COMP390 2024/25

Improving the Sample & Feedback Efficiency of Reinforcement Learning with Human Feedback

Student Name: Joel Millar

Student ID: 201625697

Supervisor Name: Dr Bei Peng

DEPARTMENT OF COMPUTER SCIENCE

University of Liverpool
Liverpool L69 3BX

Dedicated to my parents and brother

Acknowledgements

I’d like to thank my Mum and Dad as well as Dr Bei Peng for helping me throughout the project.

Parts of this project are based on the PEBBLE [1] algorithm written by Kimin Lee, Laura Smith and Pieter Abbeel.

COMP390 2024/25

Improving the Sample & Feedback Efficiency of Reinforcement Learning with Human Feedback

DEPARTMENT OF COMPUTER SCIENCE

University of Liverpool
Liverpool L69 3BX

Abstract

Reinforcement learning with human feedback allows an agent to complete a hard to define task by using a human participant as a guide. However, current state-of-the-art algorithms lack the efficiency required to deploy these models into real-world scenarios. This is due to the high cost of human feedback and how much feedback would be required using current systems. I present an improvement upon the current systems which allows the agent to have improved sample and feedback efficiency. Specifically, I improve the current methods used to select queries to present to the human by utilising machine learning techniques so that optimal queries are chosen for learning. Additionally, I explore the current methods used for the reward model ensemble and how it can be improved to create a more stable and diverse model. I also explore further methods of improvement that show potential for future updates to reinforcement learning with human feedback algorithms such as new methods of obtaining human feedback and improve the unsupervised pre-training stage. I show that my results are an improvement upon current systems by directly comparing my results against the Feedback-Efficient Interactive Reinforcement Learning via Relabelling Experience and Unsupervised Pre-training (PEBBLE) algorithm [1]. The proposed changes find an improvement in the performance of PEBBLE in terms of the episode reward gained and the rate of growth of the algorithm in more complex environments such as quadruped walk whilst maintaining the high performance achieved in simpler environments like walker. The new algorithm can also improve upon the PEBBLE baseline results even when the amount of human feedback provided is greatly reduced proving that this new method can select better queries for the human to maximise model learning and makes reinforcement learning with human feedback more suitable for real world applications.

Statement of Ethical Compliance

Data Category: A
Participant Category: 0

I confirm that I have followed the ethical guidelines during this project. Further details can be found in the relevant sections of this dissertation.

Table of Contents

1. Introduction and Background

In this project I aim to find improvements in the sample and feedback efficiency of reinforcement learning with human feedback algorithms. I will use the Feedback-Efficient Interactive Reinforcement Learning via Relabelling Experience and Unsupervised Pre-training, also known as PEBBLE, [1] as a base algorithm which I hope to update using new ideas to find improvements in its performance. To develop these ideas, I will explore the different features of PEBBLE and implement changes based on what has the best potential to find increased performance.

The problem this project aims to address is the unsuitability of current state-of-the-art reinforcement learning with human feedback algorithms for application in the real world. The amount of feedback currently required by algorithms to perform well in each environment is far too high for a human teacher to be realistically involved in. By the end of this project, I hope to have made progress in reducing the amount of human feedback required for an algorithm to perform well and therefore taking reinforcement learning with human feedback a step closer to being deployed to solve real world problems. An example of where reinforcement learning with human feedback could be used is to teach children according to their learning style. In this type of environment the human teacher, in this case the child being taught, can provide feedback on how they are being taught, and the agent can alter how they are teaching to benefit this child's requirements. This is especially useful for different types of learners, e.g. visual, audio etc. However, current reinforcement learning with human feedback algorithms would require so much feedback from the student that it would be incredibly time consuming and inefficient to require that much information to teach a potentially very simple topic. When this project is completed, I hope that tasks like this will be closer to reality as the agent will use the feedback more efficiently and therefore less will be required to complete tasks to the same performance level. Reinforcement learning with human feedback has the potential to be an incredibly powerful tool in scenarios that have previously been considered impossible to be done without humans and that this project can help their development take a step closer to usage in the real world.

1.1. Glossary

Term	Definition
Activation Function	A mathematical function that determines the output of a neural network's node
Clustering	A machine learning technique where datapoints are assigned to a centre point based off distance to create groups of similar points
Cosine Similarity	A measure of similarity between two vectors by using the cosine of the angle between them
Ensemble	A group of neural networks where their outputs are combined to improve performance
Euclidean Distance	The shortest path between two points in Euclidean space
Intrinsic Motivation	An agent's internal drive to explore its environment and learn

KMeans Clustering	Clustering technique where the centroids are updated after each point is assigned to be the mean value of all points in that group
Loss Function	Mathematical function which calculates the difference between a model's actual result and expected result
Manhattan Distance	Distance measurement which is the sum of the two points coordinates
Oracle Teacher	Data that emulates human teachers feedback where all answers are accurate and encourage learning
PCA (Principal Component Analysis)	A dimensionality reduction technique which does not remove important relationships
Policy	An agent's strategy for selecting actions in each state
Q-Function	Takes current state and actions and produces an expected reward after that action is taken
Reward Function	A function that assigns a reward value to an agent's actions
Reinforcement Learning	Machine learning technique where an agent learns to make decisions in an environment to maximise its reward
Trajectory	A short video clip 1~2 seconds which shows the agents actions

1.2. Aims and Requirements

The aims I expect to achieve in this project are as stated:

- Successfully reproduce the results of the PEBBLE algorithm on my own system to within a reasonable range of difference and garner a deep understanding of how the algorithm works.
- Implement my improvements upon the sampling method used to select the most informative queries for the human participant to provide feedback on into my own algorithm.
- Implement my improvements to the ensemble of neural networks to improve overall performance including the accuracy of predictions of key metrics like reward gained.

Essential Requirements:

- Identify key research papers to develop my reinforcement learning with human feedback understanding and help me identify areas of improvement
- Build a strong level of knowledge around reinforcement learning and reinforcement learning with human feedback through conducting a literature review
- Set up and run the PEBBLE algorithm and required libraries to create baseline results
- Investigate sampling improvements for selecting informative queries for the human teacher
- Investigate improvements to the ensemble structure and application used by PEBBLE
- Study the performance of the new preference-based reinforcement learning algorithm compared to PEBBLE

Optional Requirements:

- Implement a new unsupervised pre-training algorithm that can learn more diverse behaviours and accelerate the learning process
- Explore using different methods of obtaining human feedback
- Explore the potential of updating the way the reward function is used so that it cannot become non-stationary during training – this could include defining a reward function and not updating it in the training process

2. Related Work

Feedback-Efficient Interactive Reinforcement Learning via Relabelling Experience and Unsupervised Pre-training [1] also known as PEBBLE is the state-of-the-art reinforcement learning with human feedback algorithm which this project builds upon. PEBBLE is comprised of three steps: Unsupervised pre-training, learning a reward function, updating the agent's policy and value function using the learned reward model, then repeating steps two and three until training is complete. The unsupervised pre-training step implemented by PEBBLE is one of the large differences between PEBBLE and other reinforcement learning with human feedback algorithms and in turn helps PEBBLE outperform similar algorithms. Unsupervised learning is where the agent explores their environment using intrinsic motivation, simple goals for the agent to achieve such as being rewarded for visiting states it hasn't visited as often. This helps the agent in collecting experience and allows for better queries to be provided to the human teacher. The second step is the reward learning where the agent provides a human teacher with two short clips (1~2 seconds) called trajectories and the human then provides feedback in the form of a preference between the two clips.

A reward function is then learned using this feedback so that it will be able to train the agent in a way that aligns with the human teachers' intentions. By following the Bradley-Terry model PEBBLE produces a preference predictor for estimating which trajectory the human participant would prefer for the agent based on previously observed preferences. The predictor enables the system to infer a reward model that captures the humans desired behaviour:

$$P_\psi[\sigma^1 \succ \sigma^0] = \frac{\exp \sum_t \widehat{r}_\psi(\mathbf{s}_t^1, \mathbf{a}_t^1)}{\sum_{i\in\{0,1\}} \exp \sum_t \widehat{r}_\psi(\mathbf{s}_t^i, \mathbf{a}_t^i)},$$

Figure 1: PEBBLE equation for preference predictor where σi ≻σj denotes the event that segment i is preferable to segment j. Taken from PEBBLE paper [1]

The third step is agent learning where the policy, the agents learned strategy for selecting actions in each state, and Q-function, takes current states and action and produces expected reward, are updated using an off-policy, different to the policy being used to interact with the environment, reinforcement learning algorithm with relabelling. Because the reward model is updated frequently past experiences are relabelled with new predictions for rewards.

Benchmarking Preference-based Reinforcement Learning [2] also known as B-Pref is a benchmark providing the ability to evaluate candidate algorithms quickly which makes relying on human input for evaluation prohibitive. Different types of human behaviour can be simulated when providing feedback to the agent by choosing between two segments: myopic, skipping, equally preferable, making a mistake. To deal with these different behaviours

different variables are used such as sum of rewards with discount factor, skip threshold, checking if segments have similar sum of rewards. This helps reduce the errors a human participant could accidentally involve in their feedback. When using B-Pref to analyse algorithmic design decisions for preference-based reinforcement learning, PEBBLE [1] outperformed PrefPPO [3] in most environments. It also found that uncertainty-based sampling schemes are superior to other sampling schemes. To improve, the labels can be more robust so that they do not corrupt as easily by the human participants. E.g. making a mistake or misunderstanding the task. Overall, B-Pref is an evaluation tool that can be used to help optimise the performance of reinforcement learning with human feedback models which will then aid in knowing when and how models are improved or worsened when different algorithms or feedback methods are employed. B-Pref will be used heavily in this project for imitating the human participant, not only will this help with consistency of the human feedback collected but it will also greatly reduce the amount of time required for testing as all preferences are already saved and available. In this project only the oracle teacher will be used who provides perfect feedback for the model to learn.

Deep Reinforcement Learning from Human Preferences [4] proposes the use of short video clips of the agent, between 1 and 2 seconds long, which the human participant must analyse and decide which is the preferred choice, if they are both bad choices or if they are inseparable. The user expresses preferences over trajectory segments which are sequences of observations and actions which will help the agent in deciding its next actions. The agent will then produce a probability that the human will choose each given segment to aid in reducing the number of times the human participant must provide feedback. Similarly to PEBBLE, the reward function can be non-stationary which will make it significantly more challenging to deal with. This method also assumes that the human is indifferent about when things happen in the trajectory segment. The short clips can also be difficult to understand if there is a complex task as they supply no context even if the human participant is an expert. Overall, this method performs well in some scenarios such as Enduro game however fails to learn on QBert as the clips are too difficult to understand which emphasises a strong relationship between the quality of the clips, how much information they can provide, whilst also keeping them short so that feedback is not excessive and the performance of the learning model.

3. Design and Implementation

3.1. Reproducing PEBBLE results

Reproducing the results achieved in the PEBBLE [1] paper on my own system was vital to the success of this project as it would provide me with a suitable baseline of which to compare my results as well as providing further understanding of how the algorithm currently worked.

