Machine-Learning methods for Dark Sector searchers

Felipe Ferreira de Freitas

felipefreitas@ua.pt

my github

YouTube lectures

Felipe Ferreira de Freitas

- Graduate in Medical Physics(Catholic University of Pernambuco).
- Msc. in Particle Physics.
- Phd. in Particle Physics.
- Visiting research fellow at University of Sussex.
- Postdoctoral Researcher at Institute of Theoretical Physics-CAS.
- Postdoctoral Researcher at Aveiro University.
- Junior Researcher at Aveiro University.

Research topics:

Collider physics.

Higgs physics.

Effective field theory.

Machine learning and deep learning.

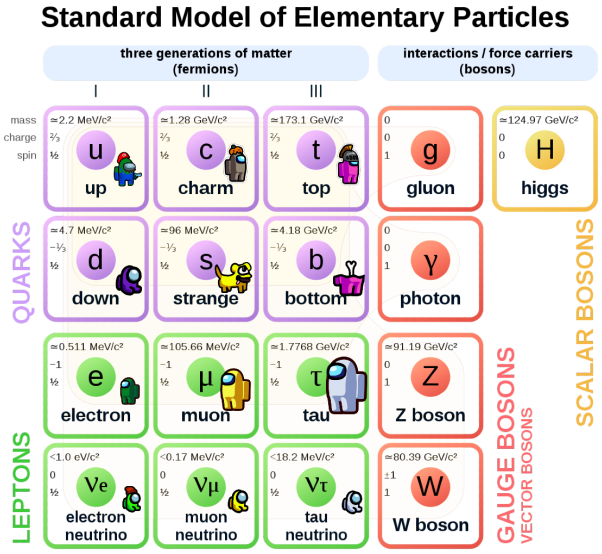
Quantum Machine Learning.

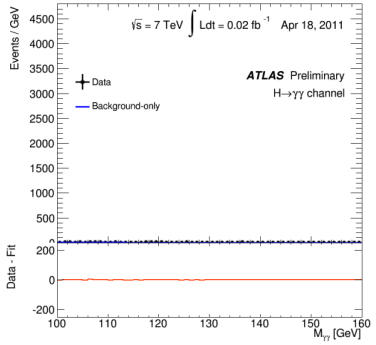
Neutrino physics.

Gravitational waves detection.

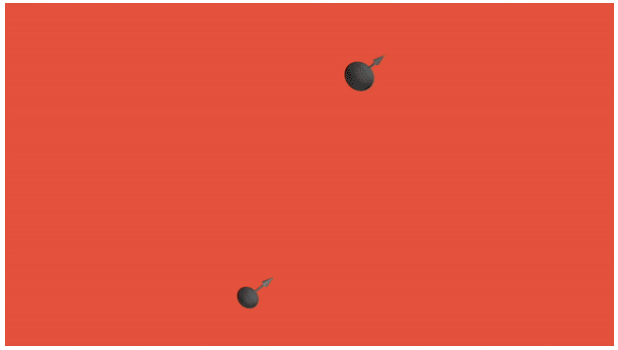
Development of DL methods.

Aplications of deep learning methods in medical physics.





Higgs working groug



Deborah Ferguson, Karan Jani, Deirdre Shoemaker, Pablo Laguna, MAYA Collaboration

First things first:

Although the name deep learning is really cool, the name itself does not carry any profound significance. Is just a stacked multilayer perceptrons with some shenanigans.

The Machine "Learning" term does not mean that the machines are learning, these things are dumb as you can get:

expectations

reality

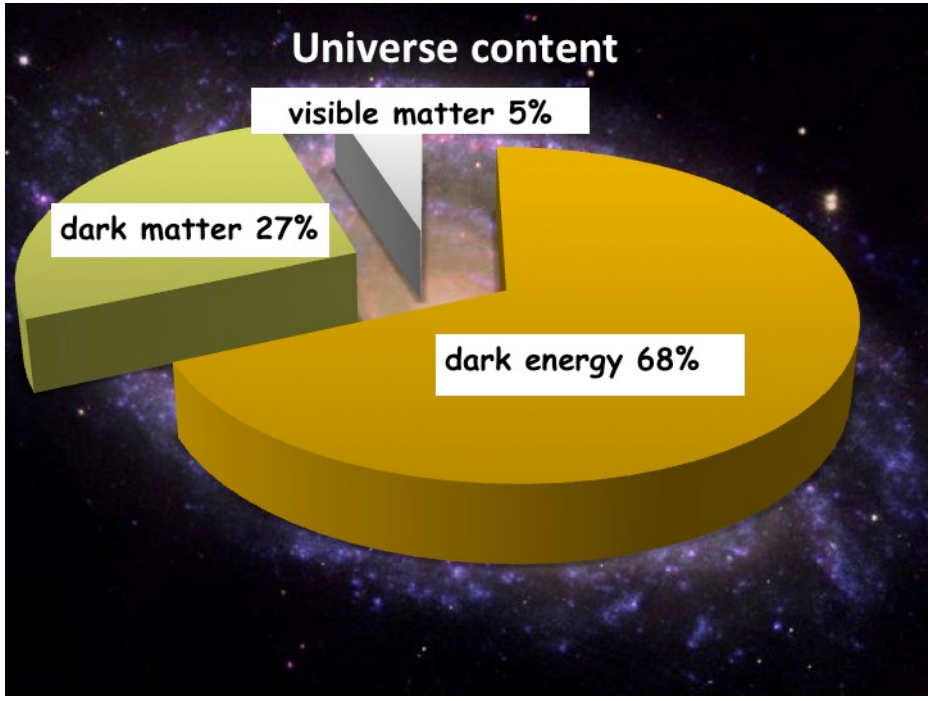
and we can easily fool them, or be fooled by them:

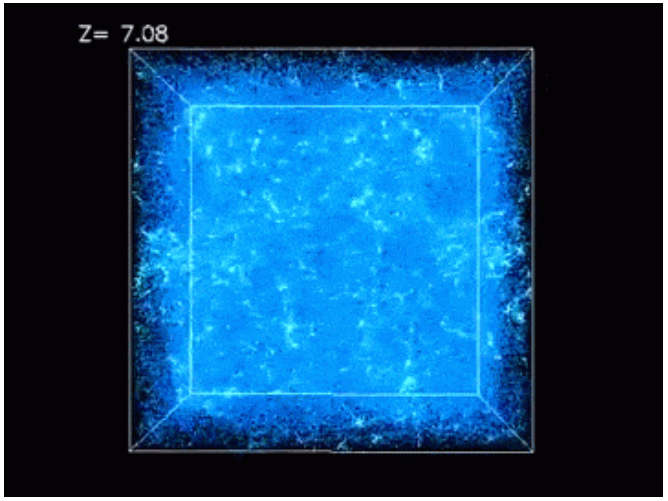
Disclaimer!!

It is a very short lecture and I can't cover everything

I can only give you the keys to the castle, but you have to open the doors

The good:



