Announcements

HW2

- Due Mon 2/28

Plan

Complete Data collection and management

- Wrap up Free text and NLP

Begin Statistical modeling and machine learning

- Intro to ML and
- Linear regression

15-388/688 - Practical Data Science: Intro to Machine Learning & Linear Regression

Pat Virtue
Carnegie Mellon University
Spring 2022

Slide credits: CMU AI, Zico Kolter

Outline

Least squares regression: a simple example

Machine learning notation

Linear regression revisited

Matrix/vector notation and analytic solutions

Implementing linear regression

Outline

Least squares regression: a simple example

Machine learning notation

Linear regression revisited

Matrix/vector notation and analytic solutions

Implementing linear regression

A simple example: predicting electricity use

What will peak power consumption be in Pittsburgh tomorrow?

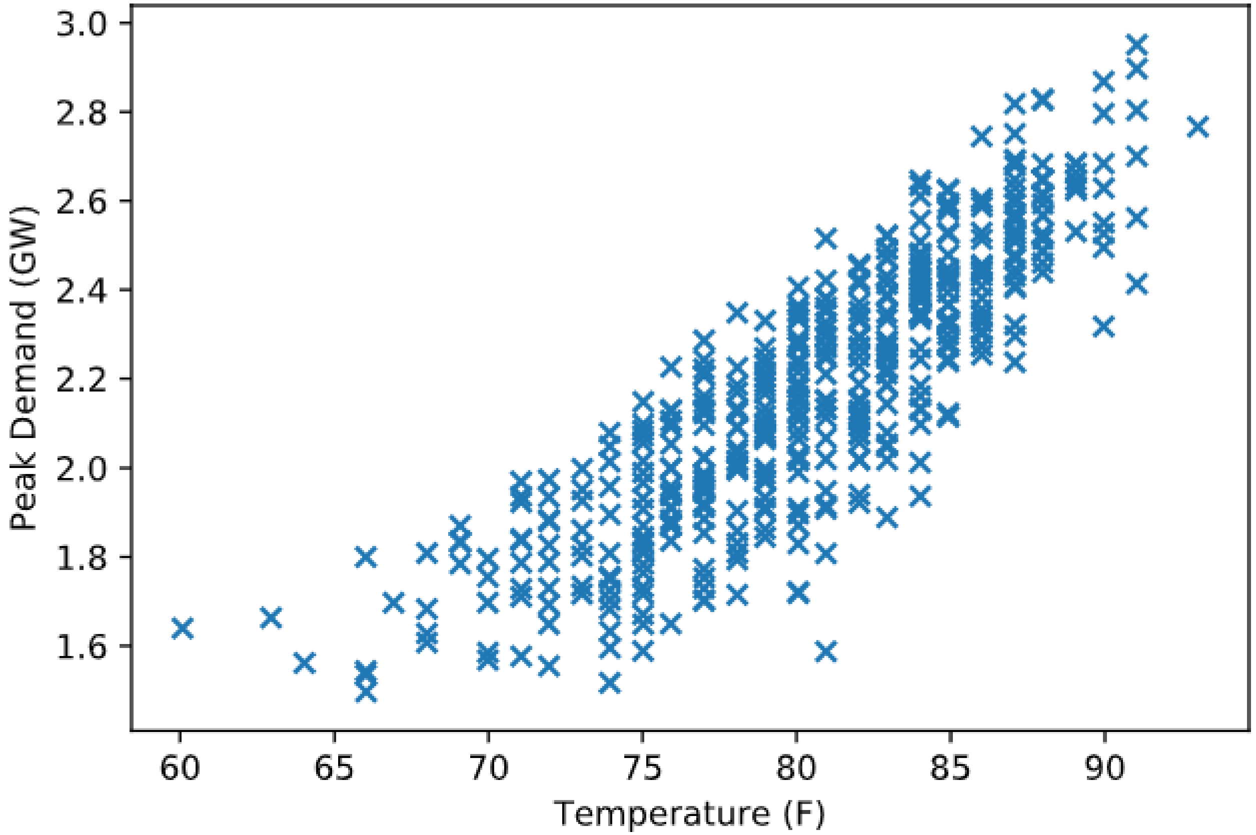
Difficult to build an “a priori” model from first principles to answer this question

But, relatively easy to record past days of consumption, plus additional features that affect consumption (i.e., weather)

Date	High Temperature (F)	Peak Demand (GW)
2011-06-01	84.0	2.651
2011-06-02	73.0	2.081
2011-06-03	75.2	1.844
2011-06-04	84.9	1.959
…	…	…

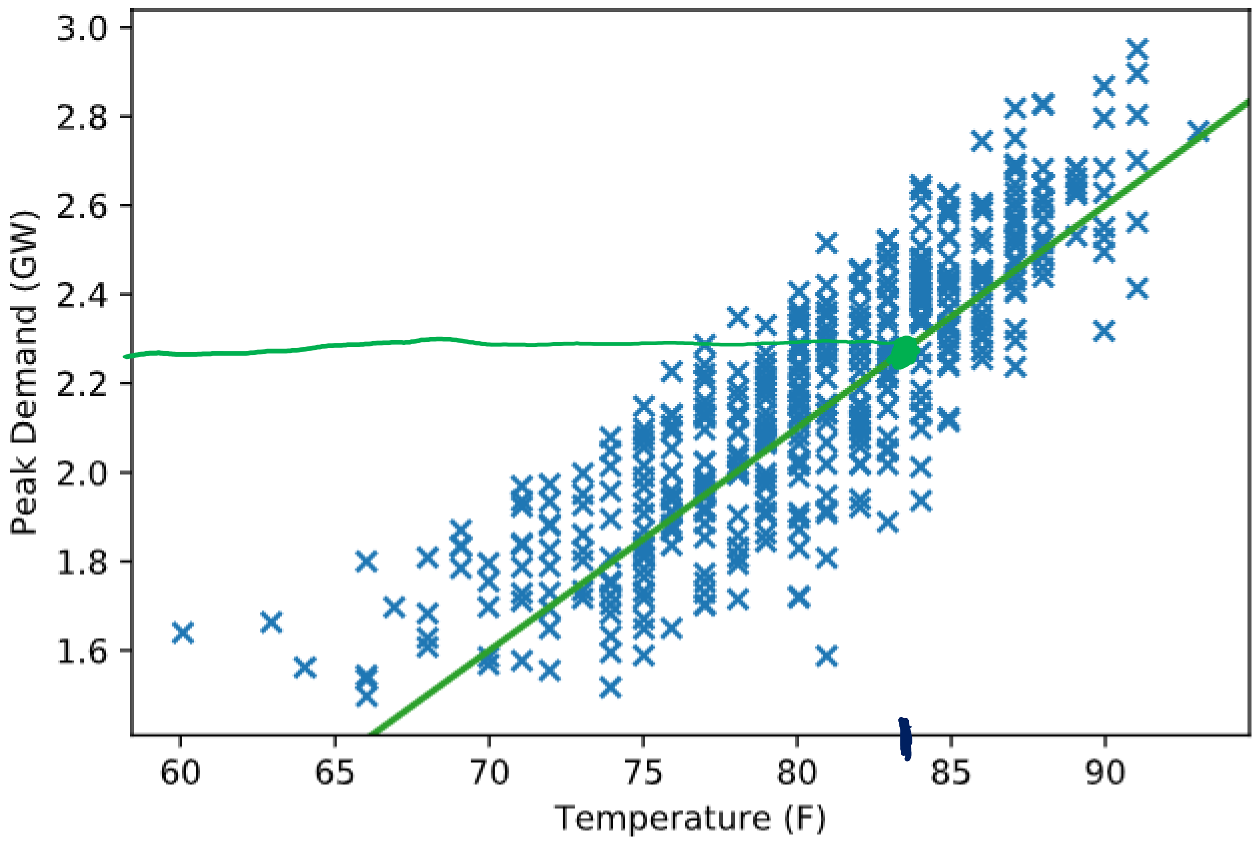
Plot of consumption vs. temperature

Plot of high temperature vs. peak demand for summer months (June – August) for past six years



Hypothesis: linear model

Let's suppose that the peak demand approximately fits a *linear model*



Hypothesis: linear model

Let’s suppose that the peak demand approximately fits a *linear model*

$$y \approx m x + b$$

$$\text{Peak\_Demand} \approx \theta_1 \cdot \text{High\_Temperature} + \theta_2$$

Here θ_1 is the “slope” of the line, and $\theta_2 =$ is the intercept

Making predictions

Importantly, our model also lets us make *predictions* about new days

What will the peak demand be tomorrow?

If we know the high temperature will be 72 degrees (ignoring for now that this is *also* a prediction), then we can predict peak demand to be:

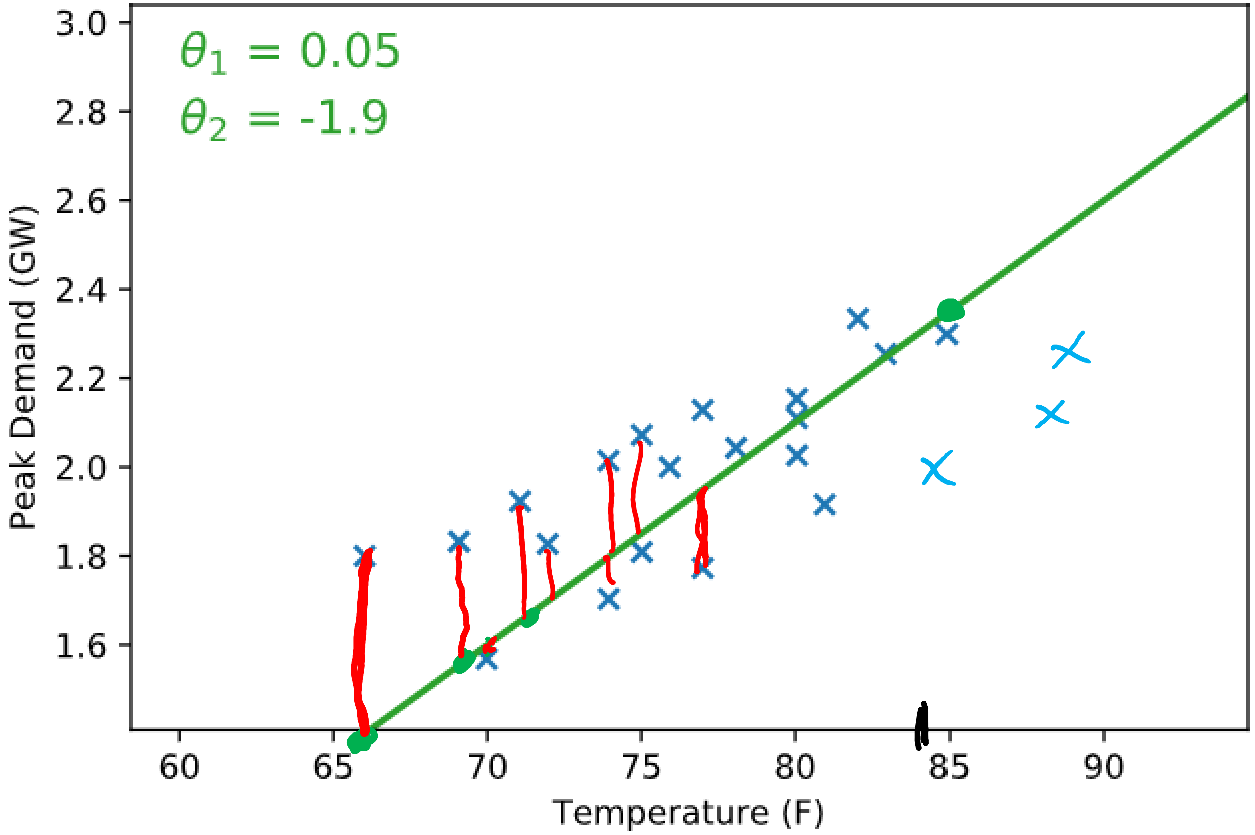
$$\text{Predicted\_Peak\_Demand} = \theta_1 \cdot 72 + \theta_2 = 1.821 \text{ GW}$$

Equivalent to just "finding the point on the line"

Predicted output for each data point

$\text{Peak\_Demand}^{(i)}$

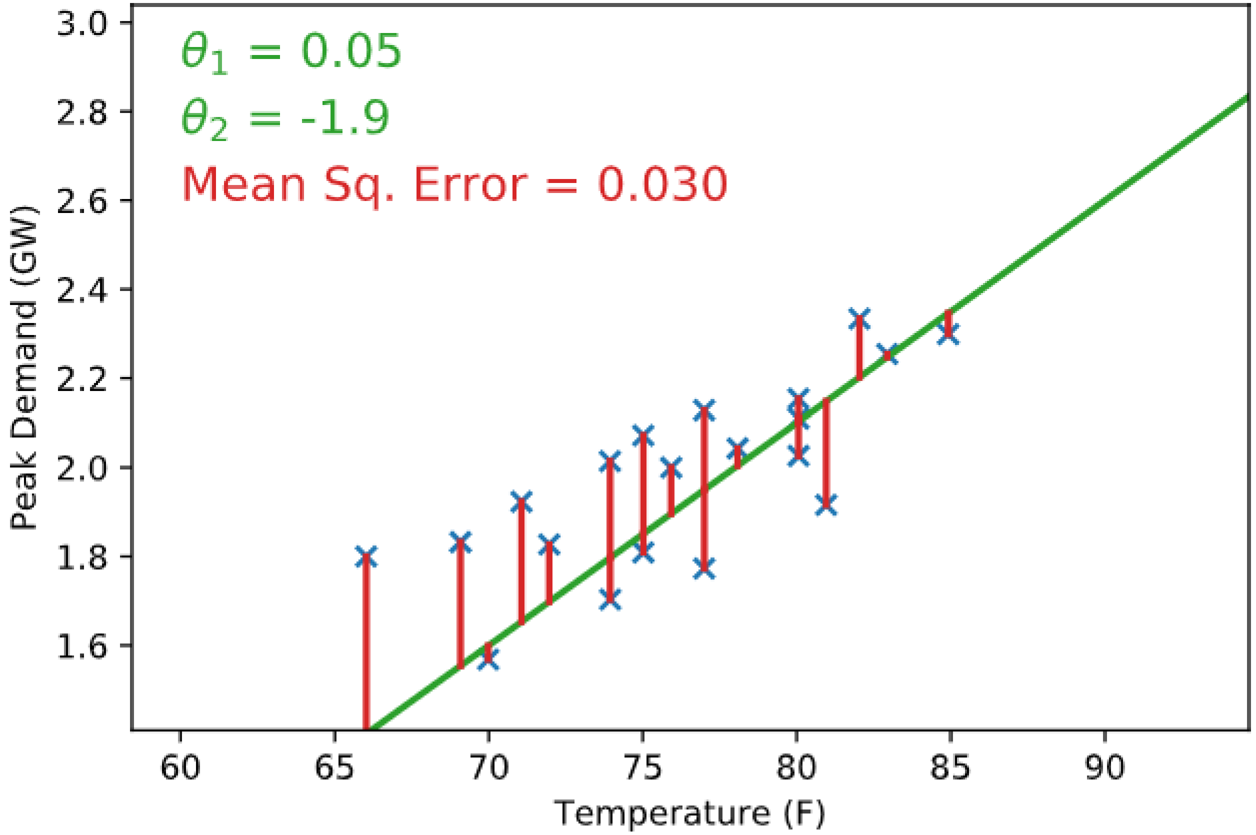
$$\text{Predicted\_Peak\_Demand}^{(i)} = \theta_1 \cdot \text{High\_Temperature}^{(i)} + \theta_2$$



Hypothesis: linear model

$\text{Peak\_Demand}^{(i)}$

$$\text{Predicted\_Peak\_Demand}^{(i)} = \theta_1 \cdot \text{High\_Temperature}^{(i)} + \theta_2$$



Hypothesis: linear model

Let's suppose that the peak demand approximately fits a *linear model*

$$\text{Predicted\_Peak\_Demand} = \theta_1 \cdot \text{High\_Temperature} + \theta_2$$

Here θ_1 is the "slope" of the line, and θ_2 is the intercept

How do we find a "good" fit to the data?

