
Acta Univ. Sapientiae, Informatica **10,** 1 (2018) 86–109

DOI: 10.2478/ausi-2018-0006

Hierarchical clustering with deep Q-learning

Richárd FORSTER
Eötvös University
email: forceuse@inf.elte.hu

Ágnes FÜLÖP
Eötvös University
email: fulop@caesar.elte.hu

Abstract. Following up on our previous study on applying hierarchical clustering algorithms to high energy particle physics, this paper explores the possibilities to use deep learning to generate models capable of processing the clusterization themselves. The technique chosen for training is reinforcement learning, that allows the system to evolve based on interactions between the model and the underlying graph. The result is a model, that by learning on a modest dataset of $10,000$ nodes during 70 epochs can reach $83,77\%$ precision for hierarchical and $86,33\%$ for high energy jet physics datasets in predicting the appropriate clusters.

1 Introduction

Different datasets should be clusterized with specific approaches. For real world networks, hierarchical algorithms, like the Louvain method, provides an efficient way to produce the clusters, that will represent elements, that have a strong connection. In high energy physics clusterization can be done by finding jets using for example the k_t jet clustering. Here a cluster will be a narrow cone of hadrons and other particles produced by the hadronization of a quark or gluon in a heavy ion experiment. Fusing these algorithms a more generic

Computing Classification System 1998: I.1.4
Mathematics Subject Classification 2010: 58A20
Key words and phrases: jet, cluster algorithm, hierarchical clustering, deep q-learning, neural network, multi-core, keras, cntk, louvain

process can be conceived, that was studied in [13]. As a consequence building a graph from the available particles, the same hierarchical clustering can be computed, like on other network related datasets. The graph of the two different input differs only in how the edges are represented, while networks have some strength associated to their connections, the weight of the edges between the different nodes, for particles this weight will be the distance between the elements.

Further generalizing the approach, in this paper a deep learning method is described based on reinforcement learning, that allows the system, to learn to clusterize the input graphs without any external user interaction, relying only on the agent's experience on the graph. This way a single algorithm can be used for the different kind of datasets without any additional specificity in the computational process. In the context of present paper this will lead to a generic model, that is capable to predict the clusterization steps of the elements in different hierarchical clustering tasks.

The evaluation is provided on real world network data taken from the U.S. Census 2010 database. Further data was generated for jet physics using the AliRoot framework [29]. A comparison is given on modularity level of the clusterization between the standard Louvain method and a modified version using the generated model for predicting future communities. Early results shows, that the neural network is capable to achieve an average precision on the Census test dataset of $83,77\%$ and $84,12\%$ for jet dataset, learning for only 70 epochs on a training set consisting from information collected only on the hierarchical dataset.

2 Hierarchical clustering

This section contains a brief introduction of the used hierarchical clustering algorithm and how it was fused with the processes of the jet algorithms from physics to achieve a generalized clustering solution.

2.1 Jet

A jet is a narrow cone of hadrons and other particles which were produced by the hadronization processes of a quark or gluon plasma in a particle physics or heavy ion experiment[27]. The constituent particles are carrying a color charge, such as quarks, they cannot exist in free form due to QCD confinement. This theory allows the colorless states only. The color-free particles formed during the fragmentation process form the jet, because the fragments all tend to

travel in the same direction, creating a narrow jet of particles (Figure 1). Jets are measured in particle detectors and researched to determine the properties of the original quarks. Jets are produced in QCD hard scattering processes, evolving high transverse momentum quarks or gluons, which is called collectively partons in the partonic picture[25]. Perturbative QCD calculations may have colored partons in the final state, but the colorless hadrons are detected in the observed experimental. It can be understand what happened in the detector, if all outgoing colored partons must first undergo parton showering and then combination of the created partons into hadrons.

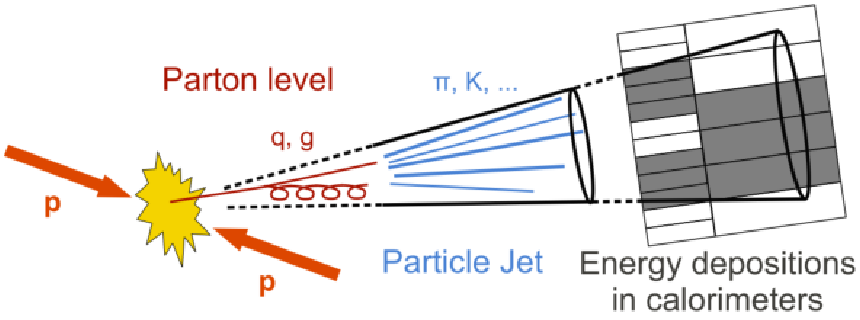

Figure 1: Structure of jet

2.2 Jet clusterisation

The jet clustererisation means when we research the jet momentum owing to the final state particles in the calorimeter [30]. For more accurate understanding the conversion results, we can take into account a muon systems. They form the clusters all together. Two problems should be noted in theoretical research: the infrared (IR) safety and collinear safety. The infrared safe means, that the measured object does not depend on the low energy theoretical physics. The collinear(C) safety is understood, when a parton is substituted by a collinear pair of partons, then the jet clustering outcome does not change. Then the jet can be observed by perturbative technic to apply the experiment, because the jet does not modulate when the particles radiates a very soft objects, or fails to two collinear particles. By theoretical consideration in the case of the infrared divergence the integral of Feynman diagram diverges due to the constituent objects hold very small energy which goes to zero. When the system consists massless particles, it can apply an infrared cutoff and it approximates to zero. The divergence stays finite quantity in the experimental data. Then the in-

frared safe and collinear safe jet reconstruction algorithm can be evaluated for the measurement which satisfies the theoretical considerations or it is used to a given order due to the IRC safe method. Because the jet mass and energy depend on the jet radius, therefore these quantities can be determined more precisely, if the jet radius becomes larger. In the case of the smaller size the cluster consists of more hadronised particles.

