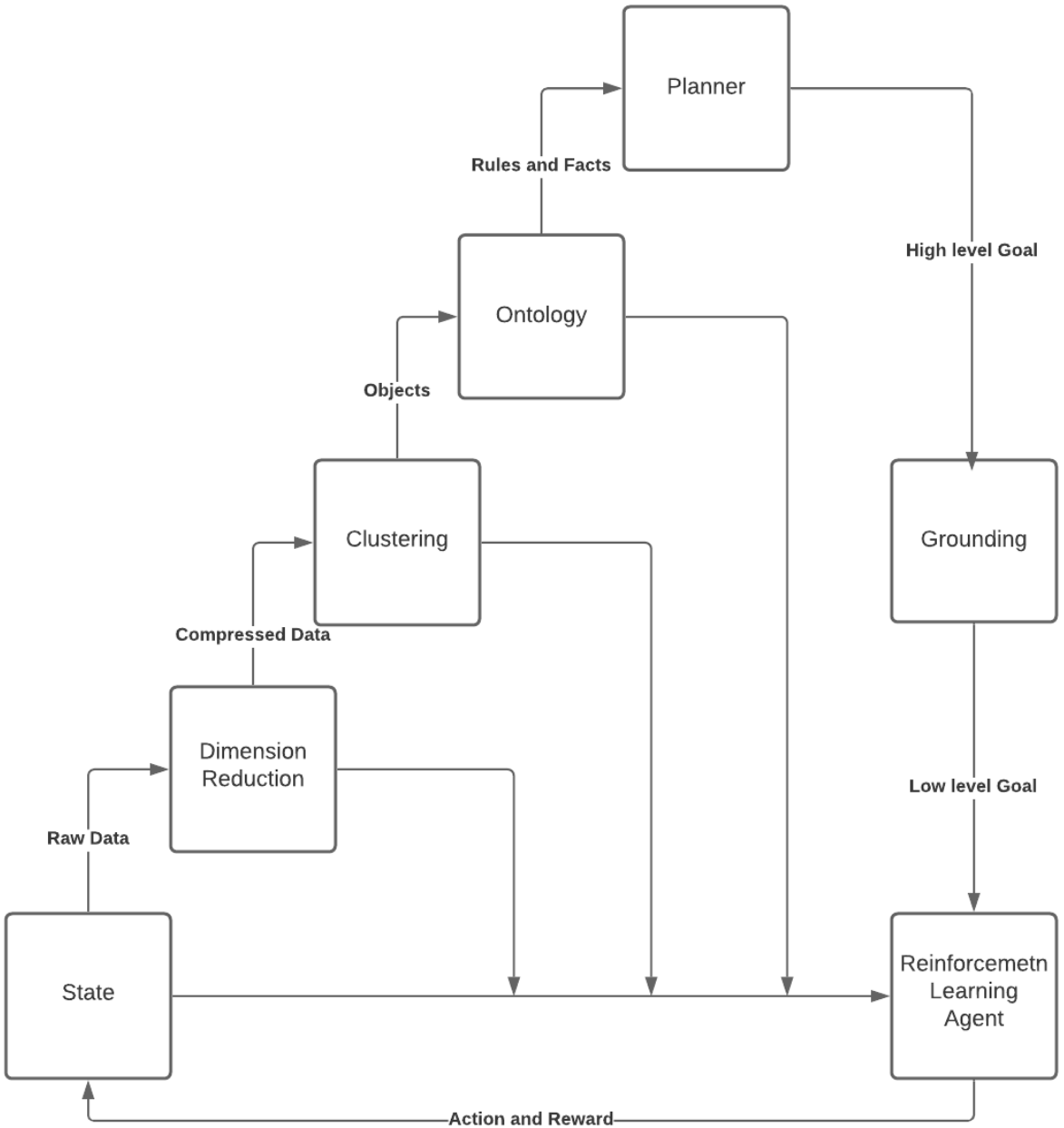
Exploration of the usage of semantic reasoning in reinforcement learning

Olaf Werner
Supervisor: Maria Ganzha

Overall Architecture



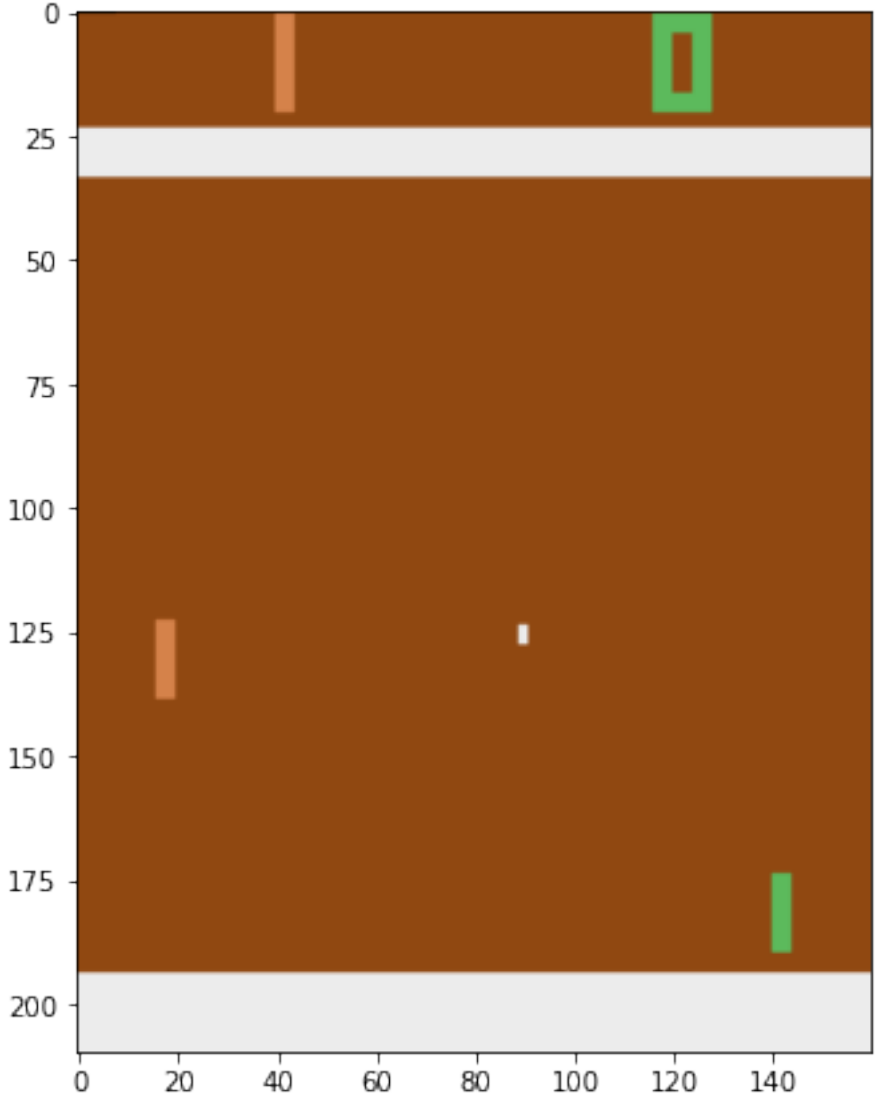
Total algorithm summary

- Rescale and grayscale image
- Find objects using otsu thresholding
- Compress objects with PCA
- Cluster objects with birch
- Transform objects into graph
- Transform graph into embedding with graph2vec
- Use Reasoner to find best path on State Map
- Take action using Q-learning Neural Network using embedding and proposed best path

State

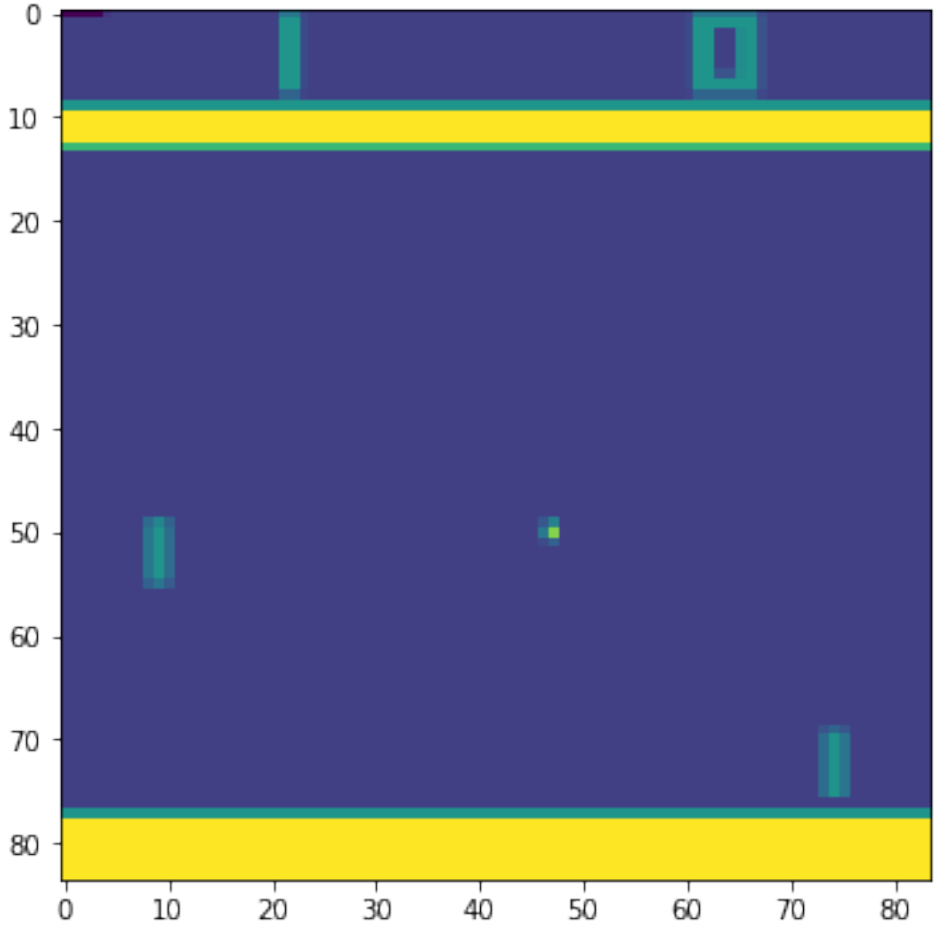
- We test architecture on Pong game. We use "PongNoFrameskip-v4" environment from OpenAI Gym.
- States are 210X160X3 RGB images or 128 bytes RAM state, but from RAM we extract ball and paddles positions directly.
- Rewards are +1 or -1 for scoring points when ball touches left or right side of the screen.
- There are 2 actions (Move paddle up and move paddle down).

Example Pong State (100800)



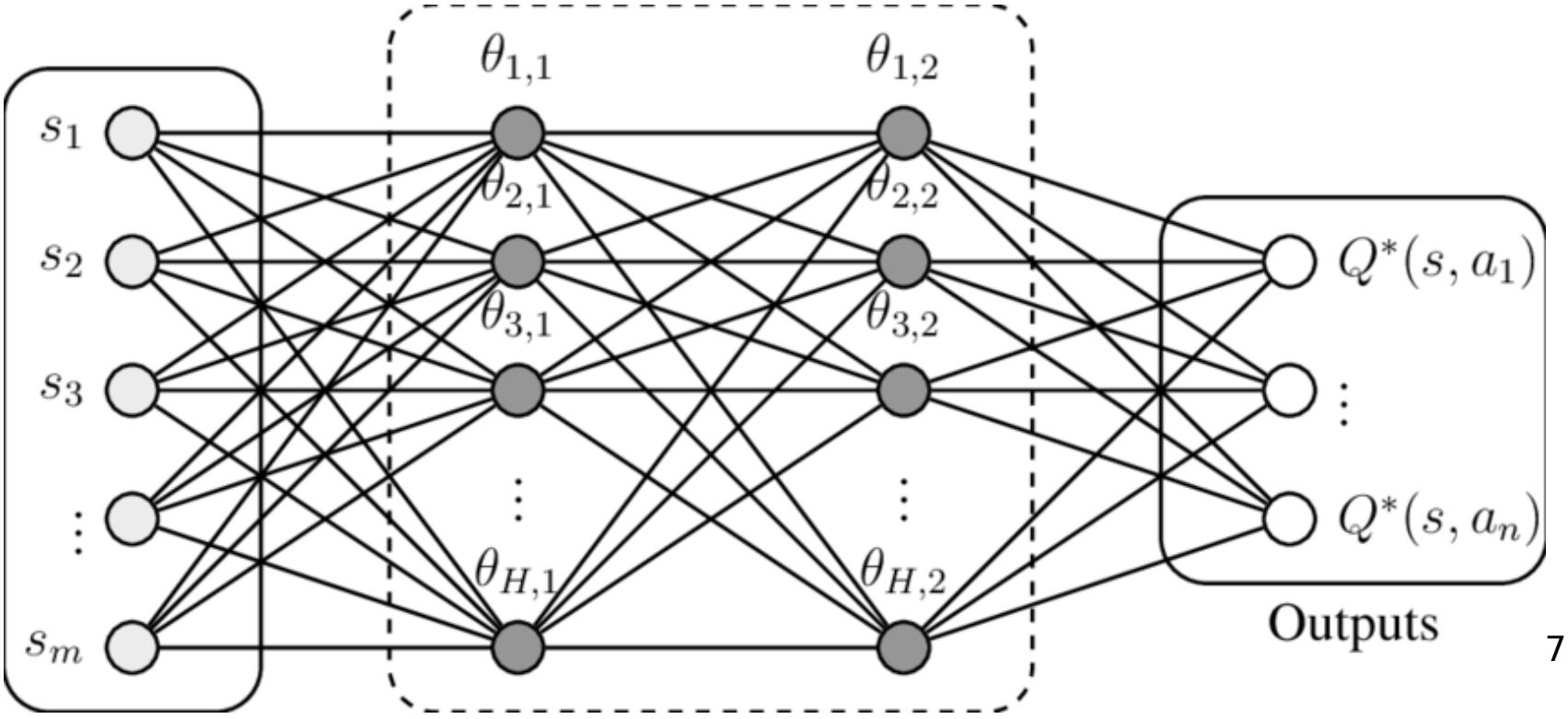
Basic State (7056)

We rescale the image to 84x84 by using inter-area interpolation and grayscale.



Reinforcement Learning

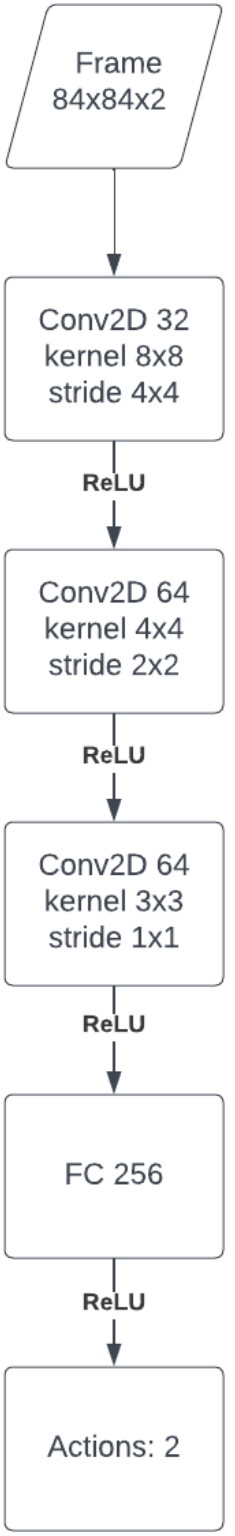
Our policy can be deep neural network. Network takes the state as input and gives Q-values as an output. To train it, we create an experience replay. In it we store the agent experiences at each timestep pooled over many episodes into a replay memory. To update it we randomly sample some experience to form a minibatch. Then we perform Q-learning updates and backpropagate changes in the deep-Q neural network.



Reinforcement Learning

We use Torch implementation of the Deep-Q learning. All activation functions are ReLU. Layers are following:

- The input Layer consists of an 84x84x2 image. The first channel represents the current state and the second channel difference between the current and previous state.
- The first convolutional layer has 32 channels with kernel size 8 and stride 4.
- The second convolutional layer has 64 channels with kernel size 8 and stride 4.
- The third convolutional layer has 64 channels with kernel size 3 and stride 1.
- The fourth layer is a dense layer made of 256 neurons.
- The last output layer represents possible actions in our case UP and DOWN.



