NATIONAL RESEARCH UNIVERSITY
HIGHER SCHOOL OF ECONOMICS

Faculty of Computer Science
Bachelor's Programme "Data Science and Business Analytics"

Research Project Report on the Topic:

Deep-learning scenarios: neuro differential equations and their parameter

Fulfilled by:
Student of the Group БПАД213
Dineev Ilshat Rishatovich

(signature) 30.05.2024 *(date)*

Assessed by the Project Supervisor:

Gromov Vasilii Aleksandrovich
Deputy Head, Professor
Faculty of Computer Science, School of Data Analysis and Artificial Intelligence

(signature) *(date)*

Moscow 2024

Contents

Annotation

The selection of hyperparameters, which define the structure and training of neural networks, remains a complex task, often regarded more as an art than a science. Neural differential equations represent an innovative direction in deep learning, where the network's structure is modeled as a continuous segment, and training is conducted through a system of differential equations. This approach transforms the choice of hyperparameters into a problem of bifurcation theory in differential equations.

Аннотация

Выбор гиперпараметров, определяющих структуру и обучение нейронных сетей, остается сложной задачей, часто считаемой больше искусством, чем наукой. Нейродифференциальные уравнения представляют собой инновационное направление в глубоком обучении, где структура сети моделируется как непрерывный отрезок, а обучение - через систему дифференциальных уравнений. Этот подход преобразует выбор гиперпараметров в проблему теории бифуркаций дифференциальных уравнений.

Key words

Deep learning, bifurcations, neural differential networks, variational calculation, ordinary differential equations

Introduction

Description of subject field

The subject field of this project is connected to the intersection of deep learning, the subject of artificial intelligence, with mathematics, especially neural differential networks (NDN), non-linear dynamic systems, functional optimization, numerical methods and bifurcation analysis. This combination of such complex and difficult fields of science is needed to solve fundamental challenges in the neural networks - the optimization and selection of hyperparameters, which play the most significant role in network performance - they define its architecture, learning and optimization algorithms.

Neural differential networks represent revolution in the paradigm of deep learning. Instead of using a discrete layer network, we represent it as a continuous entity. Continuous models allow the application of differential equations to describe the dynamics of learning. This allows us to apply bifurcation analysis to the network to obtain the best hyperparameters.

Bifurcation theory is a field of mathematics that studies structural changes in the family of mathematical models, allowing us to understand how slight changes in the hyperparameters affect the whole system. In our case it is very important, as for neural differential networks it is possible to provide such analysis and rationally choose hyperparameters using a theoretical framework, what is impossible for simple neural networks.

Relevance

The relevance of this project extends beyond the academic interest, impacting the whole field of neural networks application, including computer vision, natural language processing, generative, predicting and classification modeling. It allows the use of a more systematic approach for hyperparameters choosing, which will save a huge amount of time, money and computing power and provide strong evidence and explanation for the architecture of the model.

Goal

The project's objective is to develop an algorithm for the rational selection of hyperparameters for deep learning networks.

Tasks

- Develop and code numerical methods for the effective solution of integro-differential equations and boundary value problems relevant to neural differential network
- Experimentally check those methods
- To study a new branch of mathematics calculus of variations and solve related problems for training
- Formulate the variational problem setup for optimizing neural network hyperparameters using principles from calculus of variations.
- Solve variational problem setup and obtain system of integro-differential equations
- To study a new branch of mathematics bifurcation and solve related problems for training
- Prepare theoretical background and systems of equations for providing numerical bifurcation analysis

Analytical review of sources

In "Neural Ordinary Differential Equations" by Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud from the University of Toronto and Vector Institute [1], the authors bridge the gap between residual networks and ordinary differential equations (ODEs), showcasing the latter as the continuous limit of the former. They introduce an efficient adjoint-based method for computing the backward pass, positioning ODEs as essential modeling tools within deep learning frameworks. Leveraging ODE solvers for both the forward and backward passes significantly boosts computational efficiency. The paper presents continuous time normalizing flows, highlighting their training benefits, such as unconstrained transformations and more efficient computation of probability flows. It also explores latent ODE models for time series, integrating them with VAE-style inference networks for training. This well-constructed study emphasizes the novel use of adjoint methods and ODE solvers in deep learning, likely to have a considerable impact on the field through its methodological innovations and results.

