Stochastic coordinate descent for 01 loss and its sensitivity to adversarial attacks

1st Meiyan Xie
Department of Computer Science
New Jersey Institute of Technology
Newark, USA
mx42@njit.edu

2nd Yunzhe Xue
Department of Computer Science
New Jersey Institute of Technology
Newark, USA
yx277@njit.edu

3rd Usman Roshan
Department of Computer Science
New Jersey Institute of Technology
Newark, USA
usman@njit.edu

***Abstract*—The 01 loss while hard to optimize is least sensitive to outliers compared to its continuous differentiable counterparts, namely hinge and logistic loss. Recently the 01 loss has been shown to be most robust compared to surrogate losses against corrupted labels which can be interpreted as adversarial attacks. Here we propose a stochastic coordinate descent heuristic for linear 01 loss classification. We implement and study our heuristic on real datasets from the UCI machine learning archive and find our method to be comparable to the support vector machine in accuracy and tractable in training time. We conjecture that the 01 loss may be harder to attack in a black box setting due to its non-continuity and infinite solution space. We train our linear classifier in a one-vs-one multi-class strategy on CIFAR10 and STL10 image benchmark datasets. In both cases we find our classifier to have the same accuracy as the linear support vector machine but more resilient to black box attacks. On CIFAR10 the linear support vector machine has 0% on adversarial examples while the 01 loss classifier hovers about 10%. On STL10 the linear support vector machine has 0% accuracy whereas 01 loss is at 10%. Our work here suggests that 01 loss may be more resilient to adversarial attacks than the hinge loss and further work is required.**

***Index Terms*—Stochastic coordinate descent, 01 loss, adversarial attacks**

I. Introduction

The problem of determining the hyperplane with minimum number of misclassifications in a binary classification problem is known to be NP-hard [1]. In mainstream machine learning literature this is called minimizing the 01 loss [2] as given in Objective 1,

$$\frac{1}{2n}\underset{w,w_0}{\arg\min}\sum_i(1-sign(y_i(w^Tx_i+w_0))) \tag{1}$$

where $w \in R^d$, $w_0 \in R$ is our hyperplane solution, and $x_i \in R^d, y_i \in \{+1,-1\}.\forall i = 0...n-1$ are our training data. Popular linear classifiers such as the linear support vector machine, perceptron, and logistic regression [3] can be considered as convex approximations to this problem that yield fast gradient descent solutions [4]. However, they are also more sensitive to outliers than the 01 loss [5].

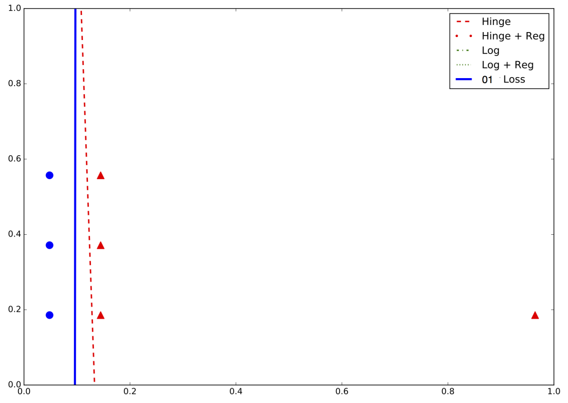
In Figure 1 we demonstrate the effect of a single outlier on the hinge, logistic loss, their regularized versions, and 01 loss. In both cases we intuitively desire a vertical hyperplane that divides (1,1), (1,2), and (1,3) from (3,1), (3,2), and (3,3) since this would likely minimize test error. When the outlier is of the same class as in Figure 1(a) all five objectives give similar vertical hyperplanes. The 0/1 loss alone has infinite solutions though and we show a single one here.

When we switch the label of the outlier in Figure 1(b) both the hinge and logistic along with their regularized counterparts give skewed hyperplanes that make several misclassifications on the training data. This is due to the fact that misclassified points increase the hinge and logistic objective (the farther misclassified the point the more the effect) and so in order to lower the objective the hyperplane is skewed towards it. The 0/1 loss however is not affected by distances of outliers and still gives a desired hyperplane.

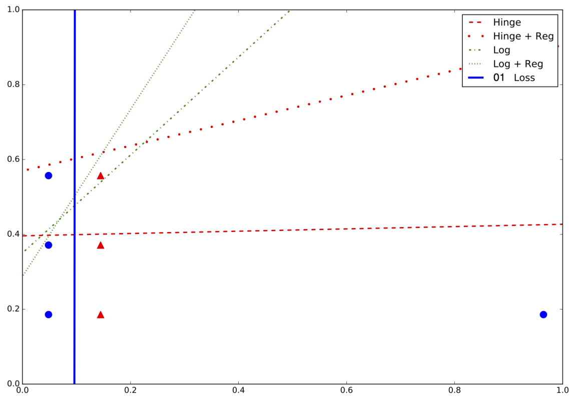
Recently the 01 loss has been shown to be more robust than surrogate loss functions against label corruption [6]. In the standard machine learning model both train and test data follow the same distribution [7]. When the test distribution is different we can interpret this as an adversarial attack where an adversary is giving test data from a different distribution. In this scenario (also called distributionally robust supervised learning [8]) we want to minimize the adversarial empirical risk. It turns out that 01 loss has a monotonic relationship between the empirical risk and adversarial empirical risk: minimizing the empirical risk under 01 loss is the same as minimizing the adversarial empirical risk [8]. This gives the 01 loss additional robustness besides outliers.

We present here a stochastic coordinate descent (SCD) heuristic for 01 loss based on the original stochastic gradient descent method [9]. While the gradient gives the direction of best descent here we perform a heuristic coordinate descent for each stochastic batch. We evaluate our method by comparing it against the a cross-validated linear support vector machine (SVM) on real datasets from the UCI machine learning archive [10]. We find that the cross-validated linear SVM performs slightly better in average and median error but not by a statistically significant margin.

We explore the SCD's sensitivity to a black-box adversarial attack [11], [12], a method that treats the classifier to be attacked as a black box whose model and parameters are unknown. We implement a simple single layer neural network that we use to estimate the black box's gradient and to produce adversarial examples targeting the black box. We perform two



(a)



(b)

Fig. 1. In (a) we see that an outlier of the same class is not a problem for hinge, logistic, and 0/1 loss objective. However, when we switch its label it affects hinge and logistic considerably while the 0/1 loss decision surface remains the same (although there are an infinite number of solutions we show just one here).

separate tests on both SCD 01 loss and the linear SVM on each CIFAR10 and STL10 image object detection benchmarks. On CIFAR1 we find that the linear SVM quickly approaches a 0% accuracy after a few epochs of the black box attack whereas the SCD 01 loss fluctuates and stays above 5%. We see the same on STL10 except there 01 loss stays above 10% accuracy.

While greater exploration is needed, such as a larger neural network as the gradient approximator and experiments on more image benchmarks, we see that the 01 loss has some potential for defending against adversarial attacks. This may be due to its discrete search space and (infinite) non-unique solutions.

II. METHODS

A. Coordinate descent

We describe in Algorithm 2 our local search based on coordinate descent. In brief, we start with a random w, make changes to it one coordinate at a time, determine the optimal w_0 for each setting of w with Algorithm 1, and stop when we reach a local minimum. There are several aspects of our local search worth discussing here.

First, we cycle the coordinates randomly. Since we modify only a single coordinate of w at a time we can update the projection w^Tx_i for all $i = 0..n-1$ in $O(n)$ time — this update is required to determine the optimal w_0 and the objective value. We perform at most 10 modifications to a given coordinate (as given by the loop '**for** $j = 1$ to 10 **do**') before considering the next one. This gives all coordinates a fair chance before we reach a local minimum. In the same loop we also update the objective if a better one is found and exit if modifying the coordinate does not improve the objective further. An alternative is to update the objective only after cycling through all the coordinates. However, we find our approach yields a faster search than the alternative while giving similar objective values.

Another aspect of our search is the determination of the optimal w_0 in Algorithm 1. For each setting of w_i (the i^{th} coordinate of w) we determine the optimal value of w_0 by considering all $O(n)$ settings of w_0 between sorted successive projected points w^Tx_k and w^Tx_{k+1} (see Algorithm 1). Since we modify w locally the new projection is similar to the previous (sorted) one and hence insertion sort (that we use for sorting the projection) takes much less than the worst case $O(n^2)$ time.

For an initial w it takes $O(n)$ to determine the optimal w_0. After that as we change w the new w_0 is less likely to be much different than the previous one. And so we don't need to consider all $O(n)$ points again to determine the optimal w_0. Instead, if the initial w_0 was found right after the projected point i then we only consider the range of points starting from $i-10$ to $i+10$ in the new projection to determine the new w_0. For a visual illustration see our toy search problem shown in Figure 2.

Algorithm 1 Opt

```
Input: $w^Tx_i \in R^d$ for $i = 0..n-1$ with labels $y_i \in \{+1,-1\}$, $start$, $end$
Output: Optimal $w_0 \in R$ with minimum 01 loss and the (balanced) 01 loss value $obj$
Procedure:
  for $i = start$ to $end - 1$ do
    $w_0' = \frac{w^Tx_i + w^Tx_{i+1}}{2}$
    if $y_i(w^Tx_i + w_0') == 0$ then
      If $y_i == 1$ then errorplus++
    else if $y_i(w^Tx_i + w_0') > 0$ then
      If $y_i == 1$ then errorplus−− else errorminus−−
    else if $y_i(w^Tx_i + w_0') < 0$ then
      If $y_i == 1$ then errorplus++ else errorminus++
    end if
    If $obj' = -1(\frac{errorplus}{n_+} + \frac{errorminus}{n_-})/2$ (a balanced version of our 01 loss objective 1) is lower than current best objective $obj$ then $obj = obj'$ and $w_0 = w_0'$.
  end for
  return $(w_0, obj)$
```

The w_{inc} parameter corresponds to the learning rate η in gradient descent optimizers. We implement a simple adap-

Algorithm 2 Coordinate descent

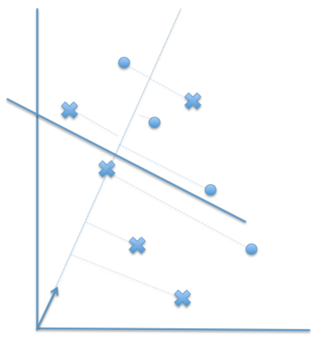
```
Input: Training data $x_i \in R^d$ for $i = 0..n-1$ with labels $y_i \in \{+1, -1\}$, $w_{inc} \in R$ (set to 100 by default), vector $w \in R^d$ and $w_0 \in R$
Output: Vector $w \in R^d$ and $w_0 \in R$
Procedure:
  1. Initialization: If $w$ and $w_0$ are null then let each feature $w_i$ of $w$ be randomly drawn from $[-1, 1]$. Set $\|w\|= 1$. Throughout our search we ensure that $\|w\|= 1$ by renormalizing each time $w$ changes.
  2. Initialize the number of misclassified points with negative $w^Tx_i$ be $errorminus = 0$ and with positive $w^Tx_i$ be $errorplus = 0$. These are later used in Algorithm 1 for fast update of our objective.
  3. Compute the initial data projection $w^Tx_i, \forall i = 0..n-1$, sort the projection with insertion sort, and initialize $(w_0, obj) = Opt(w^Tx, y, 0, n-1)$.
  4. Set $prevobj = \infty$.
  while $prevobj - obj > .01$ do
    Consider a random permutation of the $d$ feature indices.
    for $i = 0$ to $d - 1$ do
      Let $sign = 1$ if adding $w_{inc}$ to $w_i$ lowers the objective and -1 otherwise
      for $j = 1$ to $10$ do
        1. Set $prevobj = obj$
        2. Assume $w_0 = (w^Tx_j + w^Tx_{j+1})/2$ for some $j$.
        3. Set $start = w^Tx_{j-10}$ and $end = w^Tx_{j+10}$
        4. Set $w_i$ += $sign \times w_{inc}$ and compute data projection $w^Tx_i, \forall i = 0..n-1$, and sort the projection with insertion sort
        5. Set $(w_0, obj) = Opt(w^Tx, y, start, end)$
        6. Accept the new modified value of $w_i$ if it lowers the objective and update the 01 loss $obj$, otherwise we exit this loop
      end for
    end for
  end while
```

tive procedure that considers values of w_{inc} from the set $\{\pm 10^2, \pm 10^4, \pm 10^6\}$ and picks the one with the greatest decrease in our objective (see Objective 1).

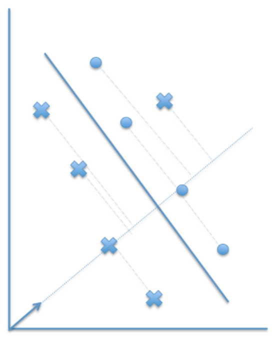
To understand why we have such large learning rates consider the toy example shown below in Figure 3. With small and constant step sizes we would be perpetually stuck in difficult local minima since 01 loss is non-unique and can have infinite solutions.

B. Stochastic coordinate descent

There is no guarantee our local search algorithm will return the global solution. The global solution may not even be unique. Once we reach a local minima we may choose the random restart approach and run the search again. An alternative is to rely on random batches of the training data across many iterations of the coordinate descent above so as to better explore the search space. We call this stochastic coordinate descent since this is essentially stochastic gradient descent [9] with the gradient descent replaced by our coordinate descent above. We describe this in detail in Algorithm 3. For a single run of our heuristic we save the best solution of w, w_0 across all random restarts and use that as the model parameters.



(a)



(b)

Fig. 2. Illustration of our coordinate search on a toy example. In (a) we show a hyperplane with an initial random normalized w. The dotted lines show where the projected points would lie on w. The optimal w_0 that minimizes our objectives lies just after the fourth projected point. In (b) we increase the x-coordinate of w thus modifying the orientation of the plane (we renormalize w after the orientation). In the new projection the optimal w_0 is also after the fourth projected point. Thus we don't need to perform a full $O(n)$ search after modifying w but instead considering just a few projected points around the previous w_0 is sufficient as a heuristic.

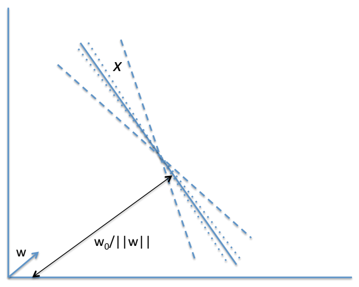


Fig. 3. The hyperplane in solid line is given by w and w_0 and it misclassifies the point x. The dotted hyperplanes are given by a small step size in the two coordinates of w and insufficient to cross over point x. The dashed hyperplanes are given by a larger step size that is sufficient to cross over x and give a potentially lower 0/1 loss.

Algorithm 3 Stochastic coordinate descent

```
Input: Feature vectors $x_i \in R^d$ with labels $y_i \in \{+1, -1\}$, number of random restarts $rr \in N$ (Natural numbers), number of iterations per random restart $it \in N$ (Natural numbers), batch size as a percent of training data $p \in [1, 100]$ , and $w_{inc} \in R$ (set to 100 by default)
Output: Total of $rr$ pairs of $bestw \in R^d$,$bestw_0 \in R$ after each random restart
Procedure:
  Set $j = 0$
  while $j < rr$ do
    1. Set $bestw = null, bestw_0 = null, bestloss = \infty$
    for $i = 0$ to $it$ do
      1. Randomly pick $p$ percent of rows as input training data to Algorithm 2 and run it to completion starting with the values of $w$ and $w_0$ from the previous call to it (if $i == 0$ we set $w = null, w_0 = null$).
      2. In the next step we calculate objectives on the full input training set
      if $objective(w, w_0) < objective(bestw, bestw_0)$ then
        Set $bestw = w$, $bestw_0 = w_0$, and $bestloss = objective(w, w_0)$
      end if
    end for
    2. Output $bestw$ and $bestw_0$
    3. Set $j = j + 1$.
  end while
  We output all $(bestw, bestw_0)$ pairs across the random restarts. We can use the pair with the lowest objective or the majority vote of all pairs for prediction.
```

C. Related work

Our coordinate descent and that of [13] differ in how the coordinates are optimized. In their case the authors project the data onto the current hyperplane (that is initially random), scale each projected value by the inverse of its projection on a random vector r, sort the projected values to determine the value that optimizes the 01 loss (call it α), and update the solution w by adding $\alpha \times r$. This is repeated for a fixed number of iterations. In our case we focus on optimizing Objective 1: we make an incremental change to a coordinate at a time, project the data onto the hyperplane, determine the optimal threshold w_0 for our objective, and repeat until the objective converges.

D. Experimental performance study on UCI datasets

In order to evaluate our stochastic coordinate descent (SCD) algorithm we study it on the below datasets in an experimental performance study. Our purpose here is to demonstrate that our SCD 01 loss works on real data and is comparable to the popular linear SVM in accuracy.

1) Datasets: We obtained 52 datasets from the UCI repository. The datasets include data from different sources such as biological, medical, robotics, and business. Some of the datasets are multi-class and since we are studying only binary classification in this paper we convert them to binary. We label the largest class to be -1 and remaining as +1 and ignore instances with missing values across the datasets. We provide our cleaned data with labels, splits, and a README file on the website http://web.njit.edu/~usman/scd01oss.

2) SCD 01 loss and SVM parameters: We compare our SCD 01 loss with random restarts $rr = 100$, number of iterations per random restart $it = 100$ and $p = 75\%$. We use the single best output $(w, w_0\ )$across all random restarts. For cross-validating linear SVM we select values of C from the set $\{100, 10, 1, .1, .01, .001, .0001, .00001, .000001\}$.

3) Train and test splits and measure of accuracy: For each dataset we create 10 random partitions into training and test datasets in the ratio of 90% to 10%. We use the number of misclassifications divided by the number of test datapoints as the measure of error throughout in our study.

E. Experimental performance study on image benchmarks CIFAR10 and STL10

In order to evaluate the adversarial sensitivity of SCD 01 loss to adversarial attacks we expose it to a black box attack method that we describe below in detail.

1) Black box adversarial attack: A black box adversarial attack approximates the model by giving it inputs and recording its outputs. It then uses the predictions as labels to train itself and then use its gradient to produce adversarial examples. We implement a double layer neural network with 200 hidden nodes in each layer and 10 nodes in the output one as the adversarial attacker (B). We fully describe our attack algorithm in Algorithm 4.

Algorithm 4 Shown here is a basic outline of the black box attack method [12]

```
Input: Model $M$ to be attacked, Adversarial attacker $B$, Feature vectors $x_i \in R^d$ with labels $y_i \in \{+1, -1\}$, number of epochs $ep \in N$ (Natural numbers)
Procedure:
  Set data $D = \{x_i, y_i\}$
  for $i = 0$ to $ep$ do
    1. Obtain predictions $y_i'$ of $D$ from black box model $M$
    2. Set adversarial training data $A$ to be $D$ except we replace each $y_i$ with the predicted label $y_i'$.
    3. Train attacker $B$ with $A$ as input training data
    4. With $B$'s gradient we produce adversarial examples.
    5. For each sample $a_i$ in $A$ create adversary $a_i = a_i + \lambda \nabla f$ where $\nabla f$ is the gradient of $B$ and $\lambda$ is randomly chosen from $[-.1, .1]$.
    6. Add new adversarial samples $\{a_i, y_i\}$ to $D$
    7. This effectively doubles the number of adversarial samples after each iteration. If we instead select a 100 random samples from $A$ then we increase the adversarial size by 100 as opposed to doubling.
  end for
```

2) Datasets: We study adversarial attacks on two image benchmarks:

- CIFAR10 [14]: Object recognition from 10 classes in 32×32 color images, training size of 50,000, test size of 10,000
- STL10 [15]: Object recognition from 10 classes in 96×96 color images, training size of 5000, and test size of 8000

3) Programs compared:

- Multi-class SCD 01 loss: We implement an one-vs-one [3] multi-class classification method on top of our SCD 01 loss. In the SCD 01 loss we use the same parameters as above.
- Multi-class linear SVM: We implement one-vs-all on top of the linear SVM with C=1.

4) Train and test splits and measure of accuracy: For the image benchmarks both train and test datasets are provided in advance. We use the number of misclassifications divided by the number of test datapoints as the measure of error throughout in our study.

III. Results

A. Classification on UCI datasets

In Table I we see the average and median error of SCD 01 loss and the cross-validated linear SVM. We see the linear SVM is better in both mean and median but the difference between them is not statistically significant. According to a simple t-test the p-values between their errors across the 52 datasets is 0.53 which is far from significant.

TABLE I
Mean and median error of SCD 01 loss and linear SVM

	SCD 01 loss	*Linear SVM*
Mean	13.5	13.2
Median	11.8	10.3

B. Black box adversarial attacks

1) CIFAR10: We generate adversarial samples on the CIFAR10 dataset and evaluate their accuracy on both the multi-class SCD 01 loss and multi-class linear SVM. We evaluate the SCD 01 loss with single and ten random restarts. For multiple random restarts we take the majority vote as the final output. Bootstrapping is a simple powerful method to boost model accuracy [3], [16]. As a matching counterpart to our SCD 01 loss random restarts we run a bootstrapped version of the linear SVM.

In Figure 4 we see the accuracy of CIFAR10 adversarial samples in both SCD 01 loss and linear SVM as the number of epochs progresses. We see both methods start losing accuracy as the black box method progresses, but interestingly we find SCD 01 loss to be relatively less sensitive. Both methods have about 40% accuracy on CIFAR10 and after epoch 10 both linear SVM and its bootstrapped version are at 0% accuracy. The SCD 01 loss also loses accuracy but with ten random restarts one demonstrates a greater defense.

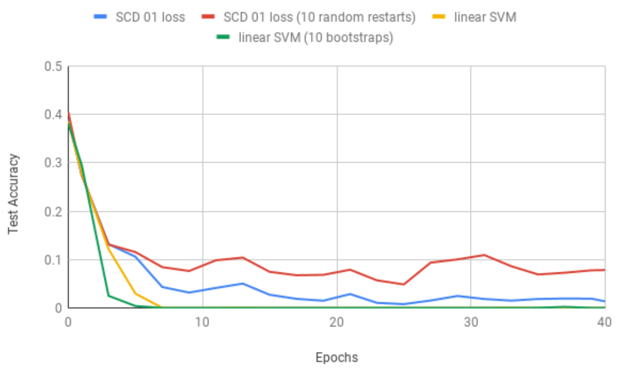


Fig. 4. CIFAR10 black box attack described in Algorithm 4. We double the number of adversarial samples per epoch.

In Figure 5 we see the adversarial attack with 100 new adversaries per epoch. Here the SCD 01 loss with 10 random restarts shows a greater resilience than the linear SVM.

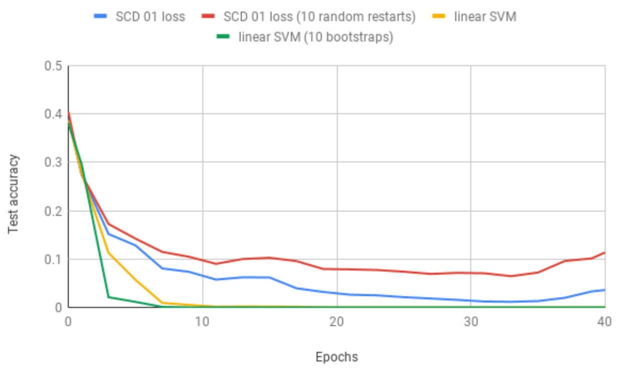


Fig. 5. CIFAR10 black box attack: we generate a 100 new adversarial samples per epoch.

2) STL10: We generate adversarial samples on the STL10 dataset and study their accuracy on SCD 01 loss and linear SVM. In Figure 6 we see that the linear SVM reaches 0% accuracy after the fifth epoch while SCD 01 loss remains above 10% at that epoch. However by epoch 15 SCD 01 loss is close to 0% accuracy but with 10 random restarts it is still at 10%.

Since these images are larger all components of the black box are slower and thus we have fewer epochs if we want to double the adversarial sample size. Thus we study accuracy on adversaries if we increase them by 100 after each epoch. In this way we can run the attack for more epochs to see if the accuracy becomes 0 for SCD 01 loss at some point.

In this setting we see that the ten random restarts SCD 01 loss remains well above 10% accuracy. Thus for STL10 we need to double adversarial samples after each epoch if we want to bring down the ten random restarts SCD 01 loss, but this requires more time and computation.

If we increase the number of adversarial samples by only 50 after each epoch we see an even smaller effect on SCD 01 loss. In fact in this setting even the linear SVM does not reach 0%

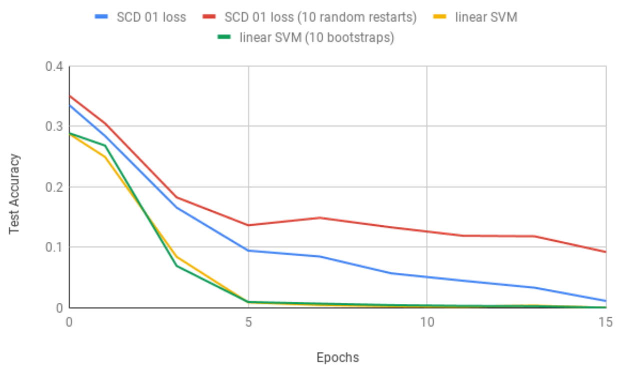


Fig. 6. STL10 black box attack: we double the number of adversarial samples per epoch.

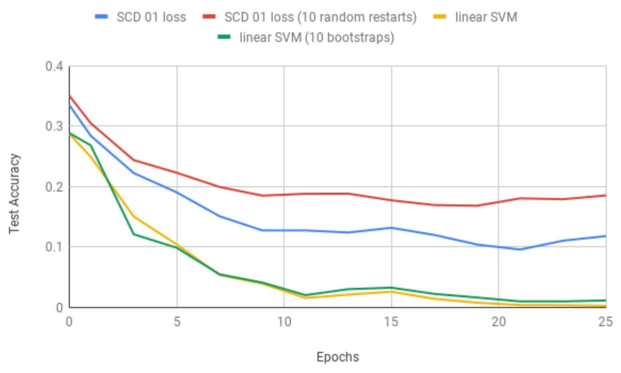


Fig. 7. STL10 black box attack: we generate a 100 new adversarial samples per epoch.

and stays above 5% accuracy. The ten random restarts SCD 01 loss however is above 20% accuracy here.

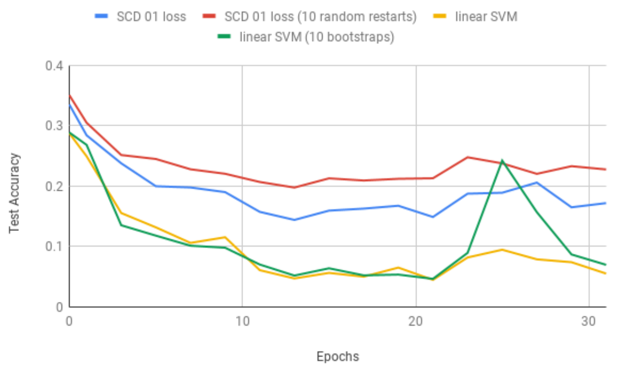


Fig. 8. STL10 black box attack: we generate 50 new adversarial samples per epoch.

IV. Discussion

We conjecture that the 01 loss's non-unique solution and discrete search space may be making it difficult for our double layer neural network to create a close approximation. This in turn makes it difficult to create adversarial samples targeting it. We also see that ten random restarts of SCD 01 loss makes it less sensitive to attacks. This may be attributed to the increased uncertainty in SCD 01 loss solutions arising from random samples of the training data. To better understand further experimental work is required such as using a more sophisticated attacker, experiments on other image benchmarks, and performance of SCD 01 loss with more random restarts. There is however no guarantee that a complex model will perform as a better attacker.

V. Conclusion

We present a stochastic coordinate descent heuristic that performs comparably to a trained cross-validated linear support vector machine but demonstrates greater defense against a black box adversarial attack on two image benchmarks. We conjecture this may be due to 01 loss's non-unique solutions and discrete loss and further work is required.

References

[1] Shai Ben-David, Nadav Eiron, and Philip M Long. On the difficulty of approximately maximizing agreements. *Journal of Computer and System Sciences*, 66(3):496–514, 2003.

[2] Shalev-Shwartz Shai, Ohad Shamir, and Karthik Sridharan. Learning linear and kernel predictors with the 0-1 loss function. *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, 22(3), 2011.

[3] Ethem Alpaydin. *Machine Learning*. MIT Press, 2004.

[4] Peter L. Bartlett, Michael I. Jordan, and Jon D. Mcauliffe. Large margin classifiers: Convex loss, low noise, and convergence rates. In S. Thrun, L.K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, pages 1173–1180. MIT Press, 2004.

[5] Tan Nguyen and Scott Sanner. Algorithms for direct 0–1 loss optimization in binary classification. In *Proceedings of The 30th International Conference on Machine Learning*, pages 1085–1093, 2013.

[6] Yueming Lyu and Ivor W Tsang. Curriculum loss: Robust learning and generalization against label corruption. *arXiv preprint arXiv:1905.10045*, 2019.

[7] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

[8] Weihua Hu, Gang Niu, Issei Sato, and Masashi Sugiyama. Does distributionally robust supervised learning give robust classifiers? *arXiv preprint arXiv:1611.02041*, 2016.

[9] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[10] D.J. Newman A. Asuncion. UCI machine learning repository, 2007.

[11] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[12] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 506–519. ACM, 2017.

[13] Ling Li and Hsuan-Tien Lin. Optimizing 0/1 loss for perceptrons by random coordinate descent. In *Neural Networks, 2007. IJCNN 2007. International Joint Conference on*, pages 749–754. IEEE, 2007.

[14] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[15] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223, 2011.

[16] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.