10701 SSL Project Final Report

Shreyan Jaiswal **Leron Julian** **Gus Welter**

1 Introduction

Semi-supervised learning methods use both labeled and unlabeled data to learn in order to achieve comparable performance with limited labeled data.

We are investigating methods to improve the performance of the self-training approach to semi-supervised learning in a scenario of constrained labeled examples and a much larger availability of unclassified examples. Specifically, we attempt to use a small labeled dataset L and a large unlabeled data pool U to train a 2-layer convolutional neural network in order to classify handwritten digits of the MNIST dataset and small images of the STL10 dataset. The inputs of the model we wish to train are pixel values where each sample is a flattened representation of a pixel image, and the outputs are probability vectors for each of the possible labels. We evaluated our results primarily by test set accuracy of the trained model.

2 Data

2.1 MNIST

We are using the MNIST dataset of handwritten digits, which consists of size-normalized, centered digits in fixed-size 28x28 pixel images. MNIST includes 60,000 training set and 10,000 test set images. The MNIST images were adapted from a two subsets of the NIST database.

The MNIST training set incorporates 30,000 samples from NIST SD-3, which are handwriting samples from American Census Bureau employees, and 30,000 samples from NIST SD-1, which are handwriting samples from American high-school students. The MNIST test set, likewise, incorporates 5,000 samples each from SD-3 and SD-1. The sets of writers represented in the training and test sets are ensured to be disjoint. While the NIST images were 20x20 black-and-white images, the MNIST are 28x28 greyscale images. The greyscale is due to antialiasing from upscaling.

The MNIST dataset is sampled from two narrow subpopulations of the general American population, and thereby it is not a representative distribution of either the general American population or the global Western Arabic numeral writing population. The MNIST dataset curators do not mention the writing utensil(s) used, and so it may not incorporate a representative sampling of writing utensil types or subtypes. The fact that the samples were represented originally in black-and-white (meaning some nuance of the written sample was lost) and then transformed into greyscale with anti-aliasing must also be noted as a bias of the dataset.

We vary the labeled data sample size from 100-500. We set our labeled dataset L to be a random sample of that size from the total dataset, putting the remainder of the 60,000 samples in the unlabeled pool (so varied from 59,900 to 59,500).

2.2 STL10

We also use the STL10 dataset which consists of 96x96 pixel images over 10 classes. STL10 is split into training, testing, and unlabeled datasets. The training dataset, which we used as our labeled dataset, includes 500 images per class, for a total of 5,000 images. The unlabeled dataset, which we used as our unlabeled pool, includes a total of 100,000 images, though the unlabeled images are extracted from a broader distribution of images. The test dataset includes 800 images per class, for

a total of 8,000 images. The STL10 images were adapted from the ImageNet database of labeled images.

3 Related Work

For our experiments, we are performing image classification on the MNIST dataset. We initially used a convolutional neural network for this supervised learning task, based on the original work of He et al. [2015] in which deep residual learning is used coinciding with 18 convolutional layers followed by typical neural network features (ResNet-18). However, we decided to simplify our model to a version of the simple 2-D convolutional layer model (Koehler [2020]) which had similar results on MNIST as the ResNet-18 model.

For our baseline, we implemented a foundational semi-supervised learning method called self-training which involves iteratively training a supervised classification model and then imputing labels onto unlabeled examples where the trained model can determine a label with a certain confidence, thereby converting these unlabeled examples into labeled examples for the next iteration. This method was summarized recently in Ruder [2018], with original early work in Yarowsky [1995] and McClosky et al. [2006].

In Berthelot et al. [2019], the latest approaches in these areas were combined into a cohesive, unified SSL approach they call MixMatch and also performed ablation studies for analysis of the contribution of each approach to their overall results. They achieved dramatically better test error rates compared to other SSL models in many cases, for example by a factor of 4 with CIFAR-10. We used this work as a road map of current consistency regularization and traditional regularization approaches.

