

# NJ-ODE for classification problems

Emanuele G., Sophia M., Genc K. and Victoria D.

June 28, 2021

# Table of contents

1. Neural Jump Ordinary Differential Equation
2. NJ-ODE for classification
3. Results

## Neural Jump Ordinary Differential Equation

## Problem statement

Our work is based on the paper : *Neural Jump Ordinary Differential Equations: Consistent Continuous-Time Prediction and Filtering* by C. Herrera and F. Krach and J. Teichmann [1].

- Forecasting of temporal data. The observed data comes from a Markovian stochastic differential equation (SDE) of the form:

$$dX_t = \mu(t, X_t)dt + \sigma(t, X_t)dW_t, \quad (1)$$

at irregularly-sampled time points.

- With respect to the  $\mathcal{L}^2$ -norm, the optimal prediction of future value of  $(X_t)_t$  is the conditional expectation given the available information.
- Because of the Markovian property, this is equivalent to look for:

$$f(x_t, t, x) = \mathbb{E}[X_{t+s} | X_t = x_t], \quad s \geq 0. \quad (2)$$

## Usual approach

- ▶ **Monte Carlo** If good estimates of the drift and diffusion are available, then the conditional expectation can be approximated by a Monte Carlo simulation.

$$\mathbb{E}[X_{t+1}|X_t] \approx X_t + \frac{1}{n} \sum_{i=1}^n \hat{\mu}(t^i, X_t^i) \Delta t + \frac{1}{n} \sum_{i=1}^n \hat{\sigma}(t^i, X_t^i) \Delta W_t^i. \quad (3)$$

*Problem:* Good estimates are usually not available, and are not needed here since we are interested in forecasting instead of sampling new paths.

- ▶ **Recurrent Neural Networks** Neural network updates a latent variable with observations of a discrete input time-series. Successful for regularly sampled data.

$$h_{n+1} = \text{RNNCell}(h_n, x_{n+1}), \quad y_n = \text{outputNN}(h_n). \quad (4)$$

*Problem:* Here, we are interested in irregularly sampled data.

# Neural Ordinary Differential Equation

- ▶ Neural ODEs are a family of continuous-time models defining a latent variable  $h_t = h(t)$  to be the solution to an ODE initial value problem:

$$h_t = h_0 + \int_{t_0}^t f(h_s, s, \theta) ds, \quad t \geq t_0, \quad (5)$$

where  $f(\cdot, \cdot, \theta) = f_\theta$  is a neural net with weights  $\theta$ .

- ▶ Latent variable can be updated continuously by solving the ODE:

$$h_t = \text{ODESolve}(f_\theta, h_0, (t_0, t)). \quad (6)$$

# ODE-RNN

- ▶ **Problem:** We are interested in continuous time series. In particular, we want to have an output stream that is generated continuously in time.
- ▶ **Idea:**
  - ▶ **ODE** We use a neural ODE between two observations.
  - ▶ **RNN** At the next observation time  $t_i$ , the latent variable is updated by an RNN with the new observation  $x_i$ .
  - ▶ Fixing  $h_0$ , the entire latent process can be computed by iteratively solving an ODE followed by applying an RNN.

$$\begin{cases} h'_t = \text{ODESolve}(f_\theta, h_{i-1}, (t_{i-1}, t)) \\ h_{t_i} = \text{RNNCell}(h'_i, x_i) \end{cases} \quad t_{i-1} < t < t_i. \quad (7)$$

# Neural Jump ODE

- ▶ **Markovian Paths**  $(X_t)_t$  is a Markovian process. Thus, the optimal prediction of the future state of  $X$  only depends on the current state rather than full history.
- ▶ **JumpNN** Because of Markovian property, the hidden state can solely be computed from the last observation, so we can replace RNN by regular neural network.
- ▶ Theoretical analysis suggests that time increment  $(t_i - t_{i-1})$  should be used, and the last observation should be part of the input.
- ▶ The architecture we use is hence

$$\hat{X}_t = \begin{cases} h'_t = \text{ODESolve}(f_\theta, (h_{i-1}, x_{i-1}, t_{i-1}, t - t_{i-1}), (t_{i-1}, t)) \\ h_{t_i} = \text{jumpNN}(x_i). \end{cases} \quad (8)$$



# NJ-ODE Algorithm

**Data:** observations with timestamps  $\{(x_i, t_i)\}_{i=0,\dots,n}$

**for**  $i = 0$  **to**  $n$  **do**

$h_{t_i} = \text{jumpNN}(x_i)$

$y_{t_i} = \text{outputNN}(h_{t_i})$

$s \leftarrow t_i$

**while**  $s + \Delta t \leq t_{i+1}$  **do**

$h_{s+\Delta t} = \text{ODESolve}(f_\theta, (h_s, x_i, t_i, s - t_i), (s, s + \Delta t))$

$y_{s+\Delta t} = \text{outputNN}(h_{s+\Delta t})$

$s \leftarrow s + \Delta t$

**end**

**end**

where  $\Delta t$  is a fixed step size and  $t_{n+1} = T$ .

