CMSC 478

KMA Solaiman

Supervised Learning: Linear Regression, Learning Algorithm and Gradient Descent

Slights are slightly adapted from Chris Re´, Stanford ML

Supervised Learning and Linear Regression

- Definitions
- Linear Regression
  - Learning Algorithm
  - Cost / Loss Function
  - Gradient Descent
- Batch and Stochastic Gradient

Supervised Learning

- A **hypothesis** or a prediction function is function $h : \mathcal{X} \to \mathcal{Y}$

Supervised Learning

- A **hypothesis** or a prediction function is function $h : \mathcal{X} \to \mathcal{Y}$
  - $\mathcal{X}$ is an image, and $\mathcal{Y}$ contains "cat" or "not."
  - $\mathcal{X}$ is a text snippet, and $\mathcal{Y}$ contains "hate speech" or "not."
  - $\mathcal{X}$ is house data, and $\mathcal{Y}$ could be the price.

Supervised Learning

- A **hypothesis** or a prediction function is function $h : \mathcal{X} \to \mathcal{Y}$
  - $\mathcal{X}$ is an image, and $\mathcal{Y}$ contains "cat" or "not."
  - $\mathcal{X}$ is a text snippet, and $\mathcal{Y}$ contains "hate speech" or "not."
  - $\mathcal{X}$ is house data, and $\mathcal{Y}$ could be the price.
- A **training set** is a set of pairs $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$ s.t. $x^{(i)} \in \mathcal{X}$ and $y^{(i)} \in \mathcal{Y}$ for $i = 1, \ldots, n$.

Supervised Learning

- A **hypothesis** or a prediction function is function $h : \mathcal{X} \to \mathcal{Y}$
  - $\mathcal{X}$ is an image, and $\mathcal{Y}$ contains "cat" or "not."
  - $\mathcal{X}$ is a text snippet, and $\mathcal{Y}$ contains "hate speech" or "not."
  - $\mathcal{X}$ is house data, and $\mathcal{Y}$ could be the price.
- A **training set** is a set of pairs $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$ s.t. $x^{(i)} \in \mathcal{X}$ and $y^{(i)} \in \mathcal{Y}$ for $i = 1, \ldots, n$.
- Given a training set our goal is to produce a *good* prediction function $h\,(or\,f\,)$
  - Defining "good" will take us a bit. It's a modeling question!
  - We will want to use h on *new* data not in the training set.

Supervised Learning

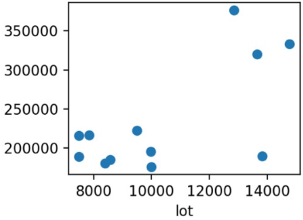
- A **hypothesis** or a prediction function is function $h : \mathcal{X} \to \mathcal{Y}$
  - $\mathcal{X}$ is an image, and $\mathcal{Y}$ contains "cat" or "not."
  - $\mathcal{X}$ is a text snippet, and $\mathcal{Y}$ contains "hate speech" or "not."
  - $\mathcal{X}$ is house data, and $\mathcal{Y}$ could be the price.
- A **training set** is a set of pairs $\{(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})$ s.t. $x^{(i)} \in \mathcal{X}$ and $y^{(i)} \in \mathcal{Y}$ for $i = 1, \dots, n$.
- Given a training set our goal is to produce a *good* prediction function h (or f)
  - Defining "good" will take us a bit. It's a modeling question!
  - We will want to use h on *new* data not in the training set.

- If $\mathcal{Y}$ is continuous, then called a *regression problem*.
- If $\mathcal{Y}$ is discrete, then called a *classification problem*.

Our first example: Regression using Housing Data.

Example Data (Housing Prices from Ames Dataset from Kaggle)

	SalePrice	Lot.Area
4	189900	13830
5	195500	9978
9	189000	7500
10	175900	10000
12	180400	8402
22	216000	7500
36	376162	12858
47	320000	13650
55	216500	7851
56	185088	8577



How do we represent h? (One popular choice)

$h(x) = \theta_0 + \theta_1 x_1$ is an *affine function*

How do we represent h? (One popular choice)

$h(x) = \theta_0 + \theta_1 x_1$ is an *affine function*

	size
$x^{(1)}$	2104
$x^{(2)}$	2500

	Price
$y^{(1)}$	400
$y^{(2)}$	900

How do we represent h? (One popular choice)

$h(x) = \theta_0 + \theta_1 x_1$ is an *affine function*

	size
$x^{(1)}$	2104
$x^{(2)}$	2500

	Price
$y^{(1)}$	400
$y^{(2)}$	900

An example prediction?

