2*

Linear Regression
{Lecture Notes}

$x^{(i)} \Rightarrow$ i^{th} input feature in training set
$y^{(i)} \Rightarrow$ i^{th} output in traing set

$(x^{(i)}, y^{(i)}) \Rightarrow$ i^{th} training example

$\{(x^{(i)}, y^{(i)}) \mid 1 \leq i \leq m\} \Rightarrow$ training set

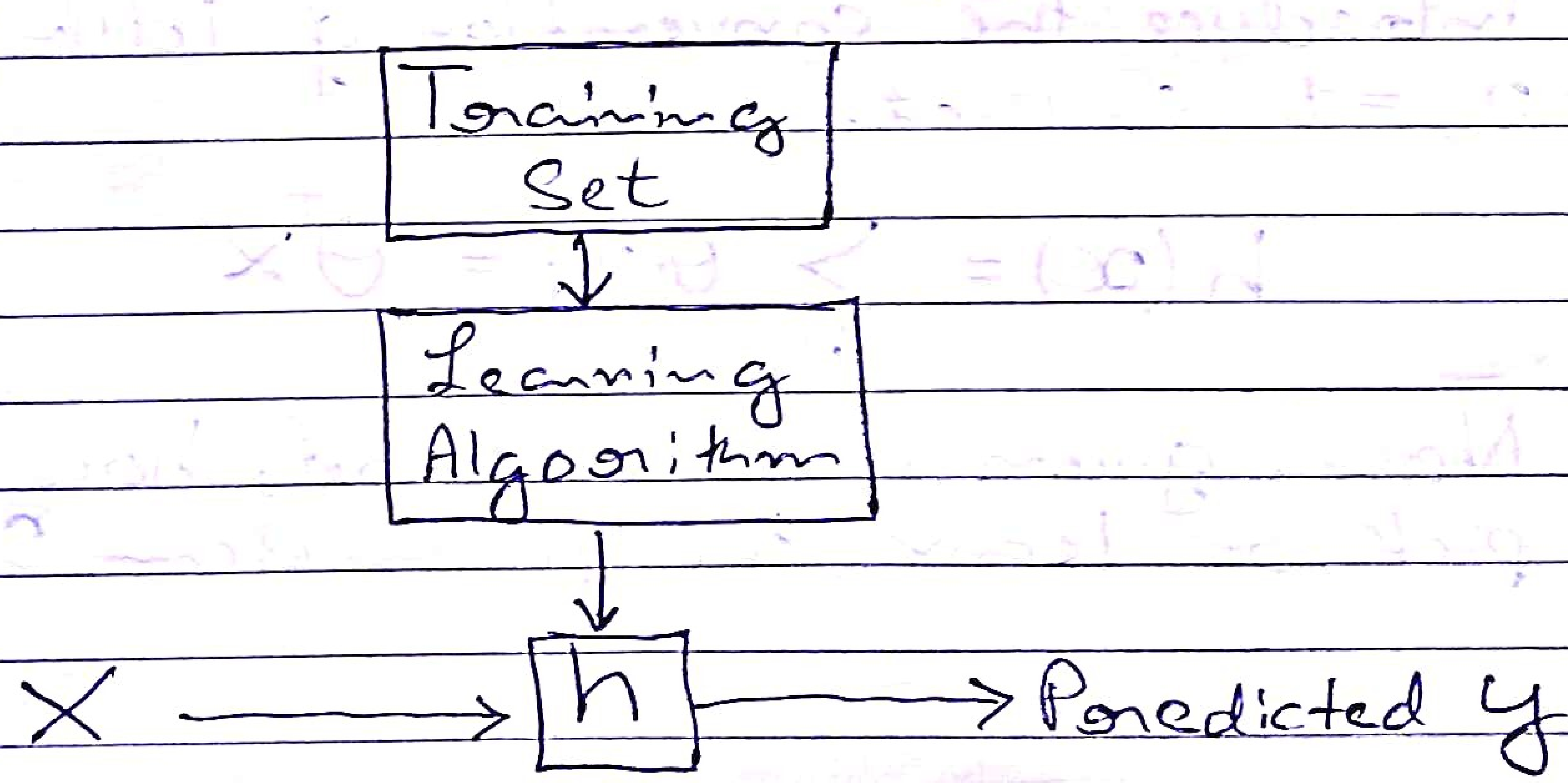
$\mathcal{X} \Rightarrow$ denotes the Space of input features.

