Lecture 15: LMS Regression and Gradient Descent

COMP 343, Spring 2022

Victoria Manfredi

WESLEYAN UNIVERSITY

Acknowledgements: These slides are based primarily on material from the book Machine Learning by Tom Mitchell (and associated slides), the book Machine Learning, An Applied Mathematics Introduction by Paul Wilmott, slides created by Vivek Srikumar (Utah), Dan Roth (Penn), Julia Hockenmaier (Illinois Urbana-Champaign), Jessica Wu (Harvey Mudd) and C. David Page (U of Wisconsin-Madison)

Today's Topics

Midterm

- Wednesday, March 30

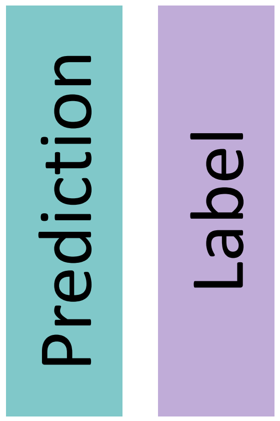
Linear Regression

- Gradient descent
- Example
- Convergence
- Stochastic gradient descent
- Regularization

Recap

Evaluation metrics (for classification)

To evaluate model, compare predicted labels to actual



Accuracy: proportion of examples where we predicted correct label

$$\text{accuracy} = \frac{\text{\# of correct predictions}}{\text{\# of examples}}$$

Error: proportion of examples where we predicted incorrect label

$$\text{error} = 1 - \text{accuracy}$$

$$\text{accuracy} = \frac{\text{\# of incorrect predictions}}{\text{\# of examples}}$$

Precision-Recall analysis

What fraction of class “label” examples did the classifier discover?

$$\text{Recall(label)} = \frac{\text{Correct predictions(label)}}{\text{Correct predictions(label)} + \text{Missed examples(label)}}$$

What fraction of classifier’s predictions of class “label” were correct

$$\text{Precision(label)} = \frac{\text{Correct predictions(label)}}{\text{Correct predictions(label)} + \text{Incorrect predictions(label)}}$$

By default, precision and recall computed for the positive label, as that is usually the case of interest and the one usually with fewer example (e.g., diagnosing diseases in patients, identifying spam emails)

Combining into one number

Sometimes easier to work with a single number as performance measure

F1 score balances precision and recall: harmonic mean of precision and recall

$$f_1 = \frac{2pr}{p + r}$$

Training to minimize F1 is difficult, but can choose hyper parameters for which F1 is maximized

Linear regression

Inputs are feature vectors: $\mathbf{x} \in \Re^d$

Outputs are real numbers: $y \in \Re$

For simplicity, we will assume that the first feature is always 1, to make notation easier

$$\mathbf{x}_i = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$$

We have a training data set:

$$D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_d, y_d)\}$$

We want to approximate y as

$$y = w_1 + w_2 x_2 + \cdots + w_d x_d$$
$$y = \mathbf{w}^T \mathbf{x}$$

$\mathbf{w}$ is the learned weight vector in $\Re^d$

Making assumption that output y is a linear function of the features $\mathbf{x}$

Goal: use the training data to find the ***best*** possible value of $\mathbf{w}$

If our hypothesis space is linear functions …

How do we know which weight vector is best one for a training set?

For an input $(\mathbf{x}_i, y_i)$ in the training set, the *cost* of a mistake is

$$|\underset{\text{True output}}{\underbrace{y_i}} - \underset{\text{Predicted output}}{\underbrace{\mathbf{w}^T\mathbf{x}_i}}|$$

How far apart is true from predicted in absolute sense?
If very different then weight vector is probably not very good

$$|y_i - w_1^T x_i| = 60000 \qquad |y_{i+1} - w_1^T x_{i+1}| = 0.1$$
$$|y_i - w_2^T x_i| = 0.1 \qquad |y_{i+1} - w_2^T x_{i+1}| = 0.3$$

But could also be that weight vector is just bad for that example

How do we decide whether weight vector is good?

How do we know which weight vector is best one for a training set?

For an input $(\mathbf{x}_i, y_i)$ in the training set, the *cost* of a mistake is

$$|y_i - \mathbf{w}^T\mathbf{x}_i| \longrightarrow$$

This tells us how good for one example

Define the cost (or *loss*) for a particular weight vector $\mathbf{w}$ to be

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(\underline{y_i - \mathbf{w}^T\mathbf{x}_i})^2$$

This tells us how good for m examples

Squared error is a popular loss function: *sum of squared costs over the training set.* Dividing by 2 rather than m will make our math work out nicely later

How do we decide whether weight vector is good?

How do we know which weight vector is best one for a training set?

For an input $(\mathbf{x}_i, y_i)$ in the training set, the *cost* of a mistake is

$$|y_i - \mathbf{w}^T\mathbf{x}_i|$$

Define the cost (or *loss*) for a particular weight vector $\mathbf{w}$ to be

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

Function of functions

J is a function that evaluates how good other functions or regressors are, e.g., $\mathbf{w}^T\mathbf{x}$. Every choice of $\mathbf{w}$ gives a different regressor. So J evaluates how good a regressor is.

$$J(\mathbf{f}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - f(\mathbf{x}_i))^2$$

How do we decide whether weight vector is good?

How do we know which weight vector is best one for a training set?

For an input $(\mathbf{x}_i, y_i)$ in the training set, the *cost* of a mistake is

$$|y_i - \mathbf{w}^T\mathbf{x}_i|$$

Define the cost (or *loss*) for a particular weight vector $\mathbf{w}$ to be

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

One strategy for learning: *Find the $\mathbf{w}$ with least cost on this data*

This is called Least Mean Squares (LMS) Regression

$$\min_{\mathbf{w}} J(\mathbf{w}) = \min_{\mathbf{w}} \frac{1}{2} \sum_{i=1}^{m} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

Goal of learning: minimize mean squared error

- This is just the training objective: you can use different learning algorithms to minimize this objective
- Properties of $J(\mathbf{w})$: differentiable and convex. Lower values mean better weight vector $\mathbf{w}$, i.e., regressor.
- Mathematical optimization: focuses on solving problems of the form $\min_{\mathbf{w}} J(\mathbf{w})$. So many algorithms exist to solve problem

This is called Least Mean Squares (LMS) Regression

$$\min_{\mathbf{w}} J(\mathbf{w}) = \min_{\mathbf{w}} \frac{1}{2} \sum_{i=1}^{m} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

Goal of learning: minimize mean squared error

Different strategies exist for learning by optimization

- **Gradient descent**: is a popular algorithm
- **Matrix inversion**: for this particular minimization objective, there is also an analytical solution; no need for gradient descent: $b = (X^T X)^{-1} X^T Y$

Linear Regression

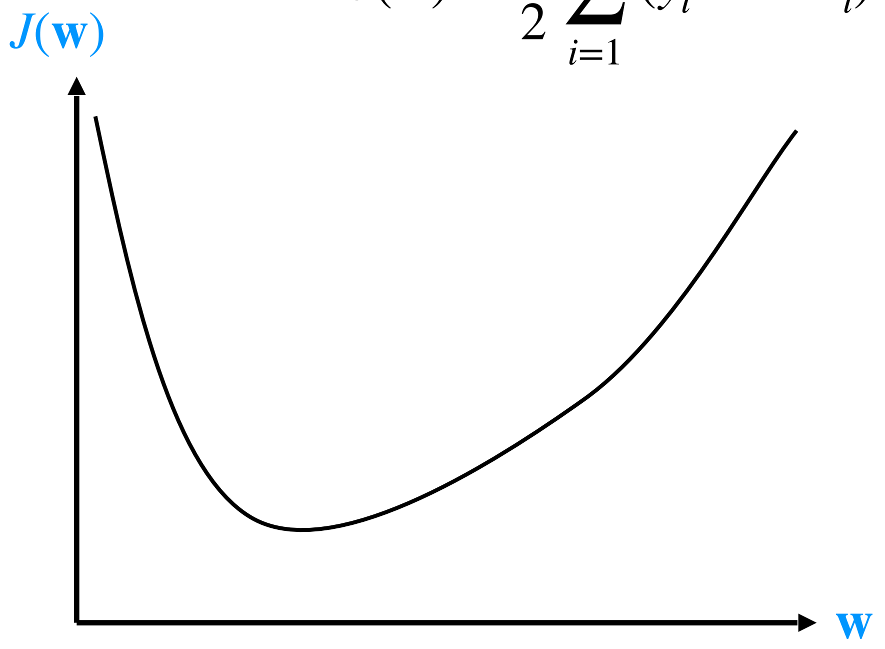
GRADIENT DESCENT

Gradient descent

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

General strategy for minimizing a function $J(\mathbf{w})$



Intuition: The gradient is the direction of steepest increase in the function. To get to the minimum, go in the opposite direction

What is gradient of a function?

