Probabilistic Machine Learning - Project Report

A hybrid of (Bayesian) Neural Networks and Polynomials for Physics Informed Machine Learning.

Meysam Alishahi and Sina Rashetnia

Abstract

In physics informed machine learning or scientific machine learning world, there are so many papers published to introduce neural networks that are trained to solve supervised learning tasks while respecting any given law of physics described by nonlinear partial differential equations. The key idea behind this project is to first add a polynomial surrogate to the neural network in order to address more difficult problems and then change the classical neural network to Bayesian neural networks to introduce more robust models for these kinds of problems.

1 Introduction

Scientists and engineers have repeatedly used knowledge of the physical world to simulate nature's solutions to complex challenges. The shortage of data leads us to make decisions under partial information. Physics-Informed Bayesian Neural Networks (PINN) are Bayesian neural networks (BNNs) that encode model equations, like Partial Differential Equations (PDE), as a component of the Bayesian neural network loss. PINNs approximate PDE solutions by training a Bayesian neural network to minimize the loss function, which forces the BNN to respect partial information as well.

2 Our Problem

It is very important to solve a PDE as it is a crucial problem in many scientific areas. However, most of the time we are not able to solve the PDE analytically and we need to find an approximation of the solution. It is a common situation that we have access to a simulation that can produce some points that come from the solution. However, such a simulation is too expensive to run, and accordingly, producing the data is time-consuming. In our setting, we assume that we have access to N points $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ generated from the solution and we want to use Neural Network to approximate the solution. Then we can use this approximation as the solution. Let us explain the problem more precisely.

2.1 Modeling the problem

Let's consider a one-dimensional time-spatial PDE as given in Equation (1).

$$u_t + \beta u_x = v(x,t) \tag{1}$$

with given data poits $((x_1,t_1),y_1),\ldots,((x_N,t_N),y_N)$ for which $u(x_i,t_i)=y_i$.

Our goal is to model the solution by

$$\hat{u} = f_\Theta(x,t) \tag{2}$$

where f_θ is a Nural Network parameterized by Θ.

2.2 Loss function

First, note that we want

$$\hat{u} = f_\Theta(x,t)$$

to approximate the solution of Equation (1). Therefore, we need to somehow impose Equation (1) to the loss function. To this end, we first sample M points (x_i',t_i') and then add the following term to our loss function.

$$\mathcal{L}_{pde}(\Theta) = \sum_{i=1}^{M}\left(\frac{\partial\hat{u}(x_i',t_i')}{\partial t} + \beta\frac{\partial\hat{u}(x_i',t_i')}{\partial x} - v(x_i',y_i')\right)^2. \tag{3}$$

As we said, we are given data poits $((x_1,t_1),y_1),\ldots,((x_N,t_N),y_N)$ for which $u(x_i,t_i)=y_i$. It guarantees the uniqueness of the solution of the PDE. Also, we can add the following term to the loss function to respect to these points.

$$\mathcal{L}_b(\Theta) = \sum_{i=1}^{N}\left(\hat{u}(x_i,t_i)-y_i\right)^2. \tag{4}$$

Finally, summing up the Equations (3,4), we obtain the desired loss:

$$\mathcal{R}(\Theta) = \mathcal{L}_{pde}(\Theta) + \mathcal{L}_b(\Theta). \tag{5}$$

2.3 Experimental Results

We want to test the model on a given PDE to see how well it works. Let us consider the following PDE:

$$u_x + u_t = 3x + t \qquad x,t \in [-1,1]. \tag{6}$$

Also, we are given 200 points

$$\{((x_1,t_1),y_1),\ldots,((x_{200},t_{200}),y_{200})\}$$

such that $(x_,t_i) \in [-1,1]^2$ and $y_i = u(x_i,t_i)$ where $u(x,t)=x^2+tx$. Therefore, we have $\mathcal{L}_b(\Theta) = \sum_{i=1}^{200}\left(\hat{u}(x_i,t_i)-y_i\right)^2$. We also randomly sample 100K points (x_i',t_i') from $[-1,1]^2$ to set up $\mathcal{L}_{pde}(\Theta)$.

