ECE 271B HW#3

Problem 1 $\phi(\cdot)$ monotonically decreasing $\rightarrow \phi'(\cdot) < 0$

a) $R_{emp}[g] = \frac{1}{n} \sum_i \phi(y_i g(x_i)) \quad y_i \in \{-1, 1\} \quad \{(x_i, y_i)\}_{i=1}^n$

In the lecture notes, we had $\phi(y_i g(x_i)) = \exp[-y_i g(x_i)]$
This problem will follow a similar procedure by first calculating the directional derivative.

$$D_u R_{emp}[g_t] = \left[\frac{d}{d\varepsilon} R_{emp}[g_t(x) + \varepsilon u(x)]\right]_{\varepsilon=0}$$

$$= \frac{d}{d\varepsilon} \frac{1}{n} \sum_i \phi\left(y_i g_t(x_i) + y_i \varepsilon u(x_i)\right]_{\varepsilon=0}$$

$$= \frac{1}{n} \sum_i \frac{d}{d\varepsilon}\left[\phi\left(y_i g_t(x_i) + y_i \varepsilon u(x_i)\right]_{\varepsilon=0}\right.$$

$$= \frac{1}{n} \sum_i \left[\phi'(y_i g_t(x_i) + y_i \varepsilon u(x_i)) \cdot y_i u(x_i)\right]_{\varepsilon=0}$$

$$= \frac{1}{n} \sum_i \left(\underbrace{\phi'(y_i g_t(x_i))}_{\text{doesn't depend on } u(x)} \underbrace{y_i u(x_i)}_{\text{depends on } u(x)}\right)$$

$\boxed{\bar{w}_i = -\phi'(y_i g_t(x_i))}$ Reweighting $\left(\begin{array}{l}\text{weights } w_i \geq 0, \\ \text{and } -\phi'(\cdot) \geq 0\end{array}\right)$

$$\alpha_t = -\nabla R_{emp}(g^{(t)}) = \boxed{\operatorname*{argmax}_{u \in U} \sum_i y_i u(x_i) \bar{w}_i}$$ Weak learner selection

b) The Perceptron loss: $\phi(v) = \max(-v, 0)$

From above, $\bar{w}_i = -\phi'(y_i g_t(x_i)) = \begin{cases} 1 \text{ for } y_i g_t(x_i) < 0 \\ 0 \text{ for } y_i g_t(x_i) \geq 0 \end{cases}$

$\phi'(v) = \begin{cases} -1 \text{ for } v < 0 \\ 0 \text{ for } v \geq 0 \end{cases}$

For the selection step, again from previous:

$$a_t = \underset{u \in U}{\operatorname{argmax}} \sum_i y_i u(x_i) \bar{w}_i \quad \text{where } w_i = \begin{cases} 1 & y_i g_t(x_i) < 0 \\ 0 & y_i g_t(x_i) \geq 0 \end{cases}$$

$$a_t = \underset{v \in V}{\operatorname{argmax}} \left(\sum_{i \mid y_i g_t(x_i) < 0} y_i u(x_i) \right).$$

c) For this scenario $w_i = \begin{cases} 1 & y_i g_t(x_i) < 0 \\ 0 & y_i g_t(x_i) \geq 0 \end{cases}$ as before

To derive the boosting algorithm, three steps should be calculated:

1) Weak learner selection $a_t(x)$
2) Step size w_t
3) Ensemble learner update $\boxed{g_{t+1}(x) = g_t(x) + w_t a_t(x)}$

① Weak learner selection:

Previously: $a_t = \underset{v \in V}{\operatorname{argmax}} \left(\sum_{i \mid y_i g_t(x_i) < 0} y_i u(x_i) \right)$

- Now, we have $u(x_i) = w^T x_i$
- Also, we have the constraint: $\|w\| = 1$

Therefore, $a_t = \underset{v \in V}{\operatorname{argmax}} \left(\sum_{i \mid y_i g_t(x_i) < 0} y_i w^T x_i \right)$ such that $\|w\| = 1$

To find the optimum, we should do constrained optimization, therefore we can use the Lagrangian:

$$L(w, \lambda) = \sum_{i \mid y_i g_t(x_i) < 0} y_i w^T x_i + \lambda (\|w\| - 1)$$

However generally using $\|w\|^2$ is a better option

$$L(w,\lambda) = \sum_{i \mid y_i g_t(x_i) < 0} y_i w^T x_i + \lambda(\|w\|^2 - 1)$$

As always, taking the gradient:

$\nabla_\lambda L(w,\lambda) = \|w^2\| - 1 = 0$, this just gives the same constraint

$$\nabla_w L(w,\lambda) = \sum_{i \mid y_i g_t(x_i) < 0} y_i x_i + 2\lambda w = 0, \quad 2\lambda w = -\sum_{i \cdots} y_i x_i$$

$$\boxed{w^* = -\frac{1}{2\lambda} \sum_{i \mid y_i g_t(x_i) < 0} y_i x_i} \quad \text{with constraint } \|w\| = 1$$

$$\|w^*\| = 1 = -\frac{1}{2\lambda} \left\| \sum_{i \mid y_i g_t(x_i) < 0} y_i x_i \right\| \rightarrow \text{call this } S_t$$

$$\lambda = \|S_t\|/2, \quad w^* = -\frac{1}{\|S_t\|} S_t = a_t$$

---

(2) For the step size:

$$w_t = \arg\min_w \text{Remp}[g_t + w a_t]$$

where $\text{Remp} = \frac{1}{n} \sum_i \phi(y_i g_t(x_i))$

$$w_t = \arg\min_w \frac{1}{n} \sum_i \phi(\cdots)$$

$$\left(y_i g_t(x_i) - \frac{1}{\|S_t\|} y_i w S_t^T x_i \right)$$

We have
$u(x) = w^T x$
therefore:

$$a_t(x) = w^{*T} x$$

$$\boxed{a_t(x) = -\frac{1}{\|S_t\|} S_t^T x}$$

With the update mentioned before, this concludes the algorithm.

d) Compare c) and Perceptron algorithm

For Boosting $\bar{w}_i = 0$ for $y_i g_t(x_i) \geq 0$ therefore it takes points $\in y_i g_t(x_i) < 0$ in its calculations. This means that points in which error is made are used for the updates. Perceptron has a similar condition in that each time an $x_i \in E$ is considered for the calculations.

The difference here is that Boosting takes every point in E into account while updating. Perceptron takes a single point each time. Therefore the updates are different, although the overall process is the same. Also in perceptron the learning rate η is irrelevant. However for boosting, rates w_t are very important and decide the voting weight of each weak learner. If we made updates for each single point in this question, we would get something very similar to the Perceptron. This is of course because we are using perceptron learner $u(x) = w^T x$ and also Perceptron loss $\phi(v) = \max(-v, 0)$.

e) No because we are using $\phi'(v)$ to decide on the weights. For 0-1 loss this derivative is 0 everywhere except at $v = 0$, where it is $-\infty$. If we used this function our weights $\bar{w}_i$ would simply blow up. Even if it did not blow up, we are just considering points for which $v = 0$, which is not very sensible.

$$1 - \frac{1}{1+e^{-v}} = \frac{1}{1+e^{v}}$$

Problem 2

$z \in \{0,1\}$, $\sigma(v) = \frac{1}{1+e^{-v}}$

a) Entropy Loss: $L(x,z) = -z \log \sigma[g(x)] - (1-z)\log(1-\sigma[g(x)])$

$y \in \{-1,1\}$, $y = 2z-1$

$L(x,y) = \log(1+e^{-yg(x)})$ → show this

Plugging the sigmoid into the loss function:

$$L(x,z) = +z\log(1+e^{-g(x)}) + (1-z)\log(1+e^{g(x)})$$

$$z \in \{0,1\} \Longrightarrow \left\{ \begin{array}{l} z=0,\ \log(1+e^{-g(x)}) \\ z=1,\ \log(1+e^{g(x)}) \end{array} \right\} L(x,z)$$

For $y = 2z-1$, $L(x,y) = \left\{ \begin{array}{l} y=-1,\ \log(1+e^{+g(x)}) \\ y=1,\ \log(1+e^{-g(x)}) \end{array} \right.$

Or, the simpler version $L(x,y) = \log(1+e^{-yg(x)})$

We have seen that $\phi(Y(x)) = \phi(yg(x))$ is the margin loss

In our case $Y(x) = yg(x)$ and $L(x,y) = \phi(Y(x))$

As we increase $Y(x)$ the function $\log(1+e^{-yg(x)})$ will decrease

So the neural networks should in general produce wider margin results. The loss function's optimization enforces bigger margins.

b) $L(x,y) = \log(1+e^{-yg(x)})$ derive boosting algorithm for part 1) a)

Again we have the following steps:

1- Find weights $\bar{w}_i$
2- The weak learner a_t
3- w_t step size
4- update equation

① The weights are $\bar{w}_i = \phi'(y_i, g_t(x_i))$ from the previous part

$$\phi' = L'(x,y) = \frac{-e^{-yg(x)}}{1+e^{-yg(x)}}$$

Again as $\phi' < 0$, the weights should be $\bar{w}_i = -\phi'(y_i, g_t(x_i))$

$$\boxed{\bar{w}_i = \frac{\exp(-yg(x))}{1+\exp(-yg(x))}}$$

② Selection of the weak learner

$$a_t = \underset{v \in V}{\operatorname{argmax}} \sum_i y_i u(x_i) \bar{w}_i = \underset{v \in V}{\operatorname{argmax}} \sum_i y_i u(x_i)(1-\sigma(y_i g_t(x_i)))$$

Where we can't go further without knowing V.

③ $w_t = \underset{w}{\operatorname{argmin}} R_{emp}[g_t + w a_t]$, $R_{emp} = \frac{1}{n}\sum_i \phi(y_i g(x_i))$

$$w_t = \underset{w}{\operatorname{argmin}} \frac{1}{n} \sum_i \phi\left(y_i\left(g_t(x_i + w a_t)\right)\right)$$

$$\phi(v) = \log(1+e^{-v})$$

$$w_t = \underset{w}{\operatorname{argmin}} \sum_i \log\left(1 + \exp\left[-y_i(g_t(x_i + w a_t))\right]\right)$$

④ The update is, as before: $g_{t+1}(x) = g_t(x) + w_t a_t(x)$

$$w_t = \underset{w}{\operatorname{argmin}} \sum_i \exp(-y_i(g_t(x_i + w a_t)))$$

because of argmin, we can drop the log and 1 term.

— This w_t is the same as the exponential loss case

— For exp. loss $a_{t_1} = \underset{v \in V}{\operatorname{argmax}} \sum_i y_i u(x_i) \exp[-y_i g_t(x_i)]$

For our loss

$$a_{t_2} = \underset{v \in V}{\operatorname{argmax}} \sum_i y_i u(x_i)(1 - \sigma(y_i g_t(x_i)))$$

$1 - \sigma$

$$a_{t_2} = \underset{v \in V}{\operatorname{argmax}} \sum_i y_i u(x_i) \log(1 + \exp(-y_i g_t(x_i)))$$

$$= \text{//} \sum_i y_i u(x_i) \exp[-y_i g_t(x_i)]$$

So $a_{t_1} = a_{t_2}$

— For the weights:

$$w_i = -\phi'(y_i g_t(x_i))$$

$$\frac{e^{-x}}{1+e^{-x}} = \frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}$$

$$= 1 - \sigma(x)$$

For our case: $-\phi'(y_i g_t(x_i)) = \sigma(-y_i g_t(x_i)) + 1$

For exponential case: $-\phi'(y_i g_t(x_i)) = \exp(-y_i g_t(x_i))$

$\sigma(x) = \frac{1}{1+e^{-x}}$ and e^{-x} behave similarly for $x > 0$ because they both decay to zero. However for $x < 0$, e^{-x} keeps increasing as x is decreasing, $\sigma(x) = \frac{1}{1+e^{-x}}$, on the other hand is bounded.

After a certain threshold of negative loss, the cross entropy case treats points equally. The exponential case however, gives more weight to a point the more negative loss it has. If a point cause very large loss, the algorithm may desperately work on it and do drastic changes. It is better when the solutions are more general for all points, instead of being too sensitive to singular points in the dataset. So, for cases that include many large negative loss points, cross entropy boost algorithms would perform better.

Problem 3

AdaBoost, stumps $u(x, j, t) = \begin{cases} 1, & x_j \geq t \\ -1, & x_j < t \end{cases}$

$U \rightarrow$ set of weak learners such that t belongs to a predefined set of T thresholds.

- For each weak learner: $u'(x, j, t) = -u(x, j, t)$ is the opposite polarity w.l.

a) There will be $d \times T$ learners to choose from d dimensions and T thresholds. Also we may score more than 50% in error so we will include the polars:

weak learners $= 2 \times d \times T$, The boosting algorithm is iterative, this $O(dT)$ computation will be done in each

iteration. We can choose the number of iterations (the only parameter we actually control). Let's call it I.

The complexity will be $O(dTI)$

b) weak learners: $u(x; t) = \text{sgn}[w^T x - t]$, w is computed by LDA.

From the LDA lecture slides, we can use either LDA or RDA as follows:

$$\left.\begin{array}{l} \text{LDA} \rightarrow S_W^{-1} S_B \\ \text{RDA} \rightarrow [S_W + \gamma I]^{-1} S_B \end{array}\right\} \text{After calculating one of these matrices, } w^* \text{ is the largest eigenvalue.}$$

$$S_B = (\mu_1 - \mu_{-1})(\mu_1 - \mu_{-1})^T, \quad S_W = \Sigma_1 + \Sigma_{-1}$$

To find the sample means, we should do a weighted average using the w_i's for each part.

$c \in \{-1, 1\}$ $\mu_c' = \sum_{i, y_i = c} \bar{w}_i x_i$, however this is may not be the exact μ_c if the weights don't add up to 1. Therefore,

$$c = \sum_i w_i', \quad \mu_c = \frac{\mu_c'}{c}, \text{ if } c = 1 \; \mu_c = \mu_c' \text{ as before.}$$

$$\Sigma_c' = \sum_{i, y_i = c} (x_i - \mu_c)(x_i - \mu_c)^T, \quad \Sigma_c = \frac{\Sigma_c'}{c}$$

The next step is to plug in these values to S_B & S_W and finally do the LDA or RDA calculations.

c) Repeat b) for weak learners implemented with a Gaussian classifier μ_i, Σ_i, π_i $i \in \{-1, 1\}$ $i \in \{-1, 1\}$
Parameters are learned by ML.

This time our $u(x) = BDR$, assuming the covariances are equal the BDR will be:

$$i^* = \underset{i}{\operatorname{argmax}}\ P(y=i \mid x)$$

$$P(y=i \mid x) = \frac{P(x \mid y=i)\, P(y=i)}{P(x) \rightarrow \text{constant}}$$

$$i^* = \underset{i}{\operatorname{argmax}}\ P(x \mid y=i)\, P(y=i)$$

$$P(y=i) = \pi_i, \quad P(x \mid y=i) = \text{Gaussian}(x, \mu_i, \Sigma_i)$$

$$i^* = \underset{i}{\operatorname{argmax}} \left(\pi_i\, G_i(x, \mu_i, \Sigma_i) \right)$$

For the class probabilities, what we know about the classes basically are the points themselves. We can use:

$$\pi_i = \frac{\#\text{points that are in class } i}{\#\text{total points}}$$

Problem 4

Decomposing signal $f(t)$ into a combination of functions $a_n(t)$ so that:

$$f(t) = \sum_n w_n a_n(t) \quad , \quad a_i(t) \text{ are not linearly independent}$$

Empirical risk is $R_{emp}(\alpha) = \frac{1}{n} \sum_{i=1}^{n} L[y_i, g(x_i)]$

we are given $L(y, g(x)) = (y - g(x))^2$

Therefore $R_{emp}[g(x)] = \frac{1}{n} \sum_{i=1}^{n} (f(t_i) - g(t_i))^2$

From where we can derive the boost algorithm.
At iteration n, we will have g_n, a_n, w_n and r_n.

We first calculate the directional derivative:

$$\nabla_u R_{emp}[g_n(t)] = \left[\frac{d}{d\varepsilon} R_{emp}[g_n(t) + \varepsilon u(t)]\right]_{\varepsilon=0}$$

$$= \frac{1}{n} \sum_i \frac{d}{d\varepsilon} \left(f(t_i) - g_n(t_i) - \varepsilon u(t_i)\right)^2 \Big|_{\varepsilon=0}$$

$$= -\frac{1}{n} \sum_i 2 \left(f(t_i) - g_n(t_i) - \varepsilon u(t_i)\right) \cdot u(t_i) \Big|_{\varepsilon=0}$$

$$= -2 \cdot \frac{1}{n} \sum_i \left(f(t_i) - g_n(t_i)\right) u(t_i)$$

We should look for the largest derivative to find an → direction of

$$a_n(t) = \underset{a(t) \in \nabla}{\operatorname{argmax}} \; -\frac{1}{n} \sum_i 2 \left(f(t_i) - g_n(t_i)\right) a(t_i)$$

$$= \underset{a(t) \in \nabla}{\operatorname{argmax}} \left| \sum_i \left(f(t_i) - g_n(t_i)\right) a(t_i) \right|$$

Step Size:

$$w_n(t) = \underset{w}{\operatorname{argmin}}\ R_{emp}\left[g_n(t) + w\, a_n(t_i)\right]$$

$$w_n(t) = \underset{w}{\operatorname{argmin}}\ \frac{1}{n}\sum \left(f(t_i) - g_n(t_i) - w\, a_n(t_i)\right)^2$$

Looking at the derivative to find the minimum:

$$\nabla w_n(t) = -2\sum \left(f(t_i) - g_n(t_i) - w\, a_n(t_i)\right) a_n(t_i) = 0$$

$$w \sum a_n^2(t_i) = \sum \left(f(t_i) - g_n(t_i)\right) a_n(t_i)$$

$$w^* = \frac{\sum \left(f(t_i) - g_n(t_i)\right) a_n(t_i)}{\sum a_n^2(t_i)}$$

To comply with the problem definition, we can write this as:

$$w = -\left| \frac{\sum \left(f(t_i) - g_n(t_i)\right) a_n(t_i)}{\sum a_n^2(t_i)} \right|$$

The boosting update is: $g_{n+1}(t) = g_n(t) + w_n a_n(t)$

The only unknown left from the matching pursuit algorithm is r_n:

$$a_n(t) = \underset{a(t) \in D}{\operatorname{argmax}} \left| \sum_i r_n(t_i)\, a_n(t_i) \right|$$

$$\Updownarrow$$

$$a_n(t) = \underset{a(t) \in D}{\operatorname{argmax}} \left| \sum_i \left(f(t_i) - g_n(t_i)\right) a_n(t_i) \right|$$

$$r_n(t) = f(t) - g_n(t)$$

We already found the a_n and w_n steps.

The last step is $r_{n+1}(t) = f(t) - g_{n+1}(t) = r_n(t) + g_n(t) - g_{n+1}(t)$

$g_n(t) - g_{n+1}(t) = w_n a_n(t)$

$r_{n+1}(t) = r_n(t) - w_n a_n(t)$, we derived all the steps.

For the computer assignment section of this report, I will first answer the questions and then include the required graphs. The organizations of the graphs, after the answers, will be as follows:

Part A) 1 Page

Part B) 1 Page

Part C) 2 Pages

Part D) 1 Page

Answers:

Part A)

The graphs of the train & test errors are provided below, after the answers. The final classifier probability of error was 0.0933.

Part B)

Looking at the graphs, we can see that as iteration counts increases, the #points with positive margin increases. This is as expected, because the error rates are dropping with more iterations. Also, the span of the margins increases as iteration count t increases. Comparing the ensemble classifiers at t=5 and t=250, we see that the maximum positive value of the margin increases considerably.

We can also see that there are always points with negative margin, even if the classifier is improving itself. Some points are too hard to classify without significantly changing the classifier. Even if they have high magnitude negative margin, these points are sacrificed for the overall success of the ensemble learner. So, the learner knowingly mislabels them to be able to correctly classify many other, easier points.

Part C)

About the first graph of Part C:

My first observation from the max. valued weight is that it belongs to a very small set of the 20000 possible indices. It bounces around between 6-7 distinct indices at most. Therefore some points always remain hard to classify by the classifier. As mentioned above, these points are probably sacrificed by the classifier, as they are too hard to classify.

About the second graph of Part C:

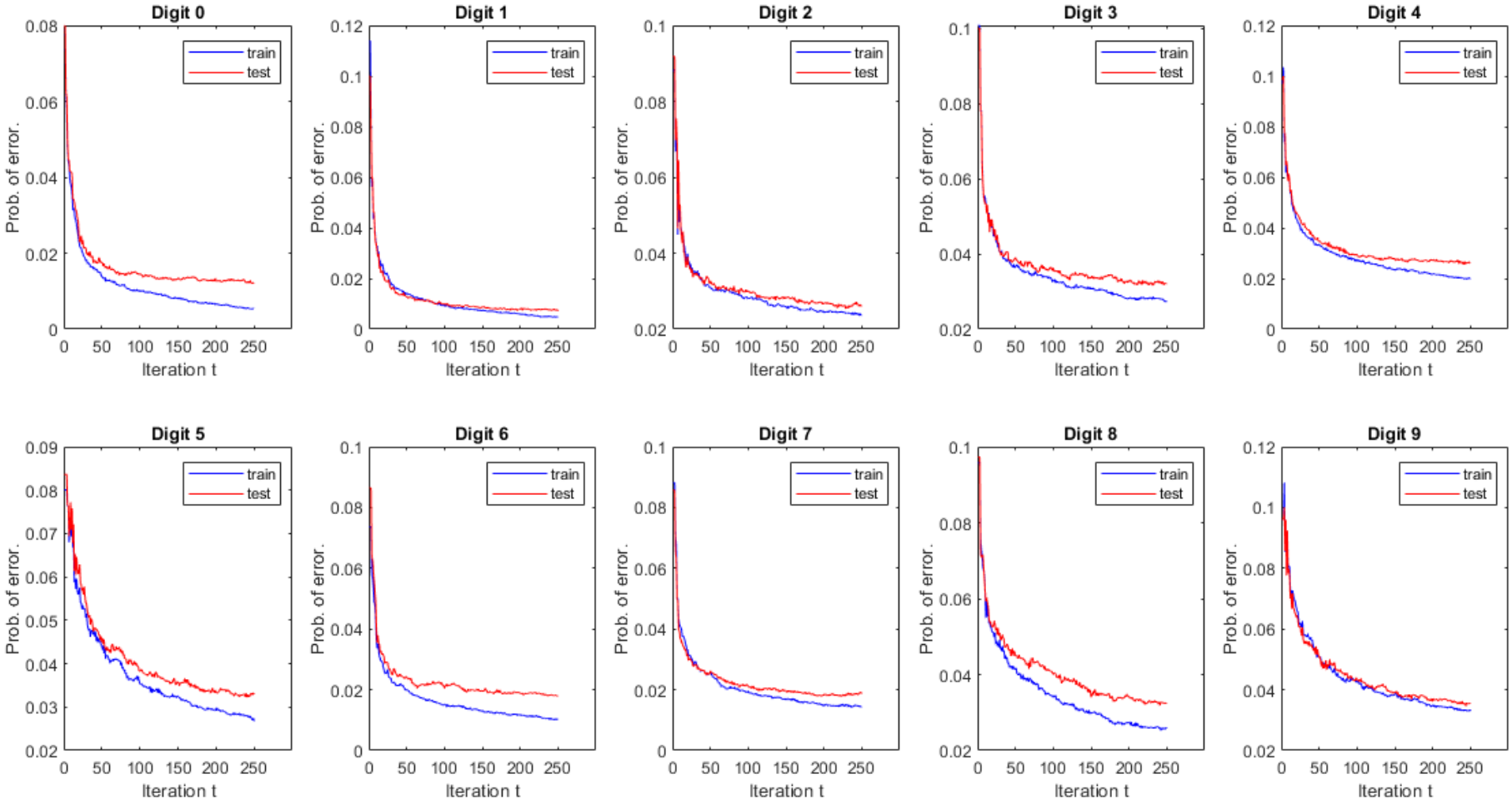
From the images of the data points corresponding to the top 3 weight indices, we can see that these digits are indeed hard to recognize. They usually belong to the base class, although there are some that are from different classes. These ones from different classes are look quite similar to the original digit. It is quite hard to discern them, even by human inspection. Therefore they always get high weights, which is directly related to how hard it is to correctly classify a point.

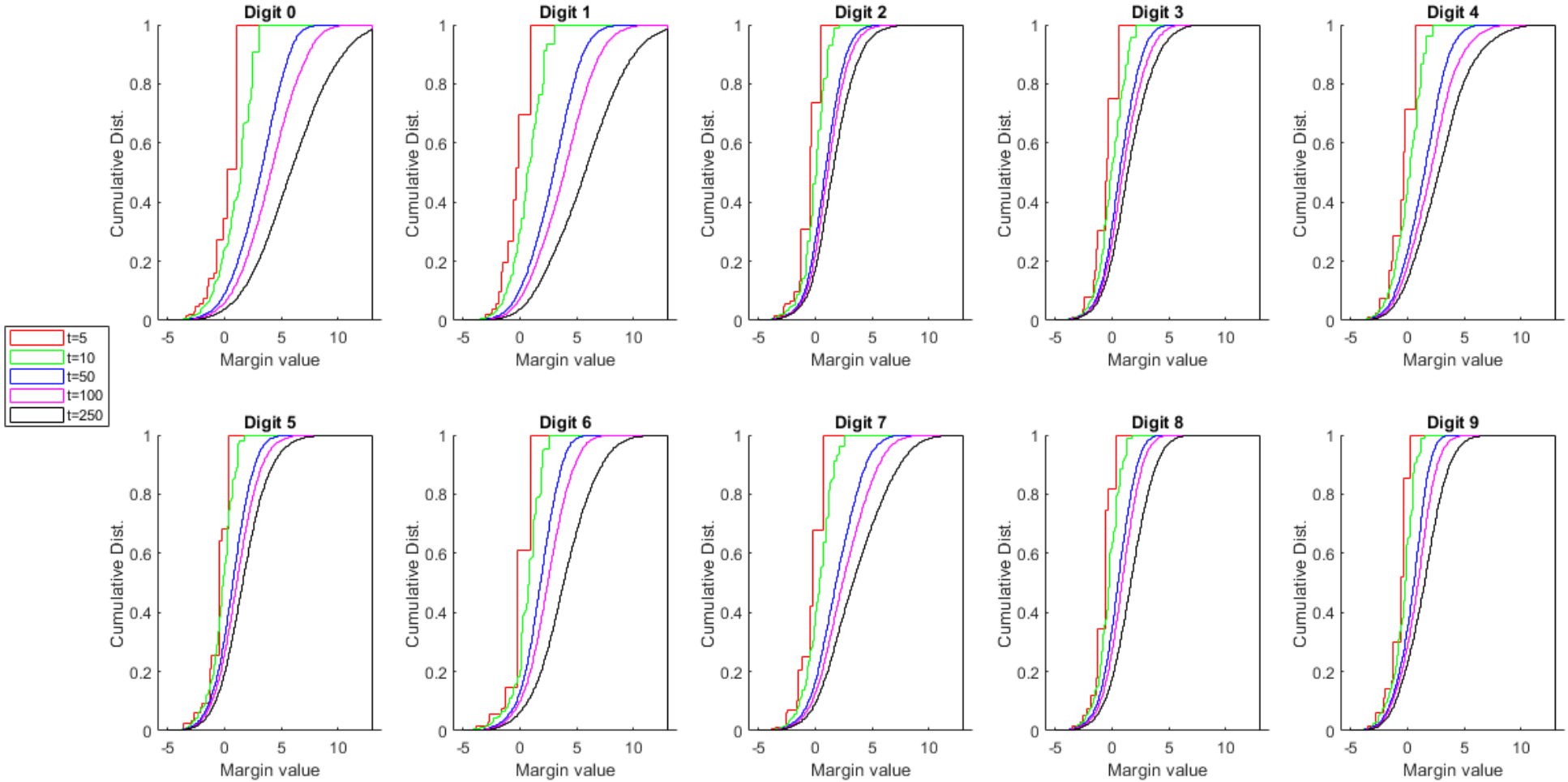
Part D)

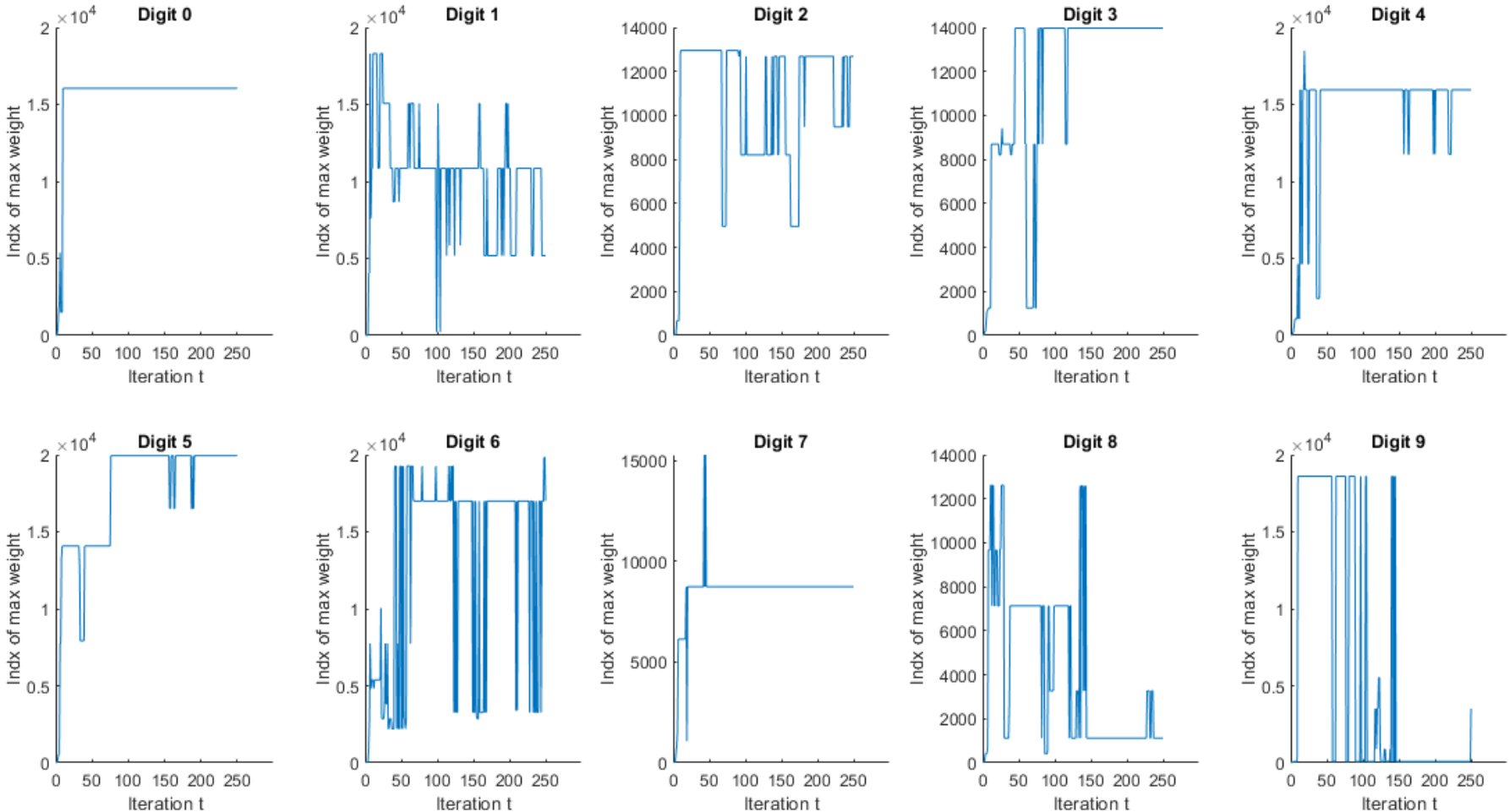
It is easier to first comment on the gray areas of these images. These areas are not mostly not thresholded by the classifiers, because they reveal no information about any digit. Mostly the border sections of these images are gray. There is a blurry section in the middle of each image. The black & white dots seem to show a very faint outline of the original digits. However the outline is very hard to see and very blurry.

Intuitively, it makes sense that the dimensions that are most used are the ones on the borders of each digit. Because the borders of a digit give the most information about that particular digit. However the digit drawings for a single digit can be vastly different among examples. Also, our weak learners are just decision stumps, which are quite basic. These may be the reasons why the outputs we get are very blurry and hard to understand.

Below, I will include the graphs required for question 5.


Digit 0
Digit 1
Digit 2
Digit 3
Digit 4
Digit 5
Digit 6
Digit 7
Digit 8
Digit 9
train
test
Prob. of error.
Iteration t


Digit 0
Digit 1
Digit 2
Digit 3
Digit 4
Digit 5
Digit 6
Digit 7
Digit 8
Digit 9
Cumulative Dist.
Margin value
t=5
t=10
t=50
t=100
t=250


Digit 0
Digit 1
Digit 2
Digit 3
Digit 4
Digit 5
Digit 6
Digit 7
Digit 8
Digit 9
Indx of max weight
Iteration t
×10^4

Digit 0

Digit 1

Digit 2

Digit 3

Digit 4

Digit 5

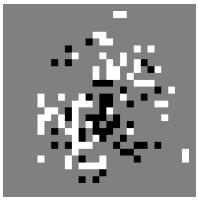
Digit 6

Digit 7

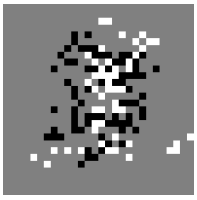
Digit 8

Digit 9

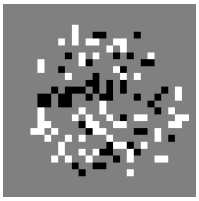
Digit 0



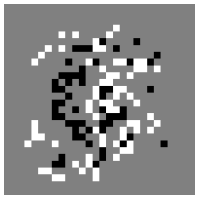
Digit 1



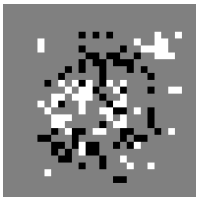
Digit 2



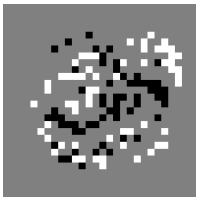
Digit 3



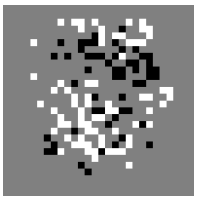
Digit 4



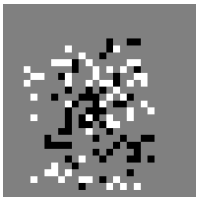
Digit 5



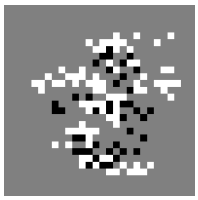
Digit 6



Digit 7



Digit 8



Digit 9

```
clc;
clear all;
% trainImg = 'train-images.idx3-ubyte';
% trainLabel = 'train-labels.idx1-ubyte';
%
% testImg = 't10k-images.idx3-ubyte';
% testLabel = 't10k-labels.idx1-ubyte';
%
% [train_imgs, train_labels] = readMNIST(trainImg, trainLabel, 20000,
 0);
% [test_imgs, test_labels] = readMNIST(testImg, testLabel, 10000, 0);
%
% save HW3_Data
load('HW3_Data.mat')

digits_train = zeros(20000,10);
%Classifier for class digit = 0
for class_index = 0:9
    digits_train(:,class_index+1) = (train_labels == class_index)*2 -
 1;
end

digits_test = zeros(10000,10);
%Classifier for class digit = 0
for class_index = 0:9
    digits_test(:,class_index+1) = (test_labels == class_index)*2 - 1;
end

train_errors = zeros(250,10);
test_errors = zeros(250,10);

wi_train = zeros(20000,10);

gt_train = zeros(20000,10);
gt_test = zeros(10000,10);
thresholds = repmat(linspace(0,1,51),1,2);

highest_weights = zeros(250,10);
margin = zeros(5,20000,10);

train_counts = zeros(10,1);
test_counts = zeros(10,1);

for i = 1:10
    train_counts(i) = sum(train_labels == i-1);
    test_counts(i) = sum(test_labels == i-1);
end

no_iter = 1;

tic
count = 0;
```

```
for iteration = 1:no_iter
    % Iteration will start here
    iteration
    %Updating weights
    wi_train = exp(-(digits_train.*gt_train));

    [val, indx] = max(wi_train);
    highest_weights(iteration, :) = indx;

    if (ismember(iteration, [5, 10, 50, 100, 250]))
        count = count + 1;
        %margin = y*g(x)
        margin(count,:,:) = digits_train.*gt_train;
    end

    %Trying each possible weak learner
    for class_index = 1:10
        at = inf*ones(102,784);
        for i = 1:51
            threshold = thresholds(i);
            for dim = 1:784
                ux = (train_imgs(:,dim) >= threshold)*2 - 1;
                ux_polar = ux*(-1);
                %errs(i,dim) = (digits_train(:,class_index) ~=
 ux)'*wi_train(:,class_index);
                %errs(i+51,dim) = (digits_train(:,class_index) ~=
 ux_polar)'*wi_train(:,class_index);
                at(i,dim) = (digits_train(:,class_index).*
 ux)'*wi_train(:,class_index);
                at(i+51,dim) = (digits_train(:,class_index) .*
 ux_polar)'*wi_train(:,class_index);
            end
        end

