NATIONAL RESEARCH UNIVERSITY
HIGHER SCHOOL OF ECONOMICS

Faculty of Computer Science
Bachelor's Programme "Applied Mathematics and Informatics"

UDC 519.61

Research Project Report on the Topic:

Matrix Exponential for Parametrizing Orthogonal Convolutions

(interim, the first stage)

Submitted by the Student:

group #БПМИ213, 3rd year of study — Gaponov Alexander Leonidovich

Approved by the Project Supervisor:

Rakhuba Maxim Vladimirovich
Research Fellow
Faculty of Computer Science, HSE University

Moscow 2024

Abstract

Convolutional neural networks with a 1-Lipschitz constraint under the l_2 norm achieve state of the art results in provable adversarial robustness, stable training and gradient interpretability. Sahil Singla and Soheil Feizi propose an architecture of a 1-Lipschitz CNN called LipConvNet that uses Skew Orthogonal Convolution (SOC) layers instead of standard convolution layers. LipConvNet shows high results in standard and robust accuracies and does not suffer from gradient vanishing. Useful properties of SOC are obtained because of the orthogonality of it's Jacobian. To achieve orthogonality, the following property is used: the exponential of the skew-symmetric matrix is orthogonal, so the problem of fast matrix exponential computation is crucial in SOC layer. Taylor series was used in the original layer implementation but it is possible to achieve better exponential approximation using Krylov subspace based approaches: Arnoldi and Lanzcos algorithms that takes less computing time providing same quality of approximation.

•Computing methodologies ∼ Machine learning ∼ Machine learning algorithms ∼ Regularization

Contents

1 Introduction

1.1 1-Lipschitz neural networks

The Lipschitz constant of the function f $\mathbb{R}^n \to \mathbb{R}^m$ is the upper bound on how the norm of the output vector can change when the norm of the input vector is changed. By default we use Euclidean norm. The property of being 1-Lipschitz (1-Lip) for convolutional neural networks leads to adversarial robustness [2], improved generalization bounds, better gradient interpretability [1], preventing from gradient explosion. To construct 1-Lip neural networks it's convenient to use the Lipschitz composition property, which can be formulated as follows:

$$Lip(f \circ g) \leq Lip(f)Lip(g)$$

It gives us a way to achieve 1-Lip constraint for the whole neural network by bounding each layer's Lipschitz constant by 1. It can be proved that 1-Lip layers can only reduce the norm of the gradient during back-propagation. As it was mentioned above this property saves the model from gradient explosion, but gradient vanishing becomes the problem [1].

1.2 Convolution orthogonality

To overcome the gradient vanishing problem were introduced Gradient Norm Preserving (GNP) architectures based on the property of orthogonal Jacobian layers to preserve gradient norm. For CNNs were proposed special activation function called GroupSort and special convolution parametrisation: Block Convolutional Orthogonal Parametrisation (BCOP) [5]. The proposed layers help to overcome gradient vanishing problems but suffer from accuracy decrease and slow training.

1.3 Skew Orthogonal Convolution

Introduced by Sahil Singla and Soheil Feizi Skew Orthogonal Convolution (SOC) layer [10] reduce training time and improves standard and certified robust accuracies. The key property used in SOC layers is that exponential of skew-symmetric matrix is orthogonal so authors construct convolution filter with orthogonal Jacobian using first k terms of Taylor series:

$$S_k(J) = \sum_{i=0}^{k-1} \frac{J^i}{i!} \tag{1}$$

where J is a skew-symmetric matrix. Authors provide following upper bound on the error

$$\|exp(J) - S_k(J)\| \leq \frac{\|J\|^k}{k!} \tag{2}$$

Also a method for generating skew-symmetric matrices was proposed: let M be some filter, then L=M-conv_transpose(M) is skew-symmetric, where conv_transpose(L) is the transposed convolution operator of tensor L. Transposed convolution is a combination of flips and transposes along some dimensions and it's formally explained in [10]. Proposed layer outperforms BCOP in time performance and both standard and robust accuracy. The problem of this method is low computational efficiency because in every SOC layer we apply k-1 standard convolution operators, where k is the number of terms in the Taylor series approximation. Using approximation methods based on Krylov subspace helps to decrease the number of convolution operators without losing accuracy. So better learning time and efficiency of inference can be achieved.

