SaaS: Speed as a Supervisor for Semi-supervised Learning

Safa Cicek, Alhussein Fawzi and Stefano Soatto

University of California, Los Angeles
{safacicek,fawzi,soatto}@ucla.edu

Abstract. We introduce the SaaS Algorithm for semi-supervised learning, which uses learning speed during stochastic gradient descent in a deep neural network to measure the quality of an iterative estimate of the posterior probability of unknown labels. Training speed in supervised learning correlates strongly with the percentage of correct labels, so we use it as an inference criterion for the unknown labels, without attempting to infer the model parameters at first. Despite its simplicity, SaaS achieves competitive results in semi-supervised learning benchmarks.

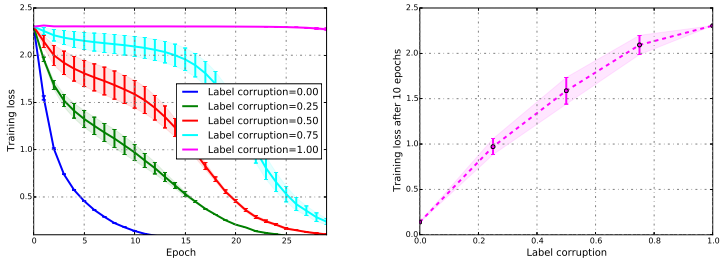

Fig. 1. Supervision quality affects learning speed. During training, the loss decreases rapidly when most labels provided are correct, and slows down significantly as the percentage of correct labels decreases. The left plot shows the loss when training a Resnet18 on CIFAR10 for different percentages of corrupted labels. The error bars show mean and standard deviation over 3 runs with random initial weights. The right panel shows the loss as a function of the percentage of incorrect labels for a unit of time corresponding to ten epochs. All the results use a fixed learning rate of 0.1, with no data augmentation or weight decay.

1 Introduction

The key idea of our approach is to *use speed of convergence as an inference criterion for the value of the unknown labels* for semi-supervised learning (SSL). Fig. 1 explicitly shows the relation between label corruption and training speed.

In SSL, one is given some labeled and some unlabeled data to train (infer the parameters of) a classifier, in hope of it performing better than if trained on the labeled data alone [1]. This is an important problem in vision where annotations are costly but unlabeled data are aplenty.

To measure learning speed, we use a small number of epochs as the unit of time, and compute the decrease of the loss in that interval when following a standard optimization procedure (stochastic gradient descent or Langevin dynamics). The main idea of our SSL algorithm is then to optimize the labels of unlabeled data (or more precisely, the posterior distribution of the unlabeled data) to maximize the loss decrease.

The resulting SaaS Algorithm is composed of an *outer loop*, which updates the distribution of unknown labels, and an *inner loop* which simulates the optimization procedure over a small number of epochs.

The proposed algorithm is unusual, as the posterior distribution of the unknown labels is initially inferred *independently of the model parameters* (weights), rather than along with them as customary in SSL.

Despite its simplicity, SaaS achieves competitive results, reported in Sect. 3. In the next section we formalize our method, and in Sect. 4 we discuss our contributions in relation to the prior art, and highlight its features and limitations.

1.1 Description of the method

We are given some labeled data $\mathbf{x}^l \doteq \{x_i^l\}_{i=1}^{N^l}$ with labels $\mathbf{y}^l \doteq \{y_i^l\}_{i=1}^{N^l}$ and some unlabeled data, $\mathbf{x}^u \doteq \{x_i^u\}_{i=1}^{N^u}$. The unknown labels $\mathbf{Y}^u \doteq \{Y_i^u\}_{i=1}^{N^u}$ are hidden variables whose "true values" y_i^u are not of interest *per se*, but must be dealt with (nuisance variables). Most SSL approaches attempt to infer or marginalize the unknown labels along with the model parameters, for us the weights w of a neural network, only to discard the former and keep the weights.

Unlike most SSL approaches, in our approach we *estimate the posterior distribution of the unknown labels* $P_i^u \doteq P(Y_i^u | X_i^u = x_i^u)$. The outer loop of the algorithm updates the estimates of the posterior P_i^u, while the inner loop optimizes over the weights (for the fixed estimate of the posterior) to estimate the loss decrease over the time interval. It is important to note that we are *not attempting to infer the weights* (but only the posterior distribution of the unknown labels), which are resampled at each (outer) iteration.[1] By design, the weights do not converge, yet empirically we observe that the posterior distribution of the unknown labels does. We then use the maximum a-posteriori estimate of the labels $\hat{y}_i^u = \arg\max_i P_i^u$ to infer a point-estimate of the weights $\hat{w}$ in a standard supervised training session. This procedure is described in the SaaS algorithm in Sect. 2, where ℓ is a loss function described in Sect. 2, $N = N^u + N^l$ is the total number of samples and η are suitable learning rates for a batch size $|B|$ in SGD.

[1] We have tested both drawing the weights from a Gaussian distribution, or resetting them to their initial value, which yields similar results.

