

Generative AI from First Principles

Lecture 2: Language Modeling

From Counts to Neural Networks

Language Modeling: From Counts to Neural Networks

Two ideas, older than neural networks

Text is a random source

Markov (1913), Shannon (1948): count what follows what, and measure how predictable text is

→ n-gram models: speech recognition and translation for decades

Words are points in a space

a word is known by the company it keeps: similar contexts, nearby vectors

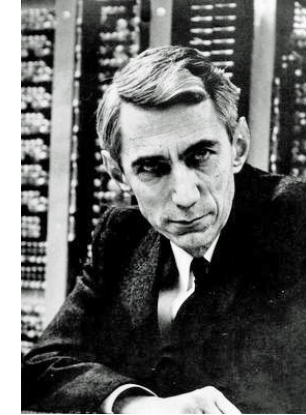
→ semantic search, recommendation, vector databases

Neural language models join the two

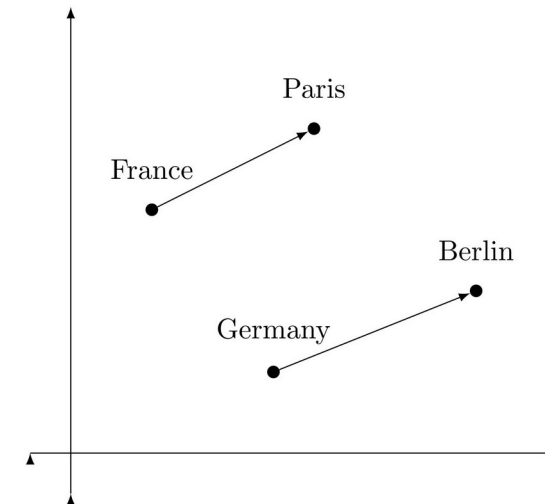
predict the next token, as counting does, but from learned vectors



Andrey Markov



Claude Shannon



Lecture 2: Language Modeling: From Counts to Neural Networks

1. Counting: text as a random source

- Markov and Shannon; tokens; the language model as a tree
- n-gram models; measuring a model; smoothing

2. Word embeddings: words as points in a space

- why vectors, and where they are used
- vectors from counts and from prediction (word2vec), and why they meet
- what similarity captures, and what it misses

3. Neural language models: from the bigram network to the LSTM

- the bigram network; Bengio's fixed window
- recurrent networks: backpropagation through time, vanishing gradients
- the additive path; the LSTM

Text as a Random Source

Markov, 1913: text as a chain

1906: the law of large numbers holds for chains, not only for independent trials

1913: a test on *Eugene Onegin*: 20,000 letters, each a vowel or a consonant

vowel pairs: **1,104**, where independent letters give $\approx 3,730$

the next letter depends on the last

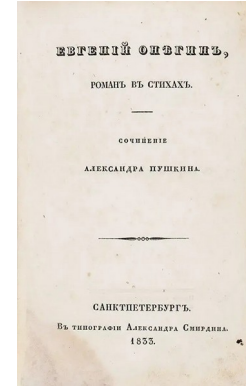
a prediction: vowels per 100-letter block vary less than coin tosses

$$\frac{\text{variance, chain}}{\text{variance, independent}} \approx \frac{1 + \delta}{1 - \delta} = 0.30 \quad \text{poem: } \mathbf{0.21}$$

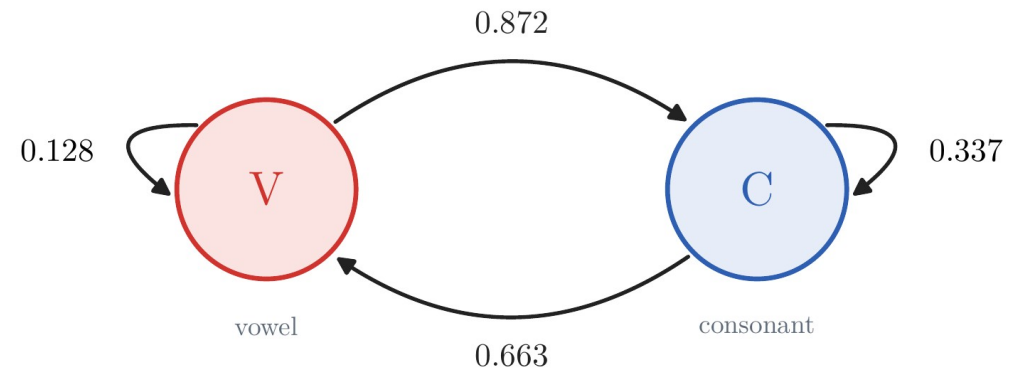
$$\delta = P(V | V) - P(V | C) = 0.128 - 0.663$$



A. A. Markov



Eugene Onegin



Text as a Random Source

Shannon, 1948: more context, better text

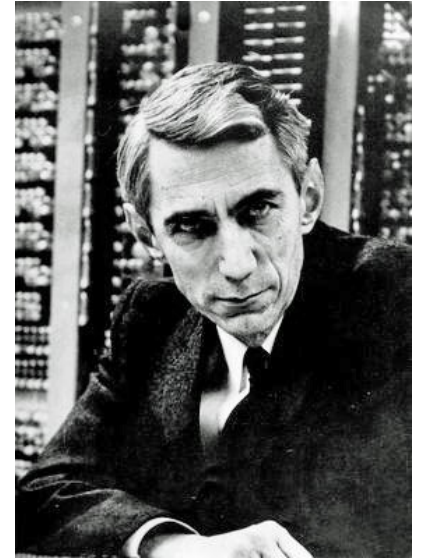
letters, $n = 1$ *ari ef o eheiowamna wactifp oe nu,lao0.smnse*

letters, $n = 3$ *saw heir he up boalk. stim. her frod their wore*

letters, $n = 5$ *once upon a tim. the pot. john and warned the*

letters, $n = 7$ *once upon a time for you are going to eat her smile.*

words, $n = 3$ *ben liked to eat some bread, cheese and ate their delicious desserts*



Claude Shannon

letters or words?

Text as a Random Source

Tokens: characters, words, or pieces

(a)

characters
vocabulary 33 + end

o n c e u p o n a t i m e , l i l y f o u n d a s h i n y p e b b l e . 44 tokens

words
vocabulary 8,681

once upon a time , lily found a shiny pebble . 11 tokens

GPT-2 pieces
vocabulary 50,257

Once upon a time , Lily found a shiny pe b ble . 13 tokens

(b)

occurs 1,000 times or more
360 words of the vocabulary



97% kept as one token

happy

garden

occurs 100 to 999 times
1,558 words of the vocabulary



92% kept as one token

butterfly

spark ly

occurs 10 to 99 times
2,952 words of the vocabulary



76% kept as one token

gir affe

pe b ble

occurs 2 to 9 times
3,384 words of the vocabulary



53% kept as one token

rac coon

sho el aces

Text as a Random Source

The language model, one token at a time

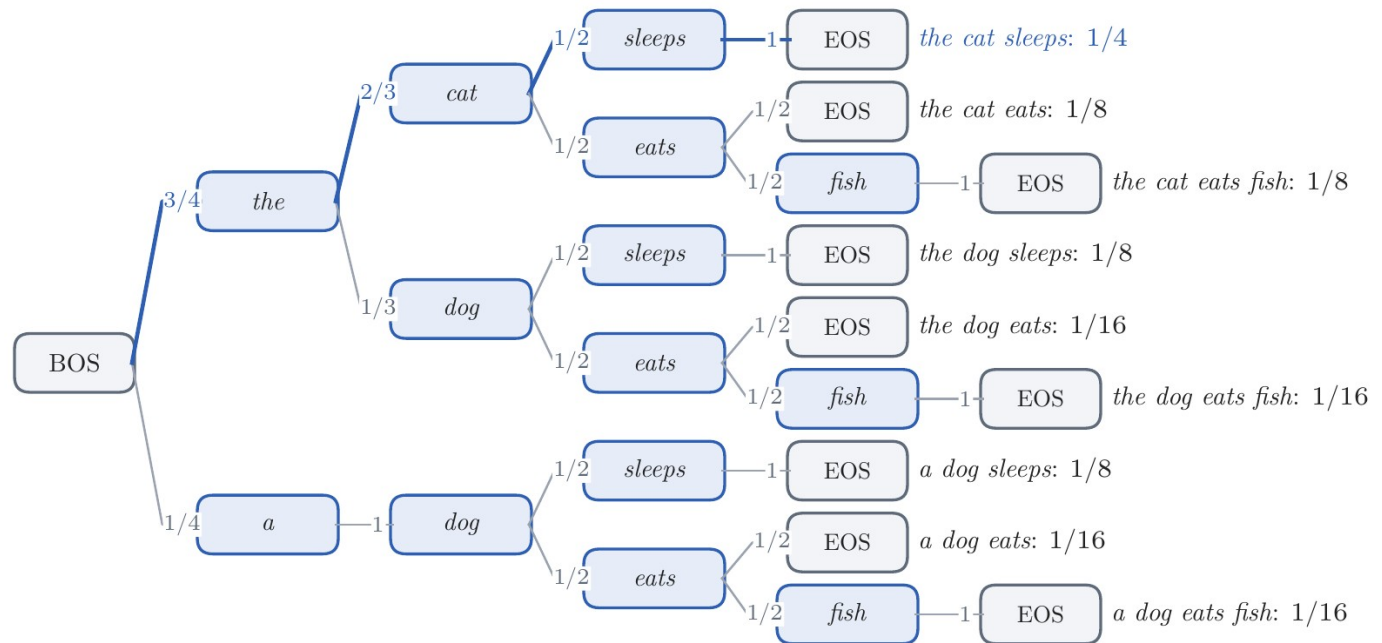
$$q(x_1 \dots x_T \text{EOS}) = \left[\prod_{t=1}^T q(x_t \mid x_{<t}) \right] q(\text{EOS} \mid x_{1:T})$$

BOS: the start, given

EOS: the stop, predicted

corpus

the cat sleeps
the cat eats fish
the dog sleeps
a dog eats



the bigram model of the corpus: every sentence is a path, and the paths' probabilities sum to one

n-gram Language Models

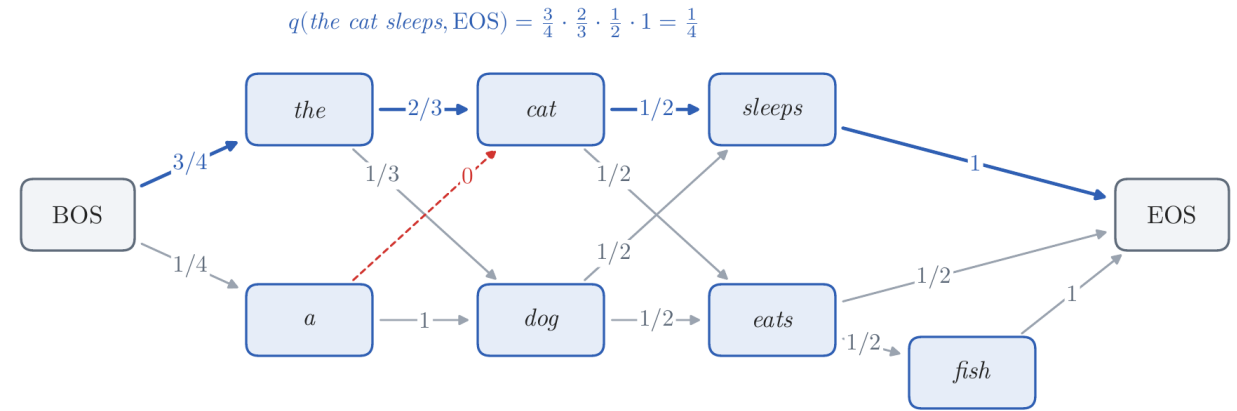
Count and divide: maximum likelihood

$$q(x_t \mid x_{<t}) \approx q(x_t \mid x_{t-n+1} \dots x_{t-1})$$

$$\max_q \prod_{h,w} q(w \mid h)^{c(h,w)} \implies q(w \mid h) = \frac{c(h,w)}{c(h)}$$

keep only the last n - 1 tokens

maximum likelihood = relative frequencies



Hands-on: Counting

[week02_handson.ipynb, stop 1](#)

```
In [3]: def complete_sentences(model, prefix=(), p=1.0):
# walk the tree of prefixes; each branch adds one word
for word, q in model.distribution((ngram.BOS,) + prefix).items():
    if q > 0 and word == END:
        yield " ".join(prefix), p * q          # an END leaf: a complete sentence
    elif q > 0:
        yield from complete_sentences(model, prefix + (word,), p * q)

leaves = sorted(complete_sentences(bigram), key=lambda leaf: -leaf[1])
for sentence, p in leaves:
    print(f"{p:.4f} {sentence}")
print(f"\n{len(leaves)} sentences, total probability {sum(p for _, p in leaves):.2f}")
```

```
0.2500 the cat sleeps
0.1250 the cat eats fish
0.1250 the cat eats
0.1250 the dog sleeps
0.1250 a dog sleeps
0.0625 the dog eats fish
0.0625 the dog eats
0.0625 a dog eats fish
0.0625 a dog eats
```

```
9 sentences, total probability 1.00
```

