Not All Sensors Are Equal

Nicolas Nytko, Sean Farhat, Nathanael Assefa

May 2, 2023

Deep Onet Review

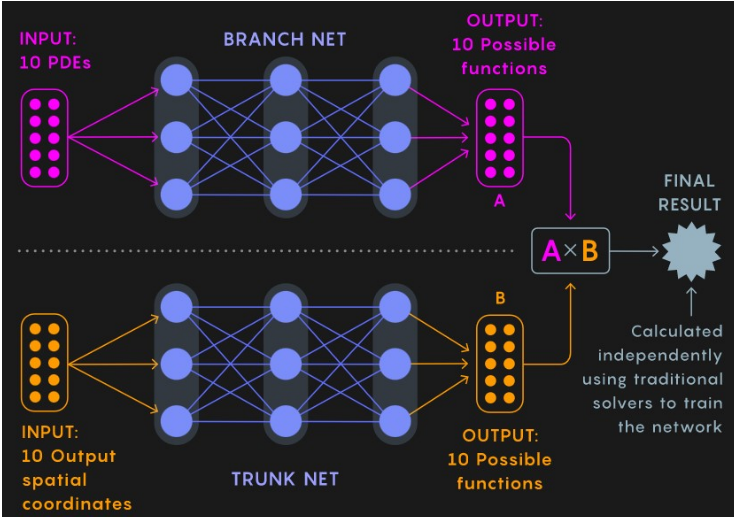
A *DeepOnet* is a mapping $\mathcal{O} : C(\mathbb{T}^d; \mathbb{R}^{d_a}) \to C(\mathbb{T}^d; \mathbb{R}^{d_v})$ of the form

$$\mathcal{O}(a)(x) = \sum_{k=1}^{p} \beta_k(a(x_1), \ldots, a(x_p))\tau_k(x), \tag{1}$$

where $\beta_i : \mathbb{R}^{m \times d_a} \to \mathbb{R}^{p \times d_u}$ and $\tau_j : \mathbb{R}^d \to \mathbb{R}^p$ are the *branch* and *trunk* networks, respectively.

The function a is evaluated at some discrete *sensor points* $x_1, \ldots, x_m$.

DeepONet Architecture



https://www.quantamagazine.org/latest-neural-nets-solve-worlds-hardest-equations-faster-than-ever-before-20210419/

DeepONet Approximation Theorem

Theorem (Universal Operator Approximation)

Suppose that X *is a Banach space,* $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ *are two compact sets in* X *and* $\mathbb{R}^d$*, respectively,* V *is a compact set in* $C(K_1)$ *. Assume that* $G : V \to C(K_2)$ *is a nonlinear continuous operator. Then, for any* $\epsilon > 0$*, there exist positive integers* m, p*, continuous vector functions* $\mathbf{g} : \mathbb{R}^m \to \mathbb{R}^p, \mathbf{f} : \mathbb{R}^d \to \mathbb{R}^p$*, and* $x_1, x_2, \ldots, x_m \in K_1$*, such that,*

$$\left| G(u)(y) - \langle \underbrace{\mathbf{g}\left(u\left(x_1\right), u\left(x_2\right), \cdots, u\left(x_m\right)\right)}_{\text{branch}}, \underbrace{\mathbf{f}(y)}_{\text{trunk}} \rangle \right| < \epsilon$$

- An integral is an example of an operator G acting on function u evaluated at y in $G(u)(y)$
- The loss function is the mean squared error (m.s.e.) between the true value of $G(u)(y)$ and the network prediction for the input $([u(x_1), u(x_2), \ldots, u(x_m)], y)$.
- DeepONet is a high-level architecture without defining the specific architectures of the inner trunk and branch nets

Sensor Points in ONets

1. The choice of sensor points $x_1, \ldots, x_m$ is chosen beforehand.
2. We are fixed to the sensor points once we have chosen them. All input functions must be evaluated at the same sensors
3. **Problem:** The choice of these points is arbitrary. Can we algorithmically find a better set of sensors?
4. **Solution:** Treat them like hyperparameters

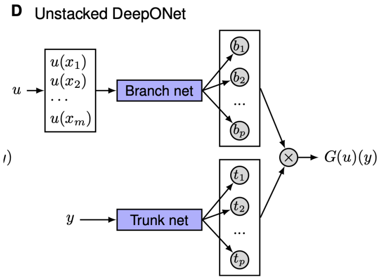


Figure: While we can probe the solution $G(u)$ at any point y, all input functions must be evaluated at the sensors x

