42186 Model-based Machine Learning

Group 20

Elysia Livia Gao s222445
Daniela Bahneanu s184366
Diana Podoroghin s194768
Mathias Hovmark s173853

Technical University of Denmark
November 16, 2023

1 Abstract

Our project aimed to enhance the performance of Reinforcement Learning (RL) agents by incorporating text information within a game playing environment. To achieve this, we developed a Bayesian neural network architecture with hierarchical weights derived from prior distributions, establishing a Bayesian framework for learning. We utilized an embedding layer generated through web-scraped game walkthroughs and employed latent Dirichlet allocation (LDA) to extract meaningful topics. For data acquisition, we extracted textual information from game walkthroughs via web scraping and used LDA to generate topic-based embeddings. Our experiments were conducted using OpenAI Gym's reinforcement learning game environment. Our approach involved implementing a Bayesian neural network with hierarchical weights and an embeddings layer. The textual data underwent a transformation process, starting from raw text and progressing through LDA topic modeling before being converted into embeddings. Analyzing the results of our experiments comparing the DQN and ComparisonDQN algorithms, we observed comparable performance in the CartPole-v1 environment. The DQN algorithm, employing a deep neural network (DNN) with multiple layers, and the ComparisonDQN algorithm, utilizing a simpler DNN architecture consisting of three linear layers, both exhibited substantial improvements over random self-play.

2 Background

Reinforcement Learning (RL) shows promise for training intelligent agents in video games, aiming to enhance their performance. Ongoing research addresses uncertainties in the Q-policy to improve RL agent decision-making. Although probabilistic frameworks like Pyro are not commonly used in RL, efforts have been made to reform classical RL algorithms as probabilistic inference problems, such as the MERLIN framework by Levine [1] that applies variational inference for maximum entropy reinforcement learning.

From Levine's proposed work, we consider a simple reinforcement learning PGM as shown on figure 1.

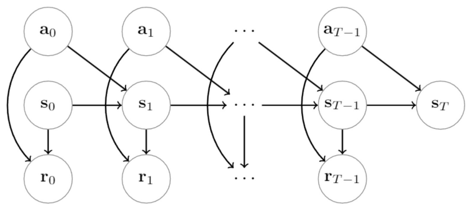


Figure 1: Standard action-state-reward pgm.

The aim is to find a policy: $p(\mathbf{a_t}|\mathbf{s_t}, \theta)$ which specifies a distribution over actions to take, conditioned on the state of the game. The policy may either **1)** output the probability of taking an action in a state, whereafter a categorical distribution (or bernoulli in the binary case) is used to sample the actual action, or **2)** output the expected reward of choosing an action, whereafter the action with the highest expected reward is chosen. Regardless of which approach is chosen, the goal is to find the set of parameters that optimizes the following maximization problem:

$$\theta^* = \arg\max_\theta \sum_{t=1}^{T} E_{(\mathbf{s_t},\mathbf{a_t})\sim p(\mathbf{s_t},\mathbf{a_t}|\theta)}[r(\mathbf{s_t}, \mathbf{a_t})] \tag{1}$$

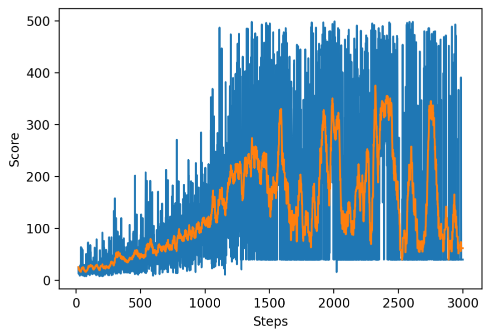
Given the set of parameters θ the probability distribution of a given game trajectory (i.e. a sequence of states and actions) is then given by:

$$p(\tau) = p(\mathbf{s_1}, \mathbf{a_t}, ..., \mathbf{s_T}, \mathbf{a_T}|\theta) = p(\mathbf{s_1}) \prod_{t=1}^{T} p(\mathbf{a_t}|\mathbf{s_t}, \theta) p(\mathbf{s_{t+1}}|\mathbf{s_t}, \mathbf{a_t}). \tag{2}$$

