Utilizing Neural Networks to Determine Event Tracks in a Particle Collider

Peter Larsen[1,+,*] **and Cameron Marcus**[1,-,*]

[1]University of Vermont, Burlington, VT
[+]plarsen@uvm.edu
[-]cmarcus@uvm.edu
[*]these authors contributed equally to this work

ABSTRACT

Using a benchmark clustering algorithm, we investigate whether a neural network can improve the classification time and accuracy of tracks resulting from events in a particle collider. The present document represents our final report for UVM Computer Science Course 254: Machine Learning.

Introduction

When physicists wish to look inside an particle, they cannot cut it. Instead, they force their way inside by smashing or "colliding" the particle in a Collider, such as the Large Hadron Collider (or LHC)[1]. These collisions, known as events, smash incoming particles together, creating outgoing particles that continue on new trajectories and are captured by a silicon detector. Due to the scale of the events occurring (a distance order of millimeters) and the speed at which they occur, catching events in the act is nearly impossible. Thus, physicists are forced to reconstruct these events using complex mathematical calculations. With so many events, however, and such an amount of hits being captured, it can be extremely hard to determine and classify what came from where, and when. As a result, traditional computations suffer from an "explosion"[2] of the CPU time (around 6 weeks) due to attempting to fit a line to tens of thousands of points. Given this, CERN, the entity that runs the LHC, posted this dataset in the hopes that someone could use the concepts in Machine Learning to perform the particle track reconstruction faster without sacrificing too much of the accuracy given by traditional methods.

Problem Definition and Algorithm

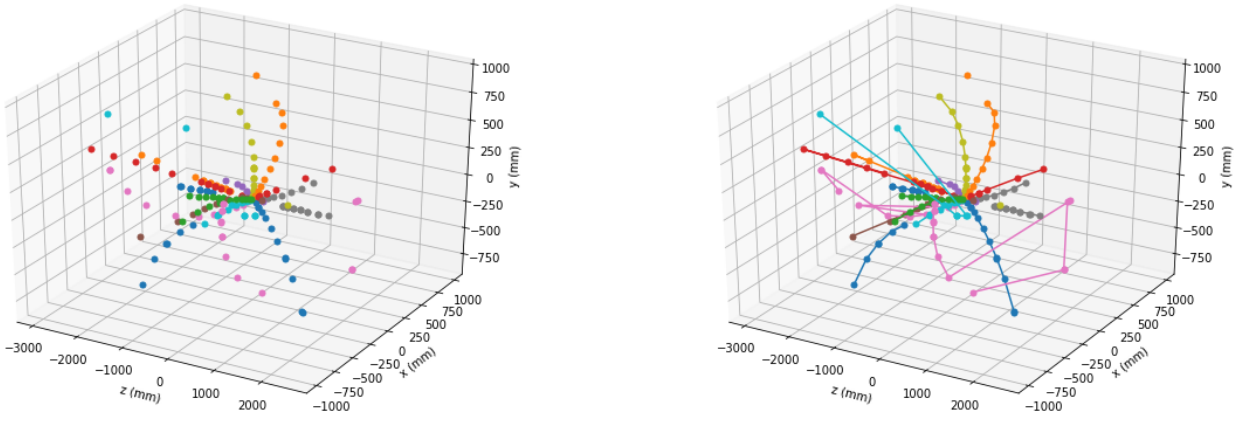


Figure 1. Example of particle tracks in dataset (graphs adapted from Joshua Bonatt)

Task Definition

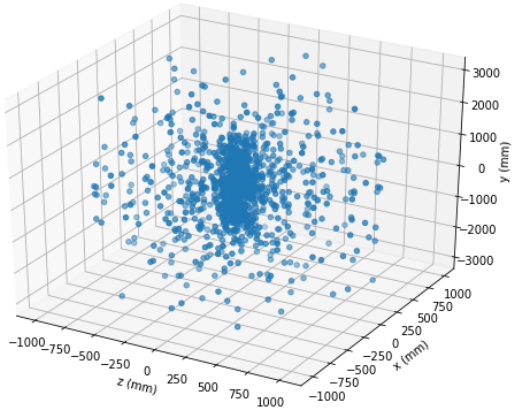
Given the 3-dimensional coordinates of outgoing particle hits registered in a silicon detector, as well as the original pre-event positions and momenta of these particles, it is desirable to accurately and quickly assign particulate to groups representing certain tracks, and comparing said tracks with a provided truth file to gauge accuracy of the program. For particle physicists, the ability to reconstruct events is key in continuing to further understanding our universe and the fundamental laws and building blocks that govern it. With Machine Learning algorithms performing faster than traditional algorithms, real results can accelerate the ability of physicists to perform and understand these experiments. A baseline clustering algorithm is used to determine whether neural networks will be necessary, and then a neural network is trained to classify points to certain tracks based on probability of pairing.

Dataset

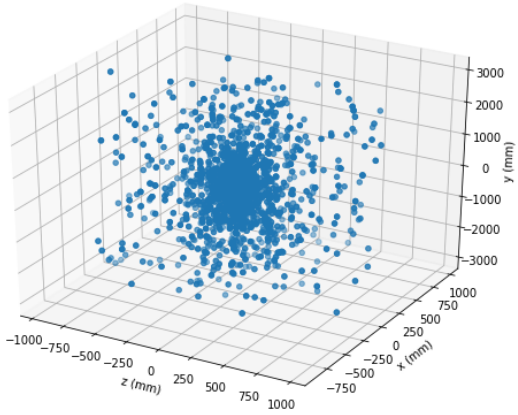
The Kaggle TrackML Challenge[2] comes with two labeled datasets: event-hits, and event-particles. event-hits contains the 3-dimensional position coordinates for each hit, regional identifiers for areas in the silicon detector, and a numerical id for each individual hit. event-particles contains a numerical id for each individual particle, pre-event 3-dimensional position coordinates, pre-event 3-dimensional momentum along global axes, charge of the particle, and the number of hits generated by that particle. Our dataset is comprised of four 70 Gigabite training sets, with each set containing 1800 separate events and each event registering around 100,000 individual hits in the silicon detector.

Algorithm Definition

First, a program was trained on four random events to classify the tracks using Density-Based Spatial Clustering of Applications with Noise, or DBSCAN[2]. This provides a benchmark measurement as to how fast a basic Machine Learning algorithm could learn and predict the classification of tracks. Then, a simple neural network of 6 fully connected layers starting at 2^{10} neurons and ending with 2^5 neurons, decreasing by a factor of 2 each successive layer, was trained on the same four events as the DBSCAN algorithm. This Neural Network utilized the Leaky ReLU activation function and its last layer used a sigmoid activation function with an output of one.



Hits



Particles

Experimental Evaluation

Methodology

When evaluating the hits to determine tracks, some base parameters need to be implemented for classification of the tracks. First, hits near the center of the detector and hits at the end of the detector are weighted higher than those in less central regions. This is due to hits in the center being more in-line with original momentum, and so more relevant to classification, and hits at the end of the detector being the end point of a line, which helps create the line itself. Hits from tracks that are straighter are also granted more weight, for the same reason as central hits. Random tracks or short tracks are given no weight. Then, once the weight of all hits is summed, it should be equal to 1. The goal of these programs (DBSCAN and our Neural Network) is then to get as close to 1 as possible. For the Neural Network, once the network is trained, we can create a prediction likelihood matrix which contains the probabilities for the likelihood that hit A is on the same track as hit B. Using a threshold of 0.65,

assembling a track becomes the process of taking the highest probability for hit A as the starting point of a track, and then adding on hits until no probability remains that is above the specified threshold. See below for a small sample.

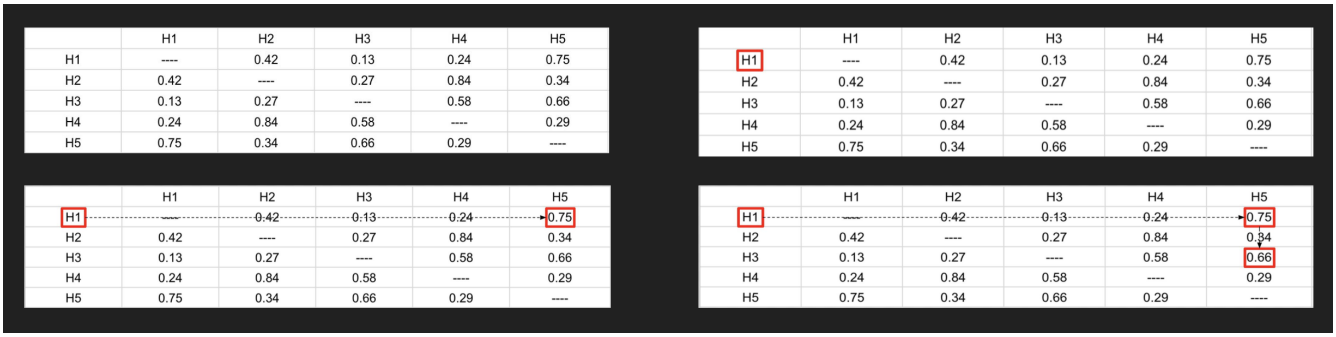


Figure 2. Starting with Hit 1, find the highest probability of sharing a track is Hit 5. From Hit 5, find the next highest probability that isn't Hit 1, being Hit 3. Thus, Track 1 is composed of Hits 1,3,and 5. Similarly, Hit 2 finds Hit 4.

Results

Running the DBSCAN model, a sum of all weights of hits of 0.20306 out of 1.0 is produced in a time to train and predict of around 30 minutes. This means that roughly 20 percent of the predicted tracks were accurate in comparison with the truth file provided. This being the best performance DBSCAN could achieve, the Neural Network is then trained, which took 40 minutes on a Nvidia GTX 1080 GPU using Tensorflow and Keras, and two and a half hours to predict based on the Neural Network. The Neural Network produces a sum of all weights of hits of 0.4535 out of 1.0, a drastic improvement on the DBSCAN results, and in a substantially smaller time to predict than the 6 weeks of the raw brute-force equations normally used.

Discussion

Examining the results based on the initial hypothesis, it is great to see that the Neural Network algorithm not only outperforms the DBSCAN algorithm by a factor of 2, but also still enjoys a drastically reduced prediction time when compared with the original mathematical modeling. Specifically, the results show that even a simple neural network can outperform straight up computational power, as the approximation of a rudimentary brain can better classify an object that a simple localized clustering algorithm. The grouping system is more complex, and thus the groups are more refined. It can be inferred then that increasing the complexity of the Neural Network will lead to an increased ability to classify these tracks while still maintaining the advantage in prediction time.

Related Work

On the Kaggle TrackML Challenge webpage[2], there are many other contestants and entries, and they utilize a variety of techniques[3,4,5,6,7,8]: some use deep learning concepts like neural networks, some stick with clustering algorithms and attempt to improve the performance of those, and some attempt to improve the classical mathematical modeling time to predict. All of those can be found here: https://www.kaggle.com/c/trackml-particle-identification/kernels. Our problems are the same, but due to the diverse range of options in the Machine Learning community, and the broad nature to which the challenge is given, the breadth of project styles and methods is incredible, and no two groups/people are performing the same experiment.

Code and Dataset

The code and dataset is freely available online at the Kaggle TrackML challenge page. The link to the GitHub is here: https://github.com/LAL/trackml-library, and is free and open source. Images of important segments of the code are in Appendix A.

Conclusion

While these preliminary results are encouraging, there is much more to do/ much more that can be done. The next step will be exploring adding convolutional layers to the neural network as well as increasing the number of events trained on. The decision to train on four events was due to the sheer volume of data and time constraints, but now that more time is available, a more complex Neural Network can be trained in the hopes that a relatively significant accuracy rate of 85-90 percent can be reached.

References

1. The Large Hadron Collider. Superconductivity — CERN Available at: https://home.cern/science/accelerators/large-hadron-collider. (Accessed: 5th December 2018)

2. TrackML Particle Tracking Challenge. Kaggle (2017). Available at: https://www.kaggle.com/c/trackml-particle-identification. (Accessed: 10th September 2018)
3. Bonatt, J. TrackML EDA, etc. Kaggle (2018). Available at: https://www.kaggle.com/jbonatt/trackml-eda-etc. (Accessed: 30th October 2018)

4. CPMP, A Faster Python Scoring Function. Kaggle (2018). Available at: http://www.kaggle.com/cpmpml/a-faster-python-scoring-function. (Accessed: 25th October 2018)

5. Elshamy, Wesam. "TrackML Problem Explanation and Data Exploration." Kaggle, Kaggle, 2018, www.kaggle.com/wesamelshamy/trackml-problem-explanation-and-data-exploration.

6. Hushchyn, Mikhail. "DBSCAN Benchmark." Kaggle, Kaggle, 2018, www.kaggle.com/mikhailhushchyn/dbscan-benchmark.

7. Silva, Luis Andre Dutra e. "Unrolling of Helices + Outliers Removal." Kaggle, Kaggle, 2018, www.kaggle.com/mindcool/unrolling-of-helices-outliers-removal.

8. Sionkowski, Grzegorz. "Bayesian Optimization." Kaggle, Kaggle, 2018, www.kaggle.com/sionek/bayesian-optimization.

Appendix A: Code

```
import os
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import pandas as pd
%matplotlib inline

from time import time

from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn import metrics
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler, MaxAbsScaler, Normalizer
from sklearn.model_selection import RandomizedSearchCV

from trackml.dataset import load_event, load_dataset
from trackml.score import score_event



from scipy import spatial
```

```
path_to_train = "E:\\train_1"
```

```
event_prefix = "event000001000"
```

```
hits, cells, particles, truth = load_event(os.path.join(path_to_train, event_prefix))
```

```
hits.head()
```

	hit_id	x	y	z	volume_id	layer_id	module_id
0	1	-64.409897	-7.163700	-1502.5	7	2	1
1	2	-55.336102	0.635342	-1502.5	7	2	1
2	3	-83.830498	-1.143010	-1502.5	7	2	1
3	4	-96.109100	-8.241030	-1502.5	7	2	1
4	5	-62.673599	-9.371200	-1502.5	7	2	1

```
#Based off of code from Mikhail Hushchyn
# Uses DBSCAN clustering algo
class Clusterer(object):

    def __init__(self, eps):
        self.eps = eps


    def _preprocess(self, hits):

        x = hits.x.values
        y = hits.y.values
        z = hits.z.values

        r = np.sqrt(x**2 + y**2 + z**2)
        hits['x2'] = x/r
        hits['y2'] = y/r

        r = np.sqrt(x**2 + y**2)
        hits['z2'] = z/r

        ss = StandardScaler()
        X = ss.fit_transform(hits[['x2', 'y2', 'z2']].values)

        return X


    def predict(self, hits):

        X = self._preprocess(hits)

        clf = DBSCAN(eps=self.eps, min_samples=1, algorithm='auto', n_jobs=-1)
        labels = clf.fit_predict(X)

        return labels
```

```
model = Clusterer(eps=0.00738)
labels = model.predict(hits)
```

```
def create_one_event_submission(event_id, hits, labels):
    sub_data = np.column_stack(([event_id]*len(hits), hits.hit_id.values, labels))
    submission = pd.DataFrame(data=sub_data, columns=["event_id", "hit_id", "track_id"]).astype(int)
    return submission
```

```
submission = create_one_event_submission(0, hits, labels)
```

```
#Alternative scoring function from CPMP
def score_event_fast(truth, submission):
    truth = truth[['hit_id', 'particle_id', 'weight']].merge(submission, how='left', on='hit_id')
    df = truth.groupby(['track_id', 'particle_id']).hit_id.count().to_frame('count_both').reset_index()
    truth = truth.merge(df, how='left', on=['track_id', 'particle_id'])

    df1 = df.groupby(['particle_id']).count_both.sum().to_frame('count_particle').reset_index()
    truth = truth.merge(df1, how='left', on='particle_id')
    df1 = df.groupby(['track_id']).count_both.sum().to_frame('count_track').reset_index()
    truth = truth.merge(df1, how='left', on='track_id')
    truth.count_both *= 2
    score = truth[(truth.count_both > truth.count_particle) & (truth.count_both > truth.count_track)].weight.sum()
    return score
```

```
score = score_event_fast(truth, submission)
print("Your score: ", score)
```

```
Your score:  0.20306711
```

```
dataset_submissions = []
dataset_scores = []
i = 0

for event_id, hits, cells, particles, truth in load_dataset(path_to_train, skip=0, nevents=100):
    i+=1
    # Track pattern recognition
    model = Clusterer(eps=0.008)
    labels = model.predict(hits)

    # Prepare submission for an event
    one_submission = create_one_event_submission(event_id, hits, labels)
    dataset_submissions.append(one_submission)

    # Score for the event
    score = score_event_fast(truth, one_submission)
    dataset_scores.append(score)

print('Mean score: %.3f' % (np.mean(dataset_scores)))
```

```
from tqdm import tqdm_notebook

prefix='E:\\'

def get_event(event):
    hits= pd.read_csv(prefix+'train_1/%s-hits.csv'%event)
    cells= pd.read_csv(prefix+'train_1/%s-cells.csv'%event)
    truth= pd.read_csv(prefix+'train_1/%s-truth.csv'%event)
    particles= pd.read_csv(prefix+'train_1/%s-particles.csv'%event)
```

```
# Adapted from outrunner-kaggle    return hits, cells, truth, particles
Train = []
for i in tqdm_notebook(range(10,20)):
    event = 'event0000010%02d'%i
    hits, cells, truth, particles = get_event(event)
    hit_cells = cells.groupby(['hit_id']).value.count().values
    hit_value = cells.groupby(['hit_id']).value.sum().values
    features = np.hstack((hits[['x','y','z']]/1000, hit_cells.reshape(len(hit_cells),1)/10,hit_value.reshape(len(hit_ce
    particle_ids = truth.particle_id.unique()
    particle_ids = particle_ids[np.where(particle_ids!=0)[0]]

    pair = []
    for particle_id in particle_ids:
        hit_ids = truth[truth.particle_id == particle_id].hit_id.values-1
        for i in hit_ids:
            for j in hit_ids:
                if i != j:
                    pair.append([i,j])
    pair = np.array(pair)
    Train1 = np.hstack((features[pair[:,0]], features[pair[:,1]], np.ones((len(pair),1))))

    if len(Train) == 0:
        Train = Train1
    else:
        Train = np.vstack((Train,Train1))

    n = len(hits)
    size = len(Train1)*3
    p_id = truth.particle_id.values
    i =np.random.randint(n, size=size)
    j =np.random.randint(n, size=size)
    pair = np.hstack((i.reshape(size,1),j.reshape(size,1)))
    pair = pair[((p_id[i]==0) | (p_id[i]!=p_id[j]))]

    Train0 = np.hstack((features[pair[:,0]], features[pair[:,1]], np.zeros((len(pair),1))))

    print(event, Train1.shape)

    Train = np.vstack((Train,Train0))


np.random.shuffle(Train)
print(Train.shape)
```

```
import pandas as pd
df = pd.DataFrame(Train)
df.head()
```

