Lecture 4 Language Modeling

Fei Tan

@econdojo @BusinessSchool101

Introduction to Neural Networks

June 11, 2025

Makemore

- Autoregressive character-level language model
  - input: a text file of words
  - output: similar words as input
- Application to name generation

```
# Input: 32K common names from ssa.gov in 2018
$ python makemore.py -i names.txt -o names

# Output: name-like words
dontell
khylum
camatena
aeriline
najlah
sherrith
```

The Road Ahead...

1. Bigram
2. Multilayer Perceptron
3. Transformer

Bigram Counts

.. 0	.a 4410	.b 1306	.c 1542	.d 1690	.e 1531	.f 417	.g 669	.h 874	.i 591	.j 2422	.k 2963	.l 1572	.m 2538	.n 1146	.o 394	.p 515	.q 92	.r 1639	.s 2055	.t 1308	.u 78	.v 376	.w 307	.x 134	.y 535	.z 929
a. 6640	aa 556	ab 541	ac 470	ad 1042	ae 692	af 134	ag 168	ah 2332	ai 1650	aj 175	ak 568	al 2528	am 1634	an 5438	ao 63	ap 82	aq 60	ar 3264	as 1118	at 687	au 381	av 834	aw 161	ax 182	ay 2050	az 435
b. 114	ba 321	bb 38	bc 1	bd 65	be 655	bf 0	bg 0	bh 41	bi 217	bj 1	bk 0	bl 103	bm 0	bn 4	bo 105	bp 0	bq 0	br 842	bs 8	bt 2	bu 45	bv 0	bw 0	bx 0	by 83	bz 0
c. 97	ca 815	cb 0	cc 42	cd 1	ce 551	cf 0	cg 2	ch 664	ci 271	cj 3	ck 316	cl 116	cm 0	cn 0	co 380	cp 1	cq 11	cr 76	cs 5	ct 35	cu 35	cv 0	cw 0	cx 3	cy 104	cz 4
d. 516	da 1303	db 1	dc 3	dd 149	de 1283	df 5	dg 25	dh 118	di 674	dj 9	dk 3	dl 60	dm 30	dn 31	do 378	dp 0	dq 1	dr 424	ds 29	dt 4	du 92	dv 17	dw 23	dx 0	dy 317	dz 1
e. 3983	ea 679	eb 121	ec 153	ed 384	ee 1271	ef 82	eg 125	eh 152	ei 818	ej 55	ek 178	el 3248	em 769	en 2675	eo 269	ep 83	eq 14	er 1958	es 861	et 580	eu 69	ev 463	ew 50	ex 132	ey 1070	ez 181
f. 80	fa 242	fb 0	fc 0	fd 0	fe 123	ff 44	fg 1	fh 1	fi 160	fj 0	fk 2	fl 20	fm 0	fn 4	fo 60	fp 0	fq 0	fr 114	fs 6	ft 18	fu 10	fv 0	fw 4	fx 0	fy 14	fz 2
g. 108	ga 330	gb 3	gc 0	gd 19	ge 334	gf 1	gg 25	gh 360	gi 190	gj 3	gk 0	gl 32	gm 6	gn 27	go 83	gp 0	gq 0	gr 201	gs 30	gt 31	gu 85	gv 1	gw 26	gx 0	gy 31	gz 1
h. 2409	ha 2244	hb 8	hc 2	hd 24	he 674	hf 2	hg 2	hh 1	hi 729	hj 9	hk 29	hl 185	hm 117	hn 138	ho 287	hp 1	hq 1	hr 204	hs 31	ht 71	hu 166	hv 39	hw 10	hx 0	hy 213	hz 20
i. 2489	ia 2445	ib 110	ic 509	id 440	ie 1653	if 101	ig 428	ih 95	ii 82	ij 76	ik 445	il 1345	im 427	in 2126	io 588	ip 53	iq 52	ir 849	is 1316	it 541	iu 109	iv 269	iw 8	ix 89	iy 779	iz 277
j. 71	ja 1473	jb 1	jc 4	jd 4	je 440	jf 0	jg 0	jh 45	ji 119	jj 2	jk 2	jl 9	jm 5	jn 2	jo 479	jp 1	jq 0	jr 11	js 7	jt 2	ju 202	jv 5	jw 6	jx 0	jy 10	jz 0
k. 363	ka 1731	kb 2	kc 2	kd 2	ke 895	kf 1	kg 0	kh 307	ki 509	kj 2	kk 20	kl 139	km 9	kn 26	ko 344	kp 0	kq 0	kr 109	ks 95	kt 17	ku 50	kv 2	kw 34	kx 0	ky 379	kz 2
