Alma mater Studiorum – University of Bologna

Deep Embedded Clustering

Report of the project on Deep Embedded Clustering

Artificial Intelligence in Industry

Professor: Michele Lombardi

Ganesh Pavan Kartikeya Bharadwaj Kolluri - ganeshpavan.kolluri@studio.unibo.it
Matricola: 0000976948

GitHub Repository: https://github.com/karthikbharadhwajKB/Deep_Embedded_Clustering-on-HPC-dataset

Introduction

Deep Clustering is a new research area where it's a combination of Deep Learning and Clustering. As it will perform feature representation and cluster assignments simultaneously and it has better results than traditional clustering algorithm like kmeans, Gmm and Hierarchical Clustering. Particularly I have implemented Deep Embedding Clustering as my project. It is state-of-the-art method for deep clustering and often used as benchmark for comparing performance for other models. The uniqueness of this model is it focus on methodology which simultaneously learns feature representation and cluster assignment using deep learning. When coming to traditional clustering algorithms which will focus on distance functions, grouping algorithms to perform clustering.

Dataset

HPC dataset was provided dataset to perform Deep clustering. Data was collected from "Marconi-100" a supercomputer hosted at Cineca. Data was collected by using "examon". The dataset is composed of several files. each file is a node. As I tried to load all files and concatenate them using pandas read_parquet() method because files are in parquet format. But the problem is dataset is huge and it crashed the ram of the google colab environment. As a solution for this problem. I tried with pyspark to load files and then convert them into pandas dataframe. But unfortunately, this solution is not working. Then I contacted Andrea regarding this and now I just loading one file for lack of computational resources. The information collected from Marconi-100 is varied, ranging from the load of the different cores, to the temperature of room where the nodes are located, the speed of fans and memory accessing in writing or reading and etc.

Data Preparation

After loading the data, I have dropped the 'label' and 'timestamp' columns from the dataset. I have checked for any missing data to treat the missing values. There are no missing values in the dataset. We have to split the dataset into X and y for further model training and evaluation of the model. Here we can observe that there are two values in y. 0 means normal state of the node, 2 means anomalous state. So, we are going to map all 2 values with 1 value for our convenience in evaluation part. Here we can see that class 0 has 14,932 entries and class 1 has only 379. There is lot of imbalance ness in the dataset. But these class 1 values are Anomalous state. We can expect that there are very rare events. K-means is sensitive to the scale of feature values because it uses Euclidean distance as similarity metrics. So, we have to scale these features with using Minmax scaler.

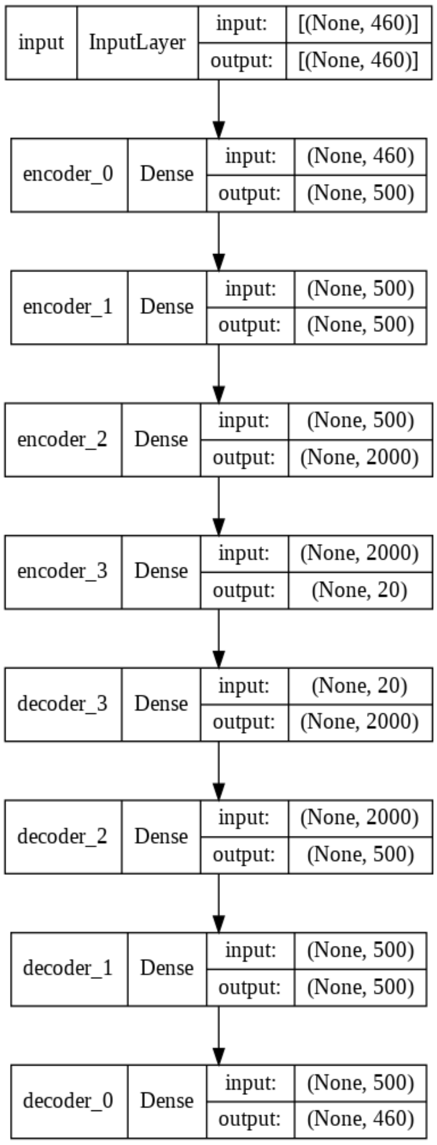
Methodology

The methodology is to simultaneously learn feature representation and cluster assignment. Firstly, we have to transform the data space "X" into a latent feature space "z" using a non-linear mapping with deep neural networks. Deep embedded clustering therefore clusters the data by learning a set of "k" cluster center in feature space "z" and cluster them. There are two phases. In first phase we will initialization parameters of deep auto-encoders and pre-train the model. In second phase, we will optimization the parameters. There are two distribution that we have to optimize. They are auxiliary target distributions and soft clustering assignment, then minimizing the kl-divergence between computed auxiliary target distribution and soft clustering assignment. This soft clustering assignment is initial cluster assignment that will initialized by k-means clustering and then refined by neural networks by optimizing the kl-divergence. At first auto-encoder model will be pre-trained to learn feature representation and then we will discard the decoder and we will attach the low-dimensional feature obtained from encoder to clustering model. Then we will be going to refine the cluster assignment in iteration by training the model with pre-train weights of auto-encoder(fine-tuning) and rest will be trained from scratch with to optimize the multi-loss objectives.

