工海三 B10505005 蔣欣倪

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

The most famous application of self-supervised learning must be Bert, which can be used on masked token prediction and next sentence prediction.

To explain it briefly, we mask some words in sentences / change the sequence of sentences, Bert is able to fill the correct words into the blanks / whether the sentences are matched.

What can we do on these abilities? Bert can be applied to dozens of downstream tasks, even though they seem to have no relationship with those abilities: Bert can be used as pre-trained model.

With pre-trained model that has the ability of SSL, which learns from data without relying on explicitly annotated labels, many tasks can be done by AI with a greater accuracy, including Natural Language processing.

For example, Sentiment Analysis in text classification .We input sentences, while AI distinguishes the potentially feelings of those sentences, between positive or negative.

It can be used in companies, when customers ask for support about their products, AI need to decide the way it respond to the question or send warning to human making sure the customers problem would be solved in some better way.

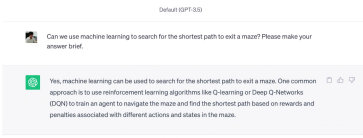
Moreover, use those data in statistics and performance evaluation .

Natural Language Processing (NLP) applications can also benefit companies across various industries by automating tasks, improving customer interactions, gaining insights from text data, and enhancing decision-making processes.

To sum up, self-supervised learning is similar to stem cell in our body, it provides pre-trained model with cost-efficiency, better performance in realistic problems, and what's more important, transfer learning.

AI learns from unlabeled data , after we feed it with some labeled data, it becomes specific AI model which can be used in specific ways.

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.



Yes, I agree with the answer.

Reinforcement learning is used on partial/implicit information.

We can't tell AI about which path is the best (yn=1 or yn=-1), but we can give reward if it exit the maze by spending a shorter time or by less turns.

By reward, AI can learn from the process and keep trying to reach a better solution, depending on our reflection.

However, when to use Q-learning or DQN should depend on situation.

If the size of the maze is small, there won't be problem when using Q-learning/DQN.
If the size is extremely large, the use of DQN will be necessary.

Although both of the two methods learn from experiments that actions interact with the environment, update and keep finding better actions in states.

Q-learning stores the results of experiment by tables, recording state, action and rewards. By going through this table, it's able to decide the action it should take in specific state.

But this makes sense only when the combinations are finite.

When the maze getting larger, the combination of states and actions grow rapidly.
It won't be possible to save records by table.

While DQN, which uses a neural network to approximate the Q-values, is allowed to solve high-dimension problems.

Experience replay, target network, and sample efficiency of DQN are able to help us with more complex maze problems

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.

"Off-the-shelf" refers to pre-trained models or algorithms that are ready to use and don't need additional training, which can save time and resources.

However, whether ml can improve off-the -shelf algorithm depends on problem complexity, algorithm adaptability...

For most of off-the-shelf algorithm, ml is unable to speed up.

In this case, the article focuses on two kind of small sort algorithm: fixed sort and variable sort.

Fixed sort is about how to sort in case of the same size of number, while variable sort is different size.

In the experiment, they introduced AI called AlphaDev, it improved sort-3, sort-6, sort-7,except for sort-4, since it can't find a way to improve human code.

I think there are reasons:

First, machine learning comes up with the algorithm by learning from procedure. We designed a game for AlphaDev, in order to count the step it took. The research isn't completed by directly asking it the better solution.

Second, ChatGPT can't update the information after the year of 2021. Most of the news nowadays are out of the range. The research was done recently, which ChatGPT is unable to reach.

Third, the definition of off-the -shelf is about the algorithm that are used in many of human's research, commonly be seen as basic algorithm, and that's might be the reason of why AI thinks ML I can't improve it, some of which are too basic to be learn or too wonderful to be improve, for example, sort-4.

Last, just like the answer from ChatGPT, it depends on the nature of the problem and the algorithm, some of off-the-shelf algorithms are unable to speed up, while some of which are improvable, just like fixed sort and variable sort.

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.

