Advanced Machine Learning - Final Project

Part 1 – Anchor paper

Yaniv Tal 031431166
Miriam Horovicz 313246704

Contents

Part 1 – Anchor paper 2
Overview - Problem and Existing Solution Approaches 2
Anchor Paper summary 3
Problem formulation 3
Model Overview 3
Cost function 3
Related Papers (“River of citations”) 4
Anchor paper implementation and results 5
Our Implementation 5
Anchor paper experiments and results 6
Appendix 1 – Train Code flow – HMNIST 7

Part 1 – 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 $x_t = [x_{t1}, \dots, x_{tj}, \dots, x_{td}]^T \in \mathbb{R}^d$ 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 $x_t^0 := [x_{tj}|x_{tj} is\ observed]$, and the missing values as $x_t^m := [x_{tj}|x_{tj} is\ missing]$. 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 $q(z_{1:T,j}|X_{1:T}^0) = \mathcal{N}(m_j, \Lambda_j^{-1})$, 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:

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

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:

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

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}).

Related Papers ("River of citations")

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.

Adrian V Dalca, John Guttag, and Mert R Sabuncu. Unsupervised data imputation via variational inference of deep subspaces. 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.

Jinsung Yoon, James Jordon, and Mihaela Van Der Schaar. Gain: Missing data imputation using generative adversarial nets. 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

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

Vincent Fortuin. Priors in Bayesian Deep Learning.
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.

Philip B. Weerakody, Kok Wai Wong, Guanjin Wang, Wendell Ela. A review of irregular time series data handling with gated recurrent neural networks.
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 implementation and results

Our Implementation

The paper provides a link to the authors git repository where their original code can be found.
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. In order to run it:

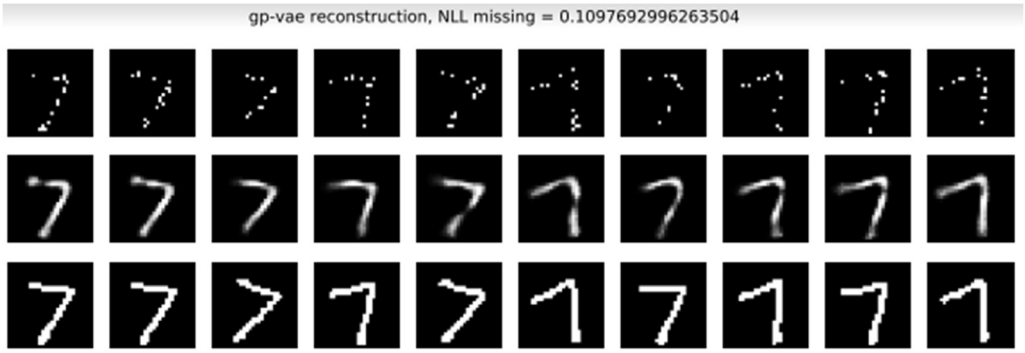
1. Download the notebook from https://github.com/mryanivtal/aml_final_project and place it in your google drive
2. Download the git repo from https://github.com/ratschlab/GP-VAE and place it in your google drive
3. Follow the notebook instructions

Anchor paper experiments and results

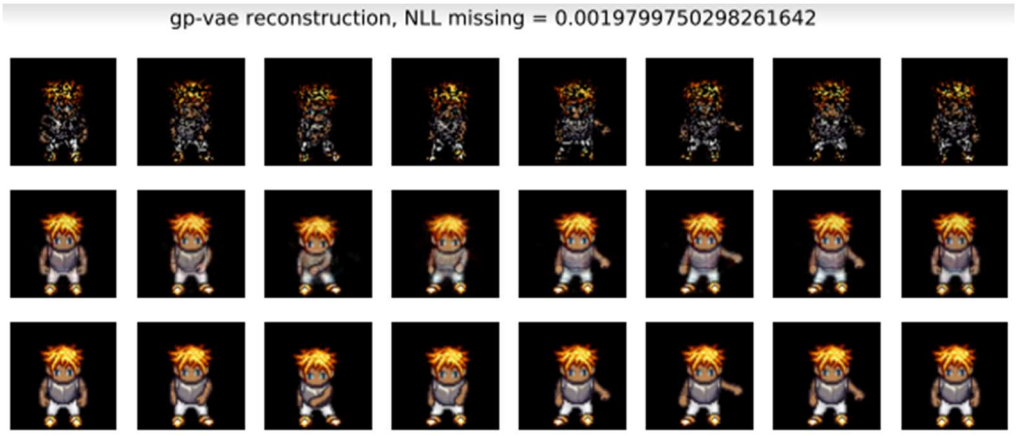
The writers of the paper tested their model on three datasets: Healing MNIST, Sprites, Real medical time series data. Since all parameters for the original run were provided, we have used the exact same ones as the original implementation, and got very similar results, as following:

	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):



Appendix 1 – Train Code flow – HMNIST

1. **Create GP-VAE model**
    a. VAE part:
        i. Latent dim = 256
        ii. 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
    b. GP-VAE part:
        i. Encoder = BandedJointEncoder **(Specific to GP-VAE!!)**
        ii. Kernel = Cauchy
        iii. Sigma = 1

2. **Train using ELBO – can also do IWAE (importance weighted AE - https://arxiv.org/abs/1509.00519)**
    a. Get X (Masked data) and m_mask (Mask Boolean)
    b. Pz = get_prior (From GP_VAE)
        i. Calculate a kernel matrix (only once) -→ (256, 10, 10)
        ii. Returns a distribution object MultivariateNormalFullCovariance
    c. Qz_x = encode(X)
        i. **Preprocess x (2d convnet) : (64, 10, 784) → (64, 10, 768)**
        ii. **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)
            2. Get mu and sigma
                a. Transpose output to (batch_size, dim, time)
                b. Split to:
                    i. mapped_mean (64, 256, 10)
                    ii. mapped_covar(64, 512, 10)
                c. Pass mapped_covar through softmax activation function (sigmoid if Physionist dataset)
            3. Obtain covariance matrix from precision one
                a. Reshape mapped to (64, 256, 20) (batch_size, z_size, 2*time_length)
                b. Prepare some indexes matrix (311296, 4)

c. Create a sparse matrix prec_sparse using the indexes and data from reshaped mapped omitting the last layer (64, 256, 10, 10)
d. Create matrix prec_tril (64, 256, 10, 10)
e. 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)

4. Return a distribution object: MultivariateNormalTriL(loc=mapped_mean, scale_tril=cov_tril_lower)

d. Z = sample from Qz_x distribution (vector 64, 256, 10)

e. px_z = self.decode(z)

i. Transpose z → (64, 10, 256)
ii. Pass through decoder NN → mapped (64, 10, 784)
iii. Return a sample from a distribution object: Bernoulli(logits=mapped)

f. Calculate cost function

i. **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)**
    1. Calc KL divergence
    2. Replace infinite with zeros
    3. Sum over z dimension – get one number per sample → (M*K*BS) = (64)
iii. **Calc elbo**
    1. elbo = -nll - self.beta * kl → shape=(M*K*BS) K=1
    2. elbo = tf.reduce_mean(elbo) → scalar

g. Optimizer step