Training Models 2: Stochastic Gradient Descent

Glenn Bruns
CSUMB

Learning outcomes

After this lecture you should be able to:

1. Explain three methods for gradient descent:
   - batch gradient descent
   - stochastic gradient descent
   - mini-batch gradient descent
2. Explain how these methods can be used to fit a linear model

Recap

How to use optimization to find the "best" line through a set of training data?

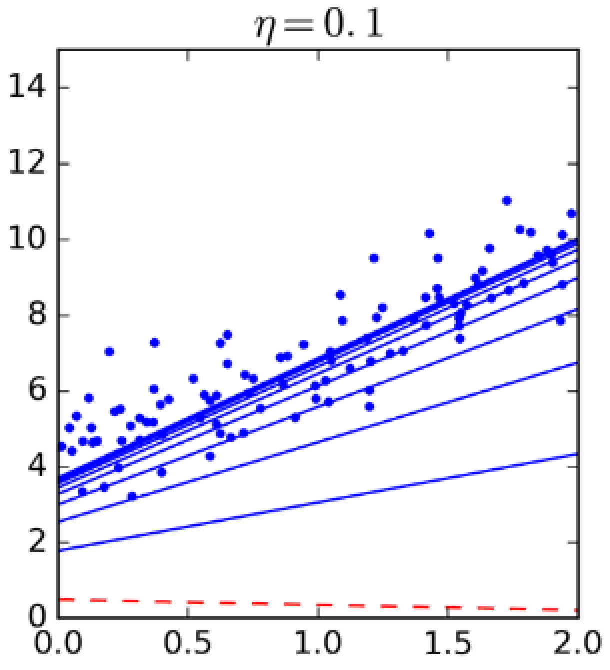
1. We have a linear model, with parameters for slope and y-intercept

2. We have a bunch of training data

3. We define a cost function, where the "cost" is high if the line fits the data poorly

4. We get the best line by finding the parameter values that minimize the cost function

5. We can find the minimum using gradient descent.



Linear regression

Reminder: Geron writes $\mathbf{x}^{(i)}$ for the ith row of matrix X.

MSE cost function for linear regression:

$$MSE(\mathbf{X}, \theta) = \frac{1}{m} \sum_{i=1}^{m} \left(\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)}\right)^2$$

Given training data $\mathbf{X}$, find the θ that *minimizes* $MSE(\mathbf{X}, \theta)$

Partial derivative of the cost function, for some θ_j:

$$\frac{\partial}{\partial \theta_j} MSE(\mathbf{X}, \theta) = \frac{2}{m} \sum_{i=1}^{m} \left(\theta^T \cdot x^{(i)} - y^{(i)}\right) x_j^{(i)}$$

Gradient of the cost function

Partial derivative of the cost function, for some θ_j:

$$\frac{\partial}{\partial \theta_j} MSE(\mathbf{X}, \theta) = \frac{2}{m} \sum_{i=1}^{m} \left(\theta^T \cdot x^{(i)} - y^{(i)}\right) x_j^{(i)}$$

The vector of all the partial derivatives is the gradient of the function:

$$\nabla_\theta \, \mathrm{MSE}(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \mathrm{MSE}(\theta) \\ \frac{\partial}{\partial \theta_1} \mathrm{MSE}(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \mathrm{MSE}(\theta) \end{pmatrix} = \frac{2}{m} \mathbf{X}^T \cdot (\mathbf{X} \cdot \theta - \mathbf{y})$$

