<need to be updated >

Advanced Machine Learning - Final Project

Yaniv Tal 031431166

Miriam Horovicz 313246704

Contents

[Part 1 – Anchor paper 2](#_Toc9155)

[Overview - Problem and Existing Solution Approaches 2](#_Toc9156)

[Anchor Paper summary 3](#_Toc9157)

[Problem formulation 3](#_Toc9158)

[Model Overview 3](#_Toc9159)

[Cost function 3](#_Toc9160)

[Related Papers (“River of citations”) 4](#_Toc9161)

[Anchor paper implementation and results 5](#_Toc9162)

[Our Implementation 5](#_Toc9163)

[Anchor paper experiments and results 6](#_Toc9164)

[Appendix 1 – Train Code flow – HMNIST 7](#_Toc9165)

Introduction

## Project Objective

Training time-series classification model with limited data is very challenging and important task.

It is challenging because training machine learning and especially deep learning models with limited data set is hard task. Those architectures are hungry for data and achieve low results when learning from small dataset.

It is important because in real-life scenarios this phenomenon is very common. Collecting real time-series data is time-consuming and require high costs in industries like finance and health care. Therefore, in many cases, only limited relevant time-series data is collected and available for data science work. In many cases, the amount of supplied real data is not enough for training accurate deep learning model.

When dealing with limited data, methods for synthesizing high quality data are valuable. The challenge with those methods for time-series data is their ability to generate fake data with real characteristics of both the spatial correlation (multivariate correlation between features) and temporal correlation.

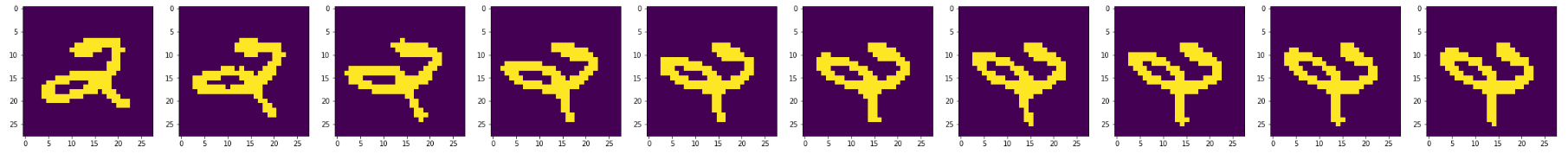
Our objective in this final project is taking GP-VAE that was originally introduced for time-series imputation task and use it for generating high quality time series data. As spatial and temporal correlation already built in its architecture, we believe that it can be a good fit for synthetic data generation method as well. The goal is to enrich our limited training data set with synthetic data generated by GP-VAE for improving our downstream classification task.

## Data Source

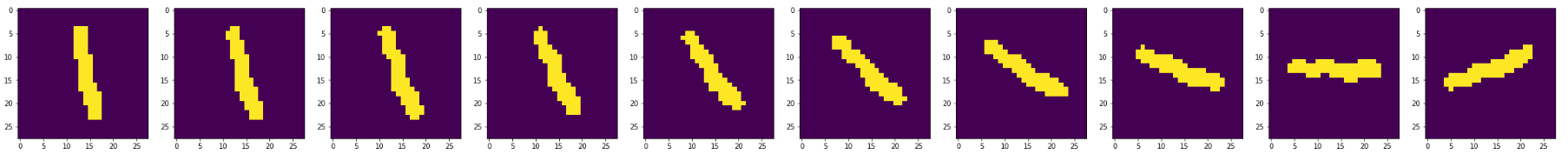
The data to be used in adjusted Healing – MNIST. The original Healing – MNIST is creating series of digits images with rotation to simulate time-series data. The label used for each series is the digit (10 options). We took it one step further and adjusted it for including both the spatial & temporal correlation in each series label. Our Healing -MNIST is creating digit series with rotation to specific random direction (right or left). Each label is combination of digit + series rotation. Overall, we have 20 different labels (10 digits X 2 directions). Another adjustment we added is the ability to create limited training set. This is required for simulating the limited data situation we need to work on. More details on it will be described later in the innovative part.

Examples of series generated:

**2 Right**



**1 Left**



## Evaluation

As synthetize time-series data is intermediate step and not a goal by itself, we will evaluate our results by assessing the improvement we gained by using generated data in classification task accuracy.

