Machine learning from scratch

Lecture 5: Linear regression: Recap

Alexis Zubiolo
`alexis.zubiolo@gmail.com`

Data Science Team Lead @ Adcash

February 2, 2017

Information

All the code that implement what is in this presentation can be found on the GitHub repository (lecture 4, `least-squares.ipynb`).

Information

All the code that implement what is in this presentation can be found on the GitHub repository (lecture 4, `least-squares.ipynb`).

The goal of this lecture is to recap what we have seen and list the lessons to remember.

OLS recap

Data: features $x \in \mathcal{X}$ (size and intercept), labels $y \in \mathcal{Y}$ (prices)
Linear model: $\hat{y} = h(x) = \theta^T x$
Loss function:

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

Gradient:

$$\nabla J(\theta) = \left[\frac{\partial}{\partial \theta_1} J(\theta), \ldots, \frac{\partial}{\partial \theta_d} J(\theta) \right]^T$$

where

$$\frac{\partial}{\partial \theta_j} J(\theta) = \sum_{i=1}^{d} \left(h\left(\boldsymbol{x}^{(i)} \right) - y^{(i)} \right) x_j^{(i)}$$

Update rule:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Batch vs. Stochastic gradient descent

The update rule:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

where

$$\frac{\partial}{\partial \theta_j} J(\theta) = \sum_{i=1}^{d} \left(h\left(\boldsymbol{x}^{(i)}\right) - y^{(i)} \right) x_j^{(i)}$$

is called **batch update gradient descent** because the gradient is computed on **the whole training set**.

Batch vs. Stochastic gradient descent

The update rule:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

where

$$\frac{\partial}{\partial \theta_j} J(\theta) = \sum_{i=1}^{d} \left(h\left(\boldsymbol{x}^{(i)}\right) - y^{(i)} \right) x_j^{(i)}$$

is called **batch update gradient descent** because the gradient is computed on **the whole training set**. For some i, the update rule

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J^{(i)}(\theta)$$

where

$$\frac{\partial}{\partial \theta_j} J^{(i)}(\theta) = \left(h\left(\boldsymbol{x}^{(i)}\right) - y^{(i)} \right) x_j^{(i)}$$

is called **stochastic gradient descent** because the gradient is computed on **a single sample**.

Rescaling the data

In our example:

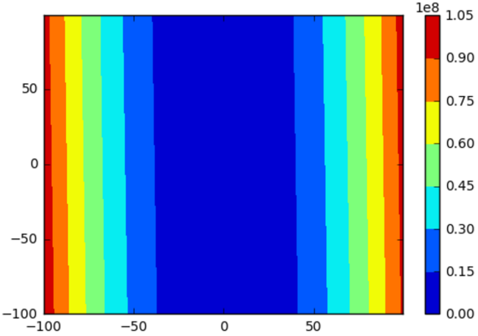
- The intercept is constantly equal to 1
- The size of the house varies between ≈ 10 and ≈ 100

Rescaling the data

In our example:

- The intercept is constantly equal to 1
- The size of the house varies between $\approx$ 10 and $\approx$ 100

Hence, there is a 10-100 difference in order of magnitude between the 2 variables, and the loss function looks like:



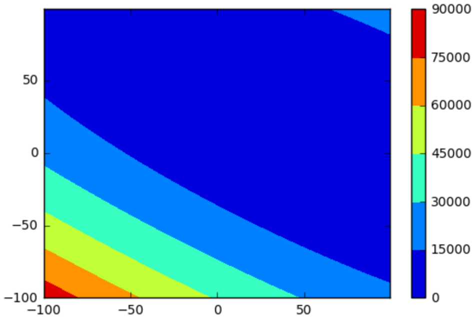
which means the function varies much faster over an axis ($\theta_1 =$ price) than the other ($\theta_0 =$ intercept).

Rescaling the data

To avoid this issue, rescaling (or standardizing) the data can help. One way to do it is:

$$z = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$$

and replace x by z in J. Hence, we get a loss function that looks like that:

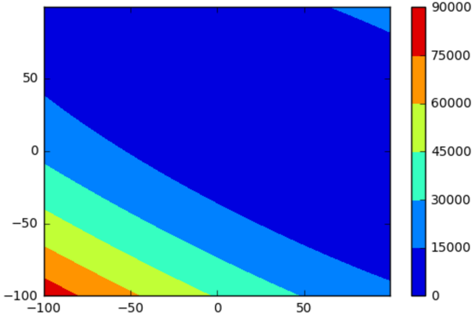


Rescaling the data

To avoid this issue, rescaling (or standardizing) the data can help. One way to do it is:

$$z = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$$

and replace x by z in J. Hence, we get a loss function that looks like that:



Here, both variable have roughly the same importance. In general, it makes the optimization step much more stable and faster.