Batch gradient descent

```
eta = 0.1  # learning rate
n_iterations = 1000
m = 100

theta = np.random.randn(2,1)  # random initialization

for iteration in range(n_iterations):
    gradients = 2/m * X_b.T.dot(X_b.dot(theta) - y)
    theta = theta - eta * gradients
```

theta that maximizes negative cost =

theta that minimizes cost

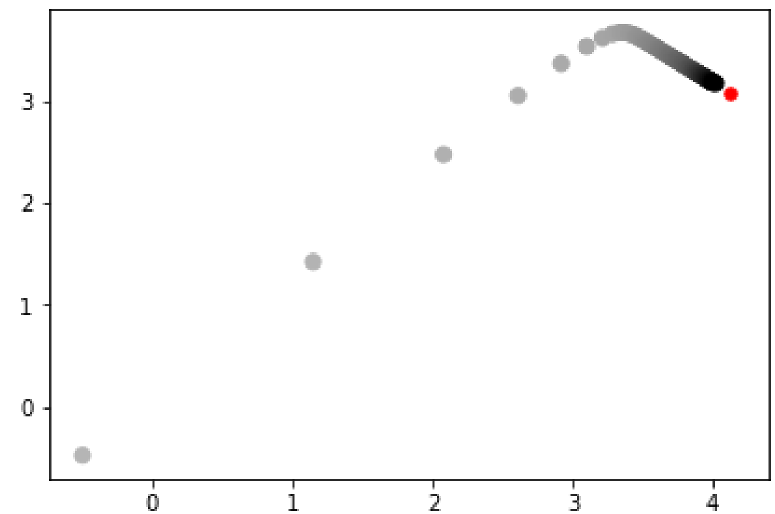
source: Géron, Hands-On Machine Learning text

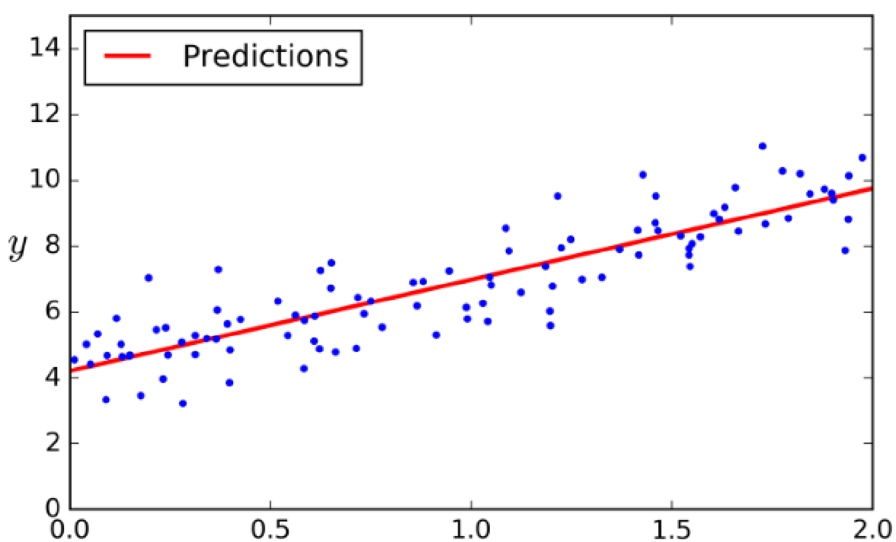
Run it on test data

generate test data:

```
m = 100
theta = np.array([4, 3])
X = 2 * np.random.rand(m, 1)
y = theta[0] + theta[1] * X +
       np.random.randn(m, 1)
```

progress of gradient descent:





source: Géron, Hands-On Machine Learning text

Stochastic gradient descent

Batch gradient descent:

- the definition of the gradient uses all the training data
- at each step, all the data is used in updating theta
- cost of algorithm is high if lots of data

Stochastic gradient descent:

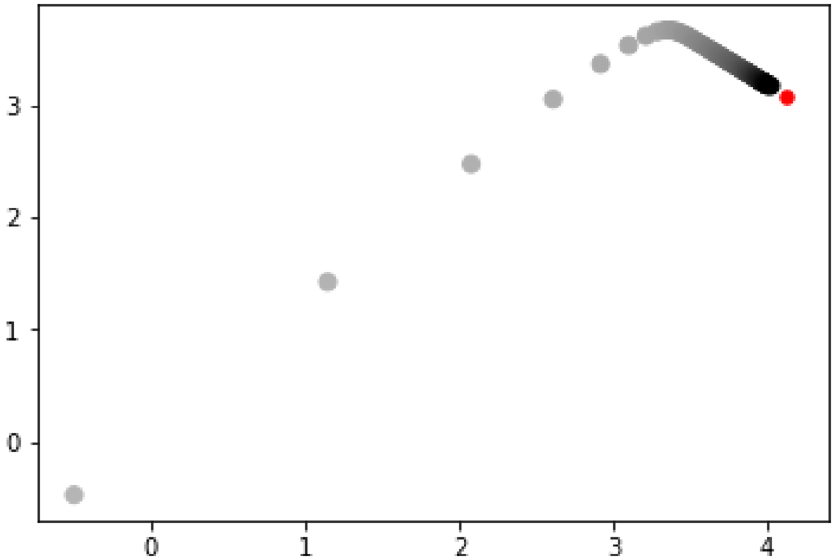
- at each step, one training example is used to update theta
- the example is chosen randomly

Stochastic gradient descent

```
n_epochs = 50
t0, t1 = 5, 50  # learning schedule hyperparameters

def learning_schedule(t):
    return t0 / (t + t1)

theta = np.random.randn(2,1)  # random initialization

for epoch in range(n_epochs):
    for i in range(m):
        random_index = np.random.randint(m)
        xi = X_b[random_index:random_index+1]
        yi = y[random_index:random_index+1]
        gradients = 2 * xi.T.dot(xi.dot(theta) - yi)
        eta = learning_schedule(epoch * m + i)
        theta = theta - eta * gradients
```

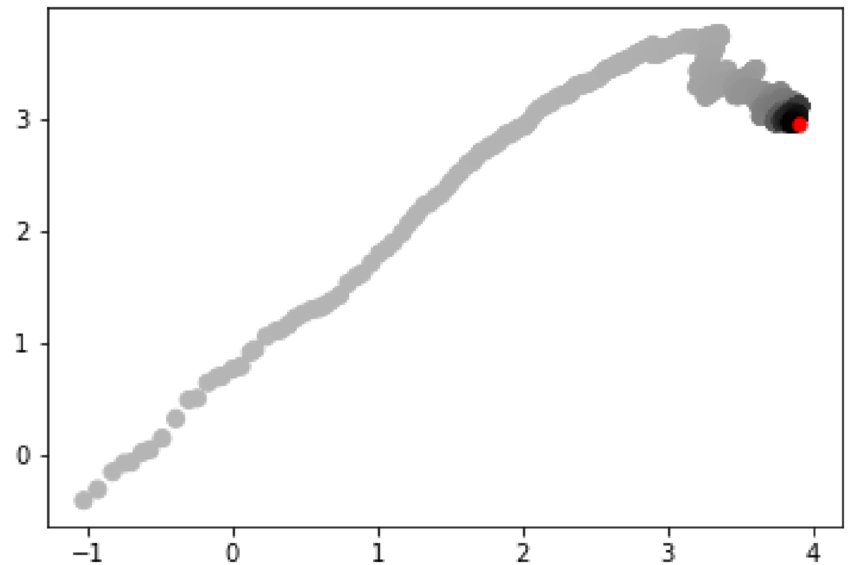
source: Géron, Hands-On Machine Learning text

Run it on test data



batch gradient descent

stochastic gradient descent



Mini-batch gradient descent

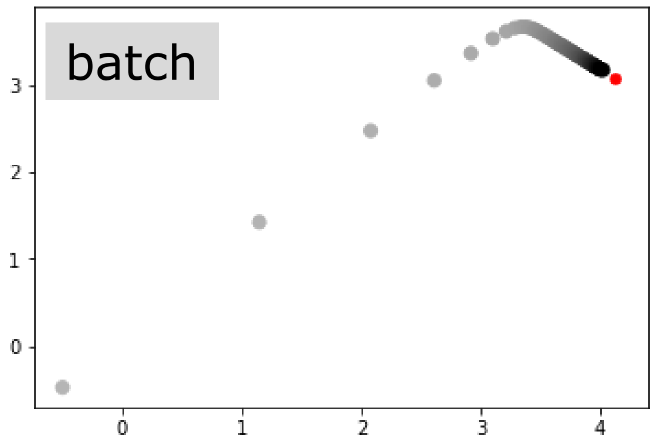
Totally simple if you understand batch and stochastic versions.

