UiO : Department of Physics
University of Oslo

FYS9429: Advanced machine learning and data analysis for the physical sciences

Project

VAE-based representation learning of simulated neural recordings

Jan Fredrik Kismul Nicolai Haug

June 23, 2024

Abstract

Representation learning is a fundamental concept in machine learning that involves automatically discovering and extracting meaningful features or representations from raw data. In this project, the focus is on applying Variational Autoencoders (VAEs) to perform representation learning of a binarized version of the MNIST dataset and neural data, specifically action potentials simulated by the Hodgkin-Huxley model. Our implementation of VAEs, using simple dense and convolutional neural networks with the regularization technique introduced by β-VAE, successfully learned latent encodings that could be reconstructed with minimal loss. In the case of MNIST, we were able to compress the original $28 \times 28 = 784$ pixels to 20 latent dimensions and still achieve adequate reconstruction. Remarkably, for neural data originally consisting of 5,000 datapoints, even a 2-dimensional latent space could achieve good reconstruction.

github.com/nicolossus/FYS9429-Project1

Contents

1. Introduction

When working with high-dimensional data, analysis and modeling often necessitate the compression of the data into a lower-dimensional representation. It is then crucial to extract the relevant information. The features of high-dimensional data can be expertly crafted using domain knowledge or found by tapping into the ability of modern machine learning methods to learn useful representations of the data directly.

In this project, the objective is to investigate whether variational autoencoders (VAEs) (D. P. Kingma and Welling, 2014) can be used to learn features from simulated neural recordings with the Hodgkin-Huxley model (Hodgkin and Huxley, 1952). We embark on this quest in the traditional way – by building simple models and testing them on the (binarized) MNIST dataset. Specifically, we will build VAEs with simple dense and convolutional neural networks as encoder/decoder architecture. In addition to reducing the dimensionality, it is desirable for the VAE to be able to extract meaningful and independent latent factors that explain the majority of the variation in the data. This is called disentangling the latent space and is facilitated by, for example, β-VAEs (Higgins et al., 2017). We will also build β-VAEs to explore the disentanglement of the latent space. Informed by experiments on the binarized MNIST dataset, we apply the VAEs on the the simulated neural data.

The report is organized as follows. In Section 2 we first provide the theoretical background of variational autoencoders and the dense and convolutional neural networks we use as encoder/decoder architectures. We also provide some brief background necessary to understand the neural data on which we ultimately will train the VAE model. In Section 3 we present the datasets and the different VAE architectures used, as well as a short description of the methods applied in the training of the VAEs. In Section 4 we present and discuss the results. Finally, in Section 5 we provide conclusions to the approach and outline possible avenues for future research.

2. Background

2.1. Autoencoders

An autoencoder is a type of artificial neural network that employs dimensionality reduction techniques for data compression and feature extraction of a dataset. The aim of an autoencoder is to find a latent feature representation of the data. It consists of two main components: an encoder and a decoder. As depicted in Figure 2.1, the encoder processes an input, such as an image, and compresses it into a compact representation of latent features. This representation is the most reduced form of feature representation that still allows reconstruction of the original input within a defined quality threshold. Subsequently, the decoder reconstructs the input from the compressed representation. The training objective of an autoencoder is to minimize the reconstruction error compared to the original input. The choice of encoder/decoder is adapted to the type of input data. For instance, convolutional neural networks (CNNs) are typically used for image data, whereas recurrent neural networks (RNNs) are suited for sequential data, such as time-series data.

Although autoencoders are effective in data compression and reconstruction, there is no stochasticity involved. This makes them inadequate for generating new data that fits a specific class. For example, an autoencoder trained on images of chairs can recreate these specific chairs, but will not create a variety of new chair images encompassing the full concept

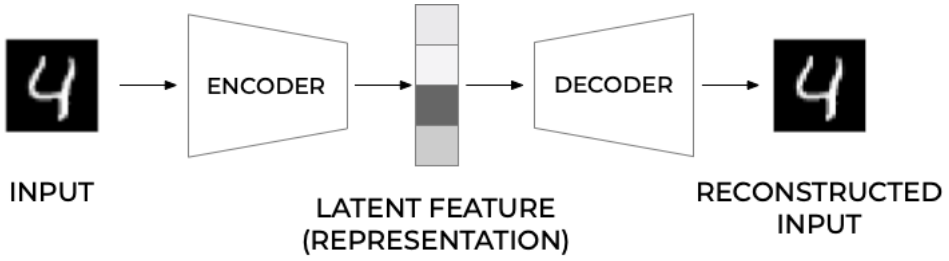

Figure 2.1: ***Autoencoder representation****: Here, an image depicting the digit 4 from the MNIST dataset is first sent through an encoder which compresses the image to a lower-dimensional (latent feature) representation. Then, the compressed representation is sent through a decoder which tries to reconstruct the original image.*

of "chairness". Moreover, the latent space of regular autoencoders may be discontinuous, making the generation and interpretation of specific features challenging or impossible. To introduce stochasticity and create continuous and practical latent spaces, Variational Autoencoders (VAEs) (D. P. Kingma and Welling, 2014) can be used. VAEs improve upon standard autoencoders by incorporating probabilistic elements, allowing the generation of new data instances that conform to the desired class.

2.2. Variational autoencoders

In a simplified way, stochasticity in VAEs is introduced within the latent feature space (see Figure 2.1). Instead of storing a single output value per feature, VAEs store a probability distribution, represented by a mean and a standard deviation for each feature. This approach ensures that specific features are continuous around their mean (via the standard deviation) and sufficiently separated to be distinguishable (via the mean).

The probability distribution in the latent feature space allows for sampling, enabling the generation of new outputs that are both novel and recognizable as belonging to the intended class, despite not being present in the training data.

To develop an effective generative model, the aim is to maximize the marginal likelihood $p(x)$ over all x. This can be challenging, as it requires either knowing the true data distribution or performing an infeasible integral over latent parameters. The solution to this problem is to derive the Evidence Lower Bound (ELBO) where evidence is the log-likelihood of the input data. Given a parametrized encoder with the goal of being as close as possible to the ground truth distributions; $q_\phi(z|x) \approx p(z|x)$, ELBO can be represented by the following equation:

$$\mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p(x,z)}{q_\phi(z|x)}\right] \tag{2.1}$$

Here, $q_\phi(z|x)$ is a parametrized encoder that approximates the true posterior distribution $p(z|x)$. The objective is to maximize the ELBO to optimize the model. By applying the chain rule of probability to this expression, the ELBO can be rewritten as:

$$\mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p(x,z)}{q_\phi(z|x)}\right] = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)||p(z)) \tag{2.2}$$

In this equation, $p_\theta(x|z)$ is a parametrized function that takes samples from the latent space z to generate new output. Stochasticity is included in creating the latent space, making the function deterministic. The first term on the right-hand side represents the reconstruction error, ensuring that the latent features can recreate the input data. The second term is the Kullback-Leibler divergence, which measures the difference between two probability distributions.

It is important to note that the function $q_\phi(z|x)$ serves as an encoder, taking data from x into z, while the function $p_\theta(x|z)$ serves as the decoder, transforming latent variable z into output x. The first term on the right-hand side in Equation 2.2 can be interpreted as the reconstruction error. This term ensures the model's ability to generate accurate latent features that can be used to recreate input data. The second term is a regularization term that measures how similar the encoder's inferred latent distribution is to the prior $p(z)$.

2.2.1. β-VAE

A challenge of VAE is that features in the latent space can become entangled. For example, in a VAE trained to generate faces, the feature controlling the head's angle might become entangled with the feature controlling the person's smile. Consequently, a generated person looking to the left might always appear sad, while one looking to the right might always appear happy. This issue can be resolved by introducing a hyperparameter $\beta > 1$ in front of the KL-divergence term in the ELBO equation (Higgins et al., 2017):

$$\mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p(x,z)}{q_\phi(z|x)}\right] = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \beta D_{KL}(q_\phi(z|x)||p(z)) \tag{2.3}$$

The β parameter forces the model to consider the chosen prior distribution during optimization. For example, if a multivariate Gaussian is chosen as the prior, the model will force the encoded values closer to this distribution. By increasing the cost of adding features to the latent layer, β reduces feature redundancy. In a VAE for generating faces, this means it would help eliminate unnecessary features, such as an extra feature encoding both hair and eye color when separate features for hair color and eye color already exist.

2.3. Multilayer perceptrons and dense neural networks

A multilayer perceptron (MLP) represents one of the fundamental types of neural networks, illustrated in Figure 2.2. The network takes a set of inputs X, which are the features or data to be studied, and connects them to the subsequent hidden layer to the right through a series of weights, each corresponding to a connection. Each node a in this hidden layer computes a weighted sum of inputs, augmented by a layer-specific bias term, and applies a non-linear activation function such as sigmoid, tanh, or ReLU. These activation functions introduce non-linearity into the model, enabling it to capture complex relationships between input variables and providing depth to the network (where multiple linear transformations cannot simply be replaced by a single linear transformation). The bias term adjusts the activation threshold of each node, analogous to a firing threshold in neuron models. The final layer $f(X)$ in Figure 2.2 integrates outputs from the preceding layer to produce the network's final output value(s). To construct a more sophisticated and potentially more powerful network, additional hidden layers are often added successively.

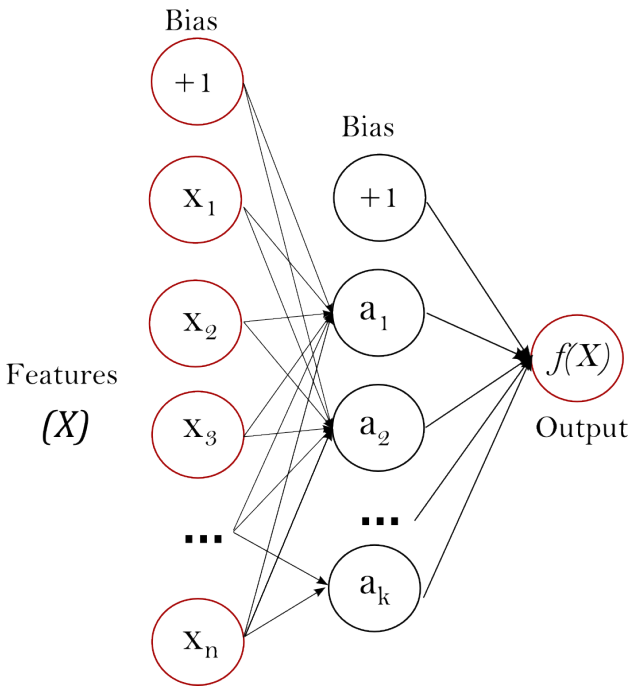

Figure 2.2: ***MLP****:Input data X is sent to a hidden layer, denoted a, and finally an output $f(X)$ is generated. Figure retrieved from sckikit-learn.org.*

Initially, the network will produce meaningless output until it is trained. Training involves adjusting the weights and biases for each hidden layer by comparing the generated output to known target data. For instance, when classifying digits from the MNIST dataset, a flattened array of grayscale values is input, and the network aims to classify them into digits 0 through 9. The error between the network's output and the correct answers is computed and propagated backward through the layers via backpropagation. At each layer, weights and biases are updated in accordance with the error to refine the model's predictions. Training continues iteratively until predefined criteria are met, such as reaching a specified number of iterations or no longer improving over several cycles. This process, known as model training, demands substantial amounts of data and computation time. A well-trained model with low error can then be deployed on unseen data to predict outputs that closely match the correct results.

