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 indepdenent and interpratable 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 indepdenent and interpratable 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

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 in equation 2.1 shown below.

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

The alphabet $\mathcal{X}$ 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 denoted in equation 2.2.

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

Hence, entropy can also be considered as the sum of "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 $-\inf$. 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.

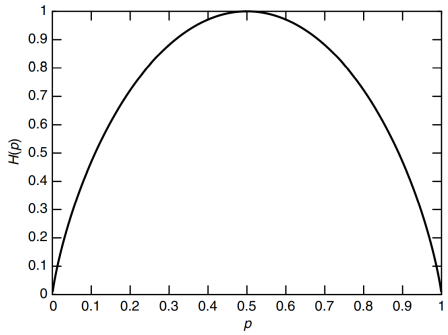

Figure 2.1: $H(p)$ vs p (originates from "Elements of Information Theory", page 16)

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 "divergence" or "distance" 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], I(X; Y) quantifies the amount of information Y describes about X. A alternative definition for $I(X;Y)$ is illustrated in 2.5. The equation provides us with an intuitive meaning for $I(X;Y)$, corresponding to the surprise caused by X, which is reduced by the knowledge of Y. In a following section, we discuss how these concepts from information theory are applied in representation learning, by maximising the mutual information between latent representations.

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

2.2 Supervised neural networks

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

Typical supervised machine learning problems consider the problem where given a training set of tuples (x_i, y_i), a mapping function $f : \mathcal{X} \to \mathcal{Y}$ must be learned, where $x_i \in \mathcal{X}$ corresponds to the input sample and $y_i \in \mathcal{Y}$ the label. A good f will then also generalise well to unseen $x_i \in \mathcal{X}$, which were not part of the training set.

Artificial Neural Networks (ANNs) in particular, tackle this problem by defining f as a fixed set of parameters consisting of a series of layers. During inference, at each layer l a transformation

matrix W^l is applied to the output vector from the previous layer a^{l-1}. This is shown in the equation below.

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

Secondly, a non-linear function $\sigma : R^d \rightarrow R^d$ is applied to z^l, as shown in equation 2.6. The resulting vector a^l may then again be the input for a following layer.

$$a^l = \sigma(z^l) \tag{2.6}$$

Hence, during inference, the input vector 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 in equation 2.7.

$$f_{W^1 \dots W^L}(x) = \hat{y}^L = \sigma(\dots \sigma(\sigma(x^T W^1)^T) W^2 \dots)^T W^L \tag{2.7}$$

During training the ANN's parameters W^1 ... 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, as shown in equation 2.8. y_i corresponds to the ground truth label of the single data sample x_i, $e(y_i, \hat{y}_i)$ the error between a ground truth y_i and the estimation $\hat{y}_i$ of a single sample. The error function e can be replaced depending on the task, for instance by mean squared error for regression or cross entropy for classification problems.

$$\mathcal{L}(W) = \frac{1}{n} \sum_{i=1}^{n} e(y_i, f_W(x_i)) \tag{2.8}$$

The parameters can be optimised by minimising the loss function $\mathcal{L}$ defined above. This can be done algebraically, however, it becomes difficult when the dimension of the input features is large. Gradient descent is used to find an "appropriate" minimum of the cost function, by iteratively adjusting the parameters W^1 ... W^L via the following learning rule.

$$W_{ij}^l \leftarrow W_{ij}^l - \alpha \frac{\partial \mathcal{L}}{\partial W_{ij}^l} \tag{2.9}$$

Although gradient descent can be successfully applied for finding local minima, neural networks may still contain many parameters which each must be optimised. This is typically resolved by applying the backpropagation algorithm in combination with gradient descent, such that fewer partial derives must be calculated, resulting in more efficient optimisation.

Although supervised learning though ANNs is considered successful, their performance is heavily dependent on the choice of the data representation [3], on which they are applied. For complex representations, a more complex architecture may be required, which requires more data to prevent overfitting. When labelled data is scarce, these models may result in seemingly well performance on the training set, but bad generalisation to data outside the training set. As a consequence, a lot of time in the machine learning pipeline is invested in transforming data into better latent representations. Good latent representations tend to disentangle the data into meaningful features, such that data can be more easily be separable into classes. In the following section we discuss how part of the labour of finding good representations can be relieved with unsupervised learning approaches, which may learn to find good latent representations. These representations could then be used as the input for supervised predictors.

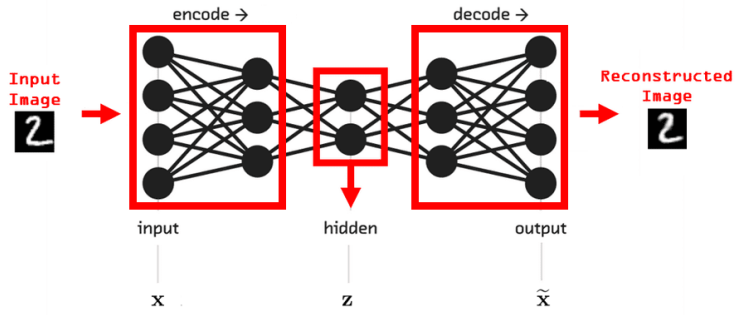

Figure 2.2: Autoencoder neural network architecture adapted from [7].

2.3 Representation learning through reconstruction error

One of the challenges in supervised learning is the constant need of large amounts of labelled data. Hence, when a labelled dataset is small, we would like to leverage a larger unlabelled dataset as basis for learning. By doing so, a mapping can be learned from the raw input data to a representation which makes downstream tasks easier. This process of learning representations from data is commonly referred to as representation learning [4]. Supervised learning algorithms can then learn directly from these disentangled latent representations with fewer labelled data.

In the following two sections we discuss two paradigms of representation learning with ANNs. The first paradigm is learning 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 section three.

2.3.1 Autoencoders

Autoencoders were introduced in 1986 by D. Rumelhart et al. [5] as a means to learn compressed representations [6]. This is achieved through a neural network architecture consisting of two blocks. The first block is the encoder E and receives input data which it encodes into a lower dimensional representation. The second block, called the decoder D, receives as input the latent representation and is tasked to reconstruct the original input. The two blocks are simultaneously optimised by *minimising the reconstruction error* shown in the following equation:

$$\mathcal{L} = \sum_{i=1}^{N} l(\mathbf{x}^{(i)}, D(E(\mathbf{x}^{(i)}))) \tag{2.10}$$

where l refers to the error for a single data point, for instance the l_2-norm and N the number data points. An example autoencoder architecture is depicted in figure 2.2. $\mathbf{x} \in \mathcal{D}$ corresponds to an input vector which is encoded into latent representation $\mathbf{z}$. The decoding of $\mathbf{z}$ corresponds to $\tilde{\mathbf{x}} = D(E(\mathbf{x}))$. The dimension of $\mathbf{z}$ is typically bottlenecked to be 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 [6]. $\mathbf{z}$ may then for instance be used for downstream tasks such as classification or clustering.

