

# 引継ぎ資料

## - 崩壊点検出 + DL -

Kiichi Goto

goto@epp.phys.kyushu-u.ac.jp

kiichi.goto.0209@gmail.com

# 目次

1. 引継ぎ資料を読む前に
2. LCFIPlus
  1. LCFIPlus
  2. LCFIPlus processor / ROOTファイルの作り方
3. Deep Learning
  1. Feedforward
  2. Recurrent
  3. Encoder-Decoder
  4. Vertex Finder
4. LCFIPlus + DL
  1. Tensorflow C++ API
  2. C++でのネットワークの動作
  3. CMakeによるLCFIPlusへの導入
  4. LCFIPlus上でのネットワークの動作
5. やり残した事

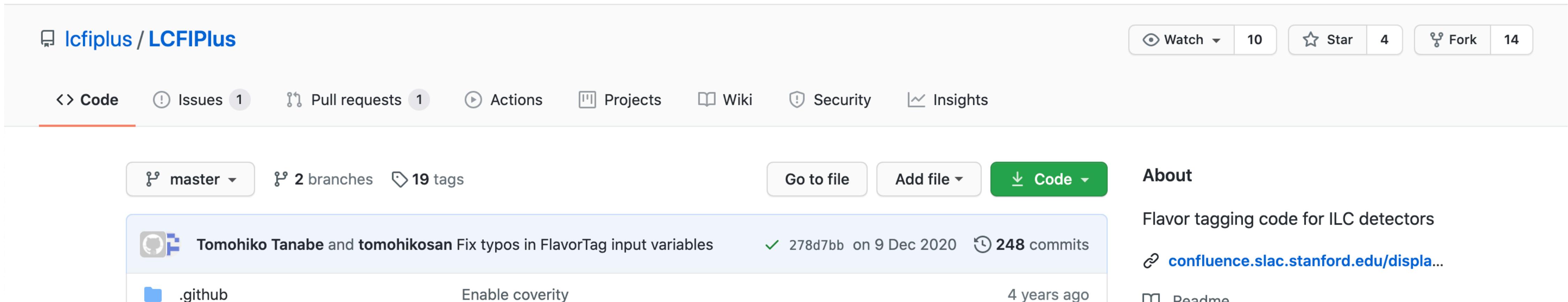
# 1. 引継ぎ資料を読む前に

- 注意事項
  - この資料はC++/Pythonの教科書ではありません。  
実装で分からぬ事があれば、調べるかメールで質問をして下さい。
  - この資料は機械学習 (ML), 深層学習 (DL) の教科書ではありません。  
細かい理論の話はせず、基本的に実装についての話だけをします。
- 引継ぎ資料を読む前に
  - まずは以下の修士論文を読んで下さい。(少なくとも2, 3章)  
[https://github.com/Goto-K/Master\\_Thesis](https://github.com/Goto-K/Master_Thesis)
  - 機械学習, 深層学習について、勉強しましょう。
    - 深層学習に出来て、他の機械学習技術に出来ない事。あるいはその逆。
    - 修士論文の2章には一通りの導入を書いていますが、ここには画像処理系の技術領域を全く書いていない為、  
画像処理関係の書籍を読みましょう。
    - 以下独断と偏見で選んだ勉強すると良い基礎領域
      - 画像処理 (CNN, GAN, Semantic Segmentation)
      - グラフ (GCN, Graph Attention Network)
      - その他 (Auto-encoder)

# 2. LCFIPlus

## LCFIPlus

- まず, LCFIPlusの論文に目を通しましょう。  
<https://www.sciencedirect.com/science/article/pii/S0168900215014199?via%3Dihub>
- 次に, iLCSofトの資料に目を通しましょう。  
iLCSofト Tutorial
- iLCSofについて理解出来たら, LCFIPlusをGitHubから落としてきましょう  
<https://github.com/lcfiplus/LCFIPlus> or <https://github.com/Goto-K/LCFIPlus>
  - 後者は僕のGitHub多少書き換えているので, 前者と若干異なります。
  - installには「4.1. Tensorflow C++ API」が必要です。
  - 一先ず前者をinstallすると良いと思います。



The screenshot shows the GitHub repository page for `lcfiplus/LCFIPlus`. The page includes the following elements:

- Repository Header:** Shows the repository name `lcfiplus / LCFIPlus`, a star count of 10, a fork count of 14, and buttons for Watch, Star, and Fork.
- Navigation Bar:** Includes links for Code, Issues (1), Pull requests (1), Actions, Projects, Wiki, Security, and Insights.
- Code Area:** Features a dropdown for the master branch, showing 2 branches and 19 tags. It also has buttons for Go to file, Add file, and Code.
- Commit History:** Displays a recent commit by Tomohiko Tanabe and tomohikosan, dated 9 Dec 2020, with 248 commits.
- About Section:** Describes the repository as "Flavor tagging code for ILC detectors" and provides a link to [confluence.slac.stanford.edu/display...](https://confluence.slac.stanford.edu/display/).

# 2. LCFIPlus

## LCFIPlus

- iLCSOftにPATHを通し, LCFIPlusを動かしてみましょう。
  - Bepp-gpuにiLCSOft v02-02環境を用意しているので, こちらを使用すると良いと思います。  
bepp:/gluster/data/ilc/ilcsoft/v02-02/
  - “bepp:/gluster/data/ilc/ilcsoft/v02-02/init\_ilcsoft.sh” というファイルがあるので, Localの適当な場所にcopyし, MarlinのLCFIPlusの部分を自身のLCFIPlusに書き換えましょう。

```
46 #--  
47 #     Marlin  
48 #---  
49 export MARLIN="/gluster/data/ilc/ilcsoft/v02-02/Marlin/v01-17-01"  
50 # --- additional Marlin commands -----  
51 export PATH="$MARLIN/bin:$PATH"  
52 export MARLIN_DLL="/gluster/data/ilc/ilcsoft/v02-02/MarlinDD4hep/v00-06/lib/libMarlinDD4hep.so:/gluster/data/ilc/ilcsoft/v02-02/DDMarlinPandora/v00-11/lib/libDDMarlinPandora.so:/gluster/data/ilc/ilcsoft/v02-02/MarlinReco/v01-28/lib/libMarlinReco.so:/gluster/data/ilc/ilcsoft/v02-02/PandoraAnalysis/v02-00-01/lib/libPandoraAnalysis.so:/gluster/data/ilc/ilcsoft/v02-02/LCFIVertex/v00-08/lib/libLCFIVertexProcessors.so:/gluster/data/ilc/ilcsoft/v02-02/CEDViewer/v01-17-01/lib/libCEDViewer.so:/gluster/data/ilc/ilcsoft/v02-02/Overlay/v00-22/lib/libOverlay.so:/gluster/data/ilc/ilcsoft/v02-02/MarlinFastJet/v00-05-02/lib/libMarlinFastJet.so:/gluster/data/ilc/ilcsoft/v02-02/LCTuple/v01-12/lib/libLCTuple.so:/gluster/data/ilc/ilcsoft/v02-02/MarlinKinfit/v00-06/lib/libMarlinKinfit.so:/gluster/data/ilc/ilcsoft/v02-02/MarlinTrkProcessors/v02-11/lib/libMarlinTrkProcessors.so:/gluster/data/ilc/ilcsoft/v02-02/MarlinKinfitProcessors/v00-04-02/lib/libMarlinKinfitProcessors.so:/gluster/data/ilc/ilcsoft/v02-02/ILDPerformance/v01-08/lib/libILDPerformance.so:/gluster/data/ilc/ilcsoft/v02-02/Clupatra/v01-03/lib/libClupatra.so:/gluster/data/ilc/ilcsoft/v02-02/Physsim/v00-04-01/lib/libPhyssim.so:/home/goto/ILC/LCFIPlus/lib/libLCFIPlus.so:/gluster/data/ilc/ilcsoft/v02-02/FCalClusterer/v01-00-01/lib/libFCalClusterer.so:/gluster/data/ilc/ilcsoft/v02-02/ForwardTracking/v01-14/lib/libForwardTracking.so:/gluster/data/ilc/ilcsoft/v02-02/ConformalTracking/v01-10/lib/libConformalTracking.so:/gluster/data/ilc/ilcsoft/v02-02/LICH/v00-01/lib/libLICH.so:/gluster/data/ilc/ilcsoft/v02-02/Garlic/v03-01/lib/libGarlic.so:  
$MARLIN_DLL"
```

## 2. LCFIPlus

### LCFIPlus

- iLCSoftにPATHを通し, LCFIPlusを動かしてみましょう。
  - PATHを通したら, ilcsoftのrunの為のdirectoryを用意し, 以下のファイルを置きます。 (名前は何でも良いです。)
    - runilcsoft
      - \*.xml
      - run\*.sh
      - log
      - output
  - \*はprocessorの名前です。 (vtxfinder, flatag, など)
  - xmlファイルは以下にサンプルがあります。  
[bepp:/gluster/data/ilc/ilcsoft/ILDConfig/v01-17-09/LCFIPlusConfig/steer/v03\\_p01/](bepp:/gluster/data/ilc/ilcsoft/ILDConfig/v01-17-09/LCFIPlusConfig/steer/v03_p01/)
  - runファイルはMarlinを実行する為のファイルで, 読み込むファイルの場所とsteer(xml)を指定します。
  - logとoutputはsymbolic linkにすると良いと思います。
  - 具体的には"bepp:/home/goto/ILC/runilcsoft"を見て下さい。 copyしても良いですがオススメはしません。

# 2. LCFIPlus

## LCFIPlus xmlファイル

```
1 <?xml version="1.0" encoding="us-ascii"?>
2
3 <!--marlin-->
4 <marlin xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://ilcsoft.desy.de/marlin/marlin.xsd">
5 <execute>
6   <!-- <processor name="InitDD4hep"/> -->
7   <processor name="VertexFindingwithDL"/>
8   <processor name="MyLCIOOutputProcessor"/>
9 </execute>
10
11 <global>
12   <parameter name="LCIOInputFiles">
13     input.slcio
14   </parameter>
15   <!-- <parameter name="GearXMLFile"> GearOutput.xml </parameter> -->
16   <parameter name="MaxRecordNumber" value="0" />
17   <parameter name="SkipNEvents" value="0" />
18   <parameter name="SupressCheck" value="false" />
19   <parameter name="Verbosity" options="DEBUG0-4,MESSAGE0-4,WARNING0-4,ERROR0-4,SILENT">MESSAGE0</parameter>
20 </global>
21
22 <processor name="InitDD4hep" type="InitializeDD4hep">
23   <!--InitializeDD4hep reads a compact xml file and initializes the DD4hep::LCDD object-->
24   <!--Name of the DD4hep compact xml file to load-->
25   <parameter name="DD4hepXMLFile" type="string"> /gluster/data/ilc/ilcsoft/v02-02/lcgeo/v00-16-06/ILD/compact/ILD_l6_v02/ILD_l6_v02.xml </parameter>
26 </processor>
27
28
29 <processor name="VertexFindingwithDL" type="LcfiplusProcessor"> 実行するプロセッサー
30   <!-- run primary and secondary vertex finders -->
31   <!--parameter name="Algorithms" type="stringVec">JetClustering TrackNtuple</parameter-->
32   <parameter name="Algorithms" type="stringVec">VertexFindingwithDL</parameter>
33
34   <parameter name="MagneticField" type="float" value="3.5" /> プロセッサーに必要な変数の設定
35
36   <!-- general parameters -->
37   <parameter name="PFOCollection" type="string" value="PandoraPF0s" /> <!-- input PF0 collection -->
38   <parameter name="UseMCP" type="int" value="1" /> <!-- MC info not used -->
39   <!-- rewrite!!!! -->
40   <parameter name="MCPCollection" type="string" value="MCParticlesSkimmed" /> <!-- used -->
41   <parameter name="MCPF0Relation" type="string" value="RecoMCTruthLink" /> <!-- used -->
42
```

