Đánh Giá Hiệu Quả của RoPE, SwiGLU và Optimizer AdamW trong Mô Hình Dịch Máy

Hoàng Duy Hưng, Đồng Tự Nguyên A, Nguyễn Hồ Bắc

Môn học: Xử lý ngôn ngữ tự nhiên

Mục tiêu của nghiên cứu này là đánh giá hiệu quả của các kỹ thuật Rotary Positional Embedding (RoPE), SwiGLU FeedForward, và Optimizer AdamW với các tham số khác nhau, bao gồm cả việc thay đổi kiến trúc nền tảng sang Mamba, nhằm cải thiện hiệu suất của mô hình dịch máy. Các thử nghiệm cho thấy RoPE mang lại cải thiện đáng kể và ổn định cho điểm BLEU. SwiGLU cho thấy tiềm năng ban đầu nhưng chưa vượt trội RoPE-only trong khung thời gian huấn luyện hiện tại. Việc chuyển sang optimizer AdamW và kiến trúc Mamba với các learning rate và weight decay cao đã dẫn đến kết quả không ổn định: mặc dù đạt được điểm BLEU cao nhất trong một epoch cụ thể, mô hình nhanh chóng bị overfitting hoặc sụp đổ ở các epoch sau. Điều này nhấn mạnh sự cần thiết của việc lựa chọn cẩn thận các siêu tham số, đặc biệt là learning rate, và việc sử dụng learning rate scheduler.

***Index Terms*—RoPE, SwiGLU, AdamW, Mamba, Dịch máy, BLEU**

I. Giới thiệu

Nghiên cứu này tập trung vào việc đánh giá các kỹ thuật hiện đại nhằm cải thiện hiệu suất của mô hình dịch máy, bao gồm Rotary Positional Embedding (RoPE), SwiGLU FeedForward, và optimizer AdamW, đồng thời khám phá kiến trúc Mamba như một sự thay thế cho Transformer truyền thống. Các thử nghiệm được thiết kế để xác định tác động của từng thành phần và sự kết hợp của chúng đối với hiệu suất mô hình, được đo bằng điểm BLEU và Validation Loss.

II. Phần 1: Chi tiết các thử nghiệm

A. Thử nghiệm 1: Tích hợp Rotary Positional Embedding (RoPE)

1) Mô tả thay đổi:

- Sửa đổi file `layers/prototypes.py`.
- Thêm hàm `apply_rope` để áp dụng mã hóa vị trí tương đối RoPE cho tensor query (Q) và key (K).
- Cập nhật hàm `forward` của lớp `MultiHeadAttention` để gọi `apply_rope` cho Q và K trước khi tính toán attention.
- Lưu ý về shape tensor: Mã nguồn `apply_rope` được điều chỉnh để xử lý shape `[batch_size, heads, seq_len, d_k]` phù hợp với output của các lớp linear và phép transpose trong `MultiHeadAttention`.

2) Mã nguồn thay đổi:

```
# Trong layers/prototypes.py
import torch
import torch.nn as nn

def apply_rope(x):
    """
    x: Tensor of shape [batch_size, heads, seq_len,
        d_k]
    Return: RoPE-applied tensor of the same shape
    """
    batch, n_heads, seq_len, d_k = x.shape
    half = d_k // 2
    freq_seq = torch.arange(half, device=x.device,
        dtype=torch.float32)
    freq_seq = 10000.0 ** (-freq_seq / half)
    pos = torch.arange(seq_len, device=x.device,
        dtype=torch.float32)
    angles = torch.einsum('i,j->ij', pos, freq_seq)
        # [seq_len, half]

    sin = torch.sin(angles)[None, None, :, :] # [1,
        1, seq_len, half]
    cos = torch.cos(angles)[None, None, :, :] # [1,
        1, seq_len, half]

    x1 = x[..., :half]
    x2 = x[..., half:]
    x_rope = torch.cat([x1 * cos - x2 * sin, x1 *
        sin + x2 * cos], dim=-1)
    return x_rope

class MultiHeadAttention(nn.Module):
    # ... (__init__ kh ng    i   ) ...
    def forward(self, q, k, v, mask=None):
        bs = q.shape[0]
        q = self.q_linear(q).view(bs, -1, self.h,
            self.d_k).transpose(1, 2)
        k = self.k_linear(k).view(bs, -1, self.h,
            self.d_k).transpose(1, 2)
        v = self.v_linear(v).view(bs, -1, self.h,
            self.d_k).transpose(1, 2)

        q = apply_rope(q)
        k = apply_rope(k)

        value, attn = self.attention(q, k, v, mask,
            self.dropout)
        concat = value.transpose(1, 2).contiguous().
            view(bs, -1, self.d_model)
        output = self.out(concat)
        return output, attn
```

3) Kết quả huấn luyện:

4) Phân tích: Việc tích hợp RoPE cho thấy hiệu quả rõ rệt. BLEU score tăng trưởng đều đặn qua các epoch, từ 0.0244 lên đến đỉnh điểm 0.2582 ở epoch 8 và duy trì ở epoch 9. Validation Loss giảm ổn định, cho thấy mô hình học tốt và

Bảng I
KẾT QUẢ HUẤN LUYỆN VỚI ROPE

Epoch	Validation Loss	BLEU Score	Evaluation Time (s)
0	4.8158	0.0244	167.4600
1	3.9094	0.0874	348.2373
2	3.4099	0.1811	188.2426
3	3.2150	0.2128	166.5316
4	3.1083	0.2293	198.6460
5	3.0533	0.2403	168.7870
6	3.0048	0.2412	242.8171
7	2.9644	0.2525	205.3884
8	2.9447	0.2582	195.0013
9	2.9403	0.2582	170.4748

không có dấu hiệu overfitting rõ ràng trong 10 epoch này.

B. Thử nghiệm 2: Tích hợp SwiGLU FeedForward

1) Mô tả thay đổi:

- Tiếp tục sửa file `layers/prototypes.py`.
- Thay thế lớp `FeedForward` ban đầu bằng một triển khai sử dụng hàm kích hoạt SwiGLU. Kiến trúc RoPE từ Thử nghiệm 1 được giữ nguyên.

2) Mã nguồn thay đổi:

```
# Trong layers/prototypes.py
import torch.nn.functional as functional

class FeedForward(nn.Module):
    """SwiGLU FeedForward layer: Linear -> SwiGLU ->
        Dropout -> Linear"""
    def __init__(self, d_model, d_ff=2048, dropout
        =0.1):
        super().__init__()
        self.linear_1 = nn.Linear(d_model, d_ff * 2)
             # T ng hidden_dim cho SwiGLU
        self.dropout = nn.Dropout(dropout)
        self.linear_2 = nn.Linear(d_ff, d_model)

    def forward(self, x):
        x_proj = self.linear_1(x)
        x1, x2 = x_proj.chunk(2, dim=-1)
        x = x1 * functional.silu(x2) # SwiGLU (SiLU
            l   Swish-1)
        x = self.linear_2(self.dropout(x))
        return x
```

Bảng II
KẾT QUẢ HUẤN LUYỆN VỚI ROPE + SWIGLU

Epoch	Valid Loss	BLEU Score	Evaluation Time (s)
0	4.7586	0.0276	160.20
1	3.8660	0.0962	289.96
2	3.3489	0.1856	236.54
3	3.1285	0.2182	252.69
4	3.0303	0.2364	205.01
5	2.9720	0.2474	248.38
6	2.9382	0.2536	218.68
7	2.9325	0-fits 8	2.9225
0.2570	182.19		
9	2.9211	0.2576	210.34

3) Kết quả huấn luyện:

4) Phân tích: So với Thử nghiệm 1, việc thêm SwiGLU giúp BLEU score khởi điểm cao hơn một chút. Tuy nhiên, BLEU score cao nhất đạt được (0.2576 tại epoch 9) thấp hơn không đáng kể so với khi chỉ dùng RoPE (0.2582).