While capable of learning compressed representations, autoencoders do not pose any restrictions on the latent space they define (the space of latent representations). As a result, the representations may be meaningful to computers, but non-interpretable to humans. For instance, given the left image depicted in figure 2.3 which depicts an autoencoder's two dimensional latent

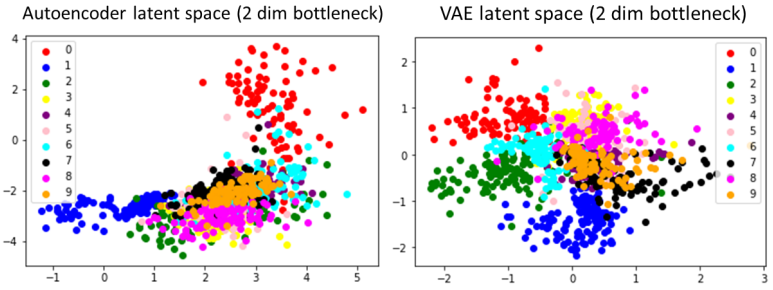

Figure 2.3: Both images represent a the two-dimensional latent space, learned from the MNIST dataset [8]. This is a dataset consisting of images of handwritten numbers between 0 and 9. Each image is associated to a class label, referring to the number. The left image consists of the space learned from a classical autoencoder, the right of a variational autoencoder (VAE). Both autoencoders have not received any explicit information of the labels of the dataset, yet learned to define representations which equal numbers are closer to each other. The VAE's latent space, depicted in the right image, is optimised to be standard normally distributed. The latent vectors $\mathbf{z}$ are distributed according to the two-dimensional normal distribution $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, where $\boldsymbol{\mu}$ is the two dimensional zero vector. $\mathbf{I}$ is the 2×2 covariance matrix with ones on the diagonal and zeroes elsewhere.

space, it is very difficult to know what the resulting images would be when interpolating between the latent representations of 0 (red) and 1 (blue). Answering this question becomes even more infeasible for higher dimensional latent vectors.

2.3.2 Variational autoencoders

Similar to traditional autoencoders, variational autoencoders (VAE) learn representations that contain the important information that is necessary to reconstruct the data, *however*, an additional constraint is applied to the latent space [9, 10, 11, 12]. The data points from the latent space are defined in such a way that they conform to a certain distribution, typically, the standard normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$ [10]. This behaviour can be observed in the right plot of figure 2.3. Since the latent representations conform a two-dimensional standard normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$, the data samples are more likely to be near the center at (0, 0).

Simulating distributions through neural networks

In variational autoencoders the latent representations of a data point $\mathbf{x}^{(i)}$ does not simply consist of a fixed deterministic vector $\mathbf{z}^{(i)}$, as was the case in the traditional autoencoder. Instead, given $\mathbf{x}^{(i)}$, its latent representation corresponds the following distribution $q_\phi(\mathbf{z} \mid \mathbf{x}^{(i)})$. A concrete vector $\mathbf{z}^{(i)}$ can be obtained taking a sample $\mathbf{z}^{(i)} \sim q_\phi(\mathbf{z} \mid \mathbf{x}^{(i)})$. As result a single $\mathbf{x}^{(i)}$ will correspond to multiple latent vectors $\mathbf{z}^{(i)}$. The latent representation $q_\phi(\mathbf{z} \mid \mathbf{x}^{(i)})$ is modelled as $\mathcal{N}(\boldsymbol{\mu}^{(i)}, \text{diag}(\boldsymbol{\sigma}^{(i)}))$. This means that all the required information to model $q_\phi(\mathbf{z} \mid \mathbf{x}^{(i)})$ is a mean vector $\boldsymbol{\mu}^{(i)}$ and a covariance matrix $\text{diag}(\boldsymbol{\sigma}^{(i)})$. Thus, given $\mathbf{x}^{(i)}$, a deterministic neural network can simulate $\text{diag}(\boldsymbol{\sigma}^{(i)})$ by predicting two vectors, namely, $\boldsymbol{\mu}^{(i)}$ and $\boldsymbol{\sigma}^{(i)}$. The latent vectors $\mathbf{z}^{(i)}$ can then be obtained by randomly sampling from the distribution. And thus, multiple $z^{(i)}$'s may correspond to a single $\mathbf{x}^{(i)}$. This method is depicted in figure 2.4. Finally, a sample $\mathbf{z}^{(i)}$ can be obtained as follows:

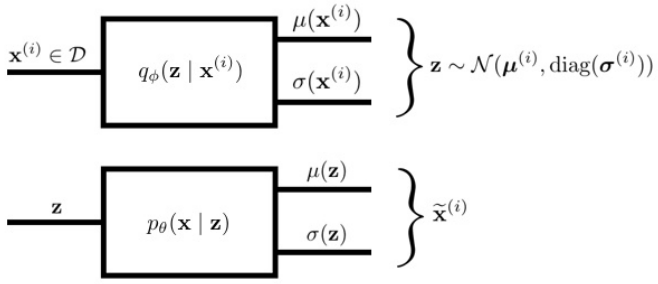

Figure 2.4: High level view of a variational autoencoder, depicting how a data point $\mathbf{x}^{(i)}$ is encoded into a latent distribution and reconstructed as $\widetilde{\mathbf{x}}^{(i)}$. Both blocks depict a neural network. The upper block is the encoder and the lower block the decoder. The upper block receives a data points $\mathbf{x}$ and produces the parameters of $q_\phi(\mathbf{z} \mid \mathbf{x}^{(i)})$. Since we choose to model $q_\phi(\mathbf{z} \mid \mathbf{x}^{(i)})$ as a Gaussian with independent components, the covariance matrix $\boldsymbol{\Sigma}$ is zero everywhere except for the diagonal. This way the diagonal values, representing the standard deviations, can be represented via a single vector $\boldsymbol{\sigma}^{(i)}$. The vectors $\boldsymbol{\mu}^{(i)}$ and $\boldsymbol{\sigma}^{(i)}$ are $\mu(\mathbf{x}^{(i)})$ and $\sigma(\mathbf{x}^{(i)})$, respectively. These are the output of the encoder block and form the parameters for $q_\phi(\mathbf{z} \mid \mathbf{x}^{(i)})$. A single neural network with parameter weights ϕ is used to simulate $q_\phi(\mathbf{z} \mid \mathbf{x}^{(i)})$ for every $\mathbf{x}^{(i)} \in \mathcal{D}$. This strategy of sharing ϕ across data points is refered to as "amortised variational inference" [12].

$$\mathbf{z}^{(i)} = \boldsymbol{\mu}^{(i)} + \boldsymbol{\sigma}^{(i)} \odot \boldsymbol{\epsilon}^{(i)} \tag{2.11}$$

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 [10].

The learning objective

So far we have discussed how predictions of a neural network can emulate predicting a distribution $q_\phi(\mathbf{z} \mid \mathbf{x}^{(i)})$, by predicting the distribution's parameters $\boldsymbol{\mu}^{(i)}$ and $\boldsymbol{\sigma}^{(i)}$. However, no constraints have been set on the quality of the distributions. We will discuss this now. The representations are optimised to minimise two measurements: one, the reconstruction error, and secondly, the distance from the latent distributions to $\mathcal{N}(\mathbf{0}, \mathbf{I})$.

The loss function to be optimised for a single data point $\mathbf{x}^{(i)}$ is shown in the equation below.