Although neural differential are an evolving topic, they are already finding applications.[2] The integration of neuro differential frameworks with deep learning, specifically through neural ordinary differential equations (ODEs), represents a cutting-edge approach in learning dynamical

models directly from data, even when faced with challenges such as noise and irregular sampling. This innovative methodology, capable of handling complex noise patterns and integrating specific dynamical constraints, marks a significant advancement in the field, enabling more accurate modeling of physical systems. The incorporation of an ensemble strategy further enhances this approach, promising improved reliability and performance in the extraction and interpretation of dynamical systems from real-world data.

“Improving neural ordinary differential equations via knowledge distillation” [3] highlights a critical limitation in Neural Ordinary Differential Equations (Neural ODEs) for image recognition tasks, attributed to insufficient supervised information from one-hot encoding vectors. It introduces a novel solution through knowledge distillation, using ResNets as teacher models to enhance supervised information and significantly improve classification accuracy and robustness against adversarial attacks. This advancement is crucial for my study as it offers a concrete methodology to overcome existing barriers in applying Neural ODEs to complex image recognition, demonstrating the potential for more accurate and secure models in this domain.

The book "Variational Calculus" by I. M. Gelfand and S. V. Fomin, published in 1961[4], delves into the mathematical discipline focusing on finding the function or functions that minimize or maximize a functional. It covers the fundamentals of the calculus of variations, including the derivation of Euler-Lagrange equations, necessary and sufficient conditions for extrema, and the theory of Hamilton. The text is structured to aid in the understanding of how these principles apply to problems in physics and engineering, emphasizing practical problem-solving and theoretical underpinnings.

In A. Gromov's doctoral dissertation, efficient numerical methods for solving boundary value problems are described, including Newton's method, the generalized Kantorovich iterative method, and their modifications with the application of the parameter continuation method, which is necessary for finding an initial approximation. All the numerical methods presented in this work are well-suited for solving differential equations within neural differential equations.

“Modern problems of nonlinear dynamics” [7] delves into contemporary nonlinear dynamics, highlighting the shift from classical sources such as physics and chemistry to newer ones like neuroscience and risk theory. It presents original contributions to mathematical modeling and time series analysis, with a focus on advanced topics like inertial manifolds, attractor reconstructions, and self-organized criticality. The text serves as a comprehensive resource for those in the field and is also accessible to students and researchers seeking to understand the mathematical underpinnings of nonlinear dynamics.

1. Problem statement

1.1. What is Neural Differential Network

Let's consider a group of neural networks that use a complicated series of transformations at the hidden state: residual networks, normalizing flows and network decoders. Formally we can describe those transformations in the next way:

$$\mathbf{h}_{t+1} = \mathbf{h}_t + f(\mathbf{h}_t, \boldsymbol{\theta}_t) \quad (1.1)$$ $\mathbf{h}_{t+1} = \mathbf{h}_t + f(\mathbf{h}_t, \boldsymbol{\theta}_t) \quad (1.1)$

In the paper [1] it is suggested to increase the number of layers and take smaller steps, and in the limit it will be possible to parametrize this network using ordinary differential equations (ODE), so we get next equation that describes dynamics of the hidden layers of the network:

$$\frac{d\mathbf{x}}{dt} = f(\mathbf{x}(t), t, \theta) \quad (1.2)$$ $\frac{d\mathbf{x}}{dt} = f(\mathbf{x}(t), t, \theta) \quad (1.2)$

In this equation x(0) is considered as input layer and x(T) as output one. This value is computed by some ODESolver. Equation (1.2) is called Neural Differential Equation.

1.2. Problem of Neural Differential Network.

In the Neural Differential Networks, especially in the NF, we have some random variable *x* at the initial layer and information about it at the final layer. Our task is to find a normalizing flow, or series of complex transformations, that will transform *x* from initial to final state. To perform this, we can derive initial and final distribution functions of *x* at time and compare them while training, until they won't become similar.

But when we go to the continuous case, our probability starts to transform continuously in time from t=0 to t=T and our vector-variable generates by dynamic system $\mathbf{x} = F(\mathbf{x}(t), \Theta(t), t)$. In this case we need to use non-linear dynamic system theory [7], and introduce a new probability function *p(x, t)* which will be a measure of our dynamic system in time.

To describe the transformation of our probability function in our network we will use the Perron-Frobenius equation [7] :

$$\frac{\partial p(x,t)}{\partial t} + \operatorname{div}\left(p(x,t)\mathbf{F}(x)\right) = 0 \quad (1.3)$$

1.3. Describing the training process.

For the Neural Differential Equations we have to somehow measure the effectiveness of learning. As our main goal is to transform the initial distribution function *p(x, 0)* to final distribution *p(x, T)* we will compare how they are similar to the empirical distribution function of initial and final moment $p_0(X)$ and $p_T(X)$.

