EC4

Methodological Guidelines for 1-Lipschitz neural networks

Document reference number for ANR

contact@confiance-ai.fr | www.confiance.ai

Document reference: XXX

Contributors

	Name	Organisation	Role
Responsible for the deliverable	Corentin FRIEDRICH	IRT Saint Exupéry	Lead EC4.32
Scientific responsible	Yannick PRUDENT	IRT Saint Exupéry	Main contributor EC4.32
Co-authors	Thibaut BOISSIN	IRT Saint Exupéry	Lead EC4.11

Document Control

Revision	Date	Commentary	Author
v1	31/12/2023	Initial version, end of batch 3	Corentin FRIEDRICH
v2	01/06/2024	Corrections after comments	Corentin FRIEDRICH

Contents

A. Introduction and abstract

A.1. General introduction to trustworthy AI challenges

Trustworthiness in AI within critical systems (systems that can directly or indirectly affect human life and moral entities) is essential for its widespread adoption (by the industry, the decision makers, the general public, etc.) and poses the following significant challenges.

- First, how to design AI models, so that, by construction, they satisfy trustworthy properties (accuracy, robustness…).
- Secondly, how to characterize these AI models, for example to understand and explain their behavior and their adequacy to the operational domain.
- Then, how to implement and embed those AI models on hardware, by making them fit for the target without losing their trustworthy properties.
- Another question is, what methods of data engineering to apply in order to, among other topics, manage important volumes of data and adapt to the evolution of the operational domain.
- At system level, what verification and certification processes to consider specifically for AI-based systems.
- Finally, a federation of all these matters is necessary to build an end-to-end methodological approach, supported by a consistent engineering environment compatible with industrial practices.

These are the challenges, among others, that the Confiance.ai program addresses.

A.2. Abstract of this guideline document

Ensuring the robustness of neural networks is crucial to guarantee that the system behaves as expected, even when inputs are slightly perturbed. 1-Lipschitz neural networks provide a method for achieving robust solutions by design, particularly in the face of adversarial attacks. These methodological guidelines are intended to offer insights into how to build, train, and assess such networks.

Scientific challenges addressed
The scientific challenges addressed by 1-Lipschitz neural networks are:

- Ensuring the theoretical robustness of neural networks to misuse and noise
- Ensuring robustness to variations from the environment

1-Lipschitz networks are robust to L_2-norm perturbation, especially adversarial attacks, and provide certified robustness thanks to their inherent properties.

Trustworthiness attributes addressed
1-Lipschitz networks address the *Robustness* trustworthiness attribute.

Target audience

This document is meant to be useful for the following audience:

- Data scientists, AI algorithm engineers, software engineers, or any users interested in training a robust neural network for their specific use case.
- System engineers and other users looking to use and integrate an already trained 1-Lipschitz network in a broader system.
- System IV&V engineers, safety engineers, or any users interested in evaluating the robustness performance of a 1-Lipschitz network and especially its certified robustness.

Tasks supported

1-Lipschitz networks have been used to solve classification, semantic segmentation and object detection tasks for vision use cases. Other tasks and types of data have not been studied yet.

Limitations and perspectives

The user must be aware that robustness may come at a cost to performance. In this document, we denote this phenomenon as the trade-off performance-robustness or accuracy-robustness. This trade-off is not specific to 1-Lipschitz networks and is encountered with any other robustness methods such as adversarial training or randomized smoothing.
Training 1-Lipschitz neural networks requires more memory and is more time-consuming than training standard networks. This is a limitation, especially for GPU with low VRAM memory. However, after training, a 1-Lipschitz neural network can be converted to a standard network with no more memory or computation overload.

Robust 1-Lipschitz networks are still an active research topic. We can expect innovation with new network architectures to overcome current limitations. New tasks can also be explored.

Document organization

This document is structured as follows:

- Chapter B provides context and describes the robust training method
- Chapter C emphasizes on how to build and train 1-Lipschitz networks to foster robustness
- Chapter D concentrates on the training and evaluation of 1-Lipschitz networks for classification tasks.
- Chapter E focuses on the training and evaluation of 1-Lipschitz networks for semantic segmentation tasks.
- Chapter F addresses the training and evaluation of 1-Lipschitz networks for object detection tasks.
- The conclusion highlights limitations and explores future perspectives.

B. Description of the method

B.1. Motivation

Ensuring the robustness of deep neural networks is primordial, particularly in the face of adversarial attacks. Numerous approaches have been explored to strengthen neural networks against vulnerabilities, ranging from advanced training techniques to incorporating adversarial training samples, to randomized smoothing methods. However, one promising avenue is 1-Lipschitz neural networks, which offers a means of enforcing robustness but also provides strong theoretical guarantees.

Traditional methods often involve augmenting training data with adversarial samples (like adversarial training) or incorporating regularization strategies. While these approaches provide some level of defense, it is necessary to have new strategies that offer robustness as an inherent property. 1-Lipschitz neural networks have gained prominence. The Lipschitz continuity property ensures that small changes in input space result in proportionally small changes in output space. In the context of deep neural networks, enforcing a 1-Lipschitz constraint limits the sensitivity of the model to input perturbations, thereby mitigating the impact of adversarial attacks.

The theoretical guarantees associated with 1-Lipschitz networks add an extra layer of motivation. These guarantees stem from the mathematical definition of Lipschitz continuity, providing a formal assurance of the model's stability. By incorporating such guarantees, practitioners gain confidence in the network's ability to resist to adversarial manipulations, fostering trust in the model's performance.