$W_{t+1} = W_t + y_{n(t)} x_{n(t)}$

$$\begin{cases} W_0(t) + y_{n(t)} \times 1 = W_0(t+1) & (y_{n(t)} = 1) \\ W_0(t) + y_{n(t)} \times 1 = W_0(t+1) & (y_{n(t)} = -1) \end{cases}$$

since PLA updates only when mistakes occur,

$$\begin{cases} W_0(t) + 1 \rightarrow T^+ \text{ mistakes} \xrightarrow{\text{updates } T^+ \text{ times}} W_0(t) + T^+ \\ W_0(t) - 1 \rightarrow T^- \text{ mistakes} \xrightarrow[\text{updates } T^- \text{ times}]{} W_0(t) - T^- \end{cases}$$

By correcting all the mistakes $\rightarrow$ $W_0(t) = T^+ - T^-$ #

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.

$\max \| X_n \|^2 = R^2 = m+1$ (+1的項數 + (-1)的項數 + x_0項共 $m+1$)

$\begin{bmatrix} x_0 \\ \vdots \\ \vdots \\ x_n \end{bmatrix}$ $\begin{matrix} +1 \\ -1 \\ +1 \\ +1 \end{matrix}$

$x_1 \cdots x_n = \pm 1$

$\| W_f \| = \sqrt{(0.5)^2 + \Sigma (1)^2 + \Sigma (-1)^2} = \sqrt{\frac{1}{4} + d}$

$\min y_n X_n W_f^T = \frac{1}{2}$

since W_f^T is the correct matrix $\rightarrow$ $y_n \cdot W_f^T x_n \geq 0$ for $n \neq 0$

$\min y_n \cdot W_f^T x_n = 0$ for $n \neq 0$

$W_0 \cdot x_0 = +0.5$

$\rho^2 = \frac{\min y_n X_n W_f^T}{\| W_f \|} = \frac{\frac{1}{4}}{d+1}$. $R^2 = m+1$

$T \leq \frac{R^2}{\rho^2} = \frac{m+1}{\frac{1}{4d+1}} = (4d+1)(m+1)$ #

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 $n(t), t = 1, 2, \ldots$ with $\mathbf{w}_0 = \mathbf{0}$ returns $\mathbf{w}_{\text{PLA}}$, and running PLA on on $\{(\mathbf{x}'_n, y_n)\}_{n=1}^N$ with the same $n(t)$ with $\mathbf{w}_0 = \mathbf{0}$ returns $\mathbf{w}'_{\text{PLA}}$. Prove or disprove that $\mathbf{w}_{\text{PLA}}$ and $\mathbf{w}'_{\text{PLA}}$ are equivalent.

Prove.

$W_{t+1} = W_t + x_n y_n$

$x_0 = 1$, $x_0' = -1$

Since following the same sequence. $n(t)$. $t = 1, 2, 3 \ldots$

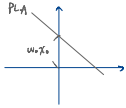
if catch a first data x_1 with $y_{n1} = +1$

→ $w_{0(1)} = w_0 + 1 = 1$, $w_{0(1)}' = w_0 - 1 = -1$

PLA = $w \cdot x_n = w' \cdot x_n'$, $(x_0 = 1,\ x_0' = -1)$

$$\begin{bmatrix} w_{t+1,0} \\ w_{t+1,1} \\ \vdots \\ w_{t+1,d} \end{bmatrix} = \begin{bmatrix} w_{t,0} \\ \vdots \\ w_{t,d} \end{bmatrix} + y_{n(t)} \begin{bmatrix} x_0 \\ x_{n(t),1} \\ \vdots \\ x_{n(t),d} \end{bmatrix}, \quad x_0 = 1$$

$$\begin{bmatrix} w_{t+1,0} \\ \vdots \\ w_{t+1,d} \end{bmatrix} = \begin{bmatrix} w_{t,0} \\ \vdots \\ w_{t,d} \end{bmatrix} + y_{n(t)} \begin{bmatrix} x_0' \\ \vdots \\ x'_{n(t),d} \end{bmatrix}, \quad x_0' = -1$$



∵ $w_0 x_0 = w_0' x_0'$, w's elements are all the same except for w_0 and w_0'

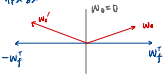
for $n \geq 0$, $w_n = w_n'$

$\begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix}$ indicated the normal vector of the lines / hyperplanes.

