Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Disentanglement for Reinforcement Learning

Bachelor's Thesis

Eric Schreiber

ericschr@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Benjamin Estermann
Prof. Dr. Roger Wattenhofer

June 7, 2022

Acknowledgements

I would like to thank my supervisor Benjamin Bestermann very much. You always took time for me and gave me constructive feedback. I enjoyed working on this thesis particularly because you gave me the liberty to do what I thought was right and nudged me in the best direction by providing me with appropriate reading material. When something did not work out you always assisted me, even when you had a lot on your plate. I am very happy to have been able to write the thesis with you.

Furthermore, I would like to thank Professor Wattenhofer for making this thesis possible in the Distributed Computing group.

Finally, I want to thank the TIK group for sharing their computational resources with me, which enabled me to run all of the experiments seamlessly and at no costs.

Abstract

In this work, we investigate to which extent disentangled inputs have an influence in reinforcement learning. We test this with a Deep-Q approach for the Atari games Pong and Breakout. As inputs to the DQ-net we use disentangled features of images from the environment. As a comparison we also train the agent directly on the raw images and on reduced features coming from a trained autoencoder. To get the disentangled features, we train three different types of variational autoencoders for both games and use the encoder of the VAEs as a preprocessing step to the DQ-net.
We found no correlation between the disentanglement scores and the overall performance of the RL agent, nor did we observe an improvement in the final reward of the agent. However, compared to training on the raw images as inputs, we did find a slightly faster increase in rewards at the beginning of the training when using the VAEs' encoded features as input. The results appear to show that this improvement is likely correlated to the disentanglement of the input variables. Lastly, the use of VAEs is accompanied by a significant increase in training time which negates the potential benefits of a faster convergence when training on the VAEs' encoded features.

Contents

Chapter 1

Introduction

If one wants to train a model for complex problems and in an unsupervised manner that require a sequence of correct inputs, such as computer games or moving a robotic arm, it is advantageous to use a reinforcement learning approach[1, 2]. In reinforcement learning an agent that can move freely in its environment is used. This agent can be trained to learn ways to reach a favourable outcome. In return, these desired outcomes are rewarded. As input for the agent, one can use either abstracted data, such as position data, or high dimensional data, like images of the environment. If you want to use images, the agent actually has to solve two problems simultaneously. On the one hand, it has to learn to understand the images and extract useful features, on the other hand, it has to learn to move through the environment in order to fulfil the task. A problem that arises with high dimensional input data is the persistence of variables that are not independent of one another. If we could first transform this input data into a few independent features, we would need a smaller network. This would require less memory in training as well as in deployment. Additionally, it would lead to less computational effort and possibly to a faster convergence at the beginning. Under certain conditions it may even induce a better final result.

Variational autoencoders can be used to compress information into some latent dimensions. It has been shown that with proper regularisation, the representation in the latent dimension can be encouraged to contain disentangled features. If little or, in the best case, no information is lost and as long as the training and task data have the same distribution these reduced and disentangled representations can be used to simplify a downstream task[3].

In this work we aim to further validate the aforementioned observation. For this, different variational autoencoders are trained on images of multiple Atari games. Firstly, we investigate how well these trained encoders can disentangle the input images. Subsequently, the encoders of these trained VAEs are used as a preprocessing step to a Deep-Q-Network[1]. With this setup we train the network to play the different Atari games and compare it to the original implementation[1] where a convolutional neural network is applied directly to the input images. Since a perfect disentangled latent space would only need to store the positional data of objects in an image we also train a DQN with the positions as inputs. We can then use this DQN as a benchmark for perfect disentanglement. These different networks will be compared based on the assumptions made above in the second part of the analysis.

Chapter 2

Theoretical Background

2.1 Variational Autoencoder

Variational autoencoders were first proposed by D.P. Kingma and M. Welling [4]. A variational autoencoder consists of two parts: an encoder and a decoder. Unlike a normal autoencoder, the latent representation obtained from the encoder is not directly fed into the decoder. The encoder learns to output the mean and logarithmic variance of a multivariate gaussian distribution, given some input. We then draw our latent space values from this distribution and feed these latent variables into the decoder.

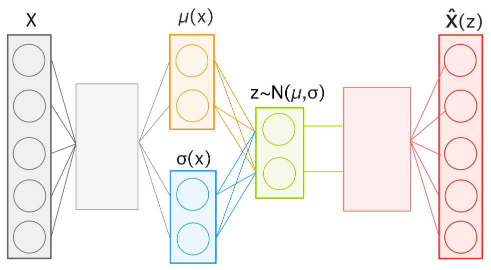


Figure 2.1: An example of a variational autoencoder with emphasis on the sampling nature of the latent space

It seems like this architecture implicitly assumes that all input variables arise from a Gaussian distribution. The reader may correctly note that our data does not at all resemble a normal distribution, in any case. However, we can map this normal Gaussian distribution to all possible distributions by utilising a sufficiently complicated function. This function is learned by the decoder. Theoretically, the decoder could learn the correct distribution independently. However, this is complicated in training because strong dependencies between the dimensions in the original input exist. In this case the encoder helps by discovering the underlying structure in the data set. Intuitively, the encoder breaks down these dependencies and stores reduced representations of the data point in the bottleneck, the so-called latent variables. Therefore, we can model our input variables x as a conditional probability distribution $p(x|z)$, where z are the variables in our latent space. The encoder can be described as $q_\phi(z|x)$ and the decoder as $p_\theta(x|z)$ with ϕ and θ being the weights of the networks that parameterize q and p respectively.

The loss function consists of two parts: a reconstruction error $\mathbb{E}_{q_\phi(z|x)}[log\, p_\theta(x|z)]$ and a Kullback-Leibler divergence term. Through the reconstruction error, the two probability distributions $q_\phi(z|x)$ and $p_\theta(x|z)$ remain as similar as possible. In other words, the encoder and the decoder should perform inverse operations. For a good reconstruction we additionally want to make sure that our model uses a smooth latent space that corresponds to a normal distribution. For this, we use the KL divergence between $q_\phi(z|x)$ and the prior $p(z)$ which we model as a normal distribution. In summary, we arrive at the following loss function [4]

$$\mathcal{L}_{VAE} = \mathbb{E}_{q_\phi(z|x)}[log\ p_\theta(x|z)] - D_{KL}(q_\phi(z|x)||p(z))$$

Since we draw randomly from a probability distribution between the encoder and decoder, back-propagation cannot directly be applied. Therefore, the reparametrization trick is used [4]. Simply put, we sample a value from a normal distribution and then scale this value with the average and variance from our trained latent space. This way we have separated the drawing from a probability distribution and the learning of the network and can use back-propagation.

