Input and Weight Space Smoothing for Semi-supervised Learning

Safa Cicek, Stefano Soatto
UCLA Vision Lab
University of California, Los Angeles, CA 90095
{safacicek,soatto}@ucla.edu

Abstract

We propose regularizing the empirical loss for semi-supervised learning by acting on both the input (data) space, and the weight (parameter) space. We propose a method to perform such smoothing, which combines known input-space smoothing with a novel weight-space smoothing, based on a min-max (adversarial) optimization. The resulting Adversarial Block Coordinate Descent (ABCD) algorithm performs gradient ascent with a small learning rate for a random subset of the weights, and standard gradient descent on the remaining weights in the same mini-batch. It is simple to implement and achieves state-of-the-art performance.

1. Introduction

In vision, we have no shortage of data, but manual annotation is costly, which has prompted interest in methods that leverage unlabeled data. In semi-supervised learning, we are given N^l labeled samples $x^l \in X^l$ with corresponding labels $y^l \in Y^l$ and N^u unlabeled samples, $x^u \in X^u$. The entire training dataset is X with cardinality $N = N^l + N^u$. For a discriminative model to exploit unlabeled data, there has to be some prior on the model parameters or on the unknown labels [5]. Such a prior can be realized through a regularization functional acting on either the parameters (weight space) or the data (input space). Both input-space regularization, or "smoothing" ([20, 12, 28, 8]) and weight-space smoothing ([18, 6]) have been shown to improve both supervised (SL) and semi-supervised learning (SSL). The first question we address is whether the two are, in some sense, equivalent. Although the two couple linearly in deep networks, composition of non-linearities complicates the analysis beyond analytical feasibility.

To answer the question, we conduct experiments that show that, for nonlinear and/or over-parametrized classifiers, input and weight smoothing are not only not equivalent, but they are complementary, suggesting that applying both may be beneficial. The second question we address, therefore, is whether this can be done efficiently to yield performance improvements relative to methods that only address one of the two.

To this end, we propose a new algorithm for weight smoothing called *Adversarial Block Coordinate Descent (ABCD)*, which we combine with a standard input-smoothing algorithm (VAT), and test the result on SSL benchmarks on the CIFAR10 and SVHN datasets. ABCD combined with VAT achieves state-of-the-art performance with minimal data augmentation (translation and reflection), without complex architectures (e.g. ResNet [16]).

While in this section our method is motivated heuristically, there are theoretical groundings for performing joint regularization in weight- and input-space, which we discuss in Section 4.2. There, we show that the two are not equivalent, and in fact are complementary, one affecting the minimality of the resulting representation, the other insensitivity to nuisance variability. In the next two subsections we describe input and weight smoothing, and in the following subsection, we show them to not be equivalent. In the next section we describe the proposed algorithm ABCD, and in the following one, we put it to the test on visual SSL benchmarks.

1.1. Input smoothing

We call a classifier "input smooth" when its predictions are robust to small perturbations in the input space. So, input smoothing can be enforced with the following optimization problem:

$$\begin{aligned} &\min_{w} \sum_{x_i \in X} \ell(f(x_i; w), f(x_i + \Delta x_i; w)) \\ &\text{subject to } \Delta x_i = \underset{||\Delta x_i|| < \epsilon_x}{\arg\max}\, \ell(f(x_i; w), f(x_i + \Delta x_i; w)) \\ &\qquad \forall x_i \in X \end{aligned} \tag{1}$$

where $\ell(\cdot)$ can be cross-entropy, Kullbach-Liebler (KL) divergence or the mean-square error. $f(x; w) \in \mathbb{R}^K$ is the network output with weights w and K is the number of classes. This problem can be solved along with minimizing the objective function designed for the task, for instance

the cross-entropy loss for classification. In other words, a desirable classifier should not change its predictions for any local perturbation within a ball of small radius ϵ_x for any input x_i. This idea is also known as max-margin or low-density assumptions in the SSL literature, championed by TSVM [20]. Although the perturbations to which we seek insensitivity are unstructured, in imaging data the largest perturbations are often due to structured nuisance variability (*e.g.*. changes in illumination, vantage point, or visibility).

A popular way of attacking this min-max problem is through the use of adversarial examples. The underlying idea is to add a (regularization) term to the loss function, that penalizes the difference between network outputs for clean samples, and samples with added adversarial noise. Adversarial training [12] applies this idea to supervised learning where they change the problem to being robust against noise by moving predictions away from the ground truth labels:

$$\begin{aligned} &\min_{w} \sum_{x_i \in X} \ell(f(x_i; w), f(x_i + \Delta x_i; w)) \\ &\text{subject to } \Delta x_i = \arg\max_{||\Delta x_i|| < \epsilon_x} \ell(P(y_i|x_i), f(x_i + \Delta x_i; w)) \\ &\forall x_i \in X \end{aligned} \tag{2}$$

For this supervised setting, ground truth labels $P(y|x)$ can be used in calculating the adversarial noise Δx. Instead of finding the exact Δx for each input x, [12] calculates the first order approximation of adversarial perturbation leading to maximum change in the classifier predictions $f(x; w)$:

$$\begin{aligned} &\Delta x \approx \epsilon_x \frac{g}{||g||_2} \\ &\text{subject to } g = \nabla_x \ell(P(y|x), f(x; w)) \end{aligned} \tag{3}$$

where $P(y|x)$ is the ground truth label for sample x. A natural extension of this idea to SSL is introduced by [29, 28]. Since SSL algorithms do not have access to ground truth labels $P(y|x)$, their adversarial noise attempts to maximize $\ell(f(x; w), f(x + \Delta x; w))$. But, in the SSL case, the first-order approximation is not useful, because the first derivative of $\ell(f(x; w), f(x + \Delta x; w))$ is always zero at $\Delta x = 0$. Hence, [29, 28] make a second-order approximation for Δx and propose the following approximation to the adversarial noise for each input x:

$$\begin{aligned} &\Delta x \approx \epsilon_x \frac{g}{||g||_2} \\ &\text{subject to } g = \nabla_{\Delta x} \ell(f(x; w), f(x + \Delta x; w))\Big|_{\Delta x = \xi d} \end{aligned} \tag{4}$$

where $d \sim N(0, 1)$. Therefore, the regularization loss of [29, 28] is

$$\begin{aligned} &\ell_{VAT}(x; w) := \ell(f(x; w), f(x + \epsilon_x \frac{g}{||g||_2}; w)) \\ &\text{subject to } g = \nabla_{\Delta x} \ell(f(x; w), f(x + \Delta x; w))\Big|_{\Delta x = \xi d} \end{aligned} \tag{5}$$

for one input sample x. We will minimize this regularizer as a way of doing input smoothing in our final SSL algorithm.

1.2. Weight smoothing

Just like input smoothing, weight smoothing can be formulated as a min-max problem. More explicitly,

$$\begin{aligned} &\min_{w} \sum_{x_i \in X} \ell(f(x_i; w), f(x_i; w + \Delta w)) \\ &\text{subject to } \Delta w = \arg\max_{||\Delta w|| < \epsilon_w} \sum_{x_i \in X} \ell(f(x_i; w), f(x_i; w + \Delta w)) \end{aligned} \tag{6}$$

Unlike input smoothing, the parameters of both minimization and maximization *are the same*, namely the weights of the network, w. It is important to note that the maximum is taken within a ball of small radius ϵ_w. This appears counter-intuitive at first.

Just like input smoothing can be associated with insensitivity to nuisance variability in the data (hence bias the representation towards invariance), weight smoothing can be associated with generalization, as it has been observed empirically that so-called "flat-minima" [18, 19] correspond to solutions that tend to yield better generalization. Recent methods [6] try to bias solutions towards such flat minima, including using a conservative penalty [26]:

$$w_t = \arg\min_{w} \ell(P(y|x); f(x; w)) + \gamma ||w - w_{t-1}||_2^2. \tag{7}$$

We follow a related, but different, approach: We explicitly find adversarial directions *with respect to random subsets of the weights,* then force the network to be robust against these directed perturbation using the remaining weights.

This problem of optimizing for the worst case is also studied in the robust optimization literature [4]. Given all the possible adversarial Δw values maximizing the loss, they suggest an optimization framework for finding descent directions with second-order cone programming (SOCP) which would guarantee an optimal solution for a convex objective. Since finding all possible adversarial perturbations is not feasible, they find the ones around a ball with gradient ascent and solve SOCP for local solutions iteratively, which is related to our method.

1.3. Input smoothing and weight smoothing do not imply each other

For a linear classifier, the Hessian of the mean-square error (MSE) loss is the data covariance matrix. The local ge-

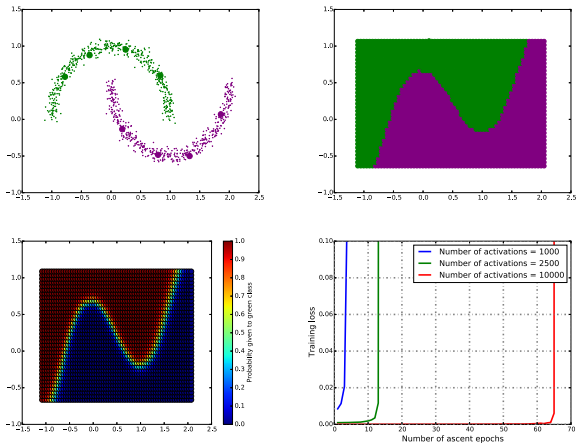

Figure 1. **Over-parametrized networks are more robust to adversarial noise in the weight space even when they have the same decision boundary (i.e. the same input smoothness).** Three MLP networks with different number of hidden units trained with VAT on the half-moon dataset (first panel) have the same decision boundary (second panel). Moreover, the network response (probability given to one of the classes) is almost identical for each of the three networks (third panel). Therefore, the robustness of the networks to input perturbation is not just same in the "error" sense but also the same in the "loss" sense. However, the larger the network, the more robust it is to adversarial perturbations in weight space. The Fourth panel shows the training loss versus the number of gradient ascent directions for varying sized networks. As it can be seen, having a visible increase in the loss takes more ascent steps for larger networks. This experiment illustrates that networks with the same robustness to input perturbations may have completely different sensitivity to perturbations in the weight space.

ometry of the loss landscape around the solution where the weights converged does not depend on the classifier structure, nor on its parametrization. The number of zero eigenvalues is determined by the dimensionality of the input data matrix alone. However, it can be easily shown that this is not necessarily the case for nonlinear classifiers. For instance, [34] shows that the number of zero eigenvalues of the approximate Hessian of the loss has a lower bound of $|w| - N$, where N is the number of training samples and $|w|$ is the number of weights. This simple Hessian argument suggests that the robustness of a network to weight perturbations depends on factors other than its robustness to input perturbations, like the number of parameters in the network, for nonlinear classifiers. Even though they also have empirical evidence for this claim, the spectrum of the Hessian alone is not necessarily a good measure of flatness [10]. So, we construct a counterexample to verify that input smoothness does not necessarily imply weight smoothness.

We use a half-moon toy dataset whereby there are 4 labeled (larger circles) and 1000 unlabeled samples from each cluster (first panel of Fig. 1). We run the VAT algorithm to find the max-margin decision boundary (second panel of Fig. 1). We repeat this experiment for 3 multi-layer perceptron (MLP) networks each having the same structure [1], except for a different number of weights. The decision boundary is the same, as it can be seen in the second panel of Fig. 1, where they show as one as they overlap perfectly. This implies that for any perturbation in the input space, the increase in the error would be the same for each of these classifiers. To show that this also the case for the loss, we also provide the network response and again they are indistinguishable (third panel of Fig. 1). Hence, input smoothness is the same for each of three MLPs.