Otsu’s thresholding

First we use Otsu's method of image thresholding. It separates images into foreground and background by minimizing weighted sum of variances of the two classes

$$\sigma_w^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t)$$

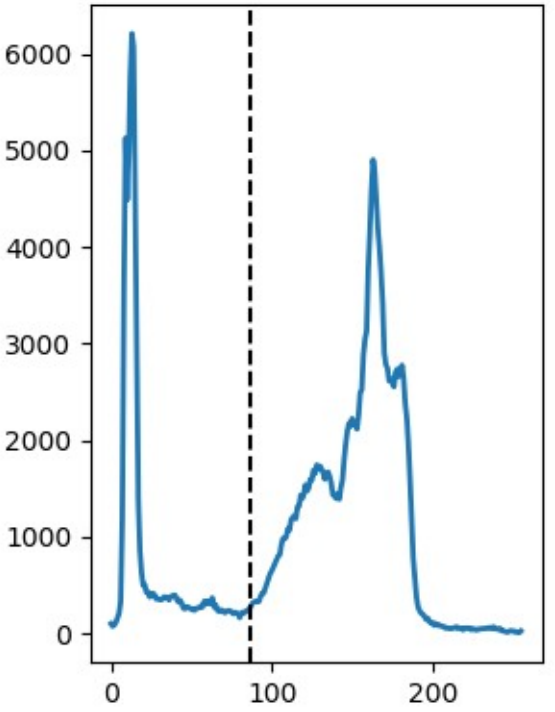
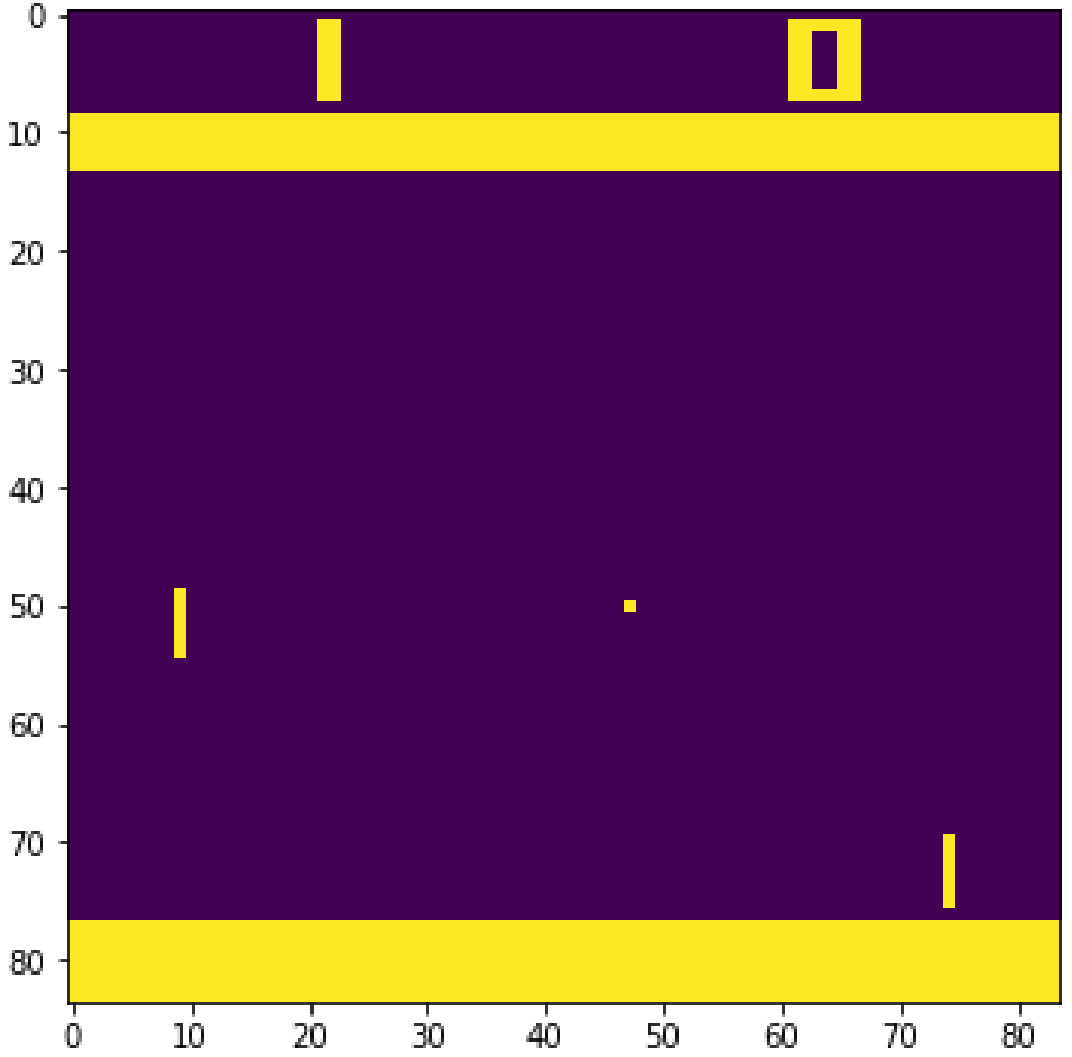
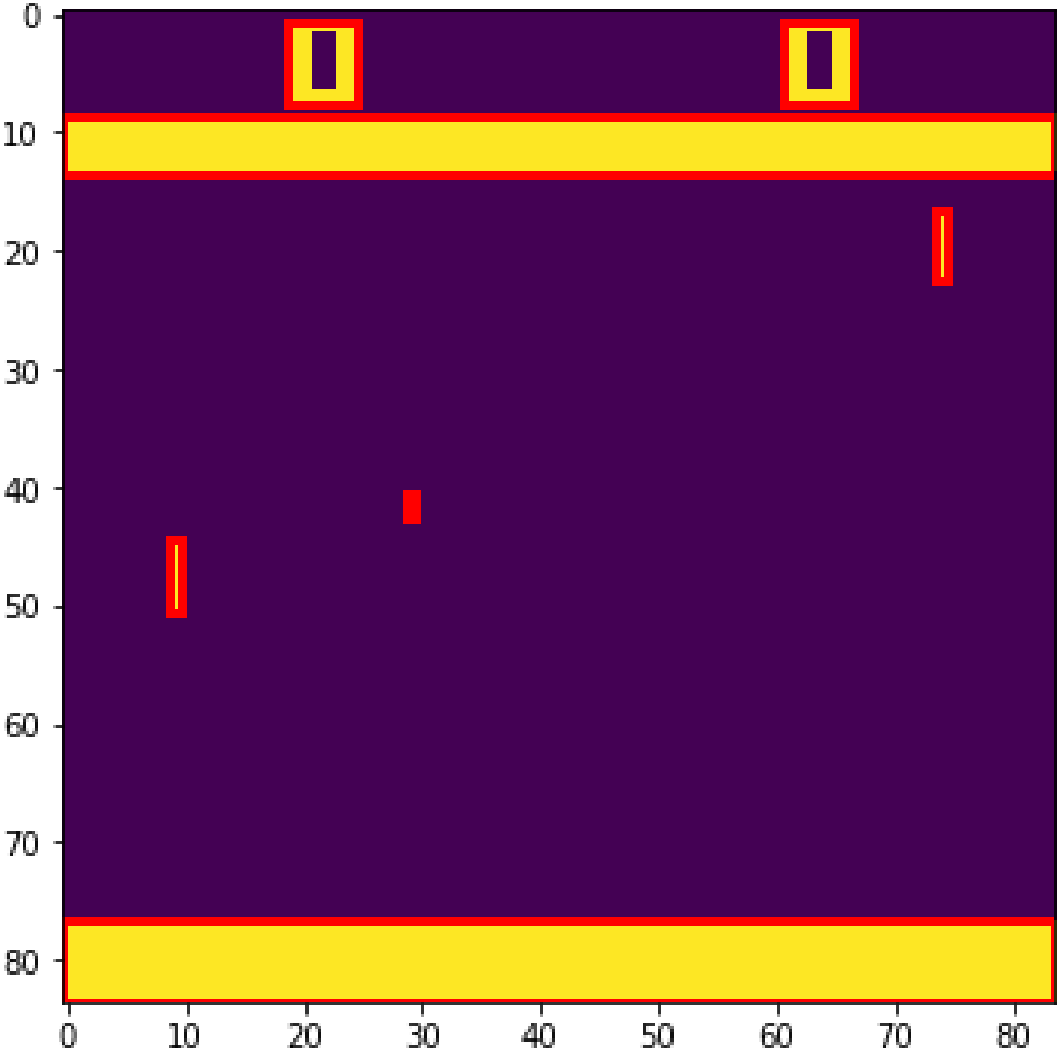


Image after thresholding



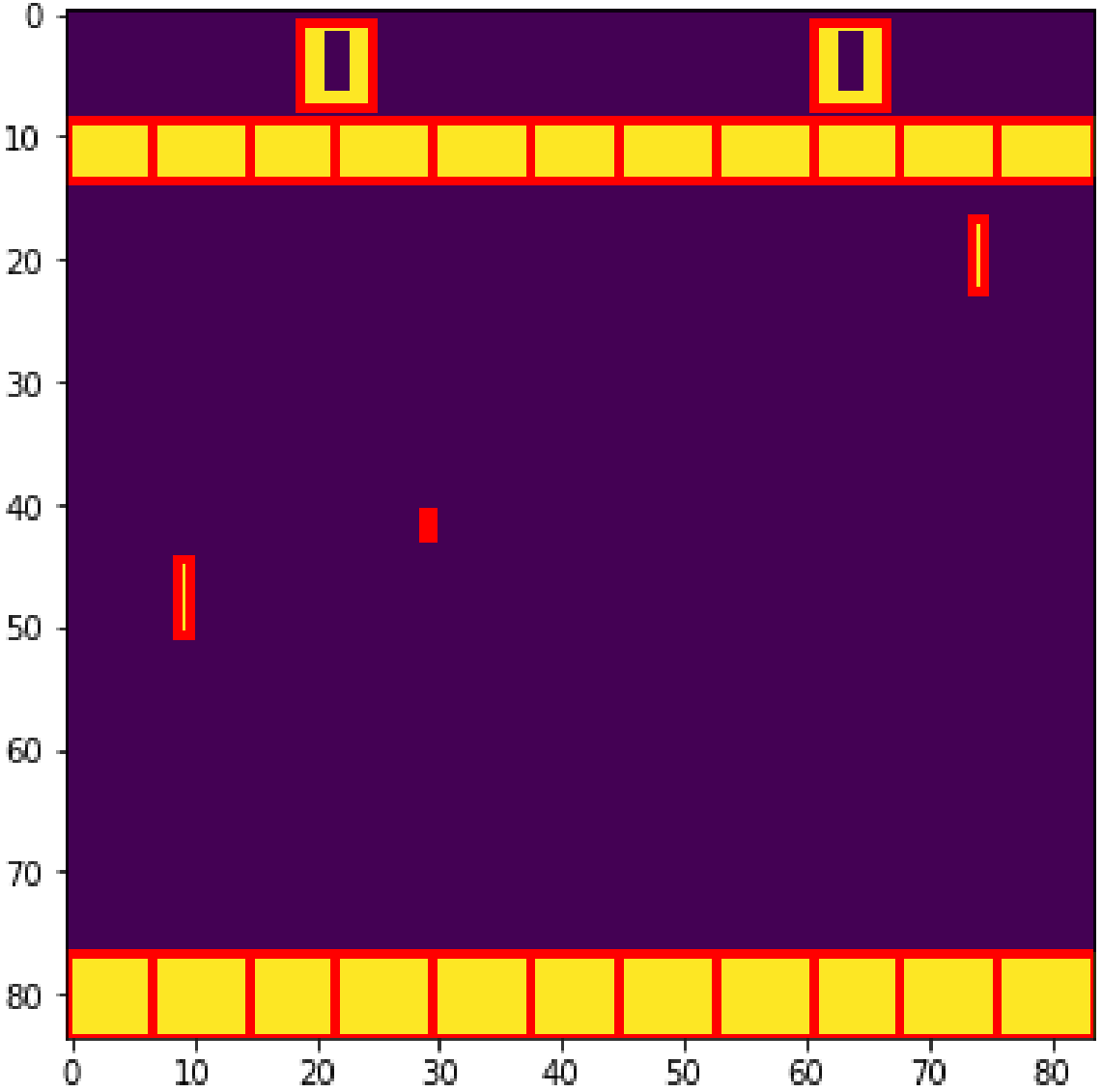
Finding Objects

We take connected components which are groups of pixels that belong to the foreground and are neighbors of each other. We treat them as a singular objects.

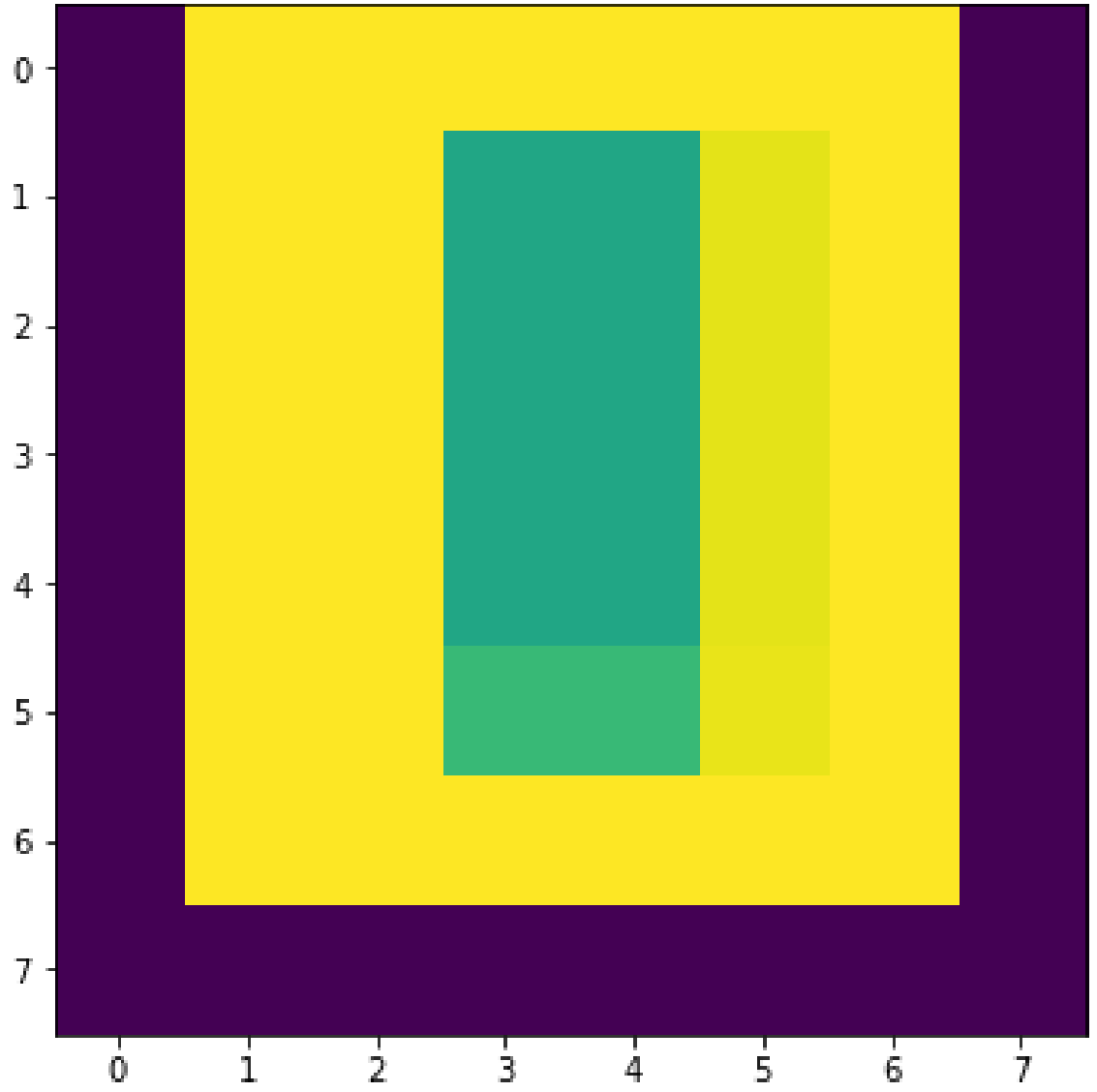


Cutting too big objects

Some of the objects are very large. We cut objects in the following way: we first define the maximum space an object can take. We use one-tenth of the image both horizontally and vertically. If the object is bigger than that we cut it into as few areas as possible so each of the objects created is of the same size and within bounds predefined earlier. If we need to cut an object both horizontally and vertically then we first cut it horizontally then we cut new objects vertically if needed.

