

Building a Autoregressive Neural Network

Part 1

Luca WB

2026-01-28

Table of contents

0.1	Brief summary	2
0.2	Setup	2
0.3	Creating a baseline	2

0.1 Brief summary

In this post, we will implement an Autoregressive Neural Network from scratch, relying solely on the PyTorch tensor class. We assume prior familiarity with Neural Networks; however, if your knowledge feels a bit rusty or you need a refresher, I recommend reading this post beforehand [Building Neural Networks from Scratch](#).

The main reason for this is to learn how an Autoregressive NN works to generate words, for this, I'm drawing on Andrej Karpathy's video series about [makemore](#), a network capable of creating more words of the same type, so if you train with names, it generates more proper names it generates more words that remember proper names, and so on with anything that is formed by letters.

In this post, I will cover how to make a simple model for our baseline, and how to implement a model with MLP and compare them.

0.2 Setup

First, you need to download PyTorch and the dataset. For PyTorch, just download in the official site <https://pytorch.org/get-started/locally/>. Now, for the dataset, you can create your own with random names that you can think, but It's much easier just download the names.txt dataset from the Andrej repository <https://github.com/karpathy/makemore/blob/master/names.txt>.

0.3 Creating a baseline

In propose of this, it's just to create the most simple and naive model. It's important because we need some baseline to compare with our future models, so we will create a model called bigram, the logic is just to look to the last character. Note that you will use just one character of context for our model, and we will consider that the most small part

of our word is a character, for models like chatGPT, they don't use characters, they use combinations of characters similar to syllables.

So, to start, we need first import our dataset and PyTorch

```
1 import torch
2
3 # Basically makes a list of all the names
4 names = open("names.txt", "r").read().splitlines()
5 names[:5]
```

```
['emma', 'olivia', 'ava', 'isabella', 'sophia']
```

Most part of models usually can't handle with characters, so it's useful to convert this letters in numbers in some way. For this, there are many possibles, but I will use just a simple dictionary to convert them. But we

```
1 chars = sorted(list(set("".join(names)))) # Creates an ordered list with all letters in ou
2 charToInt = {s:i+1 for i,s in enumerate(chars)} # Creates a dict to convert chars to int,
3 charToInt["."] = 0 # I will explain later why we need a special character
4 print(charToInt)
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8, 'i': 9, 'j': 10, 'k': 11,
```

Just to get it ready, if we convert to int, so we can read it at the end, we will need an intToChar converter, so let's get it ready

```
1 intToChar = {s:i for i,s in charToInt.items()}
2 print(intToChar)
```

```
{1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e', 6: 'f', 7: 'g', 8: 'h', 9: 'i', 10: 'j', 11: 'k',
```

Know, for our model, we need to calculate the total number that each sequence occurs, like, with we start with letter "a", how many times occurs that "m" is the next character. And it's for this that we need and special characters, because we always need something to start, after all, the autoregressive model logic and take the output of the model and put it in its input, so we need an initial input. In our case, we will use "." as the symbol to start a name/words and to stop word (without a final symbol, it would generate forever). To make more clear, see the code bellow

```

1 N = torch.zeros((27,27)).int()
2
3 for name in names:
4     chars = ["."] + list(name) + ["."] # turn the name in a list of characters and add "."
5     for ch1,ch2 in zip(chars, chars[1:]): # In each loop, pick up one letter in ch1, and t
6         id1, id2 = charToInt[ch1], charToInt[ch2]
7         N[id1, id2] += 1

```

Basically, this count how often some sequence of characters occurs, like the most common letter sequence is “n” follow by “.”, this mean, that the most commum letter to finish a name in our dataset it’s “n”. If you run with all the names, you can use the code bellow to find the most common occurrences

```

1 id1, id2 = (N == N.max()).nonzero(as_tuple=True) # Creates a boolean matrix that only it's
2 print(intToChar[id1.item()], "-->", intToChar[id2.item()], "occurs ", N.max().item())

```

```
n --> . occurs 6763
```

So let’s see how our bigrams are distributed

```

1 import matplotlib.pyplot as plt
2
3 plt.figure(figsize= (16,16))
4 plt.imshow(N, cmap="Blues")
5 for i in range(27):
6     for j in range(27):
7         chstr = intToChar[i] + intToChar[j]
8         plt.text(j,i, chstr, ha="center", va="bottom", color="gray")
9         plt.text(j,i, N[i,j].item(), ha="center", va="top", color="gray")
10
11 plt.axis("off")

