# CAP 4630 – Perceptron

**Instructor:** Aakash Kumar

University of Central Florida

# Introduction to the Perceptron

- What is the Perceptron?
  - The perceptron is one of the simplest machine learning algorithms, used for binary classification.
- What does it do?
  - It helps classify data into one of two categories by finding a decision boundary (a line) that separates the two classes.
- Key Points:
  - Works well when data can be separated by a straight line (linearly separable).
  - Lays the foundation for understanding more complex algorithms like neural networks.

# Learning Linear Separators

- What is a Linear Separator?
  - A line (in 2D) or a hyperplane (in higher dimensions) that divides data into two classes.
- Why is it important?
  - Many machine learning problems involve classifying data, and in some cases, a straight line can effectively separate different groups of data.
  - Perceptron tries to find the best possible line to make these classifications.
- Key Takeaway:
  - If the data can be separated by a line, perceptron is an effective and simple method.

# How Perceptron Works

- Step-by-step process:
  - Each input feature is given a weight.
  - These inputs are multiplied by their weights and summed up (dot product).
  - If the sum is greater than a threshold, the output is one class (e.g., 1), otherwise, it's another class (e.g., 0).
- Key Components:
  - Inputs (Features): The data you're using to make a decision.
  - Weights: These help determine the importance of each input.
  - Activation Function: Decides which class the input belongs to (often just a simple threshold).

# Perceptron Learning Rule

- Perceptron adjusts its weights based on the errors it makes.
- If the prediction is wrong, the weights are updated to correct for this mistake.
- Over time, these updates help the perceptron find the best decision boundary.

# Online Learning Model

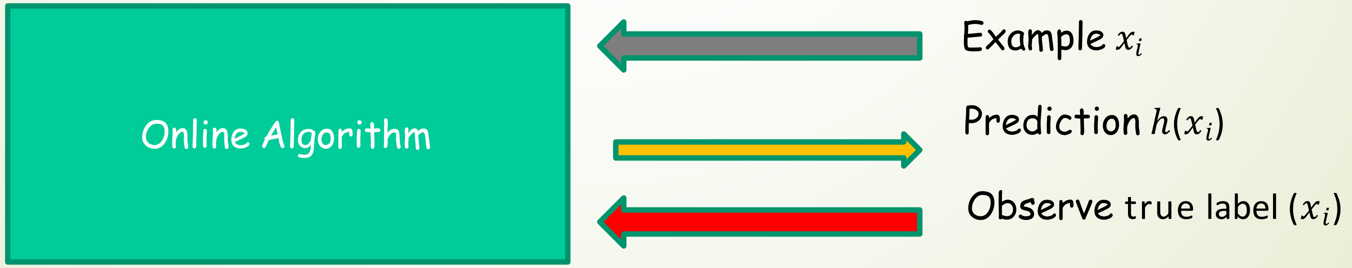
- How it works:
  - Unlike some models, which wait until they see all the data before learning (batch learning), online learning models update their knowledge immediately as each data point arrives.
  - Imagine learning a new word: instead of learning it at the end of a language course, you learn it the moment you hear it.
- Why is it useful?
  - Efficient for streaming data or large datasets where we can't afford to wait until all the data is available.
  - Useful when data arrives continuously over time, like in real-time applications (e.g., stock prices, weather updates).

# How Perceptron Learns

- Step-by-Step:
  - The perceptron sees a data point and makes a prediction.
  - It checks if the prediction was correct.
  - If the prediction was wrong, it adjusts its weights to be more accurate next time.
  - Repeat for each new data point.
- Key Concept:
  - Perceptron updates its knowledge after every data point, meaning it is constantly learning and improving.

# How the Online Learning Model Works

- Step-by-Step Process in Online Learning
  - Receive an example: The model is presented with a new data point (e.g., an email or a financial transaction).
  - Make a prediction: Based on its current knowledge, the model predicts a label or outcome (e.g., spam or not spam).
  - Observe the true label: After the prediction, the true label (actual outcome) is revealed.
  - Update the model: If the prediction was wrong, the model adjusts its weights to improve future predictions.



