UNIVERSITÀ DEGLI STUDI FIRENZE

Università degli Studi di Firenze
Dipartimento di Ingegneria dell'Informazione (DINFO)
Corso di Dottorato in Ingegneria dell'Informazione
Curriculum: Automatica, Ottimizzazione e Sistemi Complessi

Machine Learning applications in Science

Candidate
Lorenzo Buffoni

Supervisors
Prof. Duccio Fanelli

Prof. Filippo Caruso

Prof. Fabio Schoen

Prof. Michele Campisi

PhD Coordinator
Prof. Fabio Schoen

ciclo XXXIII, 2017-2020

Università degli Studi di Firenze, Dipartimento di Ingegneria dell'Informazione (DINFO).

A Francesca.

All models are wrong, but some are useful.
- George Box

Acknowledgments

I would like to acknowledge the efforts and inputs of my supervisors, Prof. Duccio Fanelli, Prof. Filippo Caruso, Prof. Fabio Schoen and Prof. Michele Campisi and all my colleagues at CSDC an QDAB who were of great help during my research. In particular my thanks go to Sara Nicoletti and Iuhsan Adam who have been in this journey with me for all these three years. I would also like to thank Walter Vinci and all the team at D-Wave for their support and hospitality during my stay at D-Wave Systems, with funding being provided by "Fondazione Angelo della Riccia". Similarly I would like to thank Tommaso Biancalani and his team at BROAD Institute for the opportunities and knowledge they have provided. One final acknowledgment goes to some people which I've been privileged to have as mentors, Andrea Trombettoni, Stefano Gherardini, Stefano Ruffo, Nicola Dalla Pozza, in addition to all my collaborators/coauthors. This thesis would have been rather empty if there had not been their support.

Contents

Chapter 1

Introduction

Machine learning (ML) [1–4] is a broad field of study, with multifaceted applications of cross-disciplinary breadth. ML ultimately aims at developing computer algorithms that improve automatically through experience. The core idea of artificial intelligence (AI) technology is that systems can learn from data, so as to identify distinctive patterns and make consequently decisions, with minimal human intervention. The range of applications of ML methodologies is extremely vast [5–8], and still growing at a steady pace due to the pressing need to cope with the efficiently handling of big data [9]. In parallel to the rise of ML techniques in industrial applications, scientists have increasingly become interested in the potential of ML for fundamental research, for example in physics, biology and engineering. To some extent, this seems like a natural step, since both ML algorithms and scientists share some of their methods as well as goals. The two fields are both concerned about the process of gathering and analyzing data to design models that can predict the behaviour of complex systems. However, they are fundamentally different in the way they realize their goals. On the one hand, Science aims at understanding the mechanisms of Nature, and uses prior knowledge and intuition to inform the models. On the other hand, ML mostly does the opposite: models are agnostic and the machine provides the knowledge by extracting it from data. Although they are often powerful, the resulting models are notoriously known to very opaque to our understanding. Machine Learning tools in Science are therefore welcomed enthusiastically by some, while being eyed with suspicions by others, albeit producing surprisingly good results in some cases.

In this thesis we will argue, using practical cases and applications, that the communication between these two fields can be not only beneficial but perhaps necessary. On the one hand Machine Learning will find benefit from the interaction with the scientific community (e.g. engineers and physicists) in order to find new ways to speed up computations by breakthroughs in physical hardware, as we argue in chapters 4 and 5. Probably ML can also benefit from the knowledge that science has in dealing with complex systems, let it be to understand better the process of learning or to improve existing models as we will see in chapter 3. On the other hand scientists have lots of experiments that do generate incredible amounts of data and ML could be a great tool to analyze those and make predictions as in chapter 2, also it can be used with good results to *control* the experiments itself as in chapter 5. On top of that, data visualization techniques and other schemes borrowed from ML can be of great use to theoreticians to have better intuition on the structure of complex manifolds or to make simulate complex theoretical models.

Outline

The text is divided in chapters, each one focusing on a particular application. The chapter are ordered by going from the most straightforward applications of existing ML to scientific data to the topics which are only subject of theoretical research. In particular, each chapter will cover one possible interplay between Machine Learning and the physical sciences. To fulfill this goal the chapters are organized as follows:

- In chapter 2 we will see how Machine Learning can be used to improve experimental techniques, in this particular case coming from a biology experiment. We will see how a well suited computational pipeline based on existing Machine Learning techniques can be used to map genomics data into space in order to visualize gene expression patterns in organs. We will see how ML can give a sizeable improvement over the existing experimental techniques and help researchers gaining previously inaccessible insights. Indeed, the application of existing ML models into experimental analysis is the most common application in the scientific domain and has already proven its effectiveness in multiple domains.

- In chapter 3 we devise a new interpretation of learning, based on the spectral properties of Neural Networks. This insight given by network theory will enable us to train a model with a fraction of the parameters employed by the standard Machine Learning techniques, while returning comparable performances. Also this type of spectral analysis could help to better understand and interpret the learning process of Neural Networks.

- In chapter 4 we introduce the possibility of accelerating existing ML models, in this case a Variational Autoencoder, using cutting-edge computing devices. In particular we will see how using a quantum computer to perform sampling in the latent space of a Variational Autoencoder can lead to improvements to the model and holds promises for a possible quantum advantage. This is a first step in exploiting the power of quantum computing to improve Machine Learning algorithms in an *hybrid* quantum-classical setting, this is also one of the first hybrid models reaching state-of-the-art performance on a real world task.

- Finally, in chapter 5 we will implement a simple supervised learning algorithm fully in the quantum domain to give a final example on how is possible to leverage the properties of quantum systems in Machine Learning. We will also deploy this algorithm on different experimental devices to see its robustness on real-world noisy environments. Then we will see how we can apply Reinforcement Learning to control the dynamics of a quantum random walker in a maze, improving the transfer rate of the walker from the entry to the exit of the maze with just some simple actions. These topics are examples of the theoretical research that is ongoing at the edge between ML and the physical sciences. These models are far from being scalable and useful in practical cases, nevertheless, they can offer interesting proof of principle results to move the boundary of scientific applications of ML a bit further.

Before going into the core of the thesis it is useful, for the sake of readability, to give a brief introduction to some basic concepts in Machine Learning. This will give the reader (independently from its scientific background) a common ground to start, while the more technical details and concepts about the specific applications will be discussed in each chapter separately.

Concepts in Machine Learning

We start by introducing the various types of learning, which can be supervised, unsupervised or reinforced. In this manuscript we will touch all of them at different points:

- **Supervised learning**: In supervised learning [1, 10] we deal with an annotated dataset $\{(x_i, y_i)\}_{i=1}^N$. Each element x_i is called an input or feature vector. It can be the vector of pixel values of an image or a feature such as height, weight and gender and so on. All input data x_i belonging to the same dataset have the same features (with different values). The label y_i it is the ground truth upon which we build the knowledge of our learning algorithm. It can be a discrete class in a set of possible objects or a real number, representing some property we want to predict, or even some complex data structure. For example if you want to build a spam classifier a good choice of labels can be $y_i = 1$ (spam) or $y_i = 0$ (not spam). The goal of a supervised learning algorithm is to use the dataset to produce a model that, given an input vector x, can predict the correct label y.

- **Unsupervised learning**: In unsupervised learning [11–13] the dataset is a collection of unlabeled vectors $\{x_i\}_{i=1}^N$. The goal of unsupervised learning is to take this input vector and extract some useful property from the single data or the overall data distribution of the dataset. Examples of unsupervised learning are clustering, where the predicted property is the cluster assignment, dimensionality reduction, where the distribution of data is mapped in a lower dimensional manifold, outlier detection where the property predicted is the "typicality" of the data with respect to its distribution and generative models, where we want to learn to generate new points from the same distribution of the dataset.

- **Reinforcement learning**: The subfield of reinforcement learning [5] assumes that the machine "lives" in an environment and can probe the state of the environment as a feature vector. The machine can perform different actions at different states, ultimately leading to different rewards. The goal of this machine (or agent) is to learn a policy. A policy is a function that associates to a particular feature vector, representing the state of the environment, the best action to execute. The optimal policy maximizes the expected average reward. Reinforcement learning has been widely employed in scenarios where decision making and long-term goals are crucial, for example in playing chess, controlling robots or logistics.

In this intricate landscape of problems that fall under the umbrella of “Machine Learning” there are lots of different models each one best suited to different tasks. Some notable examples of such models are Support Vector Machines [14] and decision tree learning [15] to solve classification tasks, k-nearest neighbors [16] for clustering and UMAP [17] for dimensionality reduction. All these algorithms share some building blocks that are the basics of any learning algorithm. The three fundamental parts of a learning algorithm are (i) some training data (ii) a loss (or objective) function (iii) an optimization routine to minimize the loss function on the training data. It is important now to point out that the minima that are achieved in learning are often local, that is partially due to the fact that optimization landscapes of learning problems are non-convex and usually quite complex, but also to the fact that in learning our goal is often to have a model with good generalization properties. That means that robustness of the model over a broader set of new data is often more appealing than a better but less generalizable one, thus favoring local minima with respect to global ones. Now we will focus our attention on describing arguably the most popular Machine Learning model, namely Neural Networks (NNs). It is important to define here some concepts and general properties about NNs because they will be used in all of the subsequent chapters. From a mathematical point of view, a NN is a parametric function defined as

$$y = f_{NN}(x) \tag{1.1}$$

the function f_{NN} has the form of a nested function reflecting the *layer* structure of the network. For example, a network with three layers will read

$$y = f_{NN}(x) = f_3(f_2(f_1(x))), \tag{1.2}$$

where the function at the k^{th} layer has the form

$$f_k(x) = g_k(W_k x + b_k), \tag{1.3}$$

Where W_k is a matrix of trainable parameters (the weights of the network) and b_k is a vector of trainable parameters (the biases of the network). Given an input dimension D of the vector x the matrix W_k has shape $M \times D$ and the vector b_k has dimension M, resulting in an output vector $z_k = W_k x + b_k$ of dimension M. The final piece of the puzzle is the function $g_k(z_k)$ that is called the *activation function* or nonlinearity. Indeed this function has to be a nonlinear function that mimics the activation (or spiking) that happens in biological neurons. Some of the most popular activation functions used in NNs are the sigmoid function

$$sigmoid(z) = \frac{1}{1 + e^{-z}}, \tag{1.4}$$

and its variant, the hyperbolic tangent $tanh(z)$. Another popular choice is the Rectified Linear Unit (ReLU) namely:

$$relu(z) = \begin{cases} 0 & if\ z < 0 \\ z & otherwise \end{cases} \tag{1.5}$$

that despite being *almost* linear is nonlinear enough to build a working NN. With this notation set, is now important to notice that all the operations in a NN are continuous and differentiable (with the notable exception of the zero point in the ReLU) meaning that we can compute the gradient of the output with respect to each one of the parameters $\{W_k, b_k\}$ with $k = 1, ..., l$. So, for example, in a supervised learning problem we would have the tuples $\{(x_i, y_i)\}_{i=1}^N$ defining our training set and a NN that is giving an output $\tilde{y}_i = f_{NN}(x_i)$ which we would like to be as close as possible to the label y_i. Following the recipe valid for all ML models, in order to train this NN we thus have to define a cost function, as an example we will take the Mean Squared Error (MSE):

$$MSE(y, \tilde{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \tilde{y}_i)^2, \tag{1.6}$$

and then optimize it with some strategy. In this case, since we know all the derivatives of the cost function with respect to the trainable parameters, we can simply optimize the function by gradient descent updating the weight matrix W_k and the bias vector b_k at each layer of our NN $k = 1, ..., l$. This iterative step of gradient descent is often referred to as *backpropagation*. Once we have trained the model with a sufficient amount of training data and for a sufficiently long amount of backpropagation steps, our NN will be in a local minimum of the cost function. At this point the model is trained and should output predictions $\tilde{y}$ which are mostly correct. The NN is the artificial analog of a biological network of spiking neurons (e.g. our brain), and it is composed by stacking together multiple artificial neurons, which are called perceptrons. Perceptrons are the "computational units" of NN as neurons are for the brain, and they are capable to perform all the mathematical operations we just described. Now it is also useful to have in mind a graphical representation of the perceptron, that can be seen in Fig.1.1 where a single artificial neuron is presented. By stacking neurons in layers and stacking lay-

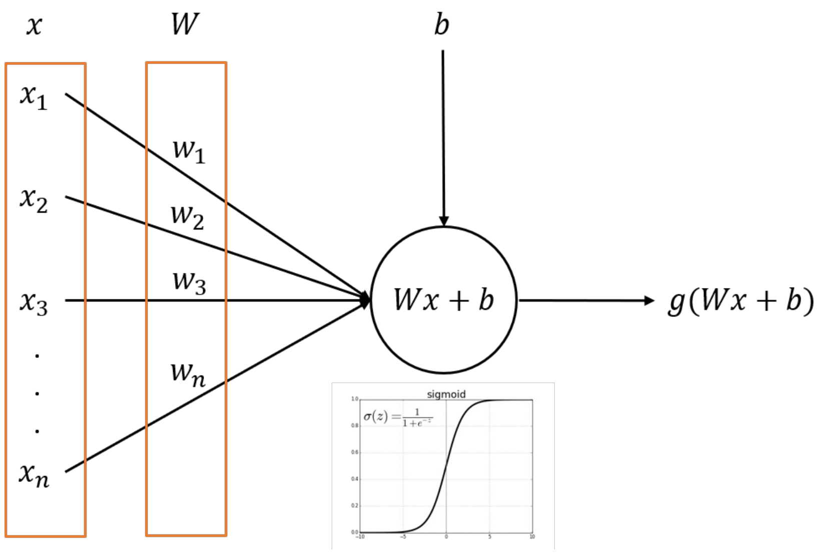


Figure 1.1: Graphical representation of a single neuron (also called perceptron) of a NN. This is the computational building block of a NN composed of an input x, some trainable parameters W (weights) and b (bias), and a nonlinear activation function $g(\cdot)$ (e.g. the sigmoid as represented here) on the output.

ers one after another we obtain the full NN structure in Fig.1.2. Since more complex data require more layers this branch of ML is often referred to as Deep Learning and networks used are called Deep Neural Networks (DNNs) where the adjective "deep" refers to the depth of the network itself (i.e. the number of layers). This is the most basic NN architecture, known as fully connected or dense network. There are a whole bunch of different architecture such as Convolutional Neural Networks (CNNs) in which the trainable parameters are the values of some filters to be convoluted with the input

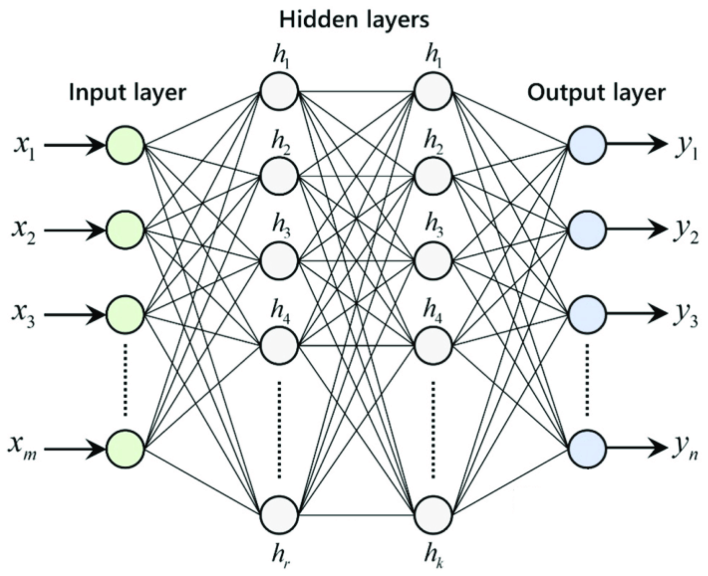


Figure 1.2: Example of a full NN obtained by stacking layers of single neurons (colored circles). Each neuron implements the same operations described in Fig.1.1 where the trainable parameters are the weights, represented by the links in this picture. The input layer reads the feature vectors from the dataset, the hidden layers (i.e. the layers which are neither input nor output ones) process the information, and the output layer gives the final prediction used to compute the cost function and subsequently perform backpropagation.

image, or Recurrent Neural Networks (RNNs) that are specifically built to deal with time series and have some memory mechanisms in it. Indeed a consistent part of the research done in Deep Learning by computer scientists is aimed at inventing new network architectures that perform better on different problems. A detailed review and explanation of the different types of NNs is out of the scope of this thesis and we will refer to the literature for more details [4, 10, 18]. One more thing that is important to specify, are the computational tools by which these algorithms are implemented. Albeit the mathematics behind NNs is rather simple, at a first glance implementing the backpropagation algorithm (i.e. computing the derivatives of the cost function with respect to each one of the trainable parameters) seems like a lot of work, even more when confronted with the typical number of parameters of a NN being in the range of $10^5 - 10^8$. In fact, ad-hoc libraries to perform *automatic differentiation* have been developed, the most popular ones being TensorFlow and PyTorch. These libraries allow the user to simply specify the NN architecture and let the library to do the heavylifting of computing the gradients and updating the parameters resulting in a much easier implementation. Now that we have introduced these concepts that are central to the following chapters, we can move onto the applications of ML in Science that are the core of this thesis.

Chapter 2

Learning on experimental data

In this chapter, we will introduce arguably the most common application of ML to scientific research, that is performing ML pipelines on experimental data. Data coming from physics, biology or chemistry experiments are almost always complex in nature and require advanced statistical tools to be processed, which is exactly what algorithms of Pattern Recognition and Machine Learning [1–3] were designed to do from the beginning. It thus seems natural that more and more researchers are starting to employ such algorithms to get better, faster pipelines to process their data. As an example we will illustrate one of these pipelines we have developed [19], in collaboration with the BROAD Institute of MIT and Harvard, to gain a huge advantage in gene throughput over existing techniques to reconstruct spatial maps of sequencing data.

2.1 Mapping sequencing data into space

Single-cell/single-nuclear RNA sequencing technology (sc/snRNA-seq) is a technique that allows for identification of transcriptional clusters at single cell level, which is instrumental in revealing cell types [20], developmental trajectories [21] and gene programs [22] that are present in a certain tissue sample. Despite the paramount attention recently received from this technology [23–25], we still lack the ability of precisely reconstructing the spatial location of cells from sc/snRNA-seq data. Indeed, the tissue samples are dissolved in a liquid solution to enable separation of single cells into single droplets that are later processed to extract the genetic information of the

cell. This process, while enabling single-cell accuracy, destroys all the spatial information about the location of the cells in the tissue. In contrast, various spatial technologies allow for in-situ measurements [26–28] of transcriptional clusters, thus providing finer spatial localization, but suffer from either lower throughput or lower resolution compared to sc/snRNA-seq counterparts. Therefore, we would like to somehow harmonize in-situ data with sc/snRNA-seq data combining various forms of spatial information from the brain region from which snRNA-seq data have been collected, including histological images, public atlases and in-situ data. We will see how to build a Deep Learning pipeline in order to automatically retrieve in-situ data from a region of interest in an histology image then we will define an objective function based on these two complementary types of data, the optimization of which leads to a spatial alignment of the snRNA-seq data.
Our goal is thus to learn a probabilistic spatial alignment of cells, via which we reconstruct patterns of cell clusters or gene expressions by transferring snRNA-seq annotations onto space. Our method builds up on previous works [23, 24], although we perform cell mapping by globally considering the cell spatial context (as opposed to local cell-by-cell integration methods [23]), instead we do not add hypotheses on gene expression patterns (such as continuity, as done in [24]) and we map by accounting for both histology and gene expression. Our strategy is based on constructing a suitable objective function by using cell density and spatial gene expression which we estimate from histology, existing atlases, in-situ data or any combination of these. By optimizing the objective function, we learn an alignment such that cell density and gene expression of the mapped cells are as similar as possible to those estimated from spatial data. We validate the alignment by showing that we recover known cell type patterns and predict gene expression of holdout genes. We will map cells using the public Allen Mouse Brain Atlas [29] ($3k$ genes at $200\mu m$ resolution): by introducing a deep learning-based registration and mapping pipeline, we reconstruct a spatial map of the primary motor area with $30k$ genes at single-cell resolution, revealing spatial gene expression patterning beyond current limitation of in-situ technologies.

Our strategy, as depicted in Fig.2.1, consists of locating the region of dissection onto the Allen Common Coordinate Framework [29] (Allen CCF) so as to query the mouse brain atlas for building the objective. To do so, we first introduce a deep learning registration pipeline. By borrowing methods from face recognition, we start by learning a latent space using a Siamese

Neural Network [30] model, a particular type of NN that we will describe in detail in the following sections, trained on mouse brain images. We train the model so that each image is encoded according to salient anatomical landmarks, whereas technical properties such as illumination or staining are factored out. Indeed, we confirm that the learned latent space displays a one-dimensional manifold structure, where the head of the manifold contains images from the olfactory bulb (at the front of the mouse brain), and the tail, images from cerebellum (at the bottom of the mouse brain). The model predicts the image from the Allen CCF at the same coronal depth of our histological image. Predictions are validated by checking consistency across the whole training set, and by inspection. We use this model to retrieve the image from the Allen CCF onto which we register our histological image. Next, we apply semantic segmentation [31], and segment five classes on our histological image: background, cortex, cerebellum, white matter and other grey matter. The goal of segmentation is to generate a custom mask for our images using the same color scheme adopted by the Allen Atlas. Once the histological image(s) are registered, we query two atlases to build the objective: from the Allen Atlas, we estimate gene expression at spatial resolution $200\mu m$; from the Blue Brain Cell Atlas [32] we compute the expected cell density in each spatial voxel; finally, we compute an anatomical map from the Allen Atlas, which we use post mapping to assess on which anatomical region each cell has been mapped. To perform mapping, we learn a mapping matrix denoting the probability of finding each cell into each spatial voxel. We show mapping predictions for cell types across the three regions of interest (ROIs), which results consistent across each other and with our expectations.

In conclusion, gene expression exhibits a variety of spatially-organized patterns whose knowledge is central to unravel biological function. Spatially resolved transcriptomic data provide an opportunity to reveal such patterns, but are currently limited by spatial resolution or gene throughput. We showed that by harmonizing snRNA-seq data with in-situ data, some of these limitations are removed. Our work focused on mouse brain tissue although the mapping method is in principle applicable to any organ. In contrast, our registration pipeline requires a CCF and is therefore applicable to a few organs at present.

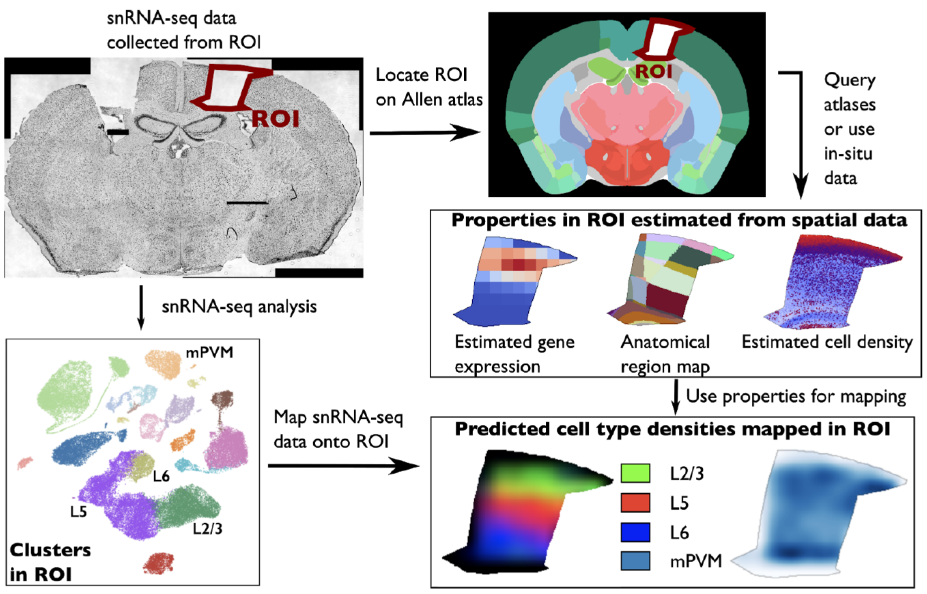

snRNA-seq data collected from ROI
ROI
Locate ROI on Allen atlas
ROI
Query atlases or use in-situ data
snRNA-seq analysis
Properties in ROI estimated from spatial data
Estimated gene expression
Anatomical region map
Estimated cell density
Use properties for mapping
mPVM
L6
L5
L2/3
Clusters in ROI
Map snRNA-seq data onto ROI
Predicted cell type densities mapped in ROI
L2/3
L5
L6
mPVM

Figure 2.1: Schematics of our computational pipeline. We start from an histological image (top-left figure) in which we highlight a ROI indicating where snRNA-seq data were collected. We compute cell clusters in snRNA-seq data using a conventional pipeline. A registration pipeline is used to locate the ROI onto the mouse common coordinate framework (top-right figure). Through registration, we estimate gene expression from the Allen Mouse Brain Atlas, an anatomical region map and cell density map. We leverage the estimated properties to map snRNA-seq data to space by solving an optimization problem. In the bottom-right figure, we show a few model predictions showing layering of cortical neurons and uniform distribution of mPVM cells [19]

2.2 Computational pipeline

To build our dataset, we assembled an integrated atlas of the somatomotor area of the healthy adult mouse brain using publicly available atlases. A total of $160,000$ snRNA-seq profiles were collected from three dissected regions of interest (ROIs) in the somatomotor area, using a lab procedure [33] where nuclei are isolated from a biopsy punch in a frozen dissected region. To help relate this region to the known anatomy, we also obtained stained histological sections on the punched section, which are approximately $200\mu m$ deep.

In many cases, only histological data is directly available for the specimens collected as part of single cell atlases, but those can serve as a bridge to pre-existing atlases, with measured in situ hybridization (ISH) data, and rich anatomical annotations in the as in the case of the Allen CCF. Using these data should allow relating cellular features (e.g., gene expression, cell types) to the histological or organ scale, especially in the brain. However, typical methods from computer vision for registration of medical images [34, 35] require human supervision, such as identification of a few corresponding anatomical landmarks in experimental and atlas images. Such supervision, albeit minimal, has prevented complete automation so far. A common strategy to remove supervision uses machine learning for identifying the few key landmarks required in registration, as has been shown in [36]. However, this method is not suitable for images that are torn or contain holes, for example, if tissue has been first dissected for profiling sc/snRNA-seq data as in our case.

To this end, we first developed a module to connect across scales by registering histology/spatial data on an anatomically annotated CCF, such as the Allen CCF for the adult mouse brain. As an alternative to methods that either require supervision or intact tissue, we combine a Siamese neural network model with a semantic segmentation algorithm to produce full segmentation masks of anatomical images. The Siamese network model builds a latent space which allows a uniform encoding irrespective of technical artifacts in the images, such as the presence of holes in dissected regions (from which cells or nuclei were collected). The semantic segmentation model produces a segmentation mask with a color scheme that is compatible with the Allen ontology. Because we produce a mask with matching colors, we can then register the images automatically as we do not need to provide corresponding landmarks; instead the anatomical regions in the mask are the landmarks.

First, we learned a latent space using a Siamese Neural Network [30] model trained on mouse brain images. We trained the model so that each image was encoded according to salient anatomical landmarks, whereas technical properties such as illumination or staining were factored out. We then used the trained model to retrieve the image from the Allen CCF onto which we register our histological image.

Next, we segmented our images to generate a custom mask for our images using the same color scheme adopted by the Allen CCF. For this, we applied semantic segmentation [31], and segmented five classes in our histological image: background, cortex, cerebellum, white matter and other grey matter. As the training set is scarce, we adopted a combination of transfer learning and heavy augmentation during training and validated it by inspecting predictions on test atlases. Finally, we combined segmentation with the Siamese model, to obtain a fully automated registration pipeline.

After we applied this anatomical mapping module to the histological images to precisely locate the region of dissection on the Allen CCF, we queried the Allen Atlas to estimate spatial gene expression at $200\mu m$ resolution and the Blue Brain Cell Atlas to compute the expected cell density in each spatial voxel. The final mapping algorithm, called Tangram, then computed an anatomical map from the Allen Reference Atlas, and used it post-mapping to estimate the anatomical region to which each cell has been mapped. We repeated this procedure for the three ROIs, and finally mapped the snRNA-seq profiles to their corresponding ROIs.

In the next section we will go into details about the implementation of each of the parts of the pipeline listed above before giving some comments on the results.

2.3 Implementation details

In this section we will focus on the implementations of the various algorithms that construct our pipeline. We will focus our attention on the first part of anatomical registration, that is because the main contribution of the author was in this particular piece of the work. However, to give context and detail we will include also the broader picture so that the reader can understand how this pipeline all comes together to fit the intended purpose.

2.3.1 Siamese Network for anatomical registration

The first goal we have is, given an histology image, to automatically register it on the Allen CCF to subsequently extract the known features of the ROI and perform the mapping of sequencing data into space. We used a Siamese Neural Network [30] model trained on mouse brain images. Siamese networks are just regular Neural Networks that share identical structure and weights (as Siamese twins). Such architectures are used to encode a complex object (e.g. an image) into a latent space to perform some operation usually in the context of one-shot-learning. For example these architectures are widely used in face recognition problems where a reference database of faces is available. With a Siamese network, one can take a new face, encode it in the latent space and find the best match in the database by just computing some distance metric and finding the nearest neighbor. Here we would like to leverage the same concept using the Allen Atlas as our face database and match it with the experimental images. For training we used images from different public datasets:

- **avg**: 1320 images/segmentation masks of coronal slices from the average template of the Allen adult mouse brain atlas at resolution $10\mu m$ (this was used as our objective Allen CCF for registration) (link).
- **ara**: 1320 images/segmentation masks of coronal slices from the Nissl template of the Allen adult mouse brain atlas at resolution $10\mu m$ (link).
- **p56c**: 132 images/segmentation masks of coronal slices from the Allen P56 coronal reference atlas (link).
- **p56d**: 504 images of coronal slices from the Allen Development Atlas P56 (link).

Training images were resized to 224×224 pixels and casted to type float32. Pixel values were rescaled in between zero and one, prior to training. All images were augmented using imgaug library with a series of deformations, noise and color transformations to prevent overfitting. Training labels are numerical coordinates indicating the spatial coronal depth (i.e. posterior) of each mouse brain image on a scale of $10\mu m$, ranging from $0 - 13200\mu m$. As an example, an image from the olfactory bulb, located at the front of the brain will have a label of, let's say, $50\mu m$. For the avg and ara datasets, labels were readily available from their tensor coordinates. Labels for the p56c and p56d datasets were also readily obtained using the AllenSDK API. We used images from two different datasets as test sets that were manually annotated:

- **brainmaps**: 111 images of coronal slices from Nissl-stained BrainMaps atlas (link), and 87 images of coronal slices from Nissl-stained BrainMaps atlas (link).
- **ish**: 30 images of coronal slices from the Allen ISH Data (link).

