Deep Learning for Language: Recurrent Neural Networks

Robin Jia USC CSCI 467, Spring 2025 March 4, 2025

Peculiarities of language data

- Peculiarity #1: Text is not a numerical format
 - Feature vector = list of numbers
 - Image = 3xWxH grid of pixel brightness values
 - Text = sequence of words, not numbers
- Peculiarity #2: Text is variable sized
 - Feature vectors are always the same size for different examples
 - Images can be cropped/rescaled to be the same size for all examples
 - Text: Different examples have different # of words

Feeding Words to a Neural Network

- Peculiarity #1: Words are not numerical
- Solution: Learn word vectors, feed word vector of each word to model!
- Original input: T words
- Vector input: T vectors, each of size d

Text input:	Α	Z 00	elephant
Vector input:	-0.4	2.1	2.1
	1.4	-1.4	-1.3
	-1.2	3.2	0.3

Word <i>w</i>	Vector v _w
A	[-0.4, 1.4, -1.2]
Aardvark	[2.2, -1.8, 0.6]
Airport	[0.7, 0.3, 3.1]
•••	
Elephant	[2.1, -1.3, 0.3
Zoo	[2.1, -1.4, 3.2]

RNN "Building Blocks"

(6) Word Vector Layer

- Input w: A word (from our vocabulary)
 - Can also input list of words
- Output: A vector of length d
 - If input is many words, output is list of vectors for each word
- Formula: Return word_vecs[w]
- Parameters:
 - For each word w in vocabulary, there is a word vector parameter $v_{\rm w}$ of shape d
 - Think of this as a dictionary called word_vecs, where the keys are words & values are learned parameter vectors
 - Can initialize using word2vec, or randomly
 - Train them further with gradient descent to help final task
- In pytorch: nn.Embedding()



Outline

- Recurrent Neural Networks (RNNs) for sequential data
- Language modeling and Long-range dependencies
- Vanishing gradients and Gated RNNs

Handling variable length

- Peculiarity #2: Documents have different numbers of words
 - Example 1: Amazing!
 - Example 2: There are many issues with this movie, such as...
- Problem: In previous models, number of parameters depends on size of inputs
- Challenge: How can we use the **same** set of model parameters to handle inputs of any size?



Recurrent Neural Networks (RNNs)



 Model parameters to do this update are same for each step

A "Vanilla"/"Elman" RNN



- Vector for current word x_t
- Learn linear function of both inputs, add bias, apply non-linearity
- Parameters: Recurrence params (W_h, W_x, b), initial hidden state h₀, word vectors

RNN as Repeated Application of 1 MLP layer



AKA One MLP layer!

Legend:

Word vectors Other learned parameters Hidden states

RNN as Repeated Application of 1 MLP layer



RNN "Building Blocks"

(7) RNN Layer

- Input: List of vectors x₁, ..., x_T, each of size d_{in}
 - E.g., x_t is word vector for t-th word in sentence
 - Equivalent to a T x d_{in} matrix
- Output: List of vectors h₁, ..., h_t, each of size d_{out}
 - d_{out}: Dimension of hidden state
 - Equivalent to a T x d_{out} matrix
- Formula (Elman RNN): $h_t = \tanh(W_h h_{t-1} + W_x x_t + b)$
- Parameters:
 - W_h: Matrix of shape (d_{out}, d_{out})
 - W_x: Matrix of shape (d_{out}, d_{in})
 - b: Vector of shape (d_{out})
 - h₀: Vector of shape (d_{out})
- In pytorch: nn.RNN(), etc.

