CS11-711 Advanced NLP

Language and Sequence Modeling I

Sean Welleck

Carnegie Mellon University
Language Technologies Institute

https://cmu-l3.github.io/anlp-spring2025/

Some slides adapted from Graham Neubig from Fall 2024

Types of Sequential Prediction Problems

Types of Prediction:
Binary, Multi-class, Structured

Types of Prediction:
Binary, Multi-class, Structured

- Two classes (**binary classification**)

Types of Prediction: Binary, Multi-class, Structured

- Two classes (**binary classification**)

positive

I hate this movie → negative

Types of Prediction:
Binary, Multi-class, Structured

- Two classes (**binary classification**)

I hate this movie

positive
negative

- Multiple classes (**multi-class classification**)

Types of Prediction: Binary, Multi-class, Structured

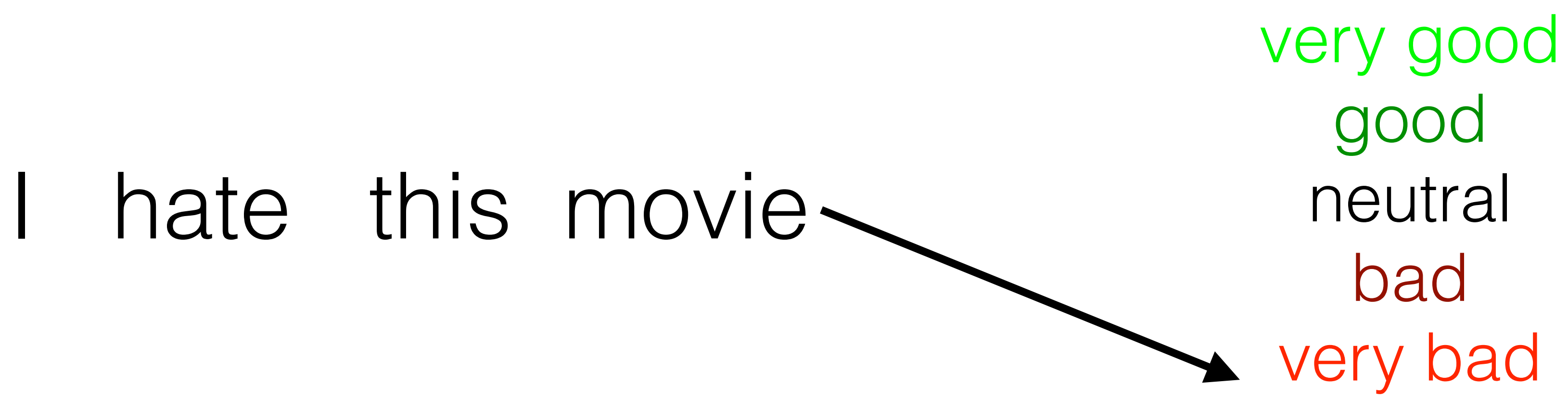
- Two classes (**binary classification**)

I hate this movie

positive

negative

- Multiple classes (**multi-class classification**)



I hate this movie

very good

good

neutral

bad

very bad

Types of Prediction: Binary, Multi-class, Structured

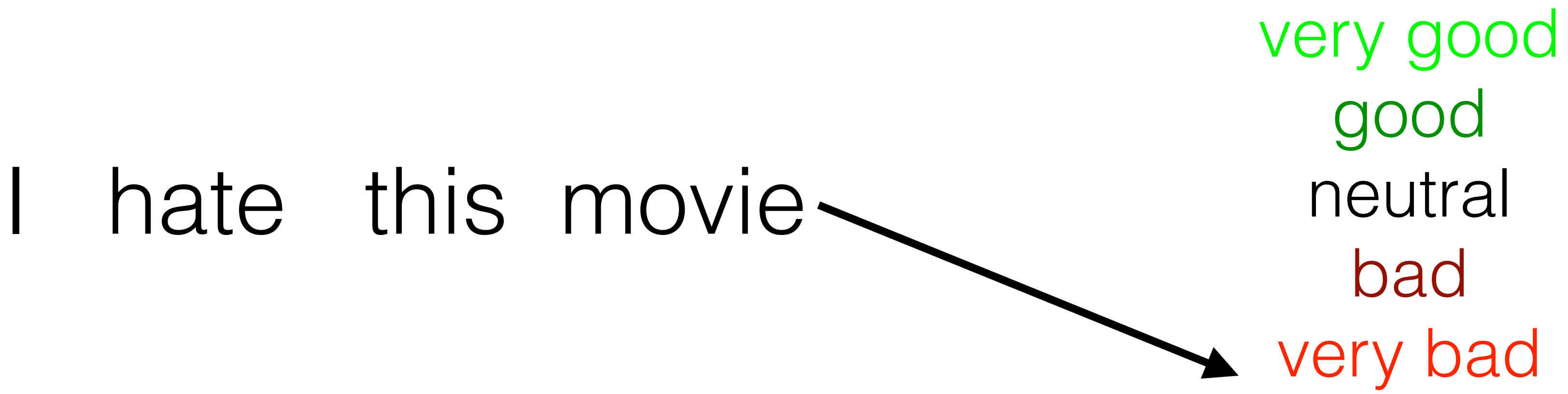
- Two classes (**binary classification**)

I hate this movie

positive

negative

- Multiple classes (**multi-class classification**)



I hate this movie

very good

good

neutral

bad

very bad

- Exponential/infinite labels (**structured prediction**)

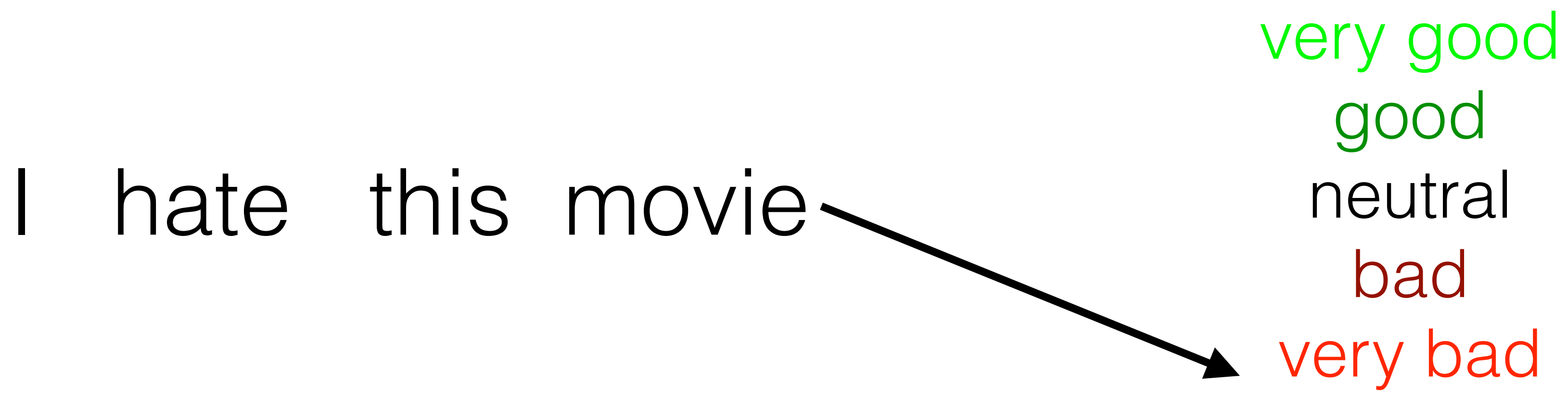
Types of Prediction: Binary, Multi-class, Structured

- Two classes (**binary classification**)

I hate this movie → negative

positive
negative

- Multiple classes (**multi-class classification**)



I hate this movie → very bad

very good
good
neutral
bad
very bad

- Exponential/infinite labels (**structured prediction**)

I hate this movie → PRP VBP DT NN

Types of Prediction:
Binary, Multi-class, Structured

- Two classes (**binary classification**)

positive
I hate this movie → negative

- Multiple classes (**multi-class classification**)

very good
good
neutral
bad
I hate this movie → very bad

- Exponential/infinite labels (**structured prediction**)

I hate this movie → PRP VBP DT NN

I hate this movie → *kono eiga ga kirai*

Types of Prediction: Unconditioned vs. Conditioned

Types of Prediction: Unconditioned vs. Conditioned

- **Unconditioned Prediction:** Predict the probability of a single variable $P(X)$

Types of Prediction: Unconditioned vs. Conditioned

- **Unconditioned Prediction:** Predict the probability of a single variable $P(X)$
- **Conditioned Prediction:** Predict the probability of an output variable given an input $P(Y|X)$

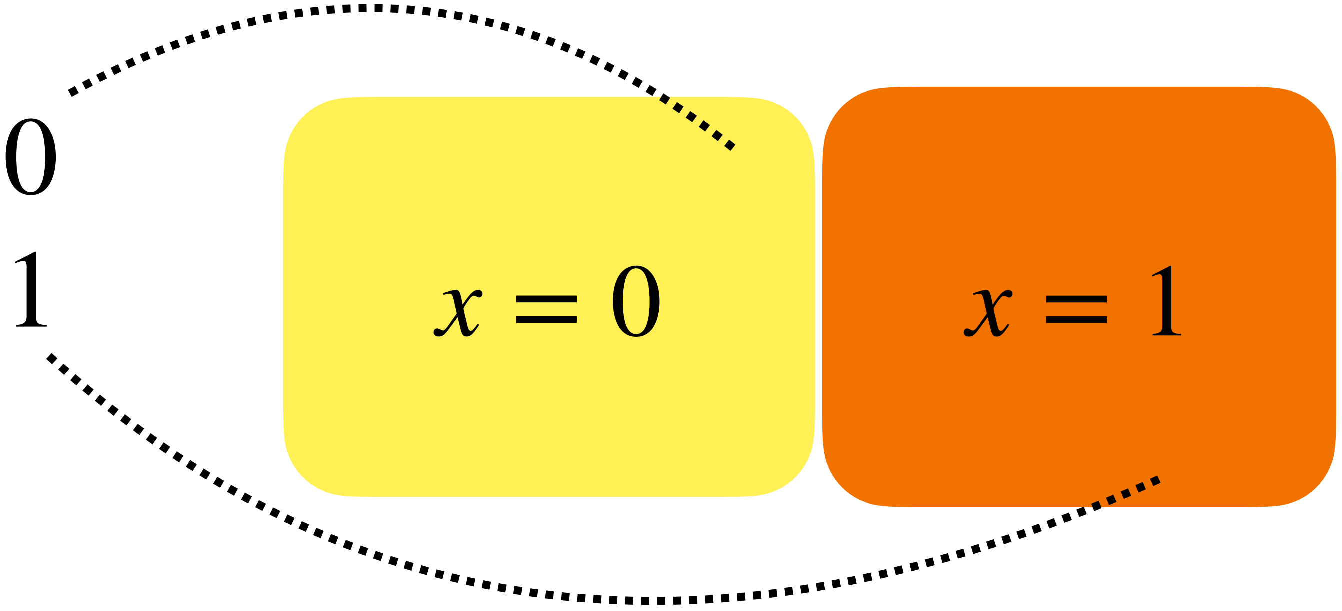
Language Modeling

What is a language model?

- A language model is a probability distribution over all sequences
    - $P(X)$

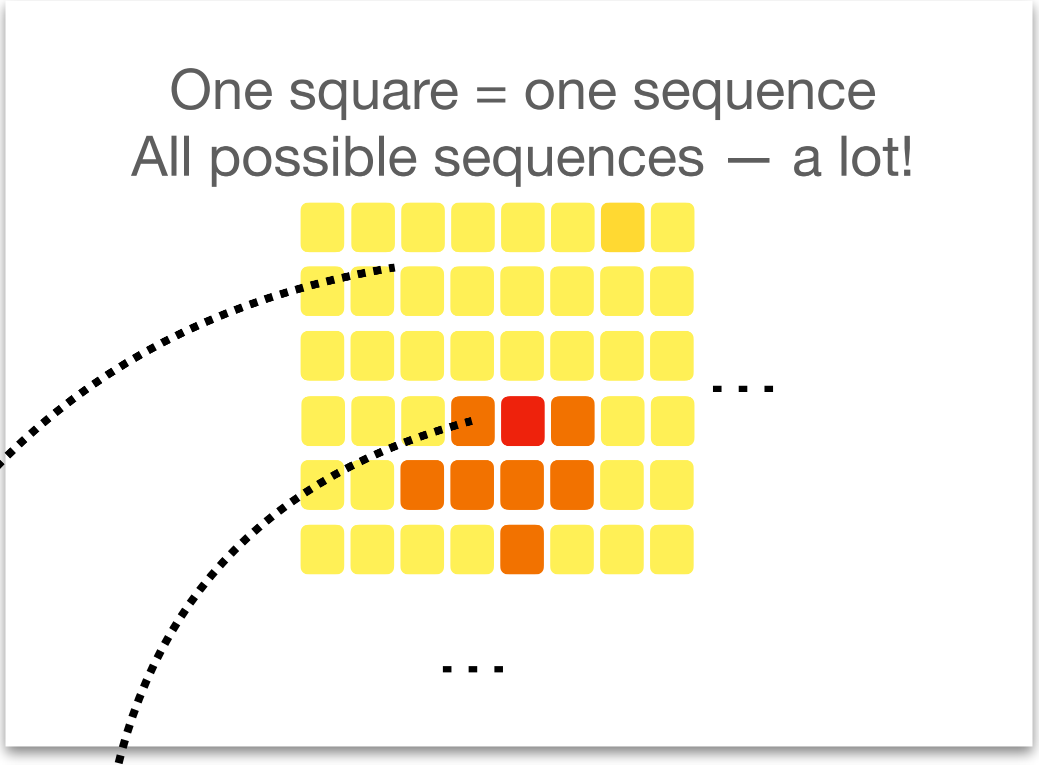
What is a language model?

- A language model is a probability distribution over all sequences
  - $P(X)$
- Example probability distribution: **biased coin**
  - $P(X) = \begin{cases} 0.4 & x \textbf{ is } 0 \\ 0.6 & x \textbf{ is } 1 \end{cases}$



What is a language model?

- A language model is a probability distribution over all sequences
  - $P(X)$
- Example language model:
  - $P(X) =$ 0.000013 if x is `a.`
    0.000001 if x is `aa.`
    …
    0.019100 if x is `a cat sat.`
    …



One square = one sequence
All possible sequences — a lot!

What can we do with language models?

What can we do with language models?

- **Score** sequences:

P(Jane went to the store .) → high
P(store to Jane went the .) → low

What can we do with language models?

- **Score** sequences:

  P(Jane went to the store .) → high
  P(store to Jane went the .) → low

- **Generate** sequences:

What can we do with language models?

- **Score** sequences:

P(Jane went to the store .) → high
P(store to Jane went the .) → low

- **Generate** sequences:

$$\hat{x} \sim P(X)$$

What can we do with language models?

- **Conditional generation**: condition on an input context

$$\hat{x}_{t+1:T} \sim P(X_{t+1:T} | x_{1:t})$$

What can we do with language models?

- **Conditional generation**: condition on an input context

$$\hat{x}_{t+1:T} \sim P(X_{t+1:T} \mid x_{1:t})$$

- Machine translation:
  - Context: sentence in English
  - Continuation: sentence in Japanese

What can we do with language models?

- **Conditional generation**: condition on an input context

$$\hat{x}_{t+1:T} \sim P(X_{t+1:T} | x_{1:t})$$

- Machine translation:
  - Context: sentence in English
  - Continuation: sentence in Japanese
- General task:
  - Context: instructions, examples, start of output
  - Continuation: output

What can we do with language models?

What can we do with language models?

- **Answer questions**

What can we do with language models?

- **Answer questions**
  - *Score* possible multiple choice answers

What can we do with language models?

- **Answer questions**
  - *Score* possible multiple choice answers
  - *Generate* a continuation of a question prompt

What can we do with language models?

- **Answer questions**
    - *Score* possible multiple choice answers
    - *Generate* a continuation of a question prompt
- **Classify text**

What can we do with language models?

- **Answer questions**
    - *Score* possible multiple choice answers
    - *Generate* a continuation of a question prompt
- **Classify text**
    - *Score* the text conditioned on a label

What can we do with language models?

- **Answer questions**
  - *Score* possible multiple choice answers
  - *Generate* a continuation of a question prompt
- **Classify text**
  - *Score* the text conditioned on a label
  - *Generate* a label given a classification prompt

What can we do with language models?

- **Answer questions**
  - *Score* possible multiple choice answers
  - *Generate* a continuation of a question prompt
- **Classify text**
  - *Score* the text conditioned on a label
  - *Generate* a label given a classification prompt
- **Correct grammar**

What can we do with language models?

- **Answer questions**
  - *Score* possible multiple choice answers
  - *Generate* a continuation of a question prompt
- **Classify text**
  - *Score* the text conditioned on a label
  - *Generate* a label given a classification prompt
- **Correct grammar**
  - *Score* each word and replace low-scoring ones

What can we do with language models?

- **Answer questions**
  - *Score* possible multiple choice answers
  - *Generate* a continuation of a question prompt
- **Classify text**
  - *Score* the text conditioned on a label
  - *Generate* a label given a classification prompt
- **Correct grammar**
  - *Score* each word and replace low-scoring ones
  - *Generate* a grammatical output

What can we do with language models?

- **Answer questions**
  - *Score* possible multiple choice answers
  - *Generate* a continuation of a question prompt
- **Classify text**
  - *Score* the text conditioned on a label
  - *Generate* a label given a classification prompt
- **Correct grammar**
  - *Score* each word and replace low-scoring ones
  - *Generate* a grammatical output
- …

Auto-regressive Language Models

$$P(X) = \prod_{t=1}^{T} P\left(x_t \mid x_1, \ldots, x_{t-1}\right)$$

Auto-regressive Language Models

$$P(X) = \prod_{t=1}^{T} P\left(\underset{\substack{\uparrow \\ \text{Next Token}}}{\underline{x_t}} \mid x_1, \ldots, x_{t-1}\right)$$

Auto-regressive Language Models

$$P(X) = \prod_{t=1}^{T} P\left(\underbrace{x_t}_{\text{Next Token}} \mid \underbrace{x_1, \ldots, x_{t-1}}_{\text{Context}}\right)$$

Auto-regressive Language Models

$$P(X) = \prod_{t=1}^{T} P\left(\underbrace{x_t}_{\text{Next Token}} \mid \underbrace{x_1, \ldots, x_{t-1}}_{\text{Context}}\right)$$

Decomposes sequence modeling into next-token modeling

Auto-regressive Language Models

$$P(X) = \prod_{t=1}^{T} P\left(\underbrace{x_t}_{\text{Next Token}} \mid \underbrace{x_1, \ldots, x_{t-1}}_{\text{Context}}\right)$$

Decomposes sequence modeling into next-token modeling

Key question: modeling

$$P\left(x_t \mid x_1, \ldots, x_{t-1}\right)$$

Roadmap

- Bigram models
- Ngram models
- Feedforward neural language model
- Practical deep learning considerations

Next 2 lectures

- Recurrent models
- Transformers

Bigram models

$$P(X) \approx \prod_{t=1}^{T} p_{\theta}\left(\underbrace{x_t}_{\text{Next Token}} \mid \underbrace{x_{t-1}}_{\text{1-token context}}\right)$$

Code:
https://github.com/cmu-l3/anlp-spring2025-code/blob/main/03_lm_fundamentals/lm_basics_bigrams.ipynb

Training language models

Problem setup

- Goal: model a *data distribution,* i.e. $p_\theta \approx p_{data}$

Training language models

Problem setup

- Goal: model a *data distribution,* i.e. $p_\theta \approx p_{data}$
- We only have a dataset of samples from p_{data}:
  - $D = \{x_n\}_{n=1}^{N}$

Training language models

Problem setup

- Goal: model a *data distribution,* i.e. $p_\theta \approx p_{data}$
- We only have a dataset of samples from p_{data}:
  - $D = \{x_n\}_{n=1}^N$
- Split the dataset into training, dev, and test sets

Training Bigram Models

Training Bigram Models

- Set next-token probabilities based on how often each token x_t appears after x_{t-1} in the training dataset:

$$p(x_t \mid x_{t-1}) = \frac{\text{count}(x_{t-1}, x_t)}{\sum_{x'} \text{count}(x_{t-1}, x')}$$

Training Bigram Models

- Set next-token probabilities based on how often each token x_t appears after x_{t-1} in the training dataset:

$$p(x_t \mid x_{t-1}) = \frac{\text{count}(x_{t-1}, x_t)}{\sum_{x'} \text{count}(x_{t-1}, x')}$$

- We can view this as training parameters $\theta_{i,j} = p(x_j | x_i)$

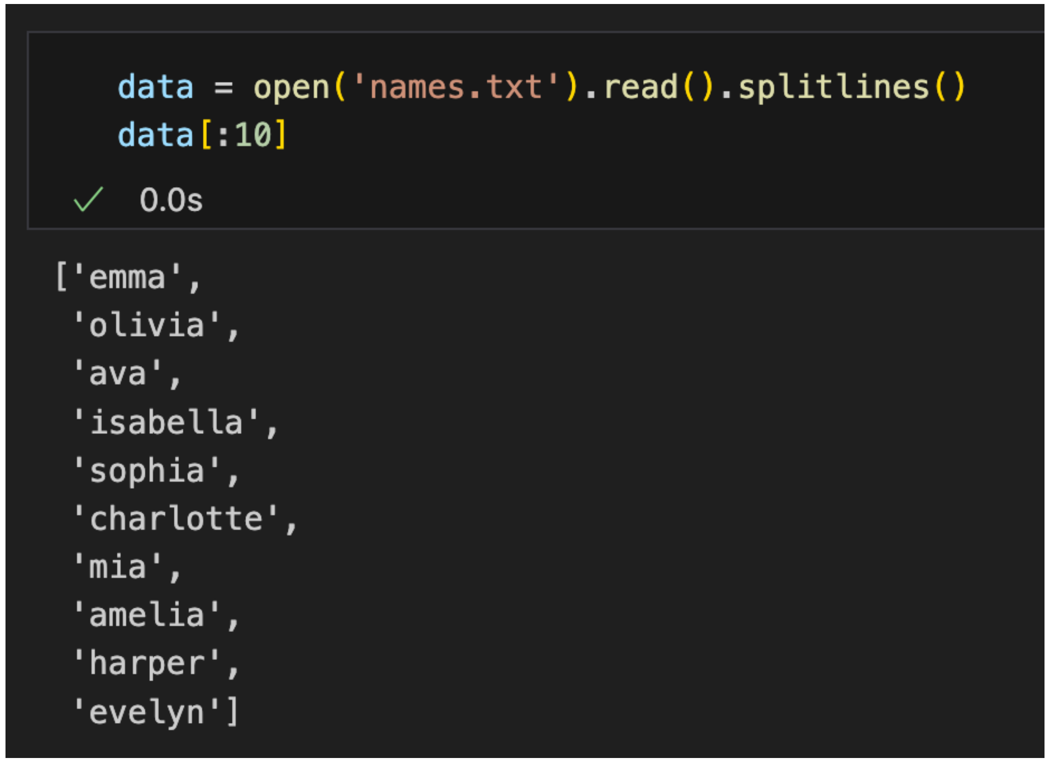
In Code

- Model a dataset of names. Character-level tokenization.

