Adrian Spataru, BSc

Recurrent Attentive Neural Processes Clustering for Unsupervised Representation Learning

Master's Thesis

to achieve the university degree of

Master of Science

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Dr. Roman Kern

Institute for Interactive Systems and Data Science
Head: Univ.-Prof. Dipl-Inf. Dr. Stefanie Lindstaedt

Graz, February 2020

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date	Signature

Abstract

Neural Processes can be trained efficiently and can learn to adapt their priors to data during inference. They have shown potential in various applications. However, their most significant drawback is that they underfit the data. In this thesis, we propose a method called Recurrent Attentive Neural Processes (RANP), which build in top of Recurrent Neural Processes (RNP) to improve the fitness of data.

Furthermore, there is a lack of research into interpretability in Neural Processes. In this thesis, we do the first step into that direction and propose an extension of RANP called Recurrent Attentive Neural Processes Clustering (RANP-Clust), which improves clusterability of multivariate time series over RANP and RNP in various of scenarios. Finally, we show that RANP has shown an improvement of 19% over RNP on the UCI household electricity consumption dataset.

Contents

Contents

List of Figures

1 Introduction

In this chapter, we provide motivation and related work of this thesis. Regarding related work, we start by giving an overview of time series clustering, followed by representation learning, and then talk about representation learning for time series. Finally, we will define the objectives of this thesis and provide a brief description of the outline of the thesis.

1.1 Motivation

In the age of big data and the need to process a higher quantity of data than ever before, it becomes increasingly harder to keep a clear understanding of such data.

For the past ten years, we have seen an increase in available data [26]. One type of these data sources is time-series data. Time series data can range from sales figures to sensors in a self-driving car. Most applications focus on forecasting of time series data. Forecasting problems can range from forecasting sales for next quarter to predict car accidents. Time series forecasting is used in many industries. One of these industries is manufacturing. With the rise of IoT and Deep Learning, this has pushed the latest trend called Industry 4.0 [43]. One of the main ideas is to allow for data-driven manufacturing and making manufacturing ”smarter”. Forecasting techniques play a significant role in this. Time to live in milling or forecasting temperature in equipment is such an example. This leads to condition monitoring, where not only we want to forecast the future behavior of machines, but understand in what state is currently situated. Deep Learning methods have shown state of the art performance in time series forecasting [37], but struggle with interpretability since they are black-box algorithms [38].

Interpretability of Deep Learning is a new challenge for researchers. In time series, we want to understand the reasons behind a forecast by a neural network. In this thesis, we present two models that may help in solving this problem. More

specifically, we present a model that can forecast future behavior and creates a latent representation that eases interpretability by implicitly grouping.

1.2 Time Series Clustering

Given a set of multivariate sequences $X = x_1, x_2, ..x_n, x \in \mathbf{R}^{n \times m}$ where n is the number of sequences and m the length of sequences, segregate each value in X into discrete groups [77], based on trait or similarity measure.

Methods like k-means [1] and agglomerative hierarchical clustering [10] are some well-known clustering algorithms. These methods, however, cannot model time-dependencies. One can model time series data in k-means, by extracting time-independent features. Such features could be the max value of a time-series, the number of points over a certain threshold or parameters of a Fourier transformation [55]. This approach has been used successfully in several use-cases. However, determining which features should be generated is not straight forward, and domain knowledge is needed to handcraft these features. Fortunately, other methods are available.

A classical machine learning approach is clustering with Dynamic Time Warping (DTW) [5]. DTW is a similarity measure for comparing two time-series. Another advantage of this method is that it can be extended to multivariate time series, where the time series length does not have to be equal. By replacing the Euclidean distance used in k-means with DTW [49], one can typically expect better results. Furthermore, in a classification setting, DTW with k-nearest neighbours [28] is comparable in performance or even outperforms more complex models. Nevertheless, DTW has a time complexity of $O(n^2)$, making it hard to scale on a large dataset. Segmented Dynamic Time Warping [29] approximates DTW and offers a scalable solution with minimal loss in accuracy. A similar model called K-shapes [44] which uses cross-correlation between the time series as a distance measure provides competitive results.

Symbolic Aggregate Approximation [35] (SAX) is a method for discretizing time series. It takes a subset of a time series as input and bins it based on the number range. This would build a sequence of discrete values, which one can cluster hierarchically. Furthermore, with this abstract representation, forecasting becomes simpler, as one has only to predict a sequence of letters. This is also strongly

linked to representation learning, which will be discussed in the next section.

More advanced methods treat time series as a graph. For example, Toeplitz inverse covariance-based clustering [21] (TICC) creates from time series windows a correlation graph. The graph is modelled over time with a Markov random field model and can be also be used to cluster the time series. For high-dimensional data, dimensionality reduction is needed. In our thesis, we will focus on autoencoder-based methods, but other methods have shown great results.

Functional Principal Component Analysis (FPCA) [65], applies Principal Component Analysis over time, which is a linear dimensionality reduction method. FPCA is one of the methods used in Functional Data Analysis and is part of methods which can be used in Representation Learning. However, if the data exhibit complex behaviour(e.g nonlinearity) which cannot be easily reduced with FPCA, then non-linear dimensionality reduction methods are needed.

1.3 Representation Learning

As the dimensionality and complexity of the data increases, the need for more complex models is needed. Moreover, this increase in complexity makes the task of a building model for clustering, outlier detection and classification harder [4]. However, if we can extract useful features of our ever-increasing complex data, we may solve problems with simple models. This is the main idea of Representation Learning. It is learning representations of the data that make it easier to extract useful information when building classifiers or other predictors [4]. Representation Learning has been successfully applied in Speech Recognition, Signal Processing, Object Recognition and Natural Language Processing. Furthermore, Representation Learning is a quintessential method used in Multi-Task Learning, Transfer Learning and Domain Adaptation.

We understand now what representation learning is, but what is a useful representation? How does one determine, if a representation is better than another representation? More specifically, what properties do good representation exert. Bengio provides examples of such properties, here we list some of them:

- **Smoothness**: given a function f which create a representation of a given input and $x \approx y$, then $f(x) \approx f(y)$.

- **Natural Clustering**: Given a set of inputs from different categories, then the representations of every category are tightly clustered with minimal overlap.
- **Multiple explanatory factors**: Different underlying factors generate data. The representation should be able to disentangle these factors.
- **Shared factors across tasks**: A representation should be able to extract factors that are shared in a variety of tasks.

In this thesis, we will focus on deep learning methods for representation learning. Variational Autoencoder (VAE) [18] are generative autoencoder based models. Variational Autoencoders can identify and separate the underlying factors of the data.Beta-VAE [23] further pushes the idea and forces the VAE to represent every factor independently in the representation vector.

Most data clustering methods use either distance or dissimilarity to distinguish the classes. As the dimensionality of the data increases, the need for finding good feature representation increases. One concept is too jointly train an autoencoder with some clustering loss. One such approach is DEC [69], which is a 2 step algorithm. In the first step, the autoencoder is trained regularly. Then based on the latent space, we determine a set of k cluster centers and train the autoencoder further with a clustering loss which uses KL-Divergence.

Variational Deep Embedding (VADE) [25] extends VAE and introduces an additional clustering loss, which models a Gaussian Mixture Model (GMM) in the latent space. A disadvantage of GMMs is that the number of clusters needs to be given. Non-Parametric models such as Dirichlet Processes do not need this prior [60]. AdapVAE [74] is a VAE where we model a Dirichlet process in the latent space. Other approaches try to discretize the latent space. One such approach is the Vector Quantized VAE (VQ-VAE) [42], which maps every latent value to a discrete vector. This vector is then passed to the decoder. VQ-VAE 2 [52] model proposed by Deepmind, buildings a hierarchical VQ-VAE, which allows creating a hierarchy of factors. In 1.1 we can see the effect of this hierarchical setup.

In the paper, they showed competitive results with Generative Adversarial Networks (GAN) [19] in image generation tasks.

GANs are the start of the art approach in image generation and representation learning in images. One notable model is StyleGan [27], which performs realistic images and can model a big range of factors such as age, colour, glasses, hairstyle.

Figure 1.1: Reconstructions [52] from a VQ-VAE 2 with three latent maps (top, middle, bottom). The rightmost image is the original. Each latent map adds extra detail to the reconstruction.

1.4 Time Series Representation Learning

In the previous section, we explored methods like VAE, which are successfully used in a range of problems. In their classical setup, VAE uses feed-forward neural networks to encode the data. An issue is that Feed Forward Neural Networks cannot encode sequences since they do not have an understanding of time-dependency between variables. (However, they can model time-independent features, which we can extract from time series) In this section, we will explore methods used to encode time series data and how they are used in combination with Representation Learning methods.

In 1989, Pearlmutter [46] introduced the idea of recurrence in a neural network. Recurrence allowed a neural network to use past encoded information when processing new data (more on the next chapter). 8 years later Hochreiter [24] introduced us to **Long Short Term Memory** (LSTM). A model which improves the limitations of the previous model and used in a wide variety of problems. [20],[70],[71] Since then researchers develop models which improve performance or efficiency of LSTMs. Highway Networks [56] is a model which builds on top LSTM by introducing ”information highways” which allows building very deep networks while retaining information flow. Gated Recurrent Units [7] simplify LSTM architecture and achieve comparable or even outperform LSTM in certain use cases. FastGRNN [32] further push the size down of these model with a fingerprint 3-4 times smaller of LSTM and GRU while keeping the performance similar.

In 2012, Krizhevsky introduced a deep convolutional network called AlexNet, which for the time was the start of the art in image classification. Since then

many new architecture have appeared such as Resnet [22] and GoogleNet [58]. Convolution Neural Networks (CNN) have been successfully in Computer Vision [66], which begs the question, do they work with time-series data? CNN's can capture spatial features from the data. Such properties in time series could be seasonality which CNNs may capture. This leads the usage of CNN in time series [75]. Furthermore combining CNN with LSTM have shown to outperform normal LSTM [73]. In 2016 Google's Deepmind Wavenet model, proposed a new generative model for raw audio [62]. A method which uses diluted casual convolution which allows processing long sequences more efficiently.

When it comes to representation learning of sequence data, combining methods from Representation Learning with models, which can encode sequential data have shown great results. For example, creating an autoencoder where LSTMs are used for encoding and decoding the sequence. Such a model called seq2seq [57] has shown promising results in neural machine translation tasks. By passing different inputs into the same RNN [48], we can determine similarity in the latent space of them. One can extend the idea of VQ-VAE [42]for time series and represent a self-organizing map in the latent space as in SOM-VAE [15]. This enables us to see the time-series over time in a low dimensionality space and has shown to improve interpretability in ECG Medical data [15]. An architecture, which takes the ideas of DEC and adapting it for time series is the Recurrent Deep Divergence-based Clustering (RDDC) [61]. Here the author proposes to encode time series in an RNN and in the embedding which is given from the RNN add a divergence-based clustering loss function.

All previous method uses some kind of autoencoder setup. In [16], they proposed an encoder setup only, by using an exponentially dilated causal convolutions encoder and a triplet loss adapted for time series. Finally, there has been research in loss function for time series. Most of the presented methods use Mean Square Error (MSE) for calculating the reconstruction error. However, methods like DTW are better suited for time-series data. One way to incorporate these methods is by making them differentiable for the neural network. Soft-DTW [8] is such differentiable loss function and has shown to outperform MSE as a loss function.

