ABEJA

Megatron-LMとGKEで作る Mixtral 8x7Bの継続事前学習

2024/05/09

中西 健太郎

目次

- Mixtral 8x7Bの語彙拡張版継続事前学習を実施
- 学習用データは合計450B token（語彙拡張tokenizer）を利用。大体の日本語：英語：コード比率 = 3：1.3：0.4
- Mixtral 8x7BをMegatron-LMに実装。load balancing lossまでMegatronで実装したのは観測できる限りでは世界初
- ハイパラのお手本がなく、ロスNaN問題に苦しんだ
- 学習用のノード並列環境はGoogle Kubernetes Engineを利用。学習開始を可能な限り自動化
- ノード間のNCCL エラー（タイムアウト）に苦戦。NCCL系で怪しい時はノード不良を疑った方が良い

学習コードやデータ・モデルの概要について、既にブログにて公開しています！

Megatron-LMとGKEで作るMixtral 8x7Bを語彙拡張継続事前学習 Part1 ~学習コードとモデルの先行公開~
https://tech-blog.abeja.asia/entry/abeja-nedo-project-part1-202404

ABEJA

目次

使用データー覧

- Common Crawl 日本語 約260B token
  - Common Crawlは2019~2023のtimestampを利用
  - 語彙拡張前のtokenizerでは約400B token
- 英語、ソースコード 約126B token
- 独自収集の日本語データ 約30B token

前処理

- Cloud Run Jobs を数万台レベルで走らせて並列処理
- テキスト抽出、言語判定、フィルタリング、重複削除、etc.

データセット・前処理の詳細についてもブログにて公開しています！
https://tech-blog.abeja.asia/entry/abeja-nedo-project-part2-202405

Mixtral 8x7B

- Mistral AI が2023年末に発表した OSS の MoE モデル
- Megatorn-LMに実装（実はMegatron-Deepspeedにも実装済み）
- 学習速度の観点で本番学習はMegatron-LMを使用
- 日本語生成の効率化のため語彙拡張を実施
- 開発中に Swallow MX や Mixtral 8x22B, Llama3 が公開

Mixtral of Experts

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, William El Sayed

Abstract

We introduce Mixtral 8x7B, a Sparse Mixture of Experts (SMoE) language model. Mixtral has the same architecture as Mistral 7B, with the difference that each layer is composed of 8 feedforward blocks (i.e. experts). For every token, at each layer, a router network selects two experts to process the current state and combine their outputs. Even though each token only sees two experts, the selected experts can be different at each timestep. As a result, each token has access to 47B parameters, but only uses 13B active parameters during inference. Mixtral was trained with a context size of 32k tokens and it outperforms or matches Llama 2 70B and GPT-3.5 across all evaluated benchmarks. In particular, Mixtral vastly outperforms Llama 2 70B on mathematics, code generation, and multilingual benchmarks. We also provide a model fine-tuned to follow instructions, Mixtral 8x7B – Instruct, that surpasses GPT-3.5 Turbo, Claude-2.1, Gemini Pro, and Llama 2 70B – chat model on human benchmarks. Both the base and instruct models are released under the Apache 2.0 license.

Code: https://github.com/mistralai/mistral-src
Webpage: https://mistral.ai/news/mixtral-of-experts/

Ubuntu on GCE で開発・検証

- COS で開発するのは現実的ではないので、データやモデル、分散学習の動作検証などは Ubuntu で実施

GKE on COS で本学習

- 初めは Kubernetes のNW周りの干渉が嫌で GCE x COS を考えたが、COSを少し触り始めた段階で辞めた
  - セキュリティの関係でファイルシステムがステートレス過ぎて再起動したら諸々設定していたものが飛ぶ。諸々工夫するより、Kubernetesのdeamonsetで流した方が早くない？
  - 複数台にデプロイする方法どうしよう。GCEにIAPでSSH相当するのも面倒だし、Kubernetesでyamlで一発デプロイの方が早くない？
- 面倒になって GKE に移行
  - GPUDirect-TCPX とマルチネットワーキングで GPU ネットワーク帯域幅を最大にする
  - 上記を参考に nccl-tests の実行まではサクッと作れた。複数台のデプロイも簡単に実現できた & ホストネットワーク使えば NW 問題も解決し、ひとまずこれで行こうとなる（が、このあと色々詰まる）
- GKE 周りの yaml は学習コードと一緒に公開済み

