Applied Machine Learning
Lecture 5-1: Optimization in machine learning

Selpi (selpi@chalmers.se)

The slides are further development of Richard Johansson's slides

February 7, 2020

Overview

Optimization in machine learning

SGD for training a linear regression model

introduction to PyTorch

regularized linear regression models

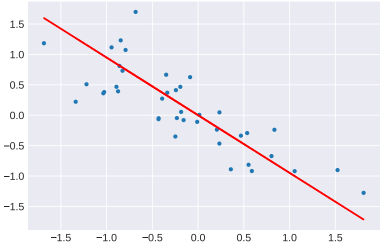
Objective functions for machine learning

- in this lecture, we'll discuss **objective functions** for our models
- training a model = minimizing (or maximizing) this objective
- often, the objective consists of a combination of
  - a **loss function** that measures how well the model fits the training data
  - a **regularizer** that measures how simple/complex the model is

minimizing squared errors

- in the **least squares** approach to linear regression, we want to minimize the mean (or sum) of **squared errors**

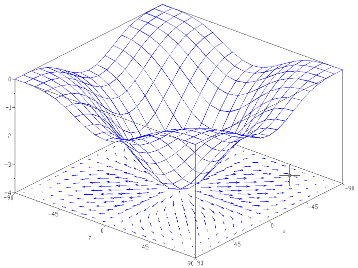
$$f(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{w}\cdot\mathbf{x}_i - y_i)^2$$



- From calculus: if a "nice" function has a maximum or minimum, then the derivative will be zero there

the gradient

- the multidimensional equivalent of the derivative is called the **gradient**
- if f is a function of n variables, then the gradient is an n-dimensional vector, often written $\nabla f(x)$
- intuition: the gradient points in the uphill direction



- again: the gradient is zero if we have an optimum
- this intuition leads to a simple idea for finding the minimum:
  - take a small step in the direction opposite to the gradient
  - repeat until the gradient is close enough to zero

Gradient descent, pseudocode

- the same thing again, in pseudocode:
  1. set x to some initial value, and select a suitable step size η
  2. compute the gradient $\nabla f(x)$
  3. if $\nabla f(x)$ is small enough, we are done
  4. otherwise, subtract $\eta \cdot \nabla f(x)$ from x and go back to step 2
- conversely, to find the maximum we can do **gradient ascent**: then we instead add $\eta \cdot \nabla f(x)$ to x

Overview

Optimization in machine learning

SGD for training a linear regression model

introduction to PyTorch

regularized linear regression models

stochastic gradient descent (SGD)

- in machine learning, our objective functions are often defined as sums over the whole training set, such as the least squares loss function

$$f(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{w} \cdot \boldsymbol{x}_i - y_i)^2$$

- if N is large, it would take time to use gradient descent to minimize the loss function
  - because gradient descent takes into account all the instances in the training set
- **stochastic gradient descent**: simplify the computation by computing the gradient using just a small part
  - in the extreme case, a **single** training instance
  - or **minibatch**: a few instances

SGD: pseudocode

1. set w to some initial value, and select a suitable step size η
2. select a single training instance x
3. compute the gradient $\nabla f(w)$ **using x only**
4. if we are "done", stop
5. otherwise, subtract $\eta \cdot \nabla f(w)$ from w and go back to step 2

SGD: pseudocode

1. set w to some initial value, and select a suitable step size η
2. select a single training instance x
3. compute the gradient $\nabla f(w)$ **using x only**
4. if we are "done", stop
5. otherwise, subtract $\eta \cdot \nabla f(w)$ from w and go back to step 2

$\boldsymbol{w} = (0, \ldots, 0)$
for $(\boldsymbol{x}_i, y_i) \ldots$
  compute gradient ∇f_i for current instance $(\boldsymbol{x}_i, y_i)$
  $\boldsymbol{w} = \boldsymbol{w} - \eta \cdot \nabla f_i(\boldsymbol{w})$
return $\boldsymbol{w}$

when to terminate SGD?