1.5 Contribution

In this thesis, following contributions have been made:

- We propose a new neural processes architecture called Recurrent Attentive Neural Processes (RANP), which outperforms Recurrent Neural Processes in regression tasks.
- We purpose an extension for RANP, called Recurrent Attentive Neural Processes Clustering (RANP-Clust) which allows the model to cluster time series as a byproduct.
- We show that l2-normalization can improve clusterability in neural processes.

1.6 Thesis Outline

In this section, we will describe the thesis outline. In chapter 2 Background Theory, we will describe all the ”building blocks” used in our proposed method. In chapter 3, we will present our proposed method called Recurrent Attentive Neural Processes and an extension to this model for clustering called Recurrent Attentive Neural Processes Clustering. In chapter 4, we will present our experiments done on the model. We will start by describing the datasets used in the experiments, followed by details implementations of the model. Finally, we present and discuss the results. We conclude the thesis with chapter 5, where we give the conclusion and future work.

2 Background Theory

In this chapter, we explore the building blocks used in our proposed method. We will first present the basic feed forward neural networks architecture and present its properties that are relevant to representation learning. Moreover, we will present the different architectures of neural networks with their respective properties, which we integrated or used as inspiration for our method.

2.1 Feed Forward Neural Networks

In this section we present neural networks and provide the general theory used in this thesis.

2.1.1 The perceptron

Artificial neural networks were created to mimic the biological neural network present in human brains. The human brain process data using a vast network of small units called neurons. One of the most popular models for representing neurons in the perceptron [53].

A single neuron is represented by one perceptron. The perceptron is a function, which takes several input $X = x_1, x_2, ...x_N, x \in R^1$, where N is the number of inputs and return and output y. The function is a weighted sum of the inputs, which is then passed to the activation function. The output of the activation function is the output y. The activation function can be freely selected. In the original paper, they use a step function that outputs either 0 or 1. One should use an activation which is differential and suits their task. For example, the sigmoid function can be used for classification. The sigmoid function is defined as:

$$f(x) = \frac{1}{1 + \exp(-x)}$$

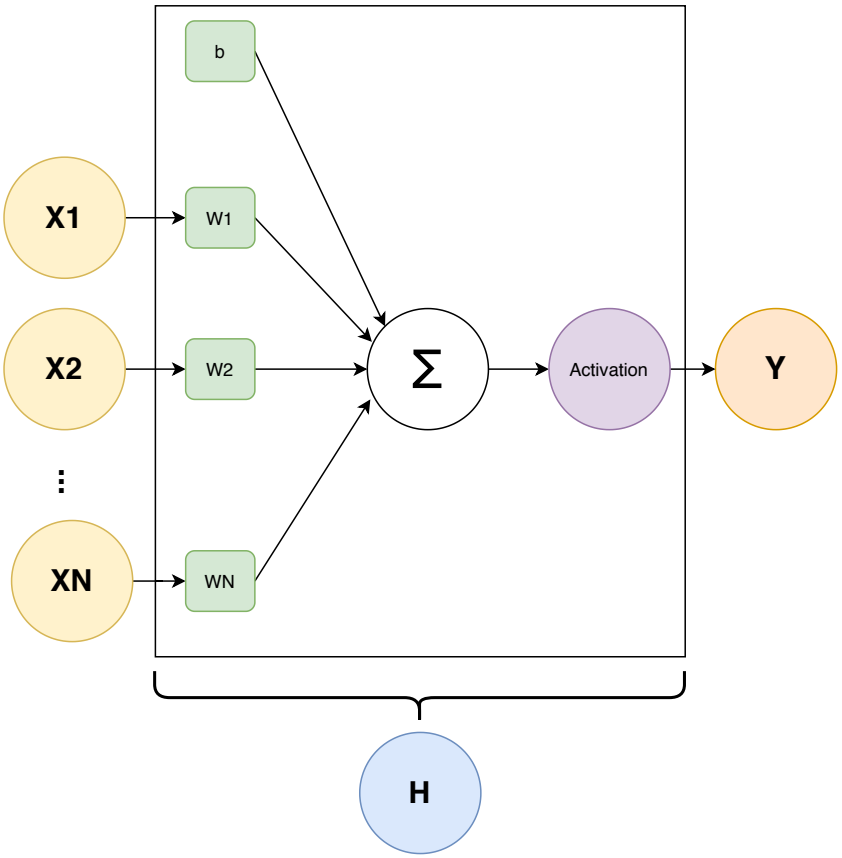


Figure 2.1: The perceptron has the input $X \in R^n$ is multiplied with the weights $W \in R^n$ and added with the bias $b \in R^1$. The result of this multiplication is passed to the activation function, which gives the output $y \in R^1$. We will represent the perceptron in future figures as H

Furthermore a perceptron which uses sigmoid [40] as the activation function is equivalent to logistic regression [68]. In 2.1, we can see a visual representation of the Perceptron. A perceptron with a sigmoid activation function is then defined as:

$$y = f(XW^T + b)$$

Where $f(x)$ is the sigmoid activation function, W the weights and b the bias. For regression, no activation function can be used. This is equivalent to the linear regression model [18]. We will discuss later on other activation functions that are used in the thesis.

2.1.2 Training Neural Networks

Until this point, we have observed how neural networks can be defined. In this section, we will explore how neural networks can be learnt. To produce the desired output y, we must find the correct weights W and bias b. To achieve this, one may define a cost function. If we want to build a regression model, we could minimize the mean square error (MSE) [18] between the true output and the prediction of the model. More exactly:

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$
$$\hat{y}_i = f(x_i; \theta)$$

Where y is the true value, $\hat{y}$ is the prediction, $\theta = [W; b]$ are all the parameters of the model $f(x; \theta)$ with x as the input. By providing the neural network a set of inputs x and their true output value y, we can evaluate the error of the model. With these elements, we can then train the neural network to minimize this error. More exactly, we want to find a set of parameters θ, which has the lowest error for the given cost function. By using gradient-based learning methods, such as stochastic gradient descent [18], we can find a set of parameters θ:

$$\theta_j := \theta_j - \eta \frac{\partial}{\partial \theta_j} J(\theta)$$

Where every parameter in θ is updated simultaneously, and η is the learning rate. Since most of the time, the cost function is non-convex, a global minimum is not guaranteed.

2.1.3 Multilayer Perceptrons

A single perceptron can learn any linear function. However, in real life, most data is often non-linear. Multilayer Perceptrons (MLP) [41] solve this issue by creating an acyclic graph of neurons. In 2.2, we can observe how this may look like. The input is passed to multiple perceptrons. This layer of the perceptron is called the hidden layer. In 2.2, we illustrated an example for one layer. However, one can have more than one hidden layer. The outputs of the hidden layer are passed to an output layer where the output y is calculated. The number of perceptrons in the output layer usually is equal to the number of output y.

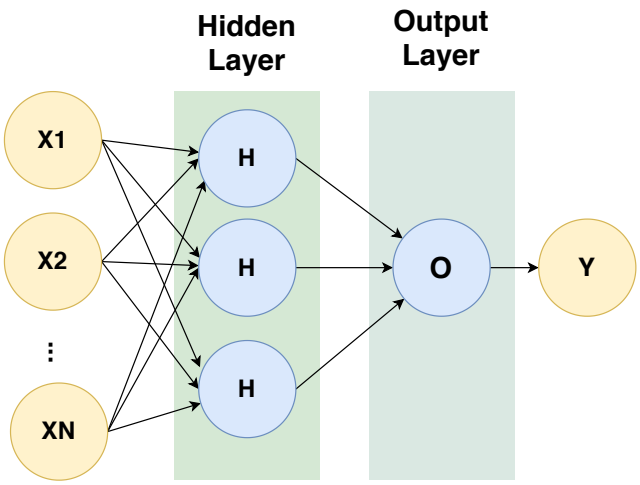


Figure 2.2: The MLP passes the input $X \in R^n$ to a layer of perceptron H. The outputs of the layer are passed to a perceptron which is part of the output layer. The result of this layer is the output y

2.2 Autoencoder

Real-life data is often highly dimensional, and we can use a feed-forward neural network to create a low dimensional representation of our data. Autoencoders [18] is a type of neural network which attempts to reconstruct its input on its output. The network consists of 2 parts. The encoder function $h = f(x)$ and a decoder function $x' = g(f(x))$ where $x, x' \in R^n$ and $h \in R^m$. As the cost function, we want to minimize the dissimilarity between input and output. This cost function is also called reconstruction error. One example of such reconstruction error would be the MSE. In 2.3, we can see how such network might look like.

To get a representation of our data, we make the size of the output of our encoder smaller than the input, hence $m < n$. This forces the network to compress the information in a way, while still being able to reproduce the original input. In this way, we hope to encode the most important information, which is representative of the input. We call this the bottleneck layer. In case $m \geq n$, then we risk learning the identity function. In other words, we learn to copy the input. Fortunately, methods like Denoising Autoencoder [64] and regularized autoencoder [18] help in preventing learning the identity function. Our proposed model uses the idea of autoencoder to generate low dimensional representations.

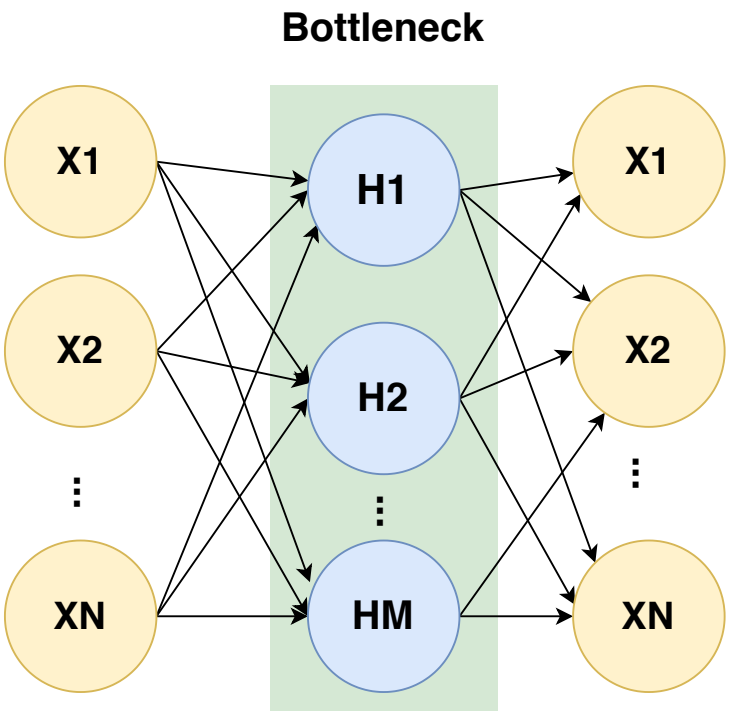


Figure 2.3: Visual representation of an autoencoder

2.3 Recurrent Neural Network

In our thesis, our focus is on time series data, which, by nature, is a sequence of data points. Feedforward neural networks fail to model such data, as future points are dependent on previous points. This is because each data point passed to a feed-forward network is dealt with independently. Recurrent Neural Networks (RNN) [18] solve this issue by introducing a recurrent connection from the output of the hidden layer to the beginning of the hidden layer. This allows the network to use information from previous data.

