University of Padova

Department of Information Engineering DEI

Master Thesis in ICT FOR INTERNET AND MULTIMEDIA - INGEGNERIA PER LE COMUNICAZIONI MULTIMEDIALI E INTERNET

Learning sensor-agent communication with variable quantizations

Supervisor
Andrea Zanella
University of Padova

Co-supervisor
Federico Chiariotti
University of Padova

Master Candidate
Pietro Talli

Student ID
2021427

Academic Year
2021-2022

Date
December 12, 2022

Abstract

In this work the possibility of training a remote (deep) reinforcement learning system was studied. The thesis focuses on the problem of learning to communicate relevant information from a sensor to a reinforcement learning agent. Different quantization strategies were tested in order to balance a trade-off between the effectiveness of the message communicated and the limited communication rate constraint.

Contents

Listing of figures

Listing of tables

1 Introduction

Automation of processes in industries has become more relevant recently, exploiting the power of Internet of Things (IoT) [5] which has become a key component. In those scenarios mobile robots have to coordinate solving a multi-agent reinforcement learning problem. Coordination and cooperation can be fully effective when the agents can communicate and exchange messages, whether they are observations about the environment they sense or decisions (actions) they take. Especially when dealing with wireless communication, optimization and efficient managing of the communication resources become extremely important. For example, consider a controller that receives real-time updates of the status via wireless transmission. To deal with this problem, new concepts have been introduced such as the Age of Information (AoI) and the Urgency of Information (UoI) [6], [7]. Those concepts are fundamental to derive some theoretical guarantees on the minimal rate needed to update the status and provide fresh information at a certain rate. Another valid and effective tool for optimizing multi-agent systems comes from the Deep Reinforcement Learning and recent advances in Neural Networks. Starting with seminal works on Deep Q-Networks by Google Deepmind [8], the usage of Deep Learning is becoming a fundamental part in advanced Control problems. Deep Neural Networks have been proved to be useful for learning complex functions from very high dimensional inputs such as images or videos. This feature makes these models a suitable choice when the message to communicate comes in form of images or videos. In this case, it is possible to exploit Joint Source-Channel Coding schemes to optimize both for the encoding and the communication among the communication channel [3], [9].

This work aims to bridge the gap between the communication aspects and the control

aspects, providing methods to optimize both sides. A deep learning architecture is used to build different blocks which handle information processing, communication and control policies. The possibility of optimizing these blocks in order to obtain a better overall performance of the system is studied. The focus of this work is on the communication aspects and the different levels of analysis which can characterize this system. The communication problem is analyzed from three levels of communication introduced in [10]. In particular, communication is optimized by using the same procedure for every level, showing the versatility of this approach.

The specific system model which comprises a communication and a control problem is formulated considering two agents. More specifically, an agent receives information from another agent and has to adapt its behavior based on the received information. The scenario is simplified from the general case, since an agent is only doing a communication problem and the other agent is solving a control problem. More specifically it is possible to name this as a **remote reinforcement learning** problem [11], [12]. The aim of the project is to demonstrate and provide a method to solve efficiently the control problem by improving the effectiveness of the information transmitted by an agent to the other. The agent which transmits the information can be seen as a Sensor node, whereas the receiver agent can be seen as an Actuator node. In the test accomplished, the receiver node performs three types of tasks. These tasks allow to study the optimization of the communication with respect to the three communication problems.

The communication channel considered is digital channel with limited capacity. In order to obtain discrete representation at the output of the layers of a Neural Network, a specific quantization strategy is designed and evaluated. The main idea is derived by the work in [4], but some modifications are introduced to manage the communication resources better. This way, the channel can be treated a the layer of a neural network where the nodes are force to represent a deterministic discrete distribution.

Quantization is often hard in high dimensional vectors and the solutions provided by the typical adaptive algorithm might converge to sub-optimal configurations. To deal with this problem, quantizers with different sizes are used in order to test how the performance of the system depends on the quantization step. Moreover, these different quantizers are used to obtain different level of compression in the information transmitted. Compression allows to discard information from the current state the Sensor has to communicate and decreases the rate needed to transmit the information. Through different quantizers, it is possible to vary the rate at which the information is transmitted. In particular, two communication strategies are studied: a fixed quantization and a variable quantization. The first strategy considers the transmitter as a passive node which can only work at a

certain rate. The second strategy considers the Sensor as an agent which can decide the amount of information to transmit, due to the current state and the task the receiver aims to solve. The work shows that the second strategy is able to reduce the average rate by learning to use the most suitable quantization.

In the final part of the results, an intuition and a discussion about the communication strategy that the Sensor learns are provided. A relationship between the communication rate and the entropy of the actions of the receiver is inferred, providing an insight on how an ideal system model should work.

2 Methods

In this chapter the main methods used in the developing of the model are explained and the relevant parts are analyzed.

2.1 Reinforcement Learning

Reinforcement learning (RL) is a well established research field which aims to adaptively learn optimal control strategies in a wide variety of environments. An extensive introduction about RL can be found in [1]. In recent years, the power of RL was proved by solving some challenging scenarios such as games (e.g. Go and Chess) [8]. In particular, researchers combined Deep Neural Networks [13] with the algorithmic procedure from the RL theory to achieve very high performances in a lot of tasks. Subsequently, these techniques were extended also to continuous control problems [14], proving the effectiveness of Deep Reinforcement Learning in those scenarios.

A typical RL problem consists on 4 parts which fully describe the setting:

- An **Agent**: the entity aiming to solve the task
- A set of **states** S_t: exhaustive description of the system in which the agent can be
- A set of **actions** A_t: (that could depend on the state) that can be taken by the agent
- An **environment** with which the agent interacts and that at each time t could either provide **rewards** R_t and move the agent to a new state –depending on the state/action pair

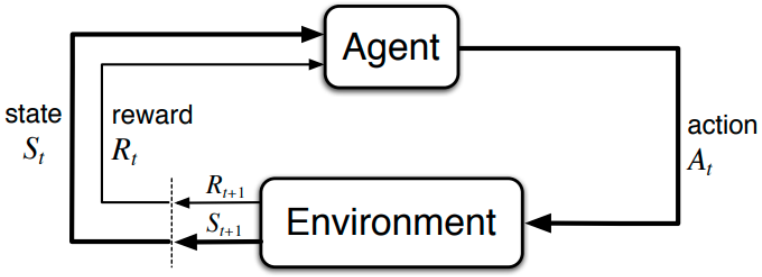

Figure 2.1: A scheme of the interaction between RL building blocks [1]

In Fig. 2.1 it is possible to see a scheme of the interaction between the building blocks of the RL system. At each time step t the agent observes the state S_t and then it chooses an action A_t to play. After the agent performs the action, it receives the new state S_{t+1} and reward R_{t+1} which the environment provides, given that the agent chose action A_t.

The goal of the agent is to maximize the cumulative rewards over time. In order to choose the action at each time step, the agent implements a mapping from the state space $\mathcal{S}$ to the action space $\mathcal{A}$. This mapping represents the probability that the agent chooses action A_t when observing S_t from the environment. In the RL terminology, this mapping is called **policy**. The policy π of an agent is a probabilistic mapping $\pi : \mathcal{S} \to \Phi(\mathcal{A})$ which represents the probability of choosing action A_t given the state S_t. The target of RL is to learn the optimal policy $\pi^\star$ that maximizes the cumulative rewards.

2.1.1 Exploration vs. exploitation

In typical training of RL agents, the policy must be set according to certain criteria to allow exploration and exploitation. Exploration means that the agent should try different combinations of actions in certain states, since all the possibilities have to be tested. Implementing an exploration behavior for the policy is crucial to avoid to learn sub-optimal policies, which consider only a restricted set of the possible actions and not the entire action space. On the other hand, exploitation means that the policy should take advantage of the experience it has gained exploring different actions and states to maximize the obtained reward. Exploitation is the opposite of exploration because the agent has to act greedily, choosing only the best actions and thus does not explore new possibilities. It is clear that exploration and exploitation cannot be performed together but have to come within a trade-off. Different implementations of this trade-off exist [1], [15].

2.2 Markov Decision Process

In order to mathematically formalize the problem of RL, a probabilistic modelling of the RL system has to be introduced. The most general description of an agent transitioning from one state to another according to some probabilities is given by a Markov Process.

A Markov process S_t is a stochastic process with the property that, given the value of X_t, the values of S_u for $u > t$ are not influenced by the values of S_v for $v < t$ [16].

This definition means that the probability of any future behavior of the process is not altered by additional knowledge concerning its past behavior. More interesting, in order to develop algorithms and procedures to train the agent, is the discrete-time Markov Process. It is a Markov process whose state space is a finite or countable set and whose time index set is $T = \{0, 1, 2...\}$. All the states S_t of the process respect the Markov property which can be written as

$$\Pr[S_{t+1} \mid S_t] = \Pr[S_{t+1} \mid S_1, ..., S_t]. \tag{2.1}$$

If the states in the considered RL scenario respect the Markov property, it means that the probability of being in state S_{t+1} only depends on state S_t and knowing any previous state does not change this probability. Thus, the state is a sufficient statistics for the present and the past meaning that it is possible to throw away the rest of the history. Defining a good state is fundamental to develop effective RL solutions: it typically requires good domain expertise. Given a discrete set of states $\mathcal{S}$, it is possible to define a transition matrix P where the entry P_{ij} is the probability to go from state i to state j

$$P_{ij} = \Pr[S_{t+1} = j \mid S_t = i] \quad \forall i \in \mathcal{S}, j \in \mathcal{S}. \tag{2.2}$$

However, this modelling of the state transition does not consider the actions and the rewards that are associated with those state transitions. In order to take into account all the building blocks of the RL setting, Markov Decision Processes are introduced.

A Markov Decision Process (MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$ such that:

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- P is a state transition probability matrix with entries

$$P^a_{ss\prime} = \Pr[S_{t+1} = s' \mid S_t = s, A_t = a]$$

- R is a reward function such that

$$R_s^a = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$

- $\gamma \in [0, 1)$ is a discount factor

Note that in the transition probability matrix of an MDP, the next state not only depends on the current state S_t but also on the chosen action A_t. The reward function R is the expected value of the immediate reward R_{t+1} that the agent receives from the environment for having performed action A_t in state S_t. In order to understand the last parameter γ, the return of the process G_t has to be defined. The return of an MDP G_t is given by

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{2.3}$$

In RL the agent is not interested in maximizing the value of a single step, but it wants to maximize the return G_t, since the goal of learning is to maximize the cumulative rewards. In the definition of G_t, the parameter γ weights the present value of future rewards. Since γ takes value in the range $[0, 1)$, the return G_t only considers the current reward when $\gamma = 0$, whereas it considers all the future rewards equally important as γ approaches 1. $\gamma = 1$ is not used in infinite time RL, since there could be convergence issues. In general, this parameter allows to define a trade-off between how much an agent should be interested in the immediate reward rather than in a long-term one.

Given the definition of MDP, the definition of stochastic policy can be introduced. A stochastic policy π is a distribution over actions $a \in \mathcal{A}$ given that we are in a certain state $S_t = s$

$$\pi(a \mid s) = \Pr[A_t = a \mid S_t = s] \tag{2.4}$$

A policy fully defines the behavior of an agent. MDP policies depend on the current state because the states enjoy the Markov property, and thus previous states do not bring any additional information. In the definition of a policy, the reward term is not included since the policy can be given or be learned with a separate dedicated procedure. It is also worth noting that, since every action is associated with a probability, it is possible to balance the exploration/exploitation trade-off by ensuring to have non zero probability for any action, which allows to explore every possible action.

2.2.1 The solution of the MDP and the Bellman Equation

In this section, a mathematical derivation of the solution of an MDP is given, further demonstrations of these results can be found in [1] and [17]. Given an MDP $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$ and a policy π the sequence of states $S_1, S_2, ...$ is a Markov process with transition probability matrix P^π such that

$$P^\pi_{ss'} = \sum_{a \in \mathcal{A}} \pi(a \mid s) P^a_{ss'}. \tag{2.5}$$

The reward process R^π_s is given the policy π is

$$R^\pi_s = \sum_{a \in \mathcal{A}} \pi(a \mid s) R^a_s. \tag{2.6}$$

In order to solve the MDP and find the optimal policy $\pi^\star$ which maximizes the return G_t, the state value function and the state-action value function have to be defined. The state value function

$$V_\pi(s) = \mathbb{E}[G_t \mid S_t = s] \tag{2.7}$$

is the expected return from state s if the agent follows policy π. The state-action value function

$$Q_\pi(s, a) = \mathbb{E}[G_t \mid S_t = s, A_t = a] \tag{2.8}$$

is the expected return from state s if the agent takes action a and then follows policy π. Using the definition of the return G_t given in Eq. (2.3) it is possible to write the value function as the sum of two terms

$$V_\pi(s) = \mathbb{E}[R_{t+1} + \gamma V_\pi(S_{t+1}) \mid S_t = s]. \tag{2.9}$$

Similarly for the $Q_\pi(s, a)$ function, as it can be written

$$Q_\pi(a, s) = \mathbb{E}[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]. \tag{2.10}$$

Also note that the state value function $V_\pi(s)$ and the state-action value function $Q_\pi(s, a)$ are related by the following equations

$$V_\pi = \sum_{a \in \mathcal{A}} \pi(a \mid s) Q_\pi(a, s) \tag{2.11}$$

$$Q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a V_\pi(s'). \tag{2.12}$$

Using Eq. (2.12) inside Eq. (2.11) a recursive definition of the value function is obtained as

$$V_\pi = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left(R_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a V_\pi(s') \right) \tag{2.13}$$

On the contrary, for the state-action value function substituting Eq. (2.11) inside Eq. (2.12) the recursive definition is given by

$$Q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a' \mid s') Q_\pi(s', a') \tag{2.14}$$

Let us denote P^π the matrix containing all the transitions as defined in Eq. (2.5) and R^π the vector containing the expected rewards defined in Eq. (2.6) for every $s \in \mathcal{S}$ it follows a matrix formulation of the recursive definition of the value function written in eq. (2.13) which is

$$V_\pi = R^\pi + \gamma P^\pi V_\pi, \tag{2.15}$$

where V_π is the vector of all the values $V_\pi(s), \forall s \in \mathcal{S}$. Eq. (2.15) admits a solution which expresses the values as

$$V_\pi = (I - \gamma P^\pi)^{-1} R^\pi. \tag{2.16}$$

The latter is also known as the Bellman expectation equation as it allows to find the expected value of the value function V_π given a generic MDP and a fixed policy π.

However, the Bellman expectation only allows to evaluate a policy, it does not provide any information about the optimal policy. In order to solve control problem and perform policy improvement, the Bellman optimality equation should be introduced. To derive the Bellman optimality equation, the optimal value function $V_\star(s)$ is introduced as

$$V_\star(s) = \max_\pi V_\pi(s), \tag{2.17}$$

and the optimal state-action value function is

$$Q_\star(s,a) = \max_\pi Q_\pi(s,a) \tag{2.18}$$

If the agent knows the optimal $Q_\star(s,a)$ values the optimal policy follows straightforwardly,

the optimal policy $\pi_\star$ is

$$\pi_\star(a \mid s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in \mathcal{A}} Q_\star(s,a) \\ 0 & \text{otherwise} \end{cases} . \tag{2.19}$$

Any MDP admits an optimal policy $\pi_\star$ as it is proved in [17]. Repeating the previous steps done for the Bellman expectation equation and substituting the expectation operation with the $\max_{a \in \mathcal{A}}$, a recursive definition of the optimal value $V_\star(s)$ and action-value $Q_\star(s,a)$ are obtained as

$$V_\star(s) = \max_a R_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a V_\star(s) \tag{2.20}$$

$$Q_\star(s,a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a \max_{a'} Q_\star(s', a'), \tag{2.21}$$

which are the Bellman optimality equations. It is worth noting that, differently from the Bellman expectation equation, there is no closed-form solution to this system as the $\max_a$ operator is non-linear. In order to learn these optimal values, the agent will resort to iterative methods such as Q-learning or SARSA [1], [17].

2.2.2 Proper Definition of Exploration/Exploitation trade-off

Recalling the exploration/exploitation trade-off introduced in Sec. 2.1.1, given the definition of optimal policy provided in Eq. (2.19), it is possible to define stochastic policies which allow to balance the exploration/exploitation trade-off. The first one introduced is the ϵ-greedy policy which is defined as

$$\pi_{\epsilon\text{-greedy}}(a \mid s) = \begin{cases} 1 - \epsilon & \text{if } a = \arg\max_{a \in \mathcal{A}} Q(s,a) \\ \epsilon & \text{otherwise} \end{cases} . \tag{2.22}$$

with $\epsilon \in [0, 1]$. It is worth noting that if $\epsilon = 0$, it is a fully greedy policy and the agent is only taking advantage of the Q values that it has learned by choosing always the action with the maximum expected return. On the other hand, if $\epsilon > 0$ the agent is acting greedily $1 - \epsilon$ of times while for a fraction ϵ of times it chooses another action uniformly between the non-optimal ones. The fact that sometimes the agent chooses non-optimal

actions allows exploration. Another possibility is given by the softmax policy defined as

$$\pi_{\text{softmax}}(a \mid s) = \frac{e^{\frac{Q(s,a)}{\tau}}}{\sum_{a' \in \mathcal{A}} e^{\frac{Q(s,a')}{\tau}}} \tag{2.23}$$

where τ is a temperature parameter. For $\tau \to \infty$ the policy converge to a uniform distribution over the actions, meaning that the agent is only exploring. For $\tau \to 0$ the softmax distribution converges to the hardmax distribution where the agent is fully exploiting and not exploring. Typically, during a training procedure of a RL agent, the parameters ϵ and τ which controls the exploration/exploitation trade-off might change because in the first part of the training the agent should explore more to acquire a lot of knowledge, whereas in the final part it should only rely on the learned policy without the need to explore anymore.

2.3 Q-learning and Deep Q-learning

Previously, it was shown how, if the optimal state-action values $Q_\star(s, a)$ are known by the agent, it is able to define an optimal policy. However, in the RL problems these values are not available but have to be learned in the training procedure. One of the most common methods which allows to learn the optimal values is Q-learning. This method is derived by temporal difference (TD) learning and has been introduced by Watkins [18]. In particular, Watkins showed how starting from random initial values and updating these values according to

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \Big] \tag{2.24}$$

it is possible to converge to the optimal $Q_\star(s, a)$ values. In Eq. (2.24), the parameter α is the learning rate and the proof holds for $0 \leq \alpha_n < 1$ and

$$\sum_{i=1}^{\infty} \alpha_n^i = \infty, \quad \sum_{i=1}^{\infty} (\alpha_n^i)^2 < \infty. \tag{2.25}$$

Although the proof is not provided in this work, it is possible to appreciate a similarity between the Bellman optimality equation for the $Q(s, a)$ values and Eq. (2.24). Indeed, in the Q-learning rule the values are iteratively updated towards the quantity $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$ which is the best estimation of the return G_t according to the current values $Q(s, a)$ available.

The procedure to learn the optimal values and thus the optimal policy through the

Algorithm 2.1 Q-learning algorithm

$\alpha \in (0, 1], \epsilon \in (0, 1]$
Initialize $Q(s, a)\ \forall s \in \mathcal{S},\ a \in \mathcal{A}$ arbitrarily
for each episode:
  Initialize s
  while s is not terminal:
    Choose a from s using policy derived from Q (e.g. ϵ-greedy)
    Take action a, observe reward r, and next state s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha\Big[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\Big]$
    $s \leftarrow s'$
  end while
end for

Q-learning rule is described in Alg. 2.1. In the algorithm there is a *for* loop over all the episodes, at the beginning of the episode the state is initialized according to some criteria, for example a random state. After that, at each time step, the agent selects an action according to the current Q values and an ϵ-greedy policy. After the action is taken, the agent receives the reward r and observes the next state s'. After that, the value corresponding to state s and action a $Q(s, a)$ is updated using Eq. (2.24). Finally, the next state s' becomes the current state until the agent reaches a terminal state. For an infinite number of episodes, the Q values converge to the optimal ones and thus the policy of the agent converges to the optimal policy $\pi_\star$. There exist other approaches based on Monte Carlo estimations to learn the Q values, however Q-learning performs usually very well, has lower variance and can be implemented online without waiting the end of each episode to update the values [1].

