Neural methods for solving calculus of variation problems

Yury Prokhorov
Optimization Class Project. MIPT

Introduction

A calculus of variations problem is minimizing an integral functional over a given set of functions. Such problems naturally arise in many practical engineering cases. A general way of tackling them involves solving an Euler-Lagrange equation which is a second order differential equation. In many cases the complexity of this equation does not allow for a solution by quadratures, thus, numerical methods are necessary.

Generally, numerical methods are categorized as direct and indirect methods. Direct methods convert a problem into a finite dimensional one, while indirect methods attempt to find a solution of an Euler-Lagrange equation. In present work a direct method using neural networks is proposed.

Problem statement

An optimization problem is stated as follows

$$J[x(t)] = \int_{t_0}^{t_\mathsf{f}} L\big(t, x(t), \dot{x}(t)\big)\, dt \longrightarrow \min_{x(t)}$$

where $x(t)$ is a differentiable function on interval $[t_0, t_\mathsf{f}]$.

Often boundary conditions are added:

$$x(t_0) = x_0, \qquad x(t_\mathsf{f}) = x_\mathsf{f}$$

There also are other variations in which one or both boundary conditions are missing, or there is an extra integral condition (isoperimetric problem).

Discretization

There are two continuous features that are evaluated numerically:

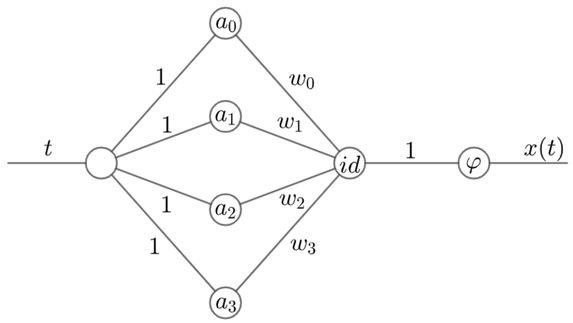
1. Derivative $\dot{x}(t)$.
   It can be estimated using finite differences. In present work the central differences method is used: $\dot{x}(t) \approx \frac{x(t+h)-x(t-h)}{2h}$.
2. Integral $J[x]$.
   The integral can be evaluated using numerical methods with an N points split. In present work trapezoid rule and Simpson's rule are applied.

The problem is reduced to minimizing $\widehat{J}(x)$, where $\widehat{J}$ is a numerical approximation of the integral and x is a finite dimensional vector whose components are used to compute $\widehat{J}$ and $\dot{x}$.

The dimensionality of such problem is $\mathcal{O}(N)$. However, knowing that components of x should form a smooth function, we can reduce the problem to $\mathcal{O}(1)$ by approximating a function in a given class. In present work, neural networks are used to achieve such result.

Neural network architecture

The neural network approximates an optimal solution, therefore, it has a single input and a single output. The simplest case can be visualized as a following diagram



The idea is to use different activation functions $a_k(t)$. Some examples:

$$a_k(t) = \cos\frac{\pi k(t-t_0)}{t_\mathsf{f}-t_0}, \qquad a_k(t) = (t-t_0)^k$$

Such network will act as basis expansion (trigonometric series or power series) of the solution. More advanced networks have more hidden layers to allow for more variability.

An extra activation layer φ does a smooth transform to conform to boundary conditions. In present work the following transform is utilized:

$$\varphi(t,x) = x_0 + \frac{x_\mathsf{f}-x_0}{t_\mathsf{f}-t_0}(t-t_0) + (t-t_0)(t-t_\mathsf{f})x$$

Training

The network is trained to minimize the integral loss $\widehat{J}$ using gradient methods:

- Vanilla GD with momentum
- L-BFGS
- Adam

The PyTorch framework with its Autograd tool was used for training.

Hyperparameters

The following parameters of the model can be altered to find better results:

- Integral approximation (e.g. Simpson's rule, etc.) and its precision
- Derivative approximations with finite differences and their precision
- Different numbers of layers and neurons, different activation functions
- Boundary condition transform
- Optimization algorithm and its parameters

Numerical example

Consider a particular problem:

$$J[x] = \int_0^\pi \left[(\dot{x}+x)^2 + 2x\sin(t)\right] dt \longrightarrow \min, \quad x(0)=0, \quad x(\pi)=1$$

Its discrete version can be solved using the gradient descent algorithm. For comparison, a neural network was used too. It contained one hidden cosine layer with 8 neurons and its training also involved the gradient descent algorithm with momentum.

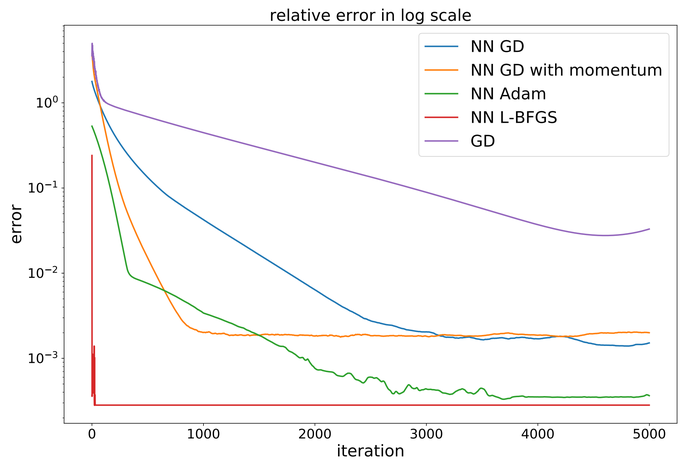
All other hyperparameters were chosen to be the same in both methods (integral approximation, learning rate scheduling, etc.)

Results

The quality metric chosen here is relative L1-error between numerical and exact solutions:

$$\mathsf{RelError}(x, x^*) = \frac{\frac{1}{N}\sum_k |x(t_k) - x^*(t_k)|}{\frac{1}{N}\sum_k |x^*(t_k)|}$$

In this numerical example the neural algorithm converges to a much better result than a naive approach in terms of precision.



Conclusion

As a result of present work, a flexible neural algorithm for solution of different calculus of variations problems has been developed. Testing has shown that it is able to achieve better precision than some naive approaches and is on par with other optimization methods for variational problems.

Link to GitHub repository.