watch: *a cat sleeps* is not in the tree

n-gram Language Models

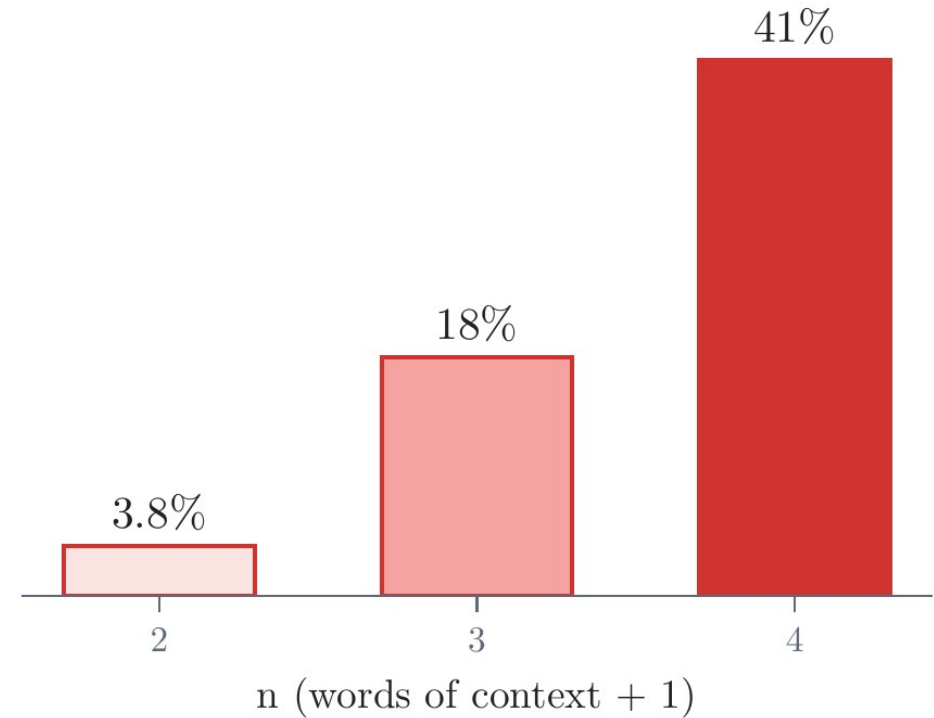
Sparsity: the trouble with counting

a cat sleeps: **probability 0**

longer context: better guesses, fewer counts

which wins? we need a score

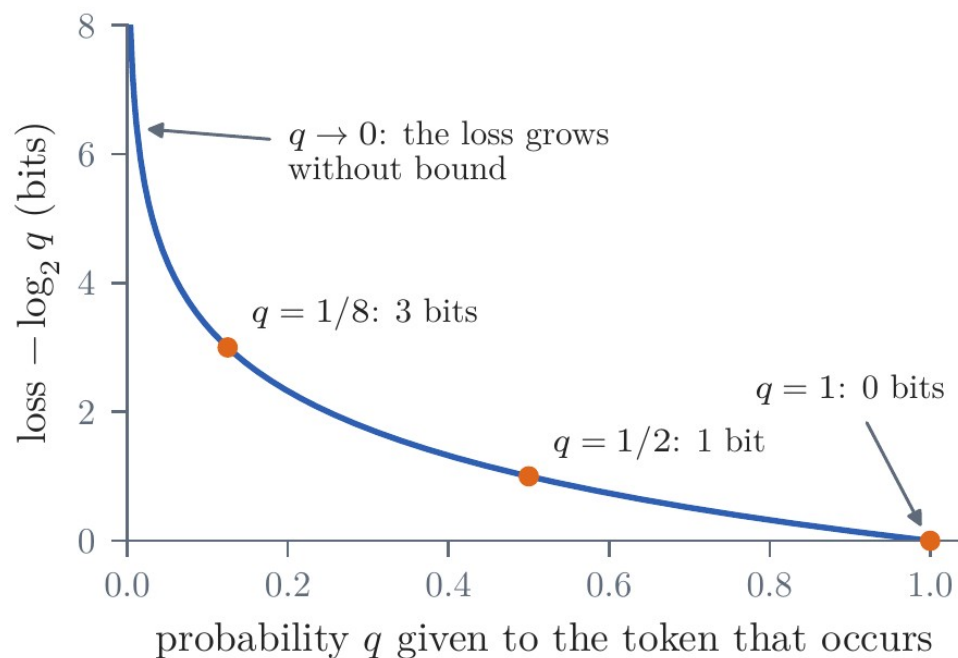
test n-grams never seen in training



Measuring a Language Model

Log loss, cross-entropy, perplexity

loss = $-\log_2 q$ (the token that occurred)



on held-out text

$$H(p, q) = \underbrace{H(p)}_{\text{the text}} + \underbrace{D_{\text{KL}}(p \parallel q)}_{\text{the model's error}}$$

$$\hat{L} = -\frac{1}{M} \sum_t \log q(x_t \mid x_{<t}), \quad \text{PPL} = e^{\hat{L}}$$

perplexity: as uncertain as a fair K -sided die

words: unigram 300 $\rightarrow$ Kneser–Ney 4-gram 25

bits per character: the average loss in bits, per character

Smoothing

Never give probability zero

add α to every count

$$q_{\alpha}(w \mid h) = \frac{c(h, w) + \alpha}{c(h) + \alpha|\mathcal{V}|}$$

α chosen on validation data

Kneser–Ney (1995): borrow from shorter contexts, the best version

TinyStories words

test perplexity	bigram	trigram	4-gram
no smoothing	∞	∞	∞
add-one	110	421	
add- α , tuned	52	59	
Kneser–Ney	47	28	25

Smoothing

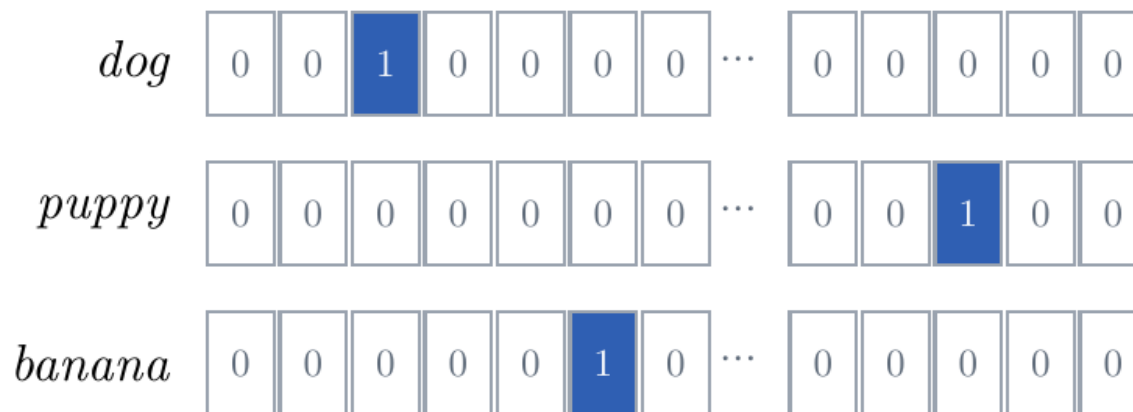
What counting cannot do

the dog barked

the puppy barked

nothing shared

one-hot: one dimension per word (8,681 in all)



every pair: cosine 0, distance $\sqrt{2}$

to a table, every word is its own dimension

Lecture 2: Language Modeling: From Counts to Neural Networks

1. Counting: text as a random source

- Markov and Shannon; tokens; the language model as a tree
- n-gram models; measuring a model; smoothing

2. Word embeddings: words as points in a space

- why vectors, and where they are used
- vectors from counts and from prediction (word2vec), and why they meet
- what similarity captures, and what it misses

3. Neural language models: from the bigram network to the LSTM

- the bigram network; Bengio's fixed window
- recurrent networks: backpropagation through time, vanishing gradients
- the additive path; the LSTM

Word Embeddings

The company a word keeps

A bottle of **tezgüino** is on the table.

Everybody likes **tezgüino**.

Tezgüino makes you drunk.

We make **tezgüino** out of corn.

What is tezgüino?

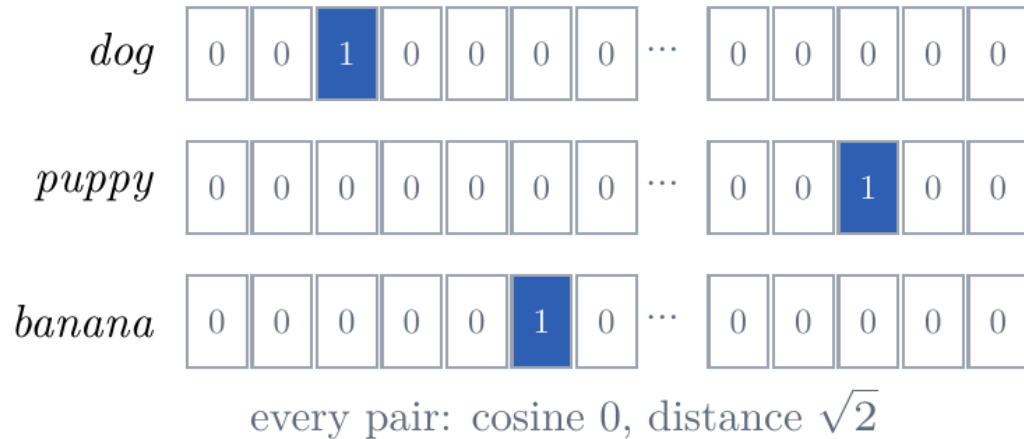
Harris, 1954: similar contexts, similar meanings

Firth, 1957: “You shall know a word by the company it keeps.”

Word Embeddings

Why vectors

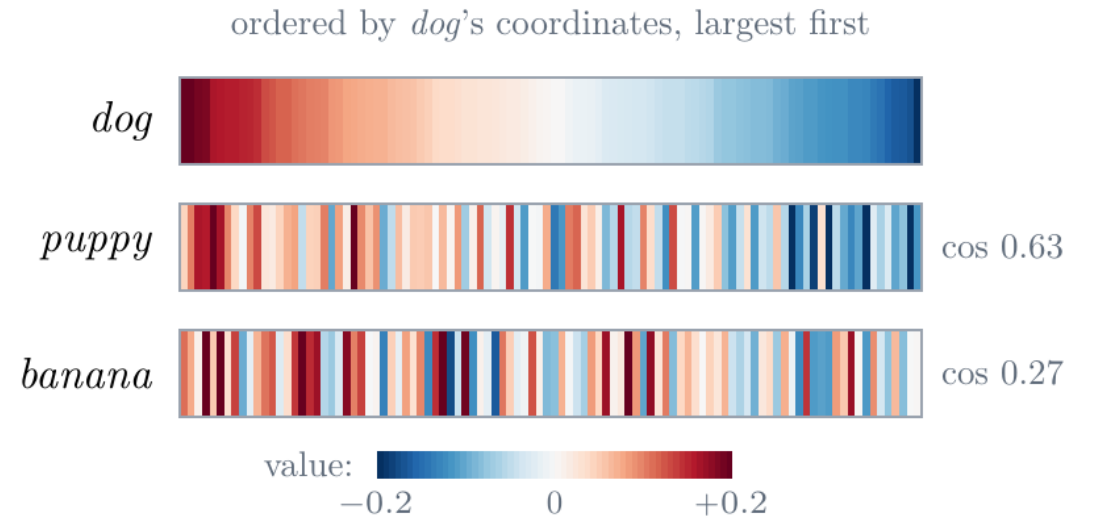
one-hot: one dimension per word (8,681 in all)



one-hot: all words equally far apart

dense: similar words point the same way

dense (skip-gram): 100 coordinates, one stripe each

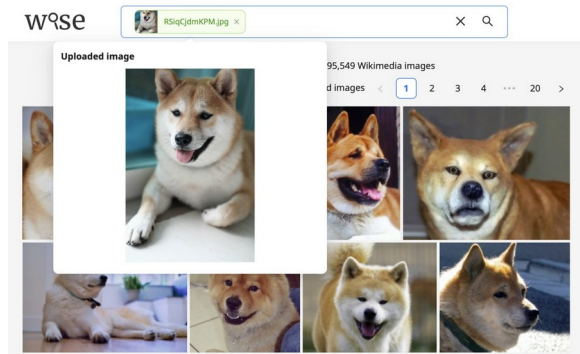


$$\cos(\mathbf{v}, \mathbf{v}') = \frac{\mathbf{v} \cdot \mathbf{v}'}{\|\mathbf{v}\| \|\mathbf{v}'\|}$$