Many possibilities, but natural objective is to minimize some difference between this line and the observed data, e.g. squared loss

$E(\theta_1, \theta_2)$ data

$$E(\theta) = \sum_{i \in \text{days}} \left(\text{Predicted\_Peak\_Demand}^{(i)} - \text{Peak\_Demand}^{(i)}\right)^2$$

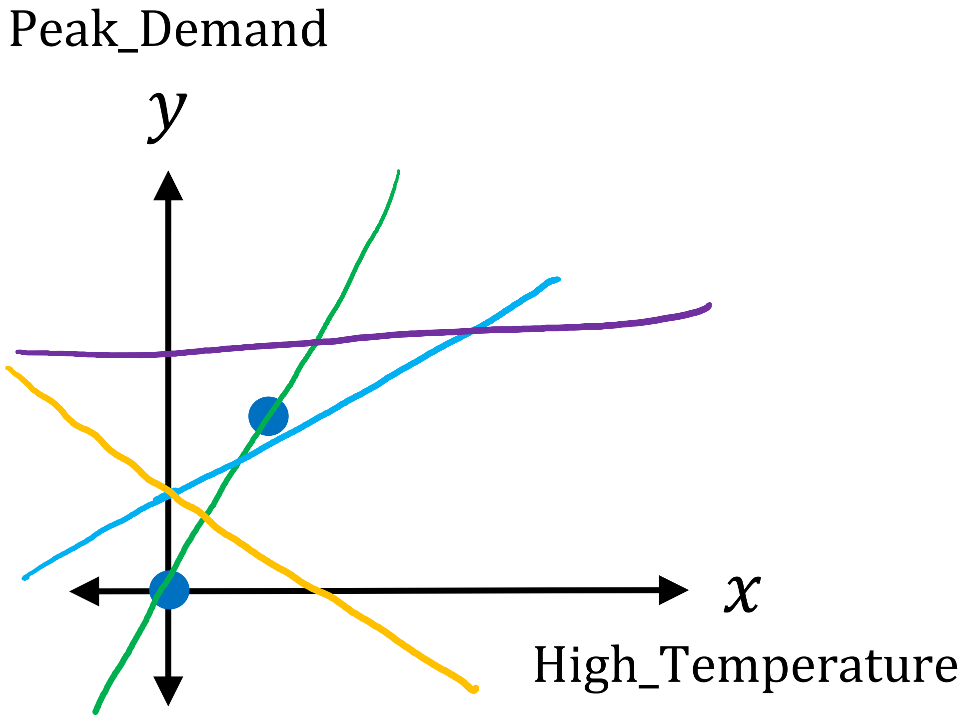
$$E(\theta) = \sum_{i \in \text{days}} \left(\theta_1 \cdot \text{High\_Temperature}^{(i)} + \theta_2 - \text{Peak\_Demand}^{(i)}\right)^2$$

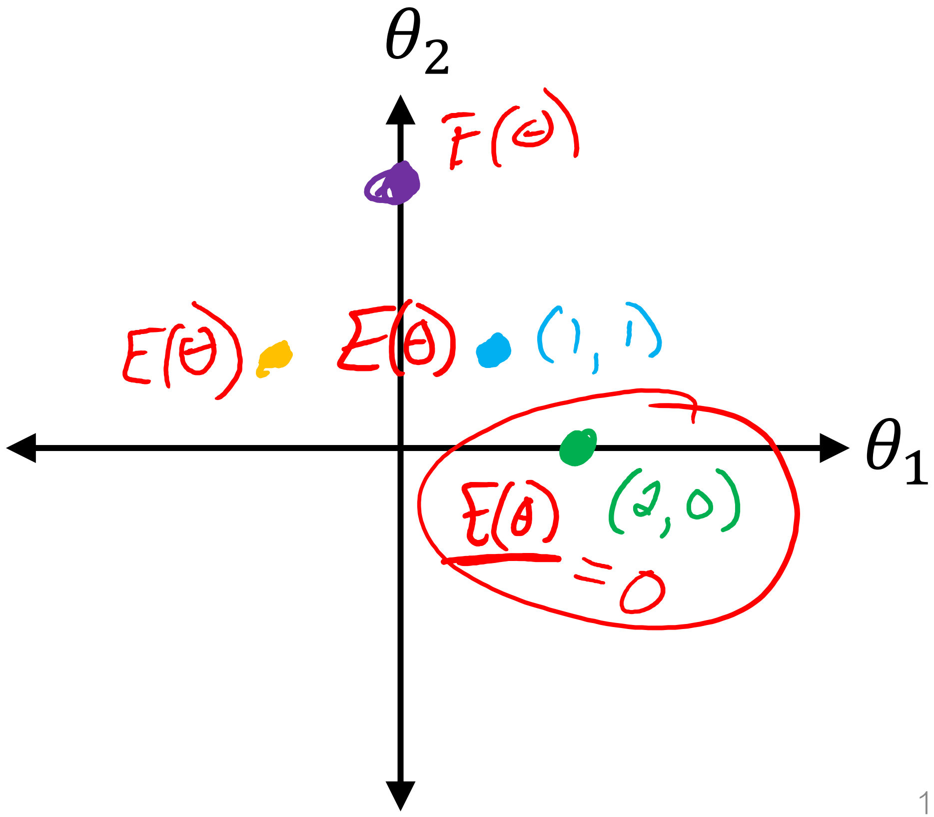
How do we find parameters?

How do we find the parameters θ_1, θ_2 that minimize the function

$$E(\theta) = \mathrm{E}(\theta_1, \theta_2) = \sum_{i \in \mathrm{days}} \left(\theta_1 \cdot \mathrm{High\_Temperature}^{(i)} + \theta_2 - \mathrm{Peak\_Demand}^{(i)}\right)^2$$

$$\equiv \sum_{i \in \mathrm{days}} \left(\theta_1 \cdot x^{(i)} + \theta_2 - y^{(i)}\right)^2$$



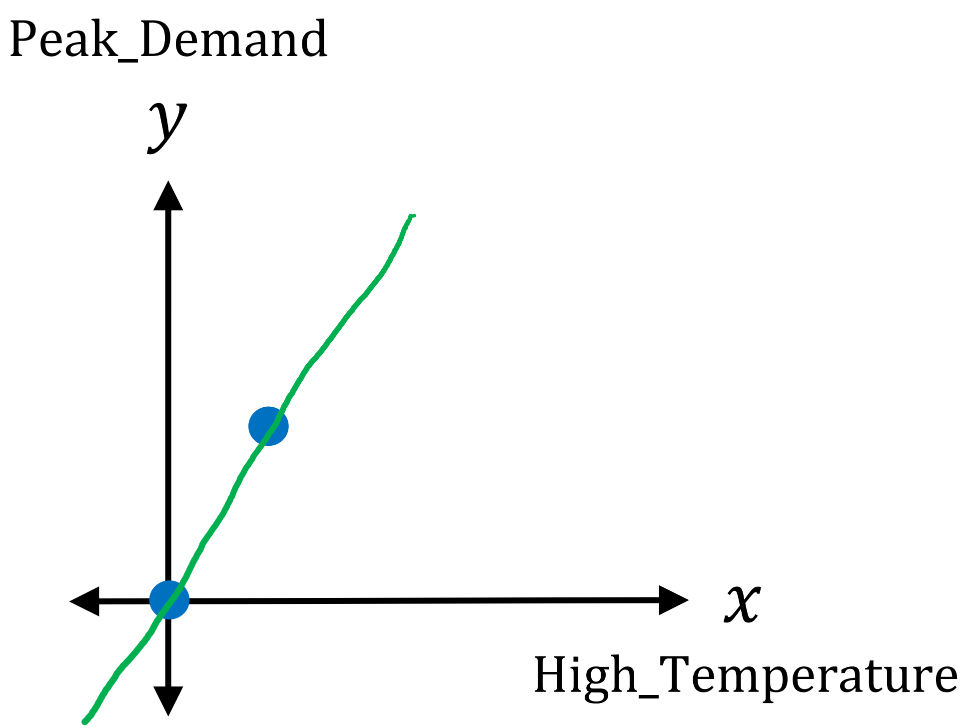


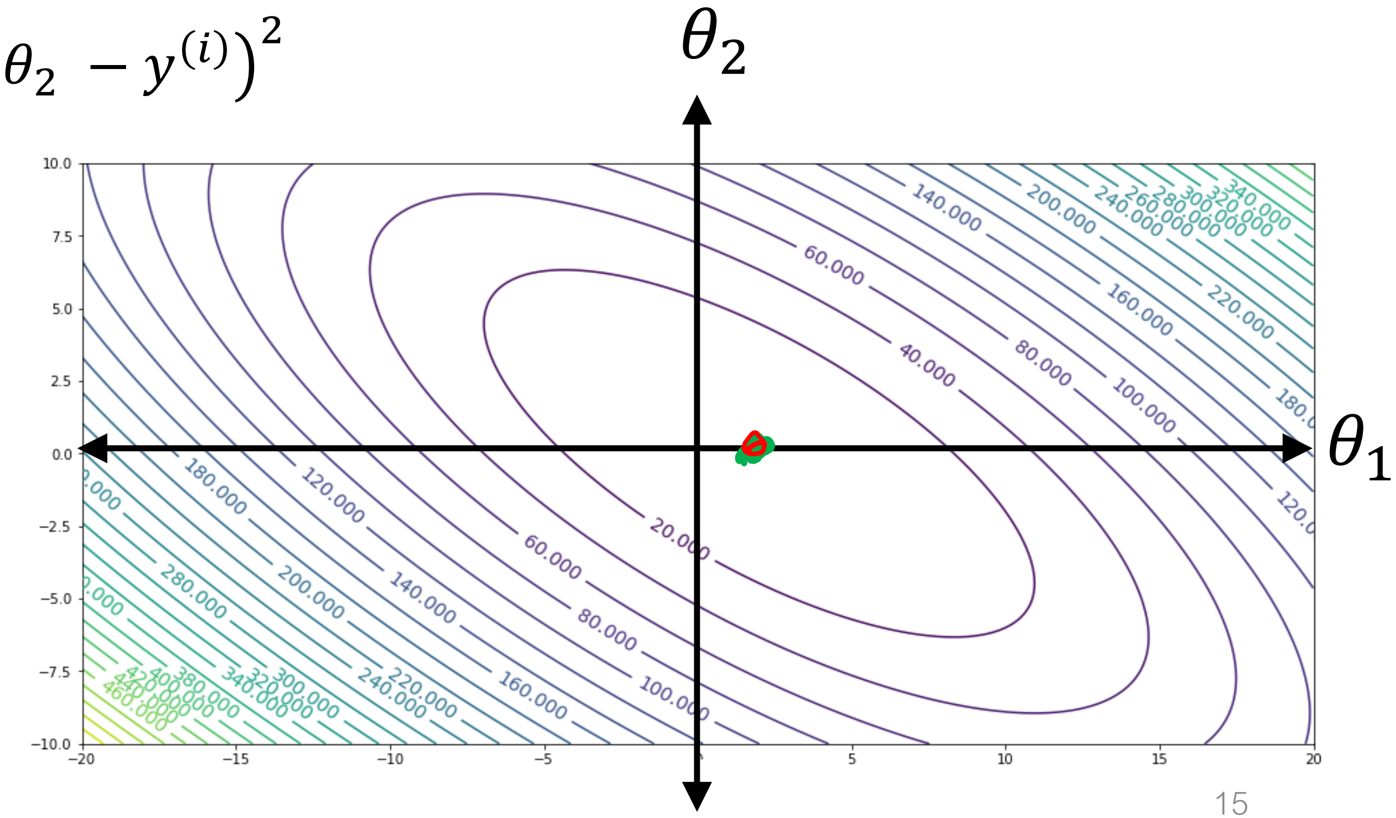
How do we find parameters?

How do we find the parameters θ_1, θ_2 that minimize the function

$$E(\theta) = \mathrm{E}(\theta_1, \theta_2) = \sum_{i \in \mathrm{days}} \left(\theta_1 \cdot \mathrm{High\_Temperature}^{(i)} + \theta_2 \ - \mathrm{Peak\_Demand}^{(i)}\right)^2$$

$$\equiv \sum_{i \in \mathrm{days}} \left(\theta_1 \cdot x^{(i)} + \theta_2 \ - y^{(i)}\right)^2$$



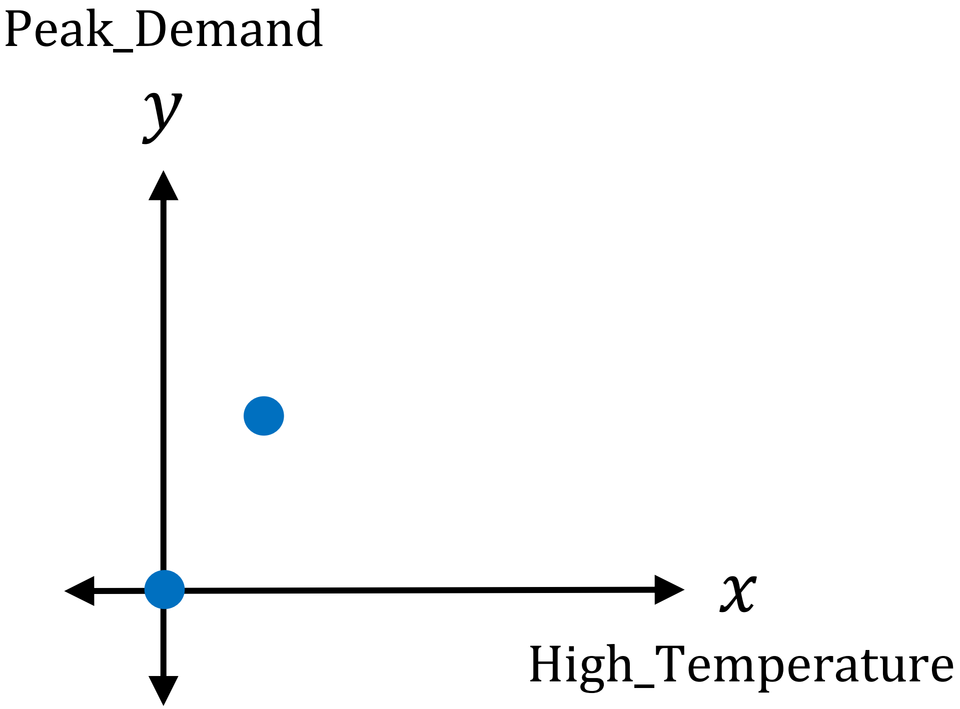


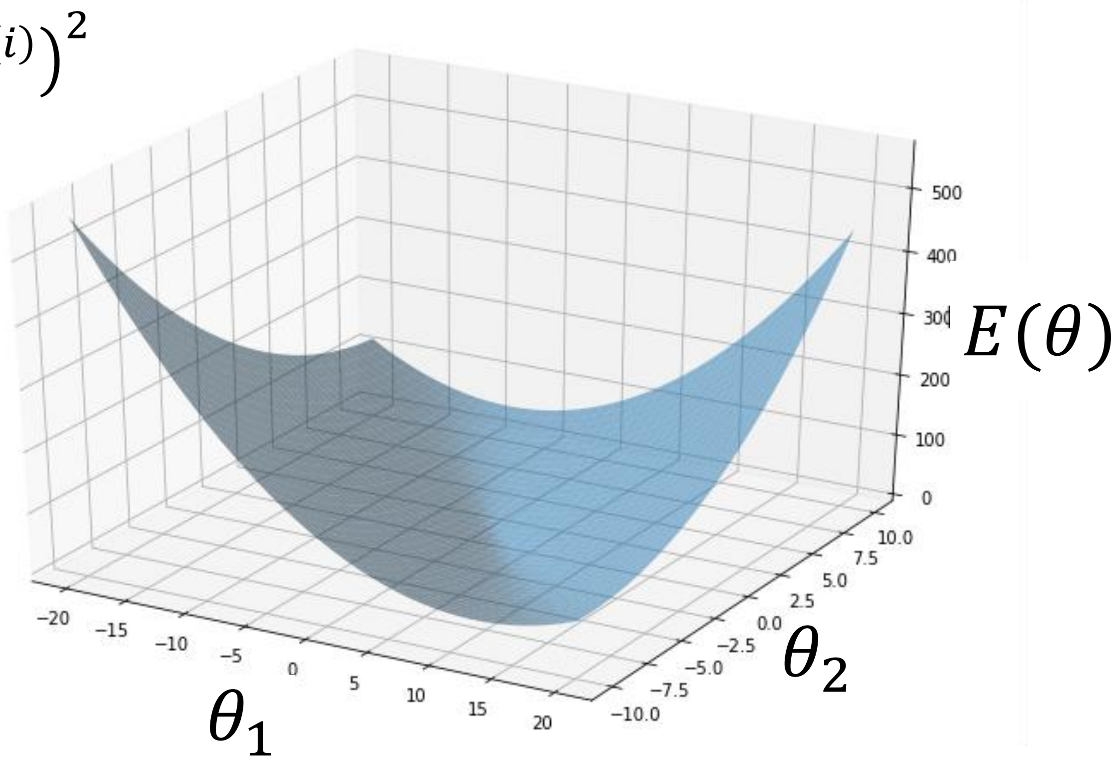
How do we find parameters?

