CS2109S Tutorial 8

Neural Networks

(AY 24/25 Semester 2)

April 4, 2025

(Prepared by Benson)

Contents

Admin Info

- For those who have reached Level 30, PS5 (Neural Networks) is *optional*. But I would still encourage you to attempt at least part of it – Pytorch is one of the most important takeaways of this course.
- Bonus submission deadlines (for EXP / bubble tea):
  - Behind Pytorch Autograd: 11 Apr (Week 12 Fri) 4pm
  - Messing with a GPT Model: 25 Apr (Reading Week Fri) 4pm

  Feel free to share me your progress and I might drop some hints!

Q1. Logic Gates

(a) Dry run the Perceptron learning algorithm with $\eta = 0.1$, all weights initialized to 0 and activation function $\text{sign}(z) = 1$ if $z \geq 0$ else 0.

AND: Hypothesis $\hat{y} = \text{sign}(b + w_1 x_1 + w_2 x_2)$.

Iteration	b	w_1	w_2
Initial	0	0	
1	-0.1	0	
2	0	0.1	
11	-0.3	0.2	0.1

$b \leftarrow b + \eta(y - \hat{y})(1)$

$b \leftarrow b + \eta(y - \hat{y})(1)$

$b \leftarrow b + \eta(y - \hat{y})(1)$

$b \leftarrow b + \eta(y - \hat{y})(1)$

$b \leftarrow b + \eta(y - \hat{y})(1)$

$b \leftarrow b + \eta(y - \hat{y})(1)$

$0.1(0 - 1)(1) = -0.1$

$b \leftarrow b + \eta(y - \hat{y})(1)$

$0.1(1 - 0)(1) = 0.1$

$w_1 \leftarrow w_1 + \eta(y - \hat{y})x_1$

$0.1(0 - 1)(0) = 0$

$w_1 \leftarrow w_1 + \eta(y - \hat{y})x_1$

$0.1(1 - 0)(1) = 0.1$

$w_2 \leftarrow w_2 + \eta(y - \hat{y})x_2$

$w_2 \leftarrow w_2 + \eta(y - \hat{y})x_2$

$w_2 \leftarrow w_2 + \eta(y - \hat{y})x_2$

$w_2 \leftarrow w_2 + \eta(y - \hat{y})x_2$

$w_2 \leftarrow w_2 + \eta(y - \hat{y})x_2$

$w_2 \leftarrow w_2 + \eta(y - \hat{y})x_2$

$w_2 \leftarrow w_2 + \eta(y - \hat{y})x_2$

$0.1(0 - 1)(1) = -0.1$

AND			
x_1	x_2	y	$\hat{y}$
0	0	0	[illegible]
0	1	0	[illegible]
1	0	0	[illegible]
1	1	1	[illegible]

Misclassified instance

Misclassified instance

Misclassified instance

Misclassified instance

Q1. Logic Gates

(a) Dry run the Perceptron learning algorithm with $\eta = 0.1$, all weights initialized to 0 and activation function $\text{sign}(z) = 1$ if $z \geq 0$ else 0.

OR: Hypothesis $\hat{y} = \text{sign}(b + w_1 x_1 + w_2 x_2)$.

Iteration	b	w_1	w_2
Initial	0	0	[illegible]
1	-0.1	0	[illegible]
2	0	0	0.1
[illegible]	0.	[illegible]	1
[illegible]	0	[illegible]	1
[illegible]	0	[illegible]	1
[illegible]	0.	[illegible]	[illegible]
[illegible]	0.	[illegible]	[illegible]
8	-0.2	0.2	0.1
9	-0.3	0.1	0.1
10	-0.2	0.2	0.2
11	-0.3	0.2	0.1

