

Karpathy -Makemode-1

makemode：通过学习名字数据库后，能根据提供的字符，（基于预测下一个字符出现的可能）自动生成一个新的婴儿名字

仓库：[karpathy/makemode: An autoregressive character-level language model for making more things](https://github.com/karpathy/makemode)

项目工程：

- + README.md
- + makemode.py （模型引擎）
- + names.txt （神经网络训练用，名字数据库，32K）

本质上，makemode是一个基于字符级别（character-level）的语言模型

makemode体现的是：语言模型预测字符串序列下一个词的能力

makemode可以更换选择不同的模型：

Bigram（通过查计数表，用一个字符预测下一个）

MLP

CNN

RNN

LSTM

GRU

Transformer（现代神经网络架构）

python使用的知识点：

splitlines()分词到list结构中

zip迭代器

python字典get方法（统计二元组出现频次）

lambda匿名函数用法

sorted函数默认只对字典的键排序，若根据值排序则使用lambda定义

Pytorch使用的知识点：

默认使用的类型是单精度浮点数float32，如果是记录计数值则需要特定声明int32类型数据

```
In [26]: import torch  
  
In [27]: a = torch.zeros((3, 5))  
a  
  
Out[27]: tensor([[0., 0., 0., 0., 0.],  
                 [0., 0., 0., 0., 0.],  
                 [0., 0., 0., 0., 0.]])
```

```
In [28]: a.dtype  
  
Out[28]: torch.float32
```

```
a = torch.zeros((3, 5), dtype=torch.int32)
a

tensor([[0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0]], dtype=torch.int32)
```

tensor张量允许我们非常高效地操作所有单独的条目

```
a[1,3] = 1
```

反转stoi：将字符表示的（字符间跟随）映射关系，转换为矩阵（张量）中的每个项目值

```
chars = sorted(list(set(''.join(words))))
stoi = {s:i for i,s in enumerate(chars)}
stoi['<S>'] = 26
stoi['<E>'] = 27
stoi
```

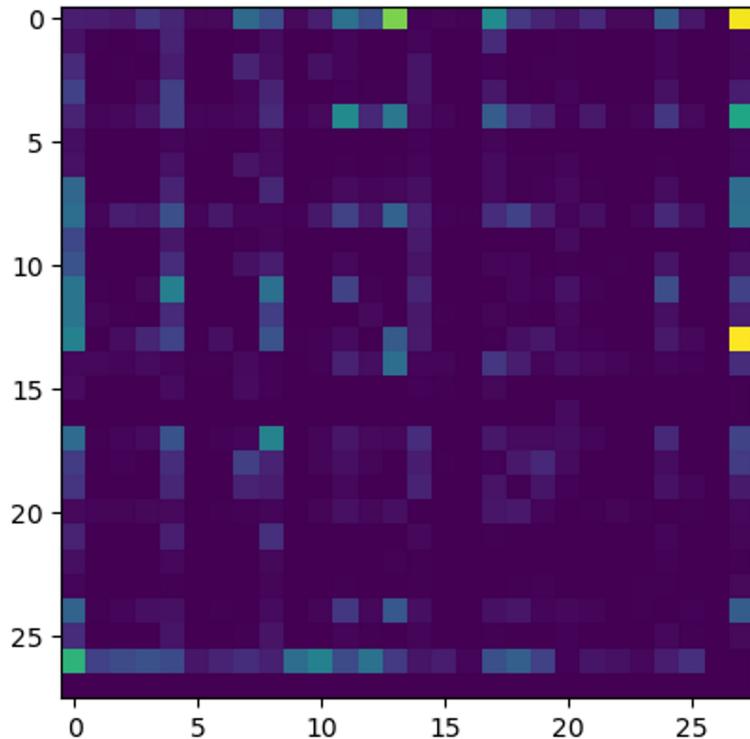
```
{'a': 0,
 'b': 1,
 'c': 2,
 'd': 3,
 'e': 4,
 ... -}
```

