# Building a GPT

Companion notebook to the Zero To Hero video on GPT.

```
# Kami selalu memulai dengan kumpulan data untuk dilatih. Mari unduh
kumpulan data shakespeare kecil
!wget https://raw.githubusercontent.com/karpathy/char-
rnn/master/data/tinyshakespeare/input.txt
```

```
--2025-01-04 15:11:34--
https://raw.githubusercontent.com/karpathy/char-rnn/master/data/tinysh
akespeare/input.txt
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|
185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1115394 (1.1M) [text/plain]
Saving to: ‘input.txt’

 input.txt             0%[                    ]       0  --.-KB/s
input.txt           100%[===================>]   1.06M  --.-KB/s    in
0.04s

2025-01-04 15:11:34 (26.9 MB/s) - ‘input.txt’ saved [1115394/1115394]
```

```
# bacalah untuk memeriksanya
with open('input.txt', 'r', encoding='utf-8') as f:
    text = f.read()

print("length of dataset in characters: ", len(text))
```

```
length of dataset in characters:  1115394
```

```
# mari kita lihat 1000 karakter pertama
print(text[:1000])
```

```
First Citizen:
Before we proceed any further, hear me speak.

All:
Speak, speak.

First Citizen:
You are all resolved rather to die than to famish?

All:
Resolved. resolved.
```

```
First Citizen:
First, you know Caius Marcius is chief enemy to the people.

All:
We know't, we know't.

First Citizen:
Let us kill him, and we'll have corn at our own price.
Is't a verdict?

All:
No more talking on't; let it be done: away, away!

Second Citizen:
One word, good citizens.

First Citizen:
We are accounted poor citizens, the patricians good.
What authority surfeits on would relieve us: if they
would yield us but the superfluity, while it were
wholesome, we might guess they relieved us humanely;
but they think we are too dear: the leanness that
afflicts us, the object of our misery, is as an
inventory to particularise their abundance; our
sufferance is a gain to them Let us revenge this with
our pikes, ere we become rakes: for the gods know I
speak this in hunger for bread, not in thirst for revenge.
```

```
# berikut adalah semua karakter unik yang muncul dalam teks ini
chars = sorted(list(set(text)))
vocab_size = len(chars)
print(''.join(chars))
print(vocab_size)
```

```
 !$&',-.3:;?ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
65
```

```
# membuat pemetaan dari karakter ke bilangan bulat
stoi = { ch:i for i,ch in enumerate(chars) }
itos = { i:ch for i,ch in enumerate(chars) }
encode = lambda s: [stoi[c] for c in s] # encoder: ambil string,
keluarkan daftar bilangan bulat
decode = lambda l: ''.join([itos[i] for i in l]) # decoder: ambil
daftar bilangan bulat, keluarkan string

print(encode("hii there"))
print(decode(encode("hii there")))
```

```
[46, 47, 47, 1, 58, 46, 43, 56, 43]
hii there
```

```python
# sekarang mari kita enkode seluruh kumpulan data teks dan simpan ke
dalam torch.Tensor
import torch # kami menggunakan PyTorch: https://pytorch.org
data = torch.tensor(encode(text), dtype=torch.long)
print(data.shape, data.dtype)
print(data[:1000]) # 1000 karakter yang kami lihat sebelumnya akan
terlihat seperti ini di GPT
```

```
torch.Size([1115394]) torch.int64
tensor([18, 47, 56, 57, 58,  1, 15, 47, 58, 47, 64, 43, 52, 10,  0,
14, 43, 44,
        53, 56, 43,  1, 61, 43,  1, 54, 56, 53, 41, 43, 43, 42,  1,
39, 52, 63,
         1, 44, 59, 56, 58, 46, 43, 56,  6,  1, 46, 43, 39, 56,  1,
51, 43,  1,
        57, 54, 43, 39, 49,  8,  0,  0, 13, 50, 50, 10,  0, 31, 54,
43, 39, 49,
         6,  1, 57, 54, 43, 39, 49,  8,  0,  0, 18, 47, 56, 57, 58,