In Sect. 3 we test the SaaS algorithm on SSL benchmarks, and in Sect. 4 we place our contribution in the context of related literature. Next, we describe the algorithm in greater detail.

2 Derivation of the model

We represent a deep neural network with parameters (weights) w, trained for classification into one of K classes, as a function $f_w(x) \in \mathbb{R}^K$ where x is the input (test) datum, and the k-th component of the output approximates the posterior probability $f_{w_t}(x_i)[k] \simeq P(y_i = k|x_i)$. Stochastic gradient descent (SGD) performs incremental updates of the unknown parameters with each iteration t computing the loss by summing on a random subset of the training set called "mini-batch" B_t. The number of iterations needed to sample all the dataset is called an epoch. We represent SGD as an operator $G(\cdot) : w_t \to w_{t+1}$, which maps the current estimate of the weights to the next one. Note that G depends on the the given (true) labels, and the hypothesized ones for the unlabeled data.

To quantify *learning speed* we use the cumulative loss in a fixed time (epoch) interval: For a given training set $\{\mathbf{x}, \mathbf{y}\}$, it is the aggregated loss during T optimization steps, *i.e.*, the area under the learning curve

$$\mathcal{L}_T = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{|B_t|}\sum_{i=1}^{|B_t|}\ell(x_i, y_i; w_t) \tag{1}$$

where $|B_t|$ denotes the cardinality of the mini-batch, composed of samples $(x_i, y_i) \sim P(x, y)$; ℓ denotes the classification loss corresponding to weights w_t. Computing the loss above over all the data points requires the labels being known. Alternatively, it can be interpreted as a loss for a joint hypothesis for the weights w_t *and* the label y_i. We use the cross-entropy loss, which is the sampled version of $H_{P,Q}(y|x) = \mathbb{E}_{P(x)}\mathbb{E}_{P(y|x)} - \log Q(y|x)$ where $Q(y = k|x) = f_w(x)[k]$ is the kth coordinate of the output of the network.

The true joint distribution $P(x)P(y|x) = P(x, y)$ is not known, but the dataset is sampled from it. In particular, if $y_i = k$ is the true label for x_i, we have $P(y_i|x_i) = \delta(y_i - k)$ where δ is Dirac's Delta. Otherwise, we represent it as an unknown K-dimensional probability vector P_i^u with k-th component $P_i^u[k] = P(y_i = k|x_i),\ k = 1, \ldots, K$, to be inferred along with the unknown weights w. We can write the sum $\sum_{k=1}^{K} P_i[k]P_j[k]$ as an inner product between the probability vectors $\langle P_i, P_j\rangle = P_i^T P_j$, so that the *cumulative loss* can be written as

$$\mathcal{L}_T(P^u) = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{|B_t^u|}\sum_{i=1}^{|B_t^u|}\underbrace{-\langle \log f_{w_t}(x_i^u), P_i^u\rangle}_{\ell(x_i^u, P_i^u; w_t)} \tag{2}$$

where B_t^u is mini-batch of unlabeled samples at iteration t. Note that cross-entropy depends on the *posterior distribution* of the unknown labels, P_i^u, rather than their sample value y_i^u. The loss depends on the posterior for the entire

unlabeled set, which we indicate as an $N^u \times K$ matrix P^u, and the entire set of weights $w = \{w_1, \dots, w_T\}$.

We also add as an explicit regularizer the entropy of the network outputs for the unlabeled samples: $-\mathbb{E}_Q \log Q(y^u|x^u)$, as common in SSL [2], which we approximate with the unlabeled samples as

$$H_Q(w) = \sum_{i=1}^{N^u} - \underbrace{\langle f_w(x_i^u), \log f_w(x_i^u) \rangle}_{q(x_i^u; w)} \tag{3}$$

We further incorporate data augmentation by averaging over group transformations $g(x) \in \mathbb{G}$, such as translation and horizontal flipping, sampled uniformly $g_i \sim \mathcal{U}(\mathbb{G})$. Let us define the following shorthand notations: $\ell(B_t^u, P^u; w_{t-1}) = \frac{1}{|B_t^u|} \sum_{i=1}^{|B_t^u|} \ell(g_i(x_i^u), P_i^u; w_{t-1})$ and $q(B_t^u; w_{t-1}) = \frac{1}{|B_t^u|} \sum_{i=1}^{|B_t^u|} q(g_i(x_i^u); w_{t-1})$. Similarly for labeled set, $\ell(B_t^l, P^l; w_{t-1}) = \frac{1}{|B_t^l|} \sum_{i=1}^{|B_t^l|} \ell(g_i(x_i^l), P_i^l; w_{t-1})$ where $P_i^l = \delta_{i,k}$ is the Kronecker Delta with k the true label associated to x_i^l. The overall learning can be framed as the following optimization

