Studies on PINNs

The PINN Module and Future Studies

A. Salih Taşdelen

METU

April 11, 2023

Outline

The Progress

- Reproduced the laminar flow over a cylinder Rao et al., 2020.
- Tried to understand the implementation and how it could be generalized. Since the source is in Tensorflow 1 (Abadi et al., 2016), the user is the one who almost directly implements the computation graph.
- Read other papers that implements PINNs in TF2:
  - **DeepXDE** by Lu et al., 2021. (Residual Based Adaptive Refinement)
  - **Elvet** by Araz et al., 2021. (Gradient Stack)
  - **dNNSolve** by Guidetti et al., 2021. (Fourier Neural Nets)
  - **IDRLnet** by Peng et al., 2021. (Constraint Importance is proportional to its domain area)
  - **PyDEns** by Koryagin et al., 2019. (Deep Galerkin)
  - **TensorDiffEq** by McClenny et al., 2021. (`tf.gradients` instead of `tf.GradientTape`)

Problem in Mathematical Terms I

The Domain

- Suppose we are given a problem in D number of space time dimensions. Namely, we have time t, and $D-1$ number of spatial dimensions. Then, the domain can be represented as:
  - $\mathrm{D} = [t_0, t_1] \times \Omega \subset \mathbb{R}^D$ where,
  - $\Omega \subseteq \mathbb{R}^{D-1}$, is the spatial domain for which,
    - ★ $\vec{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_{D-1} \end{bmatrix}^T$ is an element of Ω.
  - We can represent the elements of space-time domain as a column vector $\varphi = \begin{bmatrix} t & x_1 & x_2 & \cdots & x_{D-1} \end{bmatrix}^T \in \mathbb{R}^D$.
  - With the above definitions, let us say that $\vec{u}(\varphi)$ is the desired solution of the problem.
  - Also let $\hat{u}(\varphi)$ be the solution approximated by the Neural Network.
  - We generalize the desired solution to be in any dimension, so that it could be the measure of various quantities like; velocity, pressure etc.

Problem in Mathematical Terms II

Governing Equations - PDEs

Let us look at the governing equations as well:

- Partial Differential Equations:
  - Suppose the PDEs have the form $f_i(\varphi, \vec{u}(\varphi)) := 0$, $\varphi \in T_\Omega \subseteq \mathrm{D}$.
  - If we have n_Ω number of PDEs their representation would be:

$$\vec{f}(\varphi, \vec{u}(\varphi)) := \begin{bmatrix} f_1 & f_2 & \cdots & f_{n_\Omega} \end{bmatrix}^T = \vec{0} \tag{1}$$

  - Let us denote the number of collocation points sampled for the PDEs as N_Ω. Which is equivalent to say, $|T_\Omega| = N_\Omega$.

Problem in Mathematical Terms III

Governing Equations - ICs

- Initial Conditions:
  - Suppose the ICs have the form $g_i(\varphi, \vec{u}(\varphi)) := 0$, $\varphi \in T_{0_i} \subseteq \mathrm{D}$.
  - $|T_{0_i}| = N_{0_i}$, number of collocation points samples for the i^{th} IC.
  - For the initial condition we now that $t = t_0$, hence $T_{0_i} = \{t_0\} \times \mathrm{I}_i$.
  - The spatial domain of IC is $\mathrm{I}_i \subseteq \Omega$.
  - The spatial domains of initial conditions cannot coincide thus, $\mathrm{I}_i \cap \mathrm{I}_j = \varnothing$, $i \neq j$.
  - Let us define the set for all the sets for IC domains, $\Gamma_0 = \{T_{0_i}\}_{1 \leq i \leq n_0}$.
  - Since IC domains do not intersect we have the total number of IC collocation points as:

$$N_0 = \sum_{i=1}^{n_0} |T_{0_i}|$$

Problem in Mathematical Terms IV

Governing Equations - BCs

- Boundary Conditions:
  - Suppose the BCs have the form $h_i(\varphi, \vec{u}(\varphi)) := 0,\ \varphi \in T_{0_i} \subseteq \mathrm{D}$.
  - $|T_{\partial\Omega_i}| = N_{\partial\Omega_i}$, number of collocation points samples for the i^{th} BC.
  - For the boundary conditions, $T_{\partial\Omega_i} = [t_0, t_1] \times \mathrm{B}_i$.
  - The spatial domain of BC is $\mathrm{B}_i \subseteq \partial\Omega$.
  - The spatial domains of boundary conditions cannot coincide thus, $\mathrm{B}_i \cap \mathrm{B}_j = \varnothing,\ i \neq j$.
  - Let us define the set for all the sets for BC domains, $\Gamma_{\partial\Omega} = \{T_{\partial\Omega_i}\}_{1 \leq i \leq n_{\partial\Omega}}$.
  - Since BC domains do not intersect we have the total number of BC collocation points as:

$$N_{\partial\Omega} = \sum_{i=1}^{n_{\partial\Omega}} |T_{\partial\Omega_i}|$$

Problem in Mathematical Terms V

The Loss Functions

Suppose we use MSE to compute the Neural Networks Loss.