Word Embeddings

Where embeddings are used

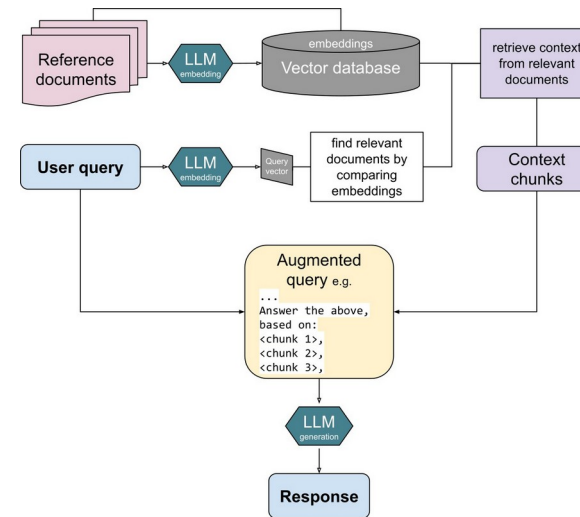
search



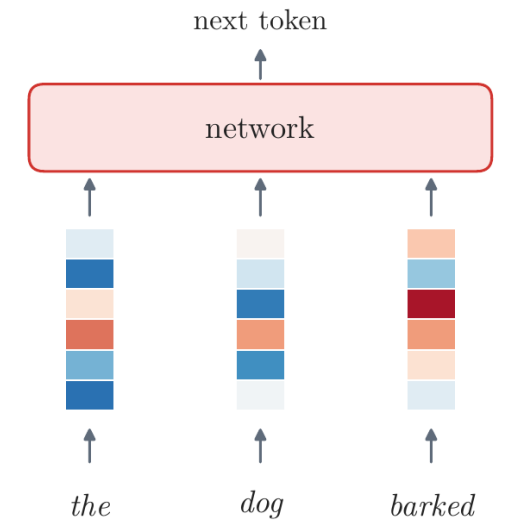
recommendation



retrieval



language models



by meaning, not keywords

items as words

nearest passages for an assistant

a vector per token, the first layer

one recipe: objects in similar contexts get nearby vectors

Word Embeddings

Vectors from counts

count: who occurs near whom

reweight by PMI

compress by SVD

mom → dad, mommy, mother

$$\text{PMI}(w, c) = \log \frac{p(w, c)}{p(w)p(c)}$$

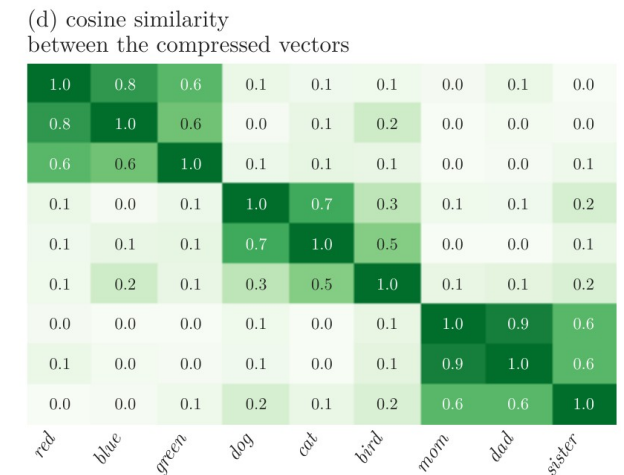
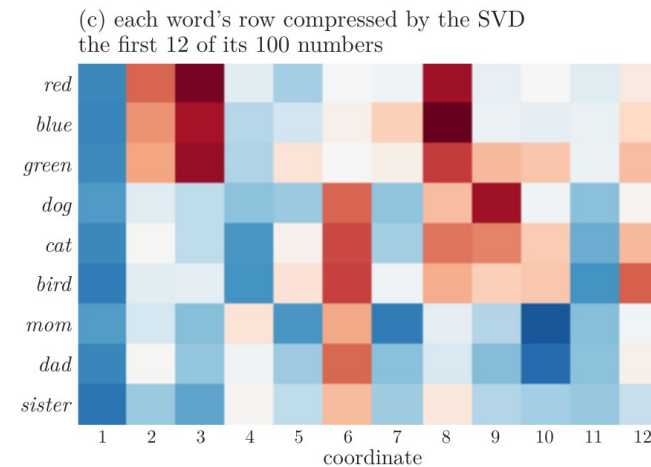
$$M \approx U_d \Sigma_d V_d^\top, \quad \mathbf{v}_w = \text{row } w \text{ of } U_d \Sigma_d^{1/2}$$

(a) counts $n(w, c)$
9 of the 3,000 rows, 8 of the 3,000 columns

word w	color	paint	barked	meow	fly	hug	said	the
red	4	5	0	0	0	1	15	354
blue	7	7	0	0	6	0	21	345
green	2	1	0	0	0	0	4	226
dog	0	0	42	0	0	11	122	2596
cat	1	1	1	6	2	2	106	1530
bird	0	0	1	0	71	1	247	3654
mom	1	1	0	1	5	108	3067	1353
dad	0	1	0	0	0	16	513	378
sister	0	0	0	0	0	1	34	48

(b) positive PMI
the same entries, reweighted

color	paint	barked	meow	fly	hug	said	the
2.1	1.6						
2.9	2.2			1.1			0.2
2.2	0.7						0.3
		3.6			0.4		1.2
0.5		0.4	3.4			0.3	1.2
				2.3		0.3	1.3
					1.2	1.5	
					0.7	1.2	
						0.5	



Word Embeddings

Vectors from prediction: word2vec

predict the neighbors of each word

$$q(c \mid w) = \frac{\exp(\mathbf{v}_w \cdot \mathbf{u}_c)}{\sum_{c'} \exp(\mathbf{v}_w \cdot \mathbf{u}_{c'})}$$

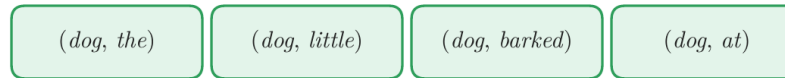
a sum over the whole vocabulary: slow

instead: real pair or made up?

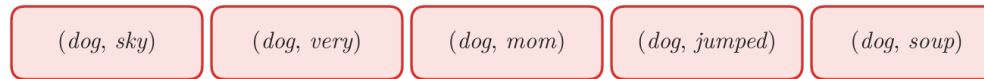
(a) the data: pairs of nearby words, and made-up pairs



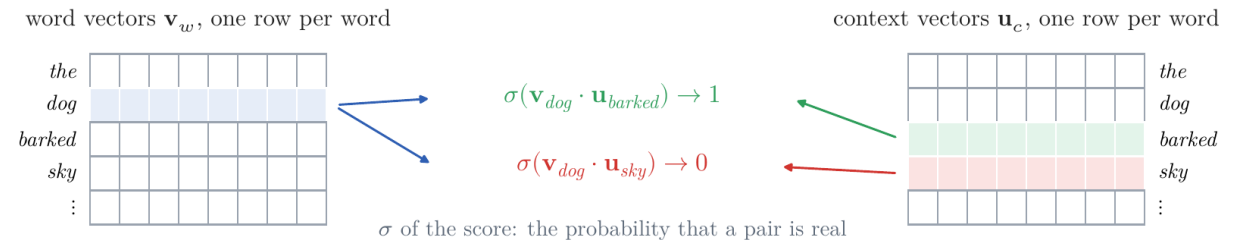
observed pairs (dog, c) : real, label 1



$k = 5$ made-up pairs for $(dog, barked)$, contexts drawn from p_n : label 0



(b) the model: two learned tables of vectors, and a logistic regression on a dot product

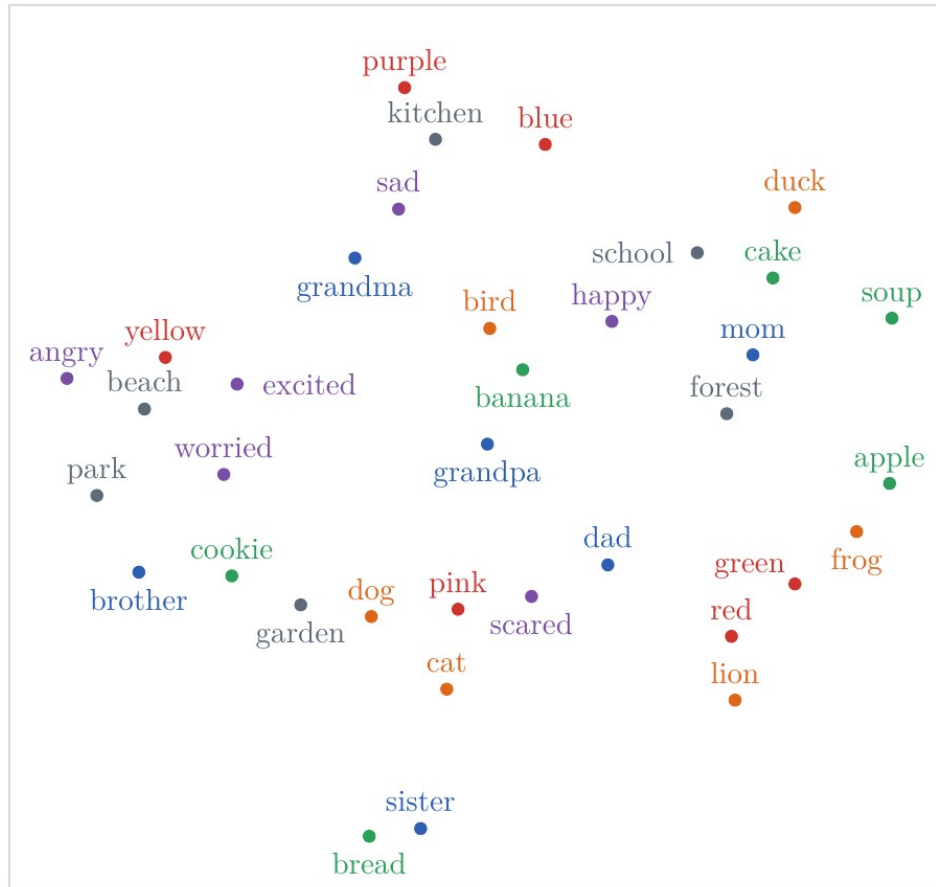


$$\log \sigma(\mathbf{v}_w \cdot \mathbf{u}_c) + \sum_{i=1}^k \log \sigma(-\mathbf{v}_w \cdot \mathbf{u}_{c'_i})$$

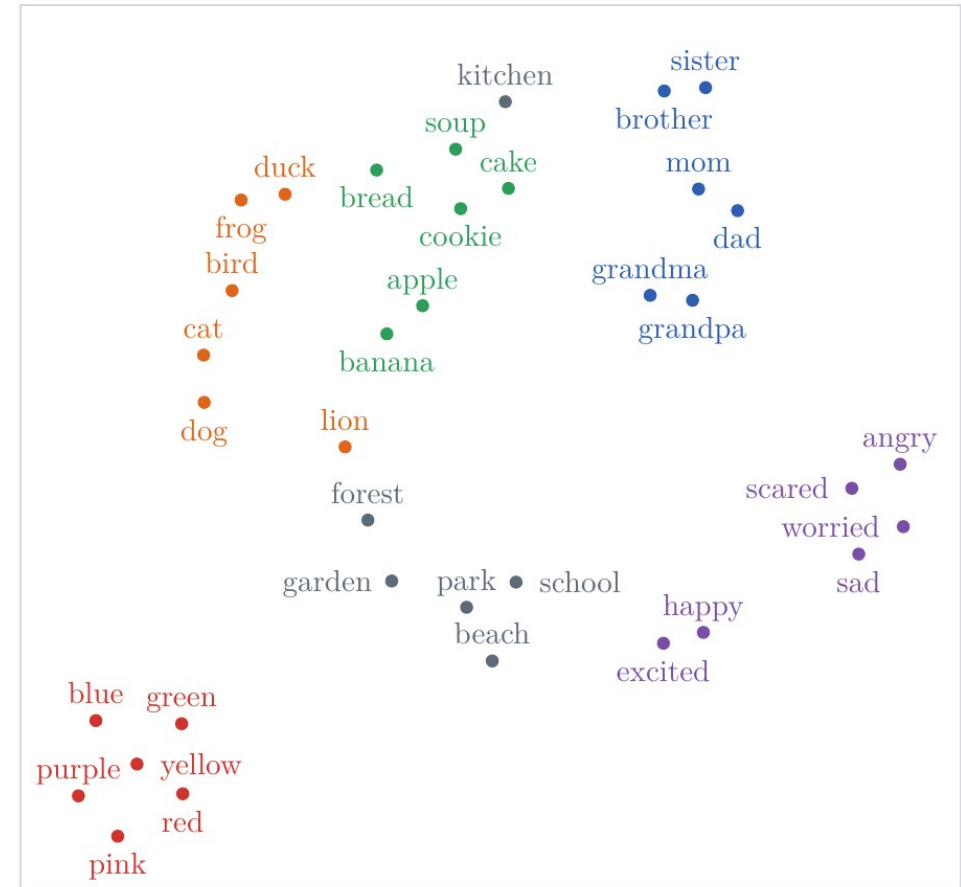
Word Embeddings

Before and after training

random vectors, before training



after ten passes of skip-gram with negative sampling



● colors ● animals ● family ● feelings ● food ● places

Word Embeddings

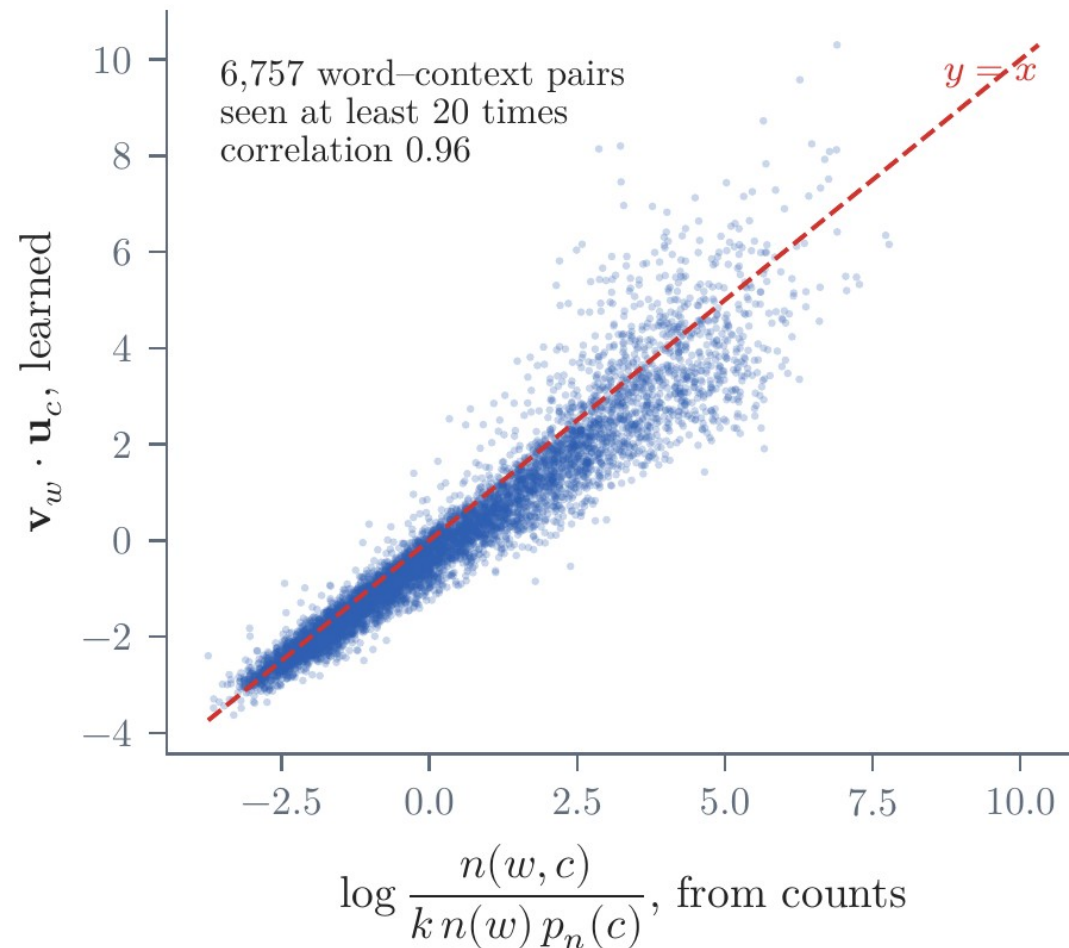
Counting and prediction meet

$$\mathbf{v}_w \cdot \mathbf{u}_c \rightarrow \log \frac{n(w, c)}{k n(w) p_n(c)} = \text{PMI}(w, c) - \log k$$