l. 1314	la 2623	lb 52	lc 25	ld 138	le 2921	lf 22	lg 6	lh 19	li 2480	lj 6	lk 24	ll 1345	lm 60	ln 14	lo 692	lp 15	lq 3	lr 18	ls 94	lt 77	lu 324	lv 72	lw 16	lx 0	ly 1588	lz 10
m. 516	ma 2590	mb 112	mc 51	md 24	me 818	mf 1	mg 0	mh 5	mi 1256	mj 7	mk 1	ml 5	mm 168	mn 20	mo 452	mp 38	mq 0	mr 97	ms 35	mt 4	mu 139	mv 3	mw 2	mx 0	my 287	mz 11
n. 6763	na 2977	nb 8	nc 213	nd 704	ne 1359	nf 11	ng 273	nh 26	ni 1725	nj 44	nk 58	nl 195	nm 19	nn 1906	no 496	np 5	nq 2	nr 44	ns 278	nt 443	nu 96	nv 55	nw 11	nx 6	ny 465	nz 145
o. 855	oa 149	ob 140	oc 114	od 190	oe 132	of 34	og 44	oh 171	oi 69	oj 16	ok 68	ol 619	om 261	on 2411	oo 115	op 95	oq 3	or 1059	os 504	ot 118	ou 275	ov 176	ow 114	ox 45	oy 103	oz 54
p. 33	pa 209	pb 2	pc 1	pd 0	pe 197	pf 1	pg 0	ph 204	pi 61	pj 1	pk 1	pl 16	pm 1	pn 1	po 59	pp 39	pq 0	pr 151	ps 16	pt 17	pu 4	pv 0	pw 0	px 0	py 12	pz 0
q. 28	qa 13	qb 0	qc 0	qd 0	qe 1	qf 0	qg 0	qh 0	qi 13	qj 0	qk 0	ql 1	qm 2	qn 0	qo 2	qp 0	qq 0	qr 1	qs 2	qt 0	qu 206	qv 0	qw 3	qx 0	qy 0	qz 0
r. 1377	ra 2356	rb 41	rc 99	rd 187	re 1697	rf 9	rg 76	rh 121	ri 3033	rj 25	rk 90	rl 413	rm 162	rn 140	ro 869	rp 14	rq 16	rr 425	rs 190	rt 208	ru 252	rv 80	rw 21	rx 3	ry 773	rz 23
s. 1169	sa 1201	sb 21	sc 60	sd 9	se 884	sf 2	sg 2	sh 1285	si 684	sj 2	sk 82	sl 279	sm 90	sn 24	so 531	sp 51	sq 1	sr 55	ss 461	st 765	su 185	sv 14	sw 24	sx 0	sy 215	sz 10
t. 483	ta 1027	tb 1	tc 17	td 0	te 716	tf 2	tg 2	th 647	ti 532	tj 3	tk 0	tl 134	tm 4	tn 22	to 667	tp 0	tq 0	tr 352	ts 35	tt 374	tu 78	tv 15	tw 11	tx 2	ty 341	tz 105
u. 155	ua 163	ub 103	uc 103	ud 136	ue 169	uf 19	ug 47	uh 58	ui 121	uj 14	uk 93	ul 301	um 154	un 275	uo 10	up 16	uq 10	ur 414	us 474	ut 82	uu 3	uv 37	uw 86	ux 34	uy 13	uz 45
v. 88	va 642	vb 1	vc 0	vd 1	ve 568	vf 0	vg 0	vh 1	vi 911	vj 0	vk 3	vl 14	vm 0	vn 8	vo 153	vp 0	vq 0	vr 48	vs 0	vt 0	vu 7	vv 7	vw 0	vx 0	vy 121	vz 0
w. 51	wa 280	wb 1	wc 0	wd 8	we 149	wf 2	wg 1	wh 23	wi 148	wj 0	wk 6	wl 13	wm 2	wn 58	wo 36	wp 0	wq 0	wr 22	ws 20	wt 8	wu 25	wv 0	ww 2	wx 0	wy 73	wz 1
x. 164	xa 103	xb 1	xc 4	xd 5	xe 36	xf 3	xg 0	xh 1	xi 102	xj 0	xk 0	xl 39	xm 1	xn 1	xo 41	xp 0	xq 0	xr 0	xs 31	xt 70	xu 5	xv 0	xw 3	xx 38	xy 30	xz 19
y. 2007	ya 2143	yb 27	yc 115	yd 272	ye 301	yf 12	yg 30	yh 22	yi 192	yj 23	yk 86	yl 1104	ym 148	yn 1826	yo 271	yp 15	yq 6	yr 291	ys 401	yt 104	yu 141	yv 106	yw 4	yx 28	yy 23	yz 78
z. 160	za 860	zb 4	zc 2	zd 2	ze 373	zf 0	zg 1	zh 43	zi 364	zj 2	zk 2	zl 123	zm 35	zn 4	zo 110	zp 2	zq 0	zr 32	zs 4	zt 4	zu 73	zv 2	zw 3	zx 1	zy 147	zz 45