In conclusion, 1-Lipschitz neural networks represent a grounded approach to enforce the robustness of deep neural networks. Thanks to the inherent Lipschitz property, theoretical guarantees make 1-Lipschitz networks a leading choice for practitioners seeking robust AI solutions.

B.2. Prerequisites for using the method

Building and training 1-Lipschitz neural networks requires careful consideration of several key prerequisites. First, a clear understanding of Lipschitz continuity and its implications is essential. Users should be familiar with the mathematical concepts that underlie Lipschitz constraints, as these constraints guide the network's behavior during training. Additionally, choosing an appropriate architecture that facilitates the enforcement of Lipschitz continuity is crucial. This may involve selecting activation functions, weight initialization methods, and network architectures that inherently lend themselves to Lipschitz constraints. Moreover, a judicious choice of loss functions is necessary to strike a balance between performance and robustness.

Hopefully, those prerequisites can be seamlessly addressed through the use of the DEEL-

LIP library. This specialized library serves as a powerful tool for practitioners by hiding the complexities underlying Lipschitz enforcement. The mathematical theory is abstracted, allowing users to focus on the aspects of their models rather than the technical details of Lipschitz constraints. The library simplifies the selection of appropriate layers and activation functions, alleviating the process of building robust neural networks. Moreover, a selection of losses and metrics is offered to enhance robustness in the training phase. Practitioners, whatever their level of expertise, benefit from a user-friendly interface that eases the building and training of 1-Lipschitz neural networks.

Finally, the **prerequisites** are limited to:

- a labelled dataset, like in any other training stage,
- well-defined requirements or expectations in terms of performance and robustness.

B.3. Maturity of the method

While 1-Lipschitz networks represent an attractive research approach in the scientific community, it is important to acknowledge that they are still in the developing stages and have not reached full maturity. In comparison to more established methods such as adversarial training and randomized smoothing, the field of 1-Lipschitz networks is actively evolving and remains a research topic with ongoing investigations and developments.
However, the provided DEEL-LIP library described below helps practitioners with an easy-to-use tool that provides current state-of-the-art solutions to build and train 1-Lipschitz neural networks. This tool is widely maintained and is continuously evolving with new features.

B.4. Related resources in the Confiance.ai program

1-Lipschitz networks have been studied in the Confiance.ai program and some reports present the theory and results on use cases. Here is the list of documents for further information about 1-Lipschitz networks:

- EC4 Robust Embeddable Deep Learning by Design (Chapter D) for theory and preliminary results
- Methods and Evaluation Tools for Robust AI (Chapter G) for results on Confiance.ai use cases

B.5. The DEEL-LIP library as a tool

Building and training 1-Lipschitz neural networks is a non-trivial challenge and conventional frameworks, like TensorFlow or PyTorch, often lack specialized tools for this task. The DEEL-LIP library emerges as an important solution filling the gap between the demand for robustness in neural networks and the limitations of traditional frameworks.

The DEEL-LIP library is an open-source library tailored for users seeking to leverage their models' robustness through 1-Lipschitz neural networks. Offering a suite of 1-Lipschitz layers and custom losses, DEEL-LIP provides a toolkit for building and training neural networks with Lipschitz continuity. It is carefully maintained and regularly

updated with new layers and loss functions. Built on Tensorflow and Keras, it seamlessly integrates into existing training workflows, ensuring compatibility and ease of adoption. DEEL-LIP has a user-friendly API, designed to simplify the complexities associated with enforcing Lipschitz constraints.

The DEEL-TORCHLIP library offers a similar API specific for PyTorch integration. Custom layers and losses can be readily incorporated in existing PyTorch training pipelines to train robust 1-Lipschitz networks.

In this document, we focus on the methodological guidelines for 1-Lipschitz networks, without any prior assumptions regarding specific tools like DEEL-LIP. The following chapters present only the theoretical information and methodology.

C. Build and train a 1-Lipschitz network

This is the core chapter that details the methodology to build, train and evaluate a 1-Lipschitz network.

Procedure in a nutshell
Building and training a robust 1-Lipschitz neural network can be summarized as follows:

1. Clearly define the expectations in terms of performance and robustness. This is use-case specific and requires a good knowledge of the system requirements.
2. (Optional) Build and train a standard neural network. This network serves as baseline reference, before training 1-Lipschitz networks. Performance and empirical robustness can be evaluated. Hyper-parameters like data augmentation or optimizer can be tuned in this early step.
3. Build a 1-Lipschitz network by stacking 1-Lipschitz layers.
4. Train it with a cosine similarity loss and evaluate its performance, empirical robustness and certified robustness.
5. Train it again (from scratch) with a loss tailored for robustness and evaluate the performance and robustness metrics.
6. Repeat step 5 with different losses and hyper-parameters until the desired requirements (performance and robustness) are reached.

The diagram in Fig. C.1 outlines this procedure: it gives the overall vision in how to build, train and evaluate a 1-Lipschitz neural network.

C.1. Build a 1-Lipschitz network

Building 1-Lipschitz neural networks consists in stacking 1-Lipschitz layers. This chapter will first explore in Sec. C.1.1 the architectural elements for 1-Lipschitz layers and which types are suitable for Lipschitz-constrained networks. We then provide tips on how to build 1-Lipschitz networks in Sec. C.1.2.