How do we represent h? (One popular choice)

$h(x) = \theta_0 + \theta_1 x_1$ is an *affine function*

	size
$x^{(1)}$	2104
$x^{(2)}$	2500

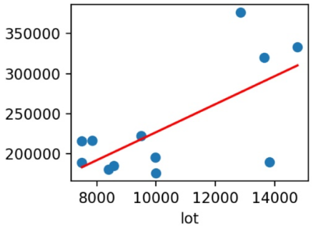
	Price
$y^{(1)}$	400
$y^{(2)}$	900

An example prediction?

Notice the prediction is defined by the *parameters* θ_0 and θ_1. This is a huge reduction in the space of functions!

Simple Line Fit

	SalePrice	Lot.Area
4	189900	13830
5	195500	9978
9	189000	7500
10	175900	10000
12	180400	8402
22	216000	7500
36	376162	12858
47	320000	13650
55	216500	7851
56	185088	8577
58	222500	9505



Slightly More Interesting Data

We add *features* (bedrooms and lot size) to incorporate more information about houses.

	size	bedrooms	lot size
$x^{(1)}$	2104	4	45k
$x^{(2)}$	2500	3	30k

	Price
$y^{(1)}$	400
$y^{(2)}$	900

Slightly More Interesting Data

We add *features* (bedrooms and lot size) to incorporate more information about houses.

	size	bedrooms	lot size
$x^{(1)}$	2104	4	45k
$x^{(2)}$	2500	3	30k

	Price
$y^{(1)}$	400
$y^{(2)}$	900

What's a prediction here?

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3.$$

Slightly More Interesting Data

We add *features* (bedrooms and lot size) to incorporate more information about houses.

	size	bedrooms	lot size		Price
$x^{(1)}$	2104	4	45k	$y^{(1)}$	400
$x^{(2)}$	2500	3	30k	$y^{(2)}$	900

What's a prediction here?

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3.$$

With the convention that $x_0 = 1$ we can write:

$$h(x) = \sum_{j=0}^{3} \theta_j x_j$$

Vector Notation for Prediction

	size	bedrooms	lot size
$x^{(1)}$	2104	4	45k
$x^{(2)}$	2500	3	30k

	Price
$y^{(1)}$	400
$y^{(2)}$	900

We write the vectors as (important notation)

$$\theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix} \text{ and } x^{(1)} = \begin{pmatrix} x_0^{(1)} \\ x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{pmatrix} = \begin{pmatrix} 1 \\ 2104 \\ 4 \\ 45 \end{pmatrix} \text{ and } y^{(1)} = 400$$

Vector Notation for Prediction

	size	bedrooms	lot size
$x^{(1)}$	2104	4	45k
$x^{(2)}$	2500	3	30k

	Price
$y^{(1)}$	400
$y^{(2)}$	900

We write the vectors as (important notation)

$$\theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix} \text{ and } x^{(1)} = \begin{pmatrix} x_0^{(1)} \\ x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{pmatrix} = \begin{pmatrix} 1 \\ 2104 \\ 4 \\ 45 \end{pmatrix} \text{ and } y^{(1)} = 400$$

We call θ (or w) **parameters**, $x^{(i)}$ is the input or the **features**, and the output or **target** is $y^{(i)}$. To be clear,

(x, y) is a training example and $(x^{(i)}, y^{(i)})$ is the i^{th} example.

Vector Notation for Prediction

	size	bedrooms	lot size		Price
$x^{(1)}$	2104	4	45k	$y^{(1)}$	400
$x^{(2)}$	2500	3	30k	$y^{(2)}$	900

We write the vectors as (important notation)

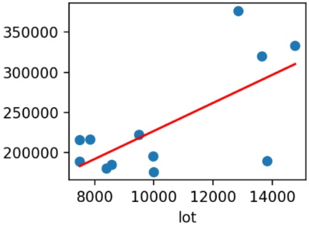
$$\theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix} \text{ and } x^{(1)} = \begin{pmatrix} x_0^{(1)} \\ x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{pmatrix} = \begin{pmatrix} 1 \\ 2104 \\ 4 \\ 45 \end{pmatrix} \text{ and } y^{(1)} = 400$$

We call θ *(or w)* **parameters**, $x^{(i)}$ is the input or the **features**, and the output or **target** is $y^{(i)}$. To be clear,

(x, y) is a training example and $(x^{(i)}, y^{(i)})$ is the i^{th} example.

We have n examples (i.e., $i = 1, \ldots, n$). There are d features so $x^{(i)}$ and θ are $d + 1$ dimensional (since $x_0 = 1$)

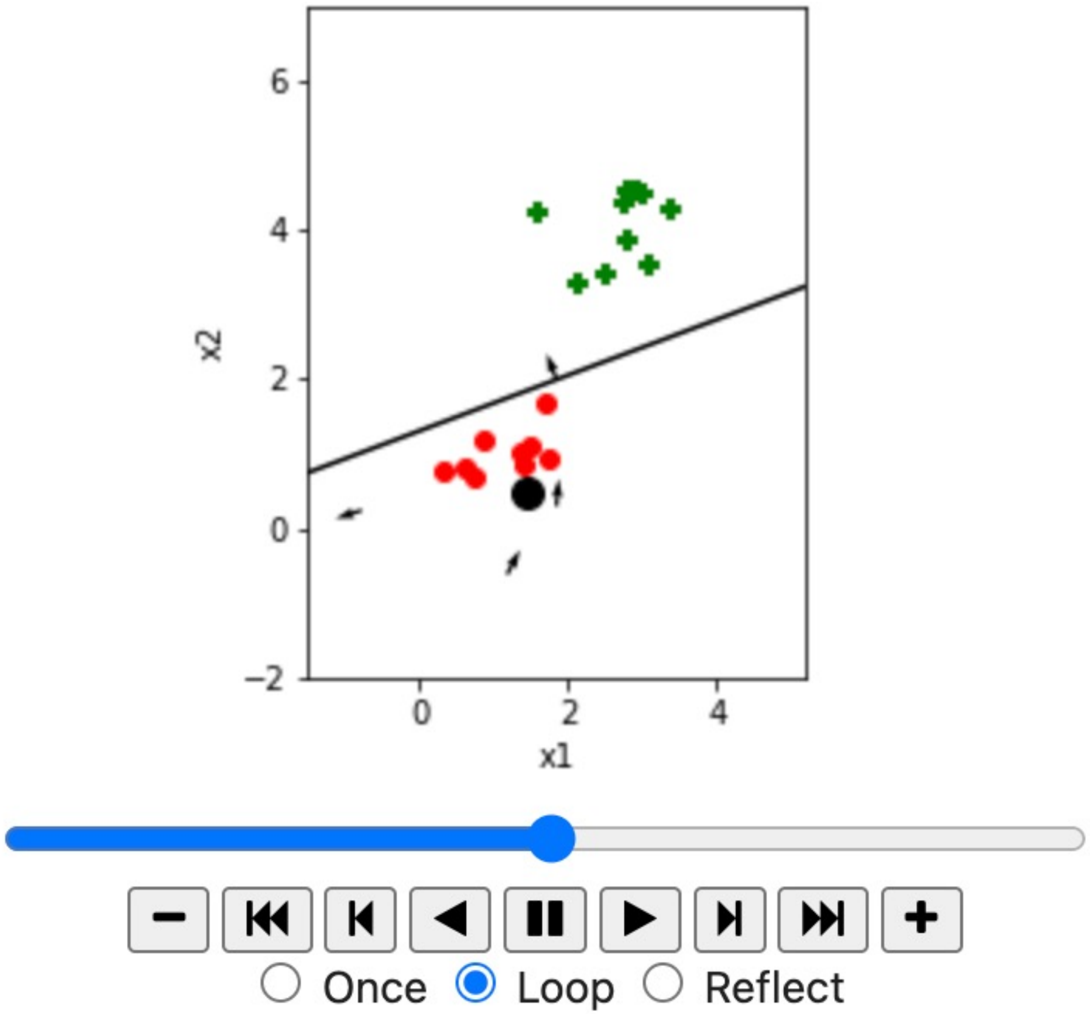
Visual version of linear regression



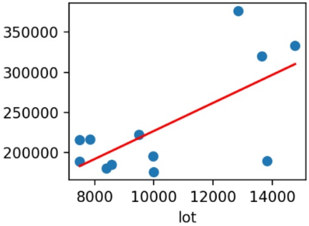
Let $h_\theta(x) = \sum_{j=0}^{d} \theta_j x_j$ want to choose θ so that $h_\theta(x) \approx y$.

Fitting a good line

Animation



Visual version of linear regression: Learning



Let $h_\theta(x) = \sum_{j=0}^{d} \theta_j x_j$ want to choose θ so that $h_\theta(x) \approx y$. One popular idea called **least squares**

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

Choose

$$\theta = \operatorname*{argmin}_{\theta} J(\theta).$$

Linear Regression Summary

- We saw our first hypothesis class *affine* or *linear* functions.
- We refreshed ourselves on notation and introduced terminology like **parameters**, **features**, etc.
- We saw this paradigm that a "good" hypothesis is some how one that *is close to* the data (objective function J).

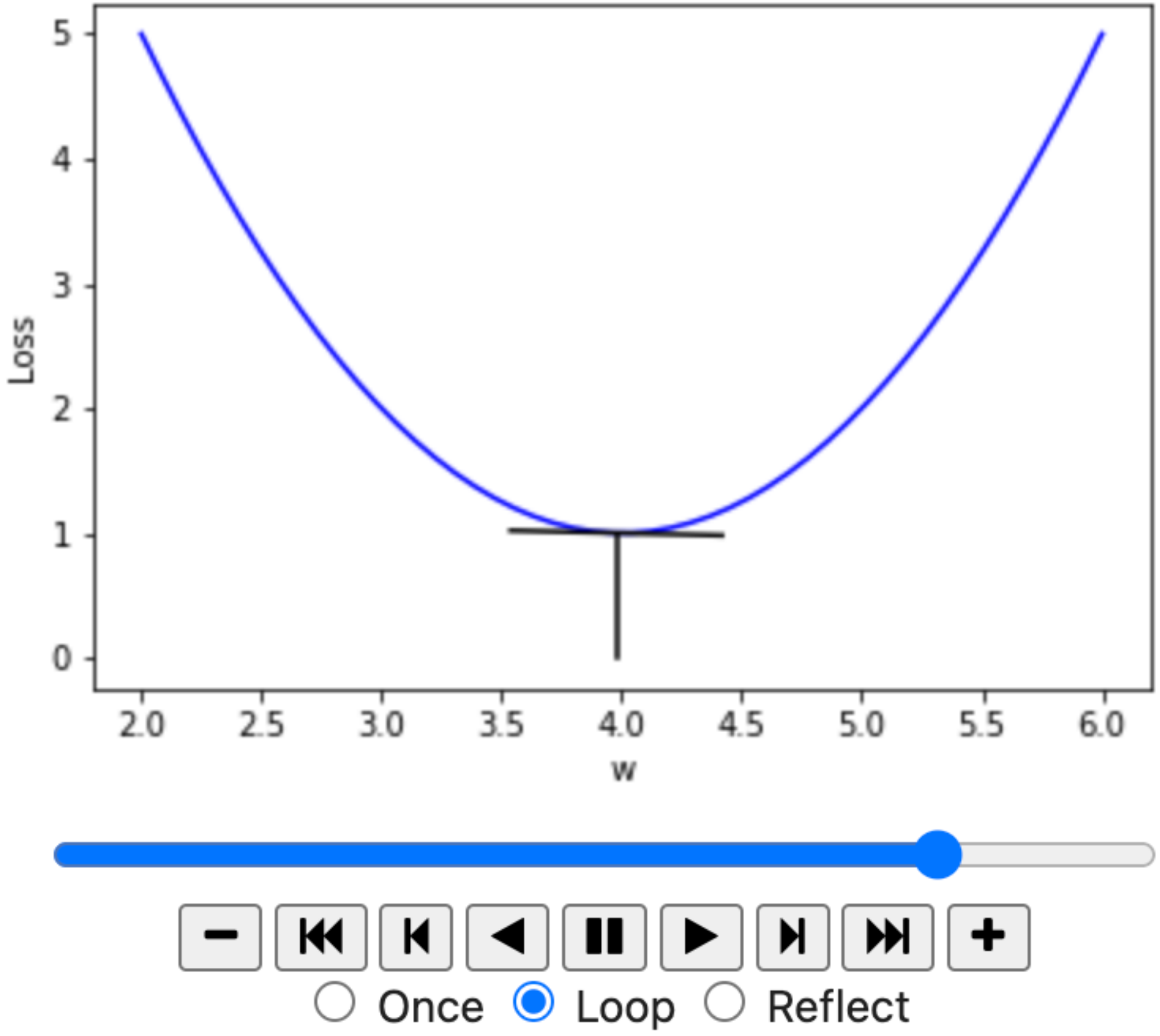
Linear Regression Summary

- We saw our first hypothesis class *affine* or *linear* functions.
- We refreshed ourselves on notation and introduced terminology like **parameters**, **features**, etc.
- We saw this paradigm that a "good" hypothesis is some how one that *is close to* the data (objective function J).
- Next, we'll see how to solve these equations.

Solving the least squares optimization problem.

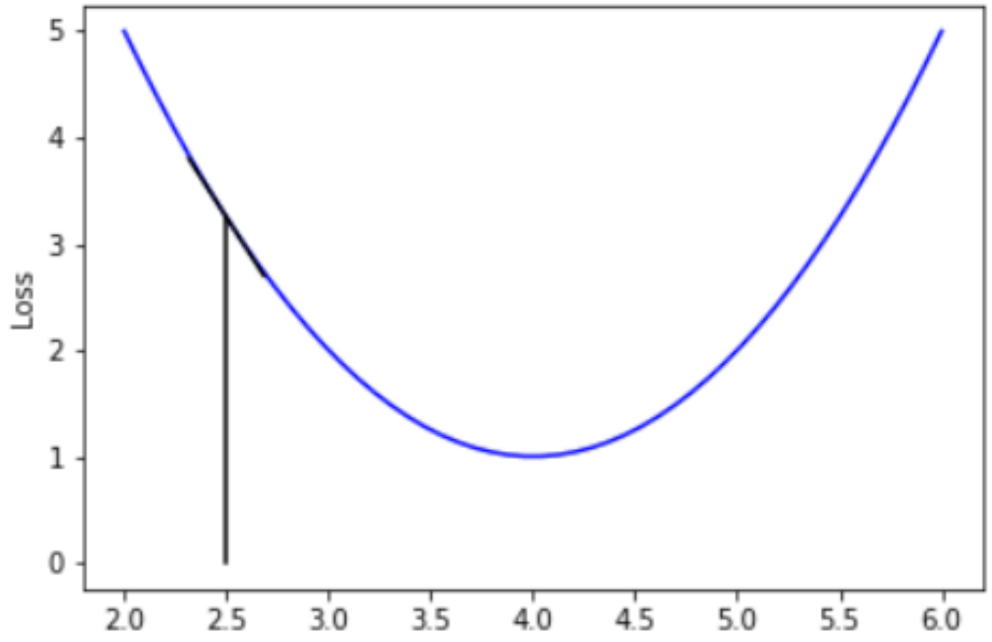
Gradient Descent

Animation



Gradient Descent

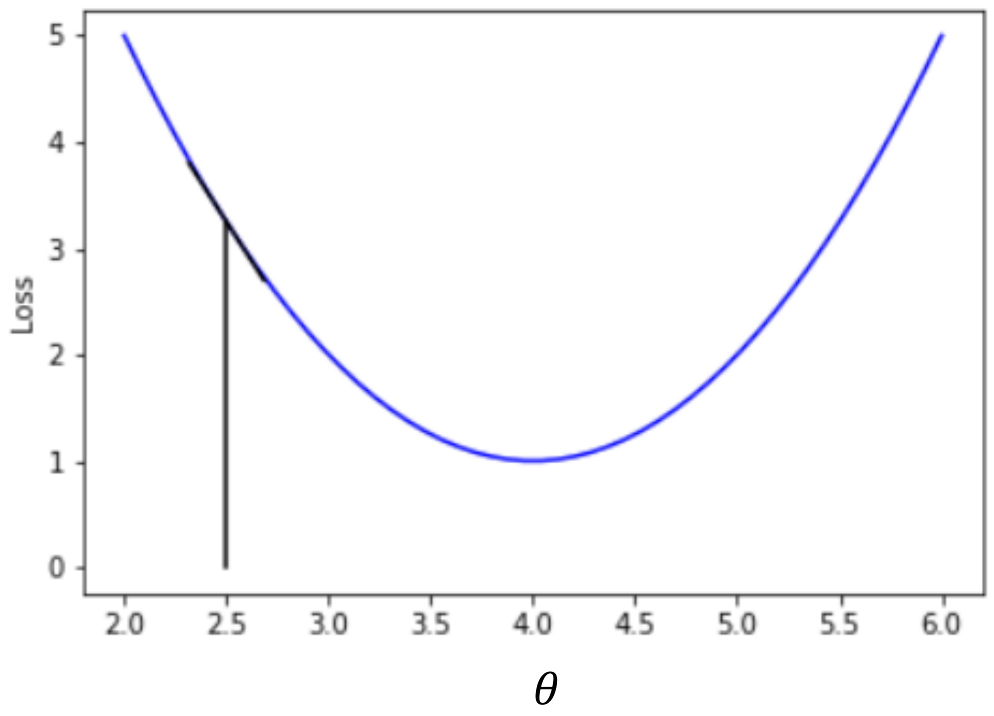
- $\mathcal{J}(\theta) = (\theta - 4)^2 + 1$
- Find the weight (value of θ) that minimizes the loss $\mathcal{J}$
- $\mathcal{J}'(\theta) = ?$
- θ = 2.5
- given the current value of w, adjusting θ by an amount that has the negative of the sign of $\mathcal{J}'(\theta)$ leads to a smaller value of $\mathcal{J}$.