1.4 Krylov subspace approximation

The convolution operator is a linear function, so it can be interpreted as matrix J multiplied by vector x, where vector x is vectorised input tensor and J is the Jacobian of the convolution. So in our problem we compute the following expression:

$$exp(J)x = (\sum_{i=0}^{k-1} \frac{J^i}{i!})x \tag{3}$$

The matrix J is $O(N^2)$ size where N is the size of input tensor, so we won't store J explicitly. Note that computing J^i is a difficult and redundant task, since we multiply matrix by the vector so we can rewrite expression 3 using only the matrix multiply vector operations:

$$exp(J)x = \sum_{i=0}^{k-1} \frac{J^i x}{i!}$$

Now let's note that vectors x, Jx, ... $J^{k-1}x$ naturally belong to some Krylov subspace. We recall the definition of Krylov subspace:

$$\mathcal{K}_k(J, x) = span\{x, Jx, ...J^{k-1}x\}$$

So our task can be formulated as finding coefficients in some convenient basis of $\mathcal{K}$. Mentioned in the article [8] method of exp(J)x approximation using the Arnoldi algorithm can help us to

reduce the number of matmul operations without losing in relative error. In this method, we first generate an orthogonal basis of the Krylov space by modifying the Gram-Schmidt process. Then we approximate exp(J)x using the following formula:

$$exp(J)x \approx \beta V_m exp(H_m) e_1$$

2 Related work

Advantages of provable 1-Lipschitz networks such as r provable adversarial robustness bounds and stable training are explained in the following article by Q. Li et al. [5] Proposed expressive parametrisation of orthogonal convolutions (BCOP) allows us to train deep 1-Lip CNNs and solves the problem of gradient vanishing. It achieves high results, but is not as accurate and efficient as SOC.

S. Singla and S. Feizi in their research [10, 9] introduce architecture of 1-Lip CNN called LipConvNet based on SOC, special activations (MaxMin and HouseHolder), Projection Pooling Layers and fast SOC gradient computation. They achieve state of the art accuracy, robustness and computation efficiency.

E. Gallopoulos and Y.Saad [8, 3, 4] in their articles present a theoretical analysis of some Krylov subspace approximations of the matrix exponential operation exp(A)v and provides a priori and a posteriori error estimates. Algorithms presented in the articles can be useful in SOC as a more optimal way of matrix exponential computation.

Various ways of matrix exponential computation are presented in the following article [7] by C. Moler and C. V. Loan

Bounding Lipchitz constant of a neural network may lead to standard accuracy decrease as a price for robustness. Trade-off between robust accuracy and standard accuracy was described in D. Tsipras et al. article[2]. This phenomena was observed in our research as well.

3 Checking orthogonality

Examining matrix orthogonality is crucial in our task as we want to obtain fair Lipschitz constant 1 that allows to estimate robust accuracy of our network. Note that robust accuracy is essentially requires network to be Lipschitz as it is not defined otherwise. So we compare methods of exponential approximation with the same level of orthogonality determined using tests proposed below. However measuring robust accuracy is not the only problem. When you train network without (or with big) Lipschitz constant you probably achieve better accuracy as network is less bounded so networks with different Lipschitz constraints become incomparable.

