Neural Network Algorithm for reconstruction of diffused track data in NEWAGE0.3b'
e-logbook for analysis

2018/05/02

Hiroshi Ito

****

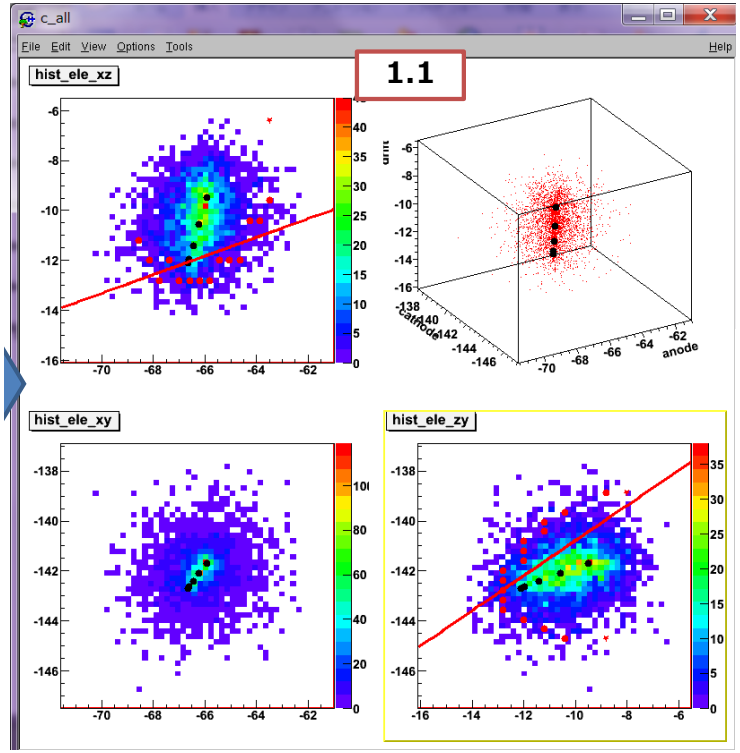
Motivation

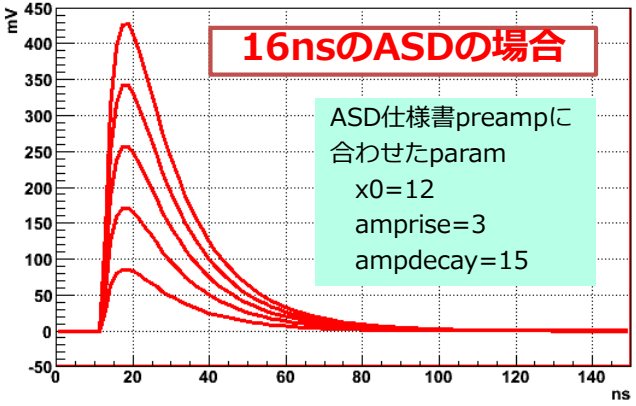
****

- 0.3b‘ Cf run analysis: skewnessXYによるhead-tail決定解析にて、skx-mxzに相関関係があることが判明
- F-recoil trackは短くdriftによって飛跡がボケて傾きmxzを算出することが困難

- フィットは一致しないが傾向だけつかめていると判断してた。
- 従来はこの不完全さをz補正で修正してきた。
- 傾きを正確に決定できるだけでも感度は飛躍的に改善される
- 傾き決定されればhead-tail決定にも恩恵がある

Nakamura simu e-log





****

README

****

狙いはNEWAGE 0.3b'のディフジョンされた飛跡データからオリジナルの飛跡を再構成するアルゴリズムを開発する。ステップごとに深層学習アルゴリズムを構築する。