Related Work

<need to describe here TimeGAN + another paper from below options>

TimeGAN

<https://papers.nips.cc/paper/2019/file/c9efe5f26cd17ba6216bbe2a7d26d490-Paper.pdf>

https://arxiv.org/pdf/2207.14406v1.pdf

<https://github.com/acphile/RTSGAN>

<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9206942>

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9441381

# Anchor paper

## Overview - Problem and Existing Solution Approaches

Multivariate time series with missing values are common in areas such as healthcare and Finance, since data in these areas tends to be sampled / recorded in different resolutions and different times. E.g. – a patient’s blood pressure may be measured several times a day, sugar blood test twice a day, heart rate continuously etc. In order to use this data with a model, it is required to impute the missing elements. Imputation of multivariate distribution over time is a challenging problem – the model should take into account both the spatial correlation (Multivariate correlation between features) and temporal correlations (Elevated heart-rate and blood sugar an hour ago could be correlated to blood pressure afterwards etc.).

There are several simpler approaches that can be used for such imputation:

1. Single imputation methods – fill missing values with a fixed value per feature – mean, median, last value observed etc. – leads to loss of spatial and temporal correlation and hence the simplest but worst approach.
2. Look at each data point in time as a separate element and impute based on spatial correlation only using a pointwise spatial model (E.g., VAE, Hi-VAE, GAN) – loses the temporal relationship over time, so less appropriate for time series data.
3. Run a regression model on each feature separately over time (E.g., Gaussian process in original data space) – loses the spatial correlations between the features, not suitable for data with spatial correlations such as medical. Based on domain knowledge, it is clear that both spatial and temporal correlations in medical data are important and relevant, and so a more complex approach is required.
4. RNN based approaches (GRUI-Gan, BRITS, others) – Recurrent neural networks are common in language models, and can catch both spatial and temporal data by modeling the patient’s “latent health state” in the network hidden state – a value that is updated and passed from one recurrent node to the next.
5. Transformers – Based on attention mechanism and vastly used in many machine learning applications. We found some references to Transformers for Time series modeling (See related papers section), but not to time series imputation.

The paper proposes a deep probabilistic generative model for multivariate time series imputation, combining ideas from variational autoencoders and gaussian processes in order to take into account both the spatial temporal aspects of the data into account.

## Anchor Paper summary

### Problem formulation

Assume a dataset Χ ∈ ℝ × with T data points 𝑥 = 𝑥 , … , 𝑥 , … , 𝑥 ∈ ℝ that were

measured in T consecutive time points 𝜏 = [𝜏 , … , 𝜏 ]. Also assume that each data point 𝑥 could have missing (Unknown) values. We can mark the observed value of data point 𝑥 as 𝑥 ≔

𝑥 𝑥 𝑖𝑠 𝑜𝑏𝑠𝑒𝑟𝑣𝑒𝑑], and the missing values as 𝑥 ≔ 𝑥 𝑥 𝑖𝑠 𝑚𝑖𝑠𝑠𝑖𝑛𝑔]. Given that, the imputation problem can be formulated as finding 𝑝(𝑥 |𝑥 : ).

### Model Overview

The GP-VAE model described in the paper is an Encoder-Decoder that approaches the problem in two stages: Transform the data from the original data plane Χ ∈ ℝ × with missing data into a latent data plane 𝑍 ∈ ℝ × , 𝑘 < 𝑑 with full representation, and apply GP in the latent space of the VAE to learn the temporal relationships. This approach decouples the filling of missing values (Done in the translation from the data space to the latent space) from the temporal dynamic aspects, which are done in the latent space.

The model training aims to approximate the (Unknows) true posterior 𝑝(𝑧 : , |𝑋 : ) with a multivariate Gaussian variational distribution 𝑞 𝑧 : , 𝑋 : = 𝒩(𝑚 , Λ ), where 𝑗 is the dimension in the latent space. This approximation assumes independence between dimensions in the latent space (Typical for VAE models), but does take into account temporal correlations in that space. The variational family used is multivariate Gaussian in time domain, with precision matrix Λ parameterized as a product of bidiagonal matrices such that:

𝑏 𝑖𝑓 𝑡 ∈ {𝑡, 𝑡 + 1}

Λ ≔ 𝐵 𝐵 , 𝐵 =

0, 𝑜𝑡ℎ𝑒𝑟𝑤𝑖𝑠𝑒

In Gaussian processes, the kernel (“Covariance function”) has great effect on the behavior of the process over time. E.g. – Gaussian or RBF will result in smoother functions with higher local correlation, periodical kernels will result in periodic behavior where remote elements with fixed distance in time are correlated to each other etc. It was noticed by the writers that medical data tends to have time correlations in different time resolutions – e.g. one feature can be correlated to another within seconds, while another could have impact on others after hours. To address this attribute of the data, they chose to use Kauchi kernel, which handles disffernet time scale correlations well:

(𝜏 − 𝜏 )

𝑘 (𝜏, 𝜏 ) = 𝑠𝑖𝑔𝑚𝑎 1 +

𝑙

### Cost function

The parameters of the generative model 𝜃 and inference network 𝜓 are trained jointly using the evidence lower bound (ELBO) cost function:

log 𝑝(𝑋 ) ≥ 𝐸 ( | : ) 𝑙𝑜𝑔𝑝 (𝑥 𝑧 ) − 𝛽𝐷 [𝑞 (𝑧 : |𝑥 : ) ∥ 𝑝(𝑧 : )]

During inference, the ELBO is evaluated only for the observed sample elements, and the missing ones (Masked in training) are set to zero in order to prevent learning the “Missingness” of data as a latent feature. The parameter 𝛽 was added to balance the ELBO parts (likelihood and 𝐷 ).

## Anchor paper implementation and results

### Our Implementation

The paper provides a link to the authors git repository where their original code can be found https://github.com/ratschlab/GP-VAE. The code is rather complex and written with TensorFlow v.1, but provided good directions for obtaining the data, training, and reproduction of the results.

Since the code is already implemented and working, we decided to invest the time in analyzing the code, in order to improve our understanding and for further use in the Innovation part. Train flow is described in details in Appendix 1.

We created a Jupyter notebook that runs the original python code and copies inputs / outputs to google drive. To run it:

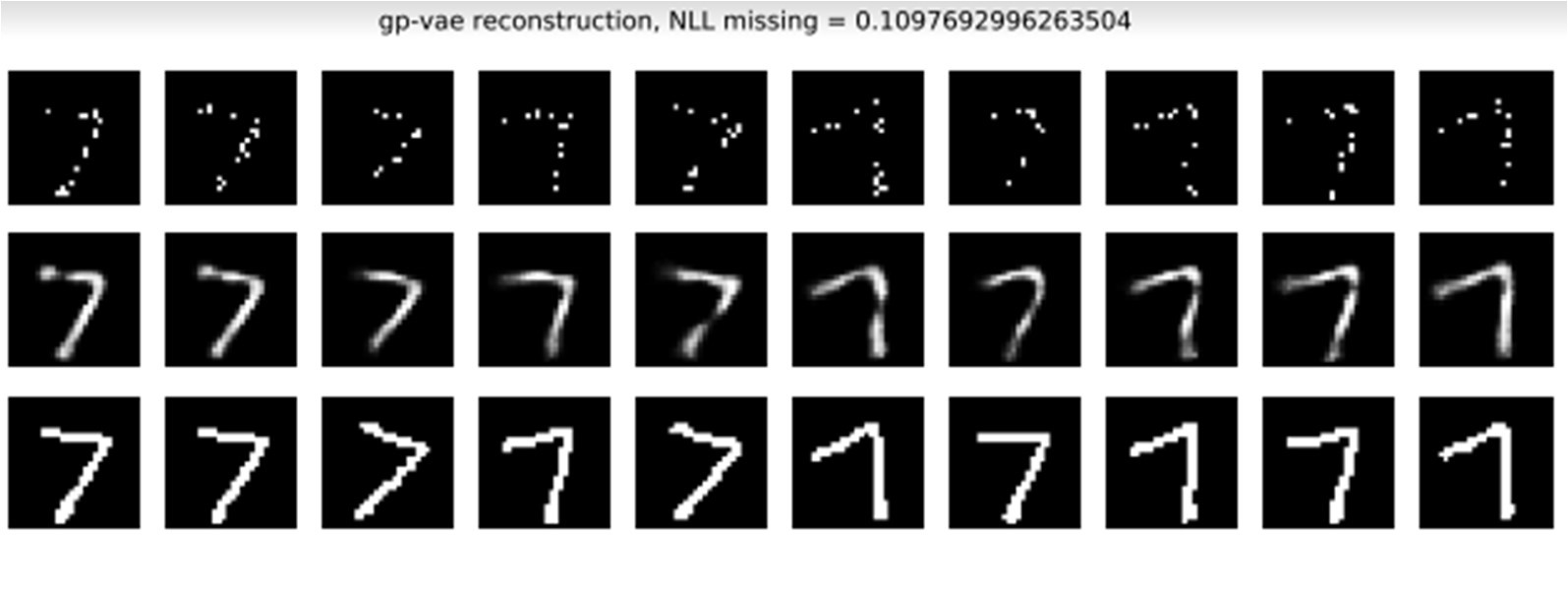
1. Download the notebook from https://github.com/mryanivtal/aml\_final\_project and place it in your google drive
2. Download the git repo from https://github.com/ratschlab/GP-VAE and place it in your google drive
3. Follow the notebook instructions