C.1.1 Architectural elements

C.1.1.1 Affine layers

Enforcing Lipschitz constant in affine layers
Various methods for estimating the Lipschitz constant exist, offering several degrees of precision. These methods include weight clipping and normalization using the Frobenius norm. However, these weight-constraining techniques often result in networks whose Lipschitz constants are significantly lower than one, leading to vanishing gradient. The preferred solution is spectral normalization, where the weights are normalized based on their spectral norm. In simple terms, this involves dividing the weights by the maximum singular value. It is even possible to enforce the weight matrix to be orthogonal, meaning that all singular values equal to 1. This special case is called Gradient Norm-Preserving (GNP) layers where the gradient norm is equal to 1 almost everywhere.

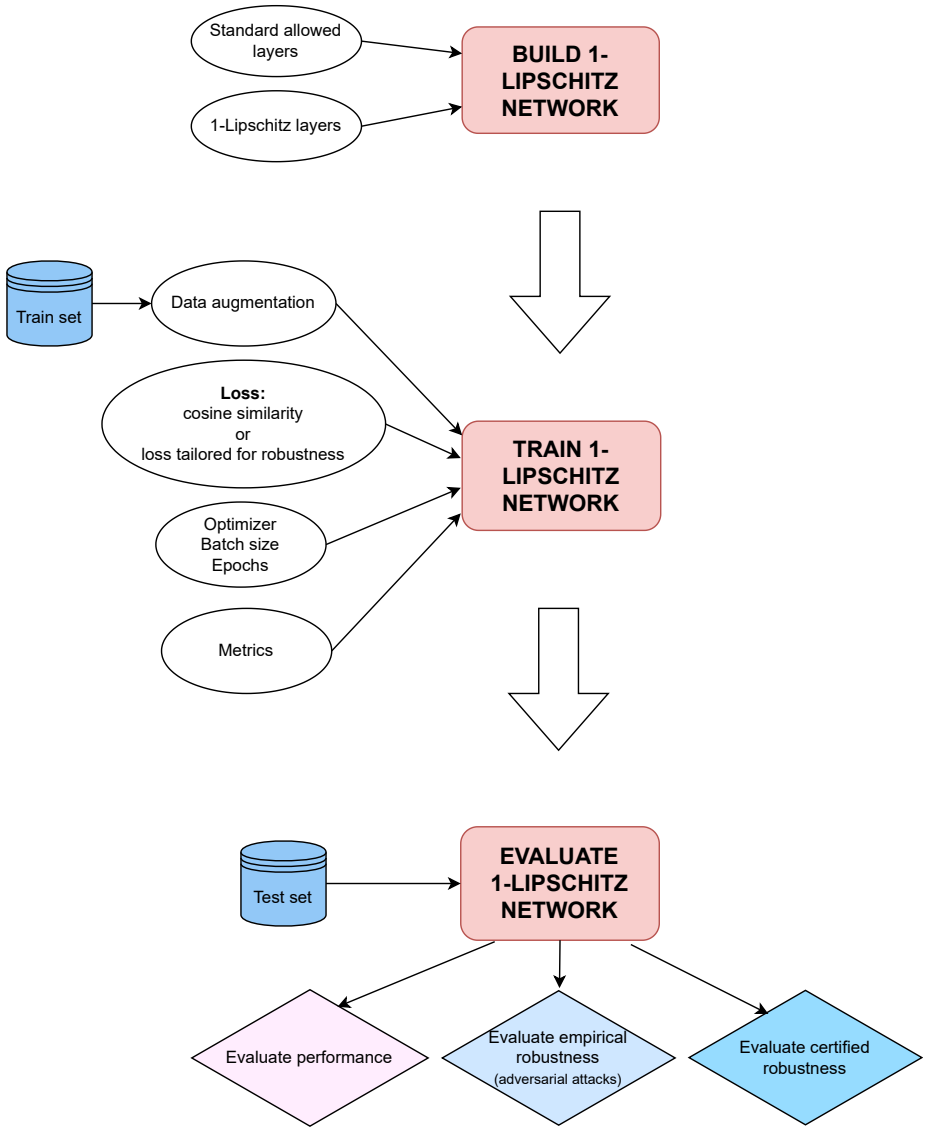

Figure C.1: Focus on how to build, train and evaluate a 1-Lipschitz neural network. Training is based on TensorFlow/Keras framework but some objects (layers, losses, metrics) must be suited for robust 1-Lipschitz networks and are available in DEEL-LIP.

The two most commonly used affine layers in deep neural networks are the fully-connected layer and the convolutional layer. In segmentation tasks, the transposed convolutional layer, sometimes referred to as 'deconvolution,' is also employed. This three affine layers can be normalized to be 1-Lipschitz or even orthogonal. The community provides many methods to parametrize a fully-connected or convolutional layer as orthogonal.

Tips for 1-Lipschitz affine layers

- The convolutional layers require padding that keeps the size of the image. Zero padding or circular padding must be used to ensure the 1-Lipschitz property.
- Initialize layers weights with orthogonal kernels.
- No need to add L1/L2 weight regularization. Weight decay is a common solution to regularize the weights at training and to avoid exploding gradients. However with 1-Lipschitz networks, the kernels are already constrained to be 1-Lipschitz and orthogonal. So weight regularization is useless.

C.1.1.2 Activations

Many common activation functions used in deep learning, such as *ReLU*, *Sigmoid*, *Tanh*, and *LeakyReLU* (when $|\alpha| < 1.0$), are 1-Lipschitz functions. This makes them suitable for use in 1-Lipschitz neural networks. However, it's important to note that these activations can have a very small gradient (even zero), which can lead to vanishing gradients. To mitigate this problem, we strongly recommend using specific activations, which are designed to be Gradient Norm-Preserving (GNP) and, therefore, maintain a healthy gradient norm throughout the network. You can find details on these specific activations in Anil et al. (2019) and in the previous Confiance.ai report.