1) まずニューラルネットワークのソースを解読しよう
2) 1次元で深層学習を成功させる
3) 2次元における深層学習を成功させる
4) 簡易な飛跡データのディフュージョンを入れて深層学習に組み込む
5) NEWAGE Cf runのデータを学習させる
6) 学習のloss関数、深層度、conv2Dなどの適正化

macOS python keras tensorflow環境設定

機種 Mac Book Air 7.1 2015 年モデル
プロセッサ 1.6 GHz Intel Core i5
メモリ 4GB 1600 MHz DDR3
グラフィックス Intel HD Graphics 6000 1536 MB

```
$brew install python3
$pip3.6 install virtualenv
$brew install pyenv
```

anaconda3-5.1.0-MacOS_64.pkgをダウンロード
>> https://repo.continuum.io/archive/

```
$pyenv rehash
$pyenv global anaconda3-5.0.1
$echo 'export PATH="$PYENV_ROOT/versions/anaconda3-5.0.1/bin/:$PATH"' >> ~/.bashrc
$source ~/.bash_profile
$conda update conda
$conda create -n py3 python=3.6 numpy scipy pandas jupyter
```

- TensorFlow

```
$pip install --upgrade virtualenv
$pip install tensorflow
```

- Theano

```
$pip install parameterized
```

```
$pip install keras
```

1) まずニューラルネットワークのソースを
解読しよう (python3x)

mnist.py

テンプレートソース

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()


x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')




y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

データ読み込み

配列変更: 3次元から
2次元へ

教師データを入力

Trainが学習用データ
Testが評価用データ
このサンプルは自動でtrain:test=9:1にランダムで振り分けてくれる

1) まずニューラルネットワークのソースを
解読しよう (python3x)

mnist.py

ニューラルネットワークの
模型を構築する

```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))

model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test)
                    )
```

1) まずニューラルネットワークのソースを
解読しよう (python3x)

mnist.py

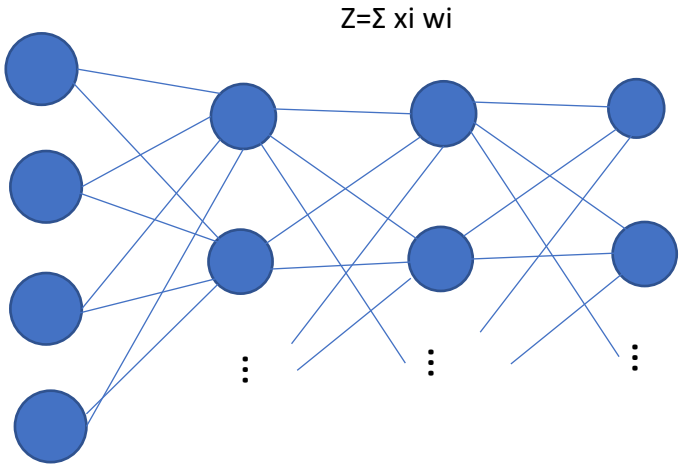
ニューラルネットワークの
模型を構築する

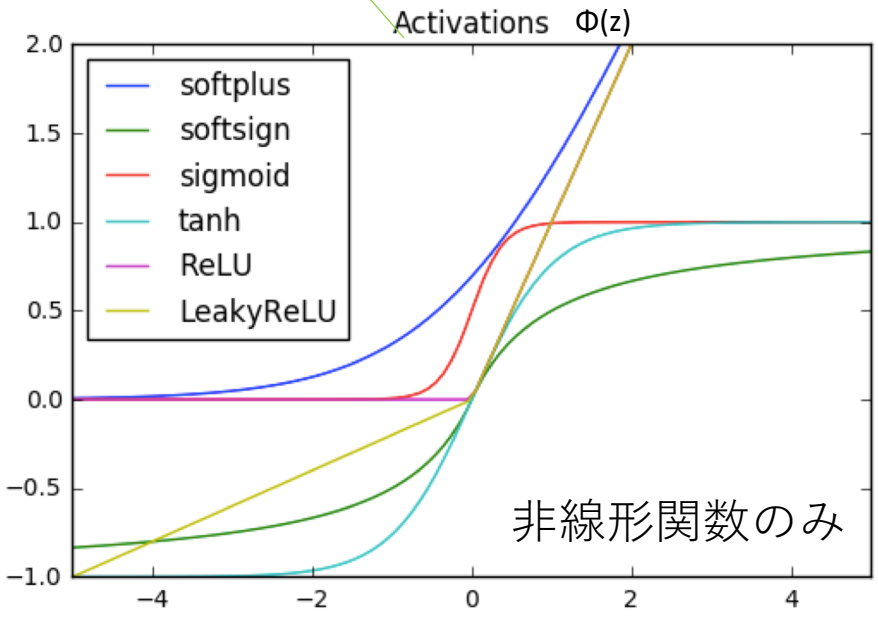
```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))

model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
```