1, 15, 47,
        58, 47, 64, 43, 52, 10,  0, 37, 53, 59,  1, 39, 56, 43,  1,
39, 50, 50,
         1, 56, 43, 57, 53, 50, 60, 43, 42,  1, 56, 39, 58, 46, 43,
56,  1, 58,
        53,  1, 42, 47, 43,  1, 58, 46, 39, 52,  1, 58, 53,  1, 44,
39, 51, 47,
        57, 46, 12,  0,  0, 13, 50, 50, 10,  0, 30, 43, 57, 53, 50,
60, 43, 42,
         8,  1, 56, 43, 57, 53, 50, 60, 43, 42,  8,  0,  0, 18, 47,
56, 57, 58,
         1, 15, 47, 58, 47, 64, 43, 52, 10,  0, 18, 47, 56, 57, 58,
6,  1, 63,
        53, 59,  1, 49, 52, 53, 61,  1, 15, 39, 47, 59, 57,  1, 25,
39, 56, 41,
        47, 59, 57,  1, 47, 57,  1, 41, 46, 47, 43, 44,  1, 43, 52,
43, 51, 63,
         1, 58, 53,  1, 58, 46, 43,  1, 54, 43, 53, 54, 50, 43,  8,
0,  0, 13,
        50, 50, 10,  0, 35, 43,  1, 49, 52, 53, 61,  5, 58,  6,  1,
61, 43,  1,
        49, 52, 53, 61,  5, 58,  8,  0,  0, 18, 47, 56, 57, 58,  1,
15, 47, 58,
        47, 64, 43, 52, 10,  0, 24, 43, 58,  1, 59, 57,  1, 49, 47,
50, 50,  1,
        46, 47, 51,  6,  1, 39, 52, 42,  1, 61, 43,  5, 50, 50,  1,
46, 39, 60,
        43,  1, 41, 53, 56, 52,  1, 39, 58,  1, 53, 59, 56,  1, 53,
61, 52,  1,
```

```
        54, 56, 47, 41, 43,  8,  0, 21, 57,  5, 58,  1, 39,  1, 60,
43, 56, 42,
        47, 41, 58, 12,  0,  0, 13, 50, 50, 10,  0, 26, 53,  1, 51,
53, 56, 43,
         1, 58, 39, 50, 49, 47, 52, 45,  1, 53, 52,  5, 58, 11,  1,
50, 43, 58,
         1, 47, 58,  1, 40, 43,  1, 42, 53, 52, 43, 10,  1, 39, 61,
39, 63,  6,
         1, 39, 61, 39, 63,  2,  0,  0, 31, 43, 41, 53, 52, 42,  1,
15, 47, 58,
        47, 64, 43, 52, 10,  0, 27, 52, 43,  1, 61, 53, 56, 42,  6,
1, 45, 53,
        53, 42,  1, 41, 47, 58, 47, 64, 43, 52, 57,  8,  0,  0, 18,
47, 56, 57,
        58,  1, 15, 47, 58, 47, 64, 43, 52, 10,  0, 35, 43,  1, 39,
56, 43,  1,
        39, 41, 41, 53, 59, 52, 58, 43, 42,  1, 54, 53, 53, 56,  1,
41, 47, 58,
        47, 64, 43, 52, 57,  6,  1, 58, 46, 43,  1, 54, 39, 58, 56,
47, 41, 47,
        39, 52, 57,  1, 45, 53, 53, 42,  8,  0, 35, 46, 39, 58,  1,
39, 59, 58,
        46, 53, 56, 47, 58, 63,  1, 57, 59, 56, 44, 43, 47, 58, 57,
1, 53, 52,
         1, 61, 53, 59, 50, 42,  1, 56, 43, 50, 47, 43, 60, 43,  1,
59, 57, 10,
         1, 47, 44,  1, 58, 46, 43, 63,  0, 61, 53, 59, 50, 42,  1,
63, 47, 43,
        50, 42,  1, 59, 57,  1, 40, 59, 58,  1, 58, 46, 43,  1, 57,
59, 54, 43,
        56, 44, 50, 59, 47, 58, 63,  6,  1, 61, 46, 47, 50, 43,  1,
47, 58,  1,
        61, 43, 56, 43,  0, 61, 46, 53, 50, 43, 57, 53, 51, 43,  6,
1, 61, 43,
         1, 51, 47, 45, 46, 58,  1, 45, 59, 43, 57, 57,  1, 58, 46,
43, 63,  1,
        56, 43, 50, 47, 43, 60, 43, 42,  1, 59, 57,  1, 46, 59, 51,
39, 52, 43,
        50, 63, 11,  0, 40, 59, 58,  1, 58, 46, 43, 63,  1, 58, 46,
47, 52, 49,
         1, 61, 43,  1, 39, 56, 43,  1, 58, 53, 53,  1, 42, 43, 39,
56, 10,  1,
        58, 46, 43,  1, 50, 43, 39, 52, 52, 43, 57, 57,  1, 58, 46,
39, 58,  0,
        39, 44, 44, 50, 47, 41, 58, 57,  1, 59, 57,  6,  1, 58, 46,
43,  1, 53,
        40, 48, 43, 41, 58,  1, 53, 44,  1, 53, 59, 56,  1, 51, 47,
57, 43, 56,
        63,  6,  1, 47, 57,  1, 39, 57,  1, 39, 52,  0, 47, 52, 60,
```

```
43, 52, 58,
        53, 56, 63,  1, 58, 53,  1, 54, 39, 56, 58, 47, 41, 59, 50,
39, 56, 47,
        57, 43,  1, 58, 46, 43, 47, 56,  1, 39, 40, 59, 52, 42, 39,
52, 41, 43,
        11,  1, 53, 59, 56,  0, 57, 59, 44, 44, 43, 56, 39, 52, 41,
43,  1, 47,
        57,  1, 39,  1, 45, 39, 47, 52,  1, 58, 53,  1, 58, 46, 43,
51,  1, 24,
        43, 58,  1, 59, 57,  1, 56, 43, 60, 43, 52, 45, 43,  1, 58,
46, 47, 57,