same raw material: pairs of nearby words

best score = shifted PMI

checked: correlation 0.96



Hands-on: Word Vectors

[week02_handson.ipynb, stop 2](#)

```
In [5]: text = embeddings.content_words(train) # words only, no punctuation
top_words, word_counts = embeddings.frequent_words(text, how_many=3000)
counts = embeddings.cooccurrence_counts(text, top_words, window=2)
count_vectors = embeddings.svd_vectors(embeddings.ppmi(counts), 100)

w2v = torch.load(ROOT / "checkpoints" / "week02_word2vec.pt")
w2v_words, w2v_vectors = w2v["words"], w2v["after"].numpy()

def neighbors(vectors, words, q, k=4):
    return ", ".join(w for w, s in embeddings.nearest(vectors, words, q, k))

print(f"{'':>7s} {'from counts':42s} from prediction (skip-gram)")
for q in ["mom", "red", "happy", "dog", "ran", "cake"]:
    print(f"{q:>7s} {neighbors(count_vectors, top_words, q):42s} {neighbors(w2v_vectors, w2v_words, q)}")
```

	from counts	from prediction (skip-gram)
mom	dad, mommy, mother, mum	dad, mommy, mother, mum
red	blue, yellow, green, pink	yellow, green, blue, polka
happy	refreshed, content, grateful, relieved	grateful, thankful, thanked, glad
dog	cat, rex, barks, tail	rex, barking, barks, barked
ran	walked, went, headed, jumped	jumped, rushed, grabbed, quickly
cake	soup, pie, cookies, pizza	pizza, cookies, oven, baked

watch: both routes give nearly the same neighbors

Word Embeddings

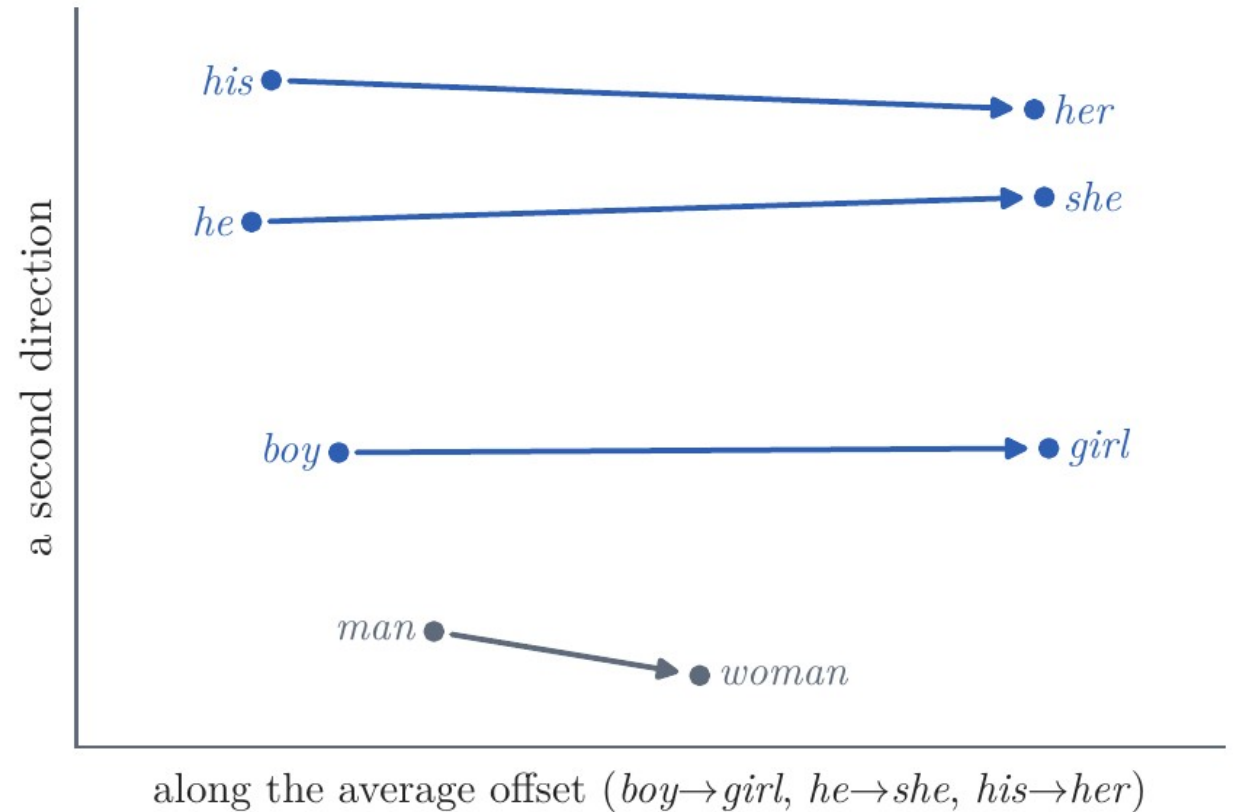
What similarity captures: relations as directions

relations are directions

$girl - boy + he \rightarrow she$

$walk + (ran - run) \rightarrow walked$

and so are stereotypes



$a : b :: c : ? \Rightarrow$ the word nearest to $\mathbf{v}_b - \mathbf{v}_a + \mathbf{v}_c$ (not a, b, c)

Word Embeddings

What similarity misses: one vector per word

we sat on the **bank** of the river

the **bank** raised its rates

→ one vector

hot / cold: cosine 0.64 — antonyms keep the same company

fix: a vector per occurrence, from its context

→ attention, next week

Break

Lecture 2: Language Modeling: From Counts to Neural Networks

1. Counting: text as a random source

- Markov and Shannon; tokens; the language model as a tree
- n-gram models; measuring a model; smoothing

2. Word embeddings: words as points in a space

- why vectors, and where they are used
- vectors from counts and from prediction (word2vec), and why they meet
- what similarity captures, and what it misses

3. Neural language models: from the bigram network to the LSTM

- the bigram network; Bengio's fixed window
- recurrent networks: backpropagation through time, vanishing gradients
- the additive path; the LSTM

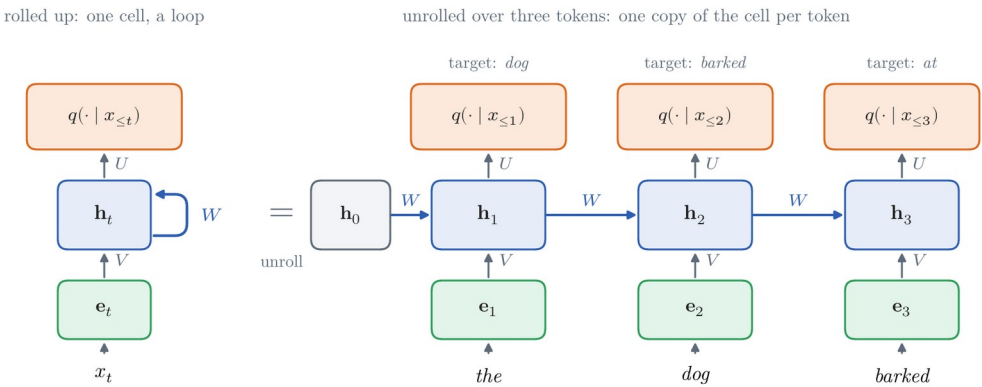
Neural Language Models

A timeline of landmark results

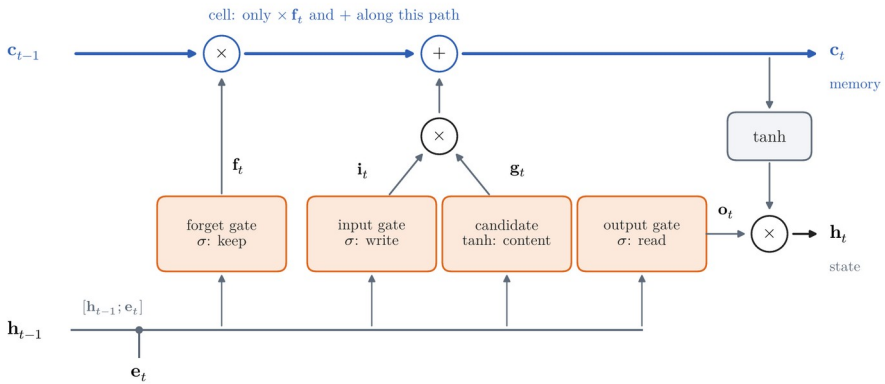
1948–2007 counting: Shannon to Google's 2 trillion words

1990 Elman: a recurrent network

1997 Hochreiter and Schmidhuber: the LSTM



a recurrent network (Elman, 1990)



the LSTM cell (1997)

Neural Language Models

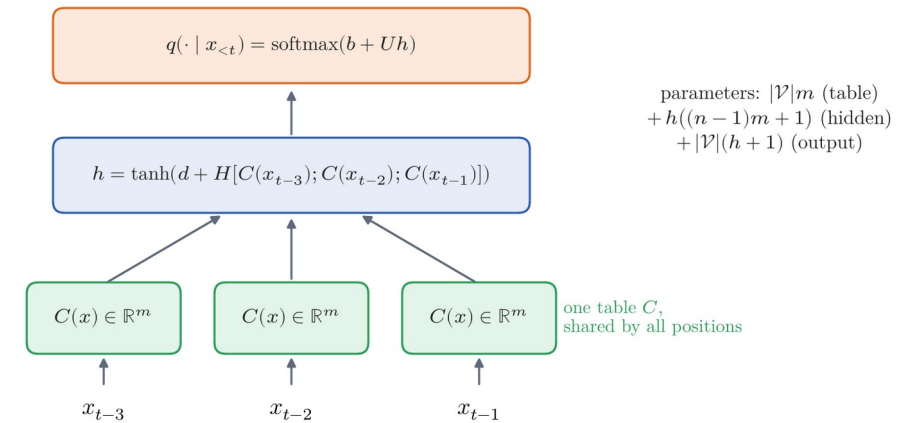
A timeline of landmark results

1948–2007 counting: Shannon to Google's 2 trillion words

1990 Elman: a recurrent network

1997 Hochreiter and Schmidhuber: the LSTM

2003 Bengio et al.: a neural language model



learned word vectors, a hidden layer, a softmax

Neural Language Models

A timeline of landmark results

1948–2007 counting: Shannon to Google's 2 trillion words

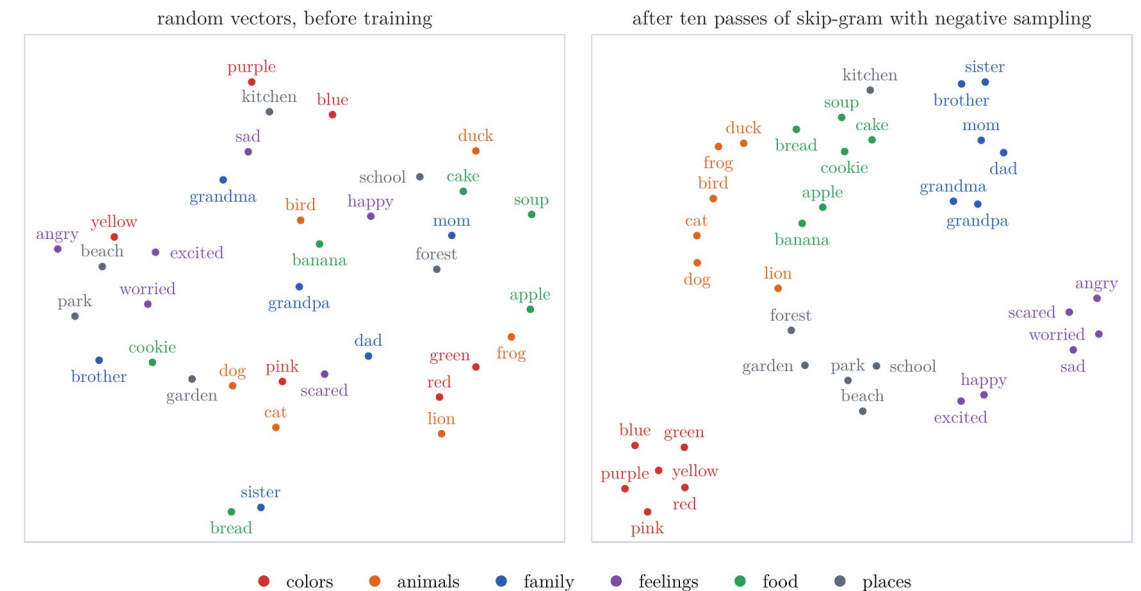
1990 Elman: a recurrent network

1997 Hochreiter and Schmidhuber: the LSTM

2003 Bengio et al.: a neural language model

2010 Mikolov: recurrent LMs beat n-grams

2013 word2vec



Neural Language Models

A timeline of landmark results

1948–2007 counting: Shannon to Google's 2 trillion words

1990 Elman: a recurrent network

1997 Hochreiter and Schmidhuber: the LSTM

2003 Bengio et al.: a neural language model

2010 Mikolov: recurrent LMs beat n-grams

2013 word2vec

2014 translation with two LSTMs; attention

2015 Karpathy's char-rnn

*once upon a time, there was an intelligent bear. he was a
wisd carefully trusted the flashith. he was a bear and a
fish. at the party, they started to walk outside and
laughed you today. max had talked to her daughter. it
felt something...*

an LSTM writing, one character at a time (trained on TinyStories, as in Karpathy's char-rnn)