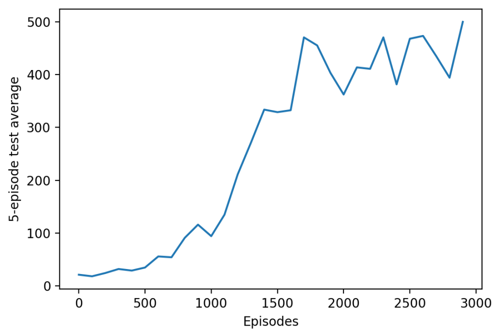
According to Levine (p. 9) [1] this distribution may be approximated using $q(\tau)$ as an approximating distribution for use with SVI:

$$q(\tau) = q(\mathbf{s_1}) \prod_{t=1}^{T} q(\mathbf{s_{t+1}}|\mathbf{s_t}, \mathbf{a_t}) q(\mathbf{a_t}|\mathbf{s_t}) \tag{3}$$

Where it is further required that $q(\mathbf{s_1}) = p(\mathbf{s_1})$ and $q(\mathbf{s_{t+1}}|\mathbf{s_t}, \mathbf{a_t}) = p(\mathbf{s_{t+1}}|\mathbf{s_t}, \mathbf{a_t})$ (i.e. we fix the initial state and use the same transition function between states). We briefly validated that reinforcement learning is possible when using SVI in this framework as results of training and testing is shown on figure 2 below. However, our project will mainly try to set up a framework for enhancing (DQN) agent performance by incorporating in-game text, tutorials, and walkthroughs. We modify the training process of Deep Q-networks (DQN) by introducing a Bayesian neural network architecture with hierarchical weights. This integration of textual knowledge, hierarchical weights, and Bayesian modeling improves decision-making, captures global-local relationships, and enhances the agent's overall performance.



(a) Training curve - fluctuating performance compared to the test performance on the right figure, is due to exploration.



(b) Test performance as a function of training time. Testing is based on 5-run averages of the policy.

Figure 2: Result of using SVI with Pyro to optimize the point parameters, θ, of $p(\tau)$

3 Data Selection

In this project, we have utilized three distinct datasets to enhance our reinforcement learning algorithm. Firstly, we selected the CartPole-v1 environment from OpenAI's gym library as our primary game environment. The CartPole-v1 environment's variables include the cart's position, cart velocity, pole angle, and pole velocity. By leveraging this dataset, we can train our algorithm to effectively learn and optimize its decision-making processes in complex and dynamic scenarios.

Secondly, we generated a self-play dataset using reinforcement learning techniques. Through multiple episodes of self-play, our algorithm interacted with the CartPole-v1 environment, generating a dataset that captures the agent's actions, observations, and rewards. This self-play dataset, gathered from our own exploration of the environment, provides valuable training data for our algorithm. It allows the algorithm to learn from its own experiences, adapt its strategies, and improve its performance over time.

Lastly, we incorporated a text dataset obtained by scraping various websites related to the Cartpole framework. This text dataset represents a diverse range of information sources, collected from multiple external sources. The inclusion of this text dataset expands the scope of our learning process beyond the game environment itself. By incorporating external information, such as tutorials and resources on applying reinforcement learning to Cartpole, we aim to further enhance our algorithm's learning capabilities and decision-making.

The combination of these three datasets—a dataset from the CartPole-v1 environment, a self-play dataset generated through reinforcement learning, and a text dataset gathered from multiple sources—provides a comprehensive and diverse training framework for our algorithm. These datasets, with its many variables and inherent layers, justifies the application of advanced methods from model-based machine learning.

4 Data Description

In this project, our investigation focused on the CartPole-v1 environment provided by OpenAI's gym library for reinforcement learning experiments. The CartPole environment encompasses a four-dimensional state space and a two-action decision space. The state space comprises the positional information of the cart, restricted within the range of -2.4 to 2.4 units, the unbounded velocity of the cart, the pole angle constrained between -41.8 and 41.8 degrees, and the unbounded tip velocity of the pole.

The primary objective of our reinforcement learning model was to maintain the pole in an upright position on the cart for an extended duration. Achieving this objective necessitated selecting appropriate actions based on the current state of the system. The action space was represented by binary values, with action 0 denoting a leftward push and action 1 representing a rightward push of the cart. A reward of +1 was assigned to each time step where the pole remained balanced.

The termination conditions for an episode in the CartPole environment were set as follows: if the pole angle deviated beyond 12 degrees from the upright position, if the cart position exceeded a deviation of 2.4 units from the initial position, or if the episode extended beyond 500-time steps.