$$\mathrm{L}_0 := \mathrm{L}_0(\Gamma_0, \theta) = \sum_{T_{0_i} \in \Gamma_0} \frac{1}{|T_{0_i}|} \sum_{\varphi \in T_{0_i}} ||g_i(\varphi, \hat{u}(\varphi))||_2^2 \tag{2}$$

$$\mathrm{L}_{\partial\Omega} := \mathrm{L}_{\partial\Omega}(\Gamma_0, \theta) = \sum_{T_{\partial\Omega_i} \in \Gamma_{\partial\Omega}} \frac{1}{|T_{\partial\Omega_i}|} \sum_{\varphi \in T_{\partial\Omega_i}} ||h_i(\varphi, \hat{u}(\varphi))||_2^2 \tag{3}$$

$$\mathrm{L}_\Omega := \mathrm{L}_\Omega(T_\Omega, \theta) = \frac{1}{|T_\Omega|} \sum_{\varphi \in T_\Omega} ||f_i(\varphi, \hat{u}(\varphi))||_2^2 \tag{4}$$

$$L = \alpha_0 \mathrm{L}_0 + \alpha_{\partial\Omega} \mathrm{L}_{\partial\Omega} + \mathrm{L}_\Omega \tag{5}$$

Residuals

Notice that the functions f_i, g_i,and h_i's are the residuals that we are trying to minimize. In other words we want them to converge to 0. Thus NNs loss can be written in terms of $\mathrm{L}(f_i(\varphi), \vec{0})$, $\mathrm{L}(g_i(\varphi), \vec{0})$, and $\mathrm{L}(h_i(\varphi), \vec{0})$.

Generalization

Instead of separately handling PDEs, BCs, and ICs we can further generalize the residuals. We can think of each equation as a constraint, then each constraint has a domain and a residual. This way we will have only a single summation as a loss function.

$$\mathrm{L} := \mathrm{L}(\Gamma, \theta) = \sum_{T_i \in \Gamma} \frac{1}{|T_i|} \sum_{\varphi \in T_i} C(g_i(\varphi, \hat{u}(\varphi)), \vec{0}) \tag{6}$$

Loss

Here $C(y_{pred}, y_{true})$ represents any loss function.

How it Works I

Example

Let us consider the 1D Heat Conduction. The PDE, BCs, and IC are as follows:

$$\frac{\partial u}{\partial t} - 0.05\frac{\partial^2 u}{\partial x^2} = 0$$

$$u(0, x) = sin(3\pi x)$$

$$\left.\frac{\partial u}{\partial x}\right|_{\partial\Omega} = 0$$

The analytical solution is: $u(t, x) = cos(3\pi x)e^{-.05(3\pi)^2 t}$

How it Works II

Defining Constraints

```
def pde_residual(phi, stack):
    return stack[1][:,0] - 0.05 * stack[2][:,1,1]

def ic_residual(phi, stack):
    return stack[0] - tf.sin(3.0 * np.pi *  phi[:,1:])

def bc_residual(phi, stack):
    return stack[1][:,1] - 0

pde_con = Constraint(lhs(2, 100), pde_residual)
ic_con = Constraint(np.array([0, 1])*lhs(2, 100), ic_residual)
bc0_con = Constraint(np.array([1, 0])*lhs(2, 100), bc_residual)
bc1_con = Constraint(np.array([1, 0])*lhs(2, 100) + np.array([0,1])
                     bc_residual)
```

How it Works III

Defining Model and Training

```
heat1 = tf.keras.Sequential([
    tf.keras.layers.Input((2,)),
    tf.keras.layers.Dense(32, activation='tanh'),
    tf.keras.layers.Dense(64, activation='tanh'),
    tf.keras.layers.Dense(32, activation='tanh'),
    tf.keras.layers.Dense(16, activation='tanh'),
    tf.keras.layers.Dense(1)
])

epochs = 25000
learning_rate = 5e-4
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
loss = tf.keras.losses.MeanSquaredError()

pinn = PINN(heat1)
pinn.constraints = [pde_con, bc0_con, bc1_con, ic_con]
pinn.compile(optimizer, loss, ['mae'], order=2)
pinn.fit(epochs=epochs)
```