Substructure of jet It can be one jet which including more than one group of gaussian- distributed clusters. Substructure is possible a non gaussian component, which is compliances to an offset. It can also contains another gaussian group of clusters, ie. second hard jet. Three different types can be defined:

I: Subjet from uncorrelated sources, overlapping the hard jet which is thought or clustered together with it. This is soft process, derived from proton-leftovers, initial state radiation, beam-rests and/or scatterings, e.g. pileup (PU) and underlying event (UE).

II: Subjet from correlated sources, clustered together with the hard jet considered, coming from the same primary vertex, but another branch of the Feynman diagram.

III: Subjet from correlated sources, deriving from the decay of a single boosted particle, clustered together into a single jet.

2.3 Jet algorithm

During the last 40 years several jet reconstruction algorithms have been developed for hadronic colliders [31, 1]. The first ever jet algorithm was published by Sterman and Weinberg in the 1970's [34]. The cone algorithm plays an important role when a jet consists of a large amount of hadronic energy in a small angular region. It is based on a combination of particles with their neighbours in $\eta - \varphi$ space within a cone of radius $R = \sqrt{(\Delta\varphi^2 + \Delta\eta^2)}$. However the sequential recombination cluster algorithms combine the pairs of objects which have very close p_t values. The particles merge into a new cluster through successive pair recombination. The starting point is the lowest p_t particles for clustering in the k_t algorithm, but in the anti-k_t recombination algorithm it is the highest momentum particles.

The jet clustering involves the reconstructed jet momentum of particles, which leaves the calorimeter together with modified values by the tracker system.

2.3.1 Cone algorithm

The Cone algorithm is one of the regularly used methods at the hadron colliders. The main steps of the iteration are the following [34]: the seed particle i belongs to the initial direction, and it is necessary to sum up the momenta of all particle j, which is situated in a circle of radius R ($\Delta R_{ij}^2 = (y_i - y_j)^2 + (\varphi_i - \varphi_j)^2 < R^2$) around i, where y_i and φ_i are the rapidity and azimuth of particle i.

The direction of the sum is applied as a new seed direction. The iteration procedure is repeated as long as the direction of the determined cone is stable.

It is worth noting what happens when two seed cone overlaps during the iteration. Two different groups of cone algorithms are discussed: One possible solution is to select the first seed particle that has the greatest transverse momentum. Have to find the corresponding stable cone, i.e. jet and delete the particles from the event, which were included in the jet. Then choose a new seed, which is the hardest particle from the remaining particles, and apply to search the next jet. The procedure is repeated until there is no particle that has not worked. This method avoids overlapping.
Other possibility is the so called "overlapping" cones with the split-merge approach. All the stable cones are found, which are determined by iteration from all particles. This avoids the same particle from appearing in multiple cones. The split-merge procedure can be used to consider combining pair of cones. In this case more than a fraction f of the transverse momentum of the softer cones derives from the harder particles; otherwise the common particles assigned to the cone, which is closer to them. The split-merge procedure applies the initial list of protojets, which contains the full list of stable cones:

1. Take the protojet with the largest p_t (i.e. hardest protojet), label it a.
2. Search the next hardest protojet that shares particles with a (i.e. overlaps), label it b. If no such protojet exists, delete a from the list of protojets and add it to the list of final jets.
3. Determine the total p_t of the particles, which is shared between the two protojets, $p_{t,shared}$.
   - If $p_{t,shared} > f$, where f is a free parameter, it is called the overlap threshold, replace protojets a and b with a single merged protojet.
   - Otherwise the protojets are scattered, for example assigning the shared particles to the protojet whose axis is closer.
4. Repeat from step 1 as long as there are protojets left.

A similar procedure to split-merge method is the so called split-drop, where the non-shared particles, which fall into the softer of two overlapping cones are dropped, i.e. are deleted from the jets altogether.

2.3.2 Sequential recombination jet algorithm

They go beyond just finding jets and implicitly assign a clustering sequence to an event, which is often closely connected with approximate probabilistic pictures that one may have for parton branching. The current work focuses on the k_t algorithm, whose parallelization was studied in [11] and [12].

The k_t algorithm for hadrons In the case of the proton-proton collision, the variables which are invariant under longitudinal boots are applied. These quantities which were introduced by [5] and the distance measures are longitudinally invariant as the following:

$$d_{ij} = \min\left(p_{ti}^2, p_{tj}^2\right)\Delta R_{ij}^2, \qquad \Delta R_{ij} = (y_i - y_j)^2 + (\varphi_i - \varphi_j)^2 \tag{1}$$

$$d_{iB} = p_{ti}^2. \tag{2}$$

In this definition the two beam jets are not distinguished.

If $p = -1$, then it gives the "anti-k_t" algorithm. In this case the clustering contains hard particles instead of soft ones. Therefore the jets extend outwards around hard seeds. Because the algorithm depends on the energy and angle through the distance measure, therefore the collinear branching will be collected at the beginning of the sequence.

2.4 The Louvain algorithm

The Louvain method [4], is a multi-phase, iterative, greedy hierarchical clusterization algorithm, working on undirected, weighted graphs. The algorithm processes through multiple phases, within each phase multiple iterations until a convergence criteria is met. Its parallelization was explored in [22], that was further evolved into a GPU based implementation as was detailed in [10]. The modularity is a monotonically increasing function, spreading across multiple iterations, giving a numerical representation on the quality of the clusters. Because the modularity is monotonically increasing, the process is guaranteed to terminate. Running on a real world dataset, termination is achieved in not more than a dozen iterations.