$$\begin{aligned} P^u = \arg\min_{P^u} & \frac{1}{T} \sum_{t=1}^{T} \ell(B_t^u, P^u; w_{t-1}) \\ \text{subject to } & w_{t-\frac{1}{2}} = w_{t-1} - \eta_w \nabla_{w_{t-1}} \left(\ell(B_t^u, P^u; w_{t-1}) + \beta q(B_t^u; w_{t-1}) \right) \\ & w_t = w_{t-\frac{1}{2}} - \eta_w \nabla_{w_{t-\frac{1}{2}}} \ell(B_t^l, P^l; w_{t-\frac{1}{2}}) \ \forall \ t = 1 \dots T \\ & P^u \in \mathcal{S} \end{aligned} \tag{4}$$

where the last constraint imposes that the rows of P^u be in the probability simplex of $\mathbb{R}^K$. The objective of the above optimization is to find the posterior of the unlabeled data that leads to the fastest learning curve, when using stochastic gradient descent to train the weights w on both labeled and unlabeled data. The update of the weights is specifically decomposed into two steps: the first step is an update equation for the weights with *unlabeled* samples and posterior P^u, while the second step updates the weights using the labeled samples and ground truth labels. We stress that the latter update is crucial in order to fit the weights to the available training data, and hence prevents from learning trivial solutions of P^u that lead to a fast convergence rate, but does not fit the data properly. We also note that the entropy term is only minimized for unlabeled samples. We set $\beta = 1$ in all the experiments.

It is customary to regularize the labels in SSL using entropy or a proxy [2–6], including mutual exclusivity [7, 8]. [9] uses mutual exclusivity adaptively by not forcing it in the early epochs for similar categories. All these losses force decision boundaries to be in the low-density region, a desired property under cluster assumptions. [5, 6] also maximize the entropy of the marginal label distribution to balance the classes. Together with entropy minimization, balancing classes is equivalent to maximizing the mutual information between estimates and the data if the label prior is uniform. However, we did not apply this loss to not

restrict ourselves to balanced datasets or to the settings where we have prior knowledge on the label distributions.

2.1 Implementation

To solve the optimization problem in Eq. (4), we perform gradient descent over the unknowns P^u, where P^u is the unknown-label posterior initialized randomly. Starting from w_0 sampled from a Gaussian distribution, the inner loop performs a few epochs of SGD to measure learning speed (cumulative loss) $\mathcal{L}_T$ *while keeping the label posterior fixed*. The outer loop then applies a gradient step to update the unknown-label posterior P^u. After each update, the weights are either reset to w_0, or resampled from the Gaussian. In the beginning of each outer epoch, label estimates $P^u \in \mathbb{R}^{N^u \times K}$ are projected with operation $\Pi(P^u)$ to the closest point on the probability simplex of dimension $N^u \times K$.

After the label posterior converges (the weights never do, by design, in the first phase), we select the maximum a-posteriori estimate $\hat{y}_i^u = \arg\max_i P_i^u$, and proceed with training as if fully supervised in the second phase. We call the resulting algorithm, described in Algorithm 1, SaaS.

Algorithm 1 SaaS Algorithm

$P^u \sim \mathcal{N}(0, I)$
Select learning rates η for the weights η_w and label posteriors η_{P^u}
Phase I: Estimate P^u
while P^u has not stabilized **do**
    $P^u = \Pi(P^u)$ (project posterior onto the probability simplex)
    $w_1 \sim \mathcal{N}(0, I)$
    $\Delta P^u = 0$
    // Run SGD for T steps (on the weights) to estimate loss decrease
    for $t = 1 : T$ **do**
        $w_{t-\frac{1}{2}} = w_{t-1} - \eta_w \nabla_{w_{t-1}} \left(\ell(B_t^u, P^u; w_{t-1}) + \beta q(B_t^u; w_{t-1})\right)$
        $w_t = w_{t-\frac{1}{2}} - \eta_w \nabla_{w_{t-\frac{1}{2}}} \ell(B_t^l, P^l; w_{t-\frac{1}{2}})$
        $\Delta P^u = \Delta P^u + \nabla_{P^u} \ell(B_t^u, P^u; w_t)$
    // Update the posterior distribution
    $P^u = P^u - \eta_{P^u} \Delta P^u$
Phase II: Estimate the weights.
$\hat{y}_i^u = \arg\max_i P_i^u \ \forall i = 1, \ldots, N^u$
$w_1 \sim \mathcal{N}(0, I)$
while w has not stabilized **do**
    $w_{t-\frac{1}{2}} = w_{t-1} - \eta_w \nabla_{w_{t-1}} \frac{1}{|B_t^u|} \sum_{i=1}^{|B_t^u|} \ell(x_i^u, \hat{y}_i^u; w_{t-1})$
    $w_t = w_{t-\frac{1}{2}} - \eta_w \nabla_{w_{t-\frac{1}{2}}} \frac{1}{|B_t^l|} \sum_{i=1}^{|B_t^l|} \ell(x_i^l, y_i^l; w_{t-\frac{1}{2}})$

It should be noted that the computation of the gradient $\nabla_{P^u} \ell(B_t^u, P^u; w_t)$ is not straightforward, as w_t is, in general, a (complex) function of P^u. In the

computation of the gradient, we omit here the dependence of w_t on P^u, and use the approximation $\nabla_{P_i^u} \ell(w_t, x_i^u, P_i^u) \approx -\log f_{w_t}(x_i^u)$. This approximation is exact whenever each data point is visited once (i.e., $T = 1$ epoch); as T is chosen to be relatively small here, we assume that this approximation holds.