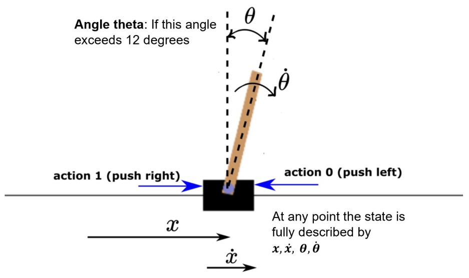

Angle theta: If this angle exceeds 12 degrees
θ
$\dot{\theta}$
action 1 (push right)
action 0 (push left)
x
$\dot{x}$
At any point the state is fully described by $x, \dot{x}, \theta, \dot{\theta}$

Figure 3: Picture of the classic CartPole environment.

For training our Deep Q-Network (DQN) algorithm, we utilized the state, action, next state, and reward tuples as inputs. As the CartPole environment is a single-agent game, our agent competed against its own previous performance to improve its overall score. Self-play data played a crucial role in enhancing the agent's strategy by providing a diverse range of experiences.

To enrich the capabilities of our DQN agent, we integrated textual data acquired through an automated search using the Bing search engine. This search process yielded a comprehensive collection of websites containing pertinent information about the game. Subsequently, we employed web scraping techniques to extract text data from these websites. To ensure consistency and data quality, we employed natural language processing techniques, including lowercasing, and removal of non-alphanumeric characters, symbols, and stopwords. The resulting dataset, referred to as `parsed_data`, consisted of clean and tokenized text data from multiple relevant websites, poised for utilization in enhancing our DQN agent through the incorporation of text embeddings.

Further insights into the dataset used for our experiments are provided in Table 1.

Metric	**Value**
Total number of words in the document	36324
Total number of unique words in the document	2021
Average length of unique words	0.381
File size (in kilobytes)	247

Table 1: Data Description

5 Description of Modelling Approach

5.1 Overview: Generative Process and PGM

The modelling approach utilizes a generative process to simulate data, allowing for the creation of synthetic datasets that follow a predefined structure. This process involves drawing global parameters, specifying level-specific coefficients, and applying transformations to generate observations.

First, each observation undergoes the specific transformations associated with each level in the hierarchy. This includes applying the level-specific coefficients and the global parameters. Additionally, an important step involves the application of the word embedding layer. This layer applies a transformation that incorporates semantic representations of words, enhancing the richness of the generated data.

Following the transformations, the hierarchical model is applied to the observation. The model utilizes a hyperbolic tangent (tanh) activation function to capture non-linearity within the data. The transformed observation is passed through the hierarchical model, incorporating the level-specific coefficients and the global parameters. This process facilitates the propagation of information across the different levels, allowing for the generation of observations that capture the hierarchical relationships and complexity in the data.

Once the observation has been processed by the hierarchical model, the generative process computes the output. The output is obtained by applying an output layer that maps the transformed observation to the desired format, such as a prediction or estimation. In this specific generative process, the output is computed as the result of the output layer followed by the application of the squeeze function to remove extra dimensions.

To complete the generative process, the target value for each observation is drawn from an appropriate distribution. In this case, a normal distribution is utilized to model the target value. The distribution is centered around the prediction obtained from the output layer, reflecting the uncertainty and variability inherent in the data. A fixed scale, such as 0.1, is used to control the spread of the distribution and introduce a level of noise or variability in the generated targets.

By following this comprehensive generative process, each observation is systematically generated, capturing the hierarchical relationships, incorporating word embeddings, applying non-linearity, and drawing target values that align with the intended characteristics. This approach allows for the generation of synthetic data that reflects the complexities and interactions present in real-world scenarios, enabling a range of analyses and investigations.

