Dynamics modeling using a combination of Variational Autoencoders and Einsum Networks

Pálsson, Guðmundur Orri

Award date:
2021

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science

Dynamics modeling using a combination of Variational Autoencoders and Einsum Networks

Master Thesis

Guðmundur Orri Pálsson

Supervisors:
Dr. Cassio de Campos
M.Sc. Alvaro Correia

1.0 version

Eindhoven, September 2021

Abstract

We propose a generative probabilistic model that aims to generate accurate control trajectories of an dynamical environment using only raw images. We base our method on a combination of variational autoencoders, for representation learning of the raw images, and an einsum network, for learning the transitional dynamics of our environment in latent space. We analyse the performance of our model, based on its ability to reconstruct images resembling the real environment, its ability to generate accurate image control trajectories, resembling the real environment and the performance of our model when applied within a deep reinforcement learning framework. Furthermore, we analyse the distribution of our predictions in latent space and we attempt to understand the meaning behind individual latent variables. To analyse the quality of our method, the results were compared with a previous method, Embed to Control (E2C).

Acknowledgements

First and foremost, my sincere thanks to my supervisors, associate professor Dr. Cassio de Campos and doctoral candidate M.Sc. Alvaro Correia. Without their support and guidance, throughout the whole process of writing my thesis, this project would not have been possible. Also, I want to thank Dr. Meng Fang for being a part of my assessment committee and taking his time to assess my thesis.

I want to give special thanks to my girlfriend, Júlía Sif Ólafsdóttir, for supporting me throughout this whole process, encouraging me during difficult times and celebrating with me when things were going well.

Finally, I want to thank my family, friends and fellow students for their love and support throughout the process of writing my thesis during difficult times where face to face contact was not always an option.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

For the past decade, the demand for mobile robots to substitute humans in many fields is ever increasing. Weather their application involves surveillance, planetary exploration, patrolling, emergency rescue operations, industrial automation or medical care, the primary objective is for the robots to be autonomous, meaning they have the ability to determine actions to be taken in an environment based on a perception system [17]. For humans, the main perception system we use to determine actions to take in an environment, is our vision. Through our vision, the human brain can very quickly learn important information about the dynamics of the environment such as how different objects in the environment look and how they behave. Knowing this, gives humans the ability to imagine what will happen in an environment if a certain action is performed without having tried it before. In robotics, this can be a very challenging subject, mainly due to two reasons. On one hand, images are high-dimensional, therefore solving a control problem with only images, is in itself a very complicated task. On the other hand, the real world dynamics of an environment can be very complicated and hard to model. Due to that reason, many previous methods do rely on either the full system model being known or the state-space being low-dimensional. For mobile robots to succeed in applying control on highly complex dynamical systems, they will however ultimately need to work using only raw sensory inputs such as images [21]. In this project we propose a generative probabilistic model that aims to generate accurate control trajectories of a dynamical environment using only raw images observed from the environment. We suggest using a combination of variational autoencoders for representation learning on the raw images as well as an einsum network to learn the transitional dynamics of the environment [14]. We propose two different methods, VAE-Einsum and VAE-Einsum-Decoupled.

Both methods will be compared with a previous method, Embed to Control (E2C), proposed by Manuel Watter et al. [21]. The methods will be analysed in terms of their ability to reconstruct images resembling the real environment, their ability to generate image trajectories resembling the real environment, the difference between the distribution of the latent spaces based on a visualization technique called *t-SNE* as well as their performance within a deep reinforcement learning framework.

Chapter 2

Background

2.1 Variational Autoencoder (VAE)

The framework of variational autoencoders (VAEs) provides a principled method for jointly learning *deep latent-variable* models and corresponding inference models using stochastic gradient descent [8]. The term latent-variable refers to variables that are a part of the model but not observed, and therefor not a part of the dataset. A deep latent variable is therefor a latent variable model whose distributions are parameterized by a neural network [8].

A variational autoencoder generally includes two components, an *encoder* and a *decoder*. The objective of an encoder, q_ϕ, also referred to as a parametric inference model, is to learn a stochastic mapping from an observed x-space, whose empirical distribution is typically complicated, and a latent z-space. The objective of the decoder is then to learn the conditional distribution $p_\theta(x|z)$. The variational parameters ϕ are optimized such that

$$q_\phi(z|x) \approx p_\theta(z|x) \tag{2.1}$$

The optimization objective of the VAE is to minimize the *evidence lower bound*, abbreviated as ELBO and given by

$$\mathcal{L}_{\theta,\phi}(x) = E_{q_\phi(z|x)}[\log p_\theta(x,z) - \log q_\phi(z|x)] \tag{2.2}$$

For any choice of $q_\phi(z|x)$ we have

$$\log p_\theta(x) = E_{q_\phi(z|x)}\left[\log\left[\frac{p_\theta(x,z)}{q_\phi(z|x)}\right]\right] + E_{q_\phi(z|x)}\left[\log\left[\frac{q_\phi(z|x)}{p_\theta(z|x)}\right]\right] \tag{2.3}$$

where the first term is the ElBO and the second term is the Kullback-Leibler (KL) divergence between $q_\phi(z|x)$ and $p_\theta(z|x)$, which is non-negative. Due to the the KL divergence being non-negative and zero if $q_\phi(z|x)$ equals the true posterior distribution, the ELBO is a lower bound on the log-likelihood of the data [8]. There for we have

$$\mathcal{L}_{\theta,\phi}(x) = \log p_\theta(x) - D_{KL}(q_\phi(z|x)||p_\theta(z|x)) \leq \log p_\theta(x) \tag{2.4}$$

The KL divergence therefore determines the gap between ELBO and the marginal likelihood $\log p_\theta(x)$. The better $q_\phi(z|x)$ approximates the true posterior distribution $p_\theta(z|x)$, in terms of the KL divergence, the smaller the gap [8].

By looking at equation 2.4, the maximization of the ELBO $\mathcal{L}_{\theta,\phi}(x)$ with respect to parameters θ and ϕ, will therefor concurrently optimize two things.

- Maximize the marginal likelihood $p_\theta(x)$, meaning that our decoder will become better.
- Minimize the KL divergence of the approximation $q_\phi(z|x)$ from the true posterior $p_\theta(z|x)$, so our encoder becomes better.

In addition to efficient latent-variable inference, the VAE framework also allows for both discrete and continuous observed variables [8].

2.2 β-VAE

Irina Higgins et al. [4] introduce a modification of the variational autoencoder framework by introducing a hyperparameter β that balances latent channel capacity and idependance constraints [4].

The authors suggest that learning a disentangled representation of generative factors in the data can be useful for a large variety of tasks and domains. A disentangled representation can be defined as one where single latent units are sensitive to changes in single generative factors, while being relatively invariant to changes in other factors [4].

The solution involves modifying the VAE framework with a single hyperparameter β that modulates the learning constraints applied to the model. These constraints impose a limit on the capacity of the latent information channel and control the emphasis on learning statistically independent latent factors [4].

The authors present the following objective function for the β-VAE

$$L_{\theta,\phi}(x) = E_{q_\phi(z|x)}[log_{p_\theta}(x,z) - \beta D_{KL}(q_\phi(z|x)||p_\theta(z))] \tag{2.5}$$

where $\beta = 1$ corresponds to the original VAE framework. With $\beta > 1$ the model is pushed to learn a more efficient latent representation of the data, which is disentangled if the data contains at least some underlying factors of variation that are independent [9].

The β-VAE was trained on a variety of datasets such as celebA [10], chairs [1] and faces [13]. Overall the β-VAE tends to consistently and robustly discover more latent factors and learn cleaner disentangled representations of them compared to previous methods such as InfoGAN [3] or DC-IGN [9].

2.3 Deep Q-Network (DQN)

Melros Roderick et al. [16] give a detailed description into their implementation of a Deep Q-Network (DQN) but the network was trained and evaluated on Atari games.

The DQN differs from regular Q-learning algorithms in mostly three ways. Firstly, instead of a tabular representation of the Q function the DQN uses a function approximator. Secondly, mini-batches of random training data are used to update the parameters of the network, rather then single-step updates on the last experience. Thirdly, older network parameters are used to estimate the Q-value for each next state.

To sample mini-batches of previous observations from the environment the DQN stores each transition in an experience replay buffer. Within the experience replay buffer, the DQN stores a set of five-tuple (s, a, s', r, T) corresponding to an agent taking an action a in state s, arriving in state s' and receiving reward r. To provide the network with a stable training target, the DQN uses older network parameters to estimate the Q-value for each next state.

The following pseudocode shows the general implementation of the DQN algorithm.

Algorithm 1: Deep Q-learning with experience replay

```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with random weights $\theta^- = \theta$
for episode $1, M$ do
    Initialize sequence $s_1 = \{x_1\}$ and preprocess sequence $\phi 1 = \phi(s_1)$
    for $t = 1, T$ do
        With probability $\epsilon$ select a random action $a_t$
        otherwise select $a_t = argmax_a Q(\phi(s_t), a; \theta)$
        Execute action $a_t$ in the emulator and observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store experience $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
        Sample random minibatch of experiences $(\phi_j, a_j, r_t, \phi_{j+1})$ from $D$
        set $y_j = \begin{cases} r_j, & \text{if episode terminates at step } j+1 \\ r_j + \gamma max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-), & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j, \theta))^2$ with respect to the weights $\theta$
        Every $C$ step reset $\hat{Q} = Q$
    end
end
```

Additionally, Melros Roderick et al. [16] address a common problem in DQN where the performance of the agent fluctuates between iterations. Therefor the agent might reach a high average number of rewards and then suddenly drop. The authors refer to this as *catastrophic forgetting.* One way of addressing this problem is to save the network parameters that resulted in the best test performance.

2.4 Model-based RL for Atari

L. Kaiser et al. [7] explore how learned video models can enable learning in the Atari Learning Environment (ALE) benchmark. According to the authors, no prior work has successfully demonstrated model-based control via predictive models that achieve competitive results with model-free RL.

The goal of the algorithm is to find a policy which maximizes the expected reward. Apart from using the Atari emulator environment *env* the algorithm will use a neural network simulated environment env', referred to as the *world model.* The world model will share the action space and reward space with *env* and produce visual observations in the same format. The initial data used to train env' comes from random rollouts of *env*. The following pseudocode shows the iterative method used by the algorithm [7].

Algorithm 2: Pseudocode for the model-based RL algorithm [7]

```
Initialize policy $\pi$
Initialize model parameters $\theta$ of $env'$
Initialize empty set $D$
while not done do
    Collect observations from real env
    $D \leftarrow D \cup COLLECT(env, \pi)$
    Update model using collected data
    $\theta \leftarrow$ TRAIN_SUPERVISED($env', D$)
    Update policy using world model
    $\pi \leftarrow$ TRAIN_RL($\pi, env'$)
end
```

The world model was constructed using a convolutional feed forward network. The input consist of four consecutive game frames and an action a. The network outputs the next frame of the game and the value of the reward. For the policy training, the algorithm uses proximal policy optimization (PPO). The algorithm generates rollouts of env' and uses them to improve the policy [7].

The algorithm was evaluated on 26 games selected on the basis of being solvable with state-of-the-art model-free RL algorithms. The results showed that the algorithm outperforms the model-free algorithms in terms of learning speed on nearly all games. In terms of limitations of the algorithm, its performance varied substantially between different runs on the same game and overall, the final scores were lower compared to the model-free methods [7].

2.5 Deep Reinforcement Learning with Double Q-Learning

Hado van Hasselt et al. [20] discuss the problem with the Q-learning algorithm in terms of overestimating action values under certain conditions. In previous work, overestimation has been linked to insufficiently flexible function approximation and noise. To test this overestimation the authors test the performance of a recent DQN algorithm. The DQN algorithm combines Q-learning with a deep neural network. Even in the setting of DQN, the authors show that the algorithm substantially overestimate the values of the actions. Alternatively, the authors propose a Double Q-learning algorithm. In the Double Q-learning algorithm, two value functions are learned by assigning experiences randomly to update one of the two value functions, resulting in two sets of weights θ and θ'. For each update, one set of weights is used to determine the greedy policy and the other to determine its value.

The idea of the Double Q-learning algorithm is to reduce over-estimations by decomposing the max operation in the target into action selection and action evaluation. The algorithm was compared to previous DQN algorithms in the setting of playing Atari 2600 games.

The results show the overestimation of the DQN in six Atari games. Furthermore the results show that the Double DQN algorithm finds better policies, obtaining new state-of-the-art results on the Atari 2600 domain [20].

2.6 The Effect of Planning Shape on Dyna-style Planning in High-dimensional State Spaces

G. Zacharias Holland et al. [6] discuss a reinforcement learning planning method referred to as Dyna-style planning. Dyna-style planning is potentially a powerful approach in large-scale problems. By indirectly influencing the agent's behaviour via the value function or policy, the computationally expensive process of planning can be asynchronous with the agent's decision-making loop, retaining a model-free agent's ability to operate on a fine-grained timescale.

The authors experiment with two different value function learners. One was based on the Sarsa algorithm, using linear value function approximation with Blob-PROST features and the other was Deep Q-Networks. The DQN used for the experiment used a deep convolutional neural network to approximate the value function as well as an experience replay buffer to collect transitions used to update the model [6].

The DQN model was extended by incorporating the Dyna architecture. This approach is called Dyna-DQN. After each step taken in the environment, a number of start states for planning are sampled from a planning buffer containing the agent's recent real experience. For each start state, an action is selected using the agents current policy and the model is used to simulate a single transition which is placed into the experience replay buffer alongside the transition observed from the real environment [6].

As an alternative to the Dyna-DQN the authors suggest planning with longer rollouts instead of just one step. They hypothesize that it may be possible to generate a more diverse experience by rolling out more than a single step from the start state during planning. Since each step in the rollout is generated using the current policy some trajectories might have changed from what was originally observed. This algorithm is called Rollout-Dyna-DQN, using the parameter k to determine the length of the rollout. Therefor if $k = 1$, it's Dyna-DQN [6].

Some different shapes of rollouts were experimented with, such as 100 rollouts of size 1, 10 rollouts of size 10 or 1 rollout of size 100. Each rollout shape requires the same amount of computation from the model and the experience generated during a multi-step rollout is still recorded as single transitions in the replay-buffer.

Dyna-DQN and Rollout-Dyna-DQN were both compared on different games including Asterix, Beam Rider, Space Invaders and Ms. Pac-Man. In all of the games, Rollout-Dyna-DQN scored higher the Dyna-DQN but the score between different rollout shapes varied between games.

2.7 Probabilistic Circuits

Probabilistic circuits are a family of probabilistic models which allow a wide range of exact and efficient inference routines [14]. Robert Peharz et al. [14] presents the following definition of a probabilistic circuit.

Definition (Probabilistic Circuit). *Given a set of random variables X, a probabilistic circuit (PC) $\mathcal{P}$ is a tuple $(\mathcal{G}, \psi)$, where $\mathcal{G}$ denoted as computational graph is a directed acyclic graph (DAG) (V,E) and $\psi : V \to 2^X$, denoted as scope function, is a function assigned a scope to each node in V, i.e. a sub-set of X. For internal nodes of G, i.e. any node $N \in V$ which has children, the scope function satisfies $\psi(V) = \cup_{N \in ch(N)} \psi(N')$. A leaf of $\mathcal{G}$ computes a probability density over its scope $\psi(L)$. All internal nodes of G are either sum nodes (S) or product nodes (P). A sum node S computes a convex combination of its children, i.e. $S = \sum_{N \in ch(N)} \omega_{S,N} N$, where $\sum_{N \in ch(N)} \omega_{S,N} = 1$, and $\forall N \in ch(S) : \omega_{S,N} \geq 0$. A product node P computes a product of its children, i.e. $P = \prod_{N \in ch(N)} N$* [14].

PCs can be seen as a special kind of neural network. The first layer computes non-linear functions, or probability densities, over sub-sets X, and all internal nodes compute either weighted sums or products. The output of a PC is the value of one or more selected nodes in the graph G [14].