# The Mistake Bound Model

- What's the goal?
  - The primary goal is to minimize the number of mistakes the model makes over time.
- What's the challenge?
  - The model doesn't assume the data follows a specific pattern or distribution.
- Key Point:
  - Unlike traditional models that wait for all data, online learning focuses on reducing errors as quickly as possible while continuously learning.

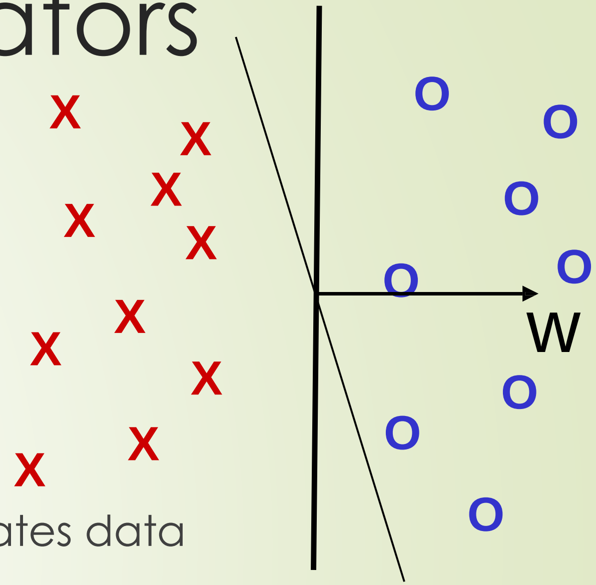
# How Does the Perceptron Fit Into This?

- Perceptron and Mistake Bound:
  - The perceptron is a classic example of an online learning algorithm that can be analyzed using the Mistake Bound model.
  - In simple terms, for linearly separable data (data that can be separated by a straight line), the perceptron algorithm will make a limited number of mistakes before it converges to the correct solution.
- Bound on the Number of Mistakes:
  - For linearly separable data, the **number of mistakes** the perceptron makes is **bounded**. This means that there is a limit to how many mistakes it will make before it learns the correct decision boundary.
  - This mistake bound depends on the **margin** of separation between the classes (how far apart the classes are) and the size of the input data. If the margin is large, the perceptron will make fewer mistakes.

# Where Do We Use Online Learning?

- Email classification (spam detection):
  - Every day, emails are classified as spam or not. Over time, the nature of spam emails may change, but the ability to recognize what is likely spam remains important.
  - An email from last year that was spam would still be considered spam today, even if the types of spam emails have evolved.
- Recommendation systems:
  - Online platforms like Netflix or YouTube use recommendation systems to suggest movies, shows, or videos based on your past preferences.
  - As you interact with the platform, the system continuously updates its recommendations in real-time.
- Predicting user interest in news articles:
  - News websites predict whether a user will be interested in a specific article based on their reading behavior.
  - As the user clicks on articles, the recommendation engine learns and refines its future predictions.

# Classifying Data with Linear Separators




- Feature Space: $X = R^d$
  - means that each data point has **d** features (e.g., a 3D space has three features).
- Linear Decision Surfaces:
  - A **linear decision surface** is a line (or hyperplane in higher dimensions) that separates data into two categories.
- How Predictions are Made:
  - The perceptron makes a prediction based on a **linear equation:**

$$h(x) = w \cdot x + w_0$$

  - Here, w is the weight vector, and x is the input vector (the features).
  - if the result of this equation is greater than or equal to 0, the data point is labeled as positive (+). Otherwise, it's labeled as negative (-).

# Linear Separators: Perceptron Algorithm

- **Step 1:** Set the initial conditions:
  - Start with **t = 1** (the first iteration).
  - Initialize the weight vector w1 to be a vector of all zeros (this means the algorithm knows nothing at first).
- Make a prediction:
  - For each new example x, predict its label as **positive** if:

$$w_t \cdot x \geq 0$$

  - If the result is positive or zero, predict the example is in the positive class. Otherwise, predict it's in the negative class.