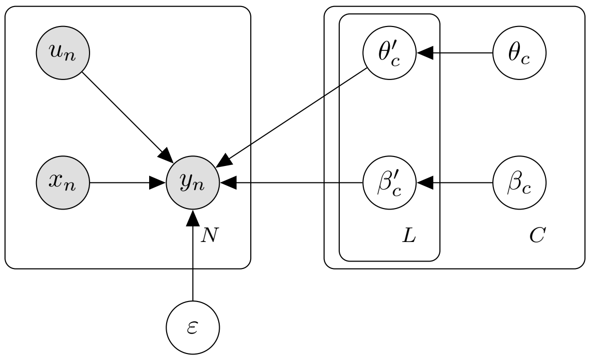


Figure 4: A probabilistic graphical model

1. Draw global linear coefficients $\beta_c \sim \text{normal}(0, 1)$
2. Draw global DNN parameters $\theta_c \sim \text{normal}(0, 1)$
3. Draw word embedding parameters $e \sim \text{normal}(0, 1)$
4. For each level $l \in \{1, \dots, L\}$:
   (a) Draw coefficients $\beta_{c_l} \sim \text{normal}(\beta_c, b_l)$
   (b) Draw coefficients $\theta_{c_l} \sim \text{normal}(\theta_c, c_l)$
5. For each observation $n \in \{1, \dots, N\}$:
   (a) Compute the linear part of the model: $X = \tanh(\beta \cdot X^T)$
   (b) Apply the word embedding layer: $X = X + \tanh(\text{word\_layer.weight} \cdot \text{word\_embedding}^T)$
   (c) Apply the hierarchical model: $X = \tanh(\theta \cdot X^T)$
   (d) Compute the output: $\text{prediction\_mean} = \text{out\_layer}(X).\text{squeeze}(-1)$
   (e) Draw target $y_n \sim \text{normal}(y_n|\text{prediction\_mean}, 0.1)$

In the given generative process, the embedding layer has been computed beforehand, independent of the probabilistic graphical model (PGM) itself. It is essential to clarify that the estimation of word embeddings is not performed as a part of the PGM. Instead, pre-trained word embeddings from the GLoVe library are utilized. These word embeddings provide low-dimensional vector representations of words, enabling their effective utilization in neural networks. Additionally, word embeddings facilitate the calculation of similarity measures between different words, which is crucial in various natural language processing tasks.

It is worth noting that the integration of word embeddings in this project serves as a proof of concept, considering the simplicity of the CartPole environment. With more time and resources, further investigation could be conducted to explore alternative ways to leverage embeddings such as converting states to text or using topic modeling to find related actions.

5.2 Generative Process and PGM of Topic Modelling

The Latent Dirichlet Allocation (LDA) model, illustrated by the probabilistic graphical model (PGM) in Figure 5, captures the generative process for analyzing web-scraped text data. In this process, we first select hyperparameters α and β to control the document-topic distribution and the topic-word distributions, respectively. For each topic, a vector of words is drawn from a Dirichlet distribution with parameter β. For each document, a vector of topics is drawn from a Dirichlet distribution with parameter α. Then, for each word in the document, a topic assignment is sampled based on the document's topic distribution, and a word is selected from the corresponding topic's word distribution. LDA assumes a mixture of topics within each document and provides a framework to uncover latent topics and their associations with words in the analyzed text data.

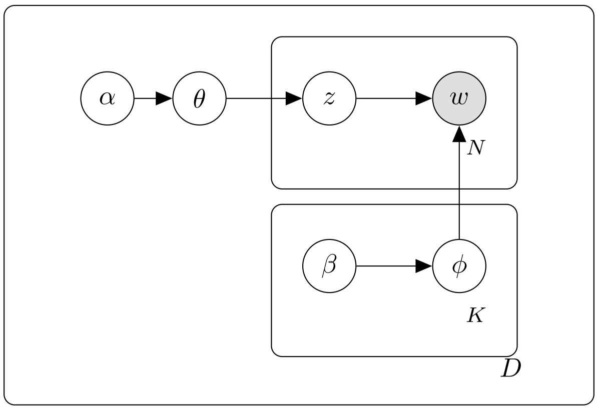


Figure 5: PGM of the Latent Dirichlet Allocation (LDA)

1. For each topic k, draw a vector of C words ϕ_k from a Dirichlet distribution with parameter β, i.e., $p(\phi_k|\beta) = \text{Dir}(\beta)$.
2. For each document d:
   - Draw a vector of K topics θ_i from a Dirichlet distribution with parameter α, i.e., $p(\theta_d|\alpha) = \text{Dir}(\alpha)$.
   - For each word j in document d (where $j = 1, 2, \ldots, N$):
     - Assign a topic $z_{d,j}$ to word j based on the distribution $p(z_{d,j}|\theta_d) = \text{Cat}(\theta_d)$, where Cat represents the Categorical distribution.
     - Select a word $w_{d,j}$ from the topic-word distribution $\phi_{z_{d,j}}$ based on the distribution $p(w_{d,j}|\phi_{z_{d,j}}) = \text{Cat}(\phi_{z_{d,j}})$.

