Matrix exponential for machine learning

Pavel Andreev
Petr Mokrov
Nadezhda Alsahanova
Sonya Ivolgina
Alexander Kagan

Skoltech

Problem statement

- **What?** explore the method of calculating matrix exponential for ML problems
- **Why?** obtain the same quality with reduced computing resources
- **Hypothesis:** it is not necessary to compute matrix exponential with machine precision
- **Application:** Machine Learning
- **How to measure quality:** Computation time vs loss in quality in ML tasks (NLL and accuracy) + number of matrix products as a measure of overall approach scalability

Application of matrix exponents in ML

- Invertible transformation for Generative flows with easy-to-compute Jacobian determinant [1]

- Nonlinearity in Neural Networks with universal approximation property (for single layer) and ability to extrapolate periodic and polynomial dependencies [2]

- Matrix Exponential Learning - approach to solving convex problems defined over sets of positive-definite matrices (with or without trace constraints)

[1] Xiao C., Liu L. Generative Flows with Matrix Exponential //International Conference on Machine Learning. – PMLR, 2020. – C. 10452-10461.
[2] Fischbacher T. et al. Intelligent Matrix Exponentiation //arXiv preprint arXiv:2008.03936. – 2020.

Methods to compute matrix exponential

1. Scaling and squaring

To control the roundoff error difficulties and computing costs of methods, we will use fundamental property:

$$e^A = (e^{A/m})^m.$$

Idea: choose the smallest *m = 2*^{n}, for which $\|A\|/m \leqq 1$

With this restriction, e$^{A/m}$ can be satisfactorily computed by either Taylor or Pade approximants. However, in practice due to limited number of matmuls, we restrict n to be less than N - k, where N is the total number of matmuls, and k is the number matmuls in Taylor/Pade aproximations

2. Pade approximation

The (m, m) Pade approximation to e^A is defined by:

$$r_m(A) = \left[p_m(-A)\right]^{-1} p_m(A),$$

where

$$p_m(x) = \sum_{j=0}^{m} \frac{(2m-j)!m!}{(2m)!(m-j)!} \frac{x^j}{j!}$$

We obtain, that $r_m(A) = e^A + \mathcal{O}(A^{2m+1})$

In practice, the evaluation of $p_m(A)$, $p_m(-A)$ is carried out trying to minimize the number of matrix products k (e.g. 2 for m = 3 and 3 for m = 5). Hence, the computational complexity is $O(kn^3)$

Note that although theoretical computational complexity of pade approximation might not differ from Taylor-based methods, in practice it performs slower due to parallelization issues of matrix inversion

3. Optimized Taylor Polynomial Approximation

Goal: given *k* < 7, calculate truncated Taylor polynomial of highest possible degree *n* with *k* matrix multiplications:

$$T_n(A) = \sum_{i=0}^{n} \frac{A^i}{i!} = e^A + \mathcal{O}(A^{n+1}).$$

Idea: Use Horner-like iterative scheme with unknown coefficients which can be found explicitly for given *k*:

$$\begin{aligned}
A_2 &= A^2, \quad A_3 = A_2 A, \quad A_6 = A_3^2,\\
B_1 &= a_{0,1} I + a_{1,1} A + a_{2,1} A_2 + a_{3,1} A_3,\\
B_2 &= b_{0,1} I + b_{1,1} A + b_{2,1} A_2 + b_{3,1} A_3 + b_{6,1} A_6,\\
B_3 &= b_{0,2} I + b_{1,2} A + b_{2,2} A_2 + b_{3,2} A_3 + b_{6,2} A_6,\\
B_4 &= b_{0,3} I + b_{1,3} A + b_{2,3} A_2 + b_{3,3} A_3 + b_{6,3} A_6,\\
B_5 &= b_{0,4} I + b_{1,4} A + b_{2,4} A_2 + b_{3,4} A_3 + b_{6,4} A_6,\\
A_9 &= B_1 B_5 + B_4,\\
T_{18}(A) &= B_2 + (B_3 + A_9) A_9.
\end{aligned}$$