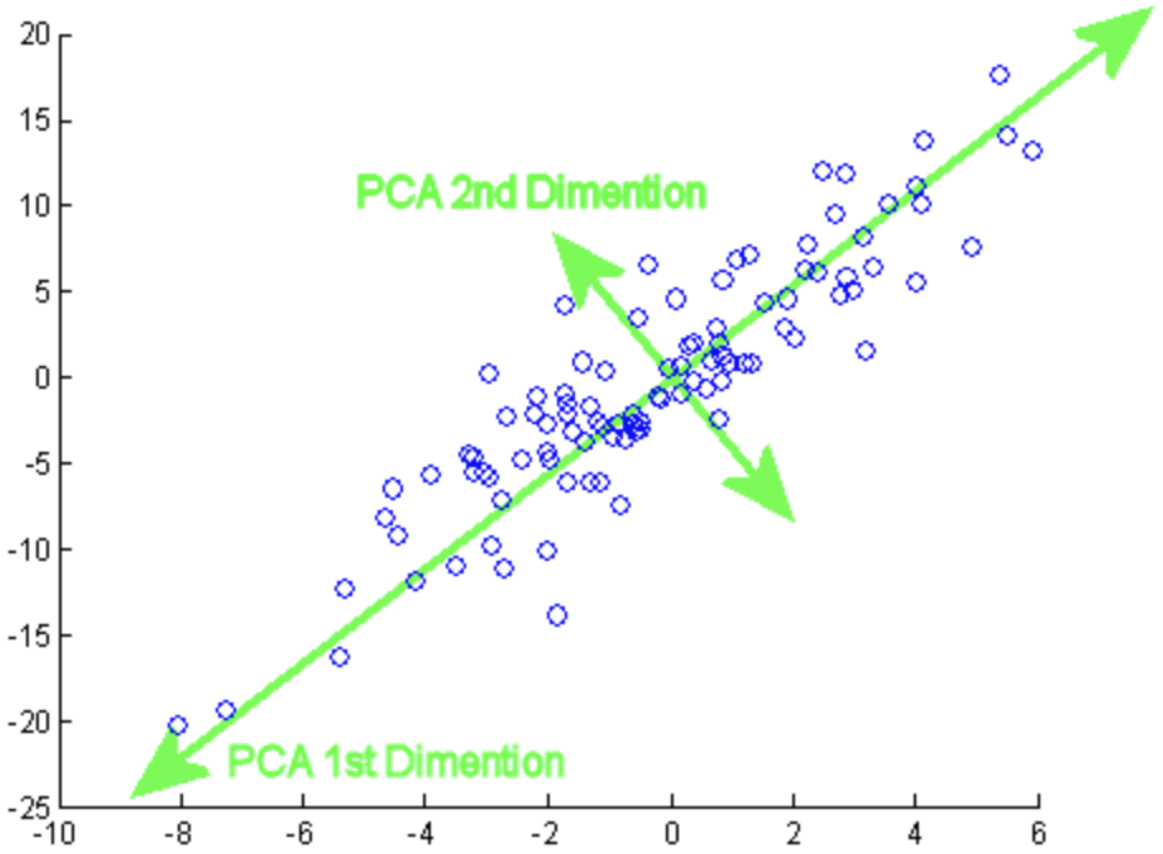


Example object (64)



Incremental PCA

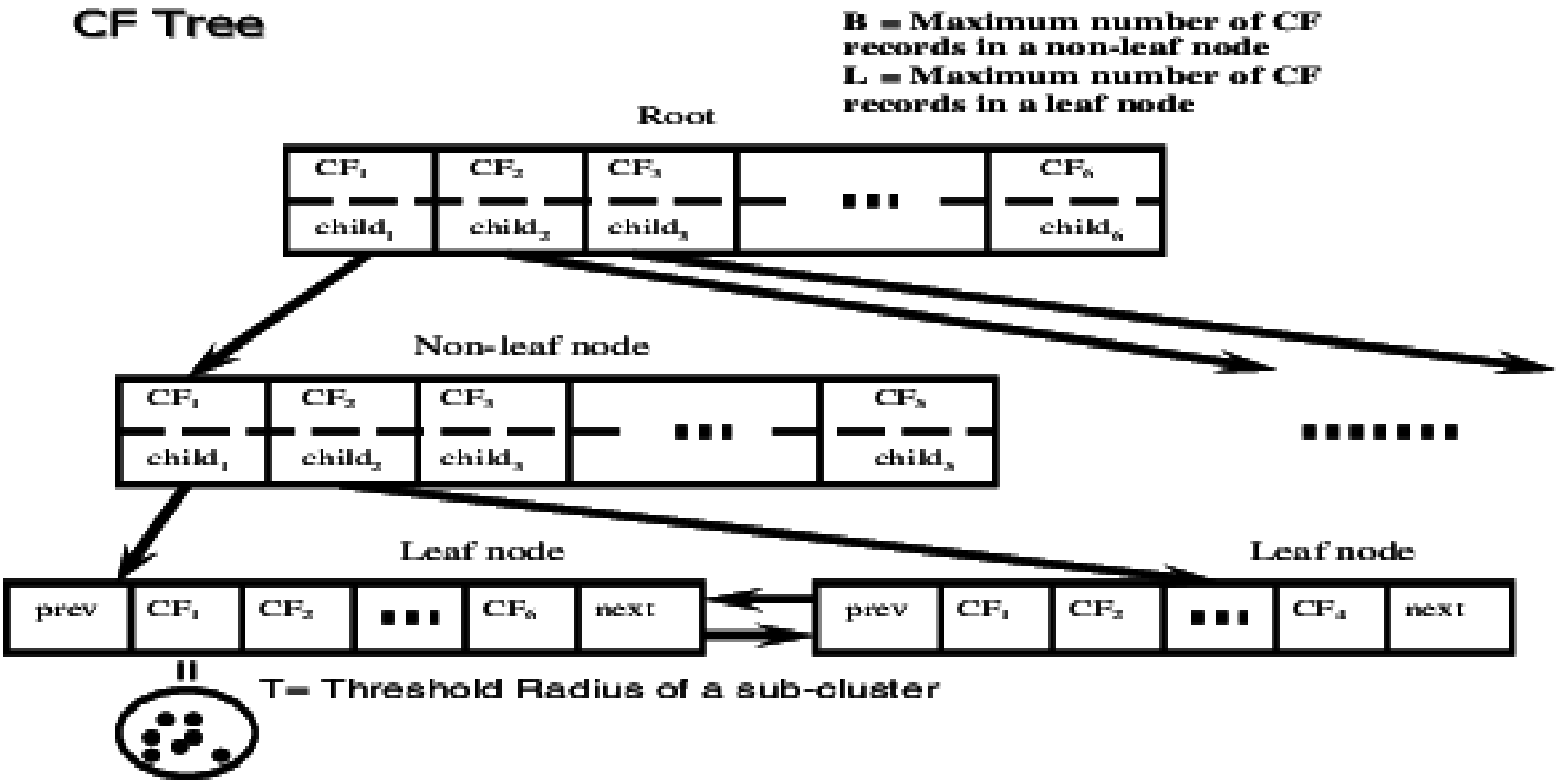
We use incremental version of PCA to lower dimensions of the objects to 10.



Birch

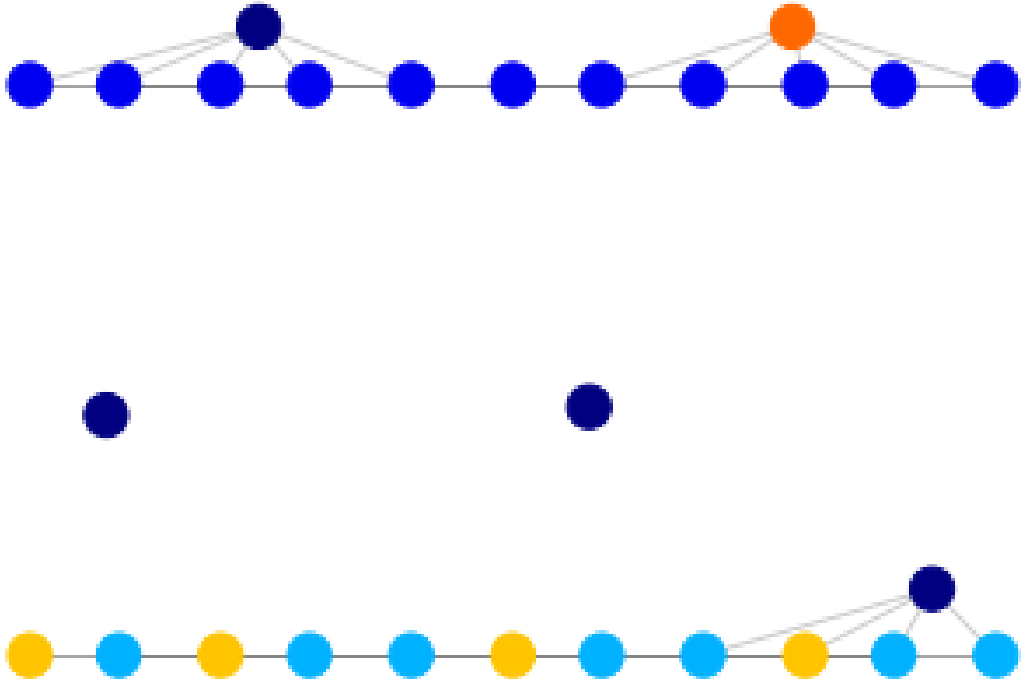
- It constructs a tree data structure with the cluster centroids being leafs. The structure is made of nodes with each node having many subclusters. The branching factor is the maximum number of subclusters in a node. Each subcluster maintains a linear sum, squared sum, and the number of samples in that subcluster. Subcluster can also have another node as its child, unless the subcluster is a leaf node.
- When a new point enters the root, it is merged with the subcluster closest to it and the linear sum, squared sum and the number of samples of that subcluster are updated. This is done recursively till the leaf node are updated.
- A leaf nodes can be either the final cluster centroids or can be further clustered as input for another clustering algorithm.

Birch


CF Tree
B = Maximum number of CF records in a non-leaf node
L = Maximum number of CF records in a leaf node
Root
CF1
CF2
CF3
CF6
child1
child2
child3
child6
Non-leaf node
CF1
CF2
CF3
CF5
child1
child2
child3
child5
Leaf node
Leaf node
prev
CF1
CF2
CF6
next
prev
CF1
CF2
CF4
next
T= Threshold Radius of a sub-cluster

Graph Form (~702)

On this Graph every node is an object and its color symbolizes type of the object. Edges are between nodes when they are close enough and symbolize distances between them.

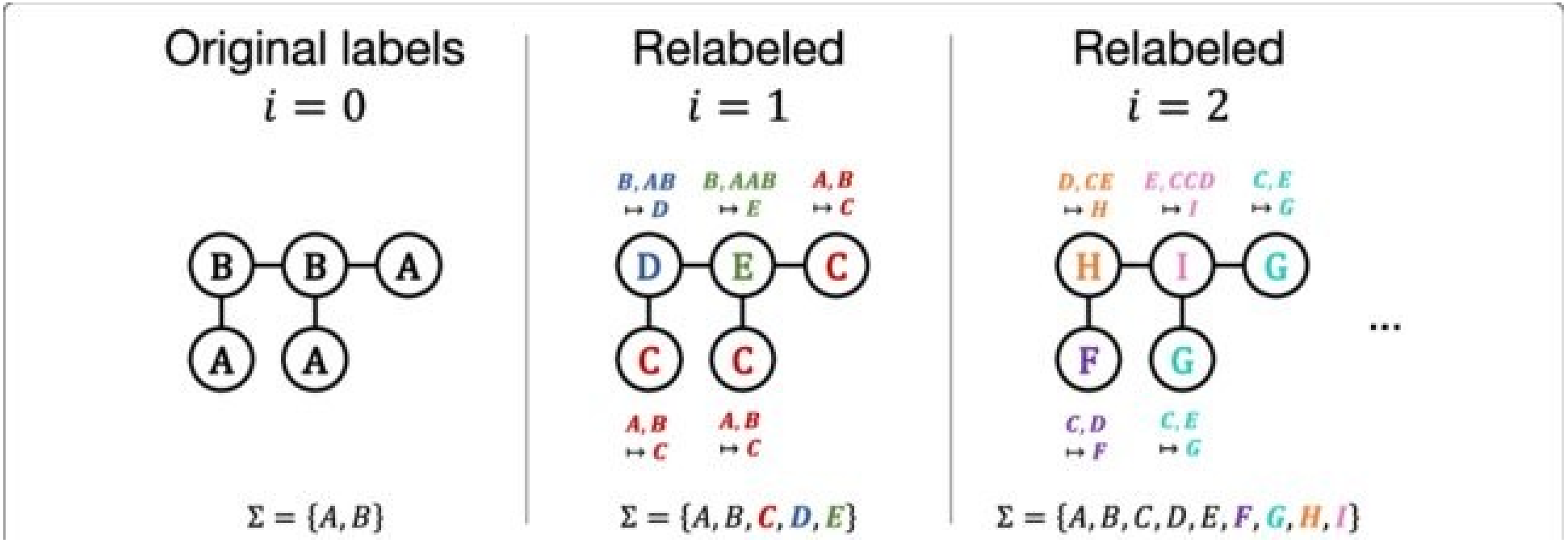


Graph2Vec

Graph2Vec is an graph embedding method which is combination of two methods: Weisfeiler Lehman Hashing which transforms nodes into words and doc2vec which is document embedding method.

Weisfeiler Lehman Hashing

We use Weisfeiler Lehman Hashing to transform Graphs into the sentences with each hash being one word.



Sentence creation

We also use other information compared to the original graph2vec. Each node is a sentence containing: its object class, x position, approximate x position (x divided by 5), approximate y position (y divided by 5), and Weisfeiler-Lehman hashes. For the distributed bag of words (PV-DBOW) we also add information regarding embedding concatenated with its class label, x position, and y position, it is because the bag of words does not look at the order of the words so we do this concatenation to differentiate between the position of the objects depending on their class, and in case of multiple objects of the same kind their mirror positions.

Example document

'8', 'pos_x4', 'pos_y21', 'apr_pos_x0', 'apr_pos_y4', '8pos_x4', '8pos_y21', '8apr_pos_x0', '8apr_pos_y4', '8apr_pos_x0apr_pos_y4', '8pos_x48pos_y21', 'apr_pos_x0apr_pos_y4', 'pos_x48pos_y21', 'dab5edacccee84e7f5d123489f3e151c', '4975d4c3aeda67b432297ab45ea7e3f8'

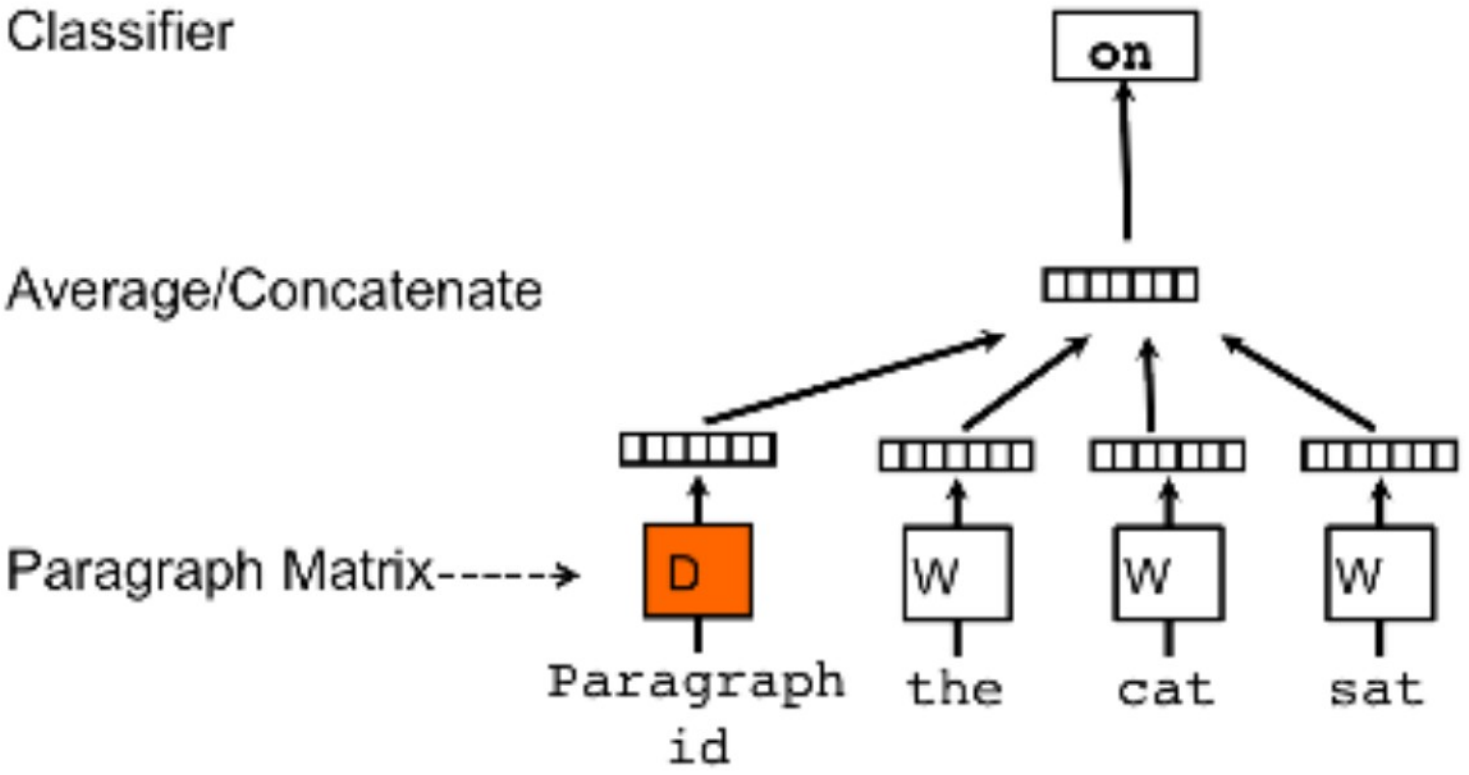
Word2Vec

- Doc2Vec is an modified word2vec with added added paragraph vector
- Every word is mapped to a unique vector, represented by a column in a matrix ***W***. The column is indexed by position of the word in the vocabulary. The concatenation or sum of the vectors is then used as features for prediction of the next word in a sentence.
- The goal is to maximize the average log probability

$$\frac{1}{T}\sum_{t=k}^{T-k}\log p(w_t|w_{t-k},...,w_{t+k})$$

Doc2Vec (128)

Then we use Doc2Vec to embed our sentences into 128 dimensional vectors.



