POLITECNICO DI TORINO

Master's Degree in ICT for Smart Societies

Master's Degree Thesis

Generative models for protein structure: A comparison between Generative Adversarial and Autoregressive networks

Supervisors

Prof. Enrico MAGLI, Supervisor

Prof. Stefano TUBARO, Co-supervisor

Prof. Andrea PAGNANI, Co-supervisor

Candidate

Letizia BERGAMASCO

Academic Year 2019/2020

Abstract

This thesis work is set in the context of synthetic protein sequences generation. Starting from a dataset of protein sequences that belong to the same protein family, the goal is to generate new sequences which are statistically indistinguishable from the ones in the same family. This is possible thanks to the recent advance in protein sequencing, which has made available a large number of protein family datasets. To do this, we use neural network generative models that are able to learn the probability distribution of a dataset, so that we can sample from that distribution and generate new synthetic data. In particular, two different kinds of models are proposed: generative adversarial networks (GANs) and autoregressive (AR) neural networks. Both approaches are implemented in Python, using the PyTorch framework. They are tested on two datasets that can be downloaded from the Pfam database, namely the multiple sequence alignments of the Kunitz/Bovine pancreatic trypsin inhibitor domain and of the Cyclic nucleotide-binding domain, respectively. The evaluation method is twofold: on the one side, we use one-point and two-point correlation plots to check if the empirical frequency counts of the amino acids in the initial dataset and in the generated dataset are similar. On the other side, we use sensitivity plots based on a method called direct coupling analysis, which can summarise a protein family's contact map or, in other words, its three-dimensional folded structure. The results show that both the implemented generative models are able to generate protein sequences that are statistically similar to the ones in the original dataset of their family. In fact, they present correlation between the amino acid frequency counts of the initial dataset and the generated dataset in the correlation plots. Moreover, the sensitivity plots reveal that the models can automatically learn also the three-dimensional folded structure of the considered protein family. With respect to this aspect, while in the Kunitz/Bovine pancreatic trypsin inhibitor domain dataset GANs result to perform better, in the Cyclic nucleotide-binding domain dataset AR networks show higher potential to capture the three-dimensional folded structure of the protein family. Overall, AR networks exhibit much shorter training times with respect to GANs.

Acknowledgements

I would like to thank my family for all the support they gave me during the course of my Master's Degree studies. Thank you mum and Virginia for always being there and believing in me, especially in the most difficult times. You have encouraged me with faith and patience, together with dad, who is accompanying us from above. Thank you also to my aunts, uncles, cousins, grandparents and godparents, for all the moments in which you made me realise that there is always someone that I can count on.

I am grateful to my friends and classmates, with whom I shared advice, experiences and knowledge during these years. The time spent with you made me become more open-minded and discover new ways to address the challenges. Each of you, in different ways and measures, has taught me something precious.

I gratefully acknowledge the assistance that I received from my supervisors, whose suggestions and guidance have contributed both to the realisation of this thesis work and to my personal growth. I would also like to extend my gratitude to the professors of the ICT for Smart Societies Master's Degree course, for their teachings and their kind availability.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis work explores the field of synthetic protein sequences generation. We start from a dataset containing protein sequences belonging to a family of proteins, namely sharing a common ancestor, and our goal is to generate new sequences which are statistically indistinguishable from the original data. This is possible thanks to the recent advance in protein sequencing, i.e. the process that allows to determine the sequence of amino acids constituting a protein, which has made available a large amount of datasets of protein sequences, each one related to a particular family. This kind of datasets can be processed by some particular mathematical models, called generative models, which can learn the probability distribution of the dataset and become able to generate new synthetic data. In particular, we have chosen to test two different kinds of generative models on the task of protein sequence generation: generative adversarial networks (GANs) and autoregressive (AR) neural networks. We have implemented them and used them to generate new protein sequences, then discussing and comparing the obtained results.

In this first Chapter, we introduce more in detail the context of protein sequences generation and we explain the architecture and the theoretical aspects behind the generative models that have been utilised in this work. In addition, we also provide a background about a method that we have used for the evaluation of the protein generation results, which is called direct coupling analysis.

1.1 Protein sequences generation

Proteins are macromolecules consisting of one or more folded polypeptides, which in turn are long chains made up of building blocks called *amino acids*. There are 20 different amino acids that can be commonly found in proteins. The single amino acids, also called *residues*, can link together in a polypeptide chain by forming amide bonds. When talking about the protein structure, we usually distinguish four different levels [1]:

1. the *primary structure*, which refers to the sequence of amino acids;
2. the *secondary structure*, which is related to the regular patterns formed by the different orientation of segments in the polypeptide backbone. The two main secondary structures are the α-helix and the β-pleated sheet;
3. the *tertiary structure*, which refers to the three-dimensional shape assumed due to the folding of the α-helixes and the β-pleated sheets;
4. the *quaternary structure*, which describes the aggregation of different protein molecules into a functional unit.

A summary of the different protein structure levels is shown in Figure 1.1. Proteins are involved in almost all cellular processes, thus being essential for life. The properties and functions of a protein depend on its specific amino acid sequence; however, within a protein family, i.e. a group of proteins sharing evolutionary ancestry, it is possible to find proteins with similar three-dimensional structure and biological functionality but with very different amino acids [4].

The synthetic generation of new protein sequences is an interesting field in the biology research, because it allows to explore the space of the possible amino acid configurations and it opens the way to the discovery of new configurations, which could improve our understanding of some biological processes, or could be used in medicine, chemistry and manufacturing applications. Manual techniques may have high costs and require domain expertise. On the other hand, generative models seem promising for this task, because they can be trained on existing protein datasets, so that they learn the data distribution and they become able to generate new realistic protein sequences.

Due to the recent advance in protein sequencing technology, nowadays there is high availability of protein sequences data. Therefore, it is possible to consider not only single amino acid sequences, but also families of proteins with a common ancestor, called homologous sequences [4]. They can be exploited to learn the probability distribution of protein sequences in a family. In particular, in this work the objective is to learn the probability of a specific amino acid to be found at a given position in the sequence. This probability depends on the position, but also on which amino acids can be found at other positions. In this way our models become generative, because they allow us to sample an amino acid at each position according to this probability, and generate novel protein sequences that

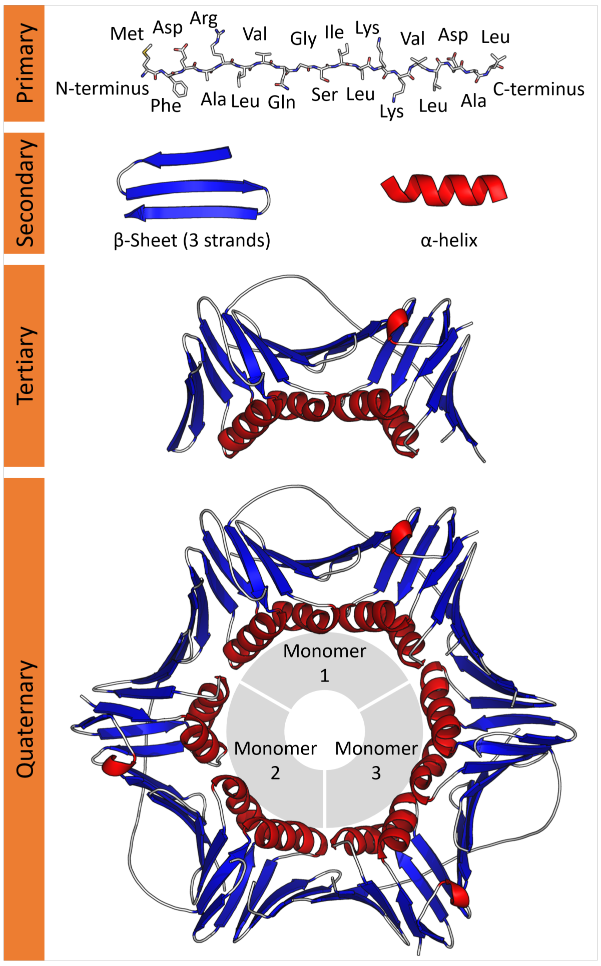

Figure 1.1: Structure of proteins. Retrieved from Wikimedia Commons, 2016 [2]. Licensed under the Creative Commons Attribution 4.0 International [3] license.

are statistically similar to those in their family.

We want to test and compare two different approaches to perform this task: generative adversarial networks (GANs) and autoregressive (AR) neural networks. GANs are a well-known models that are already having success in the generation of realistic data in very different fields, such as medicine or fashion. Autoregressive networks, instead, are more physical models, since they are based on the concept of autoregression, derived from time-series modelling. They are less known in generative modelling with respect to GANs; however, their use is spreading in different application domains and they have reached remarkable results in synthetic data generation.

Other works in literature have applied these neural network architectures in the area of synthetic biology, including studies that aim to create or redesign some biological entities. For example, in [5] the authors Anand and Huang apply GANs to the task of generating novel protein structures. Their objective is not to generate amino acid sequences, but they work with protein structural information, in terms of pairwise distances between α-carbons on the protein backbone. There are also other studies which make use of GANs in the generation of synthetic DNA sequences. One example is the work of Killoran et al. [6], aiming at creating synthetic DNA sequences with deep generative models. They explore the use of GANs and the use of a variant of a generative designed procedure called activation maximisation; in addition, they also combine these two methods. Their model is proved to capture the underlying structure in the training dataset and to generate new DNA sequences that exhibit this structure. Another work on DNA sequences generation is the one of Gupta and Zou [7], who propose a new architecture called Feedback GAN (FBGAN). It is based on a feedback loop training mechanism: a GAN model (following the WGAN-GP architecture, which will be explained in Section 1.4) is trained to generate new DNA sequences, which are then optimised for some desired properties via a function analyser, i.e. another separate neural network.

1.2 Machine learning, deep learning and neural networks

In the previous Section, we have mentioned concepts like deep generative models and neural networks. In order to better understand these concepts, let us step back and have a look at the general context in which we are. This context is the one of artificial intelligence (AI), which can be defined as a set of techniques that allow machines to mimic the human behaviour. Inside this set, there are some techniques that make use of statistical methods to improve their performance with the experience: these techniques are known as machine learning (ML). Deep learning (DL) is instead a branch of ML. The relationship between artificial intelligence, machine learning and deep learning is depicted in Figure 1.2.

Deep learning aims at replicating the behaviour of the brain in a computer. Starting from an initial dataset, the idea is to create models with many levels of processing, so that each level can learn intermediate representations of the data, with a level of abstraction that progressively increases [8]. It is mostly based on artificial neural networks (ANNs), also known simply as neural networks (NNs), since these models are made of different layers of the so-called *neurons*: these basic

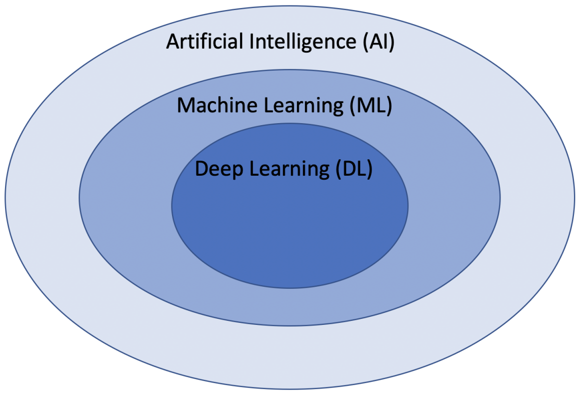

Figure 1.2: Relationship between artificial intelligence, machine learning and deep learning.

units receive inputs from the neurons of the previous layers, process them and provide an output to the neurons of the following layers, thus realising a hierarchical architecture. In general, a NN is said to be a deep neural network (DNN) when it is composed of 4 or more layers.

A representation of an example of a DNN is shown in Figure 1.3. This is a case of a fully connected network, where all the neurons in one layer are linked to all the neurons in the following layer. Each neuron is characterised by different parameters, namely the weights w_i and the biases b_i, and by an activation function σ. Assuming that a neuron has N inputs ($input_i$), its behaviour consists in combining the inputs in the following way:

$$x = \sum_{i=1}^{i=N} w_i \cdot input_i + b_i \tag{1.1}$$

The neuron's output is then calculated by applying the activation function on that result, as follows:

$$output = \sigma(x) \tag{1.2}$$

The weights and the biases of all the neurons are said to be learnable parameters for the neural network, since the models can learn them through a training process. During the training, these parameters are iteratively updated, usually to optimise a certain objective function.

In machine learning, and thus also in deep learning, we can find mainly two types of learning techniques: supervised and unsupervised. Supervised techniques

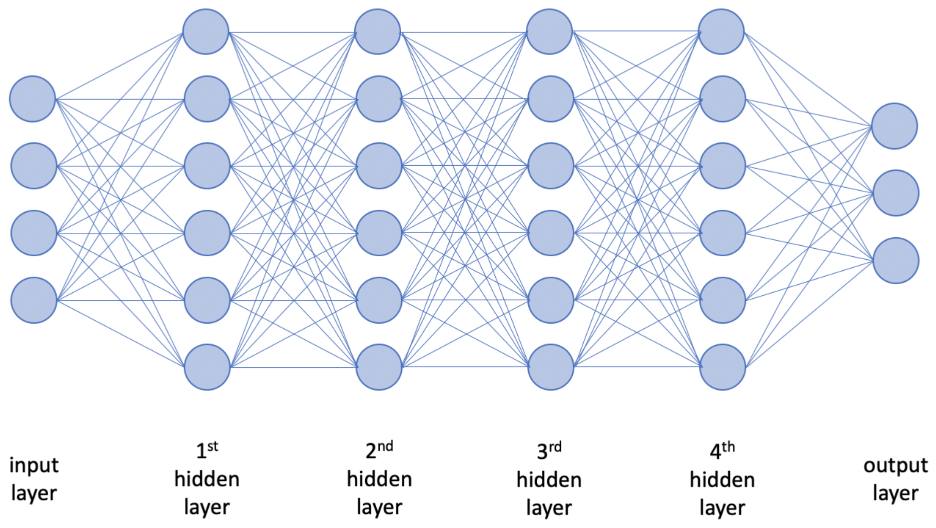

Figure 1.3: Representation of a fully-connected deep neural network.

work with labelled data: this means that they start from a dataset in which a ground truth is provided for each sample, and they aim at learning the relationship between the data samples and the corresponding ground truth. Typical examples of supervised learning are classifiers: these models start from a dataset where each data sample is associated to a class label, and they learn how to find the class label for a general data sample. On the other hand, unsupervised learning aims at finding interesting structures in the data, without having any label information about the data itself. An example of unsupervised learning is clustering: this technique allows to organise the data samples of the input dataset into groups, according to similarities in the data itself.

Another distinction that we can make in the DL field is between discriminative and generative models. The goal of discriminative models is to learn decision boundaries between data classes. Generative models, instead, learn the distribution of the input data. If we want to classify GANs and AR networks according to this framework, we can say that GANs are unsupervised generative models, while AR networks are supervised generative models. The reason why AR networks are supervised will be better understood from Section 1.5.

Before the advent of GANs, which will be discussed in Section 1.3, deep learning had little impact on generative modelling. After the success of GANs, deep generative modelling has taken hold, achieving remarkable results in the generation of realistic data in different fields of application.

1.3 Generative adversarial networks

Generative adversarial networks (GANs) are class of deep learning techniques that synthetise realistic data samples by learning from a training dataset in an unsupervised way. The term *adversarial* refers to the competitive game played by the two networks constituting the GAN framework, which are trained simultaneously:

1. the *generator*, whose goal is to create new synthetic data that is indistinguishable from the real data belonging to the training set;
2. the *discriminator*, which aims instead at correctly distinguishing the fake data produced by the generator from the real data in the training set.

While playing this game, the generator and the discriminator keep improving their capabilities: as soon as the discriminator learns to better distinguish fake from real data, the generator has to improve in generating more realistic samples, and vice versa, until the fake data are indistinguishable from the real data [9].

GANs have been introduced in 2014 by Goodfellow et al. [10]. Formally, the generator and the discriminator play a minimax game with the following loss function, which the generator tries to minimise and the discriminator tries to maximise:

$$\min_G \max_D \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}[\log(D(\boldsymbol{x}))] + \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[\log(1 - D(\tilde{\boldsymbol{x}}))] \tag{1.3}$$

where G is the differentiable function representing the generator, D is the differentiable function representing the discriminator, $\mathbb{P}_r$ is the distribution of the training dataset and $\mathbb{P}_g$ is the model distribution. In particular, $\mathbb{P}_g$ is implicitly defined by $\tilde{\boldsymbol{x}} \sim G(\boldsymbol{z}), \boldsymbol{z} \sim p(\boldsymbol{z})$, with $\boldsymbol{z}$ being the input of the generator, which is sampled from the noise distribution p [11].

When the discriminator is optimal, the loss function in equation 1.3 essentially quantifies how similar $\mathbb{P}_r$ and $\mathbb{P}_g$ are by a particular measure of similarity between two probability distributions, called Jensen-Shannon (JS) divergence [12].

A basic representation of a GAN architecture is shown in Figure 1.4. The generator and the discriminator networks can be trained by using standard gradient descent algorithms [13].