         1, 61, 47, 58, 46,  0, 53, 59, 56,  1, 54, 47, 49, 43, 57,
6,  1, 43,
        56, 43,  1, 61, 43,  1, 40, 43, 41, 53, 51, 43,  1, 56, 39,
49, 43, 57,
        10,  1, 44, 53, 56,  1, 58, 46, 43,  1, 45, 53, 42, 57,  1,
49, 52, 53,
        61,  1, 21,  0, 57, 54, 43, 39, 49,  1, 58, 46, 47, 57,  1,
47, 52,  1,
        46, 59, 52, 45, 43, 56,  1, 44, 53, 56,  1, 40, 56, 43, 39,
42,  6,  1,
        52, 53, 58,  1, 47, 52,  1, 58, 46, 47, 56, 57, 58,  1, 44,
53, 56,  1,
        56, 43, 60, 43, 52, 45, 43,  8,  0,  0])
```

```python
# Sekarang mari kita bagi data menjadi rangkaian pelatihan dan validasi
n = int(0.9*len(data)) # 90% pertama akan dilatih, sisanya val
train_data = data[:n]
val_data = data[n:]
```

```python
block_size = 8
train_data[:block_size+1]
```

```
tensor([18, 47, 56, 57, 58,  1, 15, 47, 58])
```

```python
x = train_data[:block_size]
y = train_data[1:block_size+1]
for t in range(block_size):
    context = x[:t+1]
    target = y[t]
    print(f"when input is {context} the target: {target}")
```

```
when input is tensor([18]) the target: 47
when input is tensor([18, 47]) the target: 56
when input is tensor([18, 47, 56]) the target: 57
when input is tensor([18, 47, 56, 57]) the target: 58
when input is tensor([18, 47, 56, 57, 58]) the target: 1
when input is tensor([18, 47, 56, 57, 58,  1]) the target: 15
```

```
when input is tensor([18, 47, 56, 57, 58,  1, 15]) the target: 47
when input is tensor([18, 47, 56, 57, 58,  1, 15, 47]) the target: 58
```

```
torch.manual_seed(1337)
batch_size = 4 # berapa banyak barisan independen yang akan kita
proses secara paralel?
block_size = 8 # berapa panjang konteks maksimum untuk prediksi?

def get_batch(split):
    # menghasilkan sejumlah kecil input data x dan target y
    data = train_data if split == 'train' else val_data
    ix = torch.randint(len(data) - block_size, (batch_size,))
    x = torch.stack([data[i:i+block_size] for i in ix])
    y = torch.stack([data[i+1:i+block_size+1] for i in ix])
    return x, y

xb, yb = get_batch('train')
print('inputs:')
print(xb.shape)
print(xb)
print('targets:')
print(yb.shape)
print(yb)

print('----')

for b in range(batch_size): # dimensi kumpulan
    for t in range(block_size): # dimensi waktu
        context = xb[b, :t+1]
        target = yb[b,t]
        print(f"when input is {context.tolist()} the target:
{target}")
```

```
inputs:
torch.Size([4, 8])
tensor([[24, 43, 58,  5, 57,  1, 46, 43],
        [44, 53, 56,  1, 58, 46, 39, 58],
        [52, 58,  1, 58, 46, 39, 58,  1],
        [25, 17, 27, 10,  0, 21,  1, 54]])
targets:
torch.Size([4, 8])
tensor([[43, 58,  5, 57,  1, 46, 43, 39],
        [53, 56,  1, 58, 46, 39, 58,  1],
        [58,  1, 58, 46, 39, 58,  1, 46],
        [17, 27, 10,  0, 21,  1, 54, 39]])
----
when input is [24] the target: 43
when input is [24, 43] the target: 58
when input is [24, 43, 58] the target: 5
when input is [24, 43, 58, 5] the target: 57
```

```
when input is [24, 43, 58, 5, 57] the target: 1
when input is [24, 43, 58, 5, 57, 1] the target: 46
when input is [24, 43, 58, 5, 57, 1, 46] the target: 43
when input is [24, 43, 58, 5, 57, 1, 46, 43] the target: 39
when input is [44] the target: 53
when input is [44, 53] the target: 56
when input is [44, 53, 56] the target: 1
when input is [44, 53, 56, 1] the target: 58
when input is [44, 53, 56, 1, 58] the target: 46
when input is [44, 53, 56, 1, 58, 46] the target: 39
when input is [44, 53, 56, 1, 58, 46, 39] the target: 58
when input is [44, 53, 56, 1, 58, 46, 39, 58] the target: 1