In 2-dimensions: slope of a line

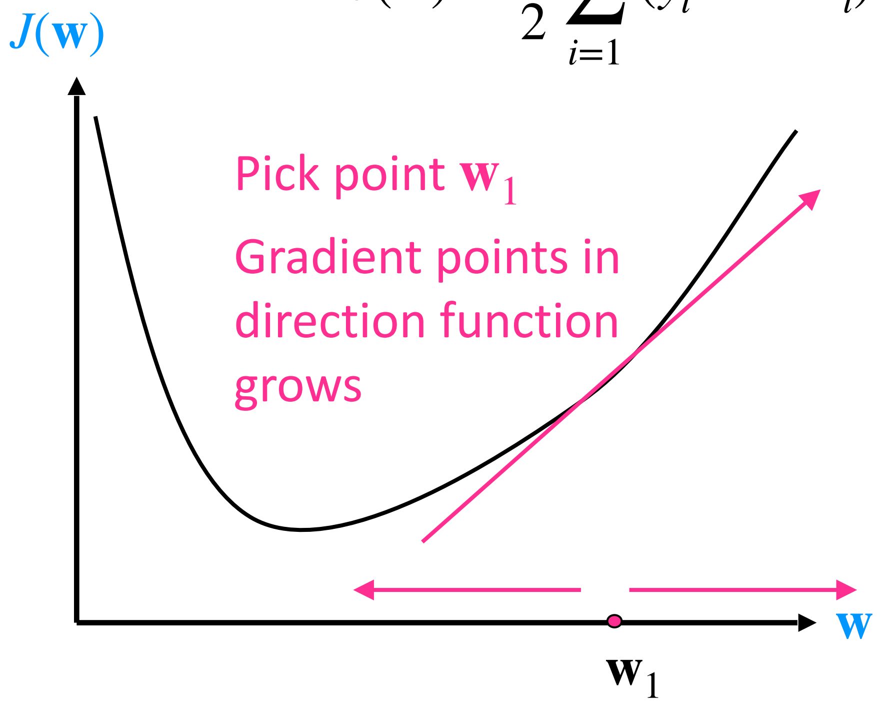
In higher dimensions: direction of steepest ascent, that is, direction in which function grows the fastest

Gradient descent

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

General strategy for minimizing a function $J(\mathbf{w})$



$J(\mathbf{w})$
Pick point $\mathbf{w}_1$
Gradient points in direction function grows
$\mathbf{w}$
$\mathbf{w}_1$

Intuition: The gradient is the direction of steepest increase in the function. To get to the minimum, go in the opposite direction

Gradient descent

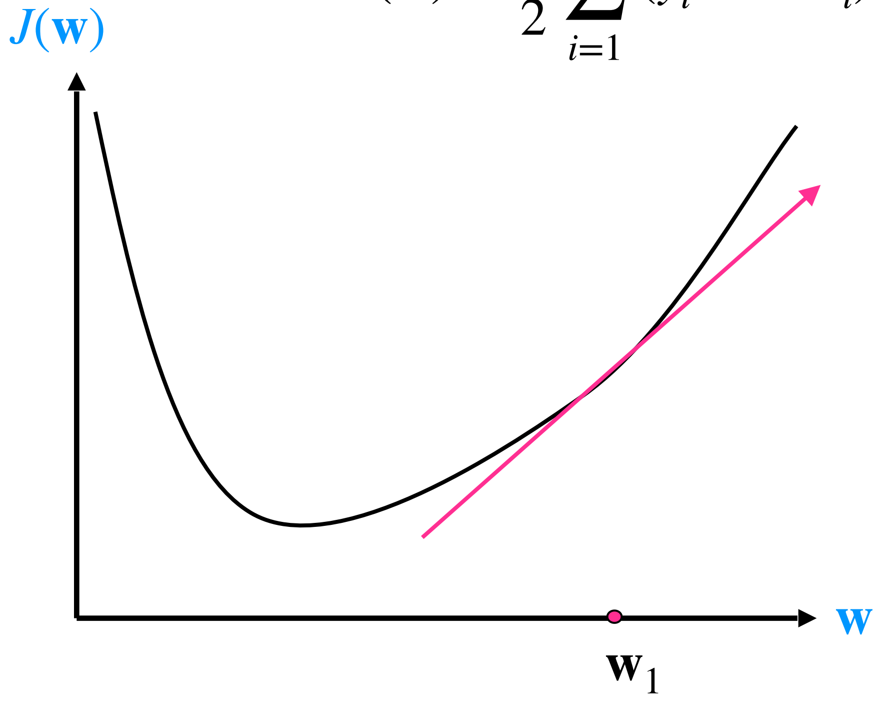
We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

General strategy for minimizing a function $J(\mathbf{w})$

1. Start with an initial guess for $\mathbf{w}$, say $\mathbf{w}_0$

Gradient descent: initialize your starting point for search for minimum anywhere



Intuition: The gradient is the direction of steepest increase in the function. To get to the minimum, go in the opposite direction

Gradient descent

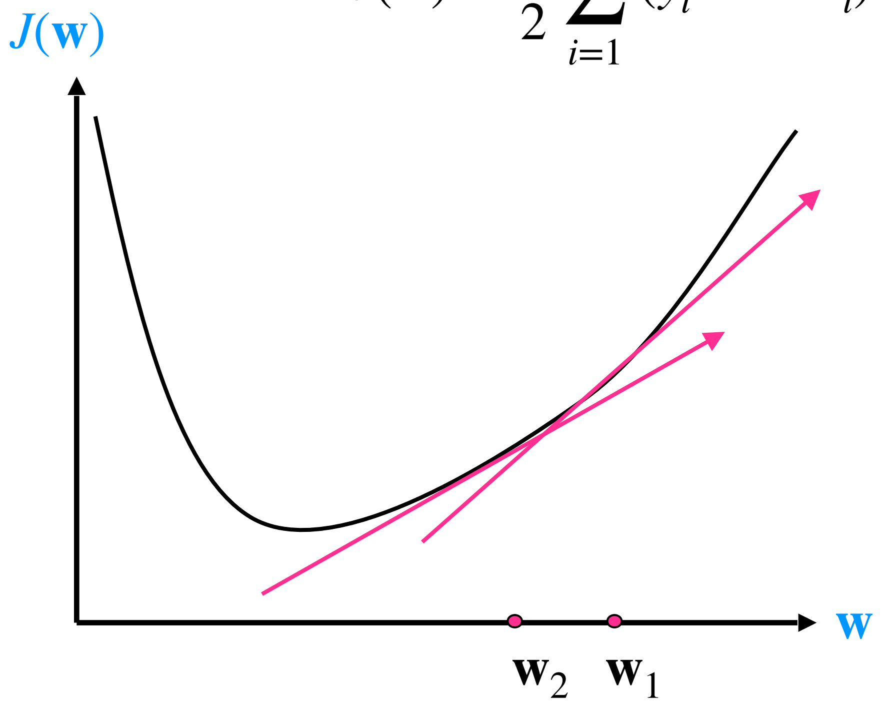
We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

