Muhammad Fauzan Aldi

1103210049

Machine Learning

Task 1 Build GPT from Scratch

!wget https://raw.githubusercontent.com/karpathy/char-rnn/master/data/tinyshakespeare/input.txt

Mengunduh file dari internet melalui terminal atau command prompt

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

Membaca dari isi file txt

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

Mencetak Panjang dataset dalam satuan karakter

```
print(text[:1000])
```

Mencetak 1000 karakter pertama dari isi dataset

```
chars = sorted(list(set(text)))
vocab_size = len(chars)
print(''.join(chars))
print(vocab_size)
```

Mengidentifikasi karakter-karakter unik dalan teks dan mengurutkannya

```
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]
decode = lambda l: ''.join([itos[i] for i in l])
```

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

encode() **Mengubah string menjadi representasi,** decode() **mengubah representasi numerik kembali menjadi string asli**

```
import torch
data = torch.tensor(encode(text), dtype=torch.long)
print(data.shape, data.dtype)
print(data[:1000])
```

Mengonversikan teks yang telah di-encode menjadi tensot PyTorch dengan tipe data long

```
n = int(0.9*len(data))
train_data = data[:n]
val_data = data[n:]
```

train_data **menyimpan 90% pertama dari data,** val_data **menyimpan 10% terakhir dari data untuk digunakan dalam validasi model**

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

Mengambil blok pertama dari train_data **yang berisi 9 elemen. Blok ini sering dugunakan dalam konteks pelatihan sekuensial**

```
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}")
```

Membuat pasangan inpu-target, membagi x dan y menjadi pasangan input-output untuk setiap Langkah t

```
torch.manual_seed(1337)
```

```
batch_size = 4
block_size = 8

def get_batch(split):
  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):
  for t in range(block_size):
    context = xb[b, :t+1]
    target = yb[b,t]
    print(f"when input is {context.tolist()} the target: {target}")
```

Mengambil batch input (xb) **dan target** (yb) **dari dataset, setiap batch berisi** batch_size **sekuens, masing-masing sepanjang** block_size

```
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__()
        self.token_embedding_table = nn.Embedding(vocab_size, vocab_size)

    def forward(self, idx, targets=None):
        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)  # Flatten untuk cross-entropy
            targets = targets.view(B * T)  # Flatten
            loss = F.cross_entropy(logits, targets)
            return logits, loss
        return logits, loss

    def generate(self, idx, max_new_tokens):
        for _ in range(max_new_tokens):
            logits, loss = self(idx)  # Ambil logits dari forward
            logits = logits[:, -1, :]  # Ambil token terakhir
            probs = F.softmax(logits, dim=-1)  # Konversi ke probabilitas
            idx_next = torch.multinomial(probs, num_samples=1)  # Sampling token berikutnya
            idx = torch.cat((idx, idx_next), dim=1)  # Tambahkan token baru ke sequence
        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()))
```

BigramLanguageModel memprediksi token berikutnya berdasarkan satu token sebelumnya

```
optimizer = torch.optim.AdamW(m.parameters(), lr=1e-3)
```

Menyiapkan optimizer AdamW untuk melatih model dengan parameter

```
batch_size = 32
for steps in range(10000):
    xb, yb = get_batch('train')
    logits, loss = m(xb, yb)
    optimizer.zero_grad(set_to_none=True)
    loss.backward()
    optimizer.step()
```

```
print(loss.item())
```

Menentukan ukuran batch (batch_size) **dan jumlah langkah pelatihan** (10000)

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

Menginisialisasi token awal (0) **untuk memulai generasi teks**

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

Menjamin angka acak yang dihasilkan oleh torch.randn **bersifat deterministic**

```
xbow = torch.zeros((B, T, C))
```

```
for b in range(B):
    for t in range(T):
        xprev = x[b, :t+1]
        xbow[b, t] = torch.mean(xprev, 0)
```

Membuat tensor kosong untuk menyimpan hasil rata-rata kumulatif

```
wei = torch.tril(torch.ones(T, T))
wei = wei / wei.sum(1, keepdim=True)
xbow2 = wei @ x
torch.allclose(xbow, xbow2)
```

Menjadikan setiap baris dalam matriks sebagai distribusi probabilitas

```
#Version 3
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)
```

Memastikan hanya elemen-elemen dibawah dan pada diagonal utama yang digunakan dalam perhatian

```
#Version 4 Self-attention!
torch.manual_seed(1337)
B, T, C = 4, 8, 32
x = torch.randn(B, T, C)

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)
q = query(x)
wei = q @ k.transpose(-2, -1)

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
```

Mengimplementasikan Casual self-attention, yang memungkinkan setiap timestep hanya memperhatikan timestep sebelumnya, digunakan dalam model seperti transformer untuk memproses data sequential seperti teks

```
wei[0]
```

Mengambil dan memeriksa attention weights untuk sequence pertama dalam batch

```
torch.tril(torch.ones(3, 3))
```

Menghindari perhatian terhadap elemen yang berada di atas diagonal

```
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)
```

Membuat matriks segitiga bawah dan menormalisasi baris-barisnya

```
import torch

class 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):
            xmean = x.mean(0, keepdim=True)
            xvar = x.var(0, keepdim=True)
            xhat = (x - xmean) / torch.sqrt(xvar + self.eps)
            self.out = self.gamma * xhat + self.beta
            return self.out

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

# Test
torch.manual_seed(1337)
module = BatchNorm1d(100)
x = torch.randn(32, 100)
```

```
x = module(x)

print(x.shape)
```

Menerapkan normalisasi batch pada pada tensor 2D dengan 32 sampel dan 100 fitur

```
x[:,0].mean(), x[:,0].std()
```

Mengevaluasi sebaran (varians) dan pusat distribusi (rata-rata) dari nilai-nilai pada baris pertama dalam data tensor