CS-473
Pattern Recognition

Tutorial for Assignment 4

T.As.: Myrto Villia , Despina - Ekaterini Argiropoulos, Michalis Savorianakis

Tutor: Panos Trahanias

April, 2025

Outline of the presentation

- **Exercise 1**: Single-sample Perceptron

- **Exercise 2**: Batch Perceptron

Dataset

You are given the dataset :

```
       feature_0  feature_1  labels
id
0       7.647359  -1.924193       0
1       7.299635   6.567092       1
2       6.871232   1.099747       0
3       6.667108   8.972409       1
4       7.388563   7.507231       1
...          ...        ...     ...
1995    5.857021  -0.015254       0
1996    7.549255   6.050091       1
1997    8.206501   0.439776       0
1998    7.276851   8.017470       1
1999    7.018212  -0.109663       0

[2000 rows x 3 columns]
[[ 7.64735858 -1.92419261  0.        ]
 [ 7.2996353   6.56709162  1.        ]
 [ 6.87123234  1.09974705  0.        ]
 ...
 [ 8.20650089  0.43977649  0.        ]
 [ 7.27685125  8.01747048  1.        ]
 [ 7.01821195 -0.10966278  0.        ]]
```

1. Create a scatter plot of the dataset. Is the dataset linearly separable? Justify your answer in 1-2 sentences.

You should separate them according to labels!

Exercise 1: Single-sample Perceptron

2. Create a function called 'train_single_sample' that implements the Fixed-Increment Single-Sample Perceptron algorithm. The arguments of the function should be:

   (a) a := weights + bias (bias trick)

   (b) y := data + '1's (bias trick)

   (c) labels

   (d) n_iterations := the number of iterations or updates

   (e) lr := learning rate

   (f) variable_lr := boolean

   and the function should return:

   (a) a := trained model

   (b) acc_history := list of accuracy values at every iteration

   Call the function 'train_single_sample' to train a linear model and print the trained model. Also, print the model's accuracy every *n_samples* iterations. Use the following hyperparameters:

   (a) n_iterations = 30001

   (b) lr = 1

   (c) variable_lr = False (Set it False for now, you will set it True in Question 5).

What are the dimensions of a and y?

Exercise 1: Single-sample Perceptron

Algorithm 4. Fixed-Increment Single-Sample Perceptron

```
1 begin initialize a, k=0
2   do  k ← (k+1) mod n
3     If y^k is misclassified by a, then a ← a + y^k
4   until all patterns properly classified
5   return a
6 end
```

For each iteration :

- **Step 1:** Pick a single random sample
- **Step 2:** Compute the score of the sample
- **Step 3:** Compute the prediction of the sample
- **Step 4:** Update a.

  There are 4 cases (for our exercise), in two of them you have to update a, in the other two you do not need to update. What are these 4 cases?

Exercise 1: Single-sample Perceptron

We are starting with a random decision boundary, a. The vector a decides how I split space.

<u>Step 2:</u> Compute the score of the sample
score=a*current_random_sample. What is the dimension of score?

<u>Step 3:</u> Compute the prediction of the sample
Define a convention: score >= 0 → predicted class 0, score < 0 → predicted class 1.

Then we train the weights so that this decision rule gets better at matching the real labels in the data.

If we flipped the rule to score >= 0 → class 1, you'd be training the weights to learn *another(?)* decision boundary.

Exercise 1: Single-sample Perceptron

Step 4: Update a.

If a sample has a wrong prediction 0 **(i.e., score >= 0)**, what is the update rule and why?

- The model **thinks it's class 0**. But if that prediction is wrong, the **true label is 1**.

- So we **need to make the score more negative**, to push it into the correct side of the decision boundary.

If a sample has a wrong prediction 1 **(i.e., score < 0)**, what is the update rule and why?

- The model thinks it's class 1. But if it's wrong, the true label is 0.

- We need to **increase the score**, to push it to the correct side.

Exercise 1: Single-sample Perceptron

Bonus 7% added to the total assignment grade: To implement the above function, you used a convention: **score** $>= 0 \rightarrow$ **predicted class a** and **score** $< 0 \rightarrow$ **predicted class b**. There are four possible (prediction, true label) combinations. List all of them. Identify which of these combinations require an update. For the cases where an update is needed, describe the update rule and explain why it is applied. Explain your answer. Could any label values be used instead of a and b?