In 2.4, we observe how this recursion looks like when it is unfolded. X is the input data, S the hidden state of the network, and O is the output. We can see that the data X_1, X_2, X_3 is passed sequentially to the model. For the first input, X_1, the network acts like a normal feed-forward network, where the state S_1 is calculated based on the given input X_1. However, for the next input X_2, we observe that S_2 uses X_1 and S_1. This results in a state S_2, which in theory may encode all

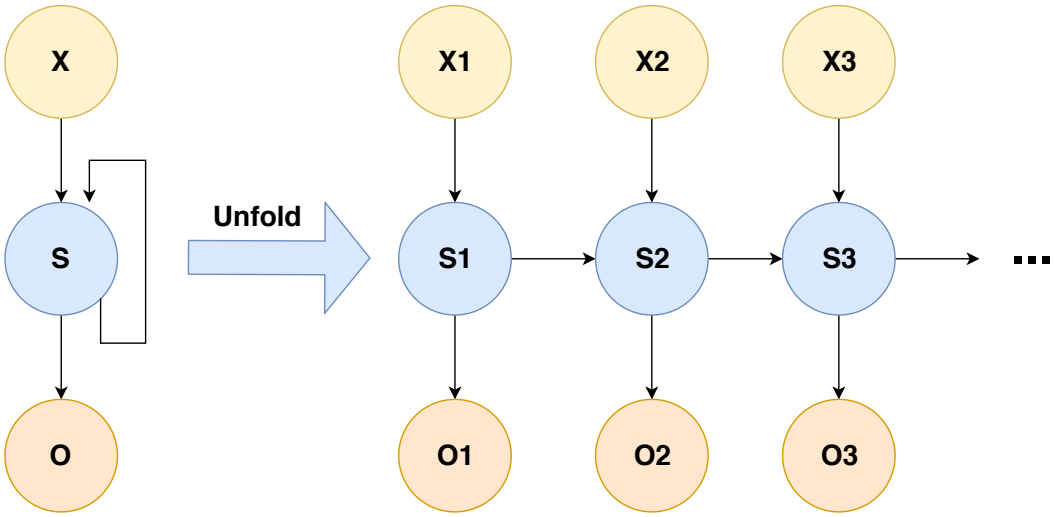


Figure 2.4: A visual representation of a Recurrent Neural Network in the folded and unfolded state.

past inputs (in this example only X_1) and current input. The RNN is defined as the following:

$$a_t = W^a x_t + R^a h_{t-1} + b^a$$

$$h_t = tanh(a_t)$$

$$o_t = W^o h_t + b^o$$

$$\hat{y}^t = softmax(o_t)$$

The neural network starts with an initial hidden layer, h_0. For hidden layers at step t, we apply the $tanh$ activation function to the weighted sum between the input at step x_t and the previously hidden layer h_{t-1}, where W^a and R^a are the weights and b^a the bias. The output is then calculated based on the hidden layer with the weights W^o and the bias b^o. Optionally, the output values o_t are then normalized with the $softmax$ activation function.

2.4 Long Short Term Memory

A problem with RNNs is the inability to model long-term dependencies, mostly due to the vanishing gradient problem. In a nutshell, due to the recurrence, the gradient

from earlier states is diluted up to zero. This results in losing information, which is needed for modeling long term dependencies [24].

A solution to the vanishing gradient problem is Long Short Term Memory (LSTM) [24]. LSTMs are the type of RNNs that introduce gates in the hidden state. These gates allow the model to control the information flow passed in the network. In order to model long-term dependencies, LSTMs introduce cell states. The cell state flows from time step to time step unchanged unless the gates are triggered. The networks input is the same as RNNs, where both input and previous hidden state are passed into a feed-forward network:

$$z_t = tanh(W^z x_t + R^z h_{t-1} + b^z)$$

The LSTM has 3 gates:

- **Input Gate** - Controls the amount of input passed to the cell.
- **Forget Gate** - Controls the amount of information the cell retains from previous time step. This allows the model to $forget$ unneeded information.
- **Output Gate** - Controls the amount of information that would flow out of the LSTM unit.

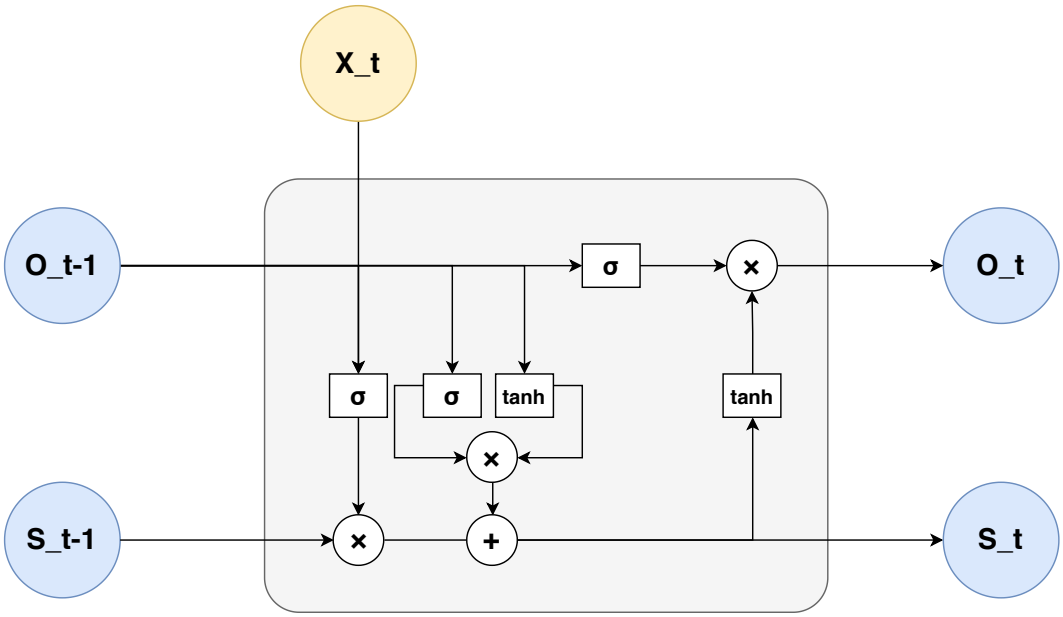


Figure 2.5: A visual representation of a LSTM unit.

The gates are computed by the weighted sum of x_t and h_{t-1}. This sum is passed to a sigmoid activation function. The sigmoid activation returns a value between 0 and 1. A value of 0 means no input to flow, where 1 means all the input to flow. The input gate i_t, forget gate f_t and output gate o_t are then defined as:

$$i_t = \sigma(W^i x_t + R^i h_{t-1} + b^i)$$

$$f_t = \sigma(W^f x_t + R^f h_{t-1} + b^f)$$

$$o_t = \sigma(W^o x_t + R^o h_{t-1} + b^o)$$

The cell state s_t is calculated based on the previous cell state with the addition of the information passed through the gates:

$$s_t = z_t i_t + s_{t-1} f_t$$

The output of the LSTM unit is calculated by passing the cell state s_t to a $tanh$ activation function, which then multiplied with the output gate.

$$h_t = tanh(s_t) o_t$$

2.5 Attention Mechanism

I am born in Germany. Therefore, I speak? French? Mandarin? The (potential) correct answer is German. However, how did you determine this? Because you saw the word Germany and born, you inferences the language. What about the rest of the words in the sentence: therefore, am, in? They are irrelevant to answering the question.

This is the main idea of attention mechanism and has been initially used successfully in Neural Machine Translation. [63] [11]

A lot of applications and attention mechanisms have been introduced since then. In this thesis, we limit ourselves to Multi-head Attention [63], which have achieved SOTA results in various domains.[9] [72] Furthermore, we will also use DotProduct Attention [63], which is a primary component used in multi-head attention. Attention uses keys, values, and query. The keys and query determine the relevance of the query to the values, mostly through their distance. These then weight the values.

2.5.1 Scaled Dot Product Attention

The scaled dot product attention uses dot multiplication of the key K and query Q to determine the relevancy of the query to the values V. To stabilize the gradients, this dot product is scaled with d_k. The result is then passed to a softmax to normalize the values. The Scaled Dot Product is defined as:

$$DotProduct(Q, K, V) := softmax(QK^T/\sqrt{d_k})V$$

2.5.2 Multi-head Attention

Multi-head Attention is a fundamental unit used in state of the art Neural Machine Translation methods. This method works by attending different positions in the data passed and concatenating the output of each view. Multi-head Attention is defined as:

$$MultiHead(Q, K, V) := concat(head_1, ...head_H)W \in R^{mxd_v}$$

$$head_h := DotProduct(QW_h^Q, KW_h^K, VW_h^V) \in R^{mxd_v}$$

2.6 Neural Processes

Deep Neural networks can learn a single function parameterized by weights and biases. The result is deterministic, leading to models that cannot express uncertainty of their predictions. A potential model that can express this uncertainty is Gaussian Processes (GP) [51]. Furthermore, GPs can express complex functions with a little amount of data by using different kernel functions in the prior, contrary to neural networks, where much data is required. However, GPs are computationally expensive with time complexity of $O(n^3)$, therefore making them hard to scale on big datasets.

On the other hand, Neural Processes [17](which are a generalization of generative query networks [13]) is a Neural Network which can also model a distribution over functions. NPs learn the prior directly from the data, instead of using kernel functions as in GPs. More precisely, NPs estimate the distribution by the output value of a target set of points, conditioned on some input and context set of points.

The objective function is defined as:

$$J = E_{q(z|x_c,y_c)}\left[\sum_{i=1}^{t} \log p(y_i|z,x_i) + \log \frac{q(z|x_t,y_t)}{q(z|x_c,y_c)}\right]$$

$$J = E_{q(z|x_c,y_c)}\left[\sum_{i=1}^{t} \log p(y_i|z,x_i)\right] - KL(q(z|x_c,y_c)||q(z|x_t,y_t))$$

where z is the latent variable, x_c, y_c the context points, x_t, y_t the target points and $p(\cdot)$ and $q(\cdot)$ are probability distributions. The idea is that given some context points x_c, y_c and unseen input x_t, we want to predict the output y_t. With $q(z|x_c, y_c)$ and $q(z|x_t, y_t)$ we approximate the posterior $p(z|x_c, y_c)$ and $p(z|x_t, y_t)$ respectively. With the given representations of the context or target points we approximate a mean and variance of a Gaussian distribution, therefore $q(z|\cdot) = N(\mu_z, \sigma_z)$. In a regression setting, $E_{q(z|x_c,y_c)}\left[\sum_{i=1}^{t} \log p(y_i|z,x_i)\right]$ can be the equivalent to $MSE(y_i, x_i)$.

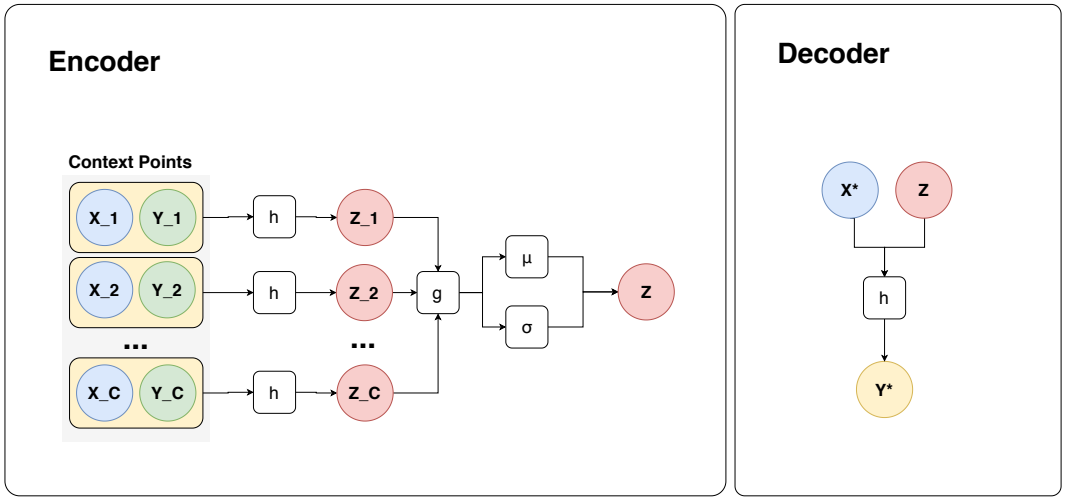


Figure 2.6: The encoder passes a set of context points separately to an MLP h. We pass every representation z_c to the aggregator g and create a latent vector z. The decoder takes the latent vector z, and a target input x^* and returns a target output y^*.

