Kindergarten-VQ-VAE
Do Transformers offer out-of-the-box disentanglement properties?

Daniele Solombrino, 1743111
solombrino.1743111@studenti.uniroma1.it

Abstract

Kindergarten-VQ-VAE **HomeworkP** aims at assessing whether **minimal changes** ([28] [25]) to the **latent space architecture** of a Transformer Neural Network [30] can make the model **sensible** to **disentanglement** properties [3] [11], without adding an explicit disentanglement-enforcing component in the loss, while optimizing for a text autoencoding task.
Qualitative analyses, performed on the dSentences dataset [22], show that disentanglement does not come in free for out-of-the-shelf architectures, even when introducing **supervision** on the latent generative factors of the data.

Code is available at https://github.com/dansolombrino/Kindergarten-VQ-VAE

1 Setup

This first section specifies details about all the components that were used to carry the text style transfer and latent arithmetic qualitative analyses out.

1.1 NLP task

Following the reference paper [20], text **autoencoding** was selected as pretext task.

1.2 Dataset

dSentences [22] is a **synthetic dataset**, composed of 330k sentences in the "subject + verb + object" grammatical form.

Each sentence includes eight **generative factors** (object grammatical plurality, subject grammatical gender, number and plurality, verb tense and progression and sentence type and negation), for which labels are provided. Table 1 shows some example entries of the dataset, while Table 2 shows all possible values for each generative factor.

1.3 Model

Trasformers [30] have taken over the NLP field, dethroning RNNs [26] [17] and LSTMs [12] and offer a flexible base architecture to work on. So, for these reasons and considering the goals of this **HomeworkP**, a **Transformer**-like architecture has been adopted.

In this first stage, both the encoder and the decoders use **pre-trained** backbones, like BERT [8] and [24], leveraging implementations from the open-source Huggingface library [31].

Both the encoder and the decoder take as input a **perturbed** version of the sequence to autoencode, where perturbation is performed by **masking** tokens out with a special token or **replacing** them with other **random** tokens. Different random perturbations are applied to the encoder and the decoder inputs, in order to promote generalization.

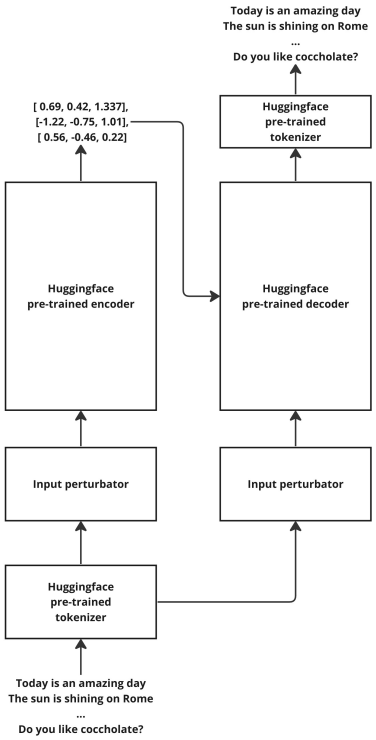

Figure 1: Bagon model architecture.

The encoder outputs a **latent representation** for the sentence, while the decoder outputs the **reconstructed** original sentence.

The entire schematics of the architecture used at this stage is in Figure 1.

Huggingface offers different possibilities of latent sentence representations, including a latent code having the same dimensionality of the input sentence ([sequence_length, embedding_size], in PyTorch [23] [1]-like notation) or a latent embedding having the same dimensionality of just one token ([embedding_size], usually 768 or 1024)

In both backbones, the string storing the input sequence is **tokenized** using a pre-trained tokenizer, provided by Huggingface as well. Huggingface allows to mask padding tokens, so as the model does not attend to them while training. This option has always been turned on for the encoder and decoder both.

Sentence	Labels
I accepted the payment	0 0 0 0 0 0 0 0
Do they accept the apology	0 1 1 1 2 0 1 0
We will be not eating lunch	1 0 0 1 0 1 2 1
They are pushing the buttons	1 0 0 1 2 0 1 1

Table 1: An example of labelled inputs provided by the dSentences dataset. Each of the eight generative factors has its own label, so the label for each sentence is a eight-elements long vector.