$b \leftarrow b + \eta(y - \hat{y})(1)$

$b \leftarrow b + \eta(y - \hat{y})(1)$

$b \leftarrow b + \eta(y - \hat{y})(1)$

$b \leftarrow b + \eta(y - \hat{y})(1)$

$b \leftarrow b + \eta(y - \hat{y})(1)$

$0.1(0 - 1)(1) = -0.1$

$w_1 \leftarrow w_1 + \eta(y - \hat{y})x_1$

$w_1 \leftarrow w_1 + \eta(y - \hat{y})x_1$

$w_1 \leftarrow w_1 + \eta(y - \hat{y})x_1$

$w_1 \leftarrow w_1 + \eta(y - \hat{y})x_1$

$w_1 \leftarrow w_1 + \eta(y - \hat{y})x_1$

$0.1(0 - 1)(0) = 0$

$w_2 \leftarrow w_2 + \eta(y - \hat{y})x_2 \quad + \eta(y - \hat{y})x_2$

$w_2 \leftarrow w_2 + \eta(y - \hat{y})x_2 \quad .)(0) = 0$

$0.1(0 - 1)(0) = 0$

OR			
x_1	x_2	y	$\hat{y}$
0	0	0	101010 sign(0)(−0.1)
0	1	1	101111 sign(0)(0.1)
1	[illegible]	[illegible]	sign(0)(0.1)
1	[illegible]	[illegible]	sign(0)(0.2)

Misclassified instance

Misclassified instance

Misclassified instance

Misclassified instance

Q1. Logic Gates

(a) Dry run the Perceptron learning algorithm with $\eta = 0.1$, all weights initialized to 0 and activation function $\text{sign}(z) = 1$ if $z \geq 0$ else 0.

NOT: Hypothesis $\hat{y} = \text{sign}(b + w_1 x)$.

Iteration	b	w_1
Initial	0	0
1	-0.1	-0.1
2	0	-0.1

$b \leftarrow b + \eta(y - \hat{y})(1)$

$b \leftarrow b + \eta(y - \hat{y})(1)$

$0.1(1 - 0)(1) = 0.1$

$w_1 \leftarrow w_1 + \eta(y - \hat{y})x_1$

$w_1 \leftarrow w_1 + \eta(y - \hat{y})x_1$

$0.1(1 - 0)(0) = 0$

6	-0.2	0.2	0
7	-0.3	0.1	0
8	-0.2	0.2	0.1
9	-0.3	0.1	0.1
10	-0.2	0.2	0.2
11	-0.3	0.2	0.1

NOT		
x	y	$\hat{y}$
0	1	101
1	0	100

sign(0)0.1)

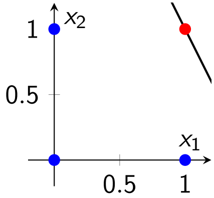
sign(0)0.2)

Misclassified instance

Misclassified instance

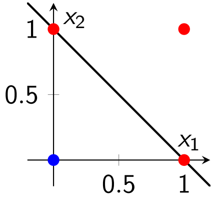
Q1. Logic Gates

AND:



$\hat{y} = \text{sign}(-0.3 + 0.2x_1 + 0.1x_2)$

OR:

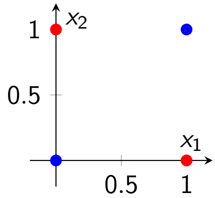


$\hat{y} = \text{sign}(-0.1 + 0.1x_1 + 0.1x_2)$

Q1. Logic Gates

(b) Is it possible to model the XOR function using a single Perceptron? Refer to Figure 2 for the truth table of the XOR gate. Comment on your answer.

Hypothesis $\hat{y} = \text{sign}(b + w_1 x_1 + w_2 x_2)$.



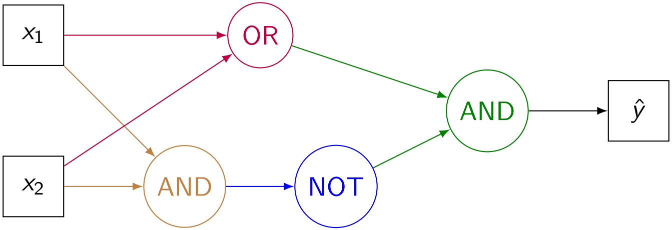
XOR		
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

- The points are not **linearly separable**.
- It is impossible to model the XOR function using a single Perceptron.

Q1. Logic Gates

(c) Model XOR function (takes 2 inputs) using a number of perceptrons that implement AND, OR, and NOT functions. Show the diagram of the final Perceptron network.

$$\mathrm{XOR}(x_1, x_2) = \mathrm{AND}(\mathrm{OR}(x_1, x_2), \mathrm{NOT}(\mathrm{AND}(x_1, x_2)))$$



Q1. Logic Gates

(d) If we change the ordering of data samples in Perceptron Update Rule, will the model converges to a different model weight for the AND operator? What can you conclude from the observation?