θ

Gradient Descent

- $\mathcal{J}(\theta) = (\theta - 4)^2 + 1$
- Find the weight (value of θ) that minimizes the loss J
- $\mathcal{J}'(\theta) = ?$
- θ = 2.5
- given the current value of w, adjusting θ by an amount that has the negative of the sign of $\mathcal{J}'(\theta)$ leads to a smaller value of $\mathcal{J}$.



$\theta = \theta - \alpha * \mathcal{J}'(\theta)$

Gradient Descent

$$\begin{aligned}\theta^{(0)} &= 0 \\ \theta_j^{(t+1)} &= \theta_j^{(t)} - \alpha \frac{\partial}{\partial \theta_j} J(\theta^{(t)}) \qquad \text{for } j = 0, \ldots, d.\end{aligned}$$

Gradient Descent Computation

$$\theta_j^{(t+1)} = \theta_j^{(t)} - \alpha \frac{\partial}{\partial \theta_j} J(\theta^{(t)}) \text{ for } j = 0, \ldots, d.$$

Note that α is called the **learning rate** or **step size**.

Let's compute the derivatives...

$$\begin{aligned} \frac{\partial}{\partial \theta_j} J(\theta^{(t)}) &= \sum_{i=1}^{n} \frac{1}{2} \frac{\partial}{\partial \theta_j} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \\ &= \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \frac{\partial}{\partial \theta_j} h_\theta(x^{(i)}) \end{aligned}$$

Gradient Descent Computation

$$\theta_j^{(t+1)} = \theta_j^{(t)} - \alpha \frac{\partial}{\partial \theta_j} J(\theta^{(t)}) \text{ for } j = 0, \ldots, d.$$

Note that α is called the **learning rate** or **step size**.

Let's compute the derivatives...

$$\begin{aligned} \frac{\partial}{\partial \theta_j} J(\theta^{(t)}) &= \sum_{i=1}^{n} \frac{1}{2} \frac{\partial}{\partial \theta_j} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \\ &= \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \frac{\partial}{\partial \theta_j} h_\theta(x^{(i)}) \end{aligned}$$

For our *particular* h_θ we have:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_d x_d \text{ so } \frac{\partial}{\partial \theta_j} h_\theta(x) = x_j$$

Gradient Descent Computation

Thus, our update rule for component j can be written:

$$\theta_j^{(t+1)} = \theta_j^{(t)} - \alpha \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}.$$

Gradient Descent Computation

Thus, our update rule for component j can be written:

$$\theta_j^{(t+1)} = \theta_j^{(t)} - \alpha \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}.$$

We write this in *vector notation* for $j = 0, \ldots, d$ as:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}.$$

Saves us a lot of writing! And easier to understand ... eventually.

Linear Classification: Mushroom and Goats

	color	width	height	label
0	-0.311688	0.358501	0.936567	edible
1	-0.472327	0.817906	0.468387	poisonous

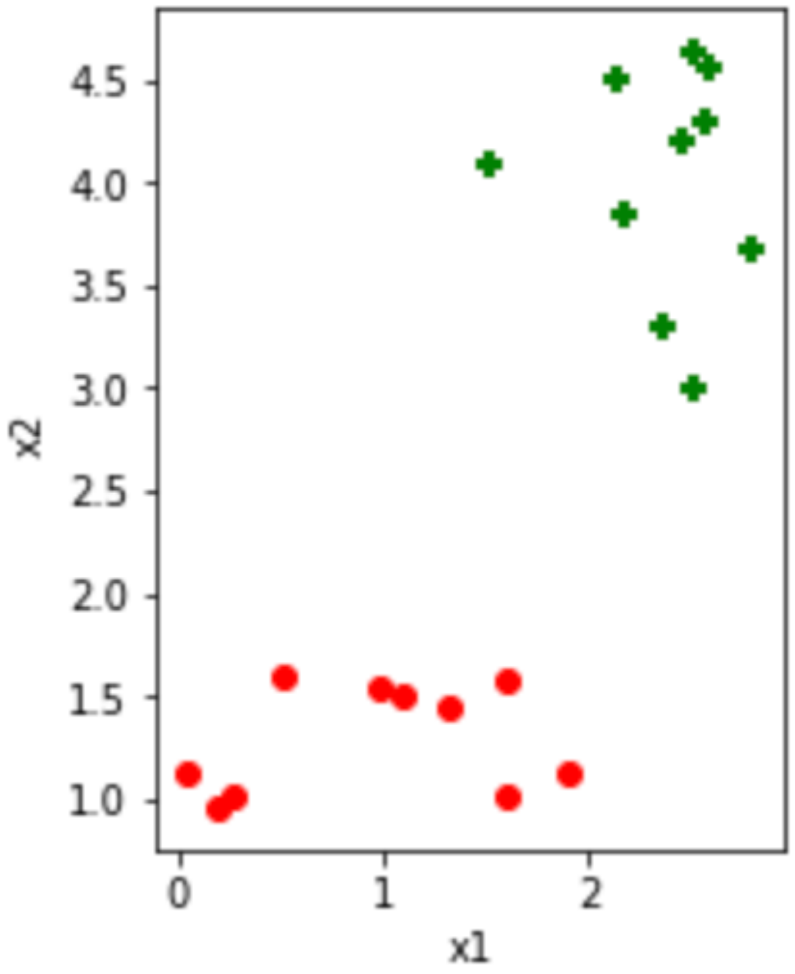
$$\mathbf{sign}(w_c * \mathbf{color} + w_w * \mathbf{width} + w_h * \mathbf{height})$$

