Machine Learning and Differential Equations

University of Bath

Darrah Kirkpatrick

April 20, 2021

Abstract

In this report we aim to teach the theory that is required to understand Generative Latent Time-series Models and its applications to make make time-series predictions. This model was first introduced in the Neural Ordinary Differential Equation paper [1], which developed a link between deep neural networks and (discretised) ordinary differential equations. The Neural Ordinary Differential Equation approach showed how theory from optimal control can be used to compute learning gradients of our model by solving a single integral. This approach demonstrated the ability to compute learning gradients quickly, whilst allowing the problem to scale linearly with size, representing a substantial advance in Deep Learning literature.

Contents

1 Introduction

In this report, we will explore the theory required to understand generative latent time-series modelling and its applications to make time-series predictions. To achieve this, we will split the report into four main parts:

1. Background Theory
2. Variational Autoencoders
3. Neural Ordinary Differential Equations
4. Generative Latent Time-series Models

Where the first three parts will introduce the theory required to understand generative latent time-series models. With the final part, putting it all together and showing its application to a make time-series predictions.

2 Statistics

Whilst this paper will not focus heavily on the statistical side of machine learning, it is important that we recap some key statistical theory as it will provide motivation and deepen our understanding to what we discuss later.

2.1 Marginal Likelihood

In this section we will introduce some important machinery that will help us quantify how "good" a model is at modelling our data. This can extremely useful as fitting a model to our data can help us understand the underlying mechanisms at play which will help aid in future predictions.

We start off by framing the problem that we would like to solve. Suppose that we have the dataset $\boldsymbol{X} = (\boldsymbol{x}^{(i)})_{i=1}^{N}$ with $\boldsymbol{x}^{(i)} \in \mathbb{R}^n$, and we want to find a parametric model $p_\theta(\cdot)$, that closely fits our data $\boldsymbol{X}$. As our parametric model depends on the value of θ, we want to find the value of θ that bests fits $p_\theta(\cdot)$ to our data.

We will demonstrate this idea with the following small example. Suppose we knew our data $\hat{X} = \{\hat{x}^{(i)}\}_{i=1}^{N}$ with $\hat{x}^{(i)} \in \mathbb{R}$, was a normally distributed with variance 0.1 and an unknown mean. Hence by using the PDF of the normal distribution $\mathcal{N}(\theta, 0.1)$, as our parametric model $p_\theta(\cdot)$. We aim to find the value of best value for θ (which we will refer to as θ^*) such that $p_\theta(\cdot)$ models our data. We will do this by minimising the difference between our model and the data (i.e, $\theta^* = \text{argmin}_\theta \sum_{i=1}^{N} ||p_\theta(x) - \hat{x}^{(i)}||^2$, for $x \in \mathbb{R}$).

We will now formally define how "good" our parametric model is at fitting our data, by how likely it is to see our data given our model,

Definition 2.1 (Likelihood function [4])**.** Given a sample $\boldsymbol{x}^{(i)}$ from our data set $\boldsymbol{X}$, and a parametric model $p_\theta(\cdot)$, the likelihood function L is defined as,

$$L(\theta, \boldsymbol{x}^{(i)}) = p_\theta(\boldsymbol{x}^{(i)}) \tag{1}$$

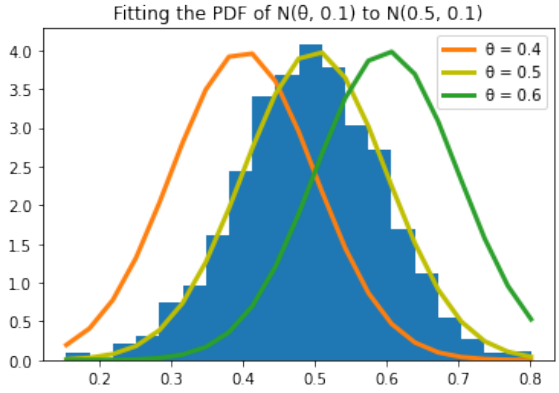


Figure 1: Fitting the PDF of $\mathcal{N}(\theta, 0.1)$ to 1000 samples of $\mathcal{N}(0.5, 0.1)$ with different values of θ (i.e $\theta = 0.4, 0.5, 0.6$). [3]

Therefore, given the sample $\boldsymbol{x}^{(i)}$, the likelihood function, $L(\theta, \boldsymbol{x}^{(i)})$, tells us how likely we would be to obtain that sample given the model parameter θ.

Moreover, by assuming that our dataset $\boldsymbol{X}$ is made up of N independent and identically distributed random variables. We define the joint likelihood function as the following,

$$L(\theta, \boldsymbol{X}) = \prod_{i=1}^{N} p_\theta(\boldsymbol{x}^{(i)}) \tag{2}$$

Now that we have the machinery to measure how likely it is to observe our data given the model parameter θ. We want to find the value of θ that maximises the likelihood of observing our data.

Definition 2.2. (Maximum likelihood estimator [4]) Given the data set $\boldsymbol{X}$, and parametric model $p_\theta(\cdot)$. The maximum likelihood estimate (MLE) is a value of θ^* which maximises the likelihood function $L(\theta, \boldsymbol{X})$,

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, L(\theta, \boldsymbol{X}) = \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{N} p_\theta(\boldsymbol{x}^{(i)})$$

This definitions produces the same outcome as our small example above, as we are maximising the probability of the data occurring given the parametric model (i.e minimising the difference between the parametric model and our data).

Up to now we are have only dealt with models that we are fully-observed as we are able to compute the maximum likelihood estimator directly from our data. Now suppose our model is made up with variables which we can not observe, and which are therefore not included in our dataset. These variables are what we refer to as latent variables, and which we will denote by $\boldsymbol{z}$.

Hence, to compute the likelihood as a function of θ over our observed data, we define the marginal likelihood for the model as the following,

$$p_\theta(\boldsymbol{x}) = \int_{\Omega_{\boldsymbol{z}}} p_\theta(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z} = \int_{\Omega_{\boldsymbol{z}}} p_\theta(\boldsymbol{z}) p_\theta(\boldsymbol{x}|\boldsymbol{z}) d\boldsymbol{z} \tag{3}$$

Where $p_\theta(\boldsymbol{x}, \boldsymbol{z})$ is the joint distribution over the observed and latent variables, and $\Omega_{\boldsymbol{z}}$ is the domain of $\boldsymbol{z}$.

We can think of the marginal likelihood as "getting rid of what is unknown". Thus, given $p_\theta(\boldsymbol{x}, \boldsymbol{z})$, in principle we can work out maximum likelihood estimator θ^*, as $p_\theta(\boldsymbol{x})$ only contains variables which are observed. We will see later in Section 4.2.3 that the integral in (3) is sometimes intractable. Hence, we can not find the maximum likelihood parameter θ^* directly. Therefore, we require some cleave tricks to overcome this limitation.

2.2 Kullback–Leibler Divergence

The Kullback–Leibler Divergence D_{KL}, measures how much two probability distributions differ from each-other and is defined as the following,

Definition 2.3 (Kullback–Leibler Divergence)**.** Let p and q be two probability distributions densities, then the Kullback–Leibler Divergence D_{KL} is defined by,

$$D_{\mathrm{KL}}(p||q) = \int_{\Omega_{\boldsymbol{x}}} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx \tag{4}$$

An important remark is that by Gibbs' inequality (Appendix 1), we have $D_{\mathrm{KL}} \geq 0$ and equality iff $p \equiv q$.

For our purposes we will think of the quantity $D_{\mathrm{KL}}(p||q)$, as a measure of how well the probability distribution p approximates q. This will be an important tool we will use in section 4.2.4, as we are unable to observe a distribution directly, so we require an approximation and hence we want to minimise difference between them.

2.2.1 Important Example

To illustrate this definition, we will compute the KL divergence of two Gaussians. This calculations will be important in Section 4.2.4, as we need to know the negative Kullback–Leibler Divergence between the two multivariate Gaussian distributions, $q_{\boldsymbol{\theta}}(\cdot) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2\boldsymbol{I})$ and, $p(\cdot) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ with dimension m.

To achieve this we firstly compute the two integrals,

$$\begin{aligned}\int q_{\boldsymbol{\theta}}(\boldsymbol{z}) \log p(\boldsymbol{z}) d\boldsymbol{z} &= \int \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2\boldsymbol{I}) \log \mathcal{N}(\boldsymbol{z}; \boldsymbol{0}, \boldsymbol{I}) d\boldsymbol{z} \\ &= -\frac{m}{2}\log(2\pi) - \frac{1}{2}\sum_{j=1}^{m}(\boldsymbol{\mu}_j^2 + \boldsymbol{\sigma}_j^2)\end{aligned}$$

and,

$$\begin{aligned}\int q_{\boldsymbol{\theta}}(\boldsymbol{z}) \log q_{\boldsymbol{\theta}}(\boldsymbol{z}) d\boldsymbol{z} &= \int \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2\boldsymbol{I}) \log \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2\boldsymbol{I}) d\boldsymbol{z} \\ &= -\frac{m}{2}\log(2\pi) - \frac{1}{2}\sum_{j=1}^{m}(1 + \log \boldsymbol{\sigma}_j^2)\end{aligned}$$

Therefore, we are able to combine these results and get the following,

$$\begin{aligned} -D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\cdot)|||p_{\boldsymbol{\theta}}(\cdot)) &= -\int q_{\boldsymbol{\phi}}(\boldsymbol{z}) \log\left(\frac{q_{\boldsymbol{\phi}}(\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{z})}\right) d\boldsymbol{z} \\ &= \int q_{\boldsymbol{\phi}}(\boldsymbol{z})(\log p_{\boldsymbol{\theta}}(\boldsymbol{z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z}))d\boldsymbol{z} \\ &= \frac{1}{2}\sum_{j=1}^{m}(1 + \log(\boldsymbol{\sigma}_j^2) - \boldsymbol{\mu}_j^2 - \boldsymbol{\sigma}_j^2) \end{aligned}$$

3 Machine Learning Basics

Throughout this section we will introduce the basic tools for solving modelling and prediction problems. However, we first must state some terminology that we will use throughout this section. Suppose that we were given the sets $\boldsymbol{X}$ and $\boldsymbol{Y}$, and we want to predict $\boldsymbol{Y}$ given $\boldsymbol{X}$, then we term $\boldsymbol{X}$ is our data and $\boldsymbol{Y}$ our observations respectively. The sets $\boldsymbol{X}$ and $\boldsymbol{Y}$ can take many of forms ranging from $\boldsymbol{X}$ being the set of x-rays scans of the arm and $\boldsymbol{Y}$ corresponding whether there is a fractured bone. To more simpler sets, $\boldsymbol{X}$ being a set of real numbers and $\boldsymbol{Y}$ corresponding to evaluations of an unknown linear function at each point of $\boldsymbol{X}$.

3.1 Parametric models

As we described in the previous chapter, a parametric model is any model that is parameterised. These parameters are quantity that we can control and are used to describe what "features we can tune" in our model. This ability allows us to produce different model dynamics that we can use to find the parameters that best describes our data.

It is important to note that we have left the definition of a parametric model very open and loosely defined. This is on purpose as throughout this paper a parametric model may take a form of probability distributions, neural networks or functions.

We demonstrate the definition of a parametric model, by providing the following example. Suppose that our parametric model takes the form of,

$$f_{\boldsymbol{\theta}} : \mathbb{R} \to \mathbb{R}, \quad f_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x + \theta_2 x^2, \quad \text{with } \boldsymbol{\theta} \in \mathbb{R}^3 \tag{5}$$

We are able to see from Figure. (2), that the choice of the values of $\theta_0, \theta_1, \theta_2$, we are able to drastically change the dynamics of the model $f_{\boldsymbol{\theta}}$.

3.2 Loss Function

A loss function is a way of describing how well our parametric model preforms on given data point and is defined as the following,

$$L(\cdot, \cdot) : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$$

Where the inputs of the loss function is our parametric model, $p_{\boldsymbol{\theta}}$ evaluated at the data point $\boldsymbol{x}^{(i)} \in \boldsymbol{X}$ and the corresponding observation, $\boldsymbol{y}^{(i)} \in \boldsymbol{Y}$. We are able

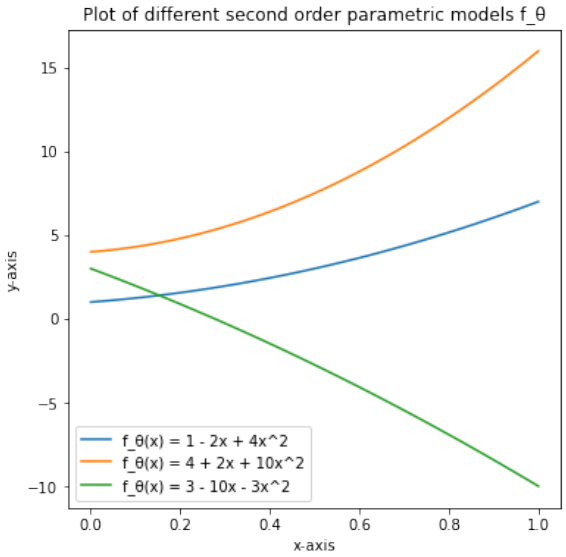


Figure 2: A plot of 3 different dynamics of the parametric models $f_{\boldsymbol{\theta}}$, demonstrating how changing the value of $\boldsymbol{\theta}$ can produce different dynamics. (blue) $f_{\boldsymbol{\theta}}(x) = 1 - 2x + 4x^2$. (orange) $f_{\boldsymbol{\theta}}(x) = 4 + 2x + 10x^2$. (green) $f_{\boldsymbol{\theta}}(x) = 3 - 10x + 3x^2$.

to think of the output of the loss function as a measure of how much information is lost by our parametric model given the data point $\boldsymbol{x}^{(i)}$.

To demonstrate this point, suppose that our data came from the function, $f(x) = 2x + 1$, and our parametric model (5), takes the form $f_{\boldsymbol{\theta}}(x) = 2x + 4$. By choosing the loss function to be the squared error between the true observation and our parametric model prediction (i.e $L(f(x), f_{\boldsymbol{\theta}}) = ||f(x) - f_{\boldsymbol{\theta}}(x)||^2$), we can see that our model loses is 9 units of information given a data point, $x \in \mathbb{R}$.

As we alluded to above, suppose that our dataset consists of multiply data points, $\boldsymbol{X} = \{\boldsymbol{x}^{(i)}\}_{i=1}^N$ and $\boldsymbol{Y} = \{\boldsymbol{y}^{(i)}\}_{i=1}^N$, respectively. We define the total loss of our model given $\boldsymbol{X}$ and $\boldsymbol{Y}$, as the following,

$$E(\boldsymbol{\theta}) = \sum_{i=1}^{N} L(\boldsymbol{y}^{(i)}, p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})) \tag{6}$$

Where $p_{\boldsymbol{\theta}}$ is a parametric model, parameterized by $\boldsymbol{\theta}$. This quantity allows us to understand how our model performance on the total data set, as our model might model well certain observations and poorly on others.