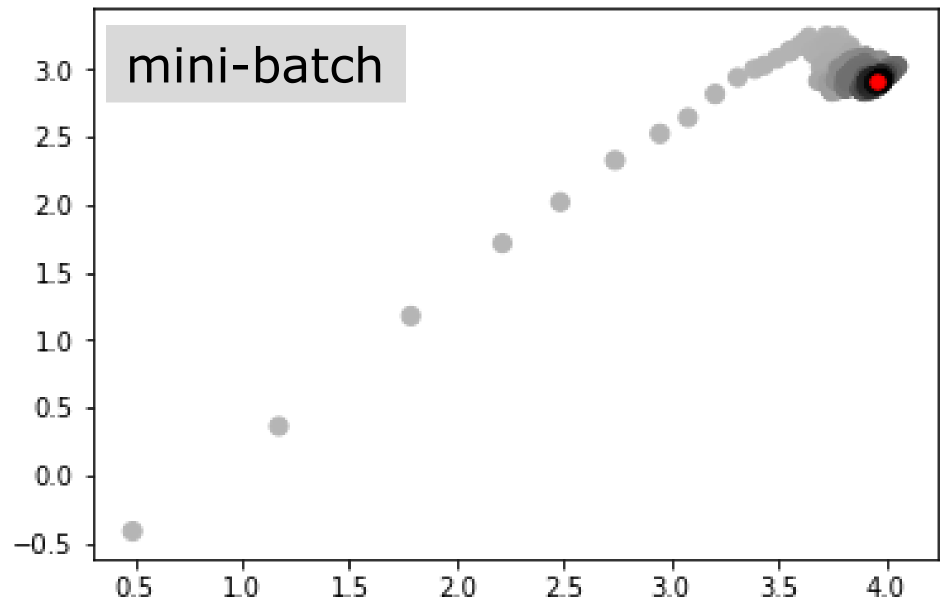
Question: what is a hybrid of the batch and stochastic versions?

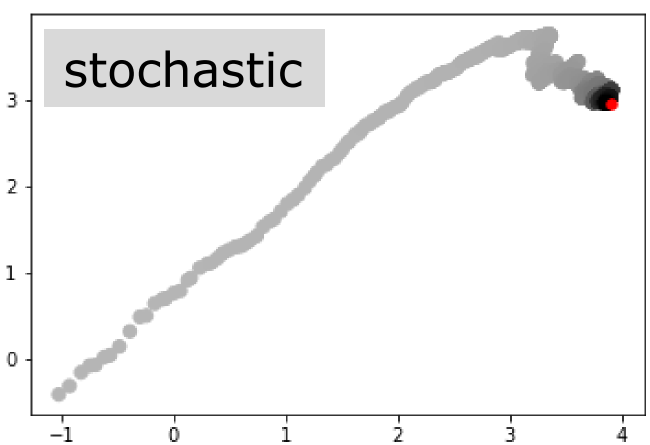
Key observation: gradient calculation can use training data of any size

- batch: each step uses all training data
- stochastic: each step uses 1 random row of training data
- mini-batch: each step uses a small random set of training data

Run it on test data







Comparing methods

method	large m	large n	hyper-params	scaling needed?	Scikit-Learn
normal equation	fast	slow	0	no	LinearRegression
batch GD	slow	fast	2	yes	N/A
stochastic GD	fast	fast	≥ 2	yes	SGDRegressor
mini-batch GD	fast	fast	≥ 2	yes	N/A

m: number of training examples
n: number of features

source: Géron, Hands-On Machine Learning text

Summary

1. We looked at three methods for gradient descent:
   - batch gradient descent
   - stochastic gradient descent
   - mini-batch gradient descent
2. Pros/cons of gradient descent for linear regression:

   \+ handle large datasets, with many features

   \- need scaling; tuning of hyperparameters