

# 2022年引き継ぎ資料

津村周作

# この引き継ぎ資料について

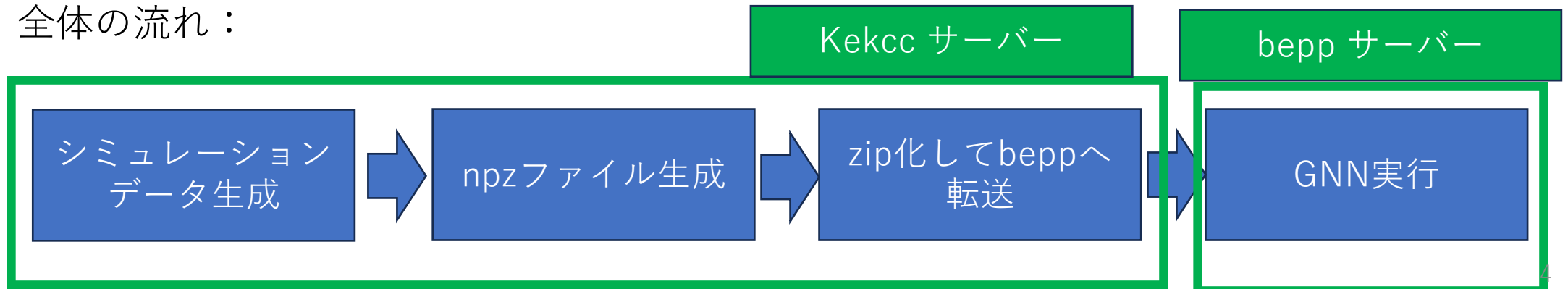
- 研究の全体的な流れは以下の修論にあります。
  - [https://github.com/Shusaku-T/MasterThesis\\_2022.git](https://github.com/Shusaku-T/MasterThesis_2022.git)
- また、機械学習・深層学習については後藤さんの資料も参考になります。
  - [https://github.com/Goto-K/Master\\_Thesis](https://github.com/Goto-K/Master_Thesis)
- 主な違いとしては、
  - ライブラリとしてTensorflowかPyTorchか
  - NetworkとしてDense・RNNかグラフか
  - TaskとしてVertexFinderかParticleFlowかなどが挙げられます
- この資料では技術的な部分についていくつか説明します。

# Contents

- GravNetを用いたPFAの実装
  - ILD検出器シミュレーション
  - LCIOファイル→NPZファイルへの変換
  - Kekcc→beppへの転送
  - GravNetによる深層学習
  - Network の評価
- EBES実験のための検出器性能評価
  - エネルギー較正
  - エネルギー分解能評価
  - HV scan

# GravNetを用いたPFAの実装

- ILD検出器シミュレーションをkekccで回して、深層学習をbepp gpuで行なっています
  - iLCSoftをbepp gpuに入れて二つを同じ環境で回してもいいと思います。ただし、CPUやメモリの消費には気をつけてください。
  - beppでの環境構築については後藤さんの資料にあります。
- kek ccの私のディレクトリpathは  
/home/ilc/stsumura  
です。(tsumuraではないので注意)
- 論文の二光子事象のシミュレーションはdoublePG/exampleで実行し、できたデータは  
/group/ilc/users/stsumuraに置いてます
- 全体の流れ：



# ILC検出器シミュレーション

- 実行手順
- まず、データを置くディレクトリを作成します
  - `./prepareSim.sh`
- 100000ファイル作成する工程を並列処理するためにbsubシステムを使っています。
  - `./bsub_sim.sh`
  - `./bsub_convert.sh`
    - `./bsub_sim.sh`が全て終わった後で`./bsub_convert.sh`を実行してください
    - 終わったかどうかは**bjobs**で確認できます。
  - 中身は`exe_LCIO_sim.sh`および`exe_LCIO_convert.sh`にあります。
  - Kekccサーバーでの一連の流れ：`lcio_particle_gun.py` → `ddsim` → `Marlin` → `LCIO2npz_dPG.py` → `zip`

# prepare.sh

- シミュレーションデータを置くディレクトリを作成しています
- 新しく作成されるディレクトリの名前は固定していて、もし古いディレクトリが作成されるとその古いディレクトリの名前が現在時刻で上書きされます。
- このコードを実行していないと、データを作成するときにエラーが出ます。

# LCIOファイル→NPZファイルへの変換

- bsubを行うと自動的に実行されます。
  - 実行ファイルはLCIO2npz\_dPG.py
- LCIOからHitの位置・正解ラベル・PDGなどを引っ張ってきます。
  - MC粒子の参照元が間違っている可能性あり
- 生成されたファイルは圧縮されてzipファイルとしても保存されています
  - beppへ送るのはこちらのファイルです

# KEKcc→Beppへの転送

- scp -rでeppを経由してbeppサーバーまで転送します
  - 場所は/gluster/maxi/ilcです
- 持ってきたファイルはunzipで解凍した上でそれぞれのイベントごとにファイルを分割します。
  - 生成した段階で分割すると時間がかかるのでここで分割しています
  - 分割するための実行コードはnpz\_concate.pyです。引数に注意してください



# 深層学習の実行

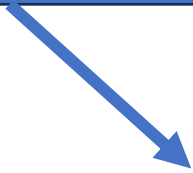
- Zipファイルをunzipで解凍します。
- ここで作成したnpzファイルは転送時間を減らすために1000イベントが一つのファイルに入っているので、npz\_concat.pyでそれぞれのイベントごとにファイルを作成します。