2.1.1 Disentanglement

A disentangled latent representation means that changing a variable in the latent space leads to a change in a single ground truth factor in the output. For example, a parameter could represent the orientation or the background colour in an image. For disentangling the latent space different approaches with particular types of VAEs are possible. In the following subchapters we will look at three different ones. To evaluate disentanglement we use two metrics. One is the DCI disentanglement score proposed by Eastwood & Williams [5] and the other is the Mutual Information Gap MIG [6]. The DCI disentanglement score is high if each latent variable learns a single ground truth factor. The MIG, on the other hand, measures if each ground truth factor is learned by a single latent variable.

2.1.2 β-VAE

So far we have looked at our input x as a single probability distribution. However, we can split x into a conditionally independent part v and a conditionally dependent part w. We want the latent factors made by $q_\phi(z|x)$ to reflect the independent factors v in a disentangled manner. For this we keep $q_\phi(z|x)$ as similar as possible to a normal distribution $p(z)$. This controls the amount of information that can be stored in this latent variable and thus, causes statistical disentanglement [7].

$$\mathcal{L}_{\beta-VAE} = \mathbb{E}_{q_\phi(z|x)}[log\ p_\theta(x|z)] - \beta D_{KL}(q_\phi(z|x)||p(z))$$

Varying the hyper-parameter $\beta > 1$ puts more pressure on the representation type in latent space than on the reconstruction. Through that, the encoder learns 'the most efficient representation of the data' [7] thus a disentangled representation of v. It was further shown that the optimal β is not the same for all data sets and thus, one has to find the optimal β for each data set separately. In this thesis this is done in a supervised manner.

To find disentangled latent representations β must be chosen to be large. However, this also limits the amount of information that can be stored in the latent space. This is a major weakness in β-VAE which other types, like the β-TCVAE, presented in the subsection 2.1.3 try to mitigate.

2.1.3 β-TCVAE

The KL-divergence from the β-VAE can be further divided into a sum of a index-code mutual information term, a total correlation term, and a dimension-wise KL term. Important is the total correlation part which measures the dependencies between the individual latent variables [8].

$$D_{KL}[q(z)||\prod_j q(z_j)]$$

It is argued that the β-VAE only disentangles its latent variables because this term is included in the KL divergence $D_{KL}(q_\phi(z|x)||p(z))$. Therefore to improve disentanglement while not loosing information capacity we want to punish the total correlation term and not the index-code MI term.

However, the total correlation term is difficult to calculate because it is based on the entire training data set. Although it can be estimated by minibatches.

$$q(z) = \mathbb{E}_{p(n)}[q(z|n)] \approx \frac{1}{M}\sum_{i=1}^{M}[log\ \frac{1}{NM}\sum_{j=1}^{M} q(z(n_i)|n_j)]$$

Note that $z(n_i)$ is a variable drawn from the distribution $q(z|n_i)$. From this, one can calculate the total correlation term. We want to punish this total correlation term and leave the index code MI and the dimension KL weighted with one. For a more straight forward implementation we can take the β-VAE loss and add the total correlation term weighted with a hyper-parameter $\beta - 1$.

$$\mathcal{L}_{\beta-TCVAE} = \mathcal{L}_{\beta-VAE} - (\beta - 1) * D_{KL}[q(z)||\prod_j q(z_j)]$$

2.1.4 JL1-VAE

The JL1-VAE loss is a β-VAE loss regularised with the L_1 term of the Jacobi matrix from the latent values to their mean [9].

$$\mathcal{L}_{JL1-VAE} = \mathcal{L}_{\beta-VAE} - \gamma \mathbb{E}_{q_\phi(z|x)}[\ \|J_g(z)\|_1\ ]$$

γ is a hyper-parameter that determines how strong the regularisation is. Due to the computational complexity of computing the full Jacobian, $\|J_g(z)\|_1$ is calculated from a sample z from the distribution $q_\phi(z|x)$ and the whole Jacobi matrix is only estimated from this.
It was shown that the regularisation term locally disentangles the latent variables better because each latent direction is made to represent groupings of pixels. Therefore, this method works best if the ground truth factors induce some confined local changes in the pixel space. Since this is often the case in games the L1 regularisation may also help us in our aim to find a good disentangled representation in games.

2.2 Deep-Q Net[1]

Reinforcement learning consists of a set of states of the environment, an agent that interacts with the environment, and a set of actions for each set of states. The goal is to assign an action to each element in the set of states so that the final reward is maximised. To achieve this we can learn a function which maps a given state-action pair to a value. This is called Q-learning. With a learned Q-function $Q(s_t, a_t)$ we then simply take the action a_t with the highest Q-value given a state s_t. We learn this Q-function with a deep neural net.

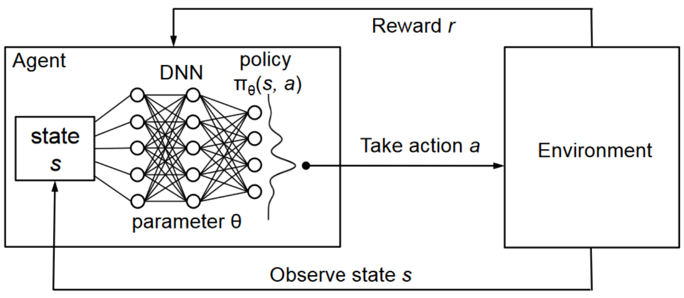


Figure 2.2: Interaction of the environment and the agent[10]

To train our Q-function we store a buffer of experiences e with $e_t = (s_t, a_t, r_t, s_{t+1})$ and train on random samples from this buffer. This has two main advantages:

[1]This work is strongly oriented towards the idea and implementation of the DQN from 'Playing Atari with Deep Reinforcement Learning'[1]

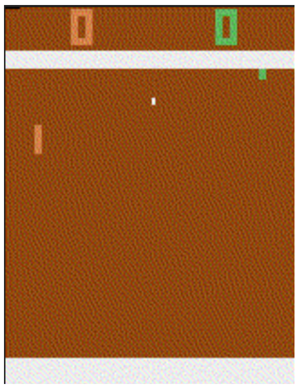
first we can sample each step potentially several times which leads to a greater sample efficiency. Second, we have a strong correlation between the samples stemming from the fact, that the same action is often used for a comparatively long time before switching to another. Randomizing over the experience buffer breaks those correlations and decreases the variance of the updates. To ensure some variance in the chosen actions and therefore to explore new paths we ignore the currently known best action with a probability of ϵ and chose an action at random. This epsilon should get smaller over time reducing the random exploration of new paths and letting the training fine tune the Q-function on the knowledge gained during exploring.

As input for the Deep-Q net one can either use a direct observation of the environment [1] or some abstraction or features of it. In this thesis we will train DQNs on all three different kinds of inputs. First we will train it directly on raw images of the observations of the environment using a convolutional neural network as DQN. Secondly, we will train a DQN on the latent space of several different VAEs giving us some abstracted features. Lastly we will train a DQN directly on handcrafted features of an observation like the positional data of the objects.

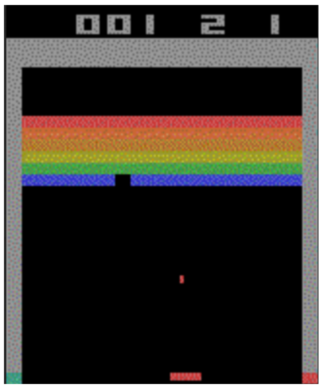
Chapter 3

Methodology

To investigate whether disentangled inputs help in reinforcement learning, we first need VAEs with good disentanglement and reconstruction performance. In a first step, we will train different VAEs and compare their performances. Then we will take the best of these VAEs and apply them as a preprocessing step in our reinforcement learning environment. By doing so, we allow the agent to directly take advantage of the extracted features, instead of the agent working on raw pixel inputs. To have comparisons, we will also train our DQN with a VAE with good reconstruction but poor disentanglement performance, the ground truth factors, and with the original approach[1]. The ground truth factors give us the possibility to simulate perfect disentanglement if the VAEs do not manage this. In order to have some diversification we will do the analysis for the two Atari games Pong and Breakout.



(a) Pong



(b) Breakout

3.1 Reinforcement Learning

To interact with the environment of our two Atari games we can load and interact with the Arcade Learning Environment [11]. From this environment our agent

gets a buffer of four preprocessed 84x84 images. Four consecutive images are used so that the agent receives further information, such as movements of the objects, without having to use a recurrent neural network approach. In the original approach, these four images are passed along to a convolutional DQ-net as the color channel. When using VAEs, we first want to preprocess the images with a VAE and forward the latent variables of the encoded images to a trainable DQ-net. Since VAEs can learn better on a single image than on buffered images, we will encode the input images one by one through the VAE onto the latent space and then stack these four latent variables into a buffer. These stacked latent variables are then passed to the DQ-net.
The DQ-net calculates estimates of the Q-values of each action. Our agent then executes the action with the best Q-value with a probability of $1-\epsilon$. On the other hand a random action is chosen with a probability of ϵ. This should help the agent to try out new paths. Since we want to try out many new paths at the beginning and only few at the end of training, we introduce an ϵ-decay. We start with a ϵ of 1, i.e. completely random moves, and go down to an ϵ_{min} of 0.02. As mentioned in section 2.2 we use a replay buffer. This replay buffer should be quite large. We will use a size of 10'000. All DQ-nets will be trained for 1 million frames in order to get to a more or less stable regime.

3.1.1 DQN-Setup in the Original Approach [1]

In order to train directly on the stack of four input images, we use three convolutional layers, each with shrinking kernel size and stride. After the convolutions we have four images of size 28x28. These four images we consolidate into one input vector and forward it into two dense layers. These dense layers have a size of 512 in between and the number of actions as output size. As an activation function we use the ReLu function between all layers.
As an ϵ-decay we use an exponential decay with a decay constant of $85 * 10^{-6}$.

3.1.2 DQN-Setup with a Variational Autoencoder

In order to get the latent variables of an image we use the encoder part of the pretrained VAEs. We give the pretrained encoder the four images and then flatten all the latent variables next to each other into a vector.
If we assume that the convolutional layers in the original approach only do feature extraction and learn the position data of the objects in the image, it should be possible to keep only the linear layers from the original DQ-net. The latent variables also contain the position data of the objects as well as other information.

In chapter 4 Results it is explained that these two linear layers are not sufficient to train a good agent. Therefore, the DQ-nets are each enlarged by three linear layers of size 3136 which is the input size of the original DQ-net. In between each layer an Instance-norm is performed and a ReLu activation function is used. Since we expect a faster convergence to a good solution we do not need to try as many random paths as the original approach. Therefore we can use a faster ϵ-decay of $85 * 10^{-5}$.

3.1.3 Preprocessing

In order to be able to use the same training methods for all games, all pictures are first scaled down to 84x84. Then we remove certain remnants such as borders at the bottom of the image. Finally, we convert all images to one color channel and scale the values between zero and one, such that the objects are all one and the background is all zero. We keep this preprocessing identical for all training runs.