2.4.1 Modularity

On a set, $S = C_1, C_2, ..., C_k$, containing every community in a given partitioning of V, where $1 \leq k \leq N$ and V is the set of nodes, N is the number of nodes. Modularity Q is given by the following [26]:

$$Q = \frac{1}{2W} \sum_{i \in V} e_{i \to C(i)} - \sum_{C \in S} \left(\frac{deg_C}{2W} \cdot \frac{deg_C}{2W} \right), \tag{3}$$

where deg_C is the sum of the degrees of all the nodes in community C, $e_{i \to C(i)}$ is the sum of weights of all edges connecting node i to all nodes in community $C(i)$ and W is the sum of the weight of all the edges.

Modularity has multiple variants, like the ones described in [37, 2, 3]. Yet the one defined in Eq. (3) is the more commonly used.

2.5 Hierarchical k_t clustering

In [13] it was studied how to do hierarchical clustering, following the rules of the k_t algorithm. First the list of particles has to be transformed into a graph, with the particles themselves appointed as nodes. The distance between the elements is a suitable selection for a weight to all edges between adjacent particles. But as it eventually leads up to $n * (n-1)/2$ links, where n is the number of nodes, a better solution is to make connections between nearest neighbours and to the second to nearest. If the particle's nearest "neighbour" is the beam, it will be represented with an isolated node. While the Louvain algorithm relies on modularity gain to drive the computation, the jet clustering variant doesn't have the modularity calculation, as it is known that the process will end, when all particles are assigned to a jet.

The result of this clustering will still be a dendogram, where the leafs will represent the jets.

3 Basic artificial neural networks

Since the beginning of the 1990s the artificial neural network (ANN) methods are employed widely in the high energy physics for the jet reconstruction and track identification [9, 18]. These methods are well-known in offline and online data analysis also.

Artificial neural networks are layered networks of artificial neurons (AN) in which biological neurons are modeled. The underlying principle of operation is as follows, each AN receives signals from another AN or from environment,

gathers these and creates an output signal which is forwarded to another AN or the environment. An ANN contains one input layer, one or more hidden layers and one output layer of ANs. Each AN in a layer is connected to the ANs in the next layer. There are such kind ANN configurations, where the feedback connections are introduced to the previous layers.

3.1 Architecture

An artificial neuron is denoted by a set of input signals $(x_1, x_2, \dots x_n)$ from the environment or from another AN. A weight w_i $(i = 1, \dots n)$ is assigned to each input signal. If the value of weight is larger than zero then the signal is excited, otherwise the signal is inhibited. AN assembles all input signals, determines a net signal and propagates an output signal.

3.1.1 Types of artificial networks

Some features of neural systems which makes them the most distinct from the properties of conventional computing:

- The associative recognition of complex structures
- Data may be non-complete, inconsistent or noisy
- The systems can train, i.e. they are able to learn and organize themselves
- The algorithm and hardware are parallel

There are many types of artificial neural networks. In the high energy particle physics the so-called multi-layer perception (MLP) is the most widespread. Here a functional mapping from input x_k to output z_k values is realised with a function f_{z_k}:

$$z_k = f_{z_k}\left(\sum_{j=1}^{m+1} w_{k_j} f_{y_j}\left(\sum_{i=1}^{n+1} v_{ji} x_i\right)\right),$$

where v_{ji} are the weights between the input layer and the hidden layer, and w_{kj} are the weights between the hidden layer and the output layer. This type of ANN is called feed-forward multi-layer ANN.

It can be extended into a layer of functional units. In this case an activation function is implemented for the input layer. This ANN type is called functional link ANN. The output of this ANN is similar such as previously ANN, without it has additional layer, which contains q functions $h_l(x_1 \dots x_n)(l = 1 \dots q)$.

The weights between the input layer and the functional layer are $u_{li} = 1$, if h_l depends on x_i, and $u_{li} = 0$ otherwise. The output of this ANN is:

$$z_k = f_{z_k}\left(\sum_{j=1}^{m+1} w_{kj} f_{y_j}\left(\sum_{l=1}^{q+1} v_{jl} h_l(x_1 \dots x_n)\right)\right).$$

The functional link ANNs provides better computational time and accuracy then the simple feed-forward multi-layer ANN.

Application in high-energy physics The first application, which was published in 1988, discussed a recurrent ANN for tracking reconstruction [8]. A recurrent ANN was also used for tracking reconstruction in LEP experiment [28]. An article published about a neural network method which was applied to find efficient mapping between certain observed hadronic kinematical variables and the quark-gluon identify. With this method it is able to separate gluon from quark jets originating from the Monte-Carlo generated e^+e^- events [21]. A possible discrimination method is presented by the combination of a neural network and QCD to separate the quark and gluon jet of e^+e^- annihilation [6].

The neural network clusterisation algorithm was applied for the ATLAS pixel detector to identify and split merged measurements created by multiple charged particles [20]. The neural network based cluster reconstruction algorithm which can identify overlapping clusters and improves overall particle position reconstruction [32].

Artificial intelligence offers the potential to automate challenging data-processing tasks in collider physics. To establish its prospects, it was explored to what extent deep learning with convolutional neural networks can discriminate quark and gluon jets [19].

