Friedrich-Alexander-Universität
Technische Fakultät

Machine Learning Data Analytics

Seminar Advances in Deep Learning for Time Series (ADLTS)

Lecture 4: Time-Aware Models

Naga Venkata Sai Jitin Jami, MSc.
Machine Learning and Data Analytics (MaD) Lab
Friedrich-Alexander-Universität Erlangen-Nürnberg
25.04.2025

Team: FAU & PUCV

Dr. Dario Zanca (FAU)
dario.zanca@fau.de

Naga Venkata Sai Jitin Jami, M. Sc. (FAU)
jitin.jami@fau.de

Dr. Christtoffer Loeffler (PUCV)
christoffer.loffler@pucv.cl

FAU & PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO

Organisational Information

Seminar Advances of Deep Learning for Time Series (MLTS)

- 5 ECTS
- Team-based project *(more details on the second lecture)*
- Evaluation:
  - FAU students: 60% written report, 40% oral presentation
  - PUCV students: 20% code, 40% written report, 40% presentation

Topics overview

Recorded Lectures

I. Introduction
II. The Tool Tracking dataset
III. DL for Time Series
IV. Time-aware models
V. XAI for Time Series - part 1
VI. Active Learning for Time Series - part 1
VII. Semi-supervised Learning
VIII. Domain-shifts, Ethics, and Bias
IX. XAI for Time Series - part 2
X. Active Learning for Time Series - part 2

Topics overview

Recorded Lectures

I. Introduction
II. The Tool Tracking dataset
III. DL for Time Series
IV. Time-Aware Models
V. XAI for Time Series - part 1
VI. Active Learning for Time Series - part 1
VII. Semi-supervised Learning
VIII. Domain-shifts, Ethics, and Bias
IX. XAI for Time Series - part 2
X. Active Learning for Time Series - part 2

Lecture outline

1. Time-Aware Models
2. Ordinary Differential Equations
3. Residual Networks and ODENet
4. Backpropogation for ODENet
5. Applications of ODENet on Time Series

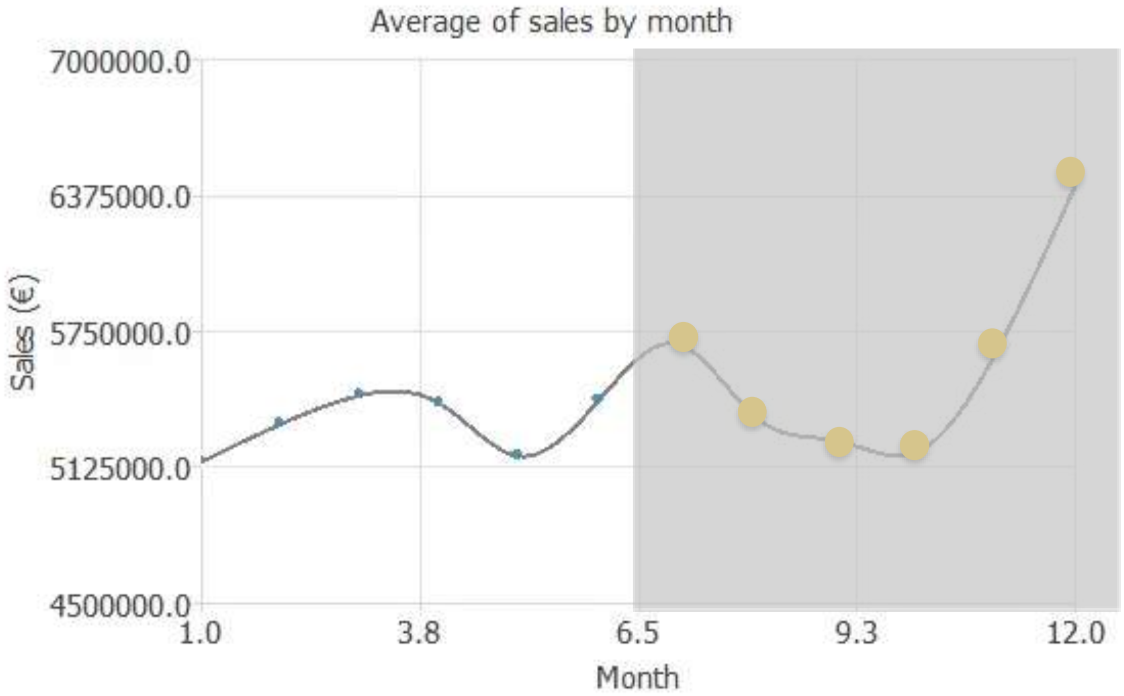
Time Aware Models

Time-Series Modeling

- Time series data is a sequence of data points collected at (usually) consistent time intervals.
- Examples:
  - Weather Data
  - Electricity Consumption
  - Patient vital signs (e.g., blood pressure, heart rate)
  - Equipment maintenance logs
- Time series analysis helps in understanding trends, seasonality, and patterns over time.

Time-Series Forecasting

- Time series forecasting involves predicting future values based on historical data.
- For example, we can predict temperature in a location by considering historical data at the same location (univariate) or surrounding areas (multi-variate).
- Some successful forecasting architectures:
  - Recurrent Neural Nets (RNN)
  - Long Short-Term Memory (LSTM)
  - Gated Recurrent Units (GRUs)
  - Temporal Convolutional Networks (TCNs)
  - Transformer

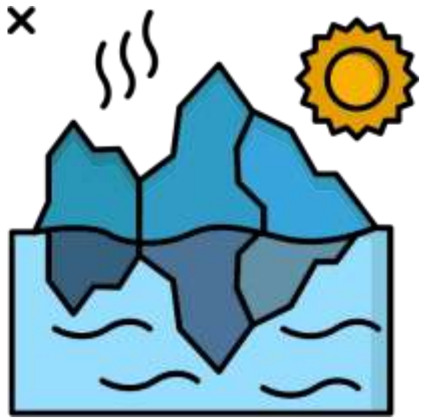


= Collected data

= Predicted by a model

Assumptions in Time-Series Forecasting

- Traditional time series models often rely on certain assumptions about the data.
- These assumptions simplify the modeling process but may not always hold true in real-world scenarios.
- Two major assumptions come to mind:
  - Constant Sampling Interval, i.e., Δt is constant
  - Prediction Interval same as input Δt.