C. Thử nghiệm 3: Optimizer AdamW + Kiến trúc Mamba

1) Mô tả thay đổi:

- Thay đổi kiến trúc lõi từ Transformer sang Mamba.
- Thay đổi optimizer sang AdamW với `learning_rate = 0.2` và `weight_decay = 0.01`.
- Các thành phần RoPE và SwiGLU từ các thử nghiệm trước vẫn được giữ nguyên.

Bảng III
KẾT QUẢ HUẤN LUYỆN VỚI MAMBA + ADAMW (LR=0.2, WD=0.01)

Epoch	Iter	Train Loss	Valid Loss	BLEU Score	Eval Time (s)
0	1334	2.71	5.8888	0.0000	89.0575
1	1334	2.25	3.6964	0.2649	604.2571
2	1334	1.99	2.9962	0.2052	573.2489
3	1334	1.77	2.8031	0.1108	1212.6881
4	1334	1.62	2.7527	0.1167	1236.4381
5	1334	1.53	2.7920	0.1273	1267.5303
6	1334	1.45	2.7897	0.1626	1254.6651
7	1334	1.36	2.8800	0.1235	1241.5605
8	1334	1.32	2.7918	0.1512	1262.8262
9	1200	1.26	N/A	N/A	N/A

2) Kết quả huấn luyện:

3) Phân tích: BLEU score ở epoch 1 đạt 0.2649, cao hơn đáng kể so với các thử nghiệm trên kiến trúc Transformer. Tuy nhiên, từ epoch 2 trở đi, BLEU score giảm mạnh, cho thấy dấu hiệu overfitting.

D. Thử nghiệm 4: Điều chỉnh learning_rate và weight_decay

1) Mô tả thay đổi:

- Tiếp tục sử dụng kiến trúc Mamba và optimizer AdamW.
- Điều chỉnh `learning_rate` thành 0.19.
- Tăng `weight_decay` lên 0.1.

Bảng IV
KẾT QUẢ HUẤN LUYỆN VỚI MAMBA + ADAMW (LR=0.19, WD=0.1)

Epoch	Iter	Train Loss	Valid Loss	BLEU Score	Eval Time (s)
0	1334	2.72	6.1026	0.0000	63.7861
1	1334	2.28	3.2490	0.2795	277.9353
2	1334	2.00	3.8590	0.0066	58.3072
3	1334	1.79	2.9430	0.1179	1196.5277

2) Kết quả huấn luyện:

3) Phân tích: Epoch 1 đạt BLEU score cao nhất trong tất cả các thử nghiệm: 0.2795. Tuy nhiên, ở epoch 2, BLEU score tụt dốc thảm hại xuống 0.0066, cho thấy sự không ổn định nghiêm trọng của mô hình.

III. PHẦN 2: TỔNG KẾT VÀ THẢO LUẬN CHUNG

1) **RoPE trên kiến trúc Transformer**: Chứng minh được hiệu quả rõ rệt và ổn định, cải thiện đáng kể BLEU score (đạt 0.2582).
2) **SwiGLU trên kiến trúc Transformer**: Cho thấy tiềm năng cải thiện ở các epoch đầu nhưng BLEU score cuối cùng (0.2576) không vượt qua được cấu hình RoPE-only.
3) **Chuyển sang kiến trúc Mamba và Optimizer AdamW**: Đạt BLEU cao nhất (0.2795) nhưng mô hình cực kỳ không ổn định.

4) **Vấn đề chung**: Overfitting, learning rate cao, và thiếu learning rate scheduler là các nguyên nhân chính gây bất ổn.

IV. Phần 3: Đề xuất hướng tiếp theo

1) **Kiến trúc cơ sở và Positional Encoding**: Giữ lại RoPE cho Transformer; nghiên cứu tích hợp positional encoding cho Mamba.
2) **SwiGLU**: Thử nghiệm thêm với các giá trị `d_ff` khác nhau.
3) **Optimizer và Learning Rate**: Sử dụng learning rate thấp hơn và áp dụng learning rate scheduler (warmup, decay).
4) **Chiến lược cho Mamba**: Tìm bộ siêu tham số tối ưu riêng cho Mamba.
5) **Kiểm tra lại quá trình đánh giá**: Đảm bảo tính nhất quán của BLEU score và điều tra biến động thời gian đánh giá.