4 Q-learning

Q-learning is a model-free reinforcement learning technique [35]. The reinforcement learning problem is meant to be learning from interactions to achieve a goal. The learner and decision-maker is called the agent. The thing it interacts with is called the environment, that contains everything from the world surrounding the agent. There's a continuous interaction between, where the agent selects an action and the environment responds by presenting new situations (states) to the agent. The environment also returns rewards, special numerical

values that the agent tries to maximize over time. A full specification of an environment defines a task, that is an instance of the reinforcement learning problem. Specifically, the agent and environment interact at each of a sequence of discrete time steps $t = 0, 1, 2, \ldots$. At each time step t, the agent receives the environment's state, $s_t \in S$, where S is the set of possible states, and based on that it selects an action, $a_t \in \mathfrak{A}(s_t)$, where $\mathfrak{A}(s_t)$ is the set of all available actions in state s_t. At the next time step as a response to the action, the agent receives a numerical reward, $r_{t+1} \in \mathfrak{R}$, and finds itself in a new state, s_{t+1} (Figure 2).

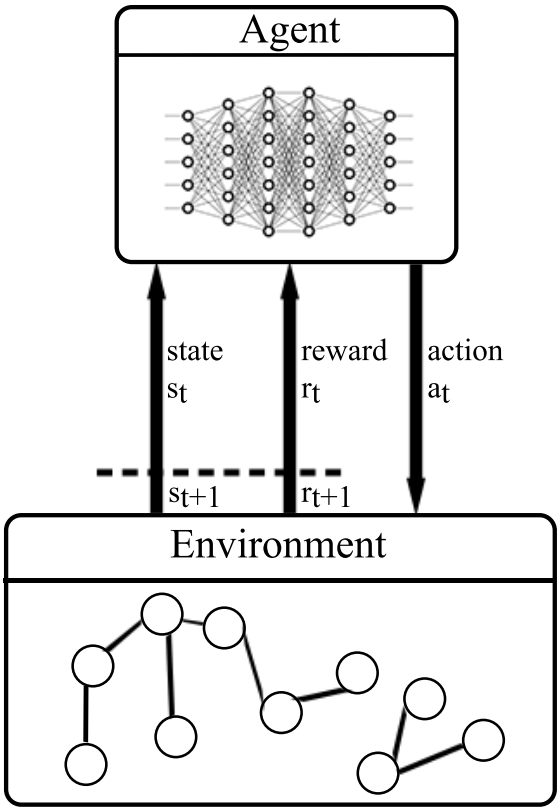

Figure 2: The agent-environment interaction in reinforcement learning

At every time step, the agent implements a mapping from states to probabilities of selecting the available actions. This is called the agent's policy and

is denoted by π_t, where $\pi_t(s, a)$ is the probability that $a_t = a$ if $s_t = s$. Reinforcement learning methods specify how the agent changes this using its experience. The agent's goal is to maximize the total amount of reward it receives over the long run.

4.1 Goals and rewards

The purpose or goal of the agent is formalized in terms of a special reward passed from the environment. At each time step, the reward is a simple number, $r_t \in \Re$. The agent's goal is to maximize the total reward it receives.

4.2 Returns

If the rewards accumulated after time step t is denoted by $r_{t+1}, r_{t+2}, r_{t+3}, \ldots$, what will be maximized by the agent is the expected return R_t, that is defined as some function of the received rewards. The simplest case is the sum of the rewards: $R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T$, where T is the final time step. This approach comes naturally, when the agent-environment interaction breaks into subsequences, or episodes. Each episode ends in a special terminal state, that is then being reset to a standard starting state. The set of all nonterminal states is denoted by S, while the set with a terminal state is denoted by S^+.

Introducing discounting, the agent tries to maximize the the sum of the discounted rewards by selecting the right actions. At time step t choosing action a_t, the discounted return will be defined with Eq. (4).

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{4}$$

where γ is a parameter, $0 \leq \gamma \leq 1$, called the discount rate. It determines the present value of future rewards: a reward received at time step $t + k$ is worth only γ^{k-1} times the immediate reward. If $\gamma < 1$, the infinite sum still is a finite value as long as the reward sequence $\{r_k\}$ is bounded. If $\gamma = 0$, the agent is concerned only with maximizing immediate rewards. If all actions influences only the immediate reward, then the agent could maximize equation 4 by separately maximizing each reward. In general, this can reduce access to future rewards and the return may get reduced. As γ approaches 1, future rewards are used more strongly.

4.3 The Markov property

Assuming a finite set of states and reward values, also considering how a general environment responds at time $t+1$ to the action taken at time t, this response may depend on everything that has happened earlier. In this case only the complete probability distribution can define the dynamics:

$$\Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0\}, \tag{5}$$

for all s',r, and all possible values of the past events: $s_t, a_t, r_t, \ldots, r_1, s_0, a_0$. If the state has the Markov property the environment's response at $t+1$ depends only on the state and action at t and the dynamics can be defined by applying only Eq. (6).

$$\Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t\}, \tag{6}$$

for all s', r, s_t, and a_t. Consequently if a state has the Markov property, then it's a Markov state, only if (6) is equal to (5) for all s', r, and histories, $s_t, a_t, r_t, \ldots, r_1, s_0, a_0$. In this case, the environment has the Markov property.

4.4 Markov cecision process

A reinforcement learning task satisfying the Markov property is a Markov decision process, or MDP. If the state and action spaces are finite, then it is a finite MDP. This is defined by its state and action sets and by the environment's one-step dynamics. Given any state, action pair, (s, a), the probability of each possible next state, s', is

$$P^a_{ss'} = \Pr\{s_{t+1} = s' | s_t = s, a_t = a\}.$$

Having the current state and action, s and a, with any next state, s', the expected value of the next reward can be computed with

$$R^a_{ss'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}.$$

These quantities, $P^a_{ss'}$ and $R^a_{ss'}$, completely specify the most important aspects of the dynamics of a finite MDP.