Two interesting attributes of PCs is that they can be both *decomposable* and *smooth*. PCs are decomposable in the sense that for each product node $P \in V$ it holds that $\psi(N) \cap \psi(N') = \emptyset$, for $N, N' \in ch(P), N \neq N'$ and the consequence of decomposability is that integrals can be computed in linear time of the circuit size. The smoothness on the other hand means that for each sum node $S \in V$ it holds that $\psi(N) = \psi(N')$ for $N, N' \in ch(S)$. The smoothness has little computational advantage on the PCs but it leads to a well-defined probabilistic interpretation [14].

Members of the PC family include *sum-product-networks (SPNs), cutset-networks (CNs), probabilistic sentential decision diagrams (PSDDs)* and *arithmetic circuits (ACs)* [14].

2.7.1 Sum-Product Network (SPN)

A sum-product network consists of a directed graph that represents a probability distribution combined in the form of sum nodes and product nodes. Like arithmetric circuits, SPNs can be built by transforming a probabilistic graphical model such as a Bayesian network or a Markov network but they can also be learned from data. One of the main advantages of SPNs is that several inference tasks can be performed in time proportional to the number of links in the graph [12]. The following figure from a survey on SPNs by Iage París et al. [12] shows the comparison between a Bayesian network and a SPN.

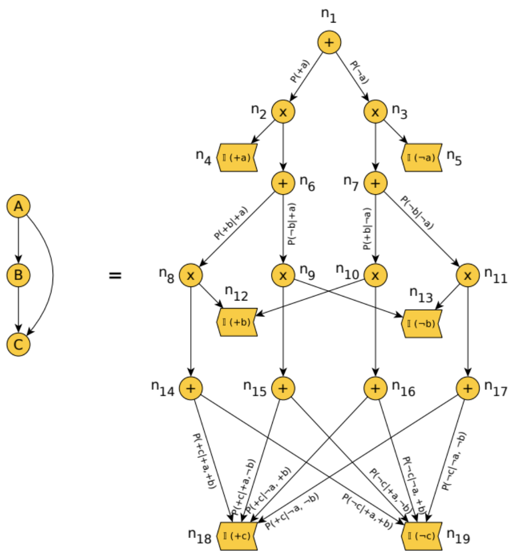


Figure 2.1: Bayesian network (left) and SPN (right). The terminal nodes in the SPN are indicators for the 3 variables in the model, A, B and C. The root node, n_1, computes the joint and marginal probabilities [12]

SPNs can be seen as a particular type of feed-forward neural network however the main difference being that SPNs have a probabilistic interpretation while standard NNs do not. Inference is also different and finding the most probable explanation (MPE) requires a backtrack from the root to the leaves [12].

2.7.2 Einsum Networks

A way to make PCs denser and thus more efficient is to vectorize them, meaning that we redefine a leaf L to be a vector of K densities over $\psi(L)$ instead of a single density. Therefor, a leaf computing Gaussian density is replaced by a vector $[\mathcal{N}(\cdot|\theta_1), ..., \mathcal{N}(\cdot|\theta_K)]^T$, each $\mathcal{N}(\cdot|\theta_K)$ being a Gaussian over $\psi(L)$, equipped with private parameters θ_k. Furthermore, a product node is re-defined to be an outer product containing the products of all possible combinations of densities coming from the child vectors and sum nodes are re-defined to be a vector of K weighted sums, where each individual sum operation has its private weights and computes a convex combination over all the densities computed by its children [14].

Basic Einsum Operation

The core computational unit in Einsum Networks is the vectorized PC shown in figure 2.2 [14].

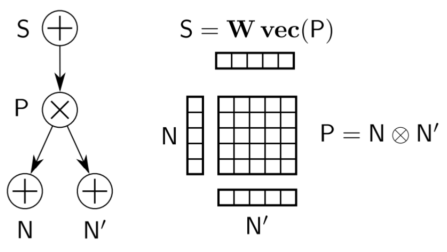


Figure 2.2: Basic einsum operation: A sum node S with a single child P which itself has 2 children. All nodes are vectorized [14]

Figure 2.2 shows a sum node S with a single product child P. The product node has two children N and N' where each one computes a vector of K densities. Mathematically this can be expressed in *Einstein notation*:

$$S_k = W_{kij} N_i N'_j \tag{2.6}$$

where W is re-shaped into a $K \times K \times K$ element wise non-negative tensor. The indicies i, j, k label the axes of N, N' and W [14].

Einsum layer

Instead of computing single vectorized sums, whole layers can be computed in parallel. First the PC is organized in a layer-wise structure. The PC is traversed top-down and a topologically sorted list of the layers of nodes, alternating between sum and products and starting with the leaf nodes [14].

All nodes in any layer i depend only on nodes in layers strictly smaller than i. Furthermore a product layer will contain exactly the inputs to the sum layer. The strategy of Einsum Networks is then to perform efficient parallel computations for the whole leaf layer and the whole sum layer in each pair of consecutive product and sum layers. The result is a matrix of log-densities, with as many rows as there are leaves [14].

The leaves of Einsum Networks compute log-densities of an exponential family which has the form $L = \log h(x) + T(X)^T \theta - A(\theta)$ where θ are the natural parameters, h is the base measure, T is the sufficients statistics and A is the log-normalizer. Many parametric distributions can be expressed as exponential families such as Gaussian, Binomial and Categorical [14].

Expectation-Maximization (EM)

A natural way to learn PCs is the EM algorithm, known for rapidly increasing the likelihood, especially in early iterations. Since Einsum Networks represent all probability values in the log-domain, the PC output is actually $\log \mathcal{P}(x)$ instead of $\mathcal{P}(x)$. Therefor calling automatic differentiation on $\log \mathcal{P}(x)$ yields the following derivative for each sum weight

$$\omega_{S,N} : \frac{\partial \log \mathcal{P}}{\omega_{S,N}} = \frac{1}{\mathcal{P}} \frac{\partial \mathcal{P}}{\partial \mathcal{S}} \frac{\partial S}{\partial \omega_{S,N}} = \frac{1}{\mathcal{P}} \frac{\partial \mathcal{P}}{\partial S} N \tag{2.7}$$

The classical EM algorithm uses a whole pass over the training set for a single update. For computational purposes it is however possible to define a stochastic version for EM where the sums over the entire dataset are replaced with sums over mini-batches. The full EM update is then replaced with gliding averages [14].

Experimental Results

The Einsum Network was tested as a generative model for street-view house numbers (SVHN) [11]. The total training set included $581k$ images of size 28×28 pixels and the network was trained on the image-tailored structure proposed by Hoifung Poon et al. [15], referred to as PD structure. Using PD structure the images are recursively decomposed into sub-rectangles using axis-aligned splits, displaced by a certain step size Δ [14].

The dataset was clustered into 100 clusters using the *sklearn* implementation of k-means and then an Einsum Network for each cluster was learned. These 100 Einsum Networks were then used as mixture components of a mixture model, using the cluster proportions as mixture coefficients [14].

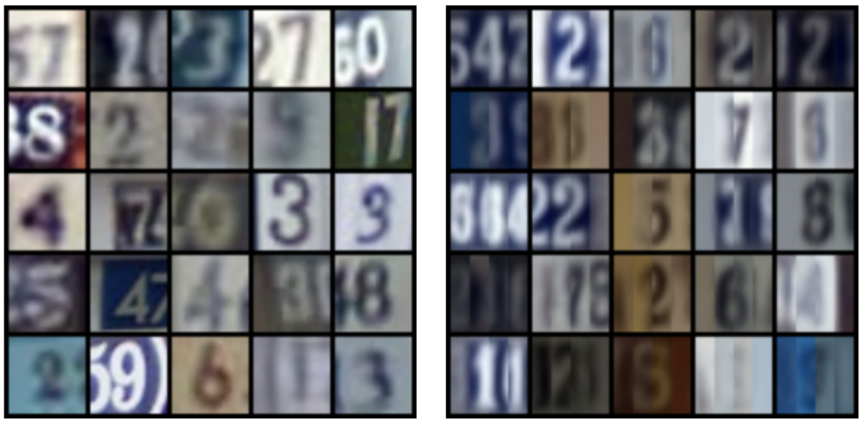


Figure 2.3: Real SVHN samples(left), samples generated by Einsum Network (right) [14]

Figure 2.3 shows the original samples as well as the Einsum Network generated samples. For each Einsum Network component a step size of $\Delta = 8$ was used for the SVHN dataset. For the leaves, Gaussians with a diagonal covariance matrix where factorized over the RGB channels. The vector length for the sums and leaves was set to $K = 40$. After each EM update the Gaussian variances were projected to the interval $[10^{-6}, 10^{-2}]$ corresponding to a maximal standard deviation of 0.1. Each component was trained for 25 epocs, using a batch size of 500 and EM stepsize of 0.5 [14].

2.8 Visualization using t-SNE

Laurens van der Maaten et al. [19] present a technique called *t-SNE* or *t-Distributed Stochastic Neighbor Embedding* that visualizes high-dimensional data by giving each datapoint a location in a two or three dimensional map. The technique is an extension to *SNE* which was presented by Hinton and Roweis in 2003 [5]. SNE starts by converting the high-dimensional Euclidean distances between datapoints into conditional probabilities that represents similarities. The similarity of datapoint x_j to datapoint x_i is the conditional probability, $p_{j|i}$, that x_i would pick x_j as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian centered at x_i [5].

The probability $p_{i|j}$ is given by the following equation

$$p_{i|j} = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k\neq i} exp(-||x_i - x_k||^2/2\sigma_i^2)} \tag{2.8}$$

where σ_i is the variance of the Gaussian that is centered on datapoint x_i. For the low-dimensional counterpars y_i and y_j of the high-dimensional datapoints x_i and x_j it is possible to compute similar conditional probability $q_{i|j}$ [5]. The following equation gives $q_{i|j}$

$$q_{i|j} = \frac{exp(-||y_i - y_j||^2)}{\sum_{k\neq i} exp(-||y_i - y_k||^2)} \tag{2.9}$$

If the map points y_i and y_j correctly model the similarity between the high-dimensional datapoints x_i and x_j the conditional probabilities $p_{j|i}$ and $q_{j|i}$ will be equal [5]. To minimize the mismatch between $p_{j|i}$ and $q_{j|i}$, SNE minimizes the sum of Kullback-Leibler divergences over all datapoints using a gradient descent method with a cost function C.

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{i|j} log \frac{p_{j|i}}{q_{j|i}} \tag{2.10}$$

where P_i represents the conditional probability distribution over all other datapoints given datapoint x_i, and Q_i represents the conditional probability distribution over all other map points given map point y_i [5].

Even though SNE constructs reasonably good visualizations, it is hampered by a cost function that is difficult to optimize and by a problem referred to as the *crowding problem* [19]. The t-SNE aims to solve these problems by introducing a new cost function that differs from the one used by SNE in two ways. On one hand it uses a symmetrized version of the SNE cost function with simpler gradients. On the other hand it uses a Student-t distribution rather than a Gaussian to compute the similarity between two points in the low-dimensional space [19]. Instead of minimizing the Kullbeack-Leibler divergence between the conditional probabilities, $p_{j|i}$ and $q_{j|i}$, it is also possible to minimize a single Kullback-Leibler divergence between a joint probability distribution, P, in the high-dimensional space and a joint probability distribution, Q, in the low-dimensional space [19].

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}} \tag{2.11}$$

The pairwise similarities in the low-dimensional map q_{ij} is given by

$$q_{ij} = \frac{exp(-||y_i - y_j||^2)}{\sum_{k \neq l} exp(-||y_k - y_l||^2)} \tag{2.12}$$

and in the high-dimensional space, p_{ij} is defined by

$$p_{ij} = \frac{exp(-||x_i - x_j||^2/2\sigma^2)}{\sum_{k \neq l} exp(-||x_k - x_l||^2/2\sigma^2)} \tag{2.13}$$

If x_i is an outlier the values of p_{ij} are very small for all j so the location of its low-dimensional map point y_i has very little effect on the cost function. To solve this problem, the joint probabilities p_{ij} in the high-dimensional space are defined to be symmetrized conditional probabilities $p_{ij} = \frac{p_{j|i}+p_{i|j}}{2n}$. This ensures that $\sum_j p_{ij} > \frac{1}{2n}$ for all datapoints x_i, so each datapoint x_i makes a significant contribution to the cost function [19]. Additionally, the gradients of the symmetric SNE is given by

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j) \tag{2.14}$$

Furthermore, t-SNE uses a Student t-distribution with one degree of freedom as the heavy-tailed distribution in the low-dimensional map [19], therefor defining the joint probabilities q_{ij} as

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)}{\sum_{k \neq 1} (1 + ||y_k - y_l||^2)^{-1}} \tag{2.15}$$

The justification for using the Student t-distribution is that it closely relates to the Gaussian distribution, as the Student t-distribution is an infinite mixture of Gaussians. The computationally convenient property is that it is much faster to evaluate a density of a point under a Student t-distribution than under a Gaussion since it does not involve an exponential [19]. The gradients of the Kullback-Leibler divergence between P and the Student-t based joint probability distribution Q is therefor given by

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1} \tag{2.16}$$

The t-SNE method was demonstrated on the MNIST dataset using a PCA to reduce the dimensionality of the data. The class of each datapoint was only used to color the points in the figure [19].

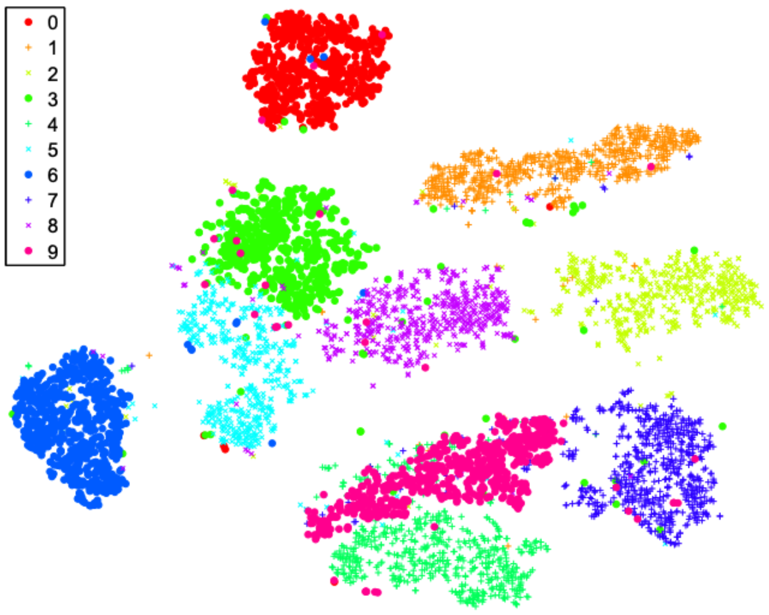


Figure 2.4: 2-dimensional mapping of the MNIST dataset [19]

Figure 2.4 shows images belonging to the same class close together with some exceptions. Additionally, the different clusters are spread out relatively to their similarities.

Chapter 3

Previous Work

3.1 Embed to Control (E2C)

For algorithms capable of solving complex dynamical systems from raw sensory data Manuel Watter et al. [21] introduce Embed to Control (E2C), a method for model learning and control of non-linear dynamical systems from raw pixel images. The E2C consists of a deep generative model from the family of variational autoencoders that learns to generate image trajectories from a latent space in which the dynamics are constrained to be locally linear.

3.1.1 Problem Formulation

Considering the control of unknown dynamical systems of the following form

$$s_{t+1} = f(s_t, u_t) + \xi, \xi \sim \mathcal{N}(0, \textstyle\sum_\xi) \tag{3.1}$$

where t denotes the time steps, $s_t \in R^{n_s}$ the system state, $u_t \in R^{n_u}$ the applied control and ξ the system noise. The function f represents the system dynamics. The goal is to infer a low-dimensional latent space model in which optimal control can be performed [21].

The idea is for the model to learn a function m such that a high-dimensional image x_t can be mapped to a low-dimensional vector z_t such that the control problem can be solved using z_t instead of x_t [21].