General strategy for minimizing a function $J(\mathbf{w})$

1. Start with an initial guess for $\mathbf{w}$, say $\mathbf{w}_0$

2. Iterate until convergence:
   - Compute the gradient of J at $\mathbf{w}_t$
   - Update $\mathbf{w}_t$ to get $\mathbf{w}_{t+1}$ by taking a step in the opposite direction of the gradient



Intuition: The gradient is the direction of steepest increase in the function. To get to the minimum, go in the opposite direction

Then at every point, compute the gradient (the arrow), and take a step in direction away from gradient (i.e., move to a point where value of function is lower)

Gradient descent

We are trying to minimize

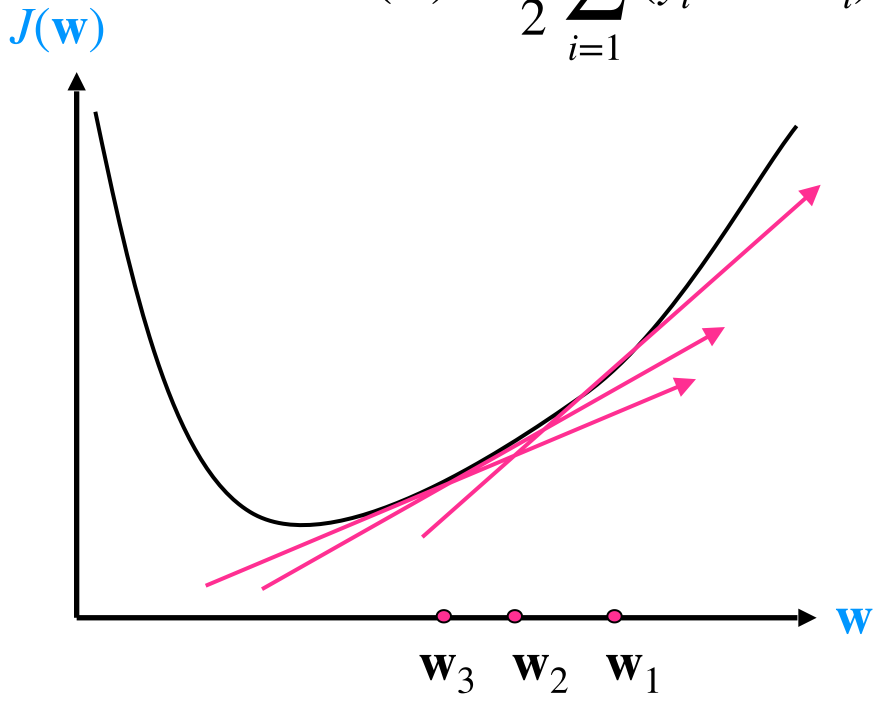
$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

General strategy for minimizing a function $J(\mathbf{w})$

1. Start with an initial guess for $\mathbf{w}$, say $\mathbf{w}_0$

2. Iterate until convergence:
   - Compute the gradient of J at $\mathbf{w}_t$
   - Update $\mathbf{w}_t$ to get $\mathbf{w}_{t+1}$ by taking a step in the opposite direction of the gradient

Keep repeating …



Intuition: The gradient is the direction of steepest increase in the function. To get to the minimum, go in the opposite direction

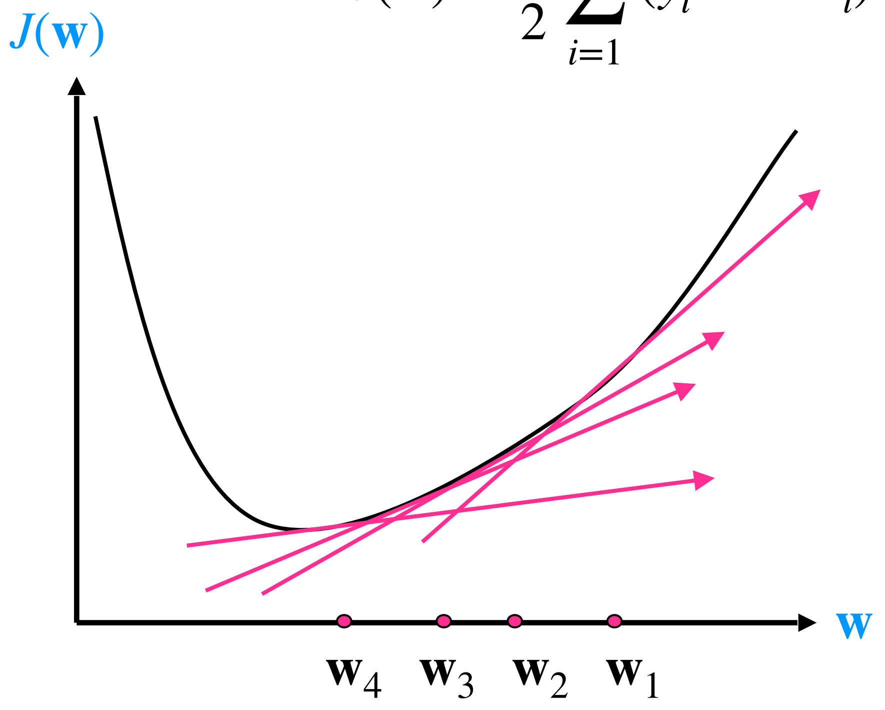
Gradient descent

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

General strategy for minimizing a function $J(\mathbf{w})$

1. Start with an initial guess for $\mathbf{w}$, say $\mathbf{w}_0$
2. Iterate until convergence:
   - Compute the gradient of J at $\mathbf{w}_t$
   - Update $\mathbf{w}_t$ to get $\mathbf{w}_{t+1}$ by taking a step in the opposite direction of the gradient



Intuition: The gradient is the direction of steepest increase in the function. To get to the minimum, go in the opposite direction

And eventually you will get to minimum

Gradient descent for LMS

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

1. Initialize $\mathbf{w}_0$ Initialize to zeroes or random (convex function, so doesn't matter where initialized)

2. For $t = 0,1,2,\ldots$
   - Compute gradient of $J(\mathbf{w})$ at $\mathbf{w}_t$. Call it $\nabla J(\mathbf{w}_t)$

     Grad J or Nabla J

   - Update $\mathbf{w}$ as follows:

     $$\mathbf{w}_{t+1} = \mathbf{w}_t - r\nabla J(\mathbf{w}_t)$$

     Use "-" since step is in opposite direction

   where r is the learning rate (a small constant)

Gradient descent for LMS

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

1. Initialize $\mathbf{w}_0$

2. For $t = 0,1,2,\ldots$

   What is the gradient of J?