Output h_1 , ..., h_T , each shape d_{out}



Input x_1 , ..., x_T , each shape d_{in}

Recurrent Neural Network Diagrams

- Can also visualize RNN's with a diagram that has a cycle ("recurrence")
 - RNN layer is just a neural network that takes in two vectors and produces a third vector
 - New vector gets fed back in next timestep
 - Previous slide is the "unrolled" diagram



Recurrence vs. Depth



- Deep networks (i.e., adding more layers)
 - Computation graph becomes longer
 - Number of parameters also grows; each step uses new parameters
- Recurrent neural networks
 - Computation graph becomes longer
 - Number of parameters **fixed**; each step uses **same parameters**

Recurrence and Depth



- You can have multiple layers of recurrence too!
 - Left-to-right axis ("time dimension"): Length is size of input, same parameters in each step
 - Top-to-bottom axis ("depth dimension"): Length is depth of network, different parameters in each row

Training an RNN



- Basic usage of RNN: Make prediction based on final hidden state
- Same recipe: Backpropagation to compute gradients + gradient descent
- Must backpropagate through whole computation graph
 - "Backpropagation through time"

Building an RNN encoder model

- A generic RNN architecture
 - Map each word to a vector
 - Feed word vectors to RNN to generate list of hidden states
 - Feed final hidden state to MLP to make final prediction (e.g., document classification)
- Basic steps are still all the same
 - Backpropagation still works
- Gradient descent needed to update all parameters



Outline

- Recurrent Neural Networks (RNNs) for sequential data
- Language modeling and Long-range dependencies
- Vanishing gradients and Gated RNNs

Autoregressive Language Modeling



- At each step, probabilistically predict the next word given current hidden state
- One step's desired output is the next step's input ("autoregressive")
- To mark end of sequence, model should predict the [END] token
- Called a "Decoder": Looks at the hidden state and "decodes" next word

Autoregressive Language Model Training

- Training example: "Convolutional neural networks are good for image classification"
- Want to maximize P("Convolutional neural networks are good for image classification")
- MLE: Take log and decompose by chain rule:

log P("Convolutional")

- + log P("neural" | "Convolutional") + log P("networks" | "Convolutional neural") + log P("are" | "Convolutional neural networks") + ...
- Decomposes into a bunch of **next-word-classification** problems
- Backpropagation + gradient descent to minimize loss •
 - Update RNN parameters
 - Update word vectors
 - Update final layer classifier over vocabulary

Generating text with LM's



- Test time: Given some prefix, "autocomplete" the rest of the sentence
- First, feed prefix as input to model
- At each timestep, choose next word based on model's predictions
 - Greedy: Choose the most likely word
 - Sampling: Sample from the model's probability distribution over words
- Feed the model's generated word back as the next word, stop if [END]

- Every step, you update the hidden state with the current word
- Over time, information from many words ago can easily get lost!
- This means RNNs can struggle to model long-range dependencies

The keys to the cabinet ___ (on the table) plural singular

- Every step, you update the hidden state with the current word
- Over time, information from many words ago can easily get lost!
- This means RNNs can struggle to model long-range dependencies

The keys to the cabinet are (on the table) plural singular

- Every step, you update the hidden state with the current word
- Over time, information from many words ago can easily get lost!
- This means RNNs can struggle to model long-range dependencies

The keys to the cabinet by the door **are** (on the table)

- Every step, you update the hidden state with the current word
- Over time, information from many words ago can easily get lost!
- This means RNNs can struggle to model long-range dependencies

The keys to the cabinet by the door on the left are (on the table)



- "Coreference": When two words refer to the same underlying person/place/thing
 - Pronouns typically corefer to an antecedent (something mentioned earlier in the text)
- Coreference relationships can even span multiple sentences

Even longer-range dependencies



- Imagine trying to generate a novel...
 - Same set of characters
 - Characters have to behave in consistent ways
 - Sensible ordering of events

Announcements

- HW2 due this Thursday, March 6
- Section Friday: Midterm preparation
- Midterm exam: Thursday March 13
 - Refer to Piazza for logistics including which room to go to
 - Practice exams released on website
 - Midterm will cover all topics through end of this week
 - Please write in pen

Outline

- Recurrent Neural Networks (RNNs) for sequential data
- Language modeling and Long-range dependencies
- Vanishing gradients and Gated RNNs

