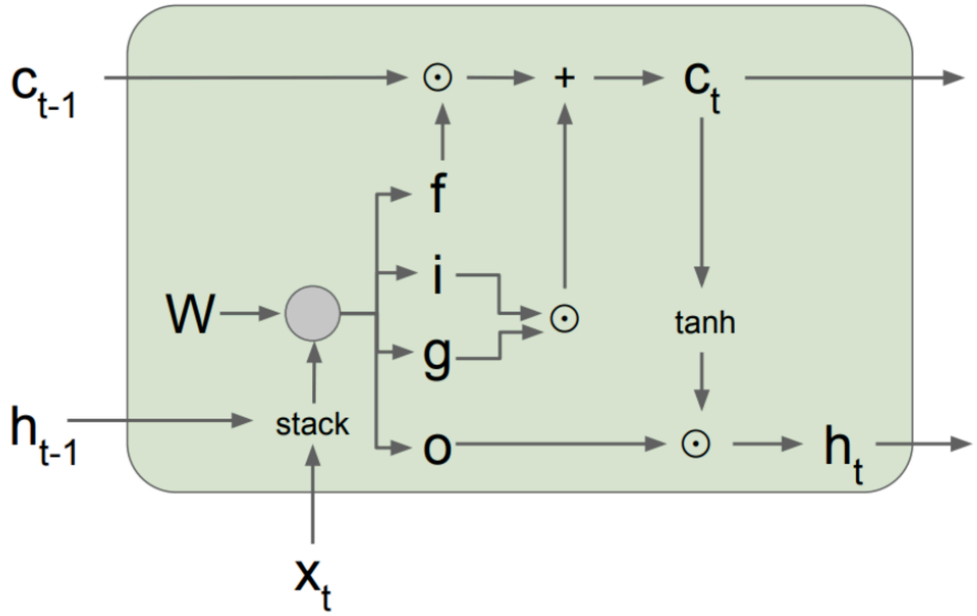
So, model weights are basically updated only with respect to near effects, not long-term effects.

Slide credit: Chris Manning

LSTMs: The intuition

Key idea: adding a "**cell**" state as a long-term memory wire, updated with an "**additive**" updates (instead of multiplication) and and using "**gates**" to control how much information to add/erase

- We write to/erase information from $\mathbf{c}_t$ after each step t
- We read $\mathbf{h}_t$ from $\mathbf{c}_t$



LSTMs: The formulation

Input gate **(how much to write)**

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \mathbf{h}_{t-1} + \mathbf{U}_i \mathbf{x}_t + \mathbf{b}_i) \in \mathbb{R}^d$$

Forget gate **(how much to erase)**

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{h}_{t-1} + \mathbf{U}_f \mathbf{x}_t + \mathbf{b}_f) \in \mathbb{R}^d$$

Output gate **(how much to reveal)**

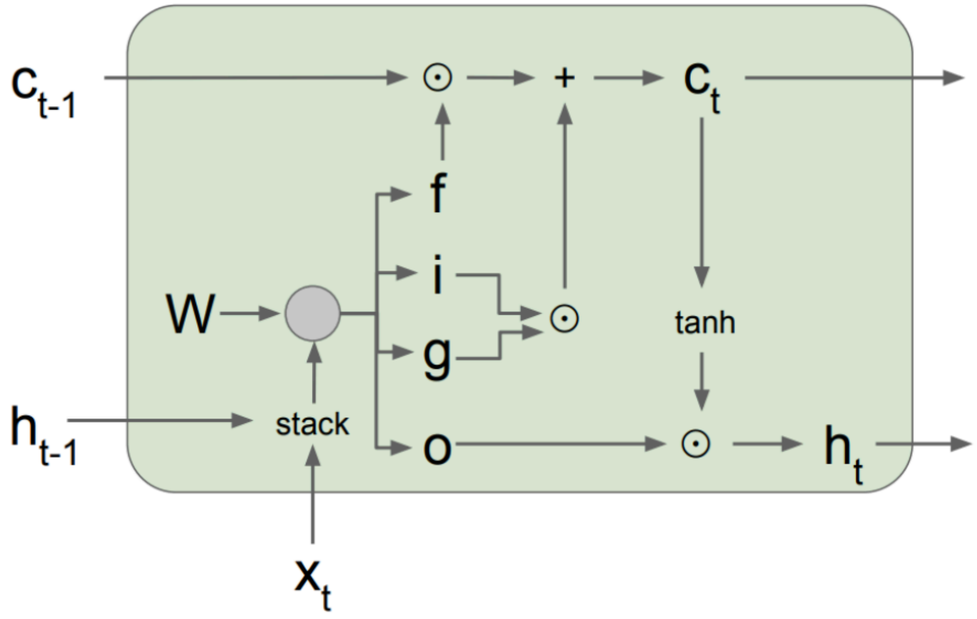
$$\mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{h}_{t-1} + \mathbf{U}_o \mathbf{x}_t + \mathbf{b}_o) \in \mathbb{R}^d$$

New memory cell **(what to write)**

$$\mathbf{g}_t = \sigma(\mathbf{W}_g \mathbf{h}_{t-1} + \mathbf{U}_g \mathbf{x}_t + \mathbf{b}_g) \in \mathbb{R}^d$$

Final memory cell: $\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t$

Finally hidden cell: $\mathbf{h}_t = \mathbf{o}_t \odot \tanh\left(\mathbf{c}_t\right)$



LSTMs: The formulation

Input gate **(how much to write)**

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \mathbf{h}_{t-1} + \mathbf{U}_i \mathbf{x}_t + \mathbf{b}_i) \in \mathbb{R}^d$$

Forget gate **(how much to erase)**

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{h}_{t-1} + \mathbf{U}_f \mathbf{x}_t + \mathbf{b}_f) \in \mathbb{R}^d$$

Output gate **(how much to reveal)**

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{h}_{t-1} + \mathbf{U}_o \mathbf{x}_t + \mathbf{b}_o) \in \mathbb{R}^d$$

New memory cell **(what to write)**

$$\mathbf{g}_t = \sigma(\mathbf{W}_g \mathbf{h}_{t-1} + \mathbf{U}_g \mathbf{x}_t + \mathbf{b}_g) \in \mathbb{R}^d$$

Final memory call: $\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t$

Finally hidden cell: $\mathbf{h}_t = \mathbf{o}_t \odot \tanh\left(\mathbf{c}_t\right)$

Quick quiz: How many parameters?

Assuming embedding matrix $\mathbf{E} \in \mathbf{R}^{|V| \times d_{\text{in}}}$,

output matrix $\mathbf{W}_o \in \mathbf{R}^{C \times d}$

Answer:

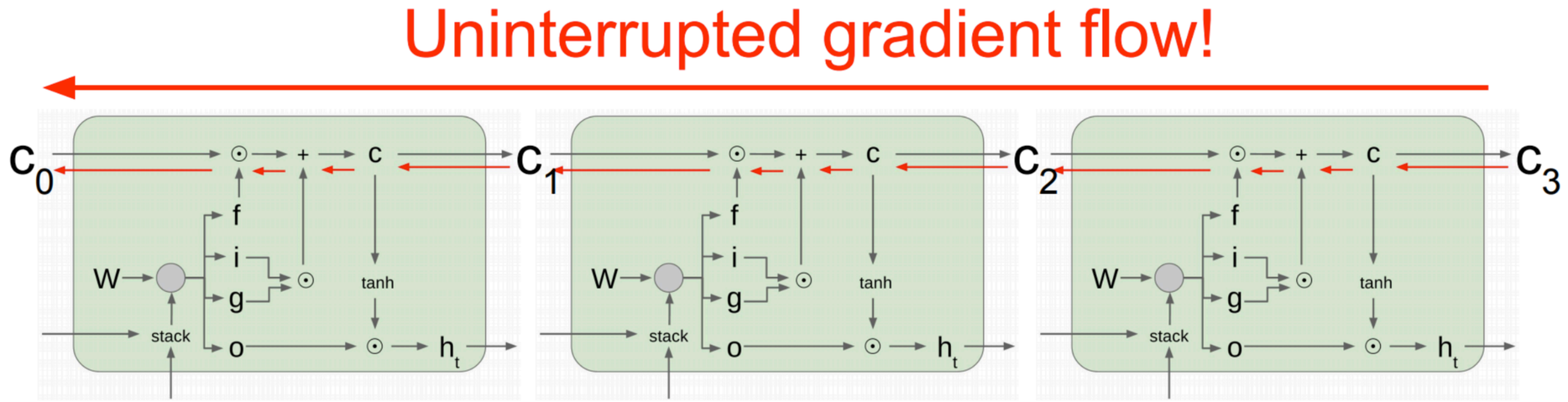
$$|V| d_{\text{in}} + 4\left(d d_{\text{in}} + d^2 + d\right) + Cd$$

In RNNs, $|V| d_{\text{in}} + d d_{\text{in}} + d^2 + d + Cd$

Because it only had one linear computation —

$$\mathbf{h}_t = \sigma(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t + \mathbf{b})$$

LSTMs: The formulation



- LSTM doesn't guarantee that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies
- LSTMs were invented in 1997 but finally got working from 2013-2015.
- These ideas influenced later designs such as "residual connection".

Gated Recurrent Units (GRUs)

Introduced by Kyunghyun Cho et al. in 2014:

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

Kyunghyun Cho
Bart van Merriënboer **Caglar Gulcehre** **Dzmitry Bahdanau**
Université de Montréal Jacobs University, Germany
firstname.lastname@umontreal.ca d.bahdanau@jacobs-university.de

Fethi Bougares **Holger Schwenk** **Yoshua Bengio**
Université du Maine, France Université de Montréal, CIFAR Senior Fellow
firstname.lastname@lium.univ-lemans.fr find.me@on.the.web

Simplified 3 gates to 2 gates: **reset** gate and **update** gate, without an explicit cell state

Gated Recurrent Units (GRUs)

Reset gate **(how much to reset)**

$$\mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{h}_{t-1} + \mathbf{U}_r \mathbf{x}_t + \mathbf{b}_r) \in \mathbb{R}^d$$

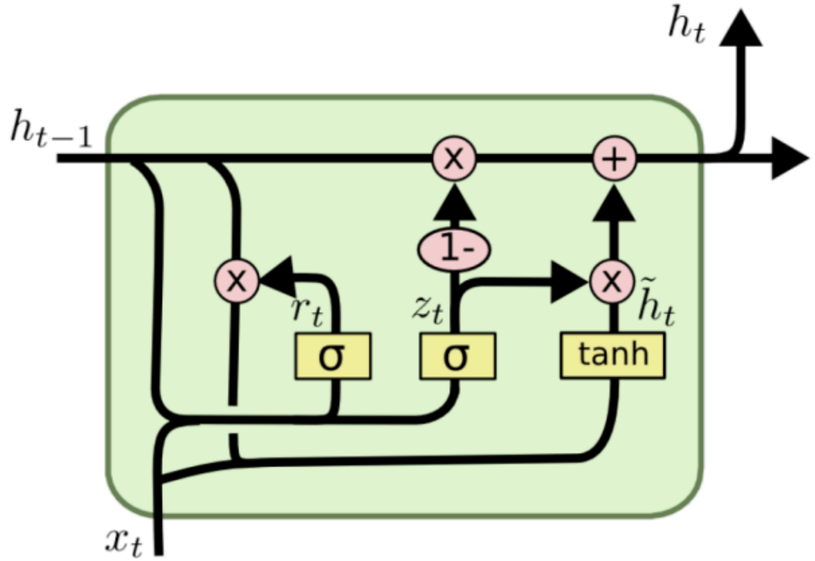
Update gate **(how much to update)**

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \mathbf{h}_{t-1} + \mathbf{U}_z \mathbf{x}_t + \mathbf{b}_z) \in \mathbb{R}^d$$

New hidden state

$$\tilde{\mathbf{h}}_t = \tanh\left(\mathbf{W}\left(\mathbf{r}_t \odot \mathbf{h}_{t-1}\right) + \mathbf{U}\mathbf{x}_t + \mathbf{b}\right)$$

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



Gated Recurrent Units (GRUs)

Reset gate **(how much to reset)**

$$\mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{h}_{t-1} + \mathbf{U}_r \mathbf{x}_t + \mathbf{b}_r) \in \mathbb{R}^d$$

Update gate **(how much to update)**

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \mathbf{h}_{t-1} + \mathbf{U}_z \mathbf{x}_t + \mathbf{b}_z) \in \mathbb{R}^d$$

New hidden state

$$\tilde{\mathbf{h}}_t = \tanh\left(\mathbf{W}\left(\mathbf{r}_t \odot \mathbf{h}_{t-1}\right) + \mathbf{U}\mathbf{x}_t + \mathbf{b}\right)$$

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

Quick quiz: How many parameters?
Assuming embedding matrix $\mathbf{E} \in \mathbf{R}^{|V| \times d_{\text{in}}}$,
output matrix $\mathbf{W}_o \in \mathbf{R}^{C \times d}$

Answer:

$$|V| d_{\text{in}} + 3\left(d d_{\text{in}} + d^2 + d\right) + Cd$$

In RNNs, $|V| d_{\text{in}} + d d_{\text{in}} + d^2 + d + Cd$

In LSTMs, $|V| d_{\text{in}} + 4\left(d d_{\text{in}} + d^2 + d\right) + Cd$

Advanced RNN variants: Overview

Vanilla RNN

$$\mathbf{h}_t = \sigma(\mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{W}_x\mathbf{x}_t + \mathbf{b})$$

Long Short-Term Memory (LSTMs)

$$\mathbf{i}_t = \sigma(\mathbf{W}_i\mathbf{h}_{t-1} + \mathbf{U}_i\mathbf{x}_t + \mathbf{b}_i) \in \mathbb{R}^d$$
$$\mathbf{f}_t = \sigma(\mathbf{W}_f\mathbf{h}_{t-1} + \mathbf{U}_f\mathbf{x}_t + \mathbf{b}_f) \in \mathbb{R}^d$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o\mathbf{h}_{t-1} + \mathbf{U}_o\mathbf{x}_t + \mathbf{b}_o) \in \mathbb{R}^d$$
$$\mathbf{g}_t = \sigma(\mathbf{W}_g\mathbf{h}_{t-1} + \mathbf{U}_g\mathbf{x}_t + \mathbf{b}_g) \in \mathbb{R}^d$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh\left(\mathbf{c}_t\right)$$

Gated Recurrent Unit (GRU)

$$\mathbf{r}_t = \sigma(\mathbf{W}_r\mathbf{h}_{t-1} + \mathbf{U}_r\mathbf{x}_t + \mathbf{b}_r) \in \mathbb{R}^d$$
$$\mathbf{z}_t = \sigma(\mathbf{W}_z\mathbf{h}_{t-1} + \mathbf{U}_z\mathbf{x}_t + \mathbf{b}_z) \in \mathbb{R}^d$$

$$\tilde{\mathbf{h}}_t = \tanh\left(\mathbf{W}\left(\mathbf{r}_t \odot \mathbf{h}_{t-1}\right) + \mathbf{U}\mathbf{x}_t + \mathbf{b}\right)$$
$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t$$

Comparison of LSTMs and GRUs

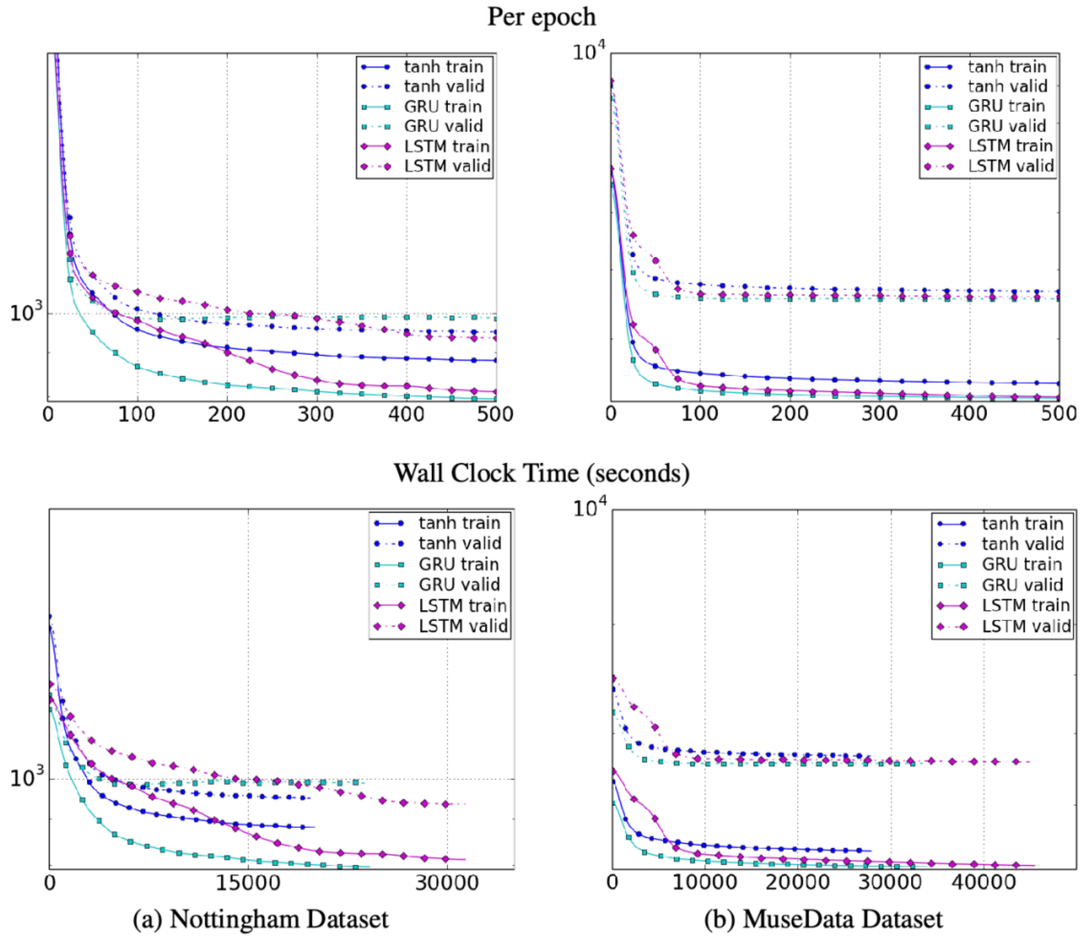
Quiz: Let's compare LSTMs and GRUs. Which of the following statements is correct?

(a) GRUs can be trained faster
(b) In theory LSTMs can capture long-term dependencies better
(c) LSTMs have a controlled exposure of memory content while GRUs don't
(d) All of the above

Answer is (d). All of these are correct :)

LSTMs vs. GRUs

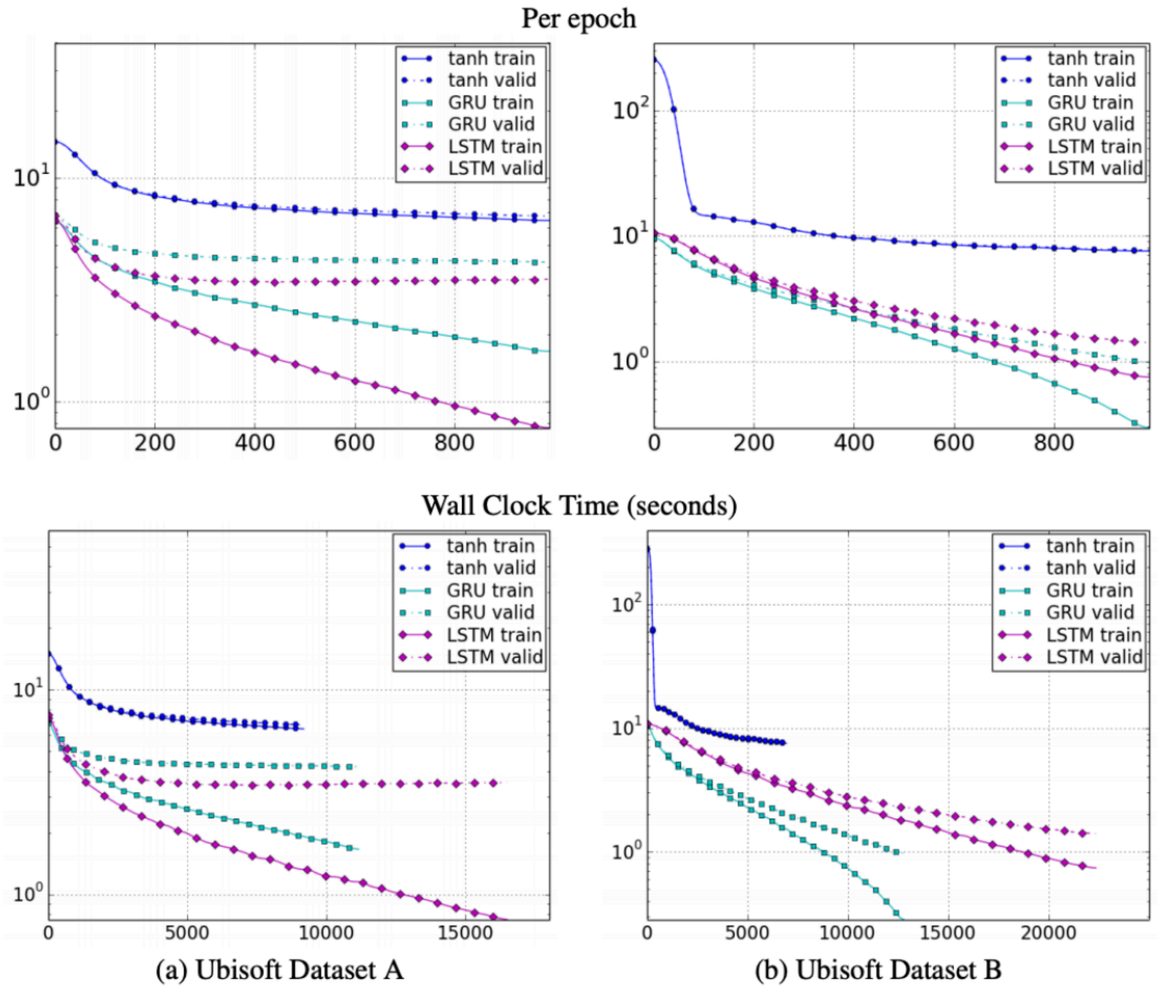
Music modeling



(Chung et al, 2014): Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling

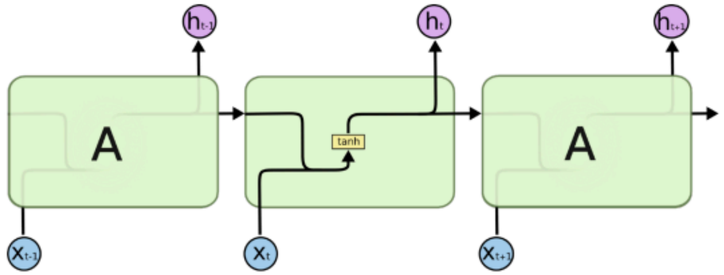
LSTMs vs. GRUs

Speech signal modeling



(Chung et al, 2014): Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling

Modern RNNs: State-space models



N-D $h_t = \boldsymbol{A_t} h_{t-1} + \boldsymbol{B_t} x_t$ 1-D

1-D $y_t = \boldsymbol{C_t} h_t$

- Much simpler recurrent update (linear)
- Long-range with large hidden units
- Efficiency: parallelizable on GPUs

(Gu et al, 2021): Efficiently Modeling Long Sequences with Structured State Spaces

Questions?

Acknowledgement

Princeton COS 484 by Danqi Chen, Tri Dao, Vikram Ramaswamy
CMU CS11-711 Advanced NLP by Graham Neubig & Sean Welleck