Example: Predicting the melting of ice in the arctic circle through satellite imagery for every day for the next 3 months.

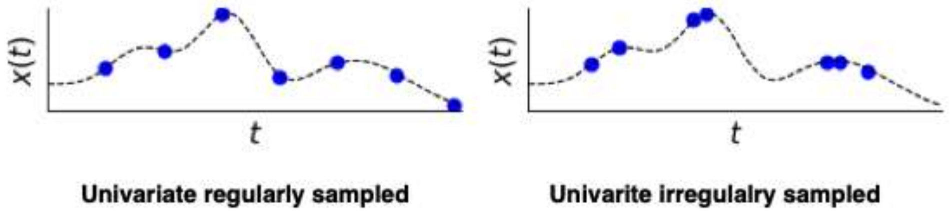
Assumption: Satellite images are recorded at high quality every day.

Reality: There's a quality drop-off from the satellite imagery and is only recorded once a week.

Problem: Data is available every week but we want to make predictions on a daily basis.

Introducing Time Aware Deep Learning models

- There has been plenty of research in irregularly sampled time series forecasting.
- They can largely be divided into the following categories:
  - Discretization: Converting irregular time series problem to a regular time series problem with missing values.
  - Interpolation:
    - Deterministic – Linear or Non-Linear Interpolation with Kernel methods
    - Probabistic – Gaussian Processes
  - Attention based: Transformers
  - Graph based
  - Recurrence: RNN-based, **ODE-based**



In this lecture, we will discuss the paper "[Neural Ordinary Differential Equations]" by Chen et al.

[1] A Survey on Principles, Models and Methods for Learning from Irregularly Sampled Time Series

Ordinary Differential Equations

Ordinary Differential Equations

1st Order ODEs

$$\frac{d\mathbf{x}(\mathbf{t})}{d\mathbf{t}} = f(\mathbf{x}(\mathbf{t}), \mathbf{t}, \theta)$$

- $x(t)$ is the dependent variable.
- t is the independent variable (often representing time).
- $\frac{dx(t)}{dt}$ is the derivative of $x(t)$ with respect to t, representing the rate of change of $x(t)$.
- f is a function that defines how the rate of change of x depends on t, $x(t)$, and possibly other parameters θ.

Ordinary Differential Equations

1st Order ODEs

$$\boxed{\frac{dx(t)}{dt} = r \cdot x(t)}$$

- $x(t)$ is the amount of money at time t.
- $\frac{dx(t)}{dt}$ is the rate of change of the amount of money with respect to time t.
- r is the interest rate, which is a constant parameter (θ).
- The function $f(x(t), t, \theta)$ from the general form is $r \cdot x(t)$.

Machine Learning Data Analytics FAU

Initial Value Problem

$$\frac{dx(t)}{dt} = f(x(t), t, \theta);\ x(t_0) \text{ is given; } x(t_1) = ?$$

Solution: Analytical Integration

$$x(t_1) = x(t_0) + \int_{t_0}^{t_1} f(x(t), t, \theta)dt$$

Compound Interest Solution

Can't be integrated?

$$x(t_1) = x(t_0) + \int_{t_0}^{t_1} r \cdot x(t)dt$$

$$\frac{dx(t)}{dt} = r \cdot x(t)$$

$$\implies \frac{1}{x}\frac{dx}{dt} = r \implies \int \frac{1}{x}\frac{dx}{dt} = \int rdt$$

$$\implies ln(x) = rt + C \implies x = e^{rt+C}$$

$$\implies x(t) = Ke^{rt} \implies x(t=0) = K$$

Suppose initial investment is x(0) = K = $20000 and r = 10% year:

$$x(t=1) = x(t=0) + \int_0^1 r \cdot x(t)dt$$

$$x(t=1) = x(t=0) + \int_0^1 r \cdot Ke^{r \cdot t}dt$$

$$x(t=1) = x(t=0) + r \cdot K \cdot \int_0^1 e^{r \cdot t}dt$$

$$x(t=1) = x(t=0) + r \cdot K \cdot (e^r - e^0)$$

$$x(t=1) = 20000 + 0.1 \times 20000 \times e^{0.1}$$

$$x(t=1) \approx 22103$$

Initial Value Problem

$$\frac{dx(t)}{dt} = f(x(t), t, \theta);\ x(t_0) \text{ is given; } x(t_1) = ?$$

Solution

Can't be integrated?

$$x(t_1) = x(t_0) + \int_{t_0}^{t_1} f(x(t), t, \theta)dt$$

Approximations to the integral i.e. **Numerical Integration**:

- Euler Method
- Runge-Kutta methods
- …

Initial Value Problem

$$\frac{dx(t)}{dt} = f(x(t), t, \theta);\ x(t_0) \text{ is given; } x(t_1) = ?$$

Solution

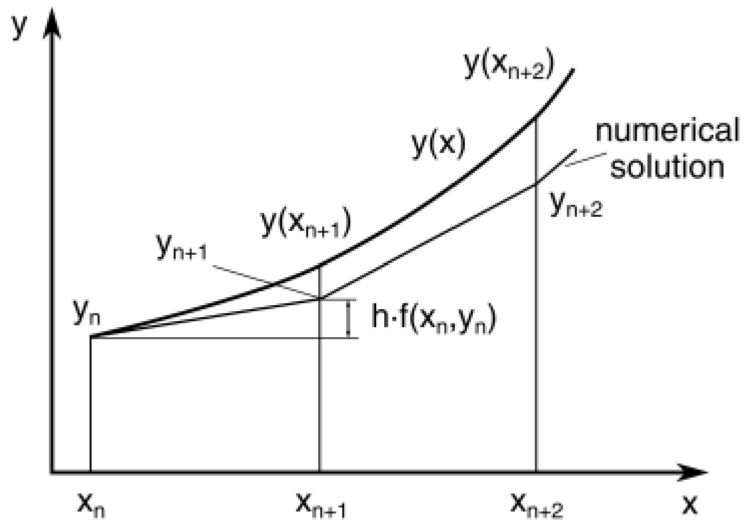
Can't be integrated?

