VRIJE
UNIVERSITEIT
BRUSSEL

Master thesis submitted in partial fulfilment of the requirements for the degree of
Master of Science In de Ingenieurswetenschappen: Computerwetenschappen

VARIATIONAL GREEDY INFOMAX

Towards independent and interpretable representations

Fabian Denoodt

2022-2023

Promotor(s): Prof. Dr. Bart de Boer
Science and Bio-Engineering Sciences

VRIJE
UNIVERSITEIT
BRUSSEL

Proefschrift ingediend met het oog op het behalen van de graad van Master of Science In de Ingenieurswetenschappen: Computerwetenschappen

[DUTCH] VARIATIONAL GREEDY INFOMAX

[Dutch] Towards independent and interpretable representations

Fabian Denoodt

2022-2023

Promotor(s): Prof. Dr. Bart de Boer

Wetenschappen en Bio-ingenieurswetenschappen

Contents

Chapter 1

Introduction

FEEDBACK BART

!!! wat is + waarom het gedaan wordt.

should also explain in my work.

should be around 60 pages.

Discussion: Sectie 4.5: (Bart wil aparte sectie) - Hoe helpt deze techniek om het netwerk interpreteerbaarder te maken. - compare techniques: whether they are better explainable. - Zou na results moeten zijn. en kan zo inleiding zijn naar future work. - bart zou verwachten dat cha 3 zeer groot is.

- UVA: wordt geschreven voor begeleiders, hun hebben meer achtergrond. - Jury aan VUB andere verwachtingen, wil meer uitleg.

- mijn discussie sectie moet langer! en zeker in vertellen waarom het beter is voor visualities.

— Defense: - verduideluikende vragen - critiek als gaten in argumentatie - hun komen met suggestie die ze uit literatuur kennen, en moet bv mening over geven. - "zou dat ook anders gekund hebben" !! —

- Context: !E GIM for representation learning: generates representations that simplify classification tasks vs when done on raw data
- Problem: If wants to know for what tasks applicable... must know what is present in data
- Solution: Analysis of learned representations on speech data
- My contributions:
  - Decoder ANN for each layer of GIM: Shows what information is maintained through the layers
  - Search correlations between kernels weights and signal features
  - Extension on CPC via VAE

Chapter 2

Background

[TODO] OUTLINE + WHY IMPORTANT TO UNDERSTAND

2.1 Entropy, relative entropy and mutual information

We first discuss specific definitions from information theory. These concepts will be relevant to understand contrastive predictive coding, which we discuss in a following section. The formal definitions are obtained from the book "Elements of Information Theory" [1]. The equations that contain a log function are assumed to be under base two.

2.1.1 Shannon's Entropy

Entropy measures the average amount of information required to describe a random variable [1]. The entropy H(X) of a discrete random variable X, is formally defined as follows:

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x) \tag{2.1}$$

where $\mathcal{X}$ represents the set of events the random variable X can take, formerly known as the *sample space*. Additionally, $p : \mathcal{X} \to [0, 1]$ denotes the probability density function of X. Hence, given an event $x \in \mathcal{X}$, $p(x)$ corresponds to the probability of event x occurring.

Assume a random variable X with possible events x_1, x_2. Intuitively, when $p(x_1)$ is low, the surprise when the event x_1 occurs will be high. The surprise for one event is defined as follows:

$$-\log p(x) \tag{2.2}$$

Hence, entropy can also be considered as the weighted average surprise over each event [2]. To understand why equation 2.2 does indeed correspond to a measure of surprise, consider an event $x \in \mathcal{X}$ with $p(x) = 1$. Note that $\log p(x) = 0$, and thus the surprise is zero. Meanwhile, if p(x) approaches 0, $\log p(x)$ goes to $-\infty$. And hence, by the negation sign in formula 2.1 the surprise is large.

Figure 2.1 displays when entropy reaches its maximum for the case of a random variable with 2 outcomes. We can see that the entropy, and thus, the information is largest when the probability of the two outcomes is equal to each other, namely $p(x_1) = p(x_2) = 0.5$. Note that for a random variable X with more than two events, $H(X)$ can be larger than one.

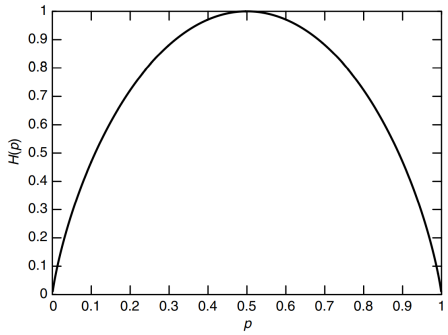


Figure 2.1: $H(p)$ vs p (originates from "Elements of Information Theory", page 16)

2.1.2 Relative entropy and mutual information

Relative entropy, also known as the Kullback Leibler (KL) divergence, is defined in equation 2.3, where p and q denote a probability density function over the same sample space $\mathcal{X}$ [1]. The KL divergence quantifies the "dissimilarity" or "divergence" between the two distributions. Note that $D(p||q)$ does not necessarily correspond to $D(q||p)$ and thus the metric is not symmetrical.

$$D_{KL}\left(q \mid\mid p\right) = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)} \tag{2.3}$$

The mutual information (MI) between two random variables X and Y can be computed as the KL divergence between their joint probability distribution, $p_{X,Y}(x, y)$, and the product of their marginal probability distributions, $p_X(x)$ and $p_Y(y)$, which is denoted as $p_X(x)p_Y(y)$ [1]. The equation for mutual information then becomes:

$$I(X;Y) = D_{KL}\left(p_{X,Y}(x,y) \mid\mid p_X(x)p_Y(y)\right) \tag{2.4}$$

As described by Cover and Thomas in their book "Elements of Information Theory" [1], MI quantifies the amount of information Y describes about X. An alternative definition is illustrated in 2.5. The equation provides us with an intuitive meaning for MI, corresponding to the surprise caused by X, which is reduced by the knowledge of Y. In sections 2.3 and 2.4, we discuss how these concepts from information theory are applied in representation learning.

$$I(X;Y) = H(X) - H(X|Y) \tag{2.5}$$

2.2 Supervised machine learning and generalisation challenges

We shall now discuss traditional supervised learning approaches, as these will lay the groundwork for the representation learning discussed in the following section.

2.2.1 Generalisation and curse of dimensionality

Consider a dataset $\mathcal{D} = \{\left(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}\right), \left(\mathbf{x}^{(2)}, \mathbf{y}^{(2)}\right), \dots\}$, with $\mathbf{x}^{(i)} \in \mathcal{X}$ correspond to a feature vector from the feature space $\mathcal{X}$ and $\mathbf{y}^{(i)} \in \mathcal{Y}$ its class label. Supervised machine learning

problems consider the task of finding a function

$$f : \mathcal{X} \to \mathcal{Y}$$

such that given a feature vector $\mathbf{x}^{(i)}$, a label $\mathbf{y}^{(i)}$ can be *inferred*. However, the entire dataset $\mathcal{D}$ is typically not available to us, and $f(\cdot)$ must be derived from only a partial dataset $\mathcal{D}_{\text{train}} \subset \mathcal{D}$. A good $f(\cdot)$ should not only ensure few errors on the training set $\mathcal{D}_{\text{train}}$, but it should also *generalise* well to unseen $\mathbf{x}^{(i)} \in \mathcal{X}$ [3, 4, 5].

Furthermore, when the number of dimensions of the input space $\mathcal{X}$ increases, the supervised task becomes even more challenging, requiring a larger $\mathcal{D}_{\text{train}}$ to ensure good generalisation. This inability of Machine Learning algorithms to manage high-dimensional data is commonly referred to as the *curse of dimensionality* [6, 7, 8].

2.2.2 Fully connected Neural Networks

Artificial Neural Networks (ANNs) address this problem of finding $f(\cdot)$, by representing it as a fixed set of parameters, known as layers, which consist of a series of transformation matrices and non-linearities [9, 10, 11]. During inference, at each layer l a transformation matrix $\mathbf{W}^l$ is applied to the output vector from the previous layer $\mathbf{a}^{l-1}$. This is shown in the equation below.

$$\mathbf{z}^l = \mathbf{W}^l \mathbf{a}^{l-1}$$

A non-linear function $\sigma : R^d \to R^d$ is then applied to $\mathbf{z}^l$, as shown in equation 2.2.2. The resulting vector a^l may then again be the input for a following layer.

$$\mathbf{a}^l = \sigma(\mathbf{z}^l)$$

Hence, during inference, the input vector $\mathbf{x}$ is propagated through each layer, resulting in a final output $\hat{y}^L$. The equation for the forward pass of a neural network with L layers is described as follows:

$$f_{\mathbf{W}^1 \dots \mathbf{W}^L}(x) = \hat{y}^L = \sigma(\dots \sigma(\sigma(x^T \mathbf{W}^1)^T)\mathbf{W}^2 \dots)^T \mathbf{W}^L$$

The complexity and power of a neural network are affected by the number of layers, and the choice of the number of parameters should depend on the complexity of the learning task. The architecture of a neural network, which includes the dimensions of the matrices, can be represented as a graph-like figure. An example is shown in 2.2 where the edges between $\mathbf{a}^{l-1}$ and $\mathbf{a}^l$ correspond to matrix $\mathbf{W}^l$.

During training the ANN's parameters $\mathbf{W} = \{\mathbf{W}^1 \dots \mathbf{W}^L\}$, are optimised according to the learning problem. The viability of the parameters is quantified by the loss function $\mathcal{L}$ over a batch of n training samples, shown as follows:

$$\mathcal{L}(\mathbf{W}) = \frac{1}{n} \sum_{i=1}^{n} e(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)})$$

where $\mathbf{y}^{(i)}$ corresponds to the ground truth label and $\hat{\mathbf{y}}^{(i)} = f_{\mathbf{W}}(\mathbf{x}^{(i)})$. The error function $e(\cdot)$ can be chosen depending on the task, for instance by mean squared error for regression or cross entropy for classification problems.

The parameters $\mathbf{W}^1 \dots \mathbf{W}^L$ are typically achieved through iterative minimisation of the loss function. This is achieved using backpropagation, an efficient algorithm which computes the partial derivates with respect to each parameter $\mathbf{W}^l_{ij}$. Updating the parameters in the opposite

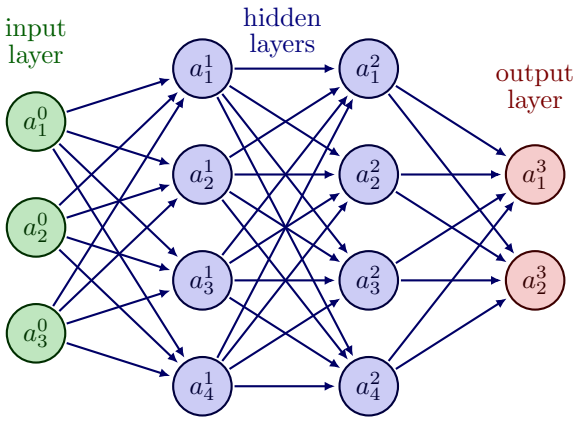


Figure 2.2: Visual representation of a neural network

direction of the partial derivative allows for an iterative estimation of a local minimum, which is demonstrated in the following equation using stochastic gradient descent:

$$\mathbf{W}_{ij}^{l} \leftarrow \mathbf{W}_{ij}^{l} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{W}_{ij}^{l}}$$

The details of computing the partial derivatives using backpropagation are beyond the scope of this thesis.

2.2.3 Towards lower dimensional feature spaces

Supervised learning through ANNs is often successful, even when dealing with complex problems. However, achieving high performance on the training set does not guarantee good generalisation to new data. As problems become more complex, the use of larger amounts of labelled data is required to ensure good generalisation. For complex feature vectors, a more sophisticated architecture may be necessary, which in turn requires even more data to prevent overfitting.

In cases where labelled data is scarce, ANNs may appear to perform well on the training set, but their generalisation performance to data outside the training set can be poor. The performance of ANNs is thus heavily dependent on the choice of data representation [12]. As a consequence, a lot of effort in the machine learning pipeline is invested in feature engineering [13], which involves extracting useful features from the feature space $\mathcal{X}$, removing redundant information, and reducing the dimensionality of the feature space [14] to obtain a good representation $\mathbf{z} \in \mathcal{Z}$ that is easier to work with. Rather than learning the mapping function directly on the data, $\mathbf{x} \in \mathcal{X}$ is first transformed into a lower-dimensional representation $\mathbf{z} \in \mathcal{Z}$. The mapping function from equation 2.2.1 is now:

$$f : \mathcal{Z} \rightarrow \mathcal{Y}$$

However, feature engineering is a time-consuming and often manual process that requires domain knowledge and expertise [15, 16]. In the following section we discuss how part of the manual labour of finding good representations from data can be relieved with unsupervised learning approaches, which automate this process. These representations could then be used as the input for a supervised predictor $f(\cdot)$ or *downstream task* [17].

2.3 Representation learning through reconstruction error

One of the challenges in supervised learning is the constant need of large amounts of labelled data $\mathcal{D}_{\text{train}} = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{y}^{(2)}), \dots\}$. When this set is scarce, or the task becomes too complex to ensure good generalisation, transforming the feature space $\mathcal{X}$ to a lower-dimensional space $\mathcal{Z}$ can be a good solution to aid with learning for the downstream task $f(\cdot)$. Meanwhile, an abundant amount of unlabelled data may already exist. Hence, when a labelled dataset is small, we would like to leverage a larger unlabelled dataset as basis for learning.

In this section we investigate two self-supervised learning paradigms which learn the following transformation function solely from a feature space $\mathcal{X}$ and thus do not require any labels:

$$T : \mathcal{X} \to \mathcal{Z}$$

such that the simplified representations obtained from $T(\cdot)$ can serve as the input for a downstream task as follows:

$$T(\mathbf{x}) = \mathbf{z}$$
$$f(\mathbf{z}) = \mathbf{y}$$

This process of learning a mapping function $T(\cdot)$ which translates a feature vector $\mathbf{x}$ to a lower-dimensional representation $\mathbf{z}$ is commonly referred to as representation learning [18].

In the following two subsections we discuss two paradigms of representation learning with ANNs. The first paradigm learns representations by minimising a reconstruction error. The second learns its representations by contrasting them against noise. These two paradigms will lay the basis for our own contributions in chapter three.

2.3.1 Autoencoders

The encoding problem, introduced by Ackley and Hinton in 1985 [19], and later referred to as the autoencoder by Rumelhart et al. in 1986 [20], considers the problem of learning compressed representations through neural networks [20, 21]. This is achieved through an ANN architecture consisting of two functions, an *encoder* $E(\mathbf{x}) = \mathbf{z}$ and a *decoder* $D(\mathbf{z}) = \tilde{\mathbf{x}}$. The lower-dimensional *encoding* $\mathbf{z}$ obtained from $E(\mathbf{x})$ will then serve as representation for downstream task $f(\cdot)$. The functions $E(\cdot)$ and $D(\cdot)$ are simultaneously optimised by minimising the reconstruction error shown in the following equation:

$$\mathcal{L} = \sum_{i=1}^{N} l(\mathbf{x}^{(i)}, \tilde{\mathbf{x}}^{(i)}) \tag{2.6}$$

where l refers to the error for a single data point, for instance the l^2-norm and N the number data points. The dimension of $\mathbf{z}$ is typically smaller than the original dimension of $\mathbf{x}$. This results in the encoder having to define encodings that are as "informative" as possible to reconstruct the original data [21].