$w_0' x_0' / w_0 x_0$ stands for the intercept of PLA , $w_0 x_0 = w_0' x_0'$

Thus, W_{PLA} and W'_{PLA} refer to the same lines / hyperplanes.

Although $W_{PLA(t)} \neq W'_{PLA(t)}$.



$w_0 = -w_0'$

$w_1^T x_n = w_1^T x_n'$ → Both refers to the same ideal PLA.

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., $\overline{\mathbf{x}}_n \leftarrow \frac{\mathbf{x}_n}{\|\mathbf{x}_n\|}$. What is PLA's upper bound on page 16/22 of Lecture 1 change with this preprocessing procedure in terms of $\rho_{\overline{\mathbf{x}}} = \min_n \frac{y_n \mathbf{w}_f^T \overline{\mathbf{x}}_n}{\|\mathbf{w}_f\|}$. Prove your answer.

$\|W_{t+1}\|^2 = \|W_t + y_{n(t)} x_{n(t)}\|^2 \le \|W_t\|^2 + \max \|x_n\|^2$ → $\|W_T\|^2 \le \|W_{t-1}\|^2 + 1 = T$

$W_f^T W_{t+1} \ge W_f^T W_t + \min y_n W_f^T x_n$ → $\frac{W_f^T W_T}{\|W_f\|} \ge \frac{\min y_n W_f^T x_n \cdot T}{\|W_f\|}$

$\sqrt{T} \cdot C \le \frac{W_f^T \cdot W_T}{\|W_f\| \|W_T\|} = \rho \cdot T \cdot \frac{1}{\sqrt{T}} = \rho \cdot \sqrt{T}$, $C = \rho$

$\because \sqrt{T} \cdot C \le 1$, $T \le \frac{1}{C^2} = \frac{1}{\rho^2}$ //

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

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

where $\tau \ge 0$ 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* $\tau = 0$).

$\|W_{t+1}\|^2 \le \|W_t + y_{n(t)} x_{n(t)}\|^2$

$= \|W_t\|^2 + 2 y_{n(t)} W_t^T x_{n(t)} + \|y_{n(t)} x_{n(t)}\|^2$

$\le \|W_t\|^2 + 2\tau + \|y_{n(t)} x_{n(t)}\|^2$

$W_f^T W_{t+1} \ge W_f^T (W_t + y_{n(t)} x_{n(t)})$

$= W_f^T W_t + y_{n(t)} x_{n(t)} W_f^T \ge W_f^T W_t + \min y_{n(t)} x_{n(t)} W_f^T \ge W_f^T W_t + \tau$

$\|W_T\|^2 \le \|W_0\|^2 + 2\tau + R^2$

$\|W_T\|^2 \le \|W_{T-1}\|^2 + 2\tau + R^2$

$W_f^T W_T \ge W_f^T W_0 + \tau$

$W_f^T W_T \ge W_f^T W_{T-1} + \tau$

→ $\|W_T\|^2 \le T(2\tau + R^2)$

$\|W_T\| \le \sqrt{T} \cdot \sqrt{2\tau + R^2}$

→ $W_f^T W_T \ge T\tau$

$1 \ge \frac{W_f^T \cdot W_T}{\|W_f\| \|W_T\|} \ge \sqrt{T} \cdot C$

$\sqrt{T} \cdot \sqrt{2\tau + R^2}$

$1 \ge \frac{\tau}{\sqrt{2\tau + R^2}} \cdot \sqrt{T}$

$\sqrt{T} \le \frac{\sqrt{2\tau + R^2}}{\tau}$

$T \le \frac{2\tau + R^2}{\tau^2}$

Experiments with Perceptron Learning Algorithm

Next, we use an artificial data set to study PLA. The data set with $N = 256$ examples is in