   – $\boxed{\text{Compute gradient of } J(\mathbf{w}) \text{ at } \mathbf{w}_t\text{. Call it } \nabla J(\mathbf{w}_t)}$

   – Update $\mathbf{w}$ as follows:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - r\nabla J(\mathbf{w}_t)$$

where r is the learning rate (a small constant)

Gradient of the cost J at point $\mathbf{w}$

Remember that $\mathbf{w}$ is a vector with d elements

$$\mathbf{w} = [w_1, w_2, w_3, \ldots, w_j, \ldots, w_d]$$

J is a function that maps $\mathbf{w}$ to real number (the total cost)

Gradient of the cost J at point $\mathbf{w}$

Remember that $\mathbf{w}$ is a vector with d elements

$$\mathbf{w} = [w_1, w_2, w_3, \ldots, w_j, \ldots, w_d]$$

To find the best direction in the weight space $\mathbf{w}$ we compute the gradient of J with respect to each of the components of

$$\nabla J(\mathbf{w}_t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d} \right]$$

Gradient will be vector with d elements since $\mathbf{w}$ is a vector with d elements

Each element is a partial derivative

Need to compute every element of $\nabla J(\mathbf{w}_t)$ to define gradient

Gradient of the cost J at point $\mathbf{w}$

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

The gradient is of the form $\nabla J(\mathbf{w}_t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$

$$\frac{\partial J}{\partial w_j} = \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

Let's compute gradient for jth weight

Gradient of the cost J at point $\mathbf{w}$

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

The gradient is of the form $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$

$$\begin{aligned}\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \\ &= \frac{1}{2}\sum_{i=1}^{m}\frac{\partial}{\partial w_j}(y_i - \mathbf{w}^T\mathbf{x}_i)^2\end{aligned}$$

Gradient of sum is just the sum of gradients so move partial derivative inside

Gradient of the cost J at point $\mathbf{w}$

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

The gradient is of the form $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$

$$\begin{aligned}\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \\ &= \frac{1}{2}\sum_{i=1}^{m}\frac{\partial}{\partial w_j}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \\ &= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)\frac{\partial}{\partial w_j}(y_i - w_1x_{i1} - \cdots w_jx_{ij} - \cdots)\end{aligned}$$

Apply chain rule for derivative

Expanded dot product

Gradient of the cost J at point $\mathbf{w}$

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

The gradient is of the form $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$

$$\begin{aligned}\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \\ &= \frac{1}{2}\sum_{i=1}^{m}\frac{\partial}{\partial w_j}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \\ &= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)\frac{\partial}{\partial w_j}(y_i - w_1x_{i1} - \cdots w_jx_{ij} - \cdots)\end{aligned}$$

Apply chain rule for derivatives

Expanded dot product

Only one element depends on j

Gradient of the cost J at point $\mathbf{w}$

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

The gradient is of the form $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$

$$\begin{aligned}
\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}\frac{\partial}{\partial w_j}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)\frac{\partial}{\partial w_j}(y_i - w_1x_{i1} - \cdots w_jx_{ij} - \cdots) \\
&= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)(-x_{ij})
\end{aligned}$$

Only one element depends on j

Gradient of the cost J at point $\mathbf{w}$

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

The gradient is of the form $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$

$$\begin{aligned}
\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}\frac{\partial}{\partial w_j}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)\frac{\partial}{\partial w_j}(y_i - w_1x_{i1} - \cdots w_jx_{ij} - \cdots) \\
&= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)(-x_{ij}) \\
&= -\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)x_{ij}
\end{aligned}$$

Move 2 and minus outside, 2s cancel

Gradient of the cost J at point $\mathbf{w}$

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

The gradient is of the form $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$

$$\begin{aligned}
\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}\frac{\partial}{\partial w_j}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)\frac{\partial}{\partial w_j}(y_i - w_1x_{i1} - \cdots w_jx_{ij} - \cdots) \\
&= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)(-x_{ij}) \\
&= \boxed{-\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)x_{ij}}
\end{aligned}$$

One element of the gradient vector

Sum of Error x Input

Gradient of the cost J at point $\mathbf{w}$

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

The gradient is of the form $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$

One element of the gradient vector

$$\begin{aligned}
\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}\frac{\partial}{\partial w_j}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)\frac{\partial}{\partial w_j}(y_i - w_1x_{i1} - \cdots w_jx_{ij} - \cdots) \\
&= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)(-x_{ij}) \\
&= \boxed{-\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)x_{ij}}
\end{aligned}$$

Sum of Error x Input

Negative of this gradient is how much to change jth weight

Larger features (x_{ij}) with larger errors will cause larger change

Gradient descent for LMS

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

1. Initialize $\mathbf{w}_0$

2. For $t = 0,1,2,\ldots$

   – Compute gradient of $J(\mathbf{w})$ at $\mathbf{w}_t$. Call it $\nabla J(\mathbf{w}_t)$

   Evaluate the function for *each* training example to compute the error and construct the gradient vector

$$\frac{\partial J}{\partial w_j} = -\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)x_{ij}$$

One element of $\nabla J(\mathbf{w}_t)$

$$\nabla J(\mathbf{w}_t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$$

Gradient descent for LMS

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

1. Initialize $\mathbf{w}_0$

2. For $t = 0,1,2,\ldots$

   – Compute gradient of $J(\mathbf{w})$ at $\mathbf{w}_t$. Call it $\nabla J(\mathbf{w}_t)$

   Evaluate the function for *each* training example to compute the error and construct the gradient vector

$$\frac{\partial J}{\partial w_j} = -\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)x_{ij}$$

One element of $\nabla J(\mathbf{w}_t)$

   – Update $\mathbf{w}$ as follows:

Take step in opposite direction of gradient, so minus

$$\mathbf{w}_{t+1} = \mathbf{w}_t - r\nabla J(\mathbf{w}_t)$$

   where r is the learning rate (for now a small constant)

Gradient descent for LMS

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

1. Initialize $\mathbf{w}_0$

2. For $t = 0,1,2,\ldots$ until error is below a threshold

   – Compute gradient of $J(\mathbf{w})$ at $\mathbf{w}_t$. Call it $\nabla J(\mathbf{w}_t)$

   Evaluate the function for *each* training example to compute the error and construct the gradient vector

$$\frac{\partial J}{\partial w_j} = -\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)x_{ij}$$

One element of $\nabla J(\mathbf{w}_t)$

   – Update $\mathbf{w}$ as follows:

Take step in opposite direction of gradient, so minus

$$\mathbf{w}_{t+1} = \mathbf{w}_t - r\nabla J(\mathbf{w}_t)$$

where r is the learning rate (for now a small constant)

Gradient descent for LMS

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

1. Initialize $\mathbf{w}_0$

2. For $t = 0,1,2,\ldots$ until error is below a threshold

– Compute gradient of $J(\mathbf{w})$ at $\mathbf{w}_t$. Call it $\nabla J(\mathbf{w}_t)$

Evaluate the function for *each* training example to compute the error and construct the gradient vector

$$\frac{\partial J}{\partial w_j} = -\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)x_{ij} \longleftarrow \boxed{\text{One element of } \nabla J(\mathbf{w}_t)}$$

After computing error for all training examples, get vector that you use to update weights all at once: basically a batch

– Update $\mathbf{w}$ as follows:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - r\nabla J(\mathbf{w}_t)$$

where r is the learning rate (for now a small constant)

Gradient Descent

EXAMPLE

What’s the mileage?

Suppose we want to predict the mileage of a car from its weight and age

Weight (x 100 lb)	Age (years)	Mileage per gallon
31.5	6	21
36.2	2	25
43.1	0	18
27.6	2	30

What’s the mileage?

Suppose we want to predict the mileage of a car from its weight and age

		Weight (x 100 lb)	Age (years)	Mileage per gallon
$\mathbf{x}_1$	$x_{10} = 1$	x_{11} 31.5	x_{12} 6	21
$\mathbf{x}_2$	$x_{20} = 1$	x_{21} 36.2	x_{22} 2	25
$\mathbf{x}_3$	$x_{30} = 1$	x_{31} 43.1	x_{32} 0	18
$\mathbf{x}_4$	$x_{40} = 1$	x_{41} 27.6	x_{42} 2	30

Example index

Feature index

$$\mathbf{x}_1 = \begin{bmatrix} x_{10} \\ x_{11} \\ x_{12} \end{bmatrix} = \begin{bmatrix} 1 \\ 31.5 \\ 6 \end{bmatrix}$$

What's the mileage?

Suppose we want to predict the mileage of a car from its weight and age

		Weight (x 100 lb)	Age (years)	Mileage per gallon
$\mathbf{x}_1$	$x_{10} = 1$	x_{11} 31.5	x_{12} 6	21
$\mathbf{x}_2$	$x_{20} = 1$	x_{21} 36.2	x_{22} 2	25
$\mathbf{x}_3$	$x_{30} = 1$	x_{31} 43.1	x_{32} 0	18
$\mathbf{x}_4$	$x_{40} = 1$	x_{41} 27.6	x_{42} 2	30

$$\mathbf{x}_1 = \begin{bmatrix} x_{10} \\ x_{11} \\ x_{12} \end{bmatrix} = \begin{bmatrix} 1 \\ 31.5 \\ 6 \end{bmatrix}$$

What does weight vector look like?

