Multi-Type Continuous Disentanglement Variational AutoEncoder

Projecting multiple categorical models into a single dimensional space.

Candidate number 1053143

-

University of Oxford

A thesis submitted for the degree of

Master of Computer Science

Trinity 2021

DEDICATION

This work is dedicated to my friends and housemates, who, through their enthusiasm and curiosity, stimulated me to have amazing discussions and ideas.

And to my family and all the people that encouraged me to follow this path and reach my goals.

And a special thank to You, for supporting me during these though last months.

ACKNOWLEDGEMENTS

Left blank for anonymity purposes.

Word count: ∼16000 words *(computed as an average of multiple tools' counts)*.

ABSTRACT

In recent years, artificial intelligence has become increasingly popular and its successful application countless, thanks to neural networks. However, most deep learning algorithms rely on huge quantities of data to produce satisfactory results. In scenarios in which plenty of high quality datasets are not available, modern models struggle to achieve human-like performance. An example of such situation is Emotion Recognition, a task for which there exists numerous datasets that, however, are designed following different categorical models and, as a consequence, are not easily comparable and usable together. In order to overcome this issue, I propose Multi-Type Continuous Disentanglement Variational AutoEncoder, an architecture that combines the versatility of unsupervised learning with the straightforwardness of supervised learning. The developed model is able to understand and represent the dimensional latent space underlying each categorical model and, therefore, to project each categorically annotated data-point into a shared continuous space. Thus, the model learns to encode each input into a disentangled embedded representation, storing the continuous real factors that generated it.

By proposing a tool to disentangle categorical annotation into their true dimensional components, I hope to provide an insight on how to link classification and regression task, two important elements of machine learning.

Contents

List of Figures

List of Tables

1 | Introduction

In the last decade computer science has experienced an exponential development, becoming part of everyone's daily life. The revolution, that modern artificial intelligence caused, can't be ignored; the shift in paradigm from statistical hand-crafted methods to more general purpose machine learning (ML) algorithms, especially thanks to neural networks, represents a milestone in current society's history. As a branch of AI, Natural Language Processing has also undergone drastic changes that have led to an incredibly fast progress. The shift from probabilistic/statistical models to neural networks and the use of increasingly sophisticated architectures has made it possible to obtain incredible performances. Nowadays, the state-of-the-art algorithms generally involve an adaptation of a sequence-to-sequence system to the given problem: pre-trained Transformer based models such as BERT [8] or GPT-2 [27] are fine tuned for a specific task and, through their, pre-learned knowledge are able to adapt to the new context with ease. Although challenges such as machine translation or text summarization can, therefore, be considered solved, it is still very difficult to explain what happens behind the surface: all these advanced systems are mainly end-to-end, behaving as black boxes. This undermines the explainability of these processes and, consequently, their scalability and applicability to exotic environments, such as multi-modal ones, or to more elaborate tasks such as emotion recognition, where the availability of training data is not sufficient to obtain perfect models, given the subjectivity of the task.

Modern architectures work by generating word and sentence embeddings: vectors able to store a compressed but useful representation of each utterance's content and characteristics. Although it is possible to capture intermediate embeddings by employing architectures such as auto-encoders, it is difficult, if not impossible, to perform an analysis and manipulation on them at a symbolic level to understand their actual meaning. This undermines the possibility of expanding and adapting the model to novel applications, as costly retraining is always required. This is particularly true as architectures get more and more complex; for simpler scenarios, such as when using Word2vec to generate word embeddings, such symbolic operations between elements are still possible and useful, albeit limited [21] [22]. Therefore, it is fundamental to have an insight into the functioning of the different blocks that make up a network and how they cooperate, both to guarantee the explainability of internal processes and to be able to divide architectures into reusable and modular units, in order to obtain scalable systems. Reaching a greater possibility of analysis and, consequently, a greater trust in artificial intelligence is an objective considered very important even outside the scientific community, having an immense repercussion in socio-economic areas.

Explainability of neural networks (i.e. understanding in a qualitative and quantitative way what each layer is learning to represent and what kind of operations it performs) can be achieved in different ways. A concept that has become very popular in recent years is disentanglement, referred to as the ability to obtain embeddings in which each encoding factor represents an independent feature of the data and, therefore, can be analyzed with ease (Fig. 1.1). Disentanglement, however, is still an active research field and numerous papers are continuously published, demonstrating new techniques to be applied to new data formats (e.g. images or text) and with different objectives (e.g. unsupervised disentanglement, where the model has to find the factors to be disentangled, versus targeted disentanglement, where the model needs to

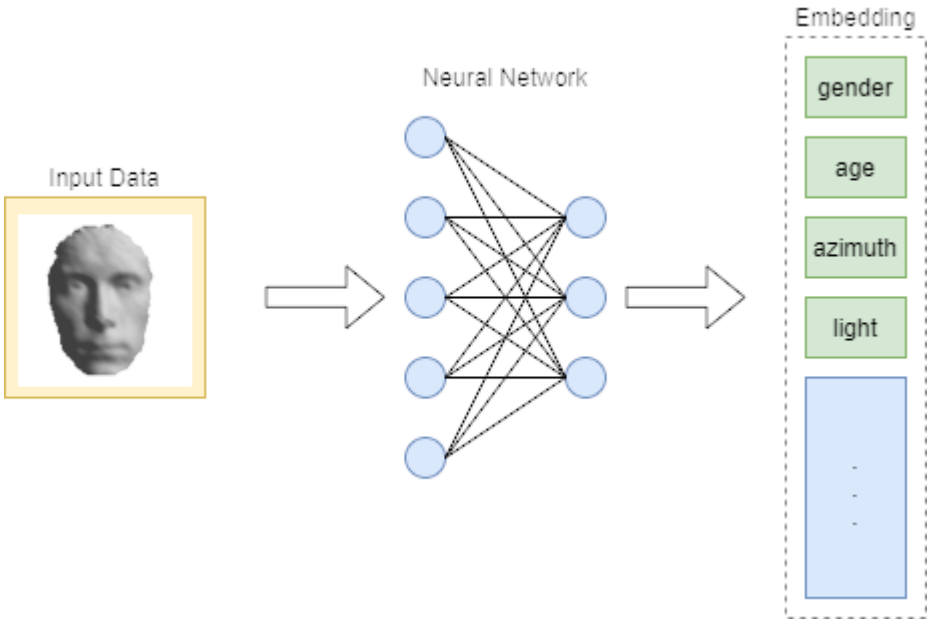

Figure 1.1: Example of image embedded into a disentangled representation. Picture taken from the UoY 3D face dataset [11].

disentangled specific target features). To the best of my knowledge, however, nobody has yet tried to apply disentanglement to cover the gap between categorical and continuous data, bridging two pillars of machine learning: classification and regression. Within this work, I suggest a first attempt to achieve so, by providing an architecture able to disentangle categorical labels into the generative real values behind them, allowing, defacto, the projection of each category in a geometrical latent space, enabling all the analysis tools typical of those spaces, and, in particular the possibility of merging different categorical models into a single dataset. With this contribution, I hope to inspire novel ways to achieve explainability within neural networks.

1.1 Contributions

This project has been carried on completely independently by myself. Based on existing knowledge and architectures, new insights and methods are provided to improve explainability inside neural networks. In particular, a way to solve some of the

difficulties present in Emotion Recognition in textual data is developed. The main contributions can be listed as following:

- Detailed overview of what Emotion Recognition (in Conversation) is and of its possible problem settings. Some of the difficulties that arise when working with it are highlighted. In particular, I describe the data sparsity problem and suggest a way to deal with it;
- Analysis of existing methods used, or usable, to solve the problem, but fail because of specific pitfalls;
- Suggestion and development of a new architecture capable of providing a solution to it. Experimental results are provided to show the feasibility of the task, and how the method can be applied to more general scenarios;
- Development of further techniques that define a general-purpose framework to disentangle categorical models into their continuous factors, allowing for a better model's explainability.

1.2 Thesis overview

This work is divided into four main chapters:

- **Chapter 2** describes the general background in which this work was developed. The ideas that inspired the project are illustrated, providing the reader with the key concepts that allow to understand the problem statement. Furthermore, an in-depth overview of already existing methods that try (or can be used) to solve it is presented. Special attention is given to the Emotion Recognition task, the different autoencoder architectures and the concept of disentanglement.

- **Chapter 3** illustrates the datasets and architectures used to obtain the results. In particular, it highlights the contributions made to existing models to solve their issues and enable new capabilities.

- **Chapter 4** shows how it is possible to use the developed architecture, Multi-Type Continuous Disentanglement Variational AutoEncoder (MTCD-VAE), to project categorical labels into a continuous latent space. This shows the feasibility of discovering the generative factors behind each category and, therefore, provides a way of merging different models together.

- **Chapter 5**, finally, contains further results that were not initially foreseen and shifted this thesis towards a more theoretical target, analyzing disentanglement from a general perspective, instead of focusing on Emotion Recognition. In this way, I hope to provide a general purpose framework that can be used to reach better explainability inside neural networks.

2 | Background and problem settings

2.1 The importance of data

Modern artificial intelligence has undergone drastic changes that have led to an exponential development. The key element behind its success, both theoretical (providing excellent experimental performance) and practical (being easily scalable to large commercial applications), is the versatility of neural networks, as they don't require expensive domain expertise to solve a specific problem and can be freely applied to a wide variety of tasks. Instead, they rely on huge quantities of data to autonomously learn the expert knowledge that, in a more canonical scenario, a human would have had to encode inside the model.

Although there have been important developments in both unsupervised (e.g. Transformer-based architectures able to embed a large amount of information through an initial unsupervised training phase [8]) and reinforcement learning (e.g. Google DeepMind's Alpha Go and its ability to take down the Go world champion with ease [31]) in the last years, the majority of applications and industries still mostly utilize supervised learning to produce real-world tools and fine-tune them to specific downstream tasks. It does not come as a surprise, therefore, that the performance and quality of the results obtained by a model is directly linked to the amount of high-quality data available during the training phase. A clear example of this phenomenon can be seen in machine translation: state-of-the-art models are nowadays

considered almost indistinguishable from human-performed translation and can even outperform it in specific tasks and metrics [25]. However, these results only apply in scenarios where both source and target languages have a large text corpora available; with low-resources languages, all the methods are far from reaching the same level of performance [1]. In particular, it has been observed by Koehn [16] that, compared to traditional statistical machine translation, neural machine translation "have a steeper learning curve with respect to the amount of training data, resulting in worse quality in low-resource settings, but better performance in high-resource setting" .

2.1.1 Neural networks

Neural networks are architectures based on a series of connected units, called neurons, that are organised in layers. Each neuron performs the following basic operation: given a input vector $\boldsymbol{x}$ and a sub-differentiable activation function f, the neuron computes

$$\boldsymbol{y} = f(\boldsymbol{w} \cdot \boldsymbol{x} + \boldsymbol{b}) \tag{2.1}$$

where $\boldsymbol{w}$ and $\boldsymbol{b}$ are internal parameters of each neuron, called weight and bias, and can be learned during training. Neural networks are basically multi-dimensional differentiable functions that, given an input $\boldsymbol{x}$, adapt to produce the desired result $\boldsymbol{y}$. Equation 2.1 is the basis of the feedforward operation: $\boldsymbol{x}$ is processed sequentially by each layer and eventually the response $\boldsymbol{y}$ is output. Formally, a standard fully-connected neural network can be described as following. Be D the number of layers composing the model, each one consisting of N_l neurons, where $N_0 = d$ is the dimension of the input and N_{D-1} the one of the output. Each layer is associated with a matrix W_l and a vector $\boldsymbol{b_l}$, that store the values for each individual neuron's weight

and bias. The feedforward dynamics is, therefore, given by:

$$\boldsymbol{x}^l = f^l(\boldsymbol{h}^l),\ \boldsymbol{h}^l = W^l \cdot \boldsymbol{x}^{l-1} + \boldsymbol{b}^l, \qquad l \in 1, 2, ..., D-2 \tag{2.2}$$

$$\boldsymbol{y} = f^{D-1}(\boldsymbol{h}^{D-1}),\ \boldsymbol{h}^{D-1} = W^{D-1} \cdot \boldsymbol{x}^{D-2} + \boldsymbol{b}^{D-1} \tag{2.3}$$

Given the output $\boldsymbol{y}$ and a target value $\hat{\boldsymbol{y}}$, an "error" value, representing the performance of the neural network, can be computed using a so-called *loss function*:

$$l = loss(\boldsymbol{y}, \hat{\boldsymbol{y}}) \tag{2.4}$$

By computing the gradient of l with respect of each W_l and b_l, it is possible to update their values (e.g. by taking a fixed-length step in the opposite direction given by the gradient), eventually reaching a local minimum for the loss function, thus trying to minimize the model's predictive error and maximizing the similarity between $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$. This step is called error back-propagation and allows neural networks to learn autonomously. In theory, after a training phase performed over a set of samples $\{(\boldsymbol{x_0}, \hat{\boldsymbol{y_0}}), (\boldsymbol{x_1}, \hat{\boldsymbol{y_1}}), ...\}$, the model should be able to generalize over unseen data.

2.1.2 Different types of learning

As described above, neural networks and, more in general, machine learning algorithms are able to learn to mimic complex high-dimensional functions. In order to achieve that, however, a loss function and a target value for each training datapoint are generally required. How these are specified determines the typology of learning problem we are trying to solve. We can identify three main categories:

- **Supervised learning**: a supervised model requires both a loss function and a target value $\hat{\boldsymbol{y}}$ for each training sample and is able to construct a mono-directional mapping between two spaces (e.g. from pictures $\in \mathbb{R}^{w \cdot h}$ to labels

of objects). Given an unseen new input it will produce a predicted output. Classification and regression problems are both examples of supervised learning problems;

- **Unsupervised learning**: compared to supervised learning, unsupervised models do not require an external target $\hat{\boldsymbol{y}}$, but can instead compute the loss function only by using the input $\boldsymbol{x}$ and the generated output $\boldsymbol{y}$. The main advantage is that they do not need costly annotated data to be trained or perform. Examples are clustering algorithms or autoencoders;

- **Reinforcement learning**: algorithms of this category are based on an agent that is able to interact with a real or simulated environment. The inputs consist of the sensory data it can query and outputs are determined by the actions it performs. Although no prior data collection is necessary, the training process is generally slower and costly (because of the hardware or complex simulations required).

With the exception of reinforcement learning, a method that, despite its huge potentiality, is still in its infancy due to its limited applicability in real-world domains, both supervised and unsupervised learning requires a lot of training data to produce satisfactory results. Therefore, as mentioned earlier, the availability of many high-quality datasets is of fundamental importance. Combining the two approaches, in what is called semi-supervised learning, has proven very successful in the last years. The method proposed can be considered to belong to this category.

2.2 Emotion Recognition in Conversation

A field in which the lack of sufficient availability of high-quality data is quite evident is Emotion Recognition (ER) and, especially, Emotion Recognition in Conversation

(ERC). As it can be evicted from the name, these natural language processing (NLP) tasks consist in the ability to mine underlying opinions and emotions of a text corpora. Due to the huge amount of comments, discussions, and controversial expressions publicly available on platforms, such as Reddit or Facebook, it is considered by the scientific community an important step in producing autonomous agents, able to fully understand the meaning of a sentence and to reply generating an emotion-aware dialogue. However, it still consists a very difficult problem, and state-of-the-art algorithm are nowhere close to human like performance. ERC, especially, is considered a real challenge, as, compared to vanilla emotion recognition, it requires context modelling and keeping track of the current state of the conversation. In order to fully understand a dialogue, in fact, emotion understanding is essential, as emotions can deeply change the meaning of an utterance: an angry "fine" is completely different from an happy one, and sarcasm can significantly twist what an interlocutor is trying to convey. On the other hand, it is the dialogue and the context itself to change the feelings of the people involved. This intricate emotion-context relationship makes ERC an incredibly complex task.

As stated before by Koehn [16] and as generally experienced in any ML setting, using more data during training is generally the safest bet to overall increase performance, as it allows to construct more complex models and gives them the ability to efficaciously generalize the knowledge learned, as the more information is available, the less prone the model is to overfitting. However, due to the nature of the task itself, it is difficult to produce high-quality reusable datasets. For this reason, semi-supervised learning has recently been proven very effective in improving the performance in ERC down-stream tasks, thanks to a technique called transfer-learning [10]. In Poria et al. [26] the main challenges linked to producing ERC datasets and a description on available ones are reported: as emotions are an incredibly subjective topic, it is hard to define an agreed-upon standard on how to generate generally useful

	EmoContext	IEMOCAP	Emotionlines	DailyDialog
Neutral		x	x	x
Happiness	x	x	x	x
Sadness	x	x	x	x
Anger	x	x	x	x
Frustrated		x		
Excited		x		
Disgust			x	x
Fear			x	x
Surprise			x	x
Other	x			

Table 2.1: Emotion taxonomies used by different datasets.

labelled data. In particular, the trade-off between dataset completeness and accuracy needs to be mentioned, as complex emotion models increase the risk of obtaining a lower inter-annotator agreement during the labelling phase, due to the subjectivity of the task.

2.2.1 Categorical and dimensional models

The majority of ERC datasets (e.g IEMOCAP [4], Emotionlines [6], DailyDialog [18], and EmoContext [5]) consist of collections of utterances and categorical emotional labels associated to them, where categorical means assigning, to each data sample, one (or more) label chosen from a fixed discrete set of categories. In the last century, many emotional taxonomies have been proposed and, consequently, different datasets use different systems, trying to specifically address the task they were designed for. EmoContext, for example, subdivides emotions in four categories (i.e. Happiness, Sadness, Anger, Other), IEMOCAP uses six of them (i.e. Neutral, Happiness, Sadness, Anger, Frustrated, Excited), while Emotionlines and DailyDialog rely on the scheme proposed by Ekam [9] (Table 2.2.1).

On the other hand, there exist dimensional models that offer a multi-dimensional space in which each sentence can be uniquely represented by a vector of real num-

bers, according to its emotional nuances. Each axis, therefore, stores an independent hidden factor that determines some of the sentence's characteristics, enabling the mapping between emotions and a continuous spectrum of values, avoiding hard (and limiting) categories. A commonly used model is the Valence, Arousal, and Dominance (VA+D) one, proposed by Russel [29]). The key feature of dimensional models is the possibility to easily compare emotional states using vector operations, and, therefore, to relate different datasets by projecting all the utterances in the same geometrical space. To some extent, it is relatively easy to merge multiple dimensional datasets into a standard format. The main drawback of such approach is that it is challenging to initially assign the correct values to each emotion, as annotators would struggle to reach an agreement, given the continuous range of possibilities. To the present date, only a couple of relevant dimensional datasets are available: SEMAINE [20] and EmoBank [3], both relying on slight modifications of the Valence and Arousal framework.