美观张量N（统计数值）的表现，用**matplotlib热力图**表示字符跟随关系的数值

```
In [41]: for w in words:
    chs = ['<S>'] + list(w) + ['<E>']
    for ch1, ch2 in zip(chs, chs[1:]):
        ix1 = stoi[ch1]
        ix2 = stoi[ch2]
        N[ix1, ix2] += 1
```

```
In [44]: import matplotlib.pyplot as plt
%matplotlib inline
plt.imshow(N)
```

Out[44]: <matplotlib.image.AxesImage at 0x2173ee6e800>



```
import matplotlib.pyplot as plt
%matplotlib inline

plt.figure(figsize=(16,16))
plt.imshow(N, cmap='Blues')
for i in range(28):
    for j in range(28):
        chstr = itos[i] + itos[j]
        plt.text(j, i, chstr, ha="center", va="bottom", color="gray")
        plt.text(j, i, N[i, j].item(), ha="center", va="top", color="gray")
plt.axis('off');
```

aa	ab	ac	ad	ae	af	ag	ah	ai	aj	ak	al	am	an	ao	ap	aq	ar	as	at	au	av	aw	ax	ay	az	a<5>	a<1>
ba	bb	bc	bd	be	bf	bg	bh	bi	bj	bk	bl	bm	bn	bo	bp	bq	br	bs	bt	bu	bv	bw	bx	by	bz	b<5>	b<1>
ca	cb	cc	cd	ce	cf	cg	ch	ci	cj	ck	cl	cm	cn	co	cp	cq	cr	cs	ct	cu	cv	cw	cx	cy	cz	c<5>	c<1>
da	db	dc	dd	de	df	dg	dh	di	dj	dk	dl	dm	dn	do	dp	dq	dr	ds	dt	du	dv	dw	dx	dy	dz	d<5>	d<1>
ea	eb	ec	ed	ef	eg	eh	ei	ej	ek	el	em	en	eo	ep	eq	er	es	et	eu	ev	ew	ex	ey	ez	e<5>	e<1>	
fa	fb	fc	fd	fe	ff	fg	fh	fi	ff	fk	fl	fm	fn	fo	fp	fq	fr	fs	ft	fu	fv	fw	fx	fy	fz	f<5>	f<1>
ga	gb	gc	gd	ge	gf	gg	gh	gi	gg	gk	gl	gm	gn	go	gp	qq	gr	gs	gt	gu	gv	gw	gx	gy	gz	g<5>	g<1>
ha	hb	hc	hd	he	hf	hg	hh	hi	hh	hk	hl	hm	hn	ho	hp	hq	hr	hs	ht	hu	hv	hw	hx	hy	hz	h<5>	h<1>
ia	ib	ic	id	ie	if	ig	ih	ii	if	ik	il	im	in	io	ip	iq	ir	is	it	iu	iv	iw	ix	iy	iz	i<5>	i<1>
ja	jb	jc	jd	je	jf	jj	jh	ji	jj	jk	jl	jm	jn	jo	jp	jq	jr	js	jt	ju	iv	iw	ix	iy	iz	j<5>	j<1>
ka	kb	kc	kd	ke	kf	kg	kh	ki	kg	kk	kl	km	kn	ko	kp	qq	kr	ks	kt	ku	kv	kw	qx	ky	z	k<5>	k<1>
la	lb	lc	ld	le	lf	lg	lh	li	lf	lk	ll	lm	ln	lo	lp	qq	lr	ls	lt	lu	lv	lw	lx	ly	z	l<5>	l<1>
ma	mb	mc	md	me	mf	mg	mh	mi	mf	mk	ml	mm	mn	mo	mp	qq	mr	ms	mt	mu	mv	mw	qx	ry	z	m<5>	m<1>
na	nb	nc	nd	ne	nf	ng	nh	ni	ng	nk	nl	nm	nn	no	np	qq	nr	ns	nt	nu	nv	nw	qx	ry	z	n<5>	n<1>
oa	ob	oc	od	oe	of	og	oh	oi	of	ok	ol	om	on	oo	op	qq	or	os	ot	ou	ov	ow	ox	oy	z	o<5>	o<1>
ra	rb	rc	rd	re	rf	rg	rh	ri	rf	rk	rl	rm	rn	ro	rp	qq	rr	rs	rt	ru	rv	rw	rx	ry	z	r<5>	r<1>
sa	sb	sc	sd	se	sf	sg	sh	si	sf	sk	sl	sm	sn	so	sp	qq	sr	ss	st	su	sv	sw	tx	ty	z	s<5>	s<1>
ta	tb	tc	td	te	tf	tg	th	ti	tf	tk	tl	tm	tn	to	tp	qq	tr	ts	tt	tu	tv	tw	tx	ty	z	t<5>	t<1>
ua	ub	uc	ud	ue	uf	ug	uh	ui	uf	uk	ul	um	un	uo	up	qq	ur	us	ut	uu	uv	uw	ux	vy	z	u<5>	u<1>
va	vb	vc	vd	ve	vf	vg	vh	vi	vf	vk	vl	vm	vn	vo	vp	qq	vr	vs	vt	vu	vv	vw	vx	yy	z	v<5>	v<1>
xa	xb	xc	xd	xe	xf	gx	hx	ix	fx	jk	ll	mm	nn	oo	pp	qq	rx	ss	tt	uu	vv	ww	xx	yy	z	x<5>	x<1>
ya	yb	yc	yd	ye	ye	yg	yh	iy	ye	jk	ll	mm	nn	oo	pp	qq	rx	ss	tt	uu	vv	ww	xx	yy	z	y<5>	y<1>
za	zb	zc	zd	ze	zf	zg	zh	zi	zf	zk	zl	zm	zn	zo	zp	qq	rz	zs	zt	zu	vu	vw	zx	zy	z	z<5>	z<1>