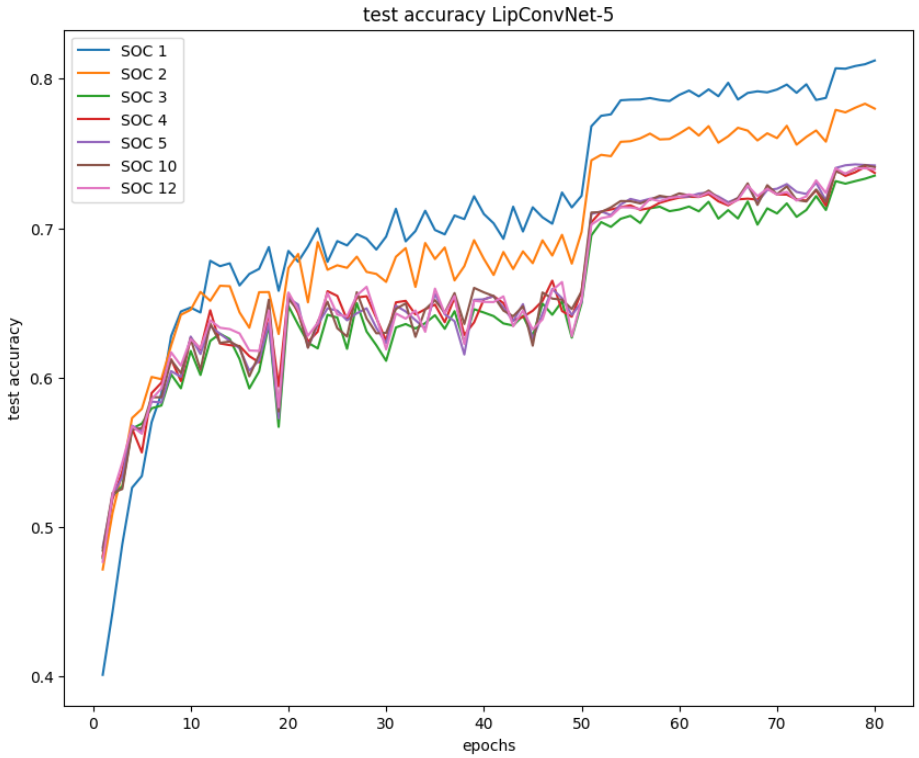

Figure 3.1: Graph shows test error of SOC based LipConvNet-5 on CIFAR-10 depending on a different number of terms in the Taylor series (k). Notice that small k gives better accuracy as network is less constrained however we have no theoretical guaranties on robustness. k=1 is equal to standard convolution layer. k=12 gives best possible exponential approximation in float16. Jump on the 50th epoch is decreasing learning rate in 10 times

As it was mentioned above we can not afford matrix multiply matrix operations because a Jacobian of convolution operator is of $O(n^2)$ size where n is an input tensor size so multiplication takes $O(n^3)$ operations while matrix multiply vector only $O(n^2)$. In addition we do not want to construct Jacobian explicitly as it is stored compactly in a tensor of $O(k^2 c_{in} c_{out})$ size, where k is a kernel size, c_{in} and c_{out} number of input and output channels respectively. To check orthogonality of learned filter we use 3 methods.

3.1 Comparison with ground-truth exponential

Let we have filter L and an input tensor x. We want to compute $y = exp^*(L, x)$ and compare result with $y_{true} = exp(L, x)$ where $exp^*(L, x)$ is some approximation of the ground-truth $exp(L, x)$ where $exp(L, x)$ is computed using enough terms of the Taylor series. Of course it is nothing but more accurate approximation of considered operator however in computer arithmetic we can assume it as the ground-truth answer. Results can be compared by computing relative error: $\frac{\|y_{true}-y\|}{\|y_{true}\|}$ This test has convenient form because many convergence theorems are presented as upper bounds for norm of the error e.g. 2 however test estimates orthogonality indirectly and not as obvious as the following tests.

Table 3.1: Table shows how many terms of the Taylor series required to achieve best possible naive approximation in computer arithmetic.

Datatype	Number of Terms
float16	12
float32	24
float64	60

3.2 Id-transform test

In this test we examine the orthogonality of Jacobian by checking if $JJ^Tx = x$ where J is Jacobian of a convolution. Note that in general case to transpose Jacobian of convolution we have to make some non-trivial operations with it's kernel but in our case we use skew-symmetry of J and construct $J^T = -J$. It is easy to see that this method requires only matrix multiply vector operations. For x we take tensors that appear during training and compute relative error:

$$\frac{\|JJ^Tx - x\|}{\|x\|} \tag{4}$$

3.3 Hatchinson test

Hutchinson estimator [6] $H_m(A) = \frac{1}{m}\sum_{i=1}^{m} g_i^T A g_i$ is approximation of $tr(A)$ where m is number of terms and g_i are taken from $\mathcal{N}$(0,I). It can be useful to compute Frobenius norm of matrix that can'b be formed explicitly. In our case we compute:

$$\|JJ^T-I\|^2 = tr((JJ^T-I)^T(JJ^T-I)) \approx \frac{1}{m}\sum_{i=1}^{m} g_i^T((JJ^T-I)^T(JJ^T-I))g_i = \frac{1}{m}\sum_{i=1}^{m}\|(JJ^T-I))g_i\|^2 \tag{5}$$

Received in 5 norm approximation must be close to 0 so it means that J is close to orthogonal. Also note that 5 is close to 4. The main difference is in the origin of tensors x and g.