Therefor we have

$$z_t = m(x_t) + \omega, \omega \sim \mathcal{N}(0, \textstyle\sum_\omega) \tag{3.2}$$

where ω accounts for system noise and $z_t \sim \mathcal{N}(m(x_t), \sum_\omega)$ [21]. Assuming locally optimal control in latent spaces where we have an inferred latent state $z_t \in R^{n_z}$ from an image x_t of state s_t and a given dynamics function $f^{lat}(z_t, u_t)$, we can assume the transition dynamics in latent space to be $z_{t+1} = f^{lat}(z_t, u_t)$ [21].

The next step is to learn an appropriate low-dimensional latent representation of x_t

$$z_t \sim P(Z_t | m(x_t), \textstyle\sum_\omega)$$

To solve our control problem using latent spaces, z_t, has to have the following properties [21].

1. z_t must capture sufficient information about x_t.
2. z_t must allow for accurate predictions of the next latent state z_{t+1}.
3. The prediction f^{lat} of the next latent state must be locally linearizable for all valid control magnitudes u_t.

Instead of learning a latent space z and a transition model f^{lat} and combining them with a stochastic optimal control algorithm, the desired transformation properties are directly imposed on the representation z_t during learning [21].

Following equation 3.2 we let the latent representation be Gaussian $P(Z|X) = \mathcal{N}(m(x_t), \sum_\omega)$. To infer z_t from x_t we resort to sampling z_t from an approximate posterior distribution $Q_\phi(Z|X)$ with parameters ϕ.

Given a reference trajectory $\bar{Z}_{1:T}$ the current estimate for the optimal trajectory, together with corresponding controls $\bar{u}_{1:T}$ the system is linearized as

$$z_{t+1} = A(\bar{z}_t)z_t + B(\bar{z}_t)u_t + o(\bar{z}_t) + \omega, \omega \sim \mathcal{N}(0, \textstyle\sum_\omega) \tag{3.3}$$

where

$$\begin{aligned} A(\bar{z}_t) &= \frac{\delta f^{lat}(\bar{z}_t, \bar{u}_t)}{\delta \bar{z}_t} \\ B(\bar{z}_t) &= \frac{\delta f^{lat}(\bar{z}_t, \bar{u}_t)}{\delta \bar{u}_t} \end{aligned} \tag{3.4}$$

are local Jacobians and $o(\hat{z}_t)$ is an offset [21].

3.1.2 Inference Model for Q_ϕ

We define the diagonal Gaussian distribution $Q_\phi(Z|X) = \mathcal{N}(\mu_t, diag(\sigma^2))$ whose mean $\mu_t \in R^{n_z}$ and covariance $\sum_t = diag(\sigma^2) \in R^{n_z \times n_z}$ are computed by an encoding neural network with outputs

$$\mu_t = W_\mu h_\phi^{enc}(x_t) + b_\mu \tag{3.5}$$

$$\log \sigma_t = W_\mu h_\phi^{enc}(x_t) + b_\sigma \tag{3.6}$$

where $h_\phi^{enc} \in R^{n_e}$ is the activation of the last hidden layer and ϕ is given by the set of all learnable parameters of the encoding network, including the weight matrices W_μ, W_σ and biases b_μ, b_σ. Parameterizing the mean and variance of a Gaussian distribution based on a neural network gives a natural and expressive model for our latent space [21].

3.1.3 Generative Model for P_θ

Using the approximate posterior distribution, Q_ϕ, samples $\tilde{x}_t$ and $\tilde{x}_{t+1}$ from latent samples z_t and z_{t+1} are generated by enforcing a locally linear relationship in the latent space following equation 3.3. The generative model can therefor be formalized as

$$\begin{aligned} z_t &\sim Q_\phi(Z|X) = \mathcal{N}(\mu_t, \textstyle\sum_t) \\ \hat{z}_{t+1} &\sim \hat{Q}_\psi(\hat{Z}|Z, u) = \mathcal{N}(A_t\mu_t + B_t u_t + C_t o_t) \\ \hat{x}_t, \hat{x}_{t+1} &\sim P_\theta(X|Z) = Bernoulli(p_t) \end{aligned} \tag{3.7}$$

Reconstruction of an image from z_t is performed by passing a sample through multiple hidden layers of a decoding neural network which computes the mean p_t of generative Bernoulli distribution $P_\theta(X|Z)$ [21].

3.1.4 Transition Model for $\hat{Q}_\psi$

The linearization matrices $A_t \in R^{n_z \times n_z}$, $B_t \in R^{n_z \times n_u}$ and offset $o_t \in R^{n_z}$ are predicted following the same approach as for the distribution means and covariance matrices. All local transformation parameters from sample z_t are predicted based on the hidden represenation $h_\psi^{trans}(z_t) \in R^{n_t}$ of a third neural network with parameters ψ. We parameterize the transformation matrices and offset as

$$\begin{aligned} vec[A_t] &= W_A h_\psi^{trans}(z_t) + b_A \\ vec[B_t] &= W_B h_\psi^{trans}(z_t) + b_B \\ o_t &= W_o h_\psi^{trans}(z_t) + b_o \end{aligned} \tag{3.8}$$

where vec denotes vectorization and therefor $vec[A_t] \in R^{n_z^2}$ and $vec[B_t] \in R^{n_z \times n_u}$ [21].

3.1.5 Learning

The model was trained on a dataset containing observation tuples with corresponding controls obtained from interaction with the environment. The complete loss function is given by

$$\mathcal{L}(\mathcal{D}) = \sum_{(x_t,u_t,x_{t+1})\in\mathcal{D}} \mathcal{L}^{bound}(x_t,u_t,x_{t+1}) + \lambda KL(\hat{Q}_\psi(\hat{Z}|\mu_t,u_t)||Q_\phi(Z|x_{t+1})) \tag{3.9}$$

The first part of the loss is the per-example variational bound on the log-likelihood and the second part is an additional KL divergence contradiction term with weight λ that enforces agreement between the transition and inference models.

The per-example variational bound on the log-likelihood can be formalized as follows

$$\mathcal{L}(x_t,u_t,x_{t+1}) = E_{z_t\sim Q_\phi,\hat{z}_t\sim_\psi}[-\log P_\theta(x_t|z_t) - \log P_\theta(x_{t+1}|\hat{z_{t+1}})] + KL(Q_\psi||P(Z)) \tag{3.10}$$

where Q_ϕ, P_θ and $\hat{Q}_\psi$ are the parametric inference, generative and transition distributions and $P(Z_t)$ is a prior on the approximate posterior Q_ϕ, which we always chose to be an isotropic Gaussian distribution with mean zero and unit variance [21].

The parameters of the model are learned by minimizing $\mathcal{L}(\mathcal{D})$ using stochastic gradient descent and the expectation in equation 3.9 is approximated via sampling [21].

3.1.6 Environments

The model was evaluated on four visual tasks.

1. **Control in a planar system**: An agent can move in a bounded two-dimensional plane by choosing a continuous offset in x- and y-direction. The high-dimensional representation of a state is a 40×40 black-and-white images. The task is to move to the bottom right of the image, starting from a random position x at the top of the image while avoiding six circular obstacles. The encodings of the obstacles are obtained prior to planning and an additional quadratic cost term is penalizing proximity to them.

2. **Swing-up for an inverted pendulum**: The goal of this task is to swing-up and balance an underactuated pendulum from a resting position. A depiction of the state is created by rendering a fixed length line starting from the center of the image at an angle corresponding to the pendulum position. The algorithm faces two additional difficulties in this environment. Firstly, the observed space is non-Markov, as the angular velocity cannot be inferred from a single image. Secondly, discretization errors due to rendering pendulum angles as small as 48×48 pixel images make exact control difficult.

3. **Cart-pole balancing**: The goal of the cartpole system is to balance a pole that is attached by an un-actuated joint to a cart which moves along a frictionless track. The control problem involves deciding at each time step weather a force should be applied on the left or right side of the cart. The objective is to prevent the pole from falling below 15 degrees from vertical or moving more than 2.4 units from the center.

4. **Simulated robotic arm**: The goal of the simulated robotic arm environment is to control a three link robotic arm.

3.1.7 Experimental Setup

Two different types of networks were used for the control problems mentioned above.

- **Standard fully connected neural network** with up to three layers was used for the planar system as well as the inverted pendulum swing-up.
- **Deep convolutional network** for the encoder and an up-convolutional network as the decoder was used for the CartPole and Simulated robotic arm environments.

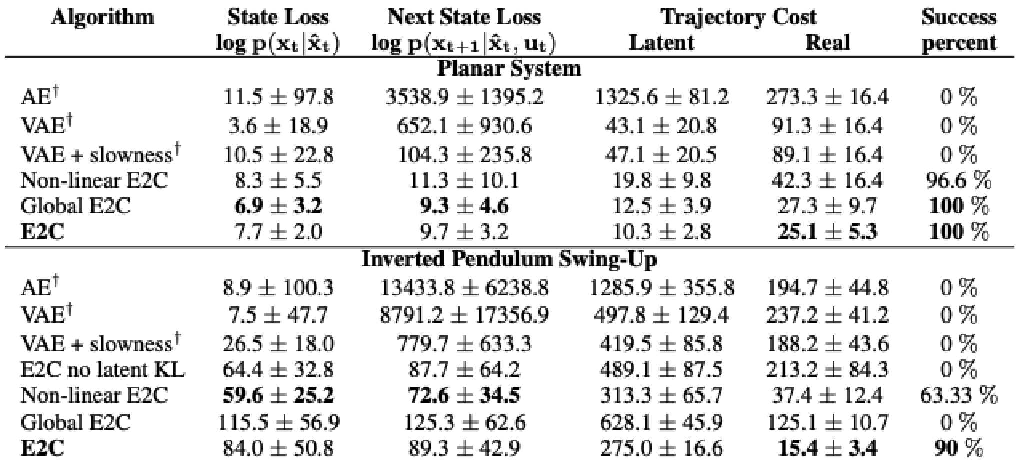
For comparison, the E2C method was compared with a standard variational autoencoder (VAE) and a deep autoencoder (AE) that were trained on the subtasks for visual problems. For the VAE and AE the action is removed from the training set, disregarding any context between images [21].

To perform optimal control in the latent space of different models, two trajectory optimization algorithms were used, *iterative linear quadratic regulation (iLQR)* and *approximate inference control (AICO)* [21].

3.1.8 Results

Control in planar system and inverted pendulum swing-up

Manuel Watter et al. [21] determined the long-term accuracy of the latent space trajectory by accumulating latent and real trajectory costs to quantify how well the imagined trajectory reflects reality. Figure 3.1 shows the state and next state loss as well as the trajectory cost and success percent of E2C compared with the benchmarks [21].



Algorithm	State Loss $\log p(x_t \mid \hat{x}_t)$	Next State Loss $\log p(x_{t+1} \mid \hat{x}_t, u_t)$	Trajectory Cost Latent	Trajectory Cost Real	Success percent
			Planar System		
AE†	11.5 ± 97.8	3538.9 ± 1395.2	1325.6 ± 81.2	273.3 ± 16.4	0 %
VAE†	3.6 ± 18.9	652.1 ± 930.6	43.1 ± 20.8	91.3 ± 16.4	0 %
VAE + slowness†	10.5 ± 22.8	104.3 ± 235.8	47.1 ± 20.5	89.1 ± 16.4	0 %
Non-linear E2C	8.3 ± 5.5	11.3 ± 10.1	19.8 ± 9.8	42.3 ± 16.4	96.6 %
Global E2C	$\mathbf{6.9 \pm 3.2}$	$\mathbf{9.3 \pm 4.6}$	12.5 ± 3.9	27.3 ± 9.7	**100** %
E2C	7.7 ± 2.0	9.7 ± 3.2	10.3 ± 2.8	$\mathbf{25.1 \pm 5.3}$	**100** %
			Inverted Pendulum Swing-Up		
AE†	8.9 ± 100.3	13433.8 ± 6238.8	1285.9 ± 355.8	194.7 ± 44.8	0 %
VAE†	7.5 ± 47.7	8791.2 ± 17356.9	497.8 ± 129.4	237.2 ± 41.2	0 %
VAE + slowness†	26.5 ± 18.0	779.7 ± 633.3	419.5 ± 85.8	188.2 ± 43.6	0 %
E2C no latent KL	64.4 ± 32.8	87.7 ± 64.2	489.1 ± 87.5	213.2 ± 84.3	0 %
Non-linear E2C	$\mathbf{59.6 \pm 25.2}$	$\mathbf{72.6 \pm 34.5}$	313.3 ± 65.7	37.4 ± 12.4	63.33 %
Global E2C	115.5 ± 56.9	125.3 ± 62.6	628.1 ± 45.9	125.1 ± 10.7	0 %
E2C	84.0 ± 50.8	89.3 ± 42.9	275.0 ± 16.6	$\mathbf{15.4 \pm 3.4}$	**90** %

Figure 3.1: Comparison of different approaches to model learning from raw pixels for the planar and pendulum systems [21]

The success was defined as reaching the goal state and staying ϵ-close to it for the rest of the trajectory. While the E2C is not the best in terms of reconstruction performance, it is the only model resulting in a stable swing-up and balance behaviour. The failure of the other models is explained with the fact that the non-linear latent dynamics model cannot be guaranteed to be linearizable for all control magnitudes, resulting in undesired behaviour around unstable fixpoints of the real system dynamics [21].

Balancing a cart-pole and controlling a simulated robot arm

As mentioned in section 3.1.7 deep convolutional and up-convolutional networks were used for the cart-pole and simulated robot arm environments. As in the previous experiments the E2C model has no problem finding a locally linear embedding of images into latent space in which control can be performed. Furthermore, the cost for trajectories obtained by the E2C are only slightly worse than trajectories obtained by AICO operating on the real system dynamics starting from the same start-state [21].

Chapter 4

Methodology

4.1 Problem statement

Assuming a dynamical system of the form

$$s_{t+1} = f(s_t, u_t) + \xi, \xi \sim N(0, \sum_{\xi}) \tag{4.1}$$

where t denotes the time step, $s_t \in R^{n_z}$ the system state, $u_t \in R^{n_u}$ the applied control and ξ the system noise [21]. The function $f(s_t, u_t)$ describes the real system dynamics. The goal of the project is to implement a method that can accurately learn a function $f(x_t, u_t)$ where x_t is a visual representation, or an image, of the real state s_t. Our method should therefor be able to predict x_{t+1} while only having access to x_t as well as the applied control u_t.

We define our overall problem into two sub-problems.

1. Learning a mapping from the high-dimensional images x_t into a low-dimensional latent space z_t.
2. Learning the transitional dynamics of the real environment in the latent space.

We will start by discussing the sub-problems individually before discussing challenges with combining the two methods.

4.2 Learning a mapping from a high-dimensional image x_t to latent space z_t

To learn a mapping m such that $z_t = m(x_t)$, we chose to implement a variational autoencoder (VAE). Our choice was made based on the fact that VAEs are proven to be highly effective, and especially useful for generative modeling. The VAE achieves the property of continuous latent spaces by having the encoder predict values for the mean μ and variance σ of a Gaussian distribution. The individual latent variables are then sampled from the distribution. Due to sampling, the encoding can vary, even though the mean and variance stay the same.

4.2.1 VAE implementation

The implementation of the VAE was relatively straightforward. We adopted a similar network design for our encoder and decoder as the one used in the E2C method, explained in section 5.2.3. We experimented with the number of layers as well as the layer sizes, but ultimately our experiments did not result in any improvements in terms of reconstruction or computational complexity.

4.3 Learning the transitional dynamics of the real environment in the latent space

Learning the transitional dynamics of the real environment in the latent space involves learning a function t such that $z_{t+1} = t(z_t, u_t)$. To learn this function we decided to use an einsum network. We motivate our choice of using an einsum network based on the following characteristics.

1. Using an einsum network to predict z_{t+1}, given z_t and u_t, eliminates the need to constrain the transition to be linear (as in the E2C method). The added flexibility might however lead to a more complex optimization problem and potentially worse reconstruction as a result.

