

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', '.'}
```

Now, 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 common 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	ö	ö	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	ü	ü	v	w	x	y	z
6640	aa	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	
97	ba	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	
c	ca	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	
d	da	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	
ea	ea	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	ez	
fa	fa	fb	fc	fd	fe	ff	fg	fh	fi	fo	fk	fl	fm	fn	fo	fp	fq	fr	fs	ft	fu	fv	fw	fx	fy	fz		
g	ga	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	gz	
h	ha	ha	hc	hd	he	hf	hg	hh	hi	hj	hk	hl	hm	hn	ho	hp	hq	hr	hs	ht	hu	hv	hw	hx	hy	hz		
i	ia	ia	ib	ic	id	ie	if	ig	ih	ii	ij	ik	il	im	in	io	ip	iq	ir	is	it	iu	iv	iw	ix	iy	iz	
j	ja	ja	jb	jc	jd	je	jf	ji	jh	jj	jk	jl	jm	jn	jo	jp	jq	jr	js	jt	ju	ju	jw	jx	jy	jz		
k	ka	ka	kb	kc	kd	ke	kf	kg	kh	ki	kj	kk	kl	km	kn	ko	kp	qq	kr	ks	kt	ku	kv	kw	ky	kj	rz	
l	la	la	lb	lc	ld	le	lf	lg	lh	li	lk	ll	lm	ln	lo	lp	lg	lr	ls	lt	lu	lv	lw	lx	ly	lz		
m	ma	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	mz	
n	na	na	nb	nc	nd	ne	nf	ng	nh	ni	nj	nk	nl	nn	no	np	nr	ns	nt	nu	nv	nw	nx	ny	nz			
o	oa	oa	ob	oc	od	oe	of	og	oh	oi	oj	ok	ol	om	on	oo	op	or	os	ot	ou	ov	ow	ox	oy	oz		
p	pa	pa	pb	pc	pd	pe	pf	pg	ph	pi	pj	pk	pl	pm	pn	po	pp	pr	ps	pt	pu	pv	pw	px	py	pz		
q	qa	qa	qb	qc	qd	qe	qf	qg	qh	qi	qk	ql	qm	qn	qo	qp	qq	qr	qs	qt	qu	qv	qw	qx	qy	qz		
r	ra	ra	rb	rc	rd	re	rf	rg	rh	ri	rl	rk	rl	rm	rd	ro	rp	rq	rr	rs	rt	ru	rv	rw	rx	ry	rz	
s	sa	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	sz	
t	ta	ta	tb	tc	td	te	tf	tg	th	ti	tj	tk	tl	tm	tn	to	tp	tr	ts	tt	tu	tv	tw	tx	ty	tz		
u	ua	ua	ub	uc	ud	ue	uf	ug	uh	ui	uj	uk	ul	um	un	uo	up	ur	ua	us	ut	uu	uv	uw	ux	uy	uz	
v	va	vb	vc	vd	ve	vf	vg	vh	vi	vj	vk	vl	vm	vn	vo	vp	vr	vs	vt	vu	vv	vw	vx	vy	vz			
w	wa	wb	wc	wd	we	wf	wg	wh	wi	wj	wk	wl	wm	wn	wo	wp	wq	wr	ws	wt	wu	ww	wx	wy	wz			
x	xa	xa	xb	xc	xd	xe	xf	xg	xh	xi	xj	xk	xl	xm	xn	xo	xp	xq	xr	xs	xt	xu	xv	xw	xz			
y	ya	yb	yc	yd	ye	yf	yg	yh	yi	yj	yk	yl	ym	yn	yo	yp	yq	yr	ys	yt	yu	yy	yz	yz				
z	za	zb	zc	zd	ze	zf	zg	zh	zi	zj	zk	zl	zm	zn	zo	zp	zr	zs	zt	zu	zv	zw	zx	zy	zz			

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

1 N = N + 1