An example autoencoder architecture is depicted in figure 2.3. Since an autoencoder is in fact a single ANN and $\mathbf{z}$ corresponds to the output produced by an intermediate hidden layer, $\mathbf{z}$ is also commonly referred to as a *latent* representation. The space of all latent representations $\mathcal{Z}$ is known as the *latent* space. While capable of learning compressed representations, autoencoders do not pose any restrictions on the latent space $\mathcal{Z}$. As a result, the representations may be meaningful to computers, but non-interpretable to humans. For instance, given the left image depicted in figure 2.4 which depicts an autoencoder's two dimensional latent space, it is almost impossible to infer in advance what the corresponding digit would be when interpolating a new

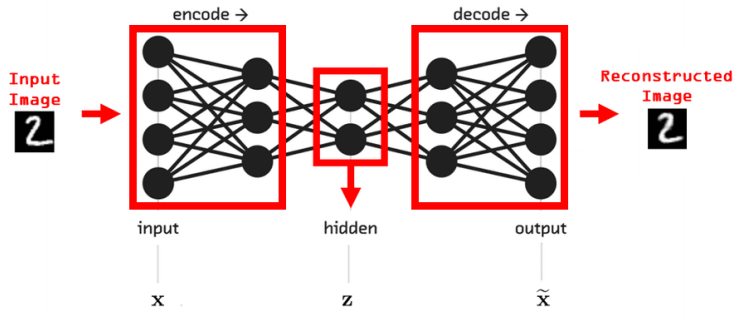


Figure 2.3: Autoencoder architecture, adapted from [22].

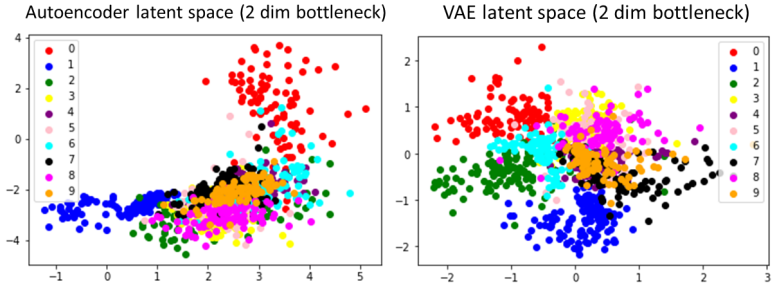


Figure 2.4: Latent space of autoencoder (left) and VAE (right), learned from the MNIST dataset [23], a dataset consisting of images of handwritten digits between 0 and 9. The ANNs have not received any explicit information of the labels of the dataset.

$\mathbf{z}$ somewhere in between the encodings corresponding to digit 0 (red) and 1 (blue). In addition, slight changes to $\mathbf{z}$ may result in significant changes to its decoded counterpart $\mathbf{x}$. Finally, attempting to understand an autoencoders latent space $\mathcal{Z}$ becomes even more infeasible as the dimension of $\mathcal{Z}$ increases.

2.3.2 Variational autoencoders

Similar to traditional autoencoders, variational autoencoders (VAE) learn representations that contain the important information necessary to reconstruct the data. VAEs maintain the same structure from autoencoders, consisting of encoder $E(\mathbf{x}) = \mathbf{z}$ and decoder $D(\mathbf{z}) = \tilde{\mathbf{x}}$. However, an additional constraint is applied to the latent space $\mathcal{Z}$ [24, 25, 26, 27, 28]. In fact, the latent representations $E(\mathbf{x}) = \mathbf{z}$ are samples from multivariate distributions, which we will denote by

$$\mathbf{z} \sim q(\mathbf{z} \mid \mathbf{x})$$

Thus, given a feature vector $\mathbf{x}$, we obtain its corresponding encoding or latent representation $\mathbf{z}$ by taking a sample from a distribution. We will define q in more detail in a following paragraph. The encoder function $E(\cdot)$ in VAEs is thus nondeterministic and computing $E(\mathbf{x}) = \mathbf{z}$ twice may result in two different $\mathbf{z}$s both corresponding to the same $\mathbf{x}$. In a following paragraph we will discuss how to achieve this nondeterministic behaviour using ANNs.

By defining the latent representations as samples from distributions, we can pose the same constraints on the quality of the representations $\mathbf{z} \in \mathcal{Z}$ as in traditional autoencoders, but also

additional constraints on how the latent space $\mathcal{Z}$ is organised. The additional constraints posed by VAEs on $\mathcal{Z}$ will result in a more interpretable space, where $\mathbf{z}$'s underlying components can be better understood. This will be achieved by enforcing each distribution $q(\cdot \mid \mathbf{x}^{(i)})$ corresponding to a particular $\mathbf{x}^{(i)}$, to be "similar" to a chosen prior distribution $p(\mathbf{z})$. Typically, the prior $p(\mathbf{z})$ is set to the standard normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$ [25]. The result of this setup can be observed in the latent space depicted in the right plot of figure 2.4 where all $\mathbf{z}$s are attracted to the origin.

In the following subsections, we will explore VAEs in more detail. We begin with a discussion on the process of imposing constraints on $\mathcal{Z}$, which leads to the VAE loss function. Next, we investigate how ANNs can be utilised to simulate sampling from distributions. Finally, we examine how VAEs enable the creation of interpretable representations and how they can be used to generate novel data.

The learning objective

So far, we have learned that Variational Autoencoders (VAEs) use an encoder $E(\mathbf{x}) = \mathbf{z}$ to create a latent representation $\mathbf{z}$ of the input feature vector $\mathbf{x}$, where $\mathbf{z}$ is a sample from the distribution $\mathbf{z} \sim q(\mathbf{z} \mid \mathbf{x})$. Equivalently, for each $\mathbf{x}$, there exists a unique distribution $q(\mathbf{z})$ that characterizes $q(\mathbf{z} \mid \mathbf{x})$ [29]. We can then define $q(\mathbf{z} \mid \mathbf{x})$ as a Gaussian with independent components, parametrised by $\boldsymbol{\mu}$ and covariance matrix $\text{diag}(\boldsymbol{\sigma})$. We obtain the the following definition:

$$q(\mathbf{z} \mid \mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma})) \tag{2.7}$$

where $\text{diag}(\boldsymbol{\sigma})$ denotes the covariance matrix with $\boldsymbol{\sigma}$ on the diagonal and zeroes elsewhere. It is important to recognise that $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are different for every $\mathbf{x}$. The two parameters will be computed by an ANN.

The representations are optimised to minimise two measurements: one, the reconstruction error, and secondly, the dissimilarity from the latent distributions to a chosen prior $p(\mathbf{z})$. The loss function to be optimised for a single data point $\mathbf{x}^{(i)}$ is shown in the equation below.

$$\mathcal{L}(\mathbf{x}^{(i)}) = \mathop{\mathbb{E}}_{\mathbf{z}^{(i)} \sim q(\cdot \mid \mathbf{x}^{(i)})} \Big[-\log p(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)}) \Big] + D_{KL} \Big(q(\cdot \mid \mathbf{x}^{(i)}) \mid\mid p(\cdot) \Big) \tag{2.8}$$

Here, $p(\cdot)$ refers to the prior distribution, which is different from $p(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)})$. Although $\mathcal{L}$ may seem daunting at first, we will decompose its components. The loss function is made up of two terms, the left term corresponds to the reconstruction error, while the second term poses constraints on the latent space. Let us focus on the reconstruction term first.

$$\mathop{\mathbb{E}}_{\mathbf{z}^{(i)} \sim q(\cdot \mid \mathbf{x}^{(i)})} \Big[-\log p(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)}) \Big] \tag{2.9}$$

In a traditional autoencoder we optimise $D(E(\mathbf{x}^{(i)})) = \tilde{\mathbf{x}}^{(i)}$ such that $\mathbf{x}^{(i)} \approx \tilde{\mathbf{x}}^{(i)}$. We will now show that the reconstruction term in equation 2.9 achieves a similar goal for VAEs. Here, $p(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)})$ is a distribution over $\mathbf{z}^{(i)}$ where $\mathbf{x}^{(i)}$ is a constant, and the latent representation $E(\mathbf{x}^{(i)}) = \mathbf{z}^{(i)}$ is sampled from the Gaussian distribution $\mathbf{z}^{(i)} \sim q(\cdot \mid \mathbf{x}^{(i)})$. The objective is to ensure that $D(\mathbf{z}^{(i)}) \approx \mathbf{x}^{(i)}$. However, since $\mathbf{z}^{(i)}$ is sampled from a Gaussian distribution, the $\mathbf{z}^{(i)}$'s close to the mean $\boldsymbol{\mu}^{(i)}$ are more likely to be sampled than those further away. Thus, for these $\mathbf{z}^{(i)}$'s, we'd like the probability of corresponding to the actual $\mathbf{x}^{(i)}$ to be high, or equivalently we want $p(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)}) \approx 1$ [28]. Finally, by adding a negative sign in front, maximising this probability is equivalent to minimising the negative probability. In practice this term is approximated through mini-batches with the mean squared error.

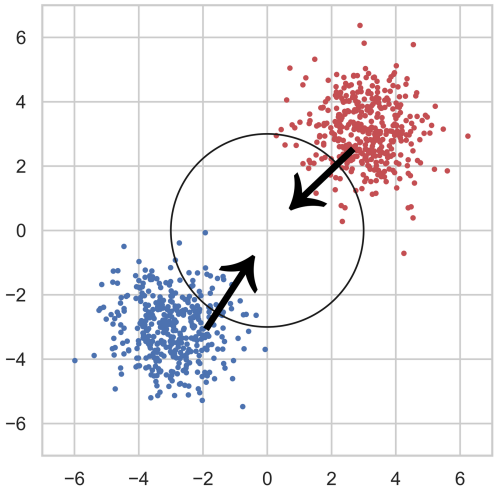


Figure 2.5: 200 samples $\mathbf{z}^{(i)}$ and $\mathbf{z}^{(j)}$ from feature vectors $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$, respectively, represented by red and blue data points. The prior $p(\mathbf{z})$ is set to the standard normal, and the regularisation term of VAE's loss pushes the latent space towards the origin.

Optimising the second term of equation 2.8 poses the constraints on the distributions $q(\cdot \mid \mathbf{x}^{(i)})$.

$$D_{KL}\left(q(\cdot \mid \mathbf{x}^{(i)}) \mid\mid p(\cdot)\right) \tag{2.10}$$

Again, this metric should be minimised. As we discussed in chapter 2.1.2, KL divergence can be considered as a dissimilarity measure between two distributions. Hence, this value is small when the two distributions are similar. The prior distribution $p(\cdot)$ is a fixed distribution which is chosen by the user, typically defined as $\mathcal{N}(\mathbf{0}, \mathbf{I})$. Minimising this metric will result in moving each distribution $q(\mathbf{z} \mid \mathbf{x}^{(i)})$, corresponding to a value $\mathbf{x}^{(i)}$, close to $\mathcal{N}(\mathbf{0}, \mathbf{I})$. This idea is depicted in figure 2.5. When $p(\cdot) = \mathcal{N}(\mathbf{0}, \mathbf{I})$, equation 2.10 becomes:

$$D_{KL}\left(q(\cdot \mid \mathbf{x}^{(i)}) \mid\mid \mathcal{N}(\mathbf{0}, \mathbf{I})\right)$$

When $q(\cdot \mid \mathbf{x}^{(i)}) = \mathcal{N}(\boldsymbol{\mu}^{(i)}, \text{diag}(\boldsymbol{\sigma}^{(i)}))$, one can algebraically prove, that optimising the above equation is equivalent to optimising the following [26]:

$$D_{KL}\left(\mathcal{N}(\boldsymbol{\mu}^{(i)}, \text{diag}(\boldsymbol{\sigma}^{(i)})) \mid\mid \mathcal{N}(\mathbf{0}, \mathbf{I})\right) = \frac{1}{2}\sum_{k=1}^{D}\left(-\log(\sigma_k^{(i)})^2 - 1 + (\sigma_k^{(i)})^2 + (\mu_k^{(i)})^2\right) \tag{2.11}$$

where $\sigma_k^{(i)}$ and $\mu_k^{(i)}$ correspond to the components of the predicted vectors $\boldsymbol{\sigma}^{(i)}$ and $\boldsymbol{\mu}^{(i)}$, respectively. The latent space has dimensionality D.

Simulating distributions through neural networks

Computing $E(\mathbf{x}^{(i)})$ results in encoding $\mathbf{z}^{(i)}$ which is sampled from $\mathbf{z}^{(i)} \sim q(\cdot \mid \mathbf{x}^{(i)})$. We will now discuss how ANNs can emulate the stochastic behaviour of sampling from a distribution.

As discussed in equation 2.7, distribution $q(\cdot \mid \mathbf{x}^{(i)})$ is parametrised as a Gaussian with mean $\boldsymbol{\mu}^{(i)}$ and covariance matrix $\text{diag}(\boldsymbol{\sigma}^{(i)})$. The entire distribution can thus be modelled by an ANN which takes as input $\mathbf{x}^{(i)}$ and generates the two corresponding vectors $\boldsymbol{\mu}^{(i)}$ and $\boldsymbol{\sigma}^{(i)}$. Finally, a sample $\mathbf{z}^{(i)}$ can be obtained as follows:

$$\mathbf{z}^{(i)} = \boldsymbol{\mu}^{(i)} + \boldsymbol{\sigma}^{(i)} \odot \boldsymbol{\epsilon}^{(i)} \tag{2.12}$$

where $\boldsymbol{\epsilon}^{(i)}$ corresponds to a sampled value $\boldsymbol{\epsilon}^{(i)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\odot$ is element-wise multiplication. Computing $\mathbf{z}^{(i)}$ through $\boldsymbol{\epsilon}^{(i)}$, rather than directly sampling from $\mathbf{z}^{(i)} \sim \mathcal{N}(\boldsymbol{\mu}^{(i)}, \text{diag}(\boldsymbol{\sigma}^{(i)}))$ is referred to as the parametrisation trick and allows for gradients to freely backpropagate through the layer [25]. The decoder $D(\mathbf{z}^{(i)}) = \tilde{\mathbf{x}}^{(i)}$ can remain identical to the traditional autoencoder's decoder. An overview is presented in figure 2.6.

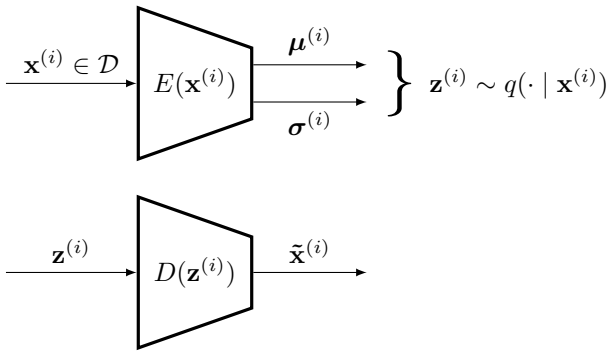


Figure 2.6: High level view of a VAE.