1.4 Wasserstein GANs

GANs have surely been a breakthrough for deep generative modelling; however, they have shown to suffer from training instabilities and convergence issues. Arjovsky et

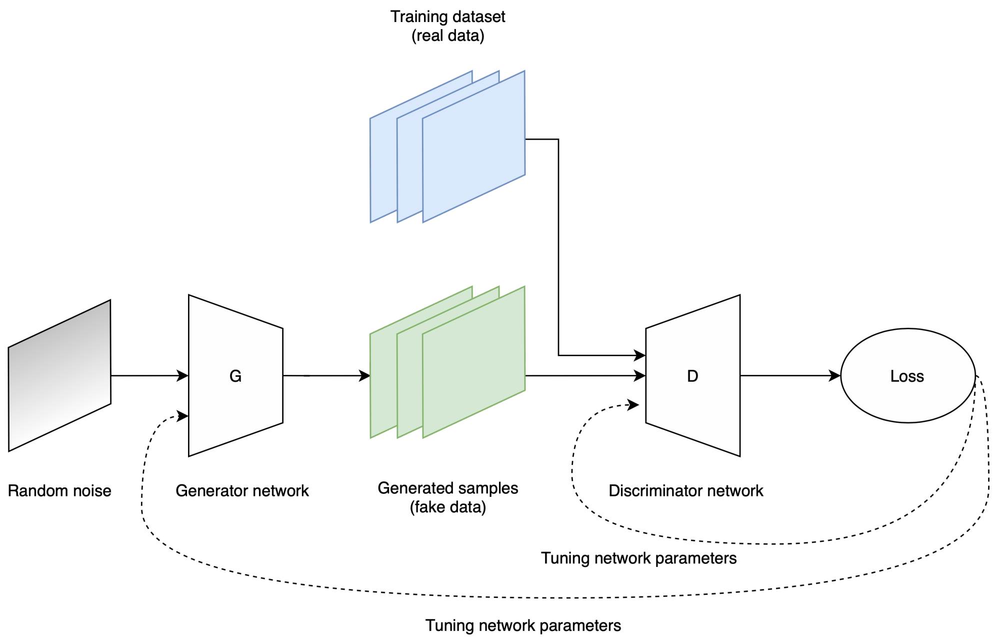

Figure 1.4: Basic representation of a GAN architecture.

al. have introduced a new form of GAN, called Wasserstein GAN (WGAN), to overcome these problems. Instead of optimising the Jensen-Shannon divergence of the distributions $\mathbb{P}_r$ and $\mathbb{P}_g$, WGAN optimises another measure of similarity between two probability distributions, called Earth-Mover (EM) distance or Wasserstein-1, which has nicer properties [14].

When the discriminator is optimal, the EM distance between $\mathbb{P}_r$ and $\mathbb{P}_g$ is minimised by minimising the new WGAN loss function [11]:

$$\min_G \max_{D \in \mathcal{D}} \mathop{\mathbb{E}}_{\boldsymbol{x} \sim \mathbb{P}_r} [D(\boldsymbol{x})] - \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}} \sim \mathbb{P}_g} [D(\tilde{\boldsymbol{x}}))] \tag{1.4}$$

where $\mathcal{D}$ is the set of 1-Lipschitz functions and $\mathbb{P}_g$ is the same as defined above for equation 1.3. This function provides another advantage with respect to GANs: it has shown to be correlated with the quality of the samples [14].

One problem that arises with WGANs is that there is a Lipschitz constraint on the discriminator, which is named as *critic* in [14]. In order to enforce this constraint, the authors propose to perform weight clipping, so that the weights of the critic lie in a compact space. However, a study by Gulrajani et al. [11] has highlighted that this practice biases the critic towards simpler functions and leads to some optimisation issues. The solution that they propose consists in enforcing the Lipschitz constraint by penalising the norm of the gradient of the critic with respect to its input. In particular, they introduce a penalty on the gradient norm for random samples $\hat{\boldsymbol{x}} \sim \mathbb{P}_{\hat{\boldsymbol{x}}}$. This results in the following new objective function:

$$L = \underbrace{\mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}} \sim \mathbb{P}_g} [D(\tilde{\boldsymbol{x}})] - \mathop{\mathbb{E}}_{\boldsymbol{x} \sim \mathbb{P}_r} [D(\boldsymbol{x})]}_{\text{Original critic loss}} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}} \sim \mathbb{P}_{\hat{\boldsymbol{x}}}} \left[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2 \right]}_{\text{New gradient penalty}} \tag{1.5}$$

where $\mathbb{P}_{\hat{\boldsymbol{x}}}$ is implicitly defined as sampling uniformly along straight lines connecting pairs points where the first term is sampled from $\mathbb{P}_r$, namely the data distribution, and the second term is sampled from $\mathbb{P}_g$, namely the generator distribution. λ is the penalty coefficient, and the authors have used $\lambda = 10$ in all their experiments, even with different datasets. One important aspect is that they avoid critic batch normalisation, since it would make the new training objective not valid anymore. This improvement version of the WGAN is called Wasserstein GAN gradient penalty (WGAN-GP), and it is now the de facto standard formulation which is employed in the research about GANs [9].

1.5 Autoregressive neural networks

The name *autoregressive network* derives from time-series modelling: an autoregressive model of order p, abbreviated as $AR(p)$, uses a linear combination of the past p values to predict the current value of the time series. It can be formulated as:

$$y_t = c + \sum_{i=1}^{i=p} w_i \cdot y_{t-i} + e_t \tag{1.6}$$

where y_t is the value of the time series at time t that has to be predicted, $y_{t-1,\dots,t-p}$ are the p past values that are used to predict y_t and e_t represents white noise [15].

In this work, we are dealing with protein sequences instead of time series, but the concept of autoregression that lies behind the utilised autoregressive networks is similar. We interpret a protein as an ordered sequence; each residue in the output sequence depends only on the residues of the input sequence that are located further behind, according to the natural primary sequence ordering of the protein family. This idea is depicted in Figure 1.5.

More in detail, autoregressive models are based on an exact Bayesian decomposition of the multivariate probability distribution induced by the empirical data. Since we are interested in learning the joint probability distribution $P(x_1, \dots, x_N)$

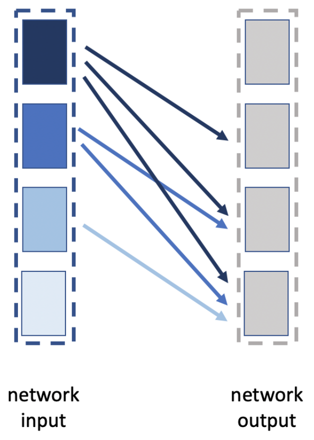

Figure 1.5: How the output sequence depends on the input sequence in the considered autoregressive networks.

of the data, we start from the observation that, according to the chain rule of probability, it can be factorised as follows [16]:

$$\begin{aligned} P(x_1, \ldots, x_N) &= P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_2, x_1), \ldots, P(x_N \mid x_1, \ldots, x_{N-1}) \\ &= \prod_{i=1}^{N} P(x_i \mid \mathbf{x_{<i}}) \end{aligned} \tag{1.7}$$

where $\mathbf{x_{<i}} = [x_1, \ldots, x_{i-1}]$ and by convention $P(x_1 \mid \mathbf{x_{<1}}) := P(x_1)$. By training a suitable neural network, we can find an approximation for $P(x_i \mid \mathbf{x_{<i}})$ for all $i \in 1, \ldots, N$. Once the model has been trained, it is easily generative: in order to sample a protein sequence $x_1, \ldots, x_N$, we can sample iteratively from univariate distributions, namely we extract $x_1 \sim P(x_1)$, then $x_2 \sim P(x_2 \mid x_1)$, and so on, until we get $x_N \sim P(x_N \mid x_1, \ldots, x_{N-1})$.

One interesting aspect of autoregressive models is that they are sequence models, thus working with sequential data, but they have been successfully applied also to non-sequential data, like images. A well-known example is PixelCNN, originally presented in [17]: it allows a sequential image generation process, since pixels are sampled one at a time and then given as input to the network. This architecture has been the starting point for further extensions and improvements, like PixelCNN++ [18], or the work of Reed et al. in text-to-image synthesis [19], or also WaveNet, a deep autoregressive model for audio generation applications such as text-to-speech and music [20].

Other remarkable works based on autoregressive neural networks can be found in literature. Among them, in [21] the authors present NADE (Neural Autoregressive Distribution Estimator), a new approach for the estimation of multivariate data distribution. Moreover, in [22] they introduce DARNS (Deep AutoRegressive Networks), deep generative autoencoders exploiting autoregressive hidden units and featuring a high-quality samples generation.

AR networks, like GANs, are feedforward, meaning that they do not contain cyclic connections between nodes. An important difference between GANs and AR networks is that the former are unsupervised, while the latter are supervised. The idea behind the fact that AR networks are supervised is that, during the training, in the loss function we compare the model's output to what is actually observed in the input sequence, which is treated as the ground truth.

1.6 Direct coupling analysis

In this work, after generating new protein sequences, we need a measure to assess how much the generated sequences feature characteristics that are similar to the ones of the corresponding protein family. For this purpose, on the one side we can use correlation plots to check the similarity of the amino acid empirical frequency counts in the training set and in the generated data, as will be explained in detail in Section 2.5.1. On the other side, we can analyse what happens to the three-dimensional folded structure, by exploiting direct coupling analysis (DCA).

DCA is a method for the identification of residue contacts in interacting proteins, which has been proposed in [23]. The word *residue* refers to a specific amino acid in a polypeptide chain, e.g. in a protein sequence. This method is an extension of a previous work [24], where a covariance-based approach was developed to identify residue contacts within a protein fold or between two interacting homologous proteins, starting from protein sequences alone. This kind of approaches rely on the basic assumption that residues in contact are constrained (not all amino acid combinations are allowed), and therefore they have different statistical properties with respect to residue pairs in other positions. In [24], Mutual Information (MI) of each residue position pair is utilised as a measure of residues proximity; however, since it is a local measure, i.e. it does not consider all the other residue positions in the protein, it does not allow to distinguish if high MI values derive from direct or indirect correlations. On the other hand, in DCA a global statistical inference step is added to overcome this shortcoming: the MI values calculated with covariance analysis are split into direct and indirect contributions, and only the direct contributions are considered [25].

Therefore, DCA allows to distinguish the directly correlated residues from the indirectly correlated ones. To compare different position pairs, the authors of [23] introduce a metric called *Direct Information* (DI), measuring the coupling strength of a position pair as the part of the mutual information due to direct coupling. They show that DI can be utilised as a reliable indicator of proximity of residues.

A further improvement to the DCA method is given in the more efficient implementation presented by Morcos et al. [26], called *mfDCA*. The reduction in the computational cost allowed the authors to perform their analysis on a larger scale, across different protein families. They showed that DCA can correctly predict a large number of contacts, defining a *contact* as a residue pair whose minimum atomic distance is less than 8Å (in order to consider only nontrivial predictions, the pairs separated by less than 5 positions along the protein's backbone were excluded from the analysis, because they would fall into the above mentioned definition of *contact*, but they would not be of interest). In other words, DCA is able to recapitulate the global structure of a family's contact map, which determines the corresponding three-dimensional folder structure. From now on, when mentioning DCA we will refer to the *mfDCA* implementation in [26].

Chapter 2

Experimental settings

In this Chapter, we firstly present the protein sequence datasets that are utilised for all the experiments in this work. Then, we provide a detailed explaination of the implementation of the selected generative models, respectively a WGAN-GP and an AR neural network. At the end, we describe the evaluation method that has been employed to judge the quality of the obtained results.

2.1 Datasets

The datasets that have been utilised for the experiments in this work come from the Pfam database [27]. It includes multiple sequence alignments (MSAs) and hidden Markov models (HMMs) for a large amount of protein families. An MSA can be seen as a rectangular matrix of size $N \times L$, with N being the number of protein sequences aligned according to similarities, and L being the number of positions over which the sequences are aligned. Each position in the sequence corresponds to one amino acid, and the standard IUPAC one-letter codes for the 20 natural amino acids are utilised [28]. The complete list of the codes is reported in Table 2.1. Since different sequences of the same family can have different lengths, in order to align them and to transform them into a MSA it is necessary to introduce a symbol representing the gaps. Therefore, in addition to the 20 letters of the alphabet used for the amino acids, the character $'-'$ is employed for the alignment gaps, resulting in a total number of symbols (named as Q) equal to 21. From now on, we will keep using the letter N to represent the number of protein sequences in the considered MSA, the letter L for the sequence length characterising the MSA and the letter Q to represent the number of possible symbols.

In particular, the MSAs of two families have been considered:

1. Kunitz_BPTI (PF00014): *Kunitz/Bovine pancreatic trypsin inhibitor* domain

Amino acid name	**Three-letter code**	**One-letter code**
alanine	Ala	A
cysteine	Cys	C
aspartic acid	Asp	D
glutamic acid	Glu	E
phenylalanine	Phe	F
glycine	Gly	G
histidine	His	H
isoleucine	Ile	I
lysine	Lys	K
leucine	Leu	L
methionine	Met	M
asparagine	Asn	N
proline	Pro	P
glutamine	Gln	Q
arginine	Arg	R
serine	Ser	S
threonine	Thr	T
valine	Val	V
tryptophan	Trp	W
tyrosine	Tyr	Y

Table 2.1: IUPAC names, three-letter codes and one-letter codes for the 20 natural amino acids [28].

[29]. This MSA contains $N = 10014$ sequences, aligned over $L = 53$ positions;

2. cNMP_binding (PF00027): *Cyclic nucleotide-binding* domain [30]. This MSA contains $N = 53017$ sequences, aligned over $L = 89$ positions.

The MSAs have been downloaded from the Pfam database [27] in FASTA format, a text format utilising the 20 alphabetical letters and the gap symbol to represent amino acids. However, in order to train all the neural network models described in Section 2.3 and Section 2.4, they have been converted into training datasets with a numerical format. In particular, the symbols

$$[A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y, -]$$

have been mapped respectively to

$$[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21].$$

2.2 PyTorch framework

All the generative models included in this work have been implemented in a Python environment. We have used PyTorch, an open source machine learning framework [31]; this framework provides accelerated tensor computation by exploiting the graphics processing units (GPUs), and is well suitable for deep learning applications, since it allows to easily build and train neural network architectures. In particular, the utilised version of Python has been 3.6.8 and the utilised version of PyTorch has been v1.5.0 with CUDA 10.1. The CUDA (Compute Unified Device Architecture) platform allows to write applications that can gain acceleration by executing parallel calculations on CUDA-enabled GPUs.

2.3 WGAN implementation

2.3.1 Motivation

A WGAN has been implemented in the gradient penalty version (WGAN-GP). This model has been chosen because it represents the de facto standard in the field of GANs [9], having a good theoretical grounding and higher performance with respect to GANs in their original formulation.

2.3.2 Discriminator network

The implemented discriminator network has the following characteristics:

- the input can be either the output of the generator or a sequence from the training set. In the latter case, the sequence from the MSA is encoded with a one-hot encoding scheme: for each sequence position a block of Q elements is created, and the element corresponding to the right symbol assumes value 1, while all the other elements assume value 0;

- 4 fully-connected layers of decreasing size (512, 256, 128, 1). The layers are fully connected because there is no meaningful reason to use convolution (i.e. applying filters to a layer's input) on this kind of data;

- activation function: Leaky Rectified Linear Unit (Leaky ReLU) or Exponential Linear Unit (ELU). The activation function is applied at the output of a neuron; since in a deep learning model there is a large number of neurons, this function is calculated many times and impacts the performance of the model. In this work we have tried both the Leaky ReLU, which is one of the most popular activations used in the DL field, and the ELU, which has been introduced in the work of Clevert et al. [32], showing a faster learning with respect to Leaky ReLU. They can be defined as follows:

$$LeakyReLU(x) = max(0, x) + c \cdot min(0, x) \tag{2.1}$$

$$ELU(x) = max(0, x) + min(0, \alpha \cdot (e^x - 1)) \tag{2.2}$$

where the coefficients c (which has to be a small negative slope) and α have been set to the default values used in PyTorch, namely $c = 0.01$ and $\alpha = 1.0$;

- the output is a scalar value. It does not represent a probability, but it can still be interpreted as a measure of how much the discriminator believes that the input comes from the real data in the training set.

A scheme of the discriminator architecture is illustrated in Figure 2.1.

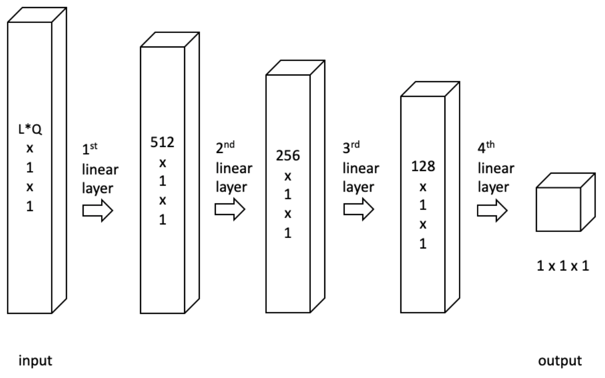

Figure 2.1: Scheme of the discriminator architecture.

2.3.3 Generator network

The generator network is characterised by the following aspects:

- the input is a random noise vector, sampled from a normal distribution. Two different values for the latent dimension have been tried: 100 and 16;
- 4 fully-connected layers of increasing size (128, 256, 512, $L \cdot Q$). Also here, the layers are fully connected because there is no meaningful reason to use convolution;
- activation function: Leaky ReLU or ELU, as for the discriminator;

- a final softmax layer, ensuring that each block of Q elements of the output sums up to 1. In this way, the output (with size $L \cdot Q$) can be interpreted in blocks of Q symbols, each one representing the probability distribution over Q symbols (20 amino acids + 1 gap symbol) for a given position in the protein sequence.

A scheme of the generator architecture is illustrated in Figure 2.2.

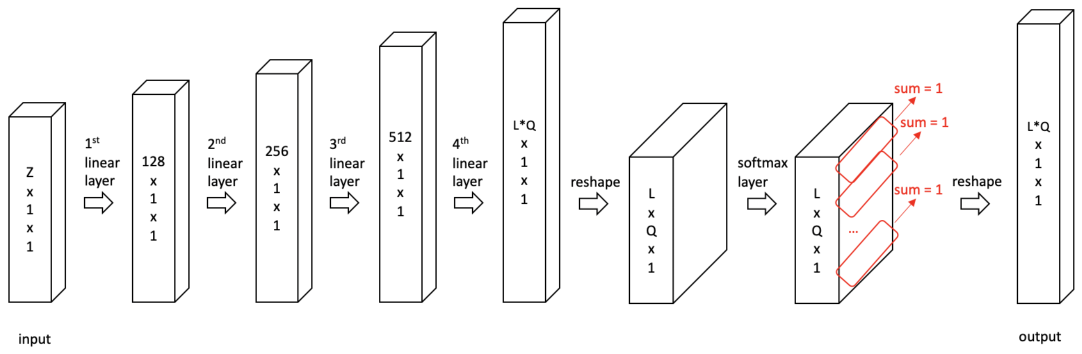

Figure 2.2: Scheme of the generator architecture.

