UNIVERSIDAD DE INVESTIGACIÓN DE TECNOLOGÍA EXPERIMENTAL YACHAY

Escuela de Ciencias Matemáticas y Computacionales

VIRTUAL ROBOT DRIVEN BY A SELF-TAUGHT AGENT FOR PROTEIN FOLDING SIMULATION

Trabajo de integración curricular presentado como requisito para la obtención del título de Ingeniero en Tecnologías de la Información

Autor:

Ulloa Bonilla Bryan Mauricio

Tutor:

Ph. D. - Rolando Armas

Co-Tutor:

Ph. D. - Fernando Gonzales

Urcuquí, diciembre 2023

Autoría

Yo, **Ulloa Bonilla Bryan Mauricio**, con cédula de identidad **0250030186**, declaro que las ideas, juicios, valoraciones, interpretaciones, consultas bibliográficas, definiciones y conceptualizaciones expuestas en el presente trabajo; así cómo, los procedimientos y herramientas utilizadas en la investigación, son de absoluta responsabilidad de el autor del trabajo de integración curricular. Así mismo, me acojo a los reglamentos internos de la Universidad de Investigación de Tecnología Experimental Yachay.

Urcuquí, Diciembre del 2023.

Bryan Ulloa

CI: 0250030186

Autorización de publicación

Yo, **Ulloa Bonilla Bryan Mauricio**, con cédula de identidad **0250030186**, cedo a la Universidad de Tecnología Experimental Yachay, los derechos de publicación de la presente obra, sin que deba haber un reconocimiento económico por este concepto. Declaro además que el texto del presente trabajo de titulación no podrá ser cedido a ninguna empresa editorial para su publicación u otros fines, sin contar previamente con la autorización escrita de la Universidad.

Asimismo, autorizo a la Universidad que realice la digitalización y publicación de este trabajo de integración curricular en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Urcuquí, Diciembre del 2023.

Bryan Ulloa

CI: 0250030186

Dedication

This thesis project is dedicated mainly to God, for having given me life and allowing me to have reached this important moment in my professional formation, to my family and friends, but in a very special way:

To my Grandparents,
who are not physically present to see this achievement that they so longed for, but I know that from heaven, they will be very proud that I have achieved this goal. I miss them a lot!.

To my Mommy Elsita,
I will always be eternally grateful for being my grandmother and I never lack her support in my life. This achievement is for you mommy Elsita. I love you!.

To my Parents,
for their unconditional love and support, they knew how to guide me during these years, I will always be eternally grateful for instilling in me the best values and education. You are the reason for my growth and improvement. I love you so much!.

To my sister,
despite having good and bad moments, never forget that I love you very much and I hope not to be an example in your life, but rather, unconditional support in your future.

Thank you very much for being an important part of my life during these years, for your support and for never mistrusting me.

Bryan Ulloa.

Acknowledgments

I extend my sincere gratitude to God for granting me the opportunity to reach this moment and for introducing me to remarkable individuals during these significant years. Special thanks to the dedicated academic staff at the Experimental Technology Research University Yachay for imparting an international quality education, evident in my current experiences.

I extend my deep gratitude to Professor Oscar Chang, my mentor, whose unwavering support and invaluable knowledge were instrumental in the completion of this project. Despite his departure from the university, his contributions are acknowledged, and I am sincerely thankful for everything.

My heartfelt thanks to my extended family, uncles, aunts, and cousins who provided diverse support throughout this significant phase of my life. Special appreciation to Israel, more than a cousin, always a brother, for his invaluable advice and unwavering support. Likewise, Javi, thank you very much Luchito for supporting me during the time I was doing the thesis, you are a greath person. I will live forever grateful to all of you.

To my dear friend Jhon, your enduring friendship and support since our university days have been foundational to my journey. To my best friends Thaly, Edisao, and Karencita, I can't imagine this journey without you, and I cherish our bond deeply. And to my friends Said, Yaacov, Paul, Marlon, Carlos, Andy, Henry, Jhonsa, Mauro, Pedro, Pily, Cynthia, and Daya, thanks to all of you for the memories, anecdotes, and experiences that we lived, grateful to life for having met them. I will always carry them in my heart and I hope to see you soon.

A special acknowledgment to the people of Urcuquí, especially Doña Mary and her sons, for embracing me as part of their family. And to Tokycita, your unwavering support has made a significant impact on my life, and I am forever grateful for the experiences we've shared.

Abstract

Currently, artificial intelligence (AI) has taken an essential role in many fields of scientific research. AI has proven to be useful for developing powerful algorithms for the control and assembling of complex robotic systems based on neural networks, in particular in the Creation of self-motivated agents capable of exploring new solutions in arbitrarily complex electromechanical environments.

This project proposes the study and improvement of a virtual multi joint robot driven by a self-motivated neural agent, capable of learning efficient protein folding policies by itself. The robot represents a peptide chain belonging to the human coronavirus Hemagglutinin-Esterase (HEs) protein, and the associated agent acquires through reinforcement learning the capacity to fold itself into a 3D shape that mimics the structure of mentioned protein. This knowledge could be very important in the manufacture of drugs that counteract virus infection.

In the operational phase of the project, the neural agent will be complemented with neural networks that support its protein folding memory. These neural networks are trained with look in to the future principies to satisfy the Bellman equation. The final goal is to create an intelligent protein folding robot with the capacity to resolve a selected section of the HEs protein.

Keywords: Protein Folding, Deep Neuronal Networks (DNN), Deep Reinforcement Learning (DRL), Sparse Code.

Resumen

Actualmente la inteligencia artificial (IA) ha tomado un papel fundamental en muchos campos de la investigación científica. La IA ha demostrado ser útil para desarrollar potentes algoritmos para el control y montaje de sistemas robóticos complejos basados en redes neuronales, en particular en la creación de agentes automotivados capaces de explorar nuevas soluciones en entornos electromecánicos arbitrariamente complejos.

Este proyecto propone el estudio y mejora de un robot virtual multiarticular impulsado por un agente neuronal automotivado, capaz de aprender por sí mismo políticas eficientes de plegamiento de proteínas. El robot representa una cadena peptídica perteneciente a la proteína hemaglutinina-esterasa (HEs) del coronavirus humano, y el agente asociado adquiere mediante aprendizaje por refuerzo la capacidad de plegarse en una forma 3D que imita la estructura de dicha proteína. Este conocimiento podría ser muy importante en la fabricación de fármacos que contrarresten la infección por virus.

En la fase operativa del proyecto, el agente neuronal se complementará con redes neuronales que respaldan su memoria de plegamiento de proteínas. Estas redes neuronales se entrenan con miras a los principios futuros requeridas por la ecuación de Bellman. El objetivo final es crear un robot inteligente de plegamiento de proteínas con capacidad para resolver una sección seleccionada de la proteína HEs.

Palabras Clave: Plegamiento de Proteinas, Doblamiento de Proteinas, Redes Neuronales Profundas, Apredizaje Profundo por Refuerzo, Código Esparsido.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Among the chemicals that make up living things, proteins hold the position of greatest significance (bio-molecules). Almost all biological functions rely on this kind of substance's presence or action. All biological functions, including protein production, are governed by your synthesis [10]. It is understood that DNA and RNA work together to generate proteins within the cell. This process results in a linear chain of distinctive amino acids for each protein, whose structure resembles a one-dimensional thread suspended in a liquid.

Complex combinatorial optimization issues about protein folding appear in various scientific disciplines, including biology, engineering, and medicine [11]. Due to the extreme difficulty in addressing them, these issues are referred to in the computing sciences as NP-hard (there is no known algorithm capable of solving them in polynomial time).

Understanding and treating diseases brought on by improper protein folding, successfully deciphering entire genomes, and the ability to predict the function of a protein by learning its three-dimensional structure from only its linear amino acid sequence serve as the driving forces behind the study of protein folding.

The current study of protein folding relies heavily on artificial intelligence (AI). Qin et al [12]. developed a neural network model to learn how a certain amino acid sequence folds into a protein structure. Similarly, by 2020, DeepMind, an artificial intelligence research facility in the UK, had already made several significant strides. Furthermore, scientists were astounded by the accuracy of the team's method for predicting protein folding when

it was made public in November of the same year [13].

1.2 Problem statement

The so-called protein folding problem is one of these issues that has been researched for more than 50 years in the domains of biochemistry and biophysics, including engineering and medicine.

The study of protein folding has significant implications from a physical and/or biological point of view and in various other disciplines. Recent research has demonstrated a direct link between protein misfolding and particular cell dysfunction for several diseases, including cystic fibrosis and familial pulmonary emphysema [14]. Effective infection depends on interactions between the viral and host cells, specifically the structure of the viral and host proteins.

As with viruses, the infectious agent in infectious illnesses is typically viewed as a macromolecular complex made up of nucleic acids and proteins. Today, viral infection is a major worry because, if it spreads widely, it could negatively affect society's ability to function. Seven coronaviruses are known to infect humans, tragically shown with the devastating COVID-19 effect of the SARS-CoV-2 virus (severe acute respiratory syndrome coronavirus-2),[15]. In order to develop a viral control strategy, it is crucial to comprehend the protein folding mechanisms of infection.

1.3 Objectives

1.3.1 General Objective

- To propose a virtual robot driven by a self-taught neural agent that has the ability to predict and resolved protein folding.

1.3.2 Specific Objectives

- To codify the angles between amino acids in a sparse code to enhance the learning possibilities of the robot.
- To increase the dimensions of the sensors used to detect the near molecular forces.

- To rise the number of sensors of each amino acids.
- To use deep reinforcement learning so that a simulated virtual robot formed by an aminoacid chain, learns to bend itself following protein rules and energy minimization.

Chapter 2

Theoretical Framework

This chapter provides a brief theoretical overview of protein folding and artificial neural networks.

In order to achieve this, a few questions were proposed:

- What proteins are?
- How do proteins fold?
- What are neural networks?
- What is a deep neural network?

To answer these previous queries, the chapter starts with some important concepts that need to be clarified to understand this work.

2.1 Amino acids.

Amino acids are fundamental molecules of life [16]. An amino acid is a molecule in chemistry that contains an amino (-NH2) group, a carboxylic acid (-COOH) group, and a specific R side chain [1], as it is shown in Figure 2.1. There are twenty different amino acids. They are classified according to their polarity and charge, see Figure 2.2, or essentiality in the human diet, see Figure 2.3.

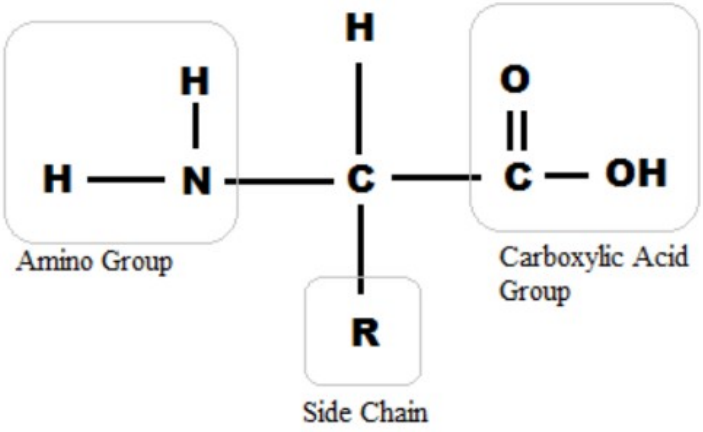

Figure 2.1: Chemical Structure of an Amino acid [1].

Nonpolar	Polar, Negatively Charged	Polar, No Charge	Polar, Positively Charged
Phenylalanine F	Aspartic acid D	Cysteine C	Histidine H
Methionine M	Glutamic acid E	Asparagine N	Lysine K
Tryptophan W		Glutamine Q	Arginine R
Isoleucine I		Threonine T	
Valine V		Tyrosine Y	
Leucine L		Serine S	
Alanine A		Glycine G	
Proline P			

Figure 2.2: Classification of amino acids by polarity and charge [2].

Essential	*Nonessential*
Histidine[a]	Alanine
Isoleucine	Arginine
Leucine	Asparagine
Lysine	Aspartic acid
Methionine	Cysteine
Phenylalanine	Glutamic acid
Threonine	Glutamine
Tryptophan	Glycine
Valine	Proline

[a]Some adults may be able to synthesize histidine on their own

Figure 2.3: Essential and Nonessential Amino Acids [3].

Amino acids are the basic building blocks of proteins and peptides linked together by peptide bonds [17]. Proteins have several thousand amino acid residues, while peptides are shorter chains of a few amino acid residues [18].

The chemistry of amino acid side chains is important in protein structure because these side chains can form covalent bonds to keep a protein structure's shape or conformation [19]. Charged amino acid side chains could form ionic interactions, while polar amino acids can form hydrogen bonds. Hydrophobic side chains interact with one another through weak Van der Waals forces. The great majority of these side chains' bonds are non-covalent. Due to side chain interactions, the sequence and placement of amino acids in a certain protein determine where its bends and folds occur.

