Local Mixup: Interpolation of closest input signals to prevent manifold intrusion

Raphael Baena, Lucas Drumetz, Vincent Gripon

Signal Processing 219 (2024) 109395

Contents lists available at ScienceDirect

Signal Processing

journal homepage: www.elsevier.com/locate/sigpro

Local Mixup: Interpolation of closest input signals to prevent manifold intrusion

Raphael Baena *, Lucas Drumetz, Vincent Gripon

IMT Atlantique, Lab-STICC, UMR CNRS 6285, Brest, F-29238, France

ARTICLE INFO

Dataset link: https://github.com/raphael-baena/Local-Mixup

Keywords:
Mixup
Regularization
Manifold intrusion
Interpolation

ABSTRACT

In Machine Learning, Mixup is a data-dependent regularization technique that consists in creating virtual samples by linearly interpolating input signals and their associated outputs. It has been shown to significantly improve accuracy on standard datasets, in particular in the field of vision. However, authors have pointed out that Mixup can produce out-of-distribution virtual samples and even contradictions in the augmented training set, potentially resulting in adversarial effects. In this paper, we introduce *Local Mixup* in which distant input samples are weighted down when computing the loss. In constrained settings we demonstrate that *Local Mixup* can create a trade-off between bias and variance, with the extreme cases reducing to vanilla training and classical Mixup. Using standardized computer vision benchmarks, we also show that *Local Mixup* can improve test accuracy.

1. Introduction

Deep Learning has become the golden standard for many tasks in the fields of machine learning and signal processing. Using a large number of tunable parameters, Deep Neural Networks (DNNs) are able to identify subtle dependencies in large training datasets to be later leveraged to perform accurate predictions on previously unseen data. Without constraints or enough samples, many models can fit the training data (high variance) and it is difficult to find the ones that would generalize correctly (low bias).

Regularization techniques have been deployed with the aim of improving generalization [1]. In Adamixup [2], the authors categorize these techniques into data-independent or data-dependent ones. For example, some data-independent regularization techniques constrain the model by penalizing the norm of the parameters, for instance through weight decay [3]. A popular data-dependent regularization technique consists of artificially increasing the size of the training set, which is referred to as *data augmentation* [4]. In the field of computer vision, for example, it is very common to generate new samples using basic class-invariant transformations [5,6].

In [7], the authors introduce *Mixup*, a data augmentation technique in which artificial training samples $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$, called *virtual* samples, are generated through linear interpolations between two training samples $(\mathbf{x}_i, \mathbf{y}_i)$ and $(\mathbf{x}_j, \mathbf{y}_j)$. The same linear weights are used to mix the input signals and their respective outputs. *Mixup* has been shown to improve generalization error of state-of-the-art models on ImageNet, CIFAR, speech, and even tabular datasets [7]. This method is also used in the context of few shot learning [8,9].

By using linear interpolation, *virtual* samples can in some cases contradict one another, or even generate out-of-distribution inputs. Such spurious virtual samples are obviously more likely to occur when the input signals lie on a complex topology on which linear interpolations in the Euclidean space in which they are embedded are not likely to produce meaningful results. This phenomenon has been recently described in Adamixup [2], where the authors use the term *manifold intrusion*. As such, it is not clear if *Mixup* is always desirable. More generally, the question arises of whether *Mixup* could be constrained to reduce the risk of generating such spurious interpolations. In this paper we introduce *Local Mixup*, where *virtual* samples are weighted in the training loss. The weight of each possible *virtual* sample depends on the distance between the endpoints of the corresponding segment $(\mathbf{x}_i, \mathbf{x}_j)$. In particular, this method can be implemented to forbid interpolations between samples that are too distant from each other in the input domain, reducing the risk of generating spurious virtual samples. Obviously, such a methodology is expected to perform better when the input space metric is meaningful with respect to the considered task.

Here are our main contributions:

- Our work contributes more broadly to better understanding the impact of *Mixup* during training.
- We introduce *Local Mixup*, a mixup method depending on a single parameter whose extremes correspond to classical *Mixup* and Vanilla (i.e. baseline without mixup).

* Corresponding author.
E-mail address: raphael.baena@enpc.fr (R. Baena).

https://doi.org/10.1016/j.sigpro.2024.109395
Received 13 December 2022; Received in revised form 30 October 2023; Accepted 15 January 2024
Available online 22 January 2024

- In dimension one, we prove that *Local Mixup* allows to select a bias/variance trade-off (Theorem 4.2 and 4.4).
- Extending our analysis to higher dimensions, we demonstrate that *Mixup* imposes a lower bound on the Lipschitz constant of the model, which can be tuned using *Local Mixup* (Theorem 4.6). To empirically support this claim, we present results illustrating the evolution of this bound on the CIFAR10 dataset.
- Using standard vision datasets, we show that *Local Mixup* can help achieving more accurate and robust models than classical *Mixup*.

2. Related work

Introducing notations: In Machine or Deep Learning, a training dataset $\mathcal{D}_{train}$ is used to learn the model's parameters, and a test one $\mathcal{D}_{test}$ is used to evaluate the performance of the model on previously unseen inputs [10]. We also consider that both input and output data lie in metric spaces $(\mathcal{X}, d_X)$ and $(\mathcal{Y}, d_Y)$. Typically, $\mathcal{X}$ and $\mathcal{Y}$ are assumed to be Euclidean spaces with the usual metrics. We denote by $f : \mathcal{X} \to \mathcal{Y}$ the parametric model to be trained and by $\mathcal{F}$ the hypothesis set, i.e. the set containing all candidate parametrizations of the model $f \in \mathcal{F}$.

To train our model, we use an *error function* $\mathcal{L}$ that measures the discrepancy between the model outputs and expected ones. Training the model amounts to minimizing the training loss while generalization may be quantitatively evaluated by the test loss:

$$L_{vanilla} = \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \mathcal{L}(f(\mathbf{x}), \mathbf{y}).$$

Data augmentation and mixup: To improve generalization one can use regularization techniques [1]. Among them, data augmentation is a form of data-dependent regularization [2]. It artificially generates new samples, resulting in increasing $\mathcal{D}_{train}$ [4], and can apply on the outputs $\mathbf{y}$ [11] or on the inputs $\mathbf{x}$ [5,6,12–15]. The use of data-dependent methods relying on some sort of *mixing* has recently emerged [7,13,14,16–23]. They usually mix two or more inputs and the corresponding labels. The pioneering mixing method is *Mixup* [7], where mixed samples $(\tilde{x}, \tilde{y})$ are generated by linear interpolations between pairs, i.e. $\tilde{\mathbf{x}}_{i,j,\lambda} = \lambda\mathbf{x}_i + (1-\lambda)\mathbf{x}_j$ and $\tilde{\mathbf{y}}_{i,j,\lambda} = \lambda\mathbf{y}_i + (1-\lambda)\mathbf{y}_j$ for some training samples $(\mathbf{x}_i, \mathbf{y}_i)$ and $(\mathbf{x}_j, \mathbf{y}_j)$ and some $\lambda \in [0,1]$. The *Mixup* training criterion is defined as:

Definition 2.1 (*Mixup Criterion*)**.** Let $\lambda \sim Beta[\alpha, \beta]$, n the size of dataset, i, j discrete variables uniformly drawn with repetitions in $\{0, \dots, n-1\}$. The function f^* that minimizes the Mixup criterion is:

$$f^* = \underset{f\in\mathcal{F}}{\operatorname{argmin}} \frac{1}{n^2}\mathbb{E}_\lambda \left[\underbrace{\sum_{\mathcal{D}^2_{train}} \mathcal{L}\left(\tilde{\mathbf{y}}_{i,j,\lambda}, f(\tilde{\mathbf{x}}_{i,j,\lambda})\right)}_{L_{mixup}} \right].$$

In other words, *Mixup* encourages the model f to associate linearly interpolated inputs with the corresponding linearly interpolated outputs [7]. The positive effect of this linear behavior in between samples questioned several authors who aimed at theoretically and empirically explaining *Mixup*'s behavior. For example, in [24] the authors show that *Mixup* can be interpreted as the combination of a data transformation and a data perturbation. A first transform shrinks both inputs and outputs towards their mean. The second transform applies a zero mean perturbation. The proof is given by reformulating the *Mixup* loss. [25] highlights that *Mixup* impacts the Lipschitz constant L of the gradient of the network.