when input is [52] the target: 58
when input is [52, 58] the target: 1
when input is [52, 58, 1] the target: 58
when input is [52, 58, 1, 58] the target: 46
when input is [52, 58, 1, 58, 46] the target: 39
when input is [52, 58, 1, 58, 46, 39] the target: 58
when input is [52, 58, 1, 58, 46, 39, 58] the target: 1
when input is [52, 58, 1, 58, 46, 39, 58, 1] the target: 46
when input is [25] the target: 17
when input is [25, 17] the target: 27
when input is [25, 17, 27] the target: 10
when input is [25, 17, 27, 10] the target: 0
when input is [25, 17, 27, 10, 0] the target: 21
when input is [25, 17, 27, 10, 0, 21] the target: 1
when input is [25, 17, 27, 10, 0, 21, 1] the target: 54
when input is [25, 17, 27, 10, 0, 21, 1, 54] the target: 39

print(xb) # masukan kita ke trafo

tensor([[24, 43, 58,  5, 57,  1, 46, 43],
        [44, 53, 56,  1, 58, 46, 39, 58],
        [52, 58,  1, 58, 46, 39, 58,  1],
        [25, 17, 27, 10,  0, 21,  1, 54]])

import torch
import torch.nn as nn
from torch.nn import functional as F
torch.manual_seed(1337)

class BigramLanguageModel(nn.Module):

    def __init__(self, vocab_size):
        super().__init__()
        # setiap token secara langsung membaca logit untuk token
berikutnya dari tabel pencarian
        self.token_embedding_table = nn.Embedding(vocab_size,
vocab_size)
```

```
    def forward(self, idx, targets=None):

        # idx dan target keduanya merupakan tensor bilangan bulat
(B,T).
        logits = self.token_embedding_table(idx) # (B,T,C)

        if targets is None:
            loss = None
        else:
            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

    def generate(self, idx, max_new_tokens):
        # idx adalah array indeks (B, T) dalam konteks saat ini
        for _ in range(max_new_tokens):
            # dapatkan prediksinya
            logits, loss = self(idx)
            # fokus hanya pada langkah terakhir kali
            logits = logits[:, -1, :] # menjadi (B,C)
            # terapkan softmax untuk mendapatkan probabilitas
            probs = F.softmax(logits, dim=-1) # (B, C)
            # sampel dari distribusi
            idx_next = torch.multinomial(probs, num_samples=1) # (B,
1)
            # tambahkan indeks sampel ke urutan yang sedang berjalan
            idx = torch.cat((idx, idx_next), dim=1) # (B, T+1)
        return idx

m = BigramLanguageModel(vocab_size)
logits, loss = m(xb, yb)
print(logits.shape)
print(loss)

print(decode(m.generate(idx = torch.zeros((1, 1), dtype=torch.long),
max_new_tokens=100)[0].tolist()))

torch.Size([32, 65])
tensor(4.8786, grad_fn=<NllLossBackward0>)

Sr?qP-QWktXoL&jLDJgOLVz'RIoDqHdhsV&vLLxatjscMpwLERSPyao.qfzs$Ys$zF-
w,;eEkzxjgCKFChs!iWW.ObzDnxA Ms$3

# buat pengoptimal PyTorch
optimizer = torch.optim.AdamW(m.parameters(), lr=1e-3)

batch_size = 32
for steps in range(100): # tingkatkan jumlah langkah untuk hasil yang
```

```
baik...

    # sampel kumpulan data
    xb, yb = get_batch('train')

    # evaluasi kerugian
    logits, loss = m(xb, yb)
    optimizer.zero_grad(set_to_none=True)
    loss.backward()
    optimizer.step()

print(loss.item())
```

```
4.587916374206543
```

```
print(decode(m.generate(idx = torch.zeros((1, 1), dtype=torch.long),
max_new_tokens=500)[0].tolist()))
```

```

xiKi-RJ:CgqVuUa!U?qMH.uk!sCuMXvv!CJFfx;LgRyJknOEti.?I&-gPlLyulId?
XlaInQ'q,lT$
3Q&sGlvHQ?mqSq-eON
x?SP fUAfCAuCX:bOlgiRQWN:Mphaw
tRLKuYXEaAXxrcq-gCUzeh3w!AcyaylgYWjmJM?
Uzw:inaY,:C&OECW:vmGGJAn3onAuMgia!ms$Vb q-gCOcPcUhOnxJGUGSPJWT:.?
ujmJFoiNL&A'DxY,prZ?qdT;hoo'dHooXXlxf'WkHK&u3Q?rqUi.kz;?Yx?