マスクして深層学習させる

最後はカテゴリーの数出力

Dense





mnist.py

1) まずニューラルネットワークのソースを
解読しよう (python3x)

a) 損失関数を計算するときの関数
categorical crossentropy, mean_squared_error, mean_absolute_error
mean_absolute_error, mean_absolute_percentage_error, …

b)フィット収束方法
RMSprop(), Adam(), sgd… <= オススメはAdamらしい

c) Batch size: keras fitは一つのバッチのデータの和で学習する、一つのバッチのデータ数

d) 何回の学習することで精度をあげる(過学習も起こりうるので注意)。その回数。

```
model.compile(loss='categorical_crossentropy',  a)
              optimizer=RMSprop(),  <-- b)
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,  <-- c)
                    epochs=epochs,  <-- d)
                    verbose=1,
                    validation_data=(x_test, y_test)
                    )
```

1) まずニューラルネットワークのソースを
解読しよう (python3x)

mnist.py

こんな感じになったら成功！

```
$ python mnist_mlp.py

Using TensorFlow backend.
60000 train samples
10000 test samples
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               401920
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/2
2018-05-02 02:42:38.515563: I tensorflow/core/platform/cpu_feature_guard.cc:140]
Your CPU supports instructions that this TensorFlow binary was not compiled to use:
AVX2 FMA
29440/60000 [=============>................] - ETA: 5s - loss: 0.3472 - acc:
0.8941
```

1) まずニューラルネットワークのソースを
解読しよう (python3x)

num.py

```
import sys
import numpy as np
np.random.seed(20160715)


from keras.datasets import mnist
from keras.utils import np_utils


(X_train, y_train), (X_test, y_test) = mnist.load_data()


ran=1
for xs in X_train[ran]:
    for x in xs:
        sys.stdout.write('%03d ' % x)
    sys.stdout.write('¥n')


print('first sample is %d' % y_train[ran])


Y_train = np_utils.to_categorical(y_train, 10)


sys.stdout.write('[')
for y in Y_train[0]:
    sys.stdout.write('%f ' % y)
sys.stdout.write(']¥n')
```

データ読み込み

データ配列格納

結果出力？

結果表示

1) まずニューラルネットワークのソースを
解読しよう (python3x)