	0	1	2	3	4	5	6	7	8	9	10
0	-0.031143	-0.007685	0.000165	0.3	0.307577	0.120749	0.344592	0.201000	0.3	3.000000	0.0
1	0.307593	0.190076	-0.335400	0.4	4.000000	0.067226	-0.026196	-1.098000	0.2	0.277347	0.0
2	1.012740	-0.140985	0.254600	0.4	4.000000	0.062592	-0.035696	0.022835	0.4	0.334701	1.0
3	0.259274	0.256592	0.176600	0.2	2.000000	0.325745	-0.967520	0.715600	0.4	4.000000	0.0
4	0.216825	0.448875	0.785720	0.5	5.000000	-0.039330	-0.059234	0.033791	0.4	0.295426	0.0

```
from keras import Sequential
from keras.layers import Dense
from keras import optimizers
from keras.utils import to_categorical
from keras.optimizers import Adam
from keras.layers import Dropout
from keras.layers.advanced_activations import LeakyReLU
```

```
fs = 10

model = Sequential()
model.add(Dense(1024, activation='linear', input_shape=(fs,)))
model.add(LeakyReLU(alpha=.001))
model.add(Dense(512, activation='linear'))
model.add(LeakyReLU(alpha=.001))
model.add(Dense(256, activation='linear'))
model.add(LeakyReLU(alpha=.001))
model.add(Dense(128, activation='linear'))
model.add(LeakyReLU(alpha=.001))
model.add(Dense(64, activation='linear'))
model.add(LeakyReLU(alpha=.001))
model.add(Dense(32, activation='linear'))
model.add(LeakyReLU(alpha=.001))

model.add(Dense(1, activation='sigmoid'))

epochs=30
lrate=0.001
model.compile(loss=['binary_crossentropy'], optimizer=Adam(lr=lrate), metrics=['accuracy'])
print(model.summary())
```

```
# Adapted from outrunner-kaggle
TestX = np.zeros((len(features), 10))
TestX[:,5:] = features

TestX1 = np.zeros((len(features), 10))
TestX1[:,:5] = features

preds = []

for i in tqdm_notebook(range(len(features)-1)):
    TestX[i+1:,:5] = np.tile(features[i], (len(TestX)-i-1, 1))

    pred = model.predict(TestX[i+1:], batch_size=20000)[:,0]
    idx = np.where(pred>0.2)[0]

    if len(idx) > 0:
        TestX1[idx+i+1,5:] = TestX[idx+i+1,:5]
        pred1 = model.predict(TestX1[idx+i+1], batch_size=20000)[:,0]
        pred[idx] = (pred[idx]+pred1)/2

    idx = np.where(pred>0.5)[0]

    preds.append([idx+i+1, pred[idx]])


preds.append([np.array([], dtype='int64'), np.array([], dtype='float32')])

# rebuild to NxN
for i in range(len(preds)):
    ii = len(preds)-i-1
    for j in range(len(preds[ii][0])):
        jj = preds[ii][0][j]
        preds[jj][0] = np.insert(preds[jj][0], 0 ,ii)
        preds[jj][1] = np.insert(preds[jj][1], 0 ,preds[ii][1][j])

    np.save('my_%s.npy'%event, preds)
```

```
# Adapted from outrunner-kaggle
def get_path2(hit, mask, thr):
    path = [hit]
    a = 0
    while True:
        c = get_predict2(path[-1])
        mask = (c > thr)*mask
        mask[path[-1]] = 0

        if 1:
            cand = np.where(c>thr)[0]
            if len(cand)>0:
                mask[cand[np.isin(module_id[cand], module_id[path])]]=0

        a = (c + a)*mask
        if a.max() < thr*len(path):
            break
        path.append(a.argmax())
    return path

def get_predict2(p):
    c = np.zeros(len(preds))
    c[preds[p, 0]] = preds[p, 1]
    return c
```

```
# Adapted from outrunner-kaggle
tracks_all = []
thr = 0.85
x4 = True
for hit in tqdm_notebook(range(len(preds))):
    m = np.ones(len(truth))
    path  = get_path2(hit, m, thr)
    if x4 and len(path) > 1:
        m[path[1]]=0
        path2  = get_path2(hit, m, thr)
        if len(path) < len(path2):
            path = path2
            m[path[1]]=0
            path2  = get_path2(hit, m, thr)
            if len(path) < len(path2):
                path = path2
        elif len(path2) > 1:
            m[path[1]]=1
            m[path2[1]]=0
            path2  = get_path2(hit, m, thr)
            if len(path) < len(path2):
                path = path2
    tracks_all.append(path)
np.save('my_tracks_all', tracks_all)
```

```
#Alternative scoring function from CPMP
def score_event_fast(truth, submission):
    truth = truth[['hit_id', 'particle_id', 'weight']].merge(submission, how='left', on='hit_id')
    df = truth.groupby(['track_id', 'particle_id']).hit_id.count().to_frame('count_both').reset_index()
    truth = truth.merge(df, how='left', on=['track_id', 'particle_id'])

    df1 = df.groupby(['particle_id']).count_both.sum().to_frame('count_particle').reset_index()
    truth = truth.merge(df1, how='left', on='particle_id')
    df1 = df.groupby(['track_id']).count_both.sum().to_frame('count_track').reset_index()
    truth = truth.merge(df1, how='left', on='track_id')
    truth.count_both *= 2
    score = truth[(truth.count_both > truth.count_particle) & (truth.count_both > truth.count_track)].weight.sum()
    return score
```

```
scores = get_track_score(tracks_all, 8)
```

```
idx = np.argsort(scores)[::-1]
tracks = np.zeros(len(hits))
track_id = 0

for hit in idx:

    path = np.array(tracks_all[hit])
    path = path[np.where(tracks[path]==0)[0]]

    if len(path)>3:
        track_id = track_id + 1
        tracks[path] = track_id
```

```
submission = pd.DataFrame({'hit_id': truth.hit_id, 'track_id': tracks})
```

```
score = score_event_fast(truth, submission)[

print(score)
```