- simple solution: fixed number of iterations
- or stop when we've seen no improvement for some time
- or evaluate on a held-out set: **early stopping**

applying SGD to the least squares loss

- the least squares loss function:

$$f(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{w} \cdot \boldsymbol{x}_i - y_i)^2$$

- let's consider just a single instance:

$$f_i(\boldsymbol{w}) = (\boldsymbol{w} \cdot \boldsymbol{x}_i - y_i)^2$$

- for this instance, the gradient of the least squares loss with respect to $\boldsymbol{w}$ is

$$\nabla f_i(\boldsymbol{w}) = 2 \cdot (\boldsymbol{w} \cdot \boldsymbol{x}_i - y_i) \cdot \boldsymbol{x}_i$$

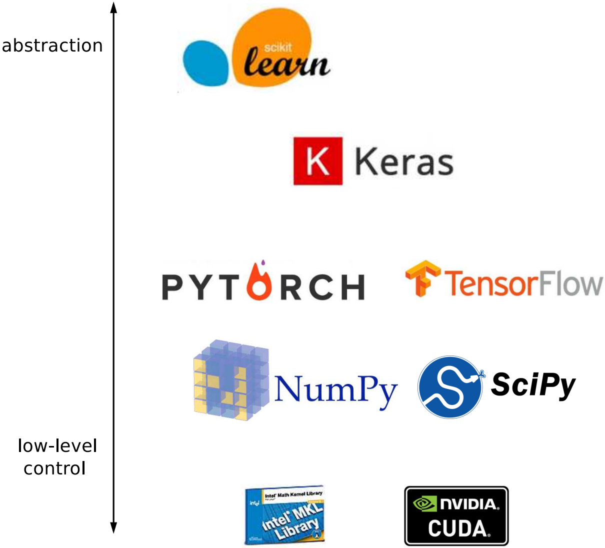
plugging it back into SGD $\Rightarrow$ Widrow-Hoff again!

$\boldsymbol{w} = (0, \ldots, 0)$
for $(\boldsymbol{x}_i, y_i)$ ...
$\nabla f_i(\boldsymbol{w}) = 2 \cdot \text{error} \cdot \boldsymbol{x}_i$
$\boldsymbol{w} = \boldsymbol{w} - \eta \cdot \nabla f_i(\boldsymbol{w})$
return $\boldsymbol{w}$

Overview

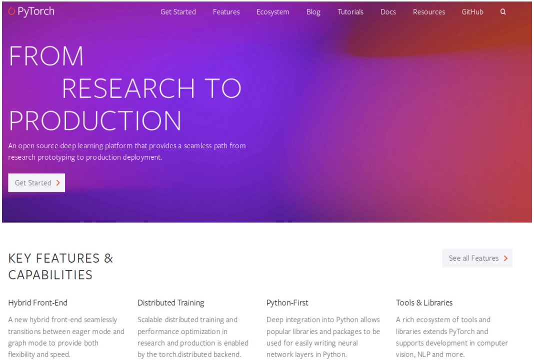
- Optimization in machine learning
- SGD for training a linear regression model
- introduction to PyTorch
- regularized linear regression models

the Python machine learning ecosystem (selection)



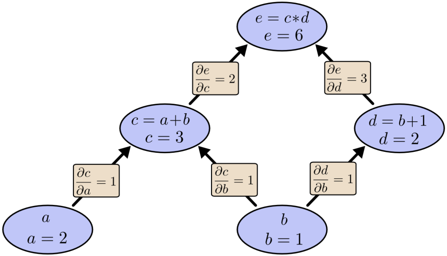
finding and installing PyTorch

- easy to install using conda
- `https://pytorch.org/`



what is the purpose of PyTorch?

- on a high level: a library for prototyping ML models
- on a low level: NumPy-like with **automatic gradients**
- easy to move computations to a GPU (if available)



[source]

Overview

Optimization in machine learning

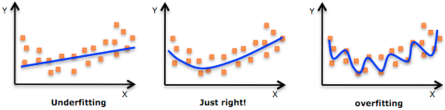
SGD for training a linear regression model

introduction to PyTorch

regularized linear regression models

recall: the fundamental tradeoff in machine learning

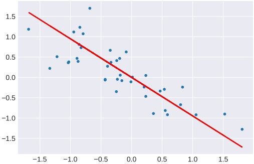
- **goodness of fit**: the learned model should describe the examples in the training data
- **regularization**: the model should be simple



[Image source]

- the least squares loss just takes care of the first part!

$$f(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{w}\cdot\mathbf{x}_i - y_i)^2$$



what does it mean to “keep the model simple”?