使用N[i, j].item()的原因是：通过索引N[i, j]获取张量值的时候会发现，这个值仍然是一个张量类型数据。所以你还会误认为这个东西的类型是一个整数！

N[1, 3]

tensor(65, dtype=torch.int32)

只有使用item()时，才会返回一个整数型数值

N[1, 3].item()

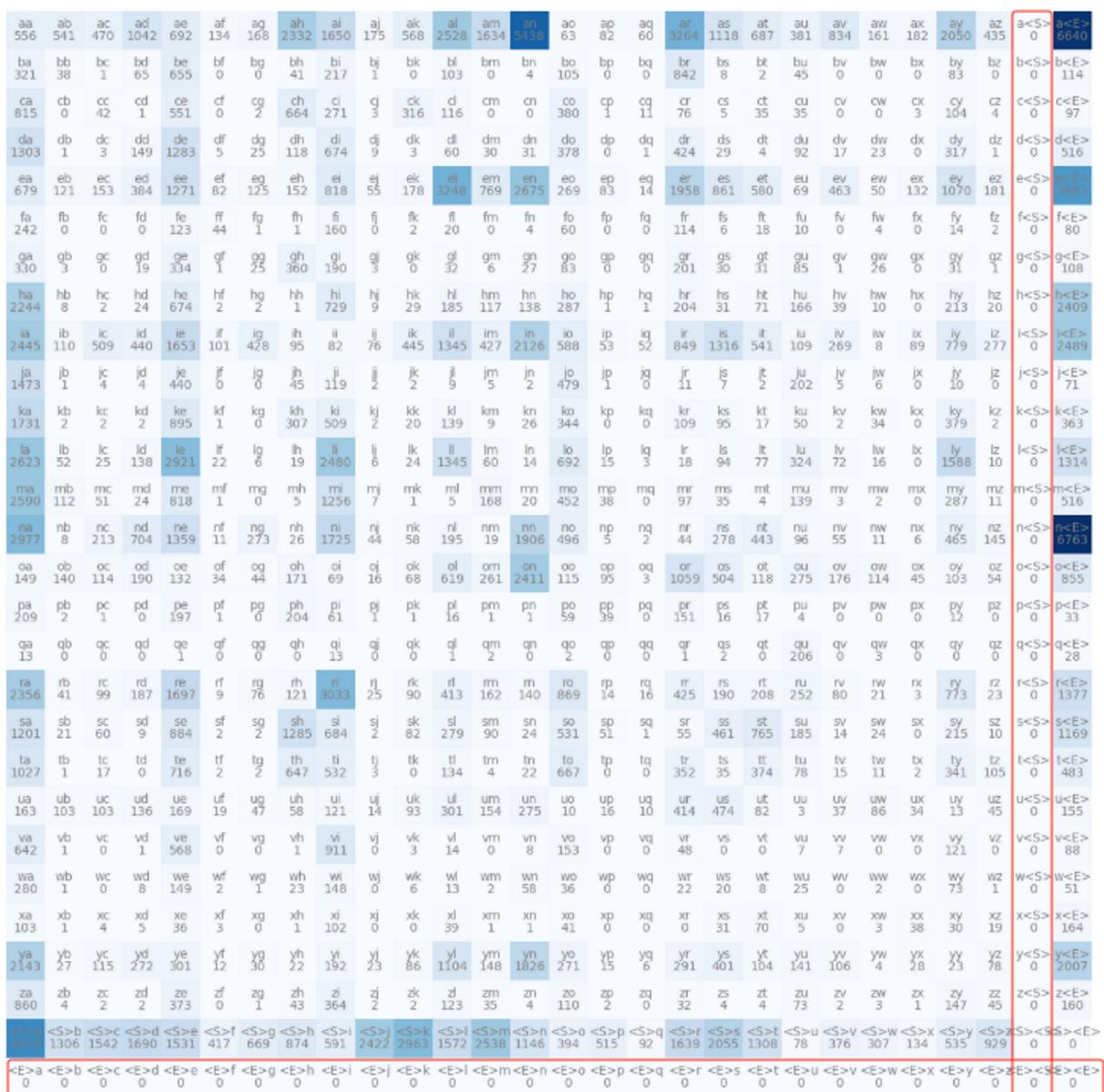
65

观察图中不合理的地方：

1、热力图反映出映射关系有两个废项，值全为0，这是因为不可能存在字符后面跟随<S>，也不可能存在<E>后面跟随字符

改正：不需要用到28x28这么大的张量，减少空间损耗，27合理

2、<E><S>的尖角符号会让图看起来很拥挤



torch.multinomial()

从多项式概率分布中返回样本，（你给我概率，我将根据这些概率分布给你一个整数样本）

```
torch.Generator()
```

```
g = torch.Generator().manual_seed(2147483647)
p = torch.rand(3, generator=g)
p = p/p.sum()
p

tensor([0.6064, 0.3033, 0.0903])
```

```
t = torch.multinomial(p, num_samples=1000, replacement=True, generator=g)
```

```
count = 0
for i in t:
    if i == 0 :
        count+=1
count/1000
```

```
0.602
```

```
count = 0
for i in t:
    if i == 1 :
        count+=1
count/1000
```

```
0.303
```

```
count = 0
for i in t:
    if i == 2 :
        count+=1
count/1000
```

```
0.095
```

multinomial采样函数：

torch.multinomial方法可以根据给定权重对数组进行多次采样，返回采样后的元素下标ix

args:

- input: 权重（必须是torch.Tensor格式），即取每个值的概率，可以是1维或2维，可以不归一化处理
- num_sample: 采样次数
- replacement: 默认值为false，即不放回采样。

rand生成随机数函数：

在深度学习中，随机数是重要的角色。它通常用于初始化神经网络权重、构建数据集、进行数据增强、随机打乱训练数据等。通过使用随机数，可以增加模型的泛化能力，防止过拟合，并提高模型的性能

1、torch.rand()