Based on Andrej Karpathy’s lecture/video: https://youtu.be/PaCmpygFfXo

In Code

- Model a dataset of names. Character-level tokenization.



```
data = open('names.txt').read().splitlines()
data[:10]
```

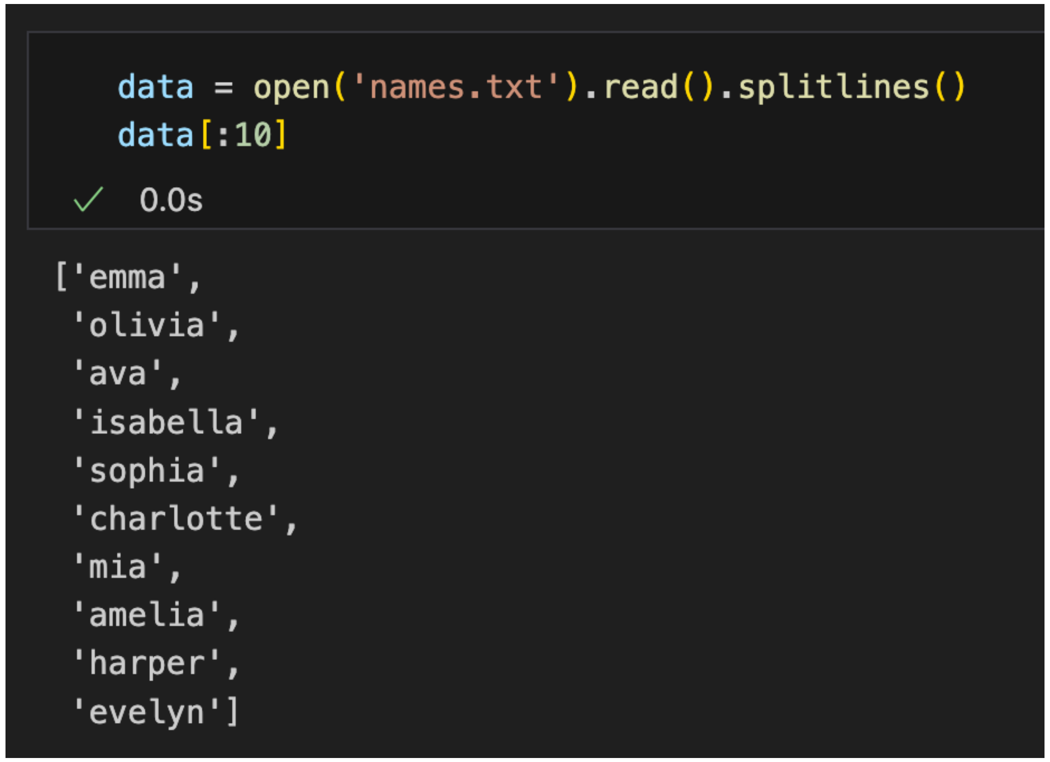
✓ 0.0s

```
['emma',
 'olivia',
 'ava',
 'isabella',
 'sophia',
 'charlotte',
 'mia',
 'amelia',
 'harper',
 'evelyn']
```

Based on Andrej Karpathy's lecture/video: https://youtu.be/PaCmpygFfXo

In Code

- Model a dataset of names. Character-level tokenization.



```
data = open('names.txt').read().splitlines()
data[:10]
```

✓ 0.0s

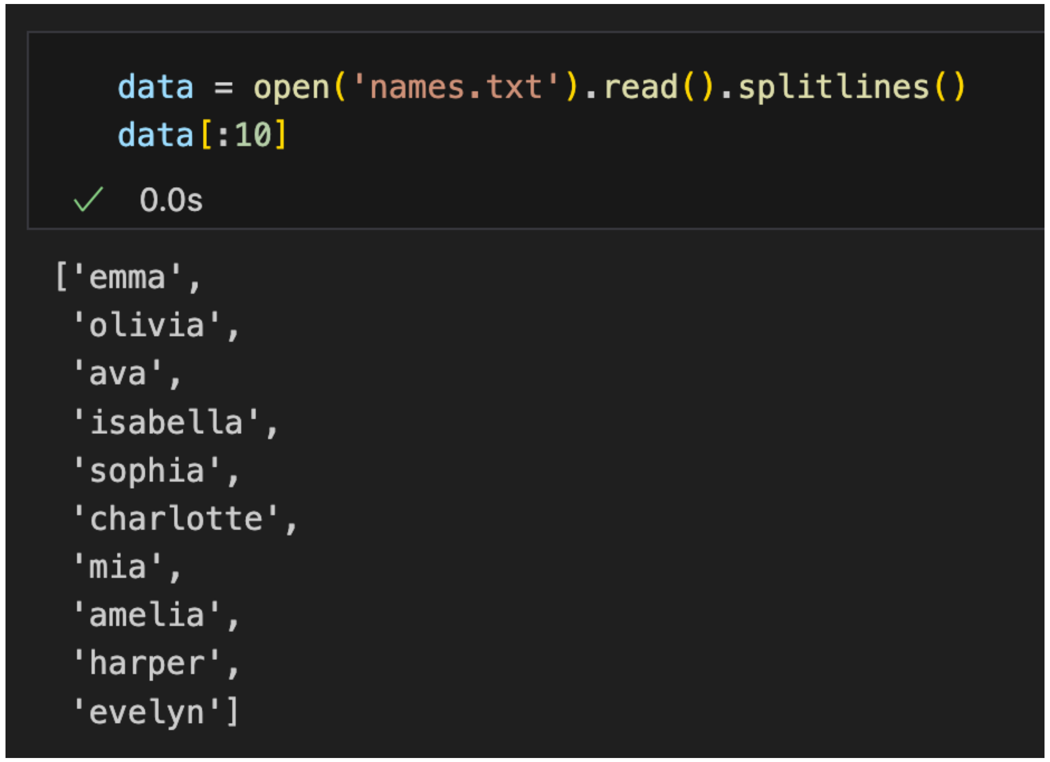
```
['emma',
 'olivia',
 'ava',
 'isabella',
 'sophia',
 'charlotte',
 'mia',
 'amelia',
 'harper',
 'evelyn']
```

```
bigram_counts = {}
for x in data:
    sequence = ['[S]'] + list(x) + ['[S]']
    for x1, x2 in zip(sequence, sequence[1:]):
        bigram = (x1, x2)
        bigram_counts[bigram] = bigram_counts.get(bigram, 0) + 1
```

Based on Andrej Karpathy's lecture/video: https://youtu.be/PaCmpygFfXo

In Code

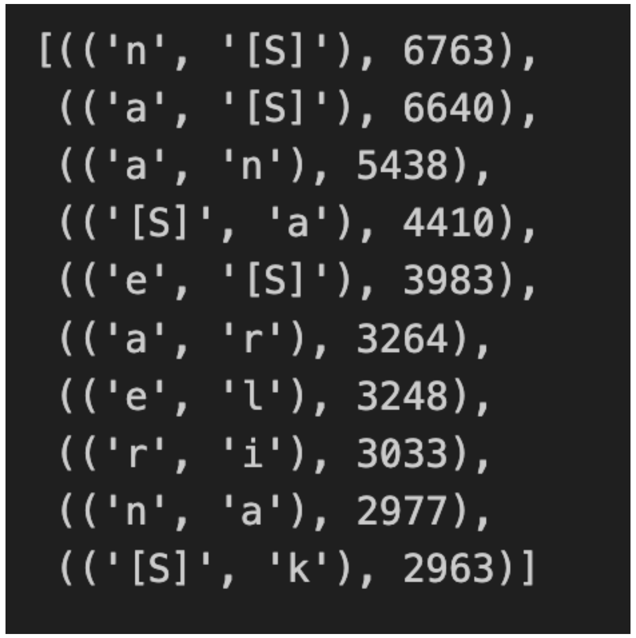
- Model a dataset of names. Character-level tokenization.



```
data = open('names.txt').read().splitlines()
data[:10]
```

```
['emma',
 'olivia',
 'ava',
 'isabella',
 'sophia',
 'charlotte',
 'mia',
 'amelia',
 'harper',
 'evelyn']
```

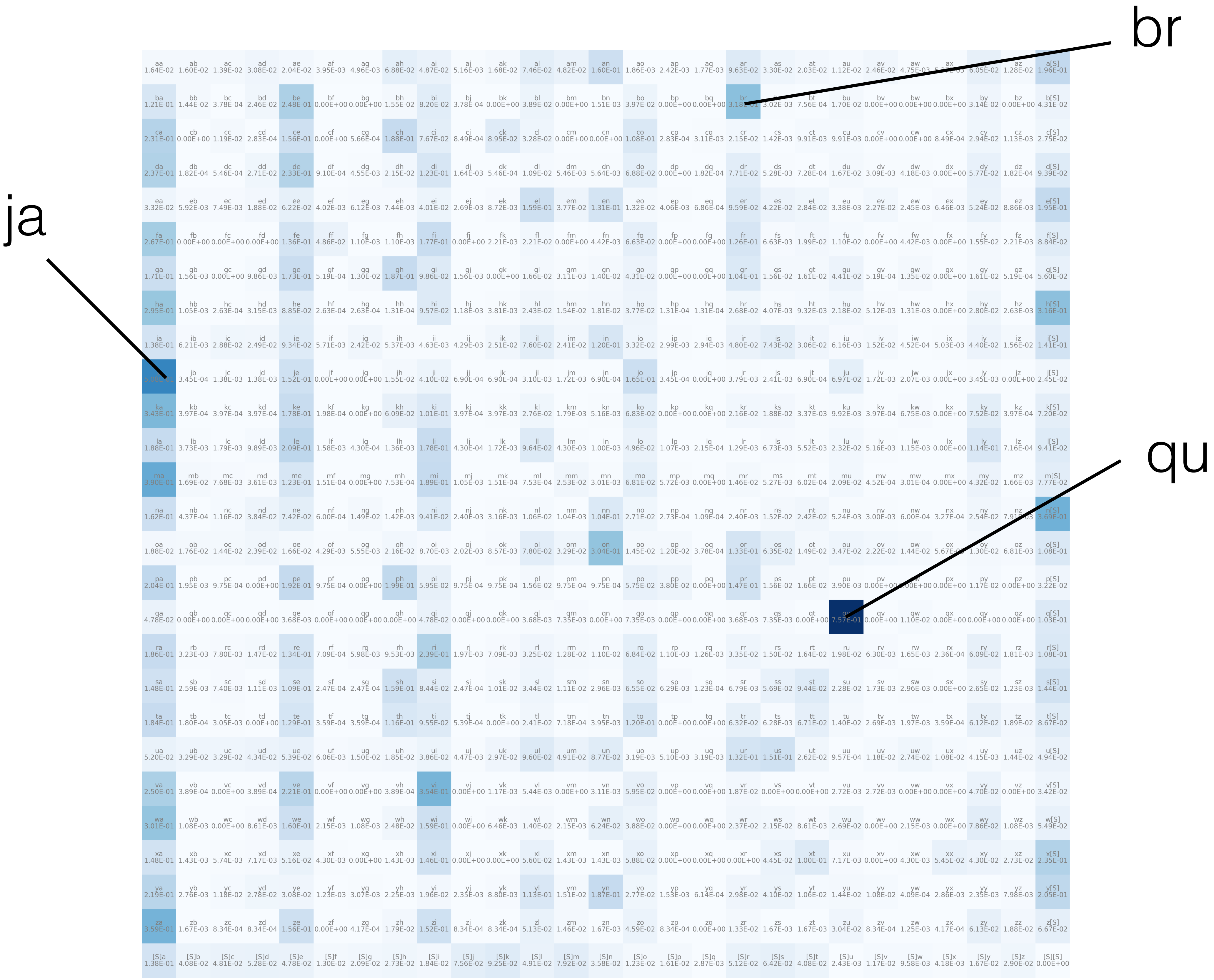
```
bigram_counts = {}
for x in data:
    sequence = ['[S]'] + list(x) + ['[S]']
    for x1, x2 in zip(sequence, sequence[1:]):
        bigram = (x1, x2)
        bigram_counts[bigram] = bigram_counts.get(bigram, 0) + 1
```



```
[(('n', '[S]'), 6763),
 (('a', '[S]'), 6640),
 (('a', 'n'), 5438),
 (('[S]', 'a'), 4410),
 (('e', '[S]'), 3983),
 (('a', 'r'), 3264),
 (('e', 'l'), 3248),
 (('r', 'i'), 3033),
 (('n', 'a'), 2977),
 (('[S]', 'k'), 2963)]
```