Neural Language Models

A timeline of landmark results

1948–2007 counting: Shannon to Google's 2 trillion words

1990 Elman: a recurrent network

1997 Hochreiter and Schmidhuber: the LSTM

2003 Bengio et al.: a neural language model

2010 Mikolov: recurrent LMs beat n-grams

2013 word2vec

2014 translation with two LSTMs; attention

2015 Karpathy's char-rnn

2015–17 LSTMs in speech and translation products



a voice assistant

Neural Language Models

A timeline of landmark results

1948–2007 counting: Shannon to Google's 2 trillion words

1990 Elman: a recurrent network

1997 Hochreiter and Schmidhuber: the LSTM

2003 Bengio et al.: a neural language model

2010 Mikolov: recurrent LMs beat n-grams

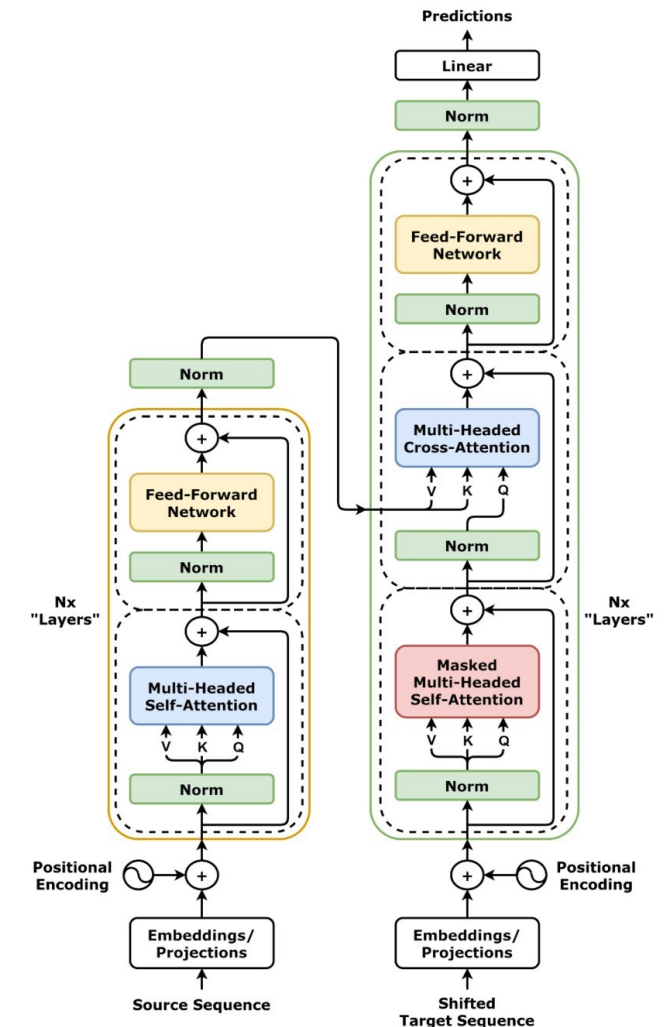
2013 word2vec

2014 translation with two LSTMs; attention

2015 Karpathy's char-rnn

2015–17 LSTMs in speech and translation products

2017 the Transformer



the Transformer

Neural Language Models

The bigram model as a network: counting becomes optimization

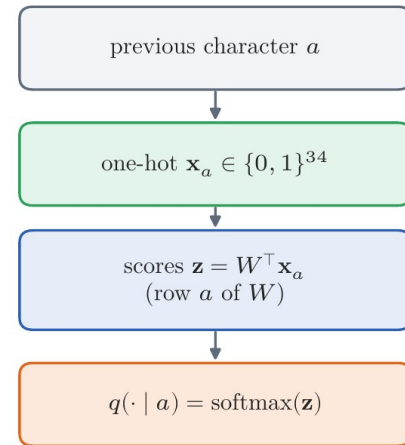
$$q(b \mid a) = \frac{\exp W_{ab}}{\sum_{b'} \exp W_{ab'}}$$

gradient descent finds the counts

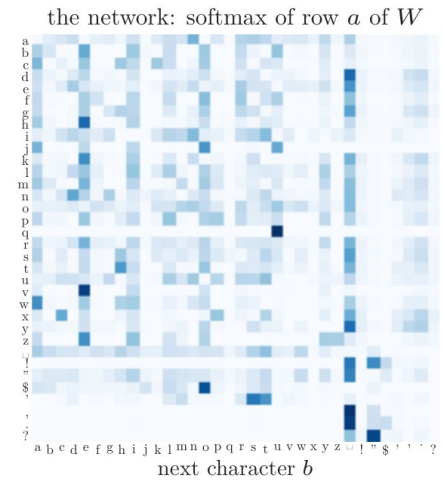
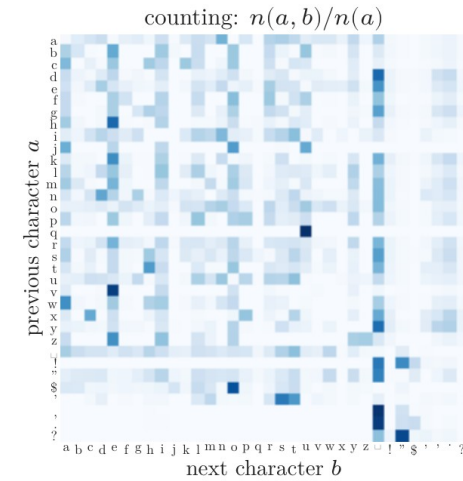
now the table is a function we can change

factor it through word vectors:

$$q(\cdot \mid v) = \text{softmax}(\mathbf{b} + U C(v))$$

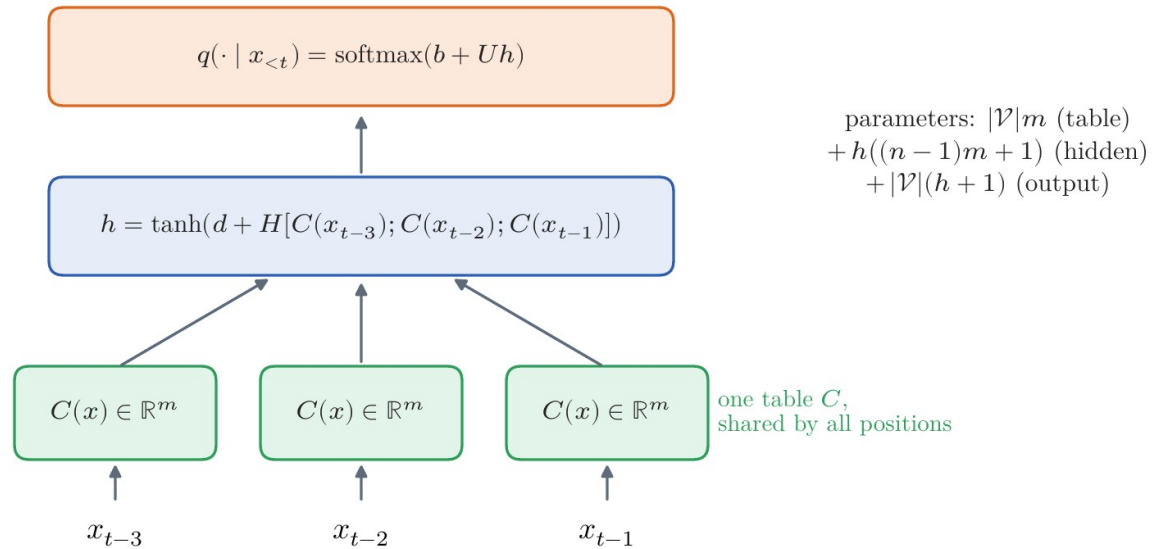


W : 34×34 , trained by gradient descent



Neural Language Models

The fixed-window model (Bengio et al., 2003)



word vectors, shared by all positions

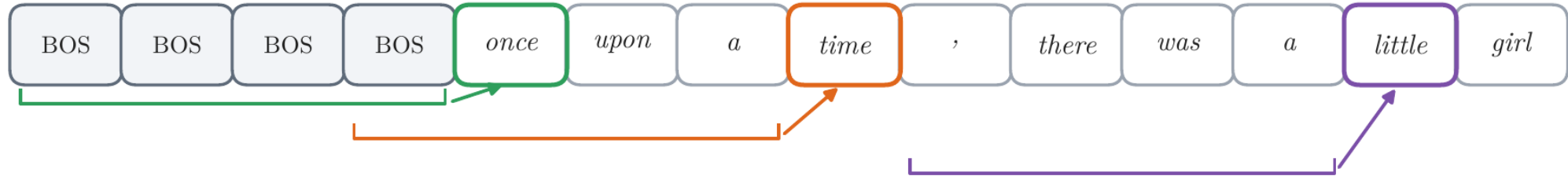
parameters grow linearly in n

$$\mathbf{h} = \tanh(\mathbf{d} + H\mathbf{e}), \quad q(\cdot | \text{window}) = \text{softmax}(\mathbf{b} + U\mathbf{h})$$

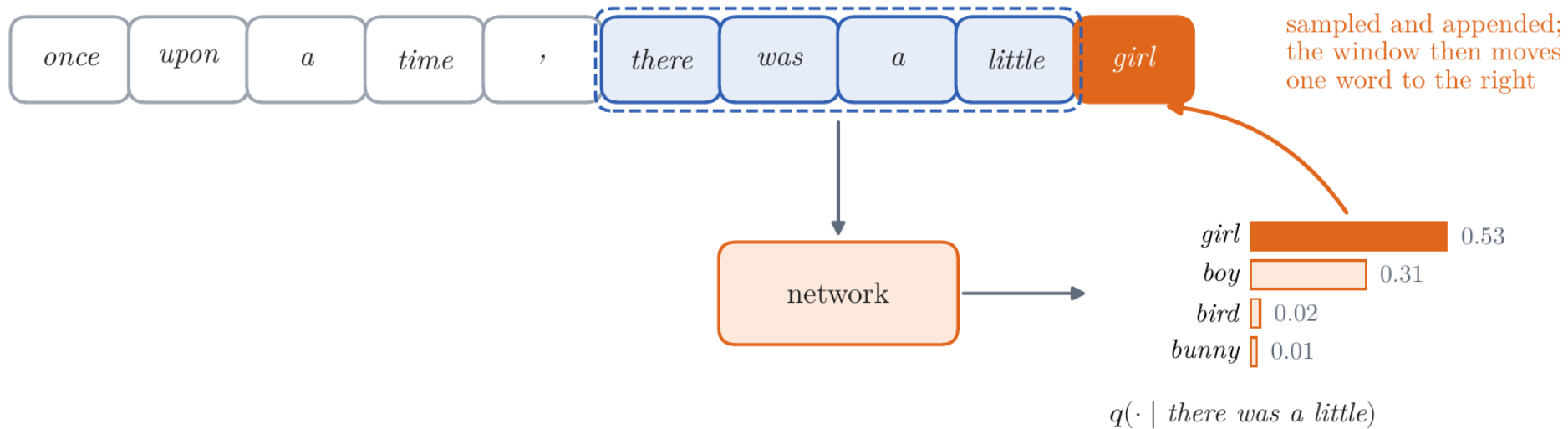
Neural Language Models

The fixed-window model (Bengio et al., 2003)

training: every position gives one example, four words and the word that follows



generation: predict from the last four words, sample, append, slide the window



Hands-on: Neural Language Models

[week02_handson.ipynb, stop 3](#)

```
In [11]: for prompt in ["she was very", "they went to the", "once upon a", "the dog wagged its"]:  
         top = neural.next_word_distribution(lm, index, vocab, prompt.split())  
         print(f"{prompt + ' ...':22s} " + ", ".join(f"{w} {p:.2f}" for w, p in top))
```

```
she was very ...      excited 0.16, happy 0.10, sad 0.06, proud 0.04, curious 0.04  
they went to the ...  park 0.23, kitchen 0.12, store 0.06, beach 0.04, airport 0.03  
once upon a ...      time 1.00, long 0.00, place 0.00, sunny 0.00, special 0.00  
the dog wagged its ... tail 0.96, friends 0.00, neck 0.00, eyes 0.00, tears 0.00
```

```
In [12]: for seed in [1, 2]:  
         print(neural.detokenize(neural.sample_words(lm, index, vocab, temperature=0.8, seed=seed)), "\n")
```

once upon a time, there was a little boy named timmy. timmy loved to play. from that day on, lily thought it was so cool. he looked around and saw a tiny mouse. he was so relieved. it was so heavy! he had to go home. he crossed the store,

once upon a time, there were two friends, squirrels, and then i will try to learn. he wanted to go home with her. she walked in the station and started to swing, mia can do it! " he said. " hi lily, " lila says. the ostrich is too.