Gradient norm-preserving activations

- *GroupSort*: the function sorts group of channels in the descending order. All the values are preserved, they are just sorted. We encourage to start with *GroupSort2* activation layers (i.e. *GroupSort* with groups of 2).
- *MaxMin*: this activation doubles the output size by concatenating $(ReLU(x), ReLU(-x))$,
- *FullSort*: GroupSort applied on the full channels
- *Householder*: the Householder activation layer, a generalization of *GroupSort2*.

Tips for 1-Lipschitz activation layers

- Using GNP activation helps maintaining gradient in the network. *GroupSort2* is preferred because of its efficiency.
- If using standard activations (e.g. ReLU, Sigmoid) instead of GNP activations, make sure the activation function is 1-Lipschitz. For instance, using ReLU layer, do not use a negative slope more than 1 and do not set a threshold different from zero, that yields discontinuity which is not 1-Lipschitz.

C.1.1.3 Pooling

Pooling layers are also widely present in computer vision networks for reducing spatial dimensions. We distinguish local pooling layers with a spatial pool size (usually 2×2 reduction) and global pooling where the spatial dimensions are reduced to a single output neuron. 1-Lipschitz pooling layers are available in DEEL-LIP and are presented below. These 1-Lipschitz pooling layers can replace standard pooling layers with no risk: the input and output shapes are the same, only the internal mechanism is slightly different.

Local pooling layers

- *ScaledL2NormPooling2D*: a pooling layer that preserves the L_2-norm (GNP layer).

For example, the output of a 2×2 pool is the L_2 norm of the vector formed by the 4 input values. We encourage to use this pooling layer.
- *ScaledAveragePooling2D*: a 1-Lipschitz average pooling layer (not GNP).
- Standard *MaxPooling2D*: this layer is already 1-Lipschitz by construction and can be used as is. Note that even if it is 1-Lipschitz, this layer is not GNP and gradient vanishing can occur.
- *InvertibleDownSampling*: a GNP pooling layer that reshapes the input. The output has smaller spatial sizes but more channels due to the reshape operation. For example, a 2×2 pool size produces an output with 4 times more channels than the input. This layer has the advantage to be GNP but does not reduce the number of neurons, unlike the other pooling layers presented here.

Global pooling layers
- *ScaledGlobalL2NormPooling2D*: a global pooling layer that preserves the L_2-norm (GNP layer). For each channel, the output is simply the L_2 norm of the input channel.
- *ScaledGlobalAveragePooling2D*: a 1-Lipschitz average pooling layer (not GNP).
- Standard *GlobalMaxPooling2D*: this layer is already 1-Lipschitz by construction and can be used as is, but it is not GNP and can introduce gradient vanishing.

Tips for 1-Lipschitz pooling layers
- While any 1-Lipschitz pooling layer is acceptable, we strongly recommend using GNP layers like *ScaledL2NormPooling2D*, *ScaledGlobalL2NormPooling2D* and *InvertibleDownSampling*.
- It's required for 1-Lipschitz local pooling layers to have the same values for pool size and stride to maintain the Lipschitz constant.

C.1.1.4 Other layers

Other layer types are frequently used in common architectures for vision: some of them are 1-Lipschitz by nature and can be safely used, others are not 1-Lipschitz and must be forbidden.
- Flatten layer ✅: this layer is GNP, as it simply flattens the input tensor. It can be safely used in 1-Lipschitz networks.
- Reshape layer ✅: this layer is GNP, as it simply reshapes the input tensor. It can be safely used in 1-Lipschitz networks.
- Batch normalization ❌: a scaling operation is applied, affecting the Lipschitz constant. This layer is not 1-Lipschitz.
- Dropout layer ❌: this layer is transparent at inference, but applies a scaling factor during training to compensate the values set to zero. The Lipschitz constant at training is then not controlled. We discourage the use of Dropout layers.
- Add layer ✅: mainly used in skip connections. This layer is 2-Lipschitz if both branches are 1-Lipschitz. Outputs must be manually divided by two to ensure 1-Lipschitz operation.
- Concatenate layer ✅: mainly used in skip connections. This layer is $\sqrt{2}$-Lipschitz if both branches are 1-Lipschitz. Outputs must be manually divided by $\sqrt{2}$ to ensure 1-Lipschitz operation and GNP.
- Patch cutting ✅: Some modern vision architectures cut the input image into

patches, apply a same function to all of them and concatenate the resulting outputs. If the applied function is 1-Lipschitz, the whole process "patch cutting + 1-Lipschitz operation + concatenation" is also 1-Lipschitz.
- Self-attention layer ❌: The self-attention module relies heavily on a product of two functions depending on x. This yields an x^2 term that has, by construction, no Lipschitz bound. Although Lipschitz-constrained vision transformers based on self-attention layers have been proposed in the recent literature, this type of architecture is out of our scope.

C.1.2 Build a Lipschitz network

Building a 1-Lipschitz network is simple as stacking 1-Lipschitz layers. The user must keep in mind that skip connections (with addition or concatenation) should be handled with care due to their Lipschitz constant different from 1.

Export 1-Lipschitz networks to standard networks
After training a 1-Lipschitz network, the affine 1-Lipschitz layers can be converted to their standard counterparts. Indeed, the 1-Lipschitz constraint is applied during training but final weights are just standard weights. The exported model is more efficient than the 1-Lipschitz network since the Lipschitz constraints are no more computed. It is encouraged to save and use the exported model when evaluating and using the model at inference. Note that, an exported model must not be fine-tuned. Otherwise, the affine layers will lose their 1-Lipschitz property. If the model should be fine-tuned, it is important to save it as is without exporting to a standard network.