Backpropagation through time, revisited

- Model needs to know that the correct word is are because of the word keys!
- Let's backpropagate the loss on generating are to the word vector parameters for keys
 - For simplicity, let's assume all the hidden states are just 1-dimensional
 - Step 1: Compute $\delta Loss/\delta(h_T)$
 - Step 2: Compute $\delta Loss/\delta(h_{T-1}) = \delta Loss/\delta(h_T) * \delta(h_T)/\delta(h_{T-1})$
 - Step 3: Compute $\delta Loss/\delta(h_{T-2}) = \delta Loss/\delta(h_T) * \delta(h_T)/\delta(h_{T-1}) * \delta(h_{T-1})/\delta(h_{T-2})$
 - .
 - Gradient through "keys" hidden state: $\delta Loss/\delta(h_T) * \delta(h_T)/\delta(h_{T-1}) * \delta(h_{T-1})/\delta(h_{T-2}) * ... * \delta(h_3)/\delta(h_2)$
 - Gradient through "keys" word vector: $\delta Loss/\delta(h_T) * \delta(h_T)/\delta(h_{T-1}) * \delta(h_{T-1})/\delta(h_{T-2}) * ... * \delta(h_3)/\delta(h_2) * \delta(h_2)/\delta(x_2)$



The Vanishing Gradient Problem

- Gradient through "keys" word vector: $\delta Loss/\delta(h_T) * \delta(h_T)/\delta(h_{T-1}) * \delta(h_{T-1})/\delta(h_{T-2}) * ... * \delta(h_3)/\delta(h_2) * \delta(h_2)/\delta(x_2)$
 - What is each individual $\delta(h_t)/\delta(h_{t-1})$ term ?
 - Elman network: $h_t = \tanh(W_h h_{t-1} + W_x x_t + b)$, $\frac{\delta h_t}{\delta h_{t-1}} = \tanh'(W_h h_{t-1} + W_x x_t + b) \cdot W_h$
 - After t timesteps, have a factor of $(W_h)^t$ (to the t-th power)!
 - If W_h << 1, this quickly becomes 0 ("vanishes")



The same

parameter

over and over!

Ignore for now

The Vanishing Gradient Problem

- Vanishing Gradients: Updates to one word/hidden state not influenced by loss on words many steps in the future
 - Illustrated only for 1-dimensional hidden states, but same thing happens when states are vectors/parameters are matrices
- Result: Hard for model to learn long-range dependencies!



Vanishing and Exploding

- Vanishing gradient occurs because
 - Gradient w.r.t. words t steps in the past has $(W_h)^t$
 - And when $W_h \ll 1$ (e.g., at initialization time)
- What if *W_h* > 1?
 - Gradients get bigger as you go backwards in time: Exploding gradients!
 - Vanishing gradients more usual, but explosion can happen too
- Quick fix: Gradient Clipping
 - If gradient is super large, "clip" it to some maximum amount
 - Rescale the total vector to some maximum norm
 - Clip each entry to be within some minimum/maximum value
- Outside of RNNs, vanishing/exploding gradients can happen whenever you have long computation graphs with lots of multiplications

Avoiding Vanishing Gradients

• Where did we go wrong?

$$h_t = \tanh \left(W_h h_{t-1} + W_x x_t + b \right),$$

Multiplicative
relationship between previous
state and next state

$$\frac{\delta h_t}{\delta h_{t-1}} = \tanh' \left(W_h h_{t-1} + W_x x_t + b \right) \cdot W_h$$

Leads to repeated multiplication by *W*_h



Avoiding Vanishing Gradients

• Extreme idea: A purely additive relationship

...

- Pro: No vanishing gradients
- Pro: Old hidden state carried through to all future times
- Con: May be good to "forget" irrelevant information about old states

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

Additive
relationship

$$\frac{\delta h_t}{\delta h_{t-1}} = 1 + \frac{\delta}{\delta h_{t-1}} g(h_{t-1}, x_t)$$

Gradients also add,
not multiply



Avoiding Vanishing Gradients

- Middle-ground: **Gated** recurrence relationship
 - Additive component makes gradients add, not multiply = less vanishing gradients
 - Forget gate allows for selectively "forgetting" some neurons within hidden state
 - When forget gate is all 1's, becomes the purely additive model (no vanishing)