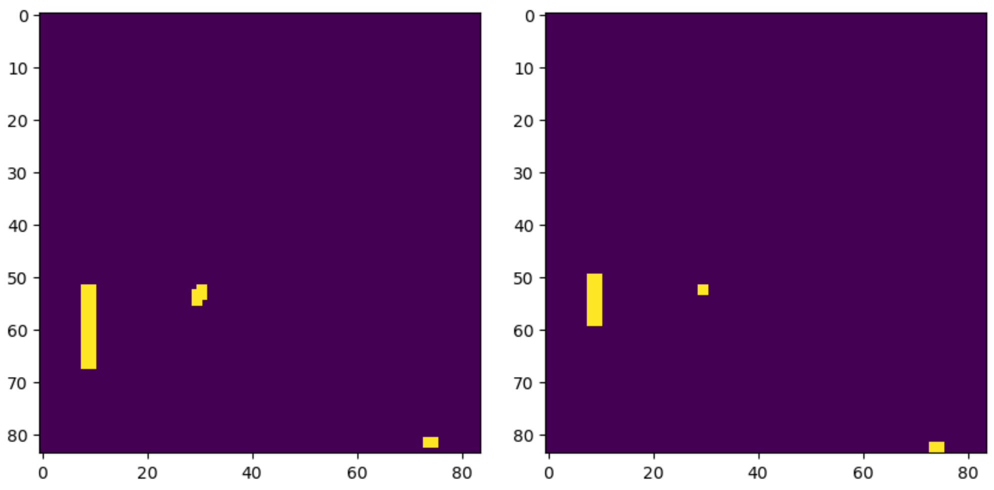
3.2 Variational Autoencoder

In order to test our VAEs with all metrics from 3.2.1 we need ground truth labels. Since we cannot get these from the ALE environment, we build our own data set where we can control where the objects are located within the image. This is advantageous, given that we ensure that the objects do not overlap which should help to disentangle the objects. A disadvantage, however, is that the images from the ALE are not always exactly the same. As one can see in picture (a), sometimes the shape of the ball or the size of the paddles changes. An example being evident in the game of Pong. These changes are not captured by my data set. Thanks to the generalisation ability of the VAEs, this still works well.

The structure of the VAEs is chosen in such a way that the encoder part corresponds to the DQN of the original approach with the change that the output layer has twice the size of the latent space (once for the mean and once for log-variance). The decoder is then the same structure as the encoder but in reverse. Additionally, the convolutional layers are replaced with convolution transpose layers. Between all layers we perform a batch normalisation. The activation function between all layers is the ReLu function. As a last step, the output of the decoder is fed into a sigmoid function. The VAEs are trained for 100 epochs with 100k images each. The hyperparameters β and γ for the JL1-VAE are chosen in such a way that we achieve near perfect reconstruction and the highest possible disentanglement value.

3.2.1 Metrics

Each trained VAE is evaluated according to different metrics. The most important metric is the visual assessment of the reconstruction. If the decoder cannot



(a) Image from ALE environment (b) equivalent own image

Figure 3.2: Possible differences between the original images and the self made ones

reconstruct all objects, we can assume that this information is not contained in the latent variables. This would make it impossible for the DQ-net to use the object while training. Therefore, we will use the models with highest disentanglement scores having good reconstruction. For the different VAEs used as a preprocessing step to the DQ-net of the game Pong we get the following scores.

Model	β	γ	Latent Dim	DCI	Completeness	MIG
β-VAE	5	-	10	0.253	0.160	0.011
β-VAE	0.001	-	10	0.022	0.014	0.017
βTC-VAE	2	-	10	0.237	0.168	0.081
βTC-VAE	1.1	-	64	0.275	0.173	0.068
JL1-VAE	1	0.01	10	0.453	0.300	0.050

Table 3.1: Metrics for the VAEs used as a preprocessing step in the game Pong

Since all VAEs in the game Pong have a bad MIG but a higher DCI score, we can judge that most latent variables learn a single ground truth factor. However, all ground truth factors are learned by multiple latent variables. Perfect disentanglement would require independence in both directions, latent space to ground truth and ground truth to latent space. Therefore, we do not achieve perfect disentanglement. However, compared to the bad disentanglement of the bad β-VAE with $\beta = 0.001$, we can see that our models can disentangle in one direction. Additionally, the JL1-VAE disentangles much better than the others. This makes sense because the JL1-VAE works especially well with small objects in an image like we have in the game of Pong.

Model	β	γ	**Latent Dim**	**DCI**	**Completeness**	**MIG**
β-VAE	0.6	-	32	0.427	0.414	0.308
β-VAE	0.0001	-	32	0.026	0.026	0.010
βTC-VAE	0.1, TC=1	-	32	0.118	0.115	0.081
JL1-VAE	0.5	0.001	32	0.296	0.288	0.198

Table 3.2: Metrics for the VAEs used as a preprocessing step in the game Breakout

Since the game Breakout with 28 ground truth factors requires a significantly larger latent space, we use VAEs with 32 latent variables each. It is all the more astonishing that the β-VAE with $\beta = 0.6$ performs so well, even in comparison to the VAEs used for the game Pong. This was probably a exceptionally good run, because neither the βTC-VAE nor the JL1-VAE could deliver equally good results here. Not surprisingly, the JL1-VAE performed better than the βTC-VAE, as it did for the game of Pong. The high MIG compared to the VAEs in 3.1 is also remarkable. This means that, in the best case of the β-VAE, around a third of the ground truth factors were learned from only one latent variable. Which in total would mean that even tough we had to learn a lot more ground truths than at the game of Pong some of our VAEs are much better.

It is also worth noting that a correlation between DCI disentanglement and performance in the games is more likely than a correlation with the MIG score as this paper [12] has found in a comparable context. Therefore, the low MIG scores in the game of Pong are undesirable but not a big deal. If there is a correlation between MIG scores and performance in the game Breakout, further research could be conducted in this direction.

Chapter 4

Results

At the beginning of this thesis we asked ourselves the following four questions

i) Whether we can achieve better final results in the games when using disentangled inputs.

ii) Whether the agent converges faster to a good result when using disentangled inputs.

iii) Whether we achieve a faster training time per frame with the VAE as a preprocessing step

iv) Whether we can overall use a smaller model

We investigate this in comparison with increasing disentanglement and to the original RL agent in the two games Pong and Breakout.

