NAMA : ARDIUS EBENEZER SIMANJUNTAK

NIM : 1103210802

KELAS : TK-45-G04

Laporan Tugas Besar Machine Learning

BUILD GPT FROM SCRATCH

Oleh :

ARDIUS EBENEZER SIMANJUNTAK/1103210208

PRODI S1 TEKNIK KOMPUTER

FAKULTAS TEKNIK ELEKTRO

UNIVERSITAS TELKOM

BANDUNG

2024

1. Latar Belakang

Transformer merevolusi dunia kecerdasan buatan. Arsitektur jaringan saraf yang kuat ini, diperkenalkan pada tahun 2017, dengan cepat menjadi pilihan utama untuk pemrosesan bahasa alami, AI generatif, dan banyak lagi. Dengan bantuan transformator, kami telah melihat penciptaan produk AI mutakhir seperti BERT, GPT-x, DALL-E, dan AlphaFold, yang mengubah cara kita berinteraksi dengan bahasa dan memecahkan masalah kompleks seperti pelipatan protein. Dan kemungkinan menarik tidak berhenti di situ - transformator juga membuat gelombang di bidang visi komputer dengan munculnya Vision Transformers.

Sebelum Transformers, dunia AI didominasi oleh jaringan saraf konvolusional untuk visi komputer (VGGNet, AlexNet), jaringan saraf berulang untuk pemrosesan bahasa alami dan masalah pengurutan (LSTM, GRU) dan Jaringan Permusuhan Generatif (GAN) untuk AI generatif. Pada 2017, CNN dan RNN masing-masing menyumbang 10,2% dan 29,4% dari makalah yang diterbitkan tentang pengenalan pola. Namun sudah pada tahun 2021 para peneliti Stanford menyebut transformator "model fondasi" karena mereka melihatnya mendorong perubahan paradigma dalam AI.

Manfaat lain yang dimiliki transformator dengan RNN adalah kemampuan untuk menangkap dependensi antar elemen dalam urutan. Tidak seperti CNN, yang memproses input dalam jendela dengan panjang tetap, transformator menggunakan mekanisme perhatian diri yang memungkinkan model untuk secara langsung menghubungkan elemen input yang berbeda satu sama lain, terlepas dari jaraknya dalam urutan. Hal ini memungkinkan transformator untuk menangkap dependensi jarak jauh antara elemen input, yang sangat penting untuk tugas-tugas seperti pemrosesan bahasa alami di mana perlu mempertimbangkan konteks kata agar dapat memahaminya secara akurat.

Andrej Karpathy telah merangkum kekuatan Transformers sebagai komputer tujuan umum yang dapat dibedakan. Ini sangat kuat dalam umpan maju, karena dapat mengekspresikan komputasi yang sangat umum. Node menyimpan vektor dan node ini saling memandang dan mereka dapat melihat apa yang penting untuk komputasi mereka di node lain.

2. Analisis Kodingan

We always start with a dataset to train on. Let's download the tiny shakespeare dataset !wget https://raw.githubusercontent.com/karpathy/char-rnn/master/data/tinyshakespeare/input.txt
Analisis: **Pengambilan datasheet**. Kode ini menggunakan perintah wget untuk mengunduh dataset tiny Shakespeare dari repositori GitHub Andrej Karpathy. Dataset ini sering digunakan untuk eksperimen pembelajaran mesin dalam pemrosesan teks. Tanda menunjukkan bahwa ini adalah perintah shell yang dijalankan dari dalam notebook Python
with open('input.txt', 'r', encoding='utf-8') as f: text = f.read()
Analisis: Pembacaan File. Menggunakan context manager () untuk membuka file secara aman. Parameter 'r' menandakan mode baca-saja. Encoding 'utf-8' memastikan teks dapat dibaca dengan benar termasuk karakter khusus dengan isi file disimpan dalam variabel
print("length of dataset in characters: ", len(text)) print(text[:1000])
Analisis: Analisis Data Awal. Menampilkan panjang total dataset dalam jumlah karakter menggunakan **len(text).** Menampilkan 1000 karakter pertama dari dataset menggunakan slicing **text[:1000]** . Ini adalah langkah eksplorasi data awal yang penting untuk memahami struktur data.
Inspects first 1000 characters of text print(text[:1000]) # Creates character-level vocabulary chars = sorted(list(set(text))) vocab_size = len(chars) # Character-integer mapping stoi = { ch:i for i,ch in enumerate(chars) } itos = { i:ch for i,ch in enumerate(chars) }
Analisis: Pemrosesan Data & Tokenisasi. Dengan bagian ini menetapkan prapemrosesan dasar, menciptakan sistem tokenisasi tingkat karakter.
data = torch.tensor(encode(text), dtype=torch.long) n = int(0.9*len(data)) train_data = data[:n] val_data = data[n:]
Analisis: Pengkodean Data & Pembuatan Dataset. Kode mengonversi teks menjadi tensor dan membagi menjadi set pelatihan (90%) dan validasi (10%).
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
```

Analisis: Sistem Generasi Batch. Menerapkan pembuatan batch yang efisien untuk pelatihan.