4.5 Value functions

Reinforcement learning algorithms are generally based on estimating value functions, that are either functions of states or state-action. They estimate

how good a given state is, or how good a given action in the present state is. How good it is, depends on future rewards that can be expected, more precisely, on the expected return. As the rewards received depends on the taken actions, the value functions are defined with respect to particular policies. A policy, π, is a mapping from each state, $s \in S$, and action, $a \in A(s)$, to the probability $\pi(s, a)$ of taking action a while in state s. The value of a state s under a policy π, denoted by $V^\pi(s)$, is the expected return when starting in s and following π. For MDPs $V^\pi(s)$ is defined as

$$V^\pi(s) = E_\pi\{R_t|s_t = s\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s\right\},$$

where E_π is the expected value given that the agent follows policy π. The value of the terminal state is always zero. V^π is the state-value function for policy π. Similarly, the value of taking action a in state s under a policy π, denoted by $Q^\pi(s, a)$ is defined as the expected return starting from s, taking the action a, and following policy π:

$$Q^\pi(s, a) = E_\pi\{R_t|s_t = s, a_t = a\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s, a_t = a\right\}.$$

Q^π is the action-value function for policy π.

V^π and Q^π can be estimated from experience. If an agent follows policy π and maintains an average of the actual return values in each encountered state, then it will converge to the state's value, $V^\pi(s)$, as the number of times that state is encountered approaches infinity. If in a given state every action has a separate average, then these will also converge to the action values, $Q^\pi(s, a)$.

4.6 Optimal value functions

To solve a reinforcement learning task, a specific policy needs to be found, that achieves a lot of reward over the long run. For finite MDPs, an optimal policy can be defined. Value functions define a partial ordering over policies. A policy π is defined to be better than or equal to a policy π' if its expected return is greater than or equal to π' for all states. Formally, $\pi \geq \pi'$ if and only if $V^\pi(s) \geq V^{\pi'}(s)$ for all $s \in S$. At least one policy exists, that is better than or equal to all other policies and this is the optimal policy. If more than one exists, the optimal policies are denoted by π^*. The state-value function among

them is the same, called the optimal state-value function, denoted by V^*, and defined as

$$V^*(s) = \max_{\pi} V^{\pi}(s),$$

for all $s \in S$. The optimal action-value functions are also shared, denoted by Q^*, and defined as

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a),$$

for all $s \in S$ and $a \in A(s)$. For the state-action pair (s, a), this gives the expected return for taking action a in state s and following an optimal policy. Thus, Q^* can be defined in terms of V^* as follows:

$$Q^*(s, a) = E\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\}.$$

5 Clustering with deep Q-learning

The Deep Q-learning (DQL) [23, 24] is about using deep learning techniques on the standard Q-learning (Section 4).

Calculating the Q state-action values using deep learning can be achieved by applying the following extensions to standard reinforcement learning problems:

1. Calculate Q for all possible actions in state s_t,
2. Make prediction for Q on the new state s_{t+1} and find the action $a_{t+1} = \max_a a \in A(s_{t+1})$, that will yield the biggest return,
3. Set the Q return for the selected action to $r + \gamma Q(s_{t+1}, a_{t+1})$. For all other actions the return should remain unchanged,
4. Update the network using back-propagation and mini-batches stochastic gradient descent.

This approach in itself leads to some additional problems. The *exploration-exploitation issue* is related to which action is taken in a given state. By selecting an action that always seems to maximize the discounted future reward, the agent is acting greedy and might miss other actions, that can yield higher overall reward in the long run. To be able to find the optimal policy the agent needs to take some exploratory steps at specific time steps. This is solved by applying the ϵ-*greedy algorithm* [35], where a small probability ϵ will choose a completely random action.

The other issue is the problem of the *local-minima* [36]. During training multiple states can be explored, that are highly correlated and this may make the network to learn replaying the same episode. This can be solved, by first storing past observations in a *replay memory* and taking random samples from there for the mini-batch, that is used to replay the experience.

5.1 Environment

The environment provides the state that the agent will react to. In case of clustering the environment will be the full input graph. The actual state the necessary information required to compute the Louvain method, packaged into a Numpy stack. These include the weights, degrees, number of loops, the actual community and the total weight of the graph. Each state represents one node of the graph with all of its neighbors. The returned rewards for each state will be based on the result of the actual Louvain clusterization, which means during training the environment will compute the real clusters. If the action selected by the agent leads to the best community, that will have a positive reward set to 10000 and in any other case the returned value will be -1000. After stepping, the next state will contain the modified community informations.

The agent's action space is finite and predefined and the environment also has to reflect this. Let the cardinality of the action space be noted for all $s \in S$ states by $|A(s)|$ For this reason, the state of the environment contains information about only $|A(s)|$ neighbors. This can lead to more nodes, than how many really is connected to a given element. In this case the additional dummy node's values are filled with extremals, in the current implementation with negative numbers. One limitation of the actual solution is that if the number of neighbors are higher, than $|A(s)|$, then only the first $|A(s)|$ neighbors will be considered, in the order in which they appear in dataset. The first "neighbor" will be currently evaluated node, so in case the clusterization will not yield any better community, the model should see, that the node stays in place.

To help avoid potential overflow during the computation, weights of the input graph are normalized to be between 0.000001 and 1.

5.2 Agent

The agent acts as the decision maker, selecting the next community for a given node. It takes the state of the environment as an input and gives back the index of the neighbor that is considered to be providing the best community.