3.3 Gradient Descent

Now that we have the total loss function $E(\boldsymbol{\theta})$, for the parametric model $p_{\boldsymbol{\theta}}$, we want to find the value of $\boldsymbol{\theta}$ that minimises $E(\boldsymbol{\theta})$ (i.e $\theta^* = \text{argmin}_{\boldsymbol{\theta}} E(\boldsymbol{\theta})$). We do this by introducing an interactive approach to find the local mimima of $E(\boldsymbol{\theta})$. We first start by initialising the parameter $\boldsymbol{\theta}_0$ to a random values and follow the procedure,

$$\begin{aligned} \boldsymbol{v}_n &= \eta_n \nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta}) \\ \boldsymbol{\theta}_{n+1} &= \boldsymbol{\theta}_n - \boldsymbol{v}_n \end{aligned}$$

Where $\nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta})$ is the derivative of $E(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$ and η_n is the learning rate at step n. The learning rate η_n, describes how big of a step in the direction $-\nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta})$ that we will take. We take steps in the direction of $-\nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta})$ as $\nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta})$ give us the direction of maximal increase of $E(\boldsymbol{\theta})$; Therefore by going in inverse direction, we will go towards the minimum of $E(\boldsymbol{\theta})$. It is important that we get the value of η_n correct, as if it is too small the algorithm will take too long and if we take too big of a step it will over shoot the minimum. This effect is demonstrated in Figure. 3, as choosing different values of the learning rate can cause various effects when finding the minima. This producure is normally terminated when the values $\boldsymbol{\theta}_{n+1}$ and $\boldsymbol{\theta}_n$ differ within a tolerance that we choose (i.e $||\boldsymbol{\theta}_{n+1} - \boldsymbol{\theta}_n||^2 < \varepsilon$ with $\varepsilon \ll 1$).

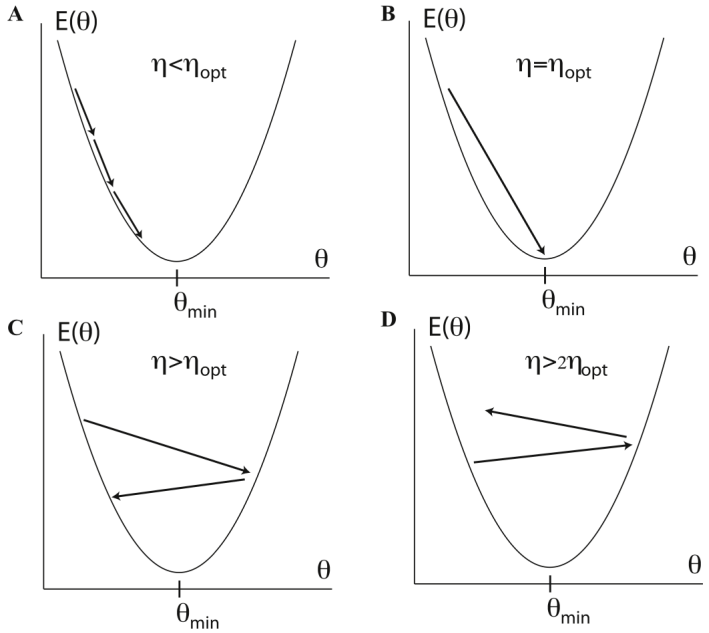


Figure 3: Effect of learning rate on convergence. For a one dimensional quadratic potential, one can show that there exists four different qualitative behaviours for gradient descent (GD) as a function of the learning rate η depending on the relationship between η and $\eta_{\text{opt}} = [\partial^2_{\boldsymbol{\theta}} E(\boldsymbol{\theta})]^{-1}$. (a) For $\eta < \eta_{\text{opt}}$, GD converges to the minimum. (b) For $\eta = \eta_{\text{opt}}$, GD converges in a single step. (c) For $\eta_{\text{opt}} < \eta < 2\eta_{\text{opt}}$, GD oscillates around the minima and eventually converges. (d) For $\eta > 2\eta_{\text{opt}}$, GD moves away from the minima. ([5], p.15)

3.4 Training

Up to now we have only dealt with fitting a model to a given data set. However one of the main advantages of modelling data comes from its ability to model data that we have not seen before. We formalise this notion by firstly splitting our data into two sets, $\boldsymbol{X}_{\text{train}}$ and $\boldsymbol{X}_{\text{test}}$, with corresponding $\boldsymbol{Y}_{\text{train}}$ and $\boldsymbol{Y}_{\text{test}}$ respectively. This splitting allows us to train our model using $\boldsymbol{X}_{\text{train}}$ and $\boldsymbol{Y}_{\text{train}}$ then test the performance of our model on data that it has has not seen before (i.e $\boldsymbol{X}_{\text{test}}$ and $\boldsymbol{Y}_{\text{test}}$). We measure this performance by defining the in-sample error $E_{\text{in}} = E(\boldsymbol{\theta}|\boldsymbol{X}_{\text{train}}, \boldsymbol{Y}_{\text{train}})$ and out-of-sample error $E_{\text{out}} = E(\boldsymbol{\theta}|\boldsymbol{X}_{\text{test}}, \boldsymbol{Y}_{\text{test}})$.

The out-of-sample error E_{out}, will naturally decrease as the number of data points increase. This due to there being less sampling noise that appears in the data, and our model tending to its true distribution. As the model tends to its true distribution from where the data is drawn, the in-sample and out-of-sample error must therefore converge to the same value; This is termed the "bias" of our model. The bias tells us how good of a model we have if we had infinite data point. However, since in piratical application we only have a finite amount of data points, it is better to minimise the out-of-sample error E_{out} rather than the bias; As this will lead to a model which has better predicting abilities. This is due to the fact that the difference between E_{in} and E_{out} is the same as fitting and predicting new data. If the difference between the in-sample error E_{in}, is much smaller than out-of-sample error E_{out}, we know that we over-fitted our model, as performs poorly on new data.

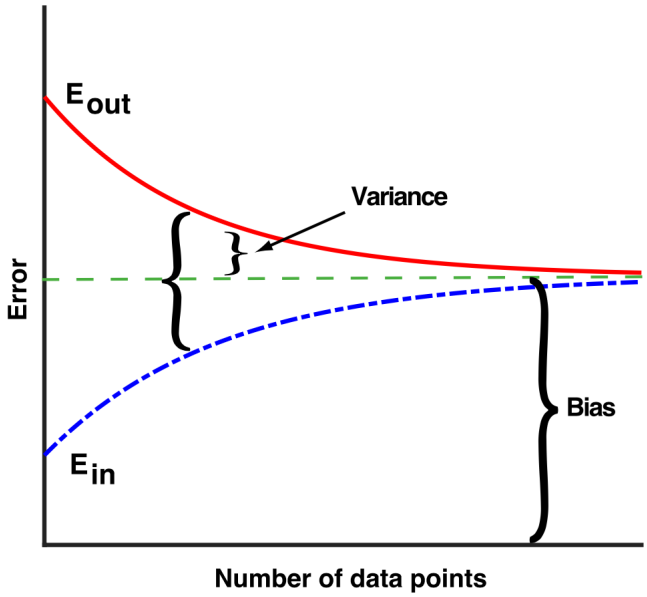


Figure 4: Schematic of typical in-sample and out-of-sample error as a function of training set size. The typical in-sample or training error, E_{in}, out-of-sample or generalisation error, E_{out}, bias, variance, and difference of errors as a function of the number of training data points. ([5], p.11)

Figure. 5, shows that the out-of-sample E_{out} is a function of "model complexity", which is how many parameters the model has. This supports the idea of over-fitting that we discussed above. As we expect increasing the number of parameters in our model we over-fit our model to the training data and hence, perform poorly on the at predicting new data. This is due to there being a finite amount of data points, even though increasing the model complexity will lead to a smaller bias due to a smaller in-sample error. Therefore it is a more favourable to choose a model with smaller variance than a less-biased model with large variance.

Finding the minima of a function is not inherently unique to this problem. Therefore, gradient decent with not the only algorithm that we can use to find the minima, with each algorithm offering their own take and limitations. To demonstrate this

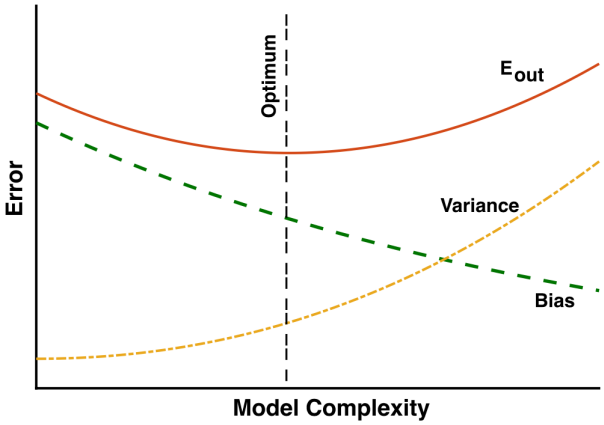


Figure 5: Bias–Variance tradeoff and model complexity. This schematic shows the typical out-of-sample error E_{out} as function of the model complexity for a training dataset of fixed size. ([5], p.12)

we will introduce RMSprop iteration, that is a form of adaptive learning method,

$$\begin{aligned} \boldsymbol{g}_t &= \nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta}) \\ \boldsymbol{s}_t &= \beta \boldsymbol{s}_{t-1} + (1-\beta)\boldsymbol{g}_t^2 \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \eta_t \frac{\boldsymbol{g}_t}{\sqrt{\boldsymbol{s}_t + \varepsilon}} \end{aligned}$$

Where β controls the averaging time of the second moment, η_t is the learning rate, and ε is a small regularisation constant.

Another way that we might control over-fitting is the use of epochs and batches. Epochs are the number of times in the we use the whole training data $\boldsymbol{X}_{\text{train}}$ and $\boldsymbol{Y}_{\text{train}}$ in training. Whilst batches split up the training data $\boldsymbol{X}_{\text{train}}$ into smaller training data sets that we minimise the total cost over.

To demonstrate this idea suppose we we have a training data set consisting of 100 data points, and apply five epochs of batch sizes 25. Therefore we use the total data set five times, and each round of epoch contrasting of four evaluations of the training algorithm on the data set,

$$E^i_{\text{in}} = E(\boldsymbol{\theta}|\boldsymbol{X}^i_{\text{train}}, \boldsymbol{Y}^i_{\text{train}}), \quad \boldsymbol{X}^i_{\text{train}} = \sum_{j=25i+1}^{25(i+1)} \boldsymbol{x}^{(j)}, \quad \boldsymbol{Y}^i_{\text{train}} = \sum_{j=25i+1}^{25(i+1)} \boldsymbol{y}^{(j)}, \quad i \in \{0,1,2,3\}$$

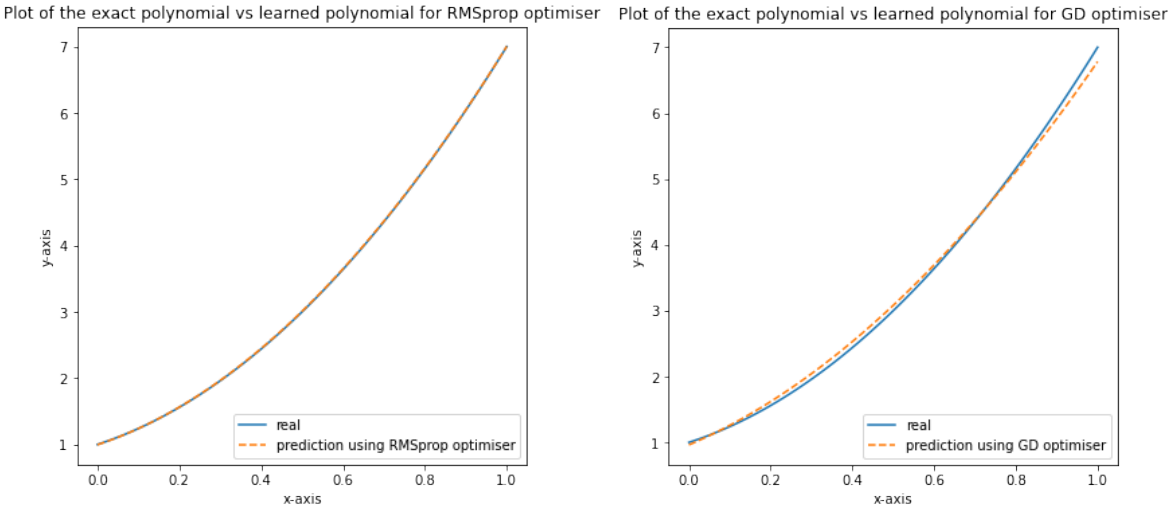
3.5 Theory Example

We will now combine the theory that we have introduced into a small example to demonstrate how each part plays a key role of solving a modelling problem.

Suppose that our data looked like it came from a quadratic distribution. We test this hypothesis by using the following quadratic parametric model, $f_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x + \theta_2 x^2$, and then trying to learn the value of $\boldsymbol{\theta}$ that best fits $f_{\boldsymbol{\theta}}$ to our

data. Finally we choose the squared error loss function to measure this difference and minimise over

$$\begin{aligned}
\boldsymbol{\theta}^* &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, E(\boldsymbol{\theta}) \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{N} L(y^{(i)}, f_{\boldsymbol{\theta}}(x^{(i)})) \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{N} ||y^{(i)} - f_{\boldsymbol{\theta}}(x^{(i)})||^2 \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{N} ||y^{(i)} - (\theta_0 + \theta_1 x^{(i)} + \theta_2 (x^{(i)})^2)||^2
\end{aligned}$$



(a) (b)

Figure 6: Plot of the exact polynomial $f(x) = 1+2x+4x^2$ vs learned polynomial for GD and RMSprop optimiser over the interval $[0, 1]$ trained using 19200 data points, learning rates of 0.01, 4 epochs, and batch-sizes of 64. (blue) exact polynomial (orange) learned polynomial (a) Plot of the exact vs learned polynomial $f_{\boldsymbol{\theta}}(x) = 0.996 + 2.045x + 3.963x^2$ for the RMSprop optimiser. (b) Plot of the exact vs learned polynomial $f_{\boldsymbol{\theta}}(x) = 0.962 + 2.656x + 3.163x^2$ for the GD optimiser. [3]

Figure. 6, shows us that RMSprop performs well at approximating the true values of $\boldsymbol{\theta}$ whilst GD struggles slightly. This further emphasise how challenging modelling data can be, as choosing the parameters such as learning algorithm, number of data points, learning rate, epochs and batch-sizes, has to be done on a case to case basis.