Compared to dimensional models, categorical datasets are, therefore, easier to generate, as the labelling process is easier and annotators can rely on self-introspection to identify the precise emotion that an utterance provokes in them (compared to assigning a set or real numbers based on some ideal scales). However, categorical datasets are difficult to work with, as comparison between models is not trivial, since there is not an explicit relation between multiple labelling systems, nor between the categories used in each single dataset. Therefore, given a specific task, it is difficult to merge different data sources to obtain a larger shared training corpus and, consequently, ERC suffers of a severe data sparsity problem. The methods suggested in this work try to address the issue, proposing a general framework to relate categorical models across datasets, projecting them in the same geometrical continuous space, offering the advantages of both worlds (i.e., simplicity of use of

categorical information and possibility to combine multiple sources through their dimensional representation).

2.3 Using multiple datasets

Merging two datasets involves understanding how the different labels are related to each other, in order to correctly create a new categorical model to be used in the produced merged dataset. However, categorical data, compared to dimensional, do not contain any information that can help in that regards. Although from a human perspective it is clear that, for example, happiness and sadness are opposites while excitement and surprise may share multiple nuances, it is not obvious how this *similarity metric* can be formalized and, therefore, how to understand the true meaning of each label. Formally, each category is a unique identifier that groups together several items, without providing any data on which features they have in common and what, instead, distinguish them from other sets. It seems, therefore, a natural step to try to convert a categorical model to a dimensional one, in order to unlock all the vector operations and properties of a geometrical space that would enable the creation of precise distance metrics. With these new tools, it becomes easier to understand the relationship between different labels by comparing the distributions of utterances and their emotions, once projected in the new space. We aim, therefore, to solve a regression problem (i.e., predicting the value for each independent factor generating a sentence) by means of using exclusively categorically labeled training data, normally designed for classification models.

2.3.1 Classification and regression

Classification and regression are both supervised learning tasks that aim to learn a mapping function from an input space X to an output space Y. The main difference

of the two methods is the typology of the target space Y.

- **Classification**: Y is a set of predefined discrete classes that does not posses any order or distance properties. A model $\mathcal{M}_c$ learns to predict the labels' distributions given an input $\boldsymbol{x_i}$, i.e. $p(l|\boldsymbol{x_i})$, by means of a loss function that maximize the probability of belonging to the right class, increasing the overall model's classification accuracy. Due to the nature of the space, the model is not able to represent any meaningful relationship existing between the different categories.

- **Regression**: Y is a continuous space (generally a subset of $\mathbb{R}^n$) on which precise notions of order and distance are defined. A model $\mathcal{M}_l$ is able to learn a continuous mapping between X and Y, usually by minimizing the distance between the output produced by each training sample and its true associated value $\boldsymbol{v}$. $\mathcal{M}_l$, by learning the distributions $p(\boldsymbol{v}|\boldsymbol{x_i})$, is also able to represent the relationships between multiple data points, given by the space's properties. Comparison between values is, therefore, easier and more insights can be obtained from the data.

The two tasks seem incompatible. Assume, however, that the features characterizing each element $\boldsymbol{x}$ are uniquely determined by a set of underlying continuous generative factors $\boldsymbol{v} = (v_1, v_2, ..., v_k)$, such that, given the true world simulator Sim, $p(\boldsymbol{x}|\boldsymbol{v}) = Sim(v)$. Since the categories used in a classification problem are obtained by grouping together elements that appear similar (e.g., by labelling them with the same value), then it is true that each group can be defined through a subset of the generative factors' values. Therefore, a regression problem can be transformed to a classification one by applying a clustering algorithm on the space Y. Unfortunately, inverting the process (i.e., disentangling each label's value into its generating factors) is not trivial, due to the loss of information that occurs when merging multiple continuous values

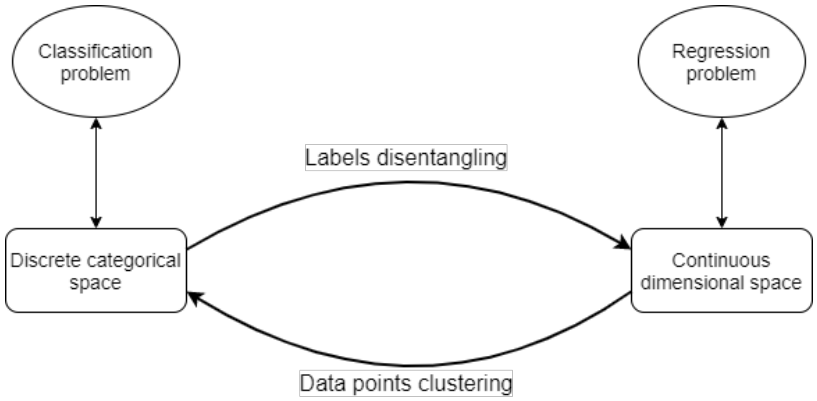

Figure 2.1: Reduction between classification and regression problems.

into a single category (Fig. 2.1). It would allow, however, to reduce one problem to the other and vice versa, exploiting advantages of both.

2.3.2 Problem statement

The techniques developed within this work can be applied to any domain, but, throughout this report, the ERC task will be used to explain the significance of the obtained results and how they can be applied on a practical problem.

Consider N datasets D_1, D_2, ..., D_N, each containing a set of S_i training sample points, $D_i = \{(\hat{\boldsymbol{x}}_i^1, \hat{y}_i^1), (\hat{\boldsymbol{x}}_i^2, \hat{y}_i^2), ..., (\hat{\boldsymbol{x}}_i^{S_i}, \hat{y}_i^{S_i})\}$, which refers to the same data domain, but is labelled according to a different categorical model of size C_i: $\hat{y}_i \in L_i = \{\hat{l}_i^1, \hat{l}_i^2, ..., \hat{l}_i^{C_i}\}$. The goal is to generate a merged dataset $D = \bigcup_{i\in\{1,...,N\}} D_i$ such that each data point $(\boldsymbol{x}, y) \in D$ is consistent with its original categorical system. Consistency here means that, if we assume that there is a set of true independent generative factors $(v_1, v_2, ..., v_k) = \boldsymbol{v} \in \mathbb{R}^k$ such that they uniquely determine how each utterance is labelled according to $L = \bigcup_{i\in\{1,...,N\}} L_i$ (i.e., they define a probability distribution over L such that $\forall l \in L,\ p(l) = \int p(l|\boldsymbol{v})\, d\boldsymbol{v}$), then the following equations must hold:

$$\forall(\boldsymbol{x}, y) \in D_i,\ p_{L_i}(\boldsymbol{v}|\hat{x}, y) \approx p_L(\boldsymbol{v}|\hat{x}, y),\ \forall i \in \{1, ..., N\} \tag{2.5}$$

$$\forall \hat{l} \in L_i,\ \exists! l \in L :\ p_{L_i}(\boldsymbol{v}|\hat{l}) \approx p_L(\boldsymbol{v}|l),\ \forall i \in \{1, ..., N\} \tag{2.6}$$

where p_M stands for the probability distribution computed accordingly to the model M. In particular, given a sample $(\boldsymbol{x}, y) \in D$, $y \in L$, we aim to compute

$$\begin{aligned} p_L(y|\boldsymbol{x}) &= \int_V p_L(y, \boldsymbol{v}|\boldsymbol{x})\, d\boldsymbol{v} = \int_V p_L(y|\boldsymbol{v}, \boldsymbol{x}) \cdot p_L(\boldsymbol{v}|\boldsymbol{x})\, d\boldsymbol{v} = \\ &= \int_V p_L(y|\boldsymbol{v}) \cdot p_L(\boldsymbol{v}|\boldsymbol{x})\, d\boldsymbol{v} \end{aligned}$$

where the last equation is justified by our initial assumption and V is the $k-$dimensional vector space defined over $v_1, v_2, ..., v_k$. This implies, for example, that

$$\forall \boldsymbol{v} \in V,\ p(\hat{l}_i^h|\boldsymbol{v}) \approx p(\hat{l}_j^k|\boldsymbol{v}) \implies \forall \boldsymbol{x},\ p(\hat{l}_i^h|\boldsymbol{x}) \approx p(\hat{l}_j^k|\boldsymbol{x}) \tag{2.7}$$

Equation 2.7 states that, if two labels $\hat{l}_i^h$ and $\hat{l}_j^k$ refer to the same category of data (i.e.e, they are generated by the same factors $\boldsymbol{v}$, despite being referred as different entities in L_i and L_j), we should be able to conclude that $\hat{l}_i^h = \hat{l}_j^k$, since, from the data, it would be impossible to distinguish them. Analogously, if two categories share similar generative factors, they should be mapped to similar distributions in V.

As mentioned earlier, working with a dimensional model has many advantages compared to a categorical approach and, therefore, it would be ideal to focus on and to compute the probability distribution $p(\boldsymbol{v}|\boldsymbol{x})$, given access to D_1, D_2, ..., D_N. However, the structure of V itself is unknown. In other words, to construct each vector $\boldsymbol{v}$, we need to discover how many dimensions are necessary and sufficient to fully represent L and how the different labels map to different subsets of the chosen vector space. Therefore, we aim at determine the number of true generative factors, k, and to learn, given a sample $(\boldsymbol{x}, y)$, $y \in L$, the following probability distributions:

$$p(v_1, v_2, ..., v_k|\boldsymbol{x}, y) \tag{2.8}$$

and, ideally,

$$p(v_1, v_2, ..., v_k|\boldsymbol{x}) \tag{2.9}$$

$$p(v_1, v_2, ..., v_k|y) \tag{2.10}$$

By knowing distribution 2.9, we are able to compute $\boldsymbol{v}$ without the need of categorical annotation and, therefore, we can enrich our dataset D with new unseen data points. Distribution 2.10, instead, represents a direct mapping from the initial distinct categorical models to the same shared continuous space, allowing the creation of a single combined dataset.

Furthermore, if a labelling system is actually necessary for a particular task, a clustering algorithm can be used to determine L, by considering the data points projections $\bar{\boldsymbol{x}}$ in the space V, obtained by considering the maximum likelihood approximation

$$\bar{\boldsymbol{x}}_{\boldsymbol{i}} = \arg\max_{\boldsymbol{v}} p(\boldsymbol{v}|\boldsymbol{x}_{\boldsymbol{i}}) \tag{2.11}$$

and a simple classification network can then be trained to learn the mapping from V to L.

2.3.3 Merging emotions

Russel [29] theorized that any emotion is uniquely determined by three independent factors: Valence, Aruousal, and Dominance. Therefore, in theory, it should be possible to project all the emotion labels showed in Table 2.2.1 to a three dimensional space, computing their position along each axis. Although, through practical psychological experiments it is possible to estimate rough boundaries for every value, each person has a different emotional perception and, across different annotators, the predicted positions may be insufficiently precise or, in the extreme case, not correct. The flexibility that deep learning posses could be a key element to construct a model

able to adjust to any combination of datasets, by autonomously learning the distributions of the true generative factors $\boldsymbol{v}$ linked to each emotion label. This would represent the shift from statistical methods, which require pre-existing knowledge, to neural networks, which has already proven so beneficial in many tasks (e.g., machine translation and image recognition). The challenges that a model would need to solve are many:

- Understanding which labels across multiple datasets refer to the same (or similar) emotional values and correctly merge them in a unique category (e.g., Joy and Happiness, generally considered synonyms);
- Vice versa, critically evaluate if the same label is used consistently to classify similar utterances, as, due to the subjectivity of the annotation task, each emotion could be considered having different nuances by different people in different settings;
- Correctly understand the relationships between categories, in order to represent those utterances that do not clearly belong to a specific group but, instead, share features with different emotional values. We can, in fact, exploit the full potentiality of a continuous space, by projecting ambiguous sentences in the space between different clusters, interpolating their position based on their true generative values;
- Verify that the VAD dimensional model (i.e. $\mathbb{R}^3$) is sufficient and necessary to represent emotions.

With these properties satisfied, any categorical dataset could be projected in the same space, allowing to solve the data sparsity problem that affects ERC. Furthermore, by extrapolating $\boldsymbol{v}$ from a given $\boldsymbol{x}$, it would be possible to obtain more insights on its properties, than just relaying on its discrete classification.

2.4 Existing Approaches

There are different approaches described in literature that can be adapted to solve the above-mentioned task. Firstly, we focus on a supervised model introduced in 2019 by Park et al. [24], targeting specifically Emotion Recognition, to describe what needs to be improved to deal with the new problem settings. Secondly, we provide an overview on a well-known unsupervised architecture used to extract latent representations from the provided data, i.e., autoencoders, as the developed solution is based on an augmented version of it.

2.4.1 Toward Dimensional Emotion Detection from Categorical Emotion Annotations

A first attempt towards dimensional emotion detection from categorical annotation has been proposed by Park et al. [24]. As they state, the model developed in the paper is able to predict fine-grained emotional values, requiring only coarse-grained categorical labels during the training phase. In particular, they rely on the VAD framework and on the NRC-VAD Lexicon [23], in order to project each category into a continuous space, $V = (v, a, d)$ (Fig 2.2). Their approach consists in computing the distribution $p(v, a, d|\boldsymbol{x}) \approx p(v|\boldsymbol{x}) \cdot p(a|\boldsymbol{x}) \cdot p(d|\boldsymbol{x})$ for each utterance $\boldsymbol{x}$ and by predicting its continuous VAD score as expectation of each distribution:

$$v_{score} = \mathbb{E}[p(v|\boldsymbol{x})]$$

$$a_{score} = \mathbb{E}[p(a|\boldsymbol{x})]$$

$$d_{score} = \mathbb{E}[p(d|\boldsymbol{x})]$$

In order to approximate $p(\cdot|\boldsymbol{x})$, they consider the ordered sequence of labels along each axis, according to the NRC-VAD Lexicon's values and, successively, they mini-

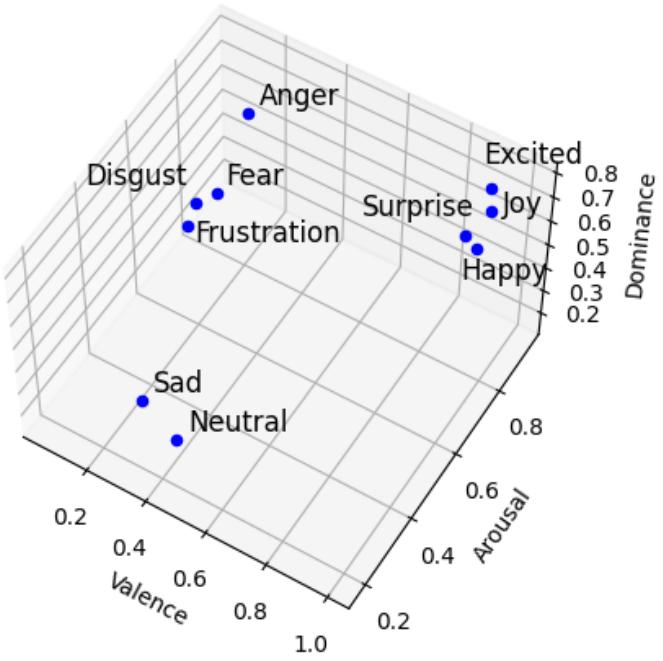

Figure 2.2: Emotion labels projected into the VAE space.

mize the distance between the true distribution $\hat{p}(\cdot|\boldsymbol{x})$, obtained from the data, and predicted one, $p(\cdot|\boldsymbol{x})$. For example, in the categorical model $C = \{joy, sad, happy, anger\}$, the four emotions, whose Valence values are, respectively, $(0.980, 0.225, 1.0000, 0.167)$, will be sorted on the v axis as $(anger, sad, joy, happy)$. The chosen order is of fundamental importance when trying to predict intermediate emotional states. If we restrict the example above to a one dimensional scenario, $e = (v)$, representing a value between *sad* and *joy* would be natural. However, it would be hard to interpolate exclusively between *anger* and *happy*, without recurring to a multi-modal distribution, which would be hard to approximate with a point estimate (e.g., by using a maximum likelihood approach). Furthermore, the loss function used by Park to train the model is the Earth Movers Distance (EMD) [14], designed to give more penalties when a class far from the correct one is predicted:

$$EMD(p, \hat{p}) = \sum_{i=1}^{C} (CDF_i(p) - CDF_i(\hat{p}))^2 \tag{2.12}$$

where CDF is the cumulative distribution function. The performance of a model trained with this loss function is, therefore, susceptible to the emotion labels' chosen order and displacement among each axis. Thus, the usage of a priori knowledge may result in inaccuracies, requires domain-specific data to be collected, and prevents the direct applicability of the developed method to other problem settings. In ERC, in particular, it is not considering the subjectivity of the annotation task that, by its nature, can't rely on fixed values, as each dataset will present slightly different emotion distributions. In addition, there is no clear evaluation metric that can help in understanding the correctness of the initial choice and how it is influencing the observed performance. Therefore, a solution able to learn and modify its core parameters would be preferable. Given the success obtained by deep neural networks, they seem a natural and immediate choice.

2.4.2 Autoencoders: an example of unsupervised learning

Autoencoders were introduced in the 1980s by Hinton [28] as a tool to *"learn without a teacher"*, by using exclusively the input data to compute and backpropagate the loss of the network. Recently, thanks to the advances in deep learning, they have become extremely popular. Modern models consist of three main components: an encoder, a decoder, and an intermediate information bottleneck (Fig. 2.3). The goal of the architecture is to learn, in an unsupervised fashion, compact and simplified representations, generally called embeddings, of the given sample data, capable of representing their key characteristics. Both encoder and decoder are multi-layer neural networks that are trained to learn the mapping between the high-dimensional data domain and the low-dimensional embeddings' latent space. The intermediate bottleneck between the two guarantees that the representation learned is efficient and stores only essential information.

The architecture is firstly trained bottom-up, by learning a task-agnostic internal

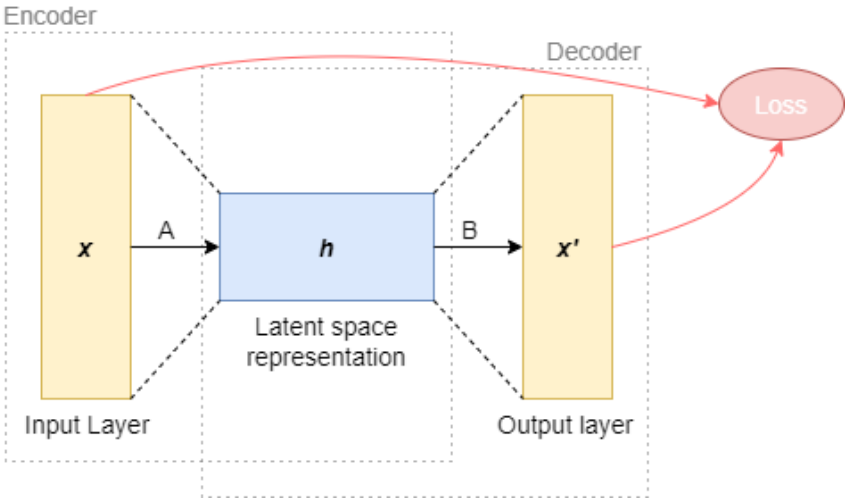

Figure 2.3: Basic autoencoder architecture.

representation, and then fine-tuned on a specific downstream task, generally by substituting the decoder with a supervised classification (or regression) network. The knowledge embedded into the model through the initial step helps obtaining state-of-the-art performance in many tasks. Formally, an autoencoder is defined by the tuple $(D, \mathbb{X}, \mathbb{H}, n, h, \mathcal{A}, \mathcal{B}, \Delta)$, where:

- D is the training dataset, composed by the data points $\{\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_m}\}$, with $\boldsymbol{x_i} \in \mathbb{X}$;
- $\mathbb{X}$ and $\mathbb{H}$ are sets (e.g., $\mathbb{R}, \mathbb{N}$, ...);
- n and h, $h < n$ are integers, representing the size of the input data space, $\mathbb{X}^n$, and latent space, $\mathbb{H}^h$;
- $\mathcal{A}$ is a class of functions from $\mathbb{X}^n$ to $\mathbb{H}^h$;
- $\mathcal{B}$ is a class of functions from $\mathbb{H}^h$ to $\mathbb{N}^n$;
- Δ is a dissimilarity function defined over $\mathbb{X}^n$.

Then, given a sample $\boldsymbol{x_i}$, the following operations are performed:

1. **Encoding**: $\boldsymbol{h_i} = A(\boldsymbol{x_i})$, with $A \in \mathcal{A}$ is the function represented by the encoder;

2. **Decoding**: $\hat{\boldsymbol{x}}_{\boldsymbol{i}} = B(\boldsymbol{h}_{\boldsymbol{i}})$, with $B \in \mathcal{B}$ is the function represented by the decoder;

3. **Error backpropagation**: $l = \Delta(\boldsymbol{x}_{\boldsymbol{i}}, \hat{\boldsymbol{x}}_{\boldsymbol{i}})$ is backpropagated through the network.

Therefore, the goal of the architecture is to find $A \in \mathcal{A}$ and $B \in \mathcal{B}$ that minimize the overall dissimilarity loss

$$L = \min_{A,B} \sum_{i=1}^{m} \Delta(B \circ A(\boldsymbol{x}_{\boldsymbol{i}}), \boldsymbol{x}_{\boldsymbol{i}}) \tag{2.13}$$

and consequently, build the most informative latent representation $\boldsymbol{h}_{\boldsymbol{i}}$. Since we are interested in identifying the true generative factors $\boldsymbol{v}$ of a given sample $\boldsymbol{x}$, analysing $\boldsymbol{h}$ could provide useful insight.

Autoencoders are deterministic networks: for each input point, the encoder is producing a single value for each encoding dimension. Therefore, they treat each sample individually and may fail to learn the relationships between different points and their features, overfitting to the training data. As the interest of this research is towards interpolations between several categorical labels, it must be avoided. Instead, a continuous and meaningful representation of the input data in the latent space should be preferred.

2.4.3 Variational Autoencoders (VAE)

Variational Autoencoders are an improvement over traditional autoencoders proposed by Kigma and Welling [15]. Instead of employing an encoder that generates a single value per latent dimension, they provide a probabilistic manner for describing an observation in the latent space, by outputting a statistical distribution over the different possibilities. A single number (for each latent unit) is then randomly sampled from its distribution and fed into the decoder. Therefore, instead of learning a single mapping for each datapoint, the model is able to predict the correct output from a

range of values. This implies that, in order to maintain reasonable performances, we are enforcing a continuous latent representation, as values which are nearby to one another in the space must correspond with very similar reconstructions. By means of this new smoothness assumption, it is guaranteed that utterances whose emotional values are intermediate between main categories, will be correctly projected close to them, by interpolating their final position.

Formally, we would like to predict we value of $\boldsymbol{h}$ given an observation $\boldsymbol{x}$, $p(\boldsymbol{h}|\boldsymbol{x})$. By Bayes' rule we have:

$$p(\boldsymbol{h}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\boldsymbol{h}) \cdot p(\boldsymbol{h})}{p(\boldsymbol{x})} \tag{2.14}$$

Unfortunately, in most cases, getting $p(x)$ requires computing an high-dimensional integral

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}|\boldsymbol{h}) \cdot p(\boldsymbol{h}\, d\boldsymbol{h}) \tag{2.15}$$

which is intractable. Instead, we try to approximate, in a variational manner, the posterior $p(\boldsymbol{h}|\boldsymbol{x})$ through a family of distributions $q_\theta(\boldsymbol{h}|\boldsymbol{x})$, where θ represents the encoder's parameters. Furthermore, to reconstruct the input from the latent, the decoder computes the likelihood function $p_\phi(\boldsymbol{x}|\boldsymbol{h})$, where ϕ are the decoder's parameters. In order to learn θ and ϕ, the model uses a double-objective loss function:

1. **Maximize the reconstruction similarity**, by maximizing the expected value over $\boldsymbol{h} \sim q_\theta(\boldsymbol{h}|\boldsymbol{x})$ of the log likelihood:

$$l_{rec} = \mathbb{E}_{\boldsymbol{h}\sim q_\theta}[\log(p_\phi(\boldsymbol{x}|\boldsymbol{h}))] \tag{2.16}$$

2. **Minimize the prior approximation error**, by minimizing the KL divergence [17] between the prior p and $q_\theta(\boldsymbol{h}|\boldsymbol{x})$:

$$l_{KL} = D_{KL}(q_\theta(\boldsymbol{h}|\boldsymbol{x})||p(\boldsymbol{h})) \tag{2.17}$$

Together, equations 2.16 and 2.17, define the $ELBO$ (Evidence Lower BOund) function:

$$ELBO = l_{rec} - l_{KL} \tag{2.18}$$

As shown in Appendix A, maximising the $ELBO$ loss function, corresponds to minimizing the KL divergence between the true posterior $p(\boldsymbol{h}|\boldsymbol{x})$ and the approximation $q_\theta(\boldsymbol{h}|\boldsymbol{x})$, $D_{KL}(q_\theta(\boldsymbol{h}|\boldsymbol{x})||p(\boldsymbol{h}|\boldsymbol{x}))$. The choice of the prior distribution $p(\boldsymbol{h})$ deeply influences the model behaviour. Firstly, setting it to be the isotropic unit Gaussian, $\mathcal{N}(0, I)$, allows for an analytical form of the KL divergence:

$$l_{KL} = \sum_j \frac{1}{2}(1 + \log(\boldsymbol{\sigma_\theta})_j^2 - (\boldsymbol{\sigma_\theta})_j^2 - (\boldsymbol{\mu_\theta})_j^2) \tag{2.19}$$

where $\boldsymbol{\mu_\theta}$ and $\boldsymbol{\Sigma_\theta}$ (diagonal matrix whose non-zero elements are $\boldsymbol{\sigma_\theta}$) are the output of the encoding operation and represent $q_\theta(\boldsymbol{h}|\boldsymbol{x})$; j indexes the latent space dimensions. Secondly, by using a diagonal co-variance matrix, we force the embedding's units to be independent from each other, minimizing the mutual information and, therefore, maximizing the efficiency of the latent representation. Since, in order to understand the meaning of each categorical label, we are interested in identifying its true generative factors $\boldsymbol{v}$, it may be helpful to obtain such representation, as each unit could be analyzed independently to get useful insights on them. Furthermore, by forcing a smooth transition between classes (i.e., there are no "unused holes" in the space), the whole latent space nicely represents different projections of realistic objects and can be used to generate new data. The decoder, in fact, can be used as a generative network that, given a random vector $\boldsymbol{h}$, can reconstruct the original corresponding $\boldsymbol{x}$.

There remains only one issue to be solved: by having an intermediate sampling operation, we are breaking the continuity of the model and, therefore, the possibility to compute the encoder's gradient (i.e., the sampling operation is not differentiable). The solution is what it is called *reparametrization trick*. Instead of directly using

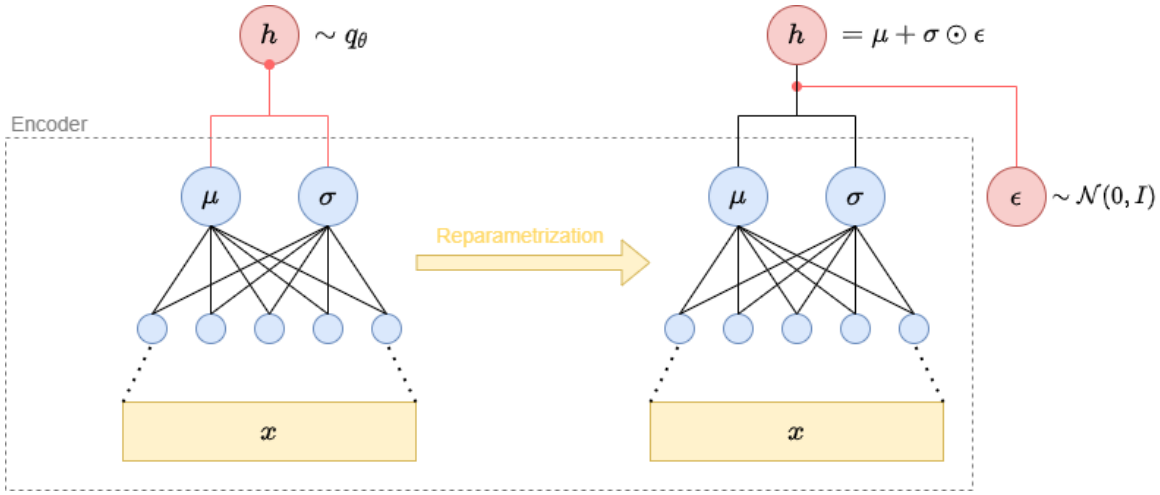

Figure 2.4: Reparametrization trick.

$\mathcal{N}(\boldsymbol{\mu_\theta}, \boldsymbol{\Sigma_\theta})$, we sample some noise $\epsilon = \mathcal{N}(0, I)$, shift it by $\boldsymbol{\mu_\theta}$ and scale it accordingly to $\boldsymbol{\sigma_\theta}$ (Fig. 2.4). In this way, we are able to backpropagate the error across the network, as the randomness is kept outside the model.

2.4.4 Disentanglement: β-VAE

The idea of isolating the ground truth factors $\boldsymbol{v}$ generating a sample $\boldsymbol{x}$ is widely studied in literature. *Disentanglement* refers to the independence among features in the embeddings that allows for an easier interpretation and analysis or, in other words, that the dimensions characterising the latent space encode different features of the data. It is considered a key element for explainability inside neural networks, as one can easily observe and understand the transformations and, therefore, the underlying processes that each input sample undergoes through each cluster of layers. In the ERC setting, any utterance's emotion is assumed to be generated by the Valence, Arousal, Dominance factors. By describing it through them, instead of a discrete category, a perfectly disentangled representation of the sentence, encoding those three factors in separate latent units, would be obtained (Fig. 2.5). It is the consequent explainability, then, that would allow for the merging of different datasets, as it is

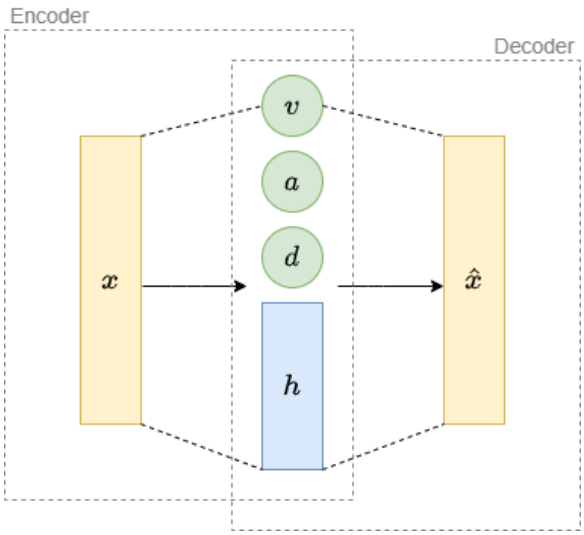

Figure 2.5: Ideally disentangled representation of emotional features in an autoencoder.

clear what each label actually means. The KL divergence term in the VAE's loss function induces disentanglement by forcing the covariance between latent units to be zero. However, as for any regularization method, balancing its weight is required in order to obtain a meaningful representation: if we focus entirely on minimizing the KL divergence, every sample observation would be mapped to the same isotropic unit Gaussian, encoding little to no information. Instead, when the two terms, i.e., reconstruction loss and KL divergence, are optimized simultaneously and balanced correctly, the model is encouraged to describe the latent state with distributions close to the prior but deviating when necessary to express salient features of the input.

β-VAE is a slight variation of the VAE architecture, developed by Higgins et al. [12], that introduce a weight factor, $\beta > 0$, into the ELBO loss function, allowing for a manual balancing between the two terms:

$$ELBO_\beta = l_{rec} - \beta \cdot l_{KL} \tag{2.20}$$

The authors, additionally, define a dataset, called *dSprites* [19], to study and establish

the capabilities of their model. It consists of binary images containing a 2D shape, chosen among three possibilities (i.e., diamond, heart, and ellipse), randomly positioned, scaled, and rotated (Fig. 2.6). Comparing the encoded latent values with these original factors, it is possible to understand the quality of the obtained disentanglement. However, a general and clear definition of disentanglement does not yet exist, nor a way of measuring it, especially in scenarios where the true factors $\boldsymbol{v}$ are not known. Higgins suggested their own metric, which has been proven not to be very stable and improved in following models (e.g., $\beta-$TCVAE, which relies on total correlation and mutual information to measure the disentanglement [7]). In particular, being autoencoders an unsupervised architecture, it is difficult to understand which factors are encoded in which latent unit. A commonly used method is performing a *traversal* for each of the hidden dimensions: by fixing all the latent units except one and generating, through the decoder, different samples corresponding to different values across the chosen dimension, it is possible to infer which basic feature is determined by it (if any at all). However, it is an entirely qualitative method and, due to its subjectivity, it would be better to utilize it only as a confirmation tool, rather than relying entirely on it.

In order to overcome this issue, semi-supervised methods, that depend partially on annotated data, have been developed. Wu et al. [32] proposed a semi-supervised VAE, SRV-SLSTM, that exploits dimensional labelled data, based on the VAD framework, to guide the training phase. By adding a regression network that works in parallel with the encoder, they are able to split the latent vector $\boldsymbol{h}$ in a *content* component, c, learnt by the autoencoder, and a *style* one, s, which is generated by the supervised network. In this way, the model is manually induced to disentangle the VAD values from other factors. However, it is not able to incorporate categorical datasets during the training phase, as it doesn't employ any classification network. Despite this, it shows that it is possible to use semi-supervised methods, combining the versatility of

Figure 2.6: Samples from the dSprites dataset.

autoencoders with the straightforwardness of supervised learning, to obtain state-of-the-art performance.

3 | Methodology and development

3.1 Overview

In order to provide a complete analysis and show the effectiveness of the architecture, I tried to reduce to the minimum the number of external variables. Therefore, instead of focusing on Emotion Recognition, a problem in which numerous factors can determine the validity of the results, I developed a collection of datasets, derived from the dSprites dataset, on which I performed a series of synthetic benchmarks, showing how all the observations made have a parallel and direct application to the ERC problem. Considering, especially, the complexity of the model developed by Sha [30], which employs a multi-target loss function, I tried to analyze each component independently, highlighting its role in producing the final result. Sha's architecture is an enhancement of an encoder-decoder structure, which works by introducing an intermediate sampling operation in the bottleneck layer. I decided to build upon $\beta-$VAE [12], but any similar framework can be experimented with (e.g., $\beta-$TCVAE [7] or Transformer-based models, by relying on the $[CLS]$ token [8]). $\beta-$VAE has the advantage that the disentanglement it produces is determined exclusively by the β factor (and by the choice of the prior, which is, however, always assumed to be the isotropic unit Gaussian). Consequently, by experimenting with different β values in Sha's multi-target loss function, it is possible to understand the impact of the new architecture, compared to the standard $\beta-$VAE.

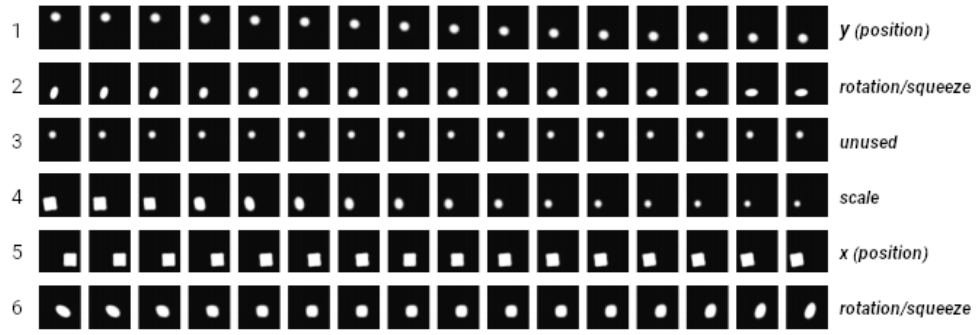

Figure 3.1: dSprites dataset, latent variable traversal. $\beta = 10.0$, $h_{dim} = 6$

As previously mentioned, different values of β determine different levels of disentanglement, with a clear trade-off with the reconstruction loss. In order to show $\beta-$VAE's capability, I trained a model with $\beta = 10.0$ and $h_{dim} = 6$ (i.e., the number of dimensions of the latent representation) on the dSprites dataset. Each image in the dataset is generated based on five generative factors: *shape*, c_x, c_y, *scale*, and *rotation*; by performing a latent traversal, it can be seen how Higgin's architecture is able to isolate all of them (with *rotation* requiring two units probably due to its periodic nature). However, there is no established order in which the factors appear and any permutation can be randomly learnt by the model, representing a valid encoding of the input images. It is the simplicity of the problem that allows for a direct identification of the feature affected by each latent dimension; in a real world scenario, it may not be as straightforward and, instead, entirely subjective. Nevertheless, for the scope of this work, latent traversals are sufficient, as novel metrics will be introduced.

β	**Disentanglement**
0.1	0.511
1	0.663
3	0.747
5	0.763
10	0.974
15	1.000

Table 3.1: Disentanglement score using different values of β, $h_{dim} = 6$.

Finally, by measuring the *disentanglement score* suggested by Higgin, we can report that, in order to obtain perfect disentanglement, high values of β (e.g. $\beta = 10.0$ or $\beta = 15.0$) are required (Table 3.1). The numbers reported in the table represent the accuracy with which a simple classification network is able to identify each disentangled factor: 1.0 means perfect disentanglement, while values below 0.3 basically represent a completely entangled embedding, with the model randomly guessing. However, I will show how, using the proposed architecture, lower values of β can be used to obtain valid results, demonstrating the efficacy of the newly developed method.

3.2 Datasets

The work done within this project can be applied to any domain where there is a gap between categorical annotations and continuous underlying values. It was initially developed to propose a solution to the ERC's data sparsity problem. However, in order to focus entirely on the properties of the developed architecture, I opted to study a less complex scenario, based on a widely known problem setting.

dSprites is a dataset composed by $N = 737280$ square images of 64 by 64 pixels [19]. As mentioned previously, each image is determined by five factors, that can assume different values:

1. **shape**: square, ellipse, or heart;
2. **scale**: 6 values in [0.5, 1];
3. **rotation**: 40 values in $[0, 2\pi)$;
4. $\boldsymbol{c_x}$: 32 values in [0, 1], corresponding to 32 positions in the range (0, 64) pixels;
5. $\boldsymbol{c_y}$: 32 values in [0, 1], corresponding to 32 positions in the range (0, 64) pixels.

Each item in the dataset is therefore a labelled tuple (*image, factors*), making it easy to retrieve, given a picture, its generative factors, in order to compare them with the disentangled values obtained by a model. However, since we are interested into working with categorical annotations, I derived three main datasets from the original dSprites one, in which I replaced some of the real value factors, in particular c_x and c_y, with discrete categories. I, then, defined three different categorical systems and labelled each data point according to them. We have, therefore, the following datasets:

- **dSprites+cAP** (categories with Average Precision): c_x and c_y are labelled according to the ranges [0, 23], [24, 39], [40, 63], respectively *left-ap, center-ap, right-ap* and *top-ap, middle-ap, bottom-ap*. We have three categories on each dimension that represent a wide range of values;

- **dSprites+cHP** (categories with High Precision): c_x and c_y are labelled according to the ranges [0, 10], [11, 25], [26, 37], [38, 52], [53, 63], respectively *far-left-hp, left-hp, center-hp, right-hp, far-right-hp* and *far-top-hp, top-hp, middle-hp, bottom-hp, far-bottom-hp*. Therefore we have five categories on each dimension that represent narrower range of values;

- **dSprites+cUP** (categories with Unbalanced Precision): c_x and c_y are labelled according to the ranges [0, 9], [10, 19], [20, 27], [28, 41], [42, 63], respectively *far-left-up, left-up, near-left-up, center-up, right-up* and *far-top-up, top-up, near-top-up, middle-up, bottom-up*. Therefore we have five categories on each dimension, but they represent uneven ranges of values.

Slight variations of these datasets will be employed to obtain some of the results; since they are linked to specific experiments, they will be described in their respective sections.

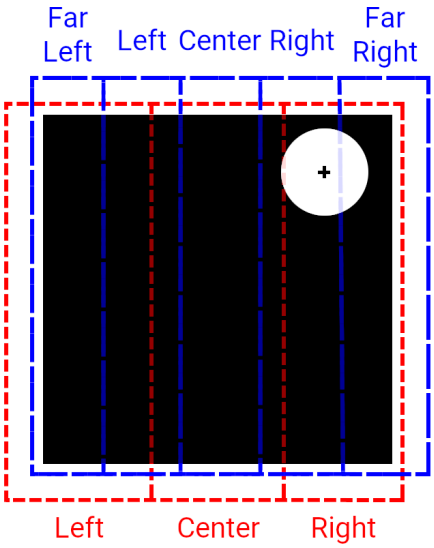

Figure 3.2: Sample image labelled according to two different categorical models: AP and HP.

Each dataset is generated by selecting a shape and sampling a subset of the original dSprites objects of that type (e.g. dSprites+cAP is generated with 80% of the ellipses). Subsequently each element's c_x and c_y coordinates are replaced with the labels describing the ranges they fall in, given the target categorical model. The image shown in figure 3.2, for example, would be labelled as *right* by both *AP* and *HP* systems. In this way, I generate several datasets whose data points belong to the same domain (i.e., black and white pictures each containing a single shape) but are not completely identical, as they have different shapes. Furthermore they are labelled according to different models and, therefore, there is no default way to merge them. However, we know that, although the categories used are not compatible, they are generated by the true generative factors c_x and c_y. This is exactly the problem setting previously defined. Thus, we aim to recover, from the categorical labels, the original c_x and c_y, map them into the same space, and merge all datasets together. Assuming that the distributions underlying each label are Gaussian, we want to compute each mean and standard deviation in order to project them into the latent space. Given the desired smoothness property, distributions of categories that represent objects close

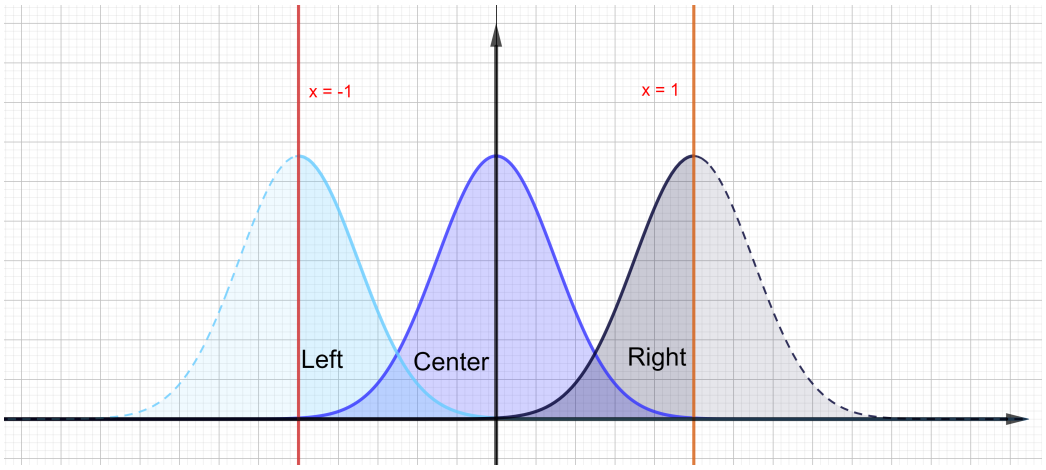

Figure 3.3: Example of successfully displaced labels into the latent space.

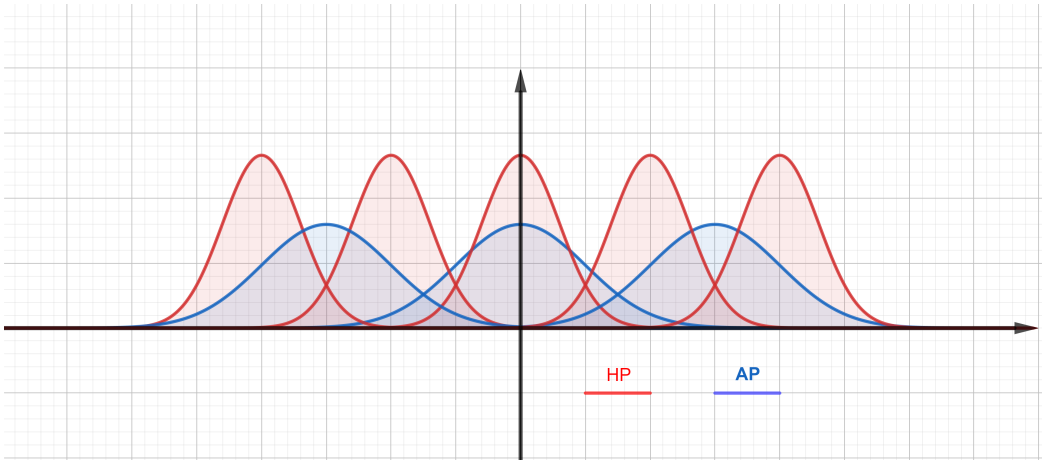

Figure 3.4: AP and HP labels projected into the same continuous space.

to each other in the feature space (e.g., *far-left* and *left*) should be close in the latent space as well, and present a smooth transition from one to another. In figure 3.3, for example, the labels used for the c_x axis in the AP dataset are correctly displaced into the latent space, in this case the interval [-1, 1]. By repeating the process for all the datasets, we obtain a clear representation of each label's meaning in the same latent space. Thus, it is possible to relate and merge them (as they share the same metric space, Fig. 3.4), combining the different datasets into a single one.

Each generated dataset, however, contains samples that, despite having their cen-

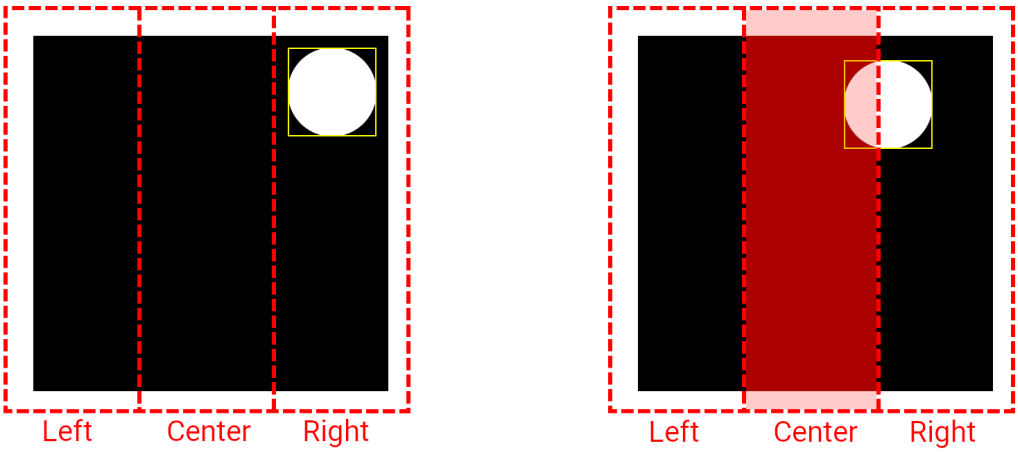

Figure 3.5: Example of shapes completely belonging to one interval (left) and to multiple ones (right).

ter positioned in a specific label range, may belong to multiple ones, if we consider the full size of the shape (Fig. 3.5). A similar situation can happen when dealing with Emotion Recognition, as some utterances may have blurrier emotional nuances, compared to others. As a consequence, due to the subjectivity of the annotation process, mislabelling is a possibility. Luckily, by working in a continuous space with strong structural properties (i.e., ordered distributions and smoothness transitions between them), it is easy to represent uncertainty, by projecting the samples to points where multiple distributions overlap. The framework proposed within this work is able to perform all the required inference autonomously, as it will be shown through experimental observations.

3.3 Architecture

I have now highlighted all the background knowledge and related work that will let us define the method developed to solve the problem of projecting different categorical labels into the same continuous space, in order to compare them and be able to merge together multiple datasets. This would result in an architecture able to represent each category's true generative factors; for ERC, this means decomposing

each emotion in its three components: Valence, Arousal, and Dominance. The ideal solution, therefore, would produce, as a side effect, a model achieving perfect style disentanglement, capable of generating latent vectors that separate the emotional nuances from the content of each utterance. The existing methods propose effective ways to derive the factors $\boldsymbol{v}$, but fail to independently exploit categorical annotations, as dimensional information is always required, either as a priori knowledge to embed into the model, or as training data. Furthermore, they rely on strong assumptions on the latent space's structure. I tried, therefore, to focus on the disentanglement itself, as a means to reach the desired goal, instead of being a secondary result. This allows to develop techniques that are applicable to any field, and do not rely on any premise. Autoencoders, as a general purpose method, seem like a good fit; the problem is how to incorporate exclusively categorical knowledge to disentangle $\boldsymbol{v}$ in the desired way.

Sha and Lukasiewicz recently published an architecture, based on an enhanced VAE, that is capable to achieve targeted *multi-type disentanglement* (Fig. 3.6) [30]; I will refer to it as *MTD-VAE*. They consider each embedding $\boldsymbol{h}$ as a vector that can be split into content c and multiple target style types $s_1, s_2, ..., s_n$, which are *"classes that represents a specific feature of text or an image, e.g., sentiment, tense, or face direction"*. The model, by employing multiple classification networks (one for each style type) applied on the bottleneck layer, is able to link each possible value for each style type, called *style values*, to a specific volume of the latent space, defining a probability distribution over it. Thus, it is capable of projecting categorical annotated points in a continuous dimensional space. However, by introducing the intermediate classification network, which relies on a random sampling operation, the architecture breaks the continuous flow of information between the encoder and the decoder. Consequently, the smoothness property of variational autoencoders is lost and, although each distribution correctly represents its features locally, the latent space lacks global structure: each category is independent from the others and the

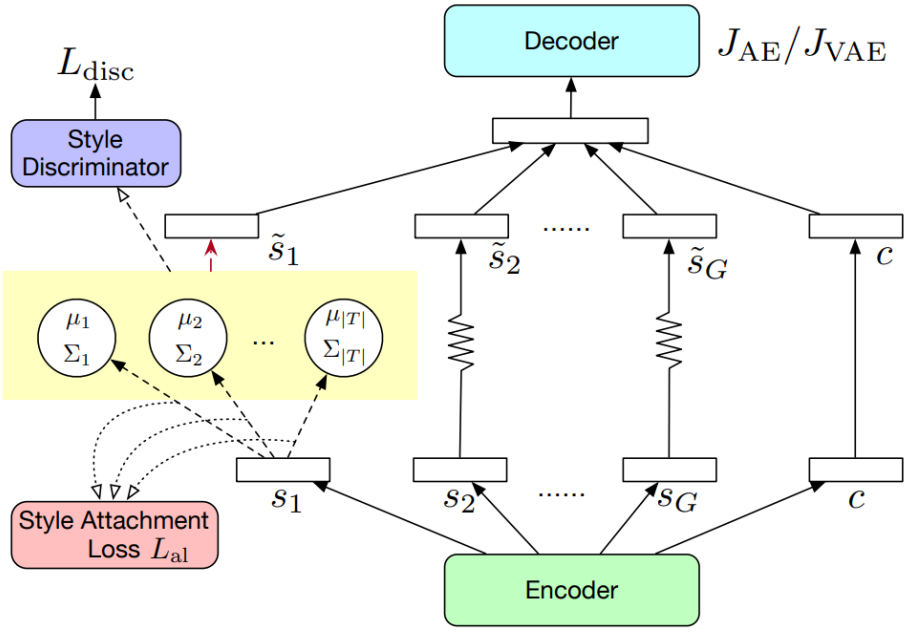

Figure 3.6: Multi-type disentanglement VAE. The yellow box and the zig-zag arrows describe and represent the intermediate sampling process. Taken from "*Multi-type Disentanglement without Adversarial Training*" [30].

model is not able to correctly interpolate intermediate samples (exactly as it happens in the plain autoencoder architecture).

By introducing some architectural changes, I aim to recover this fundamental property: the new method should be able to learn the order and positions of the labels' projections in the latent space. I will now describe in detail the original model and its improved version, used to obtain all the results in this work. I will show that it is possible to obtain targeted disentanglement (i.e., disentangle only a specific subsets of the true generative factors $\boldsymbol{v}$) by augmenting an unsupervised architecture and using exclusively categorically annotated data.

3.3.1 Multi-Type Disentanglement Variational AutoEncoder

As a baseline for the experimental results, the MTD-VAE architecture is used. For the encoder and decoder components, I employed the ones described by Higgins, in the original $\beta-$VAE paper [12]. They consist of a combination of convolutional and fully connected layers; a hidden dimension of 20 has been used for the bottleneck layer across all the reported experiments (however, multiple values has been tested without noticeable differences). The goal of Sha's architecture is to introduce a layer capable of disentangling specific target styles from the content; therefore, the embedding generated by the encoder (after the sampling operation defined by the variational approach) is split into several components: a content vector $\boldsymbol{c}$, which is directly fed into the decoder, and several style vectors, one for each of the n target style types. Each of them, instead, undergoes a further resampling operation, which guarantees the multi-type disentanglement. Consider a single style type vector $\boldsymbol{s_i}$ (generated by the encoder from a sample $(\boldsymbol{x}, y_1, ..., y_n)$), its possible assigned categories (i.e., the labels of the chosen categorical model) need to be projected into the latent space. The model, therefore, defines a multi-variate Gaussian for each of them, representing the distribution of the corresponding generative factors $\boldsymbol{v_i}$, $p(\boldsymbol{v_i}|y_i)$. Instead of directly feeding $\boldsymbol{s_i}$ to the decoder, a new value $\boldsymbol{s'_i}$ is sampled from the distribution representing the label $\boldsymbol{y_i}$. In this way, it is guaranteed that the decoder receives a value representing the correct label and, by backpropagating the reconstruction error of the autoencoder, that the chosen distribution actually encodes the label's features. At the same time, we need to make sure that the style vector $\boldsymbol{s_i}$ appears like sampled from that same distribution or, in other words, that the encoder produces values correctly encoding the object's features, according to the defined style value's probability distributions.

The key idea is to separate each label's projection from the others, in order to have a clearly different encoding for each possible style value. This behaviour is obtained

by means of a classification network attached to the architecture after the sampling operation that forces each distribution away from the others (as this allows to have a better classification accuracy). The approach, combined with other loss function components, generates a disentangled representation, where the content $\boldsymbol{c}$ and each style type $\boldsymbol{s_i}$ are clearly independent from each other. Formally, the model performs the following steps:

1. **Encoding**: the sample point $\boldsymbol{x}$ is processed by the encoder, generating a mean and standard deviation for each style type and content's dimension. Then the latent vector $\hat{\boldsymbol{h}}$ is sampled from obtained multi-variate Gaussian;

2. **Resampling**: $\hat{\boldsymbol{h}}$ is split into $\boldsymbol{c}$ and $\boldsymbol{s_1}$, ..., $\boldsymbol{s_n}$. $\boldsymbol{c}$ is fed to the decoder while each $\boldsymbol{s_i}$ is discarded. Instead a new $\boldsymbol{s'_i}$ is sampled from its style type's correct distribution, according to the style label y_i;

3. **Decoding**: the latent vector $\boldsymbol{h}$, defined by $\boldsymbol{c}$, $\boldsymbol{s'_1}$, ..., $\boldsymbol{s'_n}$, is normally decoded into $\hat{\boldsymbol{x}}$.

The original $\boldsymbol{s_1}$, ..., $\boldsymbol{s_n}$, despite not being used in the reconstruction phase, are necessary to compute the loss function, in order guide the encoder to produce the right values. The loss function is composed by the following components:

- **VAE Loss**: computed from the reconstruction loss (e.g., using the Binary Cross Entropy loss between the pixels of the original and reconstructed image) and the KL divergence according to the canonical formulation. As the priors are all assumed to be Gaussian, it can be easily computed;

- **Multi-type Disentanglement Loss**: computed to enforce each style vector to encode a different style type, by minimizing the mutual information between

each other. Sha proved that this is equal to utilizing the following loss function:

$$L_m = \sum_{i}^{n} \sum_{j,j\neq i}^{n} [\mathcal{H}(p(t_i|\boldsymbol{s_i})) - \mathcal{H}(p(t_j|\boldsymbol{s_i})) - \mathcal{H}(p(t_j|\boldsymbol{s_i'}))] \tag{3.1}$$

where $\mathcal{H}$ represents the entropy function.

and for each style type $i \in \{1, ..., n\}$:

- **Style Attachment Loss**: computed to make the encoded value appear as sampled from the correct distribution. Given a style vector $\boldsymbol{s}$ and its correct style value t, we want to maximize the probability of $\boldsymbol{s}$ belonging to the distribution of t, $p(t|\boldsymbol{s})$. Assuming that all probability distributions are Gaussian, we have by Bayes' rule:

$$p(t|\boldsymbol{s}) = \frac{\mathcal{N}(\boldsymbol{s}|\boldsymbol{\mu}_t, \Sigma_t)p(t)}{p(\boldsymbol{s})} = \frac{\mathcal{N}(\boldsymbol{s}|\boldsymbol{\mu}_t, \Sigma_t)p(t)}{\sum_{t'}(\mathcal{N}(\boldsymbol{s}|\boldsymbol{\mu}_{t'}, \Sigma_{t'})p(t'))} \tag{3.2}$$

  from which we can compute the Negative Log Likelihood (NLL) loss:

$$L_{al} = -\log(p(t|\boldsymbol{s})) \tag{3.3}$$

- **Style Classification Loss**: required to have each distribution map to the correct corresponding style value. Therefore we want to maximize the following probability:

$$p(t|\boldsymbol{s'}) = \frac{\mathcal{N}(\boldsymbol{s'}|\boldsymbol{\mu}_t, \Sigma_t)}{\sum_{t'}\mathcal{N}(\boldsymbol{s'}|\boldsymbol{\mu}_{t'}, \Sigma_{t'})} \tag{3.4}$$

  and minimize the corresponding NLL loss function:

$$L_{cl} = -\frac{1}{n}\sum_{j=1}^{n}\log(p(t|\boldsymbol{s_j'})) \tag{3.5}$$

- **Style-Content Disentanglement Loss**: used to guarantee that the content

vector does not contain any information about the style. To achieve this, Sha proposes to minimize the mutual information between the content vector and each style vector, before and after the sampling operation, namely $I(\boldsymbol{c}, \boldsymbol{s})$ and $I(\boldsymbol{c}, t)$. To minimize $I(\boldsymbol{c}, t)$, the author shows that

$$I(\boldsymbol{c}, t) \leq \mathbb{E}_x[\sum_{t'} p(t') D_{KL}(p(\boldsymbol{c}|t, \boldsymbol{x})||p(\boldsymbol{x}|t', \boldsymbol{x}))] \tag{3.6}$$

In order for it to be computable, a new distribution $\mathcal{N}_c(\boldsymbol{\mu}'_t, \Sigma'_t)$ is defined to model $p(\boldsymbol{c}|t, \boldsymbol{x})$, and all the content vectors with label t are forced to obey it, defining the NLL loss function

$$L_{pl} = -\log(\mathcal{N}_c(\boldsymbol{c}|\boldsymbol{\mu}'_t, \Sigma'_t)) \tag{3.7}$$

Therefore, the final style-content disentanglement loss function is as follows.

$$L_{sc} = \mathbb{E}_x[\sum_{t'} p(t') D_{KL}(\boldsymbol{c}|t, \boldsymbol{x}||\boldsymbol{x}|t', \boldsymbol{x})] + \lambda_{pl} L_{pl} \tag{3.8}$$

where λ_{pl} is a model's hyper-parameter.

Summing together all these components, each multiplied by its own weighting factor (i.e. model's hyperparameters), we have the final loss function used by the model:

$$L = L_{\beta-VAE} + \lambda_{al}\bar{L}_{al} + \lambda_{cl}\bar{L}_{cl} + \lambda_{sc}\bar{L}_{sc} + \lambda_m L_m \tag{3.9}$$

where $\bar{L}$ represents the sum over all style types of the corresponding loss function.

The model successfully disentangles each style type, separating each style value's distributions in the latent space. It is therefore easy to perform what is called style shift: as the content is unrelated to each style vector, it is possible to programmatically

change only specific features of a sample, by replacing the corresponding style vectors with new values sampled from the new target distribution. The new vector will encode the data required to reconstruct a modified data point through the decoder. However, despite the model ability to encode each different style, there is no information stored about samples presenting intermediate features. This is due to the discontinuity introduced by the resampling step: if, in fact, we assume that the encoder learns to correctly encode them, then sampling a new vector from a single distribution completely deletes any nuances in the embedding. For the MTD-VAE architecture, therefore, both samples in figure 3.5, for example, will be encoded exactly with the same embedding, since the only difference between them is in the c_x coordinate but it's not captured by the coarse categorical model. We want to be able, however, to represent it. This would enable the model to understand the relationships between multiple style values. In this case, for example, learning that *center* and *right* share some features and are close in space, while *left* and *right* are completely unrelated.

3.3.2 Multi-Type Continuous Disentanglement Variational AutoEncoder: MTCD-VAE

The goal of this project is to provide a tool capable of producing a merged dataset using data labelled according to different categorical models. The ability to represent intermediate samples and understand the relationships between different categories is, therefore, fundamental. In this section, I illustrate the final proposed architecture that, through multiple improvements, overcomes the issues of the original model. Chapter 4 will illustrate the obtained results. Furthermore, in Chapter 5, novel behaviours and further interesting results are presented. They are beyond the original scope of this work, but could represent a relevant advance towards better AI explainability.

The global MTCD-VAE structure remains unchanged: it consists of an encoder, a decoder, and a bottleneck component, in which a resampling operation happens for each of the style vectors. However, the way it is performed is different. We will now analyze a single-type scenario to facilitate the explanation; however, the architecture is designed to handle multi-type disentanglement by applying the following processes to each style vector $\boldsymbol{s_i}$.

Consider a sample $(\boldsymbol{x}, y)$ (where y is the style value annotated label) and its generated embedding $\boldsymbol{h}$, split into content vector $\boldsymbol{c}$ and a single style vector $\boldsymbol{s}$, whose possible m style values are $\{t_1, t_2, ..., t_m\}$. As described above, a probability distribution $\mathcal{N}_i$ is associated to each of them. In the MTD-VAE architecture, $\boldsymbol{s}$ is entirely replaced by a new vector $\boldsymbol{s'}$ sampled from $\mathcal{N}_{t_y}$. However, this discards any extra information (i.e., detailed values other than the discrete category) stored in $\boldsymbol{s}$. In the new architecture, therefore, $\boldsymbol{s'}_{MTCD}$ is generated in a way that preserves it: a vector $\hat{\boldsymbol{s}}_i$ is sampled from each distribution $\mathcal{N}_i$ and their sum, weighted by each probability $p(t_i|\boldsymbol{s})$, is summed to the original $\boldsymbol{s'}_{MTD}$. Formally:

$$\boldsymbol{s'}_{MTCD} = \boldsymbol{s'}_{MTD} + \sum_{i=1}^{m} \hat{\boldsymbol{s}}_i \cdot p(t_i|\boldsymbol{s}) \tag{3.10}$$

In this way, not only I am capturing more of the nuances stored in $\boldsymbol{s}$, as its value is used to compute the importance of each style value and, therefore, can represent intermediate values (by adjusting the probability weights accordingly), but I am also introducing an explicit relationship between each distribution, forcing them to "interact". The original $\boldsymbol{s'}_{MTD}$ is preserved into the sum as it helps the distributions to represent different values by spreading across the latent space, as in the original MTD formulation.

As it will be shown in the result chapter, MTCD-VAE is able to project each style value's distribution in an ordered and continuous manner into the latent space,

allowing to understand the range of true generative factors $\boldsymbol{v}$ linked to each category. Therefore, given a sample $(\boldsymbol{x}, y)$, we can easily compute the probability distribution $p(\boldsymbol{v}|\boldsymbol{x}, y)$ and, as the distributions used by the model are not implicitly stored into the weights of each neuron, but explicitly represented by their means and variances, it is also easily to obtain an approximation for $p(\boldsymbol{v}|y)$. It is therefore possible to achieve the original goal of this project (i.e., merging different datasets), by adding a masking layer, that let the model select which categorical model is being used and, therefore, which family of distributions to use for each style type. In fact, despite each categorical model uses an independent set of distributions, the encoder and decoder remain fixed across all the training steps and, therefore, all the models are projected into the same space, successfully merging together.

The main property of neural networks is their capability to generalize to new unseen inputs. So, it would be interesting for the model to work with completely unlabelled samples and, therefore, to be able to compute $p(\boldsymbol{v}|\boldsymbol{x})$, removing the dependency from the label y. Through multiple experiments, I discovered that it is sufficient to slowly remove the influence of the vector $\boldsymbol{s'}_{MTD}$ while computing $\boldsymbol{s'}_{MTCD}$, as its importance is relevant only until each style value's distribution has not reached a stable state into the latent space. After this happens, the model can safely learn to generalize without the use of a training label. The final formula to compute $\boldsymbol{s'}_{MTCD}$ is therefore

$$\boldsymbol{s'}_{MTCD} = \alpha \cdot \boldsymbol{s'}_{MTD} + \sum_{i=1}^{m} \hat{\boldsymbol{s}}_i \cdot p(t_i|\boldsymbol{s}) \tag{3.11}$$

where α is an hyper-parameter that decreases over time (Fig. 3.7).

Since we are interested into analyzing the relationships between the different categories, once projected into the latent space V, I decided to restrict each style vector $\boldsymbol{s}_i$ to be one dimensional (compared to the original architecture's formulation that allows for multi-dimensional representations). This introduced a clear notion of order

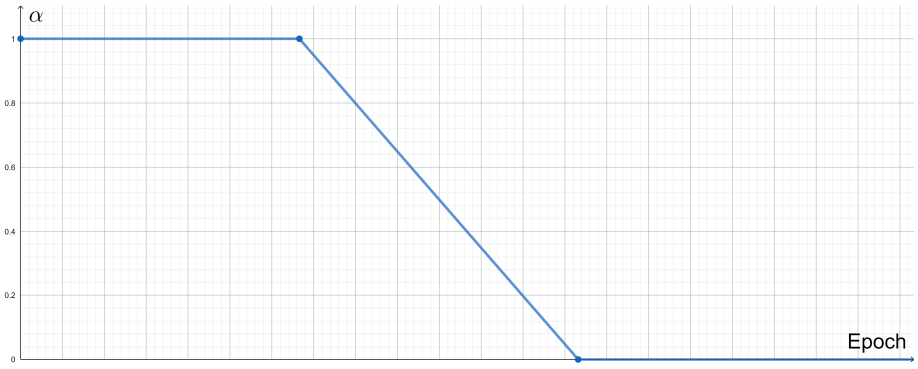

Figure 3.7: Value of α at each epoch.

among the distributions and made the analysis easier. However, it is true that a category could be identified by a specific combination of generative factors $v_{i_1}, ..., v_{i_k}$, and a single dimension may not be able to represent it. Therefore, in order to enable the model to encode all the required information, I allow multiple mono-dimensional style vectors $\boldsymbol{s}_{ij}$, $j \in \{1, ..., k\}$ to be associated with the same style type. Ideally, each one of them will learn to model each factor v_{i_j}. As a consequence, I had to modify the Style Classification Loss function: the classification network needs to be fed with the information from each $\boldsymbol{s}_{ij}$, instead of a single $\boldsymbol{s}_i$. Therefore, I updated equation 3.2 to compute $p(t|s_1, s_2, ..., s_k)$ (remembering that we are omitting the index i). As we will see in Chapter 5, this new configuration allows for even more inference on the structure of the hidden space V, given a categorical model.

The full diagram of the final architecture can be found in Appendix B.

4 | Results

In order to evaluate the performances of the developed model, it is important to clearly understand what is the purpose it has been built for. Even if the architecture is based on an autoencoder, the reconstructed sample $\hat{\boldsymbol{x}}$ is of little to no importance: the focus is on the structure of the generated embedding $\boldsymbol{h}$. As a consequence, I chose to employ the simple β-VAE architecture and not any of its improved versions, like β-TCVAE, that may greatly improve the reconstruction quality. Furthermore, we are not only interested in the value of the latent units of $\boldsymbol{h}$, but also in the parameters of the neural network that define each category's distributions. If, on one hand, they are explicit and not hidden inside the neurons weights and biases, on the other hand, there is no canonical way to measure how "good" they are and, therefore, the performance of the model itself. In fact, we are not looking at any reconstruction loss, classification accuracy, or direct disentanglement metric. Instead, since we want to project a categorical model into a continuous space V, we are interested, as defined in the problem statement, in the labels' consistency. It remains to be defined which metrics are necessary to evaluate it.

The datasets created for the experiments focus on the c_x and c_y coordinates of each shape, replacing their real values with a categorical label. Therefore, despite the model not having access to them during training, we can compare the distributions' parameters to check if they match the original c_x and c_y ranges for each category. Since the family chosen to approximate the true distributions $p(\boldsymbol{v}|\boldsymbol{x}, \boldsymbol{y})$ (where $\boldsymbol{y}$

is a set of categorical labels, one for each style type, given to a sample $\boldsymbol{x}$) is $\mathcal{N}$ (i.e., Gaussian distributions), it is straightforward. For each generative factor v_i, we can, in fact, consider its average value for the samples under each label's value and compare it with its corresponding distribution's mean. In particular, I will measure the correlation between each label's associated average value in the continuous space and its distribution's mean. In this way, I am able to evaluate how each category is projected into V and if their order is consistent with the true generative factors' order. Consider, for example, the dataset dSprites+cAP and the c_x coordinate. Each sample can be classified as one of the three categories *left-ap*, *center-ap*, *right-ap*, whose intervals are, respectively, [0, 23], [24, 39], and [40, 63] pixels. Then, the average values for each interval are 11.5, 31.5, and 51.5. Therefore, I will measure the correlation between the ordered vector $(11.5, 31.5, 51.5)$ and the ordered vector $(\mu_{left-ap}, \mu_{center-ap}, \mu_{right-ap})$. The closer the value is to 1, the better the projection in the latent space is.

Furthermore, since we are interested into obtaining a smooth transition from one category to another, I want to be sure that the distribution are not isolated in the space but, instead, overlap nicely to enable the model to represent intermediate features. Since I am working with one-dimensional distributions, it can be done through visual inspection. However, no common software was found that allows for the visualization of model parameters as probability distributions, especially when dealing with many of them in the same space (and therefore graph). Consequently a tool has been developed to show, in real time, all the loss values used by the model (as done by software like Google's TensorBoard[1]) and, in particular, visualize each category's distributions. This software is publicly available and will be further improved with explainability of neural networks as core concept in mind. An example of the board can be seen in figure 4.1.

[1]https://github.com/tensorflow/tensorboard

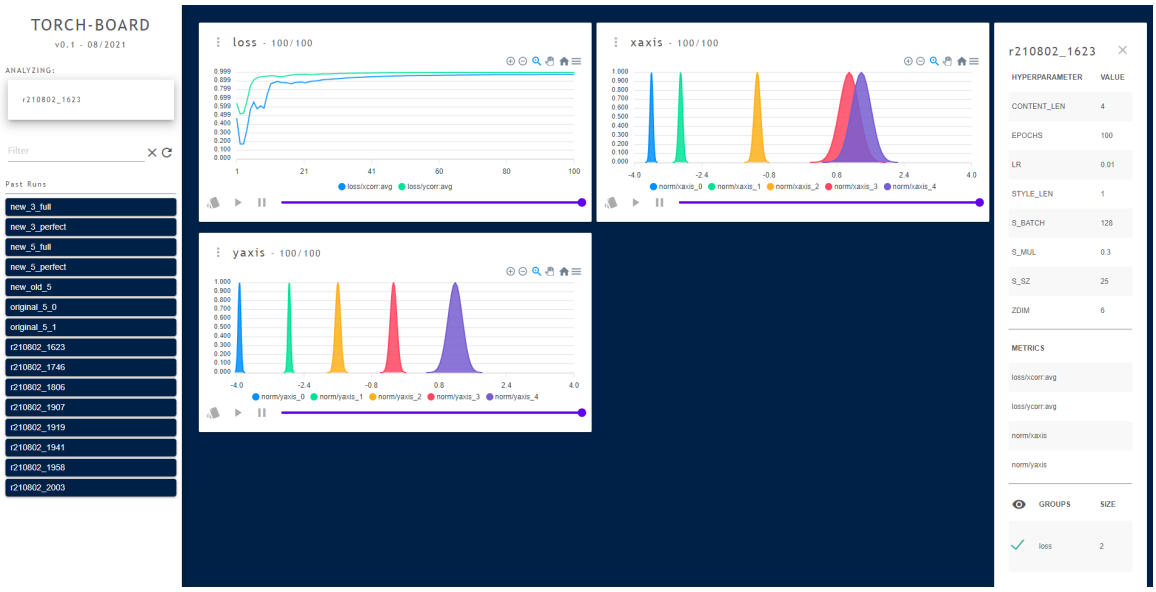

Figure 4.1: Screenshot of the developed visualization software. *xaxis* and *yaxis* graphs report the distributions of each corresponding category.

Other analysis tools and metrics will be explained in relation to their corresponding experiments. They will mostly consist of the usual performance measures used within an autoencoder with disentanglement framework, such as latent traversals.

4.1 Projection of categorical data to a dimensional model

The main result we are interested in is the projection of a categorical label to the continuous space identified by a one-dimensional style vector $\boldsymbol{s}_1$. I, firstly, present the result obtained using the standard MTD-VAE architecture and then compare it with the new MTCD-VAE. I focus on a single style type scenario, where I try to project the labels classifying the c_x value in the dSprites+cAP dataset: *left*, *center*, *right*. As we can see in figure 4.2, the original model MTD-VAE performs optimally, reaching a correlation of almost 1.0. However, as the model treats each distribution

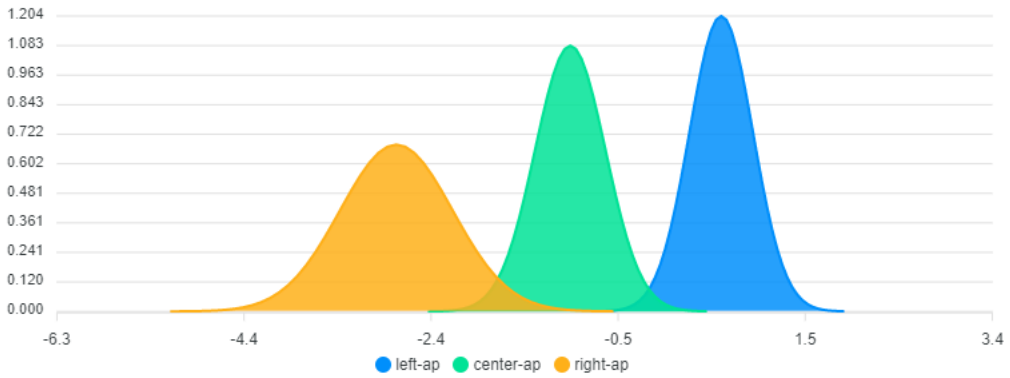

Figure 4.2: MTD-VAE: distributions corresponding to the *left-ap*, *center-ap*, *right-ap* labels. The x-axis represents the continuous value in the latent space V

independently, we can observe that there is no considerable overlapping and the resulting space does not allow for smooth transition from one style value to another. The encoded values for the c_x are all clustered around the three means -2.8, -1.0, and 0.6. Nevertheless, MTD-VAE already produces satisfactory results, proving that the unsupervised learning approach can enhance the data provided to the supervised classification network, establishing some kind of order relationship between the distributions. It has to be noticed, however, that we are working with a relatively simple scenario, with only three possible categories. Given the random initialization of each of the model's parameters, it is with high probability that it can start the learning phase from a favorable condition. In fact, we are measuring the correlation between the labels' true average and the model distributions' means; any affine transformation of the original factors' space is, therefore, valid and would result in a perfect correlation of 1.0. So, it is possible that the distributions in the reported graphs appear in reversed order, have a different scale, or are translated compared to the true generative values.

If we consider the dSprites+cHP dataset, instead, it is rather difficult that a random initialization of the distributions' means correspond to an affine transformation

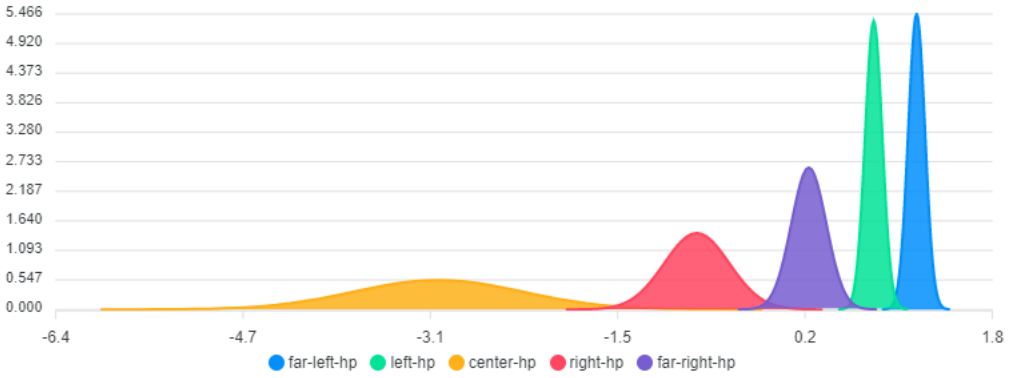

Figure 4.3: MTD-VAE: distributions corresponding to the labels in the dSprites+cHP dataset. They do not appear in the correct order, nor have a consistent standard deviation.

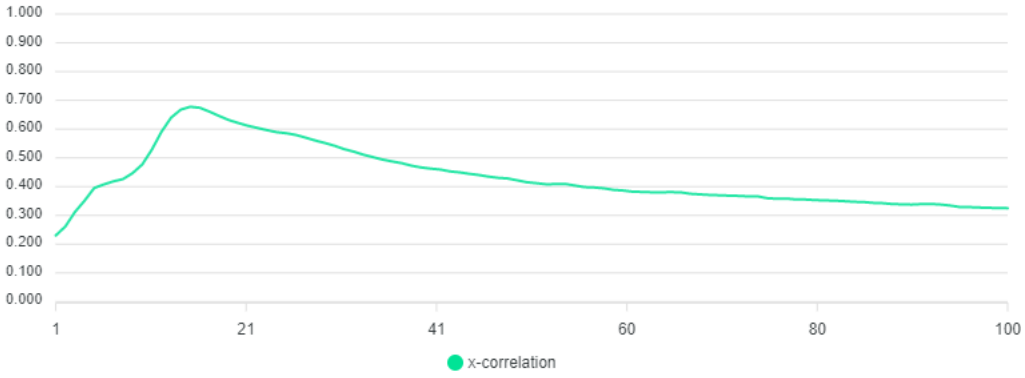

Figure 4.4: MTD-VAE: correlation between the distributions' means and labels' true averages over the training epochs.

of the labels' true average values (or to a state in which it is immediate to reach that configuration). As a consequence, we can observe that MTD-VAE fails to correctly project the categories (Fig. 4.3). Despite being able to separate each style value from the others, disentangling c_x from the other factors in the embedding $\boldsymbol{h}$, it lacks any property of order or smoothness (i.e., the distributions are not in the correct order and do not overlap) and manage to achieve a correlation of only 0.678 (Fig. 4.4). Furthermore, we can notice that this result is not even stable, as the value decreases after having peaked during the first training epochs. From this, we can deduce that the model is not actively optimizing for this objective and, instead, is only a side effect of the disentanglement task.

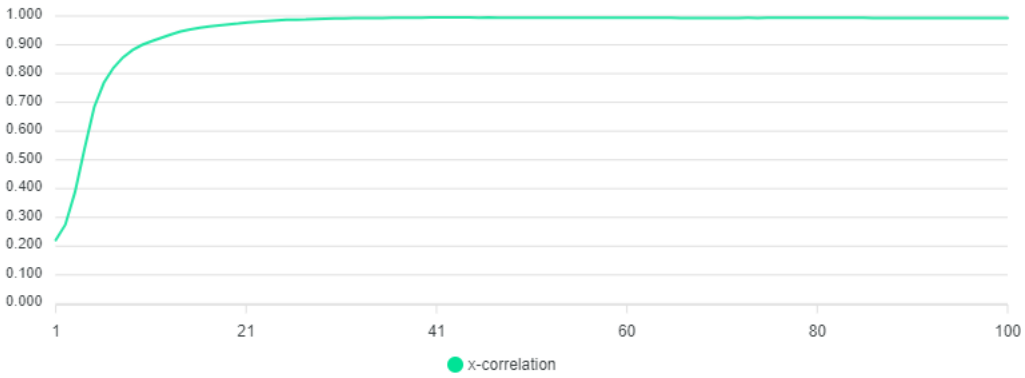

Figure 4.5: MTCD-VAE: correlation between the distributions' means and labels' true averages over the training epochs.

MTCD-VAE is built around the idea of obtaining perfect correlation and, as theoretically hypothesized, it is able to obtain perfect results with each of the three used datasets, always reaching a stable correlation of almost 1.0 (Fig. 4.5). Having run multiple simulations, I can confirm that the result is not due to a favorable random initialization of the distributions' parameters. Instead, thanks to the online and interactive nature of the developed visualization tool, it was possible to observe the model learning and sorting the labels into a perfectly correlated sequence, starting

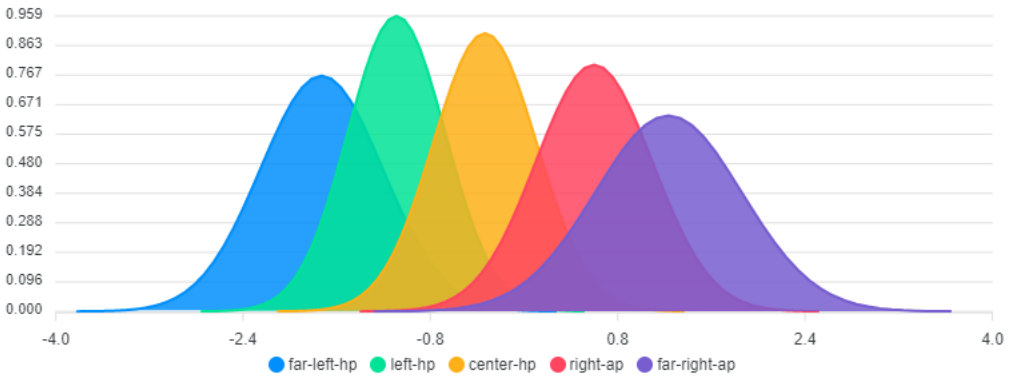

Figure 4.6: MTCD-VAE: distributions corresponding to the labels in the dSprites+cHP dataset.

Label	**Distribution range**	**Space %**	**Label range**	**True %**
far-left-hp	$[-2.4, -1.45]$	21.6%	$[0, 10]$	17.2%
left-hp	$[-1.45, -0.72]$	16.6%	$[11, 25]$	23.4%
center-hp	$[-0.72, 0.11]$	18.9%	$[26, 37]$	18.8%
right-hp	$[0.11, 0.98]$	19.8%	$[38, 52]$	23.4%
far-right-hp	$[0.98, 2.0]$	23.3%	$[53, 63]$	17.2%

Table 4.1: MTCD-VAE: ranges of value associated to each distribution and label (the latent space is clipped according to the distributions' standard deviation.)

from a totally wrong and mixed initial order. Therefore, the model is able to consistently produce the desired result. In figure 4.6, for example, we can observe that the sequence of distributions is correctly ordered according to the associated range of values of their corresponding label, despite the model never having access to that information during the training phase. There is also a clear smooth transition from one distribution to another, allowing to represent nuances of each style value. Furthermore, if we consider the maximum likelihood estimate to assign to each continuous value a label, we have a clear correspondence between the ranges defined by the distributions and the true labels' ranges (Tab. 4.1). Analogous results were obtained on the dSprites+cAP and dSprites+cUP datasets.

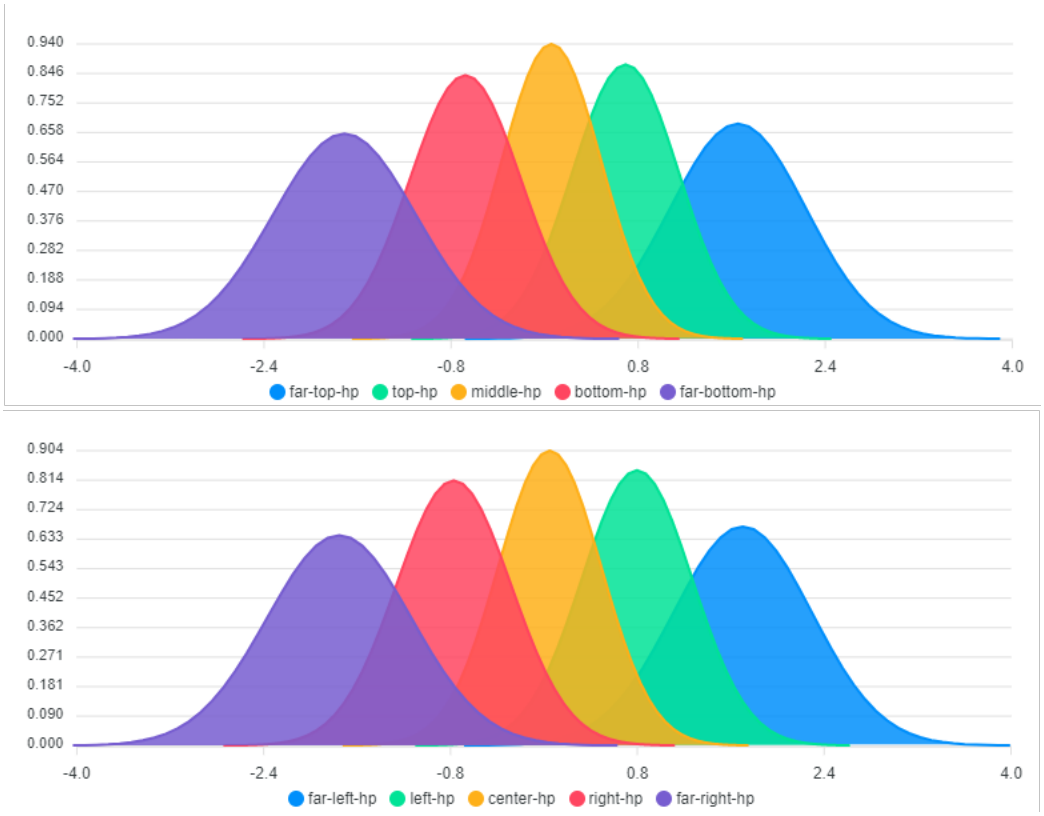

Figure 4.7: MTCD-VAE: multi-type disentanglement and simultaneous categories projection on both the x (bottom) and y (top) axis.

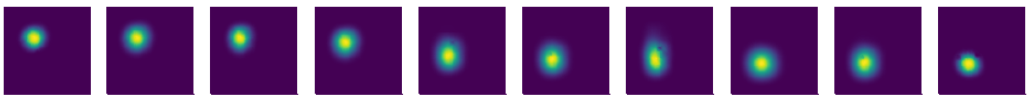

Figure 4.8: Latent traversal on the c_y unit (dSprites+cAP dataset).

4.1.1 The multi-type case

Since the model was originally designed to handle multi-type disentanglement, I tried to project both c_x and c_y labels. The result are analogous: we can clearly see the correct disposition of each distribution, according to its associated label's range of values, and obtain a correlation of 1.0 on both the used dimensions (Fig. 4.7). The model is, therefore, able to perform simultaneously both disentanglement of multiple style types and continuous projection of categorical models into a dimensional one. However, it may be worth noticing that, until now, I have been working with one-dimensional labels (i.e., labels whose value is determined by a single generative factor v) and I have embedded this a priori knowledge inside our architecture, by using a single one-dimensional style vector to encode each style type. Therefore, what it has been shown is that the model is able to correctly disentangle the generative factors and link them to their, *already disentangled*, categorical models. In other words, the model is able to learn to associate single generative factors to their label, but needs to know that the labels require exactly a single dimension to be represented. As we will see in further sections, this is not a strict requirement.

4.1.2 Further analysis

In order to further analyze the stability of the model, I performed some of the tests normally used to assert the performance of autoencoders. However, considering the results obtained already, we are capable of computing $p(\boldsymbol{v}|y)$ (i.e., the generative factors associated to each label y) by simply extracting from the model the corresponding distribution's parameters. Nevertheless, we may still be interested into automatically

computing the fine-grained $\boldsymbol{v}$ value for each sample $(\boldsymbol{x}, y)$, $p(\boldsymbol{v}|\boldsymbol{x}, y)$. Therefore, we analyzed the latent traversals on the c_x and c_y latent units. As we can see from figure 4.8 the latent unit is correctly representing the c_y coordinate. However, there is not a high diversity in the produced images and it is possible to identify the three categorical clusters used in the dSprites+cAP dataset. This is due to the fact that the model is still relying heavily on the input sample's label to generate the output. As it will be shown, by removing the dependency it is possible to achieve a smoother transition from one category to the next. Nevertheless, this shows that the model is able to compute $p(\boldsymbol{v}|\boldsymbol{x}, y)$ (in the example, in particular, $p(c_x, |\boldsymbol{x}, y)$).

Furthermore, if we focus on the accuracy of the classification network attached to the style vectors, we can observe that it is straightforward to fine-tune the model to obtain values close to 100% as the distributions are easily separable as shown above, trading-off with the reconstruction loss. The trade-off is due to the *intermediate* samples that do not clearly belong to a specific category: if I prioritize the classification loss, it is easier to correctly classify them, since the transition area between distributions is small, as they are more clearly separated into the latent space. On the other hand, if we are interested in a better reconstruction loss, I need to preserve the specific features of each sample and, therefore, allow for a greater overlapping between the distributions (to guarantee smoother transitions), which causes a degradation in the classification performance (Fig. 4.9).

4.2 Merging categorical models

Given all the previous results, it is now possible to merge different categorical models, $C_1, ...C_N$ in the same dimensional one. In order to achieve this, it is necessary to train a MTCD-VAE model with samples belonging to each of the $D_1, ..., D_N$ datasets. The architecture automatically switches to the correct set of labels (and therefore

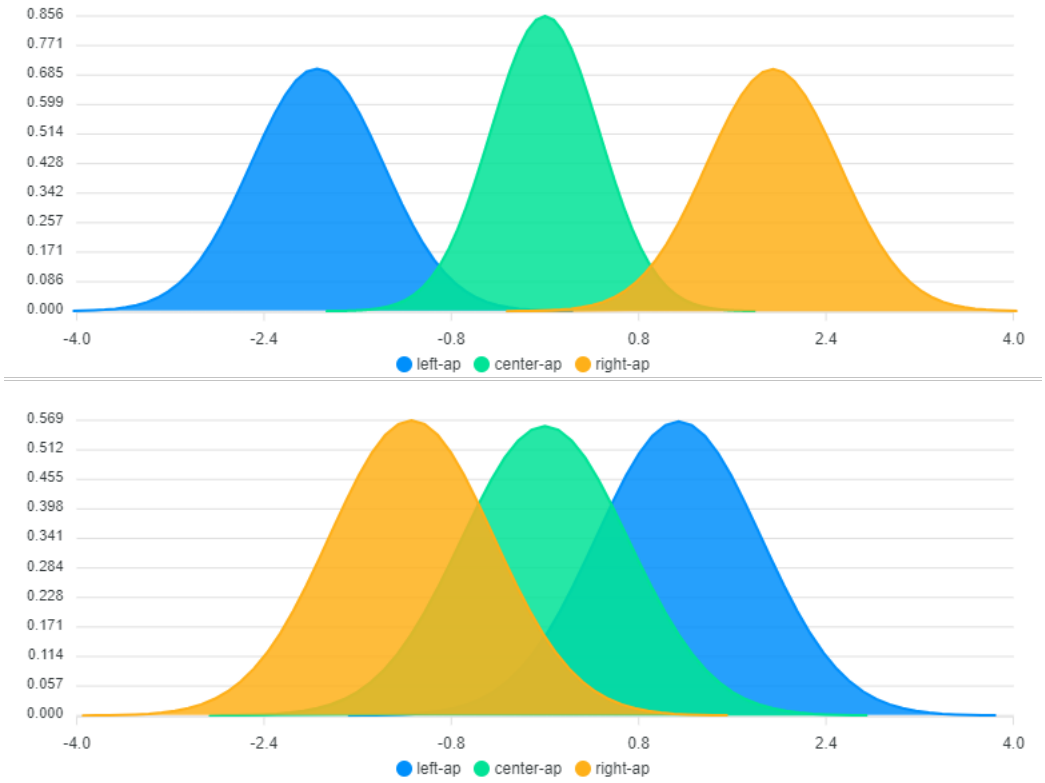

Figure 4.9: MTCD-VAE: comparison between latent spaces when prioritizing classification (top) or reconstruction (bottom) loss.

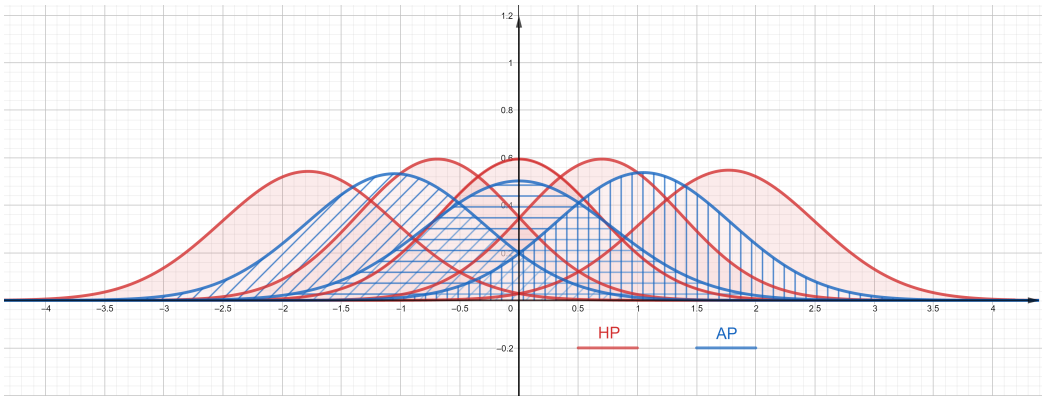

Figure 4.10: MTCD-VAE: dSprites+cAP and dSprites+cHP merged together. In red the ordered labels *far-left-hp*, *left-hp*, *center-hp*, *right-hp*, *far-right-hp*; in blue *left-ap*, *center-ap*, *right-ap*.

distributions) to use for each of the training samples by means of the masking layer. However, since the encoder and decoder are shared among the categorical models, these are projected in the same vector space V and, consequently, share the same representation of the generative factors $\boldsymbol{v}$. Using the embedding's values c_x and c_y obtained from each sample $(\boldsymbol{x}, y) \in \bigcup_{i \in \{1,\dots,N\}} D_i$, I can generate a single dataset D, employing a single unified encoding and consistent with all the previous models. An example of the result displacement of the distributions in the space V among one dimension can be seen in figure 4.10, where the datasets dSprites+cAP and dSprites+cHP were used (in order to emphasize the different categorical models, a different visualization software was employed). As expected, the labels are correctly sorted as *far-left-hp*, *left-ap*, *left-hp*, *center-hp*/*center-ap*, *right-hp*, *right-ap*, and *far-right-hp*; the two *center* labels are completely overlapping, as they represent the same data in both datasets. Furthermore, on each side (i.e., *left* and *right*) the *ap* distribution is between the two *hp* ones and covers a wider range (i.e., has an higher standard deviation), reflecting the wider *ap* label's range and their in-between mean.

5 | Further Results

5.1 Continuous disentanglement

In the previous chapter, I showed how MTCD-VAE is able to correctly approximate $p(\boldsymbol{v}|t)$ and $p(\boldsymbol{v}|\boldsymbol{x}, y)$, where t is a style value, $\boldsymbol{x}$ a sample and y the label associated to it. However, as mentioned earlier, in order to have a model able to generalize to new unseen data-points, it is required to remove the dependency on the label y and, therefore, to be able to compute $p(\boldsymbol{v}|\boldsymbol{x})$. By gradually reducing the value of α over the training epochs and balancing the different loss weights $\lambda_.$, the model is able to produce a latent representation $\boldsymbol{h}$ that depends entirely on $\boldsymbol{x}$ and on the previously learned distributions associated to each style type and value. As it can be observed in figure 5.2, the distributions' displacement remains unchanged and the correlation stable over time 5.1.

Furthermore, I now have full control over the importance of the categorical information, by adjusting the weight of the classification loss, λ_{cl}. When reducing it, there is no learning pressure to create clusters of data-points to maximize the classification accuracy and instead, the embeddings can spread evenly over all the latent space V, obtaining a smooth transition between the categories and enabling the encoder to produce the true generative factors' values, c_x and c_y. By analyzing the values directly taken from the latent $\boldsymbol{h}$, we can indeed observe this behaviour: figure 5.3 shows a smooth curve generated from the encoded value c_x, given a sequence of $\boldsymbol{x}$

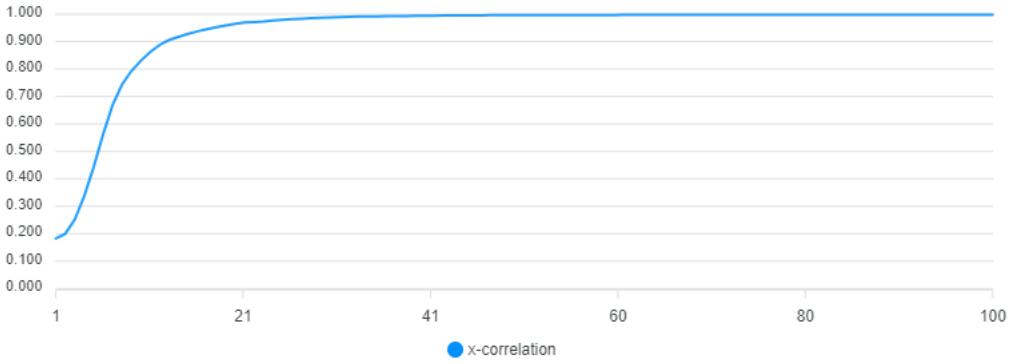

Figure 5.1: MTCD-VAE: final distributions' placement when training for label independence.

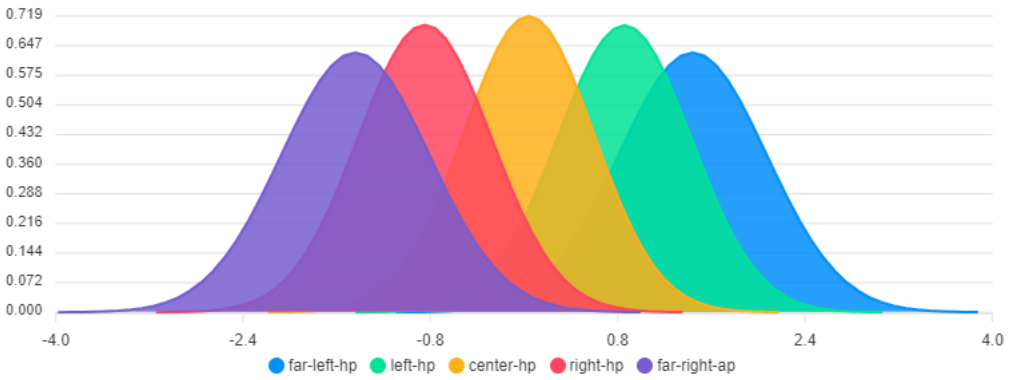

Figure 5.2: MTCD-VAE: correlation between labels and distributions' means when training for label independence.

samples ordered along the x coordinate. The peculiar shape is given by the choice of the encoder's activation function, *tanh*, that, unfortunately, squeezes all the sample points, whose shape is close to the image border, to the same value, loosing accuracy. In all the reported graphs, therefore, the displayed interval goes from 25% to 75% of the ordered available coordinate values.

By applying the inverse transformation, i.e., the *atanh* function, and comparing the result with the true c_x values, we get, after normalization, an almost perfect correlation between the two sequences of points (Fig. 5.4). This shows that the model is completely capable of disentangling the coordinate values of each data-point into the target latent units, becoming independent from the original labels and, rather, providing a greater encoding/reconstruction accuracy that the one provided by exclusively relying on a categorical model. Therefore, instead of disentangling a feature into fixed unrelated categories, it is possible to obtain latent units that encode independent and continuous sequences of values.

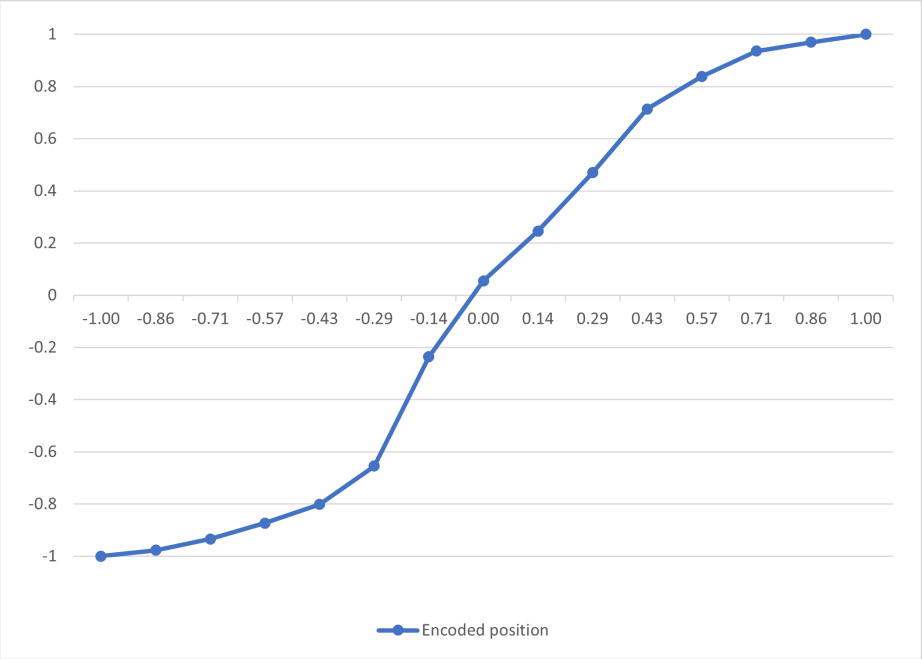

Figure 5.3: MTCD-VAE: c_x coordinates of a sequence of ordered data-points, obtained from the corresponding latent unit in the embedding $\boldsymbol{h}$

Figure 5.4: MTCD-VAE: Transformed and normalized sequence of the encoded c_x shows almost perfect correlation with the true original c_x value.

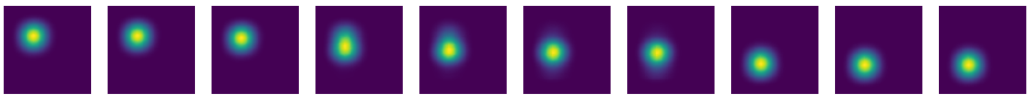

Figure 5.5: Latent traversal on the c_y unit with label independence.

Through a latent traversal, finally, it can be clearly seen how the information representable by the model goes well beyond the three categories available for each style type in the dSprites+cAP dataset, used to train the model. In fact, each latent value corresponds to a unique decoded image, generating a smooth vertical transition (Fig. 5.5). Thus, the model is able to represent the fine-grained $p(\boldsymbol{v}|\boldsymbol{x})$ values.

5.2 Multi-type label disentanglement

All the results presented until now have been produced by using one-dimensional labels, that is, labels that require a single real value to be represented (e.g., the set of

labels {*left-ap*, *center-ap*, *right-ap*} only requires the value c_x to be defined), and we have embedded this a priori knowledge into the model, by utilizing and observing only a single dimension for each style type. However, most of the information collected from the real world is not in the form of already disentangled categorical values and, actually, is quite a challenge to even discover how many does a category encode. In ERC, for example, it is theorized that each emotion can be represented by three values (i.e., Valence, Arousal, and Dominance), and, normally, this number is used as an hyper-parameter or to define the model, as it happens in the architecture proposed by Park [24]. It would be desirable, however, to have the model itself learn the value, to allow for better flexibility and representation capabilities. In order to test the performance of MTCD-VAE on this new task, I created a new dataset, derived from dSprites+cAP, called dSprites+cAPE(ntangled). Instead of providing a label for each coordinate, a single one was assigned, chosen from the Cartesian product of the original label sets. Therefore, each sample in dSprites+cAPE is labelled accordingly to its c_x and c_y with one of the following labels: *lt-ap*, *ct-ap*, *rt-ap*, *lm-ap*, *cm-ap*, *rm-ap*, *lb-ap*, *cb-ap*, *rb-ap*. As previously, the model has no access to the true values c_x and c_y, nor to any information regarding the labels' underlying ranges.

The model, in order to correctly represent each sample, needs to disentangle each label in its generative factors. This means being able to understand how many dimensions k are necessary to encode a particular categorical model and linking the right $v_{i_1}, ..., v_{i_k}$ to them, among all the generative factors $\boldsymbol{v}$. Figure 5.6 shows how the nine distributions, used to represent the nine possible label values, are correctly grouped and displaced on two dimensions by MTCD-VAE, encoding, the c_x and c_y values, as expected. We can conclude, therefore, that the model is able to perform targeted multi-type disentanglement by employing a categorical labelling system as a guide, identifying the multi-dimensional distributions underlying each label's value. Furthermore, if we try to encode a n-dimensional label with $(n + 1)$ dimensions, the

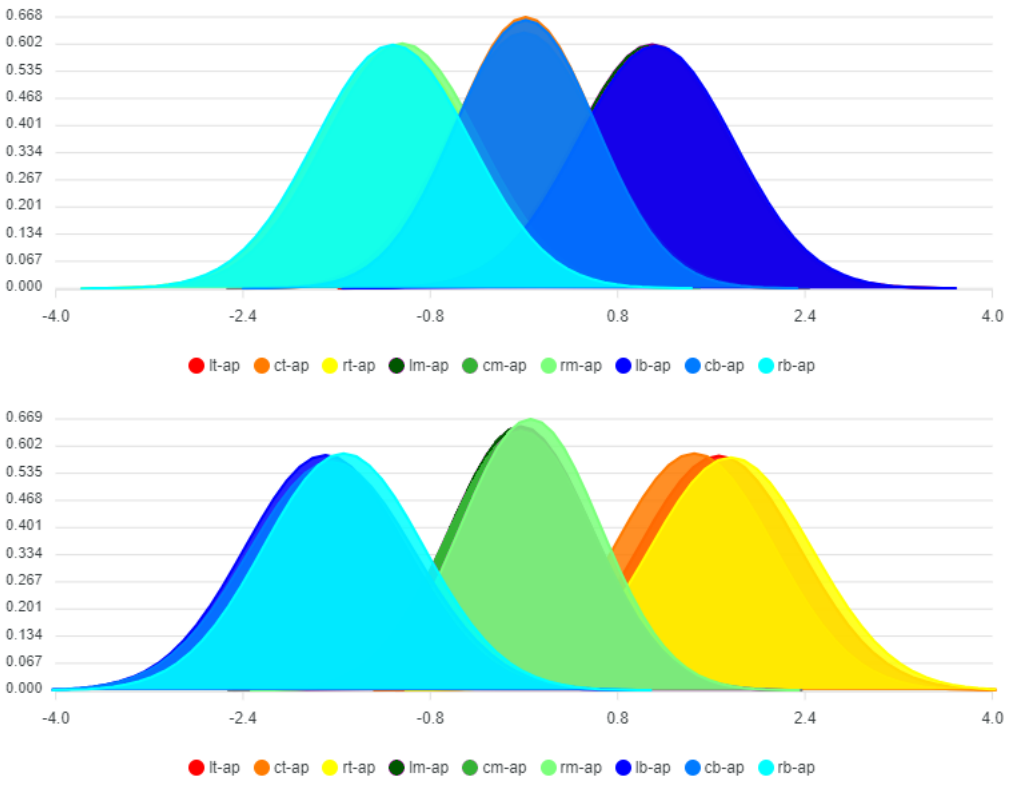

Figure 5.6: MTCD-VAE: disentanglement of a single categorical model among two dimensions; different shades of the same tonality represent categories with the same c_y value, while brightness is shared among categories with the same c_x.

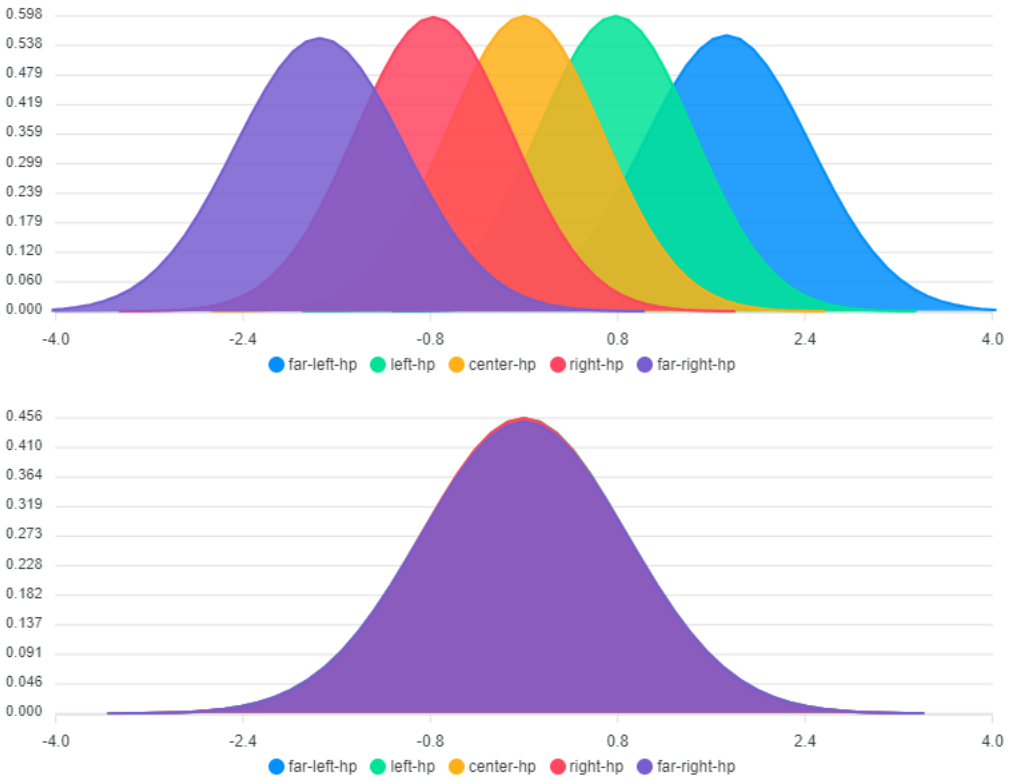

Figure 5.7: MTCD-VAE: trying to disentangle a label within more latent units than necessary results in an unused one.

model is able to recognize the unnecessary style vector, and only utilize n by storing no information in the last one. This can be deduced by the displacement of the distributions in figure 5.7. The model has to disentangle the c_x value and requires only a single dimension to do so; thus, the distributions on other axis do not encode any value, as they are all identical and, therefore, producing only noise. As a consequence, the model is, not only, able to disentangle a label in its true generative factors, but also to discover how many factors are necessary, fully understanding the structure of V.

6 | Conclusions

Multi-Type Continuous Disentanglement Variational AutoEncoder is an architecture able of projecting multiple categorical models into a single dimensional space, by employing the unsupervised learning capabilities of autoencoders combined with the guide provided by coarsely annotated categorical data. I have shown, through the use of synthetic datasets, based on the dSprites one (commonly used to evaluate disentanglement performance), how disentanglement can help not only in obtaining independent latent units, each representing a particular categorical input feature. Instead, when applied to the architecture's weights, it can provide multiple insights on the structure of the domain space we are currently working in. It is possible, in fact, to analyze the explicit distributions' parameters used to represent each category, by observing their displacement in the latent space. The experiments carried out show a clear correspondence between the produced and true real values, with the distributions displaced accordingly to an ordered smooth sequence. The shift from categorical to dimensional models allows us to understand the true meaning of each discrete label, which factors are related to it and how many dimensions are necessary to encode it. As a consequence, it is easier to discover the relationships between different categories and, therefore, between multiple categorical models, used to describe the same phenomenon. Thus, it offers a tool capable of merging datasets created in different times and settings, producing an unified consistent set of training input points. In this way, I hope to provide an inspiration on how to solve the data sparsity

problem that affects some machine learning tasks, such as Emotion Recognition in Conversation.

Furthermore, I consider the experiments and results of Chapter 5 of interesting relevance under the topic of general explainability in neural networks. They show that MTCD-VAE is able to understand what we, humans, are trying to convey and express through categorical models. In particular, it can formalize what each single category represents and according to which principle the data-points are clustered given the labels, by defining the number k of necessary encoding factors and projecting each category into the latent space V, clearly defining its relationships with the others. On the other hand, through the analysis of the explicit distributions, we are able to describe how the model is processing the data. With this first attempt to provide a completely autonomous bridge between classification (based on categorical data, more suitable to be understood by human beings) and regression (based on dimensional data, easy to process for a computer), I hope to contribute to obtain further insights on how neural networks internally work and, therefore, how we can improve them, in a safe and controlled way.

6.1 Future work

The model proposed in this work has only been tested with a single category of data, i.e., images, and on a specific task: disentangling the c_x and c_y coordinates. In order to obtain more analytical results that can confirm the efficacy and performance of MTCD-VAE in solving the defined problem statement, more tests need to be performed on different datasets. Furthermore the architecture still offers wide possibilities for further improvements that may improve the stability and quality of the obtained results. Finally, the produced datasets should be actually used to train a model for specific downstream tasks to assert if the augmented training data actually

improve the performance.

A list of key points could be the following:

- Experiment with different activation functions or methods to regularize the encoder outputs since, as mentioned, using the default *tanh* function results in a lost of accuracy near the extremes of the encoding interval. A possible idea could be using Layer Normalization [2] or ReLU activation functions;
- Modify the network to handle textual data and experiment within the VAE framework. The encoder and decoder need to be modified to work with sequences of words: LSTM layers [13] are probably the best solution, since they are already being used in the original MTC-VAE architecture;
- Explore the results on downstream tasks of models trained with the generated merged datasets;
- The model still requires a complex loss function, with several hyperparameters that need to be balanced, in order to perform correctly. However, due to the explicit nature of the distributions embedded into the model, it should be able to self-assess its own performance and stability (e.g., by observing the distributions' displacement and movements) and, consequently, automatically balance its own hyperparameters to obtain better performance.

In general, this project constitutes a theoretical framework that needs to be adapted and fine-tuned to real-world tasks and scenarios in order to completely assert its capabilities.

Bibliography

[1] Mikel Artetxe, Gorka Labaka, Eneko Agirre, and Kyunghyun Cho. Unsupervised neural machine translation. *CoRR*, abs/1710.11041, 2017.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[3] Sven Buechel and Udo Hahn. EmoBank: Studying the impact of annotation perspective and representation format on dimensional emotion analysis. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 578–585, Valencia, Spain, April 2017. Association for Computational Linguistics.

[4] Carlos Busso, Murtaza Bulut, Chi-Chun Lee, Abe Kazemzadeh, Emily Mower, Samuel Kim, Jeannette N. Chang, Sungbok Lee, and Shrikanth S. Narayanan. IEMOCAP: interactive emotional dyadic motion capture database. *Language Resources and Evaluation*, 42(4):335–359, December 2008.

[5] Ankush Chatterjee, Kedhar Nath Narahari, Meghana Joshi, and Puneet Agrawal. SemEval-2019 task 3: EmoContext contextual emotion detection in text. In *Proceedings of the 13th International Workshop on Semantic Evaluation*, pages 39–48, Minneapolis, Minnesota, USA, June 2019. Association for Computational Linguistics.

[6] Sheng-Yeh Chen, Chao-Chun Hsu, Chuan-Chun Kuo, Ting-Hao K. Huang, and Lun-Wei Ku. Emotionlines: An emotion corpus of multi-party conversations. *CoRR*, abs/1802.08379, 2018.

[7] Tian Qi Chen, Xuechen Li, Roger B. Grosse, and David Duvenaud. Isolating sources of disentanglement in variational autoencoders. *CoRR*, abs/1802.04942, 2018.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[9] P Ekman. Facial expression and emotion. *The American psychologist*, 48(4):384—392, April 1993.

[10] Devamanyu Hazarika, Soujanya Poria, Roger Zimmermann, and Rada Mihalcea. Conversational transfer learning for emotion recognition. *Information Fusion*, 65:1–12, 2021.

[11] Thomas Heseltine, Nick Pears, and Jim Austin. Three-dimensional face recognition using combinations of surface feature map subspace components. *Image and Vision Computing*, 26(3):382–396, 2008. 15th Annual British Machine Vision Conference.

[12] I. Higgins, L. Matthey, A. Pal, Christopher P. Burgess, Xavier Glorot, M. Botvinick, S. Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2017.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[14] Le Hou, Chen-Ping Yu, and Dimitris Samaras. Squared earth mover's distance-based loss for training deep neural networks. *CoRR*, abs/1611.05916, 2016.

[15] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014.

[16] Philipp Koehn and Rebecca Knowles. Six challenges for neural machine translation. *CoRR*, abs/1706.03872, 2017.

[17] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[18] Yanran Li, Hui Su, Xiaoyu Shen, Wenjie Li, Ziqiang Cao, and Shuzi Niu. Dailydialog: A manually labelled multi-turn dialogue dataset. *CoRR*, abs/1710.03957, 2017.

[19] Loic Matthey, Irina Higgins, Demis Hassabis, and Alexander Lerchner. dsprites: Disentanglement testing sprites dataset. https://github.com/deepmind/dsprites-dataset/, 2017.

[20] Gary McKeown, Michel Valstar, Roddy Cowie, Maja Pantic, and Marc Schroder. The semaine database: Annotated multimodal records of emotionally colored conversations between a person and a limited agent. *IEEE Transactions on Affective Computing*, 3(1):5–17, 2012.

[21] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.

[22] Tomás Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[23] Saif Mohammad. Obtaining reliable human ratings of valence, arousal, and dominance for 20,000 English words. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 174–184, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[24] Sungjoon Park, Jiseon Kim, Jaeyeol Jeon, Heeyoung Park, and Alice Oh. Toward dimensional emotion detection from categorical emotion annotations. *CoRR*, abs/1911.02499, 2019.

[25] M Popel, M Tomkova, J Tomek, Ł Kaiser, J Uszkoreit, O Bojar, and Z Žabokrtský. Transforming machine translation: a deep learning system reaches news translation quality comparable to human professionals. *Nature Communications*, 11(1), 2020.

[26] Soujanya Poria, Navonil Majumder, Rada Mihalcea, and Eduard H. Hovy. Emotion recognition in conversation: Research challenges, datasets, and recent advances. *CoRR*, abs/1905.02947, 2019.

[27] Alec Radford, Jeff Wu, R. Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[28] David E. Rumelhart and James L. McClelland. *Learning Internal Representations by Error Propagation*, pages 318–362. 1987.

[29] James A Russell and Albert Mehrabian. Evidence for a three-factor theory of emotions. *Journal of Research in Personality*, 11(3):273–294, 1977.

[30] Lei Sha and Thomas Lukasiewicz. Multi-type disentanglement without adversarial training. *CoRR*, abs/2012.08883, 2020.

[31] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–, October 2017.

[32] Chuhan Wu, Fangzhao Wu, Sixing Wu, Zhigang Yuan, Junxin Liu, and Yongfeng Huang. Semi-supervised dimensional sentiment analysis with variational autoencoder. *Knowledge-Based Systems*, 165:30–39, 2019.

Appendices

A Derivation of the ELBO loss function

Given a sample $\boldsymbol{x}$, we aim to approximate the posterior $p(\boldsymbol{h}|\boldsymbol{x})$ of its latent representation $\boldsymbol{h}$. To achieve this, we use a parametrised distribution q_θ, modeled through a neural network with parameters θ. Then we aim to maximize the similarity between p and q_θ; a commonly used way is to minimize the KL divergence between them. Through a series of equations is easy to show that it is equivalent to maximizing the ELBO.

$$\begin{aligned} D_{KL}(q_\theta(\boldsymbol{h}|\boldsymbol{x})||p(\boldsymbol{h}|\boldsymbol{x})) &= \mathcal{E}_{\boldsymbol{z}\sim q_\theta}[\log_{q_\theta}(\boldsymbol{h}|\boldsymbol{x}) - \log p(\boldsymbol{h}|\boldsymbol{x})] \\ &= \mathcal{E}_{\boldsymbol{z}\sim q_\theta}[\log_{q_\theta}(\boldsymbol{h}|\boldsymbol{x}) - \log p(\boldsymbol{x}|\boldsymbol{h}) - \log p(\boldsymbol{h}) + \log p(\boldsymbol{x})] \\ &= -(\mathcal{E}_{\boldsymbol{z}\sim q_\theta}[\log p(\boldsymbol{x}|\boldsymbol{h})] - D_{KL}(q_\theta(\boldsymbol{h}|\boldsymbol{x})||p(\boldsymbol{h}))) + \log p(x) \end{aligned}$$

Since $p(x)$ it's independent of the model's parameters, we get the desired equivalence. Furthermore, using the fact that the KL divergence is always greater than 0, we have:

$$\log p(x) \geq \mathcal{E}_{\boldsymbol{z}\sim q_\theta}[\log p(\boldsymbol{x}|\boldsymbol{h})] - D_{KL}(q_\theta(\boldsymbol{h}|\boldsymbol{x})||p(\boldsymbol{h}))$$

or, in other words, that the ELBO is, as the name suggests, a lower bound for the log-likelihood of the data.

B Complete diagram of the MTCD-VAE architecture

Figure 1 reports the encoder structure (the decoder is analogous to it). Figure 2, instead, shows the full diagram of the MTCD-VAE architecture. The intermediate sampling process is repeated for each of the style vectors in order to obtain the final embedding $\boldsymbol{h}$.

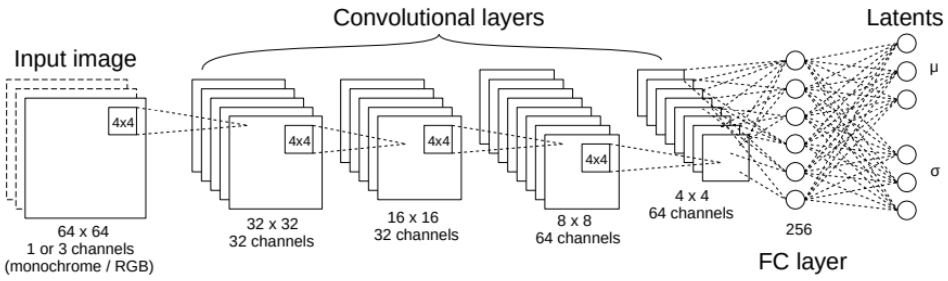

Figure 1: MTCD-VAE: encoder architecture used for the experiments.

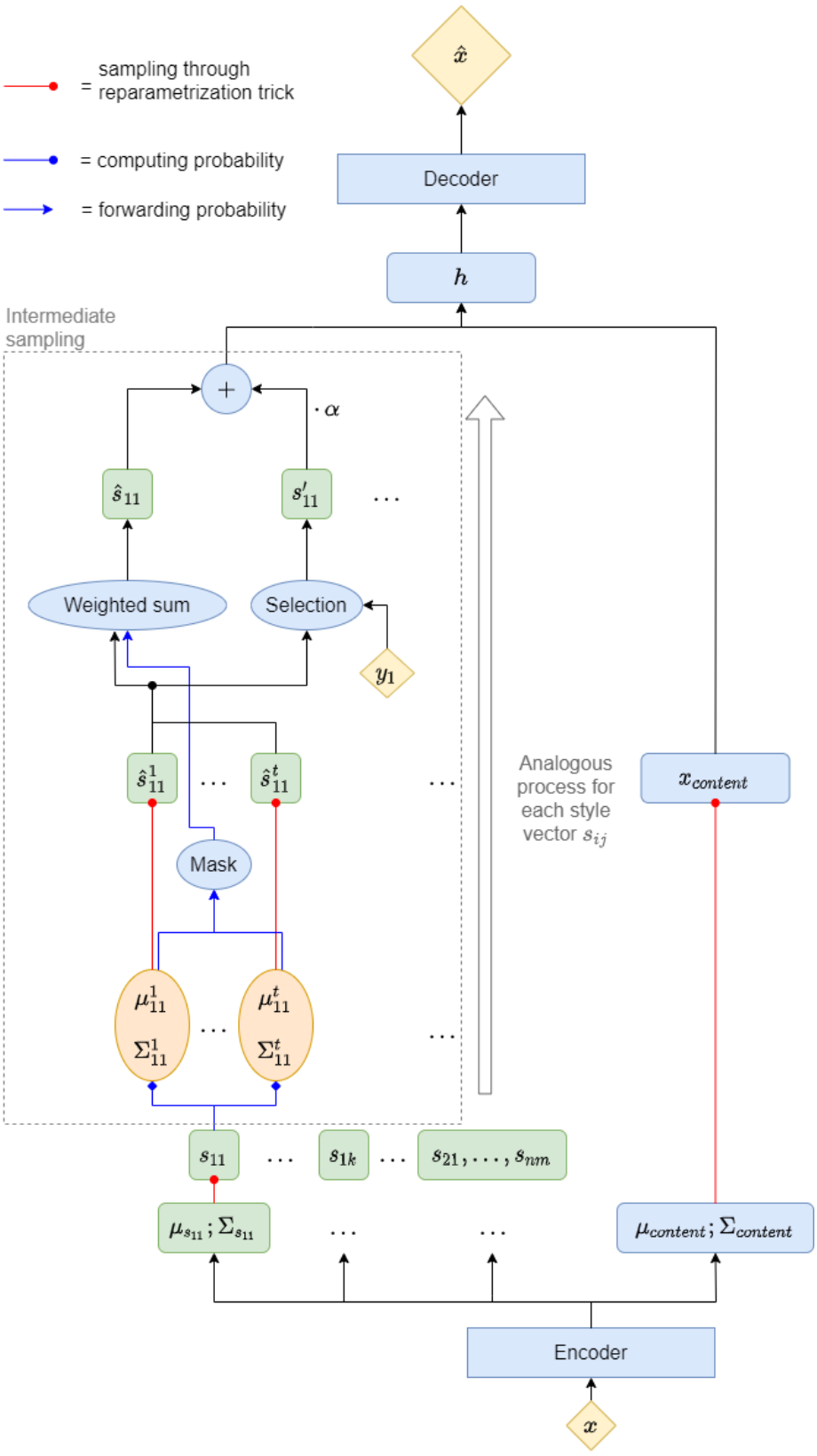

Figure 2: MTCD-VAE architecture