Tips for building 1-Lipschitz models
- Losses tailored for robustness require logits as model outputs. 1-Lipschitz models should not end with sigmoid/softmax activation.
- 1-Lipschitz networks should not be initialized with pretrained standard models. They must be trained from scratch; for now, no pretrained 1-Lipschitz backbones actually exist. It is encouraged to initialize affine layers with orthogonal weights.
- Due to the spectral normalization process, the number of parameters of a 1-Lipschitz model is about twice compared to the equivalent model.
- We recall that it is important to vanilla export your 1-Lipschitz trained model for inference efficiency.

C.2. Train a 1-Lipschitz network

In the training phase of a 1-Lipschitz neural network, the weights of affine layers are consistently constrained to maintain a Lipschitz constant of 1 during each weight update. Two approaches are used to enforce 1-Lipschitz property at training:
- constraining weights. After updating weights with an optimizer step, the weights are then constrained before the next weight update.
- parametrizing weights. Unconstrained weights are parametrized to 1-Lipschitz weights with a differentiable operation Φ, i.e $W_{\text{1-Lip}} = \Phi(W_{\text{orig}})$. When back-propagating gradients, the original weights W_{orig} are updated through the differentiable parametrization.

Although constraining weights is easier and faster, the training converges slowly and can be stuck. Weight parametrization is preferred (the DEEL-LIP library makes parametrization abstracted from the user to ensure a more seamless experience).

Training a 1-Lipschitz neural network using is very similar to conventional training. The primary distinction lies in the choice of the loss function, which is the crucial element in balancing performance and robustness (Béthune et al., 2022). Sec. C.2.1 will describe different losses tailored for robustness (available in the DEEL-LIP library). In the following sections of this chapter, we will discuss about the other hyper-parameters required during training: optimizer, batch size, metrics, etc.

C.2.1 Choosing a loss for robustness

In theory, a 1-Lipschitz model can be trained with any loss, e.g. cross-entropy, cosine similarity, hinge loss. These losses aim at enhancing accuracy but may not tend to improve robustness, since they do not necessarily increase the margin between the output logits. Indeed, a larger gap between logits corresponds to a stronger robustness. It is very important to understand that robustness and accuracy are opposite targets and a higher robustness comes with a drop in performance.
Below are losses that allow to tune this accuracy-robustness trade-off: training with these losses yields more robust classifiers. Especially, Serrurier et al. (2021) introduced a new loss based on optimal transport that encourages the class distributions to be far from each other.

C.2.1.1 Losses based on optimal transport

Two losses defined in Serrurier et al. (2021) where robustness is promoted using a Wasserstein distance term:

- ***HKR*** loss (hinge-Kantorovich-Rubinstein). It consists of two terms: the Kantorovich-Rubinstein term that maximizes the Wasserstein distance (optimal transportation problem), and the hinge loss term acting as a regularization. This loss is specific for binary classification problems.
- ***MulticlassHKR*** loss. It corresponds to the *HKR* loss for multi-class problem.
- ***MulticlassSoftHKR*** loss: an optimized version of *MulticlassHKR*, suitable for dataset with a large number of classes.

Tips for tuning HKR hyper-parameters
Two hyper-parameters should be set for *HKR* losses:

- the minimal margin to enforce in the hinge term.
- the regularization parameter α_{KR} that controls the importance of the hinge regularization term, compared to the KR term.

We first encourage the user to set an infinite value for α_{KR} (i.e. only hinge term) and find the margin that gives the best accuracy. A too small margin means that only misclassified elements are penalized, without giving importance to correctly classified inputs. In contrast, a too high `min_margin` will penalize almost all elements without focusing on misclassified inputs. In both cases, training doesn't produce an accurate network. Figure C.2 shows the effect of the margin on accuracy after training with $\alpha_{KR} = +\infty$.

Note that, depending on the scaling of your inputs, the margin must be scaled accordingly. For example, a smaller margin should be chosen with inputs in [0, 1] (as a rule of thumb, margin between 0.01 and 5), compared to inputs in [0, 255] (e.g. margin between 0.1 and 50).

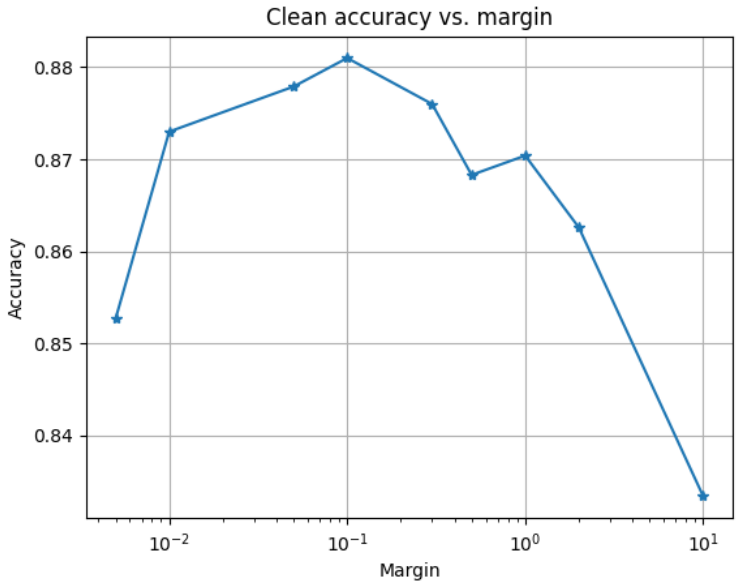

Figure C.2: Clean test accuracy vs. margin for models trained with *HingeMargin* and different `min_margin`. Trained on CIFAR-10.

