

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	iy10	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	413	1307	1543	1691	1532	418	670	875	592	2423	2964	1573	2539	1147	395	516	93	r	1640	s	2056	t	1309	u	79	v	377	w	308	x	135	y	536	z	930
a.6641	aa557	ab542	ac471	ad1043	ae693	af135	ag169	ah2333	ai1651	aj176	ak569	al2529	am1635	an8439	ao64	ap83	aq61	ar3265	as1119	at688	au382	av835	aw162	ax183	ay2051	az436									
b.115	ba322	bb39	bc2	bd66	be656	bf1	bg1	bh42	bi218	bj2	bk1	bl104	bm1	bn5	bo106	bp1	bq1	br843	bs9	bt3	bu46	bv1	bw1	bx1	by84	bz1									
c.98	ca816	cb1	cc43	cd2	ce552	cf1	cg3	ch665	ci272	cj4	ck317	cl117	cm1	cn1	co381	cp2	cq12	cr77	cs6	ct36	cu36	cv1	cw1	cx4	cy105	cz5									
d.517	da1304	db2	dc4	dd150	de1284	df6	dg26	dh119	di675	dj10	dk4	dl61	dm31	dn32	do379	dp1	dq2	dr425	ds30	dt5	du93	dv18	dw24	dx1	dy318	dz2									
e.3984	ea680	eb122	ec154	ed385	ee1272	ef83	eg126	eh153	ei819	ej56	ek179	el3249	em770	en2676	eo270	ep84	eq15	er1959	es862	et581	eu70	ev464	ew51	ex133	ey1071	ez182									
f.81	fa243	fb1	fc1	fd1	fe124	ff45	fg2	fh2	fi161	fj1	fk3	fl21	fm1	fn5	fo61	fp1	fq1	fr115	fs7	ft19	fu11	fv1	fw5	fx1	fy15	fz3									
g.109	ga331	gb4	gc1	gd20	ge335	gf2	gg26	gh361	gi191	gj4	gk1	gl33	gm7	gn28	go84	gp1	gq1	gr202	gs31	gt32	gu86	gv2	gw27	gx1	gy32	gz2									
h.2410	ha2245	hb9	hc3	hd25	he675	hf3	hg3	hh2	hi730	hj10	hk30	hl186	hm118	hn139	ho288	hp2	hq2	hr205	hs32	ht72	hu167	hv40	hw11	hx1	hy214	hz21									
i.2490	ia2446	ib111	ic510	id441	ie1654	if102	ig429	ih96	ii83	ij77	ik446	il1346	im428	in2127	io589	ip54	iq53	ir850	is1317	it542	iu110	iv270	iw9	ix90	iy780	iz278									
j.72	ja1474	jb2	jc5	jd5	je441	jf1	ig1	jh46	ji120	jj3	jk3	jl10	jm6	jn3	jo480	jp2	jq1	jr12	js8	jt3	ju203	jv6	jw7	jx1	jy11	jz1									
k.364	ka1732	kb3	kc3	kd3	ke896	kf2	kg1	kh308	ki510	kj3	kk21	kl140	km10	kn27	ko345	kp1	kq1	kr110	ks96	kt18	ku51	kv3	kw35	kx1	ky380	kz3									
l.1315	la2624	lb53	lc26	ld139	le2922	lf23	lg7	lh20	li2481	lj7	lk25	ll1346	lm61	ln15	lo693	lp16	lq4	lr19	ls95	lt78	lu325	lv73	lw17	lx1	ly1589	lz11									
m.517	ma2591	mb113	mc52	md25	me819	mf2	mg1	mh6	mi1257	mj8	mk2	ml6	mm169	mn21	mo453	mp39	mq1	mr98	ms36	mt5	mu140	mv4	mw3	mx1	my288	mz12									
n.6764	na2978	nb9	nc214	nd705	ne1360	nf12	ng274	nh27	ni1726	nj45	nk59	nl196	nm20	nn1907	no497	np6	nq3	nr45	ns279	nt444	nu97	nv56	nw12	nx7	ny466	nz146									
o.856	oa150	ob141	oc115	od191	oe133	of35	og45	oh172	oi70	oj17	ok69	ol620	om262	on2412	oo116	op96	oq4	or1060	os505	ot119	ou276	ov177	ow115	ox46	oy104	oz55									
p.34	pa210	pb3	pc2	pd1	pe198	pf2	pg1	ph205	pi62	pj2	pk2	pl17	pm2	pn2	po60	pp40	pq1	pr152	ps17	pt18	pu5	pv1	pw1	px1	py13	pz1									
q.29	qa14	qb1	qc1	qd1	qe2	qf1	qg1	qh1	qi14	qj1	qk1	ql2	qm3	qn1	qo3	qp1	qq1	qr2	qs3	qt1	qu207	qv1	qw4	qx1	qy1	qz1									
r.1378	ra2357	rb42	rc100	rd188	re1698	rf10	rg77	rh122	ri3034	rj26	rk91	rl414	rm163	rn141	ro870	rp15	rq17	rr426	rs191	rt209	ru253	rv81	rw22	rx4	ry774	rz24									
s.1170	sa1202	sb22	sc61	sd10	se885	sf3	sg3	sh1286	si685	sj3	sk83	sl280	sm91	sn25	so532	sp52	sq2	sr56	ss462	st766	su186	sv15	sw25	sx1	sy216	sz11									
t.484	ta1028	tb2	tc18	td1	te717	tf3	tg3	th648	ti533	tj4	tk1	tl135	tm5	tn23	to668	tp1	tq1	tr353	ts36	tt375	tu79	tv16	tw12	tx3	ty342	tz106									
u.156	ua164	ub104	uc104	ud137	ue170	uf20	ug48	uh59	ui122	uj15	uk94	ul302	um155	un276	uo11	up17	uq11	ur415	us475	ut83	uu4	uv38	uw87	ux35	uy14	uz46									
v.89	va643	vb2	vc1	vd2	ve569	vf1	vg1	vh2	vi912	vj1	vk4	vl15	vm1	vn9	vo154	vp1	vq1	vr49	vs1	vt1	vu8	vv8	vw1	vx1	vy122	vz1									
w.52	wa281	wb2	wc1	wd9	we150	wf3	wg2	wh24	wi149	wj1	wk7	wl14	wm3	wn59	wo37	wp1	wq1	wr23	ws21	wt9	wu26	wv1	ww3	wx1	wy74	wz2									
x.165	xa104	xb2	xc5	xd6	xe37	xf4	xg1	xh2	xi103	xj1	xk1	xl40	xm2	xn2	xo42	xp1	xq1	xr1	xs32	xt71	xu6	xv1	xw4	xx39	xy31	xz20									
y.2008	ya2144	yb28	yc116	yd273	ye302	yf13	yg31	yh23	yi193	yj24	yk87	yl1105	ym149	yn1827	yo272	yp16	yq7	yr292	ys402	yt105	yu142	yv107	yw5	yx29	yy24	yz79									
z.161	za861	zb5	zc3	zd3	ze374	zf1	zg2	zh44	zi365	zj3	zk3	zl124	zm36	zn5	zo111	zp3	zq1	zr33	zs5	zt5	zu74	zv3	zw4	zx2	zy148	zz46									