One of the largest and unexpected challenges of this project was the process of setting up the correct libraries to run PEBBLE in my own environment. Due to the complexity of both the algorithm and the environments where it is used, all the libraries must be compatible with one another as well as the version of python that was used. The source of the issues stemmed from the GitHub repository where the files responsible for setting up the virtual environment to run the code in were outdated. For example, for some libraries a version would not be given so if

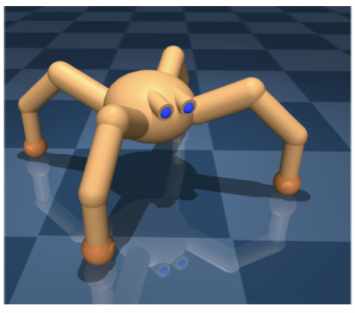
any updates had made which removed or changed features the library would no longer work. The first approach was to update all the libraries to be as recent as possible however this brought about problems with the compatibility of the python version being used: 3.6. The best way to address this problem was to use a more modern python version (3.9) and then use as many libraries in the described version as possible and for those that need to be changed a trial-and-error process was used by looking through documentation and finding what libraries it required or required it and the compatible versions. Despite this being a time consuming and frustrating process, overall, it helped me understand each of the libraries in more detail and what their specific role in the algorithm was which aided my overall understanding of PEBBLE.

To run PEBBLE I used Barkla, which is the University of Liverpool's high performance computing service which allowed me to use powerful processers with large memory and storage space to train the model in a fraction of the time it would take me if using my own device, however due to the service being used by many other users there would be problems with having to wait several hours before my code started to run or my code being interrupted several hours into its training due to other users pre-empting my jobs. There would also be times when the service would be inaccessible and therefore, I was unable to run or view the jobs I had run. However, without the use of Barkla this project would have been impossible to complete.

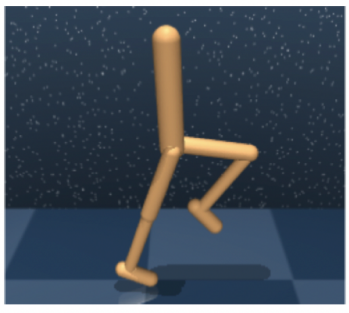
3.2. Environments

The environments my proposed changes will be tested on are locomotion tasks from Deep-Mind Control Suite [5] which is a set of continuous control environments designed for testing the suitability of reinforcement learning algorithms. Specifically, I will be using the Walker environment due to the PEBBLE's [1] high performance in this environment which I will aim to maintain with my changes and the Quadruped environment as this is one of the worst performing environments for PEBBLE, so I aim to find improvements to make the algorithm suitable in more complex environments as well as simple ones.

The walker environment is of a two-legged humanoid agent which aims to walk as far as possible in a physically realistic 3D environment.
The quadruped environment is a four-legged agent which also aims to walk as far as possible in a physically realistic environment.



Quadruped



Walker

Figure 2: Examples of the Quadruped and Walker environment that the algorithm is tested on. Taken from PEBBLE paper [1]

As seen in figure 2 the quadruped must learn to work using twice as many limbs as the walker and is therefore a much more complex environment and is challenging for PEBBLE to achieve a high episode return in. I will examine the performance of each of my proposed changes and compare this to PEBBLE in each environment.

3.3. Original PEBBLE results:

The original PEBBLE [1] results provided in the paper are shown below which show PEBBLE compared to similar state-of-the-art algorithms. For the environments Quadruped and Walker, I aim to have results within a reasonable level of difference in the episode return given that the only difference should be the system running the code.

As seen by the results in figure 3 the walker environment achieves episode returns around 900 very quickly making it the best performing environment whilst the quadruped environment achieves episode return of 800 only in its final environment steps making it one of the worst performing environments for PEBBLE. The results of my changes should be able to maintain the high performance of the walker environment whilst improving the results of the quadruped.

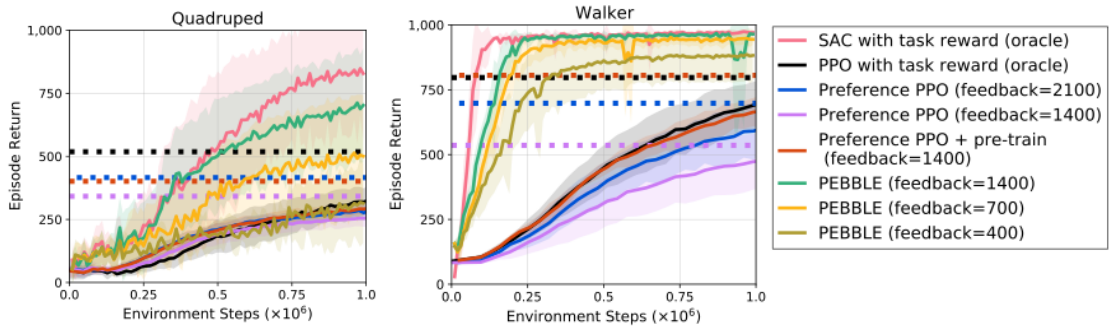


Figure 3: PEBBLE performance in the Quadruped and Walker environment with feedback 400, 700 and 1400 along with other similar state-of-the-art algorithms. The solid line represents the mean and the shaded region represents the standard deviation. Asymptotic performance of PPO and Preference PPO is indicated by dotted lines of the corresponding colour. Taken from PEBBLE paper [1]

In the PEBBLE results we observe the difference the amount of feedback has on the episodic return with the increase in feedback leading to an increase in the algorithmic performance. This highlights the significant impact the amount of feedback has on the algorithm's performance.

3.3.1. PEBBLE Baseline results:

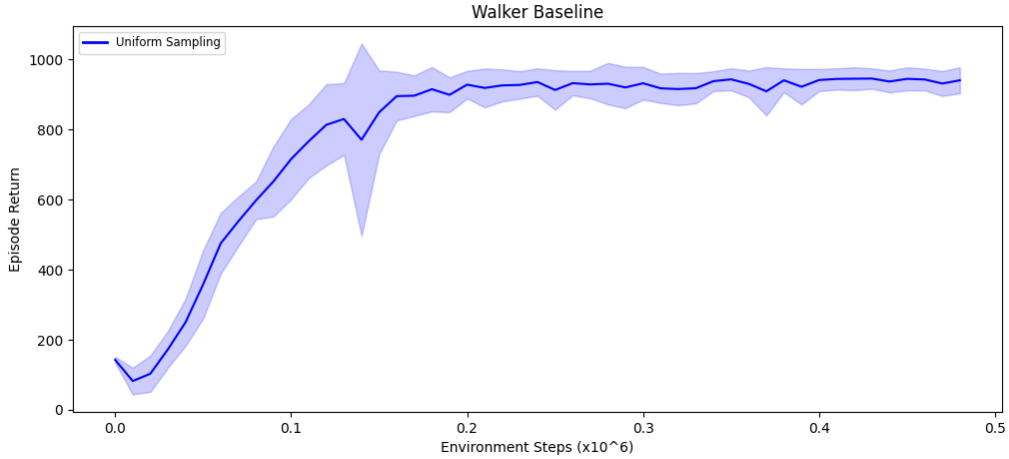


Figure 4: Walker environment with 1000 feedback using the oracle teacher and uniform sampling method. The solid line represents the mean and the shaded region represents the standard deviation.

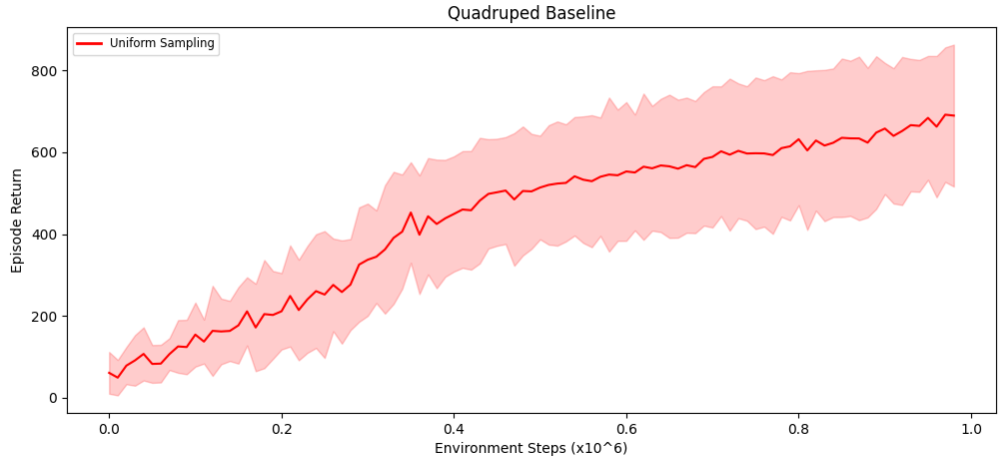


Figure 5: Quadruped environment with 1000 feedback, oracle teacher and uniform sampling method. The solid line represents the mean and the shaded region represents the standard deviation.

In figure 4 and figure 5 we observe that the pattern of the mean episode for the walker and quadruped environments match the original PEBBLE [1] results, and it achieves a similar episode return so that it can be used as a baseline to monitor how the changes I make effects the performance.

3.4. Proposed Changes

Over the course of this project many changes were made to try and find improvements and most of these changes provided no change or a decrease in performance however the experience gained in finding out why these changes negatively affected the algorithm assisted

me in understanding the code further and discovering what new changes could be made that can show improvement.

As I continued to implement and experiment with changes made to the algorithm this enabled me to identify the areas with the most potential for improvement and how that improvement could be found. My final goal for this project is to piece together the best performing changes to the algorithm and produce a new, sample efficient and high performing algorithm that is more suitable for real world applications.

3.5. Sampling Methods for Selecting Informative Queries

A key area I identified for finding improvement in PEBBLE [1] was the method used for selecting informative queries which are passed to the human teacher for their feedback. The queries take the form of two short video clips around 1 ~ 2 seconds long called trajectories where the human teacher must provide a preference as to which clip shows the most promising learning. This is done by sampling from a population of queries which have been gathered through past experience. The best performing method used by PEBBLE is uniform sampling as it achieves the best results over all the environments, however as seen in the figure below it can be worse performing than other methods and therefore by implementing a new sampling method an increase in performance can be found in the environments uniform sampling struggles with whilst maintaining the high performance in the environments it thrives in.

Figure 6 shows uniform sampling compared to disagreement and entropy sampling in the quadruped and walker environments. We observe that uniform achieves a disappointing maximum episode return and the growth gradient is also shallow which emphasises slow learning. We also see in the walker environment that uniform sampling learns slower than the other methods despite achieving a similar final episode return.