Based on Andrej Karpathy's lecture/video: https://youtu.be/PaCmpygFfXo

Model probabilities



aa 1.64E-02	ab 1.60E-02	ac 1.39E-02	ad 3.08E-02	ae 2.04E-02	af 3.95E-03	ag 4.96E-03	ah 6.88E-02	ai 4.87E-02	aj 5.16E-03	ak 1.68E-02	al 7.46E-02	am 4.82E-02	an 1.60E-01	ao 1.86E-03	ap 2.42E-03	aq 1.77E-03	ar 9.63E-02	as 3.30E-02	at 2.03E-02	au 1.12E-02	av 2.46E-02	aw 4.75E-03	ax 5.37E-03	ay 6.05E-02	az 1.28E-02	a[S] 1.96E-01
ba 1.21E-01	bb 1.44E-02	bc 3.78E-04	bd 2.46E-02	be 2.48E-01	bf 0.00E+00	bg 0.00E+00	bh 1.55E-02	bi 8.20E-02	bj 3.78E-04	bk 0.00E+00	bl 3.89E-02	bm 0.00E+00	bn 1.51E-03	bo 3.97E-02	bp 0.00E+00	bq 0.00E+00	br 3.18E-01	bs 3.02E-03	bt 7.56E-04	bu 1.70E-02	bv 0.00E+00	bw 0.00E+00	bx 0.00E+00	by 3.14E-02	bz 0.00E+00	b[S] 4.31E-02
ca 2.31E-01	cb 0.00E+00	cc 1.19E-02	cd 2.83E-04	ce 1.56E-01	cf 0.00E+00	cg 5.66E-04	ch 1.88E-01	ci 7.67E-02	cj 8.49E-04	ck 8.95E-02	cl 3.28E-02	cm 0.00E+00	cn 0.00E+00	co 1.08E-01	cp 2.83E-04	cq 3.11E-03	cr 2.15E-02	cs 1.42E-03	ct 9.91E-03	cu 9.91E-03	cv 0.00E+00	cw 0.00E+00	cx 8.49E-04	cy 2.94E-02	cz 1.13E-03	c[S] 2.75E-02
da 2.37E-01	db 1.82E-04	dc 5.46E-04	dd 2.71E-02	de 2.33E-01	df 9.10E-04	dg 4.55E-03	dh 2.15E-02	di 1.23E-01	dj 1.64E-03	dk 5.46E-04	dl 1.09E-02	dm 5.46E-03	dn 5.64E-03	do 6.88E-02	dp 0.00E+00	dq 1.82E-04	dr 7.71E-02	ds 5.28E-03	dt 7.28E-04	du 1.67E-02	dv 3.09E-03	dw 4.18E-03	dx 0.00E+00	dy 5.77E-02	dz 1.82E-04	d[S] 9.39E-02
ea 3.32E-02	eb 5.92E-03	ec 7.49E-03	ed 1.88E-02	ee 6.22E-02	ef 4.02E-03	eg 6.12E-03	eh 7.44E-03	ei 4.01E-02	ej 2.69E-03	ek 8.72E-03	el 1.59E-01	em 3.77E-02	en 1.31E-01	eo 1.32E-02	ep 4.06E-03	eq 6.86E-04	er 9.59E-02	es 4.22E-02	et 2.84E-02	eu 3.38E-03	ev 2.27E-02	ew 2.45E-03	ex 6.46E-03	ey 5.24E-02	ez 8.86E-03	e[S] 1.95E-01
fa 2.67E-01	fb 0.00E+00	fc 0.00E+00	fd 0.00E+00	fe 1.36E-01	ff 4.86E-02	fg 1.10E-03	fh 1.10E-03	fi 1.77E-01	fj 0.00E+00	fk 2.21E-03	fl 2.21E-02	fm 0.00E+00	fn 4.42E-03	fo 6.63E-02	fp 0.00E+00	fq 0.00E+00	fr 1.26E-01	fs 6.63E-03	ft 1.99E-02	fu 1.10E-02	fv 0.00E+00	fw 4.42E-03	fx 0.00E+00	fy 1.55E-02	fz 2.21E-03	f[S] 8.84E-02
ga 1.71E-01	gb 1.56E-03	gc 0.00E+00	gd 9.86E-03	ge 1.73E-01	gf 5.19E-04	gg 1.30E-02	gh 1.87E-01	gi 9.86E-02	gj 1.56E-03	gk 0.00E+00	gl 1.66E-02	gm 3.11E-03	gn 1.40E-02	go 4.31E-02	gp 0.00E+00	gq 0.00E+00	gr 1.04E-01	gs 1.56E-02	gt 1.61E-02	gu 4.41E-02	gv 5.19E-04	gw 1.35E-02	gx 0.00E+00	gy 1.61E-02	gz 5.19E-04	g[S] 5.60E-02
ha 2.95E-01	hb 1.05E-03	hc 2.63E-04	hd 3.15E-03	he 8.85E-02	hf 2.63E-04	hg 2.63E-04	hh 1.31E-04	hi 9.57E-02	hj 1.18E-03	hk 3.81E-03	hl 2.43E-02	hm 1.54E-02	hn 1.81E-02	ho 3.77E-02	hp 1.31E-04	hq 1.31E-04	hr 2.68E-02	hs 4.07E-03	ht 9.32E-03	hu 2.18E-02	hv 5.12E-03	hw 1.31E-03	hx 0.00E+00	hy 2.80E-02	hz 2.63E-03	h[S] 3.16E-01
ia 1.38E-01	ib 6.21E-03	ic 2.88E-02	id 2.49E-02	ie 9.34E-02	if 5.71E-03	ig 2.42E-02	ih 5.37E-03	ii 4.63E-03	ij 4.29E-03	ik 2.51E-02	il 7.60E-02	im 2.41E-02	in 1.20E-01	io 3.32E-02	ip 2.99E-03	iq 2.94E-03	ir 4.80E-02	is 7.43E-02	it 3.06E-02	iu 6.16E-03	iv 1.52E-02	iw 4.52E-04	ix 5.03E-03	iy 4.40E-02	iz 1.56E-02	i[S] 1.41E-01
ja 5.08E-01	jb 3.45E-04	jc 1.38E-03	jd 1.38E-03	je 1.52E-01	jf 0.00E+00	jg 0.00E+00	jh 1.55E-02	ji 4.10E-02	jj 6.90E-04	jk 6.90E-04	jl 3.10E-03	jm 1.72E-03	jn 6.90E-04	jo 1.65E-01	jp 3.45E-04	jq 0.00E+00	jr 3.79E-03	js 2.41E-03	jt 6.90E-04	ju 6.97E-02	jv 1.72E-03	jw 2.07E-03	jx 0.00E+00	jy 3.45E-03	jz 0.00E+00	j[S] 2.45E-02
ka 3.43E-01	kb 3.97E-04	kc 3.97E-04	kd 3.97E-04	ke 1.78E-01	kf 1.98E-04	kg 0.00E+00	kh 6.09E-02	ki 1.01E-01	kj 3.97E-04	kk 3.97E-03	kl 2.76E-02	km 1.79E-03	kn 5.16E-03	ko 6.83E-02	kp 0.00E+00	kq 0.00E+00	kr 2.16E-02	ks 1.88E-02	kt 3.37E-03	ku 9.92E-03	kv 3.97E-04	kw 6.75E-03	kx 0.00E+00	ky 7.52E-02	kz 3.97E-04	k[S] 7.20E-02
la 1.88E-01	lb 3.73E-03	lc 1.79E-03	ld 9.89E-03	le 2.09E-01	lf 1.58E-03	lg 4.30E-04	lh 1.36E-03	li 1.78E-01	lj 4.30E-04	lk 1.72E-03	ll 9.64E-02	lm 4.30E-03	ln 1.00E-03	lo 4.96E-02	lp 1.07E-03	lq 2.15E-04	lr 1.29E-03	ls 6.73E-03	lt 5.52E-03	lu 2.32E-02	lv 5.16E-03	lw 1.15E-03	lx 0.00E+00	ly 1.14E-01	lz 7.16E-04	l[S] 9.41E-02
ma 3.90E-01	mb 1.69E-02	mc 7.68E-03	md 3.61E-03	me 1.23E-01	mf 1.51E-04	mg 0.00E+00	mh 7.53E-04	mi 1.89E-01	mj 1.05E-03	mk 1.51E-04	ml 7.53E-04	mm 2.53E-02	mn 3.01E-03	mo 6.81E-02	mp 5.72E-03	mq 0.00E+00	mr 1.46E-02	ms 5.27E-03	mt 6.02E-04	mu 2.09E-02	mv 4.52E-04	mw 3.01E-04	mx 0.00E+00	my 4.32E-02	mz 1.66E-03	m[S] 7.77E-02
na 1.62E-01	nb 4.37E-04	nc 1.16E-02	nd 3.84E-02	ne 7.42E-02	nf 6.00E-04	ng 1.49E-02	nh 1.42E-03	ni 9.41E-02	nj 2.40E-03	nk 3.16E-03	nl 1.06E-02	nm 1.04E-03	nn 1.04E-01	no 2.71E-02	np 2.73E-04	nq 1.09E-04	nr 2.40E-03	ns 1.52E-02	nt 2.42E-02	nu 5.24E-03	nv 3.00E-03	nw 6.00E-04	nx 3.27E-04	ny 2.54E-02	nz 7.91E-03	n[S] 3.69E-01
oa 1.88E-02	ob 1.76E-02	oc 1.44E-02	od 2.39E-02	oe 1.66E-02	of 4.29E-03	og 5.55E-03	oh 2.16E-02	oi 8.70E-03	oj 2.02E-03	ok 8.57E-03	ol 7.80E-02	om 3.29E-02	on 3.04E-01	oo 1.45E-02	op 1.20E-02	oq 3.78E-04	or 1.33E-01	os 6.35E-02	ot 1.49E-02	ou 3.47E-02	ov 2.22E-02	ow 1.44E-02	ox 5.67E-03	oy 1.30E-02	oz 6.81E-03	o[S] 1.08E-01
pa 2.04E-01	pb 1.95E-03	pc 9.75E-04	pd 0.00E+00	pe 1.92E-01	pf 9.75E-04	pg 0.00E+00	ph 1.99E-01	pi 5.95E-02	pj 9.75E-04	pk 9.75E-04	pl 1.56E-02	pm 9.75E-04	pn 9.75E-04	po 5.75E-02	pp 3.80E-02	pq 0.00E+00	pr 1.47E-01	ps 1.56E-02	pt 1.66E-02	pu 3.90E-03	pv 0.00E+00	pw 0.00E+00	px 0.00E+00	py 1.17E-02	pz 0.00E+00	p[S] 3.22E-02
qa 4.78E-02	qb 0.00E+00	qc 0.00E+00	qd 0.00E+00	qe 3.68E-03	qf 0.00E+00	qg 0.00E+00	qh 0.00E+00	qi 4.78E-02	qj 0.00E+00	qk 0.00E+00	ql 3.68E-03	qm 7.35E-03	qn 0.00E+00	qo 7.35E-03	qp 0.00E+00	qq 0.00E+00	qr 3.68E-03	qs 7.35E-03	qt 0.00E+00	qu 7.57E-01	qv 0.00E+00	qw 1.10E-02	qx 0.00E+00	qy 0.00E+00	qz 0.00E+00	q[S] 1.03E-01
ra 1.86E-01	rb 3.23E-03	rc 7.80E-03	rd 1.47E-02	re 1.34E-01	rf 7.09E-04	rg 5.98E-03	rh 9.53E-03	ri 2.39E-01	rj 1.97E-03	rk 7.09E-03	rl 3.25E-02	rm 1.28E-02	rn 1.10E-02	ro 6.84E-02	rp 1.10E-03	rq 1.26E-03	rr 3.35E-02	rs 1.50E-02	rt 1.64E-02	ru 1.98E-02	rv 6.30E-03	rw 1.65E-03	rx 2.36E-04	ry 6.09E-02	rz 1.81E-03	r[S] 1.08E-01
sa 1.48E-01	sb 2.59E-03	sc 7.40E-03	sd 1.11E-03	se 1.09E-01	sf 2.47E-04	sg 2.47E-04	sh 1.59E-01	si 8.44E-02	sj 2.47E-04	sk 1.01E-02	sl 3.44E-02	sm 1.11E-02	sn 2.96E-03	so 6.55E-02	sp 6.29E-03	sq 1.23E-04	sr 6.79E-03	ss 5.69E-02	st 9.44E-02	su 2.28E-02	sv 1.73E-03	sw 2.96E-03	sx 0.00E+00	sy 2.65E-02	sz 1.23E-03	s[S] 1.44E-01
ta 1.84E-01	tb 1.80E-04	tc 3.05E-03	td 0.00E+00	te 1.29E-01	tf 3.59E-04	tg 3.59E-04	th 1.16E-01	ti 9.55E-02	tj 5.39E-04	tk 0.00E+00	tl 2.41E-02	tm 7.18E-04	tn 3.95E-03	to 1.20E-01	tp 0.00E+00	tq 0.00E+00	tr 6.32E-02	ts 6.28E-03	tt 6.71E-02	tu 1.40E-02	tv 2.69E-03	tw 1.97E-03	tx 3.59E-04	ty 6.12E-02	tz 1.89E-02	t[S] 8.67E-02
ua 5.20E-02	ub 3.29E-02	uc 3.29E-02	ud 4.34E-02	ue 5.39E-02	uf 6.06E-03	ug 1.50E-02	uh 1.85E-02	ui 3.86E-02	uj 4.47E-03	uk 2.97E-02	ul 9.60E-02	um 4.91E-02	un 8.77E-02	uo 3.19E-03	up 5.10E-03	uq 3.19E-03	ur 1.32E-01	us 1.51E-01	ut 2.62E-02	uu 9.57E-04	uv 1.18E-02	uw 2.74E-02	ux 1.08E-02	uy 4.15E-03	uz 1.44E-02	u[S] 4.94E-02
va 2.50E-01	vb 3.89E-04	vc 0.00E+00	vd 3.89E-04	ve 2.21E-01	vf 0.00E+00	vg 0.00E+00	vh 3.89E-04	vi 3.54E-01	vj 0.00E+00	vk 1.17E-03	vl 5.44E-03	vm 0.00E+00	vn 3.11E-03	vo 5.95E-02	vp 0.00E+00	vq 0.00E+00	vr 1.87E-02	vs 0.00E+00	vt 0.00E+00	vu 2.72E-03	vv 2.72E-03	vw 0.00E+00	vx 0.00E+00	vy 4.70E-02	vz 0.00E+00	v[S] 3.42E-02
wa 3.01E-01	wb 1.08E-03	wc 0.00E+00	wd 8.61E-03	we 1.60E-01	wf 2.15E-03	wg 1.08E-03	wh 2.48E-02	wi 1.59E-01	wj 0.00E+00	wk 6.46E-03	wl 1.40E-02	wm 2.15E-03	wn 6.24E-02	wo 3.88E-02	wp 0.00E+00	wq 0.00E+00	wr 2.37E-02	ws 2.15E-02	wt 8.61E-03	wu 2.69E-02	wv 0.00E+00	ww 2.15E-03	wx 0.00E+00	wy 7.86E-02	wz 1.08E-03	w[S] 5.49E-02
xa 1.48E-01	xb 1.43E-03	xc 5.74E-03	xd 7.17E-03	xe 5.16E-02	xf 4.30E-03	xg 0.00E+00	xh 1.43E-03	xi 1.46E-01	xj 0.00E+00	xk 0.00E+00	xl 5.60E-02	xm 1.43E-03	xn 1.43E-03	xo 5.88E-02	xp 0.00E+00	xq 0.00E+00	xr 0.00E+00	xs 4.45E-02	xt 1.00E-01	xu 7.17E-03	xv 0.00E+00	xw 4.30E-03	xx 5.45E-02	xy 4.30E-02	xz 2.73E-02	x[S] 2.35E-01
ya 2.19E-01	yb 2.76E-03	yc 1.18E-02	yd 2.78E-02	ye 3.08E-02	yf 1.23E-03	yg 3.07E-03	yh 2.25E-03	yi 1.96E-02	yj 2.35E-03	yk 8.80E-03	yl 1.13E-01	ym 1.51E-02	yn 1.87E-01	yo 2.77E-02	yp 1.53E-03	yq 6.14E-04	yr 2.98E-02	ys 4.10E-02	yt 1.06E-02	yu 1.44E-02	yv 1.08E-02	yw 4.09E-04	yx 2.86E-03	yy 2.35E-03	yz 7.98E-03	y[S] 2.05E-01
za 3.59E-01	zb 1.67E-03	zc 8.34E-04	zd 8.34E-04	ze 1.56E-01	zf 0.00E+00	zg 4.17E-04	zh 1.79E-02	zi 1.52E-01	zj 8.34E-04	zk 8.34E-04	zl 5.13E-02	zm 1.46E-02	zn 1.67E-03	zo 4.59E-02	zp 8.34E-04	zq 0.00E+00	zr 1.33E-02	zs 1.67E-03	zt 1.67E-03	zu 3.04E-02	zv 8.34E-04	zw 1.25E-03	zx 4.17E-04	zy 6.13E-02	zz 1.88E-02	z[S] 6.67E-02
[S]a 1.38E-01	[S]b 4.08E-02	[S]c 4.81E-02	[S]d 5.28E-02	[S]e 4.78E-02	[S]f 1.30E-02	[S]g 2.09E-02	[S]h 2.73E-02	[S]i 1.84E-02	[S]j 7.56E-02	[S]k 9.25E-02	[S]l 4.91E-02	[S]m 7.92E-02	[S]n 3.58E-02	[S]o 1.23E-02	[S]p 1.61E-02	[S]q 2.87E-03	[S]r 5.12E-02	[S]s 6.42E-02	[S]t 4.08E-02	[S]u 2.43E-03	[S]v 1.17E-02	[S]w 9.58E-03	[S]x 4.18E-03	[S]y 1.67E-02	[S]z 2.90E-02	[S][S] 0.00E+00

Based on Andrej Karpathy’s lecture/video: https://youtu.be/PaCmpygFfXo

Training : why counting?

Training : why counting?

- The counting procedure corresponds to **maximum likelihood** estimation for this model:

$$\max_{\theta} \sum_{x \in D_{train}} \log p_{\theta}(x)$$

Training : why counting?

- The counting procedure corresponds to **maximum likelihood** estimation for this model:

$$\max_{\theta} \sum_{x \in D_{train}} \log p_{\theta}(x)$$

- Intuition: train the model so that it assigns high probability to the training data D_{train}

Training : why counting?

- The counting procedure corresponds to **maximum likelihood** estimation for this model:

$$\max_{\theta} \sum_{x \in D_{train}} \log p_{\theta}(x)$$

- Intuition: train the model so that it assigns high probability to the training data D_{train}

Exercise: derive the update on the previous slide

Training: Why maximum likelihood?

Training: Why maximum likelihood?

- Makes p_θ match the data distribution p_{data} (p_* for brevity)

Training: Why maximum likelihood?

- Makes p_θ match the data distribution p_{data} (p_* for brevity)

$$\min_\theta D_{KL}(p_* || p_\theta) =$$

Training: Why maximum likelihood?

- Makes p_θ match the data distribution p_{data} (p_* for brevity)

$$\min_\theta D_{KL}(p_* || p_\theta) = \min_\theta - \sum_{x \in \mathcal{X}} p_*(x) \log \frac{p_\theta(x)}{p_*(x)}$$

Training: Why maximum likelihood?

- Makes p_θ match the data distribution p_{data} (p_* for brevity)

$$\min_\theta D_{KL}(p_* \,||\, p_\theta) = \min_\theta - \sum_{x \in \mathcal{X}} p_*(x) \log \frac{p_\theta(x)}{p_*(x)}$$

$$\equiv \min_\theta - \sum_{x \in \mathcal{X}} p_*(x) \log p_\theta(x) + \text{const}$$

Training: Why maximum likelihood?

- Makes p_θ match the data distribution p_{data} (p_* for brevity)

$$\begin{aligned}\min_\theta D_{KL}(p_* \,||\, p_\theta) &= \min_\theta - \sum_{x \in \mathcal{X}} p_*(x) \log \frac{p_\theta(x)}{p_*(x)} \\ &\equiv \min_\theta - \sum_{x \in \mathcal{X}} p_*(x) \log p_\theta(x) + \text{const} \\ &= \min_\theta - \mathbb{E}_{x \sim p_*} \log p_\theta(x)\end{aligned}$$

Training: Why maximum likelihood?

- Makes p_θ match the data distribution p_{data} (p_* for brevity)

$$\begin{aligned}\min_\theta D_{KL}(p_* \,||\, p_\theta) &= \min_\theta - \sum_{x \in \mathcal{X}} p_*(x) \log \frac{p_\theta(x)}{p_*(x)} \\ &\equiv \min_\theta - \sum_{x \in \mathcal{X}} p_*(x) \log p_\theta(x) + \text{const} \\ &= \min_\theta - \mathbb{E}_{x \sim p_*} \log p_\theta(x) \\ &\approx \min_\theta - \frac{1}{|D|} \sum_{x \in D} \log p_\theta(x)\end{aligned}$$

Training: Why maximum likelihood?

- Makes p_θ match the data distribution p_{data} (p_* for brevity)

$$\begin{aligned}\min_\theta D_{KL}(p_* || p_\theta) &= \min_\theta - \sum_{x \in \mathcal{X}} p_*(x) \log \frac{p_\theta(x)}{p_*(x)} \\ &\equiv \min_\theta - \sum_{x \in \mathcal{X}} p_*(x) \log p_\theta(x) + \text{const} \\ &= \min_\theta - \mathbb{E}_{x \sim p_*} \log p_\theta(x) \\ &\approx \min_\theta - \frac{1}{|D|} \sum_{x \in D} \log p_\theta(x)\end{aligned}$$

Dataset:
samples from p_*

Training: Why maximum likelihood?

- Makes p_θ match the data distribution p_{data} (p_* for brevity)

$$\begin{aligned}
\min_\theta D_{KL}(p_* \,||\, p_\theta) &= \min_\theta - \sum_{x \in \mathcal{X}} p_*(x) \log \frac{p_\theta(x)}{p_*(x)} \\
&\equiv \min_\theta - \sum_{x \in \mathcal{X}} p_*(x) \log p_\theta(x) + \text{const} \\
&= \min_\theta - \mathbb{E}_{x \sim p_*} \log p_\theta(x) \\
&\approx \min_\theta - \frac{1}{|D|} \sum_{x \in D} \log p_\theta(x) \\
&\equiv \max_\theta \sum_{x \in D} \log p_\theta(x)
\end{aligned}$$

Dataset: samples from p_*

Maximum likelihood!

Note: using log space

Note: using log space

- Multiplication of probabilities can be re-expressed as addition of log probabilities

$$P(X) = \prod_{i=1}^{|X|} P(x_i) \quad \longrightarrow \quad \log P(X) = \sum_{i=1}^{|X|} \log P(x_i)$$

Note: using log space

- Multiplication of probabilities can be re-expressed as addition of log probabilities

$$P(X) = \prod_{i=1}^{|X|} P(x_i) \quad \longrightarrow \quad \log P(X) = \sum_{i=1}^{|X|} \log P(x_i)$$

- **Why?:** numerical stability, other conveniences

Generation

Generation

- Generate from an autoregressive model by iteratively sampling a next token, then appending it to the context

Until [S] is generated:

$$\hat{x}_t \sim p_\theta(x_t \,|\, \hat{x}_{t-1})$$

Generation

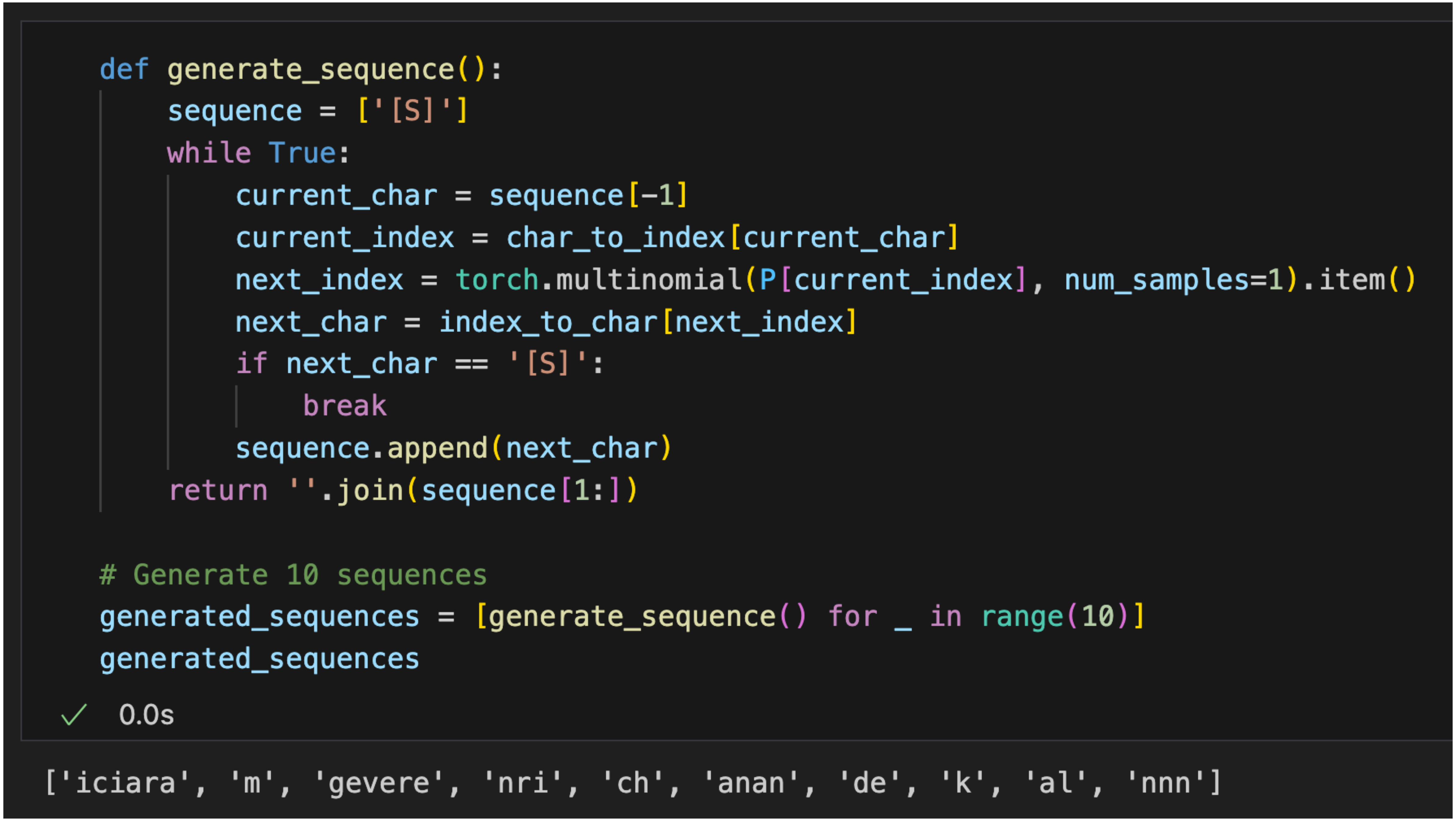
- Generate from an autoregressive model by iteratively sampling a next token, then appending it to the context

Until [S] is generated:

$$\hat{x}_t \sim p_\theta(x_t \,|\, \hat{x}_{t-1})$$

- Equivalent to sampling from the model's joint distribution over full sequences! (*More in lecture 7*)

In Code



```python
def generate_sequence():
    sequence = ['[S]']
    while True:
        current_char = sequence[-1]
        current_index = char_to_index[current_char]
        next_index = torch.multinomial(P[current_index], num_samples=1).item()
        next_char = index_to_char[next_index]
        if next_char == '[S]':
            break
        sequence.append(next_char)
    return ''.join(sequence[1:])

# Generate 10 sequences
generated_sequences = [generate_sequence() for _ in range(10)]
generated_sequences
```

```
['iciara', 'm', 'gevere', 'nri', 'ch', 'anan', 'de', 'k', 'al', 'nnn']
```

Evaluation

Evaluation

- We can evaluate a model based on the probabilities it assigns to a dataset
    - E.g., the training set or a held-out test set

Evaluation

- We can evaluate a model based on the probabilities it assigns to a dataset
  - E.g., the training set or a held-out test set
- Two widely used metrics in language modeling:
  - Log-likelihood
  - Perplexity

Log-likelihood

Log-likelihood

- **Log-likelihood:**

$$LL(\mathcal{X}_{\text{test}}) = \sum_{X \in \mathcal{X}_{\text{test}}} \log P(X))$$

Log-likelihood

- **Log-likelihood:**

$$LL(\mathcal{X}_{\text{test}}) = \sum_{X \in \mathcal{X}_{\text{test}}} \log P(X))$$

- **Per-word Log Likelihood:**

$$WLL(\mathcal{X}_{\text{test}}) = \frac{1}{\sum_{X \in \mathcal{X}_{\text{test}}} |X|} \sum_{X \in \mathcal{X}_{\text{test}}} \log P(X))$$

Log-likelihood

- **Log-likelihood:**

$$LL(\mathcal{X}_{\text{test}}) = \sum_{X \in \mathcal{X}_{\text{test}}} \log P(X))$$

- **Per-word Log Likelihood:**

$$WLL(\mathcal{X}_{\text{test}}) = \frac{1}{\sum_{X \in \mathcal{X}_{\text{test}}} |X|} \sum_{X \in \mathcal{X}_{\text{test}}} \log P(X))$$

Papers often also report negative log likelihood (lower better), as that is used in loss.