3.6 Neural Networks Architecture

The deep neural network architecture is defined as the following,

$$\boldsymbol{a}^l = \boldsymbol{\sigma}(\boldsymbol{W}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l) =: \boldsymbol{\sigma}(\boldsymbol{z}^l), \qquad l \in \{1, \dots, L\}, \qquad L \in \mathbb{N} \tag{7}$$

$$n_l \in \mathbb{N}, \qquad \boldsymbol{a}^l \in \mathbb{R}^{n_l}, \qquad l \in \{0, \dots, L\} \tag{8}$$

$$\boldsymbol{W}^l \in \mathbb{R}^{n_l \times n_{l-1}}, \qquad \boldsymbol{b}^l \in \mathbb{R}^{n_l}, \qquad l \in \{1, \dots, L\} \tag{9}$$

We refer to $\boldsymbol{a}^l$ as an activation layer. This is because this quantity represents how much "activation" each neuron/node gets when it is fed into the next layer. $\boldsymbol{W}^l$ is the weight matrix, as it represents how much of weight each neuron of the previous activation layer has on the next layer (i.e by the matrix-multiplication $\boldsymbol{W}^l \boldsymbol{a}^{l-1}$). $\boldsymbol{b}^l$ is the bias (or bias vector) as it shifts the each value of $\boldsymbol{W}^l \boldsymbol{a}^{l-1}$ either to the left (upwards) or right (downwards). Finally, $\boldsymbol{\sigma}(\cdot)$ is refereed to as the activation function, as it output how much "activation" is carried on to the next layer.

Since, $\boldsymbol{W}^l$ and $\boldsymbol{b}^l$ are not explicated defined, a neural network can be though of as a parametric model. Therefore we are apply all the theory we have just introduced to them. This is not the only way we can define neural network, as they can come in many of forms such as, Recurrent Neural Networks [6] or Convolutional Neural Networks [7].

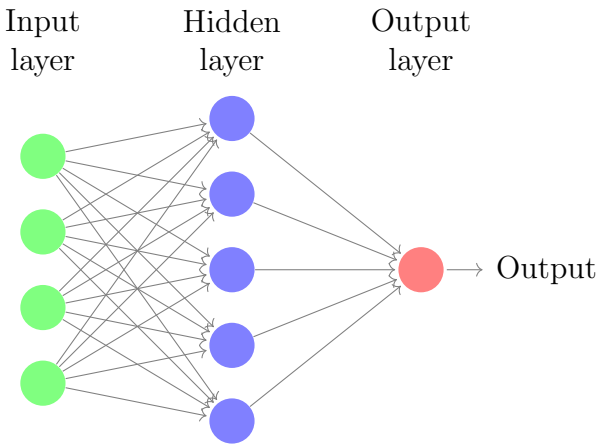


Figure 7: A deep neural network with an input dimension of 4 ($n_0 = 4$) , a hidden state of dimension of 5 ($n_1 = 5$), and an output dimension of 1 ($n_2 = 1$).

3.7 Backpropagation Algorithm

The backpropagation algorithm for a deep neural network allows us take gradients with respect to the weights and biases to find the minima of our loss function, C. Since the loss function depends on the output layer of our neural network, we define the error at the j-th neuron of the output layer, as the following,

$$\boldsymbol{\Delta}_j^L = \frac{\partial C}{\partial \boldsymbol{z}_j^L} \tag{10}$$

Where L denotes the output layer of our neural network, defined in (7). Moreover as the error of our output indirectly depends on the previous layers of our network. We can generalise (10), to get the error at the j-th neuron on the l-th layer, $\boldsymbol{\Delta}_j^l$, as the change in the loss function with respect to $\boldsymbol{z}_j^l$.

$$\boldsymbol{\Delta}_j^l = \frac{\partial C}{\partial \boldsymbol{z}_j^l} = \frac{\partial C}{\partial \boldsymbol{a}_j^l}\frac{\partial \boldsymbol{a}_j^l}{\partial \boldsymbol{z}_j^l} = \frac{\partial C}{\partial \boldsymbol{a}_j^l}\boldsymbol{\sigma}'(\boldsymbol{z}_j^l) \tag{11}$$

Where $\boldsymbol{\sigma}'(\boldsymbol{z}_j^l)$ is the derivative of the activation function $\boldsymbol{\sigma}(\cdot)$, evaluated at $\boldsymbol{z}_j^l$. It is important to note that the dimension of the activation function's derivative may change per layer.

We can also express (11) in terms of the partial derivative of the loss function with respect to the bias, by noting that $\partial \boldsymbol{b}_j^l / \partial \boldsymbol{z}_j^l = \mathbf{1}$,

$$\boldsymbol{\Delta}_j^l = \frac{\partial C}{\partial \boldsymbol{z}_j^l} = \frac{\partial C}{\partial \boldsymbol{b}_j^l}\frac{\partial \boldsymbol{b}_j^l}{\partial \boldsymbol{z}_j^l} = \frac{\partial C}{\partial \boldsymbol{b}_j^l} \tag{12}$$

Hence by using the chain rule again, we are able to express the error at the l-th layer in terms of the $l+1$-th layer. This is because error can be through of as being carried forward through the neural network.

$$\boldsymbol{\Delta}_j^l = \frac{\partial C}{\partial \boldsymbol{z}_j^l} = \sum_k \frac{\partial C}{\partial \boldsymbol{z}_k^{l+1}}\frac{\partial \boldsymbol{z}_k^{l+1}}{\partial \boldsymbol{z}_j^l} \tag{13}$$

$$= \sum_k \boldsymbol{\Delta}_l^{k+1}\frac{\partial \boldsymbol{z}_k^{l+1}}{\partial \boldsymbol{z}_j^l} = \left(\sum_k \boldsymbol{\Delta}_k^{l+1}\boldsymbol{W}_{kj}^{l+1}\right)\boldsymbol{\sigma}(\boldsymbol{z}_j^l) \tag{14}$$

Finally, we derive the derivative of the loss function with respect to the weight $\boldsymbol{W}_{jk}^l$,

$$\frac{\partial C}{\partial \boldsymbol{W}_{jk}^l} = \frac{\partial C}{\partial \boldsymbol{z}_j^l}\frac{\partial \boldsymbol{z}_j^l}{\partial \boldsymbol{W}_{jk}^l} = \boldsymbol{\Delta}_j^l \boldsymbol{a}_k^{l-1} \tag{15}$$

Algorithm 1 Backpropagation algorithm ([5] p.55)

1: **Feedforward**: Start with the first layer, exploit the feed-forward architecture through (7) to compute $\boldsymbol{z}^l$ and $\boldsymbol{a}^l$ for each subsequent layer.
2: **Error at output layer**: Calculate the error of the output layer using (11).
3: **"Backpropagate" the error**: We now use (14) to propagate the error backwards and calculate $\boldsymbol{\Delta}_j^l$ for all the layers.
4: **Calculate gradient**: Now use (12) and (15) to calculate $\frac{\partial C}{\partial \boldsymbol{b}_j^l}$ and $\frac{\partial C}{\partial \boldsymbol{W}_{jk}^l}$.

3.8 Generative Modelling

Generative modelling is a way that we can produce new data. It does this by learning the joint probability $p(\boldsymbol{X}, \boldsymbol{Y})$ or just $p(\boldsymbol{X})$ if there are no observations $\boldsymbol{Y}$ given. Therefore by sampling from its learned distribution, we are also produce samples would of likely came from our test data $\boldsymbol{X}$.

Figure 8: Samples from a generative model trained on Frey's face. [8]

4 Variational Autoencoders

4.1 Autoencoder

Before we understand variational autoencoders (VAEs), we firstly must understand their primitive cousin, autoencoders. Autoencoders encapsulate the idea that data can be expressed by a small number of key features, whilst the rest is noise. For example, if you were to communicate the key ideas of a face to someone, the main features would be the position and shape of the eyes, mouth, and nose, whilst the rest of the details like facial hair and piercings are noise.

This feature selection process is expressed formally by performing dimensional reduction to the input data. This can be done in various ways and we will refer to this action as the Encoder. We will also term the output of the Encoder the feature/latent space, as it expresses the key features that make up our input data. The autoencoder then projects the output of the Encoder back to the space containing the data, and we will refer to this operation as the Decoder. As we have assumed that our data can be represented by a small number of features, the aim is to minimise the difference between our input and reconstructed data. We will use the squared error loss function to quantify how accurate our autoencoder is at reconstructing the input data. To summarise, we have the following model,

$$\text{(Encoder) } \phi_{\mathrm{E}} : \mathbb{R}^n \to \mathbb{R}^m, \quad \text{(Decoder) } \phi_{\mathrm{D}} : \mathbb{R}^m \to \mathbb{R}^n, \quad m \ll n, \ m, n \in \mathbb{N}$$

$$\text{(L2 loss function) } L = ||\boldsymbol{x} - \hat{\boldsymbol{x}}||^2 = ||\boldsymbol{x} - \phi_{\mathrm{D}}(\phi_{\mathrm{E}}(\boldsymbol{x}))||^2$$

As we alluded to earlier, both the Encoder and Decoder can take many forms with the simplest case being linear maps, however in this paper we will only look at them being neural networks. Therefore, the autoencoder can be thought of as a single neural network, as it is just a combination of two smaller neural networks sequentially. By ensuring that $m \ll n$, our architecture creates a bottleneck for the data, allowing the key features to be used for the reconstruction. Hence, by

performing gradient descent, we are able to find the parameters that minimise the loss function for our data.

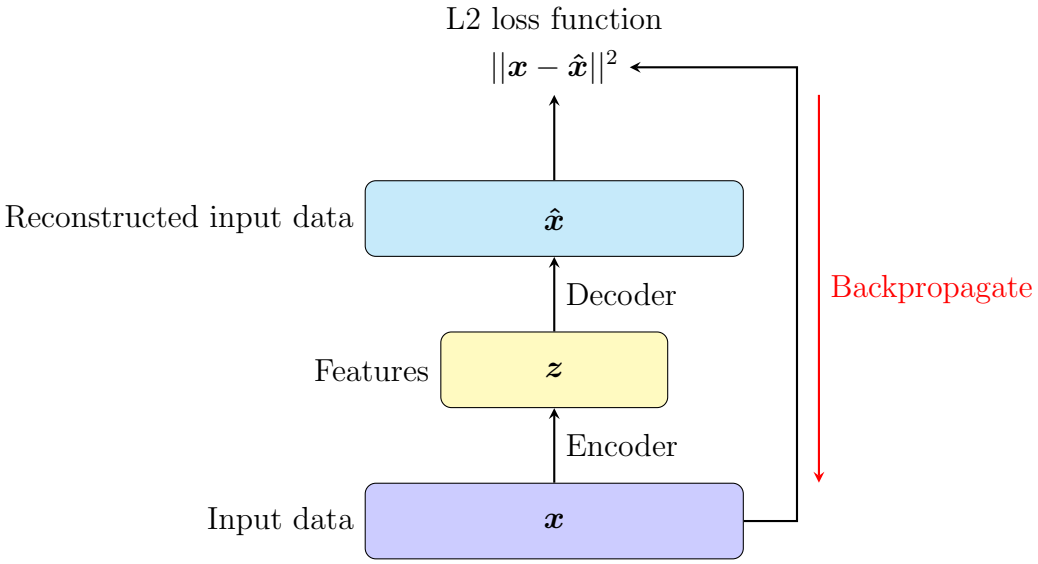


Figure 9: Training model for an autoencoder

However, this approach leads to the limitation that two points close in the feature space, may not necessary corresponds to two similarly points in data space. This renders the autoencoder model useless for generative modelling, as we want points that are close in feature space to be mapped to points close in data space because we expect them to have "similar qualities".

4.2 Variational Autoencoders

4.2.1 Motivation

Variational autoencoders are a probabilistic approach of autoencoders that aims to overcome the irregularity that appear between the feature and data space in autoencoders. This is achieved by mapping our input data to a distribution over the feature space rather than to a single point, we then sample from a decoder distribution to get our reconstructed input data. This procedure allows us to introduce regularisation at the time of training to prevent overfitting and ensures that our feature spaces has good properties for generative modelling.

Putting together the model described above, the variational autoencoder evauled at the point $\boldsymbol{x}$ takes the form:

1. Our input data $\boldsymbol{x}$ is encoded as a distribution $p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})$ in feature space.
2. We then sample the point $\boldsymbol{z}$ from the distribution $p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})$.
3. Then $\boldsymbol{z}$ is decoded into the distribution $p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{z})$.
4. Finally, $\hat{\boldsymbol{x}}$ is then drawn from the decoder distribution $p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{z})$.

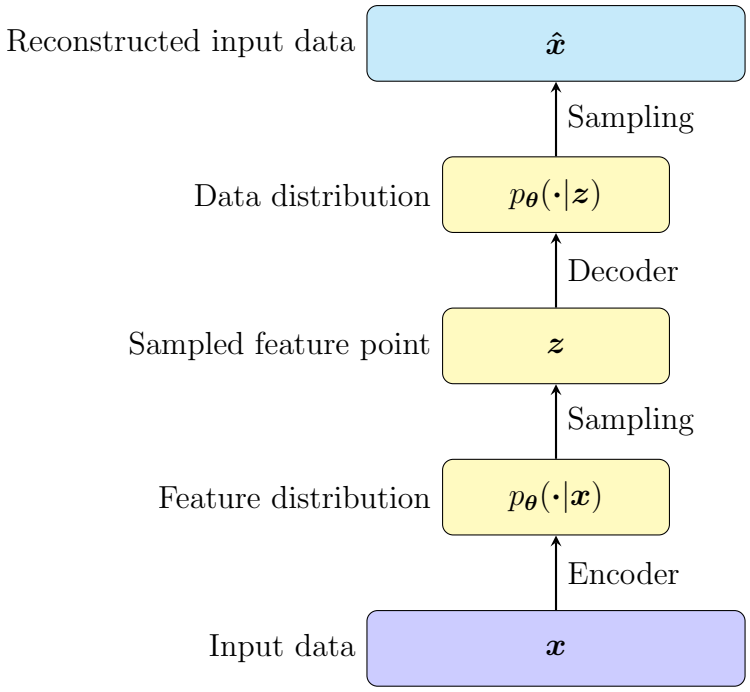


Figure 10: Graphical representation of a variational autoencoder evaluated at the point $\boldsymbol{x}$.

4.2.2 Probabilistic Framework

To support the above claims, we will create a probabilistic framework. Variational autoencoders build off the assumption that our dataset $\boldsymbol{\mathcal{D}} = (\boldsymbol{x}^{(i)})_{i=1}^{N}$, was generated by N independent samples of an unchanging underlying distribution. Where each sample $\boldsymbol{x}^{(i)}$, was generated from some random process that involves the continuous unobserved latent random variable $\boldsymbol{z}^{(i)}$. This process happens in two distinct steps:

1. The value $\boldsymbol{z}^{(i)}$ is generated from some prior distribution $p_{\boldsymbol{\theta}^*}(\boldsymbol{z})$
2. Then $\boldsymbol{x}^{(i)}$ is generated from some conditional distribution $p_{\boldsymbol{\theta}^*}(\boldsymbol{x}|\boldsymbol{z}^{(i)})$

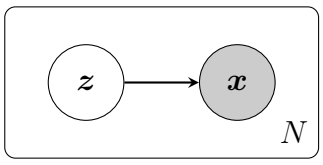


Figure 11: Probabilistic graphical model for our data generation process

Where $\boldsymbol{\theta}^*$, are the true parameters of the parametric families of differentiable distributions, $p_{\boldsymbol{\theta}}(\boldsymbol{z})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$. This in turn allows us to define the joint parametric distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})$, as $p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) = p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})$.