What's the mileage?

Suppose we want to predict the mileage of a car from its weight and age

		Weight (x 100 lb)	Age (years)	Mileage per gallon
$\mathbf{x}_1$	$x_{10} = 1$	x_{11} 31.5	x_{12} 6	21
$\mathbf{x}_2$	$x_{20} = 1$	x_{21} 36.2	x_{22} 2	25
$\mathbf{x}_3$	$x_{30} = 1$	x_{31} 43.1	x_{32} 0	18
$\mathbf{x}_4$	$x_{40} = 1$	x_{41} 27.6	x_{42} 2	30

$$\mathbf{x}_1 = \begin{bmatrix} x_{10} \\ x_{11} \\ x_{12} \end{bmatrix} = \begin{bmatrix} 1 \\ 31.5 \\ 6 \end{bmatrix}$$

$$\mathbf{w}_0 = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

What's the mileage?

Suppose we want to predict the mileage of a car from its weight and age

		Weight (x 100 lb)	Age (years)	Mileage per gallon
$\mathbf{x}_1$	$x_{10} = 1$	x_{11} 31.5	x_{12} 6	21
$\mathbf{x}_2$	$x_{20} = 1$	x_{21} 36.2	x_{22} 2	25
$\mathbf{x}_3$	$x_{30} = 1$	x_{31} 43.1	x_{32} 0	18
$\mathbf{x}_4$	$x_{40} = 1$	x_{41} 27.6	x_{42} 2	30

$$\mathbf{x}_1 = \begin{bmatrix} x_{10} \\ x_{11} \\ x_{12} \end{bmatrix} = \begin{bmatrix} 1 \\ 31.5 \\ 6 \end{bmatrix}$$

$$\mathbf{w}_0 = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

How do we update weights?

What's the mileage?

Suppose we want to predict the mileage of a car from its weight and age

		Weight (x 100 lb)	Age (years)	Mileage per gallon
$\mathbf{x}_1$	$x_{10} = 1$	x_{11} 31.5	x_{12} 6	21
$\mathbf{x}_2$	$x_{20} = 1$	x_{21} 36.2	x_{22} 2	25
$\mathbf{x}_3$	$x_{30} = 1$	x_{31} 43.1	x_{32} 0	18
$\mathbf{x}_4$	$x_{40} = 1$	x_{41} 27.6	x_{42} 2	30

$$\mathbf{x}_1 = \begin{bmatrix} x_{10} \\ x_{11} \\ x_{12} \end{bmatrix} = \begin{bmatrix} 1 \\ 31.5 \\ 6 \end{bmatrix}$$

$$\mathbf{w}_0 = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\nabla J(\mathbf{w}_t) = \left[\frac{\partial J}{\partial w_0}, \frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2} \right]$$

$$\frac{\partial J}{\partial w_j} = -\sum_{i=1}^{m} (y_i - \mathbf{w}^T \mathbf{x}_i) x_{ij} \leftarrow$$

One element of $\nabla J(\mathbf{w}_t)$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - r \nabla J(\mathbf{w}_t)$$

What's the mileage?

Suppose we want to predict the mileage of a car from its weight and age

		Weight (x 100 lb)	Age (years)	Mileage per gallon
$\mathbf{x}_1$	$x_{10} = 1$	x_{11} 31.5	x_{12} 6	21
$\mathbf{x}_2$	$x_{20} = 1$	x_{21} 36.2	x_{22} 2	25
$\mathbf{x}_3$	$x_{30} = 1$	x_{31} 43.1	x_{32} 0	18
$\mathbf{x}_4$	$x_{40} = 1$	x_{41} 27.6	x_{42} 2	30

$$\mathbf{x}_1 = \begin{bmatrix} x_{10} \\ x_{11} \\ x_{12} \end{bmatrix} = \begin{bmatrix} 1 \\ 31.5 \\ 6 \end{bmatrix}$$

$$\mathbf{w}_0 = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\nabla J(\mathbf{w}_t) = \left[\frac{\partial J}{\partial w_0}, \frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2} \right]$$

$$\frac{\partial J}{\partial w_0} = -\sum_{i=1}^{m} (y_i - \mathbf{w}^T \mathbf{x}_i) x_{i0}$$

$$\frac{\partial J}{\partial w_j} = -\sum_{i=1}^{m} (y_i - \mathbf{w}^T \mathbf{x}_i) x_{ij}$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - r \nabla J(\mathbf{w}_t)$$

What's the mileage?

Suppose we want to predict the mileage of a car from its weight and age

			Weight (x 100 lb)		Age (years)	Mileage per gallon
$\mathbf{x}_1$	$x_{10} = 1$	x_{11}	31.5	x_{12}	6	21
$\mathbf{x}_2$	$x_{20} = 1$	x_{21}	36.2	x_{22}	2	25
$\mathbf{x}_3$	$x_{30} = 1$	x_{31}	43.1	x_{32}	0	18
$\mathbf{x}_4$	$x_{40} = 1$	x_{41}	27.6	x_{42}	2	30

$$\mathbf{x}_1 = \begin{bmatrix} x_{10} \\ x_{11} \\ x_{12} \end{bmatrix} = \begin{bmatrix} 1 \\ 31.5 \\ 6 \end{bmatrix}$$

$$\mathbf{w}_0 = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\nabla J(\mathbf{w}_t) = \left[\frac{\partial J}{\partial w_0}, \frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2} \right]$$

$$\frac{\partial J}{\partial w_j} = -\sum_{i=1}^{m} (y_i - \mathbf{w}^T \mathbf{x}_i) x_{ij}$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - r \nabla J(\mathbf{w}_t)$$

$$\frac{\partial J}{\partial w_0} = -\sum_{i=1}^{m} (y_i - \mathbf{w}^T \mathbf{x}_i) x_{i0}$$

$$\frac{\partial J}{\partial w_1} = -\sum_{i=1}^{m} (y_i - \mathbf{w}^T \mathbf{x}_i) x_{i1}$$

$$\frac{\partial J}{\partial w_2} = -\sum_{i=1}^{m} (y_i - \mathbf{w}^T \mathbf{x}_i) x_{i2}$$

What's the mileage?

Suppose we want to predict the mileage of a car from its weight and age

		Weight (x 100 lb)	Age (years)	Mileage per gallon
$\mathbf{x}_1$	$x_{10} = 1$	x_{11} 31.5	x_{12} 6	21
$\mathbf{x}_2$	$x_{20} = 1$	x_{21} 36.2	x_{22} 2	25
$\mathbf{x}_3$	$x_{30} = 1$	x_{31} 43.1	x_{32} 0	18
$\mathbf{x}_4$	$x_{40} = 1$	x_{41} 27.6	x_{42} 2	30

$$\mathbf{x}_1 = \begin{bmatrix} x_{10} \\ x_{11} \\ x_{12} \end{bmatrix} = \begin{bmatrix} 1 \\ 31.5 \\ 6 \end{bmatrix}$$

$$\mathbf{w}_0 = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\frac{\partial J}{\partial w_0} = -\sum_{i=1}^{m} (y_i - \mathbf{w}^T\mathbf{x}_i)x_{i0}$$

$$= -(y_1 - \mathbf{w}^T\mathbf{x}_1)x_{10} - (y_2 - \mathbf{w}^T\mathbf{x}_2)x_{20} - (y_3 - \mathbf{w}^T\mathbf{x}_3)x_{30} - (y_4 - \mathbf{w}^T\mathbf{x}_4)x_{40}$$

What's the mileage?