2.3.4 Training

Hereafter we report the details about how the WGAN-GP has been trained. The new objective function described in Equation 1.5 is utilised. In summary, from Equation 1.4 and Equation 1.5 we get that the discriminator loss is:

$$L_D = \underset{\boldsymbol{x}\sim\mathbb{P}_r}{\mathbb{E}}[D(\boldsymbol{x})] - \underset{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}{\mathbb{E}}[D(\tilde{\boldsymbol{x}}))] + \lambda \underset{\hat{\boldsymbol{x}}\sim\mathbb{P}_{\hat{\boldsymbol{x}}}}{\mathbb{E}}\left[(\|\nabla_{\hat{\boldsymbol{x}}}D(\hat{\boldsymbol{x}})\|_2 - 1)^2\right] \tag{2.3}$$

For the generator, instead, the loss results to be:

$$L_G = \underset{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}{\mathbb{E}}[D(\tilde{\boldsymbol{x}}))] \tag{2.4}$$

since the generator can only affect the term related to fake data, i.e. $\underset{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}{\mathbb{E}}[D(\tilde{\boldsymbol{x}}))]$, and we can drop the term related to real data, i.e. $\underset{\boldsymbol{x}\sim\mathbb{P}_r}{\mathbb{E}}[D(\boldsymbol{x})]$.

Both for the discriminator loss and the generator loss, the expectation (indicated by $\mathbb{E}$) is calculated as an average over all the considered values. For the discriminator loss, the penalty coefficient has be kept fixed at $\lambda = 10$ in all the experiments. The gradient penalty term, i.e. $GP = \underset{\hat{\boldsymbol{x}}\sim\mathbb{P}_{\hat{\boldsymbol{x}}}}{\mathbb{E}}\left[(\|\nabla_{\hat{\boldsymbol{x}}}D(\hat{\boldsymbol{x}})\|_2 - 1)^2\right]$, requires to calculate the expectation over values that are sampled uniformly along straight lines connecting pairs points where the first term is sampled from $\mathbb{P}_r$ and the second term is sampled from $\mathbb{P}_g$, as explained in Section 1.4. What we have done in practice is to get random interpolations $\hat{\boldsymbol{x}}$ between real samples $\boldsymbol{x}$ and fake samples $\tilde{\boldsymbol{x}}$ in this way: $\hat{\boldsymbol{x}} = \alpha \cdot \boldsymbol{x} + (1 - \alpha) \cdot \tilde{\boldsymbol{x}}$, with the coefficient α being sampled from a uniform distribution over the interval [0,1].

The following settings have been utilised:

- Adam optimiser to train the generator, with learning rate $l_r = 0.0002$, decay of first order momentum of gradient $\beta_1 = 0.5$, decay of second order momentum of gradient $\beta_2 = 0.999$;
- Adam optimiser to train the discriminator, with learning rate $l_r = 0.0002$, decay of first order momentum of gradient $\beta_1 = 0.5$, decay of second order momentum of gradient $\beta_2 = 0.999$;
- mini-batches of size $size_{batch} = 128$;
- discriminator trained at every iteration, and generator trained only every $n_{critic} = 5$ iterations.

The algorithm which has been utilised to train the chosen WGAN-GP model is explained in Algorithm 1.

The number of parameters to be learnt during the training for the WGAN model includes both the weights and the biases for each fully-connected layer. According to the architectures shown in Figure 2.2 and in Figure 2.1, it can be calculated as:

$$\begin{aligned} n_{params,WGAN} &= n_{params,generator} + n_{params,discriminator} \\ &= [Z \cdot 128 + 128 + 128 \cdot 256 + 256 + 256 \cdot 512 + 512+ \\ &+ 512 \cdot L \cdot Q + L \cdot Q] + [L \cdot Q \cdot 512 + 512 + 512 \cdot 256+ \\ &+ 256 + 256 \cdot 128 + 128 + 128 \cdot 1 + 1] \end{aligned} \tag{2.5}$$

2.3.5 Sampling

The sampling consists in a feedforward step of the generator. Since the output of the generator represents the probability distribution for each sequence position, in order to sample a protein sequence we choose for each position the symbol with higher probability, among the Q symbols corresponding to that block.

The number of sampled sequences is equal to the number of sequences in the training set, for all the performed experiments.

Algorithm 1 WGAN-GP training algorithm.

```
1: for each epoch do
2:     for each training iteration do
3:         take a mini-batch of real sequences $\boldsymbol{x}$ from the training set
4:         sample a mini-batch of random noise vectors $\boldsymbol{z}$ from a uniform distribution $p(\boldsymbol{z})$
5:         give the mini-batch of random noise vectors as input to the generator, in order to obtain a mini-batch of fake sequences $\tilde{\boldsymbol{x}}$
6:         give the mini-batch of real sequences as input to the discriminator, obtaining $D(\boldsymbol{x})$
7:         give the mini-batch of fake sequences as input to the discriminator, obtaining $D(\tilde{\boldsymbol{x}})$
8:         calculate gradient penalty $GP$
9:         calculate discriminator loss as: $L_D = -average(D(\boldsymbol{x})) + average(D(\tilde{\boldsymbol{x}})) + \lambda * GP$
10:        backpropagate the loss to update the discriminator parameters
11:        if $n_{critic}$ iterations have passed from the last generator update then
12:            sample a mini-batch of random noise vectors $\boldsymbol{z}$ from a uniform distribution $p(\boldsymbol{z})$
13:            give the mini-batch of random noise vectors as input to the generator, in order to obtain a mini-batch of fake sequences $\tilde{\boldsymbol{x}}$
14:            give the mini-batch of fake sequences as input to the discriminator, obtaining $D(\tilde{\boldsymbol{x}})$
15:            calculate generator loss as: $L_G = -average(D(\tilde{\boldsymbol{x}}))$
16:            backpropagate the loss to update the generator parameters
17:        end if
18:    end for
19: end for
```

2.4 Autoregressive network implementation

2.4.1 Motivation

We have implemented a simple autoregressive network. It is a more physical model, and it is promising since the use of this kind of architecture is spreading and is reaching remarkable results in generative modelling.

2.4.2 Autoregressive network

The implemented autoregressive network has the following characteristics:

- the input is a sequence from the training set, without the last residue. Also here, the sequence from the MSA is firstly encoded with the same one-hot encoding scheme used for the WGAN discriminator. In this way, a block of Q elements is created for each sequence position, resulting in a total of L blocks of Q elements. Then, only the first $(L-1)$ blocks are selected as the input of the network, and the last block of Q elements is excluded;
- there is only one fully-connected layer of size $(L-1) \cdot Q$. The elements of the weight matrix of this layer are multiplied element-wise by a mask, in order to enforce the autoregressive property. A simple example of this step is illustrated in Figure 2.3;
- after the fully-connected layer, there is a final softmax layer. As for the generator of the WGAN, this layer ensures that each block of Q elements of the output sums up to 1 and can be interpreted as a probability distribution over Q symbols.

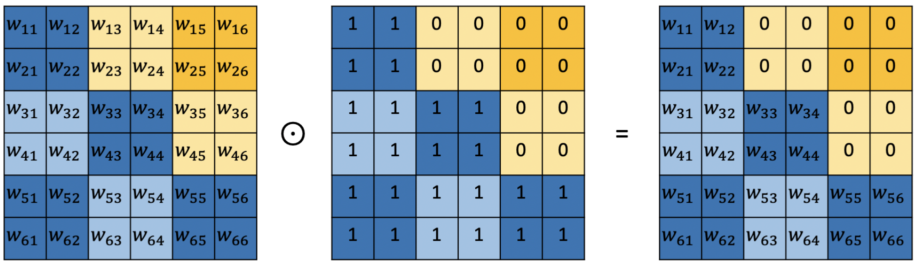
$$\begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} & w_{16} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} & w_{26} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} & w_{36} \\ w_{41} & w_{42} & w_{43} & w_{44} & w_{45} & w_{46} \\ w_{51} & w_{52} & w_{53} & w_{54} & w_{55} & w_{56} \\ w_{61} & w_{62} & w_{63} & w_{64} & w_{65} & w_{66} \end{pmatrix} \odot \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & 0 & 0 & 0 & 0 \\ w_{21} & w_{22} & 0 & 0 & 0 & 0 \\ w_{31} & w_{32} & w_{33} & w_{34} & 0 & 0 \\ w_{41} & w_{42} & w_{43} & w_{44} & 0 & 0 \\ w_{51} & w_{52} & w_{53} & w_{54} & w_{55} & w_{56} \\ w_{61} & w_{62} & w_{63} & w_{64} & w_{65} & w_{66} \end{pmatrix}$$

Figure 2.3: Example of the element-wise multiplication between the weight matrix of the fully connected layer and the mask, when $L = 4$ and $Q = 2$.

A scheme of the autoregressive network architecture is illustrated in Figure 2.4.

2.4.3 Training

The autoregressive network has been trained with the Binary Cross Entropy (BCE) loss function, defined as:

$$L_A = -\frac{1}{L-1} \sum_{i=1}^{L-1} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log(1 - \hat{y}_i) \tag{2.6}$$

where $\hat{y}_i$ is the element in the i^{th} position of the model's output sequence and y_i is the corresponding target value, namely the i^{th} position of the sequence from which the model's input has been selected. The loss function allows to compare the model's output, corresponding to $(L-1)$ positions, with the last $(L-1)$ positions

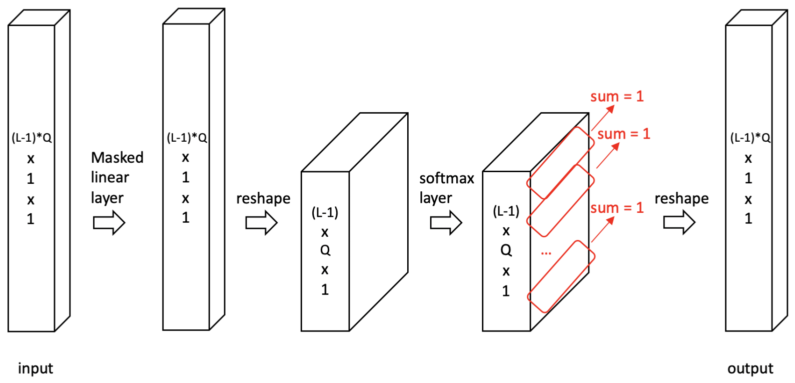

Figure 2.4: Scheme of the autoregressive network architecture.

of the original sequence. It is important to notice that, with this autoregressive network, we start from the first $(L-1)$ positions of the original sequence and we want to find the last $(L-1)$ positions, which constitute the model's output. Then, to calculate the loss, we go back to the original sequence but this time we consider only the last $(L-1)$ positions, and we compare them with the model's output, to quantify how similar they are.

L2 regularisation has been also applied. It adds a penalising term to the loss function, with the aim to obtain a model which is not too complex. The new obtained loss is the following:

$$L_{A,L2} = L_A + \lambda_{L2} \cdot \frac{L2_{reg}}{n_{params,AR}} \tag{2.7}$$

where:

- λ_{L2} is the regularisation rate. For all the experiments, we have set $\lambda_{L2} = 0.01$;
- $n_{params,AR}$ is the number of parameters of the network (including the weights and the biases). In this case, since the autoregressive network includes only one fully-connected layer of size $(L-1) \cdot Q$, whose weight matrix is multiplied by a mask (see Figure 2.3 and Figure 2.4), the total number of parameters results to be:

$$n_{params,AR} = [(L-1)^2 + (L-1)]/2 \cdot Q^2 + (L-1) \cdot Q \tag{2.8}$$

- $L2_{reg}$ is the regularisation term, calculated as the sum of the squares of the network parameters, in the following way:

$$L2_{reg} = \sum_{j=1}^{n_{params,AR}} param_j^2 \tag{2.9}$$

The other settings that have been utilised to train the autoregressive network are:

- Adam optimiser with learning rate $l_r = 0.0002$, decay of first order momentum of gradient $\beta_1 = 0.5$, decay of second order momentum of gradient $\beta_2 = 0.999$;

- mini-batches of size $size_{batch} = 32$.

2.4.4 Sampling

As introduced in Section 1.5, once we have trained the autoregressive network, we can generate new protein sequences by sampling one amino acid at a time from univariate distributions: firstly $x_1 \sim P(x_1)$, then $x_2 \sim P(x_2 \mid x_1)$, and so on, until we get $x_N \sim P(x_N \mid x_1, \ldots, x_{N-1})$.

$P(x_1)$ can be calculated as the frequency counts of the amino acids in the first position of the sequence within the training dataset. In other words, the probability assigned to each of the Q symbols is given by the ratio between the number of times in which that amino acid is found in the first position and N, i.e. the total number of sequences in the training MSA.

After having found the first amino acid x_1, we initialise a random sequence of $(L-1)$ elements and set the first element equal to x_1. The value of the other elements is not relevant, it can be any value, because this will not affect the generation of x_2. The value of x_2, in fact, depends only on the value of x_1. Then, the sequence is one-hot encoded and is given as input to the autoregressive network. The first block of Q elements of the model's output corresponds to $P(x_2 \mid x_1)$, so we sample x_2 from that distribution.

The random sequence of $(L-1)$ elements now has x_1 as a first element, and we have to set x_2 as the second element. Again, the value of the other elements in the sequence does not matter, because x_3 depends only on x_1 and x_2. The same procedure as before is applied: the sequence is one-hot encoded and is given as input to the autoregressive network. The second block of Q elements of the model's output corresponds to $P(x_3 \mid x_2, x_1)$, so we sample x_3 from that distribution. And this process goes on in the same way, sampling a symbol for each position until we have sampled the last element x_N from $P(x_N \mid x_1, \ldots, x_{N-1})$.

2.5 Evaluation method

2.5.1 Correlation plots

The first and most simple way to evaluate if the newly generated protein sequences are statistically indistinguishable from the sequences of the corresponding family is to look at the one-point and the two-point correlation plots.

The one-point correlation method consists in evaluating if the empirical frequency counts of the amino acids in the training set and in the generated data are similar. For both the considered MSAs, we calculate P_i, namely how many times we find each of the Q symbols in position i. Then, we plot these vectors one against the other: on the x-axis we have $P_{i,dataset}$, while on the y-axis we have $P_{i,generated}$. If the two vectors are correlated, we should expect this plot to be similar to a straight line.

On the other hand, in the two-point correlation method, we calculate P_{ij}, which means how many times we find a pair of symbols one in position i and the other in position j, respectively. Therefore, in the two-point correlation plot on the x-axis we have $P_{ij,dataset}$, while on the y-axis we have $P_{ij,generated}$. Also here, if the two vectors are correlated, the plot should be similar to a straight line.

The calculation of P_i and P_{ij} as carried out as formally described in [26]. With regards to the plots, they have been realised by exploiting Julia programming environment, and in particular the PyPlot module for Julia.

2.5.2 Sensitivity plot

A more sophisticated way to assess the efficacy of the protein sequence generation is to investigate if the generated MSA and the original one have the same contact map. To do this, we can utilise the DCA scores. In fact, as discussed in Section 1.6, DCA can show which residues are in close proximity and summarise a protein family's contact map.

In practice, here we carry out the same analysis as in [26]: we calculate the DCA scores for the training dataset and the generated data, and we exploit them to build a sensitivity plot, starting from 3 files:

1. a file containing a MSA in FASTA format, as described in Section 2.1;
2. a file in Profile Hidden Markov Model (HMM) save format (.hmm). This kind of files can be used with HMMER software [33], which is commonly used for the identification of homologous proteins and for making sequence alignments;

3. a file in the Protein Data Bank (PDB) format (.pdb), containing data about the three-dimensional structure of the considered proteins. This kind of files can be downloaded from the Protein Data Bank website [34].

In each sensitivity plot, for the selected MSA, on the x axis we represent the number of considered top-ranked pairs, according to the calculated DCA scores. On the y axis we report the sensitivity (i.e. the true positive rate) that is obtained considering up to each number of top-ranked pairs. The sensitivity is calculated as the fraction of pairs that are true contacts. The information about which of the predicted contacts are true contacts derives from the PDB file, containing the real three-dimensional structure with all the atomic distances. Also for the sensitivity analysis we have worked in the Julia programming environment, exploiting the PyPlot module for the plots.

With DCA, we actually do not get any direct measure of the similarity between the generated sequences and the ones in the training dataset. According to the utilised implementation of the WGAN and the AR network and the corresponding loss functions, we are not training these models to match the three-dimensional folded structure of the original data, which is determined by the real contacts between the residues, i.e. by the real contact map. In fact, in the training data there is no information about the three-dimensional structure. However, DCA can tell us to which extent the considered networks can capture it automatically, while learning the amino acids distribution.

Theoretically, for a MSA with sequences of length L, the total number of possible contacts would be $L \cdot (L-1)/2$. However, if we exclude the trivial contacts (pairs separated by less than 5 positions along the protein's backbone), we can focus only on the long range contacts, which are the ones that influence the proteins' three-dimensional structure. Within this restricted set of possible contacts, it is actually possible to determine the three-dimensional structure with only a number of contacts of the order of magnitude of L. This happens because once we fix around L contacts, then the structure becomes rigid. Therefore, if we were able to predict around L true contacts with DCA, this would allow us to recapitulate the whole three-dimensional structure.

Chapter 3

Generation of protein sequences with WGANs

In this Chapter, we report the experiments that have been carried out with WGANs on the two analysed datasets, described in Section 2.1. The WGAN implementation is the one explained in Section 2.3. For each experiment, we firstly show the correlation plots and then we consider the sensitivity plot, each time discussing the performance of the model.