After training three MLP networks with VAT, we apply gradient ascent on the converged weights with a small learning rate to evaluate the robustness of the networks with different number of weights. As it can be seen in the right panel of Fig. 1, it takes more ascent updates for large networks to diverge.[2] Thus, over-parametrized networks are more robust to perturbations in weight space. This experiment shows that input smoothing does not necessarily imply weight smoothing. That is, there are factors other than input smoothness determining the geometry of the loss around the converged weight. Another observation is that losses diverge suddenly, implying that it takes several iterations to increase the loss, during which time the landscape is almost constant, before the loss increases sharply.[3] The ascent learning rate chosen in the experiment is 0.005. During ascent, SGD without momentum and weight decay is used. MLPs are in the form $2 \rightarrow FC(n) \rightarrow FC(n) \rightarrow FC(n) \rightarrow FC(n) \rightarrow 2$ where $FC(n)$ is fully connected layer followed by a RELU and n is number of hidden units. We report results for $n = 1000$, $n = 2500$ and $n = 10000$.

Note that we choose to make our experiments in two-dimensional inputs to visualize the decision boundary easily and verify that they are the same for different classifiers; not because these results are restricted to small dimensional in-

[1] By "the same structure" we mean same number and type of layers with different number of filters. So networks have the same depth, but different widths.

[2] Number of small ascent steps it takes for loss to diverge is not the only measure for flatness of the converged minima. In Section 3.2, the same experiment is repeated with different measures. For instance, by plotting the change in the loss for various values of one ascent step norm.

[3] This plot implies that for very large networks and small dimensional inputs, reaching to a considerably high loss level set may require many ascent iterations. This might seem counter-intuitive when we consider a typical Hessian histogram of a deep network weights as there are usually few large positive eigenvalues. But, it is important to note that the MLP networks we use in this experiment have 4 hidden layers and the largest of them has 10000 units per layer. So, the size of the network we use for this experiment is much larger than those for which Hessian histograms can be calculated. Unfortunately, calculating the Hessian of such a large network is very expensive computationally. Similarly, it can take many descent iterations to reach a low-level loss if the data is too complex for the model. For instance, when training with random labels [7], it takes many epochs to reduce the loss level even though the loss goes to zero eventually [47].

puts. These results align with those of [11] where they suggest that the amount of uphill climbing for connecting arbitrary weight pairs is correlated to the size of the network. For more discussion on the effect of over-parametrization on the loss landscape of DNNs see [33, 35, 2, 40].

1.4. Joint input and weight smoothing

Once established that input smoothness and weight smoothness are not equivalent, it is natural to try enforcing both. So, the additional regularization we want to have can be framed as follows:[4]

$$\begin{aligned}
&\min_{w} \sum_{x_i \in X} \ell(f(x_i; w), f(x_i; w + \Delta w)) + \\
&\quad \ell(f(x_i; w), f(x_i + \Delta x_i; w)) \\
&\text{subject to } \Delta w = \arg\max_{||\Delta w|| < \epsilon_w} \sum_{x_i \in X} \ell(f(x_i; w), f(x_i; w + \Delta w)) \\
&\quad \Delta x_i = \arg\max_{||\Delta x_i|| < \epsilon_x} \ell(f(x_i; w), f(x_i + \Delta x_i; w)) \\
&\quad \forall x_i \in X
\end{aligned} \tag{8}$$

So, the regularization term we want to minimize for one input x is

$$\begin{aligned}
&L(x; w) = \ell(f(x; w), \\
&f(x; w + \arg\max_{||\Delta w|| < \epsilon_w} \sum_{x_i \in X} \ell(f(x_i; w), f(x_i; w + \Delta w)))) \\
&+ \ell(f(x; w), f(x + \arg\max_{||\Delta x|| < \epsilon_x} \ell(f(x; w), f(x + \Delta x; w)); w))
\end{aligned} \tag{9}$$

Then, the overall problem we want to solve in the supervised learning setting becomes

$$\min_{w} \sum_{x} \ell_{CE}(x; w) + \lambda L(x; w) \tag{10}$$

where

$$\ell_{CE}(x; w) = -\langle P(y|x), \log f(x; w) \rangle \tag{11}$$

is the cross entropy loss calculated for label estimates $f(x; w)$ and ground truth labels $P(y|x)$.

Calculating the exact Δx for each input is not trivial. Instead, we will use VAT in Eq. 4 to make our classifier robust against input space perturbations. For achieving weight space smoothness, we will use the proposed algorithm described next.

Algorithm 1 Adversarial Block Coordinate Descent (ABCD)

Input: Minibatch set B_t, loss function $\ell(\cdot)$, initial weights w_0.
Hyper-parameters: Ascent and descent learning rates η_A and η_D. Number of inner iterations L.
Outputs: Final weights w_L.
for $l = 1 : L$ **do**
  Γ_i sample from $\{0, -1\}$ for all $i \in \{1, \ldots, |w_0|\}$.
  $\Gamma_i^a = \Gamma_i$ for all $i \in \{1, \ldots, |w_0|\}$.
  $\Gamma_i^d = \Gamma_i + 1$ for all $i \in \{1, \ldots, |w_0|\}$.
  // Run stochastic gradient *ascent* with a *small* learning rate η_A
  $w_{l-\frac{1}{2}} = w_{l-1} - \eta_A \Gamma^a \odot \nabla_{w_{l-1}} \left(\frac{1}{|B_t|} \sum_{i=1}^{|B_t|} \ell(x_i; w_{l-1}) \right)$
  // Run stochastic gradient *descent* with a learning rate $\eta_D \gg \eta_A$
  $w_l = w_{l-\frac{1}{2}} - \eta_D \Gamma^d \odot \nabla_{w_{l-\frac{1}{2}}} \left(\frac{1}{|B_t|} \sum_{i=1}^{|B_t|} \ell(x_i; w_{l-\frac{1}{2}}) \right)$

2. Adversarial Block Coordinate Descent (ABCD)

Since the parameters of both minimization and maximization are the weights of the network for the min-max problem in Eq. (6), we use a subset of the weights w for finding adversarial directions in weight space and the rest to impose robust to such additive adversarial perturbations in weight space. For this, at each update, we randomly choose half of the weights and take a gradient *ascent* step along them with a *small* learning rate. Then, on the same batch, we apply gradient descent for the remaining weights with ordinary (larger) learning rate. We call this algorithm Adversarial Block Coordinate Descent (ABCD), for pseudo-code, see Alg. 1. ABCD can be considered as an extension of Dropout [17, 42] and coordinate descent [45]. However, unlike Dropout, we explicitly regularize our network to be robust against "adversarial directions" in the weight space; instead of just being robust to zeroed out weights.

ABCD can be used for SSL in place of vanilla SGD. We use ABCD for SSL by minimizing the empirical cross entropy for labeled data and the entropy of empirical estimates for the unlabeled data with ABCD. That is,

$$\ell_E(x; w) = -\langle f(x; w), \log f(x; w) \rangle \tag{12}$$

for unlabeled data which is a well-known regularizer in the SSL literature [9, 14, 22, 41].

The randomness in ABCD is due to mini-batch optimization that we have to use for computational reasons and the randomness in the mask Γ selection. If we ignore these, it would be easier to see the loss minimized by ABCD. The loss minimized by descent in ABCD is the "worst case" of the nominal loss landscape. By worst case we mean that, at

[4] Note that here we take each smoothness term separately. I.e. adversarial perturbations in weight space Δw are calculated for the original images x; not for perturbed images $x + \Delta x$. Even though it is possible to formulate the problem such that perturbations in the weight space (Δw) are functions of perturbations in the input space (Δx), problem would be even more complicated in that case.

Algorithm 2 SSL algorithm using ABCD as optimizer; VAT and entropy as regularizers. $\ell_{CE}(x;w)$, $\ell_E(x;w)$, $\ell_{VAT}(x;w)$ are as defined in Eq. 11, Eq. 12, Eq. 5 respectively.

for $t = 1 : T$ **do**
  // Run ABCD on cross entropy for weight smoothing:
  Sample B_t^l // labeled samples
  $w_{t'} = ABCD(B_t^l, \ell_{CE}(x;w), w_{t-1})$
  // Run ABCD on entropy for weight smoothing:
  Sample B_t^u // unlabeled samples
  $w_{t'} = ABCD(B_t^u, \ell_E(x;w), w_{t'})$
  // Run SGD on VAT loss for input smoothing:
  $w_t = SGD(B_t^u, \ell_{VAT}(x;w), w_{t'})$

each point in the weight space, the loss ABCD minimizes is the maximum that can be reached from that point with one small ascent step. In other words, ABCD minimizes the maximum loss around a small ball at each point.

Finding the minimum norm adversarial perturbation defined in Eqn. 6, is not possible with one ascent step as this problem is highly non-convex. Therefore, we take $L > 1$ steps for each mini-batch. Taking multiple ascent steps does not guarantee to find the minimum-norm adversarial perturbation, but empirically we find that it gives better results. Moreover, we do not want the last update for any of the weight parameter to be ascent. So, we do not apply ABCD in the last few epochs.

Also, note that the coordinate descent mask is essential for ABCD to work as intended. Assume that true adversarial perturbation Δw in Eqn. 6 is given. Then, for minimizing the objective in Eqn. 6 locally is all we need to do is apply the original SGD update. This is because the first order approximation of Δw is just the negative direction of the gradient given by the SGD. So, without block coordinate descent masks, the result of adversarial training in weight space would only be the change of effective learning rate. That is why we have dropout-like masks where half of the weights are trained to be robust when others add adversarial noise. Moreover, instead of first order method, using a second-order approximation like VAT in finding Δw for half of the weights and minimizing the objective in Eqn. 6 with the rest is possible. But, we choose to calculate Δw with the first order gradients for faster training. This is done by ascending in cross entropy and entropy for labeled and unlabeled cases respectively.

We report the performance of ABCD combined with VAT to see the effect of applying both input and weight smoothing. The pseudo-code using VAT as loss function and ABCD as an optimizer for SSL task is given in Alg. 2. For labeled data, ABCD used as an optimizer to minimize cross entropy to achieve weight smoothing. We do not minimize $\ell_{VAT}(x;w)$ for labeled data as proposed in the corresponding paper. For unlabeled data, ABCD is used to minimize entropy $\ell_E(x;w)$ and SGD is used to minimize $\ell_{VAT}(x;w)$ for weight and input smoothing respectively. We set $\eta_A = 10^{-5}$ for all of our experiments. η_D can be chosen as usual with a high initial value from $\{0.1, 0.01\}$ and is decreased during training. In all the SSL experiments, we only use the network called conv-large from [28, 43]. Only translation and horizontal flipping are used as data augmentations to allow a fair comparison with some of the previous SSL algorithms. Horizontal flipping is only used in CIFAR10. Cosine learning schedule of [27] is used for annealing the learning rate and momentum. Initial learning rates for SVHN and CIFAR10 with ZCA are 0.1 and 0.04 respectively. We decrease learning rate up to final learning rate of $\eta_D = 10^{-4}$. In CIFAR10 task, we apply ZCA. Before applying ZCA, we standardize the dataset. ZCA transformation is $D_{\text{ZCA}} = DU(S^{-0.5} + \epsilon I)U^T$ where S and U are eigenvalue and eigenvector matrices of covariance of data matrix D respectively.

