

# Machine Learning and Data Mining I: Lecture 4

## Linear Classification: Perceptron

Morgan McCarty

10 July 2023

## 1 Classification

Predict a discrete label rather than a continuous value.

### 1.1 Artificial Neural Network

- Also known as: Connectionist Model, Adaptive Network
- Inspired by human brain
  - Composed of billions of neurons
  - Fast processing of complex tasks
  - Parallel computations
- Started with Autonomous Land Vehicle (ALVINN) in 1990s.
- Deep Learning:
  - ANNs and variants
  - State of the art for speech recognition
  - Convolutional Neural Networks (CNNs) for image recognition (pattern recognition)
- Tasks:
  - Continuous input features
  - Form of tangent function is unknown
- Drawbacks:
  - Very hard to understand/complex
  - Mostly treated as black box

### 1.2 Perceptron

- Based on structure of neuron: connect to others using synapses, synapses have varying strength (exciting + or inhibiting −)

### 1.3 Binary Linear Classification

- Linearly separable Data
- Boundary is linear function of input features (line, plane, hyperplane, etc)
- Input:  $x \in \mathbb{R}^m$
- Output:  $y \in \{-1, 1\}$

### 1.4 Perceptron Classification Rule

$$w_1x_1 + w_2x_2 + \dots + w_mx_m \geq \theta \rightarrow y = 1$$

else:  $y = -1$

- $w_i$  is the weight of the  $i$ th feature
- $x_i$  is the  $i$ th feature
- $\theta$  is the threshold (bias)

### 1.5 Learning Rate

- Learn the  $m + 1$  parameters:  $w_1, w_2, \dots, w_m, \theta$
- Absorb the threshold into the weight vector:  $w_0 = -\theta$  (an extra feature)

$$f(x) = w_0x_0 + w_1x_1 + w_2x_2 + \dots + w_mx_m$$

where  $x_0 = 1$   
and  $w_0 = -\theta$

### 1.6 Algorithm

- Data: Training Data =  $\{x_i, y_i : \forall i = 1, 2, \dots, n\}$
- Learning Rate:  $\eta$
- Result: Coefficients of the hyperplane  $(w_0, w_1, w_2, \dots, w_m)$
- Algorithm:

- Initialize  $w_0, w_1, w_2, \dots, w_m$  to 0
- Repeat until convergence:
  - \* For each  $(x_i, y_i)$  in training data:
    - Compute  $\hat{y}_i = w^T x_i$
    - If  $\hat{y}_i y_i \leq 0$  then update  $w$ :

$$w = w + \eta y_i x_i$$

- Repeat until  $\hat{y}_i y_i > 0$  for all  $(x_i, y_i)$

## 1.7 Perceptron Learning Example

Learning the boolean *AND* function:

- We have a data table:

$x_0$	$x_1$	$x_2$	$AND(x_1, x_2)$
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

- We want to learn the weights  $w_0, w_1, w_2$  such that  $w_0 + w_1x_1 + w_2x_2 \geq 0$  if  $x_1 = 1$  and  $x_2 = 1$ .
- Initialize  $w_0, w_1, w_2$  to 0.
- Set  $\eta$  to 1
  - For each  $(x_i, y_i)$  in training data:
    - \* Compute  $\hat{y}_i = w^T x_i$
    - \* If  $\hat{y}_i y_i \leq 0$  then update  $w$ :

$$w = w + \eta y_i x_i$$

- \* Repeat until  $\hat{y}_i y_i > 0$  for all  $(x_i, y_i)$

- Let's go through the algorithm step by step for one epoch:

- The first data point is  $(1, 0, 0, 0)$

- $\hat{y}_1 = w_0 + w_1x_1 + w_2x_2 = 0$
  - $\hat{y}_1 y_1 = 0 \leq 0$
  - $w = w + \eta y_i x_i = (0, 0, 0) + 1(0, 0, 0) = (0, 0, 0)$