Ordering: $(0, 1, 2, 3)$

Iteration	b	w_1	w_2
Initial	0	0	0
1	-0.1	0	0
2	0	0.1	0.1
3	-0.1	0.1	0.1
4	-0.2	0.1	0
5	-0.1	0.2	0.1
6	-0.2	0.2	0
7	-0.3	0.1	0
8	-0.2	0.2	0.1
9	-0.3	0.1	0.1
10	-0.2	0.2	0.2
11	-0.3	0.2	0.1

Ordering: $(0, 2, 3, 1)$

Iteration	b	w_1	w_2
Initial	0	0	0
1	-0.1	0	0
2	0	0.1	0.1
3	-0.1	0.1	0
4	-0.2	0	0
5	-0.1	0.1	0.1
6	-0.2	0.1	0
7	-0.1	0.2	0.1
8	-0.2	0.2	0
9	-0.3	0.1	0
10	-0.2	0.2	0.1
11	-0.3	0.1	0.1
12	-0.2	0.2	0.2
13	-0.3	0.2	0.1

Ordering: $(0, 2, 1, 3)$

Iteration	b	w_1	w_2
Initial	0	0	0
1	-0.1	0	0
2	0	0.1	0.1
3	-0.1	0.1	0.1
4	-0.2	0	0.1
5	-0.1	0.1	0.2
6	-0.2	0	0.2
7	-0.3	0	0.1
8	-0.2	0.1	0.2
9	-0.3	0.1	0.1
10	-0.2	0.2	0.2
11	-0.3	0.1	0.2

Q1. Logic Gates

(d) If we change the ordering of data samples in Perceptron Update Rule, will the model converges to a different model weight for the AND operator? What can you conclude from the observation?

- Reordering data points could help the model converge much faster.
- Manipulating the ordering could direct the model to a different weight. There is no guarantee to converge to the same model even for such a simple gate function.

Q1. Logic Gates

(e) With regards to Figure 1, does your proposed model have high bias and high variance? *(Recap: What is bias? What is variance?)*

- **Low bias**: It classifies all data points correctly.
- **Low variance**: It is a simple linear model, the simplest model to perform the logic gates given.

Q1. Logic Gates

Does the Perceptron learning algorithm *always* converge on linearly separable datasets? Make your guess!

A. Yes, always!
B. Yes, if the learning rate is small enough.
C. Yes, if the parameters are ordered properly.
D. No, it can be stuck in a loop regardless.

Q2. Single vs Multi Layer Perceptron

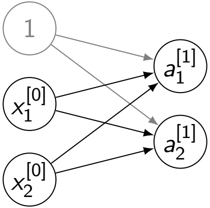
After training both networks, you obtain a mean squared error of 1000 on the training set and a mean squared error of 2000 on the validation set for the single-layer perceptron, and a mean squared error of 800 on the training set and a mean squared error of 1200 on the validation set for the multi-layer perceptron.

(a) What might be the reasons for the difference in performance between the single-layer perceptron and the multi-layer perceptron?
   - More layers $\Rightarrow$ able to learn more complex (non-linear) patterns.
   - The single-layer perceptron is limited to a linear classifier.

(b) How might you modify the single-layer perceptron to improve its performance, and what are the advantages and disadvantages of doing so?
   - Add transformed features (polynomial $(x_1)^2$, interaction $x_1 x_2$).

(c) What techniques could you use to improve the performance of the multilayer perceptron?
   - Underfitting: Increase the number of hidden layers (parameters).
   - Overfitting: Regularization.

Recap: Forward Propagation



$$a_1^{[1]} = 1 \cdot W_{01}^{[1]} + x_1^{[0]} \cdot W_{11}^{[1]} + x_2^{[0]} \cdot W_{21}^{[0]}$$

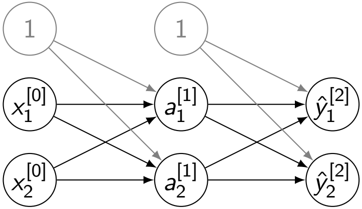
$$a_2^{[1]} = 1 \cdot W_{02}^{[1]} + x_1^{[0]} \cdot W_{12}^{[1]} + x_2^{[0]} \cdot W_{22}^{[0]}$$

$$\boldsymbol{a}^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \end{bmatrix} = \begin{bmatrix} W_{01}^{[1]} & W_{11}^{[1]} & W_{21}^{[1]} \\ W_{02}^{[1]} & W_{12}^{[1]} & W_{22}^{[1]} \end{bmatrix} \times \begin{bmatrix} 1 \\ x_1^{[0]} \\ x_2^{[0]} \end{bmatrix} = \boldsymbol{W}^\top \times \boldsymbol{x}^{[0]}$$