Learning rate choice

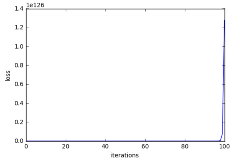
We also saw that the choice of the learning rate was **crucial**. It can be:

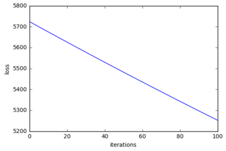
- **Too big** and the optimization can **diverge**
- **Too small** and the optimization can be **slow**
- Properly tuned to have a **fast convergence**

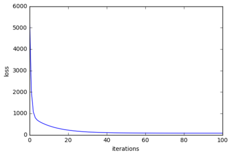
Learning rate choice

We also saw that the choice of the learning rate was **crucial**. It can be:

- **Too big** and the optimization can **diverge**
- **Too small** and the optimization can be **slow**
- Properly tuned to have a **fast convergence**

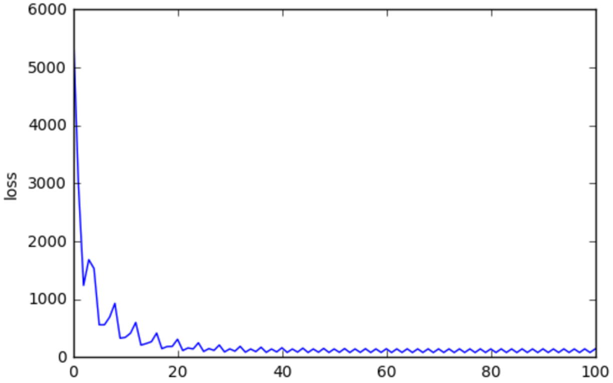






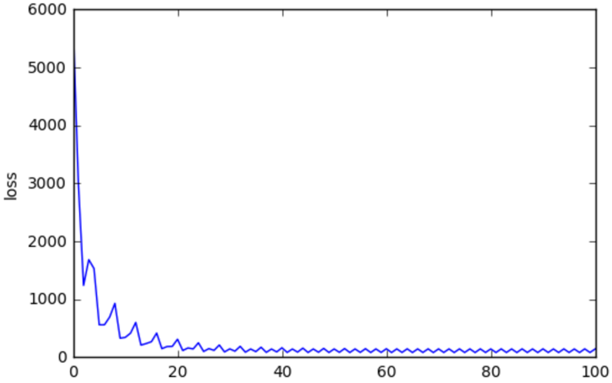
Decreasing learning rate

In stochastic gradient descent, we sometimes saw the loss function **going up and down in the end**:



Decreasing learning rate

In stochastic gradient descent, we sometimes saw the loss function **going up and down in the end**:



This is because the d loss functions $J^{(i)}(\theta)$ for $i = 1, 2, 3$ or 4 have different optimal θ.

Decreasing learning rate

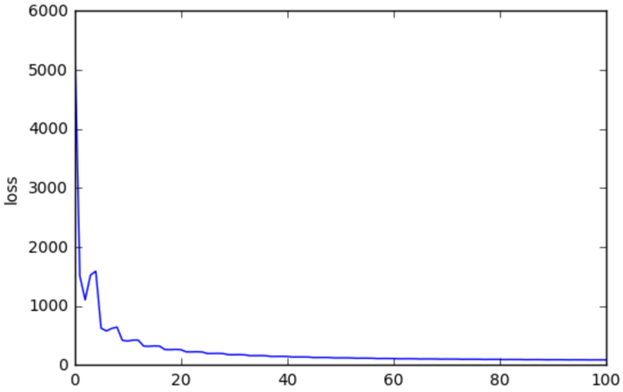
One way to avoid it is to set a **decreasing learning rate** that would go to 0 when the number of the current iteration i goes to infinity. Example

$$\alpha_i = \frac{1}{i+1} \quad \text{or} \quad \alpha_i = \frac{1}{\sqrt{i+1}}$$