Preprocessing

```
import torch

# Training dataset
xs, ys = [], []
for w in words:        # list of names
    chs = ['.'] + list(w) + ['.']
    for ch1, ch2 in zip(chs, chs[1:]):
        ix1 = stoi[ch1]  # char to index
        ix2 = stoi[ch2]
        xs.append(ix1)
        ys.append(ix2)
xs = torch.tensor(xs)
ys = torch.tensor(ys)

# Weight initialization
g = torch.Generator().manual_seed(2147483647)
W = torch.randn((27, 27), generator=g,
    requires_grad=True)
```

Training

```
import torch.nn.functional as F

# Gradient descent (maximum likelihood estimation)
for k in range(10000):
    # Forward pass
    xenc = F.one_hot(xs, num_classes=27).float() #
        one-hot encoding to avoid ordering
    logits = xenc @ W  # softmax classifier
    counts = logits.exp()
    probs = counts / counts.sum(1, keepdims=True)
    loss = -probs[torch.arange(num), ys].log().
        mean() + 0.01*(W**2).mean()
    print(loss.item())

    # Backward pass
    W.grad = None       # zero gradients
    loss.backward()
    W.data += -0.1 * W.grad
```

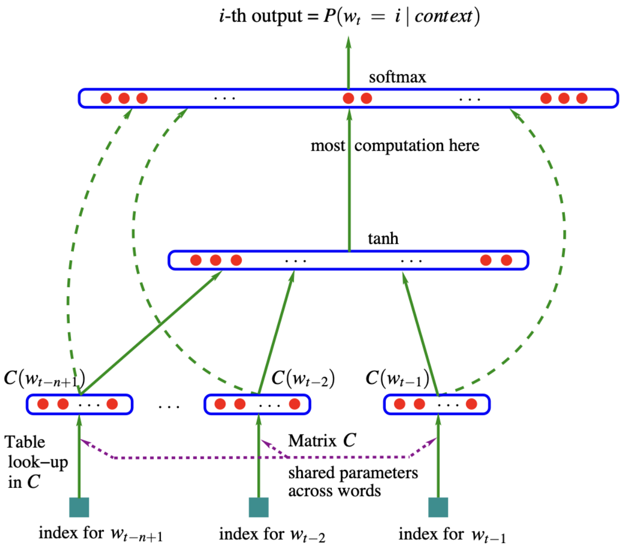
Prediction

```
# Sampling
for i in range(5):
    out = []
    ix = 0
    while True:
        xenc = F.one_hot(torch.tensor([ix]),
            num_classes=27).float()
        logits = xenc @ W
        counts = logits.exp()
        p = counts / counts.sum(1, keepdims=True)
        ix = torch.multinomial(p, num_samples=1,
            replacement=True, generator=g).item()
        out.append(itos[ix])
        if ix == 0:
            break
    print(''.join(out))  # mor. axx. minaymoryles.
        kondlaisah. anchthizarie.
```