The Architecture is defined as the following:

1. **Encoder** - The encoder passes each pair of context points (x_i, x_i) into a MLP and extract the representation R_i.

2. **Aggregator** - The aggregator g takes the representation R_i and combines them into a vector r. The aggregator needs to be invariant to the number and order of the context points. The original paper suggest to use the mean of the representation as the aggregator. More exactly, $r = g(r_1, ...r_c) = \frac{1}{c}\sum_{i=1}^{c} r_i$, where c are the number of context points.
3. **Latent value** - r is then used to create a latent distribution. Similar to Variational Autoencoder, the latent vector is assumed to be a Gaussian Distribution. We are calculating with two feed-forward networks the mean μ and standard deviation σ. This can be sampled during inference and is equivalent to $q(z|x_c, y_c)$ when passing context points to the aggregator or $q(z|x_t, y_t)$ when passing target points to the aggregator.
4. **Conditional Decoder** - The decoder receives as input a concatenated vector of the target input X^* and the output Z of the latent encoder. This concatenated vector is then passed to an MLP to predict the target output Y^*.

To train the NP, context points x_c, y_c are passed into the encoder, and the latent value is calculated. Then we pass the target points x_t, y_t also into the encoder and get its latent value. We can then calculate the KL-Divergence as described in the loss with the given two latent values. Furthermore, we pass the latent value we calculated from the context points and the input x_t into the decoder and calculate y_t. In the regression setting, we would then calculate the MSE between the predicted output and actual output of y_t. As described previously, this is calculating $E_{q(z|x_c,y_c)}\left[\sum_{i=1}^{t} \log p(y_i|z, x_i)\right]$ in the loss.

For inference, we then pass only the context points x_c, y_c into the encoder, and calculate the latent value. Then we give this latent value and a set of unseen target inputs x^* and calculate the output y^*.To inference the uncertainty of our predictions from the NP, we draw Monte Carlo samples from the latent space. For example, if the uncertainty is zero, then all outputs of the drawn samples should be the same, mimicking a deterministic function.

Neural Process can also be used in a meta-learning context. For example, we can train the NP with different datasets $D = d1, d2, ..., dn$ and then do the inference to a new datasets $D* = d1*, ...dn*$, we hope that the prior learned in z, has relevant information for predicting the new dataset $D*$. This has been explored by the authors of Neural Process paper.

2.6.1 Attentive Neural Processes

Attentive Neural Processes (ANP) [30] improve on top of Neural Processes by adding attention to the aggregator. The idea is that some context points are more important than others for inferring the output. Attentive Neural Processes contains two encoders: One which is latent, equivalent to Neural Processes, and a deterministic one. The deterministic path passes each context point x_c, y_c through a encoder to get a representation r_c. These representations r_c are passed through a self-attention layer. Finally, this is passed through a cross-attention layer where the keys are the input context points x_c, and the query is the target value x_t. The idea is that the deterministic path learns the local features, and the latent state learns the global features. The idea of having two different encoders in Neural Processes has been explored and has been shown empirically that it to outperform networks with only a latent encoder.[33] In 2.7, we can see a visual representation of this model.

2.6.2 Recurrent Neural Processes

Neural Processes do not take temporal dependencies present in the data. Recurrent Neural Processes [67] extend Neural Processes to model temporal data while keeping the advantages of Neural Processes. Recurrent Neural Processes replaces the encoder and decoder of NP with a recurrent neural network. Since we pass it through the RNN, the temporal dependencies of all past context points are modeled. In 2.8, we can see a visual representation of the model.

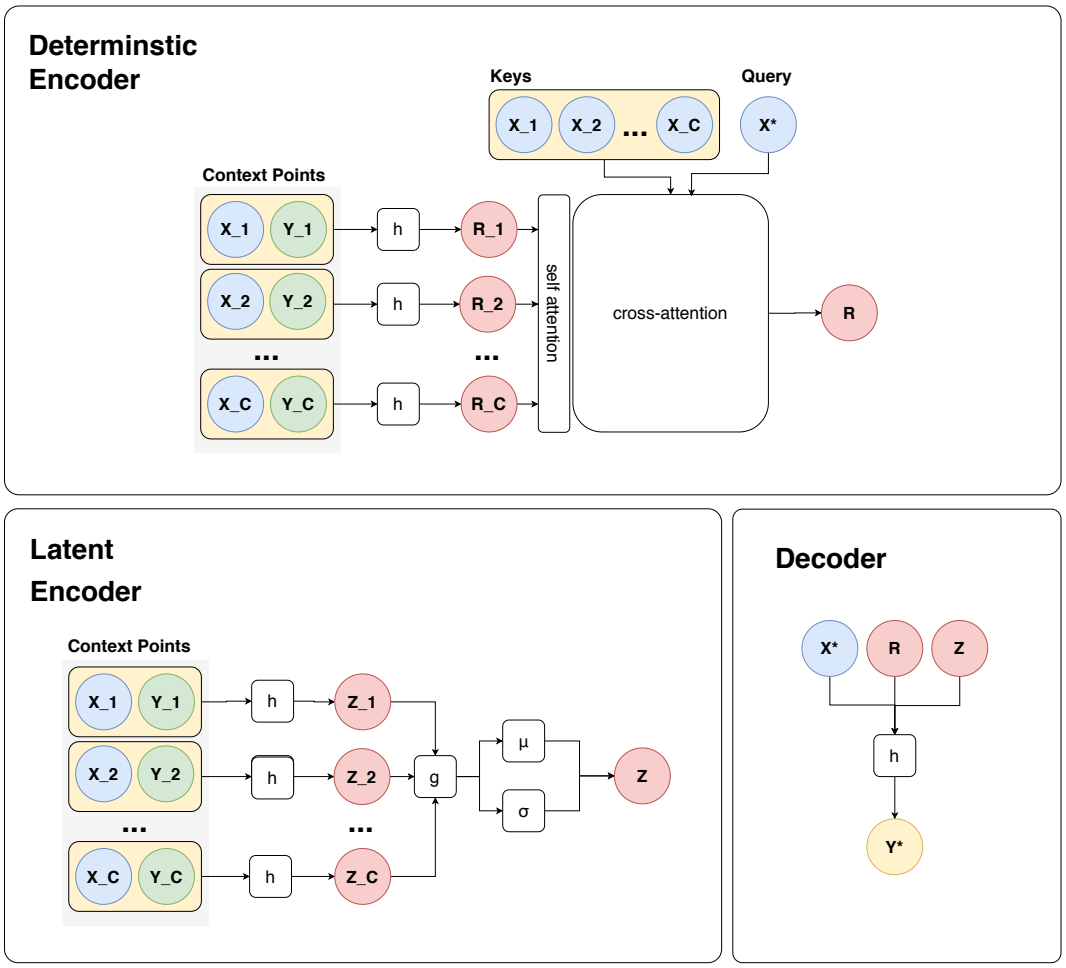


Figure 2.7: A set of context points is passed to a deterministic encoder and a latent encoder. Each encoder uses an MLP h to obtain a representation of the context points. For the latent encoder, we pass every representation z_c to the aggregator g and create a latent vector z. In the deterministic encoder, we pass every representation r_c into a self-attention layer. Afterwards, we pass the output of the self-attention layer into a cross attention layer, where the keys are the input context points x_c and query the input target point x^*. This gives us the deterministic vector R. The decoder takes the latent vector z, deterministic vector r, and a target input x^* and returns a target output y^*.

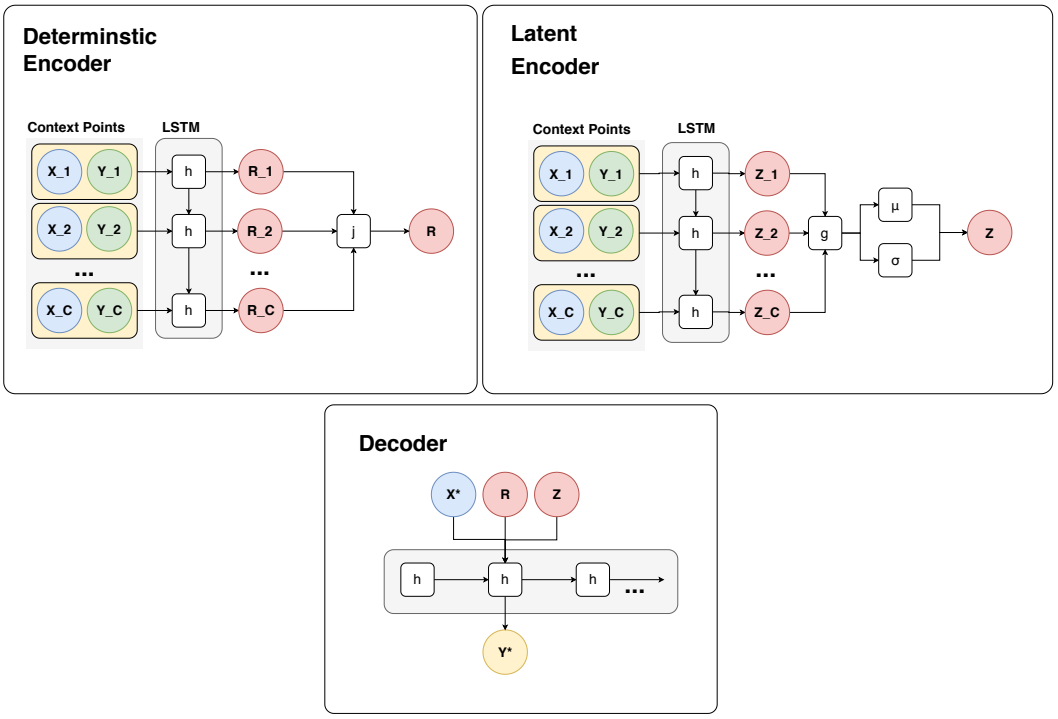


Figure 2.8: A set of context points is passed to a deterministic encoder and a latent encoder. Each encoder uses an RNN h to obtain a representation of the context points. For the latent encoder, we pass every representation z_c to the aggregator g and create a latent vector z. In the deterministic encoder, we pass every representation r_c into an aggregator j. This gives us the deterministic vector R. The decoder takes the latent vector z, deterministic vector r and a target input x^* and returns a target output y^*.

