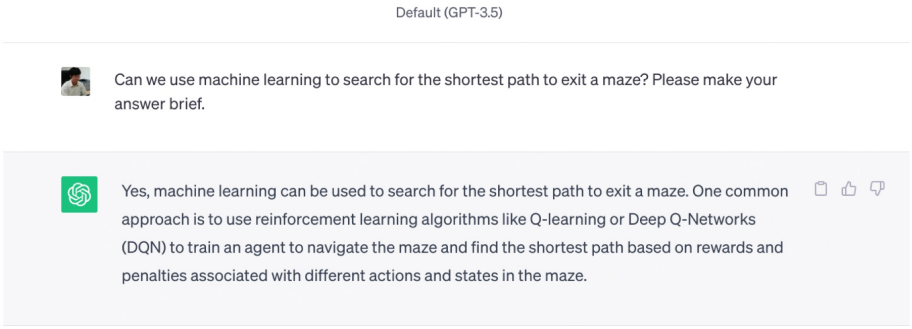
1. (20 points) Describe an application of self-supervised learning with 10-20 English or Chinese sentences.

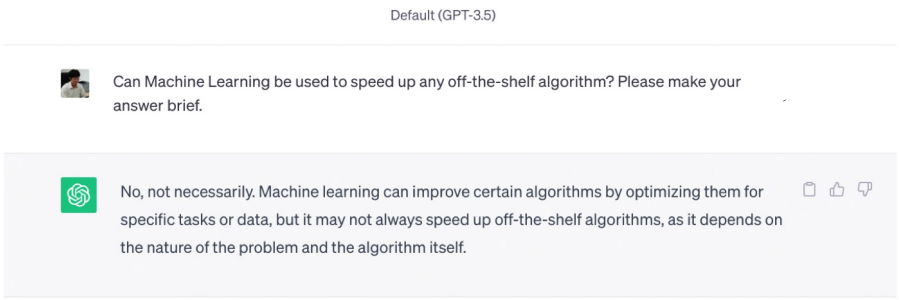
One of the famous application of self-supervised learning is healthcare. It's extremely difficult to obtain labled data in biomedical domain, so the self-supervised learning can perfectly deal with the issue. For instance, the image recognition can differentiate every organs and recognize the cancers that are too subtle for human eye. More over, it make accurate predictions and act immediately. Next, self-supervised learning are used to distinguish between L and H phenotypes of COVID-19 baied on patients lung X-rays, which help professionals make better treatment decisions. Many professionals doubt that AI can make precise decision. We still need more mature data management and data governance pratices to ensure the security.

2. (20 points) The following answer was generated when asking chatGPT "*Can we use machine learning to search for the shortest path to exit a maze? Please make your answer brief.*" Do you agree or disagree with the answer? Provide your reasoning with 10-20 English or Chinese sentences.



I agree with the answer that chatGPT reply. We can always solve the maze as long as we try enough, but the time we spend or the length of the route can be dedferent. On the other hand, the information is implicit, we can't know the exactly yes or no. So, we can use reinforcement learning algorithms, which can determine whether the route is the shortest or spend the least time by the outcome, such as the time it solve the maze. According to the answer, they train an agent to find the shortest path based on reward and penalties associated with different action seems to be the best way to search for the shortest way.

3. (20 points) The following answer was generated when asking chatGPT "*Can Machine Learning be used to speed up any off-the-shelf algorithm? Please make your answer brief.*" Somehow Google DeepMind announced that they discovered faster sorting algorithms with Machine Learning in June 2023 (`https://www.nature.com/articles/s41586-023-06004-9`). The discovery seems contradictory to the answer from chatGPT. Why is that? Provide your reasoning with 10-20 English or Chinese sentences.



The reason that chat GPT reply the contradictory answer is that the latest updating of information for language model is 2021. The principle of chat GPT is un-supervised pre-train and fine-tune with both supervised learning and reinforcement learning. Chat-GPT get the information from internet to pre-train. Therefore, chat-GPT was pre-train only by the information that exist before 2021, However, the news of Google Deep mind was announced at 2023, which is later than the pre-train of chat-GPT. So the answer of chat GPT would be contradictory to the news. Likewise, chat GPT can't answer the latest news or the event that happend after 2021.