Improvements over mixup: In other works, authors propose to improve *Mixup* using various approaches. For example in [19], the idea is to use different λ_x, λ_y to mix the input and the outputs, in [14,20,23], the authors explore using other (i.e. nonlinear) interpolation methods, while in [21,22,26] the authors extend the mixing to more than two elements.

Our proposed approach: In this paper, we aim at avoiding the phenomenon described as *manifold intrusion*, and introduced in Mixup [2]. This phenomenon is depicted in Fig. 1 on the right, where we see that virtual samples created through *Mixup* between distant red samples lie outside the manifold domain for the red class. As we do not have access to the underlying manifold domains when we train a model, the rationale of our contribution is to favor interpolations between samples that are close enough in the input domain. While the method described in [2] learns which interpolations should be kept through training, we advocate in this paper for a purely geometric approach where a decreasing weight is applied when computing the loss depending on the distance between interpolated samples.

3. Mixup in dimension 1

Let us consider the simple case where our model f is defined on $\mathbb{R}$. Without loss of generality, let us consider that the training set $\mathcal{D}_{train} = \{x_i, y_i\}$ is ordered by increasing inputs, i.e, $x_i \le x_{i+1}$.

For a given $\tilde{x}$, *Mixup*'s loss implies that the output $f^*(\tilde{x})$ of the model is determined by the set $\mathcal{E}(\tilde{x})$ of all convex combinations that give $\tilde{x}$ from any two training inputs x_i and x_j: $\mathcal{E}(\tilde{x}) = \{i, j, \lambda_{i,j} | \tilde{x} = \lambda_{ij}x_i + (1-\lambda_{ij})\, x_j\}$. In dimension 1, the $\mathcal{E}(\tilde{x})$ is non empty and finite for any $\tilde{x} \in [x_0, x_{n-1}]$, and finite. In practice, the distribution of λ can be uniform [7,16] $\lambda \sim Beta(\alpha = 1, \beta = 1) = \mathcal{U}(0,1)$. In this case, we show that the output $f^*(\tilde{x})$ for an input $x \in [x_0, x_n]$ is the barycenter of the target values corresponding to the points of $\mathcal{E}(\tilde{x})$.

Lemma 3.1. *Let us assume that the error function $\mathcal{L}$ is either the cross entropy or the L2 loss, then the function f^* is described for any x in $[x_0, x_{n-1}]$ by the following equation:*

$$f^*(\tilde{x}) = \frac{1}{card(\mathcal{E}(\tilde{x}))} \sum_{(i,j,\lambda_{i,j})\in\mathcal{E}(\tilde{x})} \lambda_{i,j} y_i + (1-\lambda_{i,j}) y_j. \tag{1}$$

Proof. Let $\tilde{x} \in [x_0, x_{n-1}]$ and $0 \le \lambda \le 1$. For a given triplet $(i, j, \lambda) \in \mathcal{E}(\tilde{x})$, we have $\mathbb{E}[\mathcal{L}(y_i, j, \lambda_{i,j}, f^*(\tilde{x}))|\tilde{x}, i, j, \lambda_{ij}] = \mathcal{L}(y_i, j, \lambda_{i,j}, f^*(\tilde{x}))$ as the values of $y_i, j, \lambda_{i,j}$ and $\tilde{x}$ are known. Then we minimize the error for all $y_i, j, \lambda_{i,j}$ given by $\mathcal{E}(\tilde{x})$. Then the value of $f^*(x)$ is only determined by the sum of the losses over $\mathcal{E}(\tilde{x})$ since the elements of $\mathcal{E}(\tilde{x})$ are equally probable (distributions of i, j, λ are uniform).

$$\begin{aligned}\mathbb{E}\mathcal{L}(f^*(\tilde{x}), y_{i,j,\lambda_{i,j}}) &= \sum_{\mathcal{E}(\tilde{x})} \mathbb{E}[\mathcal{L}(f^*(\tilde{x}), y_{i,j,\lambda_{i,j}})|\tilde{x}, i, j, \lambda_{ij}] \\ &= \sum_{\mathcal{E}(\tilde{x})} \mathcal{L}(f^*(\tilde{x}), y_{i,j,\lambda_{i,j}})\end{aligned} \tag{2}$$

We assume $\mathcal{L}$ to be either the cross entropy (for classification tasks) or the squared L2 loss (for regression tasks). In either case, by nulling the derivative of Eq. (2) w.r.t. the value $f^*(\tilde{x})$, we get:

$$f^*(\tilde{x}) = \frac{1}{card(\mathcal{E}(\tilde{x}))} \sum_{\mathcal{E}(\tilde{x})} y_{i,j,\lambda_{i,j}} \quad \square$$

A consequence of this lemma is the following theorem:

Theorem 3.2. *Considering a L2 loss or the cross entropy, the function f^* that minimizes the loss on the training set is piecewise linear on $[x_0, x_{n-1}]$, linear on each segment $[x_i, x_{i+1}]$, $0 \le i < n-1$; and defined by Eq. (1).*

When $\tilde{x}$ varies in $[x_i, x_{i+1}]$, the set of possible combinations (between training samples) leading to $\tilde{x}$ does not change, only the corresponding coefficients λ vary linearly. Since the expression of Eq. (1) is linear in each of those coefficients, f^* is itself linear as a function of $\tilde{x}$. The set of possible combinations will change whenever $\tilde{x}$ switches to another interval, e.g. $[x_{i-1}, x_i]$. In this case new combinations are possible and others may disappear, leading to another linear function. f^* is still

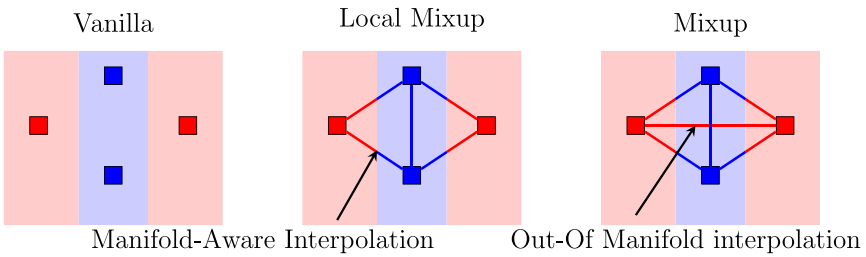

Fig. 1. Illustration of *Local Mixup* approach, an improvement of *Mixup* [2] that considers the data's geometry locally. On the left, only original data samples are used without any interpolations, and the filled regions represent the ground truth. In the middle, *Local Mixup* is shown, where **interpolations are performed only between nearby samples,** avoiding conflicts with the ground truth. Interpolations $(\tilde{x}, \tilde{y})$ follow linear equations: $\tilde{x} = \lambda x_1 + (1-\lambda)x_2$ and $\tilde{y} = \lambda y_1 + (1-\lambda)y_2$ [2]. On the right, *Mixup* is depicted, **where interpolations are applied to all data samples, leading to contradictory virtual samples**, such as the red segment (representing red intra-class interpolations) crossing the blue class area. For clarity, segments are colored based on a thresholded argmax of the interpolation outputs, rather than using a gradient of colors.

continuous everywhere because new or disappearing combinations are associated either to $\lambda = 0$ or $\lambda = 1$ for $\tilde{x} = x_j$ and $j \in \{1, \dots, n\}$.

In practice inferring a function f^* that minimizes that the *Mixup* criterion is usually not desired in machine learning, and one looks for f with a sufficiently small loss to have a regularizing effect. Indeed f^* is not likely to generalize well. Still, we note that it tends to an average of convex combinations and thus leads to a model with a low variance.

In practice, it is generally not the goal in machine learning to infer a function f^* that minimizes the *Mixup* criterion: such function is unlikely to generalize effectively. Instead, the aim is to find a function f that achieves a sufficiently low loss.