Despite being simple and appealing to use, Q-learning has one main drawback when used in continuous control scenarios or high cardinality state spaces $\mathcal{S}$. Q-learning is a tabular method and the agent has to store a state-action value for every possible combination of a state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$. The number of values can grow a lot in some cases, leading to very slow convergence and requiring a lot of examples. To solve this issue, the Q values are replaced by a Q_θ function parametrized by some parameters θ. Recently, Deep Neural Networks have been proved to be very good for parametrizing these value functions [19]. One of the most used algorithm is Deep Q-learning [8]. Deep Q-learning inherits the idea from the Q-learning rule and replaces the tabular Q values with value function approximation. Theoretical limits of RL with value function approximation are evaluated in [1].

Considering the case where the agent knows the optimal $Q_\star(s, a)$ values, the weights θ

of the neural network Q_θ which parametrizes the value function should be optimized to fit the optimal values

$$\theta = \min_\theta (Q_\star(s,a) - Q_\theta(s,a))^2 \tag{2.26}$$

However, in the training scenario, the agent does not know the optimal values and estimates them through the same target of Q-learning

$$y(s,a) = r + \gamma \max_{a'} Q(s',a') \approx Q_\star(s,a). \tag{2.27}$$

Eq. (2.26) expresses the fitting problem with a convex objective function leading to a least squares optimization. An algorithm to find the optimal weights of Q_θ is stochastic gradient descent (SGD). This method is very appealing because it is known to work very well in convex optimization [20] but also to train neural networks [21]. However, it requires batches of samples to update weights and not only one sample as in Alg. 2.1. In order to store batches of samples, the agent has to store a memory of the past states, actions and rewards in a **Replay memory buffer** D. The memory capacity should be larger than the batch size N because SGD requires the samples to be independent and identically distributed, but the sequence of states and actions of RL agent are highly correlated. Thus, the memory buffer D should be able to store a high number of samples so that the batch can be randomly sampled from the memory reducing the correlation between samples. Given the replay memory $D = \{(s_0, a_0, r_0, s'_0), (s_1, a_1, r_1, s'_1), ...\}$ and the network Q_θ, the optimization step of the SGD procedure becomes

$$\Delta\theta = -\eta \nabla_\theta \mathcal{L}_\theta = -\eta \nabla_\theta \frac{1}{N} \sum_{i=0}^{N-1} [Q_\theta(s_i, a_i) - (r_i + \max_{a'} Q^-(s'_i, a'))]^2. \tag{2.28}$$

where η is the learning rate of SGD. Note that the target value is estimated through Q^- and not using the Q_θ. This is due to the fact that the target is an estimation of the optimal value which does not have to be optimized. Q^- is called the **target network** and is initialized with the same weights of Q_θ (the **policy network**). In Alg. 2.2, Deep Q-learning procedure is shown. Note that the optimization procedure can start only when enough experience (enough samples) are accumulated in D. Another thing to take care, is to update the weights of the target network every C steps and not at every iteration of SGD. This is because updating them too frequently can lead to instability in the training phase. The algorithm terminates after a fixed number of episodes or after a certain performance score is reached by the agent.

The main difficulty when using Deep Q-learning is to keep the procedure stable and

Algorithm 2.2 Deep Q-learning algorithm

Initialize policy network Q
Initialize target network Q^-
Initialize experience replay memory D
Initialize the Agent to interact with environment
while not converged
  $\epsilon \leftarrow$ set the new epsilon with ϵ-decay
  Choose an action a from state s using ϵ-greedy policy
  Agent takes action a, observe reward r and next state s'
  Store transition $(s, a, r, s', done)$ in the experience replay memory D
  if enough experience in D
    Sample a random minibatch of N transitions from D
    for every transition $(s_i, a_i, r_i, s'_i, done_i)$ in minibatch
      if $done_i$
        $y_i = r_i$
      else
        $y_i = r_i + \max_{a' \in \mathcal{A}} Q^-(s'_i, a')$
      end if
    end for
    Calculate the loss $\mathcal{L} = \frac{1}{N} \sum_{i=0}^{N-1} (Q(s_i, a_i) - y_i)^2$
    Update Q using SGD algorithm by minimizing loss $\mathcal{L}$
    Every C steps, copy weights of Q to Q^-
  end if
end while

converge to an optimal value. Indeed, the two step procedure which separately updates the policy network and the target network can be very unstable. One solution introduced in [14] is to update the target network slowly with an Exponential Moving Average (EMA). Given policy network Q_θ with parameters θ and target network Q_ϕ with parameters ϕ and a parameter $\beta \in (0, 1)$, the parameters ϕ are updated as

$$\phi \leftarrow \beta\phi + (1 - \beta)\theta. \tag{2.29}$$

This allows the target network values to converge slowly towards the policy network values avoiding bad updates due to sub-optimal improvements during the training procedure.

2.4 Partially observable Markov Decision Process

MDPs are a very useful description of RL problems and work very well in scenarios where the agent has access to reliable state signals. However, sometimes the state signal provides only a partial description of the state, mostly because of limited sensing capabilities of the agent. Partially Observable Markov Decision Processes (POMDPs) allow for principled decision making under conditions of uncertain sensing [22]. Starting from the definition of an MDP, it is possible to define a POMDP as a tuple $\langle \mathcal{S}, \mathcal{A}, \Omega, P, O, R, \gamma \rangle$ where

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- Ω is a finite set of observations
- P is a state transition probability matrix with entries

$$P^a_{ss\prime} = \Pr[S_{t+1} = s' \mid S_t = s, A_t = a]$$

- $O : \Omega \times \mathcal{S} \to [0, 1]$ is an observation function such that:

$$O(s, o) = \Pr[S_t = s \mid o]$$

- R is a reward function such that

$$R^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$

- $\gamma \in [0, 1]$ is a discount factor

In other words, a POMDP has the underlying dynamic of the system governed by an MDP, but the state is not fully observable by the agent. There are several approaches

to treat POMDPs, but one main issue highlighted in [23] is that standard RL algorithms might fail. For example the authors emphasise that even when only two states are confused, the performance of the RL procedure can degrade arbitrarily. However, authors in [23] shown how, using a stochastic policy and batched version of the Q-learning, it is possible to converge to the optimal values w.p.1. The batched version of the Q-learning algorithm is the same as online Q-learning, but the values are updated after a batch of M samples is acquired. The agent operates on the observation $o \in \mathcal{O}$ and does not have access to the environment state $s \in \mathcal{S}$. Let:

- $M_k(o, a)$ be the number of times action a is executed in observation o within the k^{th} batch of size M
- $n_k(s \mid o, a)$ be the number of times the actual underlying state was s when the observation-action pair was (o, a)
- $n(o, o' \mid a)$ be the number of times a transition took place from observation o to observation o' given action a was executed.

Then the Q(o,a) after the k^{th} batch is collected is given by:

$$Q_{k+1}(o,a) = (1 - \alpha M_k(o,a) Q_k(o,a) + \alpha M_k(o,a) \Big[\sum_{s \in \mathcal{S}} \frac{n(s \mid o,a)}{M_k(0,a)} r_k^a(s) + \gamma \sum_{o'} \frac{n(o, o' \mid a)}{M_k(o,a)} \max_{a' \in \mathcal{A}} Q_k(o', a') \Big] \tag{2.30}$$

where $r_k^a(s)$ is the sample average of the immediate reward received on executing action a in state s in the k^{th} batch. Let

$$F_k(o,a) = \sum_{s \in \mathcal{S}} \frac{n(s \mid o,a)}{M_k(0,a)} r_k^a(s) + \gamma \sum_{o'} \frac{n(o, o' \mid a)}{M_k(o,a)} \max_{a' \in \mathcal{A}} Q_k(o', a') - Q_\star(o,a), \tag{2.31}$$

then, if $V_k(o) = \max_a Q_k(o,a)$ and $V_\star(o) = \max_a Q_\star(o,a)$

$$F_k(o,a) = \sum_{s \in \mathcal{S}} \Big(\frac{n(s \mid o,a)}{M_k(0,a)} r_k^a(s) - P^\pi(s \mid o,a) \Big) R_s^a + \gamma \sum_{o'} \frac{n(o, o' \mid a)}{M_k(o,a)} [V_k(o') - v_\star(o')] + \gamma \sum_{o'} \Big(\frac{n(o, o' \mid a)}{M_k(o,a)} - P^a(o, o' \mid \pi) \Big) V_\star(o'), \tag{2.32}$$

where

$$P^a(o, o' \mid \pi) = \sum_s P^\pi(s \mid o, a) \Big[\sum_{s'} P^a(s, s') O(o' \mid s') \Big]$$

The expected value of $F_k(o, a)$ can be bounded by

$$\begin{aligned} \|\mathbb{E}[F_k(o,a)]\| \leq \gamma \|V_k - V_\star\| + \\ \left\| \mathbb{E}\Big[\sum_{s \in \mathcal{S}} \Big(\frac{n(s \mid o, a)}{M_k(o, a)} r_k^a(s) - P^\pi(s \mid o, a) \Big) R_s^a \Big] \right\| + \\ \gamma \left\| \mathbb{E}\Big[\gamma \sum_{o'} \Big(\frac{n(o, o' \mid a)}{M_k(o, a)} - P^a(o, o' \mid \pi) \Big) V_\star(o') \Big] \right\| \\ \leq \gamma \|V_k - V_\star\| + C \epsilon_k^M, \end{aligned} \tag{2.33}$$

where ϵ_k^M is the largest between

$$\max_{s,o,a} \left| \mathbb{E}\Big[\frac{n(s \mid o, a)}{M_k(o, a)} r_k^a(s) \Big] - P^\pi(s \mid o, a) \right|$$

and

$$\max_{s,o,a} \left| \mathbb{E}\Big[\frac{n(o, o' \mid a)}{M_k(o, a)} \Big] - P^a(o, o' \mid \pi) \right|$$

For any $\epsilon > 0, \exists M_\epsilon$ such that $\epsilon_k^M < \epsilon$. Therefore, for the theorem of convergence of iterative processes [23], for any $\epsilon > 0$, with probability $1 - \epsilon$, $Q_k(o, a) \to Q_\infty(o, a)$, where $|Q_\infty(o, a) - Q_\star(o, a) \leq C\epsilon|$. Thus, the theorem ensures that, if the batch size M is large enough, Q-learning algorithm will learn the optimal state-action values. The random process $F_k(o, a)$, which is the gap between the values at iteration k and the optimal values, is converging (in probability) to zero, given that M is large enough.

Intuitively, the proof works because any action is taken infinitely often in every state (this is ensured by the stochastic policy and the fact that every action has a non zero probability). This proof notwithstanding, it might be difficult for RL procedure to converge in practice, but in general, the standard Deep Q-learning can be used even in POMDP.

2.5 Autoencoders

Sensory input of the RL agent can be of different types and very difficult to understand. This is especially the case when the input is a multi-modal signal such as an image or a video. What is more, if the sensory inputs is received by the agent through a (wireless) communication channel, it becomes clear how being able to compress these sensory inputs

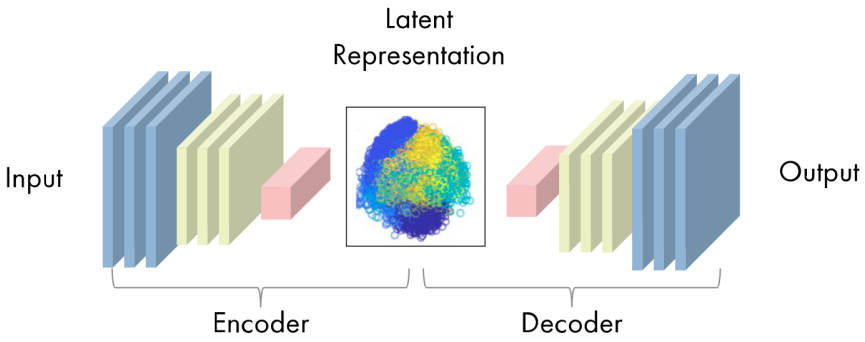

Figure 2.2: A sketch of the architecture of an autoencoder [2]

is extremely important. As a matter of fact, even if input signals are high-dimensional, most of the time they do not cover the entire input space but a lower dimensional manifold. For example, in the case of remote control, the representational domain that the sensor has to be able to communicate to the agent comprises all the possible positions and movements that the agent is doing.

Given this premise, the sensor is interested in learning this lower-dimensional manifold. A Statistical method to achieve this scope is Principal Component Analysis (PCA) [24], [25]. PCA is a linear method, which consists in projecting the input data into the linear principal components of the input dataset which are the ones which maximize the explained variance. However, a linear transformation of the data is insufficient to achieve low dimensional representation for image or video inputs. Starting from the idea of PCA, extensions to non-linear transformation have been found to work very well in the image domain. Some seminal works about non-linear PCA can be found in [26] and [27]. In particular in [27] the authors propose to use a symmetric architecture composed by two neural networks: an Encoder and a Decoder, naming the entire architecture Autoencoder (AE).

An autoencoder is a neural network which transforms the input signal through the stacked layers and then reconstructs the original one in the final layer. In the typical architecture, the layers have a smaller dimension than the input. Consequently, the transformation obtained in the hidden layers approximate the lower dimensional manifold the inputs can be represented with. In Fig. 2.2 a scheme of AE is shown.

In particular the main components of an AE are:

- The **Input** $x \in D$, where D is the dataset.
- The **Encoder** $ENC_\theta(x)$, a neural network (parametrized by θ) which transforms

the input into a lower dimensional vector z

- The **Latent representation** z, the output of the encoder, a low dimensional vector which represents the input.
- The **Decoder** $DEC_\phi(z)$, a neural network (parametrized by ϕ) which reconstructs the original input from the latent representation z.
- The **Output** $\hat{x}$, the reconstructed input at the output of the decoder.

The AE is very efficient to train as it is possible to optimize all the parameters by minimizing a single loss function. In particular, given an input x and the reconstructed input $\hat{x}$, the training of the AE can be performed by minimizing the Mean Square Error (MSE) between x and $\hat{x}$, leading to the following optimization problem

$$\theta, \phi = \min_{\theta,\phi} \frac{1}{|D|} \sum_{x \in D} \|x - \hat{x}\|_2^2 = \min_{\theta,\phi} \frac{1}{|D|} \sum_{x \in D} \|x - DEC_\phi(ENC_\theta(x))\|_2^2. \tag{2.34}$$

This optimization problem can be efficiently solved by defining the loss function

$$\mathcal{L}_{\theta,\phi} = \frac{1}{|D|} \sum_{x \in D} \|x - \hat{x}\|_2^2. \tag{2.35}$$

and minimizing it through SGD.

In order to deal with images, the layers often used in the encoder and decoder networks are **Convolutional Layers**. Introduced by LeCun and Bengio in [28], convolutional layers exploit spacial correlation of input pixels and shared weights to use a smaller number of parameters with respect to a fully connected layer. Several contributions has been brought during the recent years making convolutional networks an excellent tool for image classification and visual input representation. The most important works are for example AlexNet [29], VGGNet [30], inception layers introduced in GoogLeNet [31] and residual layers introduced in ResNet [32].

Note that AE are generative models since they learn a latent representations of the dataset which can be used also to generate new samples starting from random point in z.

2.6 Variational Autoencoders and compression

Autoencoders are a powerful tool which someone might exploits for different purposes, however, the latent vector z is not properly a "compressed" version of the input image x. Compression means represent the input x with a smaller number of bits. A formal

definition of the average number of bits for representing a variable x is the Shannon entropy $H(X)$ [33]. Given a random variable $x \in \mathcal{X}$ with p.m.f. $p(x)$ the Shannon entropy of x is defined as

$$H(X) = \mathbb{E}[-\log p(x)] = \sum_{x \in \mathcal{X}} p(x) \log \frac{1}{p(x)} \tag{2.36}$$

where X is used to represent the random process and x the realizations of this process. From this definition, it is clear how the latent of an AE is not guaranteed to be a compressed version of the input. Indeed ENC_θ is a deterministic function of the input and thus

$$H(X, Z) = H(X) - \underbrace{H(Z \mid X)}_{=0 \text{ (since } z = ENC_\theta(x))} = H(Z) - \underbrace{H(X \mid Z)}_{\geq 0} \tag{2.37}$$

where $H(X, Z) \geq 0$ is the joint entropy and $H(X \mid Z)$ is the conditional entropy. From Eq. (2.37) it is possible to derive

$$H(X) \geq H(Z) \tag{2.38}$$

with equality if and only if ENC_θ is a bijective function on $\mathcal{X}$. Although it might be that the latent space has a lower entropy, in the standard AE there is no guarantee of how much smaller it is, meaning there is no way to control the compression.

In order to measure the compression, Shannon introduced the rate-distortion theory [33]. Given a source $\mathcal{X}$ and a distortion measure $d(x, \hat{x})$ (for example the mean squared error), the information rate distortion function is

$$R(D) = \min_{p(\hat{x}|x) \in \Gamma_D} = I(X, \hat{X}) \tag{2.39}$$

where $\Gamma_D = \{p(\hat{x} \mid x) : \mathbb{E}[d(x, \hat{x})] \leq D\}$ and $I(X, \hat{X})$ is the mutual information between the process X and $\hat{X}$. Since the reconstructed input $DEC_\phi(z) = \hat{x}$ is a deterministic function of z, it is possible to write also

$$I(X, \hat{X}) \leq I(X, Z) \tag{2.40}$$

Thus, reducing the mutual information between the input and the latent space actually reduces the rate with which the input can be encoded and thus is a proper compressed version of the input, in the sense that it is possible to represent z with an average lower number of bits with respect to x.

In order to practically optimize the rate and realize an AE where it is possible to arbitrarily reduce the mutual information $I(X, Z)$, it is possible to use a **Variational Autoencoder** (VAE). Proposed by Kingma et al. in [34], a variational autoencoder is an AE where the encoder is replaced by a parametrical encoding distribution $p_\theta(z \mid x)$. In a VAE the input is encoded into an encoding distribution, then a latent vector z is sampled from this distribution and in the final stage, the input is reconstructed from z with the decoder (as in AE). Below it is represented the forward pass of a VAE

$$\underbrace{x}_{\text{input}} \rightarrow \underbrace{p_\theta(z \mid x)}_{\text{encoded distribution}} \rightarrow \underbrace{z \sim p_\theta(z \mid x)}_{\text{sample } z} \rightarrow \underbrace{\hat{x}}_{\text{reconstructed input}} .$$

The loss function of a VAE can be written as

$$\mathcal{L}_{\theta,\phi} = \mathbb{E}[\|x - \hat{x}\|_2^2] + \beta D_{KL}\big(p_\theta(z \mid x) \| p(z)\big) \tag{2.41}$$

where $D_{KL}\big(p_\theta(z \mid x) \| p(z)\big)$ is the Kullback-Leibler divergence between the encoding distribution $p_\theta(z \mid x)$ and a target distribution $p(z)$. It is possible to show that

$$D_{KL}\big(p_\theta(z \mid x) \| p(z)\big) = \mathbb{E}\Big[\log \frac{p(z \mid x)}{p(z)}\Big] = \mathbb{E}\Big[\log \frac{p(z, x)}{p(x)p(z)}\Big] = I(X, Z), \tag{2.42}$$

which means that in Eq. (2.41) the mutual information $I(X, Z)$ is actually minimized. The parameter β controls the amount of compression and allows to optimize both the reconstruction loss and the rate constraint similarly to lagrangian multiplier [35]. However, in practice, the optimal target distribution $p(z)$ is not known and is replaced by a prior distribution such as the normal distribution $\mathcal{N}(0, 1)$ [34].

Despite being very similar to the AE, the sampling step is not differentiable, thus when training the VAE, it is not possible to backpropagate the error as it is done in the standard AE. To overcome this problem, it is possible to use the reparametrization trick [34]. Instead of encoding the input into a distribution, the encoder estimates the mean μ_x and the standard deviation σ_x of the distribution and then z is sampled as

$$z = \mu_x + \epsilon\sigma_x, \tag{2.43}$$

where $\epsilon \sim \mathcal{N}(0, 1)$. In this way, z is differentiable with respect to μ_x and σ_x since the sampling of ϵ does not depend on the parameters of the encoder.

2.7 Quantization and Vector Quantization

In previous sections, VAEs have been proved to provide compressed representations of high dimensional inputs. However, coding those representations to transmit them easily through a communication channel requires some further considerations. Indeed, in the standard formulation of VAE its latent space is a continuous vector space $z \in \mathbb{R}^N$. In order to communicate the values of the latent space, they have to be quantized. Quantization allows us to represent continuous intervals with a finite number of bits. There exist two types of quantization:

- **Scalar Quantization**: quantize each sample of the vector at a time.
- **Vector Quantization**: quantize group of samples jointly.

Vector quantization (VQ) can perform better than the scalar one, as it can exploits the correlation between the samples it is jointly quantizing. A VQ is specified by:

1. A set of decision regions or cells

$$I_i \subset \mathbb{R}^N, \quad i = 1, ..., K \quad \text{where } I_i \cap I_j = \text{ for } i \neq j \text{ and } \bigcup_{i=1}^{K} I_i = \mathbb{R}^L$$

2. a set of code vectors (also called codewords)

$$y_i \in \mathbb{R}^L, \quad i = 1, ..., K$$

3. A quantization rule

$$Q : \mathbb{R}^L \to \{y_1, ..., y_K\} \text{ such that } Q(x) = y_i \text{ if } x \in I_i$$

When quantizing the vector x with the quantizer Q and the codewords y_i it means that each vector belonging to the region space I_i is represented with the vector y_i. Of course when quantized, the values of the vector are distorted by a certain amount which is

$$d(x, y) = \sum_{n=1}^{L} (x_n - y_n)^2 = \|x - y\|_2^2.$$

Given a distribution of vectors x with p.d.f. $f(x)$ the MSE of the quantization Q is given by

$$\text{MSE} = \frac{1}{L}\mathbb{E}\big[d(x, y)\big] = \frac{1}{L}\int_{\mathbb{R}^L} \|x - Q(x)\|_2^2 f(x)dx = \frac{1}{L}\sum_{i=1}^{K}\int_{I_i} \|x - y_i\|_2^2 f(x)dx. \quad (2.44)$$

There exist two conditions which are necessary for a quantization to be optimal. It can be shown also that if the p.d.f. $f(x)$ is sufficiently smooth these are also sufficient conditions [36].

1. **Nearest Neighbor Condition**: given the set of vectors y_i for $i = 1, ..., K$ the optimal portion I_i of $\mathbb{R}^L$ is

$$I_i = \left\{ x : \|x - y_i\|_2^2 \leq \|x - y_j\|_2^2 \text{ for } i \neq j \right\}$$

2. **Centroid Condition:** given the partitions of $\mathbb{R}^L$ $\{I_i : I_i \subset \mathbb{R}^L \ , \ i = 1, ..., K\}$ the set of codewords that minimizes the distortion is

$$y_i = \frac{\int_{I_i} x f(x) dx}{\int_{I_i} f(x) dx} = \int_{I_i} x f(x \mid x \in I_i) dx \quad \text{for } i = 1, ..., K$$

Those two conditions can be used to solve algorithmically the optimal quantization problem. Introduced by Linde, Buzo and Gray in [37] the LBG algorithm for learning vector quantization is a modified version of the Lloyd Max algorithm [25] which exploits an expectation/maximization (EM) procedure to converge to a local optimal solution.

In the LBG algorithm, the two conditions mentioned are iteratively satisfied until the regions and the codebook converges to an optimal solution. Of course, most of the times, the p.d.f. of x is unknown and a sampled version of the procedure is used. In Alg. 2.3 the discrete version of the LBG is given. Instead of having a distribution $f(x)$ of the vectors, a dataset X of them is collected and the integrals are replaced by summations over the dataset. The termination condition $\frac{D^{(n-1)} - D^{(n)}}{D^{(n)}} < \epsilon$ is satisfied when the distortion D does not improve anymore, meaning that a local minima is being reached. This simple iterative procedure is very similar to K-means clustering [25]. The reason why it converges to a local minima and not to a global one is that vector quantization and more in general k-means clustering is known to be a NP-hard problem, so the general approach is to use an approximated solution.

2.8 Vector Quantized Variational Autoencoders

VQ can be useful to encode the latent space of a VAE into a discrete distribution, which is easier to be transmitted through a communication channel. However, if the quantization is performed after the VAE has been trained, it results in a drop in performance. This is due to the fact that the decoder is not trained on the quantized version of the latent vectors z^q and thus might not generalize well. Also the VQ procedure optimizes the codebook

Algorithm 2.3 LBG algorithm

Given $X = \{x_1, ..., x_N\}$ a set of vectors
Start an initial codebook $\{y_i^{(0)} \ , \ i = 1, ..., K\}$
Initialize distortion $D^{(0)} = \infty$, $n = 1$ and $\epsilon > 0$
for each iteration
Compute the decision cells

$$I_i^{(n)} = \{x \in X : \|x - y_i^{(n-1)}\|_2^2 \leq \|x - y_j^{(n-1)}\|_2^2, \forall i \neq j\}$$

Compute the new codebook

$$y_i^{(n)} = \frac{1}{|I_i^{(n)}|} \sum_{x \in I_i^{(n)}} x$$

Compute the distortion

$$D^{(n)} = \frac{1}{|X|} \sum_{x \in X} \|x - Q(x)\|_2^2$$

if $\frac{D^{(n-1)} - D^{(n)}}{D^{(n)}} < \epsilon$
Terminate
else
$n \leftarrow n + 1$
end if
end for

towards a good reconstruction of the latent vectors and not a good reconstruction of the input.

If the decoder is trained with the quantized (discrete) version of the latent vectors, it might learn a better decoding function than a two step procedure. Recalling the previous forward computation pass of the VAE, when the latent space is a discrete distribution, it becomes

$$\underbrace{x}_{\text{input}} \rightarrow \underbrace{z}_{\text{encoded input}} \rightarrow \underbrace{Q(z) = z^q}_{\text{quantize } z} \rightarrow \underbrace{\hat{x}}_{\text{reconstructed input}} .$$

where the sampling step is replaced by the quantization. Indeed, given the K centroids y_i for $i = 1, ..., K$ the quantized version of the latent vector is

$$z^q = y_i \ , \quad \text{where } i = \arg\min_j \|z - y_j\|_2^2 \tag{2.45}$$

The same problem as in the VAE arises backpropagating the error through the layers of the autoencoder when the quantization step is added. Indeed, the $\arg\min_j$ function is not differentiable, thus some expedients are needed.

2.8.1 Straight through estimation of the gradient

Proposed by Bengio in [38], straight through estimation is a way to allow differentiation in the backward pass of the training of a neural network. The idea is to rewrite the quantized latent vector as

$$z^q = z + \text{sg}(z^q - z) \tag{2.46}$$

where sg() indicates the stop gradient operator, meaning that the quantity $\text{sg}(z^q - z)$ is not taken into account in the computation of the gradient. This makes it possible to compute the gradient of the loss function with respect to z which depends only on the parameters of the encoder θ and not the derivative with respect to z^q which is obtained through a non differentiable operation. The quantized vector is being expressed as

$$z^q = z + q \tag{2.47}$$

where q is the quantization error which does not depend on the parameters of the network. Moreover, it is possible to show that the quantization error has zero mean and is uncorrelated with the latent vector z.

Exploiting the straight through trick, it is possible to train a VAE with a quantization layer. However, one final step is lasting in order to properly train this modified VAE. Indeed, the codewords are not updated during the minimization of the loss function $\mathcal{L}_{\theta,\phi}$

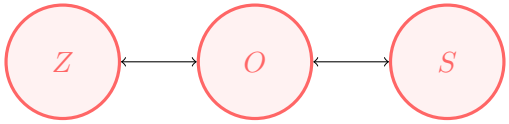

Figure 2.3: Markov constraint on the three processes which are involved in the model

since they are excluded from the gradient estimation.

An architecture which solves this issue is presented in Vector Quantized Variational Autoencdoers (VQ-VAE) [4]. In this VAE, the continuous (Gaussian) latent distribution $p_\theta(z \mid x)$ is replaced with a discrete distribution as explained before.

2.9 Compression and Information Bottleneck

In order to optimize the the performance of the system, the sensor needs to implement an efficient and agile encoding of the observations o_t. Moreover, it is fundamental that this encoding compresses the observed frames in order to use less communication resources to communicate the state to the agent. There exist two types of compression: lossless and lossy compression. Obviously, in this scenario, the most suitable approach for the Sensor is to implement a lossy encoding. The reason is that the agent can tolerate to receive less information than the one contained in the observed images or video and still be able to solve the problem. For example, if the agent receives a less detailed version of the state with a lower definition, it might still be able to retrieve the correct actions to take.

In the system model considered on this project, it is possible to define three processes: one is the process of the observations O, one is the process the encoded messages Z and one is the process of the abstract states S. In the diagram in Fig.2.3 the markov constraint on the three processes is represented, meaning that $Z \leftrightarrow O \leftrightarrow S$. The observation process O is what the Sensor sees, Z is the encoded version of the observation and S is an abstract representation of the state of the MDP that the agent has to solve. In general, the observation of the Sensor is a high dimensional multimodal signal that contains more than the information needed by the agent to solve the task. On the other hand, S represent an abstraction of the state which represents the simplest description of the state according to some metric. In order to compress the observation O, the sensor aims to find an encoding distribution $p(z \mid o)$ such that

$$p(z \mid o) = \min_{p(z|o)} I(O, Z) - \beta I(Z, S). \tag{2.48}$$

This corresponds to the Information Bottleneck problem proposed in [39]. Finding this en-

coding distribution $p(z \mid o)$ means optimizing the encoding distribution such that $I(O, Z)$ is minimal and $I(Z, S)$ is maximal. This means that process Z discards bits of information from O as much as possible while keeping the maximal amount of information about S.

2.10 Three levels of communication problems

The definition of the abstract process can be analyzed from different perspectives according to the three levels of communication introduced by Shannon and Weaver in [10]. In particular, these levels are:

- **Level A.** The technical problem
- **Level B.** The semantic problem
- **Level C.** The Effectiveness problem

Level A corresponds to how accurately the symbols of communication are transmitted; this is the lower level of abstraction as the accuracy between the symbols of the sent message and the received message is considered. Level B considers a semantic definition of the information contained in the received message. In this case, the abstract process is represented by the correct interpretation of the desired meaning from the received messages. Of course, it is not trivial to define this semantic content univocally, as this implies the existence of a shared knowledge between the sender and the receiver. The last level, level C, is concerned with the success with which the meaning conveyed to the receiver leads to the desired conduct on his part. This means that the communicated messages has to allow the receiver to effectively adapt its behavior in the desired way. Despite methods developed by Shannon apply to Level A, in [10] Weaver emphasises how theory developed for the first level, applies to a significant degree also to level B and C.

3
Related Works

This chapter will concentrate on the relevant works which investigated different aspects of the considered problem. Literature has been included to contextualize the scientific area this work will contribute to.

3.1 Joint source channel coding

Joint source channel coding is a paradigm in the designing of communication systems where the underlying idea is to learn an optimal source coding together with an optimal channel coding. The separation theorem, proved by Shannon in [33], gives theoretical guarantees that the separate optimization of source compression and channel coding can approach the optimal performance in the asymptotic limit (size length $N \to \infty$). However, Authors in [3] claim that there exist multiple scenarios where this limit is very difficult to achieve in case of non-ergodic sources or channel models. In a similar work [9], Authors stated that in many emerging scenarios, such as Internet of Things (IoT) systems, there is no need to reconstruct the original input. For example, in [9], an identification task based on face recognition needed to be carried out at the receiver. It has been noted that since the task of face recognition is done through feature extraction, the receiver can work in the feature space without implementing the decoder. Thus, deep neural networks are employed to extract the features, then those features are encoded and transmitted on the channel. At the receiver, the features are reconstructed and directly sent in input to a classifier. It is clear how this procedure is aimed at learning jointly a good source coding which can be optimized together with the channel coding scheme. The channel becomes

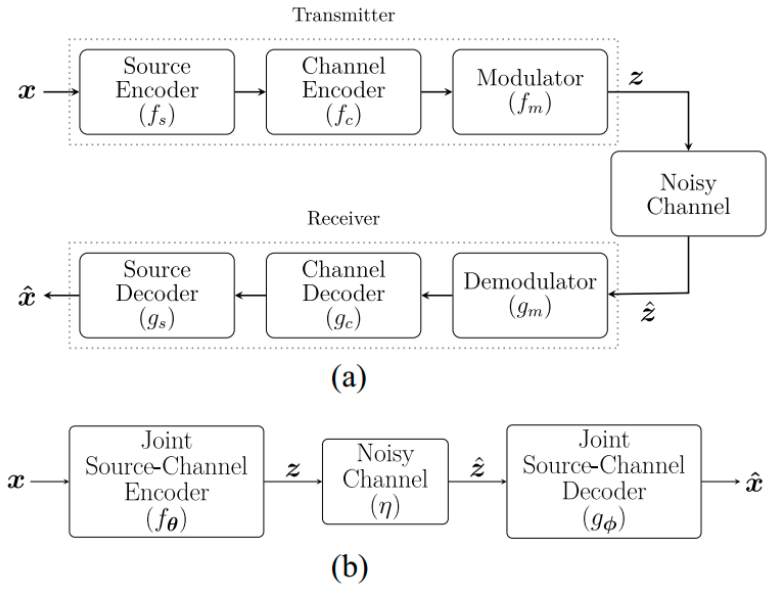

Figure 3.1: Block diagram of the point-to-point image transmission system: (a) components of the conventional processing pipeline and (b) components of the proposed deep JSCC algorithm, [3].

just a layer in the whole deep neural network architecture. In this work, researchers advocate for task-based compression and coding, intending that intelligent coding schemes can be obtained optimizing the compression subject to the task that must be performed.

In Fig. 3.1, a schematic view of the paradigm introduced by joint source channel coding is represented. Training this architecture end-to-end can lead to better performance in the case of task-based problems, where the receiver can use directly the latent space representation z of the input x. One main thing pointed out by the Authors is that the channel model might vary over time and thus using a fixed rate channel coding scheme can suffer from the "cliff effect". This name refers to the condition where the reconstruction quality is not improved when the Signal to Noise Ratio (SNR) of the channel improves.

3.2 Context-Aware Cyber-Physical systems and Digital Twin

Another relevant contribution is given by context-aware communication systems. The context is given by some side information that the sender can acquire in different ways. For example, there might be distributed sensors which inform a cloud server of the situation

of different parts of the environment. In [40], a vehicular network with a cloud service assisting system is considered. The cloud service exploits the information coming from the traffic measures, free parking slots and road conditions, to provide indication to the vehicles. This system can be designed in multiple layers deciding where the context-based decision has to be done. For example, the cloud server analyzes the environment condition and provides only the relevant information for the vehicles to optimize their paths or to find a parking slot. This reduces the computational burden on the vehicles, which do not have to process all the environmental information. Another recent work proposed the idea of Digital Twin, coming from the literature of Artificial Intelligence (AI) [41]. A digital twin is a virtual representation of a physical system, which helps in real-time prediction, optimization, monitoring, controlling, and improved decision-making through simulations. Similarly to context-based systems, the underlying idea of these models is that the sender can simulate and predict the dynamic of the physical system and thus provide to the system only the relevant information. In the paper proposed [41], a real implementation of a remotely controlled robotic arm is presented. The information retrieved by the digital twin is a prediction of the movements of the physical robotic arm. The digital model is a Long Short Term Memory (LSTM) which exploits the autoregressive nature of the (x, y, z) coordinates of the arm position to predict the next coordinates. This can be seen as a sequence-to-sequence prediction task in machine learning theory.

3.3 Control Under communication Constraints

Previously mentioned works provided a unified view of machine learning architectures and communication systems. However, these powerful machine learning models often lack in theoretical bounds on the feasibility of the problem. In [42], a characterization is given based on information theory. Authors recognize that in a sender-receiver system, where the receiver must accomplish a control task with respect to the received information, the choices at the encoding might depend on the policy of the receiver. This argument holds also for the receiver which has to adapt its control policy to subject to the information received. The paper considers a continuous linear control problem

$$X_0 \in \Lambda_0, \quad X_{t+1} = AX_t + BU_t, \tag{3.1}$$

which is observed through a noiseless digital communication channel. The observation process is

$$Y_t = CX_t. \tag{3.2}$$

The state is encoded in a message with m chosen between 2^R possible messages, meaning that R bits can be accurately transmitted through the communication channel. This means that each state X_t is transmitted with an error which is called e_t. The Authors proved that for the system to be asymptotically observable a necessary condition is that

$$R \geq \sum_{\lambda(A)} \max\{0, \log|\lambda(A)|\}. \tag{3.3}$$

The work considers two classes of encoders

- Class 1: the encoder knows the control signal U_t,
- Class 2: the encoder does not know the control signal but knows the control policy.

In the work two results for the optimal encoder, decoder and controller are derived. Despite being a quite restrictive setting, this work provides a bound on the rate in order to guarantee control performances. In another similar work developed in [43], a stochastic control problem is considered deriving analogous bounds. Despite these interesting results, the setting of the system is quite restrictive and does not generalize well to non-linear control or to partially observable models.

4

System Model and practical implementation

In this chapter, the system model analyzed in this project is presented. In the first part, a description of the setting and the building blocks is provided. Subsequently, the strategies adopted to solve the problem are analyzed, discussing relevant choices based on literature and implementation limits. The system model considered in this work comprises a system with two agents which interact in the same environment. However, the interaction happens only through a communication channel: one agent called the **Sensor** only senses the environment through a camera, while the **Controller** agent cannot see the environment and interacts in the environment taking actions.

In Fig. 4.1 the blocks of the system model considered are represented. The underlying dynamic of the environment can be described with an MDP as presented in 2.2. Therefore, the camera can not see the true state of the system, but it only observes some frames of the environment where the agent is represented. In particular, using the same notation

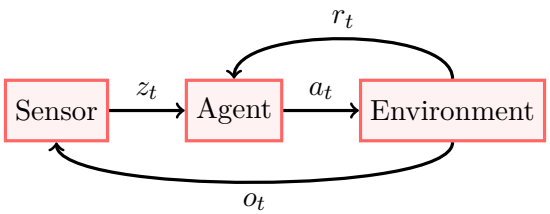

Figure 4.1: System model considered in this project.

used for the POMDPs in Sec 2.2, the sensor observes o_t, incomplete information of the environment. Then it must encode these frames in a message z_t which is communicated to the agent using a communication channel with finite capacity. The controller learns a policy based only on the received messages. When the Controller takes action a_t in state s_t it receives a reward r_t. This way, the Controller can learn a policy which maximizes the cumulative rewards as in typical RL scenarios. The Sensor chooses what to communicate to the Controller based on the observations it senses, while the Controller takes actions according to the messages received from the Sensor. The flow of information is similar to the one described in the section of the information bottleneck problem in Sec. 2.9 . Indeed, the Sensor wishes to optimize the communication resources used to communicate to the controller. On the other hand, the Controller needs to receive a certain amount of information about the observations in order to learn to control the agent.

4.1 The CartPole environment

In this project a "toy" control problem was considered. The Environment used is the CartPole environment provided by the *OpenAI Gym* python library [44]. The control problem consists of an inverted pendulum which has to be kept vertical and centered, moving a Cart to the left or to the right. The state is described by four values:

- Cart Position,
- Cart Velocity,
- Pole Angle,
- Pole Angular velocity.

At the beginning of each episode, the four values are initialized at random sampling from a uniform distribution $\mathcal{U}(-0.05, 0.05)$. Note that the Cart position takes values in the range (-4.8, 4.8). At the beginning, the Cart is in the center of the scene with the pole almost vertical and a small initial velocity. The two possible actions that the agent can take are to push the Cart either to the left or to the right. The velocity that is reduced or increased by the applied force is not fixed and it depends on the angle the pole is pointing. The center of gravity of the pole varies the amount of energy needed to move the cart underneath it. The episode terminates when the Cart exits the position range (-2.4, 2.4) or the pole angle is not in the range (-.2095, .2095) [rad], corresponding to $\pm 12°$.

The reward function is obtained as the sum of two terms:

- A position penalty: $r_1 = \frac{x_{max} - |x|}{x_{max}} - 0.8$,

- An angle penalty: $r_2 = \frac{\theta_{max} - |\theta|}{\theta_{max}} - 0.5$.

x is the current position of the Cart, $x_{max} = 4.8$, θ is the current angle of the Pole and $\theta_{max} = 0.418$[rad]. At each time step t, the environment returns a reward $r_t = r_1 + r_2$ according to the current position x and angle θ. The frequency with which the agent chooses an action is $F = 50$ Hz.

This environment has been proposed for the first time in [45]. The equations governing the dynamic of the system are:

$$\ddot{x} = \frac{F_t + ml[\dot{\theta}_l^2 \sin\theta_t - \ddot{\theta}_t \cos\theta_t] - \mu_c \text{sgn}(\dot{x}_t)}{m_c + m},$$

$$\ddot{\theta}_t = \frac{g \sin\theta_t + \cos\theta_t \left[\frac{-F_t - ml\dot{\theta}_t^2 \sin\theta_t + \mu_c \text{sgn}(\dot{x}_t)}{m_c + m}\right] - \frac{\mu_p \dot{\theta}_t}{ml}}{l\left[\frac{4}{3} - \frac{m\cos^2\theta_t}{m_c + m}\right]}.$$

In the problem considered there is no access to the four values describing the state of the agent. In fact, the Sensor observes the environment through video frames which are refreshed every time an action is chosen, so the video has the same frequency of $F = 50$ Hz. The frame is an RGB image of size $[400 \times 600 \times 3]$. It is worth noting that even if there was not any communication channel between the Sensor and the Controller, the problem would still be a partially observable problem. Indeed, the initial values are set randomly at the beginning of the episode, but the Sensor observes only one (static) frame. There is no possibility to obtain information about the velocity and angular velocity. Only an estimation on the Pole angle and Cart position can be done by the Sensor at the beginning of the episode, thus the state is partially observable. More than one frame is needed to estimate these velocities. However, this type of partial observability does not preclude to control the agent from the video frames. After an initial transient, the unknown initial conditions do not influence the system anymore.

4.2 Initial tests

To investigate the feasibility of the control problem and future design choices, some tests are performed considering an ideal communication channel. The first test is to understand whether it is possible to estimate the state parameters from the video frames. This is just a test to understand if it is possible to extract relevant features from the video frames which help the controller to solve the task. Note that this first task does not consider a communication channel between the Sensor and the Controller. This first analysis is a

supervised learning task, more technically it is a regression task. The main objective of the learning algorithm is to learn a function $f_\theta(\cdot)$ which takes in input the observations o_t for $t = 0, 1, 2, ...$ and estimates the four-dimensional vector $s_t = [x_t, \dot{x}_t, \theta_t, \dot{\theta}_t]$, containing the state parameters. Since these values are continuous, this is a regression task.

One first thing to observe is that at least two frames are needed to estimate the velocity. Thus, the function f_θ at time t receives in input o_t and o_{t-1}. The function f_θ is parametrized by a vector of parameters θ which has to be set to solve the following optimization problem

$$\theta^\star = \min_{\theta \in \Theta} \mathbb{E}\big[\|\hat{s}_t - s_t\|_2^2\big], \tag{4.1}$$

where $\hat{s}_t = f_\theta(o_{t-1}, o_t)$ is the output of the function and the estimated vector of the state parameters. In this case, the true state s_t is assumed to be known as an oracle. This is just an initial test to justify the implementation choices; this assumption will not be used in the subsequent scenarios. A neural network is used to do the regression because of its versatility to deal with images. In particular, convolutional layers are used to extract relevant features from the input frames and then the final layer dimension is set to four, since it has to match the dimension of the state vector.

The first thing to do is to acquire a dataset containing a wide variety of frames, so that during the learning procedure the network sees a lot of data and adapts its weights converging to a good solution. To ensure that the function is really able to estimate the true state, for any possible policy used by the Controller, a random policy is used. This means that between frame o_t and frame o_{t+1} in the dataset, the Controller takes an action

$$a_t = \begin{cases} \text{left}, & \text{w.p.} \frac{1}{2} \\ \text{right}, & \text{w.p.} \frac{1}{2} \end{cases} .$$

This random policy ensures that the range of the dynamics collected are not biased by a restrictive policy which only visits certain states. A dataset containing 50000 states is collected. This implies that the dataset contains 50000 tuples of the form (o_t, o_{t+1}, s_{t+1}). Note that a_t is not considered in input in the regression task. This complicates the problem since, at every time step, the action of the Controller applies a force (to the left or to the right) to the CartPole. For example, it is not possible to obtain the velocity $\dot{x}_t$ as $x_{t-1} - x_t$ because the acceleration is not zero due to the action taken. However, the acceleration is typically small if the pole does not oscillates too much and thus a good estimation of the four state values would be possible.

The Neural Network used for the regression task has three convolutional layers and one final linear layer. In Tab 4.1 the layers of the regression network with the respective

Layer	Input	Output	Activation function
1^{st} Conv. layer	2 channels	16 channels	ReLU
2^{nd} Conv. layer	16 channels	32 channels	ReLU
3^{rd} Conv. layer	32 channels	32 channels	ReLU
Flatten layer	$32 \times 18 \times 10$	5760	/
Linear layer	5760	4	Linear

Table 4.1: Layers of the regression network.

	MSE
Training	0,051
Test	0.059

Table 4.2: Mean Square Error of the Training and Test in the regression task.

dimensions are highlighted.

The kernel used in the three convolutional layers is a 2-dimensional 5×5 filter with stride equal to 2. From Tab. 4.1 note that the number of input channels of the first layer is 2. This is because the frames are converted from RGB images to grayscale. Thus, two grayscale images corresponding to o_t and o_{t+1} are given in input to the network. It is possible to do this without loss of generality since the only object in the frame is the CartPole and the shape is perfectly preserved when converted to grayscale. This way, it is possible to reduce the computational burden which derives from using the frames in the RGB space. Also the images are cropped, so that the moving CartPole is perfectly visible but the pixel number is reduced. From a 400×600 image, the actual size of the frame in input to the network is 180×360. The activation function used in between each layer is the Rectified Linear Unit (ReLU), but in the last linear layer a linear activation function is used in order to obtain a range of values which spans from the negative values to the positive ones. After each convolutional layer, batch normalization is used to normalize the input of the following layers.

The network has been trained for 1000 epochs with a batch size $B = 128$, randomly sampled from 8000 samples. The optimizer used is the Adam oprimizer [46] with a learning rate $lr = 10^{-4}$. The remaining 2000 samples were used to test the performance after the training phase. The training and test results are shown in Tab. 4.2.

It can be observed that the MSE is actually low. This can be stated by looking at the ranges of the physical values which are between (-3,3) for the velocity and the angular velocity, (-2.4,2.4) for the position and (-0.2,0.2) for the angle. Indeed, it is possible to retrieve information about the dynamic of the system by looking at the video frames of

Layer	Input	Output
1st Linear layer	4	128
2nd Linear layer	128	128
3rd Linear layer	128	2

Table 4.3: Layers of the Q-function network for the supervised setting.

the environment.

4.3 Training the entire system with supervision

The result obtained in the previous section justifies the first test that was done to train the complete system model with supervision. In this analysis, the possibility of learning a good control policy from the estimated values is being tested. In this scenario, the regression network is already trained and the Controller uses the state estimated by the regression network to learn a control policy for the task. Note that there are no restrictions given by the communication channel. In this case the link between the Sensor and the Controller can be considered an ideal communication channel which is able to communicate values with infinite precision.

To train the Controller, Deep Q-learning is used. To parametrize the Q-function of the Controller a neural network is used. The policy takes in input 4 values ($\hat{s}_t$ estimated by the Sensor) and gives as outputs 2 Q-values which correspond to an estimation of the expected cumulative return for the two possible actions. The network has three layers and the technical details are listed in Tab. 4.3. The nonlinear activation function used after layer 1 and 2 is the hyperbolic tangent activation function, whereas the linear activation function is used at the output of the last layer.

In this setting, the information flow is the following. The Sensor sees the new frame and it estimates the state values using the already trained regression network. After that, the estimated state is used by the Controller as the only available state information to learn the control policy. Thus, the estimated state is used in input to the Controller's Q-function to obtain the Q-values. The Controller uses a softmax policy to choose an action using the estimated Q-values.

As explained in Sec. 2.3, training a RL agent with Deep Q-learning requires an exploration stage where the agent interacts with the environment to collect an experience and to store it in a Replay Memory. In this part, the Controller does not use a random policy, but it uses the softmax policy with a high temperature parameter. In this phase of the training, the Q-network is not updated since the updating step can occur only when

enough experience is acquired.

Algorithm 4.1 RL training algorithm

Given the regression Network $f_\theta(o_{t-1}, o_t)$
Initialize the policy network Q_θ and target network Q^-
Initialize an exploration profile for the parameter β
$n = 0$ set the episode number to 0
for each episode
  Initialize the environment to s
  Obtain the observation o
  Estimate the value of the state $\hat{s} = f_\theta(o, o)$
  while s is not terminal
    $a \leftarrow \pi_\beta(\hat{s})$
    Take action a and observe r, s', o'
    Estimate $\hat{s}'$
    Store $(\hat{s}, a, r, \hat{s}')$ in the Replay Memory
    if Enough memory
      Sample a batch B form the the Replay Memory
      Update the policy network using loss function

$$\mathcal{L}_\theta = \sum_{i \in B} \frac{1}{|B|} (Q_\theta(\hat{s}_i, a_i) - r_i - \max_{a'} Q^-(\hat{s}'_i, a'))^2$$

    end if
    $s \leftarrow s'$
  end while
  if n % 10 = 0
    Update Q^- using Exponential Moving Average
  end if
  $n \leftarrow n + 1$
end for

In Alg. 4.1 the training procedure to train the Controller policy is shown. The Q_θ network and Q^- network are initialized in the same way, with the same weights values. The exploration profile is a sequence of values of the parameter β of the softmax function. To allow for exploration/exploitation trade-off, the initial value is set to 3 and then it decreases exponentially. Given the initial value $\beta_0 = 3$, the temperature value at the n-th episode is

$$\beta_n = e^{-n \log \frac{6\beta_0}{N_{ep}}},$$

where N_{ep} is the total number of episodes. Changing the constant of the formula enables to modify how long the exploration period lasts compared to the exploitation one. with

the respect to the exploitation one. At every episode n the parameter becomes smaller ($\beta_n \to 0$ for $n \to \infty$). This implies that the Controller is exploring in the first part, whereas near the end of the training it is exploiting and refining the policy only considering the experience it has acquired. It is very important for the exploration profile to be smooth, in order to ensure that the trade-off between exploration and exploitation changes slowly and does not affect the stability of the Deep Q-learning.

At every step of the episode the controller samples an action according to the softmax policy derived from the Q_θ network. More specifically,

$$\Pr(a \mid \hat{s}) = \pi_\beta(a \mid \hat{s}) = \frac{e^{\beta Q_\theta(\hat{s},a)}}{\sum_{a'} e^{\beta Q_\theta(\hat{s},a')}}. \tag{4.2}$$

Note that the learning procedure does not use the true state s anywhere in the estimated values. Indeed, in the Replay Memory, the estimated current state $\hat{s}$ and next state $\hat{s}'$ are stored. The memory is implemented as a fixed capacity queue. Samples in the queue enter and are stored until the maximum capacity is reached. When the queue is full, the oldest tuple is discarded and a new tuple can enter the queue. This permits to maintain an updated memory which only remembers the newest experience. In fact, as the Controller learns, bad state action pairs (s, a), associated with low Q-values, will not be selected anymore and there is no need to re-estimate their Q-values. Looking at Alg. 4.1, it is possible to note that the Q_θ network is updated only when there is enough memory in the Replay Memory. This ensures that the Controller explores a little bit before updating the weights. Moreover, sampling a batch B of i.i.d. tuples is crucial to guarantee that SGD converges to good minima.

A detail is that only the tuples where s' is not a terminal state are used: this fact is not shown in the algorithmic procedure to make the pseudocode easier to be read. When a tuple ends with a terminal state, the expected return associated with that state should be zero as the episode terminates. However, the Q_θ network is not able to recognize the terminal state and so this case is just avoided in the updating of the policy network. This solution might not teach to the Controller to avoid terminal states however, since states near to terminal states are associated with low rewards, in the long run the Controller will learn to avoid such states.

The target network is updated every 10 episodes using an exponential moving average formula to change its weights. The parameter controlling this updating is set to 0.5. It was seen that this was a good trade-off since lower values cause instability in the training phase, whereas higher values require many more episodes due to slower convergence.

The agent is trained for $N_{ep} = 3000$ episodes, with a Replay Memory of capacity

$C = 10000$. The minimum size to train (the condition to update the policy network) is 1000, meaning that at least 1000 should be stored in the memory before training Q_θ. The optimizer used to update the parameters of the network is SGD with a learning rate of 10^{-3}. A good control policy was learned by the Controller which was able to control the pole for an average of 308.2 steps. This is a good result since the target for the CartPole control problem is to reach 200 steps.

Another thing which was noted is that it was necessary to re-train the regression network every 100 episodes in order to keep it updated and able to estimate values correctly. This is done to ensure that the regression network is able to correctly estimate all the values that the agent uses to train the policy. Otherwise, it might happen that some states are inaccurately estimated avoiding the Controller to discriminate the action properly.

4.4 No oracle: the Unsupervised Scenario

Given the good results in the training of the Controller, it is possible to recognize that the high-dimensionality of the observation space can be reduced using small dimensional vectors to represent the state and still control the environment optimally. In the previous scenario, it is possible to say that the learning of the these low dimensional representations is done with supervision. The Sensor has access to the exact values describing the dynamic of the environment. This case is unrealistic as in a lot of domains the Sensor is not provided with the true state of the MDP, but it only observes a partially observable state. This case is more realistic as in a proper environment problem the Sensor is only observing the environment without any other knowledge about the underlying dynamic of the system.

However, the results of the supervised approach guarantee that the observations can be encoded with a lower-dimensional signal. This signal can be learned with an unsupervised learning approach, more specifically using a generative model to discover a low dimensional embedding for the observations. This can be done with an Autoencoder (AE), which is presented in Sec. 2.5. The AE is made of an Encoder and a Decoder and the middle layer is the latent space. The Encoder can be realized similarly to the regression network used before. The main idea is still to have two frames in input to a convololutional neural network. After the convolutional part, a linear layer forms the latent space and then a decoder made of deconvolutional layers is used to reconstruct the image from the linear latent space.

In Tab. 4.4 the details of layers of the AE are shown. The three convolutional layers are the same of the regression network: 5×5 1D kernel with stride equal to 2. The latent space is set to 64: this is an arbitrary choice. The activation function is the ReLU

Layer	Input	Output
ENCODER		
1st Conv. layer	2 channels	16 channels
2nd Conv. layer	16 channels	32 channels
3rd Conv. layer	32 channels	32 channels
Flatten layer	$32 \times 18 \times 10$	5760
Linear layer	5760	64
DECODER		
Linear layer	64	5760
Unflatten layer	5760	$32 \times 18 \times 10$
1st Deconv. layer	32 channels	32 channels
2nd Deconv. layer	32 channels	16 channels
3rd Deconv. layer	16 channels	2 channels

Table 4.4: Autoencoder architecture.

	Accuracy
Training	0.985
Test	0.973

Table 4.5: Reconstruction accuracy of the AE.

function except for the Linear layer which has a linear activation function. This enables to exploit all the possible values of the latent space. This means that the AE will encode two subsequent frames into a single vector with 64 values. Given z the latent vector, it will be used by the Controller to learn a policy to control the CartPole. Note that it is essential to encode at least two frames in the same latent vector, as the embedded representation should resemble a description of the current dynamic which is happening in the environment.

Training the AE can be done similarly to the regression task: couples of frames can be collected, while the Controller acts using a random policy. Note that, as in the regression task, the action is not given in input to the AE as in the realistic scenario the Sensor can not access to this information. A dataset containing 10000 couples of frames is collected, 8000 are used for training the AE and 2000 are used for testing the reconstruction accuracy. A batch size of 128 samples is used and the AE is trained for 3000 epochs. At every epoch the batch is randomly sampled to reduce the correlation between the data which might be correlated due to the procedure used to collect them. The optimizer used is the Adam optimizer with a learning rate of 10^{-4}.

In Tab. 4.5 the reconstruction accuracy obtained with the AE is reported. This

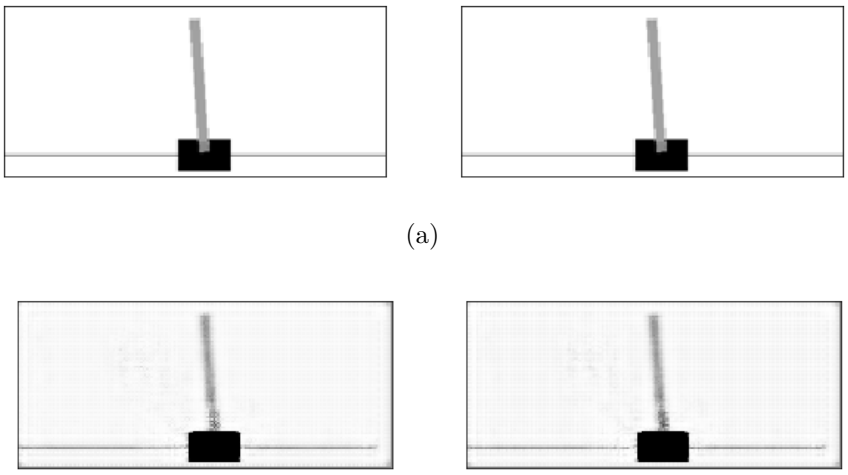

Figure 4.2: Example of two frames (a) original (b) reconstructed by the AE.

measure is obtained as 1 - MSE(x,$\hat{x}$). More specifically, the input is $x \in \mathbb{R}^{2\times180\times360}$, which is a matrix containing two frames corresponding to o_t and o_{t-1}. $\hat{x}$ has the same size of x and is the output of the AE. In Fig. 4.2 an example of two frames and the corresponding reconstruction obtained with the AE is pictured. It is possible to see that the reconstruction is accurate, so that it is possible to say that very little information is lost when representing the pair (o_t, o_{t-1}) with z_t.

Using the latter result, it is possible to train the Controller giving in input to its policy only the embedded representation of the frames. On one side, this will reduce the size of the neural network to use at the Controller, but it also avoids to pass all the frames to the Controller. Hypothetically, the Sensor will sense the frames and then send to the Controller the encoded version obtained with the Encoder of the previously trained AE.

It is possible to train the Controller using Deep Q-learning as mentioned before. The architecture used is the same of the supervised scenario, but the size of the input layer is 64 as the size of z. The training setting is similar to the supervised case: softmax policy with the same exploration profile. However, it has been noted that the training procedure requires more time to converge (4000 episodes).

4.5 A finite capacity communication channel

The system described before, whether it is the supervised or unsupervised version, provides a very good benchmark on the possible performance of this model. However, an application to a real scenario will need to satisfy other requirements. For example, if the Sensor has to communicate the observations to the Controller, it must use a communication channel. Typically, this communication channel has a finite capacity. In particular, in this work, a digital channel with a maximum number of bits per message is considered. Defined B the maximum number of bits that it is possible to communicate through the channel at every time step, the Sensor has to encode the observations choosing among 2^B messages. From this perspective, one might think to use the estimation of the true states (the 4 values representing a physical description of the state) using a predefined number of bits per value. This approach has two main drawbacks:

- The system dynamic might be unknown to the Sensor, meaning that there is not any possibility of learning to estimate it in a supervised way.
- The system performance degrades poorly as the number of bits for representing the values decreases.

The reason of the latter drawback is that there is no way to allocate the communication resources (bits) in an optimal way. If the Control performance is bad, the only way possible is to increase the number of bits used for representing the values.

The other unsupervised approach solves the issue of having to know the physical value of the system to train the estimator, but it does not improve from the side of optimal allocation of the limited resources. One thing which can be done is to implement a Vector Quantization (VQ) after the Encoder. In Sec 2.7, it was shown how VQ provides a theoretical guarantee to return a (locally) optimal solution given a maximum number of bits to represent a dataset of vectors. However, it was previously highlighted how a joint training of the AE and quantization can lead to bad performance, since the decoder is not trained on the quantized vectors. One way to understand this problem is to consider two latent vectors which yield different reconstructions at the decoder, but quite similar embedded representations. The VQ will encode these latent vectors with the same codeword, making it impossible for the decoder to return different reconstructions.