Perplexity

- **Perplexity:**

$$PPL(\mathcal{X}_{\text{test}}) = 2^{H(\mathcal{X}_{\text{test}})} = e^{-WLL(\mathcal{X}_{\text{test}})}$$

Perplexity

- **Perplexity:**

$$PPL(\mathcal{X}_{\text{test}}) = 2^{H(\mathcal{X}_{\text{test}})} = e^{-WLL(\mathcal{X}_{\text{test}})}$$

When a dog sees a squirrel it will usually ___

Perplexity

- **Perplexity:**

$$PPL(\mathcal{X}_{\text{test}}) = 2^{H(\mathcal{X}_{\text{test}})} = e^{-WLL(\mathcal{X}_{\text{test}})}$$

When a dog sees a squirrel it will usually ___

```
Token: ' be' - Probability: 0.0352
Token: ' jump' - Probability: 0.0338
Token: ' start' - Probability: 0.0289
Token: ' run' - Probability: 0.0277
Token: ' try' - Probability: 0.0219
```

Perplexity

- **Perplexity:**

$$PPL(\mathcal{X}_{\text{test}}) = 2^{H(\mathcal{X}_{\text{test}})} = e^{-WLL(\mathcal{X}_{\text{test}})}$$

When a dog sees a squirrel it will usually ___

Token: ' be' - Probability: 0.0352 → PPL= 28.4
Token: ' jump' - Probability: 0.0338
Token: ' start' - Probability: 0.0289
Token: ' run' - Probability: 0.0277
Token: ' try' - Probability: 0.0219

Perplexity

- **Perplexity:**

$$PPL(\mathcal{X}_{\text{test}}) = 2^{H(\mathcal{X}_{\text{test}})} = e^{-WLL(\mathcal{X}_{\text{test}})}$$

When a dog sees a squirrel it will usually ___

Token: ' be' - Probability: 0.0352 → PPL= 28.4
Token: ' jump' - Probability: 0.0338 → PPL= 29.6
Token: ' start' - Probability: 0.0289
Token: ' run' - Probability: 0.0277
Token: ' try' - Probability: 0.0219

Perplexity

- **Perplexity:**

$$PPL(\mathcal{X}_{\text{test}}) = 2^{H(\mathcal{X}_{\text{test}})} = e^{-WLL(\mathcal{X}_{\text{test}})}$$

When a dog sees a squirrel it will usually ___

Token: ' be' - Probability: 0.0352 → PPL= 28.4
Token: ' jump' - Probability: 0.0338 → PPL= 29.6
Token: ' start' - Probability: 0.0289 → PPL= 34.6
Token: ' run' - Probability: 0.0277
Token: ' try' - Probability: 0.0219

Perplexity

- **Perplexity:**

$$PPL(\mathcal{X}_{\text{test}}) = 2^{H(\mathcal{X}_{\text{test}})} = e^{-WLL(\mathcal{X}_{\text{test}})}$$

When a dog sees a squirrel it will usually ___

Token: ' be' - Probability: 0.0352 → PPL= 28.4
Token: ' jump' - Probability: 0.0338 → PPL= 29.6
Token: ' start' - Probability: 0.0289 → PPL= 34.6
Token: ' run' - Probability: 0.0277 → PPL= 36.1
Token: ' try' - Probability: 0.0219

Perplexity

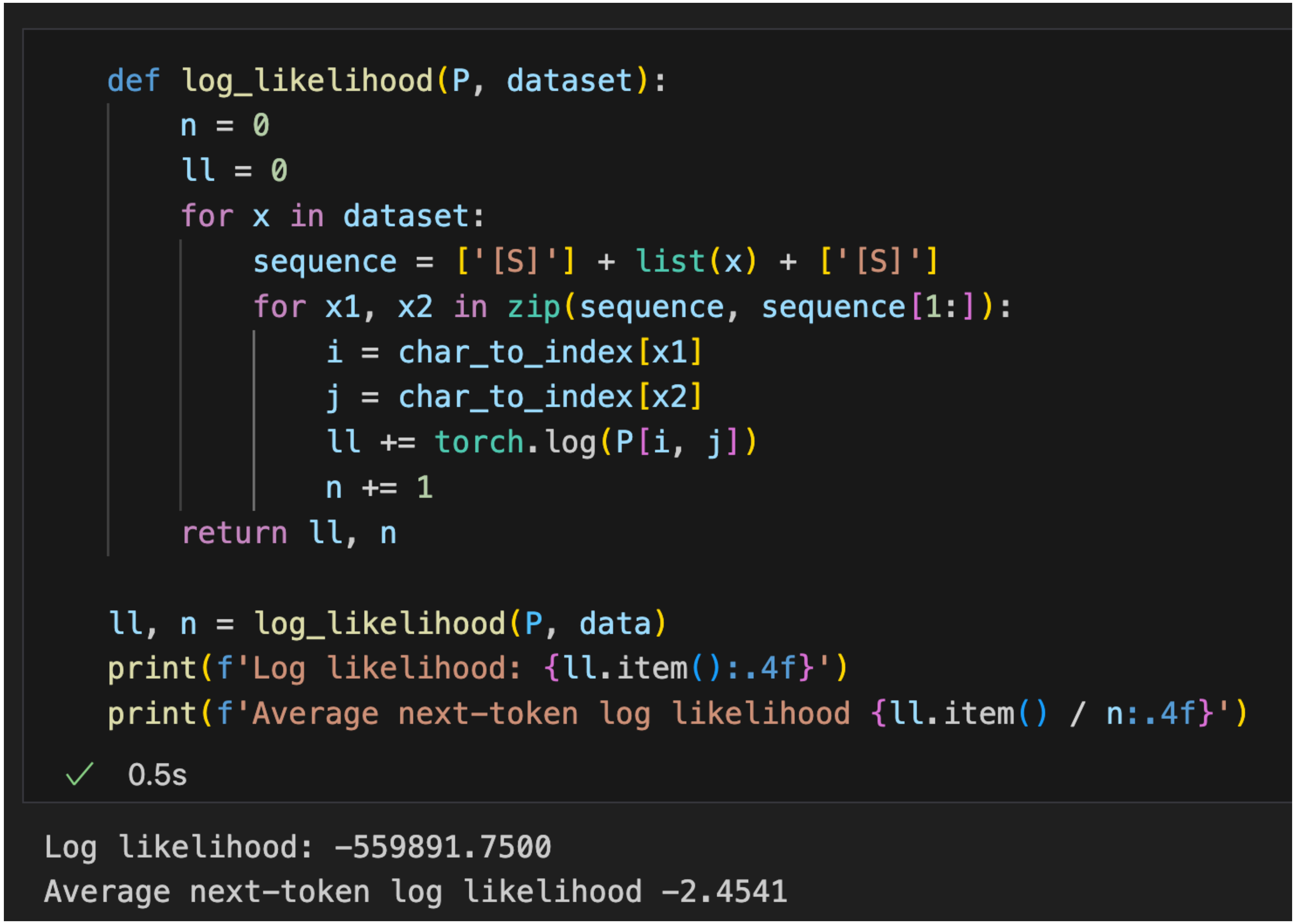
- **Perplexity:**

$$PPL(\mathcal{X}_{\text{test}}) = 2^{H(\mathcal{X}_{\text{test}})} = e^{-WLL(\mathcal{X}_{\text{test}})}$$

When a dog sees a squirrel it will usually ___

Token: ' be' - Probability: 0.0352 → PPL= 28.4
Token: ' jump' - Probability: 0.0338 → PPL= 29.6
Token: ' start' - Probability: 0.0289 → PPL= 34.6
Token: ' run' - Probability: 0.0277 → PPL= 36.1
Token: ' try' - Probability: 0.0219 → PPL= 45.7

In Code

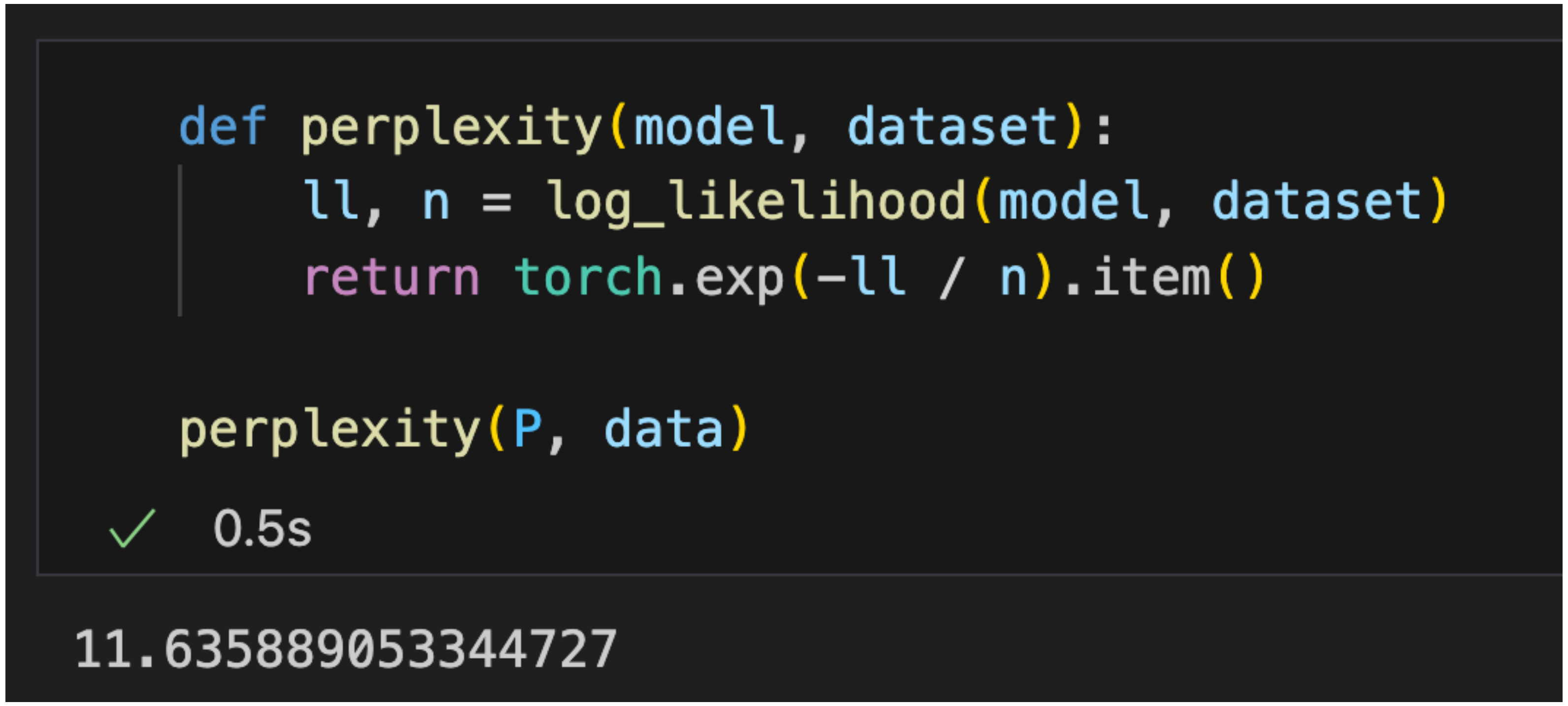


```python
def log_likelihood(P, dataset):
    n = 0
    ll = 0
    for x in dataset:
        sequence = ['[S]'] + list(x) + ['[S]']
        for x1, x2 in zip(sequence, sequence[1:]):
            i = char_to_index[x1]
            j = char_to_index[x2]
            ll += torch.log(P[i, j])
            n += 1
    return ll, n

ll, n = log_likelihood(P, data)
print(f'Log likelihood: {ll.item():.4f}')
print(f'Average next-token log likelihood {ll.item() / n:.4f}')
```

```
Log likelihood: -559891.7500
Average next-token log likelihood -2.4541
```

Disclaimer: don’t implement it like this in production

In Code



```python
def perplexity(model, dataset):
    ll, n = log_likelihood(model, dataset)
    return torch.exp(-ll / n).item()

perplexity(P, data)
```

```
11.635889053344727
```

Recap: Bigram models

Recap: Bigram models

- A simple language model, but we saw several key concepts:
  - Maximum likelihood estimation
  - Log space
  - Autoregressive generation
  - Evaluating log-likelihood and perplexity
  - Limited context size

Recap: Bigram models

- A simple language model, but we saw several key concepts:
  - Maximum likelihood estimation
  - Log space
  - Autoregressive generation
  - Evaluating log-likelihood and perplexity
  - Limited context size
- **Next**: Ngram models

Ngram models

$$P(X) \approx \prod_{t=1}^{T} p_\theta \left(\underbrace{x_t}_{\text{Next Token}} \mid \underbrace{x_{t-1}, x_{t-2}, \ldots, x_{t-n+1}}_{\text{n-token context}} \right)$$

- Use an analogous counting procedure to train

Training Ngram Models

Training Ngram Models

- Use an analogous counting procedure to train

$$p(x_t \mid x_{t-n+1:t-1}) = \frac{\text{count}(x_{t-n+1:t-1}, x_t)}{\sum_{x'} \text{count}(x_{t-n+1:t-1}, x')}$$

Training Ngram Models

- Add a ‘fake count’ to each possible ngram to avoid zero probability ngrams

Training Ngram Models

- Add a ‘fake count’ to each possible ngram to avoid zero probability ngrams

$$p(x_t \mid x_{t-n+1:t-1}) = \frac{1 + \text{count}(x_{t-n+1:t-1}, x_t)}{|V| \sum_{x'} \text{count}(x_{t-n+1:t-1}, x')}$$

Training Ngram Models

- Add a ‘fake count’ to each possible ngram to avoid zero probability ngrams

$$p(x_t \mid x_{t-n+1:t-1}) = \frac{1 + \text{count}(x_{t-n+1:t-1}, x_t)}{|V| \sum_{x'} \text{count}(x_{t-n+1:t-1}, x')}$$

- An example of *smoothing*

Problems

Problems

- Cannot share strength among **similar words**

she bought a car	she bought a bicycle
she purchased a car	she purchased a bicycle

→ solution: neural networks

Problems

- Cannot share strength among **similar words**

she bought a car she bought a bicycle
she purchased a car she purchased a bicycle

→ solution: neural networks

- Cannot condition on context with **intervening words**

Dr. Jane Smith Dr. Gertrude Smith

→ solution: neural networks

Problems

- Cannot share strength among **similar words**

she bought a car / she bought a bicycle
she purchased a car / she purchased a bicycle

→ solution: neural networks

- Cannot condition on context with **intervening words**

Dr. Jane Smith / Dr. Gertrude Smith

→ solution: neural networks

- Cannot handle **long-distance dependencies**

for tennis class he wanted to buy his own racquet

for programming class he wanted to buy his own computer

→ solution: neural networks in future lectures

When to use n-gram models?

- Neural language models achieve better performance, but
- n-gram models are extremely fast to estimate/apply
- Perfect memorization can be useful
- **Toolkit:** kenlm

https://github.com/kpu/kenlm

Feedforward neural language model

$$P(X) \approx \prod_{t=1}^{T} p_{\theta}\left(\underbrace{x_t}_{\text{Next Token}} \mid \underbrace{x_{t-1}, x_{t-2}, \ldots, x_{t-n+1}}_{\text{n-token context}}\right)$$

Neural network parameters :)

Bengio et al 2003, *A Neural Probabilistic Language Model*

Code: https://github.com/cmu-l3/anlp-spring2025-code/blob/main/03_lm_fundamentals/lm_basics_neural.ipynb

Neural language model

Bengio et al 2003, *A Neural Probabilistic Language Model*

Neural language model

- Ngram language models do not take into account the similarity of words or contexts

Bengio et al 2003, *A Neural Probabilistic Language Model*

Neural language model

- Ngram language models do not take into account the similarity of words or contexts
    - *The cat was walking in the bedroom*

Bengio et al 2003, *A Neural Probabilistic Language Model*

Neural language model

- Ngram language models do not take into account the similarity of words or contexts
    - *The cat was walking in the bedroom*
    - *The dog was running in a room*

Bengio et al 2003, *A Neural Probabilistic Language Model*

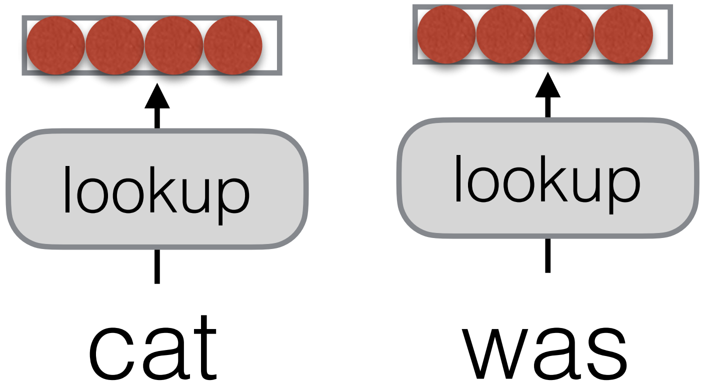
Neural language model

- Ngram language models do not take into account the similarity of words or contexts
  - *The cat was walking in the bedroom*
  - *The dog was running in a room*
- *Solution*: use learned, distributed representations