# 2. LCFIPlus

## LCFIPlus xmlファイル

```
67     <parameter name="VertexFindingwithDL.ThresholdPrimaryVertexMerge" type="double" value="20" />
68     <parameter name="VertexFindingwithDL.ThresholdLikeAFake" type="double" value="0" />
69
70     <parameter name="VertexFindingwithDL.Debug" type="bool" value="0" />
71
72     <parameter name="VertexFindingwithDL.PairModelPath" type="string" value="/home/goto/ILC/VertexFinderwithDL/models/Pair_Model_Standard" />
73     <parameter name="VertexFindingwithDL.LSTMModelPath" type="string" value="/home/goto/ILC/VertexFinderwithDL/models/VLSTM_Model_Standard_PV" />
74     <parameter name="VertexFindingwithDL.SLSTMModelPath" type="string" value="/home/goto/ILC/VertexFinderwithDL/models/VLSTM_Model_Standard_SV" />
75
76     <!-- jet clustering parameters -->
77     <parameter name="JetClustering.InputVertexCollectionName" type="string" value="BuildUpVertexDL" /> <!-- vertex collections to be used in JC -->
78     <parameter name="JetClustering.OutputJetCollectionName" type="stringVec" value="VertexJets" /> <!-- output collection name, may be multiple -->
79     <parameter name="JetClustering.NJetsRequested" type="intVec" value="2" /> <!-- Multiple NJets can be specified -->
80
81     <parameter name="JetClustering.YCut" type="doubleVec" value="0." /> <!-- specify 0 if not used -->
82     <parameter name="JetClustering.UseMuonID" type="int" value="1" /> <!-- jet-muon ID for jet clustering -->
83     <parameter name="JetClustering.VertexSelectionMinimumDistance" type="double" value="0.3" /> <!-- in mm -->
84     <parameter name="JetClustering.VertexSelectionMaximumDistance" type="double" value="30." /> <!-- in mm -->
85     <parameter name="JetClustering.VertexSelectionK0MassWidth" type="double" value="0.02" /> <!-- in GeV -->
86     <parameter name="JetClustering.YAddedForJetVertexVertex" type="double" value="100"/> <!-- add penalty for combining vertices -->
87     <parameter name="JetClustering.YAddedForJetLeptonVertex" type="double" value="100"/> <!-- add penalty for combining lepton and vertex -->
88     <parameter name="JetClustering.YAddedForJetLeptonLepton" type="double" value="100"/> <!-- add penalty for combining leptons -->
89
90     <!-- <parameter name="PrimaryVertexCollectionName" type="string" value="PrimaryVertex" /> -->
91 </processor>
92
93 <processor name="MyLCIOOutputProcessor" type="LCIOOutputProcessor">          実行するプロセッサー
94     <!-- standard output: full reconstruction keep all collections -->
95     <parameter name="LCIOOutputFile" type="string" >
96         ./output/nobeam_Pqq_Gwhizard-1_95_nopol_2_75_50_20_MTFIG1.slcio    プロセッサーに必要な変数の設定 (Output File Name)
97     </parameter>
98     <parameter name="LCIOWriteMode" type="string" value="WRITE_NEW"/>
99     <!--parameter name="SplitFileSizekB" type="int" value="1992294"-->
100 </processor>
101
102 </marlin>
```

# 2. LCFIPlus

## LCFIPlus

### runファイル

```
1 #!/bin/bash
2
3 gearfile=/home/goto/ILC/runilcsoft/GearOutput.xml
4
5 maxrecords=0
6 dstdir=/gluster/maxi/ilc/reco/dbd/train-lcfiplus/bb91new
7
8 i=0
9 dstfiles=""
10 for dstfile in `ls -d ${dstdir}/*.slcio | grep "Pbb.Gwhizard-1_95.nopol.08.0"`;
11   dstfiles="${dstfiles} ${dstfile}"
12 done
13
14 Marlin --global.LCIOInputFiles="${dstfiles}" \
15       --global.MaxRecordNumber=${maxrecords} \
16       vtxfinder.xml      xmlファイル
17 
```

Input File Name

## 2. LCFIPlus

### LCFIPlus

- iLCSoftにPATHを通し, LCFIPlusを動かしてみましょう。
  - “sh run\*.sh”などを実行し, LCFIPlusが動作するか確認しましょう。
  - 磁場で止まる場合は僕のLCFIPlusからsrcファイルをいくつか持ってくると解決するかもしれません。  
(iLCSoftが新しすぎる為, 磁場の設定ファイルが読み込めず上手く回らない場合がある)

# 2. LCFIPlus

## LCFIPlus Processor

- LCFIPlusの動作が確認できたら, LCFIPlus Processorの作成を行いましょう。
- 新しいファイルを用意して, (あるいは既にあるファイルを名前を変えてcopyして) LCFIPlus/include, src以下に置きます。  
(例えば僕の “bepp:/home/goto/ILC/LCFIPlus/include, src” 以下にある “makerootfileforDL.h, makerootfileforDL.cc”など)
- これらのプロセッサーがLCFIPlusから認識される様に以下を書き換えます。(足りないかも)
  - include/LinkDef.h
  - CMakeList.txt

```
--  
33 #pragma link C++ class lcfiplus::Algorithm;  
34 #pragma link C++ class lcfiplus::BuildUpVertex;  
35 #pragma link C++ class lcfiplus::JetClustering;  
36 #pragma link C++ class lcfiplus::JetVertexRefiner;  
37 #pragma link C++ class lcfiplus::PrimaryVertexFinder;  
38 #pragma link C++ class lcfiplus::FlavorTag;  
39 #pragma link C++ class lcfiplus::MakeNtuple;  
40 #pragma link C++ class lcfiplus::TrainMVA;  
41 #pragma link C++ class lcfiplus::ReadMVA;  
42 #pragma link C++ class lcfiplus::TrackNtuple;  
43 #pragma link C++ class lcfiplus::VertexMassRecovery+;  
44 #pragma link C++ class lcfiplus::VertexNtuple;  
45  
46 #pragma link C++ class lcfiplus::VertexFindingwithDL;  
47 #pragma link C++ class lcfiplus::MakeROOTFileCC;  
48 #pragma link C++ class lcfiplus::MakeROOTFileBB;  
49 #pragma link C++ class lcfiplus::MakeROOTFileTracks;
```

```
59  
60 SET( ROOT_DICT_INPUT_HEADERS  
61         ${PROJECT_SOURCE_DIR}/include/lcfiplus.h  
62         ${PROJECT_SOURCE_DIR}/include/EventStore.h  
63         ${PROJECT_SOURCE_DIR}/include/LCIOStorer.h  
64         ${PROJECT_SOURCE_DIR}/include/TreeStorer.h  
65         ${PROJECT_SOURCE_DIR}/include/JetFinder.h  
66         ${PROJECT_SOURCE_DIR}/include/process.h  
67         ${PROJECT_SOURCE_DIR}/include/FlavorTag.h  
68         ${PROJECT_SOURCE_DIR}/include/MakeNtuple.h  
69         ${PROJECT_SOURCE_DIR}/include/TrainMVA.h  
70         ${PROJECT_SOURCE_DIR}/include/ReadMVA.h  
71         ${PROJECT_SOURCE_DIR}/include/testproc.h  
72         ${PROJECT_SOURCE_DIR}/include/TrackNtuple.h  
73         ${PROJECT_SOURCE_DIR}/include/VertexMassRecovery.h  
74         ${PROJECT_SOURCE_DIR}/include/VertexNtuple.h  
75  
76         ${PROJECT_SOURCE_DIR}/include/processwithDL.h  
77         ${PROJECT_SOURCE_DIR}/include/makerootfileforDL.h  
78  
79         ${PROJECT_SOURCE_DIR}/include/LinkDef.h  
80 )
```

# 2. LCFIPlus

## LCFIPlus Processor

- make installして上手く動作するか確認しましょう。
- 動作していたら、少しづつ書き換えて理解していきます。
  - Event, Track, Vertex, JetそれぞれのObjectの意味や変数。  
[https://ilcsoft.desy.de/LCIO/current/doc/doxygen\\_api/html/annotated.html](https://ilcsoft.desy.de/LCIO/current/doc/doxygen_api/html/annotated.html)
  - ROOTファイルにどの様な値を詰め込むか。(Tree, Fill, などの理解)
- 基本的に深層学習のトレーニングでは、このROOTファイル内の情報を NumpyというPython形式のファイルに落として使用します。
-

# 3.0. - 0.1. ROOT2Numpy

- ROOT to Numpyは非常に簡単です。
- 実行はPythonから行います。

```
1 import ROOTTOOLS
2 import TOOLS
3 import sys
4 import numpy as np
5
6
7 if __name__ == "__main__":
8     pdpath = "/home/goto/ILC/runilcsoft/output/pair/" 出力Numpyファイル
9     npyname = pdpath + "bb_08_pair_lcfiplusmc_v2.npy"
10    fnames = ["/home/goto/ILC/runilcsoft/output/pair/bb_08_pair_lcfiplusmc_v2.root"] 入力ROOTファイル
11
12    bnames = ['nevent', 'ntr1track', 'ntr2track', 'ROOTファイル内のブランチ名
13        # Track 1 : 22 input variables
14        'tr1d0', 'tr1z0', 'tr1phi', 'tr1omega', 'tr1tanlam', 'tr1charge', 'tr1energy',
15        'tr1covmatrixd0d0', 'tr1covmatrixd0z0', 'tr1covmatrixd0ph', 'tr1covmatrixd0om',
16        'tr1covmatrixz0z0', 'tr1covmatrixz0ph', 'tr1covmatrixz0om', 'tr1covmatrixz0tl',
17        'tr1covmatrixphom', 'tr1covmatrixphtl', 'tr1covmatrixomom', 'tr1covmatrixomtl',
18        # Track 2 : 22 input variables
19        'tr2d0', 'tr2z0', 'tr2phi', 'tr2omega', 'tr2tanlam', 'tr2charge', 'tr2energy',
20        'tr2covmatrixd0d0', 'tr2covmatrixd0z0', 'tr2covmatrixd0ph', 'tr2covmatrixd0om',
21        'tr2covmatrixz0z0', 'tr2covmatrixz0ph', 'tr2covmatrixz0om', 'tr2covmatrixz0tl',
22        'tr2covmatrixphom', 'tr2covmatrixphtl', 'tr2covmatrixomom', 'tr2covmatrixomtl',
23        # fitter feature value
24        'vchi2', 'vposx', 'vposy', 'vposz', 'mass', 'mag', 'vec', 'tr1selection', 'tr2s
25        'connect', 'lcfiplusustag',
26        'tr1id', 'tr1pdg', 'tr1ssid', 'tr1sspdg', 'tr1ssc', 'tr1ssb', 'tr1oth', 'tr1pri
27        'tr2id', 'tr2pdg', 'tr2ssid', 'tr2sspdg', 'tr2ssc', 'tr2ssb', 'tr2oth', 'tr2pri
28    ]
29
30    data, nevent = ROOTTOOLS.fileload_setnames(fnames, "track0", bnames)
31    data = np.array(data).reshape(nevent, len(bnames))
32    np.save(npyname, data, fix_imports=True)
33
```

```
7 import ROOT
8
9
10 def fileread(fname, tname):
11     f = ROOT.TFile(fname, "read")
12     tree = f.Get(tname)
13
14     lb = tree.GetListOfBranches()
15     bnames = [lb.At(s).GetName() for s in range(lb.GetEntries())]
16
17     return f, tree, bnames
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37 def fileload_setnames(fnames, tname, bnames):
38     data = []
39     ievent = 0
40
41     for fname in fnames:
42         f, tree, _ = fileread(fname, tname)
43
44         for i in tqdm(range(tree.GetEntries())):
45             #for i in tqdm(range(10000)):
46             tree.GetEntry(i)
47             #if tree.connect<0: continue
48             data.append([])
49             data[ievent].append([getattr(tree, bname) for bname in bnames])
50             ievent += 1
51
52     return data, ievent
53
```