Empirical distribution function is nonparametric estimator of the cumulative distribution function. For a sample $X_1, X_2, \dots, X_n$ EDF is defined as follows:

$$\hat{F}_n(x) = \frac{1}{n}\sum_{i=1}^{n} I(X_i \leq x) \quad (1.4)$$

Then we need to somehow measure the difference between our 2 distributions. To perform this comparison we will use Kullback-Leibler Divergence (D_{KL}) defined by next formula:

$$D_{KL}(P \parallel Q) = \int_{-\infty}^{\infty} P(x) \log\left(\frac{P(x)}{Q(x)}\right) dx \quad (1.5)$$

All in all, we suggest next functional describing losses in training:

$$\lambda_1 D_{KL}(p(X,0) \parallel p_0(X)) + \lambda_2 D_{KL}(p(X,T) \parallel p_T(X)) \quad (1.6)$$

where λ_1 and λ_2 are some parameters, $p_0(X)$ is estimated probability distribution of input layer and $p_T(X)$ is estimated probability distribution of output layer.

1.4. Variational setup of the problem

For this problem we will formulate variational problem setup. That means that we will try to find the optimal function to get the extremum of the functional. Additionally, we have the Perron-Frobenius (1.3) constraint. So here's our variational setup:

$$\min_p L = \lambda_1 D_{KL}(p(X,0) \parallel p_0(X)) + \lambda_2 D_{KL}(p(X,T) \parallel p_T(X)) \quad (1.7)$$

With constraint:

$$\frac{\partial p(x,t)}{\partial t} + \operatorname{div}\left(p(x,t)\mathbf{F}(x)\right) = 0$$

Intuition behind this formulation is next: we have a functional describing the difference between the distribution we want and distribution we get with a neural differential network. We want to minimize this difference, and also we have some constraints. In calculus of variation the common way to deal with this type of problem is to introduce Lagrangian, but in variational setup Lagrangian multipliers will be not variable but unknown function. All in all we get next

Lagrangian:

$$\min_p L = \lambda_1 D_{KL}(p(X,0) \parallel p_0(X)) + \lambda_2 D_{KL}(p(X,T) \parallel p_T(X)) +$$
$$+ \int_{t_0}^{t_1} \int_{-\infty}^{\infty} \mu(x,t) \left(\frac{\partial p(x,t)}{\partial t} + \operatorname{div}\left(p(x,t)\mathbf{F}(x)\right) \right) dx\, dt \quad (1.8)$$

2. Methods

2.1. Variational calculation

One of the most important parts of this project is varying our functional (1.8). Functional is a mathematical object that takes function as argument and return number. Most common example of functionality is integral.

Variation is an instrument that allows us to see how small changes in the function impact the whole function. If we make variation equal to zero, after solving such an equation we will get a function that gives us extremum. Here is the definition that we will use to calculate variation

$$\delta J = \frac{\partial}{\partial \alpha} J\left[y(x) + \alpha \delta y(x)\right]\Big|_{\alpha=0} \qquad (2.1)$$

2.2 Newton method for solving boundary value problem.

To solve the obtained system after varying functionality we need some numerical methods. For this purpose we have chosen Newton method. If we have boundary value problem with system of differential equations and boundaries in the vector form:

$$\begin{cases} y'(x) = f(x, y(x)) \\ g(x_0, x_1, y(x_0), y(x_1)) = 0 \end{cases} \qquad (2.2)$$

Using this method allows us to reduce problem (2.2) to equivalent initial value problem (2.3)

$$\begin{cases} y'(x) = f(x, y(x)) \\ y(\tilde{x}) = A \end{cases} \qquad (2.3)$$

Here we choose some value of *x* from our interval and choose some initial form of solution to our problem. After that we compute our function at the chosen point and get the vector of the initial approximation A (2.3).

Equation (2.3) is solved by using a method of numerical integration. In my code I use the Runge-Kutta method of 4-th degree.

Now we can compute the vector of residuals (2.4) by comparing our initial approximation value with boundary condition. We want there difference to be zero:

$$R(A) = g(x_0, x_1, y(x_0, A), y(x_1, A)) = 0 \quad (2.4)$$

During iterations we will vary our vector of initial approximation to get closer to the solution of our problem. To perform these actions we will use next iterative formula (2.5) for finding needed vector of initial approximation:

$$A^{(i+1)} = A^{(i)} - \frac{R(A^{(i)}, \gamma)}{R'(A^{(i)}, \gamma)} \quad (2.5)$$

To compute the derivative R′ in this algorithm we will use the Freschet matrix. Here is the formula to calculate ij-th element of this matrix:

$$\mathcal{F}_{i,j} = \frac{R_i(A_0, \ldots, A_j + \Delta, \ldots, A_n) - R_i(A_0, \ldots, A_j, \ldots, A_n)}{\Delta} \quad (2.6)$$

Intuition behind this matrix is next: it contains partial derivatives by variables and functions. Δ is a small increment, so we numerically calculate how small change in a particular argument affects the whole function.

