Gradient Descent

IN3120/IN4120

Sources

- A great tutorial of the different types of gradient descent
  - http://ruder.io/optimizing-gradient-descent/
- Pedro Domingos. A few useful things to know about machine learning
  - https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf
- https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/
- https://spin.atomicobject.com/2014/06/24/gradient-descent-linear-regression/

What Have We Learned

- Models
  - Linear regression
  - Logistic regression
  - Decision trees
- Evaluating models
  - ROC
  - AUC

- Today: We will start to learn how to train models!

This Lecture

- Loss Functions
- Gradient Descent
- Parameters and hyperparameters
- Regularization

This Lecture

- Loss Functions
- Gradient Descent
- Parameters and hyperparameters
- Regularization

What does it mean to learn?

- ML has three parts:
  - Representation
    - What is the model, what is the landscape of allowed models
  - Evaluation
    - Utility function, loss function, score. How do we compare one model versus another model.
  - Optimization
    - How do we search the space of possible models

What does it mean to learn?

- ML has three parts:
  - Representation
  - Evaluation
  - Optimization

- With your neighbor: Give some examples of each

What does it mean to learn? (Contd.)

Representation	Evaluation	Optimization
Instances	Accuracy/Error rate	Combinatorial optimization
K-nearest neighbor	Precision and recall	Greedy search
Support vector machines	Squared error	Beam search
Hyperplanes	Likelihood	Branch-and-bound
Naive Bayes	Posterior probability	Continuous optimization
Logistic regression	Information gain	Unconstrained
Decision trees	K-L divergence	Gradient descent
Sets of rules	Cost/Utility	Conjugate gradient
Propositional rules	Margin	Quasi-Newton methods
Logic programs		Constrained
Neural networks		Linear programming
Graphical models		Quadratic programming
Bayesian networks		
Conditional random fields		

From: https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf

J

Optimization

Searching the space of possible models

This search can be formulated as an optimization problem

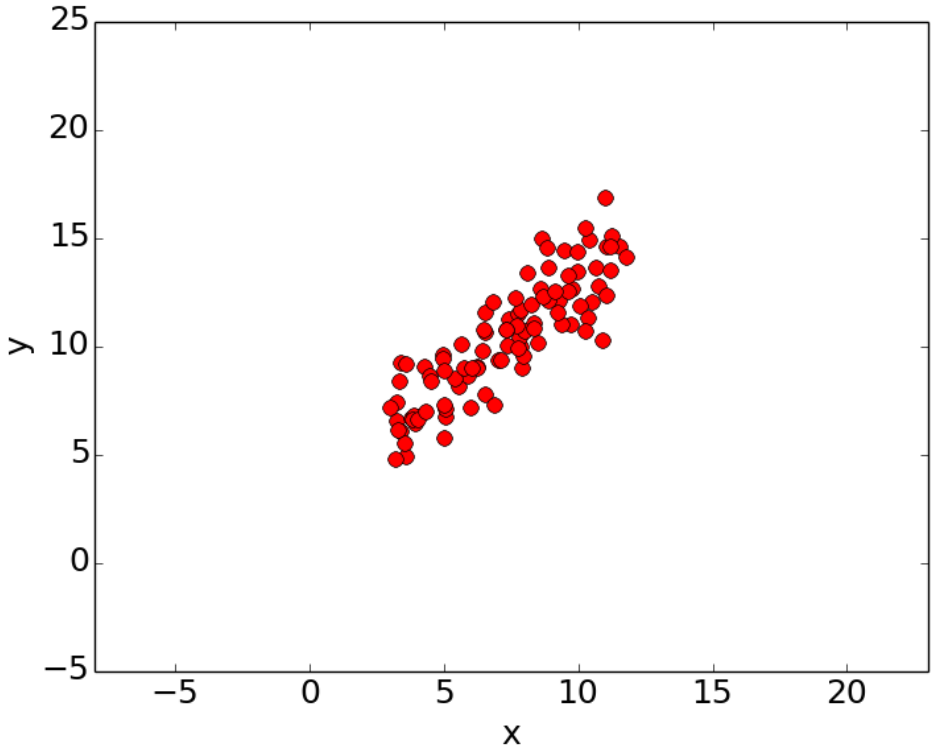
- Create the right function and then minimize it