The three-dimensional shape of a protein is ultimately determined by its primary structure (its amino acid sequence). Hydrogen bonds between amino and carboxyl groups in adjacent regions of a protein chain can occasionally result in forming particular folding patterns [4]. These stable folding patterns, known as alpha helices and beta sheets, constitute the secondary structure of a protein. Most proteins contain multiple helices, sheets, and other less frequent patterns. Tertiary structure is the collection of formations and folds in a single amino acid linear chain, also known as a polypeptide. A protein's quaternary structure refers to macromolecules made up of multiple polypeptide chains or subunits [4]. As it is shown in Figure 2.4.

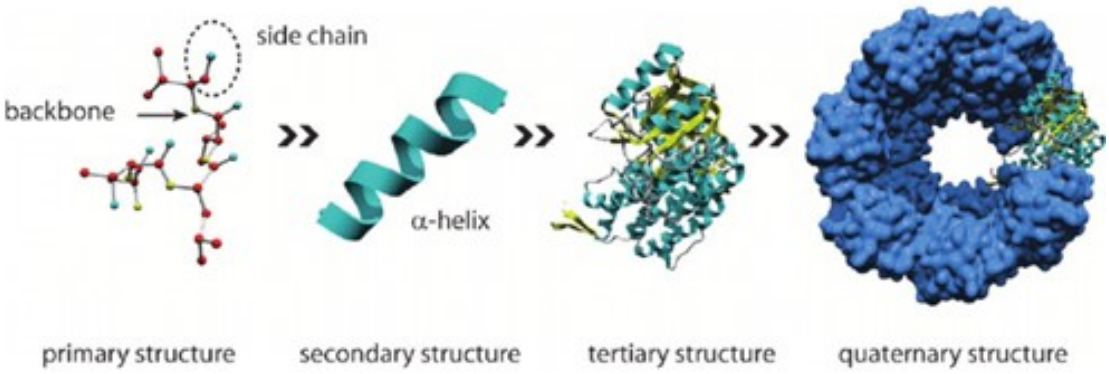

Figure 2.4: The hierarchical structure of proteins [4].

2.2 Proteins.

In reality, proteins are lengthy polypeptide chains consisting of 50 or more "residues", which are not rigid structures.

Many scientists believe that the amino acid sequences of proteins influence the specific structures they will take on [20].

On the other hand, proteins do not exist solely in one fixed conformation but rather in a variety of related ones that create ensembles.

Such transitions have been connected to functionally important phenomena, including enzyme catalysis [21] and allosteric signaling [22]. They occur on length scales ranging from tens of angstroms to nanometers with timespan from nanoseconds to seconds.

While the transitions between these states are the primary focus of protein dynamics research, the nature and equilibrium populations of these states are also relevant aspects of the field.

Energy landscapes provide a conceptual unification of kinetics and thermodynamics energies, wherein the depth of energy wells and the height of energy barriers characterize densely populated states and the kinetics of transitions between them, respectively [23].

The study of proteins, their structure, function, and impact as enzymes, etc., is called "Protein Science." In order to fully grasp how proteins work, one must have an in-depth understanding of their dynamics or the possible configurations the system might take on.

2.3 Protein Folding.

The folding of a polypeptide chain into its native three-dimensional shape is the process through which proteins are created. There is a strong correlation between the structure and function of proteins. A wide variety of molecular interactions stabilizes protein folds.

Thermodynamics explains the factors that lead to protein folding, while kinetics provides insight into the folding process. Protein structure is formed via H-bonds, hydrostatic interactions, hydrophobic effects, and Van der Waals forces.

Different types of interactions can have radically different energy profiles. Folding is enabled by several beneficial interactions, including Vander Waals packing interactions and the Hydrophobic enthalpy effect (H_20 entropy) III. Accumulation of H bonds between

proteins Impact of electricity [20].

Protein folding into its correct native structure is critical to its function. Failure to fold properly results in inactive or toxic proteins that cause various diseases [5].

Protein folding is a complex four-stage process that results in various 3D protein structures required for various human body functions. A protein's structure is organized hierarchically, from primary to quaternary. The various conformations in protein structure are explained by the wide variation in amino acid sequences.

- **Primary structure** refers to the linear sequence of amino-acid residues in the polypeptide chain [5].
- **Secondary structure** is generated by forming hydrogen bonds between atoms in the polypeptide backbone, which folds the chains into either alpha helices or beta-sheets [5].
- **Tertiary structure** of a protein is the geometric shape of the protein. It usually has a polypeptide chain as a backbone, with one or more secondary structures. A tertiary structure is formed by folding the secondary structure sheets or helices into one another. The tertiary structure is determined by the interactions and bonding of the amino acid side chains in the protein [5].
- **Quaternary structure** results from folded amino-acid chains in tertiary structures interacting further to give rise to a functional protein such as hemoglobin or DNA polymerase [5].

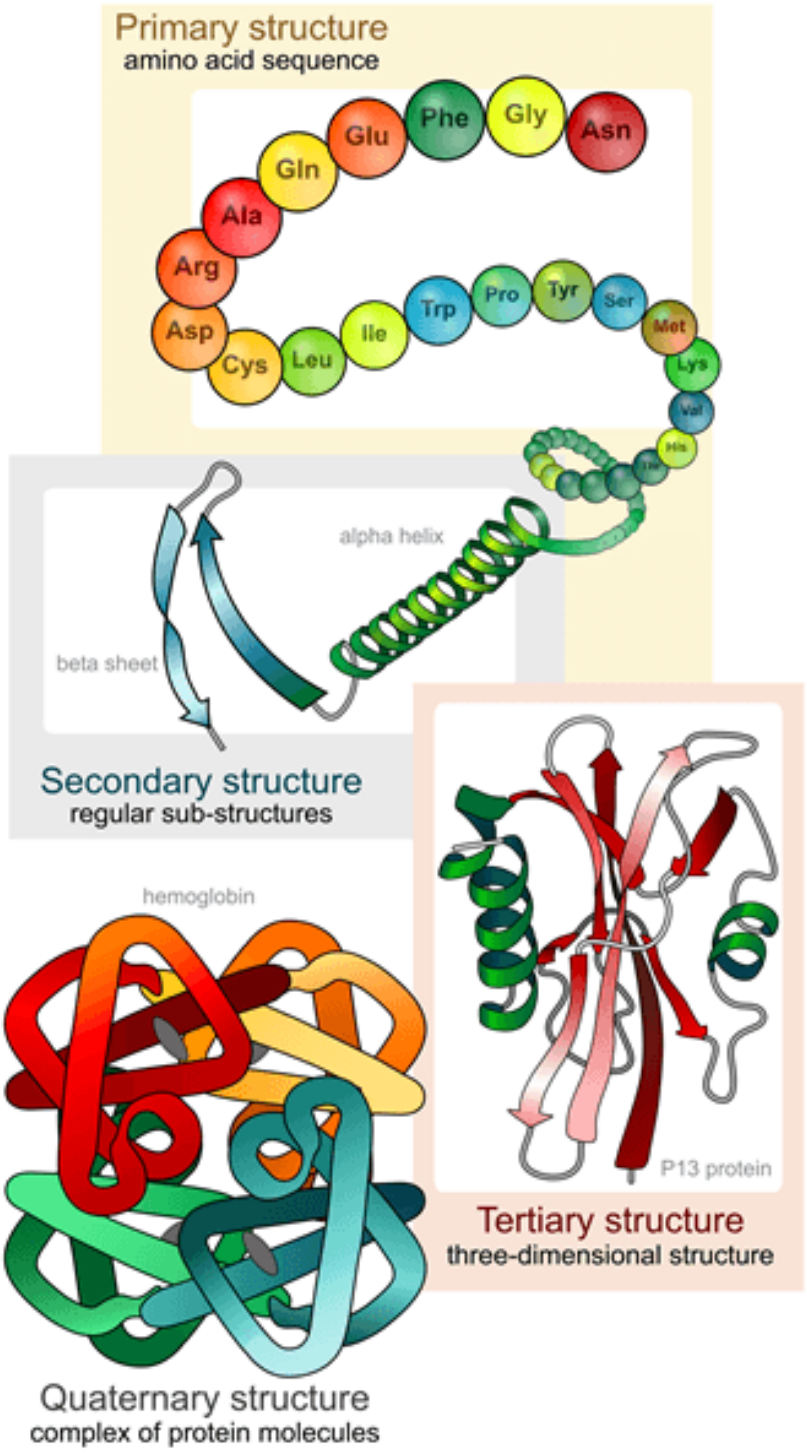

Figure 2.5: Four stages of protein folding [5].

2.4 Bellman Equation.

The Bellman equation expresses the relationship between a state's value and the values of its successor state [24]. Moreover, this is how it is defined:

$$\upsilon_{\pi}(S) = \mathbb{E}_{\pi}\left[G_t | S_t = s\right] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right] \tag{2.1}$$

Where $\mathbb{E}_{\pi}[\cdot]$ denotes the expected value of a random variable if the agent follows policy π and t is any time step, the function υ is known as the state-value function for policy.

Similarly, for policy π, the action-value function defined as $q_{\pi}(s, a)$ defines the value of taking action a in state s under a policy π. This function is defined as follows:

$$q_{\pi}(s, a) = \mathbb{E}_{\pi}\left[G_t | S_t = s, A_t = a\right] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right] \tag{2.2}$$

As a result, the general equation used in reinforcement learning problems is as follows:

$$Q(s, a) = r + \gamma \max_{a'} Q\left(s', a'\right) \tag{2.3}$$

2.5 Q-Learning.

The model-free reinforcement learning technique known as Q-learning was introduced in 1989, which can also be considered an approach to asynchronous dynamic programming [25]. Every state-action pair has one entry in the lookup table of values $Q(s, a)$ (Equation 2.5) maintained by the basic form of Q-learning. It is another reinforcement learning technique that can teach complex task execution without needing an environment model [25]. It functions by gradually improving its assessments of the effectiveness of specific acts in specific states. Therefore, Q-Learning aspires to learn a set of guidelines that instructs an agent on what to do and when to do it. The value of taking action a_t in a state s_t is represented by $Q(s_t, a_t)$. The foundation of the Q-learning algorithm is equation 2.3, which allows for the decomposition of the value $Q(s_t, a_t)$ of a current state and action into the immediate reward r and the discounted maximum future predicted reward following the transition to a next state s_{t+1}. The Bellman equation can be expressed as the following

[26]:

$$Q\left(s_t, a_t\right) = r + \gamma \max_a Q\left(s_{t+1}, a_{t+1}\right) \tag{2.4}$$

Where γ represents the discount factor. The agent computes the value $Q\left(s_{t+1}, a_{t+1}\right)$ and then uses the following equation 2.4 to update its estimate of $Q^*\left(s_t, a_t\right)$. The equation is defined as follows:

$$Q^*\left(s_t, a_t\right) = Q\left(s_t, a_t\right) + \alpha\left[r + \gamma \max_a Q\left(s_{t+1}, a_{t+1}\right) - Q\left(s_t, a_t\right)\right] \tag{2.5}$$

α represents the rate of learning. The maximum value for all actions in the following state is provided by the $\max_a Q\left(s_{t+1}, a_{t+1}\right)$. Since Q-learning changes the Q-values without assuming anything about the policy being followed, it is an off-policy algorithm [26].

2.6 Neural Networks (NN).

A simplified representation of the nervous system's operation is a neural network. The neuron, the fundamental unit, is self-organized, as depicted in the diagram below.

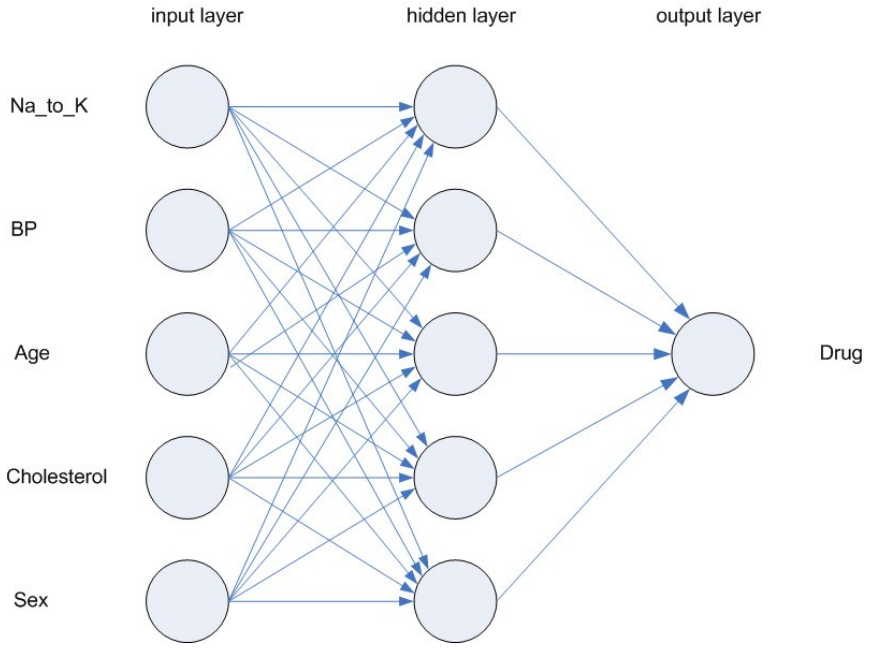

Figure 2.6: Neuronal Network Structure.