For this numerical method stopping criteria is residuals - they must be less than some delta. If we obtain residuals less than delta then our algorithm converges and we get a solution.

Huge advantage of the Newton method for solving boundary values is that it has quadratic convergence, so it works very fast. For linear problems it converges in 1 iteration, for nonlinear problems it converges in 5-15 iterations. For systems of equations, time for convergence can be significantly bigger.

One more reason why we use Newton method for boundary value problem is because it is possible to provide bifurcation analysis.

2.3. Parameter continuation method.

Huge disadvantage of the Newton method for boundary value problems is that if we have chosen an initial approximation vector that is far from the real solution, then our method won't converge in rational time or at all. To solve this problem we can use parameter continuation method [5].

In this method we parametrize a vector of coefficients of our system. Then we manually try to find the value of coefficients where we easily get the solution of the system.

After that we steadily increase the value of parameters and obtain solutions. Based on this solution we can build interpolation polynom that will give us value for the next initial

approximation using the next formula:

$$U_0^{(0)} = \sum_{s=0}^{n} a_s \lambda_{n+1}^s$$

If our method didn't converge using the initial approximation using this extrapolation formula, then we can decrease the increment and build a new extrapolation polynomial. If after a big amount of decreasing the increment parameter continuation method did not converge, we can change the parameter. Actually we can take as parameter any monotonically changing component.

If the parameter continuation method converges quite fast, we can increase the increment.

2.4. Generalized Iterative Kantarovich method.

As we are using nonlinear dynamics, our functions depend not only on x, but also on time, in other words, we have two-dimensional functions. This causes problems with the Newton method, because it works only with one-dimensional functions.

To solve this problem we will use the Generalized Iterative Kantarovich method [5]. Main idea is to apply a series of approximations to our bi-dimensional function to represent it as a product of two one dimensional functions.

Here's how we represent two dimensional function:

$$u(x, y) = h(x) \cdot g(y)$$

Iterative process of this method is concluded in sequential solving the systems with x or y. At the first iteration we choose some function h(x) and g(y). After that, for example, we solve systems for h(x), then we use the obtained solution to solve the system for g(y). So we iteratively perform Newton method for systems of one-dimensional functions, and use solutions obtained on the previous iterations.

Stopping criteria for this method is next: we choose some small delta and check if the difference of functions between current and previous is smaller than this delta.

2.5. Finding bifurcation points.

To find bifurcation points, first we need to define critical points.

To find the critical point we will need to use our Freschet matrix (2.6) from Newton method for boundary value problem. Let's introduce parameter β that we will use to provide bifurcation analysis. If we get a singular matrix at the last iteration of the Newton method, in other words the

determinant of the Freschet matrix is zero, then point (A, β) is critical point.

Obtained critical points can be either bifurcation or limit point. To define one's type, we need to delete the k-th row from the Freschet matrix and append columns of derivatives of residuals with respect to the parameter β we are researching for bifurcation. Such transformation gives us augmented Fresche matrix:

$$\tilde{F}_{i,n+1} = \frac{R_i(A, \beta + \Delta) - R_i(A, \beta)}{\Delta} \quad (2.7)$$

Using this this matrix we can introduce criteria for classification of the critical points:

1. If for all values of k determinant of the augmented Frechet matrix is 0, then point (A, β) is bifurcation point
2. If there exists a value of k, where the determinant of the augmented Frechet matrix is not zero, but the determinant of the Frechet matrix is zero, then point (A, β) is a limit point.

2.6. Bifurcation points classification

To classify obtained bifurcation points we need to use next criterias:

1. We can classify bifurcation point as the caspoid if the smallest singular value of Frechet matrix in the Newton method is equal zero
2. We can classify bifurcation point as the umbilic if the two smallest singular value of Frechet matrix in the Newton method is equal zero

3. Experiments

3.1. Newton method for boundary value problem for linear case

As I said in the previous chapters, this method converges very fast for the linear case. Let's consider next boundary value problem:

$$y'' = -4y,\ y(0) = 1,\ y(1) = 0$$

For the Newton method for boundary value problems, we can't give equations in such form to the algorithm. Usual way to solve this problem is to represent such an equation as the system, where in the left-hand side we have derivatives of the function and introduce substitution of these derivatives to express them in the other equations of the system.

