# Optimization for Machine Learning 

Nick Henderson, AJ Friend (Stanford University)<br>Kevin Carlberg (Sandia National Laboratories)

August 13, 2019

## Model fitting

## Outline

Model fitting
Linear least squares: 1D case with linear data
Linear least squares: 1D case with non-linear data
Linear least squares: general formulation and matrix-vector form
Examples
Nonlinear least squares
Beyond least squares
Deep Feedforward Networks
Stochastic methods

## Notation

The notation between these worlds is not consistent

- Optimization
- $f$ : optimization objective function
- $x$ : optimization variables
- Machine learning (this set of slides)
- $\phi$ : optimization objective function (i.e., loss function)
- $\beta$ or $\theta$ : optimization variables (i.e., model parameters)
- $f$ : regression function mapping inputs to outputs
- $x$ : model inputs (i.e., independent variable)
- $y$ : model outputs (i.e., response variable)


## Least-squares regression

- A type of model fitting with many applications
- Goal: find a model that best fits training data in the least-squares sense
- Illuminates the connection between unconstrained optimization and statistics/machine learning
- We will use the following iPython notebooks
- least-squares.ipynb
- polynomial-fit.ipynb
- smooth.ipynb
- huber.ipynb


## Linear least squares: 1D case with linear data

## Outline

## Model fitting

Linear least squares: 1D case with linear data
Linear least squares: 1D case with non-linear data
Linear least squares: general formulation and matrix-vector form
Examples
Nonlinear least squares
Beyond least squares
Deep Feedforward Networks
Stochastic methods

Linear least squares: 1D case with linear data

## Problem set up

- Given: $m$ training examples (i.e., training set)

| $x_{i}$ : independent variable | $y_{i}:$ response variable |
| :---: | :---: |
| 0.0 | 0.46 |
| 0.11 | 0.31 |
| 0.22 | 0.38 |
| 0.33 | 0.39 |
| 0.44 | 0.65 |
| 0.56 | 0.40 |
| 0.67 | 0.87 |
| 0.78 | 0.69 |
| 0.89 | 0.87 |
| 1.0 | 0.88 |

- Goal: construct a model that can predict $y$ from $x$


## Where might these data come from?

| $x:$ independent variable | $y$ : response variable |
| :---: | :---: |
| height | weight |
| square feet | price of home |
| device property | failure rate |
| stock market return | individual asset return |

## Regression: Approach

Goal: construct a model that can predict $y$ from $x$

- In general, we do not know the mathematical model characterizing the underlying process that actually generated the data
- So, we assume that the data were generated from a model comprising the sum of a (deterministic) function and (stochastic) iid Gaussian noise:

$$
y_{i}=f_{\text {true }}\left(x_{i}\right)+\sigma \cdot \epsilon_{i}, \quad i=1, \ldots, m
$$

with $f_{\text {true }}\left(x_{i}\right)$ unknown and $\epsilon_{i} \sim N(0,1)$

- We aim to construct $f(x)$ such that $f(x) \approx f_{\text {true }}(x)$ in some sense
- This is known as regression and is performed via optimization
- objective function: residual sum of squares $\frac{1}{2} \sum_{i=1}^{m}\left(f\left(x_{i}\right)-y_{i}\right)^{2}$
- optimization variables: parameters within the assumed form of $f(x)$
- Then, we can make predictions $y \approx f(x)$ for new values of $x$.


## Follow along in Python

- See least-squares.ipynb
- In this case, we have set $f_{\text {true }}(x)=\theta_{\text {true }} \cdot x_{i}+b_{\text {true }}$ and $\sigma=0.1$
- $\theta_{\text {true }}=0.6$
- $b_{\text {true }}=0.3$
- Run the first three cells of least-squares.ipynb
- Python code to generate data (in second cell):
np.random.seed(1)
theta $=0.6$
$\mathrm{b}=0.3$
sigma = . 1
$\mathrm{x}=\mathrm{np}$.linspace $(0,1,10)$
$\mathrm{y}=$ theta*x $+\mathrm{b}+$ sigma*np.random.standard_normal(x.shape)