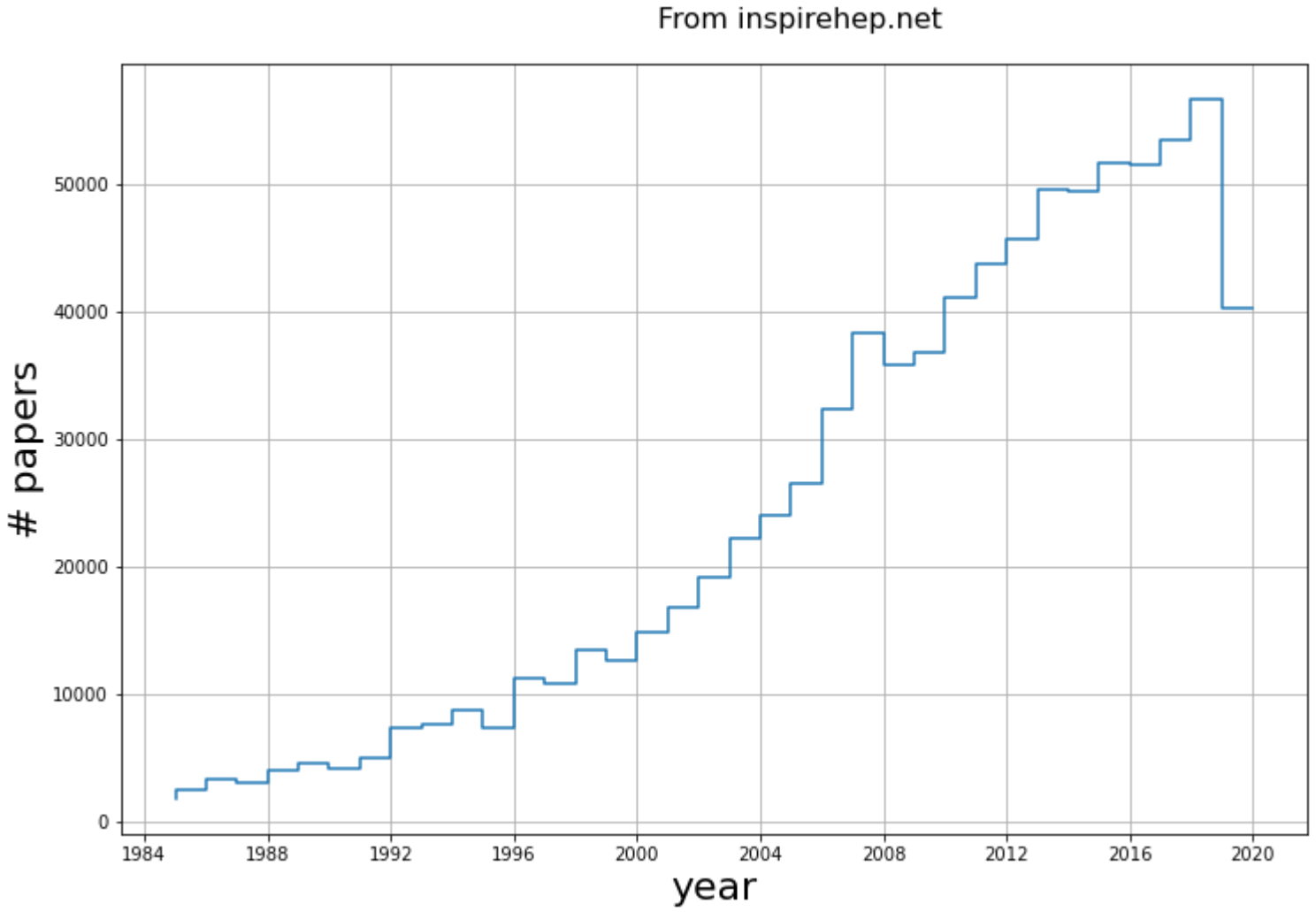


url

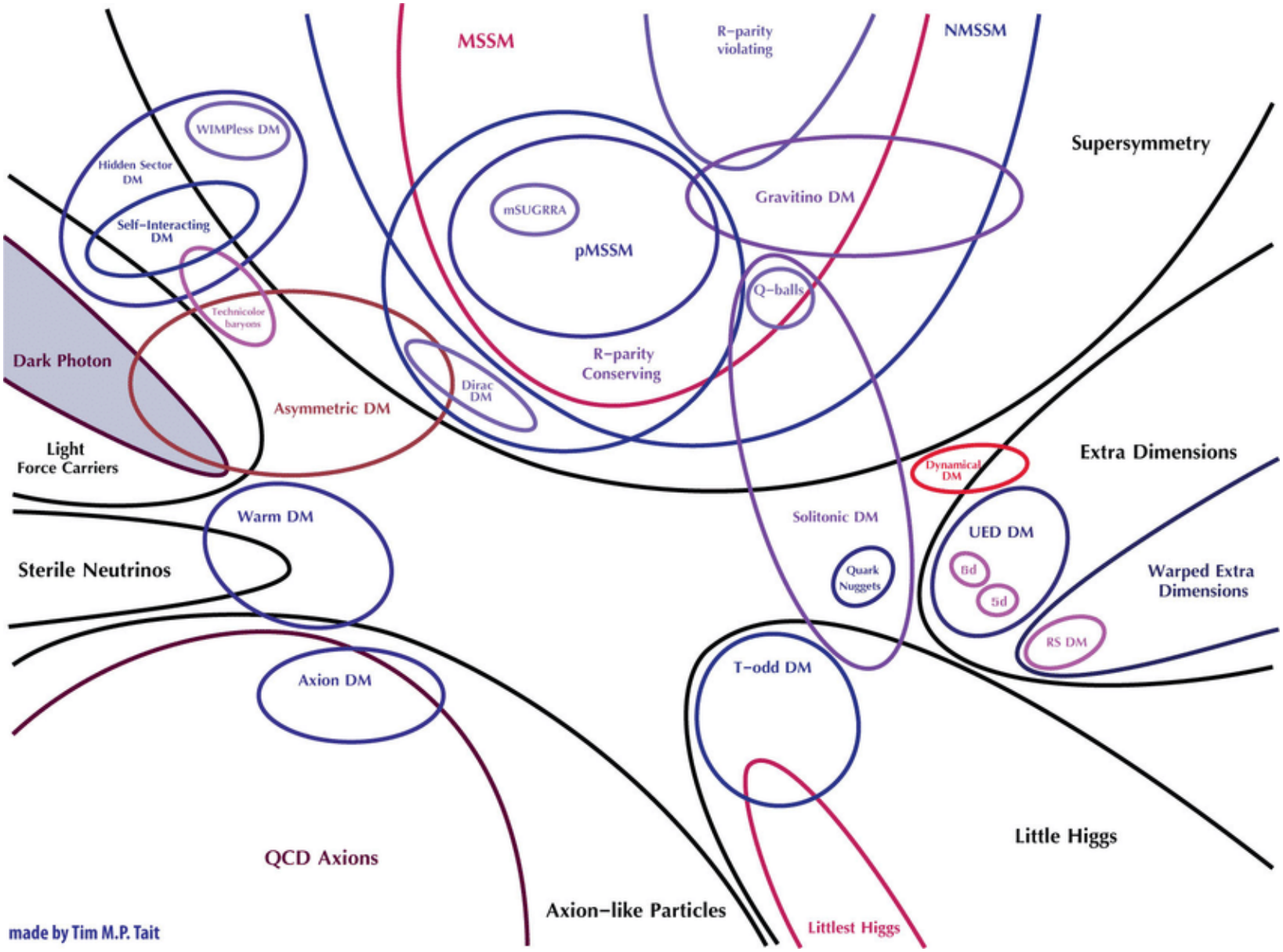
[Quantum Diaries](https://www.quantumdiaries.org/2013/06/26/does-dark-matter-really-exist/)

In [5]:

```
papers = pd.read_csv('./data/papaers on DM from 1985.csv')
papers['years'] = np.arange('1985-01', '2021-01', dtype='datetime64[Y]')
plt.figure(figsize=(12,8))
plt.step(papers['years'], (papers['height']/papers['height'].max()) * 56694, where='pre', la
plt.ylabel('# papers', fontsize=22)
plt.xlabel('year', fontsize=22)
plt.grid(':')
text = plt.annotate("From inspirehep.net", xy=(0.5,1), xytext=(0.5,0.96),
                    xycoords='figure fraction',
                    verticalalignment='top',
                    url='https://inspirehep.net/literature?sort=mostrecent&size=25&page=1&c
```


From inspirehep.net
papers
year
0
10000
20000
30000
40000
50000
1984
1988
1992
1996
2000
2004
2008
2012
2016
2020

The bad



[Planning the Future of U.S. Particle Physics (Snowmass 2013): Chapter 4: Cosmic Frontier](https://arxiv.org

/abs/1401.6085)

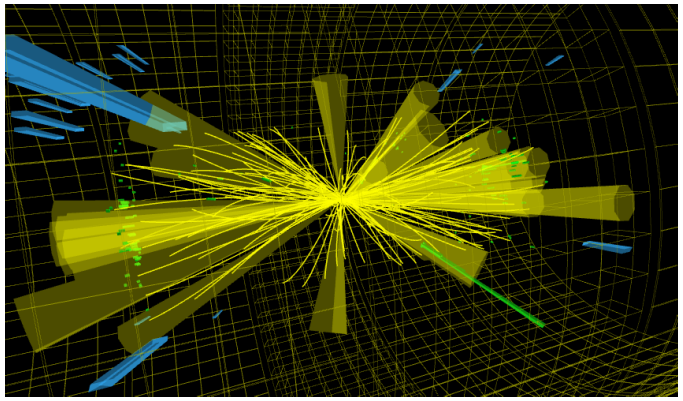
The ugly

Congratulations!! you are a Phd. (or a post-doc) searching for DM candidates, how can we find them using deep learning?

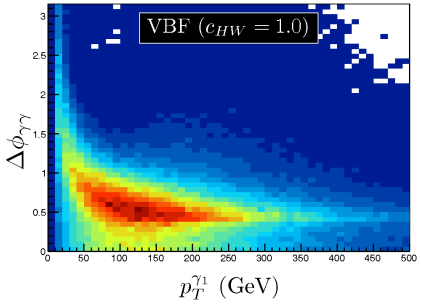
The data from particle physics experiment can comes in many flavours:

	ptb1	ptb2	misset	pth	ptz	etah	phih	mtvh	ptvh	dphib1met	signal
0	125.407039	86.117322	205.932526	209.328455	205.932526	1.277426	1.929536	449.712567	4.603307	3.037227	1
1	131.907616	70.616237	187.837401	190.132443	187.837401	-0.563147	-0.758883	415.367124	3.029103	2.895890	1
2	142.205381	29.009372	121.233232	120.412595	121.233232	-2.029023	-1.629733	294.761943	1.941833	3.006520	1
3	106.761004	27.168427	119.788307	120.780482	119.788307	-0.135757	-2.948970	293.614796	0.997513	2.935436	1
4	94.397035	73.704720	132.123257	134.300456	132.123257	2.286593	-1.262787	315.572440	3.875016	2.604910	1

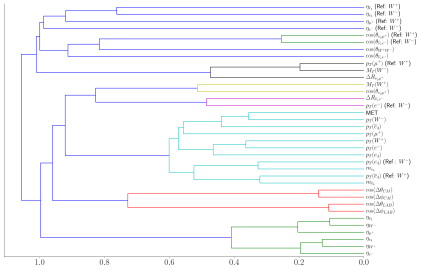
Tabular data



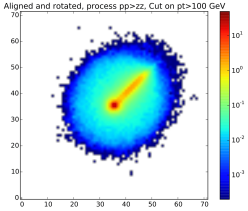
Event images



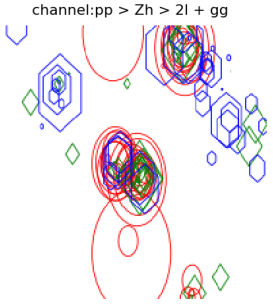
1702.05106 [hep-ph]



2010.01307 [hep-ph]



1811.01961 [hep-ph]

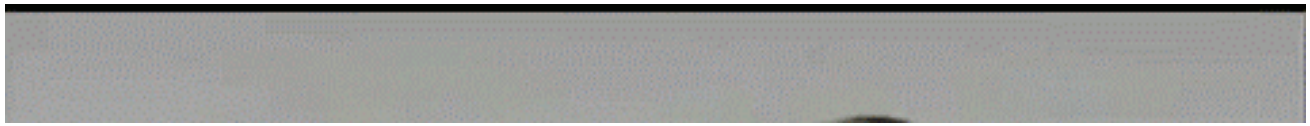


1912.12532 [hep-ph]

Start from the data

The PHENOMLDATA

- HEP new datasets.
- Always a good way to start a new project.