For this task method converged in 4 iterations.

To check if our method works correctly, we need to analytically solve this boundary value problem. Performing this, we get next solution:

$$y = -2\cos(2x) + 10\sin(2x) \quad (3.1)$$

Now let's compare the numerical solution of the method and the analytical solution (Figure 1). As we can see, solutions match. So our method converges and gives correct answers for linear problems.

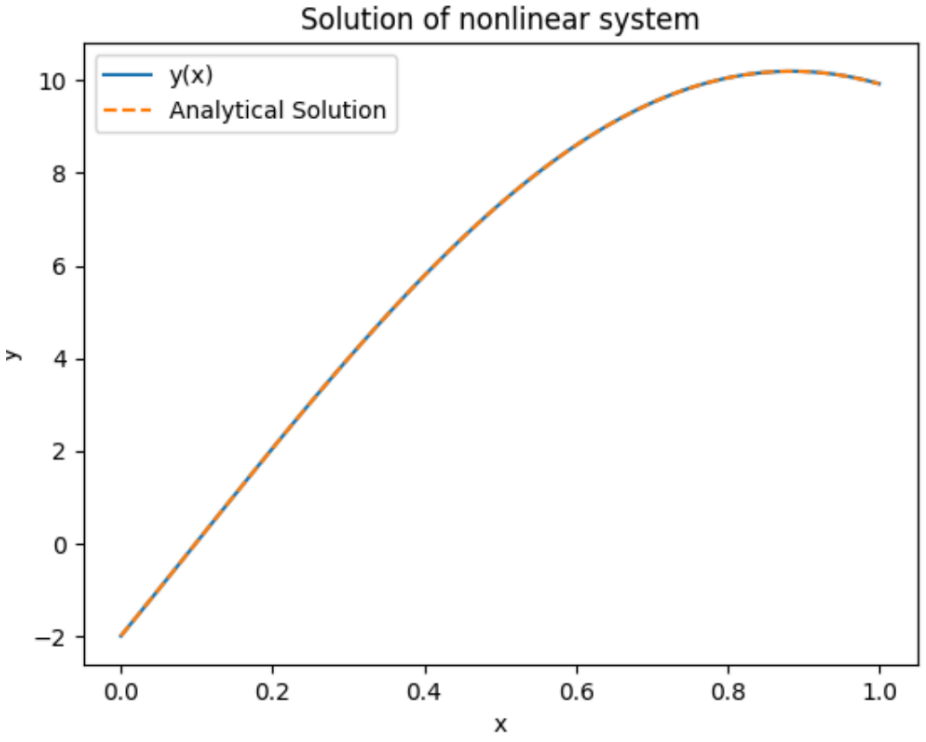

Figure 1 - Comparison of analytical and numerical solution

3.2. Newton method for boundary value problem for non-linear case

Now let's go to the non-linear problem. From the theoretical part, such equations should converge in more time.

Let's consider next problem:

$$\begin{cases} y'' = xy^2 \\ y(0) = 1 \\ y(1) = 2 \end{cases} \quad (3.2)$$

The solution of this problem is shown on Figure 2. Norm of difference between analytical and numerical solution is $-\ 2.35518271551882e\ -\ 11$ what is acceptable result