该函数用于生成具有均匀分布的随机数，范围在0~1之间，接收一个形状参数

(shape) , 返回一个指定形状的张量

二元模型的效果不好：

```
g = torch.Generator().manual_seed(2147483647)

for i in range(20):
    out = []
    ix = 0
    while True:
        p = N[ix].float()
        p = p/p.sum()

        #使p为值1的27个元素，目的是均匀分布各元素可能性相等
        #p = torch.ones(27) / 27.0

        ix = torch.multinomial(p, num_samples=1, replacement=True, generator=g)
        out.append(itos[ix])
        if ix == 0:
            break
    print(''.join(out))
```

```
mor.
axx.
minaymoryles.
kondlaisah.
anchshizarie.
odaren.
iaddash.
h.
jhinatien.
egushl.
h.
br.
a.
jayn.
ilemannariaenien.
be.
f.
akiinela.
trttanakeroruceyaaxatona.
lamoynayrkiedengin.
```

均匀分布概率的话效果更差：

```

g = torch.Generator().manual_seed(2147483647)

for i in range(20):
    out = []
    ix = 0
    while True:
        #p = N[ix].float()
        #p = p/p.sum()

        #使p为值1的27个元素，目的是均匀分布各元素可能性相等
        p = torch.ones(27) / 27.0

        ix = torch.multinomial(p, num_samples=1, replacement=True, generator=g)
        out.append(itos[ix])
        if ix == 0:
            break
    print(''.join(out))

```

qvsayxbqrqmyqwuznivanukotdjvdhd.
qnoymtzduqkatdetkpfjdgiqvlejfkrsqlwnirghhwlu.
idcx.
cekmzucjnjoeofvjvrggqrjr.
cfbhhabkslpokc.
xtxbpmpknusuxdgzfexhwqpldpdnwzvkyxsqjforqqpxstwkfoufhvwfhmsuyyotcvvvqpfcb djco
uhkajkhqnnpqmmllaordqy.
gszpw.
zlgijinangzzuulsyvqrufuawavsdbnwvlmrypvgrsfgps hgnmwafqmsjdvhngvoiigxhkwdlrdkw
nagzyknqv.
lfstdqigvncdoidetsukgd p.
cfpjxseqjcsmjwguzes.
woflfjxflylgbegpjdpovdtw.
dlzysqtrbxhcdneium.
xtyslfbmaboanyjpojuujflcsaucqcg t jmlzqtbaisvxrtgupkppigxudejdzsroqeigovuxmvt.
jlxfolkozci.
tkhdivkdifaxcevlpk tkwwvuxlymtwylgpzauwdvxfvboofl dphmjeomjgjcqeqwt.
.
wlcclcjbm.
quuyijtnzmycshclormjyrerqslomdrlbuwqn lmitbrmqhtbdwbyv lsmwnborwcdhjotezw nsxuvffv
inrmedelubhdfgtavxqfgmnyqrygyevxaapbjtnwfnewq xerd yttvfo.
iauarz.
tynoqkyp.

`torch.sum()`

若对输入张量的某一维度求和，一共2种方法

输入一个张量

1、`torch.sum(input, dtype=None)`

2、`torch.sum(input, list: dim, bool: keepdim=False, dtype=None)` 返回一个张量

```
P = N.float()  
#仅对行求和。压缩0维度(纵向压缩)：  
#P.sum(0, keepdim=True)  
#P.sum(0, keepdim=True).shape  
P = P/P.sum(0, keepdim=True)
```

★ torch的广播机制 (broadcasting) :

是PyTorch的一种自动扩展张量形状的机制

“广播”用于描述如何在形状不一的数组上应用算术运算，背后是：通过在执行元素级操作时自动扩展张量的形状

在满足特定限制前提下，较小的数组“广播至”较大的数组，使两者形状互相兼容

广播提供了一个向量化数组操作的机制，这个机制发生在C层面（non python），速度更快

torch中“广播”操作可直接使用，无需额外的显式调用expand、repeat，减少冗余代码

减少内存消耗：广播机制通过虚拟扩展（不实际复制数据）实现形状匹配，避免创建大量中间张量，节省内存；底层优化使广播操作比显式循环或复制更高效，尤其适合大规模张量运算（批量处理图像或自然语言数据）