3.4 Testing orthogonality in naive approach

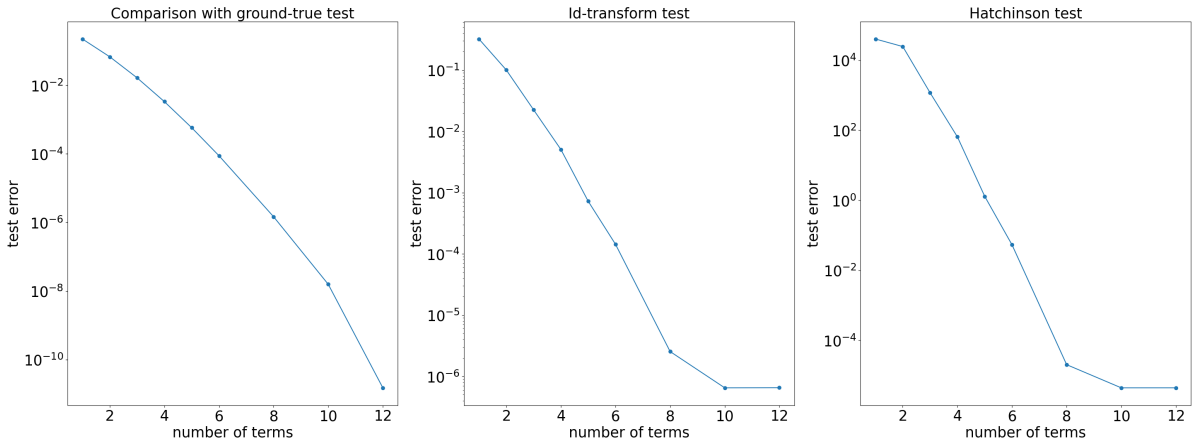

Figure 3.2: Figures show how errors decrease until stabilising for naive exponential approximation using the Taylor series in float32. Results are shown for convolution operator with $c_{in} = c_{out} = 32$, $kernel\_size = 3$ and $filter\_size = 64$. SOC authors [10] propose to use 12 terms of the Taylor series in float16 to achieve best possible approximation. As we see using 12 terms is reasonable choice for float32 as well.

4 Krylov based algorithms of matrix exponential approximation

4.1 Method describtion

We consider two approaches that are known as Arnoldi and Lanzcos algorithms of computing $exp(A)x$. Both algorithms firstly build convenient basis of Krylov subspace: orthogonal basis in Arnoldi and bi-orthogonal in Lanzcos. Than use this basis to approximate matrix exponential multiply vector by the following formula:

$$exp(J)x \approx \beta V_m exp(H_m)e_1 \tag{6}$$

Matrix H_m is Hessenberg or tridiagonal matrix that comes from Arnoldi or Lanczos algorithms respectively. More detailed algorithms described in Y. Saad paper [8]. Firstly we implemented proposed algorithms and convinced their effectiveness for explicit matrix exponential multiply vector task.

Table 4.1: Table shows time and relative error of exp(A)v approximation using naive with 12 terms and Arnoldi with basis size of 6 $A \in \mathbb{R}^{N \times N}$. Ground-true exponential approximation was taken from scipy

	Naive		Arnoldi	
	Time	Error	Time	Error
N=100	**127 µs**	1.65e-10	538 µs	8.93e-11
N=1000	**693 µs**	6.61e-10	912 µs	**3.32e-15**
N=10000	261 ms	1.62e-10	**142 ms**	**3.84e-14**

Note that original algorithms work with explicit matrix multiplications however in CNNs we deal with tensors and convolutions so we modified both algorithms to make them applicable in computer vision tasks. Also we developed improvements that significantly speed up forward and backward passes.

4.2 Time analyze of Arnoldi and Lanzcos algorithms

Time complexity of Arnoldi algorithm is $O(nm^2 + mnc)$ where m is Krylov space basis size, n is input tensor size and c is a constant of convolution operator that depends on it's parameters e.g. number of filters and kernel size. On the opponent side we have $O(mnc)$ complexity for naive and Lanzcos approaches, where m is a number of the Taylor series terms and basis size respectively.

The main advantage of Krylov-based algorithms is reduced number of convolutions comparing to naive approach so when we increase c by taking more filters and using big-size kernels we achieve more obvious outperforming in time.

As you see in 6 both Arnoldi and Lanzcos algorithms require exponential computation but fortunately H_m is $O(m^2)$ size so it's almost not contribute in total computation time as requires $O(m^3)$ operations, m in our experiments is not bigger than 10. In case of big m proposed more effective way of computing this step [8].

It'll be shown bellow that Arnoldi algorithm requires fewer terms then naive approach and as a result fewer convolution operations. However orthogonalisation has quadratic dependence on m and affects the complexity of the algorithm the most 4.3 4.1. In addition orthogonalisation significantly increase computation graph size which leads to memory and time costs as need Hessenberg matrix to be cloned in every iteration. The valuable improvement is blocking the gradient flow when building Hessenberg matrix and computing it's exponential. So H_m and $exp(H_m)$ are interpreted as constants in the computational graph that improve memory usage and time of backward and forward 4.2.

Table 4.2: Comparison of original Arnoldi algorithm to modified version. Time measured on forward and backward. Our modification achieves almost twice better performance on backward without losing in quality of learning.