As a second step, α_{KR} can be adjusted to find a trade-off between accuracy and robustness. Reducing α_{KR} yields an accuracy drop but a more robust network. Figure C.3 plots the accuracy vs. robustness for different values of α_{KR} for models trained on CIFAR-10.

C.2.1.2 Standard losses tailored for robustness

Common loss functions used in deep learning are typically not tailored for robustness. Nevertheless, they can often be slightly adjusted to manage the trade-off between accuracy and robustness using a hyper-parameter that is hidden in the standard version.

Margin-based losses

The ***HingeMargin*** loss corresponds to the hinge loss for binary classification with the margin as hyper-parameter. Setting a large margin enforces robustness by having large logit values. It is identical to the *HKR* loss with α_{KR} set to infinity.

The ***MulticlassHinge*** loss is the extension for multi-class classification tasks. Since hinge loss in multi-class case has not a unique definition, it is possible to define other hinge versions based on Keras and PyTorch own implementations with the additional margin hyper-parameter: ***CategoricalHinge*** for Keras version and ***MultiMargin*** for PyTorch implementation.

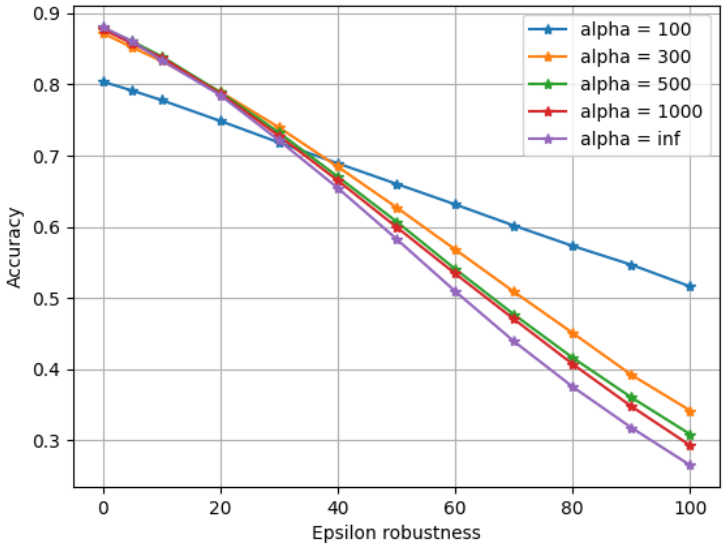

Figure C.3: Robust test accuracy vs. robustness with L2PGD attacks for models trained with *HKR* and different α_{KR}. Trained on CIFAR-10.

Cross-entropy losses
In common deep learning frameworks, the cross-entropy loss has no hyper-parameter. The softmax operation involved in cross-entropy actually has a hidden parameter called temperature[1], denoted τ here. The temperature can be adjusted to manage the trade-off between performance and robustness. For large τ, the loss yields accurate networks but not robust. In contrast, for small (positive) τ, the network will be less accurate but more robust.

C.2.1.3 Cosine similarity loss

While cosine similarity is not a standard classification loss in computer vision, it has good properties that make it interesting when used with 1-Lipschitz networks. Cosine similarity computes a loss only based on the angle between two vectors, regardless of the vectors' magnitudes. Consequently, the loss term is independent of the scale of the output logits, a characteristic relevant for 1-Lipschitz networks. However, unlike the aforementioned losses, there is no hidden hyper-parameter for balancing accuracy and robustness. This loss can be employed as an initial attempt to train a 1-Lipschitz network without the need to fine-tune loss hyper-parameters. It allows to focus the training on other hyper-parameters, such as network architecture, data augmentation, etc. It is important to note that the resulting model may exhibit less robustness.

C.2.1.4 No need for L_2 regularization (or weight decay)

A usual approach to reduce the Lipschitz constant is to add L_2 weight regularization as an auxiliary loss function. This technique is also called weight decay. The idea

[1]See details on softmax temperature in https://en.wikipedia.org/wiki/Softmax_function

is to enforce weights to be small, thus avoiding exploding weights. For 1-Lipschitz training, this regularization is not useful, since the weights are constrained by the spectral normalization. There is then no risk of large weights.

C.2.1.5 Tips for loss configuration

It is good practice to initially train a 1-Lipschitz network using the cosine similarity loss to gain insights into the model's performance. During training, focus can be directed towards other variables such as network architecture, data augmentation, and more. Additionally, training an equivalent standard network, featuring the same architecture but using standard layers along with batch normalization/dropout layers, provides a valuable basis for comparing the performance between standard and 1-Lipschitz models.

The second step involves training the 1-Lipschitz network with a loss function tailored for robustness. This step allows to achieve the desired trade-off between performance and robustness. It is essential to conduct multiple trainings with varying loss hyper-parameters to identify the ideal trade-off that the user expects.

C.2.2 Choosing an optimizer

The choice of the optimizer is not specific to 1-Lipschitz training and there is no restriction on this choice. We do not recommend a specific one and the user is free to choose and try different optimization algorithms. In our experiments, we mostly used *Adam* that is easier to tune, compared to other optimizers like *SGD*.

The same advice can be given for the learning rate and the number of epochs. Like any standard training, these parameters should be chosen depending on the data set, the model and the loss. A learning rate scheduler can be used to improve convergence rate.