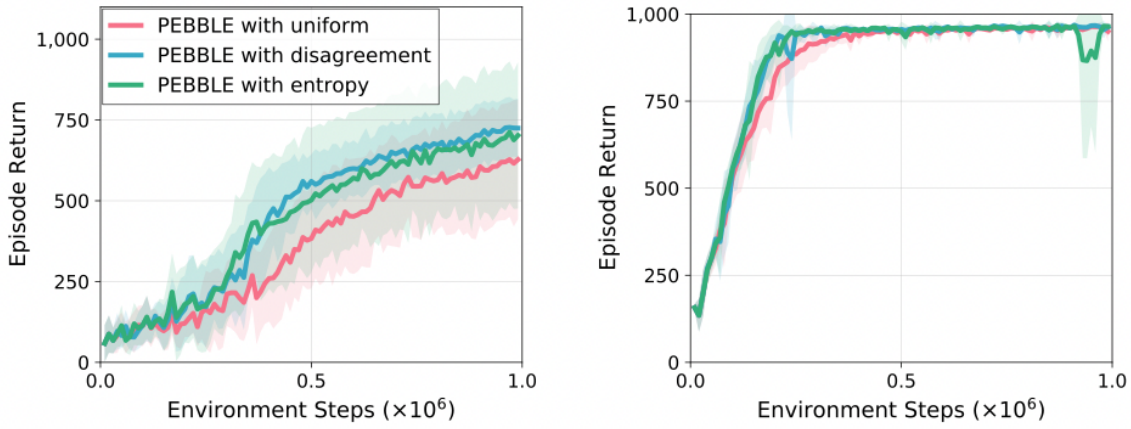


Figure 6: Comparison of the uniform, disagreement and entropy sampling methods on the Quadruped and Walker environments with 1400 feedback. The solid line represents the mean and the shaded region represents the standard deviation. Taken from PEBBLE paper [1]

3.5.1. Extreme Disagreement Based Sampling

The first and simplest method I implemented to find improvement through the sampling method was extreme disagreement-based sampling. The motivation behind developing this method was to build understanding of how the samples relate to one another and how taking extreme measures in selecting them influences the training process, specifically I examined how picking the datapoints with the highest disagreement in the population affects the algorithms performance.

```
def custom_sampling(self):
    # Extreme Disagreement-based sampling

    # get queries
    sa_t_1, sa_t_2, r_t_1, r_t_2 = self.get_queries(
        mb_size=self.mb_size*self.large_batch)

    # Probability rankings and disagreement score
    _, disagree = self.get_rank_probability(sa_t_1, sa_t_2)

    # Select top and bottom samples based on disagreement
    top_k_index = (-disagree).argsort()[:self.mb_size]
    bottom_k_index = (disagree).argsort()[:self.mb_size]

    # Select samples
    r_t_1, sa_t_1 = r_t_1[top_k_index], sa_t_1[top_k_index]
    r_t_2, sa_t_2 = r_t_2[bottom_k_index], sa_t_2[bottom_k_index]

    # Generate labels for selected samples
    sa_t_1, sa_t_2, r_t_1, r_t_2, labels = self.get_label(
        sa_t_1, sa_t_2, r_t_1, r_t_2)

    if len(labels) > 0:
        self.put_queries(sa_t_1, sa_t_2, labels)

    return len(labels)
```

Figure 7: Python code for extreme disagreement-based sampling for selecting informative queries

This function was formulated by using the same general architecture as the uniform sampling method where two batches of queries are selected using the 'mb_size' and 'large_batch' parameters and the function 'put_queries' is used once the queries have been selected. The key difference between this function is the implementation of more advanced selection logic. The 'get_queries' function selects a random number of samples given a parameter, so an increased selection is chosen and disagreement scores are calculated for each query. The queries with the highest and lowest disagreement scores are then selected as the final queries to be sent to the human teacher.

3.5.2. Clustering

The sampling method I chose to implement next was clustering. The inspiration behind choosing clustering was due to its ability to split the data into similar groups so the diversity of the samples selected is easier to control. The reason for uniform samplings high performance was due to the high diversity in samples it achieved, however due to the random nature this was not a consistent method to use. Clustering can split the data into smaller groups and then select samples which would provide a better representation of the population and by employing more advanced techniques to select the clustering centroids and the samples from within each cluster there is a great variety of ways clustering can be fine-tuned to maximise the episodic return.

The way I would use clustering is by taking the population, where previously samples were just randomly selected from, and cluster the datapoints into k number of clusters, where k is a changeable parameter, once each datapoint had been assigned to a cluster a selection algorithm would be used to then pick equal samples from each cluster so that the total number of selected samples does not change. These final selected samples are used to query the human teacher.

First Clustering Attempt:

The first working implementation of the clustering sampling method was comprised of a simple algorithm which would randomly select datapoints from the population for the clustering centroids, it would then iterate through the datapoints and assign each of them to the closest centroid. Once all datapoints had been sorted, the final samples were chosen randomly from each cluster.

```python
def kmeans_live(self, start=0, max_iter=1000, crit=1e-5):

        # Select cluster initiation algorithm
        if start == 0:
            # Random starting points
            sp = self.reg_start()
        elif start == 1:
            # KMeans ++
            sp = self.start_plus()
        elif start == 2:
            # Min max start - Pick furthest points
            sp = self.min_max_start()

        self.centroids = sp
        data_copy = self.data.copy()

        data_points = []
        data_clusters = []
        original_pos = []

        # Iterate through every datapoint and assign it to the closest centroid
        for j in range(len(data_copy)):
            point = data_copy[j]
            closest_d = float('inf')
            closest_cluster = 0

            # Find closest centroid for this point
            for i in range(len(sp)):
                distance = self.calculate_distance(sp[i], point)
                if distance < closest_d:
                    closest_d = distance
                    closest_cluster = i

            data_points.append(point)
            data_clusters.append(closest_cluster)
            original_pos.append(j)

        return data_points, data_clusters, original_pos
```

Figure 8: Python code for the first version of clustering implemented to select informative queries.

This clustering code is computationally more expensive compared to uniform sampling because of having to sort through each datapoint and assign it before then selecting the final samples from the clusters. This produces a time complexity of O(s*n*k) where n is the number of datapoints, k is the number of centroids and s is the number of features of the data, this may change due to pre-processing techniques used. However, the quality of the clusters created can be very poor unless the algorithm gets lucky with the initial centroid choice, this is an attribute I was keen to change for future iterations.

Overall, this algorithm is computationally fast relative to other clustering algorithms, but the quality of its cluster groups was worse.

KMeans Clustering:

KMeans clustering [6] would include convergence of the centroid starting points to improve the quality of clustering groupings. This method would employ the techniques used by KMeans clustering where once the clusters have been formed their average datapoint is found and assigned as the new centroid, then all the datapoints are re-assigned, this process is repeated for a set number of steps or until the difference between the new datapoint and old datapoint is below a given criteria which indicates enough convergence has taken place. The main reason for this approach to be the next step after the original clustering method was due to the increased quality of the clusters because not only would it have multiple chances to group values together, but it would also update the centroid for each group so that the groups are more tightly clustered and therefore a better selection of datapoints. The major downside to this change in clustering is the increase in computational time when running the algorithm so some hyperparameters had to be changed for this sampling method to work properly.

```
def kmeans_live(self, start=0, max_iter=1000, crit=1e-5):

        # Select cluster initiation algorithm
        if start == 0:
            # Random starting points
            sp = self.reg_start()
        elif start == 1:
            # KMeans ++
            sp = self.start_plus()
        elif start == 2:
            # Min max start - Pick furthest points
            sp = self.min_max_start()

        self.centroids = sp
        data_copy = self.data.copy()

        # Continue until max iterations value reached
        # or the difference criteria between centroids is met
        for iters in range(max_iter):
            data_points = []
            data_clusters = []
            original_pos = []

            # Iterate through every datapoint and assign it to the closest centro
            for j in range(len(data_copy)):
                point = data_copy[j]
                closest_d = float('inf')
                closest_cluster = 0

                # Find closest centroid for this point
                for i in range(len(sp)):
                    distance = self.calculate_distance(sp[i], point)
                    if distance < closest_d:
                        closest_d = distance
                        closest_cluster = i

                data_points.append(point)
                data_clusters.append(closest_cluster)
                original_pos.append(j)

            # Calculate new centroids
            # Calculate the mean values for each cluster and then replace origina
            new_centroids = []
            for k in range(self.n_clusters):
                clusters = []
                for i in range(len(data_points)):
                    if data_clusters[i] == k:
                        clusters.append(data_points[i])
                new_centroids.append(np.mean(clusters, axis=0))

            # Check criteria
            if np.linalg.norm(np.array(new_centroids) - np.array(sp)) < crit:
                break

            sp = new_centroids

        return data_points, data_clusters, original_pos
```

Figure 9: Final Python implementation of the clustering algorithm for selecting informative queries.

This algorithm works by picking initial centroids, iterating through each datapoint and assigning them to the nearest centroid until all datapoints have been sorted. The new centroids are then calculated by taking the average datapoint from each cluster. The difference between the old and new centroids is then checked against the convergence criteria and if it is not met the loop continues until the criteria is met or the maximum steps is met.

Initialisation algorithms:

One of the key factors when evaluating the quality of the produced centroids was how well chosen the centroids were. For example, if the centre points chosen were right next to one another there would be a lot of overlap between clusters and therefore a poor representation of the nature is given. To counteract this, I implemented three algorithms for selecting centroids and compared how well they worked for selecting informative queries.

Random Initialisation:

This is the simplest algorithm which simply selects random points from the datapoints to allocate centroids. This has the potential to pick the optimal centroids but is equally likely to pick the worst centroids so despite it being the most effective in terms of time complexity it lacks consistency of the quality of selection which other algorithms offer.

KMeans++ Initialisation:

KMeans++ [7] uses an initially random choice and then picks the next centroid based on the probability proportional to the square of the distance from the nearest centroid. This ensures that centroids are selected far away from previous ones. This method still has elements of random initialisation which can hurt its performance, and the time complexity makes it slower than random due to having to calculate the distance from each datapoint to its nearest centroid.

Minmax Initialisation

This method selects the first point randomly and then for each centroid afterwards it will alternate between the largest and smallest magnitudes of the datapoints. This method holds the most promise since it encourages high diversity of samples for selection which is what made the uniform sampling method so effective in the baselines.

Repeat Initialisation:

A key hyperparameter to my algorithm was the number of times clustering would be attempted before using the best formed clusters. Due to the random starting point in all initialisation algorithms implementing a simple loop to repeat the clustering process and assess which cluster was best helped in improving the consistency and overall performance of the algorithm as it would greatly decrease the chances that a bad starting centroid was chosen. The clear drawback of this approach is the computational cost of clustering hundreds of datapoints with thousands of features multiple times.

By employing repeated initialisation, a metric had to be used to establish which was the best clustering attempt. To ensure the most effective evaluation of clustering attempts a combination of three metrics was used: Silhouette score [8], Davies-Bouldin score [9] and Calinski-Harabasz score. Where the Silhouette score measures how similar a datapoint is to its own cluster compared to other clusters, Davies-Bouldin measures the average similarity between each cluster and its most similar one and Calinski-Harabasz [10] measures the ratio of between cluster dispersion to within cluster dispersion. After combining these scores this served as a strong performance metric for measuring how well the datapoints were clustered. Another method explored in conjunction with the repeat initialization technique was to randomly select a initialisation method to be used for each attempt as it would provide a strong variety of centroids and would be able to tackle different distributions of datapoints however when applied practically this method proved to be disappointing in final performance as instead of using one method multiple times so it is optimized it would use multiple and find suboptimal starting points leading to worse performance than if just one initialisation method was used. The use of randomly selecting initialisation algorithms also meant the number of times the clustering was attempted had to be increased so that they could all be used with a strong starting point.