Q3. Forward Propagation

Suppose there is a data input $\boldsymbol{x} = (2,3)^\top$ and the actual output label is $\boldsymbol{y} = (0.1, 0.9)^\top$. The weights for the network are

$$\boldsymbol{W}^{[1]} = \begin{bmatrix} 0.1 & 0.1 \\ -0.1 & 0.2 \\ 0.3 & -0.4 \end{bmatrix}, \boldsymbol{W}^{[2]} = \begin{bmatrix} 0.1 & 0.1 \\ 0.5 & -0.6 \\ 0.7 & -0.8 \end{bmatrix}$$



Calculate $\boldsymbol{a}^{[1]}$, $\hat{\boldsymbol{y}}^{[2]}$ and $L(\hat{\boldsymbol{y}}^{[2]}, \boldsymbol{y})$.

Activation Func: $\mathrm{ReLU}(x) = \max(0, x)$.

$$\boldsymbol{a}^{[1]} = \mathrm{ReLU}((\boldsymbol{W}^{[1]})^\top \boldsymbol{X}) = \mathrm{ReLU}\left(\begin{bmatrix} 0.1 & -0.1 & 0.3 \\ 0.1 & 0.2 & -0.4 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}\right) = \mathrm{ReLU}\left(\begin{bmatrix} 0.8 \\ -0.7 \end{bmatrix}\right) = \begin{bmatrix} 0.8 \\ 0 \end{bmatrix}$$

$$\hat{\boldsymbol{y}}^{[2]} = \mathrm{ReLU}((\boldsymbol{W}^{[2]})^\top \boldsymbol{a}^{[1]}) = \mathrm{ReLU}\left(\begin{bmatrix} 0.1 & 0.5 & 0.7 \\ 0.1 & -0.6 & -0.8 \end{bmatrix}\begin{bmatrix} 1 \\ 0.8 \\ 0 \end{bmatrix}\right) = \mathrm{ReLU}\left(\begin{bmatrix} 0.5 \\ -0.38 \end{bmatrix}\right) = \begin{bmatrix} 0.5 \\ 0 \end{bmatrix}$$

$$L(\hat{\boldsymbol{y}}^{[2]}, \boldsymbol{y}) = \frac{1}{2}\left((\hat{y}_1^{[2]} - y_1)^2 + (\hat{y}_2^{[2]} - y_2)^2\right) = \frac{1}{2}\left((0.5-0.1)^2 + (0-0.9)^2\right) = 0.485$$

Q4. Let's Activate!

We can define a neural network as follows:

$$\hat{y} = g((\boldsymbol{W}^{[L]})^\top \ldots g(\boldsymbol{W}^{[2]})^\top \cdot g(\boldsymbol{W}^{[1]})^\top \boldsymbol{x})$$

where $\boldsymbol{W}^{[l]\in\{1,\cdots,L\}}$ is a weight matrix. You're given the following weight matrices:

$$\boldsymbol{W}^{[3]} = \begin{bmatrix} 1.2 & -2.2 \\ 1.2 & 1.3 \end{bmatrix}, \boldsymbol{W}^{[2]} = \begin{bmatrix} 2.1 & -0.5 \\ 0.7 & 1.9 \end{bmatrix}, \boldsymbol{W}^{[1]} = \begin{bmatrix} 1.4 & 0.6 \\ 0.8 & 0.6 \end{bmatrix}$$

You are given $g(z) = \mathrm{SiLU}(z) = \dfrac{z}{1+e^{-z}}$ between all layers except the last layer.
Is it possible to replace the whole neural network with just one matrix in both cases **with** and **without** non-linear activations $g(z)$?

Q4. Let's Activate!

