Advanced Machine Learning - Final Project

Submitted by:
Yaniv Tal 031431166
Miriam Horovicz 313246704

Table of Contents

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.

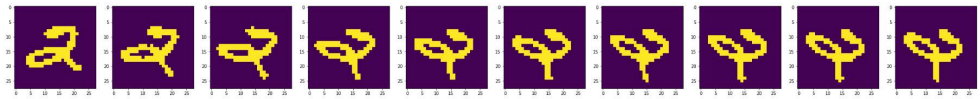
Our objective in this final project is taking GP-VAE that was originally introduced for time-series imputation task and use it for generating high quality time series data. As spatial and temporal correlation already built in its architecture, we believe that it can be a good fit for synthetic data generation 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

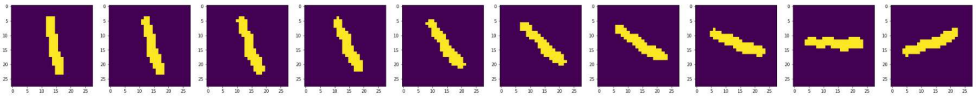
The data to be used in adjusted Healing – MNIST. The original Healing – MNIST is creating series of digits images with rotation to simulate time-series data. The label used for each series is the digit (10 options). We took it one step further and adjusted it for including both the spatial & temporal 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 Papers (Updated per feedback comments)

In this work we encountered and read many papers related to the subject – some through common applications, some through common models, some otherwise. Due to the size limit of this paper, we decided to describe some of the more relevant / interesting ones.

Time-series Generative Adversarial Networks (TimeGAN)

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada

In this paper, the authors use generative adversarial networks to train a generative model that takes into account temporal relationship between samples. The problem formulation looks at each sample as consisting of two attribute sets: $(S, X_{1:T})$, where S is the set of "static" attributes, and X_t is the set of attributes that changes over time.

The model combines two objectives:

- The first tries to learn the probability $\hat{p}(S, X_{1:T})$ by minimizing the distance between probabilities: $\min_{\hat{p}} \mathcal{D}(p(S, X_{1:T}) || \hat{p}(S, X_{1:T}))$.
- The second uses the conditional decomposition $\hat{p}(S, X_{1:T}) = \hat{p}(S) \prod \hat{p}(x_t | S, X_{1:t-1})$ and minimizes the distance between probabilities: $\min_{\hat{p}} \mathcal{D}(p(S) \prod p(x_t | S, X_{1:t-1}) || \hat{p}(S) \prod \hat{p}(x_t | S, X_{1:t-1}))$

The model itself consists of four network components - embedding function, recovery function, sequence generator, and sequence discriminator, which are trained together. The embedding and recovery functions convert samples between their original and latent representations, and the generator / discriminator are the "Classical" GAN components and operate on the latent space sample representations.

The model defines several loss components as following:

- $\mathcal{L}_R$ - Reconstruction loss – The ability to convert samples accurately between the data and latent spaces
- $\mathcal{L}_U$ – Unsupervised loss - The log likelihood to provide correct classifications for a sample (either generated or real embedded
- $\mathcal{L}_S$ – Supervised loss – The ability to predict correctly the next-timestep temporal attributes of a sample given the static attributes and previous temporal attributes: $\hat{p}(x_t | S, X_{1:t-1})$

The embedding and recovery networks are trained using: $\min_{\theta_e, \theta_r} (\lambda \mathcal{L}_S + \mathcal{L}_R)$

The Generator and Discriminator networks are trained using: $\min_{\theta_g} \left(\eta \mathcal{L}_S + \max_{\theta_d} \mathcal{L}_U \right)$

Unsupervised data imputation via variational inference of deep subspaces

Adrian V Dalca, John Guttag, and Mert R Sabuncu. arXiv preprint arXiv:1903.03503, 2019
In this work, they introduced a general probabilistic model that describes sparse high dimensional imaging data as being generated by a deep nonlinear embedding. They derive a learning algorithm using a variational approximation based on convolutional neural networks and discuss its relationship to linear imputation models, the variational auto encoder, and deep image priors. They introduce sparsity-aware network building blocks that explicitly model observed and missing data. They analyze proposed sparsity-aware network building blocks, evaluate the method on public domain imaging datasets, and conclude by showing that the method enables imputation in an important real-world problem involving medical images.