Generating new data and interpretability

Due to the VAE's regularisation term all encodings are pushed towards the centre as depicted in figure 2.5. Additionally, the space of encodings corresponding to a single $\mathbf{x}^{(i)}$ is an entire neighbourhood around a particular $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$. These two properties ensure a well covered and uninterrupted space around the origin with smooth transitions on the generative factors between latent representations [25]. As a result, new samples from the dataset can be generated by discarding the VAE's encoder and providing standard normal noise to the decoder instead. It is important to understand that this method only works for VAEs and not for the traditional autoencoder. This is because autoencoders do not pose any additional constraints on the latent space. Generating new data points from random noise via a traditional autoencoder may result in nonsense generations as the random noise may be too different from the latent space it was trained on. As a result, there are no guarantees that the autoencoder's decoder will generalise to these random representations.

Additionally, for the same reason, we can interpolate between encodings obtained from the data. The interpolated encoding can then decoded to the original space. We can observe the specific information contained in each of the representations' features through the decoder, resulting in a significant improvement in interpretability.

Disentanglement of latent representations and posterior collapse

Higgins et al. extend the VAE framework through the introduction of a slight modification of the loss function, resulting in β-VAE [30]. An additional hyper-parameter β is introduced in

VAE's loss function to control the weight of the regularisation term:

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}^{(i)}) = \mathop{\mathbb{E}}_{\mathbf{z}^{(i)}\sim q(\cdot|\mathbf{x}^{(i)})}\Big[-\log p(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)})\Big] + \beta\, D_{KL}\Big(q(\cdot \mid \mathbf{x}^{(i)}) \,||\, p(\cdot)\Big)$$

Setting the prior to a standard normal distribution ($p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$) encourages disentangled representations, where each variable is sensitive to a specific generative factor [30, 12]. For example, in an image of a face, altering one component of the representation would only change the smile, while other features like skin colour and brightness remain constant. Such disentangled representations are interpretable, making it easier to understand the underlying structure of the representation and what information each variable contains.

However, a trade-off must be made between accuracy and disentanglement. A small β value (close to 0) results in high reconstruction accuracy but little constraint on disentanglement, while a large β value (above 1) emphasises disentanglement at the cost of accuracy. If β becomes too large, the posteriors $q(\mathbf{z} \mid \mathbf{x}^{(i)})$ corresponding to each data point $\mathbf{x}^{(i)}$ can become equal to the prior $p(\mathbf{z})$, resulting in no information contribution and posterior collapse [31].

2.4 Representation learning through Noise-contrastive estimation

2.4.1 Contrastive predictive coding

In what follows next, we discuss Contrastive Predictive Coding (CPC), a representation learning approach that we use as the basis for our own experiment in the following chapter. CPC is an unsupervised learning approach, again with the objective of learning (lower dimensional) representations from high dimensional data [32]. While the objective is thus the same as for the autoencoders discussed in the previous section, CPC achieves its representations entirely differently. An autoencoder's objective is to define a compressed representation from which the original data can be recovered. However, when working with sequential data $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t, \ldots$, simply compressing patches $\mathbf{x}_i$ of the sequence without considering the relation with nearby patches can result in suboptimal encodings, as contextual information between patches is not encoded directly into the representation [33].

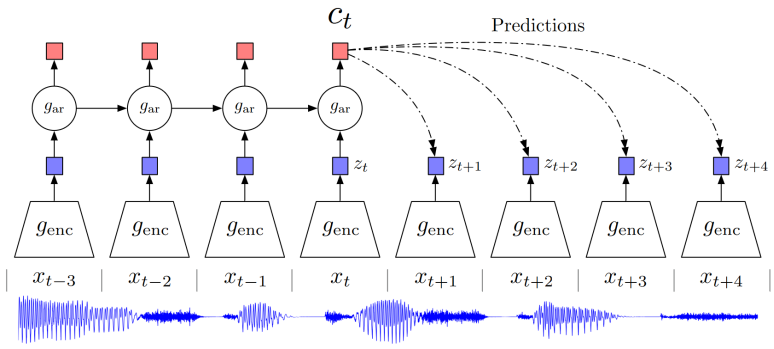


Figure 2.7: Overview of Contrastive Predictive Coding, originates from [32].

CPC deals with these context issues by maximising the shared information between the extracted representations of temporally nearby patches [34]. We will discuss this concept in

more detail in this section. For now, we would like to draw the reader's attention to figure 2.7. The figure depicts a high level view of how this idea is achieved by producing two latent latent representations $\mathbf{z}_i$ and $\mathbf{c}_i$. An audio sequence of undefined length is split up into patches $\mathbf{x}_1 \dots \mathbf{x}_n$ where each $\mathbf{x}_i$ is a vector of fixed length, containing for instance 10ms of speech audio. Each patch $\mathbf{x}_i$ is encoded into latent representation $\mathbf{z}_t$, defined as follows:

$$\mathbf{z}_t = g_{enc}(\mathbf{x}_t).$$

$g_{enc}(\cdot)$ is for instance a Convolutional Neural Network (CNN). The latent representations $\mathbf{z}_1..\mathbf{z}_n$ are obtained independently from each other and do not yet contain any contextual information. This is achieved through $g_{ar}(\cdot)$, an auto-regressor, which encodes all previous $\mathbf{z}_1 \dots \mathbf{z}_t$ into a single representation $\mathbf{c}_t$:

$$\mathbf{c}_t = g_{ar}(\mathbf{z}_1 \dots \mathbf{z}_t)$$

Either $\mathbf{z}_t$ or $\mathbf{c}_t$ could be used as latent representation for downstream tasks. Oord et al. suggest to use $\mathbf{c}_t$ for tasks where context about the past is useful, for instance speech recognition, and $\mathbf{z}_t$ when historic context is not useful [32]. As shown in figure 2.7, the encodings from sequential data of undefined length, may correspond a series of latent representations $\mathbf{c}_1, \mathbf{c}_2, \dots$ or $\mathbf{z}_1, \mathbf{z}_2, \dots$. In the case of downstream tasks which require a single representation vector, Oord et al. propose to pool the sequence of vector representations into a single vector.

Slowly varying features

The temporally nearby patches $\mathbf{z}_{t+1}$ and $\mathbf{c}_t$ are optimised to preserve shared information, while discarding differences. Before we discuss how to obtain such representations, we first motivate why defining representations in this fashion makes sense.

Consider we would like to define useful representations for sequential data such as speech signals. Then it is not unlikely to believe that the conveyed information at time step t and $t+k$ contains some redundancy, such as pitch, frequency, tone, etc. [35]. Meanwhile, large changes of the signal in a small time window, may be the result of noise. Sequential data which poses these slowly varying features, are commonly referred to as "slow-features" [36]. CPC leverages these slowly varying features, by encoding the underlying shared information between different patches, while at the same time discarding low-level information and noise that is more local [32].

The learning objective

CPC will learn to preserve information between temporally nearby representations, by solving another task. In particular, CPC learns to discriminate subsequent *positive* samples $\mathbf{z}_{t+k}$ from *negative* random samples $\mathbf{z}_j$. This is achieved through a similarity function $f_k(\cdot)$, which scores the similarity between two latent representations [34]. It is defined as a log bilinear model as follows:

$$f_k(\mathbf{z}_j, \mathbf{c}_t) = \exp(\mathbf{z}_j^T W_k \mathbf{c}_t) \tag{2.13}$$

where W_k is a weight matrix which is learned. $f_k(\mathbf{z}_j, \mathbf{c}_t)$ thus quantifies how likely the context $\mathbf{c}_t$ corresponds to a random vector $\mathbf{z}_j$. Due to the slowly varying data assumption, a good representation for successive representations $\mathbf{z}_{t+k}$ and $\mathbf{c}_t$ is one where $f_k(\mathbf{z}_{t+1}, \mathbf{c}_{t+1})$ is high and $f_k(\mathbf{z}_j, \mathbf{c}_t)$ is small for random $\mathbf{z}_j$. Or equivalently, maximising the shared information between temporally nearby patches, while discarding the temporal noise results in large values $f_k(\mathbf{z}_{t+1}, \mathbf{c}_{t+1})$.

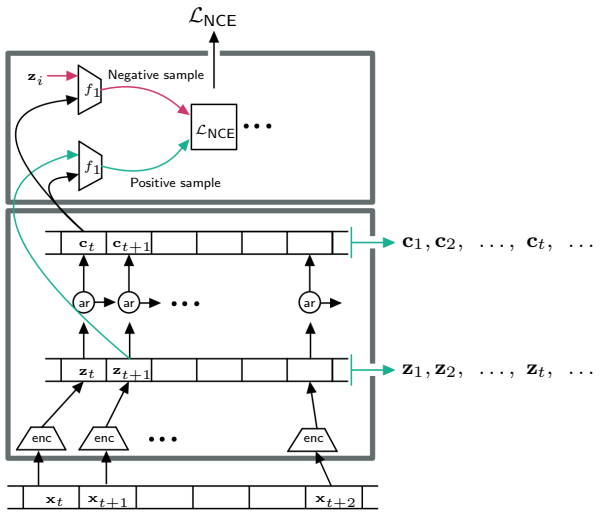


Figure 2.8: Overview of CPC. The encoder, autoregressor and f are ANNs and their parameters are optimised via $\mathcal{L}_{\text{NCE}}$.

The InfoNCE loss, used to optimise g_{enc}, g_{ar} and W_k simultaneously is shown below.

$$\mathcal{L}_{\text{NCE}} = -\sum_{k} \mathop{\mathbb{E}}_{X} \left[\log \frac{f_k(\mathbf{z}_{t+k}, \mathbf{c}_t)}{\sum_{\mathbf{z}_j \in X} f_k(\mathbf{z}_j, \mathbf{c}_t)} \right] \tag{2.14}$$

Here, X corresponds to the set $\{\mathbf{z}_{t+k}, \mathbf{z}_1, \mathbf{z}_2, \ldots\}$. Notice that there exists exactly one $\mathbf{z}_{t+k} \in X$, which corresponds to a positive sample for $\mathbf{c}_t$ and all other $\mathbf{z}_j \in X$ are negative samples. Hence good representations should result in a large similarity score $f_k(\mathbf{z}_{t+k}, \mathbf{c}_t)$, while $f_k(\mathbf{z}_i, \mathbf{c}_t) \approx 0$ for negative samples, resulting in a fraction equal to 1. This would lead to a loss of 0 by the $\log(\cdot)$ function. On the other hand, $\mathcal{L}_{\text{NCE}}$ is large when the denominator is large, which occurs when $f(\cdot)$ is often unsure about negative samples and produces large values for those as well. An overview of the three ANNs which each must be optimised is depicted in figure 2.8.

Ties with mutual information

Earlier we argued that CPC's encodings will preserve shared information between temporally nearby patches, while discarding the local noise. Oord et al. make this claim even stronger by making ties with mutual information, which we discussed in a previous chapter. In particular, Oord et al. prove that optimising InfoNCE is equivalent to maximising the mutual information between $\mathbf{c}_t$ and $\mathbf{z}_{t+1}$ [32].

$$I(\mathbf{z}_{t+1}; \mathbf{c}_t) = \sum_{\mathbf{z}_{t+1}, \mathbf{c}_t} p(\mathbf{z}_{t+1}, \mathbf{c}_t) \log \frac{p(\mathbf{z}_{t+1} \mid \mathbf{c}_{t+1})}{p(\mathbf{z}_{t+1})} \tag{2.15}$$

This proof is available in their appendix. Although, we do not repeat the proof here, we give a high level overview.

The first step in proving the relation between the InfoNCE loss and mutual information is to model $f_k(\mathbf{z}_{t+k}, \mathbf{c}_t)$ in a probabilistic manner. The InfoNCE loss is in fact the categorical cross-entropy of classifying the positive sample correctly with $\frac{f_k}{\sum_X f_k}$ as the predicted model [32].

Since this equation may take values between zero and one, it can be considered as a probability. In particular, the optimal probability for the loss can then be written as

$$p(i \mid X, \mathbf{c}_t)$$

where X corresponds the set of samples $\{\mathbf{z}_{t+k}, \mathbf{z}_1, \mathbf{z}_2, \dots\}$ as discussed in the InfoNCE loss, and i corresponds to indicator that sample $\mathbf{z}_i$ is the "positive" sample. By doing so, one can eventually obtain a proportionality relation to the density distribution presented below.

$$f_k(\mathbf{z}_{t+k}, \mathbf{c}_t) \propto \frac{p(\mathbf{z}_{t+k} \mid \mathbf{c}_t)}{p(\mathbf{z}_{t+k})} \tag{2.16}$$

Oord et al. utilise this proportionality relation to reformulate $-\mathcal{L}_{\text{NCE}}$ as a lower bound on the mutual information between $\mathbf{z}_{t+1}$ and $\mathbf{c}_t$ as follows [34, 32]:

$$I(\mathbf{z}_{t+1}; \mathbf{c}_t) \geq \log(N) - \mathcal{L}_{\text{NCE}} \tag{2.17}$$

Since the number of samples N is a constant, the mutual information between $\mathbf{z}_{t+1}$ and $\mathbf{c}_t$ becomes greater when $\mathcal{L}_{\text{NCE}}$ becomes smaller. Additionally, when the number of samples N increases, the bound becomes tighter.

2.4.2 Greedy InfoMax

So far we discussed how CPC encodes patches of sequential data into latent representations by maximising the mutual information between temporally nearby representations. This method has shown great success in recent years and is considered state-of-the art in self-supervised learning for encoding sequential data [37]. Additionally, CPC has been successfully applied to multiple use cases [37, 38, 39, 40, 41, 42]. This is achieved by minimising the InfoNCE loss discussed earlier in equation 2.14. Through this *global* loss function all parameters are optimised end-to-end via backpropagation.

Even though empirical evidence has shown that backpropagation is highly effective [43, 44], it still suffers from multiple constraints. Firstly, there is a biological perspective to consider, as the human brain lacks a global objective function that can be optimised by backpropagating an error signal [45]. This is especially significant when considering how children can learn to categorise from a few examples, whereas end-to-end backpropagation often requires extensive datasets to achieve good generalisation [34]. Moreover, end-to-end backpropagation also suffers from computational constraints, such as requiring the entire computational graph, including all parameters, activations and gradients to fit in memory [34]. Additionally, during the training process of a neural network, each layer has to wait for the gradients of its subsequent layer, which reduces locality and impedes the efficiency of hardware accelerator design [34].

Towards greedy learning

To overcome these computational constraints, Löwe et al. introduce Greedy InfoMax (GIM) [34], an extension on CPC inspired from the biological brain. Whereas CPC obtains representations $\mathbf{z}_t$ and $\mathbf{c}_t$ through encoder $g_{enc}(\cdot)$ and autoregressor $g_{ar}(\cdot)$, Löwe et al. split up $g_{enc}(\cdot)$'s neural network architecture by depth into M so called "*modules*":

$$g_{enc}^1(\cdot),\ g_{enc}^2(\cdot),\ \dots,\ g_{enc}^M(\cdot)$$

A single module may for instance represent one or more layers. Each module's output is the input of the successive module.

$$g^m_{enc}(\mathbf{z}^{m-1}_t) = \mathbf{z}^m_t$$
$$g_{ar}(\mathbf{z}^M_1 \ \dots \ \mathbf{z}^M_t) = \mathbf{c}_t$$

The final representation $\mathbf{z}^M_t$ is obtained by propagating $\mathbf{x}_t$ through each module as follows:

$$g^M_{enc}(\dots g^2_{enc}(g^1_{enc}(\mathbf{x}_t))) = \mathbf{z}^M_t$$

Both $\mathbf{z}^M_t$ or $\mathbf{c}_t$ may serve as representation for following downstream tasks.