$$x(t_{n+1}) = x(t_n) + \int_{t_n}^{t_{n+1}} f(x(t), t, \theta)dt$$

Euler's Method:

$$t_{n+1} = t_n + h$$

$$x(t_{n+1}) = x(t_n) + h \cdot f(x(t), t, \theta)$$



https://www.freecodecamp.org/news/eulers-method-explained-with-examples/

Euler's Method on our Compound Interest example:

$\frac{dx(t)}{dt} = f(x(t), t, r) = r \cdot x(t);\ x(0) = 20000;\ r = 0.1;\ x(1) = ?$

(Solution: $x(t) = Ke^{r \cdot t};\ x(t = 1) \approx 22103$)

$h = 0.25$

$x(t = 0.25) = x(t = 0) + 0.25 \cdot f(x(t = 0)) = 20000 + 0.25 \cdot 0.1 \cdot 20000 = 20500$

$x(t = 0.5) = x(t = 0.25) + 0.25 \cdot f(x(t = 0.25)) = 20500 + 0.25 \cdot 0.1 \cdot 20500 = 21012.5$

$x(t = 0.75) = x(t = 0.5) + 0.25 \cdot f(x(t = 0.5)) = 21012.5 + 0.25 \cdot 0.1 \cdot 21012.5 = 21537.8125$

$x(t = 1) = x(t = 0.75) + 0.25 \cdot f(x(t = 0.75)) = 21537.81 + 0.25 \cdot 0.1 \cdot 21537.81 = 22,076.25$

Ordinary Differential Equations

Numerical Integration

Initial Value Problem

$$\frac{dx(t)}{dt} = f(x(t), t, \theta);\ x(t_n) \text{ is given};\ x(t_{n+1}) = ?$$

Solution

$$x(t_{n+1}) = x(t_n) + \int_{t_n}^{t_{n+1}} f(x(t), t, \theta)dt$$

Initial Time

Final Time

$$x(t_{n+1}) = \mathbf{ODESolve}(f(x(t), t, \theta), x(t_n), t_n, t_{n+1})$$

Initial Value

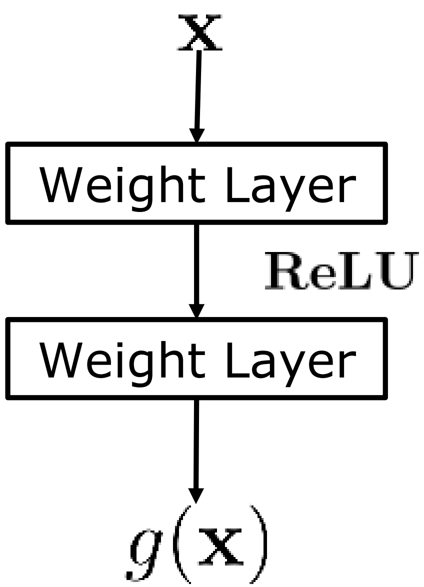
Differential

Solver of any choice

Residual Networks

Residual Networks

Vanilla Neural Networks



Here we have a simple schematic of a block in a Vanilla Neural Network.

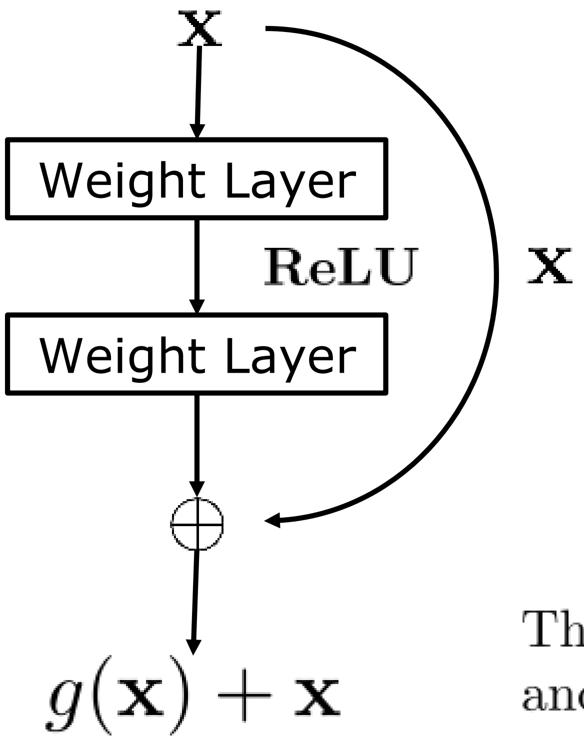
- There's an input vector $\mathbf{x}$.
- It gets passed into a neuron layer where its multiplied to a weight matrix.
- It goes through a **ReLU** Activation layer.
- Perhaps it goes through another layer to give the final output $g(\mathbf{x})$.

This can be reformulated as block t, in larger Neural Network, with weights θ_t and input $\mathbf{x}_t$ has output $\mathbf{x}_{t+1}$, where:

$$\boxed{\mathbf{x}_{t+1} = g(\mathbf{x}_t, \theta_t)}$$

Residual Networks

ResNet[1]



Here we have a similar schematic of a block but in a ResNet.

- There's an input vector $\mathbf{x}$.
- It gets passed into a neuron layer where its multiplied to a weight matrix.
- It goes through a **ReLU** Activation layer.
- Perhaps it goes through another layer.
- Finally the input $\mathbf{x}$ is added to give the final output $g(\mathbf{x}) + \mathbf{x}$.