C&u3Qbfzxlyh'Vl:zyxjKXgC?
lv'QKFiBeviNxO'm!Upm$srm&TqViqiBD3HBP!juEOpmZJyF$Fwfy!PlvWPFC
&WDdP!Ko,px
x
tREOE;AJ.BeXkylOVD3KHp$e?nD,.SFbWWI'ubcL!q-tU;aXmJ&uGXHxJXI&Z!
gHRpajj;l.
pTErIBjx;JKIgoCnLGXrJSP!AU-AcbczR?
```

## The mathematical trick in self-attention

```
# contoh mainan yang mengilustrasikan bagaimana perkalian matriks
dapat digunakan untuk "agregasi berbobot"
torch.manual_seed(42)
a = torch.tril(torch.ones(3, 3))
a = a / torch.sum(a, 1, keepdim=True)
b = torch.randint(0,10,(3,2)).float()
c = a @ b
print('a=')
print(a)
print('--')
print('b=')
print(b)
print('--')
```

```
print('c=')
print(c)
```

```
a=
tensor([[1.0000, 0.0000, 0.0000],
        [0.5000, 0.5000, 0.0000],
        [0.3333, 0.3333, 0.3333]])
--
b=
tensor([[2., 7.],
        [6., 4.],
        [6., 5.]])
--
c=
tensor([[2.0000, 7.0000],
        [4.0000, 5.5000],
        [4.6667, 5.3333]])
```

```
# perhatikan contoh mainan berikut :

torch.manual_seed(1337)
B,T,C = 4,8,2 # batch, waktu, saluran
x = torch.randn(B,T,C)
x.shape
```

```
torch.Size([4, 8, 2])
```

```
# Kita ingin x[b,t] = mean_{i<=t} x[b,i]
xbow = torch.zeros((B,T,C))
for b in range(B):
    for t in range(T):
        xprev = x[b,:t+1] # (t,C)
        xbow[b,t] = torch.mean(xprev, 0)
```

```
# versi 2: menggunakan perkalian matriks untuk agregasi berbobot
wei = torch.tril(torch.ones(T, T))
wei = wei / wei.sum(1, keepdim=True)
xbow2 = wei @ x # (B, T, T) @ (B, T, C) ----> (B, T, C)
torch.allclose(xbow, xbow2)
```

```
False
```

```
# versi 3: gunakan Softmax
tril = torch.tril(torch.ones(T, T))
wei = torch.zeros((T,T))
wei = wei.masked_fill(tril == 0, float('-inf'))
wei = F.softmax(wei, dim=-1)
xbow3 = wei @ x
torch.allclose(xbow, xbow3)
```

```
False
```

```
# versi 4: perhatian diri!
torch.manual_seed(1337)
B,T,C = 4,8,32 # batch, waktu, saluran
x = torch.randn(B,T,C)

# mari kita lihat seorang Kepala melakukan perhatian diri
head_size = 16
key = nn.Linear(C, head_size, bias=False)
query = nn.Linear(C, head_size, bias=False)
value = nn.Linear(C, head_size, bias=False)
k = key(x)   # (B, T, 16)
q = query(x) # (B, T, 16)
wei =  q @ k.transpose(-2, -1) # (B, T, 16) @ (B, 16, T) ---> (B, T, T)

tril = torch.tril(torch.ones(T, T))
#wei = torch.zeros((T,T))
wei = wei.masked_fill(tril == 0, float('-inf'))
wei = F.softmax(wei, dim=-1)

v = value(x)
out = wei @ v
#out = wei @ x

out.shape
```

```
torch.Size([4, 8, 16])
```

```
wei[0]
```

```
tensor([[1.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.1574, 0.8426, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.2088, 0.1646, 0.6266, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.5792, 0.1187, 0.1889, 0.1131, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.0294, 0.1052, 0.0469, 0.0276, 0.7909, 0.0000, 0.0000, 0.0000],
        [0.0176, 0.2689, 0.0215, 0.0089, 0.6812, 0.0019, 0.0000, 0.0000],
        [0.1691, 0.4066, 0.0438, 0.0416, 0.1048, 0.2012, 0.0329, 0.0000],
        [0.0210, 0.0843, 0.0555, 0.2297, 0.0573, 0.0709, 0.2423, 0.2391]],
       grad_fn=<SelectBackward0>)
```

Notes:

- Attention is a **communication mechanism**. Can be seen as nodes in a directed graph looking at each other and aggregating information with a weighted sum from all nodes that point to them, with data-dependent weights.
- There is no notion of space. Attention simply acts over a set of vectors. This is why we need to positionally encode tokens.
- Each example across batch dimension is of course processed completely independently and never "talk" to each other
- In an "encoder" attention block just delete the single line that does masking with `tril`, allowing all tokens to communicate. This block here is called a "decoder" attention block because it has triangular masking, and is usually used in autoregressive settings, like language modeling.