Without non-linear activations:

$$
\begin{aligned}
\hat{y} &= (\boldsymbol{W}^{[3]})^\top (\boldsymbol{W}^{[2]})^\top (\boldsymbol{W}^{[1]})^\top \boldsymbol{x} \\
&= \begin{bmatrix} 1.2 & 1.2 \\ -2.2 & 1.3 \end{bmatrix} \begin{bmatrix} 2.1 & 0.7 \\ -0.5 & 1.9 \end{bmatrix} \begin{bmatrix} 1.4 & 0.8 \\ 0.6 & 0.6 \end{bmatrix} \boldsymbol{x} \\
&= \left(\begin{bmatrix} 1.2 & 1.2 \\ -2.2 & 1.3 \end{bmatrix} \begin{bmatrix} 2.1 & 0.7 \\ -0.5 & 1.9 \end{bmatrix} \begin{bmatrix} 1.4 & 0.8 \\ 0.6 & 0.6 \end{bmatrix} \right) \boldsymbol{x} \\
&= \begin{bmatrix} 4.56 & 3.408 \\ -6.82 & -3.658 \end{bmatrix} \boldsymbol{x} \\
&= \boldsymbol{M}^\top \boldsymbol{x}
\end{aligned}
$$

$\therefore$ It is possible to replace the whole neural network with $M = \begin{bmatrix} 4.56 & -6.82 \\ 3.408 & -3.658 \end{bmatrix}$.

Q4. Let's Activate!

With non-linear activations: Suppose $\hat{y} = \boldsymbol{M}^\top \boldsymbol{x}$. We expect $\hat{y}$ to be doubled if we double $\boldsymbol{x}$.

- When $\boldsymbol{x} = \begin{bmatrix} 1 & 0 \end{bmatrix}^\top$:

$$\hat{y} = g\left((\boldsymbol{W}^{[3]})^\top g\left((\boldsymbol{W}^{[2]})^\top g\left((\boldsymbol{W}^{[1]})^\top \boldsymbol{x}\right)\right)\right) = \begin{bmatrix} 3.0571 \\ -5.2727 \end{bmatrix}$$

Intuition: Prove that it is impossible since the network is no longer "linear".

- When $\boldsymbol{x} = \begin{bmatrix} 2 & 0 \end{bmatrix}^\top$:

$$\hat{y} = g\left((\boldsymbol{W}^{[3]})^\top g\left((\boldsymbol{W}^{[2]})^\top g\left((\boldsymbol{W}^{[1]})^\top \boldsymbol{x}\right)\right)\right) = \begin{bmatrix} 7.7257 \\ -13.2458 \end{bmatrix} \neq 2 \cdot \begin{bmatrix} 3.0571 \\ -5.2727 \end{bmatrix}$$

The outputs are not linear $\Rightarrow$ a contradiction! $\therefore$ No such $\boldsymbol{M}$ exists.

Q4. Let's Activate!

Conclusion:

- Without non-linear activations, the entire network **collapses to a simple linear model**.

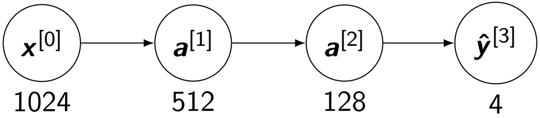
$$\begin{aligned}\hat{y} &= (\boldsymbol{W}^{[L]})^\top \dots (\boldsymbol{W}^{[2]})^\top (\boldsymbol{W}^{[1]})^\top \boldsymbol{x} \\ &= \left((\boldsymbol{W}^{[L]})^\top \dots (\boldsymbol{W}^{[2]})^\top (\boldsymbol{W}^{[1]})^\top\right) \boldsymbol{x}\end{aligned}$$

- Non-linear activation functions let the network model non-linear relationships in the data. Increasing the depth of the network will help the model learn more complex relationships.

Q5. Working with Dimensions

You're building a self-driving car program that takes in grayscale images of size 32×32 where 32 is the image height and width. There are 4 classes your simplified program has to classify: {car, person, traffic light, stop sign}. You start off experimenting with a Multi-layer Perceptron composed of three linear layers of the form $\boldsymbol{y} = \boldsymbol{W}^\top \boldsymbol{x}$, where $\boldsymbol{x} \in R^d$ is the input vector, $\boldsymbol{W}$ is the weight matrix, and $\boldsymbol{y}$ is the network output.

Layer	Input dim	Weight Matrix dim	Output dim
Linear layer 1	$\underline{\ 1024\ } \times 1$	$\underline{\ 1024\ } \times \underline{\ 512\ }$	512×1
Linear layer 2	512×1	$\underline{\ 512\ } \times 128$	$\underline{\ 128\ } \times 1$
Linear layer 3	128×1	$\underline{\ 128\ } \times 4$	$\underline{\ 4\ } \times 1$



Bonus. Perceptron Learning Algorithm

Extra Slide

Prove that the Perceptron Learning Algorithm always converges on a **linearly separable dataset**! You may assume the initial weights are 0 and $y^{(i)} \in \{-1, 1\}$.