## Plot the data



## The residuals

- Any given data point will result in some error or residual

$$
r_{i}=f\left(x_{i}\right)-y_{i}
$$

- Due to the Gaussian noise, $y_{i}=f_{\text {true }}\left(x_{i}\right)+\sigma \cdot \epsilon_{i} \neq f_{\text {true }}\left(x_{i}\right)$. Thus, the true function $f_{\text {true }}(x)$ will yield residuals

$$
r_{\text {true }, i}=f_{\text {true }}\left(x_{i}\right)-y_{i}=-\sigma \cdot \epsilon_{i}
$$

## The residuals for $f_{\text {true }}(x)$



## Linear regression in one dimension

- In linear regression, we enforce the regression function $f(x)$ to be linear

$$
f(x ; \theta, b)=\theta \cdot x+b
$$

- regression function has two parameters: the slope $\theta$ and the $y$-intercept $b$
- semicolon separates model input from model parameters
- Note: the form of $f(x)$ usually does not match the (generally unknown) form of $f_{\text {true }}(x)$. We are lucky if this happens!


## Fit the model via optimization

- Given training data $\left(x_{i}, y_{i}\right)_{i=1}^{m}$ with $x_{i} \in \mathbf{R}$ and $y_{i} \in \mathbf{R}$
- To fit the model, construct an optimization problem

$$
\underset{\theta, b}{\operatorname{minimize}} \quad \phi(\theta, b)=\frac{1}{2} \sum_{i=1}^{m} r_{i}(\theta, b)^{2}=\frac{1}{2} \sum_{i=1}^{m}\left(f\left(x_{i} ; \theta, b\right)-y_{i}\right)^{2}
$$

- Optimization objective function: residual sum of squares (RSS)
- one contribution from each of the $m$ training examples
- Optimization variables: parameters $\theta$ and $b$
- If the true underlying model actually is $y_{i}=\theta_{\text {true }} \cdot x_{i}+b_{\text {true }}+\sigma \cdot \epsilon_{i}$ with $\epsilon_{i}$ mean-zero Gaussian, then $\theta$ and $b$ are the maximum-likelihood estimates of $\theta_{\text {true }}$ and $b_{\text {true }}$


## Objective function



- The objective function $\phi(\theta, b)$ is appears to be convex (it is!)
- The global minimum occurs around $\theta^{\star} \approx 0.6$ and $b^{\star} \approx 0.35$


## Optimizing by hand

Recall the sufficient conditions for (unconstrained) optimality:

1. $\nabla \phi\left(\theta^{\star}, b^{\star}\right)=0$
2. $\nabla^{2} \phi\left(\theta^{\star}, b^{\star}\right) \succ 0$. This holds everywhere!

- The objective function is strongly convex
- This simplifies things: we only need to find a stationary point satisfying condition 1
- This is one reason why convex optimization is so nice!

Let's compute $\theta^{\star}$ and $b^{\star}$ such that that the first condition holds.

## Compute gradient analytically and set to zero

Analytical gradient computation:

$$
\begin{gathered}
\frac{\partial \phi}{\partial \theta}=\frac{1}{2} \sum_{i=1}^{m} \frac{\partial}{\partial \theta}\left(\theta \cdot x_{i}+b-y_{i}\right)^{2}=\theta \sum x_{i}^{2}+b \sum x_{i}-\sum x_{i} y_{i} \\
\frac{\partial \phi}{\partial b}=\frac{1}{2} \sum_{i=1}^{m} \frac{\partial}{\partial b}\left(\theta \cdot x_{i}+b-y_{i}\right)^{2}=\theta \sum x_{i}+n b-\sum y_{i}
\end{gathered}
$$

Set analytical gradient to zero and obtain a system of equations:

$$
\begin{aligned}
& \frac{\partial \phi}{\partial \theta}=0 \\
& \frac{\partial \phi}{\partial b}=0
\end{aligned}
$$

## Solution

$$
\begin{gathered}
\theta=\frac{\sum x_{i} y_{i}-\frac{1}{m} \sum x_{i} \sum y_{i}}{\sum x_{i}^{2}-\frac{1}{m}\left(\sum x_{i}\right)^{2}} \\
b=\frac{\sum y_{i}-\theta \sum x_{i}}{m}
\end{gathered}
$$