C.2.3 Importance of batch size

Losses based on optimal transport compute conditional expectations for each class. It is thus important to have batches containing enough elements from all classes: for a large number of classes, it is advised to choose a large batch size to reduce the probability to have a random batch with a class not being represented. For example, for CIFAR-10 data set containing 10 classes, batches with 128 or 256 images are a proper choice. For data sets like ImageNet containing 1000 classes, an even larger batch size must be adopted.

The analysis above is valid for correctly balanced dataset. However, unbalanced datasets are a crucial issue when using *HKR* losses, and must be tackled differently.

C.3. Evaluate a 1-Lipschitz network

Like any standard network, 1-Lipschitz networks can be assessed using standard metrics, such as accuracy for classification tasks, mIoU for segmentation or mAP for object detection. Robustness can also be measured with specific metrics. Adding metrics during training can help to get a quick preview of the evolution of robustness. The

following metrics are specific to 1-Lipschitz networks and cannot be used for standard networks (because of the Lipschitz constant is not constrained):

- ***KR*** metric gives an estimation of the Wasserstein distance for a binary classification problem. It is equivalent to the *HKR* loss where the hinge term is zero ($\alpha_{KR} = 0$). The higher the KR term, the more robust the network is. ***MulticlassKR*** metric is the multi-class counterpart of the *KR* metric.
- ***BinaryProvableAvgRobustness*** and ***CategoricalProvableAvgRobustness*** for binary and multi-class certified robustness metric. It corresponds to the average of certificates (see previous report).
- ***BinaryProvableRobustAccuracy*** and ***CategoricalProvableRobustAccuracy***. For a given perturbation radius, it computes the certified robust accuracy, i.e. the percentage of elements for which the robustness certificate is above the given perturbation radius.

When training is done, it is possible to measure empirical robustness of the trained model by running adversarial attacks. As this operation is costly, it cannot be done during training.

D. Guidelines for classification tasks

This chapter focuses on image classification tasks.

D.1. Architectures for classification

CNN architectures like VGG, ResNet, and MobileNet are prevalent for 1-Lipschitz networks. Since training 1-Lipschitz networks can be resource-intensive, smaller versions are recommended, depending on available hardware resources. In the Confiance.ai program, only VGG and ResNet architectures have been tested.

D.2. Evaluate the empirical robustness

Empirical robustness of a classification model is tested through adversarial attacks, aiming for minimal L_2 norm perturbations. Various perturbations like L_∞ attacks, noise, brightness, blur, and contrast changes can be applied, though 1-Lipschitz networks specifically resist L_2 perturbations. A common metric is *robust accuracy within a given attack budget*, quantifying the percentage of correctly classified images under a specified perturbation budget ε. Robust accuracy can be visualized as a function of ε, aiding in balancing accuracy and robustness.

D.3. Evaluate the certified robustness

Certified robustness provides a lower bound for adversarial perturbations: certificates guarantee no adversarial perturbation smaller than the bound can alter the prediction. Certified robustness is only available for 1-Lipschitz networks and is computation-free (based on outputs of the model).
Certified robustness is assessed through *robust certified accuracy* at a fixed budget ε, representing the percentage of images with certificates exceeding ε. This accuracy is a lower bound compared to empirical robustness.

E. Guidelines for semantic segmentation tasks

Semantic segmentation tasks can be seen as a classification task for each pixel in the input image. The user should be acquainted with the previous section about classification tasks, before reading this chapter.

E.1. Architectures suited for 1-Lipschitz segmentation networks

Various architectures differ mainly in their encoder and decoder definitions. Decoders may use upsampling, transposed convolutions, or *atrous* convolutions. Some include post-processing (e.g., DeepLab with Conditional Random Field). However, only some architectures, like Fully Convolutional Networks and U-Net, can be created with the 1-Lipschitz layers. Others, such as DeepLab or PSPNet, are not supported yet.

E.2. Train 1-Lipschitz segmentation networks

Segmentation networks are usually trained with the cross-entropy loss, since it is considered as a pixelwise classification problem. Due to a higher risk of imbalanced data (more background than foreground pixels for instance), other losses such as the focal loss can be used. For 1-Lipschitz training, the tempered cross-entropy variant *TauCategoricalCrossentropy* can be used. Like classification tasks, it is advised to train and compare with different losses suited for robustness, such as *MulticlassHKR*.
Like in classification tasks, the hyper-parameters of the loss must be tuned: the temperature for *TauCategoricalCrossentropy*, the margin and the regularization factor for *MulticlassHKR*. The trade-off between performance (usually mIoU in segmentation) and robustness is handled by varying these hyper-parameters. However, the values to set are not the same as in the classification. The loss hyper-parameters must be set accordingly, e.g. higher temperatures for tempered cross-entropy loss or smaller margins for hinge-based losses.

E.3. Evaluation of empirical robustness with adversarial attacks

Empirical robustness can be assessed with adversarial attacks, like in classification tasks. However, the methods for classification are not fully appropriate for segmentation and specific adversarial attacks were recently proposed. There is no Python package combining attacks for segmentation in TensorFlow but the library *adversarial-library*[1] in PyTorch provides various implementations.

In classification, the success of an attack is well defined: the attack is successful when the adversarial image is misclassified. However, the definition of a “successful attack”

[1] https://github.com/jeromerony/adversarial-library

is not trivial for segmentation. We can consider a successful attack when all pixels are misclassified but it usually requires a large perturbation which is not meaningful. Instead of measuring success, a more common practice is to evaluate the empirical robustness with a given budget: for a given perturbation budget ε, what is the mIoU obtained for the adversarial images? A robust model is expected to have a higher robust mIoU than for a standard network. There is no standard choice for ε in the literature. Like in classification, this choice depends on the use case and the requirements.