2. The einsum network, by design, has a general way of expressing the parametric distributions as Gaussians, same as with the latent space in the VAE.

3. The einsum network learns the probability distribution, $p(z_t, u_t, z_{t+1})$, and can therefor compute any subset of these variables, allowing for more control over the latent space predictions.

4.3.1 Einsum network implementation

Our implementation of the einsum network was based on a code provided by Robert Peharz et al. [14]. The main challenges involved choosing a suitable structure for the einsum network, adjusting hyper parameters as well as getting the einsum network to work with categorical actions.

Choosing the structure of the einsum network

Choosing a suitable structure for the einsum network did proof to be particularly difficult. On one hand, the "optimal" structure of an einsum network does vary a lot depending on the problem. On the other hand, we were unable to find any literature where an einsum network was applied to a similar problem. Therefor, we had to spend a lot of time testing the effect of changing the structure of our network as well as adjusting the hyper parameters. Additionally, the structure was also highly dependant on the VAE, which we will discuss in a different section.

Einsum network with categorical actions

The current implementation of einsum networks does not include learning the probability distribution $p(z_t, u_t, z_{t+1})$ where the action, u_t, is a categorical value. To solve this problem, we define the different actions as classes, which essentially creates a single model with an early split on u.

Training the einsum network

For the einsum network to learn the probability distribution $p(z_t, u_t, z_{t+1})$ the values of z_t and z_{t+1} are passed through the einsum networks. The einsum network outputs a vector of size $n_z \times n_u$, where n_z is the number of latent variables and n_u the number of categorical actions, containing the log-likelihood densities computed by the einsum network. For the einsum network to learn the probability distribution, $p(z_t, u_t, z_{t+1})$, the output of the einsum is multiplied by a one-hot encoding of the categorical action. This leads to a vector of size n_z, containing the log-likelihood densities for the action u_t. Finally, the gradients of the einsum network are computed using the log-likelihood densities for u_t and the EM algorithm used to update the parameters of the network.

Predicting using the einsum network

When making predictions of the latent space embedding of the next state, z_{t+1}, using the einsum network, the latent space embedding of the current state z_t is used as evidence to compute $p(z_{t+1}|z_t, u_t)$ using MPE.

4.4 Combining the VAE and Einsum Network

We have now defined the individual components of our overall model so the next step is to define an overall architecture. Figure 4.1 shows the architecture of our overall model but the design is largely based on the architecture of the E2C.

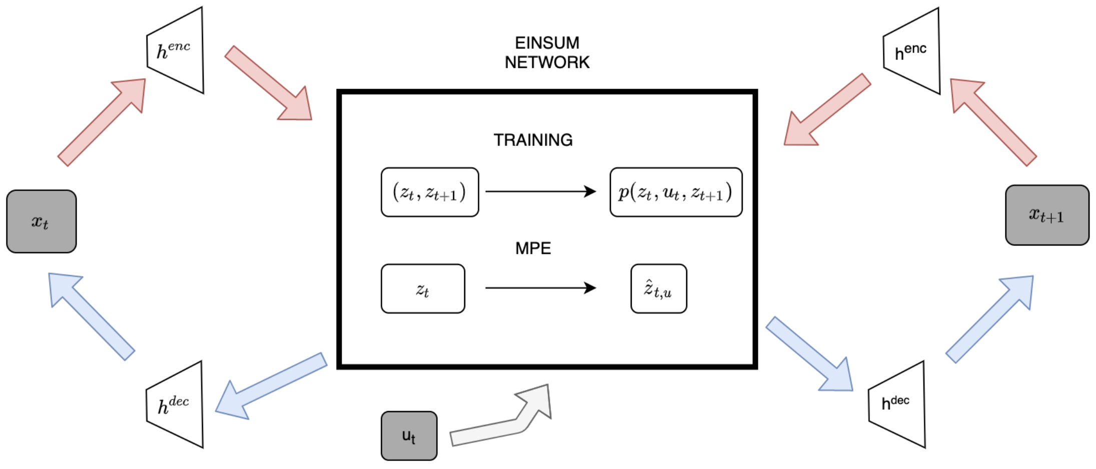


Figure 4.1: VAE-Einsum architecture

The grey boxes represent the observed image states, x_t and x_{t+1}, observed from the real environment, along with the observed action u_t applied at the time step t. The image states are passed through the encoder to get the latent space embeddings z_t and z_{t+1}. The latent spaces are then passed through the einsum network as well as the decoder to compute the loss function and update the parameters of the model.

A crucial part of getting the VAE and einsum network to work together was to enforce an agreement between the latent space embeddings coming from the encoder and the predicted latent space coming from the einsum. In the following sections we will discuss our method of choosing the optimal structure of our einsum network as well as our approach of enforcing an agreement between the two networks in the latent space.

4.4.1 Choosing the structure of the einsum network

As mentioned in section 4.3.1, due to a lack of existing research in applying einsum networks on a similar problem, our approach of choosing the optimal structure of our einsum network involved a lot of trial-and-error. The following structural parameters had to be chosen.

- The graph structure.
- The depth of the tree.
- The number of repetitions.
- The vector length for the sums and leaves.

As a starting point, we selected a simplified version of the structure presented by Robert Pehars et al. [14] for generative image modeling by choosing a random binary tree graph structure with *depth* $= 2$, number of repetitions $r = 5$ and the vector length for the sums and leaves as $K = 5$.

To optimize the structural parameters of our network we analysed the reconstruction error between the MPE prediction of $\hat{z}_{t+1}$, from the einsum network and z_{t+1}, given by the encoder by computing the mean-squared-error between the two. We then chose the combination of structural parameters that resulted in the lowest reconstruction error.

The same method was applied for the hyper parameters but the following hyper parameters had to be chosen.

- Update frequency of the EM algorithm.
- Step-size of the EM algorithm.
- The maximum and minimum values of the projection of Gaussian variances after each EM update.

To prevent overfitting of our einsum network, we had to chose a relatively small step size for our EM algorithm as well as a large update frequency. The results of our parameter optimization are available in appendix B. We were able to achieve the best results by choosing an EM update frequency of 10 and a EM step size of 0.001. On a larger dataset we did however encounter instabilities in our training process, where we observed the log likelihood rapidly increasing, so in our experiments we chose an EM update frequency of 50. This slowed down the training process of the einsum network but also prevented the model from being unstable. For the interval of the Gaussian variances we also noted that the predictions of $\hat{z}_{t+1}$ did improve as the interval was set to a lower number, but at the same time if the variances were set too low, the training process also became unstable.

4.4.2 Enforcing an agreement in the latent space

In the E2C method, an agreement between the latent space predictions, $\hat{z}_{t+1}$, coming from the transition system and the encoding of the next state, z_{t+1}, is enforced by the contradiction term proposed in equation 3.9. Since the E2C uses linear transformations, this KL term can be computed analytically, but the same does not apply for the einsum network.

In an attempt to solve this problem, we experimented with two different implementations of our VAE-Einsum. We refer to these methods as *VAE-Einsum* and *VAE-Einsum-Decoupled.*

VAE-Einsum-Decoupled

For our implementation of the VAE-Einsum-Decoupled, our hypothesis is, that the VAE and einsum network can be trained separately, and that the agreement between the latent spaces is enforced by having the einsum learn $p(z_t, u_t, z_{t+1})$, where z_t and z_{t+1} are both coming from the decoder. Additionally, the agreement is enforced by passing the MPE reconstruction $\hat{z}_{t+1}$ to the decoder during training.

The VAE and einsum networks are trained as two separate networks, meaning that for each forward pass through the model as a whole, there are two backward passes, used to compute the gradients, one for the VAE and one for the einsum networks. After z_t and z_{t+1} are past through the einsum network, the gradients are computed directly, before using MPE to predict $\hat{z}_{t+1}$. The gradients for the VAE are however computed after each forward pass through the model as a whole.

VAE-Einsum

For our implementation of the VAE-Einsum, our hypothesis is, that the VAE and einsum network have to be trained as a whole and therefor enforcing the einsum network to act as a regulizer for the VAE. To achieve this, our approach requires us to modify the regular ELBO loss in the following way

$$\log p(x, u, x') \geq E_{z(.|x), z'(.|x')} \log \left[\frac{p(x|z)p(x'|z')p(z, u, z')}{q(z|x)q(z'|x')} \right] \tag{4.2}$$

Using this modification of the ELBO loss, we are able to eliminate the term $p(z)$ from the regular ELBO loss and instead the einsum network acts as a regulizer by computing $p(z_t, u_t, z_{t+1})$.

Therefor the terms $p(x|z)$ and $p(x'|z')$ are given by our decoder, $q(z|x)$ and $q(z'|x')$ are given by the encoder and $p(z, u, z')$ by the einsum network.

The EM optimizer as well as the optimizer used for the VAE therefor minimize the negative value of our loss function. Since the EM optimizer is only implemented to maximize the log-likelihood, our work-around approach was to multiply the gradients of the einsum network by -1 before each EM update.

Chapter 5

Experimental Setup

5.1 Environments

The environments used for the experiments were acquired from the OpenAI Gym toolkit for reinforcement learning research [2].

5.1.1 CartPole-v0

The cartpole environment includes a pole that is attached by an un-actuated joint to a cart which moves along a frictionless track. The control problem involves deciding at each time step weather a force should be applied on the left or right side of the cart. The objective is to prevent the pole from falling below 15 degrees from vertical or moving more than 2.4 units from the center. The original state available to the agent at each timestep includes four value; the position of the cart, velocity of the cart, angle of the pole and the rotation rate of the pole.

At each timestep the agent has a choice between two discrete actions *left* and *right*. A reward of +1 is provided for each timestep that the pole remains upright and the environment is considered solved when an agent is able to reach a reward of 200 for 5 consecutive episodes [2].

5.1.2 LunarLander-v2

The LunarLander environment includes a lander spacecraft where the objective is to move the lander from the top of the screen to a landing pad. To control the lander the agent has a set of four actions: do nothing, fire left orientation engine, fire main engine and fire right orientation engine.

Each episode finishes if the lander crashes or comes to rest receiving an aditional -100 to 100 points. Each leg ground contact is +10 points. Firing the main engine is -0.3 points each frame. The environment is considered solved if the agent receives 200 points [2].

5.1.3 Acrobot-v1

The Acrobot environment includes two joints and two links, where the joint between the two links is actuated. At the beginning of each episode both joints are hanging downwards and the goal is the swing the lower end up to a given height [2]. The agent has a set of three actions; apply torque to the left, apply torque to the right or do nothing.

5.1.4 Preprocessing

For all the environments the state of the environment at each timestep is a rendered image of the environment provided by the toolkit. The original size of the images was 400×600 for all of the environments. To limit computational complexity the images were pre-processed by gray scaling them as well as limiting their size to 80×80 pixels.

The training set used for all the models contains tuples (s_t, u_t, s_{t+1}) where s_t is an image at timestep t, u_t was the action taken at that timestep and s_{t+1} is the resulting state after performing the action [2]. The overall number of tuples collected for each environment was 30000 where 24000 tuples were used for our training set and 6000 for our test set.

5.2 E2C

5.2.1 E2C architecture

As explained by Manuel Watter et al. [21] the E2C method was implemented based on the following architecture shown in figure 5.1.

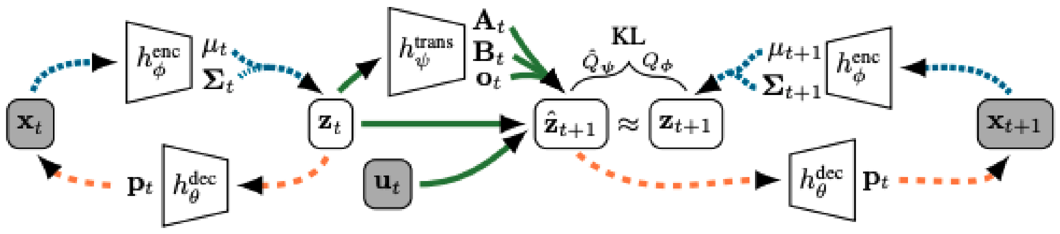


Figure 5.1: E2C architecture, provided by Manuel Watter et al. [21]

The grey boxes in figure 5.1 are observed state and action tuples (x_t, u_t, x_{t+1}) collected from the environment where x_t represents the current state, u_t the action taken from the current state and x_{t+1} the state observed after taking the action.

Furthermore, h_θ^{enc} corresponds to the encoder, h_θ^{dec} the decoder and h_ψ^{trans} the transition system. The current observed state x_t is passed through the encoder which outputs the mean μ_t and variance $\sum_t$. Using the reparameterization trick the latent space embedding z_t is constructed from these two variables. The latent space z_t is then passed back through the decoder to reconstruct x_t as well as through the transition h_ψ^{trans} to produce the estimation of the latent space $\hat{z}_{t+1}$ of the next state x_{t+1}. The latent space estimation $\hat{z}_{t+1}$ is then passed through the decoder to reconstruct the next state x_{t+1}. Finally, the real observed next state x_{t+1} is passed through the encoder to get z_{t+1}.

As discussed in section 3.1 the transition predicts linearization matrices $A_t \in R^{n_z \times n_z}$, $B_t \in R^{n_z \times n_u}$ and offset $o_t \in R^{n_z}$. The transformation matrices and offset where parameterized using equation 3.8. To reduce the parameters being estimated from n_z^2 to $2n_z$, A_t of size $n_z \times n_z$ was chosen to be a perturabation of the identity matrix $A_t = (I + v_t r_t^T)$.

5.2.2 E2C loss function

The loss function was implemented following equation 3.9 where the terms $\log P_\theta(x_t|z_t)$ and $\log P_\theta(x_{t+1}|\hat{z}_{t+1})$ represent the reconstruction losses from z_t to x_t and $\hat{z}_{t+1}$ to x_{t+1}, $KL(Q_\phi||P(Z))$ represents the KL-divergence between the parametric inference distribution from the encoder h_θ^{enc} and the prior $P(Z_t)$ on the approximate posterior Q_ϕ. The prior is always chosen to be an isotropic Gaussian distribution with mean zero and unit variance.

The final term in equation 3.9, $KL(\hat{Q}_\psi(\hat{Z}|\mu_t, u_t)||Q_\phi(Z|x_{t+1}))$ represents the KL-divergence between the encoding of the next state from the encoder and the estimation of the encoding of the next state from the transition. This term forces an agreement between the transition and inference model.

5.2.3 Network architecture

Encoder

The encoder consists of three convolutional layers, each followed by a 2×2 max-pooling layer, batch normalization and a $ReLU$ activation function. The input to the encoder is an image of size $80 \times 80 \times 1$ and the network outputs flat features of size 512. The flat features are passed through a linear layer with output of size 8 and a $tanh$ activation function, to estimate the log of the variance, and a linear layer with output of size 8 and $ReLU$ activation function to estimate the mean value. Figure 5.2 shows the architecture of the encoder.

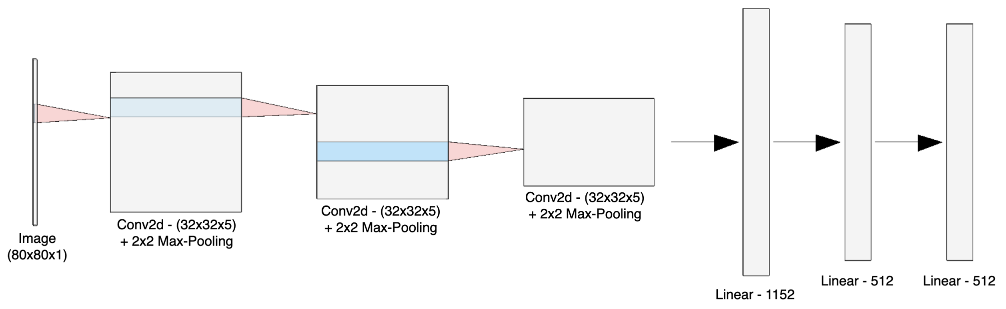


Image (80x80x1)
Conv2d - (32x32x5) + 2x2 Max-Pooling
Conv2d - (32x32x5) + 2x2 Max-Pooling
Conv2d - (32x32x5) + 2x2 Max-Pooling
Linear - 1152
Linear - 512
Linear - 512

Figure 5.2: Encoder architecture

Decoder

The decoder is the reverse of the encoder reconstructing a full image from the latent space. As with the encoder, the decoder has three convolutional layers, each followed by a 2×2 up-sampling layer and a $ReLU$ activation function. Figure 5.3 shows the architecture of the decoder.

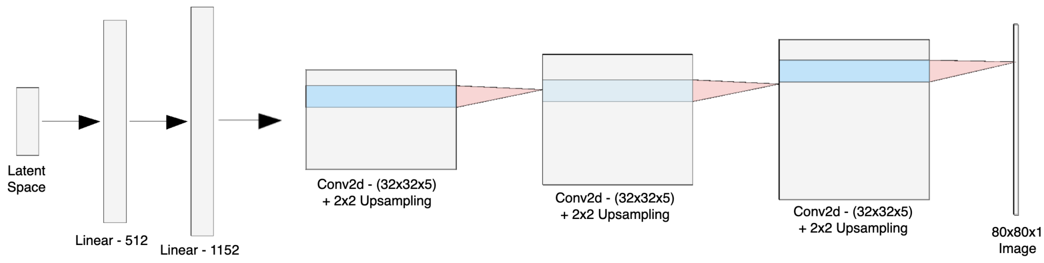

Latent Space
Linear - 512
Linear - 1152
Conv2d - (32x32x5) + 2x2 Upsampling
Conv2d - (32x32x5) + 2x2 Upsampling
Conv2d - (32x32x5) + 2x2 Upsampling
80x80x1 Image

Figure 5.3: Decoder architecture

Transition

The neural network implemented for the transition system had three hidden layers of size 200. The outputs of the neural network were parameterized following equation 3.8 where transition matrices A_t, B_t and offset o_t where estimated. To avoid estimating the full matrix A_t of size $n_z \times n_z$, we choose it to be a perturbation of the identity matrix $A_t = (I + v_t r_{t^T})$. Therefor we only have to estimate $2n_z$ parameters for A_t.

5.2.4 E2C hyperparameters

We chose the Adam optimizer with a learning rate $lr = 0.0001$, weight-decay of $\epsilon = 10^{-8}$ and our β values were chosen as $\beta_1 = 0.9$ and $\beta_2 = 0.8$. The discount factor λ in front of the contradiction term in 3.9 was chosen as $\lambda = 0.9$.

5.3 VAE-Einsum and VAE-Einsum-Decoupled

5.3.1 VAE design

For the VAE, the same network design was used for the encoder and decoder as for the E2C (explained in section 5.2.3). Additionally, we used the Adam optimizer with a learning rate $lr = 10^{-5}$, weight decay of $\epsilon = 10^{-8}$ and our β values were chosen as $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

5.3.2 Einsum network design

The same network design was used for both implementations of the VAE-Einsum. The structure we chose for our einsum network was a random binary tree graph with depth $d = 3$ and number of repititions $r = 20$. The vector length for sums and leaves was set to $K = 10$ and after each EM update the Gaussian variances were projected to the interval $[10^{-6}, 0.5]$. The EM stepsize was chosen to be $em_{step} = 0.01$ and the EM frequency was set to $em_{freq} = 50$.

5.3.3 Balancing the VAE-Einsum during training

To balance the learning process of the VAE and the einsum network in the VAE-Einsum implementation, we also introduce a variable β in front of the terms $p(z, u, z')$, $q(z|x)$ and $q(z'|x')$ that increases every epoch, starting at $\beta = 0.1$ and ending in $\beta = 1.0$, with a step size of $\Delta\beta = 0.1$.

5.4 Reinforcement Learning

To compare the E2C and VAE-Einsum in terms of control in the latent space both methods were tested within a reinforcement learning framework utilizing imagined trajectory planning. In this chapter we will refer to the E2C and VAE-Einsum models as *world models*.

5.4.1 Dyna-DQN

The reinforcement learning framework used to compare the methods was Dyna-DQN inspired by G. Zacharias Holland et al. [6]. Since our focus is on control in latent space the Dyna-DQN will only be applied on latent space embeddings of real states observed from the environment as well as imagined latent space trajectories predicted by our world model.

After each step in the real environment the Dyna-DQN stores transitions (z_t, u_t, r_t, z_t) inside an experience replay buffer R_{real} where z_t is the latent space embedding of the previous state, u_t is the action taken from z_t, r_t is the reward received after taking the action and z_{t+1} is the latent space embedding of the next state. Each action was chosen based on an ϵ-greedy policy with $\epsilon = 0.1$.

Additionally, after each step in the real environment, the Dyna-DQN samples a starting state from the experience replay buffer and uses the world model to predict a trajectory of length K using random actions taken from the sampled starting state. The predicted transitions are then added to another experience replay buffer R_{total} that contains a mixture of real observed transitions, also in R_{real}, along with predicted transitions from the world models.

To estimate the reward received for each step in the imagined trajectories, the world models use a linear neural network with a single hidden layer of size 64 that learns from samples in R_{real}.

The following pseudo code shows how the Dyna-DQN was implemented.

Algorithm 3: Dyna-DQN

```
Initialize experience replay buffer $R_{real}$
Initialize experience replay buffer $R_{total}$
Initialize previously trained world model $ENV$
Initialize $DQN$
while Training do
    Restart episode
    while Episode not finished do
        Get current state from environment $S_t$
        Embed current state to latent space $z_t$ using $ENV$
        Choose an action $a_t$ using $DQN$ ($\epsilon$-greedy)
        Take step $u_k$ in real environment and observe $s_{t+1}$
        Embed $s_{t+1}$ into latent space $z_{t+1}$ using world model
        Store transition $(z_t, u_t, r_t, z_{t+1})$ in $R_{real}$ Store transition $(z_t, u_t, r_t, z_{t+1})$ in $R_{total}$
        Sample a starting state $s_0$ from $R_{real}$
        Predict a trajectory $((z_0, u_0, r_0, z_{0+1})...(z_k, u_k, r_k, z_{k+1}))$ using world model
        Store all trajectory transitions in $R_{total}$
    end
    Update DQN using random transitions from $R_{total}$
end
```

With $K = 0$, where no trajectory predictions are made in latent space, the implementation of Dyna-DQN becomes regular DQN.

5.5 Comparing E2C and VAE-Einsum

The three previously mentioned models, E2C, VAE-Einsum and VAE-Einsum-Decoupled, were analysed based on the following criteria.

- Performance on the test set based on the reconstruction of the next state s_{t+1} given the current state s_t and action u_t.
- Their ability to generate image trajectories, resembling the real environment.
- The distribution of the predicted latent-spaces using t-SNE.
- Performance when applied within a reinforcement learning framework for control in latent space.

All of the models were trained on three different environments, using a training set of 24000 observations and a test set of 8000 observations. We chose a batch size of 10 for all of our models.

For our reinforcement learning experiments, the methods were trained for a total of 2000 episodes. We compare the average score of all our methods using Dyna-DQN with $K = 0$ (DQN) with an addition to VAE-Einsum and VAE-Einsum-Decoupled using Dyna-DQN with $K = 10$.

Chapter 6

Results

6.1 Training performance

The models were trained for a total of 60 epochs and after each epoch, the models were compared in terms of the reconstruction error on the test set. The following figure shows the performance of each model on the test set, after each epoch in the CartPole environment.

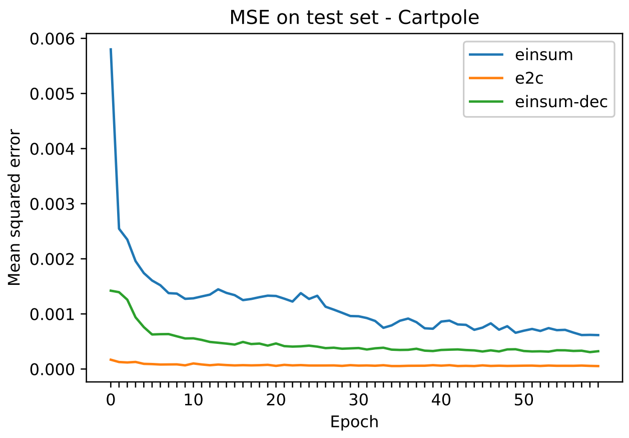


Figure 6.1: Average reconstruction error (MSE) of the next state in the CartPole environment

The E2C outperforms the other methods in terms of reconstruction error but the other models are still performing well and after training for 60 epochs the models are achieving the following average reconstruction error on the test set.

CartPole	E2C	VAE-Einsum	VAE-Einsum-Decoupled
Avg. MSE	5.34×10^{-5}	6.16×10^{-4}	3.23×10^{-4}

Table 6.1: Average MSE for the CartPole environment

Additionally, the reconstruction of the latent spaces was analysed for the einsum networks where the real state was assumed to be the encoding of the next state. The following figure shows the average mean-square-error of the einsum networks in the CartPole environment.

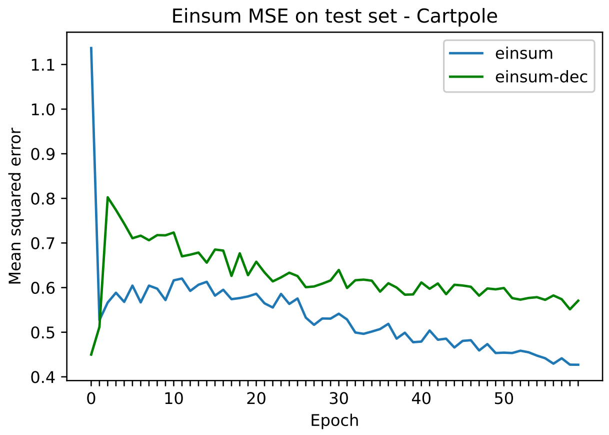


Figure 6.2: Average reconstruction error (MSE) of the latent space in the CartPole environment

The figure shows the einsum networks gradually improving in terms of reconstructing the latent space embedding of the next state. During the first few iterations, the encoder is still improving and therefor the predictions from the einsum network are unstable. The following figures show the same results obtained from the LunarLander and Acrobot environments.

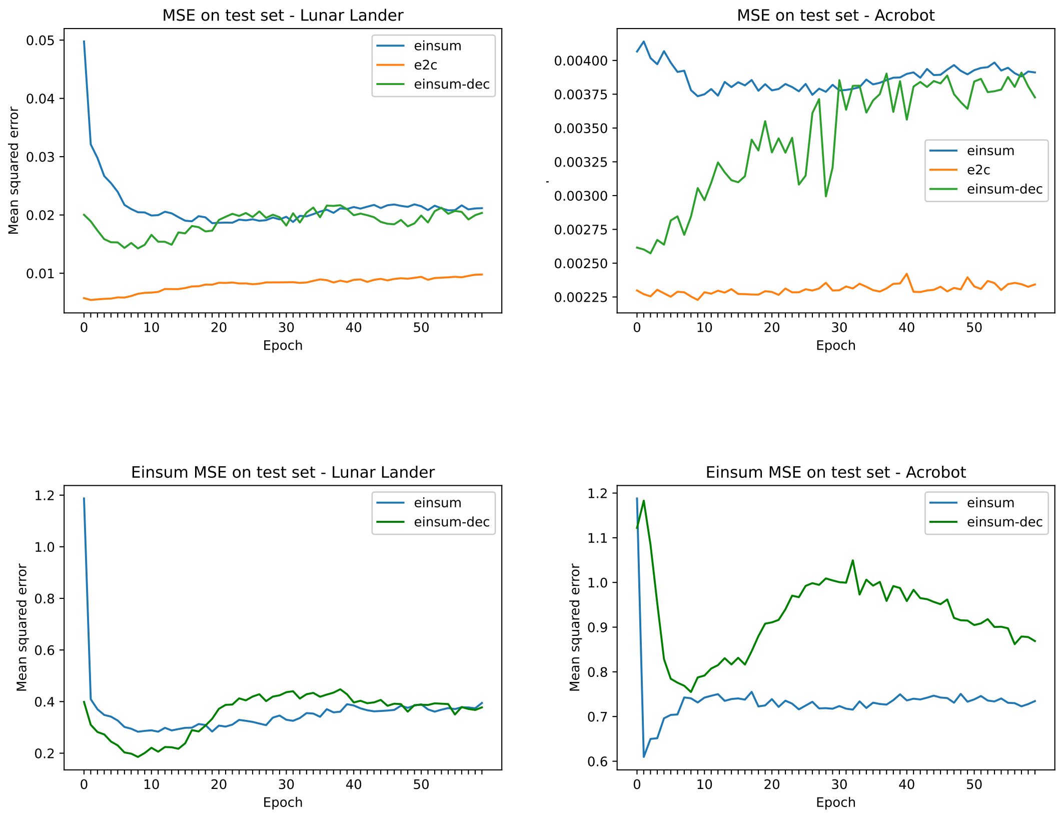


Figure 6.3: Average reconstruction error (MSE) of the next state (top) and average reconstruction error in the latent space (bottom)

The reconstruction loss is considerably higher in the LunarLander environment compared to the other two. However, the reconstruction of the latent spaces is similar to the results in the CartPole environment. For the Acrobot environment the reconstructions are better then in the LunarLander environment but still not as good as in the CartPole environment.

The following table shows the overall results the three methods on all of the environments.

E2C	**VAE-Einsum**	**VAE-Einsum-Decoupled**	**Environment**
5.34×10^{-5}	6.16×10^{-4}	3.23×10^{-4}	Cartpole
9.78×10^{-3}	2.12×10^{-2}	2.04×10^{-2}	LunarLander
2.34×10^{-3}	3.91×10^{-3}	3.72×10^{-3}	Acrobot

Table 6.2: Average reconstruction error (MSE) for each method in each environment

To visualize the performance of each model in terms of reconstruction the following samples were obtained. The samples were obtained by sampling a random data point from our dataset and performing a one step prediction with each model.

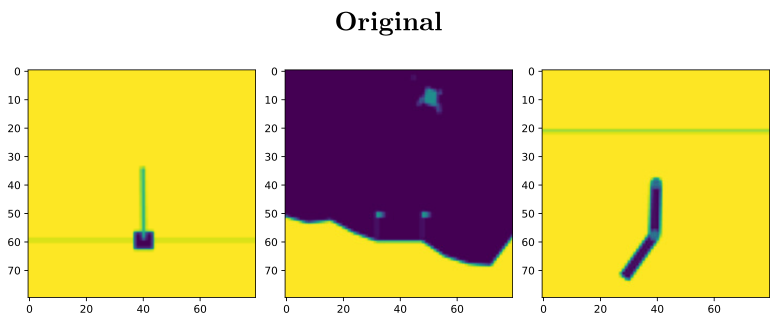


Figure 6.4: Real observed next state of our sample

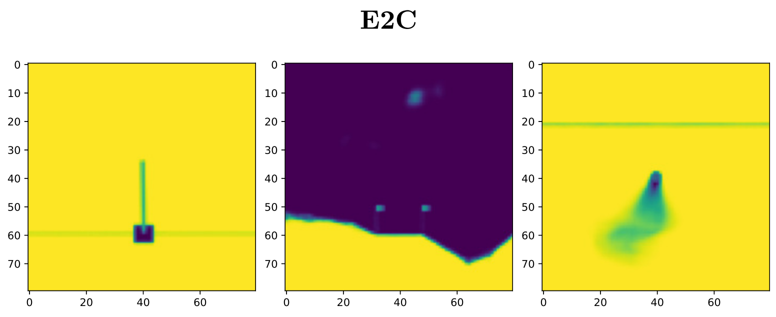


Figure 6.5: Predicted next state using E2C

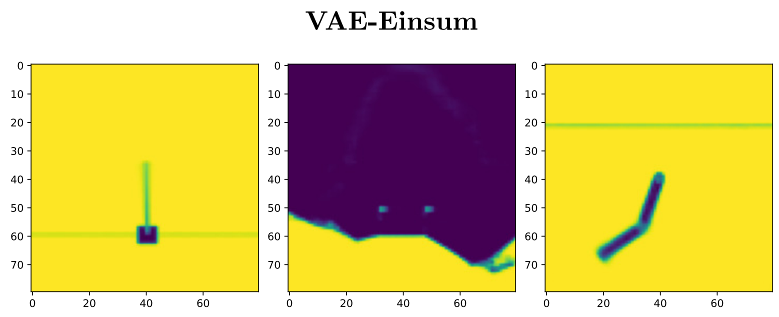


Figure 6.6: Predicted next state using VAE-Einsum

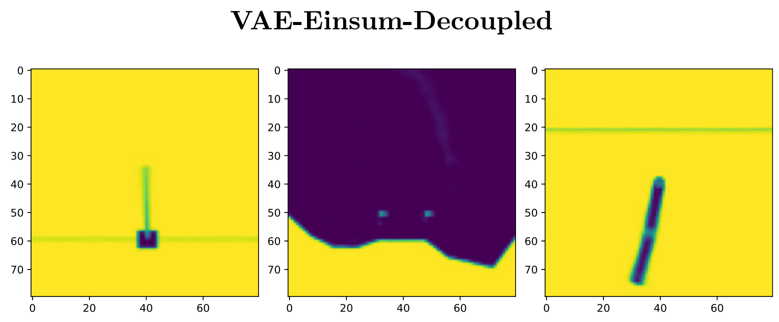


Figure 6.7: Predicted next state using VAE-Einsum-Decoupled

The VAE-Einsum and VAE-Einsum-Decoupled have no problem reconstructing the states in the Acrobot and CartPole environments but they struggle with the LunarLander environment, especially in terms of reconstructing the spaceship itself.