Doc2Vec training

- We first train PCA and Birch till they reach stability (we compare old predictions to new predictions on given sample after 200 additional batches of training).
- Then we gather 30000 frames/documents and train our model on this corpus
- We do this because gensim implementation of doc2vec DOES NOT support online learning.

Ontology

For our ontology and reasoning, we will be using the graph. While calling our graph ontology may be a stretch it holds information regarding state dynamics and we could represent this graph as a collection of RDF triples.

Ontology will be made out of following elements:

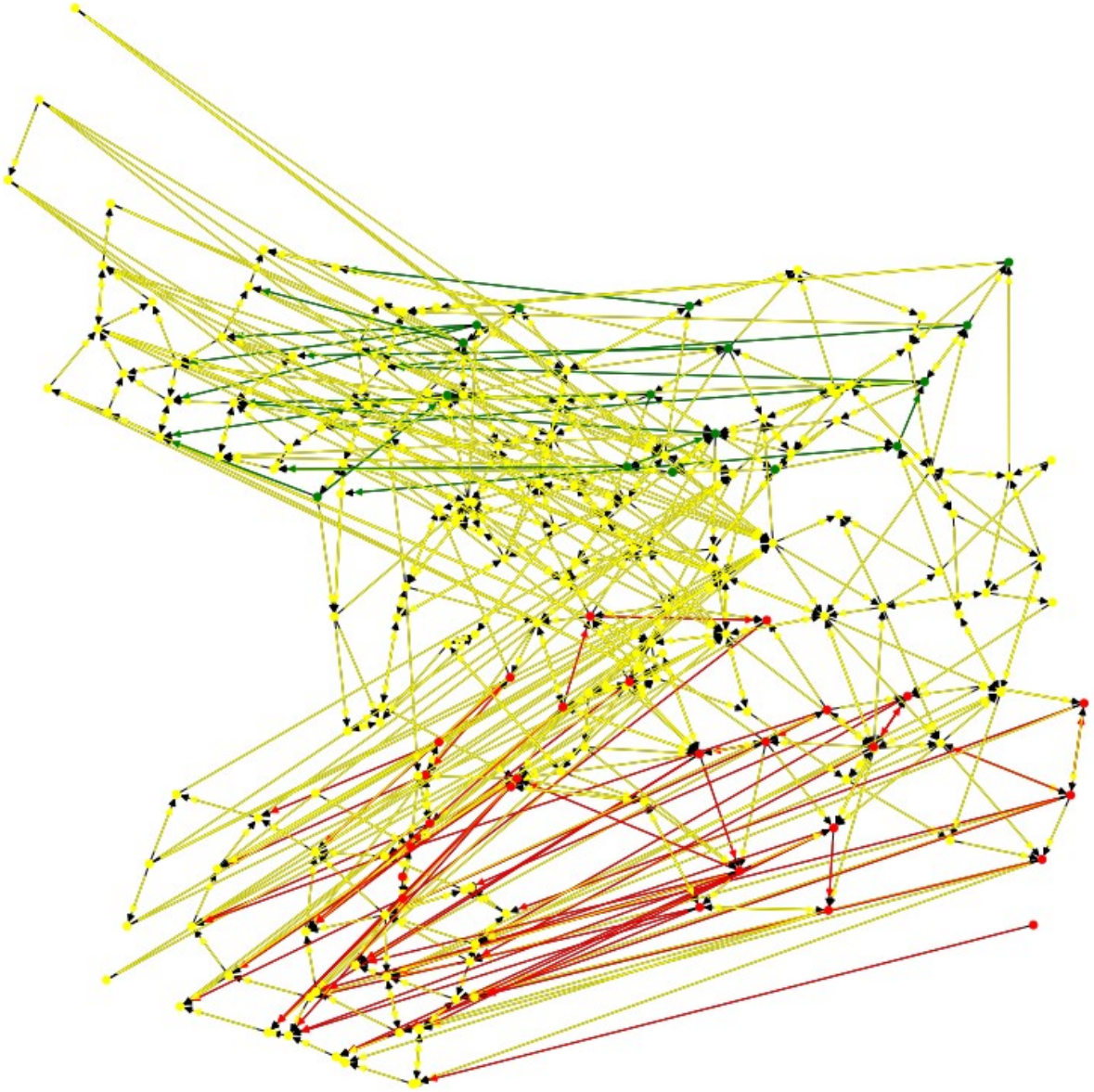
- Node birch tree for storing nodes
- Edge birch tree for storing nodes
- State prediction neural network
- Reward prediction SVM

State graph creation

We have two birch trees one for nodes and one for edges

- Node tree is clustering embeddings and this clusters are nodes.
- Edge tree is clustering transitions between embeddings and this clusters are edges.
- We create graph by classifying start and end embeddings of edge tree clusters using node tree

Example State graph

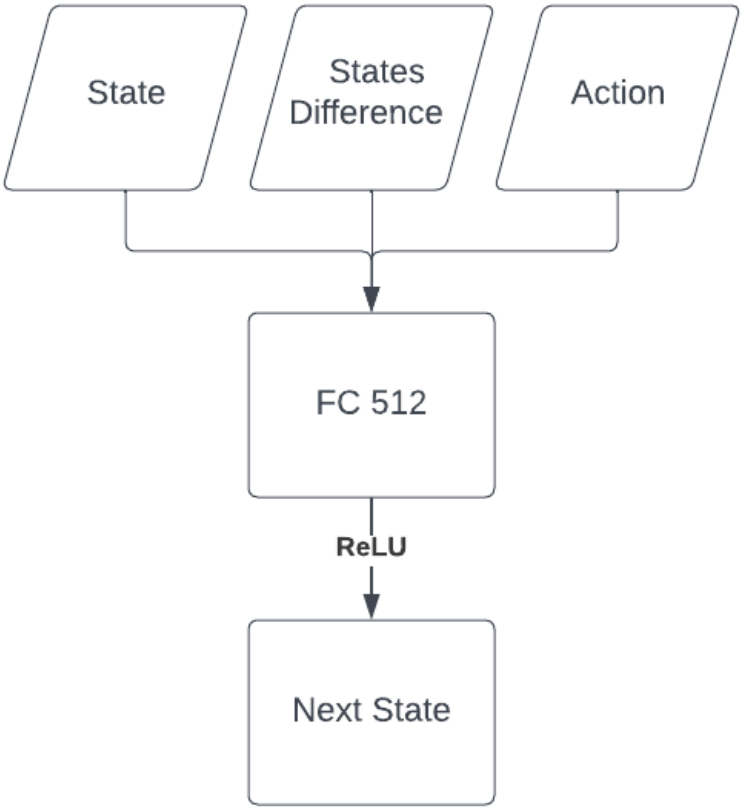


Reasoning

Because centroids are not "real" states which occurred edges in reality can take several actions to cross. Therefore we also train a models for predicting the next state and reward given the current state and chosen action.

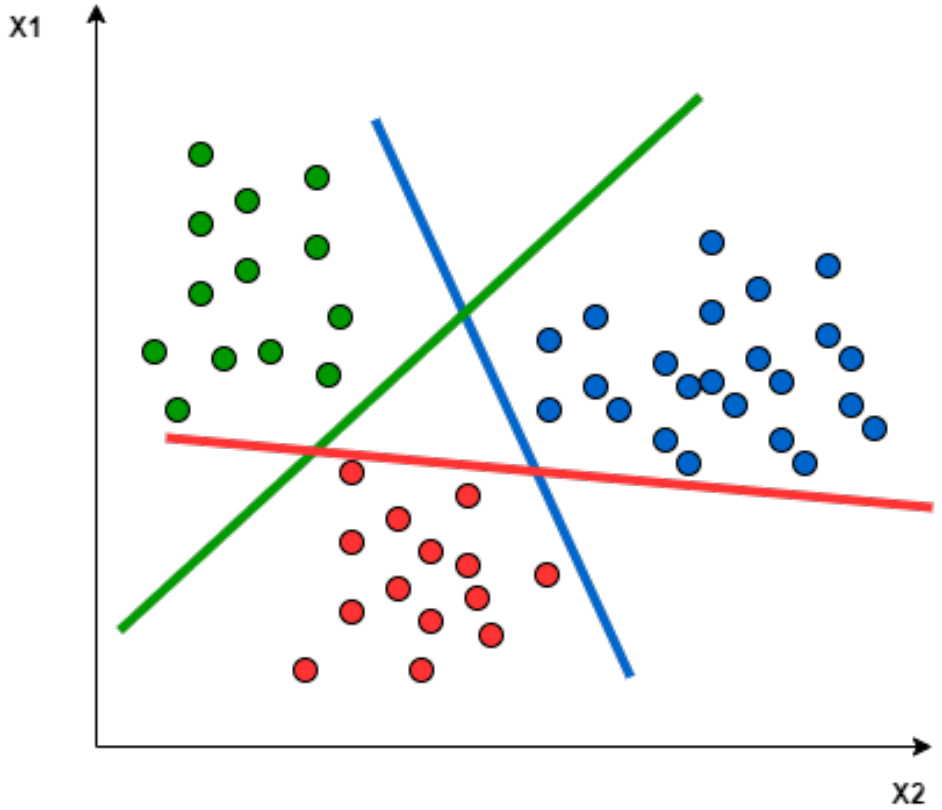
State prediction neural network

This neural network takes an state, difference between current state and previous state and action. It has one fully connected layer with 512 neurons. As an output it gives prediction of the next state.



Reward prediction SVM

This task is classification problem with classes being positive, negative, and neutral rewards. Inputs are: state, the difference between the current state and previous state, and action. Because this is a multiclass problem we will be using One Versus All approach.



Reasoning: long term

We are calculating the path following which we will get the greatest reward. This problem can be seen as the shortest path problem with weighted edges. Positive rewards are negative weights and vice versa. The existence of negative weights introduces a few problems. First is that Dijkstra algorithm for finding shortest paths does not work when we have negative edges. Other algorithms like Bellman-Ford can work with negative edges, however they still fail when there are negative cycles in the graph.

Reasoning: long term

To handle these issues, we will be setting neutral rewards to 1, positive rewards to 0, and negative rewards to 100, and instead of finding the overall best path, we will be finding the closest states with positive rewards. Our algorithm for finding this path will be Dijkstra. We will be using a cutoff equal to 100 so paths longer than 100 will be ignored. In this way we will be avoiding negative rewards.

Reasoning: short term

When we can predict both states and rewards we can implement short-term planning. It works like this: from starting state generate all possible action sequences with a given depth. Depth is the length of these sequences. Then for each sequence predict the end state and total reward, using predictions of the previous states.

Reasoning Summary

Complete reasoning procedure is as follows:

- Assign cluster to current state using node clustering Birch tree.
- Use the Djikstra algorithm to find the shortest path to the state which has a positive reward edge adjacent to it.
- Generate scenarios using state and reward prediction models with chosen depth.
- Sort scenarios with the following priorities: reward, furthest node on the proposed path, euclidean distance to the final node centroid.
- For grounding, return information about the best scenario. This information is the proposed action sequence, predicted state, and reward while following this sequence.

Results: PCA

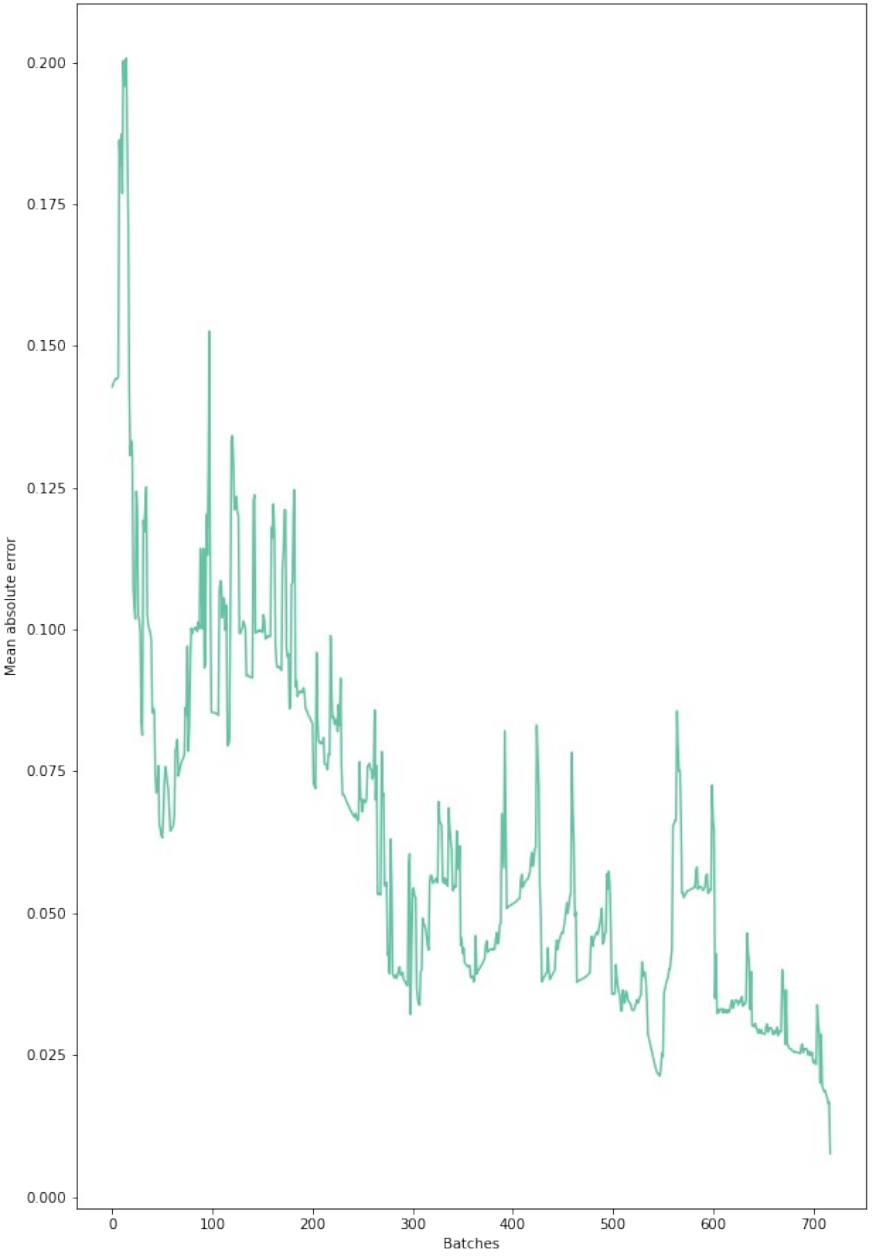
We will test our embedding on an entirely new run. This run was random as training of Dimension Reduction and Clustering modules takes place before we train our Agent.

The new example run had 1188 timesteps and number of all detected objects was 94902.

To asses quality of PCA we will simply take percentage of explained variance. It turned out to be 99\%.