How do we find the parameters θ_1, θ_2 that minimize the function

$$E(\theta) = \mathrm{E}(\theta_1, \theta_2) = \sum_{i \in \text{days}} \left(\theta_1 \cdot \text{High\_Temperature}^{(i)} + \theta_2 \; - \text{Peak\_Demand}^{(i)}\right)^2$$

$$\equiv \textstyle\sum_{i \in \text{days}} \left(\theta_1 \cdot x^{(i)} + \theta_2 \; - y^{(i)}\right)^2$$

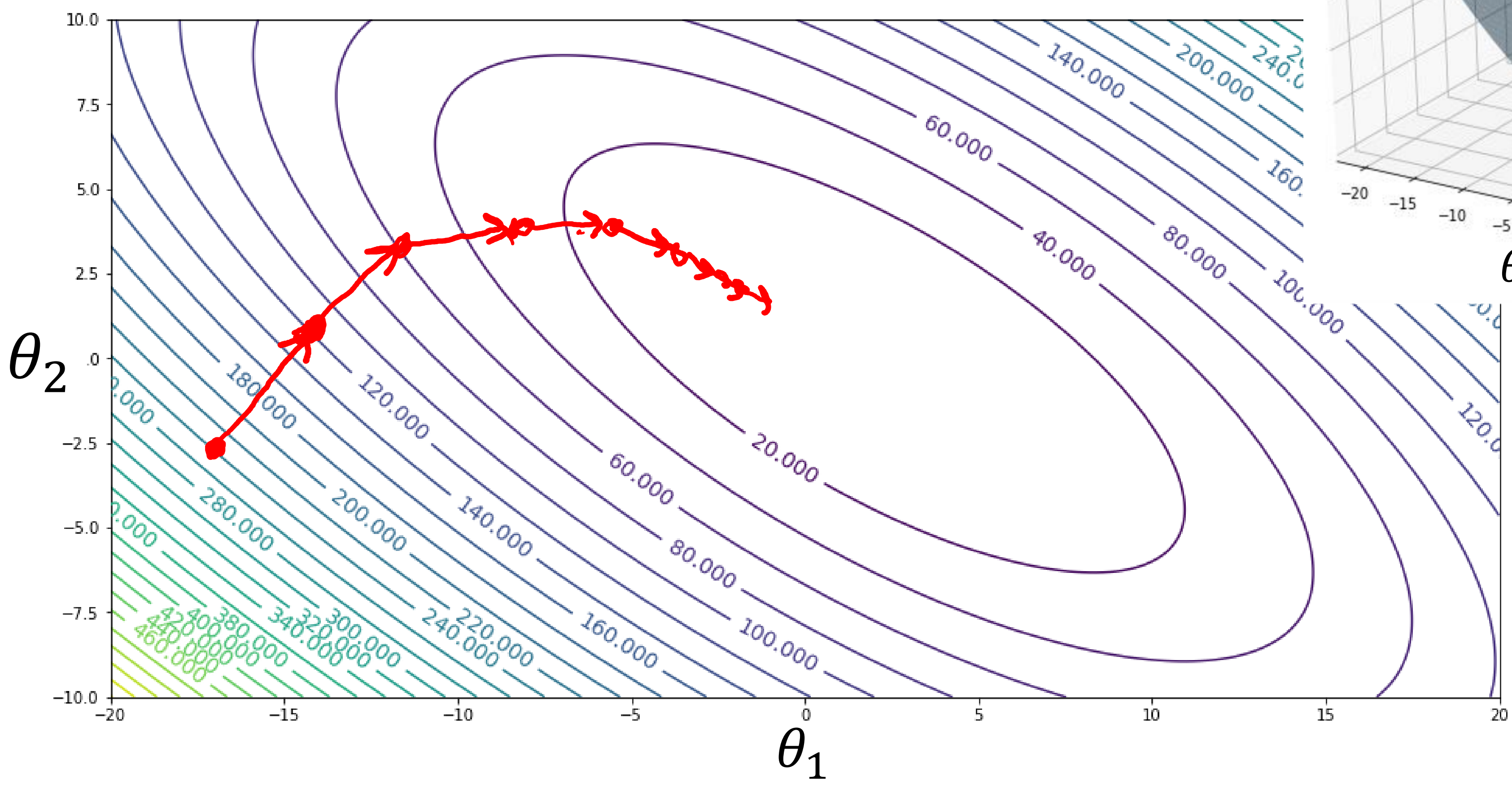


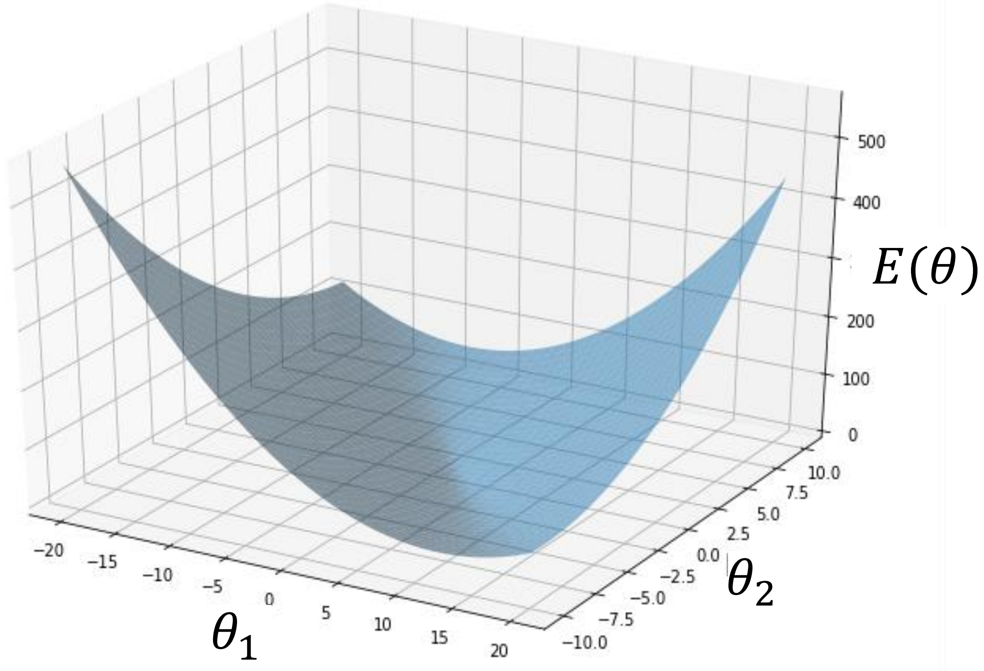


Gradient descent

How do we find the parameters θ_1, θ_2 that minimize:

$$E(\theta) = \mathrm{E}(\theta_1, \theta_2) = \sum_{i \in \text{days}} \left(\theta_1 \cdot x^{(i)} + \theta_2 - y^{(i)}\right)^2$$

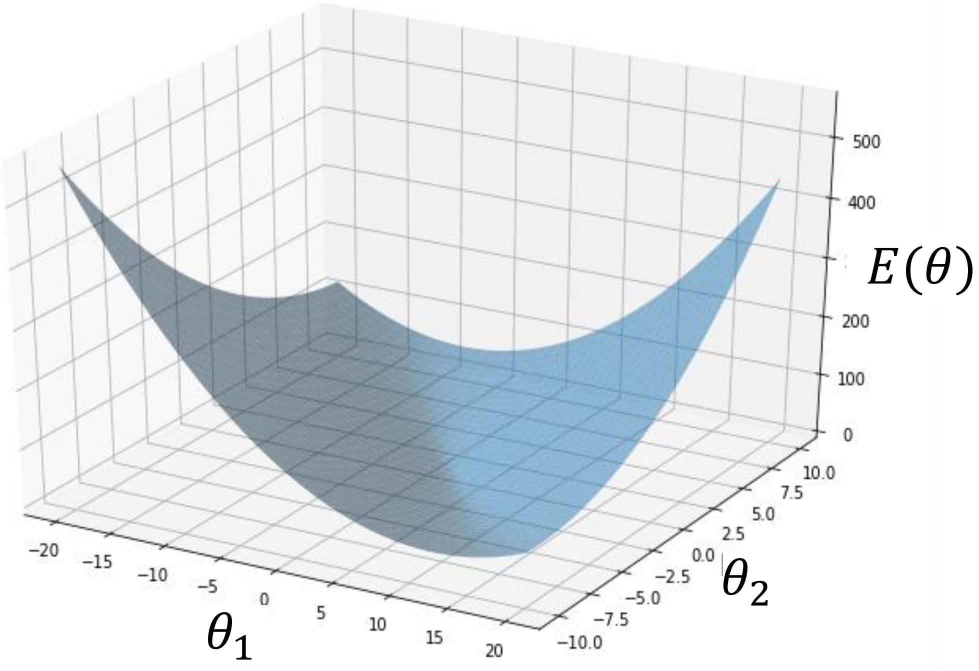


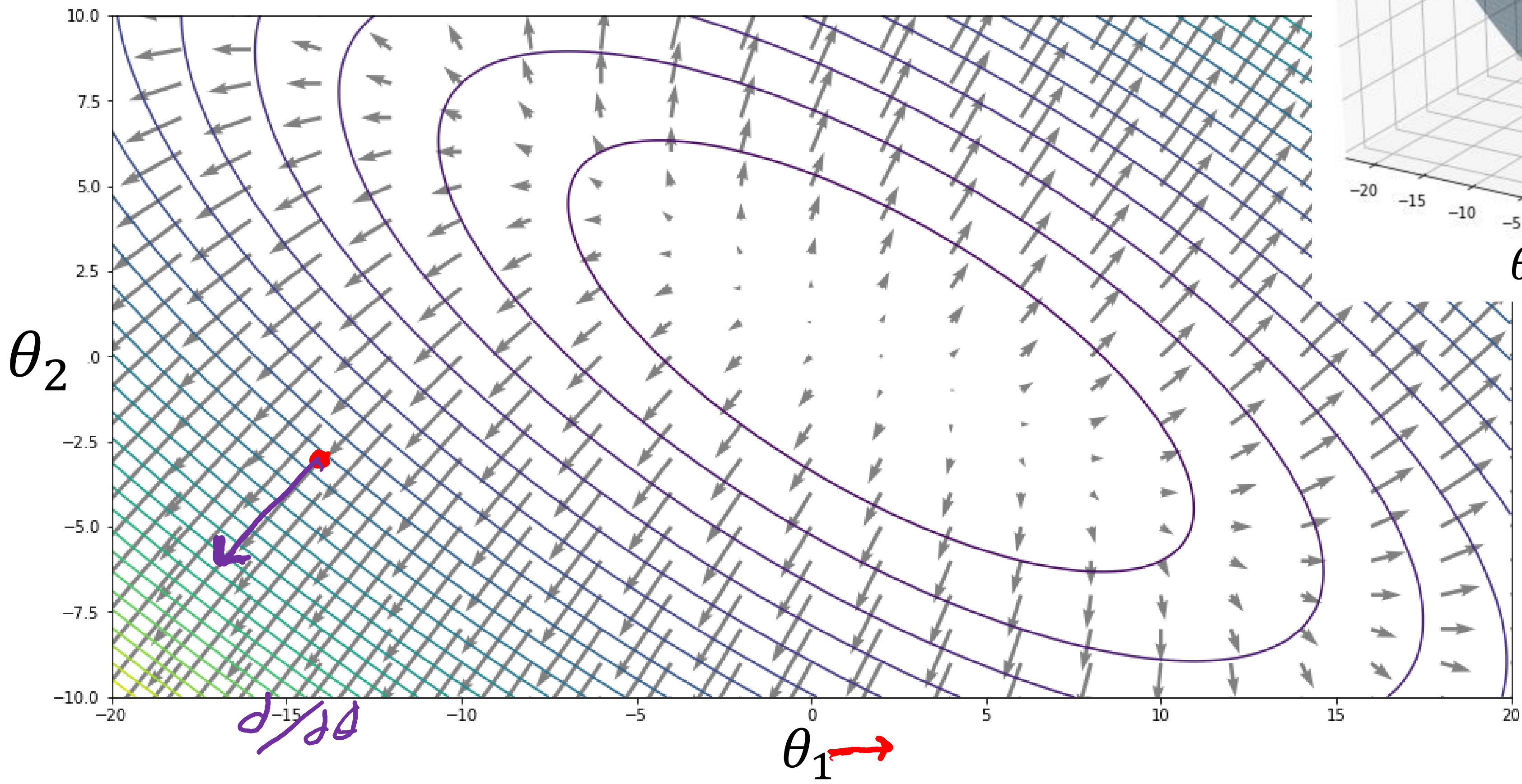


Gradient descent

How do we find the parameters θ_1, θ_2 that minimize:

$$E(\theta) = \mathrm{E}(\theta_1, \theta_2) = \sum_{i \in \text{days}} \left(\theta_1 \cdot x^{(i)} + \theta_2 - y^{(i)}\right)^2$$





Gradient descent

To find a good value of θ, we can repeatedly take steps in the direction of the negative derivatives for each value

Repeat:

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} E(\theta_1, \theta_2)$$

$$\theta_2 := \theta_2 - \alpha \frac{\partial}{\partial \theta_2} E(\theta_1, \theta_2)$$

where α is some small positive number called the *step size*

This is the *gradient decent algorithm*, the workhorse of modern machine learning

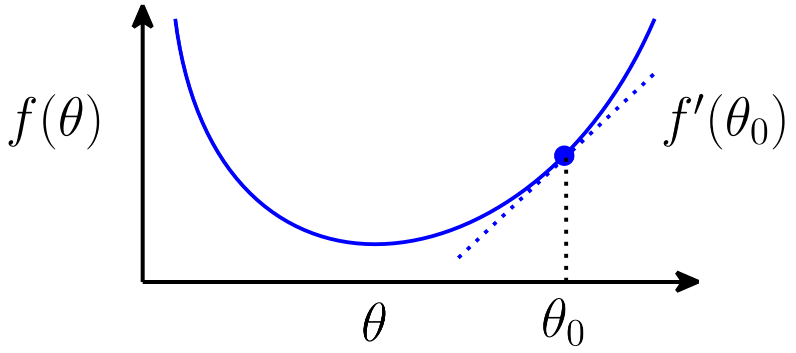
Computing gradients (partial derivatives)

How do we find the parameters θ_1, θ_2 that minimize the function

$$E(\theta) = \mathrm{E}(\theta_1, \theta_2) = \sum_{i \in \mathrm{days}} \left(\theta_1 \cdot \mathrm{High\_Temperature}^{(i)} + \theta_2 \ - \mathrm{Peak\_Demand}^{(i)}\right)^2$$

$$\equiv \sum_{i \in \mathrm{days}} \left(\theta_1 \cdot x^{(i)} + \theta_2 \ - y^{(i)}\right)^2$$

General idea: suppose we want to minimize some function $f(\theta)$



Derivative is slope of the function, so negative derivative points "downhill"

Calculus worksheet

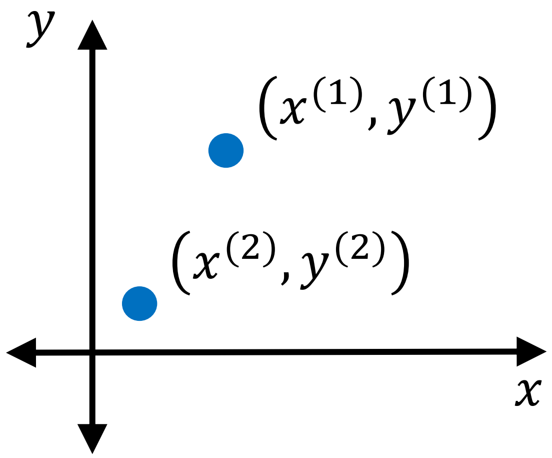
A. $f(x) = x^2 + 5x^3$ $\qquad \frac{df}{dx} = 2x + 15x^2$

B. $f(x) = (3 - 5x)^2$ $\qquad \frac{df}{dx} = 2(3-5x)(-5)$

C. $f(x,z) = 2x + 3z + 5x^2z$ $\qquad \frac{\partial f}{\partial z} = 0 + 3 + 5x^2$

D. $f(x,z) = 2x + 3z + 5x^2z$ $\qquad \frac{\partial f}{\partial x} = 2 + 0 + 10xz$

Computing the derivatives



Assume we just have m=2 points $x^{(1)}, y^{(1)}$ and $x^{(2)}, y^{(2)}$

$$\frac{\partial}{\partial \theta_1} E(\theta) = \frac{\partial}{\partial \theta_1} \sum_{i=1}^{m} \left(\theta_1 \cdot x^{(i)} + \theta_2 - y^{(i)}\right)^2$$

$$= \frac{\partial}{\partial \theta_1} \left[\left(\theta_1 x^{(1)} + \theta_2 - y^{(1)}\right)^2 + \left(\theta_1 x^{(2)} + \theta_2 - y^{(2)}\right)^2 \right]$$

$$= 2\left(\theta_1 x^{(1)} + \theta_2 - y^{(1)}\right) x^{(1)} + 2\left(\theta_1 x^{(2)} - \theta_2 - y^{(2)}\right) x^{(2)}$$

$$= \sum_{i=1}^{m} 2\left(\theta_1 x^{(i)} + \theta_2 - y^{(i)}\right) x^{(i)}$$

$$\frac{\partial}{\partial \theta_2} E(\theta) = \sum_{i=1}^{m} 2\left(\theta_1 x^{(i)} + \theta_2 - y^{(i)}\right) \cdot 1$$

Computing the derivatives

What are the derivatives of the error function with respect to each parameter θ_1 and θ_2?

$$\frac{\partial}{\partial \theta_1} E(\theta) = \frac{\partial}{\partial \theta_1} \sum_{i \in \text{days}} \left(\theta_1 \cdot x^{(i)} + \theta_2 - y^{(i)}\right)^2$$

$$= \sum_{i \in \text{days}} \frac{\partial}{\partial \theta_1} \left(\theta_1 \cdot x^{(i)} + \theta_2 - y^{(i)}\right)^2$$

$$= \sum_{i \in \text{days}} 2\left(\theta_1 \cdot x^{(i)} + \theta_2 - y^{(i)}\right) \cdot \frac{\partial}{\partial \theta_1} \theta_1 \cdot x^{(i)}$$

$$= \sum_{i \in \text{days}} 2\left(\theta_1 \cdot x^{(i)} + \theta_2 - y^{(i)}\right) \cdot x^{(i)}$$

$$\frac{\partial}{\partial \theta_2} E(\theta) = \sum_{i \in \text{days}} 2\left(\theta_1 \cdot x^{(i)} + \theta_2 - y^{(i)}\right)$$