# What Happens When We Make a Mistake?

- **Mistake on a positive example**:
  - If the algorithm predicts that the example is negative but it's actually positive, we **increase** the weight vector.
  - Update rule:

$$w_{t+1} \leftarrow w_t + x$$

  - This adjustment moves the decision boundary closer to where the mistake was made.
- **Mistake on a negative example**:
  - If the algorithm predicts that the example is positive but it's actually negative, we **decrease** the weight vector.
  - Update rule: $$w_{t+1} \leftarrow w_t - x$$
  - This moves the boundary away from the incorrectly classified point.

# Perceptron Algorithm: Example

Example: $(-1,2)$ – ✗

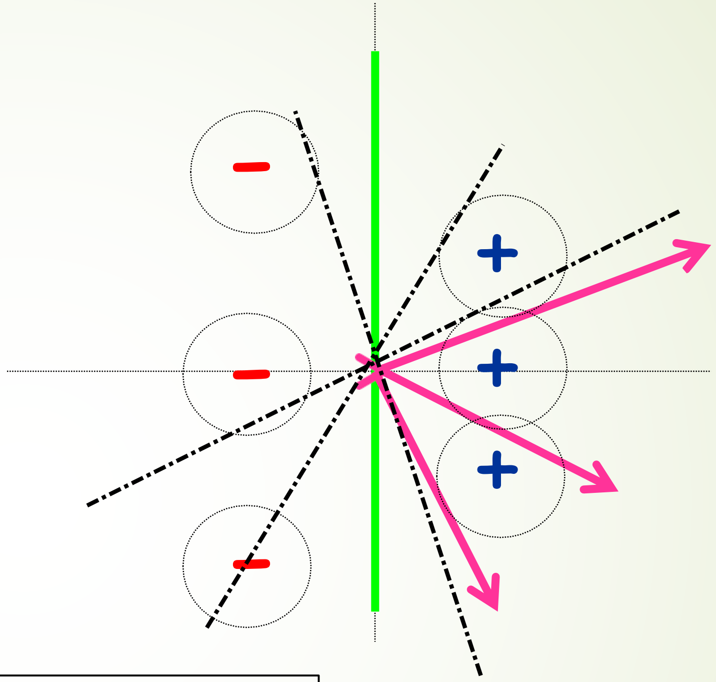
$(1,0)$ + ✓

$(1,1)$ + ✗

$(-1,0)$ – ✓

$(-1,-2)$ – ✗

$(1,-1)$ + ✓




$w_1 = (0,0)$

$w_2 = w_1 - (-1,2) = (1,-2)$

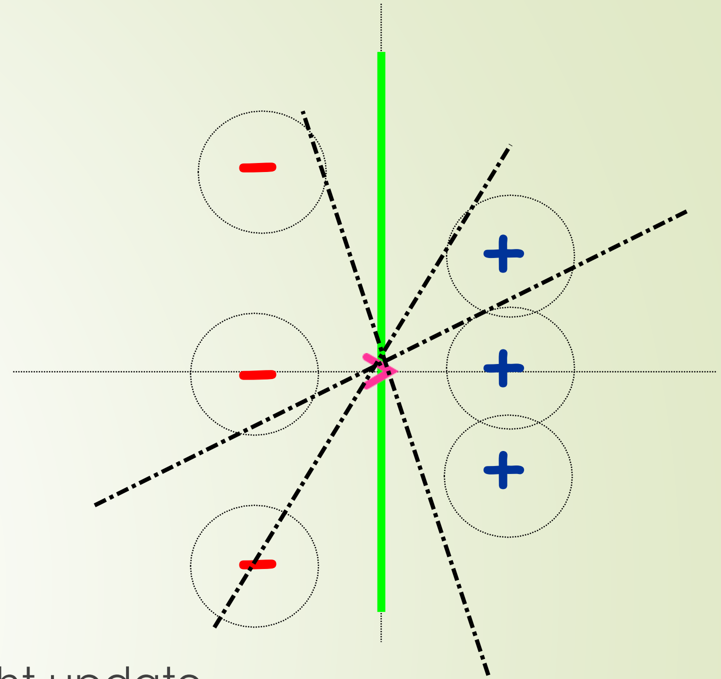
$w_3 = w_2 + (1,1) = (2,-1)$

$w_4 = w_3 - (-1,-2) = (3,1)$

**Algorithm:**

- Set t=1, start with all-zeroes weight vector $w_1$.
- Given example $x$, predict positive iff $w_t \cdot x \geq 0$.
- On a mistake, update as follows:
  - Mistake on positive, update $w_{t+1} \leftarrow w_t + x$
  - Mistake on negative, update $w_{t+1} \leftarrow w_t - x$