		Original		Modified	
Kernel size	Number of filters	Forward time	Backward time	Forward time	Backward time
3	32	0.022	0.104	0.021	**0.063**
3	64	0.044	0.201	0.043	**0.104**
3	128	0.084	0.399	0.086	**0.209**
5	32	0.024	0.112	0.025	**0.057**
5	64	0.047	0.213	0.046	**0.113**
5	128	0.090	0.432	0.089	**0.237**
7	32	0.026	0.132	0.025	**0.077**
7	64	0.113	0.293	0.112	**0.192**
7	128	0.327	0.584	0.325	**0.546**

Table 4.3: Table shows how many percents of all time different parts of Arnoldi algorithm take while doing forward path. Results are averaged for multiple runs for convolutions with various parameters. input tensor channel size and m are fixed. $m = 8$. In this table and further we work with modified version of Arnoldi. Even after boosting orthogonalisation part it takes significant proportion of computation time

		Module	
Kernel size	Number of filters	Convolutions	Orthogonalisation
3	32	24%	63%
3	64	33%	57%
3	128	34%	56%
5	32	35%	55%
5	64	35%	55%
5	128	35%	55%
7	32	40%	50%
7	64	72%	23%
7	128	82%	16%

Choosing m equal to 8 is not accident. As it wll be shown below it is enough to achieve appropriate level of orthogonalisation. Notice that we can use different basis sizes on train and evaluation modes. It's important to compute exponential accurately only during inference so basis size on train can be reduced twice or even more. However taking not enough terms on train can lead to problems with learning as the model on train is significantly different from the inference model. So in this case we observe quality growth on train but not on test. As it was mentioned basis size is the most important parameter of Arnoldi algorithm so it is important to understand how orthogonalisation time depends on it 4.1.

Lanzcos algorithm does not need basis to be orthogonal however require two convolution operations per iteration. This problem is complemented with a big computation constant while performing $O(n)$ operations to construct bi-orthogonal basis. Lanzcos algorithm can be modified in the same way as Arnoldi: computing coefficients for tridiagonal matrix with no gradient mode.

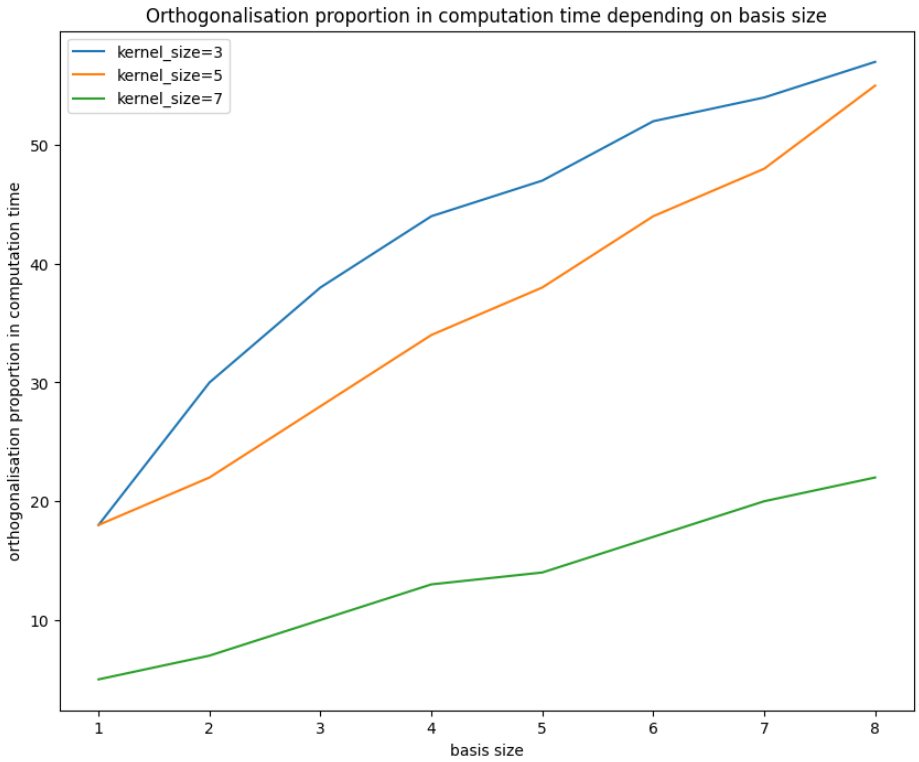

Figure 4.1: Figure shows how much time (%) orthogonalisation takes comparing to the whole algorithm depending on the basis size. Growth is close to linear. For small k orthogonalisation makes a minor contribution to the total time