4 Methods/Model

In this section, we introduce our proposed SSL method for image classification, which combines several of the methods introduced in the above section. Given a small batch L of labeled images with one-hot targets in $\{1, \ldots, M\}$ (representing a possible label on each image) and a larger batch U of unlabeled images, we aim to produce a model with parameters θ which outputs on an image x a probability vector $p_{\text{model}}(y|x;\theta)$.

4.1 The Model

Initially, the supervised model we use by Koehler [2020] is a neural network which utilizes two 2-D convolutional layers followed by two fully-connected (or linear) layers. We implement our convolutional layers with small filter sizes starting with 3-channels as input into the first layer and 20-channels as output from the final layer of the convolutional layers. For the linear layers, our hidden layer consists of 50 features with 10 output features at the end of the network representing the 10 possible classes. Each convolutional layer is followed by a maxPool and ReLU as an activation function. The linear layers are followed by ReLUs. As a means of regularization, we use two dropout layers. The final output is put through a softmax layer.

As a means of improvement on our fairly simple and shallow network, we include 2D batch normalization after each convolutional layer in our network. Batch normalization reduces internal co-variate shift by normalizing layer inputs. As a result, this not only improves our accuracy during training but also during testing as well. This ultimately has a positive effect when imputing pseudo-labels during our self-training loop.

Also as a baseline, we use stochastic gradient descent as our optimizer with a learning rate of 0.001 and cross entropy loss as our loss function

4.1.1 The Loss Function

The loss function is defined to be the cross entropy loss of the output of the model and one-hot vector of the desired target. More formally, on some batch $B \subset L$, we define the loss to be

$$\mathcal{L}(B) = \frac{1}{|B|} \sum_{(x,\hat{y}) \in B} \sum_{c=1}^{M} -\hat{y}_c \log(p_{\text{model}}(y = c|x, \theta))$$

where $\hat{y}_c$ is a binary indicator of the label of the image x.

4.2 The Baseline: Self-Training

Our baseline approach is self-training, adopted from Ruder [2018]. At each iteration, we train the model on the labeled data with SGD. Then, for each unlabeled example x, the model yields a vector of probabilities, one for each class. If the probability assigned to the most likely class is higher than some threshold τ, we add a proxy label for the most likely class and move x from the unlabeled to the labeled examples along side with the proxy label. We repeat this iteration many times.

Algorithm 1 Baseline Self Training

for num_self_learning_reps iterations **do**
  for num_epochs iterations **do**
    Train the model with SGD to minimize $\mathcal{L}$ on one pass of L
  end for
  for $x \in U$ **do**
    if $\max_y p_{\text{model}}(y|x,\theta) \geq \tau$ **then**
      $L = L \cup \{(x, \arg\max_y p_{\text{model}}(y|x,\theta))\}$
      $U = U \setminus \{x\}$
    end if
  end for
end for

4.2.1 Alternate Self-training loops

We can modify this loop in two ways depending on our needs.

Instead of training for a fixed number of epochs, we can train on a fixed number of examples, as shown in Algorithm 2.

Algorithm 2 Baseline Self Training

for num_self_learning_reps iterations **do**
  Train the model with SGD to minimize $\mathcal{L}$ for num_examples examples over L, repeating examples if necessary
  for $x \in U$ **do**
    if $\max_y p_{\text{model}}(y|x,\theta) \geq \tau$ **then**
      $L = L \cup \{(x, \arg\max_y p_{\text{model}}(y|x,\theta))\}$
      $U = U \setminus \{x\}$
    end if
  end for
end for

Also, we can train until we hit a desired accuracy, as shown in Algorithm 3.