The Road Ahead...

1. Bigram
2. Multilayer Perceptron
3. Transformer

Model Architecture



PyTorch-Like API

```
class Linear:  # torch.nn.Linear
    def __init__(self,fan_in,fan_out,bias=True):
        self.weight = torch.randn((fan_in, fan_out
            ), generator=g)  # or Kaiming init
        self.bias = torch.zeros(fan_out) if bias
            else None
    def __call__(self, x):
        self.out = x @ self.weight
        if self.bias is not None:
            self.out += self.bias
        return self.out
    def parameters(self):
        return [self.weight] + ([] if self.bias is
             None else [self.bias])

class Tanh:    # torch.nn.Tanh
    def __call__(self, x):
        self.out = torch.tanh(x)
        return self.out
```

PyTorch-Like API (Cont'd)

```
class BatchNorm1d:  # torch.nn.BatchNorm1d
    def __init__(self, dim, eps=1e-5, momentum
        =0.1):
        self.eps = eps
        self.momentum = momentum
        self.training = True
        # Trained with backprop
        self.gamma = torch.ones(dim)
        self.beta = torch.zeros(dim)
        # Trained with running momentum update
        self.running_mean = torch.zeros(dim)
        self.running_var = torch.ones(dim)

    def parameters(self):
        return [self.gamma, self.beta]
    ...
```

PyTorch-Like API (Cont'd)

```
class BatchNorm1d:  # torch.nn.BatchNorm1d
    def __call__(self, x):
        if self.training:
            xmean = x.mean(0, keepdim=True)
            xvar = x.var(0, keepdim=True)
        else:
            xmean = self.running_mean
            xvar = self.running_var
        xhat = (x - xmean) / torch.sqrt(xvar +
            self.eps)  # normalization
        self.out = self.gamma * xhat + self.beta
        if self.training:
            with torch.no_grad():
                self.running_mean = (1 - self.
                    momentum) * self.running_mean
                    + self.momentum * xmean
                self.running_var = (1 - self.
                    momentum) * self.running_var +
                     self.momentum * xvar
        return self.out
```

Preprocessing

```
for w in words:
    context = [0] * block_size
    for ch in w + '.':
        ix = stoi[ch]
        X.append(context)
        Y.append(ix)
        context = context[1:] + [ix]    # shift
X = torch.tensor(X)
Y = torch.tensor(Y)

C = torch.randn((vocab_size, n_embd), generator=g)
layers = [
    Linear(n_embd * block_size, n_hidden, bias=
        False), BatchNorm1d(n_hidden), Tanh(),
    Linear(n_hidden, n_hidden, bias=False),
        BatchNorm1d(n_hidden), Tanh(),
    ...
    Linear(n_hidden, vocab_size, bias=False),
        BatchNorm1d(vocab_size)]
```