# Handling Mistakes



- First Example (−1,2):
  - Prediction: $W_1 \cdot (-1,2) = 0$. Since this is 0, we predict positive.
  - **Mistake:** The true label is negative (–), so we update the weights using:

$$w_2 = w_1 - (-1,2) = (0,0) - (-1,2) = (1,-2)$$

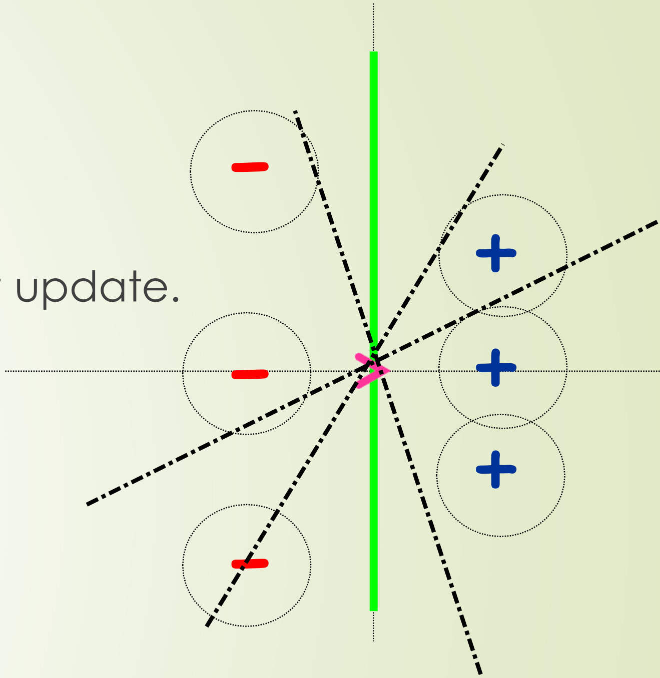
- Second Example (1,0):
  - Prediction: W2 · (1,0) = 1. We predict positive, which is correct, so no weight update.
- Third Example (1,1):
  - Prediction: W2 · (1,1) = -1. We predict negative, but the true label is positive.
  - **Mistake:** Update the weights using

$$w_3 = w_2 + (1,1) = (1,-2) + (1,1) = (2,-1)$$

# Handling Mistakes

- Fourth Example (−1,0):
  - Prediction: W3 · (−1,0) = -2, We predict negative, which is correct, so no weight update.
- Fifth Example (−1,−2):
  - Prediction: W3 · (-1,-2) = 0. We predict positive, but the true label is negative.
  - **Mistake:** Update the weights using:

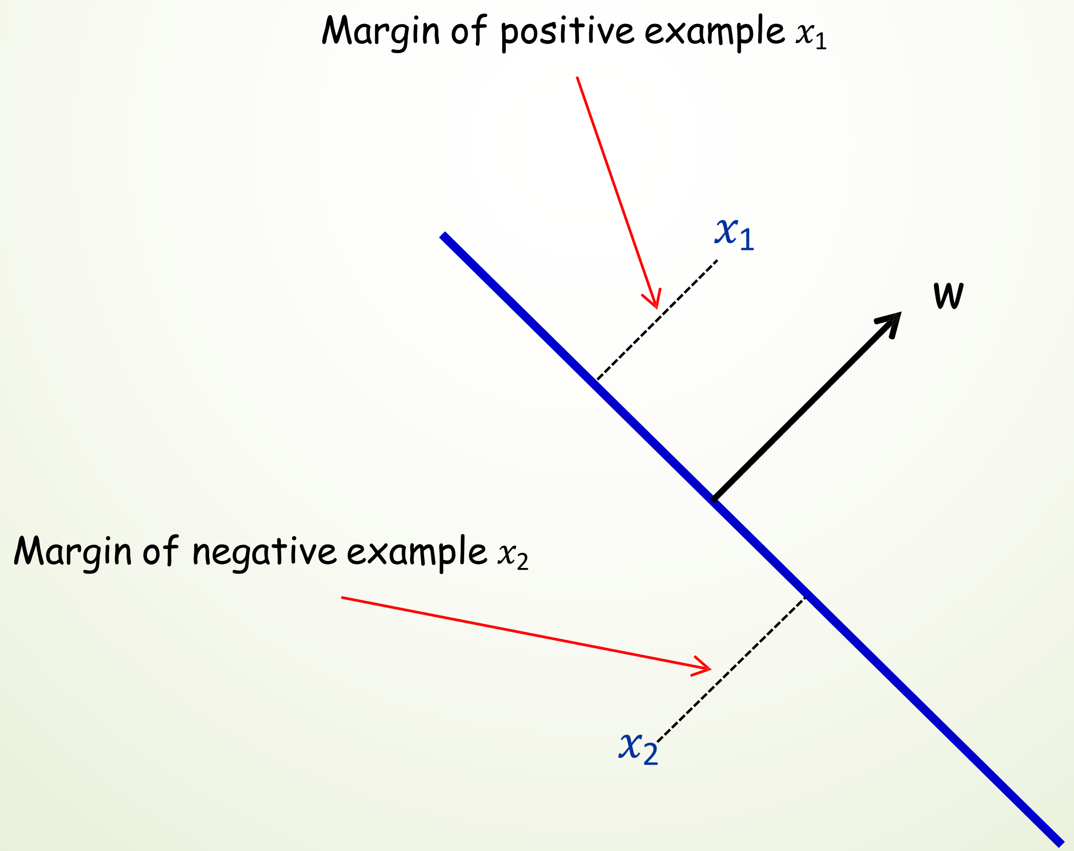
$$w_4 = w_3 - (-1,-2) = (2,-1) - (-1,-2) = (3,1)$$




- Sixth Example (1,−1):
  - Prediction:  W4 · (1,−1) = 2. We predict positive, which is correct, so no weight update.

# Geometric Margin

- The **geometric margin** tells us **how far** a point is from the decision boundary.
  - A **larger margin** means the point is far from the boundary, which typically indicates a more confident and accurate classification.
  - A **smaller margin** means the point is closer to the boundary, making the classification less confident.



# Geometric Margin

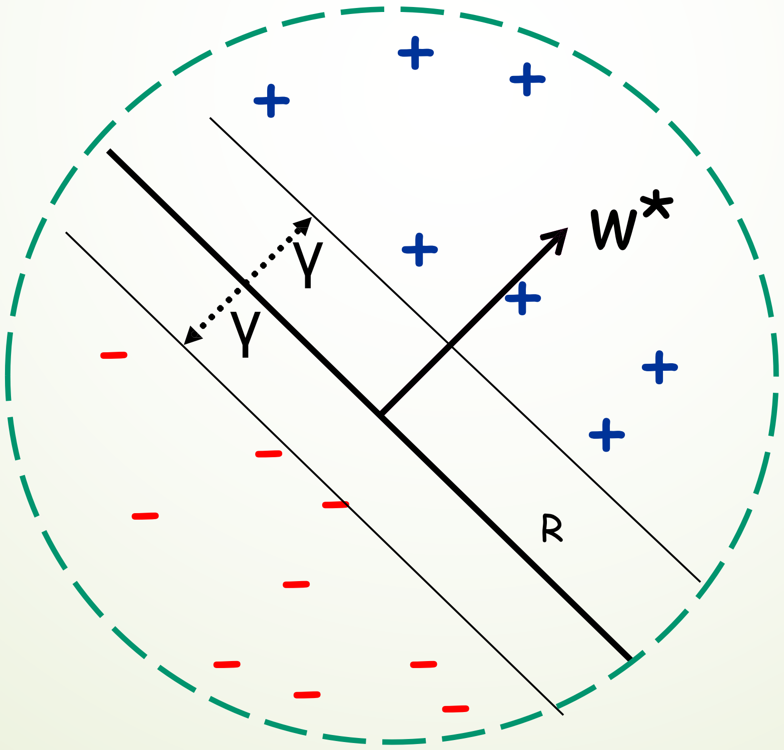
- Larger Margins are Better:
  - If the margin is large, the model has made a strong, confident decision. For example, a point far from the boundary is clearly in one class or the other.
  - Models that maximize the margin (such as Support Vector Machines) are often more robust because they leave a wider "buffer" zone between classes.