A deep neural network (DNN) is constructed using the same fundamental elements as the MLP in Figure 2.2, but with increased complexity including additional hidden layers as illustrated in Figure 2.3. For instance, taking the input of 28x28 dimensions of MNIST data flattened to a 784-dimensional vector into consideration, MLP and DNN show different network architectures. In the case of MLP, the input is fed into a single hidden layer comprising 500 features, where the output, serving as the latent space in a VAE, is set to 20 features. DNN follows the same initial structures, but includes additional hidden layers subsequent to the first and features 150 features. Therefore, MLPs and DNNs utilize common building blocks, but differs in the depth and configuration of their hidden layers.

2.4. Convolutional neural networks

Convolutional Neural Networks (CNNs) are predominantly used in image processing tasks such as image recognition. They differ from other neural network types by incorporating

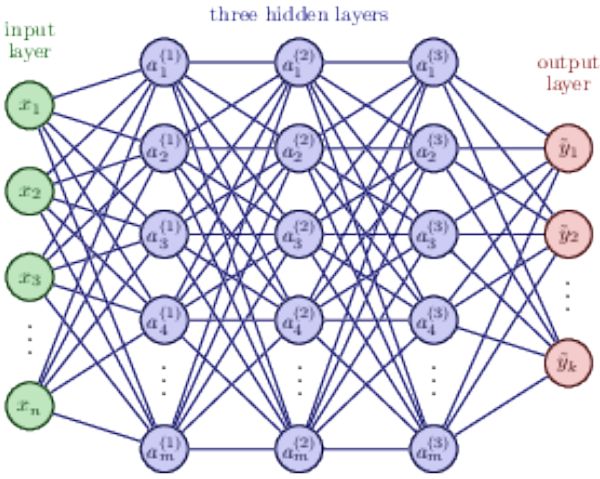

Figure 2.3: ***DNN****: Architecture of a deep neural network. An array of input data is sent via some weights and biases to a hidden layer a_1. Instead of sending the output of a_1 straight to an output layer, several more hidden layers are included.*

convolution operations in one or more layers instead of typically used matrix multiplications. This architectural choice is tailored to leverage the typical spatial structure of images, characterized by dimensions of width, height, and depth, which helps optimize network performance compared to more general architectures. In CNNs, the 3D nature of input data (width, height, and channels) is preserved across hidden layers, which are specialized into specific types. The key of convolution lies in the application of kernels (typically NxNxF filters) across the input data. Each kernel slides over the input data with a predefined stride, computing outputs by performing a dot product operation within its receptive field. This process, illustrated in Figure 2.4, transforms an initial 28x28x1 input image, for example, into a 28x28x32 output using a 5x5x32 kernel. In convolution operations like this, the focus is not on numerous weights and biases as in deep neural nets, but rather a smaller set of chosen parameters:

- **Kernel size**: The spatial dimensions of each filter, as well as the number of filters (NxNxF)
- **Stride**: The distance to move the kernel for each computation. The kernel is moved one stride length to the left each time until it reaches the end of the current row, then it will return to the initial position, move one stride down, and then repeat the whole process.
- **Padding**: Padding refers to adding a rim of zeros (typically) around your image, avoiding potential issues near the edges of the image, as well as to conserve the spatial dimension of the image being convolved.

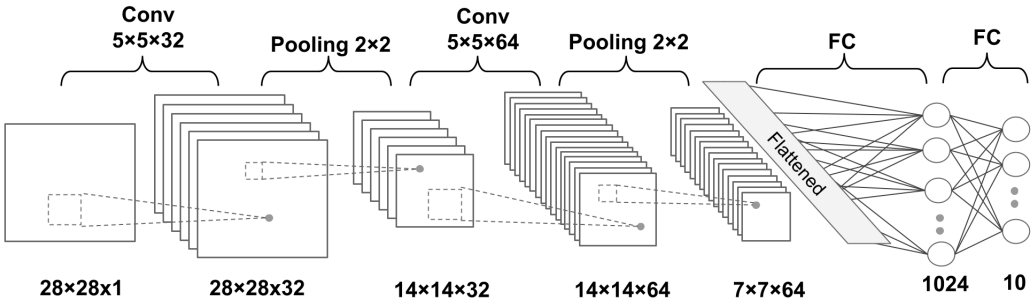

Figure 2.4: ***CNN architecture*** *A simple input of a grayscale image, of dimensions 28x28x1 is sent through two sets of convolutions and pooling before it is flattened, and sent through a set of fully connected layers (FC) in the end.*

Some benefits of such a network compared to a deep neural network are:

- they take higher dimensional data as input
- not flattening input allows for learning local correlations in all dimensions. Good for feature extraction.
- instead of a heap of parameters to chose from, they are very limited per operation
- can do unsupervised classification and feature extraction/recognition

Unlike deep neural networks that rely on extensive parameters, CNNs utilize a compact set of parameters per operation. Additionally, CNNs employ pooling layers to reduce spatial dimensions. Pooling involves applying an NxN window across the data and aggregating it into a single value, typically by taking the maximum or average. While effective for dimensionality reduction, pooling can discard nuanced features, thus smaller pool sizes are preferred. Following a sequence of convolutional and pooling layers, the final output is prepared for classification or another designated task. The output of the last pooling layer is flattened into a vector and fed into one or more fully connected layers, which perform the final computation.

2.5. Action potentials and the Hodgkin-Huxley model

Action potentials

Neurons are the fundamental units of the brain and are responsible for carrying information throughout the body. They do so by relaying electric signals called action potentials. Action potentials are the result of ionic currents that pass through the neuronal membrane. The neuronal membrane potential is affected by signals received from other neurons in its circuit, which can make the membrane potential less negative (depolarized) or more negative (hyperpolarized). If a neuron is depolarized sufficiently to raise the membrane potential above the membrane's threshold voltage, a positive feedback process is initiated which triggers an action potential. The temporal evolution of action potentials can be divided into different phases, as shown and described in Figure 2.5.

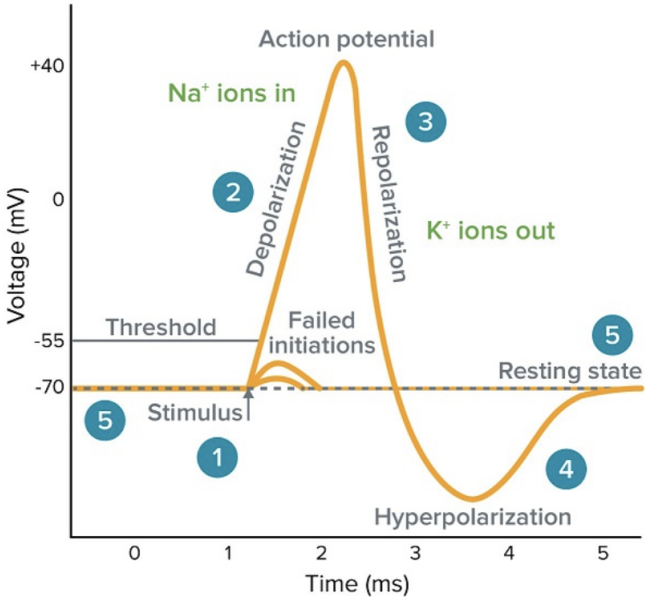

Figure 2.5: *The temporal evolution of the membrane potential during a typical action potential. The membrane potential initially rests at its baseline level when (1) a stimulus pushes it over the threshold. (2) This triggers a rapid rise in the membrane potential, known as depolarization, caused by the opening of voltage-gated sodium (Na^+) channels which give an influx of Na^+ ions into the neuron. (3) Following depolarization, there is a sharp decrease in the membrane potential called repolarization. This occurs as the sodium channels close and potassium (K^+) channels open, resulting in an efflux of K^+ ions. Typically, the membrane potential undershoots its resting level, leading to a brief period of hyperpolarization. (5) Finally, the membrane potential gradually returns to its resting state, completing the action potential. Retrieved from www.moleculardevices.com, 2023.*

The Hodgkin-Huxley model

The seminal Hodgkin-Huxley (HH) model (Hodgkin and Huxley, 1952) was the first biophysically detailed description of the ionic mechanisms underlying action potentials in neurons. At the core of the model are four ordinary differential equations that describe how the opening and closing of ion channels in the neuronal membrane are influenced by changes in membrane potential:

$$C_m \frac{\mathrm{d}V}{\mathrm{d}t} = -\bar{g}_{\mathrm{K}} n^4 (V - E_{\mathrm{K}}) - \bar{g}_{\mathrm{Na}} m^3 h (V - E_{\mathrm{Na}}) - \bar{g}_{\mathrm{L}} (V - E_{\mathrm{L}}) + I \tag{2.4a}$$

$$\frac{\mathrm{d}n}{\mathrm{d}t} = \alpha_n(V)(1 - n) - \beta_n(V) n \tag{2.4b}$$

$$\frac{\mathrm{d}m}{\mathrm{d}t} = \alpha_m(V)(1 - m) - \beta_m(V) m \tag{2.4c}$$

$$\frac{\mathrm{d}h}{\mathrm{d}t} = \alpha_h(V)(1 - h) - \beta_h(V) h \tag{2.4d}$$

By solving these equations, the model can simulate the behavior of a neuron and accurately reproduce the generation and propagation of action potentials. In particular, Equation 2.4a describes how the membrane potential V evolves over time. The variables n, m and h are dimensionless quantities that are associated with potassium channel activation, sodium channel activation and sodium channel inactivation, respectively. The terms on the form $\bar{g}_X(V - E_X)$ describe the current through a ion channel X as the product of that channel's total conductance $\bar{g}_X$ and the driving potential $(V - E_X)$, where E_X is the reversal potential, for the specific ion. For more details on the model, we refer to the original article or, e.g., the textbook by Sterratt et al., 2011. The values of the original model parameters are summarized in Table 2.1. Figure 2.6 shows the numerical solutions of a simulation of the HH model.