Gradient descent

To find a good value of θ, we can repeatedly take steps in the direction of the negative derivatives for each value

∇E

Repeat:

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} E(\theta_1, \theta_2)$$

$$\theta_2 := \theta_2 - \alpha \frac{\partial}{\partial \theta_2} E(\theta_1, \theta_2)$$

$$\theta := \theta - \alpha \nabla_\theta E(\theta)$$

where α is some small positive number called the *step size*

This is the *gradient decent algorithm*, the workhorse of modern machine learning

Finding the best θ

To find a good value of θ, we can repeatedly take steps in the direction of the negative derivatives for each value

Repeat:

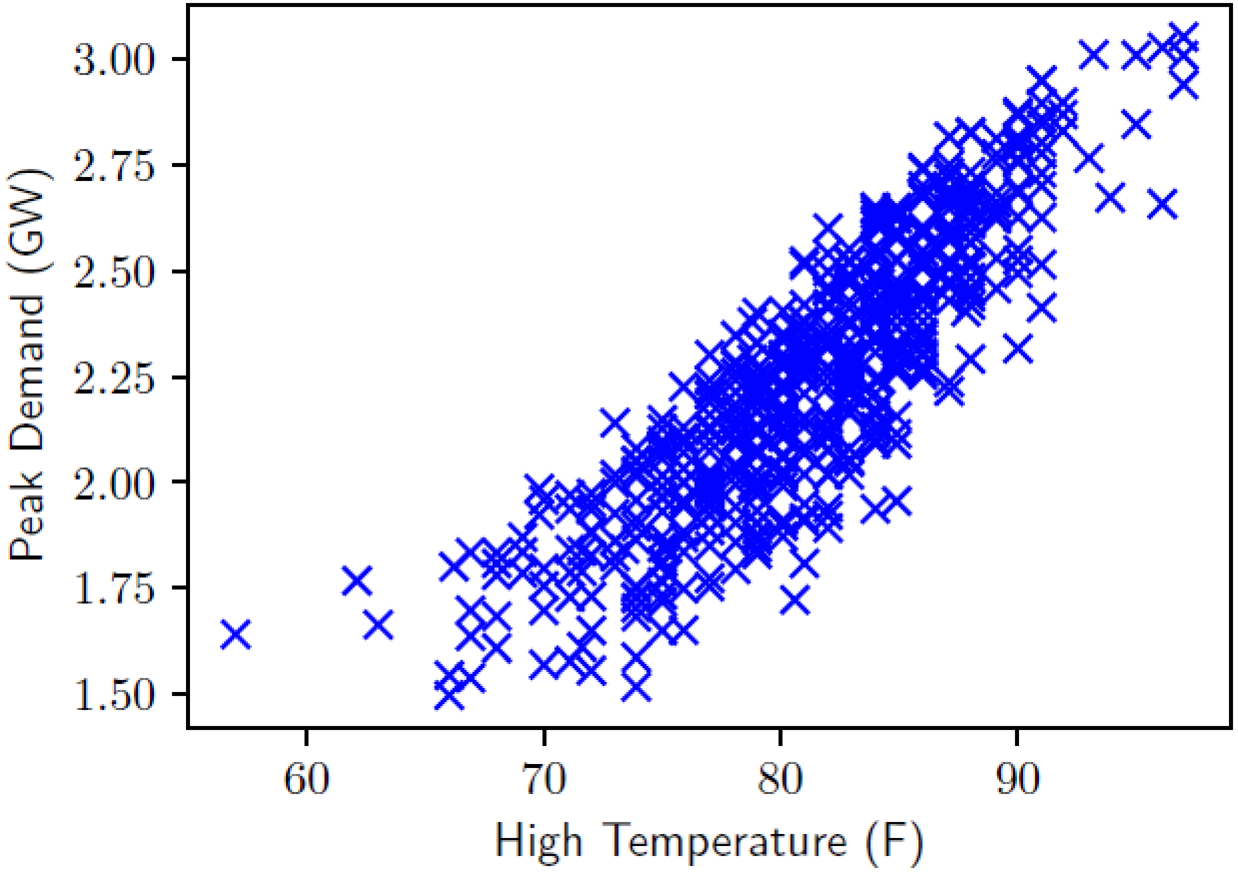
$$\theta_1 \coloneqq \theta_1 - \alpha \sum_{i \in \text{days}} 2\left(\theta_1 \cdot x^{(i)} + \theta_2 - y^{(i)}\right) \cdot x^{(i)}$$

$$\theta_2 \coloneqq \theta_2 - \alpha \sum_{i \in \text{days}} 2\left(\theta_1 \cdot x^{(i)} + \theta_2 - y^{(i)}\right)$$

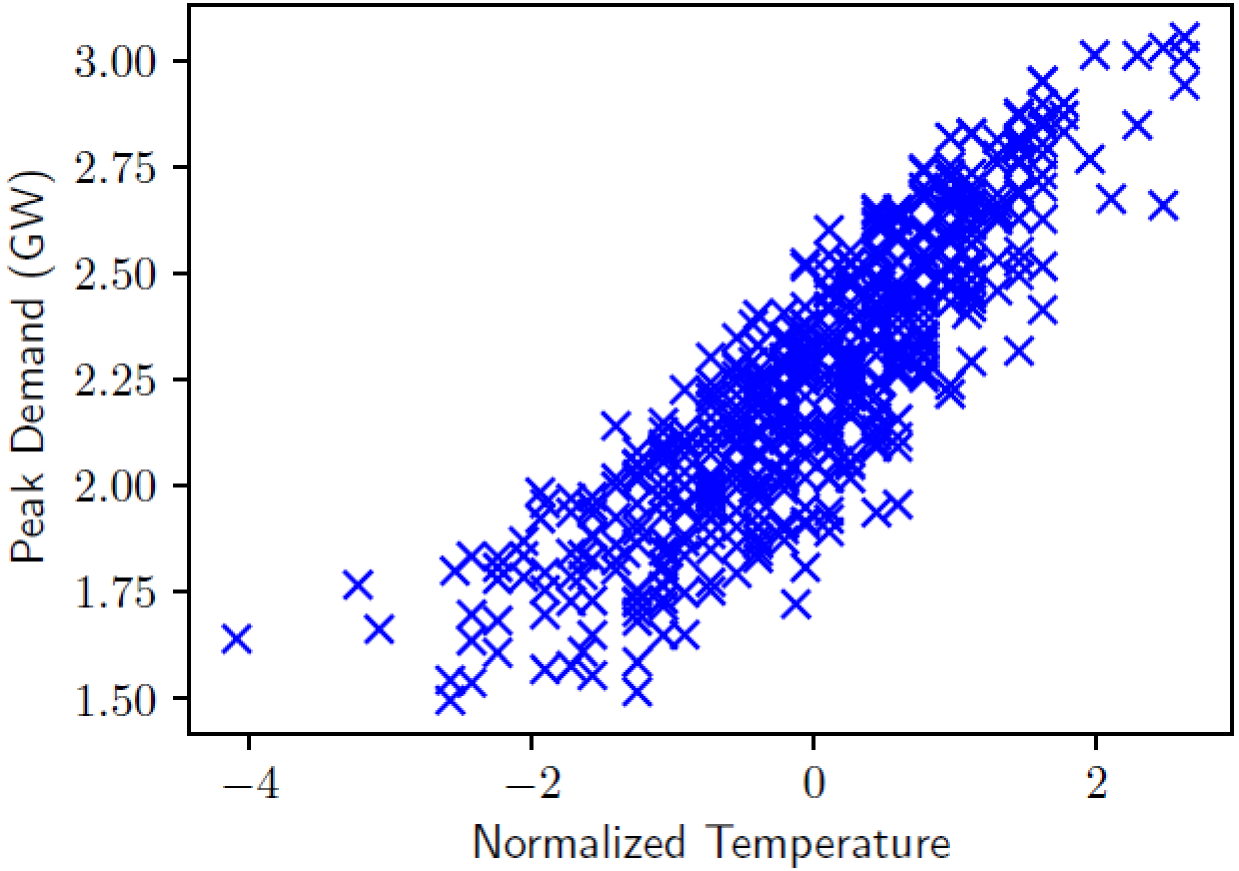
where α is some small positive number called the *step size*

This is the *gradient decent algorithm*, the workhorse of modern machine learning

Gradient descent

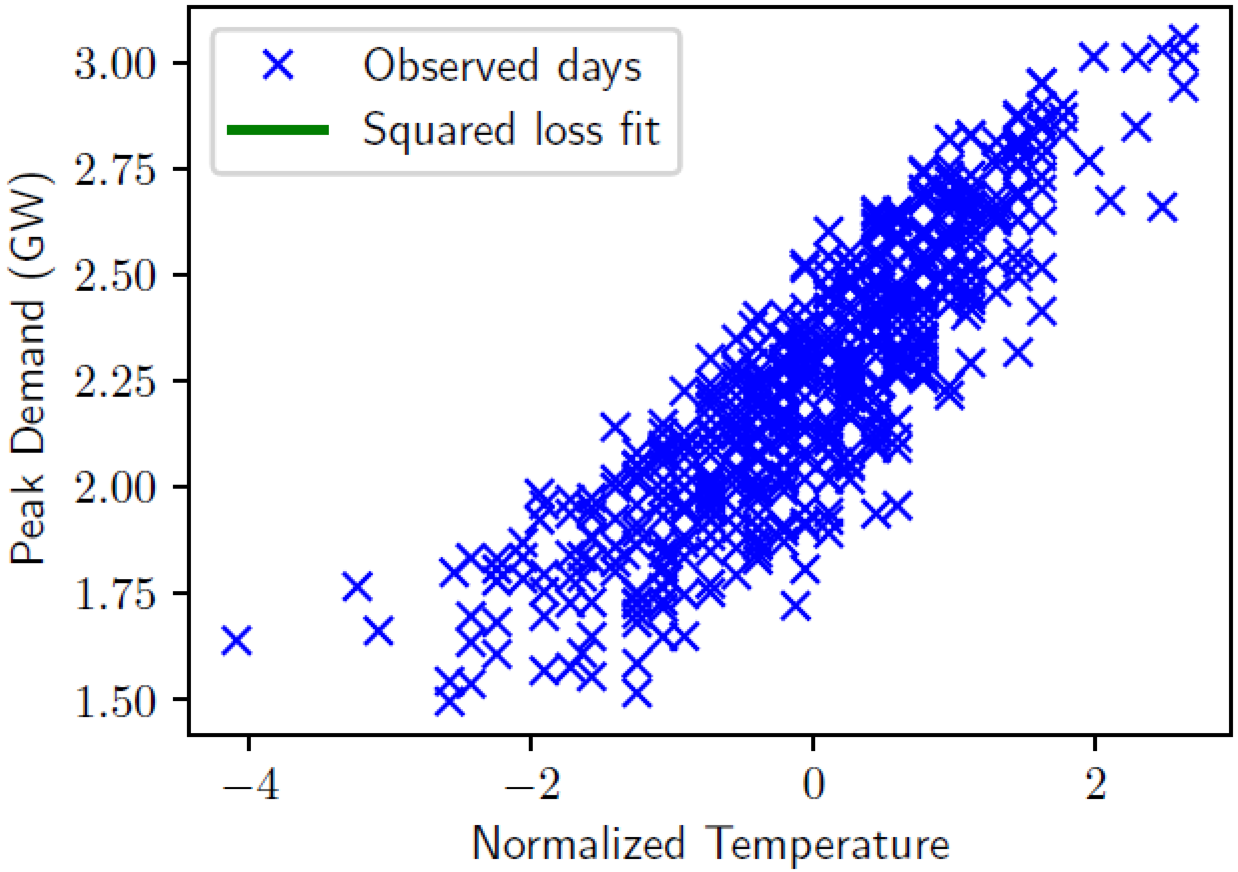


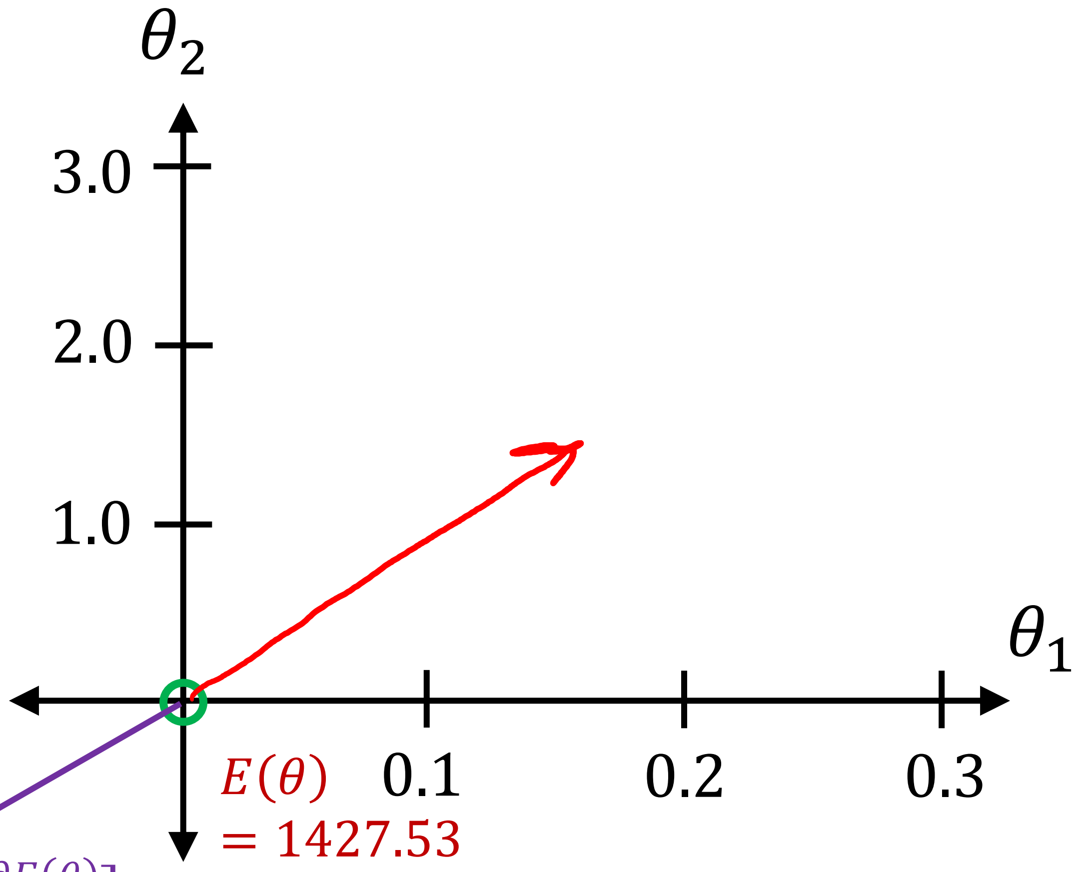
Gradient descent



Normalize input by subtracting the mean and dividing by the standard deviation

Gradient descent – Iteration 1





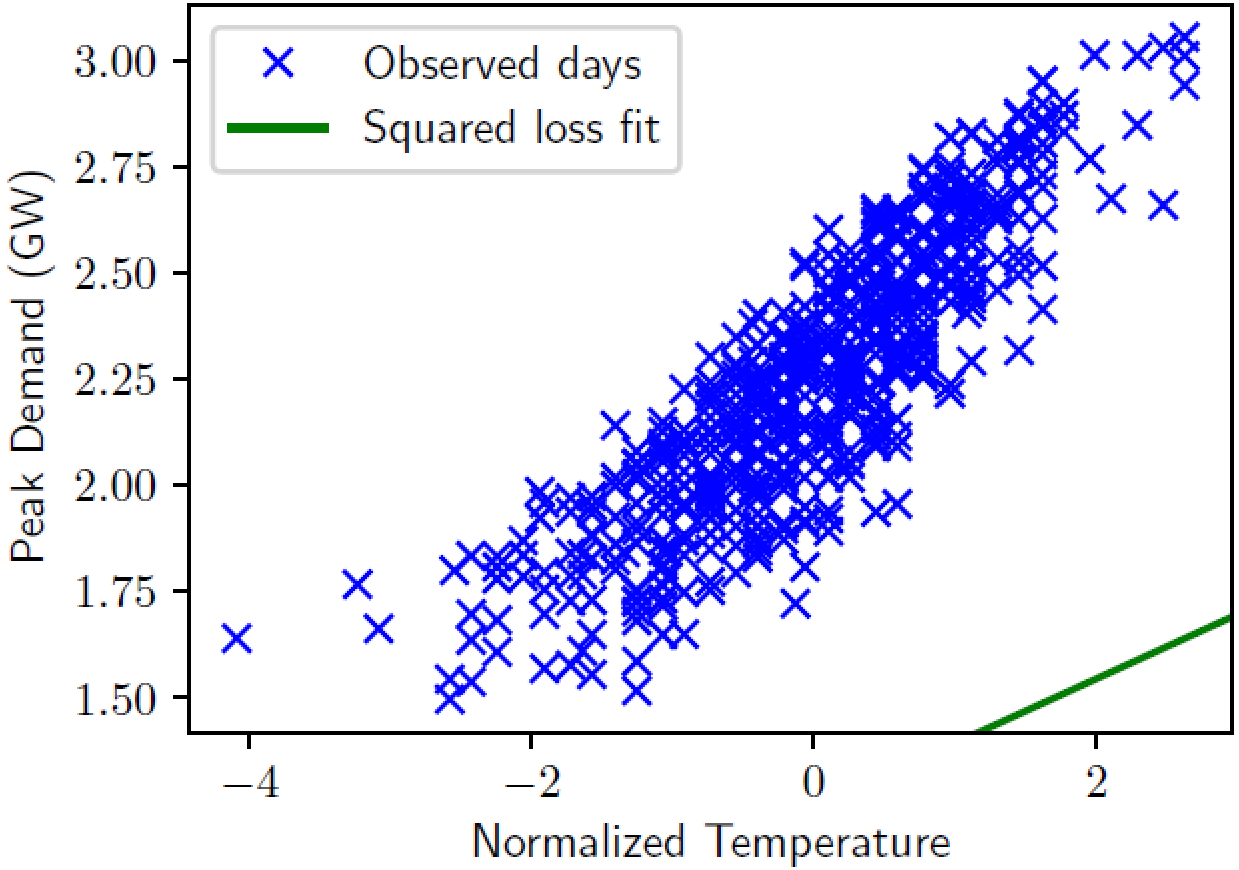
$\theta = (0.00, 0.00)$

$E(\theta) = 1427.53$

$(\frac{\partial E(\theta)}{\partial \theta_1}, \frac{\partial E(\theta)}{\partial \theta_2}) = (-151.20, -1243.10)$

$$\alpha \begin{bmatrix} \frac{\partial E(\theta)}{\partial \theta_2} \\ \frac{\partial E(\theta)}{\partial \theta_2} \end{bmatrix} = 0.001 \begin{bmatrix} -151.20 \\ -1243.10 \end{bmatrix}$$