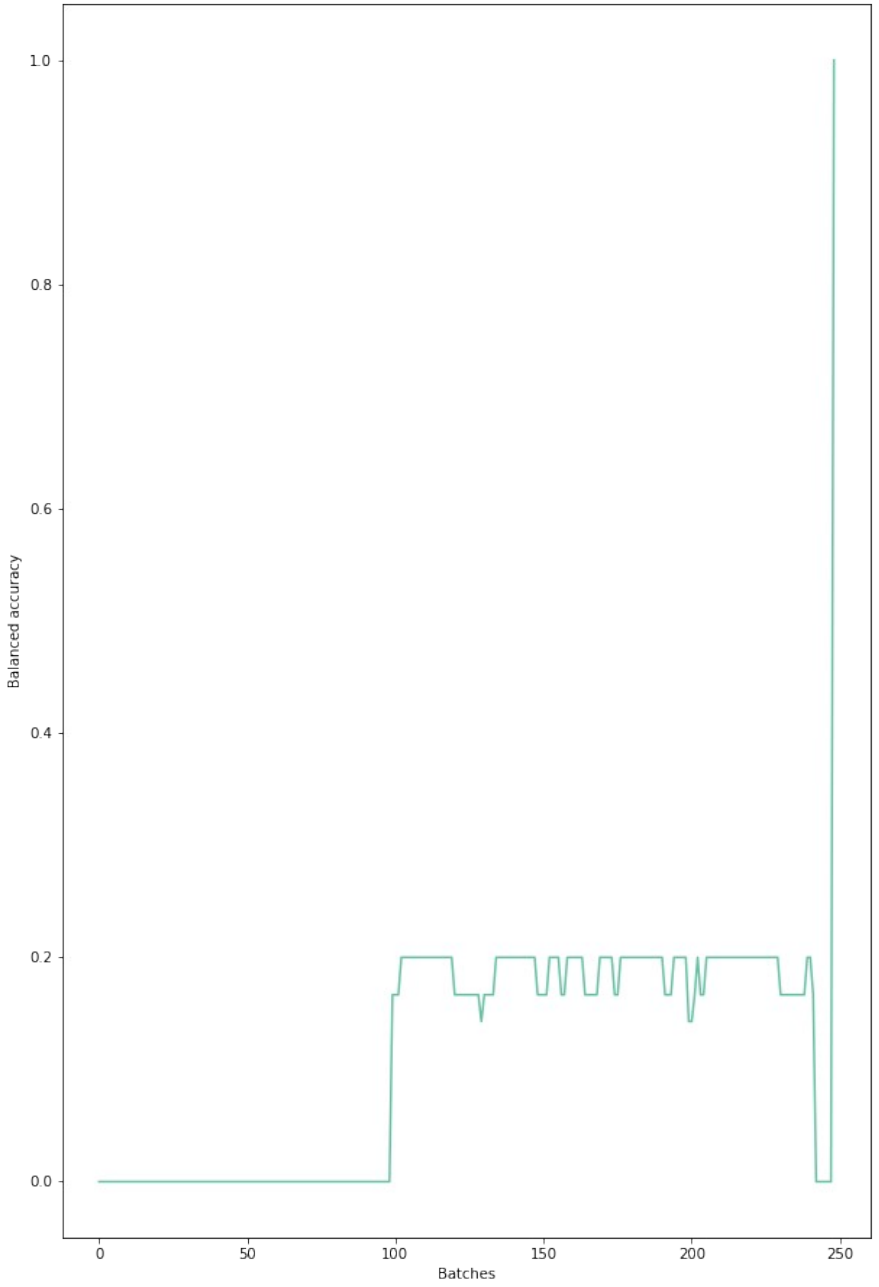
PCA stability

When training PCA we check its stability by comparing old predictions to new predictions on given sample after 200 additional batches of training. We use mean absolute error for that.

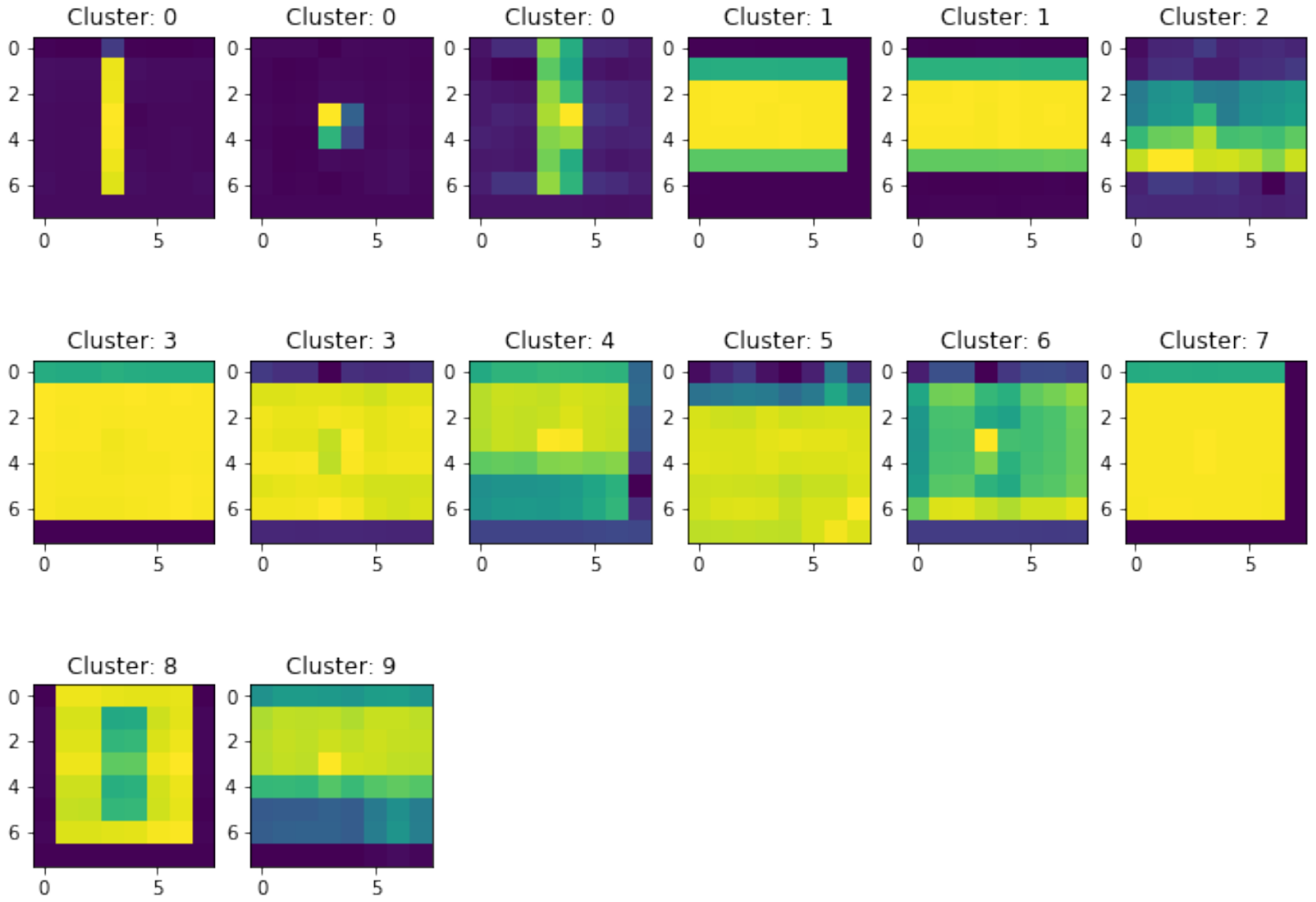


Birch stability

When training Birch we check its stability by comparing old predictions to new predictions on given sample after 200 additional batches of training. We use balanced accuracy for that.

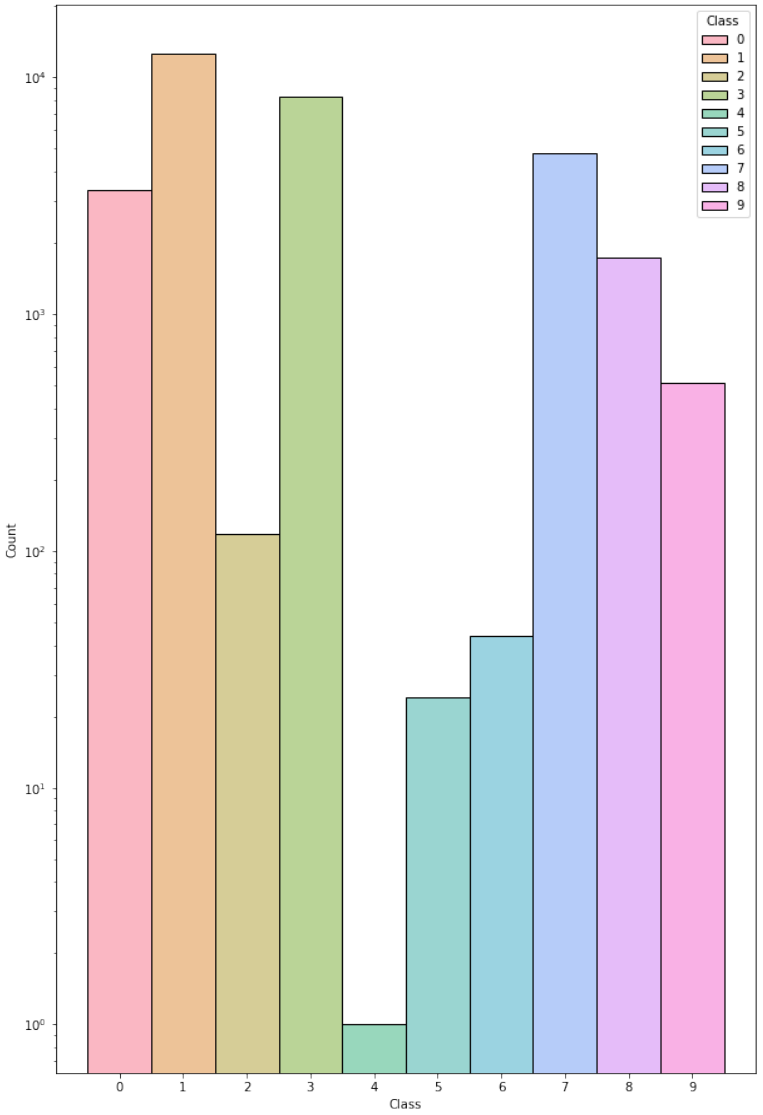


Birch found clusters



Pong object distribution

Here we can see that some classes are much rarer compared to different ones. That's because some objects are two different objects which collided.

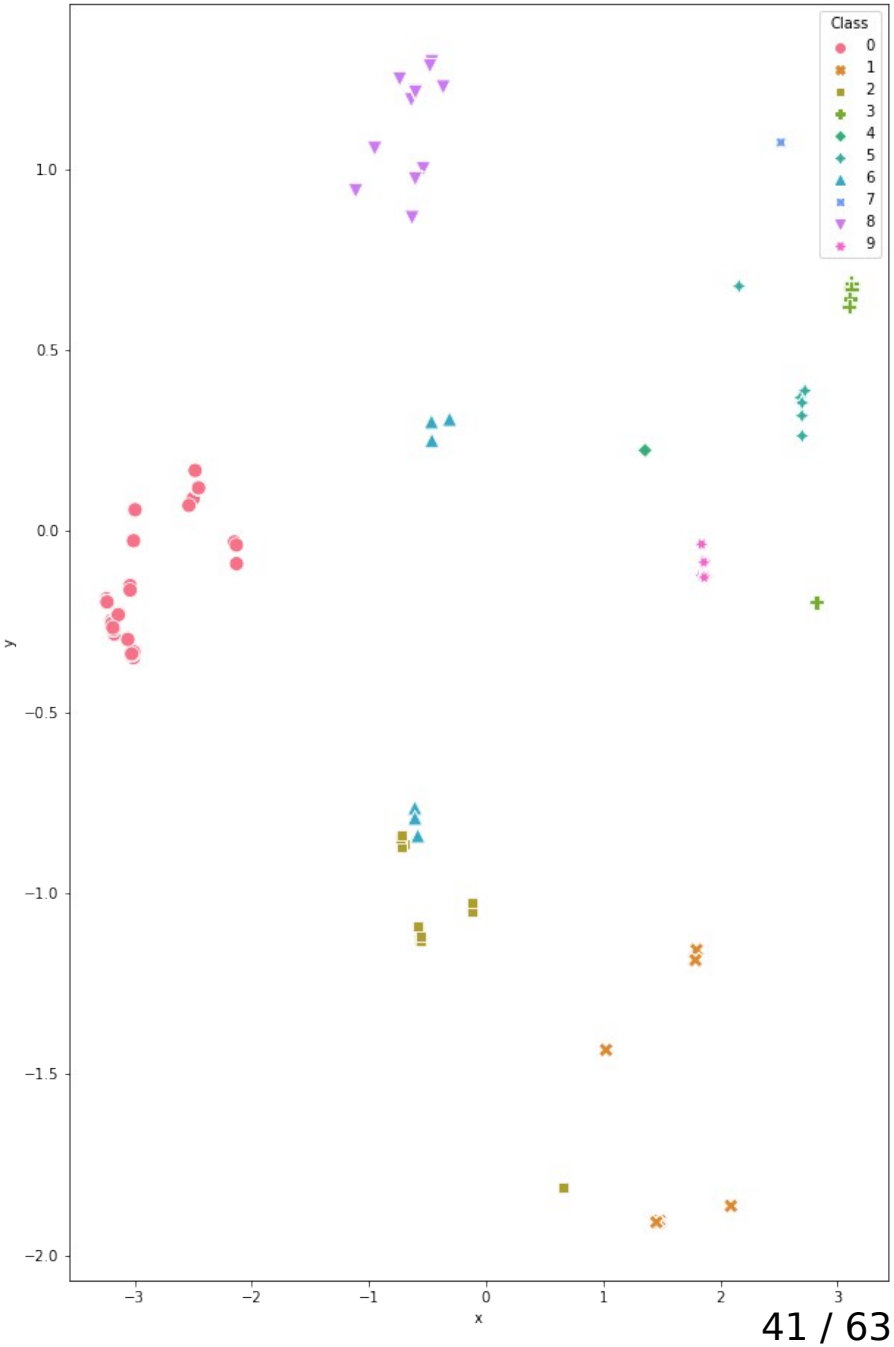


Birch metrics:

- Silhouette metric: 0.887
- Davies Bouldin: 0.528
- Dunn Index (using farthest distance for diameter, nearest distance for inter cluster distances): 0.114

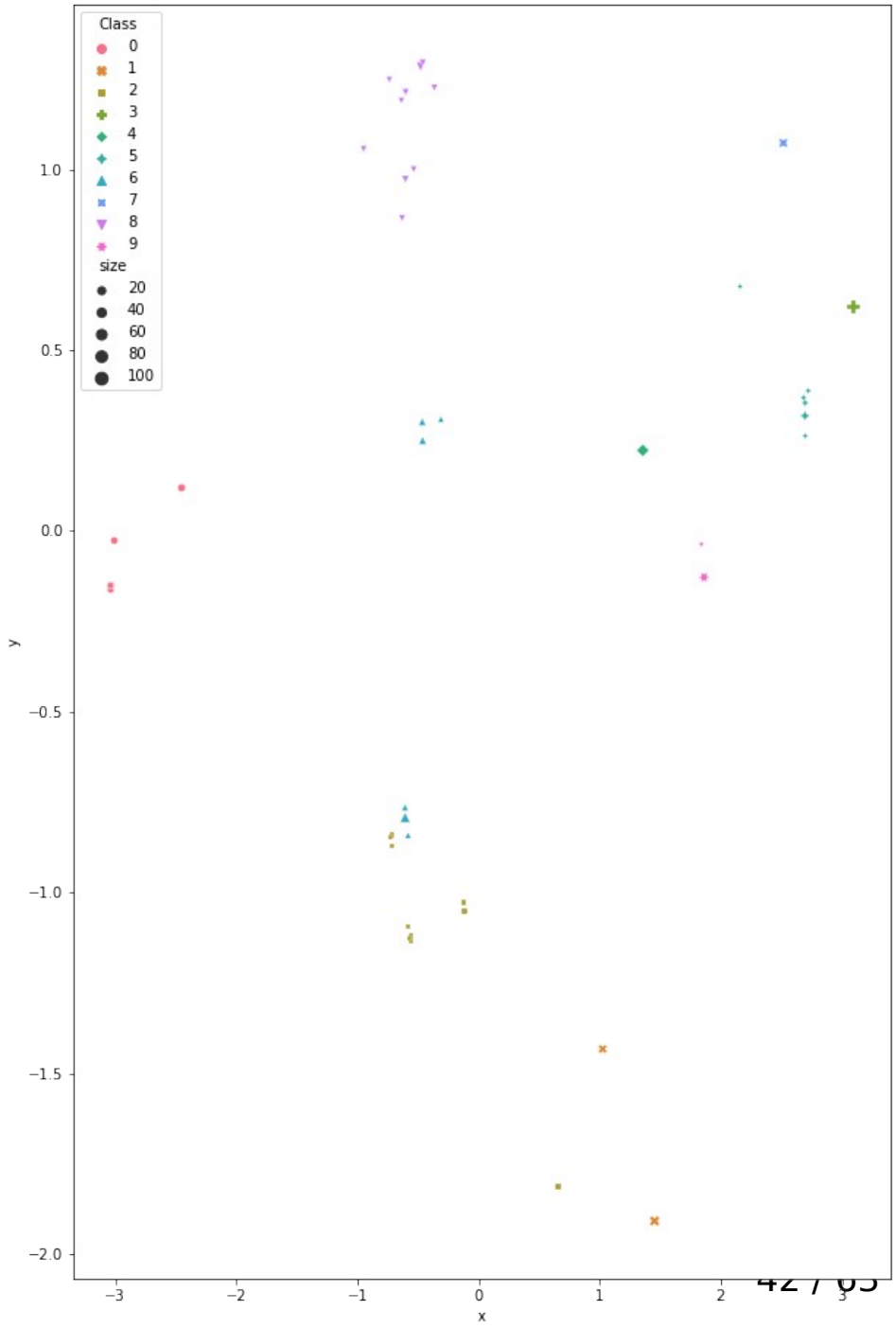
Object plot

Here we can see scatter plot with found objects. X and Y are the first and second Principal Components and together they explain 90% of the variance. We can see that some clusters are not perfect so Dunn Index is poor however Silhouette score is still good.