However, we note that attempting to minimize the *Mixup* criterion tends to yield solutions that converge towards an average of convex combinations, resulting in a model with reduced variance.

4. Local mixup

4.1. Locality graphs

Consider a (training) dataset D made of pairs $(\mathbf{x}, \mathbf{y})$. We propose to build a graph from D as follows. We define $G_D = \langle V, \mathbf{W} \rangle$ where $V = \{\mathbf{x} \mid \exists \mathbf{y}, (\mathbf{x}, \mathbf{y}) \in D\}$. The real symmetric matrix $\mathbf{W}$ is based on D, the pairwise distance matrix $D[i,j] = d_{\mathcal{X}}(\mathbf{x}_i, \mathbf{x}_j)$. The distance used in our experiments is the Euclidean metric.

In this work, we consider various ways to obtain $\mathbf{W}$, but the rationale is always the same: to obtain a similarity matrix where large weights correspond to closest pairs of samples. Namely, we consider K-nearest neighbor graphs, where we set to 1 weights of target vertices corresponding to the K closest samples for a given source vertex and 0 otherwise; thresholded graphs where $\mathbf{W}[i,j] = \phi(D[i,j])$ and $\phi(d) = \mathbf{1}_{d \leq \varepsilon}$; smooth decreasing exponential graphs where $\mathbf{W}[i,j] = \exp(-\alpha \mathbf{D}[i,j])$. The loss is then weighted using $\mathbf{W}$:

$$L_{\text{local mixup}} = \sum_{\mathcal{D}^2_{train}} \mathbf{W}[i,j] \mathcal{L}\left(\tilde{\mathbf{y}}_{i,j,\lambda}, f(\tilde{\mathbf{x}}_{i,j,\lambda})\right). \tag{3}$$

For computational cost considerations, we compute a graph for each batch (random subset) of samples during stochastic gradient descent. As such, the weights associating two samples can vary depending on the chosen graph and random batch.

In the extreme case where some weights are 0, the corresponding virtual samples are discarded during gradient descent, resulting in only considering local interpolations of samples, hence the name *Local Mixup*.

4.2. Low dimension

In this section, we are interested in proving that *Local Mixup* allows to tune a trade-off between bias and variance on trained models. For this purpose, we simplify the problem to dimension 1 and we consider a thresholded graph. In this case, note that varying the threshold can create a range of settings where a threshold of 0 boils down to vanilla training and N, where N is the number of training samples, boils down to classical *Mixup*.

4.2.1. Local mixup and the bias/variance trade-off

Let us first recall the definitions of the bias and variance in the context of a machine learning problem.

Definition 4.1 (*Bias and Variance*)**.** Let us consider a training set $\mathcal{D}_{train}$ and a function f from $\mathcal{X}$ to $\mathcal{Y}$. We define Bias and Variance as follow:

- *Bias*: $Bias(f)^2 = \mathbb{E}_{train}[(f(x) - y)^2]$.
- *Variance*: $Var(f) = \mathbb{E}_{train}[(f - \mathbb{E}_{train}[f])^2]$.

We consider two settings. In the first one, the input domain $\mathbb{Z}/n\mathbb{Z}$ is periodic and thus the number of samples is finite. In the second one, the input domain $\mathbb{Z}$ is infinite and outputs are independent and identically distributed (i.i.d) using a random variable. In both settings, the size of the dataset N can be arbitrary large so we can study the asymptotic cases.

Periodic setting

Let us consider that the training set $\mathcal{D}_{train}$ is made of pairs (x, y), where $\{x \mid \exists y, (x,y) \in \mathcal{D}_{train}\} = \mathbb{Z}/n\mathbb{Z}$. We also consider $d_{\mathcal{X}}(x, x') = |x - x'| \in \{0, \dots, n-1\}$. The reason for this assumption is that it considerably simplify proofs due to samples being regularly spaced, but we believe similar results could be drawn even if inputs are not defined over $\mathbb{Z}/n\mathbb{Z}$.

As the input domain is discrete, we can write K the threshold parameter of the graph as an integer. Note that each time K is increased by 1, each sample is connected to 2 additional neighbors. In this case, we can write an explicit formulation of f^*_K, the function that minimizes the *Local Mixup* criterion for K-thresholded graphs. Following similar arguments to those used to obtain Eq. (1): for a given x_i we know that the optimal value for $f^*_K(x_i)$ would be an average of the $\tilde{y}$ that correspond to the possible interpolations. we obtain:

$$\forall x_i \in \mathbb{Z}/n\mathbb{Z}, f^*_K(x_i) = \frac{1}{K(K+3)/2}(2Ky_i + S_K(x_i)), \tag{4}$$

where $S_K(x_i)$ is defined recursively as follows:

$$S_{K+1} = \begin{cases} 0 & \text{if } K = 0 \\ S_K(x_i) + A_{K+1}(x_i) & \forall K \geq 1 \end{cases}. \tag{5}$$

and:

$$A_K(x_i) = \frac{1}{K} \sum_{k=1}^{K-1} (K-k) \cdot y_{i-k} + k \cdot y_{i+K-k}.$$

Proof. These expressions can be proved by induction over K. Let $x_i \in \mathbb{Z}$, $K > 1$. The term $\frac{1}{K(K+3)/2}$ is the cardinality of $\mathcal{E}(x_i)$, i.e the number of interpolations such that $x_i = \lambda x + (1-\lambda)x'$. More precisely, we distinguish *direct interpolations* and *indirect interpolations*. Direct interpolations are interpolations between x_i and its direct neighbors x_j ($\lambda = 1$): $x_i = x_i + 0.x_j$; they lead to $y = \lambda y_i + (1-\lambda)y_j = y_i$. Indirect interpolations are interpolations between points other than x_i that happen to intersect x_i. On Fig. 2 we depicted these interpolations.

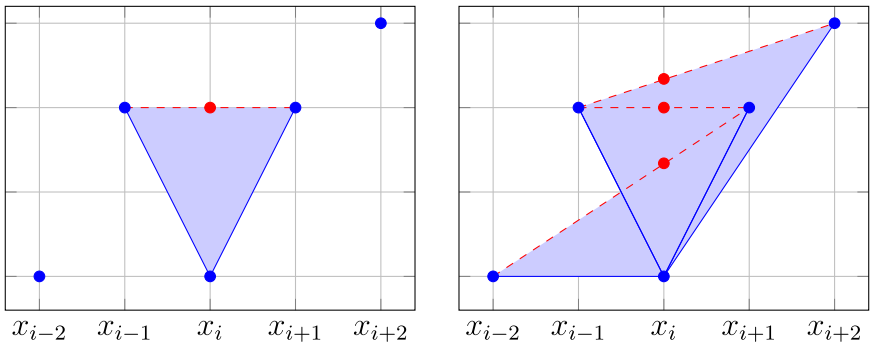

Fig. 2. We depict here the terms of $f_K^*(x_i)$ given by Eq (4) for different K. In blue the interpolations corresponding to $2Ky_i$ and in red the terms of the sum S_K. On the right, $K = 2$ and on the left $K = 1$.

As we increase K, the influence of S_K (red points) increases as we will prove below.

We show by induction over K that there are $\frac{1}{K(K+3)/2}$ interpolations or more precisely $2K$ direct interpolations and $K(K-1)/2$ indirect interpolations.

- For $K = 1$, x_i is connected to x_{i-1} and x_{i+1} so there are 2 neighbors (see Fig. 2); No indirect interpolation can occur since $|x_{i-1} - x_{i+1}| = 2$ and $K = 1$.
- For $K = 2$, x_i is connected to $x_{i-1}, x_{i+1}, x_{i-2}, x_{i+2}$ so there are 4 neighbors; there is 1 indirect interpolation: $1/2(y_{i-1} + y_{i+1})$ (see Fig. 2).
- Assuming the expression is true for some K, f_{K+1}^* contains: the $2K + K(K-1)/2$ interpolations (from f_K^*), the direct interpolations: (x_i, x_{i+K+1}), (x_i, x_{i-K-1}) and indirect interpolations: $(x_{i-1} + x_{i+K}), (x_{i-1}, x_{i+K}), \dots, (x_{i-K}, x_{i+1})$. In total there are $K(K+3)/2 + 2 + K = (K+1)(K+1+3)/2$ interpolations.