In [6]:

```
%%html
<iframe src="https://www.phenomldata.org/" width="1200" height="1000"></iframe>
```

PHENOMLDATA

Datasets / repositories

Dark Machines Unsupervised LHC Hackathon dataset

Arxiv:XYZ

Dark Machines Secret LHC dataset

The darkmachines unsupervised challenge project:

This dataset consist of:

- LHC events corresponding to a center-of-mass energy of 13 TeV.
- The final state objects, as described in Table 1, are stored in a one-line-per-event text (csv) file, where each line has variable length and contains 3 event-specifiers, followed by the kinematic features for each object in the event. The format of CSV files is:

| event ID| process ID| event weight| MET| METphi| obj1| E1| pt1| eta1| phi1| obj2| E2| pt2| eta2| phi2| · · ·| | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

- An event consists of a variable number of objects. An event is stored when at least one of the following requirements are fulfilled:

- At least one (b)-jet with transverse momentum $p_T > 60$ GeV and pseudorapidity $|\eta| < 2.8$,

or

- at least one electron with p_T > 25 GeV and $|\eta| < 2.47$, except for $1.37 < |\eta| < 1.52$, or
- at least one muon with p_T > 25 GeV and $|\eta| < 2.7$, or
- at least one photon with p_T > 25 GeV and $|\eta| < 2.37$.

We have the following signals:

- Channel 1
    - Z’ (2 TeV) + monojet (monojet_Zp2000.0_DM_50.0_chan1.csv), Z' decays fully invisible
    - Z’ (200 GeV) + single (anti-)top (monotop_200_A_chan1.csv), Z' decays fully invisible
    - SUSY R-parity violating stop stop (1 TeV) production (stlp_st1000_chan1.csv), stop decay into leptons and b-quarks
    - SUSY R-parity violating squark-squark production (sqsq1_sq1400_neut800_chan1.csv 1.4 TeV squark and 800 GeV neutralino), squarks decaying to jets
    - SUSY gluino-gluino production (glgl1400_neutralino1100_chan1.csv 1.4 TeV gluino and 1.1 TeV neutralino, glgl1600_neutralino800_chan1.csv 1.6 TeV gluino and 800 GeV neutralino), gluinos decay to jets and MET
    - SUSY stop-stop (1 TeV) production (stop2b1000_neutralino300_chan1.csv), stops decaying to a top quark and a neutralino (300 GeV)
    - SUSY squark-squark production (sqsq_sq1800_neut800_chan1.csv 1.8 TeV squark and 800 GeV neutralino), squarks decaying to jets and MET

Let’s get our hands dirt!

In [7]:

```
!head -n 2 /media/felipe/home\ 2/COST\ lectures\ and\ codes/Unsupervised\ dataset/training_1
```

```
354;monojet_Zp2000.0_DM_50.0;2.44336e-05;832841;-1.22431;b,947383,824498,-0.523103,1.64421;j,258722,2436
75,0.328962,2.25014;j,520092,108590,-2.2473,-1.85679;j,383024,88405.6,2.14516,-1.95635;j,39507.6,35365.
1,0.47046,-1.16445;j,225430,26878.2,-2.81608,-2.25938;
354;monojet_Zp2000.0_DM_50.0;2.44336e-05;342586;-0.432385;j,397771,391036,-0.0431479,-2.67217;j,274121,2
11069,-0.753259,0.821227;j,326774,103781,-1.81083,2.8251;j,166237,75907.3,-1.41593,1.51757;j,106044,7281
8.4,0.91515,1.60007;
```

In [4]:

```
head = ['MET','METphi',
        'E_j1','pt_j1', 'eta_j1', 'phi_j1',
        'E_j2','pt_j2', 'eta_j2', 'phi_j2',
        'E_j3','pt_j3', 'eta_j3', 'phi_j3',
        'E_b1','pt_b1', 'eta_b1', 'phi_b1',
        'E_b2','pt_b2', 'eta_b2', 'phi_b2',
        'E_pos1','pt_pos1', 'eta_pos1', 'phi_pos1',
        'E_pos2','pt_pos2', 'eta_pos2', 'phi_pos2',
        'E_e1','pt_e1', 'eta_e1', 'phi_e1',
        'E_e2','pt_e2', 'eta_e2', 'phi_e2',
        'E_mup1','pt_mup1', 'eta_mup1', 'phi_mup1',
        'E_mup2','pt_mup2', 'eta_mup2', 'phi_mup2',
        'E_mu1','pt_mu1', 'eta_mu1', 'phi_mu1',
        'E_mu2','pt_mu2', 'eta_mu2', 'phi_mu2',
        'E_a1','pt_a1', 'eta_a1', 'phi_a1'
       ]
signal_channel1 = pd.read_csv('/media/felipe/home 2/COST lectures and codes/Unsupervised dat
```

In [5]:
```
signal_channel1
```

Out[5]:

	MET	METphi	E_j1	pt_j1	eta_j1	phi_j1	E_j2	pt_j2	eta_j2	phi_j2	...	eta_mu1	phi_mu
0	832841.0	-1.224310	258722.0	243675.0	0.328962	2.250140	520092.0	108590.0	-2.247300	-1.856790	...	0	
1	342586.0	-0.432385	397771.0	391036.0	-0.043148	-2.672170	274121.0	211069.0	-0.753259	0.821227	...	0	
2	515780.0	-3.127580	491042.0	364491.0	0.793556	0.636385	421281.0	188296.0	1.442850	-1.025080	...	0	
3	421450.0	-0.648650	313887.0	233644.0	0.793024	2.874530	552236.0	202542.0	-1.660220	1.660090	...	0	
4	1012070.0	0.110458	2176690.0	1013880.0	-1.393980	2.874880	330239.0	321794.0	-0.120541	-1.523400	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
1125	400352.0	0.770201	200687.0	188717.0	-0.240538	1.362980	180146.0	153742.0	-0.568710	2.017770	...	0	
1126	756981.0	-2.584060	661465.0	391780.0	-1.110920	0.841679	534243.0	171910.0	-1.799260	0.672057	...	0	
1127	344639.0	1.347270	495518.0	439982.0	0.492304	-1.926320	135264.0	96291.0	-0.863806	-1.307740	...	0	
1128	914355.0	1.529030	1119940.0	1012690.0	0.453265	-1.772780	402070.0	77864.5	2.324450	0.929480	...	0	
1129	645619.0	-2.483990	787964.0	263158.0	1.759810	0.456786	408087.0	258903.0	-1.026900	1.530730	...	0	