6 Data Analysis and Modelling

6.1 Deep Q-Network

We define the Deep Q-Network using PyTorch and Pyro, which combines a probabilistic graphical method (PGM) with a neural network. The architecture of the DQN is a Feed-Forward Neural Network (FFNN) with three fully connected layers, which takes in the state of the environment as input and outputs the Q-values for each possible action. The FFNN is defined as a PyroModule and includes an additional word embedding layer.

The `FFNN` class represents the neural network component of the DQN. It has an initialization method that defines the architecture of the network. The network consists of an input layer, a hidden layer, and an output layer. We use an FFNN because it is well-suited for approximating complex functions and capturing non-linear relationships in the data.

The weights of these layers are initialized using normal distributions. The FFNN also includes hierarchical global and local weights, represented by β and θ parameters, respectively to capture hierarchical relationships and to allow different levels of the network to specialize in different features or patterns. The β_{global} and θ_{global} weights are shared across all levels, while the β_l and θ_l weights are specific to each level. The number of levels is specified by the n_{levels} parameter. Additionally, there is a word embedding layer that takes a word embedding matrix as input. We use embeddings to represent categorical variables in a continuous and distributed manner, enabling the model to capture semantic relationships between different categories or words.

The `forward` method of the `FFNN` class takes an input X, an optional `level_idx`, and an optional target output y. If `level_idx` is provided, it uses the corresponding local weights (β_l and θ_l), otherwise, it uses the global weights (β_{global} and θ_{global}). The input X is passed through the network, starting with the

β weights, followed by the word embedding layer, and finally the θ weights. The output of the network is then passed through the output layer to obtain the predicted mean Q-values. The predicted mean Q-values are used to sample from a normal distribution with a fixed variance of 0.1, and the resulting sample is compared to the target output y.

The `DQN` class represents the overall DQN model and is also defined as a `PyroModule`. It has an initialization method that creates an instance of the `FFNN` class with the specified parameters. The `forward` method of the `DQN` class simply calls the `forward` method of the `FFNN` class.

The below provided pseudo-code represents the integration of the aforementioned DQN classes and the reinforcement learning loop where an agent interacts with an environment over multiple episodes and steps. In each step, the agent observes the current state of the environment, selects a hierarchical level, and obtains the output value from a Deep Q-Network (DQN) model. Based on this output, the agent performs an action in the environment and observes the resulting reward. The DQN model is then trained by performing a backward pass to update its weights and improve its performance. This loop repeats for each episode, allowing the agent to learn and refine its decision-making process over time. By combining a hierarchical model with the DQN architecture, the agent can leverage global and local weights as well as word embeddings to enhance its understanding and decision-making capabilities.

Algorithm 1 Deep Q-Network with Hierarchical Model

```
FFNN.initialize($n_{\text{in}}, n_{\text{hidden}}, n_{\text{out}}, n_{\text{levels}}, b_{\text{global}}, c_{\text{global}}, b_l, c_l$, word_embedding)
DQN.initialize($n_{\text{observations}}, n_{\text{actions}}, n_{\text{levels}}, b_{\text{global}}, c_{\text{global}}, b_l, c_l$, word_embedding)
for each episode do
    for each step do
        // Observe the current state
        X ← observe_state() // Select the hierarchical level
        level_idx ← select_hierarchical_level() // Get the output value from the DQN
        y ← DQN.forward(X, level_idx)
        // Perform the action based on the output value
        perform_action(y) // Observe the reward
        observe_reward() // Perform a backward pass for training
        FFNN.backward_pass()
    end
end
```