Selection from cluster algorithms:

Once the population had been split into clusters an algorithm would then be used to select the samples from each cluster. The selection algorithm should ensure that the total selected samples does not exceed or fall short of the required number of samples. The method in which the samples are selected from the cluster can have a large impact on the episode return because clusters can still be very large so the datapoints can be very different despite being assigned to the same class.

Random Selection:

Using random selection to select samples from each cluster was the first and simplest method implemented. Despite this method being extremely computationally cheap if there was only one cluster it would perform identically to the uniform sampling method originally used so it has the potential to be too much like the original PEBBLE algorithm and not make any improvement.

Distance Based Selection:

Distance based selection would select those datapoints closest to the centroid of each cluster as the final sample. This method produces a more accurate representation of the cluster however provides very little diversity to the samples as they are selected by being close to one another. When there are fewer cluster centroids this method performs poorly because the samples selected are too similar.

Minmax Selection:

Like the logic of the minmax initialisation algorithm this selection method randomly picks the first point and then iteratively selects the furthest different point from the previous choice until the number of samples is met. This method maximises the diversity of samples which helps the

model learn from a range of datapoints however it encourages the selection of outliers which could harm the learning process.

Preprocessing of data:

A key process before the clustering takes place is the preprocessing of the data. This is used to improve the computational efficiency of the clustering, improve the end classification of the datapoints and remove outliers from the population which aides in stopping bad centroid selection and supplying the human teacher with queries that are not helpful to the training of the model.

Flatten the Data:

This process takes the given data points and removes dimensions so that the data becomes a one-dimensional vector. This helps the clustering algorithms perform as well as possible. Flattening the data is also helpful when calculating the distance between datapoints as the distance metrics used (Euclidean, Manhattan, Cosine) require the datapoints to be in vector form.

To flatten the data functions from NumPy 1.23.5 are used. The original data is converted to a NumPy array where the function reshape() is then applied so that the number of values is not changed but the number of features is reduced to one.

Standardisation of Data:

Standardisation of the data puts all features on the same scale regardless of their original ranges. For this preprocessing z-score standardisation is used which takes each datapoint subtracts the mean of all data points and then divides by the standard deviation of all data points to produce the final value. By standardising the data there is better convergence for the KMeans algorithm because the initial centroids are more balanced, and the clusters are more meaningful because extreme datapoint values don't have as much influence. Other methods of standardisation include 0-1 scaling however one extreme value can throw every other datapoint so that the difference between them is minimal, so the z-score method was more suitable and reliable in the context of clustering.

The z-score standardisation formula is given by:

$$z = \frac{x - \mu}{\sigma}$$

Figure 10: Formula for z-score standardisation where x – datapoint, μ - mean of dataset, σ = standard deviation of dataset

PCA Feature Reduction:

The most significant stage of the data preprocessing in preparation for clustering was the inclusion of principal component analysis (PCA) feature reduction [11]. I explored two types of feature reduction, using a fixed feature value and explained variance method. PCA works by reducing the number of features in data whilst maintaining the relationships between the datapoints so that the calculated distances between datapoints are more efficient and more

effective. One of the key benefits of PCA relevant to this project was its improvements upon the computational efficiency of the clustering since it is done over hundreds of datapoints each with thousands of features. The fixed feature version of PCA was first used and this is where a set feature amount was hardcoded before training, this value had to be tuned over multiple runs and therefore when one value was suitable for a certain seed it may be inappropriate in another. Whilst this method allowed me to have more control over the shape of the clustered data and therefore the computational speed of the algorithm it was ineffective to be used as it needed to be tuned frequently to maximise the final performance. The second method was the explained variance method where a variance threshold is given and in the preprocessing stage of the algorithm the number of features is reduced to be as low as possible whilst maintaining the variance within a given threshold parameter. This method allowed the number of features to be optimised by using as little as possible features whilst retaining the key properties of the data. This method proved to be much more effective as it could adapt the number of features to the context however it does mean that the computational efficiency of the algorithm can change drastically over multiple seeds.

The PCA process is completed using functions from NumPy library. Once the data has been flattened and standardised using z-score a covariance matrix is calculated which tells you how many features vary with respect to each other for all the datapoints. Eigenvectors and eigenvalues are then calculated which represent the directions of maximum variance and how much variance is captured by each component. The array storing each eigenvalue is then sorted in descending order so that the top components are the most meaningful directions in the data. The number of features is then selected based on getting the feature value as small as possible whilst maintaining the thresholds data variance, in my case a threshold of 95% was chosen. The top features eigenvectors are selected and projects the standardised data onto them. The data has now been reduced to the optimal number of features to maintain 95% variance so is much more suitable for clustering.

Distance Metrics:

A major influence for the selection of datapoints for each cluster and therefore the quality of the clusters overall is the distance metric used to measure which datapoint should be matched with each centroid. In summary, the distance metric has a massive impact on how the clusters are formed and therefore the overall quality of the queries presented to the human teacher.

Euclidean Distance

$$d(A, B) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_n - b_n)^2}$$

Figure 11: Formula for Euclidean Distance

Euclidean calculates the straight-line distance between two points. It calculates the ‘as the bird flies’ distance.

Manhattan Distance

$$d(A, B) = |a_1 - b_1| + |a_2 - b_2| + \cdots + |a_n - b_n|$$

Figure 12: Formula for Manhattan distance

The Manhattan calculates the total distance along the axes between two points.

Cosine Similarity

$$\text{Cosine Similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

Figure 13: Formula for Cosine Similarity

The Cosine similarity measures how similar the direction of the two vectors are, regardless of their magnitude.

3.6. Reward Model Ensemble

PEBBLE [1] uses an ensemble of neural networks for the reward model. An ensemble is a collection of multiple neural networks that work together to produce a more accurate result. In the case of PEBBLE this ensemble is trained as a reward model and the average result between each member of the ensemble is used as the final output. As well as improved performance an ensemble also increases the robustness and flexibility compared to a single neural network however an increase in models means an increase in implementation complexity and an increase in training time. The current PEBBLE implementation uses three identical neural networks as the ensemble members.

Current Ensemble:

The current ensemble used by the PEBBLE algorithm is comprised of three identical neural networks. The networks are made up of a fully connected layer and a LeakyReLU activation function. The loss function and activation functions for each network in the ensemble is also identical being cross entropy loss and the Adam optimiser. This immediately stood out as a point of potential improvement due to the simplicity of the current implementation, improving the ensemble by using multiple different models or even changing the loss and activation functions for each ensemble member was an area I focussed on to find increases in performance of the algorithm.

Loss Functions:

The loss function for a neural network quantifies how well a neural network is performing and the network then will try to minimise it making it extremely influential in the training of a neural network. The current algorithm uses the same loss function for every member of the ensemble, Cross entropy loss. A simple change like changing the loss function so that each model uses a unique loss function could improve the performance of the ensemble as the shortcomings of a single loss function are not present in every ensemble member.

Activation Functions:

In a neural network activation functions are mathematical functions which are applied to each neurons output, their main job is to introduce non-linearity, which allows the network to learn complex patterns. The current function being used by PEBBLE is leaky rectified linear unit (Leaky ReLU).

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$

Figure 14: Function for Leaky ReLU activation function.

Whilst this function is effective and simple it can be improved upon by employing a more advanced function.

The functions I have experimented with are the Parametric rectified linear unit (PReLU) [12] and Swish [13] activation functions. The PReLU activation function is an improvement and more advanced variation of the ReLU activation function and primarily addresses the dying neuron problem. This is where if the neuron output gets 'stuck' as a negative value the activation function will always produce a value of 0 and therefore the neuron stops learning and 'dies'. The function used by PReLU is:

$$f(y_i) = \begin{cases} y_i & \text{if } y_i \geq 0 \\ a_i y_i & \text{if } y_i \leq 0 \end{cases}$$

Figure 15: Function for PReLU activation function. Where α is a pre-defined small positive constant.

The Swish activation function is a relatively new activation function and my reason for selecting it to test was this recency factor. Like the PReLU swish also avoids the dying neuron problem but it is also smooth, meaning its derivative doesn't have sudden changes which aides in modelling complex relationships.

The Swish function is given by:

$$f(x) = x \cdot \sigma(x) = \frac{x}{1 + e^{-x}}$$

Figure 16: Function for Swish activation function. Where σ is the Sigmoid function.

Using Different Architectures:

Since the current algorithm uses three identical neural network architectures, they will be trained very similarly despite being trained using different data. By adding two additional architectures this could create a more diverse ensemble which can then produce better results.

More Advanced Architectures:

The current algorithm uses three identical neural network architectures which are all simple with only fully connected layers and activation functions repeated. By making this single network more complex but keeping three of the same this could improve the performance of the ensemble because of the ability to prevent underfitting and overfitting. The layers I looked at implementing were batch normalisation layers, dropout layers and adding weight decay. The addition of these layers work at reducing the overfitting of the model and therefore encouraging learning. The downside of making the neural network architecture more advanced is it increases the training time.

Final Ensemble Structure:

The final implementation of the ensemble uses three unique neural network architectures which maintain simplicity whilst offering a wider variety of layers to improve the stability and diversity of the networks without increasing the training complexity a substantial amount. The method of combing the ensemble members outputs was kept the same as originally used in PEBBLE which is averaging.

```
def gen_net(in_size=1, out_size=1, H=128, n_layers=3, activation='tanh'):
    net = []

    for i in range(n_layers):
        net.append(nn.utils.weight_norm(nn.Linear(in_size, H)))
        net.append(nn.PReLU())
        in_size = H
    net.append(nn.Linear(in_size, out_size))

    if activation == 'tanh':
        net.append(nn.Tanh())
    elif activation == 'sig':
        net.append(nn.Sigmoid())
    else:
        net.append(nn.ReLU())

    return net


def gen_net2(in_size=1, out_size=1, H=128, n_layers=3, activation='sig'):
    net = []
    # Inclusion of Dropout layer (0.5 prob)
    for i in range(n_layers):
        net.append(nn.Linear(in_size, H))
        net.append(nn.Dropout(0.2))
        in_size = H
    net.append(nn.Linear(in_size, out_size))

    if activation == 'tanh':
        net.append(nn.Tanh())
    elif activation == 'sig':
        net.append(nn.Sigmoid())
    else:
        net.append(nn.ReLU())

    return net

def gen_net3(in_size=1, out_size=1, H=128, n_layers=3, activation='sig'):
    net = []
    # 3 times increase in number of layers
    for i in range(n_layers*3):
        net.append(nn.Linear(in_size, H))
        in_size = H
    net.append(nn.Linear(in_size, out_size))

    if activation == 'tanh':
        net.append(nn.Tanh())
    elif activation == 'sig':
        net.append(nn.Sigmoid())
    else:
        net.append(nn.ReLU())

    return net
```

Figure 17: Python implementation of three neural networks using torch.nn module from PyTorch [14] *which are then combined and used in an ensemble*

Overall, whilst the ensemble presents an interesting method of altering the code as the complexity of the ensemble increases, through additional layers and multiple model architectures the likelihood of the model underfitting also increases. The challenge with addressing underfitting is that it requires more training which goes against the primary aim of this project which is to improve the sample and feedback efficiency of reinforcement learning. For this reason, only minor changes were made to the ensemble and improving the selection of informative queries was selected as the optimal path to find improvements.