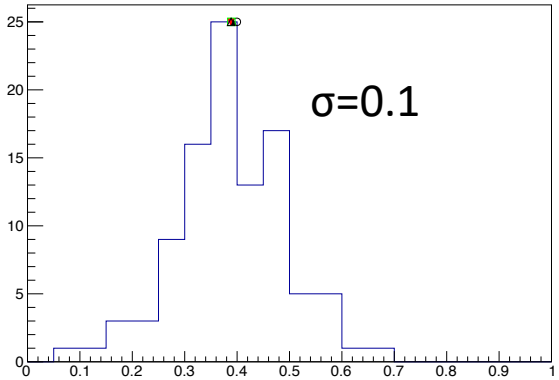
num.py

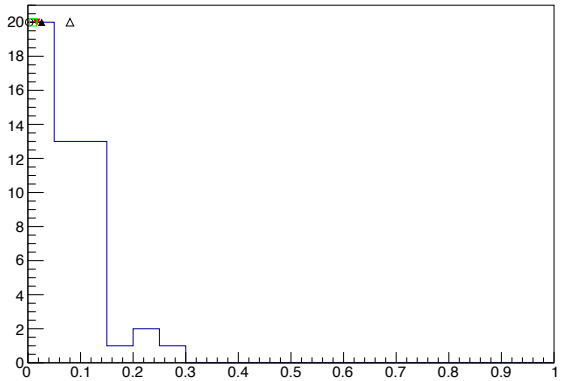
こんなの出たら成功

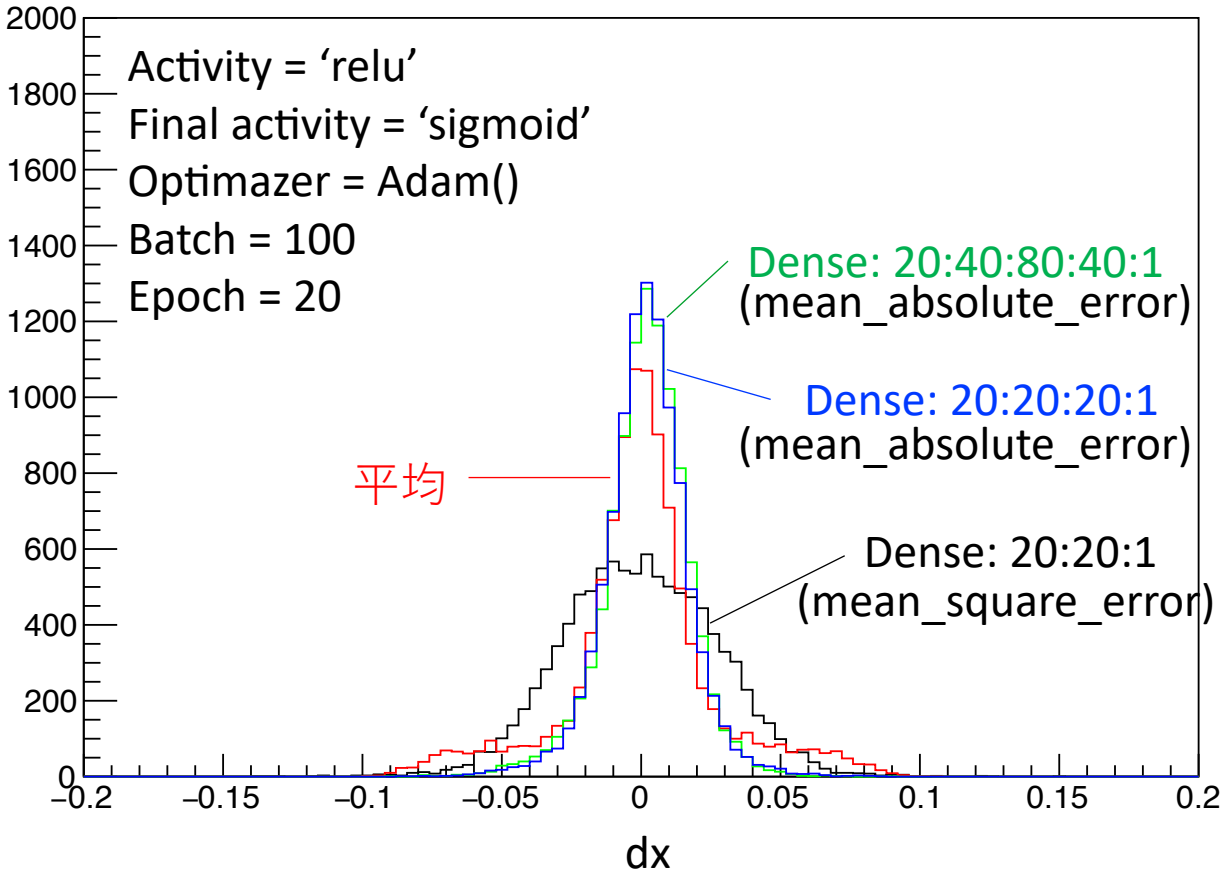
```
$ python num.py

/usr/local/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning: Conversion of the second argument of
issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 ==
np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 051 159 253 159 050 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 048 238 252 252 252 237 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 054 227 253 252 239 233 252 057 006 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 010 060 224 252 253 252 202 084 252 253 122 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 163 252 252 252 253 252 252 096 189 253 167 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 051 238 253 253 190 114 253 228 047 079 255 168 000 000 000 000 000
000 000 000 000 000 000 000 000 000 048 238 252 252 179 012 075 121 021 000 000 253 243 050 000 000 000 000
000 000 000 000 000 000 000 000 038 165 253 233 208 084 000 000 000 000 000 000 253 252 165 000 000 000 000
000 000 000 000 000 000 000 007 178 252 240 071 019 028 000 000 000 000 000 000 253 252 195 000 000 000 000
