1 Write Up

General Concept:

Consider a general PDE

$$\nabla^2 f + \nabla f + f + c = 0 \tag{1}$$

with boundary conditions

$$f(\cdot, T) = g(\cdot) \tag{2}$$

where ∇^2 denotes second derivatives, and ∇ denotes first derivatives. Assume for now that at least one solution exists, and that if there are multiple solutions we have no preference among them. We can label the domain S.

Our goal is to approximate the solution function f with a neural network estimate of f that we can call $\hat{f}$. This can be thought of as a minimization problem, where the goal is to minimize this quantity $\mathscr{L}$:

$$\mathscr{L}_S = \int\int \cdots \int \int_S ((\nabla^2 f + \nabla f + f + c) - (\nabla^2 \hat{f} + \nabla \hat{f} + \hat{f} + c))^2 (\boldsymbol{x}, t) dx_1 dx_2 \ldots dx_n dt \tag{3}$$

$$\mathscr{L}_B = \int\int \cdots \int_B ((g(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}, T))^2 dx_1 dx_2 \ldots dx_n \tag{4}$$

$$\mathscr{L} = \mathscr{L}_B + \mathscr{L}_S \tag{5}$$

For a function $\hat{f}$ to be a solution of (1) subject to (2), its a necessary condition that $\frac{\partial \mathscr{L}}{\partial \theta_i} = 0$ where θ_i is a parameter in $\hat{f}$.

Fitting $\hat{f}$:

So one way of fitting $\hat{f}$ could be to sample repeatedly through some scheme a collection of n points in S, and a collection of k points on the boundary B and then let this quantity be our objective:

$$\frac{1}{k}\sum_{i=1}^{k}(g(\boldsymbol{x}_i) - \hat{f}(\boldsymbol{x}_i, T))^2 + \frac{1}{n}\sum_{i=1}^{n}(\nabla^2 \hat{f} + \nabla \hat{f} + \hat{f} + c)(\boldsymbol{x}_i, t_i)^2 \tag{6}$$

and minimize it by doing our usual gradient descent algorithm updating the parameters of $\hat{f}$. It's possible that there exists some function ψ that isn't a solution to (1) subject to (2), but does satisfy the necessary condition, so this might be a local minima that our algorithm gets stuck at.

Mesh Sampling:

One sampling scheme could be by sampling points on a mesh. Definitely a good place to start but it doesn't seem like it would be any faster than a non-NN method since its using a mesh. Maybe it could be if the NN somehow finds a pattern to exploit.

GAN sampling:

Another complicated sampling scheme could be to introduce another network(s) that tries to generate points in the domain that our approximator network $\hat{f}$ performs badly on $((\nabla^2\hat{f} + \nabla\hat{f} + \hat{f} + c)^2(\boldsymbol{x}, t) >> 0)$. Lets call these networks G, and G_B. If the domain of our PDE is n-dimensional, G takes in an n-dimensional noise vector and the outputs another n dimensional vector. G_B would take in an n-1 dimensional noise vector, and spit out another n-1 dimensional vector. So our new objective could be

$$\frac{1}{k}\sum_{i=1}^{k}(g(G_B(\sigma_{i1})) - \hat{f}(G_B(\sigma_{i1}), T))^2 + \frac{1}{n}\sum_{i=1}^{n}(\nabla^2\hat{f} + \nabla\hat{f} + \hat{f} + c)(G(\sigma_{i2}), t_i)^2 \tag{7}$$

We would try to minimize this with respect to the parameters of $\hat{f}$ and maximize it WRT the parameters of G and G_B like in a GAN. The hope is that through this adversarial setup, when we deal with high dimensional PDEs this would be some intelligent way of sampling points and be faster than a mesh.

Monte-Carlo Sampling:

Lots of PDEs arise from some sort of dynamic programming problem. If our PDE is a stochastic Hamilton Jacobi Bellman equation, one way of solving the PDE could be to start with some expression that we already have, either for a simplified version of the problem or some sort of analytic approximation to the optimal solution. Armed with this starting solution, we could replay the stochastic environment over and over, taking the actions that our starting solution to the HJB equation implies that we should take. Doing this over and over will produce a monte-carlo exploration of our state space with some sub-optimal policy. We can store the paths through state space that are produced by this method, and then fit $\hat{f}$ using the objective defined above, evaluated on point along these paths. Once we have made some progress, (fitted $\hat{f}$ to a point where it performs better than our starting solution), we can produce more data points with our $\hat{f}$, and then train on these paths. This should converge to some optimum.

Potential Issues

1. GAN training is very hard in general
2. There are lots of GANs here. Maybe we can find a way to do a transformation on the output of $\hat{f}$ like in the paper I sent you so that we can simplify our loss function and use less GANs.
3. AFAIK neural networks are probably better at classification than something like this (having a discriminator label things 0 or 1 is probably a lot more stable than what this is). Maybe this GAN sampling can be combined with the classification that the paper I sent you does somehow to remedy this.