5.2.1 Implementation in Keras

Keras [38] is a Python based high-level neural networks API, compatible with the TensorFlow, CNTK, and Theano machine learning frameworks. This API encourages experimentation as it supports rapid development of neural networks. It allows easy and fast prototyping, with a user friendly, modular, and extensible structure. Both *convolutional networks* and *recurrent networks* can be developed, also their combinations are also possible in the same agent. As all modern neural network API it both runs on CPU and GPU for higher performance.

The core data structure is a model, that is a collection of layers. The simplest type is the *Sequential model*, a linear stack of layers. More complex architectures also can be achieved using the Keras functional API.

The clustering agent utilizes a sequential model:

```
from keras.models import Sequential

model = Sequential()
```

Stacking layers into a model is done through the *add* function:

```
from keras.layers import Dense

model.add(Dense(128, input_shape=(self.state_size,
            self.action_size), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
```

The first layer will handle the input and has a mandatory parameter defining its size. In this case *input_shape* is provided as a 2-dimensional matrix, where *state_size* is the number of parameters stored in the state and *action_size* is the number of possible actions. The first parameter tells how big the output dimension will be, so in this case the input will be propagated into a 128-dimensional output.

The following two layers are hidden layers (Section 3) with 128 internal nodes, with *rectified linear unit* (ReLU) activation. The rectifier is an activation function given by the positive part of its argument: $f(x) = x^+ = \max(0, x)$, where x is the input to a neuron. The rectifier was first introduced to a dynamical network in [16]. It has been demonstrated in [14] to enable better

training of deeper networks, compared to the widely used activation function prior 2011, the logistic sigmoid [15].

During training overfitting happens, when the ANN goes to memorize the training patterns. In this case the network is weak in generalizing on new datasets. This appears for example, when an ANN is very large, namely it has too many hidden nodes and hence, there are too many weights which need to be optimized.
The dropout for the hidden layers is used to prevent overfitting on the learning dataset. Dropout is a technique that makes some randomly selected neurons ignored during training. Their contribution to the activation of neurons on deeper layers is removed temporally and the weight updates are not applied back to the neurons. If neurons are randomly dropped during training, then others will have to handle the representation, that is required to make predictions, that is normally handled by the dropped elements. This results in multiple independent internal representations for the given features [33]. This way the network becomes capable of better generalization and avoids potential overfitting on the training data.

The output so far will still be a matrix with the same shape as the input. This is flatten into a 1-dimensional array by adding the following layer:

```
model.add(Flatten())
```

Finally to have the output provide the returns on each available actions, the last layer changes the output dimension to *action_size*:

```
model.add(Dense(self.action_size, activation='linear'))
```

Once the model is set up, the learning process can be configured with the *compile* function:

```
model.compile(loss='mse', optimizer=Adam(lr=self.learning_rate)),
```

where *learning_rate* has been set to 0.001. For the loss function *mean squared error* is used, optimizer is an instance of *Adam* [17] with the mentioned learning rate. The discount rate for future rewards have been set to $\gamma = 0.001$. This way the model will try to select actions, that yield the maximum rewards in the short term. While maximizing the reward in long term can eventually lead to a policy, that computes the communities correctly, choosing it this small makes the model learn to select the correct neighbors faster.

To make a prediction on the current state, the *predict* function is used:

```
model.predict(state.reshape(1, self.state_size,
                            self.action_size))
```

For Keras to work on the input state, it always have to be reshaped into dimensions $(1, \texttt{state\_size}, \texttt{action\_size})$, while the change always has to keep the same number of state elements.

6 Results

Evaluation of the proposed solution is done by processing network clustering on undirected, weighted graphs. These graphs contain real network information, instead of evaluating on physics related datasets (Section 2.3), as it is more suitable for the original Louvain method. Because of this, the modularity can be used as a sort of metric to measure the quality (Subsection 2.4) of the results. Additionally the number of correct predictions and misses are used to describe the deep Q-learning (Section 5) based method's efficiency.

Numerical evaluations are done by generating one iteration on the first level of the dendogram as the top level takes the most time to generate as it is based on all the original input nodes. The GPU implementation of the Louvain method being used was first described in [10].

6.1 Dataset

The proposed model, as well as the Louvain clustering works on undirected, weighted graphs. Such graphs can be generated from U.S. Census 2010 and Tiger/Line 2010 shapefiles, that are freely available from [40] and from jet physics information simulated by the AliRoot framework.

6.1.1 Census dataset

The Census dataset contains the following:

- the vertices are the Census Blocks;
- there's an edge between two vertices if the corresponding Census Blocks share a line segment on their border
- each vertex has two weights:
  - Census2010 POP100 or the number of people living in that Census Block
  - Land Area of the Census Block in square meters
- the edge weights are the pseudo-length of the shared borderlines.
- each Census Block is identified by a point, that is given longitudinal and latitudinal coordinates

A census block is the smallest geographical unit used by the United States Census Bureau for tabulation of 100-percent data. The pseudo-length is given by $\sqrt{(x^2 + y^2)}$, where x and y are the differences in longitudes and latitudes of each line segment on the shared borderlines. The final result is multiplied by 10^7 to make the edge weights integers. For clusterization the node weights are not used.

The matrices used for evaluation contains the information related to New York, Oregon and Texas (Table 1), that was arbitrarily selected from the SuiteSparse Matrix Collection [39]. The graph details can be found in [7].

	New York	Oregon	Texas
Nodes	350,169	196,621	914,231
Edges	1,709,544	979,512	4,456,272

Table 1: Size of the Census datasets

6.1.2 Jet dataset

AliRoot is the Off-line framework for simulation, reconstruction and analysis for CERN's ALICE experiment. The simulation covers all processes of primary collisions and generates the newly created particles, follows through their transportation and calculates the hits in each component.