Sensors as Hyperparameters

For an O-Net parameterzied by θ, f_θ optimized via gradient descent, we can nest its learning process into an outer hyper-optimization that can find a better set of sensors to probe inputs at:

$$\min_{\mathbf{x}} \mathcal{L}\left(u(\mathbf{x}), \mathbf{y}, \underbrace{\theta - \eta \nabla_\theta \mathcal{L}(u(\mathbf{x}), \mathbf{y}, \theta)}_{\text{learning steps to optimize } f_\theta} \right)$$

In practice: let the inner optimization run for a few steps, optimize the sensors, and repeat.

Algorithm Considerations

1. **Complexity:** The nested optimization scales in resources with the number of inner optimization steps taken. However, approximations leveraging the *Implicit Function Theorem*[1] allow us to let the process unroll for much longer at a reasonable cost.
2. **Generalization:** While a generally 'optimal' set of sensors may exist, we believe it is more useful to constrain the O-Net to learn from a family of related PDE problems s.t. the learned sensors can reasonably generalize.
3. **Implications:** The learned sensors can be used in the O-Net to solve a PDE, but only up to a certain timestep. We can also investigate the quality of the sensors on the same problem solved by a standard PDE solver.

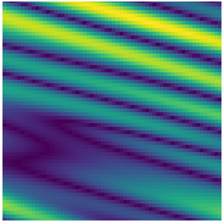
[1] Jonathan Lorraine, Paul Vicol, and David Duvenaud. *Optimizing Millions of Hyperparameters by Implicit Differentiation*. 2019. arXiv: 1911.02590 [cs.LG].

Steady-state Diffusion PDE

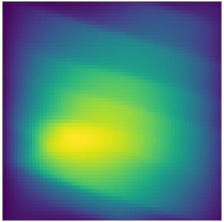
On the unit square, $\Omega = [0, 1]^2$,

$$-\nabla \cdot (\kappa(x, y)\nabla u) = f, \tag{2}$$

with $u(\partial\Omega) = 0$, and $f \equiv 1$. Randomly generate $\kappa(x, y)$ with Simplex noise.



$\kappa(x, y)$



$u(x, y)$

Experiment Setup

1. 200 outer epochs
2. 50 inner epochs
3. ADAM optimizes the O-Net
4. RMSProp optimizes the sensors
5. Inner loss on train set
6. Outer loss on validation set
7. Nested *hypergradient* requires an inverse Hessian that we approximate with a Neumann series with 3 terms
8. Clip the new sensor points to ensure they're inside the domain

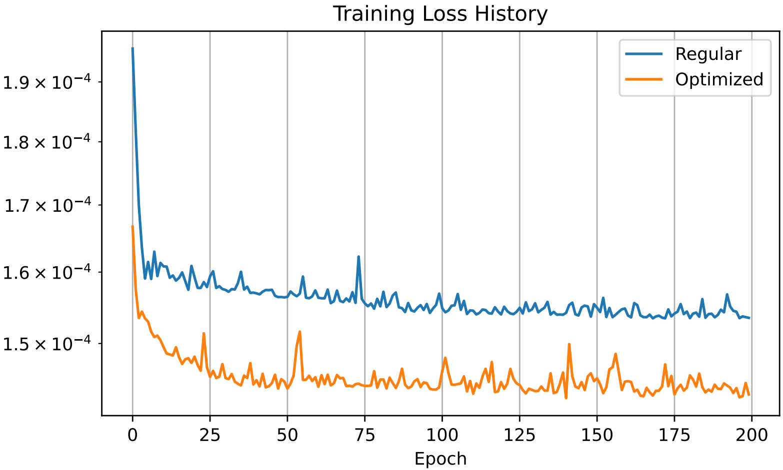
MSE Loss

We compute the mean-squared-error between the ONet evaluated at a uniform set of points (not sensor points), and the interpolated true solution from a PDE solver

$$\ell := \frac{1}{B}\frac{1}{N}\sum_{b=1}^{B}\sum_{i=1}^{N}\big(\mathcal{N}(\kappa)(x_i, y_i) - u(x_i, y_i)\big)^2,$$

where $\mathcal{N}$ is our learned ONet operating on $\kappa(x, y)$.

Loss History



Optimized points