E.4. No certified robustness for segmentation tasks

In classification tasks, a robustness guarantee is easily computable since the definition of success is established (i.e. misclassifying the image). However, because of a lack of definition of success, certifying robustness for segmentation tasks is not well defined. What does a robustness radius mean in semantic segmentation? Do all pixels must be misclassified? Should mIoU be considered as a metric for certification? Recent works on *randomized smoothing* tried to define certified segmentation robustness but there is no consensus. Moreover, randomized smoothing only gives probabilistic guarantees.

To our knowledge, there is no literature on certified robustness for segmentation with 1-Lipschitz networks and this is a current line of research in the DEEL project.

F. Guidelines for object detection tasks

Object detection task involves identifying and localizing multiple objects within an image. The primary goal is to detect the presence of objects belonging to specific classes and provide precise bounding box coordinates around them. In recent years, deep learning approaches have revolutionized object detection by leveraging convolutional neural networks (CNNs) to automatically learn hierarchical features from data.

F.1. Architectures suited for 1-Lipschitz object detectors

The Fully Convolutional One-Stage (FCOS) model, introduced in Tian et al. (2019), is a state-of-the-art approach in object detection within computer vision. Unlike traditional two-stage detectors, FCOS operates as a single-stage, end-to-end convolutional neural network. Its fully convolutional nature allows it to predict bounding box coordinates, objectness scores, and category labels simultaneously for each position on a feature map. This eliminates the need for anchor boxes and complex region proposal networks. By leveraging the advantages of a fully convolutional architecture and centerness prediction, FCOS achieves good results in terms of accuracy and speed, making it a notable candidate for 1-Lipschitz object detection.
Other one-stage architectures have not been tested in the Confiance.ai program. But as long as the layers have 1-Lipschitz counterparts, it is possible to build and train different architectures. Note that, unlike classification heads, regression heads cannot be 1-Lipschitz since it is necessary to represent any function to locate the bounding boxes. These heads should not be 1-Lipschitz: the layers are kept as is.

F.2. Train 1-Lipschitz object detection networks

The classification loss, initially a focal loss, must be adapted to a focal loss with temperature. This is similar to other losses presented in chapters about classification and segmentation: the temperature hyper-parameter allows to tune the trade-off between performance and robustness for the classification part.
The regression loss term, used to predict the bounding box location and size, remains untouched. Remember that regression heads are not 1-Lipschitz.

F.3. Evaluation of empirical robustness with adversarial attacks

Like classification and segmentation tasks, empirical robustness can be assessed with adversarial attacks: vanishing objects, fabricating objects, changing classes of predicted boxes, etc. With a fixed perturbation budget ε, it is possible to measure the drop in mAP (the standard metric used in object detection) due to the attack.
As in segmentation evaluation, there is no definition of certified robustness for object detection task.

G. Conclusion

The significance of adversarial robustness in ensuring trustworthy AI cannot be underestimated: the vulnerability to adversarial attacks poses a serious threat. While several methods have been developed to address adversarial robustness, it is evident that 1-Lipschitz neural networks offer a competitive solution with substantial benefits. The DEEL-LIP library is a valuable tool to assist users in building and training 1-Lipschitz networks.
This document serves as a comprehensive guide, summarizing essential guidelines, tips, and advice for the correct construction and training of 1-Lipschitz networks, empowering practitioners to enhance the robustness and reliability of AI systems in the face of potential adversarial challenges. The emphasis is placed on three tasks investigated within the Confiance.ai program: image classification, semantic segmentation, and object detection.

Despite their efficacy in enhancing adversarial robustness, 1-Lipschitz neural networks are not without limitations. One principal challenge shared with many adversarial defense methods is the inherent trade-off between performance and robustness: a better robustness often comes at the expense of model accuracy on clean data, leading to a balance that must be well-understood by practitioners. Additionally, 1-Lipschitz networks may exhibit higher memory usage and increased time requirements during training, compared to standard training scheme. The enforcement of the Lipschitz constant requires specific computations which contribute to resource-intensive processes. As a result, the deployment of 1-Lipschitz networks demands careful consideration of the particular application context and the acceptable trade-offs between robustness, performance, and resource utilization.

Robust 1-Lipschitz networks are still an active research topic. Innovation is expected in terms of network architectures to overcome current limitations. Furthermore, there is potential for exploring new tasks to expand the applicability and capabilities of 1-Lipschitz networks.

Bibliography

Anil, C., Lucas, J., and Grosse, R. (2019). Sorting out lipschitz function approximation.

Béthune, L., Boissin, T., Serrurier, M., Mamalet, F., Friedrich, C., and Gonzalez Sanz, A. (2022). Pay attention to your loss : understanding misconceptions about lipschitz neural networks. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A., editors, *Advances in Neural Information Processing Systems*, volume 35, pages 20077–20091. Curran Associates, Inc.

Serrurier, M., Mamalet, F., González-Sanz, A., Boissin, T., Loubes, J.-M., and del Barrio, E. (2021). Achieving robustness in classification using optimal transport with hinge regularization.

Tian, Z., Shen, C., Chen, H., and He, T. (2019). Fcos: Fully convolutional one-stage object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9627–9636.

Title: Methodological Guidelines for 1-Lipschitz neural networks

Keywords: Robustness, 1-Lipschitz networks

This document serves as a starting point for understanding and implementing the training of 1-Lipschitz networks to achieve robust properties, particularly against adversarial attacks.

Our partners: