Memory-Efficient Transformer Training Techniques
Lingwistyka Obliczeniowa | Laboratorium 4

Wojciech Bartoszek

1 Cel i zakres zadania

Celem laboratorium było porównanie nowoczesnych technik optymalizacji pamięci podczas trenowania modeli Transformer. Dla identycznego zbioru danych, architektury modelu i hiperparametrów (poza ustawieniami związanymi z pamięcią) przeprowadzono eksperymenty mierzące:

- Zużycie pamięci GPU
- Maksymalny rozmiar batcha mieszczący się w pamięci
- Szybkość trenowania (czas na krok i całkowity czas dla 1 epoki)
- Końcową jakość modelu (perplexity po 1 epoce)

2 Przegląd technik optymalizacji

2.1 Baseline (TF32/FP32)

Trening w pełnej precyzji z wykorzystaniem TF32 dla operacji macierzowych. Służy jako punkt odniesienia dla pozostałych technik.

2.2 BF16 Mixed Precision

Automatyczne mieszanie precyzji (Automatic Mixed Precision) z użyciem typu BFloat16. Wagi i aktywacje są przechowywane w 16-bitowym formacie, co zmniejsza zapotrzebowanie na pamięć o około 50%.

2.3 FlashAttention

FlashAttention 2 to zoptymalizowana implementacja mechanizmu uwagi, która:

- Unika materializacji pełnej macierzy uwagi $O(n^2)$
- Wykorzystuje tiling dla optymalnego wykorzystania cache GPU
- Redukuje złożoność pamięciową do $O(n)$

2.4 Windowed (Local) Attention

Uwaga okienkowa ogranicza zakres uwagi do lokalnego kontekstu (sliding window). Zamiast uwagi do wszystkich tokenów, każdy token może zwracać uwagę tylko na w poprzednich tokenów, gdzie w to rozmiar okna.

2.5 Gradient Checkpointing

Gradient checkpointing (activation checkpointing) to technika wymiany obliczenia na pamięć. Zamiast przechowywać wszystkie aktywacje pośrednie podczas przejścia w przód, są one przeliczane podczas przejścia wstecznego.

3 Konfiguracja eksperymentalna

3.1 Zbiór danych

- Zbiór: `mikex86/stackoverflow-posts`
- Pole: Body (treść postów)
- Tryb: Streaming
- Zadanie: Modelowanie języka (next-token prediction)

3.2 Architektura modelu

Decoder-only Transformer z następującymi parametrami:

- Wymiar embeddings: 256
- Liczba głów uwagi: 8
- Liczba warstw: 4
- Wymiar FFN: 1024
- Maksymalna długość sekwencji: 512
- Dropout: 0.1
- Tokenizer: GPT-2 (vocab_size = 50257)

3.3 Hiperparametry treningu

- Kroki na epokę: 200
- Długość sekwencji: 256
- Learning rate: 3×10^{-4}
- Warmup steps: 200
- Gradient clipping: 1.0
- Optymalizator: AdamW

3.4 Środowisko

Eksperymenty przeprowadzono w następującym środowisku sprzętowo-programowym:

- **PyTorch:** 2.8.0+cu128
- **CUDA:** 12.8
- **GPU:** NVIDIA GeForce RTX 4090
- **GPU Memory:** 25.26 GB

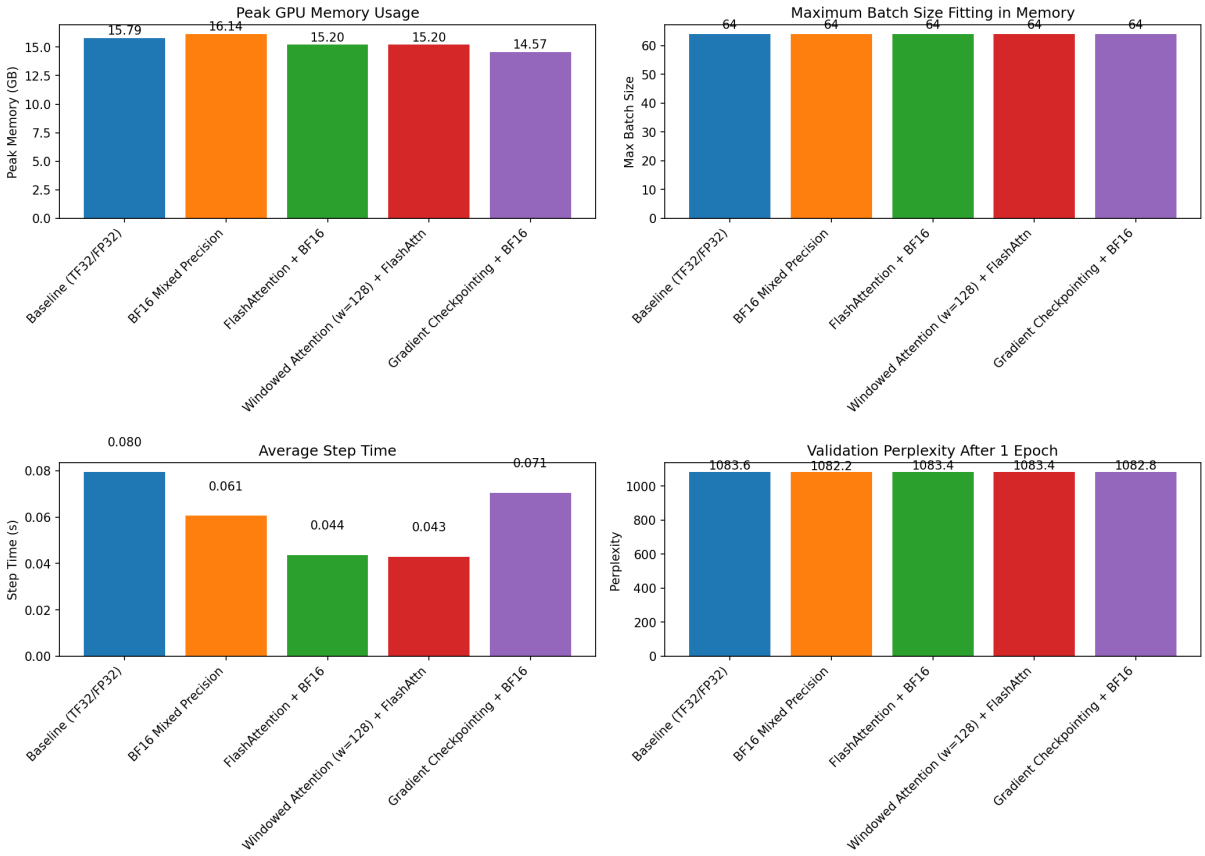
4 Wyniki eksperymentów

Wyniki przeprowadzonych eksperymentów przedstawiono w Tabeli 1. Porównano w niej maksymalny rozmiar batcha, zużycie pamięci, czas treningu oraz końcową jakość modelu (perplexity na zbiorze walidacyjnym).

Tabela 1: Porównanie technik optymalizacji treningu Transformerów

Metoda	Max Batch	Peak Memory (GB)	Step Time (s)	Total Time (s)	Val PPL
Baseline (TF32/FP32)	64	15.79	0.080	16.06	1083.63
BF16 Mixed Precision	64	16.14	0.061	12.14	1082.19
FlashAttention + BF16	64	15.20	0.044	8.89	1083.40
Windowed Attn + FlashAttn	64	15.20	0.043	8.63	1083.40
Grad Checkpointing + BF16	64	14.57	0.071	14.14	1082.79

Poniżej przedstawiono wykresy porównujące poszczególne metryki dla badanych metod.



Rysunek 1: Porównanie zużycia pamięci, czasu treningu i maksymalnego rozmiaru batcha.

5 Analiza wyników i wnioski

5.1 Zużycie pamięci

Najmniejsze zużycie pamięci szczytowej (Peak Memory) zaobserwowano dla techniki **Gradient Checkpointing** (14.57 GB). Jest to zgodne z oczekiwaniami, ponieważ technika ta nie przechowuje wszystkich aktywacji pośrednich, lecz przelicza je w razie potrzeby. **FlashAttention** oraz **Windowed Attention** również wykazały mniejsze zużycie pamięci (15.20 GB) w porównaniu

do Baseline (15.79 GB). Co ciekawe, samo użycie **BF16 Mixed Precision** w tym eksperymencie wykazało nieco wyższe zużycie pamięci szczytowej (16.14 GB) niż Baseline. Może to wynikać z narzutu związanego z przechowywaniem kopii wag w FP32 (dla optymalizatora) oraz specyfiki alokatora pamięci PyTorch w tym konkretnym przypadku.

5.2 Szybkość treningu

Najszybszymi metodami okazały się **FlashAttention** oraz **Windowed Attention**, osiągając czas kroku na poziomie ok. 0.043-0.044 s, co stanowi niemal dwukrotne przyspieszenie względem Baseline (0.080 s). Wynika to z efektywnego wykorzystania pamięci cache GPU i redukcji transferów pamięci. **BF16 Mixed Precision** również przyspieszyło trening (0.061 s/krok) dzięki mniejszej precyzji obliczeń. **Gradient Checkpointing** (0.071 s/krok) był wolniejszy niż czyste BF16 czy FlashAttention, co jest kosztem ponoszonym za oszczędność pamięci – konieczność ponownego obliczania aktywacji w fazie backward wydłuża czas obliczeń. Mimo to, nadal był szybszy od Baseline.

5.3 Jakość modelu

Wszystkie metody osiągnęły bardzo zbliżone wyniki Perplexity na zbiorze walidacyjnym (ok. 1082-1083). Oznacza to, że zastosowane optymalizacje (obniżona precyzja BF16, aproksymacje w FlashAttention, czy gradient checkpointing) nie wpływają negatywnie na proces uczenia i jakość końcowego modelu w tym zadaniu.

5.4 Podsumowanie

- **FlashAttention** oferuje najlepszy kompromis między szybkością a zużyciem pamięci, znacząco przyspieszając trening przy jednoczesnej redukcji zapotrzebowania na VRAM.
- **Gradient Checkpointing** jest najlepszym wyborem, gdy priorytetem jest minimalizacja zużycia pamięci (np. aby zmieścić większy model), nawet kosztem nieco dłuższego czasu treningu.
- **BF16** samo w sobie przyspiesza obliczenia, ale w połączeniu z innymi technikami (jak FlashAttention) daje najlepsze rezultaty.
- Wszystkie techniki są bezpieczne z punktu widzenia jakości modelu (brak degradacji wyników).