# Training

- ▶ Training set: observations of  $N$  independent realization of  $d$ -dim. stochastic process  $X$ .

Realisation  $j$ : observed at times  $t_1^{(j)}, \dots, t_{n_j}^{(j)}$  with values  $x_1^{(j)}, \dots, x_{n_j}^{(j)} \in \mathbb{R}^d$ .

- ▶ We train the model with as a loss function

$$L_N(\theta) = \frac{1}{N} \sum_{j=1}^N \frac{1}{n_j} \sum_{i=1}^{n_j} \left( |x_i^{(j)} - y_i^{(j)}| + |y_i^{(j)} - y_{i-}^{(j)}| \right)^2. \quad (9)$$

The **blue part** forces the jumpNN to output good updates.

The **orange part** forces the jumps sizes to be small.

# Theoretical convergence guarantee

- ▶ Assume that for each number of paths  $N$  and for each size  $M$  of networks we chose assume that the weights  $\theta_M$  are chosen optimally in order to minimize  $L_N(\theta)$ .
- ▶ As  $N, M \rightarrow \infty$ , the output of the NJ-ODE model converges in mean (L1 convergence) to the conditional expectation of  $X$  given the current information.

## Training on synthetic datasets

- ▶ **Evaluation metric** We evaluate the distance of the model output to the conditional expectation when an analytical formula is available.
- ▶ We use an equidistant step size  $\Delta_t := \frac{T}{K}$ ,  $K \in \mathbb{N}$ , true conditional expectation  $\hat{x}_t^{(j)}$ , predicted conditional expectation  $y_t^{(j)}$ ,  $N_2$  test samples. The evaluation metric is defined as:

$$\text{eval}(\hat{x}, y) := \frac{1}{N_2} \sum_{j=1}^{N_2} \frac{1}{K+1} \sum_{i=0}^K (\hat{x}_{i\Delta_t}^{(j)} - y_{i\Delta_t}^{(j)})^2. \quad (10)$$

- ▶ **Black-Scholes, Ornstein-Uhlenbeck and Heston.** They generate  $N = 20'000$  paths on the time interval  $[0, 1]$  using the Euler-scheme with 100 time steps. 10% of the grid points are randomly chosen as observation times. Training set represents 80% of the data.

# Results

During training, the relative difference becomes very small, hence the true conditional expectation is nearly replicated.

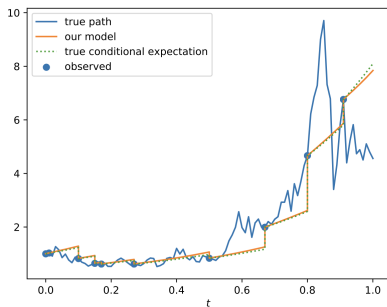


Figure 1: Predicted and true conditional expectation on a test sample of the Heston dataset.

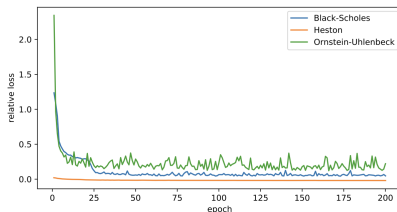


Figure 2: The relative difference between the loss function computed with the predicted conditional expectation and the loss function computed with the true conditional expectation is shown with respect to the number of epochs.

# Dealing with incomplete observations

- ▶ Suppose observation  $i$  is incomplete: we are missing some coordinates of the vector  $x_i \in \mathbb{R}^d$
- ▶ Use a **mask**  $m_i \in \{0, 1\}^d$ :  $m_i^k = 1$  if  $k$ -th coordinate of  $x_i$  is observed,  $m_i^k = 0$  otherwise.
- ▶ Imputed observation vector is

$$\tilde{x}_i := m_i \odot x_i + (1_d - m_i) \odot y_{i-}. \quad (11)$$

- ▶ Input becomes  $(\tilde{x}_i, m_i)$
- ▶ Output  $y_i$  used in ODE part instead of  $x_i$ , multiply each term in the sum of loss function with the mask.

NJ-ODE for classification

# NJ-ODE for classification

- ▶ **Goal:** Adapt the NJ-ODE model to classify irregularly sampled time series.
- ▶ **Our work:** During our project we concentrated on the following three data sets.



# Data sets

- ▶ One-dimensional geometric Brownian motion with different volatility coefficients.
- ▶ Multi-dimensional time series with incomplete observations:
  - i Multi-dimensional (correlated coordinates) geometric Brownian motion with different volatility coefficients.
  - ii Sepsis data set.