Example: for k=5, maximal known n is 18

$$\begin{aligned}
A_2 &= A^2,\\
A_3 &= A_2 A,\\
B_1 &= a_{0,1} I + a_{1,1} A + a_{2,1} A_2 + a_{3,1} A_3,\\
B_2 &= a_{0,2} I + a_{1,2} A + a_{2,2} A_2 + a_{3,2} A_3,\\
B_3 &= a_{0,3} I + a_{1,3} A + a_{2,3} A_2 + a_{3,3} A_3,\\
B_4 &= a_{0,4} I + a_{1,4} A + a_{2,4} A_2 + a_{3,4} A_3,\\
A_6 &= B_3 + B_4^2\\
T_{12}(A) &= B_1 + (B_2 + A_6) A_6.
\end{aligned}$$

Example: for k=4, maximal known n is 12

Skoltech

4. Baseline methods

Baseline of M-layer (non-linearity) based on:

$$\exp(M) = \lim_{n\to\infty}(I + \frac{M}{n})^n$$
$$\exp(M) \approx (I + \frac{M}{2^k})^{2^k}$$

The computational complexity is *O(kn³)*, where *n* is the matrix size.

Baseline for generative flows: compute Taylor series of order k using Horner's rule and scaling and squaring

$$T_k(\boldsymbol{W}) = \sum_{i=0}^{k} \frac{\boldsymbol{W}^i}{i!} \qquad \boldsymbol{e}^{\boldsymbol{W}} = (\boldsymbol{e}^{\boldsymbol{W}/2^s})^{2^s}$$

The computational complexity is $O((s+k-1)n^3)$, where s is a degree of 2 in scaling and squaring. In our experiments we fix the approximation degree k and the number of matmuls N = s + k - 1, so s = N - k + 1

Experiments

Generative flows (training)

Method	Number of matmuls	NLL on test set
Baseline	Unlimited (~10)	3.32
Baseline (3)	< 7	Failed to converge
Pade* (5/5)	< 7	3.32
Optimized Taylor (8)	< 7	3.32

*the time for computing inverse is roughly set to the time of 1 matmul, although in real world it is less practical

CIFAR10 samples

Experiments

Generative flows (test time)

Method	Number of matmuls	test NLL	Time (whole test set)
Baseline	Unlimited (~10)	3.32	123s
Baseline (3)	< 5	8.24	88s
Pade (3/3)	< 5	3.34	110s
Optimized Taylor (8)	< 5	3.32	91s