In [4], Authors proposed an architecture to train jointly the AE and the VQ by optimizing a single loss function. This model is named Vector Quantized Variational Autoencoder (VQ-VAE). It is possible to think a VQ-VAE as a VAE with discrete latent variables. Consider a VAE with a discete embedding space $y \in \mathbb{R}^{K \times D}$ where K is the size of the discrete

latent space. Recalling the VQ, K is the number of $\mathbb{R}^D$ regions with a relative codeword associated. z_e is the output of the Encoder of the AE, it is the vector representing the input in a continuous latent vector space. This discrete distribution is defined as a one-hot encoding:

$$q(z = k \mid x) = \begin{cases} 1 & \text{for } k = \text{argmin}_j \|z_e(x) - y_j\|_2 \\ 0 & \text{otherwise} \end{cases} . \tag{4.3}$$

This is exactly the nearest neighbor assignment of the VQ. Note that in this case, a deterministic distribution is used. Thus, defining a uniform prior distribution over the latent space, the KL divergence term of the VAE is constant and equal to $\log K$. Given a uniform prior $p(z = k)$ on the codewords, the probability of each codeword is $\frac{1}{K}$ leading to

$$\begin{aligned} D_{KL}(q(z = k \mid x), p(z = k)) = \mathbb{E}\Big[\log \frac{q(z = k \mid x)}{p(z)}\Big] = \\ \underbrace{\mathbb{E}[\log q(z = k \mid x)]}_{=0} - \sum_{k=1}^{K} \frac{1}{K} \log \frac{1}{K} = \log K. \end{aligned} \tag{4.4}$$

The term $\mathbb{E}[\log q(z = k \mid x)]$ is equal to zero since the joint probability $q(x, z)$ is zero for every k except one, but the $\log q(z = k \mid x)$ is equal to zero for $q(z = k \mid x) = 1$. It is possible to ignore the KL term in the loss function of the VQ-VAE. This implies that every deterministic discrete distribution over the codewords will be accepted.

The quantized latent vector can be written as

$$z_q(x) = y_k, \quad \text{where} \quad k = \text{argmin}_j \|z_e(x) - y_j\|_2. \tag{4.5}$$

The latter quantization function is not differentiable, since no gradient can be defined for the argmin_j function. In the VQ-VAE paper [4], Authors proposed to use straight-through gradient estimation to solve this issue. There exist another method to do differentiate through a non differentiable function, which is the Grumbel-Softmax reparametrization [47]. However, in this work the straight-through method is considered, since it is known to converge much faster.

In Fig. 4.3, the forward and backward pass of the quantization layer is reported. In the forward pass, z_e is quantized using one of the codebook vectors y_k. In particular, in the figure it is highlighted as the quantized vector z_q which is given in input to the Decoder $DEC(z_q)$, and is written as $z_q = z_e + q$ (the embedded input z_e plus a term q, which refers to the quantization noise). In particular, $q = y_k - z_e$, thus $z_q = z_e + (y_k - z_e) = y_k$.

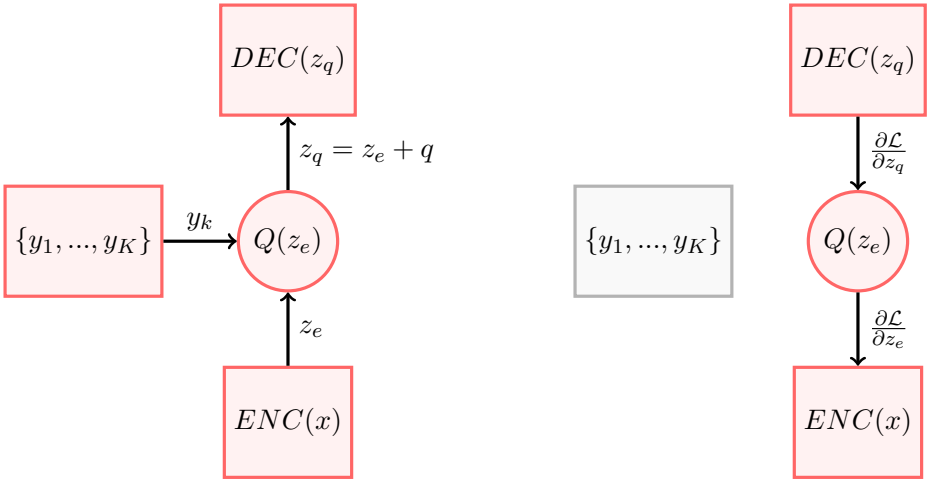

Figure 4.3: Forward and Backward pass using the straight through estimation of the gradient.

However, in the procedure of the forward pass in the neural network, z_q is written as

$$z_q = z_e + \text{sg}(y_k - z_e), \tag{4.6}$$

where $\text{sg}(\cdot)$ is the *stop gradient* operator. This operator excludes the input term from the gradient computation. This trick is similar to the reparametrization trick of the VAE and permits to exclude the non-differentiable computation from the gradient of the neural network. In the second picture of Fig. 4.3, the backward computation is shown: it is possible to see that no gradient is computed with respect to y_k.

This architecture enables to use the standard loss of the AE and to train the Encoder and Decoder parameters. However, there is still one part missing out: the codebook vectors are not updated, since they are excluded from the gradient computation. Following the idea of the LBG algorithm, the codeword $y_1, ..., y_K$ should be updated towards the points which fall in their region. This can be done by simply reducing the distance between the codeword y_k and the point z_e which is inside the region I_k. In particular the distance which has to be reduced is

$$\|y_k - \text{sg}(z_e)\|_2^2. \tag{4.7}$$

Note how z_e is the target point, so that y_k is updated towards z_e but z_e is treated as a constant value (using $\text{sg}(\cdot)$). It is possible to jointly optimize the Encoder and Decoder parameters and the codewords by adding Eq. (4.7) in the loss function of the AE. The

new loss function becomes

$$\mathcal{L} = \|x - \hat{x}\|_2^2 + \|y_k - \text{sg}(z_e)\|_2^2. \tag{4.8}$$

Authors in [4] introduced a third term which they called **commitment term** to guarantee that z_e is forced to stay near the codeword e_k. This term should stabilize the learning procedure avoiding that the latent space z_e changes too rapidly with respect to the codebook. The final loss function is made by three terms and can be written as

$$\mathcal{L} = \underbrace{\|x - \hat{x}\|_2^2 \quad + \quad \beta\|z_e - \text{sg}(y_k)\|_2^2}_{\text{updates the parameters of the Encoder and Decoder}} \quad + \quad \underbrace{\|y_k - \text{sg}(z_e)\|_2^2}_{\text{updates the codewords}} . \tag{4.9}$$

The term $\beta\|z_e - \text{sg}(y_k)\|_2^2$ is the **commitment term** and β is the **commitment cost** which controls the importance of this term with respect to $\|x-\hat{x}\|_2^2$. The Authors proposed to use $\beta = 0.25$.

The main improvement of this architecture is the possibility to obtain a discrete latent representation of the input x. However, when dealing with images, it can be very challenging to encode one image with one single vector. This is because the input dataset can have a lot of diverse inputs which require many different codewords y_i to allow a small reconstruction error. To deal with this problem, authors proposed to encode the features in output of the convolution with one codeword each. Recalling AE architecture in Tab. 4.4, it is possible to see that before the latent space there is a flatten layer which takes the features of the convolution and creates a 1-dimensional vector containing all the values. However, in the VQ-VAE, the output of the last convolutional layer is quantized, leading to a latent space where each input is represented as a set of quantized features.

In Fig. 4.4, a visual representation of the architecture of the VQ-VAE is shown. It is possible to see that the output of the convolution is a three dimensional tensor. In details, if the last convolutional layer reduces the input image to a height H and a width W and uses C filters, the results output for each image will be a $H \times W \times C$ matrix. The quantization layer will quantize each one of the $H \times W$ vectors of length C using the codewords y_i. At the decoder, a deconvolutional layer will reconstruct the input starting from the quantized version of the image. Note that in this way the latent representation of an image $q(z \mid x)$ is a matrix containing the indexes of the codeword used for every sample of height $h = 1, ..., H$ and width $w = 1, ..., W$. Using this tensor representation might not seem necessary. However, this reduces the size of the dictionary (codebook) that has to be used a lot. For example, given a codebook of size K, with a 1-dimensional discrete latent space, it is possible to represent each input x in at most K ways. However, with more

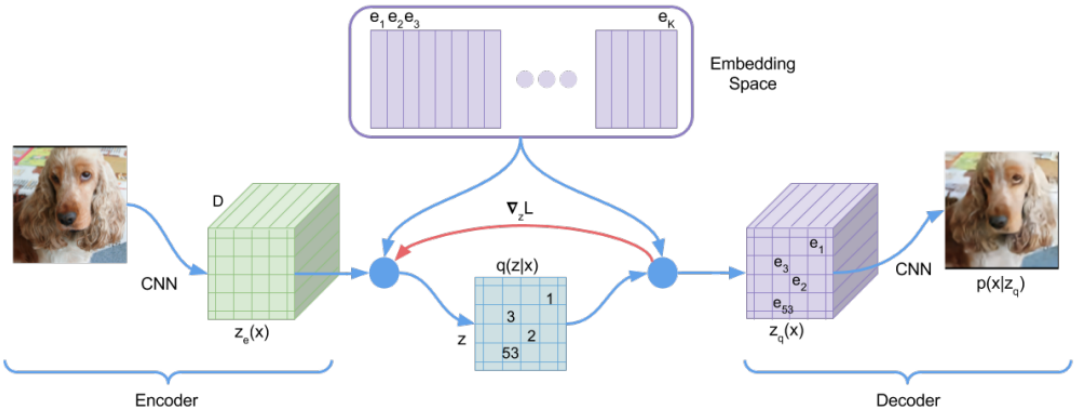

Figure 4.4: Visual representation of the VQ-VAE architecture, [4].

features it is possible to exploit the combinations of more codewords. This means that the size of the discrete distribution in architecture in Fig. 4.4 is $K^{H \times W}$. This drastically reduces K, but it still has many possible ways to encode the input. Dealing with small codebooks can improve the efficiency of the system, since it reduces the communication overhead necessary to communicate the new codewords to the Controller and also reduces the storage resources necessary to store the dictionary of codewords.

Given this premise, the VQ-VAE is a suitable architecture which can deal with a lot of problems related to the considered system model. Given that both the Sensor and the Receiver know the optimal codewords discovered by the VQ-VAE, it is possible to communicate the input x, from the Sensor to the Controller, just by sending the sequence of indexes which corresponds to the selected codeword for every feature. A rough estimation of the number of bits necessary to do this is $H \times W \log_2(K)$. It might be possible to further reduce the average number of bits by collecting a statistics on the number of time each codeword is communicated and implement various entropy encoding techniques. However, this latter approach is not investigated in this work.

4.5.1 Optimizing the Codebook

Further details about the way codewords are updated need to be discussed. At the beginning of the training phase, the codewords are randomly initialized. Then, for each batch of inputs of size B, a set of feature vectors $\{z_e^1, ..., z_e^{n_i}\}$ will be encoded with the

codeword y_i. This means that the best codeword which optimizes the loss function $\mathcal{L}$ is

$$y_i = \min_{y_i \in \mathbb{R}} \sum_{j=1}^{n_i} \|\text{sg}(z_e^j) - y_i\|_2^2, \tag{4.10}$$

which is a least square problem that allows a closed form solution. The new codeword y_i will be

$$y_i = \frac{1}{n_i} \sum_{j=1}^{n_i} z_e^j. \tag{4.11}$$

This is indeed the same updating rule of the LBG (and the K-means) algorithm. However, this is not practical to be used in the training of a Neural Network. Typically, a NN requires to use batches of inputs and not the entire dataset in the same iteration. This update rule for the codeword might not work well in the batch version. Thus, Authors in [4] and in [48] proposed to use an exponential moving average (EMA) updating rule for the codewords. This allows the codewords to change slowly with respect to the latent space, leading to a more stable learning dynamics. This can be implemented by using a cluster size vector which stores all the number of times a codeword is used in the same batch at iteration t. For the codeword y_i, $N_i^{(t)}$ stores this value. Another vector keeps track of the sum of the z_e^j which are encoded with y_i. At the beginning of the training phase, $N_i^{(0)} = 0 \forall i \in \{1, ..., K\}$ and $m_i^{(0)} = 0 \forall i \in \{1, ..., K\}$. At every iteration t, these values are updated as

$$N_i^{(t)} \leftarrow \gamma N_i^{(t-1)} + (1 - \gamma) n_i^{(t)}, \tag{4.12}$$

$$m_i^{(t)} \leftarrow \gamma m_i^{(t-1)} + (1 - \gamma) \sum_{j=1}^{n_i^{(t)}} z_e^j. \tag{4.13}$$

After obtaining these values, the new codeword at ever iteration t is given by

$$y_i^{(t)} = \frac{m_i^{(t)}}{N_i^{(t)}}. \tag{4.14}$$

Using $\gamma = 0.99$, as proposed by the Authors, maintains the update of the codewords stable, avoiding them to collapse around a small region.

This way, the codewords are slowly converging to the optimal values, avoiding to change too rapidly.

Another issue often faced when learning the codebook is that it may happen that not all

the codewords are used to encode the latent features. This issue is not investigated much in [4], since the authors are not concerned to use a limited number of K. However, in the problem considered in this work, using all the possible messages (all the codewords) is highly relevant. Having codewords which are never used to encode the input means that it might be possible to reduce the size K and thus the message length $H \times W \times \log_2(K)$. This is also a known problem in K-means, where the problem is typically solved using multiple execution of the algorithm and through multiple reinitializations of the codewords. This is impractical to be done in this context. Another way to make sure to use all the codewords is to use the split-LBG [37]. This method consists on starting the LBG algorithm with one codeword and updating it until convergence. Once converged to the optimal codeword, the codeword is splitted in two by slowly perturbing the optimal codeword with a small random ϵ. This procedure is repeated for a lot of iterations until the desired number of codewords is obtained. This ensures that all the codwords are used but it is still impractical to be used with batches of inputs.

It is possible to see that the main issue which causes a codeword not to be used is that it might be initialized far from the encoding region of the Encoder. Based on some observations made in [48] and inspired by the split-LBG version of the LBG algorithm, a novel approach to deal with the problem of non-used codewords is proposed. This method can be implemented in a batched training procedure and allows to mitigate this issue. Consider a batch of inputs $\mathcal{B} = \{x_1, ..., x_B\}$. At the output of the Encoder, there will be $B \times H \times W$ feature vectors to be encoded. Then, every feature vector will be encoded using the nearest neighbor condition. Let's call E the tensor of size (B, H, W, C) where all the entries are zeros except for the entry $E_{j,h,w,i} = 1$ when the feature z_e of index h, w of input x_j is encoded with codeword y_i. Summing the entries of E with respect to the batch, height and width dimensions returns the vector of length K, containing the number of feature vector encoded with each codeword. If this number is zero, it means that the codeword has never been selected within that batch. If the batch is large enough, all the codewords should be used within the same batch. If some codewords are not selected, they are re-initialized near a latent point z_e, selected randomly from the matrix. Certainly, this reset of the codeword cannot be done every iteration, but it should be done after a fixed number of iterations.

In Alg. 4.2, the training procedure of the VQ-VAE is presented. In the first part of the algorithm, the forward pass is computed obtaining the reconstructed input. Then, the parameters of the Encoder and Decoder are updated using the loss function $\mathcal{L}$. The codewords are updated using the previously described EMA procedure. The last part of the algorithm shows the codeword reset introduced by this work. It is possible to

Algorithm 4.2 Training Algorithm of the VQ-VAE with codewords reset

Initialize the Encoder ENC_θ
Initialize the Decoder DEC_ϕ
Initialize the K codewords randomly
$n \leftarrow 0$
for each iteration:
  Sample a batch $\mathcal{B}$ randomly from the dataset D
  $z_e \leftarrow ENC_\theta(x) \ \ \forall x \in \mathcal{B}$ (Pass the input through the Encoder)
  Select $y_k \ \ \forall z_e$ (Quantize the latent vectors)
  $\hat{x} \leftarrow DEC_\phi(y_k) \forall y_k \in E$ (Reconstruct the through the Decoder)
  Update θ and ϕ according to $\mathcal{L}(x, \hat{x})$
  Update the codewords y_i for $i = 1, ..., K$ using the EMA equations
  if $n \ \% \ 100 = 0$:
    Find $\{y_j\}$ for $j \in \mathcal{U}$ (set of unused codewords)
    Sample $|\mathcal{U}|$ latent vectors from the current latent vectors $\{z_e^j\}$
    $y_j \leftarrow z_e^j + \epsilon \ \ \forall j \in \mathcal{U}$
  end if
  $n \leftarrow n + 1$
end for

see that the subroutine for resetting the unused codewords is done every 100 iterations; this is because, after the reset, the VQ-VAE should adapt to the new codewords before searching for the new unused ones. In the algorithm, $\mathcal{U}$ is the set containing the indexes of the unused codewords, while ϵ is a small random vector which is added to the latent vector to reset the codeword near that point. This is a simple method, but it has been found to work effectively to enable the usage of all the codebook. Intuitively, resetting the codewords near the latent vectors should make them more likely to be selected as the encoding codeword in the following iterations. Also note that regions of the latent space where there are more points (more latent vectors) are more likely to be selected to be the region where the codeoword is reset. This is something useful, as the high density regions of the latent space should be quantized with higher precision.

To track the average number of codewords used in every iteration, the **Perplexity** on the codewords is tracked during the training. The perplexity $PP(p(x))$ of a discrete distribution $p(x)$ is defined as

$$PP(p(x)) = 2^{H(p)} = 2^{-\sum_x p(x) \log_2 p(x)} = \prod_x p(x)^{-p(x)}. \tag{4.15}$$

$H(p)$ is the entropy of the discrete distribution $p(x)$ and represents the average number of bits required to encode the outcome of the random variable X. Thus, $PP(p(x))$ is

the expected number of different outcomes when observing the output of the random variable X. A low perplexity over the distribution $p(x)$ implies that the random variable X is highly predictable, whereas a high perplexity means the random variable is not predictable.

In the context of the codewords usage, it is possible to compute the perplexity among a discrete distribution of the codewords computed during an iteration (within the batch of samples). In particular, $N_{\text{latent}} = B \times H \times W$ the total number of codewords used in the batch, the probability of the codeword y_i is computed as

$$p(y_i) = \sum_{j=1}^{N_{\text{latent}}} \frac{\delta_i(z_e^j)}{N_{\text{latent}}} \qquad \text{for } i = 1, ..., K \tag{4.16}$$

where

$$\delta_i(z_e^j) = \begin{cases} 1 & \text{if } z_e^j \text{ is encoded with } y_i \\ 0 & \text{otherwise} \end{cases} . \tag{4.17}$$

Perplexity is telling something more than how many codewords are used inside the batch; it tells how often the codewords are chosen. For example, if the perplexity is near K, it means that every codeword is chosen approximately $\frac{1}{K}$ times, meaning that all the codewords are used on average the same number of times. If the perplexity is low, there is a high chance that not all the codewords are used or that some codewords are used a lot and some others very few times. Ideally, the perplexity should be near to K, meaning that the Encoder is equally exploiting every codeword to represent the inputs. However, the perplexity can be lower than K even if the quantization step is using all the codewords. This is due to the fact that some codewords might be used a lot of times whereas others less times.

4.5.2 Results on the VQ-VAE

The VQ-VAE is trained on a dataset containing 60000 pairs of frames representing two subsequent obervations of the environment. 50000 are used in the training, whereas 10000 are used to test the final performances.

The layers of the VQ-VAE implemented are reported in Tab 4.6. It is possible to note that a residual layer is added at the end of the first three convolutional layers as suggested by [4]. The pre-quantization layer is a convolutional layer which ensures that the number of channels at the output of the Encoder is equal to the desired embedding dimension which must match the size of the codeword vectors. Another aspect to consider carefully is the kernel sizes. Indeed, the choice of the convolutional kernels related to the input

Layer	Input	Output	Kernel Size	Stride
ENCODER				
1st Conv. Layer	2 channels	64 channels	(11,10)	(8,9)
2nd Conv. Layer	64 channels	64 channels	(12,12)	(10,10)
3rd Conv. Layer	64 channels	128 channels	(3,3)	(1,1)
Residual layer	128 channels	128 channels		
Pre-Quant. layer	128 channels	64 channels	(1,1)	(1,1)
Quantization layer	64 channels	64 channels		
DECODER				
1st Deconv. Layer	64 channels	128 channels	(3,3)	(1,1)
2nd Deconv. Layer	128 channels	64 channels	(12,12)	(10,10)
3rd Deconv. Layer	64 channels	2 channels	(11,10)	(8,9)

Table 4.6: Layers of the VQ-VAE implemented.

Parameter	Value
Learning Rate	10^{-3}
Decay (EMA)	0.99
Commitment Cost β	0.25
Random Noise ϵ	$\mathcal{N}(0, 10^{-4})$
Number of Training Iterations	7500
embedding dimension	64
number of codewords K	64

Table 4.7: List of parameters used for the practical implementation of the training of the VQ-VAE.

images height and width determines the number of latent features which represent the input. If the input frames have 160×360 pixels, the latent space results to have $2 \times 4 = 8$ features.