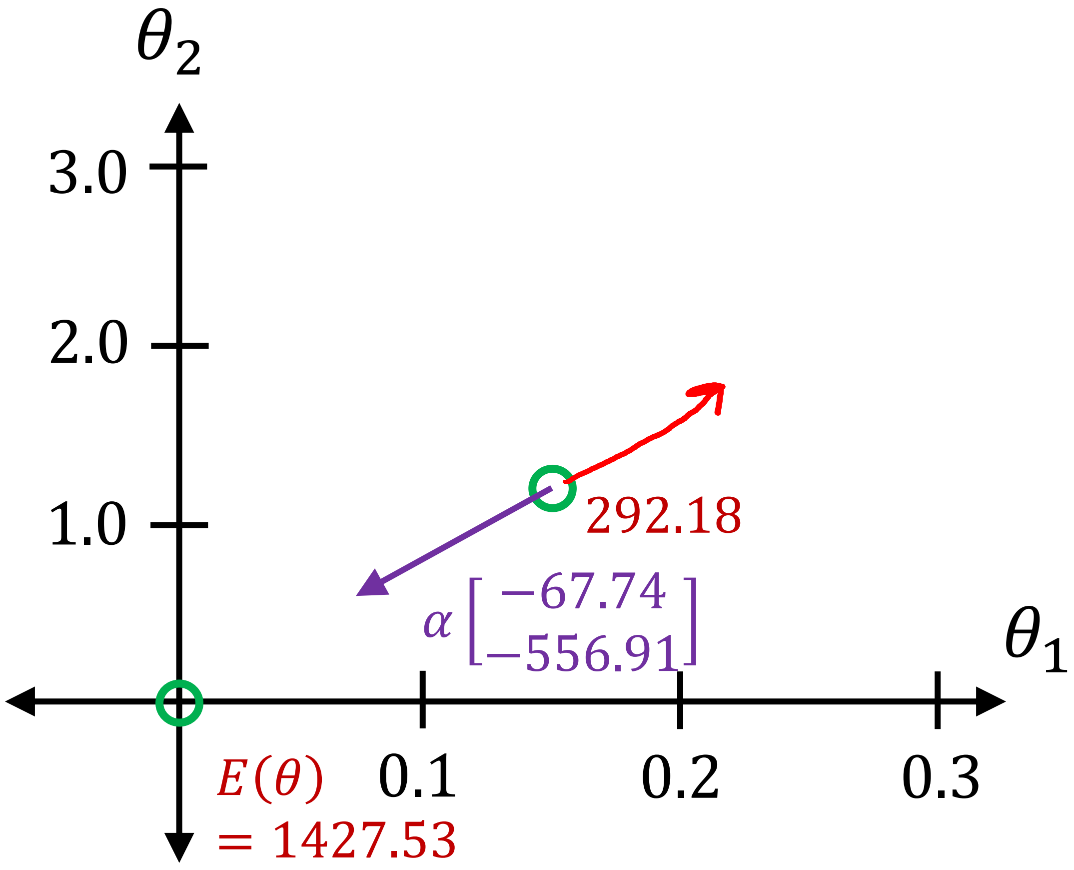
Gradient descent – Iteration 2



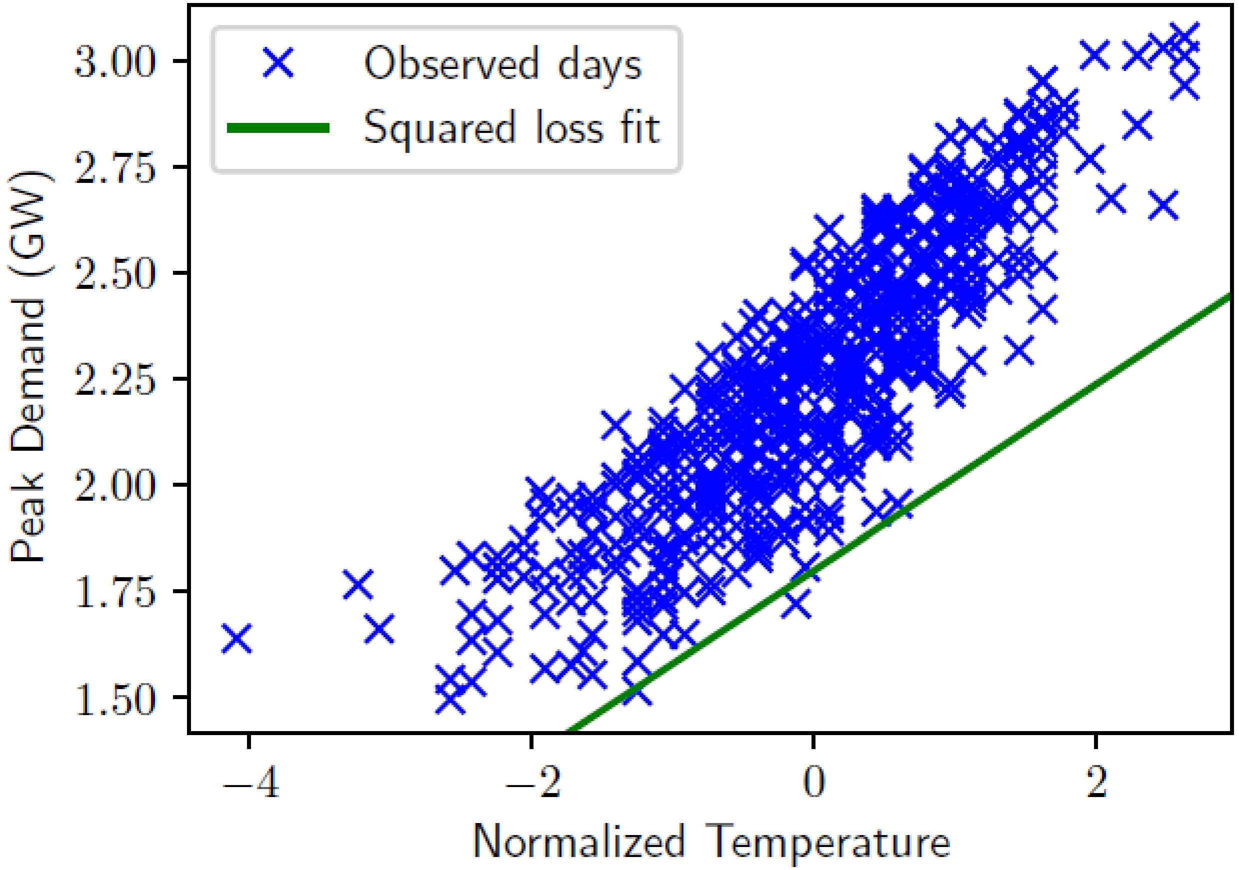
$\theta = (0.15, 1.24)$

$E(\theta) = 292.18$

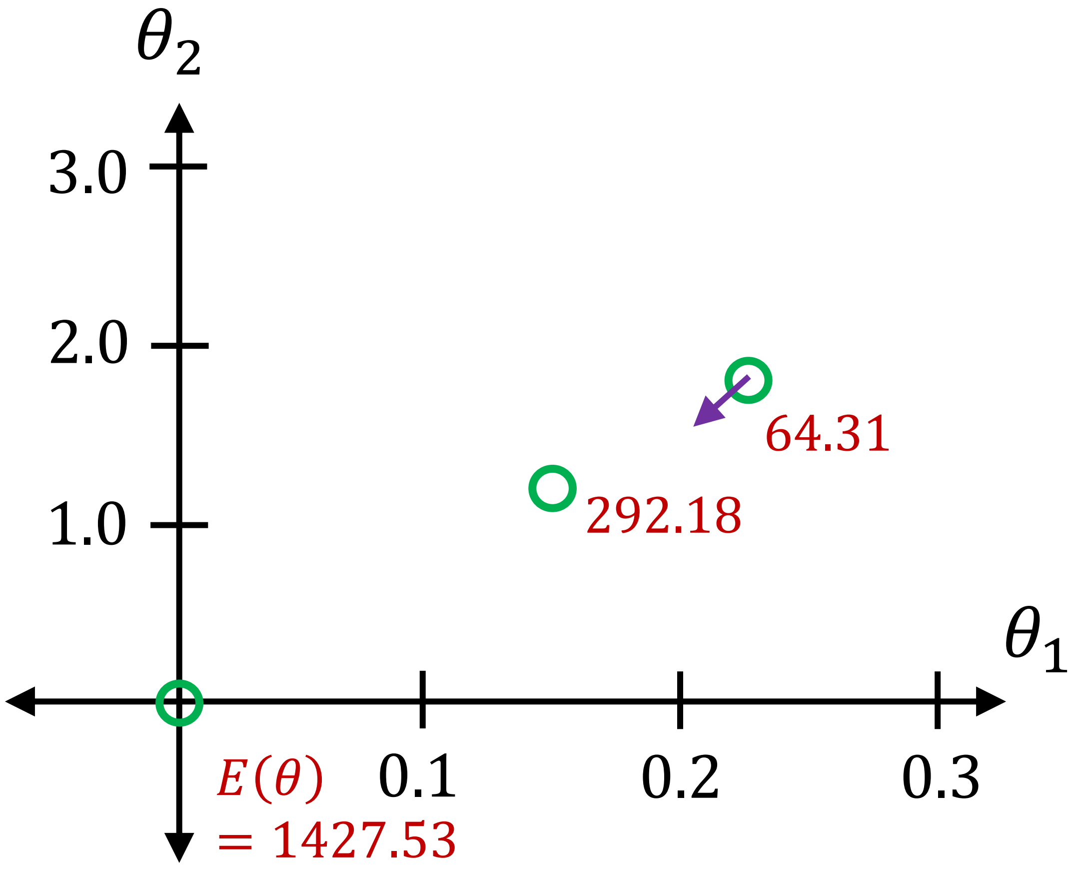
$(\frac{\partial E(\theta)}{\partial \theta_1}, \frac{\partial E(\theta)}{\partial \theta_2}) = (-67.74, -556.91)$



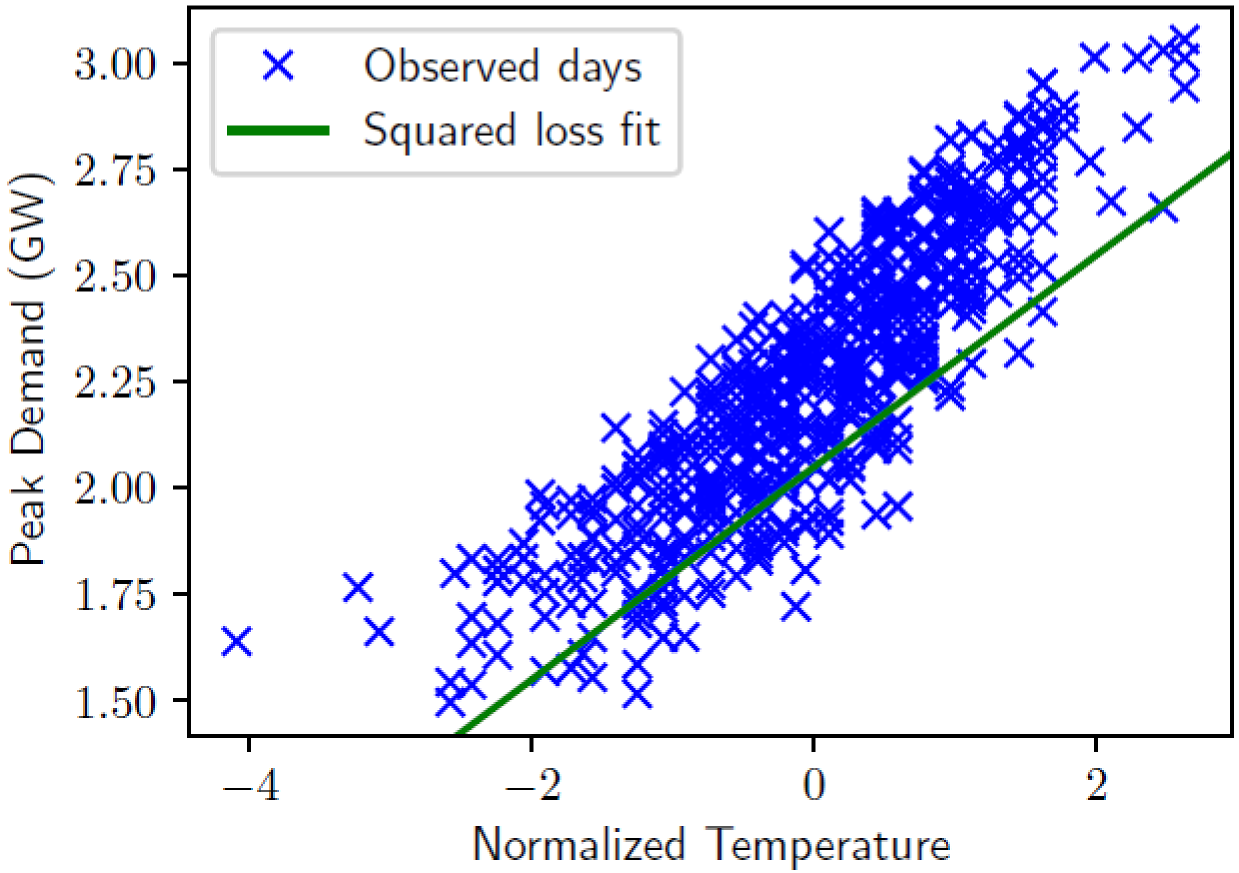
Gradient descent – Iteration 3



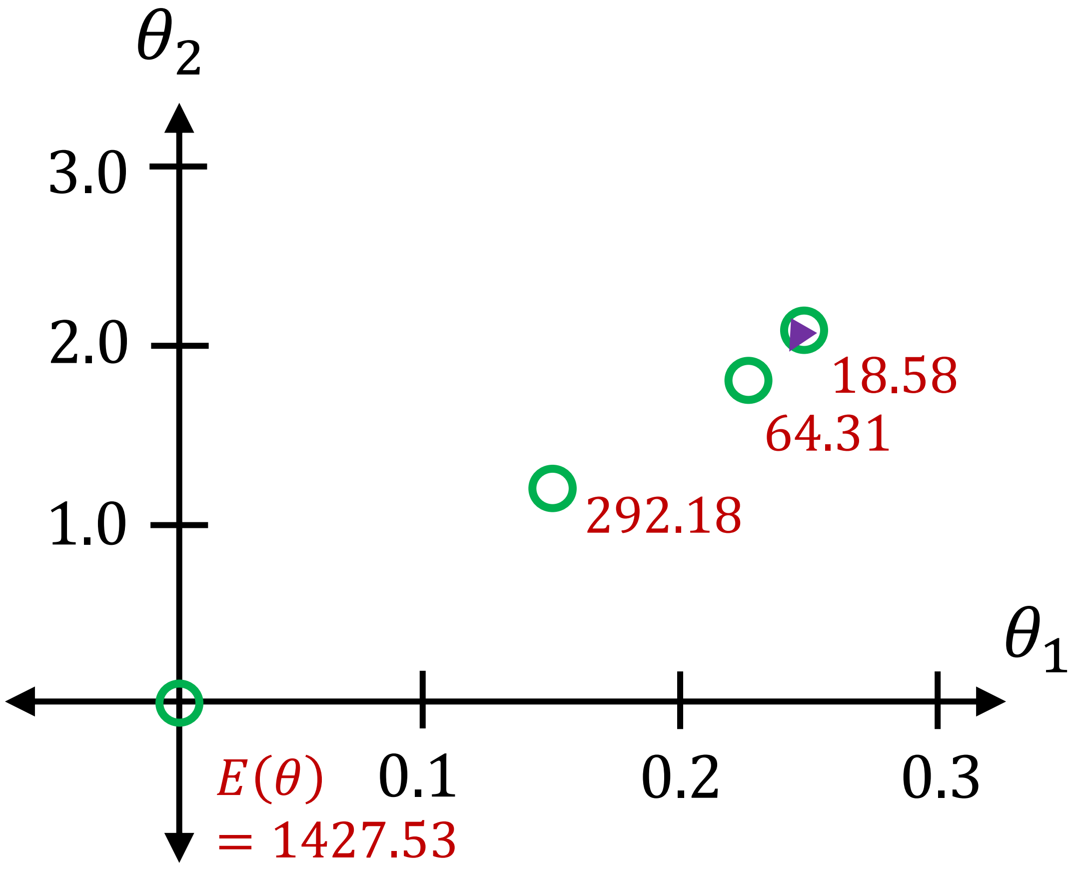
$\theta = (0.22, 1.80)$
$E(\theta) = 64.31$
$(\frac{\partial E(\theta)}{\partial \theta_1}, \frac{\partial E(\theta)}{\partial \theta_2}) = (-30.35, -249.50)$