Now, one can see that direct interpolations correspond to the term $2Ky_i$ and indirect interpolations to the term S_K. □

We obtain the following Lemma, showing that the expected value of f_K^* is invariant with respect to K:

Lemma 4.1. *[Expected value of f_k^*] For any K, the expected value of f_K^* is*

$$\mathbb{E}_{train}[f_K^*] = \mathbb{E}_{train}[y]. \tag{6}$$

Proof.

$$\mathbb{E}_{train}[f_K^*] = \frac{1}{n}\sum_{i=1}^{n} f_K^*(x_i)$$

$$= \frac{2}{nK(K+3)}(2nK\mathbb{E}_{train}[y] + \sum_{i=1}^{n}\sum_{k=1}^{K} A_k(x_i)).$$

and using the fact that $y_{i+n} = y_i$:

$$\sum_{i=1}^{n}\sum_{k=1}^{K} A_k(x_i) = \sum_{i=1}^{n}\sum_{k=1}^{K}\sum_{l=1}^{k-1} \frac{k-l}{k} y_{i-l} + \frac{l}{k} y_{i+k-l}$$

$$= \sum_{i=1}^{n} y_i \sum_{k=1}^{K}\sum_{l=1}^{k-1} 1 = n\mathbb{E}_{train}[y]\frac{K(K-1)}{2}.$$

then $\mathbb{E}_{train}[f_K^*] = \mathbb{E}_{train}[y]$. □

We obtain the following theorem:

Theorem 4.2 (*Convergence of f_K^* in the periodic setting*)**.** *As K grows, it holds that:*

$$\forall x_i \in \mathbb{Z}/n\mathbb{Z},$$

$$f_K^*(x_i) \to \mathbb{E}_{D_{train}}[y], \tag{7}$$

$$Bias^2(f_K^*) \to \mathbb{E}_{train}[(y_i - \mathbb{E}_{train}[y])^2], \tag{8}$$

$$Var(f_K^*) = \mathbb{E}_{train}[\left(f_K^*(x_i) - \mathbb{E}_{train}[f_K^* x_i]\right)^2] \to 0, \tag{9}$$

$Var(f_K^*)$ is eventually nonincreasing.

Proof. We can explicitly write the limit of S_K. We first prove this lemma:

Lemma 4.3. *Let $K = Mn + r$, $M \in \mathbb{N}^*$ and $0 < r < n - 1$. We assume $\mathbb{E}_{train}(y) \geq 0$, then:*

$$(M+1)n \cdot \mathbb{E}_{train}(y) + \eta \geq A_K \geq Mn \cdot \mathbb{E}_{train}(y) - \eta, \tag{10}$$

with $\eta = \mathcal{O}(K\mathbb{E}_{train}(y))$.

Proof. Let be $K = Mn + r$, $M > 1$, $n - 1 > r > 0$. We have:

$$A_K = \frac{1}{K}\sum_{k=1}^{K-1}(K-k)\cdot y_{i-k} + k \cdot y_{i+K-k}$$

The sum above can be decomposed into the sum:

$$A_K = \frac{1}{K}\sum_{m=0}^{M-1}\sum_{k=1}^{n-1}(K-k-mn)y_{i-k-mn} + (k+mn)y_{i+K-(mn+k)} + \sum_{k=1}^{r-1}(K-k-Mn)y_{i-k} + (K-k)y_{i+K-k}$$

For $m \geq 0$ and $1 \leq k \leq n-1$, we have $y_{i-mn-k} = y_{i-k}$ and $y_{i+K-(mn+k)} = y_{i+(M-m)n+r-k} = x_{i+r-k}$ since y is a periodic signal of period n and $K = Mn + r$. Then:

$$A_K = \frac{1}{K}\sum_{m=0}^{M-1}\sum_{k=1}^{n}(K-k-mn)y_{i-k} + (k+mn)y_{i+r-k} + \sum_{k=1}^{r-1}(K-k-Mn)y_{i-k} + (k+Mn)y_{i+r-k}$$

$$= \frac{1}{K}\sum_{m=0}^{M-1}\sum_{k=1}^{n}(K-k-mn)y_{i-k} + (k+r+mn)y_{i-k} + \sum_{k=1}^{r-1}(K-k-Mn)y_{i-k} + (k+Mn)y_{i+r-k}$$

$$= \sum_{k=1}^{n} y_{i-k}M(1+r/K) + \eta$$

with $\eta = \| \sum_{k=1}^{r-1}(K-k-Mn)y_{i-k} + (k+Mn)y_{i+r-k} = \mathcal{O}(K\mathbb{E}[y])\|$, the quantity η compared to $K\mathbb{E}[y] \geq 0$ will be negligible. For now, let assume that $\mathbb{E}[y] \geq 0$. We have:

$$Mn\mathbb{E}[y] - \eta \leq A_K \leq (M+1)n\mathbb{E}[y] + \eta. \tag{11}$$

Similarly, if $\mathbb{E}[y] \leq 0$, we have:

$$(M+1)n\mathbb{E}[y] - \eta \leq A_K \leq Mn\mathbb{E}[y] + \eta. \tag{12}$$

Then in both cases, eventually, $K = Mn+r \sim Mn$ and $A_K \sim KE[y]$. □

With the previous Lemma combined and Eq. (5) we can demonstrate the convergence of the sum S_K and find its limit with the following corollary.:

Corollary 4.3.1. *For $K = nM \to \infty$*

$$S_K \to \frac{1}{2}\sum_{i=1}^{n} y_i M^2 n = \frac{1}{2}\mathbb{E}_{train}(y)K^2. \tag{13}$$

Proof. First, we show the convergence of S_K/K^2. This is direct with the equations above: one can write $\|A_K\| \sim K\|\mathbb{E}[y]\|$ and $\sum_{k=1}^{K} K\mathbb{E}[y]/K^2 = \mathbb{E}[y]$. So S_K/K^2 is absolutely convergent. Now we seek an equivalent of S_K/K^2. To do so, we use either Eq. (11) or (12) (depending on the sign of $\mathbb{E}[y]$). Without loss of generality let us use Eq. (11):

As $S_K = \sum_k^K A_k$, for $K = Mn$ we have:

$$n\sum_{m=0}^{M-1}(m+1)n\mathbb{E}[y] \geq S_K \geq \sum_{m=0}^{M-1} mn\mathbb{E}[y]$$

$$n^2M(M+1)/2\ \mathbb{E}[y] \geq S_K \geq n^2(M-1)(M+1)/2\ \mathbb{E}[y]$$

Then,

$$S_k \to \frac{1}{2}\mathbb{E}(y)K^2 \tag{14}$$

The same result holds if $\mathbb{E}[y] \leq 0$. □

To prove the monotonicity of the variance we want to show: $Var(f^*_{K+1}) \leq Var(f^*_K)$ for K large enough. We use the König-Huygens theorem and Lemma 4.1 to compute the difference between the two variances:

$$\begin{aligned} &Var(f_{K+1}) - Var(f_K) \\ &= \mathbb{E}_{D_{train}}[\left(f_{K+1}(x)\right)^2] - \mathbb{E}_{D_{train}}[\left(f_K(x)\right)^2] \\ &= \mathbb{E}_{D_{train}}[\left(f_{K+1}(x)\right)^2 - \left(f_K(x)\right)^2]. \end{aligned}$$

We then show that for any $x \in [x_0, x_{n-1}]$ and K large enough, $\left(f_{K+1}(x)\right)^2 \leq \left(f_K(x)\right)^2$. To do so we get an asymptotic equivalent:

$$\left(f_{K+1}(x)\right)^2 - \left(f_K(x)\right)^2 \sim -\frac{K}{C}\cdot E^2_{train}[y],$$

where C is a positive constant. □

This theorem states two main results: (1) in the case of *Mixup* the function that minimizes the loss f^* has zero variance and converges to $\mathbb{E}_{train}[y]$. (2) Eventually the variance of the function that minimizes the *Local Mixup* criterion is decreasing, showing that the proposed *Local Mixup* can indeed tune the trade-off between the bias and variance.

i.i.d random output setting

Let us now consider that the training set is made of inputs $\{x \mid \exists y, (x,y) \in D_{train}\} = \mathbb{Z}$ and y_i are i.i.d. according to a random variable R of variance σ^2.