Gain: Missing data imputation using generative adversarial nets.

Jinsung Yoon, James Jordon, and Mihaela Van Der Schaar. International Conference on Machine Learning, 2018.
They propose a novel method for imputing missing data by adapting the well-known Generative Adversarial Nets (GAN) framework. Accordingly, they call the method Generative Adversarial Imputation Nets (GAIN). The generator (G) observes some components of a real data vector, imputes the missing components conditioned on what is observed, and outputs a completed vector. The discriminator (D) then takes a completed vector and attempts to determine which components were observed and which were imputed. To ensure that D forces G to learn the desired distribution, they provide D with some additional information in the form of a hint vector. The hint reveals to D partial information about the missingness of the original sample, which is used by D to focus its attention on the imputation quality of components. This hint ensures that G does in fact learn to generate according to the true data distribution. They tested the method on various datasets and found that GAIN significantly outperforms state-of-the-art imputation methods.
The big advantage of GP-VAE compared to above papers in the GP part as none of these methods explicitly take the temporal dynamics of time series data into account

Transformers in Time Series: A Survey

Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, Liang Sun.
An overview paper which reviews existing transformers model architectures for different time-series related applications.

Priors in Bayesian Deep Learning

Vincent Fortuin.
This paper discusses the importance of a good choice of prior in Bayesian models and the role of the prior in the model, and reviews some prior families.

A review of irregular time series data handling with gated recurrent neural networks.

Philip B. Weerakody, Kok Wai Wong, Guanjin Wang, Wendell Ela.
This paper discusses the use of Gated RNN models (GRU, LSTM etc.) for modeling irregular / sparse time series data, including applications such as imputation.

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:

(i) 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.

(ii) 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.

(iii) 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.

(iv) 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.

(v) 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 X $\in \mathbb{R}^{T\times d}$ with T data points $x_t = \left[x_{t1}, \dots, x_{tj}, \dots, x_{td}\right]^T \in \mathbb{R}^d$ that were measured in T consecutive time points $\tau = [\tau_1, \dots, \tau_T]$. Also assume that each data point x_t could have missing (Unknown) values. We can mark the observed value of data point x_t as $x_t^0 := \left[x_{tj} | x_{tj} is\ observed\right]$, and the missing values as $x_t^m := \left[x_{tj} | x_{tj} is\ missing\right]$. Given that, the imputation problem can be formulated as finding $p(x_t^m | x_{1:T}^0)$.

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 $X \in \mathbb{R}^{T\times d}$ with missing data into a latent data plane $Z \in \mathbb{R}^{T\times k}$, $k < d$ 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 $p(z_{1:T,j}|X_{1:T}^0)$ with a multivariate Gaussian variational distribution $q\left(z_{1:T,j}\middle|X_{1:T}^0\right) = \mathcal{N}(m_j, \Lambda_j^{-1})$, 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_j parameterized as a product of bidiagonal matrices such that:

$$\Lambda_j \coloneqq B_j^T B_j, \quad \{B_j\}_{tt'} = \begin{cases} b_{tt'}^j, & if\ t' \in \{t, t+1\} \\ 0, & otherwise \end{cases}$$

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:

$$k_{Cau}(\tau, \tau') =' sigma^2 \left(1 + \frac{(\tau - \tau')^2}{l^2}\right)^{-1}$$

Cost function

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

$$\log p(X_0) \geq \sum_{t=1}^{T} E_{q_\psi(z_t|x_{1:T})}\left[logp_\theta(x_t^0|z_t)\right] - \beta D_{kl}[q_\psi(z_{1:T}|x_{1:T}) \parallel p(z_{1:T})]$$

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_{kl}).