3.7. Unsupervised Pre-training

A major feature added by PEBBLE [1] compared to other state-of-the-art reinforcement learning with human feedback was an unsupervised pre-training stage where the agent would explore their environment to build experience before the human participant is included for the main training stage. During unsupervised pre-training the agent explores using intrinsic motivation so that the experiences collected improve the queries that can be passed to the teacher and therefore the human teacher can be used to provide higher quality feedback. The intrinsic reward the agent aims to maximise in PEBBLE are prediction errors, count-based state novelty, mutual information and state entropy.

A possible area for improvement could be to alter the unsupervised pre-training algorithm to collect more experience for the agent, which would give the model a larger population to select queries from to gather feedback from the human, this would then allow the training progress to be quicker at the beginning

After analysing the performance of the current unsupervised pre-training compared to alternate methods the potential for improvement is lower than expected so time was spent on more promising methods of improvement such as the algorithms for selecting informative queries.

3.8. Methods of Obtaining Human Feedback

A powerful method of finding improvement in PEBBLE [1] could be to change the way human feedback is obtained from the human teacher. Currently, the human gives a preference between two trajectories, and this is used to train the reward model. However, more advanced methods could be implemented that allow more information to be gathered despite querying the human participant the same number of times.

A proposed method was an inference-based system where if the human provided feedback between two trajectories and the preferred choice is then compared against another trajectory which the human then selects to be favoured then it can be inferred that the final chosen trajectory is also going to be preferred over the first trajectory. By using this knowledge, the model can gather a much more comprehensive set of training data without increasing the amount of feedback the human provides.

Whilst this method showed promise it would require an entire new databank of human feedback which would not only be extremely time consuming to gather and implement within the current PEBBLE scripts, but it would also require additional ethical compliance. This change was also not guaranteed to find any improvement so methods of selecting informative queries was prioritised due to the promising nature.

3.9. Final Algorithm

The best performing algorithm included the KMeans clustering sampling algorithm with 1000 maximum convergence steps and 1e-5 convergence criteria with minmax initialisation algorithm, the minmax selection algorithm, preprocessing of data with PCA explained variance method, five clustering initialisation attempts, combined score metric to evaluate the clustering, Euclidean distance and 2 centroids. Which can be simplified to Diverse Minmax KMeans Sampling.

The final ensemble structure is comprised of three unique neural network architectures which are based on the original architecture used by PEBBLE but with a distinct feature added, the first network included weight normalised fully connected layer which improves training stability and speed over a standard fully connected layer. The second network included a dropout layer which reduces overfitting of the model. The third network triples the size of the original architecture to reduce underfitting. These changes allow an increase in ensemble diversity and a reduction in overfitting whilst only marginally increasing the training complexity.

The final algorithm replaces the uniform sampling method originally used in PEBBLE with my Diverse Minmax KMeans Sampling method and updates the ensemble to my advanced version.

4. Experiments

The experiments have been designed to directly compare the performance of the original PEBBLE [1] algorithm with my proposed changes so that the differences can be seen and analysed. The experiments will demonstrate different methods and how these different approaches produce different results.

4.1. Setup

To ensure that the experiments being run are consistent the algorithm run on the walker and quadruped environments would use identical hyperparameters for each run and therefore the only changed feature would be the algorithm and the changes I had implemented. Unless otherwise specified, the walker and quadruped environments were run with the following hyperparameters:

Walker:

```
python ~/PEBBLE/BPref/train_PEBBLE.py  env=walker_walk seed=$seed agent.params.actor_lr=0.0005
agent.params.critic_lr=0.0005 gradient_update=1 activation=tanh num_unsup_steps=9000
```

```
num_train_steps=500000 num_interact=20000 max_feedback=1000 reward_batch=100 reward_update=50
feed_type=$1 teacher_beta=-1 teacher_gamma=1 teacher_eps_mistake=0 teacher_eps_skip=0
teacher_eps_equal=0
```

Quadruped:

```
  python ~/PEBBLE/BPref/train_PEBBLE.py  env=quadruped_walk seed=$seed agent.params.actor_lr=0.0001
agent.params.critic_lr=0.0001 gradient_update=1 activation=tanh num_unsup_steps=9000
num_train_steps=1000000 num_interact=30000 max_feedback=1000 reward_batch=100 reward_update=50
feed_type=$1 teacher_beta=-1 teacher_gamma=1 teacher_eps_mistake=0 teacher_eps_skip=0
teacher_eps_equal=0
```

As the experiments progress only the max_feedback parameter was changed to test how my algorithm would perform with human feedback reduced. This is outlined in the experiments where this took place. The other hyperparameters detail the number of steps unsupervised pre-training should have, the number of training steps for the human in the loop learning, the quality of teacher in terms of the feedback they will provide and hyperparameters for the ensemble neural networks like activation functions.

For all my experiments the oracle teacher will be used which is a simulated human that only provides optimal feedback for learning and never makes any mistakes.

The algorithm the experiments are run on is described in Final Algorithm and the experiments below show the different sampling methods implemented and how they affect the performance.

4.2. Seeds

Seeds are used to make the algorithmic results as reproducible as possible. Seeds ensure the results of a machine learning task are robust and not due to favourable randomness. By using the same seed, a random number generator will produce the same sequence of numbers making the code deterministic.

Seeds mean the code can be tested using different algorithms and the performance will not be affected by the random components. For example, the initialisation of weights can have a huge impact but by using the same seed they will always initialise as the same.

The use of seeds is vital to this project as it allowed me to properly compare algorithmic changes and not have to consider the random values that cannot be controlled by have massive implications on the algorithm.

4.3. Selecting Informative Queries using Diverse Minmax KMeans Sampling

The Diverse Minmax KMeans sampling method is the best performing clustering algorithm to replace the uniform sampling method. The motivation behind this algorithm was to maximise the diversity of the samples, this meant using both minmax initialisation and minmax selection algorithms with two cluster centroids. Preprocessing included PCA with explained variance method to ensure that the optimal number of features was used and z-score standardisation. Hyperparameters were maximum convergence steps as 1000, convergence criteria as 1e-5 and the number of clustering steps as 5. These values were all set using past experiments and optimising them in terms of maximising episode reward whilst keeping the algorithm as computationally efficient as possible. Euclidean distance was used and a combined score of all three metrics was used to evaluate the quality of the clustering after each attempt.

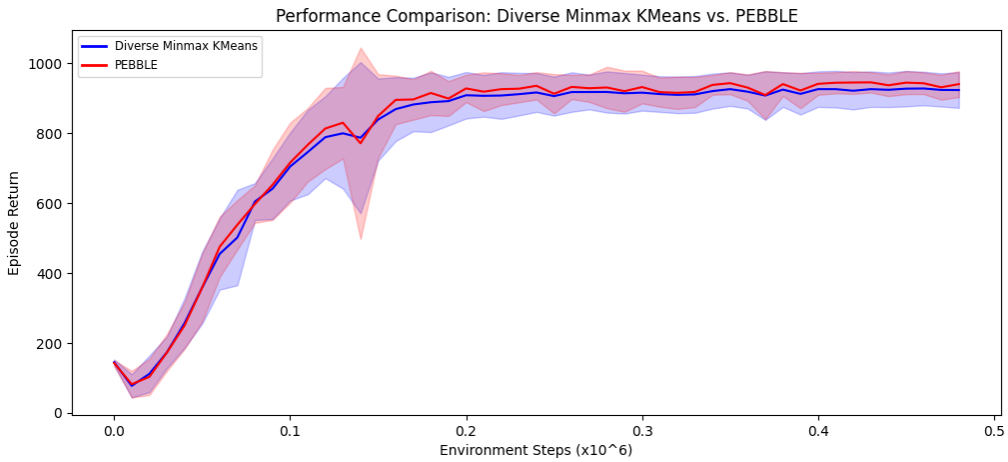


Figure 18: Walker environment using Diverse Minmax KMeans sampling for selecting informative queries with 1000 feedback and oracle teacher. The solid line represents the mean and the shaded region represents the standard deviation.

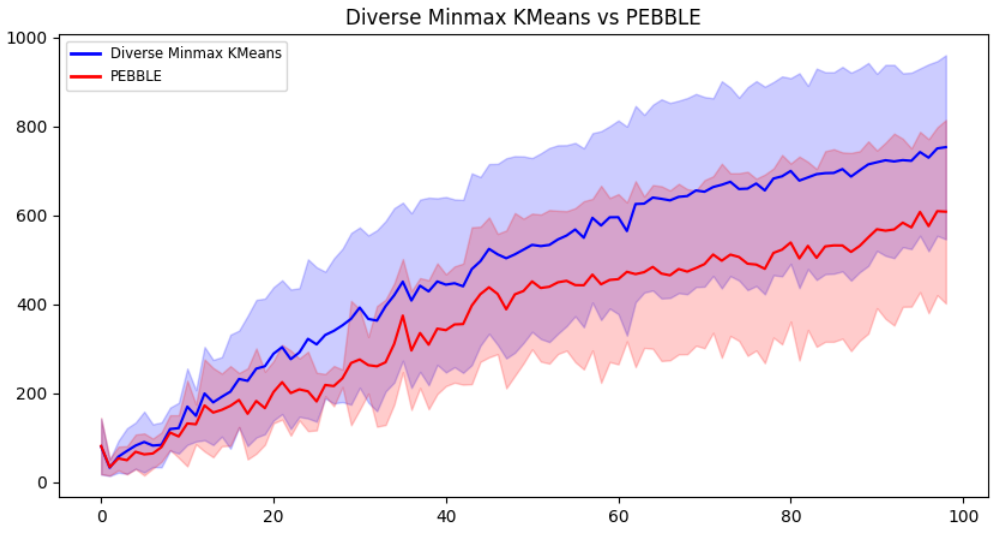


Figure 19: Quadruped environment using Diverse Minmax KMeans sampling for selecting informative queries with 1000 feedback and oracle teacher. The solid line represents the mean and the shaded region represents the standard deviation.

In figure 18 we observe that my proposed algorithm underperforms compared to the PEBBLE [1] baseline in terms of a higher average episode return and a superior standard deviation meaning the results are more consistent. We see that the PEBBLE baseline achieves a higher maximum episode return compared to my changes but only very marginally. The similarity between the growth of both the original PEBBLE algorithm and my proposed changes emphasises how for simpler environments the application of reinforcement learning with human feedback is very effective. The standard deviation of my changes is slightly worse than the original PEBBLE which means there is more variance across the seeds. The key reason for using the walker environment to test my changes was to ensure the performance of the new algorithm is strong in the simpler environments and the more complex ones.

The reason for the difference in standard deviation in figure 18 is since my method selects a much more diverse range of samples for the human participant than the original PEBBLE algorithm which can have a negative effect on some seeds. The mean performance for my algorithm and the original PEBBLE algorithm is very similar and the growth pattern of each is also similar. Due to the simplicity of the environment the sample choice does not have as much of an impact as in complex environments.