watch: the names in the samples

Neural Language Models

What the window cannot see

once upon a time, there was a little boy named **timmy**. timmy loved to play.
from that day on, **lily** thought it was so cool.

four words of memory

share the weights across positions

carry a state

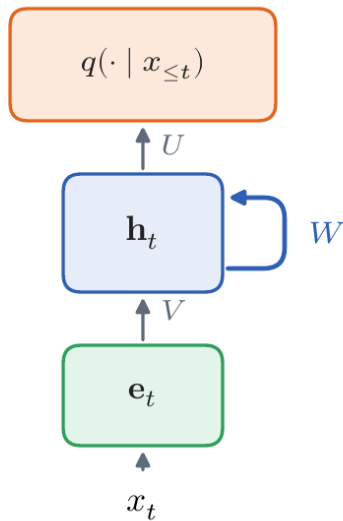
Recurrent Neural Networks

The recurrence: a machine with a state

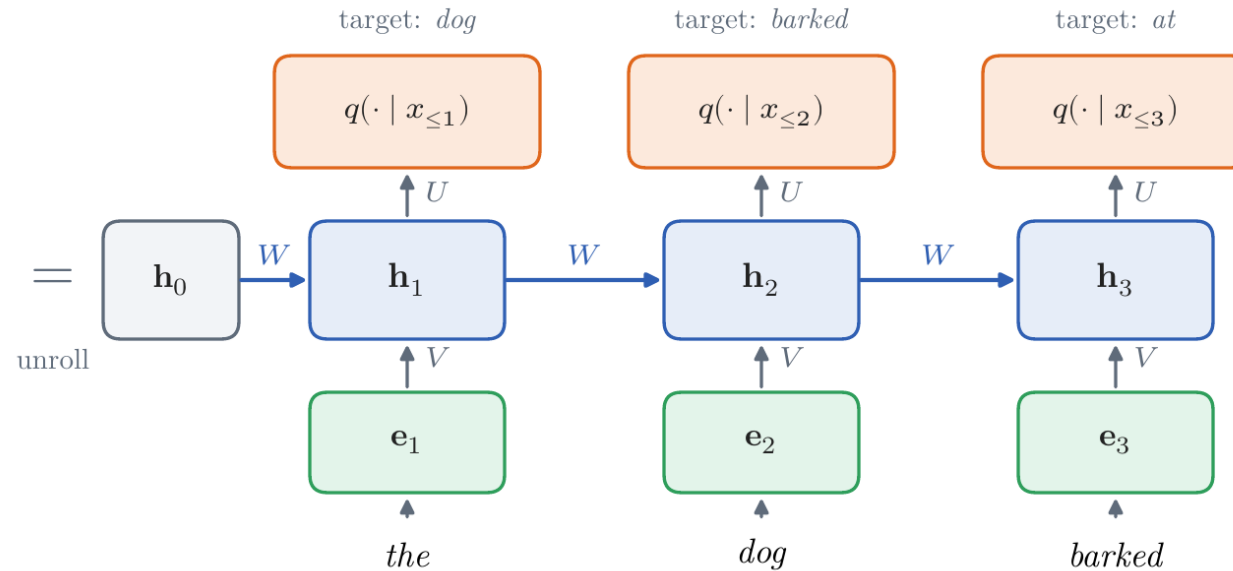
```
h = h0
for x in text:
    h = tanh(W @ h + V @ C[x] + d)    # read
    q = softmax(b + U @ h)            # predict
```

$$\mathbf{h}_t = \tanh(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{V}\mathbf{e}_t + \mathbf{d}), \quad q(\cdot | x_{\leq t}) = \text{softmax}(\mathbf{b} + \mathbf{U}\mathbf{h}_t)$$

rolled up: one cell, a loop

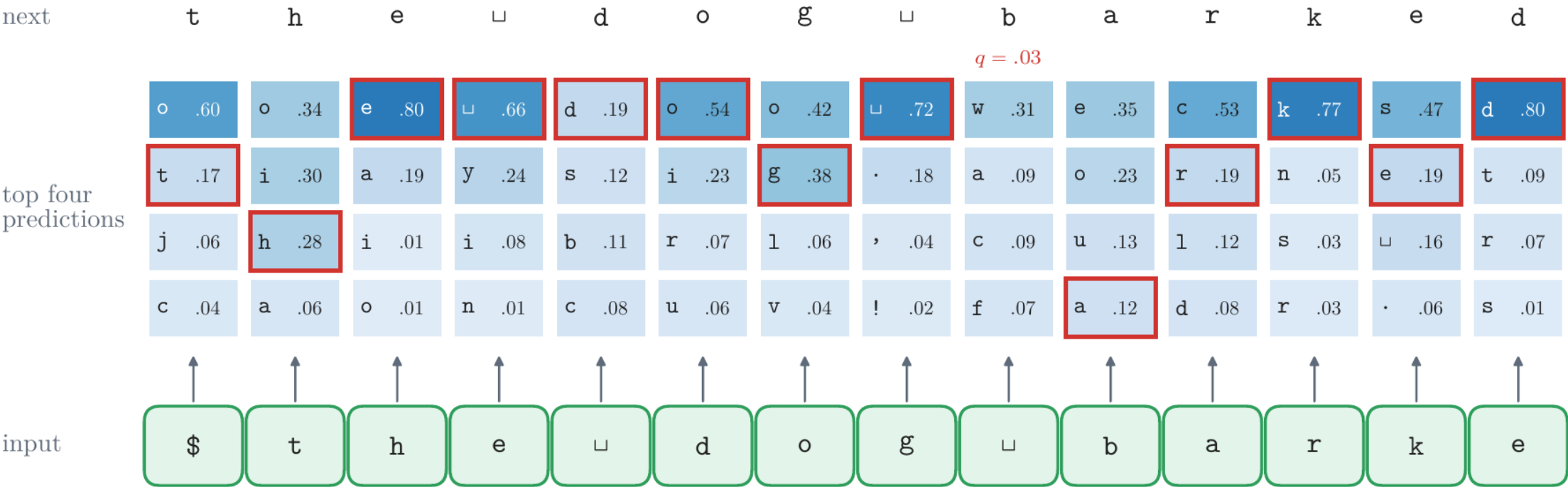


unrolled over three tokens: one copy of the cell per token



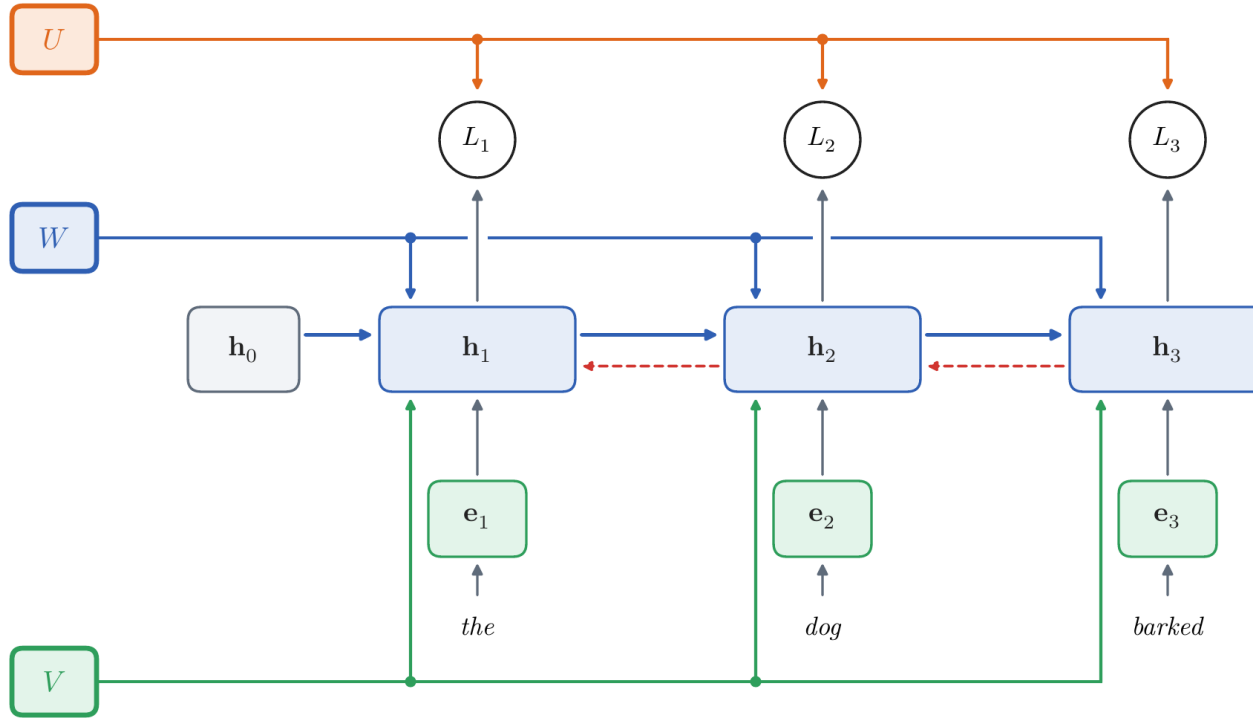
Recurrent Neural Networks

One character at a time



Recurrent Neural Networks

Training: backpropagation through time



shared leaves: gradients add

$$\frac{\partial L}{\partial W} = \sum_t \left(\text{gradient through the use of } W \text{ at step } t \right)$$

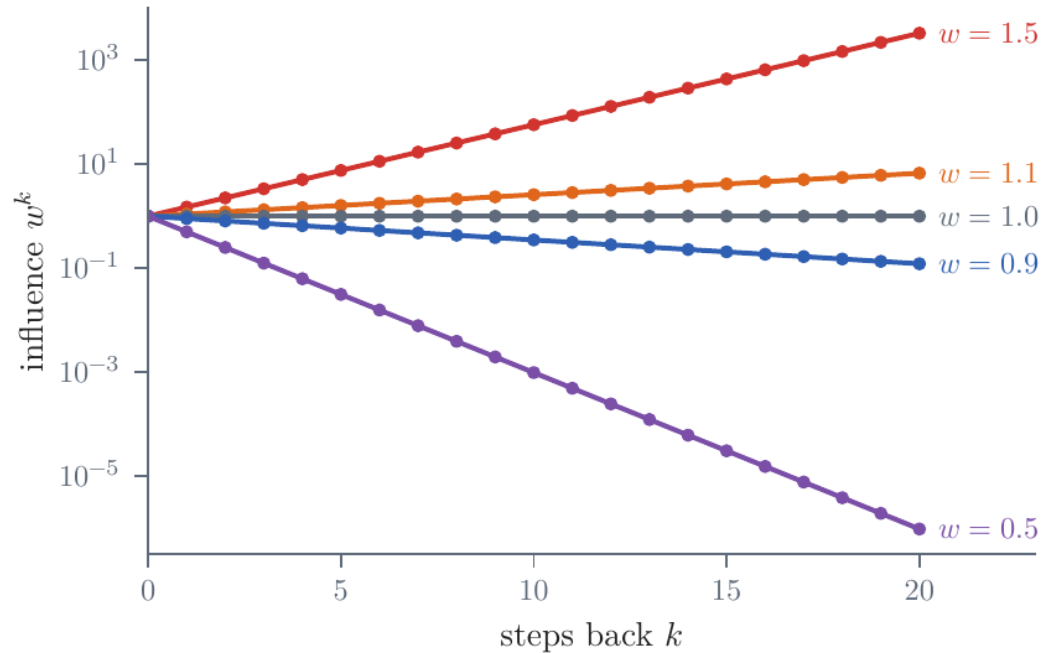
backwards in time

$$\frac{\partial L}{\partial \mathbf{h}_t} = \frac{\partial L_t}{\partial \mathbf{h}_t} + \left(\frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t} \right)^\top \frac{\partial L}{\partial \mathbf{h}_{t+1}}$$

Recurrent Neural Networks

Why the plain network forgets

$$h_t = w h_{t-1} + x_t \Rightarrow \frac{\partial h_t}{\partial h_{t-k}} = w^k : \quad 0.5^{10} \approx 0.001, \quad 1.5^{10} \approx 58$$