This can be reformulated as block t, in larger Neural Network, with weights θ_t and input $\mathbf{x}_t$ has output $\mathbf{x}_{t+1}$, where:

$$\boxed{\mathbf{x}_{t+1} = \mathbf{x}_t + g(\mathbf{x}_t, \theta_t)}$$

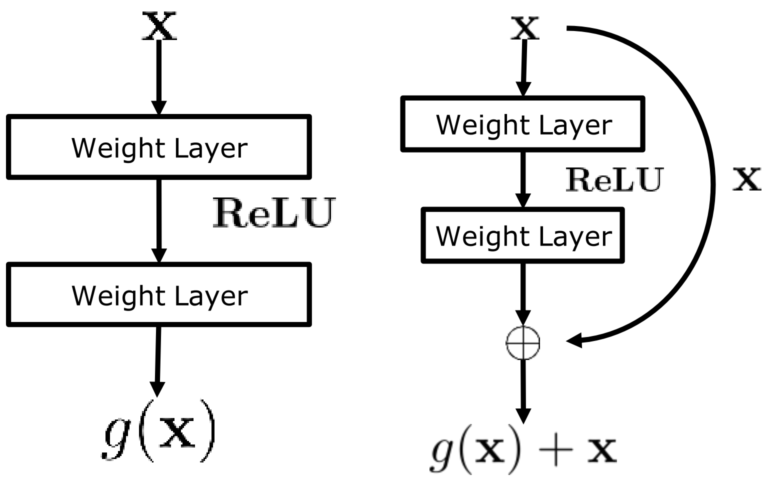
[2] Deep Residual Learning for Image Recognition

Residual Networks

ResNet[1] vs Vanilla Neural Network

Why do Residual Blocks help networks achieve higher accuracies and grow deeper?

- Skip connections help information flow through the network.
- This helps stabilize the training as only the skip connections are sending information in the beginning.
- ResNet blocks allow for stacking, forming very deep networks.
- This is because of the nature of backpropogation of the $\frac{dL}{d\theta}$.
- The chain rule calculation in the intermediate layers has a higher probability for explosive or vanishing gradients the deep the network is.



[2] Deep Residual Learning for Image Recognition

Residual Networks

ResNet[1] and its similarity with Euler Method

Residual Networks

$$\mathbf{x}_{t+1} = \mathbf{x}_t + g(\mathbf{x}_t, \theta_t)$$

Forward pass in a ResNet looks like this:

$$\begin{aligned}
\mathbf{x}_1 &= \mathbf{x}_0 + g(\mathbf{x}_0, \theta_0) \\
\mathbf{x}_2 &= \mathbf{x}_1 + g(\mathbf{x}_1, \theta_1) \\
\mathbf{x}_3 &= \mathbf{x}_2 + g(\mathbf{x}_2, \theta_2) \\
&\vdots \\
\mathbf{x}_t &= \mathbf{x}_{t-1} + g(\mathbf{x}_{t-1}, \theta_{t-1})
\end{aligned}$$

$$\mathbf{y}_{pred} = \mathbf{ResNet}(\mathbf{x}_0)$$

Euler Method for ODEs

$$\mathbf{x}_{t+1} = \mathbf{x}_t + h \cdot f(\mathbf{x}_t, t, \theta)$$

Euler calculation between t=0 and t=t, looks like this:

$$\begin{aligned}
\mathbf{x}_1 &= \mathbf{x}_0 + h \cdot f(\mathbf{x}_0, 0, \theta) \\
\mathbf{x}_2 &= \mathbf{x}_1 + h \cdot f(\mathbf{x}_1, 1, \theta) \\
\mathbf{x}_3 &= \mathbf{x}_2 + h \cdot f(\mathbf{x}_2, 2, \theta) \\
&\vdots \\
\mathbf{x}_t &= \mathbf{x}_{t-1} + h \cdot f(\mathbf{x}_{t-1}, t-1, \theta)
\end{aligned}$$

$$\mathbf{x}_t = \mathbf{ODESolve}(f(\mathbf{x}(t), t, \theta), \mathbf{x}_0, 0, t)$$

Residual Networks

$$\mathbf{x}_{t+1} = \mathbf{x}_t + g(\mathbf{x}_t, \theta_t)$$

Forward pass in a ResNet looks like this:

$$\mathbf{y}_{pred} = \mathbf{ResNet}(\mathbf{x}_0)$$

Euler Method for ODEs

$$\mathbf{x}_{t+1} = \mathbf{x}_t + h \cdot f(\mathbf{x}_t, t, \theta)$$

Euler calculation between t=0 and t=t, looks like this:

$$\mathbf{x}_t = \mathbf{ODESolve}(f(\mathbf{x}(t), t, \theta), \mathbf{x}_0, 0, t)$$

f from **ODESolve** is a **Neural Network!**

Earlier a neural network was pre-defined/hand-designed according to the domain, here we would estimate a function ***f*** that suits our objective.

- The depth of the network ***t*** is equivalent to time ***t*** in the ODE formulation.
- Hence, a forward pass through the ResNet is equivalent to going though the iterations and finding the value of ***f*** at ***t*** with the Euler Method and a constant step size ***h***.
- Since there are better higher order methods than Euler for ODESolve, we can replace ResNet with these methods.

Introducing ODENet

Replacing NNet.*forward()* with ODESolve

NNet

```
def f(x, t, θ):
    return nnet(x, θ_t)

def ResNet(x, θ):
    for t in range(1, T):
        x = x + g(x, t, θ)
    return x
```

- As demonstrated, we iteratively pass through the depths of the Neural Network.
- Evaluating the function ***g*** means inputting the value at the specific depth of the Neural Network.
- Parameters ϴ are specific to layers/depth.
- Function evaluations are carried out just once per layer failing to capture the complexity.