```

ö	4640	1306	1542	1690	1531	417	669	874	591	2422	2963	1572	2538	1146	394	515	92	1639	2055	1308	78	376	307	134	535	929
a640	aa56	ab541	ac470	ad1042	ae692	af134	ag168	ah2332	ai1650	aj175	ak568	al2528	am1634	an6438	ao63	ap82	aq60	ar3264	as1118	at687	au381	av834	aw161	ax182	ay2050	az435
b114	ba321	bb38	bc1	bd65	be655	bf0	bg0	bh41	bi217	bj1	bk0	bl103	bm0	bn4	bo105	bp0	bq0	br842	bs8	bt2	bu45	bv0	bw0	bx0	by83	bz0
c97	ca815	cb0	cc42	cd1	ce551	cf0	cg2	ch664	ci271	cj3	ck316	cl116	cm0	cn0	co380	cp1	cq11	cr76	cs5	ct35	cu35	cv0	cw0	cx3	cy104	cz4
d516	da1303	db1	dc3	dd149	de1283	df5	dg25	dh118	di674	dj9	dk3	dl60	dm30	dn31	do378	dp0	dq1	dr424	ds29	dt4	du92	dv17	dw23	dx0	dy317	dz1
e993	ea679	eb121	ec153	ed384	ee1271	ef82	eg125	eh152	ei818	ej55	ek178	el3248	em769	en2675	eo269	ep83	eq14	er1958	es861	et580	eu69	ev463	ew50	ex132	ey1070	ez181
f80	fa242	fb0	fc0	fd0	fe123	ff44	fg1	fh1	fi160	fj0	fk2	fl20	fm0	fn4	fo60	fp0	fq0	fr114	fs6	ft18	fu10	fv0	fw4	fx0	fy14	fz2
g108	ga330	gb3	gc0	gd19	ge334	gf1	gg25	gh360	gi190	gj3	gk0	gl32	gm6	gn27	go83	gp0	gq0	gr201	gs30	gt31	gu85	gv1	gw26	gx0	gy31	gz1
h2409	ha2244	hb8	hc2	hd24	he674	hf2	hg2	hh1	hi729	hj9	hk29	hl185	hm117	hn138	ho287	hp1	hq1	hr204	hs31	ht71	hu166	hv39	hw10	hx0	hy213	hz20
i2489	ia2445	ib110	ic509	id440	ie1653	if101	ig428	ih95	ii82	ij76	ik445	il1345	im427	in2126	io588	ip53	iq52	ir849	is1316	it541	iu109	iv269	iw8	ix89	iy779	iz277
j71	ja1473	jb1	jc4	jd4	je440	jf0	ig0	ih45	ii119	ij2	jk2	jl9	jm5	jn2	jo479	jp1	jq0	jr11	js7	jt2	ju202	jv5	jw6	jx0	jy10	jz0
k363	ka1731	kb2	kc2	kd2	ke895	kf1	kg0	kh307	ki509	kj2	kk20	kl139	km9	kn26	ko344	kp0	kq0	kr109	ks95	kt17	ku50	kv2	kw34	kx0	ky379	kz2
l1314	la2623	lb52	lc25	ld138	le2921	lf22	lg6	lh19	li2480	lj6	lk24	ll1345	lm60	ln14	lo692	lp15	lq3	lr18	ls94	lt77	lu324	lv72	lw16	lx0	ly1588	lz10