1130 rows × 58 columns

In [6]:
```
signal_channel1.isnull().values.any()
```

Out[6]:
```
False
```

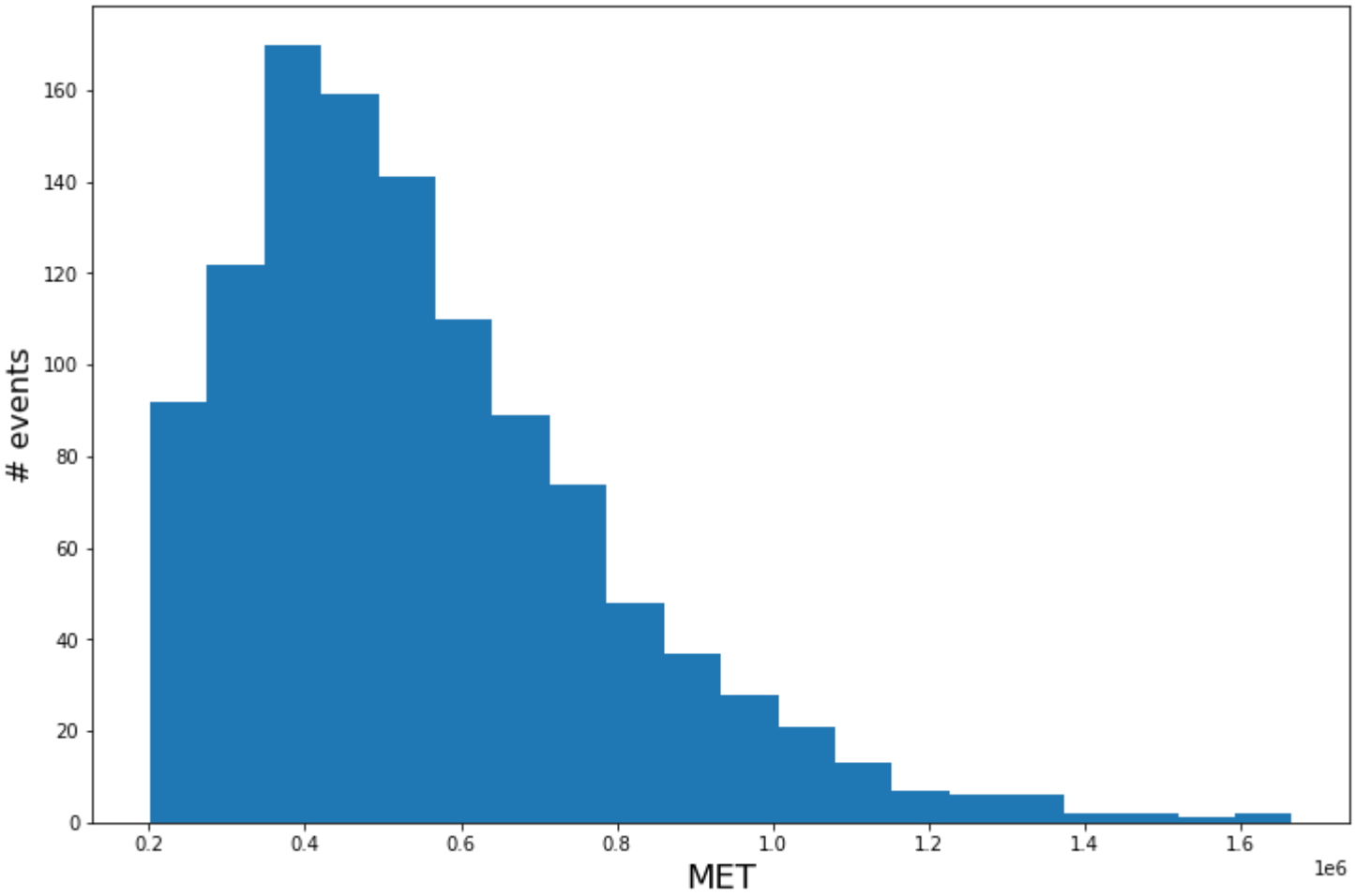
We can also make some plots!!!

In [19]:

```
fig, ax = plt.subplots(figsize=(12,8))
signal_channel1['MET'].plot.hist(bins=20, ax=ax)
plt.xlabel('MET', fontsize=18)
plt.ylabel('# events', fontsize=16)
```

Out[19]:

```
Text(0, 0.5, '# events')
```

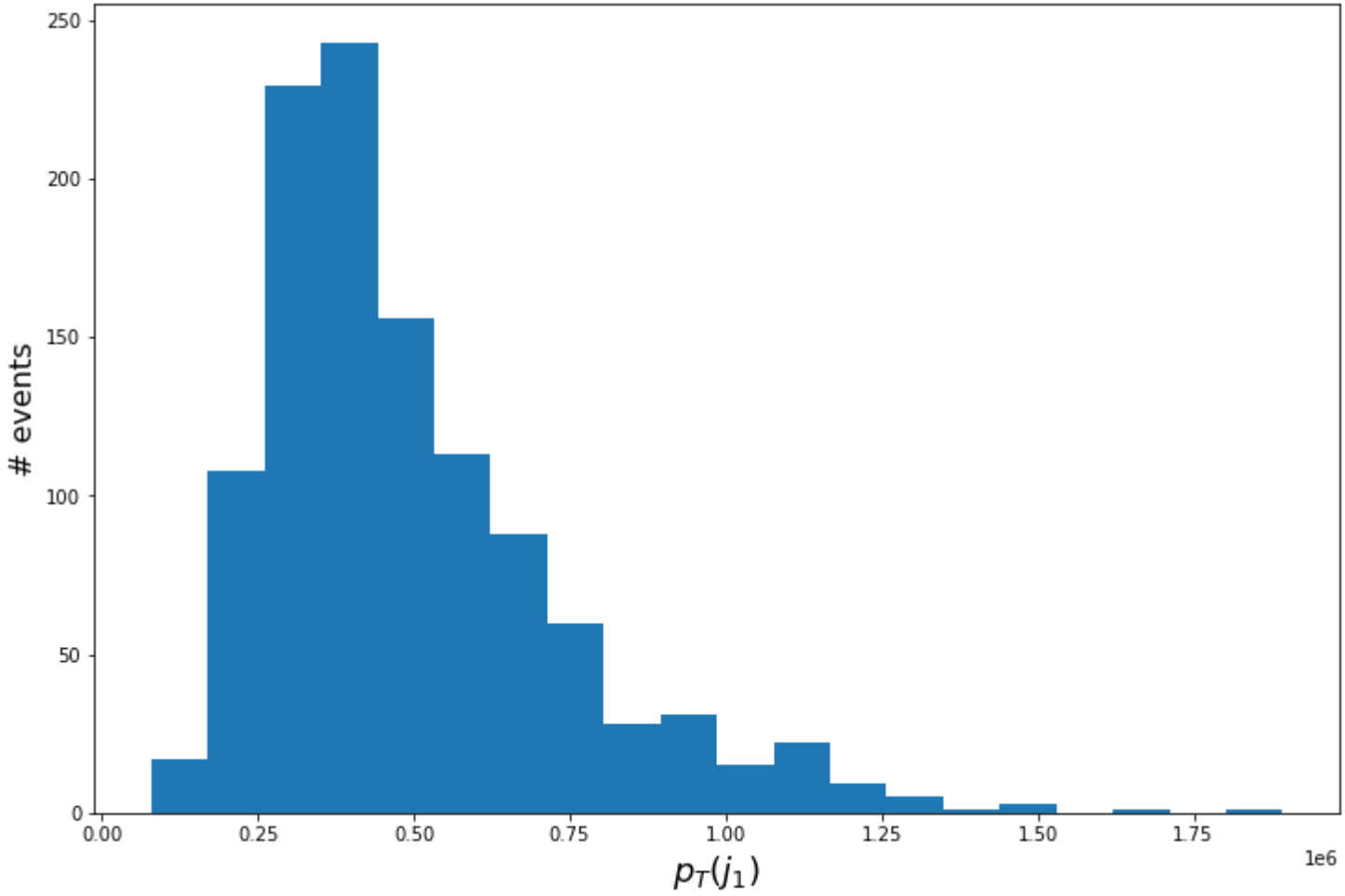


In [21]:

```
fig, ax = plt.subplots(figsize=(12,8))
signal_channel1['pt_j1'].plot.hist(bins=20, ax=ax)
plt.xlabel('$p_T(j_1)$', fontsize=18)
plt.ylabel('# events', fontsize=16)
```

Out[21]:

```
Text(0, 0.5, '# events')
```



Loading everything

In [7]:

```
signal_Zp_2TeV = pd.read_csv('/media/felipe/home 2/COST lectures and codes/Unsupervised data
signal_Zp_200GeV = pd.read_csv('/media/felipe/home 2/COST lectures and codes/Unsupervised da
signal_RP_stop = pd.read_csv('/media/felipe/home 2/COST lectures and codes/Unsupervised data
signal_RP_squark = pd.read_csv('/media/felipe/home 2/COST lectures and codes/Unsupervised da
signal_gluglu_1d4TeV = pd.read_csv('/media/felipe/home 2/COST lectures and codes/Unsupervise
signal_gluglu_1d6TeV = pd.read_csv('/media/felipe/home 2/COST lectures and codes/Unsupervise
signal_stostop = pd.read_csv('/media/felipe/home 2/COST lectures and codes/Unsupervised data
signal_squarksquark = pd.read_csv('/media/felipe/home 2/COST lectures and codes/Unsupervised
```

In [8]:

```
signals = [signal_Zp_2TeV, signal_Zp_200GeV, signal_RP_stop, signal_RP_squark, signal_gluglu
```

In [9]:

```
bg = pd.read_csv('/media/felipe/home 2/COST lectures and codes/Unsupervised dataset/training
```

create some labels to help us latter:

In [10]:
```
for i, df in enumerate(signals):
    df['label'] = i + 1
```

In [11]:
```
bg['label'] = 0
```

In [12]:

```
signals_df = pd.concat(signals)
```

In [13]:

```
full_data = pd.concat([bg,signals_df])
```

In [15]:
```
full_data
```

Out[15]:

	MET	METphi	E_j1	pt_j1	eta_j1	phi_j1	E_j2	pt_j2	eta_j2	phi_j2	...	phi_mu1	E_mu2
0	390017.0	0.538031	407492.0	393095.0	-0.266637	-2.685050	633731.0	283693.0	1.441900	-0.451286	...	0.0	0.0
1	217448.0	0.663545	707690.0	390901.0	1.195770	-2.486650	165970.0	162374.0	-0.073124	0.558267	...	0.0	0.0
2	220238.0	-1.458910	484710.0	388499.0	0.669438	2.159500	466224.0	209644.0	1.437090	-1.480850	...	0.0	0.0
3	237999.0	1.306910	913056.0	245434.0	-1.987670	0.978061	228497.0	202999.0	0.478876	-2.018830	...	0.0	0.0
4	229855.0	-0.953545	302325.0	294936.0	0.181659	3.036340	360228.0	251837.0	-0.890300	0.173217	...	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2817	606812.0	-0.442706	883702.0	812099.0	0.416829	1.870050	378177.0	358172.0	0.028231	-1.833950	...	0.0	0.0
2818	814470.0	-1.950550	1538040.0	1513350.0	0.048340	0.964906	893118.0	386514.0	1.479800	-2.759750	...	0.0	0.0
2819	1087940.0	0.885948	819996.0	680139.0	-0.623360	-2.586290	419022.0	367481.0	0.490984	-1.539520	...	0.0	0.0
2820	645607.0	2.499230	514778.0	436949.0	0.509218	0.017766	773149.0	395855.0	-1.275240	-1.313080	...	0.0	0.0
2821	452120.0	-1.154160	486474.0	332515.0	0.927945	0.268392	925750.0	311716.0	-1.750270	2.903240	...	0.0	0.0

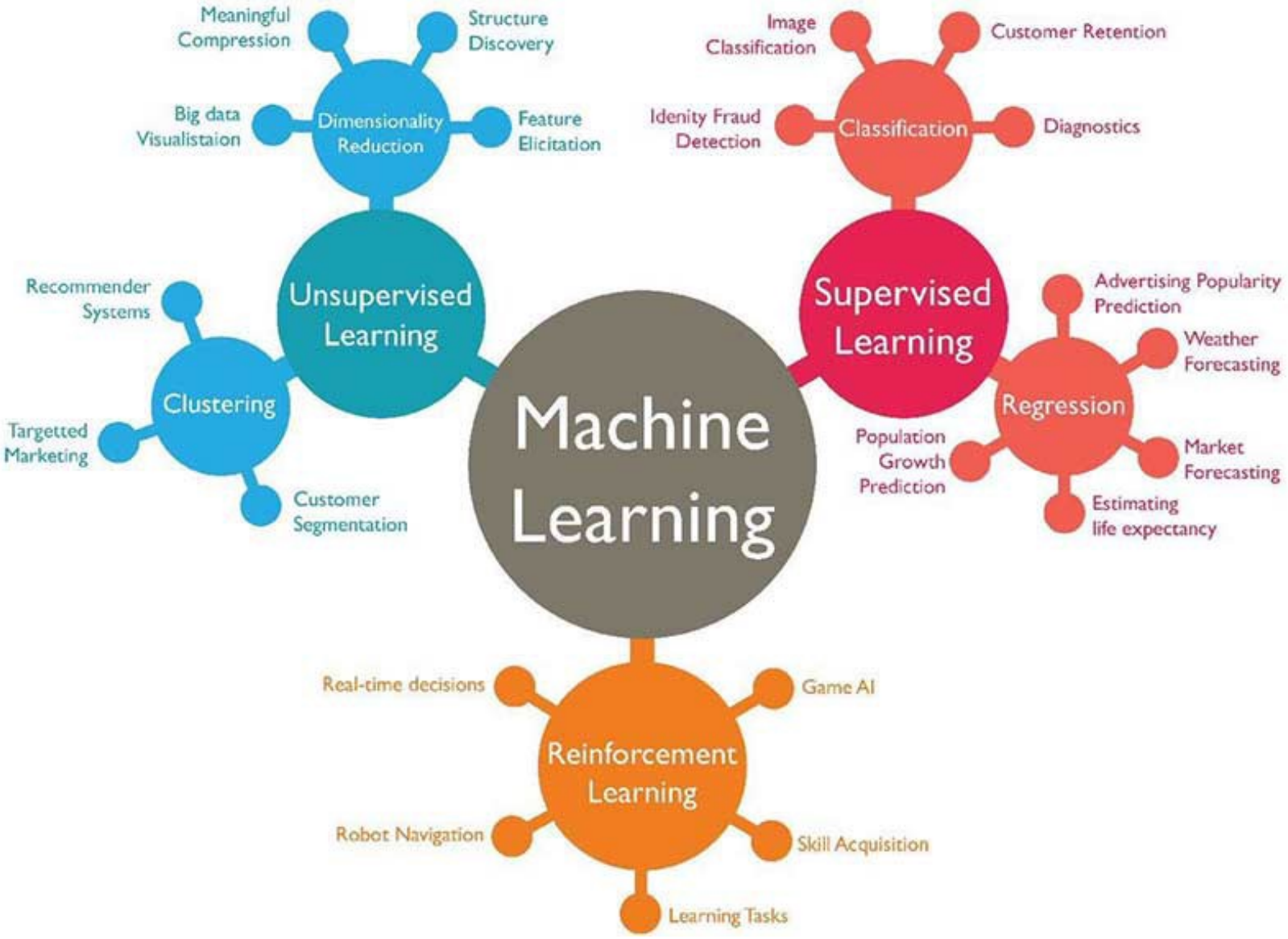
252851 rows × 59 columns

In [16]:
```
full_data.isnull().values.any()
```

Out[16]:
```
False
```

We have the data!!!

We can start wit a simple unsupervised analysis with all data


Machine Learning
Unsupervised Learning
Dimensionality Reduction
Meaningful Compression
Structure Discovery
Big data Visualistaion
Feature Elicitation
Clustering
Recommender Systems
Targetted Marketing
Customer Segmentation
Supervised Learning
Classification
Image Classification
Customer Retention
Idenity Fraud Detection
Diagnostics
Regression
Advertising Popularity Prediction
Weather Forecasting
Population Growth Prediction
Market Forecasting
Estimating life expectancy
Reinforcement Learning
Real-time decisions
Game AI
Robot Navigation
Skill Acquisition
Learning Tasks

We can use Scikit learn for a quick test:

In [17]:

```
embeddings = {
    "Random projection embedding": SparseRandomProjection(
        n_components=3, random_state=42
    ),
    "Truncated SVD embedding": TruncatedSVD(n_components=3),
    "Random Trees embedding": make_pipeline(
        RandomTreesEmbedding(n_estimators=200, max_depth=5, random_state=0),
        TruncatedSVD(n_components=3),
    ),
}
```

In [18]:

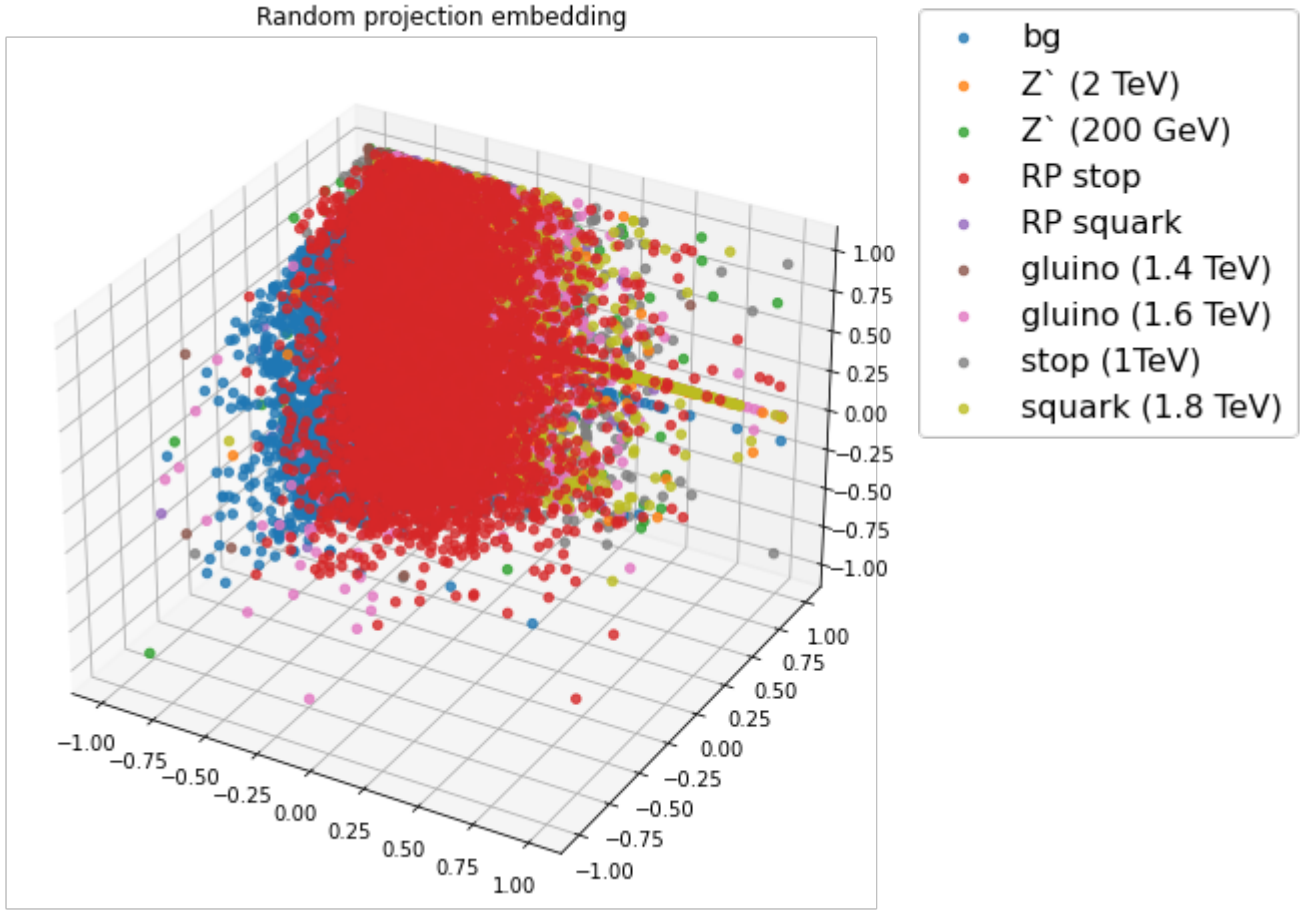
```
from time import time

projections, timing = {}, {}
for name, transformer in embeddings.items():
    data, y = full_data.drop(labels='label',axis=1), full_data['label']

    print(f"Computing {name}...")
    start_time = time()
    projections[name] = transformer.fit_transform(data, y)
    timing[name] = time() - start_time
```

```
Computing Random projection embedding...
Computing Truncated SVD embedding...
Computing Random Trees embedding...
```

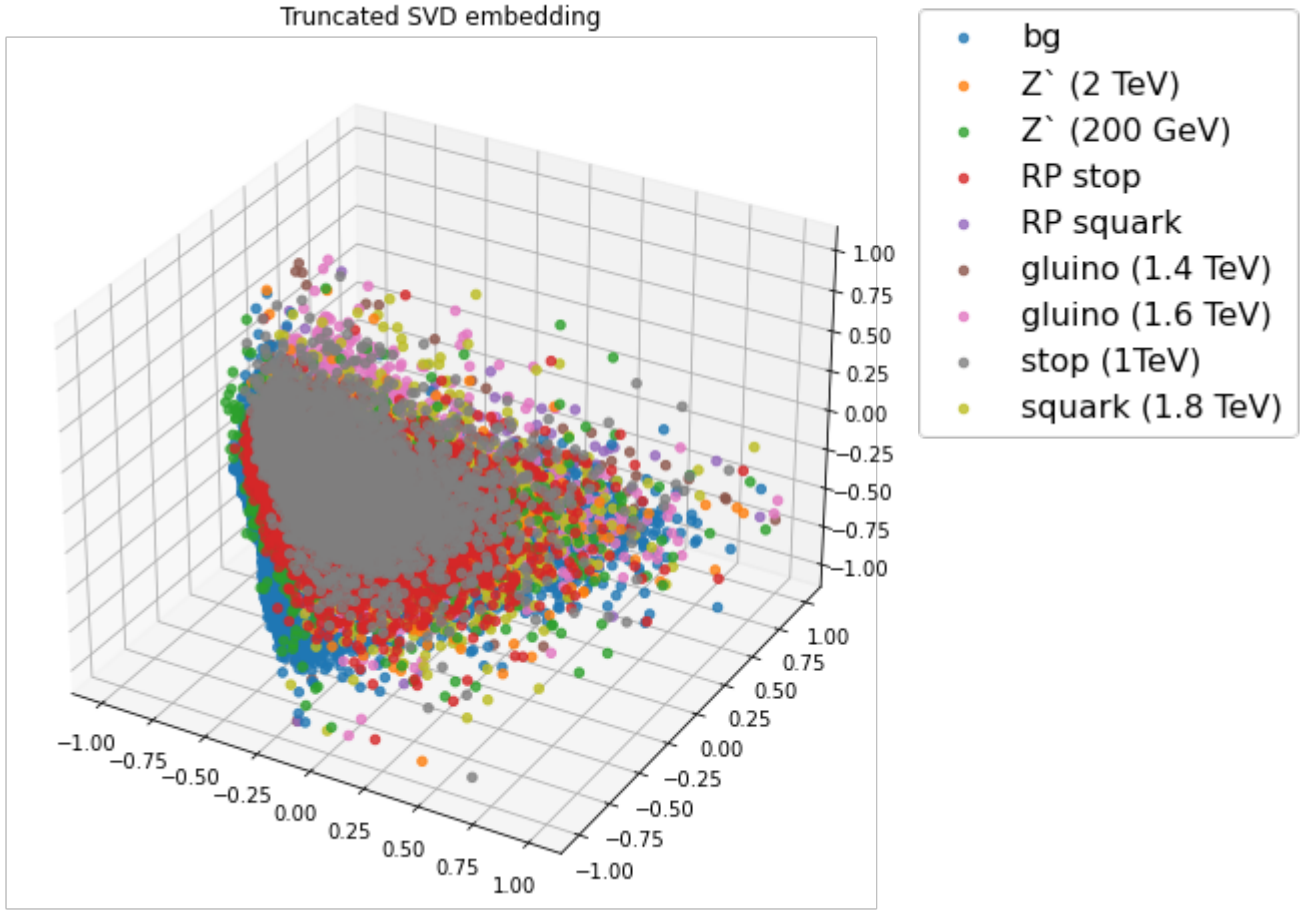
In [20]:

```
def plot_embedding(dataframe, model, title):
    labels = dataframe['label'].unique()
    names = ['bg', 'Z` (2 TeV)', 'Z` (200 GeV)', 'RP stop', 'RP squark', 'gluino (1.4 TeV)',
    projections = [MinMaxScaler(feature_range=(-1, 1)).fit_transform(model.transform(datafra
    colors_p = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brown',
    fig = plt.figure(figsize=(12,8))
    ax = fig.add_subplot(1, 1, 1)
    ax = fig.gca(projection='3d')
    for label, projection, colors_p in zip(labels, projections, colors_p):
        ax.scatter(projection[:,0],projection[:,1],projection[:,2],c=colors_p,alpha=0.8, lab
    ax.set_title(title)
    ax.legend(fontsize=16, bbox_to_anchor=(1.5, 1.05))
```

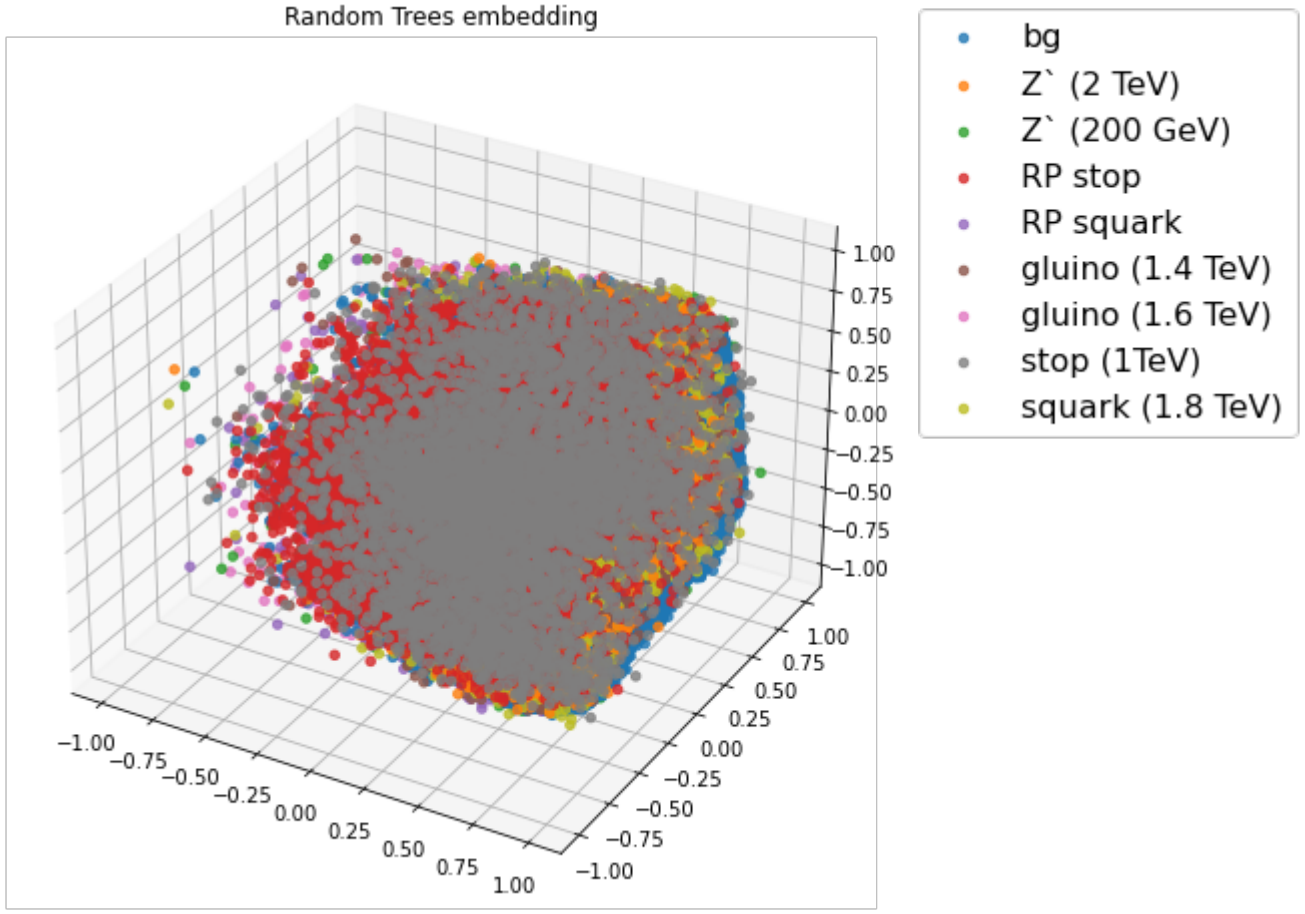
In [21]:
```
plot_embedding(full_data,embeddings['Random projection embedding'],'Random projection embedc
```



In [26]:
```
plot_embedding(full_data,embeddings['Truncated SVD embedding'],'Truncated SVD embedding')
```



In [27]:
```
plot_embedding(full_data,embeddings['Random Trees embedding'],'Random Trees embedding')
```



Can we do better?