4.2.3 The Problem With This Approach

Therefore by using the the joint parametric distribution above $p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) = p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})$, the marginal likelihood is defined as the following,

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int_{\Omega_{\boldsymbol{z}}} p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) dz = \int_{\Omega_{\boldsymbol{z}}} p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})\, d\boldsymbol{z}$$

So in principle we are able find the optimal parameter $\boldsymbol{\theta}^*$ by applying GD to the likelihood given data set. However, due to integral being intractable as it does not have an analytical solution or an efficient estimator. We are unable take derivative with respect to the parameter $\boldsymbol{\theta}$ and find $\boldsymbol{\theta}^*$ that maximises $p_{\boldsymbol{\theta}}(\boldsymbol{x})$.

The intractability of $p_{\boldsymbol{\theta}}(\boldsymbol{x})$, is related to the intractability of the unobserved posterior distribution $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$, by the relationship,

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) = \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{x})} = \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})}{\int_{\Omega_{\boldsymbol{z}}} p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})\,d\boldsymbol{z}}$$

Therefore, if we have a tractable $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ (or $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ respectably) this will lead to a tractable $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ (or $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ respectably), as $p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})$ is tractable due to it being modelled as the product of two known parametric distribution.

4.2.4 Overcoming This Problem

To overcome this problem we try to approximate the true unobserved posterior distribution $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ by using the encoder model $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, parameterised by $\boldsymbol{\phi}$. We aim to finding the value for $\boldsymbol{\phi}$ such that the distribution $q_{\boldsymbol{\phi}^*}(\boldsymbol{z}|\boldsymbol{x})$, closely approximates the true intractable posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$. This also equivalent to solving the minimisation problem,

$$\boldsymbol{\phi}^* = \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\, D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\cdot|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})). \tag{16}$$

Since we have assumed that each data point $\boldsymbol{x}^{(i)}$ are independent, the total marginal likelihood is the sum of the individual likelihoods. Hence, by using our encoder model, we will compute the following marginal likelihood,

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) &= \mathbb{E}_{\boldsymbol{z}\sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})] \quad (\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) \text{ does not depend on } \boldsymbol{z}) \\
&= \mathbb{E}_{\boldsymbol{z}}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\right] \quad \text{(Bayes' Rule twice)} \\
&= \mathbb{E}_{\boldsymbol{z}}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\right] \quad \text{(Multiplication by 1)} \\
&= \mathbb{E}_{\boldsymbol{z}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})] - \mathbb{E}_{\boldsymbol{z}}\left[\log \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}{p_{\boldsymbol{\theta}}(\boldsymbol{z})}\right] + \mathbb{E}_{\boldsymbol{z}}\left[\log \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\right] \quad \text{(Logs)} \\
&= \underbrace{\mathbb{E}_{\boldsymbol{z}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})] - D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_{\boldsymbol{\theta}}(\boldsymbol{z}))}_{\mathcal{L}(\boldsymbol{x}^{(i)};\boldsymbol{\theta},\boldsymbol{\phi})} + \underbrace{D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}^{(i)}))}_{\geq 0}
\end{aligned}$$

The true intractable posterior $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ still appears in the last term of the marginal likelihood. However, we are able to use the fact that the Kullback–Leibler divergence is non-negative and zero if and only if $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ equals the true posterior, and obtain an evidence lower bound (ELBO), $\mathcal{L}(\boldsymbol{x}^{(i)};\boldsymbol{\theta},\boldsymbol{\phi})$, for our marginal likelihood.

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) \geq \mathcal{L}(\boldsymbol{x}^{(i)};\boldsymbol{\theta},\boldsymbol{\phi}) = \mathbb{E}_{\boldsymbol{z}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})] - D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_{\boldsymbol{\theta}}(\boldsymbol{z}))$$

Hence, we aim to maximise $\mathcal{L}(\boldsymbol{x}^{(i)};\boldsymbol{\theta},\boldsymbol{\phi})$ through gradient descent and obtain the optimal parameters $\boldsymbol{\theta}^*$ and $\boldsymbol{\phi}^*$ that maximises the evidence lower bound.

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) \geq \mathcal{L}(\boldsymbol{x}^{(i)};\boldsymbol{\theta}^*,\boldsymbol{\phi}^*) \geq \mathcal{L}(\boldsymbol{x}^{(i)};\boldsymbol{\theta},\boldsymbol{\phi})$$

We will now look at the terms in $\mathcal{L}(\boldsymbol{x}^{(i)};\boldsymbol{\theta},\boldsymbol{\phi})$, as it will allow us to understand what the ELBO is maximising for:

$\mathbb{E}_{\boldsymbol{z}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})]$, can be thought of as how well the model done at reconstructing the input data. Therefore maximising for $\boldsymbol{\theta}$ will give use a good reconstruction of our input data.

$D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_{\boldsymbol{\theta}}(\boldsymbol{z}))$, can be though as regularising $\boldsymbol{\phi}$ such that $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})$ approximate the distribution $p_{\boldsymbol{\theta}}(\boldsymbol{z})$. It is not entirely obvious that this will yield the correct $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})$ that we set out to obtain; However, if we fix $\boldsymbol{\theta}$ and think about finding the value of $\boldsymbol{\phi}$ that minimises the KL-divergence between $q_{\boldsymbol{\phi}}(\cdot|\boldsymbol{x})$ and $p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})$, we get the following,

$$
\begin{aligned}
\boldsymbol{\phi}^* &= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left(D_{KL}(q_{\boldsymbol{\phi}}(\cdot|\boldsymbol{x}^{(i)})||p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x}^{(i)}))\right) \text{ (Using the notation } \mathbb{E}_z(\cdot) \equiv \mathbb{E}_{\boldsymbol{z}\sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}(\cdot)) \\
&= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left(\mathbb{E}_{\boldsymbol{z}}[\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})] - \mathbb{E}_{\boldsymbol{z}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})]\right) \quad \text{(Logs)} \\
&= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left(\mathbb{E}_{\boldsymbol{z}}[\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})] - \mathbb{E}_{\boldsymbol{z}}\left[\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})}\right]\right) \quad \text{(Bayes' Rule twice)} \\
&= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left(\mathbb{E}_{\boldsymbol{z}}[\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})] - \mathbb{E}_{\boldsymbol{z}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{z})] - \mathbb{E}_{\boldsymbol{z}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})] + \mathbb{E}_{\boldsymbol{z}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})]\right) \text{ (Logs)}
\end{aligned}
$$

Since, $\mathbb{E}_{\boldsymbol{z}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})]$ is independent of $\boldsymbol{\phi}$, we can remove it without effecting the argmin.

$$
\begin{aligned}
&= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left(\mathbb{E}_{\boldsymbol{z}}[\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})] - \mathbb{E}_{\boldsymbol{z}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{z})] - \mathbb{E}_{\boldsymbol{z}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})]\right) \\
&= \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \left(-\mathbb{E}_{\boldsymbol{z}}[\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})] + \mathbb{E}_{\boldsymbol{z}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{z})] + \mathbb{E}_{\boldsymbol{z}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})]\right) \text{ (argmin to argmax)} \\
&= \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \left(\mathbb{E}_{\boldsymbol{z}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})] - D_{KL}(q_{\boldsymbol{\phi}}(\cdot|\boldsymbol{x}^{(i)})||p_{\boldsymbol{\theta}}(\cdot))\right) \quad (D_{\text{KL}} \text{ definition})
\end{aligned}
$$

Hence, we arrive at the same expression as $\mathcal{L}(\boldsymbol{x}^{(i)};\boldsymbol{\theta},\boldsymbol{\phi})$. Therefore, maximising $\mathcal{L}(\boldsymbol{x}^{(i)};\boldsymbol{\theta},\boldsymbol{\phi})$ for $\boldsymbol{\phi}$ will produce the correct $\boldsymbol{\phi}*$.

Therefore, by maximising $\mathcal{L}(\boldsymbol{x}^{(i)};\boldsymbol{\theta},\boldsymbol{\phi})$, we have the maximal trade-off between good reconstruction of our input data and $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})$ approximating the true intractable posterior, $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$.

4.2.5 Reparameterization Trick

Now that we have a computable lower bound for the marginal likelihood, we want to differentiate $\mathcal{L}(\boldsymbol{x}^{(i)};\boldsymbol{\theta},\boldsymbol{\phi})$ with respect to $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ to find their optimal parameters $\boldsymbol{\theta}^*$ and $\boldsymbol{\phi}^*$, such that $\mathcal{L}(\boldsymbol{x}^{(i)};\boldsymbol{\theta}^*,\boldsymbol{\phi}^*) \geq \mathcal{L}(\boldsymbol{x}^{(i)};\boldsymbol{\theta},\boldsymbol{\phi})$. However, we are unable to take the able to take the derivative of $p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{z})$ as depends on a random and non-deterministic sample $\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\cdot|\boldsymbol{x}^{(i)})$. We overcome this by choosing our encoder model $q_{\boldsymbol{\phi}}(\cdot|\boldsymbol{x}^{(i)})$ under certain conditions (explained below) such that we are able to reparameterise the random variable $\boldsymbol{z}$, as a differentiable transformation function $g_{\boldsymbol{\phi}}(\boldsymbol{\varepsilon},\boldsymbol{x}^{(i)})$, where $\boldsymbol{\varepsilon}$ is some auxiliary noise.

$$
\boldsymbol{z} = g_{\boldsymbol{\phi}}(\boldsymbol{\varepsilon},\boldsymbol{x}^{(i)}), \qquad \boldsymbol{\varepsilon} \sim p(\boldsymbol{\varepsilon}) \tag{17}
$$

We will now state the conditions where we can choose a differentiable transformation $g_{\boldsymbol{\phi}}(\boldsymbol{\varepsilon}, \boldsymbol{x}^{(i)})$ and auxiliary variable $\boldsymbol{\varepsilon} \sim p(\boldsymbol{\varepsilon})$ to replace $\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\cdot|\boldsymbol{x})$ for $\boldsymbol{z} = g_{\boldsymbol{\phi}}(\boldsymbol{\varepsilon}, \boldsymbol{x}^{(i)})$. [8]

1. $q_{\boldsymbol{\phi}}(\cdot|\boldsymbol{x})$ has a tractable inverse CDF. In this case, let $\varepsilon \sim \mathcal{U}(\mathbf{0}, \boldsymbol{I})$, and let $g_{\boldsymbol{\phi}}(\boldsymbol{\varepsilon}, \boldsymbol{x})$ be the inverse CDF of $q_{\boldsymbol{\phi}}(\cdot|\boldsymbol{x})$. Examples: Exponential, Cauchy, Logistic, Rayleigh, Pareto, Weibull, Reciprocal, Gompertz, Gumbel and Erlang distributions.

2. $q_{\boldsymbol{\phi}}(\cdot|\boldsymbol{x})$ is from any "location-scale" family of distributions. In this we can choose the standard distribution (with location = 0, scale = 1) as the auxiliary variable $\boldsymbol{\varepsilon}$, and then let $g_{\boldsymbol{\phi}}(\boldsymbol{\varepsilon}, \boldsymbol{x}) = \text{location} + \text{scale} \times \boldsymbol{\varepsilon}$. Examples: Laplace, Elliptical, Student's t, Logistic, Uniform, Triangular and Gaussian distributions.

3. $q_{\boldsymbol{\phi}}(\cdot|\boldsymbol{x})$ is a composition of random variables as different transformations of auxiliary variables. Then we take $g_{\boldsymbol{\phi}}(\boldsymbol{\varepsilon}, \boldsymbol{x})$ their composition of the above reparmeterizations. Examples: Log-Normal (exponentiation of normally distributed variable), Gamma (a sum over exponentially distributed variables), Dirichlet (weighted sum of Gamma variates), Beta, Chi-Squared, and F distributions.

Hence, by using the reparameterization (17), we are able to form a Monte Carlo estimate of expectation for the function $\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})$ with respect to our encoder model $q_{\boldsymbol{\phi}}(\cdot|\boldsymbol{x}^{(i)})$, as the following,

$$\mathbb{E}_{\boldsymbol{z}\sim q_{\boldsymbol{\phi}}(\cdot|\boldsymbol{x}^{(i)})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})] = \mathbb{E}_{\boldsymbol{\varepsilon}\sim p(\boldsymbol{\varepsilon})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|g_{\boldsymbol{\phi}}(\boldsymbol{\varepsilon}, \boldsymbol{x}^{(i)}))] \tag{18}$$

$$\simeq \frac{1}{L}\sum_{l=1}^{L} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|g_{\boldsymbol{\phi}}(\boldsymbol{\varepsilon}^{(l)}, \boldsymbol{x}^{(i)})), \quad \boldsymbol{\varepsilon}^{(l)} \sim p(\boldsymbol{\varepsilon}) \tag{19}$$

Now applying the method above to our ELBO, we obtain a differentiable estimator $\mathcal{L}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}, \boldsymbol{\phi}) \simeq \tilde{\mathcal{L}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}, \boldsymbol{\phi})$.

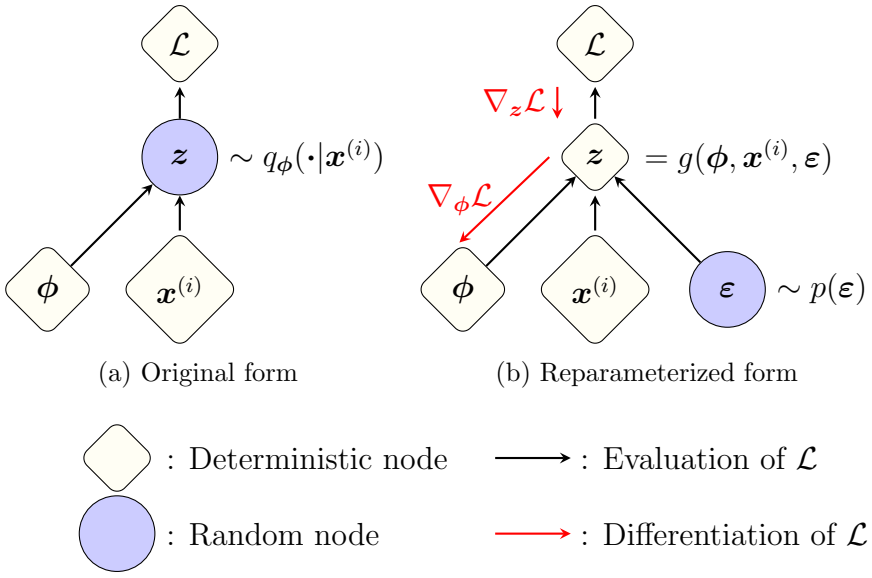
$$\tilde{\mathcal{L}}(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}, \boldsymbol{\phi}) = \frac{1}{L}\sum_{l=1}^{L} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z}^{(i,l)}) - D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_{\boldsymbol{\theta}}(\boldsymbol{z})) \tag{20}$$

$$\text{where} \quad \boldsymbol{z}^{(i,l)} = g_{\boldsymbol{\phi}}(\boldsymbol{\varepsilon}^{(i,l)}, \boldsymbol{x}^{(i)}), \quad \text{and} \quad \boldsymbol{\varepsilon}^{(l)} \sim p(\boldsymbol{\varepsilon})$$

Now suppose our dateset $\boldsymbol{X}$ consisting of N data points. We are able to preformed minibatch training to the above result, to get the following approximation of $\mathcal{L}$,

$$\mathcal{L}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}, \boldsymbol{\phi}) \simeq \tilde{\mathcal{L}}^{M}(\boldsymbol{X}^{M}; \boldsymbol{\theta}, \boldsymbol{\phi}) = \frac{N}{M}\sum_{i=1}^{M} \tilde{\mathcal{L}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}, \boldsymbol{\phi})$$

Where, $X^M = (\boldsymbol{x}^{(i)})_{i=1}^{M}$ are M randomly chosen data points from our training data $\boldsymbol{X}$. If we perform minibatch training with M is large enough (i.e. $M = 100$), we are able to set $L = 1$ in (20). As experiments show that there is no significant performance decrease by taking a single sample [8]. This allows us to save time training as we don't have to preform multiple samples for each data point, decreasing the complexity of training.