3. Evaluation

3.1. Performance of ABCD in SSL Benchmarks

In Table 1,2 we compare the performance of ABCD with state-of-the-art SSL algorithms. We report performance of the proposed algorithm on SVHN [30] and CIFAR10 [23] in SSL setting. SVHN consists of 32×32 images of house numbers. We use $73,257$ samples for training, rather than the entire $600,000$ images; $26,032$ images are separated for evaluation. CIFAR10 has $60,000$ 32×32 images, of which $50,000$ are used for training and $10,000$ for testing. We choose labeled samples randomly. We also choose them to be uniform over the classes as it is done in previous works [28]. In CIFAR10, $4,000$ and in SVHN $1,000$ of training samples are labeled. Except [36], all the methods use modest augmentations (translation and horizontal flipping) and do not exploit recent deep learning models like ResNet [15]. We report ABCD with entropy minimization alone and combined with VAT. In CIFAR10, when we run ABCD only with entropy minimization, it yields better performance than previous ensemble methods [24, 43]. Combining ABCD with VAT improves the scores in both datasets verifying that they are complementary.

In our implementation of VAT [28], we set ϵ_x in Eq. 4 to be 128 for CIFAR10 and 0.25 for SVHN. Even though we search ϵ_x in a fine grid, we could not get to the performance reported in their paper for CIFAR10; possibly because we use a different optimizer (SGD instead of ADAM), different whitening and different learning rate scheme. The performance of our VAT implementation is given in Table 1,2. Our implementation (13.01%) of VAT without entropy minimization is about 1.5 percent worse than what is reported in [28] (11.36%) for CIFAR10. But, we still use the numbers reported in [28] for comparison purposes in Table 1,2.

SSL Method	Test error rate (%)
VAT+EntMin [28]	10.55
Stochastic Transformation [36]	11.29
Temporal Ensemble [24]	12.16
GAN+FM [37]	15.59
Mean Teacher [43]	12.31
VAdD [31]	**9.22**
EntMin (*)	15.29 ± 0.23
VAT without EntMin (*)	13.01 ± 0.14
VAT+EntMin (*)	11.63 ± 0.12
ABCD+EntMin	11.96 ± 0.06
ABCD+EntMin+VAT	9.28 ± 0.21

Table 1. **Comparison with the state-of-the-art on CIFAR10 SSL task.** Error rates on the test set are given for CIFAR10. CIFAR10 is trained using 4,000 labeled and 46,000 unlabeled samples. Results are averaged over three random labeled sets. We report performance of ABCD alone and combined with VAT. ABCD+EntMin+VAT refers to algorithm in Alg. 2 where ABCD is used as an optimizer; entropy and VAT are used as regularizers in the loss function. ABCD+EntMin uses only entropy for unlabeled data to report performance of ABCD without VAT. **SSL baselines.** Baseline algorithms are EntMin, VAT and VAT+EntMin. EntMin minimizes the entropy of estimates for unlabeled data with standard SGD. Similarly, VAT minimizes ℓ_{VAT} from Eq. 5 and VAT+EntMin minimizes both on unlabeled data. Note that (*) means our own implementation.

SSL Method	Test error rate (%)
VAT+EntMin [28]	3.86
Stochastic Transformation [36]	NR
Temporal Ensemble [24]	4.42
GAN+FM [37]	5.88
Mean Teacher [43]	3.95
VAdD [31]	3.55
EntMin (*)	5.68 ± 0.03
VAT without EntMin (*)	5.36 ± 0.22
VAT+EntMin (*)	4.01 ± 0.07
ABCD+EntMin	4.52 ± 0.15
ABCD+EntMin+VAT	$\mathbf{3.53 \pm 0.24}$

Table 2. **Comparison with the state-of-the-art on SVHN SSL task.** Same as Table 1 except results are given for SVHN. NR stands for "not reported." SVHN is trained using 1,000 labeled and 72,257 unlabeled samples. Proposed algorithm achieves the state-of-the-art performance on SVHN SSL task.

3.2. Robustness of ABCD to weight perturbations

First, we report Hessian histograms to see the curvature of the point converged by ABCD. We calculate the histograms of Hessian spectra for random weights, SGD-trained weights and ABCD-trained weights in Fig. 2.

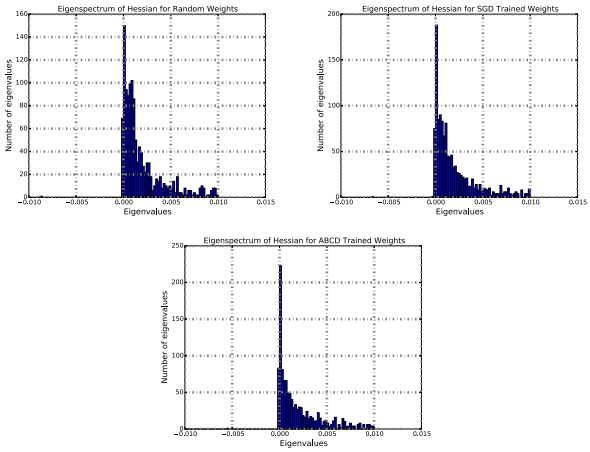

Figure 2. **Hessian spectra** of 1 hidden layer convolutional network with 1476 parameters for the loss calculated in MNIST. Left panel is for random weights, middle panel is for SGD trained weights and the right panel is for ABCD trained weights. The number of almost-zero eigenvalues is maximum for ABCD-trained networks. Histograms are cropped for eigenvalues in $[-0.01, 0.01]$.

We use the automatic differentiation package of PyTorch [32] for calculating Hessians. The number of eigenvalues smaller than 10^{-4} for random weights, SGD trained weights and ABCD weights are 185, 226, 262 respectively. So, the number of almost-zero eigenvalues for ABCD-trained weight network is larger than that of SGD-trained weights. This implies that ABCD converges to wider minima than SGD. A small convolutional network with one hidden layer is trained on MNIST dataset for this experiment due to computational constraints.