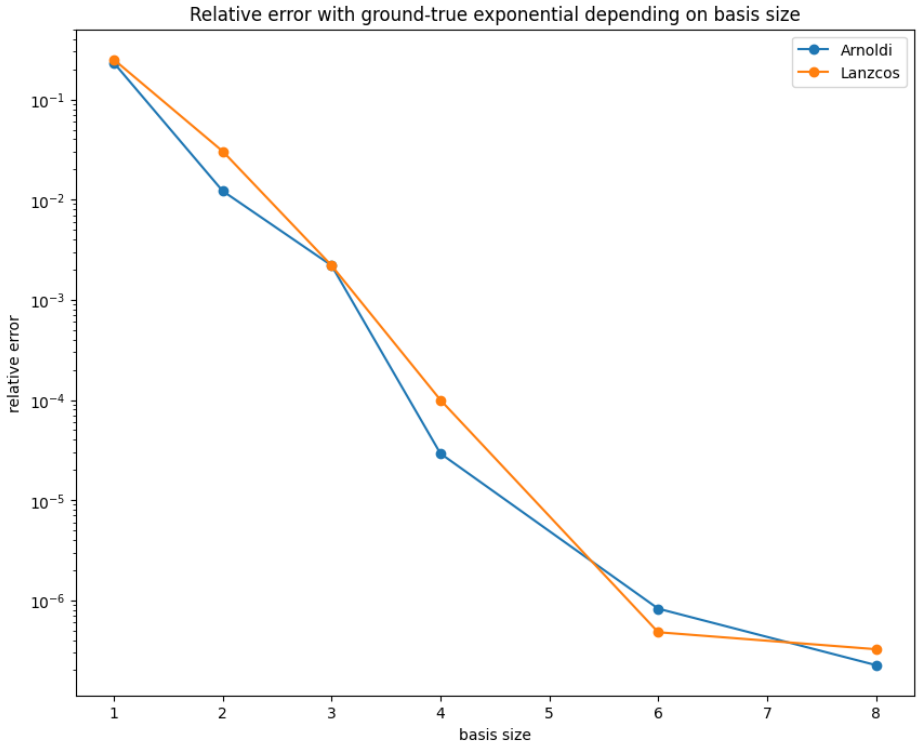

Figure 4.2: Arnoldi and Lanzcos methods achieve same relative error with true exponential depending on basis size. That's the reason why we compare their performance using same basis size.

Comparison of Krylov-based methods are presented below 4.4. Both methods are potentially useful so next section introduce results of their comparison to naive approach.

Note that in 4.4 we use same number of basis vectors in Arnoldi and Lanzcos methods. It's reasonable as both methods have close orthogonality tests results having same basis size. More obviously it can be seen in the figure 4.2.

Table 4.4: Table shows Arnoldi and Lanzcos approaches comparison. Arnoldi has significantly faster forward path in most of cases. However Lanzcos outperforms Arnoldi in backward efficiency in cases with big basis size.

			Modified Arnoldi		Modified Lanzcos	
Basis size	Kernel size	Number of filters	Forward time	Backward time	Forward time	Backward time
4	3	32	0.017	0.015	0.018	0.012
4	3	64	**0.038**	0.026	0.047	0.022
4	3	128	**0.083**	0.051	0.102	0.047
4	5	32	**0.027**	0.016	0.038	0.017
4	5	64	**0.046**	0.029	0.059	0.027
4	5	128	**0.102**	0.064	0.127	0.061
4	7	32	0.030	0.026	0.033	0.026
4	7	64	0.244	0.071	0.289	0.069
4	7	128	**0.804**	0.217	0.948	0.215
8	3	32	0.027	0.055	0.029	0.046
8	3	64	**0.066**	0.104	0.087	**0.073**
8	3	128	**0.138**	0.206	0.157	**0.126**
8	5	32	**0.038**	0.058	0.052	**0.041**
8	5	64	**0.076**	0.113	0.096	**0.072**
8	5	128	**0.160**	0.237	0.188	**0.154**
8	7	32	0.050	0.093	0.056	**0.058**
8	7	64	**0.310**	0.192	0.384	**0.153**
8	7	128	0.985	0.391	0.982	**0.312**

5 Comparison naive approach and Krylov-based

5.1 Criteria

There are two important quality metrics for us: accuracy and robust accuracy. If standard accuracy is familiar to everyone, robust accuracy needs some words to be introduced. Robust accuracy shows if our predictions still correct if we change the input using some budget ϵ. It resembles white box adversarial attack but in our case we don't train input image as we have upper bound on how logits can change if we change the input. Upper bound is achieved by the Lipschitz property. Assume correct class logit is maximal so it must be bigger enough than the second maximal logit to stay maximal after input perturbations. If it is true we assume object being robustly classified. More detailed it is presented in the SOC article [10]. As it was mentioned above there no much sense in comparing approximations of different quality as they become incomparable in proposed set of metrics. That is why in our experiments we compare approximation with same orthogonality tests error. As we witnessed during experiments 5.1 different methods of approximation with same orthogonality rate achieve very close classification quality.