Bengio et al 2003, *A Neural Probabilistic Language Model*

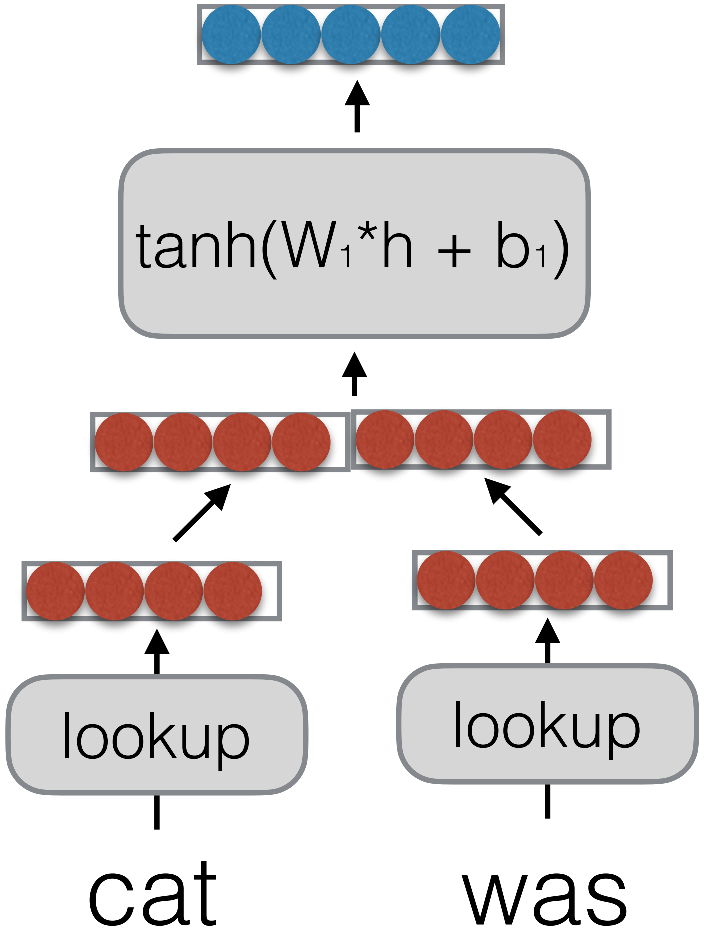
Feedforward neural language model

Feedforward neural language model



Embedding

Feedforward neural language model



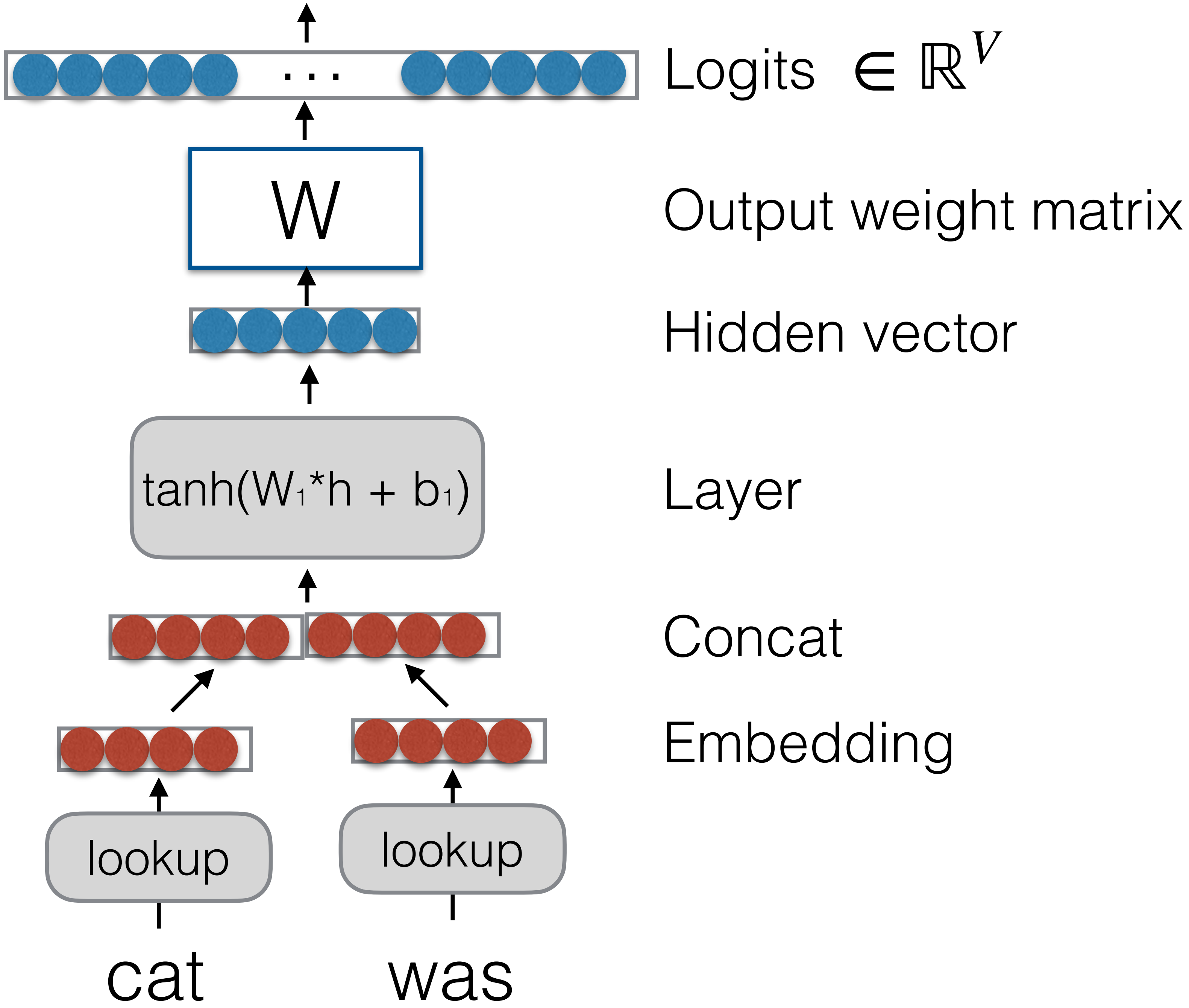
Hidden vector

Layer

Concat

Embedding

Feedforward neural language model



Logits $\in \mathbb{R}^V$

W

Output weight matrix

Hidden vector

$\tanh(W_1 * h + b_1)$

Layer

Concat

Embedding

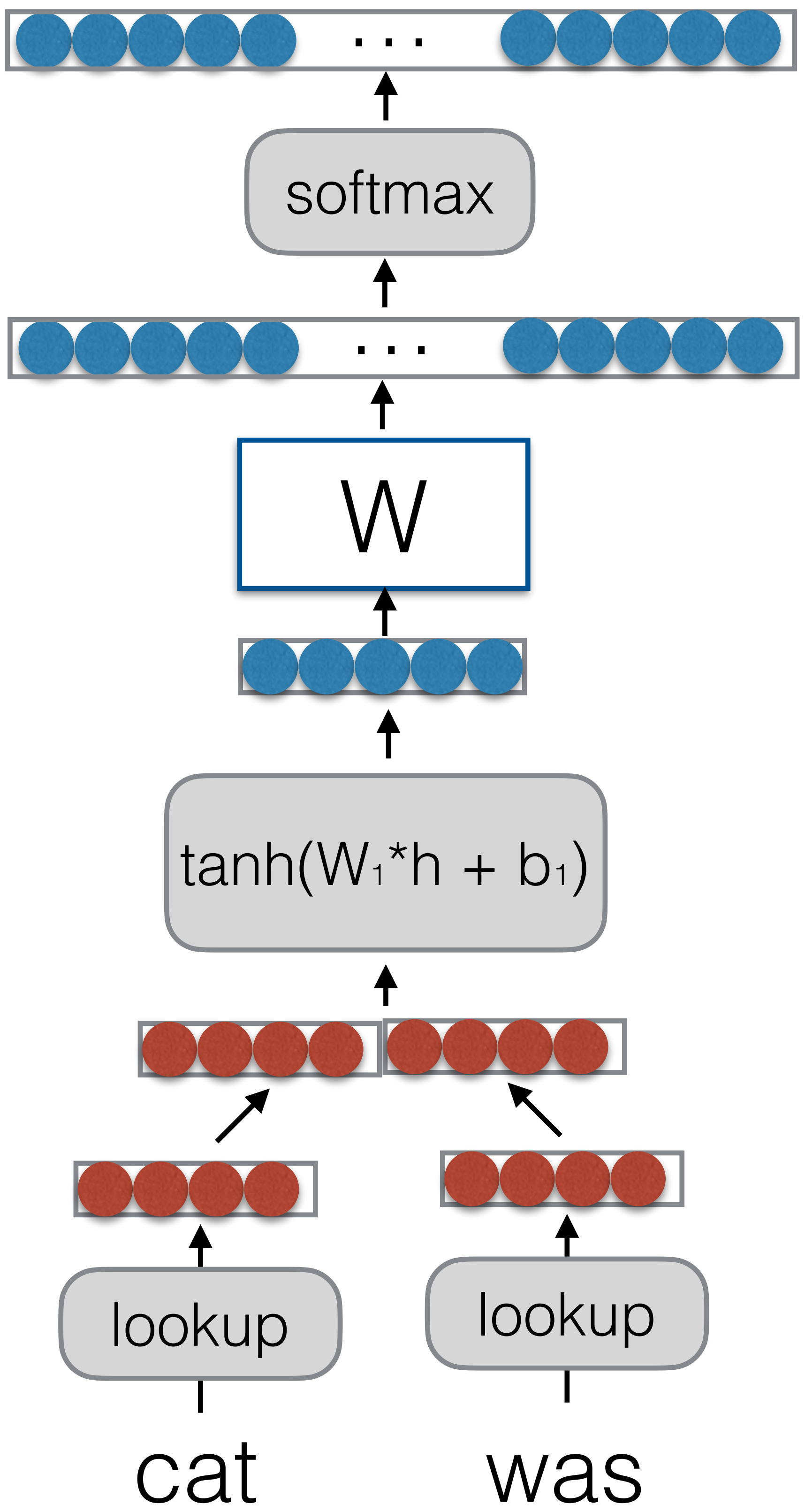
lookup

lookup

cat

was

Feedforward neural language model



$p_\theta(x_t \mid \text{cat,was})$

Logits $\in \mathbb{R}^V$

Output weight matrix

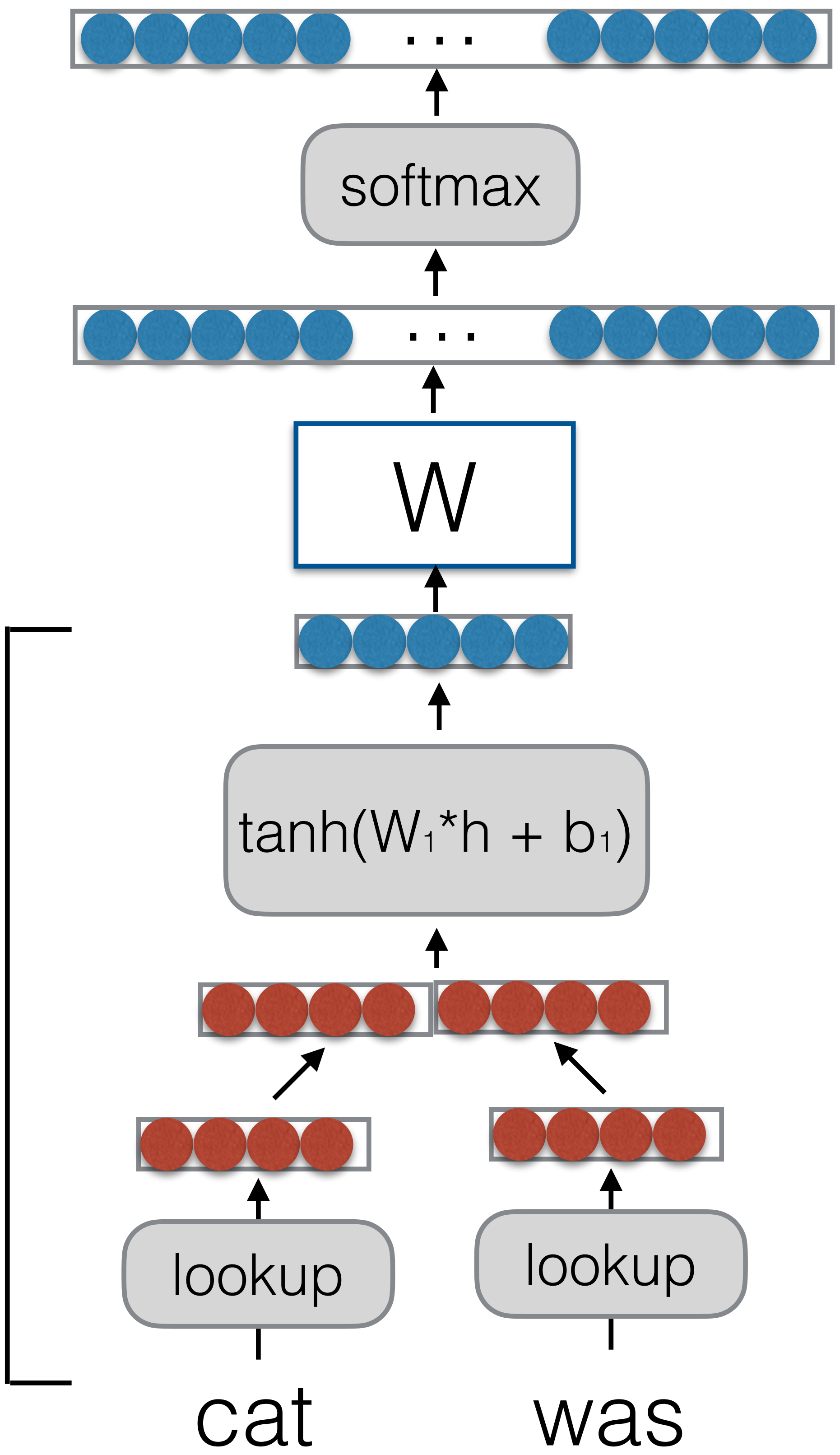
Hidden vector

Layer

Concat

Embedding

Feedforward neural language model



$p_\theta(x_t \mid \text{cat,was})$

Logits $\in \mathbb{R}^V$

Output weight matrix

Hidden vector

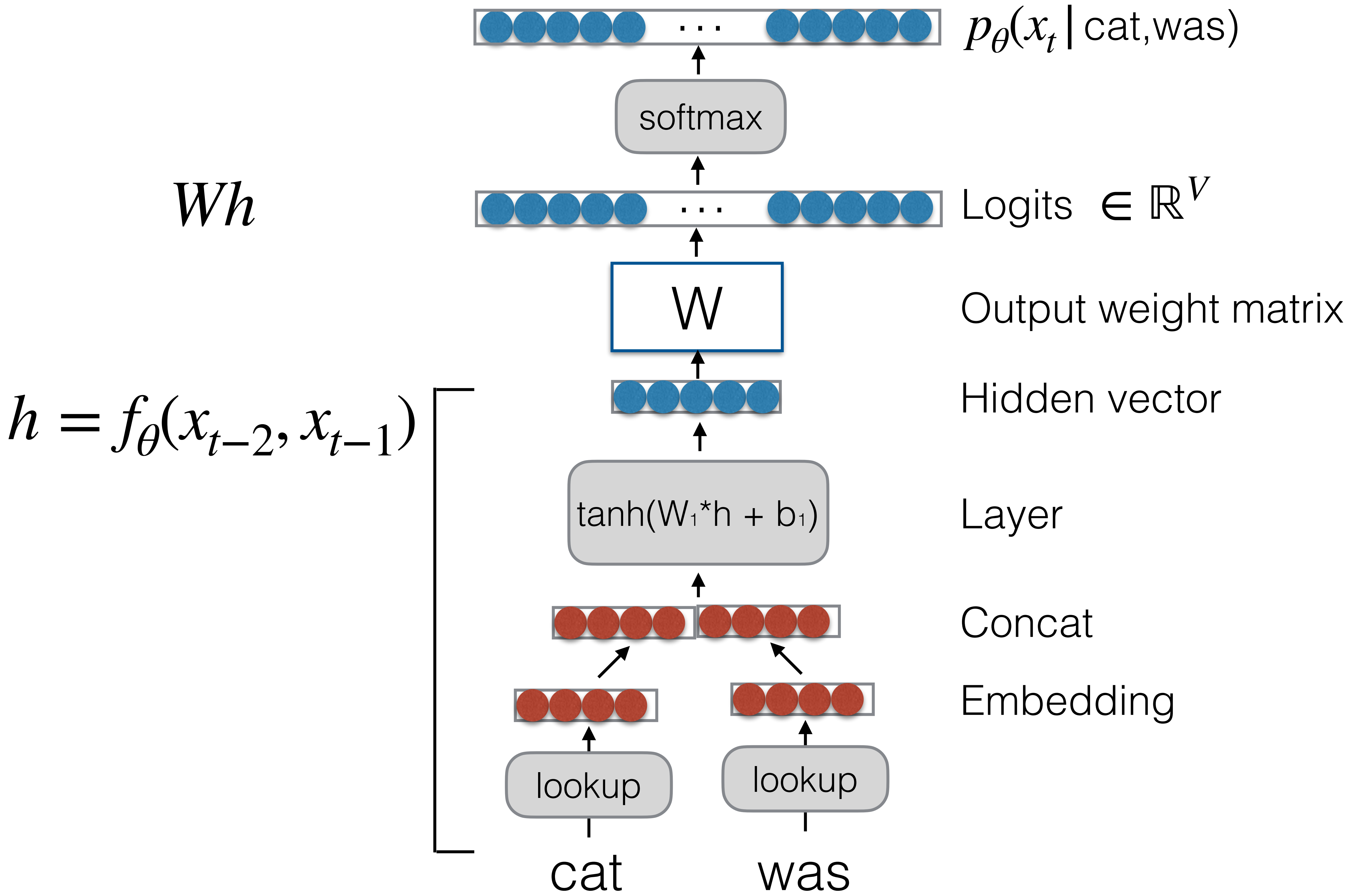
$h = f_\theta(x_{t-2}, x_{t-1})$

Layer

Concat

Embedding

Feedforward neural language model



Feedforward neural language model

- Training: maximum likelihood estimation

Feedforward neural language model

- Training: maximum likelihood estimation

$$\arg\max_{\theta} \sum_{x \in D_{train}} \log p_{\theta}(x)$$

Feedforward neural language model

- Training: maximum likelihood estimation

$$
\begin{aligned}
\arg\max_{\theta} \sum_{x \in D_{train}} \log p_\theta(x) \\
= \sum_{x \in D_{train}} \sum_{t=1}^{T} \log p_\theta(x_t \mid x_{1:t-1})
\end{aligned}
$$

Feedforward neural language model

- Training: maximum likelihood estimation

$$\arg\max_{\theta} \sum_{x \in D_{train}} \log p_{\theta}(x)$$

$$= \sum_{x \in D_{train}} \sum_{t=1}^{T} \log p_{\theta}(x_t \mid x_{1:t-1})$$

- Loss: increase probability of target next-token

Feedforward neural language model

- Training: maximum likelihood estimation

$$\arg\max_{\theta} \sum_{x \in D_{train}} \log p_{\theta}(x)$$

$$= \sum_{x \in D_{train}} \sum_{t=1}^{T} \log p_{\theta}(x_t \mid x_{1:t-1})$$

- Loss: increase probability of target next-token

Loss: $$L_t = -\log p_{\theta}(\underset{\uparrow}{x_t} \mid x_{1:t-1})$$

Feedforward neural language model

- Cross-entropy loss!

$$L = -\log p_\theta(x_t \mid x_{1:t-1})$$

Bengio et al 2003, *A Neural Probabilistic Language Model*

Feedforward neural language model

- Cross-entropy loss!

$$L = -\log p_\theta(x_t \mid x_{1:t-1})$$

  - Recall from lecture 2:

$$L_{CE} = -\sum_{i=1}^{\text{num classes}} y_i \log(p_i)$$

Bengio et al 2003, *A Neural Probabilistic Language Model*

Feedforward neural language model

- Cross-entropy loss!
  - Recall from lecture 2:
    - y_i: one-hot next-token

$$L = -\log p_\theta(x_t \mid x_{1:t-1})$$

$$L_{CE} = -\sum_{i=1}^{\text{num classes}} y_i \log(p_i)$$

Bengio et al 2003, *A Neural Probabilistic Language Model*

Feedforward neural language model

- Cross-entropy loss!
  - Recall from lecture 2:
    - y_i: one-hot next-token
    - p_i: LM probability on that token

$$L = -\log p_\theta(x_t \mid x_{1:t-1})$$

$$L_{CE} = -\sum_{i=1}^{\text{num classes}} y_i \log(p_i)$$

Bengio et al 2003, *A Neural Probabilistic Language Model*

Feedforward neural language model

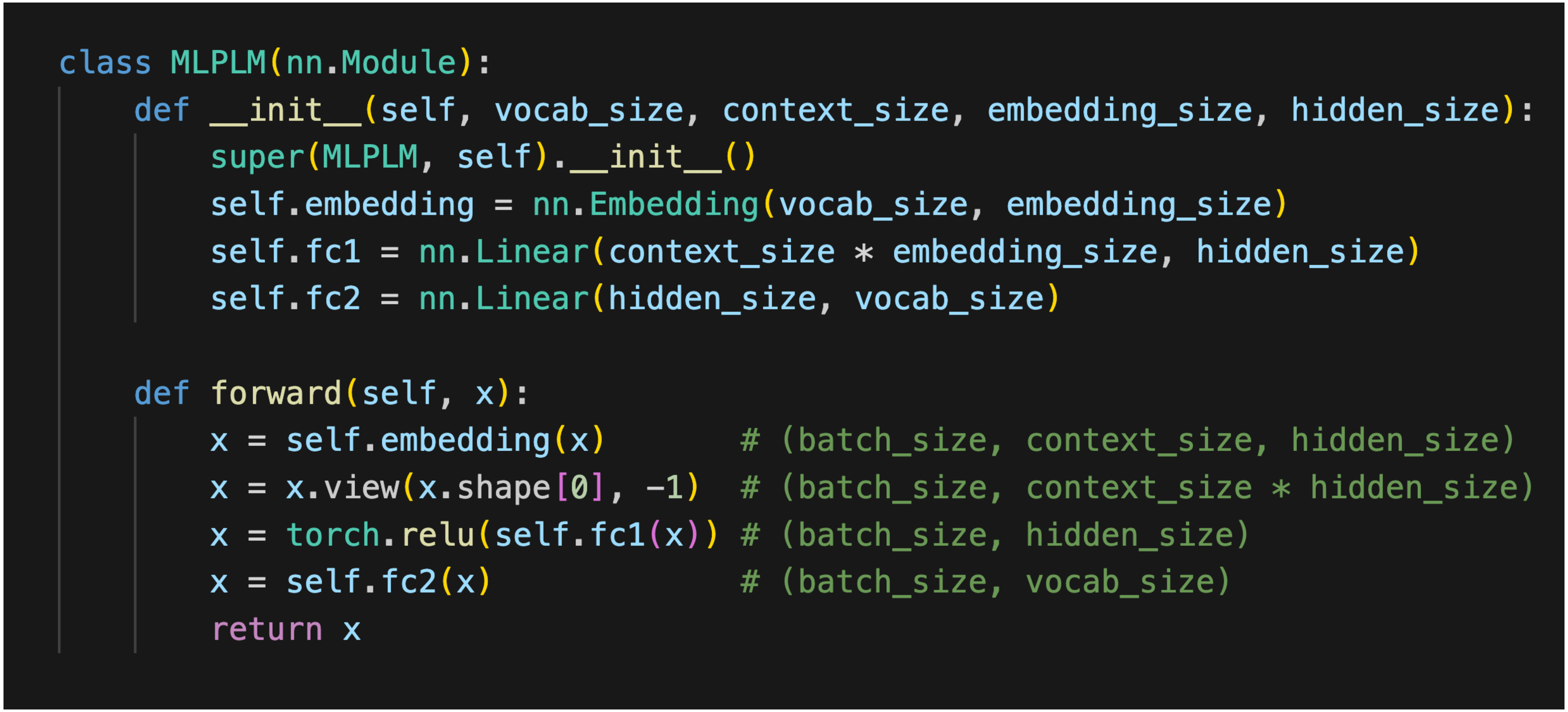
- Cross-entropy loss!
  - Recall from lecture 2:
    - y_i: one-hot next-token
    - p_i: LM probability on that token
    - Classes: possible next-tokens (vocabulary)

$$L = -\log p_\theta(x_t \mid x_{1:t-1})$$

$$L_{CE} = -\sum_{i=1}^{\text{num classes}} y_i \log(p_i)$$

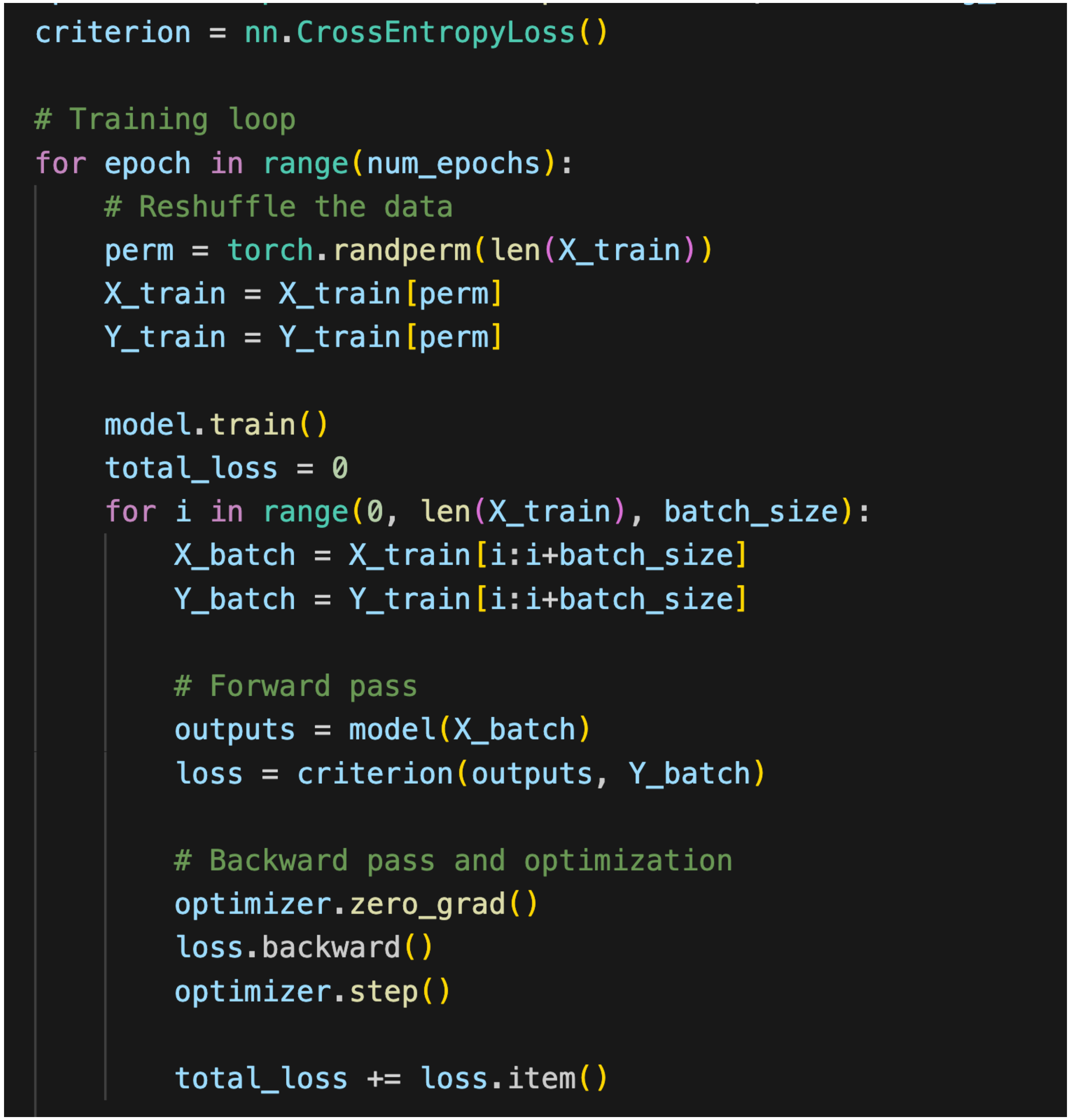
Bengio et al 2003, *A Neural Probabilistic Language Model*

In code



```python
class MLPLM(nn.Module):
    def __init__(self, vocab_size, context_size, embedding_size, hidden_size):
        super(MLPLM, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_size)
        self.fc1 = nn.Linear(context_size * embedding_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, vocab_size)

    def forward(self, x):
        x = self.embedding(x)       # (batch_size, context_size, hidden_size)
        x = x.view(x.shape[0], -1)  # (batch_size, context_size * hidden_size)
        x = torch.relu(self.fc1(x)) # (batch_size, hidden_size)
        x = self.fc2(x)             # (batch_size, vocab_size)
        return x
```

In code



```
criterion = nn.CrossEntropyLoss()

# Training loop
for epoch in range(num_epochs):
    # Reshuffle the data
    perm = torch.randperm(len(X_train))
    X_train = X_train[perm]
    Y_train = Y_train[perm]

    model.train()
    total_loss = 0
    for i in range(0, len(X_train), batch_size):
        X_batch = X_train[i:i+batch_size]
        Y_batch = Y_train[i:i+batch_size]

        # Forward pass
        outputs = model(X_batch)
        loss = criterion(outputs, Y_batch)

        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
```

Example of Combination Features

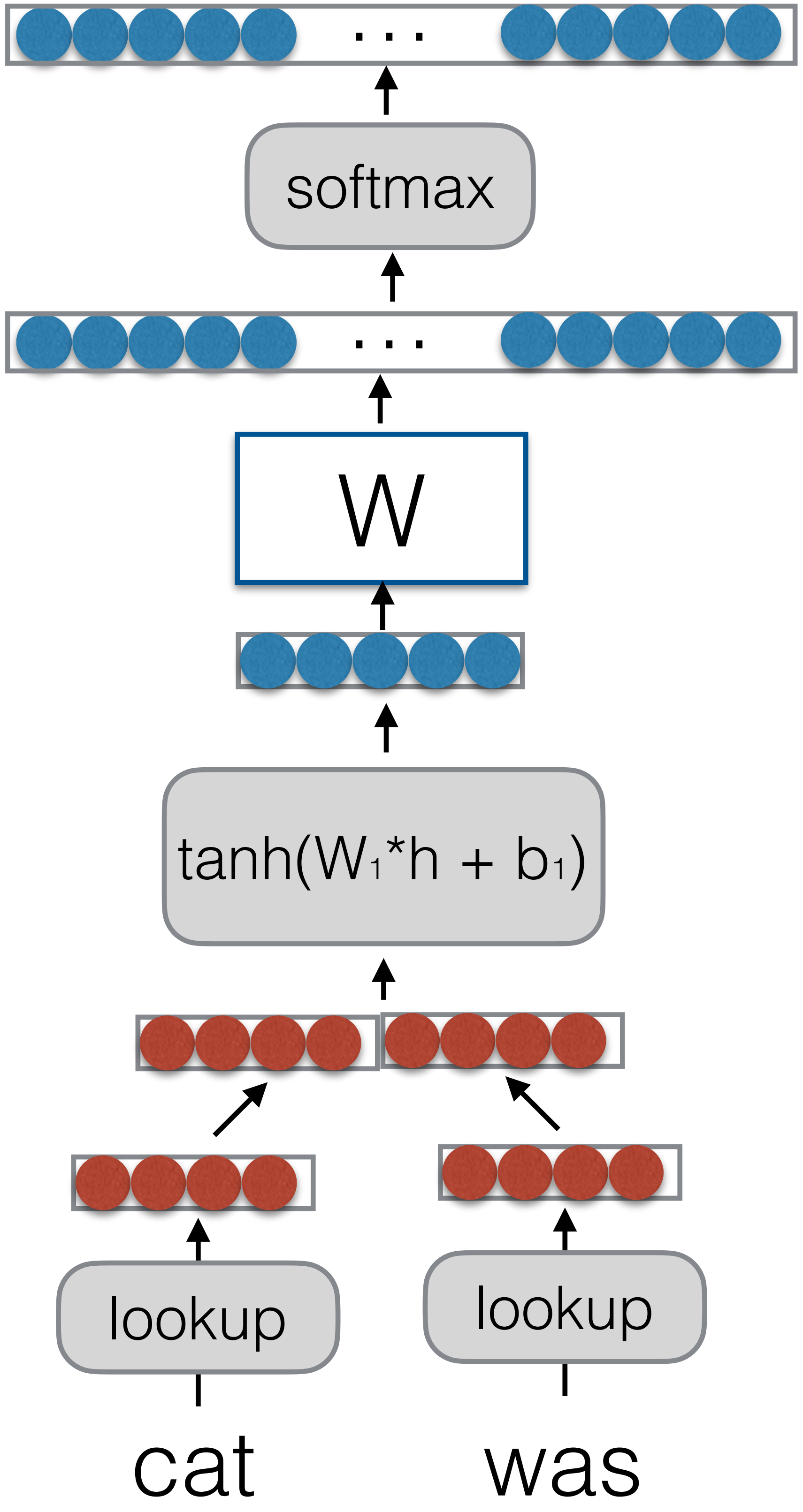
- A row in the weight matrix can capture particular *combinations* of token embedding features
  - E.g. the 34th row in the weight matrix:

$$\begin{array}{cc} \text{giving} & \begin{bmatrix}1.2\\-0.1\\0.7\\-2.1\\0.5\end{bmatrix} \\ \text{a} & \begin{bmatrix}-0.3\\2.0\\0.6\\-0.8\\-0.4\end{bmatrix}\end{array} * \overset{\mathbf{w}_{34}}{\begin{bmatrix}1.5\\0\\0\\0\\0\\ \\0\\1.3\\0\\0\\0\end{bmatrix}} + \overset{b_{34}}{-2} =$$

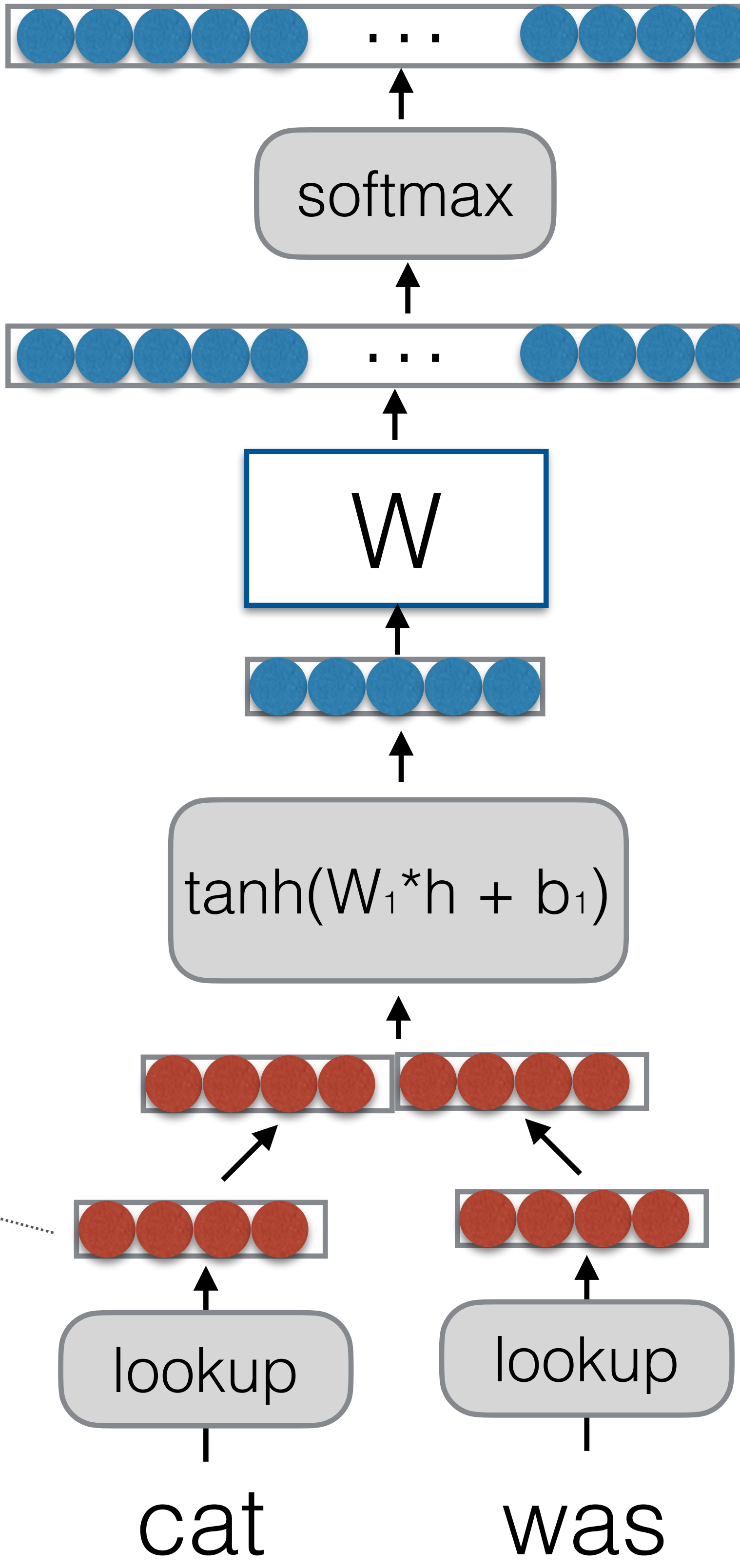
Example possibility:

positive number if the previous word is a determiner and second-to-previous word is a verb

Where is strength shared?

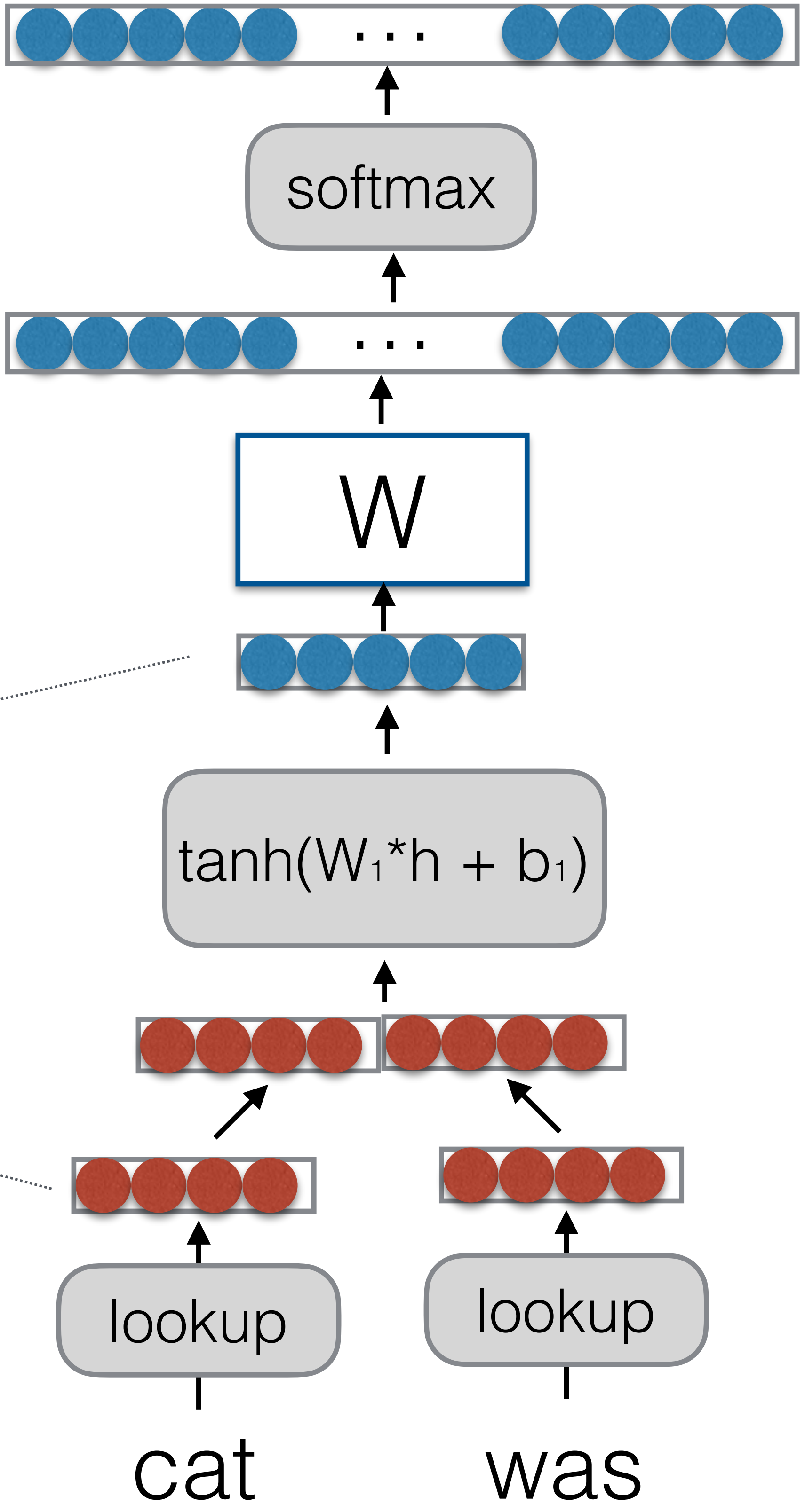


Where is strength shared?



Similar words get similar embeddings

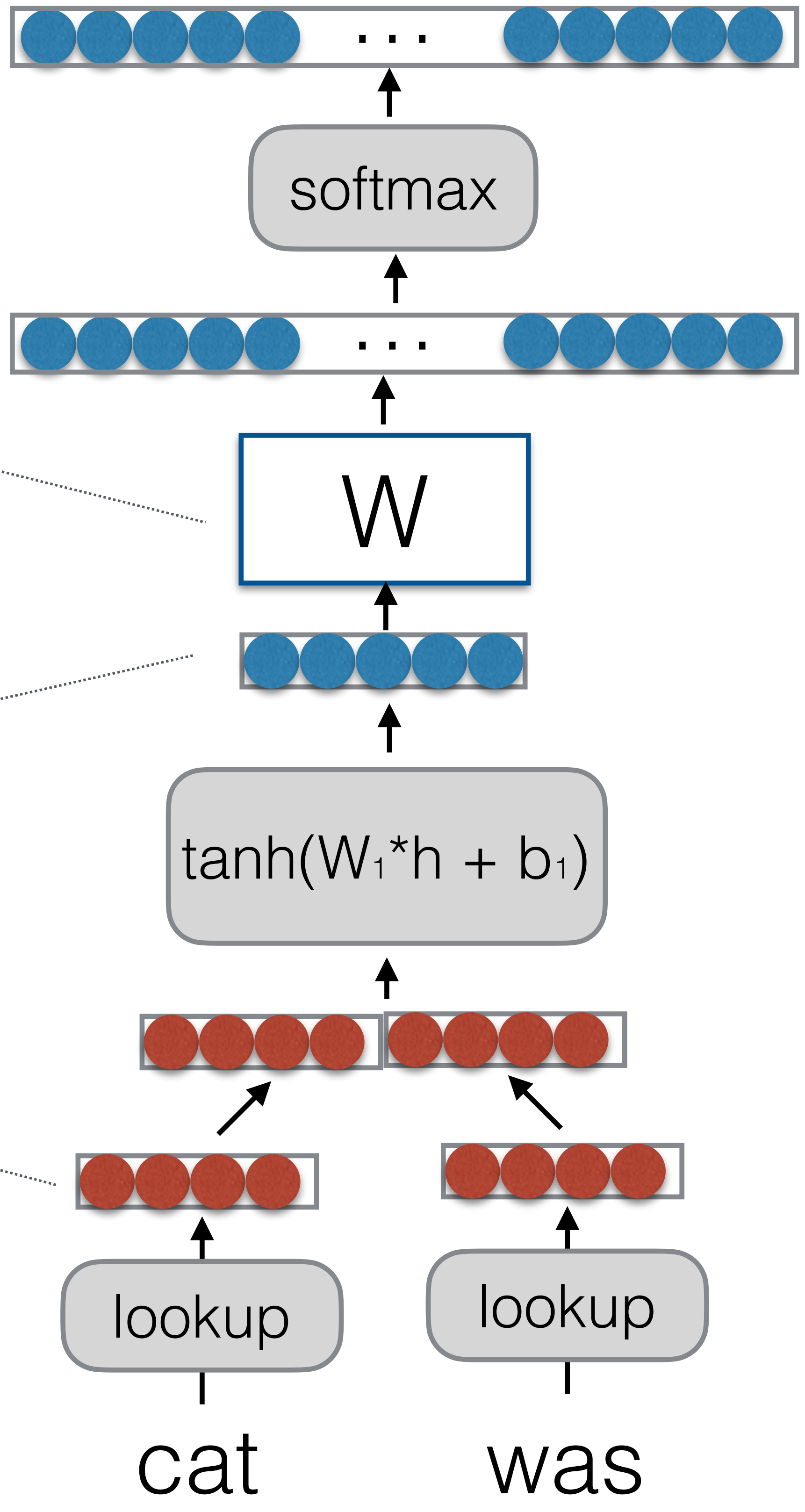
Where is strength shared?



Similar contexts get similar hidden states

Similar words get similar embeddings

Where is strength shared?



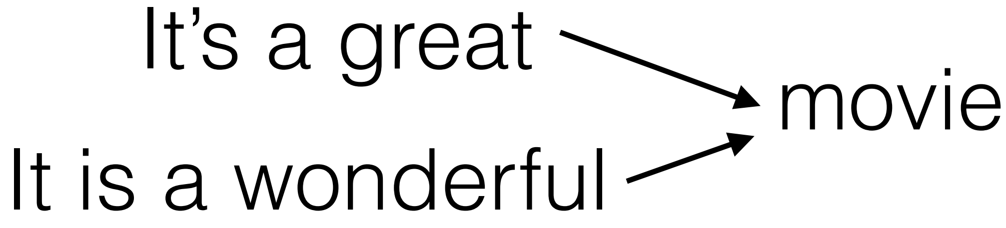
Similar output words get similar output weights

Similar contexts get similar hidden states

Similar words get similar embeddings

Where is strength shared?

- Consider predicting word w with two similar contexts h_j and h_k



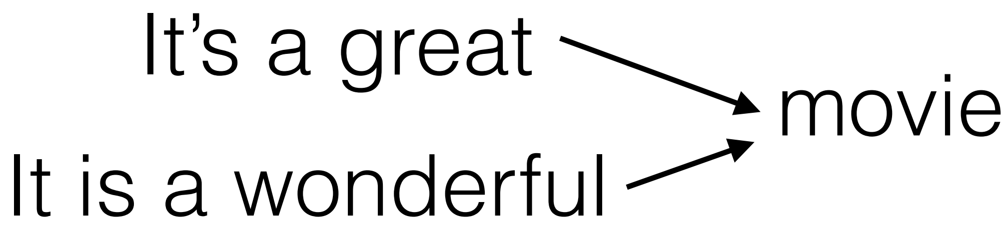
It's a great

It is a wonderful

movie

Where is strength shared?

- Consider predicting word w with two similar contexts h_j and h_k
- $p_j^w = p(w \mid h_j) = \frac{1}{Z_j} \exp\left(w^\top h_j\right)$



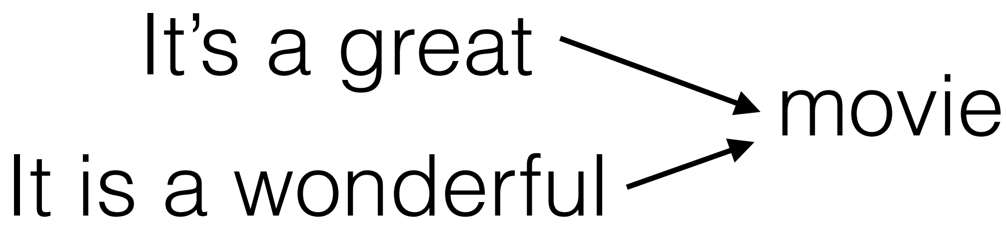
It's a great

It is a wonderful

movie

Where is strength shared?

- Consider predicting word w with two similar contexts h_j and h_k
- $p_j^w = p(w \mid h_j) = \dfrac{1}{Z_j} \exp\left(w^\top h_j\right)$
- $p_k^w = p(w \mid h_k) = \dfrac{1}{Z_k} \exp\left(w^\top h_k\right)$



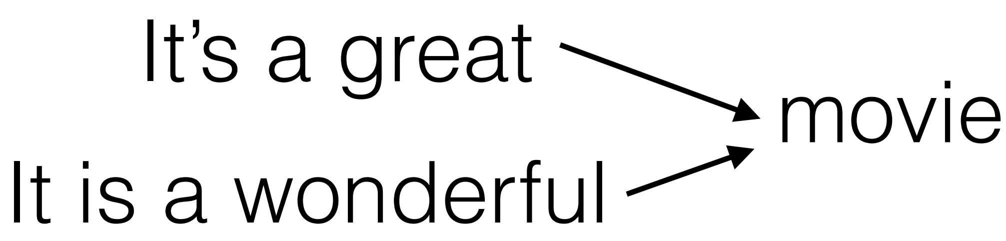
It's a great

It is a wonderful

movie

Where is strength shared?

- Consider predicting word w with two similar contexts h_j and h_k
- $p_j^w = p(w \mid h_j) = \frac{1}{Z_j} \exp\left(w^\top h_j\right)$
- $p_k^w = p(w \mid h_k) = \frac{1}{Z_k} \exp\left(w^\top h_k\right)$
- $\frac{p_j^w}{p_k^w} = \frac{Z_k}{Z_j} \exp\left(w^\top (h_j - h_k)\right)$



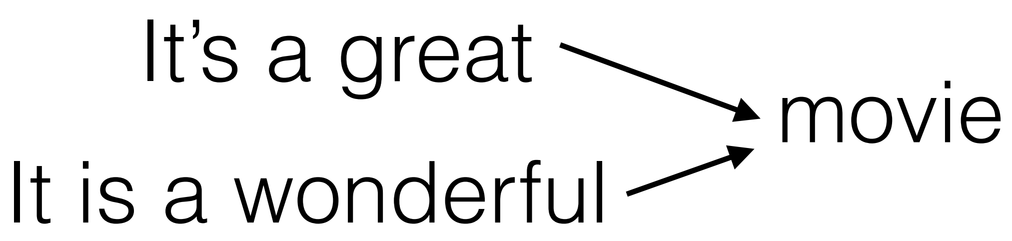
It's a great

It is a wonderful

movie

Where is strength shared?

- Consider predicting word w with two similar contexts h_j and h_k
- $p_j^w = p(w \mid h_j) = \frac{1}{Z_j} \exp\left(w^\top h_j\right)$
- $p_k^w = p(w \mid h_k) = \frac{1}{Z_k} \exp\left(w^\top h_k\right)$
- $\frac{p_j^w}{p_k^w} = \frac{Z_k}{Z_j} \exp\left(w^\top (h_j - h_k)\right)$
- The ratio is 1 when $w^\top (h_j - h_k) = 0$


It's a great

It is a wonderful

movie

"make hidden vectors h_j and h_k close to each other"

What Problems are Handled?

- Cannot share strength among **similar words**

she bought a car
she purchased a car

she bought a bicycle
she purchased a bicycle

→ solved, and similar contexts as well! 😀

- Cannot condition on context with **intervening words**

Dr. Jane Smith   Dr. Gertrude Smith

→ solved! 😀

- Cannot handle **long-distance dependencies**

for tennis class he wanted to buy his own racquet
for programming class he wanted to buy his own computer

→ not solved yet 😞

Recap

Recap

- Bigram language models and fundamental concepts

Recap

- Bigram language models and fundamental concepts
- Ngram language models: count-based

Recap

- Bigram language models and fundamental concepts
- Ngram language models: count-based
- Neural network language model

Recap

- Bigram language models and fundamental concepts
- Ngram language models: count-based
- Neural network language model
- Next: some important practical concepts

Important practical concepts

Important practical concepts

- A deep learning system has multiple moving parts:
  - The model architecture, the optimizer, the weights, the hyperparameters, …

Important practical concepts

- A deep learning system has multiple moving parts:
    - The model architecture, the optimizer, the weights, the hyperparameters, …
- We want our experiments to give us data that leads to reliable conclusions

Important practical concepts

- A deep learning system has multiple moving parts:
  - The model architecture, the optimizer, the weights, the hyperparameters, …
- We want our experiments to give us data that leads to reliable conclusions
- Here are a few helpful ideas that are often implicit in most deep learning experiments

Splitting into train, valid, and test

Splitting into train, valid, and test

- Goal: fit a target distribution p_*

Splitting into train, valid, and test

- Goal: fit a target distribution p_*
  - **Training data**: samples from p_*, used to fit the model p_θ

Splitting into train, valid, and test

- Goal: fit a target distribution p_*
  - **Training data**: samples from p_*, used to fit the model p_θ
  - **Validation data**: hold out samples from p_* to check generalization. We try different configurations and choose one with good generalization.

Splitting into train, valid, and test

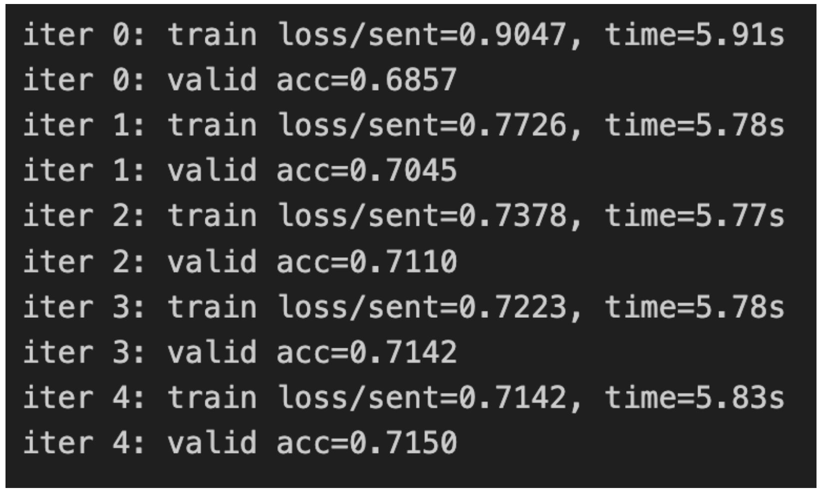
- Goal: fit a target distribution p_*
  - **Training data**: samples from p_*, used to fit the model p_θ
  - **Validation data**: hold out samples from p_* to check generalization. We try different configurations and choose one with good generalization.
  - **Test data**: hold out samples from p_* as an unbiased check of the final configuration's generalization

Splitting into train, valid, and test

- In other words:
  - **Training data**: use it to train the model
  - **Validation data**: use it to tune hyperparameters, perform ablations, select a model
  - **Test data**: use it *once at the end* and don't look at it during development

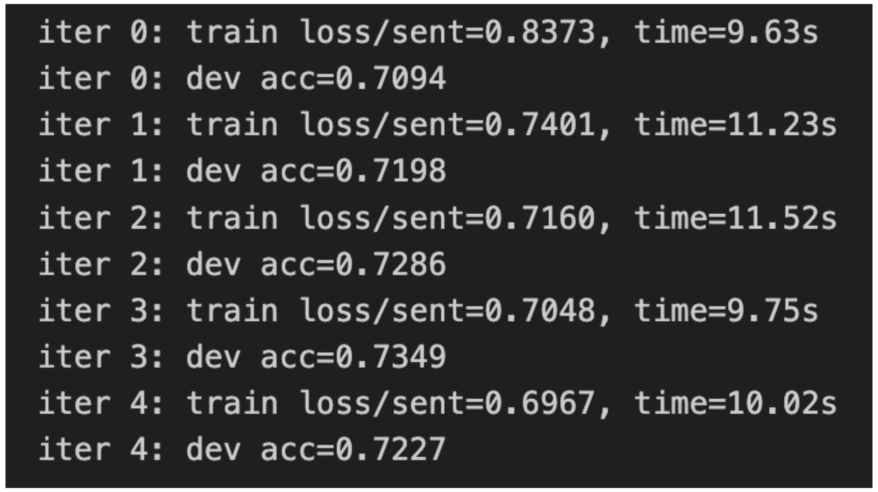
Splitting into train, valid, and test

Model 1



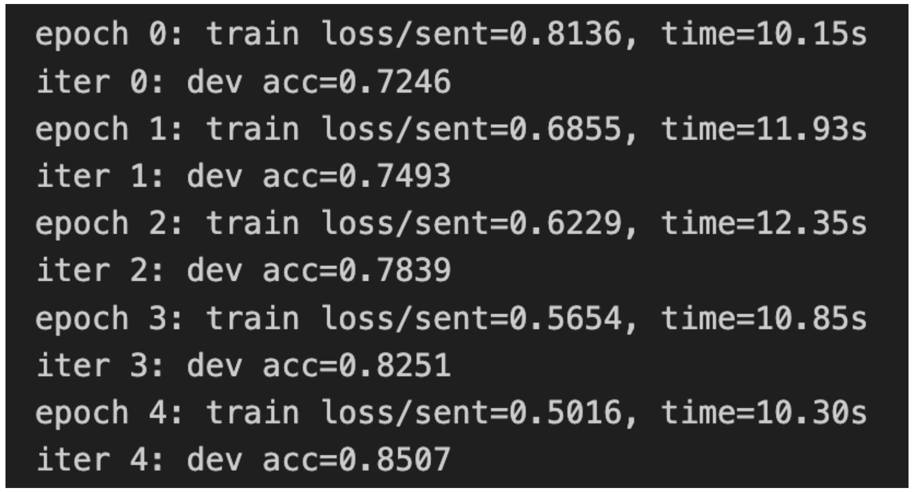
```
iter 0: train loss/sent=0.9047, time=5.91s
iter 0: valid acc=0.6857
iter 1: train loss/sent=0.7726, time=5.78s
iter 1: valid acc=0.7045
iter 2: train loss/sent=0.7378, time=5.77s
iter 2: valid acc=0.7110
iter 3: train loss/sent=0.7223, time=5.78s
iter 3: valid acc=0.7142
iter 4: train loss/sent=0.7142, time=5.83s
iter 4: valid acc=0.7150
```

Model 2



```
iter 0: train loss/sent=0.8373, time=9.63s
iter 0: dev acc=0.7094
iter 1: train loss/sent=0.7401, time=11.23s
iter 1: dev acc=0.7198
iter 2: train loss/sent=0.7160, time=11.52s
iter 2: dev acc=0.7286
iter 3: train loss/sent=0.7048, time=9.75s
iter 3: dev acc=0.7349
iter 4: train loss/sent=0.6967, time=10.02s
iter 4: dev acc=0.7227
```

Model 3



```
epoch 0: train loss/sent=0.8136, time=10.15s
iter 0: dev acc=0.7246
epoch 1: train loss/sent=0.6855, time=11.93s
iter 1: dev acc=0.7493
epoch 2: train loss/sent=0.6229, time=12.35s
iter 2: dev acc=0.7839
epoch 3: train loss/sent=0.5654, time=10.85s
iter 3: dev acc=0.8251
epoch 4: train loss/sent=0.5016, time=10.30s
iter 4: dev acc=0.8507
```

From bow.ipynb: based on this information, which model would you select?

Overfitting

- Goal: fit a target distribution p_*
  - The model may fit the training data (a sample from p_*), but the model may not generalize

Overfitting

- Goal: fit a target distribution p_*
  - The model may fit the training data (a sample from p_*), but the model may not generalize
- **Symptom**: training loss is decreasing, validation loss is increasing

Overfitting

- Goal: fit a target distribution p_*
  - The model may fit the training data (a sample from p_*), but the model may not generalize
- **Symptom**: training loss is decreasing, validation loss is increasing
  - Choose different hyperparameters

Overfitting

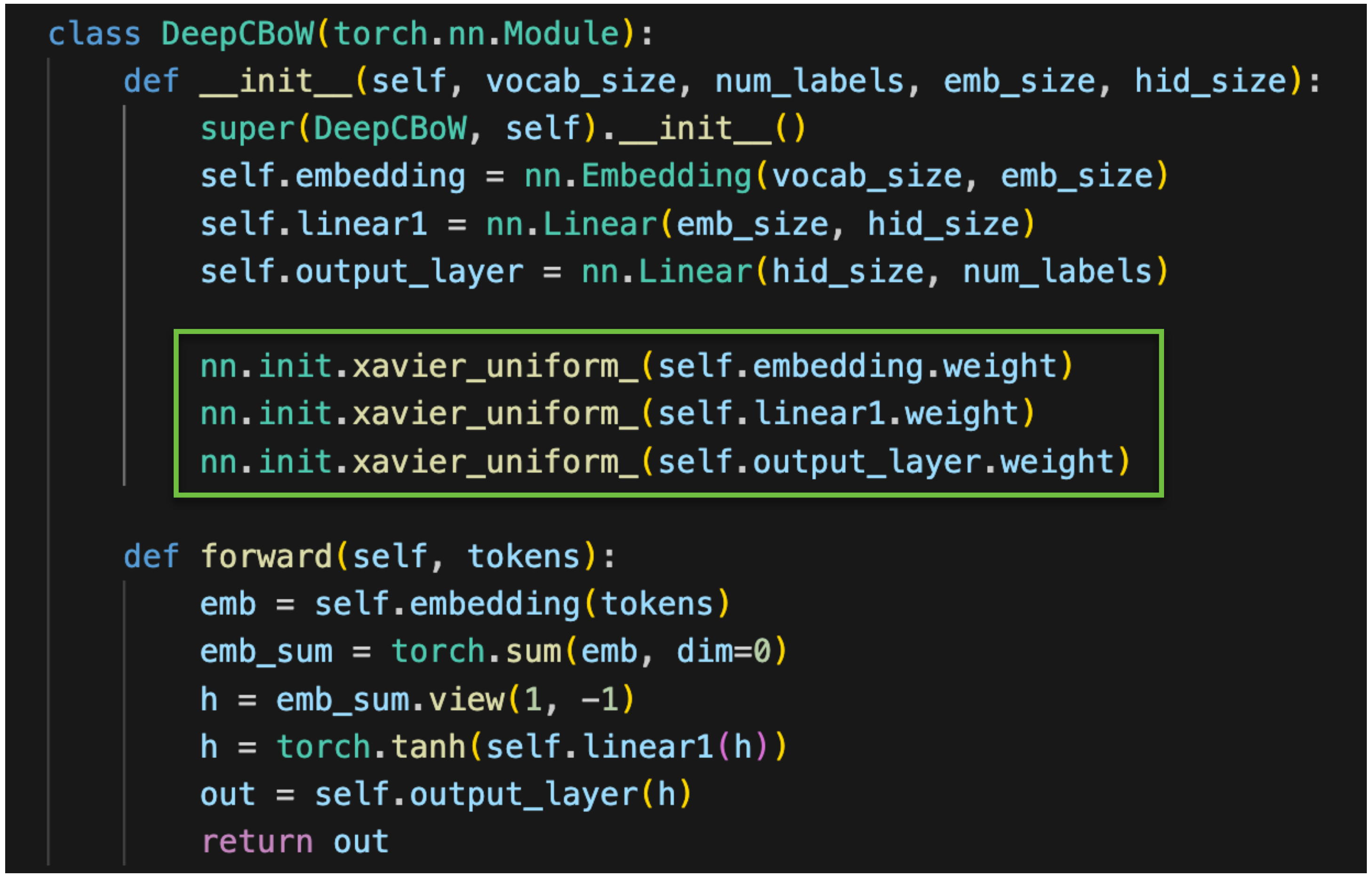
- Goal: fit a target distribution p_*
  - The model may fit the training data (a sample from p_*), but the model may not generalize
- **Symptom**: training loss is decreasing, validation loss is increasing
  - Choose different hyperparameters
  - Add regularization

Overfitting

- Goal: fit a target distribution p_*
  - The model may fit the training data (a sample from p_*), but the model may not generalize
- **Symptom**: training loss is decreasing, validation loss is increasing
  - Choose different hyperparameters
  - Add regularization
  - Choose the model with minimum validation loss

Initialization

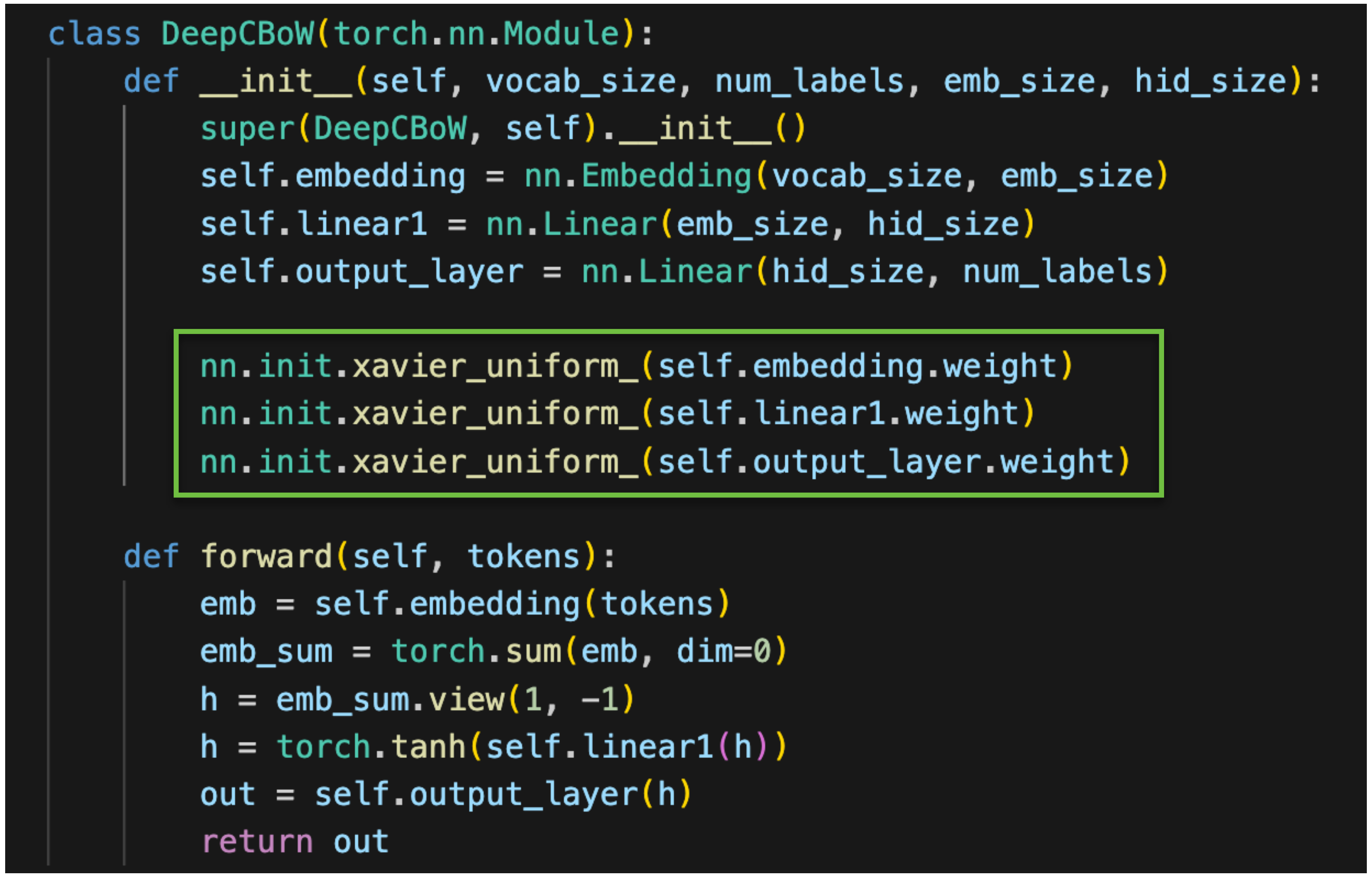
- Weight initialization impacts the optimization trajectory



```
class DeepCBoW(torch.nn.Module):
    def __init__(self, vocab_size, num_labels, emb_size, hid_size):
        super(DeepCBoW, self).__init__()
        self.embedding = nn.Embedding(vocab_size, emb_size)
        self.linear1 = nn.Linear(emb_size, hid_size)
        self.output_layer = nn.Linear(hid_size, num_labels)

        nn.init.xavier_uniform_(self.embedding.weight)
        nn.init.xavier_uniform_(self.linear1.weight)
        nn.init.xavier_uniform_(self.output_layer.weight)

    def forward(self, tokens):
        emb = self.embedding(tokens)
        emb_sum = torch.sum(emb, dim=0)
        h = emb_sum.view(1, -1)
        h = torch.tanh(self.linear1(h))
        out = self.output_layer(h)
        return out
```

Initialization

- Weight initialization impacts the optimization trajectory

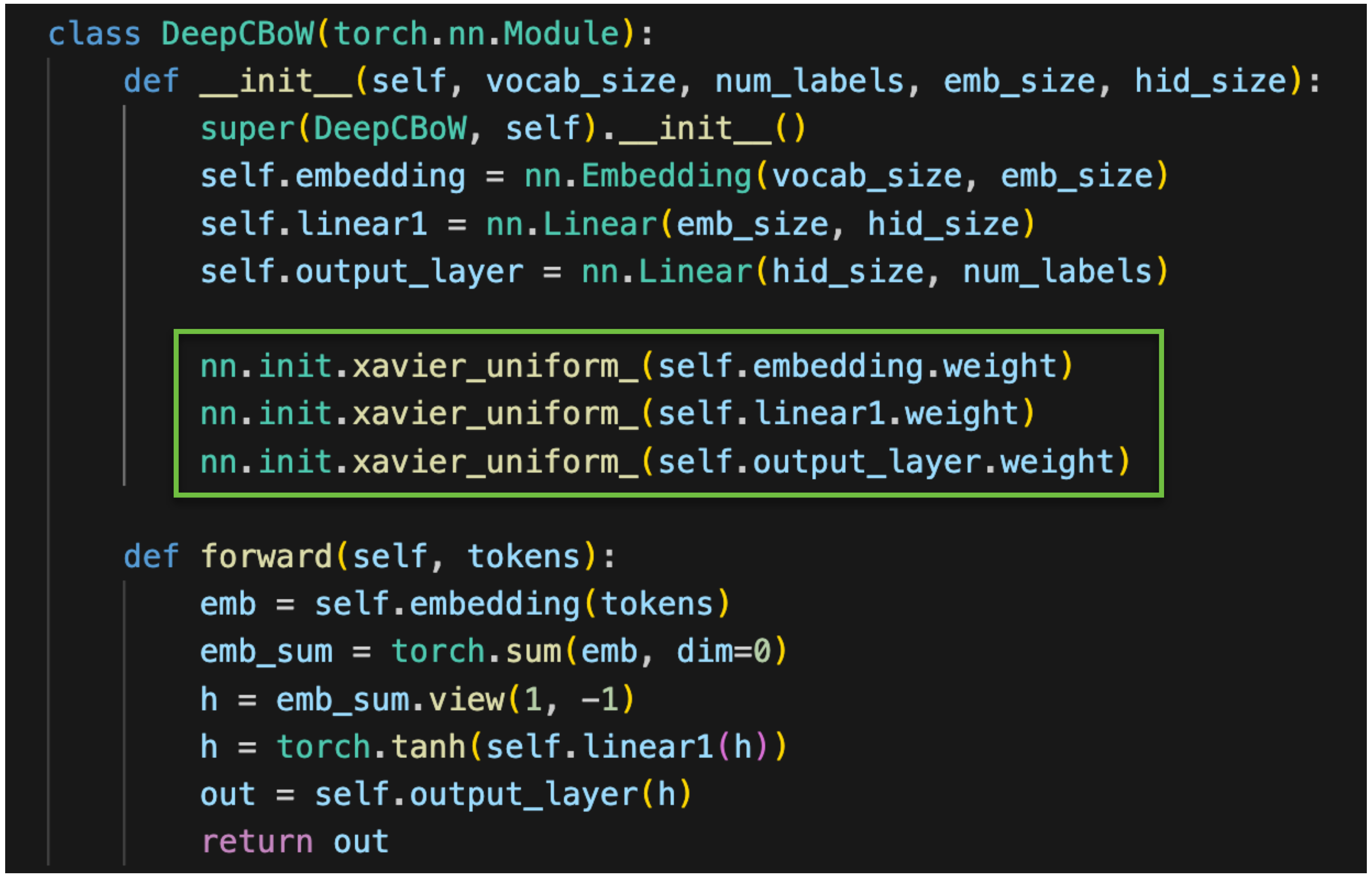


```
class DeepCBoW(torch.nn.Module):
    def __init__(self, vocab_size, num_labels, emb_size, hid_size):
        super(DeepCBoW, self).__init__()
        self.embedding = nn.Embedding(vocab_size, emb_size)
        self.linear1 = nn.Linear(emb_size, hid_size)
        self.output_layer = nn.Linear(hid_size, num_labels)

        nn.init.xavier_uniform_(self.embedding.weight)
        nn.init.xavier_uniform_(self.linear1.weight)
        nn.init.xavier_uniform_(self.output_layer.weight)

    def forward(self, tokens):
        emb = self.embedding(tokens)
        emb_sum = torch.sum(emb, dim=0)
        h = emb_sum.view(1, -1)
        h = torch.tanh(self.linear1(h))
        out = self.output_layer(h)
        return out
```

Xavier initialization [Glorot and Bengio 2010]: $W \sim \mathcal{U}\left(-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}\right)$

Initialization

- Weight initialization impacts the optimization trajectory



```
class DeepCBoW(torch.nn.Module):
    def __init__(self, vocab_size, num_labels, emb_size, hid_size):
        super(DeepCBoW, self).__init__()
        self.embedding = nn.Embedding(vocab_size, emb_size)
        self.linear1 = nn.Linear(emb_size, hid_size)
        self.output_layer = nn.Linear(hid_size, num_labels)

        nn.init.xavier_uniform_(self.embedding.weight)
        nn.init.xavier_uniform_(self.linear1.weight)
        nn.init.xavier_uniform_(self.output_layer.weight)

    def forward(self, tokens):
        emb = self.embedding(tokens)
        emb_sum = torch.sum(emb, dim=0)
        h = emb_sum.view(1, -1)
        h = torch.tanh(self.linear1(h))
        out = self.output_layer(h)
        return out
```

Xavier initialization [Glorot and Bengio 2010]: $W \sim \mathcal{U}\left(-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}\right)$

Weights are drawn from a uniform distribution around zero, scaled to balance variance across layers.