To evaluate the robustness of ABCD-trained weights to adversarial weight space perturbations, we report the number of ascent updates necessary for the loss to diverge. In Fig. 3 (left), the plot of training loss v.s. the number of ascent updates is given for SGD and ABCD-trained weights. As can be seen, the number of ascent updates needed for the loss to diverge for ABCD is more than for SGD.

Another way of comparing the local geometrical properties of the weights is by visualizing the loss landscape around the converged weights. As the deep networks we use have very high dimension, several visualization tricks have been suggested to visualize the loss landscapes in 1- or 2-dimensional subsets of the weight space [25, 13, 21]. We employ the technique suggested in [13] in Fig. 3 (right): we plot the loss on the curve $\alpha * w_{SGD} + (1-\alpha) * w_{ABCD}$ where $\alpha \in [-0.2, 1.2]$. w_{SGD} and w_{ABCD} are the weights converged when training with SGD and ABCD respectively. As it can be seen, the loss does not increase around ABCD trained weights. To be consistent with Hessian experiments, these experiments are also conducted on MNIST dataset with the same network.

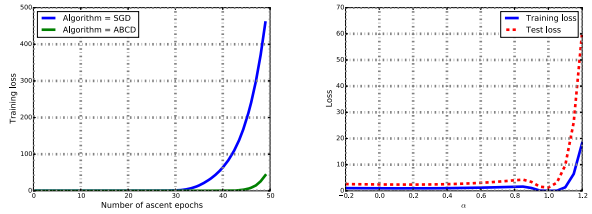

Figure 3. **Robustness of ABCD vs SGD trained networks to ascent updates**. Starting from the ABCD and SGD trained weights, we apply gradient ascent with learning rate 0.001 on the cross entropy loss. This plot shows the increase in the training loss versus the number of ascent steps. Both weights diverge quickly, but ABCD trained network diverges later than that of SGD verifying the robustness of ABCD to weight space perturbations. **1D visualization of loss landscapes of SGD and ABCD trained weights**. Training and test losses over the curve $\alpha * w_{SGD} + (1-\alpha) * w_{ABCD}$ are given as suggested by [13] where $\alpha \in [-0.2, 1.2]$. $\alpha = 0$ and $\alpha = 1$ correspond to the weights of ABCD and SGD trained networks respectively. This method compares the flatness of two methods in one direction only and in that direction, ABCD seems much more robust to weight perturbations.

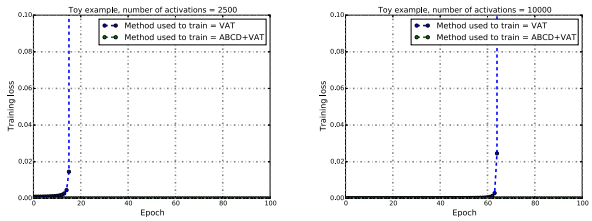

Figure 4. **ABCD vs VAT+ABCD-trained networks for the toy example in Fig. 1:** VAT+ABCD trained network is much more robust to weight perturbations than VAT-trained network for all sizes of networks. Here, we only report results for two networks of the same type: one with 2500 filters (left), another with 10000 filters (right).

In the toy example of Fig. 1, we compared the robustness of different sized networks to weight perturbations. Now, we introduce ABCD, we can compare VAT-trained networks with VAT+ABCD trained networks in the same setting. As it can be seen in Fig. 4, VAT+ABCD trained network diverges later than VAT-trained network for two different sizes of networks. We do not plot ABCD because ABCD alone is not enough to converge to 0 training loss. This is because perfect classification in SSL setting of half-moon clusters requires a regularizer encouraging input smoothing. Since ABCD and VAT trained networks are not in the same loss level, weight smoothness comparison between them is not straightforward. Also, note that the experiment in Fig. 1 is in the SSL setting where there are 4 labeled samples from each class. Hence, there is no way for SGD to exploit the unlabeled data. So, we do not include SGD into the comparison.

In Fig. 1 and 3-left, we measure the robustness by comparing the losses for the increasing number of small ascent steps. In Fig. 5-left, we repeat the experiment in Fig. 1 with a different criterion for measuring the robustness of algorithms to weight perturbations. This time, we plot the loss after one ascent step for various ascent learning rates. When we compare losses for different learning rate of one ascent steps, over-parametrized networks are again more robust. Similarly, in Fig. 5-right, we repeat Fig. 3-left with this criterion and again ABCD trained network is more robust to weight perturbations than SGD. So, whether we take the number of ascent steps or the length of one ascent step to diverge as a measure of flatness, ABCD-trained networks systematically find wider solutions.

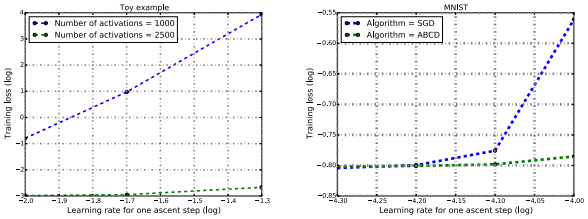

Figure 5. **Different robustness criteria for the experiments in Fig. 1 and 3-left.** Here, we plot loss vs. learning rate of one ascent step. Over-parameterized networks are again more robust to perturbations in the toy example (left). ABCD trained network in Fig. 3-left is also more robust than SGD trained network (right).

4. Related Work and Discussion

4.1. Related Work

Next, we discuss our contribution in the context of related and recent work in SSL. **Input smoothing in SSL.** In addition to work referenced earlier, there are graph-based methods ([39, 46, 49]) that penalize having different labels for similar input pairs. For instance, given that s_{ij} is the measure of similarity for input samples x_i and x_j, they minimize the energy $s_{ij}(f(x_i;w) - f(x_j;w))^2$ performing label propagation under the constraint of fitting to labeled data. This forces the discriminant to change little in response to different inputs with large s_{ij}. The generative model by [41] suggests that they maximize the margin with the help of fake samples.

