Preferred Networks 2022年度インターン選考 テーマ別課題

JE01. 深層学習モデルを社会実装するためのフレームワーク、ライブラリ開発

課題について

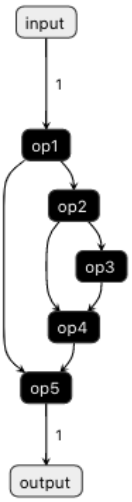
原則として、PFVM開発への参加を希望される方は課題Aに、CuPy開発への参加を希望される方は課題Bに、それぞれ取り組んでください。

課題A

問題: 再計算によるメモリ削減の実装

ニューラルネットワークの消費メモリを減らす方法として再計算が知られています。
再計算は名前の通り、いますぐ必要でない変数のメモリを解放し、将来その変数が必要になったときに、再び計算することでピークのメモリを下げるテクニックです。
再計算では、一般的にピークメモリを下げられる代わりに、実行時間が伸びるというトレードオフがあります。

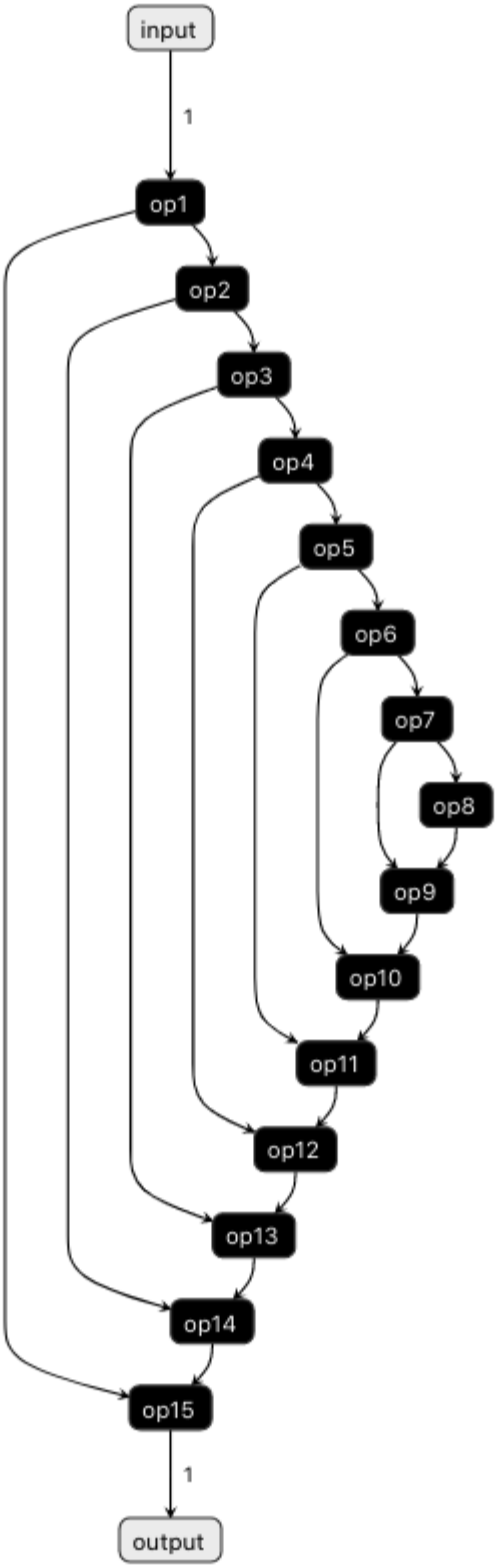
例として次のような計算グラフを考えてみましょう。この計算グラフはop1, …, op5の5つのオペレータを持ちます。この課題において、オペレータの出力は常に1つで、計算グラフに現れる変数のサイズはすべて1とします。



この計算グラフを [op1, op2, op3, op4, op5] の順に愚直に処理した場合、inputは常にメモリに乗り続けるとすると、op4を計算する時点でop1からop4の出力がすべて生きているので合計5のメモリを必要とします。
しかし、[op1, op2, op3, op4, op1, op5] の順に計算する場合（最初に計算するop1の出力はop2の計算後解放されます）、消費メモリは4で済みます。

問題

以下の計算グラフについて、再計算を考慮した計算順序を考え、愚直に計算した場合とのトレードオフをレポートしてください。



必要に応じて https://tech.preferred.jp/ja/blog/recomputation/ を参考にしても構いません。

提出物

以下をzipファイルにまとめて提出してください。

1. 実験に使用したプログラム（あれば）
2. 計算順序がvalidなもので、その消費メモリを求めるプログラム
3. レポート

レポートはA4 2枚程度にまとめ、以下の内容を含めてください。

- 問題の計算グラフにおける、再計算を考慮した計算順序、およびその説明
- 愚直に計算した場合とのトレードオフ
- 以下の再計算の手法に関する論文から1つ選び、その内容を簡潔に説明してください。
    - Chen, Tianqi, et al. "Training deep nets with sublinear memory cost." arXiv preprint arXiv:1604.06174 (2016).
    - Kusumoto, Mitsuru, et al. "A graph theoretic framework of recomputation algorithms for memory-efficient backpropagation." Advances in Neural Information Processing Systems 32 (2019).
    - Kumar, Ravi, et al. "Efficient rematerialization for deep networks." Advances in Neural Information Processing Systems 32 (2019).

課題B

以下の2問から1問選択してください。

問題1: 最頻値を求めるプログラムのCUDA実装

入力として符号なし64ビット整数の配列がひとつ与えられます。配列の中で最も出現回数の多い値のうち最小のものを出力するCUDAプログラムを実装してください。必要であれば、CUDAに付属する標準ライブラリ（Thrust, CUBを含む）を使用しても構いませんが、それ以外のライブラリは使用しないでください。

入力について

入力の配列の長さは 2^24 です。以下の手順で生成されます。

1. 符号なし64ビット整数の全ての範囲から値の候補を一様ランダムに重複を含めて 2^20 個選ぶ。
2. 値の候補から一様ランダムに 2^24 回選んで配列を生成する。

例えば、入力を生成する関数を C++ で実装すると、以下のようになります。

```
```
```
void generate_testcase(uint64_t* a, int seed) {
  const int N = 1 << 24;
  const int M = 1 << 20;
  std::mt19937 mt(seed);
  std::uniform_int_distribution<uint64_t> dist1;
  std::vector<uint64_t> x(M);
  for (int i = 0; i < M; ++i) {
    x[i] = dist1(mt);
  }
  std::uniform_int_distribution<int> dist2(0, M - 1);
  for (size_t i = 0; i < N; ++i) {
    a[i] = x[dist2(mt)];
  }
```

```
}
```

提出物

以下をzipファイルにまとめて提出してください。

1. 実装コード
    - 入力の最頻値を求める関数の実装
    - コンパイルを行うためのコマンドライン
    - 出力の値の正当性が確認できるテストコード
    - 関数呼び出し1回あたりの実行時間を計測するベンチマーク
    - プログラムの可読性のために必要な最小限のコメント
2. レポート
    - レポートはA4 2枚程度にまとめてください。
    - 工夫した点や、性能評価に対する考察を記載してください。

問題2: Python関数からC++関数へのトランスパイラ

与えられたPythonの関数をastモジュールを利用して解析し、C++関数に変換する関数をPythonで実装してください。例えば、

```
def func(a, n):
    i = 0
    out = 0.
    while i < n:
        out += a[i]
        i += 1
    return out

transpile(func, (list[float], int))
```

上のプログラムを実行したときに、以下のような C++ 関数を文字列として返す関数 `transpile` を実装してください。

```
double func(double *a, int n) {
  int i = 0;
  double out = 0;
  while (i < n) {
    out += a[i];
    i += 1;
  }
  return out;
}
```

以上の出力はあくまでひとつの例であり、上の入力例に対してこの出力例と一字一句同じ文字列を出力する必要はありません。

トランスパイラの仕様

- 以下の機能を使用したPython関数を処理できること。
    - 引数型: bool, int, float, list[int], list[double] のいずれか
    - 算術演算 (+, -, *, /, //, %)、比較演算 (==, !=, <, <=, >, >=)、ブール演算 (not, and, or)
    - 代入、if 文、while 文、関数呼び出し (再帰関数を含む)
    - 最低限のリスト操作: 要素の取得、書き込み
        - リストのスライスや、リストの構築は考えなくてよい
- 生成する C++ の関数の中で auto は使用しないこと。
- その他、上記で定義されていない仕様については任意に定めてよい。

提出物

以下をzipファイルにまとめて提出してください。

1. トランスパイラの実装コード
    - コード中に可読性のための必要最小限のコメントを書いてください。
    - トランスパイラの動作が確認できる入力例も含めてください。
2. レポート
    - レポートはA4 2枚程度にまとめてください。
    - 上記で定義されていない追加の仕様や、工夫した点について書いてください。
    - トランスパイラによる変換が困難な入力例がある場合や、実装しきれなかったアイデアがある場合は書いてください。