ODENet

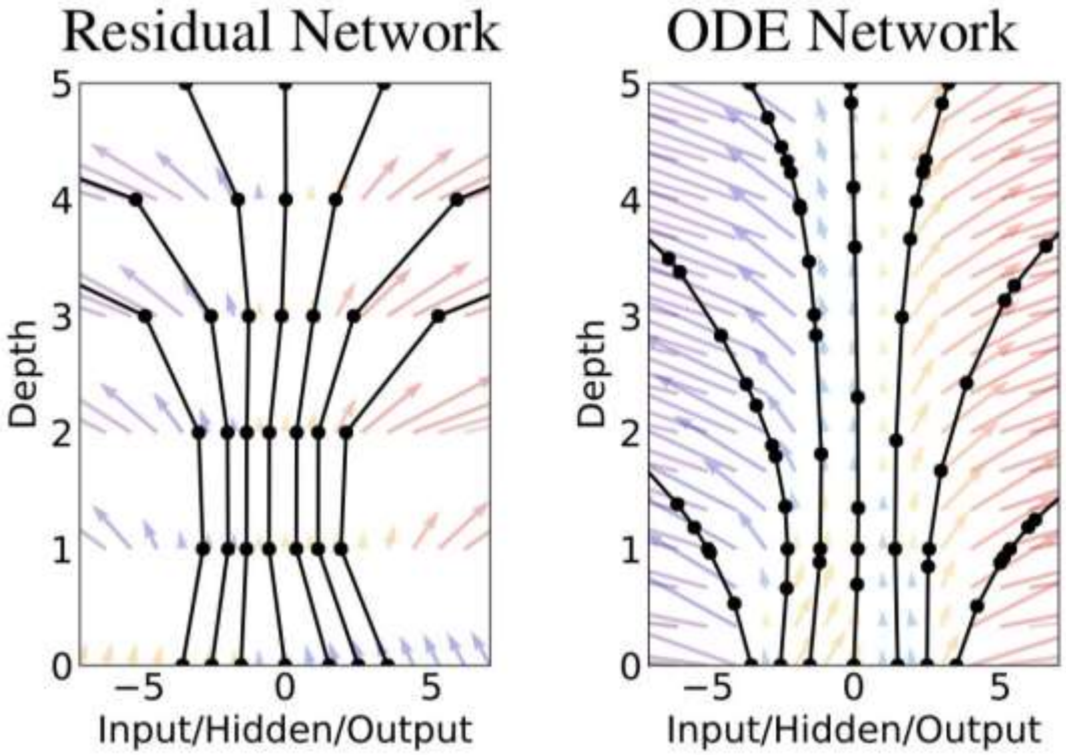
```
def f(x, t, θ):
    return nnet([x,t], θ)

def ODENet(x, θ):
    for t in range(1, T):
        x = x + f(x, t, θ)
    return x
```

- As demonstrated, we iteratively pass through the network as if it is an ODESolve
- Evaluating the fuunction ***f*** mean inputing the value into the ODESolve an solving for that depth.
- In each function call, there‘s intermediate steps based on which ODESolve is chosen.
- Parameters ϴ are shared across layers/depth.
- The number of total executions within a layer depicts how complex the function is, this is based on the task of the network.

ResNet vs ODENet

Function evaluations



- Consider the underlying function ***f*** (or ***g***) to be modeled as a continuous function.
- ResNet like architectures only evaluate this function at specific depths/times.
- The image on the left depicts different inputs in the same function space as different black lines.
- The black dots represent function calls/evaluations, which are done only at specific points along the depth.
- The Proposed ODENet replaces ResNet with ODESolve.
- Based on the algorithm, function is evaluated at different points irrespective of “depth”, as by definition it has intermediate depths.
- This allows ODENet to capture the complexity better.

[2] Neural Ordinary Differential Equations

Backpropogation

Backpropogation

Loss Calculation

Neural Networks

- Loss calculation

$$\mathbf{L} = Criterion(\mathbf{y}_{pred} - \mathbf{y}_{true})$$

- Backpropogation:

$$\mathbf{L}(\mathbf{y}_{pred}) \rightarrow \frac{d\mathbf{L}}{d\theta}$$

Update θ to reduce $\mathbf{L}$.

ODENet

- Loss calculation

$$\mathbf{L} = Criterion(\mathbf{x}_{t_{pred}} - \mathbf{x}_{t_{true}})$$

- Backpropogation:

$$\mathbf{L}(\mathbf{x}_{t_{pred}}) \rightarrow \frac{d\mathbf{L}}{d\theta}$$

Update θ to reduce $\mathbf{L}$.

- But how?
- Backpropogate through the ODE Solver, but that has high memory cost as you would have to save all steps in the middle.

Adjoint Method for Backpropogation

Adjoint State

- Let us first consider calculating how our loss changes with respect of ***x(t)*** at ***t***:

$$a(t) = \frac{dL}{d\mathbf{x}(t)}$$

- We want to know ***a(t)*** at every time/depth ***t***.
- At the output, this is rather easy as we have the loss and the $\boldsymbol{x_t}$:

$$a(t_{pred}) = \frac{dL}{d\mathbf{x}_{t_{pred}}}$$

- Since we want to know ***a(t)*** at every ***t*** we need to find:

$$\frac{da(t)}{dt}$$

- But lets go back a step, what we really need is:

$$\frac{dL}{d\mathbf{x}_t} = \frac{dL}{d\mathbf{x}_{t+\epsilon}} \frac{d\mathbf{x}_{t+\epsilon}}{d\mathbf{x}_t} \implies \frac{dL}{d\mathbf{x}(t)} = \frac{dL}{d\mathbf{x}(t+\epsilon)} \frac{d\mathbf{x}(t+\epsilon)}{d\mathbf{x}(t)}$$

- But since we know ***x(t)*** follows our original ODE:

$$\mathbf{x}(t+\epsilon) = \mathbf{x}(t) + \int_t^{t+\epsilon} f(\mathbf{x}(t), t, \theta)dt = T_\epsilon(\mathbf{x}(t), t)$$