## Let's look at $\theta$

Something looks nice here:

$$
\theta=\frac{\sum x_{i} y_{i}-\frac{1}{m} \sum x_{i} \sum y_{i}}{\sum x_{i}^{2}-\frac{1}{m}\left(\sum x_{i}\right)^{2}}
$$

Multiply both numerator and denominator by $1 / m$ :

$$
\theta=\frac{\frac{1}{m} \sum x_{i} y_{i}-\frac{1}{m} \sum x_{i} \frac{1}{m} \sum y_{i}}{\frac{1}{m} \sum x_{i}^{2}-\left(\frac{1}{m} \sum x_{i}\right)^{2}}
$$

We see sample covariance and variance here!

$$
\theta=\frac{\operatorname{cov}(X, Y)}{\operatorname{var}(X)}
$$

## Let's solve in Python!

Code:
\# solve via numpy covariance function
$\mathrm{A}=\mathrm{np} \cdot \mathrm{vstack}((\mathrm{x}, \mathrm{y}))$
$\mathrm{V}=\mathrm{np} \cdot \operatorname{cov}(\mathrm{A})$
theta_est $=\mathrm{V}[0,1] / \mathrm{V}[0,0]$
b_est $=$ (y.sum() - theta_est*x.sum()) / len(x)
print (theta_est)
print(b_est)
Result:
theta_est $=0.56604$ (true value $=0.6$ )
b_est $=0.30727$ (true value $=0.3$ )

## Look at the plot



## Solve in CVXPY

Remember the optimization problem: minimize $\frac{1}{2} \sum_{i=1}^{m}\left(\theta \cdot x_{i}+b-y_{i}\right)^{2}$ We can write this directly in CVXPY:

```
from cvxpy import *
# Construct the problem.
theta_cvx = Variable()
b_cvx = Variable()
objective = Minimize(sum_squares(theta_cvx*x + b_cvx - y))
prob = Problem(objective)
# The optimal objective is returned by prob.solve().
result = prob.solve()
theta_cvx.value = 0.56604, b_cvx.value = 0.30727
```


## Linear least squares: 1D case with non-linear data

## Outline

## Model fitting

Linear least squares: 1D case with linear data
Linear least squares: 1D case with non-linear data
Linear least squares: general formulation and matrix-vector form

## Examples

Nonlinear least squares
Beyond least squares
Deep Feedforward Networks
Stochastic methods

Linear least squares: 1D case with non-linear data

## What about these data?



- Here, $f_{\text {true }}(x)=\theta_{\text {true }} \exp (x)+b_{\text {true }}$, which we do not know
- We just have access to the data!


## We could fit a linear model

- Given our ignorance of $f_{\text {true }}$, we could fit a linear model

$$
f(x ; \theta, b)=\theta \cdot x+b,
$$



- This yields an objective-function value of $\phi(\theta, b)=283.63$


## We can also fit an exponential model

- If we think that the underlying model may be exponential, we can also try

$$
f(x ; \theta, b)=\theta \cdot \exp (x)+b
$$

- Model still linear in the parameters $\theta$ and $b$ : "Linear least squares" (same optimization problem)
- But model nonlinear in the parameters: "Nonlinear regression"

CVXPY code:

```
theta = Variable()
b = Variable()
objective = Minimize(sum_squares(theta*np.exp(x) + b - y))
prob = Problem(objective)
result = prob.solve()
```


## Result



- This yields a smaller objective-function value of $\phi(\theta, b)=122.80$
- better fit to training data
- Caution: can overfit training data
- must assess generalization error on an independent test set


## Linear least squares: general formulation and matrix-vector form

## Outline

## Model fitting

Linear least squares: 1D case with linear data
Linear least squares: 1D case with non-linear data
Linear least squares: general formulation and matrix-vector form

## Examples

Nonlinear least squares
Beyond least squares
Deep Feedforward Networks
Stochastic methods