Weight smoothing in SSL. Teacher-student methods [24, 43] average over many predictions or weights in a way that the teacher network can attract student networks towards itself. A similar algorithm is suggested by [48] for parallel computing under communication constraint where each replica is attracted to the reference system. [3] studies such algorithms for models with discrete variables and they argue that they find robust local minima. Thus, one can relate the success of teacher-student models of state-of-the-art deep SSL algorithms for their ability to converge to robust weights. Recent work of [31] also combines VAT

with Virtual Adversarial Dropout (VAdD) and, like us, improve upon the VAT baseline as a result. VAdD finds a zero mask of dropout adversarially at each update rather than trying to be robust against adversarial "directions". VadD is not motivated with the flat-minima [18] and there is no experiment supporting whether it is able to find wide valleys. In this work, we establish that input smoothing and weight smoothing are complementary and applying both is useful for SSL tasks. We achieve state-of-the-art results by combining VAT of [28] with the proposed algorithm ABCD. We conduct many experiments to verify that ABCD-trained networks are indeed robust against adversarial perturbations in the weight space.

Effect of noise on generalization. Adding random noise to gradients is known to improve the generalization [44]. [19] analyzes the effect of the inherent noise due to mini-batch usage in the properties of point converged by SGD. They conclude that for larger noise, the network favors wider minima under the assumption that the noise is isotropic. ABCD differs from these works by adding adversarial noise to weights instead of random noise.

4.2. Theoretical Grounding

Our algorithm yields a function (discriminant) $f(x;w)$ that maps a (test) datum x onto a class label y. More precisely, f maps x onto a vector $f(x;w) \in \mathbb{R}^K$ with K the number of labels, which can be evaluated for each class hypothesis $y = k \in \{1, \dots, K\}$ to yield a positive real number $f(x;w)[k]$ where $[k]$ denotes the k-th component. Ideally, such a discriminant would be the posterior probability $f(x;w)[k] = P(y = k|x)$, which is the (Bayesian) optimal discriminant. If we could compute and minimize the *expected* cross-entropy loss $\ell_{CE}(x;w)$ within a universally approximating family of functions $f(\cdot;w)$, we would obtain just that. Unfortunately, we only have a finite sample of the true distribution $\mathcal{D} \doteq \{(x,y) \sim p(x,y)\}_{i=1}^N$, from which we can only compute and minimize the *empirical* cross entropy $\ell_{CE}^{\mathcal{D}}(x;w)$. Nevertheless, we can construct a proxy loss that captures the properties of the optimal discriminant even with a finite sample, as we discuss next.

The posterior $P(y|x)$ is a function of the test datum that is *minimal, sufficient and invariant* [38]. It is sufficient (informationally equivalent to the data) for the task of classification; it is invariant to nuisance variability in the data; and it is minimal in the sense of having smallest (information) complexity, which relates to generalization [1]. On the other hand, the learned discriminant $f(\cdot;w)$ is a function of the dataset $\mathcal{D}$ that does not know anything about the test datum x. If, however, we could construct f to be any minimal, sufficient, and invariant function of the *training set* $\mathcal{D}$, constructed with a deep neural network, then the Emergence Theory of [1] guarantees that the resulting discriminant is also a minimal sufficient invariant of the test datum. The key question then is: *What is the loss function that is computable from the training set that would yield a minimal, sufficient invariant?*

Sufficiency is captured by empirical cross entropy: Any function that minimizes it (possibly by overfitting) is a sufficient representation of the training set.

Invariance is captured by minimizing the sensitivity to nuisance variability in the data. This can be done by considering the (worst-case) perturbations that do not modify the class (hence they are, by definition, nuisance factors). This is precisely the criterion we have used to *define* input-space smoothing.

Minimality is measured *not* by the dimension of the weights, but by their *information content*. While in the Emergence theory this is measured in the sense of Shannon [1], measuring information in the sense of Fisher yields, under suitable approximation, the second-order criterion. As we verify with our Hessian histograms, the proposed method finds flat minima in this sense.

Thus, although derived heuristically, the loss function we construct by imposing both input and space smoothing, in addition to the small empirical loss, captures precisely the three defining properties of the optimal (Bayesian) discriminant, grounded in recent theoretical work [1]. Of course, the approximation of the optimal discriminant is only as good as the given data, so there is no (finite-sample) guarantee on the approximation of the posterior, but at least the loss function we adopt better captures the properties of the posterior than the raw empirical loss, which affords no invariance and no minimality.

4.3. Conclusion

We propose an adversarial training algorithm for SSL tasks to be robust against adversarial perturbations in both input and weight spaces where we combine the known input smoothing algorithm with a novel weight smoothing algorithm. We establish with a toy example that input and weight smoothing are complementary. In SSL benchmarks, combining input and weight smoothing algorithms resulted in a performance better than applying either algorithms alone. This further verifies that smoothing in input and weight spaces are complementary for high dimensional data as well. As a result, the proposed algorithm achieves competitive results in semi-supervised learning benchmarks. Furthermore, we conduct extensive experiments proving the robustness of the proposed algorithm to the adversarial perturbations in weight space.

Acknowledgment

Research supported by ONR - N00014-17-1-2072 and ARO MURI - W911NF-17-1-0304.