6.2 Trajectory predictions

The following figures show imagined trajectories, predicted by each method, in the CartPole environment where the starting state was randomly sampled from the environment along with a fixed action moving the cart to the right.

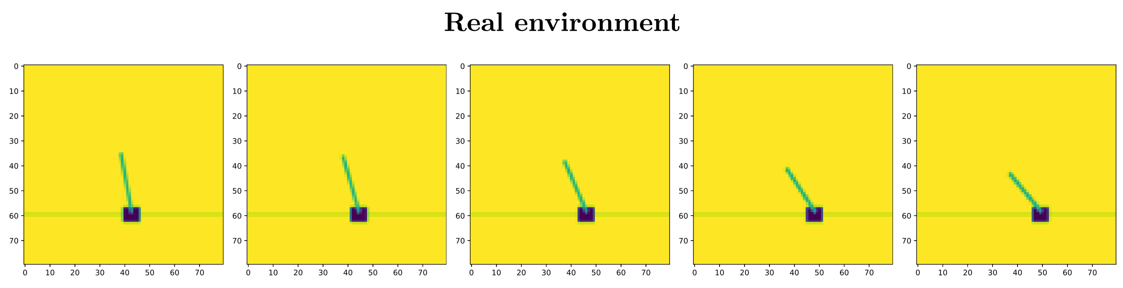


Figure 6.8: Trajectory observed from the real environment

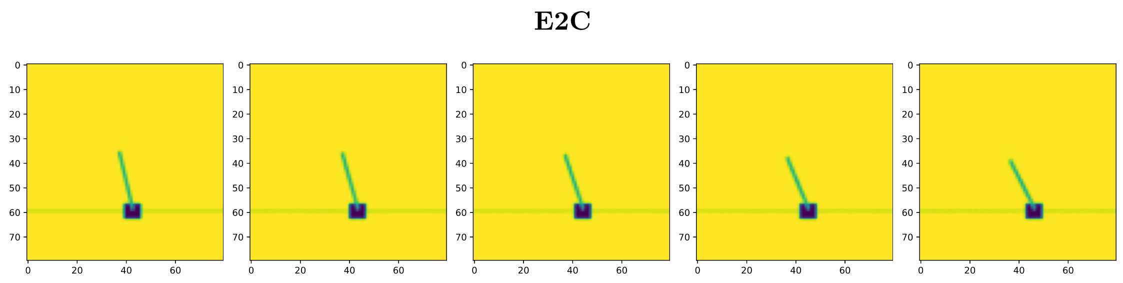


Figure 6.9: Trajectory prediction obtained from E2C

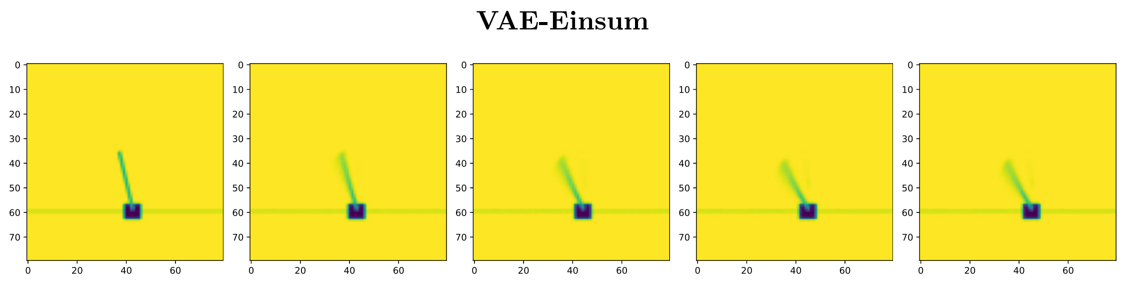


Figure 6.10: Trajectory prediction obtained from VAE-Einsum

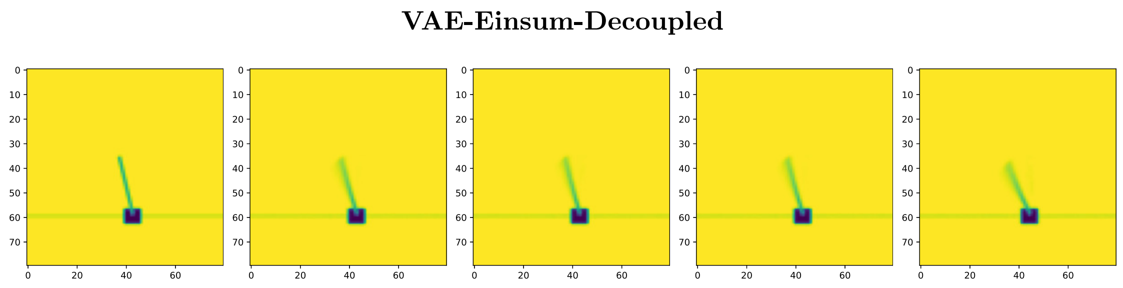


Figure 6.11: Trajectory prediction obtained from E2C

All of the methods succeed in capturing the dynamics of the environment by generating an imagined trajectory representing the real dynamics of the environment. For the VAE-Einsum and VAE-Einsum-Decoupled the predictions tend to be a little bit blurrier compared to the E2C. We also noted that even though both the VAE-Einsum and VAE-Einsum-Decoupled are generally able to predict good trajectories, resembling the real environment, our models tend to less reliable than E2C in their predictions of the next state, mainly in two ways. Firstly, our models occasionally get stuck, predicting the same next state over and over. Secondly, our models occasionally go "off-track", meaning that the predictions are completely different from the real environment, for example predicting the pole in a horizontal position if the previous state was the pole in an upright position.

For the LunarLander environment the sample reconstructions shown in section 6.1 indicate that the models are not doing very well in terms of the reconstruction of the next state therefor generating imagined trajectories was unsuccesfull. The following figures show some sample trajectories obtained for the Acrobot environment.

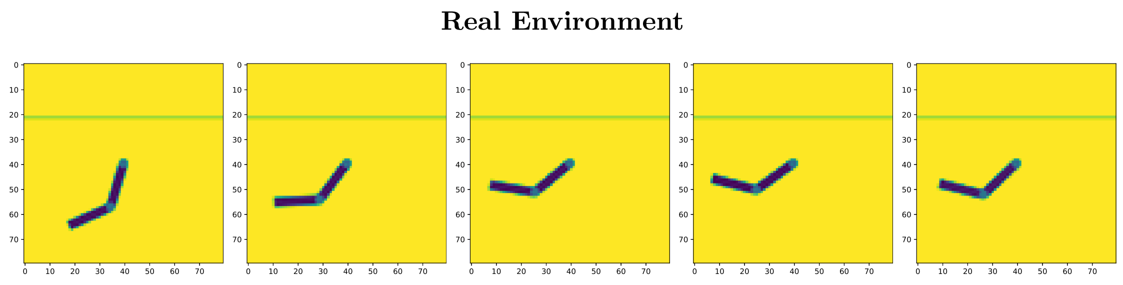


Figure 6.12: Trajectory observed from the real environment

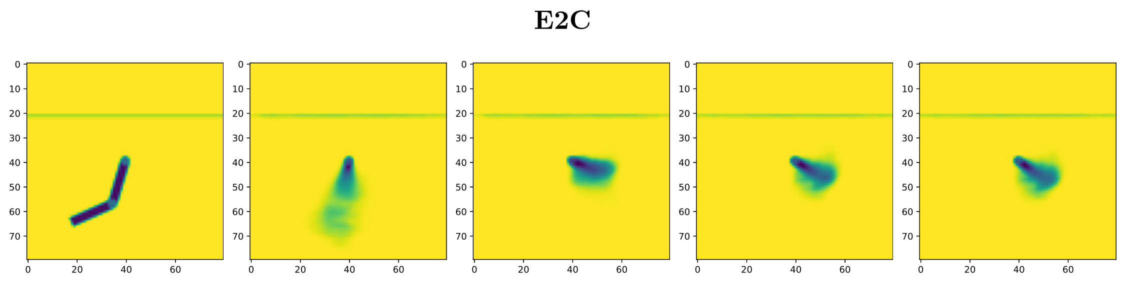


Figure 6.13: Trajectory prediction obtained from E2C

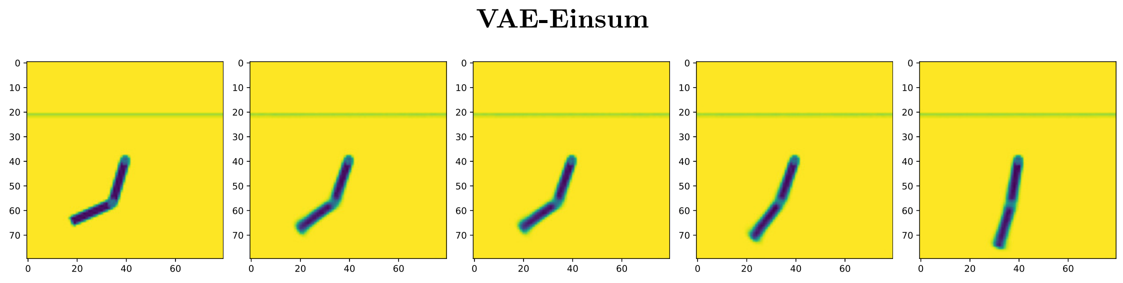


Figure 6.14: Trajectory prediction obtained from VAE-Einsum

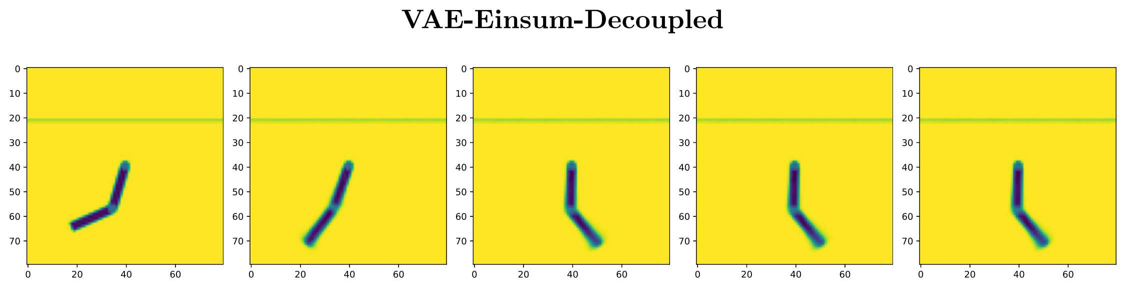


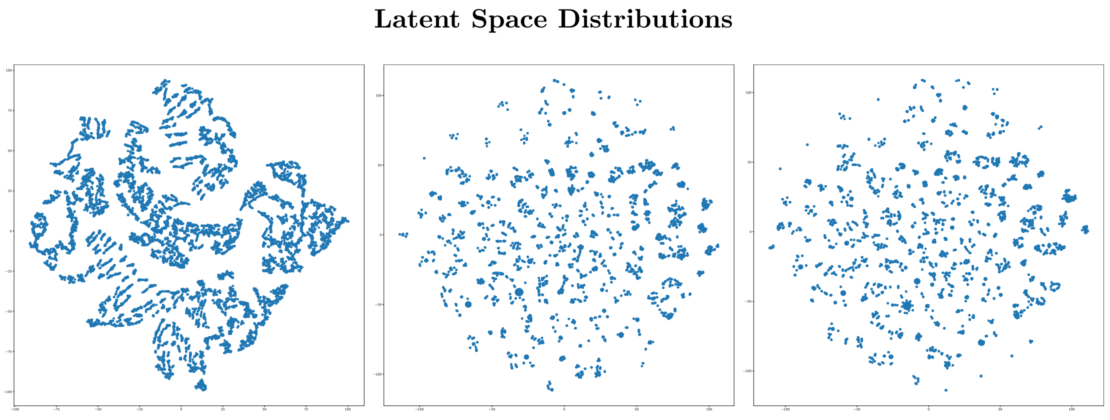
Figure 6.15: Trajectory prediction obtained from VAE-Einsum-Decoupled

All of the models struggle to accurately predict the real trajectory obtained from the environment. The E2C method seems to show signs of uncertainty in the predictions judging from the blurriness of the reconstructions. Both VAE-Einsum methods seem to be more deterministic and the trajectories obtained do more closely resemble trajectories observed from the real environment.

We observe both VAE-Einsum networks capturing the dynamics in reverse compared to the real environment. This is somewhat expected behaviour. Predicting the next step of the environment from our starting position is highly dependant on the magnitude and direction of the velocity of the pendulum before reaching that state. If the pendulum was already moving downwards at our starting position, the force applied would merely slow down the movement of our pendulum to the right side. In the real environment the pendulum was in fact moving upward, therefor the real dynamics involved the pendulum moving further to the left. Without any additional information, it is therefore impossible for our methods to determine in which direction the pendulum was moving before the original state.

6.3 Latent space analysis

The latent space predictions obtained from each method were analysed by embedding them into a two-dimensional space using t-SNE. The following figure shows the distribution of the latent spaces using a set of 18000 observations from our test set in the CartPole environment. The latent space was obtained from the transition system so each point in figure 6.16 represents a next state latent space predicted by the transition system.



(a) E2C (b) VAE-Einsum (c) VAE-Einsum-Decoupled

Figure 6.16: Latent space distribution for all of the models

Figure 6.16 shows the difference between the distribution of the latent spaces for each method. To gain a better understanding for what each point in our plot means, we mapped the resulting reconstruction of the full images onto our distribution. The visualization including the reconstructed images is available in appendix A.

The distribution of the latent spaces are very similar for VAE-Einsum and VAE-Einsum-Decoupled but quite different from E2C. For E2C we can clearly visualize different larger clusters, where each cluster results in similar reconstructed images. Furthermore, within each cluster we notice a linear pattern between each point. For VAE-Einsum and VAE-Einsum-Decoupled, we do not observe the same formation of larger clusters. Instead, the latent spaces are more evenly spread out with the formation of small circular clusters. Judging from our visualization of the reconstructed images, we do observe similar images being close to each other however the boundaries are not as clear as in E2C.

We continue our analysis by analysing the effect of individual latent variables on the overall distribution. For our analysis we use our VAE-Einsum method and we marginalize individual latent variables when making predictions with our einsum network. When marginalizing a latent variable we are essentially removing it from the evidence used to make predictions in our einsum network. Figure 6.17 shows the overall distribution of the predicted latent spaces for a set of 2000 observations. For each observation we make predictions while singling out one of our 8 latent variables, resulting in a total of 16000 latent spaces.