In figure 19 we see that my proposed algorithm outperforms the original PEBBLE [1] algorithm in terms of the average episode return and the mean reward gradient is steeper. The standard deviation is very similar to the PEBBLE baseline in terms of seed consistency however the lowest performing seed for my algorithm performs similarly to the average for the PEBBLE baseline which emphasises the increase in performance found by my changes.
When analysing the two different algorithm versions we see that my algorithm achieves a higher maximum return than the original PEBBLE algorithm almost immediately and maintains this over the rest of the episodes. The lack of change in standard deviation is disappointing as by removing the random sampling method I expected the results to be more consistent over the different seeds however the improvement in mean episode reward makes up for this.

The reason my proposed algorithm performs better than the PEBBLE baseline in figure 19 is due to the increased diversity that the clustering sampling method offers compared to the uniform method whilst also representing the less extreme and average datapoints in the population. The increase in sample diversity means that the human feedback given to the model for training based off these samples is more impactful and therefore allows the algorithm to achieve an increase in episode return. This balance of training the model with extreme datapoints as well as the mean datapoints allows the model to learn quicker than just the randomly sampled algorithm implemented by the PEBBLE baseline and reach a higher peak. This version of the clustering sampling method, with minmax initialisation and minmax selection algorithm is custom built to maximise the Euclidean distance between samples once they have been sorted into clusters and therefore ensuring diversity is maximised.

Diverse Minmax KMeans Sampling with 500 Feedback

After achieving an improvement upon the original PEBBLE [1] algorithm using Diverse Minmax KMeans sampling with 1000 feedback I then reduced the max feedback hyperparameter to 500 to

analyse the effect that reducing the amount of human feedback the model receives would affect the performance.

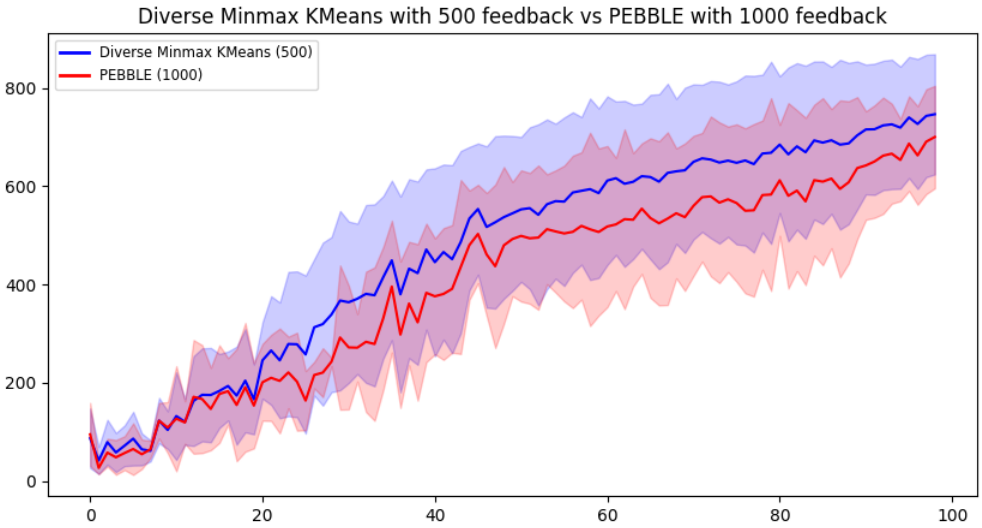


Figure 20: Diverse Minmax KMeans using 500 max feedback with oracle teacher and PEBBLE using 1000 feedback with oracle teacher. The solid line represents the mean and the shaded region represents the standard deviation.

As seen in figure 20 we observe increased performance from my proposed algorithm over PEBBLE even when the amount of feedback it receives is halved. This is extremely encouraging to see as this highlights how the sample efficiency of the PEBBLE algorithm has been improved due to my changes. This means that the algorithm now selects queries to present to the human in a much more intelligent manner which means that the human feedback is more helpful and improves the learning of the algorithm both in terms of a faster growth rate and a higher overall average episode reward.

Overall, the Diverse Minmax KMeans method improves upon the performance of PEBBLE and can not only match the high performance in simpler environments like walker but also improve upon the performance in complex environments like quadruped walk. Most significantly, Diverse Minmax KMeans can outperform the PEBBLE baseline when given half the amount of human feedback.

4.4. Selecting Informative Queries using Clustering with Fixed Centroids

This was the first iteration of clustering sampling and uses fixed centroids. This means once the clusters have been formed, the datapoints it contains will stay the same and their centroids will not be updated. The preprocessing of data for this sampling method uses z-score standardisation, data flattening and PCA feature reduction with the number of features set to 50. It uses two cluster centroids and the KMeans++ initialisation algorithm. Five attempts are used to select optimal cluster centroids and the final sample selection algorithm from the

clusters is random selection. The Manhattan distance is used to assign datapoints to clusters and to evaluate the quality of the clustering the Silhouette score metric is used.

The motivation behind these hyperparameters was to implement a clustering sampling method that was not too computationally expensive compared to the original uniform sampling method whilst also trying to find an improvement in the episode return. The exclusion of the centroids updating and not including convergence was implemented for this purpose. The KMeans++ initialisation algorithm was chosen because of the significant improvement it offered over a random centroid selection and because of its high performance in other tasks. The number of clustering attempts was set to 5 as from my own experimentation there was never any poor centroid selection when given at least 5 unique starting points, so this helped to ensure the centroids were well chosen which is especially important in this algorithm as they do not change once selected. The random selection from cluster groups was implemented due to its similarity to the uniform random sampling so that the good performance of the baseline was not just completely ignored. The uniform random sampling provided a wide range of samples just not in a consistent manner so by first splitting the data into two clusters and then randomly sampling I hoped this algorithm would help to improve the performance and coverage of the population. The preprocessing step of including PCA feature reduction was hard coded to 50 features to significantly improve the computational efficiency of the algorithm as without PCA the number of features is 4500.

Figure 21: Walker environment with Clustering with fixed centroids sampling method for selecting informative queries with 1000 feedback and the oracle teacher. The solid line represents the mean and the shaded region represents the standard deviation.

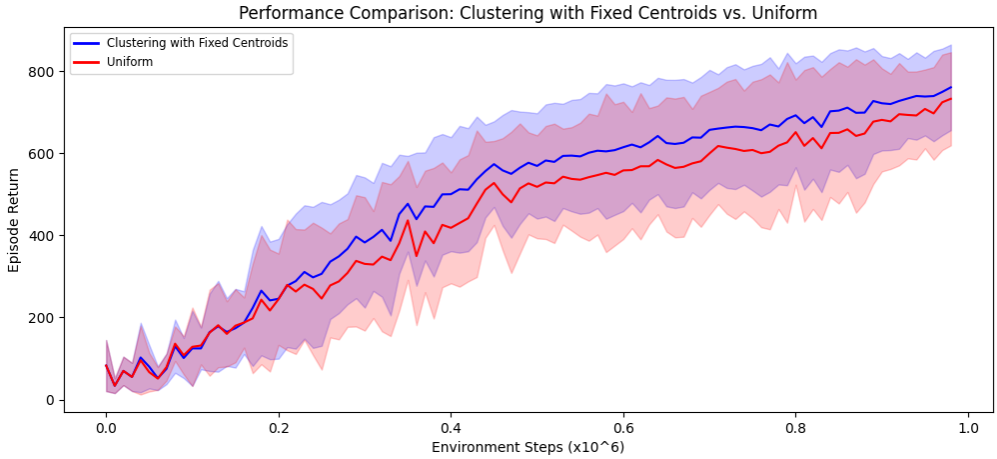


Figure 22: Quadruped environment with Clustering with fixed centroids sampling method for selecting informative queries with 1000 feedback and the oracle teacher. The solid line represents the mean and the shaded region represents the standard deviation.

In figure 21, we observe that clustering with fixed centroids performs extremely similarly to the PEBBLE [1] baseline. The main difference being in the early learning stages PEBBLE performs slightly better in terms of a higher average episode reward and a better standard deviation. Both methods also have the same spike in reward where across the seeds a very high performance is found as well as a very poor performance. Overall, this figure emphasises how in simpler environments the clustering with fixed centroids method performs almost identically to the high performing uniform sampling method used by PEBBLE.

In figure 22, we see that the changes do produce an improvement over the baseline but percentage increase of 3.8% is underwhelming. The changed algorithm achieves a more consistent growth trajectory and achieves performance milestones earlier than the base PEBBLE which is slightly slower in the initial learning phase. The standard deviation shows that the worst performing seed for the changes achieves higher episode return than the baseline and the highest performing seed for the changes is also higher than the baseline. The range of the standard deviation is near identical and so the consistency of the algorithm has not changed.

When evaluating these changes and the effects they had on the algorithms performance there was an underwhelming difference in results. The main reason for these results being so like the baseline, which uses a unform sampling method, is because of the simple method clusters are formed and then the random sample selection algorithm being used. By using the same uniform algorithm to select samples once split into the clusters this also creates very similar logic which is reflected in the results. The biggest area that should be changed is the exclusion of updating the centroids in the algorithm.

4.5. Selecting Informative Queries using Extreme Disagreement sampling

Extreme disagreement-based sampling were the first experiments run to attempt to find improvements in the PEBBLE algorithm.

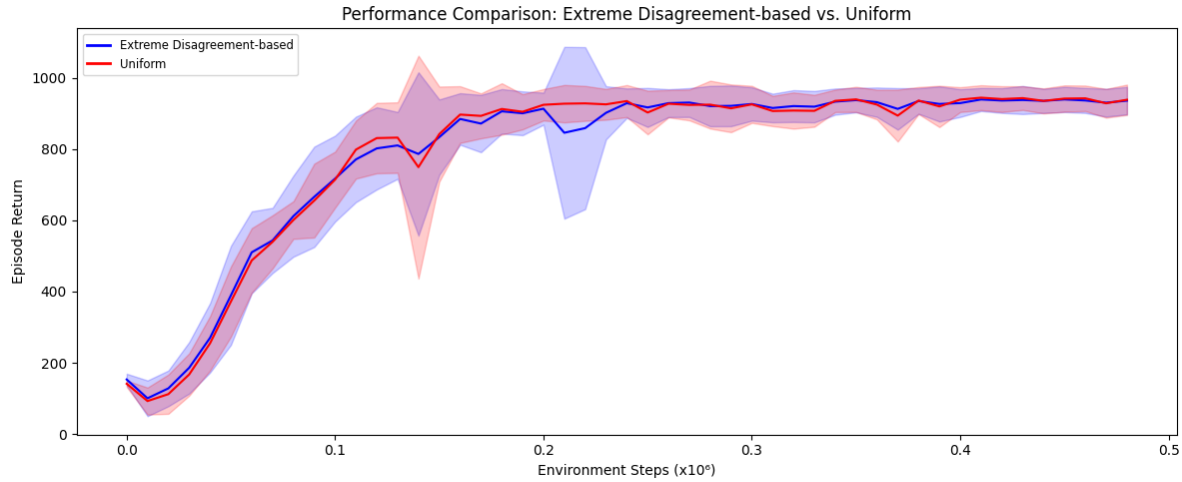


Figure 23: Walker environment with Extreme disagreement sampling method for selecting informative queries with 1000 feedback and the oracle teacher. The solid line represents the mean and the shaded region represents the standard deviation.