Game	Model	Mean Max Reward	Achieved after	Absolute Max	Times
	βTC 2-VAE	18.4	**598k**	20	2
Pong	βTC 1.1-VAE	**19.6**	692k	20	**4**
	β **5-VAE**	18.0	648k	**21**	2
	Bad β-VAE	18.0	782k	20	1
	JL1-VA	15.4	676k	20	1
	Ground Truths	10.0	714k	16	1
	Original Approach	19.4	679k	20	2
	Bad β-VAE	**26.8**	480k	29	1
Breakout	β 0.6-VAE	22.6	616k	27	1
	β0.1 TC 1-VAE	20.2	877k	22	**2**
	JL1-VAE	25	**390k**	**31**	1
	Original Approach	26.4	623k	28	**2**

Table 4.1: The mean max reward and the number of frames until it was achieved is computed as a mean over 5 runs with different random seeds. The absolute maximum is the highest maximum achieved at least by a single run e.g. a single random seed. With times being how often this maximum was achieved

From the table 4.1 we can see that in the game Pong all agents, except the one trained directly on the ground truth labels, have reached a very high maximum in at least one run. With 21 being the highest value possible in the game. Whereby the agent who got the inputs from the βTC-VAE had at least one game with 20 points in 4 out of 5 random seeds. This is twice as often as the original approach without the VAE-preprocesssing step. This is important because at the maximum the training should be stopped and we would have an agent who wins the game. Since this succeeds with 4 out of 5 of the random seeds tested, we have a higher success rate than with the original approach. In the Breakout game, the agent that gets the inputs from the JL1-VAE had the absolute highest reward.

If one compares the average of the maximum rewards, a significant improvement in any game using disentangled inputs compared to without, cannot be observed. It should also be noted that there were four times more runs with VAEs than without. Since the maximum rewards vary greatly and depend on many factors as well as on the random seeds, we can say that it is not surprising that in both games a VAE agent got the maximum reward. If we plot the dependence of DCI Disentagnlement and MIG versus the average maximum rewards 4.1 it becomes even clearer that there is no dependence here. We are probably reaching the limit of what is feasible for the original architecture. For better results, one would not only have to change the inputs to the DQ-net but the whole setup itself and probably use something other than deep-Q learning.

The second point is similar. From table 4.1 we note that the highest rewards were found the fastest by a VAE agent for each game. However, there are also VAE agents that took significantly longer than the original approach. Further we look at the reward plots 4.4. From this we observe that there could only be a correlation between DCI disentanglement and the time it takes to get an acceptable result of 80% of the maximum achieved result. However, we can only see

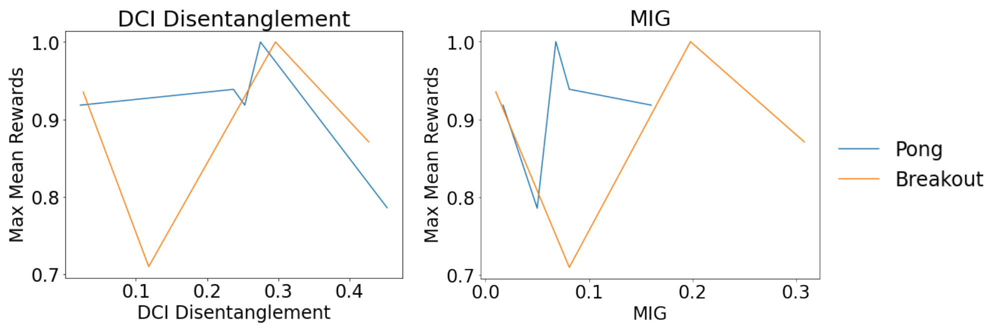
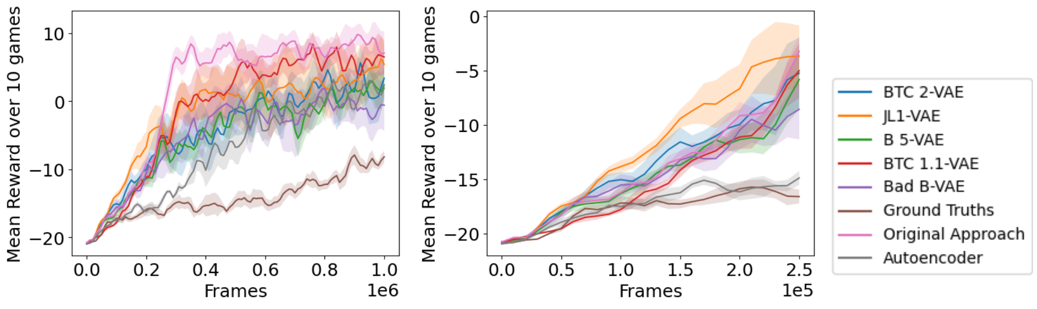


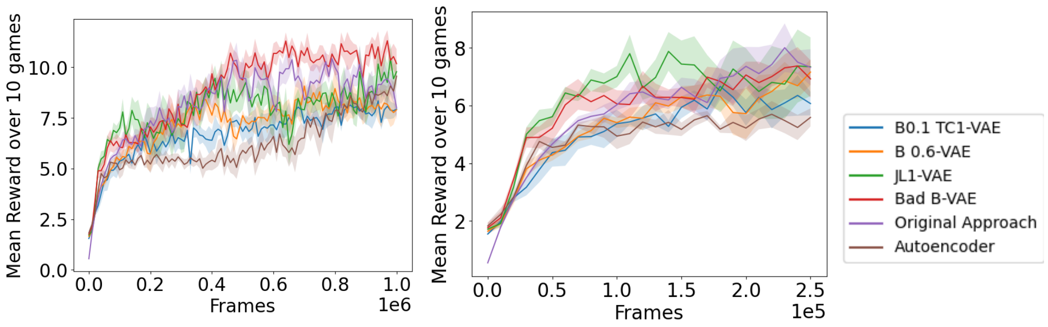
Figure 4.1: Maximal achieved mean reward against the DCI disentanglement and MIG scores corresponding to the VAEs used for preprocessing. All mean rewards are normalised to allow a comparison between the games.

this in the game of Pong and neither is it a strong trend. The threshold of 80% for a good final reward is arbitrarily taken. It was verified that slightly changing this threshold does not change the final orders.