Theorem 4.4. *For a signal with i.i.d outputs, the variance is eventually bounded by:*

$$\frac{4^2\sigma^2}{K^2} \leq Var(f_K(x_i)) \leq \frac{8\sigma^2}{K}. \tag{15}$$

Proof. Let us choose x_i and $K > 1$. First observe that $f^*_K(x_i)$ is a sum of random variables. We rewrite S_K with the coefficients $a^K_k = \sum_{l=k+1}^{K} \frac{l-k}{l}$: $S_K = \sum_{k=1}^{K-1}(y_{i-k} + y_{i+k})a^K_k$. We obtain:

$$Var(f^*_K(x_i)) = Var\left(\frac{2\cdot(2Ky_i + S_K)}{K(K+3)}.\right)$$

leading to:

$$Var(f^*_K(x_i)) = 4^2\left(\frac{K}{K(K+3)}\right)^2 Var(y_i)$$

$$+ \sum_{k=1}^{K-1}\left(\frac{2a^{(K)}_k}{K(K+3)}\right)^2 (Var(y_{i-k}) + Var(y_{i+k})).$$

We use the fact that $\frac{1}{K} \leq a^K_k \leq K$. Then when $K \to \infty$:

$$\frac{4^2\sigma^2}{K^2} \leq Var(f_K(x_i)) \leq \frac{8\sigma^2}{K}. \quad \square$$

This proof can be generalized without difficulty to signals on $\mathbb{R}$ as long as the dataset is finite an large enough.

4.2.2. Invariance of linear models

Interestingly, we can show that both *Mixup* and *Local Mixup* lead to the same optimal linear models, as stated in the following theorem:

Theorem 4.5. *For a linear model: $f(x) = ax + b,\ a, b \in \mathbb{R}$, the functions that minimizes the loss of Mixup and Local Mixup are equal (that function is denoted as f^*).*

Proof. For *Mixup*, we showed with Eq. (1) the function f^* is a piecewise linear function. The same equation applies for *Local Mixup* except that the set E_x is smaller for *Local Mixup* as the number of endpoints is restricted. As a piecewise linear function, linear on each segment $[x_i, x_{i+1}]$: f^* can be written as $f^* = a_i x + b_i$ where each (a_i, b_i) are defined on $[x_i, x_{i+1}]$. Let us consider $\mathcal{F}$ to be restricted to linear functions, then the coefficients a, b are the averages of the (a_i, b_i). □

4.3. High dimension and lipschitz constraint

The proofs given in low dimension have some limitations. Basically, the averaging effect happens since any point x within the interval $[x_1, x_n]$ can be written as at least one convex combination of pairs from the training set. Contradictions may occur as illustrated above when several combinations corresponds to x. In higher dimension such explicit contradictions are not necessarily expected since the probability of a training sample being an interpolation of two others training samples tends to zero [27]. Still, we show that *Local Mixup* has an impact on the Lipschitz constant of the networks.

First recall the definition of a q-Lipschitz function:

Definition 4.2 (*Lipschitz Continuity and Lipschitz Constant*)**.** Given two metric spaces $(\mathcal{X}, d_X), (\mathcal{Y}, d_\mathcal{Y})$ and a function $f : \mathcal{X} \to \mathcal{Y}$, f is Lipschitz continuous if there exists a real constant $q \geq 0$ s.t for all x_i and x_j in $\mathcal{X}$,

$$d_\mathcal{Y}\left(f(x_i), f(x_j)\right) \leq q d_\mathcal{X}(x_i, x_j). \tag{16}$$

If f is q-Lipschitz continuous, we define the optimal Lipschitz constant Q_{sup} as

$$Q_{sup} = \sup_{x_i, x_j \in X, x_i \neq x_j} \frac{d_\mathcal{Y}\left(f(x_i), f(x_j)\right)}{d_\mathcal{X}(x_i, x_j)}. \tag{17}$$

For simplicity, let us consider a classification problem where $d_\mathcal{Y}$ is 0 if the two considered samples are of the same class and 1 otherwise.

Then the training set imposes a lower bound on the Lipschitz constant:

$$Q_{sup} \geq \underbrace{\left(\min_{\mathbf{x}_i, \mathbf{x}_j \in D, y_i \neq y_j} d_\mathcal{X}(x_i, x_j)\right)^{-1}}_{Q(D)}. \tag{18}$$

For *Mixup* and *Local Mixup*, the virtual samples increase the size of the training set, resulting in stronger constraints on the optimal Lipschitz constant.

In more details, consider the case of a thresholded graph with parameter ε when using *Local Mixup*. In this case, the increased training set for each class $\mathbf{y}$ can be written as $S_\varepsilon(\mathbf{y}) = \{\lambda\mathbf{x}_i + (1-\lambda)\mathbf{x}_j \mid 0 \leq \lambda \leq 1\ \mathbf{y}_i = \mathbf{y}_j = \mathbf{y}, d_\mathcal{X}(\mathbf{x}_i, \mathbf{x}_j) \leq \varepsilon\}$, the set of all segments constructed from two samples that are close enough in the input domain and sharing the same label $\mathbf{y}$. We then obtain the following theorem:

Table 1
Error rates (%) on CIFAR-10, CIFAR-100 (Resnet18) Fashion-MNIST (DenseNet) and SVHN (LeNet). Values are averaged on 100 runs for CIFAR-10 and 10 runs for CIFAR-100, Fashion-MNIST and SVHN. Mean errors with their confidence interval are given.

METHOD	CIFAR-10	CIFAR-100	Fashion-MNIST	SVHN
Baseline	4.98 ± 0.03	30.6 ± 0.27	6.20 ± 0.2	10.01 ± 0.15
Mixup	4.13 ± 0.03	29.23 ± 0.4	6.36 ± 0.16	8.31 ± 0.14
Local Mixup	$\mathbf{4.03 \pm 0.03}$	$\mathbf{29.08 \pm 0.34}$	$\mathbf{5.84 \pm 0.13}$	$\mathbf{8.0 \pm 0.13}$

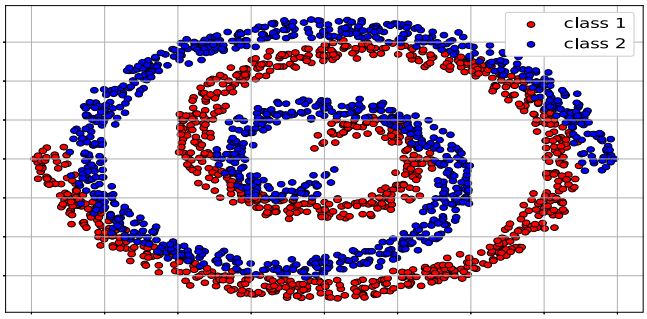

Fig. 3. Illustration of the two coiling spiral dataset with 1000 samples per class and $\sigma = 1.5$.

Theorem 4.6. *The lower bound $Q(D)$ is increasing with ε.*

Proof. We directly use the inclusion $S_\varepsilon(\mathbf{y}) \subset S_{\varepsilon'}(\mathbf{y}), \forall \varepsilon \leq \varepsilon'$. □

We shall show in the experiments that ε can indeed impact $Q(D)$ on standard vision datasets.

5. Experiments

5.1. Low dimension

As stated in the introduction and by the author of *Mixup*, this method leads to interpolations that may be misleading for the model [2]. To illustrate this effect, we consider a 2d toy dataset of two coiling spirals where such interpolations occur frequently. The two coiling spirals is a binary classification dataset: each spiral corresponds to a different class. We expect to retrieve better performance for *Local Mixup* compared to *Mixup*: local interpolations are likely to stay in the same spiral and therefore avoid manifold intrusion. For this experiment we use a thresholded graph with parameter ε.