Training

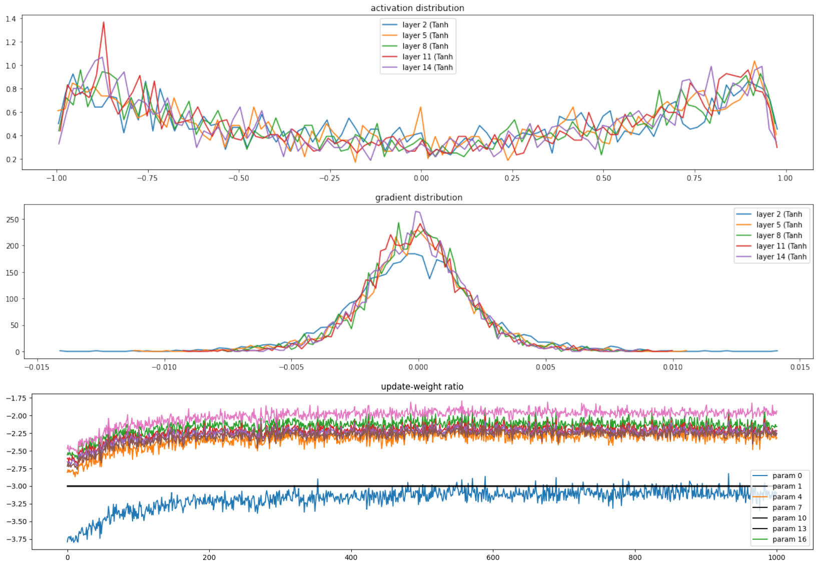
```
for i in range(200000):
    # Forward pass
    ix = torch.randint(0, X.shape[0], (batch_size
        ,)) # minibatch
    emb = C[X[ix]]
    x = emb.view(emb.shape[0], -1) # concatenate
    for layer in layers:
        x = layer(x)
    loss = F.cross_entropy(x, Y[ix])

    # Backward pass
    for layer in layers:
        layer.out.retain_grad()     # debug
    for p in parameters:
        p.grad = None
    loss.backward()
    lr = 0.1 if i < 100000 else 0.01
    for p in parameters:
        p.data += -lr * p.grad
```

Prediction

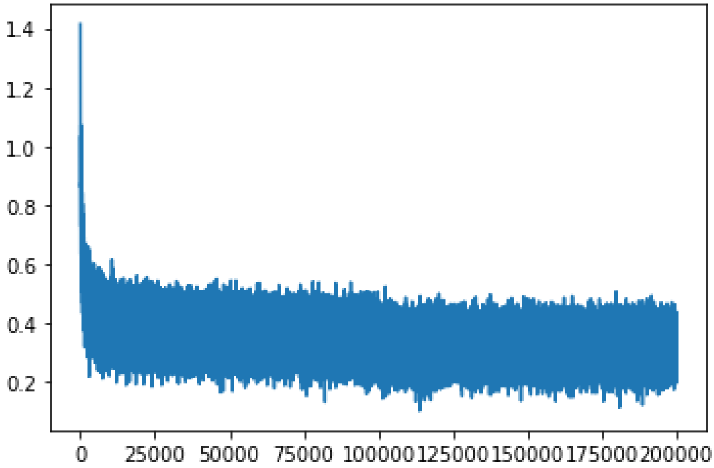
```
for _ in range(5):
    out = []
    context = [0] * block_size
    while True:
        emb = C[torch.tensor([context])]
        x = emb.view(emb.shape[0], -1)
        for layer in layers:
            x = layer(x)
        probs = F.softmax(x, dim=1)
        ix = torch.multinomial(probs, num_samples
            =1, generator=g).item()
        context = context[1:] + [ix]
        out.append(ix)
        if ix == 0:
            break
    print(''.join(itos[i] for i in out))  # carpah
        . qarlileif. jmrix. thty. sacansa.
```