It is important to note that, with the SaaS algorithm, we are not attempting to solve the optimization problem: $\min_{w,P^u} \sum_{i=1}^{N} \ell(x_i, P_i^u; w)$. This problem has many trivial solutions, as observed by [10], as deep neural networks can easily fit random labels when trained long enough. Thus, for many posteriors P^u, there are weights w achieving zero loss on this objective. One of many such trivial solutions is setting the label posterior P^u to the outputs of the network trained only with the labeled samples. This would result in the same test performance as that of a supervised baseline and does not utilize the unlabeled samples at all. On the other hand, SaaS uses the cumulative loss up to a *fixed*, small iteration T as an inference criterion for label posterior P^u.

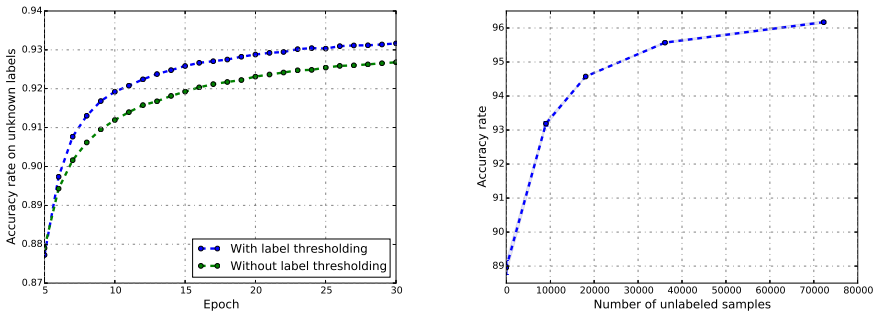

Fig. 2. (Left) **Effect of label thresholding.** We project P^u to the closest probability simplex with minimum probability for a class being 0.05. Plot is given for unlabeled accuracy as label thresholding used in the first phase of the SaaS. The plot is given from epoch 5 to epoch 30. (Right) **Test accuracy vs. number of unlabeled samples.** Accuracy on the test data versus number of unlabeled samples for the SVHN dataset using ResNet18. As the number of unlabeled samples increases, performance improves significantly, as expected in a semi-supervised learning scheme. Results are averaged over three random labeled sets, but error bars are not visible because deviations are smaller than the line-width.

Finally, instead of projecting P^u onto the probability simplex $\mathcal{S}$, we have found that the projection onto a slightly modified set $\mathcal{S}_\alpha = \{x \in \mathbb{R}^K : \sum_i x_i = 1, x_i \geq \alpha\}$ (with $\alpha \geq 0$ chosen to be small) lead to better optimization results for P^u. This is in line with recent work in supervised classification, where this technique is used in order to improve the accuracy of deep neural networks [11, 12]. Fig. 2 (left) illustrates the effect of this approach for SaaS, and shows a clear improvement in SVHN dataset.

3 Empirical evaluation

We test the SaaS algorithm against the state-of-the-art in the most common benchmarks, described next.

Datasets. SVHN [13] consists of 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. CIFAR-10 [14] has $60,000$ 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 [3]. For both datasets, 10% of the training set used for hyper-parameter tuning.

Training. As pointed out by [10], deep networks can easily (over)fit random labels. We set T small enough (40 epochs for CIFAR10 and 5 epochs for SVHN) so that simulated weights cannot fit randomly initialized posterior estimates in the early epochs. We use ResNet18 [15] as our architecture and vanilla SGD with momentum 0.9 as an optimizer. We perform random affine transformations as data augmentations both in SVHN and CIFAR10. We additionally use horizontal flip and color jitter in CIFAR10. Learning rates for w and P^u are chosen as $\eta_w = 0.01$ and $\eta_{P^u} = 1$ respectively. We keep these rates fixed when learning P^u. We fixed the number of outer epochs as well for the first phase of the algorithm by setting it to 75 for SVHN and 135 for CIFAR10. For the second phase of SaaS (supervised part), training is not limited to a small epoch T. Instead, learning rate initialized as 0.1 and halved after 50 epochs unless accuracy in validation is increasing. We stop when the learning rate reaches 0.001.