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}^{(i)}) = \mathbb{E}_{\mathbf{z} \sim q_\phi(\cdot \mid \mathbf{x}^{(i)})} \Big[-\log p_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}) \Big] + D_{KL} \Big(q_\phi(\cdot \mid \mathbf{x}^{(i)}) \mid\mid p_\theta(\cdot) \Big) \tag{2.12}$$

Although the equation 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.

$$\mathbb{E}_{\mathbf{z} \sim q_\phi(\cdot \mid \mathbf{x}^{(i)})} \Big[-\log p_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}) \Big]$$

The distribution $p_\theta(\mathbf{x}^{(i)} \mid \mathbf{z})$ is a distribution over $\mathbf{x}^{(i)}$ where $\mathbf{z}$ is instantiated. What comes out are thus probabilities. Notice $\mathbf{z}$ is sampled from the Gaussian distribution $z \sim q_\phi(\mathbf{z} \mid \mathbf{x}^{(i)})$.

So the $\mathbf{z}$'s close to the mean $\mu(\mathbf{x}^{(i)}) = \boldsymbol{\mu}^{(i)}$ are more likely to be sampled than those further away. So for these $\mathbf{z}$'s, we'd like the probability of corresponding to the actual $\mathbf{x}^{(i)}$ to be high. 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.

Optimising the second term of equation 2.12 poses the constraints on the latent distributions $q_\phi(\mathbf{z} \mid \mathbf{x}^{(i)})$. Again, this metric should be minimised. As we discussed in the chapter on Entropy, the KL divergence can be considered as a (non-symmetric) distance measure between two distributions. Hence, this value is small when the two distributions are similar. As we discussed earlier $p_\theta(\mathbf{Z})$ is often replaced by $\mathcal{N}(\mathbf{0}, \mathbf{I})$ as shown below. Minimising this equation will result in moving each distribution $q_\phi(\mathbf{z} \mid \mathbf{x}^{(i)})$, corresponding to a value $\mathbf{x}^{(i)}$, close to $\mathcal{N}(\mathbf{0}, \mathbf{I})$.

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

One can algebraically prove, that optimising this equation is equivalent to optimising the following equation [11]:

$$D_{KL}\left(\mathcal{N}(\boldsymbol{\mu}^{(i)}, \text{diag}(\boldsymbol{\sigma}^{(i)})) \mid\mid \mathcal{N}(\mathbf{0}, \mathbf{I})\right) = \frac{1}{2}\sum_{k=1}^{D}\left(-\log(\sigma_k^{(i)})^2 - 1 + (\sigma_k^{(i)})^2 + (\mu_k^{(i)})^2\right) \tag{2.13}$$

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

[TODO] Relation with variational inference / derivation of ELBO loss

$$\begin{aligned} -\mathcal{L}_{ELBO} &= -\mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot\mid\mathbf{x}^{(i)})}\left[\log p(\mathbf{Z}, \mathbf{x}^{(i)}) - \log q(\mathbf{z} \mid \mathbf{x}^{(i)})\right] \\ &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\cdot\mid\mathbf{x}^{(i)})}\left[-\log p_\theta(\mathbf{x}^{(i)} \mid \mathbf{z})\right] + D_{KL}\left(q_\phi(\cdot \mid \mathbf{x}^{(i)}) \mid\mid p_\theta(\cdot)\right) \end{aligned} \tag{2.14}$$

Where the first term is reconstruction error and second is regularisation. Hence minimising the KL divergence between the two posteriors is equivalent to minimising the divergence between the approximate posterior $q(\mathbf{z} \mid \mathbf{x}^{(i)})$ and the **marginal or prior?** $p_\theta(\mathbf{Z})$.

$p(z)$ in the equation is usually chosen to be the standard normal distribution $\mathcal{N}(0, I)$, such that a closed form solution exists. As such, when $p(z)$ corresponds to the standard normal, and $q(\mathbf{z} \mid \mathbf{x}^{(i)})$ is a multidimensional Gaussian with mean vector $\boldsymbol{\mu}^{(i)}$ and covariance matrix with independent dimensions, such that the diagonal corresponds of a vector standard devisations $\boldsymbol{\sigma}^{(i)}$, then the KL divergence has the following closed form:

$$D_{KL}\left(\mathcal{N}(\boldsymbol{\mu}^{(i)}, \text{diag}(\boldsymbol{\sigma}^{(i)})) \mid\mid \mathcal{N}(\mathbf{0}, \mathbf{I})\right) = \frac{1}{2}\sum_{k=1}^{D}\left(-\log(\sigma_k^{(i)})^2 - 1 + (\sigma_k^{(i)})^2 + (\mu_k^{(i)})^2\right) \tag{2.15}$$

For the equation above, gradients can easily be back propagated through machine learning libraries such as Tensor Flow. We still require a method for computing the gradient of the first term in $\mathcal{L}_{ELBO}$.

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 [13]. 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, compressing patches of the sequence without considering the relation with nearby patches, will result in lost information as the context between patches is not encoded into the representation.

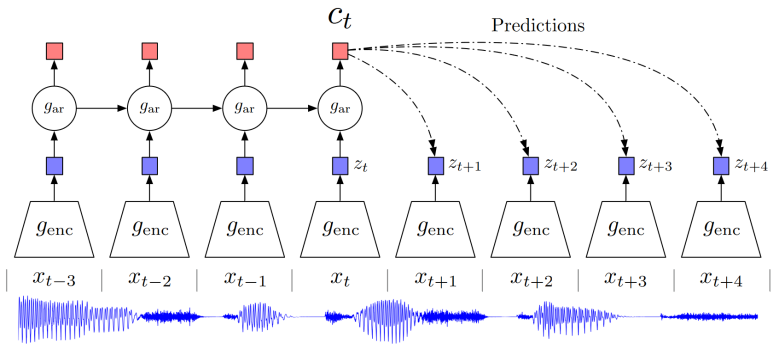

Figure 2.5: Overview of Contrastive Predictive Coding, originates from [13]

CPC deals with these context issues by maximising the shared information between the extracted representations of temporally nearby patches [14]. We will discuss this concept in more detail in a following section. For now, we would like to draw the readers attention to figure 2.5. 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 $\mathbf{x}_1 ... \mathbf{x}_n$ 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 fully connected ANN. The latent representations $\mathbf{z}_1 .. \mathbf{z}_n$ are obtained independently from each other and do not yet contain any information on an historic context. This historic context is achieved through $g_{ar}(\cdot)$, an auto-regressor, which encodes all previous $\mathbf{z}_1 ... \mathbf{z}_t$ into a single representation $\mathbf{c}_t$:

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

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 [13]. As shown in figure 2.5, the encodings from sequential data of undefined length, may correspond a series of latent representations $\mathbf{c}_1, \mathbf{c}_2, ...$

or $\mathbf{z}_1, \mathbf{z}_2, ...$ 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 not unlikely to believe that the conveyed information at time step t and $t + k$ contains some redundant information, such as pitch, frequency, tone, etc. [15]. 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" [16]. 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 [13].

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 [14]. It is defined as a log bilinear model as follows:

$$f_k(\mathbf{z}_j, \mathbf{c}_t) = \exp(\mathbf{z}_j^T W_k \mathbf{c}_t) \tag{2.16}$$

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 $f_k(\mathbf{z}_{t+1}, \mathbf{c}_{t+1})$.