Object plot

To explain that let us look at the new scatter plot but it is adjusted for number of objects and overlapping points which covered less than 2% of overall cluster size got removed. Now we can see that our clusters are centralized so inner cluster distance is low so Silhouette score is very good.



Embedding

After training, our total vocabulary size is 9277 unique words. It is quite a lot for a language whose entire purpose is to describe the Pong game. On the other hand, some of the words are specific like '2apr_pos_y2' which means "paddle at approximately 15 pixel horizontally" or 'ec7a9c75e534f5a438c36cfb0c72ca7b' which means "wall surrounded by exactly two walls and nothing else".

Embedding testing

To test embedding, we first need to establish criteria for comparing two different graphs which we are going to embed. We will use the following metric:

- First, let's define the distance between two objects. If they are of the same kind then the distance between them will be the euclidean distance between their centers, if they are of two different kinds we add a class difference penalty to their distance.
- To define the distance between two graphs, we take the sum of minimums of distances from objects from one graph to objects from the other graph.
- If the number of objects is different, then we take minimums from the smaller graph and add the object difference penalty. We will use 100 for both kinds of penalties.

$$\sum_{a \in A} min_{b \in B}(dist(a,b) + class(a,b) * classPenalty) + (\|A\| - \|B\|) * numberPenalty$$

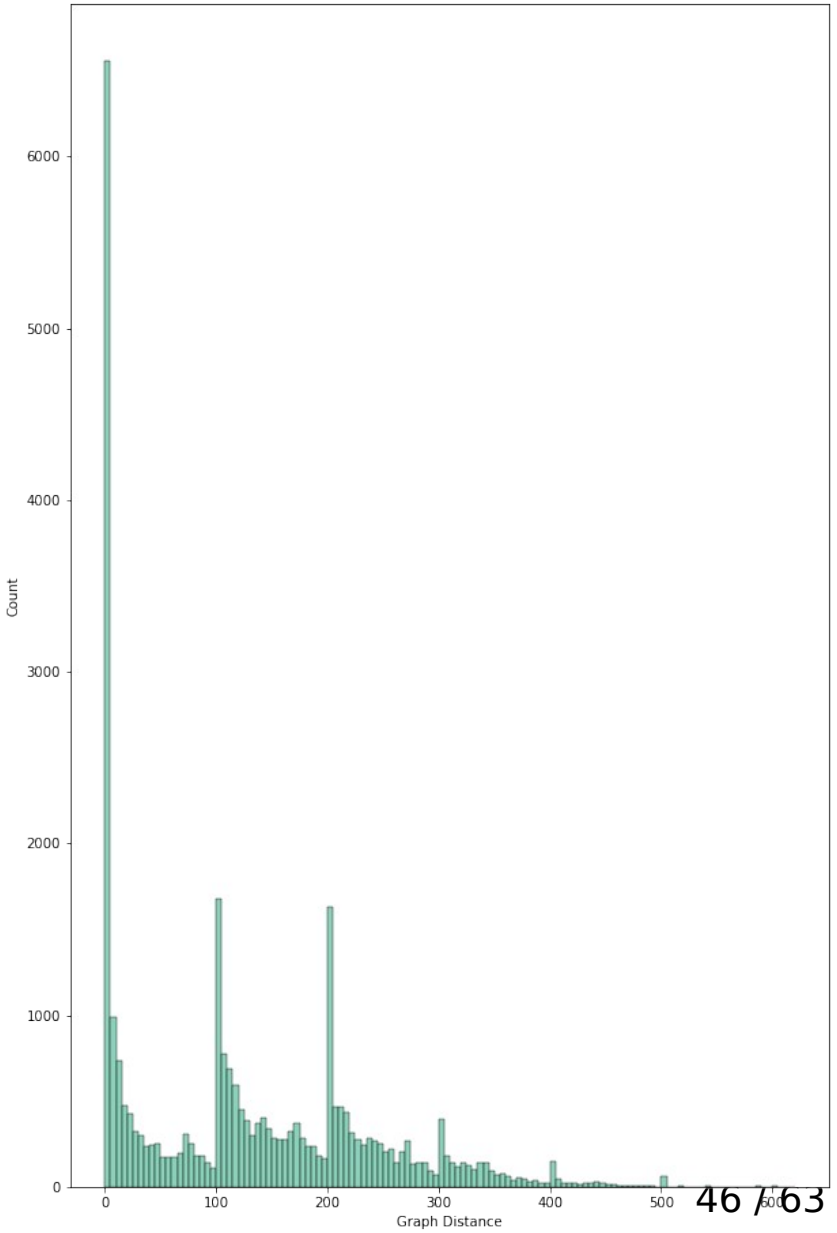
Embedding testing

For comparing embedding we will use traditional cosine similarity. This measure is used in NLP because it focuses on the words that the two documents do have in common, and the occurrence frequency of such words. This metric is also used by Gensim to find the most similar documents.

$$\text{cosine similarity} = S_C(A, B) := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

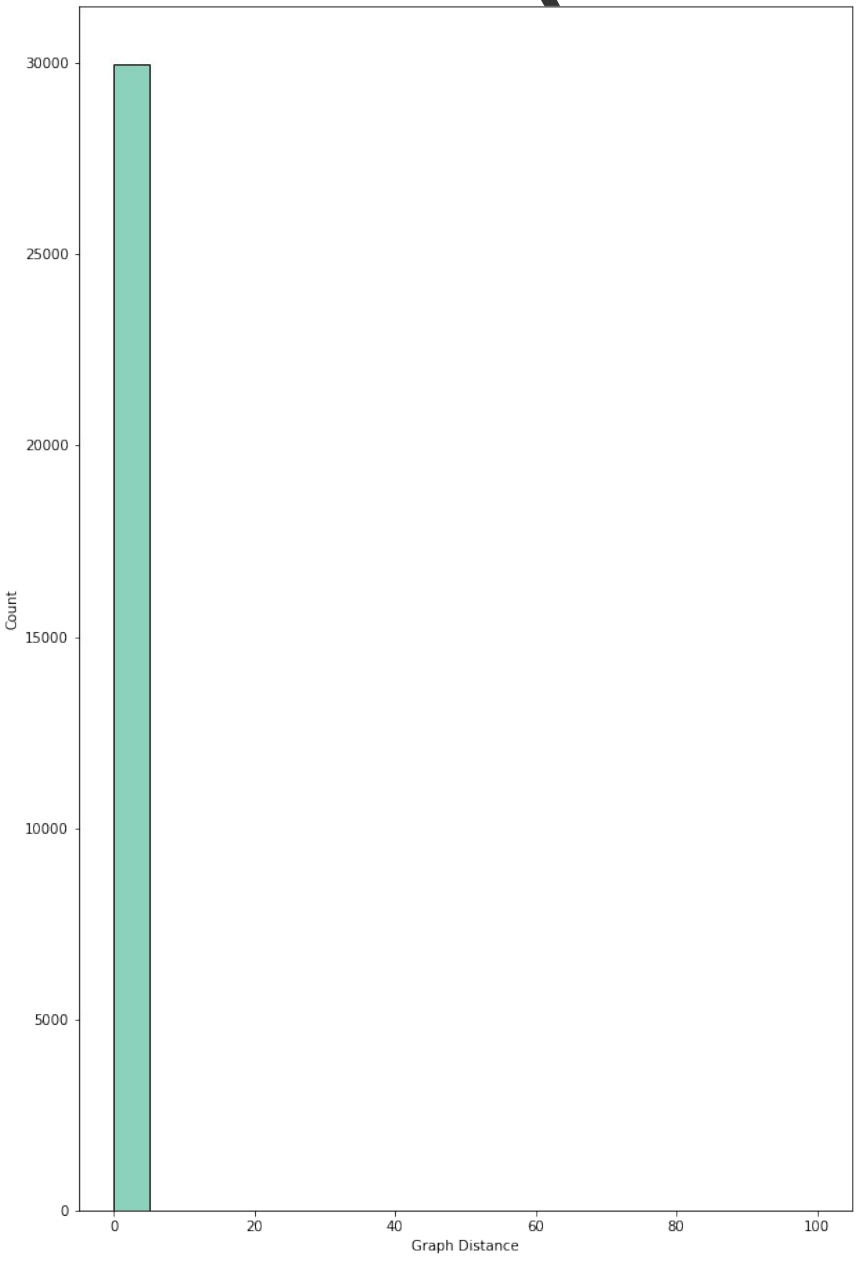
Distributed memory (PV-DM)

Distributed memory (PV-DM) achieved fatal results. The score we achieved is 0.138. However many states are very similar or outright identical, especially for starting states, so what is the accuracy if we allow the distance between to be less than 5? The answer is 0.218, which is still very bad. This algorithm failed for our case and later tests won't be performed for it.



Distributed bag of words (PV-DBOW)

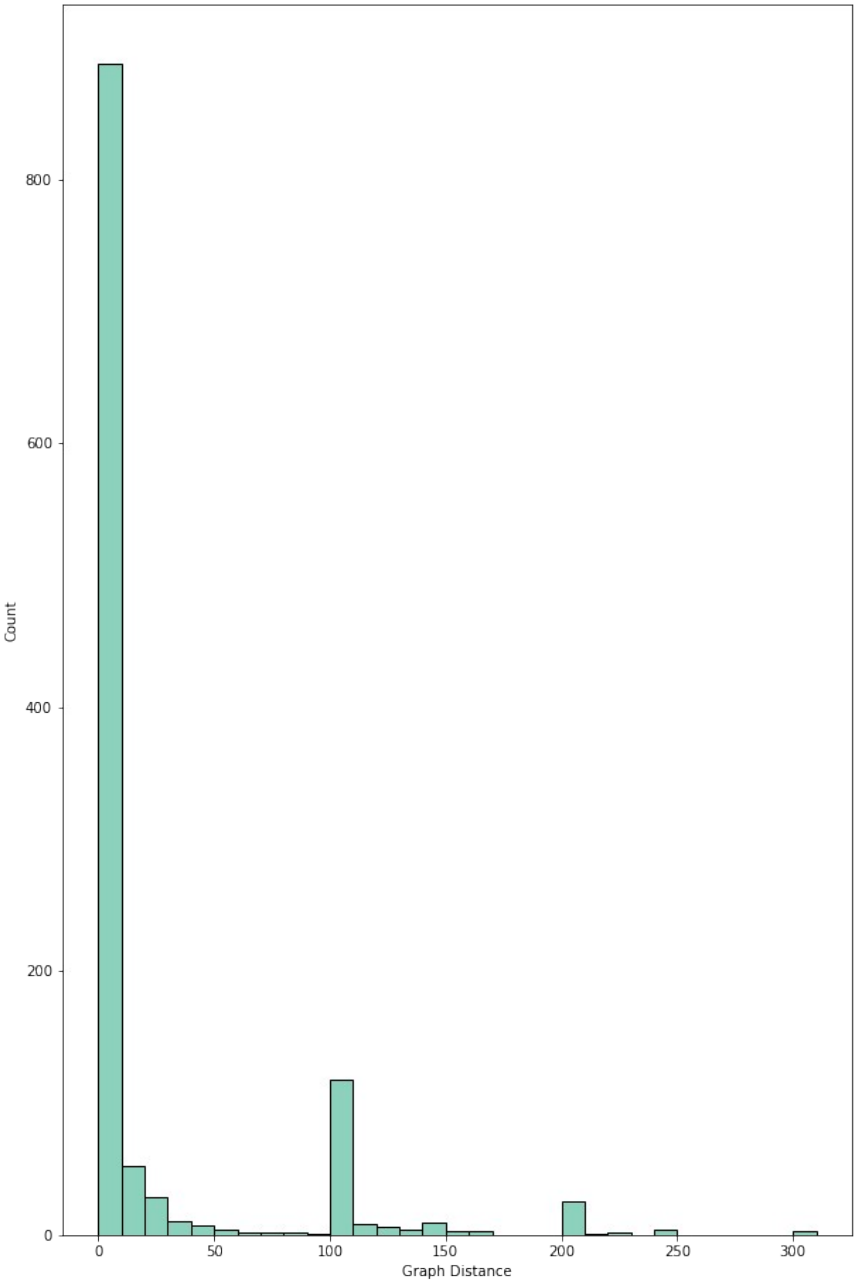
Score we achieved is 0.965, which is alright. However, as in the previous case, many states are very similar or outright identical, especially for starting state. If we allow the distance between graphs to be less than 5 score is 0.999. However it is obvious that on the training set we will get a very good result, so this test is mostly just a sanity check of our model.



Distributed bag of words (PV-DBOW) Validation

First, we embed states from the validation run then we find the most similar embedding from the training set and compare the two graphs.

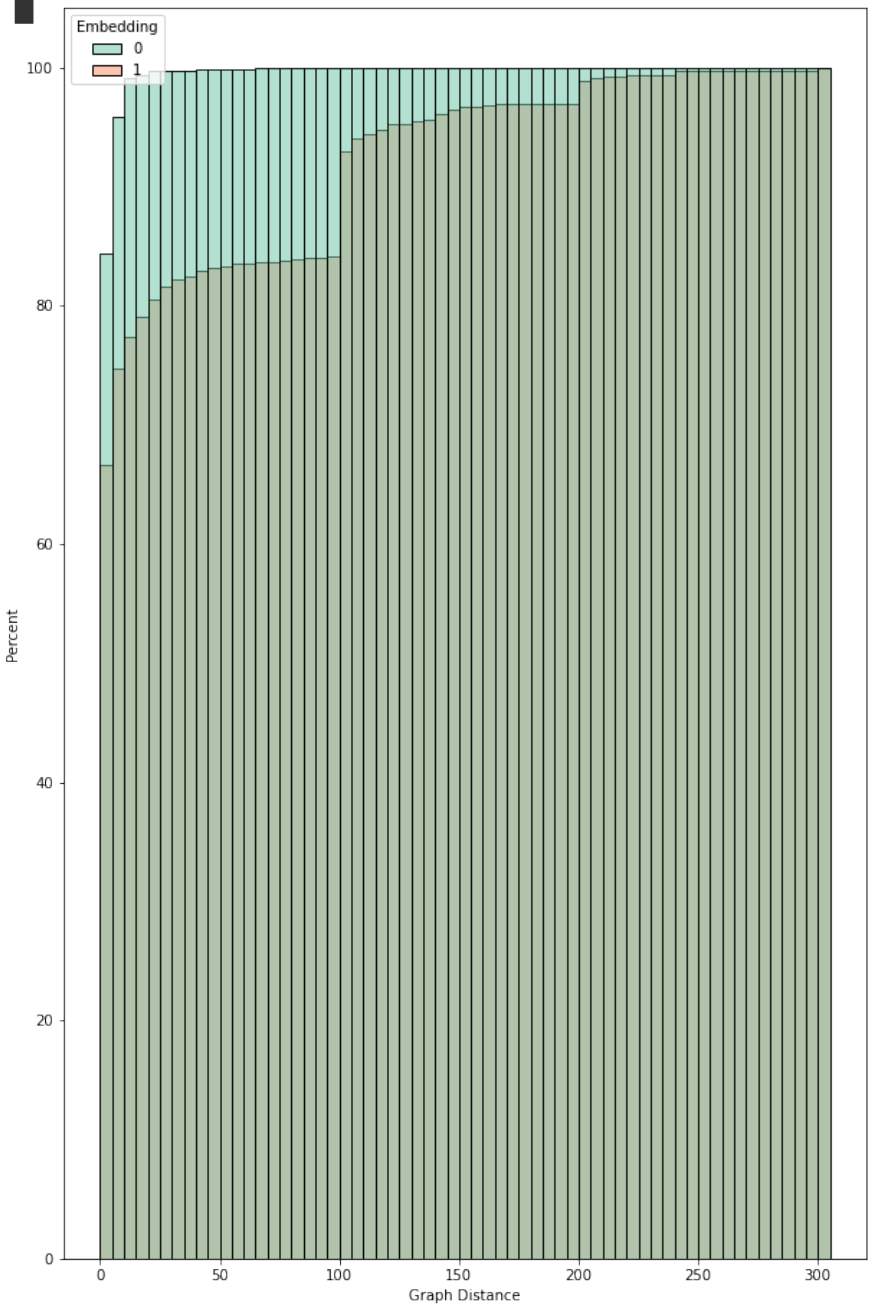
- If we allow state difference to be less then 5 then we get 0.665.
- If we relax our demands to less than 100 so still no missing/mislabeled objects are allowed we get 0.840
- if we want our demands to be less than 200 so 1 missing/mislabeled object is allowed we get 0.969.