4.3 Traditional Regularization

We apply a penalty on the L2 norm of the parameters of the model. We use a parameter λ_{TR} to dictate how much of a penalty is applied. More formally, we implement traditional regularization by modifying the loss function to

$$\mathcal{L}_{\text{with TR}}(B) = \mathcal{L}(B) + \lambda_{\text{TR}}|\theta|^2$$

4.4 Consistency regularization

We experimented with consistently regulation, a method to encourage our model to produce the same results on an augmentation Augment(x) of an input x. The basic form of consistency regularization

Algorithm 3 Baseline Self Training

for num_self_learning_reps iterations **do**
  while the training accuracy is less than some training accuracy threshold **do**
    Train the model with SGD to minimize $\mathcal{L}$ on one pass of L
  end while
  for $x \in U$ **do**
    if $\max_y p_{\text{model}}(y|x,\theta) \geq \tau$ **then**
      $L = L \cup \{(x, \arg\max_y p_{\text{model}}(y|x,\theta))\}$
      $U = U \setminus \{x\}$
    end if
  end for
end for

seeks to minimize the loss

$$\mathcal{L}_{\text{CR}}(B) = \frac{1}{|B|} \sum_{x \in B} \|p_{\text{model}}(y|\text{Augment}(x),\theta) - p_{\text{model}}(y|\text{Augment}(x),\theta)\|^2$$

First, we remark that the Augment function applies a random transformation to x, so the two terms are in general not the same.

We also remark that this loss function does not require any true labels of the data, so we can calculate this loss for data without labels in U.

We attempted to implement this technique using two methods of training with consistency regularization.

4.4.1 Consistency Regularization on the full dataset

After each epoch of SGD on L minimizing $\mathcal{L}$, we run a round of SGD on $L \cup U$ minimizing $\mathcal{L}_{\text{CR}}$. However, instead of running SGD on a full pass of $L \cup U$, we run it only for a subset of it, as otherwise this step would run far too slowly.

Algorithm 4 Self Training with Consistency Regularization on the full dataset

for num_self_learning_reps iterations **do**
  for num_epochs iterations **do**
    Train the model with SGD to minimize $\mathcal{L}$ on one pass of L
    Train the model with SGD to minimize $\mathcal{L}_{\text{CR}}$ on one pass of a subset of $L \cup U$ of size CR_size
  end for
  for $x \in U$ **do**
    if $\max_y p_{\text{model}}(y|x,\theta) \geq \tau$ **then**
      $L = L \cup \{(x, \arg\max_y p_{\text{model}}(y|x,\theta))\}$
      $U = U \setminus \{x\}$
    end if
  end for
end for

4.4.2 Consistency Regularization alongside labeled loss

Instead of training on the just the cross-entropy loss (or just the cross-entropy loss plus a traditional regularization term), we train on a modified loss function that includes a consistency regularization term. Here, the regularization is only applied to the labeled inputs in L, even though the labels are not used to compute this term.

Algorithm 5 Self Training with Consistency Regularization on alongside labeled loss

for num_self_learning_reps iterations **do**
  for num_epochs iterations **do**
    Train the model with SGD to minimize $\mathcal{L} + \lambda_{\text{CR}}\mathcal{L}_{\text{CR}}$ on one pass of L
  end for
  for $x \in U$ **do**
    if $\max_y p_{\text{model}}(y|x,\theta) \geq \tau$ **then**
      $L = L \cup \{(x, \arg\max_y p_{\text{model}}(y|x,\theta))\}$
      $U = U \setminus \{x\}$
    end if
  end for
end for

5 Background

In earlier work, we evaluated the testing accuracy of the baseline Algorithm 2 with num_examples = 60000 on MNIST with $\tau = 0.5$, $\tau = 0.7$, and $\tau = 0.9$, and at labeled data sizes from 100 to 500. We compared these with a control of supervised learning on the small labeled dataset. The results are plotted in Figure 1.