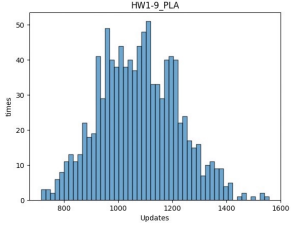
`http://www.csie.ntu.edu.tw/~htlin/course/ml23fall/hw1/hw1_train.dat`

Each line of the data set contains one $(\mathbf{x}_n, y_n)$ with $\mathbf{x}_n \in \mathbb{R}^{12}$. The first 10 numbers of the line contains the components of $\mathbf{x}_n$ orderly, the last number is y_n. Please initialize your algorithm with $\mathbf{w} = \mathbf{0}$ and take sign(0) as -1.

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.

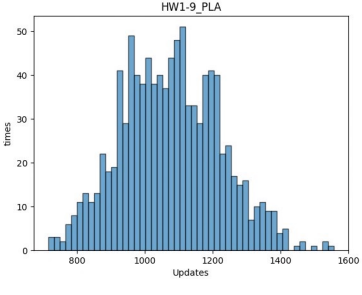
Hint: (1) The update procedure described above is equivalent to the procedure of gathering all the incorrect examples first and then randomly picking an example among the incorrect ones. But the description above is usually much easier to implement. (2) The stopping criterson above is a randomized, more efficient implementation of checking whether $\mathbf{w}_t$ makes no mistakes on the data set.

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?



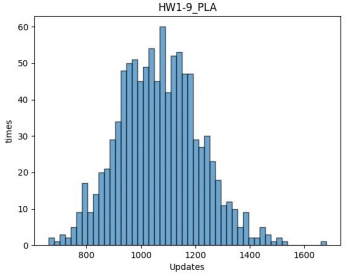
Median: 1072.9

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.



Median: 1072.5
The whole distribution is the same as problem 9.

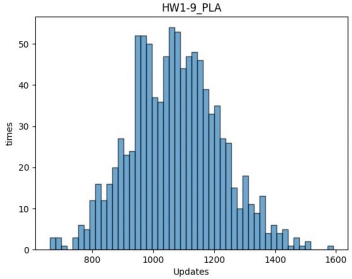
11. (20 points, *) Set $x_0 = 11.26$ to every $\mathbf{x}_n$ instead of $x_0 = 1$, and do not do any scaling. Repeat the 1000 experiments above. Plot a histogram to visualize the distribution of the number of updates needed before returning $\mathbf{w}_{\text{PLA}}$. What is the median number of updates?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



Median: 1065.0

The largest time of update changes from 50->60

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.



Median: 1071.0

The median is slightly smaller than problem 9.

The distribution is more concentrated in the range of 1000 to 2000.

Bonus: Does Normalization Help PLA?

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.

Yes

In problem 7, the result shows that the maximum update time T change from $(R/\rho)^2$ to $1/(\rho^2)$

R refers to max $||x||$, while ρ indicates the minimum distance from data to PLA

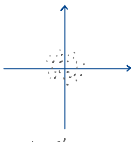
After we normalized the data, R will reduce to 1, however, ρ would also reduce from ρ to ρ′

The maximum update time depends on the radio of R and ρ.

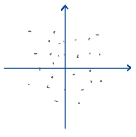
If the data itself were scattered, $\rho >> \rho'$, $T' < T$, normalization did speed up.
If were concentrated, normalization may not improve PLA.(but won't be worse than origin)

We take four situation for example.

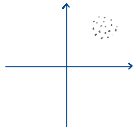
In conclusion, though the maximum update time reduce, it doesn't mean it will definitely improve.
After normalization, update time won't be larger, but whether it will be smaller depends on data.



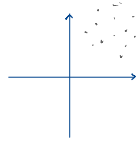
$R \approx R'$
$\rho \sim \rho'$



$R > R'$
$\rho > \rho'$



$R > R'$
$\rho \approx \rho'$



$R > R'$
$\rho > \rho'$