For the current work the selected dataset is based on the points detected by the system's TPC. The Time Projection Chamber (TPC) detector is the main tracking component of ALICE. Particles passing through this detector ionizes the gas molecules inside and these ionization points are registered [29].

The datasets were simulated with the framework's PbPbbench test application. Three events were generated, the sizes are detailed in Table 2.

	Event1	Event2	Event3
Nodes	140,535	139,162	67,778
Edges	140,535	139,162	67,778

Table 2: Size of the Jet datasets

Due to the limitations of the proposed solution as was described in Subsection 5.1, in all cases only 4 neighbors are kept for each nodes during the computation.

6.2 Precision of the neural network

The precision depends on how well the model can generalize the learned information. In clustering this will highly depend on the structure of the graph. The model described in Section 5 have been trained on a training set built from the Oregon graph. The first 10000 nodes based on the order how they are first mentioned in the original dataset was taken as a subgraph and the Louvain method was applied on it, generating the communities. In each step a matrix was built, where the lines represents the nodes and the columns contains the necessary values for the computation (current community, weight, degree). Learning phase was running for 70 epochs. The ratio of the good and bad predictions for Census data are shown in Table 3 and for jet data are shown in Table 4.

	New York	Oregon	Texas
Positive	310,972	172,028	750,563
Negative	39,197	24,593	163,668

Table 3: Positive/negative predictions of the model for Census data

The deep learning solution's precision in average is 83,77%. Specifically on the datasets it's respectively 87,4%, 85,7% and 78,2%. Precision can be further increased by running the training for more epochs or by further tune the hyper-parameters. Looking at the number of connected components of each graph, New York has 3, Oregon 1 and Texas 1. The highest number of connected components lead to the highest precision, since subgraphs are containing less nodes, than in graphs with less components. As a consequence of this the model performs better with smaller subgraphs, than on more complex structures.

	Event1	Event2	Event3
Positive	122,405	123,436	56,391
Negative	18,130	15,726	11,387

Table 4: Positive/negative predictions of the model dor jet data

The average precision in this case is 86,33%. For each event it's respectively 87,1%, 88,7% and 83,2%. This could be further increased by teaching the model on training sets that also contains information on the jet structure.

6.3 Modularity comparison

The Louvain method assumes nothing of the input graph. The clusterization can be done without any prior information of the groups being present in the network. The modularity (Subsection 2.4.1) for the Census data is presented (Table 5) for all 3 test matrices for both the Louvain algorithm and the deep Q-learning based solution. The same results are presented for the jet data in Table 6.

	New York	Oregon	Texas
Louvain	0.82	0.68	0.76
DQL	0.72	0.58	0.59

Table 5: Modularities achieved by Louvain and with the DQL solution for Census data

The modularities showing similar results to the precision of the network: the New York graph has a modularity less with $14,53\%$ compared to the Louvain computation, while Oregon is less with $13,8\%$ and Texas is less with $17,36\%$. This proves, that by loosing from the precision, the qualities of the clusters do not degrade more than, what is lost on the precision.

	Event1	Event2	Event3
Louvain	0.76	0.78	0.81
DQL	0.66	0.69	0.67

Table 6: Modularities achieved by Louvain and with the DQL solution for jet data

The changes in modularities is comparable to that of the Census dataset results, proving that the generic model learning only on hierarchical data can also work with comparable efficiency on jet related datasets.

7 Summary

In this paper a reinforcement learning based model was devised to use a community processor during the generation the dendogram in clusterization tasks. The training dataset was based on the Louvain method's processing and was learned using deep learning from one of the test graphs. The model detailed in Section 5 predicted the next level of the dendogram with $83,77\%$ of precision

for the Census dataset and 86.33% for the jet dataset, achieving that, while only being learned for 70 epochs and on 10000 nodes, that were selected from the same graph. Reinforcement learning produces a model, that can generalize based on the training data, as such without a broader selection of structures it cannot reach higher efficiency.

8 Future work

Increasing the model's effectiveness, the learning process should be extended, so it will generalize on any kind of graph with arbitrary number of children. Also the training set needs to extended with data from graphs with different structures and complexity. The model should be further evaluated on datasets with different structural complexities. A coherent method to test the performance of the new clusterization should be explored. Thanks to the GPU based deep learning frameworks, the learning rate increases in accordance with the power of the underlying GPU, but how the inferencing performance affects the clustering should be further evaluated.

References

[1] A. Ali, G. Kramer, Jets and QCD: A historical review of the discovery of the quark and gluon jets and its impact on QCD *Eur. Phys. J. H.* **36** (2011) 245–326. [arXiv:1012.2288 [hep-ph]]. ⇒89

[2] D. Bader, J. McCloskey, Modularity and graph algorithms, *SIAM AN10 Minisymposium on Analyzing Massive Real-World Graphs*, 2009, pp. 12-16. ⇒92

[3] J. W. Berry, B. Hendrickson, R. A. LaViolette, C. A. Phillips, Tolerating the community detection resolution limit with edge weighting, *Phys. Rev. E* **83,** 5 (2011) 056119. ⇒92

[4] V. D. Blondel, J-L. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of communities in large networks, *Journal of Statistical Mechanics: Theory and Experiment* **10** (2008) P10008 ⇒91

[5] S. Carani, Yu. L Dokshitzer, M. H. Seymour, B. R. Webher, Longitudinally-invariant $k_\perp$-clustering algorithms for hadron-hadron collisions, *Nuclear Physics B* **406** (1993) 187–224. ⇒91

[6] I. Csabai, F. Czakó, Z. Fodor, Quark- and gluon-jet separations using neural networks, *Phys. Rev. D* **44** 7 (1991) R1905–R1908. ⇒94