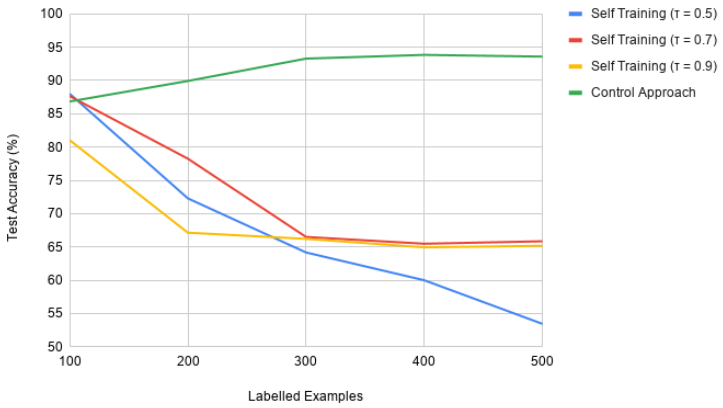


Figure 1: Test accuracy of control and baseline (Algorithm 2) approaches

We observed that the baseline self-training algorithm performs worse than the control algorithm, especially when more labeled examples are added. We attributed our baseline's worse testing accuracy to a major drawback of self training: the inability to correct its own errors. If the model's predictions on the unlabeled data are confident but wrong, it is added to the labeled training data for the error to be amplified by further training. We suspect that many wrong predictions were made in the earlier stages of self-learning, causing more erroneous training.

We attempted to remedy this by running Algorithm 3 with threshold 85%. This would allow more supervised training to happen before imputing labels on the unlabeled data. The results are plotted in Figure 2.

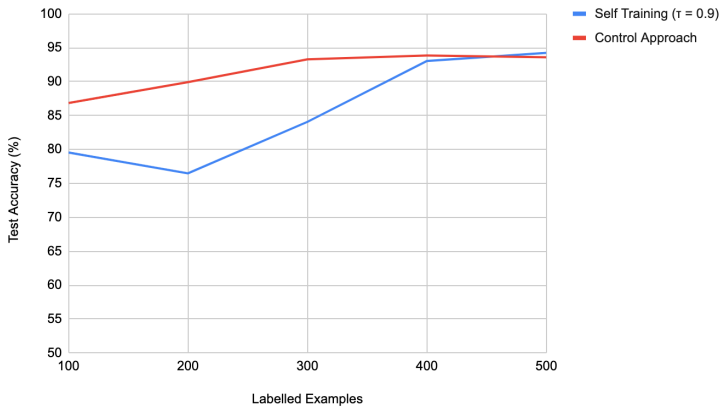


Figure 2: Test accuracy of control and baseline (Algorithm 3) approaches

We observed that the baseline self-training algorithm performs slightly worse and less consistently than the control algorithm. However, this version of self-training performs significantly better than the previous version.

6 Results

The results in Figure 3 were obtained by running Self-Learning SSL on the STL10 dataset with these hyperparameters:

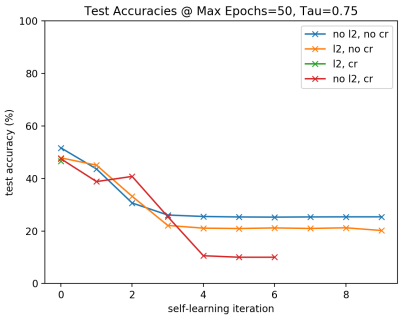
- Traditional Regularization Method: Variable
- Consistency Regularization Method: Variable (none or alongside labeled loss)
- Epochs per Self-Learning Iteration: Variable (specified in figure)
- τ: Variable (specified in figure)
- Batch Size: 32
- Learning Rate: 0.001
- λ_{TR}: 0.01
- λ_{CR}: 0.1
- Max Self-Learning Repetitions: 10 (or when $\#\text{LabelsImputed} \leq 100$)

We did not test consistency regularization on the full dataset, as it ran too slowly for our experiments.

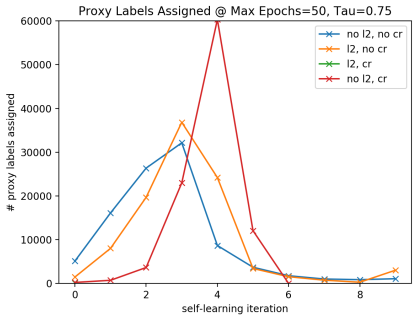
We focused on test accuracy as an evaluation of performance, and we also tracked proxy labeling to understand self-learning behavior.

Self-learning generally worsened test set accuracies across the board, with the sole exception of self-learning iteration #1 with no regularization at max epochs 100 and τ of 0.75. Self-learning with no regularization worsened test accuracy the least. Consistency regularization worsened test accuracy the most. Self-labeling generally imputed the most proxy labels between the second and fifth self-learning iterations, which is when the test accuracies also would be at or nearing their lowest. Consistency regularization also produced the highest number of proxy label assignments.

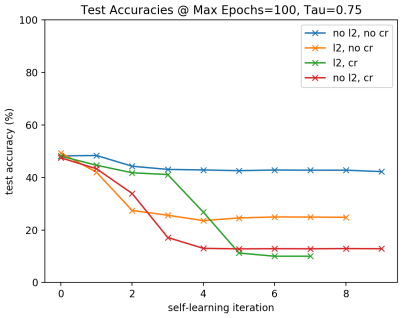
Our final results on STL10 are generally consistent with our midway results on MNIST in that self-learning generally achieved worse test accuracies than our control of pure supervised learning. Regularization worsened these results. In our baseline, algorithm 3 was the only approach to achieve test accuracy better than control. This is likely due to not overfitting as much, since algorithm 3 stopped at a max threshold training accuracy. In our final STL10 approaches, we did not monitor test or holdout accuracies in order to find a global minimum and hold there. Therefore, overfitting likely had the largest detrimental impact on our final result test accuracies. We will discuss further in the Discussion and Analysis section below.



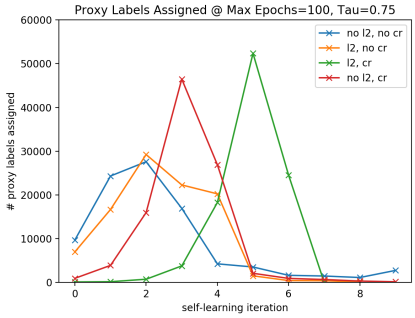
(a) Test accuracies on runs with num_epochs = 50 and $\tau = 0.75$



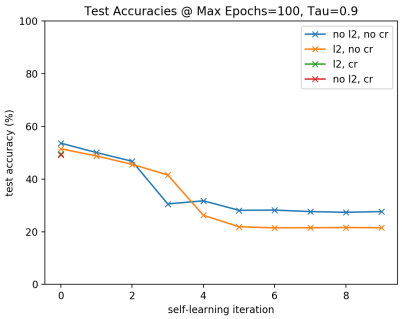
(b) Number of proxy labels assigned at self-learning iterations on runs with num_epochs = 50 and $\tau = 0.75$



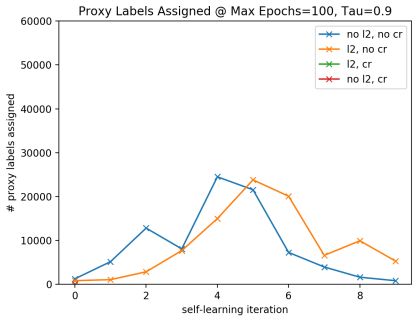
(c) Test accuracies on runs with num_epochs = 100 and $\tau = 0.75$



(d) Number of proxy labels assigned at self-learning iterations on runs with num_epochs = 100 and $\tau = 0.75$



(e) Test accuracies on runs with num_epochs = 100 and $\tau = 0.9$



(f) Number of proxy labels assigned at self-learning iterations on runs with num_epochs = 100 and $\tau = 0.9$

Figure 3: Varied regularization methods in runs of num_epochs $\in \{50, 100\}$ and $\tau \in \{0.75, 0.9\}$

7 Discussion and Analysis

We set out to find a self-training-based SSL approach to train a model to perform better than a model trained by just the labeled dataset. The metric we used to evaluate performance was accuracy on a separate testing set, which we believe is an appropriate measurement to make to evaluate these

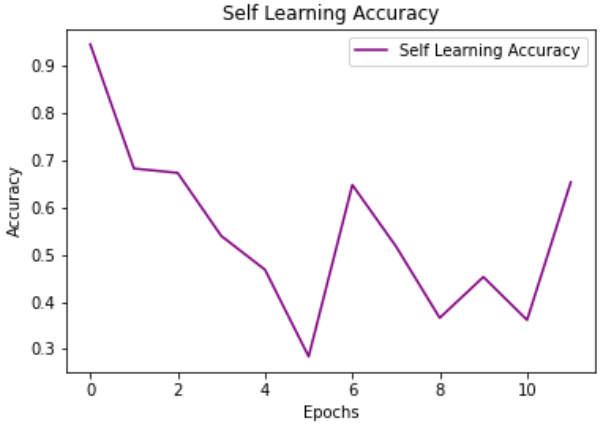


Figure 4: Self-learning test accuracy of baseline (Algorithm 2) approach, 100 labeled examples

image-classifying models. We conclude that none of the approaches we experimented with were able to perform to this standard.