Elementwise multiplication

$$h_t = h_{t-1} \odot f(h_{t-1}, x_t) + g(h_{t-1}, x_t)$$

"forget gate" Additive
in [0, 1] relationship



Gated Recurrent Units (GRUs)

- One type of gated RNN
 - Here z is the "forget gate" vector
 - Where *z_i* = 0:
 - Forget this neuron
 - Allow updating its value
 - Where *z_i* = 1:
 - Don't forget this neuron
 - Do not allow updating its value
- Parameters: *W*, *U*, plus parameters of *g*
 - (g has a slightly complicated form not shown, has its own parameters)

"forget gate" $h_{t} = h_{t-1} \odot z + g(x_{t}, h_{t-1}) \odot (1-z)$ $z = \sigma(Wx_{t} + Uh_{t-1})$ Sigmoid ensures gate is

between 0 and 1

Long Short-Term Memory (LSTM)



$$egin{aligned} f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \ i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \ o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \ c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \ h_t &= o_t \circ \sigma_h(c_t) \end{aligned}$$

- Another, more complicated gated RNN
- Commonly used in practice
- What's the idea?
 - Split the hidden state into normal hidden state h_t and "cell" state c_t
 - Cell state uses gated recurrence
 - Hidden state is gated function of cell state

What do LSTMs learn?

- Here: a characterlevel LSTM (not word-level)
- Blue/Green: Low/high values of 1 neuron
- Below: Top-5 predictions for next character

This neuron seems to detect whether we're inside a URL

t	t	р	:	1	1	w	w	w		у	n	е	t	n	е	w	s		С	0	m	1]		E	n	g I	i	S	h	- 1	T	а	n	g	u	a	g	e	v	N e	e k) s	i	t	e		0	f		I.	s	r	а	e	1			1	а	r
t	p			1	w	w	w		b	а	С	а	h	е	t	s	403	с	0	m	1			-	x	g	1	1	i h		1	i	n	g	u	a	g	e :	s a	a i	r	s	i	t	e		0	f		t	s	1	а	е	L	i	s	s	i	n	g
	d		:	х	n	е		w	а	е	а			a	w	a	t	0	а	•		s		&	n	t	1	a	a	-	s	a	r	d	е	e	I	h	0	o a	a r	1	t	b	i	s	a	n	f	а	n	r	е	i	f	1		a	a	t	d
m	w	-	2	Ŷ	р	i	i	i	s	0	е	s	s	i	s		1	е	r	n		c]	(d	С	e	e r	1	e	p	e	s	a	a	i	k	i	i	e	e	e	e	d	h	,	i	r	t	h	r	a	0	n	s	e	,	С	0	s	е
d	r		<	:	а	h	b	-1	n	р	t	w	t		x	i		g	h	1	m	a)	Т	v	d	- 1	1 2	: i		С	0	u	е	d	1	s	u	: t	h	n a	a -	C	0			t	u	,	s	t	u	i	f		1	Ve	e p	е	r	у
s	t	р	,	t	с	0	а	2	d	r	u	I	w	0	С	1	e	n	s	r]	р		1	1	v	a	0	1,	,	e	у	t	С	-	n		dI	m -	. c) i	b	o u	ı v	s]	b	b		i	m	S	u	I	t	a	t	I	у	b	n