To carry out this experiment, we generate 1000 samples for each class and add a Gaussian noise with standard deviation $\sigma = 1.5$ (controlling the spirals' thickness). A typical draw is depicted in Fig. 3. We use a large value of σ to avoid trivial solutions to the problem. Once the dataset is generated we split it randomly into two parts: a training set containing 80% of the samples and a test set containing the remaining 20% (used to compute the error rates).

We then use a fully connected neural network made of two hidden layers with 100 neurons and use the ReLU function as non linearity. We average the test errors over 1000 runs. For small values of ε many weights of the graph are zero and thus the corresponding interpolations are disregarded into the loss. This means that for a given batch only a small proportion of samples are regarded to compute the loss. Without any correction, different values of ε lead to different batch sizes. To avoid side effects, we vary the batch size so that in average the same number of samples are used to update the loss.

To select an appropriate value of ε, we first looked at the distribution of distances between pairs of inputs in the training set. This distribution is depicted in Fig. 4. We observe that the distribution is relatively uniform between 0 and 4, and as such in our experiments we vary ε between 0 and 4 using steps of 0.5.

In Fig. 4, we depict the evolution of the average error rate as a function of the parameter ε. Recall that the extremes for $\varepsilon = 0$ and $\varepsilon = 4$ correspond respectively to Vanilla and *Mixup*. One can note the significant benefit of *Mixup* and *Local Mixup* over Vanilla. As expected, *Local Mixup* presents a minimum error rate which is significantly smaller than *Mixup*'s error rate. We can note that the minimum is reached with a value of ε smaller than the first quantile. This means that for this dataset *Mixup* interpolations given above this threshold are either useless or misleading for the network's training.

It is worth pointing out that this toy dataset is particularly suitable to generate contradictory virtual samples. We delve into more complex and real world datasets in the following subsection.

5.2. High dimension

Lipschitz lower bound

To illustrate the impact of ε on the optimal Lipschitz constant, we use the dataset CIFAR-10 [28] which is made of small images of size 32×32 pixels and 3 colors. There are 50,000 images in the training set corresponding to 10 classes.

We are interested in showcasing the evolution of $Q(D)$ when varying ε. The results are depicted in Fig. 5.

For classical *Mixup* we obtained $Q(D) = 0.11$ and for Vanilla $Q(D) = 0.073$. Note that these two extremes are reached with *Local Mixup* when $\varepsilon = 0$ and $\varepsilon \geq 50$.

We observe that ε can be used to smoothly tune the lower bound $Q(D)$. In practice, a lower $Q(D)$ is preferable, but this only accounts for the optimal Lipschitz constant. Larger values of ε lead to larger training sets and thus potentially better generalization.

Experiments on classification datasets

We now test our proposition on different classification datasets and architectures. We consider the datasets CIFAR10 [28], Fashion-MNIST [29] and SVHN [30]. Fashion-MNIST is composed of clothes images of size 28×28 pixels (grayscale) . There are 60,000 images in the training set corresponding to 10 classes. SVHN is a real-world image dataset made of small cropped digits of size 32×32 pixels and 3 colors. There are 73 257 digits in the training set corresponding to 10 classes. For these tests, we use a smooth decreasing exponential graph parametrized by α.

For CIFAR10, we implement a ResNet18 [31] as in [7], and average the error rates over 100 runs. We report the mean and confidence interval at 95%. We observed that *Local Mixup* with a value of $\alpha = 0.003$ showed a smaller error rate than the Vanilla network and *Mixup*, with disjoint confidence intervals. For Fashion MNIST, we implement a Densenet [32] and average the error rates over 10 runs. We also report the mean and confidence intervals at 95%. Again, *Local Mixup* with a value of $\alpha = 1e-3$ presents a smaller error rate than both the baseline and *Mixup*. Note that for this dataset and this network architecture *Mixup* impacts negatively the error rate, suggesting that on this dataset *Mixup* creates spurious interpolations as discussed in [2]. For SVHN we implement a LeNet-5 [33] architecture (3 convolution layers). Again, *Local Mixup* performs better than both Vanilla and *Mixup*.

The decision to conduct these specific numbers of runs in our experiments was made to ensure the robustness and reliability of our results. By running multiple experiments, we were able to compute means and confidence intervals with small enough margins to distinguish the performance of different methods effectively. We did not conduct additional runs as our confidence intervals were satisfactory, and conducting more runs would have been computationally expensive without significantly altering our findings.

For these experiments, we also tried to use a K-nearest neighbor graph or a thresholded graph but without being able to achieve smaller error rates compared to *Mixup* or even Vanilla. This may indicate that some segments generated by *Mixup* are important to act as a regularizer

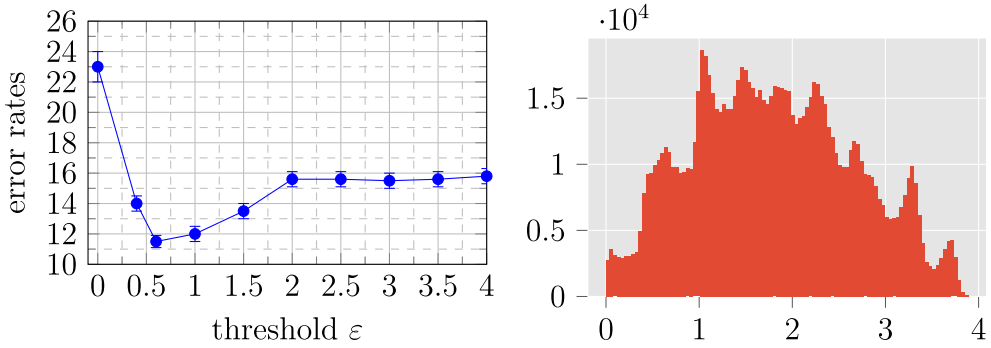

Fig. 4. On the right: Error rate as a function of ε for the two coiling spirals dataset. Values are averaged over 1000 runs. Extremes correspond respectively to Vanilla ($\varepsilon = 0$) and *Mixup* ($\varepsilon > 4$). On the left: Histogram of Euclidean distances $d_{\mathcal{X}}$ between pairs of inputs on the two coiling spirals dataset.

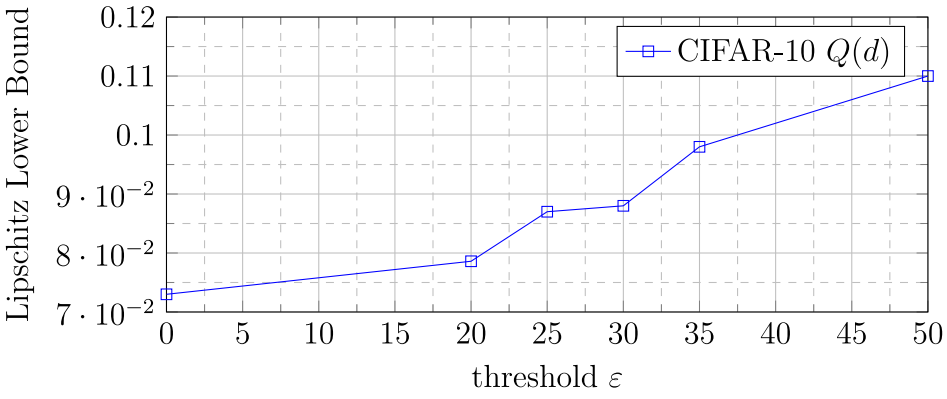

Fig. 5. Evolution of $Q(D)$ on the dataset CIFAR10. Note that $\varepsilon = 0$ corresponds to Vanilla. $\varepsilon = 50$ corresponds to classical *Mixup*.

Table 2
Black box attack: error rates (%) on CIFAR-10, for different values of noise ε.

Epsilon ε	Vanilla	Local Mixup	Mixup
0.0025	7, 55	6, 30	**5**, **75**
0.005	16, 70	12, 82	**12**, **28**
0.0075	30, 90	**22**, **81**	23, 60
0.01	45, 60	**33**, **59**	37, 00

during training even if some of them may generate manifold intrusions. By tuning α, we weigh the importance of this regularization.