000 000 000 000 000 000 000 057 252 252 063 000 000 000 000 000 000 000 000 000 253 252 195 000 000 000 000
000 000 000 000 000 000 000 198 253 190 000 000 000 000 000 000 000 000 000 000 255 253 196 000 000 000 000
000 000 000 000 000 000 076 246 252 112 000 000 000 000 000 000 000 000 000 000 253 252 148 000 000 000 000
000 000 000 000 000 000 085 252 230 025 000 000 000 000 000 000 000 000 007 135 253 186 012 000 000 000 000
000 000 000 000 000 000 085 252 223 000 000 000 000 000 000 000 000 000 007 131 252 225 071 000 000 000 000
000 000 000 000 000 000 085 252 145 000 000 000 000 000 000 000 000 048 165 252 173 000 000 000 000 000 000
000 000 000 000 000 000 086 253 225 000 000 000 000 000 000 000 114 238 253 162 000 000 000 000 000 000 000
000 000 000 000 000 000 085 252 249 146 048 029 085 178 225 253 223 167 056 000 000 000 000 000 000 000 000
000 000 000 000 000 000 085 252 252 252 229 215 252 252 252 196 130 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 028 199 252 252 253 252 252 233 145 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 025 128 252 253 252 141 037 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
first sample is 0
[0.000000 0.000000 0.000000 0.000000 0.000000 1.000000 0.000000 0.000000 0.000000 0.000000 ]
```