# 3. Deep Learning

## Deep Learning

- Deep LearningはTensorflow / Kerasを用いて実装します。
  - PyTorchなどに変更しても良いです
- 動作versionは以下のものを使用しています。
  - Tensorflow 2.1.0
  - Keras 2.3.1
- 実装に関しても、ある程度書籍/webなどで前提知識を学んでおくと良いと思います。
- ここでは、本研究で使用したコードの内、特に説明が必要なものを解説します。
- 完全な実装は以下を確認してください
  - <https://github.com/Goto-K/VertexFinderwithDL>
  - “bepp:/home/goto/ILC/VertexFinderwithDL”
  - “lto:/home/usr5/p70545c/ILC/VertexFinderwithDL”
- VertexFinderwithDL
  - Networks : Networkの定義
  - data
  - log
  - models

# 3. Deep Learning

## Deep Learning

- VertexFinderwithDL
  - Networks
    - PairModel : 飛跡対についてのネットワーク
    - VLSTMModel : 任意の数の飛跡についてのネットワーク
    - Tools : データ作成, モデル作成, 評価の為のコード
- “./train\_vlstm\_model.py, train\_pair\_model.py”などを実行するとネットワークが “./models” 以下に作成されます。
- models内には既に学習済みにmodelが置いてあります。
- data内には学習に必要な訓練データが置いてあります。
- データの作成は “./make\_vlstm\_data.py, make\_pair\_data.py” などの実行で作成できますが、ディレクトリの依存関係があるかもしれません。
- “./plot\_\*.py, eval\_\*.py” は修論作成に使用したコード類です。
- GPUを共同で使用する場合は必ずメモリの制限を行いましょう。

```
7 physical_devices = tf.config.experimental.list_physical_devices('GPU')
8 if len(physical_devices) > 0:
9     for k in range(len(physical_devices)):
10        tf.config.experimental.set_memory_growth(physical_devices[k], True)
11        print('memory growth:', tf.config.experimental.get_memory_growth(physical_devices[k]))
12 else:
13     print("Not enough GPU hardware devices available")
```

# 3. Deep Learning

## Feedforward Network

```
20 def PairModelTraining(model, model_name, x_train, vertex_train, position_train, BATCH_SIZE=1024, NB_EPOCHS=2500, VALIDATION_SPLIT=0.2, LR=0.001,
21                         Custom_Weights=[0.0090, 0.0175, 0.3375, 0.1800, 0.3509, 0.1260, 1.0], Loss_Weights=[0.5, 0.5]):
22     # Tensor Board
23     set_dir_name = 'PairTensorBoard'
24     set_dir = os.path.join(os.path.abspath(os.path.dirname(__file__)), "../../log/" + set_dir_name)
25     if not os.path.exists(set_dir):
26         os.mkdir(set_dir)
27     tictoc = strftime("%Y%m%d%H%M", gmtime())
28     directory_time = tictoc
29     log_dir = os.path.join(os.path.abspath(os.path.dirname(__file__)), "../../log/" + set_dir_name + "/" + model_name + directory_time)
30     if not os.path.exists(log_dir):
31         os.mkdir(log_dir)

32             custom object scopeを使ってcustom loss functionを読み込む
33
34     with CustomObjectScope({'custom_categorical_crossentropy': loss.custom_categorical_crossentropy(Custom_Weights)}):
35         model.compile(loss={'Vertex_Output': 'custom_categorical_crossentropy', 'Position_Output': 'mean_squared_logarithmic_error'},
36                         loss_weights={'Vertex_Output':Loss_Weights[0], 'Position_Output':Loss_Weights[1]},
37                         optimizer=SGD(learning_rate=LR),
38                         metrics=['accuracy', 'mae'])

39     callbacks = [TensorBoard(log_dir=log_dir)]

40
41     history = model.fit(x_train, [vertex_train, position_train],
42                           batch_size=BATCH_SIZE,
43                           epochs=NB_EPOCHS,
44                           callbacks=callbacks,
45                           verbose=1,
46                           validation_split=VALIDATION_SPLIT)

47
48     return model, history
```

```
6 def custom_categorical_crossentropy(y_weight, from_logits=False, label_smoothing=0):
7
8     def custom_categorical_crossentropy_loss(y_true, y_pred):
9
```

# 3. Deep Learning

## Recurrent Network

```
8 def VLSTMModelSimple(pair, tracks, UNITS=256):  
9  
10    MAX_TRACK_NUM = tracks.shape[1]  
11    INPUT_DIM = tracks.shape[2]  
12    PAIR_DIM = pair.shape[1]  
13  
14    pair_input = Input(shape=(PAIR_DIM,), name='Pair_Input')  
15  
16    track_input = Input(shape=(None, INPUT_DIM), name='Input')  
17    track_embedd = TimeDistributed(Dense(UNITS, name='Embedding_Dense', activation='relu'))(track_input)  
18  
19    init_state = Dense(UNITS, name='Init_State_Dense_1')(pair_input)  
20    init_state = BatchNormalization(name='Init_State_BatchNorm_1')(init_state)  
21    init_state = Activation('relu', name='Init_State_Activation_1')(init_state)  
22    init_state = Dense(UNITS, name='Init_State_Dense_2')(init_state)  
23    init_state = BatchNormalization(name='Init_State_BatchNorm_2')(init_state)  
24    init_state = Activation('relu', name='Init_State_Activation_2')(init_state)  
25  
26    cell = layers.VLSTMCellSimple(UNITS, 1)  
27  
28    rnn = RNN(cell, return_sequences=True, name='Vertex_LSTM_Simple')(track_embedd, initial_state=[init_state, init_state])  
29  
30    model = Model(inputs=[pair_input, track_input], outputs=rnn)  
31  
32    model.summary()  
33  
34    return model
```

Simple Dedicated LSTM

Embedding

Trainable initial state

Custom RNNCellの呼び出し

# 3. Deep Learning

## Recurrent Network

- RNNCellはRNNの1 step分の演算を行います。
- RNNCellを用いることにより、様々な構造のRNNを作成することができます。
- RNNCellは以下の関数で成り立っています。
  - `_init_` : 関数の設定値
  - `state_size` : hidden state (initial state) のサイズ
  - `build` : 重み行列のサイズ/個数
  - `call` : 1 stepの演算
  - `get_config` : 読み出しの際に必要な設定値
  - `_XXX` : internalな演算
- 基本的には、`call`を適宜書き換え、重み行列を増やす場合は`build`に追記し、初期パラメータを追加する場合は`_init_`に書き込む  
`_init_`に書き込んだ場合は、必要であれば`get_config`にも追記するという感じで良いと思います。

# 3. Deep Learning

## Recurrent Network

```
12 class VLSTMCellSimple(AbstractRNNCell):
13
14     def __init__(self,
15                  units,
16                  units_out,
17                  activation='tanh',
18                  recurrent_activation='hard_sigmoid',
19                  dense_activation='sigmoid',
20                  use_bias=True,
21                  kernel_initializer='glorot_uniform',
22                  recurrent_initializer='orthogonal',
23                  bias_initializer='zeros',
24                  unit_forget_bias=True,
25                  kernel_regularizer=None,
26                  recurrent_regularizer=None,
27                  bias_regularizer=None,
28                  kernel_constraint=None,
29                  recurrent_constraint=None,
30                  bias_constraint=None,
31                  implementation=1,
32                  **kwargs):
33
34     super(VLSTMCellSimple, self).__init__(**kwargs)
35     self.units = units
36     self.units_out = units_out
37     self.activation = activations.get(activation)
38     self.recurrent_activation = activations.get(recurrent_activation)
39     self.dense_activation = activations.get(dense_activation)
40     self.use_bias = use_bias
41
42     self.kernel_initializer = initializers.get(kernel_initializer)
43     self.recurrent_initializer = initializers.get(recurrent_initializer)
44     self.bias_initializer = initializers.get(bias_initializer)
45     self.unit_forget_bias = unit_forget_bias
46
47     self.kernel_regularizer = regularizers.get(kernel_regularizer)
48     self.recurrent_regularizer = regularizers.get(recurrent_regularizer)
49     self.bias_regularizer = regularizers.get(bias_regularizer)
50
51     self.kernel_constraint = constraints.get(kernel_constraint)
52     self.recurrent_constraint = constraints.get(recurrent_constraint)
53     self.bias_constraint = constraints.get(bias_constraint)
54
55     if implementation != 1:
56         logging.debug(RECURRENT_DROPOUT_WARNING_MSG)
57         self.implementation = 1
58     else:
59         self.implementation = implementation
```

### \_\_init\_\_

```
175     def get_config(self):
176         config = {
177             'units':
178                 self.units,
179             'units_out':
180                 self.units_out,
181             'activation':
182                 activations.serialize(self.activation),
183             'recurrent_activation':
184                 activations.serialize(self.recurrent_activation),
185             'use_bias':
186                 self.use_bias,
187             'kernel_initializer':
188                 initializers.serialize(self.kernel_initializer),
189             'recurrent_initializer':
190                 initializers.serialize(self.recurrent_initializer),
191             'bias_initializer':
192                 initializers.serialize(self.bias_initializer),
193             'unit_forget_bias':
194                 self.unit_forget_bias,
195             'kernel_regularizer':
196                 regularizers.serialize(self.kernel_regularizer),
197             'recurrent_regularizer':
198                 regularizers.serialize(self.recurrent_regularizer),
199             'bias_regularizer':
200                 regularizers.serialize(self.bias_regularizer),
201             'kernel_constraint':
202                 constraints.serialize(self.kernel_constraint),
203             'recurrent_constraint':
204                 constraints.serialize(self.recurrent_constraint),
205             'bias_constraint':
206                 constraints.serialize(self.bias_constraint),
207             'implementation':
208                 self.implementation
209         }
210         base_config = super(VLSTMCellSimple, self).get_config()
211         return dict(list(base_config.items()) + list(config.items()))
```

### get\_config

# 3. Deep Learning

## Recurrent Network

```
60
61     @property
62     def state_size(self):
63         return [self.units, self.units]
64
65     #@tf_utils.shape_type_conversion
66     def build(self, input_shape): # definition of the weights
67         input_dim = input_shape[-1]
68         self.kernel = self.add_weight( # W
69             shape=(input_dim, self.units * 4), # "* 4" means "o, f, i, z"
70             name='kernel',
71             initializer=self.kernel_initializer,
72             regularizer=self.kernel_regularizer,
73             constraint=self.kernel_constraint)
74         self.recurrent_kernel = self.add_weight( # R
75             shape=(self.units, self.units * 4),
76             name='recurrent_kernel',
77             initializer=self.recurrent_initializer,
78             regularizer=self.recurrent_regularizer,
79             constraint=self.recurrent_constraint)
80         self.dense_kernel = self.add_weight( # Last Dense Kernel
81             shape=(self.units * 1, self.units_out),
82             name='dense_kernel',
83             initializer=self.recurrent_initializer,
84             regularizer=self.recurrent_regularizer,
85             constraint=self.recurrent_constraint)
86
87         if self.use_bias:
88             if self.unit_forget_bias:
89
90
91             name= bias ,
92             initializer=bias_initializer
```