3.1 PF00014 dataset

3.1.1 Case 1 - WGAN with Leaky ReLU activation and latent dimension equal to 100

As a first experiment, the dimension of the latent space has been set to 100 and the chosen activation function has been the Leaky ReLU. The model has been trained for 700000 iterations, corresponding to around 8860 epochs. By looking at the one-point correlation plot and the two-point correlation plot, respectively in Figure 3.1 and Figure 3.2, we can see that in both cases the points lie on a straight line. This suggests that the frequency counts of the training data and the generated data are correlated, giving us a first clue about the quality of the generated sequences.

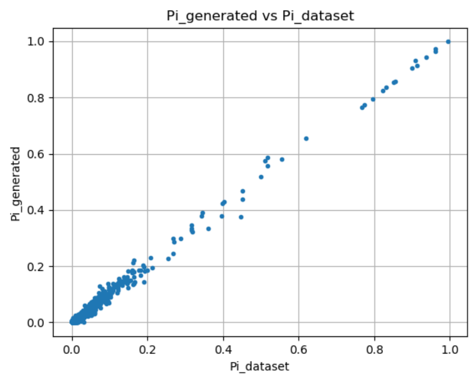

Figure 3.1: One point correlation. WGAN with latent dimension=100, activation=Leaky ReLU. Number of iterations=700000.

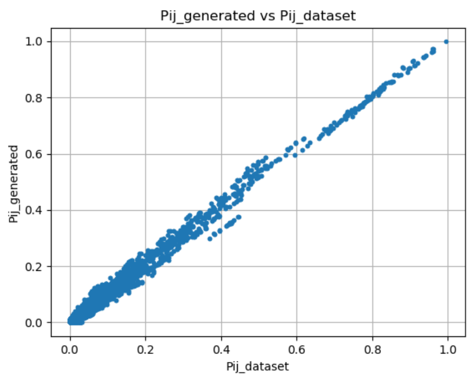

Figure 3.2: Two point correlation. WGAN with latent dimension=100, activation=Leaky ReLU. Number of iterations=700000.

The next step in the evaluation of the data generated with the considered WGAN is the analysis of the sensitivity plot. The original graph includes a higher number of points on the x axis (up to 1000), and is shown in Figure 3.3.

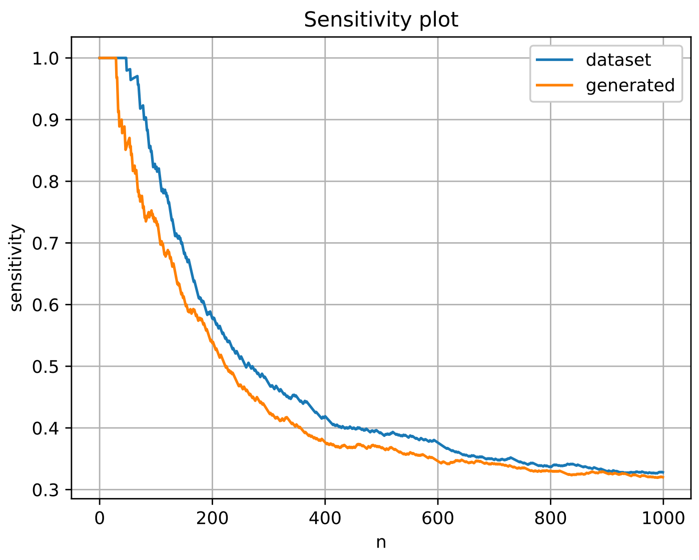

Figure 3.3: Original sensitivity plot, with all the points. WGAN with latent dimension=100, activation=Leaky ReLU. Number of iterations=700000.

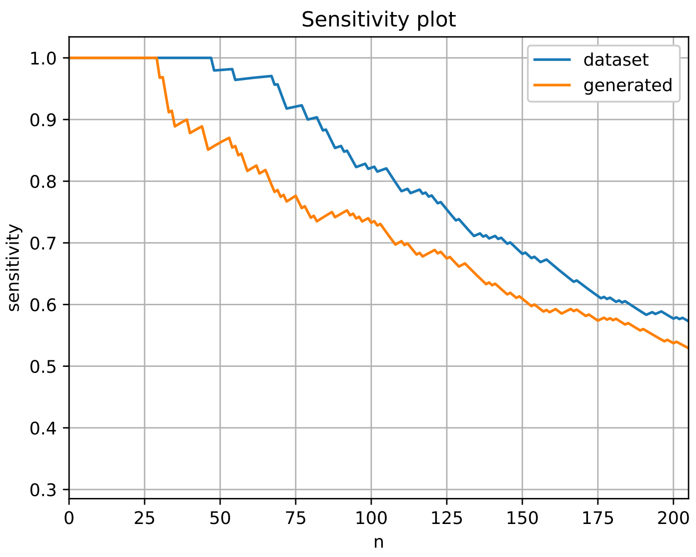

Figure 3.4: Sensitivity plot. WGAN with latent dimension=100, activation=Leaky ReLU. Number of iterations=700000.

From now on, for clarity, we will consider a zoomed version of the sensitivity plot where only the first 200 points are highlighted, since they are the most relevant. For this experiment, the zoomed sensitivity plot is reported in Figure 3.4. The blue curve represents the sensitivity curve generated from the DCA scores that are obtained directly from the training data. The orange curve, instead, has been computed by applying DCA to the synthetic data and assessing which of the predicted contacts are true contacts. As introduced in Section 2.5.2, the sensitivity for each point on the x axis is calculated as the fraction of pairs that are true contacts, considering up to that number of pairs, ranked according their DCA scores.

We can say for sure that the sequences generated by means of the WGAN are not completely random: in such a case, in fact, the sensitivity curve of the generated data would not have the same shape as the one of the training data; it would instead converge to the asymptotic value of the sensitivity curve of the training data (i.e., the asymptotic value of the blue curve in the original plot shown in Figure 3.3). In Figure 3.4, instead, the shape of the two curves is similar, and the sensitivity for the contact prediction with DCA stays above 0.8 just when considering the top 67 residue pairs. This means that 80% of the top 67 pairs, ranked according to DCA scores, are true contacts. Each point of the sensitivity curve can be read in this way: the sensitivity value (on the y axis) represents the percentage of true contacts that we have if we take into account only that specific number of top residue pairs (on the x axis) among the DCA predictions. This information is important, because it allows us to know how many predictions we need to consider if we want to know the true contacts with a given level of precision, given by the sensitivity value.

In addition to the point with sensitivity equal to 0.8, another value of interest in this kind of graph is the number of pairs that we can consider before that the sensitivity becomes lower than 1. In this case, we know from the sensitivity plot of the generated data that the first 30 contacts predicted by DCA are true contacts. If we were able to push this number closer to L, which is equal to 53 for the PF00014 dataset, we could actually say that we have determined most of the three-dimensional structure of that protein family.

One thing that can be checked to understand what the generator has learnt is its output. As already introduced in Section 2.3.3, this output has size $L \cdot Q$ and can be considered as composed by L blocks of Q symbols; each block represents the probability distribution over the Q symbols for the corresponding sequence position. When we sample from the generator, what we do is simply to select the symbol with maximum probability at each position from the generator's output. Therefore, we would like that each of the L probability distributions were as close as possible to a delta distribution. An example of the generator's output is plotted in Figure 3.5; on the x axis there are the Q symbols, while on the y axis there are the values of the probability distribution (between 0 and 1) for each of the L positions. Here the WGAN has been trained for 700000 iterations, and it is possible to see that almost all the distributions have the shape of a delta: in these cases, when we choose the symbol with the maximum probability for those positions, the confidence in the choice is high. Only in few cases the distribution has a different shape: in these cases, when we choose the value with highest probability, the confidence is lower. In Figure 3.5 we have reported just one example of the analysis of the generator's output; however, this analysis has been carried out for all the experiments, to monitor the learning process of the WGAN.

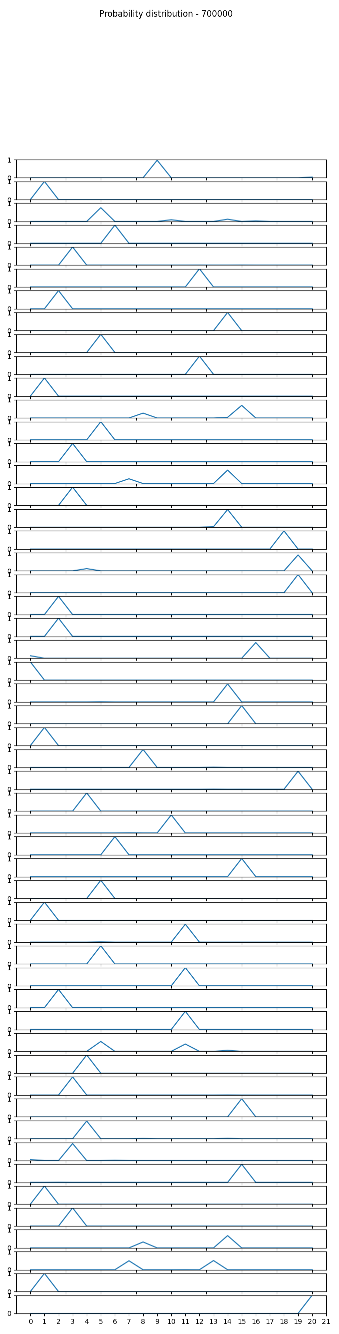

Figure 3.5: Visual representation of an example of the generator's output. WGAN with latent dimension=100, activation=Leaky ReLU. Number of iterations=700000.

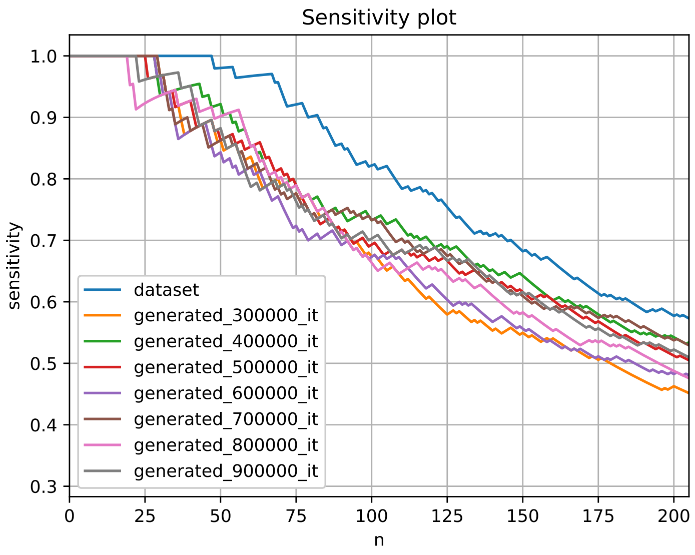

Figure 3.6: Sensitivity plot. WGAN with latent dimension=100, activation=Leaky ReLU. Number of iterations=300000, 400000, 500000, 600000, 700000, 800000, and 900000, respectively.

Figure 3.6 describes how the sensitivity curve varies when the same model is trained for different numbers of iterations. When working with ANNs to perform any classification task, training a model for too many iterations may result in overfitting, namely the model performs better in classifying the training data than in classifying other unseen testing data. In other words, the model learns the peculiarities and the noise of the training data and thus fails to generalise well on unseen data. On the other hand, defining and detecting overfitting conditions for generative models is still an open research field, including different opinions about the possibility for a GAN to memorise the training data [35] [36]. Therefore, in this work it is not clear if and how it is possible to detect overfitting conditions. In order to decide at which number of iterations to stop the training of a model, we will just analyse the trend of the sensitivity curves obtained with different number of training iterations. If it happens that, after a certain number of iterations, the sensitivity curves of the generated data start becoming less and less similar to the sensitivity curve of the training set, then we will consider only the model which gives the sensitivity curve that is the closest to the one of the dataset. As an example, in Figure 3.6 we can see that, overall, the best sensitivity curve is the one obtained after 700000 training iterations, therefore we are not interested in training the model for more iterations.

3.1.2 Case 2 - WGAN with Leaky ReLU activation and latent dimension equal to 16

Also for this second experiment, the chosen activation function has been the Leaky ReLU and the model has been trained for 700000 iterations (around 8860 epochs). However, the dimension of the latent space has been set to 16 instead of 100, in order to see if we can facilitate the training and improve the performance in sequence generation.

The one-point correlation plot and the two-point correlation plot are reported respectively in Figure 3.7 and Figure 3.8. The points lie again on a straight line, as expected. The highlighted correlation between the amino acid frequency counts of the training dataset and the generated dataset thus suggests the capability of the considered WGAN to generate new protein sequences that are statistically indistinguishable from the ones in their family.

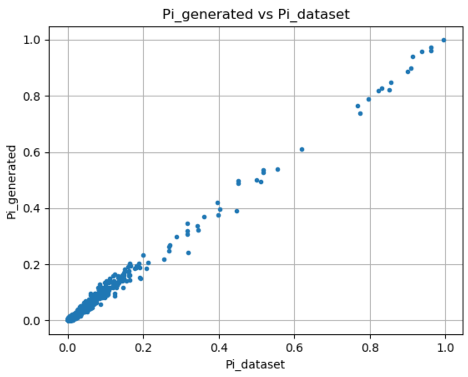

Figure 3.7: One point correlation. WGAN with latent dimension=16, activation=Leaky ReLU. Number of iterations=700000.

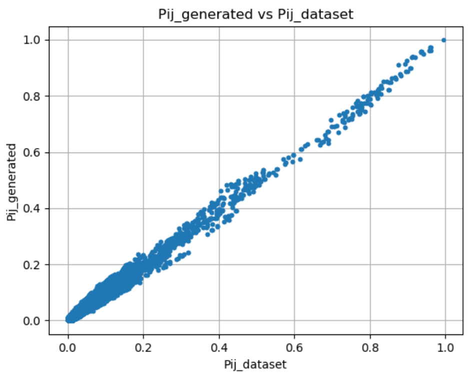

Figure 3.8: Two point correlation. WGAN with latent dimension=16, activation=Leaky ReLU. Number of iterations=700000.

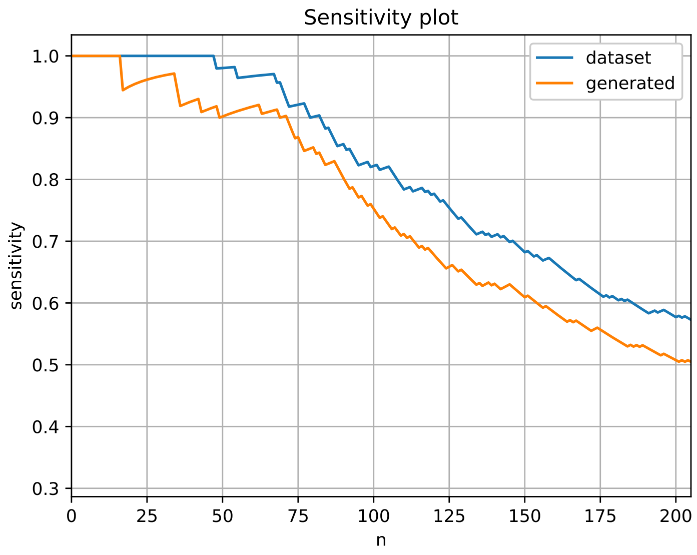

Figure 3.9: Sensitivity plot. WGAN with latent dimension=16, activation=Leaky ReLU. Number of iterations=700000.

Figure 3.9 shows the sensitivity plot for this experiment. When the size of the generator's input is set to 16 instead of 100, we can have an improvement in the performance: in fact, by training the WGAN for the same number of iterations, we get to a sensitivity curve for the generated pairs that has a trend more similar to the curve related to the initial dataset. This also results in a higher number of pairs that can be taken into account, still keeping the sensitivity above 0.8: while in the case of Section 3.1.1 we could consider only the top 67 residue pairs, here we can consider the top 91 pairs. A drawback with respect to the case of Section 3.1.1, instead, is that less than 30 top predicted contacts are true contacts: in fact, only the top 17 are true contacts.

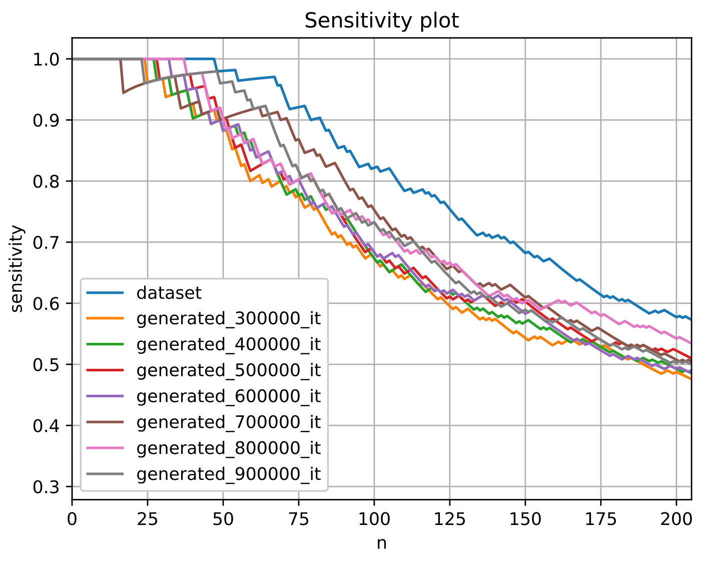

Figure 3.10: Sensitivity plot. WGAN with latent dimension=16, activation=Leaky ReLU. Number of iterations=300000, 400000, 500000, 600000, 700000, 800000, and 900000, respectively.

If we look at Figure 3.10, showing the sensitivity plot obtained by sampling the model after different numbers of iterations between 300000 and 900000, we can see that training the model for more than 700000 iterations allows to have a larger number of top predicted contacts which are also true contacts, but then the sensitivity falls down in a steeper way. Therefore, overall the sensitivity curve with the most similar shape with respect to the dataset curve is obtained with 700000 training iterations, and we will consider the corresponding model as the reference model for Case 2.

3.1.3 Case 3 - WGAN with ELU activation and latent dimension equal to 16

Given the improvement derived from the reduced dimensionality of the latent space, we have kept the dimension of the latent space equal to 16. This also facilitates the training, since there are less variables. In this experiment we have changed the activation function, utilising ELU in place of Leaky ReLU.