## Changes to the model: classification readout

- ▶ We want the model to predict out of the hidden state  $h$  a class probability.
- ▶ We added a new readout map  $\text{ClassificationOutputNN}(\cdot)$  with the same architecture as  $\text{OutputNN}(\cdot)$ .
- ▶ The class probability at final time  $T$  is then given by  $p = \text{ClassificationOutputNN}(h_T)$ .

## Changes to the model: loss function

- ▶ Training set: observations of  $N$  independent realization of  $d$ -dim. stochastic processes  $X_1, \dots, X_k$ .  
Realization  $j$  of process  $X_l$ : observed at times  $t_1^{(j)}, \dots, t_{n_j}^{(j)}$  with values  $x_1^{(j)}, \dots, x_{n_j}^{(j)} \in \mathbb{R}^d$ , and label  $l_j = l$ .
- ▶ We added a classification part to the loss in (9) and obtained

$$L_N(\theta) = \frac{1}{N} \sum_{j=1}^N \frac{1}{n_j} \sum_{i=1}^{n_j} \left( |x_i^{(j)} - y_i^{(j)}| + |y_i^{(j)} - y_{i-}^{(j)}| \right)^2 \\ + \text{CrossEntropy}(p_j, l_j).$$

The original loss forces the model to learn a good approximation of  $X_l$ 's.

The classification loss forces the model to distinguish between the different  $X_l$ 's.

## Changes to the model: jumpNN

- ▶ After each new observation  $x_i$  the value of the hidden state is updated using both  $x_i$  and  $h_{t_i-}$ .
- ▶ Namely, the feed-forward neural network  $\text{jumpNN}(\cdot)$  is replaced by a recurrent network and  $h_{t_i} = \text{jumpRNN}(x_{t_i}, h_{t_i-})$ .
- ▶ This accounts for the fact that volatility is a property of the trajectory, hence the full history of observations is needed for prediction.

## Resulting Model: NJ-ODE for classification

**Data:** observations with timestamps  $\{(x_i, t_i)\}_{i=0,\dots,n}$

```
for  $i = 0$  to  $n$  do  
   $h_{t_i} = \text{jumpRNN}(x_i, h_{t_i-})$   
   $y_{t_i} = \text{outputNN}(h_{t_i})$   
   $s \leftarrow t_i$   
  while  $s + \Delta t \leq t_{i+1}$  do  
     $h_{s+\Delta t} = \text{ODESolve}(f_\theta, (h_s, x_i, t_i, s - t_i), (s, s + \Delta t))$   
     $y_{s+\Delta t} = \text{outputNN}(h_{s+\Delta t})$   
     $s \leftarrow s + \Delta t$   
  end  
end  
 $p = \text{ClassificationOutputNN}(h_T)$ 
```

where  $\Delta t$  is a fixed step size and  $t_{n+1} = T$ .

## Results

# Baseline

- ▶ For the one-dimensional geometric Brownian motion data set, we compared the performances of the NJ-ODE against a random forest classifier.
- ▶ As input for the random forest classifier we extracted features from the observations of the trajectory.
- ▶ We used 7 features, as maximal size of jump, mean size of jump, ...

## Results

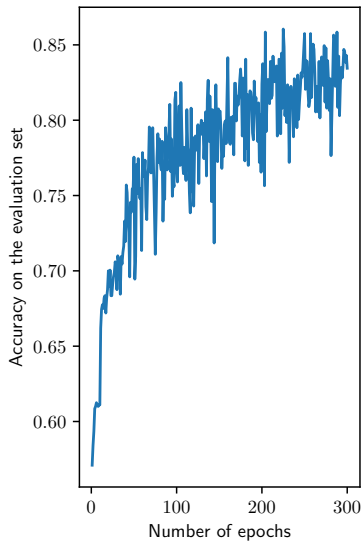
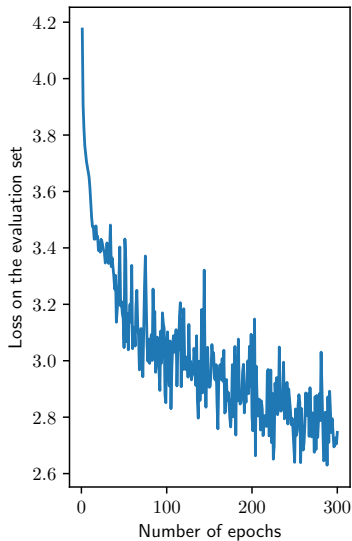
- For one-dimensional geometric Brownian motions with volatility coefficients  $\sigma_1 = 0.6$ ,  $\sigma_2 = 0.3$  we obtained