Models

1. Auto-encoder Model

Auto-encoder models are special kind of neural network where it consists of encoder part and decoder part. Encoder part encodes inputs into lower feature representation and decoder then decodes it from lower feature representation to original input space. Generally, we will train this model using reconstruction loss as objective. In this case I have used specific dimension for encoder and decoder parts d-500-500-2000-20. Here d is the input dimensions, and all layers are dense connected. In our case, input shape is 460. In pre-training, I have trained auto-encoder model for 100 epochs with SGD optimizer with 0.1 learning rate and 0.9 momentum. Our objective loss function is mean squared error and batch size is 128. After pre-training I have saved the weights of auto-encoders for further utilization.



input | InputLayer | input: [(None, 460)] | output: [(None, 460)]
encoder_0 | Dense | input: (None, 460) | output: (None, 500)
encoder_1 | Dense | input: (None, 500) | output: (None, 500)
encoder_2 | Dense | input: (None, 500) | output: (None, 2000)
encoder_3 | Dense | input: (None, 2000) | output: (None, 20)
decoder_3 | Dense | input: (None, 20) | output: (None, 2000)
decoder_2 | Dense | input: (None, 2000) | output: (None, 500)
decoder_1 | Dense | input: (None, 500) | output: (None, 500)
decoder_0 | Dense | input: (None, 500) | output: (None, 460)

Figure: Autoencoder model

2. <u>Deep Embedded Clustering Model</u>

Here I have implemented the clustering layer which will do soft clustering assignment(q) and I have implemented auxiliary target distribution(p) method. As for deep embedded clustering we have to refine the cluster assignments in an iterative way, so we have to define some values for training our model. Those values are 2000 max iteration, 140 update interval, 0.001 tolerance threshold to stop training. In iterative training, we will get the soft cluster assignments(q) from our model, and we will pass the soft cluster assignments to the auxiliary target distribution method, and we will update the current soft cluster assignments and we have delta label that will used as stop criterion to stop the iteration. Delta label is sum of correct cluster assignment predictions divided by total number of predictions. If this delta label less than tolerance values, then we will going to stop the iterative training. In this way we are going to train our model. for training this model we have to define our objective. Here we have multiple objectives to minimize. they are kl-divergence and mean squared error and we have tell model that which loss has more weight in training. For this we have to provide loss_weight as 0.5 for ‘kld’ and 1 for ‘mse’.