4. (20 points) For the PLA algorithm introduced in class (page 8/22 of Lecture 2), assume that T_+ mistakes happened during $y_{n(t)} = 1$, and T_- mistakes happened during $y_{n(t)} = -1$. Express w_0, the zero-th component of the PLA solution, in terms of T_+ and T_-, and prove the result.

For $t = 0, 1 \cdots$ $\quad w_{t+1} = w_t + y_{n(t)} x_{n(t)}$

$\Rightarrow w_{0(t+1)} = w_{0t} + y_{n(t)} \times 1 \quad (x_0 = 1)$

When $y = +1$ $\quad \Delta w_0 (T_+) = 1 \times T_+$

when $y = -1$ $\quad \Delta w_0 (T_-) = -1 \times T_-$

assume $w_{00} = 0$

$\Rightarrow w_0 = T_+ - T_-$

#

5. (20 points) Consider online spam detection with machine learning. We will represent each email $\mathbf{x}$ by the distinct words that it contains. In particular, assume that there are at most m distinct words in each email, and each word belongs to a big dictionary of size $d \geq m$. The i-th component x_i is defined as $[\![\text{word } i \text{ is in email } \mathbf{x}]\!]$ for $i = 1, 2, \ldots, d$, and $x_0 = 1$ as always. We will assume that d_+ of the words in the dictionary are more spam-like, and $d_- = d - d_+$ of the words are less spam-like. A simple function that classifies whether an email is a spam is to count $z_+(\mathbf{x})$, the number of more spam-like words with the email (ignoring duplicates), and $z_-(\mathbf{x})$, the number of less spam-like words in the email, and classify by

$$f(\mathbf{x}) = \text{sign}(z_+(\mathbf{x}) - z_-(\mathbf{x}) - 0.5).$$

That is, an email $\mathbf{x}$ is classified as a spam iff the integer $z_+(\mathbf{x})$ is more than the integer $z_-(\mathbf{x})$.

Assume that f can perfectly classify any email into spam/non-spam, but is unknown to us. We now run an online version of Perceptron Learning Algorithm (PLA) to try to approximate f. That is, we maintain a weight vector $\mathbf{w}_t$ in the online PLA, initialized with $\mathbf{w}_0 = \mathbf{0}$. Then for every email $\mathbf{x}_t$ encountered at time t, the algorithm makes a prediction $\text{sign}(\mathbf{w}_t^T \mathbf{x}_t)$, and receives a true label y_t. If the prediction is not the same as the true label (i.e. a mistake), the algorithm updates $\mathbf{w}_t$ by

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_t \mathbf{x}_t.$$

Otherwise the algorithm keeps $\mathbf{w}_t$ without updating

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t.$$

Prove or disprove that $(4d+1)(m+1)$ upper bounds the number of mistakes that the online PLA can make for this spam classification problem.

Note: For those who know the bag-of-words representation for documents, the representation we use is a simplification that ignores duplicates of the same word.

$$T = \frac{W_f \cdot W_T}{\|W_f\| \cdot \|W_T\|} \leq 1 \qquad X = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix} \rightarrow X_0 \quad \left.\begin{matrix} \\ \\ \\ \\ \\ \end{matrix}\right\} m+1 \text{개} \qquad W = \begin{bmatrix} -0.5 \\ 1 \\ -1 \\ -1 \\ 1 \\ \vdots \end{bmatrix} \left.\begin{matrix} \\ \\ \\ \\ \\ \end{matrix}\right\} (d+1) \text{개}$$

$$R = \max_n \|X_n\| = \sqrt{m+1} \qquad R^2 = m+1$$

$$\rho^2 = \left(\frac{\min_n y_n W_f x_n}{\|W_f\|}\right)^2 = \left(\frac{\frac{1}{2}}{\sqrt{\frac{1}{4}+d}}\right)^2 = \frac{\frac{1}{4}}{\frac{1}{4}+d} = \frac{1}{4d+1}$$

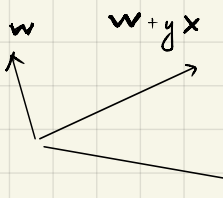
$$T_{max} = \left(\frac{R^2}{\rho^2}\right) = (m+1)(4d+1)$$

6. (20 points) Before running PLA, our class convention adds $x_0 = 1$ to every $\mathbf{x}_n$ vector, forming $\mathbf{x}_n = (1, \mathbf{x}_n^{\text{orig}})$. Suppose that $x_0' = -1$ is added instead to form $\mathbf{x}_n' = (-1, \mathbf{x}_n^{\text{orig}})$. Assume that running PLA on $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ with a particular sequence $n(t), t = 1, 2, \ldots$ returns $\mathbf{w}_{\text{PLA}}$, and running PLA on on $\{(\mathbf{x}_n', y_n)\}_{n=1}^N$ with the same $n(t)$ returns $\mathbf{w}_{\text{PLA}}'$. Prove or disprove that $\mathbf{w}_{\text{PLA}}$ and $\mathbf{w}_{\text{PLA}}'$ are equivalent.