3. Plot the history of the accuracy during training.

You need to compute an accuracy value in each iteration. Be careful how to compute accuracy.
Use all the samples.

Exercise 1: Single-sample Perceptron

4. Create a function called 'plot_model' that takes as input the trained weights (+ bias), the data, and the labels and returns a scatter plot with the decision boundaries of the model (**Hint**: you can use plt.contour()). Call the function you implemented to plot the data along with the trained model.

The model learns a weight vector a=[a1,a2,b].

For a 2D input point x=[x1,x2], the classifier computes a score.

The **decision boundary** is the set of all points for which score=0 — this forms a line in 2D space.

Generate a grid of 2D points over the data space and compute the score z for each one using the model.

If we had 1D data (a single feature), the decision boundary would be a **point** on the real line. In 2D, that boundary becomes a line; in 3D, it would be a plane; in higher dimensions, it becomes a hyperplane.

Exercise 1: Single-sample Perceptron

5. Now we are going to retrain our model but with a variable learning rate. Create a 'Scheduler' class that implements a function 'get_next_lr' that every time it is called returns the next learning rate. The object should work according to the Equation at the beginning of the assignment. Now configure the training function 'train_single_sample' to use this object when 'variable_lr = True'. Retrain the model with a variable learning rate. Plot the dataset and the trained model as before using the function 'plot_model' . What is the main difference compared to training with a fixed learning rate? What method do you think is better? Justify your answer.

Exercise 2: Batch Perceptron

1. Create a function called 'train_batch' that implements the Batch Perceptron algorithm. The arguments of the function should be:

   (a) a := weights + bias (bias trick)

   (b) y := data + '1's (bias trick)

   (c) labels

   (d) theta := the value for the theta criterion

   (e) batch_size

   (f) lr := learning rate

   (g) variable_lr := boolean

   and the function should return:

   (a) a := trained model

   (b) acc_history := list of accuracy values

   (c) error_history := list of error values

   The function should print the model's accuracy and the error at every iteration. The error here is the sum of the absolute values of the updates.

Exercise 2: Batch Perceptron

Algorithm 3. Batch Perceptron

1 **begin** **initialize** a, criterion θ, $\eta(0)>0$, k=0
2 **do** k ← k+1
3 $\mathbf{a} \leftarrow \mathbf{a} + \eta(k)\sum_{y \in \mathcal{Y}_k} \mathbf{y}$
4 **until** $\left|\eta(k)\sum_{y \in \mathcal{Y}_k} \mathbf{y}\right| < \theta$
5 **return** a
6 **end**

= **Standard perceptron update (for a single sample):**

$$\mathbf{a}_{k+1} = \mathbf{a}_k + \eta \cdot y_i \quad \text{(only if } y_i \text{ misclassified)}$$

Batch version:

$$\mathbf{a}_{k+1} = \mathbf{a}_k + \eta \cdot \sum_{i \in \text{misclassified}} y_i$$

So the **total correction** is the **sum of all the bad samples**.

The Batch Perceptron algorithm updates the weight vector using multiple samples at once (a *batch*), instead of just one at a time like the single-sample version.

Intuition:

Rather than reacting to each mistake immediately, the Batch Perceptron says:

> "Let me see all the mistakes I'm currently making in this batch and then update once, based on the sum of all those mistakes."

Exercise 2: Batch Perceptron

```
while error >= theta:
```

Step 1: Pick batch_size random samples from the dataset.

Step 2: Compute the Scores. What is the dimension?

Step 3: Predict Class Labels: If score < 0 → predicted class 1, else → predicted class 0. How many predictions at each iteration?

Step 4: Update Weight

Step 5: Recalculate predictions across entire dataset for accuracy.

Exercise 2: Batch Perceptron

Step 4: Update Weight

Weight vector: a = [a1,a2,b], Sample: y = [x1,x2,1], Label: label is either 0 or 1, Prediction: checking the sign of score = a · x

When we're training with batch perceptron, we go over many samples at once.