Decreasing learning rate

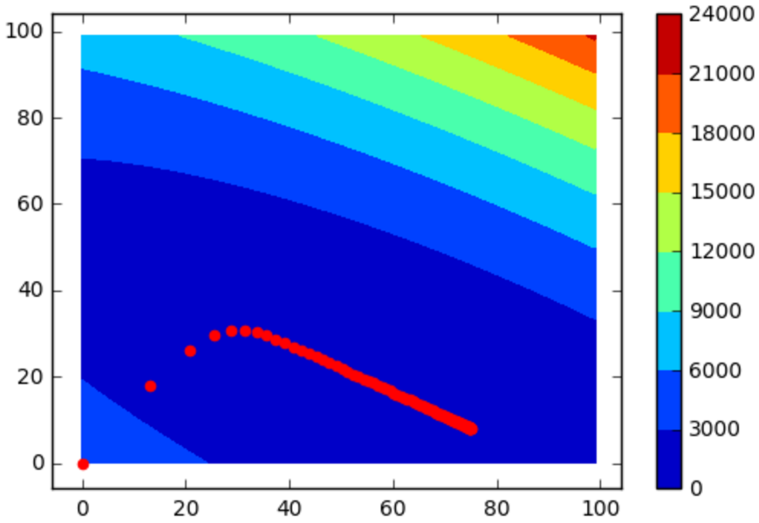
One way to avoid it is to set a **decreasing learning rate** that would go to 0 when the number of the current iteration i goes to infinity. Example

$$\alpha_i = \frac{1}{i+1} \quad \text{or} \quad \alpha_i = \frac{1}{\sqrt{i+1}}$$



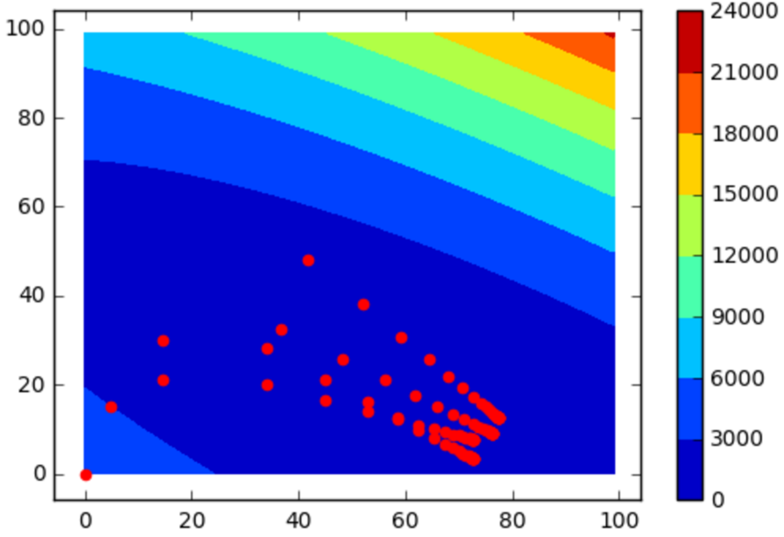
Another way to track the evolution of the optimization

We can plot the loss function (in 2D) and show where θ is at each iteration. Example with the **standard gradient descent**:



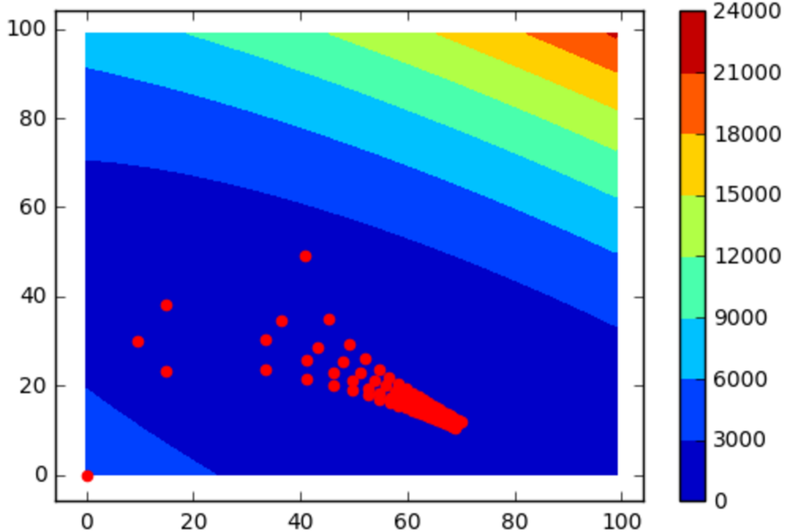
Another way to track the evolution of the optimization