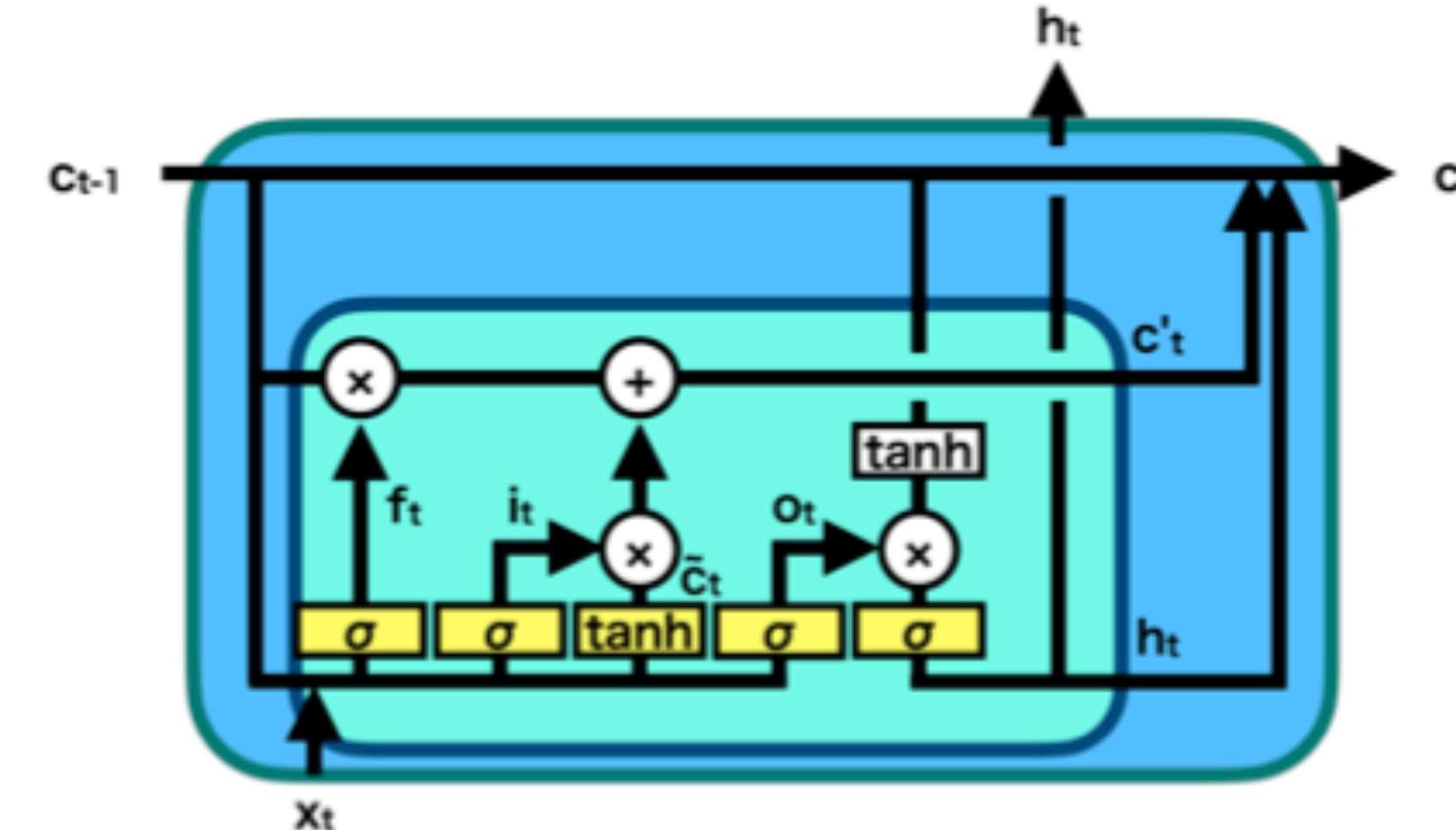
build

$W_{o, f, i, c}$

$R_{o, f, i, c}$

$d_h$

崩壊点生成のための  
リカレントニューラルネットワーク



$$c_t = (1 - h_t)c_{t-1} + h_t c'_t$$

$$c'_t = c_{t-1} \odot \sigma(W_f x_t + R_f c_{t-1}) + \tanh(W_c x_t + R_c c_{t-1}) \odot \sigma(W_i x_t + R_i c_{t-1}) \quad (3.4)$$

$$h_t = \sigma(d_h [\tanh(c_{t-1}) \odot \sigma(W_o x_t + R_o c_{t-1})])$$

# 3. Deep Learning

## Recurrent Network

```

137 def call(self, inputs, states, training=None):
138     V_tm1 = states[0] # previous Vertex state
139
140     if self.implementation == 1:
141         # input = track
142         inputs_i = inputs
143         inputs_f = inputs
144         inputs_c = inputs
145         inputs_o = inputs
146         # k = W
147         k_i, k_f, k_c, k_o = array_ops.split(
148             self.kernel, num_or_size_splits=4, axis=1)
149         x_i = K.dot(inputs_i, k_i)
150         x_f = K.dot(inputs_f, k_f)
151         x_c = K.dot(inputs_c, k_c)
152         x_o = K.dot(inputs_o, k_o)
153         if self.use_bias:
154             b_i, b_f, b_c, b_o = array_ops.split(
155                 self.bias, num_or_size_splits=4, axis=0)
156             x_i = K.bias_add(x_i, b_i)
157             x_f = K.bias_add(x_f, b_f)
158             x_c = K.bias_add(x_c, b_c)
159             x_o = K.bias_add(x_o, b_o)
160
161         V_tm1_i = V_tm1
162         V_tm1_f = V_tm1
163         V_tm1_c = V_tm1
164         V_tm1_o = V_tm1
165         V_tm1_o2 = V_tm1
166         V_tm1_u = V_tm1
167         V_tm1_v = V_tm1
168         x = (x_i, x_f, x_c, x_o)
169
170         V_tm1 = (V_tm1_i, V_tm1_f, V_tm1_c, V_tm1_o, V_tm1_o2, V_tm1_u, V_tm1_v)
171         h, V = self._compute_update_vertex(x, V_tm1)
172
173     return h, [V, V]

```

call

$V_{tm1} * : c_{t-1}$   
 $input * : x_t$

```

108     def _compute_update_vertex(self, x, V_tm1):
109         """Computes carry and output using split kernels."""
110         # x = W * track
111         x_i, x_f, x_c, x_o = x
112         V_tm1_i, V_tm1_f, V_tm1_c, V_tm1_o, V_tm1_o2, V_tm1_u, V_tm1_v = V_tm1
113         # i = x_i + V_tm1_i * R_i
114         #   = W_i * track + V_tm1_1 * R_i
115
116         i = self.recurrent_activation(
117             x_i + K.dot(V_tm1_i, self.recurrent_kernel[:, :self.units]))
118         f = self.recurrent_activation(
119             x_f + K.dot(V_tm1_f, self.recurrent_kernel[:, self.units:self.units * 2]))
120         c = self.activation(
121             x_c + K.dot(V_tm1_c, self.recurrent_kernel[:, self.units * 2:self.units * 3]))
122
123         # U = update vertex
124         U = f * V_tm1_u + i * c ]  $U : c'_{t-1}$ 
125
126         o = self.recurrent_activation(
127             x_o + K.dot(V_tm1_o, self.recurrent_kernel[:, self.units * 3:]))
128
129         h_temp = o * self.activation(V_tm1_o2)
130         # h size [self.units]
131         h = self.dense_activation(K.dot(h_temp, self.dense_kernel))
132         # h size [1] activated sigmoid
133
134         V = h * U + (1-h) * V_tm1_v
135         return h, V

```

$h : h_t$

$$c_t = (1 - h_t)c_{t-1} + h_t c'_t$$

$$c'_t = c_{t-1} \odot \sigma(W_f x_t + R_f c_{t-1}) + \tanh(W_c x_t + R_c c_{t-1}) \odot \sigma(W_i x_t + R_i c_{t-1}) \quad (3.4)$$

$$h_t = \sigma(d_h [\tanh(c_{t-1}) \odot \sigma(W_o x_t + R_o c_{t-1})])$$

ここでは、実装の都合上同じものを二つreturnしている  $[V, V]$

# 3. Deep Learning

## Recurrent Network

```
22 def VLSTMModelSimpleTraining(model, model_name, pair, tracks, labels, BATCH_SIZE=32, NB_EPOCHS=100, NB_SAMPLES=50000, VALIDATION_SPLIT=0.2, LR=0.001, pair_reinforce=False):
23
24     full_size = len(labels)
25     train_size = int(full_size*(1-VALIDATION_SPLIT))
26     Eindex = np.random.permutation(full_size)
27     pair_train, tracks_train, labels_train = pair[Eindex][:train_size], tracks[Eindex][:train_size], labels[Eindex][:train_size]
28     pair_valid, tracks_valid, labels_valid = pair[Eindex][train_size:], tracks[Eindex][train_size:], labels[Eindex][train_size:]
29
30     del pair, tracks, labels
31     gc.collect()
32
33     # Tensor Board
34     set_dir_name='VLSTMTensorBoard'
35     set_dir = os.path.join(os.path.abspath(os.path.dirname(__file__)), "../../log/" + set_dir_name)
36     if not os.path.exists(set_dir):
37         os.mkdir(set_dir)
38     tictoc = strftime("%Y%m%d%H%M", gmtime())
39     directory_time = tictoc
40     log_dir = os.path.join(os.path.abspath(os.path.dirname(__file__)), "../../log/" + set_dir_name + "/" + model_name + directory_time)
41     if not os.path.exists(log_dir):
42         os.mkdir(log_dir)
43
44     model.compile(loss=loss.binary_crossentropy(pair_reinforce=pair_reinforce),
45                   optimizer=Adam(lr=LR),
46                   metrics=[loss.accuracy_all, loss.accuracy,
47                             loss.true_positive, loss.true_negative, loss.false_positive, loss.false_negative])
48
49     callbacks = [TensorBoard(log_dir=log_dir)]
```