### Anchor paper experiments and results

The writers of the paper tested their model on three datasets: Healing MNIST, Sprites, Real medical time series data. Since all parameters for the original run were provided, we have used the exact same ones as the original implementation, and got very similar results, as following:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Healing MNIST | |  | SPRITES | Medical |
| Run | NLL | MSE | AUROC | MSE | AUROC |
| Original paper | 0.350 ± 0.007 | 0.114 ± 0.002 | 0.960 ± 0.002 | 0.002 ± 0.000 | 0.730 ± 0.006 |
| Our reproduction | 0.3451 | 0.1098 | 0.9591 | 0.0018 | 0.7182 |

HMNIST imputation demonstration (Our run):



Sprites imputation demonstration (Our run):



# Innovative Part

## Overview

As described in the introduction part, our goal is dealing with one of the most real challenges of training deep learning model with limited data. We will use GP-VAE for creating synthetic data to enrich our train set and evaluate the impact of it by reporting classification model accuracy on the test set. By this we took GP-VAE as intermediate step in time-series classification task. GP-VAE is not the goal here or the final stage but intermediate step to improve our time-series classification task results.

The innovative part here is using GP-VAE which is architecture originally built for imputation and leverage it to different task of synthetic data generation. This is very different approach from the classical synthetic data generation exist in the field as described above. Those methods are designed to learn data distribution and then generate new samples to get synthetic data from the same distribution of the original train data. We took different approach here by using model that was designed to impute data properly and train it on our limited train set. Then, we take our limited train data, add mask on it to get “missing” data and give GP-VAE to impute it. In this way we enrich our limited train set with new samples with real characteristics of our original train data.

Our method is also different from classical approaches in operative standpoint. As the classical methods learn data distribution, to use them for enriching train set of classification task you need to train on each category data separately to generate appropriate data. In our case, since we have 20 different categories, we need to train 20 (!) different deep learning data generation models i.e., TimeGAN. When using our method, we are training only 1 model of GP-VAE to be used to impute our train set data and create new data. Clearly, our approach is much more efficient in computation power and training time manners.

## Solution

Our solution comes to improve time-series classification accuracy in limited training set scenario. The following steps are taken:

* Train GP-VAE on limited train set. The trained model is capable for impute missing data in the time-series data.
* Add artificial masks on the limited train set to create artificial “missing” data points in the data
* Give the trained GP-VAE model to impute the missing values
* The imputed data is new generated data with similar characteristics to the original train set
* Add the generated data to the original limited train set
* Train time-series classification model on the joint train set
* Evaluate accuracy result on the test set

## Implementation

We implemented the following parts:

### Healing MNIST

We implemented Healing MNIST based on the original implementation with the following adjustments:

* Ability to control train set size created, it is essential for simulating the limited train set scenario
* Add direction as part of the label for each digit direction. Each series is rotated to one direction only right or left along the series and this info is part of the label.

### GP-VAE

We implemented GP-VAE based on the original implementation with the following adjustments:

* Ability to control train set size used for GP-VAE training. It is essential for simulation the limited train set scenario.
* Separate the train and inference mode. It is essential for training model once and then use it for generating data in inference mode.

### TimeGAN

We implemented Time-GAN flow based on the original implementation with the following adjustments:

* Use limited train set as input, split it to different data sets based on label
* For each different data set, train TimeGAN model and generate relevant data

### Time-Series Classification

we implemented basic deep learning model for time-series classification. We run the following experiments:

* Training on limited train set
* Training on limited train set plus GP-VAE generated data
* Training on limited train set plus TimeGAN generated data

we compared the above results to understand our baseline accuracy results on the limited train set and each generation method impact. Results are reported below.