A neural network is a streamlined representation of how the human brain functions and are also known as artificial neural networks (ANNs) or simulated neural networks (SNNs)

[27].

Like an abstract neuron, it operates by simultaneously joining several interconnected processing units. Levels are used to organize the processing units. Typically, a neural network is made up of three components [27]. An input level where the units correspond to input fields. A hidden level or levels. A layer of output includes units that stand in for the intended area. The devices are coupled together using various coupling pressures (or weights). At the first level, the inputs are visible, and each neuron in the following level propagates the values. The output file then sends the outcomes [28].

2.7 Deep Neural Network (DNN).

We must first describe the idea of a credit allocation path (CBT) or the number of layers a system needs to finish a task to comprehend deep neural networks. The neural network is considered deep if the CAP index is higher than [29]. Deep neural networks are helpful when independent tasks must take the place of human tasks without sacrificing efficiency. Numerous real-world applications for deep neural networks can be found used.

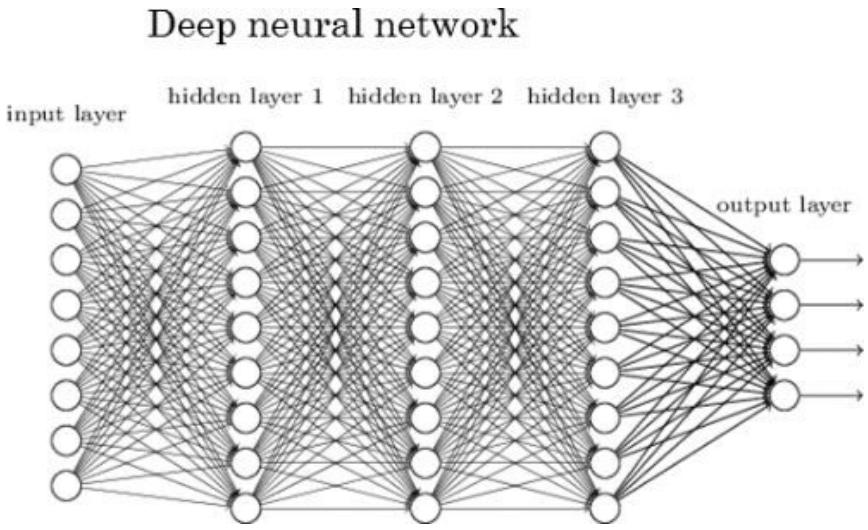

Figure 2.7: Deep Neural Network.

2.8 Convolutional Neural Network (CNN).

Convolutional Neural Networks (CNN) are primarily used for image recognition and are only infrequently used for speech recognition. This CNN is especially true for images since

you do not have to check each pixel individually. A CNN analyzes an image block by block. Start at the top left corner and move pixel by pixel until you get the image. The result of each check is passed through resolution levels with and without associated data functions. Based on this data, the system can generate test results and conclude what the image shows.[29]

2.9 Sparse code.

Sparse coding is a representation in which a small number of neurons are active, with the bulk of neurons showing low activity or even inactive [6]. However, this definition is quite simple. So, taking a deeper look into sparse code, it is possible to view it as a class of unsupervised algorithms for learning sets of over-complete bases to represent data as brain neurons efficiently. In this way, the percentage of actively firing neurons is a crucial and straightforward feature of this code. The density of the code is defined as the average of this proportion of overall information items for a set of N binary neurons (which can be either active or inactive). A sparse code is one with a low density. Nevertheless, this is extremely difficult to understand with just concepts. So, in the next Figure 2.8. it is possible to take a simpler view of the goals of sparse representation.

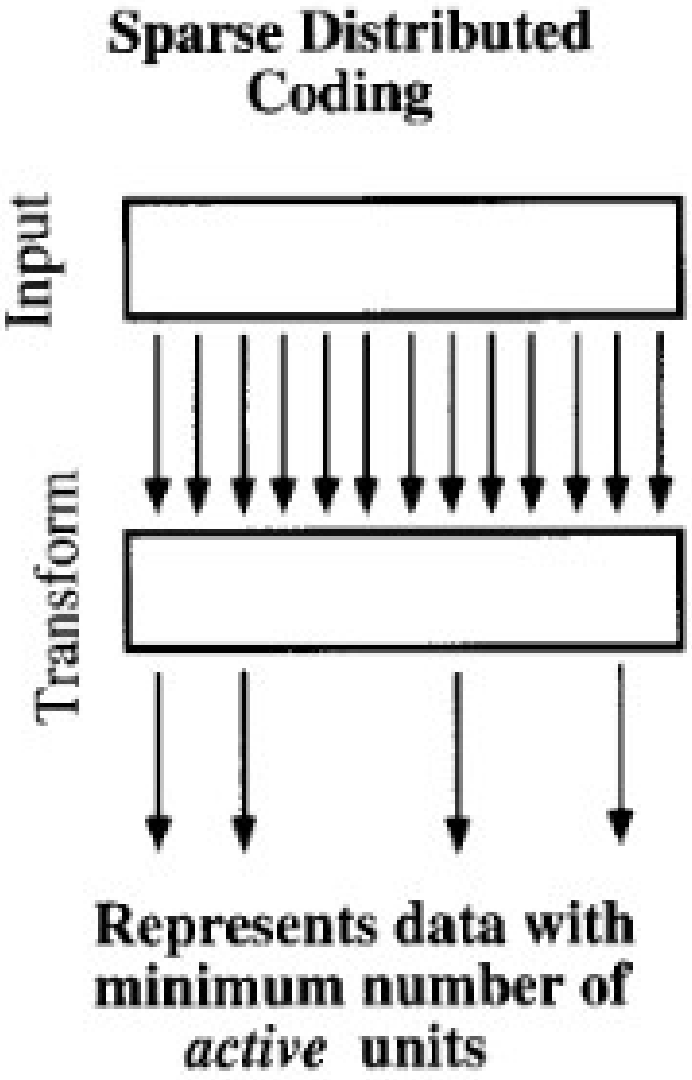

Figure 2.8: Sparse Distributed Coding [6]

Then, in the next Figure 2.9 it is possible to see what a sparse representation looks like in real life.

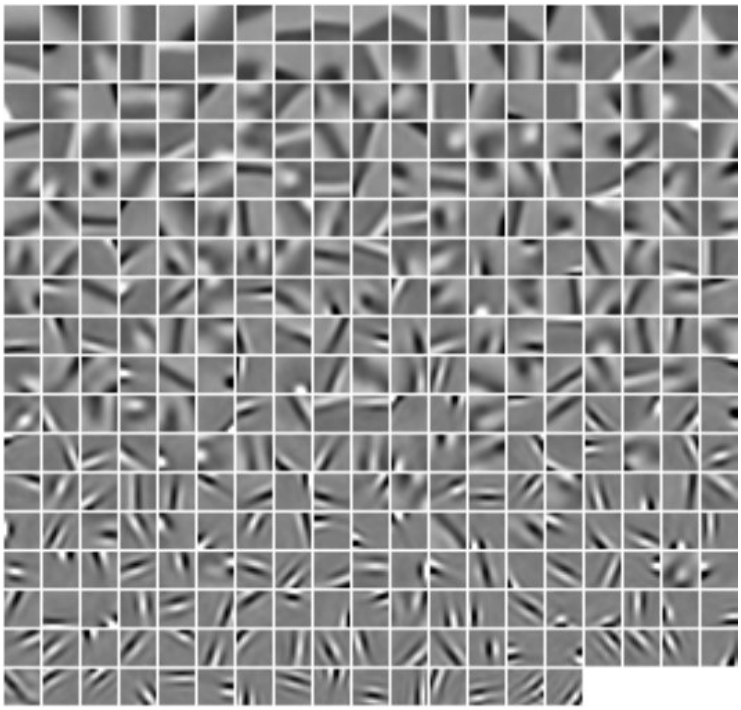

Figure 2.9: Sparsely encoding basis functions learned from natural images.

In its basic form, sparse coding yields a representation known as a grandmother cell code. Every material thing can be reduced to a single cell in this code. One may say that a huge brain with billions of neurons can probably manage a few hundred thousand at the object level. Further, numerous studies have demonstrated the existence of neurons that are exceptionally selective to faces, and other objects [30].

Local codes, in which each item is represented by a single neuron or a small network of neurons, fall on the low end of the average activity fraction spectrum. This code ensures that no neuron is involved in representing more than one object, preventing any potential for confusion between the representations of different items. Similarly, each key on a standard computer keyboard (one lacking the Shift and Control buttons) represents a single character in a code [31]. It is important to remember that the "locality of coding" indicates that the neurons are highly selective, with each neuron encoding only a single, critically important feature of the environment. The simplicity and readability of this design are two of its main selling points. Hebbian strengthening of synaptic connections between their neural representations in a neural network allows for the formation of associations between a locally stored item and any output with only a single trial. With local encoding, your memories will not accidentally mix. Multiple locally coded things can often be represented

unambiguously by the sum of the activities of the relevant neurons [30].

Dense coding is present on the other end of the spectrum. Here, the neurons' actions reflect a single bit of information. This means that N binary neurons can store 2N bits of information. Given that the human brain has billions of neurons, 2N is ridiculously large. Since, there are many more neurons than receptor cells in the brain (or even in a single cortical area, such as the primary visual cortex), sensory processing is hampered. The sparse codes (low average activity ratio) represent a happy medium between the two extremes of dense and local codes [31].

Those who argue for the value of sparse representations are not arguing that the end aim should be one neuron per object or even each view of each object. However, being overly detailed or ambiguous can affect learning [32]. We argue that even in the earliest phases of sensory processing, sparseness aids learning and prediction.

2.10 Reinforcement Learning.

Reinforcement learning (RL) is a machine learning technique inspired by behavioral psychology that sets the parameters of an artificial neural network where no data is normally provided but is generated through interaction with the environment [33]. It has been used to solve various problems, including robot control, telecommunications, and games like chess and other sequential decision-making activities [33]. Furthermore, the neural network reinforcement learning approach allows resolving difficult temporal (time-dependent) problems [34].

RL is the problem of a learning agent interacting with its environment to achieve a goal [35]. It primarily combines two tasks; the first is the discovery of new situations because the agent receives no examples or instructions of the desired behavior; this is accomplished through trial and error. The second step is to apply that knowledge to make the best decisions and reap the greatest rewards. In other words, the agent must exploit what it already knows to be rewarded, but it must also explore to make better action decisions in the future [24].

Aside from the agent and the environment, four major components of an empowering learning system can be identified: the policy, the reward signal, the value function, and,

optionally, the model [24].

- The *policy* of a reinforcement learning agent is at its heart because it defines how to behave at any given time. Policies are typically simple functions or lookup tables, whereas others, such as lookup processes, may involve complex calculations [24].
- The *reward signal* in a reinforcement learning problem, signal, defines the good and bad events for the agent. As a result, in the long run, the agent's sole goal is to maximize the total reward [24].
- The *value function* determines how the agent chooses which actions to perform.
- The *model* of the environment, which is used in some reinforcement learning systems, is something that simulates the behavior of the environment and allows inferences about how the environment will behave. Given a state and an action, for example, the model could predict the next, resulting state and the next reward [35].

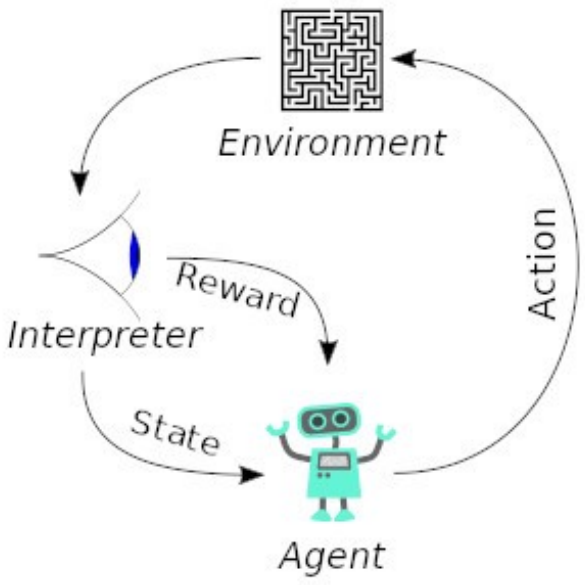

Figure 2.10: Reinforcement Learning [7].

2.10.1 Exploration

During the exploration phase, the agent must test actions to gather information and, as a result, make better action selections in the future to obtain the highest reward.

2.10.2 Exploitation

During the exploitation phase, the agent selects actions that it has previously performed and learned, and that maximize its accumulated reward, where it has proven to be more efficient.

2.11 Deep Reinforcement Learning.

Deep Reinforcement Learning (DRL) is one of the most rapidly growing areas of industry and research. It is a step toward developing self-contained systems with a deep understanding of the visual world. DRL combines deep networking and reinforcement learning to solve previously unsolved problems, such as learning to play video games directly from pixels [36].