- "self-attention" just means that the keys and values are produced from the same source as queries. In "cross-attention", the queries still get produced from x, but the keys and values come from some other, external source (e.g. an encoder module)
- "Scaled" attention additional divides `wei` by 1/sqrt(head_size). This makes it so when input Q,K are unit variance, wei will be unit variance too and Softmax will stay diffuse and not saturate too much. Illustration below

```
k = torch.randn(B,T,head_size)
q = torch.randn(B,T,head_size)
wei = q @ k.transpose(-2, -1) * head_size**-0.5

k.var()

tensor(1.0449)

q.var()

tensor(1.0700)

wei.var()

tensor(1.0918)

torch.softmax(torch.tensor([0.1, -0.2, 0.3, -0.2, 0.5]), dim=-1)

tensor([0.1925, 0.1426, 0.2351, 0.1426, 0.2872])

torch.softmax(torch.tensor([0.1, -0.2, 0.3, -0.2, 0.5])*8, dim=-1) #
menjadi terlalu pucat, menyatu menjadi satu-panas

tensor([0.0326, 0.0030, 0.1615, 0.0030, 0.8000])

class LayerNorm1d: # (dulu BatchNorm1d)

  def __init__(self, dim, eps=1e-5, momentum=0.1):
    self.eps = eps
    self.gamma = torch.ones(dim)
    self.beta = torch.zeros(dim)

  def __call__(self, x):
```

```
    # hitung umpan maju
    xmean = x.mean(1, keepdim=True) # maksud kumpulan
    xvar = x.var(1, keepdim=True) # varians batch
    xhat = (x - xmean) / torch.sqrt(xvar + self.eps) # normalisasi ke
unit varians
    self.out = self.gamma * xhat + self.beta
    return self.out

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

torch.manual_seed(1337)
module = LayerNorm1d(100)
x = torch.randn(32, 100) # ukuran batch 32 dari vektor 100 dimensi
x = module(x)
x.shape
```

```
torch.Size([32, 100])
```

```
x[:,0].mean(), x[:,0].std() # mean,std dari satu fitur di semua input
batch
```

```
(tensor(0.1469), tensor(0.8803))
```

```
x[0,:].mean(), x[0,:].std() # mean,std dari satu masukan dari batch,
fitur-fiturnya
```

```
(tensor(-9.5367e-09), tensor(1.0000))
```

```
# Contoh terjemahan Bahasa Prancis ke Bahasa Inggris:

# <--------- ENCODE ------------------><--------------- DECODE
----------------->
# les réseaux de neurones sont géniaux! <START> jaringan neural luar
biasa!<END>
```

## Full finished code, for reference

You may want to refer directly to the git repo instead though.

```
import torch
import torch.nn as nn
from torch.nn import functional as F

# hyperparameters
batch_size = 16 #berapa banyak barisan independen yang akan kita
proses secara paralel?
block_size = 32 # berapa panjang konteks maksimum untuk prediksi?
max_iters = 5000
eval_interval = 100
```

```python
learning_rate = 1e-3
device = 'cuda' if torch.cuda.is_available() else 'cpu'
eval_iters = 200
n_embd = 64
n_head = 4
n_layer = 4
dropout = 0.0
# ------------

torch.manual_seed(1337)

# wget
https://raw.githubusercontent.com/karpathy/char-rnn/master/data/tinysh
akespeare/input.txt
with open('input.txt', 'r', encoding='utf-8') as f:
    text = f.read()

# berikut adalah semua karakter unik yang muncul dalam teks ini
chars = sorted(list(set(text)))
vocab_size = len(chars)
# membuat pemetaan dari karakter ke bilangan bulat
stoi = { ch:i for i,ch in enumerate(chars) }
itos = { i:ch for i,ch in enumerate(chars) }
encode = lambda s: [stoi[c] for c in s] # encoder: ambil string,
keluarkan daftar bilangan bulat
decode = lambda l: ''.join([itos[i] for i in l]) # decoder: ambil
daftar bilangan bulat, keluarkan string

# Latih dan uji perpecahan
data = torch.tensor(encode(text), dtype=torch.long)
n = int(0.9*len(data)) # 90% pertama akan dilatih, sisanya val
train_data = data[:n]
val_data = data[n:]

# memuat data
def get_batch(split):
    # menghasilkan sejumlah kecil data input x dan target y
    data = train_data if split == 'train' else val_data
    ix = torch.randint(len(data) - block_size, (batch_size,))
    x = torch.stack([data[i:i+block_size] for i in ix])
    y = torch.stack([data[i+1:i+block_size+1] for i in ix])
    x, y = x.to(device), y.to(device)
    return x, y

@torch.no_grad()
def estimate_loss():
    out = {}
    model.eval()
    for split in ['train', 'val']:
        losses = torch.zeros(eval_iters)
```

```
        for k in range(eval_iters):
            X, Y = get_batch(split)
            logits, loss = model(X, Y)
            losses[k] = loss.item()