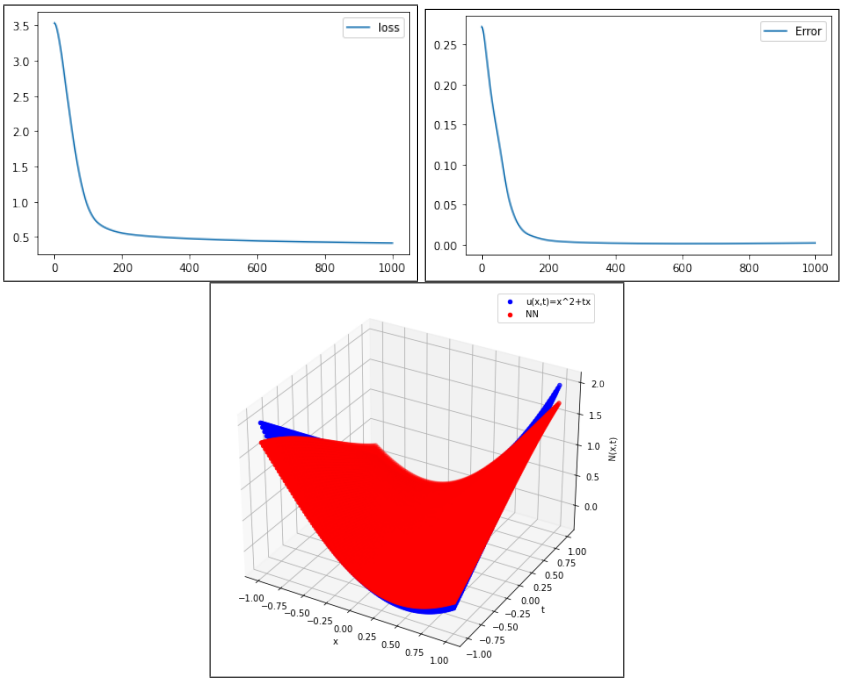

Figure 1: The true solution vs the solution given by Neural network

3 Bayesian Physics Informed Neural Networks

In this section, we want to define a probabilistic version of the above model. In our proposed model, we have

$$u(x,t) \sim \mathcal{N}\left(\mu(x,t), \sigma\right).$$

Now, we are given the following one-dimensional time-spatial PDE to solve

$$\mu_t + \beta \mu_v = v(x,t) \tag{7}$$

with boundary condition $\mu(x,0) = g(x)$. Indeed, we are given N data points $\mathcal{D} = \{u(x_1,0), \ldots, u(x_N,0)\}$ sampled from $\mathcal{N}\left(\mu(x,0), \sigma\right)$ as our boundary points. It is worth noting that if we set $\sigma = 0$, we get back to the deterministic model in the previous section.

3.1 Model

Our goal is to approximate $\mu(x,t)$. To this end, we suppose that $\mu(x,t)$ can be approximated by a model

$$\hat{\mu}(x,t) = f_\Theta(x,t) \tag{8}$$

where $f_\Theta(x,t)$ is a Nural Network parameterized by Θ.

3.2 Prior on Parameters and Likelihood

As a prior on the parameters Θ, we consider a fully factorized Gaussian distribution. Indeed, we assume, for each $w_i \in \Theta$,

$$w_i \sim \mathcal{N}(0,1),$$

i.e.,

$$p(\Theta) = \prod_{w \in \Theta} \mathcal{N}(w|0,1).$$

We also suppose that our likelihood is Gaussian which results in

$$p(D) = \int p(\Theta, \mathbf{M}) p(D|\Theta, \mathbf{M}) d\Theta d\mathbf{M} \tag{9}$$

$$= \int \prod_{w \in \Theta} \mathcal{N}(w|0,1) \prod_{n=1}^{N} \mathcal{N}(y_n|\hat{\mu}_\Theta(x_n, t_n), \sigma) d\Theta \tag{10}$$

3.3 Introducing variational posterior and constructing variational evidence lower bound

We choose a fully factorized Gaussian posterior:

$$\begin{aligned} q(\Theta) &= \prod_{w \in \Theta} q(w) \\ &= \prod_{w_i \in \Theta} \mathcal{N}\left(w_i|\alpha_i, \sqrt{\log(1+\exp(\beta_i))}\right). \end{aligned}$$

Now, using Jenson inequality, we obtain the following variational evidence lower bound

$$\begin{aligned} \log p(D) &\geq \mathcal{L}(\Theta) \\ &= \int q(\Theta) \log \frac{p(\Theta) p(D|\Theta)}{q(\Theta)} d\Theta \\ &= \underbrace{\mathbf{E}_q(\log p(\Theta))}_{\text{analytically computable}} + \mathbf{E}_q(\log p(D|\Theta)) - \underbrace{\mathbf{E}_q(\log q(\Theta))}_{\text{analytically computable}}. \end{aligned}$$