深入理解广播机制（否则代码很容易出现bug）：

torch.tensor和torch.Tensor的区别？

```
xs = torch.Tensor(xs)  
ys = torch.tensor(ys)  
  
. e  
e m  
m m  
m a  
a .
```

这个训练集中有五个独立的输入例子供神经网络学习，当ix=0我们希望很高概率输出5；当ix=5我们希望很高概率输出13；当ix=13我们希望很高概率输出13或者1

```
In [108]: xs.dtype  
Out[108]: torch.float32
```

```
In [109]: ys.dtype  
Out[109]: torch.int64
```

torch.nn.functional.one_hot

torch.randn()

用于生成一个具有**正态分布**的随机数，均值=0，标准差=1，接收一个张量形状参数，填充张量并返回

@是Pytorch特有的“矩阵乘法运算符”

单个神经元: (5, 27) @ (27, 1) = (5, 1)即生成五个激活值

```
W = torch.randn((27, 1))
```

```
xenc @ W
```

- ★ (5,1) 是因为将5个输入同时馈送到同一个神经元，并且PyTorch并行地、独立地计算了5个 $W \times x$ (没有偏置bias)，Andrej说上面只是一个神经元 (可理解为一个单层，层的形状为 (27,1) 目的是解码。每层负责评估是否更可能跟随某一字符)，所以我们希望拥有27个神经元 (27层)，并行地为这个词向量评估所有的跟随情况

```
W = torch.randn((27, 27))
```

当W是27x27时，这将并行地评估所有的27个神经元在所有5个输入上的情况。这将给我们一个更好、大得多的结果。

```
# (5, 27) @ (27, 27) -> (5, 27)
```

解释这27个输出是什么：对于每一个输入样本 (etc, .emma)，我们试图在这里产生的是为序列中的下一个字符生成某种概率分布，即假设：?=27个字符中更有可能跟随的那个，!=概率值

```
prob(., ?)=!, prob(e, ?)=!, prob(m, ?)=!, prob(a, ?)=!
```

等价操作：

```
xs
```

```
tensor([ 0,  5, 13, 13,  1])
```

```
ys
```

```
tensor([ 5, 13, 13,  1,  0])
```

```
# randomly initialize 27 neurons' weights. each neuron receives 27 inputs
g = torch.Generator().manual_seed(2147483647)
W = torch.randn((27, 27), generator=g)
```

```
# forward pass:
xenc = F.one_hot(xs, num_classes=27).float() # input to the network: one-hot
logits = xenc @ W # predict log-counts
counts = logits.exp() # counts, equivalent to N
probs = counts / counts.sum(1, keepdim=True) # probabilities for next character
```

```
probs.shape
```

```
torch.Size([5, 27])
```

```
probs[0, 5], probs[1, 13], probs[2, 13], probs[3, 1], probs[4, 0]
(tensor(0.0123),
 tensor(0.0181),
 tensor(0.0267),
 tensor(0.0737),
 tensor(0.0150))
```

```
torch.arange(5)
```

```
tensor([0, 1, 2, 3, 4])
```

```
probs[torch.arange(5), ys]
```

```
tensor([0.0123, 0.0181, 0.0267, 0.0737, 0.0150])
```

```

# gradient descent
for k in range(150):

    # forward pass:
    xenc = F.one_hot(xs, num_classes=27).float()
    logits = xenc @ W
    counts = logits.exp()
    probs = counts / counts.sum(1, keepdim=True)
    loss = -probs[torch.arange(num), ys].log().mean()
    print(loss.item())

    # backward pass
    W.grad = None
    loss.backward()

    # update
    W.data += -50 * W.grad