→ This lecture gives an intuition of this and how to solve a class of these optimization problems

Let's Start Simple: Linear Regression

y=mx+b

$$\text{Error}_{(m,b)} = \frac{1}{N}\sum_{i=1}^{N}(y_i - (mx_i + b))^2$$




```
# y = mx + b
# m is slope, b is y-intercept
def computeErrorForLineGivenPoints(b, m, points):
    totalError = 0
    for i in range(0, len(points)):
        totalError += (points[i].y - (m * points[i].x + b)) ** 2
    return totalError / float(len(points))
```

Material in this and the following slides from https://spin.atomicobject.com/2014/06/24/gradient-descent-linear-regression/

J

Now let's translate this into ML Jargon

- We have our model structure -- linear regression
  - y=mx+b

- We defined what we want to minimize – a *loss function*

$$\text{Error}_{(m,b)} = \frac{1}{N}\sum_{i=1}^{N}(y_i - (mx_i + b))^2$$

- Minimizing the loss function → setting the right parameters in the model
  - Our parameters are m and b

J

Loss Functions

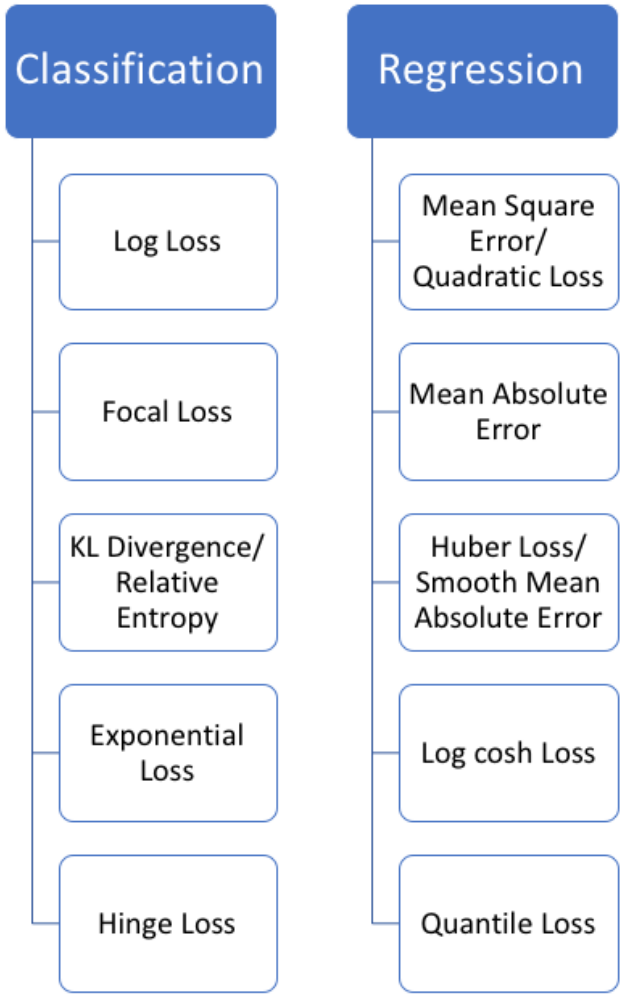
- Two questions:
  - What is the right loss function?
  - How do we actually minimize it?

Loss Functions

- Two questions:
  - What is the right loss function?
  - How do we actually minimize it?

Loss Functions

- What is the right loss function?