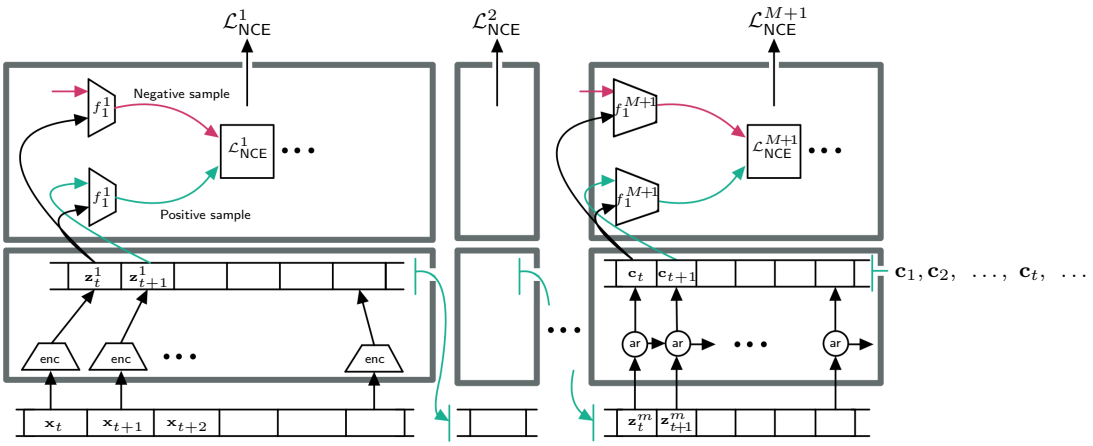


Figure 2.9: Overview of Greedy InfoMax

Each module $g^m_{enc}(\cdot)$ is *greedily* optimised with an adaptation of the InfoNCE Loss. This idea is depicted in figure 2.9, which displays M modules, each trained with their own instance of the InfoNCE loss.

In contrast to CPC where a single loss function $\mathcal{L}_{\text{NCE}}$ and scoring function $f_k(\cdot)$ are required, we now require $\mathcal{L}^m_{\text{NCE}}$ and $f^m_k(\cdot)$ for each module. The equations to optimise for $g^m_{enc}(\cdot)$ then become:

$$f^m_k(\mathbf{z}^m_{t+k}, \mathbf{z}^m_t) = \exp({\mathbf{z}^m_{t+k}}^T W^m_k \mathbf{z}^m_t) \tag{2.18}$$

$$\mathcal{L}^m_{\text{NCE}} = -\sum_k \mathop{\mathbb{E}}_{\text{X}} \left[\log \frac{f^m_k(\mathbf{z}^m_{t+k}, \mathbf{z}^m_t)}{\sum_{\mathbf{z}^m_j \in X} f^m_k(\mathbf{z}^m_j, \mathbf{z}^m_t)} \right] \tag{2.19}$$

From equation 2.18 we observe, $f^m_k(\cdot)$ no longer receives as input an aggregate $\mathbf{c}_t$, but instead a second $\mathbf{z}_t$. This is in contrast to CPC where $f^m_k(\mathbf{z}^m_{t+k}, \mathbf{c}^m_t)$ is maximised instead. Löwe et al. thus omit the autoregressive function within each module. Optionally, for downstream tasks where broad context is helpful, such as classification of phonetic structures in speech recognition, an autoregressive model $g_{ar}(\mathbf{z}^M_1 \ \dots \ \mathbf{z}^M_t) = \mathbf{c}_t$ can be appended as a final $M+1$'th module. The altered scoring function then becomes:

$$f^{M+1}_k\left(\mathbf{z}^M_{t+k}, \mathbf{c}_t\right) = \exp\left({\mathbf{z}^M_{t+k}}^T W^{M+1}_k c_t\right)$$

As a result of this approach, the mutual information $I(\mathbf{z}_{t+k}^m, \mathbf{z}_t^m)$ is maximised, rather than $I(\mathbf{z}_{t+k}, \mathbf{c}_t)$ as in CPC.

As a result of GIM's greedy approach, modules can be trained in parallel, but also sequentially. By training one module after the other, the memory cost can be decreased during training which is beneficial for memory-constrained scenarios. Furthermore, when a higher level of abstraction is needed, additional modules can be appended to the architecture later on without having to fine-tune previous modules.

Chapter 3

Variational Greedy InfoMax

3.1 Motivation

In the previous section we discussed two categories of representation learning through deep learning. First, we discussed the autoencoder and its variational counterpart, which minimise the reconstruction error. Secondly, we discussed Contrastive Predictive Coding and Greedy InfoMax, both of which optimise the InfoNCE objective. This category seeks to maximise the mutual information between the encodings of data patches that are temporally nearby. The latent representations obtained from all four methods can then be utilised for downstream tasks [12, 46, 32, 34]

The autoencoder's sole objective is to define representations to reconstruct the original data. As a result, the representations may serve well for data compression, however, no additional constraints are enforced, such as feature disentanglement and thus the latent space may still be hard to work with for downstream tasks [47]. Meanwhile, VAEs with a standard normal prior, enforce representations which break down or disentangle each feature into a narrowly defined variable and encodes them as separate dimensions [46]. This additional constrained may result in better suited representations for downstream tasks, as unit components are less influenced by multiple generative factors at the same time.

Both autoencoders and VAEs merely learn to reconstruct the data. Hence, all the "information" that is important to reconstruct the data will be maintained in the latent representation, whether the information is useful for the downstream task or not. Meanwhile, optimising latent representations for the InfoNCE objective will maintain shared information between temporally nearby patches, while discarding local noise. Reconstruction is thus not needed for training. This strategy has the tremendous benefit that a decoder block is not required, resulting in a significantly simplified architecture, meanwhile maintaining state-of-the-art performance [37]. A second benefit of these mutual information maximisation models is that they are directly compatible with sequential data.

Both categories (reconstruction and information maximisation algorithms) possess the ability to obtain useful representations for various downstream tasks. However, the content of these representations may not always be intuitive to humans and their structure may be difficult to comprehend. While CPC and GIM are considered state-of-the-art, their performance comes at a cost of having the least interpretable representations. Autoencoders maintain interpretability by using a decoder to reveal the information contained in the latent representation. The same transparency can also be achieved with VAEs. Additionally, by using a standard Gaussian as a prior and constraining the latent distributions to be similar to this prior, we can interpolate

between representations and observe the effects through the decoder. As such, we can observe the specific information that is contained in each of the representation's features. VAEs can also result in disentangled features, further enhancing interpretability [48]. In contrast, CPC and GIM do not contain a built in decoder mechanism, nor pose constraints on the latent space, significantly reducing interpretability.

3.2 Towards decoupled training for probabilistic representations

In what follows next we introduce Variational Greedy InfoMax (V-GIM), maintaining the state-of-the-art performance obtained from optimising InfoNCE, while leveraging the interpretable and disentangled benefits from VAEs. This is achieved by optimising a novel loss function, *Variational-InfoNCE*, a combination of InfoNCE and the regularisation term from VAEs. Additionally, by splitting up the neural network into modules, as introduced in [34], we greedily optimise each module with its own instance of this loss function. As a result, the interpretability benefits from VAEs will also be applicable in-between modules. This is in contrast to VAEs where solely the final output representations are interpretable.

As discussed in the section on Contrastive Predictive Coding (CPC), a patch of sequential data $\mathbf{x}_t$ is encoded through $g_{enc}(\mathbf{x}_t) = \mathbf{z}_t$ and aggregated over previous encodings through auto-regressor $g_{ar}(\mathbf{z}_1 \dots \mathbf{z}_t) = \mathbf{c}_t$, where both $\mathbf{z}_t$ or $\mathbf{c}_t$ may serve as representations for downstream tasks. The encoder function $g_{enc}(\cdot)$ is represented as neural network, eg via a CNN, and $g_{ar}(\cdot)$ for instance as a Gated recurrent unit (GRU). Finally, the encoding functions $g_{enc}(\cdot)$ and $g_{ar}(\cdot)$ are obtained by optimising a global loss function, the InfoNCE loss, end-to-end via backpropagation.

Instead, in this study, we split up $g_{enc}(\cdot)$'s network architecture by depth into M modules

$$g^1_{enc}(\cdot),\ g^2_{enc}(\cdot),\ \dots,\ g^M_{enc}(\cdot)$$

and prevent gradients from flowing between modules, as introduced in [34]. An additional optional $M+1$'th module $g_{ar}(\cdot)$ can be appended to the architecture. Each module is greedily optimised via a novel loss function, $\mathcal{L}_{\text{V-NCE}}$, which we will define in a following subsection. Each module's output serves as input for the successive module, as presented in the following equations.

$$\begin{aligned} g^1_{enc}(\mathbf{x}_t) &= \mathbf{z}^1_t \\ g^m_{enc}(\mathbf{z}^{m-1}_t) &= \mathbf{z}^m_t \\ g_{ar}(\mathbf{z}^M_1 \ \dots \ \mathbf{z}^M_t) &= \mathbf{c}_t \end{aligned}$$

The final representation $\mathbf{z}^M_t$ is obtained by propagating $\mathbf{x}_t$ through each modules as follows:

$$g^M_{enc}(\dots g^2_{enc}(g^1_{enc}(\mathbf{x}_t))) = \mathbf{z}^M_t$$

Additionally, taking inspriation from VAEs, the outputs from $g^m_{enc}(\cdot)$ and $g_{ar}(\cdot)$ are in fact samples from a distribution denoted by $q(\mathbf{z}^m_t \mid \mathbf{z}^{m-1}_t)$, defined as a multivariate Gaussian with diagonal covariance matrix, as follows:

$$q(\cdot \mid \mathbf{z}^{m-1}_t) = \mathcal{N}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2)) \tag{3.1}$$

with $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ dependent on $\mathbf{z}^{m-1}_t$, specified in more detail in a following subsection. The outputs for $g^m_{enc}(\cdot)$ and $g_{ar}(\cdot)$ are obtained by sampling from this distribution, denoted respectively, as

follows:

$$\mathbf{z}_t^m \sim q(\cdot \mid \mathbf{z}_t^{m-1}) \tag{3.2}$$

$$\mathbf{c}_t \sim q(\cdot \mid \mathbf{z}_t^M) \tag{3.3}$$

Modules are thus stochastic and computing $g_{enc}^m(\mathbf{z}_t^{m-1})$ twice will likely result in two different representations of $\mathbf{z}_t^m$. This is in contrast to CPC and GIM's latent representations which remain fixed depending to the input [32, 34].

We achieve these stochastic modules by defining each module $g_{enc}^m(\cdot)$ consisting of two blocks. The first block receives as input $\mathbf{z}_t^{m-1}$ and predicts the parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, which fully describe the distribution $q(\cdot \mid \mathbf{z}_t^{m-1})$ defined in equation 3.1. The second block samples $\mathbf{z}_t^m \sim q(\cdot \mid \mathbf{z}_t^{m-1})$ from this distribution and produces an output representation. This is depicted in figure 3.1.

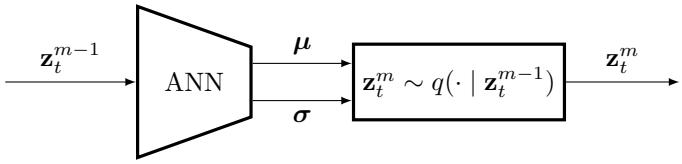


Figure 3.1: A single module.

In practice, sampling from q is achieved through a reparametrisation trick, as introduced in [26]. The equation to compute $\mathbf{z}_t^m$ then becomes:

$$\mathbf{z}_t^m = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$$

where $\boldsymbol{\epsilon}$ corresponds to a sampled value $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\odot$ is element-wise multiplication. The procedure to obtain $\mathbf{c}_t$ is analogous to $\mathbf{z}_t^m$.

3.3 The learning objective

Instead of training the neural network's modules end-to-end with a global loss function, each module is optimised greedily with its own personal loss function. Through the introduction of the novel *Variational-InfoNCE* loss, mutual information between temporally nearby representations is maximised, while regularising the latent space to be approximate to the standard Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$. The Variational-InfoNCE loss is defined as follows:

$$\mathcal{L}_{\text{V-NCE}}^m = \underbrace{\sum_k \mathop{\mathbb{E}}_{\substack{\mathbf{z}_{t+k}^m \sim q(\cdot \mid \mathbf{z}_{t+k}^{m-1}) \\ \mathbf{z}_t^m \sim q(\cdot \mid \mathbf{z}_t^{m-1})}} \left[\log \frac{f_k^m(\mathbf{z}_{t+k}^m, \mathbf{z}_t^m)}{\sum_{\mathbf{z}_j^m \in X} f_k^m(\mathbf{z}_j^m, \mathbf{z}_t^m)} \right]}_{\text{Maximise } I(\mathbf{z}_{t+k}^m, \mathbf{z}_t^m)} + \underbrace{\beta \; D_{KL}\left(q(\cdot \mid \mathbf{z}_t^{m-1}) \;||\; \mathcal{N}(\mathbf{0}, \mathbf{I})\right)}_{\text{Regularisation}} \tag{3.4}$$

Here, $m \in \mathbb{N}$ refers to the m'th module and $k \in \mathbb{N}$ the number of patches in the future the similarity score $f_k^m(\mathbf{z}_{t+k}^m, \mathbf{z}_t^m)$ must rate. The latent representations $\mathbf{z}_{t+k}^m$ and $\mathbf{z}_t^m$ are encoded samples produced by $g_{enc}^m(\mathbf{z}_{t+k}^{m-1})$ and $g_{enc}^m(\mathbf{z}_t^{m-1})$, respectively and X is a set of samples $\left\{\mathbf{z}_{t+k}^m, \mathbf{z}_1^m, \mathbf{z}_2^m, \dots\right\}$ where $\mathbf{z}_j^m$ with $j \neq t+k$ are random samples.

The similarity score $f_k^m(\cdot)$'s definition is identical to [34]:

$$f_k^m(\mathbf{z}_{t+k}^m, \mathbf{z}_t^m) = \exp({\mathbf{z}_{t+k}^m}^T W_k^m \mathbf{z}_t^m)$$

$\mathcal{L}_{\text{V-NCE}}^m$ consists of two terms. The first term ensures that latent representations of temporally nearby patches contain maximised mutual information. The second pushes the latent representations close to the origin. Finally, β is a hyper-parameter which decides the relative importance between the two terms. $\beta >> 1$ will weight more importance to regularisation, but may result in posterior collapse [31]. On the other hand $\beta \approx 0$ will attach more importance to the mutual information maximisation term while forgetting about the regularisation term. When $\beta = 0$, V-GIM is identical to GIM but with an altered neural network architecture which supports probabilistic latent representations.