$\mathcal{Y} \Rightarrow$ denotes the Space of output
$m \Rightarrow$ # training example

★ **Supervised learning**

"Given a training set, learn a function $h : \mathcal{X} \mapsto \mathcal{Y}$, so that $h(x)$ is a "good" predictor for the corresponding value of y"

↳ For historical reasons, this function h is called a hypothesis.



Linear Regression

⇒ To perform supervised learning, we must decide how we're going to represent functions/hypotheses h in a Computer.

⇒ As an initial choice, lets say we decide to approximate y as a linear function of x:

$$h_\theta(x) = \theta_0 + \theta_1 X_1 + \theta_2 X_2$$

$$\forall\, x = R^2$$

⇒ Here, the θ_i's are the parameters parameterizing the space of linear functions mapping X to Y.

⇒ To simplify our notation, we also introduce the convention of letting $x_0 = 1$ so that,

$$h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T X$$

⇒ Now, given a training set, how do we pick, or learn, the parameters θ?

Date
Page

⇒ One reasonable method seems to be to make h(x) close to y, at least for the training examples we have.

$$\boxed{J(\theta) = \frac{1}{2} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2}$$

★ <u>LMS algorithm</u>

⇒ We want to choose θ so as to minimize J(θ).

⇒ To do so, lets use a search algorithm that starts with some initial "guess" for θ and that repeatedly change θ to make J(θ) smaller, until hopefully we converge to a value of θ that minimizes J(θ)

⇒ Let us consider the gradient descent algorithm.

⇒ Start with some initial θ, and repeatedly perform the update

$$\theta_j := \theta_j - \alpha \frac{\delta}{\delta \theta_j} J(\theta)$$

for $j = 0, \ldots n$

α ⇒ Learning rate

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

$$\frac{\delta J(\theta)}{\delta \theta_j} = \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \frac{\delta}{\delta \theta_j} h_\theta(x^{(i)})$$

$$\frac{\delta}{\delta \theta_j} \sum_{K=0}^{m} \theta_K X_K^{(i)}$$

$$X_j^{(i)}$$

$$\frac{\delta J(\theta)}{\delta \theta_j} = \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) X_j^{(i)}$$

$$\boxed{\theta_j := \theta_j - \alpha \sum_{i=1}^{m} \left(h_\theta(X^{(i)}) - Y^{(i)} \right) X_j^{(i)}}$$

⇒ This rule is called the LMS update rule and is also known as the Widrow - Hoff learning rule.

⇒ This method looks at every example in the entire training set on every step, and is called batch gradient descent.

Date
Page

⇒ The cost function $J(\theta)$ is a convex quadratic function, so it only has one global optimum.

↳ Hence gradient descent will always converge to global optimum.

⇒ There is an alternative to batch gradient descent that also works very well.

for $i=1$ to m {

$$\theta_j := \theta_j - \alpha\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)} \quad \left\{\begin{matrix}\text{FOR} \\ \text{EVERY} \\ j\end{matrix}\right\}$$

}

⇒ This algorithm is called stochastic gradient descent.

⇒ Batch gradient descent has to scan through the entire training set before taking a single step.

{A costly operation if m is large}

⇒ Stochastic gradient descent can start making progress right away, and continue to make progress with each examples it looks at.

⇒ On the down side, Stochastic gradient descent may never converge to the minimum, and the parameters θ will keep oscillating around the minimum of $J(\theta)$.

↳ But in practice most of the values near the minimum will be reasonable good approximations to the true minimum.

★ <u>The normal equations</u>

SAME