g	е	s	t		n	е	w	5	pa	a	o e	r		•	•	[]	Y	е	d	i	o t	h		Α	h	r I	o n	0	t	h]	1	1		•	• •	H	l e	b	r	е	w	-	a	n	g	u	a	g	е	1	e	r	i.	0	d
e		t		а	а	w	s	2	a p	0 0	e r	s	0	1	[]	Т	е	I		t		i	(f	e	aI	n e	m	t	i	1	1		*		'	e	e r	r	е	w	S	(e r	g	u	a	g	е	•••	a r	0	S	0	d	i
i	r		s	С	0	е		(e r	n a	a		i	Т	Г	h	A	0	a	i	n	n ł	1	S	r	m	u \	N]		е	У		S	;			[]	i	n	е	i	а	•	s i	v	v d	d	е	'	h	s	0	r	i	f	r	:
u	s			s	е	t	1 9	9 (o r		s		a	s	; a	ı t	C	a	r	е	e	g '	a	С	I	r	i s	s z]	i	е	· :	:	,	#	:	T	A a	a a	a	а	t		Ba	a s	e	е	i	L	0	•	ia	a n	f	v	I	
Г	-			t	u	a	e v	/ 1	r t	i	d	Ι,	t	E	3 A	A n	n S	u	s	у	u 1	:]]	A	s	a	o i	g	s]]	, .			:	s	M	3 c		0	u	s	: '	T	o u	a	-	n	:	d		w	a		р	n	u
а	,	d	,	i	i	u	i t	i	(ο.]	(1	S	S V	H	l v	t	u	S	ı i	e	D	n	0	e	g a	n	0		. ,]	:	{		C	CL	ıi	b	0	h	e	C	/ b	k	s	I	s	:	r	- () p	С	n	t	s

i	c	a	L	s	:	•	•	•		*	•	•	[[G	1	0	b	e	S	1]	1		E	h	t	t	р	: 1	1	w	w	w		g I	C	b	е	s		С	ο.	i	1	I]	b	u	s	i	n	e s	s		d	а
С	а	L			۲	•	*		*	•	•	•	[Т	а	а	b	а			1	1		(]	t	t	p			V	v w	W		b	u	b	a	1		С	0	m u	n	1	s	Α	-	у	t	i	n	е	S S		а	е	t
s		t	Ι	•								[h	Α	е	0	v	е	L	t		5	5	a	h	а		d		;)	(g	е		w	a	o i	r		r	t	0	a	. е	1		i	Т	8	a	i		е	g		е	0	0	у
t	t	•	•					'	&	[&	&	m	С	0	е	r	0	n	е	•	: :	,	i	'	0	d	w		; :	n	i	i	i	s	aa	a U	ı e		е	n	i	/ 0	n	n I	С	С	. (е	f	t	g	i	r		i	i	u
а	'	n	:	,	С	:	&	:	#	*	:	a	f	D	r	u	s	u]	I		,		0	m	n e	L		р	< ,	d	h	a	;	d	eι	1 0	0	t	1	i	h	n c	s	i	f	S	,	u	r	h	0	s	t	,	t	u	n
n	k		i		<]	:	&	1	1	s		Т	G	u	i	t	r	s	i	,		:	b	a	С	m	r	-)	x t	p	0	b	-	g	r e	s	i	s	I	е	r	l n	a	f	а	D]	0	s	р	t	a	d,	i	f	r	m

i	1	у		*	•	•	[H	1 2	a	r	e	t	z	1	Н	a	•	A	r	e		: 1	1	•	•		[h	1	P	:	1	1	w	w	w.	h	a	а	r:	e	t :	z.	c	0		i	1	1]		R	e	a	t	i	v
1	у		*		٠	[1	e	e r	r	d	n			F	е	r	a	n	t	a	h []	1		1		([t	t p	3	F	1	w	w	w.	t	0 0	n	m	d	s	t.	. 0	: 0	n	n u	n	1	s		-	е	S	a t	e	0	i
r	e			1		•	h /	A i	1	n	n	t	t	е	H	la	1	s	r	С	n	0			s		а	h	a	C	ł	:	x	n	е	. 1	w a	aa	m	r	t	d	h	ec	h		0	1		С		&	0	pi	i n	i	v	е
k	i		:	*	s	С	0 5	S a	a r		t		h	i	Т	i	m	'	L	i]	e			:	,	i	m	C	d v	v -	2	Ŷ	р	h	i i	s	s e	r	d	i	t	. i	i r	n a	1	С	m	n f	i		(а	f	С	a	n	а
d	s	-	!	[t	B	Т	0	n	n n	n g	d]]	N	Vo	n			а		a	э,	:			b	a	e	r		<	t	а	i	b -	- 0	u k	I	С	n	n	c /	1 8	a r	n	е	s	i]	L	i	C	e y	s	t	0
n	C	s	#	&	:	G	1) L	ı v	C	С	s	a	0	S	u	С	I	t	е	1	2	z	,		:	0	1	0	m t]	,	:	е	0	a	2 r	n i	v	f	s	r	0	οε	e i	u	n	a	1	a)		u	v	v r	0		

What do LSTMs learn?