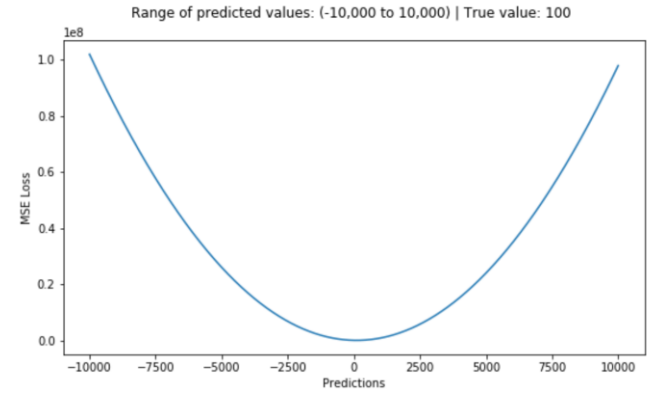


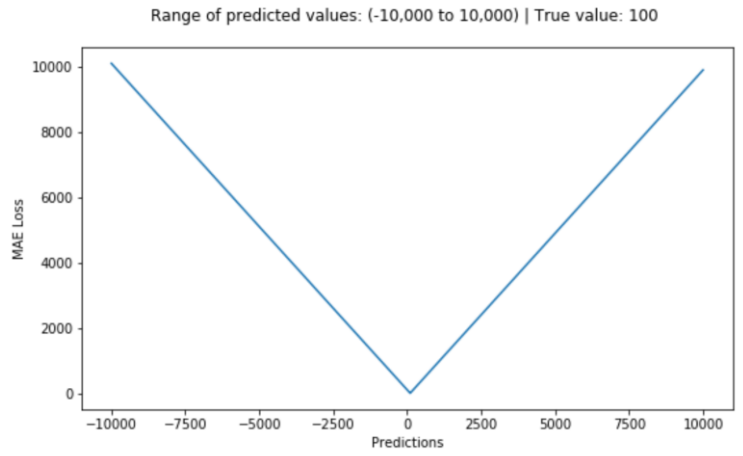
J

Regression Loss

- L2: $$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

- L1: $$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$





Plot of MAE Loss (Y-axis) vs. Predictions (X-axis)

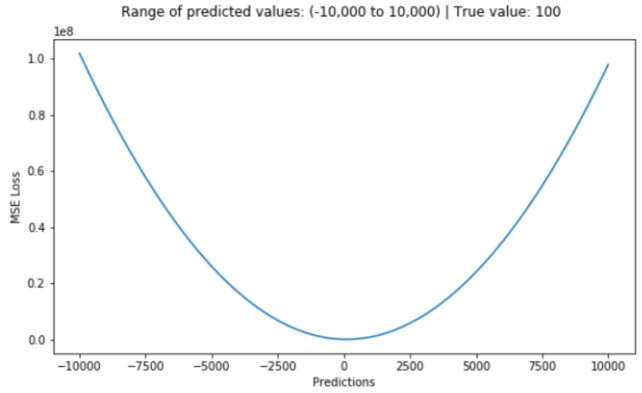
- With your neighbor: What are the tradeoffs between the two?

Images from https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0

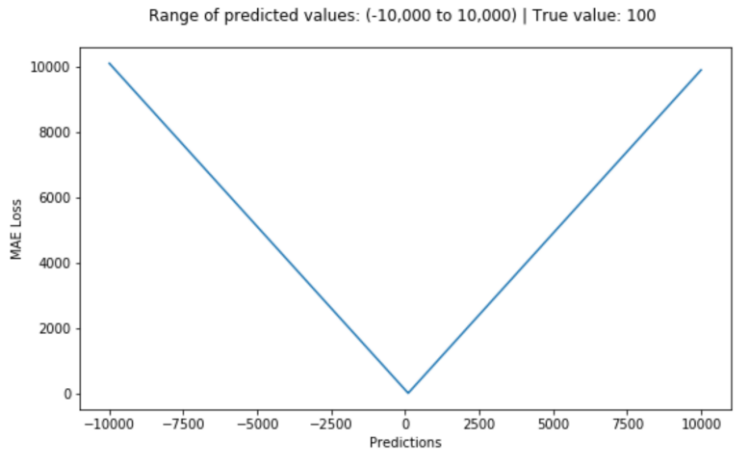
J

Regression Loss

- L2: $$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$



- L1: $$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$



Plot of MAE Loss (Y-axis) vs. Predictions (X-axis)

- *L1 loss is more robust to outliers, but its derivatives are not continuous, making it inefficient to find the solution. L2 loss is sensitive to outliers, but gives a more stable and closed form solution (by setting its derivative to 0.)*