Example with the **stochastic gradient descent with constant learning rate**:



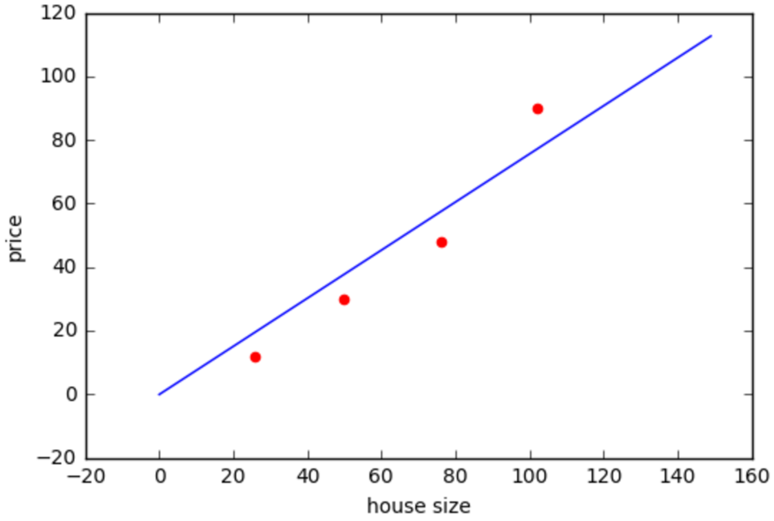
Another way to track the evolution of the optimization

Example with **the stochastic gradient descent with decreasing learning rate**:



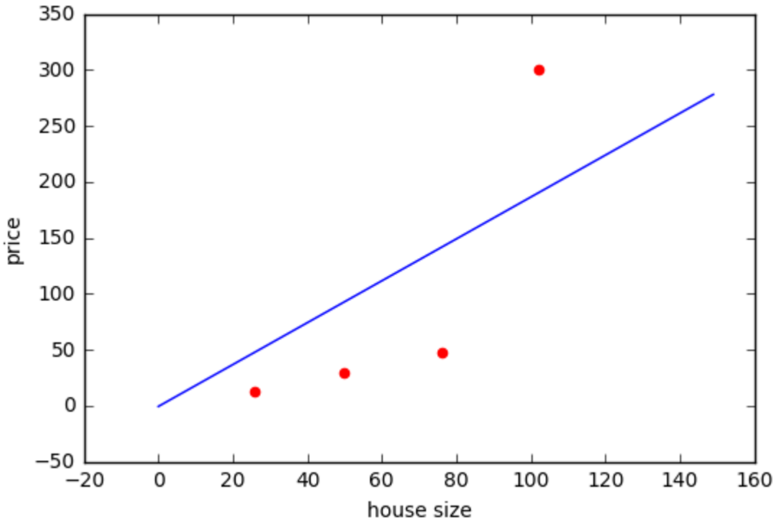
Outliers, overfitting and regularization

If we look at the regression model we've trained, it looks like that
...



Outliers, overfitting and regularization

. . . but if we change the price of the latest house to, say, 300k instead of 90k (in this case it becomes an **outlier**), we would have:



Outliers, overfitting and regularization

This means we are trying too hard to fit the outlier.

Outliers, overfitting and regularization

This means we are trying too hard to fit the outlier.

This phenomenon is called **overfitting**. In this case, we generally notice some of the coefficients are very big. Hence, a way to avoid overfitting is to penalize too big coefficients. This process is often called **regularization** and aims at making models better on unseen data.

Conclusion and next session

We've reviewed the main steps of the ordinary least-squares. The steps are often the same:

- Define a model (linear in OLS)
- Define a loss function
- Optimize the loss function

Conclusion and next session

We've reviewed the main steps of the ordinary least-squares. The steps are often the same:

- Define a model (linear in OLS)
- Define a loss function
- Optimize the loss function

What's next?

- Implementation of regularization
- Implementation of more sophisticated models
- Extension to classification

Conclusion and next session

We've reviewed the main steps of the ordinary least-squares. The steps are often the same:

- Define a model (linear in OLS)
- Define a loss function
- Optimize the loss function

What's next?

- Implementation of regularization
- Implementation of more sophisticated models
- Extension to classification

Please check the lecture 4 repository on GitHub by next time.

Thank you! Questions?

alexis.zubiolo@gmail.com

https://github.com/azubiolo/itstep