データの分割

Training/validation

custom loss function

Networks/VLSTMModel/loss.py

# 3. Deep Learning

## Recurrent Network

```
51 for epochs in range(NB_EPOCHS):
52     EindexTrain = np.random.permutation(len(labels_train))
53     EindexValid = np.random.permutation(len(labels_valid))
54     TindexTrain = np.random.permutation(len(labels_train[0]))
55     pair_train_use = pair_train[EindexTrain][:NB_SAMPLES]
56     pair_valid_use = pair_valid[EindexValid][:int(NB_SAMPLES*VALIDATION_SPLIT)]
57     tracks_valid_use = tracks_valid[EindexValid][:int(NB_SAMPLES*VALIDATION_SPLIT)]
58     labels_valid_use = labels_valid[EindexValid][:int(NB_SAMPLES*VALIDATION_SPLIT)]
59
60     shuffle_tracks_train = []
61     shuffle_labels_train = []
62     for t, l in zip(tracks_train[EindexTrain][:NB_SAMPLES], labels_train[EindexTrain][:NB_SAMPLES]):
63         shuffle_tracks_train.append(t[TindexTrain])
64         shuffle_labels_train.append(l[TindexTrain])
65     shuffle_tracks_train, shuffle_labels_train = np.array(shuffle_tracks_train), np.array(shuffle_labels_train)
66     print("======" + str(epochs+1) + "/" + str(NB_EPOCHS) + " epochs" +
67           " =====")
68
69     new_history = model.fit([pair_train_use, shuffle_tracks_train], shuffle_labels_train,
70                           batch_size=BATCH_SIZE,
71                           epochs=1,
72                           callbacks=callbacks,
73                           verbose=1,
74                           validation_data=[pair_valid_use, tracks_valid_use], labels_valid_use))
75
76     if epochs == 0:
77         history = {}
78     history = modeltools.appendHist(history, new_history.history)
79
80     del pair_train_use, shuffle_tracks_train, shuffle_labels_train
81     del pair_valid_use, tracks_valid_use, labels_valid_use
82     gc.collect()
83
84
85 return model, history
```

1 epoch毎に50000 sample選び  
系列順をシャッフル

historyに追記  
Networks/Tools/modeltools.py

# 3. Deep Learning

## Encoder-Decoder Model

- Encoder-Decoder Modelは基本的にSimple LSTMと同じですが、Modelの定義やRNN Cellが少し変わります。

```
65 def AttentionVLSTMModel(pair, tracks, ENCODER_UNITS=256, DECODER_UNITS=256):
66
67     MAX_TRACK_NUM = tracks.shape[1]
68     INPUT_DIM = tracks.shape[2]
69     PAIR_DIM = pair.shape[1]
70
71     pair_input = Input(shape=(PAIR_DIM,), name='Pair_Input')
72
73     encoder_input = Input(shape=(MAX_TRACK_NUM, INPUT_DIM), name='Encoder_Input')
74     encoder_embedd = TimeDistributed(Dense(ENCODER_UNITS, name='Encoder_EMBEDDING_DENSE', activation='relu'))(encoder_input)
75
76     decoder_input = Input(shape=(None, INPUT_DIM), name='Decoder_Input')
77     decoder_embedd = TimeDistributed(Dense(DECODER_UNITS, name='Decoder_EMBEDDING_DENSE', activation='relu'))(decoder_input)
78
79     encoder_init_state_f = Dense(ENCODER_UNITS, name='Encoder_Forward_Dense_1')(pair_input)
80     encoder_init_state_f = BatchNormalization(name='Encoder_Forward_BatchNorm_1')(encoder_init_state_f)
81     encoder_init_state_f = Activation('relu', name='Encoder_Forward_Activation_1')(encoder_init_state_f)
82     encoder_init_state_f = Dense(ENCODER_UNITS, name='Encoder_Forward_Dense_2')(encoder_init_state_f)
83     encoder_init_state_f = BatchNormalization(name='Encoder_Forward_BatchNorm_2')(encoder_init_state_f)
84     encoder_init_state_f = Activation('relu', name='Encoder_Forward_Activation_2')(encoder_init_state_f)
85
86     encoder_init_state_b = Dense(ENCODER_UNITS, name='Encoder_Backward_Dense_1')(pair_input)
87     encoder_init_state_b = BatchNormalization(name='Encoder_Backward_BatchNorm_1')(encoder_init_state_b)
88     encoder_init_state_b = Activation('relu', name='Encoder_Backward_Activation_1')(encoder_init_state_b)
89     encoder_init_state_b = Dense(ENCODER_UNITS, name='Encoder_Backward_Dense_2')(encoder_init_state_b)
90     encoder_init_state_b = BatchNormalization(name='Encoder_Backward_BatchNorm_2')(encoder_init_state_b)
91     encoder_init_state_b = Activation('relu', name='Encoder_Backward_Activation_2')(encoder_init_state_b)
92
93     decoder_init_state = Dense(DECODER_UNITS, name='Decoder_Dense_1')(pair_input)
94     decoder_init_state = BatchNormalization(name='Decoder_BatchNorm_1')(decoder_init_state)
95     decoder_init_state = Activation('relu', name='Decoder_Activation_1')(decoder_init_state)
96     decoder_init_state = Dense(DECODER_UNITS, name='Decoder_Dense_2')(decoder_init_state)
97     decoder_init_state = BatchNormalization(name='Decoder_BatchNorm_2')(decoder_init_state)
98     decoder_init_state = Activation('relu', name='Decoder_Activation_2')(decoder_init_state)
```

Embedding

Trainable initial state

# 3. Deep Learning

## Encoder-Decoder Model

```
100    vlstm_cell_f = layers.VLSTMCellEncoder(ENCODER_UNITS)
101    vlstm_cell_b = layers.VLSTMCellEncoder(ENCODER_UNITS)
102    encoder_f = RNN(vlstm_cell_f, return_sequences=True, name="Encoder_Forward_VLSTM", go_backwards=False)
103    encoder_b = RNN(vlstm_cell_b, return_sequences=True, name="Encoder_Backward_VLSTM", go_backwards=True)
104
105    with CustomObjectScope({"VLSTMCellEncoder": layers.VLSTMCellEncoder}):
106        biencoder = Bidirectional(encoder_f, backward_layer=encoder_b, name='Bidirectional_Encoder_VLSTM')(encoder_embedd,
107                                                                 initial_state=[encoder_init_state_f, encoder_init_state_f,
108                                                                 encoder_init_state_b, encoder_init_state_b])
109
110    biencoder = Reshape(target_shape=(MAX_TRACK_NUM*ENCODER_UNITS*2,), name='Reshape_Bidirectional_Encoder')(biencoder)
111
112    # DECODER_UNITS, ENCODER_UNITS, DECODER_OUTPUT, MAX_TRACK_NUM
113    attentionvlstm_cell = layers.AttentionVLSTMCell(DECODER_UNITS, ENCODER_UNITS*2, 1, MAX_TRACK_NUM)
114
115    decoder, attention = RNN(attentionvlstm_cell, return_sequences=True, name='Decoder_Attention_VLSTM')(decoder_embedd, initial_state=[biencoder, decoder_init_state])
116
117    model = Model(inputs=[pair_input, encoder_input, decoder_input], outputs=decoder)
118
119    model.summary()
120
121    return model
```

Bidirectional RNN

Attention VLSTM

エンコーダーの出力は、デコーダーの入力に使用する為に、一度平坦化する

# 3. Deep Learning

## Encoder-Decoder Model

- Encoder RNN Cellの違いは1 stepの演算の最後にdenseを使用していないという点のみです。
- Encoder/Decoder共にbuildなども書き換えている

### Simple

```
108 def _compute_update_vertex(self, x, V_tm1):  
109     """Computes carry and output using split kernels."""  
110     # x = W * track  
111     x_i, x_f, x_c, x_o = x  
112     V_tm1_i, V_tm1_f, V_tm1_c, V_tm1_o, V_tm1_o2, V_tm1_u, V_tm1_v = V_tm1  
113     # i = x_i + V_tm1_i * R_i  
114     #   = W_i * track + V_tm1_1 * R_i  
115  
116     i = self.recurrent_activation(  
117         x_i + K.dot(V_tm1_i, self.recurrent_kernel[:, :self.units]))  
118     f = self.recurrent_activation(  
119         x_f + K.dot(V_tm1_f, self.recurrent_kernel[:, self.units:self.units * 2]))  
120     c = self.activation(  
121         x_c + K.dot(V_tm1_c, self.recurrent_kernel[:, self.units * 2:self.units * 3]))  
122  
123     # U = update vertex  
124     U = f * V_tm1_u + i * c  
125  
126     o = self.recurrent_activation(  
127         x_o + K.dot(V_tm1_o, self.recurrent_kernel[:, self.units * 3:])))  
128  
129     h_temp = o * self.activation(V_tm1_o2)  
130     # h size [self.units]  
131     h = self.dense_activation(K.dot(h_temp, self.dense_kernel))  
132     # h size [1] activated sigmoid  
133  
134     V = h * U + (1-h) * V_tm1_v  
135     return h, V
```

### Encoder

```
306 def _compute_update_vertex(self, x, V_tm1):  
307     """Computes carry and output using split kernels."""  
308     x_i, x_f, x_c, x_o = x  
309     V_tm1_i, V_tm1_f, V_tm1_c, V_tm1_o, V_tm1_o2, V_tm1_u, V_tm1_v = V_tm1  
310  
311     i = self.recurrent_activation(  
312         x_i + K.dot(V_tm1_i, self.recurrent_kernel[:, :self.units]))  
313     f = self.recurrent_activation(  
314         x_f + K.dot(V_tm1_f, self.recurrent_kernel[:, self.units:self.units * 2]))  
315     c = self.activation(  
316         x_c + K.dot(V_tm1_c, self.recurrent_kernel[:, self.units * 2:self.units * 3]))  
317  
318     U = f * V_tm1_u + i * c  
319  
320     o = self.recurrent_activation(  
321         x_o + K.dot(V_tm1_o, self.recurrent_kernel[:, self.units * 3:])))  
322  
323     h = o * self.activation(V_tm1_o2)  
324  
325     V = h * U + (1-h) * V_tm1_v  
326     #the size of h is [self.units]  
327     return h, V
```

# 3. Deep Learning

## Encoder-Decoder Model

- Attention RNN CellはAttentionの為の演算が追加されています。

```

615 def call(self, inputs, states, training=None):
616     # store the whole sequence so we can "attend" to it at each timestep
617
618     att = states[0] # Attention input (track num, input dim) / key
619     self.x_seq = K.reshape(att, (-1, self.timestep, self.att_input_dim)) # Attention input (track num,
620     V_tm1 = states[1] # previous Vertex state (units)
621
622     # Additive Attention Bahdanau et al., 2015
623     self._uxpb = self._time_distributed_dense(self.x_seq,
624                                               self.attention_kernel_U,
625                                               b=self.attention_kernel_b,
626                                               input_dim=self.att_input_dim,
627                                               timesteps=self.timestep,
628                                               output_dim=self.units) _Wxtt : X_t U_query
629
630     # repeat the input track to the length of the sequence (track num, feature dim)
631     _tt = K.repeat(inputs, self.timestep) # inputs / query
632     _Wxtt = K.dot(_tt, self.attention_kernel_W)
633     et = K.dot(activations.tanh(_Wxtt + self._uxpb), K.expand_dims(self.attention_kernel_V)) # Energy
634
635 """
636 #Dot-Product Attention Luong et al., 2015 / Scaled Dot-Product Attention Vaswani 2017
637 self.x_seq /= np.sqrt(self.att_input_dim)
638 et = K.batch_dot(K.expand_dims(inputs), self.x_seq, axes=[1, 2])
639 et = K.reshape(et, (-1, self.timestep, 1))
640 """
641
642 at = K.exp(et)
643 at_sum = K.sum(at, axis=1)
644 at_sum_repeated = K.repeat(at_sum, self.timestep)
645 at /= at_sum_repeated # attention weights ({batchsize}, track num, 1)
646 context = K.squeeze(K.batch_dot(at, self.x_seq, axes=1), axis=1) # context