m516	ma2590	mb112	mc51	md24	me818	mf1	mg0	mh5	mi1256	mj7	mk1	ml5	mm168	mn20	mo452	mp38	mq0	mr97	ms35	mt4	mu139	mv3	mw2	mx0	my287	mz11
n6763	na2977	nb8	nc13	nd704	ne1359	nf11	ng273	nh26	ni1725	nj58	nk195	nl195	nm19	nn1906	no496	np5	nq2	nr44	ns278	nt443	nu96	nv55	nw11	nx6	ny465	nz145
o855	oa149	ob140	oc114	od190	oe132	of34	og44	oh171	oi69	oj16	ok68	ol619	om261	on2411	oo115	op95	oq3	or1059	os504	ot118	ou275	ov176	ow114	ox45	oy103	oz54
p33	pa209	pb2	pc1	pd0	pe197	pf1	pg0	ph204	pi61	pj1	pk1	pl16	pm1	pn1	po59	pp39	pq0	pr151	ps16	pt17	pu4	pv0	pw0	px0	py12	pz0
q28	qa13	qb0	qc0	qd0	qe1	qf0	qg0	qh0	qi13	qj0	qk0	ql1	qm2	qn0	qo2	qp0	qq0	qr1	qs2	qt0	qu206	qv0	qw3	qx0	qy0	qz0
r1377	ra2356	rb41	rc99	rd187	re1697	rf9	rg76	rh121	ri3033	rj25	rk90	rl413	rm162	rn140	ro869	rp14	rq16	rr425	rs190	rt208	ru252	rv80	rw21	rx3	ry773	rz23
s1169	sa1201	sb21	sc60	sd9	se884	sf2	sg2	sh1285	si684	sj2	sk82	sl279	sm90	sn24	so531	sp51	sq1	sr55	ss461	st765	su185	sv14	sw24	sx0	sy215	sz10
t483	ta1027	tb1	tc17	td0	te716	tf2	tg2	th647	ti532	tj3	tk0	tl134	tm4	tn22	to667	tp0	tq0	tr352	ts35	tt374	tu78	tv15	tw11	tx2	ty341	tz105
u155	ua163	ub103	uc103	ud136	ue169	uf19	ug47	uh58	ui121	uj14	uk93	ul301	um154	un275	uo10	up16	uq10	ur414	us474	ut82	uu3	uv37	uw86	ux34	uy13	uz45
v88	va642	vb1	vc0	vd1	ve568	vf0	vg0	vh1	vi911	vj0	vk3	vl14	vm0	vn8	vo153	vp0	vq0	vr48	vs0	vt0	vu7	vv7	vw0	vx0	vy121	vz0
w51	wa280	wb1	wc0	wd8	we149	wf2	wg1	wh23	wi148	wj0	wk6	wl3	wm2	wn58	wo36	wp0	wq0	wr22	ws20	wt8	wu25	wv0	ww2	wx0	wy73	wz1
x164	xa103	xb1	xc4	xd5	xe36	xf3	xg0	xh1	xi102	xj0	xk0	xl31	xm1	xn1	xo41	xp0	xq0	xr0	xs31	xt70	xu5	xv0	xw3	xx38	xy30	xz19
y2007	ya2143	yb27	yc115	yd272	ye301	yf12	yg30	yh22	yi192	yj23	yk86	yl1104	ym148	yn1826	yo271	yp15	yq6	yr291	ys401	yt104	yu141	yv106	yw4	yx28	yy23	yz78
z160	za860	zb4	zc2	zd2	ze373	zf0	zg1	zh43	zi364	zj2	zk2	zl123	zm35	zn4	zo110	zp2	zq0	zr32	zs4	zt4	zu73	zv2	zw3	zx1	zy147	zz45