- Plugging it back in, we get:

$$a(t) = \frac{dL}{d\mathbf{x}(t)} = \frac{dL}{d\mathbf{x}(t+\epsilon)} \frac{d\mathbf{x}(t+\epsilon)}{d\mathbf{x}(t)}$$

$$\implies a(t) = a(t+\epsilon) \frac{dT_\epsilon(\mathbf{x}(t), t)}{d\mathbf{x}(t)}$$

Adjoint Method for Backpropogation

Adjoint State

- So we have:

$$a(t) = a(t+\epsilon)\frac{dT_\epsilon(\mathbf{x}(t),t)}{d\mathbf{x}(t)}$$

- Since we want to know ***a(t)*** at every ***t*** we need to find:

$$\frac{da(t)}{dt} = -a(t)\frac{df(\mathbf{x}(t),t,\theta)}{d\mathbf{x}(t)}$$

(Proof in Appendix B.1 of the original paper)

- This is basically another ODE, more complex than the older one but an ODE nonetheless.
- Now it can be solved for any t_k by using t_{output} as initial value:

$$a(t_k) = a(t_{pred}) + \int_{t_{pred}}^{t_k} \frac{da(t)}{dt}dt$$

- Or to get how our loss relates to the initial state $x(t_0)$:

$$a(t_0) = a(t_{pred}) + \int_{t_{pred}}^{t_0} -a(t)\frac{df(\mathbf{x}(t),t,\theta)}{d\mathbf{x}(t)}dt$$

$$\implies a(t_0) = \mathbf{ODESolve}\left(-a(t)\frac{df(\mathbf{x}(t),t,\theta)}{d\mathbf{x}(t)}, a(t_{pred}), t_{pred}, t_0\right)$$

$$\implies \frac{dL}{d\mathbf{x}_{t_0}} = \mathbf{ODESolve}\left(-\frac{dL}{d\mathbf{x}(t)}\frac{df(\mathbf{x}(t),t,\theta)}{d\mathbf{x}(t)}, \frac{dL}{d\mathbf{x}_{t_{pred}}}, t_{pred}, t_0\right)$$

$$\because a(t_0) = \frac{dL}{d\mathbf{x}_{t_0}}; a(t_{pred}) = \frac{dL}{d\mathbf{x}_{t_{pred}}}$$

- But since we need the intermediate depth between t_{pred} and t_0, we also need to solve in tandem:

$$x(t_0) = \mathbf{ODESolve}\left(f(\mathbf{x}(t),t,\theta), x_{t_{pred}}, t_{pred}, t_0\right)$$

- The original ODESolve but in reverse!

Adjoint Method for Backpropogation

Simultaneuous ODESolve for Backpropogation

- So we have:

$$x(t_0) = \textbf{ODESolve}\left(f(\mathbf{x}(t), t, \theta), x_{t_{pred}}, t_{pred}, t_0\right)$$

$$\frac{dL}{d\mathbf{x}_{t_0}} = \textbf{ODESolve}\left(-\frac{dL}{d\mathbf{x}(t)}\frac{df(\mathbf{x}(t), t, \theta)}{d\mathbf{x}(t)}, \frac{dL}{d\mathbf{x}_{t_{pred}}}, t_{pred}, t_0\right)$$

- We have from the paper (Appendix B.2):

$$\frac{dL}{d\theta} = \int_{t_{pred}}^{t_0} -a(t)\frac{df(\mathbf{x}(t), t, \theta)}{d\theta}dt$$

$$\implies \frac{dL}{d\theta} = \textbf{ODESolve}\left(-a(t)\frac{df(\mathbf{x}(t), t, \theta)}{d\theta}, \mathbf{0}_{|\theta|}, t_{pred}, t_0\right)$$

$$\implies \frac{dL}{d\theta} = \textbf{ODESolve}\left(-\frac{dL}{d\mathbf{x}(t)}\frac{df(\mathbf{x}(t), t, \theta)}{d\theta}, \mathbf{0}_{|\theta|}, t_{pred}, t_0\right)$$

$$\begin{bmatrix} x(t_0) \\ \frac{dL}{d\mathbf{x}_{t_0}} \\ \frac{dL}{d\theta} \end{bmatrix} = \textbf{ODESolve}\left(\begin{bmatrix} f(\mathbf{x}(t), t, \theta) \\ -\frac{dL}{d\mathbf{x}(t)}\frac{df(\mathbf{x}(t),t,\theta)}{d\mathbf{x}(t)} \\ -\frac{dL}{d\mathbf{x}(t)}\frac{df(\mathbf{x}(t),t,\theta)}{d\theta} \end{bmatrix}, \begin{bmatrix} x(t_{pred}) \\ \frac{dL}{d\mathbf{x}_{t_{pred}}} \\ \mathbf{0}_{|\theta|} \end{bmatrix}, t_{pred}, t_0\right)$$