```

$x_{\text{seq}} : V, K$   
 $\_uxpb : K \ U_{\text{key}}$   
 $\_tt : X_t$   
 $\_Wxtt : X_t \ U_{\text{query}}$

```

548 @tf.function
549 def _time_distributed_dense(self, x, w, b=None, dropout=None,
550                             input_dim=None, output_dim=None,
551                             timesteps=None, training=None):
552     if input_dim is None:
553         input_dim = K.shape(x)[2]
554     if timesteps is None:
555         timesteps = K.shape(x)[1]
556     if output_dim is None:
557         output_dim = K.shape(w)[1]
558
559     if dropout is not None and 0. < dropout < 1.:
560         # apply the same dropout pattern at every timestep
561         ones = K.ones_like(K.reshape(x[:, 0, :], (-1, input_dim)))
562         dropout_matrix = K.dropout(ones, dropout)
563         expanded_dropout_matrix = K.repeat(dropout_matrix, timesteps)
564         x = K.in_train_phase(x * expanded_dropout_matrix, x, training=training)
565
566     # collapse time dimension and batch dimension together
567     x = K.reshape(x, (-1, input_dim))
568     x = K.dot(x, w)
569     if b is not None:
570         x = K.bias_add(x, b)
571     # reshape to 3D tensor
572     if K.backend() == 'tensorflow':
573         x = K.reshape(x, K.stack([-1, timesteps, output_dim]))
574         x.set_shape([None, None, output_dim])
575     else:
576         x = K.reshape(x, (-1, timesteps, output_dim))
577
578     return x

```

$$\begin{aligned}
\gamma_t &= \alpha_t V \\
\alpha_t &= (\alpha_{t,0}, \alpha_{t,1}, \alpha_{t,2}, \dots \alpha_{t,i}, \dots) \\
&= \left( \frac{\exp(e_{t,0})}{\sum_j \exp(e_{t,j})}, \frac{\exp(e_{t,1})}{\sum_j \exp(e_{t,j})}, \frac{\exp(e_{t,2})}{\sum_j \exp(e_{t,j})}, \dots \frac{\exp(e_{t,i})}{\sum_j \exp(e_{t,j})}, \dots \right) \\
e_t &= (K \ U_{\text{key}} + X_t \ U_{\text{query}}) \mathbf{u}_{\text{energy}}
\end{aligned}$$

# 3. Deep Learning

## Encoder-Decoder Model

- Attention RNN CellはAttentionの為の演算が追加されています。

```
579 def _compute_update_vertex(self, x, V_tm1, c):
580     """Computes carry and output using split kernels."""
581     x_i, x_f, x_c, x_o = x
582     V_tm1_i, V_tm1_f, V_tm1_c, V_tm1_o, V_tm1_o2, V_tm1_u, V_tm1_v = V_tm1
583     c_i, c_f, c_c, c_o = c
584
585     i = self.recurrent_activation(
586         x_i
587         + K.dot(V_tm1_i, self.recurrent_kernel[:, :self.units])
588         + K.dot(c_i, self.context_kernel[:, :self.units])) # Attention information
589
590     f = self.recurrent_activation(
591         x_f
592         + K.dot(V_tm1_f, self.recurrent_kernel[:, self.units:self.units * 2])
593         + K.dot(c_f, self.context_kernel[:, self.units:self.units * 2]))
594
595     c = self.activation(
596         x_c
597         + K.dot(V_tm1_c, self.recurrent_kernel[:, self.units * 2:self.units * 3])
598         + K.dot(c_c, self.context_kernel[:, self.units * 2:self.units * 3]))
599
600     U = f * V_tm1_u + i * c
601
602     o = self.recurrent_activation(
603         x_o
604         + K.dot(V_tm1_o, self.recurrent_kernel[:, self.units * 3:])
605         + K.dot(c_o, self.context_kernel[:, self.units * 3:]))
606
607     h_temp = o * self.activation(V_tm1_o2)
608     # h size [self.units]
609     h = self.dense_activation(K.dot(h_temp, self.dense_kernel))
610     # h size [1] activated sigmoid
611
612     V = h * U + (1-h) * V_tm1_v
613     return h, V
```

$$\begin{aligned} \mathbf{c}_t &= (1 - \mathbf{h}_t)\mathbf{c}_{t-1} + \mathbf{h}_t \mathbf{c}'_t \\ \mathbf{c}'_t &= \mathbf{c}_{t-1} \odot \sigma(W_f \mathbf{x}_t + R_f \mathbf{c}_{t-1} + C_f \boldsymbol{\gamma}_t) \\ &\quad + \tanh(W_c \mathbf{x}_t + R_c \mathbf{c}_{t-1} + C_c \boldsymbol{\gamma}_t) \odot \sigma(W_i \mathbf{x}_t + R_i \mathbf{c}_{t-1} + C_i \boldsymbol{\gamma}_t) \\ \mathbf{h}_t &= \sigma(\mathbf{d}_h [\tanh(\mathbf{c}_{t-1}) \odot \sigma(W_o \mathbf{x}_t + R_o \mathbf{c}_{t-1} + C_o \boldsymbol{\gamma}_t)]) \end{aligned}$$

# 3. Deep Learning

## Vertex Finder

- “./vertex\_finder.py” を実行するとvertex finderを行うことが出来ます。
- 結果は標準出力なので、リダイレクトしてください。

```
1 from VertexFinder import tools
2 import numpy as np
3
4
5 if __name__ == "__main__":
6
7     MaxEvent = 1000
8     MaxSample = -1
9     MaxTrack = 60
10    MaxPrimaryVertexLoop = 3
11    ThresholdPairSecondaryScoreBBCC = 0.6
12    ThresholdPairSecondaryScore = 0.88
13    ThresholdPairPosScore = 30
14    ThresholdPrimaryScore = 0.55
15    ThresholdSecondaryScore = 0.70
16    debug = False
17
18    NumPVEvent = 0
19    NumOthersEvent = 0
20    NumBBEvent = 0
21    NumCCEvent = 0
22
23    MCPrimaryRecoSV = 0
24    MCOthersRecoSV = 0
25
26    MCBottomRecoSV = 0
27    MCBottomRecoSVSameChain = 0
28    MCBottomRecoSVSameParticle = 0
29    MCCcharmRecoSV = 0
30    MCCcharmRecoSVSameChain = 0
31    MCCcharmRecoSVSameParticle = 0
32
33    MCPPrimaryRecoSVTrack = 0
34    MCOthersRecoSVTrack = 0
35    MCBottomRecoSVTrack = 0
36    MCBottomRecoSVSameChainTrack = 0
37    MCBottomRecoSVSameParticleTrack = 0
38    MCCcharmRecoSVTrack = 0
39    MCCcharmRecoSVSameChainTrack = 0
40    MCCcharmRecoSVSameParticleTrack = 0
41
42    NumPVTrack = 0
43    NumOthersTrack = 0
44    NumBBTrack = 0
45    NumCCTrack = 0
46
47    data_path = "data/numpy/Vertex_Finder_bb08_reshaped.npy"
48
49    pair_model_name = "Pair_Model_Standard"
50
51    lstm_model_name = "VLSTM_Model_Standard_PV"
52    slstm_model_name = "VLSTM_Model_Standard_SV"
53
54    print("Data Loading ...")
55    data = np.load(data_path)
56    variables = data[:MaxSample, 3:47]
57
58    print("Model Loading ...")
59    pair_model, lstm_model, slstm_model = tools.ModelsLoad(pair_model_name, lstm_model_name, slstm_model_name)
60
61    pred_vertex, pred_position = tools.PairInference(pair_model, variables)
62    data = np.concatenate([data[:MaxSample], pred_vertex], 1)
63    data = np.concatenate([data, pred_position], 1) # -8:NC, -7:PV, -6:SVCC, -5:SVBB, -4:TVCC, -3:SVBC, -2:Others, -1:Position
```

データ読み出し

Seed Finding

# 3. Deep Learning

## Vertex Finder

```
64
65     vertices_list = []
66     for ievent in range(MaxEvent): event毎のloop
67         print("====")
68         print("EVENT NUMBER " + str(ievent))
69         print("====")
70         event_data = [datum for datum in data if datum[0]==ievent]
71         event_data = np.array(event_data)
72         NTrack = (1 + np.sqrt(1 + 8*event_data.shape[0]))/2
73
74         if debug==True: print("The Number of Tracks in this event is " + str(NTrack))
75
76         # ===== #
77         # Making Tracks / True Labels ===== # Trackの作成/mask
78         # ===== #
79         encoder_tracks, decoder_tracks, true_label, chain_label = tools.GetEncoderDecoderTracksandTrue(debug, event_data, NTrack, MaxTrack)
80
81         # ===== #
82         # Secondary Seed Selection===== # Seed Selection
83         # ===== #
84         print("Secondary Seed Selection ...")
85         secondary_event_data = tools.SecondarySeedSelectionOne(debug, event_data, ThresholdPairSecondaryScore, ThresholdPairPosScore)
86         #secondary_seed_data = SecondarySeedSelectionTwo(debug, secondary_event_data, )
87
88         # ===== #
89         # Primary Vertex Finder ===== # Primary Vertex Production
90         # ===== #
91         print("Primary Vertex Prediction ...")
92         primary_track_list, bigger_primary_scores = tools.PrimaryVertexFinder(debug, MaxPrimaryVertexLoop, ThresholdPrimaryScore, event_data,
93                                     encoder_tracks, decoder_tracks, lstm_model)
94
95         # ===== #
96         # Secondary Vertex Finder ===== # Secondary Vertex Production
97         # ===== #
98         print("Secondary Vertex Prediction ...")
99         primary_track_list, secondary_track_lists = tools.SecondaryVertexFinder(debug, ThresholdSecondaryScore, bigger_primary_scores, primary_track_list, secondary_event_data,
100                                    encoder_tracks, decoder_tracks, slstm_model);
101
```

# 3. Deep Learning

## Vertex Finder

```
120 def SecondarySeedSelectionOne(debug, event_data, ThresholdPairSecondaryScore, ThresholdPairPosScore):  
121  
122     predict_vertex_labels = np.argmax(event_data[:, -8:-1], axis=1)  
123     secondary_event_data = []  
124     tmp_secondary_scores = []  
125     for event_datum, predict_vertex_label in zip(event_data, predict_vertex_labels):  
126         tmp_secondary_score = event_datum[-6] + event_datum[-5] + event_datum[-4] + event_datum[-3]  
127         if predict_vertex_label==0 or predict_vertex_label==1 or predict_vertex_label==6: continue  
128         if tmp_secondary_score < ThresholdPairSecondaryScore: continue  
129         if event_datum[-1] > ThresholdPairPosScore: continue  
130  
131         secondary_event_data.append(event_datum)  
132         tmp_secondary_scores.append(tmp_secondary_score)  
133  
134     tmp_secondary_scores = np.array(tmp_secondary_scores)  
135     secondary_event_data = np.array(secondary_event_data)  
136     index = np.argsort(-tmp_secondary_scores)  
137  
138     if debug==True:  
139         for i, secondary_event_datum in enumerate(secondary_event_data[index]):  
140             print("Secondary Seeds " + str(i) + " Track 1: " + str(secondary_event_datum[1]) + " Track 2: " + str(secondary_event_datum[2]) + "  
141                 + " SV Score: " + str(secondary_event_datum[-6] + secondary_event_datum[-5] + secondary_event_datum[-4] + secondary_event_datum[-3]))  
142  
143     return secondary_event_data[index]
```