目次

Huggingface <-> Megatron-LM 間のモデルの変換

- Mixtral からの継続事前学習なので、 Huggingface 形式の重みを Megatron-LM 形式に変換する必要がある
  - 逆も然りで、使うときは Megatron-LM 形式から Huggingface 形式に戻す
- パイプラインで分割されるモデル構造を完全理解する必要があり苦労した
  - 特に Grouped Query Attention と Tensor Parallel の組み合わせに苦戦した
  - qqqqqqqqkkvvではなく、qqqqkvqqqqkvです

Huggingface Mixtral の attention

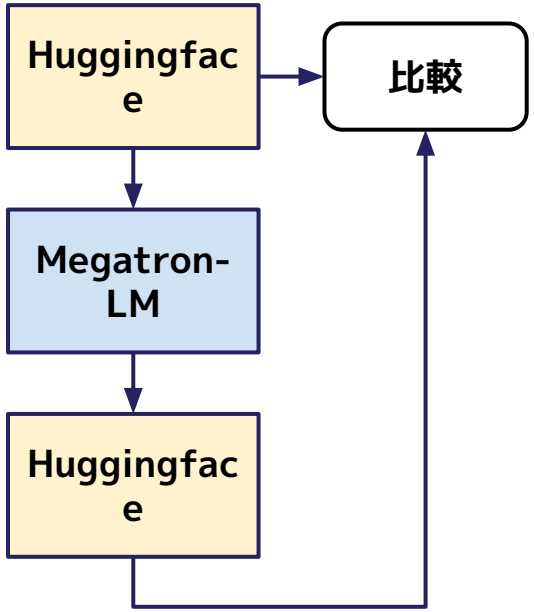
```
self.q_proj = nn.Linear(
self.k_proj = nn.Linear(
self.v_proj = nn.Linear(
self.o_proj = nn.Linear(
```

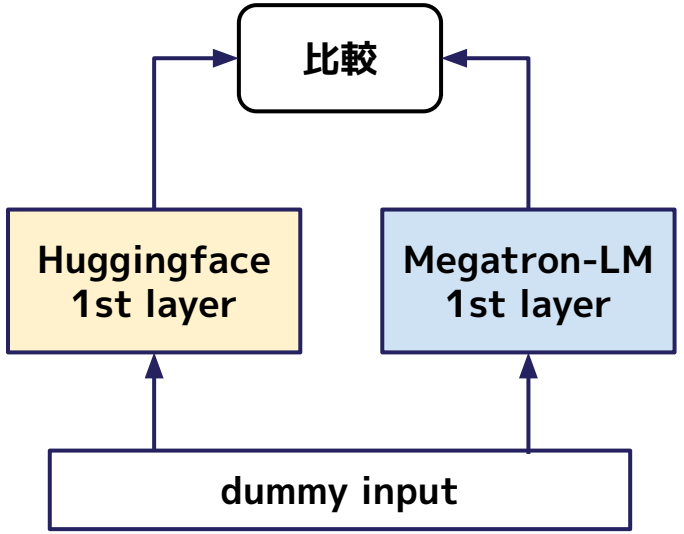
Megatron の attention

```
(query_layer,
key_layer,
value_layer) = torch.split(
    mixed_x_layer,
    [
        (
            self.num_attention_heads_per_partition // self.num_query_groups_per_partition
            * self.hidden_size_per_attention_head
        ),
        self.hidden_size_per_attention_head,
        self.hidden_size_per_attention_head
    ],
    dim=3)
```

モデル変換のデバッグ

- 元の Huggingface モデルと Megatron 形式に変換後再び Huggingface 形式に戻したモデルの重みにズレがないか比較
  - モデル変換にバグがないかの確認
- Megatron 形式に変換後、適当な入力に対して forward を走らせて1層目の出力を Huggingface モデルと比較
  - 変換した Megatron 形式のモデルパラメータと処理があっているかの確認
  - 前述した Grouped Query Attention 部分が怪しかったため入念に行った





MoE 対応

- Megatron にあがっていた PR のコードを参考に作成、基本は FFN 部分に gate と experts を足すだけ

```
self.gate = torch.nn.Linear(self.hidden_dim, self.num_experts, bias=False)

self.experts = torch.nn.ModuleList(
    [MixtralParallelMLP(config) for _ in range(self.num_experts)]
)
```

load balanced loss 対応

- sinkhorn などのアルゴリズムも試しつつ、普通に LB loss のみを適用
- tuple で loss を返すと処理が大変なので、既存の loss とくっつけて処理