Application of ODENet on Time Series

Combination with RNN

To solve for future/past and intermediate time step

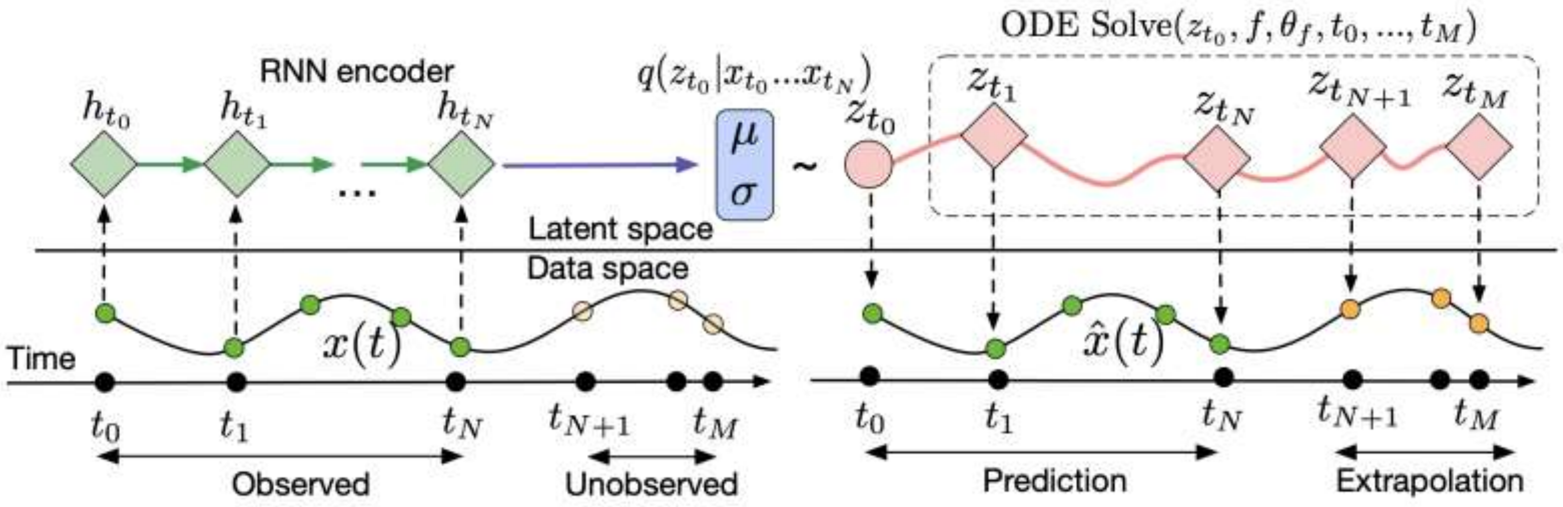


Figure 6: Computation graph of the latent ODE model.

- So lets say we have observations depicting stock market or other time series data at time steps t_0, t_1, … t_N.
- In a regular RNN, this information can be encoded into the hidden states of the RNN and get to a Latent Space.
- We encode this hidden state latent space into a continuous distribution (like a Gaussian) just like in a VAE.
- We then sample from that gaussian a z_{t0} that will serve as the initial value of your ODESolve.

[2] Neural Ordinary Differential Equations

Combination with RNN

To solve for future/past and intermediate time step

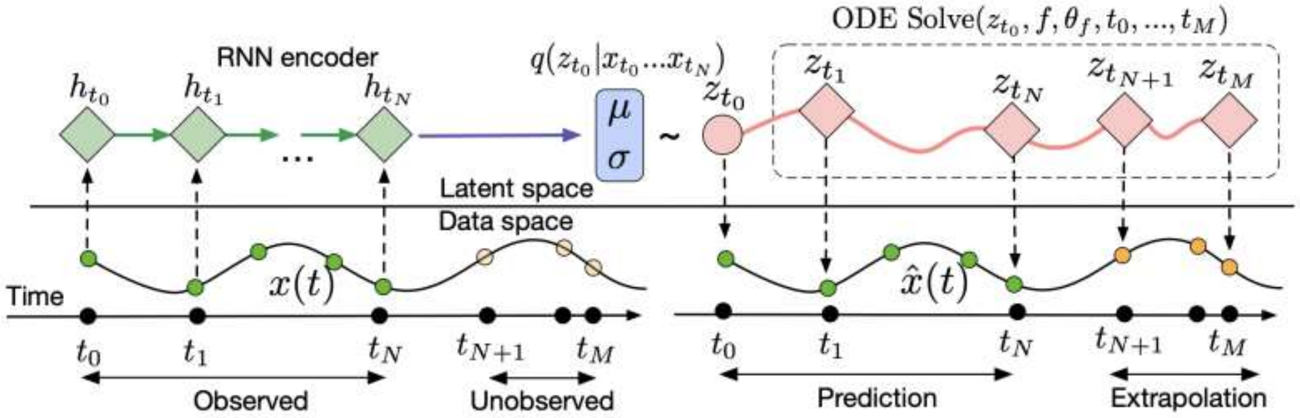


Figure 6: Computation graph of the latent ODE model.

- Now that we z_{t0} as the initial value, we perform an ODESolve to get the latent space representations of all the other time steps z_{t1} till z_{tN}.
- We then decode them into the data space to get our prediction for t1 to tN.
- To then calculate the reconstruction loss from observed and predicted values.
- The advantage of this once trained, is that we can extrapolate the same ODESolve+Decode to a time step in the future.
- This method has no limitations when it comes to irregularly sampled data or extrapolation for different Δt.

[2] Neural Ordinary Differential Equations

Combination with RNN

To solve for future/past and intermediate time step

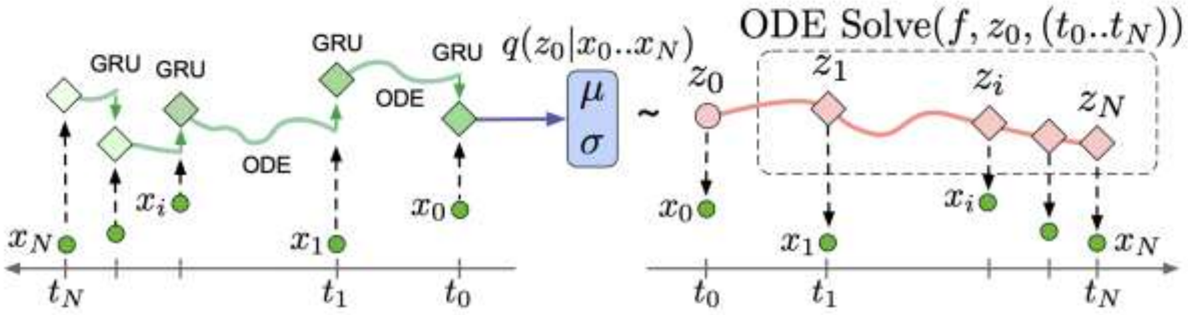


Figure 2: The Latent ODE model with an ODE-RNN encoder. To make predictions in this model, the ODE-RNN encoder is run backwards in time to produce an approximate posterior over the initial state: $q(z_0|\{x_i, t_i\}_{i=0}^N)$. Given a sample of z_0, we can find the latent state at any point of interest by solving an ODE initial-value problem. Figure adapted from Chen et al. [2018].