(a) Original form (b) Reparameterized form

(c) Figure Key

Figure 12: Illustration of the reparameterization trick used to get a differentiable $\mathcal{L}$ w.r.t $\boldsymbol{\phi}$. ([9], p.22)

4.3 Gaussian VAE Example

We will now demonstrate this reparameterization trick, by assuming that our prior over the latent variables, is given by an isotropic multivariate Gaussian $p_{\boldsymbol{\theta}}(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{0}, \boldsymbol{I})$ and our true posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ is a multivariate Gaussian. Therefore, we will approximate the true posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ with a multivariate parametric Gaussian model that has a diagonal covariance, $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2 I)$. Where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ are function of $\boldsymbol{\phi}$. We chose a diagonal covariance structure to promote the latent variables learning unique generalise features.

Since our parametric model $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ is a "location-scale" distribution, we will use the following reparameterisation,

$$\boldsymbol{z} = g_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\varepsilon}) = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$$

Where $\odot$ is element-wise multiplication. Therefore, by using the calculation that we carried out in Section 2.2, we obtain the following approximation for $\tilde{\mathcal{L}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}, \boldsymbol{\phi})$,

$$\tilde{\mathcal{L}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}, \boldsymbol{\phi}) \simeq \frac{1}{2}\sum_{j=1}^{m}(1 + \log((\boldsymbol{\sigma}_j^{(i)})^2) - (\boldsymbol{\mu}_j^{(i)})^2 - (\boldsymbol{\sigma}_j^{(i)})^2) + \frac{1}{L}\sum_{l=1}^{L}\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z}^{(i,l)})$$

$$\text{where} \quad \boldsymbol{z}^{(i,l)} = \boldsymbol{\mu}^{(i)} + \boldsymbol{\sigma}^{(i)} \odot \boldsymbol{\varepsilon}^{(l)}, \quad \boldsymbol{\varepsilon}^{(l)} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}) \tag{21}$$

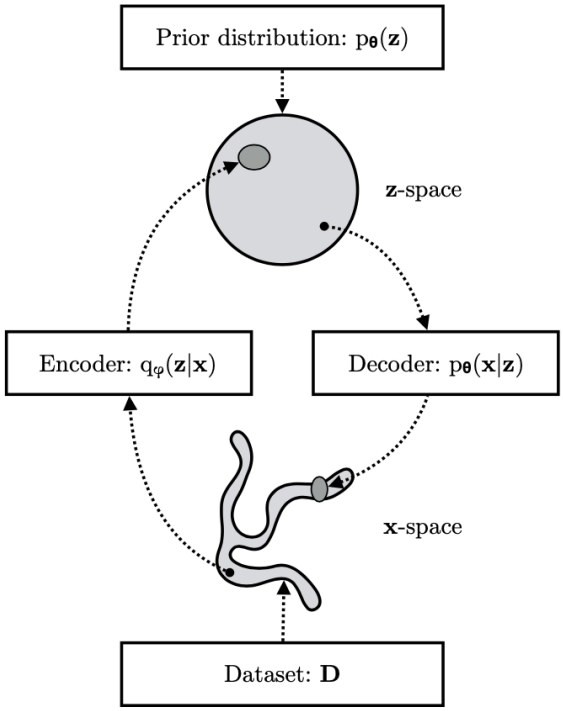


Figure 13: A VAE learns the mappings between our observed data space which its distribution which is typically complicated, and a feature space, whose distribution can be relatively simple (such as Gaussian, as in this figure). ([9], p.17)

4.4 Application of VAE to MNIST Dataset

In this section we will show how variation autoencoder trained on the MNIST data set, can be used to generate images of handwritten digits. We demonstrate this by following the same set up as the previous section. Suppose, that the prior over the latent variable that are this given by an isotropic multivariate Gaussian $p_{\boldsymbol{\theta}}(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and our true posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ is a multivariate Gaussian. Therefore, we will approximate the true posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$, by a multivariate parametric Gaussian model that has a diagonal covariance, $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2 I)$. Where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ are functions of $\boldsymbol{\phi}$. Therefore by the previous section we have,

$$\tilde{\mathcal{L}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}, \boldsymbol{\phi}) \simeq \frac{1}{2} \sum_{j=1}^{m} (1 + \log((\boldsymbol{\sigma}_j^{(i)})^2) - (\boldsymbol{\mu}_j^{(i)})^2 - (\boldsymbol{\sigma}_j^{(i)})^2) + \frac{1}{L} \sum_{l=1}^{L} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z}^{(i,l)})$$

$$\text{where} \quad \boldsymbol{z}^{(i,l)} = \boldsymbol{\mu}^{(i)} + \boldsymbol{\sigma}^{(i)} \odot \boldsymbol{\varepsilon}^{(l)}, \quad \boldsymbol{\varepsilon}^{(l)} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$$

To produce the $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ that will be fed into $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, we define EncoderNeuralNet$_{\boldsymbol{\phi}}$ that encodes our data $\boldsymbol{x}^i$ into a $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$. EncoderNeuralNet$_{\boldsymbol{\phi}}$ is composed of a convolutional neural network (CNN) and dense layers, which inputs an image from the MNIST data sets (which are a 28×28 matrix) and produces the encoded $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$.

We will also make the additional assumption that $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ is a multivariate parametric Gaussian of the form, $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\hat{\boldsymbol{x}}^{(i)}, \sigma_2 \boldsymbol{I})$, with $\hat{\boldsymbol{x}}^{(i)} = \text{DecoderNeuralNet}_{\boldsymbol{\theta}}(\boldsymbol{z}^{(i,l)})$. Where DecoderNeuralNet$_{\boldsymbol{\theta}}$ is composed of a convolutional neural network (CNN) and dense layers, that takes in our sampled point $\boldsymbol{z}^{(i,l)}$ and produces our mean $\hat{\boldsymbol{x}}^{(i)}$.

parameterised by $\boldsymbol{\theta}$ and σ_2 is variant in our data is. formally, we would learn the parameter σ_2, however, we will simplify our problem by setting it to 1 (i.e $\sigma_2 = 1$). Therefore, $\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z}^{(i,l)})$ is given by,

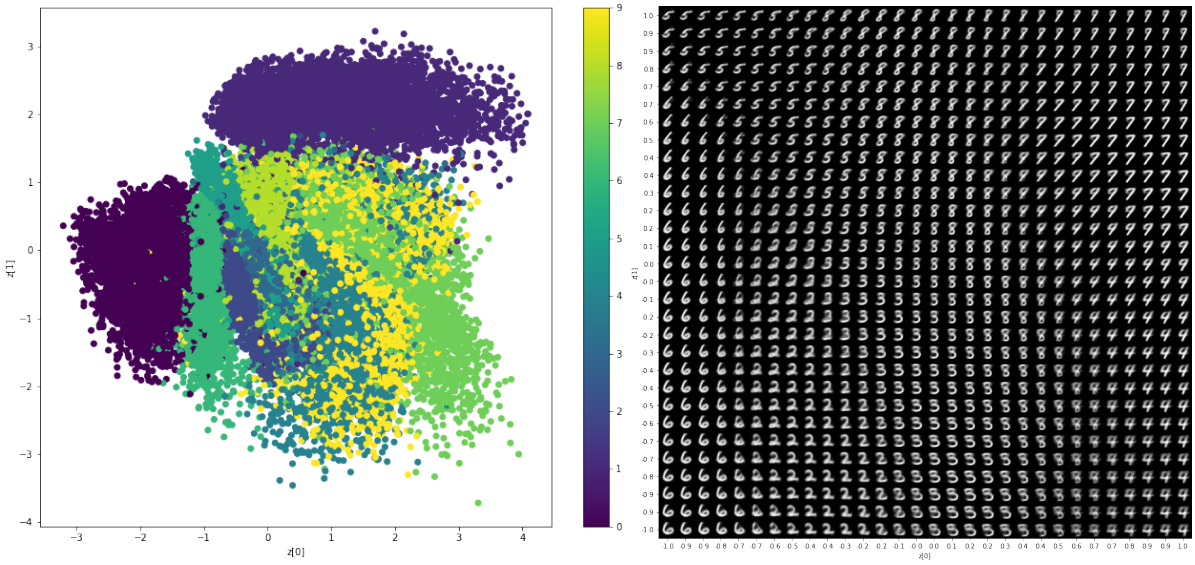
$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z}^{(i,l)}) = -\frac{1}{2}||\boldsymbol{x}^{(i)} - \hat{\boldsymbol{x}}^{(i)}||^2 + C$$

Where C is a constant. Also since we are planning on using minibatch training when in the training procedure, we can simplify our expression for $\tilde{\mathcal{L}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}, \boldsymbol{\phi})$ as we can set $L = 1$. Therefore $\tilde{\mathcal{L}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}, \boldsymbol{\phi})$ takes the form of,

$$\begin{aligned}\tilde{\mathcal{L}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}, \boldsymbol{\phi}) &\simeq \frac{1}{2}\sum_{j=1}^{m}(1 + \log((\boldsymbol{\sigma}_j^{(i)})^2) - (\boldsymbol{\mu}_j^{(i)})^2 - (\boldsymbol{\sigma}_j^{(i)})^2) + \log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z}^{(i)}) \\ &= \frac{1}{2}\sum_{j=1}^{m}(1 + \log((\boldsymbol{\sigma}_j^{(i)})^2) - (\boldsymbol{\mu}_j^{(i)})^2 - (\boldsymbol{\sigma}_j^{(i)})^2) - \frac{1}{2}||\boldsymbol{x}^{(i)} - \hat{\boldsymbol{x}}^{(i)}||^2 + C\end{aligned}$$

$$\text{where} \quad \boldsymbol{z}^{(i)} = \boldsymbol{\mu}^{(i)} + \boldsymbol{\sigma}^{(i)} \odot \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$$

After training our model (described in Figure. 15) with 70,000 samples, batch sizes of 128, and 30 epochs. We get the following plots that describes the behaviour in latent space,



(a) (b)

Figure 14: (a) Spatial distribution of latent space for MNIST digits (b) Learnt MNIST manifold in data-space.

We can see in both the figures that the model does a good job at regularising the distribution in latent space, since we chose a Gaussian centred at the origin. This allows for the KL divergence term in the ELBO to be smaller if the encoder is well-behaved, (i.e. is a relatively smooth distribution which is small far away from the origin). Thus, the KL term regularises the VAE. We can see this effect taking hold as most of the MNIST digits get regularised close to the origin.

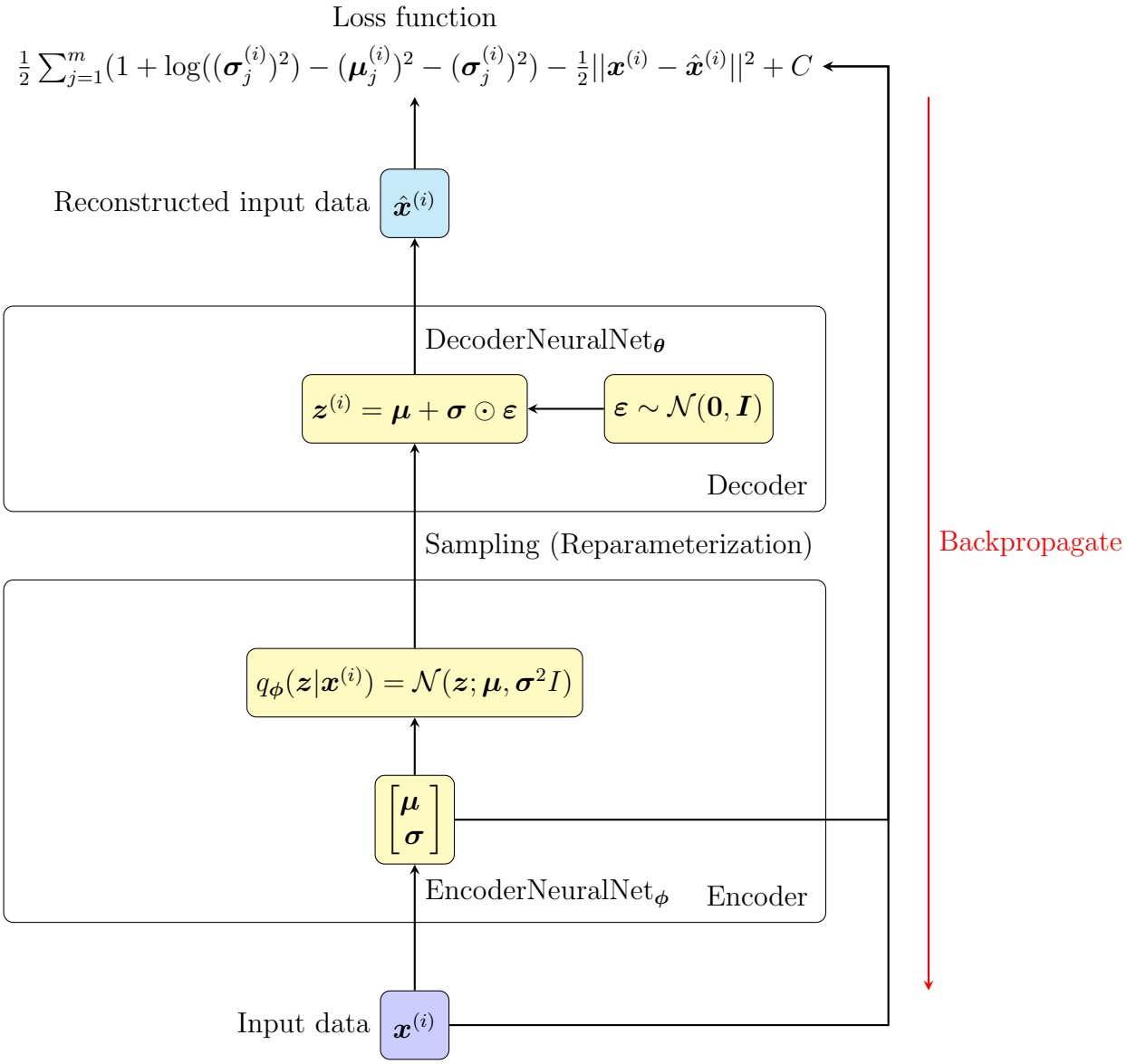


Figure 15: Trainable variational autoencoder

5 Neural Ordinary Differential Equations

We will now introduce the final piece of the background knowledge required before we tie it all together in the next chapter. As the title of the chapter may suggest, we will be looking at the combination of neural networks and ordinary differential equations (ODEs). The reason for studying this architecture is because in the next chapter we are going to be looking at modelling time series data with the use of VAE. However, we have only looked at VAEs without a time dependency and therefore the Neural ODE architecture will be used to help us close that gap.