- The second data point is  $(1, 0, 1, 0)$

- $\hat{y}_2 = w_0 + w_1x_1 + w_2x_2 = 0$
  - $\hat{y}_2 y_2 = 0 \leq 0$
  - $w = w + \eta y_i x_i = (0, 0, 0) + 1(0, 0, 1) = (0, 0, 1)$

- The third data point is  $(1, 1, 0, 0)$

- $\hat{y}_3 = w_0 + w_1x_1 + w_2x_2 = 1$
  - $\hat{y}_3 y_3 = 0 \leq 0$
  - $w = w + \eta y_i x_i = (0, 0, 1) + 1(0, 1, 0) = (0, 1, 1)$

- The fourth data point is  $(1, 1, 1, 1)$

- $\hat{y}_4 = w_0 + w_1x_1 + w_2x_2 = 2$
  - $\hat{y}_4 y_4 = 1 > 0$
  - We do not update  $w$ .

- We have completed one epoch (one iteration through the data).

- The current weights are  $w = (0, 1, 1)$ ,  $y_w(x) = 0 + 1x_1 + 1x_2 = x_1 + x_2$ .
- As we still had errors for the first three data points, we will continue to iterate (we have not converged).
- When we finally have no errors, we have converged.
- After convergence, we have  $w = (-4, 3, 2)$ ,  $y_w(x) = -4 + 3x_1 + 2x_2$ .
- The exact values of the final weight vector depend on the initial values as well as the learning rate.
- This would not work for the *XOR* function, as it is not linearly separable.

## 1.8 Questions?

- Will the algorithm always converge for linearly separable data?
  - For two-class linearly separable datasets there is a separating hyperplane  $w_{opt}$ .
  - Will the algorithm converge over finite iterations?
- What loss function does the algorithm minimize?
- We assume that the training data has bounded Euclidean norm. (i.e.  $\|x_i\| \leq R$  for all  $i$ )
- The optimal classifier perfectly separates the data (i.e.  $y_i w_{opt}^T x_i > 0$  for all  $i$ )

## 1.9 Convergence

- The algorithm learns the boundary in a finite number of steps based on the number of mistakes it makes starting from  $w_0$ .
- The mistake bound is:

$$k \leq \frac{R^2}{\gamma} \|w_{opt}\|^2 \quad (1)$$

- However most real-world data is not linearly separable.

## 1.10 Loss to Optimize

- The loss is 0 when the label and prediction agree

$$y_i f_w(x_i) > 0 \quad (2)$$

- The loss is  $\geq 0$  when the label and prediction values do not agree (i.e. have opposite signs)

$$y_i f_w(x_i) \leq 0 \quad (3)$$

- Therefore the loss function is:

$$L_p(x, y, w) = \max(0, -yw^T x) \quad (4)$$

- Non-zero for misclassified points

## 1.11 Gradient of Loss Function

- The gradient of the loss function is:

$$L_p = \sum_{i=1}^N \max(0, -y_i w^T x_i)$$

$$L_p = -\sum_{i \in \mu_w} y_i w^T x_i$$

$$\nabla L_p = -\sum_{i \in \mu_w} y_i x_i$$

## 1.12 Stochastic Gradient Descent

- Batch Gradient Descent

Processing the entire training set for each epoch can be prohibitive for large datasets

- Stochastic Gradient Descent

- Update weights after each training example
- Faster convergence
- Noisy gradient

## 1.13 Linearly Inseparable Data

- Can we apply the perceptron to linearly inseparable data?
- Change in criteria:
  - Loss: find separator that makes no mistakes  $\rightarrow$  find separator that makes least misclassifications (“best fit”)
- Squared Loss:
  - Between perceptron output and class labels
  - Output: Un-thresholded
  - $o(x) = w^T x$
  - $E(w) = \frac{1}{2} \sum_{i=1}^N (y_i - o_i)^2$
  - $y_i \in \{0, 1\}$
  - $o_i \in \mathbb{R}$
  - Squared loss increases with values on correct side of boundary