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

$$\mathcal{L}_N = -\sum_k \mathbb{E}_{\mathrm{X}} \left[\log \frac{f_k(\mathbf{z}_{t+k}, \mathbf{c}_t)}{\sum_{\mathbf{z}_j \in X} f_k(\mathbf{z}_j, \mathbf{c}_t)} \right] \tag{2.17}$$

where X corresponds to the set $\{\mathbf{z}_{t+k}, \mathbf{z}_1, \mathbf{z}_2, ...\}$. Notice that there exists exactly one $\mathbf{z}_{t+k} \in X$, which corresponds to a positive sample and all other $\mathbf{z}_j \in X$ correspond to negative samples (with respect to the context $\mathbf{c}_t$). Hence good representations for $\mathbf{z}, \mathbf{c}$ and $f_k(\cdot)$, will result in a large similarity score for positive samples and approximate 0 for negative samples, resulting in a minimum fraction equal to 1. This would then be cancelled out by the $\log(\cdot)$ function. Meanwhile, $\mathcal{L}_n$ is large when the denomitator is large, indicating in a large high similarity score for negative samples.

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. proof that optimising InfoNCE is equivalent to maximising the mutual information between $\mathbf{c}_t$ and $\mathbf{z}_{t+1}$ [13].

$$I(\mathbf{z}_{t+1}; \mathbf{c}_t) = \sum_{\mathbf{z}_{t+1}, \mathbf{c}_t} p(\mathbf{z}_{t+1}, \mathbf{c}_t) \log \frac{p(\mathbf{z}_{t+1} \mid \mathbf{c}_{t+1})}{p(\mathbf{z}_{t+1})} \tag{2.18}$$

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 InfoceNCE 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 [13]. 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 $\{\mathbf{z}_{t+k}, \mathbf{z}_1, \mathbf{z}_2, ...\}$ 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.

$$f_k(\mathbf{z}_{t+k}, \mathbf{c}_t) \propto \frac{p(\mathbf{z}_{t+k} \mid \mathbf{c}_t)}{p(\mathbf{z}_{t+k})} \tag{2.19}$$

Oord et al. utilise this proportionality relation to reformulate $-\mathcal{L}_N$ as a lower bound on the mutual information between $\mathbf{z}_{t+1}$ and $\mathbf{c}_t$ as follows [14, 13]:

$$I(\mathbf{z}_{t+1}; \mathbf{c}_t) \geq \log(N) - \mathcal{L}_N \tag{2.20}$$

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

TODO: if time permits: compare against VAE, so doesn't have a decoder layer so simpler architecture + allows for sequential data.

2.4.2 Greedy InfoMax

So far we discussed how CPC encodes latent representations by maximising the mutual information between temporally nearby patches of data. This method has shown great success in recent years and is considered state of the art in self-supervised learning for encoding sequential data [17]. Additionally, CPC has been successfully applied to multiple use cases [17, 18, 19, 20, 21, 22]. This is achieved by minimising the InfoNCE loss discussed earlier in equation 2.17. Through this *global* loss function all parameters are optimised end-to-end via backpropagation. Although, backpropagation is considered effective, it still suffers from multiple constraints. Löwe et al. group the constraints in two categories: biological and computational.

Biological constraints of backpropagation [TODO]

- **Local Error Representation, Weight Symmetry, ...** its performance is still uncompanionable to

When comparing its performance against how humans learn, humans can still learn much faster from only a few examples...

The human brain does not appear to have a global objective function, which is optimized by backpropagating an error signal [23]. In the following section we discuss the changes Löwe et al. have made to make CPC more biologically plausible.

Computational constraints of backpropagation [TODO]

- - memory overhead, synchronous training, vanishing gradients.

Löwe et al. work further on the representation learning method proposed in CPC by taking a biologically inspired approach. This is done via the introduction of Greedy InfoMax (GIM), an extension on CPC. As in CPC, mutual information between latent representations is maximized by optimising the InfoNCE loss function. However, neurons in the biological brain primarily learn from local information and there does not appear to be a global error function that is optimized [24]. Hence, end-to-end backpropagation cannot be used as an accurate learning model for the brain.

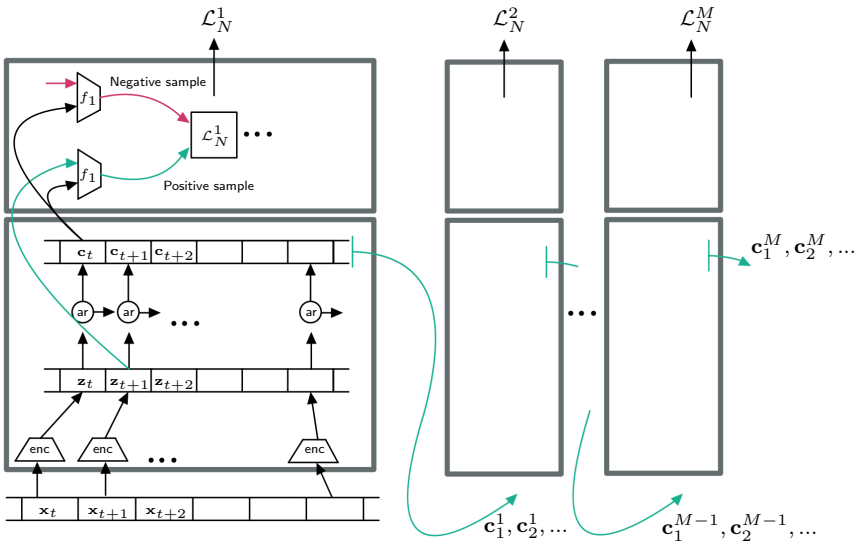

Figure 2.6: Overview of Greedy InfoMax

Towards greedy learning

Instead of optimising a single loss function (as in end-to-end backpropagation), Löwe et al. split the neural network up into "modules". Each module is optimised with its own personal instance of the InfoNCE Loss function $\mathcal{L}_N$ described in equation 2.17. This idea is depicted in figure 2.6, which displays M modules, each trained with their own instance of the InfoNCE loss. Each "module" can be considered as an instance of the CPC framework where high dimensional data goes in and a lower dimensional representation comes out. The output representations of a one module serve as the input for the next module. One can thus think of Greedy InfoMax as "stacking" multiple ANN's, which were each trained with the InfoNCE loss on top of each other. A module may consist of one or more layers of the neural network.

Optimising Greedy InfoMax

By splitting up the neural network into modules and allowing them to optimise their own loss, modules can be trained asynchronously, improving upon training time when multiple GPUs are used. Note that similar to tradition neural networks, during forward propagation each module (or layer) requires the output of the previous layer as its input. Hence, modules are still dependent

on previous modules for training, however, they can learn without the gradients of subsequent modules.

We now revisit the function f_k and NCE Loss, and describe it in the context of GIM. Instead of a single f_k and L_N, we now have a function f_k^m for each module m, each with their own respective loss ${\mathcal{L}_{\mathcal{N}}}^m$. The attentive reader may notice $f_k^m(\cdot)$ no longer receives as input c_t, but instead a second z_t. Löwe omits the autoregressive function **TODO: F(X) = CT OF ZO** to obtain c_t and immediately applies gives opted for this function instead, as she did not notice any improvement in her preliminary results.