$$\mathbf{sign}(0 * -0.472327 + 1 * 0.817906 - 1 * 0.468387) = \mathbf{sign}(0.349519) = +1$$

$$\mathbf{sign}(0 * -0.311688 + 1 * 0.358501 - 1 * 0.936567) = \mathbf{sign}(-0.578066) = -1$$

Linear Classification

	x1	x2	y
0	0.048589	1.120275	-1
1	0.200023	0.956716	-1
2	1.595538	1.023582	-1
3	1.315929	1.452371	-1
4	1.087080	1.513219	-1
5	0.512235	1.594651	-1
6	0.265039	1.008506	-1
7	1.606480	1.571889	-1
8	0.977585	1.550227	-1
9	1.908708	1.121259	-1
10	2.503476	3.002576	1



Loss Function for Classification: 0-1 Loss

L_{0-1}	$\hat{y} = -1$	$\hat{y} = 1$
$y = -1$	0	1
$y = 1$	1	0

Loss Function for Classification: 0-1 Loss

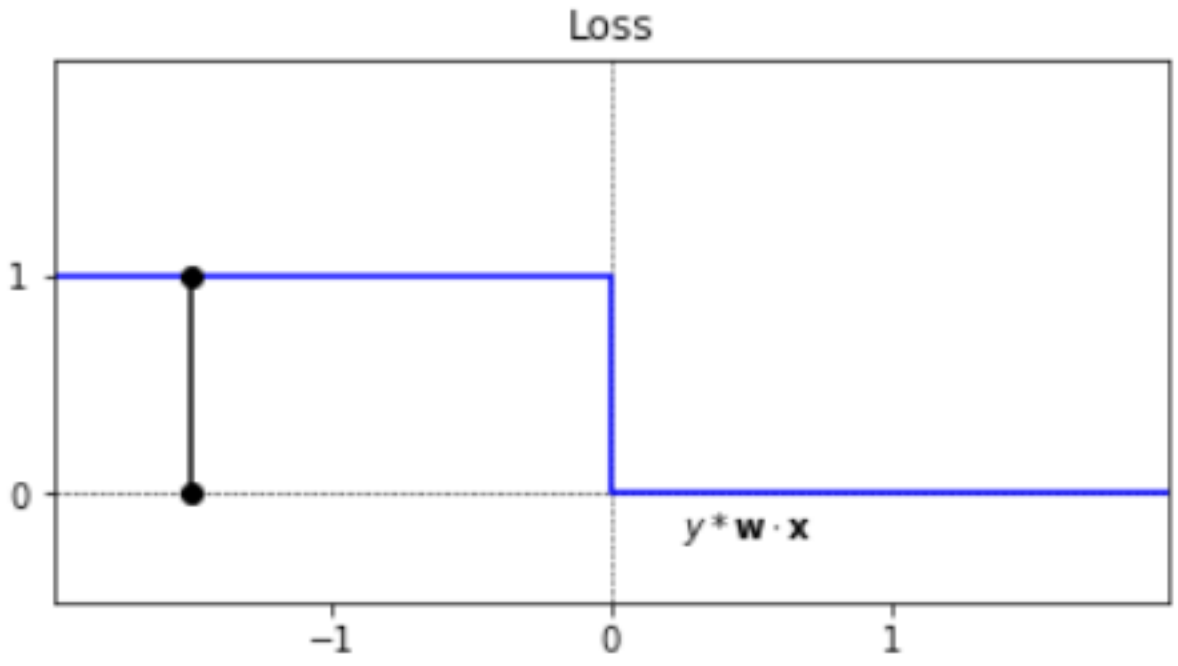
L_{0-1}	$\hat{y} = -1$	$\hat{y} = 1$
$y = -1$	0	1
$y = 1$	1	0

$$L_{0-1}(y, \mathbf{w} \cdot \mathbf{x}) = \begin{cases} 0 & \text{if } y * \mathbf{w} \cdot \mathbf{x} > 0 \\ 1 & \text{otherwise} \end{cases}$$