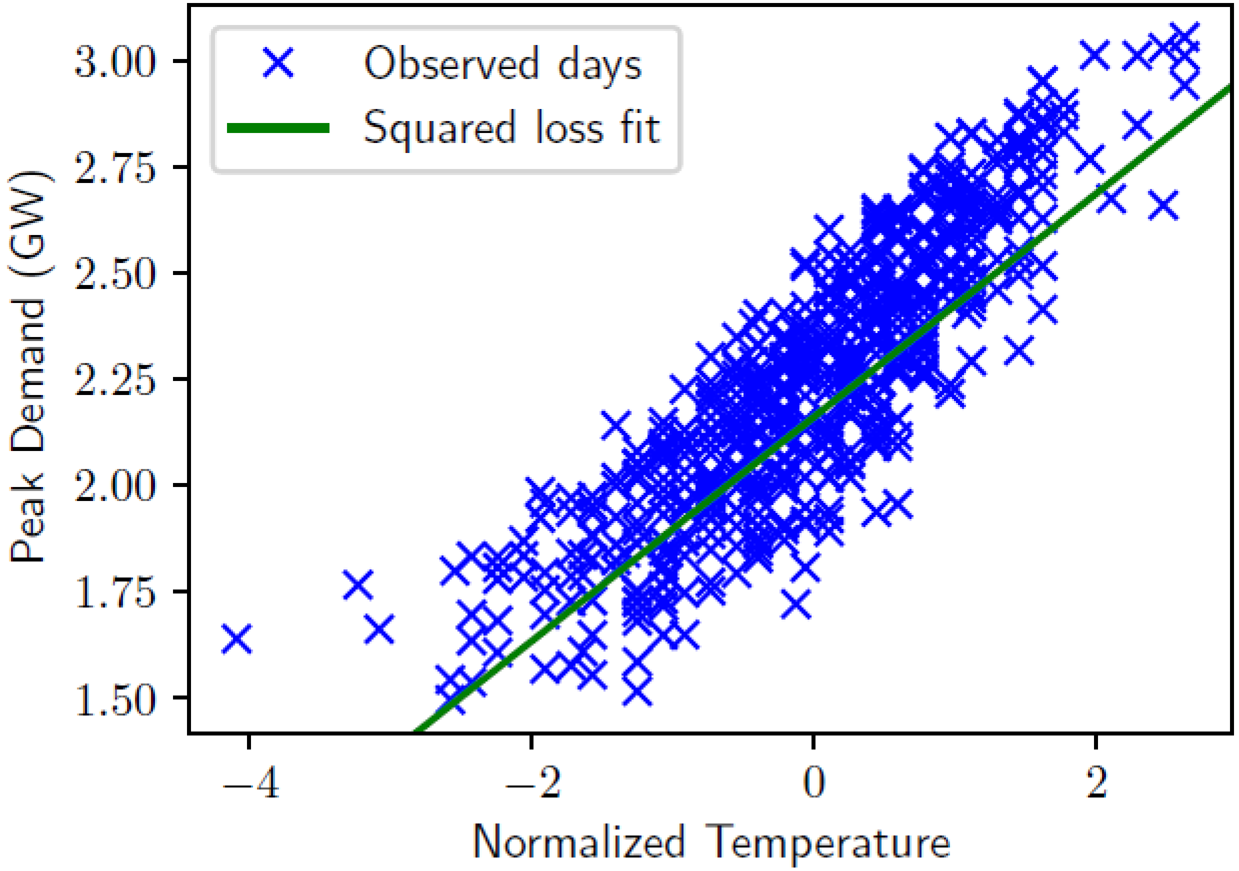
Gradient descent – Iteration 4



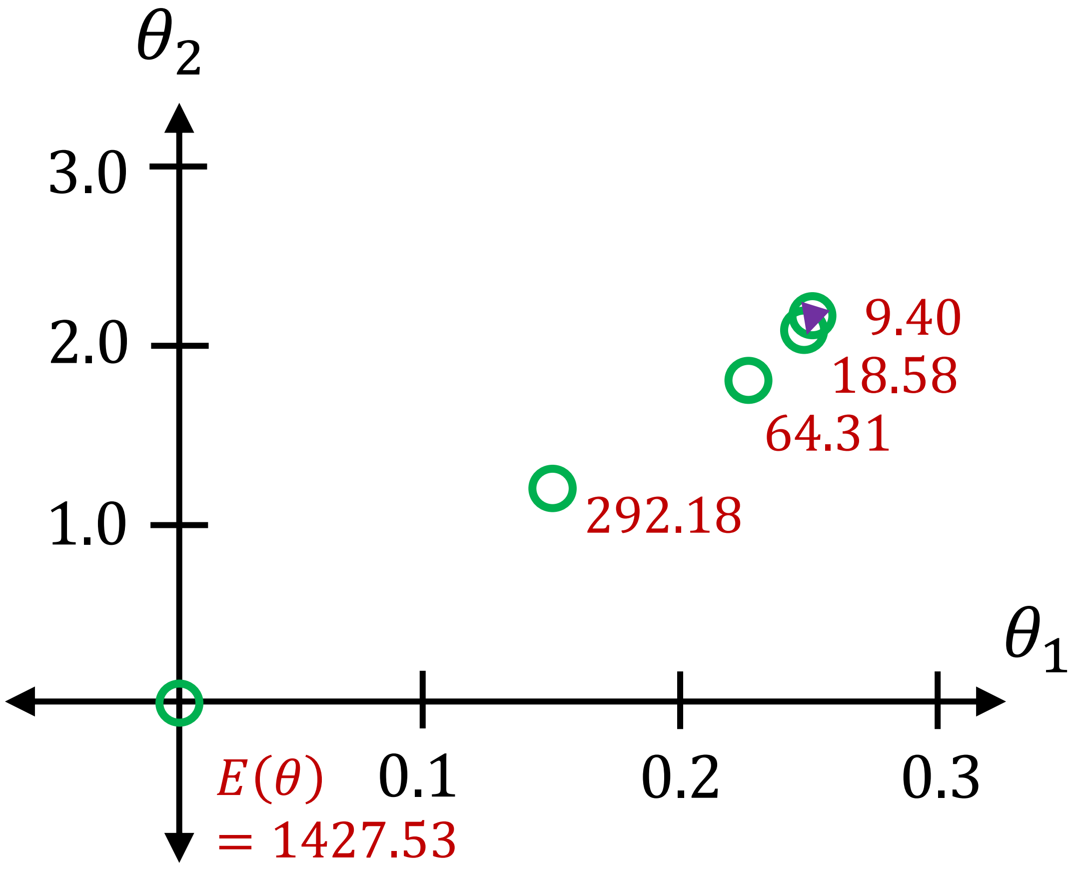
$\theta = (0.25, 2.05)$
$E(\theta) = 18.58$
$(\frac{\partial E(\theta)}{\partial \theta_1}, \frac{\partial E(\theta)}{\partial \theta_2}) = (-13.60, -111.77)$



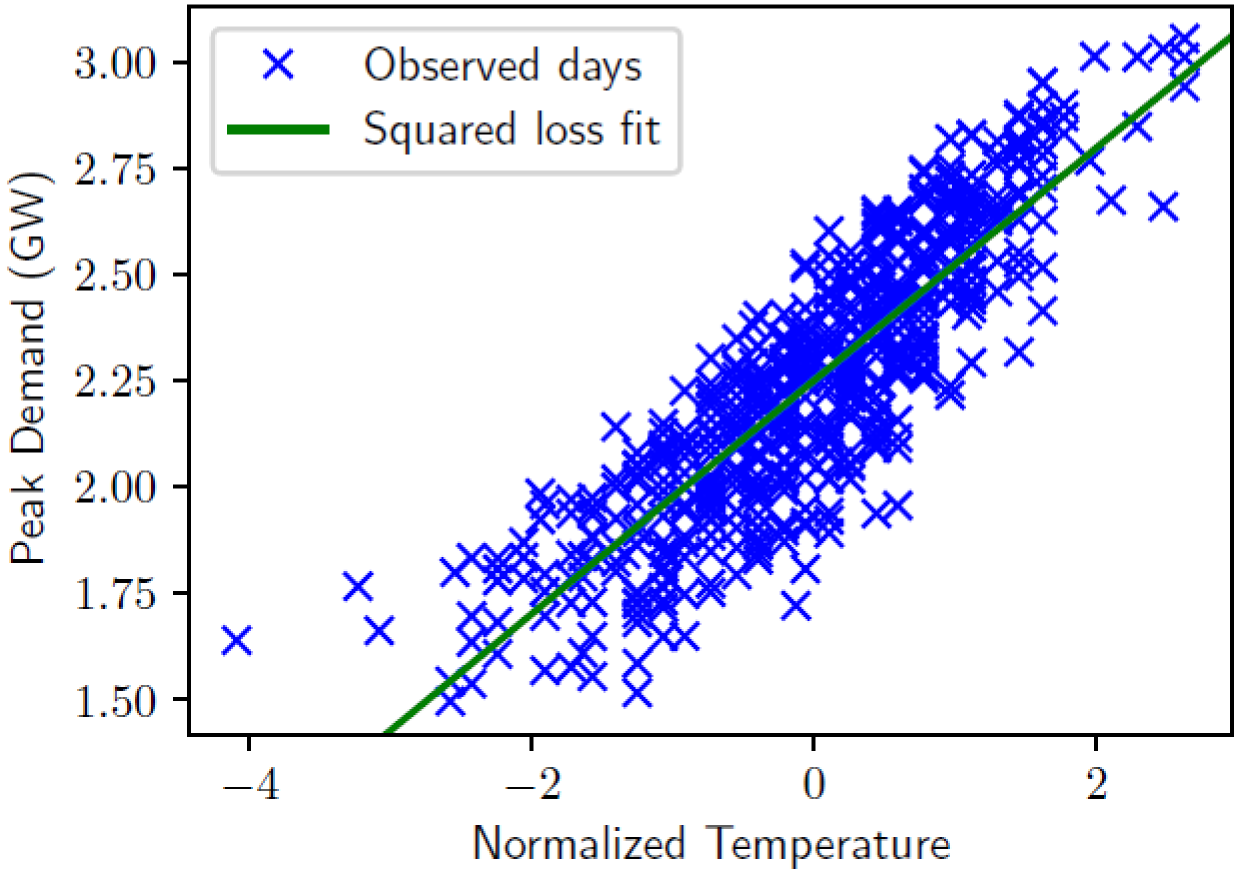
Gradient descent – Iteration 5



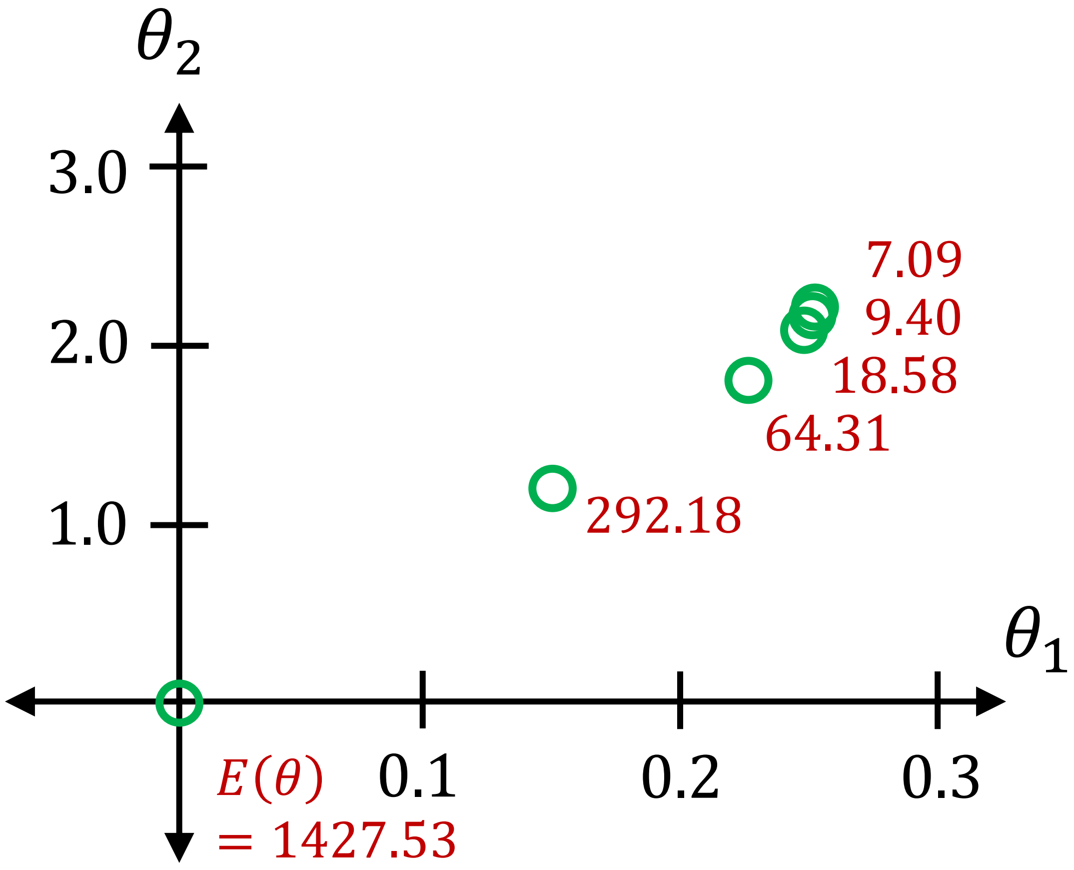
$\theta = (0.26, 2.16)$
$E(\theta) = 9.40$
$(\frac{\partial E(\theta)}{\partial \theta_1}, \frac{\partial E(\theta)}{\partial \theta_2}) = (-6.09, -50.07)$



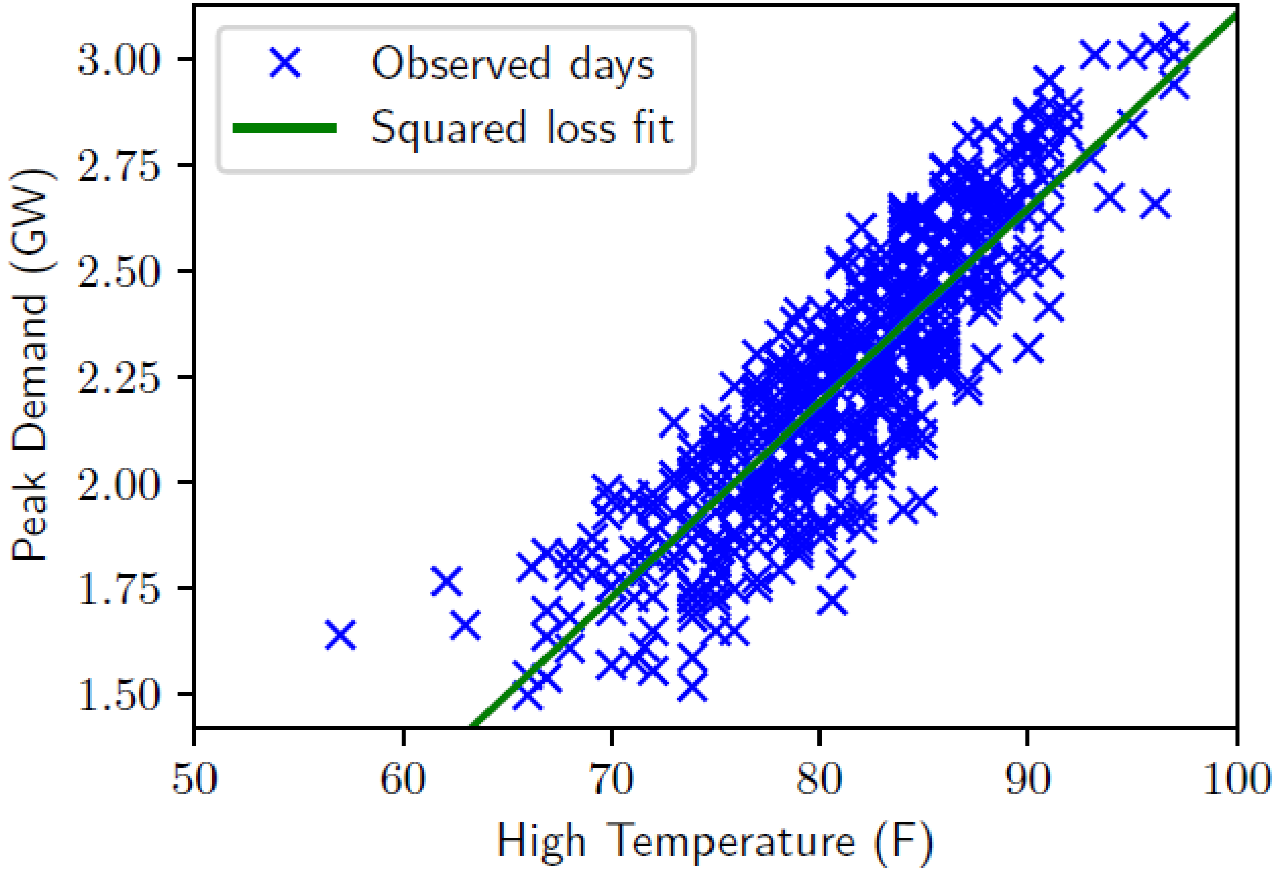
Gradient descent – Iteration 10



$\theta = (0.27, 2.25)$
$E(\theta) = 7.09$
$(\frac{\partial E(\theta)}{\partial \theta_1}, \frac{\partial E(\theta)}{\partial \theta_2}) = (-0.11, -0.90)$