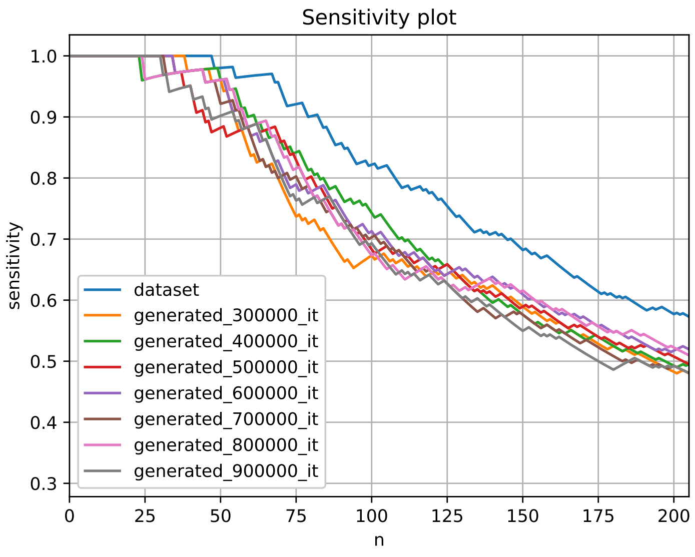

Figure 3.11: Sensitivity plot. WGAN with latent dimension=16, activation=ELU. Number of iterations=300000, 400000, 500000, 600000, 700000, 800000, and 900000, respectively.

Also this time, the model has been sampled after different numbers of iterations. However, in this case we can see from Figure 3.11 that 700000 iterations should be too many: after 400000 iterations, the sensitivity curve starts worsening. Therefore, we have decided to keep the model obtained after 400000 training iterations (around 5063 epochs). The corresponding correlations plot are shown in Figure 3.12 and Figure 3.13 respectively. The behaviour is still the expected one, namely following a straight line. The sensitivity plot is reported immediately after, in Figure 3.14.

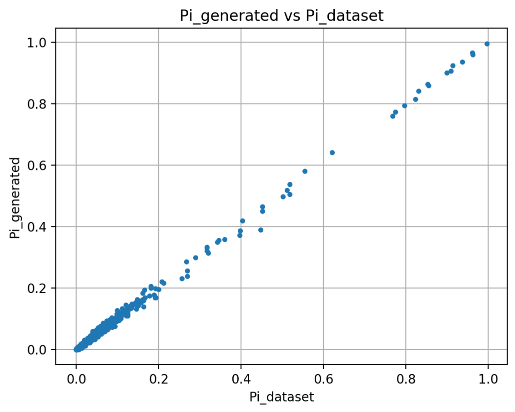

Figure 3.12: One point correlation. WGAN with latent dimension=16, activation=ELU. Number of iterations=400000.

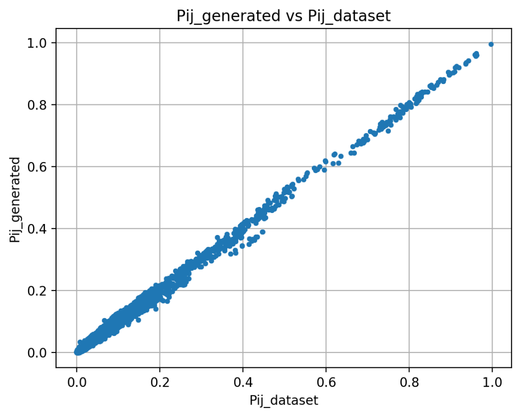

Figure 3.13: Two point correlation. WGAN with latent dimension=16, activation=ELU. Number of iterations=400000.

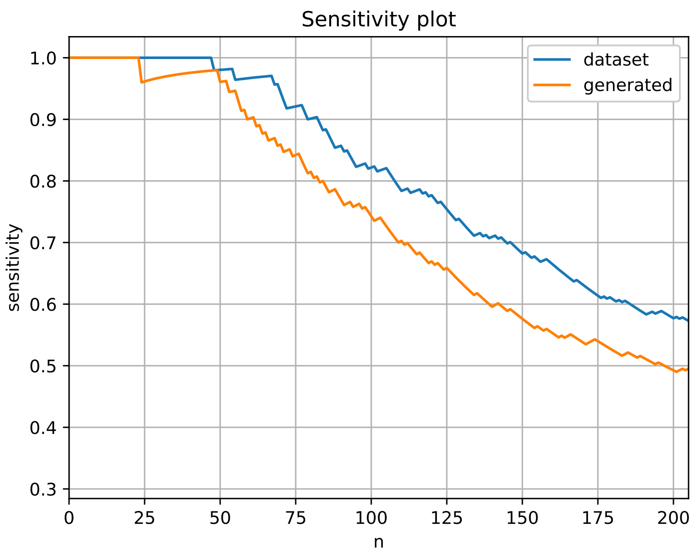

Figure 3.14: Sensitivity plot. WGAN with latent dimension=16, activation=ELU. Number of iterations=400000.

In this case, when we change the activation function from Leaky ReLU to ELU, still keeping all the other parameters fixed, we get a performance similar to the case of Section 3.1.2: the sensitivity is above 0.8 for the top 83 residue pairs. However, the first 24 pairs predicted as contacts by DCA are true contacts, and this is an improvement with respect to the case with Leaky ReLU activation. Moreover, an important difference with respect to Section 3.1.2 is that the model has been trained for less iterations: 400000 iterations instead of 700000 iterations, which means that the training iterations have been almost halved. This is a considerable saving in the computational time needed for the training.

3.2 PF00027 dataset

3.2.1 Case 4 - WGAN with Leaky ReLU activation and latent dimension equal to 16

With the PF00027 dataset, we have chosen to keep the dimension of the latent space equal to 16, and to experiment again different kinds of activation function, i.e. Leaky ReLU and ELU. Therefore, in this Section we start by considering what happens with Leaky ReLU.

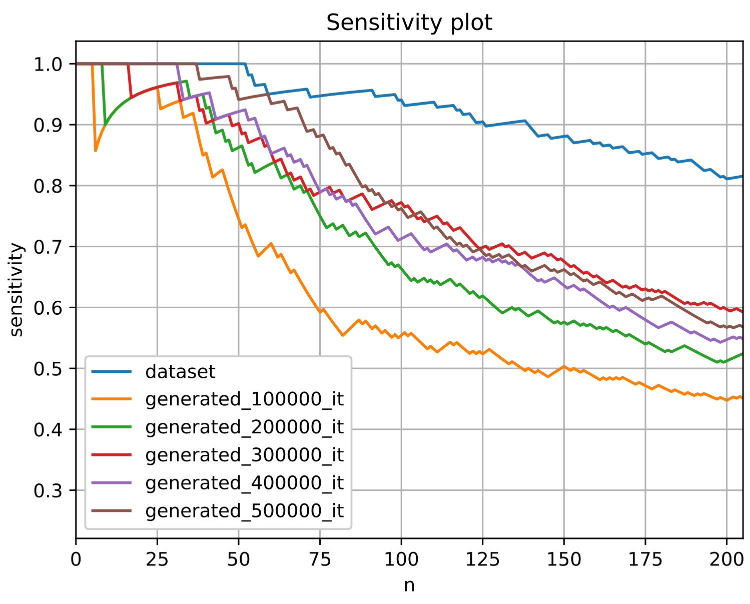

Figure 3.15: Sensitivity plot. WGAN with latent dimension=16, activation=Leaky ReLU. Number of iterations=100000, 200000, 300000, 400000, and 500000, respectively.

In order to decide a suitable number of training iterations, we have tried different possibilities and we have plotted the corresponding sensitivity curves in Figure 3.15. We can notice that the sensitivity curve that is closer to the one of the training dataset is the one obtained with the generated data sampled after 500000 iterations (around 1204 epochs). Therefore, in Figure 3.16 and Figure 3.17 we report the corresponding correlation plots, while in Figure 3.18 the sensitivity plot related only to that configuration is illustrated.

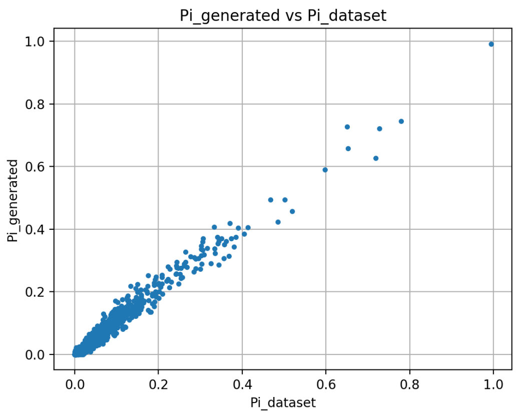

Figure 3.16: One point correlation. WGAN with latent dimension=16, activation=Leaky ReLU. Number of iterations=500000.

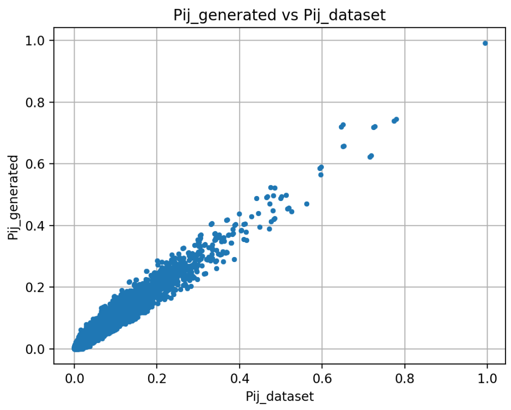

Figure 3.17: Two point correlation. WGAN with latent dimension=16, activation=Leaky ReLU. Number of iterations=500000.

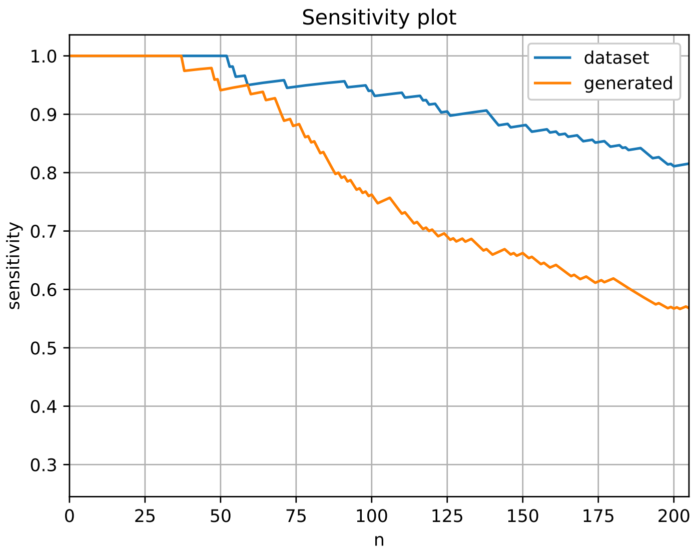

Figure 3.18: Sensitivity plot. WGAN with latent dimension=16, activation=Leaky ReLU. Number of iterations=500000.

The correlation plots, as for the other cases, allow us to highlight the correlation between the amino acid frequency counts of the training dataset and the generated dataset. In addition, from Figure 3.18 we can see that the sensitivity for the contact prediction with DCA stays above 0.8 when considering the top 88 residue pairs. The first 38 top predicted contacts are true contacts. This number has to be compared with L, which is equal to 89 for the PF00027 dataset. In fact, if we could predict $L = 89$ true contacts, this would allow us to determine most of the protein family's three-dimensional structure.

3.2.2 Case 5 - WGAN with ELU activation and latent dimension equal to 16

In this Section we consider instead what happens with ELU activation function. The same analysis with respect to Section 3.1.3 is carried out, so firstly we show in Figure 3.19 the sensitivity curves obtained after different numbers of iterations. It is evident that the curves of the generated data are more similar to the curve of the dataset, with respect to the previous case, illustrated in Figure 3.15. All the curves are closer to each other, and the best one seems to be the one obtained after 300000 iterations (around 722 epochs), therefore we will analyse that configuration. In Figure 3.20 and Figure 3.21 we report the corresponding correlation plots, while in Figure 3.22 there is the sensitivity plot with only the curve related to the dataset and the curve of the data generated after 300000 iterations.

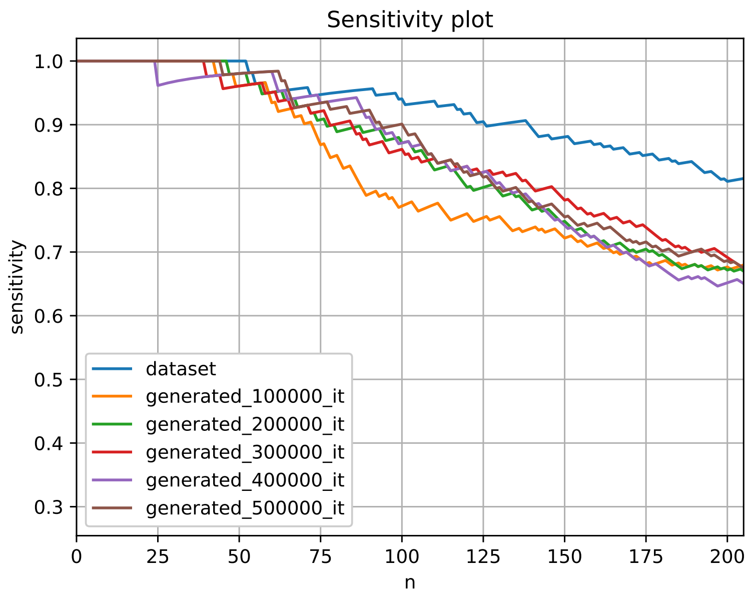

Figure 3.19: Sensitivity plot. WGAN with latent dimension=16, activation=ELU. Number of iterations=100000, 200000, 300000, 400000, and 500000, respectively.

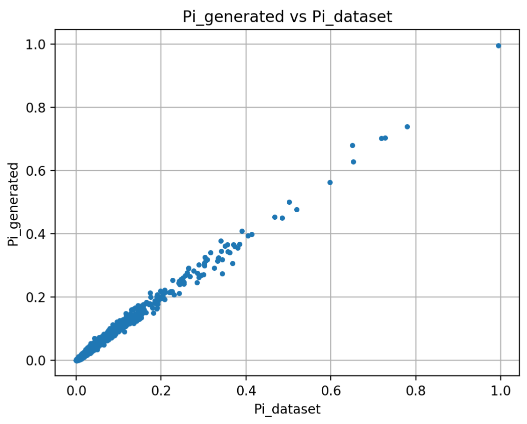

Figure 3.20: One point correlation. WGAN with latent dimension=16, activation=ELU. Number of iterations=300000.

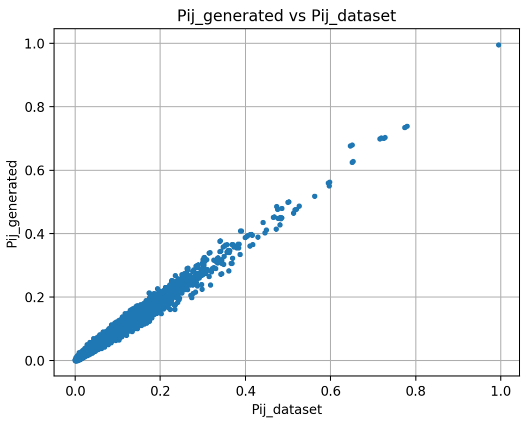

Figure 3.21: Two point correlation. WGAN with latent dimension=16, activation=ELU. Number of iterations=300000.

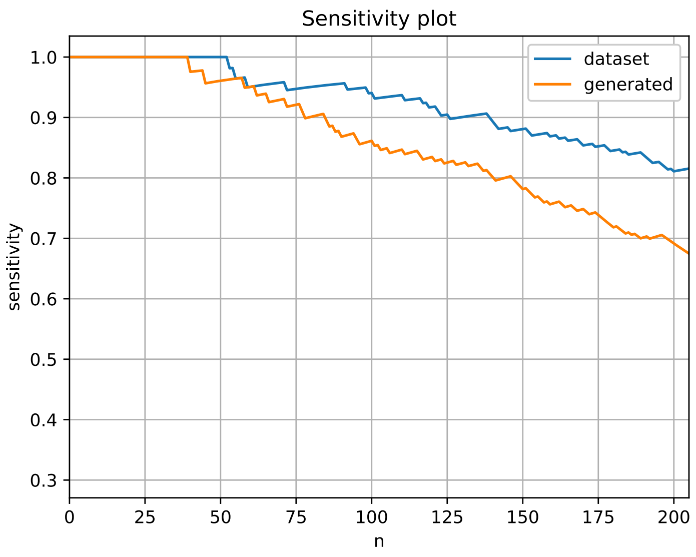

Figure 3.22: Sensitivity plot. WGAN with latent dimension=16, activation=ELU. Number of iterations=300000.

In Figure 3.18, the sensitivity for the contact prediction with DCA stays above 0.8 when considering the top 141 residue pairs. The number of top predicted contacts that are true contacts is comparable in the two cases: here with ELU activation is equal to 40, while with Leaky ReLU activation it was equal to 38. We can thus observe that with ELU activation function we can obtain better results in almost half of the iterations with respect to the case with Leaky ReLU activation.

Chapter 4

Generation of protein sequences with AR networks

In this Chapter, we report the experiments that have been carried out with autoregressive networks, implemented as explained in Section 2.4. The employed datasets are still the two MSAs described in Section 2.1. As we did for Chapter 3, also here we show the correlation plots and the sensitivity plot for each experiment, and we discuss the performance. Moreover, we present a comparison between the results obtained with WGANs and the ones obtained with AR networks.

4.1 PF00014 dataset

4.1.1 Case 6 - AR network

In this experiment, the autoregressive network model described in Section 2.4 has been trained for 40000 iterations (around 127 epochs). The one-point correlation plot and the two-point correlation plot, respectively in Figure 4.1 and Figure 4.2, include points lying on a straight line, thus indicating the presence of correlation between the frequency counts of the training data and of the generated data. This confirms that also AR networks, as the previously analysed WGANs, are capable of performing the considered protein generation task.