# 深層学習の実行

- Zipで解凍したディレクトリを深層学習のinputファイルとして設定します。
- Settingにあるtxtファイルをこのinputのディレクトリに設定してください。
- train\_ILC.pyのReadFileメソッドの引数に設定されているのが上記のtxtファイルであることを確認して、train\_ILC.pyを実行してください

# ネットワークの評価

- train\_ILC.pyを実行すると、ディレクトリcheckpointの中に各エポックごとのGravNetモデルが出力されています。
- Settingのtxtファイルのinput modelにそれぞれのモデルの名前を入力することで評価対象のモデルを指定できます
- `plot_statistics ILC.py`を実行してネットワークの定量的評価を行います。



P12

# ネットワークの評価

一行ごとの詳細は  
次ページ以降

- plot\_statistics\_ILC.py --get\_stats関数

```
def get_stats(tbeta=.2, td=.5, nmax=4, yielder=None, accType="number"):  
    stats = ev.Stats()  
    if yielder is None: yielder = ev.TestYielder()  
    count_cluster_inEvent = [[], []]  
    rate_accuracy = [[], []]  
    energy_rate_accuracy = [[], []]  
    check = True  
    for event, prediction, clustering, matches in yielder.iter_matches(tbeta, td, nmax):  
        #count_cluster_inEvent=event_count(event, clustering, count_cluster_inEvent)  
        #rate_accuracy=count_hit_inCluster(event, clustering, rate_accuracy)  
        #energy_rate_accuracy, rate_1st, rate_2nd = cal_energy_rate(event, clustering, energy_rate_accu  
        racy)  
        #if check == True:  
        #    print(f"matches: {matches}")  
        #    if len(matches[1]) > 2: check = False  
        #stats.extend(ev.statistics_per_match(event, clustering, matches))  
        if accType == "number":  
            stats.extend(ev.get_hit_matched_vs_unmatched(event, clustering, matches))  
        elif accType == "energy":  
            stats.extend(ev.get_hit_matched_vs_unmatched_energy(event, clustering, matches))  
        #stats.extend(ev.show_Stats(event, clustering, matches))  
        #stats.add('confmat', ev.signal_to_noise_confusion_matrix(event, clustering, norm=True))  
    return stats
```

データを入れるコンテナを用意→P13

データローダーを用意 P14

今回は使っていない(statsを使用)

データローダー  
からデータを持ってくる  
→P14

コンテナに値を  
詰める  
→P13/15

# Plot\_statistics\_ILC.py –getstats()

- Stats: Evaluation\_noNoise.pyにおいて実装

```
class Stats:
    """Container for statistics per object (event or match)

    Useful to keep track of quantities per object, and then combine
    for many objects.
    """
    def __init__(self):
        self.d = {}

    def __getitem__(self, key):
        return self.d[key]

    def add(self, key, val):
        """Add single value to a key"""
        val = np.expand_dims(np.array(val), 0)
        if key not in self.d:
            self.d[key] = val
        else:
            self.d[key] = np.concatenate((self.d[key], val))

    def extend(self, other):
        """Extend with another Stats object"""
        for k, v in other.d.items():
            if k in self.d:
                self.d[k] = np.concatenate((self.d[k], v))
            else:
                self.d[k] = v

    def __len__(self):
        for key, val in self.d.items():
            return len(val)
```

各クラスターの学習元データや解析結果などをdict型でキープするためのクラス

例：各イベントごとに一つのクラスターに含まれるヒットの数を、イベントごとのリスト  
stats['num\_pred']  
として保存

add()で新しくキーを追加

extend()ですでに作成されたキーに新しく要素を追加

※単にデータを見たいだけであれば、このクラスを使わなくてもいい

# TestYielder -iter\_matches()

- Evaluation\_noNoise.pyにおいて実装

```
def iter_matches(self, tbeta, td, nmax=None):  
    for event, prediction, clustering in self.iter_clustering(tbeta, td, nmax):  
        matches = make_matches(event, prediction, clustering=clustering)  
        yield event, prediction, clustering, matches
```

Event : 入力パラメータ

prediction : 予測されるbetaの値

clustering : predictionの値からどのクラスターかを推測

matches : 下のmake\_matchesを用いてclusteringの情報から同じクラスターに属するヒットをイベントごとに一つのリストにまとめる。

```
def make_matches(event, prediction, tbeta=.2, td=.5, clustering=None):  
    if clustering is None: clustering = cluster(prediction, tbeta, td)  
    i1s, i2s, _ = match(event.y, clustering, weights=event.energy)  
    matches = group_matching(i1s, i2s)  
    return matches
```



# Get\_hit\_matched\_vs\_unmatched()

- Evaluation\_noNoise.pyにおいて実装

```
def get_hit_matched_vs_unmatched(event:Event, clustering, matches, noise_index=0):
    stats = Stats()
    #if len(set(event.y))>2 : return stats
    for matches_i in matches:
        rate_pred=[]
        matched_pred = np.array([])
        matched_truth = np.where(event.y == matches_i[0])
        for matches_pred_id in matches_i[1]:
            matched_pred = np.append(matched_pred,np.count_nonzero(clustering[matched_truth] == mat\
ches_pred_id))
        stats.add('num_pred',len(matched_truth[0]))
        for matched_pred_i in matched_pred:
            rate_pred=np.append(rate_pred,matched_pred_i/len(matched_truth[0]))
        stats.add('rate_pred',np.max(rate_pred))
    return stats
```

matchesのリストの中から、正解ラベルに一致しているヒットを一つのリストにまとめ、その数から正解率を計算する

# EBES実験のための 検出器性能評価



# 内容

- ここではコードの解説のみにとどめます
- mcpAnalysis.ccの流れは以下の通りです。