Seed Selection

NC, PV, Othersではない  
スコアの和が閾値以上  
崩壊点の位置が閾値以下

# 3. Deep Learning

## Vertex Finder

```
149 def PrimaryVertexFinder(debug, MaxPrimaryVertexLoop, ThresholdPrimaryScore, event_data,
150                         encoder_tracks, decoder_tracks, lstm_model):
151
152     primary_track_list = []
153     bigger_primary_scores = []
154     primary_pairs = []
155     for event_datum in event_data[np.argsort(-event_data[:, -7])][:MaxPrimaryVertexLoop]: スコアの高い種にタネを選ぶ
156         primary_pairs.append(event_datum[3:47])
157     primary_encoder_tracks = np.tile(encoder_tracks, (MaxPrimaryVertexLoop, 1, 1)) # tracks.shape = (MaxPrimaryVertexLoop, MaxTrack, 23)
158     primary_decoder_tracks = np.tile(decoder_tracks, (MaxPrimaryVertexLoop, 1, 1)) # tracks.shape = (MaxPrimaryVertexLoop, NTrack, 23)
159     if debug==True: print("Primary Encoder Trach Shape " + str(primary_encoder_tracks.shape))
160     if debug==True: print("Primary Decoder Trach Shape " + str(primary_decoder_tracks.shape))
161     primary_scores = lstm_model.predict([primary_pairs, primary_encoder_tracks, primary_decoder_tracks])
162
163     for i in range(len(primary_scores[0])):
164         tmpbigger_primary_scores = 0
165         for j in range(len(primary_scores)):
166             score = primary_scores[j][i]
167             if tmpbigger_primary_scores < score: tmpbigger_primary_scores = score
168
169         bigger_primary_scores.append(tmpbigger_primary_scores)
170         if tmpbigger_primary_scores < ThresholdPrimaryScore: continue
171         if debug==True: print("Track " + str(i) + " Primary Score: " + str(tmpbigger_primary_scores)) primary vertexと見なす
172         primary_track_list.append(i)
173
174     return primary_track_list, bigger_primary_scores
```

Primary Vertex Production

Primary vertex prediction

Secondary vertexとの比較の為に  
各飛跡の最もスコアの大きいものを選ぶ

少なくとも一度以上、閾値以上であれば  
primary vertexと見なす

# 3. Deep Learning

## Vertex Finder

```
177 def SecondaryVertexFinder(debug, ThresholdSecondaryScore, bigger_primary_scores, primary_track_list, secondary_event_data,
178                         encoder_tracks, decoder_tracks, slstm_model):
179
180     track_list = np.arange(decoder_tracks.shape[0])
181     secondary_track_lists = []                                Secondary Vertexのタネのloop
182     for secondary_event_datum in secondary_event_data:
183         track1, track2 = secondary_event_datum[1], secondary_event_datum[2]
184         if (track1 in primary_track_list) or (track2 in primary_track_list): continue
185         if (track1 not in list(track_list)) or (track2 not in list(track_list)): continue
186
187         #if debug==True: print("Track List " + str(track_list))      Secondary Vertexのタネを構成している飛跡が
188
189         remain_decoder_tracks = decoder_tracks[track_list]           Primary Vertex / これまでのSecondary Vertexに入っていないか
190         secondary_pair = np.tile(secondary_event_datum[3:47], (1, 1)) # pair.shape = (1, 44)
191         secondary_encoder_tracks = np.tile(encoder_tracks, (1, 1, 1)) # tracks.shape = (1, MaxTrack, 23)
192         secondary_decoder_tracks = np.tile(remain_decoder_tracks, (1, 1, 1)) # tracks.shape = (1, RemainNTrack, 23)
193         secondary_scores = slstm_model.predict([secondary_pair, secondary_encoder_tracks, secondary_decoder_tracks])
194         secondary_scores = np.array(secondary_scores).reshape((-1, 1))
```

Decoder tracksには残っている飛跡を, Encoder tracksには全ての飛跡を使う

# 3. Deep Learning

## Vertex Finder

```
196     tmptrack_list = np.copy(track_list)
197     primary_track_list = np.array(primary_track_list, dtype=int)
198     tmpsecondary_track_list = []
199     for i, t in enumerate(track_list):          スコアが閾値以上
200         if secondary_scores[i] > ThresholdSecondaryScore:
201             if t not in primary_track_list:        Primary Vertexに含まれていない
202                 tmpsecondary_track_list.append(t)
203                 tmptrack_list = tmptrack_list[~(tmptrack_list == t)]      Primary Vertexに含まれていて、スコアが高い
204             elif (t in primary_track_list) and (secondary_scores[i] > bigger_primary_scores[t]):
205                 tmpsecondary_track_list.append(t)
206             if debug==True:
207                 print("Scramble Track Number " + str(t) + " SV Score " + str(secondary_scores[i]))
208                     + " PV Score " + str(bigger_primary_scores[t]))
209             primary_track_list = primary_track_list[~(primary_track_list == t)]
210             tmptrack_list = tmptrack_list[~(tmptrack_list == t)]
211             if len(tmpsecondary_track_list)!=0: secondary_track_lists.append(tmpsecondary_track_list)
212             track_list = np.copy(tmptrack_list)
213
214     return primary_track_list, secondary_track_lists
```

# 4.0. - 0.1. Keras2Tensorflow

- ・今回用いているTensorflowは2系と言われるversionであり、標準的なmodelの保存形式は “json / h5” となっています。 (恐らく)
  - ・jsonはネットワークの構造 / h5はネットワークの重みをそれぞれ保持しています。
- ・一方, Tensorflow C++ APIで読み込み可能な保存形式は ”pb” のみである為, C++ベースでmodelを動かす場合は json/h5 to pbを行う必要があります。
  - ・あるいはmodel保存時に予め, pb形式で保存しておけばこの様な手間は必要ありません。
- ・手法は簡単で, modelをロードした後,

```
tf.saved_model.save(model, model_path)
```

を実行すると, “model\_path/“ 以下に次のファイルが作成されます。
  - assets
  - saved\_model.pb
  - variables
- ・この他にもネットワークの構造によってはTensorRTというソフトを使用してより高速な推論ができる場合もあります。

# 4. LCFIPlus + DL

## LCFIPlus + DL

- これまで述べてきた様に, LCFIPlusはcmake/c++で書かれていますが, 対して, Tensorflowで作成したネットワークはPythonで書かれています。
- したがって, LCFIPlus上で vertex finder を動作させる為には, Pythonからc++へ環境を移さなければなりません。
- 具体的には, 以下の手順で実現します。
  1. Tensorflow C++ APIをinstallする
  2. Tensorflow C++ APIを用いて, c++上でvertex finderを動作させる
  3. LCFIPlusをinstallする際にTensorflowを認識させる
  4. LCFIPlus内にvertex finderのプロセッサを作成する
  5. Vertex finderをLCFIPlus / Marlinから動作させる
- 僕のGitHubのLCFIPlusはこのTensorflowが動作可能な状態になっている為, iLCSoftのみのbuildではうまく動作しない可能性があります。

# 4. LCFIPlus + DL

## Tensorflow C++ API

- Tensorflow C++ APIをinstallしましょう。
- Tensorflowは周辺ソフトのversionや環境に非常にセンシティブなので、(必ず) 動作環境のversionを揃える必要があります。
- まずはTensorflowをGitHubからダウンロードしましょう。  
以下から version 2.1.0 を探してきます。
  - <https://github.com/tensorflow/tensorflow/releases>
- Tensorflowのinstallには Bazel (v0.29.1) というソフトウェアが必要なので、これもインストールします。
  - <https://github.com/bazelbuild/bazel/releases?after=1.2.1>
- 更に、周辺環境も作っておきます。
  - Eigen v3.3.90 / Protobuf v3.8 をダウンロード/インストールします。
- 以下のコマンドを実行し、共有ライブラリを作成します。(cuda, cublasなどで引っかかるかもしれません)  
`./configure  
bazel build -c opt //tensorflow:libtensorflow_cc.so`

# 4. LCFIPlus + DL

## Tensorflow C++ API

- 作成されたファイルをコピーします。

```
cp bazel-genfiles/ /usr/local/include/tf/; cp tensorflow /usr/local/include/tf/;  
cp third_party /usr/local/include/tf/; cp bazel-bin/libtensorflow_cc.so /usr/local/lib/
```

- 動的ライブラリはcpでは動作しないかもしれません
-

# 4. LCFIPlus + DL

## C++でのネットワークの動作

- Tensorflow C++ APIでのmodelの動作は次の点順で行います。

- modelをロードする
- Tensor形式でデータの準備をする
- 推論を行う
- 得られた結果を確認する

使用するheaderファイル / cnpyは後述

```
11 #include <tensorflow/cc/saved_model/loader.h>
12 #include <tensorflow/cc/saved_model/tag_constants.h>
13 #include "cnpy.h"
```

- まず、modelのロードについて

```
419 std::cout << "Model Loading ..." << std::endl;
420
421 tensorflow::SessionOptions session_options = tensorflow::SessionOptions();
422 tensorflow::RunOptions run_options = tensorflow::RunOptions();
423 tensorflow::SavedModelBundleLite pair_model_bundle, pair_pos_model_bundle, lstm_model_bundle, slstm_model_bundle; model loadの為のoption
424
425 tensorflow::Status pair_status = LoadSavedModel(session_options, run_options, pair_path, {tensorflow::kSavedModelTagServe}, &pair_model_bundle);
426 tensorflow::Status pair_pos_status = LoadSavedModel(session_options, run_options, pair_pos_path, {tensorflow::kSavedModelTagServe}, &pair_pos_model_bundle);
427 tensorflow::Status lstm_status = LoadSavedModel(session_options, run_options, lstm_path, {tensorflow::kSavedModelTagServe}, &lstm_model_bundle);
428 tensorflow::Status slstm_status = LoadSavedModel(session_options, run_options, slstm_path, {tensorflow::kSavedModelTagServe}, &slstm_model_bundle); model load
```

# 4. LCFIPlus + DL

## C++でのネットワークの動作

- Tensorflowでの入力データはTensor型です。
  - Tensor型については自身で調べてください。
- また、ここまで使用していたNumpy形式のデータも扱いたい為、cnpv というライブラリを install します。
  - cnpvでのデータのロードはnpy\_load()という関数を使用します。

```
409 std::string npypath = "/home/goto/ILC/Deep_Learning/data/vfdnn/vfdnn_08_shap  
410 tensorflow::string pair_path = "/home/goto/ILC/Deep_Learning/model/Pair_Mode  
411 tensorflow::string pair_pos_path = "/home/goto/ILC/Deep_Learning/model/Pair_  
412 tensorflow::string lstm_path = "/home/goto/ILC/Deep_Learning/model/Attention  
413 tensorflow::string slstm_path = "/home/goto/ILC/Deep_Learning/model/Attention  
414  
415 std::cout << "Data Loading ..." << std::endl;  
416  
417 std::vector<std::vector<double> > data = LoadNpy(npypath), event_data;  
418
```

```
17 std::vector<std::vector<double> > LoadNpy(std::string path){  
18     cnpv::NpyArray arr = cnpv::npy_load(path);  
19     double* loaded_data = arr.data<double>();  
20     size_t nrows = arr.shape[0];  
21     size_t ncols = arr.shape[1];  
22  
23     std::vector<std::vector<double> > data;  
24  
25     std::cout << "Data shape row: " << nrows << " col: " << ncols << std::endl;  
26  
27     for(size_t row=0; row<nrows; row++){  
28         data.emplace_back(ncols);  
29         for(size_t col=0; col<ncols; col++){  
30             data[row][col] = loaded_data[row*ncols+col];  
31         }  
32     }  
33     return data;  
34 }
```