(a) Full training length (b) First 250k frames

Figure 4.2: Average rewards over 10 games for Pong with a confidence interval of 70% and smoothed out



(a) Full training length (b) First 250k frames

Figure 4.3: Average rewards over 10 games for Breakout with a confidence interval of 70% and smoothed out

Finally, we can look at starting performance in 4.2 and 4.3. Here we observe that most VAE agents do actually achieve higher rewards than the original approach at the very beginning. If we graph the starting performance versus the disentanglement scores 4.5 we observe that there possibly is a connection between the DCI disentanglement score and the starting performance. However, we have too few VAEs with different disentanglement scores to state this conclusively only from the plots. Additionally, the MIG scores were all too low to conclude anything. Once the original agent has learned to extract features from the pictures, its reward explodes significantly and grows much faster than all VAE agents. Therefore, one might suspect that the faster increase of the VAE agents at the beginning can be explained by the inputs lower dimensionality. To test this assumption, we can use a normal autoencoder instead of a VAE. Appendix

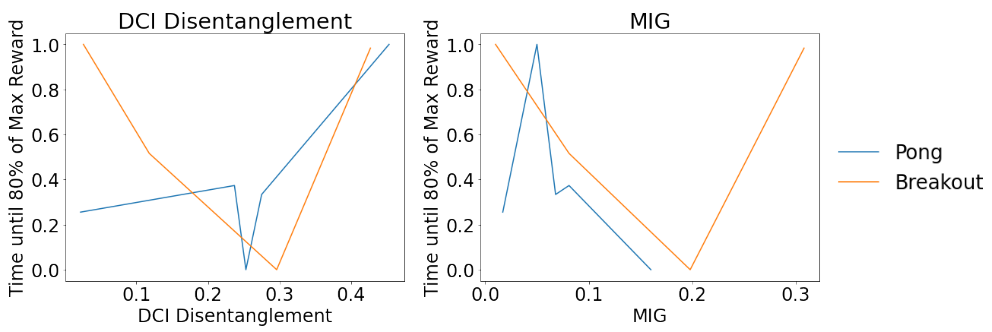


Figure 4.4: Mean of the number of frames it took until 80% of the maximal reward was achieved. The axis first to 80% was normalised in such a way, that lower number of frames are closer to 1 e.g. the higher the better.

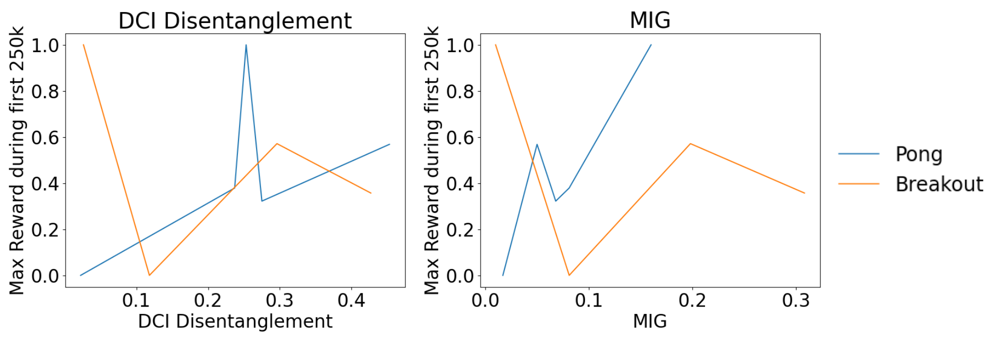


Figure 4.5: Maximal achieved mean reward during the first 250k frames against the DCI disentanglement and MIG scores corresponding to the VAEs used for preprocessing. All mean rewards are normalised to allow a comparison between the games.

C contains a clearer table and plots with the start performances of the VAEs and the AEs than plots 4.2 and 4.3. Since the AE performs worse in the beginning than the VAEs, we can conclude that the improvement of the rewards at the very beginning of the training was derived by the VAEs and is not only related to a dimensionality reduction. Together with the aforementioned findings we can conclude, that disentanglement helps in the beginning of training to get better rewards. However, as shown later, training with VAEs takes much longer. Therefore, in our case it would be faster to use the original approach even with the best VAE. This would be the quickest way to achieve the best possible result at any time also at the very beginning. It is still unclear why the agent who got the ground truths did so badly. I think there might be a bug in the way I calculate the ground truths from the pictures. If you want to take a closer look at this you should first try to get the ground truths directly from the environment instead of calculating them from the pictures like I did.
We can summarise that there is a better starting performance with the VAE

agents and this probably does weakly depend on the DCI disentanglement scores of the used VAEs. Since the original approach still performs best it could be exciting to use an encoder from a normal autoencoder as a DQN-net and train it to learn the Q-function. This could have the same good starting performance as we see with the VAE agents, but have the added advantage that the long-term performance is the same as in the original approach.

Since the original DQ-net consists of three convolutional layers followed by two linear layers, one might think that the convolutional layers only extract features such as the location of objects in the image. Therefore, when using the latent variables as input to our DQ-net, our net could consist of only the two linear layers. This would have made the training per frame faster than in the original approach. However, we never achieved an acceptable performance in training with this small network. Therefore, we can conclude that the small DQ-net did not have enough capacity and more importantly, that the original DQ-net used the convolutional layers not only for feature extraction, but also for Q-function learning. Since these convolutional layers are omitted in our DQ-net, we have to add more linear layers. Therefore, the DQ-net was enlarged until the average rewards did not increase any more. With the larger DQ-net, however, we also lose the possibility of a faster training time per frame, as we have practically the same number of trainable parameters with the enlarged DQ-net. Additionally, we need to encode each image, which is quite computationally intensive. To estimate the difference in training time, the average training time was calculated in table 4.2. There we can see that the original approach was indeed much faster. Although all runs were carried out on the same resources, the training times varied greatly. I suspect this had to do with a memory bottleneck on the cluster when it was fully loaded. Therefore, we cannot give reasonable training times, only estimates of how long it takes to train per frame.

Game		**Average Time per Frame**
Pong	VAE-Agent	16.08 ms
	Original Agent	7.92 ms
Breakout	VAE-Agent	18.64 ms
	Original Agent	9.95 ms

Table 4.2: Mean training times per frame with all models trained on a Titan XP.