Try a DL model

DM COLLIDER PHENOMENOLOGY

We need three ingredients

- Dataset + Dataloader
- Model
- Train loop

Dataset + Dataloader

In [22]:
```
class my_Dataset(Dataset):
    def __init__(self, dataframe, scaler):
        self.data = dataframe.drop(labels='label',axis=1)
        self.y = dataframe['label']
        self.scaler = scaler
        self.scaler.fit(self.data)
        data_t = self.scaler.transform(self.data)
        self.dataframe = pd.DataFrame({k:data_t[:,i] for i,k in enumerate(self.data.keys())})



    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, index):
        x = self.dataframe.iloc[index]
        y = self.y.iloc[index]
        x = torch.from_numpy(x.values).float()

        return x, y

    def get_scaller(self):
        return self.scaler
```

In [23]:
```
DM_dataset = my_Dataset(full_data, MinMaxScaler(feature_range=(-1,1)))
```

In [24]:

```
# spliting into Train and validation dataset
len_DM = len(DM_dataset)
len_train = int(.8*len_DM)
len_val = len_DM - len_train
train_ds, val_ds = random_split(DM_dataset, [len_train, len_val])
```

In [25]:

```
# building the Dataloaders
train_dl = DataLoader(train_ds, batch_size=512, shuffle=True, pin_memory=True)
val_dl = DataLoader(val_ds, batch_size=512, shuffle=False, pin_memory=True)
```

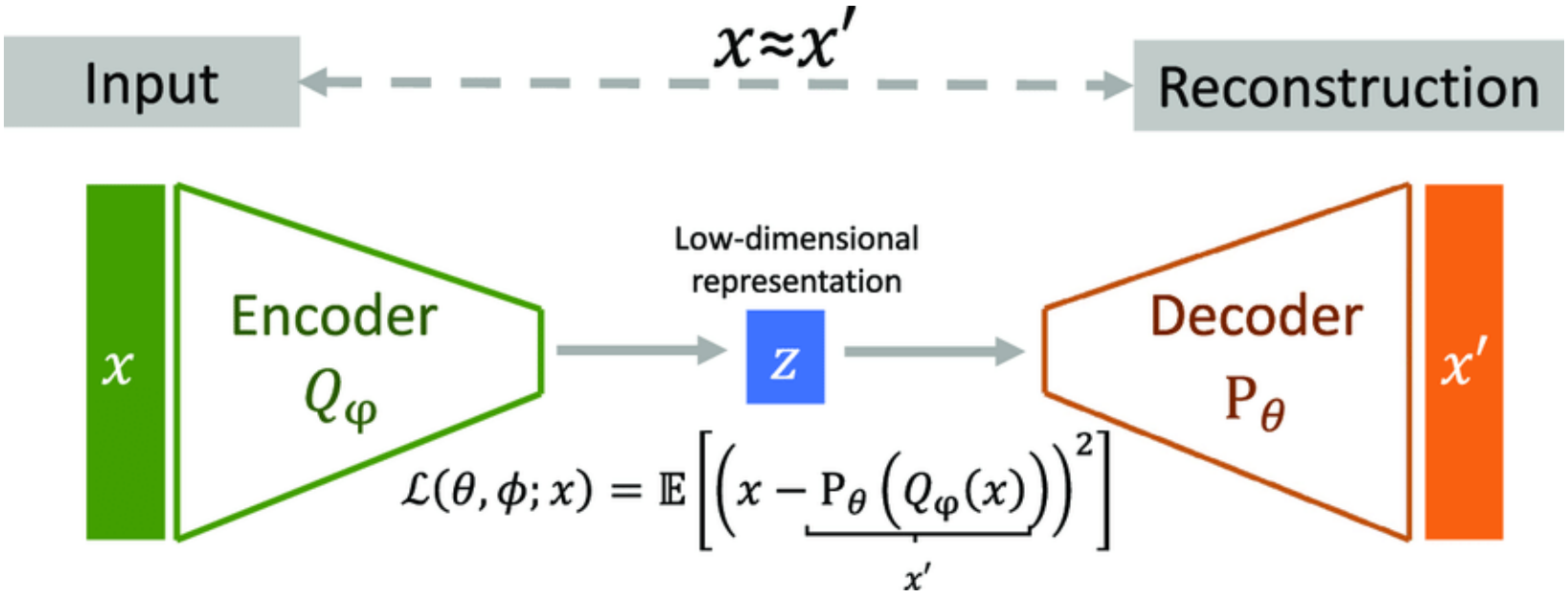
Chek one batch:

In [26]:
```
bt = next(iter(train_dl))
```

In [27]:
```
bt
```

Out[27]:
```
[tensor([[-0.9868,  0.8095, -0.7939,  ..., -1.0000, -1.0000, -1.0000],
         [-0.9440, -0.6934, -0.7961,  ..., -1.0000, -1.0000, -1.0000],
         [-0.6887,  0.5252, -0.8419,  ..., -1.0000, -1.0000, -1.0000],
         ...,
         [-0.9159,  0.6738, -0.3327,  ..., -1.0000, -1.0000, -1.0000],
         [-0.9308,  0.3893, -0.7457,  ..., -1.0000, -1.0000, -1.0000],
         [-0.9332,  0.9672, -0.8745,  ..., -1.0000, -1.0000, -1.0000]]),
 tensor([0, 0, 5, 4, 0, 0, 0, 3, 0, 0, 0, 0, 0, 3, 0, 0, 0, 1, 0, 0, 0, 7, 0, 0,
         0, 0, 0, 3, 7, 8, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 4,
         0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 3, 0, 6, 0, 6, 0, 7, 0, 0, 0, 0, 0, 6, 0,
         0, 1, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 7, 7, 0, 0, 6, 0, 0, 0, 0, 0, 0,
         0, 0, 3, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 6,
         0, 7, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 2,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
         5, 0, 0, 2, 0, 3, 0, 3, 0, 0, 0, 0, 0, 0, 8, 0, 0, 8, 6, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 6, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0,
         0, 0, 3, 0, 0, 2, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0,
         0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 7,
         0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 6, 0, 0, 5, 7, 6, 0, 0,
         0, 0, 0, 0, 3, 0, 5, 0, 6, 0, 0, 0, 6, 0, 0, 3, 0, 7, 0, 0, 0, 0, 0, 0,
         0, 0, 6, 7, 5, 3, 0, 7, 0, 0, 2, 0, 0, 0, 0, 6, 3, 8, 0, 0, 3, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 7, 0, 0, 3, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0,
         0, 0, 0, 0, 0, 0, 0, 0])]
```

The model


$x \approx x'$
Input
Reconstruction
x
Encoder
Q_φ
Low-dimensional
representation
z
Decoder
P_θ
x'
$\mathcal{L}(\theta, \phi; x) = \mathbb{E}\left[\left(x - P_\theta\left(Q_\varphi(x)\right)\right)^2\right]$
x'

In [29]:
```
bt[0].shape
```

Out[29]:
```
torch.Size([512, 58])
```

In [30]:
```
class autoencoder(nn.Module):
    def __init__(self):
        super(autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(58, 128),
            nn.ReLU(True),
            nn.Linear(128, 64),
            nn.ReLU(True),
            nn.Linear(64, 12),
            nn.ReLU(True),
            nn.Linear(12, 3))
        self.decoder = nn.Sequential(
            nn.Linear(3, 12),
            nn.ReLU(True),
            nn.Linear(12, 64),
            nn.ReLU(True),
            nn.Linear(64, 128),
            nn.ReLU(True),
            nn.Linear(128, 58),
            nn.Tanh())

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

Now the train loop:

- criterion
- optimizer
- train loop itself

In [31]:

```
model = autoencoder().to('cuda')
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(
    model.parameters(), lr=1e-3, weight_decay=1e-5)
```

In [32]:

```
len_data_train = len(train_ds)
len_data_val = len(val_ds)
num_epochs = 5
for epoch in range(num_epochs):
    running_training_loss = 0.0
    for data in train_dl:
        kin, _ = data
        kin = kin.to('cuda')
        # ==================forward=====================
        output = model(kin)
        loss = criterion(output, kin)
        running_training_loss += loss.item()
        # ==================backward====================
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    training_loss = running_training_loss/float(len_data_train)
    # ==================log=========================
    print(f'epoch [{epoch + 1}/{num_epochs}], train loss:{training_loss:.4e}')
    # ==================val=========================
    running_val_loss = 0.0
    for data in val_dl:
        kin, _ = data
        kin = kin.to('cuda')
        # ==================forward=====================
        with torch.no_grad():
            output = model(kin)
            loss = criterion(output, kin)
            running_val_loss += loss.item()
    val_loss = running_val_loss/float(len_data_val)
    # ==================log=========================
    print(f'epoch [{epoch + 1}/{num_epochs}], val loss:{val_loss:.4e}')
    print('-*-'*10)
```

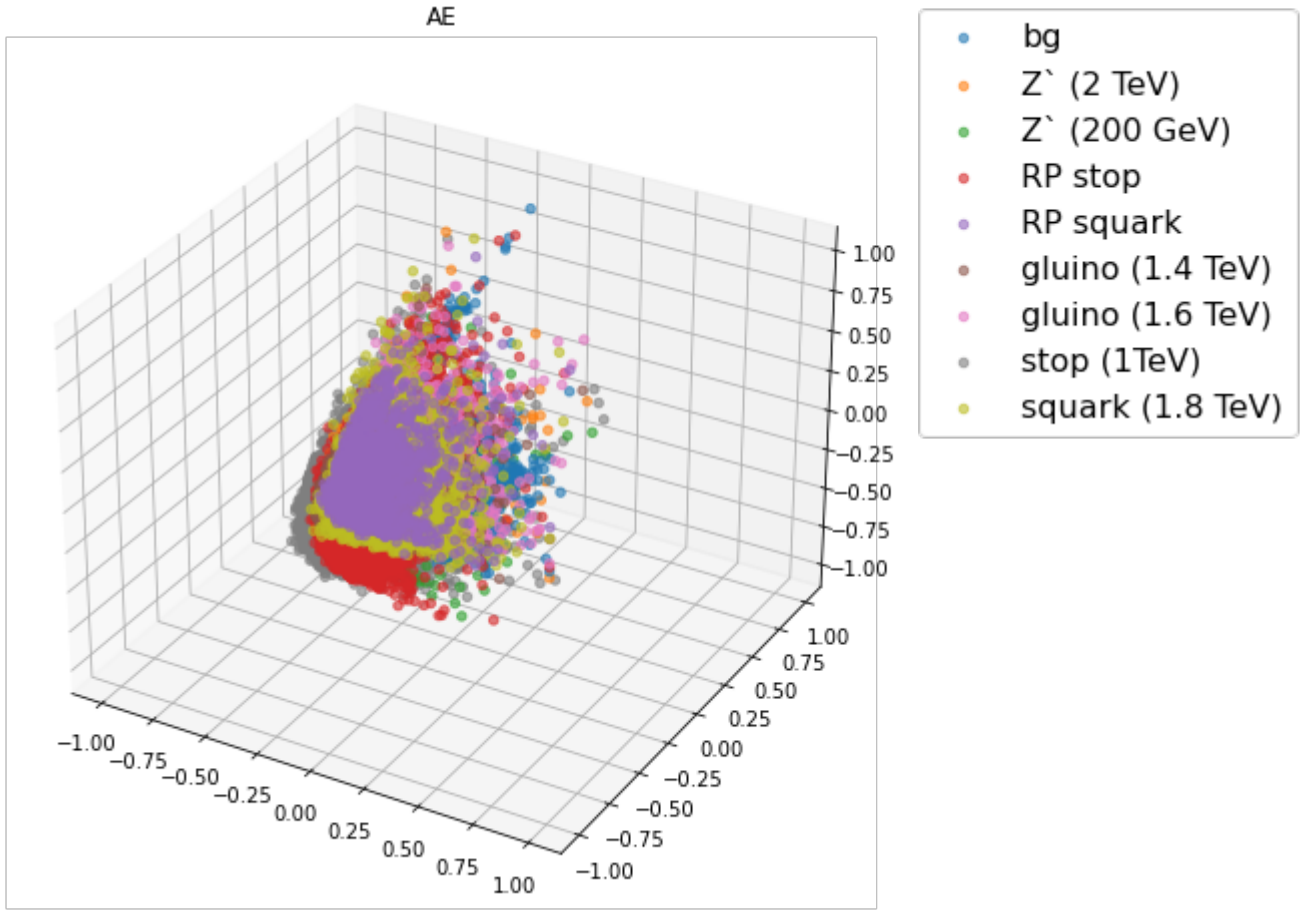
```
epoch [1/5], train loss:9.4322e-05
epoch [1/5], val loss:3.7158e-05
-*--*--*--*--*--*--*--*--*--*-
epoch [2/5], train loss:3.1551e-05
epoch [2/5], val loss:2.8939e-05
-*--*--*--*--*--*--*--*--*--*-
epoch [3/5], train loss:2.8643e-05
epoch [3/5], val loss:2.8484e-05
-*--*--*--*--*--*--*--*--*--*-
```

```
epoch [4/5], train loss:2.7483e-05
epoch [4/5], val loss:2.7061e-05
-*--*--*--*--*--*--*--*--*-
epoch [5/5], train loss:2.6276e-05
epoch [5/5], val loss:2.5503e-05
-*--*--*--*--*--*--*--*--*-
```

In [33]:

```python
def plot_embedding_AE(dataframe, dataset, model, title):
    model.eval()
    scaler = dataset.get_scaller()
    labels = dataframe['label'].unique()
    names = ['bg', 'Z` (2 TeV)', 'Z` (200 GeV)', 'RP stop', 'RP squark', 'gluino (1.4 TeV)',
    colors_p = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brown',
    projections = {}
    with torch.no_grad():
        for i,name in enumerate(names):
            data = torch.from_numpy(dataframe.drop(labels='label', axis=1).loc[full_data['la
            projections[name] = MinMaxScaler(feature_range=(-1,1)).fit_transform(model.encoc
    fig = plt.figure(figsize=(12,8))
    ax = fig.add_subplot(1, 1, 1)
    ax = fig.gca(projection='3d')
    for name, colors_p in zip(names, colors_p):
        ax.scatter(projections[name][:,0],
                   projections[name][:,1],
                   projections[name][:,2],
                   c=colors_p,alpha=0.6,
                   label=name)
    ax.set_title(title)
    ax.legend(fontsize=16, bbox_to_anchor=(1.5, 1.05))
```

In [39]:
```
plot_embedding_AE(full_data, DM_dataset, new_model.to('cuda'), 'AE')
```



Saving and Loading Models

When it comes to saving and loading models, there are three core functions to be familiar with:

- torch.save: Saves a serialized object to disk. This function uses Python's pickle utility for serialization. Models, tensors, and dictionaries of all kinds of objects can be saved using this function.
- torch.load: Uses pickle's unpickling facilities to deserialize pickled object files to memory. This function also facilitates the device to load the data into (see Saving & Loading Model Across Devices).
- torch.nn.Module.load_state_dict: Loads a model's parameter dictionary using a deserialized state_dict.

what is a state_dict ?

A state_dict is simply a Python dictionary object that maps each layer to its parameter tensor.

In [35]:

```python
# Print model's state_dict
print("Model's state_dict:")
for param_tensor in model.state_dict():
    print(param_tensor, "\t", model.state_dict()[param_tensor].size())
```

```
Model's state_dict:
encoder.0.weight         torch.Size([128, 58])
encoder.0.bias   torch.Size([128])
encoder.2.weight         torch.Size([64, 128])
encoder.2.bias   torch.Size([64])
encoder.4.weight         torch.Size([12, 64])
encoder.4.bias   torch.Size([12])
encoder.6.weight         torch.Size([3, 12])
encoder.6.bias   torch.Size([3])
decoder.0.weight         torch.Size([12, 3])
decoder.0.bias   torch.Size([12])
decoder.2.weight         torch.Size([64, 12])
decoder.2.bias   torch.Size([64])
decoder.4.weight         torch.Size([128, 64])
decoder.4.bias   torch.Size([128])
decoder.6.weight         torch.Size([58, 128])
decoder.6.bias   torch.Size([58])
```

Save a model:

In [53]:
```
torch.save(model.state_dict(), '/media/felipe/home 2/COST lectures and codes/models/AE test
```

Loading a model

In [37]:
```
new_model = autoencoder()
new_model.load_state_dict(torch.load('/media/felipe/home 2/COST lectures and codes/models/AE
```

Out[37]:
```
<All keys matched successfully>
```

In [40]:

```
%%html
<iframe src="https://playground.tensorflow.org" width="1200" height="1000"></iframe>
```

Tinker With a **Neural Network** in Your Br(https://gi
/tensorflo
/playgrou
Don't Worry, You Can't Break It. We Promise

Epoch
000,000

Learning rate: 0.03

Activation: Tanh

Regularization: None

Regul: 0

DATA

Which dataset do you want to use?

Ratio of training to test data: 50%

FEATURES

Which properties do you want to feed in?

X1

X2

X12

+ − 2 HIDDEN LAYERS

+ − 4 neurons

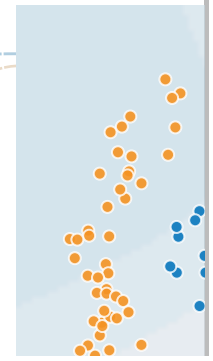
+ − 2 neurons

The outputs are mixed with varying weights, shown by

OUTPUT

Test loss 0.512

Training loss 0.5



Thanks (obrigado)