Linear least squares: general formulation and matrix-vector form

## General formulation for linear least squares

- $x \in \mathbf{R}^{p}: p$-dimensional model inputs (i.e., independent variables)
- $y \in \mathbf{R}$ : model outputs (i.e., response variable)
- $f: \mathbf{R}^{p} \rightarrow \mathbf{R}$ : model a linear combination of $n$ functions $f_{i}: \mathbf{R}^{n} \rightarrow \mathbf{R}, i=1, \ldots, n$ :

$$
f(x ; \beta)=\sum_{i=1}^{n} f_{i}(x) \beta_{i}
$$

- If $f_{i}$ is nonlinear in $x$, then this is "nonlinear regression"
- Previous example: $m=2 ; f_{1}(x)=1 ; f_{2}(x)=x$ or $f_{2}(x)=\exp (x) ; \beta_{1}=\theta, \beta_{2}=m$
- $\beta=\left(\beta_{1}, \ldots, \beta_{n}\right) \in \mathbf{R}^{n}$ : optimization variables (i.e., model parameters)


## Matrix-vector form

- Assume input-output data of the form $\left(x_{j}, y_{j}\right)_{j=1}^{m}$
- The residual for the $j$ th data point is $r_{j}(\beta)=f\left(x_{j} ; \beta\right)-y_{j}$
- Residual sum of squares (RSS) objective function is

$$
\phi(\beta)=\frac{1}{2} \sum_{j=1}^{m} r_{j}(\beta)^{2}=\frac{1}{2} \sum_{j=1}^{m}\left(f\left(x_{j} ; \beta\right)-y_{j}\right)^{2}=\frac{1}{2} \sum_{j=1}^{m}\left(\sum_{i=1}^{n} f_{i}\left(x_{j}\right) \beta_{i}-y_{j}\right)^{2}
$$

- Defining

$$
A=\left[\begin{array}{ccc}
f_{1}\left(x_{1}\right) & \cdots & f_{n}\left(x_{1}\right) \\
\vdots & \ddots & \vdots \\
f_{1}\left(x_{m}\right) & \cdots & f_{n}\left(x_{m}\right)
\end{array}\right], \quad \beta=\left[\begin{array}{c}
\beta_{1} \\
\vdots \\
\beta_{n}
\end{array}\right], \quad b=\left[\begin{array}{c}
y_{1} \\
\vdots \\
y_{m}
\end{array}\right]
$$

we can write the objective function as $\phi(\beta)=\frac{1}{2}\|A \beta-b\|_{2}^{2}$

## Standard form for least squares

$$
\underset{x}{\operatorname{minimize}} \quad \frac{1}{2}\|A x-b\|_{2}^{2}
$$

In the context of model fitting:

- $A \in \mathbf{R}^{m \times n}$ is the matrix that contains data from independent variables
- $b \in \mathbf{R}^{m}$ is the vector containing response data ( $\beta$ on last slide)
- $x \in \mathbf{R}^{n}$ is the vector of model parameters
- For each of the $m$ training examples, the residual is we have the equation

$$
r_{i}=a_{i}^{T} x-b_{i}
$$

- $a_{i}^{T} \in \mathbf{R}^{1 \times n}$ is the $i$ th row of $A$
- Notation from statistics:

$$
\underset{\beta}{\operatorname{minimize}} \frac{1}{2}\|\mathbf{X} \beta-\mathbf{y}\|_{2}^{2}
$$