7. (20 points) Dr. Norman thinks PLA will be highly influenced by very long examples as $\mathbf{w}$ changes

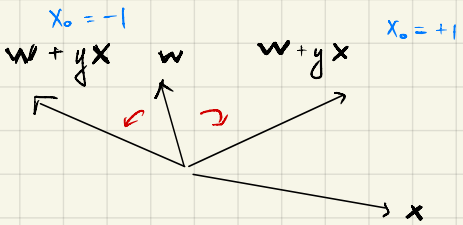
given $w_{t+1} = w_t + y_{n(t)} x_{n(t)}$ where $x_0 = 1$

$$\begin{bmatrix} w_{t+1,0} \\ w_{t+1,1} \\ \vdots \\ w_{t+1,d} \end{bmatrix} = \begin{bmatrix} w_{t,0} \\ w_{t,1} \\ \vdots \\ w_{t,d} \end{bmatrix} + y_{n(t)} \begin{bmatrix} 1 \\ x_{n(t),1} \\ \vdots \\ x_{n(t),d} \end{bmatrix}$$



$$w_{PLA} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}$$

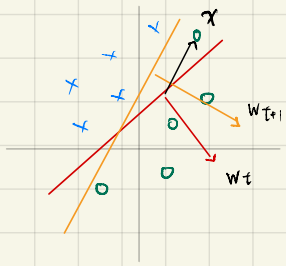
if $x_0 = -1$ $w_{PLA} = \begin{bmatrix} -w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}$, which is different from w_{PLA} only at



w_0 while $w_1, w_2 \cdots w_d$ are the same.

So the lines (or heperplanes in R^d)

$$[w_0 \; w_1 \cdots w_d] \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_d \end{bmatrix} = w_0 x_0 + w_1 x_1 \cdots w_d x_d$$



no matter $x = +1$ or -1 the sign of $w_0 x_0$ would be the same.

$w_1, w_2 \cdots w_d$ decide the normal vector of the hyperplanes and $w_0 x_0$ determine the intercept of the hyperplanes. Although $(w_0, w_1 \cdots w_d)$ turns in different direction, the way heperplane moves is the same. Therefore, two algorithm are the same.

7. (20 points) Dr. Norman thinks PLA will be highly influenced by very long examples, as $\mathbf{w}_t$ changes drastically if $\|\mathbf{x}_{n(t)}\|$ is large. Hence, ze decides to preprocess the training data by normalizing each input vector i.e., $\mathbf{z}_n \leftarrow \frac{\mathbf{x}_n}{\|\mathbf{x}_n\|}$. What is PLA's upper bound on page 16/22 of Lecture 1 change with this preprocessing procedure in terms of $\rho_{\mathbf{z}} = \min_n \frac{y_n \mathbf{w}_f^T \mathbf{z}_n}{\|\mathbf{w}_f\|}$. Prove your answer.

8. (20 points) In PLA, we defined $\rho = \min_n \mathbf{w}^T \mathbf{x}_n$ and ρ is often called the unnormalized **margin** of

$$w_f^T \cdot w_T \geq w_f^T w_t + \min_n y_n w_f^T z_n \times T$$

$$\|w_T\| = \sqrt{\|w_0\|^2 + \max_n \|z_n\|^2 T} = \max_n \|z_n\| \sqrt{T} = \sqrt{T}$$

$$1 \geq \frac{w_f^T \cdot w_T}{\|w_f\| \|w_T\|} \geq \frac{T \times \min_n y_n w_f^T z_n}{\|w_f\| \cdot \sqrt{T}} = \sqrt{T} \cdot C$$

$$C = \frac{\min_n y_n w_f^T z_n}{\|w_f\|}$$

$$T_{max} = \frac{1}{C} = \left(\frac{\|w_f\|}{\min_n y_n w_f^T z_n}\right)^2 = \frac{1}{\rho_z^2}$$