Autoencoding tasks do not require **causal** masking, contrary to other cases where the model should not be able to look at the future (e.g. language generation), so experiments were conducted with causal masking turned on and off, with basically no difference in the various analyses, for the encoder and decoder both.

At this stage, the encoder part may seem unnecessary, as autoencoding may be perfectly performed without the help of the prior conditioning provided by the encoder. However, this serves as a base for the subsequent stages, where **learnable transformations** will replace the straight through encoder-decoder connection path (Figure 4), in order to learn a representation for the generative factors in the data.

1.4 Loss function

In typical autoencoding fashion, the loss function used at this stage aims at minimizing the **distance** between the input sentence and the reconstructed one.

Our inputs are **sequences** of tokens, where a single token is sampled from a **distribution** over the entire vocabulary, using an argmax operation.

For these reasons, we can use a Kullback–Leibler divergence loss term, applied

Generative Factor Name	Possible Values	Example
Object grammatical plurality	Singular, Plural	”Car”, ”Cars”
Subject grammatical gender	Male, Female, Neutral	”He”, ”She”, ”It”
Subject grammatical number	1st, 2nd, 3rd	”I”, ”You”, ”They”
Subject grammatical plurality	Singular, Plural	”I”, ”We”
Subject grammatical plurality	Singular, Plural	”I”, ”We”
Verb tense	Past, Present, Future	”Liked”, ”Like”, ”Will like”
Verb progression	Progressive, Not progressive	”Watch”, ”Watching” ”I like pizza”,
Sentence type	Declarative, Interrogative	”Do I like pizza?” ”I like pizza”,
Sentence negation	Positive, Negative	”I do not like pizza”

Table 2: All possible values that each of the eight generative factors of dSentences dataset can assume.

token-wise, since every token is a probability distribution over the vocabulary.

1.5 Hyperparameters

Since both encoder and decoder are well known backbones, the same optimizer and hyperparameters proposed in their papers [8] [24] were used in the various fine-tuning stages.

Train/validation/test splits were set to 70/20/10%, while a batch size of 512 was used.

Tokenized sentences were padded to reach the same length across different batches.

Models were fine-tuned for up to 5 epochs.

Hyperparameter fine-tuning was not necessary, as all configurations (Table 3) perfectly succeeded in the autoencoding task using the default hyperparameters.

2 Evaluation

Text autoencoding **accuracy**, **style transfer** and **latent traversals** have been adopted in the evaluation stages across all attempts.

Accuracy has been chosen as it provides an easy, human-readable, architecture, loss and hyperparameters-agnostic assessment of the quality of the text autoencoding task, contrary to loss values.

Style transfer and latent traversals have been chosen as qualitative evaluation metrics, as quantitative approaches, like [6], [18] and [11] require additional specific training to be performed on meticulously-prepared data.

2.1 Accuracy

Accuracy is computed sentence-wise as

$$A_i = \frac{\text{\# correctly reconstructed tokens}}{\text{total \# tokens}} \quad (1)$$

, then these values are averaged across the batch and finally the batch-wise accuracies are averaged at the end of each epoch.

Padding tokens are included in accuracy calculations, as ablation studies across all attempts showed no significant difference resulting from their exclusion.

2.2 Text style transfer

Text style transfer aims at **altering** the **style** of an input text while **preserving** its **content**, it can be used as a disentaglement evaluation metric of autoencoders and can be performed following different approaches.

Encoder Decoder	Fine-tuning configuration	Trainable params
BERT BERT	full full	250M
BERT BERT	head	24M
BERT BERT	cross-attns + head	52M
BERT BERT	head head	32M
BERT BERT	head cross-attns + head	52M
BERT GPT-2	full full	250M
BERT GPT-2	head	38M
BERT GPT-2	cross-attns + head	67M
BERT GPT-2	head head	46M
BERT GPT-2	head cross-attns + head	75M

Table 3: Bagon fine-tuning configurations. BERT refers to its "base, uncased" version and GPT-2 refers to its smallest possible version as well. "head" is a linear layer used to perform language modeling.

Kindergarten-VQ-VAE uses **latent arithmetic** [21] approach, as it is used in the reference paper [20] and it perfectly blends in with the adopted Transformer architecture.