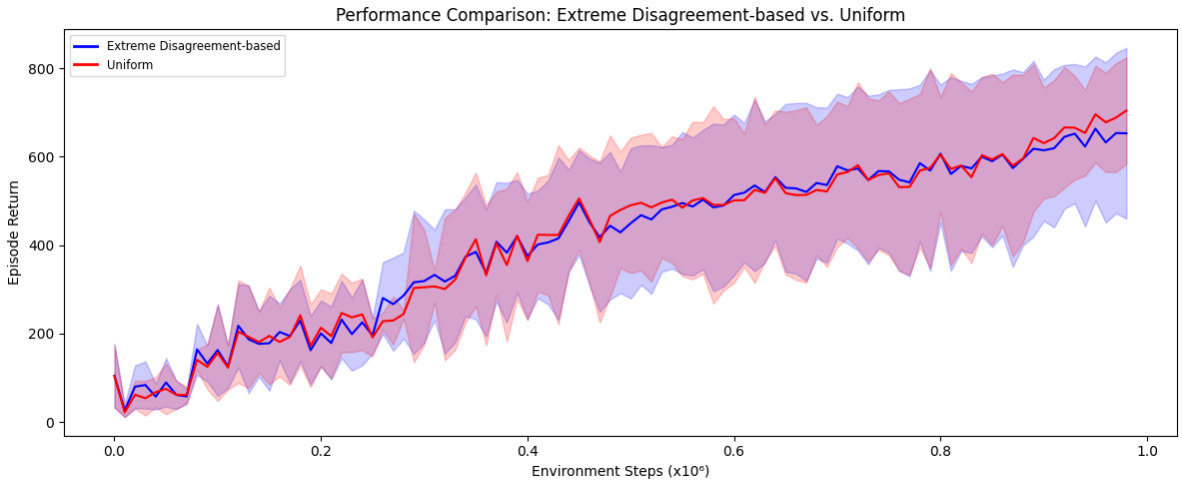


Figure 24: Quadruped environment with Extreme disagreement sampling method for selecting informative queries with 1000 feedback and the oracle teacher. The solid line represents the mean and the shaded region represents the standard deviation.

In figure 23 we observe the performance of extreme disagreement-based sampling on the Walker environment where we see that it performs very similarly to the PEBBLE [1] baseline. The mean episode return of both algorithms is almost identical aside from a few fluctuation areas. The main difference between the two methods is the standard deviation where extreme disagreement-based performs slightly worse in the first half of the environment steps. The key difference in the walker environment is the additional area of a large range between the best performing and worst performing seed which also cause the mean to dip slightly. This

highlights how extreme disagreement is more inconsistent than uniform even in simple environments.

As seen in figure 24 above the proposed changes of extreme disagreement-based sampling do not provide an improvement on the PEBBLE algorithm. Whilst this set of results is disappointing it is not unsurprising given the primary purpose of this sampling method was to understand more about the relationships between datapoints.

In the quadruped environment the maximum episode return achieved by the base PEBBLE is 704.31 whereas the maximum episode return achieved by extreme disagreement-based sampling method is 663.31 which is a percentage decrease of 5.82%. We also observe that my changes have a better initial performance but falls short in the stabilisation phase as the episode reward climbs. The baseline PEBBLE has a more consistent growth trend compared to the changed algorithm. This highlights the disappointing performance for extreme disagreement-based sampling.

The reason behind the results being so similar is due to the uncertainty of samples being similar for all datapoints, this explains why the samples chosen to perform so similarly to the samples chosen randomly using the uniform method. Furthermore, the reason behind the performance being worse than uniform in some episodes is because the challenging or confusing samples are being chosen, and the model is not able to properly learn from them. To improve upon this sampling method, I would adapt it so that it doesn't only select the most extreme disagreement datapoints but also some average and low disagreement datapoints to allow the model to learn from simpler datapoints but also gain experience from the more challenging points.

Overall, we see that both sampling methods follow a similar learning pattern in the mean reward. We also see that the standard deviation is very similar and is especially noticeable for certain environment steps when both change erratically in a similar way.

5. Discussion

The results presented show an increase in the sample and feedback efficiency in my algorithm compared to the original algorithm presented in the PEBBLE paper [1]. In summary, my new algorithm improves the episodic return, and not only does it achieve improved results with the same amount of feedback, but it also improves the results when given less feedback. This means that not as much human participation is required for the algorithm to perform well which means it is better made for more complex tasks and tasks where a short training process is required.

These results matter because of how they allow the model to function at a similar level of performance despite being given less information from the human participant. If reinforcement learning with human feedback models are to be used in the real world it is not practical to have a human sat for hours providing feedback to a model that will only complete a very simple task. Therefore, improving current state of the art systems to perform better when given less feedback brings reinforcement learning with human feedback a step closer to being used in the real world. These results do not inform us as to the suitability for when the human teacher

provides poor feedback. The current hyperparameters in place assume perfect feedback from the human teacher who will also be aiming to maximise the agents learning, however in real world scenarios it is unlikely that the human teacher will be perfect in every piece of feedback provided. Scenarios where the human makes a mistake or misunderstands the trajectories being presented to them is increasingly likely as reinforcement learning with human feedback is used in more complex environments to complete more harder tasks. My recommendations for future work are to enable the model to learn despite poor feedback being given from the human participant. Whether this is to know when to ignore the human feedback as it is most likely wrong or to put more weighting on feedback that is certainly correct. Overall, it is vital to have reinforcement learning with human feedback models be effective even with small amounts of human feedback which this project helps to achieve however the quality of the human teacher and the feedback they provide still has a large influence on the training process.

6. Project Ethics

This project will use synthetic human data due to the time restraints and complexity involved with garnering real human feedback with each execution of the algorithm. For this reason, the Project Ethics & Human Participants section does not apply to this project.

7. Conclusion and Future Work

In conclusion, the project was a success and the results I achieved prove that reinforcement learning with human feedback is one step closer to being deployed to complete real world tasks. I have confidence that the improvement found by changing the sampling method for informative queries and making changes to the ensemble structure are only the start of the optimisation that can be found within PEBBLE [1]. Over the course of this project I have achieved all of my essential requirements by researching reinforcement learning with human feedback papers (Related Work), being able to set up and reproduce the results achieved in the PEBBLE paper (PEBBLE baseline), implementing my own improved methods (Design and Implementation) and finally analysing how my changes improved the algorithm in terms of achieving a higher performance when given the same amount of human feedback as well as when given less human feedback (Experiments). Overall, this achieves my goal of improving the sample and feedback efficiency of reinforcement learning with human feedback.

For future work on this project I would focus on making my improved algorithm more robust for human teachers who do not provide perfect feedback. In this project, the only teacher used was oracle which always selected the trajectory that was best for the training of the model, in the real world this won't be the case so the model should be able to deal with suboptimal human feedback. The teachers I would focus on in further work would be the equal and skip teacher. This is when the human participant cannot determine the difference between the trajectories, or they skip the comparison and give no feedback. These would be the most common scenarios that could occur in the real world and therefore the algorithm should be able to handle them accordingly. To address these issues, I would attempt to implement a system that could use the reward model that is part way through training to try and provide a

prediction as to which would be the better trajectory, however this could result in an erroneous piece of feedback being given which could be worse than no feedback.

A second area I would look to continue work on is the method in which human feedback is collected. The current comparison-based system provides the human with two choices, the user selects the preferred direction for learning and the next two clips are shown. Whilst this method is quick and simple for the human it does not provide much information for the agent. An improved method could be a king of the hill type implementation where the trajectories aer compared against one another until only one remains and it can be assumed that this trajectory is favourable over every other. This method would allow significantly more information to be gained from the same number of trajectory comparisons. This method could vastly reduce the amount of feedback required from the user. However, if the user makes a mistake in this format the consequences are much more profound.

8. BCS Criteria and Self-Reflection

An ability to apply practical and analytical skills gained during the degree programme – achieved in the Design and Implementation section where machine learning sampling techniques were learned and developed so they could be implemented into a larger and more complex algorithm. This included neural network knowledge and overall knowledge of reinforcement learning to understand why changes made had the result they did and whether they could be changed further to produce even better algorithmic performance.

Innovation and/or creativity – achieved in the Design and Implementation section where ideas were developed and implemented so that they were suitable for the algorithm and worked effectively. If an implementation worsened performance, it would be analysed, and additional changes would be made to attempt to address the lacklustre performance.

Synthesis of information, ideas and practices to provide a quality solution with an evaluation of that solution – Achieved in the Design and Implementation section and the where ideas were implemented and then the results of these ideas are seen in the Experiments section. The experiments section also covers an analysis of why the changes implemented had the effect that they did, whether that was good or bad.

That your project meets a real need in a wider context – Achieved in the Design and Implementation section and Experiments results by progressing reinforcement learning with human feedback, so it becomes more suitable for real world use by reducing the amount of human interaction required.

An ability to self-manage a significant piece of work

Over the course of this project, I encountered many aspects which required rigorous planning and time management. One of the most frequently experienced challenges was the long training times required each time I wanted to test the functionality of the changes I had made to the code. It took twelve hours to train the model on two seeds, so to create graphs for analysis on at least five seeds it took around 30 hours. This meant I had to develop a strict plan

for when to start the training so that as little time as possible was wated and I could still make progress on the project during this training time. The best regiment was starting the training process so it would train overnight and then the next morning analysing the results and starting the training process again so it would be completed by the evening so the cycle could start over again. I also ensured that when the model was being trained during the day I had planned other tasks to be completed such as implementing a new section of code or working on a written task. The only problem with this routine was if there was an error discovered in the morning which took too long to fix so the model hadn't finished training by the evening, whilst this caused disruption and lead to time being wasted it was unavoidable due to the challenging technical nature of the project. Another key area I had to manage was the availability of the Barkla high performance computing server which if down for maintenance no model would be training. When this maintenance was known to me, I immediately knew to then plan around it and use that time for implementing code or to use another service such as Google Colab. Over the course of this project the PEBBLE algorithm with and without my changes has been run hundreds of times and with each run data detailing the results was produced. Keeping track of this data and what it meant was imperative to the project and helped guide me in which direction to take the improvements and whether a certain change helped or hindered the projects progress. To do this a directory was setup which would store the results of each run and documentation would be made on whether the path of improvement was unlikely to ever find improvement, delivered improvement or needed additional changes to find improvement. This data was also converted into a visual format using matplotlib. The graphs created were made as similarly to the original PEBBLE graphs as possible to highlight the differences. If I to undertake this project again, I would improve the self-management of the project by not saving every single result from the model. In some cases, it was clear that the changes I had made had significantly worsened the algorithm and once I had learnt why this was the case there was no need to keep a record. The reason I would make this change is due to the storage space this data was needlessly taking up and the confusion it causes when weeks later I looked at the data and had forgotten why it was saved. In terms of improving the project for time efficiency I would use a separate device to test whether the changes I made to the algorithm worked properly before leaving them running overnight to only see an error message. To do this I would alter the code by making it significantly less intensive, e.g. removing the unsupervised pre-training step, or reducing the amount of feedback, and then test the code works before running it. This way much less time would have been wasted, and more tests could be completed to find improvements in PEBBLE.