        % Choosing the weak learner with the smallest error
        [~,idx] = max(at(:));
        [row,col]=ind2sub(size(at),idx);
        threshold = thresholds(row);

        % Step size calculation
        sgn = (row <= 51)*2 - 1; %Choosing the original or the polar
        ux_train = ((train_imgs(:,col) >= threshold)*2 - 1)*sgn;
        sum1_train = (digits_train(:,class_index) ~=
 ux_train)'*wi_train(:,class_index);
        sum2_train = sum(wi_train(:,class_index));
        E_train = sum1_train / sum2_train;
        wt = (1/2)*(log((1-E_train)/E_train));

        % Update
        gt_train(:,class_index) = gt_train(:,class_index) +
 wt*ux_train;

        % Train error calculations
```

```
        train_err  = (gt_train(:,class_index) >= 0) ~=
 (digits_train(:,class_index) >= 0);
        train_errors(iteration, class_index) = sum(train_err)/20000;

        % Test error calculations
        % Update

        ux_test = ((test_imgs(:,col) >= threshold)*2 - 1)*sgn;
        gt_test(:,class_index) = gt_test(:,class_index) + wt*ux_test;

        test_err = (gt_test(:,class_index) > 0) ~=
 (digits_test(:,class_index) > 0);
        test_errors(iteration, class_index) = sum(test_err)/10000;
    end
end
toc

train_errors(train_errors > 1) = 1;
test_errors(test_errors > 1) = 1;

[~, decisions] = max(gt_test,[],2);
decisions = decisions - 1;
err_rate = sum(test_labels ~= decisions) / size(decisions,1);
fprintf('The error rate of the final classifier on test set is %f:
 ',err_rate);

%save HW3_Data_Loop_Complete_NewVersion

figure()
for i = 1:10
    subplot(2,5,i);
    plot(1:250,train_errors(:,i)/10,'b')
    hold on
    plot(1:250,test_errors(:,i)/10,'r')
    title(sprintf('Digit %d', i-1)); xlabel('Iteration t');
 ylabel('Prob. of error.');
    legend('train','test');
    xticks(0:50:250)
end

figure()
for i = 1:10
    subplot(2,5,i);
    hold on
    edges = linspace(-5,13,500);
    h1 =
 histogram(margin(1,:,i),500,'BinEdges',edges,'Normalization','cdf','DisplayStyle'
    h2 =
 histogram(margin(2,:,i),500,'BinEdges',edges,'Normalization','cdf','DisplayStyle'
    h3 =
 histogram(margin(3,:,i),500,'BinEdges',edges,'Normalization','cdf','DisplayStyle'
    h4 =
 histogram(margin(4,:,i),500,'BinEdges',edges,'Normalization','cdf','DisplayStyle'
```

```
    h5 =
 histogram(margin(5,:,i),500,'BinEdges',edges,'Normalization','cdf','DisplayStyle'
    title(sprintf('Digit %d', i-1)); xlabel('Margin value');
 ylabel('Cumulative Dist.');
end
legend('t=5','t=10','t=50','t=100','t=250');

figure()
for i = 1:10
    subplot(2,5,i);
    hold on
    plot(highest_weights(:,i));
    title(sprintf('Digit %d', i-1)); xlabel('Iteration t');
 ylabel('Indx of max weight');
    xticks(0:50:250)
end

figure()
for i = 1:10
    for j = 1:3
        x = highest_weights(:,i);
        u = unique(x);
        [n,bin] = hist(x,u);
        [~,indices] = sort(-n);
        hardest_indices = u(indices(1:3));

        loop_image = [];
        for k = 1:3
            loop_image = [loop_image;
 reshape(train_imgs(hardest_indices(k),:),[28 28])];
        end
        subplot(5,2,i), imshow(loop_image')
        title(sprintf('Digit %d', i-1));
    end
end

max_iter = 55;
figure();
for class = 1:10
    Threshold_matrix = ones(28,28)*128;
    for i = 1:max_iter
        threshold = at_save(class,i,1);
        dim = at_save(class,i,2);
        sign = (threshold <=51);
        row = floor(dim/28) + 1;
        column = rem(dim,28);
        if column == 0
            column = 28;
        end
        Threshold_matrix (row,column) = sign*255;
    end

    subplot(4,3,class), imshow(Threshold_matrix, [0 255])
    title(sprintf('Digit %d', class-1));
```

```
end
```

Published with MATLAB® R2018b