6.1.1 Training and Hyperparameters

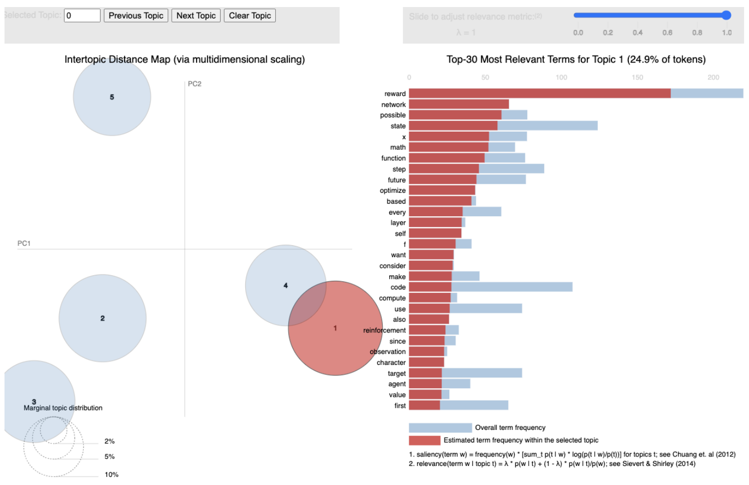
We utilize Pyro's implementation of the Deep Q-Network (DQN) with standard reinforcement learning procedures. Key hyperparameters, including epsilon-greedy policy values, target network update rate, and learning rate, are defined. The optimization process involves retrieving random transitions from a replay memory buffer, calculating loss using the Huber loss function, updating the target network with a soft update mechanism, and optimizing the loss using the AdamW optimizer with gradient clipping.

6.2 LDA and Topic Modelling

In natural language processing, Latent Dirichlet Allocation is a probabilistic model commonly used for topic modeling. It assumes that the data is generated by unobserved groups or topics. Each document in the corpus is viewed as a mixture of these topics, and each topic is represented as a distribution over words. Thus, LDA can identify co-occurring words and group them into topics, providing a representation of documents as collections of topics.

In our corpus, each document is represented as a combination of different topics. The mixture proportions of these topics within each document are denoted by θ, while the topic-word distributions are represented by ϕ in the PGM. The latent variable z represents the topics associated with the words in the documents, and the observable words are represented by w. Both the topic proportions and word proportions within topics are modeled using a Dirichlet distribution.

To perform inference in LDA, we employ variational inference and optimize the parameters using the Adam method. The guide function approximates the posterior distribution. The LDA graph in Figure 8 visually represents the model. Larger circles indicate more prevalent topics within the document while overlapping circles indicate related topics with common words. On the other hand, further apart circles suggest less correlation between the corresponding topics.



1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

Figure 6: LDA visual representation

The graph showing the Latent Dirichlet Allocation (LDA) 7 model's Evolution of Evidence Lower Bound (ELBO) offers insights into the optimization procedure. Where the y-axis displays the ELBO values, which are used as a gauge of the accuracy of the approximation made by the model, while the x-axis depicts the algorithm's iterations or steps.
An early upward trend can be seen in the ELBO evolution graph for LDA, which shows development toward optimizing approximation quality. The trade-off between model complexity and the accuracy of the approximation obtained is highlighted by the fact that it finally levels off, indicating convergence to a local optimum.

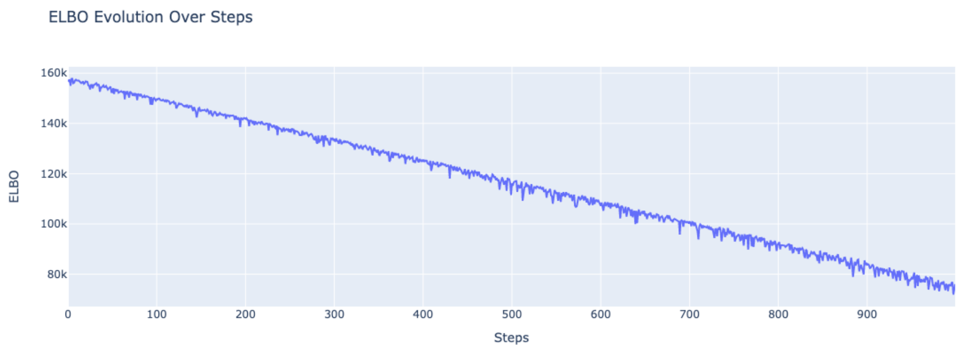


Figure 7: ELBO Evolution Over Steps

7 Results

The results of our experiments comparing the DQN and ComparisonDQN algorithms, which utilize different neural network architectures, demonstrate comparable performance in the CartPole-v1 environment. The text embeddings used in the DQN agent are proof-of-concept, but both algorithms demonstrate significant improvements over random self-play, as shown in the episode rewards graph. Compared to random self-play, which yields lower rewards indicating an ineffective strategy, both DQN and ComparisonDQN algorithms achieve consistently high rewards in the CartPole task, demonstrating their ability to learn and improve performance in reinforcement learning.