```
class BigramLanguageModel(nn.Module):
    def __init__(self, vocab_size):
        super().__init__()
        self.token_embedding_table = nn.Embedding(vocab_size, vocab_size)
```

Analisis: Implementasi BigramLanguageModel. Mendefinisikan model jaringan saraf untuk pemodelan bahasa dengan Penyematan token, Logika umpan maju dan Kemampuan pembuatan teks.

```
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)
```

Analisis: Mekanisme Perhatian Diri. Mengimplementasikan perhatian dot-product berskala dengan Transformasi kueri, Kunci, dan Nilai. Perhitungan skor perhatian serta Masking untuk perhatian kausal.

```
optimizer = torch.optim.AdamW(m.parameters(), lr=1e-3)
for steps in range(100):
    xb, yb = get_batch('train')
    logits, loss = m(xb, yb)
    optimizer.zero_grad(set_to_none=True)
    loss.backward()
    optimizer.step()
```

Analisis: Lingkaran Pelatihan, Menangani pelatihan model menggunakan Pengoptimal AdamW,pemrosesan batch serta Komputasi kerugian dan propagasi balik.
Fitur utama yang digunakan pada step tersebut:

1. Menggunakan tokenisasi tingkat karakter
2. Menerapkan perhatian diri kausal (bertopeng)
3. Menggunakan pemrosesan batch untuk pelatihan yang efisien
4. Termasuk kemampuan pembuatan teks
5. Menggunakan kehilangan entropi silang untuk pelatihan
6. Mengimplementasikan beberapa versi komputasi perhatian.

```
class Head(nn.Module):
    def __init__(self, head_size):
        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)
```

Analisis : Implementasi attention dibagi menjadi beberapa versi. Menggunakan query, key, dan value transformations.Menerapkan masked attention untuk prediksi kausal.Menggunakan multiple attention heads secara paralel.

3. Kesimpulan :

Aspek regularisasi tidak luput dari perhatian. Implementasi dropout yang strategis pada beberapa titik kunci dalam arsitektur membantu mencegah overfitting. Yang menarik adalah bagaimana Karpathy menyesuaikan tingkat dropout berdasarkan ukuran model, menunjukkan pemahaman mendalam tentang scaling laws dalam model bahasa.

Sistem evaluasi yang diimplementasikan mencakup berbagai metrik performa. Karpathy tidak hanya fokus pada loss function standar, tetapi juga mengimplementasikan mekanisme untuk mengukur perplexity dan sampling efficiency. Sistem evaluasi yang komprehensif ini

Implementasi kode Karpathy mendemonstrasikan keseimbangan yang baik antara keterbacaan dan efisiensi. Meskipun ditujukan sebagai implementasi edukatif, kode tersebut tetap mempertahankan praktik-praktik engineering yang solid dan optimisasi yang cerdas. Hal ini membuat implementasinya tidak hanya berharga sebagai alat pembelajaran, tetapi juga sebagai referensi untuk implementasi praktis.