How it Works IV

Gradient Computation Algorithm

```
def grads(y, order):
    def _grads(x):
        size = x.shape[0]
        dim_x = x.shape[1]
        stack = []
        def _grad(y, x, order):
            with tf.GradientTape(persistent=True) as tape:
                tape.watch(x)
                if order == 1: stack.append(y(x))
                if order > 1: _grad(y, x, order - 1)
                components = tf.unstack(
                    tf.reshape(stack[-1], (size, -1)),
                    axis=1
                )
            derivatives = tf.stack(
                [tape.gradient(component, x) for component in components],
                axis = 2
            )
            stack.append(
               tf.reshape(derivatives, (size,) + order * (dim_x,) + (-1,))
            )
            del tape
        _grad(y, x, order)
        return stack
    return _grads
```

How it Works V

Training Algorithm

```
@tf.function
def train_step(self):
    trainable_params = self.model.trainable_variables
    with tf.GradientTape() as model_tape:
        resloss = 0
        for constraint in self.constraints:
            domain = constraint.domain
            residual = constraint.residual
            stack = self.grads(domain)
            resloss += self.loss(
                residual(domain, stack),
                tf.constant(0.0, dtype=tf.float32))
    gradients = model_tape.gradient(resloss, trainable_params)
    self.optimizer.apply_gradients(zip(gradients, trainable_params))
    return resloss
```

New Features

- A seperate module for domain generation.
- Efficient gradient calculation.
- Ready to use constraints, Neumann BCs, Dirichlet BCs, etc.
- Numeric derivative computation.

Future Work

- Residual based adaptive refinement.
- Constraint weights proportional to its domain area.
- Fourier Networks in combination to standard networks.
- Dimensionless inputs and outputs.

References I

Lee, H., & Kang, I. S. (1990). Neural algorithm for solving differential equations. *Journal of Computational Physics*, *91*(1), 110–131. https://doi.org/10.1016/0021-9991(90)90007-N

Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1997). Artificial neural network methods in quantum mechanics. *Computer Physics Communications*, *104*(1), 1–14. https://doi.org/10.1016/S0010-4655(97)00054-4

Lagaris, I., Likas, A., & Fotiadis, D. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, *9*(5), 987–1000. https://doi.org/10.1109/72.712178

Silvescu, A. (1999). Fourier neural networks [ISSN: 1098-7576]. *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No.99CH36339)*, *1*, 488–491 vol.1. https://doi.org/10.1109/IJCNN.1999.831544

References II

Lagaris, I., Likas, A., & Papageorgiou, D. (2000). Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, *11*(5), 1041–1049. https://doi.org/10.1109/72.870037

Sainio, J. (2010). CUDAEASY - a GPU accelerated cosmological lattice program. *Computer Physics Communications*, *181*(5), 906–912. https://doi.org/10.1016/j.cpc.2010.01.002

Rudd, K., Muro, G. D., & Ferrari, S. (2014). A constrained backpropagation approach for the adaptive solution of partial differential equations [Conference Name: IEEE Transactions on Neural Networks and Learning Systems]. *IEEE Transactions on Neural Networks and Learning Systems*, *25*(3), 571–584. https://doi.org/10.1109/TNNLS.2013.2277601

References III

Rudd, K., & Ferrari, S. (2015). A constrained integration (CINT) approach to solving partial differential equations using artificial neural networks. *Neurocomputing*, *155*, 277–285. https://doi.org/10.1016/j.neucom.2014.11.058

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., . . . Zheng, X. (2016, March 16). TensorFlow: Large-scale machine learning on heterogeneous distributed systems. https://doi.org/10.48550/arXiv.1603.04467

Klambauer, G., Unterthiner, T., Mayr, A., & Hochreiter, S. (2017, September 7). Self-normalizing neural networks. https://doi.org/10.48550/arXiv.1706.02515

Betancourt, M. (2018, December 30). A geometric theory of higher-order automatic differentiation. https://doi.org/10.48550/arXiv.1812.11592

References IV

Han, J., Jentzen, A., & E, W. (2018, July 3). Solving high-dimensional partial differential equations using deep learning. https://doi.org/10.1073/pnas.1718942115

Magill, M., Qureshi, F., & de Haan, H. W. (2018, June 29). Neural networks trained to solve differential equations learn general representations. https://doi.org/10.48550/arXiv.1807.00042