Table 2.1: *The original parametrization of the HH model.*

Parameter	**Value**	**Description**
C_m	$1.0\,\mu\mathrm{F\,cm^{-2}}$	Membrane capacitance
$\bar{g}_\mathrm{K}$	$36.0\,\mathrm{mS/cm^2}$	Maximum potassium channel conductance
$\bar{g}_\mathrm{Na}$	$120.0\,\mathrm{mS/cm^2}$	Maximum sodium channel conductance
$\bar{g}_\mathrm{L}$	$0.3\,\mathrm{mS/cm^2}$	Maximum leakage channel conductance
E_K	$-77.0\,\mathrm{mV}$	Potassium reversal potential
E_Na	$50.0\,\mathrm{mV}$	Sodium reversal potential
E_L	$-54.4\,\mathrm{mV}$	Leak reversal potential

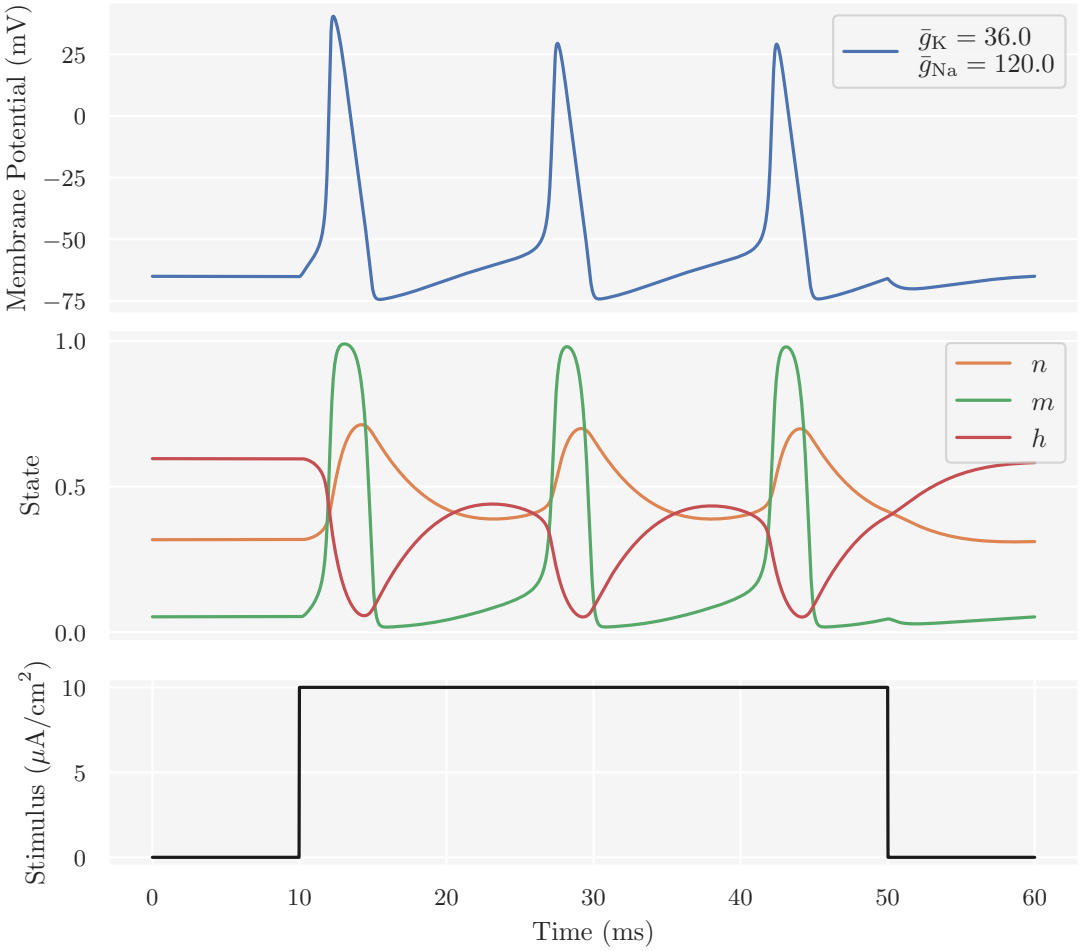

Figure 2.6: *Simulated dynamics of V, n, m and h in the HH model during the firing of action potentials. The system is simulated for $T = 60$ ms with time resolution $\Delta t = 0.025$ ms. The input stimulus received by the neuron (bottom panel) is a step current with amplitude $I = 10\,\mu\text{A/cm}^2$, and onset and offset at 10 ms and 50 ms, respectively. The parametrization of the model is given by Table 2.1.*

3. Methodology

3.1. The data

3.1.1. The MNIST dataset

The original MNIST dataset consists of 60,000 training and 10,000 test images of 28x28 pixel grayscale images of handwritten digits (0-9). Each image is labeled with the corresponding digit. The pixel values in the original dataset range from 0 to 255, representing the intensity of grayscale. In our experiments, we reduce the complexity of the dataset by binarizing the images. All pixel values greater than or equal to 127 are set to 1 (white), while values below 127 are set to 0 (black). The binarized MNIST dataset is commonly used in machine learning tasks where feature extraction is based on the presence or absence of pixels rather than the grayscale intensity.

In order to speed up training for our experiments, we only train and test the model on images with the labels 0-3. Samples from the training set are shown in Figure 3.1.

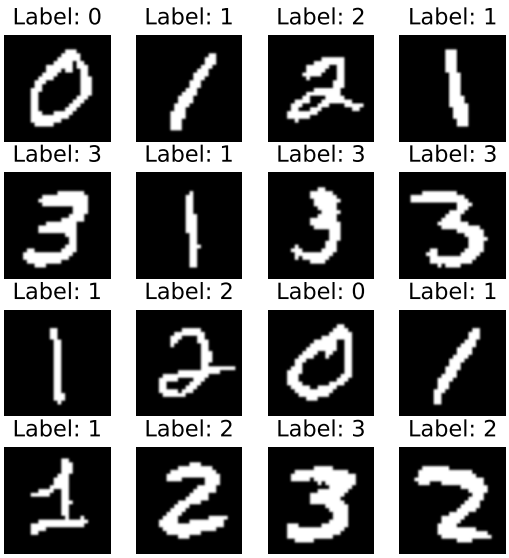

Figure 3.1: *Samples from the binarized MNIST dataset with labels 0-3.*

3.1.2. Simulated neural data

We created a dataset of 10,000 simulated voltage traces from the Hodgkin-Huxley model using the following simulation protocol:

- The active conductance parameters were sampled randomly to generate distinct voltage traces, with $\bar{g}_K \sim \mathrm{U}(26, 46)\,\mathrm{mS/cm}^2$ and $\bar{g}_{Na} \sim \mathrm{U}(110, 130)\,\mathrm{mS/cm}^2$.
- The rest of the model parameters were set to the values provided in Table 2.1.
- In order to equilibrate the system, the system was first simulated for 200 ms without external stimuli. Then, we started recording and turned on a step current pulse with amplitude $I = 15\,\mu\mathrm{A/cm}^2$ after 10 ms that lasted throughout the rest of the recorded simulation.
- We kept a recording of 50 ms. The time step of the simulation was 0.01 ms. This means that the time series we will train the VAEs on have a length of 5000 data points.

As a post-processing step, we applied min-max scaling to ensure voltage values on $[0, 1]$:

$$V_{\text{scaled}} = \frac{V - \min(V)}{\max(V) - \min(V)} \tag{3.1}$$

8,000 of the simulation form the training set, and the remaining 2,000 the test set.

3.2. VAE models

3.2.1. MLP- and DNN-VAE

Our first model is a MLP-VAE, for which the encoder and decoder consists of an identical single hidden dense layer with the number of units set to 500. To investigate if there are some performance gains by using a slightly deeper dense network, we also build a DNN-VAE

with 500 units in the first layer and 150 in the second of the encoder. The decoder of the DNN-VAE mirrors its encoder.

3.2.2. Convolutional VAE

We also built a convolutional VAE. The encoder network employs two convolutional layers, the first with 32 filters and the second 64, followed by a fully-connected layer. The decoder network mirrors this architecture by using a fully-connected layer followed by three convolution transpose layers in reverse order of the encoder.

Activation function and weight initialization

As activation function, we use the Rectified Linear Unit (ReLU):

$$\text{ReLU}(x) = \max(0,\ x) \tag{3.2}$$

To maintain the variance of ReLU activations across layers, thereby preventing vanishing or exploding gradients, He et al., 2015 showed that weights, w, should be initialized with values drawn from a normal distribution with a mean of 0 and a standard deviation of $\sqrt{\frac{2}{n_{in}}}$, where n_{in} is the number of input units in the weight tensor. This is promptly known as He initialization.

3.2.3. Latent distribution prior

As prior on the latent distribution, $p(z)$, we choose the commonly used multivariate standard normal distribution:

$$p(z) = \text{N}\Big(\mu = \mathbf{0}, \sigma^2 = \mathbf{1}\Big)$$

This encourages encodings to distribute evenly around the center of the latent space.

3.3. Training the VAEs

3.3.1. The NAdamW optimizer

We use the NAdamW optimizer as the stochastic gradient descent method for updating the parameters of the VAE's encoder and decoder. NAdamW is an optimizer combining Adam (Diederik P. Kingma and Ba, 2014) with weight decay (Loshchilov and Hutter, 2019), then called AdamW, and momentum a la Nesterov introduced by Dozat, 2016. The weight decay regularize learning towards small weights, which leads to better generalization.