(Hint: Let $\boldsymbol{w}^*$ *be a* **unit** *weight vector that linearly separates the data, we have* $y^{(i)}(\boldsymbol{w}^* \cdot \boldsymbol{x}^{(i)}) \geq \gamma$ *since the margin is positive. Also, denote* $R = \max_i \|\boldsymbol{x}^{(i)}\|$*. Prove that after* k *iterations, (1)* $\boldsymbol{w} \cdot \boldsymbol{w}^* \geq k\eta\gamma$ *and (2)* $|\boldsymbol{w}\|^2 \leq k\eta^2 R^2$*. Finally, show that* $k \leq \dfrac{R^2}{\gamma^2}$*, i.e. the algorithm never takes more than this number of iterations.)*

Bonus. Perceptron Learning Algorithm

Solution:

- Consider the k-th iteration with initial weights $\boldsymbol{w_0}$ and misclassified instance $(\boldsymbol{x}^{(i)}, y^{(i)})$. The update rule gives $\boldsymbol{w} = \boldsymbol{w_0} + \eta y^{(i)} \boldsymbol{x}^{(i)}$. Then

$$\begin{aligned} \boldsymbol{w} \cdot \boldsymbol{w}^* &= \left(\boldsymbol{w_0} + \eta y^{(i)} \boldsymbol{x}^{(i)}\right) \cdot \boldsymbol{w}^* \\ &= \boldsymbol{w_0} \cdot \boldsymbol{w}^* + \eta y^{(i)} (\boldsymbol{x}^{(i)} \cdot \boldsymbol{w}^*) \\ &\geq \boldsymbol{w_0} \cdot \boldsymbol{w}^* + \eta\gamma \end{aligned}$$

  This implies $\boldsymbol{w} \cdot \boldsymbol{w}^*$ increases by at least $\eta\gamma$ in each iteration. After k iterations, $\boldsymbol{w} \cdot \boldsymbol{w}^* \geq k\eta\gamma$.

- Since $\boldsymbol{w}^*$ is a unit vector, we have $\boldsymbol{w} \cdot \boldsymbol{w}^* = \|\boldsymbol{w}\| \|\boldsymbol{w}^*\| \cos\theta = \|\boldsymbol{w}\| \cos\theta \leq \|\boldsymbol{w}\|$. Hence $\|\boldsymbol{w}\|^2 \geq (\boldsymbol{w} \cdot \boldsymbol{w}^*)^2 \geq k^2\eta^2\gamma^2$.

Bonus. Perceptron Learning Algorithm

Extra Slide

Solution:

- Since point $(x^{(i)}, y^{(i)})$ is misclassified, we also have $y^{(i)}(\boldsymbol{w_0} \cdot \boldsymbol{x}^{(i)}) \leq 0$. Then

$$\begin{aligned}
\|\boldsymbol{w}\|^2 &= \|\boldsymbol{w_0} + \eta y^{(i)}\boldsymbol{x}^{(i)}\|^2 \\
&= \|\boldsymbol{w_0}\|^2 + \eta^2(y^{(i)})^2\|\boldsymbol{x}^{(i)}\|^2 + 2\eta y^{(i)}(\boldsymbol{w} \cdot x^{(i)}) \\
&= \|\boldsymbol{w_0}\|^2 + \eta^2\|\boldsymbol{x}^{(i)}\|^2 + 2\eta y^{(i)}(\boldsymbol{w} \cdot x^{(i)}) \\
&\leq \|\boldsymbol{w_0}\|^2 + \eta^2\|\boldsymbol{x}^{(i)}\|^2 \\
&\leq \|\boldsymbol{w_0}\|^2 + \eta^2 R^2
\end{aligned}$$

  This implies $\|\boldsymbol{w}\|^2$ increases by at most $\eta^2 R^2$ in each iteration. After k iterations, $\|\boldsymbol{w}\|^2 \leq k\eta^2 R^2$.

- Combining both inequalities, we have $k^2\eta^2\gamma^2 \leq k\eta^2 R^2 \Rightarrow k \leq \dfrac{R^2}{\gamma^2}$.