Each sample might:

Be classified correctly ⟶ we do nothing
Be classified wrongly ⟶ we want to update a

Let batch_size=4

- y1 = [x1, x2, 1], classified correctly or wrong? If prediction=class 0 and true=class 1-> subtract
- y2 = [x3, x4, 1], classified correctly or wrong? If prediction=class 1 and true=class 0-> add
- y3 = [x5, x6, 1], classified correctly or wrong? …
- y4 = [x7, x8, 1], classified correctly or wrong? …

Algorithm 3. Batch Perceptron

1 **begin initialize** a, criterion θ, $\eta(0)>0$, k=0
2 **do** k ← k+1
3 $\mathbf{a} \leftarrow \mathbf{a}+\eta(k)\sum_{\mathbf{y}\in\mathcal{Y}_k}\mathbf{y}$
4 **until** $\left|\eta(k)\sum_{\mathbf{y}\in\mathcal{Y}_k}\mathbf{y}\right|<\theta$
5 **return** a
6 **end**

That means: Only the wrong classified samples will be used in the sum of the update rule. You have two options for the implementation of the update rule. Either loop or multiplier mask of batch size length.

Exercise 2: Batch Perceptron

To define multipliers first you need to decide if you will use the rule a=a-... or a=a+...

Label = 1, Prediction = 0

- score = a · x >= 0 ⟶ predicted class 0
- True label is 1 ⟶ wrong prediction
- **We want to make the score smaller**
- ⟶ Subtract y from a
- Multiplier = **+1-> using the multipliers is an easy way to avoid loops and ifs. It is not necessary.**

a = a - lr * (+1 * x) ⟶ a = a - lr * x

Label = 0, Prediction = 1

- score = a · x < 0 ⟶ predicted class 1
- True label is 0 ⟶ wrong prediction
- **We want to make the score bigger**
- ⟶ Add x to a
- Multiplier = **-1**

a = a - lr * (-1 * x) ⟶ a = a + lr * x

Exercise 2: Batch Perceptron

EXAMPLE 1

Two samples (with bias added):

x1 = [1, 1], label = **1** (class 1)
x2 = [-1, 1], label = **0** (class 0)

Initial weights: a = [0, 0]
Compute scores

- score(x1) = a · x1 = 0
- score(x2) = a · x2 = 0

Prediction rule:

- If score ≥ 0 ⟶ class 0
- If score < 0 ⟶ class 1

So we predict:

- x1: score = 0 ⟶ class 0 ❌ (wrong)
- x2: score = 0 ⟶ class 0 ✅ (correct)

We need to fix x1, not x2

Build multiplier_mask:

- x1: wrong, predicted class 0, should be class 1 ⟶ subtract it ⟶ +1
- x2: correct ⟶ 0

So:multiplier_mask = [1, 0]

update_step= x1 * 1 + x2 * 0 = [1, 1] + [0, 0] = [1, 1]

a = a - lr * update_step = [0, 0] - [1, 1] = [-1, -1]

Exercise 2: Batch Perceptron

EXAMPLE 2

Samples:

- x1 = [2, 1, 1], label = 1
- x2 = [1, 2, 1], label = 0

Initial weights: a = [0, 0, 0]

Step 1: Compute scores

- score(x1) = 0 ⟶ predict class 0 ❌ (should be 1)
- score(x2) = 0 ⟶ predict class 0 ✅ (correct)

Step 2: Multiplier mask

- x1 ⟶ wrong ⟶ +1
- x2 ⟶ correct ⟶ 0

⟶ multiplier_mask = [1, 0]

Step 3: Compute error

update_step = 1 * [2, 1, 1] + 0 * [1, 2, 1] = [2, 1, 1]

Step 4: Update weights

a = a - lr * update_step = [0, 0, 0] - 1 * [2, 1, 1] = [-2, -1, -1]

Exercise 2: Batch Perceptron

EXAMPLE 3

Samples:

x1 = [2, 2, 1], label = **1**

x2 = [-3, 1, 1], label = **1**

Initial weights:

a = [0, 0, 0]

Prediction rule:

score = a · x

if score ≥ 0 ⟶ class 0

if score < 0 ⟶ class 1

Step 1: Compute Scores

score(x1) = [0, 0, 0] · [2, 2, 1] = 0 ⟶ prediction = 0 ⟶ ❌ wrong

score(x2) = [0, 0, 0] · [-3, 1, 1] = 0 ⟶ prediction = 0 ⟶ ❌ wrong

Step 2: Ground Truth

Both labels = 1
Both predictions = 0 ⟶ **both wrong**

Step 3: Multiplier mask

Both predictions wrong ⟶ multiplier = +1 for both

Step 4: Compute Error

update_step = 1 * x1 + 1 * x2 = [2, 2, 1] + [-3, 1, 1] = [-1, 3, 2]

Step 5: Update weights

a = a - lr *update_step

Thank you!