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:

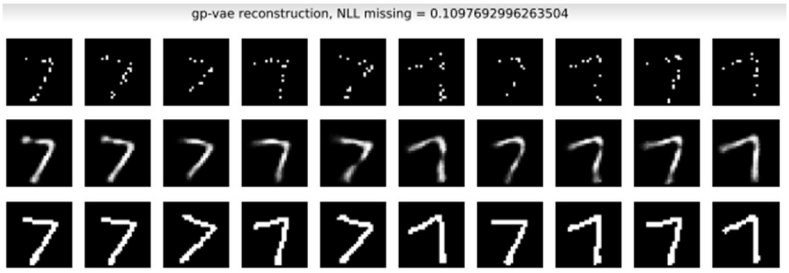
1. Download the notebook from https://github.com/mryanivtal/aml_final_project and place it in your google drive
2. Download the git repo from https://github.com/ratschlab/GP-VAE and place it in your google drive
3. Follow the notebook instructions

Anchor paper experiments and results

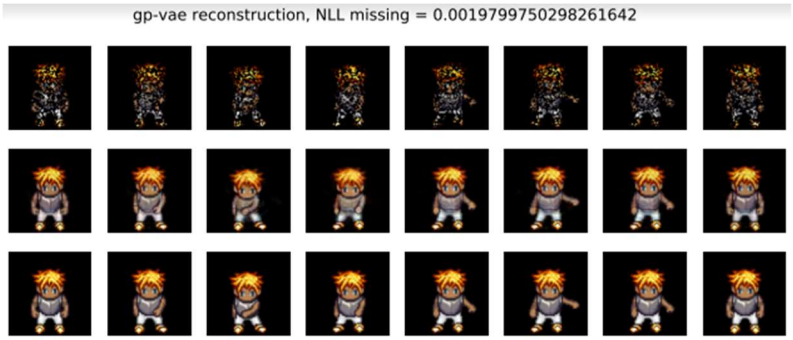
The writers of the paper tested their model on three datasets: Healing MNIST, Sprites, Real medical time series data. Since all parameters for the original run were provided, we have used the exact same ones as the original implementation, and got very similar results, as following:

	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 - training deep learning model with limited data. We will use GP-VAE for creating synthetic data to enrich and diversify our small 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, but an intermediate synthetic data generation step for improving our time-series classification model results.

The innovative part here is using a GP-VAE architecture originally built for imputation, and leverage it for a different task of synthetic data generation. This is a very different approach from the classical synthetic data generation that 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 from an operative standpoint. As the classical methods learn data distribution 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, and aims to represent the change over time of the sample

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 with generated data	Train run time (Hours)	Accuracy with generated data2	Run time
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

Summary of the effort

In this project, we have learned fascinating and new (to us) concepts and approaches to time series and time series data imputation. Main efforts:

- Learn the concept of Gaussian process and GP regression, it's applications and variants
- Learn the GP-VAE and TimeGAN models thoroughly
- Update the HMNIST dataset generation code and MNAR (Masking Not at Random) code to suit our needs, learned some Tensorflow v.1 on the fly
- Update the GP-VAE code for training and generation based on custom masks
- Write a very basic time series HMNIST classifier

Bottom line, we managed to improve our classifier performance by generating synthetic data using the GP-VAE and the original small train dataset, getting comparable results to a competitive approach with much shorter run time, which is better than we hoped for.

Open questions and future direction for further research

Due to the timeline and scope of this project, many questions were left open. Some questions we would like to investigate in future research:

- **Tweaking the generator model and the mask to generate "Noisier" or different samples** – Although we did try several configurations, it would be interesting to learn the effect of modifying generator model parameters (add randomness to the sample in the latent space before reconstruction, use MNAR masks with specific patterns for more generation "freedom" etc.)
- **VAE vs. GP-VAE in time series data generation** – What is the added value of capturing the temporal correlations in the data for synthetic data generation? It would be interesting to run the same test CAE vs. GP-VAE and quantify the added value of the Gaussian Process addition
- **Performance on different datasets** – it would be interesting to see the generation work on different real-world datasets. This is challenging because of the model assumption that there is a "Real" sample state represented in the latent space – such labeled dataset for training will be very difficult to come by.
- **Time series data distribution distance metrics** – How can you tell if a synthetically generated time-series dataset is "good"? It would be very interesting to learn and test the effect of different metrics for the quality of synthetically generated time series data – distance measures etc.