Let α_t represent the learning rate, β_1, β_2 the exponential decay factors for the first and second moments, respectively, and ε, $\bar{\varepsilon}$ small constants applied to the denominator outside and inside, respectively, the square root to avoid dividing by zero when rescaling. The learning rate is indexed by t since the learning rate may also be provided by a schedule function (see Section 3.3.2). Let λ be the weight decay and θ_t the parameter vector at time t. The optimizers internal state $S_t := (m_t, v_t)$ represents the running average of the first and second moment of the gradient. We initialize the internal state as $S_0 = (m_0, v_0) = (0, 0)$. At step t, the optimizer takes incoming gradients g_t and computes updates u_t and the new state S_{t+1}. For $t > 0$, we have,

$$
\begin{aligned}
m_t &\leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t \\
v_t &\leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot {g_t}^2 \\
\hat{m}_t &\leftarrow \beta_1 m_t/(1-\beta_1^{t+1}) + (1-\beta_1) g_t/(1-\beta_1^t) \\
\hat{v}_t &\leftarrow v_t/(1-\beta_2^t) \\
u_t &\leftarrow -\alpha_t \cdot \left(\hat{m}_t / \left(\sqrt{\hat{v}_t + \bar{\varepsilon}} + \varepsilon\right) + \lambda \theta_t\right) \\
S_t &\leftarrow (m_t, v_t).
\end{aligned}
$$

3.3.2. Learning rate scheduler

The learning rate is one of the most important hyperparameters for training deep neural networks, as it determines the size of the steps taken during the optimization process. Instead of employing a fixed learning rate, it is common to dynamically adjust the learning rate during training according to a schedule. A learning rate scheduler can notably enhance convergence and overall model performance. In the training of the VAEs, we will use the *cosine decay schedule* (Loshchilov and Hutter, 2016). It starts with an initial learning rate and gradually decreases the learning rate following a cosine curve. The learning rate at iteration t is given by

$$
\frac{I(1-\alpha)}{2}\left(1 + \cos\left(\pi \frac{t}{T}\right)^p\right) + \alpha, \tag{3.3}
$$

where T is the number of decay steps, p is an exponent that can be used to modify the decay curve, I is the initial learning rate and α determines the final learning rate as a fraction of the initial learning rate. During training, we set

$$
\begin{aligned}
T &= \text{number of examples in a batch} \times \text{number of epochs}, \\
p &= 1, \\
I &= 10^{-3},
\end{aligned}
$$

and

$$
\alpha = 10^{-2}
$$

which means that the final learning rate will be 10^{-5}. Figure 3.2 illustrates the cosine decay scheduler.

3.3.3. Loss functions

For a standard normal prior, the KL-divergence (KLD) term of the ELBO becomes:

$$
D_{KL}(q_\theta(z|x)||p(z)) = -\frac{1}{2}\sum_j \left(1 + \log\left(\sigma_j^2\right) - \mu_j^2 - \sigma_j^2\right) \tag{3.4}
$$

The reconstruction error of the binarized MNIST images is computed as the binary cross-entropy (BCE):

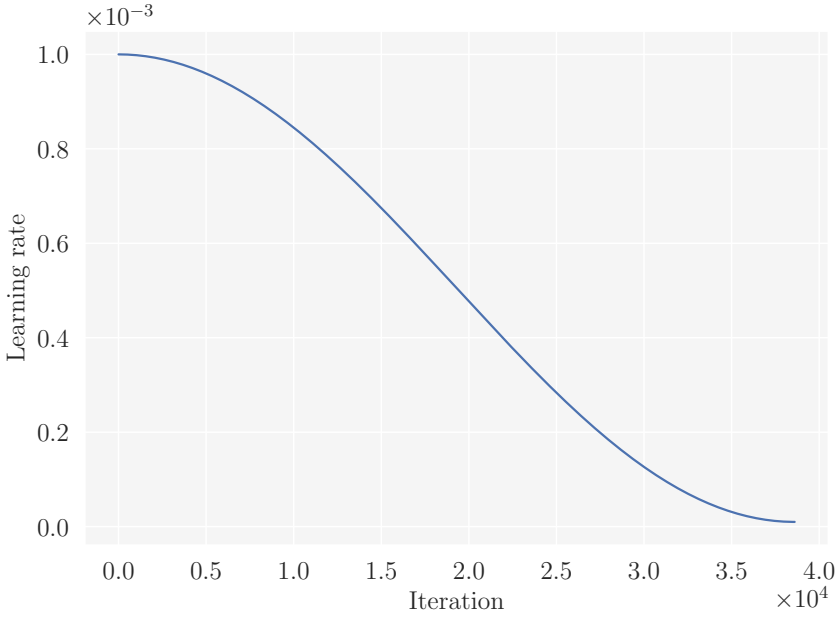

Figure 3.2: *Illustration of the cosine decay learning rate scheduler given by Equation 3.3. The variables of the cosine decay scheduler are set according to the description in the main text with 386 examples in one batch and 50 epochs.*

$$BCE = -\frac{1}{n}\sum_{i=1}^{n}\left(x_i \log\left(\hat{x}_i\right) + \left(1 - x_i\right)\log\left(1 - \hat{x}_i\right)\right), \tag{3.5}$$

where x_i is an original image and $\hat{x}_i$ its reconstruction.

The reconstruction error of the simulated neural data is computed as the mean-squared error (MSE):

$$MSE = \frac{1}{n}\sum_{i}^{n}\left(x_i - \hat{x}_i\right)^2, \tag{3.6}$$

where x_i is the original signal and $\hat{x}_i$ the reconstruction.

4. Results and Discussion

4.1. Binarized MNIST experiments

We initially examine the VAE models constructed using the traditional binarized MNIST dataset. In Figure 4.1, we observe the ELBO and its components, specifically the reconstruction error (BCE) and the KL-divergence (KLD), throughout training for various latent space dimensions. The figure illustrates that increasing the dimension reduces the BCE but increases the KLD. This is consistent with the anticipated outcome that higher latent dimensionality leads to better reconstruction. Nonetheless, VAEs are intended to compress data, so maintaining a low dimensionality is also desirable. The figure indicates that a latent dimension of 20 yields comparable performance to one with 100, suggesting 20 as an optimal dimension for the latent space. Although the different VAE models demonstrate similar performance, Conv-VAE generally exhibits slightly better results compared to MLP- and DNN-VAE in terms of loss. This outcome aligns with expectations, as CNNs are generally superior feature extractors compared to simple DNNs.

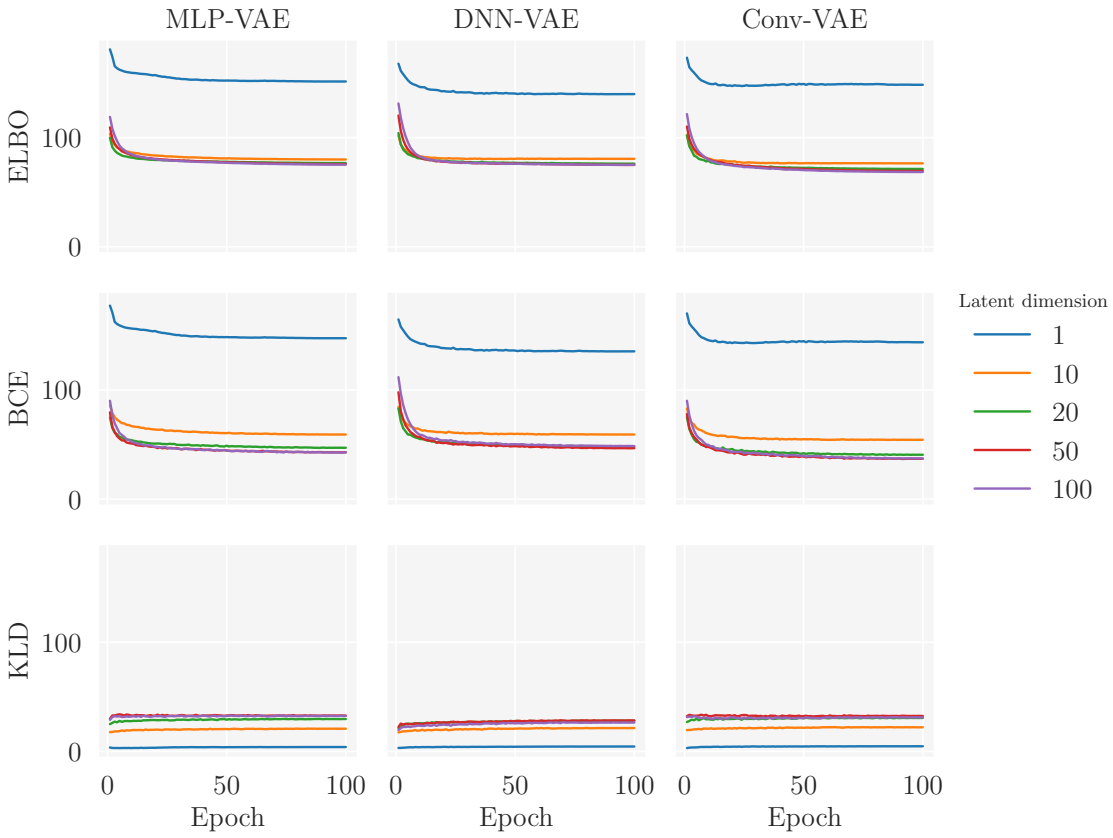

Figure 4.1: *The loss during training of the VAEs for various latent dimensions.*

Once the training was completed, we proceeded to input test set images into the VAEs. In Figure 4.2, we present a comparison between a subset of the reconstructed images alongside the original images. The left column (a, d, g) displays outputs from the MLP-VAE, the middle column (b, e, h) the DNN-VAE, and the right column (c, f, i) the Conv-VAE. The top, middle, and bottom rows shows the outcomes for 1, 20, and 100 latent dimensions, respectively. Analyzing the figure, it becomes evident that reducing the latent dimension results in increased blurriness in the reconstructed images. This correlates with the fact that a higher latent dimension leads to improved reconstruction. However, the disparity in

blurriness between a latent dimension of 20 and 100 is not significant, further suggesting that 20 latent dimensions are an optimal choice.