## CVXPY for least squares

```
    # generate input and response data
    np.random.seed(1); n = 10 # number of data points
    input_data = np.linspace(0,1,n)
    response_data = 0.6*input_data + 0.3 + 0.1*np.random.standard_normal(n)
    # least-squares matrix and vector
    A = np.vstack([input_data,np.ones(n)]).T; b = response_data
    # CVX problem
    x = Variable(A.shape[1])
    objective = Minimize(sum_squares(A*x - b))
    prob = Problem(objective); result = prob.solve()
    # get value & print
    x_star = np.array(x.value)
    print('slope = {:.4}, intercept = {:.4}'.format(x_star[0,0],x_star [1,0]))
    slope = 0.566, intercept = 0.3073
Linear least squares: general formulation and matrix-vector form

\section*{Examples}

\section*{Outline}

\section*{Model fitting}

Linear least squares: 1D case with linear data
Linear least squares: 1D case with non-linear data
Linear least squares: general formulation and matrix-vector form

\section*{Examples}

Nonlinear least squares
Beyond least squares
Deep Feedforward Networks
Stochastic methods

\section*{What about these data?}


\section*{Polynomial regression}
- Polynomial model:
\[
y \approx f(x ; \beta)=\beta_{1}+\beta_{2} x+\beta_{3} x^{2}+\cdots+\beta_{n} x^{n-1}
\]
- \(\beta_{i}, i=1, \ldots, n\) are the model parameters and optimization variables
- Linear least-squares framework: \(f(x)=\sum_{i=1}^{n} f_{i}(x) \beta_{i}\) with monomials
\[
f_{i}(x)=x^{i-1}, i=1, \ldots, n
\]

\section*{Polynomial regression}
- As before, define \(A, \beta, b\) to put in standard form for least squares
\[
A=\left[\begin{array}{ccccc}
1 & x_{1} & x_{1}^{2} & \ldots & x_{1}^{n-1} \\
1 & x_{2} & x_{2}^{2} & \ldots & x_{2}^{n-1} \\
1 & x_{3} & x_{3}^{2} & \ldots & x_{3}^{n-1} \\
1 & x_{4} & x_{4}^{2} & \ldots & x_{4}^{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & x_{m} & x_{m}^{2} & \ldots & x_{m}^{n-1}
\end{array}\right], \quad \beta=\left[\begin{array}{c}
\beta_{1} \\
\beta_{2} \\
\beta_{3} \\
\beta_{4} \\
\vdots \\
\beta_{n}
\end{array}\right], \quad b=\left[\begin{array}{c}
y_{1} \\
y_{2} \\
y_{3} \\
y_{4} \\
\vdots \\
y_{m}
\end{array}\right]
\]
- Solve the least-squares problem
\[
\underset{\beta}{\operatorname{minimize}} \frac{1}{2}\|A \beta-b\|_{2}^{2}
\]
- This form for \(A\) is called the Vandermonde matrix

\section*{Solve with CVXPY}
- See polynomial-fit.ipynb
```

def cvxpy_poly_fit(x,y,degree):
\# construct data matrix
A = np.vander(x,degree+1)
b = y
beta_cvx = Variable(degree+1)
\# set up optimization problem
objective = Minimize(sum_squares(A*beta_cvx - b))
constraints = []
\# solve the problem
prob = Problem(objective,constraints)
prob.solve()
\# return the polynomial coefficients
return np.array(beta_cvx.value)

```

\section*{Linear fit}


\section*{Quadratic fit}


\section*{Cubic fit}


\section*{True model}


Exarnplesh his was the true (but unknown) model that generated the data

\section*{Example: time series smoothing}
- See smooth.ipynb
- Noisy observations \(\left(x_{i}, y_{i}\right), i=1, \ldots, m\) at regular intervals (discretized curve)
- New modeling approach
- We assume we don't have a model for the curve (linear, polynomial, ...)
- But we do believe that the curve should be smooth
- Idea: find \(\beta_{i}, i=1, \ldots, m\) that are close to \(y_{i}\), but are penalized for being nonsmooth
- Linear least squares with \(f_{i}\left(x_{j}\right)=\delta_{i j}\left(x_{j}\right), i=1, \ldots, m\) (Kronecker delta)
- The number of optimization variables \(n\) is equal to number of data points \(m\)

Time series data


\section*{Optimization problem}
- Want \(\beta_{i} \approx y_{i}, i=1, \ldots, m\)
- Want \(f\left(x_{j}\right)=\sum_{i=1}^{n} \delta_{i j}\left(x_{j}\right) \beta_{i}\) to be smooth on the grid \(x_{j}, j=1, \ldots, m\)
- Optimization problem
\[
\underset{\beta}{\operatorname{minimize}}\|\beta-b\|_{2}^{2}+\rho \cdot \operatorname{penalty}(\beta)
\]
- Introduce a penalty function to encourage smoothness
- Penalty parameter \(\rho\) enables trading off two competing objectives:
1. \(\rho\) small: \(\|\beta-b\|_{2}^{2}\) small and model is a better fit to training data
2. \(\rho\) large: penalty \((\beta)\) small and model is smoother

\section*{How to quantify smoothness?}
- Smoothness: a curve whose slope does not change much
- The second derivative measures the rate of change of the slope
- Approximate the second derivative via second-order finite differences as \(D \beta\), where
\[
D=\left(\begin{array}{cccccccc}
1 & -2 & 1 & 0 & \ldots & & & 0 \\
0 & 1 & -2 & 1 & 0 & \ldots & & 0 \\
0 & 0 & 1 & -2 & -1 & 0 & \ldots & 0 \\
\vdots & & & & & & &
\end{array}\right)
\]
assuming a uniform grid \(x_{j}, j=1, \ldots, m\).

\section*{Least squares model}
- Updated optimization problem:
\[
\underset{\beta}{\operatorname{minimize}}\|\beta-b\|_{2}^{2}+\rho\|D \beta\|_{2}^{2}
\]
- Standard form:
\[
\underset{\beta}{\operatorname{minimize}}\left\|\binom{I}{\rho D} \beta-\binom{b}{0}\right\|_{2}^{2}
\]

\section*{Solve the problem in CVXPY}
```