5.1 Motivation

Many models in machine learning can be described as discrete transformations of hidden states, and can be expressed by the following relationship,

$$\boldsymbol{z}_{t+1} = \boldsymbol{z}_t + \boldsymbol{\phi}(\boldsymbol{z}_t, \boldsymbol{\theta}_t), \qquad t \in \{0, \dots, T\}, \;\; \boldsymbol{z}_t \in \mathbb{R}^m, \;\; \boldsymbol{\phi}(\cdot, \boldsymbol{\theta}_t) : \mathbb{R}^m \to \mathbb{R}^m \tag{22}$$

Where t is the depth of the network and $\boldsymbol{\theta}_t$ parameterize the network at depth t. This relationship closely resembles the forward Euler method for solving ordinary differential equations for discrete time steps, $t \in \{0, \dots, T\}$. Therefore, analogously we can think of the network at depth t, representing the network at time t.

Hence, with the idea that (22) is the solution of the forward Euler method for solving an ODE at discrete layers $\{0, \dots, T\}$, we can hypothesise that $\boldsymbol{z}(t)$ is continuous and governed by,

$$\frac{d\boldsymbol{z}(t)}{dt} = \boldsymbol{f}(\boldsymbol{z}(t), t, \boldsymbol{\theta}), \qquad \boldsymbol{z}(0) = \boldsymbol{z}_0 \tag{23}$$

Where $\boldsymbol{f} : \mathbb{R}^m \to \mathbb{R}^m$ is a neural network, parameterized by $\boldsymbol{\theta}$. This can always be achieved as we can ensure by our choice of $\boldsymbol{\theta}$ that (22) holds (i.e $\boldsymbol{f}(\boldsymbol{z}(t), t, \boldsymbol{\theta}) = \phi(\boldsymbol{z}_t, \boldsymbol{\theta}_t), \forall t \in \{0, \dots, T\})$.

Thus, we can think of the layers of our network in the continuous sense. Therefore, given an initial state of the network $\boldsymbol{z}_0$, we can solve (23) forwards to obtain the network's value at an arbitrary layer. This solution is unique and can always be achieved by treating the ODE solver as a black-box.

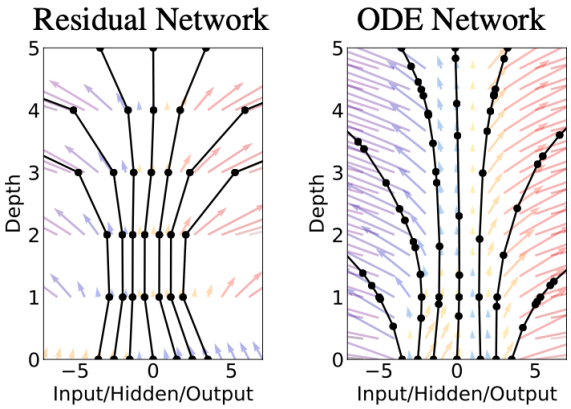


Figure 16: Left: A Residual network defines a discrete sequence of finite transformations. Right: An ODE network defines a vector field, of continuously state transforms. [1]

5.1.1 Toy Circular Dynamics Example

Since we are able to make the analogous link between layer depth of our network and time. We will use this idea to try and learn a continuous function of time. To achieve this, we will frame our problem using (23) and find the value of $\boldsymbol{\theta}$ that

minimises the difference between $\boldsymbol{z}(t)$ and the true solution.

We will make life easier for ourselves, we will try to learn a continuous function of time that we know its form as an initial value problem,

$$\dot{\boldsymbol{x}}(t) = \frac{d}{dt}\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} -y(t) \\ x(t) \end{bmatrix} =: \boldsymbol{f}(\boldsymbol{x}(t)), \quad \boldsymbol{x}(0) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad t \in [0, 2\pi] \tag{24}$$

We notice the following relationship,

$$\ddot{x}(t) = -\dot{y} = -x(t), \qquad x(0) = 1$$

Therefore, by the symmetry of our problem, we obtain,

$$\ddot{y}(t) = -y(t), \qquad y(0) = 0$$

Hence, the unique solution to our initial value problem is given by,

$$\boldsymbol{x}(t) = \begin{bmatrix} \cos(t) \\ \sin(t) \end{bmatrix}, \qquad t \in [0, 2\pi] \tag{25}$$

To learn the correct $\boldsymbol{z}(t)$, we will use the linear function, $\boldsymbol{f}_\theta : \mathbb{R}^2 \to \mathbb{R}^2$ as our neural network.

$$\dot{\boldsymbol{z}}(t) = \frac{d}{dt}\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \boldsymbol{f}_\theta(\boldsymbol{z}(t)) \coloneqq \begin{bmatrix} \theta_1 & \theta_2 \\ \theta_3 & \theta_4 \end{bmatrix}\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} + \begin{bmatrix} \theta_5 \\ \theta_6 \end{bmatrix}, \boldsymbol{z}(0) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, t \in [0, 2\pi] \tag{26}$$

Since we have the explicit form of the continuous function that we want to model (given by (25)), we will artificially simulate data points for our problem. This is done by randomly sampling a stopping time t_{stop} from $\mathcal{U}(0, 2\pi)$ (i.e $t_{\text{stop}} \sim \mathcal{U}(0, 2\pi)$) and using (25) to calculated the true value of the solution at that time. After doing this procedure 12,800 times, we have a large enough data set that we can use for training. To form a prediction from our network, we will use the fourth-order Runge–Kutta (RK4) method to integrate (26) forward in time to t_{stop}. Then we will use the Euclidean norm between our prediction and the true solution to quantify how well our prediction is and finally, we will backpropagate the error through the RK4 integration, as it has discretised the ODE.

$$\begin{aligned} \text{L2 loss function } L &= ||\boldsymbol{x}_{\text{stop}} - \boldsymbol{z}_{\text{stop}}||^2 \\ &= ||\boldsymbol{x}_{\text{stop}} - \text{RK4}((1,0)^T, \boldsymbol{f}_\theta, 0, t_{\text{stop}})||^2 \\ \implies \frac{\partial L}{\partial \boldsymbol{\theta}} &= \frac{\partial ||\boldsymbol{x}_{\text{stop}} - \text{RK4}((1,0)^T, \boldsymbol{f}_\theta, 0, t_{\text{stop}})||^2}{\partial \boldsymbol{\theta}} \end{aligned}$$

We demonstrate this whole training process by the following diagram,

After training using the process described in Fig (17), with 12,800 sample points, batch sizes of 64, and 4 epochs, we get the following forcing function,

$$\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z}(t)) = \begin{bmatrix} -0.00 & 1.01 \\ -1.01 & 0.00 \end{bmatrix}\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} + \begin{bmatrix} -0.01 \\ -0.00 \end{bmatrix} \approx \begin{bmatrix} y(t) \\ -x(t) \end{bmatrix}$$

Due to the symmetrical nature of our ODE example we still get same set of equations to solve,

$$\ddot{x}(t) = \dot{y}(t) = -x(t), \quad x(0) = 1, \quad \text{and} \quad \ddot{y}(t) = -y(t), \quad y(0) = 0$$

As this example was done this a linear function, this reinforces that our theory should hold for neural network as they are a compositions of linear functions.

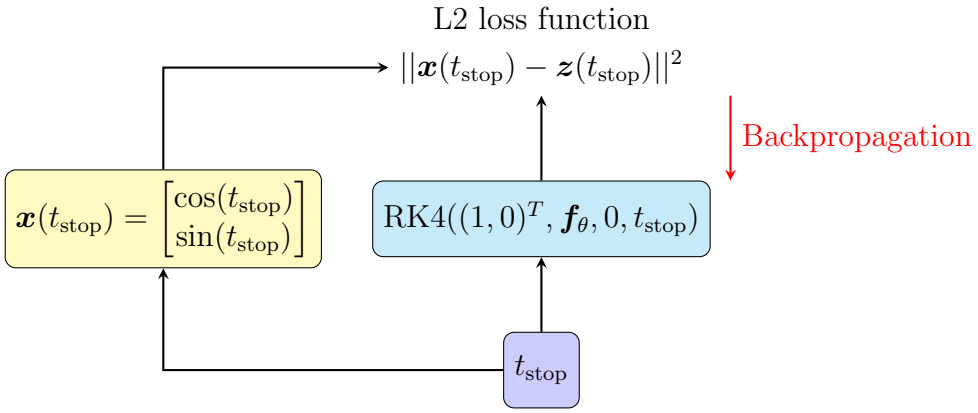
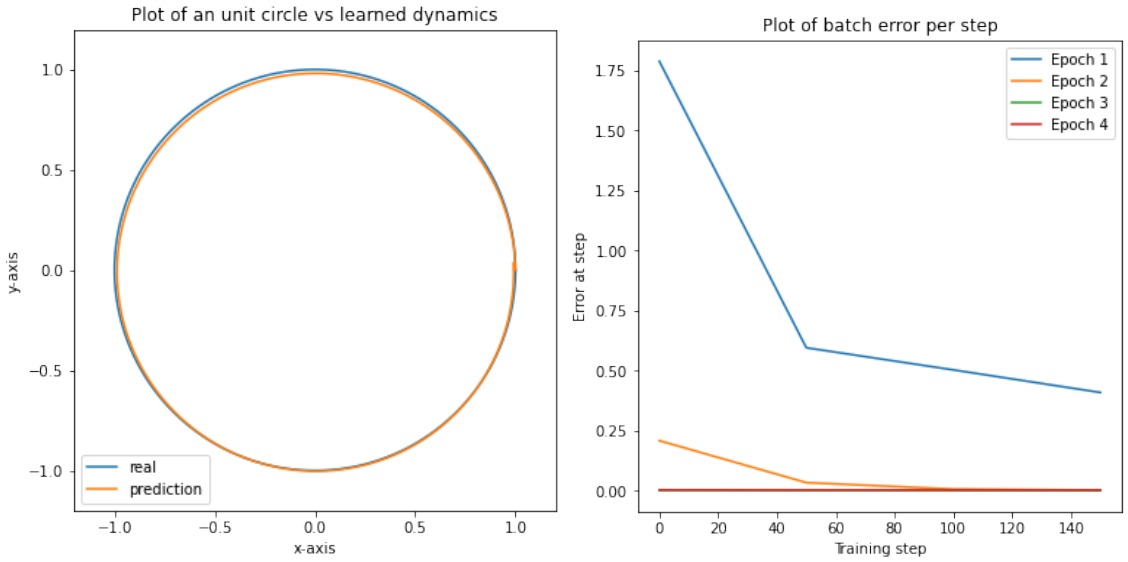


Figure 17: Training model for learning (25) using (26) and the RK4 method.



(a) (b)

Figure 18: (a) Plot of the networks learnt trajectory $\boldsymbol{z}(t)$ (orange) vs the true solution's trajectory $\boldsymbol{x}(t)$ (blue) to the problem (24). (b) Plot of the every 50th batch error over four epochs (converge to an error of 0.5×10^{-3}). [3]

5.2 Formalisation and Improvements

Like we did in our toy example (5.1.1), the native approach for training (23) is to apply backpropagation through the black-box solver. This works due to the block-box solver discretising the ODE, allowing us to take the derivative with respect to the parameters. However, this leads to high memory cost and introduces additional numerical error. This is because the black-box solver can be thought of as discrete transformation and compositions of the forcing function. As this procedure will be done a lot, this causes the derivative with respect to the parameters to be expensive, as they appear in the compositions. Hence, we seek to find a better solution to overcome this problem. We achieve this by deriving an expression for the derivative of the loss function with respect to the model parameters, that involves solving an ODE backward through the layers. This is more appealing approach as modern ODE solvers allow for the problem to scale linearly with size. We can also to control

the tolerance of our solution, allowing for the ability to dictate how accurate our training is; Thus, helping us prevent over-fitting.

To make understanding the theory more digestible, we will impose the following simplification to our problem. We will only consider the network at being evaluated at two discrete points, t_{start} and t_{stop}, and we will look at minimising the loss produced over that interval. This simplification does not cause any initiations to our theory, as we can extend this theory to multiple observations (i.e minimising the loss function over $t_0, \ldots, t_N$ observations) and is shown in the Appendix 2.2.

We will now define the scalar loss function $L : \mathbb{R}^m \to \mathbb{R}$, that we alluded to above, where the input is the output of our black-box solver.

$$L(\boldsymbol{z}(t_1)) = L\left(\boldsymbol{z}(t_0) + \int_{t_0}^{t_1} \boldsymbol{f}(\boldsymbol{z}(t), t, \theta)dt\right) = L(\text{ODESolve}(\boldsymbol{z}(t_0), \boldsymbol{f}, t_0, t_1, \theta))$$

We aim to find the value of $\boldsymbol{\theta}$ that minimises L, as this will provide us with a model that captures the dynamics of our data closely. To obtain this we firstly define the adjoint, which is gradient of L with respect to the hidden state $\boldsymbol{z}(t)$,

$$\boldsymbol{a}(t) = \frac{\partial L}{\partial \boldsymbol{z}(t)} \tag{27}$$

It is important to remember that t is the layer depth and it is now continuous, so this holds for all $t \geq 0$.