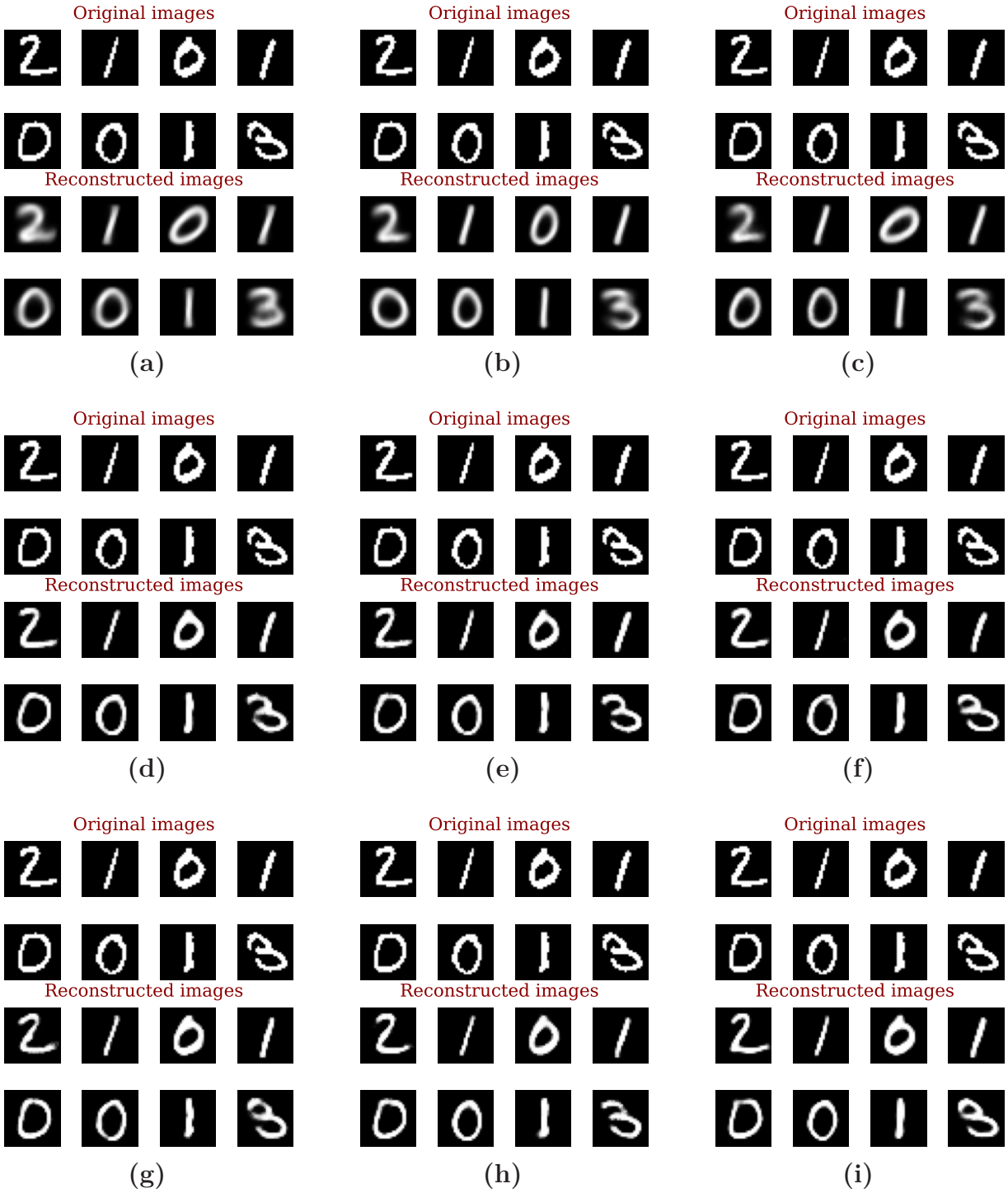

Figure 4.2: *Comparison of original and reconstructed images with the MLP-VAE on the left column, DNN-VAE on the middle column and Conv-VAE on the right column. The top row displays images for 1 latent dimension, the middle row for 20 latent dimensions, and the bottom row for 100 latent dimensions.*

VAEs are generative models that can generate new samples by sampling from the latent space. This aspect of VAEs is outside the scope of this study. We do, however, provide a glimpse of the generated images from the VAE models above in Figure A.1.

Isolating meaningful and independent factors of variation in the latent space, which is called disentangling the latent space, is important for dimension reduction. Next, we explore the disentanglement of the latent space by employing β-VAEs. The previous findings can be considered as outcomes from a β-VAE with a value of $\beta = 1$. Informed by the previous findings, we will only use the Conv-VAE and a latent dimensionality of 20.

Figure 4.3 shows the trade-off relationship of the β-VAE's specific regularization term to the loss. The figure clearly shows the trade-off between reconstruction accuracy and disentanglement, represented by the BCE and KLD, respectively. When β is low, the VAE prioritizes reconstruction accuracy over disentanglement. This means that the latent variables might not be well-separated or disentangled, and their distributions might overlap. As β increases, the model puts more emphasis on disentangling the latent variables. This encourages each latent variable to capture a specific and identifiable feature of the data. The intersection of the curves in the figure indicates that $\beta = 2$ is a sensible choice for this data.

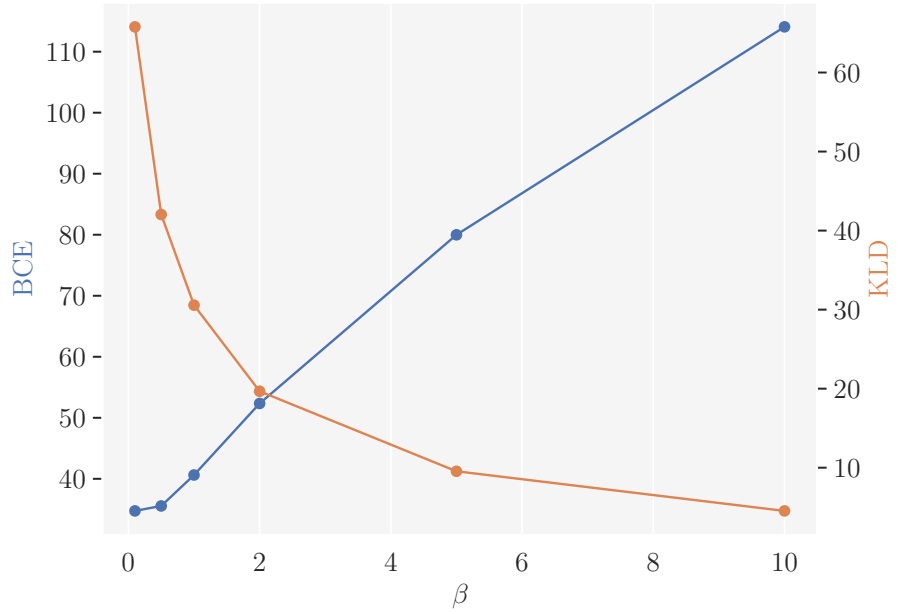

Figure 4.3: *The trade-off relationship between the loss, decomposed into the reconstruction error (BCE) and KL-divergence (KLD), and β in a convolutional β-VAE.*

Figure 4.4 compares a subset of the reconstructed images with the originals. In the left panel, $\beta = 0.1$ and we see that the reconstruction error is minimal. However, cf. the discussion for Figure 4.3, the latent space is more entangled. It is hence likely that the VAE "cheats" by clustering points in specific regions, i.e., memorizing the data. The middle panel shows the reconstructed images with $\beta = 2$, which was identified as a reasonable choice above. We see here that reconstruction is fairly good. Finally, the right panel shows the reconstructed images with $\beta = 10$. Here, the reconstruction is more blurry, but it is still possible to recognize the original digit in the reconstruction.

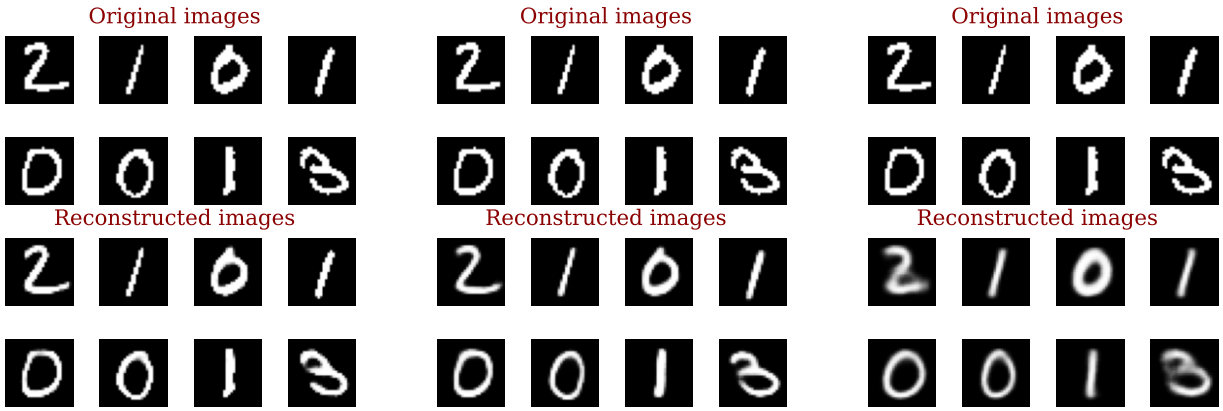

Figure 4.4: *Comparison of original and reconstructed images with the convolutional β-VAE. The left, middle and right panels shows the reconstruction with $\beta = 0.1, 2, 10$, respectively.*

4.2. Neural data experiments

We now move on to the study's primary objective – learning latent representations of the neural data simulated with the Hodgkin-Huxley model. Here, we exclusively use the convolutional β-VAE.

The loss progression during training for various values of β and latent dimensions is shown in Figure 4.5. Each figure title and legend represents the specific values used. Notably, the figure shows that the ELBO and its constituents, the MSE and KLD, exhibit striking similarity across all latent space dimensions for each instance of the β-VAE. However, when using a value of $\beta = 2$, the training becomes more unstable and the final loss is slightly higher compared to $\beta = 1$ and $\beta = 0.5$. Additionally, the loss does not seem to have fully converged, suggesting that regularizing for a more disentangled latent space requires additional epochs for better optimization.

One remarkable observation from the figure is that a 2-dimensional latent space performance is similar to higher-dimensional spaces. Figure 4.6, 4.7 and 4.8 shows the reconstructed voltage traces for the convolutional β-VAE with $\beta = 1$ and latent dimensionality 20, 10 and 2, respectively. All reconstructions closely match the original signal and exhibit smoothness in the suprathreshold region. In the subthreshold region, however, the reconstruction exhibits increased fluctuation, suggesting that capturing features of the dynamics in this area is more challenging. Notably, there are no discernible differences in the reconstruction quality across the various latent dimensions, further reinforcing the earlier observation that a 2-dimensional latent space sufficiently captures the relevant information. We obtain similar observations with $\beta = 0.5$ and $\beta = 2$, see Figure A.3 and Figure A.4, respectively.