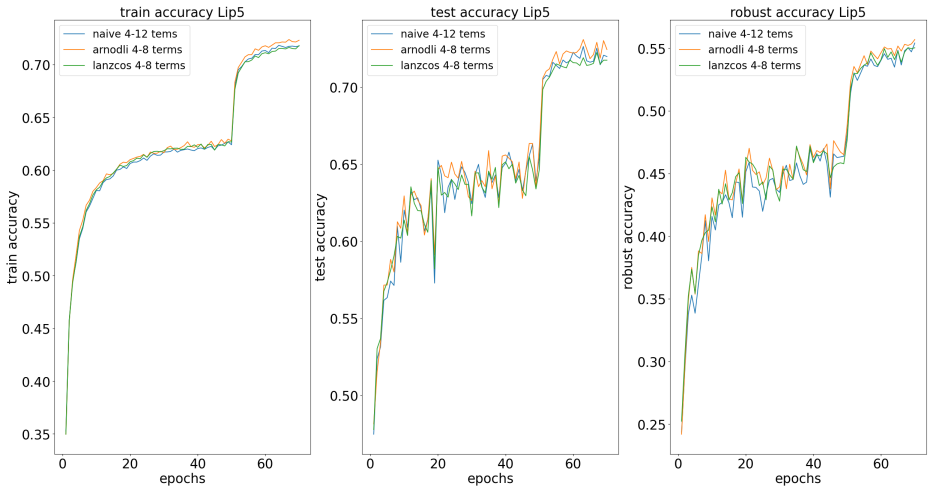

Figure 5.1: Different methods with same level of orthogonality achieve same quality.

5.2 Single layer comparison

In this subsection we separate one SOC layer and compare 3 proposed methods of exponential approximation. Parameters are chosen to achieve same orthogonality in all three approaches.

Table 5.1: Table shows time on forward comparison of matrix approximation methods. As you see Krylov-based methods outperform naive approach in some cases where kernel size is big. Basis size column has format x (y) where x is Krylov basis size and y number of Taylor terms. x and y are chosen to achieve same orthogonality level

Basis size	Kernel size	Number of filters	Naive forward time	Arnoldi forward time	Lanzcos forward time
4 (4)	3	64	**0.062**	0.072	0.081
4 (4)	3	128	**0.141**	0.162	0.182
4 (4)	5	64	**0.084**	0.094	0.105
4 (4)	5	128	**0.187**	0.211	0.233
4 (4)	7	64	0.543	**0.535**	0.580
4 (4)	7	128	**0.354**	0.386	0.401
8 (12)	3	64	**0.068**	0.100	0.107
8 (12)	3	128	**0.164**	0.289	0.301
8 (12)	5	64	**0.096**	0.122	0.134
8 (12)	5	128	**0.220**	0.329	0.343
8 (12)	7	64	0.637	**0.598**	0.675
8 (12)	7	128	0.973	**0.969**	0.982

5.3 LipConvNet

LipConvNet [10] is a state of the art architecture of robust CNN that by default consists of SOC and MaxMin activation layers. Authors propose to use 6 Taylor series terms on train and 12 on test, convolutions has kernel size equal to 3 but we also experiment with 5 and 7. As you can see in the following table, results from single layer experiments are transferred to LipConNet-5 learning. Basis size and number of terms are chosen to achieve same orthogonality error.

Table 5.2: Comparison of naive, Arnoldi and Lanzcos approaches. For LipConvNet with big kernel size Arnoldi outperforms other methods.

	Naive		Arnoldi		Lanzcos	
Kernel size	Epoch train time	Eval time	Epoch train time	Eval time	Epoch train time	Eval time
3	**51.3**	**5.4**	53.1	7.8	58.3	8.4
5	**97.0**	**11.2**	97.6	12.7	126.8	18.1
7	150.6	**15.1**	**129.7**	15.2	154.3	28.2

6 Further research directions

Krylov-based methods can outperform naive approach in some special cases however big computation constant of Arnoldi and Lanzcos methods cause lot's of difficulties. There are some possible ways of further modifications:

As it was mentioned Arnoldi algorithm build orthogonal basis that takes a lot of computation time. We developed modification that does not make orthogonalisation of k last vectors in basis. This heuristic makes formula 6 incorrect as it's vitally require V_m to be orthogonal. However it can be useful while training when we don't need fair Lipschitz constant.

We discussed that decreasing the number of terms (basis size) we can achieve better standard accuracy. It can be possible to achieve fair Lipschitz constant and high standard accuracy by slowly increasing number of terms. We carried out experiments with schedulers that change basis size. We observed that when basis size increase standard accuracy sharply decrease and then stabilize but lower than before increasing basis size.

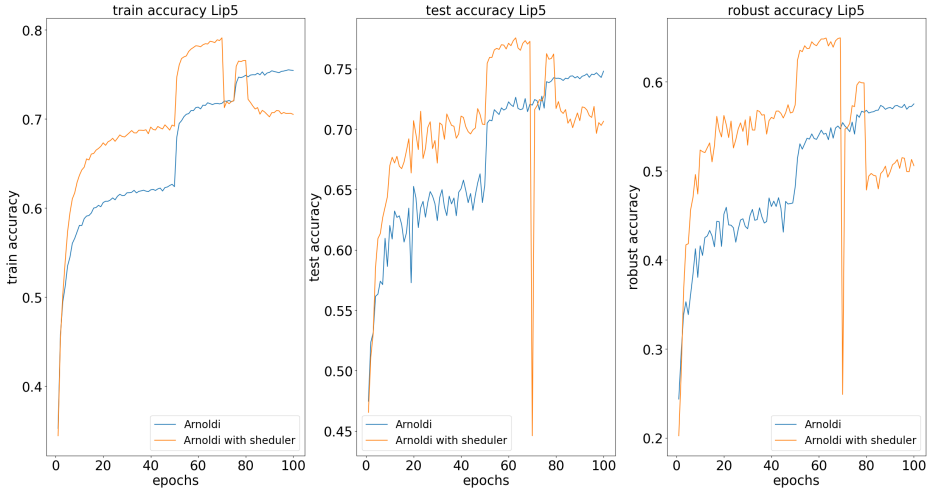

Figure 6.1: Comparison of Arnoldi (8 terms on test) and Arnoldi with scheduler that change basis size from 2 to 5

7 Conclusion

Robustness is one of the most crucial features of neural networks as it provides reliability and makes usage of CNNs appropriate in medicine, self-driving transport and other spheres with high error cost. Our research gives experimental evidences on the process of robust learning. We developed convenient and efficient tests of convolution Jacobian orthogonality estimating and propose a process of correct approximations comparison.

Long computation time is a common problem for the majority of 1-Lipschitz CNN architectures so we significantly impacted on time efficiency by providing modifications of Arnoldi and Lanzcos algorithm. Various Krylov-based methods were compared to find optimal that outperforms original SOC implementation on more than 13%. We also introduced promising ideas for further research including experiments with basis size shedulers.

References

[1] Cem Anil, James Lucas, and Roger Grosse. "Sorting Out Lipschitz Function Approximation". In: *ICML*. 2018.

[2] Tsipras D., Santurkar S., Engstrom L., Turner A., and Madry A. "Robustness May Be at Odds with Accuracy". In: *ICLR*. 2018.

[3] Y. Saad E. Gallopoulos. "Efficient Solution Of Parabolic Equations By Krylov Approximation Methods". In: *SIAM Journal on Scientific and Statistical Computing* (2000).

[4] Y. Saad E. Gallopoulos. "On the parallel solution of parabolic equations". In: (1989).

[5] Qiyang Li, Saminul Haque, Cem Anil, James Lucas, Roger Grosse, and Jörn-Henrik Jacobsen. "Preventing Gradient Attenuation in Lipschitz Constrained Convolutional Networks". In: *Conference on Neural Information Processing Systems*. 2019.

[6] R. A. Meyer and C. Musco. "Hutch++: Optimal Stochastic Trace Estimation". In: (2022).

[7] Cleve Moler and Charles Van Loans. "Nineteen dubious ways to compute the exponential of a matrix". In: *SIAM Review* 20.4 (1978).

[8] Y. Saad. "Analysis of some Krylov subspace approximations to the matrix exponential operator". In: *SIAM Journal on Numerical Analysis* (1992).

[9] Sahil Singla and Soheil Feizi. "Improved techniques for deterministic l2 robustness". In: *NeurIPS*. 2022.

[10] Sahil Singla and Soheil Feizi. "Skew Orthogonal Convolutions". In: *ICML*. 2021.