Learning to integrate

DANIEL MAÎTRE
IPPP, DURHAM UNIVERSITY

Based on arXiv:2211.02834 with Roi Santos-Mateos
Consider parametric integrals of the form

\[ I(s_1, \ldots, s_m) = \int_0^1 dx_1 \cdots \int_0^1 dx_k f(s_1, \ldots, s_m; x_1, \ldots, x_k) \]

- \( x_i \) are auxiliary variables and \( s_i \) are the parameters
- Example: sector decomposition of loop integrals
Typical solution

- Monte Carlo integration for each value of the parameters
- Each run is independent
Alternative

- Sample the $x$-$s$ space more uniformly

- Can leverage information on the integrand between separate evaluations
Suppose we had a function with

\[
d^k F(s_1, \ldots, s_m; x_1, \ldots, x_k) = f(s_1, \ldots, s_m; x_1, \ldots, x_k)
\]

We can evaluate the integral as

\[
I(s_1, \ldots, s_m) = \sum_{x_1, \ldots, x_k = 0,1} (-1)^{k-\sum x_i} F(s_1, \ldots, s_m; x_1, \ldots, x_k)
\]
We introduce a neural network approximation for the primitive function 

\[ \mathcal{N}(s_1, \ldots, s_m; x_1, \ldots, x_k) \]

Train it such that its derivative matches the integrand function

\[ L = \text{MSE} \left( f(s_1, \ldots, s_m; x_1, \ldots, x_k), \frac{d\mathcal{N}(s_1, \ldots, s_m; x_1, \ldots, x_k)}{dx_1 \ldots dx_k} \right) \]
Standard network with $L$ hidden layers

\[ a_i^{(l)} = \phi \left( z_i^{(l)} \right), \quad z_i^{(l)} = \sum_j w_{ij}^{(l)} a_j^{(l-1)} + b_i^{l}. \]

Inputs

\[ a_i^{(0)} = x_i \text{ for } i \leq k, \quad a_i^{(0)} = s_{i-k} \text{ for } i > k. \]

output

\[ y = \sum_j w_j^{(L+1)} a_j^{(L)} + b^{(L)}. \]
Derivatives in the loss function

- The derivative in the loss function contains all the derivatives of the activation function up to degree k

\[
\frac{dp_{z_i}^{(l)}}{dx_1 dx_2 \ldots dx_p} = \sum_j w_{ij}^{(l)} \frac{dp_{a_i}^{(l-1)}}{dx_1 dx_2 \ldots dx_p}
\]

\[
\frac{d^3 a_i^{(l)}}{dx_1 dx_2 dx_3} = \phi''(z_i^{(l)}) \left[ \frac{d^2 z_i^{(l)}}{dx_1 dx_2} \frac{dz_i^{(l)}}{dx_3} + \frac{dz_i^{(l)}}{dx_1} \frac{d^2 z_i^{(l)}}{dx_2 dx_3} + \frac{d^2 z_i^{(l)}}{dx_1 dx_2} \frac{dz_i^{(l)}}{dx_3} \right]
\]

\[
+ \phi'(z_i^{(l)}) \frac{d^2 z_i^{(l)}}{dx_1 dx_2 dx_3}
\]
Example 1

- 1-loop box for $gg \rightarrow HH$
- Four physical parameters: $m_t^2$, $m_H^2$, $s_{12}$, $s_{14}$
- Three Feynman parameters $x_1$, $x_2$, $x_3$
- 3 sectors generated by pySecDec
- Euclidean region

$$-30 \leq s_{12}/m_t^2 \leq -3 \quad -30 \leq s_{14}/m_t^2 \leq -3 \quad -30 \leq m_H^2/m_t^2 \leq -3$$
Result

- 100 nodes
- 4 hidden layers
- $4M \times 800 = 3.2B$ PS points

$$p = \log_{10} \left| \frac{e - t}{t} \right|$$
Example 2

- 2-loop box for $gg \rightarrow HH$
- Same physical parameters
- 6 Feynman parameters
- 1 of 30 sectors from pySecDec
Results

- 30 nodes
- 4 hidden layers
- 800k x 200 PS points
Error estimate

- Use 4 replicas of the network
- Use average as the prediction
- Standard deviation as error estimate
Reducing variance

- **Usual subtraction**

\[
\int_{0}^{1} \cdots \int_{0}^{1} dx_1 \cdots dx_k \left( f(x_1, \ldots, x_k) - s(x_1, \ldots, x_k) \right) + S
\]

- **Using our neural network**

\[
\int_{0}^{1} \cdots \int_{0}^{1} dx_1 \cdots dx_k \left( f(x_1, \ldots, x_k) - \frac{d^k N}{dx_1 \cdots dx_k} \right) + \sum_{b_i=0,1} \pm N(b_i, \ldots, b_k)
\]
Reducing variance

- Usual subtraction

\[ \int_0^1 dx_1 \ldots \int_0^1 dx_k \left( f(x_1, \ldots, x_k) - s(x_1, \ldots, x_k) \right) + S = \int_0^1 dx_1 \ldots \int_0^1 dx_k s(x_1, \ldots, x_k) \]

- Using our neural network

\[ \int_0^1 dx_1 \ldots \int_0^1 dx_k \left( f(x_1, \ldots, x_k) - \frac{d^k N}{dx_1 \ldots dx_k} \right) + \sum_{b_i=0,1} \pm N(b_i, \ldots, b_k) \]

Lower variance!
Reduce variance

- 1-Loop example
- Ratio of variance with NN subtraction wrt without subtraction
Outlook

- More work on training
  - Initialisation
  - Training data sequencing
- Size / depth of networks / number of replica
- More complicated examples
  - More integration variables
  - Combined sectors, combined integrals
  - Minkowsky space
- Use as a parametrized Gibbs sampler
Conclusion

- Proof of concept
- Integral from fitting integrand
- Lots to learn about the behaviour/training of network with derivative loss
Neural network training is similar to standard network but

- Take care of initialization
- Can choose our data
  - Random vs qmc grids
  - Size of sample
  - Re-use or generate new data
- Pick activation function
  - Tanh/sigmoid in $N$: derivatives in Loss
  - Antiderivatives of tanh/sigmoid in $N$: tanh and sigmoid in loss (and lower antiderivatives)
Preprocessing

- Korobov transform

\[ x = t^2(3 - 2t), \quad \int_0^1 dx f(x) = \int_0^1 dt \, 6t(1 - t)f(x(t)) \]

- Remove overall scaling

\[ f \rightarrow \tilde{f}(s_1, \ldots, s_m; x_1, \ldots, x_k) \equiv \frac{f(s_1, \ldots, s_m; x_1, \ldots, x_k)}{f(s_1, \ldots, s_m; \frac{1}{2}, \frac{1}{2}, \ldots, \frac{1}{2})} \]