- Small or Negative Margins:
  - If the margin is small or negative, the point is close to being misclassified, or it’s already misclassified.
  - This can be a sign that the decision boundary needs adjusting.

# Perceptron: Mistake Bound

**Guarantee**: If data has margin $\gamma$ and all points inside a ball of radius $R$, then Perceptron makes ≤ $(R/\gamma)^2$ mistakes.

(Normalized margin: multiplying all points by 100, or dividing all points by 100, doesn't change the number of mistakes; algo is invariant to scaling.)



# Perceptron: Mistake Bound

- Guarantee:
  - The **mistake bound** for the perceptron algorithm gives us a mathematical guarantee of how many mistakes the perceptron will make during training.
  - Specifically, if the data has a margin γ and all points lie inside a ball of radius R, then the perceptron will make no more than following mistakes.

$$\left(\frac{R}{\gamma}\right)^2$$

- What does this mean?
  - The number of mistakes is **inversely proportional** to the margin γ. A **larger margin** means fewer mistakes, while a **smaller margin** means more mistakes.
  - The number of mistakes also depends on the radius R of the data points. Larger data points (points spread out over a larger space) can lead to more mistakes.

# Perceptron Extensions

- Perceptron in a Batch Setting:
  - While the perceptron algorithm is typically used in an online setting (learning from one example at a time), we can also use it in a batch setting.
  - In the batch setting, we have a set S of labeled examples, and we want to find a linear separator that is consistent with all examples.
- In this scenario, we repeatedly feed the entire set S of labeled examples into the perceptron algorithm until we find a linear separator that correctly classifies all the points.
- If the data is **linearly separable** by margin $\gamma$, the perceptron will make at most following number of passes over the entire set before finding a consistent hypothesis (correct separator).

$$\left(\frac{R}{\gamma}\right)^2 + 1$$

# Conclusion: Key Takeaways

- The Perceptron Algorithm:
  - A simple but powerful linear classifier used for binary classification.
  - Works well when data is linearly separable.
  - Adjusts the decision boundary based on mistakes and updates the weight vector accordingly.
- Online and Batch Settings:
  - Perceptron can be used in an online setting (one data point at a time) or in a batch setting (entire dataset at once).
  - Both settings allow the algorithm to find a consistent linear separator, given separable data.
- Mistake Bound:
  - This gives a bound on how long it will take the perceptron to find a correct separator.

$$\left(\frac{R}{\gamma}\right)^2 + 1$$

# Strengths of the Perceptron

- Simplicity:
  - The perceptron is easy to understand and implement.
  - It requires simple updates based on the current classification errors.
- Guaranteed Convergence:
  - If the data is linearly separable, the perceptron is guaranteed to find a correct decision boundary.
- Online Learning:
  - The perceptron can be used for online learning, making it suitable for scenarios where data arrives continuously or in a stream.

# Limitations of the Perceptron

- Linearly Separable Data:
  - The perceptron works only when the data is linearly separable. If the data cannot be separated by a straight line, the perceptron will not converge.
- No Confidence Scores:
  - Unlike more advanced algorithms, the perceptron doesn’t provide confidence or probability estimates for its predictions.
- Sensitive to Noisy Data:
  - If the data contains outliers or mislabeled points, the perceptron may struggle to find a good boundary, especially if the margin $\gamma$ is small.

# Extensions and Beyond

- Support Vector Machines (SVMs):
  - SVMs are an extension of the perceptron that maximize the margin, resulting in more robust classifiers.
  - SVMs can also handle non-linearly separable data using kernel tricks, which transform the data into higher dimensions where linear separation is possible.
- Neural Networks:
  - The perceptron is the foundation for more complex neural networks, which can handle non-linear decision boundaries by stacking multiple layers of perceptrons (neurons).