Sirignano, J., & Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, *375*, 1339–1364. https://doi.org/10.1016/j.jcp.2018.08.029

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., & Duvenaud, D. (2019, December 13). Neural ordinary differential equations. https://doi.org/10.48550/arXiv.1806.07366

Dockhorn, T. (2019, April 15). A discussion on solving partial differential equations using neural networks. https://doi.org/10.48550/arXiv.1904.07200

References V

Koryagin, A., Khudorozkov, R., & Tsimfer, S. (2019, September 25). PyDEns: A python framework for solving differential equations with neural networks. https://doi.org/10.48550/arXiv.1909.11544

Piscopo, M. L., Spannowsky, M., & Waite, P. (2019). Solving differential equations with neural networks: Applications to the calculation of cosmological phase transitions. *Physical Review D*, *100*(1), 016002. https://doi.org/10.1103/PhysRevD.100.016002

Regazzoni, F., Dedè, L., & Quarteroni, A. (2019). Machine learning for fast and reliable solution of time-dependent differential equations. *Journal of Computational Physics*, *397*, 108852. https://doi.org/10.1016/j.jcp.2019.07.050

Avrutskiy, V. I. (2020). Neural networks catching up with finite differences in solving partial differential equations in higher dimensions. *Neural Computing and Applications*, *32*(17), 13425–13440. https://doi.org/10.1007/s00521-020-04743-8

References VI

Hennigh, O., Narasimhan, S., Nabian, M. A., Subramaniam, A., Tangsali, K., Rietmann, M., Ferrandis, J. d. A., Byeon, W., Fang, Z., & Choudhry, S. (2020, December 14). NVIDIA SimNet^{TM}: An AI-accelerated multi-physics simulation framework. https://doi.org/10.48550/arXiv.2012.07938

Rao, C., Sun, H., & Liu, Y. (2020). Physics-informed deep learning for incompressible laminar flows. *Theoretical and Applied Mechanics Letters*, *10*(3), 207–212. https://doi.org/10.1016/j.taml.2020.01.039

Shen, X., Cheng, X., & Liang, K. (2020, March 21). Deep euler method: Solving ODEs by approximating the local truncation error of the euler method. https://doi.org/10.48550/arXiv.2003.09573

Sitzmann, V., Martel, J. N. P., Bergman, A. W., Lindell, D. B., & Wetzstein, G. (2020, June 17). Implicit neural representations with periodic activation functions. https://doi.org/10.48550/arXiv.2006.09661

References VII

Araz, J. Y., Criado, J. C., & Spannowsky, M. (2021, March 30). Elvet – a neural network-based differential equation and variational problem solver. https://doi.org/10.48550/arXiv.2103.14575

Guidetti, V., Muia, F., Welling, Y., & Westphal, A. (2021, March 15). dNNsolve: An efficient NN-based PDE solver. https://doi.org/10.48550/arXiv.2103.08662

Lu, L., Meng, X., Mao, Z., & Karniadakis, G. E. (2021). DeepXDE: A deep learning library for solving differential equations [Publisher: Society for Industrial and Applied Mathematics]. *SIAM Review*, *63*(1), 208–228. https://doi.org/10.1137/19M1274067

McClenny, L. D., Haile, M. A., & Braga-Neto, U. M. (2021, March 29). TensorDiffEq: Scalable multi-GPU forward and inverse solvers for physics informed neural networks [version: 1]. https://doi.org/10.48550/arXiv.2103.16034

References VIII

Peng, W., Zhang, J., Zhou, W., Zhao, X., Yao, W., & Chen, X. (2021, July 9). IDRLnet: A physics-informed neural network library. https://doi.org/10.48550/arXiv.2107.04320

Racca, A., & Magri, L. (2021, March 24). Automatic-differentiated physics-informed echo state network (API-ESN). https://doi.org/10.48550/arXiv.2101.00002

Bettencourt, J., Johnson, M. J., & Duvenaud, D. (2022). Taylor-mode automatic differentiation for higher-order derivatives in JAX. Retrieved March 18, 2023, from https://openreview.net/forum?id=SkxEF3FNPH

Margenberg, N., Hartmann, D., Lessig, C., & Richter, T. (2022). A neural network multigrid solver for the navier-stokes equations. *Journal of Computational Physics*, *460*, 110983. https://doi.org/10.1016/j.jcp.2022.110983

References IX

Passing tf.keras.model as tf.function argument does not create concrete function · issue #38875 · tensorflow/tensorflow [GitHub]. (n.d.). Retrieved March 16, 2023, from https://github.com/tensorflow/tensorflow/issues/38875