One limitation of self-training-based SSL methods is that they have difficulty correcting errors in imputed labels. Our current approaches have no way of correcting or removing an imputed label after it has been assigned. Although this could be added, it's unlikely that it would be able to correct these labels in practice. When an unlabeled image is imputed with an improper label, the model will shortly after be trained on this image-label pair. This reinforces the error that was made when imputing the label.

Though this assumption was not directly made, we find it important to note that in STL10, the unlabeled images come from a "similar but broader distribution of images". For example, it contains other types of animals (bears and rabbits) and vehicles (trains and buses) than those in the labeled set. Another limitation of self-training is that it alone cannot properly use unlabeled data outside of the labeled distribution. In a perfect world, self-training should ignore these images. However, it may have been the case that labels may be mistakenly imputed on these images, distorting the meaning of that label in the training data. We did attempt to enhance self-training with consistency regularization to utilize this data, but it ultimately did more harm than good to the performance of the model.

Although we were unable to measure the accuracy of imputed labels for our runs on the STL10 datasets, our experiments on MNIST suggest that the imputed labels are accurate during the first few rounds of self training, but get much worse in further repetitions (Figure 4). This suggests that our approaches overfit on the training data and then impute erroneous labels on the unlabeled data. This causes more overfitting and more erroneous labels, resulting in the testing accuracy becoming increasingly poor.

One extreme possibility is that our model converges close to a constant function (i.e. one that predicts the same label for every input image). When labels are imputed on the unlabeled data, the labeled set gets further concentrated with that same label, further encouraging the model to continue this degenerate behavior. This would explain why in some cases, the accuracy converged to exactly 10%. This would also explain the "spike" in the number of proxy labels assigned around iteration four to six, as the model could be confidently classifying almost the entire unlabeled images as one label. Furthermore, since consistency regularization encourages the model to predict the same images for augmented versions of the same image, it can further lead the model down this incorrect path.

One possible set of approaches to address the error-correcting limitations is to train with multiple models, which work to correct each other's self-training errors. In these approaches, the models alternately act as teachers and students, where the teachers provide proxy labels for the student models (Ruder [2018]). Although our self-training-based SSL methods did not meet our performance benchmark, we believe that the improvements above and some more tinkering with the hyperparameters could alleviate self-training's inherent limitations to meet this benchmark.

References

D. Berthelot, N. Carlini, I. J. Goodfellow, N. Papernot, A. Oliver, and C. Raffel. Mixmatch: A holistic approach to semi-supervised learning. *CoRR*, abs/1905.02249, 2019. URL http://arxiv.org/abs/1905.02249.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

G. Koehler. Mnist handwritten digit recognition in pytorch. https://nextjournal.com/gkoehler/pytorch-mnist, 2020. Accessed: 2020-03-02.

D. McClosky, E. Charniak, and M. Johnson. Effective self-training for parsing. In *Proceedings of the Main Conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, HLT-NAACL '06, page 152–159, USA, 2006. Association for Computational Linguistics. doi: 10.3115/1220835.1220855. URL https://doi.org/10.3115/1220835.1220855.

S. Ruder. An overview of proxy-label approaches for semi-supervised learning. https://ruder.io/semi-supervised/, 2018. Accessed: 2020-03-02.

D. Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *33rd Annual Meeting of the Association for Computational Linguistics*, pages 189–196, Cambridge, Massachusetts, USA, June 1995. Association for Computational Linguistics. doi: 10.3115/981658.981684. URL https://www.aclweb.org/anthology/P95-1026.