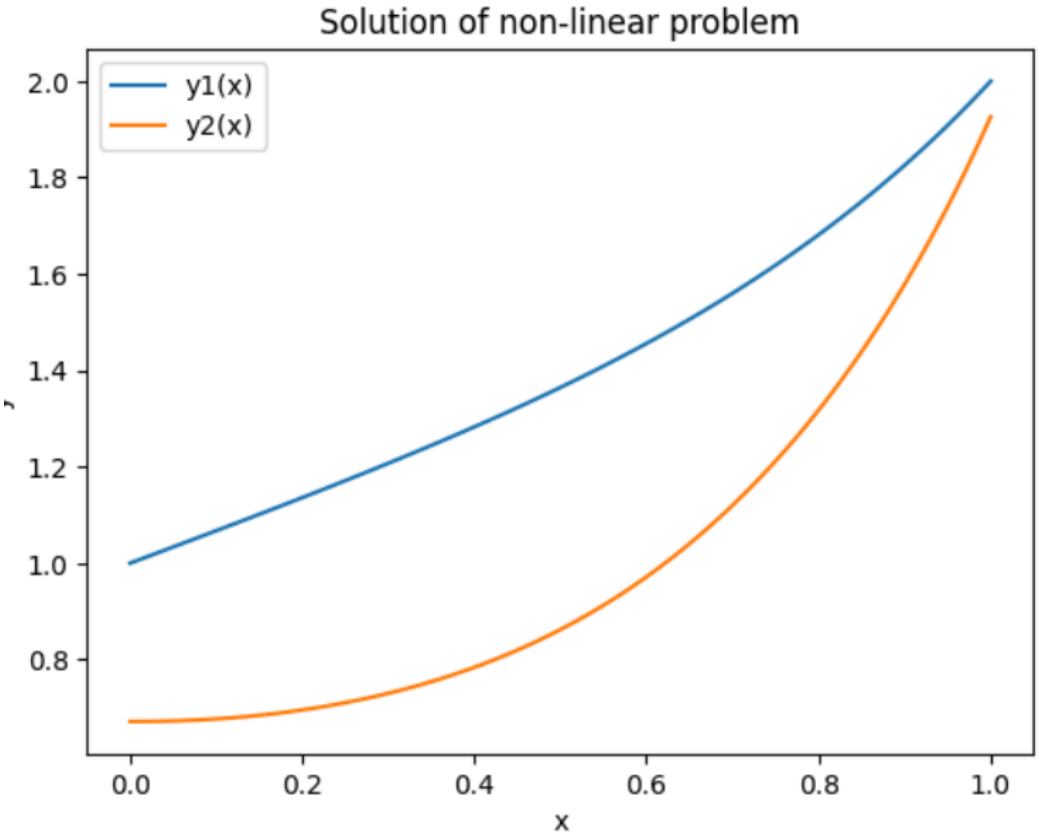

Figure 2 - Comparison of analytical and numerical solution

3.3. Parameter continuation method

In the parameter continuation method we will build a branch of solutions for the equation.

Let's consider next variational problem with parameter

$$y'' + \gamma y = 0 \quad (3.3)$$

With boundary conditions y(0) = 0, y(1) = 1. Here's branch of solution that our method made based on parameter (Figure 3)