Note that the first and third term of $\mathcal{L}(\Theta)$ can be computed as follows:

$$\begin{aligned} \mathbf{E}_q(\log p(\Theta)) &= C - \frac{1}{2} \sum_{w_i \in \Theta} \mathbf{E}_{q(w_i)}(w_i^2) \\ &= -\frac{1}{2} \sum \left[\text{var}_{q(w_i)}(w_i) + \mathbf{E}_{q(w_i)}(w_i)^2\right] + C \\ &= \underbrace{-\frac{1}{2} \sum \left[\log(1+\exp(\beta_i)) + \alpha_i^2\right]}_{=A(\Theta)} + C \end{aligned}$$

and

$$\begin{aligned}\mathbf{E}_q(\log q(\Theta, \mathbf{M})) &= \sum_{w_i \in \Theta} \mathbf{E}_{q(w_i)}(\log q(w_i)) \\ &= C \underbrace{-\frac{1}{2} \sum_{w_i \in \Theta} \log\left(\log(1 + \exp(\beta_i))\right)}_{B(\Theta)}.\end{aligned}$$

With the use of the reparameterization trick, we can rewrite the second term as

$$\begin{aligned}\mathbf{E}_q(\log p(D|\Theta)) &= \sum_{n=1}^{N} \mathbf{E}_q\big(\log p(y_n|\hat{\mu}_{(\Theta)}(x_n, t_n))\big) \\ &= \underbrace{\sum_{n=1}^{N} \mathbf{E}_{p(\varepsilon)}\big(\log p(y_n|\hat{\mu}_{(\Theta)}(x_n, t_n))\big)}_{C(\Theta)},\end{aligned}$$

where $p(\varepsilon_i) = N(0, 1)$ and

$$w_i = \alpha_i + \varepsilon_i \sqrt{\log(1 + \exp(\beta_i))}.$$

Wrapping everything up, our goal is to maximize $\mathcal{L}(\Theta)$ to fing α's and β's. However, we should respect to condition (3) as well. Therefore, our loss function colud be defined as follows:

$$\mathcal{R}(\Theta) = -A(\Theta) + B(\Theta) - C(\Theta) + \mathcal{L}_{pde}(\Theta)$$

Although $C(\Theta)$ is analytically intractable, similar to the Bayesian neural network by backpropagation, we can take care of $C(\Theta)$ by stochastic optimization.

3.4 Experimental Results

We apply our Bayesian model to the problem defined on Secion 2.3. It does not work as well as the non-Bayesian model. It could be because of tuning the parameters or the nature of the problem (see Figure 2).

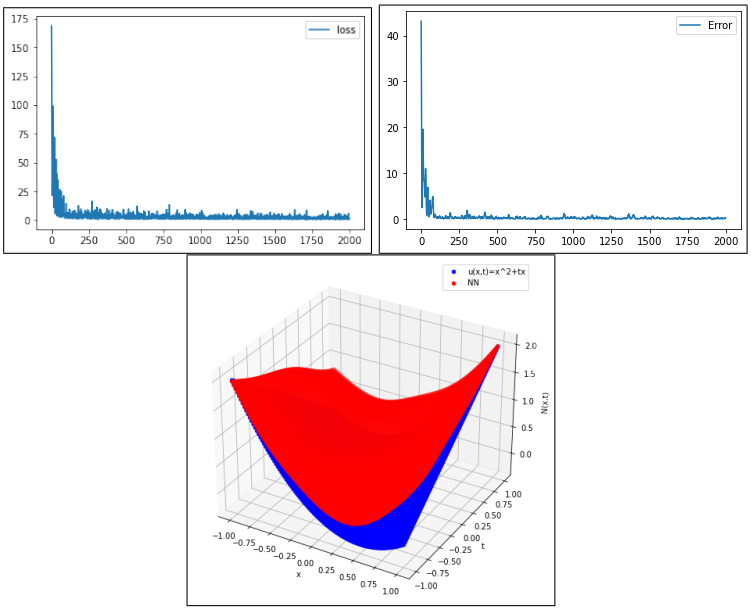

Figure 2: The true solution vs the solution given by Bayesian Neural network

4 Why not the traditional method

Most likely solving a PDE analytically is a very difficult task. Also, in many applied problems, we are involved with a PDE and we need to (at least approximately) solve it. Therefore, if we can use a modern method in machine learning to solve a PDE with good accuracy, then we obtain a very effective and also influential method.

5 Future Works

As we mentioned, one of the goals of this project besides introducing a polynomial surrogate in order to address more difficult problems in physics-informed research areas is to change the classical Neural Network method to Bayesian Neural Network. Bayesian Neural Networks (BNNs) have the potential to combine the scalability, flexibility, and predictive performance of neural networks with the ethics of Bayesian uncertainty modeling. However, the practical effectiveness of BNNs is limited by our ability to specify meaningful prior distributions and by the intractability of posterior inference. Choosing a meaningful prior distribution over network weights is difficult because the weights have a complicated relationship to the function computed by the network. One possible future plan is to use another method of stochastic variational inference.

The codes are uploaded in Github, see the following link:
`https://github.com/SinaRashetnia/PML-CS6190-project`