```

这里的权重 (W) 实际上和下面的表格是同一东西，但是W是log count，而不是计数本身。准确地说W进行指数运算，即W.exp()，才是这个表格数组。基于梯度的框架中，我们开始时随机初始化w，然后让损失引导我们得到完全相同的表格数组。所以这个表格数组基本上就是优化计数后的权重张量 (W)，两者最终得到的损失值是一样的

ö	ö	ö	b	b	c	d	d	e	f	g	h	j	j	k	k	m	n	p	q	r	s	t	ü	v	w	x	y	z					
a	aa	556	541	ab	ac	1042	1542	1690	1531	e	417	669	874	591	2422	2363	1572	2538	1146	394	515	92	1639	2055	1308	78	376	307	134	535	929		
ä	ä	6640	321	bb	bc	1	bd	be	bf	bg	bh	bi	bj	bl	bk	bl	bm	bn	bo	ao	ap	aq	ar	as	at	au	av	aw	ax	ay	az	az	435
b	114	ba	38	bb	bc	1	bd	be	bf	bg	bh	bi	bj	bl	bk	bl	bm	bn	bo	bp	bo	br	bs	bt	bu	bv	bw	bx	by	bz	0		
c	97	ca	0	cb	cc	1	cd	ce	cf	cg	ch	di	ej	ek	el	em	en	eo	ep	eq	er	es	et	eu	ev	ew	ex	ey	ez	cz	4		
d	516	815	1303	db	dc	1	dd	de	df	dg	dh	di	dj	dk	dl	dm	dn	do	dp	dr	ds	dt	du	dv	dw	dx	dy	dz	1				
e	679	ea	121	eb	ec	153	ed	ee	ef	eg	eh	ei	ej	ek	el	em	en	eo	ep	eq	er	es	et	eu	ev	ew	ex	ey	ez	1			
f	80	242	fb	fc	0	fd	fe	ff	fg	fh	fi	fi	fo	fk	fl	fm	fn	fo	fp	fr	fs	ft	fu	fv	fw	fx	fy	fz	2				
g	108	gg	350	gb	gc	0	gd	ge	gf	gg	gh	gi	gj	gk	gl	gm	gn	go	gp	gr	gt	gu	gv	gw	gx	gy	gz	1					
h	2409	ha	8	hb	hc	2	hd	he	hf	hg	hh	hi	hj	hk	hl	hm	hn	ho	hp	hq	hr	hs	ht	hu	hv	hw	hx	hy	hz	20			
i	2489	ia	110	ib	ic	509	id	je	if	ig	ih	ii	is	it	ik	il	im	ip	iq	ir	is	it	iu	iv	iw	ix	iy	iz	277				
j	71	ij	1473	jb	jc	4	jd	je	jf	jo	jh	ji	jj	jk	jl	jm	jp	jq	jr	jt	ju	iv	iv	iw	ix	iy	iz	0					
k	363	ka	1731	kb	kc	2	kd	ke	kf	kg	kh	ki	kj	kk	kl	km	kn	ko	kp	kr	ks	kt	ku	kv	kw	kx	ky	kz	2				
l	1314	la	2623	lb	lc	25	ld	le	lf	lg	lh	li	li	lk	ll	lm	ln	lo	lp	lr	ls	lt	lu	lv	lw	lx	ly	lz	10				