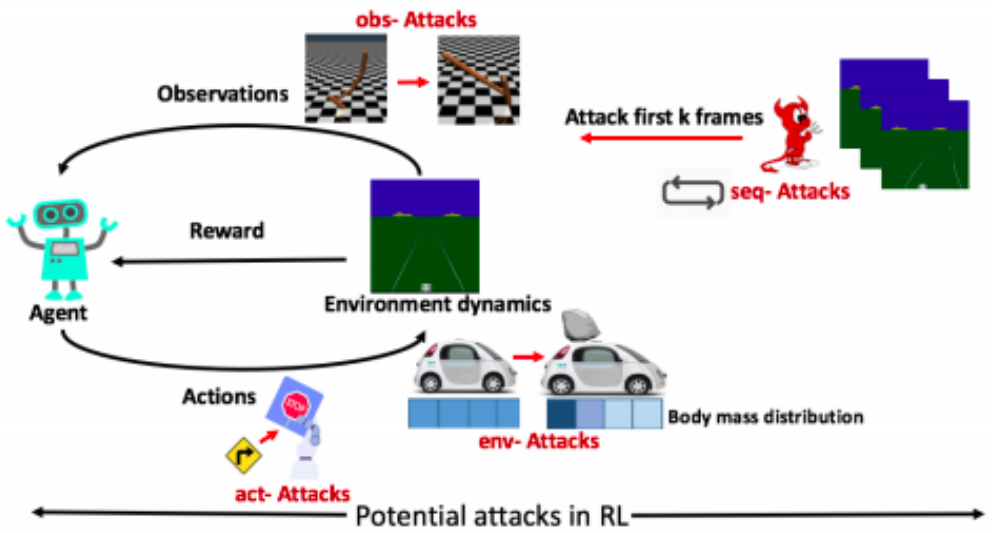

Figure 2.11: Deep Reinforcement Learning [8]

One of DRL's goals is to create systems that can learn to adapt to their surroundings. As a result, several previous studies and DRL studies relied on extending previous RL studies to higher dimensional issues. This DRL includes learning low-dimensional feature representations and neural networks' powerful feature approximation properties. For example, convolutional neural networks (CNNs) can be used as components of RL agents to learn directly from visual data. To approximate the optimal policy and optimal value

function, DRL generally relies on learning deep neural networks [36].

Chapter 3

State of the Art

This chapter were reviewed deep reinforcement learning works, including some methods for creating a neural agent that learns how to fold a specific protein.

3.1 Chang, Zhinin-Vera & Quinga

This work by Chang, Zhinin-Vera, and Quinga [37] proposes a Tic-Tac-Toe learning environment based on a self-motivated neural agent that learns the game scenarios before using the information in actual competitions, where it imitates a Markov model. The authors' work attempted to create self-taught agents that adopt a future view of the game, or an "I already won" or IAW+4 game vision, which is a brilliantly foreseen sequence of movements. Nine sigmoidal neurons connected in a chain that operates in real-time and inhibits one another with modest shared negative weights make up their implementation of the model of the self-motivated neural agent. A repetitive ramp K excites all neurons in the same way. On each repetition, the agent burns "dark energy," fires, and declares a single winner, which is used to select a tile from the board. Even if it is disconnected from the (external) advisory neurons, the agent will continue to make the right moves.

In addition, they initially took the state of each tile for the learning networks, represented by the three neural signals 010, 001, and 100, which stand for filled and empty tiles, "O" and "X," respectively. The resulting 27 signals are routed into a network of 27 inputs, 67 hidden, and 9 output neurons. They serve as a sparse encoding representation of the board's state. Many of these subnets are used and educated as indexable advisers that suggest prudent actions to the agent. The agent learns game patterns that guarantee

winning scenarios by exploring potential plays based on the Bellman equation, which uses the equation's first three terms. After training, the most intriguing output neuron will show what should be filled, or, in other words, what move should be made, as the optimal policy indicated by the network.

Finally, during the operation phase, the neural agent recognizes and executes IAW+4 game circumstances with high security after receiving guidance from the trained networks. The authors have thus successfully demonstrated how the self-motivated neural network could be utilized as a free-running random agent that investigates all potential game scenarios and is trained with backpropagation to memorize excellent game sequences using advisory indexable subnets. Additionally, the reinforcement learning technique supports a successful future quest for maximal rewards. This thesis suggests that the work be improved by incorporating deep reinforcement learning into the model.

3.2 Chang & Zhinin-Vera

Chang and Zhinin-Vera [38] described a creative robot that could independently pick up complex tic-tac-toe game rules and use them to compete favorably against people. A robotic arm, a machine vision system, and a self-driven neural agent were used to construct the robot. The self-taught neural agent has made logical conclusions that the robot has been programmed to carry out physically.

The Bellman equation with three terms has been presented to search for future rewards, and the agent is built on the architecture used in the related work of section 3.1. The robot's mechanical design incorporates three axes, a power supply, an Arduino board, connections from the computer to the servo motor, and shoulder, elbow, and finger servo motors. Artificial vision has been implemented using a camera that records color images of the real world. OpenCV is used to process these images before sending them to a neural network trained to recognize Xs, Os, and blank spaces.

As a result, the authors created a robot that observes the game board and moves using its robotic arm. These actions are made possible by the neural agent, who has learned to play tic-tac-toe in an unusual manner, and the artificial vision system, which looks at the board in the real world. The robot created can be used to test the deep reinforcement

learning software that will be utilized in this thesis. Hence this work has been considered as it relates to 3.1.

3.3 Senior, Evans, Jumper, Kirkpatrick, Sifre, et al.

Senior et al [39]. demonstrate how a neural network can be trained to make accurate predictions of distances between pairs of residues, which provide more structural information than contact predictions. Furthermore, the authors used this information to optimize the resulting mean force potential using a simple gradient descent algorithm, which can generate structures and accurately describe the shape of a protein without the use of complex sampling procedures.

The system that emerged from this research, AlphaFold, was created by Google and can make extremely accurate predictions about the folding of unknown proteins. Its main component is a convolutional neural network that has been trained using a database of protein structures, even in cases when there are fewer homologous sequences. The approach's foundation is the notion that by examining covariation in homologous sequences, we can learn about the evolution of the genome. It is possible to determine which amino acid residues are in contact, making it easier to predict protein structures.

Finally, the authors concluded that the AlphaFold results show that a neural network can be trained to quickly and accurately estimate the lengths between pairs of residues. In addition, the system is optimized using the gradient descent algorithm and represents a significant advancement in protein structure prediction. Compared to the competition, AlphaFold's score of 68.3 was significantly higher.

3.4 Kuo-Chen Chou

The review's primary emphasis was on developing methods for anticipating and classifying tight bends. Chou [40] focused mostly on recent developments in this field, grounded in the sequence-coupled model developed from Markov chain theory.

The smallest tight turn is a $\delta - turn$. Variously referred to as the C_8 form, the $1 \rightarrow 2$ type, and the $2 \rightarrow 3$ type, this structure has only two amino acid residues. A $\gamma - turn$,

involving only three amino acid residues, is the next-smallest tight turn. The backbone's CO(i+1) and NH(i) establish the intraturn hydrogen bond in a $\gamma - turn$ (i+2).

They discovered that the sequence of a protein with a tight-turn shape might be expressed typically in terms of oligopeptides (groups of two to six consecutive residues) using the formula:

$$R_i R_{i+1} R_{i+2} \cdots R_{i+n} \tag{3.1}$$

Where $n = 1$ for a $\delta - turn$, $n = 2$ for a $\gamma - turn$, $n = 3$ for a $\beta - turn$, $n = 4$ for an $\alpha - turn$, and $n = 5$ for a $\pi - turn$; R_i represents the amino acid at the protein sequence position i, R_{i+1} represents the amino acid at the protein sequence position $i + 1$, and so forth.

So they proposed to focus on the correlation effect between the first and fourth residues along a tetrapeptide to forecast $\beta - turns$; the same effect was claimed for predicting $\beta - turns$ between the second and third residues. The most common sequence of residue types for $\beta - turns$ was found to be hydrophobic, hydrophilic, hydrophobic, and hydrophobic.

The authors conclude that the sequence-coupled algorithm they formulate in their review is a universal one that may be used to anticipate not only the $\beta - turns$, $\alpha - turns$, and their kinds as shown here but also additional tight turns and their types.

In reality, users can forecast any tight turns they care about using their training dataset.

3.5 Chang, Gonzales, Zhinin, Valencia, Pineda & Diaz

Chang et al [9]. Research details a robotic construction mimics the Hemagglutinin Esterase's internal structure, functioning as a stable peptide chain. Where a self-guided agent is created which can anticipate its own needs and learn on its own is the brain behind this robotic framework.

Robotics, ANNs, PD, and RL all come together in this approach for effective machine learning. Using a robot's underlying structure, they showed that a self-taught agent is capable of learning a protein-folding procedure that is both efficient and effective. To start the training, they used a protein with a straight position, and then they used the neural agent to generate new positions and find rewards.

The maximum duration of the agent's training episodes is 4000 iterations. Each episode sees the agent place the peptide chain in a known initial condition before venturing into uncharted territory by generating numerous angles and folding variations in the surrounding area.

Finally, they show the extensive results acquired, demonstrating that their agent operating on this robot-like structure can learn to solve protein folding challenges. It was also discovered that folding knowledge could be retained in stable weights parameters.

Chapter 4

Methodology

This chapter describes the entire process used to achieve the project's goals. It describes the implementation of a neural agent that learns policies for accurately folding proteins.

4.1 Phases of Problem Solving

1. To expand and debug the code used in previously developed work to improve the virtual robot learning capacities. This required the development of new operating routines [9].

2. To create sparse code (more 0's that 1's) to represent the 17 individual angles existing between each pair of amino acids. This generate a 16x17 neurons matrix, excited with sparse code patterns and used as the first input information to the folding agent.

3. To enlarge the size of the cysteine sensors, simulating the iterations of the amino acids that exist between them and improving the sensibility of the model.

4. To implement the deep neural network architecture by defining the number of convolutional layers, filters, top group layer, inputs (input layer), the number of hidden neurons (hidden layer), and the number of output neurons (output layer).

5. To implement the architecture's graphical model to visualize the agent's behavior during the deep neural network training process.

6. To use deep network training during the exploitation stage to analyze the behavior of the agent, its movements, its convergence to a stable and efficient solution, and its

ability to generate movements in anticipation of protein folding.

4.1.1 Description of the Problem

Significant advancements in deep reinforcement learning have enabled extracting high-level features from raw pixels sensory data, resulting in significant advances in artificial intelligence [41]. These methods employ various neural network models, such as convolutional, perceptron, multilayer, and recurrent neural networks. These methods have already found use in various problems, including robotics, where control policies for robots can now be learned from the neural network directly from either complete or incomplete (pixel) image inputs [36]. This is the challenge this work presents: a neural agent must learn protein folding policies from a raw pixel image to fold the protein. The first problem is to create a sparse matrix from the sparse code angles and amino acids. The second problem is that complex forces determine how the cysteines will interact with each other; another issue is to increase the number of sensors to the amino acids to improve the bending capacities for the model. The third problem is to develop a reinforcement learning agent that can learn an optimal protein folding policy using the information expressed in raw pixel images and a fully connected network. For this policy to function at its best, it must recognize protein folding characteristics and apply this knowledge in its decision-making process, beginning with the agent's initial bends and ending with the cysteine joining. Finally, the agent's ability to fold proteins will be evaluated by previous work [9]. The main goal of reinforcement learning is to link current and future states.

4.1.2 Analysis of the Problem

To solve the previous problems, we have a robot with bending joints representing the union between amino acids; each joint is controlled by a "muscle" and one neuron, which is the output of a deep neural network. This network will train with reinforcement learning so that the robot bends following some properties of proteins.

The input to the Deep Neuronal Network is two raw pixels images representing the angles between sparse code and amino acids codified in sparse code. Following a simplified model of the HEs protein, the immediate reward is that the two cysteines in the chain

become closer.

The maximal reward is given when the cysteines touch each other.

4.1.3 Reinforcement Learning Algorithm

This work uses a learning algorithm in which the estate of the robot is defined by the angle matrix and the sensor matrix. From here, the robot does a future exploration movement by using a set of competing neurons which defines how much each angle in the robot is going to chain; if the movement brings the cysteines closer to each other, the agent receives a reward, and a backpropagation cycle is done with the targets announced by the racing neurons. This cycle is executed several times until the agent learns to bend the chain until the cysteine touches each other. If not, a new exploring sequence is carried out.

The pseudo-code is shown in Algorithm 1.

Algorithm 1: Reinforcement Learning in Proteins Folding

```
1 Protein start at straight position state s(n) ;
2 fire competing neurons ;
3 check the arrival order ;
4 set the angles variation according to the arrival order s(n+1) ;
5 if cysteine distance decrease then
6 |   - Do backpropagation using previous state as input s(n) and angle
  |     variation+output as targets ;
7 |   - Repeat until cysteines touch each other ;
8 end
9 Repeat until trained ;
```

This section describes the algorithm that learns to fold proteins. The protein is initially displayed in a slightly stretched position. The neural agent expends energy to generate new positions and investigate future rewards. When a reward is discovered, the agent memorizes the paths leading to even greater rewards through reinforcement learning (good protein folding).