Assuming:

- $S = \{s_1, s_2, \ldots, s_N\}$ a sentences dataset
- $G = \{g_1, g_2, \ldots, g_M\}$ some latent generative factors of the sentences (e.g. 2).
- $M = E \cdot D$ a text autoencoder model, where E is the encoder and D is the decoder, trained on S
- $R = \{r_1, r_2, \ldots, r_n\} = \{E(s_1), E(s_2), \ldots, E(s_n)\}$ latent representations for S, learnt by training M

We can use text style transfer as **qualitative** disentanglement **metric** as follows:

- Select a generative factor, $g \in G$, e.g. sentence negation
- Extract three sets of sentences, one for affirmative sentences (e.g. "I love pizza"), S_a, one for negative sentences (e.g. "I do not like vegetables"), S_n, and another one for negative sentences, $\hat{S_n}$, s.t. $S_n \cap \hat{S_n} = \emptyset$
- Encode $S_a, S_n, \hat{S_n}$, i.e. $R_a = E(S_a), R_n = E(S_n), \hat{R_n} = E(\hat{S_n})$
- Compute the difference $D = R_a - R_n$
- Apply D to $\hat{R_n}$, i.e. $\hat{R_a} = D + \hat{R_n}$
- Decode $\hat{R_a}$, i.e. $\hat{S_a} = D(\hat{R_a})$

Intuitively, the difference $D = R_a - R_n$ removes the negative style from the positive style, so, when decoding the latent codes, we expect the sentences to be in an affirmative style.

This is not trivial and we will discuss requirements for this to happen in subsequent sections, when we will dive deeper in the encodings produced by the different models.

2.3 Latent traversals

Latent traversals aim at observing the **impact** of **perturbing** one or more **dimensions** of the latent representations of an input, when feeding such perturbed latent code to the decoder.

It can be used as a disentaglement evaluation metric of autoencoders, by means of testing whether perturbating one specific dimension changes just one and one only specific generative factor in the data.

Assuming the same S, G, M and R from text style transfer example, we can use latent

traversals as qualitative disentanglement metric as follows:

- Get a sentence s_i, e.g. "Do you believe in true love?"
- Get its latent code, $l_i = E(s_i)$
- Perturb l_i. E.g. Set the value of it's j-th dimension to a specific value, i.e. $\hat{l}_i = l_i; \hat{l}_i[j] = 42$
- Decode $\hat{l}_i$, $\hat{s}_i = D(\hat{l}_i)$

If $\hat{s}_i$ shows a change only in one of its generative factors, then M has indeed learnt disentangled representations. In fact, it has learnt some representations that correctly separate the various generative factors in the data.

3 Related works

Text disentanglement is a very addressed topic in NLP, considering its importance in many downstream tasks.

Many approaches have been proposed over the years. Most of the solutions require **specific loss** terms, like [20], [7], [32] and [29], while in other attempts **multi-task datasets** are necessary [16].

[5] is the closest work to Kindergarten-VQ-VAE, as it tried to supervise the conditioning of the decoder. However, it requires additional, external training data and it uses LSTM-based encoder and decoders.

4 Baseline results

This section presents quantitative and qualitative results for the first model architecture, Bagon, shown in Figure 1

4.1 Text autoencoding accuracy

Bagon performs **extremely well** (train, validation and test accuracies $\geq 99\%$) in the autoencoding tasks, across all the different fine-tuning strategies (Table 3), even when applying input perturbations of up to 20% of the input tokens.

This was expected, as the encoder and decoder backbones are pre-trained. In fact, to confirm this, in an ablation study the vanilla Transformer architercture was used and it resulted in way lower performances (up to 50% train, validation and test accuracies, after hyperparameter tuning).

4.2 Text style transfer

Bagon performs **extremely bad** in the text style transfer task.

Applying the difference vector D (2.2) either keeps the result unchanged or does not produce the desired change.

The first case was investigated by looking at the **cross-attention coefficients** of the decoder, since when the decoder ignores the encoder conditioning, the cross-attentions follow an uniform distribution.