i	a	b	c	d	e	f	g	h	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z			
6641	4411	1307	1543	1691	1532	418	670	875	592	2423	2964	1573	2539	1147	395	516	93	1640	2056	1309	79	377	308	135	536	930		
a	aa	ab	ac	ad	ae	af	ag	ah	ai	ak	al	am	an	ao	ap	aq	ar	as	at	au	av	aw	ax	ay	az	436		
115	322	39	2	bd	66	be	656	bf	1	bg	42	bi	218	bj	2	bk	104	bm	1	bn	5	bo	106	bp	1	br	843	265
c	ca	cb	cc	cd	ce	cf	cg	ch	ci	ck	cl	cm	cn	co	cp	cq	cr	cs	ct	cu	cv	cw	cx	cy	cz	5		
d	da	db	dc	dd	de	df	dg	dh	di	dk	dl	dm	dn	do	dp	dq	dr	ds	dt	du	dv	dw	dx	dy	dz	2		
e	ea	eb	ec	ed	ee	ef	eg	eh	ei	ek	el	em	en	eo	ep	eq	er	es	et	eu	ev	ew	ex	ev	ez	182		
f	fa	fb	fc	fd	fe	ff	fg	fh	fi	fk	fl	fm	fn	fo	fp	fq	fr	fs	ft	fu	fv	fw	fx	fy	fz	3		
g	ga	gb	gc	gd	ge	gf	gg	gh	gi	gk	gl	gm	gn	go	gp	gq	gr	gs	gt	gu	gv	gw	qx	gy	gz	2		
h	ha	hb	hc	hd	he	hf	hg	hh	hi	hk	hl	hm	hn	ho	hp	hq	hr	hs	ht	hu	hv	hw	hx	hy	hz	21		
i	ia	ib	ic	id	ie	if	ig	ih	ii	ik	il	im	in	io	ip	iq	ir	is	it	iu	iv	iw	ix	iy	iz	278		
j	ja	jb	jc	jd	je	jf	ji	jh	jj	jk	jl	jm	jn	jo	jp	jq	jr	js	jt	ju	qv	ju	ix	iy	iz	1		
k	ka	kb	kc	kd	ke	kf	kg	kh	ki	kj	kk	kl	km	kn	ko	kp	qq	kr	ks	kt	ku	kv	kw	ky	380	3		
l	la	lb	lc	ld	le	lf	lg	lh	li	lk	lm	ln	lo	lp	lq	lr	ls	lt	lu	lv	lw	lx	ly	lz	11			
m	ma	mb	mc	md	me	mf	mg	mh	mi	mk	ml	mm	mn	mo	mp	mq	mr	ms	mt	mu	mv	mw	mx	my	mz	12		
n	na	nb	nc	nd	ne	nf	ng	nh	ni	nj	nk	nl	nm	nn	no	np	nr	ns	nt	nu	nv	nw	nx	ny	nz	146		
o	oa	ob	oc	od	oe	of	og	oh	oi	ok	ol	om	on	oo	op	oq	or	os	ot	ou	ov	ow	ox	oy	oz	55		
p	pa	pb	pc	pd	pe	pf	pg	ph	pi	pj	pk	pl	pm	pn	po	pp	pr	ps	pt	pu	pv	pw	px	py	pz	1		
q	qa	qb	qc	qd	qe	qf	qg	qh	qi	qk	ql	qm	qn	qo	qp	qq	qr	qs	qt	qu	qv	qw	qx	qy	qz	1		
r	ra	rb	rc	rd	re	rf	rg	rh	ri	rk	rl	rm	rn	ro	rp	rq	rr	rs	rt	ru	rv	rw	rx	ry	rz	24		
s	sa	sb	sc	sd	se	sf	sg	sh	si	sk	sl	sm	sn	so	sp	sq	sr	ss	st	su	sv	sw	sx	sy	sz	11		
t	ta	tb	tc	td	te	tf	tg	th	ti	tk	tl	tm	tn	to	tp	tz	tr	ts	tt	tu	tv	tw	tx	ty	tz	106		
u	ua	ub	uc	ud	ue	uf	ug	uh	ui	uk	ul	um	un	uo	up	uq	ur	us	ut	uu	uv	uw	ux	uy	uz	46		
v	va	vb	vc	vd	ve	vf	vg	vh	vi	vk	vl	vm	vn	vo	vp	vr	vs	vt	vu	vv	vw	vx	vy	vz	1			
w	wa	wb	wc	wd	we	wf	wg	wh	wi	wj	wk	wl	wm	wn	wo	wp	wq	wr	ws	wt	wu	ww	wx	wy	wz	2		
x	xa	xb	xc	xd	xe	xf	xg	xh	xi	xj	xk	xl	xm	xn	xo	xp	xq	xr	xz	xt	xu	xv	xw	xx	xy	xz	20	
y	ya	yb	yc	yd	ye	yf	yg	yh	yi	yk	yl	ym	yn	yo	yp	yq	yr	ys	yt	yu	yy	yz	24	yz	29			
z	za	zb	zc	zd	ze	zf	zg	zh	zi	zk	zl	zm	zn	zo	zp	zq	zr	zs	zt	zu	zv	zw	zx	zy	zz	46		

So, lets transform our probability matrix