Loss Function for Classification: 0-1 Loss

L_{0-1}	$\hat{y} = -1$	$\hat{y} = 1$
$y = -1$	0	1
$y = 1$	1	0

$$L_{0-1}(y, \mathbf{w} \cdot \mathbf{x}) = \begin{cases} 0 & \text{if } y * \mathbf{w} \cdot \mathbf{x} > 0 \\ 1 & \text{otherwise} \end{cases}$$



Batch Versus Stochastic Minibatch: Motivation

Consider our update rule:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}.$$

- A single update, our rule examines *all* n data points.

Batch Versus Stochastic Minibatch: Motivation

Consider our update rule:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}.$$

- A single update, our rule examines *all* n data points.
- In some modern applications (more later) n may be in the billions or trillions!
  - E.g., we try to "predict" every word on the web.

Batch Versus Stochastic Minibatch: Motivation

Consider our update rule:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}.$$

- A single update, our rule examines *all* n data points.
- In some modern applications (more later) n may be in the billions or trillions!
  - E.g., we try to "predict" every word on the web.
- **Idea** Sample a few points (maybe even just one!) to *approximate* the gradient called **Stochastic Gradient** (SGD).
  - SGD is the workhorse of modern ML, e.g., pytorch and tensorflow.

Stochastic Minibatch

- We randomly select a **batch** of $B \subseteq \{1, \ldots, n\}$ where $|B| < n$.
- We approximate the gradient using just those B points as follows (vs. gradient descent)

$$\frac{1}{|B|} \sum_{j \in B} \left(h_\theta(x^{(j)}) - y^{(j)} \right) x^{(j)} \text{ v.s. } \frac{1}{n} \sum_{j=1}^{n} \left(h_\theta(x^{(j)}) - y^{(j)} \right) x^{(j)}.$$

Stochastic Minibatch

- We randomly select a **batch** of $B \subseteq \{1, \ldots, n\}$ where $|B| < n$.
- We approximate the gradient using just those B points as follows (vs. gradient descent)

$$\frac{1}{|B|} \sum_{j \in B} \left(h_\theta(x^{(j)}) - y^{(j)} \right) x^{(j)} \text{ v.s. } \frac{1}{n} \sum_{j=1}^{n} \left(h_\theta(x^{(j)}) - y^{(j)} \right) x^{(j)}.$$

- So our update rule for SGD is:

All minibatches are used for each iteration, or epoch and then start the next one

$$\theta^{(t+1)} = \theta^{(t)} - \alpha_B \sum_{j \in B} \left(h_\theta(x^{(j)}) - y^{(j)} \right) x^{(j)}.$$

- NB: scaling of $|B|$ versus n is "hidden" inside choice of α_B.

Stochastic Minibatch vs. Gradient Descent

- Recall our rule B points as follows:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha_B \sum_{j \in B} \left(h_\theta(x^{(j)}) - y^{(j)} \right) x^{(j)}.$$

- If $|B| = \{1, \ldots, n\}$ (the whole set), then they coincide.
- Smaller B implies a lower quality approximation of the gradient (higher variance).
- Nevertheless, it may actually converge faster! (Case where the dataset has many copies of the same point–extreme, but lots of redundancy)

Stochastic Minibatch vs. Gradient Descent

- Recall our rule B points as follows:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha_B \sum_{j \in B} \left(h_\theta(x^{(j)}) - y^{(j)} \right) x^{(j)}.$$

- If $|B| = \{1, \ldots, n\}$ (the whole set), then they coincide.
- Smaller B implies a lower quality approximation of the gradient (higher variance).
- Nevertheless, it may actually converge faster! (Case where the dataset has many copies of the same point–extreme, but lots of redundancy)
- In practice, choose B proportional to what works well on modern parallel hardware (GPUs).

Summary of this Subsection of Optimization

- Our goal was to optimize a loss function to find a good predictor.
- We learned about gradient descent and the workhorse algorithm for ML, **Stochastic Gradient Descent** (SGD).
- We touched on the tradeoffs of choosing the right batch size.

Summary from Today

- We saw a *lot of notation*
- We learned about linear regression: the model, how to solve, and more.
- We learned the workhorse algorithm for ML called **SGD**.
- Next time: Classification!