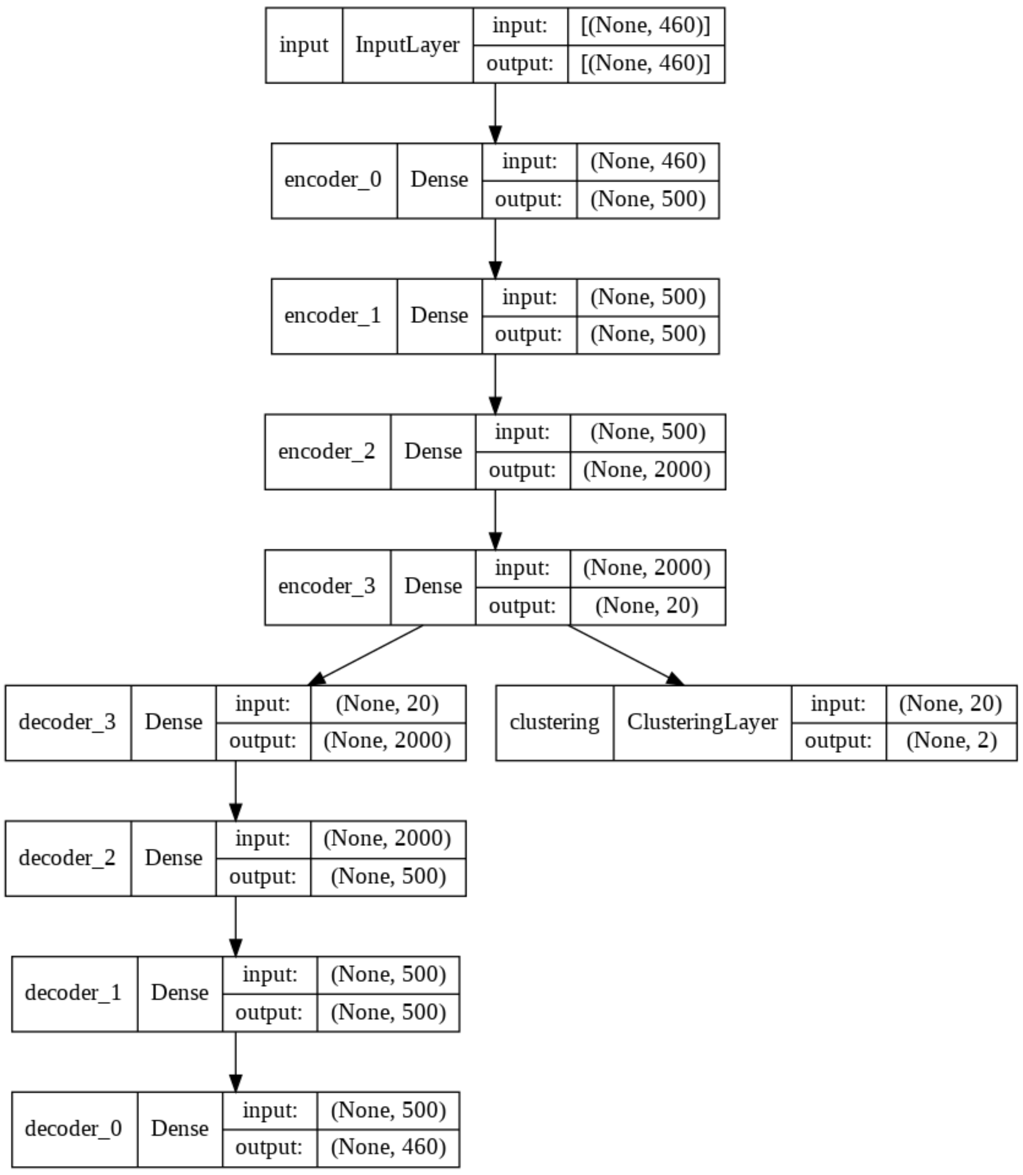

input | InputLayer | input: [(None, 460)] | output: [(None, 460)]
encoder_0 | Dense | input: (None, 460) | output: (None, 500)
encoder_1 | Dense | input: (None, 500) | output: (None, 500)
encoder_2 | Dense | input: (None, 500) | output: (None, 2000)
encoder_3 | Dense | input: (None, 2000) | output: (None, 20)
decoder_3 | Dense | input: (None, 20) | output: (None, 2000)
clustering | ClusteringLayer | input: (None, 20) | output: (None, 2)
decoder_2 | Dense | input: (None, 2000) | output: (None, 500)
decoder_1 | Dense | input: (None, 500) | output: (None, 500)
decoder_0 | Dense | input: (None, 500) | output: (None, 460)

Figure: Deep Embedded Clustering model

Visualization the cluster assignments in embedded space

Here I am going to visualize the cluster assignments in embedded space by using t-Distributed Stochastic Neighbour Embedding (t-SNE) technique.

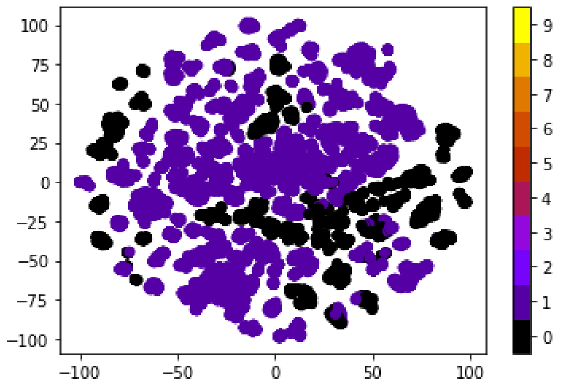


Figure: cluster assignments in embedded space

Experiments

There are lot of experiments I have done on this model like changing of hyper parameters. I have a thought of doing hyper-parameter tuning using keras-tuner. But it will take too much of time to training the model with several combination in search space.

Firstly, I want to do some experiments on Auto-encoder model with bottleneck dimension at first, I took 10-d for the embedded space, and I performed the Deep embedded clustering model and after that I have changed to embedded space dimension to 100-d and training and I have used 50-d for embedded space, and I also used 20-d for embedded space. After trained these models I saved them for further use.

Next, I have experimented with tolerance threshold with these values 0.001, 0.0001, 0.01,0.1 and update interval with these values 100, 120, 140. And max iteration with 1000, 2000, 4000. Finally, I also experimented with loss weights with ‘kld’: 0.1,0.3,0.5,0.8 and set 1 for ‘mse’ as fixed value. And also did some experiments on batch size with 32, 64, 128. But got better results with batch size with 128. I have also experimented the optimizers: Adam and SGD with 0.1 learning rate and 0.9 momentum. As in my experiments I got better results with SGD.

Evaluation of model

1. **Clustering Accuracy**
   I have obtained 0.69 clustering accuracy.
2. **Silhouette Score**
   I have obtained 0.40 silhouette score.
3. **Normalized Mutual Information**
   I have obtained 0.0032 normalized mutual information.

Conclusion

As want to conclude that with my experiments on the model and I have got better results with these hyper-parameters, they are tolerance threshold: 0.01, max iteration: 1000 and update_interval: 140 and loss weight for ‘kld’:0.5. As my further work I will concatenate more nodes to existed node to increase more information for anomalous state of the nodes. Because it is an imbalanced dataset. There are more normal state node than anomalous state. And I will try more combination with hyper-parameters to obtain some better results.

References:

https://deepnotes.io/deep-clustering

https://arxiv.org/abs/1511.06335

https://github.com/XifengGuo/DEC-keras