So, lets transform our probability matrix

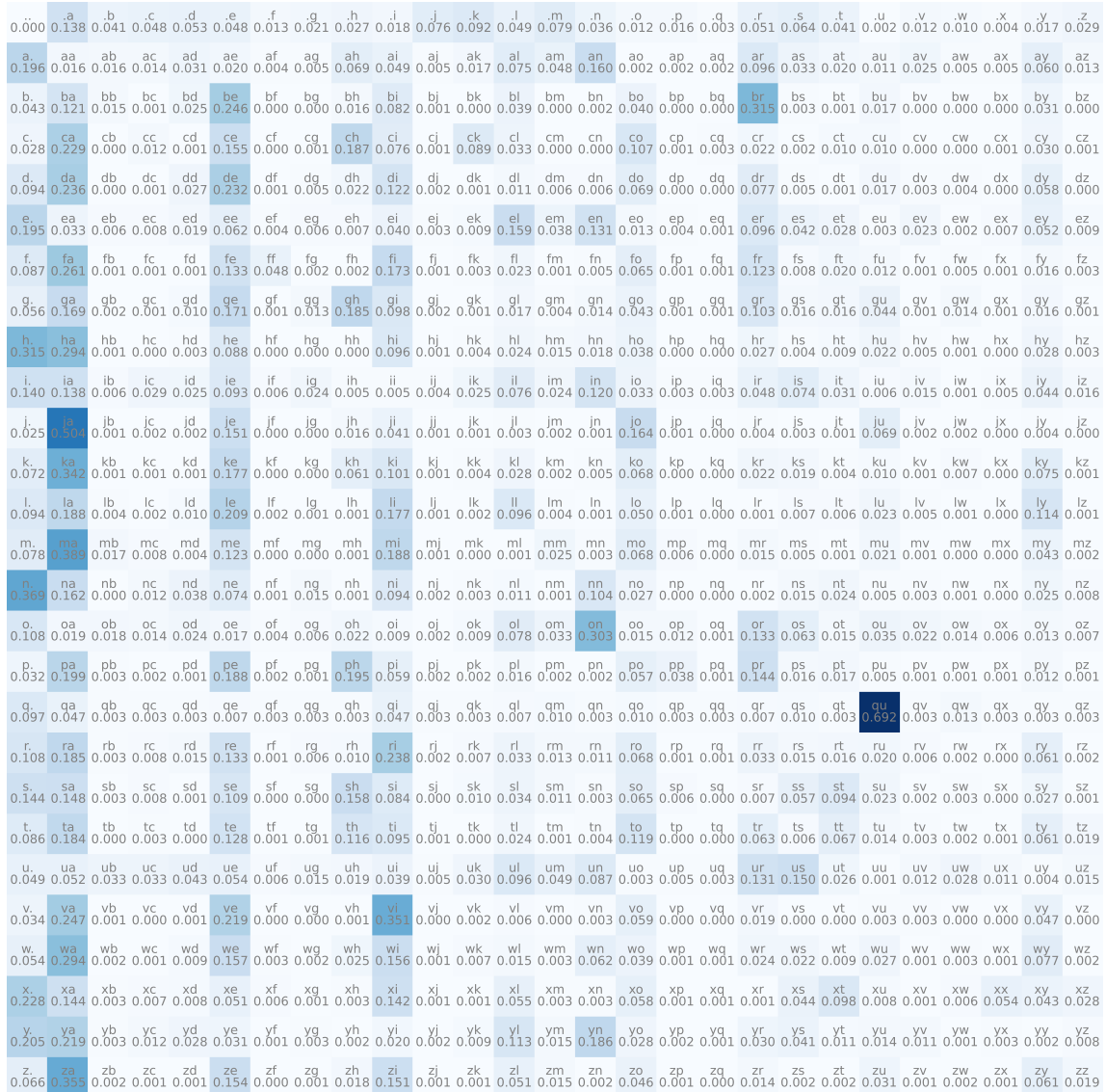
```
1 P = N
2 P = P / P.sum(dim=1, keepdims=True)
```

```
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")

```



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

```