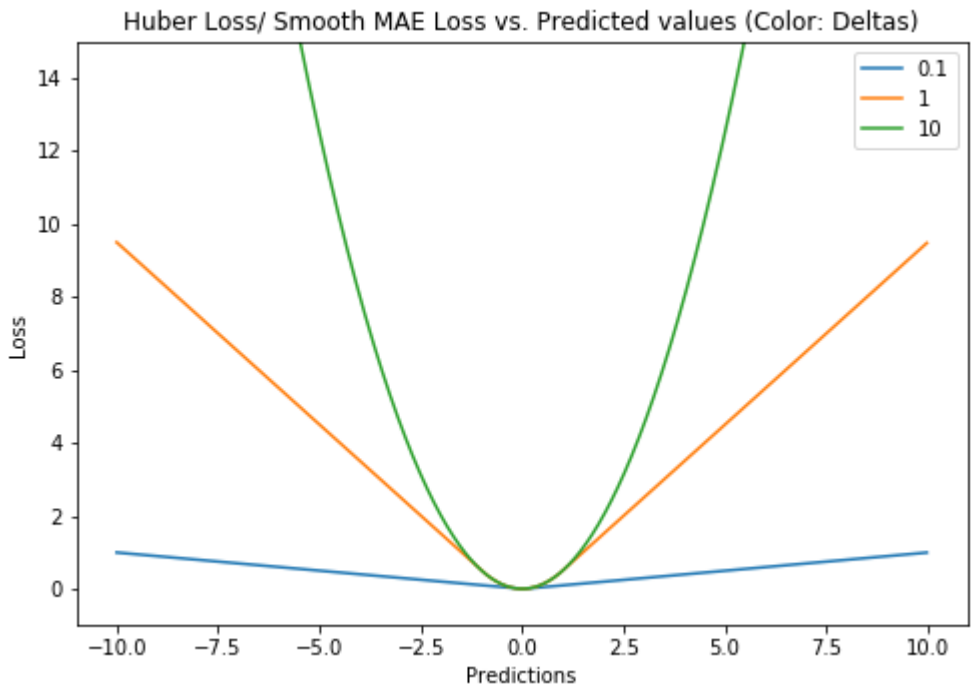
Images from https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0

J

Regression Loss (Contd.)

