Jet Tagging and architectures used

Representing jets as images has its roots in the reconstruction of jets with calorimeters. A calorimeter measures the energy deposition of a jet on fine-grained spatial cells. Treating the energy deposition on each cell as the pixel intensity naturally creates an image for a jet. Although useful, this image based technique is very computationally inefficient due to its sparse nature, where 90% of the pixels are blank.

A more natural way to represent jets would be to represent them as an unordered and permutation-invariant set of particles. We can think of these representations of a jet as ***particle clouds*** , analogous to the point cloud representation of 3D shapes used in computer vision.

How do Graph Neural Networks help?

One of the justifications for representing these particle clouds as graphs can be sought from CNNs.

CNNs are widely known for their success in all kinds of machine learning tasks on visual images. They owe their success to mainly two key features:

First, the convolution operation exploits translational symmetry of images by using shared kernels across the whole image. This not only greatly reduces the number of parameters in the network but also allows the parameters to be learned more effectively, as each set of weights will use all locations of the image for learning.

Second, CNNs exploit a hierarchical approach for learning image features. The convolution operations can be effectively stacked to form a deep network. Different layers in the CNNs have different receptive fields and therefore can learn features at different scales, with the shallower layers exploiting local neighbourhood information and the deeper layers learning more global structures. Such a hierarchical approach proves an effective way to learn images.

We could therefore try to implement a similar approach on particle cloud data as well. However, regular convolution operation cannot be applied on point clouds, as the points there can be distributed irregularly, rather than following

some uniform grids as the pixels in an image. Therefore, the basis for a convolution, i.e., a “local patch” of each point on which the convolution kernel operates, remains to be defined for point clouds. Moreover, a regular convolution operation is not invariant under permutation of the points. Thus, the form of a convolution also needs to be modified to respect the permutation symmetry of point clouds. Recently, the edge convolution (“EdgeConv”) operation has been proposed as a convolution-like operation for point clouds. EdgeConv starts by representing a point cloud as a graph, whose vertices are the points themselves, and the edges are constructed as connections between each point to its k nearest neighboring points. In this way, a local patch needed for convolution is defined for each point as the k nearest neighboring points connected to it.

The stackability of EdgeConv operations also brings another interesting possibility. Basically, the feature vectors learned by EdgeConv can be viewed as new coordinates of the original points in a latent space, and then, the distances between points, used in the determination of the k nearest neighbors, can be computed in this latent space. In other words, the proximity of points can be dynamically learned with EdgeConv operations. This results in the DGCNN (Dynamic Graph CNN), in which the graph describing the point clouds are dynamically updated to reflect the changes in the edges, i.e., the neighbors of each point. It has been found that this leads to better performance than keeping the graph static.

This is where the first architecture comes in:

- ParticleNet

The ParticleNet architecture makes extensive use of EdgeConv operations and also adopts the dynamic graph update approach. However, a number of different design choices are made in ParticleNet compared to the original DGCNN to better suit the jet tagging task, including the number of neighbors, the configuration of the MLP in EdgeConv, the use of shortcut connection, etc.

This architecture was introduced in the paper:-

Jet Tagging via Particle Clouds:- https://arxiv.org/pdf/1902.08570.pdf

The second architecture used is very close in implementation to the first and is an attention based architecture.

- ABCNet

The main novelties introduced by ABCNet are the treatment of particle collision data as a set of permutation invariant objects, enhanced by attention mechanisms to filter out the particles that are not relevant for the tasks we want to accomplish. ParticleNet uses a similar approach, using point clouds for jet identification. The main difference between ABCNet and ParticleNet is that ABCNet takes advantage of attention mechanisms to enhance the local feature extraction, allowing for a more compact and efficient architecture.

ABCNet follows closely the implementation described for GAPNet , with key differences to adapt the implementation to our problems of interest. The key aspect of GAPNet is the development of a graph attention pooling layer (GAPLayer) using the edge convolution operation proposed in ParticleNet , which defines a convolution-like operation on point clouds together with attention mechanisms to operate on graph-structured data .

This architecture was proposed in :

ABCNet: An attention-based method for particle tagging : https://arxiv.org/pdf/2001.05311.pdf

Evaluation of performance

The performance of both the architectures on a 100000 training events and 40000 validation events are as following:

For 10 epochs on each, the accuracies were-

- ParticleNet:------ 0.7677
- ABCNet:---------- 0.7762

It can be seen that for small amount of training points, the attention based mechanism has performed better that that with no attention.

As mentioned above, ABCNet takes advantage of attention mechanisms to enhance the local feature extraction, allowing for a more compact and efficient architecture. I believe this is the reason for better performance of ABCNet as compared to ParticleNet.