[7] T. Davis, Y. Hu, The University of Florida Sparse Matrix Collection, Mathematical Software, Vol 38, Issue 1, 2011, pp 1:1–1:25. ⇒104

[8] B. Denby, Neural networks and cellular automata in experimental high energy physics, *Computer Physics Communications* **49** (1988) 429–448. ⇒94

[9] B. Denby, Neural networks in high energy physics: a ten year perspective, *Computer Physics Communications* **119** (1999) 219. ⇒92
[10] R. Forster, Louvain community detection with parallel heuristics on GPUs, *20th Jubilee IEEE International Conference on Intelligent Engineering Systems* **20** (2016) doi: 10.1109/INES.2016.7555126 ⇒91, 103
[11] R. Forster, A. Fülöp, Jet browser model accelerated by GPUs, *Acta Univ. Sapientiae Informatica* **8,** 2 (2016) 171–185. ⇒91
[12] R. Forster, A. Fülöp, Parallel k_t jet clustering algorithm, *Acta Univ. Sapientiae Informatica* **9,** 1 (2017) 49–64. ⇒91
[13] R. Forster, A. Fülöp, Hierarchical k_t jet clustering for parallel achitectures, *Acta Univ. Sapientiae Informatica* **9,** 2 (2017) 195–213. ⇒87, 92
[14] X. Glorot, A. Bordes, Y- Bengio, Deep sparse rectifier neural networks, *Proc 14th International Conference on Artificial Intelligence and Statistics (AISTATS) 2011*, Fort Lauderdale, FL, USA. Volume 15 of JMLR:W&CP 15. ⇒101
[15] J. Han. C. Moraga, The influence of the sigmoid function parameters on the speed of backpropagation learning, *IWANN '96 Proc. of the Int. Workshop on Artificial Neural Networks: From Natural to Artificial Neural Computation*, 1995, pp. 195-201. ⇒102
[16] R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, H. S. Seung, Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature* **405** (2000) 947-951. ⇒101
[17] D. P. Kingma, J. B. Adam, A method for stochastic optimization, 2014, arXiv:1412.6980 ⇒102
[18] H. Kolanoski, Application of artifical neural networks in particle physics, *Nuclear Instruments and Methods in Physics Research A* **367** (1995) 14–20. ⇒92
[19] P. T. Komiske, E. M. Metodiev, M. D. Schwartz, Deep learning in color: towards automated quark/gluon jet discrimination, *J. High Energy Physics* (2017) 110. ⇒94
[20] K. J. C. Leney, A neural-network clusterisation algorithm for the ATLAS silicon pixel detector, *J. of Physics: Conbnference Series* **523** (2014) 012023. ⇒94
[21] L. Lönnblad, C. Peterson, T. Rögnvaldsson, Using neural networks to identify jets, *Nuclear Physics* **B349** (1991) 675–702. ⇒94
[22] H. Lu, Mahantesh Halappanavar, A. Kalyanaraman, Parallel heuristics for scalable community detection, *Parallel Computing* **47** (2015) 1937. ⇒91
[23] V. Mnih et al., Playing Atari with deep reinforcement learning, 2013, arXiv:1312.5602 ⇒99
[24] V. Mnih et al., Human-level control through deep reinforcement learning, *Nature*, 2015, doi:10.1038/nature14236 ⇒99
[25] T. Muta, *Foundation of Quantum Chrodinamics*, World Scientific Press, 1986. ⇒88
[26] M. E. J. Newman, M. Girvan, Finding and evaluating community structure in networks, *Phys. Rev. E* **69** 2 (2004) 026113. ⇒92
[27] M. E. Peskin, D. V. Schroeder, *Quantum Field Theory*, Westview Press, 1995. ⇒87

[28] C. Peterson, Track finding with neural networks, *Nuclear Instruments and Methods* **A279** (1988) 537. ⇒94
[29] D. Rohr, S. Gorbunov, A. Szostak, M. Kretz, T. Kollegger, T. Breitner, T. Alt, ALICE HLT TPC Tracking of Pb-Pb Events on GPUs, *Journal of Physics: Conference Series* **396** (2012) doi:10.1088/1742-6596/396/1/012044 ⇒87, 104
[30] G. P. Salam, Towards jetography, *Eur. Phys. J.* C**67** (2010) 637-686. ⇒88
[31] S. Salur, Full Jet reconstruction in heavy ion collisions, *Nuclear Physics A* **830,** 1-4 (2009) 139c–146c. ⇒89
[32] K. E. Selbach, Neural network based cluster reconstruction in the ATLAS pixel detector, *Nuclear Instruments and Methods in Physics Research A* **718** (2013) 363–365. ⇒94
[33] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, *JMLR* **15** (2014) 1929-1958. ⇒102
[34] G. Sterman, S. Weinberg, Jets from quantum chromodynamics, *Phys. Rev. Lett.* **39** (1977) 1436. ⇒89, 90
[35] R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction*, A Bradford Book, 1998, ISBN: 978-0262193986 ⇒94, 99
[36] G. Swirszcz, W. M. Czarnecki, R. Pascanu, Local minima in training of neural networks, 2016, arXiv:1611.06310 ⇒100
[37] V. A. Traag, P. Van Dooren, Y. Nesterov, Narrow scope for resolution-limit-free community detection, *Phys. Rev. E* **84,** 1 (2011) 016114. ⇒92
[38] ∗ ∗ ∗ Keras: The Python Deep Learning library ⇒101
[39] ∗ ∗ ∗ SuiteSparse Matrix Collection ⇒104
[40] ∗ ∗ ∗ United States Census Bureau ⇒103

Received: April 5, 2018 • Revised: July 30, 2018