Critical self-evaluation of the process: Given that this is my first time completing a project of this complexity and over such a long-time frame I am proud of the work that I have achieved. If I were to approach this project again with the knowledge gained, I would change some aspects which I now see as being unproductive and wasteful. The key areas I believe I did best and would not change are the way I went about selecting the methods to improve the algorithm. I intentionally chose a sampling method which was significantly more advanced than currently implemented but also a method that I was familiar and had worked in some extent with clustering techniques so that the learning of reinforcement learning was paired with a machine learning technique I felt confident in implementing in a much more advanced code base. One of the largest challenges I faced during the completion of this project was how to approach the complexity and scale of the pre-existing code for the PEBBLE algorithm. The task of understanding code written by a team of other people paired with my starting lack of knowledge surrounding reinforcement learning with human feedback felt a colossal task when

I first started this project however I am now pleased with the skills I have developed when understanding code and the knowledge I have gained surrounding reinforcement learning with human feedback. If I were to approach the problem again, I would still use textbooks and research papers to build knowledge around the area I was working on as well as simpler reinforcement learning models to ensure I understood basic concepts. What I would change about my approach is the way I went about breaking down the PEBBLE code. Instead of going through each function and building my knowledge up through learning what functions did when put together I would start with a top-down approach and learn what the python files did before breaking them down into functions. This method would also allow me to identify the most important functions to learn and dedicate more time to understanding them rather than functions that are only used once and complete a simple task. A further improvement I would make to my approach to this project would be a more rigorous testing process. One of the most frustrating and time-consuming aspects of this project was the issue of logic errors. These errors were challenging to find because it could just be the case that the code I had implemented didn't provide any improvement to the PEBBLE algorithm. Unlike syntax errors, logic errors also ran in the whole code for the maximum available training time and was only apparent after completion. To address this issue, I would test the code I was implementing by running a simpler version of the algorithm in an environment and with hyperparameters where I can easily predict what the results should be so if any logic errors exist, they are much easier to spot and fix. This simpler version of the algorithm would also display important information so I can check the code was working as intended. An example of these logic errors is when developing my Minmax selection algorithm I unintentionally reduced the number of samples being selected by half which led to a decrease in performance however this error was challenging to spot and took time to correct the code and find the eventual increase in performance. In summary, I am proud of the work completed and I believe this project was well managed and achieved the goals I had set.

9. References

[1] L. S. P. A. Kimin Lee, "PEBBLE: Feedback-Efficient Interactive Reinforcement Learning via Relabeling Experience and Unsupervised Pre-training," in *International Conference on Machine Learning*, 2021.

[2] L. S. A. D. P. A. Kimin Lee, "B-Pref: Benchmarking Preference-Based Reinforcement Learning," in *Neural Information Processing Systems*, 2021.

[3] F. W. P. D. A. R. O. K. John Schulman, "Proximal Policy Optimization Algorithms," 2017.

[4] J. L. T. B. B. M. M. S. L. D. A. Paul Christiano, "Deep reinforcement learning from human preferences," in *Neural Information Processing Systems*, 2017.

[5] S. T. A. M. Y. D. P. T. S. L. S. B. J. M. ,. T. E. T. L. N. H. Yuval Tassa, "dm_control: Software and Tasks for Continuous Control," *Software Impacts,* 8 September 2020.

[6] J. H. Xin Jin, Encyclopedia of Machine Learning, Springer, 2011.

[7] S. V. David Arthur, "k-means++: the advantages of careful seeding," *Association for Computing Machinery,* 2007.

[8] P. J. Rousseeuw, "Silhouettes: A Graphical Aid to Cluster Validation," in *Journal of Computational and Applied Mathematics**, 1987.

[9] D. L. &. B. D. W. Davies, “A Cluster Separation Measure,” in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1979.

[10] J. H. T. Calinski, “A Dendrite Method for Cluster Analysis,” *Communications in Statistics,* 1974.

[11] I. Jolliffe, Principal Component Analysis, Springer, 2002.

[12] X. Z. S. R. J. S. Kaiming He, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,” in *International Conference on Computer Vision*, 2015.

[13] B. Z. Q. V. L. Prajit Ramachandran, “Searching for Activation Functions,” in *International Conference On Learning Representations*, 2016.

[14] S. G. F. M. A. L. J. B. G. C. T. K. Z. L. N. G. L. A. A. D. A. K. E. Y. Z. D. M. R. A. T. S. C. Adam Paszke, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” in *Neural Information Processing Systems*, 2019.

10. Appendix

A. Individual Seed Comparisons

Figures 25, 26 and 27 show a comparison of the individual seed executions for the Diverse Minmax KMeans implementation vs the PEBBLE [1] baseline.

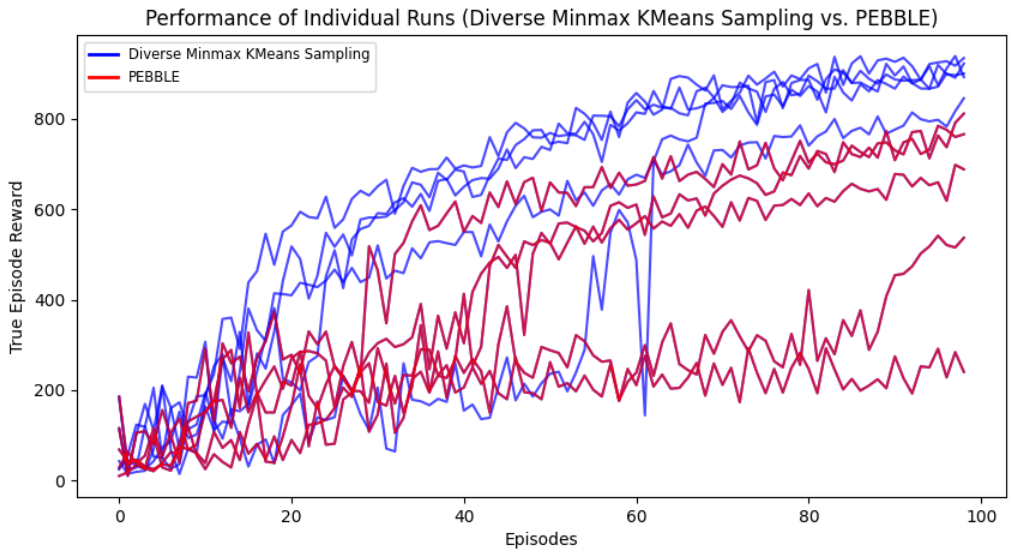


Figure 25: Comparison of individual seed performance for Diverse Minmax KMeans Sampling and PEBBLE baseline. In the Quadruped environment using 1000 feedback and the oracle teacher. The solid line represents the episode reward.

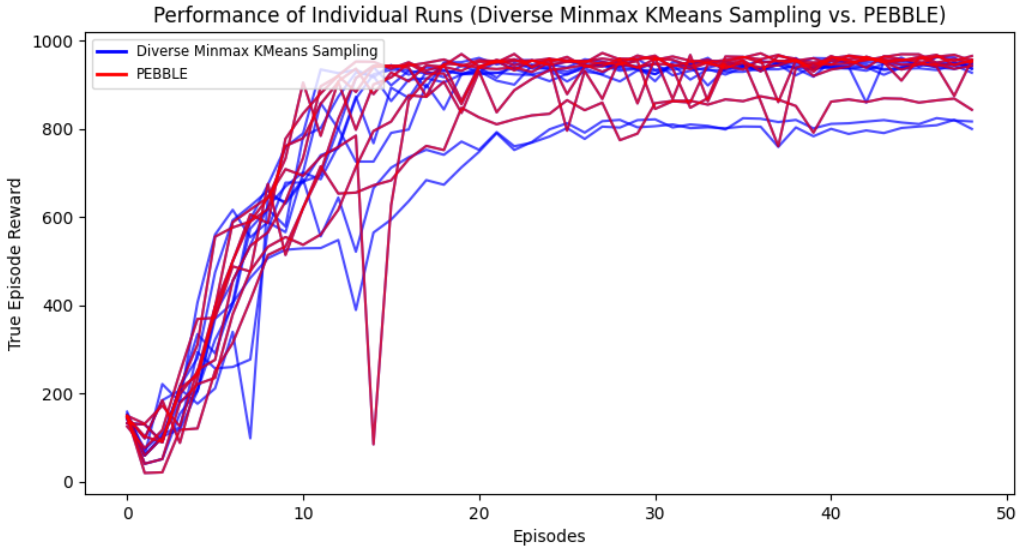


Figure 26: Comparison of individual seed performance for Diverse Minmax KMeans Sampling and PEBBLE baseline. In the Walker environment using 1000 feedback and the oracle teacher. The solid line represents the episode reward.

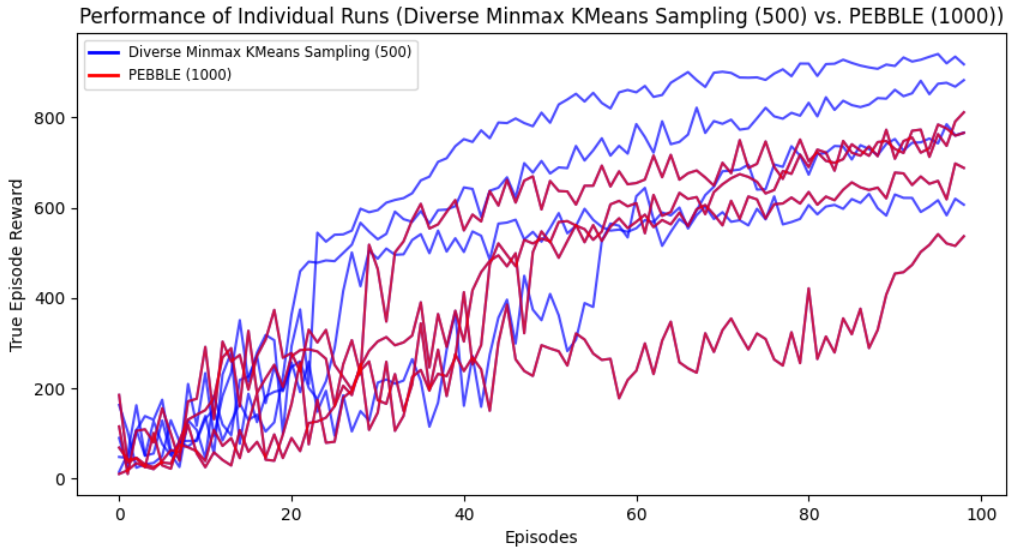


Figure 27: Comparison of individual seed performance for Diverse Minmax KMeans Sampling with 500 feedback and the PEBBLE baseline with 1000 feedback. In the Quadruped environment using the oracle teacher. The solid line represents the episode reward.