Diagnoses

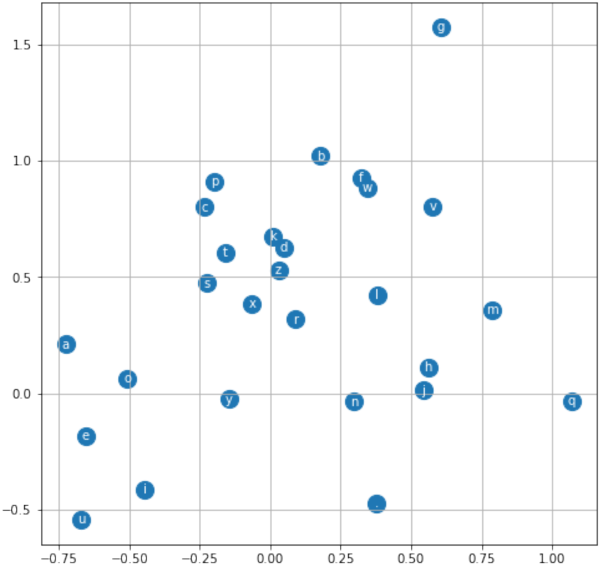


activation distribution
layer 2 (Tanh
layer 5 (Tanh
layer 8 (Tanh
layer 11 (Tanh
layer 14 (Tanh
gradient distribution
layer 2 (Tanh
layer 5 (Tanh
layer 8 (Tanh
layer 11 (Tanh
layer 14 (Tanh
update-weight ratio
param 0
param 1
param 4
param 7
param 10
param 13
param 16

Loss Trace (log10)



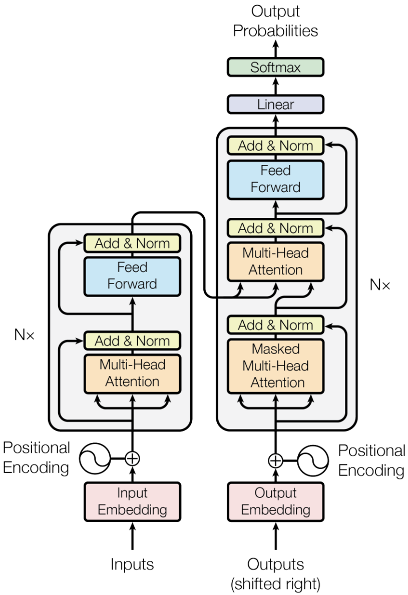
Character Embeddings



The Road Ahead...

1. Bigram
2. Multilayer Perceptron
3. Transformer

Model Architecture



Attention

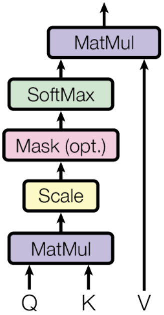
Scaled dot-product attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

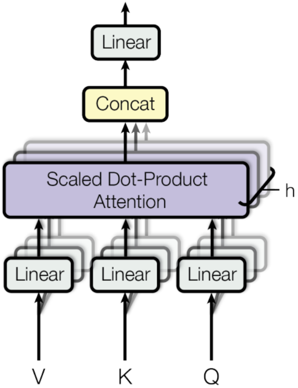
- Key concepts
  - *query* (Q) represents current word, *key* (K) captures other words, *value* (V) carries information
  - encoder vs. decoder, self-attention vs. cross-attention
- Why attention mechanism?
  - eliminate need for recurrence or convolution
  - superior performance while maintaining parallelization

Attention (Cont'd)

Scaled Dot-Product Attention



Multi-Head Attention



Single-Head Attention

```python
class Head(nn.Module):
    def __init__(self, head_size):
        super().__init__()
        self.key = nn.Linear(n_embd, head_size,
            bias=False)
        self.query = nn.Linear(n_embd, head_size,
            bias=False)
        self.value = nn.Linear(n_embd, head_size,
            bias=False)
        self.register_buffer('tril', torch.tril(
            torch.ones(block_size, block_size)))
            # not updated during backprop
        self.dropout = nn.Dropout(dropout)
    ...
```

Single-Head Attention (Cont'd)

```
class Head(nn.Module):
    def forward(self, x):
        B,T,C = x.shape # (batch, time, channels)
        k = self.key(x)
        q = self.query(x)
        wei = q @ k.transpose(-2,-1) * k.shape
            [-1]**-0.5  # attention scores
        wei = wei.masked_fill(self.tril[:T, :T] ==
             0, float('-inf'))   # decoder
        wei = F.softmax(wei, dim=-1)
        wei = self.dropout(wei)
        v = self.value(x)
        out = wei @ v   # (batch, time, head size)
        return out
```