In Tab. 4.7, the parameters used to train the VQ-VAE architecture are reported. During the training procedure, the Peak Signal to Nosie Ratio (PSNR) and the perplexity of the codebook is tracked. The PSNR is measured in a decibel scale (dB), the actual form used in the project is

$$\text{PSNR} = 10 \log_{10} \frac{\text{MAX}^2}{\text{MSE}}, \tag{4.18}$$

where MAX is the maximum value that the pixel of the image can assume (in this case MAX = 255, since the images are represented with 8 bit per pixel). The MSE is the mean squared error between the input image and the reconstructed one. A higher value of the PSNR implies a higher accuracy and a lower MSE between the original and reconstructed

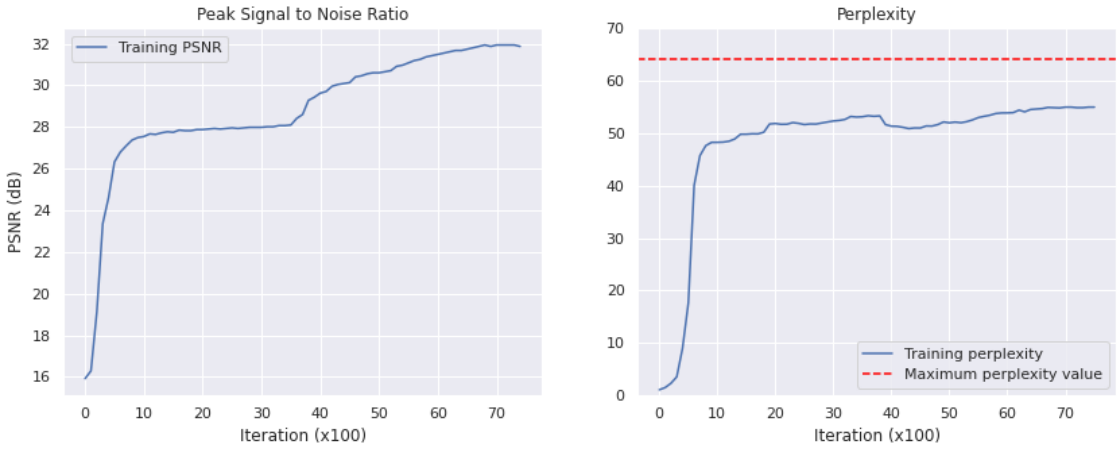

Figure 4.5: Training profile of the VQ-VAE for the $K = 64$.

images.

In Fig. 4.5, the training curves of the VQ-VAE are shown. In the first plot, the training PSNR is represented. In the second picture, the average perplexity within the batch is reported. It is possible to see that, during the training phase, the PSNR might remain stable while the perplexity is still changing. Indeed, it is possible to say that the training has converged when the PSNR and the perplexity do not change anymore. In the second plot, in red, the maximum perplexity value for a codebook of $K = 64$ codewords is shown.

The final PSNR is 31.871 dB and the final perplexity is 54.965. Despite the final perplexity does not reach 64, it is still acceptable as a result.

4.5.3 Training the Controller on the quantized latent representations

In this section, some tests are accomplished: the main goal is to verify if the given quantized representation is able to inform the Controller about the current state of the system and to enable the RL algorithm to learn a control policy. A strategy similar to the supervised and unsupervised case is used. The main difference is that now the Sensor is able to transmit the encoded observations to the Controller, using a limited capacity digital channel. More specifically, the number of bits required for encoding 8 features with 64 different possible values is $8 \times \log_2(64) = 48$ bits. The learning pipeline of the Controller is equal to the one presented before. The Sensor encodes two subsequent frames into 8 quantized features (the latent representation) and transmits the corresponding symbols to the Controller. Then, the Sensor (which knows the codebook) receives the features and

uses this input as the available state to learn a control policy. Since the number of feature are 8 and the embedding dimension is 64, the state at the Controller is represented by a $8 \times 64 = 512$ dimensional vector obtained stacking the feature vectors.

The training is done for 8000 episodes, using a ϵ-greedy policy with exponentially decaying ϵ. The softmax exploration policy has been found to lead to unstable results, so the ϵ-greedy policy has been preferred. The same Deep Q-learning algorithm, exploiting a policy network and a target network has been used. It was possible to reach an overall performance similar to the unsupervised scenario. The average episode length for the quantized latent space is 277.24.

5

Optimizing the Communication Rate

Until now, it has been shown how it is possible to solve a control problem by encoding and quantizing the high dimensional observations. The approach has focused on a system model which is similar to a simplified joint source channel coding architecture. The simplification comes from the fact that a maximum rate R [bits/sample] is decided and then a compression architecture, which uses at most R bits, is used to encode the observations at each time step t. The analysis in this chapter will concentrate on investigating the possibility of reaching a lower rate. In particular, multiple quantizers with different decreasing number of codewords will be exploited. This will allow to explore hybrid strategies which take advantage of the different quantizations to dynamically encode the input observations.

5.1 Codebook Size Analysis

To explore the effect of reducing the number of codewords for the vector quantization, different tests are accomplished. In particular, the previously trained VQ-VAE (with 64 codewords), is used. The same Encoder and Decoder are used, but a different codebook replaces the previous one. The way to discover the codebook is the same training procedure of the AE. However, in this case, it is possible to use the Encoder and Decoder already trained and just learn a new set of codewords with a different size of codebook.

To practically obtain a procedure which learns a new quantization, it is possible to use the same algorithm and loss function but excluding the parameters of the Encoder and the Decoder from the optimization step. Recalling the loss function for training the

K	**PSNR [dB]**	**Perplexity**
2	22.44	1.839
4	24.495	3.752
8	27.894	7.214
16	30.621	14.192
32	31.438	29.875
64	31.871	54.965

Table 5.1: PSNR and Perplexity for different sizes of the codebook.

VQ-VAE, the new loss function will contain only one term:

$$\mathcal{L} = \|y_k - \text{sg}(z_e)\|. \tag{5.1}$$

All the terms which update the Encoder and Decoder parameters can be cancelled from the loss function. In this case, it is not essential to use an EMA update of the codewords. Indeed, the latent space is kept fixed and does not change, so there are not stability issues during the training.

This approach allows to obtain different quantizations for the same Encoder. In this way, it is possible to test how the size of the codebook affects the performance. The different sizes used are 2,4,8,16,32 and 64. Note that these are all powers of 2. Indeed, the maximum number of bits to represent K codewords distinctly is $\log K$. This implies that the corresponding number of bits to represent the used codebooks are 1,2,3,4,5 and 6.

In Tab. 5.1, the PSNR and Perplexity for different sizes of the codebook are shown.

5.2 Testing the performance for three levels of communication

In this section, the three levels of communication considered in this project are analyzed. In Sec. 2.10, the three levels of communication formulated in [10] were presented. In this project, it is possible to test the developed encoding architecture on the three levels by performing different tests and analysis.

- **Level A. The technical problem:** in this scenario, the receiver is interested in reconstructing accurately the input observations at each time step t. The Agent exploits the trained Decoder to reconstruct the observations. This problem is exactly the first level of communication: the PSNR between the original and the recon-

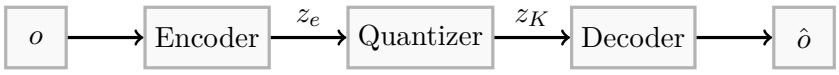

Figure 5.1: Pipeline for testing the performance in the technical problem for different size of codebooks K.

structed observations is a measure of how well the symbols of the communication allows to reconstruct the original message.

- **Level B. The semantic problem:** in this scenario, the receiver learns to reconstruct the physical state form the received features. This can be considered a semantic problem since the receiver is interested in extracting some information from the received message. Indeed, this information is not directly expressed in the observations which are encoded, but it has to be retrieved from them. So, this analysis will understand whether the compressed observations still contain the semantic information these inputs bring.
- **Level C. The effectiveness problem:** in this scenario, the receiver controls the CartPole from the received features. What is tested here is whether it is possible to learn a policy from the compressed observations. Indeed, since the compressed state is the only information available to the receiver, this analysis will test the conduct of the Agent based on the received message.

To analyze the three levels, the receiver should be able to perform three different computations on the received states. For the technical problem, it uses the Decoder to reconstruct the observations. For the semantic problem, it uses a regression network to estimate the physical state of the system. While for the effectiveness problem, it uses a policy network to choose an action and observe the reward.

Level A is implicitly tested when the different codebooks for the quantization are learned. In fact, in this simplified model, there are no loss due to the channel, and so the reconstruction performance is the same reached at the Sensor side while training the VQ-VAE.

In Fig. 5.1, the system model to test the different quantization for the technical problem is reported. Note that all the blocks in this case are colored in "gray" since there is no more training in these blocks. Once the quantization codebooks has been learned, the Receiver node uses the Decoder to reconstruct the input. The overall performance can be evaluated measuring the PSNR between the original input o and the reconstructed $\hat{o}$.

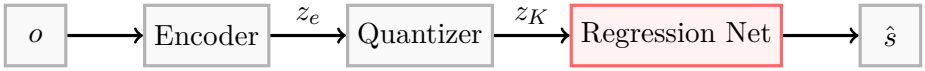

Figure 5.2: Pipeline for testing the performance of the semantic problem for different size of codebooks K.

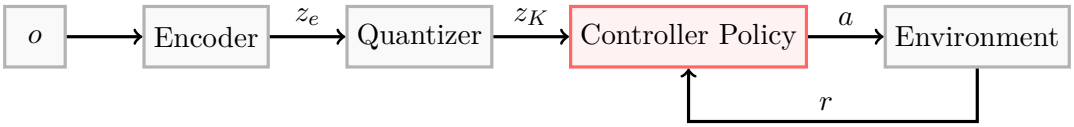

Figure 5.3: Pipeline for testing the performance of the effectiveness problem for different size of codebooks K_i.

To obtain a measure of the effectiveness of the different quantizers for the semantic problem, a different pipeline has to be adopted. As it is possible to see in Fig. 5.2, the Decoder block has been replaced with a regression network, which is trained to reconstruct the true physical state s by estimating it from the received quantized features z_K. The regression network block is colored in red because it has to be trained. The training procedure is identical to the previously implemented ones. The true state s is assumed to be known during the training and the MSE between the s and $\hat{s}$ is used as the objective function to be minimized during the training.

After training the regression network, it is possible to obtain an estimation of the semantic problem performance, meaning it is possible to verify the amount of information about the true state process s which is still available in the encoded and quantized version of the observations.

In Fig. 5.3, the pipeline for the level C communication problem is reported. In this case, the Controller receives the quantized features and has to learn a policy to control the CartPole. The performance that the Controller is able to reach in this context, is an estimation of the Level C communication paradigm. The test will show how the quantization codebook adopted by the Sensor allows the Controller to learn a good policy and to adapt its behavior. In this case, the training procedure considers the Controller directly interacting with the environment. In level A and level B, there is no need to consider the Controller policy. Indeed, in these two cases, the precision in reconstructing the original input or a hidden information which can be retrieved by the input is tested. On the other hand, in Level C, the behavior of the Controller is what matters and so the interaction with the Environment is considered. To learn a policy, the Controller receives

Layer	Input	Output
1st Linear Layer	512	512
2nd Linear Layer	512	256
3rd Linear Layer	256	4

Table 5.2: Regression network architecture.

Layer	Input	Output
1st Linear Layer	512	512
2nd Linear Layer	512	256
3rd Linear Layer	256	2

Table 5.3: Policy network architecture.

a reward signal r, as described in the previous sections.

Note that for every K, a dedicated regression network and policy network are learned. This implies that for every $K \in \{2, 4, 8, 16, 32, 64\}$ a regression network $f^{(K)}(z_K)$ and policy $\pi^{(K)}(z_K)$ are defined. The architectures for the regression network and the policy network are similar. In particular, the same input shape can be used in both cases. The Quantizer codebooks encode the same latent space of 64 dimensional vectors with different number of codewords. Recall that the designed Encoder extracts 8 features at the output of the last convolutional layer. This implies that for every different quantization, the same final latent dimension is represented. More specifically, the feature vectors are stacked in a single vector of size 512 (8×64).

In Tab. 5.2, the architecture of the regression network used in the semantic problem tests is shown. The first two layers has a ReLU activation function, whereas the last layer has linear activation function. Note that the output layer has 4 nodes which are the 4 physical state values.

A similar architecture is used for the policy network (Tab. 5.3). The only difference with respect to the regression network is that the number of output nodes is 2, as the available actions to the Controller. This network estimates the Q-values corresponding to each action given the input z_K. These Q-values are then used to obtain an ϵ-greedy policy for the Controller.

In Level B communication problem, the regression network is tested with a dataset containing 50000 samples, iterating for 3000 epochs in a training loop with batch size equal to 256 samples. The optimizer used is Adam with a learning rate of 10^{-4}.

The effectiveness problem requires a RL procedure to learn a dedicated policy for each quantization codebook. The already presented Deep Q-learning is used and the same

Number of codewords	Bits per feature	Message length (in bits)
2	1	8
4	2	16
8	3	24
16	4	32
32	5	40
64	6	48

Table 5.4: Number of bits per feature and length of the message for different sizes codebook.

setting of Sec. 4.5.3 is used. The number of episodes varies since the convergence time can vary depending of the codebook used. A more precise description of this phenomenon will be given in the following sections.

5.3 Learning-Based Adaptive Encoding

In the previous scenarios, the Sensor is just a passive encoding system. It just uses the fixed number of codewords inside a Quantizer and encodes the features to be transmitted through the digital channel. However, by having different quantizers, it might be possible to design different quantizing strategies at the Sensor side, in order to reduce the rate but still guarantee an average target performance. Indeed, when a fixed codebook is used to encode the features, the number of bits required to encode the current observation is $8 \times \log_2 K$.

In Tab. 5.4, the number of bits per feature and the length of the encoded message are shown. For different sizes of the codebook, it is possible to see that the message length increases of 8 bits every time the codebook size doubles. This means that, when the codebook to use is decided, the message length is always the same and the average rate is trivially equal to the message length.

In this section, a dynamic selection of the codebook to use will be implemented. This can be done by designing a method which allows the Sensor to decide which codebook to use among the six available for every observation. The aim of this procedure is to learn a policy at the Sensor side and exploit all the available Quantizers. This way, it is possible to obtain different message lengths based on the selected Quantizer and possibly obtain an average rate which is lower than the fixed Quantizer case. The main intuition behind this method is that the current observation can be easy or difficult to understand by the Controller, depending on the current observation. Thus, the Sensor might learn to compress more or less, depending on the ability of the Controller to perform the target

tasks. The working principle is that the current observations are distorted in different ways by the compression scheme adopted. Hence, depending on the current time and observations, the amount of information (relevant to the Controller task) which is lost when compressing varies.

To be able to choose dynamically the Quantizer to use, the Sensor has to learn a policy. This policy receives in input the current observation frames o_{t-1} and o_t, and outputs an action which corresponds to the codebook to use to quantize the features. It is possible to treat the process of choosing the correct quantization codebook as an MDP. The reward that the Sensor receives for choosing a certain codebook can vary depending on the task the receiving node is performing. Given the three levels of communication presented before, it is possible to define three different rewards for the Sensor MDP. In general, the reward has to punish the Sensor for using a quantizer with a high number of codewords. Indeed, this choice yields a higher number of bits for encoding the message. However, the Sensor should be rewarded positively if the performance at the receiver side is good. It is possible to define three rewards:

- **Reward for the technical problem (Level A)**

$$r_t = \mathrm{PSNR}(o_t, \hat{o}_t) - \beta \log_2 K_t.$$

- **Reward for the semantic problem (Level B)**

$$r_t = -\mathrm{MSE}(s_t, \hat{s}_t) - \beta \log_2 K_t.$$

- **Reward for the effectiveness problem (Level C)**

$$r_t = Q^{(K_t)}(z_{K_t}, a_t) - \beta \log_2 K_t.$$

The term $-\beta \log_2 K_t$ indicates the cost that the Sensor pays for choosing the quantizer with K_t codewords. β sets a trade-off between how much the Sensor should be interested in the performance of the receiver with respect to the cost of the communication. Intuitively, for a large β, only the communication cost matters and the Sensor policy will always select the quantizer with the smaller number of codewords. For a small β, the communication cost will be neglected by the Sensor and only the performance at the receiver will be taken into account. Note that, for the technical problem, the PSNR between the original input and the reconstructed one is taken as a positive reward for the Sensor policy. Indeed, a higher PSNR indicates higher reconstruction accuracy. For the semantic problem, the performance of the receiver is given by $-\mathrm{MSE}(s_t, \hat{s}_t)$. This time, a high value in the mean squared error between the true state s_t and the estimated state $\hat{s}_t$ indicates worse

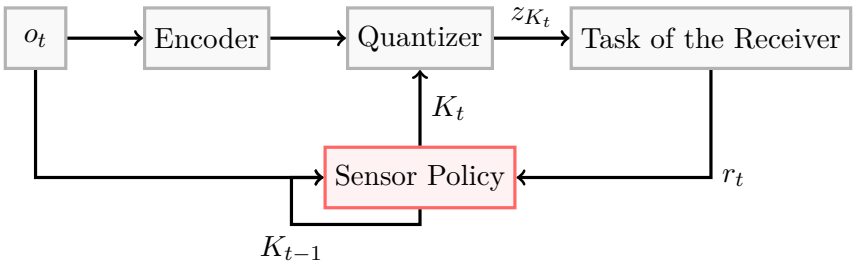

Figure 5.4: Pipeline for learning to select K_t.

performance and thus this value is inserted in the reward with a negative sign. This means that the Sensor receives a lower reward if the error is higher. The last scenario considers the effectiveness problem. In this case, the Sensor receives a reward which is given by the Q-value of the policy of the Controller. Indeed, this Q-value is an estimation of the cumulative returns that the Controller receives for taking action a_t in the state z_{K_t}. More precisely, given a discount factor γ,

$$Q^{(K_t)}(z_{K_t}, a_t) = \mathbb{E}\Big[\sum_{i=t}^{\infty} \gamma^{i-t} r_i\Big]. \tag{5.2}$$

The policy of the controller has already been trained, thus this is the best control performance estimation available at the receiver. Note that the policy network used to obtain the Q-value is the one trained on the selected quantizer.

In Fig. 5.4, the system model to train the Sensor policy to select the quantizer optimized for the technical problem is shown. From the figure, it is possible to see that the policy of the Sensor receives in input the observations at time t and also the previously selected number of codewords K_{t-1}. This latter input is not relevant in all the levels of communication. Indeed, the past action of the Sensor K_{t-1} is not useful for the optimization in the technical problem and in the semantic problem. This is because the performance at the receiver does not depend on the previously encoded frames, but just on the distortion introduced by the quantization step in the current observations. This information becomes relevant in the effectiveness problem, as the performance in the current state depends on the choice done in previous steps. Ideally, the Sensor might learn to use a higher precision (a higher number of codewords) also because the previously sent message contained a more compressed representation. However, the same network

architecture is used for all the three communication problems, meaning that the Sensor will learn to ignore this additional information in level A and B.

To learn the Sensor policy, Deep Q-learning is used. It is possible to use the same architecture for the three levels of communication. However, the Q-values learned by the Sensor policy represent different quantities. As a matter of facts, for the level A and B, the RL procedure can be represented as a multi-arm bandit scenario [1]. In fact, the future rewards do not depend on the choices performed in the current observations. This result can be formalized using the already presented notation of MDP and setting the discount factor γ equal to 0. In this way, the Sensor will automatically ignore the future rewards, learning Q-values which only represent an estimation of the immediate reward. Note from Fig. 5.4 that the only thing that is trained is the Sensor policy. The Decoder, the regression network and the Controller policy have been trained before learning to dynamically adapt the rate with respect to the input.