- A future paper from the same group suggested a LatentODE model to specifically combat irregularly sampled time series.
- They propose getting latent representations of the time-series data through a GRU where its regularly sampled.
- But wherever irregularly sampled they push it through an ODE as this helps fill up the missing values.
- The remaining structure remains the same.

[3] Latent ODEs for Irregularly-Sampled Time Series

References

To solve for future/past and intermediate time step

[1] A Survey on Principles, Models and Methods for Learning from Irregularly Sampled Time Series.

[2] Neural Ordinary Differential Equations.

[3] Latent ODEs for Irregularly-Sampled Time Series.

[4] Dr. Vikram Voleti’s talk and slides.

[5] Machine Leaning @ Berkeley blog post by Aidan Abudlali.

[6] Lecture by Dr. Andriy Drozdyuk.

Tutorials:

1. Jupyter notebook from UCL Artificial Intelligence Society
2. University of Amsterdam Deep Learning Lectures by Phillip Lippe
3. Blog post from Mikhail Surtsukov.

[3] Latent ODEs for Irregularly-Sampled Time Series

Thank you
for your attention

Appendix B.2 Explanation

Appendix B.2

- From appendix B.1, we have the following:

$$\frac{da(t)}{dt} = -a(t)\frac{df(\mathbf{x}(t),t,\theta)}{d\mathbf{x}(t)}$$

- This rule applies to any ODE and its corresponding adjoint defined in the same way:

$$a(t) = \frac{dL}{d\mathbf{x}(t)}$$

- We have from our original formaultion:

$$\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t),t,\theta)$$

- Similarly: $a_\theta(t) = \frac{dL}{d\theta}; a_t(t) = \frac{dL}{dt(t)}; \frac{d\theta}{dt} = 0; \frac{dt}{dt} = 1$

- We can combine these to form an augmented state

$$\frac{d}{d\theta}\begin{bmatrix}\mathbf{x}(t)\\ \theta\\ t\end{bmatrix} = f_{aug}([\mathbf{x}(t),\theta,t]) := \begin{bmatrix}f([\mathbf{x}(t),\theta,t])\\ 0\\ 1\end{bmatrix}$$

- Similarly an augmented adjoint can also be defined:

$$a_{aug}(t) = \begin{bmatrix}\frac{dL}{d\mathbf{x}(t)}\\ \frac{dL}{d\theta(t)}\\ \frac{dL}{dt(t)}\end{bmatrix} = \begin{bmatrix}a(t)\\ a_\theta(t)\\ a_t(t)\end{bmatrix}$$

Appendix B.2

- The Jacobian of f_{aug} , i.e., f_{aug} differentiated with all its variables is:

$$\frac{df_{aug}}{d([\mathbf{x}(t), \theta, t])} = \begin{bmatrix} \frac{df}{d\mathbf{x}(t)} & \frac{df}{d\theta} & \frac{df}{dt} \\ \frac{d\theta}{d\mathbf{x}(t)} & \frac{d\theta}{d\theta} & \frac{d\theta}{dt} \\ \frac{d1}{d\mathbf{x}(t)} & \frac{d1}{d\theta} & \frac{d1}{dt} \end{bmatrix} = \begin{bmatrix} \frac{df}{d\mathbf{x}(t)} & \frac{df}{d\theta} & \frac{df}{dt} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

- Remember, that adjoint can be defined as:

$$\frac{da(t)}{dt} = -a(t)\frac{df(\mathbf{x}(t), t, \theta)}{d\mathbf{x}(t)}$$

- Similarly, our augmented adjoint is:

$$\frac{da_{aug}(t)}{dt} = -a_{aug}(t)\frac{df_{aug}}{d([\mathbf{x}(t), \theta, t])}$$

- Substituting our Jacobian into the equation:

$$\frac{da_{aug}(t)}{dt} = -\begin{bmatrix} a(t)\frac{df}{d\mathbf{x}(t)} & a(t)\frac{df}{d\theta} & a(t)\frac{df}{dt} \end{bmatrix}$$

- This implies that:

$$\frac{da_{\theta}(t)}{dt} = -a(t)\frac{df}{d\theta}$$

- Similar to:

$$\frac{da(t)}{dt} = -a(t)\frac{df(\mathbf{x}(t), t, \theta)}{d\mathbf{x}(t)}$$

Appendix B.2

- So we have:

$$\frac{da_\theta(t)}{dt} = -a(t)\frac{df}{d\theta}$$

- Similar to:

$$\frac{da(t)}{dt} = -a(t)\frac{df(\mathbf{x}(t), t, \theta)}{d\mathbf{x}(t)}$$

- So we can solve it similar to an ODE, integrating from end to beginning:

$$a_\theta(t_k) = a_\theta(t_{pred}) + \int_{t_{pred}}^{t_k} \frac{da_\theta(t)}{dt} dt$$

- Or to get how our loss relates to the parameters $\boldsymbol{\theta}$:

$$a_\theta(t_0) = a_\theta(t_{pred}) + \int_{t_{pred}}^{t_0} -a_\theta(t)\frac{f(\mathbf{x}(t), t, \theta)}{d\theta} dt$$

$$\implies a_\theta(t_0) = \mathbf{ODESolve}\left(-a_\theta(t)\frac{f(\mathbf{x}(t), t, \theta)}{d\theta}, a_\theta(t_{pred}), t_{pred}, t_0\right)$$

$$\implies \frac{dL}{d\theta(t_0)} = \mathbf{ODESolve}\left(-\frac{dL}{d\theta}\frac{f(\mathbf{x}(t), t, \theta)}{d\theta}, \frac{dL}{d\theta(t_{pred})}, t_{pred}, t_0\right)$$

$$\because a_\theta(t_0) = \frac{dL}{d\theta(t_0)}; a_\theta(t_{pred}) = \frac{dL}{d\theta(t_{pred})}$$

Thank you for your attention