PV-DBOW Validation compared to the best match

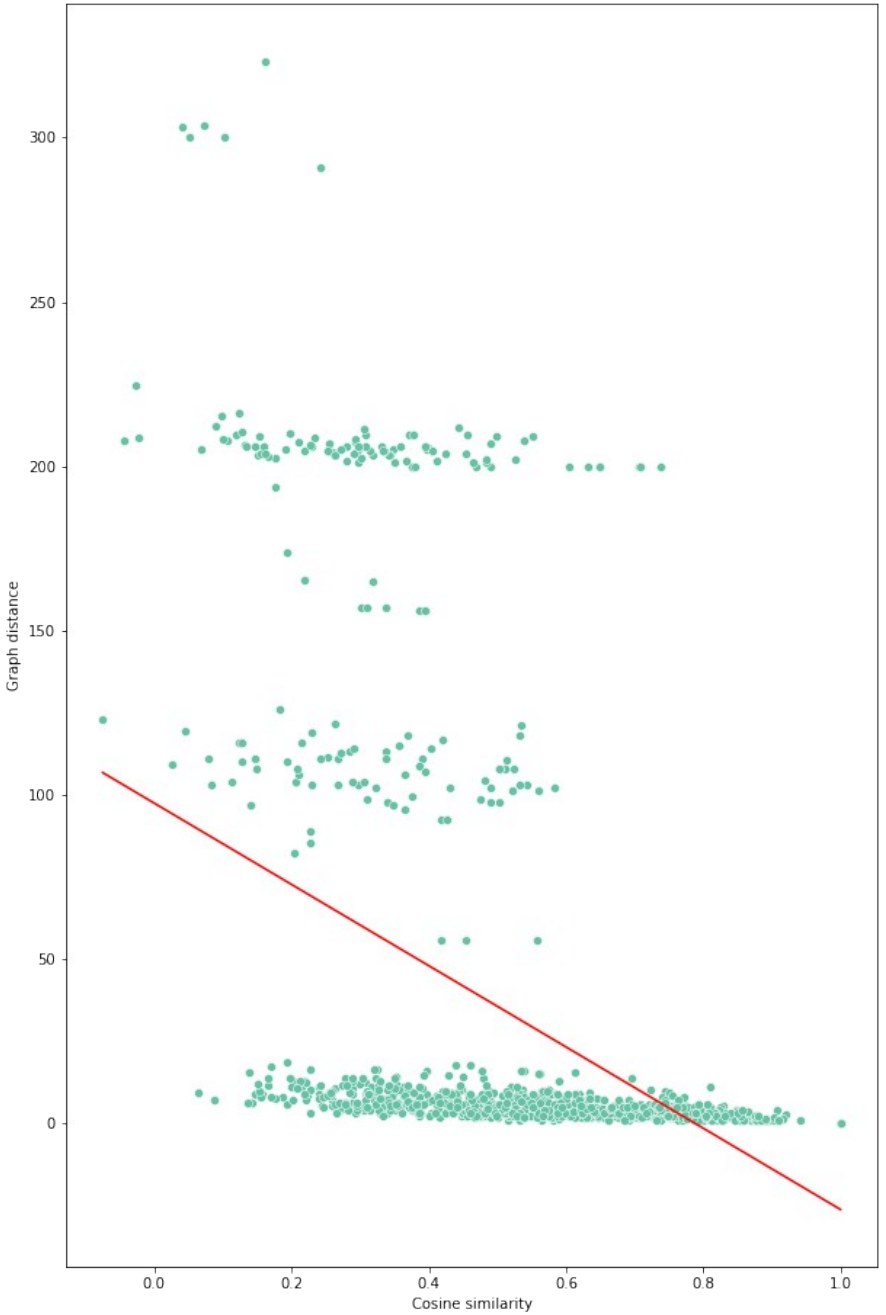
However, it is worth noting that often it is impossible to find a perfectly close graph from the training set. So we use the graph similarity metric directly for comparison.

- If we allow state difference to be less then 5 then we get 0.843 and ratio is 0.789
- If we allow state difference to be less then 100 then we get 0.999 and ratio is 0.841
- If we allow state difference to be less then 200 then we get 1.0 and ratio is 0.969



Checking correlation

Let's also look at how the cosine similarity of embedding correlates to the difference in graphs. For the validation run, we will calculate differences between adjacent frames. The Pearson Correlation between those two metrics is -0.463. That means that there is a quite strong negative correlation between them. Correlation is negative because when two documents are semantically similar the cosine similarity goes to 1 and when they are similar graph distance should go down.



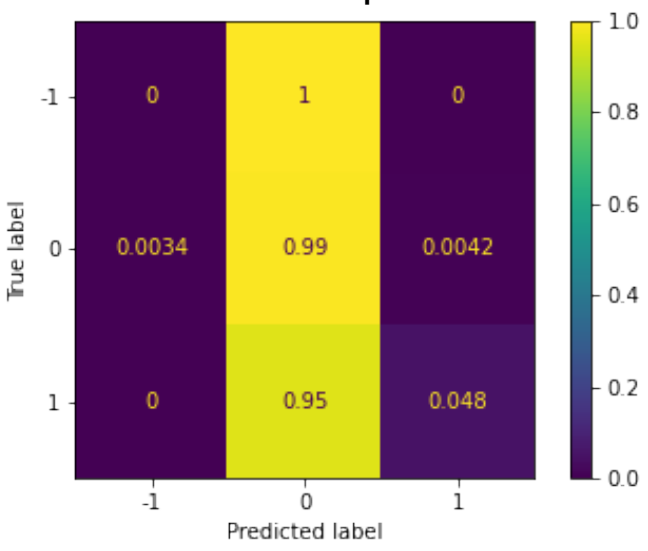
Testing State prediction

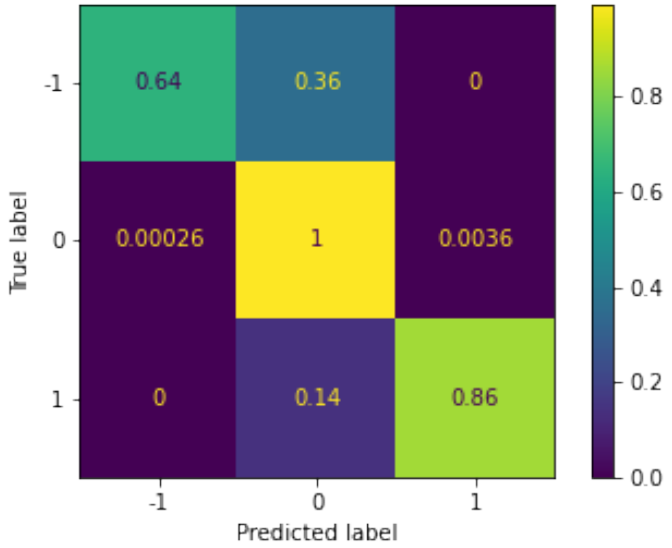
To test state prediction we made a new run. This run had 3875 timesteps.

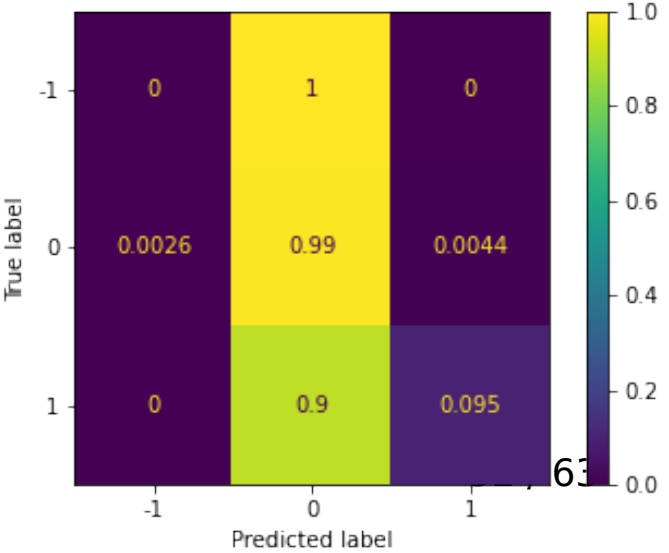
- Original neural network: 31.5% R2 score.
- New neural network trained and tested on the new trajectory: 75.3% R2 score.
- New neural network trained and tested on the new trajectory with 5-fold cross-validation: 2.5% R2 score from best split.
- New neural network with 128 neurons trained and tested on the new trajectory: 49.9% R2 score.
- New neural network with 128 neurons trained and tested on the new trajectory with 5-fold cross-validation: 17.9% R2 score from best split.
- Original neural network retrained with new trajectory: 69.3% R2 score.

Testing reward prediction

Validation run had 21 positive rewards, 14 negative rewards, and 3840 neutral rewards. This data is very imbalanced so we are using balanced accuracy. Our reward prediction model got 34.6% balanced accuracy. If we train and test SVM from scratch on the new trajectory we get 83.2% balanced accuracy. If we use 5-fold cross-validation on this new network we get 72.1% from the best split. If we update our old SVM with new data we get 36.2%. When we look at confusion matrix of our original model (left matrix). We can see that almost all transitions were classified as neutral. We can see that none of the positive rewards were classified as negative rewards and vice versa. After update (right matrix) we can see that more positive rewards were correctly classified. When we look at the confusion matrix of the model trained on the whole trajectory from the beginning (center matrix) we see that it was easier to classify positive rewards than negative rewards. We get 64% accuracy for negative rewards and 86% for positive rewards.





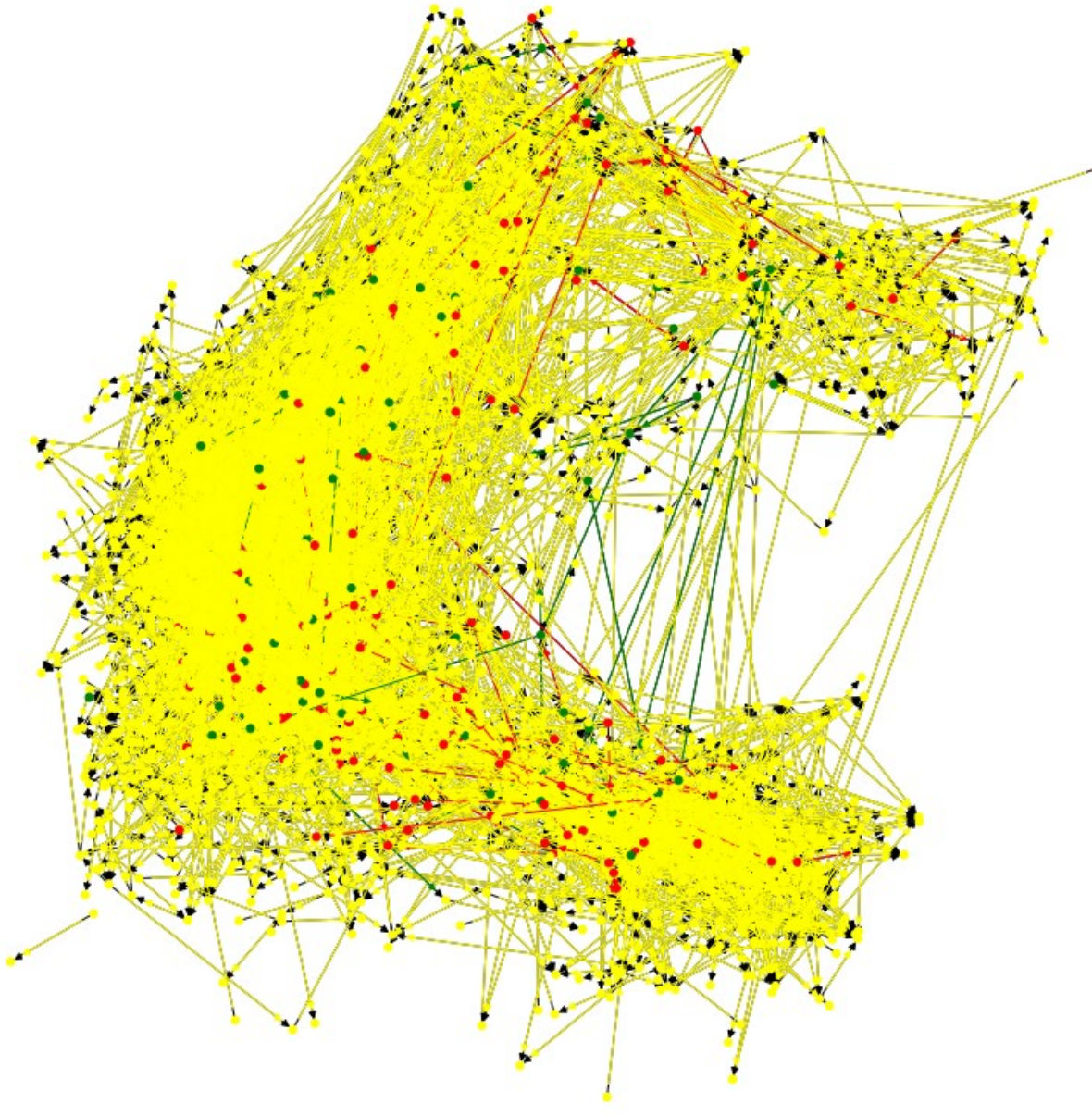


State map Birch trees description

After finishing training our node Birch tree got 2184 centroids and our edge Birch tree got 6969 centroids. If we try to classify states and transitions of our 3875 timesteps long test run we get 960 unique node labels and 1213 unique edge labels. So we compressed states by a factor of 4 and transitions by a factor of around 3. If we try to fit Birch trees from the beginning we get 321 unique node labels and 744 unique edge labels. The reason for this difference is that as the number of samples grows it is easier for the Birch tree to form new subclusters. Another metric is the reward stored by the edge centroids. Here 1.6\% of centroids had positive rewards and 2.5\% had negative rewards.

State map description

Here red color represents negative rewards, green positive rewards, and yellow neutral rewards. positions of nodes were determined by the first two Principal Components which together explain only 10.8% of variance which may be the reason why red and green nodes are not separable in the picture. The diameter of this graph is 20 and the radius is 9.



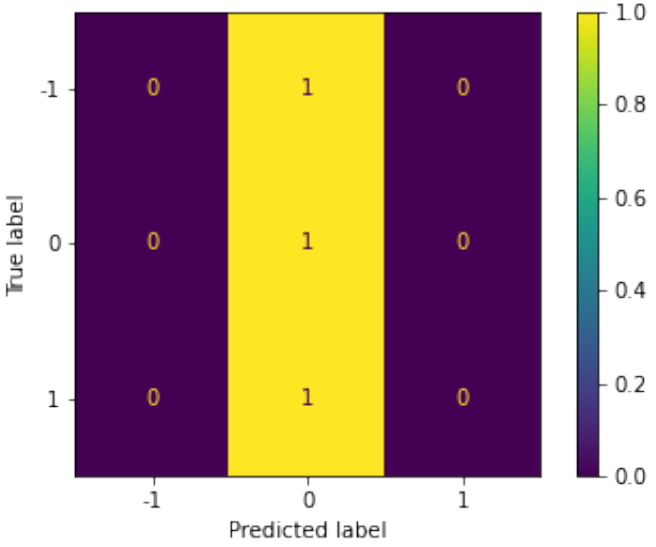
Testing State prediction on RAM

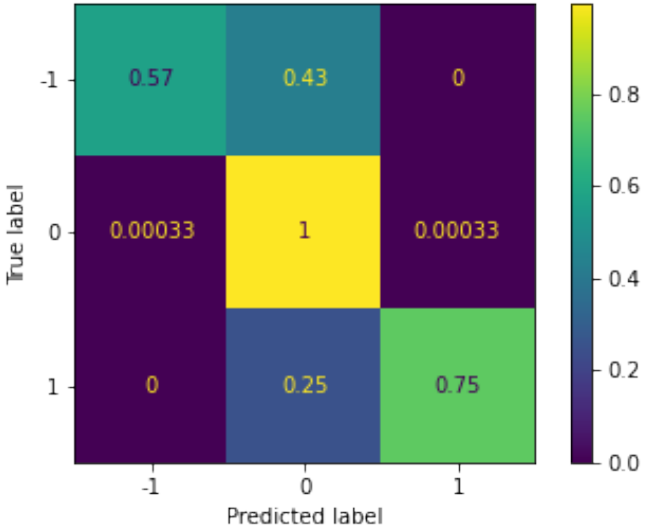
To test state prediction we made a new run. This run had 9022 timesteps.