# get second-order difference matrix

D = diff(n, 2) \# user-defined function
rho = 1

# construct and solve problem

beta = cvx.Variable(n)
cvx.Problem(cvx.Minimize(cvx.sum_squares(beta-b)
+rho*cvx.sum_squares(D*beta))).solve()
beta = np.array(beta.value).flatten()

```
\[
\rho=1
\]

\[
\rho=10
\]

\[
\rho=1000
\]


Nonlinear least squares

\section*{Outline}
```

    Model fitting
    Linear least squares: 1D case with linear data
    Linear least squares: 1D case with non-linear data
Linear least squares: general formulation and matrix-vector form

```
Examples

Nonlinear least squares
Beyond least squares
Deep Feedforward Networks
Stochastic methods

Nonlinear least squares

\section*{Nonlinear least squares}

\section*{Linear least squares:}
1. Model is linear in the parameters
\[
f(x ; \beta)=\sum_{i=1}^{n} \beta_{i} f_{i}(x)
\]
- Linear regression: \(f_{i}\) is also linear in \(x\)
- Nonlinear regression: \(f_{i}\) is nonlinear in \(x\) (e.g., polynomials, exponential)
2. Minimize the residual sum of squares (RSS)

\section*{Nonlinear least squares:}
1. Model \(f(x ; \beta)\) is nonlinear in the parameters \(\beta\)
2. Minimize the same objective function: residual sum of squares (RSS)
- Again equivalent to maximum likelihood if additive Gaussian noise
- Algorithms: line-search (Gauss-Newton) and trust-region (Levenberg-Marquardt)

\section*{Beyond least squares}

\section*{Outline}

\section*{Model fitting}

Linear least squares: 1D case with linear data
Linear least squares: 1D case with non-linear data
Linear least squares: general formulation and matrix-vector form

\section*{Examples}

Nonlinear least squares
Beyond least squares
Deep Feedforward Networks
Stochastic methods

Beyond least squares

\section*{Quadratic loss function}
- See huber.ipynb
- Least squares employs a quadratic loss function

- This function imposes a severe penalty on large values
- As a result, the fit model is very sensitive to outliers (can overfit)

Beyotrd Cast wereause a different loss function?

\section*{Huber loss function}
- The Huber function allows us to better handle outliers in data
- Usual quadratic loss in interval \([-M, M]\)
- Linear loss for \(|x|>M\)
\[
h_{M}(x)= \begin{cases}x^{2} & |x| \leq M \\ 2 M|x|-M^{2} & |x|>M\end{cases}
\]


\section*{Huber loss function}
- This function imposes a less severe penalty on large values
- Let's repeat the time-series example, but include extreme outliers
- Penalize closeness to data with Huber function \(h_{M}\) to reduce outlier influence:
\[
\underset{\beta}{\operatorname{minimize}} \sum_{i=1}^{m} h_{M}\left(\beta_{i}-y_{i}\right)+\rho\|D \beta\|_{2}^{2}
\]
- \(M\) parameter controls width of quadratic region, or "non-outlier" errors
- This is no longer least squares!
- CVXPY has implemented the Huber loss function

\section*{Huber data}


\section*{Least-squares smoothing}
```