# 4. LCFIPlus + DL

## C++でのネットワークの動作

- 1入力、1出力のmodelの推論は次の様に行います。

```
430 std::cout << "Pair Prediction ..." << std::endl;
431
432 std::vector<std::vector<double> > variables = SliceN2DVector(data, 0, MaxSample, 3, 47);
433 tensorflow::Tensor tvariables = N2DVector2Tensor(variables);
434
435 std::vector<tensorflow::Tensor> tmppair_outputs;
436 tensorflow::Status pair_runStatus = pair_model_bundle.GetSession()->Run({{"serving_default_input_1:0", tvariables},
437                                         {"StatefulPartitionedCall:0"}, 
438                                         {}, &tmppair_outputs);
439 std::vector<std::vector<double> > labels = Tensor2N2DVector(tmppair_outputs[0]);
```

```
68 tensorflow::Tensor N2DVector2Tensor(std::vector<std::vector<double> > vec){
69     const int row = vec.size(), col = vec.at(0).size();
70     tensorflow::Tensor tensor = tensorflow::Tensor(tensorflow::DT_FLOAT, tensorflow::TensorShape({row, col}));
71     tensorflow::TTypes<float, 2>::Tensor tensor_element = tensor.tensor<float, 2>();
72     tensor_element.setZero();
73     for(int i=0; i<row; i++){
74         for(int j=0; j<col; j++){
75             tensor_element(i, j) = float(vec.at(i).at(j));
76         }
77     }
78     return tensor;
79 }
```

```
90 std::vector<std::vector<double> > Tensor2N2DVector(tensorflow::Tensor tensor){
91     const int row = tensor.dim_size(0), col = tensor.dim_size(1);
92     std::vector<std::vector<double> > vec(row, std::vector<double>(col));
93     tensorflow::TTypes<float, 2>::Tensor tensor_element = tensor.tensor<float, 2>();
94     for(int i=0; i<row; i++){
95         for(int j=0; j<col; j++){
96             vec.at(i).at(j) = double(tensor_element(i, j));
97         }
98     }
99     return vec;
100 }
```

# 4. LCFIPlus + DL

## C++でのネットワークの動作

- 1入力, 1出力のmodelの推論は次の様に行います。

```
430 std::cout << "Pair Prediction ..." << std::endl;
431
432 std::vector<std::vector<double> > variables = SliceN2DVector(data, 0, MaxSample, 3, 47);
433 tensorflow::Tensor tvariables = N2DVector2Tensor(variables);
434
435 std::vector<tensorflow::Tensor> tmppair_outputs;
436 tensorflow::Status pair_runStatus = pair_model_bundle.GetSession()->Run({{"serving_default_input_1:0", tvariables}},
437 {"StatefulPartitionedCall:0"}, 
438 {}, &tmppair_outputs);
439 std::vector<std::vector<double> > labels = Tensor2N2DVector(tmppair_outputs[0]);
```

(入力, 出力, {}, 出力vector)

- 入力と出力の名前は “saved\_model\_cli show --dir “model\_path” --all” の出力から確認することができます。
- 複数出力の場合は, “出力”を{“出力名:0”, “出力名:1”}として, “出力vector” の[0], [1]を指定することで推論を得ることができます。

```
297 tensorflow::Tensor tvariables = VertexFinderwithDL::N2DVector2Tensor(variables);
298 std::vector<tensorflow::Tensor> tmppair_outputs;
299 tensorflow::Status pair_runStatus = pair_model_bundle.GetSession()->Run({{"serving_default_Pair_Input:0", tvariables}},
300 {"StatefulPartitionedCall:0", "StatefulPartitionedCall:1"}, 
301 {}, &tmppair_outputs);
302 std::vector<std::vector<double> > _labels = VertexFinderwithDL::Tensor2N2DVector(tmppair_outputs[1]);
303 std::vector<std::vector<double> > _positions = VertexFinderwithDL::Tensor2N2DVector(tmppair_outputs[0]);
```

# 4. LCFIPlus + DL

## C++でのネットワークの動作

- 複数入力の場合は，“入力”を{{“入力名:0”, 入力Tensor}, {"入力名:0", 入力Tensor}, {"入力名:0", 入力Tensor}}とします。

```
306 tensorflow::Tensor tprimary_pairs = N2DVector2Tensor(primary_pairs);
307 tensorflow::Tensor tprimary_encoder_tracks = N3DVector2Tensor(primary_encoder_tracks);
308 tensorflow::Tensor tprimary_decoder_tracks = N3DVector2Tensor(primary_decoder_tracks);
309
310 std::vector<tensorflow::Tensor> tmpprimary_outputs;
311 tensorflow::Status runStatus = lstm_model_bundle.GetSession()->Run({{"serving_default_Decoder_Input:0", tprimary_decoder_tracks},
312 {"serving_default_Encoder_Input:0", tprimary_encoder_tracks},
313 {"serving_default_Pair_Input:0", tprimary_pairs}},
314 {"StatefulPartitionedCall:0"}, 
315 {}, &tmpprimary_outputs);
316
317 primary_scores = Tensor2N3DVector(tmpprimary_outputs[0]);
```

- コンパイルは

**g++ -std=c++11 -I/…/tf -I/…/eigen3 -L/…/lib -ltensorflow\_cc -ltensorflow\_framework**

で良いと思います

- もしくは **cnpv** を使用する場合は

**-lcnpv -lz**

などを適宜追加してください

# 4. LCFIPlus + DL

## CMakeによるLCFIPlusへの導入

- Tensorflowが問題なく動作していることを確認できたら、  
次はTensorflowをCmake環境の中で動作させます。
- CMakeは "CMakeList.txt" を用いてパッケージを管理しています。  
ここでは、この "CMakeList.txt" を書き換えて、Tensorflow を含めたbuildを行います。
- 今回は既に、Tensorflow C++ APIの共有ライブラリを持っている為、  
単に以下の様にスクリプトを書き加え、"Tensorflow, Protobuf, Eigen3" を認識させます。
  - 他にも細々とした部分を書き換える必要があります。詳細は"GitHub:LCFIPlus"を確認してください。

```
22 FIND_PACKAGE( ILCUTIL COMPONENTS ILC_SOFT_CMAKE_MODULES REQUIRED )
23
24 # load default settings from ILC_SOFT_CMAKE_MODULES
25 INCLUDE( ilcsoft_default_settings )
26
27
28 FIND_PACKAGE( Marlin 1.0 REQUIRED )
29 FIND_PACKAGE( MarlinUtil REQUIRED )
30 #FIND_PACKAGE( ROOT REQUIRED COMPONENTS Minuit2 TMVA TreePlayer )
31 FIND_PACKAGE( ROOT REQUIRED COMPONENTS Minuit2 TMVA TreePlayer Gui Geom Eve Minuit XMLIO RGL Ged EG MLP )
32 FIND_PACKAGE( LCFIVertex REQUIRED )
33
34 ######
35 # add tensorflow by Goto
36 LIST(APPEND CMAKE_MODULE_PATH ${PROJECT_SOURCE_DIR}/cmake)
37 FIND_PACKAGE( Tensorflow REQUIRED )
38 FIND_PACKAGE( Protobuf REQUIRED )
39 FIND_PACKAGE( Eigen3 REQUIRED )
40 INCLUDE_DIRECTORIES( SYSTEM ${Tensorflow_INCLUDE_DIRS} )
41 INCLUDE_DIRECTORIES( SYSTEM ${Protobuf_INCLUDE_DIRS} )
42 ADD_DEFINITIONS( ${Tensorflow_DEFINITIONS} )
43 INCLUDE_DIRECTORIES( SYSTEM ${EIGEN3_INCLUDE_DIRS} )
44 INCLUDE_DIRECTORIES( /home/goto/local/include/eigen3 )
45 ######
```

```
-- Found LCFIVertex: /gluster/data/ilc/ilcsoft/v02-02/LCFIVertex/v00-08
-- Found Tensorflow: /home/goto/local/include/tf
-- Found Protobuf:
-- Found Eigen3: /home/goto/local/include/eigen3 (Required is at least version "2.91.0")
-- Check for ROOT_CINT_EXECUTABLE: /gluster/data/ilc/ilcsoft/v02-02/root/6.18.04/bin/rootcint
-- Check for ROOT_DICT_OUTPUT_DIR: /home/goto/ILC/LCFIPlus/build/rootdict
-- Check for ROOT_DICT_CINT_DEFINITIONS:
-- Found Doxygen: /usr/bin/doxygen (found version "1.8.14") found components: doxygen dot
--
```

# 4. LCFIPlus + DL

## LCFIPlus上でのネットワークの動作

- 最後にLCFIPlus上でネットワークを動作させます。
- 具体的には、Tensorflowをincludeしたプロセッサーを作成し、推論を行います。
- 新しいプロセッサーの作成は、“2.2 LCFIPlus Processor”で行いました。
- 今回は同様の手法で、プロセッサーの作成を行い、`<tensorflow/cc/hoge.h>` ファイルをincludeします。
- 基本的にはこれまで行ったプロセッサーの作成にC++でのネットワークの動作を組み合わせれば完成します。
  - modelの推奨動作方法は `init` で modelのロードを行い、`process` で推論を行いましょう。
  - また、GPUメモリ確保も行ったほうが良いでしょう。
  - 詳細は "GitHub:LCFIPlus/src/\*\*\*/withDL.cc" を確認してください。

```
79 session_options = tensorflow::SessionOptions();    モデルのOption 宣言はheaderに      init
80 run_options = tensorflow::RunOptions();
81
82 session_options.config.mutable_gpu_options()->set_visible_device_list("0");
83 //session_options.config.mutable_gpu_options()->set_per_process_gpu_memory_fraction(0.2);
84 session_options.config.mutable_gpu_options()->set_allow_growth(true);      メモリの制限
85
86 tensorflow::Status pair_status = LoadSavedModel(session_options, run_options, pair_path, {tensorflow::kSavedModelTagServe}, &pair_model_bundle);
87 tensorflow::Status lstm_status = LoadSavedModel(session_options, run_options, lstm_path, {tensorflow::kSavedModelTagServe}, &lstm_model_bundle);
88 tensorflow::Status slstm_status = LoadSavedModel(session_options, run_options, slstm_path, {tensorflow::kSavedModelTagServe}, &slstm_model_bundle);      モデルのロード、宣言もccに
```

# 5. やり残した事

- ・以上が引継ぎ資料の本文です。
    - ・ネットワークを作り、現行のフレームワークで動かすという点では、どの様なアルゴリズムであっても共通していると思います。
  - ・不明なことがあれば、[goto@epp.phys.kyushu-u.ac.jp](mailto:goto@epp.phys.kyushu-u.ac.jp)などで聞いてください。
  - ・ここからは、vertex finderでやり残したことなどを挙げていきます。
- 
- ・Attentionのミス
    - ・AttentionのKeyとQueryを完全に同じものにしてしまっている。  
本来はエンコーダーの出力をdenseで変換しなければならない。
  - ・損失関数への物理量の導入
    - ・損失関数にカットベースからの重みを加える。
    - ・質量などを逐次計算し、それによる罰則を与える。
    - ・ついでに、最後の隠れ状態でb/cフレーバーの推論を行えば更に良くなるはず。
  - ・TensorRTによる高速化
    - ・TensorRTが、一部のRNN系に対応したので、高速化が適応できるか試してみたかった。
  - ・ペアについてのネットワークの改良
    - ・距離を用いた手法
    - ・畳み込みを用いた手法