#

8. (20 points) In PLA, we defined $\rho = \min_n y_n \mathbf{w}_f^T \mathbf{x}_n$, and ρ is often called the unnormalized **margin** of $\mathbf{w}_f$. Margin is related to the minimal distance between $\mathbf{x}_n$ and hyperplane $\mathbf{w}_f$, and will be a main concept when we introduce the Support Vector Machine (SVM) later in this class. Before that, let us play with a variant of PLA, the Perceptron Algorithm using Margins (PAM). The difference between PAM and the original PLA is that PAM updates on $\mathbf{x}_n$ if

$$y_n \mathbf{w}_t^T \mathbf{x}_n \leq \tau,$$

where $\tau \geq 0$ is the margin we would like to achieve.

For any given τ, assume that the data set $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ is linearly τ-separable. That is, $\mathbf{w}_f$ satisfies $\rho > \tau$. Prove that PAM halts in a finite number of steps (*Hint: PLA is just a special case with* $\tau = 0$).

$y_n w_n^T x_n \leq \tau$

$w_f^T \cdot w_T > w_f^T w_t + \min_n y_n w_f^T x_n \times T$ (0, $\rho > \tau$)

$\|w_T\|^2 < (\max_n \|x_n\|^2 + 2 y_n w_t^T x_n(t)) T$ (2τ)

$\|w_T\| < \sqrt{(\max_n \|x_n\|^2 + 2\tau) T}$

$\frac{w_f^T}{\|w_f\|} \cdot \frac{w_T}{\|w_T\|} \geq \frac{T \times \min_n y_n w_f^T x_n}{\|w_f\| \cdot \sqrt{\max_n \|x_n\|^2 + 2\tau} \times \sqrt{T}} = \sqrt{T} \cdot C$

$C = \frac{\min_n y_n w_f^T x_n}{\|w_f\| \cdot \sqrt{\max_n \|x_n\|^2 + 2\tau}}$

$T_{max} = \frac{1}{C^2} = \left(\frac{\|w_f\| \cdot \sqrt{\max_n \|x_n\|^2 + 2\tau}}{\min_n y_n w_f^T x_n} \right)^2$

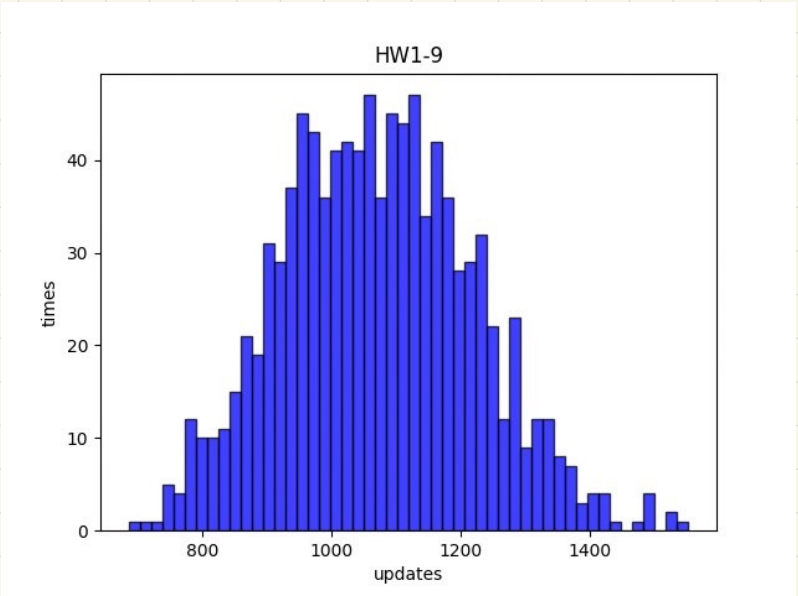
$= \frac{\|w_f\|^2 \cdot (\rho^2 + 2\tau)}{\tau^2}$

$\Rightarrow$ PAM halt in a finite number

9. (20 points, *) Please first follow page 4/22 of Lecture 2, and add $x_0 = 1$ to every $\mathbf{x}_n$. Implement a version of PLA that randomly picks an example $(\mathbf{x}_n, y_n)$ in every iteration, and updates $\mathbf{w}_t$ if and only if $\mathbf{w}_t$ is incorrect on the example. Note that the random picking can be simply implemented *with replacement*—that is, the same example can be picked multiple times, even consecutively. Stop updating and return $\mathbf{w}_t$ as $\mathbf{w}_{\text{PLA}}$ if $\mathbf{w}_t$ is correct consecutively after checking $5N$ randomly-picked examples.