Chapter 5

Conclusion

In this work, we investigated the extent to which disentangled inputs have an influence on reinforcement learning. We tested this with a Deep-Q approach for the Atari games Pong and Breakout. We trained three different types of variational autoencoders for both games respectively and took the VAE with the highest DCI disentanglement score for each type. We used the encoders of these VAEs as a preprocessing step for the DQ-net in our agent. Additionally the same preprocessing was done with trained autoencoders. This gives us a range of disentangled inputs for our agent and allows us to see which changes in performance are due to the disentanglement and which would most likely be found with similar pretrained feature extractors.

We have seen that encoding with a VAE generates a slight performance increase at the beginning of the training. However, this does not affect the final performance of the model. Furthermore, we could show that this performance increase is related to disentanglement. But the performance increase is not sufficient to offset the increased training time. One particular point that can be further worked on is the effectiveness of the used VAEs. Theoretically, it should be possible to train a VAE so that its latent variables are perfectly disentangled. Although we have seen a trend in disentanglement to start performance, we cannot rule out that with perfect disentanglement this trend will be even stronger or flatten out. Therefore, one should continue to try to train VAEs with perfect disentanglement.

Overall, we can conclude that even tough disentanglement can help, the pretrained features used in the VAE probably matter far more than their disentanglement. Therefore, I suggest that the next step is to take an encoder pretrained by an AE as the DQ-net of the agent and train it to estimate the Q-function. The fact that the encoder has already found features could improve the training time and simultaneously, allowing us to keep the advantages of the original approach: the agent can learn its features by itself and we have a smaller total net. I don't suspect that the final scores will improve, but the training time required and the variance between the individual runs could be reduced.

Bibliography

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013. [Online]. Available: https://arxiv.org/abs/1312.5602

[2] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: lessons we have learned," *The International Journal of Robotics Research*, vol. 40, no. 4-5, pp. 698–721, jan 2021. [Online]. Available: https://doi.org/10.1177%2F0278364920987859

[3] A. Dittadi, F. Träuble, F. Locatello, M. Wüthrich, V. Agrawal, O. Winther, S. Bauer, and B. Schölkopf, "On the transfer of disentangled representations in realistic settings," 2020. [Online]. Available: https://arxiv.org/abs/2010.14407

[4] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013. [Online]. Available: https://arxiv.org/abs/1312.6114

[5] C. Eastwood and C. Williams, "A framework for the quantitative evaluation of disentangled representations," in *Sixth International Conference on Learning Representations (ICLR 2018)*, May 2018, 6th International Conference on Learning Representations, ICLR 2018 ; Conference date: 30-04-2018 Through 03-05-2018. [Online]. Available: https://iclr.cc/Conferences/2018

[6] R. T. Q. Chen, X. Li, R. Grosse, and D. Duvenaud, "Isolating sources of disentanglement in variational autoencoders," 2018. [Online]. Available: https://arxiv.org/abs/1802.04942

[7] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "β-vae: Learning basic visual concepts with a constrained variational framework," 2017. [Online]. Available: https://openreview.net/pdf?id=Sy2fzU9gl

[8] R. T. Q. Chen, X. Li, R. B. Grosse, and D. K. Duvenaud, "Isolating sources of disentanglement in variational autoencoders," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/1ee3dfcd8a0645a25a35977997223d22-Paper.pdf

[9] T. Rhodes and D. D. Lee, "Local disentanglement in variational auto-encoders using jacobian l_1 regularization," 2021. [Online]. Available: https://arxiv.org/abs/2106.02923

[10] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, 2016.

[11] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, jun 2013. [Online]. Available: https://doi.org/10.1613%2Fjair.3912

[12] S. van Steenkiste, F. Locatello, J. Schmidhuber, and O. Bachem, "Are disentangled representations helpful for abstract visual reasoning?" 2019. [Online]. Available: https://arxiv.org/abs/1905.12506

Appendix A

Model Architectures

Model	**Architecture**	
	Encoder	Decoder
VAE for Pong	Input: 84x84 1 channel Conv2d: 32, kernel 8, stride 4 Conv2d: 32, kernel 4, stride 2 Conv2d: 64, kernel 3, stride 1 Linear: 512 Linear: 2* 10 *Batchnorm between all layers*	Input: 10 values Linear: 512 Linear: 6400 ConvT2d: 32, kernel 3, stride 1 ConvT2d: 32, kernel 4, stride 2 ConvT2d: 32, kernel 8, stride 4 Sigmoid activation function *Batchnorm between all layers*
VAE for Breakout	Input: 84x84 1 channel Conv2d: 32, kernel 8, stride 4 Conv2d: 32, kernel 4, stride 2 Conv2d: 64, kernel 3, stride 1 Linear: 512 Linear: 2* 32 *Batchnorm between all layers*	Input: 32 values Linear: 512 Linear: 6400 ConvT2d: 32, kernel 3, stride 1 ConvT2d: 32, kernel 4, stride 2 ConvT2d: 32, kernel 8, stride 4 Sigmoid activation function *Batchnorm between all layers*
AE for Pong	Input: 84x84 1 channel Conv2d: 32, kernel 8, stride 4 Conv2d: 32, kernel 4, stride 2 Conv2d: 64, kernel 3, stride 1 Linear: 512 Linear: 4 *Batchnorm between all layers*	Input: 4 values Linear: 512 Linear: 6400 ConvT2d: 32, kernel 3, stride 1 ConvT2d: 32, kernel 4, stride 2 ConvT2d: 32, kernel 8, stride 4 Sigmoid activation function *Batchnorm between all layers*
AE for Breakout	Input: 84x84 1 channel Conv2d: 32, kernel 8, stride 4 Conv2d: 32, kernel 4, stride 2 Conv2d: 64, kernel 3, stride 1 Linear: 512 Linear: 32 *Batchnorm between all layers*	Input: 32 values Linear: 512 Linear: 6400 ConvT2d: 32, kernel 3, stride 1 ConvT2d: 32, kernel 4, stride 2 ConvT2d: 32, kernel 8, stride 4 Sigmoid activation function *Batchnorm between all layers*

Table A.1: All final VAE and AE model architectures used in this thesis.