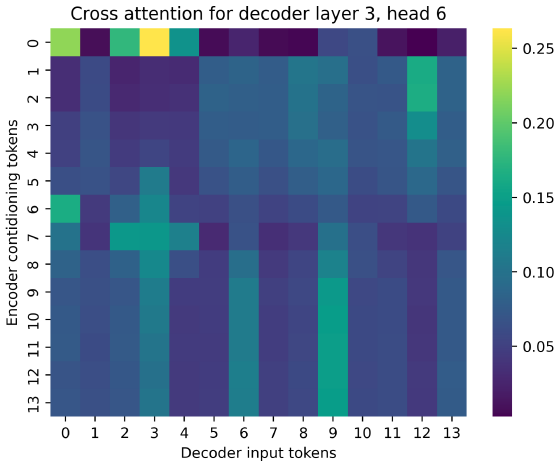

Figure 2: Cross attentions coefficients for an attention head of Bagon decoder layer 3. All other heads across all other decoder layers present similar, not uniform distributions.

Figure 2 shows cross-attention values for one attention head of a decoder layer and shows a non-uniform distribution. As all other attention heads across all decoder layers produce similar results, we can state that the

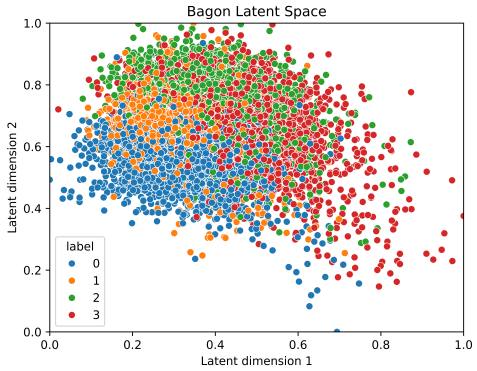

Figure 3: Bagon latent space, min-max normalized. Only four out of all possible classes randomly sampled for practicality.

decoder is indeed **paying attention** to the encoder conditioning. Additionally, feeding random noise as conditioning to the decoder completely messes the reconstruction up, confirming that the decoder is not ignoring the encoder conditioning.

The second case is probably to be attributed to the **structure** of the **latent codes**, or rather lack thereof (see Figure 3). No loss term induces any kind of structure in the latent space, so performing algebraic operations on such codes is not guaranteed to yield meaningful results, as is the case for Bagon.

4.3 Latent traversals

Bagon performs **extremely bad** in the latent traversals task.

The result makes sense, considering the characteristic of Bagon latent encodes discussed in Section 4.2.

This very last consideration bridges us to the next section, where Bagon model will evolve into Shelgon, which includes some **learnable transformations** in the encoder-decoder pathway.

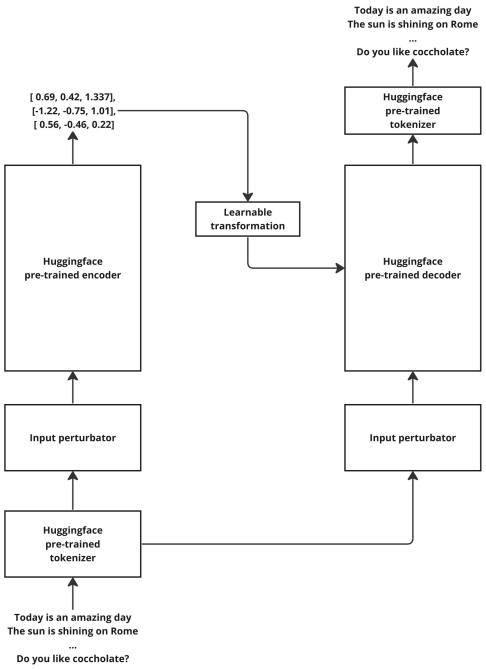

Figure 4: Shelgon model architecture. Unsupervised learnable transformation pictured for practicality, as the bottleneck can be supervised too.

5 Learnable encoder-decoder path

Shelgon (Figure 4) introduces a **learnable transformation** in the encoder-decoder path with a twofold idea:

1. **control** the disposition of the latent representations in the latent space
2. **improve** the conditioning provided to the decoder, making it similar to the reference paper [20]

As far as 1. is concerned, when supervision on the learnable transformation is used, the Network is forced to arrange the latent representations in a meaningful way. The goal of the upcoming experiments is to understand whether this meaningful-for-the-supervision disposition also leads to disentanglement, basically **for free**.

Regarding 2., the learnable transformations can be designed to mimic the reference paper

[20], which sports eight discrete distributions, one for each generative factor.