3.3.1 Gradient

The first term in $\mathcal{L}_{\text{V-NCE}}^m$ can be approximated through mini-batches such that the approximation can be automatically differentiated using PyTorch [49]. With regards to the second term, since $q(\cdot \mid \mathbf{z}_t^{m-1})$ is a Gaussian defined by parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, a closed form solution exists [26], as follows.

$$D_{KL}\left(q(\cdot \mid \mathbf{z}_t^{m-1}) \,||\, \mathcal{N}(\mathbf{0}, \mathbf{I})\right) = \frac{1}{2}\sum_{k=1}^{D}\left(-\log\sigma_k^2 - 1 + \sigma_k^2 + \mu_k^2\right) \tag{3.5}$$

As such, this term can be directly computed and does not need to be approximated through mini-batches.

3.3.2 Properties of the latent space

In this section, we present two important claims regarding the structure of the latent space defined by each of V-GIM's modules. For each module $g_{enc}^m(\cdot)$ (or analogously for $g_{ar}(\cdot)$), an input space $\mathcal{Z}^{m-1}$ is mapped to a latent space $\mathcal{Z}^m$ as follows:

$$g_{enc}^m : \mathcal{Z}^{m-1} \to \mathcal{Z}^m$$

Here, $\mathcal{Z}^0$ equals the feature space $\mathcal{X}$. The two claims hold true for every $\mathcal{Z}^m$ if and only if $\mathcal{L}_{\text{V-NCE}}$ is optimal. The two claims will serve as the main argument for why V-GIM's representations are interpretable, discussed in section 3.4. Meanwhile, traditional techniques such as CPC and GIM lack these interpretable benefits.

Claim 1: $\mathcal{Z}^m$ is uninterrupted and well-covered around the origin.

In V-GIM, a latent representation $\mathbf{z}_t^m \in \mathcal{Z}^m$ of a data point $\mathbf{z}_t^{m-1} \in \mathcal{Z}^{m-1}$ is a sample from the Gaussian distribution $q(\cdot \mid \mathbf{z}_t^{m-1})$. Thus, encoding the same $\mathbf{z}_t^{m-1}$ an infinite number of times results in an entire circular region (around a particular mean $\boldsymbol{\mu}$) in $\mathcal{Z}^m$ that is entirely covered by the latent representations corresponding to $\mathbf{z}_t^{m-1}$, without any interruptions in this region. This is different from GIM and CPC where a data point merely covers a single point of the latent space (and not an entire region). Furthermore, because each region should be close to the origin, regions are more likely to utilise the limited space efficiently around the origin, resulting in a lower chance of obtaining large holes between two regions from different data points. As a result, the latent representations from the dataset should cover the entire latent space around the origin, such that all latent representations around the origin have a corresponding data point that is similar to a data point from the dataset.

Claim 2: $\mathcal{L}_{\text{V-NCE}}$ enforces smooth and consistent transitions in the latent space with respect to mutual information of the original data points.

As we discussed in section 3.3, the first term in $\mathcal{L}_{\text{V-NCE}}$ maximises the mutual information $I(\mathbf{z}_t^m, \mathbf{z}_{t+k}^m)$, which was proven by Oord et al. in [32]. However, in addition, Löwe et al. argue that when modules are trained in a greedy setting using this loss, the mutual information between outputs of successive modules $I(\mathbf{z}_t^{m-1}, \mathbf{z}_t^m)$ is also maximised [34], and therefore also $I(\mathbf{x}_t, \mathbf{z}_t^m)$. Or equivalently, given an data point $\mathbf{x}_t^{(i)}$, the output produced by $g_{enc}^m(\dots g_{enc}^2(g_{enc}^1(\mathbf{x}_t))) = \mathbf{z}_t^m$ will ensure $I(\mathbf{x}_t^{(i)}, \mathbf{z}_t^m)$ to be maximum. This is an important observation. Now, since the latent representations corresponding to $\mathbf{x}_t^{(i)}$ cover an entire circular region (as discussed in Claim 1), the mutual information $I(\mathbf{x}_t^{(i)}, \mathbf{z}_t^m)$ for all the latent representations $\mathbf{z}_t^m$ in this region should be maximised.

Let us now consider a second data point $\mathbf{x}_t^{(j)}$ whose corresponding regions in $\mathcal{Z}^m$ overlap with $\mathbf{x}_t^{(i)}$'s region. Since regions are fairly large and the latent space is small, this is likely to happen. Due to the overlap between the regions, there exists a latent representation $\mathbf{z}_t^{m\prime}$ that corresponds to both $\mathbf{x}_t^{(i)}$ and $\mathbf{x}_t^{(j)}$. Since $\mathbf{z}_t^{m\prime}$ is lower dimensional and is defined such that $I(\mathbf{z}_t^{m\prime}, \mathbf{x}_t^{(i)})$ and $I(\mathbf{z}_t^{m\prime}, \mathbf{x}_t^{(j)})$ are both maximised, $\mathbf{z}_t^{m\prime}$ contains information that is common to both $\mathbf{x}_t^{(i)}$ and $\mathbf{x}_t^{(j)}$ (if not, $\mathbf{z}_t^{m\prime}$ would not have been in the two regions).

If small changes to the components of $\mathbf{z}_t^{m\prime}$ caused abrupt changes in the corresponding feature vectors in $\mathcal{X}$, this would be heavily penalised by $\mathcal{L}_{\text{V-NCE}}$ due to the width of the Gaussian regions. Therefore, the transitions in the latent space must be smooth to ensure that small changes in $\mathbf{z}_t^{m\prime}$ lead to small changes in the corresponding feature vectors in $\mathcal{X}$.

Furthermore, due to the definition of q with covariance matrix containing only non zero values on the diagonal, the components of q are independent. Therefore, moving a latent representation in a single direction of a particular dimension should cause the same effect on the mutual information with respect to the corresponding data point in $\mathcal{X}$ as moving it in a different dimension. This property ensures that the transitions in the latent space are consistent.

Finally, the result of these two claims is an uninterrupted latent space around the origin where moving a latent representation towards a particular dimension in the latent space causes smooth and consistent changes in the corresponding data point from the previous module's latent space. This crucial observation will serve as the main argument for why V-GIM's representations are interpretable, while traditional techniques such as CPC and GIM do not have these guarantees.

3.4 Computational benefits

Each module in V-GIM is greedily trained through the variational InfoNCE loss. This allows for decoupled, memory-efficient asynchronous distributed training and mitigates the vanishing gradient problem, which are benefits introduced by GIM [34]. However, V-GIM adds a constraint to the latent space produced by each module, leading to additional practical benefits. Since V-GIM adopts a greedy training approach, these benefits apply not only to the final module's output but also to the outputs of intermediate modules.

[TODO] If time permits, go deeper in benefits introduced by GIM.

Interpretability of final and intermediate modules

By regularising the latent space resulting in the properties discussed in section 3.3.2, each of V-GIM's modules define a space where sampling from a point around the origin is likely to correspond to a data point that is similar to the dataset. A decoder trained on this space will

generalise well to unseen data when given a latent representations around the origin. This has significant implications to V-GIM's interpretability. Not only can we assess the information contained in a latent representation through a decoder, we can also attempt to understand the underlying structure of the latent representation. This is achieved in a similar way as with VAEs, by altering a one unit component of a latent representation at a time and observing the effect through the decoder.

Furthermore, since V-GIM's neural network architecture consists of a variable number of modules, each with their own interpretable latent space, these benefits are applicable to the latent representations produced by intermediate modules, allowing us to observe the internal mechanism of the neural network. Additionally, due to the CNN working, different modules encode different sequence lengths, encouraging different abstractions to be learned at different levels. By having intermediate modules be interpretable, we can analyse these abstractions as well. This is in contrast to VAEs, where only the output latent representation is interpretable, and intermediate representations in the architecture are not.

In contrast, GIM and CPC do not impose constraints on the latent space. Although a decoder can still be trained on the latent representations of GIM and CPC, the latent space is highly unpredictable, and generating a meaningful output from a randomly sampled latent representation is not guaranteed. Moreover, attempting to decode an interpolated representation from two existing latent representations may not lead to a meaningful output, since the space the decoder was trained on may contain large gaps. This makes it challenging to interpret the underlying structure of the latent representations defined by CPC and GIM.

Disentanglement

In section 2.3.2 on β-VAEs and disentanglement, Higgins et al. argue that setting the prior $p(\mathbf{z})$ to an isotropic Gaussian encourages disentanglement in the encodings [30]. When encodings are fully disentangled, this results in each unit component from the encoding to capture a different feature from the original data. Since $\mathcal{L}_{\text{V-NCE}}$ enforces the latent space to be standard normal, this theorem is also applicable to V-GIM and choosing a large value for β in $\mathcal{L}_{\text{V-NCE}}$ applies more pressure for encodings to be disentangled further increasing interpretability.

Improved generalisation performance through representation variance

V-GIM's encodings are samples from a distribution, which means that a single patch of data $\mathbf{x}_t$ may have multiple encoded representations $\mathbf{z}_t^M$. For downstream tasks with very little labelled data, the variability in representations could serve as an internal data augmentation method, which may improve generalisation to unseen data for downstream tasks.

Internal batch normalisation mechanism

During training of ANNs, the weights of different layers undergo changes, which can lead to a problem known as "internal covariate shift" [44]. This shift causes the distributions of each layer's input to change over time, which in turn can negatively affect subsequent layers, slowing down the training process [50, 51]. This issue is typically addressed via batch normalisation [52, 50], a mechanism which normalises the activations of internal layers, allowing for a higher learning rate during training and thus accelerating the process.

Batch normalisation is especially important in a greedy setting such as GIM and V-GIM, where modules are independently trained in parallel. Without intermediate normalisation, subsequent modules may learn slower than previous modules, resulting in the subsequent modules not being able to catch up in time to the changes from previous modules made during training.

The result would be that modules may have to be trained with different learning rates, resulting in an additional hyperparameter that must be obtained for each module.

In contrast, V-GIM already contains an internal normalisation mechanism in-between modules. This is indirectly caused by regularising each module's latent space to the standard normal. This results in encodings with mean of zero and standard deviation of one, that are produced by every module. Finally, the normalised encodings can also be beneficial for downstream tasks. If normalisation is desired, computing the mean and standard deviation over a potentially very large dataset is not required, as this is already built in to the encodings.

Chapter 4

Experiments

In this chapter, we evaluate the effectiveness of the latent representations obtained from Variational Greedy InfoMax (V-GIM) in comparison to its non-variational counterpart, GIM, by training both models on sequential data from the audio domain. To assess the quality of the representations, we project them into a two-dimensional space using t-SNE and analyse the emergence of potential clusters. Moreover, we employ a linear classifier that takes these representations as input, and evaluate its accuracy to gain insights into these models their performance. To further examine the efficacy of these representations, we investigate their potential for downstream task generalisation by training the linear classifier on smaller subsets of the dataset and assessing the amount of labelled data required to achieve satisfactory performance. Finally, we delve into the underlying structure of V-GIM's representations by training a decoder on top of each of V-GIM's modules, providing insights into their composition.

4.1 Experimental details V-GIM

GIM and V-GIM are trained on raw speech signals of fixed length sampled at 16kHz. The dataset consists of 851 audio files and is randomly shuffled and split up into 80% training data (680 files) and 20% test data (171 files). Each audio file consists of a single spoken sound consisting of three consonants and three vowels, where the consonants and vowels alternate each other. Some examples are the sounds "gi-ga-bu" and "ba-bi-gu". The words consists of three syllables from the following list: ba, bi, bu, da, di, du, ga, gi and gu. All the sounds are spoken by the same person, at a constant and peaceful tone. We crop the files to a fixed length of 10240 samples, or 640 milliseconds, which is slightly longer than half a second.

GIM and V-GIM are trained on the same architecture, consisting of two encoder modules $g^1_{enc}(\cdot)$ and $g^2_{enc}(\cdot)$, no autoregressor module $g_{ar}(\cdot)$ is used. Modules are trained in parallel. Each module consists of solely convolution and pooling layers. Since the architecture contains no fully connected layers, the length of the audio files can therefore be variable during inference. The architecture details for the two modules are presented in tables 4.1 and 4.2. ReLu non-linearity is applied after each convolution, except in the last layer in each module. No batch normalisation is used. The number of hidden channels produced by each convolution and pooling layer remains constant, at 32 channels. In each module, data flows synchronously from one layer to the successive layer, except in the final two layers. These run in parallel and both take the same input from the preceding layer. This architecture choice is related to the parametrisation trick we discussed in section 3.2. One layer generates $\boldsymbol{\mu}$ and the other $\boldsymbol{\sigma}$ such that $\mathbf{z}^m_t$ can be computed via the reparametrisation trick. The architecture is visualised in figure 4.1. An input

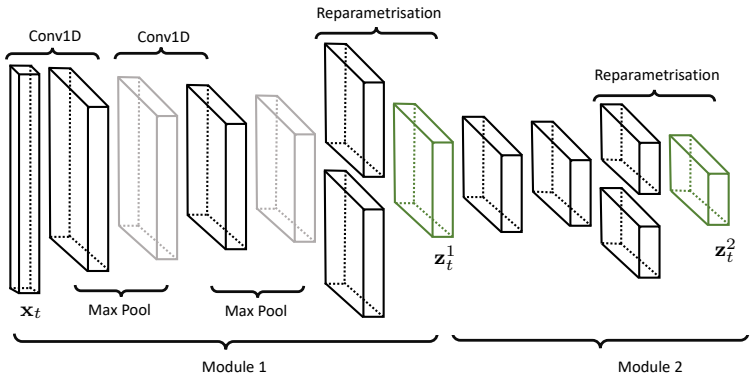


Figure 4.1: Visualisation of V-GIM's architecture.

signal of length 1×10240 is downsampled to 32×52 by the first module and to 32×13 by the second module. Each latent representation $\mathbf{z}_t^M = \mathbf{z}_t^2$ captures 49ms of speech.

Layer	**Kernel Size**	**Stride**	**Padding**
Conv1D	10	4	2
Max Pool	8	4	0
Conv1D	8	3	2
Max Pool	8	4	0
In parallel:			
Conv1D	3	1	2
Conv1D	3	1	2

Table 4.1: Module 1 architecture.

Layer	**Kernel Size**	**Stride**	**Padding**
Conv1D	6	2	2
Conv1D	6	2	2
In parallel:			
Conv1D	3	1	1
Conv1D	3	1	1

Table 4.2: Module 2 architecture.

Importantly, the reparametrisation trick is included in our implementation of GIM's architecture as well. However, by enforcing no constraints on the latent space, which is achieved by assigning $\beta = 0$ in $\mathcal{L}_{\text{V-NCE}}$, we observed that $\boldsymbol{\sigma}$ moves towards $\mathbf{0}$ such that all the randomness from sampling from a Gaussian distribution is removed. As such, our implementation of GIM is equivalent to GIM introduced in [34] but with an altered architecture.

Both GIM and V-GIM are trained using the Adam optimiser for 800 epochs with an exponential learning rate scheduler [53], with initial learning rate $lr_{init} = 0.01$ and $\gamma = 0.995$, such that lr is updated as follows:

$$lr_{epoch} = \gamma * lr_{epoch-1}$$

The batch size is set to 171, which is exactly the size of the test set. The number of patches to predict in the future k, is set to 12. Implementation details with regards to drawing negative samples for $f_k^m(\cdot)$ remain identical to the experiments from [32] and [34]. The regularisation importance term β is set to 0 for GIM and 0.0035 for V-GIM, which is the largest value we could obtain without causing posterior collapse.

After training V-GIM and GIM for 800 epochs, we train a linear multi-class classifier for 100 epochs on their respective latent representations for every module, evaluated using Cross-Entropy [54]. We set $lr = 0.001$ and use the Adam optimiser. The classifier is tasked to predict the syllable corresponding to its latent representation. Details on the algorithm used to split up audio files by individual syllables is provided in appendix A. Representations are average pooled

to a single time step, the number of channels remains unchanged at 32. The classifier thus consists of a single fully connected layer with 32 input nodes and 9 output nodes, conforming the number of input features and number of classes respectively. We reshuffle the dataset and use 80% for training and 20% for testing.

4.2 Results

4.2.1 Results GIM and V-GIM

Training error

Figure 4.2 displays the loss curves corresponding to GIM ($\beta = 0$) and V-GIM ($\beta = 0.0035$) for both modules. In the first module, the test loss for GIM is lower compared to V-GIM. This is as expected because V-GIM includes an additional regularisation term in its loss function. In the second module, both GIM and V-GIM have loss curves that converge to lower values than their previous module. This suggests that the task becomes easier for the subsequent modules, indicating that the preceding module does indeed make simplified representations for the speech data.

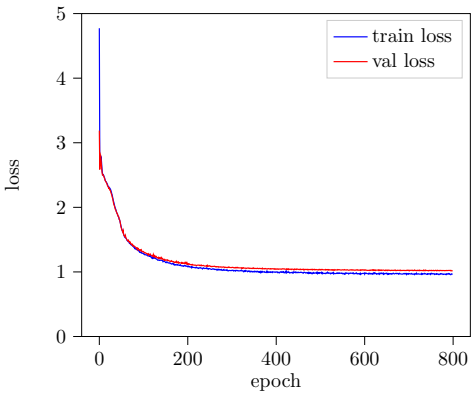
Interestingly, in the second module, V-GIM has a lower loss compared to GIM, even though V-GIM includes the regularization term in its loss function. We argue that this is related to internal covariate shift, which we discussed in section 3.4. The large learning rate from the previous module leads to a significant "drift" in the distribution of activations during training, making it difficult for GIM's successive modules to keep up with these changes. V-GIM is less susceptible to this issue due to the internal batch-normalisation mechanism that is built into the regularisation term. Consequently, V-GIM's modules can train with larger learning rates without affecting the performance of successive modules, resulting in faster convergence.

T-SNE

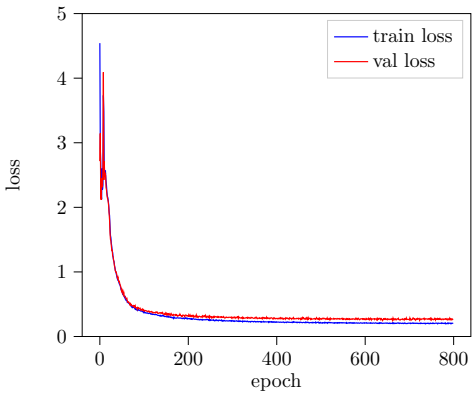
We project the latent representations obtained from each module to a two-dimensional plane using t-SNE [55]. This enables us to observe potential clusters, as similar data points are mapped close together, while dissimilar points remain far away. Similar to the classifier discussed in section 4.1, t-SNE is trained on flattened representations that are split into individual syllables, without performing any pooling. We run t-SNE with random initialisation, perplexity of 30 and a learning rate of 200 for 1000 iterations.

The graphs for GIM and V-GIM are displayed in figure 4.3. T-SNE was not provided any information about the class labels. Syllables containing the vowel "a" are represented with a red tint, "u" with green, and "i" with blue. We observe similar results in the first module of both GIM and V-GIM. T-SNE identifies two clusters, with one cluster corresponding to syllables containing an "a" and the other cluster to containing "u" or "i". Within the "u/i" cluster, there is still a grouping of "u" and "i" data points. However, distinguishing between "b", "d" or "g" is more challenging as data points within the clusters are entangled.

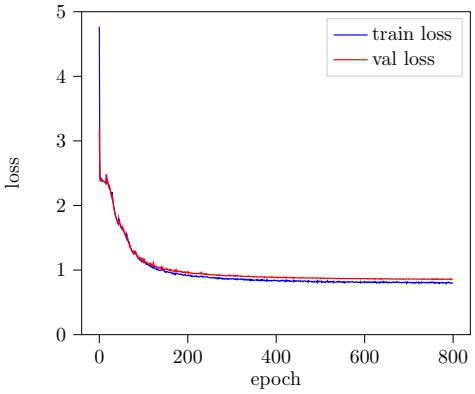
In the second module, we observe more significant differences between GIM and V-GIM. In V-GIM's second module, there is clearer separation between the "i" and "u" syllables, indicating that the second module contributes to improved latent representations. However, distinguishing between the pronounced consonant remains difficult. On the other hand, GIM's second module does not appear to have converged well, as t-SNE struggles to separate the representations into meaningful clusters. Consequently, syllables are mixed together without much structure. This further emphasises how GIM is affected by internal covariate shift, while V-GIM is not.



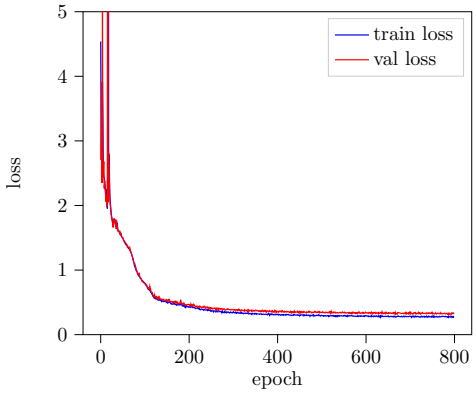
(a) Module 1: $\beta = 0.0035$



(b) Module 2: $\beta = 0.0035$



(c) Module 1: $\beta = 0$



(d) Module 2: $\beta = 0$

Figure 4.2: Training and validation loss (GIM: $\beta = 0$, V-GIM: $\beta = 0.0035$).

Classifier

Table 4.3 shows accuracies of the classifier. Results are in line with performance on t-SNE, klinkers are easily distuingished, but medeklinkers not so much.

Method	**Accuracy Module 1**	**Accuracy Module 2**
GIM	0.5	0.5
V-GIM	0.5	0.5

Table 4.3: Accuracies on test data

Distributions

Figure X, Y, Z depict the distributions of latent distributions corresponding to an individual dimension. Each figure depicts the distributions for a single module. We show distributions for the first 6 dimensions. We observe that V-GIM does indeed learn to to make distributions similar to a Gaussian, meanwhile in GIM we observe strong thin peaks, that the first module learned to frequently predict the same value with little noise due to $\boldsymbol{\sigma} \approx 0$. The peaks are less dominant in

the second module, which may be attributed to suboptimal convergence, which is attributed to suboptimal learning rate, as discussed in section **XX: prev paragraph**.

4.2.2 Generalisation study

help!!

4.3 V-GIM's Interpretability analysis

4.3.1 Decoders for Variational Greedy InfoMax

To investigate what information is contained in V-GIM's representations, we train a decoder on top of each of V-GIM's modules. Contrary to variational autoencoders, V-GIM is fully independent from the decoder and does not require one for training. Furthermore, we analyse the underlying structure the representations obtained from each module. This is achieved by altering representation's component values and observing the effects through the decoder. As we argued in the previous section, this is only possible because V-GIM's encodings is optimised to be approximate to the standard normal. As long as this is case, there are no "gaps" in the latent space around the origin. The decoder will be able to generalise to the altered representations as long as the representations are close to the origin.

We train a decoder for each of V-GIM's modules. As such we can assess the information contained in the final representation, but also the representations of intermediate modules.

timesteps in second module captures much wider time frame and first module, so observes different content. first module: each component respoble for different frequency.

second module: combination of frequencies.

Decoder architecture

We develop two decoders, one for each module.

$$\text{decoder}^1(\mathbf{z}_t^1) = \mathbf{x}_t$$

$$\text{decoder}^2(\mathbf{z}_t^2) = \mathbf{x}_t$$

Both decoders' architectures are symmetric to the architecture the V-GIM's encoder. Where the convolutional and max pooling layers are both replaced by transposed convolution layers. The architecture for

Layer	Filter size	Stride	Padding
TransConv1	3	1	1

The decoder consists of a convolutional neural network which takes as input the encodings from V-GIM and is tasked to reconstruct the original data. The decoder is optimised to minimise the mean squared error between mel spectrograms of the original data and the reconstructed data.

We optimise the decoder with the loss function introduced in [cite] and alter it to use mel spectrograms to better capture the important speech features according to the auditory system. The loss function we use is the following:

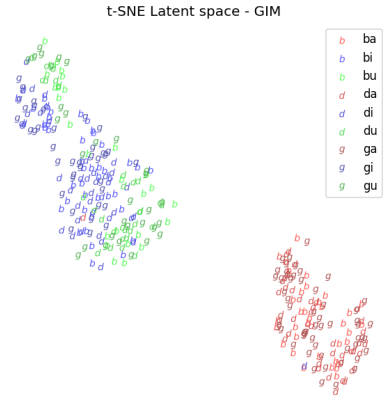
$$\mathcal{L}_{\text{decoder}} = \frac{1}{n}\sum_{i=1}^{n}\Big(\log(MEL(y^{(i)})) - \log(MEL(\hat{y}^{(i)}))\Big)^2$$

4.3.2 Decoder results

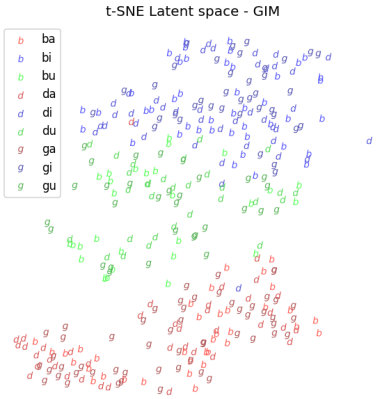
Loss function

4.3.3 Decoder: predictions on test set

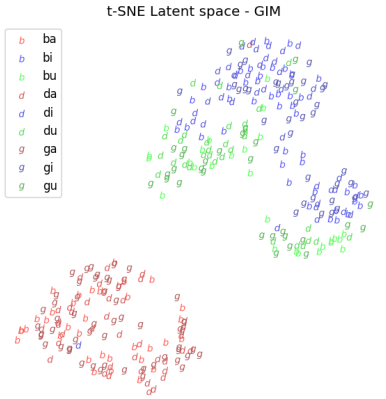
Fig **??** displays the reconstructed signal from the vocal sound "ba-gi-di". The two images on the left displays the original signal, while the right two images contain the reconstructed signal. The upper images displays the signals in time domain, the bottom images spectral domain. The reconstructed signal is an audio sample, for instance which is encoded via Greedy Infomax (up to the fourth (and final) convolution layer), this output is then given to a decoder to reconstruct the original signal.



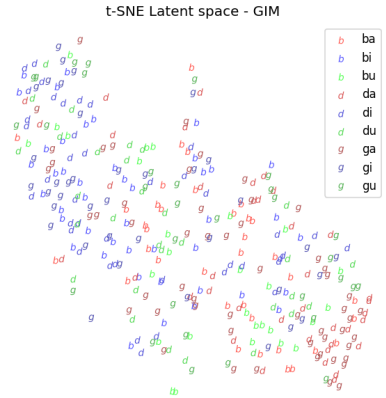
(a) Module 1: $\beta = 0.0035$



(b) Module 2: $\beta = 0.0035$



(c) Module 1: $\beta = 0$



(d) Module 2: $\beta = 0.0035$

Figure 4.3: T-SNE plots (GIM: $\beta = 0$, V-GIM: $\beta = 0.0035$).

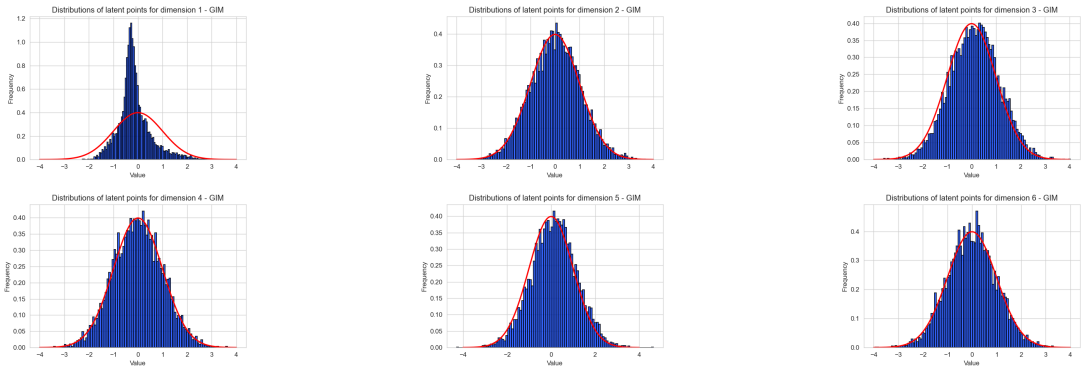


Figure 4.4: Module 1: $\beta = 0.0035$

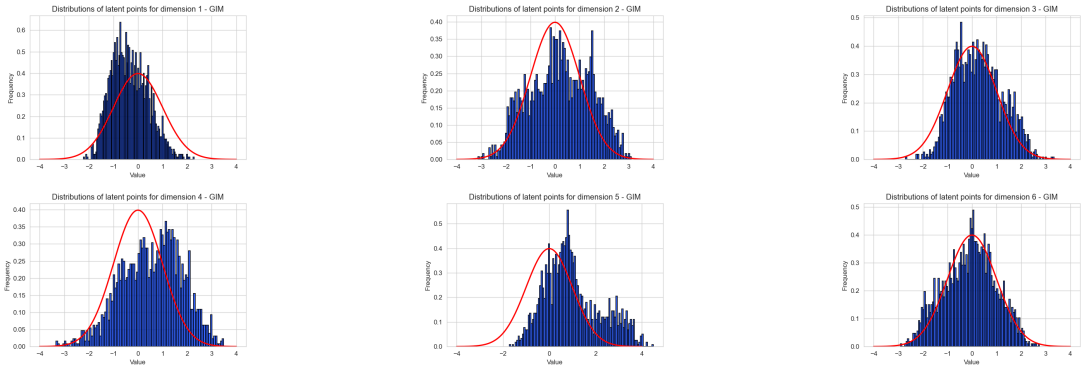


Figure 4.5: Module 2: $\beta = 0.0035$

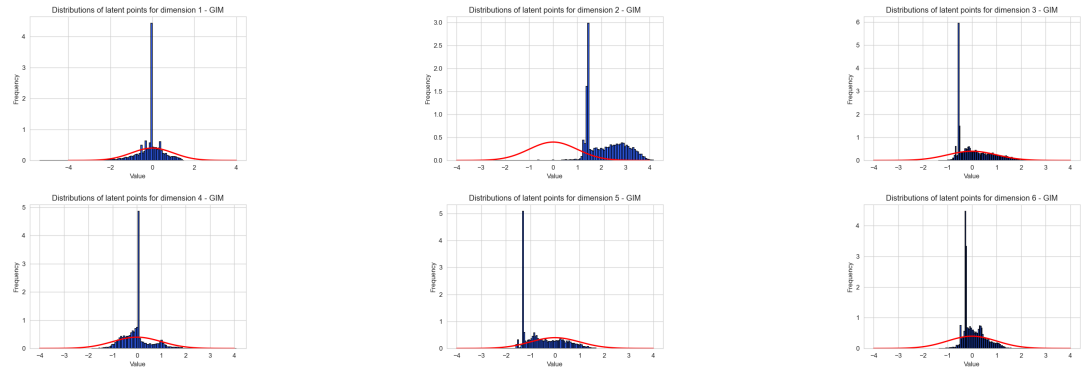


Figure 4.6: Module 1: $\beta = 0$

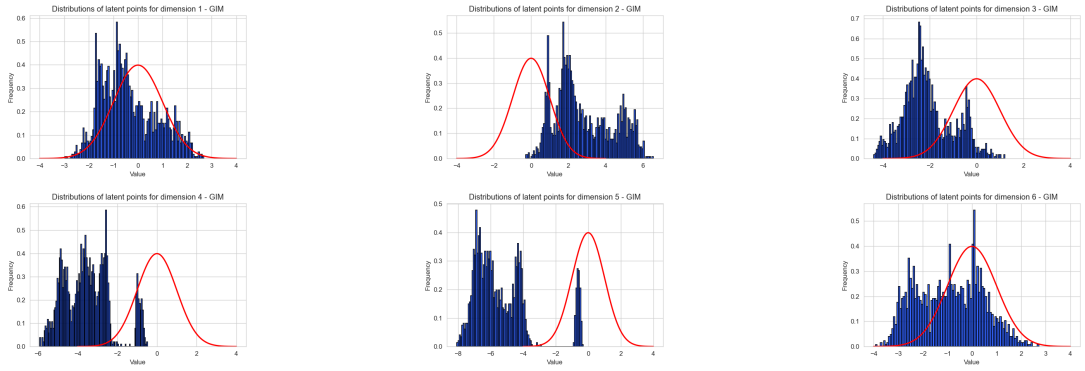


Figure 4.7: Module 2: $\beta = 0$

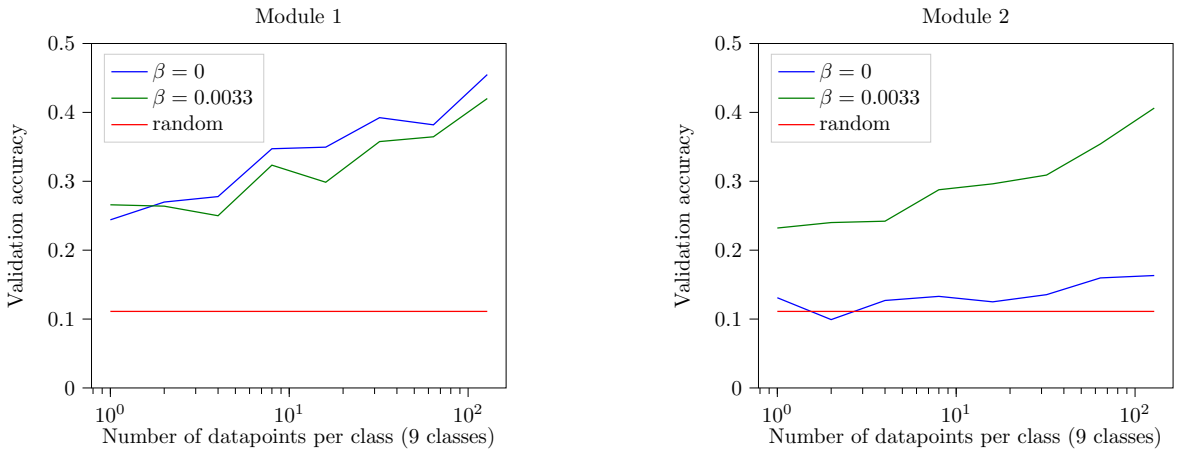


Figure 4.8: Validation accuracy for different subset sizes

Chapter 5

Related work

5.1 Representation learning: explainable

5.2 Variational learning

There already were a few papers with variational contrastive predictive coding

5.3 Links I should investigate

- Adversarial and Contrastive Variational Autoencoder for Sequential Recommendation https://arxiv.org/pdf/2103.10693.
- S3VAE: Self-Supervised Sequential VAE for Representation Disentanglement and Data Generation https://arxiv.org/abs/2005.11437
- Implementation of Sequential VAE https://github.com/ermongroup/Sequential-Variational-Autoencoder
- Contrastively Disentangled Sequential Variational Autoencoder https://proceedings.neurips.cc/paper/2021/file/53c5
Paper.pdf
- Sequential Variational Autoencoders for Collaborative Filtering https://arxiv.org/pdf/1811.09975.pdf
- !! Variational noise contrastive estimation: https://arxiv.org/abs/1810.08010

Chapter 6

Discussion

– decaying learing rate: we train using decaying lr, because models must first learn distributions and goes too slow if lr is too small. and a learning rate scheduler ExponentialLR decay rate 0.995

— batch norm: - sindy didn't have issues of batch norm, but believe this is because each module consisted of a single layer, ours contain a number of layers. potentially: outputs from first module change too fast for second module to catch up.

while GIM argues to resolve memory constraints, not entirely true. In fact we even countered the opposite as containing multiple neural networks, each with their own personal loss function (the loss function is based on fk which contains parameters that must be learned), and thus for early layers where the sequence is still long, a lot of memory is required. We went for a compromise on GIM by splitting up the architecture in merely two modules, significantly reducing the memory constraints.

— The second module in GIM clearly doesn't have as much effect. This can be explained because there may not be as much common information anymore between the patches. There may be a source that says that cpc learns low level features, but the second module is supposed to learn more high level features, which cpc may have trouble with? —

Future work: - Related work in VAE shows that gradually increasing regularisation term, results in better disentengledment, while avoiding posterior collapse. could have a kldweight scheduler.

- not constrained solely to InfoNCE loss, the GIM architecture could work for other losses too that allow for greedy optimisation.

- I didn't add an autoregressor as i didn't find a performance benefit. Potentially, with larger architecture could further improve performance.

—- Towards production setting: encodings are thus optimised to be close the standard normal. When in a production environment and new data is given, could in fact have an idea of how well generalisation to the production data: eg via anomaly detection if encodings are too far away from center. = gives automated way of verifying generalisation.

can then maybe see to which data that doesn't generalise well via outliers.

—-

future work: - disentanglement should do more investations

— GIM: Modular training could incrementally increase numb of modules and observe performance increase for downstream tasks. based on this, could find smallest gim architecture depth which satisfies required accuracies.

—- interpretability: most dimensions sensitive around 75 to 150 hz. this is as expected as the adult man speaks around 80 to 180 hz.

- Explainability of latents is dependent on the performance of the decoder.
- Intermediate loss function with kld resulted in similar behaviour as batch normalisation. Resulting in faster convergence than without kld.
- We observed no quality loss in the learned representations. Data was equally easily separable.

Bibliography

[1] T. M. Cover and J. A. Thomas, *ELEMENTS OF INFORMATION THEORY*.

[2] "Ali Ghodsi, Lec : Deep Learning, Variational Autoencoder, Oct 12 2017 [Lect 6.2]." [Online]. Available: https://www.youtube.com/watch?v=uaaqyVS9-rM

[3] B. Neyshabur, S. Bhojanapalli, D. Mcallester, and N. Srebro, "Exploring Generalization in Deep Learning," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/hash/10ce03a1ed01077e3e289f3e53c72813-Abstract.html

[4] Y. Chung, P. J. Haas, E. Upfal, and T. Kraska. Unknown Examples & Machine Learning Model Generalization. [Online]. Available: http://arxiv.org/abs/1808.08294

[5] P. Barbiero, G. Squillero, and A. Tonda. Modeling Generalization in Machine Learning: A Methodological and Computational Study. [Online]. Available: http://arxiv.org/abs/2006.15680

[6] J. Rust, "Using Randomization to Break the Curse of Dimensionality," vol. 65, no. 3, pp. 487–516. [Online]. Available: https://www.jstor.org/stable/2171751

[7] O. O. Aremu, D. Hyland-Wood, and P. R. McAree, "A machine learning approach to circumventing the curse of dimensionality in discontinuous time series machine data," vol. 195, p. 106706. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0951832019304752

[8] Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach, 4th US Ed.*, 4th ed. [Online]. Available: https://aima.cs.berkeley.edu/

[9] A. Jain, J. Mao, and K. Mohiuddin, "Artificial neural networks: A tutorial," vol. 29, no. 3, pp. 31–44.

[10] A. Krogh, "What are artificial neural networks?" vol. 26, no. 2, pp. 195–197. [Online]. Available: https://www.nature.com/articles/nbt1386

[11] Z. Zhang, "Artificial Neural Network," in *Multivariate Time Series Analysis in Climate and Environmental Research*, Z. Zhang, Ed. Springer International Publishing, pp. 1–35. [Online]. Available: https://doi.org/10.1007/978-3-319-67340-0_1

[12] Y. Bengio, A. Courville, and P. Vincent, "Representation Learning: A Review and New Perspectives," vol. 35, pp. 1798–1828.

[13] A. Zheng and A. Casari, *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists.* "O'Reilly Media, Inc.".

[14] A. Vali, S. Comai, and M. Matteucci, "Deep Learning for Land Use and Land Cover Classification Based on Hyperspectral and Multispectral Earth Observation Data: A Review," vol. 12, no. 15, p. 2495. [Online]. Available: https://www.mdpi.com/2072-4292/12/15/2495

[15] P. Przybyszewski, S. Dziewiatkowski, S. Jaszczur, M. Smiech, and M. Szczuka, "Use of domain knowledge and feature engineering in helping AI to play Hearthstone," in *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pp. 143–148.

[16] W. Wei, X. Hu, H. Liu, M. Zhou, and Y. Song, "Towards Integration of Domain Knowledge-Guided Feature Engineering and Deep Feature Learning in Surface Electromyography-Based Hand Movement Recognition," vol. 2021, p. e4454648. [Online]. Available: https://www.hindawi.com/journals/cin/2021/4454648/

[17] X. Zhang, Y. Xing, K. Sun, and Y. Guo, "OmiEmbed: A Unified Multi-Task Deep Learning Framework for Multi-Omics Data," vol. 13, no. 12, p. 3047. [Online]. Available: https://www.mdpi.com/2072-6694/13/12/3047

[18] P. H. Le-Khac, G. Healy, and A. F. Smeaton, "Contrastive Representation Learning: A Framework and Review," vol. 8, pp. 193 907–193 934.

[19] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines," vol. 9, no. 1, pp. 147–169. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0364021385800124

[20] D. Rumelhart, G. Hinton, and R. Williams, "Learning Internal Representations by Error Propagation," in *Readings in Cognitive Science*. Elsevier, pp. 399–421. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/B9781483214467500352

[21] D. Bank, N. Koenigstein, and R. Giryes. Autoencoders. [Online]. Available: http://arxiv.org/abs/2003.05991

[22] S. Karagiannakos. How to Generate Images using Autoencoders. AI Summer. [Online]. Available: https://theaisummer.com/Autoencoder/

[23] Papers with Code - MNIST Dataset. [Online]. Available: https://paperswithcode.com/dataset/mnist

[24] C. Doersch. Tutorial on Variational Autoencoders. [Online]. Available: http://arxiv.org/abs/1606.05908

[25] David Foster, "3. Variational Autoencoders," in *Generative Deep Learning, 2nd Edition*, 2nd ed. [Online]. Available: https://www.oreilly.com/library/view/generative-deep-learning/9781098134174/

[26] D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. [Online]. Available: http://arxiv.org/abs/1312.6114

[27] ——, "An Introduction to Variational Autoencoders," vol. 12, no. 4, pp. 307–392. [Online]. Available: http://arxiv.org/abs/1906.02691

[28] L. P. Cinelli, M. A. Marins, E. A. Barros da Silva, and S. L. Netto, *Variational Methods for Machine Learning with Applications to Deep Networks*, 1st ed. Springer International Publishing. [Online]. Available: https://link.springer.com/10.1007/978-3-030-70679-1

[29] Volodymyr Kuleshov and Stefano Ermon. The variational auto-encoder. [Online]. Available: https://ermongroup.github.io/cs228-notes/extras/vae/

[30] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "Beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework." [Online]. Available: https://openreview.net/forum?id=Sy2fzU9gl

[31] J. Lucas, G. Tucker, R. Grosse, and M. Norouzi, "Understanding Posterior Collapse in Generative Latent Variable Models." [Online]. Available: https://openreview.net/forum?id=r1xaVLUYuE

[32] p. d. u. family=Oord, given=Aaron, Y. Li, and O. Vinyals. Representation Learning with Contrastive Predictive Coding. [Online]. Available: http://arxiv.org/abs/1807.03748

[33] R. Shah, "[AN #92]: Learning good representations with contrastive predictive coding." [Online]. Available: https://www.lesswrong.com/posts/XE6LD2c9NtB7gMdEm/an-92-learning-good-representations-with-contrastive

[34] S. Löwe, P. O'Connor, and B. S. Veeling. Putting An End to End-to-End: Gradient-Isolated Learning of Representations. [Online]. Available: http://arxiv.org/abs/1905.11786

[35] S. S. Rao. Understanding the Gradient-Isolated Learning of Representations and intuition to the Greedy.... Analytics Vidhya. [Online]. Available: https://medium.com/analytics-vidhya/understanding-the-gradient-isolated-learning-of-representations-and-intuition-to-the-greedy-cb6c3598e317

[36] Z. Zhang and D. Tao, "Slow Feature Analysis for Human Action Recognition," vol. 34, no. 3, pp. 436–450.

[37] K. Stacke, C. Lundström, J. Unger, and G. Eilertsen, "Evaluation of Contrastive Predictive Coding for Histopathology Applications," in *Proceedings of the Machine Learning for Health NeurIPS Workshop*. PMLR, pp. 328–340. [Online]. Available: https://proceedings.mlr.press/v136/stacke20a.html

[38] p. u. family=Haan, given=Puck and S. Löwe. Contrastive Predictive Coding for Anomaly Detection. [Online]. Available: http://arxiv.org/abs/2107.07820

[39] M. Y. Lu, R. J. Chen, J. Wang, D. Dillon, and F. Mahmood. Semi-Supervised Histology Classification using Deep Multiple Instance Learning and Contrastive Predictive Coding. [Online]. Available: http://arxiv.org/abs/1910.10825

[40] S. Bhati, J. Villalba, P. Żelasko, L. Moro-Velazquez, and N. Dehak. Segmental Contrastive Predictive Coding for Unsupervised Word Segmentation. [Online]. Available: http://arxiv.org/abs/2106.02170

[41] S. Deldari, D. V. Smith, H. Xue, and F. D. Salim, "Time Series Change Point Detection with Self-Supervised Contrastive Predictive Coding," in *Proceedings of the Web Conference 2021*, ser. WWW '21. Association for Computing Machinery, pp. 3124–3135. [Online]. Available: https://dl.acm.org/doi/10.1145/3442381.3449903

[42] O. Henaff, "Data-Efficient Image Recognition with Contrastive Predictive Coding," in *Proceedings of the 37th International Conference on Machine Learning*. PMLR, pp. 4182–4192. [Online]. Available: https://proceedings.mlr.press/v119/henaff20a.html

[43] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[44] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. [Online]. Available: http://arxiv.org/abs/1502.03167

[45] A. H. Marblestone, G. Wayne, and K. P. Kording, "Toward an Integration of Deep Learning and Neuroscience," vol. 10. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fncom.2016.00094

[46] R. Wei and A. Mahmood, "Recent Advances in Variational Autoencoders With Representation Learning for Biomedical Informatics: A Survey," vol. 9, pp. 4939–4956.