3 Proposed Approach

3.1 Recurrent Attentive Neural Processes

In this section, we present our first proposed method called Recurrent Attentive Neural Processes (RANP). During the writing of this thesis, several papers [50] [76] have proposed the same or very similar architecture. In seq2seq autoencoders, which uses RNN as the encoder and decoder, researchers have observed better performance by introducing attention [3]. Furthermore, we observed that the introduction of attention in the aggregator of the ANP, have shown improvement over NP. In RANP, we combine RNP and ANP in one model. The assumption is that certain context points are more relevant than others for predicting the target variable. Our proposed method, as in 3.1, contains the following modules:

1. **Deterministic Encoder** - The encoder passes each pair of context points (X_i, Y_i) into a recurrent neural network (LSTM in the figure) and extract the features R_i. The features extracted are concatenated and are passed to a self-attention layer. The output of the self-attention layer is passed into a cross-attention layer (can be dot product or multi-head attention) where the keys are the input context points X_i and the query the target input X^*.
2. **Latent Encoder** - The encoder passes each pair of context points (X_i, Y_i) into a recurrent neural network (LSTM in the picture) and extract the features R_i. Note that this recurrent neural network is not shared with the deterministic encoder. The output is passed into an aggregator function g, which calculates the mean of the given points. The aggregated value is then passed to 2 feedforward networks μ and σ to calculate the mean and standard deviation of the latent value.
3. **Decoder** - The decoder receives as input a concatenated vector of the target input X^* the output R of the deterministic encoder and the output Z of the latent encoder. This concatenated vector is then passed to a recurrent neural network to predict the target output Y^*.

The model can be trained in an end-to-end manner. Similar to the neural process, we optimize the parameters by maximizing the maximum likelihood of target data conditioned on the context data. The objective function is defined as:

$$J = E_{q(z|x_c,y_c)}\left[\sum_{i=1}^{t} \log p(y_i|z,x_i)\right] - KL(q(z|x_c,y_c)||q(z|x_t,y_t))$$

In our experiments, we train our model in a regression setting. $E_{q(z|x_c,y_c)}\left[\sum_{i=1}^{t} \log p(y_i|z,x_i)\right]$ could be the $MSE(y_t, RANP(x_t,x_c,y_c))$ or a time series similarity measure such as the Differential DTW loss. Other training setting such as classification are not explored in this thesis, but are possible with the model.

3.2 Recurrent Attentive Neural Processes Clustering

In this section, we present our proposed method, Recurrent Attentive Neural Processes Clustering (RANP-Clust). RANP-Clust is an extension of RANP, which focuses on time series clustering. First, to improve clusterability, we add an L2-normalization layer [2] on the deterministic latent values:

$$R = \frac{R}{\|R\|_2}$$

This small addition has been shown to improve clustering in autoencoders [2].In 3.2, we can see the updated deterministic encoder. We train the model with the same loss as RANP. The setup has the following steps:

1. **Training** - The model first is trained the same as in RANP with the addition of the L2 normalization layer.
2. **Clustering** - After the model is trained, we calculate all deterministic latent values, run the k-means algorithm [1], and determine the centroids of the clusters. The number of K can be either be predefined if we already know the number of clusters. Alternatively, the number of clusters K can be determined by using clustering metrics such as the silhouette score [54]. In our thesis, we use the scikit-learn implementation of k-means [47].

3. **Inference** - To cluster unseen data, we pass it through the deterministic encoder and get the deterministic latent values. We then determine the closest centroid, which we calculated in the previous step, and assign it to that cluster.

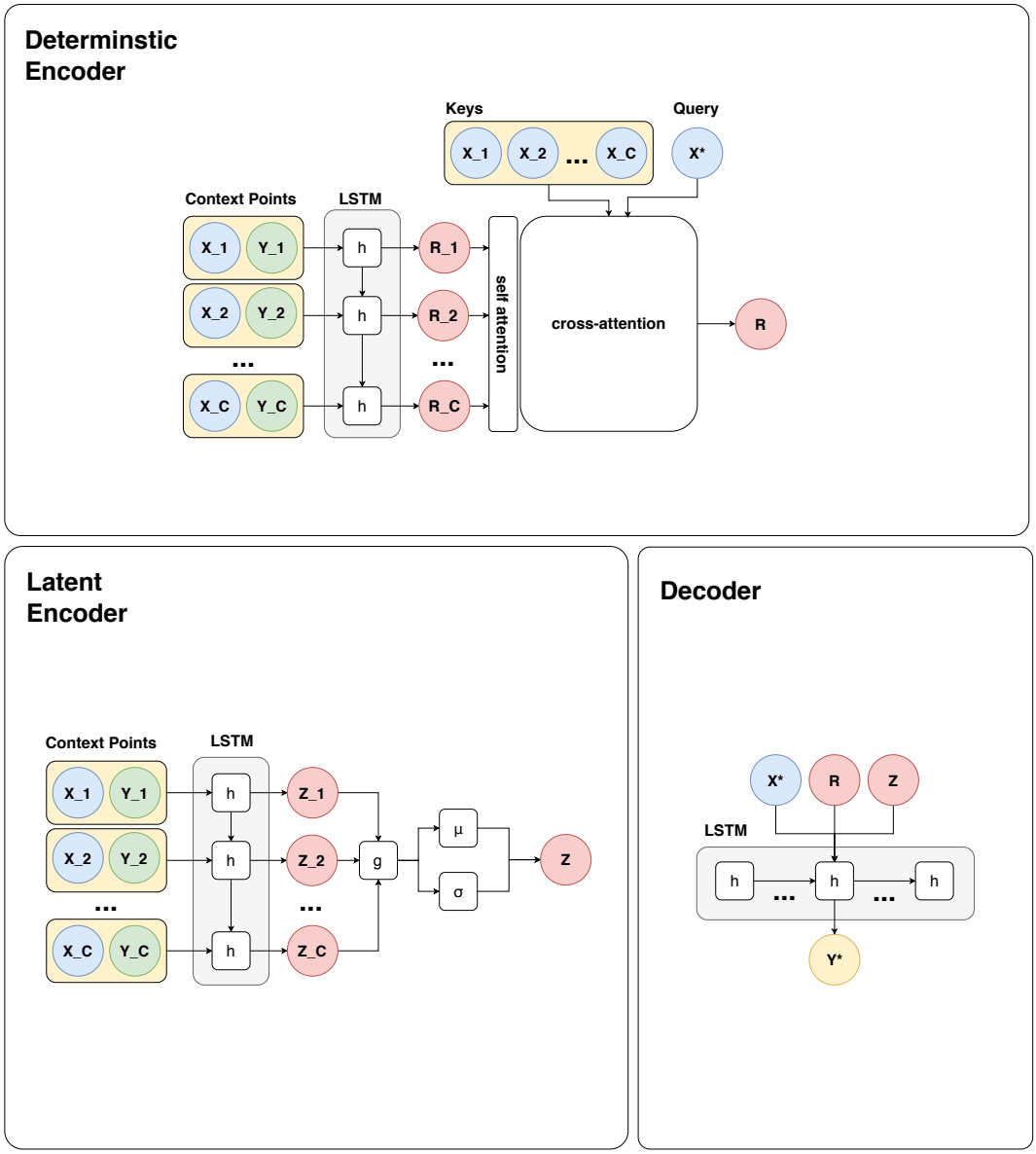


Figure 3.1: A set of context points is passed to a deterministic encoder and a latent encoder. Each encoder uses an RNN h to obtain a representation of the context points. For the latent encoder, we pass every representation z_c to the aggregator g and create a latent vector z. In the deterministic encoder, we pass every representation r_c into a self-attention layer. Afterward, we pass the output of the self-attention layer into a cross attention layer, where the keys are the input context points x_c and query the input target point x^*. This gives us the deterministic vector R. The decoder takes the latent vector z, deterministic vector r, and a target input x^* and passes into an RNN. The decoder returns a target output of y^*.

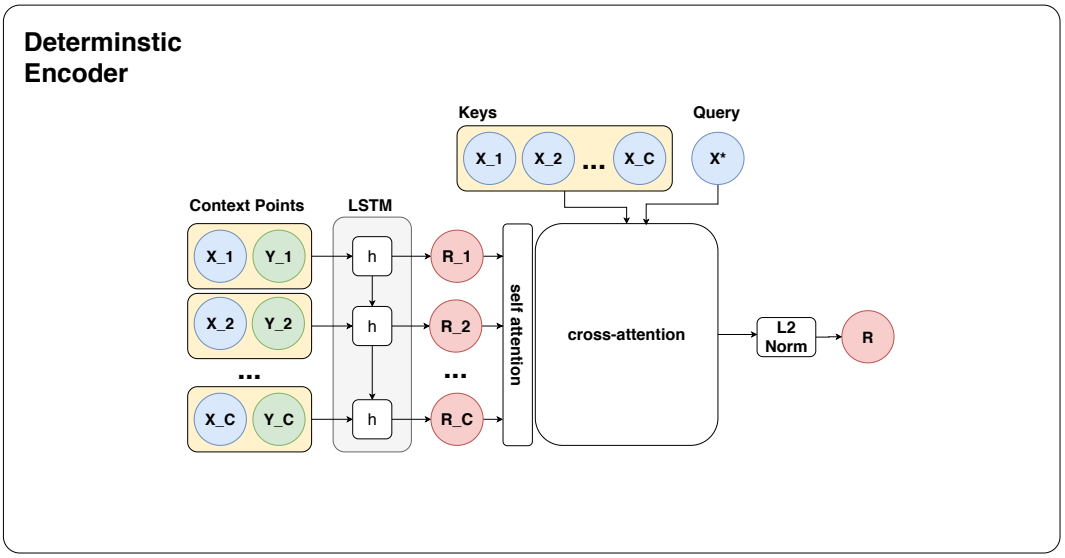


Figure 3.2: Deterministic Encoder of RANP-Clust - The output of the cross-attention R is passed into an L2-normalization layer.

4 Experiments

In this chapter, we explore how RANP performs on a variety of tasks. In the first experiment, we want to see if the addition of attention layers improves the performance of RNP. For this task, we will use the datasets used in the RNP paper.

4.1 Datasets

In this section, we will present the datasets used in our experiments.

4.1.1 Random walk Dataset

To test the model forecasting and clustering capability, we created a multivariate time series dataset based on random walks. A random walk (RW)[6] is defined as:

$$x_t = x_{t-1} + a$$
$$a \sim \mathcal{N}(\mu, \sigma^2)$$

where $x, a \in R^n$. We generate 5-dimensional RWs, with a length of 500. The last dimension of the RW is our output y, and the first four dimensions are our input x. As seen in 4.1, we split the data further into context and target points. In this example, we defined the first 100 points as context points and the rest as target points. However, the number of points can be arbitrarily set.

To test its clustering capability, we will create a finite number of RW samples. In 4.2, we can observe the different outputs y of 3 RWs. Furthermore, for each of these random walks, we create variations by adding random white noise (gaussian noise). In 4.3 we can observe how these variations may look like. By creating similar, but yet different RWs, we hope that the model can cluster them. Finally, we can control the standard deviation in the white noise. This results in noisier RW, which makes

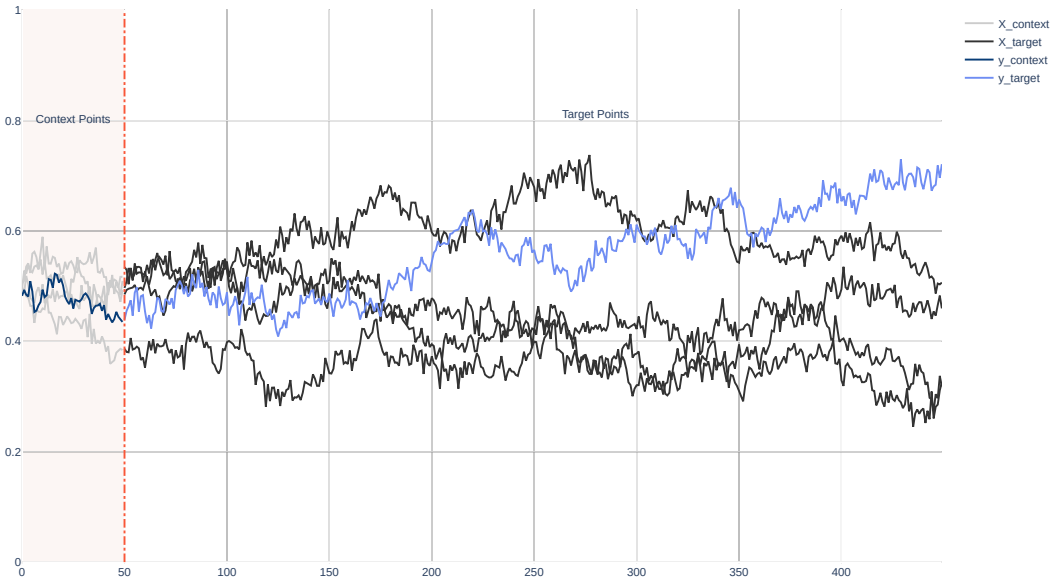


Figure 4.1: The following 5 dimensional RW is split into context and target points. One of the dimensions is defined as the output y, where the rest are the input x.

the initially generated RW samples harder to distinguish. In 4.4 we can observe this behaviour.