The term learnable transformation is voluntarily vague, as any kind of Neural Network layers combination can be plugged in there. In the following paragraphs, we will describe all the tested learnable transformations, highlighting motivations that led to the choice, results, comments about observed behaviours and possible future directions.

6 Supervised linear transformation

The model in this section will be referred to as Shelgon$_1$.

Shelgon$_1$ applies a simple **linear projection** (Figure 5) that maps the encoder output to a $|G| \times 3$ matrix, which stores the label for each generative factor $g \in G$.

Supervision happens by means of reducing the **cross-entropy loss** between the predicted labels, for each generative factor. This appropriately-scaled loss term is then added to the autoencoding loss term, as both components compose the final loss for Shelgon$_1$.

The supervised linear transformation adds a total of 5.5k trainable parameters, almost equally partitioned between "Multi-label classification" (2.3k) and "Projection out" (3.2k) layers shown in Figure 5

6.1 Accuracy

Shelgon$_1$ achieves train, validation and test accuracies $\geq$ 99% for both text autoencoding and multi-class, multi-label classification tasks, even when perturning up to 20% of the input sequences (see Section 1.3 for details about input perturbation).

Considering the huge difference in magnitude of number of trainable parameters (5.5k vs. millions), it would be safe to assume that the classification layers had a huge help from the encoder, which probably did all the heavy lifting.

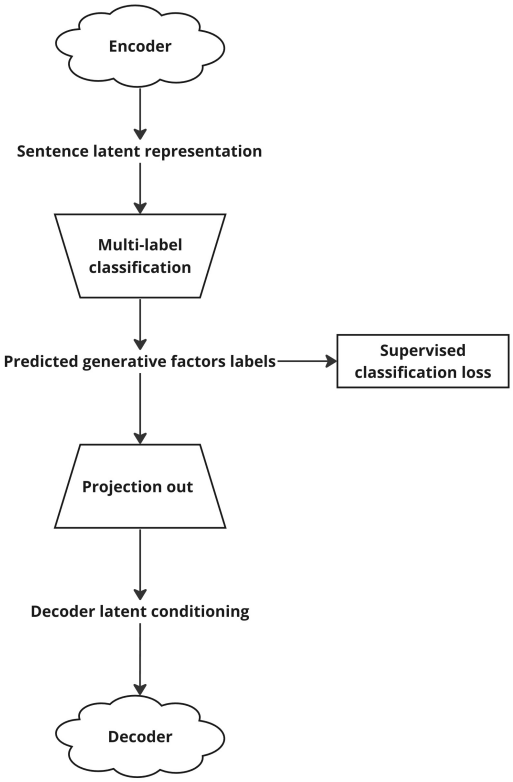

Figure 5: Supervised linear transformation located in between the encoder-decoder path of Shelgon$_1$. Encoder and Decoder omitted for practicality, see Figure 4.

The following two subsections aim at understanding whether the supervision provided by the classification layers actually make disentanglement properties emerge.

6.2 Text style transfer

Shelgon$_1$ shows the the same behavior of Bagon in text style transfer (4.2, i.e. latent arithmetics either changes undesired parts of the sentence or has **no effect** whatsoever).

The structure of the learnable linear transformation allows us to compute the D vector (see 2.2) on either "Sentence latent representation", "Predicted generative factors labels" or "Decoder latent conditioning" of Figure 5. The idea is that one of the parts of the bottle-

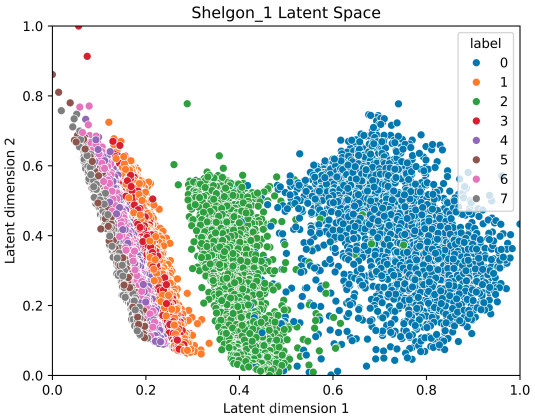

Figure 6: Shelgon$_1$ latent space, min-max normalized. Only eight out of all possible classes randomly sampled for practicality.

neck may characterize the data according to the generative factors we aim to disentangle, as it has to perform classification using these learnt representations.

