Notation

- Supervised Learning involves supplying a number (m) of examples.
- Each example is a pair consisting of
    - vector $\mathbf{x}$ consisting of $n \geq 1$ *features* (attributes)
    - scalar (sometimes a vector) $\mathbf{y}$
        - referred to as the *target* value or *label* associated with $\mathbf{x}$
- we use **bold face** to indicate a vector (e.g, $\mathbf{x}$)

- We use superscript $\mathbf{(i)}$ to index examples, when we have more than one
  - $\mathbf{x}^{\mathbf{(i)}}, \mathbf{x}^{(i')}, i \neq i'$ are two distinct examples
  - denote an element i of a collection of m examples (e.g., $\mathbf{x}^{\mathbf{(i)}}$)
- We use subscript j to index element j of a vector, e.g., $\mathbf{x}_j^{\mathbf{(i)}}$

- So $\mathbf{x^{(i)}}$ is

$$\mathbf{x^{(i)}} = \begin{pmatrix} \mathbf{x_1^{(i)}} \\ \mathbf{x_2^{(i)}} \\ \vdots \\ \mathbf{x_n^{(i)}} \end{pmatrix}$$

Each element of $\mathbf{x^{(i)}}$ is a "feature"

- $\mathbf{x_j^{(i)}}$ is the j^{th} feature of example i

Training set

- The collection of examples used for fitting (training) a model is called the *training set*:

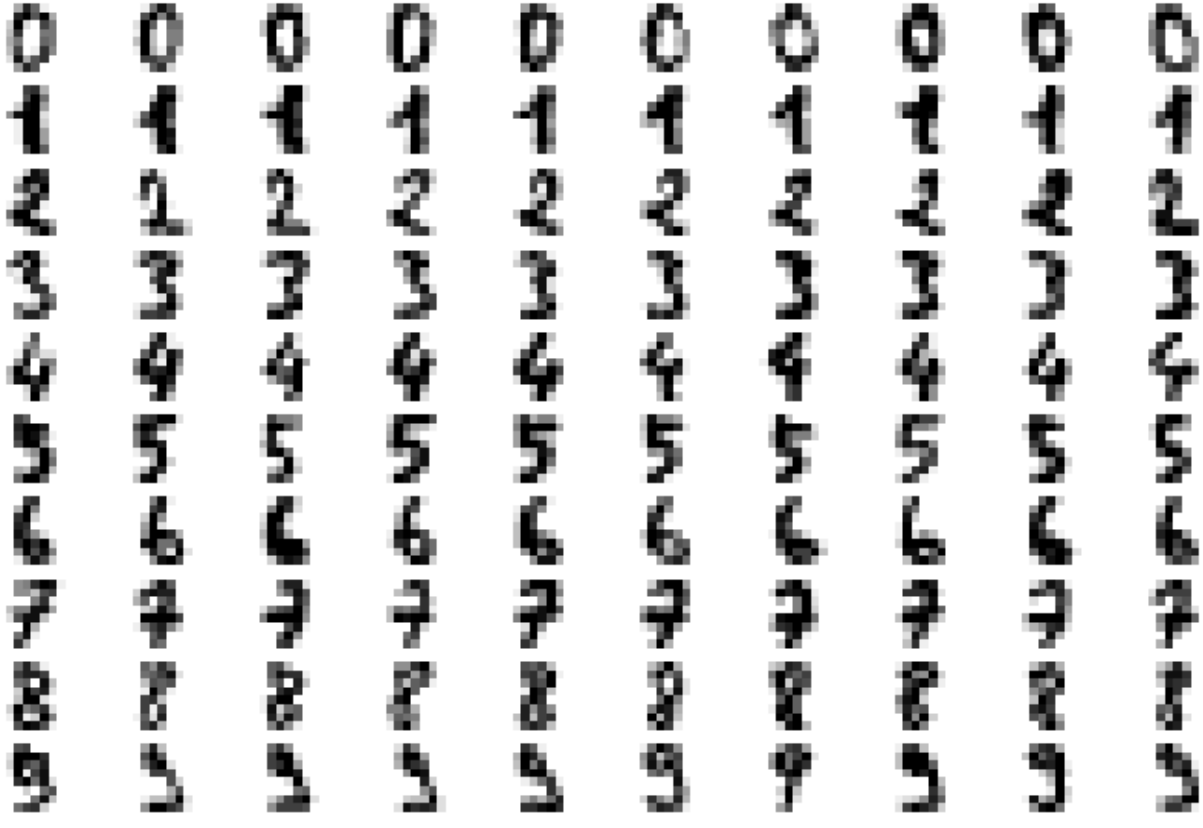
$$\langle \mathbf{X}, \mathbf{y} \rangle = [\mathbf{x^{(i)}}, \mathbf{y^{(i)}} | 1 \leq i \leq m]$$

where m is the size of training set and each $\mathbf{x^{(i)}}$ is a feature vector of length n.

- By seeing many (m) pairs of feature vectors and associated labels we will try to infer the correct label $\mathbf{y^{(i)}}$ from the features in $\mathbf{x^{(i)}}$
- $\mathbf{X}$ is an $(m \times n)$ matrix and $\mathbf{y}$ is an $(m \times 1)$ vector of targets.

$$\mathbf{X} = \begin{pmatrix} (\mathbf{x}^{(1)})^T \\ (\mathbf{x}^{(2)})^T \\ \vdots \\ (\mathbf{x}^{(m)})^T \end{pmatrix} = \begin{pmatrix} \mathbf{x}_1^{(1)} \dots \mathbf{x}_n^{(1)} \\ \mathbf{x}_1^{(2)} \dots \mathbf{x}_n^{(2)} \\ \vdots \\ \mathbf{x}_1^{(m)} \dots \mathbf{x}_n^{(m)} \end{pmatrix}$$

Training set



- We will sometimes add a "constant" feature by setting $\mathbf{x}_0^{(\mathbf{i})} = 1, 0 \leq i \leq m$ so that the first column of $\mathbf{x}$ is 1:

$$\mathbf{X} = \begin{pmatrix} 1 & \mathbf{x}_1^{(1)} & \dots & \mathbf{x}_n^{(1)} \\ 1 & \mathbf{x}_1^{(2)} & \dots & \mathbf{x}_n^{(2)} \\ \vdots & \vdots & \dots & \vdots \\ 1 & \mathbf{x}_1^{(m)} & \dots & \mathbf{x}_n^{(m)} \end{pmatrix}$$

- So each of the m rows is an example and each of the n columns is a feature.

Not just numbers !

The features *aren't restricted to be numeric* !

In this course, we will deal with data that is

- numeric
- categorical
- text
- image
- sound (not this course)

Of course, you'll have to encode this data as numbers in order for numerical algorithms to handle them.

Prediction

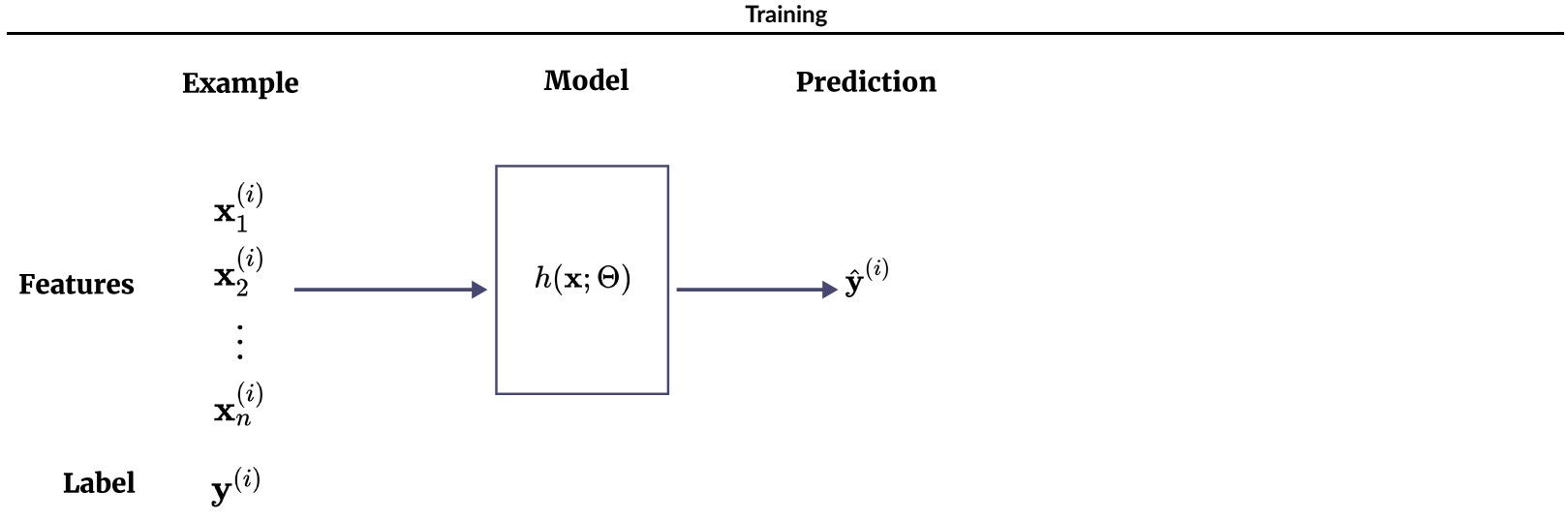
- Given training example $\mathbf{x^{(i)}}$, we construct a function h to predict its label

$$\hat{\mathbf{y}}^{\mathbf{(i)}} = h(\mathbf{x^{(i)}})$$

- We use a "hat" to denote predictions: $\hat{\mathbf{y}}^{\mathbf{(i)}}$
- The function h will often be parameterized (by Θ) so, for clarity, we should write

$$\hat{\mathbf{y}}^{\mathbf{(i)}} = h(\mathbf{x^{(i)}}; \Theta)$$

- We will often drop Θ for ease of reading.
- Since h is a function, it should also be possible to make a prediction for a vector $\mathbf{x}$ that is **not** part of the training set.
- That is, we are able *generalize* to non-training examples: to make out of sample predictions

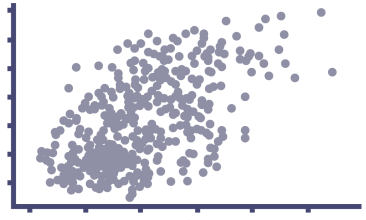

Training
Example
Model
Prediction
Features
$\mathbf{x}_1^{(i)}$
$\mathbf{x}_2^{(i)}$
$\mathbf{x}_n^{(i)}$
Label
$\mathbf{y}^{(i)}$
$h(\mathbf{x};\Theta)$
$\hat{\mathbf{y}}^{(i)}$

The key task of Machine Learning is finding the "best" values for parameters Θ.

The process of using training examples $\mathbf{X}$ to find Θ

- is called *fitting* the model
- is solved as an optimization problem (to be described)

Fitting a Linear Regression model



Training examples

$\langle \mathbf{X}, \mathbf{y} \rangle$

$\mathbf{X} : m \times n$

$\mathbf{y} : m \times 1$

Model

$h_\Theta(\mathbf{x}) = \Theta_0 * \mathbf{x}_0 + \Theta_1 * \mathbf{x}_1$

$= \Theta^T \cdot \mathbf{x}$

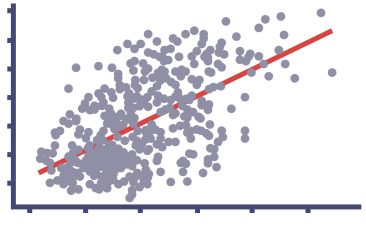
Fit

Fitting, training

$\Theta = [\Theta_0, \Theta_1] = [\text{ intercept, slope }]$

$\Theta : (n+1) \times 1$

$\mathbf{x}^{(i)} = \left[1, \mathbf{x}_1^{(i)}, \ldots, \mathbf{x}_n\right] : (n+1) \times 1$



Summary

- a training example is a pair $(\mathbf{x}^{(\mathbf{i})}, \mathbf{y}^{(\mathbf{i})})$ drawn from training set $\langle \mathbf{X}, \mathbf{y} \rangle$ consisting of
    - a feature vector $\mathbf{x}^{(\mathbf{i})}$ of length n
    - the associated label (target) $\mathbf{y}^{(\mathbf{i})}$
    - $\mathbf{X}$ is of dimension $m \times n$
    - $\mathbf{y}$ is dimension $m \times 1$, i.e., target is a single, continuous value per example
- predictions are indicated with a "hat:
    - $\hat{\mathbf{y}}^{(\mathbf{i})}$ is the prediction made given $\mathbf{x}^{(\mathbf{i})}$ as input

Loss/Cost, Utility

- The prediction $\hat{\mathbf{y}}^{(i)}$ for example $\mathbf{x}^{(\mathbf{i})}$ is perfect if it matches the true label $\mathbf{y}^{(\mathbf{i})}$

$$\hat{\mathbf{y}}^{(\mathbf{i})} = \mathbf{y}^{(\mathbf{i})}$$

- Perfection is hard (at least at first) so we need a measure for "how far off" the prediction is.

- We will call the distance between $\hat{\mathbf{y}}^{(\mathbf{i})}, \mathbf{y}^{(\mathbf{i})}$ the *Loss* (or *Cost*) for example i:

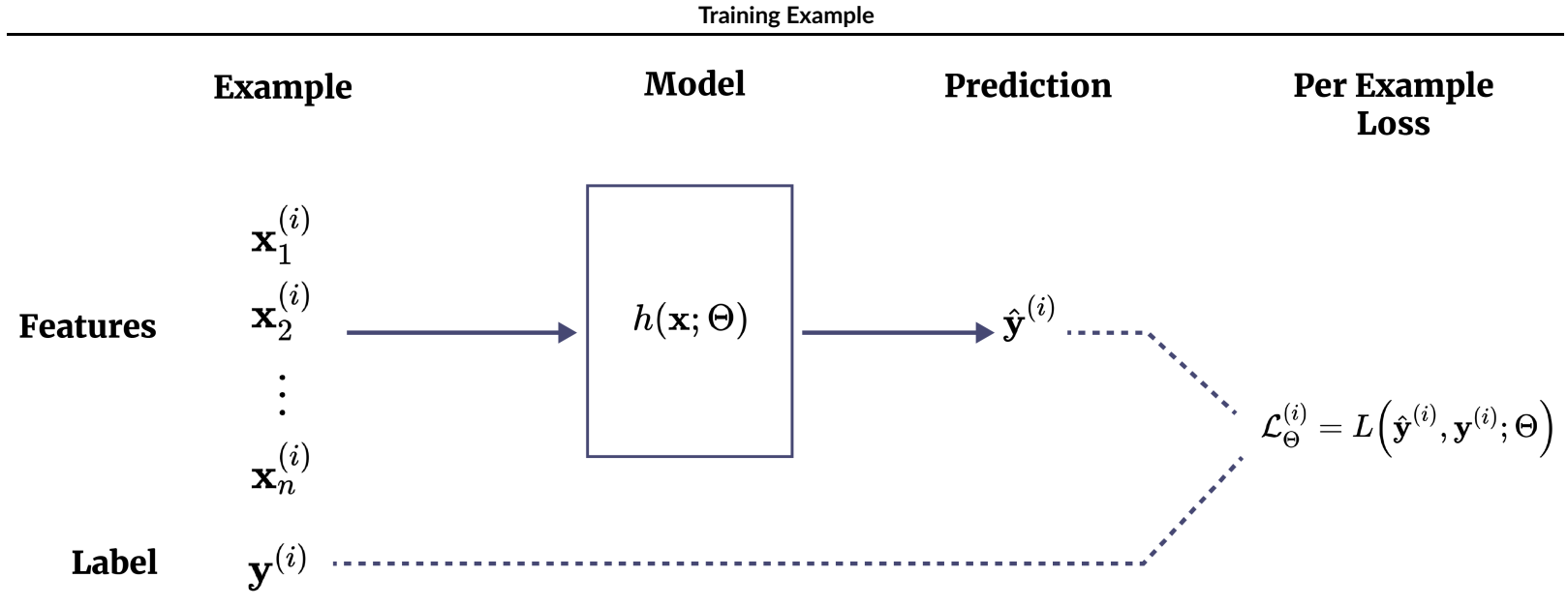
$$\mathcal{L}_{\Theta}^{(\mathbf{i})} = L(\ h(\mathbf{x}^{(\mathbf{i})}; \Theta), \mathbf{y}^{(\mathbf{i})}\ ) = L(\hat{\mathbf{y}}^{(\mathbf{i})}, \mathbf{y}^{(\mathbf{i})})$$

where $L(a, b)$ is a function that is 0 when $a = b$ and increasing as a increasingly differs from b.

Two common forms of L are Mean Squared Error (for Regression) and Cross Entropy Loss (for classification).

The Loss for the entire training set is simply the average (across examples) of the Loss for the example

$$\mathcal{L}_{\Theta} = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}_{\Theta}^{(\mathbf{i})}$$


Training Example
Example
Model
Prediction
Per Example Loss
Features
$\mathbf{x}_1^{(i)}$
$\mathbf{x}_2^{(i)}$
$\vdots$
$\mathbf{x}_n^{(i)}$
$h(\mathbf{x};\Theta)$
$\hat{\mathbf{y}}^{(i)}$
$\mathcal{L}_\Theta^{(i)} = L\left(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)}; \Theta\right)$
Label
$\mathbf{y}^{(i)}$

Whereas Loss describes how "bad" our prediction is, we sometimes refer to the converse -- how "good" the prediction is.

We call the "goodness" of the prediction the *Utility* U_{Θ}.

So we could state the optimization objective either as "minimize Cost" or "maximize Utility".

By convention, the DL optimization problem is usually framed as one of minimization (of cost or loss) rather than maximization of utility.

Since Cost is inversely related to Utility, you will sometimes see the minimization objective written as "minimize -1 times Utility".

So be forewarned that you will often see Loss function with leading "negation" signs.

Creating Loss functions is a key part of Deep Learning

As you will come to see, particularly for Deep Learning, the essence of many problems is in creating a Loss Function that captures the objective of your problem.

This is far from a trivial part of the process.

Fitting/Training a Model

The best (optimal) Θ is the one that minimizes the Average (across training examples) Loss

$$\Theta^* = \underset{\Theta}{\operatorname{argmin}}\, \mathcal{L}_\Theta$$

- The goal of fitting/training is to solve for the Θ that minimizes the training set loss L_Θ
- The method for finding Θ is called optimization.

The dot product: Template matching

- The "dot product" (special case of inner product) is one function that often appears in template matching
- It measures the similarity of two vectors

$$\mathbf{v} \cdot \mathbf{v}' = \sum_{i=1}^{n} \mathbf{v}_i \mathbf{v}'_i$$

- As a similarity measure (rather than as a distance) high dot product means "more similar".

- There are several intuitions for the dot product
- The dot product is maximized when large (resp., small) values appear in similar positions in both vectors
    - this becomes even more obvious if we 0-center both vectors such that "small" values become negative
    - this looks like the statistical formula for covariance
        - if we normalize both vectors to unit length, then this looks like correlation

We can generalize dot product to higher dimensions

- Compute pair-wise product of corresponding entries
- Reduce to a scalar by summing

In [ ]:
```
print("Done")
```