The baseline for comparison is performance on the same datasets using only the labeled set (i.e. 4K samples for CIFAR10 and 1K samples for SVHN) (Table 1). When training the (supervised) baseline, we employ the same learning parameters, architecture and augmentations as Phase II of SaaS. As expected, SaaS substantially improves baseline results, which is indicative that unlabeled data being effectively exploited by the algorithm. (Table 1)

In Table 2, we compare SaaS with state-of-the-art SSL methods on standard SSL benchmarks. In CIFAR-10, algorithms are trained with 4,000 labeled and 46,000 unlabeled samples. In SVHN, they are trained with 1,000 labeled and 72,257 unlabeled samples. The means and deviations of the test errors are reported by averaging over three random labeled sets. The state-of-the-art methods we compare include input smoothing algorithms [3], ensembling models [16, 17], generative models [18] and models employing problem specific prior [19]. SaaS is comparable to state-of-the-art methods. Specifically, SaaS achieves the best performance in SVHN and second best result in CIFAR10 after VAT. Considering that VAT does input smoothing by adversarial training, our performance can be improved by combining with it.

An SSL algorithm is expected to be more accurate when the number of unlabeled data increases. As it can be seen in Fig.2 (right), we consistently get better results with more unlabeled samples.

Dataset	CIFAR10-4k	SVHN-1k
Error rate by supervised baseline on test data	17.64 ± 0.58	11.04 ± 0.50
Error rate by SaaS on unlabeled data	12.81 ± 0.08	6.22 ± 0.02
Error rate by SaaS on test data	10.94 ± 0.07	3.82± 0.09

Table 1. Baseline error rates. Error rates on the benchmark test set for the baseline system trained with 4K labeled samples on CIFAR10 and 1K labeled samples on SVHN. **Error rate on unknown labels.** SaaS performance on the *unlabeled* set. **Error rate on test data.** SaaS performance on the *test* set. Results are averaged over three random labeled sets. As it can be seen, results of SaaS on test data are significantly better than that of baseline supervised algorithm.

Method-Dataset	CIFAR10-4k	SVHN-1k
VAT+EntMin [3]	**10.55**	3.86
Stochastic Transformation [19]	11.29	NR
Temporal Ensemble [17]	12.16	4.42
GAN+FM [18]	15.59	5.88
Mean Teacher [16]	12.31	3.95
SaaS	10.94 ± 0.07	**3.82± 0.09**

Table 2. Comparison with the state-of-the-art. Error rates on the test set are given for CIFAR10 and SVHN. NR stands for "not reported." CIFAR10 is trained using 4K labels, SVHN using 1K. Results are averaged over three random labeled sets. Despite its simplicity, SaaS performs at the state-of-the-art. It could be combined with adversarial examples (VAT) but here we report the naked results to highlight the role of speed as a proxy for learning in a semi-supervised setting while maintaining a simple learning scheme.

We motivated SaaS as a method finding labels for which training decrease in a fix small number of epochs (e.g. 10) is the maximum. To verify that our algorithm actually does what is intended to do, we train networks on the pseudo-labels generated by SaaS. One can see in Fig.3 (left) that as SaaS iterates more (i.e. as the number of updates for P^u increases), resulting pseudo-labels leads to larger training loss decrease (faster training) in the early epochs. This experiment verifies that SaaS gives pseudo-labels on which training would be faster.

The results reported in Table 1 are with ResNet18 and affine augmentations. Our method uses augmentation, but for direct comparison with some of the

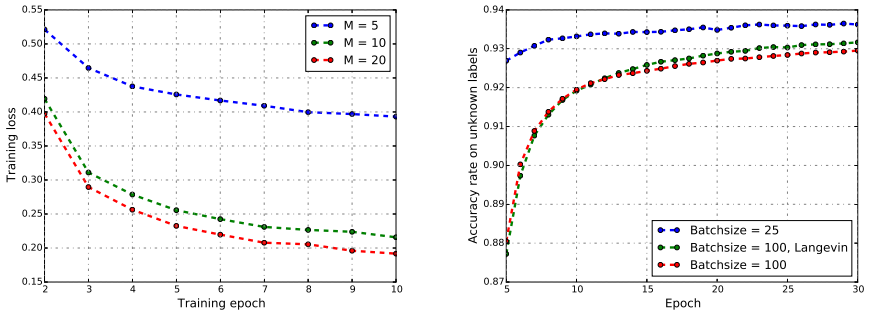

Fig. 3. (Left) **SaaS finds pseudo-labels on which training is faster.** Training loss for a network trained with given P^u, as estimated by the first phase of our algorithm with different numbers of outer epochs M. M is the number of epoch at which outer iteration stopped in the SaaS. Label hypotheses generated by our algorithm lead to faster training as the iteration count increases. Losses are plotted starting from epoch 2. This plot verifies that our algorithm finds labels on which training is faster. (Right) **Effect of Langevin.** Performance improves with smaller batch size, albeit at a significant computational cost: algorithm is about three times slower for $|B| = 25$ (plots shown for SVHN). We therefore choose $|B| = 100$ and add zero-mean Gaussian noise to the weight updates for comparable results (Langevin). The results converge to those of $|B| = 25$ when we train longer. Plot is given for unlabeled accuracy because Langevin is only used in the first phase of SaaS. The plot is given from epoch 5 to epoch 30.

previous papers, we also report results with the convolutional network "conv-large" and translational augmentations as used in [3, 16] in Table 3. Additionally, horizontal flipping is used in CIFAR10. Moreover, we applied pre-processing by centering relative to the Mahalanobis metric (known as ZCA) as in [3, 16].