2) 1次元で深層学習を成功させる

ニューラルちゃんが動作しているかチェックする
アイデアとしては、０～１まで一様乱数で教師値を決定
その後ガウスで振って100エントリー、20ビンのヒストグラムを生成
配列20のデータを10,000データ学習させて、元の値を返すかチェックする。
ライバルはヒストグラム平均値

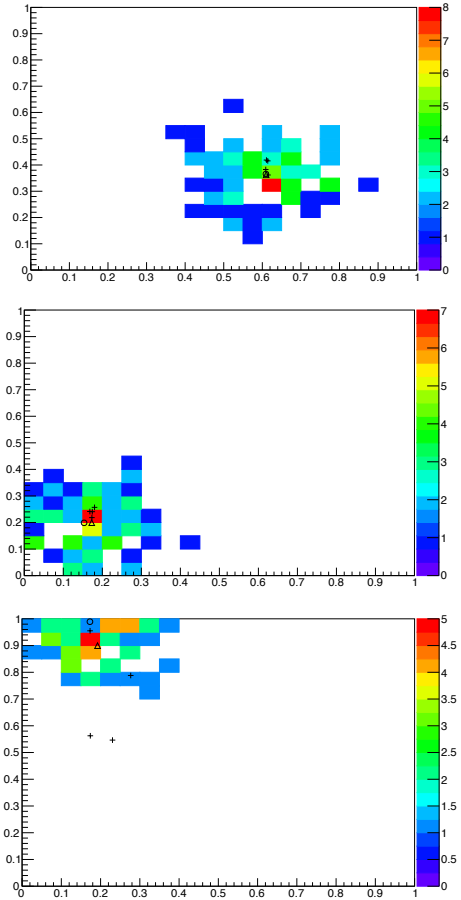


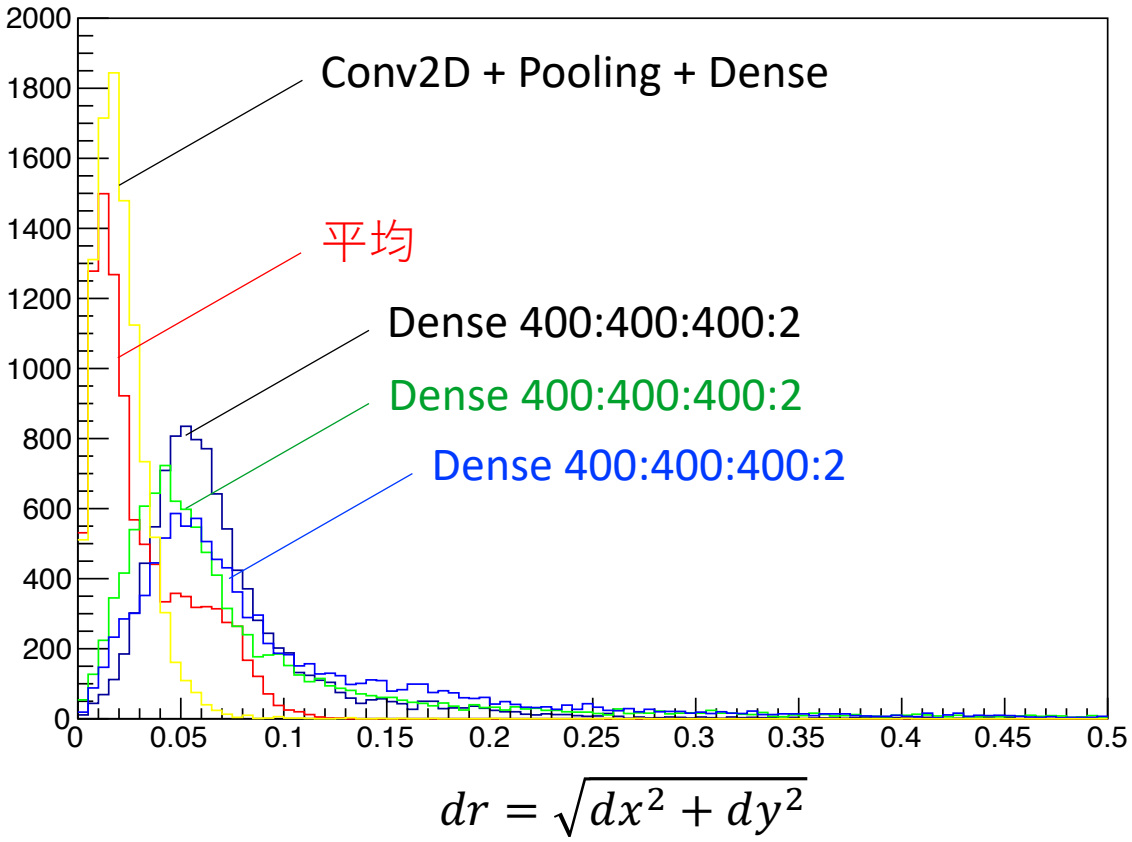




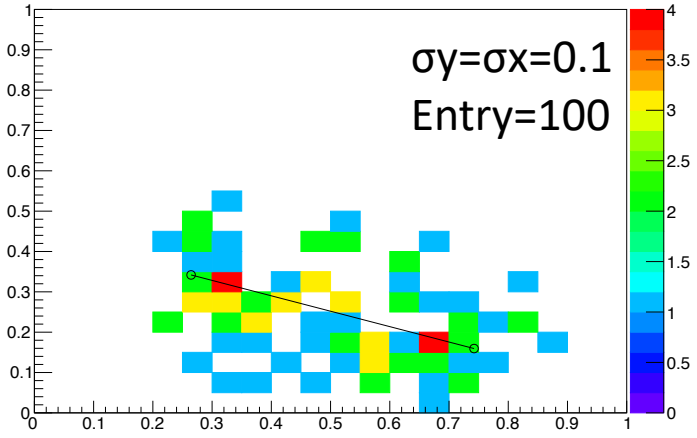
3) 2次元における深層学習を成功させる

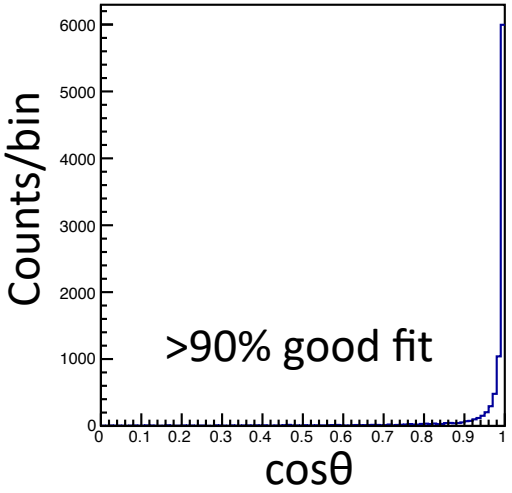
今度は、x=0～1, y=0～1の一様乱数で2次元値を決定
その後縦横ガウスで振って100エントリー、20×20ビンのヒストグラムを生成
配列400のデータを10,000データ学習させて、元の値を返すかチェックする。
ライバルはヒストグラム平均値