One thing very interesting you can note, it's that have many combinations that don't exist, like "bk" or "gc". This makes it impossible for our model to generate a name with this combination, it is ok to leave it like this, but it would be a good practice to add 1 in all values, thus ensuring that at least there is the minimal possibility of generating a rare sequence

$$N = N + 1$$

i	6641	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	557	542	471	1043	693	135	169	2333	1651	176	569	2529	1635	439	64	83	61	3265	1119	688	382	835	162	183	2051	436	
b	115	322	39	2	66	656	1	1	42	218	2	1	104	1	5	106	1	1	843	9	3	46	1	1	1	1	
c	98	816	cb	cc	cd	ce	cf	cg	ch	ci	cj	ck	cl	cm	cn	co	cp	cq	cr	cs	ct	cu	cv	cw	cx	cy	
d	517	1304	db	dc	dd	de	df	dg	dh	di	dj	dk	dl	dm	dn	do	dp	dq	dr	ds	dt	du	dv	dw	dx	dz	
e	3984	ea	eb	ec	ed	ee	ef	eg	eh	ei	ej	ek	el	em	en	eo	ep	eq	er	es	et	eu	ev	ew	ex	ey	
f	81	243	fb	fc	fd	fe	ff	fg	fh	fi	fj	fk	fl	fm	fn	fo	fp	fq	fr	fs	ft	fu	fv	fw	fx	fy	
g	109	ga	gb	gc	gd	ge	gf	gg	gh	gi	gj	gk	gl	gm	gn	go	gp	gq	gr	gs	gt	gu	gv	gw	gx	gy	
h	2410	ha	hb	hc	hd	he	hf	hg	hh	hi	hj	hk	hl	hm	hn	ho	hp	hq	hr	hs	ht	hu	hv	hw	hx	hy	
i	2490	ia	ib	ic	id	ie	if	ig	ih	ii	ij	ik	il	im	in	io	ip	iq	ir	is	it	iu	iv	iw	ix	iz	
j	72	ja	jb	jc	jd	je	jf	ig	ih	ii	ij	ik	jl	jm	jn	jo	jp	jq	jr	js	jt	ju	jv	jw	jx	jz	
k	364	ka	kb	kc	kd	ke	kf	kg	kh	ki	kj	kk	kl	km	kn	ko	kp	kq	kr	ks	kt	ku	kv	kw	kx	ky	
l	1315	la	lb	lc	ld	le	lf	lg	lh	li	lj	lk	ll	lm	ln	lo	lp	lq	lr	ls	lt	lu	lv	lw	lx	ly	
m	517	ma	mb	mc	md	me	mf	mg	mh	mi	mj	mk	ml	mm	mn	mo	mp	mq	mr	ms	mt	mu	mv	mw	mx	my	
n	6764	na	nb	nc	nd	ne	nf	ng	nh	ni	nj	nk	nl	nm	nn	no	np	nq	nr	ns	nt	nu	nv	nw	nx	ny	
o	856	oa	ob	oc	od	oe	of	og	oh	oi	oj	ok	ol	om	on	oo	op	oq	or	os	ot	ou	ov	ow	ox	oy	
p	34	pa	pb	pc	pd	pe	pf	pg	ph	pi	pj	pk	pl	pm	pn	po	pp	pq	pr	ps	pt	pu	pv	pw	px	py	
q	29	qa	qb	qc	qd	qe	qf	qg	qh	qi	qj	qk	ql	qm	qn	qo	qp	qq	qr	qs	qt	qu	qv	qw	qx	qy	
r	1378	ra	rb	rc	rd	re	rf	rg	rh	ri	rj	rk	rl	rm	rn	ro	rp	rq	rr	rs	rt	ru	rv	rw	rx	ry	
s	1170	sa	sb	sc	sd	se	sf	sg	sh	si	sj	sk	sl	sm	sn	so	sp	sq	sr	ss	st	su	sv	sw	sx	sy	
t	484	ta	tb	tc	td	te	tf	tg	th	ti	tj	tk	tl	tm	tn	to	tp	tq	tr	ts	tt	tu	tv	tw	tx	ty	
u	156	ua	ub	uc	ud	ue	uf	ug	uh	ui	uj	uk	ul	um	un	uo	up	uq	ur	us	ut	uu	uv	uw	ux	uy	
v	89	va	vb	vc	vd	ve	vf	vg	vh	vi	vj	vk	vl	vm	vn	vo	vp	vq	vr	vs	vt	vu	vv	vw	vx	vy	
w	52	wa	wb	wc	wd	we	wf	wg	wh	wi	wj	wk	wl	wm	wn	wo	wp	wq	wr	ws	wt	wu	wv	ww	wx	wy	
x	165	xa	xb	xc	xd	xe	xf	xg	xh	xi	xj	xk	xl	xm	xn	xo	xp	xq	xr	xs	xt	xu	xv	xw	xx	xy	
y	2008	ya	yb	yc	yd	ye	yf	yg	yh	yi	yj	yk	yl	ym	yn	yo	yp	yq	yr	ys	yt	yu	yv	yw	yx	yz	
z	161	za	zb	zc	zd	ze	zf	zg	zh	zi	zj	zk	zl	zm	zn	zo	zp	zq	zr	zs	zt	zu	zv	zw	zx	zy	

So, lets transform our probability matrix

```

1 P = N
2 P = P / P.sum(dim=1, keepdims=True)

```

```

1 import matplotlib.pyplot as plt
2
3 plt.figure(figsize= (16,16))
4 plt.imshow(P, cmap="Blues")
5 for i in range(27):
6     for j in range(27):
7         chstr = intToChar[i] + intToChar[j]
8         plt.text(j,i, chstr, ha="center", va="bottom", color="gray")

```

```

9         plt.text(j,i, f"{P[i,j].item():.3f}", ha="center", va="top", color="gray")
10
11 plt.axis("off")

```

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	
a	0.138	0.041	0.048	0.053	0.048	0.013	0.021	0.027	0.018	0.076	0.092	0.149	0.079	0.036	0.012	0.016	0.003	0.051	0.064	0.041	0.002	0.012	0.010	0.004	0.017	0.029	
b	0.016	0.016	0.016	0.014	0.031	0.020	0.004	0.005	0.069	0.049	0.005	0.017	0.075	0.048	0.160	0.002	0.002	0.002	0.096	0.033	0.020	0.011	0.025	0.005	0.005	0.060	0.013
c	0.121	0.015	0.001	0.025	0.246	0.000	0.000	0.016	0.082	0.001	0.000	0.039	0.000	0.002	0.040	0.000	0.000	0.315	0.003	0.001	0.017	0.000	0.000	0.000	0.031	0.000	
d	0.229	0.000	0.012	0.001	0.155	0.000	0.001	0.187	0.076	0.001	0.089	0.033	0.000	0.000	0.107	0.001	0.003	0.022	0.002	0.010	0.010	0.000	0.000	0.001	0.030	0.001	
e	0.236	0.000	0.001	0.027	0.232	0.001	0.005	0.022	0.122	0.002	0.001	0.011	0.006	0.006	0.069	0.000	0.000	0.077	0.005	0.001	0.017	0.003	0.004	0.000	0.058	0.000	
f	0.033	0.006	0.008	0.019	0.062	0.004	0.006	0.007	0.040	0.003	0.009	0.159	0.038	0.131	0.013	0.004	0.001	0.096	0.042	0.028	0.003	0.023	0.002	0.007	0.052	0.009	
g	0.261	0.001	0.001	0.001	0.133	0.048	0.002	0.002	0.173	0.001	0.003	0.023	0.001	0.005	0.065	0.001	0.001	0.123	0.008	0.020	0.012	0.001	0.005	0.001	0.016	0.003	
h	0.169	0.002	0.001	0.010	0.171	0.001	0.013	0.185	0.098	0.002	0.001	0.017	0.004	0.014	0.043	0.001	0.001	0.103	0.016	0.016	0.044	0.001	0.014	0.001	0.016	0.001	
i	0.294	0.001	0.000	0.003	0.088	0.000	0.000	0.000	0.096	0.001	0.004	0.024	0.015	0.018	0.038	0.000	0.000	0.027	0.004	0.009	0.022	0.005	0.001	0.000	0.028	0.003	
j	0.138	0.006	0.029	0.025	0.093	0.006	0.024	0.005	0.004	0.025	0.017	0.004	0.012	0.033	0.003	0.003	0.048	0.074	0.031	0.006	0.015	0.001	0.005	0.044	0.016		
k	0.304	0.001	0.002	0.002	0.151	0.000	0.000	0.016	0.041	0.001	0.001	0.003	0.002	0.001	0.164	0.001	0.000	0.004	0.003	0.001	0.069	0.002	0.002	0.000	0.004	0.000	
l	0.342	0.001	0.001	0.001	0.177	0.000	0.000	0.061	0.101	0.001	0.004	0.028	0.002	0.005	0.068	0.000	0.000	0.022	0.019	0.004	0.010	0.001	0.007	0.000	0.075	0.001	
m	0.188	0.004	0.002	0.010	0.209	0.002	0.001	0.177	0.001	0.002	0.002	0.096	0.004	0.001	0.050	0.001	0.001	0.007	0.006	0.023	0.005	0.001	0.000	0.000	0.114	0.001	
n	0.388	0.017	0.008	0.004	0.123	0.000	0.000	0.001	0.188	0.001	0.000	0.001	0.025	0.003	0.068	0.006	0.000	0.015	0.005	0.001	0.021	0.001	0.000	0.000	0.043	0.002	
o	0.162	0.000	0.012	0.038	0.074	0.001	0.001	0.001	0.094	0.002	0.003	0.011	0.001	0.104	0.027	0.000	0.000	0.002	0.015	0.024	0.005	0.003	0.001	0.000	0.025	0.008	
p	0.019	0.018	0.014	0.024	0.017	0.004	0.006	0.022	0.009	0.002	0.009	0.078	0.033	0.303	0.015	0.012	0.001	0.133	0.063	0.015	0.035	0.022	0.014	0.006	0.013	0.007	
q	0.199	0.003	0.002	0.001	0.188	0.002	0.001	0.195	0.059	0.002	0.002	0.016	0.002	0.002	0.057	0.038	0.001	0.144	0.016	0.017	0.005	0.001	0.001	0.001	0.012	0.001	
r	0.047	0.003	0.003	0.003	0.007	0.003	0.003	0.003	0.047	0.003	0.003	0.007	0.010	0.003	0.010	0.003	0.003	0.007	0.010	0.003	0.692	0.003	0.013	0.003	0.003	0.003	
s	0.185	0.003	0.008	0.015	0.133	0.001	0.006	0.010	0.238	0.002	0.007	0.033	0.013	0.011	0.068	0.001	0.001	0.033	0.015	0.016	0.020	0.006	0.002	0.000	0.061	0.002	
t	0.148	0.003	0.008	0.001	0.109	0.000	0.000	0.158	0.084	0.000	0.010	0.034	0.011	0.003	0.065	0.006	0.007	0.057	0.094	0.023	0.002	0.003	0.000	0.027	0.001	0.001	
u	0.184	0.000	0.003	0.000	0.128	0.001	0.001	0.116	0.095	0.001	0.000	0.024	0.001	0.004	0.119	0.000	0.000	0.063	0.006	0.067	0.014	0.003	0.002	0.001	0.061	0.019	
v	0.052	0.033	0.033	0.043	0.054	0.006	0.015	0.019	0.039	0.005	0.030	0.096	0.049	0.087	0.003	0.005	0.003	0.131	0.150	0.026	0.001	0.012	0.028	0.011	0.004	0.015	
w	0.247	0.001	0.000	0.001	0.219	0.000	0.000	0.001	0.351	0.000	0.002	0.006	0.000	0.003	0.059	0.000	0.000	0.019	0.000	0.000	0.003	0.003	0.000	0.000	0.047	0.000	
x	0.294	0.002	0.001	0.009	0.157	0.003	0.002	0.025	0.156	0.001	0.007	0.015	0.003	0.062	0.039	0.001	0.001	0.024	0.022	0.009	0.027	0.001	0.003	0.001	0.077	0.002	
y	0.144	0.003	0.007	0.008	0.051	0.006	0.001	0.003	0.142	0.001	0.001	0.055	0.003	0.003	0.058	0.001	0.001	0.001	0.044	0.098	0.008	0.001	0.006	0.054	0.043	0.028	
z	0.219	0.003	0.012	0.028	0.031	0.001	0.003	0.002	0.020	0.002	0.009	0.113	0.015	0.186	0.028	0.002	0.001	0.030	0.041	0.011	0.014	0.011	0.001	0.003	0.002	0.008	
	0.066	0.355	0.002	0.001	0.154	0.000	0.001	0.018	0.151	0.001	0.001	0.051	0.015	0.002	0.046	0.001	0.000	0.014	0.002	0.002	0.031	0.001	0.002	0.001	0.061	0.019	

Some probabilities stay in 0 because the visualization it's limited to 3 decimal numbers. Now we already have our model, it's just our probability matrix P, bellow I will show how to use it.

```

1 import random
2
3 for i in range(10):
4     out = []

```

```
5     init = 0
6     while True:
7         id = torch.multinomial(P[init], num_samples=1, replacement=True).item()
8
9         if id == 0:
10             break
11
12         out.append(intToChar[id])
13         init = id
14     print("".join(out))
```

ladhai
ken
ile
chiliariali
jamitt
janany
h
sekal
trlelerilynemi
llsdrgaabr