Latent Space Distributions (combined)

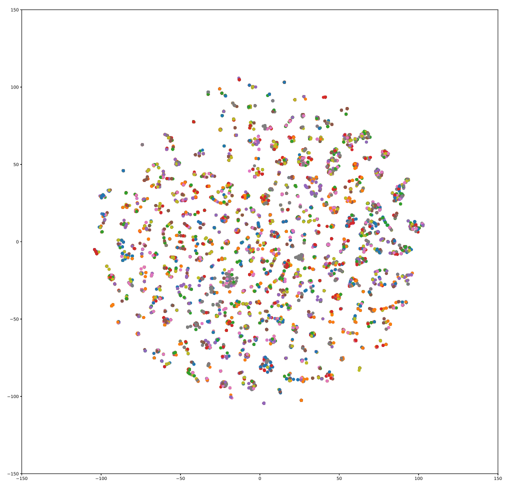
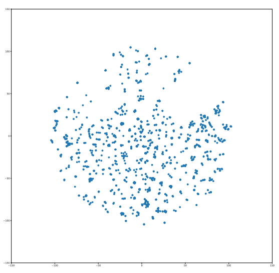


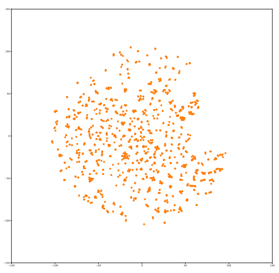
Figure 6.17: t-SNE visualization of the distribution of latent space predictions. Each color represents the predictions after marginalizing an individual latent variable (total of 8 variables)

The overall distribution is very similar to the distribution shown in figure 6.16 and we do not observe a clear distinction between the effect of each latent variable. To continue our analysis we separate our overall distribution into 8 subplots. Each subplot represents the removal of one latent variable. We use n to indicate the index of the latent variable removed.

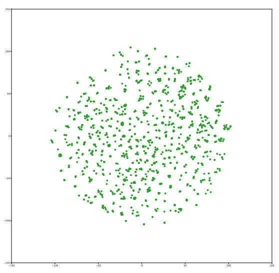
Latent Space Distributions (individual)



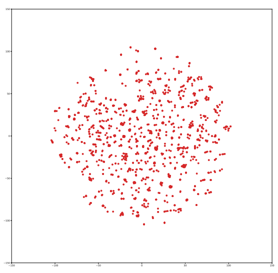
(a) n=0



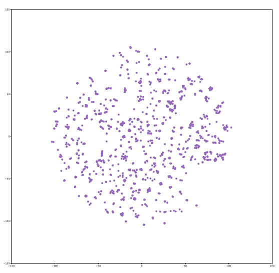
(b) n=1



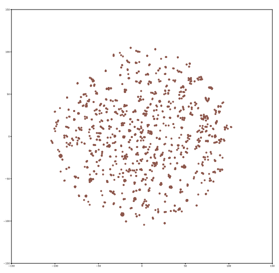
(c) n=2



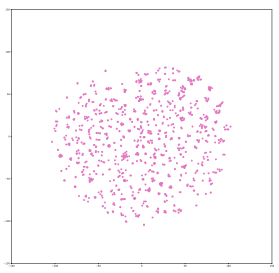
(d) n=3



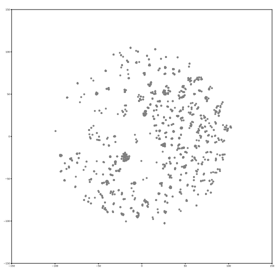
(e) n=4



(f) n=5



(g) n=6



(h) n=7

Figure 6.18: Predicted latent space distributions when removing individual latent variables from the evidence when making predictions. The letter n represents the index of the latent variable removed

From figure 6.18 we observe different gaps appearing in the latent space predictions depending on which latent variable was removed. This analysis might give us an indication into the meaning behind each latent variable. As an example, if we remove the first latent variable $n = 0$ we observe gaps in the upper half of our latent space distribution, which would suggest that latent spaces falling in the upper half of our distribution are highly dependant on the first latent variable. We can compare our results by plotting the reconstructed images on top of the distribution.

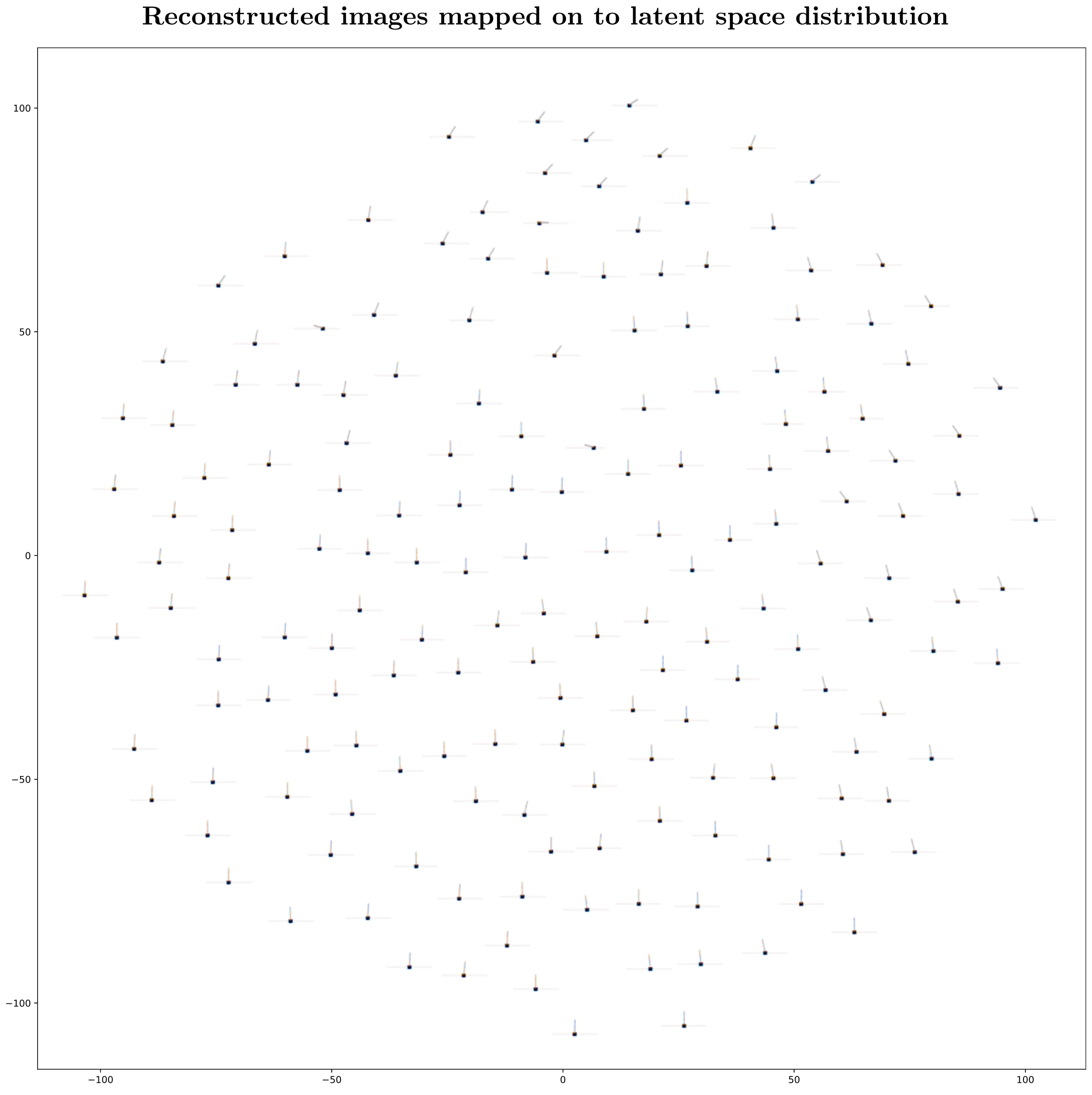


Figure 6.19: Visualization of the reconstructed images on top of the latent space distribution

When comparing figure 6.19 with figure 6.18 we observe that the first two latent variables ($n = 0$ and $n = 1$) seem to be provide our model with evidence for more extreme tilting of the pole to either side while the final latent variable, $n = 7$, seems to be provide our model with evidence for the pole in a central position. For the other latent variables it is more difficult to analyse their effect. There are some gaps in the latent space distribution that suggest that some information was lost when making predictions without those variables, but there are also other factors that are not as easily visible such as the overall information on the reconstruction of the pole, the position of the cart along the line as well as information regarding the background. It is also possible that the interaction between the other latent variables is more complex and therefor harder to distinguish between them by marginalizing them individually.

6.4 Reinforcement learning performance

The following figure shows the results obtained from testing our methods when applied within DQN and Dyna-DQN in the CartPole environment.

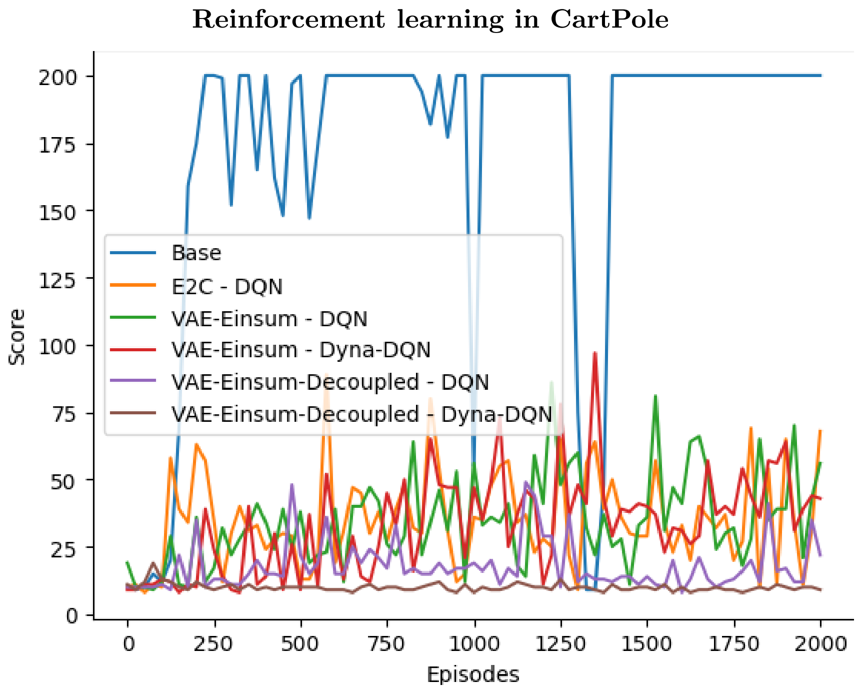


Figure 6.20: Average score received over 2000 episodes

The baseline method was a regular DQN applied on the original state available from the environment. The original state of the environment includes four numbers, explained in 5.1.

The methods were compared based on their performance in the real environment when applying control in latent space. We chose $\epsilon = 0.1$ for the ϵ-greedy policy for all of the methods.

The following table shows the average reward per episode after training the model for 2000 episodes.

Method	Avg. Reward
Baseline	170.03
E2C	34.87
VAE-Einsum	34.59
VAE-Einsum-Dyna	33.17
VAE-Einsum-Decoupled	17.90
VAE-Einsum-Decoupled-Dyna	9.90

Table 6.3: Average score received over 2000 episodes

The CartPole environment is considered solved if a reward of 200 is received for 5 consecutive episodes. E2C and both VAE-Einsum methods were unable to solve the environment and our implementation of Dyna-DQN did not result in any improvements.

For VAE-Einsum and E2C, the results are similar. In Manuel Watters's et al. [21] paper, E2C showed promising results using AICO [18], but no results were presented where E2C was applied within any deep reinforcement learning framework similar to DQN. We expected our methods to perform better when using DQN. One one hand, the bad performance might be due to a poor model of the environment. On the other hand, the bad performance might be due limitations to DQN, such as overestimation of the action values, discussed by Hado van Hasselt et al. [20], or *catastrophic forgetting*, discussed by Melros Roderick et al. [16].

Chapter 7

Conclusions

Combining the einsum network and variational autoencoder proved to be very difficult, mostly due to two problems. Firstly, for our model to be able to generate imagined trajectories resembling the real environment, an agreement between the VAE and einsum network in the latent space was crucial. Our solution to that problem mainly involved adjusting the Gaussian variances of our einsum network. Ultimately our variances had to be chosen low enough such that our einsum network would be able to accurately predict the values of the latent variables, but still high enough such that the einsum network was not imposing a distribution too complicated for the decoder. Secondly, our problem involved stabilizing the training process between the two methods. The difference between the VAE and einsum network in terms of structure and optimization often let to instabilities during the training process. Our solution to stabilizing the training process was to choose a very low step-size for the EM algorithm as well as the frequency of its updates. The EM algorithm, by design, does converge relatively quickly, which ultimately lead to instabilities when combined with the VAE. By lowering the step-size and update frequency we were able to slow down the training process of the einsum network and therefor give the VAE a chance to improve before the einsum network. Slowing down the training process also included adding the β value to our loss function, explained in section 4.4.2.

During training, the VAE had no problems with accurately learning a low-dimensional embedding of the high-dimensional images and overall the reconstructions, in the CartPole and Acrobot environments, were very good. For the LunarLander environment, the VAE found it more difficult to reconstruct the images. The LunarLander environment is a very complicated environment to reconstruct, mostly due to the spaceship itself being a relatively small part of the full image as well as the fact that the background environment changes every episode.

In terms of the trajectory predictions the VAE-Einsum and VAE-Einsum-Decoupled were both able to generate trajectories in the CartPole environment, that closely resembled the real environment. Both methods were able to show results similar to the E2C. For some samples, we however notice the VAE-Einsum and VAE-Einsum-Decoupled going "off track", meaning their next-step prediction were far from the next step observed in the real environment. One possible reason for this behaviour is that, compared to the E2C, our methods do not constrain the relationship between the transitions to be linear and therefor we are "allowing" the predictions of the einsum network to go off track.

In the Acrobot environment, the VAE-Einsum and VAE-Einsum-Decoupled were both able to successfully generate imagined trajectories that resemble the behaviour of the real environment. None of the methods were however able to correctly match the simulated trajectory from the real environment. The reconstruction of the images, coming from the VAE-Einsum and VAE-Einsum-Decoupled, were however notably better then the ones coming from the E2C. The Acrobot environment is however a very dynamical system and outside factors, not represented with the images, such as the velocity of the pole, are crucial to be able to accurately predict the correct trajectories.

By visualizing the distributions of the latent spaces for all of the methods we noted an interesting difference between the E2C and our methods. For the VAE-Einsum and VAE-Einsum-Decoupled the distributions are very similar, but at the same time very different from the distribution observed for the E2C. As discussed in section 6.3, we observed that the distribution of the latent spaces coming from the einsum network were more evenly distributed, there was not a clear formulation of larger clusters and we did not observe a linear relationship between points within a cluster.

This might provide us with an explanation to some of the behaviour we noticed earlier. Without having a clear distinction between larger clusters, representing similar reconstructed images, the next state predictions of our model might be more likely to go "off-track", meaning that our model makes a next state prediction completely different from the previous state.

Furthermore, due to the distribution consisting of multiple smaller clusters it might be difficult for the decoder to recognize the meaning behind different latent spaces within a cluster and therefore the decoder may find it difficult to recognize smaller changes in our reconstruction. This might also provide us with an explanation for why the trajectory predictions coming from VAE-Einsum and VAE-Einsum-Decoupled were generally blurrier than the reconstructions from E2C.

As discussed in section 4.3, one of the motivations for using einsum networks in latent space is more flexibility in terms of what we want to predict. By marginalizing individual latent variables and visualizing the predictions using t-SNE, we were able to analyse the effect of individual variables. Our analysis resulted in an interesting visualization of the effect of different latent variables, which allowed us to make an estimation on what information each latent variable might store.