Suppose we want to predict the mileage of a car from its weight and age

			Weight (x 100 lb)		Age (years)	Mileage per gallon
$\mathbf{x}_1$	$x_{10}=1$	x_{11}	31.5	x_{12}	6	21
$\mathbf{x}_2$	$x_{20}=1$	x_{21}	36.2	x_{22}	2	25
$\mathbf{x}_3$	$x_{30}=1$	x_{31}	43.1	x_{32}	0	18
$\mathbf{x}_4$	$x_{40}=1$	x_{41}	27.6	x_{42}	2	30

$$\mathbf{x}_1 = \begin{bmatrix} x_{10} \\ x_{11} \\ x_{12} \end{bmatrix} = \begin{bmatrix} 1 \\ 31.5 \\ 6 \end{bmatrix}$$

$$\mathbf{w}_0 = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\frac{\partial J}{\partial w_0} = -\sum_{i=1}^{m} (y_i - \mathbf{w}^T\mathbf{x}_i)x_{i0}$$
$$= -(y_1 - \mathbf{w}^T\mathbf{x}_1)x_{10} - (y_2 - \mathbf{w}^T\mathbf{x}_2)x_{20} - (y_3 - \mathbf{w}^T\mathbf{x}_3)x_{30} - (y_4 - \mathbf{w}^T\mathbf{x}_4)x_{40}$$

$$\frac{\partial J}{\partial w_1} = -\sum_{i=1}^{m} (y_i - \mathbf{w}^T\mathbf{x}_i)x_{i1}$$
$$= -(y_1 - \mathbf{w}^T\mathbf{x}_1)x_{11} - (y_2 - \mathbf{w}^T\mathbf{x}_2)x_{21} - (y_3 - \mathbf{w}^T\mathbf{x}_3)x_{31} - (y_4 - \mathbf{w}^T\mathbf{x}_4)x_{41}$$

$$\frac{\partial J}{\partial w_2} = -\sum_{i=1}^{m} (y_i - \mathbf{w}^T\mathbf{x}_i)x_{i2}$$
$$= -(y_1 - \mathbf{w}^T\mathbf{x}_1)x_{12} - (y_2 - \mathbf{w}^T\mathbf{x}_2)x_{22} - (y_3 - \mathbf{w}^T\mathbf{x}_3)x_{32} - (y_4 - \mathbf{w}^T\mathbf{x}_4)x_{42}$$

What's the mileage?

Suppose we want to predict the mileage of a car from its weight and age

		Weight (x 100 lb)	Age (years)	Mileage per gallon
$\mathbf{x}_1$	$x_{10} = 1$	x_{11} 31.5	x_{12} 6	21
$\mathbf{x}_2$	$x_{20} = 1$	x_{21} 36.2	x_{22} 2	25
$\mathbf{x}_3$	$x_{30} = 1$	x_{31} 43.1	x_{32} 0	18
$\mathbf{x}_4$	$x_{40} = 1$	x_{41} 27.6	x_{42} 2	30

$$\mathbf{x}_1 = \begin{bmatrix} x_{10} \\ x_{11} \\ x_{12} \end{bmatrix} = \begin{bmatrix} 1 \\ 31.5 \\ 6 \end{bmatrix}$$

$$\mathbf{w}_0 = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{aligned} \frac{\partial J}{\partial w_0} &= -\sum_{i=1}^{m} (y_i - \mathbf{w}^T\mathbf{x}_i)x_{i0} \\ &= -(y_1 - \mathbf{w}^T\mathbf{x}_1)x_{10} - (y_2 - \mathbf{w}^T\mathbf{x}_2)x_{20} - (y_3 - \mathbf{w}^T\mathbf{x}_3)x_{30} - (y_4 - \mathbf{w}^T\mathbf{x}_4)x_{40} \\ &= -(21 - \mathbf{w}^T\mathbf{x}_1)1 - (25 - \mathbf{w}^T\mathbf{x}_2)1 - (18 - \mathbf{w}^T\mathbf{x}_3)1 - (30 - \mathbf{w}^T\mathbf{x}_4)1 \\ &= -(21-0)1 - (25-0)1 - (18-0)1 - (30-0)1 \\ &= -94 \end{aligned}$$

What's the mileage?

Suppose we want to predict the mileage of a car from its weight and age

			Weight (x 100 lb)		Age (years)	Mileage per gallon
$\mathbf{x}_1$	$x_{10} = 1$	x_{11}	31.5	x_{12}	6	21
$\mathbf{x}_2$	$x_{20} = 1$	x_{21}	36.2	x_{22}	2	25
$\mathbf{x}_3$	$x_{30} = 1$	x_{31}	43.1	x_{32}	0	18
$\mathbf{x}_4$	$x_{40} = 1$	x_{41}	27.6	x_{42}	2	30

$$\mathbf{x}_1 = \begin{bmatrix} x_{10} \\ x_{11} \\ x_{12} \end{bmatrix} = \begin{bmatrix} 1 \\ 31.5 \\ 6 \end{bmatrix}$$

$$\mathbf{w}_0 = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{aligned}
\frac{\partial J}{\partial w_1} &= -\sum_{i=1}^{m} (y_i - \mathbf{w}^T\mathbf{x}_i)x_{i1} \\
&= -(y_1 - \mathbf{w}^T\mathbf{x}_1)x_{11} - (y_2 - \mathbf{w}^T\mathbf{x}_2)x_{21} - (y_3 - \mathbf{w}^T\mathbf{x}_3)x_{31} - (y_4 - \mathbf{w}^T\mathbf{x}_4)x_{41} \\
&= -(21 - \mathbf{w}^T\mathbf{x}_1)31.5 - (25 - \mathbf{w}^T\mathbf{x}_2)36.2 - (18 - \mathbf{w}^T\mathbf{x}_3)43.1 - (30 - \mathbf{w}^T\mathbf{x}_4)27.6 \\
&= -(21 - 0)31.5 - (25 - 0)36.2 - (18 - 0)43.1 - (30 - 0)27.6 \\
&= -661.5 - 905 - 775 - 828 \\
&= -3169.5
\end{aligned}$$

What’s the mileage?

Suppose we want to predict the mileage of a car from its weight and age

		Weight (x 100 lb)	Age (years)	Mileage per gallon
$\mathbf{x}_1$	$x_{10} = 1$	x_{11} 31.5	x_{12} 6	21
$\mathbf{x}_2$	$x_{20} = 1$	x_{21} 36.2	x_{22} 2	25
$\mathbf{x}_3$	$x_{30} = 1$	x_{31} 43.1	x_{32} 0	18
$\mathbf{x}_4$	$x_{40} = 1$	x_{41} 27.6	x_{42} 2	30

$$\mathbf{x}_1 = \begin{bmatrix} x_{10} \\ x_{11} \\ x_{12} \end{bmatrix} = \begin{bmatrix} 1 \\ 31.5 \\ 6 \end{bmatrix}$$

$$\mathbf{w}_0 = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{aligned}
\frac{\partial J}{\partial w_2} &= -\sum_{i=1}^{m} (y_i - \mathbf{w}^T\mathbf{x}_i)x_{i2} \\
&= -(y_1 - \mathbf{w}^T\mathbf{x}_1)x_{12} - (y_2 - \mathbf{w}^T\mathbf{x}_2)x_{22} - (y_3 - \mathbf{w}^T\mathbf{x}_3)x_{32} - (y_4 - \mathbf{w}^T\mathbf{x}_4)x_{42} \\
&= -(21 - \mathbf{w}^T\mathbf{x}_1)6 - (25 - \mathbf{w}^T\mathbf{x}_2)2 - (18 - \mathbf{w}^T\mathbf{x}_3)0 - (30 - \mathbf{w}^T\mathbf{x}_4)2 \\
&= -(21 - 0)6 - (25 - 0)2 - (18 - 0)0 - (30 - 0)2 \\
&= -126 - 50 - 0 - 60 \\
&= -236
\end{aligned}$$

What's the mileage?