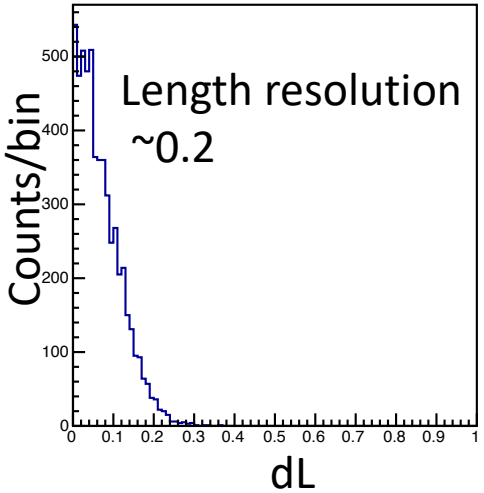




4) 簡易な飛跡データのディフュージョンを入れて深層学習に組み込む

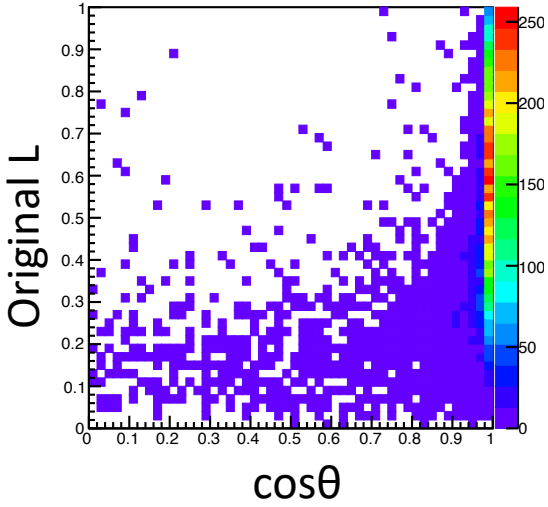


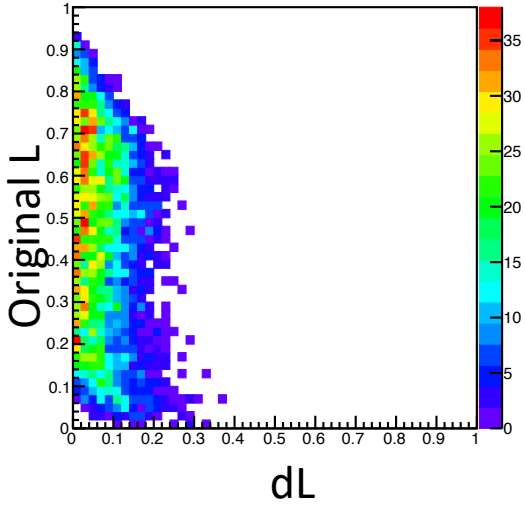




Model
- (20x20)x9,000
- Conv2D 16x(10x10), relu
- Conv2D 32x(5x5), relu
- Pooling (2,2)
- Flatten
- Dense(100), relu
- Dense(3), sigmoid

Loss=mean_absolute_error
Optimazer=Adam()
Batch=100, Epoch=10





****

Progress

****

狙いはNEWAGE 0.3b'のディフジョンされた飛跡データからオリジナルの飛跡を再構成するアルゴリズムを開発する。ステップごとに深層学習アルゴリズムを構築する。

1) まずニューラルネットワークのソースを解読しよう
2) 1次元で深層学習を成功させる
3) 2次元における深層学習を成功させる
4) 簡易な飛跡データのディフュージョンを入れて深層学習に組み込む
4.1) 3D飛跡データのXZ, YZ射影から再構成
4.2) Energyデータもどきを追加
4.x) SL6.9 OS @ susy11にkeras環境構築
5) NEWAGE Cf runのデータを学習させる
6) 学習のloss関数、深層度、conv2Dなどの適正化