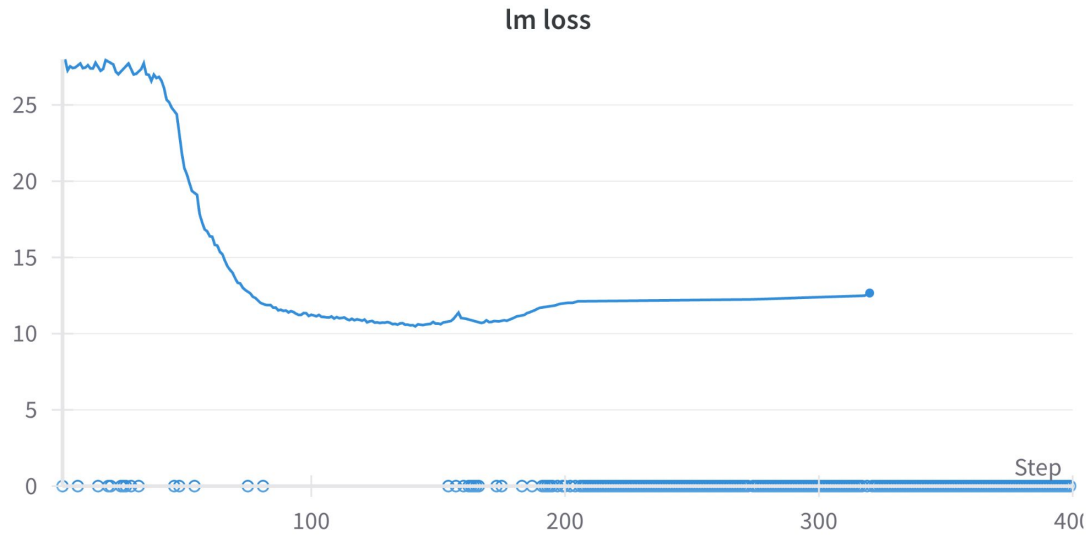
```
if all_router_logits is not None and len(all_router_logits) > 0:
    aux_loss = torch.broadcast_to(aux_loss, loss.shape)
    return torch.stack([loss, aux_loss], 0)
```

```
if args.moe_type == "mixtral" and output_tensor.shape[0] == 2:
    # 1st element is the lm loss + load balanced loss, 2nd element is the
    output_tensor, aux_loss_tensor = torch.chunk(output_tensor, 2, dim=0)
```

未対応のものも

- Expert Parallel, Rolling Buffer Cache, SWA だと GPU 効率が大きく下がる現象

学習後徐々に Loss が nan になる



- 学習開始後 loss に nan が発生し、徐々に増える現象
- Loss の学習曲線も最初は順調だが途中から単調増加
- **継続事前学習ではなくスクラッチだと発生せず**、ハイパーパラメータが原因と考えて試行錯誤
- **fp16 -> bf16 にすることで解決**。

GPUDirect-TCPX はまったポイント

- GPUDirect-TCPX とマルチネットワーキングで GPU ネットワーク帯域幅を最大にする の nccl-tests は動くが、僕らのコンテナは動かない。
- どうも公式手順の「Pod で GPUDirect-TCPX を使用するために Kubernetes マニフェストに追加する必要がある必須フィールド」だけだと**GPUDirect-TCPX/NCCLの環境変数が20個ほど足りず**に、公式手順にある nccl-tests の yaml の中身を見に行ってConfigMapを参考に環境変数を設定すると動き始めた（詳しくはSlackでご質問ください）
- 公式手順のイメージ指定先が間違えてて動かないとかそういうちょいミスも多々あります。該当のDockerレジストリに行ってそれぽいやつ探すとか必要
- 何かしら公式系のテストでも動く環境を作って、それをトレースしにいくことが最短

目次

後続事業者の方にお伝えしたいこと

- 学習が始まると祈りの時間が増えます
  - 弊社はあまり祈れずにトラブル続きでした・・・
- 環境系のハマりポイントはある程度踏んでいると思うので、ぜひお気軽にご質問ください！

ABEJA