Suppose we want to predict the mileage of a car from its weight and age

		Weight (x 100 lb)	Age (years)	Mileage per gallon
$\mathbf{x}_1$	$x_{10} = 1$	x_{11} 31.5	x_{12} 6	21
$\mathbf{x}_2$	$x_{20} = 1$	x_{21} 36.2	x_{22} 2	25
$\mathbf{x}_3$	$x_{30} = 1$	x_{31} 43.1	x_{32} 0	18
$\mathbf{x}_4$	$x_{40} = 1$	x_{41} 27.6	x_{42} 2	30

$$\mathbf{x}_1 = \begin{bmatrix} x_{10} \\ x_{11} \\ x_{12} \end{bmatrix} = \begin{bmatrix} 1 \\ 31.5 \\ 6 \end{bmatrix}$$

$$\mathbf{w}_0 = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\nabla J(\mathbf{w}_t) = \left[\frac{\partial J}{\partial w_0}, \frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}\right]$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - r\nabla J(\mathbf{w}_t)$$

$$\frac{\partial J}{\partial w_0} = -\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)x_{i0} = -94$$

$$\frac{\partial J}{\partial w_1} = -\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)x_{i1} = -3169.5$$

$$\frac{\partial J}{\partial w_2} = -\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)x_{i2} = -236$$

What's the mileage?

Suppose we want to predict the mileage of a car from its weight and age

			Weight (x 100 lb)		Age (years)	Mileage per gallon
$\mathbf{x}_1$	$x_{10} = 1$	x_{11}	31.5	x_{12}	6	21
$\mathbf{x}_2$	$x_{20} = 1$	x_{21}	36.2	x_{22}	2	25
$\mathbf{x}_3$	$x_{30} = 1$	x_{31}	43.1	x_{32}	0	18
$\mathbf{x}_4$	$x_{40} = 1$	x_{41}	27.6	x_{42}	2	30

$$\mathbf{x}_1 = \begin{bmatrix} x_{10} \\ x_{11} \\ x_{12} \end{bmatrix} = \begin{bmatrix} 1 \\ 31.5 \\ 6 \end{bmatrix}$$

$$\mathbf{w}_0 = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\nabla J(\mathbf{w}_t) = \left[-94, 3169.5, -236\right]$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - r\nabla J(\mathbf{w}_t)$$

$$\mathbf{w}_{t+1} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} - r \begin{bmatrix} -94 \\ -3169.5 \\ -236 \end{bmatrix}$$

$$= \begin{bmatrix} 94 \\ 3169.5 \\ 236 \end{bmatrix}$$

$$\frac{\partial J}{\partial w_0} = -\sum_{i=1}^{m} (y_i - \mathbf{w}^T\mathbf{x}_i)x_{i0} = -94$$

$$\frac{\partial J}{\partial w_1} = -\sum_{i=1}^{m} (y_i - \mathbf{w}^T\mathbf{x}_i)x_{i1} = -3169.5$$

$$\frac{\partial J}{\partial w_2} = -\sum_{i=1}^{m} (y_i - \mathbf{w}^T\mathbf{x}_i)x_{i2} = -236$$

Gradient Descent

CONVERGENCE

How to improve likelihood of convergence

- Normalize values of features and labels: Important to normalize features when using gradient descent (otherwise takes longer to converge). All features should have a similar scale

- Decrease learning rate over time

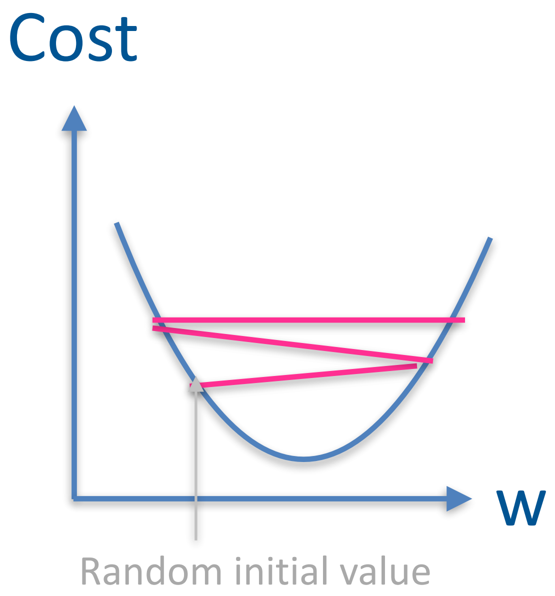
- Check for weights converging

- Cross-validation to determine how best to set hyper-parameters like number of epochs or learning rate

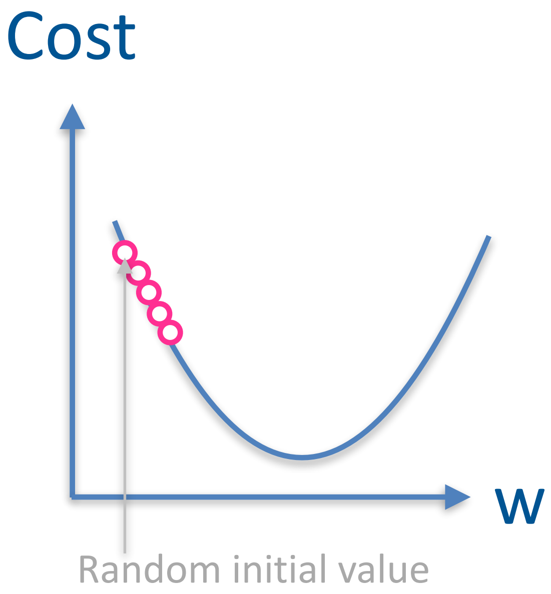
Learning Rates and Convergence

- In the general ("non-separable") case the learning rate r must decrease to zero to guarantee convergence.
- The learning rate is called the *step size*. There are more sophisticated algorithms that choose the step size automatically and converge faster.
- Choosing a better starting point also has impact.

Impact of learning rate



Learning rate too large



Learning rate too small

Gradient descent

Algorithm is guaranteed to converge to the minimum of J if learning rate r is small enough (small enough steps) or the learning rate is decreased appropriately

Why? The objective J is a convex function here (LMS for linear regression): the surface contains only a single global minimum. The surface may have local minimum if the loss function is different.

Decreasing learning rate over time

In order to guarantee that the algorithm will converge, the learning rate should decrease over time. Here is a general formula

- At iteration t

$$r_t = \frac{c_1}{t^a + c_2} \text{ where}$$

$$0.5 < a < 2$$

$$c_1 > 0$$

$$c_2 \geq 0$$

When should algorithm stop?

1. Stop after fixed number of iterations

2. Stop once prediction error is less than threshold

3. Stop when validation loss stops changing

Stopping criteria

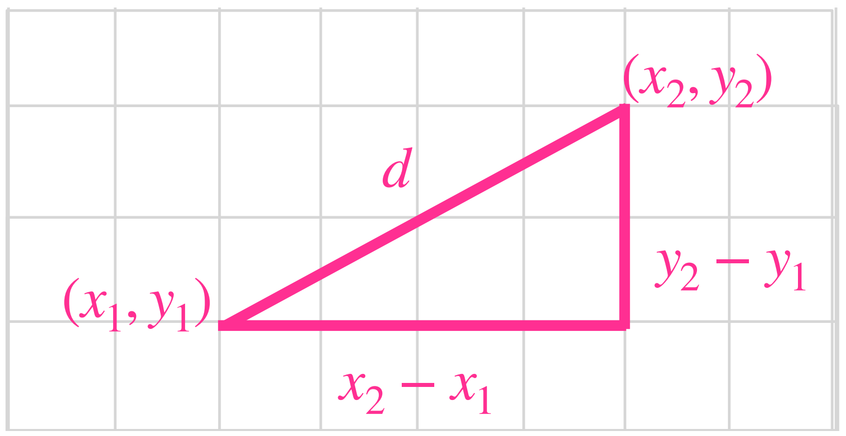
For most functions, you probably won't get the gradient to be exactly equal to $\mathbf{0}$ in a reasonable amount of time

Once the gradient is sufficiently close to $\mathbf{0}$, stop trying to minimize further

How do we measure how close a gradient is to $\mathbf{0}$?
Gradient is just a vector, so can compute distance.