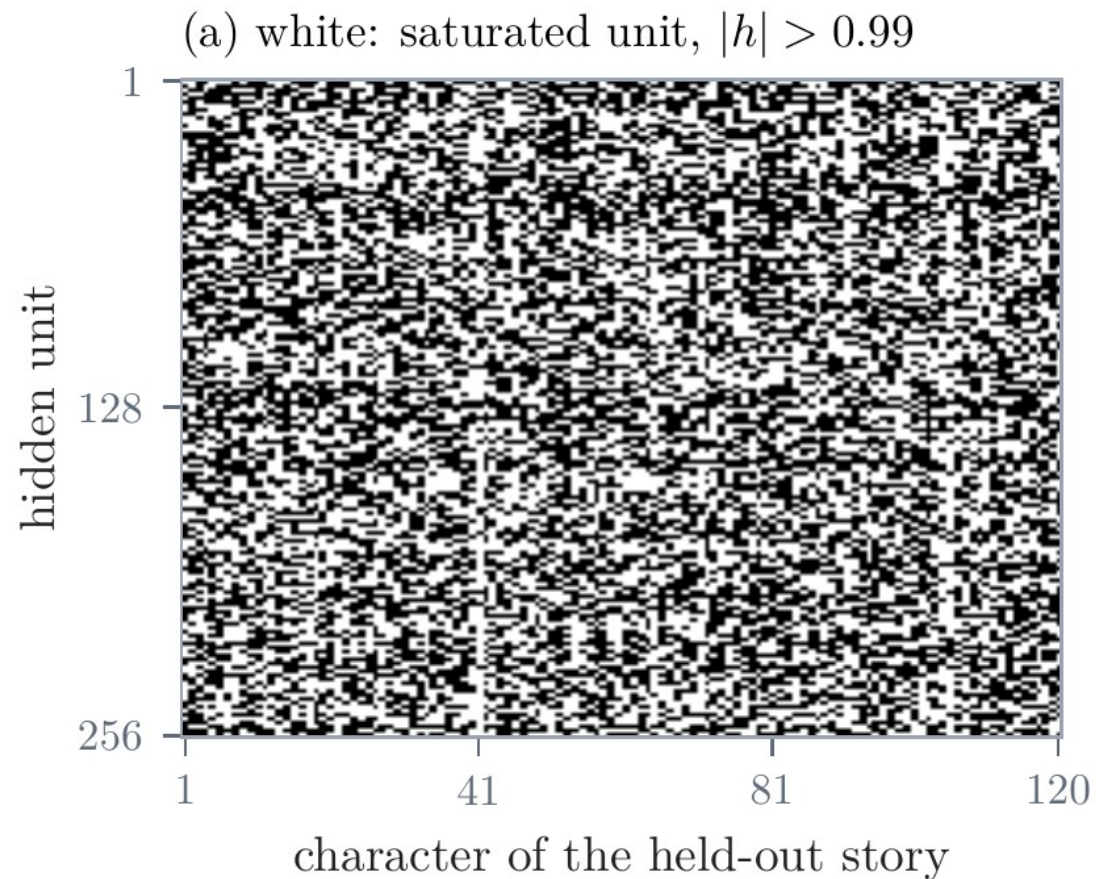
$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-k}} = \prod_{j=t-k+1}^t \text{diag}(1 - \mathbf{h}_j^2) W$$

exploding: clip the gradient

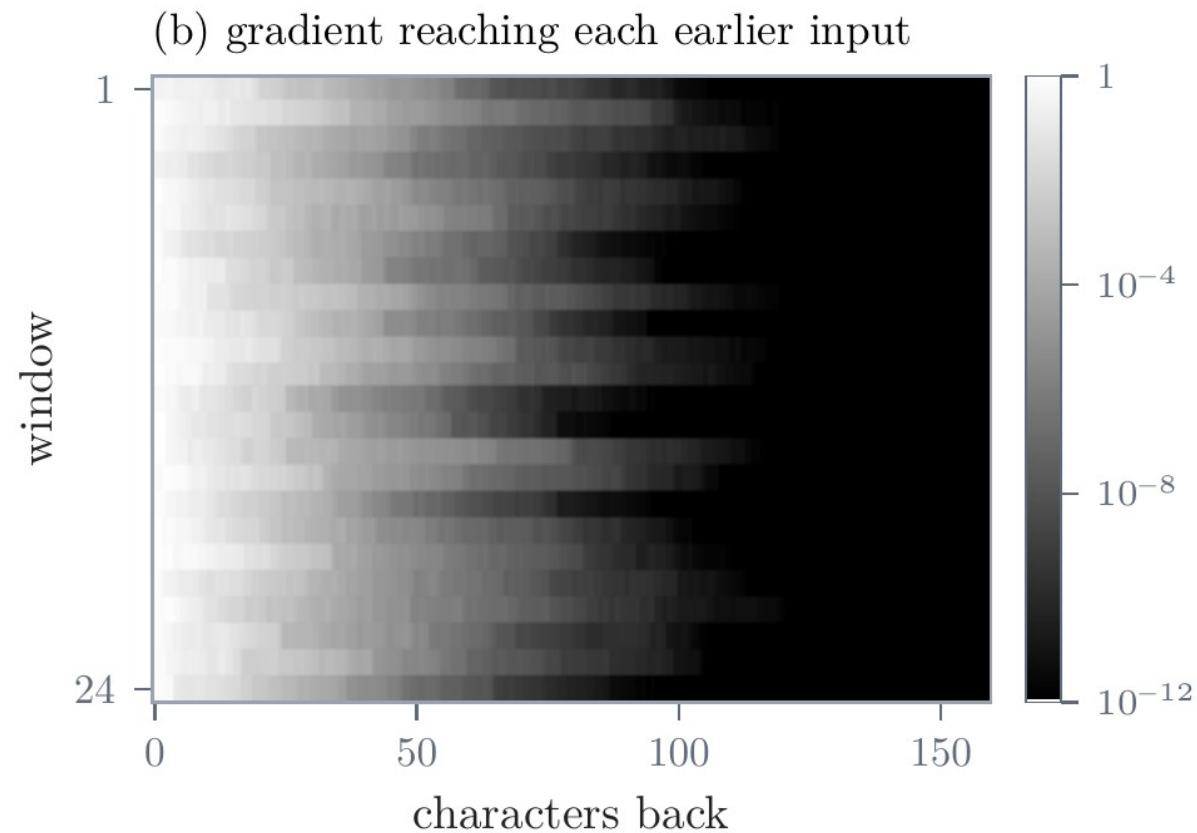
vanishing: no signal from far back

Recurrent Neural Networks

Seeing the gradient vanish



white: saturated unit, slope ≈ 0



gradient reaching each earlier character

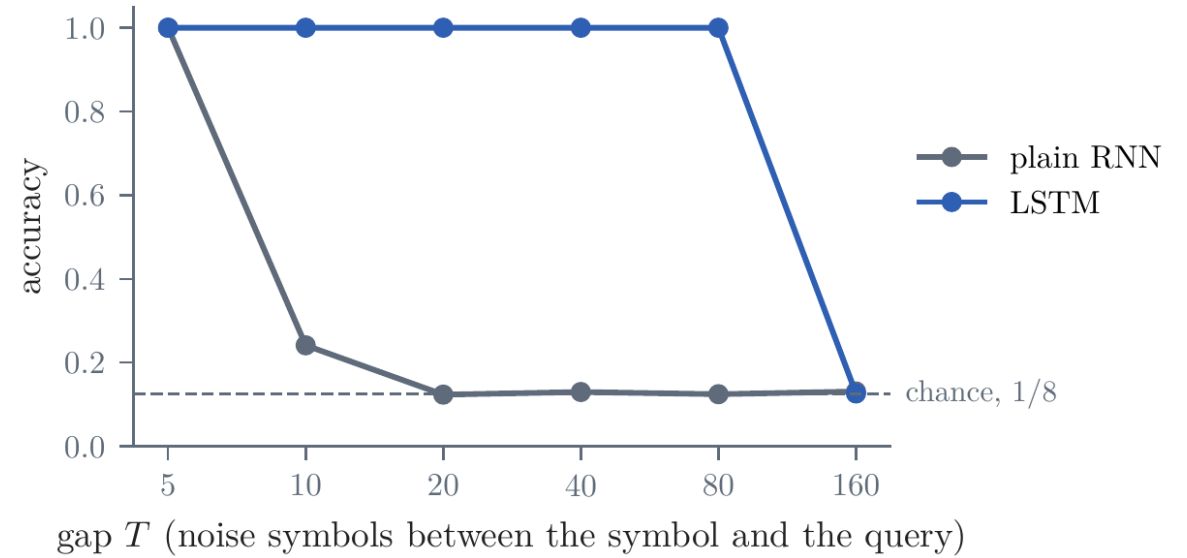
Recurrent Neural Networks

A memory test

5 12 9 14 11 8 15 13 10 ? → 5

store the key, keep it, read it

plain RNN: chance from $T = 20$



Recurrent Neural Networks

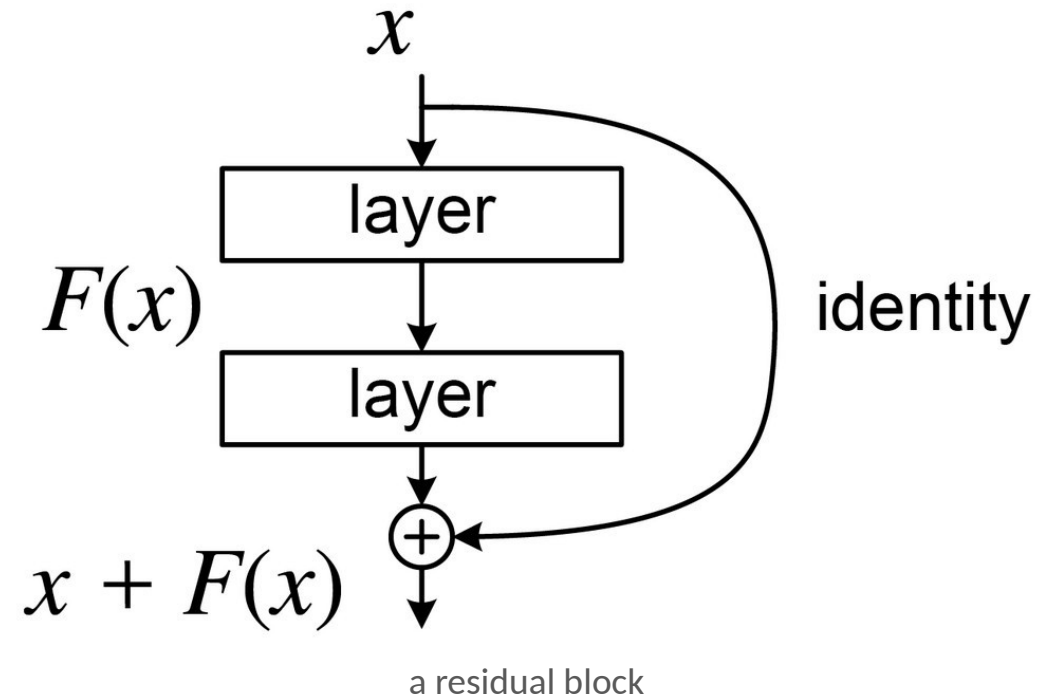
The fix: an additive path

LSTM (1997), across time: $\mathbf{c}_t = \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t$

ResNet (2015), across depth: $\mathbf{x}_{l+1} = \mathbf{x}_l + F(\mathbf{x}_l)$

add, don't rewrite: the gradient passes through

highway networks, ResNet, every Transformer layer



Hands-on: Recurrent Networks

[week02_handson.ipynb, stop 4](#)

```
In [15]: for T in [0.5, 0.8, 1.2]:  
         print(f"T = {T}:", neural.sample_chars(models["lstm512"], chars, max_chars=300, temperature=T, seed=1), "\n")
```

T = 0.5: one day, a little girl was walking around the park. she was so happy that she had learned that it's important to listen to learn new things around her for bed. one day, lily's mom said they would take the toy car in the box and the potatoes. they wanted to keep it safe and warm. they had been still

T = 0.8: one day, a little girl watched as a big dog running toys and scattered rover. they saw a big red ball in the sand. they hopped and saw it was a good spot. the moral of the story is that it's okay to be kind and grateful for the drink!

T = 1.2: one day molly and her mom found a almost boat around her house. one day, lily went for a walk in the park when the swall had wander of it.