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

$$\mathcal{L}_N^m = -\sum_k \mathbb{E}_{\mathrm{X}} \left[\log \frac{f_k^m(\mathbf{z}_{t+k}^m, \mathbf{z}_t^m)}{\sum_{\mathbf{z}_j^m \in X} f_k^m(\mathbf{z}_j^m, \mathbf{z}_t^m)} \right]$$

Chapter 3

Variational contrastive predictive coding

Learned latent representations are meaningful to a computer. However, the meaning of the latent representations is unknown to humans. While variational auto-encoders can omit this challenge by constraining the latent space in such a way that makes it more interpreteable, similar to the autoanecoder, they "merely" learn to reconstruct image. Hence, all "information" that is considered important to reconstruct the data will be kept in the latent representation, whether the information is useful or not. TOTOOT: CPC DOET EFFECTIEF DINGEN WEG DIE WE USELESS VINDEN, MAAR LESS EXPLAINABLE.

We introduce Variational Contrastive Predictive coding, which uses the InfoNCE loss function to obtain latent space, however, by taking inspiration from variational autoencoders, we constrain the obtained latent space to be Gaussian. Doing so allows for obtaining meaninful representations from interpolating between two variables. We argue that these Gaussian distributed latent representations are more explainable. We develop a decoder to reconstruct the original data. By doing so, we can apply meaningful interpolated latent representations to the decoder, to observe what the effect is on different dimensions from the latent representations.

3.1 Loss function

$$\mathcal{L}_{N+KL} = -\sum_{k} \mathbb{E}_{\mathrm{X}} \left[\log \frac{f_k(z_{t+k}, c_t)}{\sum_{z_j \in X} f_k(z_j, c_t)} \right] + \lambda \mathrm{KL}(q(z|X)||\mathcal{N}(0, I)) \tag{3.1}$$

Where the KL divergence for a single sample $x^{(i)}$ is approximated as follows:

$$\frac{1}{2} \sum_{j=1}^{J} \left(1 + \log((\sigma_j^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2 \right) \tag{3.2}$$

idk what the right part of the equation means, i did not have it in the code. L is number of samples per data point, so could be the reconstruction error. (In our case L is always 1)

where $z^(i, l) = \sigma^{(i)} \odot \epsilon^{(l)}$ and $\epsilon^(l)\mathcal{N}(0, I)$ **todo: variables should maybe be bold.**

3.2 Learning benefits

Overfitting during inference: - The same datapoint has multiple (similar) representations, such that learning techniques for downstream tasks will not be able to "memorise" the latent space as easily. - Holes: more predictable inference, such that unseen data is more likely to be near clusters. And thus downstream tasks receive latents that are more similar to what is seen before. - Independent latent dimensions - During training similar behaviour to batch normalization in-between layers

Chapter 4

Experiments

4.1 Greedy Infomax model

- train CNN
- architecture: 4 layers ...
- dataset

4.1.1 Dataset

The Greedy Infomax model is trained on speech data. The model takes as input a raw speech signal of a fixed length and outputs a latent representation for that signal. The dataset is split up into 729 training files and 122 test files. In each 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". All the sounds are spoken by the same person, at a constant and peaceful **todo: describe emotional aspects of speech audio**.

The following transformations are applied to the audio files. Although the original contains a sample rate of 441 Khz, the audio files are downsampled to 16 Khz, matching the sample rate used by Löwe [14]. This significantly reduces the size of the latent representations, and thus the required amount of VRAM during training. Additionally, two types of noise are added to the data. We apply Gaussian white noise, at different decibels ranging between zero and fifteen **TODO ...**.

- also background noise from dataset. is a way to enlargen our dataset.

- Each audio file is cut to have length **10240**. Additionally,

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: 4.1. 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.

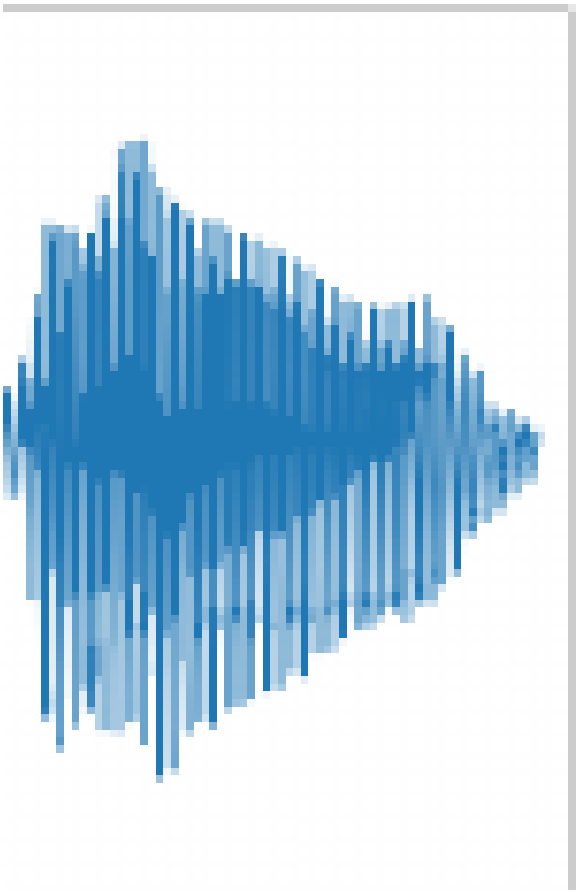

Figure 4.1:

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)

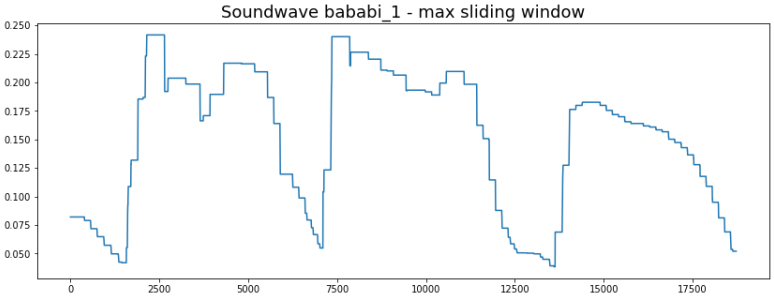

Figure 4.2:

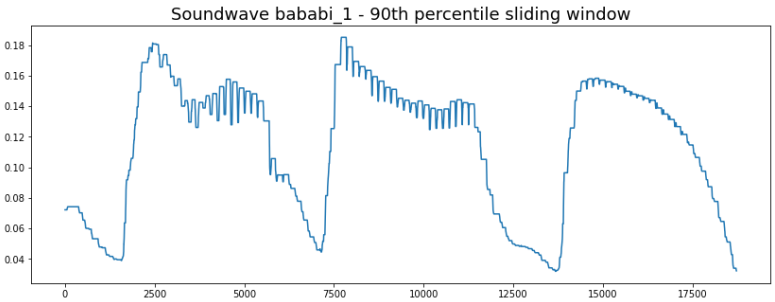

Figure 4.3:

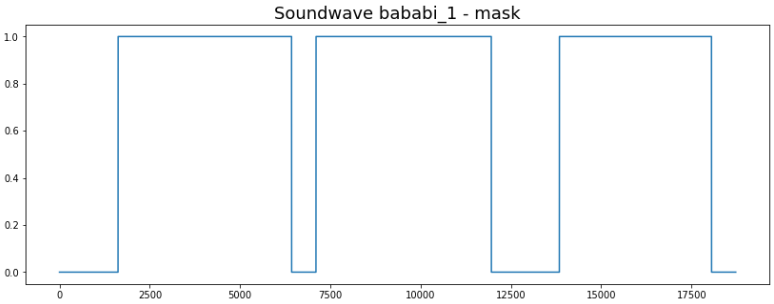

Figure 4.4:

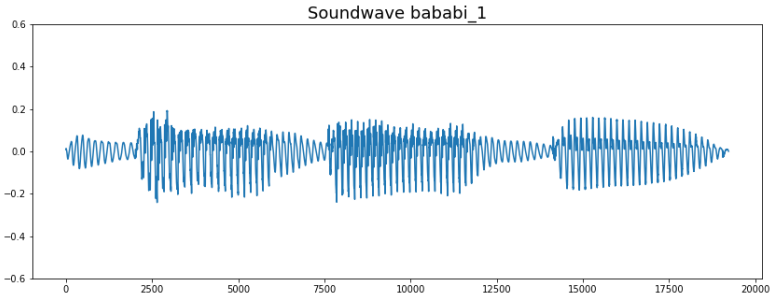

Figure 4.5:

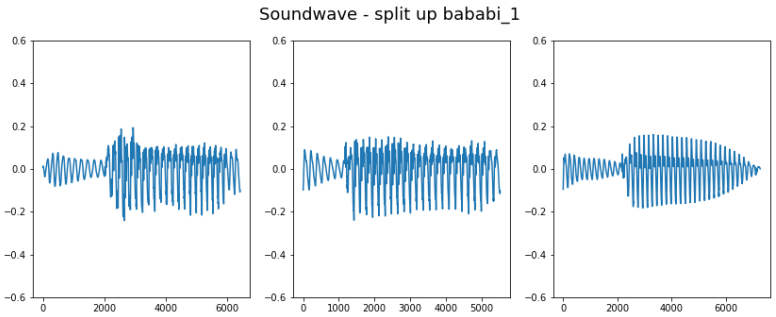

Figure 4.6:

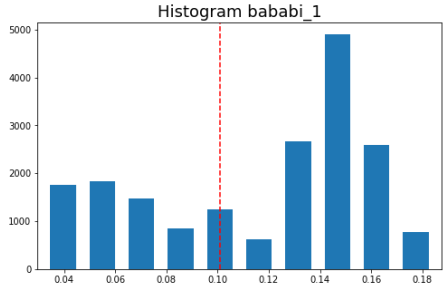

Figure 4.7:

4.1.2 Training and validation during training

Although loss is used as evaluation metric, it only shows part of the picture. The main objective for the representations is to obtain some form of "decoupled/dis-entanged" features that are more easily separable. This evaluation is done by projecting the latent representations to a 2D plane, via t-SNE. Then datapoints are coloured in depending on their the syllable that was pronounced, eg: "gi" or "ga".

Training is done speech signals of fixed length, eg 8,800 samples. Notice however, that the neural network only makes use of convolutional neural networks layers and GRU's, no fully connected layers. The input dimensions can therefore be variable during inference. Only the number of channels in the latent representations should be constant, but the length can change.

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.

4.2 Decoders for Greedy Infomax

Objective: understand GIM learned representations. To do so we first discuss the model's architecture. To verify whether GIM does in fact contain the necessary information, we reconstruct the original speech signal from a GIM representations. We define train a decoder model for each of the four layers of the GIM model.

- Layer 1 ... 4
- Dataset from Bart
- Reduce audio to 16khz, reducing the size of the dataset and speeding up training time
- Add background noise to the data.
- Use this dataset: openslr.org instead of white noise as is more representative.

4.3 Feature selection for visualisation

The GIM architecture consists of 4 layers, each layer 512 kernels Due to the large number of learned kernels, we filter based on standard deviation. This will result in looking at weights which are sensitive to particular features.

4.4 Variational contrastive predictive coding

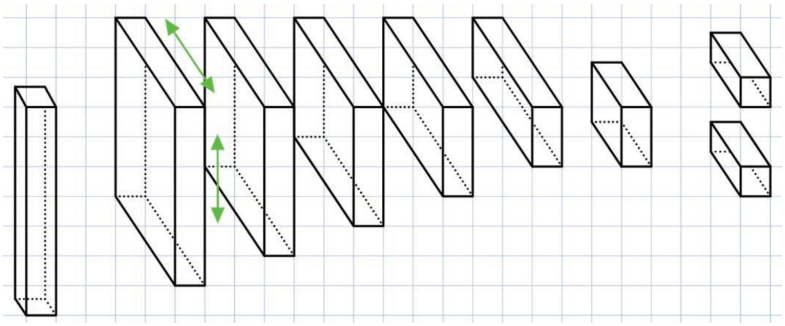

Figure 4.8:

Figure 4.8 displays how via contrastive predictive coding an input speech signal is transformed into two latent vectors. The two vectors combined describe a Gaussian distribution for each feature of the latent representation. One vector corresponds to the means and the other to the standard deviations of the latent distributions. We variational autoencoders assume independent latent features, such that the covariance matrix is non negative on the diagonal and zero off the diagonal. This allows the covariance matrix to be discribed using a single vector. We again, make use of this (plausible incorrect?) assumption of having independent latent features.

4.5 CHATGPT ANSWERS FOR VAE

Yes, you can incorporate the regularization component from a VAE into the contrastive predictive coding (CPC) loss function to encourage the learned latent representations to be Gaussian distributed. This can be achieved by adding a KL divergence term between the learned latent distribution and a Gaussian prior distribution to the CPC loss function.

The modified loss function would be:

$$\mathcal{L}N + KL = -\sum k\mathbb{E}_{\mathrm{X}}\left[log\frac{f_k(zt+k,c_t)}{\sum_{z_j\in X} f_k(z_j,c_t)}\right] + \lambda \mathrm{KL}(q(z|X)||\mathcal{N}(0,I)) \tag{4.1}$$

where $q(z|X)$ is the learned distribution of latent representations, and $\mathcal{N}(0,I)$ is the standard Gaussian prior distribution. The hyperparameter λ controls the weight of the KL divergence term in the overall loss function.

To compute the KL divergence term, we need to calculate the mean and covariance of the learned latent distribution $q(z|X)$. Let μ and Σ be the mean and covariance of $q(z|X)$, respectively. Then the KL divergence can be computed as:

$$\mathrm{KL}(q(z|X)||\mathcal{N}(0,I)) = \frac{1}{2}\left(\mathrm{tr}(\Sigma) + \mu^T\mu - k - \log\det(\Sigma)\right) \tag{4.2}$$

where k is the dimensionality of the latent space.

The function $f_k(z_j, c_t)$ can be defined using a linear layer as follows:

code

Here, 'hidden size' is the size of the hidden layer, 'latent size' is the size of the latent space, and c is the context vector for a given time step. The function $W_k(z)$ applies a linear transformation to the input z, resulting in a latent vector of size 'latent size'.