Black Box Attack

As performed in the original Mixup [7] paper, we carry out a black box attack on CIFAR-10. We rescale the images between $[0, 1]$ and add a Gaussian noise $\mathcal{N}(0, \varepsilon)$ with different values of the noise standard deviation. We report the error rates for different values of noise in Table 2. We note that for low values of noise *Mixup* and *Local Mixup* are very similar. For greater values, the difference between the two methods increases and *Local Mixup* outperforms *Mixup*. This is expected, since we showed in the theoretical section that our method relaxes the Lipschitz bound imposed by *Mixup* leading to smaller values of the constant.

Comparison with *Adamixup*

We compare our proposed approach with *Adamixup* [16], another method presented as capable of preventing manifold intrusion. We use the GitHub Repository of the author and we changed only the Mixup part to implement our method. On CIFAR-10 we used 1400 epochs as well for Local Mixup and *Adamixup* since the authors used this number for *Adamixup*. We observe a slight advantage for *Adamixup* on MNIST: 0.49% ± 0.03 for *Adamixup*, 0.54% ± 0.02 for *Local Mixup* (error rates averaged over 10 runs). Still, on CIFAR-10 our method outperforms *Adamixup*: 4.11% ± 0.12 for *Adamixup* and 3.89% ± 0.12 for *Local Mixup* (averaged over 5 runs)

Note that *Local Mixup* does not completely discard interpolations but weighs them, contrary to *Adamixup* that prevents the interpolations which are considered as causing manifold intrusions. As a consequence, the use of *Adamixup* involves computing gradients on smaller batches, potentially explaining the slower convergence. Notably, the authors of Adamixup [16] extended their training by three times the standard number of epochs. While this approach poses no issues on simpler datasets such as MNIST, it becomes problematic when applied to larger datasets, significantly elongating the training process.

Furthermore, this may indicates that even the interpolations causing manifold intrusion could be beneficial as long as their are not predominant in the loss.

We also note that our method seems to converge faster on MNIST as shown on Fig. 6. It is important to emphasize that quicker convergence may not necessarily translate to better prediction performance (results on MNIST). The faster convergence observed with Local Mixup is likely attributable to the fact that AdaMixup relies on an additional neural network, resulting in less smooth loss landscapes that are more challenging to optimize. Furthermore, AdaMixup, by removing interpolations, leads to smaller batches for computing the gradient. In contrast, our approach (exponential graph) computes gradients over the entire batch. Others benefits of our proposed approach are the simplicity and the small number of parameters (CIFAR-10): 836522 for *Local Mixup* and 11171146 for *Adamixup*.

5.3. Discussion

Inter and Intra Mixup

Considering our method, one could ask whether or not *Local Mixup* it beneficial because it restricts the interpolations between samples of

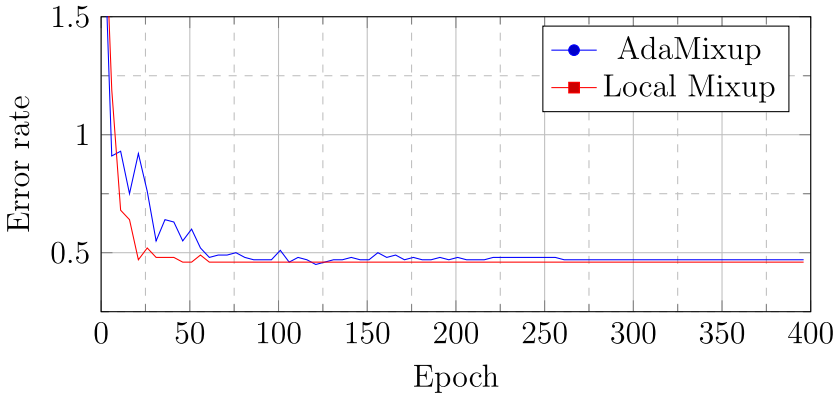

Fig. 6. Evolution of the error rate for *Local Mixup* and *Mixup* on MNIST (training data. We observe a faster convergence for *Local Mixup*.

Table 3
Error rates for different value of K when using a K-nearest neighbor graph on CIFAR-10.

K = 1	K = 5	K = 10
0.607%	0.606%	0.601%

the same class (Intra Mixup). On CIFAR-10, we run additional experiments where we allow interpolations only between samples of the same class (Intra Mixup) or only between samples of different classes (Inter-Mixup). In both cases, the error rates were worse than classical *Mixup*: for *Inter-Mixup* the error rate is 4.5% and 4.7% for Intra-Mixup. On the spiral dataset, we computed the proportion of inter-class interpolations vs. intra-class interpolation carried out by *Local Mixup*. We got 30% of intra-class interpolations and 70% for inter class. Thus, it seems necessary to use both intra and inter class mixing.

Limitations

Experiments in both low and high dimensions demonstrated the capacity of *Local Mixup* to outperform *Mixup* thanks to the use of locality. Still, the choice of the added hyper-parameter (α, ϵ or K) is essential and data dependent. For now, we reported results selecting the parameter leading to the best test error rate among a small number of possibilities. In future work we would like to rely on quantitative information given on the topology such as the histogram of the distance or persistence diagrams [34] to tune these hyper-parameters.

Note also that to embed the notion of locality we decided to use the Euclidean metric, although in general datasets lie in nonlinear manifolds. On CIFAR10 for example, in [35] the authors show that it is possible to achieve classification scores significantly better than the chance level using the Euclidean metric, but very far from state-of-the-art. There would be many possibilities to improve over using the Euclidean metric, including using pullback metrics [36,37] given by the Euclidean distance between the samples once in the feature space corresponding to the penultimate layer.

5.4. Ablation studies

Graph Construction in High Dimension Instead of smooth decreasing exponential graphs we tried to use K-nearest neighbor graphs for high dimension datasets. The graph is computed for each batch. In Table 3, we report the test error rates on CIFAR-10 for different value of K. We obtain results that significantly underperform compared to the baseline presented in Table 1.

The issue with a KNN-graph approach is its batch-dependent nature, as nearest neighbors can vary between batches. This could lead to inconsistent interpolation behavior across different batches, which is not ideal for our purposes.

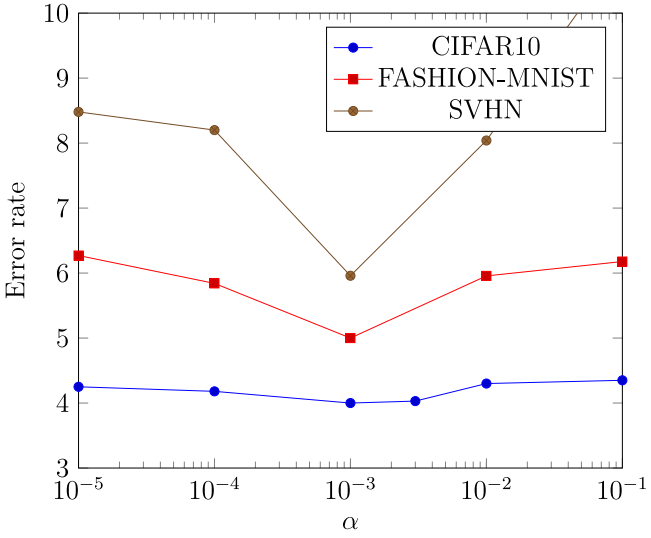

Fig. 7. Error rate evolution with respect to the hyperparameter α on various datasets. Notably, across all datasets, the optimal values tend to cluster around 10^{-3}.

We tried to compute the graph on the whole dataset but we did not notice any significant on the error rates. These results encourages us not to discard completely the interpolations outside the manifold but rather to reduce the influence in the loss.

Influence of Hyperparameter: The hyperparameters were selected through cross-validation using the test datasets provided in CIFAR10, SVHN, and Fashion-MNIST. We present the results of our additional studies conducted on various datasets to validate our hyperparameters. Given the wide range of confidence intervals, our objective is to determine an order of magnitude rather than precise values. Fig. 7 illustrates the variation in error rates concerning the hyperparameter α across different datasets. Our observations reveal that the optimal value is typically achieved at approximately 10^{-3}.