The agent operates in episodes that last no more than 4000 training cycles. During each episode, the agent learns by initially positioning the peptide chain in one state and

then exploring new territory by producing several angles and folding variations in the neighborhood. When the cysteine-cysteine bond is formed, the reward should be maximal. In other words, the agent will learn the folding when the cysteine-cysteine bond is formed (the stronger attraction between the given amino acids).

The maximum reward is obtained when sensors detect a close encounter between cysteines during cysteine bonding. Gradient descent takes the previous folding state and sensors as input and the current delta folding vector as a target.

For each training cycle, the agent read the internal information and sensors information, get decision (moves to a new state folding), if cysteines distances decrease, execute decision and captures the MAX future reward. Once this is determined, a reward discount in backpropagation cycles is applied. [38].

Algorithm 2 depicts the pseudo-code.

Algorithm 2: Folding Proteins with Self-taught Neural Agents

```
1  Learning Phase;
2  while cysteines do not come closer do
3      - Agent explore the future;
4      - Random moves;
5      if cysteines come closer then
6          - Agent memorizes the future;
7          - Do backpropagation;
8      end
9  end
10 Operative Phase;
11 while episode do
12     - Read the internal information and sensors ;
13     - Get decision;
14     - Execute decision until cysteines touch each other;
15 end
```

Algorithm 3 shows the pseudo-code of how the sparse code has been implemented to find 180 16-bit binary numbers. Sparse coding may be a general strategy of neural systems

to augment memory capacity [42].

The first step of the sparse code is to search for 16-bit binary numbers randomly, containing more zeros than ones; a number must have at least 9 bits of zeros to be considered a sparse number. The next step is to check that the number obtained is not repeated in the 180 numbers; if this is the case, only the repeated number returns to look for another random number that meets the abovementioned conditions. So, in the end, it finds the 180 numbers necessary to obtain a sparse matrix with bending angles and amino acids later.

Algorithm 3: Sparse code

```
for 180 16-bit binary numbers do
    - Search random 16-bit numbers whit more zeros than ones ;
    if 16-bit binary number is repeated then
        - search another one ;
    end
    else
        - save 16-bit binary number;
    end
end
```

4.1.4 Testing

To test the agent, we first check if the robot does good protein folding based on the previous study [9].

In the Figure 4.1, we can see the robot before and after bending.

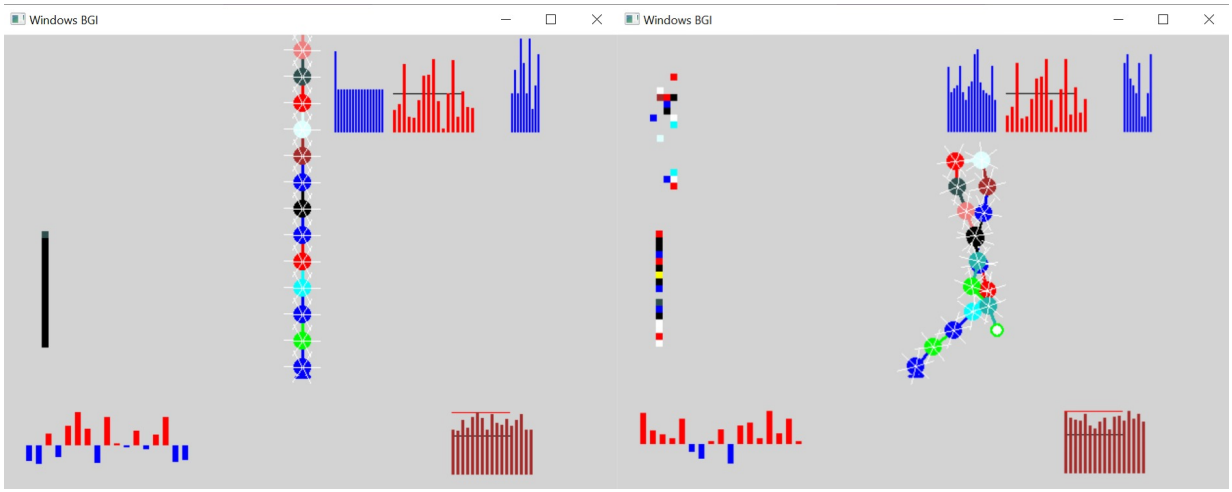

Figure 4.1: First test before folding/after folding. the left image shows an initial representation of the 17 amino acid peptide chain. The image on the right represents the folding of the complete amino acid peptide chain that forms the protein (the target of the training).

Once protein folding was verified, the second important test was to implement a sparse matrix that represent the 17 angles between the amino acids. This raw pixel matrix is given as input to the learning agent, an a typical bending image is shown in Figure 4.2.

Figure 4.2: Sparse pixel matrix of 16x17bits representing the current angles between the 17 amino acids.

After the robot has learned to do the bending, we obtain the results shown in the Figure 4.3:

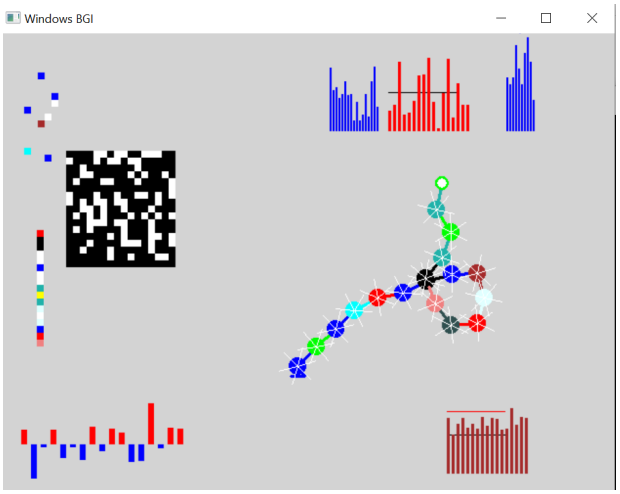

Figure 4.3: Result of protein folding including algorithm 3. Sparse Code, to generate the sparse matrix in the folding of the amino acid chain.

The next test we performed was to length the cysteine sensors because, in our model, the dominant force is considered to be the covalent bond that exists between cysteine and cysteine, due to the tight turn effect of the amino acids located between them (proline and glycine). See Figure 4.4.

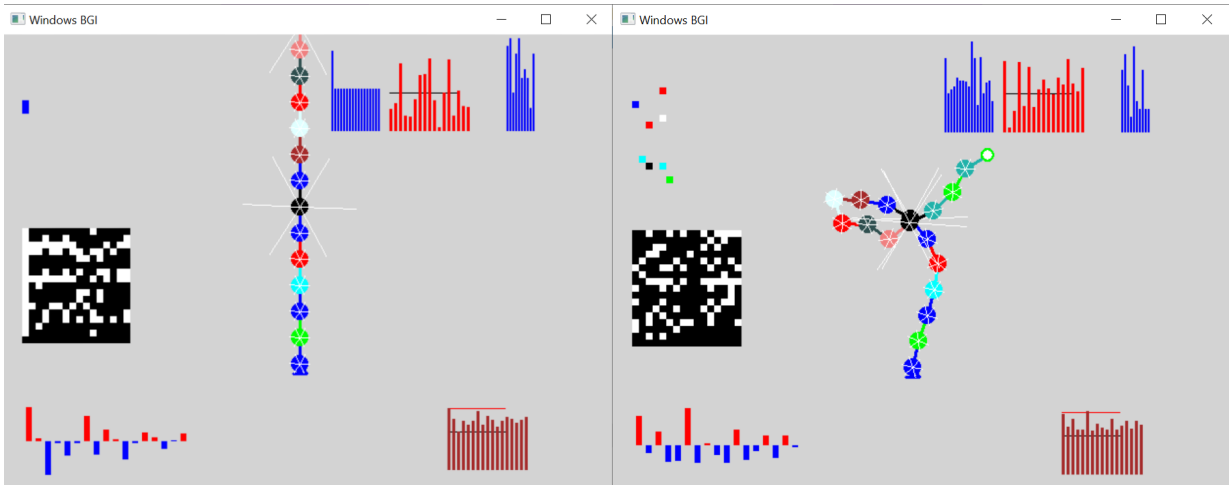

Figure 4.4: Same as Figure 4.1, but this time using the sparse matrix approach for protein folding. In addition, in this test we increase the size of the sensors to detect more information about near molecular forces.

Another test that we carry out is the increase of the sensors to all the amino acids. As shown in Figure 4.5

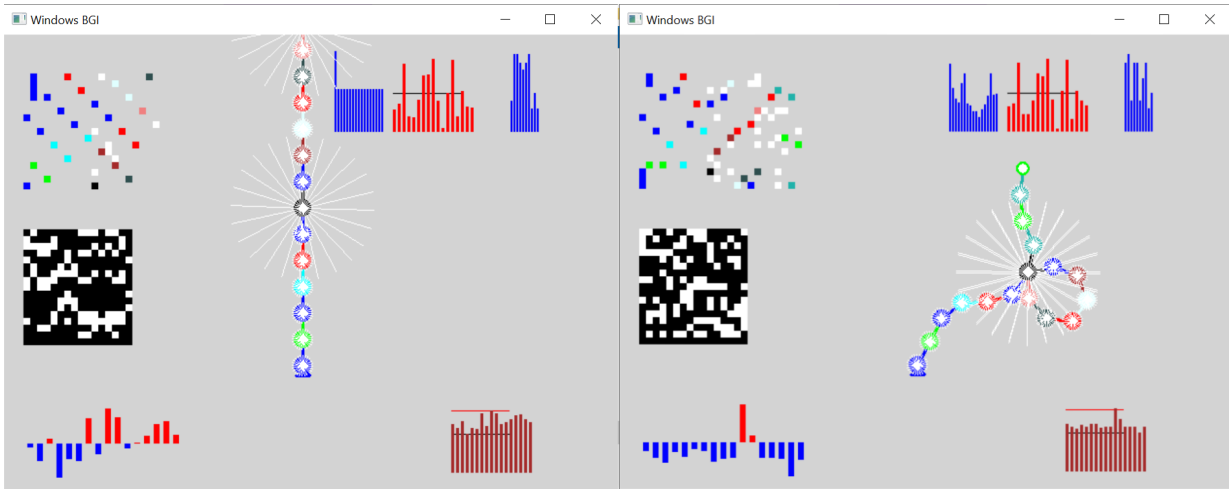

Figure 4.5: On the left side the initial state is shown where the white lines represent the increase of the 24 sensors, unlike the previous tests that only consisted of 6 sensors. On the right side of the image the folded protein is shown as the final result of the training.

Finally, after completing these tests, we add the reinforcement learning training to the input images to the network that we obtained in the Figure 4.6

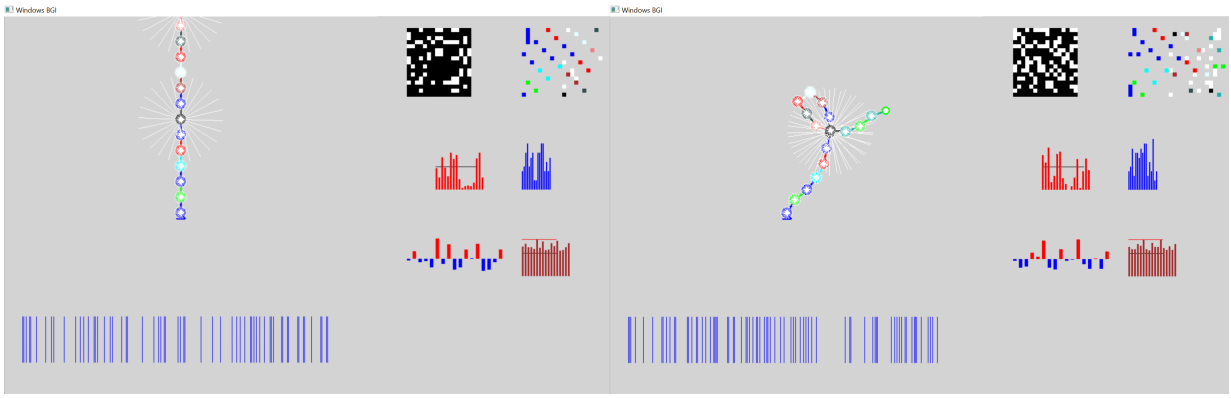

Figure 4.6: As shown in the previous figures, the amino acid chain begins with its initial state and the blue lines below this chain represent the input of the network architecture used. On the right side of the image we can see the objective of applying Deep Reinforcement Learning for successful protein folding.

4.2 Model Proposal

We proposed this model using sparse coding. From knowledge of biology bending theory, we know that there is a tendency between the amino acids proline and glycine to create a tight turn between them, which brings the cysteines to come close together. For this reason, in our proposed model, we assume that the cysteines had longer sensors, enhancing

the probability that they sense each other and come into touch. With this assumption, we prove our main objective: to check if the reinforcement learning method works properly in protein folding problems.

4.2.1 Sparse code

Sparse coding may be a general strategy of neural networks to increase memory capacity [43]. To adapt to its environment, the agent must learn which stimuli are associated with rewards or punishments and distinguish these reinforced stimuli from similar but irrelevant ones[44].

In this model, we develop the sparse code by generating non-repetitive binary numbers with sixteen digits each, where the number of 0s is always bigger than that of 1s. (more 0's than 1's) (see Figure 4.7). These sixteen digits will represent the unique angles between each pair of amino acids and form a sparse matrix. In other words, these sixteen digits will represent the information about the angle at which each amino acid will bend. Thus we have 16 analog variables converted to sparse code, which are the folding angles of the 17 amino acids, and we generate a 16x17 neuron matrix excited with sparse code patterns.

We will use this information obtained from the sparse matrix as the first input information to the folding agent (see figure 4.2). This conversion of the 17 variables to a sparse matrix is similar to a simulated deep network.

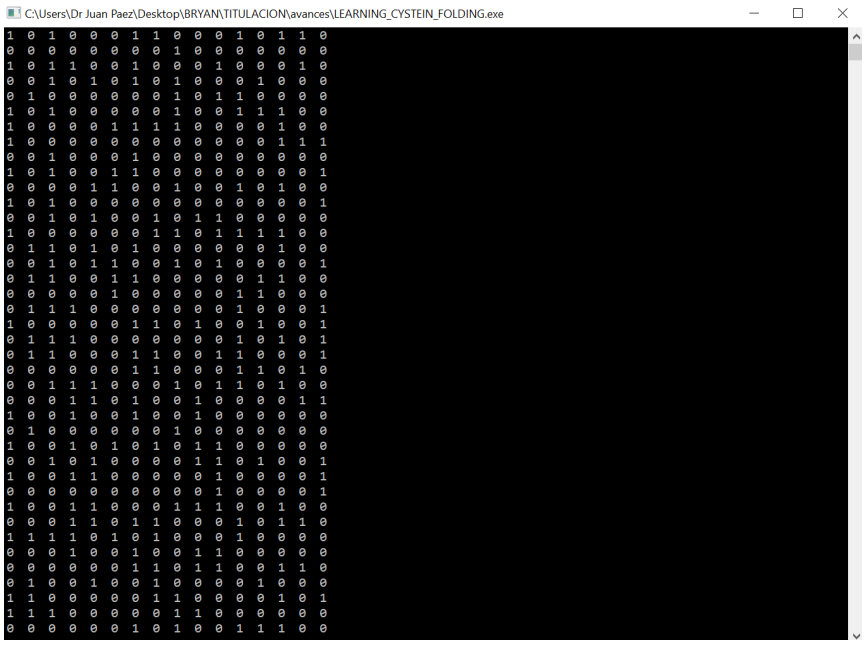

Figure 4.7: Sparse code generated with more 0's than 1's

4.2.2 Protein Folding

To test the virtual robot/ability agent to learn efficient folding policies, a peptidic region from the protein HCoV - HKU1, Hemagglutinin-Esterase (HEs), was chosen. (See Figure 4.8A.) HEs are made up of two identical polypeptide chains, each with three important domains: the proximal membrane domain (MPD), the esterase domain (E), and the receptor domain (R) [45].

The first step in a protein's folding process to adopt its native structure is forming a secondary structure. According to RCSB Protein Data Bank (www.rcsb.org), the receptor domain comprises eleven secondary structures (one α-helix and ten β-strands) based on the recently solved structure of HEs ([46], [47]). These two β-strands, $_{182}$LYLVPLCL$_{189}$ and $_{218}$DCIYI$_{222}$, form an anti-parallel β-pleated sheet structure supported by a disulfide linkage and hydrogen bonds between the amino acids in both sequences (Figure 4.8B).

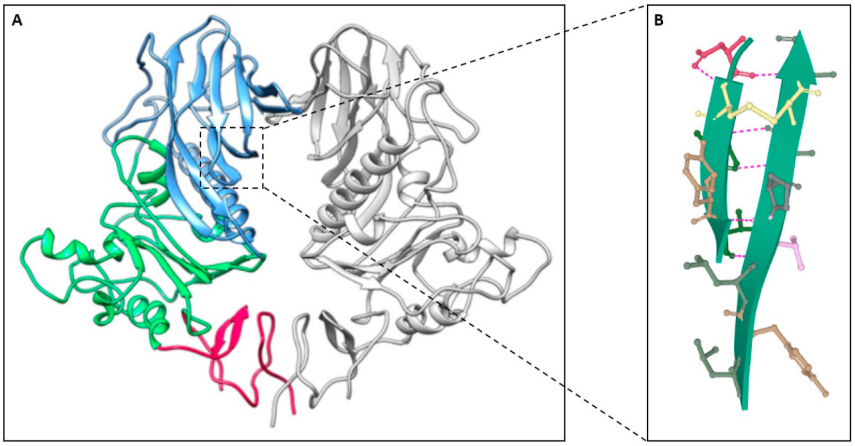

Figure 4.8: **A.** Ribbon representation of the dimeric Hemagglutinin-Esterase and domains: membrane-proximal domain (red), esterase domain (green), and receptor domain (blue); the dotted square represents the beta-pleated used in this study. **B.** Hemagglutinin-Esterase receptor domain anti-parallel beta-pleated sheet. Cysteines and amino acids are covalently linked in yellow, and dotted lines represent hydrogen bonds between amino acids [9].

The sequence from residue 190 to residue 217 was replaced by a β-turn structure containing the amino acids GSPN to study the folding of this structure. The final peptide to test the proposed agent's self-teaching capacity is then: $_1$LYLVPLCLGSPNDCIYI$_{17}$

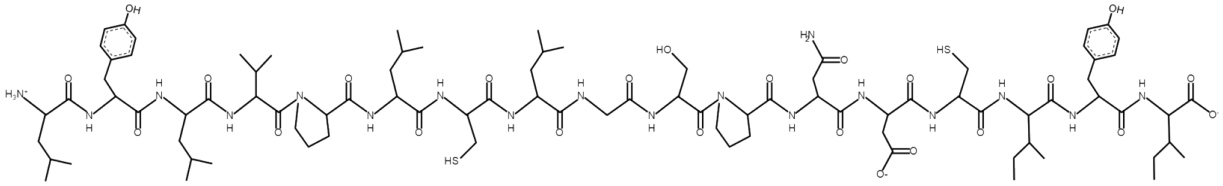

Figure 4.9: Chemical representation of the final peptide to test the self-taught capacity of the proposed agent.

4.2.3 The Protein Folding Robot

A robot is programmed to perform the space-time physical actions of folding a peptide chain of 17 amino acids from the HEs protein's receptor domain, specifically the sequence:

$$_1\text{LYLVPLCLGSPNDCIYI}_{17}$$

Amino acids are linked together by a strong peptide bond, and each bond has an actuator (muscle) that controls the folding angle (angle[i]) between them; this angle is codified in a 16-bit signal (sparse code) that spans the range from -90 to +90 degrees from the center or aligned position. The set of folding angles is a known parameter that defines the robot's internal representation. This internal representation is one of the input raw pixels matrix vectors used by the agent to generate the set of variations or delta[i] required to produce the next folding state using the actual angle plus the delta angle generated in Equation 4.1. Each amino acid has twenty-four sensors that detect the type of amino acid in its neighborhood, close enough to produce short-range molecular effects such as strong bonds or polarity arrangement. Furthermore, each cysteine has increased the size of the sensors to improve the model's sensitivity. Each actuator is controlled by a single sigmoidal neuron in a larger network. Each output neuron generates a signal that represents the increment or decrement that each angle will have in the next cycle. The network is in charge of the strand's overall space-time folding. Figure 4.16 depicts a complete protein-folding robot schema.

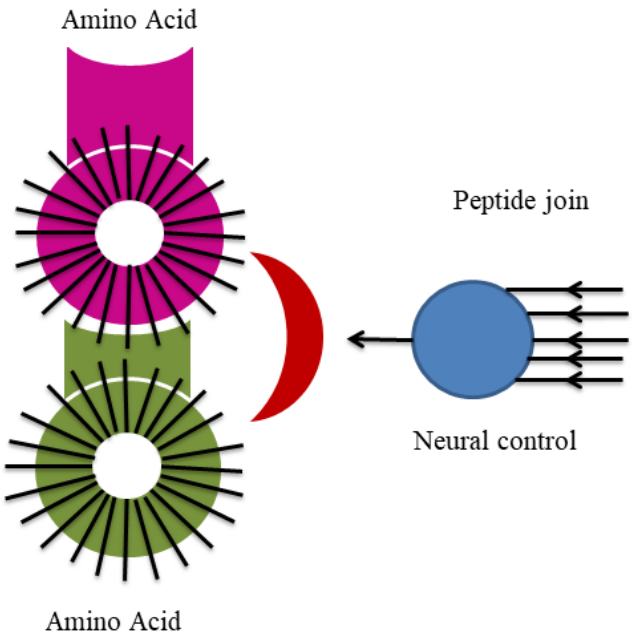

Figure 4.10: A multi-joint robot and its associated neural blocks have to learn to fold a peptide chain belonging to the HEs protein. The mechanical simulator is a chain of operative joints with independent muscles and neural controllability.

4.2.4 Controlling Network

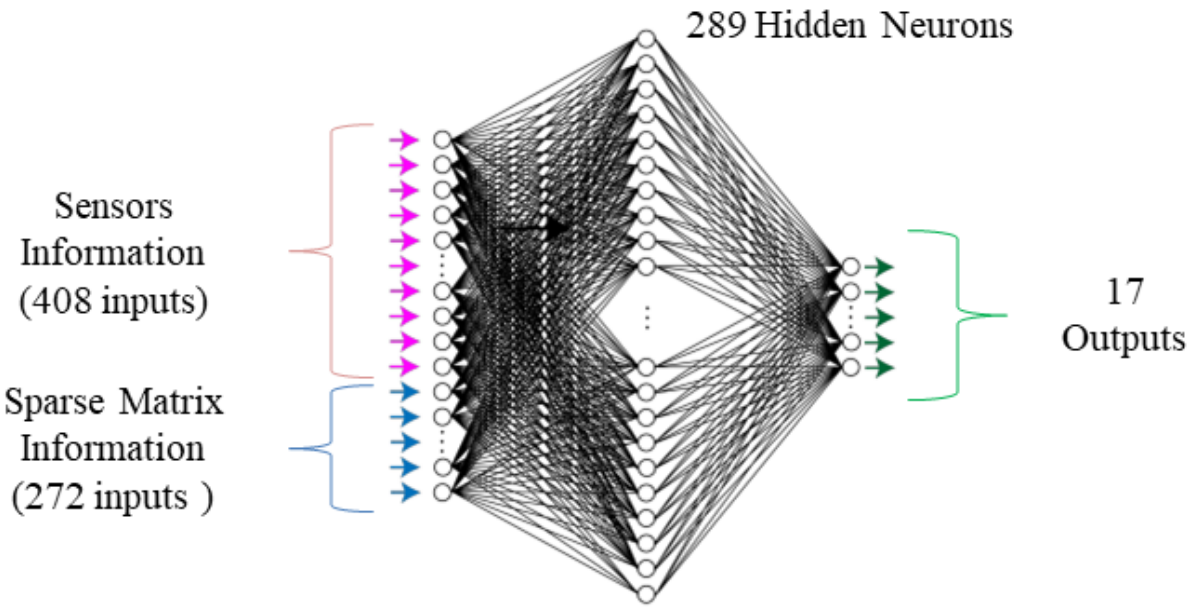

Figure 4.11: Neural Network Architecture. A reinforcement learning trained network

This is the network that is going to be trained for protein folding with the help of an exploring neural agent (exploration), so that it ultimately controls how the robot is going to fold (exploitation). This network starts with random weights (raw pixels) and with this composite ”image” the network has to learn to generate changes (predictions) in the protein folding that improve the probability of obtaining functional structures.

4.2.5 Self-taught Neural Agent

The model proposal was refined: The peptide chain comprises 17 mobile parts, each representing an amino acid and supported by a rigid structure (the molecular bond) and a set of sensors that detect and process close molecular forces. Because the cysteine amino acids lay between tight turn-forming amino acids (proline and glycine), the model assumes that the cysteine sensors will have a longer range than the other amino acids, which will help to achieve a proper folding goal.

The self-taught neural agent comprised 17 sigmoidal neurons, which inhibit each other with balanced negative weights and share a common self-activating excitatory input K-ramp (see Figure 4.12); when K grows in a ramp fashion, forces all participating neurons

to race toward a 0.86 output. Its function is to produce random angle variations that will be used to move the robot and produce the reward signal to train the controlling network.