Dataset	CIFAR10-4k	SVHN-1k
Error rate by supervised baseline on test data	17.88 ± 0.19	12.72 ± 1.13
Error rate by SaaS on unlabeled data	14.26 ± 0.30	7.26 ± 0.19
Error rate by SaaS on test data	13.22 ± 0.31	4.77 ± 0.27

Table 3. For direct comparison, we implement SaaS with the "conv-large" architecture of [3] and the same augmentation scheme. Baseline performance (supervised) is also shown. SaaS improves both on the unlabeled set and test set. Results are averaged over three random labeled sets.

Small batch-size and Langevin dynamics. Finally, we discuss a method we use to reduce the training time for SaaS. We achieve better performance with smaller batch size $|B| = 25$ for both labeled and unlabeled data. When

$|B| = 100$, generalization performance degrades as expected [20]. Unfortunately, small batch-size slows down training, so we use $|B| = 100$ for both labeled and unlabeled data and add zero-mean Gaussian noise to the weight updates, a process known as stochastic gradient Langevin dynamics (SGLD) [21–23], with variance $10^{-5}\eta_w$ for all the datasets. Comparison of small and large batches without noise and large batches with noise can be seen in Fig. 3 (right). With this, first phase of the algorithm (getting the estimates of unknown labels) takes about 1 day for SVHN and 4 days for CIFAR10 using GeForce GTX 1080 when we use ResNet18.

Failure case. As we have shown, our algorithm performs well even with few augmentations (e.g. only translation). However, when we do not use any augmentation at all, our method does not perform as well. With no augmentation and ResNet18, our algorithm suffers a very large drop in performance, achieving error rates of 40.19 ± 3.89 for SVHN and 64.05 ± 1.79 for CIFAR10 on unlabeled data. We next explain this substantial change in the performance.

Fig. 1 suggests that there is a strong correlation between label accuracy and training speed. However, note that this plot is an average over different realizations of labels and initial weights. This does not suggest that for every realization of random labels, this correlation would hold. A simple example is having constant labeling on all the samples for which training would be immediate. In this case, most of the labels would be incorrect for a balanced dataset meaning that correlation between training speed and label accuracy does not hold for every single realization. Hence, we need to have a way of eliminating the degenerate solutions. While the first constraint we put to eliminate these solutions is to impose a small training loss on labeled examples, this might not be enough in many semi-supervised settings. Data augmentation further puts constraints on the desired unknown label posterior: labels of images have to remain constant with respect to image transformations. This constraint hence guides the algorithm to the desired label posterior, and leads to significant performance gains on SaaS.

4 Discussion and related work

The key idea of our approach to SSL is to leverage on training speed as a proxy to measure the quality of putative labels as they are iteratively refined in a differentiable manner.

That speed of convergence relates to generalization is implicit in the work of [24], who derive an upper bound on generalization error as a function of a constant times the sum of step sizes, suggesting that faster training correlates with better generalization.

Another way of understanding our method is via shooting algorithms used to solve boundary value problems (BVP). In a BVP with second order dynamics, a trajectory is found by simulating it with a guess of initial state; then, the initial state is refined iteratively such that the target error would be minimized.

In our problem, dynamics are given by SGD. Assuming that we use SGD without momentum, we have a first order differential equation. The first boundary condition is the initialization of weights and the second boundary condition is a small cumulative loss. The latter one is used to refine P^u which is a parameter of the dynamics instead of the initial state.

In the next paragraphs we discuss our contribution in relation to the vast and growing literature on SSL.

Ensemble Methods include teacher-student models, that use a combination of estimates (or weights) of classifiers trained under stochastic transformations. Although we train only one network, our method resembles the teachers-student models: Our P^u update is similar to a teacher classifier in the teacher-student models. However, we randomly start a student model at each outer epoch. In [17] the prediction of the network over the training epochs are averaged, whereby in each epoch a different augmentation is applied. [16] minimize the consistency cost, which is the distance between two network outputs. Hence, the student network minimizes classification and consistency costs with labeled data and only consistency with the unlabeled data. The weights of the teacher model are the running average of the weights of the student network.

Cluster assumption. The cluster assumption posits that inputs with the same class are in the same cluster under an appropriate metric. It takes many forms (max-margin, low-density separation, smoothness, manifold). In general, it could be framed as $\int ||\nabla_x f_w(x)||^2 d\mu_x$ being small where μ_x is the probability distribution over some manifold. VAT [25, 3] is a recent application of this idea to deep networks, realized by adding a regularization term to minimize the difference between the network outputs for clean and adversarial noise-added-inputs. This state-of-the-art method is similar to the adversarial training of [26], the main difference being that it does not require label information, and thus can be applied to SSL. Our method is orthogonal to VAT and can be improved by combining with it.