6. Conclusion

In this paper, we present an enhancement of Mixup called *Local Mixup*, in which pairs of samples are interpolated and weighted in the loss depending on the distance between them in the input domain. The underlying rationale behind this approach is to generate interpolations that adhere to the data's geometric structure, thus mitigating potential adverse effects associated with certain interpolations. A noteworthy aspect of this methodology is the introduction of a hyper-parameter that enables the fine-tuning of solutions across a continuous spectrum, ranging from Vanilla and classical *Mixup* setting.

Within a straightforward theoretical framework, we showed that *Local Mixup* can control the bias/variance trade-off of trained models. In more general settings, we showed that *Local Mixup* can tune a lower bound on the Lipschitz constant of the trained model. Experimentally, we used real world datasets to prove the ability of *Local Mixup* to achieve better generalization, as measured using the test error rate, than Vanilla and classical *Mixup*. Furthermore, Local Mixup demonstrates enhanced robustness against black-box attacks. We also conduct a comparative analysis with another method, AdaMixup, which enhances Mixup by respecting the data's geometric structure. Our results indicate that Local Mixup outperforms AdaMixup in terms of both performance on CIFAR10 and convergence speed.

Overall, our methodology introduces a simple way to incorporate locality notions into *Mixup*. We believe that such a notion of locality is beneficial and could be leveraged to a greater level in future work, or could be incorporated to the various *Mixup* extensions that have been proposed in the community. In future work, we would like to investigate further the choice of the graph, the choice of the hyperparameter that comes with it, and trainable versions of *Local Mixup*. Extending the theoretical results to more general contexts would definitely allow to gain further intuition on the effect of locality on *Mixup*.

CRediT authorship contribution statement

Raphael Baena: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization. **Lucas Drumetz:** Conceptualization, Methodology, Software, Validation, Formal analysis, Writing – original draft, Writing – review & editing, Visualization, Supervision, Project administration. **Vincent Gripon:** Conceptualization, Methodology, Software, Validation, Formal analysis, Resources, Writing – original draft, Writing – review & editing, Visualization, Supervision, Project administration, Funding acquisition.

Data availability

Code and data at available on github: https://github.com/raphael-baena/Local-Mixup.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.sigpro.2024.109395.

References

[1] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.
[2] H. Guo, Y. Mao, R. Zhang, Mixup as locally linear out-of-manifold regularization, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, No. 01, 2019, pp. 3714–3722.
[3] I. Loshchilov, F. Hutter, Decoupled weight decay regularization, 2017, arXiv preprint arXiv:1711.05101.
[4] P. Simard, Y. Lecun, J. Denker, B. Victorri, Transformation invariance in pattern recognition – Tangent distance and tangent propagation, Int. J. Imaging Syst. Technol. 11 (2001).
[5] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Adv. Neural Inf. Process. Syst. 25 (2012) 1097–1105.
[6] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[7] H. Zhang, M. Cisse, Y.N. Dauphin, D. Lopez-Paz, mixup: Beyond empirical risk minimization, 2017, arXiv preprint arXiv:1710.09412.
[8] P. Mangla, N. Kumari, A. Sinha, M. Singh, B. Krishnamurthy, V.N. Balasubramanian, Charting the right manifold: Manifold mixup for few-shot learning, in: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2020, pp. 2218–2227.
[9] G.S. Dhillon, P. Chaudhari, A. Ravichandran, S. Soatto, A baseline for few-shot image classification, 2019, arXiv preprint arXiv:1909.02729.
[10] C.M. Bishop, Pattern recognition, Mach. Learn. 128 (9) (2006).
[11] S. Sukhbaatar, J. Bruna, M. Paluri, L. Bourdev, R. Fergus, Training convolutional networks with noisy labels, 2014, arXiv preprint arXiv:1406.2080.
[12] R. Zhang, P. Isola, A.A. Efros, Colorful image colorization, in: European Conference on Computer Vision, Springer, 2016, pp. 649–666.
[13] T. DeVries, G.W. Taylor, Improved regularization of convolutional neural networks with cutout, 2017, arXiv preprint arXiv:1708.04552.
[14] S. Yun, D. Han, S.J. Oh, S. Chun, J. Choe, Y. Yoo, Cutmix: Regularization strategy to train strong classifiers with localizable features, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 6023–6032.
[15] E.D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, Q.V. Le, Autoaugment: Learning augmentation policies from data, 2018, arXiv preprint arXiv:1805.09501.
[16] V. Verma, A. Lamb, C. Beckham, A. Najafi, I. Mitliagkas, D. Lopez-Paz, Y. Bengio, Manifold mixup: Better representations by interpolating hidden states, in: International Conference on Machine Learning, PMLR, 2019, pp. 6438–6447.
[17] D. Hendrycks, N. Mu, E.D. Cubuk, B. Zoph, J. Gilmer, B. Lakshminarayanan, Augmix: A simple data processing method to improve robustness and uncertainty, 2019, arXiv preprint arXiv:1912.02781.
[18] J.-H. Kim, W. Choo, H.O. Song, Puzzle mix: Exploiting saliency and local statistics for optimal mixup, in: International Conference on Machine Learning, PMLR, 2020, pp. 5275–5285.
[19] H.-P. Chou, S.-C. Chang, J.-Y. Pan, W. Wei, D.-C. Juan, Remix: Rebalanced mixup, in: European Conference on Computer Vision, Springer, 2020, pp. 95–110.
[20] Z. Liu, S. Li, D. Wu, Z. Chen, L. Wu, J. Guo, S.Z. Li, AutoMix: Unveiling the power of mixup, 2021, arXiv preprint arXiv:2103.13027.
[21] J. Chen, S. Sinha, A. Kyrillidis, StackMix: A complementary mix algorithm, 2020, arXiv preprint arXiv:2011.12618.
[22] W. Yin, H. Wang, J. Qu, C. Xiong, BATCHMIXUP: Improving training by interpolating hidden states of the entire mini-batch, 2021.
[23] A. Rame, R. Sun, M. Cord, MixMo: Mixing multiple inputs for multiple outputs via deep subnetworks, 2021, arXiv preprint arXiv:2103.06132.
[24] L. Carratino, M. Cissé, R. Jenatton, J.-P. Vert, On mixup regularization, 2020, arXiv preprint arXiv:2006.06049.
[25] P.K. Gyawali, S. Ghimire, L. Wang, Enhancing mixup-based semi-supervised learning with explicit Lipschitz regularization, in: 2020 IEEE International Conference on Data Mining, ICDM, IEEE, 2020, pp. 1046–1051.
[26] K. Greenewald, A. Gu, M. Yurochkin, J. Solomon, E. Chien, k-Mixup regularization for deep learning via optimal transport, 2021, arXiv preprint arXiv:2106.02933.
[27] R. Balestriero, J. Pesenti, Y. LeCun, Learning in high dimension always amounts to extrapolation, 2021, arXiv:2110.09485.
[28] A. Krizhevsky, Learning Multiple Layers of Features from Tiny Images, University of Toronto, 2012.
[29] H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017, arXiv preprint arXiv:1708.07747.
[30] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, A.Y. Ng, Reading digits in natural images with unsupervised feature learning, 2011.
[31] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[32] G. Huang, Z. Liu, L. Van Der Maaten, K.Q. Weinberger, Densely connected convolutional networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 4700–4708.
[33] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[34] L. Wasserman, Topological data analysis, Annu. Rev. Stat. Appl. 5 (2018) 501–532.
[35] Y. Abouelnaga, O.S. Ali, H. Rady, M. Moustafa, Cifar-10: Knn-based ensemble of classifiers, in: 2016 International Conference on Computational Science and Computational Intelligence, CSCI, IEEE, 2016, pp. 1192–1195.
[36] J. Jost, J. Jost, Riemannian Geometry and Geometric Analysis, Vol. 42005, Springer, 2008.
[37] D. Kalatzis, D. Eklund, G. Arvanitidis, S.r. Hauberg, Variational autoencoders with riemannian brownian motion priors, 2020, arXiv preprint arXiv:2002.05227.