- Here: a characterlevel LSTM (not word-level)
- Blue/Green: Low/high values of 1 neuron
- Below: Top-5 predictions for next character
- This neuron fires only inside a Markdown [[link]] (so it knows when to close the square brackets?)

	* *]]	J	e	r	u	s	a	F	e	n	1	R	е	p	0	r	t	1]	•		I	h	t	t	p	:	Γ	1	w	w	v .	j	r	e	P		C	0	m	1	1	L	е	f	t	-	0	f	-	C	e r	ı t	е	r	I	E n	1
*	1	[h	Т	0	a	u	s	а	L		n	n	а	0	g	u	r	t]	1	•		(h	t	t	р			1	w	w	w .	b	s	i	n	i	0	0	m	1		-	i	a	t		а	f		t	e	n t	е	r		(1	ı g	
	[']	С	a	s	s	m	е	n	e]	B	е	a	0	n	d	s			s	a	[a		d		:	x	n	e.	. v	v a	a	a	a	0	С	a			s	5	k a	t	0	-	-	n	f	h	h I	S	u	m	-	0 1	l C	
	's	;	n	n F	u	r	n	1	s		i	a	e	t	a	1	1	s	а		'	:	: ,	i	۲	С	d	w	- 1	2	t	р	i li	i	s	0	e	g		е	r	1		a] (0	s	е	s	w	r	-	с	i	d c	i r	s		[n t	
:	* :	A	A q	D	e	n	е	b	i	u	t	n	1	C	i	р	r	e		е	1	,		b	1	e	m	r		9	:	a	hI	b -	r	ı p	u	m	u	g	h	n	m	p) 1	Ге	i	r	е	t	u	:	e	0 5	s e	e 0	d	s	a I	d	
#	T 8	k T	f	S	i	w	r	р		е]]	a	1	u	v	е	I	r	u	,	s	:	-	n	n p	r	t	s	<	Ŷ	m	0	a 2	2 0	l e	у	s	h	i	I	r]	С	. /	A u	g	1	,	1	р	,	I	a r	c	:		:	fa	a e	
						-	_		-		1-	14		-		-	-				-	-	_			1.	-					_	_	_			1-	-					- 1				1	-		-					-	-		_		_	

g	1	i	s	h		I	[w	е	e	k	1	у		n	e	e v	v s	s p	a	p	e	r	1	1		*	1	•	[[Y	N	e	t	N	e	w	5]]				[h	t	t	p	:	Ι	1	w	w	w		у	n	e	t	n	e	w	s .	c
1	i	s	h		С	1	С	а	а	k	1	у]	С	а	V	v s	F	a	р	e	r	1	1		*		1]	h	Т	а	A			a	t			1	1		(h	t	t	p		1		w	w	W		b	а	С	а	h	е	t	s.	С	C
i	а	С	i	-	L	h	S	0	i	p	1	i		s	e	C	;]	e	n	p]	s				•		۲.	[С	0	2	*	w	e	s	s]			S	5	a	[a		d		:	x	n	е		w	а	е	а			а	w	a t	t o	a
е	е	n	а	,	р	С	С	i	e	t	n	e	d	1	0	X	(]	g	j i	С	i	I	1		s	'	[s	Α	m	F	е	S	a	h	0	n []]	!	:	:	,	i	m	1 0	m	w	-	2	•	р	i	i	i	s	0	е	s	s	i	s .	. /	e	r
s	у	z			s	f	р	е	n	n		a	1	r	u	e	• I	1	r	r	a				'	#	*	:		0	D	u	F	r	e	i	u	e	o	,		:	b	1	e	d	r		<	:	a	h	b		n	р	t	w	t		x i	i	g	h
a		d	р	е	a	m	A	r	b	d	e	0	r	p	i	t	e	; e	9]	d	t	s	-	1		Т	{	[В	а	A	v	Т	р	0	S	w	a	э,				0	a	С	s	t	p	,	t	С	0	a	2	d	r	u	I	w	0	c I	(e n	s

0	m]		E	Er	n g	1	i	s	h	-	L	а	n	g	u	a	g	e	W	е	b	s	i	t e		0	f		1	s r	a	е	L	•	s	I	a	r	g	е	s t		n	е	w	s	p	a p	e	r]	[Y	e d
m	1		-	×	((g I	i	s	h		L	i	n	g	u	a	g	e	s a	i	r	s	i I	t	е	C	f		t	s	e	a e	1	i –	s		s i	n	g	е		t	a	а	w	s	р	a) e	r	s	0	1	[Т	е	1
		S	8	k r	n t	i	a	c	a	-	s	а	r	d	е	e	1	h	0	a	n		t	b	is	a	n	f	а	n	r e	e i	f		'		a a	a t	d	i	r		s c	0	е			eI	n a			i	Т	Th	A	0	a i
n	. (0]	(C	d d	c (e	n		е	р	е	s	а	а	i	k i		i	е	е	I	е	d	h,	i	r	t	h	r a	a c	o n	s	е		,	c c	s	е	u	s		s	е	t	I	g	0 1	•	s		а	s	a t	С	а	r e
1	ma	a)	Т	r v	/ 0	d r	y	z	i		С	0	u	е	d	I	s	u :	t	h	a	-	0	0		t	u	,	s	t	u i	f		I	v	e	p e	e r	У		-		t	u	а	е	v	r 1	i	d	,	t	в	A n	n S	u	s y
r]	ο.	1	1	N	1 2	0	d	,	,	е	у	t	С		n		d	n -	0	i	b	u	v	s]	b	b		i	m	s l	I I	t	а		t	l y	b	n	а	,	d,	i	i	u	i	t	i	c p]	(1	S v	Н	v	t u

i	0	t h	1	A	h	r	o	n	0	t	h	1	1	*	•		*	•	•	H	e I	b	r	e v	v -	1	a	n	g	u	a	g	e	F	o	e r	i	0	d	i.	C	a	1	s	:	•	•		*	٠	i.	[[GI	C	b	е	s	1		
t		i	(f	е	а	n	e	m	t	i	I	٠	۲		*	1	•]	e	r I	r I	e	N S	1	е	n	g	u	a	g	е	:	a r	•	o s	0	d	i	С	а	L		:	• •	1		*	۲	•	•	L	Т	a	a b	a			1		
n	n	h	S	S r	m	u	w]		е	у			s				[•	i	n	e	i	a '	s	i	N	/ d	d	е	•	h	s	0 1		r i	f	r	:	s		t	I	1							[h	A	e	o v	e	1	t		S	;
е	g	a			r	i	s	z]	i	e	•	:	:	,	#	:	Т	A	a	a	a	al		E	3 a	s	e	е	i	1	0	'	ia	a li	n f	V	1		t	t	E.	1				'	&]	&	&	m	С	0 6	e r	0	n	е	1	: :	Į.
u	t	1]	A	A s	а	0	i	g	s]]	,				:	s	Μ	В	0	(0	u :	s :	Т	0	u	a	-	n	:	d		w	5	a	р	n	u	а	•	n	:	,	C :	8	:	#	*	:	а	f	D	rι	l s	u]	I	,	,	
s	u	i e) [) n	0	е	g	a	n	0			,]	:	{		С	С	u	i	b	0	n e	• 0) y	b	k	S	I	s	:	r	- e	e	o c	n	t	s	n	k		i		<]	:	8	1	1	s		Т	G	u i	t	r	s	i	,		

Conclusion

- Deep Learning for Language must deal with possibly long inputs
- RNNs handle arbitrarily long inputs with fixed number of parameters
- Need to handle long-range dependencies, but hard to learn due to vanishing gradients
- Gated RNNs (GRUs, LSTMs) can reduce vanishing gradient problems