Source code with run instructions and code references can be found here: <https://github.com/mryanivtal/aml_final_project>

## Results

The results on the experiments we ran are reported below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | Ours | | TimeGAN |
| Experiment | # per class | Accuracy using train set | Accuracy Train Run  With generated data Time (hours) | | Accuracy Train Run  With generated data Time (hours) |
| 1 | 200 | 71% | **94%** | **0.5** | **94%**  8 |
| 2 | 300 | 72% | 90% | **0.5** | **91%** 8 |

As can be shown, our accuracy results are competitive and pretty much like TimeGAN. But our method has significant advantage in running time, this factor is very important in real life work.

# Summary

<need to add summary here>

# Appendix 1 – Train Code flow – HMNIST

1. Create GP-VAE model
   1. VAE part:
      1. Latent dim = 256
      2. Data dim = 784 (28x28 image)
         1. Conv1d 256x256, kernel size=3
         2. Dense+Relu 256 iii. Decoder = Bernulli decoder

1. Dense 256  256 2. Dense 256  256 3. Dense 256  256

4. Dense 256  784 iv. Image preprocessor: sequential:

1. Conv2d, kernel size=3x3 2. Conv2d, kernel size=3x3

* 1. GP-VAE part:
     1. Encoder = BandedJointEncoder (Specific to GP-VAE!!)
     2. Kernel = Cauchy iii. Sigma = 1

1. Train using ELBO – can also do IWAE (importance weighted AE - https://arxiv.org/abs/1509.00519)
   1. Get X (Masked data) and m\_mask (Mask Boolean)
   2. Pz = get\_prior (From GP\_VAE)
      1. Calculate a kernel matrix (only once) - (256, 10, 10)
      2. Returns a distribution object MultivariateNormalFullCovariance
   3. Qz\_x = encode(X)
      1. Preprocess x (2d convnet) : (64, 10, 784)  (64, 10, 768)
      2. Activate banded encoder on preprocessed x:
         1. Pass X through VAE encoder, get mapped as output: (64, 10, 784) 

(64, 10, 768) (batch size, time range, sample dim)

* + - 1. Get mu and sigma
         1. Transpose output to (batch\_size, dim, time)
         2. Split to:

mapped\_mean (64, 256, 10)

mapped\_covar(64, 512, 10)

* + - * 1. Pass mapped\_covar through softmax activation function

(sigmoid if Physionist dataset)

1. Obtain covariance matrix from precision one
   * + - 1. Reshape mapped to (64, 256, 20) (batch\_size, z\_size,

2\*time\_length)

* + - * 1. Prepare some indexes matrix (311296, 4)
        2. Create a sparse matrix prec\_sparse using the indexes and data from reshaped mapped omitting the last layer (64, 256, 10, 10)
        3. Create matrix prec\_tril (64, 256, 10, 10)
        4. Get matrix cov\_tril (64, 256, 10, 10)

i. Lower triangular matrix????? Related to Cholesky???)

f. Transpose the new matrix to cov\_tril\_lower (64, 256, 10, 10)

1. Return a distribution object:

MultivariateNormalTriL(loc=mapped\_mean, scale\_tril=cov\_tril\_lower)

1. Z = sample from Qz\_x distribution (vector 64, 256, 10)
2. px\_z = self.decode(z)
   1. Transpose z  (64, 10, 256)
   2. Pass through decoder NN  mapped (64, 10, 784)
   3. Return a sample from a distribution object: Bernoulli(logits=mapped)
3. Calculate cost function
   1. Calculate Negative Log Likelihood – The log probability of the observed samples according to p(x|z)
      1. nll = -px\_z.log\_prob(x)  (64, 10, 784)
      2. replace infinite elements of nll wtih zeros
      3. replace masked elements in nll with zeros based on m\_mask
      4. nll = tf.reduce\_sum(nll, [1, 2])  shape=(M\*K\*BS) (64) ii. Calculate kl\_divergence(qz\_x, pz) – Either analytically or with monte-carlo sampling  shape=(M\*K\*BS, TL or d) (64, 256)
      5. Calc KL divergence
      6. Replace infinite with zeros
      7. Sum over z dimension – get one number per sample  (M\*K\*BS) = (64) iii. Calc elbo
      8. elbo = -nll - self.beta \* kl  shape=(M\*K\*BS) K=1
      9. elbo = tf.reduce\_mean(elbo)  scalar

1. Optimizer step