Proposal: solving PDEs using SVMs by symbolically integrating

Zander Op de Beeck

July 15, 2024

1 Problem statement

A PDE can be written as

$$\begin{aligned} \mathcal{L}u &= f \quad \text{on} \quad \Sigma \\ \mathcal{B}u &= g \quad \text{on} \quad \partial\Sigma \end{aligned}$$

where

u	is the (vector-)function to be found,
$\mathcal{L}$	is the (differential) operator specifying the PDE,
f	further specifies the PDE,
Σ	is the domain on which the problem is to be solved,
$\mathcal{B}$	is the (differential) operator specifying the boundary conditions,
g	further specifies the boundary conditions, and
$\partial\Sigma$	is the boundary of the domain.

2 Previous methods

Outside of the usual methods, including FEM, FDM, spectral methods and BIEs, as well as approaches based on neural networks, [2] introduced a way of solving PDEs using LS-SVMs. In this method,

1. Two sets of collocation points are chosen; one in the domain ($\mathcal{Z}_\mathcal{D}$) and one on the boundary ($\mathcal{Z}_\mathcal{B}$).
2. A loss function is set up by summing the error $\frac{1}{2}\left(\mathcal{L}u - f\right)^2$ at all points in $\mathcal{Z}_\mathcal{D}$.
3. The boundary conditions are incorporated as hard constraints; $\mathcal{B}u = g$ is to be satisfied for all points in $\mathcal{Z}_\mathcal{B}$.
4. u is assumed to be of the form $u(z) = w^T\varphi(z) + d$, as in LS-SVM, and the problem can be solved either in the primal or the dual.

The final objective then becomes

$$\begin{aligned} \underset{u}{\text{minimize}} \quad & \frac{1}{2}\sum_{i=1}^{|\mathcal{Z}_\mathcal{D}|}\left[\left(\mathcal{L}[u] - f\right)\left(z_\mathcal{D}^i\right)\right]^2 \\ \text{subject to} \quad & \forall j \in 1 \ldots |\mathcal{Z}_\mathcal{B}| : \ \mathcal{B}\left[u\left(z_\mathcal{B}^i\right)\right] = g\left(z_\mathcal{B}^i\right). \end{aligned}$$

3 Proposed method

3.1 Problems to overcome

One of the niceties PDEs have over data-based tasks is that, in principle, the loss function can be calculated anywhere on the domain. [2] chooses to first pick its collocation points and then leave them where they are, providing a rather efficient and very tractable method. This, however, also comes with a few disadvantages:

1. The location (and number) of collocation points may not be optimal.
2. for high-dimensional problems, the number of points needed to adequately sample the domain grows exponentially (curse of dimensionality).
3. Points spaced too far apart may not pick up on fine detail, and may not even account for it in the loss.

3.2 Proposed method

3.2.1 Loss and model

On way of alleviating these problems somewhat may be to learn the location of the collocation points themselves in addition to what [2] does. This would, however, come with a serious problem: the loss function

$$\underset{u,\mathcal{Z}_\mathcal{D}}{\text{minimize}} \quad \frac{1}{2}\sum_{i=1}^{|\mathcal{Z}_\mathcal{D}|}\left[\left(\mathcal{L}[u] - f\right)\left(z_\mathcal{D}^i\right)\right]^2$$

calculates the error only at the collocation points themselves, thus incentivizing the model to choose collocation points at places where the error is low anyways, which would result in a rather poor solution.

A loss independent of the location of the point is thus required[1], which an explicit integration over the domain would accomplish:

$$\underset{u,\mathcal{Z}_\mathcal{D}}{\text{minimize}} \quad \frac{1}{2}\int_\mathcal{D}\left[\left(\mathcal{L}[u] - f\right)(z)\right]^2 dz.$$

This integral may not be intractable, for example, if the polynomial kernel $K(x,y) = \left(x^Ty + c\right)^d$ is chosen, the resulting model is a polynomial as well, which can easily be integrated symbolically, as can its derivatives and products with other polynomials, and so on. Assuming Σ is finite, any nonpolynomial function can be

[1] Thus relieving the collocation points from their role in calculation the errors, making them sample points.

approximated on the domain as a polynomial, and if a bound on the range of u is known, any nonpolynomial function that would take u as an argument could be polynomially approximated over this range as well.