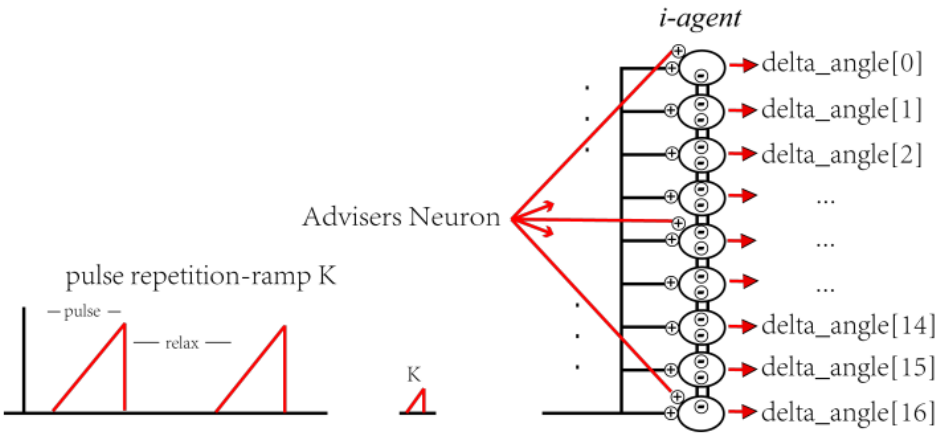

Figure 4.12: Self-taugth Neural Agent. Neurons are all equally excited by a repetitive ramp K.

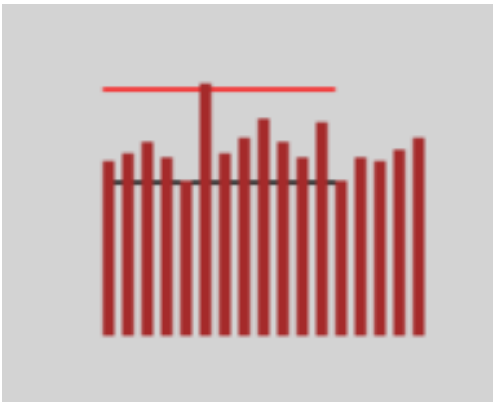

Figure 4.13: Race of each participants neurons.

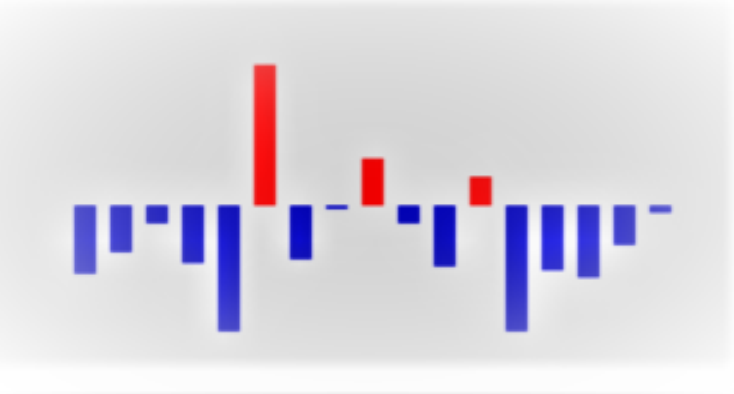

Figure 4.14: Negative (blue) and positive(red) angle variations obtained after a race.

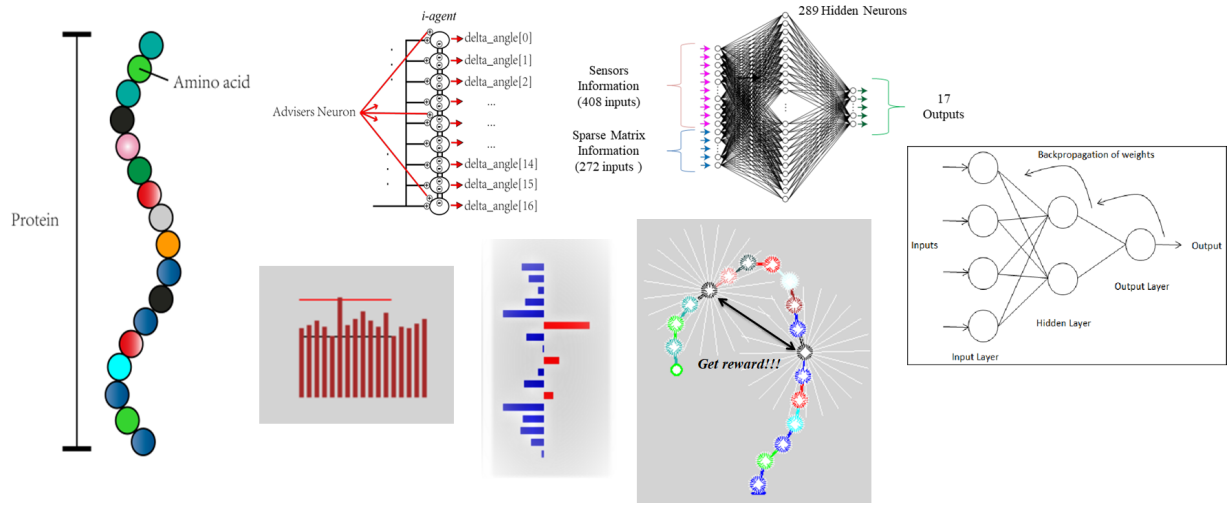

Figure 4.15: Cycle performed by the agent to obtain the values of the new bending angles of the robot.

The structure of the self-taught neural agent is depicted in Figure 4.15. During a ramp, after a certain amount of time, one participant (winner) crosses a threshold (end line) setup at 0.85. The finishing order is frozen, and values are assigned to the agent outputs based on the arrival order of the other neurons.

Taking as reference the 0.7 value, the race output is converted to a vector of bounded positive and negative numbers, which is then used to calculate the angle variation for each amino acid using the formula:

$$\text{delta\_angle}\,[i] = \text{alpha} * \text{out\_agent}\,[i] - 0.7 \tag{4.1}$$

Suppose the new state of the cysteines comes close together. In that case, the vector of a bounded positive and negative value is added to the previous value of the angle (the outputs of the controlling network), and this new vector value is taken as the vector target after that one cycle of backpropagation is performed. Where alpha controls the degree of bending deformation per cycle, the value 0.7 is an offset required to generate positive and negative quantities.

In resume (see Figure 4.15), to check the future and satisfy the Bellman equation, in each race, the agent generates a set of random bending information (increment/decrement) in the form of variations or ”delta” angles. These deltas are added to the current angle values bringing the folding to a new bending state s+1. The system checks if the distance

between cysteines decreases in this new state. If this is, the agent receives a reward, the previous input image (raw pixels) is taken as input, and the previous network vector output is added to the variation vector and taken as vector target. The idea is that the most excited neurons that contribute to the approach of the cysteines are the rewarded while the others are penalized.

4.3 Experimental Setup

Several episodes, each with a maximum of 9000 cycles, are run. The episode begins with the peptide chain in a straight random position. In each cycle, the proximity of cysteine is calculated if this is so, the delta angles are added to all previous angles as a reward and one cycle, backpropagation is performed. This process is repeated until a maximal number of cycles is completed or when the cysteines touch each other, concluding an episode. After training, the controlling network's knowledge is now used to calculate the path of the robot, since now this knowledge represents a valid policy for the bending process (exploitation).

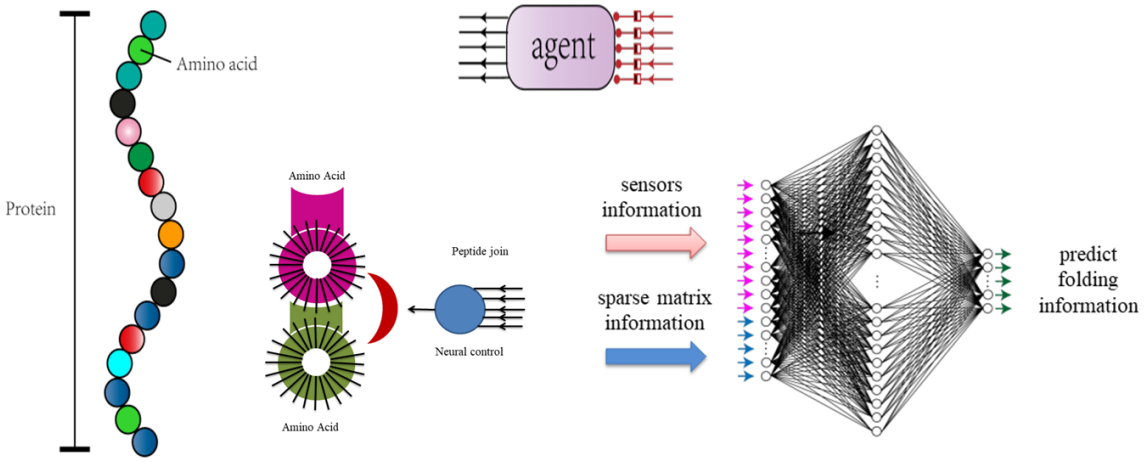

Figure 4.16: The protein folding robot.

Chapter 5

Results and Discussion

This chapter summarizes the experimental and final results obtained by the neural agent we created.

After calibrating, debugging, and testing all the routines, the agent was set to a self-training session until a valid policy was obtained. Due to sparse angles code and the cysteine sensors' enlargement, the obtained reinforcement learning algorithm works satisfactorily. The agent efficiently learns to fold the selected protein section according to the rules established by the amino acids between the cysteines.

With tests and experiments carried out in the exploration phase of the agent and after 9000 training cycles, all the possible movements that the agent can take to meet its training objective were obtained. So that later, in the exploitation phase, it finds that the network becomes trained. The found policies allow the robot to bend from a starting correctly (almost stretched random). Position to a folding state that satisfies the cysteines approaching, according to the intermediate amino acids placed between them. In other words, the agent learns a protein folding policy in the exploitation phase, takes control of the robot, and correctly folds it, using raw pixels as input.

We can see in Figure 5.1 depicts the program's performance, showing how the agent's error decreases as it becomes a stable protein structure guided by rewards.

The measured error corresponds to the difference between the targets and the output neurons, as calculated using the following formula:

$$\textbf{GlobalError} = \sum_{i=1}^{i<=\text{n out}} \left[T\left(i\right) - out\left(i\right)\right]^2 \tag{5.1}$$

Where ***n_out*** represents the number of output neurons.

T(i) represents the targets.

out(i) represents the output neurons.

Using the formula mentioned earlier, we obtained an error of 0.86 in the first training cycle, and after 9000 cycles, the error was reduced to 0.0037. As illustrated in Figure 5.1.

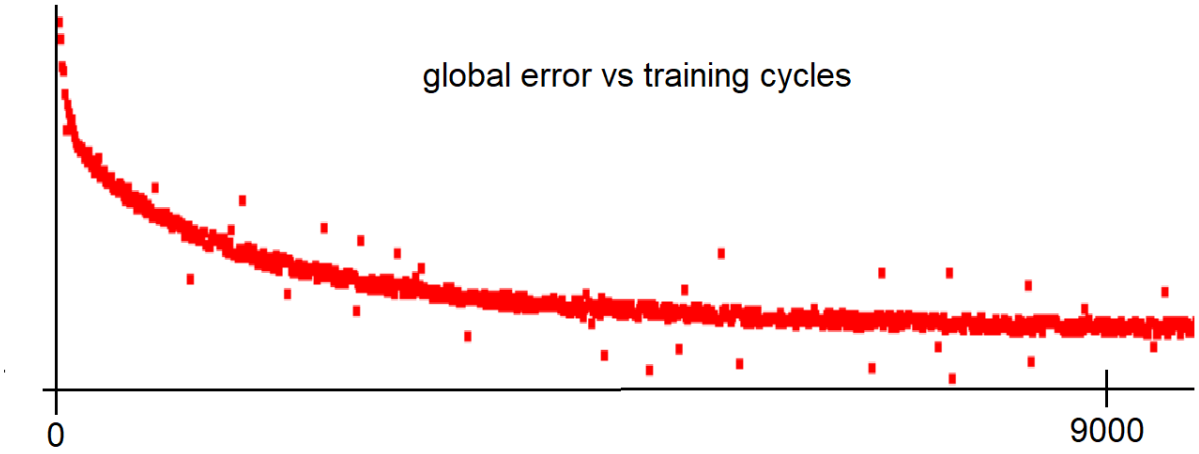

Figure 5.1: Global Error vs Training Cycles. Self-taught agent error decreases as the number of neural network iterations increases.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

1. This research created a virtual robot model that solves specific protein folding problems using raw pixels as input and Deep Reinforcement Learning as a training method.

2. An algorithm that converts information about the angles between amino acids to sparse code has been developed as a first contribution. This algorithm simulates the effect of a deep convolutional network, creating the ambient required for deep reinforcement learning.

3. Reinforcement learning is carried out with an agent that explores and captures rewards in the future, thus satisfying the bellman equation and its exploration requirements.

4. Compared with previous work [9], the enlargement of the cysteine sensors produces a richer sparse matrix, which contributes to the efficient learning of bending policies.

5. Finally, the main objective of determining the correct way to fold proteins was to test the prediction method by Deep Reinforcement Learning. In order to facilitate the folding process in any existing piece of protein efficiently and quickly.

6.2 Future Work

In theory, it would be possible in the future to create more complex models. The new models with more sensors and amino acids. Produce bindings of great computational complexity by using a supercomputer or a CUDA-GPU ambient. And, making the creation of synthetic proteins and medicines for viral diseases possible.

In addition, we can modify a new neural network architecture to the existing network, adding more layers in the prediction stage. Based on theory, the more layers a network has, the can often process information more deeply.