To understand the limitations of the model, we have generated the following dataset:

- **Cluster datasets** - In these datasets, we have generated 3,5,10 and 20 independent RWs. Each RW is 500 in length, and we set the first 50 points as context points. For each of these RWs, we have generated 500 variations with white noise. The standard deviation of the noise is 1.0. With these datasets, we want to see how many clusters the model can identify, given a limited capacity. More details about the model setup will be explained in the next section.
- **Noise datasets** - Each dataset contains five independent RWs. Each RW is 500 in length, and we set the first 50 points as context points. We generate 500 variations with white noise. However, the standard noise deviation is different for every dataset, with values 1, 5, 10, and 50, respectively. As the noise increases, the clusters are becoming harder to distinguish. Here we want to see how the model performs with different levels of noise. In other

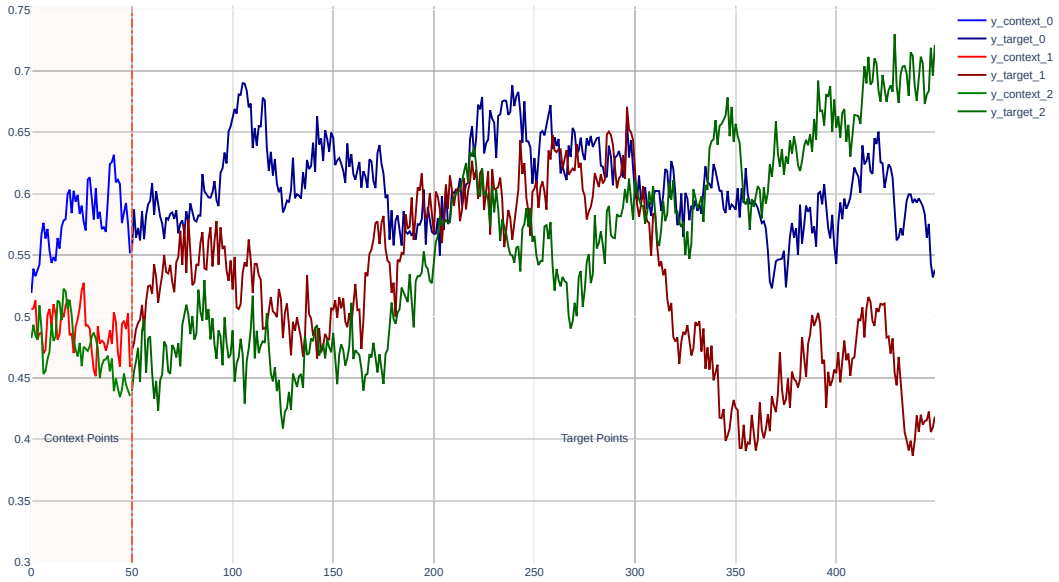


Figure 4.2: By using different random seeds, we can generate different looking RWs. We can visually distinguish them not only based on the context points but also on the target points.

words, we want to test the robustness of the model to noise.

- **Context dataset** - Each dataset contains five independent RWs. We generate 500 variations with white noise. The standard noise deviation is 50. However, the number of context points is different for every set, with 1, 50, 250, and 499 points, respectively. Given this very noise dataset, we want to explore if the context points help in identifying the noisy clusters.

For all the datasets, we have min-max normalized them with the scikit-learn library [47]. The random walks were generated with the help of the tslearn library [59]. Furthermore, we randomly select 20% as the validation set and 20% as the test set. We compare RANP and RANP-Clust. The objective function for all models is the same. We treat all datasets as a regression problem. The early stopping is based on the validation loss. Then we calculate the deterministic latent vectors for every value in the test set. Since RANP latent vectors are not normalized, we will apply L2 normalization as a postprocessing step. Afterward, we will run k-means++ on the latent values as in RANP-Clust. We use the scikit-learn implementation of k-means. The number of K is set to the number of clusters predetermined by the dataset. Finally, we assign the cluster labels to the true labels with the Hungarian

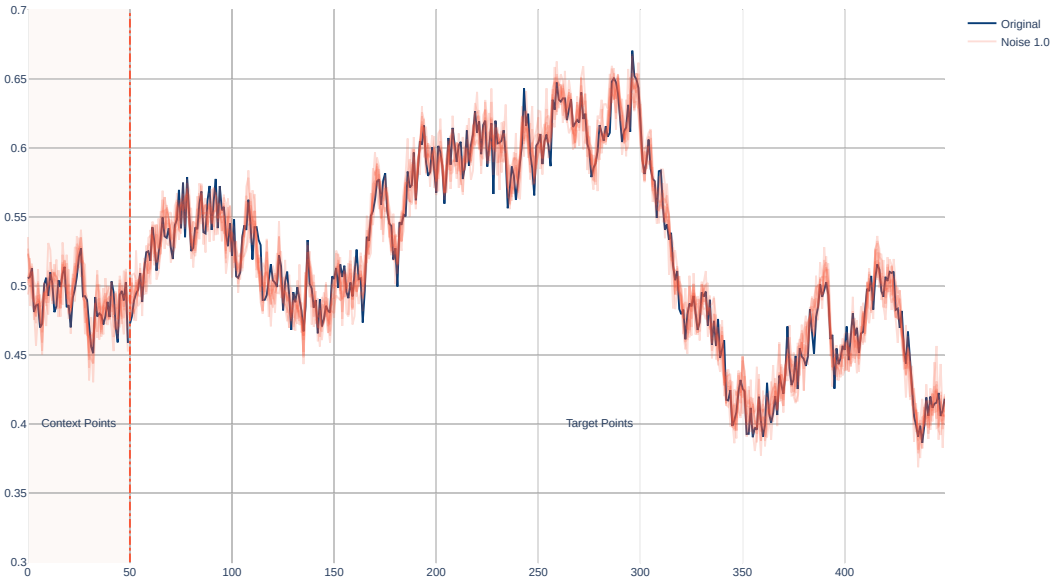


Figure 4.3: The dark blue line is a one dimensional RW sample. By adding white noise (marked with red), we can generate similar looking samples from the initially generated RW.

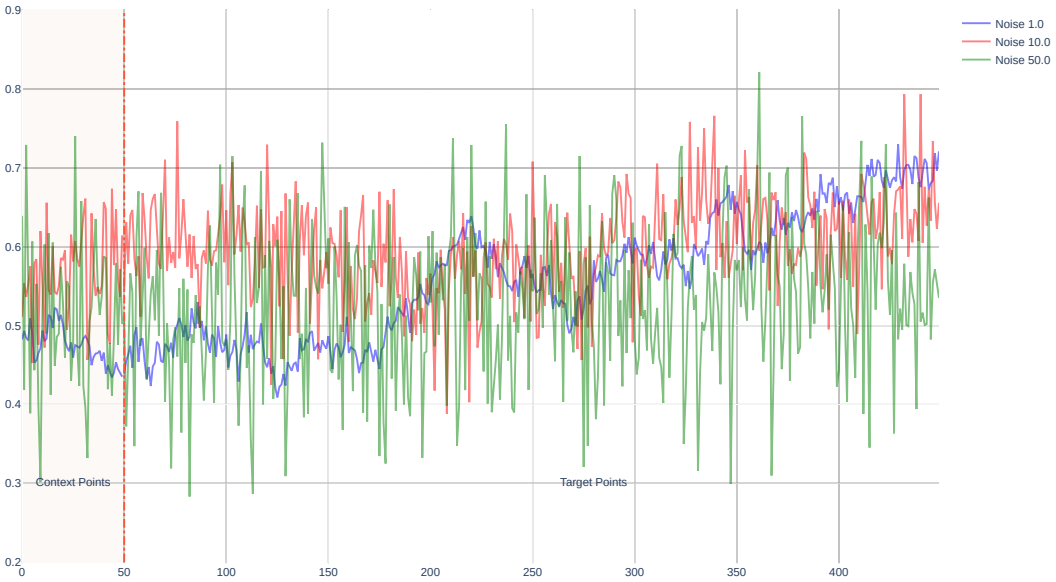


Figure 4.4: By increasing the standard deviation of the noise, we can observe more distorted RWs.

matching algorithm [31] and calculate the classification accuracy score. Note during training, the model has no prior knowledge of the number of clusters. Furthermore, during the training of the neural networks, the objective is only to minimize the regression error. The settings of the model are described in the implementation Details section. Finally, we will project the deterministic latent space in 2D using Universal Approximate Manifold (UMAP) as our dimensionality reduction. [39]

4.1.2 Electricity Dataset

The Electricity Dataset [12](also known as "Individual Household Electric Power Consumption Dataset") is a multivariate time series dataset. The dataset contains measurements for four years taken at a 1-minute interval. The features are:

- **Active Power** - household minute-averaged active power (in kilowatt)
- **Reactive Power** - household minute-averaged reactive power (in kilowatt)
- **Intesity** - household global minute-averaged current intensity (in ampere)
- **Voltage** - minute-averaged voltage (in volt)
- **Sub Metering 1** - It corresponds to the kitchen energy consumption
- **Sub Metering 2** - It corresponds to the laundry room energy consumption
- **Sub Metering 3** - It corresponds to an electric water-heater and an air-conditioner energy consumption

The dataset is split into training, validation, and test where the first two years are training, third-year validation, and test the fourth year. To compare with previous results (RNP [67] and Recurrent Neural Filters (RNF) [34]), we will set the feature "Active Power" as our target and the rest of the remaining features as our input. Furthermore, we used the same preprocessing steps of the data, which is described in the RNF paper [34]. Furthermore, to compare results with our proposed model, we trained the RNP model as described in [67]. In the next section, we will provide details of the hyperparameters used in RANP and RANP-Clust. As in [67], the models are evaluated for one-step prediction MSE. The reason we selected this dataset is to see if RANP and RANP-Clust outperform RNP in a real-life dataset.

4.2 Implementation Details

In this section, we describe the different architecture setups of our model and baselines. In the following figure, we can see the different hyperparameters for the different datasets. We optimize all models using Radam [36]. We train all model for 300 epochs with early stopping. We use the patience of 7 epochs, meaning that if no improvement on the validation loss for seven epochs has been achieved, then the training will stop. All models have been written and trained on Pytorch [45]. Additionally, we used Pytorch-lightning [14] to streamline the process of training and calculating results.