	Accuracy	AUC
NJ-ODE	0.85	0.92
random forest	0.78	0.81



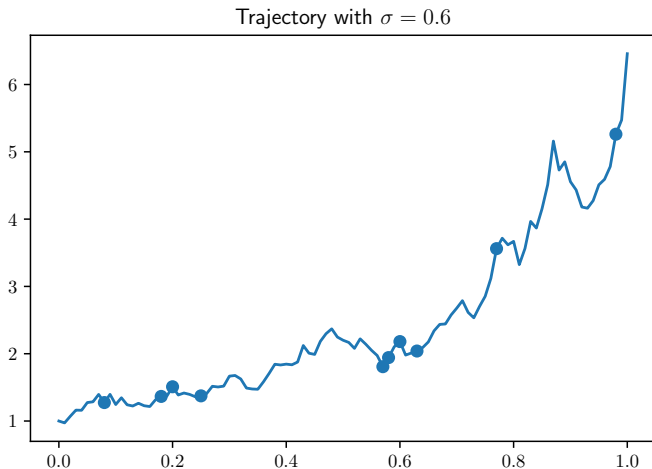
# Training

Performances on the evaluation set



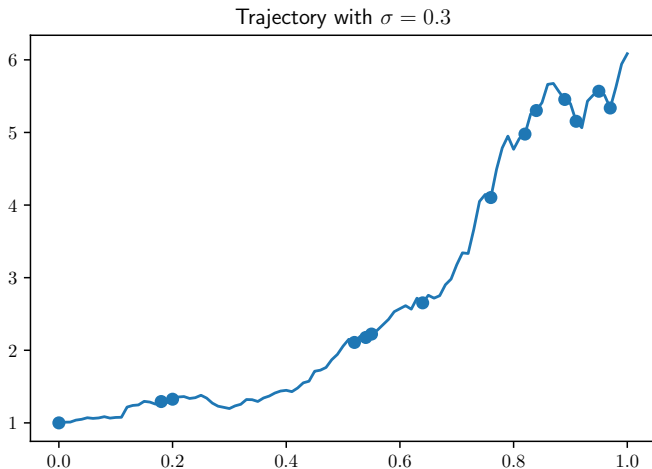
# Errors

- Predicted  $\sigma = 0.3$  with probability 0.85



# Errors

- Predicted  $\sigma = 0.6$  with probability 1



## Sepsis data

- ▶ Early Prediction of Sepsis from Clinical Data – the PhysioNet Computing in Cardiology Challenge 2019
- ▶ Data is sourced from ICU patients in two separate hospital systems. Available patient co-variables consist of Demographics(Age, Gender, ICU unit, ICU length of stay), Vital Signs(Heart rate, Temperature, etc.) and Laboratory values(Calcium, Lactic acid, Hemoglobin, etc.).
- ▶ The goal is to predict whether the patients develop a sepsis during their entire stay at the hospital.

# Pre-processing

- ▶ **Training Data:** This is a dataset of 40335 time series, 34 time-dependent features are included. We only consider the first 72 hours of a patient's stay.
- ▶ **Different Length:** The duration of each patient's stay is quite different. To fit our classification model, we made all time series same length.

## Pre-processing

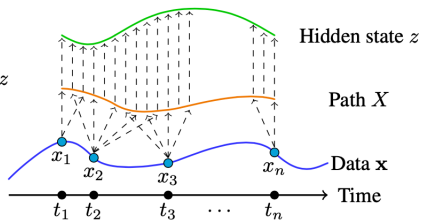
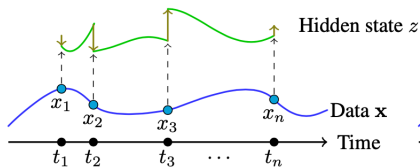
- ▶ **Missing value:** Most values are missing, only 10.3% of value are observed.
- ▶ **Imbalance:** As the dataset is highly imbalanced(5% positive rate), we consider to report AUC rather than accuracy.
- ▶ **Mask:** Since the model might learn to transfer the information about an observed coordinate to an unobserved one, we extend the input to also include the information which coordinates were observed.

# Results

- For comparison, we used an existing model: NeuralCDE from paper **Neural Controlled Differential Equations for Irregular Time Series** [2].



Test AUC		
	Mask	No Mask
NeuralCDE	$0.880 \pm 0.006$	$0.776 \pm 0.009$
NJ-ODE	?	?

# Neural CDE





# Bibliography

-  C. Herrera, F. Krach, and J. Teichmann, “Neural jump ordinary differential equations: Consistent continuous-time prediction and filtering,” 2021.
-  P. Kidger, J. Morrill, J. Foster, and T. Lyons, “Neural controlled differential equations for irregular time series,” 2020.

*Thank You*