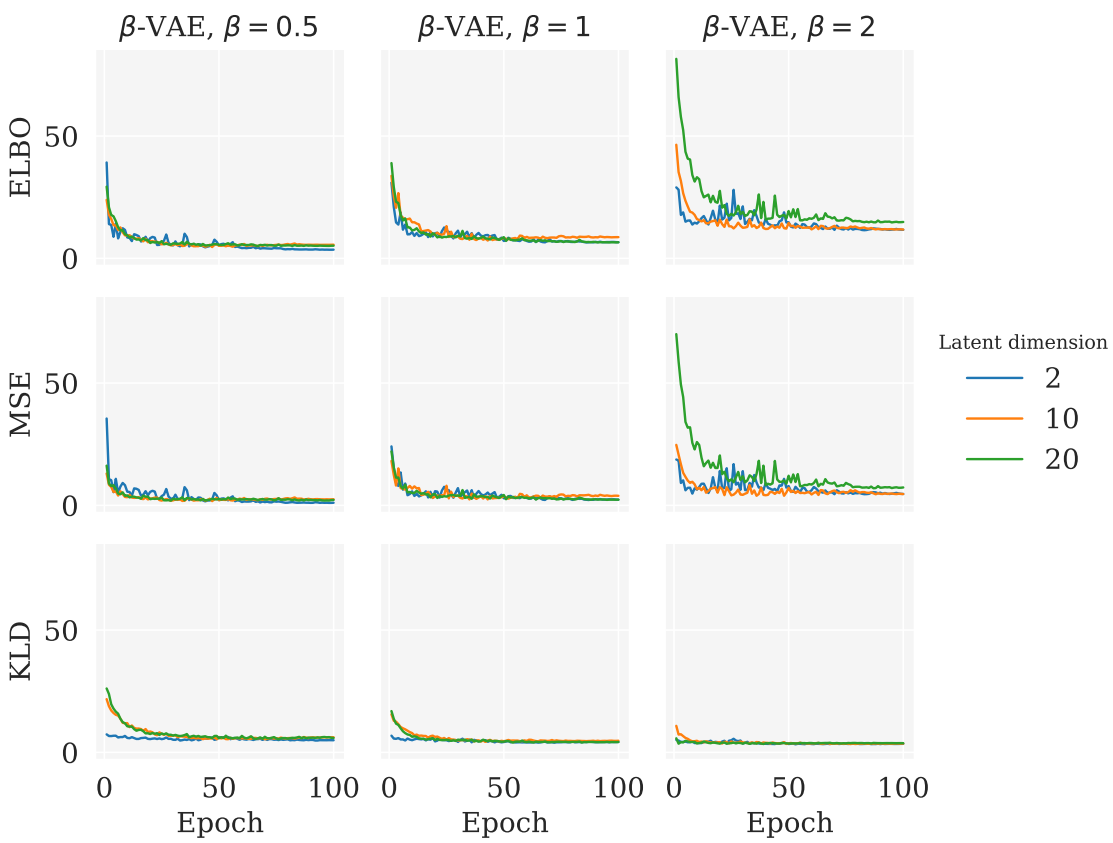

Figure 4.5: *The loss during training of the VAEs for various latent dimensions.*

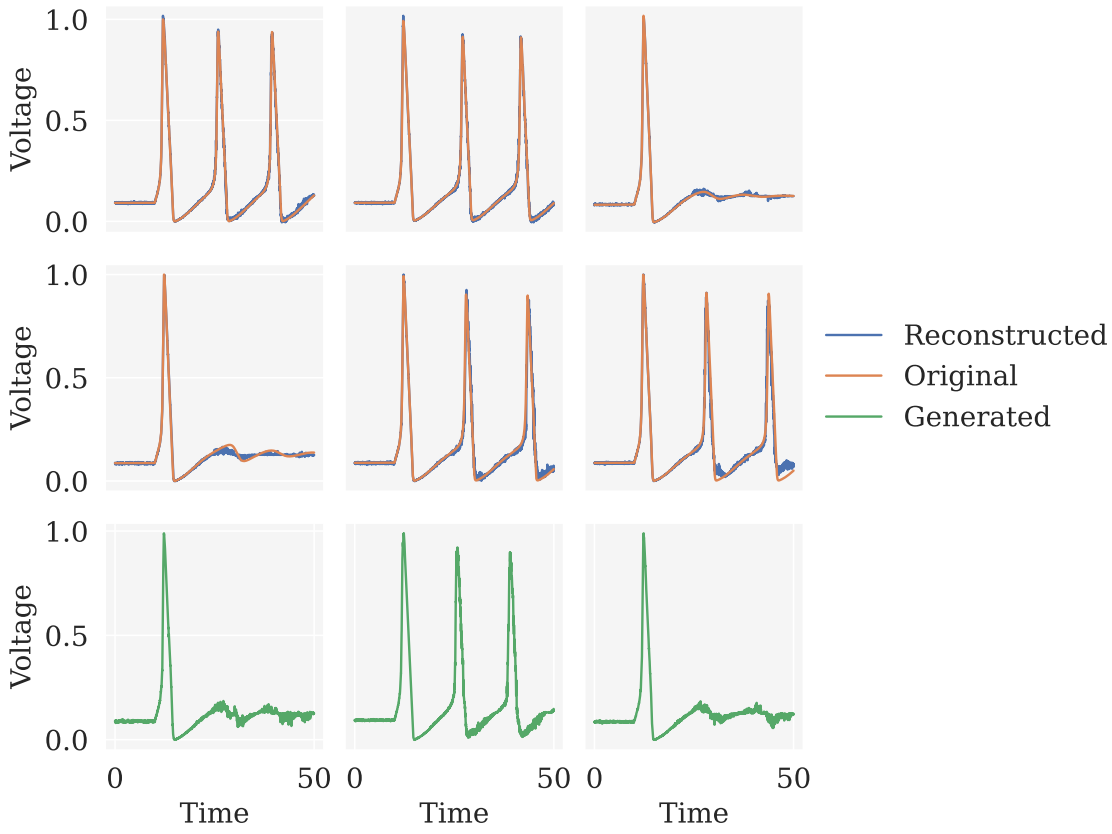

Figure 4.6: *Reconstructed voltage traces overlayed with the original signals for a convolutional β-VAE with $\beta = 1$ and 20 latent dimensions (top and middle panels). The bottom panel shows novel voltage traces generated from the latent space.*

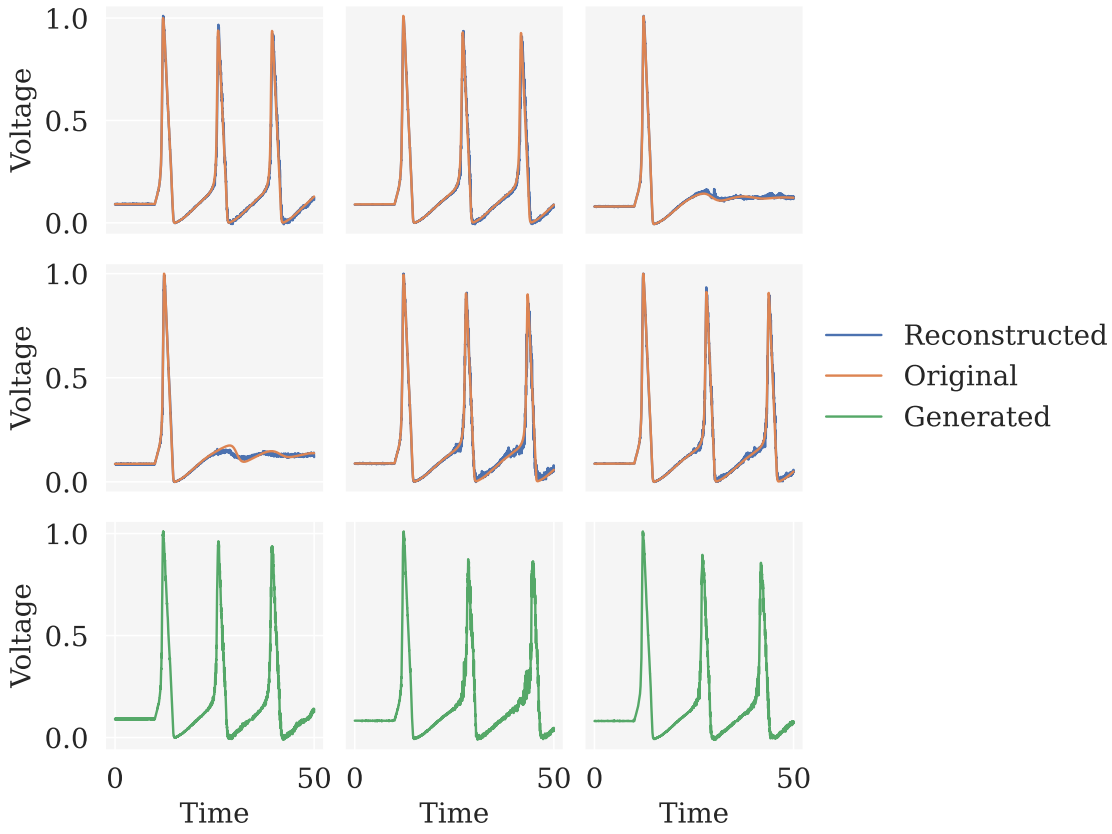

Figure 4.7: *Reconstructed voltage traces overlayed with the original signals for a convolutional β-VAE with $\beta = 1$ and 10 latent dimensions (top and middle panels). The bottom panel shows novel voltage traces generated from the latent space.*