        out[split] = losses.mean()
    model.train()
    return out

class Head(nn.Module):
    """ one head of self-attention """

    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)))

        self.dropout = nn.Dropout(dropout)

    def forward(self, x):
        B,T,C = x.shape
        k = self.key(x)   # (B,T,C)
        q = self.query(x) # (B,T,C)
        # menghitung skor perhatian ("afinitas")
        wei = q @ k.transpose(-2,-1) * C**-0.5 # (B, T, C) @ (B, C, T)
-> (B, T, T)
        wei = wei.masked_fill(self.tril[:T, :T] == 0, float('-inf')) #
(B, T, T)
        wei = F.softmax(wei, dim=-1) # (B, T, T)
        wei = self.dropout(wei)
        # melakukan agregasi nilai tertimbang
        v = self.value(x) # (B,T,C)
        out = wei @ v # (B, T, T) @ (B, T, C) -> (B, T, C)
        return out

class MultiHeadAttention(nn.Module):
    """ multiple heads of self-attention in parallel """

    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(n_embd, 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))
```

```python
        return out

class FeedFoward(nn.Module):
    """ a simple linear layer followed by a non-linearity """

    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)

class Block(nn.Module):
    """ Transformer block: communication followed by computation """

    def __init__(self, n_embd, n_head):
        # n_embd: menyematkan dimensi, n_head: jumlah kepala yang kita
inginkan
        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

# model bigram super sederhana
class BigramLanguageModel(nn.Module):

    def __init__(self):
        super().__init__()
        # setiap token secara langsung membaca logit untuk token
berikutnya dari tabel pencarian
        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) # norma lapisan akhir
        self.lm_head = nn.Linear(n_embd, vocab_size)
```

```
    def forward(self, idx, targets=None):
        B, T = idx.shape

        # idx dan target keduanya merupakan tensor bilangan bulat
(B,T).
        tok_emb = self.token_embedding_table(idx) # (B,T,C)
        pos_emb = self.position_embedding_table(torch.arange(T,
device=device)) # (T,C)
        x = tok_emb + pos_emb # (B,T,C)
        x = self.blocks(x) # (B,T,C)
        x = self.ln_f(x) # (B,T,C)
        logits = self.lm_head(x) # (B,T,vocab_size)

        if targets is None:
            loss = None
        else:
            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

    def generate(self, idx, max_new_tokens):
        # idx adalah array indeks (B, T) dalam konteks saat ini
        for _ in range(max_new_tokens):
            # pangkas idx ke token block_size terakhir
            idx_cond = idx[:, -block_size:]
            # dapatkan prediksinya
            logits, loss = self(idx_cond)
            # fokus hanya pada langkah terakhir kali
            logits = logits[:, -1, :] # menjadi (B, C)
            # terapkan softmax untuk mendapatkan probabilitas
            probs = F.softmax(logits, dim=-1) # (B, C)
            # sampel dari distribusi
            idx_next = torch.multinomial(probs, num_samples=1) # (B,
1)
            # tambahkan indeks sampel ke urutan yang sedang berjalan
            idx = torch.cat((idx, idx_next), dim=1) # (B, T+1)
        return idx

model = BigramLanguageModel()
m = model.to(device)
# mencetak jumlah parameter dalam model
print(sum(p.numel() for p in m.parameters())/1e6, 'M parameters')

# buat pengoptimal PyTorch
optimizer = torch.optim.AdamW(model.parameters(), lr=learning_rate)

for iter in range(max_iters):
```

```python
    # sesekali mengevaluasi kerugian pada set kereta dan val
    if iter % eval_interval == 0 or iter == max_iters - 1:
        losses = estimate_loss()
        print(f"step {iter}: train loss {losses['train']:.4f}, val
loss {losses['val']:.4f}")

    # sampel kumpulan data
    xb, yb = get_batch('train')

    # evaluasi kerugian
    logits, loss = model(xb, yb)
    optimizer.zero_grad(set_to_none=True)
    loss.backward()
    optimizer.step()

# hasilkan dari model
context = torch.zeros((1, 1), dtype=torch.long, device=device)
print(decode(m.generate(context, max_new_tokens=2000)[0].tolist()))
```

```
0.209729 M parameters
step 0: train loss 4.4116, val loss 4.4022
step 100: train loss 2.6568, val loss 2.6670