m	516	mm	2590	mb	mc	112	md	me	mf	mg	mh	mi	mj	mk	ml	mm	mn	mo	mp	mq	mr	ms	mt	mu	mv	mw	mx	my	mz	11			
n	6763	na	2977	nb	nc	8	nd	ne	nf	ng	nh	ni	nj	nk	nl	nm	nn	no	np	nr	ns	nt	nu	nv	nw	nx	ny	nz	145				
o	855	149	aa	140	ab	114	ac	109	ad	ae	af	ag	ah	oi	oj	ok	ol	om	on	op	or	os	ot	ou	ov	ow	ox	oy	oz	54			
p	33	pa	209	pb	pc	1	pd	pe	pf	pg	ph	pi	pj	pk	pl	pm	pn	po	pp	pr	ps	pt	pu	pv	pw	px	py	pz	0				
q	28	qa	13	qb	qc	0	qd	qe	qf	qg	qh	qi	qj	qk	ql	qm	qn	qo	qp	qr	qs	qt	qu	qv	qw	qx	qy	qz	0				
r	1377	ra	2356	rb	rc	41	rd	re	rf	rg	rh	ri	rn	rk	rl	rm	ro	rp	rr	rt	rv	rw	rv	rt	rs	ry	rz	23					
s	1169	sa	1201	sb	sc	21	sd	se	sf	sg	sh	si	sj	sk	sl	sm	sn	so	sp	sq	sr	ss	st	su	sv	sw	sx	sy	sz	10			
t	483	ta	1027	tb	tc	17	td	te	tf	tg	th	ti	tz	tk	tl	tm	tn	to	tp	ta	tr	ts	tt	tu	tv	tw	tx	ty	tz	105			
u	155	ua	163	ub	uc	103	ud	ue	uf	ug	uh	ui	uj	uk	ul	um	un	uo	up	ut	uv	uw	ux	uy	uz	145	45						
v	88	va	642	vb	vc	0	vd	ve	vf	vg	vh	vi	vj	vk	vl	vm	vn	vo	vp	vr	vs	vt	vu	vw	vx	vy	vz	0					
w	51	wa	280	wb	wc	1	wd	we	wf	wg	wh	wi	wj	wk	wl	wm	wn	wo	wp	wr	ws	wt	ww	wx	wy	wz	1						
x	164	xa	103	xb	xc	4	xd	xe	xf	xf	xg	xh	xi	xj	xk	xl	xm	xn	xo	xp	xq	xr	xs	xt	xu	xv	xw	xz	19				
y	2007	ya	2143	yb	yc	7	yd	ye	yf	yg	yh	yi	yz	yk	yl	ym	yn	yo	yr	ys	yt	yu	yt	ys	yt	yz	23	yz	78				
z	160	za	860	zb	zc	4	zd	ze	zf	zg	zh	zi	zj	zk	zl	zm	zn	zo	zp	zr	zs	zt	zu	zw	zx	zy	zz	45					

```

chars = sorted(list(set(''.join(words))))
stoi = {s:i+1 for i,s in enumerate(chars)}
stoi['.'] = 0
itos = {i:s for s,i in stoi.items()}