Hint: You can simply follow the algorithm above to solve this problem. But if you are interested in knowing why the algorithm above is somewhat equivalent to the PLA algorithm that you learned in class, here is some more information. (1) The update procedure described above is equivalent to the procedure of gathering all the incorrect examples first and then randomly picking an example among the incorrect ones. But the description above is usually much easier to implement. (2) The stopping criterion above is a randomized, more efficient implementation of checking whether $\mathbf{w}_t$ makes no mistakes on the data set. Passing $5N$ times of correctness checking means that $\mathbf{w}_t$ is mistake-free with more than 99% of probability.

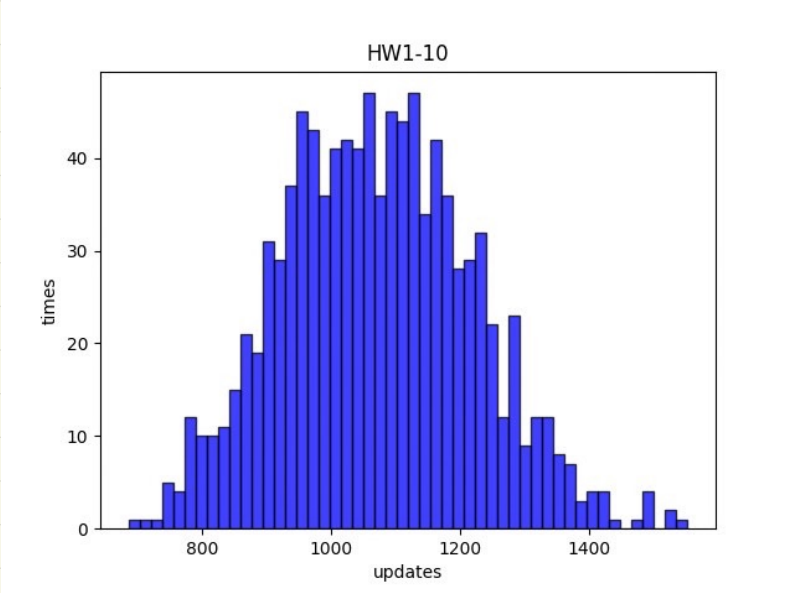
Repeat your experiment for 1000 times, each with a different random seed. Plot a histogram to visualize the distribution of the number of updates needed before returning $\mathbf{w}_{\text{PLA}}$. What is the median number of updates?




```
[1132, 1049, 1238,
1074.5
```

medium

10. (20 points, *) Scale up each $\mathbf{x}_n$ by 11.26, including scaling each x_0 from 1 to 11.26. Then, run PLA on the scaled examples for 1000 experiments, each with a different random seed. Plot a histogram to visualize the distribution of the number of updates needed before returning $\mathbf{w}_{\text{PLA}}$. What is the median number of updates? Compare your result to that of Problem 9. Describe your findings.



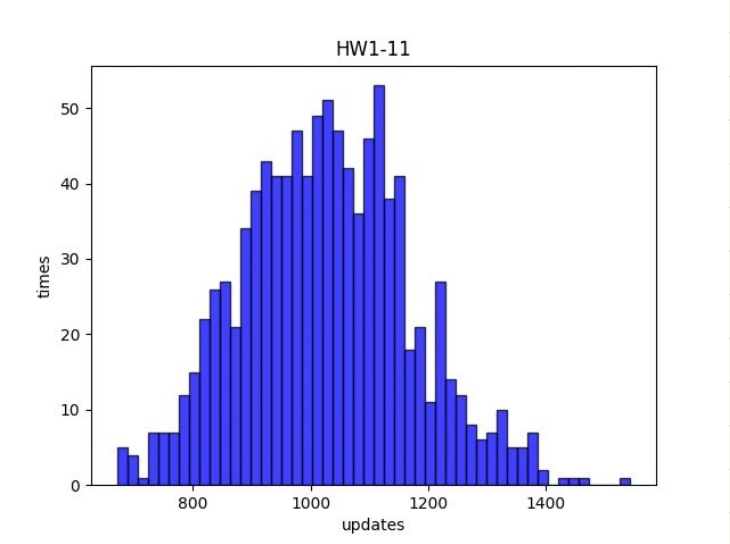

```
[1132, 1049, 1238,
1074.5
```

medium

The distribution and the medium are same to Problem 9

The reason is that scale up every x_n is just like scale up the whole dimesion the intersection and normal vector would not change

