Second-Order Differential Machine Learning

Neil Kichler

Monte Carlo methods for sampling option pricing models are expensive. Instead:

- Find efficient neural-network based surrogates for option pricing models.
- Can we go beyond Differential Machine Learning as proposed by Huge & Savine [3], using second-order information?

Key Idea

Add second-order differential data to the learning process [3]:

- Use PCA via SVD to find meaningful directions.
- Apply Hessian Vector Product on found principal components.
- (optionally) take k0 most important principal components.
- Adaptive loss balancing using singular value information.
- Implementation on the GPU/TPU using JAX.
- Potential parallel, on-the-fly data generation on the CPU with C++ using AD [3] via, e.g., dco/c++.

Algorithm

Require: Initialized…

- Surrogate model \(N(\theta)\) with parameters \(\theta\).
- Reference model \(S\).
- Optimizer \(G\).
- hyperparameter \(\kappa\), for principal components.
- Loss function \(L\).
- loss balancing parameters \(\lambda_0, \lambda_1, \lambda_2\).

1. while \(\theta\) not converged do
   2. \(\{(x_i, y_i, \nabla_x y_i)_i\}\) \(\sim S\) \(\triangleright\) Sample training data
   3. \(\mu \leftarrow \left(\frac{1}{m} \sum_{i=1}^{m} \nabla_x y_i\right)\) \(\triangleright\) Mean of pathwise gradients
   4. \(\{\nabla_x y_i\}_{i=1}^{m} \leftarrow \{\nabla_x y_i, -\mu\}_{i=1}^{m}\) \(\triangleright\) Mean subtracted data
   5. \((U_s, V_s) \leftarrow SVD\left(\{\nabla_x y_i\}_{i=1}^{m}\right)\) \(\triangleright\) Principal components
   6. \(v_{i=1}^{m} \leftarrow \text{diag}(s)V\) \(\triangleright\) mean adjusted
   7. \(s \leftarrow s/\sum(s^2)\) \(\triangleright\) Scaled \(s\) to represent % of variance
   8. \(k_0 \leftarrow \arg\max(\text{cumsun}(s_i) > \kappa)\)
   9. Gradient \(g\) of minibatch:
      \[g \leftarrow \frac{1}{m} \sum_{i=1}^{m} \lambda_i \mathcal{L}(f_0(x_i), y_i) + \lambda_i \mathcal{L}(\nabla_x f_0(x_i), \nabla_x y_i)\]
      \[+ \lambda_0 \sum_{i=1}^{k_0} \left(\frac{\partial^2 g^i(f_0(x_i), v_i, \partial^2 f^i(x_i, v_i))}{\partial v_i}\right)\n   10. \(\theta \leftarrow G(\theta, g)\) \(\triangleright\) Update surrogate parameters
11. end while
12. return \(N\)

Details

How to balance the loss parameters?

\(\Rightarrow\) Use \(\lambda_0\) (most important principal components)

\[c = 1 + \alpha c + \beta \eta^2, \quad \lambda_0 = \frac{1}{c}, \quad \lambda_1 = \frac{\alpha c}{c}, \quad \lambda_2 = \frac{\beta \eta^2}{c}\]

where e.g., \(\alpha = 1, \beta = 2\eta^2/c^2\).

How to deal with pathwise (derivative) payoff discontinuities?

\(\Rightarrow\) Use smoothing, e.g., sigmoidal smoothing.

References:


Future Directions

• More complicated models? (E.g., Heston? \(\Rightarrow\) requires variance reduction)
• PCA using Krylov subspace iteration solver
• Alternatives to PCA capturing non-linearities? (e.g., Kernel PCA, Autoencoder)
• Even higher-order differential data?