Fitted line in “original” coordinates

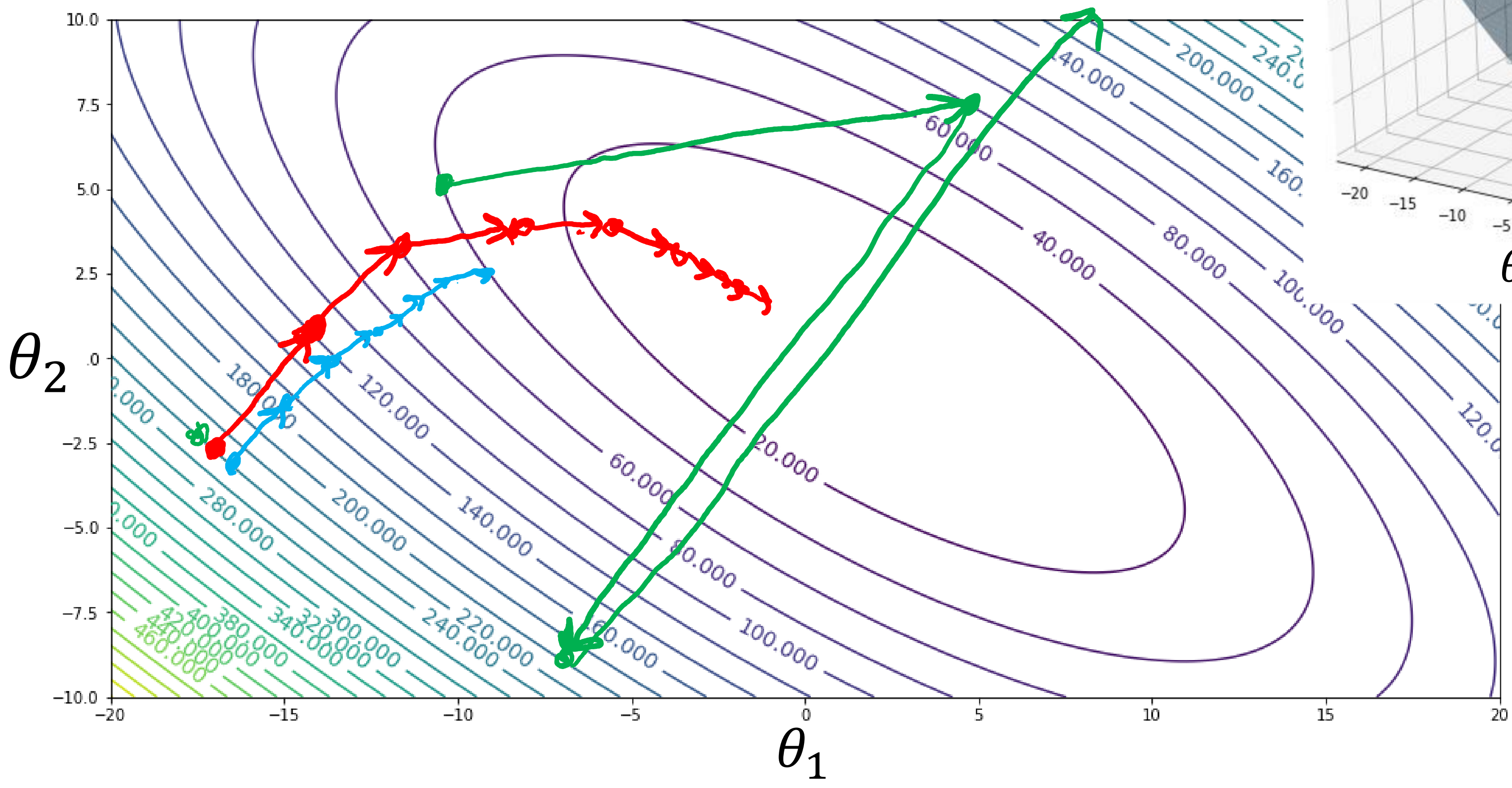


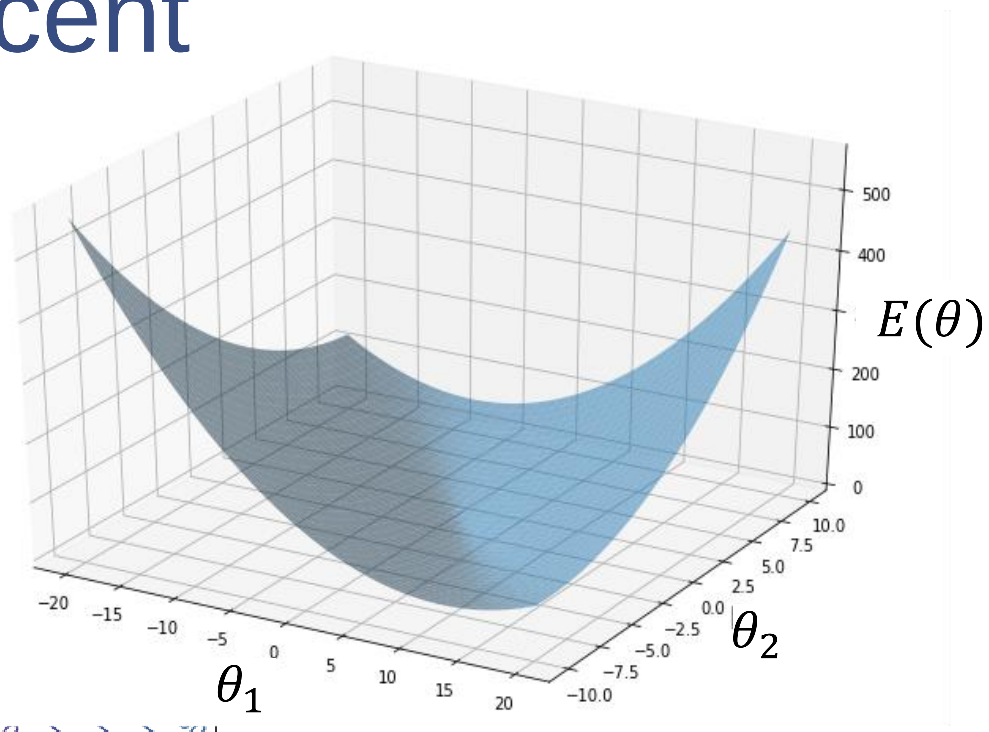
Important note: requires that we also rescale θ when un-normalizing

Gradient descent

How do we find the parameters θ_1, θ_2 that minimize:

$$E(\theta) = \mathrm{E}(\theta_1, \theta_2) = \sum_{i \in \text{days}} \left(\theta_1 \cdot x^{(i)} + \theta_2 - y^{(i)}\right)^2$$





Extensions

What if we want to add additional features, e.g. day of week, instead of just temperature?

What if we want to use a different loss function instead of squared error (i.e., absolute error)?

What if we want to use a non-linear prediction instead of a linear one?

We can easily reason about all these things by adopting some additional notation…

Outline

Least squares regression: a simple example

Machine learning notation

Linear regression revisited

Matrix/vector notation and analytic solutions

Implementing linear regression

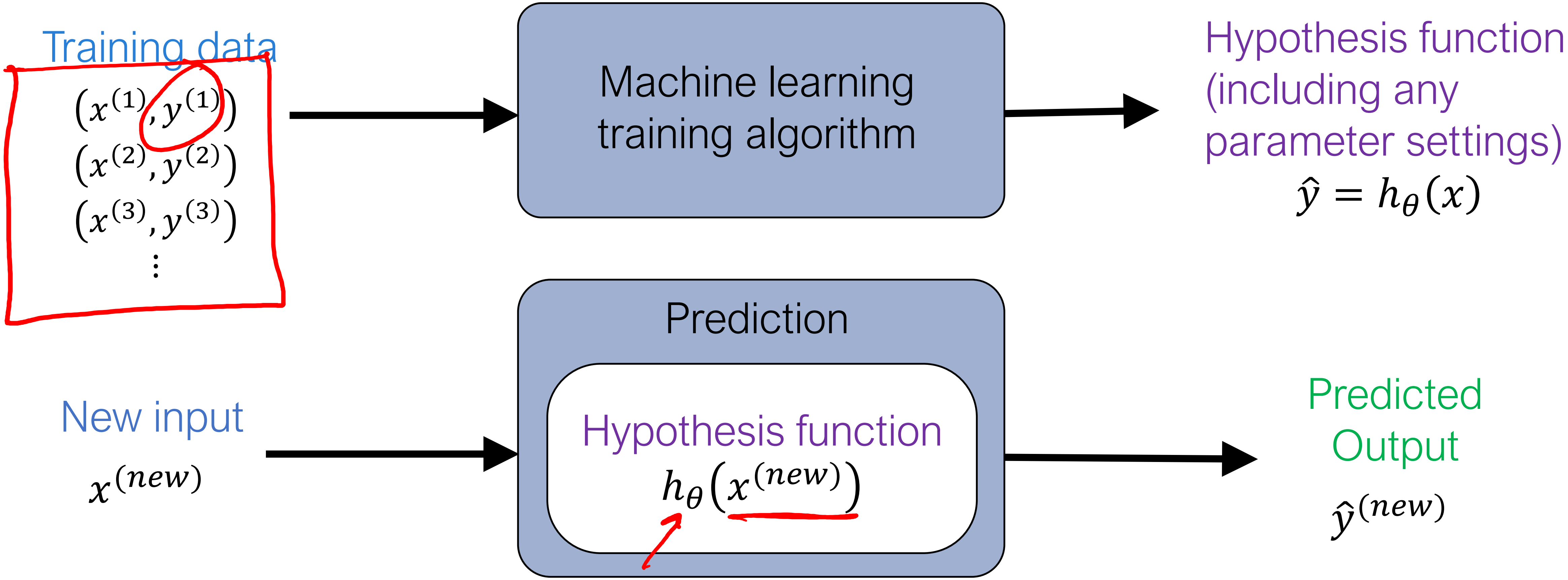
Machine learning

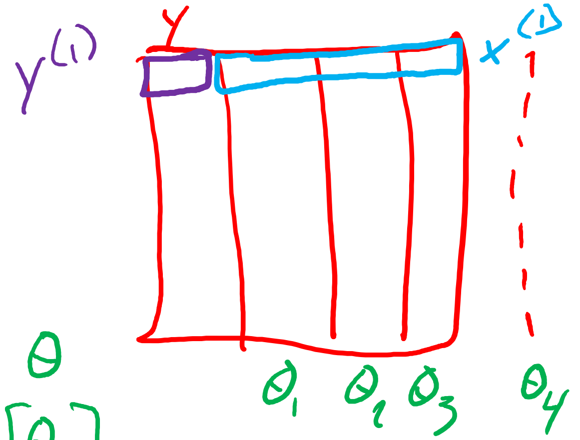
Gradient descent to find the parameters to minimize MSE for a linear model is an example of a *machine learning algorithm*

Basic idea: in many domains, it is difficult to hand-build a predictive model, but easy to collect lots of data; machine learning provides a way to automatically infer the predictive model from data

Machine learning

The basic process (supervised learning):

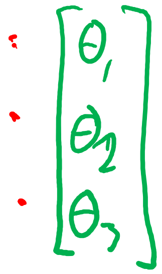




Terminology

Input features: $x^{(i)} \in \mathbb{R}^n, i = 1, \dots, m$

$$\text{E.g.: } x^{(i)} = \begin{bmatrix} \text{High\_Temperature}^{(i)} \\ \text{Is\_Weekday}^{(i)} \\ 1 \end{bmatrix}$$



Outputs: $y^{(i)} \in \mathcal{Y}, i = 1, \dots, m$

$$\text{E.g.: } y^{(i)} \in \mathbb{R} = \text{Peak\_Demand}^{(i)}$$

Model parameters: $\theta \in \mathbb{R}^n$

Hypothesis function: $h_\theta : \mathbb{R}^n \to \mathcal{Y}$, predicts output given input

$$\text{E.g.: } h_\theta(x) = \sum_{j=1}^{n} \theta_j \cdot x_j$$

Terminology

Loss function: $\ell: \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}_+$, measures the difference between a prediction and an actual output

$$\text{E.g.:}\ \ell(\hat{y}, y) = (\hat{y} - y)^2$$

The canonical machine learning optimization problem:

$$= \quad \underset{\theta}{\text{minimize}} \sum_{i=1}^{m} \ell\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}$$

Virtually every machine learning algorithm has this form, just specify

- What is the hypothesis function?
- What is the loss function?
- How do we solve the optimization problem?

Example machine learning algorithms

Note: we (machine learning researchers) have not been consistent in naming conventions, many machine learning algorithms actually only specify some of these three elements

- **Least squares:** {linear hypothesis, squared loss, (usually) analytical solution}
- **Linear regression:** {linear hypothesis, *, *}
- **Support vector machine:** {linear or kernel hypothesis, hinge loss, *}
- **Neural network:** {Composed non-linear function, *, (usually) gradient descent)
- **Decision tree:** {Hierarchical axis-aligned halfplanes, *, greedy optimization}
- **Naïve Bayes:** {Linear hypothesis, joint probability under certain independence assumptions, analytical solution}

Outline

Least squares regression: a simple example

Machine learning notation

Linear regression revisited

Matrix/vector notation and analytic solutions

Implementing linear regression

Least squares revisited

Using our new terminology, plus matrix notion, let's revisit how to solve linear regression with a squared error loss

temp weekday 1

$\theta_1 x_1 + \theta_2 x_2 + \theta_3 \cdot x_3$

Setup:

- Linear hypothesis function: $h_\theta(x) = \sum_{j=1}^{n} \theta_j \cdot x_j$
- Squared error loss: $\ell(\hat{y}, y) = (\hat{y} - y)^2$
- Resulting machine learning optimization problem:

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{m} \left(\sum_{j=1}^{n} \theta_j \cdot x_j^{(i)} - y^{(i)} \right)^2 \equiv \underset{\theta}{\text{minimize}}\, E(\theta)$$

Derivative of the least squares objective

Compute the partial derivative with respect to an arbitrary model parameter θ_j

$$\frac{\partial E(\theta)}{\partial \theta_k} = \frac{\partial}{\partial \theta_k} \sum_{i=1}^{m} \left(\sum_{j=1}^{n} \theta_j \cdot x_j^{(i)} - y^{(i)} \right)^2$$

$$= \sum_{i=1}^{m} \frac{\partial}{\partial \theta_k} \left(\sum_{j=1}^{n} \theta_j \cdot x_j^{(i)} - y^{(i)} \right)^2$$

$$= \sum_{i=1}^{m} 2 \left(\sum_{j=1}^{n} \theta_j \cdot x_j^{(i)} - y^{(i)} \right) \frac{\partial}{\partial \theta_k} \sum_{j=1}^{n} \theta_j \cdot x_j^{(i)}$$

$$= \sum_{i=1}^{m} 2 \left(\sum_{j=1}^{n} \theta_j \cdot x_j^{(i)} - y^{(i)} \right) x_k^{(i)}$$

Gradient descent algorithm

1. Initialize $\theta_k := 0,\ k = 1, \dots, n$

2. Repeat:
   - For $k = 1, \dots, n$:

$$\theta_k := \theta_k - \alpha \sum_{i=1}^{m} 2\left(\sum_{j=1}^{n} \theta_j \cdot x_j^{(i)} - y^{(i)}\right) x_k^{(i)}$$

Note: do not actually implement it like this, you'll want to use the matrix/vector notation we will over soon