Bibliography

[1] D. Sparkman, Z. Penton, and F. Kitson, *Gas Chromatography and Mass Spectrometry: A Practical Guide*, 2nd ed. Academic Press, 2011.

[2] S. Dutta, G. Pregartner, F. G. Rücker, E. Heitzer, A. Zebisch, L. Bullinger, A. Berghold, K. Döhner, and H. Sill, "Functional classification of tp53 mutations in acute myeloid leukemia," *Cancers*, vol. 12, 2020.

[3] M. Greenwood, J. Antonio, and D. Kalman, *Nutritional supplements in sports and exercise*. Humana Press, 2008.

[4] M. Heim, L. Römer, and T. Scheibel, "Hierarchical structures made of proteins. the complex architecture of spider webs and their constituent silk proteins," *Chemical Society Reviews*, vol. 39, 2010.

[5] S. Cheriyedath, "Protein Folding," 2 2019. [Online]. Available: https://www.news-medical.net/life-sciences/Protein-Folding.aspx

[6] D. J. Field, "What Is the Goal of Sensory Coding?" *Neural Computation*, vol. 6, no. 4, pp. 559–601, 07 1994. [Online]. Available: https://doi.org/10.1162/neco.1994.6.4.559

[7] D. Mwiti, "10 Real-Life Applications of Reinforcement Learning," 7 2022. [Online]. Available: https://neptune.ai/blog/reinforcement-learning-applications

[8] ODSC - Open Data Science, "Best Deep Reinforcement Learning Research of 2019 So Far," 12 2021. [Online]. Available: https://odsc.medium.com/best-deep-reinforcement-learning-research-of-2019-so-far-e8e83a08c449

[9] O. Chang, F. A. Gonzales-Zubiate, L. Zhinin-Vera, R. Valencia-Ramos, I. Pineda, and A. Diaz-Barrios, "A protein folding robot driven by a self-taught agent,"

Biosystems, vol. 201, p. 104315, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0303264720301891

[10] S. Wu and Y. Zhang, "Protein Structure Prediction," *Bioinformatics*, pp. 225–242, 2009.

[11] B. Kuhlman and P. Bradley, "Advances in protein structure prediction and design," *Nature Reviews Molecular Cell Biology*, vol. 20, no. 11, pp. 681–697, 8 2019. [Online]. Available: http://dx.doi.org/10.1038/s41580-019-0163-x

[12] Z. Qin, L. Wu, H. Sun, S. Huo, T. Ma, E. Lim, P.-Y. Chen, B. Marelli, and M. J. Buehler, "Artificial intelligence method to design and fold alpha-helical structural proteins from the primary amino acid sequence," *Extreme Mechanics Letters*, vol. 36, p. 100652, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352431620300274

[13] W. D. Heaven, "AI for protein folding," 04 2022. [Online]. Available: https://www.technologyreview.com/2022/02/23/1044957/ai-protein-folding-deepmind/

[14] C. M. Dobson, "Protein misfolding, evolution and disease," vol. 24, no. 9, pp. 329–332.

[15] A. E. Gorbalenya, S. C. Baker, R. S. Baric, R. J. de Groot, C. Drosten, A. A. Gulyaeva, B. L. Haagmans, C. Lauber, A. M. Leontovich, B. W. Neuman, D. Penzar, S. Perlman, L. L. Poon, D. V. Samborskiy, I. A. Sidorov, I. Sola, and J. Ziebuhr, "The species severe acute respiratory syndrome-related coronavirus: classifying 2019-ncov and naming it sars-cov-2," pp. 536–544, 4 2020.

[16] B. Ratner, A. Hoffman, F. Schoen, and J. Lemons, *Biomaterials Science Second Edition: An Introduction to Materials in Medicine*, 2nd ed. Maarssen, Países Bajos: Elsevier Gezondheidszorg, 2004, vol. 5.

[17] M. J. Lopez and S. S. Mohiuddin, "Biochemistry, essential amino acids," 2020.

[18] D. L. Nelson and M. M. Cox, "Lehninger principles of biochemistry 6th edition," 2012.

[19] M. Deraitus and K. Freeman, "Essentials of cell biology," 2001.

[20] K. G. Roshni, "Protein folding, misfolding, and coping mechanism of cells - a short discussion." *doi. org/10.17352/ojcps*, vol. 4, 2021.

[21] J. S. Fraser, M. W. Clarkson, S. C. Degnan, R. Erion, D. Kern, and T. Alber, "Hidden alternative structures of proline isomerase essential for catalysis," *Nature*, vol. 462, no. 7273, pp. 669–673, 12 2009. [Online]. Available: http://dx.doi.org/10.1038/nature08615

[22] Z. Bu and D. J. E. Callaway, "Proteins move! protein dynamics and long-range allostery in cell signaling," *Advances in protein chemistry and structural biology*, vol. 83, pp. 163–221, 2011.

[23] H. Frauenfelder, S. G. Sligar, and P. G. Wolynes, "The energy landscapes and motions of proteins," *Science*, vol. 254, no. 5038, pp. 1598–1603, 1991.

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

[25] C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 5 1992. [Online]. Available: http://dx.doi.org/10.1007/bf00992698

[26] M.-J. Li, A.-H. Li, Y.-J. Huang, and S.-I. Chu, "Implementation of deep reinforcement learning," in *Proceedings of the 2019 2nd International Conference on Information Science and Systems*, ser. ICISS 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 232–236. [Online]. Available: https://doi.org/10.1145/3322645.3322693

[27] S. Agatonovic-Kustrin and R. Beresford, "Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research," pp. 717–727, 2000. [Online]. Available: www.elsevier.com/locate/jpba

[28] B. D. Ripley, *"Pattern recognition and neural networks"*. Cambridge University Press, 2007.

[29] M. W. Gardner and S. R. Dorling, "Artificial neural networks (the multilayer perceptron)-a review of applications in the atmospheric sciences," pp. 2627–2636, 1998.

[30] D. J. Graham and D. J. Field, "Sparse coding in the neocortex," *Evolution of nervous systems*, vol. 3, pp. 181–187, 2006.

[31] P. Foldiak and D. Endres, "Sparse coding," *Scholarpedia*, vol. 3, no. 1, p. 2984, 2008.

[32] A. Montalto, G. Tessitore, and R. Prevete, "A linear approach for sparse coding by a two-layer neural network," *CoRR*, vol. abs/1507.01892, 2015. [Online]. Available: http://arxiv.org/abs/1507.01892

[33] A. Krenker, J. Bester, and A. Kos, "Introduction to the Artificial Neural Networks," *Artificial Neural Networks - Methodological Advances and Biomedical Applications*, 4 2011. [Online]. Available: http://dx.doi.org/10.5772/15751

[34] J. P. Bigus, *Data mining with neural networks: solving business problems from application development to decision support*. McGraw-Hill, Inc., 1996, ch. Neural Network Models and Architechtures, pp. 63–64.

[35] P. Winder, *Reinforcement Learning: Industrial Applications of Intelligent Agents*. Culemborg, Países Bajos: Van Duuren Media, 11 2020.

[36] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," 8 2017. [Online]. Available: http://arxiv.org/abs/1708.05866http://dx.doi.org/10.1109/MSP.2017.2743240

[37] O. Chang, L. Zhinin-Vera, and F. Quinga-Socasi, "Self-taught neural agents in clever game playing," in *Proceedings of the Future Technologies Conference (FTC) 2020, Volume 1*, K. Arai, S. Kapoor, and R. Bhatia, Eds. Cham: Springer International Publishing, 2021, pp. 512–524.

[38] O. Chang and L. Zhinin-Vera, "A wise up visual robot driven by a self-taught neural agent," in *Proceedings of the Future Technologies Conference (FTC) 2020, Volume 1*, K. Arai, S. Kapoor, and R. Bhatia, Eds. Cham: Springer International Publishing, 2021, pp. 606–617.

[39] A. W. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. Žídek, A. W. Nelson, A. Bridgland *et al.*, "Improved protein structure prediction using potentials from deep learning," *Nature*, vol. 577, no. 7792, pp. 706–710, 2020.

[40] K.-C. Chou, "Prediction of tight turns and their types in proteins," *Analytical biochemistry*, vol. 286, no. 1, pp. 1–16, 2000.

[41] D. Wilson and T. Martinez, "The need for small learning rates on large problems," in *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*, vol. 1, 2001, pp. 115–119 vol.1.

[42] P. Kanerva, *Sparse distributed memory*. MIT press, 1988.

[43] A. C. Lin, A. M. Bygrave, A. de Calignon, T. Lee, and G. Miesenböck, "Sparse, decorrelated odor coding in the mushroom body enhances learned odor discrimination," *Nature Neuroscience*, vol. 17, no. 4, pp. 559–568, 2 2014. [Online]. Available: http://dx.doi.org/10.1038/nn.3660

[44] Wikipedia contributors, "Neural coding," 2022, [Online; accessed 25-October-2022]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Neural_coding&oldid=1116392448

[45] R. J. D. Groot, "Structure, function and evolution of the hemagglutinin-esterase proteins of corona-and toroviruses," *Glycoconjugate journal*, vol. 23, no. 1, pp. 59–72, 2006.

[46] D. L. Hurdiss, I. Drulyte, Y. Lang, T. M. Shamorkina, M. F. Pronker, F. J. van Kuppeveld, J. Snijder, and R. J. de Groot, "Cryo-em structure of coronavirus-hku1 haemagglutinin esterase reveals architectural changes arising from prolonged circulation in humans," *Nature communications*, vol. 11, no. 1, pp. 1–10, 2020.

[47] D. Sehnal, A. Rose, J. Koca, S. Burley, and S. Velankar, "Mol*: Towards a Common Library and Tools for Web Molecular Graphics," in *Workshop on Molecular Graphics and Visual Analysis of Molecular Data*, J. Byska, M. Krone, and B. Sommer, Eds. The Eurographics Association, 2018.

[48] Z. Wang, "Sparsednn: Fast sparse deep learning inference on cpus," *ArXiv*, vol. abs/2101.07948, 2021.

Appendices

Appendix 1.

Implementation

This phase consists of planning and executing the proposed model's coding. We chose the C++ programming language because it is a high-level language used for writing applications where performance and resource management are critical. It is commonly used in resource-intensive applications, AI in games, and robot locomotion. Also the adviser of this thesis and his students have developed extensive neural networks and agent libraries in C++. The proposed model's compiler is Borland C ++ 5.5, a C and C++ IDE (integrated development environment) that includes our neural libraries. The neural agent's modules and functions are listed below; some of these modules were created from scratch for this thesis, while others were adapted or refined from existing libraries.

- ***FireAgent.h***

  This module contains the neuronal agent, that is, a set of neurons that compete with each other, burn energy, and behave like a horse race, where the order of arrival determines the variation of the angles between amino acids; containing the following functions:

  - ***init_in_pesos_flop*** function that initializes the weights with the flop inputs.
  - ***get_winner*** function that finds the winner of the race.
  - ***fire_agent_net*** function where the neural agent burns dark energy, learns from the winner and checks for bending faults.

- ***NeuralLibmmt.h***

  Module containing the network's structure, such as its hidden layer and output layer, as well as its hyperparameters and neural network calculation functions.

  - ***random_weights*** function that generates random weights.
  - ***initialize_weights*** function that initializes the weights of the hidden layer and the output layer.
  - ***clean_weights*** function that clean the weights of the hidden layer and the output layer.

- ***correct_weights*** function that figures out both the output layer's and the hidden layer's errors. Additionally, it adjusts the hidden layer and output layer weights.
- ***sigmoide*** is the activation function used for network training.
- ***calculate_hidden_layer*** function to calculate the values of the hidden layer.
- ***calculate_output_layer*** function to calculate the values of the output layer.
- ***backpropagation*** function that contains the function calculate_hidden_layer and calculate_output_layer.

- ***PlotItems.h***

  Module that graphs the parameters of the network

  - ***plot_hidden_outputs*** function that plots the neurons of the hidden layer.
  - ***plot_hidden_weights*** function that plots the hidden weights.
  - ***plot_inputs*** function that plots the neurons of the input layer.
  - ***plot_outputs*** function that plots the neurons of the output layer.
  - ***plot_targets*** function that plots the targets, that is, the output neuron that shows the best move.
  - ***trim_weights*** function that trims the weights of the hidden layer

- ***PlotProteinLib.h***

  Module that graphs the parameters of the network

  - ***erase_protein*** function that erases the data of the path traveled by the protein
  - ***plot_sparse*** function that plots the sparse matrix.
  - ***plot_protein*** function that plot all the protein and their sensors.
  - ***vertical_protein*** function that plot the protein of vertical way.
  - ***plot_protein_sensors*** function that plots the reading (pixels) of the protein sensors.

- ***SparseCode.h***

  It was developed from scratch for this thesis. It converts the 17 angles between amino acids into a sparse code of 16-bit binary neurons. It mimics the effect of a deep neural network that converts dense code to sparse code [48].

  - ***is_sparse*** function that check if the number is sparse.
  - ***fill_sparse*** function that fill the number with the 16-bit.

- ***ProteinFolding.cpp***

  It is the main program that contains all the modules described above to be able to set up the environment and that the agent can realize the protein folding.