We show in Section 2.1, that the derivative of the adjoint with respect to the layer is given by,

$$\frac{d\boldsymbol{a}^T(t)}{dt} = -\boldsymbol{a}^T \frac{\partial \boldsymbol{f}(\boldsymbol{z}(t), \theta, t)}{\partial \boldsymbol{z}} \tag{28}$$

Since this does not tell use how L changes with respect to the parameters $\boldsymbol{\theta}$ and t. Therefore we define the augmented state $\boldsymbol{z}_{\text{aug}}(t)$, to allow us to do so.

$$\boldsymbol{z}_{\text{aug}}(t) = \begin{bmatrix} \boldsymbol{z}(t) \\ \theta \\ t \end{bmatrix}$$

We now derive the new dynamical system,

$$\frac{d\boldsymbol{z}_{\text{aug}}(t)}{dt} = \boldsymbol{f}_{aug}([\boldsymbol{z}, \theta, t]) := \begin{bmatrix} \boldsymbol{f}([\boldsymbol{z}, \theta, t]) \\ \boldsymbol{0} \\ 1 \end{bmatrix}$$

$$\boldsymbol{a}_{\text{aug}}(t) := \begin{bmatrix} \boldsymbol{a} \\ \boldsymbol{a}_\theta \\ \boldsymbol{a}_t \end{bmatrix}(t), \quad \boldsymbol{a}_\theta(t) := \frac{dL}{d\theta(t)}, \quad \boldsymbol{a}_t(t) := \frac{dL}{dt(t)}$$

Therefore, we are able to use (28) on our new dynamics to obtain,

$$\frac{d\boldsymbol{a}_{\text{aug}}^T(t)}{dt} = -\boldsymbol{a}_{\text{aug}}^T \frac{\partial \boldsymbol{f}_{\text{aug}}}{\partial [\boldsymbol{z}, \theta, t]} \tag{29}$$

Thus, by the calculating (29) (which is shown in Appendix 2.2), we obtain the derivative of $\boldsymbol{a}_\theta(t)$ with respect to the layer,

$$\frac{d\boldsymbol{a}_\theta^T(t)}{dt} = \frac{d}{dt}\left(\frac{dL^T}{d\theta}\right) = -\boldsymbol{a}^T\frac{\partial \boldsymbol{f}(\boldsymbol{z}(t),\theta,t)}{\partial\theta}$$

Hence, by integrating this expression with backwards over the layers and assuming that $\boldsymbol{a}_\theta(t_1) = \mathbf{0}$ (explained in the Appendix 2.2), we achieved what we set out to obtain,

$$\frac{dL^T}{d\theta} = -\int_{t_1}^{t_0} \boldsymbol{a}^T(t)\frac{\partial \boldsymbol{f}(\boldsymbol{z}(t),\theta,t)}{\partial\theta}dt \tag{30}$$

To understand why we integrate backwards over the layers, we draw parallels from the discrete case. As the loss function depends of the output of the network (or in our case at t_1) we backpropgrate the error through the network updating the models as we go. This is the same idea in the continuous case by integrating from t_1 to t_0 (i.e from the output to the input of our network) allowing us to compute the overall sensitivity of the model parameters.

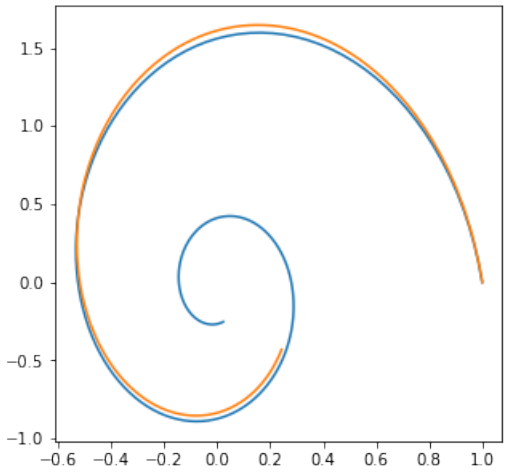


Figure 19: Plot of the networks learnt trajectory (orange) vs the true solution's trajectory (blue) using the adjoint sensitivity method to compute the gradients for a spiral dynamics. [3]

6 Generative Latent Time-series Models

6.1 Overview

We will now tie together the theory established throughout this paper by developing a generative approach for modelling continuous time series-data. This approach is a continuation of variational autoencoders that learns the time-trajectory of our sampled feature point $z_{t_0} \sim p_{\boldsymbol{\theta}}(\boldsymbol{z})$ in feature space.

We pose the question, we want to solve. Suppose that we have the data, $\boldsymbol{x}_{t_0}, \ldots, \boldsymbol{x}_{t_N}$ which are $N+1$ data observation at times $t_0, \ldots, t_N$. Therefore

We do this by making the following adaptations to the variational autoencoder model:

1. The input data $\boldsymbol{x}_{t_0}, \ldots, \boldsymbol{x}_{t_N}$ is encoded as a distribution $p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x}_{t_0}, \ldots, \boldsymbol{x}_{t_N})$ in feature space

2. We then sample the point $\boldsymbol{z}_{t_0}$ from the distribution in feature space

3. Then we integrate the sampled point to the observation times $t_0, \ldots, t_N$, $\boldsymbol{z}_{t_0}, \ldots, \boldsymbol{z}_{t_N} = \text{ODESolve}(\boldsymbol{z}_{t_0}, \boldsymbol{f}, t_0, \ldots, t_N, \boldsymbol{\theta})$. Where $\boldsymbol{f}$ is the forcing function of that describes the feature point through time,

$$\frac{d\boldsymbol{z}(t)}{dt} = \boldsymbol{f}(\boldsymbol{z}(t)), \qquad \boldsymbol{z}(t_0) = \boldsymbol{z}_{t_0} \tag{31}$$

4. Finally, the points $\boldsymbol{z}_{t_0}, \ldots, \boldsymbol{z}_{t_N}$ are then decoded into data-space by $\hat{\boldsymbol{x}}_{t_i} = p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_{t_i})$ to get the reconstructed input data $\hat{\boldsymbol{x}}_{t_0}, \ldots, \hat{\boldsymbol{x}}_{t_N}$

Because the forcing function $\boldsymbol{f}$ is unknown, we will use a neural ordinary differential equation to learn the correct trajectory in feature space. To ensure that our trajectory in feature space is uniquely defined, we make the assumption that the neural network, $\boldsymbol{f}_{\boldsymbol{\theta}} : \mathbb{R}^m \to \mathbb{R}^m$ parameterized by $\boldsymbol{\theta}$ is a time-invariant function.

We can express this whole process probabilistically by the following the sampling procedure,

$$\boldsymbol{z}_{t_0} \sim p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \tag{32}$$

$$\boldsymbol{z}_{t_0}, \ldots, \boldsymbol{z}_{t_N} = \text{ODESolve}(\boldsymbol{z}_{t_0}, \boldsymbol{f}, t_0, \ldots, t_N, \boldsymbol{\theta}) \tag{33}$$

$$\text{draw each } \boldsymbol{x}_{t_i} \sim p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{z}_{t_i}) \tag{34}$$

6.2 Training

Since this model is a modification of variational autoencoders, we will build off its training model (Fig 15) to accommodate these changes. We start by modifying our $\text{EncoderNeuralNet}_{\boldsymbol{\phi}}$ to allow for multiple observations to be used in the encoding of $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$. This is done by feeding our observation through a composition of a recurrent neural network (RNN) and a dense layer,

$$\text{RNN}(\boldsymbol{x}_{t_0}, \ldots, \boldsymbol{x}_{t_N}) : \mathbb{R}^n \times \ldots \times \mathbb{R}^n \to \mathbb{R}^n, \ \text{DenseLayer}(\cdot) = \begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\sigma} \end{bmatrix} : \mathbb{R}^n \to \mathbb{R}^{2m}$$

$$\text{EncoderNeuralNet}_{\boldsymbol{\phi}}(\cdot) \coloneqq \text{DenseLayer}(\text{RNN}(\cdot)) : \mathbb{R}^n \times \ldots \times \mathbb{R}^n \to \mathbb{R}^{2m}$$

We choose our encoder in this way to ensure that the temporal nature of our data gets encapsulates into the encoding of our $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$.

Like we did in our example 4.4, we assume that the our prior over the latent variables is given by an isotropic multivariate Gaussian $p_{\boldsymbol{\theta}}(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{0}, \boldsymbol{I})$ and our true posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}_{t_0}, \ldots, \boldsymbol{x}_{t_N})$ takes the form of a multivariate Gaussian; Therefore, we will approximate $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}_{t_0}, \ldots, \boldsymbol{x}_{t_N})$ with a multivariate Gaussian with diagonal covariance, $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_{t_0}, \ldots, \boldsymbol{x}_{t_N}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2 I)$.

Hence, our sampling procedure from our encoder now takes the form,

$$\boldsymbol{z}_{t_0} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_{t_0}, \ldots, \boldsymbol{x}_{t_N}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2 I)$$

We will use the same reparameteriation trick described in section 4.3, to get,

$$\boldsymbol{z}_{t_0} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$$

Now we will integrated $\boldsymbol{f_\theta}$ forwards in feature-space to our observations times,

$$\boldsymbol{z}_{t_0}, \ldots, \boldsymbol{z}_{t_N} = \text{ODESolve}(\boldsymbol{z}_{t_0}, \boldsymbol{f_\theta}, t_0, \ldots, t_N, \boldsymbol{\theta})$$

We then decode these points into data-space by the following mapping,

$$\text{DecoderNeuralNet}_{\boldsymbol{\theta}}(\boldsymbol{x}|\cdot) : \mathbb{R}^m \to \mathbb{R}^n$$

$$\hat{\boldsymbol{x}}_{t_i} = \text{DecoderNeuralNet}_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_{t_i}), \quad i \in \{0, \ldots, N\}$$

This whole processes is described by the following figure,

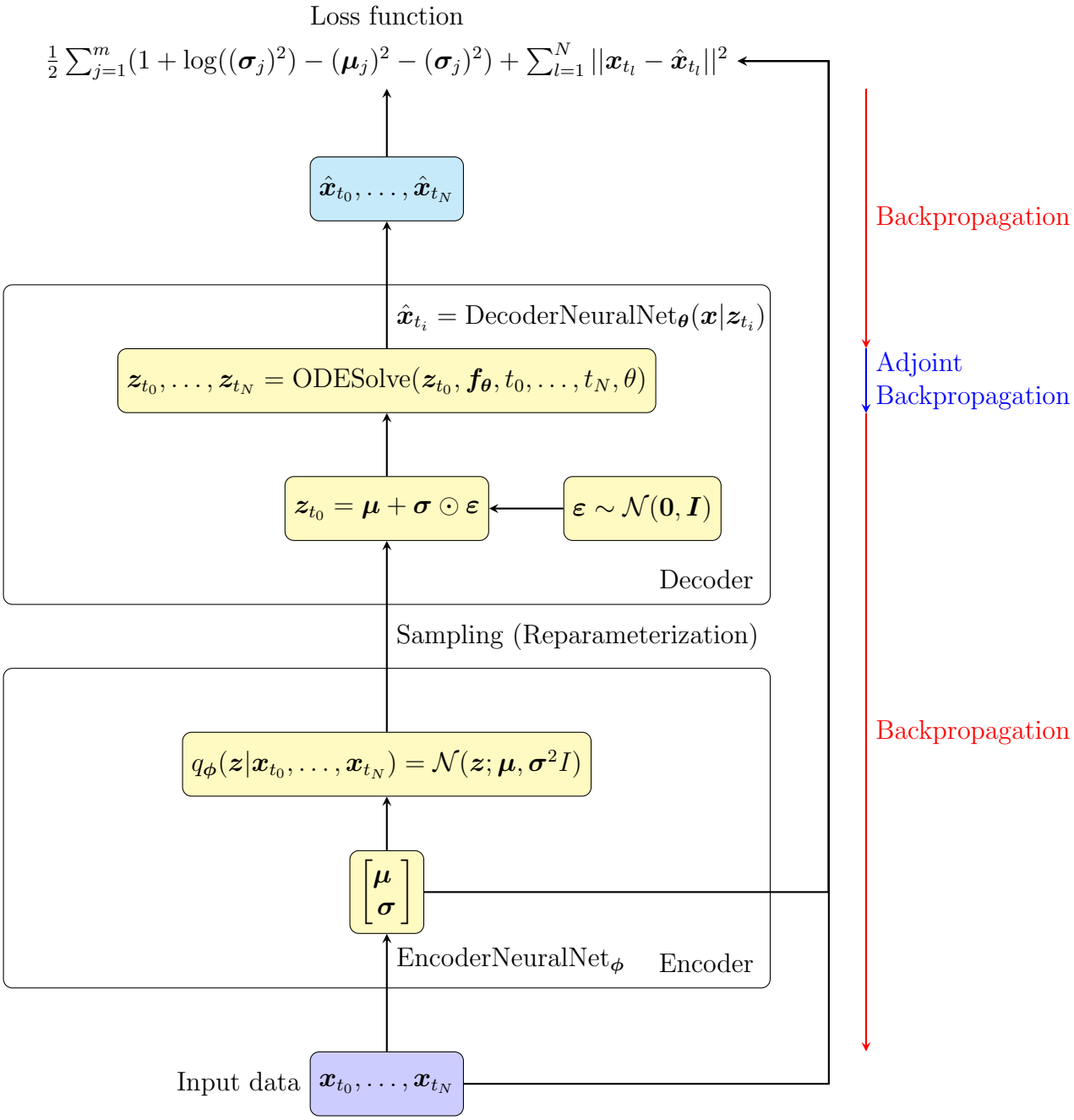


Figure 20: Trainable generative latent time-series model

It may not be apparently clear what form the adjoint takes at our observed

values which is used in alg **??**.

$$
\begin{aligned}
\boldsymbol{a}(t_i) &= \left.\frac{\partial L}{\partial \boldsymbol{z}(t)}\right|_{t=t_i} \\
&= \left.\frac{\partial}{\partial \boldsymbol{z}(t)}\left(\frac{1}{2}\sum_{j=1}^{m}(1+\log((\boldsymbol{\sigma}_j)^2)-(\boldsymbol{\mu}_j)^2-(\boldsymbol{\sigma}_j)^2)-\frac{1}{2}\sum_{l=1}^{N}||\boldsymbol{x}_{t_l}-\hat{\boldsymbol{x}}_{t_l}||^2\right)\right|_{t=t_i} \\
&= \left.\frac{\partial}{\partial \boldsymbol{z}(t)}\left(-\frac{1}{2}\sum_{l=1}^{N}||\boldsymbol{x}_{t_l}-\text{DecoderNeuralNet}_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_{t_i})||^2\right)\right|_{t=t_i} \\
&= ||\boldsymbol{x}_{t_i}-\text{DecoderNeuralNet}_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_{t_i})||\left.\frac{\partial \text{DecoderNeuralNet}_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_{t_i})}{\partial \boldsymbol{z}(t)}\right|_{t=t_i} \qquad \text{(Chain Rule)}
\end{aligned}
$$

This applications haves made the implicit assumption that our oberservation in data space is uniformly sampled; However, this is not normally the case. To overcome this to make the following adaptation to the above model,

$$\text{RNN}(\{\boldsymbol{x}_{t_0},t_0\},\ldots,\{\boldsymbol{x}_{t_N},t_N\}):\mathbb{R}^{(n+1)}\times\ldots\times\mathbb{R}^{(n+1)}\to\mathbb{R}^{(n+1)}$$

$$\text{DenseLayer}(\cdot)=\begin{bmatrix}\boldsymbol{\mu}\\ \boldsymbol{\sigma}\end{bmatrix}:\mathbb{R}^{(n+1)}\to\mathbb{R}^{2m}$$

$$\text{EncoderNeuralNet}_{\boldsymbol{\phi}}(\cdot)\coloneqq\text{DenseLayer}(\text{RNN}(\cdot)):\mathbb{R}^{(n+1)}\times\ldots\times\mathbb{R}^{(n+1)}\to\mathbb{R}^{2m}$$

This concatenation of the observation and its time, allows for the learning of the distribution of the sampled observations times.