To incorporate the KL divergence term into the CPC loss function, you can compute the mean and covariance of the learned latent distribution using the torch.mean and torch.var functions, respectively. The modified loss function can then be implemented as follows:

code

Here, z is the learned latent representation, c is the context vector, and lambd is the weight of the KL divergence term. The 'compute logits' function computes the logits for the CPC loss as described in the original CPC paper.

Note that this implementation assumes that the ....

—

To incorporate the regularization component from VAE into the CPC loss function, we can add the Kullback-Leibler (KL) divergence term to the loss function, which will encourage the latent representations to be Gaussian distributed. The KL divergence term measures the difference between the learned latent distribution and the target Gaussian distribution.

The KL divergence term can be defined as:

$$\mathcal{L}KL = \frac{1}{2}\sum i = 1^d(\mu_i^2 + \sigma_i^2 - \log(\sigma_i^2) - 1)$$

where μ_i and σ_i are the mean and standard deviation of the i-th dimension of the learned latent distribution, and d is the dimension of the latent space.

To combine the CPC loss and the KL divergence term, we can simply add them together, with a weighting factor β to balance their contributions:

$$\mathcal{L} = \mathcal{L}N + \beta\mathcal{L}KL$$

where $\mathcal{L}_N$ is the CPC loss function defined in equation (2.17), and β is a hyperparameter that controls the strength of the regularization term.

The final loss function will encourage the learned latent representations to be both informative for the CPC task and Gaussian distributed, which can be useful for downstream tasks such as clustering or classification.

—

Sure! In the context of contrastive predictive coding (CPC), the 'compute logits' function is used to compute the similarity scores (logits) between the predicted future and the actual future.

In more detail, 'compute logits' takes in two inputs, z and c, which represent the latent representation and the context for a given time step, respectively. The function computes the similarity score between the predicted future representations and the actual future representations by performing a dot product between the predicted future representation and a set of parameters (i.e., weights) that are learned during training. Specifically, 'compute logits' applies a linear transformation to the context c and uses it to score each element in the predicted future z.

The output of 'compute logits' is a vector of logits that corresponds to the similarity scores between each predicted future representation and the actual future representation. These logits are then used to compute the contrastive loss, which is a measure of the similarity between the predicted future and the actual future.

The 'compute logits' function is a crucial component of the CPC algorithm, as it determines how well the predicted future representations match the actual future representations, which in turn affects the quality of the learned latent representation.

4.6 Results

4.6.1 Results CPC Simple v2 w/ 2 modules

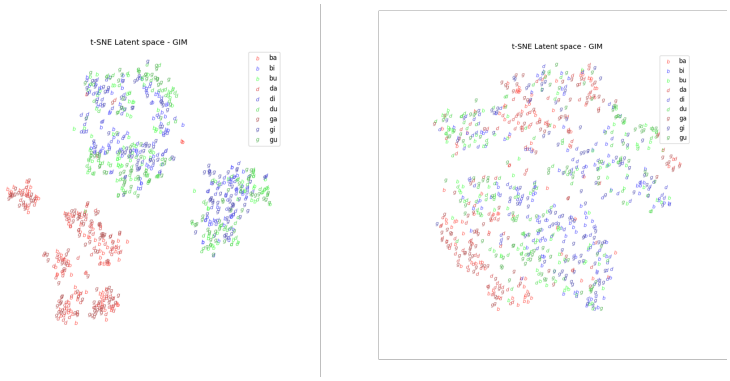

Figure 4.9:

4.9: can see that second model harms performance. We believe this can be explained via the learning rate. 4.10, there we see that second module performs better separation, indicating that the intermediate KL convergence constraint (causing the normal Gaussian distributions) also serves as a batch normalisation term, and thus resulting in faster convergence.

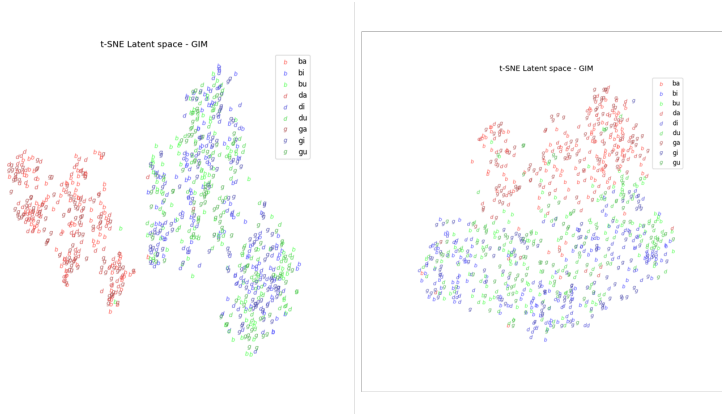

Figure 4.10:

4.7 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?

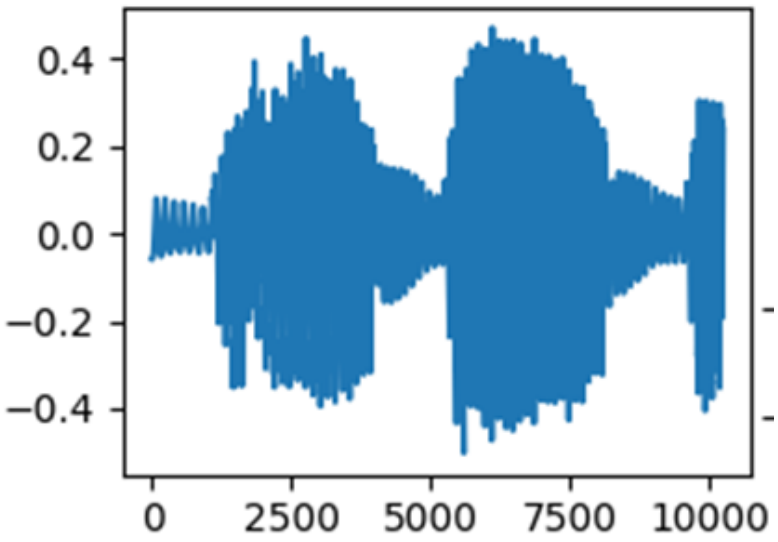

Figure 4.11: "BA-BA-BA" time domain

No batch normalisation, so although channels appear to have larger activations than other

Figure 4.12: Activations of the sound "BA-BA-BA" through GIM

channels, size of activation does not really say anything about information. eg activations 0.01 could still contain more information than 3.0 activation.

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 4.13.

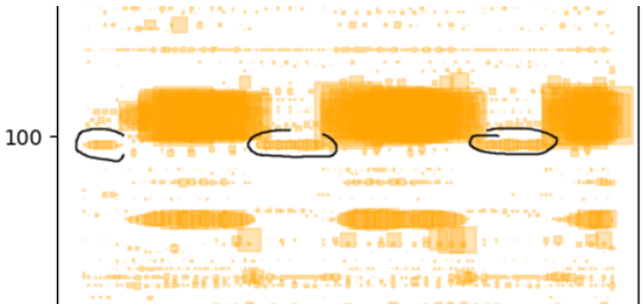

Figure 4.13: zoomed in

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 4.14 also nicely shows how different channels have clusters of activations at slightly different times.