Optimizer: Adam

Optimizer: Adam

- Most standard optimization option in NLP and beyond

Optimizer: Adam

- Most standard optimization option in NLP and beyond
- Considers rolling average of gradient, and momentum

Optimizer: Adam

- Most standard optimization option in NLP and beyond
- Considers rolling average of gradient, and momentum

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t \odot g_t$$

Optimizer: Adam

- Most standard optimization option in NLP and beyond
- Considers rolling average of gradient, and momentum

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

Momentum

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t \odot g_t$$

Optimizer: Adam

- Most standard optimization option in NLP and beyond
- Considers rolling average of gradient, and momentum

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

Momentum

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t \odot g_t$$

Rolling Average of Gradient

Optimizer: Adam

- Most standard optimization option in NLP and beyond
- Considers rolling average of gradient, and momentum

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$ Momentum

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t \odot g_t$$ Rolling Average of Gradient

- Correction of bias early in training

Optimizer: Adam

- Most standard optimization option in NLP and beyond
- Considers rolling average of gradient, and momentum

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

Momentum

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t \odot g_t$$

Rolling Average of Gradient

- Correction of bias early in training

$$\hat{m}_t = \frac{m_t}{1 - (\beta_1)^t}$$

Optimizer: Adam

- Most standard optimization option in NLP and beyond
- Considers rolling average of gradient, and momentum

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$ Momentum

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t \odot g_t$$ Rolling Average of Gradient

- Correction of bias early in training

$$\hat{m}_t = \frac{m_t}{1 - (\beta_1)^t} \quad \hat{v}_t = \frac{v_t}{1 - (\beta_2)^t}$$

Optimizer: Adam

- Most standard optimization option in NLP and beyond
- Considers rolling average of gradient, and momentum

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

Momentum

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t \odot g_t$$

Rolling Average of Gradient

- Correction of bias early in training

$$\hat{m}_t = \frac{m_t}{1 - (\beta_1)^t} \quad \hat{v}_t = \frac{v_t}{1 - (\beta_2)^t}$$

- Final update

Optimizer: Adam

- Most standard optimization option in NLP and beyond
- Considers rolling average of gradient, and momentum

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$ Momentum

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t \odot g_t$$ Rolling Average of Gradient

- Correction of bias early in training

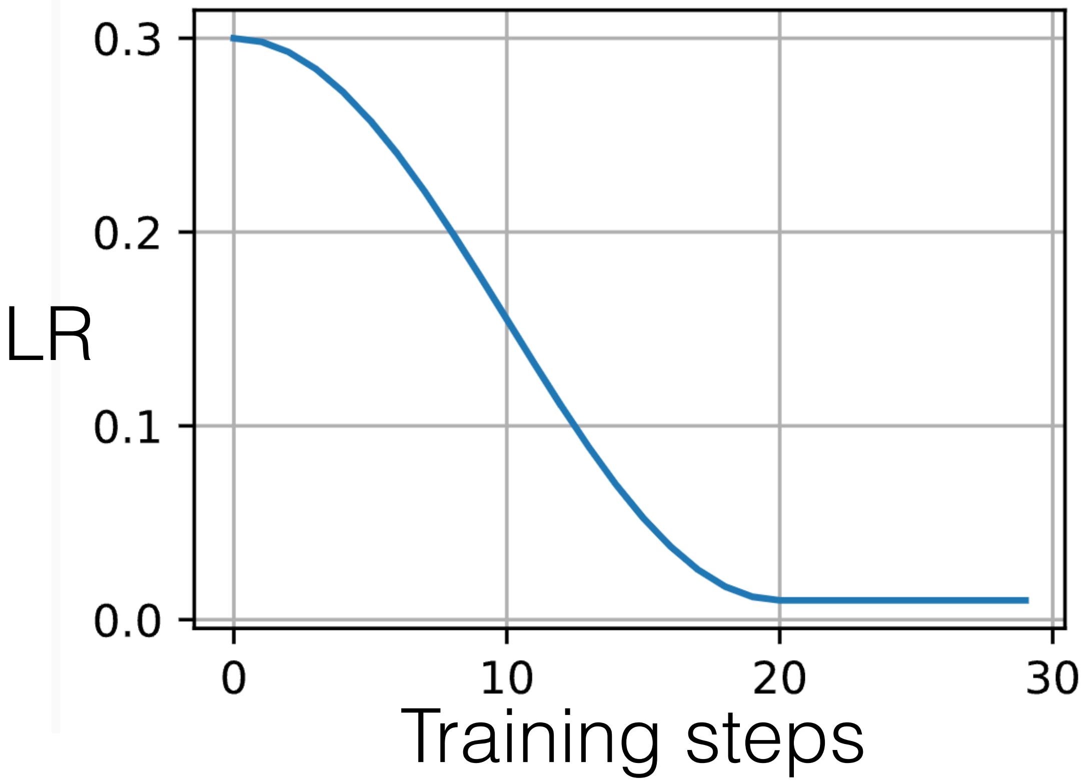
$$\hat{m}_t = \frac{m_t}{1 - (\beta_1)^t} \quad \hat{v}_t = \frac{v_t}{1 - (\beta_2)^t}$$

- Final update

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

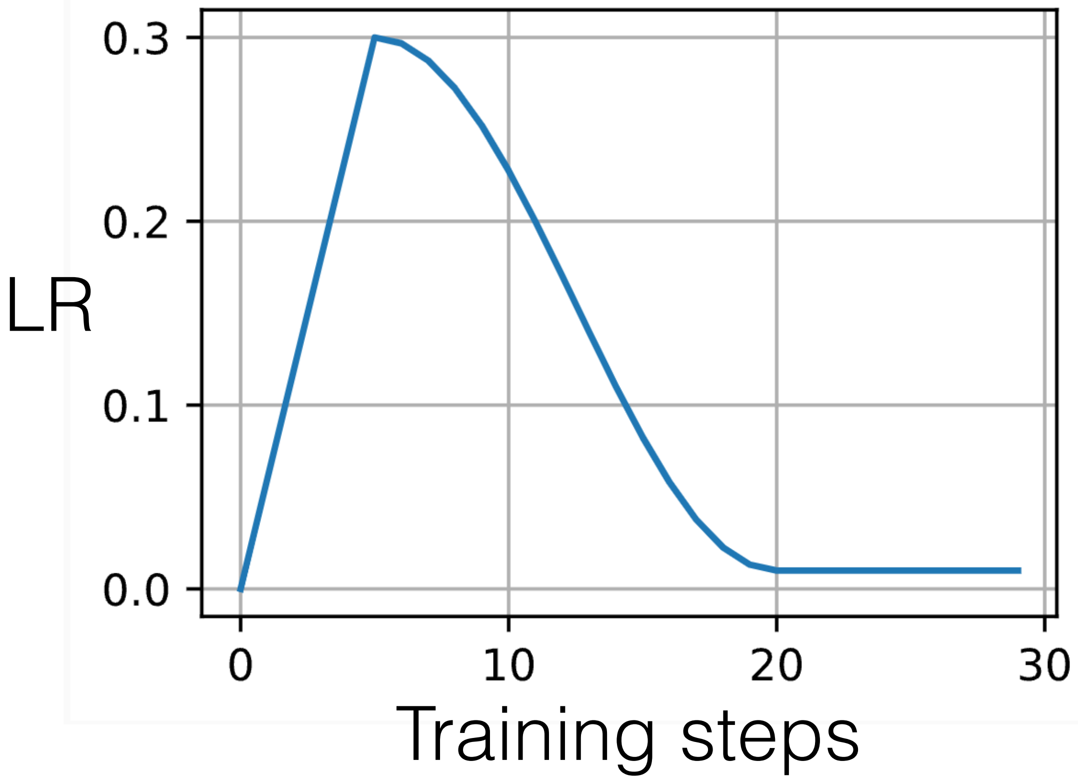
Learning rate schedule & warmup

Cosine schedule



- A *schedule* can help balance between exploration (large updates) and convergence (small updates)

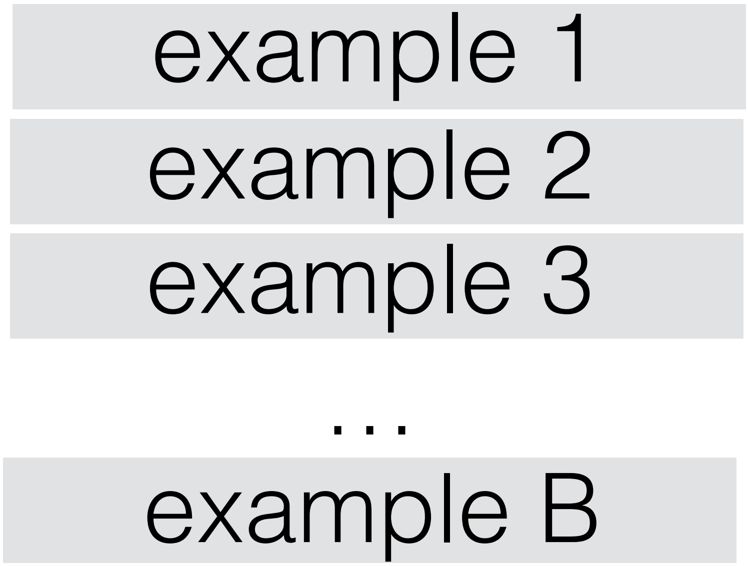
With warmup



- *Warmup* can help stabilize gradients early in training

Batching

- We typically process multiple examples at once (a batch)
  - Takes advantage of parallel hardware (GPU)
  - Can smooth out noise in individual gradients



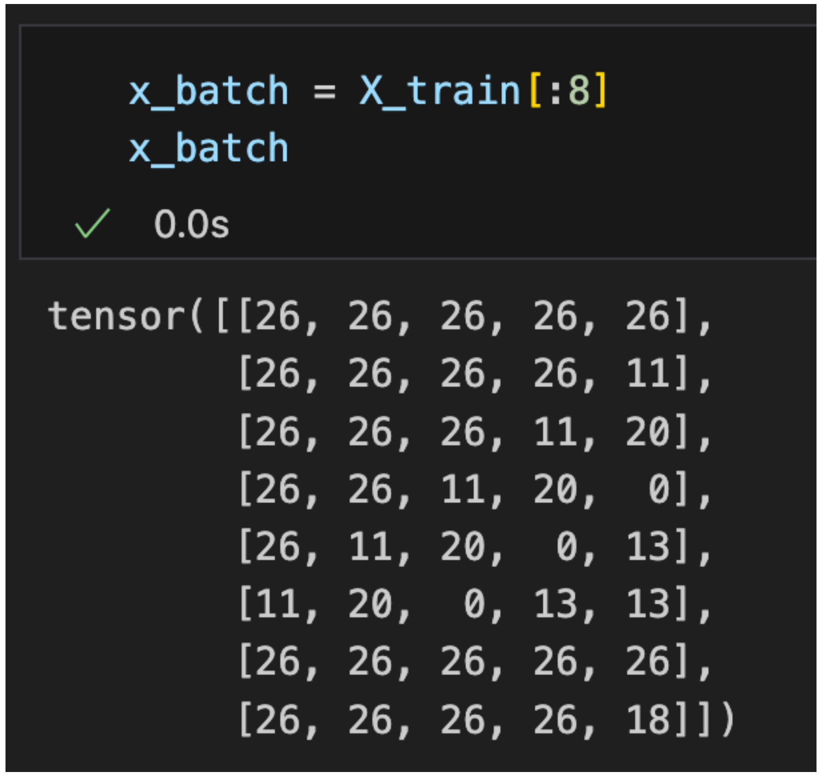
example 1

example 2

example 3

…

example B



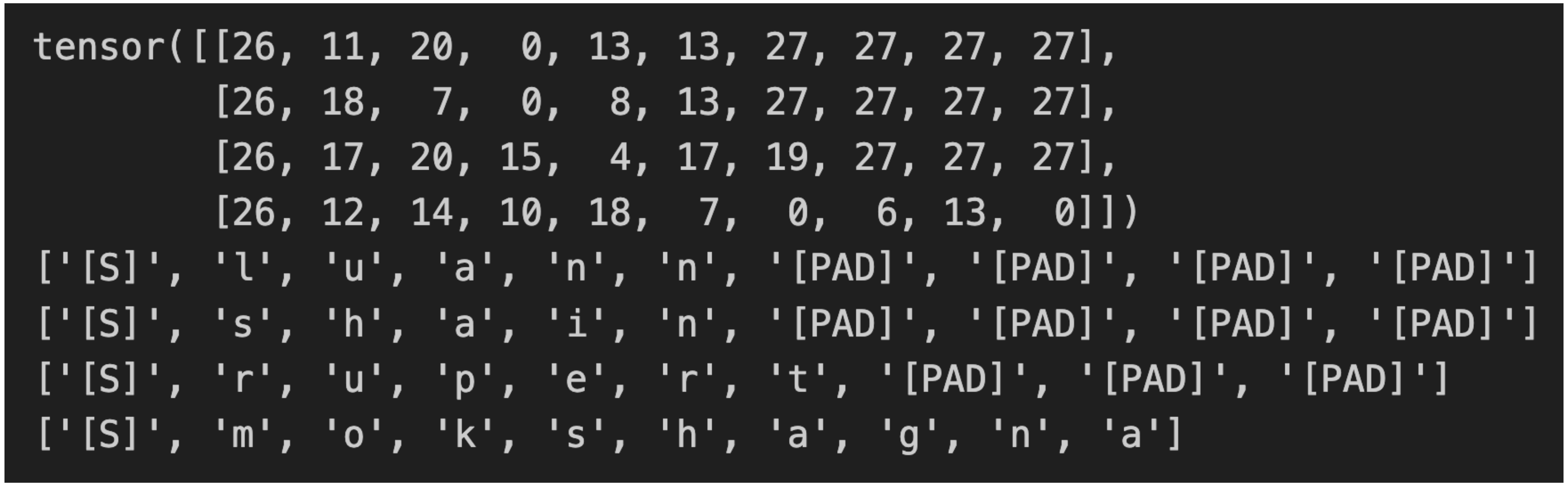
```
x_batch = X_train[:8]
x_batch
```

✓ 0.0s

```
tensor([[26, 26, 26, 26, 26],
        [26, 26, 26, 26, 11],
        [26, 26, 26, 11, 20],
        [26, 26, 11, 20,  0],
        [26, 11, 20,  0, 13],
        [11, 20,  0, 13, 13],
        [26, 26, 26, 26, 26],
        [26, 26, 26, 26, 18]])
```

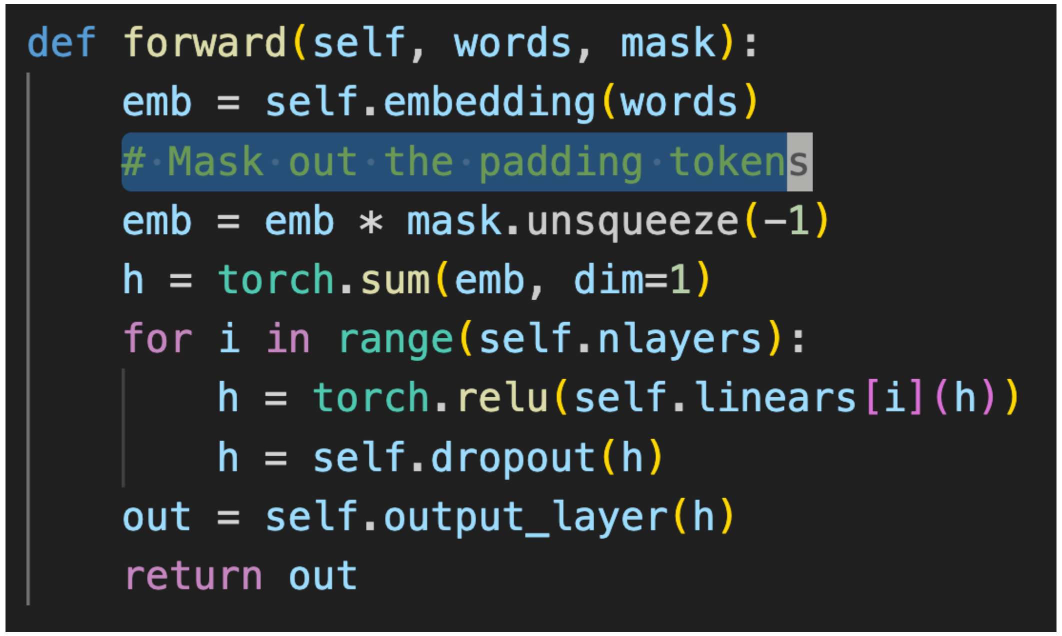
Batching

- When *inputs* are of variable length, we use a *pad token*



```
tensor([[26, 11, 20,  0, 13, 13, 27, 27, 27, 27],
        [26, 18,  7,  0,  8, 13, 27, 27, 27, 27],
        [26, 17, 20, 15,  4, 17, 19, 27, 27, 27],
        [26, 12, 14, 10, 18,  7,  0,  6, 13,  0]])
['[S]', 'l', 'u', 'a', 'n', 'n', '[PAD]', '[PAD]', '[PAD]', '[PAD]']
['[S]', 's', 'h', 'a', 'i', 'n', '[PAD]', '[PAD]', '[PAD]', '[PAD]']
['[S]', 'r', 'u', 'p', 'e', 'r', 't', '[PAD]', '[PAD]', '[PAD]']
['[S]', 'm', 'o', 'k', 's', 'h', 'a', 'g', 'n', 'a']
```

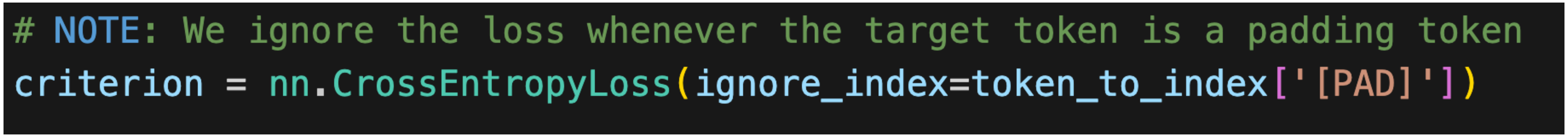
- We may need to *mask out* operations involving pad tokens



```python
def forward(self, words, mask):
    emb = self.embedding(words)
    # Mask out the padding tokens
    emb = emb * mask.unsqueeze(-1)
    h = torch.sum(emb, dim=1)
    for i in range(self.nlayers):
        h = torch.relu(self.linears[i](h))
        h = self.dropout(h)
    out = self.output_layer(h)
    return out
```

Batching

- When *outputs* are of variable length, we mask out the loss for pad tokens



```
# NOTE: We ignore the loss whenever the target token is a padding token
criterion = nn.CrossEntropyLoss(ignore_index=token_to_index['[PAD]'])
```

We'll see a concrete example next class!

Recap: important practical concepts

Recap: important practical concepts

- Dataset splits

Recap: important practical concepts

- Dataset splits
- Overfitting

Recap: important practical concepts

- Dataset splits
- Overfitting
- Weight initialization

Recap: important practical concepts

- Dataset splits
- Overfitting
- Weight initialization
- Optimizer

Recap: important practical concepts

- Dataset splits
- Overfitting
- Weight initialization
- Optimizer
- Learning rate schedules

Recap: important practical concepts

- Dataset splits
- Overfitting
- Weight initialization
- Optimizer
- Learning rate schedules
- Batching

Overall recap

Next 2 lectures

- Recurrent (language) models
- Transformer (language) models

Overall recap

- Language modeling

Next 2 lectures

- Recurrent (language) models
- Transformer (language) models

Overall recap

- Language modeling
- Basic methods: bigram/ngram, feedforward neural

Next 2 lectures

- Recurrent (language) models
- Transformer (language) models

Questions?