When applied within a reinforcement learning framework in the CartPole environment, VAE-Einsum and E2C were very similar in terms of the overall score but both methods were however unable to solve the control problem. The ability of VAE-Einsum to generate imagined trajectories in latent space, using Dyna-DQN, did not result in any improvements but the overall score was still only slightly below regular DQN.

We expected a better performance from all of the models using DQN and Dyna-DQN, but weather improving on our results involves improving our models, or improving on the DQN algorithm, requires further analysis.

When comparing VAE-Einsum and VAE-Einsum-Decoupled, we did not observe a big difference in terms of reconstruction or trajectory predictions, but when applied within DQN, the performance of VAE-Einsum was considerably better. We believe that the difference in performance can be explained based on the fact, that in VAE-Einsum, an agreement between the VAE and the einsum network is enforced by having the einsum network act as a regulizer but this agreement between the two, might be crucial when applying control in latent space. Furthermore, by comparing the performance of VAE-Einsum and VAE-Einsum-Decoupled when using Dyna-DQN we notice an even larger difference between the two. In Dyna-DQN, we rely more on the einsum network to make accurate predictions in latent space, compared to DQN. We can therefor assume that the einsum network in VAE-Einsum-Decoupled is performed considerably worse in generating accurate latent space trajectories compared to VAE-Einsum.

Overall, our implementation of combining VAEs and einsum networks did not improve the performance of E2C in terms of generating accurate image trajectories or control in latent space. Our results were however promising and the increased flexibility in latent space did allow us to further analyse the meaning behind individual latent variables as well as enabling us to do planning in latent space using Dyna-DQN.

Bibliography

[1] Mathieu Aubry, Daniel Maturana, Alexei A. Efros, Bryan C. Russell, and Josef Sivic. Seeing 3d chairs: Exemplar part-based 2d-3d alignment using a large dataset of cad models. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3762–3769, 2014. 4

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. 25, 26

[3] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *CoRR*, abs/1606.03657, 2016. 4

[4] I. Higgins, Loïc Matthey, A. Pal, Christopher P. Burgess, Xavier Glorot, M. Botvinick, S. Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2017. 4

[5] Geoffrey E Hinton and Sam Roweis. Stochastic neighbor embedding. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15. MIT Press, 2003. 10, 11

[6] G. Zacharias Holland, Erik Talvitie, and Michael Bowling. The effect of planning shape on dyna-style planning in high-dimensional state spaces. *CoRR*, abs/1806.01825, 2018. 6, 7, 28

[7] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model-based reinforcement learning for atari. *CoRR*, abs/1903.00374, 2019. 5, 6

[8] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *CoRR*, abs/1906.02691, 2019. 3, 4

[9] Tejas D. Kulkarni, Will Whitney, Pushmeet Kohli, and Joshua B. Tenenbaum. Deep convolutional inverse graphics network. *CoRR*, abs/1503.03167, 2015. 4

[10] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015. 4

[11] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011. 9

[12] Iago París, Raquel Sánchez-Cauce, and Francisco Javier Díez. Sum-product networks: A survey. *CoRR*, abs/2004.01167, 2020. ix, 7, 8

[13] Pascal Paysan, Reinhard Knothe, Brian Amberg, Sami Romdhani, and Thomas Vetter. A 3d face model for pose and illumination invariant face recognition. In *2009 Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance*, pages 296–301, 2009. 4

[14] Robert Peharz, Steven Lang, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Guy Van den Broeck, Kristian Kersting, and Zoubin Ghahramani. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. *CoRR*, abs/2004.06231, 2020. ix, ix, 1, 7, 8, 9, 10, 20, 21

[15] Hoifung Poon and Pedro M. Domingos. Sum-product networks: A new deep architecture. *CoRR*, abs/1202.3732, 2012. 9

[16] Melrose Roderick, James MacGlashan, and Stefanie Tellex. Implementing the deep q-network. *CoRR*, abs/1711.07478, 2017. 4, 5, 41

[17] Francisco Rubio, Francisco Valero, and Carlos Llopis-Albert. A review of mobile robots: Concepts, methods, theoretical framework, and applications. *International Journal of Advanced Robotic Systems*, 16(2):1729881419839596, 2019. 1

[18] Marc Toussaint. Robot trajectory optimization using approximate inference. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 1049–1056, New York, NY, USA, 2009. Association for Computing Machinery. 40

[19] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. ix, 10, 11, 12

[20] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), Mar. 2016. 6, 41

[21] Manuel Watter, Jost Tobias Springenberg, Joschka Boedecker, and Martin A. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. *CoRR*, abs/1506.07365, 2015. ix, ix, 1, 13, 14, 15, 16, 17, 19, 26, 40

Appendix A

2-dimensional mapping of latent space with images

The following figures show the distribution of the latent spaces with full images instead of points.

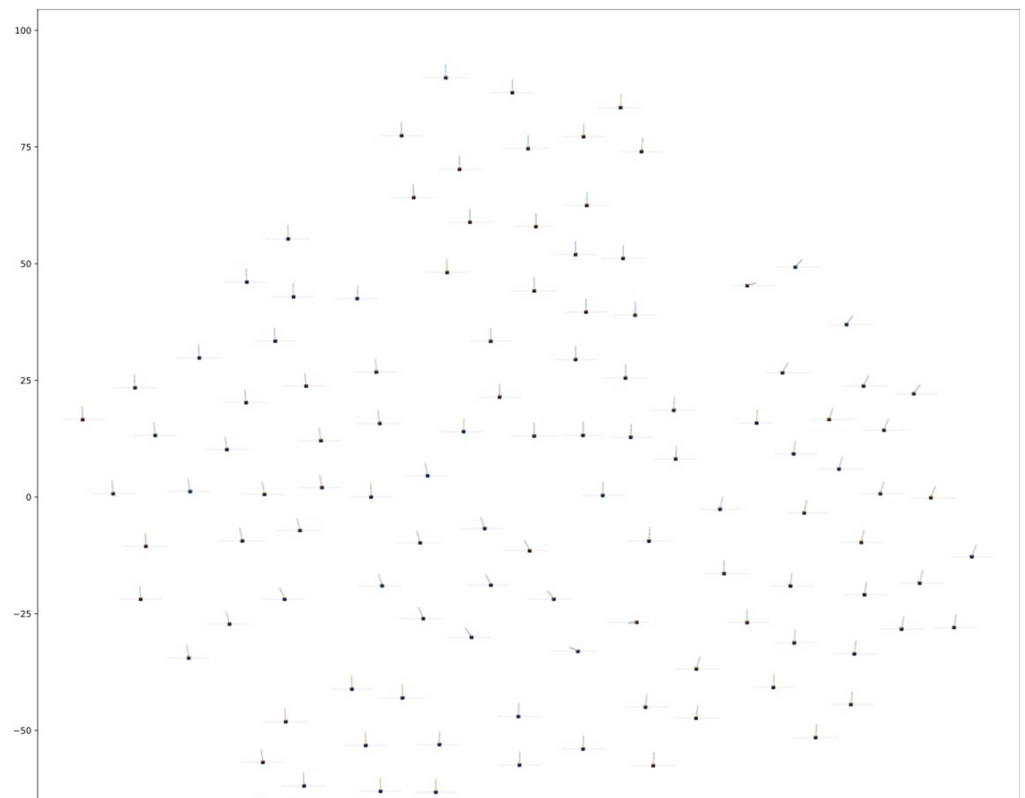


Figure A.1: E2C

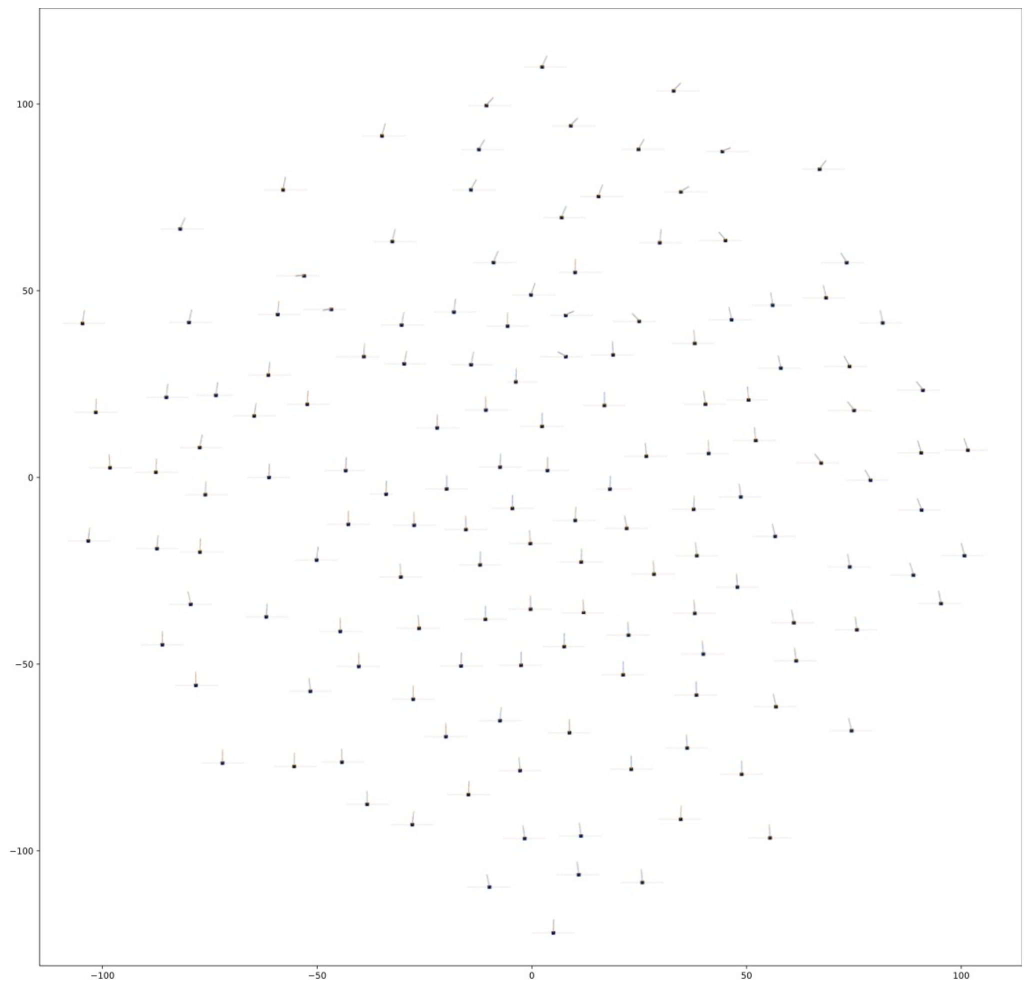


Figure A.2: VAE-Einsum

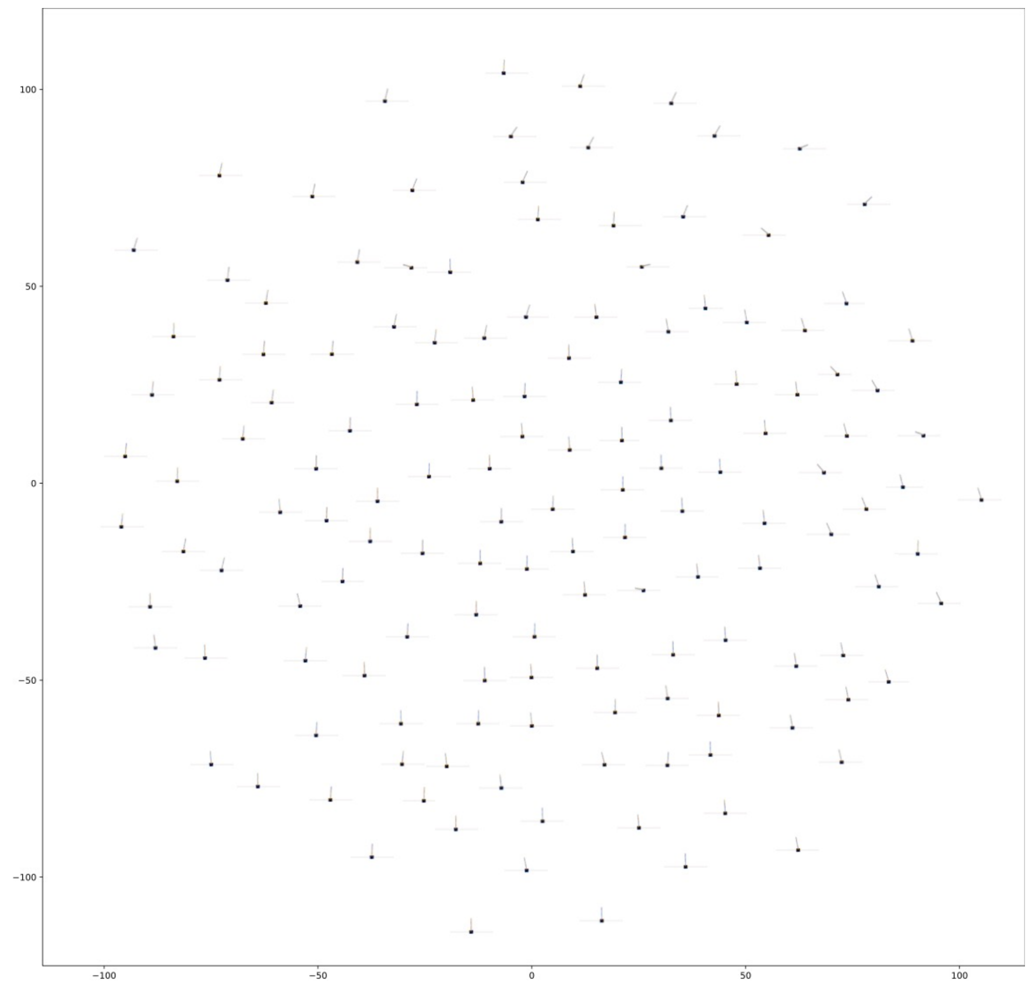


Figure A.3: VAE-Einsum-Decoupled

Appendix B

Parameter optimization - EM algorithm

We analysed the effect that the update frequency and step size of the EM algorithm had on the reconstruction of the einsum network in latent space. We chose three different values for each parameter and we analysed the effect of all possible combination. For the EM update frequency we chose values 1, 10 and 50. For the EM step size we chose values 0.1, 0.01 and 0.001. The following figure shows the results obtained for each combination on a subset of 4000 observations.

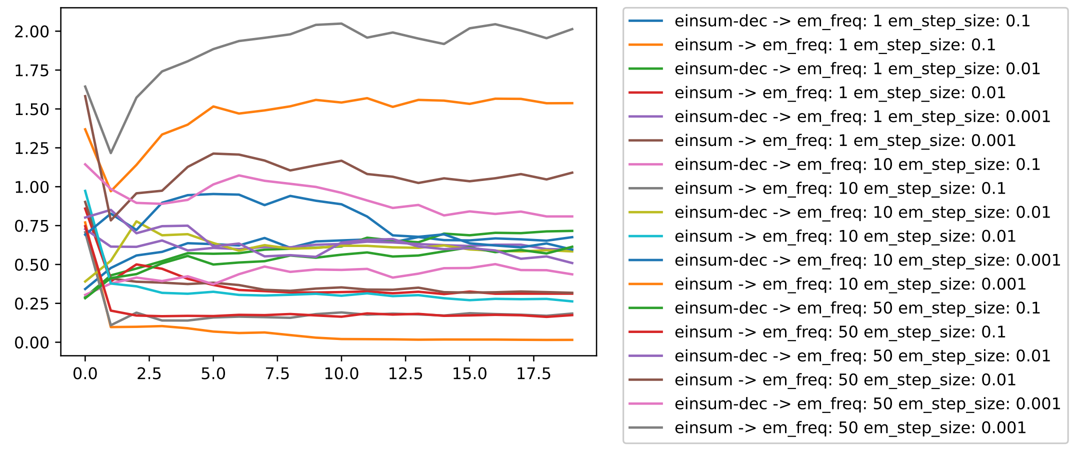


Figure B.1: Parameter optimization of EM algorithm parameters

We observed the best results for an update frequency of 10 and a step size 0.001.