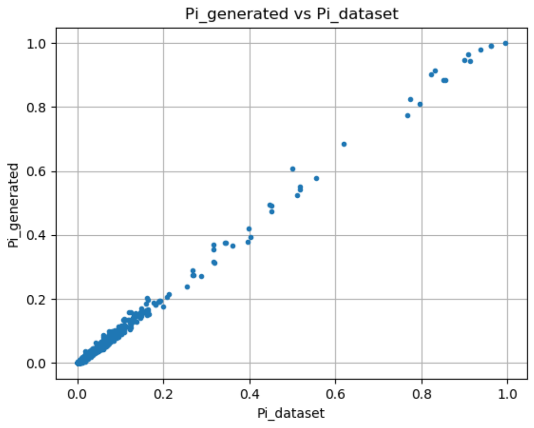

Figure 4.1: One point correlation. AR network. Number of iterations=40000.

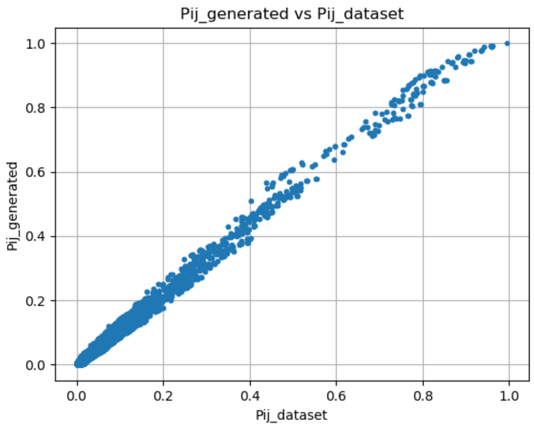

Figure 4.2: Two point correlation. AR network. Number of iterations=40000.

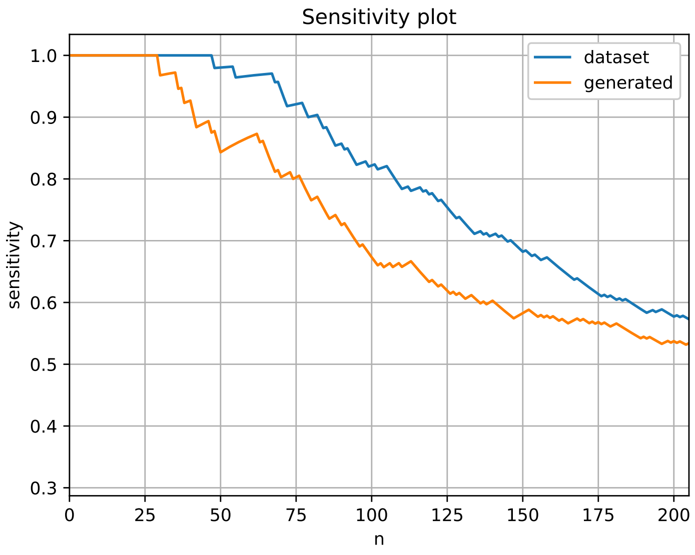

Figure 4.3: Sensitivity plot. AR network. Number of iterations=40000.

In Figure 4.3 we have the sensitivity plot. The curve of the generated data has the same shape with respect to the curve of the training dataset. The sensitivity for the contact prediction with DCA stays above 0.8 when considering the top 77 residue pairs, while it is equal to 1 for the top 30 predicted contacts, which are true contacts. When trying to increase the number of training iterations, there is no improvement of the sensitivity curve: from Figure 4.4 we can see that the sensitivity gets even worse when we train for 50000 iterations or for 60000 iterations.

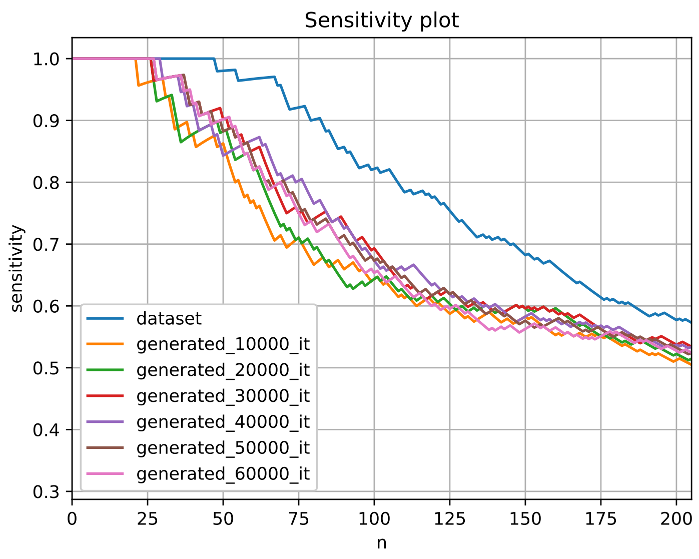

Figure 4.4: Sensitivity plot. AR network. Number of iterations=10000, 20000, 30000, 40000, 50000, and 60000, respectively.

4.1.2 Comparison between AR network and WGANs

If we compare the results obtained with the AR network in Section 4.1.1 with the ones obtained with WGANs on the same dataset, discussed in Section 3.1.2 and Section 3.1.3, in the AR case we have a slightly worse sensitivity curve. However, at the same time we have a higher number of top predicted contacts that are true contacts. In Figure 4.5 it is possible to visualise all the sensitivity curves together.

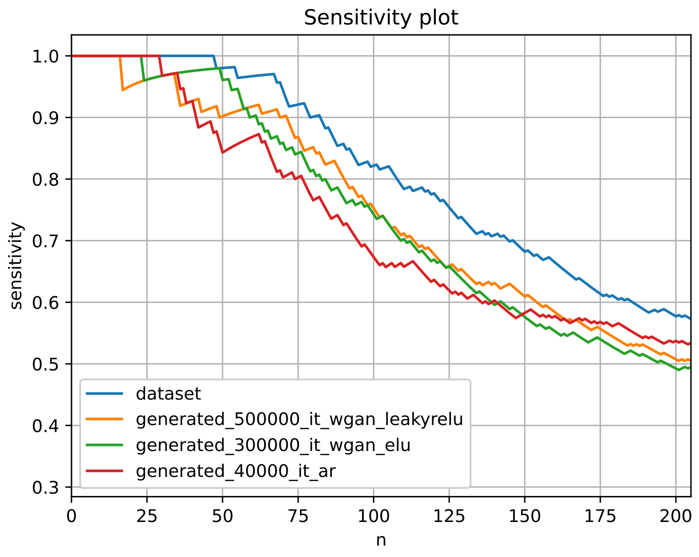

Figure 4.5: Sensitivity plot. Comparison between WGANs and AR network for PF00014 dataset.

The two GANs result to perform better in general, in particular the one with Leaky ReLU activation: in that case, if we consider the top 91 predicted contacts, we know that the 80% of them are true contacts, and this allows to have a more clear description of the three-dimensional folded structure of the protein family. However, if we wanted to consider only the number of top pairs with sensitivity equal to 1, since they are true contacts and provide more reliable information on the three-dimensional structure, then we should prefer the AR model.

Another important aspect which differentiates the implemented WGANs from the AR network is the amount of training time required, as anticipated in Section 3.1.3. In Table 4.1 we report the training times related to the experiments carried out on PF00014 dataset. All of them have been performed on the same machine, namely a server equipped with Nvidia GeForce 1080 Ti as GPU. We can notice that training the AR network has required only few hours, while training the WGANs has required several days. In this work the training time has not been a constraint, but different applications with tighter timing constraints would privilege the utilisation of AR networks.

Model	Training time
WGAN with Leaky ReLU activation (Case 2)	around 11.5 days
WGAN with Leaky ELU activation (Case 3)	around 6.5 days
AR network (Case 6)	around 4.5 hours

Table 4.1: Training times. Comparison between WGANs and AR network for PF00014 dataset.

Together with the training times, we also report the total number of parameters for both WGANs and the AR network, which can be visualised in Table 4.2. These calculations are based on the formulas in Equation 2.5 and Equation 2.8.

Model	Number of parameters
WGAN with Leaky ReLU activation (Case 2)	1472474
WGAN with Leaky ELU activation (Case 3)	1472474
AR network (Case 6)	608790

Table 4.2: Number of parameters. Comparison between WGANs and AR network for PF00014 dataset.

After having compared the results of WGANs and AR network in the sensitivity plot, one may wonder if the two types models are predicting more or less the same set of true contacts, or if they are able to predict different sets of contacts. In the latter case, it would be interesting to investigate the possibility of integrating the predictions of a WGAN and an AR network in order to predict a globally better set of contacts, thus obtaining a more accurate guess on the three-dimensional folded structure by means of DCA.

To perform this analysis, we have realised some density plots at fixed sensitivity, obtained from the same DCA results presented in the sensitivity plot of Figure 4.5. In particular, Figure 4.6 shows, both for the WGAN with Leaky ReLU activation and for the AR network, which of the predicted contacts are real contacts (i.e. with atomic distance less than 8Å; the trivial predictions, separated by less than 5 positions along the protein's backbone, are excluded from the analysis) and which are not real contacts, considering only the results with sensitivity equal to 1. In the background, in grey, we can see the real contacts; they are represented by a $L \times L$ matrix, where each grey dot with coordinates (i, j) indicates that the residue in position i in the alignment is in contact with the residue in position j. The matrix is symmetric, because if residue i is in contact with residue j, also residue j is in contact with residue i. In the upper triangular part, the contacts predicted with WGAN are superimposed, while in the lower triangular part we have superimposed the contacts predicted with the AR network. Moreover, in Figure 4.7 we have reported a similar plot, highlighting which true contacts are predicted by WGAN but not by AR (upper triangular part, in green), and which true contacts are predicted by AR but not by WGAN (lower triangular part, in blue), still with sensitivity fixed to 1. The same density plots have been realised at different levels of sensitivity: Figure 4.8 and Figure 4.9 for sensitivity greater than or equal to 0.9, and Figure 4.10 and Figure 4.11 for sensitivity greater than or equal to 0.8.

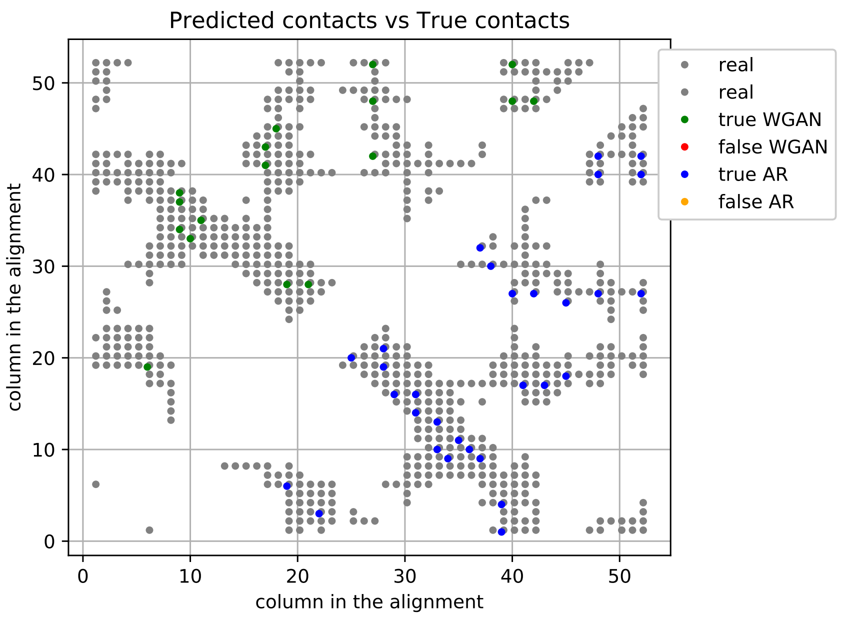

Figure 4.6: Density plot with sensitivity=1 for PF00014 dataset. Real contacts (grey), true predicted contacts (WGAN in green, AR in blue), false predicted contacts (WGAN in red, AR in yellow).

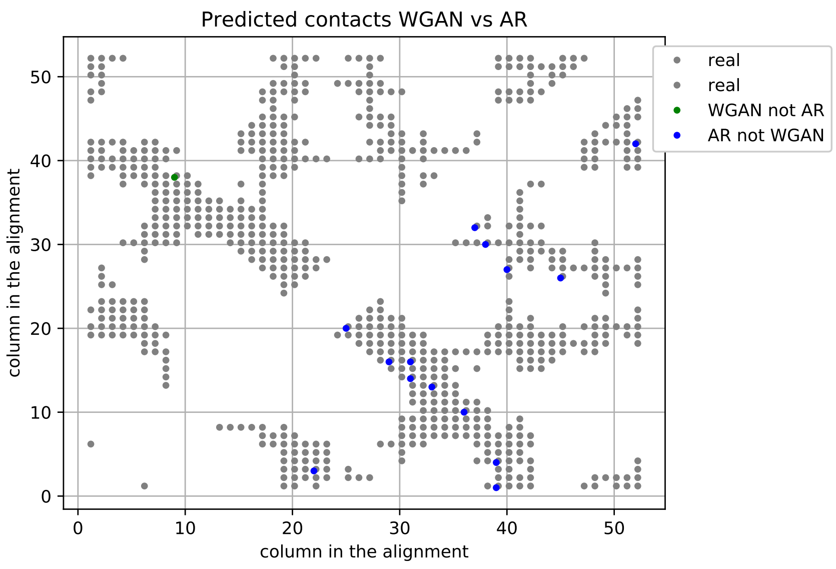

Figure 4.7: Density plot with sensitivity=1 for PF00014 dataset. Real contacts (grey), true contacts predicted with WGAN but not with AR (green), true contacts predicted with AR but not with WGAN (blue).

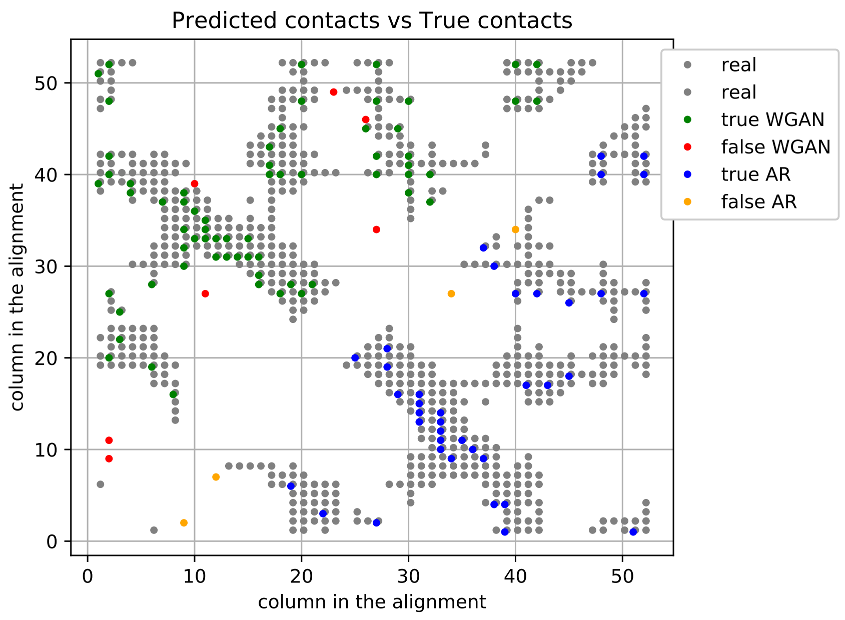

Figure 4.8: Density plot with sensitivity>=0.9 for PF00014 dataset. Real contacts (grey), true predicted contacts (WGAN in green, AR in blue), false predicted contacts (WGAN in red, AR in yellow).

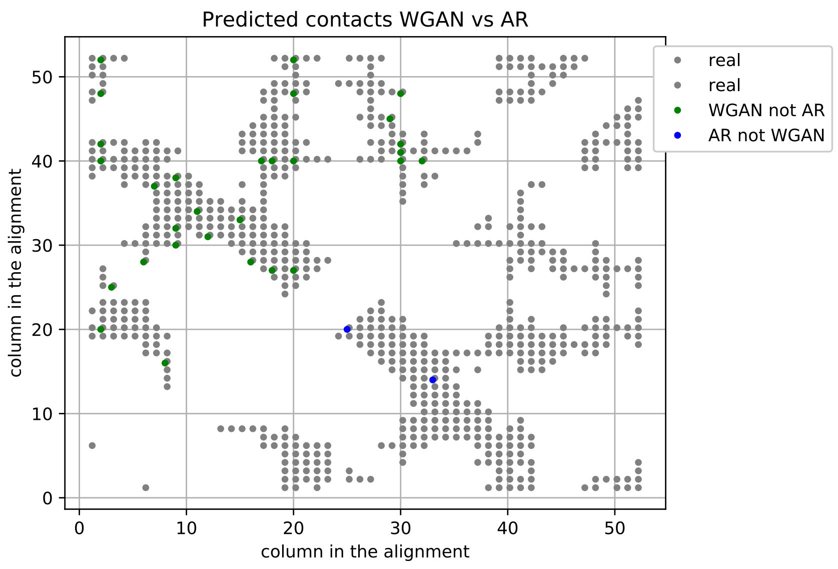

Figure 4.9: Density plot with sensitivity>=0.9 for PF00014 dataset. Real contacts (grey), true contacts predicted with WGAN but not with AR (green), true contacts predicted with AR but not with WGAN (blue).