In designing the Siamese network model, we used a DenseNet169 [37] encoder pretrained on the ImageNet [38] dataset and open-sourced through Keras Applications. We fine-tuned the encoder by training the last convolutional layer. We added two fully connected layers on top of the encoder in order to map the extracted features to our 512-dimensional latent space. We then take the $L1$ distance between the experimental and atlas images in the encoded latent spaces of the Siamese network. A last fully connected layer was used to map the distance in latent space to the model output as represented in Fig.2.2. All fully connected layers were trained. A training

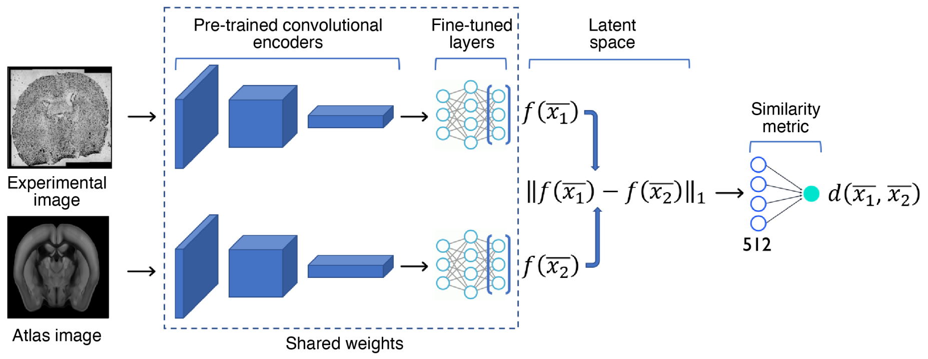


Figure 2.2: Schematic of Siamese Neural Network architecture. A pair of images is fed to two convolutional encoders, which encode them into a 512-dimensional latent space. The image pair is labeled by the spatial coordinate (i.e., coronal depth) difference between the two images.

sample consisted of two random images from the annotated datasets. The difference of the spatial depth coordinates between the two images, denoted by $\hat{d}_i$, was used as a label. For example, if the first image were at posterior (depth) $500\mu m$ and the second at a posterior $700\mu m$ the corresponding label would be $\hat{d}_i = 200$. We used as penalty the MSE between the spatial depth difference predicted by our network d_i, and the labels $\hat{d}_i$:

$$MSE(d, \hat{d}) = \frac{1}{N} \sum_{i=2}^{N} \left(d_i - \hat{d}_i \right)^2 \tag{2.1}$$

where N indicates the number of training samples. We trained the model for 50 epochs using $18k$ image couples per epoch, subdivided in batches of 16 images. After training the NN, we wanted to investigate the structure of the latent space. We thus had to employ to a dimensionality reduction algorithm in order to plot the data in 2 dimensions but preserving the topology of the original space that has dimension 512. For that, we choose the popular UMAP [17] algorithm. The learned latent space displayed a one-dimensional manifold structure as we can see in in the UMAP plot of Fig.2.3, where the head of the manifold contains images from the olfactory bulb, and the tail, images from cerebellum. To have a better prediction and also some

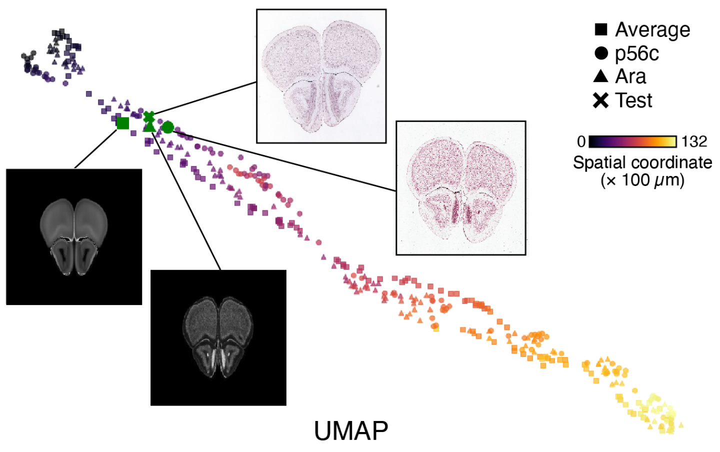


Figure 2.3: The learned latent space is a 1D-manifold ordered by spatial coordinates. UMAP plot of the encoded training images from individual atlases (legend) colored by spatial depth (color bar). Insets illustrate four anatomically similar images from three different atlases and a test image.

confidence on the prediction itself we check every image against all the Allen CCF to obtain a series of predictions as in Fig.2.4. Then we use a piecewise linear function to fit these predictions and we pick the minimum of the fit as our best estimate of the depth, this adds very little overhead to the model since we already have the latent space of the Allen CCF and improves our precision. Furthermore, if the predictions of our model are not precisely on a linear shape as in Fig.2.4, but they have flat spots, multiple minimums etc., we know at a glance that our model is having trouble with that image and its prediction should be checked manually. This last bit is a great improvement in reliability that comes at a really small computational cost. The model predicted the image from the Allen CCF at the same coronal depth of our histological images. We validated the predictions by checking consistency across the whole training set, and by visual inspection, see Fig.2.4. We also validated our model with images from which a ROI was previously removed

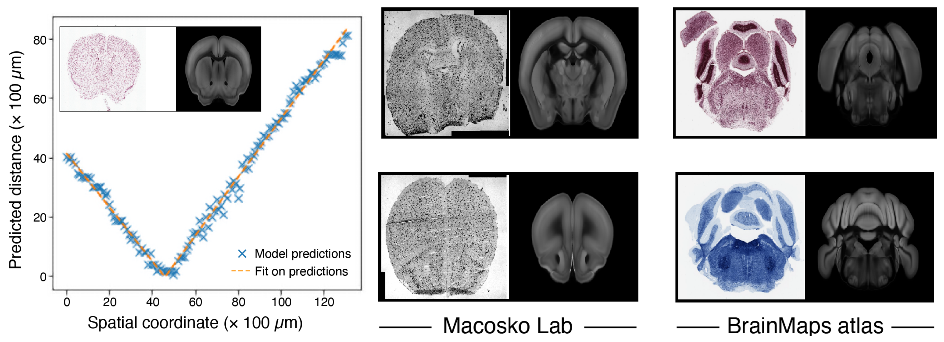


Figure 2.4: On the left: predicted spatial coordinate distance (y-axis) between a test image (inset, left panel) and each image of the training set obtained at different spatial coordinates (x-axis). Dashed orange line: piecewise linear fit on predictions. The minimum of the fit is the predicted spatial coordinate (associated image is in the inset, right panel). On the right: some test predictions from different sources. We can see that, from very different types of images (staining, noise, illumination), the model outputs predictions that are consistent with the anatomical features present in the images.

in order to sequence its cells. Indeed that is the kind of images our model will need to deal with, and for all the images tested it performed as well as the regular images. The code to reproduce these results can be found in a public version here [39]. Once we have registered our histology image onto the Allen CCF is time to estract all the available informations about the ROI of our sample.

2.3.2 Semantic segmentation

Here, we used datasets avg, ara and p56c as training sets, since masks were available. Training images were resized to 512 × 512 and casted to type float32. Pixel values were rescaled in between zero and one. Labels are superimposable segmentation masks with the same dimension of the training images. Each mask was one-hot encoded into a 5-channel tensor to annotate each pixel into five different classes: background (black), cortex (green), cerebellum (yellow), other grey matter (grey), and white matter (brown). We used colors consistent with the Allen ontology to facilitate registration. For avg and ara datasets, we used masks from the Allen CCFv3 ontology 2017 (available at this link). For the p56c dataset, we downloaded the SVG masks from the Allen Institute website, and rendered them into images. Both images and masks were augmented using the same pipeline adopted for the Siamese model. In transforming the masks, we ensured that the one-hot structure was preserved in the masks after augmentation. We used a

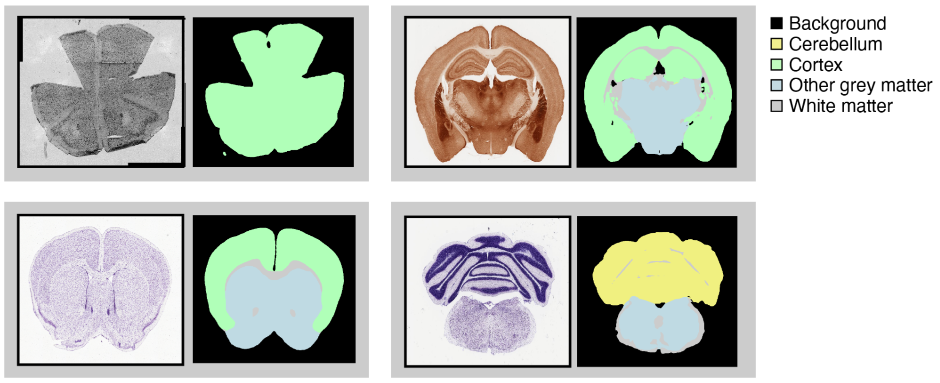


Figure 2.5: Prediction examples. Experimental images (left) and their predicted anatomical region calls (right).

semantic segmentation model from the Tensorflow Keras version of the segmentation models library. Specifically, we chose a U-NET [31] architecture with a ResNet50 [40] backbone. All weights have been randomly initialized following the He scheme, with the exception of the ResNet50 encoder which was pre-trained on ImageNet. Model was trained to optimize the superposition of the cross entropy and Jaccard index (i.e. intersection-over-union). Denoting by L such loss function, by g a ground truth image and by p the corresponding prediction of the model, the loss we used reads:

$$L(g, p) = -g \log(p) - \frac{p \cap g}{p \cup g} \tag{2.2}$$

The model last unit employs a softmax activation function, thus outputting the probability of each pixel to be in each of the five classes. By applying an argmax function, we assign each pixel to its most probable class. Finally, we relied on test-time augmentation to increase model performances: each test image was augmented twelve times, and final predictions were de-augmented and averaged. An example of the result of the Siamese network plus semantic segmentation can be seen in Fig.2.5.

2.3.3 Tangram mapping algorithm

Here we briefly illustrate the final mapping of the single-cell RNA sequencing data into space using an method we named Tangram. In the following we will use the index i for cells (i.e. snRNA-seq data), k for genes and j for spatial voxels (circular spots, pucks, etc.). From scRNA-seq we obtain a matrix S with dimensions $n_{cells} \times n_{genes}$, where n_{cells} is the number of single cells, such that $S_{ik} \geq 0$ is the expression level of gene k in cell i. In order to map, we voxelize the spatial volume at the finest possible resolution (which depends on the mapping case, e.g. $200\mu m$ when mapping with the Allen Brain Atlas), and index the voxels in an arbitrary one-dimensional fashion. We then introduce two quantities: the $n_{voxels} \times n_{genes}$ gene expression matrix G, where $G_{jk} \geq 0$ denotes the expression of gene k in voxel j (we do not assume that G and S measure gene expression using the same unit of measures), and a vector $\vec{d}$ of length n_{voxel} representing cell densities, where $0 \leq d_j \leq 1$ is the cell density in voxel j, and $\sum_j^{n_{voxel}} d_j = 1$. We aim to learn a mapping matrix M with dimension $n_{cells} \times n_{voxels}$, such that $M_{ij} \geq 0$ is the probability of cell i of being in voxel j. Therefore, we require a probability constraint $\sum_j^{n_{voxel}} M_{ij} = 1$. Our mapping strategy is probabilistic, perform a soft assignment. From the mapping matrix M, we further define two quantities: $M^T S$, the spatial gene expression as predicted by the mapping matrix, and the vector $\vec{m}$ with components $m_j = \sum_i^{n_{cells}} M_{ij}/n_{cells}$ for the predicted cell density in voxel j. Finally, we define the softmax function along the voxel axis for any given matrix $\tilde{M}$ (with dimensions $n_{cells} \times n_{voxels}$). The resulting matrix M has elements:

$$M_{ij} = softmax(\tilde{M})_{ij} = \frac{e^{\tilde{M}_{ij}}}{\sum_{l=1}^{n_{voxels}} e^{\tilde{M}_{il}}} \tag{2.3}$$

By applying the softmax, we ensure that $0 \leq M_{ij} \leq 1$ and $\sum_{j=1}^{n_{voxels}} M_{ij} = 1$. To learn the mapping matrix, we minimize the following objective function with respect to $\tilde{M}$ (note that in the objective we use $M = softmax(\tilde{M})$):

$$\Phi(\tilde{M}) = KL(\vec{m}, \vec{d}) - \sum_{k=1}^{n_{genes}} \cos_{sim}((M^T S)_{*,k}, G_{*,k}) - \sum_{j=1}^{n_{voxels}} \cos_{sim}((M^T S)_{j,*}, G_{j,*}) \tag{2.4}$$

where KL indicates the Kullback-Leibler divergence and $\cos_{sim}()$ is the cosine similarity function. The first term is the density term: we enforce that

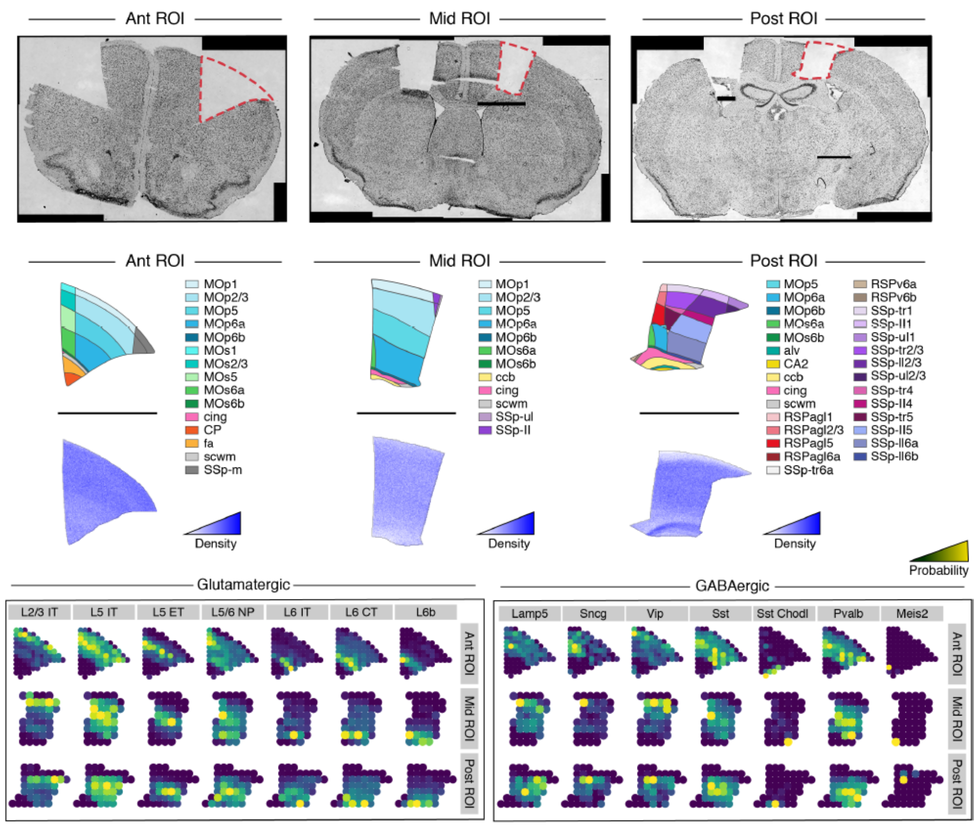


Figure 2.6: Top row: Regions of interests. Nissl-stained images of coronal mouse brain slices highlighting the three regions of interest from which snRNA-seq data from the motor area were collected. Middle row: Registration pipeline generates anatomical region and cell density maps. Anatomical region (color legend, from the Allen Common Coordinate Framework) and cell density (color bar, from the Blue Brain Cell Atlas) maps of each of the three dissected ROIs. Bottom row: Probabilistic mapping of some sample snRNA-seq data on the ROI. Probability of mapping (color bar) of each cell subset (grey label) from each of 3 major categories within each ROI (rows). We can appreciate how different genes are spatially organized in the brain layers.

the learned density distribution is as similar as possible to the expected density. The second term is the gene/voxel expression term: it enforces that, for each gene, its predicted expression over the voxels is proportional to the expected gene expression over the voxels. The third term is the voxel/gene expression term: for each voxel, the predicted gene expression needs to be proportional to the expected gene expression. We minimize the objective function using gradient-based optimization, written using the PyTorch library (training converges after ~ 150 epochs for the case of the Allen atlas). Tangram does not contain any hyperparameters, maps a hundred thousand cells in a few minutes (using a single P100 GPU). With this final piece our pipeline is complete, we can see an example result in Fig.2.6 where three different ROIs from different regions have been processed, we were thus able to map *all* the genetic expression into space at the same resolution of the Allen atlas effectively increasing the number of available genes from $3k$ to $30k$.

2.4 Conclusions

We mapped cells using the information of the public Allen Mouse Brain Atlas ($3k$ genes at $200\mu m$ resolution): by introducing a deep learning-based registration and mapping pipeline, we were able to map $30k$ genes obtained by scRNA-seq of the primary motor area, revealing spatial gene expression patterning beyond the limitations of current technologies.
The mapping predictions for cell types across the three ROIs examined, were self-consistent albeit less accurate than mappings using the higher resolution spatial technologies (e.g. MERFISH) we tried in our original work [19], that could push the resolution of our method from $200\mu m$ to single-cell resolution. Cortical layers were successfully recovered across the three ROIs. While our work focused on a specific region in the mouse brain is applicable to any brain region, towards its complete atlas, and to any other organ, as well as disease tissue. To integrate across scales, registration pipeline requires a CCF and is therefore currently applicable to a few organs. At present, the mouse brain possesses the most advanced and well-developed CCF, but efforts are underway to construct analogous reference maps for different organs, towards the construction of cell atlases of all organ in mouse and human.
This case makes clear how using machine learning pipelines can help improving experimental techniques, this area of research is indeed already very active and, of the ones we will explore in our dissertation, the one that is giving the bigger returns in terms of applicability. Of course, the case shown was a single example but similar techniques can be found not only on biological data, but also in high energy physics [41], quantum mechanics [42]

and chemistry [43] to name a few.

Chapter 3

Network theory for Machine Learning

In this chapter, we will focus on how to use Network Theory to improve existing ML models. In particular here we will focus on Multilayer Perceptrons (MLP) which are trained as supervised classifiers but, as we will argue in the conclusions, the intuitions behind this chapter are broader, and, in principle, applicable to whatever NN architecture. To this extent, is worth pointing out that a MLP is nothing but a bipartite fully connected network with directed weights which are adjusted via some form of gradient descent during the training phase. This is the same basic architecture we introduced in chapter 1. The network can be described by the transfer (or adjacency) matrix between nodes with some interesting properties that, as we will see in the following, can be leveraged to improve MLPs in several ways [44].
The aims of this chapter are multifold. On one side, we will develop a novel learning scheme which is anchored on reciprocal space. Instead of iteratively adjusting the weights of the edges that define the connection among nodes, we will modify the spectra of a collection of suitably engineered matrices that bridge adjacent layers. To eventually recover a multilayered feedforward architecture in direct space, we postulate a nested indentation of the associated eigenvectors. These latter act as the effective gears of a processing device operated in reciprocal space. The directed indentation between stacks of adjacent eigenvectors yield a compression of the activation pattern, which is eventually delivered to the detection nodes.

As a starting point, assume eigenvectors are frozen to a reference setting which fulfills the prescribed conditions. The learning is hence solely restricted to the eigenvalues, a choice which amounts to performing a *global* training, targeted to identifying key collective modes, the selected eigen-directions, for carrying out the assigned classification task. The idea of conducting a global training on a subset of parameters has been also proposed in other works [45, 46]. This is at odd with the usual approach to machine learning where *local* adjustments of pairwise weights are implemented in direct space. As we shall prove, by tuning the eigenvalues, while freezing the eigenvectors, yields performances superior to those reached with usual (local) techniques bound to operate with an identical number of free parameters, within an equivalent network architecture. Eigenvalues are therefore identified as key target of the learning process, proving more fundamental than any other set of identical cardinality, allocated in direct space. Remarkably, the distribution of weights obtained when applying the spectral learning technique restricted to the eigenvalues is close to that recovered when training the NN in direct space, with no restrictions on the parameters to be adjusted. In this respect, spectral learning bound to the eigenvalues could provide a viable strategy for pre-training of DNN. Further, the set of trainable eigenvalues can be expanded at will by inserting linear processing units between the adjacent layers of a non-linear multilayered perceptron. Added linear layers act as veritable booms of a *telescopic neural network*, which can be extracted during the learning phase and retracted in operational mode, yielding compact networks with improved classification skills. The effect of the linear expansion is instead negligible, if applied to learning of standard conception. The entries of the indented eigenvectors can be also trained resulting in enhanced performance, as compared to the setting where eigenvalues are exclusively modulated by the learning algorithm. To demonstrate the principles which underly spectral training, we employ the MNIST database, a collection of handwritten digits to be classified. The examined problem is relatively simple: a modest number of tunable parameters is indeed necessary for achieving remarkable success rates. When allowing for the simultaneous training of the eigenvalues and (a limited fraction of) eigenvectors, the NN quickly saturates to accuracy scores which are indistinguishable from those obtained via conventional approaches to supervised learning. More challenging tasks should be probably faced to fully appreciate the role played by a progressive optimization of the eigenmodes, the collective directions in reciprocal space where information flows. As remarked above, the eigenvectors have been here constructed so as to yield a feedforward multi-layered architecture in direct space. By relaxing this assumption, comes to altering the network topology and thus exporting the spectral learning strategy to other frameworks. In general terms, working in the spectral domain corresponds to optimizing a set of *non orthogonal* directions (in the high dimensional space of the nodes) and associated weights (the eigenvalues), a global outlook which could contribute to shed novel light on the theoretical foundations of supervised learning.

3.1 Linear and non-linear spectral learning

To introduce and test the proposed method we will consider a special task, i.e. recognition of handwritten digits. To this end, we will make use of the MNIST database [47] which has a training set of 60,000 examples, and a test set of 10,000 examples. Each image is made of $N_1 = 28 \times 28$ pixels and each pixel bears an 8-bit numerical intensity value, see Fig. 3.1. A DNN can be trained using standard backpropagation [13] algorithms to assign the weights that link the nodes (or perceptrons) belonging to consecutive layers. The first layer has N_1 nodes and the input is set to the corresponding pixel's intensity. The highest error rate reported on the original website of the database [47] is 12 %, which is achieved using a simple linear classifier, with no preprocessing. In early 2020, researchers announced 0.16 % error [48] with a DNN made of branching and merging convolutional networks. Our goal here is to contribute to the analysis with a radically different approach to the learning, rather than joining the efforts to break current limit in terms of performance and classification accuracy. More specifically, and referring to the MNIST database as a benchmark application, we will assemble a network made of N nodes, organized in successive ℓ layers, tying the training to reciprocal space.

Directed connections between nodes belonging to consecutive layers are encoded in a set of $\ell - 1$, $N \times N$ adjacency matrices. The eigenvectors of these latter matrices are engineered so as to favour the information transfer from the reading frame to the output layer, upon proper encoding. The associated eigenvalues represent the primary target of the novel learning scheme. In the following we will set up the method, both with reference to its linear and non-linear versions. Tests performed on the MNIST database are discussed in the next Section.

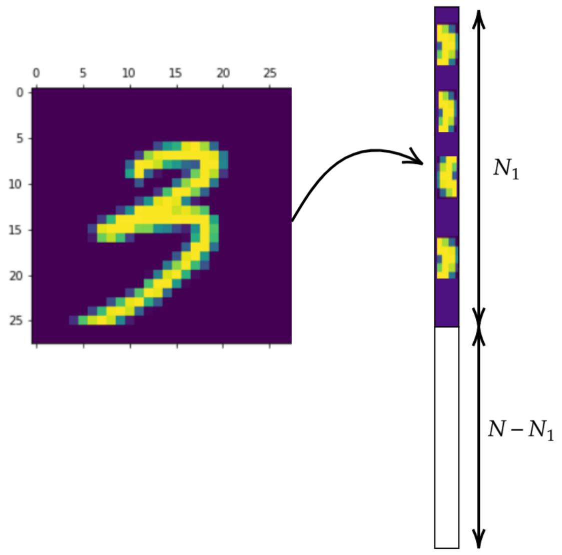


Figure 3.1: Each image of the training set is mapped into a column vector $\vec{n}_1$, of size N, whose first $N_1 = 28\times\ 28$ entries are the intensities displayed on the pixels of the image. A pictorial view of this flattening mechanism is reported in the figure.

3.1.1 Single-layer spectral learning

Assume N_i to label the nodes assigned to layer i, and define $N = \sum_{i=1}^{\ell} N_i$. For the specific case here inspected the output layer is composed by ten nodes ($N_\ell = 10$), where recognition takes eventually place. Select one image from the training set and be n_1 $(= 0, 1, 2.., 9)$ the generic number therein displayed. We then construct a column vector $\vec{n}_1$, of size N, whose first N_1 entries are the intensities displayed on the pixels of the selected image (from the top-left to the bottom-right, moving horizontally), as illustrated in Fig. 3.1. All other entries are initially set to zero. As we shall explain in the following, our goal is to transform the input $\vec{n}_1$ into an output vector with same dimensions. The last N_ℓ elements of this latter vector represent the output nodes where reading is eventually performed.

To set the stage, we begin by reporting on a simplified scenario that, as we shall prove in the following, yields a single layer perceptron. The extension to multi-layered architectures will be discussed right after.

Consider the entry layer made of N_1 nodes and the outer one composed of N_2 elements. In this case $N = N_1 + N_2$. The input vector $\vec{n}_1$ undergoes a linear transformation to yield $\vec{n}_2 = \mathbf{A}_1\vec{n}_1$ where $\mathbf{A}_1$ is a $N \times N$ matrix that we shall characterize in the following. Introduce matrix Φ_1: this is the identity matrix $\mathbb{I}_{N\times N}$ modified by the inclusion of a sub-diagonal block $N_2 \times N_1$, e.g. filled with uniformly distributed random numbers, defined in a bounded interval, see Fig. 3.2. The columns of Φ_1, hereafter $\left(\vec{\phi}_1\right)_k$ with $k = 1, ..., N$, define a basis of the N dimensional space to which $\vec{n}_1$ and $\vec{n}_2$ belong. Then, we introduce the diagonal matrix Λ_1. The entries of Λ_1 are set to random (uniform) numbers spanning a suitable interval. A straightforward calculation returns $(\Phi_1)^{-1} = 2\mathbb{I}_{N\times N} - \Phi_1$. We hence define $\mathbf{A}_1 = \Phi_1\Lambda_1\left(2\mathbb{I}_{N\times N} - \Phi_1\right)$ as the matrix that transforms $\vec{n}_1$ into $\vec{n}_2$. Because of the specific structure of the input vector, and owing the nature of $\mathbf{A}_1$, the information stored in the first N_1 elements of $\vec{n}_1$ is passed to the N_2 successive entries of $\vec{n}_2$, in a compactified form which reflects both the imposed eigenvectors' indentation and the chosen non trivial eigenvalues.

To see this more clearly, expand the N-dimensional input vector $\vec{n}_1$ on the basis made of $\left(\vec{\phi}_1\right)_k$ to yield $\vec{n}_1 = \sum_{k=1}^{N} c_k \left(\vec{\phi}_1\right)_k$ where c_k stands for the coefficients of the expansion. The first N_1 vectors are necessarily engaged to explain the non zero content of $\vec{n}_1$ and, because of the imposed indentation, rebound on the successive N_2 elements of the basis. These latter need to adjust their associated weights c_k to compensate for the echoed perturbation. The action of matrix $\mathbf{A}_1$ on the input vector $\vec{n}_1$ can be exemplified as follows:

$$\vec{n}_2 = \mathbf{A}_1\vec{n}_1 = \mathbf{A}_1 \sum_{k=1}^{N} c_k \left(\vec{\phi}_1\right)_k = \sum_{k=1}^{N_1+N_2} c_k \left(\Lambda_1\right)_k \left(\vec{\phi}_1\right)_k \tag{3.1}$$

where $\left(\Lambda_1\right)_k$ are the element of matrix Λ_1. In short, the entries of $\vec{n}_2$ from position $N_1 + 1$ to position $N_1 + N_2$ represent a compressed (if $N_2 < N_1$) rendering of the supplied input signal, the key to decipher the folding of the message being stored in the $N_2 \times N_1$ sub-diagonal block of Φ_1, (i.e. the eigenvector indentation) and in the first set of $N = N_1 + N_2$ eigenvalues $\left(\Lambda_1\right)_k$. The key idea is to propagate this message passing scheme, from the input to the output in a multi-layer setting, and adjust (a subset of) the spectral parameters involved so as to optimize the encoding of the information.

To this end, we introduce the $N \times N$ matrix operator Φ_k, for $k = 2, ..., \ell - 1$. In analogy with the above, Φ_k is the identity matrix $\mathbb{I}_{N\times N}$ modified with a sub-diagonal block $N_{k+1} \times N_k$, which extends from rows N_k to $N_k + N_{k+1}$, and

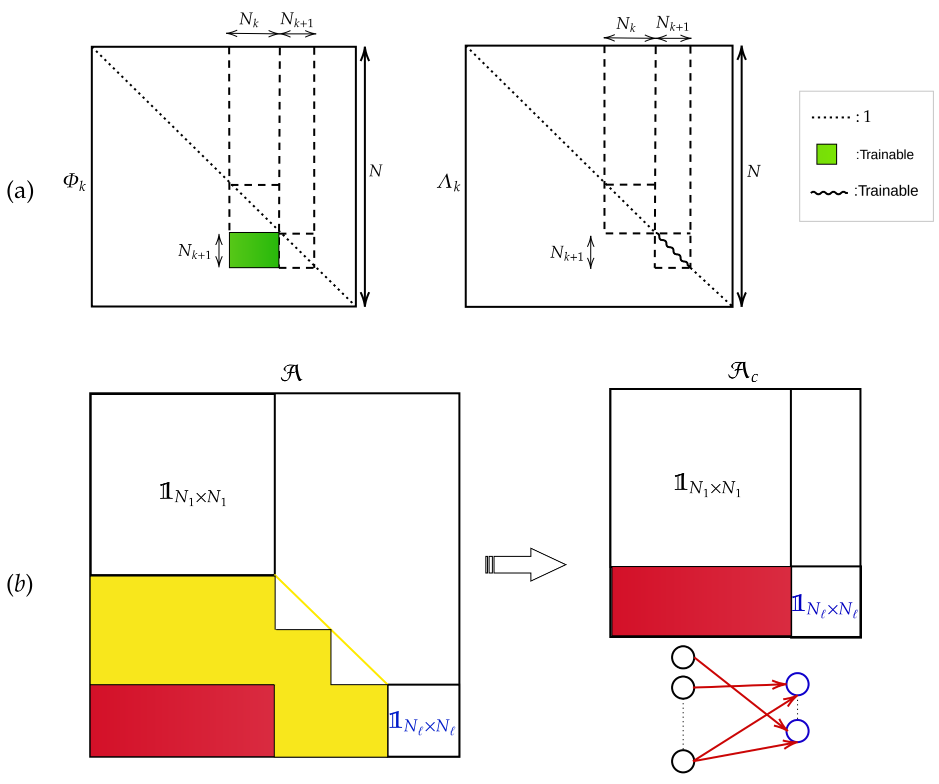


Figure 3.2: Panel (a): the structure of matrix Φ_k is schematically depicted. The diagonal entries of Φ_k are unities. The sub-diagonal block of size $N_{k+1} \times N_k$ for $k = 1, \ell - 1$ is filled with uniform random numbers in $[a, b]$, with $a, b \in \mathbb{R}$. These blocks yields an effective indentation between successive stacks of linearly independent eigenvectors. The diagonal matrix of the eigenvalues Λ_k is also represented. The sub-portions of Φ_k and Λ_k that get modified by the training performed in spectral domain are highlighted (see legend). In the experiments reported here, the initial eigenvectors entries are uniform random variables distributed in $[-0.5, 0.5]$. The eigenvalues are uniform random numbers distributed in the interval $[-0.01, 0.01]$. Optimizing the range to which the initial guesses belong (for both eigenvalues and eigenvectors) is an open problem that we have not tackled.Panel (b): a $(N_1 + N_\ell) \times (N_1 + N_\ell)$ matrix $\mathcal{A}_c$ can be obtained from $\mathcal{A} = \left(\Pi_{k=1}^{\ell-1} \mathbf{A}_k\right)$, which provides the weights for a single layer perceptron, that maps the input into the output, in direct space.

touches tangentially the diagonal, as schematically illustrated in Fig. 3.2 (a). Similarly, we introduce Λ_k, for $k = 2, ..., \ell - 1$, which is obtained from the identity matrix $\mathbb{I}_{N\times N}$ upon mutating to uniformly distributed random entries the diagonal elements that range from $\sum_{i=1}^{k} N_i$ (not included) to $\sum_{i=1}^{k+1} N_i$ (included). Finally, we define $\mathbf{A}_k = \Phi_k \Lambda_k \left(2\mathbb{I}_{N\times N} - \Phi_k\right)$, as the matrix that transforms $\vec{n}_k$ into $\vec{n}_{k+1}$, with $k = 2, ..., \ell - 1$. In principle, both non trivial eigenvalues' and eigenvectors' input can be self-consistently adjusted by the envisaged learning strategy. The input signal $\vec{n}_1$ is hence transformed into an output vector $\vec{n}_\ell$ following a cascade of linear transformations implemented via matrices $\mathbf{A}_k$. In formulae:

$$\vec{n}_\ell = \mathbf{A}_{\ell-1}...\mathbf{A}_1\vec{n}_1 = \left(\Pi_{k=1}^{\ell-1}\Phi_k\Lambda_k\left(2\mathbb{I}_{N\times N} - \Phi_k\right)\right)\vec{n}_1 \tag{3.2}$$

where in the last step we made use of the representation of $\mathbf{A}_k$ in dual space. The generic vector $\vec{n}_{k+1}$, for $k = 1, ..., \ell - 1$ is obtained by applying matrix $\mathbf{A}_k$ to $\vec{n}_k$. The first $N_1 + N_2 + ... + N_k$ components of $\vec{n}_{k+1}$ coincide with the corresponding entries of $\vec{n}_k$, namely $\left[\vec{n}_{k+1}\right]_m \equiv \left[\vec{n}_k\right]_m$ for $m < N_1 + N_2 + ... + N_k$. Here, $\left[(\vec{\cdot})\right]_m$ identifies the m-th component of the vector $(\vec{\cdot})$. Recall that, by construction, $\left[\vec{n}_k\right]_m = 0$ for $m > N_1 + N_2 + ... + N_k$. On the contrary, the components $\left[\vec{n}_{k+1}\right]_m$ with $N_1 + N_2 + ... + N_k + 1 < m < N_1 + N_2 + ... + N_k + N_{k+1}$ are populated by non trivial values which reflect the eigenvectors indentation, as well as the associated eigenvalues. This observation can be mathematically proven as follows. Write $\vec{n}_k$ on the basis formed by the eigenvectors $\left(\vec{\phi}_k\right)_l$ to eventually get:

$$\vec{n}_k = \sum_{l=1}^{N_1+N_2+...+N_{k+1}} c_l \left(\vec{\phi}_k\right)_l \equiv \sum_{l=1}^{N_1+N_2+...+N_k} c_l \vec{e}_l \tag{3.3}$$

where $(\vec{e}_1, \vec{e}_2...)$ stand for the canonical basis and the last inequality follows the specific structure of the eigenvectors (remark that the leftmost sum in the above equation includes N_{k+1} more elements than the second). By definition:

$$\vec{n}_{k+1} = \mathbf{A}_k\vec{n}_k = \sum_{l=1}^{N_1+N_2..+N_{k+1}} c_l \left(\Lambda_k\right)_l \left(\vec{\phi}_k\right)_l \tag{3.4}$$

From the above relation, one gets for $m \leq N_1 + N_2 + ... + N_k$

$$[\vec{n}_{k+1}]_m = \sum_{l=1}^{N_1+N_2..+N_k} c_l \, [\vec{e}_l]_m \equiv [\vec{n}_k]_m \tag{3.5}$$

where the first equality sign follows from the observation that $\left(\vec{\phi}_k\right)_l$ coincides with $\vec{e}_l$ and $(\Lambda_k)_l = 1$, over the explored range of m. For $N_1 + N_2 + ... + N_k + 1 \leq m \leq N_1 + N_2 + ... + N_k + N_{k+1}$, we obtain instead:

$$[\vec{n}_{k+1}]_m = \sum_{l=N_1+N_2..+N_{k-1}}^{N_1+N_2..+N_{k+1}} c_l \, (\Lambda_k)_l \left[\left(\vec{\phi}_k\right)_l\right]_m \tag{3.6}$$

Finally, it is immediate to show that $[\vec{n}_{k+1}]_m = 0$ for $m > N_1 + N_2 + ... + N_k + N_{k+1}$, because of the specific form of the employed eigenvectors. In short, the information contained in the last non trivial N_k entries of $\vec{n}_k$ rebound on the successive N_{k+1} elements of $\vec{n}_{k+1}$, funnelling the information downstream from the input to the output. The successive information processing relies on the indented (non orthogonal) eigenvectors and the associated eigenvalues, which hence define the target of the training in reciprocal space.

To carry out the learning procedure one needs to introduce a loss function $L(\vec{n}_1)$. For illustrative purposes this latter can be written as:

$$L(\vec{n}_1) = \left\| l(\vec{n}_1) - \sigma \left[\left(\Pi_{k=1}^{\ell} \Phi_k \Lambda_k \left(2\mathbb{I}_{N\times N} - \Phi_k \right) \right) \vec{n}_1 \right] \right\|^2 \tag{3.7}$$

where $\sigma(\cdot)$ is the softmax operation applied to the last entries of the ℓ-th image of the input vector $\vec{n}_1$. In the above expression, $l(\vec{n}_1)$ stands for the label attached to $\vec{n}_1$ depending on its category. More into details, the k-th entry of $l(\vec{n}_1)$ is equal unit (and the rest identically equal to zero) if the number supplied as an input is identical to k, with $k = 0, 1, ..., 9$. The loss function can be minimized by acting on the free parameters of the learning scheme. Specifically, the learning can be restricted to the set of N non trivial eigenvalues, split in ℓ distinct groups, each referred to one of the $\mathbf{A}_k$ matrices (i.e. $N_1 + N_2$ eigenvalues of $\mathbf{A}_1$, N_3 eigenvalues of $\mathbf{A}_2$,...., N_ℓ eigenvalues of $\mathbf{A}_{\ell-1}$). In addition, the sub-diagonal block entries of Φ_k, the elements of the basis which dictate the successive indentation between adjacent layers, can be adjusted as follows the training scheme. In the following section we will report about the performance of the method, implemented in its different modalities, against those obtained with a classical approach to the learning anchored in direct space. In the actual implementation we have chosen to deal with a categorical cross-entropy loss function.

Before ending this section a few remarks are mandatory. Introduce $\mathcal{A} = \Pi_{k=1}^{\ell}\mathbf{A}_k$. The linear transformation that links the input vector $\vec{n}_1$ to the generated output $\vec{n}_\ell$, can be compactly expressed as $\vec{n}_\ell = \mathcal{A}\vec{n}_1$. Then, recall that the classification relies on examining the last N_ℓ entries of $\vec{n}_\ell$. Hence, for the specific setting here examined, where the mapping is obtained as a cascade of linear transformations, one can imagine to recast the whole procedure in a space of reduced dimensionality. Be $\vec{z}$ a column vector made of $N_1 + N_\ell$ elements. The first N_1 entries of $\vec{z}$ are the intensities on the pixels of the selected image, as for the homologous $\vec{n}_1$ quantity. The other elements are set to zero. Then, consider the $(N_1{+}N_\ell)\times(N_1{+}N_\ell)$ matrix $\mathcal{A}_c$ (the label c stands for *compact*), constructed from $\mathcal{A}$ by trimming out all the information that pertain to the intermediate layers, as introduced in the reciprocal space (see Fig. 3.2(b)). Stated differently, matrix $\mathcal{A}_c$ provides the weighted links that feed from the input to the output layer in direct space, via the linear transformation $\mathcal{A}_c\vec{z}$: this is a single layer perceptron, shown in Fig. 3.2(b), which was trained by endowing reciprocal space with an arbitrary number of additional dimensions, the intermediate stacks responsible for the sequential embedding of the information. Intermediate layers can be literally extracted, during the training phase, and subsequently retracted in operational mode. The importance to allowing for additional layers, and so provide the NN of a telescopic attribute, will be assessed in the forthcoming sections.

From the algorithmic point of view the process outlined above can be rephrased in simpler, although equivalent terms. For all practical purposes, one could take the (column) input vector $\vec{n}_1$ to have $N_1 + N_2$ elements. Following the scheme depicted above, the first N_1 entries are the intensities on the pixels of the selected image, while the remaining N_2 elements are set to zero. We now introduce a $(N_1 + N_2) \times (N_1 + N_2)$ matrix $\mathbf{A}_1$. This is the identity matrix $\mathbb{I}_{(N_1+N_2)\times(N_1+N_2)}$ with the inclusion of a sub-diagonal block $N_2 \times N_1$, which handles the information processing that will populate the second N_2 elements of the output vector $\vec{n}_2 = \mathbf{A}_1\vec{n}_1$. Then, we formally replace the $(N_1 + N_2)$ column vector $\vec{n}_2$ with a column vector made of $(N_2 + N_3)$ elements, termed $\vec{n}_{2t}$, whose first N_2 elements are the final entries of $\vec{n}_2$. The remaining N_3 elements of $\vec{n}_{2t}$ are set to zero. Now, rename $\vec{n}_{2t}$ as $\vec{n}_2$ and presents it as the input of a $(N_2 + N_3) \times (N_2 + N_3)$ matrix $\mathbf{A}_2$, with a non trivial sub-diagonal $N_3 \times N_2$ block. This latter maps the first N_2 elements of the input vector, into the successive N_3 of the output one, by completing the second step of an algorithmic scheme which can be iteratively repeated. In analogy with the above, each $(N_k + N_{k+1}) \times (N_k + N_{k+1})$ matrix $\mathbf{A}_k$ can be written as $\mathbf{A}_k = \Phi_k \Lambda_k \left(2\mathbb{I}_{(N_k+N_{k+1})\times(N_k+N_{k+1})} - \Phi_k\right)$, where now the column vectors of Φ_k are the eigevenctors of $\mathbf{A}_k$ and form a non-orthogonal basis of the $(N_k + N_{k+1})$ space where input and output vectors belong. Λ_k is a diagonal matrix of the eigenvalues: the first N_k are set to one, while the other N_{k+1} are non trivial entries to be adjusted self-consistently via the learning scheme. Framing the process in the augmented space of N dimensions, as done earlier, allows us to avoid adapting the dimensions of the involved vectors at each iteration. On the contrary, this is a convenient procedure to be followed when aiming at a numerical implementation of the envisaged scheme. Notice that to discuss the algorithmic variant of the method, we made use of the same symbols employed earlier. The notation clash is however solely confined to this paragraph.

In the following, we will discuss how these ideas extend to the more general setting of non-linear multi-layered NN.

3.1.2 Multi-layer networks in the spectral domain

In analogy with the above, the image to be processes is again organized in a $N \times 1$ column vector $\vec{n}_1$. This latter is transformed into $\vec{n}_2 = \mathbf{A}_1 \vec{n}_1$, where matrix $N \times N$ matrix $\mathbf{A}_1$ is recovered from its spectral properties, respectively encoded in Φ_1 and Λ_1. The output vector $\vec{n}_2$ is now filtered via a suitable *non-linear* function $f(\cdot)$. This step marks a distinction between, respectively, the linear and non-linear versions of the learning schemes. For the applications here reported we have chosen to work with a rectified linear unit (ReLU) $f(\cdot) = max(0, \cdot)$. Another possibility is to set $f(\cdot, \beta_1) = \tanh[\beta_1(\cdot)]$, where β_1 is a control parameter which could be in principle self-consistently adjusted all along the learning procedure. We are now in a position to iterate the same reasoning carried out in the preceding section, adapted to the case at hand. More specifically, we introduce the generic $N \times N$ matrix $\mathbf{A}_k = \Phi_k \Lambda_k \left(2\mathbb{I}_{N\times N} - \Phi_k\right)$ which transforms $\vec{n}_k$ into $\vec{n}_{k+1}$, with $k = 2, ..., \ell-1$. The outcome of this linear transformation goes through the non-linear filter.

The loss function $L(\vec{n})$ generalizes to:

$$L(\vec{n}) = \|l(\vec{n}_1) - \sigma\left(f\left(\mathbf{A}_{\ell-1}....f\left(\mathbf{A}_2 f\left(\mathbf{A}_1\vec{n_1}, \beta_1\right), \beta_2\right), \beta_{\ell-1}\right)\right)\|^2 \tag{3.8}$$

with an obvious meaning of the involved symbols. In the set of experiments reported below we assume, in analogy with the above, a categorical cross-entropy loss function. The loss function is minimized upon adjusting the free parameters of the learning scheme: the $\ell - 1$ blocks of tunable eigenvalues, the elements that define the successive indentation of the nested basis which commands the transfer of the information (and e.g. the quantities β_k, if the sigmoidal hyperbolic function is chosen as a non-linear filter). This eventually yields a fully trained network, in direct space, which can be unfolded into a layered architecture to perform pattern recognition (see Fig. 3.3). Remarkably, self-loop links are also present. The limit of a linear single layer perceptron is recovered when silencing the non-linearities: a $(N_1 + N_\ell) \times (N_1 + N_\ell)$ matrix $\mathcal{A}_c$ can be generated from the $N \times N$ matrices $\mathbf{A}_k$, following the same strategy outlined above. A sequence of linear layers can be also interposed between two consecutive non-linear stacks. The interposed layers allow to enlarge the space of parameters employed in the learning scheme, and can be retracted when operating the DNN after completion of the learning stage. Their role is *de facto* encapsulated in the entries of the linear operator that bridges the gap between the adjacent non-linear stacks, as explained above when referring to the telescopic operational modality.

3.2 Results

To build and train the aforementioned models we used TensorFlow and created a custom spectral layer matrix that could be integrated in virtually every TensorFlow or Keras model. That allowed us to leverage on the automatic differentiation capabilities and the built-in optimizers of TensorFlow. Recall that we aim at training just a a portion of the diagonal of Λ_k and a block of Φ_k. To reach this goal we generated two fully trainable matrices, for each layer in the spectral domain, and applied a suitably designed mask to filter out the sub-parts of the matrices to be excluded from the training. This is easy to implement and, although improvable from the point of view of computational efficiency, it works perfectly, given the size of the problem to be handled. We then trained all our models with the AdaMax optimizer [49] by using a learning rate of 0.03 for the linear case and 0.01 for the non-linear

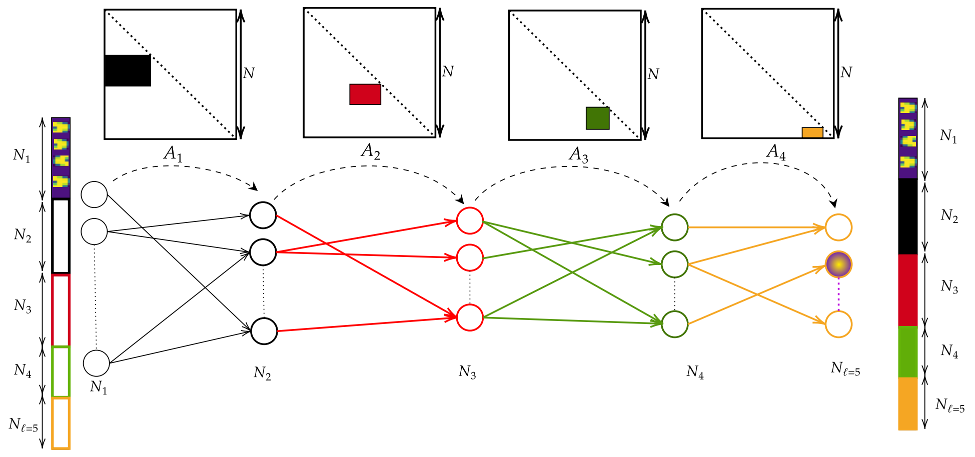


Figure 3.3: The non-linear version of the training scheme returns a multi-layered architecture with self-loops links in direct space. Linear and non-linear transformation can be combined at will, matrices $\mathbf{A}_k$ providing the connection between successive layers. Linear layers can be retracted in operational mode, following a straightforward variant of the compactification procedure described in the main text.

one. The training proceeded for about 20 epochs and during each epoch the network was fed with batches of images of different size, ranging from 300 to 800. These hyperparameters have been chosen so as to improve on GPU efficiency, accuracy and stability. However, we did not perform a systematic study to look for the optimal setting. All our models have been trained on a virtual machine hosted by Google Colaboratory. Standard NN have been trained on the same machine using identical software and hyperparameters, for a fair comparison. Further details about the implementation, as well as a notebook to reproduce our results, can be found in the public repository of this project [50].

We shall start by reporting on the performance of the linear scheme. The simplest setting is that of a perceptron made of two layers: the input layer with $N_1 = 28 \times 28 = 784$ nodes and the output one made of $N_2 = 10$ elements. The perceptron can be trained in the spectral domain by e.g. tuning the $N = N_1 + N_2 = 794$ eigenvalues of $\mathbf{A}_1$, the matrix that links the input ($\vec{n}_1$) and output ($\vec{n}_2$) vectors. The learning restricted to the eigenvalues returns a perceptron which performs the sought classification task with an accuracy (the fraction of correctly recognized images in the test-set) of

$(82 \pm 2)\%$ (averaging over 5 independent runs). This figure is to be confronted with the accuracy of a perceptron trained with standard techniques in direct space. For a fair comparison, the number of adjustable weights should be limited to N. To this aim, we randomly select a subset of weights to be trained and carry out the optimization on these latter. The process is repeated a few (5 in this case) times and, for each realization, the associated accuracy computed. Combining the results yields an average performance of $(79 \pm 3)\%$, i.e. a slightly smaller score (although compatible within error precision) than that achieved when the learning takes place in the spectral domain. When the training extends to all the $N_1 \times N_2$ weights (plus $N_1 + N_2$ bias), conventional learning yields a final accuracy of $(92.7 \pm 0.1)\%$. This is practically identical to the score obtained in the spectral domain, specifically $(92.5 \pm 0.2)\%$, when the sub-diagonal entries of the eigenvectors matrix are also optimized (for a total of $N_1 + N_2 + N_1 \times N_2$ free parameters). The remarkable observation is however that the distribution of the weights as obtained when the learning is restricted on the eigenvalues (i.e using about the 10 % of the parameters employed for a full training in direct space) matches quite closely that retrieved by means of conventional learning schemes, see Fig. 3.4 . This is not the case when the learning in direct space acts on a subset of N, randomly selected, weights (data not shown). Based on the above, it can be therefore surmised that optimizing the eigenvalues constitutes a rather effective pre-training strategy, which engages a modest computational load.

To further elaborate on the potentiality of the proposed technique, we modify the simple two-layers perceptron, with the inclusion of supplementary computing layers. As explained above the newly added layers plays an active role during the learning stage, but can be retracted in operating mode so as to return a two-layers perceptron. The weights of this latter bear however an imprint of the training carried out for the linear network in the expanded configuration. Two alternative strategies will be in particular contemplated. On the one side, we will consider a sole additional layer, endowed with N_2 nodes, interposed between the input and output layers made of, respectively, $N_1 = 784$ and $N_\ell \equiv N_3 = 10$ nodes. We will refer to this as to the *wide linear* configuration. The performance of the method can be tested by letting N_2 to progressively grow. On the other side, the *deep linear* configuration is obtained when interposing a sequence of successive (linear) stacks between the input ($N_1 = 784$) and the output ($N_\ell = 10$) layers.

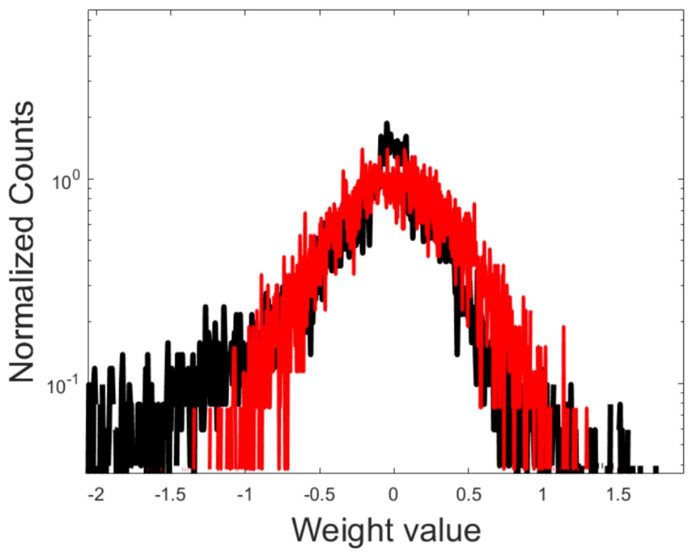


Figure 3.4: Distribution of the weights of a perceptron. The red line follows the spectral training limited the $N_1 + N_2$ eigenvalues. The black line follows the training in direct space. Here, $N_1 \times N_2$ parameters are adjusted in the space of the nodes. The distribution are very similar, but the spectral learning employs about 10% of the parameters used in direct space. The distributions obtained when forcing the training in direct space to operate on a subset of $N_1 + N_2$ weights are very different from the one displayed (for every choice of the randomly selected family of weights to be trained).

In Fig. 3.5, we report on the performance of the wide learning scheme as a function of $N_2 + N_3$. As we shall clarify, this latter stands for the number of trained parameters for (i) the spectral learning acted on a subset of the tunable eigenvalues and for (ii) the conventional learning in direct space restricted to operate on a limited portion of the weights. The red line in the main panel of Fig. 3.5 refers to the simplified scheme where a subset of the eigenvalues are solely tuned (while leaving the eigenvectors fixed at the random realization set by the initial condition). We have in particular chosen to train the second bunch of N_2 eigenvalues of the transfer matrix $\mathbf{A}_1$ and the $N_3 = 10$ non trivial eigenvalues of matrix $\mathbf{A}_2$, in line with the prescriptions reported in the preceding Section. The blue line reports on the accuracy of the NN trained in direct space: the target of the optimization is a subset of cardinality $N_2 + N_3$ of the $N_1N_2 + N_2N_3$ weights which could be in principle adjusted in the space of the nodes. The performance of the spectral method proves clearly superior, as it can be readily appreciated by visual inspection of Fig. 3.5. The black line displays the accuracy of the linear NN when the optimization acts on the full set of $N_1N_2 + N_2N_3$ trainable parameters. No improvement is detectable when increasing the size of the intermediate layer: the displayed accuracy is substantially identical to that obtained for the basic perceptron trained with $N_1N_2 = 7840$ parameters. The spectral learning allows to reach comparable performance already at $N_2 = 1000$ (13% of the parameters used for the standard two layers perceptron with $N_1 \times N_2$ parameters, as discussed above). In the inset of Fig. 3.5, the distribution of the entries of matrix $\mathcal{A}_c$, the equivalent perceptron, is depicted in red for the setting highlighted in the zoom. The black line refers to the two-layers equivalent of the NN trained in direct space, employing the full set of trainable parameters (black dot enclosed in the top-left dashed rectangle drawn in the main panel of Fig. 3.5). The two distributions look remarkably close, despite the considerable reduction in terms of training parameters, as implemented in the spectral domain (for the case highlighted, 0.13% of the parameters employed under the standard training). Similarly to the above, the distribution obtained when forcing the training in direct space to act on a subset of $N_1 + N_2$ weights are just a modest modulation of the initially assigned profile, owing to the *local* nature of the learning in the space of the nodes.

In Fig. 3.6, we report the results of the tests performed when operating under the deep linear configuration. Symbols are analogous to those employed in Fig. 3.5. In all inspected cases, the entry layer is made of $N_1 = 784$ elements and the output one has $N_\ell = 10$ nodes. The first five points, from left to right, refer to a three layers (linear) NN. Hence, $\ell = 3$ and the size of the intermediate layer is progressively increased, $N_2 = 20, 80, 100, 500, 800$. The total number of trained eigenvalues is $N_2 + N_3$, and gets therefore larger as the size of the intermediate layer grows. The successive four points of the collections are obtained by setting $\ell = 4$. Here, $N_2 = 800$ while N_3 is varied $(= 100, 200, 400, 600)$. The training impacts on $N_2 + N_3 + N_4$ parameters. Finally the last point in each displayed curve is obtained by working with a five layers DNN, $\ell = 5$. In particular $N_2 = 800$, $N_3 = 600$ and $N_4 = 500$, for a total of $N_2 + N_3 + N_4 + N_5$ tunable parameters. Also in this case, the spectral algorithm performs better than conventional learning schemes constrained to operate with an identical number of free parameters. Similarly, the distribution of the weights of an equivalent perceptron trained in reciprocal space matches that obtained when operating in the space of the nodes and resting on a considerably larger number of training parameters. To sum up, eigenvalues are parameters of key importance for NNs training, way more strategic than any other set of equivalent cardinality in the space of the nodes. As such, they allow for a global approach to the learning, with significant reflexes of fundamental and applied interest. In all cases here considered, the learning can extend to the eigenvectors: an optimized indentation of the eigen-directions contribute to enhance the overall performance of the trained device.

We now turn to considering a non-linear architecture. More specifically, we will assume a four layers network with, respectively, $N_1 = 784, N_2, N_3 = 120, N_4 = 10$. The non-linear ReLU filter acts on the third layer of the collection, while the second is a linear processing unit. As in the spirit of the wide network configuration evoked above, we set at testing the performance of the NN for increasing N_2. For every choice of N_2, the linear layer can be retracted yielding a three-layered effective non-linear configurations. We recall however that training the network in the enlarged space where the linear unit is present leaves a non trivial imprint in the weights that set the strength of the links in direct space.

In Fig 3.7, we plot the computed accuracy as a function of N_2, the size of the linear layer. In analogy with the above analysis, the red curve refers to the training restricted to $N_2+N_3+N_4$ eigenvalues; the blue profile is obtained when the DNN is trained in direct space by adjusting an identical number of inter-nodes weights. As for the case of a fully linear architecture, by adjusting the eigenvalues yields better classification performances. The black line shows the accuracy of the NN when the full set of $N_1N_2 + N_2N_3 + N_3N_4$ is optimized in direct space. The green line refer instead to the spectral learning when the eigenvalues and eigenvectors are trained simultaneously. The accuracies estimated for these two latter settings agree within statistical error, even if the spectral scheme seems more robust to overfitting (the black circles declines slightly when increasing N_2, while the collection of green points appears rather stable).

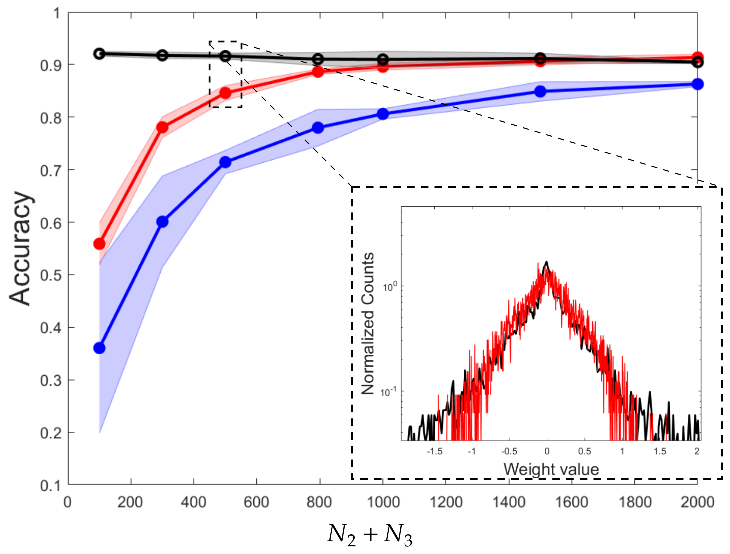


Figure 3.5: A three layer NN is considered. The accuracy of the network is plotted as a function of the number of parameters that we chose to train with the spectral algorithm, $N_2 + N_3$. The red line reports on the performance of the spectral training. The blue line refers to the network trained in direct space: the optimization runs on $N_2 + N_3$ parameters, a subset of the total number of adjustable weights $N_1N_2 + N_2N_3$. The black line stands for the accuracy of the linear NN when training the full set of $N_1N_2 + N_2N_3$ parameters. Notice that the reported accuracy is comparable to that obtained for a standard two layer perceptron. Inset: the distribution of the entries of the equivalent perceptrons are plotted. The red curve refer to the spectral learning restricted to operate on the eigenvalues; the black profile to the network trained in direct space, employing the full set of adjustable parameters. In both cases, the weights refer to the two layers configuration obtained by retracting the intermediate linear layer employed during the learning stage.

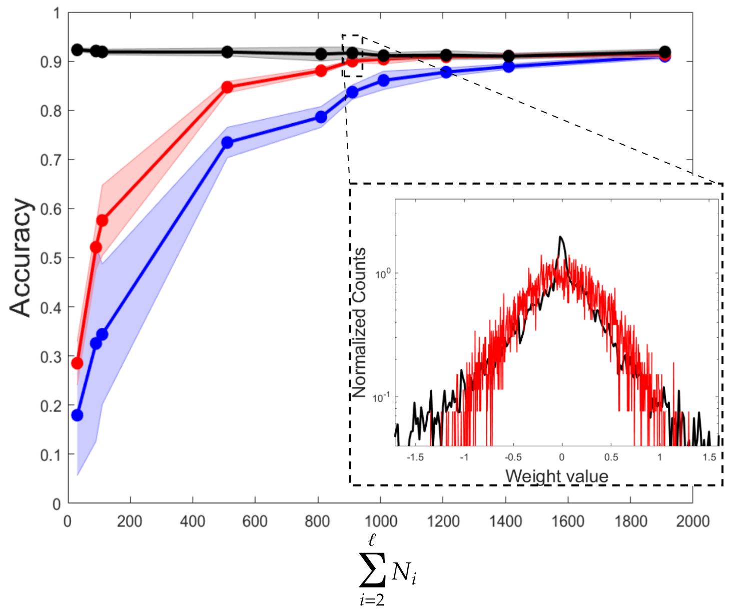


Figure 3.6: The performance of the spectral algorithm are tested for a multi-layered linear configuration. Symbols are chosen in analogy to Fig. 3.5. In all cases, the input layer is made of $N_1 = 784$ elements and the output layer has $N_\ell = 10$ nodes. The first five points, from left to right in each of the curves depicted in the main panel, refer to a three layers (linear) NN. The size of the intermediate layer is progressively increased, as $N_2 = 20, 80, 100, 500, 800$. The total number of trained eigenvalues is $N_2 + N_3$. The subsequent four points are obtained by considering a four layers architecture. In particular, $N_2 = 800$ while N_3 takes values in the interval $(100, 200, 400, 600)$. The training acts on $N_2 + N_3 + N_4$ eigenvalues. The final point in each curve is obtained with a four layers DNN. Here, $N_2 = 800$, $N_3 = 600$ and $N_3 = 500$, for a total of $N_2 + N_3 + N_4 + N_5$ tunable parameters in the spectral setting. Inset: the distribution of the entries of the equivalent perceptrons are displayed, with the same color code adopted in Fig. 3.5. Also in this case, the weights refer to the two layers configuration obtained by retracting the intermediate linear layers employed in the learning stage.

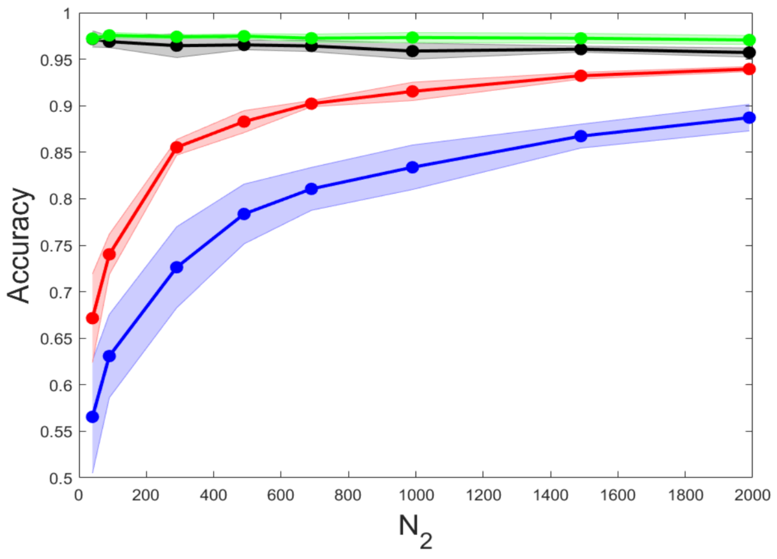


Figure 3.7: The accuracy of the non-linear DNN is tested. We assume a four layers network with, respectively, $N_1 = 784, N_2, N_3 = 120, N_4 = 10$; N_2 is changed so as to enlarge the set of parameters to be trained. The red line refers to the spectral training, with $N_2 + N_3 + N_4$ adjusted eigenvalues. The blue line stands for a network trained in direct space, the target of the optimization being a subset made of $N_2+N_3+N_4$ weights, randomly selected from the available pool of $N_1N_2 + N_2N_3 + N_3N_4$ tunable parameters. The black line reports the accuracy of the linear NN when training the full set of $N_1N_2 + N_2N_3 + N_3N_4$ weights. The green line refer to the spectral learning when eigenvalues and eigenvectors are simultaneously trained.

3.3 Conclusions

Summing up, we have here proposed a novel approach to the training of DNNs which is bound to the spectral, hence reciprocal, domain. The eigenvalues and eigenvectors of the adjacency matrices that connects consecutive layers via directed feed-forward links are trained, instead of adjusting the weights that bridge each pair of nodes of the collection, as it is customarily done in the framework of conventional ML approaches.

The first conclusion of our analysis is that optimizing the eigenvalues, when freezing the eigenvectors, yields performances which are superior to those attained with conventional methods *restricted* to a operate with an identical number of free parameters. It is therefore surmised that eigenvalues are key target parameters for NN training, in that they allow for a *global* handling of the learning. This is at variance with conventional approaches which seek at modulating the weights of the links among mutually connected nodes. Secondly, the spectral learning restricted to the eigenvalues yields a distribution of the weights which resembles quite closely that obtained with conventional algorithms bound to operate in direct space. For this reason, the proposed method could be used in combination with existing ML algorithms for an effective (and computationally advantageous) pre-training of DNNs. We have also shown that linear processing units inserted in between consecutive, non-linearly activated layers produce an enlargement of the learning parameters space, with beneficial effects in terms of performance of the trained device. Extending the learning so as to optimize the eigenvectors enhances the ability of the network to operate the sought classification. In the proposed implementation, and to recover a feed-forward architecture in direct space, we have assumed a nested indentation of the eigenvectors. Entangling the eigenvectors referred to successive stacks is the key for a recursive processing of the data, from the input to the output layer. Employing other non-orthogonal basis could eventually allow to challenge different topologies in direct space and shed novel light on the surprising ability of deep networks to cope with the assigned tasks.

Chapter 4

Hybrid Variational Autoencoders

In this chapter we will focus on using cutting-edge physical hardware to improve (or even accelerate) classical machine learning tasks. This part of the work has been carried out during a visit at D-Wave Systems Inc., in collaboration with the Quantum Machine Learning team of D-Wave. In particular our focus was to harness the complexity of D-Wave's quantum processor in order to create a powerful hybrid quantum-classical deep neural network [51] which could be trained end-to-end using standard ML libraries as TensorFlow. In general, research in the hybrid quantum-classical models focuses on deep supervised learning [13, 18, 52–54], in which a labeled dataset is used to train a statistical model to solve classification tasks. In this context, deep neural networks are now commonly used in many scientific and industrial applications and that's a great driver to find compelling applications of quantum algorithms. However there's more than can be done using such networks, for example *unsupervised learning.* Unlike supervised learning [10], unsupervised learning is a much harder, and still largely unsolved, problem. And yet, it has the appealing potential to learn the hidden statistical correlations of large unlabeled datasets [11–13], which constitute the vast majority of data available today.

Training and deployment of large-scale machine learning models, especially for unsupervised learning, faces computational challenges [55] that are only partially met by the development of special purpose classical computing units such as GPUs. This has led to an interest in applying quantum computing to ML tasks [56–60] and to the development of several quantum algorithms [61–64] with the potential to accelerate training. Most quantum machine learning algorithms need fault-tolerant quantum computation [65–67], which requires the large-scale integration of millions of qubits and is still not available today. It is however possible that quantum machine learning will provide the first breakthrough algorithms to be implemented on commercially available quantum annealers [68, 69] and gate-model devices [70, 71]. For example, small gate-model devices and quantum annealers have been used to perform quantum heuristic optimization [71–76] to solve clustering [77] and classification problems [78–81]. In this chapter we will implement an hybrid quantum-classical Variational Autoencoder (VAE) using the commercially available D-Wave 2000Q quantum annealer. The chapter is structured as follows. In Sec. 4.1 we review VAE and the implementations of discrete latent variables, a necessary step to implement quantum and classical Boltzmann Machines (BMs) in their latent space. In Sec. 4.2 we will introduce quantum annealers as samplers to train quantum and classical BMs. In Sec. 4.3 we report our experiments in training VAEs with D-Wave 2000Q systems. In Sec. 4.4 we discuss a possible path toward quantum advantage in our setup. Finally, we present our conclusions in Sec. 4.5.

4.1 Variational Autoencoders

In this section, we will briefly introduce VAEs and describe their extension to discrete latent variables, a necessary step to hybridize with quantum processors and to perform quantum-assisted training.

In generative modeling, the goal is to train a probabilistic model such that the model distribution $p_{\boldsymbol{\theta}}(\mathbf{X})$ (where $\boldsymbol{\theta}$ are the parameters of the model) is as close as possible to the data distribution, $p_{\text{data}}(\mathbf{X})$, which is unknown but assumed to exist. The ensemble $\mathbf{X} = \{\mathbf{x}^d\}_{d=1}^N$ represents the training set, i.e. N independent and identically distributed samples coming from $p_{\text{data}}(\mathbf{X})$. The preferred method to training probabilistic models is arguably maximum likelihood estimation (MLE), which means the optimal model parameters are obtained by maximizing the log-likelihood $L(\mathbf{X}, \boldsymbol{\theta})$ of the dataset with respect to the model:

$$L(\mathbf{X}, \boldsymbol{\theta}) = \sum_{\mathbf{x} \in \mathbf{X}} p_{\text{data}}(\mathbf{x}) \log p_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}}[\log p_{\boldsymbol{\theta}}(\mathbf{x})] \,, \tag{4.1}$$

where $\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}}[\ldots]$ is the expectation over $p_{\text{data}}(\mathbf{x})$. Similarly to generative adversarial networks (GANs) [82], VAEs [83] are "directed" probabilistic models with latent variables (see Fig. 4.1): the model distribution, defined as the joint distribution between the visible units $\mathbf{x}$ and latent units $\boldsymbol{\zeta}$, is explicitly parametrized as the product of the "prior" $p_{\boldsymbol{\theta}}(\boldsymbol{\zeta})$ and "marginal" $p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{\zeta})$ distributions, $p_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\zeta}) = p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{\zeta}) p_{\boldsymbol{\theta}}(\boldsymbol{\zeta})$. The model prediction for the data is then obtained by marginalizing over the latent units:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{\zeta}) p_{\boldsymbol{\theta}}(\boldsymbol{\zeta}) d\boldsymbol{\zeta} \,. \tag{4.2}$$

Generative models with latent variables can potentially learn and encode useful representations of the data in the latent space. This is an important property that can be exploited in many practical applications [84–87] to improve other tasks such as supervised and semi-supervised learning [88]. The drawback is "intractable inference" due to the appearance of integrals such as the one in Eq. 4.2. Essentially, VAEs remove the necessity to evaluate such integrals by introducing a variational approximation $q_{\boldsymbol{\phi}}(\boldsymbol{\zeta}|\mathbf{x})$ to the true posterior $p_{\boldsymbol{\theta}}(\boldsymbol{\zeta}|\mathbf{x})$. A so-called "reparameterization trick" is also introduced to obtain an efficient and low-variance estimate of the gradients needed for training. We will briefly review these two important elements in the next two sections.

Variational inference

Training generative models with latent variables via MLE requires the evaluation of the intractable integral of Eq. 4.2 to calculate the posterior distribution $p_{\boldsymbol{\theta}}(\boldsymbol{\zeta}|\mathbf{x})$. VAEs circumvent this problem by introducing a tractable variational approximation $q_{\boldsymbol{\phi}}(\boldsymbol{\zeta}|\mathbf{x})$ to the true posterior [89], with variational parameters $\boldsymbol{\phi}$ (see Fig. 4.1). VAEs are then trained by maximizing a variational lower bound $\mathcal{L}(\mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\phi})$ to the log-probabilities $\log p_{\boldsymbol{\theta}}(\mathbf{x})$:

$$\begin{aligned} \mathcal{L}(\mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\phi}) &= \log p_{\boldsymbol{\theta}}(\mathbf{x}) - \mathbb{E}_{\boldsymbol{\zeta} \sim q_{\boldsymbol{\phi}}(\boldsymbol{\zeta}|\mathbf{x})} \left[\log \frac{q_{\boldsymbol{\phi}}(\boldsymbol{\zeta}|\mathbf{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{\zeta}|\mathbf{x})} \right] \equiv \\ &\equiv \log p_{\boldsymbol{\theta}}(\mathbf{x}) - D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{\zeta}|\mathbf{x}) || p_{\boldsymbol{\theta}}(\boldsymbol{\zeta}|\mathbf{x})) \,, \end{aligned} \tag{4.3}$$

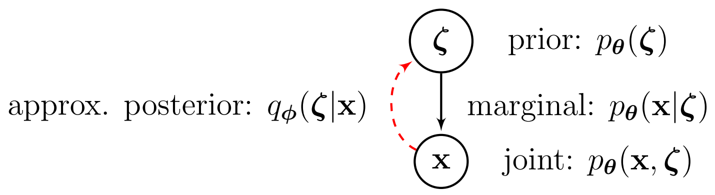

prior: $p_{\boldsymbol{\theta}}(\boldsymbol{\zeta})$
approx. posterior: $q_{\boldsymbol{\phi}}(\boldsymbol{\zeta}|\mathbf{x})$
marginal: $p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{\zeta})$
joint: $p_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\zeta})$
$\boldsymbol{\zeta}$
$\mathbf{x}$

Figure 4.1: Generative models with latent variables can be represented as probabilistic graphical models that describe conditional relationships among variables. In a directed generative model, the joint probability distribution $p_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\zeta})$, is decomposed as $p_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\zeta}) = p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{\zeta})p_{\boldsymbol{\theta}}(\boldsymbol{\zeta})$. The prior distribution over the latent variables $p_{\boldsymbol{\theta}}(\boldsymbol{\zeta})$ and the marginal (decoder) distribution $p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{\zeta})$ are hard-coded to explicitly define the model. The computation of the true posterior, $p_{\boldsymbol{\theta}}(\boldsymbol{\zeta}|\mathbf{x})$ is intractable. In VAEs, an approximating posterior $q_{\boldsymbol{\phi}}(\boldsymbol{\zeta}|\mathbf{x})$ (decoder) is introduced to replace the true posterior.

where $D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{\zeta}|\mathbf{x})||p_{\boldsymbol{\theta}}(\boldsymbol{\zeta}|\mathbf{x}))$ is the Kullback-Leibler divergence (KL divergence) between the true and approximating posteriors. Since KL divergences are always non-negative, we have

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\phi}) \leq \log p_{\boldsymbol{\theta}}(\mathbf{x}), \tag{4.4}$$

which immediately gives:

$$\mathcal{L}(\mathbf{X}, \boldsymbol{\theta}, \boldsymbol{\phi}) \equiv \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\mathcal{L}(\mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\phi})] \leq \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log p_{\boldsymbol{\theta}}(\mathbf{x})] \equiv L(\mathbf{X}, \boldsymbol{\theta}), \tag{4.5}$$

where $\mathcal{L}(\mathbf{X}, \boldsymbol{\theta}, \boldsymbol{\phi})$ is called the evidence lower bound (ELBO). The ELBO can be written in terms of tractable quantities:

$$\mathcal{L}(\mathbf{X}, \boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\Big[\mathbb{E}_{\boldsymbol{\zeta}\sim q_{\boldsymbol{\phi}}(\boldsymbol{\zeta}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{\zeta})] - D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{\zeta}|\mathbf{x})||p_{\boldsymbol{\theta}}(\boldsymbol{\zeta}))\Big]. \tag{4.6}$$

The marginal $p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{\zeta})$ and approximating posterior $q_{\boldsymbol{\phi}}(\boldsymbol{\zeta}|\mathbf{x})$, also called "decoder" and "encoder" respectively, are commonly parameterized using deep neural networks.

The reparameterization trick

To train VAEs, we need to calculate the derivatives of the objective function (Eq. 4.6) with respect to the generative ($\boldsymbol{\theta}$) and inference ($\boldsymbol{\phi}$) parameters.

The naive evaluation of $\partial_{\boldsymbol{\phi}}$ of terms of the type $\mathbb{E}_{\boldsymbol{\zeta}\sim q_\phi}[f(\boldsymbol{\zeta})]$ is called REINFORCE [90]. With the use of the identity $\partial_{\boldsymbol{\phi}} q_\phi = q_\phi \partial_{\boldsymbol{\phi}} \log q_\phi$, one has:

$$\partial_{\boldsymbol{\phi}} \mathbb{E}_{\boldsymbol{\zeta}\sim q_\phi}\left[f(\boldsymbol{\zeta})\right] = \mathbb{E}_{\boldsymbol{\zeta}\sim q_\phi}\left[f(\boldsymbol{\zeta})\partial_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}\right] . \tag{4.7}$$

However, the term above has high variance and requires intricate variance-reduction mechanisms to be of practical use [91].

A better approach is to write the random variable $\boldsymbol{\zeta}$ as a deterministic function of the distribution parameters $\boldsymbol{\phi}$ and of an additional auxiliary random variable $\boldsymbol{\rho}$. The latter is given by a probability distribution $p(\boldsymbol{\rho})$ that does not depend on $\boldsymbol{\phi}$. This reparameterization $\boldsymbol{\zeta}(\boldsymbol{\phi},\boldsymbol{\rho})$ is appropriately chosen so that one can write $\mathbb{E}_{\boldsymbol{\zeta}\sim q_\phi}[f(\boldsymbol{\zeta})] = \mathbb{E}_{\boldsymbol{\rho}\sim p(\boldsymbol{\rho})}[f(\boldsymbol{\zeta}(\boldsymbol{\phi},\boldsymbol{\rho}))]$. Therefore, we can move the derivative inside the expectation with no difficulties:

$$\partial_{\boldsymbol{\phi}} \mathbb{E}_{\boldsymbol{\zeta}\sim q_\phi}\left[f(\boldsymbol{\zeta})\right] = \mathbb{E}_{\boldsymbol{\rho}\sim p(\boldsymbol{\rho})}\left[\partial_{\boldsymbol{\phi}} f(\boldsymbol{\zeta}(\boldsymbol{\phi},\boldsymbol{\rho}))\right] . \tag{4.8}$$

This is called the *reparameterization trick* [83] and its efficient implementation is responsible for the recent success and proliferation of VAE.

4.1.1 VAE with discrete latent variables

The application of the reparameterization trick as in Eq. 4.8 requires that $f(\boldsymbol{\zeta}(\boldsymbol{\phi},\boldsymbol{\rho}))$ be differentiable, so the latent variables $\boldsymbol{\zeta}$ are continuous. However, discrete latent units can be indispensable to represent the right distributions, such as in attention models, language modeling, and reinforcement learning [88, 92, 93]. For example, a latent space composed of discrete variables can learn to disentangle content and style information of images in an unsupervised fashion [94]. Several methods have thus been developed to circumvent the non-differentiability of discrete latent units [91, 95–97]. In the context of VAE, the reparameterization trick has been extended to discrete variables by either relaxation of discrete variables into continuous variables [92, 98, 99] or by introducing smoothing functions [100]. In Ref. [101], QVAE was introduced based on the implementation of Ref. [100]. Here, we follow the implementation of Ref. [99], which gives biased estimates but provides a much simpler and flexible implementation.

To set up a notation that we keep throughout the chapter, we now assume the prior distribution is defined on a set of discrete variables $\mathbf{z} \sim p_{\boldsymbol{\theta}}(\mathbf{z})$, with $\mathbf{z} \in \{0,1\}^L$. Given a discrete variable z with mean q and logit $l = \sigma^{-1}(q) = \log(q) - \log(1-q)$ (where $\sigma = 1/[1+\exp(-l)]$ is the sigmoid function), a non-differentiable implementation of Eq. 4.8 for discrete variables can be obtained as

$$z = \Theta[\rho - (1-q)] = \Theta[\sigma^{-1}(\rho) + l)] \,, \tag{4.9}$$

where Θ is the Heaviside function and the random variable $\rho \in [0,1]$ is distributed according to a uniform distribution $\mathcal{U}$. In the second equality, we have used the fact that the inverse sigmoid function is monotonic.

A continuous smoothing (also known as the Gumbel trick [98]), is performed by replacing the Heaviside function with the sigmoid function:

$$z = \Theta[\sigma^{-1}(\rho) + l] \rightsquigarrow \zeta = \sigma\left(\frac{\sigma^{-1}(\rho) + l}{\tau}\right) \,, \tag{4.10}$$

where τ is a temperature parameter introduced to control the smoothing. Typically, τ is annealed from large to low values during training. For large values of τ, the bias introduced by substituting $\mathbf{z}$ with $\boldsymbol{\zeta}$ everywhere in the loss function is large, but the gradients propagating through $\boldsymbol{\zeta}$ are also large, facilitating training. Conversely, for low values of τ the bias is reduced but gradients vanish and training stops. Evaluation of trained models is done in the limit $\tau \to 0$, where $\boldsymbol{\zeta} \to \mathbf{z}$.

Throughout this chapter, we will use BMs to provide powerful and expressive prior distributions defined on discrete variables:

$$\begin{aligned} p_{\boldsymbol{\theta}}(\mathbf{z}) &\equiv e^{-\mathcal{H}_{\boldsymbol{\theta}}(\mathbf{z})}/Z_{\boldsymbol{\theta}} \,, \quad Z_{\boldsymbol{\theta}} \equiv \sum_{\mathbf{z}} e^{-\mathcal{H}_{\boldsymbol{\theta}}(\mathbf{z})} \,, \\ \mathcal{H}_{\boldsymbol{\theta}}(\mathbf{z}) &\equiv \sum_{l} \sigma^z_l b_l + \sum_{l<m} W_{lm} \sigma^z_l \sigma^z_m \,. \end{aligned} \tag{4.11}$$

To train a VAE with BM prior, following the prescription of the previous section, we formally replace $p_{\boldsymbol{\theta}}(\mathbf{z}) \rightsquigarrow p_{\boldsymbol{\theta}}(\boldsymbol{\zeta})$. As usual, the gradients of the log-probability is given by the difference between a positive and negative phase:

$$\partial \log p_{\boldsymbol{\theta}}(\boldsymbol{\zeta}_{\boldsymbol{\phi}}) = -\partial \mathcal{H}_{\boldsymbol{\theta}}(\boldsymbol{\zeta}_{\boldsymbol{\phi}}) + \mathbb{E}_{\bar{\mathbf{z}} \sim p_\theta}[\partial \mathcal{H}_{\boldsymbol{\theta}}(\bar{\mathbf{z}})] \,. \tag{4.12}$$

In the equation above, we have highlighted the fact that the smoothed latent samples $\boldsymbol{\zeta}_{\boldsymbol{\phi}}$ depend on the variational parameters $\boldsymbol{\phi}$. The model samples $\bar{\mathbf{z}}$, however, remain discrete variables sampled from the BM, and are thus not smoothed during training [99].

4.2 Sampling with Quantum Annealers

Currently manufactured quantum annealers physically implement a transverse-field Ising model:

$$\mathcal{H}(s) = A(s)\sum_l \sigma_l^x + B(s)\left[\sum_l \sigma_l^z h_l + \sum_{l<m} J_{lm}\sigma_l^z\sigma_m^z\right] ,$$

where $s \in [0,1]$ is a control parameter, and $A(s)$ and $B(s)$ are respectively decreasing and increasing monotonic functions of the parameter s with $A(0) \gg B(0)$ and $A(1) \ll B(1)$. Quantum annealers operate immersed in a thermal environment. There is theoretical and numerical evidence [102–105] that when the anneal is performed sufficiently slowly the system above is in thermal equilibrium with the environment. This property can be exploited to turn quantum annealers into programmable Boltzmann samplers. Thermal relaxation rates are controlled by the intensity of the transverse field $A(s)$. At the beginning of the anneal, relaxation times are small, and the system proceeds through a sequence of thermal states. As the anneal proceeds, relaxation times become larger and eventually the state of the system freezes at the point s^* where relaxation times roughly become larger than the annealing time t_a.

With the above picture in mind, we can use quantum annealers to sample from the distribution:

$$\begin{aligned} p_{\boldsymbol{\theta}}(\mathbf{z}) &\equiv \mathrm{Tr}[\Lambda_{\mathbf{z}} e^{-\mathcal{H}_{\boldsymbol{\theta}}}]/Z_{\boldsymbol{\theta}}\,, \quad Z_{\boldsymbol{\theta}} \equiv \mathrm{Tr}[e^{-\mathcal{H}_{\boldsymbol{\theta}}}]\,, \\ \mathcal{H}_{\boldsymbol{\theta}} &= \sum_l \sigma_l^x \Gamma_l + \sum_l \sigma_l^z b_l + \sum_{l<m} W_{lm}\sigma_l^z\sigma_m^z, \end{aligned} \tag{4.13}$$

where $\boldsymbol{\Gamma}, \mathbf{h}, \mathbf{W} \in \{\boldsymbol{\theta}\}$, $\Lambda_{\mathbf{z}} \equiv |\mathbf{z}\rangle\langle\mathbf{z}|$ is the projector on the classical state $\mathbf{z}$, and $\sigma_l^{x,z}$ are Pauli operators, and:

$$\begin{aligned} b_l &= \beta^*_{eff} h_l, \quad W_{lm} = \beta^*_{eff} J_{lm}, \quad \Gamma_l = \beta^*_{eff}\Gamma^*, \\ \beta_{eff} &\equiv B(s^*)/\beta_{phys}, \quad \Gamma^* \equiv A(s^*)/B(s^*) \end{aligned} \tag{4.14}$$

as defined in Eq. 4.13. Advanced control techniques for the anneal schedule (such as pauses and fast ramps present in the latest generation of D-Wave quantum annealers) allow in principle to control the freezing point s^*. To perform such sampling we used the publicly available Solver API provided by D-Wave [106].

It is useful to point out now that knowledge of the effective transverse field Γ^* is unnecessary. As we will see below QBMs are trained via the Q-ELBO loss function, in which the transverse field does not appear explicitly, but only implicitly in sampling from the model. In the following we assume that for the models and datasets under consideration freezing happens late in the anneal. This means we effectively sample from a QBM that is very close to a classical BM. More specifically, we use quantum annealers to quantum-assist training of VAEs with classical latent-space BMs.

4.2.1 VAE hybridization with quantum prior

Once we have a framework to train VAEs with discrete latent variables, we can consider quantum-classical hybrid VAEs in which the generative process $\mathbf{z} \sim p_{\boldsymbol{\theta}}(\mathbf{z})$ is realized by measuring the computational basis on a given quantum state $\rho_{\boldsymbol{\theta}}$ realized by a quantum annealing process controlled by a set of parameters $\boldsymbol{\theta}$ we wish to adjust during training of the model.

As introduced in Ref. [101], a QVAE can be obtained by assuming the quantum state $\rho_{\boldsymbol{\theta}}$ is a thermal state of a transverse field Ising model; *i.e.*, a QBM [107]. The prior $p_{\boldsymbol{\theta}}(\mathbf{z})$ distribution is then given by Eq. 4.13. Unlike for a classical BM, the direct evaluation of the gradients of the distribution is intractable. As discussed in Ref. [107], a possible workaround is to perform the following substitution:

$$p_{\boldsymbol{\theta}}(\mathbf{z}) = \mathrm{Tr}[\Lambda_{\mathbf{z}} e^{-\mathcal{H}_{\boldsymbol{\theta}}}]/Z_{\boldsymbol{\theta}} \to \tilde{p}_{\boldsymbol{\theta}}(\mathbf{z}) = e^{-\mathcal{H}_{\boldsymbol{\theta}}(\mathbf{z})}/Z_{\boldsymbol{\theta}} \tag{4.15}$$

in the ELBO $\mathcal{L}$ to obtain the so-called quantum ELBO (Q-ELBO) $\tilde{\mathcal{L}}$. As a consequence of the Golden-Thompson inequality $\mathrm{Tr}[e^A e^B] \geq \mathrm{Tr}[e^{A+B}]$, one has:

$$p_{\boldsymbol{\theta}}(\mathbf{z}) \geq \tilde{p}_{\boldsymbol{\theta}}(\mathbf{z}) \quad \Rightarrow \quad \mathcal{L} \geq \tilde{\mathcal{L}}\,. \tag{4.16}$$

The Q-ELBO $\tilde{\mathcal{L}}$ is thus a lower bound to the ELBO with tractable gradients that can be used during training. The derivatives of the log-probabilities $\log \tilde{p}_{\boldsymbol{\theta}}(\mathbf{z})$ can be estimated via sampling from the QBM [107]:

$$\partial \log \tilde{p}_{\boldsymbol{\theta}}(\mathbf{z}) = -\partial \mathcal{H}_{\boldsymbol{\theta}}(\mathbf{z}) + \mathbb{E}_{\bar{\mathbf{z}} \sim p_\theta}[\partial \mathcal{H}_{\boldsymbol{\theta}}(\bar{\mathbf{z}})]\,, \tag{4.17}$$

where $\bar{\mathbf{z}}$ are the model samples distributed according to the quantum Boltzmann distribution. The use of the Q-ELBO and its gradients precludes the training of the transverse field $\mathbf{\Gamma}$ [107], which is treated as a constant (hyper-parameter) throughout the training. Training via the Q-ELBO is performed as in the BM case, by smoothing $\mathbf{z} \rightsquigarrow \boldsymbol{\zeta}$.

4.3 Training VAE with quantum annealers

We have implemented a convolutional VAE whose prior is implemented by a BM and sampling is offloaded to a D-Wave 2000Q quantum annealer. To improve the performance of the model, we use several techniques such as learning-rate and KL-term annealing, importance-weight annealing, convolution gating, and batch normalization. We give a detailed description of the model in Appendix A.1. In this section, we restrict ourselves to a Chimera structured restricted BM (RBM) with 288 latent units (a six-by-six patch of Chimera cells see Appendix A.4) and present our results with models trained end-to-end by using samples drawn with D-Wave 2000Q quantum annealers on the same MNIST dataset [47] we have already used in chapter 3. Samples used to estimate the negative phase (second term of Eq. 4.17) are obtained following the prescription given in Appendix A.2.

The effective temperature β^*_{eff} must be chosen appropriately to correctly train the parameters of the inference network $q_{\boldsymbol{\phi}}$ (see appendix A.3 for a more detailed discussion). The parameter β^*_{eff} can be considered as a multiplicative correction for the learning rate of the prior parameters $b, W \in \boldsymbol{\theta}$. However, this observation is not true for the inference parameters $\boldsymbol{\phi}$, whose gradients also propagate through the first term in Eq. 4.17 via $\boldsymbol{\zeta}(\boldsymbol{\phi}, \boldsymbol{\rho})$. Due to our simple forward-anneal schedule, we expect the value of β^*_{eff} to change during training. To account for this effect, here we employ a real-time β^*_{eff} estimation as explained in Appendix A.3. It should be possible, in the future, to train at a fixed-temperature with appropriate pause-and-ramp annealing schedules [108, 109].

Training is performed jointly on the parameters of the classical networks and on the parameters of the quantum device. The gradients of the latter parameters require estimation of the negative phase (a thermal expectation of the energy) in Eq. 4.17. At each gradient update, such expectations are computed using samples from the quantum annealer only, and do not involve any classical Gibbs sampling such as persistent contrastive divergence, or any classical post-processing of the samples obtained by the annealer. We typically trained our models for 2000 epochs and a batch size of 1000. Figure 4.2

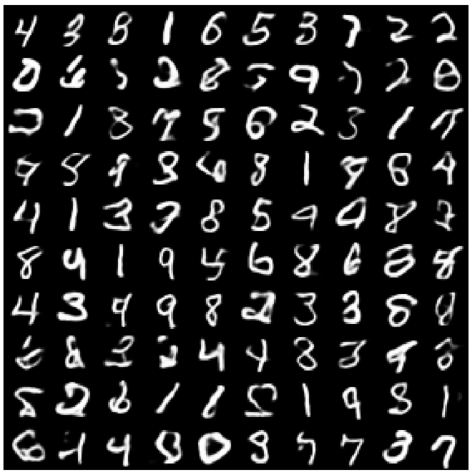
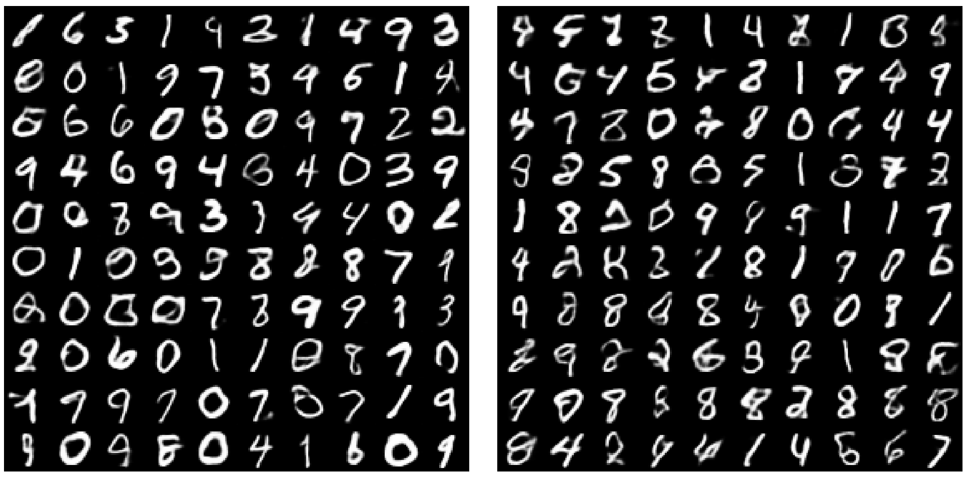


Figure 4.2: Images generated by sampling latent configurations with a quantum annealer that are subsequently transformed by a classical deconvolutional decoder. The classical networks and the quantum annealer weights have been trained end-to-end for 2000 epochs on the MNIST dataset using the quantum annealer as a sampler for estimating the gradients of the annealing parameters.

shows a set of images generated by a VAE trained end-to-end using a D-Wave 2000Q system. The set of images, obtained by generating latent samples $\mathbf{z}$ with the quantum annealer and subsequently decoded as $\mathbf{x} \sim p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$, shows a good amount of global consistency and consistent statistical variety.

4.3.1 Validation of training

In this section we give evidence that we have successfully exploited the Chimera-structured RBM prior in the latent space of our convolutional VAE. Validating the training of quantum-classical hybrid generative models can be nontrivial and must be assessed carefully, especially when training uses a large amount of classical processing. We trained the deep networks of our model using GTX 1080 Ti GPUs, and the quantum annealer is called only to estimate the negative phase. A principled validation strategy is thus to compare a model trained with quantum assistance to a fully classical baseline for which the quantum hardware is not required. In our case, a convenient baseline is a model that has the same classical networks but whose prior is



(a) Chimera (b) Bernoulli

Figure 4.3: Visual validation of training with Chimera connectivities is not conclusive.

trivial. As a trivial prior, we choose a set of independent Bernoulli variables, which is equivalent to an RBM with vanishing weights between latent units. For simplicity, in the following we refer to such prior as RBM with Bernoulli prior. While generative models with latent variables are also powerful tools to perform supervised and semi-supervised tasks, here we focus on training VAEs as purely generative models.

For image processing, comparison between different generative models could be done qualitatively by visually inspecting the generated samples. In our research however, visual comparison is inconclusive [110]. In Fig. 4.3, for example, we compare samples generated by a trained model with Chimera (Fig. 4.3(a)) and Bernoulli (Fig. 4.3(b)) priors. Both models have been trained by evaluating the negative phase with a D-Wave 2000Q system. The images are also generated using samples coming from the annealers. Given our specific implementation, it is difficult to discern an improvement of visual quality when using a Chimera-structured RBM rather than a Bernoulli prior. To overcome this difficulty, we evaluate quantitatively [1] the generative performance of VAEs by computing the ELBO defined in Eq. 4.3 or by esti-

[1]The existence of a well-defined loss function that allows quantitative validation is an important advantage of VAE, compared to other generative models such as Generative Adversarial Networks

mating the log-likelihood via an importance sampling technique as described in Ref. [111].

These quantities are however not accessible when training with analog devices as samplers. In fact, while we assume that samples generated by quantum annealers are distributed according to the required Boltzmann distribution, we must treat quantum annealers as black-box samplers during testing and validation. In other words, the log-probabilities for the quantum generative process $\log p_{\boldsymbol{\theta}}^{\mathrm{DW}}(\mathbf{z})$ must be assumed to be unknown. Therefore, we validate results by replacing the unknown hardware log-probabilities with those of an auxiliary RBM whose weights are given by the relations in Eq. 4.14 [2]:

$$\log p_{h,J}^{\mathrm{DW}}(\mathbf{z}) \rightsquigarrow \log p_{b,W}^{\mathrm{RBM}}(\mathbf{z})\,. \tag{4.18}$$

This approach can be more rigorously interpreted as validating the fully classical model in which we replace the quantum annealer by the auxiliary RBM defined by the relations above.

In Tab. 4.1 we report the LL of the auxiliary VAE with 288 latent unit on a Chimera connectivity trained with a D-Wave 2000Q quantum annealer. We compare it to the same model trained end-to-end with population annealing [101]. We also compare each model with its respective Bernoulli baseline. We have reported the mean and the standard error over 5 independent training runs. Training with a Chimera-structured RBM improves

MNIST (dynamic binarization) **LL**		
Sampler	Chimera	Bernoulli
DW2000Q	-82.8 ± 0.2	-83.7 ± 0.2
PA	-82.8 ± 0.1	-84.2 ± 0.05

Table 4.1: Log-likelihood of convolutional VAEs trained with samples coming from either D-Wave 2000Q or PA. All models share the same encoding and decoding networks, but are trained independently for 2000 epochs on the MNIST dataset.

significantly the log-likelihood over the Bernoulli baseline. Moreover, the models trained with quantum annealers achieved the same log-likelihood as

[2]The computation of the log-probabilities requires the estimation of the partition function, which is done using annealed population sampling as in Ref. [101]

the models trained with PA. Notice that each model and its baseline employed exactly the same amount of classical computational resources. Models trained with structured RBMs achieve better performance by requiring thermal sampling, a computational task that can be offloaded to a quantum annealer.

In general we would like to use quantum annealers to sample from the trained generative model. As explained above, treating quantum annealers as black-boxes means we cannot quantitatively evaluate such a model. However, we argue that the log-likelihood of such a VAE is likely very close to that of the auxiliary VAE. After all, the training assumes the hardware samples are distributed according to a Boltzmann distribution of the auxiliary RBM, and in the previous section we have shown that this assumption is accurate enough to correctly train the auxiliary RBM. We can confirm this visually in Fig. 4.4. On the left panel we show a set of digits generated by sampling from a D-Wave 2000Q. On the right panel we use the same trained model but sample from a D-Wave 2000Q after setting its weights to zero. We see that while the annealer still generates plausible digits, it does not generate a number of digits with the correct statistics (in the right panel of Fig. 4.4, digits 9 and 4 seem to dominate the scene). This gives evidence that D-Wave 2000Q quantum annealers sampled consistently, such that the classical networks were able to correctly learn the correlations between latent units existing due to the RBM with non-vanishing weights.

4.4 A path towards quantum advantage with VAE

In the previous section we have shown that it is possible to use quantum annealers as Boltzmann samplers to train RBM-structured priors placed in the latent space of deep convolutional VAE. In the experiments presented, we have settled on relatively small, Chimera-structured RBMs with 288 latent units. We found that larger RBMs did not appreciably improve performance of the overall VAE when training on the MNIST dataset. We will explain why this is the case in this section. Sampling from RBMs with a few hundred units can still be done classically with relative ease, and the natural question that arises is whether our approach offers a path towards obtaining quantum advantage with quantum annealers in machine learning applications. To

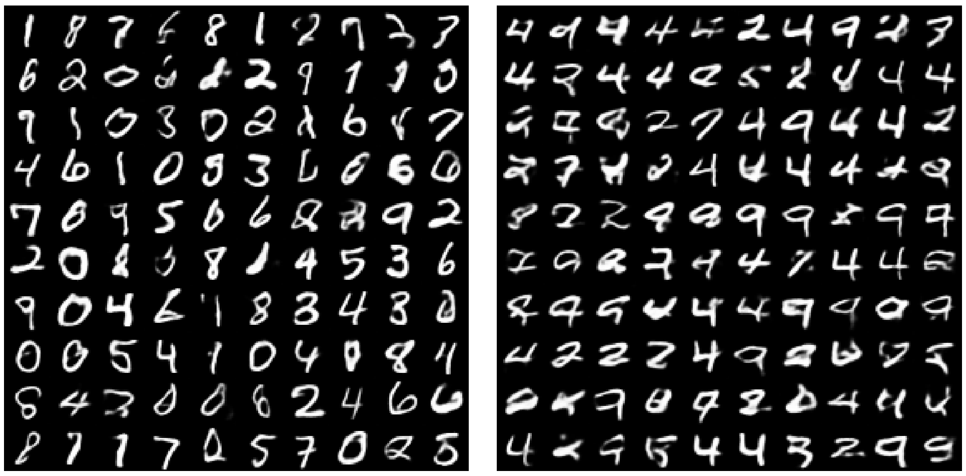


Figure 4.4: Left panel: Samples generated by D-Wave 2000Q using a fully trained model. Right panel: Samples generated by D-Wave 2000Q after setting the couplings of the annealer to zero.

achieve such a goal, we need to exploit large latent-space RBMs that develop complex multimodal probability distributions (*i.e.*, a complex energy landscape) from which sampling is classically inefficient. We give evidence that this is indeed possible. For example, we demonstrate that the BMs placed in the latent space develop nontrivial modes that are likely to cause classical Monte Carlo algorithms to have long mixing times. Moreover, we show that training on more complex datasets likely takes advantage of larger BMs to improve performance. In addition, we discuss the role of connectivity, emphasizing its importance even in this hybrid approach, and the necessity to develop device-specific classical NN to better exploit physical connectivities such as the Chimera graph.

In the next sections, we give evidence of the existence of a natural path towards obtaining quantum advantage by applying quantum annealing to generative modeling within the proposed VAE framework.

4.4.1 Exploit large latent-space RBMs

Exploiting a larger number of latent units to improve the generative performance of a VAE is a popular and active research area. One known obstacle in achieving this is the loss function used for training. We rewrite the ELBO here for convenience by highlighting its two terms:

$$\mathcal{L}(\mathbf{x}, \mathbf{z}, \boldsymbol{\phi}) = \underbrace{\mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})]}_{\text{autoencoding term}} - \underbrace{D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\mathbf{z}}(\mathbf{z}))}_{\text{KL}-\text{regularization}} . \quad (4.19)$$

The first term is sometimes called the "autoencoding" term and can be thought of as a reconstruction error: an encoded latent configuration ζ is first sampled from the encoder $q_{\boldsymbol{\phi}}(\boldsymbol{\zeta}|\mathbf{x})$ and is subsequently decoded by $p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{\zeta})$. When both approximating posterior and marginal distributions are factorized distributions, this term can be also interpreted as a reconstruction error. Maximizing the ELBO results in maximizing this term, which tends to maximize the number of latent units used to prevent information loss during the encoding-decoding steps. The second term, the KL divergence between the approximating posterior and the prior, has the effect of a regularization term and it is sometimes also called KL regularization. Maximizing the ELBO results in minimizing the KL term, which pushes the approximating posterior close to the prior. This also means the approximating posterior depends less sharply on the inputs $\mathbf{x}$. In the case of factorized distributions, this usually means some latent units are conditionally independent from the input ("inactive units"): $z_{inact} \sim q_{\boldsymbol{\theta}}(z_{inact}|\mathbf{x}) = q_{\boldsymbol{\theta}}(z_{inact})$.

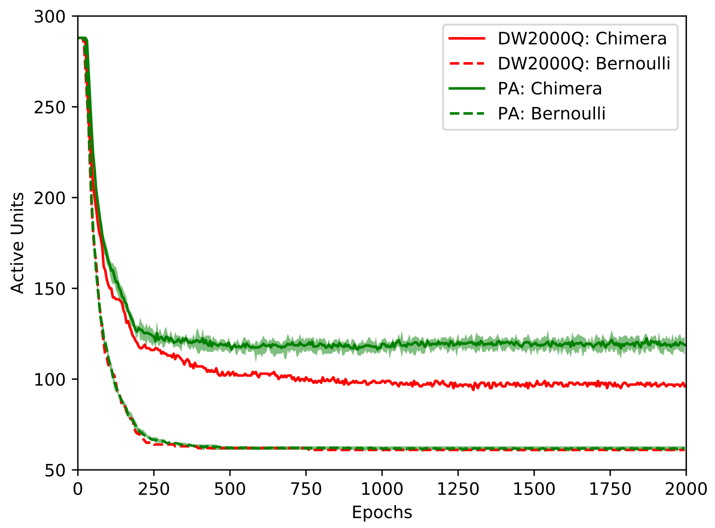


Figure 4.5: Comparison of active units during training between a D-Wave 2000Q and population annealing on both Chimera and Bernoulli priors.

The balance between the autoencoding and KL terms means using VAE can be efficient at lossy data compression by using the right number of latent units. However, the tension between KL and autoencoding terms results in an optimization challenge: during training the model is usually stuck in a local minimum with a suboptimal number of latent units. The main takeaway, for our purposes, is that the number of latent units effectively used is highly dependent on the model, the optimization technique, and the training set. To exploit a larger RBM, one thus has to work on all these elements. In Fig. 4.5 we show the number of active units during a training run of 2000 epochs when the models are trained with either PA or D-Wave 2000Q, with either a Chimera-structured RBM or a Bernoulli prior. Lines (bold or dashed) are the means over 5 independent runs while light-color areas delimit the smallest and largest values among the 5 runs. To identify whether a latent unit is active or not, we compute the variance σ of the value of each unit z over the test set and we set a threshold of $\sigma > 0.01$ as definition of an active unit.

Figure 4.5 shows several key points that we will expand upon in the next sections. First, it shows the use of a KL annealing technique: the KL term is turned off at the beginning of the training and it is slowly (linearly) turned on within 200 epochs. The figure clearly shows the effect of the KL term in shutting down a large number of active units. Since the number of active units plateaus around a number much lower than 288, using a larger RBM usually does not improve performance of our implementation on MNIST. It also shows that connectivity of the RBM plays a major role in determining the number of active units, which is much higher with a Chimera-structured RBM. Notice also that in the Bernoulli case, both samplers (PA and D-Wave 2000Q) train a model that uses a very similar number of latent units. However, when sampling is nontrivial (as in a Chimera-structured RBM) the model trained with the quantum annealer uses a number of units larger than the Bernoulli case, but smaller than the model trained with PA. This is a manifestation (which we will discuss later) of biased sampling with the quantum annealer: sampling quality is good enough to train the model (indeed we obtained a log-likelihood as good as that of the model trained with PA) but exploits a smaller number of latent units.

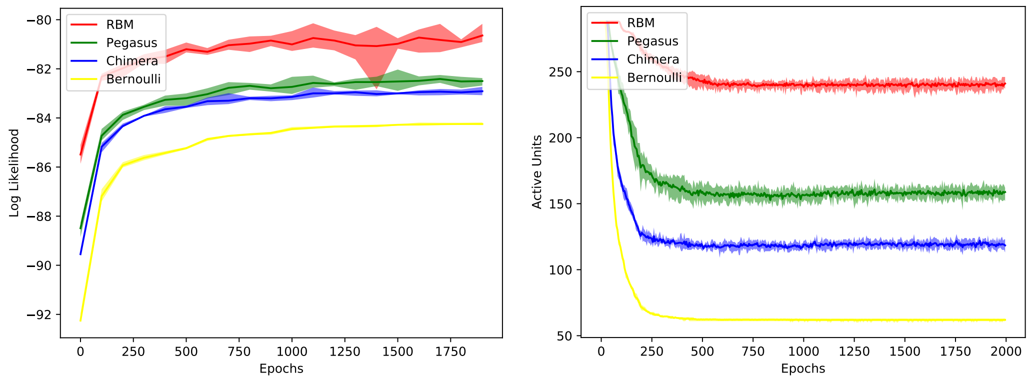


Figure 4.6: Comparison between VAE using different latent-space RBM connectivities and same convolutional networks. Both log-likelihood (left panel) and latent-space utilization (right panel) improve with more densely connected RBM. Performance on Pegasus graph increases despite the development of an architecture fairly optimized for Chimera graphs.

In the next three sections we discuss three important elements that would allow to build quantum-classical hybrid VAEs that can effectively exploit large latent-space RBMs. This is a necessary condition to search for quantum advantage in these models: speed-up and scale-up sampling from large RBMs using quantum annealers rather than inefficient classical sampling techniques.

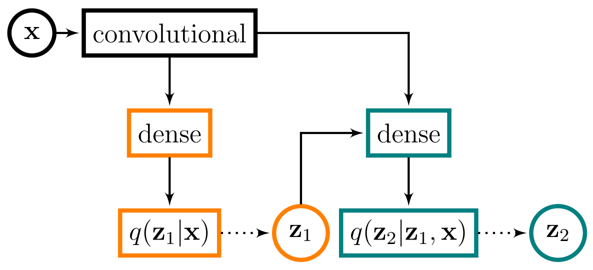
Denser connectivities

Connectivity of the RBM in the latent space plays an important role in determining both performance of the generative model and number of active latent units. While these two elements are not directly related, we typically observe a correlation between them. In this section, we investigate in more detail the effects of implementing denser connectivities by performing numerical experiments in four different cases: Bernoulli prior and Chimera, Pegasus, and fully connected RBM. Together with Bernoulli and RBM, we pick the connectivities of currently available D-Wave 2000Q quantum annealers and D-Wave's next-generation Pegasus architecture [112] (see also Appendix A.4).

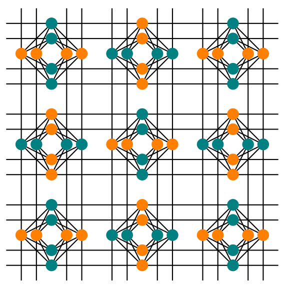
In Fig. 4.6 we compare the log-likelihood and the number of active units along training runs obtained using the same classical networks but using a different connectivity for the latent-space RBM prior. At each step during the training run, we employed roughly the same amount of classical resources for back-propagation. Unsurprisingly, using a more capable (dense) RBM results in better generative performance (log-likelihood) of the model (left panel)). It also results in using a larger number of latent units (right panel). Notice how Bernoulli and Chimera priors effectively use a number of latent units well below 150, while Pegasus and fully connected use well above 150. Because of this, we could not improve generative performance of Bernoulli and Chimera models by just using larger graphs, at least when training on MNIST. On the other hand, using larger Pegasus and fully connected RBMs would likely have improved the log-likelihood of the model. In Fig. 4.6 we show results on the same number of latent units (288) for proper comparison.

Working with VAEs allows us to easily take advantage of new connectivities without having to implement a new convolutional VAE. In fact, our model has been fairly optimized to improve performance on Chimera graphs (see the next section). Despite this architecture-specific optimization, just using the denser Pegasus graphs improve performance of the model. Moreover, the flexibility of the VAE hybrid approach allows us to easily adapt the implementation to the slightly different working graphs of different processors with different active/inactive qubits. By using the same implementation, during training the model naturally learns to deactivate latent units corresponding to uncalibrated qubits. We have indeed seamlessly used the same model to train on different D-Wave 2000Q processors, as well as using different groups of qubits within the same processor. In no case was a hard-coded connectivity (which would change for each processor) necessary.

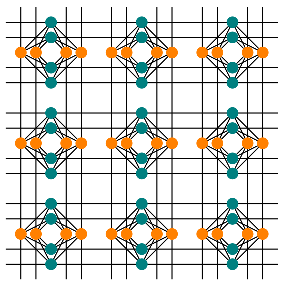
The results of Fig. 4.6 show that developing quantum annealers with denser connectivities (such as Pegasus) naturally leads to exploiting larger latent-space RBMs, possibly getting us closer to a regime where quantum advantage is possible. Indeed we expect that for a VAE trained on a complex dataset, requiring a high amount of latent units (in the order of 10^3) we would see a substantial advantage in sampling the latent space with a quantum annealer.


$\mathbf{x}$
convolutional
dense
dense
$q(\mathbf{z}_1|\mathbf{x})$
$\mathbf{z}_1$
$q(\mathbf{z}_2|\mathbf{z}_1,\mathbf{x})$
$\mathbf{z}_2$

(a) Conditional posterior



(b) Bipartite mapping



(c) Chains mapping

Figure 4.7: Implementation of a conditional approximate posterior (left), with two possible mappings between groups.

Hardware-specific optimization of classical networks

In our implementation, we have used fairly conventional convolutional neural networks. As we will discuss in this section, we have implemented only one specific architecture-dependent element in the encoder network that turned out to be very effective at improving performance on the Chimera graph. In general, we believe more elaborate, hardware-specific implementations of both the approximating posterior $q_{\phi}(\mathbf{z}|\mathbf{x})$ and the marginal distribution $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ will significantly improve performance of VAE trained with analog devices with quasi two-dimensional connectivities. This is not just a problem of building a model with more capacity. As we discussed in the case of latent-unit use, it is also a problem of optimizing the model hyperparameters. When working with sparsely connected RBMs, it is easier for the KL term to push both approximating posterior and RBM prior to a local minimum of the loss function in which they are both trivial. Developing hardware-specific hybrid models will thus also aim at reaching local minima during training in which the RBM prior is as expressive as possible, exploiting the largest amount of correlations between latent units.

In the context of hybrid VAE models trained with quantum annealers, the considerations above could replace more standard mapping techniques used in the quantum annealing community such as minor embedding [113, 114] and majority voting. The latter techniques typically require a hard-coded specification of the hardware connectivity of each processor, which makes adapting the code to different processors cumbersome. Our perspective in the contest of hybrid generative modeling is to work by adapting classical networks using a high-level specification of the connectivity and letting the model, through stochastic gradient descent, learn the details of the connectivity of each processor (such as the locations of uncalibrated qubits).

We now discuss an example of how the classical networks can be optimized for a given architecture (in our case the Chimera graph). A common technique to implement a more expressive approximating posterior $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ (with a tighter variational bound) is to introduce conditional relationships among latent units, also called hierarchies. In the case of two hierarchies, we first define an approximating posterior for a subset of latent units $\mathbf{z}_1$ and sample from it, then define a second approximating posterior for the remaining latent units $\mathbf{z}_2$ that depends conditionally on both the input data and the sampled values of the first group of latent variables:

$$\mathbf{z}_1 \sim q_{1,\boldsymbol{\phi}}(\mathbf{z}_1|\mathbf{x})\,, \quad \mathbf{z}_2 \sim q_{2,\boldsymbol{\phi}}(\mathbf{z}_2,|\boldsymbol{\zeta}_1,\mathbf{x})\,. \tag{4.20}$$

The schematic of our implementation is shown in Fig. 4.7(a). We notice that models with a large number of hierarchies (possibly as large as the number of latent units) are possible. Such models, also referred to as autoregressive models [115], are very powerful but have inefficient inference, since sampling must be performed sequentially and cannot be parallelized on modern GPU hardware.

The two hierarchical groups $(\mathbf{z}_1, \mathbf{z}_2)$, as well as the physical qubits on the Chimera connectivity, are not equivalent. We can thus build different models by simply choosing the mapping between the two hierarchical groups and an arbitrary bipartition of the physical qubits. Notice that training and deploying each of these models will involve exactly the same amount of classical and quantum computational resources. In our experiments we consider two possible mappings of the two hierarchical groups onto the physical qubits of the Chimera graph, which we call “Bipartite” and “Chains”. The Bipartite mapping corresponds to the bipartite structure of the Chimera graph (see Fig. 4.7(b)). The Chains mapping corresponds to the vertical and horizontal physical layout of qubits of the Chimera architecture (see Fig. 4.7(c)). The identification of vertical and horizontal chains of qubits is commonly used to perform a minor embedding of a fully connected RBM on the Chimera graph. We stress again, however, that we never perform any minor embedding, and we always sample from the native Chimera graph in all cases.

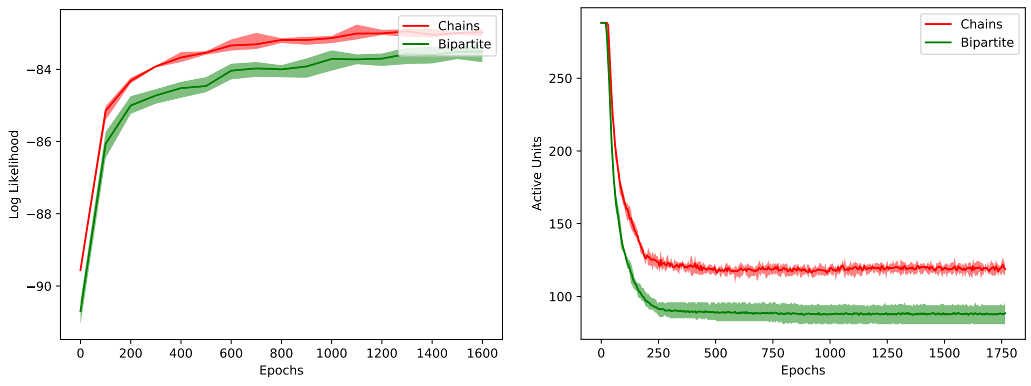


Figure 4.8: Left panel: Log-likelihood comparison for chains and bipartite mappings. Right panel: Active units comparison for chains and bipartite mappings.

Comparative results of the two mappings, obtained with classical sampling, are shown in Fig. 4.8. In the left panel we see that there is a sizeable difference in generative performance between the two mappings, with the Chains mapping performing remarkably better. This better performance is also reflected in the much higher number of latent units exploited by the Chains mapping (see right panel of Fig. 4.8). An intuitive explanation of the results above can be given as follows. Let us first write the KL term as:

$$\begin{aligned} D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z})) &= D_{KL}(q_{1,\boldsymbol{\phi}}(\mathbf{z}_1|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}_1)) + \\ &+ D_{KL}(q_{2,\boldsymbol{\phi}}(\mathbf{z}_2|\mathbf{z}_1,\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}_2|\mathbf{z}_1))\,. \end{aligned} \tag{4.21}$$

In the Bipartite mapping, the conditional $p_{\boldsymbol{\theta}}(\mathbf{z}_2|\mathbf{z}_1)$ has a simple form that can be computed analytically due to the bipartite structure of the Chimera RBM. During training, it is very easy for the model to use the capacity of $q_{2,\boldsymbol{\phi}}(\mathbf{z}_2|\mathbf{z}_1, \mathbf{x})$ to match the simple prior marginal $p_{\boldsymbol{\theta}}(\mathbf{z}_2|\mathbf{z}_1)$ and be independent from $\mathbf{x}$. As a consequence, a large portion of the representational capacity of the approximating posterior is wasted in representing the simple marginal $p_{\boldsymbol{\theta}}(\mathbf{z}_2|\mathbf{z}_1)$. In the Chains mapping, in contrast, the marginal $p_{\boldsymbol{\theta}}(\mathbf{z}_2|\mathbf{z}_1)$ is nontrivial, and the approximating posterior $q_{2,\boldsymbol{\phi}}(\mathbf{z}_2|\mathbf{z}_1, \mathbf{x})$ has more difficulties in matching it and decoupling $\mathbf{x}$. As a consequence, the model ends up using more efficiently all the variational parameters $\boldsymbol{\phi}$. This is another manifestation of the optimization challenge present with VAE models mentioned before, which in this case it is exploited to find better local minima of the loss function.

In this section we have shown how a simple architecture-aware modification of the encoder network allows us to train better models and to exploit a given architecture more efficiently. This architectural work done on the VAE model has also produced, as a side effect, a new state-of-the-art classical VAE architecture [116]. We expect that architecture-aware model-engineering will be crucial to fully exploit large physical connectivities in the latent space of VAE.

Training on larger datasets

Implementing more highly connected RBMs and developing classical encoders and decoders tailored to a given connectivity can only go so far in helping to exploit larger latent spaces. Together with other techniques such as KL-term anneal, the ideas mentioned in the previous two sections help reduce the pressure of the KL term to reach suboptimal local minima. In essence, VAE are also efficient lossy encoders. An alternative direction to increase latent space utilization is thus to train on more complex datasets. By doing so, a larger number of latent units is necessary to store enough information such that the reconstruction term (first term in Eq. 4.19) is large enough.

We give numerical evidence of the intuition above by training the same VAE models used in the previous sections on the Fashion MNIST (FMNIST) dataset. A set of images from the FMINST dataset is shown in the left panel of Fig. 4.9. FMNIST is the same size as MNIST (50000, 28×28 images) and has the same number of classes. However, its images are more complex with more fine details, including grey-scale features that are important for correct image classification (whereas MNIST digits are substantially black and white). In the right panel of Fig. 4.9, we train the same models on MNIST (shaded) and FMNIST (solid) and compare the number of active units during training. We see that, apart from the case with fully connected RBMs, all other models use a substantially larger number of latent units.

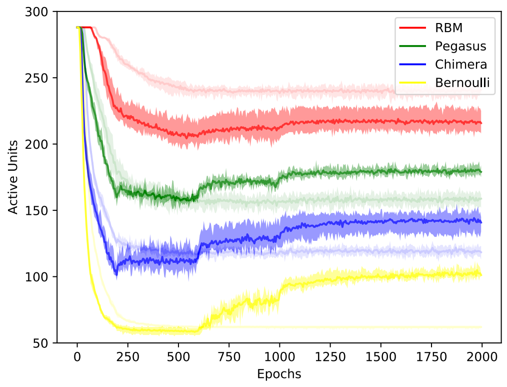


Figure 4.9: Left panel: Fashion MNIST dataset. Right panel: Active units for the same VAE models trained on FMNIST (solid colors) and on MNIST (shaded colors). All models with sparse latent connectivities use a much larger number of latent units when trained on the more complex dataset.

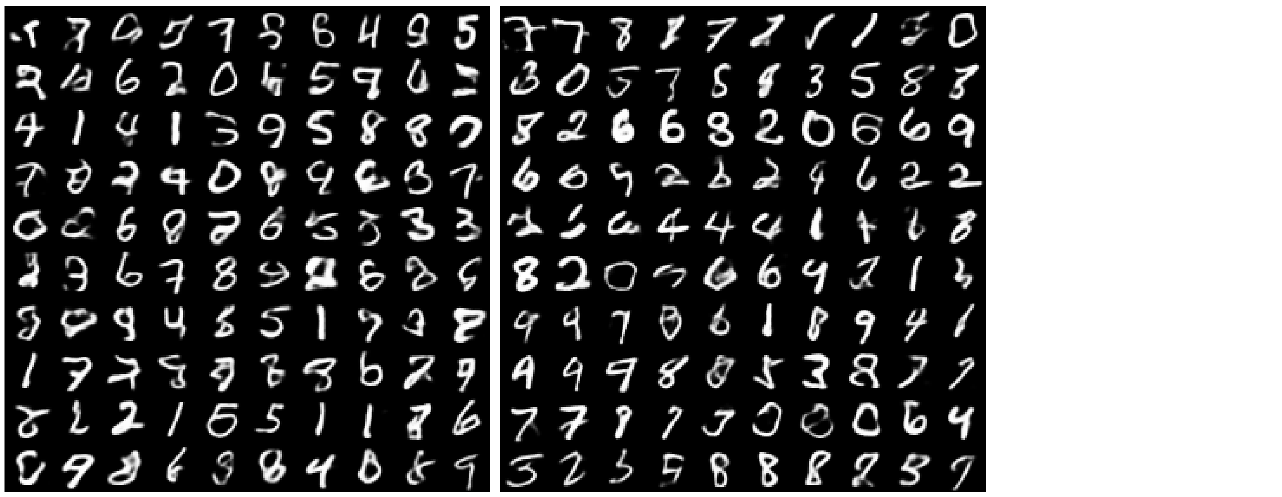
4.4.2 Multi-modality of latent-space RBMs

Exploiting large latent-space RBMs is a necessary condition to eventually achieve quantum advantage when sampling with quantum annealers. This condition is however not sufficient. The typical computational bottleneck in training an RBM is due to the appearance of well-defined modes. These modes make classical sampling techniques inefficient and slow-mixing, resulting in highly correlated samples used both during training and generation. While making sampling harder, the development of multi-modal distributions is actually an appealing property of RBMs, since it allows such models to represent complex and powerful probability distributions. The idea behind searching for quantum advantage in training RBM with quantum annealers is, indeed, to exploit quantum resources (such as tunneling) to more efficiently mix between different modes in the landscape defined by the RBM.

When trained on visible data, an RBM naturally develops complex landscapes to match the complexity of the statistical relationship present in the training data. However, while RBMs trained on latent representations can potentially develop well-defined modes, they do not necessarily do so. In fact, one of the capabilities of generative models with latent variables is finding a set of statistically independent latent features [117]. This is typically enforced during model building by using trivial priors such as the product of independent Gaussian (for continuous latent units) or the product of independent Bernoulli (for discrete latent spaces). Even when the prior is potentially complex and trainable, as an RBM, the presence of the KL term can push the model during training into local minima in which the trained RBM develops a trivial landscape.

In this section we give evidence that RBMs trained in the latent space of a VAE model do indeed develop a nontrivial landscape with well-defined modes. As we have shown in Sec. 4.4.1 (see Fig. 4.6), RBMs with denser connectivities naturally lead to better performing VAEs. This is an indirect indication that we are indeed exploiting the additional capacity and expressivity of more connected RBMs. In this section we give more explicit evidence of this. In Fig. 4.10, we generate a sequence of images via block Gibbs sampling. The top left image is generated by picking a latent configuration $\mathbf{z}$ out of a uniform distribution over all configurations. This latent sample is then sent through the decoder. Going from left-to-right, top-to-bottom, each subsequent image is obtained after updating all latent units with a sequence of block Gibbs updates (one for Bernoulli, two for the bipartite connectivities Chimera and RBM, four for the quadripartite Pegasus connectivity). As expected, in the Bernoulli case (Fig. 4.10(a)), each update results in uncorrelated samples. The Chimera connectivity (Fig. 4.10(b)) is able to develop weakly correlated samples, as shown by short sequences of similar images. Correlated samples with well-defined modes are more clearly visible with the Pegasus connectivity ((Fig. 4.10(c))). Finally, we confirm that increasing the connectivity up to a fully connected RBM (Fig. 4.10(d)) results in long sequences of correlated samples and related to the deep valleys of the RBM energy landscape.

The results shown in Fig. 4.10 show that RBMs trained as priors of generative models with latent variables naturally learn multi-modal, nontrivial probability distributions. These distributions are expressive, making the


(a) Bernoulli. (b) Chimera.

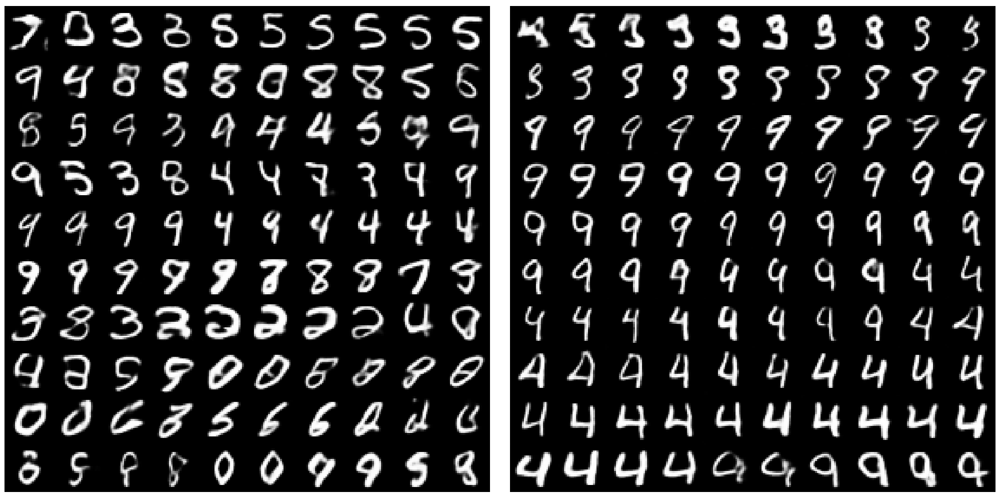

(c) Pegasus. (d) RBM.

Figure 4.10: Block Gibbs sampling with different connectivities. Going from left to right, denser connectivities result in more well-defined modes developed in the trained RBM. Especially in the case of Pegasus and an RBM, for example, it is clearly visible how the block Gibbs chain is trapped in a typical basin of the landscape for MNIST connected to the digits 4 and 9.

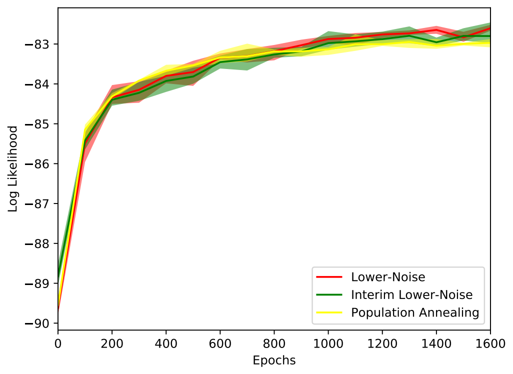
whole VAE more expressive, while at the same time developing the same types of computational bottlenecks that make classical sampling algorithms inefficient. This paves the way to effectively use quantum annealers as means to more scalable quantum-assisted sampling, enabling us to sample from RBMs of sizes and complexity that are infeasible with classical methods.

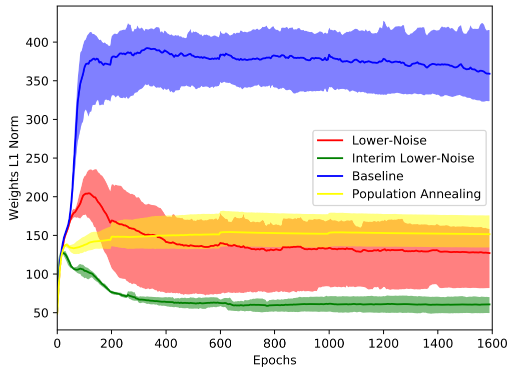
4.4.3 Robustness to noise and control errors

Using quantum annealers to train large RBMs directly on complex data remains challenging. Apart from the unsatisfying performance of using RBMs with quasi two-dimensional connectivities on visible data, a major difficulty is biased sampling (and thus inaccurate gradients) obtained with quantum annealers. There are two main sources of bias: control errors and imperfect or incomplete thermalization at the freezing point. While the latter can be improved with appropriate pause-and-ramp annealing schedules, the former can only be improved with technological advancements. Despite these known difficulties, we have shown in the previous sections we have successfully trained large RBM (hundreds of units) in the latent space of a VAE solely using samples coming from a D-Wave 2000Q, without using any hard-coded pre or post-processing to the raw data obtained from the annealer.

We interpret our positive results as an indication that training RBMs with our setup is relatively robust to noise and control errors. In fact, we can interpret both the encoder and decoder as powerful tools to pre- and post-process data to be sent to the quantum annealer. Using stochastic gradient descent, we train the encoder and decoder to generate a set of latent features that are more easily modeled by the latent-space RBM. For example, real images might have strongly correlated, sharp features (such as regions with black or white pixels), which require large weights to be modeled correctly. A precise implementation of such large weights might be challenging for analog devices with finite range such as quantum annealers. Additionally, both encoders and decoders might be able to learn and correct, or at least reduce the effects of, systematically biased sampling.

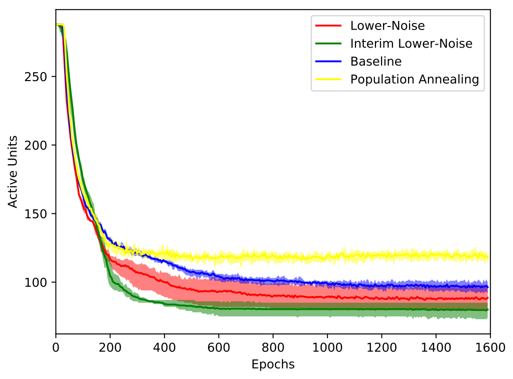
To investigate the role of noise and control errors in determining sampling quality and performance of the trained models, we perform a set of comparative experiments in which we train the same model on MNIST dataset, using samples coming from three D-Wave 2000Q with different noise profiles. The Baseline and Lower-Noise D-Wave 2000Q are both publicly available on D-

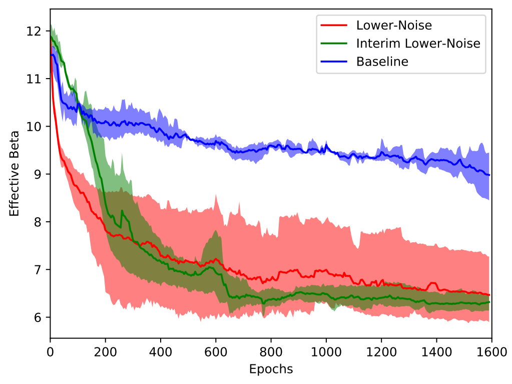




(a) Models trained on different D-Wave 2000Q achieved performance comparable to training with population annealing. Log-likelihood evaluation for Baseline D-Wave 2000Q did not converge due to diverging weights.

(b) The weights of the Baseline D-Wave 2000Q weights diverge after about 100 epochs. For the Interim Lower Noise processors weights are suboptimally small. Weight size of the Lower Noise D-Wave 2000Q is closest to population annealing.





(c) The Lower-Noise D-Wave 2000Q is able to exploit a larger number of latent units than the more noisy Interim Lower-Noise D-Wave 2000Q.

(d) Different temperature profiles during training for the three D-Wave 2000Q.

Figure 4.11: Training on different quantum annealers with different noise profiles.

Wave's Leap$^{\text{TM}}$ cloud service. We have also included an Interim Lower-Noise processor with an intermediate noise profile that is internally available at D-Wave.

Results are shown in Fig. 4.11. In Fig. 4.11(a) we report the log-likelihood during training. Models trained on the Lower and Interim Lower-Noise D-Wave 2000Q achieved performance comparable to training with population annealing. The evaluation of the log-likelihood for the Baseline D-Wave 2000Q diverged due to diverging weights, as can be seen in Fig. 4.11(b). The weights of the Baseline processor start diverging after about 100 epochs. The Interim Lower-Noise processor shows an opposite behavior, with weights getting small and plateauing after about 500 epochs. For the Lower-Noise processor, the L1 of the weights plateaus at a value that is closer to that obtained with "noiseless" population annealing. Notice that a consistent comparison in Fig. 4.11(b) we have reported the weights W rescaled by the effective temperature (see Eq. 4.14, and not the "bare" annealing values J. Figure 4.11(b) highlights the remarkably different response of three different quantum annealers to our model. Despite such differences, the performance of our hybrid implementation is robust (as shown in Fig. 4.11(a)) and does not require any hardware-specific adaptations or fine-tuning. Only while training with the Baseline D-Wave 2000Q, we needed a more aggressive clipping (that is restricting the weights and biases to have narrower range than the maximum allowed) to achieve similar performance and converged log-likelihood estimation.

Noise and control errors also manifest in a less efficient use of the latent space, as seen in Fig. 4.11(c). All models trained with D-Wave 2000Q use fewer active units than population annealing. Since the estimate of the log-likelihood of the models trained with the Baseline processor did not converge, we focus on the comparison between the Interim and Lower Noise processor: the latter can exploit a larger number of latent units. We finally show in Fig. 4.11(d) the profile for the extracted effective temperature during training, which is remarkably different for the three D-Wave 2000Q. The results shown in Fig. 4.11(d) underlines the importance of developing advanced annealing techniques to stabilize temperature during training.

In this section we have demonstrated the robustness to noise of our implementation, as highlighted in Fig. 4.11(a). At the same time, Fig. 4.11(c) shows an important effect of noise, which is to make it harder for our hybrid model to exploit the optimal number of latent units. We thus anticipate that exploiting large RBMs (with thousands of units, eventually) following the directions indicated in the previous sections must be accompanied by continued efforts in reducing sampling bias due to noise and control errors of future-generation quantum annealing devices.

4.5 Conclusions

In this chapter, we have demonstrated the use of quantum annealers (specifically D-Wave 2000Q) as Boltzmann samplers to estimate the negative phase of classical RBMs placed in the latent space of deep convolutional variational autoencoders. This setup allows for the construction of quantum-classical hybrid generative models that can be scaled to large, realistic datasets. We have mostly experimented with MNIST, a common testbed dataset which includes $60,000$, 28×28 binarized handwritten digits to achieve a log-likelihood of about -82.2 ± 0.2 nats, which compares favorably to state-of-the-art achieved with autoregressive models (-78.5 nats (natural unit of information)). In addition to demonstrating scalability and performance, we have discussed several other features of our hybrid approach.

First, we are able to use quantum annealers as “native samplers”, that is, samplers from their native graph: we do not use any hard-coded encoding-decoding scheme such as minor embedding or majority vote. Arguably this is one of the most effective ways to exploit the computational capabilities of quantum annealers. The encoding and decoding process is indeed efficiently performed by deep convolutional networks, which are trained to extract relevant feature via stochastic gradient descent. As we have shown, this approach is particularly flexible, since it naturally adapts to different connectivities and arbitrary working graphs.

Second, by successfully training the same model on three quantum annealers with different noise profiles, we have shown that our implementation is fairly robust to noise and control errors. Indeed, the deep convolutional networks can be seen as learned pre- and post-processing steps that regularize both the visible data and the effects of noise. A key reason of the success of our implementation is indeed the fact that the weights and biases as implemented on the quantum annealers rarely grow (during training) beyond their allowed range, even with minimal or no regularization. This result is to be contrasted to training RBMs directly on visible data, for which weights are typically much larger and regularization is critical to avoid overfitting.

The latter case is much more challenging for analog devices with limited allowed range.

This application, albeit not showing any definitive advantage or speedup, is an example of how new physical hardware can improve ML capabilities in the future. Here we focused on quantum hardware but efforts are being made also in the classical domain, for example with optical computing devices or special purpose electronic devices.

Chapter 5

Quantum Machine Learning

In this final chapter we will focus on two applications of Machine Learning in the quantum domain, putting them under the generic umbrella of “Quantum Machine Learning” models. In the first part we will consider the implementation of a simple classification model on different quantum platforms. This model differs from the one we have just discussed in chapter 4 for two main reasons: first is a supervised classification task instead of an unsupervised generative model and second, and most importantly, it is a *completely* quantum model with no external GPUs or classical post-processing required to obtain the results. The other application is about applying a Reinforcement Learning model to the stochastic quantum walk on a maze. Having only partial information about the system we will see how ML can find ways to further improve the exit probability of the quantum walker using few simple actions. This new framework also has some potential to be applied in future devices for example to improve energy or information transfer in the quantum domain.

5.1 Quantum Embeddings

To classify big data it is usually required to map them into new data clusters that later can be more feasibly and possibly linearly separated by well-trained artificial neural networks. However, in most cases the huge amount of data needs to be pre-processed in a very clever way in order to more feasibly apply the available machine learning algorithms on the post-processed data. For instance, in classification problems the main goal is so classify the original data into different groups with a tight border among them. In the case of a two-class classification problem, the simplest scenario is to find a classifier by splitting the high-dimensional space of the input data with a hyperplane. It is named as linear classifier. Then, all points living on one side of the hyperplane are classified as 'group A', while the others as 'group B'. Let us point out that this leads to a new metric in the new data space that faithfully reproduces the unknown, and usually much more complex, metric of the original data (e.g. the human-perceived "distance" between cat and dog pictures) where the classifier would be highly non-linear (hence more difficult to be found).

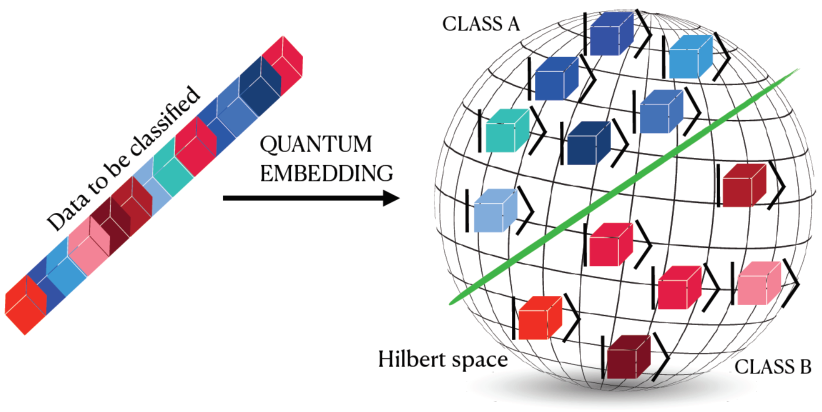


Figure 5.1: Pictorial view of the quantum embedding process where classical data (originally in a high-complex set) can be embedded into the larger (Hilbert) space of quantum states in order to be mapped into tight, more distant and linearly separable clusters and then be later (quantum) classified into different groups. We have implemented several experimental tests (based on quantum optics, IBM and Rigetti superconducting systems) of this quantum embedding, originally proposed in Ref. [118], in order to exploit their complementarity that is really crucial towards practical and more feasible hybrid quantum technologies [119].

Here, we investigate a scheme, originally proposed in Ref. [118], where classical data are embedded (as in Fig. 5.1) into a larger quantum (Hilbert) space describing the state of a physical system in the microscopic world where classical mechanics laws fail. We implement these ideas by engineering an experimental platform, based on quantum optics and publicly available Noisy Intermediate Scale Quantum (NISQ) processors, where we adapt and numerically optimize the quantum embedding protocol by machine learning methods, and test it for some validation data on the various platforms. Therefore, we will see that the quantum embedding approach successfully works also at the experimental level and, in particular, we show how different platforms could work in a complementary fashion to achieve this task. These studies might pave the way for future investigations on quantum machine learning techniques especially based on NISQ technologies.

5.1.1 Theory

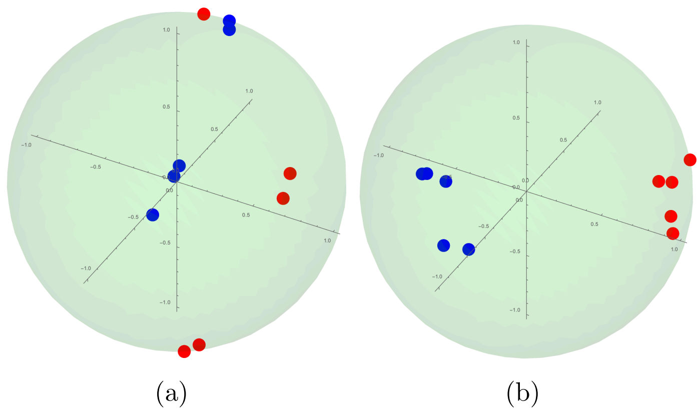
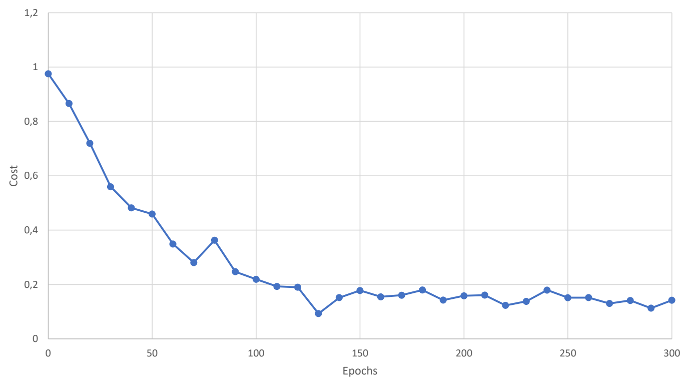


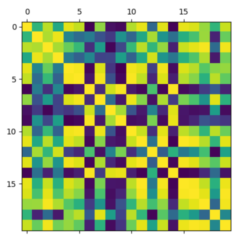
Figure 5.2: Bloch representation of the embedded before training (a) and at the final learning step (b). After learning the states are clustered into families that can be easily classified by a plane, this was not originally possible at lower dimension [119].

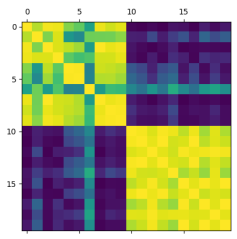
This approach is similar in spirit to the classical Support Vector Machines (SVM) commonly used in Machine Learning to perform classification [14]. SVMs maps complex data (i.e. non linearly separable) via a nonlinear kernel into an high dimensional space where the data can be easily classified by an hyperplane.

A quantum embedding is a representation of classical points x from a data domain X as a (quantum) feature state $|x\rangle$. Either the full embedding, or part of it, can be facilitated by a *quantum feature map*, a quantum circuit $\Phi(x)$ that depends on the input. If the circuit has additional parameters θ



(a)





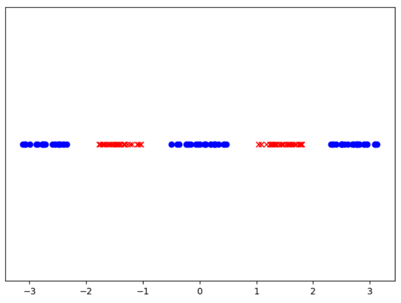
(b) (c)

Figure 5.3: Learning curve of the quantum embedding algorithm (a). Gram matrix of some embedded states at step 0 (b) and at the final training step (b). It shows the presence of two classes that can be now linearly classified [119].

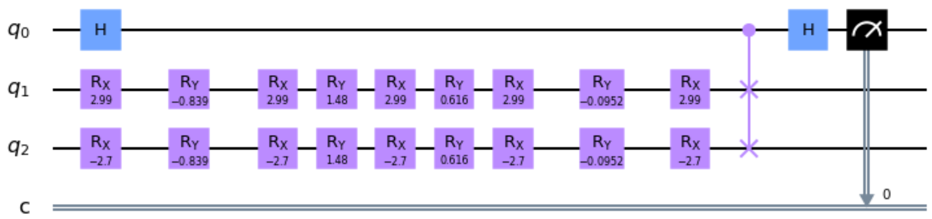
that are adaptable, $\Phi(x) = \Phi(x, \theta)$, the quantum feature map can be trained via optimization. So if we have some embedded data points $|a\rangle$ from class A and $|b\rangle$ from class B what we want to do is to separate them as much as possible in the Hilbert space. That means to compute the overlaps $|\langle a|b\rangle|^2$, $|\langle a|a\rangle|^2$, $|\langle b|b\rangle|^2$ and optimize the parameters of the embedding in order to minimize the cost function:

$$C = 1 - \frac{1}{2}\left(\sum_{i,i'} |\langle a_i|a_{i'}\rangle|^2 + \sum_{j,j'} |\langle b_j|b_{j'}\rangle|^2\right) + \sum_{i,j} |\langle a_i|b_j\rangle|^2. \tag{5.1}$$

That means to maximize the Hilbert-Schmidt norm between the two classes. The optimal results can be shown in terms of the Gram matrix containing all the scalar products between the embedded states and then showing the presence of two cluster of states – see Gram matrices in Fig. 5.3 and the corresponding Bloch sphere representations in Fig. 5.2. The classical data set is reported in Fig. 5.4, while the representation of the embedded states on the Bloch sphere, before and after training, is shown in Fig. 5.2.



(a)



(b)

Figure 5.4: In panel (a) the 1D syntetic dataset used as classification benchmark. Class A are the blue dots while class B are the red crosses. Note that in the 1D space the dataset can't be classified linearly, i.e. there's not a simple threshold we could use to separate the two classes. In panel (b) an example of the embedding circuit including the final SWAP test to compute the overlap between the two embeddings [119].

The training is achieved by using the open-source software Pennylane [120] and the quantum circuit in Ref. [118] that is based on a sequence of rotations on non-commuting axes. As the system is trained we can see from the Gram matrix that it learns to separate the points in the Hilbert space. The training is done taking two datapoints, embedding them into two separate qubits and using a third qubit to perform a SWAP test between the two embeddings, hence giving us the overlaps we need to compute the cost function in Eq. (5.1). The parameters of the embedding circuit are then updated by gradient descent using Pennylane. After some hundreds of training steps we have been able to replicate the results in Ref. [118] – see Fig. 5.3 – and then to adapt this approach to a syntetic dataset to provide a flexible scheme to be tested via different experimental platforms we have at disposal for this task.

More specifically, we implement some experiments of data embedding on a single qubit but on different complementary platforms, with the aim to prove that this quantum embedding is really feasible and also robust against experimental errors. To start with, the training is performed offline, simulating the series of rotations allowed by each experiment itself. Once we have the optimal trained parameters we exploit each experiment to validate the training by performing the embedding of some test data. Notice that the scalar products in the Gram matrix are replaced by the Ulhmann fidelity in the case of the optical data since the quantum tomography-reconstructed density matrices correspond to mixed (non-pure) states because of the unavoidable presence of noise and experimental imperfections.

As an embedding circuit we consider a series of rotations (the number of rotations can be adjusted) along two non-commuting axes. The first rotation will be input-dependent and thus must be recomputed for each datapoint with the range of angles in $[-\pi, \pi]$. The rotations along the second axis are fixed by the training process but in general the angles will be different as they depend of different parameters. Of course, the considered ranges for the trainable parameters can be adjusted to fit the constraints for each experiment. The resulting sequence is:

$$\{R_X(\phi), R_Y(\psi_1), R_X(\phi), R_Y(\psi_2), R_X(\phi), R_Y(\psi_3), R_X(\phi)\} \longrightarrow |\phi\rangle \tag{5.2}$$

where the $R_X(\cdot)$ and $R_Y(\cdot)$ are rotations along the respective axis and ψ_i are the parameters to be trained. We assumed the initial state to be $|0\rangle$ but we could have opted for any other single qubit state. A graphical representation of this embedding circuit can be found in Fig.5.4.

5.1.2 Experiments

Here we deployed the quantum embedding scheme (discussed above) with our trained parameters on different experimental platforms and we compare the results with our theoretical predictions in order to show whether such

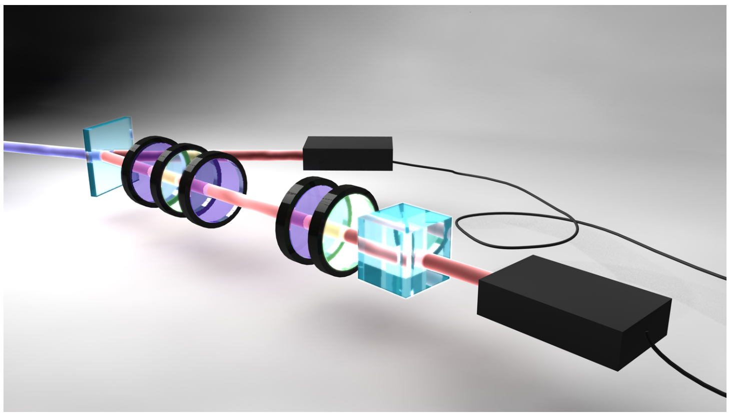


Figure 5.5: Optical setup: we first generate photon-pairs via SPDC, then we prepare one photon by applying a rotation dictated by the embedding parameters and perform its tomography heralded by the other photon [119].

schemes are feasible for practical applications where the presence of environmental noise is almost always unavoidable.

Optics platform

The constraint of the optical setup of M. Barbieri's group, allows for a slightly different approach employing only one rotation along an arbitrary axis. In other terms, for each datapoint we will compute the total resultant rotation from the sequence in (5.2) and then perform such rotation (which now will be along an arbitrary axis). This approach requires a bigger controllability of the system since we cannot choose the rotation axis in advance but it is simplified using only a one-shot embedding instead of a step-by-step approach. Mathematically, our embedding becomes:

$$R_{\hat{n}}(\phi) = e^{-i\phi\hat{n}\cdot\hat{\sigma}/2} = \cos(\phi/2)\mathcal{I} - i\sin(\phi/2)(n_x\sigma_x + n_y\sigma_y + n_z\sigma_z), \tag{5.3}$$

where $\hat{n} = (n_x, n_y, n_z)$ is a real unit vector. For example the embedding of one random point (1.68) becomes

$$R_{(-0.43,0.87,-0.20)}(3.76) \longrightarrow |1.68\rangle \,. \tag{5.4}$$

In our experimental realisation, the embedding is performed into the polarization state of a single photon. The rotation R is achieved by adopting, as customary, a Quarter (θ_{Q1}) - Half (θ_{H2}) - Quarter(θ_{Q3}) wave plate system. As you can see from Fig.5.6 the classification achieved between the test

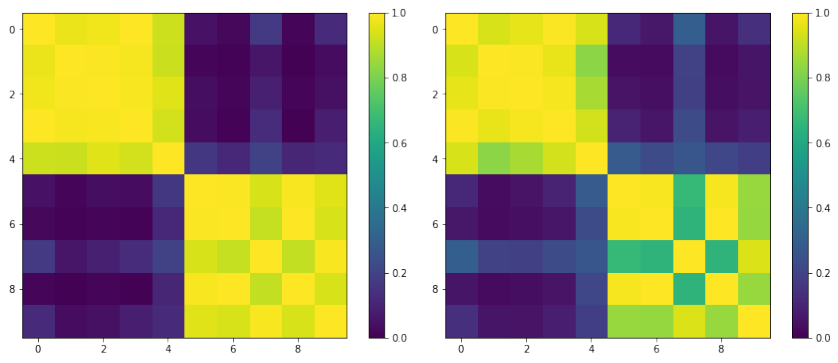


Figure 5.6: Gram matrix obtained by theory, taking 10 validation points the model has never seen at training time (a). Results of embedding on the optics setup with the same validation points. We can see that albeit the experiment introduces some noise results are still close to simulations [119].

points in the Gram matrix is good despite the experimental noise and the constraints given by our particular platform.

Of course this scenario is sub-optimal because we couldn't compute the overlaps with a SWAP test but we had to perform a full state tomograpy. That being said the fidelity achieved, as one can see in Fig.5.7, is good and despide the embedding algorithm was trained in a way completely agnostic to the platform and its specific noise was able to classify correctly our test set.

Rigetti platform

As another experimental demonstration, we test the quantum embedding circuit in one of the NISQ devices that are available on-premise nowadays, again by using the exact same rotation sequence learned for Eq.5.2 already used for the optics experiment setup. The experiment is performed on the Aspen-8 Rigetti QPU, sampling each circuit 2000 times for each of the 100 datapoints necessary to build the Gram matrix, using Rigetti's cloud service. On this particular platform we ran two set of experiments. First we tried the embedding algorithm as intended using the SWAP test to compute the overlap. Indeed the Aspen-8 QPU has 30 qubits that is more than enough to embed a 3-qubit gate. Unfortunately results, as can be seen in Fig.5.8 are

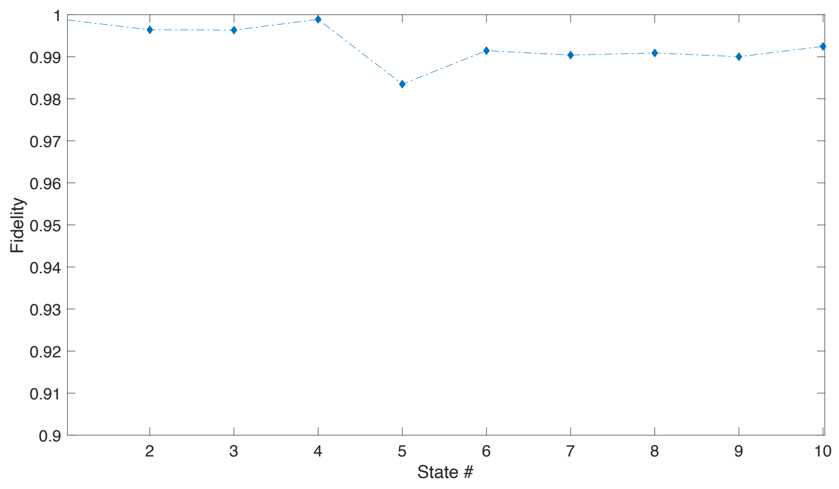
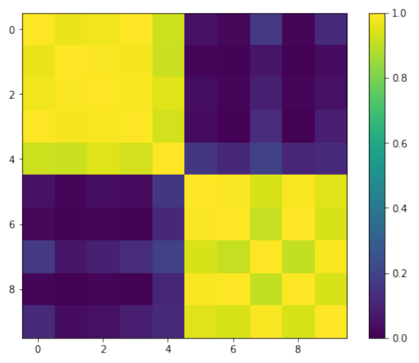


Figure 5.7: Fidelity between the predicted state and the one experimentally reconstructed by quantum state tomography after applying the quantum embedding circuit, calculated for a set of 10 states [119].

pretty disappointing with the experimental result being pretty indistinguishable from noise. This problem emerges due to the connectivity lattice of the QPU that is quasi-one dimensional. That means that in order to embed the CSWAP operation required by our SWAP test on the Aspen-8 lattice, we need a number of operations that far exceeds the coherence time of the device. We thus had to resort to a trick similar to the optics experiment. We performed the embedding on 3 qubits and for each one we measured one component of the $(\sigma_x, \sigma_y, \sigma_z)$ and then computed the overlap with the other embeddings. In Fig.5.9 we show the corresponding Gram matrix as compared with our theoretical predictions.

This results exhibit better overlaps in the intra-class elements but are a little too high on the inter-class ones that we would like to be zero. It is possible that here the choice of the axis has played a role in this behavior. Nevertheless, the classification boundary is present and the system can act as a classifier. The advantage of this experiment is that it could be performed without the need of an ad-hoc lab but it was carried out remotely by just reserving some time on the Rigetti system and programming it. The entirety of the experiments done on this platform took a total of ~ 5 minutes to run.


(a)

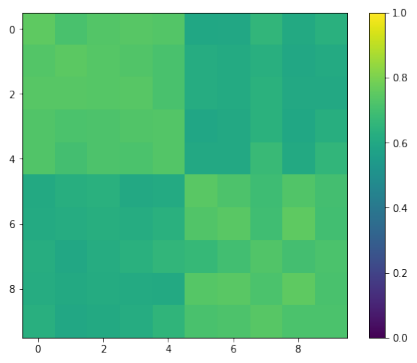
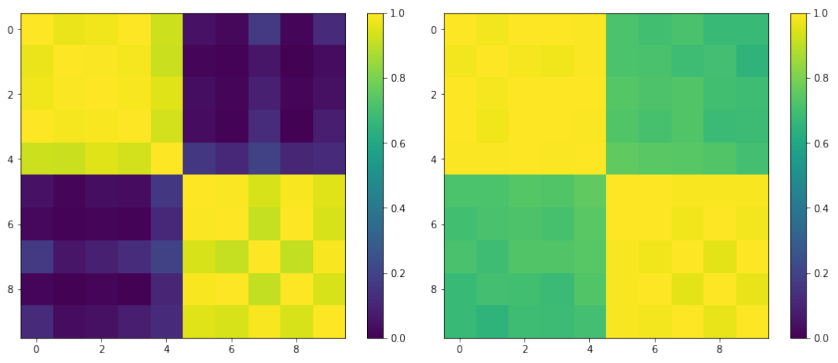

(b)

Figure 5.8: Gram matrix obtained by theory, taking 10 validation points the model has never seen at training time (a). Results of embedding on the Rigetti QPU with the SWAP test, the data is almost indistinguishable from noise even if one could argue that the classification boundary between the two classes is still somehow present [119].

IBM platform

The same process has been carried out in the IBM quantum platform using the Valencia QPU made of 5 qubits. Albeit taking a bit longer to run than Rigetti due to queues on the systems this platform was the only one in which we could use the SWAP test to compute the overlaps. Now the number of samples was reduced to 100 samples per point due to the aforementioned queue problem and the fact that we need to compute 100 overlaps to build the Gram matrix for our 10 test points. As one can see in Fig.5.10 the results, even if noisier than the ones of the previous setups are still good and able to achieve a good classification boundary between the two classes. That is quite remarkable as in this setup the overlap between the embeddings was computed by the quantum processor without needing any tomography or other tricks. Let's also remind that the error on the overlap depends on the number of samples we take and it could very much be that our 100 samples are a sample too small for our case.

These results are useful to prove the fact that Machine Learning algorithms are quite robust to noise, indeed we succesfully implemented the embedding algorithm across different experimental devices each one having different constraints and noise profiles. That fact means that ML could be a



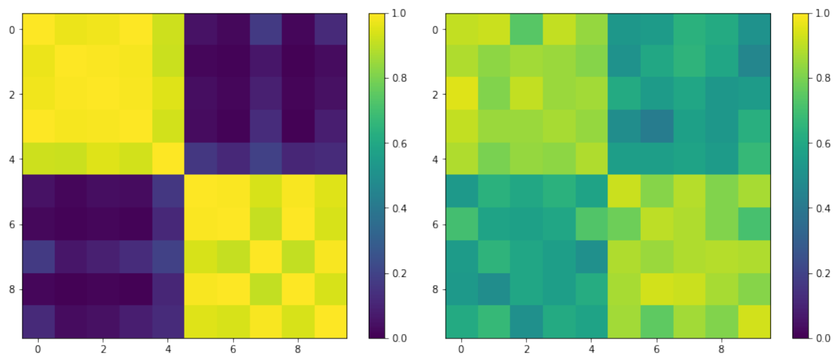
(a) (b)

Figure 5.9: Gram matrix obtained by theory, taking 10 validation points the model has never seen at training time (a). Results of embedding on the Rigetti QPU and measuring the state on different axis, now the problem about the CSWAP overhead has been solved and results are looking good [119].

good candidate to find some form of quantum advantage using NISQ devices since it doesn't require error correction and sometimes it doesn't even require a large number of qubits. Indeed as argued by the authors of the original embedding paper [118] we can map high dimensional data in an *n-qubit* state using the same protocol:

$$|x\rangle = \Psi(x, \theta) |0...00\rangle . \tag{5.5}$$

And if we could build a circuit of 100 qubits with circuit depth 100 and a decoherence time of $10^{-3}s$, this could be capable to embed $\mathcal{O}(10^{10})$ bits of classical information, a task which is classically unattainable. In conclusion, high-dimensional embeddings of large data sets for quantum machine learning could be accessible in the future on NISQ devices and we proved these embeddings to be robust to the real-world noise of them.



(a) (b)

Figure 5.10: Gram matrix obtained by theory, taking 10 validation points the model has never seen at training time (a). Embedding results on the IBM Valencia QPU, in this case the SWAP test still gives results that are pretty consistent with the theory albeit a bit noisier than the other setups.

5.2 Learning on a Quantum Maze

Reinforcement learning has been already extensively studied in closed quantum systems [121]. Yet, the setting of an agent acting on an environment has a natural analogue in open quantum systems [122]. In this section we are exploring whether this two paradigms can work together, in particular we will see how a Stochastic Qantum Walker can benefit from the additional "noise" introduced into the system by an external agent.

5.2.1 Reinforcement learning

In Reinforcement Learning (RL) [123], the system we are interested in is an agent in an environment that has some information about the environment itself and the ability to perform some actions in order to gain some advantage in the form of a reward. In this setting a Markov Decision Process (MDP) is a 5-tuple $(S, A, P.(\cdot,\cdot), R.(\cdot,\cdot), \gamma)$,where

- S is a finite set of states of the agent,
- A is a finite set of actions (alternatively, A_s is the finite set of actions available from state s),

- $P_a(s,s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the probability that action a in state s at time t will lead to state s' at time $t+1$,
- $R_a(s,s')$ is the immediate reward (or expected immediate reward) received after transitioning from state s to state s', due to action a
- $\gamma \in [0,1]$ is the discount factor, which represents the difference in importance between future rewards and present rewards.

In this MDP setting we can define different kinds of problems, based on the information we have at disposal:

- Stochastic multi-armed bandits. There is no state, reward is deterministic. The only stochastic part is the outcome of an action.
- Contextual multi-armed bandits. There is a state and it follows a probability distribution. The actions do not have an impact on the state.
- MDPs. The agent has information on the state and the actions have an effect on the state itself.
- Partially observable MDPs (POMDPs): The state s is partially observable or unknown.

Our goal, or we should say the goal of the agent, is to learn a policy (π). That is a rule according to which the agent selects its actions at each possible state defined as a mapping from past observations to a distribution over the set of possible actions. A policy is called Markovian if the distribution depends only on the last state of the observation sequence. A policy is called stationary if it does not change over time [124].

Te agent aims at learning the policy that maximizes the expected cumulative reward that is represented by the value function. Given a state s, the value function is defined as $V^\pi(s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(Z_t) | Z_0 = (s, \pi(.|s))]$, where Z_t is a random variable over state-action pairs. The policy π giving the optimal value function $V^*(s) = \sup_\pi V^\pi(s)$ would be our intended goal. We also know [123, 124] that the optimal value function must satisfy the Bellman equation.

$$V^\pi(s) = R^\pi(s) + \gamma \int_S P^\pi(s'|s) V^\pi(s') ds' \tag{5.6}$$

In Deep Reinforcement Learning, the policy is learned by a Deep Neural Nework. The objective function of the NN is the Bellman equation itself. The network starts by randomly exploring the space of possible actions and iteratively reinforcing its policy trough the Bellman equation given the reward obtained with each action. A pictorial view of this iterative process can be found in Fig.5.11.

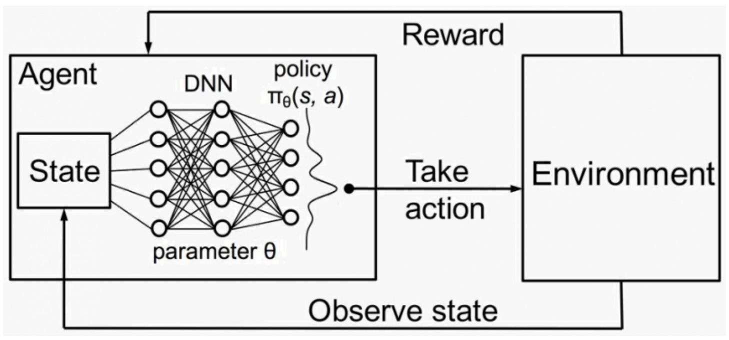


Figure 5.11: Schematic depiction of a Deep Reinforcement Learning Scheme. A DNN learns the policy π that the agent uses to perform an action on the environment. A reward and the information about the new state of the system are given back to the agent that learns its policy accordingly.

5.2.2 Quantum Maze

We plan to apply RL in the scenario of a Quantum Maze. A Quantum Maze is a network where each node corresponds to a pure quantum state in the basis of a Hilbert space. The paths of the maze are defined by links between pair of nodes, which are described by an adjacency matrix A. The coefficient $A_{i,j} = 1$ indicates the presence of the link, while $A_{i,j} = 0$ indicates its absence.

The entrance to the maze is a node which contains the initial population of the density matrix. The density matrix evolves in time with its population spreading across the nodes, i.e., walking along the paths of the maze. The evolution is given by a Lindblad master equation [125] where the Hamiltonian of the system corresponds to the adjacency matrix itself, $H = A$, and with a set of Lindblad operators acting as noise for the quantum walker, i.e.

$$\dot{\rho} = -(1-p)\ i[H,\rho] + p\ \mathcal{L}_{CRW}(\rho) + \mathcal{L}_{sink}(\rho) \tag{5.7}$$

with

$$\mathcal{L}_{CRW}(\rho) = \sum_{i,j} L_{ij}\rho L_{ij}^\dagger - \frac{1}{2}\{L_{ij}^\dagger L_{ij}, \rho\}. \tag{5.8}$$

The Lindblad operators are defined as $L_{ij} = (A_{ij}/d_j) \left|i\right\rangle \left\langle j\right|$, where $\{d_j\}$ are the vertex degrees. These Lindblad operators give a noisy evolution (as in a Classical Random Walk) plus

$$\mathcal{L}_{sink}(\rho) = \Gamma \left[2 \left|S\right\rangle \left\langle n\right| \rho \left|n\right\rangle \left\langle S\right| - \{\left|n\right\rangle \left\langle n\right|, \rho\}\right]. \tag{5.9}$$

is the Lindblad operator that irreversibly transfer the population from the "exit" node n to the sink S, effectively removing it from the maze.

The parameter p defines the trade-off between Hamiltonian evolution ($p = 0$) and the completely noisy (classical) evolution ($p = 1$) with only Lindblad operators [126]. The exit of the maze corresponds to the sink S which traps the population inside of it. Equation 5.7 gives the following solution for the population of the sink,

$$p_{sink}(t) = 2\Gamma \int_0^t \rho_{n,n}(t')\ \mathrm{d}t' \tag{5.10}$$

Ideally, we want all the population to be transferred to the sink, possibly in the shortest amount of time.

We consider a *perfect* quantum maze, i.e., a maze with a single entrance and a single exit. There is a single path to exit the maze, even thought there may be multiple dead ends. In this scenario, an external control (us or some other mechanism) is the agent that has some information about the state of the system, the maze is the environment. The available actions are:

1 – Building walls. During the evolution, at certain periodic instants, the system can change the adjacency matrix of the environment removing a link (changing from 1 to 0 an entry $A_{i,j}$). This emulates the closing of a door that lead to a dead end.

2 – Punching holes into the walls. During the evolution, at certain periodic instants, the system can change the adjacency matrix of the environment adding a link (changing from 0 to 1 an entry $A_{i,j}$). This emulates the creation of a hole in a wall that creates a shortcut.).

In this setup, the objective is the amount of time required to exit the maze (to be minimized) or the amount of population that exited in the sink in a given amount of time (to be maximized).

If the adjacency matrix can change – either intrinsically by random flips or as a result of actions taken by the agent – we are in an MDP setting, which is the canonical scenario for RL. We thus want to learn a policy by which, dynamically changing the topology of the maze, even by just adding or removing a small number of walls we can significantly improve the transfer rate of our walker to the sink. All in all, we could introduce a few more parameters to be optimized, for instance, the noise parameter p that governs the "quantumness" of the dynamics. However for this first proof of principle we wanted to focus only on the topological changes to the maze, in this way we can imagine the action of modifying the adjacency matrix as a sort of additional, engineered "noise" that, if carefully tuned, can improve the performances of our stochastic quantum walker.

5.2.3 Implementation and Results

The setting is the one illustrated in Section 5.2.2, with a maze describing the Hamiltonian for the evolution of the quantum system. Figure 5.12 shows an example of mazes that we have used. The goal is to transfer the initial quantum state (localized at the entrance of the maze, node 0) to the exit of the maze, i.e. to the sink, possibly by modifying some links of the adjacency matrix.

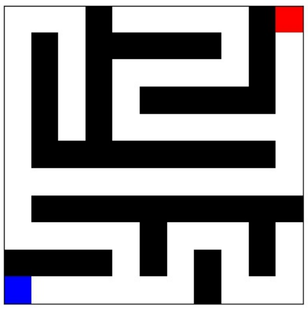


Figure 5.12: Example of a 6×6 maze. In white the possible paths, in black the walls. In the lower left corner, the node in blue is the entrance to the maze, corresponding to the initial quantum state while the node in red is the exit, i.e., the node connected to the sink.

We set a time limit T for the overall evolution of the system and define the time instants $t_k = k\tau,\ \tau = T/N, k = 0, \ldots N-1$ where one of the links can be changed. The quantum system evolves according to its Lindblad equation in the time interval between t_k and t_{k+1}. From the point of view of

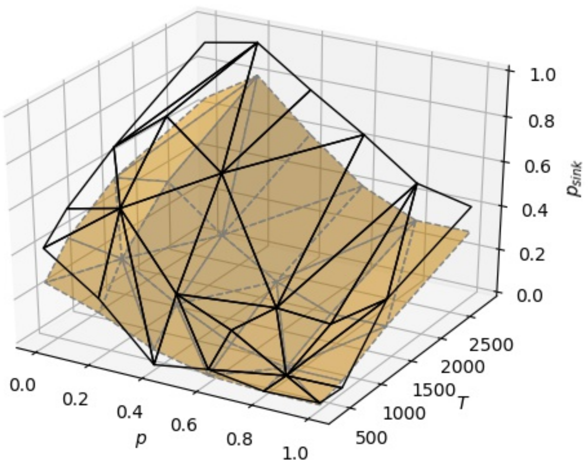


Figure 5.13: Results of learning on a random 6×6 maze. The black surface represent the performance of the quantum walker when the policy is added, while the solid beige surface below is the baseline on the same maze with no action by the agent. The performance is plotted against both p and τ.

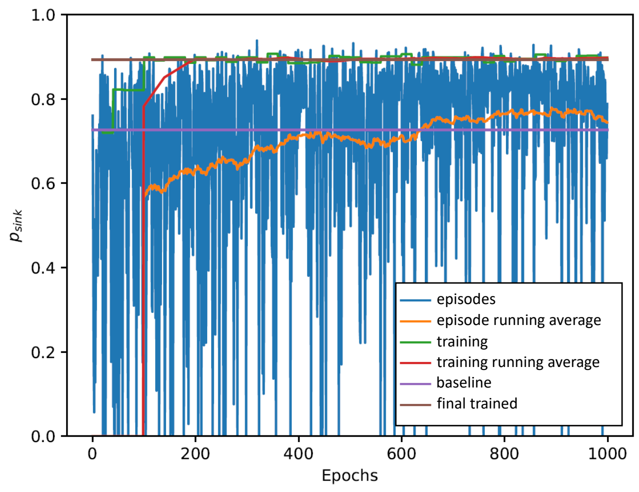
RL in this scenario we are the agent that observes the state configuration and performs some actions, i.e., change the adjacency matrix. The environment is the maze and the state is the quantum system that evolves with it. The possible actions are indexed with the link to modify, so that the action space is discrete and finite. In the current implementation we do not define any penalty for changing a link, even though we could put a negative reward that would force the learning to minimize the number of actions. Instead, we define the reward as the amount of the population transfer that goes into the sink in the time interval following the action. The state of the system that the agent observes in order to inform the policy are the entries of the density matrix (the off-diagonal ones splitted into real and imaginary part). We have implemented Deep Reinforcement Learning with ϵ-greedy algorithm for the policy improvement, and run it with the following set of parameters:

maze size	6x6
p	$\{0, 0.2, 0.4, 0.6, 0.8, 1.0\}$
time samples	$\{350, 700, 1400, 2800\}$
time steps t_k	$k = 1, \ldots 8$
training epochs	1000

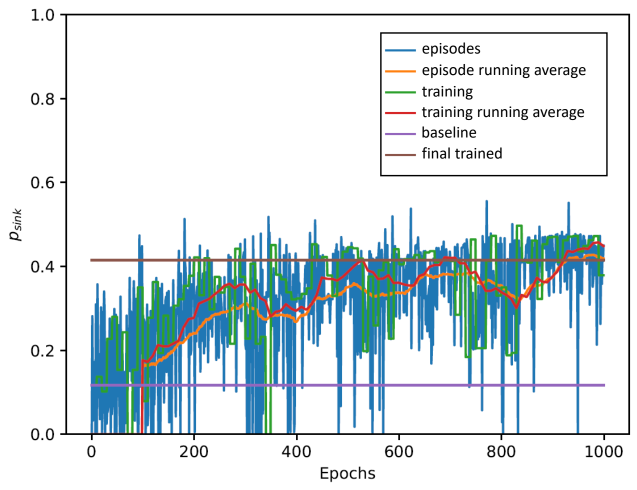
At each step the agent can choose to modify whatever link in the maze, albeit we would expect its actions to be localized around the places where it has the chance to move the most population out of the maze. The ϵ-greedy algorithm we used implies that the agent picks the action suggested by the policy with probability ϵ or a random action with probability $1 - \epsilon$, that increases the chances of the policy to explore different strategies searching for the best one instead of just reinforcing a sub-optimal solution. The value of ϵ is slowly increased during training so that, at the end, the agent is just applying the policy without much further exploration. With this environment set up we would like to observe how the "quantumness" of the system p and the time τ (which since the number N of actions is fixed directly influences the total time T) affect the learning and the final amount of population that ultimately can exit the maze. We can see the results of the training in Fig.5.13 where the exit probability of the quantum walker p_{sink} is plotted against the parameters p and τ. We can see how the added "noise" from the agent is able to consistently increase the population transfer to the sink. The limit case being at low τ where, apart from the case of a perfectly quantum walker ($p = 0$), the evolution time of the system is too low to have a significant signal and the two surfaces are on top of each other.

Also we plotted some training curves to check how our agent explores the space of the possible actions. This curves in Fig.5.14 show that at the beginning of the training the agent starts exploring rather at random and with average performances similar to the baseline. Then as it starts to see some positive reinforcement it learns to consistently perform better actions outperforming the baseline. It is also interesting to see how for the quantum case, where the interference between paths is strong, is easy to do and action that completely disrupts the quantum dynamics leading to a very poor population transfer. Nevertheless the agent is able to learn the from the successful runs and consistently avoid this scenario.

This setting thus seems, after our preliminary analysis, a promising way to improve the transfer efficiency of a stochastic quantum walker in a maze by adding some cleverly engineered topological noise during its dynamics.



(a)



(b)

Figure 5.14: Training curves for an agent doing actions at times $\tau = 2800$ and for $p = 0.2$ (a) and $p = 0.8$ (b). The curves show the rewards from the single episodes and their running average over 100 episodes as well as the training curve of the agent with the running average again over 100 episodes. The two constant lines are the baseline quantum walker with no actions performed by the agent and the final trained policy.

5.3 Conclusions

In this final chapter we have explored two cutting-edge applications of ML in science, namely in the realm of quantum physics, that are generally called Quantum Machine Learning models. We have seen the implementation of a simple classification model on different quantum platforms showing a good robustness to noise. That has a lot of implications on how this model can be trained in a way that is agnostic to the particular platform on which the algorithm is implemented and the possibility to have some form of quantum advantage using NISQ devices. Even without the use of any error mitigation scheme, which might help improving the results especially on the noisier superconducting chips, we were able to successfully deploy the model on three different quantum devices. In the other section we have applied a Reinforcement Learning model to the stochastic quantum walk in a maze. We have shown that just by training the agent has been able to use the partial information that was accessible, to dynamically change the environment and substantially improve the transfer rate of the walker to the sink. This new framework also has some potential to be applied in future devices for example to improve energy or information transfer in the quantum domain.
These two applications are an example of the more theoretical applications of Machine Learning in science and, albeit useless from a practical point of view at this stage, could have interesting implications once the hardware and the technology will allow to scale them up.

Chapter 6

Conclusion

In this thesis we have presented a series of research applications with the aim of demonstrating the various ways in which Machine Learning can be successfully employed in fundamental science and research. Here we summarize the key results and point to some interesting directions for future research.
After we have gone through the ways in which Machine Learning is used in scientific research. It is clear that this interest that surged in recent years is here to stay. We have seen examples of research ranging from the use of real experimental data to exploratory efforts on toy models, trying to understand both the power and the limitations of these approaches. We will recap here the main results presented in this thesis and then we will outline some interesting developments that could be pursued as future research in this domain.
We started by showcasing, in chapter 2, one of the most widespread application of Machine Learning in fundamental sciences, that is applying ML algorithms on experimental pipelines. Working on a biology problem we mapped single cell RNA sequencing data into space using the information of the public Allen Mouse Brain Atlas. By introducing a deep learning-based registration and mapping pipeline, we were able to reveal spatial gene expression patterning beyond the limitations of current technologies. While our work focused on a specific region in the mouse brain it is applicable to any brain region, towards its complete atlas, and to any other organ, as well as disease tissue. To integrate across scales, registration pipeline requires a CCF and is therefore currently applicable to a few organs. At present, the mouse brain possesses the most advanced and well-developed CCF, but efforts are underway to construct analogous reference maps for different or-

gans. These results obtained open up the way to the possibility of having single-cell resolution maps of genetic expression with an high throughput of genes. The fact we were able to achieve this using a Deep Learning based pipeline is the first example of how existing ML technology can be successfully employed in scientific environments.

In chapter 3, we have proposed a novel approach to the training of deep neural networks which is bound to the spectral domain. The eigenvectors and eigenvalues of the adjacency matrices that connects consecutive layers via directed feed-forward links are trained, instead of adjusting the weights that bridge each pair of nodes of the collection, as it is customarily done in the framework of conventional ML approaches. This choice results in a considerable reduction of the computational costs, while still returning good results in terms of classification ability. This example shows that ML itself can benefit from the tools developed by fundamental sciences, in this case network theory. This area of interplay, albeit promising, is still mainly a research topic with significantly fewer applications than the one presented in the above paragraph.

Then, in chapter 4, we have demonstrated the use of quantum annealers (specifically D-Wave 2000Q) as Boltzmann samplers to estimate the negative phase of classical RBMs placed in the latent space of deep convolutional variational autoencoders. This setup allows for the construction of quantum-classical hybrid generative models that can be scaled to large, realistic datasets. The encoding and decoding process is indeed efficiently performed by deep convolutional networks, which are trained to extract relevant features via stochastic gradient descent. As we have shown, this approach is particularly flexible, since it naturally adapts to different connectivities and arbitrary working graphs. By successfully training the same model on three quantum annealers with different noise profiles, we have shown that our implementation is fairly robust to noise and control errors. This result is at variance with training quantum samplers directly on the data distribution, for which weights are typically much larger and regularization is critical to avoid overfitting. Other than being one of the first hybrid quantum-classical applications to get state-of-the-art results, our hybrid variational autoencoder is also a good example on how new physical hardware can be exploited to obtain some gains in ML tasks.

Finally in chapter 5 we have explored two cutting-edge applications of ML in quantum physics, that are generally called Quantum Machine Learning

models. We have seen the implementation of a simple classification model on different quantum platforms showing a good robustness to noise. That has a lot of implications on how this model can be trained in a way that is agnostic to the particular platform on which the algorithm is implemented and the possibility to have some form of quantum advantage using NISQ devices. In the other section we have applied a Reinforcement Learning model to the stochastic quantum walk in a maze. We have shown that just by training the agent has been able to use the partial information that was accessible, to dynamically change the environment and substantially improve the transfer rate of the walker to the sink. This new framework also has some potential to be applied in future devices for example to improve energy or information transfer in the quantum domain.
These two applications are an example of the more theoretical applications of Machine Learning in science and, albeit useless from a practical point of view at this stage, could have interesting implications once the hardware and the technology will allow to scale them up.
All these results are just a set of examples, but they show how deep are the connections and the possibilities of interplay between Machine Learning and fundamental science. In the following years these techniques will have a paramount importance in the frontiers of science. In the next section we will thus list some of the research directions that are a natural follow-up of the work described in this manuscript.

6.1 Directions for future work

Some interesting research ideas that could directly stem from the topics covered in this thesis are worth mentioning to have an understanding of how the research could progress further at the edge between Machine Learning and Science. One rather simple, but important work, would be to extend the experimental results obtained on the mouse brain to other organs, both in mice and humans. That would open up the way tho a whole spatial genome map of the human body that is the ultimate goal of the Human Gene Atlas. Another interesting thing would be to devise some techniques to explore the "latent space" of the mouse brain to understand the important features and do some exploration.

From the Network Theory perspective there's a lot that could be done but perhaps the two most promising directions are to apply the spectral learning method on CNNs to build bigger and better models than the multilayer perceptron investigated in this work, and to exploit the eigenvectors and eigenvalues to have some form of interpretability in the model.

The quantum-classical hybrid models we have considered in this manuscript employ a large amount of classical computing power performed on modern GPUs. The computational task that we offloaded to the quantum annealer (sampling from the latent-space RBMs) can still be performed classically at a fraction of the overall computational cost. To achieve any form of quantum advantage in this framework, we need to offload generative capacity to the prior, by exploiting large RBMs capable of representing complex probability distributions from which classical sampling becomes too expensive. We have provided evidence that this path to quantum advantage is possible by deploying annealers with denser connectivities and lower noise, engineering classical neural nets that better exploit physical connectivities and by working with more complex datasets. All these improvements seem achievable in the near future, and represent possible interesting lines of research. Also the extension to a *fully quantum* model where the encoding and the decoding networks are implemented on a quantum devices be worth exploring.

As for Quantum Machine Learning, it is definitely necessary to scale up the embedding algorithms (and all the other models that have been proposed recently) to the limits of existing devices doing an scaling analysis of performances and errors. The prominent question to answer in this field is that we need to prove if a quantum speedup is theoretically and experimentally possible with Quantum Machine Learning models on NISQ.

There are also many more applications of Machine Learning to Science that we could not cover in this manuscript, but we referenced them in the main text as in following years they will become for sure an important part of both experimental and theoretical research in many domains of physics, biology, engineering, chemistry and, of course, computer science.

Appendix A

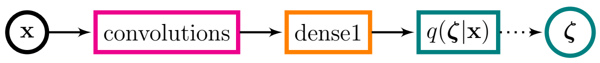
Implementation: Hybrid VAE

This appendix is related to some implementation details of the hybrid quantum-classical Variational Autoencoder previously presented in Chapter 4.

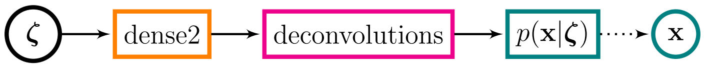
A.1 Convolutional VAE

The VAE employed in our experiments is schematically represented in Fig. A.1. Both approximating posterior $q(\mathbf{z}|\mathbf{x})$ (encoder) and marginal $p(\mathbf{x}|\mathbf{z})$ (decoder) are constructed using deep convolutional networks, see Figs. A.1(a) and A.1(b). Although not technically necessary, we use (approximately) mirror implementations for encoder and decoder. In the encoder, down-sampling is achieved by employing strided convolutions, while in the decoder up-sampling is similarly obtained with strided deconvolutions. The last (first) layer of the encoder (decoder) network is a dense network with two (one) layers (see Fig. A.1(c)). In the case of the encoder, a hierarchical (conditional) relationship among variables is implemented as described in Fig. 4.7(a) in the main text. The convolutional networks are implemented as a simple sequence of five gated convolutions, whose detailed implementation is given in Fig. A.1(c). Notice the use of batch normalization and dropout. The latter was only used in the decoder, to prevent over-fitting, with a drop-rate of 0.2.

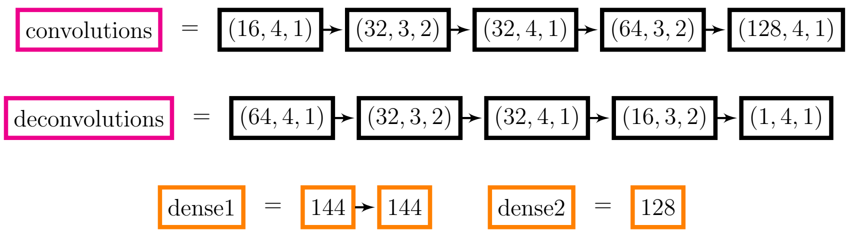
We trained our models using batches of size 100, and the Adam optimizer with an initial learning rate of $3e^{-3}$, exponentially decaying to a minimum learning rate of $1e^{-4}$ after 1800 epochs. The temperature parameter τ defined in Eq. 4.10 for the Gumbel trick is typically annealed from large to small



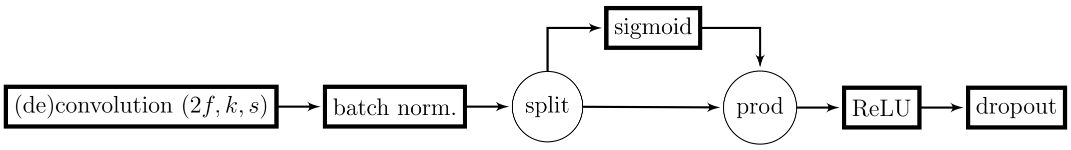
(a) Encoder



(b) Decoder


convolutions = (16, 4, 1) → (32, 3, 2) → (32, 4, 1) → (64, 3, 2) → (128, 4, 1)
deconvolutions = (64, 4, 1) → (32, 3, 2) → (32, 4, 1) → (16, 3, 2) → (1, 4, 1)
dense1 = 144 → 144
dense2 = 128

(c) Convolutional and dense networks



(d) Gated (de)convolution

Figure A.1: Detailed specification of the networks employed in our experiments.

values. We however did not find a real advantage in doing so, and we fixed the parameter to the low value $\tau = 1/7$ throughout the training. To improve training and avoid collapse of the approximating posterior to trivial local minima, we have linearly annealed the KL term from zero to its full value within 200 epochs.

In general we have trained our models using an importance-weighted estimate of the likelihood. As first described in Ref. [111], a K-sample weighting estimate of the log-likelihood can be written as:

$$\mathcal{L}_K = \mathbb{E}_{\zeta_1,\ldots,\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{1}{K}\sum_{k=1}^{K}\frac{p_\theta(\mathbf{z},\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})}\right], \tag{A.1}$$

which is equivalent to the ELBO defined in Eq. 4.3 for $K = 1$ and converges to the exact log-likelihood for $K \to \infty$. We also found useful, to reduce the variance of the gradients of $\mathcal{L}_K$, to use a multi sample evaluation of the gradients per data point $\mathbf{x}$. In other words, we can use the following for training:

$$\mathcal{L}_{K,D} = \mathbb{E}_{\mathbf{z}_{k,d} \sim q_{\phi}(\mathbf{z}|\mathbf{x})} \left[\log \frac{1}{K} \sum_{k=1}^{K} \frac{p_{\mathbf{z}}(\mathbf{z}_{k,d}, \mathbf{x})}{q_{\phi}(\mathbf{z}_{k,d}|\mathbf{x})} \right] , \tag{A.2}$$

with $k = 1, \ldots, K$ and $d = 1, \ldots, D$. Notice that Eq. A.2 requires sampling KD latent configurations per data-point $\mathbf{x}$. This can be parallelized on GPU by effectively working, in our case, with batches of size $KD \times 100$. In our experiments we found it effective to have $KD = 8$ and to change the relative values of K and D while keeping their product constant. Every 200 epochs we changed their value as follows: $(K, D) = (1, 8) \rightarrow (2, 4) \rightarrow (2, 4) \rightarrow (4, 2) \rightarrow (4, 2) \rightarrow (8, 1)$ and kept it constant afterwards. While a larger K results in a tighter variational lower bound, it also makes harder training the approximating posterior, the reason being that in the limit of large K the bound $\mathcal{L}_K$ does not depend on $q_{\phi}(\boldsymbol{\zeta}|\mathbf{x})$. We found this $K \leftrightarrow D$ anneal to be more efficient at both training the approximating posterior and training on a tighter bound to the log-likelihood. We used the same technique, with $K = 1000, D = 1$ as the estimate of the log-likelihood.

A.2 Sample collection with D-Wave 2000Q

To estimate the negative phase with D-Wave annealers, we used 1000 samples obtained with independent annealing runs. For each gradient evaluation, we performed 5 random spin-reversal transformations and collected 200 samples each time. We used a forward annealing schedule with a $1\,\mu s$ forward anneal up to $s = 0.5$, where we paused for $10\ \mu s$. After the pause we performed a $10\,ns$ quench to finish the anneal. After a bit of experimentation, we found this particular annealing schedule to slightly improve training, although a simple forward annealing without pause-and-quench also worked well. We did not perform any post-processing of the samples, which we used as-is to compute the negative phase.

An important question for future works is whether a more careful choice of the annealing schedule, possibly with longer pauses, can stabilize the effective temperature at which samples are drawn from the hardware. We discuss the importance of this aspect in the next section.

A.3 Estimating effective temperature during training

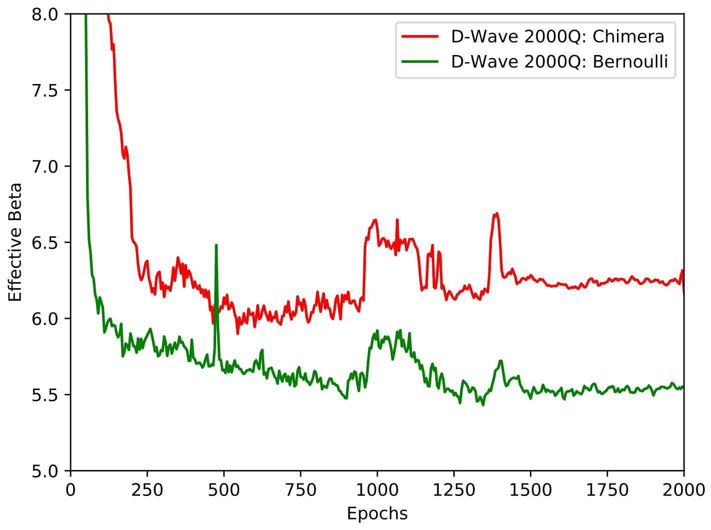


Figure A.2: β^*_{eff} evaluated on two simultaneous runs on a D-Wave 2000Q (Chimera and Bernoulli priors). The fluctuations of its value on the two runs are correlated, indicating our evaluation of β^*_{eff} is effectively probing fluctuations of the physical temperature of the device.

As noticed in Ref. [127], training a BM with a quantum annealer does not necessarily require the knowledge of the effective sampling temperature introduced in Eq. 4.14. Indeed, β^*_{eff} can be absorbed into the learning rate γ:

$$\begin{aligned}\partial \log \tilde{p}_{b,W}(\mathbf{z}) &= \gamma \left(-\partial \mathcal{H}_{b,W}(\mathbf{z}) + \mathbb{E}_{\bar{\mathbf{z}}\sim p_{b,W}}[\partial \mathcal{H}_{b,W}(\bar{\mathbf{z}})]\right) = \\ &= \gamma' \left(-\partial \mathcal{H}_{h,J}(\mathbf{z}) + \mathbb{E}_{\bar{\mathbf{z}}\sim p_{b,W}}[\partial \mathcal{H}_{h,J}(\bar{\mathbf{z}})]\right) \\ \gamma' &= \gamma \beta^*_{eff} \,. \end{aligned} \tag{A.3}$$

While this is still true for the gradients of the parameters of the RBM placed in the latent space of a VAE, correctly evaluating the gradients of the inference parameters $\boldsymbol{\phi}$ requires knowledge of β^*_{eff}. To see this it suffices to note that the samples in the positive phase depends on the inference parameters through the reparameterization trick. During training: $\mathbf{z} \to \boldsymbol{\zeta}(\boldsymbol{\phi}, \boldsymbol{\rho})$.

Tracking where these gradients come from, we have:

$$\gamma\partial_{\boldsymbol{\phi}}(ELBO) \;=\; -\gamma\partial_{\boldsymbol{\phi}}\log q_{\boldsymbol{\phi}}(\boldsymbol{\zeta}(\boldsymbol{\phi},\boldsymbol{\rho})|\mathbf{x}) - \gamma\beta^*_{eff}\partial_{\boldsymbol{\phi}}\mathcal{H}_{h,J}(\boldsymbol{\zeta}(\boldsymbol{\phi},\boldsymbol{\rho})\;, \tag{A.4}$$

so that the correct evaluation of the gradients with respect to the inference parameters requires the (approximate) knowledge of the effective temperature β^*_{eff}.

In our experiments, we have performed a real-time estimation of β^*_{eff}, which we used as in the equation above to correctly estimate the gradients for the inference parameters. To do so, we employed an auxiliary BM that we trained in parallel with the VAE on the negative samples obtained by the quantum annealers. The parameters of the BM are shared according to Eq. 4.14, with the only trainable parameter being β^*_{eff}. In other words, we update β^*_{eff} as follows:

$$\beta^*_{eff} \;\to\; \beta^*_{eff} + \gamma\left(-\mathbb{E}_{\bar{\mathbf{z}}\sim p^{HW}_{h,J}}[\mathcal{H}_{h,J}(\mathbf{z})] + \mathbb{E}_{\bar{\mathbf{z}}\sim p^{BM}_{b,W}}[\mathcal{H}_{h,J}(\bar{\mathbf{z}})]\right)\;, \tag{A.5}$$

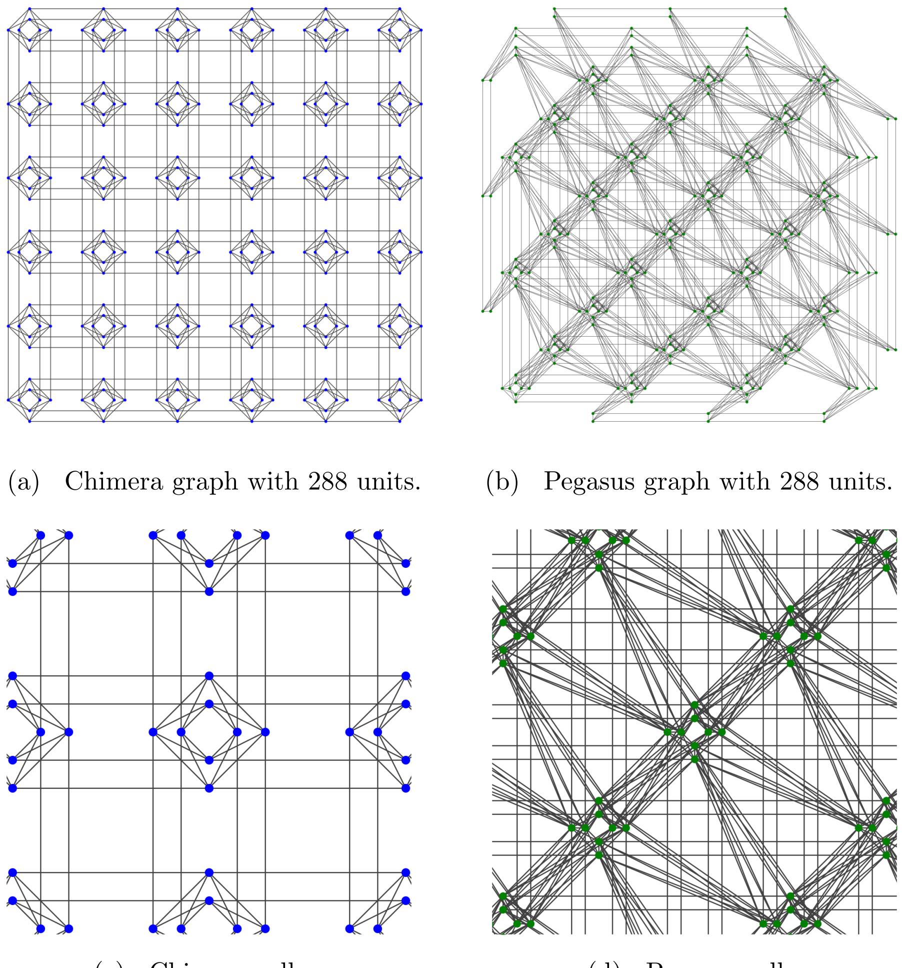
where the first expectation is evaluated with the hardware samples; the second, with thermal samples from the auxiliary BM (obtained with PA).

In Fig. A.2 we show the value of β^*_{eff} estimated with the method above on two simultaneous runs on a D-Wave 2000Q. Its value typically drops while the KL term is annealed (200 epochs in our experiments), and subsequently stabilizes. Some fluctuations are correlated among independent runs, and are related to real fluctuations of the physical temperature of the device.

We have noticed that, due the use of the KL anneal and the presence of a non-negligible change in β^*_{eff} during training, using a time-dependent evaluation of the effective temperature is important to stabilize training. While computing a single gradient as in Eq. A.5 is much more robust than training all the weights of a comparable BM, the method is not completely scalable and requires thermal sampling with classical algorithms. It will be critical, in future works, to implement training procedures with stable values of β^*_{eff}, which could be kept constant, using values predetermined by previous experiments or simply treated as a hyper parameter whose value must be appropriately fixed. Eventually, the use of more advanced annealing schedules, with longer pauses and more carefully chosen pause-points, should allow a direct connection between β^*_{eff} and the physical temperature of the annealer, thus removing the necessity of learning β^*_{eff} from experiments.

A.4 Chimera and Pegasus connectivities

In Fig. A.3 we show the Chimera and Pegasus connectivities on 288 qubits used in all the experiments performed. The Chimera graph (Fig. A.3(a)) is a bipartite, two-dimensional tiling of a unit cell (Fig. A.3(c)) with 8 qubits. The Pegasus graph (Fig. A.3(b)) is a quadri-partite, two-dimensional tiling of a unit cell (Fig. A.3(d)) with 8 qubits [112].



(a) Chimera graph with 288 units. (b) Pegasus graph with 288 units.

(c) Chimera cells. (d) Pegasus cells.

Figure A.3: Physical connectivities used in the experiments.

Appendix B

Publications

This research activity has led to several publications in international journals and conferences. These are summarized below. Here, are also present some works that do not cover topics of Machine Learning. Those come from research in Quantum Thermodynamics that has been carried out in parallel (with some occasional intersection) with the main topics discussed in this thesis. [1]

International Peer-reviewed Journals

1. S. Gherardini, **L. Buffoni**, M.M. Mueller, F. Caruso, M. Campisi, A. Trombettoni, S. Ruffo. "Non-equilibrium quantum-heat statistics under stochastic projective measurements", *Phys. Rev. E* 98 (3), 032108 (2018)

2. **L. Buffoni**, A. Solfanelli, P. Verrucchi, A. Cuccoli, M. Campisi. "Quantum Measurement Cooling", *Phys. Rev. Lett.* 122 (7), 070603 (2019)

3. A. Solfanelli, **L. Buffoni**, A. Cuccoli, M. Campisi. "Maximal energy extraction via quantum measurement", *J. Stat. Mech.: Theory Exp.* (9), 094003 (2019)

4. **L. Buffoni**, M. Campisi. "Thermodynamics of a Quantum Annealer", *Quantum. Sci. Tech.* (2020)

5. V. Cimini, S. Gherardini, M. Barbieri, I. Gianani, M. Sbroscia, **L. Buffoni**, M. Paternostro, F. Caruso. "Experimental characterization of the energetics of quantum logic gates", *npj Quantum Information* in press (2020)

[1]The author's bibliometric indices are the following: H-index = 4, total number of citations = 78 (source: Google Scholar on October, 2020).

6. W. Vinci, **L. Buffoni**, H. Sadeghi, A. Khoshaman, E. Andriyash, M.H. Amin. "A path towards quantum advantage in training deep generative models with quantum annealers", *Mach. Learn.: Sci. Technol.* (2020)

Preprints

1. H. Sadeghi, E. Andriyash, W. Vinci, **L. Buffoni**, M.H. Amin. "PixelVAE++: Improved PixelVAE with Discrete Prior", *arXiv preprint* arXiv:1908.09948 (2019)

2. L. Giambagli, **L. Buffoni**, T. Carletti, W. Nocentini, D. Fanelli. "Machine Learning in spectral domain", *arXiv preprint* arXiv:2005.14436 (2020)

3. T. Biancalani, G. Scalia, **L. Buffoni**, R. Avasthi, Z. Lu, A. Sanger, N. Tokcan, C.R. Vanderburg, A. Segerstolpe, M. Zhang, I. Avraham-Davidi, S. Vickovic, M. Nitzan, S. Ma, J. Buenrostro, N. Bear Brown, D. Fanelli, X. Zhuang, E. Z Macosko, A. Regev. "Deep learning and alignment of spatially-resolved whole transcriptomes of single cells in the mouse brain with Tangram", *biorXiv preprint* 2020.08.29.272831 (2020)

Bibliography

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer, New York, 1st ed. 2006. corr. 2nd printing 2011 edition edition, April 2011.

[2] T. M. Cover and Joy A. Thomas. *Elements of information theory.* Wiley series in telecommunications. Wiley, New York, 1991.

[3] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction.* Springer Science & Business Media, 2009.

[4] Andriy Burkov. *The Hundred-Page Machine Learning Book.* `http://themlbook.com/`, 2019.

[5] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

[6] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[7] Nicu Sebe, Ira Cohen, Ashutosh Garg, and Thomas S Huang. *Machine learning in computer vision*, volume 29. Springer Science & Business Media, 2005.

[8] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386, 2020.

[9] Min Chen, Shiwen Mao, and Yunhao Liu. Big data: A survey. *Mobile networks and applications*, 19(2):171–209, 2014.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. book in preparation for mit press. *http://www. deeplearningbook.org*, 2016.

[11] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[12] Geoffrey E Hinton, Peter Dayan, Brendan J Frey, and Radford M Neal. The" wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158, 1995.

[13] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[14] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[15] Yoav Freund and Llew Mason. The alternating decision tree learning algorithm. In *icml*, volume 99, pages 124–133, 1999.

[16] Keinosuke Fukunaga and Patrenahalli M. Narendra. A branch and bound algorithm for computing k-nearest neighbors. *IEEE transactions on computers*, 100(7):750–753, 1975.

[17] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[18] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[19] Tommaso Biancalani, Gabriele Scalia, Lorenzo Buffoni, Raghav Avasthi, Ziqing Lu, Aman Sanger, Neriman Tokcan, Charles R Vanderburg, Asa Segerstolpe, Meng Zhang, et al. Deep learning and alignment of spatially-resolved whole transcriptomes of single cells in the mouse brain with tangram. *bioRxiv*, 2020.

[20] Evan Z Macosko, Anindita Basu, Rahul Satija, James Nemesh, Karthik Shekhar, Melissa Goldman, Itay Tirosh, Allison R Bialas, Nolan Kamitaki, Emily M Martersteck, et al. Highly parallel genome-wide expression profiling of individual cells using nanoliter droplets. *Cell*, 161(5):1202–1214, 2015.

[21] Laleh Haghverdi, Maren Büttner, F Alexander Wolf, Florian Buettner, and Fabian J Theis. Diffusion pseudotime robustly reconstructs lineage branching. *Nature methods*, 13(10):845, 2016.

[22] Dylan Kotliar, Adrian Veres, M Aurel Nagy, Shervin Tabrizi, Eran Hodis, Douglas A Melton, and Pardis C Sabeti. Identifying gene expression programs of cell-type identity and cellular activity with single-cell rna-seq. *Elife*, 8:e43803, 2019.

[23] Tim Stuart, Andrew Butler, Paul Hoffman, Christoph Hafemeister, Efthymia Papalexi, William M Mauck III, Yuhan Hao, Marlon Stoeckius, Peter Smibert, and Rahul Satija. Comprehensive integration of single-cell data. *Cell*, 177(7):1888–1902, 2019.

[24] Mor Nitzan, Nikos Karaiskos, Nir Friedman, and Nikolaus Rajewsky. Gene expression cartography. *Nature*, 576(7785):132–137, 2019.

[25] Qian Zhu, Sheel Shah, Ruben Dries, Long Cai, and Guo-Cheng Yuan. Identification of spatially associated subpopulations by combining scrnaseq and sequential fluorescence in situ hybridization data. *Nature biotechnology*, 36(12):1183, 2018.

[26] Samuel G Rodriques, Robert R Stickels, Aleksandrina Goeva, Carly A Martin, Evan Murray, Charles R Vanderburg, Joshua Welch, Linlin M Chen, Fei Chen, and Evan Z Macosko. Slide-seq: A scalable technology for measuring genome-wide expression at high spatial resolution. *Science*, 363(6434):1463–1467, 2019.

[27] Patrik L Ståhl, Fredrik Salmén, Sanja Vickovic, Anna Lundmark, José Fernández Navarro, Jens Magnusson, Stefania Giacomello, Michaela Asp, Jakub O Westholm, Mikael Huss, et al. Visualization and analysis of gene expression in tissue sections by spatial transcriptomics. *Science*, 353(6294):78–82, 2016.

[28] Kok Hao Chen, Alistair N Boettiger, Jeffrey R Moffitt, Siyuan Wang, and Xiaowei Zhuang. Spatially resolved, highly multiplexed rna profiling in single cells. *Science*, 348(6233), 2015.

[29] Quanxin Wang, Song-Lin Ding, Yang Li, Josh Royall, David Feng, Phil Lesnar, Nile Graddis, Maitham Naeemi, Benjamin Facer, Anh Ho, et al. The allen mouse brain common coordinate framework: A 3d reference atlas. *Cell*, 2020.

[30] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille, 2015.

[31] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[32] Csaba Erö, Marc-Oliver Gewaltig, Daniel Keller, and Henry Markram. A cell atlas for the mouse brain. *Frontiers in neuroinformatics*, 12:84, 2018.

[33] Velina Kozareva, Caroline Martin, Tomas Osorno, Stephanie Rudolph, Chong Guo, Charles Vanderburg, Naeem M Nadaf, Aviv Regev, Wade Regehr, and Evan Macosko. A transcriptomic atlas of the mouse cerebellum reveals regional specializations and novel cell types. *bioRxiv*, 2020.

[34] Nicholas J Tustison, Philip A Cook, Arno Klein, Gang Song, Sandhitsu R Das, Jeffrey T Duda, Benjamin M Kandel, Niels van Strien, James R Stone, James C Gee, et al. Large-scale evaluation of ants and freesurfer cortical thickness measurements. *Neuroimage*, 99:166–179, 2014.

[35] Guha Balakrishnan, Amy Zhao, Mert R Sabuncu, John Guttag, and Adrian V Dalca. Voxelmorph: a learning framework for deformable medical image registration. *IEEE transactions on medical imaging*, 38(8):1788–1800, 2019.

[36] Yuncong Chen, Lauren E McElvain, Alexander S Tolpygo, Daniel Ferrante, Beth Friedman, Partha P Mitra, Harvey J Karten, Yoav Freund, and David Kleinfeld. An active texture-based digital atlas enables automated mapping of structures and markers across brains. *Nature methods*, 16(4):341–350, 2019.

[37] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[38] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[39] `https://github.com/broadinstitute/one-shot-atlas`.

[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[41] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5(1):1–9, 2014.

[42] Benno S Rem, Niklas Käming, Matthias Tarnowski, Luca Asteria, Nick Fläschner, Christoph Becker, Klaus Sengstock, and Christof Weitenberg. Identifying quantum phase transitions using artificial neural networks on experimental data. *Nature Physics*, 15(9):917–920, 2019.

[43] Pavlo O Dral. Quantum chemistry in the age of machine learning. *The Journal of Physical Chemistry Letters*, 11(6):2336–2347, 2020.

[44] Lorenzo Giambagli, Lorenzo Buffoni, Timoteo Carletti, Walter Nocentini, and Duccio Fanelli. Machine learning in spectral domain. *arXiv preprint arXiv:2005.14436*, 2020.

[45] Jonathan Frankle, David J. Schwab, and Ari S. Morcos. Training batchnorm and only batchnorm: On the expressive power of random features in cnns, 2020.

[46] Marylou Gabrié, Andre Manoel, Clément Luneau, Jean Barbier, Nicolas Macris, Florent Krzakala, and Lenka Zdeborová. Entropy and mutual information in models of deep neural networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124014, dec 2019.

[47] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[48] Adam Byerly, Tatiana Kalganova, and Ian Dear. A branching and merging convolutional network with homogeneous filter capsules. *arXiv preprint arXiv:2001.09136*, 2020.

[49] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[50] `https://github.com/Buffoni/spectral_learning`.

[51] Walter Vinci, Lorenzo Buffoni, Hossein Sadeghi, Amir Khoshaman, Evgeny Andriyash, and Mohammad Amin. A path towards quantum advantage in training deep generative models with quantum annealers. *Machine Learning: Science and Technology*, 2020.

[52] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[53] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[54] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[55] Tijmen Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th international conference on Machine learning*, pages 1064–1071. ACM, 2008.

[56] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. An introduction to quantum machine learning. *Contemporary Physics*, 56(2):172–185, 2015.

[57] Peter Wittek. *Quantum machine learning: what quantum computing means to data mining.* Academic Press, 2014.

[58] Jeremy Adcock, Euan Allen, Matthew Day, Stefan Frick, Janna Hinchliff, Mack Johnson, Sam Morley-Short, Sam Pallister, Alasdair Price, and Stasja Stanisic. Advances in quantum machine learning. *arXiv preprint arXiv:1512.02900*, 2015.

[59] Srinivasan Arunachalam and Ronald de Wolf. A survey of quantum learning theory. *arXiv preprint arXiv:1701.06806*, 2017.

[60] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549:195–202, 2017.

[61] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical review letters*, 103(15):150502, 2009.

[62] Nathan Wiebe, Daniel Braun, and Seth Lloyd. Quantum algorithm for data fitting. *Physical review letters*, 109(5):050505, 2012.

[63] Andrew M Childs, Robin Kothari, and Rolando D Somma. Quantum linear systems algorithm with exponentially improved dependence on precision. *arXiv preprint arXiv:1511.02306*, 2015.

[64] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum principal component analysis. *Nature Physics*, 10(9):631–633, 2014.

[65] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

[66] Daniel A Lidar and Todd A Brun. *Quantum error correction.* Cambridge University Press, 2013.

[67] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3):032324, 2012.

[68] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, Joshua Lapan, Andrew Lundgren, and Daniel Preda. A quantum adiabatic evolution algorithm applied to random instances of an np-complete problem. *Science*, 292(5516):472–475, 2001.

[69] Mark W Johnson, Mohammad HS Amin, Suzanne Gildert, Trevor Lanting, Firas Hamze, Neil Dickson, R Harris, Andrew J Berkley, Jan Johansson, Paul Bunyk, et al. Quantum annealing with manufactured spins. *Nature*, 473(7346):194–198, 2011.

[70] C Neill, P Roushan, K Kechedzhi, S Boixo, SV Isakov, V Smelyanskiy, R Barends, B Burkett, Y Chen, Z Chen, et al. A blueprint for demonstrating quantum supremacy with superconducting qubits. *arXiv preprint arXiv:1709.06678*, 2017.

[71] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M Chow, and Jay M Gambetta. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242, 2017.

[72] Tadashi Kadowaki and Hidetoshi Nishimori. Quantum annealing in the transverse ising model. *Physical Review E*, 58(5):5355, 1998.

[73] Giuseppe E Santoro, Roman Martoňák, Erio Tosatti, and Roberto Car. Theory of quantum annealing of an ising spin glass. *Science*, 295(5564):2427–2430, 2002.

[74] J Brooke, TF Rosenbaum, and G Aeppli. Tunable quantum tunnelling of magnetic domain walls. *Nature*, 413(6856):610–613, 2001.

[75] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014.

[76] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L O'brien. A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, 5, 2014.

[77] JS Otterbach, R Manenti, N Alidoust, A Bestwick, M Block, B Bloom, S Caldwell, N Didier, E Schuyler Fried, S Hong, et al. Unsupervised machine learning on a hybrid quantum computer. *arXiv preprint arXiv:1712.05771*, 2017.

[78] Hartmut Neven, Vasil S Denchev, Geordie Rose, and William G Macready. Training a binary classifier with the quantum adiabatic algorithm. *arXiv preprint arXiv:0811.0416*, 2008.

[79] Vasil S Denchev, Nan Ding, SVN Vishwanathan, and Hartmut Neven. Robust classification with adiabatic quantum optimization. *arXiv preprint arXiv:1205.1148*, 2012.

[80] Kristen L Pudenz and Daniel A Lidar. Quantum adiabatic machine learning. *Quantum information processing*, 12(5):2027–2070, 2013.

[81] Alex Mott, Joshua Job, Jean-Roch Vlimant, Daniel Lidar, and Maria Spiropulu. Solving a higgs optimization problem with quantum annealing for machine learning. *Nature*, 550(7676):375, 2017.

[82] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. *arXiv:1406.2661 [cs, stat]*, June 2014. arXiv: 1406.2661.

[83] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[84] Rob Fergus, Yair Weiss, and Antonio Torralba. Semi-supervised learning in gigantic image collections. In *Advances in neural information processing systems*, pages 522–530, 2009.

[85] Yuzong Liu and Katrin Kirchhoff. Graph-based semi-supervised learning for phone and segment classification. In *INTERSPEECH*, pages 1840–1843, 2013.

[86] Mingguang Shi and Bing Zhang. Semi-supervised learning improves gene expression-based prediction of cancer recurrence. *Bioinformatics*, 27(21):3017–3023, 2011.

[87] Hailin Chen and Zuping Zhang. A semi-supervised method for drug-target interaction prediction with consistency in networks. *PloS one*, 8(5):e62975, 2013.

[88] Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589, 2014.

[89] Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

[90] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[91] Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. *arXiv preprint arXiv:1402.0030*, 2014.

[92] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[93] Lars Maaløe, Marco Fraccaro, and Ole Winther. Semi-supervised generation with cluster-aware generative models. *arXiv preprint arXiv:1704.00637*, 2017.

[94] Alireza Makhzani and Brendan Frey. Pixelgan autoencoders. *arXiv preprint arXiv:1706.00531*, 2017.

[95] John Paisley, David Blei, and Michael Jordan. Variational bayesian inference with stochastic search. *arXiv preprint arXiv:1206.6430*, 2012.

[96] Shixiang Gu, Sergey Levine, Ilya Sutskever, and Andriy Mnih. Muprop: Unbiased backpropagation for stochastic neural networks. *arXiv preprint arXiv:1511.05176*, 2015.

[97] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[98] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[99] Amir H Khoshaman and Mohammad H Amin. Gumbolt: Extending gumbel trick to boltzmann priors. *arXiv preprint arXiv:1805.07349*, 2018.

[100] Jason Tyler Rolfe. Discrete variational autoencoders. *arXiv preprint arXiv:1609.02200*, 2016.

[101] Amir Khoshaman, Walter Vinci, Brandon Denis, Evgeny Andriyash, and Mohammad H Amin. Quantum variational autoencoder. *Quantum Science and Technology*, 4(1):014001, 2019.

[102] Mohammad H Amin. Searching for quantum speedup in quasistatic quantum annealers. *Physical Review A*, 92(5):052323, 2015.

[103] Jeffrey Marshall, Eleanor G Rieffel, and Itay Hen. Thermalization, freeze-out, and noise: Deciphering experimental quantum annealers. *Physical Review Applied*, 8(6):064025, 2017.

[104] Jeffrey Marshall, Davide Venturelli, Itay Hen, and Eleanor G Rieffel. Power of pausing: Advancing understanding of thermalization in experimental quantum annealers. *Physical Review Applied*, 11(4):044083, 2019.

[105] Jack Raymond, Sheir Yarkoni, and Evgeny Andriyash. Global warming: Temperature estimation in annealers. *arXiv preprint arXiv:1606.00919*, 2016.

[106] D-wave system documentation. `https://docs.dwavesys.com/docs/latest/index.html`.

[107] Mohammad H Amin, Evgeny Andriyash, Jason Rolfe, Bohdan Kulchytskyy, and Roger Melko. Quantum boltzmann machine. *Physical Review X*, 8(2):021050, 2018.

[108] R Harris, Y Sato, AJ Berkley, M Reis, F Altomare, MH Amin, K Boothby, P Bunyk, C Deng, C Enderud, et al. Phase transitions in a programmable quantum spin glass simulator. *Science*, 361(6398):162–165, 2018.

[109] Andrew D King, Juan Carrasquilla, Jack Raymond, Isil Ozfidan, Evgeny Andriyash, Andrew Berkley, Mauricio Reis, Trevor Lanting, Richard Harris, Fabio Altomare, et al. Observation of topological phenomena in a programmable lattice of 1,800 qubits. *Nature*, 560(7719):456, 2018.

[110] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015.

[111] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

[112] Kelly Boothby, Paul Bunyk, Jack Raymond, and Aidan Roy. Next-generation topology of d-wave quantum processors. Technical report, Technical report, 2019.

[113] Vicky Choi. Minor-embedding in adiabatic quantum computation: I. the parameter setting problem. *Quantum Information Processing*, 7(5):193–209, 2008.

[114] Vicky Choi. Minor-embedding in adiabatic quantum computation: Ii. minor-universal graph design. *Quantum Information Processing*, 10(3):343–353, 2011.

[115] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

[116] Hossein Sadeghi, Evgeny Andriyash, Walter Vinci, Lorenzo Buffoni, and Mohammad H Amin. Pixelvae++: Improved pixelvae with discrete prior. *arXiv preprint arXiv:1908.09948*, 2019.

[117] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[118] Seth Lloyd, Maria Schuld, Aroosa Ijaz, Josh Izaac, and Nathan Killoran. Quantum embeddings for machine learning. *arXiv preprint arXiv:2001.03622*, 2020.

[119] Lorenzo Buffoni, Ilaria Gianani, Marco Barbieri, and Filippo Caruso. Experimental platforms for quantum embedding. *in preparation*, 2020.

[120] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, Carsten Blank, Keri McKiernan, and Nathan Killoran. Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968*, 2018.

[121] Vedran Dunjko, Jacob M. Taylor, and Hans J. Briegel. Quantum-enhanced machine learning. *Physical Review Letters*, 117(13):130501, September 2016.

[122] Heinz-Peter Breuer and Francesco Petruccione. *The theory of open quantum systems*. Oxford University Press, 2002.

[123] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[124] Mohammed Ghavamzadeh, Shie Mannor, Joelle Pineau, and Aviv Tamar. Bayesian reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 8(5-6):359–483, 2015.

[125] Goran Lindblad. On the generators of quantum dynamical semigroups. *Communications in Mathematical Physics*, 48(2):119–130, 1976.

[126] Filippo Caruso. Universally optimal noisy quantum walks on complex networks. *New Journal of Physics*, 16(5):055015, May 2014.

[127] Marcello Benedetti, John Realpe-Gómez, Rupak Biswas, and Alejandro Perdomo-Ortiz. Estimation of effective temperatures in quantum annealers for sampling applications: A case study with possible applications in deep learning. *Physical Review A*, 94(2):022308, 2016.