Multi-Head Attention

```
class MultiHead(nn.Module):
    def __init__(self, num_heads, head_size):
        super().__init__()
        self.heads = nn.ModuleList([Head(head_size
            ) for _ in range(num_heads)])
        self.proj = nn.Linear(head_size *
            num_heads, n_embd)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x):
        out = torch.cat([h(x) for h in self.heads
            ], dim=-1)
        out = self.dropout(self.proj(out))
        return out
```

Feed Forward

```
class FeedFoward(nn.Module):
    def __init__(self, n_embd):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(n_embd, 4 * n_embd),
            nn.ReLU(),
            nn.Linear(4 * n_embd, n_embd),
            nn.Dropout(dropout))

    def forward(self, x):
        return self.net(x)
```

Transformer Block

```
class Block(nn.Module):
    def __init__(self, n_embd, n_head):
        super().__init__()
        head_size = n_embd // n_head
        self.sa = MultiHeadAttention(n_head,
            head_size)
        self.ffwd = FeedFoward(n_embd)
        self.ln1 = nn.LayerNorm(n_embd)
        self.ln2 = nn.LayerNorm(n_embd)

    def forward(self, x):
        x = x + self.sa(self.ln1(x))
        x = x + self.ffwd(self.ln2(x))
        return x
```

GPT

```
class GPT(nn.Module):
    def __init__(self):
        super().__init__()
        self.token_embedding_table = nn.Embedding(
            vocab_size, n_embd)
        self.position_embedding_table = nn.
            Embedding(block_size, n_embd)
        self.blocks = nn.Sequential(*[Block(n_embd
            , n_head=n_head) for _ in range(
            n_layer)])
        self.ln_f = nn.LayerNorm(n_embd)
        self.lm_head = nn.Linear(n_embd,
            vocab_size)
    ...
```

GPT (Cont'd)

```
class GPT(nn.Module):
    def forward(self, idx, targets):
        B, T = idx.shape
        tok_emb = self.token_embedding_table(idx)
        pos_emb = self.position_embedding_table(
            torch.arange(T))
        x = tok_emb + pos_emb
        x = self.blocks(x)
        x = self.ln_f(x)
        logits = self.lm_head(x)
        B, T, C = logits.shape
        logits = logits.view(B*T, C)
        targets = targets.view(B*T)
        loss = F.cross_entropy(logits, targets)
        return logits, loss

device = torch.device("mps" if torch.backends.mps.
    is_available() else "cpu")
model = GPT().to(device)
```

Prediction

```
# wget https://raw.githubusercontent.com/karpathy/
    char-rnn/master/data/tinyshakespeare/input.txt
First Citizen:
Before we proceed any further, hear me speak.

All:
Speak, speak.

# Output: Shakespeare-like text

VALHASINA:
Nobleman; go, then both groans to us.

AUFIDIUS:
O those prepation!
```

Tokenizer

- Byte-pair encoding for tokenizing text
  - iteratively replace most frequent pair of consecutive UTF-8 bytes (characters) with single byte
  - reduce vocabulary while being able to encode *any* word
- Example: "aaabdaaabac" $\rightarrow$ "XdXac", X=ZY, Y=ab, Z=aa

```
from minbpe import BasicTokenizer
tokenizer = BasicTokenizer()
text = "aaabdaaabac"
tokenizer.train(text, 256 + 3)  # 256 byte
    tokens + 3 merges
print(tokenizer.encode(text))   # [258, 100,
    258, 97, 99]
print(tokenizer.decode([258, 100, 258, 97,
    99]))  # aaabdaaabac
```

References

- github.com/karpathy/makemore – An autoregressive character-level language model for making more things
- github.com/karpathy/nanoGPT – The simplest, fastest repository for training/finetuning medium-sized GPTs
- github.com/karpathy/minbpe – Minimal, clean code for the Byte Pair Encoding (BPE) algorithm commonly used in LLM tokenization
- Bengio et al. (2003), "A Neural Probabilistic Language Model", *Journal of Machine Learning Research*
- Vaswani et al. (2017), "Attention is All You Need", arXiv:1706.03762