Self-training is an iterative process where confident labels from previous iterations are used as ground truth. In [27], disjoint subsets of features of labeled samples are used to produce different hypotheses on randomly selected subsets of the unlabeled data. Labeled data are extended with the most confident estimates on this subset. This approach fails to enlarge the sigma-algebra generated by the labeled samples and generally fails if the classifier does not give correct estimates for at least one feature subset. We maintain an estimate of the posterior probability of each label, and only force a point estimate in the refinement (second) phase of the algorithm.

Encoding priors. In image classification one can enforce invariance of labels to some transformations. This is achieved by minimizing the difference between network outputs under different transformations. In [19], transformations are affine

(translation, rotation, flipping, stretching and shearing). Although they achieve good results, their improvement on baseline supervised performance (using only labeled data) is marginal. *E.g.*, in CIFAR-10 supervised error is 13.6% while semi-supervised error is 11.29%. Similarly, [28] suggests minimizing the norm of directional derivatives of the network with respect to small transformations. We also employ augmentations like most SSL papers on image classification.

Generative models used to be the standard for SSL, but the high dimensionality of problems in vision presents a challenge. Adversarial methods like GANs have been recently applied, whereby an additional $C+1$-th (fake) class is used. The loss function is designed to force the discriminator output to be low for the fake class for the unlabeled samples while making it high for the generated samples. [18] suggested a regularizer for the generator, called feature matching (FM), whereby the generator tries to match the first-order statistics of the generated sample features to those of the real data. According to [4], the discriminator benefits the generator if it has samples within the data manifold, but around subspaces in which the density of samples is low. Unlike feature matching, they match the inverse distribution in non-zero density areas rather than their means. Instead of one generator network, [29] uses an encoder-decoder network generating images and labels from which the discriminator tries to differentiate.

Graph based methods. [30] assumes that an affinity matrix of size $N \times N$ is given, which has information independent from the one in the features of the data. In the loss function, they have a term penalizing different labels assigned to similar samples based on this similarity matrix. [31, 32] finds a sparse clustering using the ℓ_1-norm. [33] propagates pairwise *must* and *cannot* constraints in an efficient way. [34] uses the current hypothesis for the unknown labels in learning as in our algorithm. They update the affinity matrix and estimates of the unknown labels iteratively. [35] suggests a dictionary learning method which can be used for SSL. Recent graph based methods [36–39] exploit deep networks for function approximation in a manner that can be used for SSL.

Within this rich and multi-faceted context, our approach provides one more element to consider: The fact that the speed of convergence when optimizing with respect to the probability of unknown labels is highly dependent on their correctness, even when starting from a random initial condition. This frees us from having to jointly optimized the parameters and the posterior on the labels, which would blow up the dimensionality, and allows us to focus sequentially on first estimating the unknown label distribution – irrespective of the model parameters/weight – and then retrieve the weights using the maximum a-posteriori estimate of the labels.

Our method can be combined with other ideas recently introduced in SSL, including using adversarial examples. We do not do so in our experiments, to isolate the contribution of our algorithm. Nevertheless, just the method alone, with some data augmentation but without sophisticated tricks, achieves promising performance.

Acknowledgment Research supported by ONR N00014-13-1-0563 and ARO W911NF-17-1-0304.

References

1. Chapelle, O., Scholkopf, B., Zien, A.: Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. IEEE Transactions on Neural Networks **20**(3) (2009) 542–542
2. Grandvalet, Y., Bengio, Y.: Semi-supervised learning by entropy minimization. In: Advances in neural information processing systems. (2005) 529–536
3. Miyato, T., Maeda, S.i., Koyama, M., Ishii, S.: Virtual adversarial training: a regularization method for supervised and semi-supervised learning. arXiv preprint arXiv:1704.03976 (2017)
4. Dai, Z., Yang, Z., Yang, F., Cohen, W.W., Salakhutdinov, R.R.: Good semi-supervised learning that requires a bad gan. In: Advances in Neural Information Processing Systems. (2017) 6513–6523
5. Krause, A., Perona, P., Gomes, R.G.: Discriminative clustering by regularized information maximization. In: Advances in neural information processing systems. (2010) 775–783
6. Springenberg, J.T.: Unsupervised and semi-supervised learning with categorical generative adversarial networks. arXiv preprint arXiv:1511.06390 (2015)
7. Sajjadi, M., Javanmardi, M., Tasdizen, T.: Mutual exclusivity loss for semi-supervised deep learning. In: Image Processing (ICIP), 2016 IEEE International Conference on, IEEE (2016) 1908–1912
8. Xu, J., Zhang, Z., Friedman, T., Liang, Y., Broeck, G.V.d.: A semantic loss function for deep learning with symbolic knowledge. arXiv preprint arXiv:1711.11157 (2017)
9. Shrivastava, A., Singh, S., Gupta, A.: Constrained semi-supervised learning using attributes and comparative attributes. In: European Conference on Computer Vision, Springer (2012) 369–383
10. Zhang, C., Bengio, S., Hardt, M., Recht, B., Vinyals, O.: Understanding deep learning requires rethinking generalization. arXiv preprint arXiv:1611.03530 (2016)
11. Pereyra, G., Tucker, G., Chorowski, J., Kaiser, Ł., Hinton, G.: Regularizing neural networks by penalizing confident output distributions. arXiv preprint arXiv:1701.06548 (2017)
12. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2016) 2818–2826
13. Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y.: Reading digits in natural images with unsupervised feature learning. In: NIPS workshop on deep learning and unsupervised feature learning. Volume 2011. (2011) 5
14. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. (2009)
15. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: European conference on computer vision, Springer (2016) 630–645
16. Tarvainen, A., Valpola, H.: Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In: Advances in neural information processing systems. (2017) 1195–1204
17. Laine, S., Aila, T.: Temporal ensembling for semi-supervised learning. arXiv preprint arXiv:1610.02242 (2016)

18. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X.: Improved techniques for training gans. In: Advances in Neural Information Processing Systems. (2016) 2234–2242
19. Sajjadi, M., Javanmardi, M., Tasdizen, T.: Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In: Advances in Neural Information Processing Systems. (2016) 1163–1171
20. Keskar, N.S., Mudigere, D., Nocedal, J., Smelyanskiy, M., Tang, P.T.P.: On large-batch training for deep learning: Generalization gap and sharp minima. arXiv preprint arXiv:1609.04836 (2016)
21. Welling, M., Teh, Y.W.: Bayesian learning via stochastic gradient langevin dynamics. In: Proceedings of the 28th International Conference on Machine Learning (ICML-11). (2011) 681–688
22. Raginsky, M., Rakhlin, A., Telgarsky, M.: Non-convex learning via stochastic gradient langevin dynamics: a nonasymptotic analysis. In: Proceedings of the 30th Conference on Learning Theory, COLT 2017, Amsterdam, The Netherlands, 7-10 July 2017. (2017) 1674–1703
23. Chaudhari, P., Choromanska, A., Soatto, S., LeCun, Y.: Entropy-sgd: Biasing gradient descent into wide valleys. arXiv preprint arXiv:1611.01838 (2016)
24. Hardt, M., Recht, B., Singer, Y.: Train faster, generalize better: Stability of stochastic gradient descent. In: Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016. (2016) 1225–1234
25. Miyato, T., Maeda, S.i., Koyama, M., Nakae, K., Ishii, S.: Distributional smoothing with virtual adversarial training. arXiv preprint arXiv:1507.00677 (2015)
26. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
27. Blum, A., Mitchell, T.: Combining labeled and unlabeled data with co-training. In: Proceedings of the eleventh annual conference on Computational learning theory, ACM (1998) 92–100
28. Simard, P., Victorri, B., LeCun, Y., Denker, J.: Tangent prop-a formalism for specifying selected invariances in an adaptive network. In: Advances in neural information processing systems. (1992) 895–903
29. Dumoulin, V., Belghazi, I., Poole, B., Lamb, A., Arjovsky, M., Mastropietro, O., Courville, A.: Adversarially learned inference. arXiv preprint arXiv:1606.00704 (2016)
30. Yang, Z., Cohen, W.W., Salakhutdinov, R.: Revisiting semi-supervised learning with graph embeddings. In: Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016. (2016) 40–48
31. Nie, F., Wang, H., Huang, H., Ding, C.: Unsupervised and semi-supervised learning via 1-norm graph. In: Computer Vision (ICCV), 2011 IEEE International Conference on, IEEE (2011) 2268–2273
32. Su, H., Zhu, J., Yin, Z., Dong, Y., Zhang, B.: Efficient and robust semi-supervised learning over a sparse-regularized graph. In: European Conference on Computer Vision, Springer (2016) 583–598
33. Lu, Z., Ip, H.H.: Constrained spectral clustering via exhaustive and efficient constraint propagation. In: European Conference on Computer Vision, Springer (2010) 1–14
34. Li, C.G., Lin, Z., Zhang, H., Guo, J.: Learning semi-supervised representation towards a unified optimization framework for semi-supervised learning. In: Proceedings of the IEEE International Conference on Computer Vision. (2015) 2767–2775

35. Wang, X., Guo, X., Li, S.Z.: Adaptively unified semi-supervised dictionary learning with active points. In: Proceedings of the IEEE International Conference on Computer Vision. (2015) 1787–1795
36. Haeusser, P., Mordvintsev, A., Cremers, D.: Learning by association-a versatile semi-supervised training method for neural networks. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR). (2017)
37. Gaunt, A., Tarlow, D., Brockschmidt, M., Urtasun, R., Liao, R., Zemel, R.: Graph partition neural networks for semi-supervised classification. (2018)
38. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
39. Weston, J., Ratle, F., Mobahi, H., Collobert, R.: Deep learning via semi-supervised embedding. In: Neural Networks: Tricks of the Trade. Springer (2012) 639–655