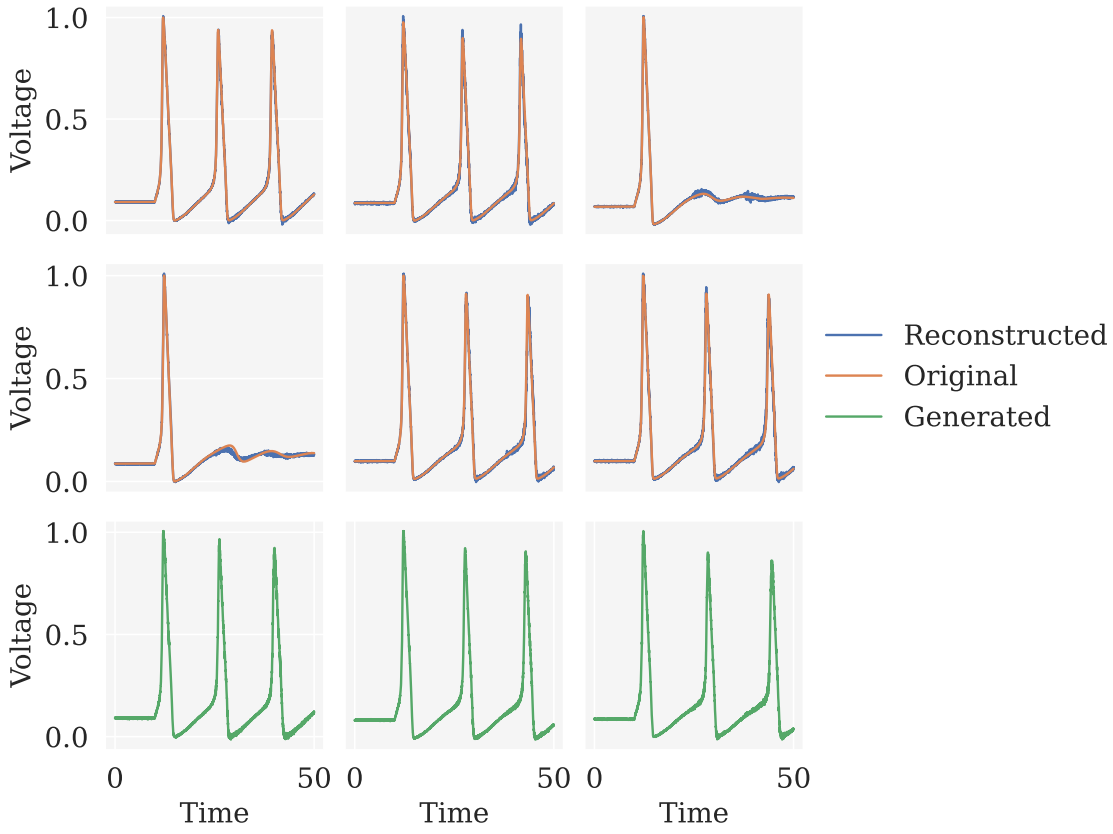

Figure 4.8: *Reconstructed voltage traces overlayed with the original signals for a convolutional β-VAE with $\beta = 1$ and 2 latent dimensions (top and middle panels). The bottom panel shows novel voltage traces generated from the latent space.*

5. Conclusion and future research

In this project, we have applied VAEs for representation learning on the binarized MNIST dataset and neural data simulated by the Hodgkin-Huxley model.

In experiments with the binarized MNIST dataset, we utilized two simple dense and one convolutional neural networks as encoder/decoder architecture in our VAE models to compare differences in performance and outcomes. All three models successfully learned low-dimensional latent encodings of the original $28 \times 28 = 784$ pixel images that could be reconstructed with minimal loss, but the convolutional VAE proved to be slightly superior as a feature extractor. Interestingly, there was little discrepancy in performance between a latent dimension of 20 and 100, suggesting that 20 latent dimensions may be an optimal choice. This notion was further supported by the comparison of original and reconstructed images from MNIST. We also used a convolutional β -VAE to facilitate disentanglement of the 20-dimensional latent space. Analysis of the trade-off relationship between reconstruction accuracy and disentanglement suggested $\beta = 2$ as an optimal value, which was also supported by comparing the original and reconstructed images.

In experiments with neural data simulated from the Hodgkin-Huxley model, we used the convolutional β-VAE. From the original size of 5,000 data points, the convolutional β-VAE was able to learn 20-, 10- and 2-dimensional latent representations that could be adequately reconstructed. The reconstruction was more smooth in the suprathreshold region than in the subthreshold region, which had small fluctuations. However, the encoder and decoder networks we chose here are relatively simple. Applying more complex networks will likely make the reconstruction smoother on both sides of the action potential firing threshold.

This study serves as a proof-of-concept that VAEs can be used to learn latent representations of neural data. However, how much information that is retained in this compressed representation compared to expertly-crafted features, remains an exciting avenue for further research.

References

Kingma, D. P. and M. Welling (2014). *Auto-Encoding Variational Bayes*. arXiv: 1312.6114 (cit. on pp. 1, 2).

Hodgkin, A. L and A. F Huxley (1952). "A quantitative description of membrane current and its application to conduction and excitation in nerve". In: *The Journal of physiology* 117.4, pp. 500–544 (cit. on pp. 1, 7).

Higgins, Irina, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner (2017). "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=Sy2fzU9gl (cit. on pp. 1, 3).

www.moleculardevices.com (2023). URL: https://www.moleculardevices.com/applications/patch-clamp-electrophysiology/what-action-potential (visited on 06/20/2023) (cit. on p. 7).

Sterratt, David, Bruce Graham, Andrew Gillies, and David Willshaw (2011). *Principles of Computational Modelling in Neuroscience*. 1st ed. Cambridge, UK: Cambridge University Press (cit. on p. 8).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2015). "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *IEEE International Conference on Computer Vision (ICCV 2015)* 1502. DOI: 10.1109/ICCV.2015.123 (cit. on p. 11).

Kingma, Diederik P. and Jimmy Ba (2014). *Adam: A Method for Stochastic Optimization*. arXiv: 1412.6980 (cit. on p. 11).

Loshchilov, Ilya and Frank Hutter (2019). *Decoupled Weight Decay Regularization*. arXiv: 1711.05101 (cit. on p. 11).

Dozat, Timothy (2016). "Incorporating Nesterov Momentum into Adam". In: *Proceedings of the 4th International Conference on Learning Representations*, pp. 1–4 (cit. on p. 11).

Loshchilov, Ilya and Frank Hutter (2016). "SGDR: Stochastic Gradient Descent with Restarts". In: *CoRR* abs/1608.03983. DOI: 10.48550/arXiv.1608.03983. arXiv: 1608.03983 (cit. on p. 12).

A. Appendix

A.1. Additional Results

Binarized MNIST experiments

Figure A.1 shows generated samples. The left column (a, d, g) displays outputs from the MLP-VAE, the middle column (b, e, h) the DNN-VAE, and the right column (c, f, i) the Conv-VAE. The top, middle, and bottom rows shows the outcomes for 1, 20, and 100 latent dimensions, respectively.

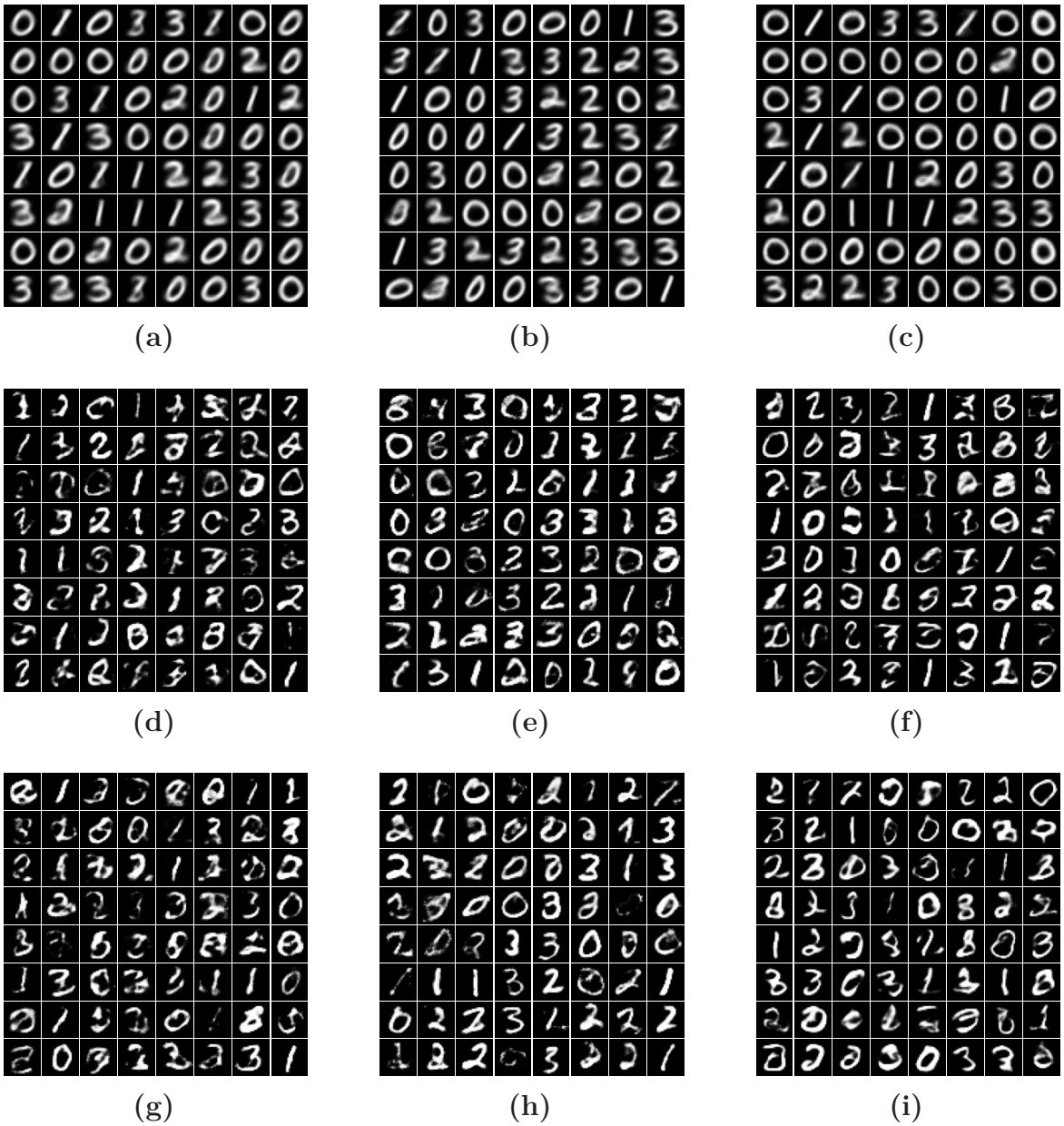

Figure A.1: *Comparison of generated images with the MLP-VAE on the left, the DNN-VAE in the middle, and the Conv-VAE on the right. The top row displays generated images for 1 latent dimension, the middle row for 20 latent dimensions, and the bottom row for 100 latent dimensions.*

Neural data experiments

Figure A.2 shows the ELBO loss components after training the convolutional β-VAE on the HH dataset over 100 epochs for different latent space dimensions, $\dim(z)$.