(I have also done some small-scale experiments with RBF kernels in a computer algebra system, which worked as intended as well.)

For other kernels, other approximations of functions in the domain could be made. Moreover, picking a "working domain" that contains Σ but has a more convenient shape would be possible.

3.2.2 Boundary conditions

[2] chose to incorporate the boundary conditions as hard constraints. It would be impossible to symbolically integrate over (an approximation of) the boundary and still treat it as a constraint in this way; for one, the model has a finite number of parameters, which, in general, would not suffice for perfectly fitting u on the entire boundary. The boundary, however, can be associated with a loss

$$(\mathcal{B}[u(z)] - g(z))^2,$$

leaving four options:

1. Pick a set of boundary points $\mathcal{Z}_\mathcal{B}$ beforehand and impose $\forall j \in 1 \ldots |\mathcal{Z}_\mathcal{B}| : \mathcal{B}[u(z^i_\mathcal{B})] = g(z^i_\mathcal{B})$ as a hard constraint, as was done in [2].
2. Pick a set of boundary points $\mathcal{Z}_\mathcal{B}$ beforehand and add their loss to the objective function (perhaps using a weighting).
3. Symbolically integrate over (an approximation of) the boundary conditions and add to the loss (perhaps using a weighting).
4. A mixture of these.

The third option would result in an optimization problem

$$\begin{aligned}\underset{u,\mathcal{Z}}{\text{minimize}} \quad & \frac{1}{2}\int_\mathcal{D} [(\mathcal{L}[u] - f)(z)]^2\, dz \\ & + \gamma_b \int_{\partial\Sigma} [(\mathcal{B}[u] - g)(z)]^2\, dz.\end{aligned}$$

Where γ_b is a hyperparameter specifying the importance of fitting the boundary conditions. The same remarks about the integration often being symbolically tractable as in the section on the PDE itself apply here.

Picking points for boundary conditions scales a bit better than picking points in Σ, after all, $\partial\Sigma$ is of lower dimensionality.

3.2.3 Supposed advantages and limitations

While symbolic integration over the domain does away with the curse of dimensionality for calculating a loss, it does not in any way give the model more flexibility to fit a high-dimensional function. This would have to be done by picking enough sample points and having a flexible-enough model in the first place. Having the points place themselves as they wish might help in that.

The glaring limitation of this approach is that the integrals need be symbolically doable, though this can be alleviated somewhat by picking easily integrable approximations to the original problem.

3.3 Extensions and considerations

Some extensions and remarks:

1. Optimization could be done ether all at once or alternately updating sample points and SVM model.
2. Instead of calculating (higher) derivatives in $\mathcal{L}$ explicitly, one could use finite differences $\frac{u(z+\delta)-u(z)}{\delta}$, resulting in a linear combination of u and u shifted by δ. This may make integrals easier.
3. Many structures are given as meshes in 3D; sets of triangles. Especially for polynomial models, this seems easy to integrate over, providing boundary conditions.
4. (For low-dimensional problems), maybe monte-carlo integration could be used to create a stochastic optimization method, keeping the sample points distinct from the points used for approximation.
5. The general idea of this scheme needs not apply only to basic kernel-based methods, other (tensor-based, deep) models could be used as well, provided the integrations can be carried out.
6. Recently, in computer graphics, a method called "Gaussian splatting" has been used to great success for reconstructing 3D scenes from images [1]. The model used here consists of 3D Gaussians, not unlike (the dual representation of) SVMs using the RBF kernel, except for the covariance matrix being part of the learnable parameters (which could also be added in the SVM formulation). Given its success in approximating the real world, it may be a good model for PDEs as well.
7. The number of points taken has thusfar been assumed to be fixed. A method for adding or removing points depending on how much they contribute to the overall model could be implemented, as has been done successfully in computer graphics [1].
8. Instead of support vectors, which are summed over in an SVM model, a "support function" could be used, (which may be a low-rank outer product of functions in each variable). The dual representation of the model would use an integral over an α-function instead of a discrete sum over α_k.
9. Say one has a system of PDEs, i.e. u is a vector function. Would is be best to have different sets of points $\mathcal{Z}$ for each component of u or might one

have only a single set $\mathcal{Z}$ together with weights on how much each part of the model contributes to each component of u?

References

[1] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023.

[2] Siamak Mehrkanoon and Johan Suykens. Learning solutions to partial differential equations using ls-svm. 2015.