step 200: train loss 2.5091, val loss 2.5058
step 300: train loss 2.4197, val loss 2.4336
step 400: train loss 2.3501, val loss 2.3562
step 500: train loss 2.2963, val loss 2.3125
step 600: train loss 2.2407, val loss 2.2496
step 700: train loss 2.2054, val loss 2.2187
step 800: train loss 2.1633, val loss 2.1866
step 900: train loss 2.1241, val loss 2.1504
step 1000: train loss 2.1036, val loss 2.1306
step 1100: train loss 2.0698, val loss 2.1180
step 1200: train loss 2.0380, val loss 2.0791
step 1300: train loss 2.0248, val loss 2.0634
step 1400: train loss 1.9926, val loss 2.0359
step 1500: train loss 1.9697, val loss 2.0287
step 1600: train loss 1.9627, val loss 2.0477
step 1700: train loss 1.9403, val loss 2.0115
step 1800: train loss 1.9090, val loss 1.9941
step 1900: train loss 1.9092, val loss 1.9858
step 2000: train loss 1.8847, val loss 1.9925
step 2100: train loss 1.8724, val loss 1.9757
step 2200: train loss 1.8580, val loss 1.9594
step 2300: train loss 1.8560, val loss 1.9537
step 2400: train loss 1.8412, val loss 1.9427
step 2500: train loss 1.8141, val loss 1.9402
step 2600: train loss 1.8292, val loss 1.9397
step 2700: train loss 1.8116, val loss 1.9322
step 2800: train loss 1.8032, val loss 1.9218
```

```
step 2900: train loss 1.8022, val loss 1.9285
step 3000: train loss 1.7955, val loss 1.9195
step 3100: train loss 1.7672, val loss 1.9192
step 3200: train loss 1.7568, val loss 1.9138
step 3300: train loss 1.7551, val loss 1.9059
step 3400: train loss 1.7549, val loss 1.8945
step 3500: train loss 1.7383, val loss 1.8956
step 3600: train loss 1.7242, val loss 1.8868
step 3700: train loss 1.7273, val loss 1.8822
step 3800: train loss 1.7176, val loss 1.8923
step 3900: train loss 1.7219, val loss 1.8750
step 4000: train loss 1.7131, val loss 1.8603
step 4100: train loss 1.7105, val loss 1.8777
step 4200: train loss 1.7033, val loss 1.8675
step 4300: train loss 1.7038, val loss 1.8556
step 4400: train loss 1.7057, val loss 1.8643
step 4500: train loss 1.6875, val loss 1.8528
step 4600: train loss 1.6887, val loss 1.8405
step 4700: train loss 1.6834, val loss 1.8501
step 4800: train loss 1.6675, val loss 1.8437
step 4900: train loss 1.6684, val loss 1.8407
step 4999: train loss 1.6645, val loss 1.8286


KING RICHARD II:
Shal lifest made to bub, to take Our my dagatants:
Whith foul his vetward that a endrer, my fears' to zorm heavens,
Oof it heart my would but
With ensengmin latest in ov the doest not.

WARWICK:
Welll now, and thus quechiry: there's speak you love.
In Bodiet, and whom the sclittle
Enout-now what evily well most rive with is compon to the me
Town danters, If so;
Ange to shall do aleous, for dear?

KING HENRY VI:
Hark, but a
ards bring Edward?

GROKE:
As is no Rurnts I am you! who neet.
Pom mary thou contrantym so a thense.

QUEEN VINCENTIO:
O, sir, may in God't well ow, whom confessy.
Which migh.

ARCHILINIUS:
```

```
Dithul seaze Peed me: very it passce of's cruport;
How what make you fear tals: there loves
Tunkistren in deed, is xment.

CORIONIUS:
What comforts me. I with self From the walt I?

GRINION:
Which ushold.

KING HENRY Gindner:
Withrief I doot, is onter now.

Securming:
Intande whose no crown some Eiverely marry sold;
For for me watch the
our torguet! Goy, know our her and brut what I, I huself as humsell.

APTOLYCUM:
Laitance and toarth or word
As beherefitions so me worting.

CORIOLINA:
What a wouldds,
An but branedy wouldIng my a canity:
Was you be any in Becausing watcess the Regreast men is what see would
in thas jury your Hrannertandless;
As there'erliacter me band frind through he crown, I she love is stay
just torment:
Slaw you behoth unserving of vonby the post,
Whave baste hold; I they nengety may's fries
To there's fince, I heave arrow old,
Thee best sincess soul be
that Lord, as;
River thou a-latsteer:
Out.

PORALLINA:
Where but
Braight gentle, drieven the know you
for that to this mack a rishn. Prawity arm as is infectely,
Ah, sinstats o' no, this send; commant to love,
Go fly this fathal
I cortuns cold, offrong to old, the courtly thee? before a gace.

KING RICHARD III:
A life he pusict
It. Vitters, and were not fanturs, thy promind thy awonse than a
braute comforn,
Will Roman! you brain shown'd for a dresss me; he heavison!
```

```
MENE
```