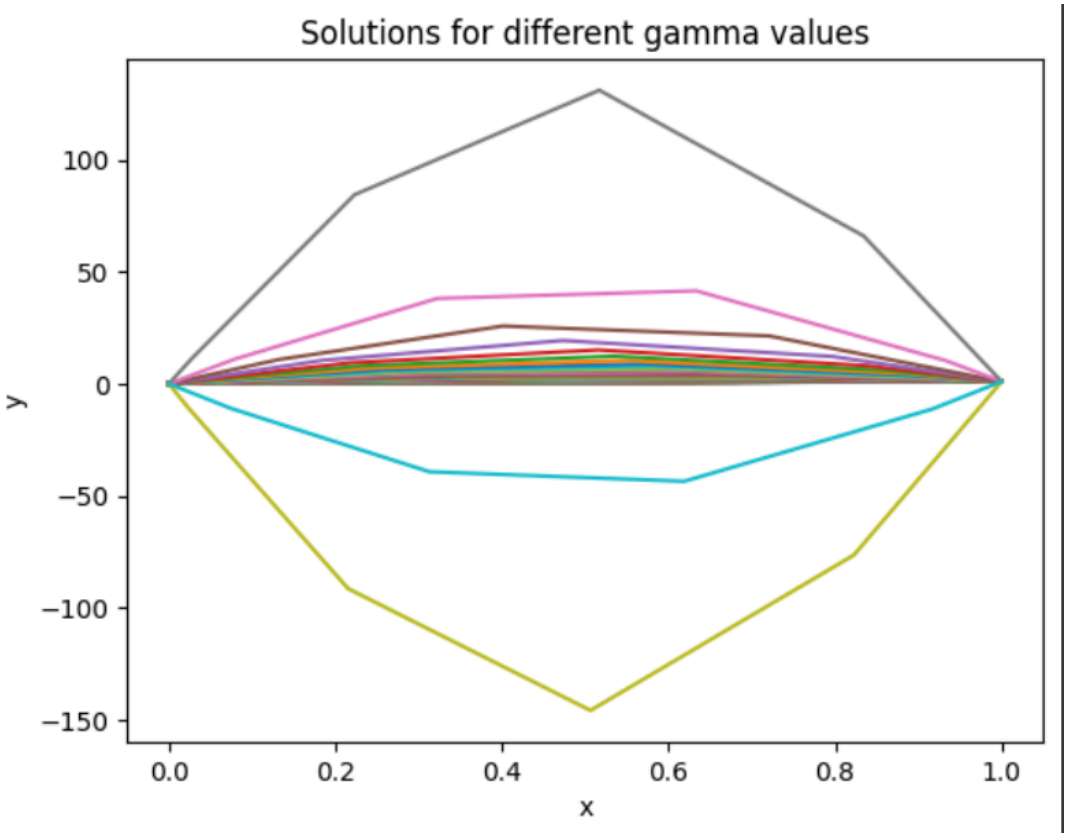

Figure 3 - Branch of solutions

3.4. Generalized Kantarovich Iterative method for Neural Differential Networks.

Let's apply the Generalized Kantarovich Iterative Method for our variational setup of the Neural Differential Network. As we can see, our probability and Lagrange function has 2 variables in the argument, that's why we need to apply an approximation form the GKIM.

Here how we will represent these functions:

$$\begin{cases} p(x,t) = p_x(x) \cdot p_t(t) \\ \mu(x,t) = \mu_x(x) \cdot \mu_t(t) \end{cases} \quad (3.4)$$

In variational setup problem we have derivative of p(x,t) by t, in such case we have next approximation of the function:

$$\frac{\partial p(x,t)}{\partial t} = p_x(x) \cdot \frac{\partial p_t(t)}{\partial t} \quad (3.5)$$

Here is the new variational formulation for our problem using decomposition from Kantarovich Generalized Iterative method:

3.6. How we will calculate variations

Some of the functions we are varying in (1.8) have derivatives, which makes calculations harder, as our variations must be without derivatives. Usual way to overcome such problems is to use the Euler equation [4]. But if we take a closer look at them, we will see that they are derived using some assumptions, which doesn't match with our problem. Talking more precisely, they work only if we remove all non-main parts when we integrate by parts in order to remove differential form variation signs.

That's why we will calculate all variations by definition, which will take a lot of time, in comparison with using formulas for the Euler equation.

As we prepare those equations for Newton method, we will need to get boundary conditions for every function. In order to obtain one, in the process of variation we will derive Natural Boundary Conditions, which will appear while varying by hands.

3.7. Varying the functional

Here is the variations and Natural boundary conditions that we obtain after varying the functional:

Natural boundaries:

$$\begin{cases} \ln\left(\int_{-\infty}^{\infty} p_t(0)\,dx\right) = \ln\left(\int_{-\infty}^{\infty} \frac{\hat{p}_0(x)}{p_x} \cdot e^{-p_x}\,dx\right) \\ \ln\left(\int_{-\infty}^{\infty} p_t(T)\,dx\right) = \ln\left(\int_{-\infty}^{\infty} \frac{p_T(x)}{p_x} \cdot e^{-p_x}\,dx\right) \\ \int_0^T \left(\mu_x \cdot \mu_t \cdot f_x \cdot \delta p_x|_{-\infty}^{\infty}\right) = 0 \\ \int_{-\infty}^{\infty} \mu_x \mu_t \cdot p_x \cdot \delta p_t\Big|_0^T = 0 \quad (3.6) \end{cases}$$

Variations:

$\delta\mu_x$:

$$\int_{-\infty}^{\infty} \delta\mu_x \int_0^T \mu_t \cdot \left(p_t'(t) \cdot p_x(x) + p_t(t) \cdot \frac{\partial}{\partial x}(p_x \cdot f_x)' \right) dt\, dx = 0$$

$\delta\mu_t$:

$$\int_0^T \delta\mu_t \int_{-\infty}^{\infty} \mu_x \cdot \left(p_t'(t) \cdot p_x(x) + p_t(t) \cdot \frac{\partial}{\partial x}(p_x \cdot f_x)' \right) dt\, dx = 0$$

δp_x :

$$\int_{-\infty}^{\infty} \delta p_x \int_0^T \frac{\lambda_1}{T} \cdot p_t(0)(1 + \ln(x))\, dt\, dx +$$
$$\int_{-\infty}^{\infty} \delta p_x \int_0^T \frac{\lambda_2}{T} \cdot p_t(T) \left(1 + \ln \frac{p_x \cdot p_t(T)}{p_T(x)} \right) dt\, dx +$$
$$\int_{-\infty}^{\infty} \delta p_x \int_0^T \mu_x \cdot \mu_t \cdot p_t' - (\mu_x \cdot \mu_t \cdot f_x)'\, dt\, dx = 0$$

δp_t :

$$\int_0^T \delta p_t \int_{-\infty}^{\infty} \frac{d}{dt} \cdot (\mu_x \cdot \mu_t \cdot p_x) + \mu_x \cdot \mu_t \cdot \frac{d}{dx} \cdot (f_x \cdot p_x) dx dt = 0 \quad (3.7)$$

We obtained Natural Boundary conditions when we were removing derivatives from the variation sign by using integration by parts. Another type of boundary conditions was obtained p_t at point 0 and T. Now let's solve this system.