In this context, the term "duration" refers to the length of each episode in terms of the number of time steps taken by the RL agent in the CartPole environment before the episode terminates. The duration can be seen as a measure of the agent's ability to maintain balance and perform well in the CartPole task. Lower durations indicate better performance, as the agent is able to keep the pole balanced on the cart for a longer duration and earn higher rewards during each episode.

Overall, the duration plots provide a visual representation of how the episode duration changes over time, indicating the agent's ability to maintain balance. Although the rewards plot for the DNN algorithms is not showcased due to the near-perfect rewards achieved in each episode of the CartPole task, the duration plot serves as a valuable indicator of the agent's performance and learning progress.

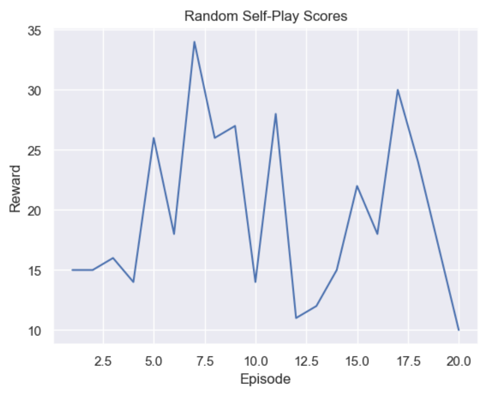
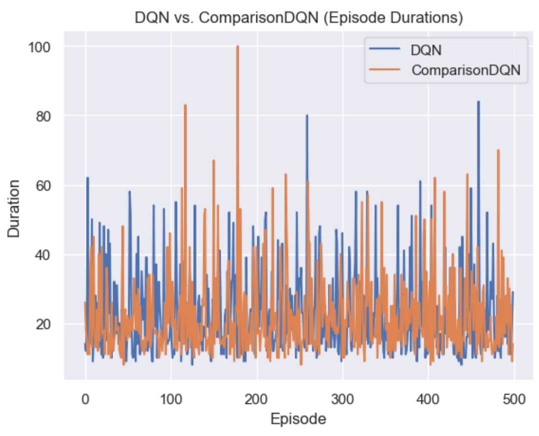
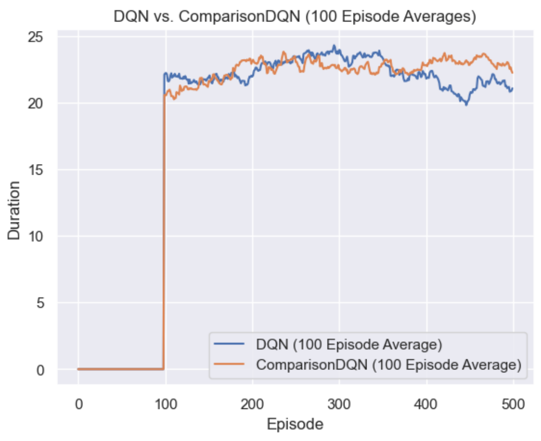


Figure 8: Random Self-Play Reward Results (No Training)



(a) Model Comparison



(b) Model Comparison (100 Episode Averages - averages are not report until after first 100 episodes.)

Figure 9: Model Comparison and Results

8 Conclusion

In conclusion, our project aimed to enhance the performance of Reinforcement Learning (RL) agents by incorporating text information within a game playing environment. We successfully developed a Bayesian neural network architecture with hierarchical weights, integrating uncertainties and probabilistic reasoning into the decision-making process. By leveraging in-game text, tutorials, and walkthroughs, our approach showed promising performance of Deep Q-Networks (DQN) agents.

Through the utilization of the CartPole-v1 environment and web-scraped text data, we established a comprehensive dataset for training and enhancing our RL algorithm. The integration of textual knowledge, hierarchical weights, and Bayesian modeling proved effective in improving decision-making, capturing global-local relationships, and enhancing the overall performance of the DQN agent. The experiments comparing the DQN and ComparisonDQN algorithms demonstrated comparable performance in the CartPole-v1 environment. Both algorithms exhibited substantial improvements over random self-play, highlighting their ability to learn and adapt strategies over time. The utilization of text embeddings served as a proof of concept, showcasing the potential for leveraging textual information in reinforcement learning tasks.

Overall, our project successfully explored and implemented novel techniques to enhance RL agent performance through the incorporation of text information.