References

[1] A. Achille and S. Soatto. On the emergence of invariance and disentangling in deep representations. *arXiv preprint arXiv:1706.01350*, 2017. 8

[2] M. Baity-Jesi, L. Sagun, M. Geiger, S. Spigler, G. B. Arous, C. Cammarota, Y. LeCun, M. Wyart, and G. Biroli. Comparing dynamics: Deep neural networks versus glassy systems. *arXiv preprint arXiv:1803.06969*, 2018. 4

[3] C. Baldassi, C. Borgs, J. T. Chayes, A. Ingrosso, C. Lucibello, L. Saglietti, and R. Zecchina. Unreasonable effectiveness of learning neural networks: From accessible states and robust ensembles to basic algorithmic schemes. *Proceedings of the National Academy of Sciences*, 113(48):E7655–E7662, 2016. 7

[4] D. Bertsimas, O. Nohadani, and K. M. Teo. Robust optimization for unconstrained simulation-based problems. *Operations research*, 58(1):161–178, 2010. 2

[5] O. Chapelle, B. Scholkopf, and A. Zien. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009. 1

[6] P. Chaudhari, A. Choromanska, S. Soatto, and Y. LeCun. Entropy-sgd: Biasing gradient descent into wide valleys. *arXiv preprint arXiv:1611.01838*, 2016. 1, 2

[7] S. Cicek, A. Fawzi, and S. Soatto. Saas: Speed as a supervisor for semi-supervised learning. In *The European Conference on Computer Vision (ECCV)*, September 2018. 3

[8] S. Cicek and S. Soatto. Unsupervised domain adaptation via regularized conditional alignment. *arXiv preprint arXiv:1905.10885*, 2019. 1

[9] Z. Dai, Z. Yang, F. Yang, W. W. Cohen, and R. Salakhutdinov. Good semi-supervised learning that requires a bad gan. *arXiv preprint arXiv:1705.09783*, 2017. 4

[10] L. Dinh, R. Pascanu, S. Bengio, and Y. Bengio. Sharp minima can generalize for deep nets. *arXiv preprint arXiv:1703.04933*, 2017. 3

[11] C. D. Freeman and J. Bruna. Topology and geometry of half-rectified network optimization. *arXiv preprint arXiv:1611.01540*, 2016. 4

[12] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014. 1, 2

[13] I. J. Goodfellow, O. Vinyals, and A. M. Saxe. Qualitatively characterizing neural network optimization problems. *arXiv preprint arXiv:1412.6544*, 2014. 6, 7

[14] Y. Grandvalet and Y. Bengio. Semi-supervised learning by entropy minimization. In *Advances in neural information processing systems*, pages 529–536, 2005. 4

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 5

[16] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016. 1

[17] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. 4

[18] S. Hochreiter and J. Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997. 1, 2, 8

[19] S. Jastrzebski, Z. Kenton, D. Arpit, N. Ballas, A. Fischer, Y. Bengio, and A. Storkey. Three factors influencing minima in sgd. *arXiv preprint arXiv:1711.04623*, 2017. 2, 8

[20] T. Joachims. Transductive inference for text classification using support vector machines. In *ICML*, volume 99, pages 200–209, 1999. 1, 2

[21] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016. 6

[22] A. Krause, P. Perona, and R. G. Gomes. Discriminative clustering by regularized information maximization. In *Advances in neural information processing systems*, pages 775–783, 2010. 4

[23] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009. 5

[24] S. Laine and T. Aila. Temporal ensembling for semi-supervised learning. *arXiv preprint arXiv:1610.02242*, 2016. 5, 6, 7

[25] H. Li, Z. Xu, G. Taylor, and T. Goldstein. Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913*, 2017. 6

[26] M. Li, T. Zhang, Y. Chen, and A. J. Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 661–670. ACM, 2014. 2

[27] I. Loshchilov and F. Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016. 5

[28] T. Miyato, S.-i. Maeda, M. Koyama, and S. Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *arXiv preprint arXiv:1704.03976*, 2017. 1, 2, 5, 6, 8

[29] T. Miyato, S.-i. Maeda, M. Koyama, K. Nakae, and S. Ishii. Distributional smoothing with virtual adversarial training. *arXiv preprint arXiv:1507.00677*, 2015. 2

[30] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5, 2011. 5

[31] S. Park, J.-K. Park, S.-J. Shin, and I.-C. Moon. Adversarial dropout for supervised and semi-supervised learning. *arXiv preprint arXiv:1707.03631*, 2017. 6, 7

[32] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017. 6

[33] I. Safran and O. Shamir. On the quality of the initial basin in overspecified neural networks. In *International Conference on Machine Learning*, pages 774–782, 2016. 4

[34] L. Sagun, U. Evci, V. U. Guney, Y. Dauphin, and L. Bottou. Empirical analysis of the hessian of over-parametrized neural networks. *arXiv preprint arXiv:1706.04454*, 2017. 3

[35] L. Sagun, V. U. Guney, G. B. Arous, and Y. LeCun. Explorations on high dimensional landscapes. *arXiv preprint arXiv:1412.6615*, 2014. 4

[36] M. Sajjadi, M. Javanmardi, and T. Tasdizen. Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In *Advances in Neural Information Processing Systems*, pages 1163–1171, 2016. 5, 6

[37] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016. 6

[38] S. Soatto and A. Chiuso. Visual representations: Defining properties and deep approximations. *arXiv preprint arXiv:1411.7676*, 2014. 8

[39] J. Solomon, R. Rustamov, L. Guibas, and A. Butscher. Wasserstein propagation for semi-supervised learning. In *International Conference on Machine Learning*, pages 306–314, 2014. 7

[40] D. Soudry and Y. Carmon. No bad local minima: Data independent training error guarantees for multilayer neural networks. *arXiv preprint arXiv:1605.08361*, 2016. 4

[41] J. T. Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks. *arXiv preprint arXiv:1511.06390*, 2015. 4, 7

[42] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014. 4

[43] A. Tarvainen and H. Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. 2017. 5, 6, 7

[44] M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 681–688, 2011. 8

[45] S. J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015. 4

[46] Z. Yang, W. W. Cohen, and R. Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*, 2016. 7

[47] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016. 3

[48] S. Zhang, A. E. Choromanska, and Y. LeCun. Deep learning with elastic averaging sgd. In *Advances in Neural Information Processing Systems*, pages 685–693, 2015. 7

[49] X. Zhu, Z. Ghahramani, and J. D. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pages 912–919, 2003. 7