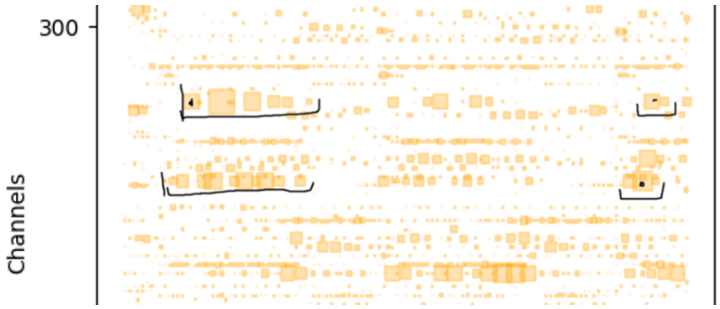

Figure 4.14: Zoomed in

T-sne visualisations: Multiple models trained, trained GIM, but single module (exact CPC architecture), one with autoregressor and once without autoregressor layer, so only CNN layers. 200 epochs, trained on split up data samples.

1) Only CNN: We observe better linear separability

2) CNN + 1 autoregressor layer:

3) This is the pure data visualised (no latent representation). Were at least doing a bit better than the original data, so that's good! our work is not for nothing.

4) old GIM with all modules each one layer. l1 .. 5 - cnn, l6 = gru. img shows l5 = cnn: model can more easily distinguish A's from other **klinkers**. Partly, it kinda makes sense for GIM to learn to separate **klinkers**. Since they last longer (longer duration), the loss function will more likely randomly sample a subwindow from the "aa's" than from the **medeklinker** part.

4.8 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.

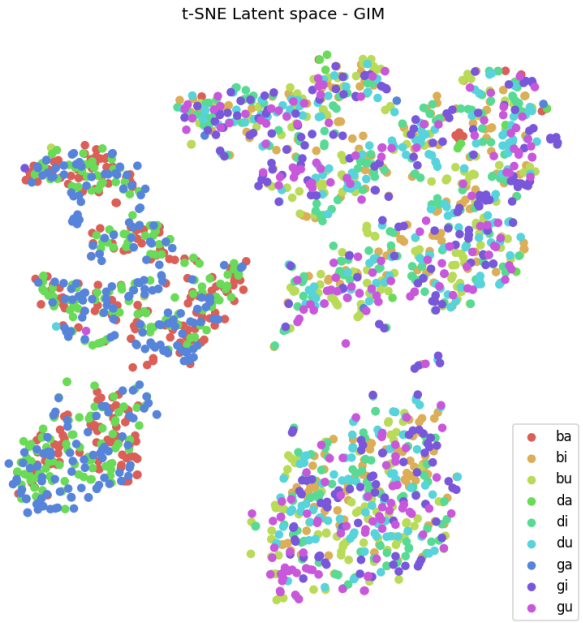

Figure 4.15:

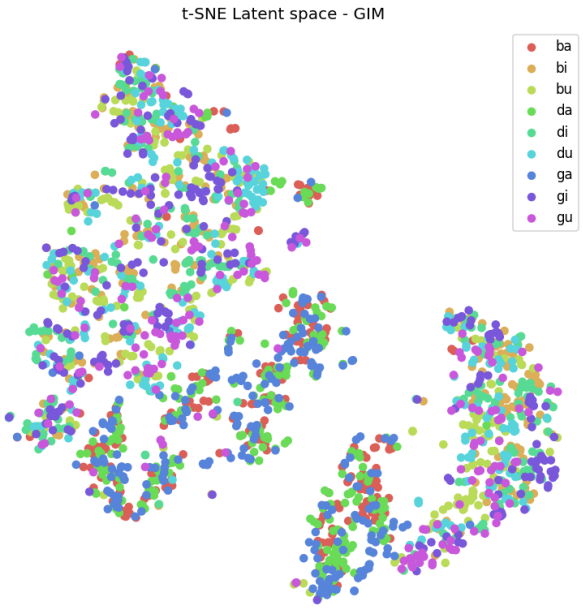

Figure 4.16:

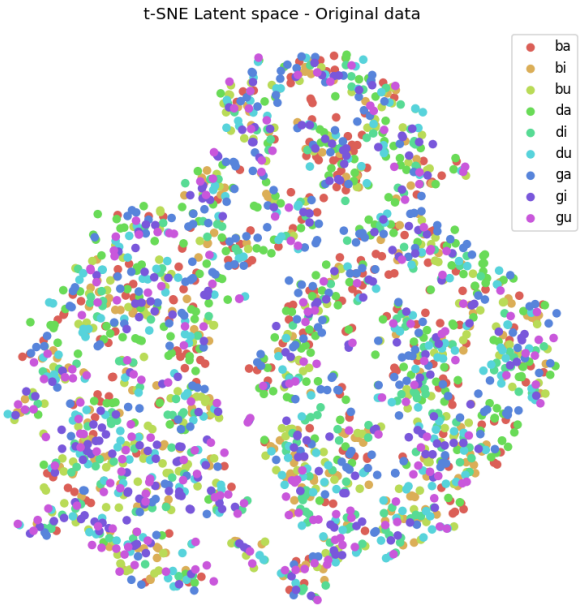

Figure 4.17:

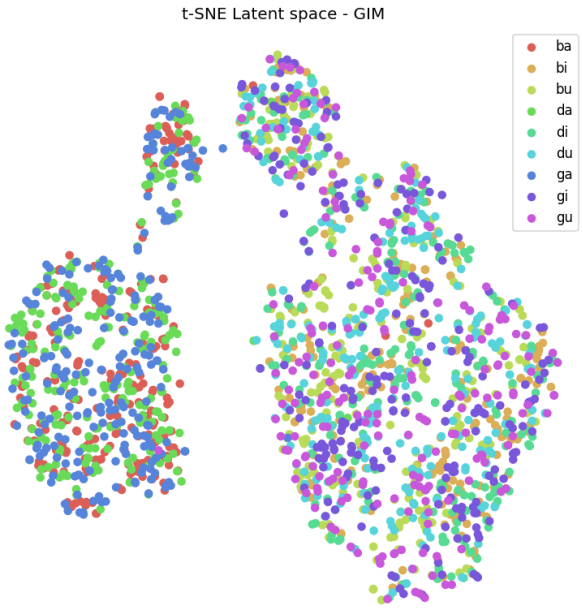

Figure 4.18:

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

Chapter 6

Discussion

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.

- 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.

xx does what [14] yes! idk [1] cool

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

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

[5] 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

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

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

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

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

[10] 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/

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

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

[13] A. van den Oord, Y. Li, and O. Vinyals. Representation Learning with Contrastive Predictive Coding. [Online]. Available: http://arxiv.org/abs/1807.03748

[14] 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

[15] 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

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

[17] 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

[18] P. de Haan and S. Löwe. Contrastive Predictive Coding for Anomaly Detection. [Online]. Available: http://arxiv.org/abs/2107.07820

[19] 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

[20] 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

[21] 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

[22] 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

[23] 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

[24] N. Caporale and Y. Dan, "Spike timing-dependent plasticity: A Hebbian learning rule," vol. 31, pp. 25–46.