# get second-order difference matrix

D = diff(n, 2)
rho = 20
beta = Variable(n)
obj = sum_squares(beta-b) + rho*sum_squares(D*beta)
Problem(Minimize(obj)).solve()
beta = np.array(beta.value).flatten()

```

\section*{Least-squares smoothing result}

- Model overfits the outliers

\section*{Huber smoothing}
```


# get second-order difference matrix

D = diff(n, 2)
rho = 20
M = . 15 \# huber radius
beta = Variable(n)
obj = sum_entries(huber(beta-b, M)) + rho*sum_squares(D*beta)
Problem(Minimize(obj)).solve()
x = np.array(x.value).flatten()

```

\section*{Huber smoothing result}

- The model is less sensitive to outliers!

\section*{Deep Feedforward Networks}

\section*{Outline}
Model fitting
Linear least squares: 1D case with linear data
Linear least squares: 1D case with non-linear data
Linear least squares: general formulation and matrix-vector form
Examples
Nonlinear least squares
Beyond least squares
Deep Feedforward Networks
Stochastic methods
Deep Feedforward Networks ..... 70

\section*{Deep Feedforward Networks}

A deep feedforward network defineds a particular model \(f(x ; \beta)\)
- \(f(x ; \beta)=f^{(3)}\left(f^{(2)}\left(f^{(1)}\left(x ; \beta_{1}\right) ; \beta_{2}\right) ; \beta_{3}\right)\) is a 'network' (function composition)
- \(f^{(i)}\left(x ; \beta_{i}\right)\) : function charactering the \(i\) th layer with parameters \(\beta_{i}\)
- parameters \(\beta=\left(\beta_{1}, \beta_{2}, \beta_{3}\right) \in \mathbf{R}^{n}\)
- Evaluating \(f\) is 'forward propagation': start at the beginning \(\left(f^{(1)}\right)\) and evaluate forward sequentially
- It is 'deep' if there are many composed functions, and thus \(\beta\) is high-dimensional
- \(f\) is genearlly nonlinear in the parameters \(\beta\)
- if additive Gaussian noise, then MLE leads to nonlinear least squares
- other loss functions possible (e.g., non-Gaussian noise); then no longer least squares

\section*{Deep Feedforward Networks}
- Computing the gradient can be done by applying the chain rule, e.g.,
\[
\frac{\partial \phi}{\partial \beta_{2}}=\frac{\partial \phi}{\partial f^{(3)}} \frac{\partial f^{(3)}}{\partial x} \frac{\partial f^{(2)}}{\partial \beta_{2}}, \quad \frac{\partial \phi}{\partial \beta_{1}}=\frac{\partial \phi}{\partial f^{(3)}} \frac{\partial f^{(3)}}{\partial x} \frac{\partial f^{(2)}}{\partial x} \frac{\partial f^{(1)}}{\partial \beta_{1}}
\]
- Computing the gradient is referred to as back propagation: the chain rule 'propagates' information from the end of the network ( \(f^{(3)}\) ) upstream (e.g., to \(f^{(1)}\) )

\section*{Deep Feedforward Networks: optimization challenges in optimization}
\[
\operatorname{minimize} \quad \phi(\beta)=\frac{1}{2} \sum_{i=1}^{m}\left(f\left(x_{i} ; \beta\right)-y_{i}\right)^{2}
\]

\section*{High-dimensional}
- many \(\beta\) parameters \(n\) (due to many layers)
- many training samples \(m\) and (need lots of data to tune many parameters)
- solution: stochastic/minibatch methods (e.g., stochastic gradient descent)

\section*{Non-convex}
- can get trapped in local minima
- solution: local minima seem to yield a "low-enough" cost-function value

III conditioning
- solution: second-order methods (but hard for NNs)

\section*{Stochastic methods}

\section*{Outline}
Model fitting
Linear least squares: 1D case with linear dataLinear least squares: 1D case with non-linear data
Linear least squares: general formulation and matrix-vector form
Examples
Nonlinear least squares
Beyond least squares
Deep Feedforward Networks
Stochastic methods
Stochastic methods ..... 75

\section*{What does 'Big Data' mean for model fitting?}
- In model fitting, the objective function is usually composed of a sum of \(m\) contributions:
\[
\phi(\beta)=\frac{1}{m} \sum_{i=1}^{m} \phi_{i}(\beta)
\]
- \(\phi_{i}\) : is the loss associated with the \(i\) th training example
- \(\phi\) : a sampling-based approximation of the expected loss
- 'Big Data' can refer to:
- many training examples: \(m\) large
- many parameters: \(n\) large
- deep learning falls in this category!
- Specialized methods have been developed for these cases!
- stochastic/minibatch methods (next)
- distributed optimization (see 'Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers' by Boyd et al.)

\section*{Stochastic methods}

Here, the gradient is also a sum of \(m\) contributions:
\[
\nabla \phi(\beta)=\frac{1}{m} \sum_{i=1}^{m} \nabla \phi_{i}(\beta)
\]
- Batch methods use this within gradient-based optimization
- Benefit: Preserves traditional convergence rates
- Drawbacks:
- Requires accessing all \(m\) data points each iteration (costly)
- Many data points are likely redundant
- Can we make this less expensive yet still maintain convergence?

\section*{Observations:}
1. The objective is (usually) just the sample mean of the loss function
2. Expectations via Monte Carlo sampling converge slowly (rate \(m^{-1 / 2}\) )
3. Exact gradients aren't needed for convergence

\section*{Stochastic methods}

Stochastic methods: compute approximate the gradient as
\[
\nabla \phi(\beta) \approx \nabla \phi_{i}(\beta)
\]
- \(i\) is a randomly chosen training example
- Stochastic gradient descent (SGD): stochastic approximation to gradient descent:
\[
x_{i+1}=x_{k}-\alpha_{k} \nabla \phi_{i}(\beta)
\]
- Benefits:
- each iteration is much cheaper
- often observe faster rate of convergence as a function of accessed data points
- a descent direction in expectation, i.e., \(\mathbb{E}\left[\nabla \phi_{i}(\beta)\right]=\nabla \phi(\beta)\)
- Drawbacks
- slower rate of convergence as a function of iteration (sublinear for SGD)
- observed slowdown as iterations progress due to noisy gradients

\section*{SGD performance in practice}


Fig. 3.1 Empirical risk \(R_{n}\) as a function of the number of accessed data points (ADPs) for a batch \(L-B F G S\) method and the \(S G\) method (3.7) on a binary classification problem with a logistic loss objective and the \(R C V 1\) dataset. \(S G\) was run with a fixed stepsize of \(\alpha=4\).

Reference: Bottou, L., Curtis, F.E. and Nocedal, J., 2018. Optimization methods for large-scale machine learning. SIAM Review, 60(2), pp.223-311. Stochastic methods

\section*{Improving the convergence rate of stochastic methods}

Noise reduction: reduce variance gradient estimate
- Dynamic sampling: use minibatch estimates of the gradient at iteration \(k\)
\[
\nabla \phi(\beta) \approx \frac{1}{\left|\mathcal{S}_{k}\right|} \sum_{i \in \mathcal{S}_{k}} \nabla \phi_{i}(\beta)
\]
where the minibatch size \(\left|\mathcal{S}_{k}\right|\) increases with \(k\).
- Gradient aggregation: reuse recently computed gradient information
- Example: stochastic variance reduced gradient (SVRD):
\[
\nabla \phi(\beta) \approx \nabla \phi_{i}(\beta)-\left(\nabla \phi_{i}(\bar{\beta})-\nabla \phi(\bar{\beta})\right)
\]
- \(\bar{\beta}\) : variables the last time the true batch gradient was computed

Improving the convergence rate of stochastic methods

Second-order methods: use sampled Hessian information
- Subsampled Hessian-Free Newton Methods: minibatch estimate of the Hessian
\[
\nabla^{2} \phi(\beta) \approx \frac{1}{\left|\mathcal{S}_{k}^{H}\right|} \sum_{i \in \mathcal{S}_{k}^{H}} \nabla^{2} \phi_{i}(\beta)
\]
- Can also enforce positive definiteness via subsampled Gauss-Newton approximations - Subsampled Quasi-Newton Methods:
- typical quasi-Newton methods with stochastic estimates of the gradient```