```

字典推导式 {s:i+1}

索引映射逻辑：将字符 s 作为字典的键，索引 i 加1后的值作为对应的值（目的是使编号从1开

始而非0)

避免零值：常见于需要非零索引的场景（如数据预处理中0通常用于填充或无效标记）

python切片操作 (slice)

语法定义：

`a[start:end:step]`

- start - 切片起始索引（包含该位置）
- end - 切片结束索引（不包含该位置，左闭右开）
- step - 步长（默认为1）

当参数省略：

`a[1:]` - 省略end，表示从索引1到末尾

`a[:1]` - 省略start，表示从开头到索引1（不包含1）

torch矩阵的索引

```
C = torch.randn((27, 2))
```

```
C[5]
```

```
tensor([-0.8173,  0.6349])
```

```
C[[5,6,7]]
```

```
tensor([[ -0.8173,   0.6349],
       [  2.2825,   1.0506],
       [ -1.0324,  -0.5456]])
```

```
C[torch.tensor([5,6,7])]
```

```
tensor([[ -0.8173,   0.6349],
       [  2.2825,   1.0506],
       [ -1.0324,  -0.5456]])
```

★ torch创建的矩阵索引用法很多，如这里的`C[5]`就是“请求（单个）5号元素”，还可以使用列表索引，甚至还可以使用一个整数的张量（多维的张量也可以）

```
C[5]
```

```
tensor([-0.0594,  0.5394])
```

```
F.one_hot(torch.tensor(5), num_classes=27).float() @ C
```

```
tensor([-0.0594,  0.5394])
```

we have一个独热向量乘以C的列。由于所有的0，它们实际上最后最终屏蔽了C中除了第五行之外所有内容，