Personal note

This project was our first "Open ended" research experience.
Although limited in scope, it was extremely interesting, and also truly challenging on several fronts.
We thank you for the opportunity and time investment in this.

Appendix 1 – Train Code flow – HMNIST

```
Get x_seq, m_seq (Masked data, Mask Boolean) batch
loss = model.compute_loss(x_seq, m_mask=m_seq)
        tile x K*M times
        tile m K*M times

        pz = self._get_prior()                  # from GP_VAE
                # runs only once and keep the value for later usage
                Calculate a (Cauchi) kernel_matrix (10, 10)
                Returns a distribution object MultivariateNormalFullCovariance with mean=0 and cov=kernel_matrix

        qz_x = self.encode(x)
                Preprocess x (2d convnet) : (64, 10, 784) --> (64, 10, 768)
                        # preprocess runs on each 28x28 image separately (No time dimension!)

                return self.encoder(x)
                        # apply banded encoder (conv1d, dense, dense) on preprocessed x and return result
                        Pass X through VAE encoder, get mapped as output:
                                (64, 10, 784) --> (64, 10, 768) (batch  size, time range, sample dim)
                        Transpose output to (batch_size, dim, time)
                        Get mu and covar from output split:
                                mapped_mean (64, 256, 10)
                                mapped_covar(64, 512, 10)
                        Pass mapped_covar through softmax activation function (sigmoid if Physionist dataset)

                        Obtain covariance matrix from precision one
                                mapped_reshaped = mapped_covar reshaped to
                                        (64, 256, 20) (batch_size, z_size, 2*time_length)
                                Prepare some indexes matrix (311296, 4)
                                Create a sparse matrix prec_sparse using the indexes
                                        and data from mapped_reshaped omitting the last layer (64, 256, 10, 10)
                                Create matrix prec_tril (64, 256, 10, 10)
                                Get matrix cov_tril (64, 256, 10, 10) by solving the prec_tril linear equation set
                                cov_tril - replace infinites and zeros
                                cov_tril_lower = Transpose cov_tril to (64, 256, 10, 10)

                z_dist = a distribution object: MultivariateNormalTriL(loc=mapped_mean, scale_tril=cov_tril_lower)
                return z_dist

        Z = sample from qz_x distribution (vector 64, 256, 10)

        px_z = self.decode(z)                   # GP-VAE.decode
                Transpose z --> (64, 10, 256)
                Pass z through decoder NN --> mapped (64, 10, 784)
                Return a Bernoulli distribution object:  Bernoulli(logits=mapped)
```

```
        ########################
        # Summary till now:
        # pz - p(z) prior - MultivariateNormalFullCovariance dist object with mean=0 and cov=cauchi kernel matrix
        # qz_x - q(z|x) - MultivariateNormalTriL dist object - with mean and cov based on x encoded by VAE NN
        # z - sample vector from qz_x dist
        # px_z - p(x|z) posterior - Bernoulli dist object (Because the pixels are either 1 or 0)
                with logits = output of z decoded by decoder NN
        ########################


        nll = -px_z.log_prob(x)  # shape=(M*K*BS, TL, D)
                # negative log likelihood - The log probability of the observed samples x according to p(x|z)

        # Calculate Negative Log Likelihood -log(p(X=x|z)) –
        # The log probability of the observed samples x according to p(x|z)
                nll = -px_z.log_prob(x)  # shape=(M*K*BS, TL, D) (64, 10, 784)
                replace infinite elements of nll wtih zeros
                replace masked elements in nll with zeros based on m_mask to remove them from training cost
                nll = tf.reduce_sum(nll, [1, 2])  #shape=(M*K*BS) (64)

        # Calculate kl_divergence(qz_x, pz) – Can calculate either analytically or with monte-carlo sampling
                kl = self.kl_divergence(qz_x, pz)  # Calc KL divergence: shape=(M*K*BS, TL or d)  (64, 256)
                Replace infinites and zeros
                Sum over z latent dimension – get one number per sample - shape=(M*K*BS) = (64)

        # Calc elbo
                elbo = -nll - self.beta * kl   #  shape=(M*K*BS) K=1
                elbo = tf.reduce_mean(elbo)  # scalar

        return -elbo
```

Optimizer step update grads etc.