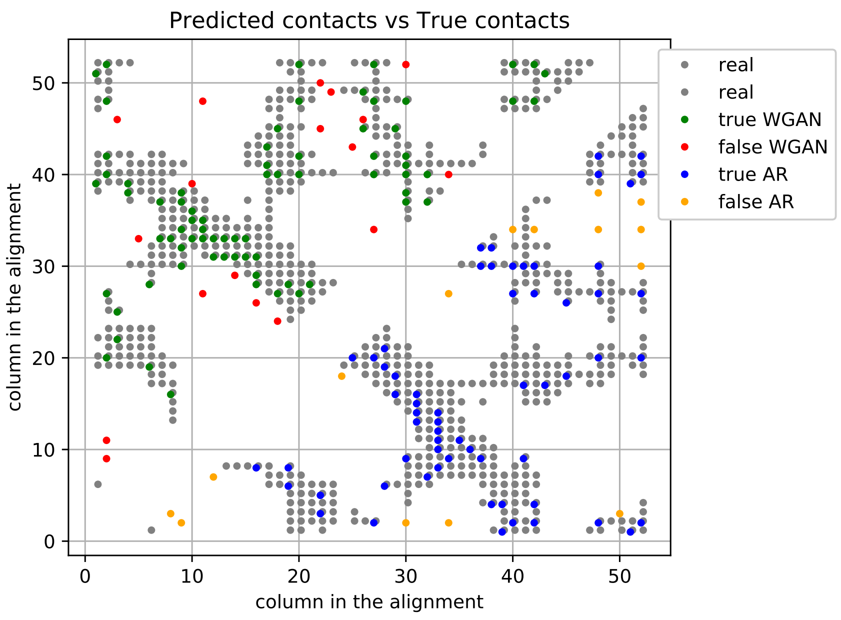

Figure 4.10: Density plot with sensitivity>=0.8 for PF00014 dataset. Real contacts (grey), true predicted contacts (WGAN in green, AR in blue), false predicted contacts (WGAN in red, AR in yellow).

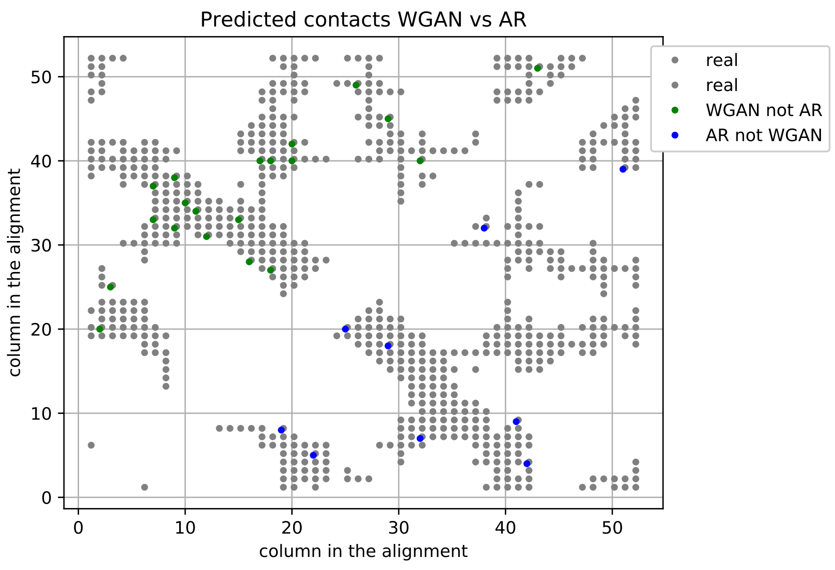

Figure 4.11: Density plot with sensitivity>=0.8 for PF00014 dataset. Real contacts (grey), true contacts predicted with WGAN but not with AR (green), true contacts predicted with AR but not with WGAN (blue).

In general, we can see from Figure 4.7 that, when the sensitivity is equal to 1, the AR network can predict almost all the true contacts predicted also by WGAN, and, in addition, also other contacts in different positions in the alignment. However, if we consider sensitivity greater than or equal to 0.9, in Figure 4.9 we can notice that the situation is reversed: this time it is the WGAN that can predict almost all the true contacts predicted by the AR, plus many other contacts. If we decide to consider sensitivity greater than or equal to 0.8, instead, the situation becomes more balanced, as we can see in Figure 4.11: for both the generative models considered, there are some common predicted true contacts, but also some true contacts that are predicted by one method and not by the other. Depending on the desired level of accuracy for the determination of the three-dimensional folded structure, one may decide to use only one of the two models for the protein generation task or to try to integrate the two approaches, as for example is suggested by the results obtained with sensitivity greater than or equal to 0.8.

4.2 PF00027 dataset

4.2.1 Case 7 - AR network

In this experiment, the same autoregressive model is trained on the PF00027 dataset. The one-point correlation plot and the two-point correlation plot obtained after 140000 iterations are shown respectively in Figure 4.12 and Figure 4.13. They both include points lying on a straight line, as in the previous case.

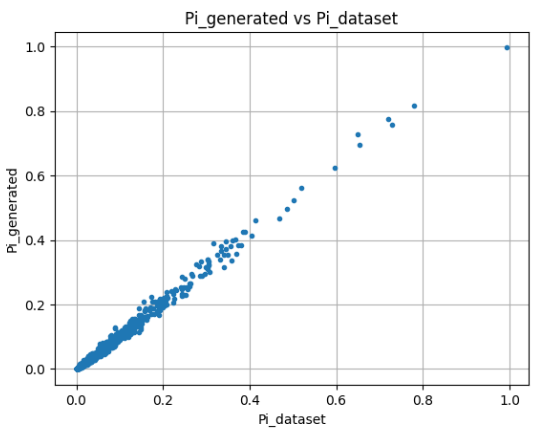

Figure 4.12: One point correlation. AR network. Number of iterations=140000.

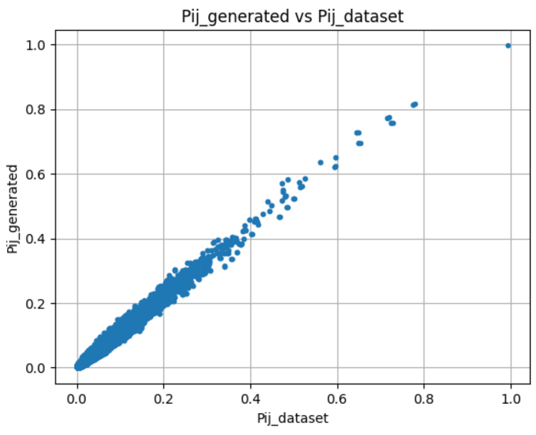

Figure 4.13: Two point correlation. AR network. Number of iterations=140000.

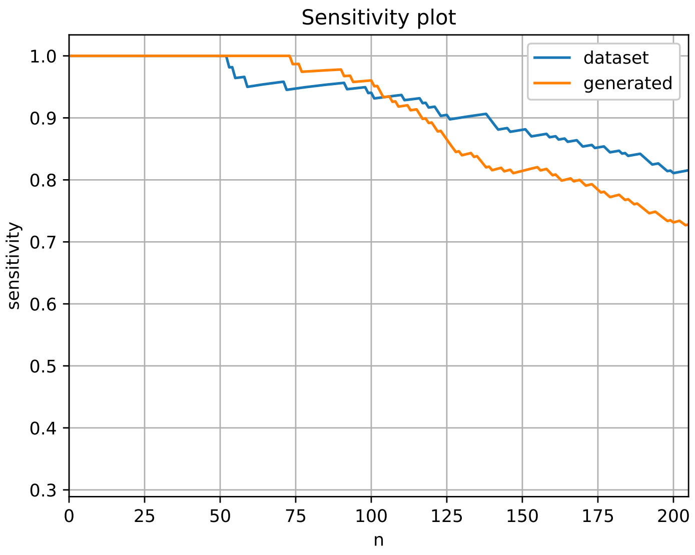

Figure 4.14: Sensitivity plot. AR network. Number of iterations=140000.

Figure 4.14 presents the sensitivity plot obtained by sampling the autoregressive model after 140000 training iterations (around 84 epochs). The sensitivity for the contact prediction with DCA stays above 0.8 when considering the top 163 residue pairs. The number of top predicted contacts that are true contacts is higher in the generated dataset than in the training dataset. This suggests that, in this experiment, for the generated data the three-dimensional structure can be better determined even than in the original protein family data. In fact, the 74 top predicted pairs are true contacts, and this number is closer to L, which is equal to 89 for the PF00027 dataset. This means that this model can almost learn the whole three-dimensional structure of this protein family.

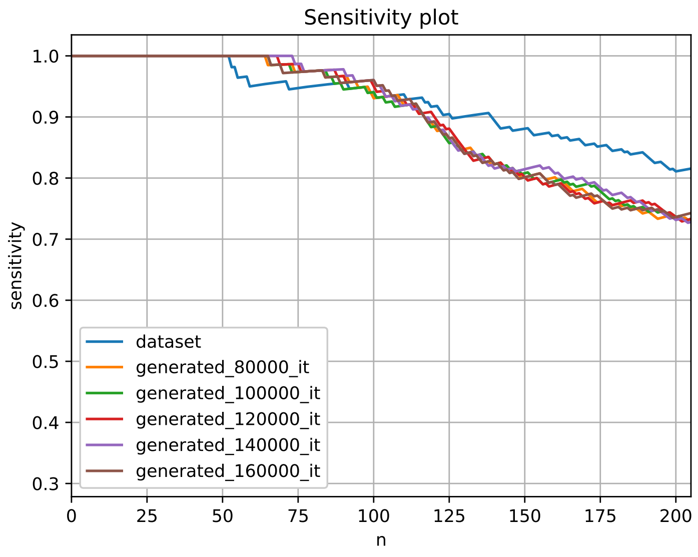

Figure 4.15: Sensitivity plot. AR network. Number of iterations=80000, 100000, 120000, 140000, and 160000, respectively.

Also here, increasing the number of training iterations does not lead to a better sensitivity curve. In fact, Figure 4.15 shows that the curve obtained with a model sampled at 160000 iterations reaches the sensitivity value of 0.8 when the number of considered contacts is less than 163.

4.2.2 Comparison between AR network and WGANs

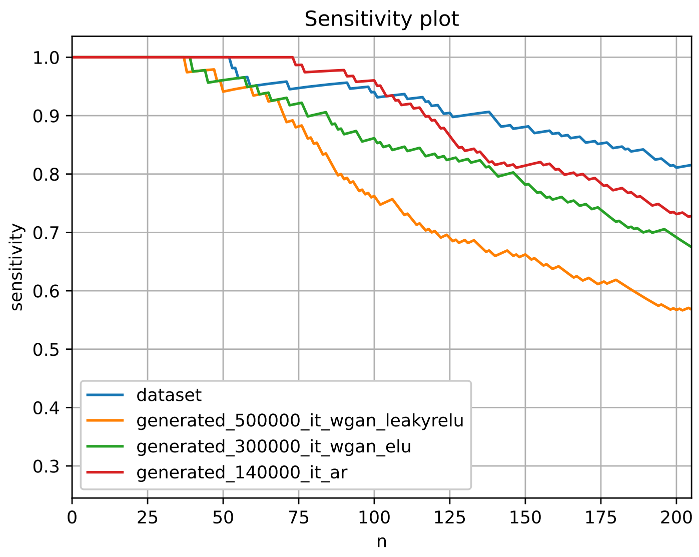

Figure 4.16: Sensitivity plot. Comparison between WGANs and AR network for PF00027 dataset.

With respect to the experiments carried out on the same dataset with WGANs, which have been presented in Section 3.2.1 and Section 3.2.2, in the case of AR networks presented in Section 4.2.1 the performance according to the sensitivity plot is better. This can be well visualised in Figure 4.16, which shows all the sensitivity curves together. The AR sensitivity curve not only has a shape which is more similar to the one of the dataset, but it also outperforms it in the number of top predicted contacts that are true contacts. In the experiments carried out on PF00027 dataset, AR networks have thus shown a higher potential to capture the three-dimensional folded structure of the protein family.

Moreover, the number of training iterations needed to achieve these results with the AR network is significantly lower with respect to the case of WGANs: for the autoregressive case, we need 140000 iterations, while with WGANs we need 300000 iterations in the best configuration, i.e. the one with ELU activation. This difference in the number of iterations (one case is almost half of the other case) means that we can save a considerable amount of computational cost if we use autoregressive networks instead of WGANs for this specific task. This observation is confirmed by the actual training times that have been required for the models' training; they are all reported in Table 4.3. As for PF00014 dataset, also for

PF00027 dataset we have performed the experiments on the same server, equipped with Nvidia GeForce 1080 Ti as GPU. We can notice that AR network clearly outperform WGANs with respect to the training time: around 24 hours have been employed to train the AR network, while several days have been required for WGANs.

Model	**Training time**
WGAN with Leaky ReLU activation (Case 4)	around 13.5 days
WGAN with Leaky ELU activation (Case 5)	around 8.5 days
AR network (Case 7)	around 24 hours

Table 4.3: Training times. Comparison between WGANs and AR network for PF00027 dataset.

The total number of parameters for both WGANs and the AR network has been calculated again according to Equation 2.5 and Equation 2.8, and they are reported in Table 4.4.

Model	**Number of parameters**
WGAN with Leaky ReLU activation (Case 4)	2247374
WGAN with Leaky ELU activation (Case 5)	2247374
AR network (Case 7)	1728804

Table 4.4: Number of parameters. Comparison between WGANs and AR network for PF00027 dataset.

Also for PF00027 dataset, we have investigated the relation between the sets of true contacts predicted by WGANs and AR network respectively. To do this, we have realised the same density plots that we have presented in Section 4.1.2 for PF00014 dataset; this time we have started from the results presented in Figure 4.16 and we have compared the AR network and the WGAN with ELU activation, since it was better performing with respect to the one with Leaky ReLU activation. The plots are reported in Figure 4.17, Figure 4.18, Figure 4.19, Figure 4.20, Figure 4.21 and Figure 4.22, respectively.

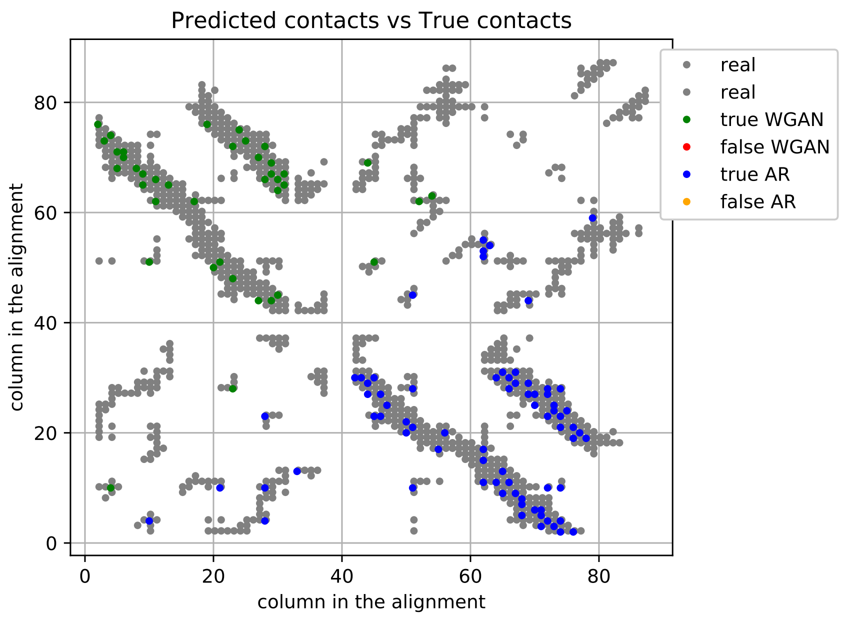

Figure 4.17: Density plot with sensitivity=1 for PF00027 dataset. Real contacts (grey), true predicted contacts (WGAN in green, AR in blue), false predicted contacts (WGAN in red, AR in yellow).

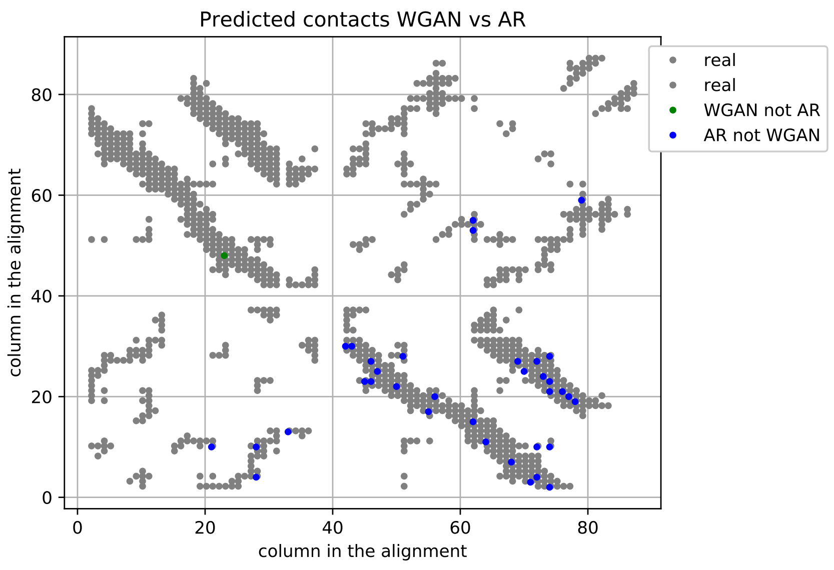

Figure 4.18: Density plot with sensitivity=1 for PF00027 dataset. Real contacts (grey), true contacts predicted with WGAN but not with AR (green), true contacts predicted with AR but not with WGAN (blue).

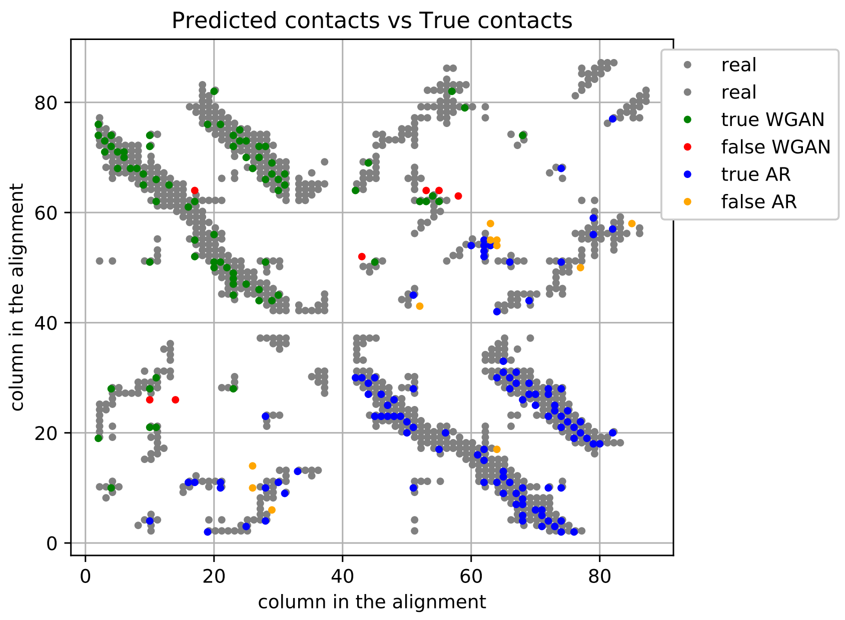

Figure 4.19: Density plot with sensitivity>=0.9 for PF00027 dataset. Real contacts (grey), true predicted contacts (WGAN in green, AR in blue), false predicted contacts (WGAN in red, AR in yellow).

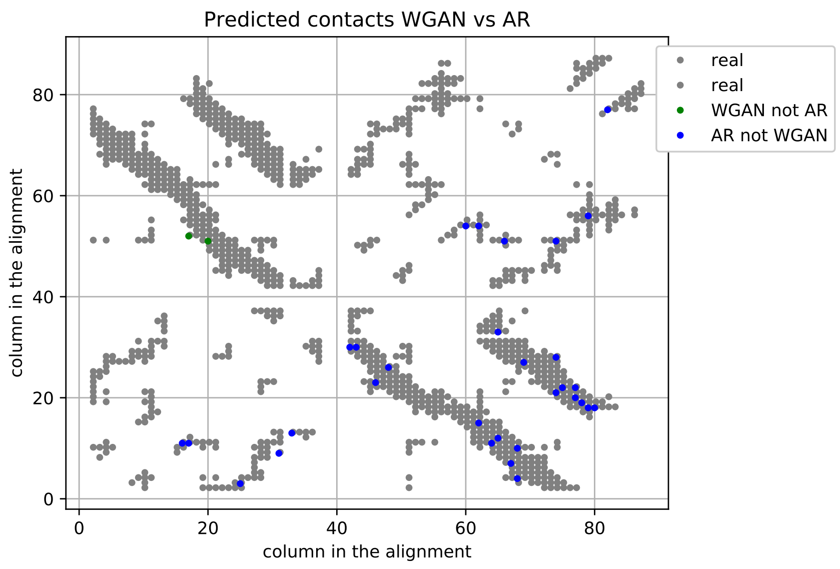

Figure 4.20: Density plot with sensitivity>=0.9 for PF00027 dataset. Real contacts (grey), true contacts predicted with WGAN but not with AR (green), true contacts predicted with AR but not with WGAN (blue).

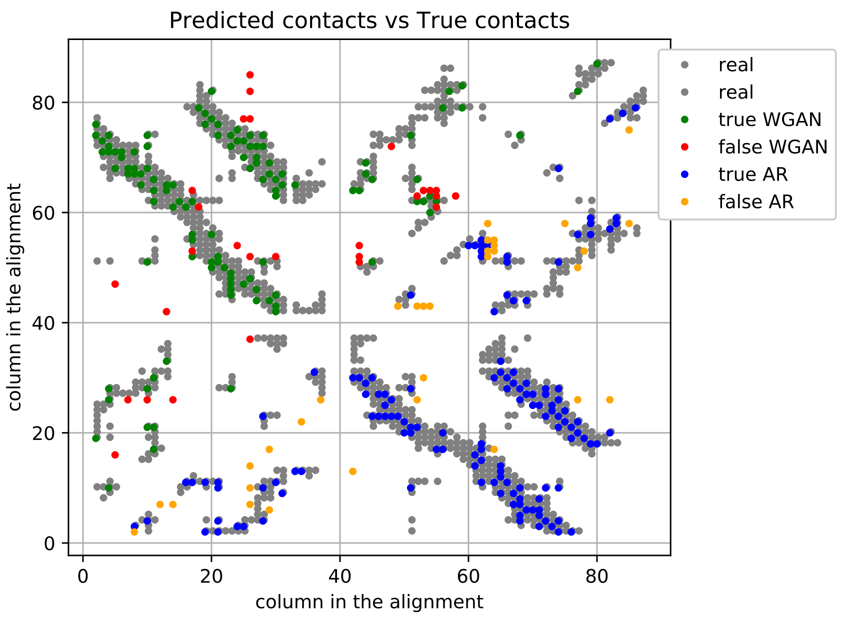

Figure 4.21: Density plot with sensitivity>=0.8 for PF00027 dataset. Real contacts (grey), true predicted contacts (WGAN in green, AR in blue), false predicted contacts (WGAN in red, AR in yellow).

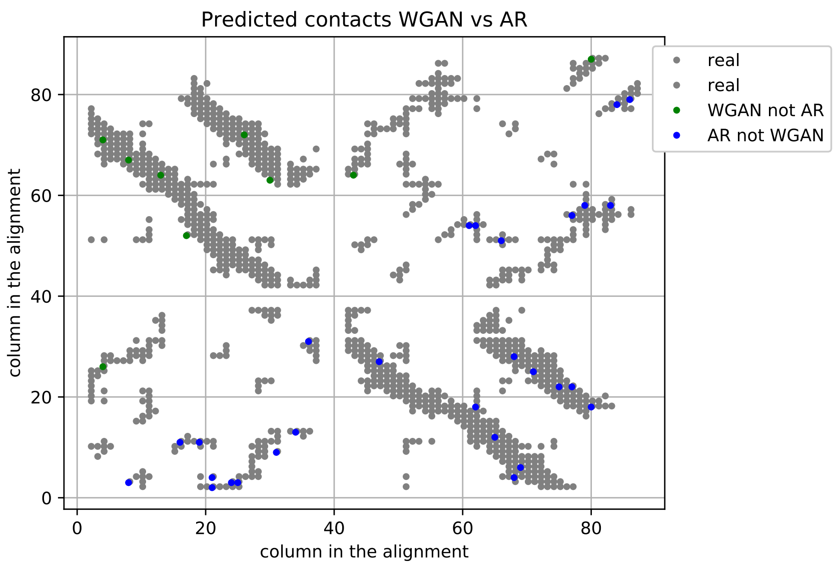

Figure 4.22: Density plot with sensitivity>=0.8 for PF00027 dataset. Real contacts (grey), true contacts predicted with WGAN but not with AR (green), true contacts predicted with AR but not with WGAN (blue).

From Figure 4.18 and Figure 4.20 we can clearly see that the AR network can predict many more true contacts with respect to the WGAN, besides all the real contacts predicted by both the approaches, both in the case of sensitivity equal to 1 and in the case of sensitivity greater than or equal to 0.9. Only when accepting a sensitivity greater than or equal to 0.8, as illustrated in Figure 4.22, the performance of the two models becomes more similar and we can notice two different sets of true predicted contacts. Therefore, only for this sensitivity level it could be worthy to try to merge the two approaches, to predict a larger set of true contacts.

Chapter 5

Conclusions

In this work, we have introduced the context of synthetic protein sequences generation and its importance in exploring the space of the possible amino acid configurations and in discovering new configurations. The goal of the work has been to generate novel protein sequences that are statistically indistinguishable from those belonging to the same family. This has been possible thanks to the availability of sequence datasets related to protein families, enabled by the advance in protein sequencing technology.

We have adopted two different neural network approaches: the one of GANs, and in particular of WGANs, and the one of autoregressive neural networks. For both approaches, we have provided a theoretical background and explained in detail the implementation. Using these two kind of models, we have carried out different experiments and reported the corresponding results, together with the specific experimental settings. We have tested the models on two different datasets, corresponding to the protein families of *Kunitz/Bovine pancreatic trypsin inhibitor* (PF00014 dataset) and *Cyclic nucleotide-binding* (PF00027 dataset). The results of the protein sequences generation task have been evaluated by means of one-point and two-point correlation plots, and a sensitivity plot. On the one side, correlation plots have allowed us to check if the empirical frequency counts of the amino acids in the initial dataset and in the generated dataset were similar; on the other side, sensitivity plots have led us to investigate the potentiality of the generated data to predict the contact map of the corresponding protein family, and therefore to get insight on its three-dimensional folded structure.

The results of these analyses have shown that both the implemented generative models, i.e. WGANs and AR networks, are capable of generating protein sequences that are statistically similar to the ones in the original dataset of their family. This has been highlighted especially by the correlation plots, which have reported a correlation between the amino acid frequency counts of the initial dataset and the generated dataset for all the experiments. The sensitivity plots have shown sensitivity curves for the generated datasets that had the same shape of the curve of the original dataset, thus indicating that both the utilised approaches actually learn something about the contact map of a protein family, and so they have the potential to replicate also its three-dimensional folded structure.

The experiments with WGANs have also indicated that the ELU activation function speeds up the training of the network, still reaching similar or even higher performance, with respect to the case of Leaky ReLU activation. Moreover, also reducing the dimensionality of the latent space has improved the WGAN training. Overall, with AR networks we have obtained a performance slightly worse than the one of WGANs on the PF00014 dataset, for what concerns the sensitivity plot; however, when considering only sensitivity equal to 1, AR networks have been able to predict more true contacts with respect to WGANs. On the PF00027 dataset, instead, AR networks have outperformed WGANs, showing higher potential to capture the three-dimensional folded structure of the protein family.

The training times have been very different for AR networks and WGANs: for both the datasets, WGANs have required a training of several days, while AR have required only some hours, thus being more suitable in case of training time constraints.

Bibliography

[1] J. McMurry. *Organic chemistry.* Belmont, CA: Brooks/Cole, Cengage Learning, 2012. ISBN: 978-0840054449 (cit. on p. 2).

[2] Wikimedia Commons. *File:Protein structure (full).png.* Author: Thomas Shafee. Source: Own work. [Online; accessed 28-September-2020]. Nov. 2016. URL: `https://commons.wikimedia.org/wiki/File:Protein_structure_(full).png` (cit. on p. 3).

[3] Creative Commons. *Attribution 4.0 International (CC BY 4.0).* `https://creativecommons.org/licenses/by/4.0/deed.en`. [Online; accessed 28-September-2020] (cit. on p. 3).

[4] S. Cocco, C. Feinauer, M. Figliuzzi, R. Monasson, and M. Weigt. «Inverse statistical physics of protein sequences: a key issues review». In: *Reports on Progress in Physics* 81.3 (2018), p. 032601. DOI: `https://doi.org/10.1088/1361-6633/aa9965` (cit. on p. 2).

[5] N. Anand and P. Huang. «Generative modeling for protein structures». In: *Advances in Neural Information Processing Systems 31.* Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc., 2018, pp. 7494–7505 (cit. on p. 4).

[6] N. Killoran, L. J. Lee, A. Delong, D. Duvenaud, and B. J. Frey. «Generating and designing DNA with deep generative models». In: (2017). arXiv: `1712.06148`. URL: `http://arxiv.org/abs/1712.06148` (cit. on p. 4).

[7] A. Gupta and J. Zou. «Feedback GAN (FBGAN) for DNA: a Novel Feedback-Loop Architecture for Optimizing Protein Functions». In: (2018). arXiv: `1804.01694 [q-bio.GN]` (cit. on p. 4).

[8] K. P. Murphy. *Machine Learning: A Probabilistic Perspective.* Cambridge, Massachussets 02142: The MIT Press, 2013. ISBN: 978-0262018029 (cit. on p. 4).

[9] J. Langr and V. Bok. *GANs in Action: Deep learning with Generative Adversarial Networks.* Manning Publications, 2019. ISBN: 978-1617295560 (cit. on pp. 7, 9, 15).

[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. «Generative Adversarial Nets». In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. Curran Associates, Inc., 2014, pp. 2672–2680 (cit. on p. 7).

[11] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. «Improved Training of Wasserstein GANs». In: *Advances in Neural Information Processing Systems 30*. Ed. by Guyon. I., U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 5767–5777 (cit. on pp. 7, 8).

[12] L. Weng. «From GAN to WGAN». In: *CoRR* abs/1904.08994 (2019). arXiv: `1904.08994`. URL: `http://arxiv.org/abs/1904.08994` (cit. on p. 7).

[13] S. Ruder. «An overview of gradient descent optimization algorithms». In: *CoRR* abs/1609.04747 (2016). arXiv: `1609.04747`. URL: `http://arxiv.org/abs/1609.04747` (cit. on p. 7).

[14] M. Arjovsky, S. Chintala, and L. Bottou. «Wasserstein Generative Adversarial Networks». In: ed. by D. Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, Aug. 2017, pp. 214–223 (cit. on p. 8).

[15] O. Triebe, N. Laptev, and R. Rajagopal. «AR-Net: A simple Auto-Regressive Neural Network for time-series». In: *arXiv preprint arXiv:1911.12436* (Nov. 2019). arXiv: `1911.12436 [cs.LG]` (cit. on p. 9).

[16] D. Wu, L. Wang, and P. Zhang. «Solving Statistical Mechanics Using Variational Autoregressive Networks». In: *Physical review letters* 122.8 (2019), p. 080602. DOI: `https://doi.org/10.1103/physrevlett.122.080602` (cit. on p. 10).

[17] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. «Pixel Recurrent Neural Networks». In: (2016). arXiv: `1601.06759`. URL: `http://arxiv.org/abs/1601.06759` (cit. on p. 10).

[18] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. «PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications». In: (2017). arXiv: `1701.05517 [cs.LG]` (cit. on p. 10).

[19] S. Reed, A. van den Oord, N. Kalchbrenner, V. Bapst, M. Botvinick, and N. D. Freitas. «Generating Interpretable Images with Controllable Structure». In: 2017 (cit. on p. 10).

[20] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Grave, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. «WaveNet: A Generative Model for Raw Audio». In: *CoRR* abs/1609.03499 (2016). arXiv: `1609.03499`. URL: `http://arxiv.org/abs/1609.03499` (cit. on p. 10).

[21] H. Larochelle and I. Murray. «The Neural Autoregressive Distribution Estimator». In: ed. by G. Gordon, D. Dunson, and M. Dudík. Vol. 15. Proceedings of Machine Learning Research. JMLR Workshop and Conference Proceedings, Apr. 2011, pp. 29–37 (cit. on p. 11).

[22] K. Gregor, I. Danihelka, A. Mnih, C. Blundell, and D. Wierstra. «Deep AutoRegressive Networks». In: ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 2. Bejing, China: PMLR, June 2014, pp. 1242–1250. URL: `http://proceedings.mlr.press/v32/gregor14.html` (cit. on p. 11).

[23] M. Weigt, R. A. White, H. Szurmant, J. A. Hoch, and T. Hwa. «Identification of direct residue contacts in protein-protein interaction by message passing». In: *Proceedings of the National Academy of Sciences of the United States of America* 106.1 (2009), p. 67. DOI: `https://doi.org/10.1073/pnas.0805923106` (cit. on pp. 11, 12).

[24] R. A. White, H. Szurmant, J. A. Hoch, and T. Hwa. «Features of Protein–Protein Interactions in Two-Component Signaling Deduced from Genomic Libraries». In: *Methods in enzymology* 422 (Feb. 2007), pp. 75–101. DOI: `10.1016/S0076-6879(06)22004-4` (cit. on p. 11).

[25] B. Lunt, H. Szurmant, A. Procaccini, J. Hoch, T. Hwa, and M. Weigt. «Inference of Direct Residue Contacts in Two-Component Signaling». In: *Methods in enzymology* 471 (Dec. 2010), pp. 17–41. DOI: `10.1016/S0076-6879(10)71002-8` (cit. on p. 12).

[26] F. Morcos et al. «Direct-coupling analysis of residue coevolution captures native contacts across many protein families». In: *Proceedings of the National Academy of Sciences of the United States of America* 108.49 (2011), E1293. DOI: `https://doi.org/10.1073/pnas.1111471108` (cit. on pp. 12, 23).

[27] S. El-Gebali et al. «The Pfam protein families database in 2019». In: *Nucleic Acids Research* 47.D1 (Oct. 2018), pp. D427–D432. DOI: `10.1093/nar/gky995` (cit. on pp. 13, 14).

[28] «IUPAC-IUB Joint Commission on Biochemical Nomenclature (JCBN). Nomenclature and symbolism for amino acids and peptides. Recommendations 1983». In: *The Biochemical journal* 219.2 (1984), p. 345. DOI: `10.1111/j.1432-1033.1984.tb07877.x` (cit. on pp. 13, 14).

[29] *Pfam database. Family: Kunitz_BPTI (PF00014).* `https://pfam.xfam.org/family/PF00014`. [Online; accessed 09-September-2020] (cit. on p. 14).

[30] *Pfam database. Family: cNMP_binding (PF00027).* `https://pfam.xfam.org/family/PF00027`. [Online; accessed 09-September-2020] (cit. on p. 14).

[31] *PyTorch.* `https://pytorch.org`. [Online; accessed 09-September-2020] (cit. on p. 15).

[32] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. «Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)». In: (2015). arXiv: `1511.07289 [cs.LG]` (cit. on p. 16).

[33] *HMMER: biosequence analysis using profile hidden Markov models* (cit. on p. 23).

[34] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. «The Protein Data Bank». In: *Nucleic Acids Research* 28.1 (Jan. 2000), pp. 235–242. DOI: `10.1093/nar/28.1.235` (cit. on p. 24).

[35] Y. Yazici, C.-S. Foo, S. Winkler, K.-H. Yap, and V. Chandrasekhar. «Empirical Analysis of Overfitting and Mode Drop in GAN Training». In: (2020). arXiv: `2006.14265 [cs.LG]` (cit. on p. 30).

[36] B. Adlam, C. Weill, and A. Kapoor. «Investigating Under and Overfitting in Wasserstein Generative Adversarial Networks». In: (2019). arXiv: `1910.14137 [stat.ML]` (cit. on p. 30).