Model	**Architecture**
Original DQN	Input: 84x84 4 channels Conv2d: 32, kernel 8, stride 4 Conv2d: 32, kernel 4, stride 2 Conv2d: 64, kernel 3, stride 1 Linear: 512 Linear: number of actions
DQN for VAE	Input: latent space * 4 Linear: 3136 Linear: 3136 Linear: 3136 Linear: 512 Linear: number of actions *Instance norm between all layers*

Table A.2: All final model DQN architectures used in this thesis.

Appendix B

Insight into the Generalisation Ability of VAEs

I mistakenly trained a batch of 5 runs on the game Pong with a VAE trained on the game Breakout. Interestingly, the agent learned to play the game surprisingly well. We can compare this to the blue line which represents a training with a VAE who did not learn to represent the ball. So if the latent variables could not extract enough information, like the ball from the wrong image, we should see something similar to the blue line. However, the orange line is much better which means that even though we are using the wrong VAE, the latent variables have all the necessary information for both games.

Further, we can compare the runs with the green one, where a too small net of three layers was taken as the DQ-net. We see that while the green line flattens out and can no longer store more information in the net, the orange line continues to rise. Together with the red run with a correct VAE we can conclude that it is much harder for the agent to use the variables, but the necessary information is stored in them. Actually, this could be an indication that the training performance does depend on the structure of the latent variables. Since we have shown in Results that the performance does not depend on disentanglement, it must be a different kind of linkage within the latent variables.

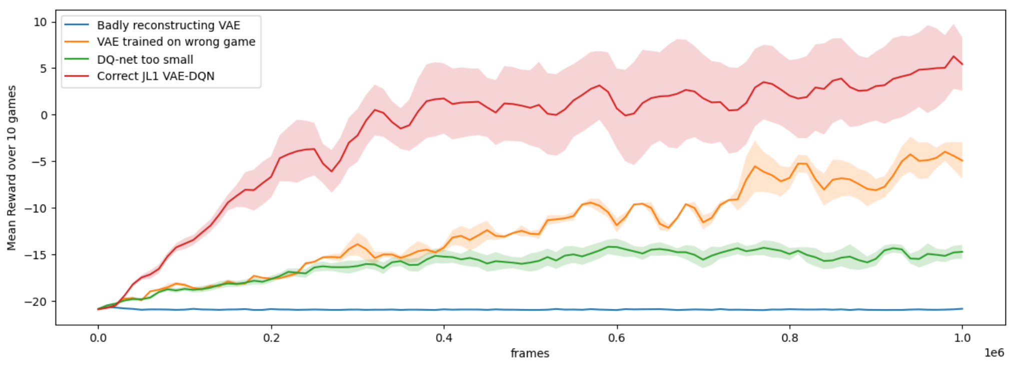


Figure B.1: Four different set of runs trained on the game Pong. The badly reconstructing VAE is a β 5-VAE which did not manage to learn the ball. The too small DQ-net has two linear layers less then our DQ-net has. The orange VAE was trained on the game Breakout and then used in the game of Pong. Finally, the correct JL1-VAE is the same as in the results table.

Appendix C

Simplified Reward Plots and Tables

Game	Model	Mean Max Reward	Absolute Max Reward	Achieved after
Pong	βTC 2-VAE	-4.26	4.3	230k
	βTC 1.1-VAE	-4.8	3.4	230k
	β 5-VAE	-5.5	1.6	234k
	Bad β-VAE	-7.9	0.8	242k
	JL1-VA	**-2.5**	**5**	222k
	Ground Truths	-14.8	-13.1	**166k**
	Original Approach	-3.2	0.5	250k
	Autoencoder	-7.9	-3.6	228k
Breakout	**Bad β-VAE**	**22**	**28**	148k
	β 0.6-VAE	18.4	23	121k
	β0.1 TC 1-VAE	16.4	19	169k
	JL1-VAE	19.6	23	**109k**
	Original Approach	18.2	21	171k
	Autoencoder	19	20	156k

Table C.1: Mean maximal rewards over 5 runs with different random seeds and the absolute maximum achieved by at least a single run during the first 250k frames as well as the number of frames it took to achieve this.

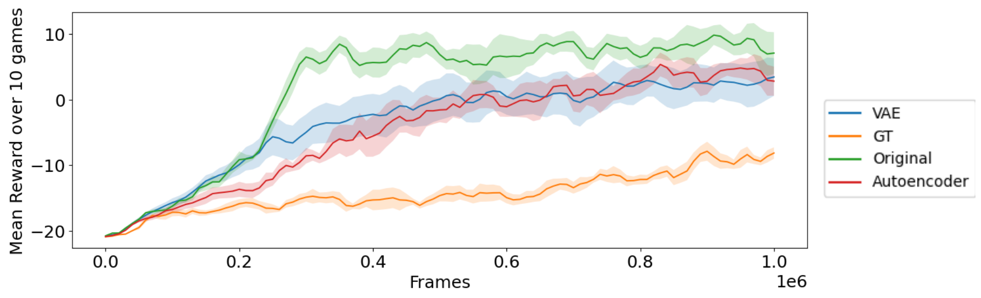


Figure C.1: Combining all VAEs to show the difference between preprocessing with VAEs and AEs for the game Pong.

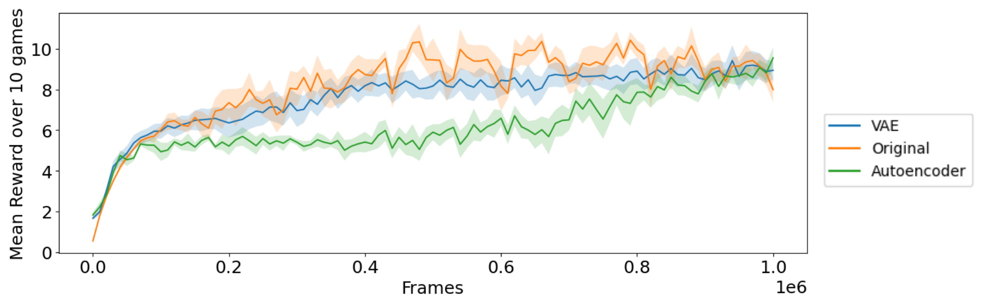


Figure C.2: Combining all VAEs to show the difference between preprocessing with VAEs and AEs for the game Breakout.