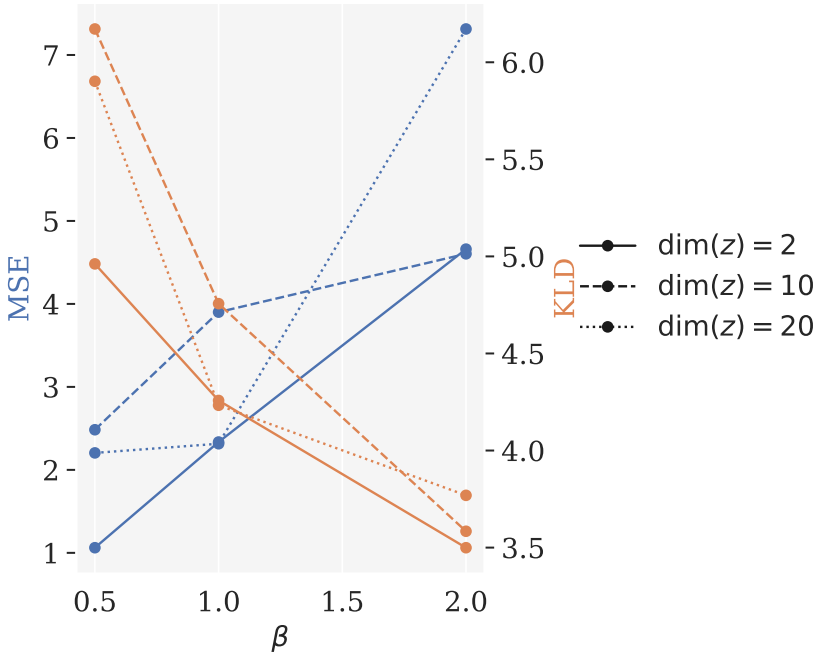

Figure A.2: *The tradeoff between the reconstruction (MSE) and regularization (KLD) terms of the ELBO.*

Figure A.3 and Figure A.4 are supplementary to Figure 4.6, 4.7 and 4.8, and show the convolutional β-VAE's reconstructed and generated voltage traces for $\beta = 0.5$ and $\beta = 2$, respectively, with $\dim(z) = \{2, 10, 20\}$.

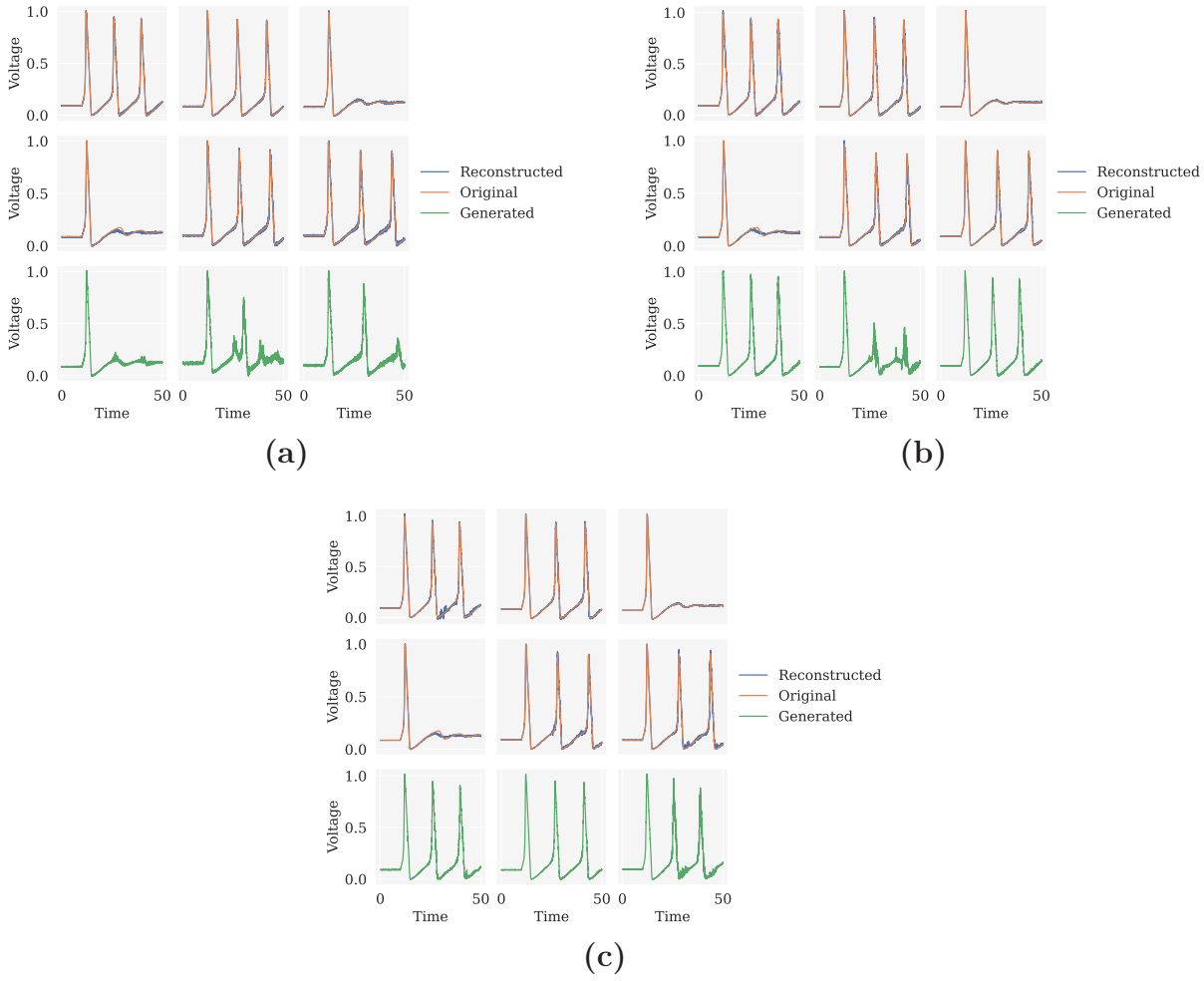

Figure A.3: *Reconstructed voltage traces overlayed with the original signals for a convolutional β-VAE with $\beta = 0.5$, together with novel voltage traces generated from the latent space. The latent dimensions are (a) 2, (b), 10, (c) 20.*

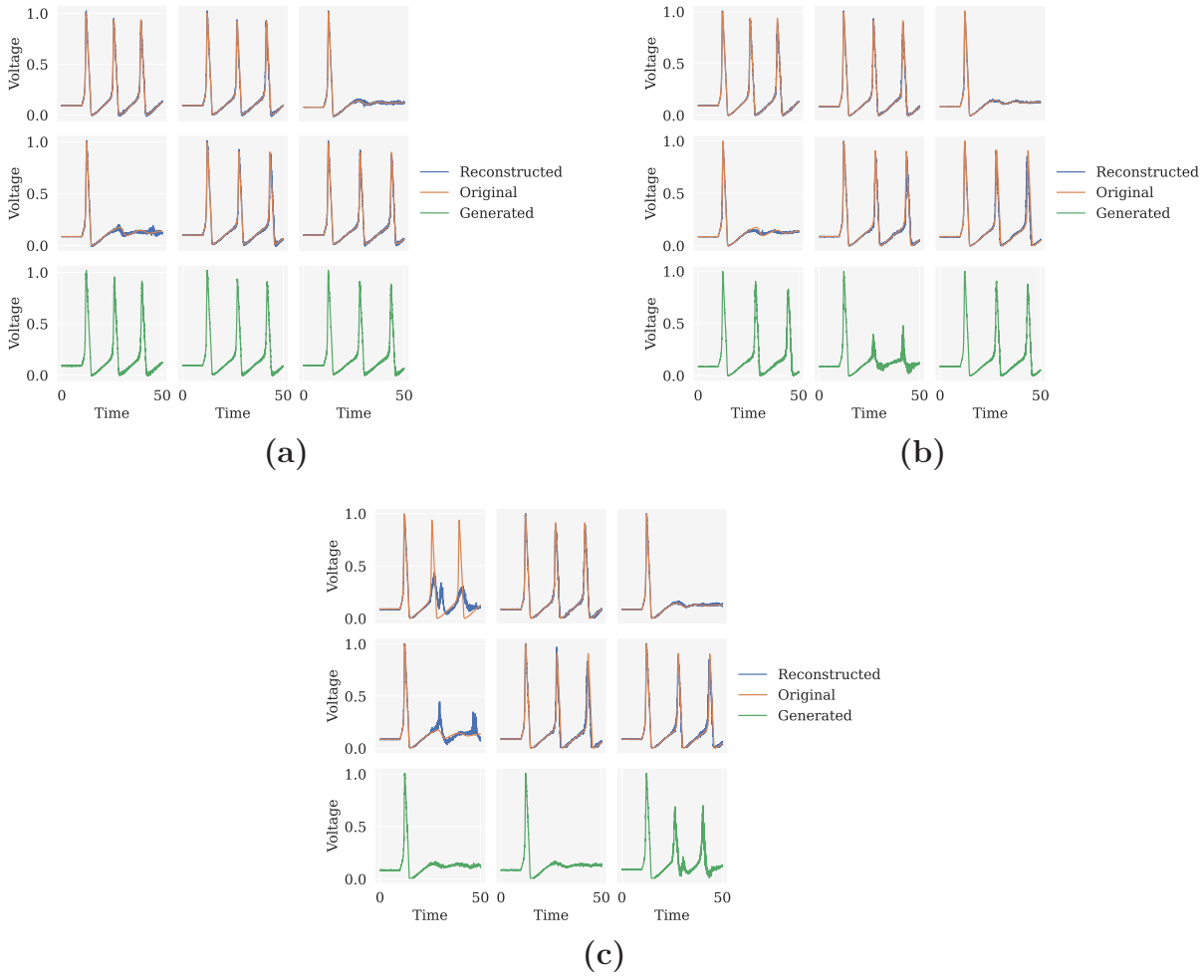

Figure A.4: *Reconstructed voltage traces overlayed with the original signals for a convolutional β-VAE with $\beta = 2$, together with novel voltage traces generated from the latent space. The latent dimensions are (a) 2, (b), 10, (c) 20.*