- concretely, how can we add regularization to the linear regression model?
- the most common approach is to add a term that **keeps the weights small**
- for instance, by penalizing the squared length (norm) of the weight vector should be small:

$$\|\mathbf{w}\|^2 = w_1 \cdot w_1 + \ldots + w_n \cdot w_n = \mathbf{w} \cdot \mathbf{w}$$

- this is called a L_2 **regularizer** (or a L_2 penalty)

combining the pieces

- we combine the loss and the regularizer:

$$\frac{1}{N}\sum_{i=1}^{N}\text{Loss}(\boldsymbol{w}, \boldsymbol{x}_i, y_i) + \alpha \cdot \text{Regularizer}(\boldsymbol{w})$$

$$= \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{w} \cdot x_i - y_i)^2 + \alpha \cdot \|\boldsymbol{w}\|^2$$

- in this formula, α is a "tweaking" parameter that controls the tradeoff between loss and regularization

ridge regression

- the combination of least squares loss with L_2 regularization is called **ridge regression**

$$\frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{w} \cdot x_i - y_i)^2 + \alpha \cdot \|\boldsymbol{w}\|^2$$

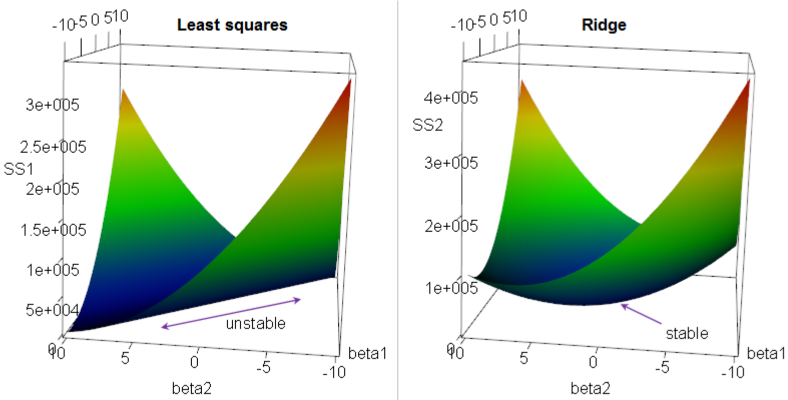
- in scikit-learn: `sklearn.linear_model.Ridge`
- (unregularized least squares: `sklearn.linear_model.LinearRegression`)

discussion

$$\frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{w}\cdot x_i - y_i)^2 + \alpha \cdot \|\boldsymbol{w}\|^2$$

- what happens if we set α to 0?
- what happens if we set α to a huge number?

why "ridge"?



[Image source]

SGD for the ridge regression model

$$\frac{1}{N}\sum(\boldsymbol{w}\cdot x_i - y_i)^2 + \alpha\cdot\|\boldsymbol{w}\|^2$$

- the gradient with respect to a single training instance is

$$\nabla f_i(\boldsymbol{w}) = 2\cdot\text{error}\cdot\boldsymbol{x}_i + 2\cdot\alpha\cdot\boldsymbol{w}$$

$\boldsymbol{w} = (0,\ldots,0)$
for $(\boldsymbol{x}_i, y_i)$ in the training set
  $g = \boldsymbol{w}\cdot\boldsymbol{x}_i$
  $\text{error} = g - y_i$
  $\boldsymbol{w} = \boldsymbol{w} - \eta\cdot(\text{error}\cdot\boldsymbol{x}_i + \alpha\cdot\boldsymbol{w})$
return $\boldsymbol{w}$

the Lasso and ElasticNet models

- another common regularizer is the **L_1 norm**

$$\|\boldsymbol{w}\|_1 = |w_1| + \ldots + |w_n|$$

- the L_1 norm gives the **Lasso** model
- combination of L_1 and L_2 gives the **ElasticNet** model
- in scikit-learn:
  - `sklearn.linear_model.Lasso`
  - `sklearn.linear_model.ElasticNet`

why does Lasso give zero coefficients?

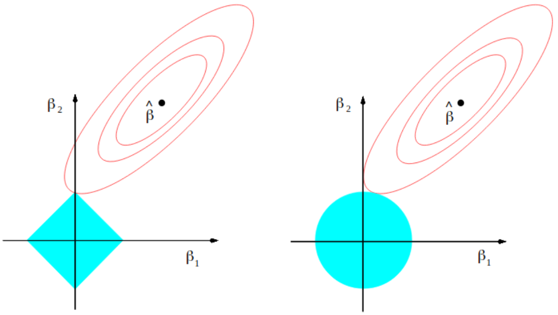


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions $|\beta_1| + |\beta_2| \leq t$ and $\beta_1^2 + \beta_2^2 \leq t^2$, respectively, while the red ellipses are the contours of the least squares error function.*

- figure from Hastie et al. book