[47] M. Tschannen, O. Bachem, and M. Lucic. Recent Advances in Autoencoder-Based Representation Learning. [Online]. Available: http://arxiv.org/abs/1812.05069

[48] M. Grossutti, J. D'Amico, J. Quintal, H. MacFarlane, A. Quirk, and J. R. Dutcher, "Deep Learning and Infrared Spectroscopy: Representation Learning with a Beta-Variational Autoencoder," vol. 13, no. 25, pp. 5787–5793. [Online]. Available: https://doi.org/10.1021/acs.jpclett.2c01328

[49] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch." [Online]. Available: https://openreview.net/forum?id=BJJsrmfCZ

[50] N. Bjorck, C. P. Gomes, B. Selman, and K. Q. Weinberger, "Understanding Batch Normalization," in *Advances in Neural Information Processing Systems*, vol. 31. Curran Associates, Inc. [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/36072923bfc3cf47745d704feb489480-Abstract.html

[51] Y. LeCun, L. Bottou, G. B. Orr, and K. R. Müller, "Efficient BackProp," in *Neural Networks: Tricks of the Trade*, ser. Lecture Notes in Computer Science, G. B. Orr and K.-R. Müller, Eds. Springer, pp. 9–50. [Online]. Available: https://doi.org/10.1007/3-540-49430-8

[52] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How Does Batch Normalization Help Optimization?" in *Advances in Neural Information Processing Systems*, vol. 31. Curran Associates, Inc. [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/905056c1ac1dad141560467e0a99e1cf-Abstract.html

[53] Bhargav Lad. Guide to Pytorch Learning Rate Scheduling. [Online]. Available: https://kaggle.com/code/isbhargav/guide-to-pytorch-learning-rate-scheduling

[54] Y. Ho and S. Wookey, "The Real-World-Weight Cross-Entropy Loss Function: Modeling the Costs of Mislabeling," vol. 8, pp. 4806–4813.

[55] p. d. u. family=Maaten, given=Laurens and G. Hinton, "Visualizing Data using t-SNE," vol. 9, no. 86, pp. 2579–2605. [Online]. Available: http://jmlr.org/papers/v9/vandermaaten08a.html

Appendices

Appendix A

Syllable classification through histogram segmentation

probabily bs: During inference, (in this context obtaining the latent representations for our input signals), depending on the length of the input signal, the length of the output latent representation will differ. If we wish to look at how separable latent representations are for syllables, the length can be variable. Some input sounds could be 6,600 samples, while others 8,800 samples. We therefore pad the syllables with zeroes in front and end of the signal, to obtain fixed length of equal to that of the longest syllable; 8,800 samples.

Training happens on longer data samples, and every **X** epochs t-SNE visualisations are made to observe evolutional of dis-entanglement.

Since the recordings are very consistent in loudness and are noise free, we can split up the files per syllable, obtaining three files per original sound (one for each syllable).

A sliding window is used of size 0.02 seconds. With a sample rate of 22050, this corresponds to roughly 500 samples per window. The maximum is computed for each window. Speech signals can then be split up when a severe dip happens in the signal. Regions where the amplitude is greater than 0.2 are considered **klinkers**, the regions with with lower values are considered **medeklinkers**. Apart from a few edge cases, this technique worked well enough for this purpose. In those cases, the splitting points closest to the one-third and two-third splitting points were considered.

ERR: OUDE AFBEELDINGEN ZIJN WEG

note: we do need a hard threshold which is based on the signal's intentisity level. One could consider the alternative approach of looking at the gradient at each point and selecting the points with largest negative gradient. This will work in many cases, however, not for temporal envelops which gradually move towards zero, s.t: A.1. Instead we use a dynamic threshold. This threshold is computed by creating transforming the signal into bins of 90'th percentile, creating a histogram of the single signal and applying otsu's image segmentation algorithm to obtain the threshold of that single audio sample. We also tried directly applying otsu to the moving average and maximum of the bins. This either gave a threshold that was too small or too large. the 90th percent resulted in an acceptable compromise.

Example where the explained strategy does not work:

Reference images for in the text:

audio padded to maximum length. (added zeros in front and back)

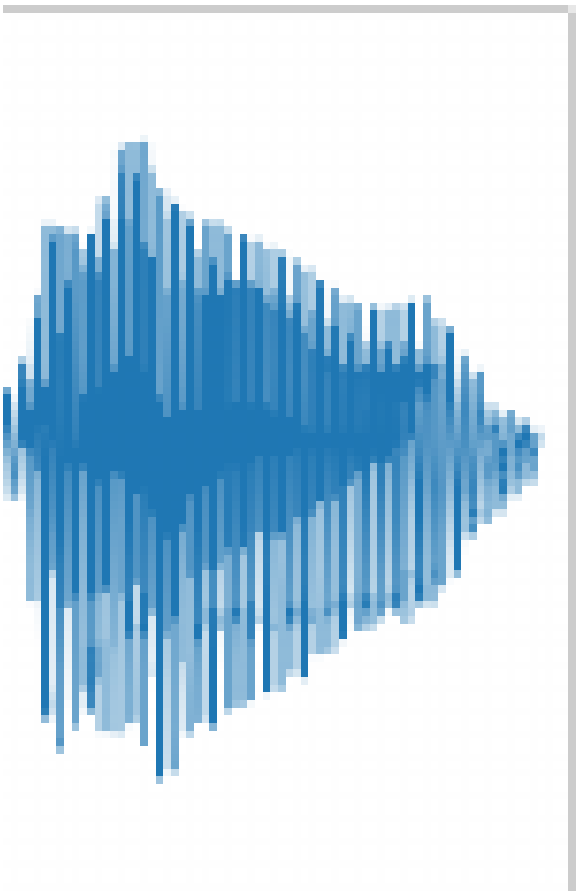


Figure A.1

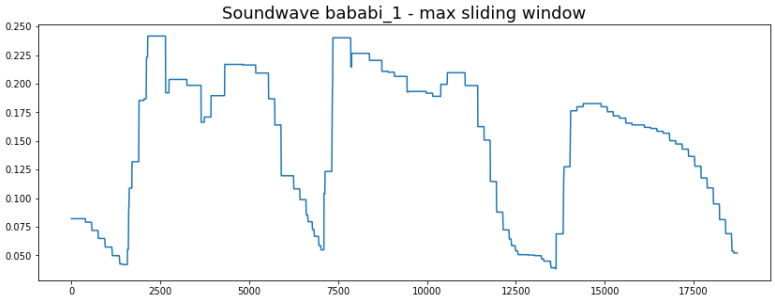


Figure A.2

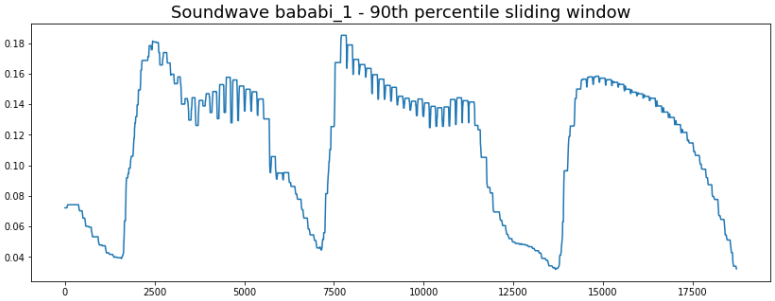


Figure A.3

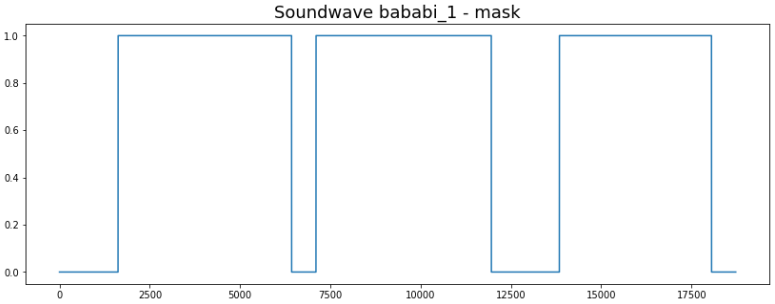


Figure A.4

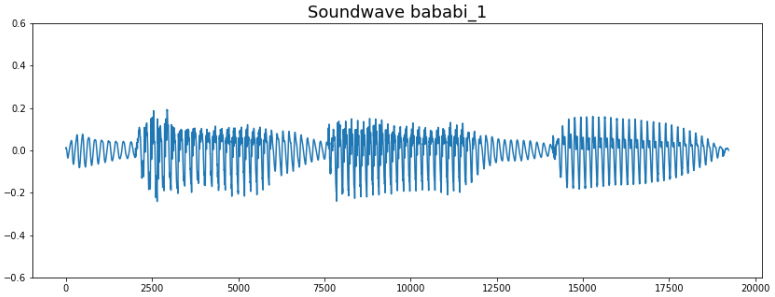


Figure A.5

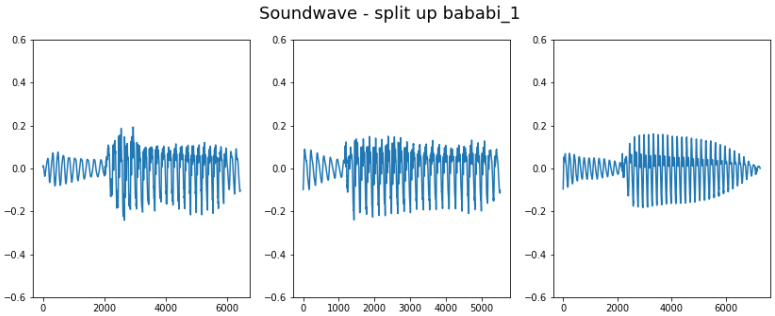


Figure A.6

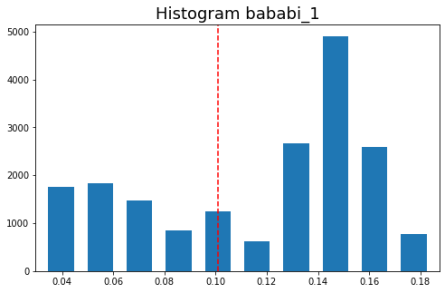

Histogram bababi_1
5000
4000
3000
2000
1000
0
0.04
0.06
0.08
0.10
0.12
0.14
0.16
0.18

Figure A.7

Appendix B

Some more appendix

B.1 GIM: Activations visualisations

thought for later: its actually weird i was able to play enc as audio as enc is 512 x something so huh? that means that a lot of info is already in first channel? what do other 511 channels then contain? """ Observations: First layer decoded still contains the same sound, but with some added noise (could be because decoder hasn't trained very). However, the encoded first layer, still contains the exact sound as the original sound. It is however downsampled a lot -¿ from 16khz to 3khz """ thought for later: its actually weird i was able to play enc as audio as enc is 512 x something so huh? that means that a lot of info is already in first channel? what do other 511 channels then contain?

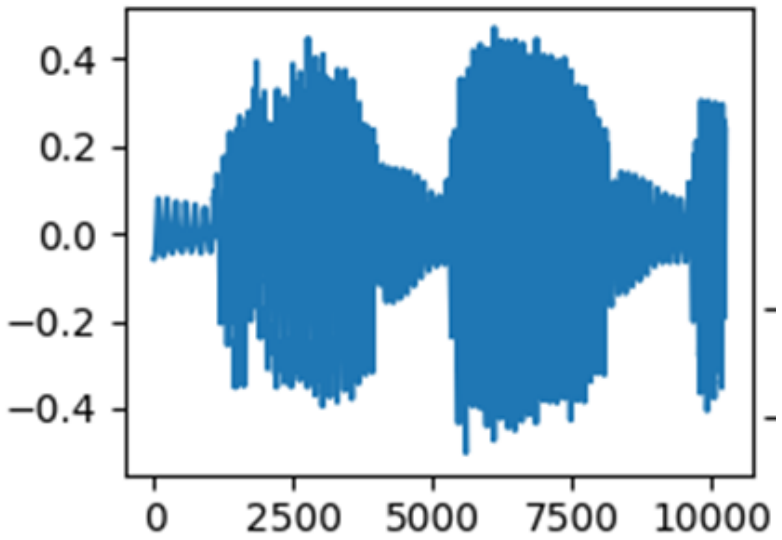


Figure B.1: "BA-BA-BA" time domain

No batch normalisation, so although channels appear to have larger activations than other channels, size of activation does not really say anything about information. eg activations 0.01 could still contain more information than 3.0 activation.

Figure B.2: Activations of the sound "BA-BA-BA" through GIM

Since the activations from convolutional neural networks, the order is still maintained. Hence, can align activations with original signal.

Observations in latent representations:

Layer 1: The activations of the first decoder still contain a lot of similarity with the original signal, in terms of structure. There is a lot of redundant data within the representation. Eg: the one channel could be replied

Layer 2

Layer 3:

Layer 4: Still notices multiple channels which have high activations when signal is has high amplitudes and small activations when amplitude is low.

Also activations which are high when volume is low. –¿ indicates that certain kernel weights are sensitive for **"klinkers"** and other kernels for **medeklinkers**. see B.3.

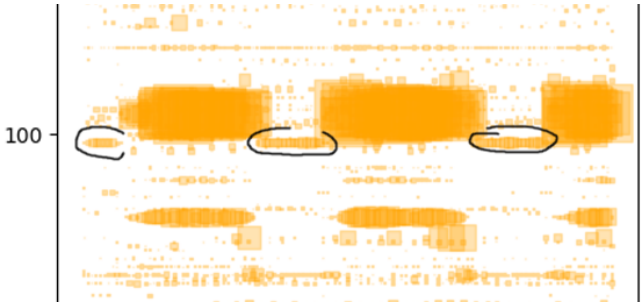


Figure B.3: zoomed in

Observe that activations happen in clusters/sequences. So it is usually a patch of signal samples that cause high activations. This could for instance indicate that both kernels are sensitive for the **medeklinker** "b", but sensitive for different features. eg the letter B has spoken sound "buh". so maybe one is sensitive for "b" and other for "uh".

Figure B.4 also nicely shows how different channels have clusters of activations at slightly different times.

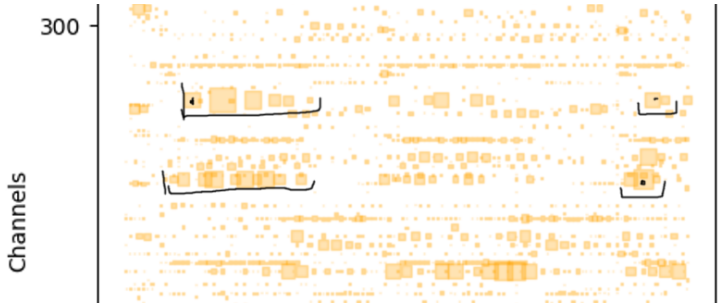


Figure B.4: Zoomed in