Evaluation Test: E2E

Specific Task 2. Jets as graphs

Submitted by Sarthak Rastogi (thesarthakrastogi@gmail.com)

To ML4SCI, CERN

Dataset

Pythia8 Quark and Gluon Jets for Energy Flow

(without charm and bottom jets)

https://zenodo.org/record/3164691#.YHXOO-gvNID

In [10]:
```
!wget -qq "https://zenodo.org/record/3164691/files/QG_jets.npz"
```

In [51]:
```
import numpy as np
df = np.load('QG_jets.npz')
df.files
```

Out[51]: ['X', 'y']

In [12]:
```
#Alternatively, using energyflow:
#!pip install energyflow
#import energyflow
#df = energyflow.qg_jets.load(num_data=100000, pad=True, ncol=4, generator='pythia',
```

In [52]:
```
X = df['X']
X.shape
```

Out[52]: (100000, 139, 4)

In [53]:
```
y = df['y']
y.shape
```

Out[53]: (100000,)

Loading the dataset from a numpy array into a Pandas dataframe:

In [54]:
```
X1 = []
for i in X:
  X1.append(i[0])
```

In [55]:
```
import pandas as pd
X = pd.DataFrame(X1)
y = pd.DataFrame(y, columns=['y'])
X['y'] = y
X = X[:50000]
X.head()
```

Out[55]:

	0	1	2	3	y
0	0.268769	0.356903	4.741387	22.0	1.0
1	1.212663	-0.112853	3.047088	-211.0	1.0
2	0.216829	-0.997057	0.532569	22.0	1.0

	0	1	2	3	y
3	0.413806	0.956889	5.742566	211.0	1.0
4	0.476434	-0.403307	4.126747	22.0	1.0

In [56]:

```
y = X["y"]
X = X.drop(["y"], axis=1)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
```

Provide a description on what considerations you have taken to project this point-cloud dataset to a set of interconnected nodes and edges.

For every data point, we need to collect the set of neighbour data points form the graph, and that depends on the noiose level we have. In this dataset I found a moderate to low amount of noise so it was safe to collect the direct neighbours of the data points.

Creating the graph

In [18]:

```
!pip install dgl==0.4.3
```

```
Collecting dgl==0.4.3
  Downloading https://files.pythonhosted.org/packages/3f/9e/7757847d45eb20cf96fe649d
c3bfed97550ee25909a679a3ea404da5e92d/dgl-0.4.3-cp37-cp37m-manylinux1_x86_64.whl (3.0
MB)
     |████████████████████████████████| 3.0MB 20.4MB/s
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-package
s (from dgl==0.4.3) (1.4.1)
Requirement already satisfied: networkx>=2.1 in /usr/local/lib/python3.7/dist-packag
es (from dgl==0.4.3) (2.5.1)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.7/dist-packag
es (from dgl==0.4.3) (1.19.5)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.7/dist-pac
kages (from dgl==0.4.3) (2.23.0)
Requirement already satisfied: decorator<5,>=4.3 in /usr/local/lib/python3.7/dist-pa
ckages (from networkx>=2.1->dgl==0.4.3) (4.4.2)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pa
ckages (from requests>=2.19.0->dgl==0.4.3) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-p
ackages (from requests>=2.19.0->dgl==0.4.3) (2020.12.5)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-package
s (from requests>=2.19.0->dgl==0.4.3) (2.10)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/loca
l/lib/python3.7/dist-packages (from requests>=2.19.0->dgl==0.4.3) (1.24.3)
Installing collected packages: dgl
Successfully installed dgl-0.4.3
```

In [57]:

```
import dgl
def create_graph(e):
  graph = dgl.DGLGraph()
  graph.add_nodes(len(X))

  src, dst = tuple(zip(*e))

  #defining a bidirectional graph
  graph.add_edges(src, dst)
  graph.add_edges(dst, src)

  return graph
```

In [58]:
```
edges = []
from itertools import combinations
import random

edges = random.sample(range(1, len(X_train)), int(len(X_train)/30))

edges = combinations(edges, 2)
```

In [59]:
```
edges = list(edges)
len(edges)
```

Out[59]:
```
679195
```

In [60]:
```
G1 = create_graph(edges)
print(G1)
print(G1.number_of_nodes(), G1.number_of_edges())
```

```
DGLGraph(num_nodes=50000, num_edges=1358390,
         ndata_schemes={}
         edata_schemes={})
50000 1358390
```

Defining model 1

In [61]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
```

In [62]:
```
def message(e):
  return {"message": e.src["h"]}

def reduceg(n):
  return {"h": torch.sum(n.mailbox["message"], dim=1)}
```

In [26]:
```
class GCNLayer(nn.Module):
  def __init__(self, in_features, out_features):
    super(GCNLayer, self).__init__()
    self.Linear = nn.Linear(in_features, out_features)

  def forward(self, graph, inputs):
    graph.ndata["h"] = inputs

    graph.send(graph.edges(), message)
    graph.recv(graph.nodes(), reduceg)

    h = graph.ndata.pop("h")

    return self.Linear(h)
```

In [27]:
```
class GCN(nn.Module):
  def __init__(self, in_features, hidden_size, num_classes):
    super(GCN, self).__init__()

    self.gcn1 = GCNLayer(in_features, hidden_size)
    self.gcn2 = GCNLayer(hidden_size, num_classes)
    self.softmax = nn.Softmax()

  def forward(self, graph, inputs):
    h = self.gcn1(graph, inputs)
    h = torch.relu(h)
    h = self.gcn2(graph, h)
```

```
        h = self.softmax(h)
        return h
```

In [28]:
```
net = GCN(4, 16, 2)
```

In [29]:
```
inputs = torch.tensor(X.values)
labeled_nodes = torch.tensor(y_train[::3].index)
labels = torch.tensor(y_train[::3].values)
```

Compiling and training model 1

In [30]:
```
optimizer = torch.optim.Adam(net.parameters(), lr=0.05)
preds = []
losses = []

for epoch in range(3):
  pred = net(G1, inputs.float())
  preds.append(pred)

  loss = F.cross_entropy(pred[labeled_nodes], labels.long())
  losses.append(loss)

  optimizer.zero_grad()
  loss.backward()
  optimizer.step()

  print("Epoch: ", epoch+1, "Loss: ", loss.item())
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:13: UserWarning: Implic
it dimension choice for softmax has been deprecated. Change the call to include dim=
X as an argument.
  del sys.path[0]
Epoch:  1 Loss:  0.6956616044044495
Epoch:  2 Loss:  0.6954920887947083
Epoch:  3 Loss:  0.6954820156097412
```

Defining model 2

In [33]:
```
class GCN(nn.Module):
  def __init__(self, in_features, hidden_size, num_classes):
    super(GCN, self).__init__()

    self.gcn1 = GCNLayer(in_features, hidden_size)
    self.gcn2 = GCNLayer(hidden_size, hidden_size)
    self.gcn3 = GCNLayer(hidden_size, num_classes)
    self.softmax = nn.Softmax()

  def forward(self, graph, inputs):
    h = self.gcn1(graph, inputs)
    h = torch.relu(h)
    h = self.gcn2(graph, h)
    h = torch.relu(h)
    h = self.gcn3(graph, h)
    h = self.softmax(h)
    return h
```

In [47]:
```
net = GCN(4, 32, 2)
```

In [50]:
```
optimizer = torch.optim.Adam(net.parameters(), lr=0.1)
preds = []
```

```
losses = []

for epoch in range(3):
  pred = net(G1, inputs.float())
  preds.append(pred)

  loss = F.cross_entropy(pred[labeled_nodes], labels.long())
  losses.append(loss)

  optimizer.zero_grad()
  loss.backward()
  optimizer.step()

  print("Epoch: ", epoch+1, "Loss: ", loss.item())
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:16: UserWarning: Implic
it dimension choice for softmax has been deprecated. Change the call to include dim=
X as an argument.
  app.launch_new_instance()
Epoch:  1 Loss:  0.696419894695282
Epoch:  2 Loss:  0.6971690058708191
Epoch:  3 Loss:  0.6964251399040222
```

Discuss the resulting performance of the 2 chosen architectures.

The first architecture is a shallower GCN with a lower hidden_size. In the second architecture I experimented with increasing the hidden_size hyperparameter and found that it leads to a higher loss, which can be attributed to overfitting. The presence of only four features also indicates that our feature map may not be strong enough to make higher quality predictions/

```
In [ ]:
```