Outline

Least squares regression: a simple example

Machine learning notation

Linear regression revisited

Matrix/vector notation and analytic solutions

Implementing linear regression

The gradient

It is typically more convenient to work with a vector of all partial derivatives, called the **gradient**

For a function $f: \mathbb{R}^n \to \mathbb{R}$, the gradient is a vector

$$\nabla_\theta f(\theta) = \begin{bmatrix} \frac{\partial f(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial f(\theta)}{\partial \theta_n} \end{bmatrix} \in \mathbb{R}^n$$

Gradient in vector notation

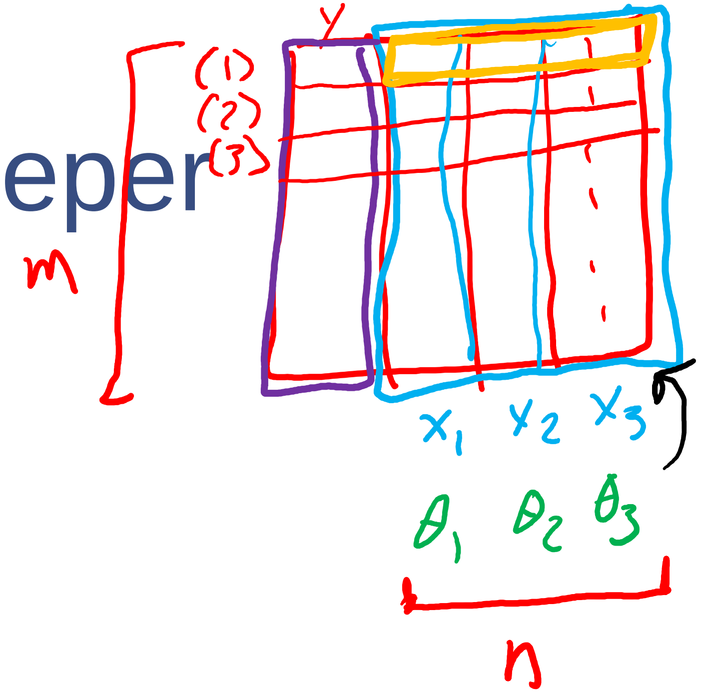
We can actually *simplify* the gradient computation (both notationally and computationally) substantially using matrix/vector notation

$$\frac{\partial E(\theta)}{\partial \theta_k} = 2\sum_{i=1}^{m}\left(\sum_{j=1}^{n}\theta_j \cdot x_j^{(i)} - y^{(i)}\right)x_k^{(i)}$$

$$\Leftrightarrow \nabla_\theta E(\theta) = 2\sum_{i=1}^{m} x^{(i)}\left({x^{(i)}}^T\theta - y^{(i)}\right)$$

Putting things in this form also make it more clear how to analytically find the optimal solution for last squares

Matrix notation, one level deeper



Let's define the matrices

$$X = \begin{bmatrix} - & x^{(1)^T} & - \\ - & x^{(2)^T} & - \\ & \vdots & \\ - & x^{(m)^T} & - \end{bmatrix}, \quad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

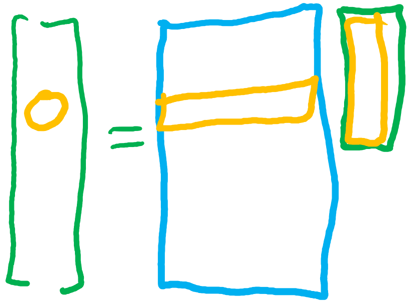
$$\sum_{i=1}^{M} \left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

$$\|\hat{y} - y\|_2^2$$

$$\hat{y}^{(i)} = \sum_{j=1}^{N} \theta_j x_j^{(i)}$$

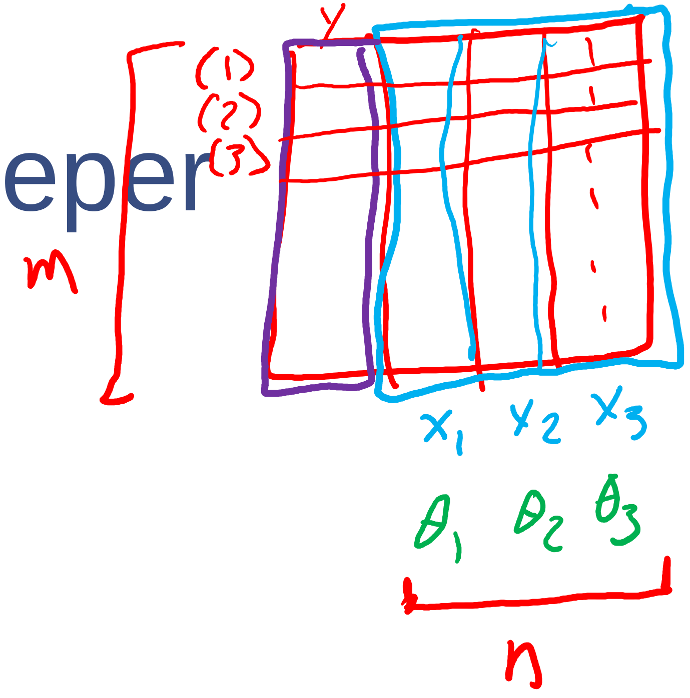
$$\hat{y} = X\theta$$

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \rightarrow \hat{y}^{(i)} = \theta^T x^{(i)} = x^{(i)T}\theta$$



Euclidean (L2) norm: $\|z\|_2 = \left(\sum_i z_i^2\right)^{\frac{1}{2}}$ $\quad \|z\|_2^2 = \sum_i z_i^2$

Matrix notation, one level deeper



Let's define the matrices

$$X = \begin{bmatrix} - \; x^{(1)^T} \; - \\ - \; x^{(2)^T} \; - \\ \vdots \\ - \; x^{(m)^T} \; - \end{bmatrix}, \quad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

$$E(\theta) = \|\hat{y} - y\|_2^2$$

$$E(\theta) = \|X\theta - y\|_2^2$$

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$$

Euclidean (L2) norm: $\|z\|_2 = \left(\sum_i z_i^2\right)^{\frac{1}{2}}$ $\qquad \|z\|_2^2 = \sum_i z_i^2$

Gradient in linear algebra notation

$y = X\theta$

We can actually *simplify* the gradient computation (both notationally and computationally) substantially using matrix/vector notation

$$E(\theta) = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} \theta_j \cdot x_j^{(i)} - y^{(i)} \right)^2 \qquad \nabla_\theta E(\theta) = 2 \sum_{i=1}^{m} x^{(i)} \left(x^{(i)^T} \theta - y^{(i)} \right)$$

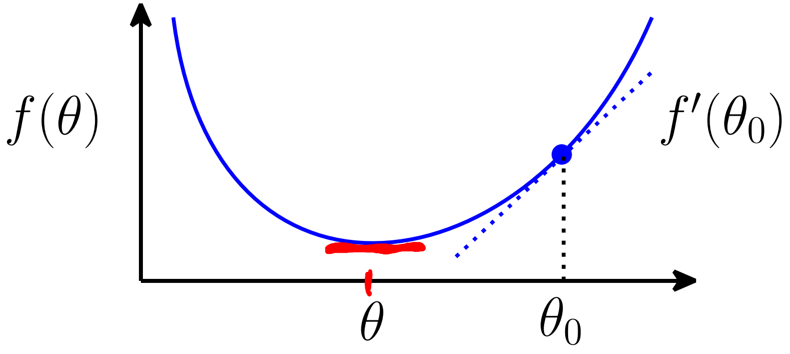
$$E(\theta) = \|X\theta - y\|_2^2 \qquad \nabla_\theta E(\theta) = 2X^T(X\theta - y)$$

Putting things in this form also make it more clear how to analytically find the optimal solution for last squares

Solving least squares

Gradient also gives a condition for optimality:

- Gradient must equal zero



Solving for $\nabla_\theta E(\theta) = 0$:

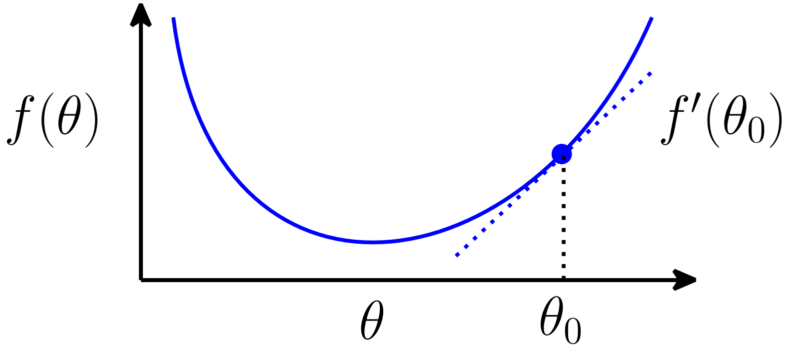
$$2X^T(X\theta - y) = 0$$

These are known as the normal equations an extremely convenient closed-form solution for least squares

Solving least squares

Gradient also gives a condition for optimality:

- Gradient must equal zero



Solving for $\nabla_\theta E(\theta) = 0$:

$$2\sum_{i=1}^{m} x^{(i)}\left({x^{(i)}}^T\theta - y^{(i)}\right) = 0$$

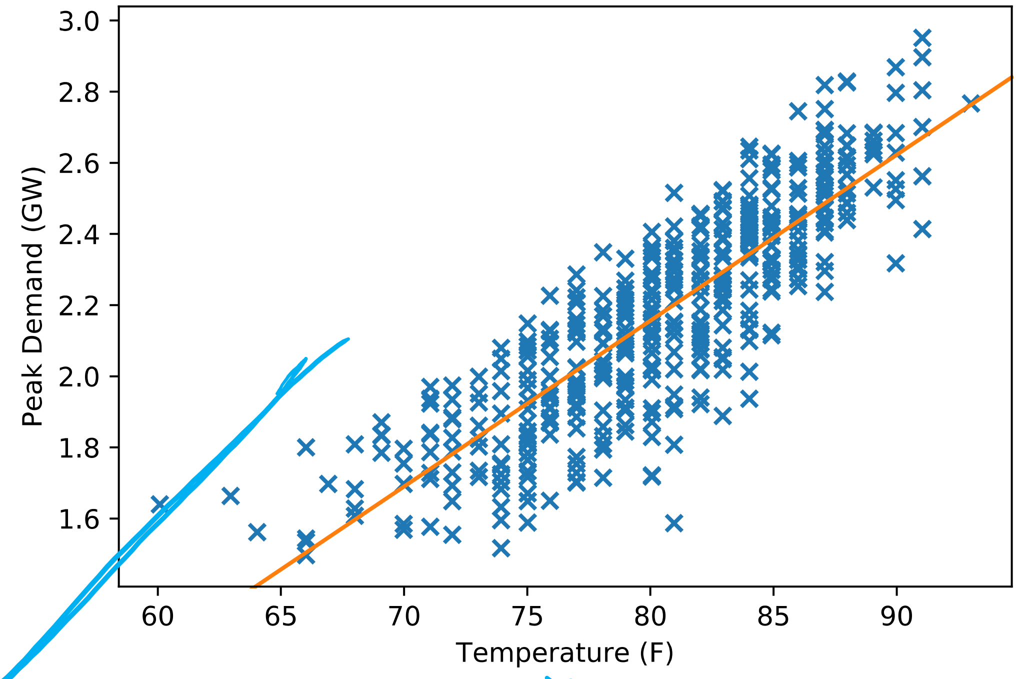
$$\Rightarrow \left(\sum_{i=1}^{m} x^{(i)}{x^{(i)}}^T\right)\theta - \sum_{i=1}^{m} x^{(i)}y^{(i)} = 0$$

$$\Rightarrow \theta^\star = \left(\sum_{i=1}^{m} x^{(i)}{x^{(i)}}^T\right)^{-1}\left(\sum_{i=1}^{m} x^{(i)}y^{(i)}\right)$$

Example: electricity demand

Returning to our electricity demand example:

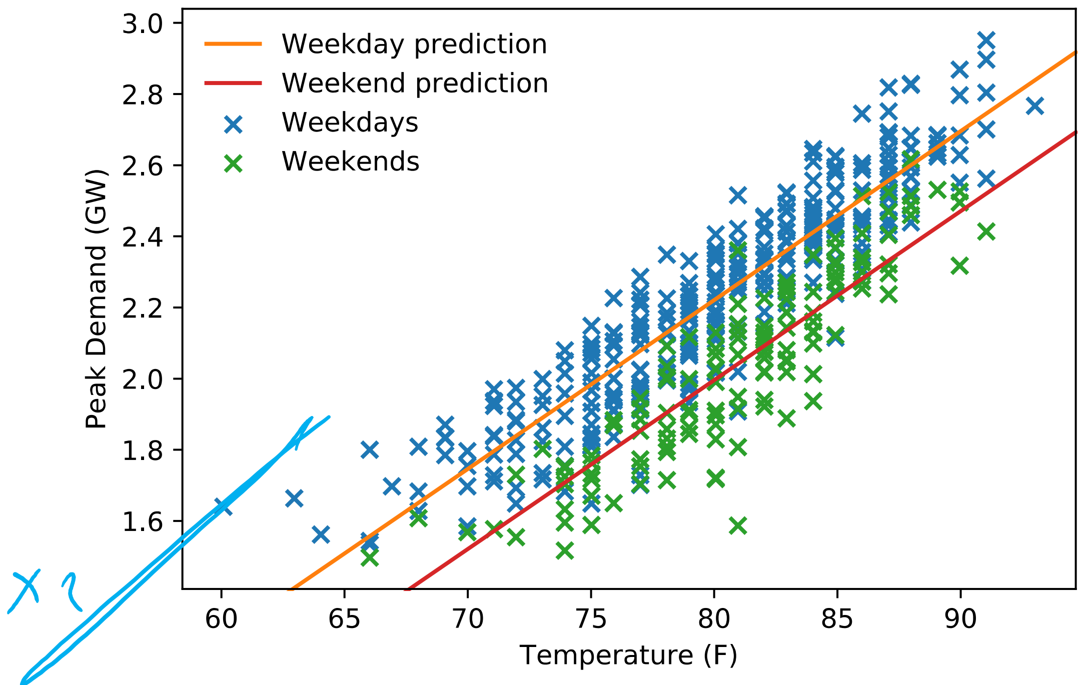
$$x^{(i)} = \begin{bmatrix} \text{High\_Temperature}^{(i)} \\ 1 \end{bmatrix}, \qquad \theta^\star = (X^T X)^{-1} X^T y = \begin{bmatrix} 0.046 \\ -1.574 \end{bmatrix}$$



Example: electricity demand

Returning to our electricity demand example:

$$x^{(i)} = \begin{bmatrix} \text{High\_Temperature}^{(i)} \\ \text{Is\_Weekday}^{(i)} \\ 1 \end{bmatrix}, \qquad \theta^\star = (X^TX)^{-1}X^Ty = \begin{bmatrix} 0.047 \\ 0.225 \\ -1.803 \end{bmatrix}$$



Outline

Least squares regression: a simple example

Machine learning notation

Linear regression revisited

Matrix/vector notation and analytic solutions

Implementing linear regression

Manual implementation of linear regression

Create data matrices:

```
# initialize X matrix and y vector
X = np.array([df["Temp"], df["IsWeekday"], np.ones(len(df))]).T
y = df_summer["Load"].values
```

Compute solution:

```
# solve least squares
theta = np.linalg.solve(X.T @ X, X.T @ y)
print(theta)
# [ 0.04747948  0.22462824 -1.80260016]
```

Make predictions:

```
# predict on new data
Xnew = np.array([[77, 1, 1], [80, 0, 1]])
ypred = Xnew @ theta
print(ypred)
# [ 2.07794778  1.99575797]
```

Scikit-learn

By far the most popular machine learning library in Python is the scikit-learn library (http://scikit-learn.org/)

Reasonable (usually) implementation of many different learning algorithms, usually fast enough for small/medium problems

Important: you *need* to understand the very basics of how these algorithms work in order to use them effectively

Sadly, a lot of data science in practice seems to be driven by the default parameters for scikit-learn classifiers…

Linear regression in scikit-learn

Fit a model and predict on new data

```
from sklearn.linear_model import LinearRegression

# don't include constant term in X
X = np.array([df_summer["Temp"], df_summer["IsWeekday"]]).T
model = LinearRegression(fit_intercept=True, normalize=False)
model.fit(X, y)

# predict on new data
Xnew = np.array([[77, 1], [80, 0]])
model.predict(Xnew)
# [ 2.07794778  1.99575797]
```

Inspect internal model coefficients

```
print(model.coef_, model.intercept_)
# [ 0.04747948  0.22462824]  -1.80260016
```

Scikit-learn-like model, manually

We can easily implement a class that contains a scikit-learn-like interface

```
class MyLinearRegression:
    def __init__(self, fit_intercept=True):
        self.fit_intercept = fit_intercept

    def fit(self, X, y):
        if self.fit_intercept:
            X = np.hstack([X, np.ones((X.shape[0],1))])

        self.coef_ = np.linalg.solve(X.T @ X, X.T @ y)

        if self.fit_intercept:
            self.intercept_ = self.coef_[-1]
            self.coef_ = self.coef_[:-1]

    def predict(self, X):
        pred = X @ self.coef_
        if self.fit_intercept:
            pred += self.intercept_
        return pred
```