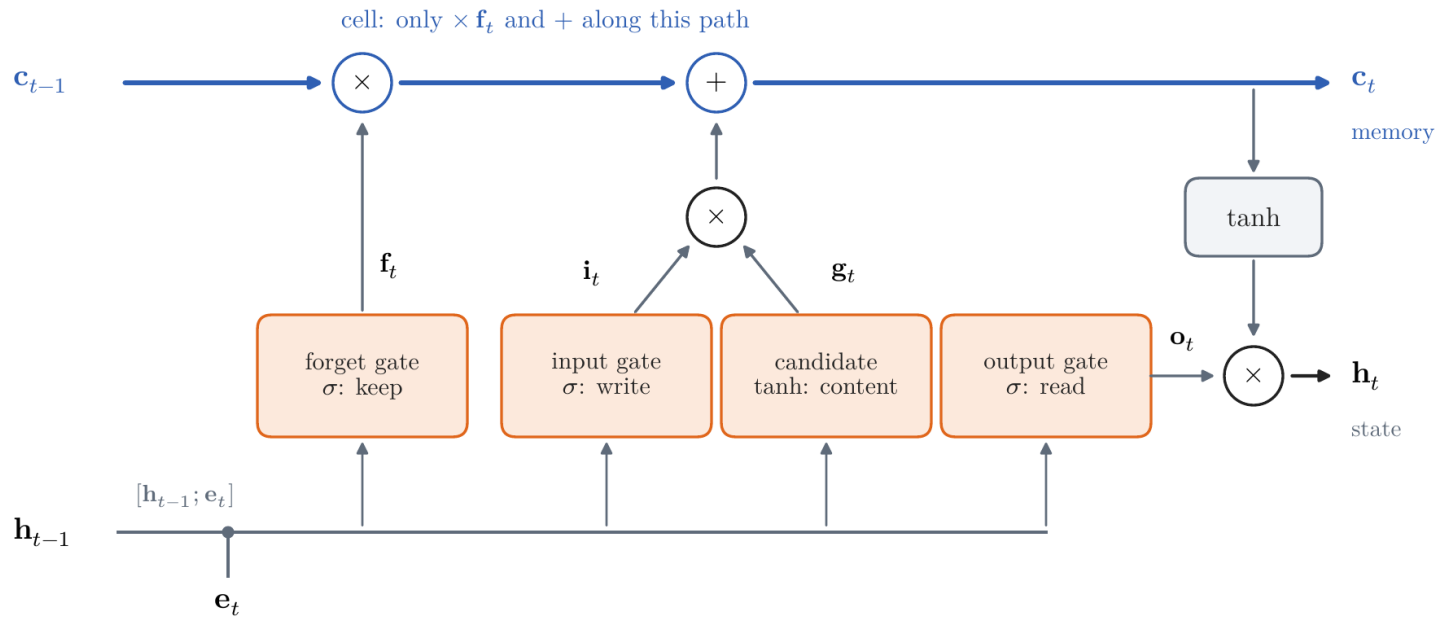
an example with T = 8: [5, 11, 12, 8, 15, 9, 11, 13, 15, 16] (16 is the query) -> answer 5

T = 20: plain RNN right 12% of the time, LSTM 100% (chance 12%; 32 s)

watch: plain RNN at chance, LSTM always right

Long Short-Term Memory

Gated memory: write, keep, read



write: input gate

keep: forget gate

read: output gate

$$\mathbf{f}_t = \sigma(W_f \mathbf{h}_{t-1} + V_f \mathbf{e}_t + \mathbf{b}_f) \in (0, 1)^h$$

each gate: a sigmoid per unit, between 0 and 1

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

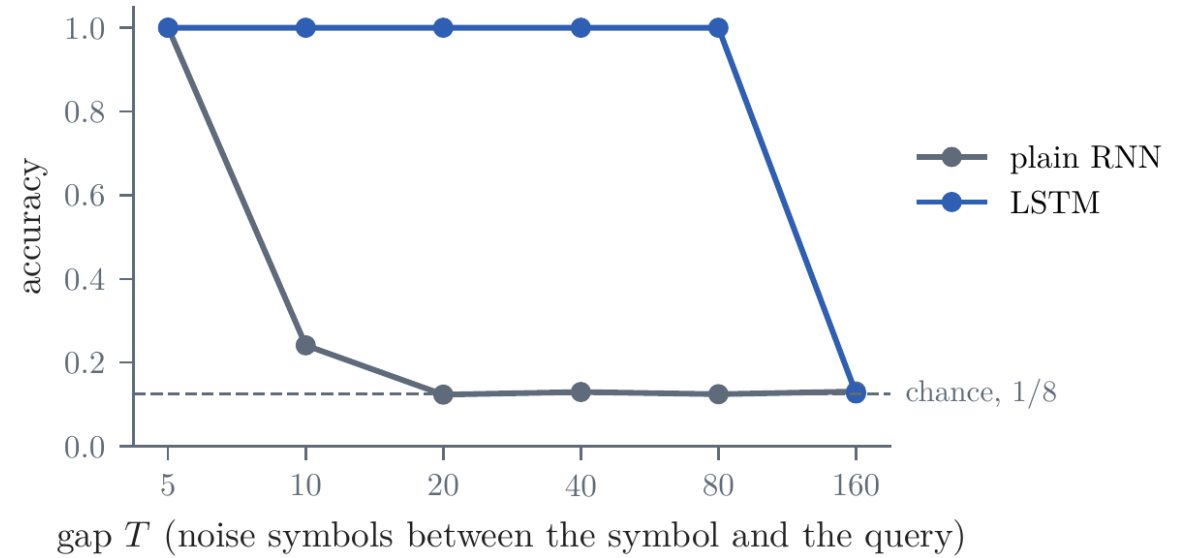
Long Short-Term Memory

Why it trains

$$\mathbf{c}_t = \mathbf{c}_{t-1} + (\text{new content}) \quad \Rightarrow \quad \frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} = I$$

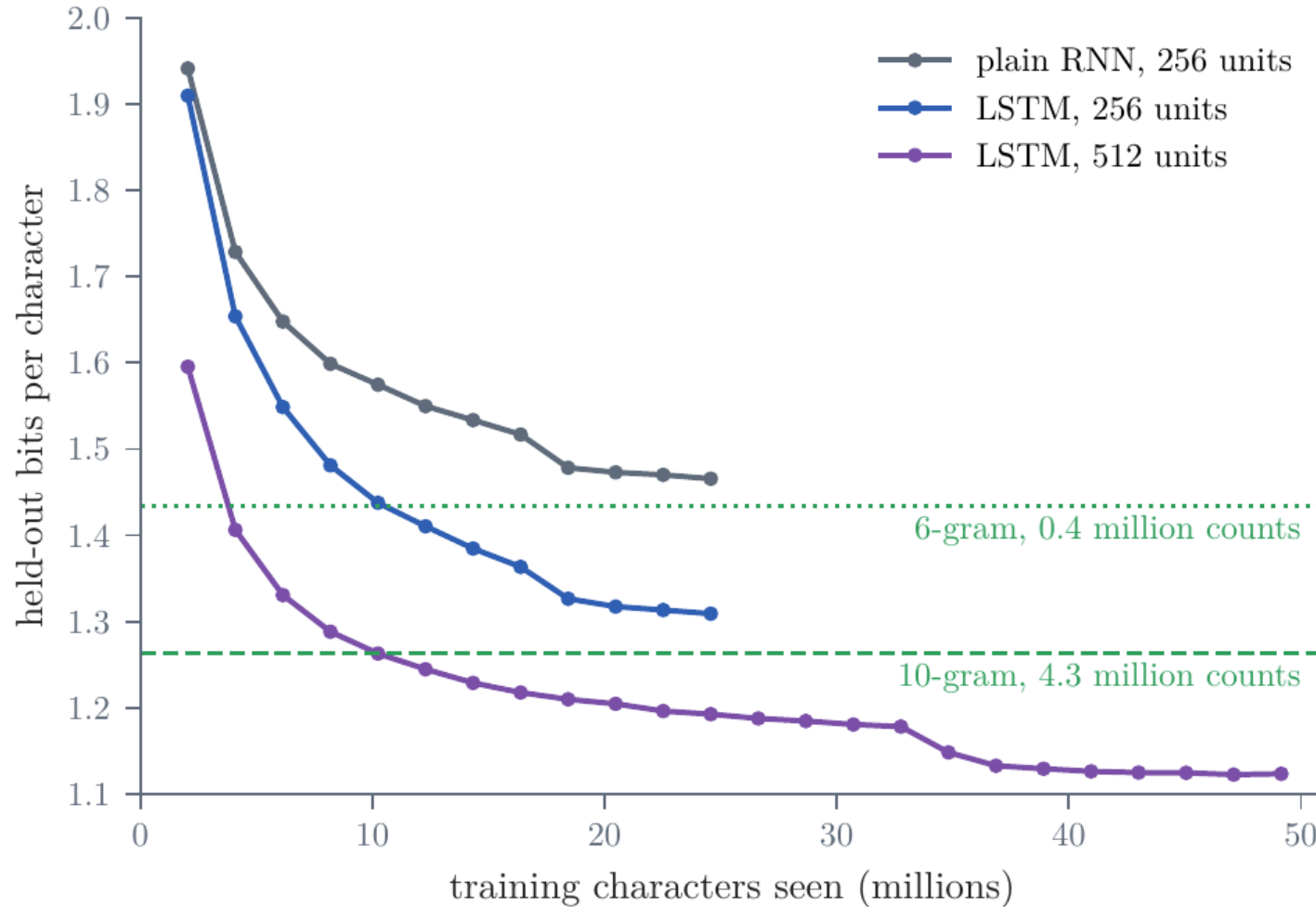
forget gates near 1: the gradient survives

start them near 1



Long Short-Term Memory

The LSTM against counting



same size: far better

keeps improving

512 units: 1.12 bits per character

Long Short-Term **Memory**

Watching it learn

0 steps *yuj?nvfaslgt"ykzir hfd*

100 steps *yucind as gome, in the pupptle in the too ba ont cougerte a saw hwaws*

1,000 steps *yucked as here, in the park. every happy, back to the barras and when*

3,000 steps *once upon a time, there was an incredible back to the barras away.*

512 units *once upon a time, there was an intelligent bear. he was a wisd carefully tr*

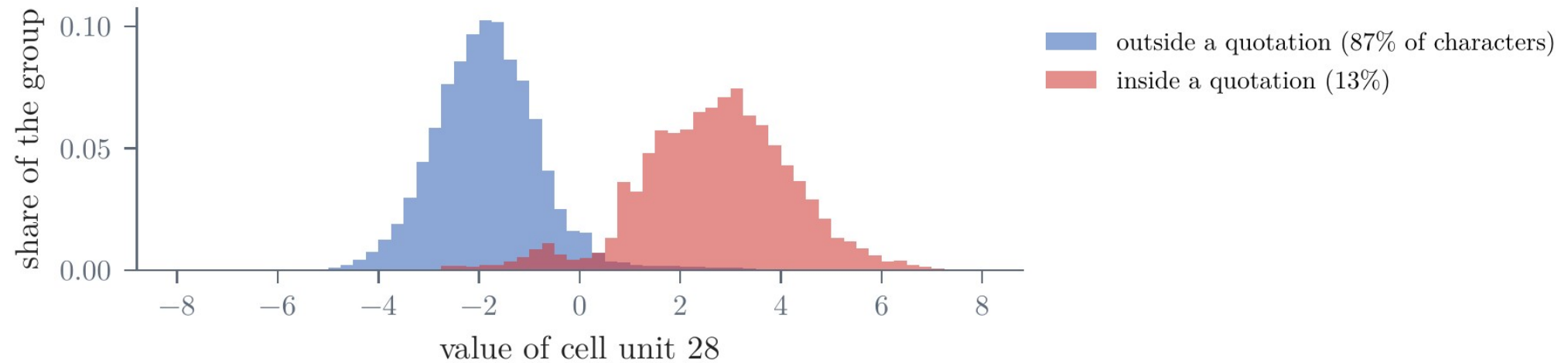
Long Short-Term Memory

What the cells track: a quotation cell

(a)

the man replied, 'i lost my wallet with all my money in it. i feel guilty for being so careless.' timmy wanted to help the man, so he asked, 'what's your name and phone number? maybe someone will find your wallet and call you.' the man smiled and gave timmy his phone number. later that day, timmy's

(b)



Long Short-Term **Memory**

Words instead of characters

test perplexity

Kneser–Ney 4-gram three words

24.6

fixed window four words

26.9

LSTM the whole story so far

23.1

Long Short-Term **Memory**

What recurrence leaves unsolved

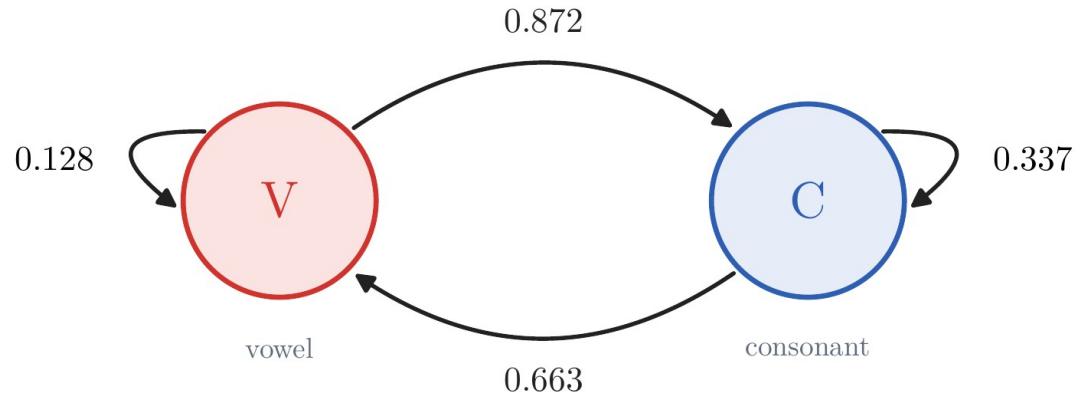
one vector holds everything

one step at a time

→ attention (Week 3), the Transformer (Week 4)

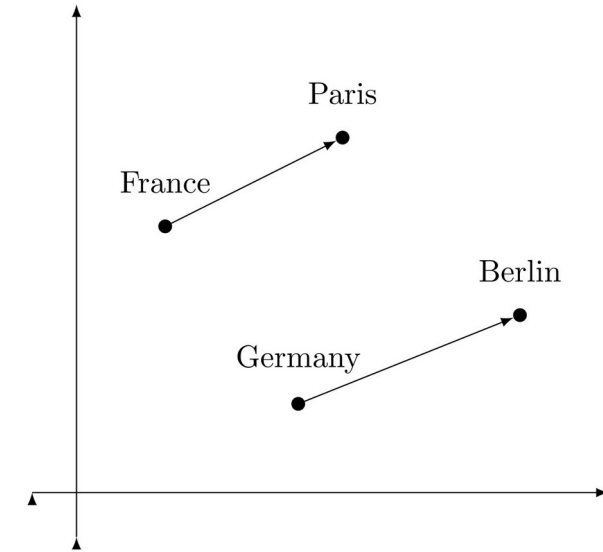
What Carries Forward

Two ideas



text is a random source

chain rule, log loss on held-out text



words are points in a space

every model starts with a table of vectors

next: attention and the Transformer