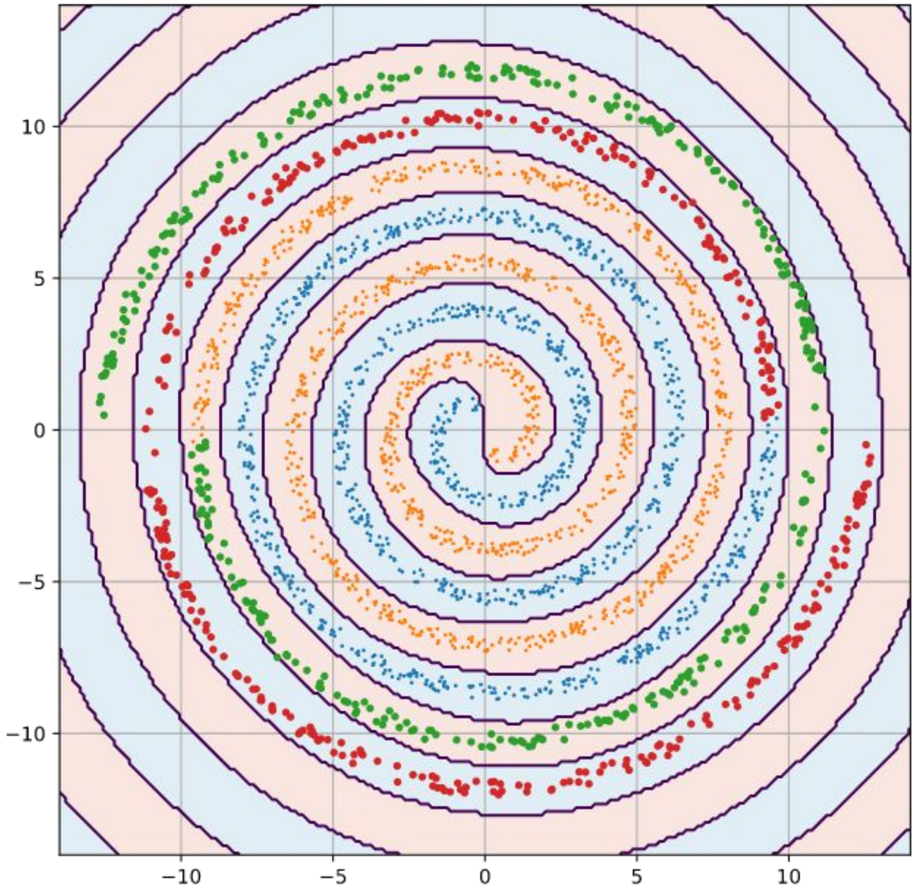
CIFAR10 samples

Skoltech

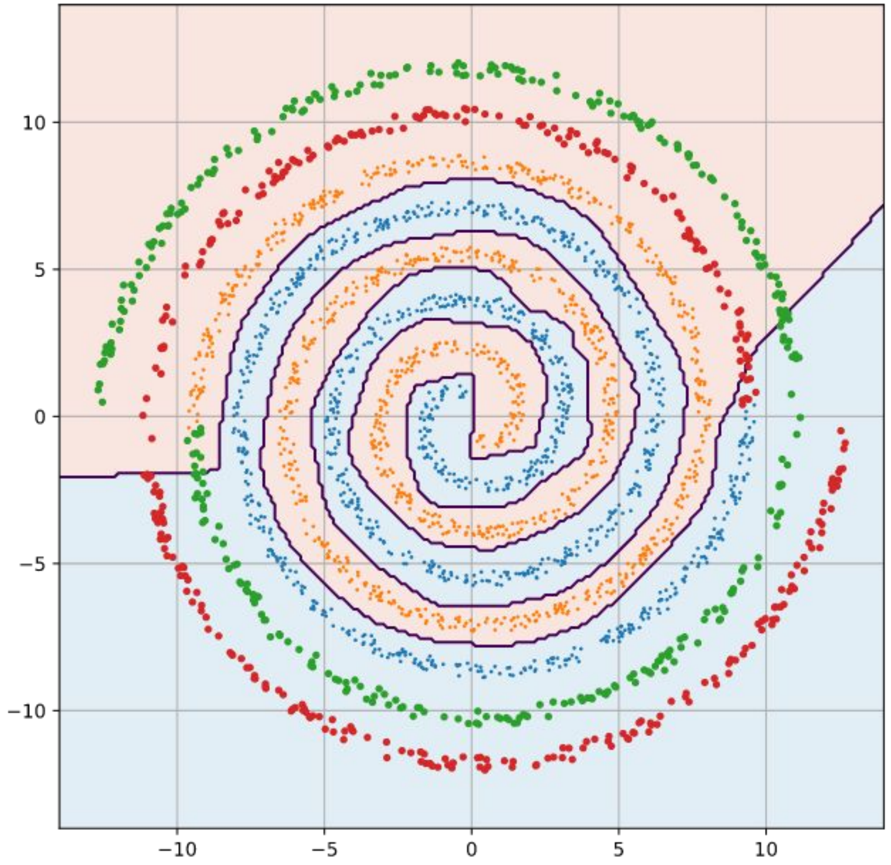
Experiments

Comparison Mlayer and ReLU on swiss roll dataset

NN with Mlayer nonlinearity.
1231 parameters



NN with standard ReLU nonlinearities
4417 parameters



blue - class 1, train
orange - class 2, train
red - class 1, test
green - class 2, test

Skoltech

Experiments

Mlayer with different matrix exponential implementations on swiss roll dataset

Method	Number of matmuls	Comp. time (per call)	Train acc.	Test acc.
Pade(3/3)	≤ 6	1.16 ms	1.	1.0
Pade(5/5)	≤ 6	1.4 ms	1.	1.0
Opt.Taylor (6)	≤ 6	1.4 ms	1.	1.0
baseline(6)	≤ 6	0.65 ms	1.	1.0

Conclusion

- We observed better expressiveness of NN with MLayer compared to standard ReLU networks on data with periodic structure, this expressiveness appeared to marginally depend on computational procedure of matrix exponential
- Using Optimized Taylor approximation of order 8 we achieved the same quality for the density estimation on CIFAR10 with generative flows as in the case of using unlimited number of multiplications, but 25% faster.
- If machine precision accuracy is not mandatory in exponent calculation for ML applications, it is wise to constrain the number of matrix multiplications to speed up the performance
- The swiss roll problem turned out to be too easy for the Mlayer algorithm. So more challenging datasets should be also considered in the future in order to better determine the influence of particular computational methods of matrix exponential

Skoltech

thx.

Skoltech