```

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

```

```

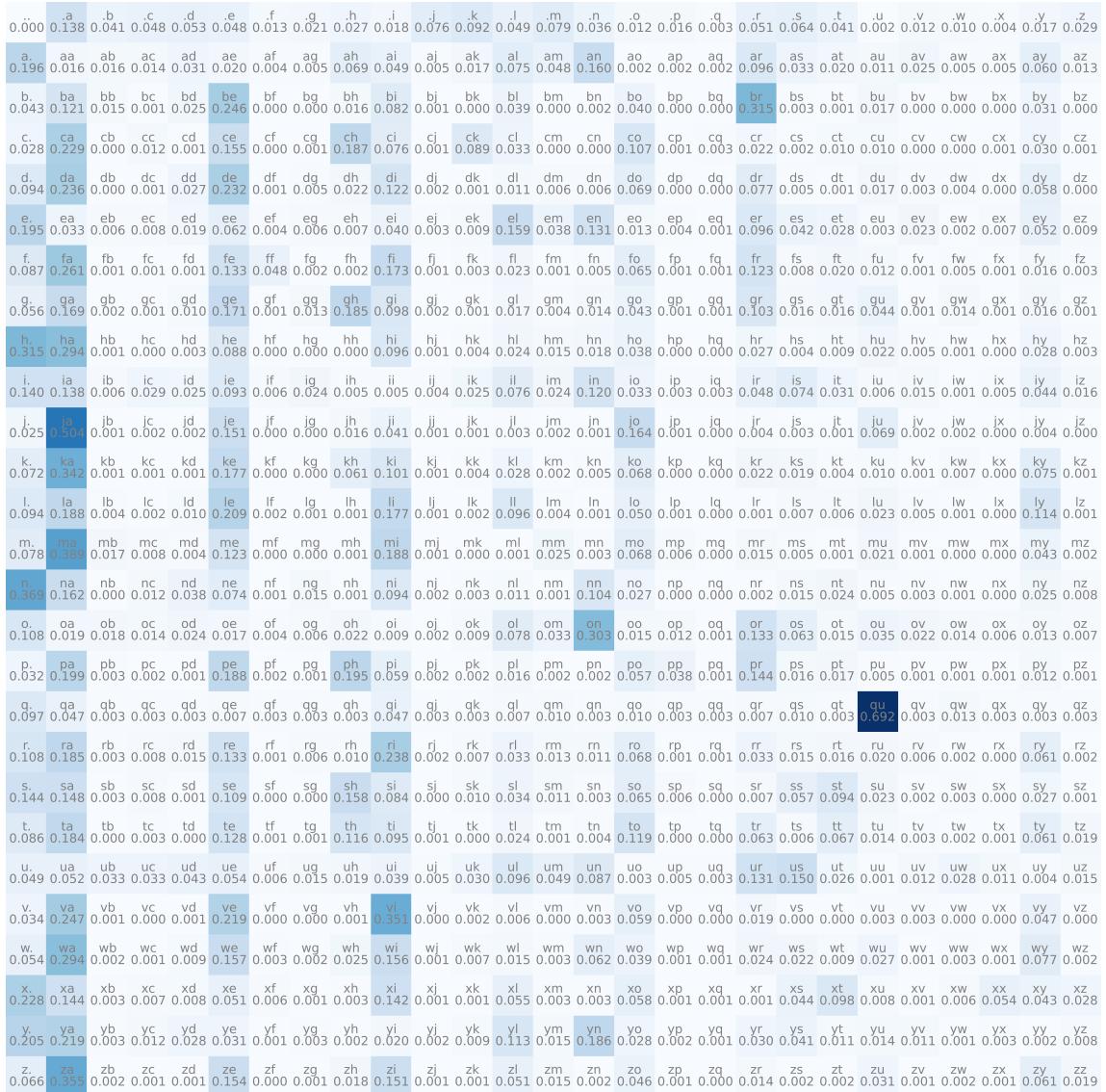
1 import matplotlib.pyplot as plt
2
3 plt.figure(figsize= (16,16))
4 plt.imshow(P, cmap="Blues")
5 for i in range(27):
6     for j in range(27):
7         chstr = intToChar[i] + intToChar[j]
8         plt.text(j,i, chstr, ha="center", va="bottom", color="gray")

```

```

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