6.3 Bi-directional Spiral Example

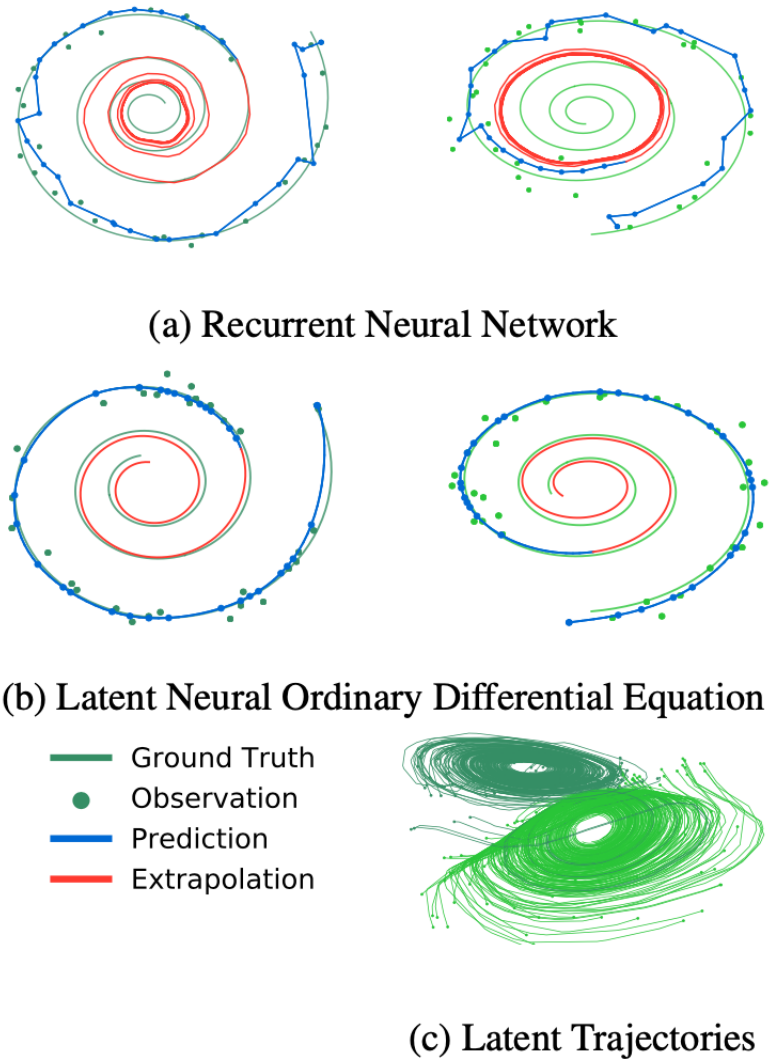


Figure 21: (a): Reconstruction and extrapolation of spirals with irregular time points by a recurrent neural network. (b): Reconstructions and extrapo- lations by a latent neural ODE. Blue curve shows model prediction. Red shows extrapolation. (c) A projection of inferred 4-dimensional latent ODE trajectories onto their first two dimensions. Color indicates the direction of the corresponding trajec- tory. The model has learned latent dynamics which distinguishes the two directions.[1]

Appendix A

Little Results and Derivations

1 Gibbs' Inequality [2]

Let $p = \{p_1, \ldots, p_k\}$ and $q = \{q_1, \ldots, q_k\}$ be two arbitrary discrete probability distributions, then Gibbs' inequality states that,

$$-\sum_{i=1}^{k} p_i \log p_i \leq -\sum_{i=1}^{k} p_i \log q_i$$

and equality if and only if,

$$p_i = q_i, \;\; i \in \{1, \ldots, k\}$$

2 Adjoint and Augmented Dynamics Derivations

We will now expand upon the derivations given in Neural Ordinary Differential Equations paper [1] to include details that were omitted from the original paper.

2.1 Adjoint Derivation

In this section we will prove the statement claimed in Section 5.2, that the layer derivative of $\boldsymbol{a}(t)$ is given by,

$$\frac{d\boldsymbol{a}^T(t)}{dt} = -\boldsymbol{a}^T(t)\frac{\partial \boldsymbol{f}(\boldsymbol{z}(t), t, \theta)}{\partial \boldsymbol{z}(t)}. \tag{A.1}$$

It is important to remember that t is the layer depth and is now continuous.

We start by drawing parallels from a general (discrete) neural network. We use the fact that the gradient of the loss function with respect to the hidden layer $\boldsymbol{z}_t$, can be expressed in terms of the next layer $\boldsymbol{z}_{t+1}$,

$$\frac{dL}{d\boldsymbol{z}_t} = \frac{dL}{d\boldsymbol{z}_{t+1}}\frac{d\boldsymbol{z}_{t+1}}{d\boldsymbol{z}_t}$$

We now generalise this approach to a neural network with continuous hidden states, by defining $\boldsymbol{z}(t+\varepsilon)$, as the following,

$$\boldsymbol{z}(t+\varepsilon) = \int_t^{t+\varepsilon} \boldsymbol{f}(\boldsymbol{z}(t), t, \theta)dt + \boldsymbol{z}(t) =: T_\varepsilon(\boldsymbol{z}(t), t), \quad \varepsilon > 0.$$

Hence, by applying the chain rule to $\boldsymbol{z}(t+\varepsilon)$, we are able to express the gradient of the loss function with respect to the continuous hidden state $\boldsymbol{z}(t)$, in terms of gradients that include, $\boldsymbol{z}(t+\varepsilon)$,

$$\frac{dL^T}{d\boldsymbol{z}(t)} = \frac{dL^T}{d\boldsymbol{z}(t+\varepsilon)}\frac{d\boldsymbol{z}(t+\varepsilon)}{d\boldsymbol{z}(t)} \quad \text{or} \quad \boldsymbol{a}^T(t) = \boldsymbol{a}^T(t+\varepsilon)\frac{\partial T_\varepsilon(\boldsymbol{z}(t),t)}{\partial \boldsymbol{z}(t)} \tag{A.2}$$

Thus, we have the required machinery to prove (A.1),

$$\begin{aligned}
\frac{d\boldsymbol{a}^T(t)}{dt} &= \lim_{\varepsilon\to 0^+} \frac{\boldsymbol{a}^T(t+\varepsilon) - \boldsymbol{a}(t)^T}{\varepsilon} && \text{(Derivative definition)} \\
&= \lim_{\varepsilon\to 0^+} \frac{\boldsymbol{a}^T(t+\varepsilon) - \boldsymbol{a}^T(t+\varepsilon)\frac{\partial}{\partial \boldsymbol{z}(t)}T_\varepsilon(\boldsymbol{z}(t))}{\varepsilon} && \text{(using (A.2))} \\
&\text{Now by expanding } T_\varepsilon(\boldsymbol{z}(t)) \text{ around } \boldsymbol{z}(t), \text{ we get,} \\
&= \lim_{\varepsilon\to 0^+} \frac{\boldsymbol{a}^T(t+\varepsilon) - \boldsymbol{a}^T(t+\varepsilon)\frac{\partial}{\partial \boldsymbol{z}(t)}(\boldsymbol{z}(t) + \varepsilon \boldsymbol{f}(\boldsymbol{z}(t),t,\theta) + \mathcal{O}(\varepsilon^2))}{\varepsilon} \\
&= \lim_{\varepsilon\to 0^+} \frac{\boldsymbol{a}^T(t+\varepsilon) - \boldsymbol{a}^T(t+\varepsilon)\left(I + \varepsilon\frac{\partial \boldsymbol{f}(\boldsymbol{z}(t),t,\theta)}{\partial \boldsymbol{z}(t)} + \mathcal{O}(\varepsilon^2)\right)}{\varepsilon} && \text{(Applying the derivative)} \\
&= \lim_{\varepsilon\to 0^+} \frac{-\varepsilon \boldsymbol{a}^T(t+\varepsilon)\frac{\partial \boldsymbol{f}(\boldsymbol{z}(t),t,\theta)}{\partial \boldsymbol{z}(t)} + \mathcal{O}(\varepsilon^2)}{\varepsilon} && \text{(Expanding bracket)} \\
&= \lim_{\varepsilon\to 0^+} -\boldsymbol{a}^T(t+\varepsilon)\frac{\partial \boldsymbol{f}(\boldsymbol{z}(t),t,\theta)}{\partial \boldsymbol{z}(t)} + \mathcal{O}(\varepsilon) && \text{(Cancelling terms)} \\
&= -\boldsymbol{a}^T(t)\frac{\partial \boldsymbol{f}(\boldsymbol{z}(t),t,\theta)}{\partial \boldsymbol{z}(t)} && \text{(Taking limit)}
\end{aligned}$$

Hence, we have arrived at what we set out to prove, that the layer derivative of $\boldsymbol{a}(t)$ is given by,

$$\frac{d\boldsymbol{a}^T(t)}{dt} = -\boldsymbol{a}^T(t)\frac{\partial \boldsymbol{f}(\boldsymbol{z}(t),t,\theta)}{\partial \boldsymbol{z}(t)}$$

2.2 Augmented Dynamics

By assuming that the model parameters $\boldsymbol{\theta}$ is layer invariant, we are able to generalise (28) to obtain the gradient of L with respect to $\boldsymbol{\theta}$ and t respectively.

To achieve this we introduce the augmented state dynamics for our system, defined as the following,

$$\frac{d\boldsymbol{z}_{\text{aug}}(t)}{dt} = \frac{d}{dt}\begin{bmatrix}\boldsymbol{z}\\ \theta \\ t\end{bmatrix}(t) = \boldsymbol{f}_{\text{aug}}([\boldsymbol{z},\theta,t]) := \begin{bmatrix}\boldsymbol{f}([\boldsymbol{z},\theta,t])\\ \boldsymbol{0} \\ 1\end{bmatrix} \tag{A.3}$$

$$\boldsymbol{a}_{\text{aug}}(t) := \begin{bmatrix}\boldsymbol{a}\\ \boldsymbol{a}_\theta \\ \boldsymbol{a}_t\end{bmatrix}(t), \quad \boldsymbol{a}_\theta(t) := \frac{dL}{d\theta(t)}, \quad \boldsymbol{a}_t(t) := \frac{dL}{dt(t)} \tag{A.4}$$

$$\frac{\partial \boldsymbol{f}_{\text{aug}}}{\partial[\boldsymbol{z}, \theta, t]} = \begin{bmatrix} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}} & \frac{\partial \boldsymbol{f}}{\partial \theta} & \frac{\partial \boldsymbol{f}}{\partial t} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}(t) \tag{A.5}$$

Where each $\mathbf{0}$ is a matrix of zeroes with the appropriate dimensions.

Therefore, by applying (A.1) to our augmented system, we can understand how the augmented adjoint changes with respect to the layer,

$$\begin{aligned} \frac{d\boldsymbol{a}_{\text{aug}}^T(t)}{dt} &= -\begin{bmatrix} \boldsymbol{a}^T(t) & \boldsymbol{a}_\theta^T(t) & \boldsymbol{a}_t^T(t) \end{bmatrix} \frac{\partial \boldsymbol{f}_{\text{aug}}}{\partial[\boldsymbol{z}, \theta, t]}(t) \\ &= -\begin{bmatrix} \boldsymbol{a}^T \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}} & \boldsymbol{a}^T \frac{\partial \boldsymbol{f}}{\partial \theta} & \boldsymbol{a}^T \frac{\partial \boldsymbol{f}}{\partial t} \end{bmatrix}(t) \end{aligned} \tag{A.6}$$

Thus, by looking at the second element of equation (A.6), we get the following relationship,

$$\frac{d\boldsymbol{a}_\theta^T(t)}{dt} = \frac{d}{dt}\left(\frac{dL^T}{d\theta}\right) = \boldsymbol{a}^T(t)\frac{\partial \boldsymbol{f}}{\partial \theta}$$

Therefore, by integrating backwards over the layers and setting the adjoint of θ at time t_1 to zero as we have not "accumulated" any error yet. We therefore obtain the overall sensitivity of the model parameters,

$$\implies \frac{dL^T}{d\theta} = \boldsymbol{a}_\theta^T(t_0) = -\int_{t_1}^{t_0} \boldsymbol{a}^T(t)\frac{\partial \boldsymbol{f}(\boldsymbol{z}(t), \theta, t)}{\partial \theta} dt$$

Finally, we can work out expressions for the derivative of L with respect to the times t_0 and t_1, respectively. To derive the expression at t_1, we apply the chain rule to get,

$$\frac{\partial L^T}{\partial t_N} = \frac{\partial L^T}{\partial \boldsymbol{z}(t)}\frac{d\boldsymbol{z}(t)}{dt}\Big|_{t=t_N} = \boldsymbol{a}^T(t_N)\boldsymbol{f}(\boldsymbol{z}(t_N), \theta, t) \tag{A.7}$$

Hence, by using the final term in equation (A.6), and integrating backwards over the layers we get,

$$\implies \frac{\partial L^T}{\partial t_0} = \boldsymbol{a}_t^T(t_0) = \boldsymbol{a}_t^T(t_1) - \int_{t_1}^{t_0} \boldsymbol{a}^T(t)\frac{\partial \boldsymbol{f}(\boldsymbol{z}(t), \theta, t)}{\partial t} dt \tag{A.8}$$

Up to know we have only considered our loss function L depends between t_0 and t_1. However, if we wanted to evaluate the ODE at multiple time observations $t_0, \ldots, t_N$, we would adapt the above process by iterating the process over the intervals, $[t_{N-1}, t_N], [t_{N-2}, t_{N-1}], \ldots$, accumulating the corrections for each interval.

Bibliography

[1] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583, 2018.

[2] Pierre Bremaud. *An Introduction to Probabilistic Modeling*. Undergraduate Texts in Mathematics. Springer New York : Imprint: Springer, New York, NY, 1st ed. 1988. edition, 1988.

[3] Kirkpatrick Darrah. MMath Code. `https://github.com/DarrahK/MMath-Code`, 2021.

[4] Dr. Jonathan Bartlett. MA20226: Statistics 2A Lecture Notes. *University of Bath*, January 2019.

[5] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre GR Day, Clint Richardson, Charles K Fisher, and David J Schwab. A high-bias, low-variance introduction to machine learning for physicists. *Physics reports*, 810:1–124, 2019.

[6] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature (London)*, 323(6088):533–536, 1986.

[7] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, 12 1989.

[8] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014.

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