Since it is possible to evaluate the trade-off between the performance at the receiver node and the communication constraints only for fixed values of β, the definition of *Pareto dominance* is introduced. Let $\mathcal{C}$ be the set of all the possible configurations of the system model. Let $(\varphi, \bar{l})$ be a feasible point at which the system can work where φ is the average performance at the receiver and $\bar{l}$ is the average length of the message of the communication between the transmitter and the receiver. Then a configuration $x \in \mathcal{C}$ is *Pareto dominant* with respect to the configuration $y \in \mathcal{C}$ if and only if

$$x \succeq y \iff \varphi(x) \geq \varphi(y) \wedge \bar{l}(x) \leq \bar{l}(y). \tag{5.3}$$

This means that the configuration x improves either the performance or the average rate with respect to configuration y. Consider an algorithm (or a solution) A which yields a set of feasible configurations $x \in A$. It is possible to define the *Pareto frontier* of the solution A as

$$\mathcal{F}_A = \{x \in A : \forall y \in A \ , \ y \not\succ x\}. \tag{5.4}$$

From this definition, the *Pareto region* of the solution A is introduced as

$$\mathcal{P}_A = \{y : \exists x \in \mathcal{F}_A, x \succeq y\}, \tag{5.5}$$

which means that $\mathcal{P}_A$ is the set of all the configurations dominated by the *Pareto frontier* $\mathcal{F}_A$. The last definition to be introduced is the *Pareto dominance* with respect to two solutions, A and B:

$$A \succeq B \iff \mathcal{P}_B \subseteq \mathcal{P}_A. \tag{5.6}$$

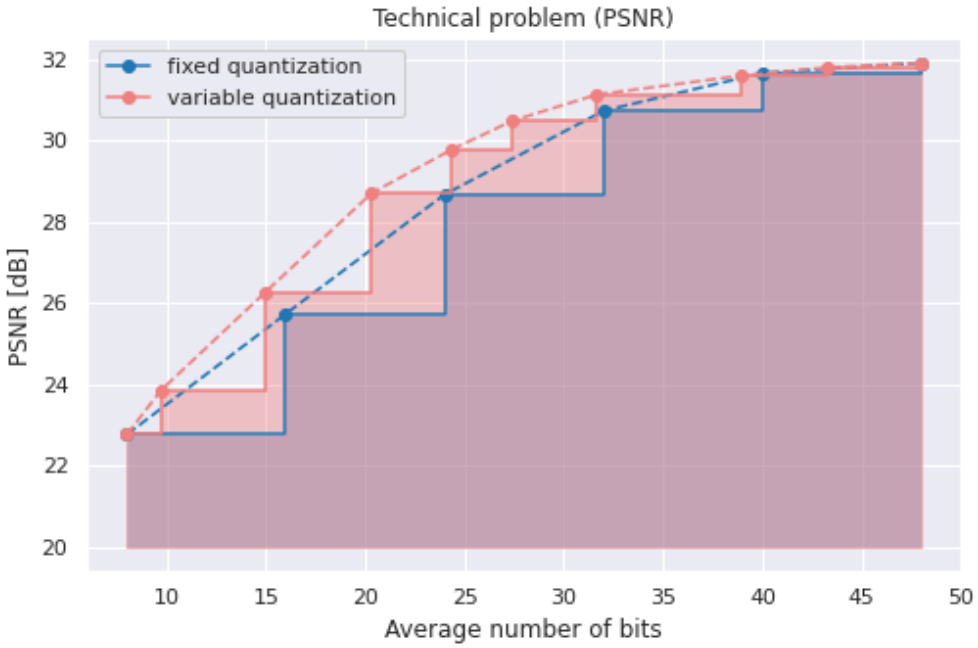

Figure 5.5: Results for the technical problem.

In words, a solution A is *Pareto dominant* with respect to solution B if and only if the *Pareto region* of B is contained in the *Pareto region* of A. This means that the solution A is able to return a set of configurations which are outside $\mathcal{F}_B$ but inside $\mathcal{F}_A$. This set of configurations is $\mathcal{P}_A \setminus \mathcal{P}_B$.

5.4 Results for the technical problem

In this section, the performance of the system model with respect to the reconstruction of the observation at the receiver side is presented. Two tests are performed with two quantization strategies. The first is a fixed quantization where only one of the available quantizers is used to represent the features and then the Decoder reconstructs the input frames. The second strategy is the dynamic strategy, where the Sensor learns the quantizer to use, depending on the input frames. For the fixed strategy, the six quantizers are used and evaluated over 100 episodes while using a random control policy at the Controller side. This control policy does not influence the reconstruction task.

In Fig. 5.5, the results for the technical problem with the two quantization strategies are shown. The fixed quantization results are represented in blue. Note that the average

β	**Average message length [bits]**	**PSNR [dB]**
0.25	43.3063	31.8023
0.5	38.955	31.5997
1	31.6596	31.1333
1.5	27.3993	30.4854
2	24.3734	29.783
2.5	20.3290	28.7065
2.75	14.9403	26.2575
3	9.743	23.8643

Table 5.5: Average message length and PSNR for different values of β.

message length for this strategy is fixed at $8 \times \log_2 K$. The general trend in this case is that the PSNR increases as a higher number of codewords is employed. This is because a larger codebook introduces a smaller quantization noise. This way, the Decoder can reconstruct the input frames more precisely. The dotted line is plotted just to show the general trend of the curve, but does not represent a set of feasible (message length - PSNR) pair. Since the quantization strategy uses a fixed number of bits, there is no way to reach an average message length in the region between the two points tested. The blue curve represents the Pareto frontier for the fixed quantization strategy. This way, the curve which represents the true rate-PSNR performance is the stair plot. The region under the plot is colored since it is a feasible region. Meaning that it is possible to design a compression scheme that works in this region.

The plot represented in pink shows the results for the dynamic Sensor policy. In particular, the results shows seven average PSNR values obtained with seven different values of the trade-off parameter β.

In Tab. 5.5 the results of the average message length and the average PSNR for different values of β are shown. The values of β are arbitrarily chosen in order to allow the trade-off between receiver performance and communication cost to be meaningful to the Sensor. The PSNR values increase as the β decreases, this is because the Sensor chooses more frequently the higher bits quantizer leading to an average higher accuracy. From Fig. 5.5 it is possible to see that the pink plot is always over the blue plot meaning that there is a gain in overall performance when using the dynamic quantization strategy at the Sensor side. This means that the adaptive quantization strategy is Pareto dominant with respect to the fixed quantization solution. Despite the feasible region where this model can operate is the one below the curve, the higher performance is obtained when the system is working near the curve. The reason is that, near these values, it is possible to guarantee the indicated performance at the receiver while using the least communication

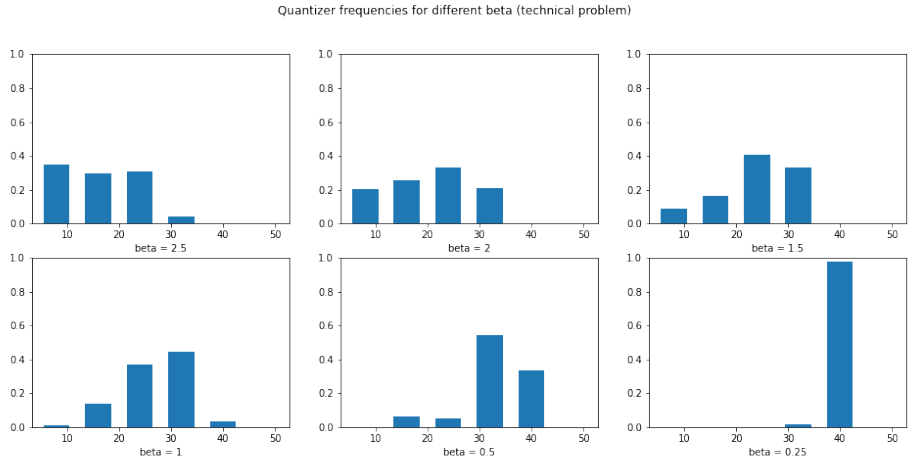

Figure 5.6: Frequencies of the different quantizers for different values of β.

resources.

As the pink curve reaches the minimum and the maximum average number of bits for the message, it is possible to see that the performance is equal to the fixed quantization strategy. Indeed, in these cases, the Sensor policy will use respectively the minimum and the maximum number of quantization bits every time step, reaching the same performance of the fixed quantization strategy.

In Fig. 5.6 the frequencies of the different quantizers for different values of β are shown. It is possible to see that the distribution of the encoding message length is biased towards the lower lengths for higher β and towards the higher lengths for smaller β. This result confirms the general trend which is possible to see in Fig. 5.5 and in Tab. 5.5.

5.5 Results for the semantic problem

This section analyzes the results for the semantic problem. As in the previous section, the fixed quantization strategy and the dynamic quantization strategy are compared in order to evaluate the results. Recall that in the semantic scenario, the reward function for the Sensor RL problem is given by minus the MSE between the physical state and the estimated physical state at the receiver side, minus a penalty term for the communication resources employed. In the training and testing phase, the receiver has six regression networks available and previously trained which estimate the physical state values given the encoded message and the quantizer used.

In Fig. 5.7 it is possible to see the semantic communication problem results. The fixed

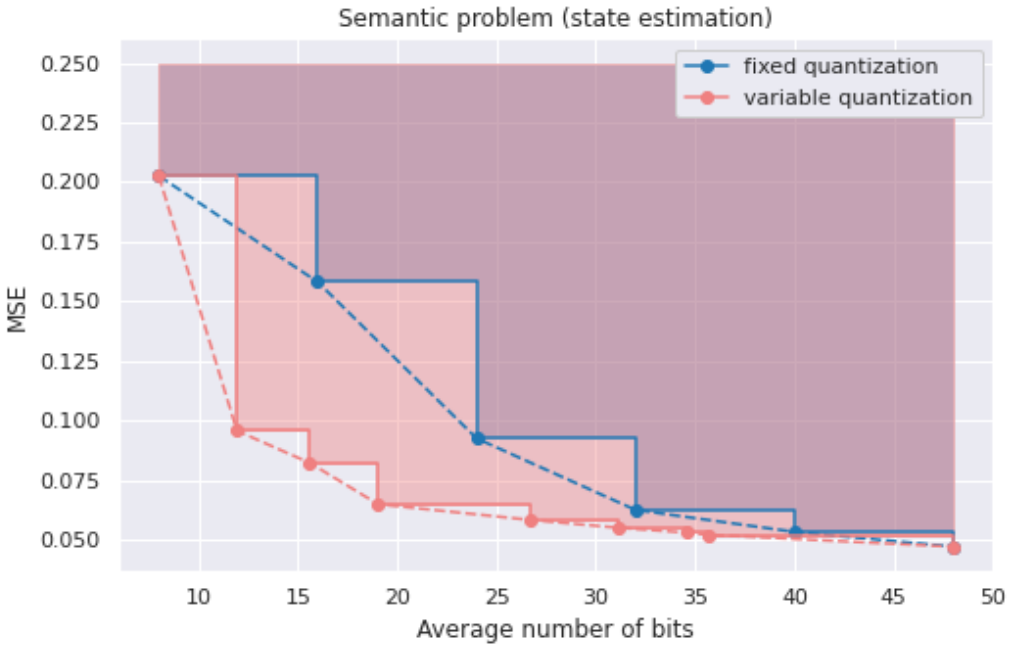

Figure 5.7: Results for the semantic communication problem.

strategy curve is reported in blue. It is possible to see that the estimation error decreases as the message length increases. This means that a higher precision in the quantization of the features yields also a more precise estimation. Another thing that it is worth noting is that, among the four state values, the position x and the angle θ obtain an average lower error with respect to the velocity $\dot{x}$ and angular velocity $\dot{\theta}$. The reason is that, given a noisy variable, the derivative of this variable will noisier because the errors on the original values of the variable are added.

The pink curve represents the results obtained with the dynamic quantization strategy. It is possible to see that the pink curve is under the blue one, meaning that using a dynamic Sensor policy allows to improve semantic performance while keeping the communication rate lower.

For both the plots it is possible to see that the feasible region where the system model can operate is the region above the curve. Indeed, the (length-MSE) pairs sets a lower bound on the estimation error which can be reached by the system. Any other sub-optimal strategy will stay in the region above the curves.

In Tab. 5.6 it is possible to see the average message length and MSE for different values of the trade-off parameter β. The trend is the same of the technical problem:

β	**Average message length [bits]**	**MSE**
0.0001	35.6792	0.0521
0.001	34.5910	0.0530
0.005	31.1811	0.0550
0.01	26.7286	0.0583
0.025	19.0424	0.0649
0.05	15.6087	0.0822
0.1	11.939	0.0957

Table 5.6: Results average message length a MSE for different values of β.

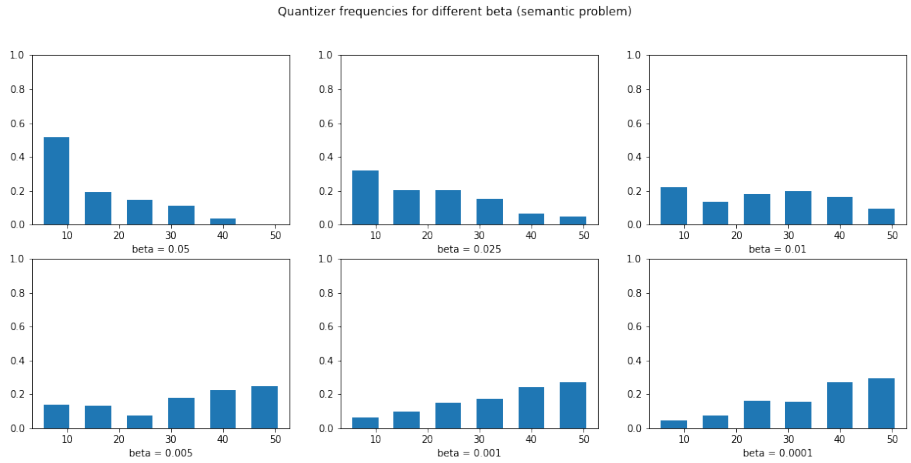

Figure 5.8: Frequencies of the different quantizers for different values of β.

as the communication cost increases, the average message length and the performance decrease. In this case, the fact that the MSE increases translates in the worsening of the performance.

In Fig. 5.8 the distributions over the different quantizers are shown. The seven bar plot correspond to the seven values of β which have been used. As expected, it is possible to see that, for a high communication cost, the smaller codebook quantizers are preferred. As the communication cost lowers, a higher number of bits is employed to represent the features. In the middle, it is possible to appreciate a distribution spread over the six quantizers.

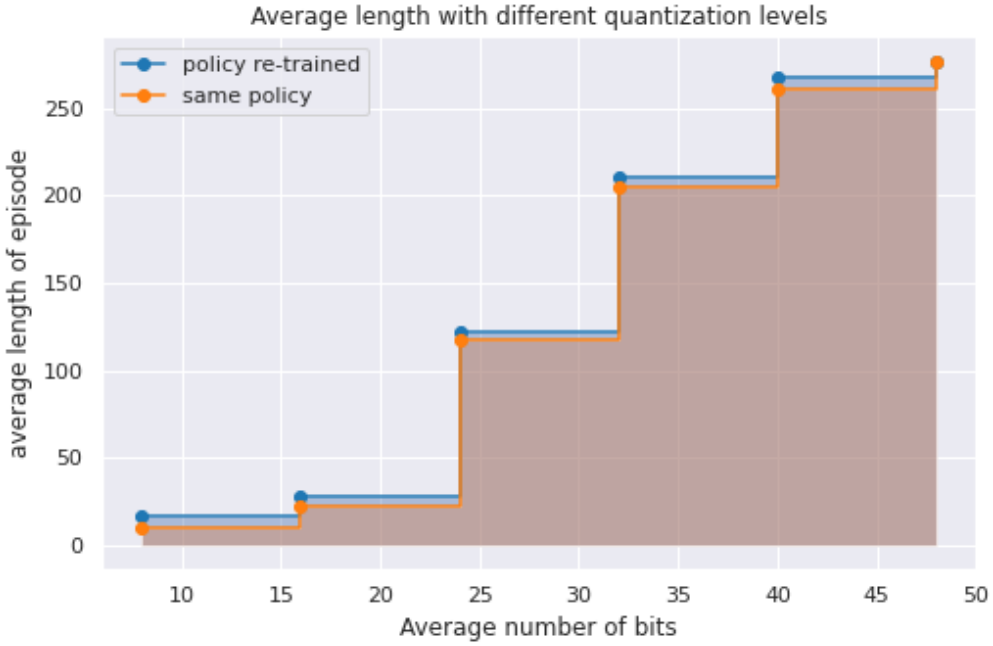

Figure 5.9: Results for the effectiveness communication problem and a fixed quantization strategy.

5.6 Results for the effectiveness problem

In this section the results of the effectiveness communication problem are reported. To measure the way the encoding strategy affects the Controller behavior, the performance at the receiver is tracked measuring the episode length.

In Fig. 5.9, the results for the communication problem are reported. The figure shows two analysis that have been accomplished. In the figure, the episode length is used as a measure of the performance of the agent. Two curves report the average episode lengths for different average numbers of bits.

The blue plot shows the result for the fixed quantization strategy and a dedicated policy for every quantizer. This means that the Sensor fixed the quantizer to use and then the Controller learns a policy through the RL procedure. Then, the results over 100 episodes are evaluated to yield an average episode length.

The orange curve shows the performance of the Controller when the policy trained on the quantizer with more codewords is used. In this case, the policy has been trained only ones for the quantizer with 64 codewords ad then this policy is used with lower cardinality

Number of codewords	Number of training episodes	Average length
64	8000	277.24
32	12000	268.11
16	16000	210.68
8	20000	121.80
4	20000	27.48
2	8000	17.14

Table 5.7: Number of training episodes and average final performance for the different size of codebook used.

dictionaries. This is possible due to the fact that the quantizer is using always the same Encoder to extract the features. This way, using a different quantizer introduces more distortion in the latent vectors which encode the features, but the space represented by the features is always the same.

In the figure it is possible to see that the performance on the blue curve is always slightly better than the orange. This result confirms the theory on POMDP. Indeed, a procedure which learns a new policy for every different quantizer will converge to the optimal solution for any POMDP. So, this policy trained specifically with a given quantizer will be better than any possible policy that the system can find. These results confirm the optimality of the solution which Q-learning find.

In Tab. 5.7, the number of episodes that have been used in the training loop of the RL procedure are listed. It is possible to see that, in order to converge to a stable policy (which does not improve anymore), the number of episodes to use changes with the number of codewords in the quantizer. This phenomenon can be explained by the fact that a lower number of quantization bits introduces more noise in the compressed observation. A noisier state for the Controller means that the RL problem that has to be solved is more "partially observable". As demonstrated in Sec. 2.4, POMDP can still be solved by Q-learning with the condition that the agent visits a state infinitely many times. A more noisy state will introduce higher variance in the Q-value estimation and thus the agent has to see the state a lot of times to converge near to the true Q-value. On the other hand, if the noise in the state is smaller (less partially observable) the variance on the estimation of the cumulative return will be smaller and the RL procedure will converge more rapidly. However, it is possible to see that for the 2 codewords quantizer the number of training episodes is 8000 which is smaller than the previous one. This result can be explained by the fact that this small codebook encodes the features with a smaller complexity. Indeed, 2 codewords for eight features means that the total number of possible messages is 256 (2^8). With such a low number of states, the Q-function can

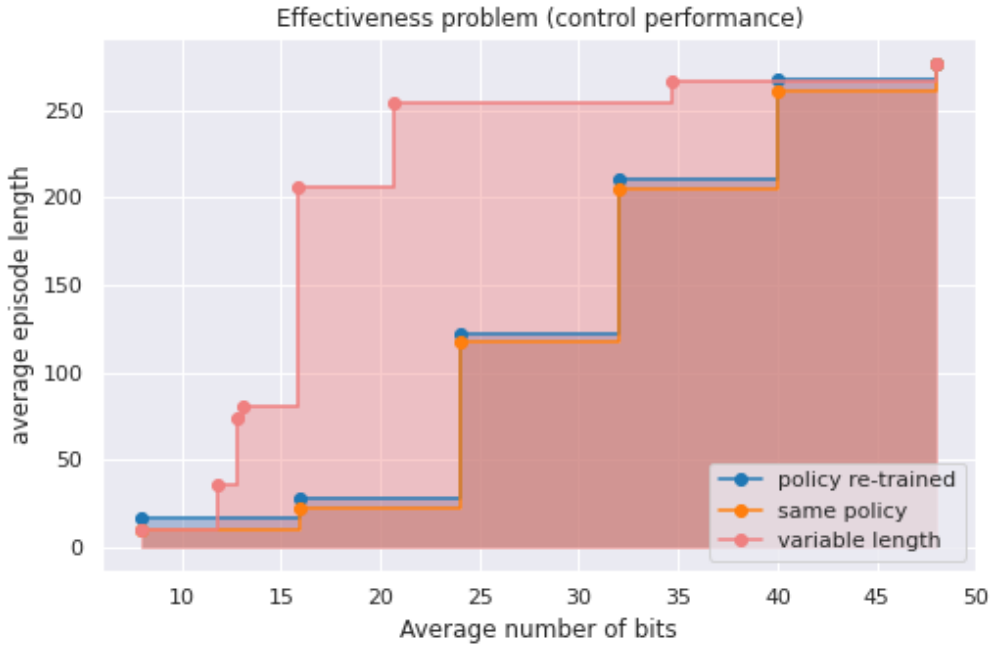

Figure 5.10: Results for the effectiveness communication problem.

learn the values faster even if these values are noisier. In general, it is possible to say that the convergence speed of the training procedure is a trade-off between the complexity and the noise that the state introduces. A higher complexity slows down the convergence, however, less noise will still be able to guarantee less variance and faster convergence. On the other side, higher noise introduces more variance and, even if the state exhibits less complexity, the convergence will be delayed.