11. (20 points, *) Set $x_0 = 11.26$ to every $\mathbf{x}_n$ instead of $x_0 = 1$, and do not do any scaling. Repeat the 1000 experiments above. Plot a histogram to visualize the distribution of the number of updates needed before returning $\mathbf{w}_{\text{PLA}}$. What is the median number of updates? Compare your result to that of Problem 9. Describe your findings.

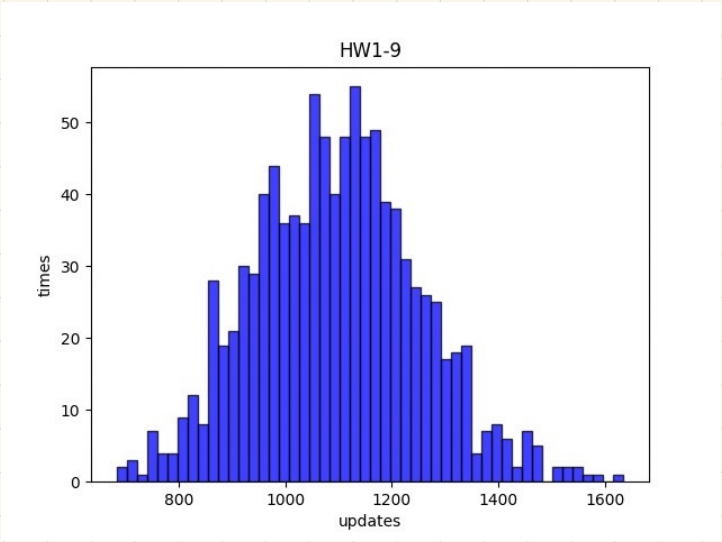



```
[873, 1043, 887,
1032.0
```

medium

The distribution becomes wider than Problem 9 and the medium is smaller than Problem 9

12. (20 points, *) Set $x_0 = 1$ to every $\mathbf{x}_n$, and do not do any scaling. Modify your PLA above to a variant that keeps correcting the same example until it is perfectly classified. That is, when selecting an incorrect example $(\mathbf{x}_{n(t)}, y_{n(t)})$ for updating, the algorithm keeps using that example (that is, $n(t+1) = n(t)$) to update until the weight vector perfectly classifies the example (and each update counts!). Repeat the 1000 experiments above. Plot a histogram to visualize the distribution of the number of updates needed before returning $\mathbf{w}_{\text{PLA}}$. What is the median number of updates? Compare your result to that of Problem 9. Describe your findings.




```
[1069, 977, 1266,
1107.0
```

medium

The distribution is similar to Problem 9 but the medium is larger than Problem 9

13. (Bonus 20 points) In Problem 7, you showed that the upper bound of PLA changes when running on the normalized data instead of the original data. Does it mean that normalization can speed up PLA? Why or why not? Note that this is a bonus problem and TAs can set a high standard on your logical arguments when deciding whether to give you the points.

$$\rho = \min_n y_n(x_n \cdot w)$$

$$\rho_2 = \min_n y_n\left(\frac{x_n}{\|x_n\|} w\right)$$

$$= \min_n y_n(x_n \cdot w) \frac{1}{\|x_t\|}$$

$$\geq \min_n y_n(x_n \cdot w) \left(\min_n \frac{1}{\|x_n\|}\right)$$

$$\geq \frac{\rho}{R}$$

According to Problem 7. $T_{n\,max} = \frac{1}{\rho_2^2}$, the T_{max} of unnormalize is $\frac{R^2}{\rho^2}$, we can get $T_{n\,max} \leq T_{max}$. the upper bound is always at least as good as regular PLA. It does lower the upper bound in some cases. However. it is not necessarily better than regular PLA in every experiment. It depends on the distribution of data. Usually the normalized PLA would be faster if the $\max_n \|x_n\|$ is large and $\min_n y_n(x_n w)$ is small.