All three attempts perform the same, as described at the beginning of the section

Same tests of 4.2 confirm that the **decoder** does **pay attention** to the conditioning provided by the learnable transformation.

Exploring the latent space, we start to notice some **clustering** happening for some classes (see Figure 6), however, text style transfer fails on those classes as well, keeping considerations on the latent space written in 4.2 valid.

6.3 Latent traversals

Shelgon$_1$ shows the the same **bad** behavior of Bagon (4.3) in latent traversals as well.

Similarly to 6.2, attempts to traverse "Sentence latent representation", "Predicted generative factors labels" or "Decoder latent conditioning" of Figure 5 were made, in order to understand whether one of the three shows some kind of disentanglement ability.

7 Non-linear transformation

Following the results from Section 6, we speculate that learning disentangled representations is **not** a **linear** phenomenon, so we add non-linerities in-between each layer composing the 5.5k trainable parameters of the now-non-linear learnable transformation. Its schema remains exactly the same as before (Figure 5) and we refer to this model as Shelgon$_2$

Different **activation** functions were considered and GELU [10] was selected, as it is the same used in the encoder and decoder backbones, resulting in absence of **internal covariate shift** [27] [13].

Text style transfer and latent traversals performed as **bad** as Shelgon$_1$ (Sections 6.2 and 6.3), effectively showing that **non-linearities** do **not help** the model organize the data in the latent space in a disentangled way.

8 Deeper transformations

Following the results from Section 7, we speculate that the huge difference in trainable parameters magnitude (5.5k vs. millions) between the encoder-decoder and the learnable transformation does not allow the latter to catch disentanglement properties.

For this reason, Shelgon$_3$ adds **more layers** in the "Multi-label classification" and "Projection out" parts of the learnable transformation, keeping the overall high-level structure of Figure 5.

Linear and non-linear attempts were tested, all yielding the same unconclusive results of Shelgon$_1$ (Section 6) and Shelgon$_2$ (Section 7), showing that larger transformation do **not help** in learning **disentangled representations**.

9 Vector Quantized latent spaces

Vector **quantization** techniques introduced in [28] and in [25] can be applied to transform a continuous latent space, like the one used in all previous approaches (Section 4, 6, 7 and 8), into a discrete one.

The discretization happens by means of **mapping** the continuous encoder outputs to the closest vector in a predefined set of vectors, called the "**codebook**".

Vectors in the codebook and the quantization mapping are trained together with the autoencoder, using specific loss terms.

As far as the codebook is concerned, number of vectors, their initialization and what mapping function to use are some key hyperparameters. We opted for 243 vectors, three different mappings and two different initialization strategies.

The 243 vectors result from having to classify five generative factors, each of which can assume three different values and wanting to assign each possible combination of the five generative factors to its own latent code.

Vector quantizers from [28] [25], [14] and [20] were used.

Attempts using random codebook initialization yield to **notebook collapse** [2], a phenomenon where only a small selection of notebooks gets used, similarly to representation collapse of contrastive learning [15]. One common solution is to use a K-Nearest Neighbours [4]-based initializations, which did not work in our case.

The codebook collapse phenomena happening across all attempts clearly shows **lack of disentanglement**, as sentences with different labels have been placed in the same codebook, multiple times, across the entire dataset.

10 Conclusion

Kindergarten-VQ-VAE **HomeworkP** studies whether **minimal changes** to the structure of the **latent space** of Transformers can encourage **disentanglement** in the learnt representation, while performing text autoencoding on the dSentences dataset [22], using text style transfer and latent traversal **quantitative** analyses.

Across all attempts (Section 4, 6, 7, 8, 9), the qualitative disentanglement analyses clearly tell us that all **proposed modifications** to Transformers do **not help** the model to learn disentangled representations.

Bagon (Section 4) is trained on an autoencoding loss and the model clearly does not use the generative factors as reconstructing factors nor it organizes the data in the latent space in a disentangled way, as shown in Section 4.2

Learnable, **supervised transformations** (Section 6) in the encoder-decoder pathway do **not help** disentanglement either. The model relies on an entangled geometrical positioning of the data in the latent space, in order to draw decision boundaries used to perform supervised classification on the labels. **Non-linear** 7 and **deeper** 8 transformations do **not** yield any **improvement** either, excluding the hypothesis that the transformation has to be large and/or non-linear to show disentanglement properties.

Unsupervised, **quantized** approaches 9 suffer from codebook **collapse**, which places sentences with different generative factors in the same latent code, a scenario in which clearly disentangled representations can not be learnt.

Code is available at `https://github.com/dansolombrino/Kindergarten-VQ-VAE`

11 Future works

An obvious direction would encompass the adoption of a disentanglement-encouraging term in the loss, like the reference paper [20] and related works did.

Different labeling strategies, like **hierarchical labels**, may improve latent space supervision.

Hierarchical vector **quantizations** [9] may be integrated in approaches proposed in Section 9, although a priority would be solving the notebook collapse issue, as it undermines any hope for learning disentangled representations.

Loss terms inspired by **adversarial** approaches may help in learning disentangled representations, like shown in [33] and [19].

References

[1] Meta AI and Linux Foundation. Pytorch. 2

[2] Gulcin Baykal, Melih Kandemir, and Gozde Unal. Edvae: Mitigating codebook collapse with evidential discrete variational autoencoders. 2023. 9

[3] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. 2012. 1

[4] Nitin Bhatia and Vandana. Survey of nearest neighbor techniques. 2010. 9

[5] Danilo S. Carvalho, Giangiacomo Mercatali, Yingji Zhang, and Andre Freitas. Learning disentangled representations for natural language definitions. 2022. 5

[6] Ricky T. Q. Chen, Xuechen Li, Roger Grosse, and David Duvenaud. Isolating sources of disentanglement in variational autoencoders. 2018. 3

[7] Pengyu Cheng, Martin Renqiang Min, Dinghan Shen, Christopher Malon, Yizhe Zhang, Yitong Li, and Lawrence Carin. Improving disentangled text representation learning with information-theoretic guidance. 2020. 5

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018. 1, 3

[9] A. Gersho and Y. Shoham. Hierarchical vector quantization of speech with dynamic codebook allocation. 1984. 10

[10] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). 2016. 8

[11] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. Representation learning: A review and new perspectives. 2017. 1, 3

[12] Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. 2019. 1

[13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 2015. 8

[14] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. 2016. 9

[15] Li Jing, Pascal Vincent, Yann LeCun, and Yuandong Tian. Understanding dimensional collapse in contrastive self-supervised learning. 2021. 9

[16] Vineet John, Lili Mou, Hareesh Bahuleyan, and Olga Vechtomova. Disentangled representation learning for non-parallel text style transfer. 2018. 5

[17] Michael I. Jordan. Serial order: A parallel distributed processing approach. 1986. 1

[18] Hyunjik Kim and Andriy Mnih. Disentangling by factorising. 2019. 3

[19] Michael Mathieu, Junbo Zhao, Pablo Sprechmann, Aditya Ramesh, and Yann

LeCun. Disentangling factors of variation in deep representations using adversarial training. 2016. 10

[20] Giangiacomo Mercatali and André Freitas. Disentangling generative factors in natural language with discrete variational autoencoders. Conference on Empirical Methods in Natural Language Processing, 2019. 1, 4, 5, 6, 7, 9

[21] Tomas Mikolov, Wen tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. 2013. 4

[22] Amine M'Charrak. Deep learning for natural language processing (nlp) using variational autoencoders (vae). 2018. 1, 9

[23] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017. 2

[24] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019. 1, 3

[25] Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. 2019. 1, 9

[26] David E. Rumelhart and James L. McClelland. Learning internal representations by error propagation. 1987. 1

[27] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. 2000. 8

[28] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. 2017. 1, 9

[29] Jake Vasilakes, Chrysoula Zerva, Makoto Miwa, and Sophia Ananiadou. Learning disentangled representations of negation and uncertainty. 2022. 5

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017. 1

[31] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing. 2019. 1

[32] Jiawei Wu, Xiaoya Li, Xiang Ao, Yuxian Meng, Fei Wu, and Jiwei Li. Improving robustness and generality of nlp models using disentangled representations. 2020. 5

[33] Jiahao Zhao and Wenji Mao. Disentangled text representation learning with information-theoretic perspective for adversarial robustness. 2022. 10