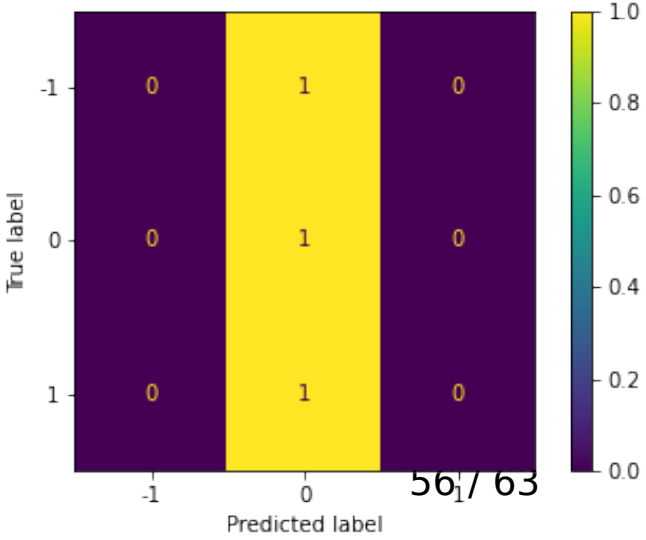
- Original neural network: 97.3% R2 score.
- New neural network trained and tested on the new trajectory: 97.4% R2 score.
- New neural network trained and tested on the new trajectory with 5-fold cross-validation: 98.1% R2 score from best split.
- New neural network with 128 neurons trained and tested on the new trajectory: 97.8% R2 score.
- New neural network with 128 neurons trained and tested on the new trajectory with 5-fold cross-validation: 97.2% R2 score from best split.
- Original neural network retrained with new trajectory: 98.6% R2 score.

Testing reward prediction on RAM

Validation run had 21 positive rewards, 21 negative rewards, and 8989 neutral rewards. This data is very imbalanced so we are using balanced accuracy. Our reward prediction model got 33.3% balanced accuracy. If we train and test SVM from scratch on the new trajectory we get 77.3% balanced accuracy. If we use 5-fold cross-validation on this new network we get 77.3% from the best split. If we update our old SVM with new data we get 33.3%. When we look at confusion matrix of our original model (left matrix). We can see that all transitions were classified as neutral. We can see that none of the positive rewards were classified as negative rewards and vice versa. After update (right table) we see no change. When we look at the confusion matrix of the model trained on the whole trajectory from the beginning (center table) we see that it was easier to classify positive rewards than negative rewards. We get 57% accuracy for negative rewards and 75% for positive rewards.





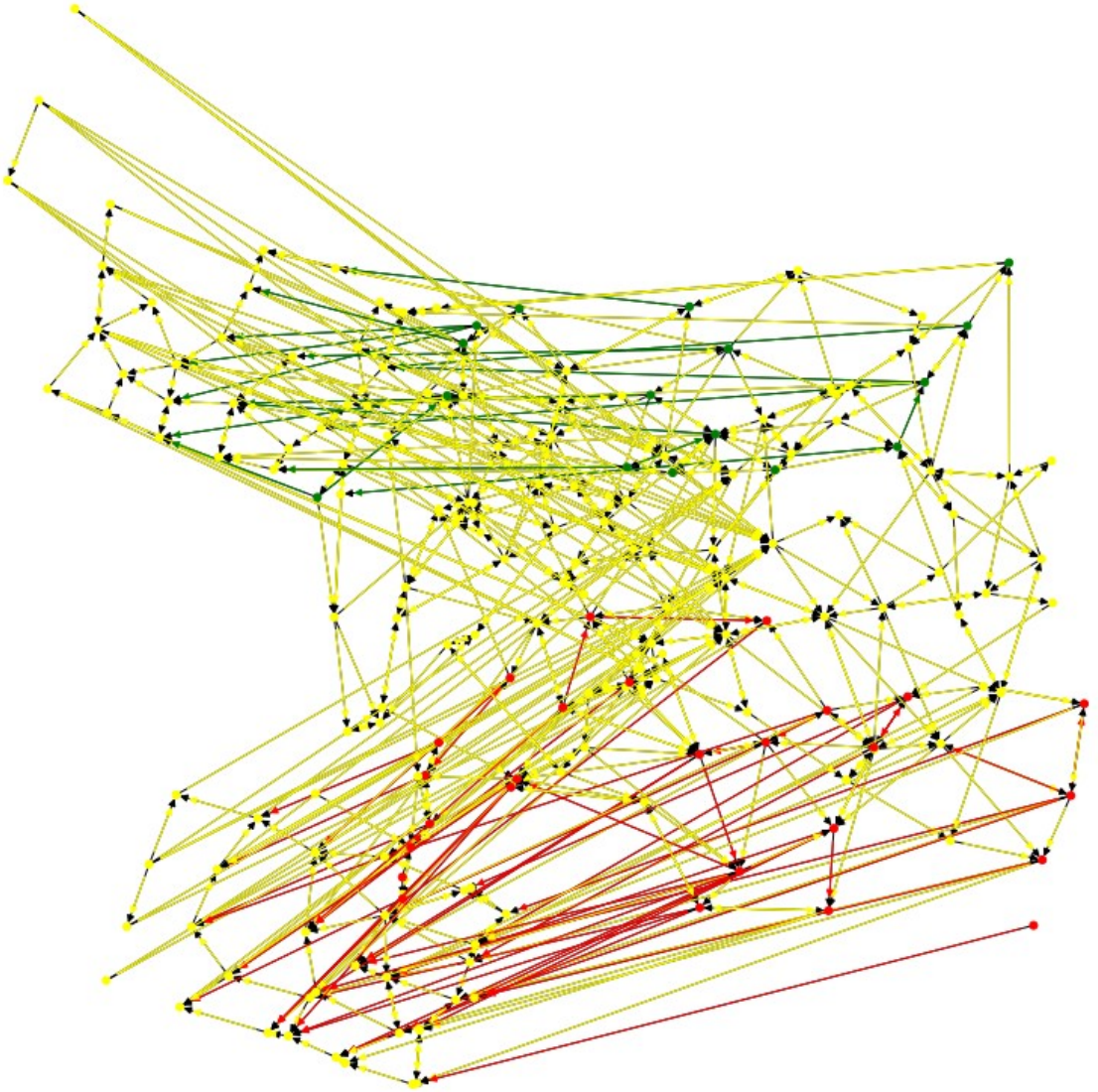


State map Birch trees description RAM

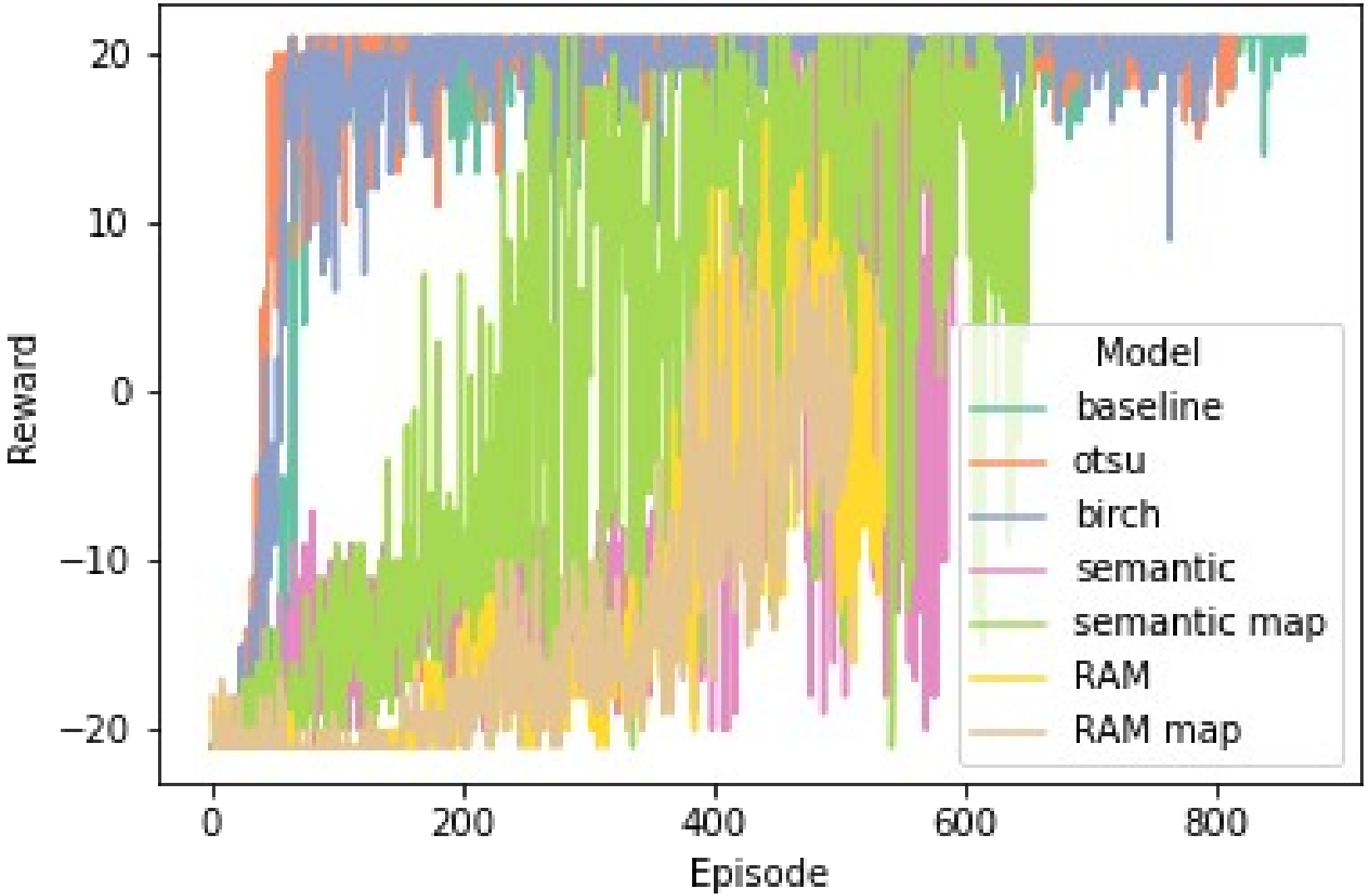
After finishing training our node Birch tree got 185 centroids and our edge Birch tree got 1966 centroids. If we try to classify states and transitions of our 9022 timesteps long test run we get 146 unique node labels and 802 unique edge labels. If we try to fit Birch trees from the beginning we get 79 unique node labels and 373 unique edge labels. The reason for this difference is that as the number of samples grows it is easier for the Birch tree to form new subclusters. Another metric is the reward stored by the edge centroids. Here 1.8% of centroids had positive rewards and 3.1% had negative rewards.

State map description RAM

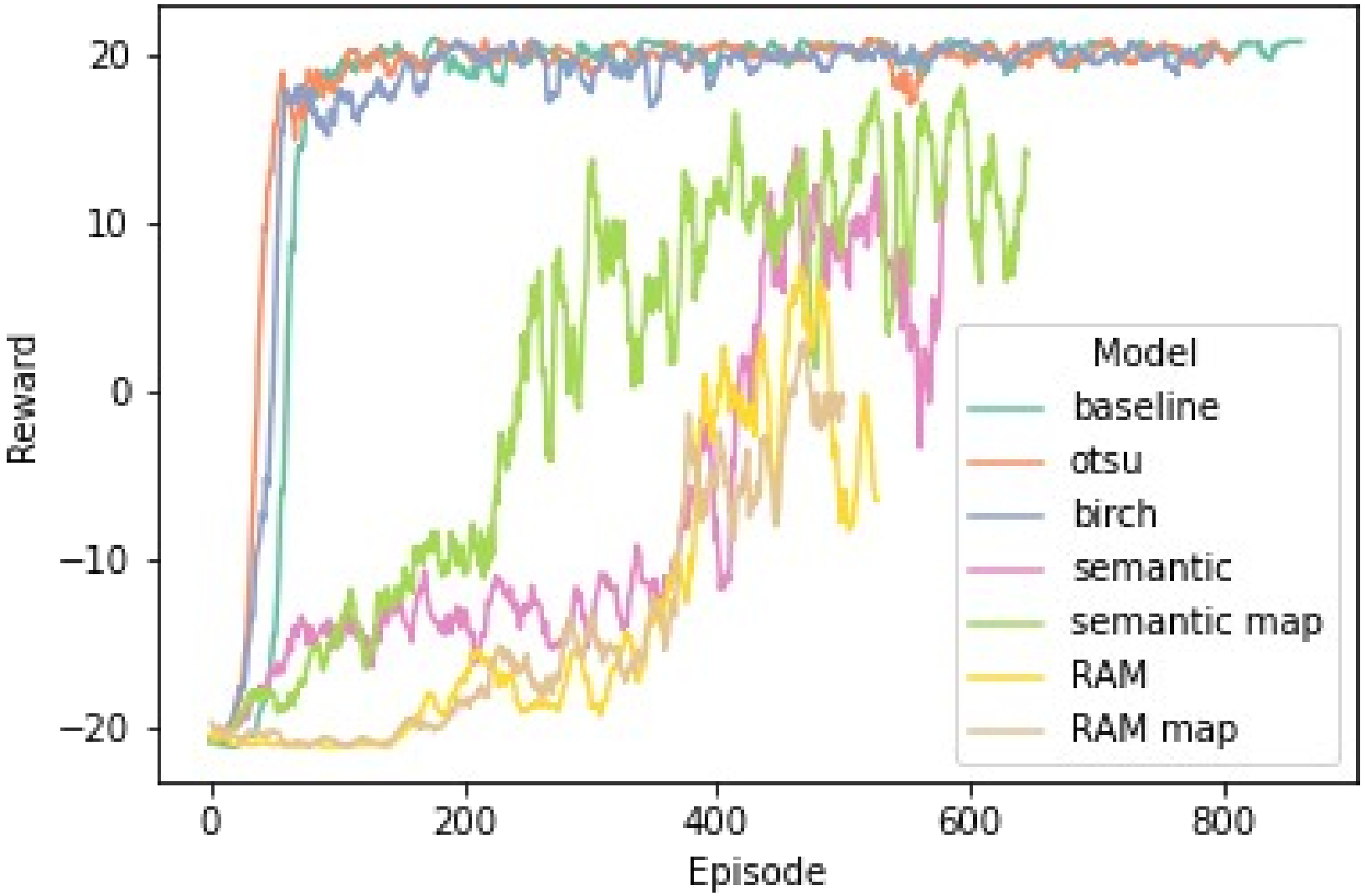
Here red color represents negative rewards, green positive rewards, and yellow neutral rewards. positions of nodes were determined by the first two Principal Components which together explain 66.4% of variance. We can easily see separation between red and green nodes. The diameter of this graph is 17 and the radius is 8.



Final Results



Final Results Average



Results: last 10 runs

Model	Mean	Max	Min	Std
baseline	20.8	21	20	0.42
otsu	20.1	21	18	1.19
birch	20.4	21	18	0.96
semantic	13.6	20	-10	9.60
semantic map	14.1	21	3	6.26
RAM	-6.4	4	-12	4.55
RAM map	-0.6	8	-6	4.76

Summary

- The fastest converging was one based on Otsu detection.
- Models based on semantic embedding and RAM were lagging behind. These models also displayed very high variance.
- Model based on semantic embedding outperforms RAM based one. However semantic model displays very high variance.
- Models which got access to generated goals by our reasoners got better results than ones without it.
- However by using only proposed actions by planner we could not score any points.

Bibliography

- My master thesis: Exploration of the usage of semantic reasoning in reinforcement learning
- https://www.baeldung.com/wp-content/uploads/sites/4/2020/10/multiclass-svm3-e1601952776445.png
- https://scipy-lectures.org/packages/scikit-image/auto_examples/plot_threshold.html