Looking a Fig. 5.10, it is possible to see that the policy trained on the higher quality state still performs near the optimal values when used with smaller codebook quantizers. This result allows to use the same policy for every quantization strategy and still guarantees near optimal performance. Moreover, the fact that this policy used the smaller number of episodes to converge allows to reduce the training period before being able to deploy the system.

In Fig. 5.10, the results of the variable quantization strategy are compared with the previous results. It is possible to see that the dynamic quantization strategy reaches higher accuracy for smaller average message lengths. In this case, the gap between the fixed strategy and the dynamic one is a lot higher than in Level A and B. This result shows how optimizing the system for an effective communication can obtain higher results than

β	**Average message length [bits]**	**Average length of the episode**
1	8	9.4040
0.7	11.7827	36.23
0.5	12.824	73.92
0.4	13.1104	80.81
0.33	15.8968	205.79
0.2	20.6696	254.52
0.1	34.712	266.396

Table 5.8: Results for the effectiveness problem fro different values of β.

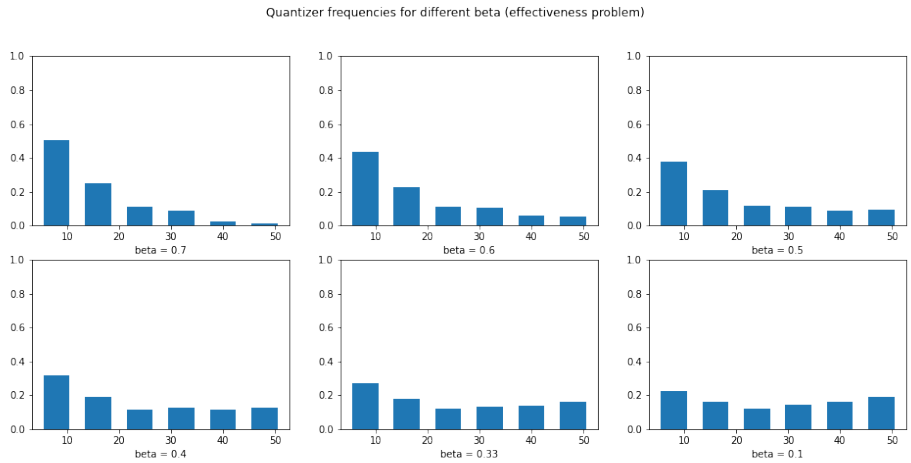

Figure 5.11: Frequencies of the different quantizers for different values of β.

considering standard metrics such as the distortion of the reconstruction error. However, it is important to note that this method considers the policy already trained and so this quantization strategy can be obtained only once the Controller has learned a control policy.

In Tab. 5.8 the performance in terms of average length of the episode is reported. These results show the average message length with respect to the average performance in the effectiveness communication problem.

In Fig. 5.11, the plot of the quantizer frequencies for different values of β are reported. These results confirms the trend which was seen for the level A and B: for a higher communication cost, the average length decreases and the policy will choose the lower size quantizers.

5.7 Intuition behind the different levels of quantization

In this section, a heuristic evaluation of the dynamic quantization of the observation is done. In particular, given the Sensor policy $\kappa(K \mid o)$ and the Control policy $\pi(a \mid z^{(K)})$ already trained, a relationship between the number of bits used for the quantization with respect to the Controller policy is outlined. This result is obtained by testing the optimal dynamic quantization strategy for the effectiveness problem. This test considers 500 episodes where the system model works with the two agents (Sensor and Controller) exploiting their policies. In particular, the Sensor uses the policy to select the number of codewords to quantize the observation frames while the Controller uses its policy to select the best action to take according to the received message. All the actions of the two agents are collected and stored. Let's call the dataset containing all the quantization choices and the actions taken $D = \{(K_1, a_1), ..., (K_N, a_N)\}$.

After the 500 episodes, the physical state space is divided in small intervals of state. Only two variables are used: the θ and $\dot{(\theta)}$. Combining the two variables, it is possible to obtain a 2D map of the state space divided in small squares (bins) according to the intervals. In particular, 70 intervals have been considered for both the variables. The interval considered for the angle is [-0.10, 0.10], whereas for the angular velocity is [-1.3, 1.3]. This setting creates a 2D map containing 1400 bins. Let's define two matrices M_K and M_a. The entries of the matrix M_K store the expected number of bits used to quantized the observations for every θ and $\dot{\theta}$ interval:

$$M_K^{i,j} = \sum_{i=1}^{N} \frac{\delta(K_i) \log_2 K_i}{\delta(K_i)} \tag{5.7}$$

where $\delta(K_i)$ is the indicator function which is 1 if the action K_1 is done inside the state region of the bin (i, j) and 0 otherwise. $\log_2 K_i$ is used to obtain the number of quantization bits which is the logarithm of the number of codewords of the quantizer chosen.

In the matrix M_a the expected action taken is stored. Since the number of action are two, the values 1 and 0 are used to represent the actions "left" and "right". In the matrix M_a, the entries are:

$$M_a^{i,j} = \sum_{i=1}^{N} \frac{\delta(a_i) a_i}{\delta(a_i)} \tag{5.8}$$

where $\delta(a_i)$ is the same indicator function defined before but for the action a_i. With

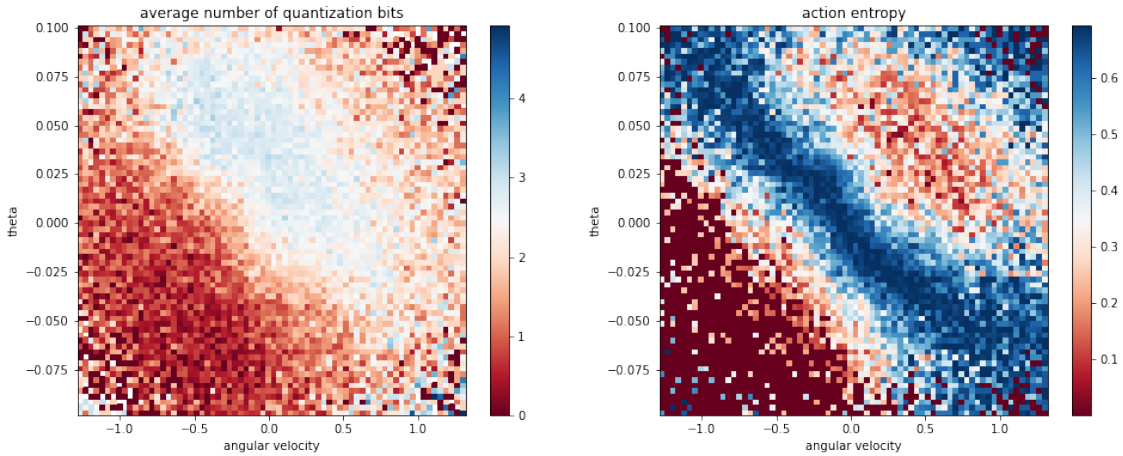

Figure 5.12: Colormap for the average number of quantization bits and the entropy of the actions.

such a definition, it is possible to say that $M_a^{i,j}$ is the probability of taking action 1 in the bin (i, j) whereas $1 - M_a^{i,j}$ is the probability of taking action 0. Comparing these two matrices it is possible to discuss how the two policies behave with respect to each other.

In Fig. 5.12 there are two colormaps which represent the average number of quantization bits and the entropy of the actions. In the first map, it is possible to see the average number of bits used to quantize the state for every bin. In the red region, few bits are used on average to quantize the relative observations. As the color changes to blue, a higher number of bits are used respectively. The second map shows the entropy of the action for every bin. The entropy $H(x)$ of a random variable x can be obtained as

$$H(x) = -\mathbb{E}[\log_2 p(x)] = -\sum_x p(x) \log_2 p(x). \tag{5.9}$$

In the case of a binary variable, as in the case of the Controller action, the entropy is easy to obtain as

$$H^{i,j}(a) = -M_a^{i,j} \log_2 M_a^{i,j} - (1 - M_a^{i,j}) \log_2 (1 - M_a^{i,j}), \tag{5.10}$$

where $H^{i,j}(a)$ is the action entropy relative to the bin (i, j).

The entropy of a random variable represents the uncertainty of the outcome of that variable. Indeed, Eq. 5.9 for a binary variable with probability p has a maximum for $p = \frac{1}{2}$ and is equal to 0 for $p = 0$ and $p = 1$. The maximum of the entropy corresponds to

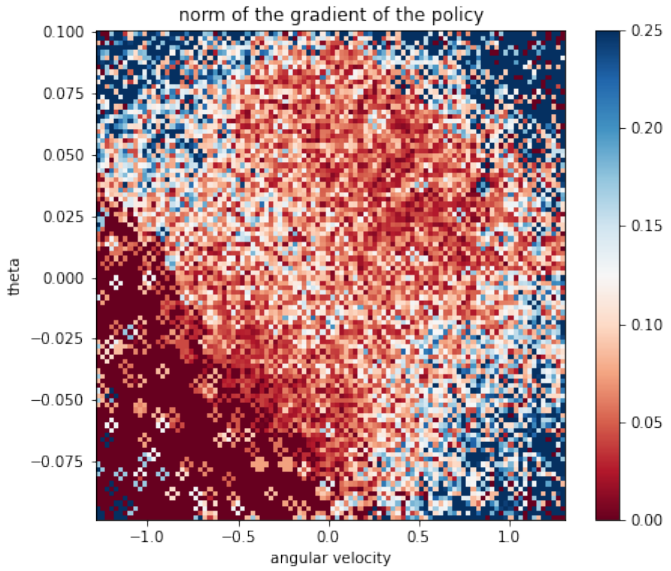

Figure 5.13: Magnitude of the gradient of the Controller policy.

the most uncertain case where the variable can be either 1 or 0 with the same probability. It is possible to say that Fig. 5.12 shows how the communication resources are allocated with respect to the confidence on the action to take. Intuitively, in a region where the Controller knows with high confidence (high action probability) the action to take, the Sensor can learn to compress more the observation. This means that, even if less details are available to the Controller, it will still be able to take the correct action. On the other hand, if the Controller is uncertain about the action to take (the two actions have similar probabilities) the Sensor should provide a more precise representation of the observations and thus use more communication resources. However, the uncertainty on the action might not fully depend on the amount of information that is lost when quantizing the features, but it could be due to the partial observations collected by the Sensor. This uncertainty cannot be alleviated by sending more precise features. Another quantity that can be used to discuss the Sensor policy choices is the derivative of the Controller policy.

In Fig. 5.13, the magnitude of the policy with respect to the angular velocity and the angle is shown. The intuition behind this figure is that, in the regions that the gradient norm is higher, the action is changing rapidly from one interval to the other. Similarly to the entropy, this means that the Controller is not certain about the action to take in this state. Hence, noise introduced by the quantization procedure can confuse the Controller.

The ideal policy of the Sensor should provide lower noise in the states where the gradient is higher and can tolerate higher noise in the state where the Controller policy is constant.

6
Conclusion

In this project a remote reinforcement learning problem has been implemented in a specific control problem. In particular, the interaction between a remote Sensor which is able to observe the Environment and a Controller which can take actions in the Environment has been studied. The problem consisted on communicating sequences of frames from Sensor to the Controller through a digital communication channel with limited capacity. In the first analysis methods for implementing encoding of the observations have been proposed. Tests on the possibility of learning compact representation of the input observations were accomplished by means of an Autoencoder. Subsequent tests demonstrated how it is possible to learn a Control policy using the encoded frames and highlighted the Reinforcement Learning procedure to discover this policy. These tests demonstrated that it is possible to build such a system model and proved the feasibility of certain performance without the channel.

After this, a digital communication channel with limited capacity has been introduced, inducing a bottleneck on the amount of information that is possible to send from the Sensor to the Receiver. A quantization step has been introduced to ensure that the encoded frames were easily communicated through the channel. Two types of quantization strategies have been proposed implemented: a fixed quantization and a dynamic quantization.

Once obtained this versatile compression system, three communication levels have been analyzed. The first is the technical problem which considers the possibility of reconstructing the original message at the receiver. The PSNR between the original observations and the ones reconstructed at the receiver has been used as a performance metric. A general trend observed is that the reconstruction accuracy increases as more bits are used to

quantize the encoded observations. A less trivial result, is that the dynamic quantization strategy that the Sensor learns to allocate the quantization bits efficiently, outperforms the fixed quantization strategy.

The second level of communication considered is the semantic communication. In particular, the possibility to communicate relevant information about a process which can be retrieved by the receiver. Even for this case, it is possible to reduce the communication rate while maintaining good average performance with respect to the fixed quantization strategy. This can be explained by the fact that the loss of semantic information due to the quantization is not equal for all the frames. With a dynamic quantization it is possible to achieve an operational rate which is optimized to the regression task which the receiver has to solve.

The third communication problem studied is the real remote reinforcement learning setting. In this scenario, the Controller has to control the CartPole keeping it vertical for as many steps as possible. The only information available to the receiver is the compressed frame that the Sensor communicates. Two fixed quantization performance are evaluated, considering the condition where the policy is retrained for every quantizer or just for the higher quality one. Then, a comparison with the dynamic quantization strategy is given. A great improve in the performance can be obtained when using the different quantizers to encode the system. Recalling the arguments about the information bottleneck, it is possible to say that the dynamic quantization improves the bottleneck, reducing the redundant information and letting pass only the relevant one.

A final heuristic study has been done to visualize the Sensor policy in the Level C communication scenario. This analysis proposed some insights about how the policy of the Sensor is related to the policy of the Controller. In particular, it has been proved how the communication resources are minimal in regions with low action entropy, showing that the Controller can still retrieve the action correctly even with low resolution information. On the other hand, more communication resources are allocated in the region of higher action entropy. In the latter case, there is no superposition between the Sensor policy and the action entropy, but only a general trend is observed.

To conclude, this work proposed a reinforcement learning method to optimize communication in a remote reinforcement learning scenario. The framework provided is general and allows to optimize the rate for multiple levels of communication by designing specific reward signals. This project has been developed in a simulated environment without a real communication channel between the Sensor and the Controller. Further analysis might consider more realistic scenario where the communication channel might affect the overall performance due to packet losses. On the simulation part, another aspect to inves-

tigate could be the possibility of jointly training the different building block of the system model, merging the proposed algorithmic procedures in a single one.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] MATLAB. Reconstruct inputs to detect anomalies, remove noise, and generate images and text. [Online]. Available: https://www.mathworks.com/discovery/autoencoder.html

[3] E. Bourtsoulatze, D. B. Kurka, and D. Gündüz, "Deep joint source-channel coding for wireless image transmission," *IEEE Transactions on Cognitive Communications and Networking*, vol. 5, no. 3, pp. 567–579, 2019.

[4] A. Van Den Oord, O. Vinyals *et al.*, "Neural discrete representation learning," *Advances in neural information processing systems*, vol. 30, 2017.

[5] J. Cheng, W. Chen, F. Tao, and C.-L. Lin, "Industrial iot in 5g environment towards smart manufacturing," *Journal of Industrial Information Integration*, vol. 10, pp. 10–19, 2018.

[6] S. Kaul, R. Yates, and M. Gruteser, "Real-time status: How often should one update?" in *2012 Proceedings IEEE INFOCOM*. IEEE, 2012, pp. 2731–2735.

[7] X. Zheng, S. Zhou, and Z. Niu, "Urgency of information for context-aware timely status updates in remote control systems," *IEEE Transactions on Wireless Communications*, vol. 19, no. 11, pp. 7237–7250, 2020.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[9] M. Jankowski, D. Gündüz, and K. Mikolajczyk, "Deep joint source-channel coding for wireless image retrieval," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 5070–5074.

[10] W. Weaver, "Recent contributions to the mathematical theory of communication," *ETC: a review of general semantics*, pp. 261–281, 1953.

[11] Y. Cui, B. Hou, Q. Wu, B. Ren, S. Wang, and L. Jiao, "Remote sensing object tracking with deep reinforcement learning under occlusion," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–13, 2021.

[12] X. Shen, B. Liu, Y. Zhou, J. Zhao, and M. Liu, "Remote sensing image captioning via variational autoencoder and reinforcement learning," *Knowledge-Based Systems*, vol. 203, p. 105920, 2020.

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[14] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[15] B. C. Stadie, S. Levine, and P. Abbeel, "Incentivizing exploration in reinforcement learning with deep predictive models," *arXiv preprint arXiv:1507.00814*, 2015.

[16] M. Pinsky and S. Karlin, *An introduction to stochastic modeling*. Academic press, 2010.

[17] C. Szepesvári, "Algorithms for reinforcement learning," *Synthesis lectures on artificial intelligence and machine learning*, vol. 4, no. 1, pp. 1–103, 2010.

[18] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[19] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[20] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[21] L. Bottou, "Stochastic gradient descent tricks," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 421–436.

[22] M. T. Spaan, "Partially observable markov decision processes," in *Reinforcement Learning*. Springer, 2012, pp. 387–414.

[23] T. Jaakkola, M. Jordan, and S. Singh, "Convergence of stochastic iterative dynamic programming algorithms," *Advances in neural information processing systems*, vol. 6, 1993.

[24] I. T. Jolliffe and J. Cadima, "Principal component analysis: a review and recent developments," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, p. 20150202, 2016.

[25] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4, no. 4.

[26] M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE journal*, vol. 37, no. 2, pp. 233–243, 1991.

[27] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[28] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[30] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[31] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[33] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[34] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[35] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework," 2016.

[36] A. Trushkin, "Sufficient conditions for uniqueness of a locally optimal quantizer for a class of convex error weighting functions," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 187–198, 1982.

[37] Y. Linde, A. Buzo, and R. Gray, "An algorithm for vector quantizer design," *IEEE Transactions on communications*, vol. 28, no. 1, pp. 84–95, 1980.

[38] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[39] N. Tishby, F. C. Pereira, and W. Bialek, "The information bottleneck method," *arXiv preprint physics/0004057*, 2000.

[40] J. Wan, D. Zhang, S. Zhao, L. T. Yang, and J. Lloret, "Context-aware vehicular cyber-physical systems with cloud support: architecture, challenges, and solutions," *IEEE Communications Magazine*, vol. 52, no. 8, pp. 106–113, 2014.

[41] M. Groshev, C. Guimarães, J. Martín-Pérez, and A. de la Oliva, "Toward intelligent cyber-physical systems: Digital twin meets artificial intelligence," *IEEE Communications Magazine*, vol. 59, no. 8, pp. 14–20, 2021.

[42] S. Tatikonda and S. Mitter, "Control under communication constraints," *IEEE Transactions on automatic control*, vol. 49, no. 7, pp. 1056–1068, 2004.

[43] S. Tatikonda, A. Sahai, and S. Mitter, "Stochastic linear control over a communication channel," *IEEE transactions on Automatic Control*, vol. 49, no. 9, pp. 1549–1561, 2004.

[44] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[45] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE transactions on systems, man, and cybernetics*, no. 5, pp. 834–846, 1983.

[46] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

[47] Y. Takida, T. Shibuya, W. Liao, C.-H. Lai, J. Ohmura, T. Uesaka, N. Murata, S. Takahashi, T. Kumakura, and Y. Mitsufuji, "Sq-vae: Variational bayes on

discrete representation with self-annealed stochastic quantization," *arXiv preprint arXiv:2205.07547*, 2022.

[48] A. Łańcucki, J. Chorowski, G. Sanchez, R. Marxer, N. Chen, H. J. Dolfing, S. Khurana, T. Alumäe, and A. Laurent, "Robust training of vector quantized bottleneck models," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–7.

Acknowledgments

I would like to thank Professor Andrea Zanella for the opportunity to work on this project, learning a lot about these interesting topics. Also, I am very thankful to Federico Chiariotti and Francesco Pase for the time they dedicated to me and the support they gave me to take the project forward. I would like to thank my parents and my sister for helping and encouraging me during my studies. Finally I would like to express my thanks to my girlfriend Beatrice who supported me during the years at the University.