Distance

How far apart are two points?



Euclidean distance between 2 points in 2 dimensions:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

In 3 dimensions (x, y, z):

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Distance

General formula for Euclidean distance between 2 points with k dimensions:

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{k} (p_i - q_i)^2}$$

where $\mathbf{p}$ and $\mathbf{q}$ are 2 points (each represents a k -dimensional vector)

Distance

A special case is the distance between a point and zero (the origin)

$$d(\mathbf{p}, \mathbf{0}) = \sqrt{\sum_{i=1}^{k} p_i^2}$$ also written $||\mathbf{p}||$

This is called the **Euclidean norm** of $\mathbf{p}$

- A norm is a measure of a vector's length
- The Euclidean norm is also called the **L2 norm**

Stopping criteria

Stop when the norm of the gradient is below some threshold, θ:

$$||\nabla L(\mathbf{w})|| < \theta$$

Common values of θ are around .01, but if it is taking too long, you can make the threshold larger

Gradient descent

1. Initialize the parameters $\mathbf{w}$ to some guess (usually all zeroes, or random values)

2. Update the parameters:

$$\mathbf{w} = \mathbf{w} - r \nabla L(\mathbf{w})$$

$$r_t = \frac{c_1}{t^a + c_2}$$

3. Repeat step 2 until $|| \nabla L(\mathbf{w}) || < \theta$ or until the maximum number of iterations is reached

Linear Regression

INCREMENTAL/STOCHASTIC GRADIENT DESCENT

Gradient descent for LMS

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

1. Initialize $\mathbf{w}^0$

2. For $t = 0,1,2,\ldots$ until error is below a threshold
   - Compute gradient of $J(\mathbf{w})$ at $\mathbf{w}^t$. Call it $\nabla J(\mathbf{w}^t)$

     Evaluate the function for *each* training example to compute the error and construct the gradient vector

     $$\frac{\partial J}{\partial w_j} = -\sum_{i=1}^{m}(y_i - \mathbf{w}^T x_i)x_{ij}$$

   - Update $\mathbf{w}$ as follows:

     $$\mathbf{w}^{t+1} = \mathbf{w}^t - r\nabla J(\mathbf{w}^t)$$

The weight vector is not updated until *all* errors are calculated.

Why not make early updates to the weight vector as soon as we encounter errors instead of waiting for a full pass over the data?

Incremental/stochastic gradient descent

Repeat for each example $(\mathbf{x}_i, y_i)$

- Pretend that entire training set is represented by this single example
- Use this example to calculate the gradient and update the model

Contrast with batch gradient descent which makes one update to the weight vector for every pass over the data

Incremental/stochastic gradient descent

1. Initialize $\mathbf{w}$

2. For $t = 0,1,2,\ldots$ until error is below a threshold
   - For each training example $(\mathbf{x}_i, y_i)$, update $\mathbf{w}$. For each element of the weight vector (w_j)

$$w_j^{t+1} = w_j^t - r(y_i - \mathbf{w}^T\mathbf{x}_i)x_{ij}$$

Contrast with the previous method, where the weights are updated only after all examples are processed once

Incremental/stochastic gradient descent

1. Initialize $\mathbf{w}$

2. For $t = 0,1,2,\ldots$ until error is below a threshold
   - For each training example $(\mathbf{x}_i, y_i)$, update $\mathbf{w}$. For each element of the weight vector (w_j)

$$w_j^{t+1} = w_j^t - r(y_i - \mathbf{w}^T\mathbf{x}_i)x_{ij}$$

This update rule is also called the Widrow-Hoff rule in the neural networks literature

May get close to optimum much faster than the batch version. In general - does not converge to global minimum. Decreasing r with time guarantees convergence. But, online/incremental algorithms are often preferred when the training set is very large

Linear regression: summary

- What we want: predict a real-valued output using feature representation of the input

- Assumption: output is a linear function of the inputs

- Learning by minimizing total cost
  - gradient descent and stochastic gradient descent to find the *best* weight vector
  - This particular optimization can be computed directly by framing the problem as a matrix problem

Linear Regression

REGULARIZATION

Generalization

- Prediction functions that work on the training data might not work on other data

- Minimizing the training error is a reasonable thing to do, but it’s possible to minimize it “too well”

- **Overfitting:** your function matches the training data well but is not learning general rules that will work for new data

Regularization

- Modify learning algorithm to favor “simpler” prediction rules to avoid overfitting

- Most commonly, regularization refers to modifying the loss function to **penalize** certain values of the weights you are learning. Specifically, penalize weights that are large.

Regularization

- How do we define whether weights are large?

$$d(\mathbf{w}, \mathbf{0}) = \sqrt{\sum_{i=1}^{k} (w_i)^2} = ||\mathbf{w}||$$

Note that bias term w_0 is not regularized

- This is called the L2 norm of $\mathbf{w}$
  - A norm is a measure of a vector’s length
  - Also called the Euclidean norm

Regularization

- New goal for minimization

$$L(\mathbf{w}) + \lambda ||\mathbf{w}||^2$$

Square to eliminate square root: easier to work with mathematically

This is whatever loss function we are using

By minimizing this we prefer solutions where $\mathbf{w}$ is closer to $\mathbf{0}$

λ is a hyperparameter that adjusts trade-off between low training loss and having low weights

Regularization

- Regularization helps the computational problem because gradient descent won’t try to make some feature weights grow larger and larger

- At some point, the penalty of having too large $||w||^2$ will outweigh whatever gain you would make in your loss function.

Regularization

- This also helps with generalization because it won’t give large weight to features unless there is sufficient evidence that they are useful

- The usefulness of a feature toward improving the loss has to outweigh the cost of having large feature weights

Regularization

- More generally

$$L(\mathbf{w}) + \lambda R(\mathbf{w})$$

λ is called the regularization strength.

Other common names for λ: *alpha* in sklearn, *C* in many algorithms. Usually C actually refers to the inverse regularization strength, 1/λ. Figure out which one your implementation is using (whether this will increase or decrease regularization)

This is called the regularization term or regularizer or penalty. The squared L2 norm is one kind of penalty, but there are others

L2 Regularization

- When the regularizer is the squared L2 norm $||\mathbf{w}||^2$, this is called L2 regularization.
- This is the most common type of regularization
- When used with linear regression, this is called Ridge regression
- Logistic regression implementations usually use L2 regularization by default
- L2 regularization can be added to other algorithms like perceptron (or any gradient descent algorithm)

L2 Regularization

- The function $R(\mathbf{w}) = ||\mathbf{w}||^2$ is convex, so if it is added to a convex loss function, the combined function will still be convex.

L1 Regularization

- Another common regularizer is the L1 norm:

$$||\mathbf{w}||_1 = \sum_{j=1}^{k} |w_j|$$

- When used with linear regression, this is called Lasso

- Often results in many weights being exactly 0 (while L2 just makes them small but nonzero)

L1+L1 Regularization

- L2 and L1 regularization can be combined

$$R(\mathbf{w}) = \lambda_2 ||\mathbf{w}||^2 + \lambda_1 ||\mathbf{w}||_1$$

- Also called ElasticNet
- Can work better than either type alone
- Can adjust hyperparameters to control which of the two penalties is more important
- Once training is done, remove regularization term to measure model performance

Feature normalization

- The scale of the feature values matters when using regularization
- If one feature has values between [0, 1] and another between [0, 10000], the learned weights might be on very different scales – but whatever weights are “naturally” larger are going to get penalized more by the regularizer.
- Feature normalization or standardization refers to converting the values to a standard range.