We can provide bifurcation analysis of system (3.8) using Kantarovich Generalized Iterative method.

$$(1): \frac{d}{dx} \cdot \mu_x = \frac{\mu_x \left(\int_0^T \mu_t \cdot p_t' dt - f_x^1 \int_0^T \mu_t \right) + \int_0^T \frac{\lambda_1}{T} \cdot p_t(0) \left(1 + \ln \frac{p_x \cdot p_t(0)}{p_0(x)} \right) dt + \int_0^T \frac{\lambda_2}{T} \cdot p_t(T) \left(1 + \ln \frac{p_x \cdot p_t(T)}{p_T(x)} \right) dt}{f_x \int_0^T \mu_t dt}$$

$$(2)\ \frac{d}{dt} \cdot \mu_t = \frac{-\mu_t \int_{-\infty}^{\infty} \mu_x \cdot \frac{\partial}{\partial x}(f_x \cdot p_x) dx}{\int_{-\infty}^{\infty} \mu_x \cdot p_x dx}$$

$$(3)\ p_t'(t) = -\frac{p_t \int_{-\infty}^{\infty} \frac{\partial}{\partial x}(p_x \cdot f_x) dx}{\int_{-\infty}^{\infty} \mu_x \cdot p_x dx} p_x' f_x + p_x f_x'$$

$$(4)\ p_x' = -\frac{p_x \left(\int_0^T \mu_t \cdot p_t' dt + f_x' \int_0^T p_t dt \right)}{\int \mu_t \cdot p_t' dt} \quad (3.8)$$

Conclusion

In the end of this project, we can highlight next obtained results:

1. New variational setup for the problem of the Neural Differential Networks was proposed
2. Was chosen and coded methods for the solution of the differential boundary values problem
3. We prepared theoretical framework for the providing bifurcation numerical analysis based on the Catastrophe theory
4. Suggested way for rational choice of the hyperparameters of the neural differential networks
5. Varied the Lagrangian Functional for the Neural Differential Networks
6. Solved the system of variations

In the future, obtained numerical methods, theoretical framework and variational systems can be used to provide bifurcation analysis of the Neural Differential Networks. This will allows us to rationally choose the hyperparameters of the network

List of resources

[1] Chen, R. T. Q., Rubanova, Y., Bettencourt, J., & Duvenaud, D. (2018). Neural Ordinary Differential Equations. arXiv preprint arXiv:1806.07366. Retrieved from https://arxiv.org/abs/1806.07366

[2] Goyal, Pawan & Benner, Peter. (2023). Neural ordinary differential equations with irregular and noisy data. Royal Society Open Science. 10. 10.1098/rsos.221475.

[3] Chu, Haoyu & Wei, Shikui & Lu, Qiming & Zhao, Yao. (2023). Improving neural ordinary differential equations via knowledge distillation. IET Computer Vision. n/a-n/a. 10.1049/cvi2.12248.

[4] I. M. Gelfand and S. V. Fomin. Variational Calculus. Moscow, 1961

[5] A. Gromov's doctoral dissertation

[6] Wolfram Research, Inc. (n.d.). Numerical Solution of Boundary Value Problems—Wolfram Language Documentation. Retrieved from https://reference.wolfram.com/language/tutorial/NDSolveBVP.html

[7] Malinetsky G.G., Potapov A.B., Modern problems of nonlinear dynamics. Moscow, 2000

[8] Kazuki Irie, Francesco Faccio, Jürgen Schmidhuber. “Neural Differential Equations for Learning to Program Neural Nets Through Continuous Learning Rules”. arXiv preprint, arXiv:2206.01649, version 2. 2022.

[9] Wikipedia contributors. (n.d.). Kullback–Leibler divergence. In Wikipedia, The Free Encyclopedia. Retrieved February 14, 2024, from https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence

[10] Wikipedia contributors. (n.d.). Perron–Frobenius theorem. In Wikipedia, The Free Encyclopedia. Retrieved February 14, 2024, from https://en.wikipedia.org/wiki/Perron%E2%80%93Frobenius_theorem