For all Random Walk Datasets, we train all models with 16 LSTM Units for both encoders and decoder. The latent size is for all 16. We use DotProduct attention for self-attention and Multi-head Attention for cross-attention and use a learning rate of 0.001. For the Electricity Dataset, we train all models with the same parameters as in [67]. All models use 64 LSTM Units for both encoders and decoder. The latent size is equal to 4. We use dot product attention for self-attention and Multi-head Attention for cross-attention and a learning rate of 0.01. Same as in [67], we use 20 points as the context points and 60 for the target points.

5 Results

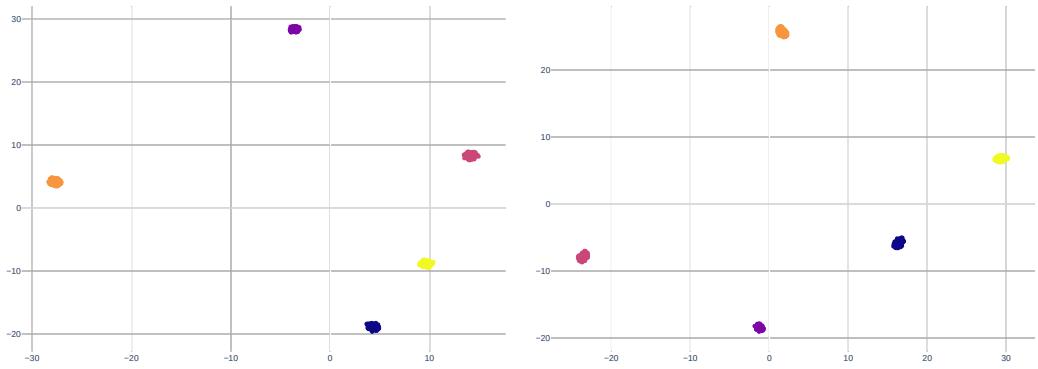
5.1 Random Walk - Clusters Dataset

We have trained RNP, RANP, and RANP-Clust on the clusters dataset. As previously described, we are trained in the models in a regression setting. After training, we then apply k-means and calculate the accuracy. In the following Table 5.1, we can observe that all models can cluster the data correctly when given 3 clusters. However, we can see that by increasing the cluster size, the RNP model cannot keep up. Furthermore, RANP-Clust outperforms RANP and RNP at all the cluster sizes. That said, we observed that all model achieved perfect accuracy, when increasing the number of units in the LSTM and size of the latent space.

In 5.2, we observe the deterministic latent space of RANP and RANP-Clust. We can observe that both models separate each cluster perfectly. In 5.3, data is RANP-Clust has less overlapping clusters than in RANP. However, it is hard to tell, since this is a projection and not the original space of the data.

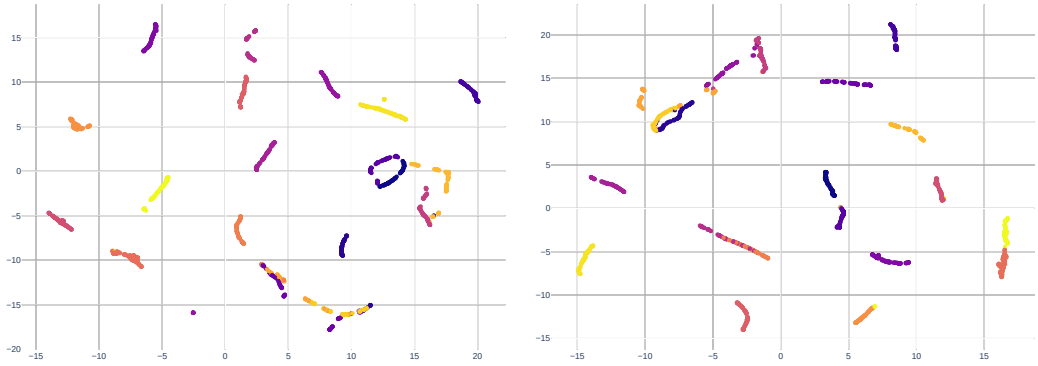
Clusters	RNP	RANP	RANP-Clust
3	**1.0**	**1.0**	**1.0**
5	0.68	**1.0**	**1.0**
10	0.79	**1.0**	**1.0**
20	0.44	0.66	**0.70**

Figure 5.1: Accuracy Score on the Cluster Dataset



(a) RANP (b) RANP-Clust

Figure 5.2: Deterministic latent space projected with UMAP on the 5 cluster dataset. With each color representing the ground truth label.

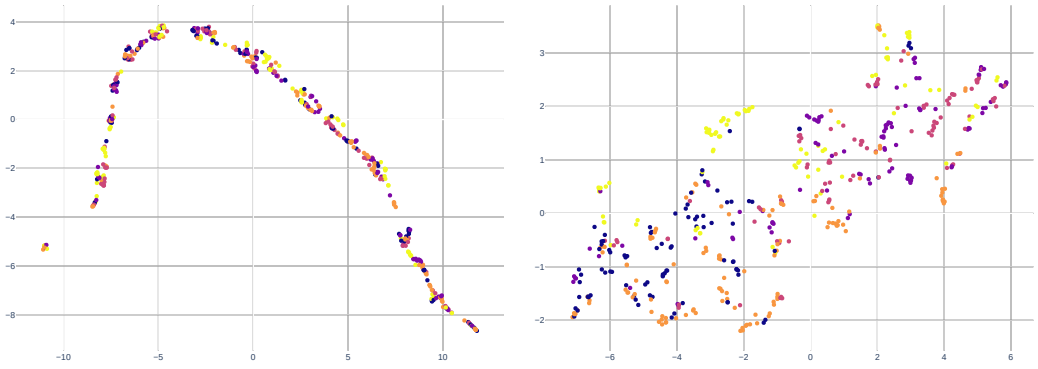


(a) RANP (b) RANP-Clust

Figure 5.3: Deterministic latent space projected with UMAP on the 20 cluster dataset. With each color representing the ground truth label.

Noise	RNP	RANP	RANP-Clust
1	0.68	**1.0**	**1.0**
5	0.55	**1.0**	**1.0**
10	0.89	0.75	**0.95**
50	0.37	0.25	**0.40**

Figure 5.4: Accuracy Score on the Noise Dataset



(a) RANP (b) RANP-Clust

Figure 5.5: Deterministic latent space projected with UMAP on the noise 50 dataset. With each color representing the ground truth label.

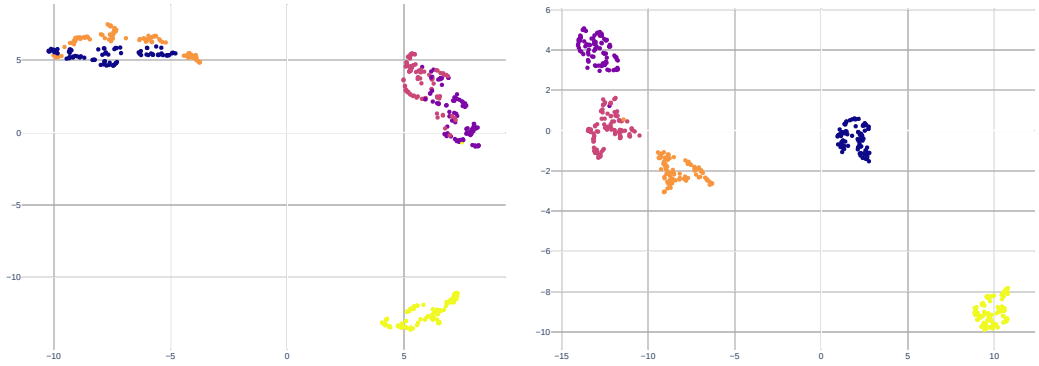
5.2 Random Walk - Noise Dataset

We have observed that RANP and RANP-Clust can cluster correctly when given dataset with only 5 clusters. However, does the model keep performing when increasing the noise? While increasing the noise, we expect a decrease in performance, since the noise makes it harder to distinguish the cluster from each other. In the following Table 5.4, we observe again that RANP-Clust outperforms at all noise levels. Interesting is that RNP seems to outperform RANP when the noise levels are high. We suspect this is due to attention overfitting to the noise.

In 5.5 and 5.6, we can observe the discrimitive latent space for noise level 50 and 10 respectively. In 5.6, we can observe how RANP-Clust clusters are more spread than the ones from RANP.

5.3 Random Walk - Context Points Dataset

Finally, we want to observe the effect of the context points. When given a noisy dataset, does the accuracy improve when given more context points? In the following Table 5.7, we can observe that RANP-Clust for all cases except one, outperforms RANP and RNP. Furthermore, RNP outperforms RANP in all cases, and when given 250 context points, it seems to outperform RANP-Clust. We suspect that this again due to the high noise in the dataset, making it easier for attention to overfit to

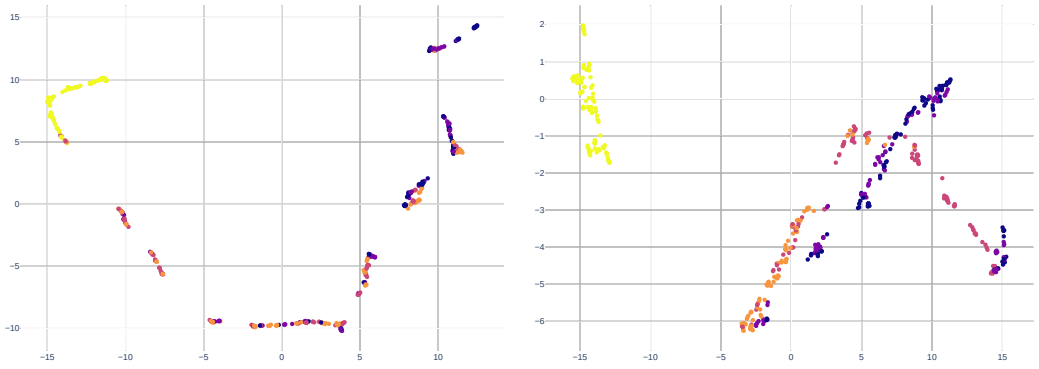


(a) RANP (b) RANP-Clust

Figure 5.6: Deterministic latent space projected with UMAP on the noise 10 dataset. With each color representing the ground truth label.

Context Points	RNP	RANP	RANP-Clust
1	0.24	0.25	**0.26**
50	0.37	0.25	**0.4**
250	**0.61**	0.44	0.54
499	0.94	0.90	**1.0**

Figure 5.7: Accuracy Score on the Context Points Dataset



(a) RANP (b) RANP-Clust

Figure 5.8: Deterministic latent space projected with UMAP on the noise 50 dataset with 250 context points. With each color representing the ground truth label.

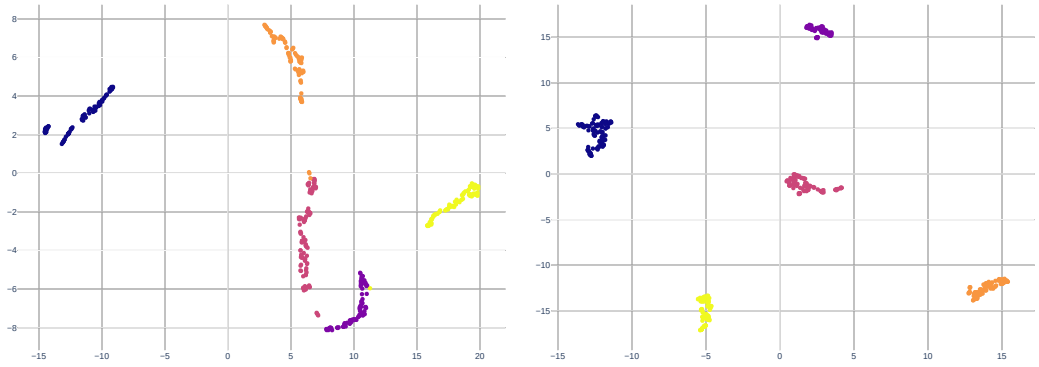
noise. Finally, when given a single context point, all models perform similar to each other.

In 5.8 and 5.9, we can observe the deterministic latent space for 250 and 499 context points. In 5.9, similarly as in 5.6, we can observe how RANP-Clust clusters are more spread out than RANP. We remind the reader again that this is a 2D projection with UMAP and not the actual latent space.

5.4 Electricity Dataset

With the RWs datasets, we want to explore the limitations of the model. With the electricity dataset, we want to explore the performance in real-life datasets. In the following Table 5.10, we present the normalized MSE based on the LSTM score for a one-step prediction.

We observe that RANP outperforms RNP and most other architectures. However, RNF-NP still outperforms our proposed architecture. Moreover, we observe that RANP-Clust, with its L2-Normalization, seems to perform worse than LSTM. Based on these results, we can see that adding attention to RNP does provide better performance. However, if the tasks are only regression, RANP-Clust will provide worse performance.



(a) RANP (b) RANP-Clust

Figure 5.9: Deterministic latent space projected with UMAP on the noise 50 dataset with 499 context points. With each color representing the ground truth label.

Model	**Normalized MSE**
LSTM	1.000
VRNN	1.902
DFK	1.252
DSSM	1.131
RNF-LG	0.918
RNF-NP	**0.856**
RNP	1.111
RANP	**0.899**
RANP-Clust	1.263

Figure 5.10: Normalized MSE for One-Step Predictions

6 Conclusion and Future Work

In this thesis, we proposed a new architecture called Recurrent Attentive Neural Processes for multivariate time series prediction and shown state of the art performance on the Electrical consumption dataset. We will also present an extension called Recurrent Neural Processes Clustering, which allows clustering time-series data. We show that the addition of L2-normalization during training provides better performance in clustering time-series data. Furthermore, we show how RANP-Clust outperforms RANP and RNP in this task.

In this thesis, we barely scratched the surface. More experimentation is needed with the proposed method in other datasets. As described in the background section, NPs have meta-learning capabilities. One direction we could explore if RANP is better or worse in meta-learning tasks than RNP. Regarding RANP-Clust, we could combine methods explored in the introduction section. One approach would be to add an auxiliary loss for clustering the model and for example, training a GMM as in VADE [25], a self-organizing map in latent space as in SOM-VAE [15] or something simple as a k-means loss trained jointly with the model.

Moreover, we would like to explore the capabilities of anomaly detection of the model. Here there are several methods such as determining outliers based on the reconstruction error or based on the centroids we determined after running the k-means algorithm in RANP-Clust [2].

Finally, one could explore different ways to encode time series data in neural processes. We explored a method based on recurrent neural networks and attention mechanism. As we discussed in the time series representation learning section, there is a much different way to encode time series data. Two exciting directions would be to explore casual diluted convolution and models which uses only attentions as in Transformers [63].

Bibliography

[1] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, 2006.

[2] C. Aytekin, X. Ni, F. Cricri, and E. Aksu. Clustering and unsupervised anomaly detection with l 2 normalized deep auto-encoder representations. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6. IEEE, 2018.

[3] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[5] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series.

[6] P. J. Brockwell and R. A. Davis. *Introduction to time series and forecasting*. springer, 2016.

[7] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[8] M. Cuturi and M. Blondel. Soft-dtw: A differentiable loss function for time-series. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 894–903. JMLR.org, 2017.

[9] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[10] I. Davidson and S. Ravi. Agglomerative hierarchical clustering with constraints: Theoretical and empirical results. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 59–70. Springer, 2005.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[12] D. Dua and C. Graf. Individual household electric power consumption data set.

[13] S. M. A. Eslami, D. Jimenez Rezende, F. Besse, F. Viola, A. S. Morcos, M. Garnelo, A. Ruderman, A. A. Rusu, I. Danihelka, K. Gregor, D. P. Reichert, L. Buesing, T. Weber, O. Vinyals, D. Rosenbaum, N. Rabinowitz, H. King, C. Hillier, M. Botvinick, D. Wierstra, K. Kavukcuoglu, and D. Hassabis. Neural scene representation and rendering. *Science*, 360(6394):1204–1210, 2018.

[14] W. e. a. Falcon. Pytorch lightning. https://github.com/PytorchLightning/pytorch-lightning, 2019.

[15] V. Fortuin, M. Hüser, F. Locatello, H. Strathmann, and G. Rätsch. Som-vae: Interpretable discrete representation learning on time series. *arXiv preprint arXiv:1806.02199*, 2018.

[16] J.-Y. Franceschi, A. Dieuleveut, and M. Jaggi. Unsupervised scalable representation learning for multivariate time series. In *Advances in Neural Information Processing Systems*, pages 4652–4663, 2019.

[17] M. Garnelo, J. Schwarz, D. Rosenbaum, F. Viola, D. J. Rezende, S. Eslami, and Y. W. Teh. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018.

[18] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. 2016.

[19] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[20] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[21] D. Hallac, S. Vare, S. Boyd, and J. Leskovec. Toeplitz inverse covariance-based

clustering of multivariate time series data. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 215–223, 2017.

[22] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[23] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework.

[24] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[25] Z. Jiang, Y. Zheng, H. Tan, B. Tang, and H. Zhou. Variational deep embedding: An unsupervised and generative approach to clustering. *arXiv preprint arXiv:1611.05148*, 2016.

[26] S. Kaisler, F. Armour, J. A. Espinosa, and W. Money. Big data: Issues and challenges moving forward. In *2013 46th Hawaii International Conference on System Sciences*, pages 995–1004. IEEE, 2013.

[27] T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[28] R. J. Kate. Using dynamic time warping distances as features for improved time series classification. *Data Mining and Knowledge Discovery*, 30(2):283–312, 2016.

[29] E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping to massive datasets. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 1–11. Springer, 1999.

[30] H. Kim, A. Mnih, J. Schwarz, M. Garnelo, A. Eslami, D. Rosenbaum, O. Vinyals, and Y. W. Teh. Attentive neural processes. *arXiv preprint arXiv:1901.05761*, 2019.

[31] H. W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[32] A. Kusupati, M. Singh, K. Bhatia, A. Kumar, P. Jain, and M. Varma. Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network, 2019.

[33] T. A. Le, H. Kim, M. Garnelo, D. Rosenbaum, J. Schwarz, and Y. W. Teh. Empirical evaluation of neural process objectives.

[34] B. Lim, S. Zohren, and S. Roberts. Recurrent neural filters: Learning independent bayesian filtering steps for time series prediction, 2019.

[35] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 2–11, 2003.

[36] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

[37] S. Makridakis, E. Spiliotis, and V. Assimakopoulos. Statistical and machine learning forecasting methods: Concerns and ways forward. *PloS one*, 13(3), 2018.

[38] G. Marcus. Deep learning: A critical appraisal, 2018.

[39] L. McInnes, J. Healy, and J. Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[40] H. N. Mhaskar and C. A. Micchelli. How to choose an activation function. In *Advances in Neural Information Processing Systems*, pages 319–326, 1994.

[41] F. Murtagh. Multilayer perceptrons for classification and regression. *Neurocomputing*, 2(5-6):183–197, 1991.

[42] A. v. d. Oord, O. Vinyals, and K. Kavukcuoglu. Neural discrete representation learning. *arXiv preprint arXiv:1711.00937*, 2017.

[43] E. Oztemel and S. Gursev. Literature review of industry 4.0 and related technologies. *Journal of Intelligent Manufacturing*, 31(1):127–182, 2020.

[44] J. Paparrizos and L. Gravano. k-shape: Efficient and accurate clustering of time series. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1855–1870, 2015.

[45] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[46] B. A. Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Comput.*, 1(2):263–269, June 1989.

[47] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[48] W. Pei, D. M. Tax, and L. van der Maaten. Modeling time series similarity with siamese recurrent networks. *arXiv preprint arXiv:1603.04713*, 2016.

[49] F. Petitjean, G. Forestier, G. I. Webb, A. E. Nicholson, Y. Chen, and E. J. Keogh. Dynamic time warping averaging of time series allows faster and more accurate classification. *2014 IEEE International Conference on Data Mining*, pages 470–479, 2014.

[50] S. Qin, J. Zhu, J. Qin, W. Wang, and D. Zhao. Recurrent attentive neural process for sequential data, 2019.

[51] C. E. Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[52] A. Razavi, A. van den Oord, and O. Vinyals. Generating diverse high-fidelity images with vq-vae-2. In *Advances in Neural Information Processing Systems*, pages 14837–14847, 2019.

[53] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[54] P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[55] H. Shatkay. The fourier transform-a primer. 1995.

[56] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks, 2015.

[57] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[58] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[59] R. Tavenard, J. Faouzi, G. Vandewiele, F. Divo, G. Androz, C. Holtz, M. Payne, R. Yurchak, M. Rußwurm, K. Kolar, and E. Woods. tslearn: A machine learning toolkit dedicated to time-series data, 2017. https://github.com/rtavenar/tslearn.

[60] Y. W. Teh. Dirichlet process.

[61] D. J. Trosten, A. S. Strauman, M. Kampffmeyer, and R. Jenssen. Recurrent deep divergence-based clustering for simultaneous feature learning and clustering of variable length time series. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3257–3261. IEEE, 2019.

[62] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio, 2016.

[63] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[64] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

[65] J.-L. Wang, J.-M. Chiou, and H.-G. Mueller. Review of functional data analysis, 2015.

[66] V. Wiley and T. Lucas. Computer vision and image processing: a paper review. *International Journal of Artificial Intelligence Research*, 2(1):29–36, 2018.

[67] T. Willi, J. Masci, J. Schmidhuber, and C. Osendorfer. Recurrent neural processes. *arXiv preprint arXiv:1906.05915*, 2019.

[68] R. E. Wright. Logistic regression. 1995.

[69] J. Xie, R. Girshick, and A. Farhadi. Unsupervised deep embedding for clustering analysis, 2015.

[70] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

[71] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4694–4702, 2015.

[72] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena. Self-attention generative adversarial networks. *arXiv preprint arXiv:1805.08318*, 2018.

[73] J. Zhao, X. Mao, and L. Chen. Speech emotion recognition using deep 1d 2d cnn lstm networks. *Biomedical Signal Processing and Control*, 47:312 – 323, 2019.

[74] T. Zhao, Z. Wang, A. Masoomi, and J. G. Dy. Adaptive nonparametric variational autoencoder. *arXiv preprint arXiv:1906.03288*, 2019.

[75] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao. Time series classification using multi-channels deep convolutional neural networks. In *International Conference on Web-Age Information Management*, pages 298–310. Springer, 2014.

[76] J. Zhu, S. Qin, W. Wang, and D. Zhao. Probabilistic trajectory prediction for autonomous vehicles with attentive recurrent neural process. *arXiv preprint arXiv:1910.08102*, 2019.

[77] S. Zolhavarieh, S. Aghabozorgi, and Y. W. Teh. A review of subsequence time series clustering. *The Scientific World Journal*, 2014, 2014.