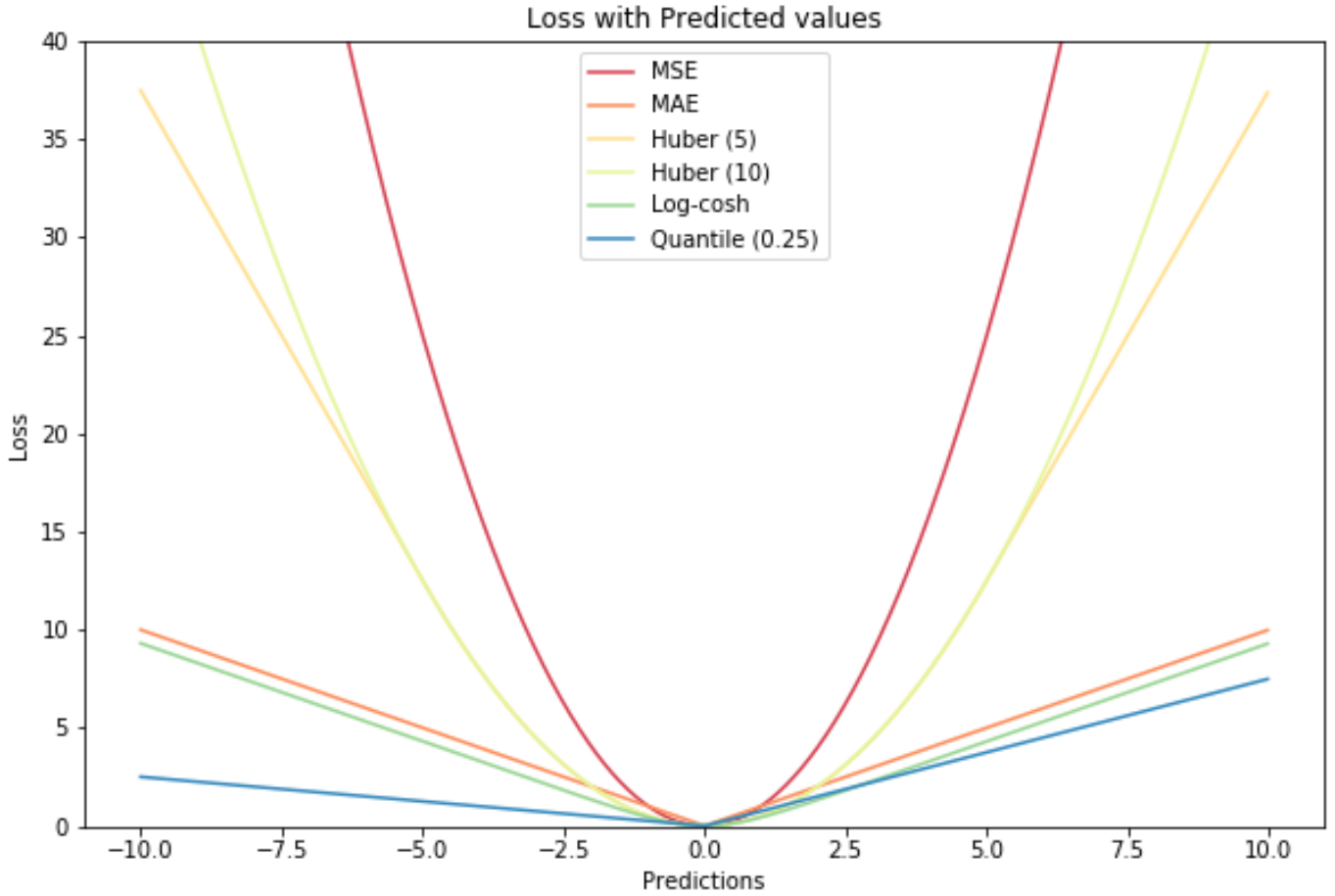
- Huber loss

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta\,|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$



J

And Many Others



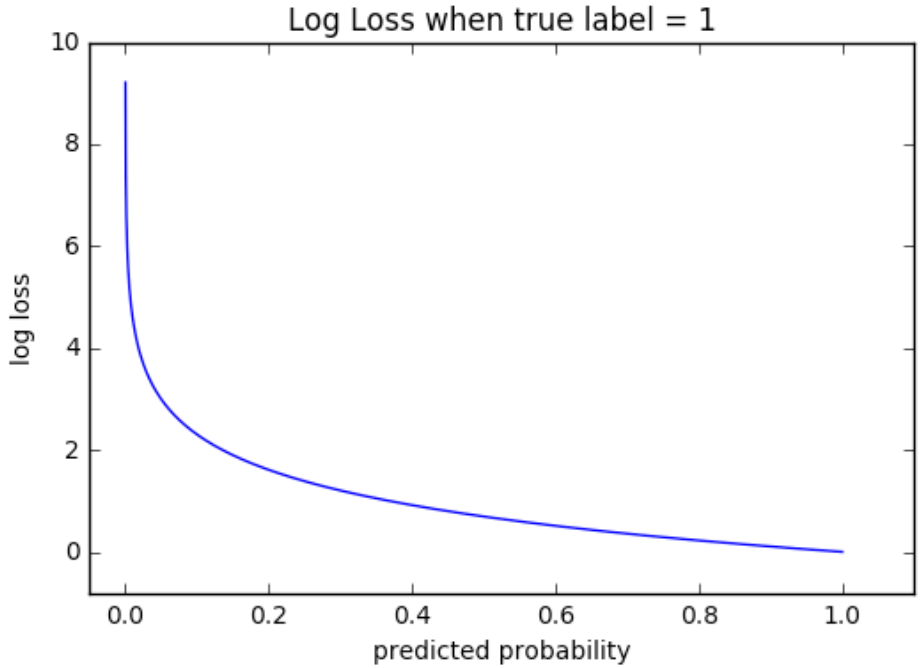
From: https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0

J

Classification Loss Functions

- Cross-entropy

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$



Note

- M - number of classes (dog, cat, fish)
- log - the natural log
- y - binary indicator (0 or 1) if class label c is the correct classification for observation o
- p - predicted probability observation o is of class c

J

Classification Loss Functions: Others

- Likelihood loss
- Hinge loss or multi-class SVM loss
  →We will cover it when we introduce SVMs
- Soft-max classifier
  → We will cover it when we talk about deep nets

Loss Functions

- Two questions:
  - What is the right loss function?
  - How do we actually minimize it?

J

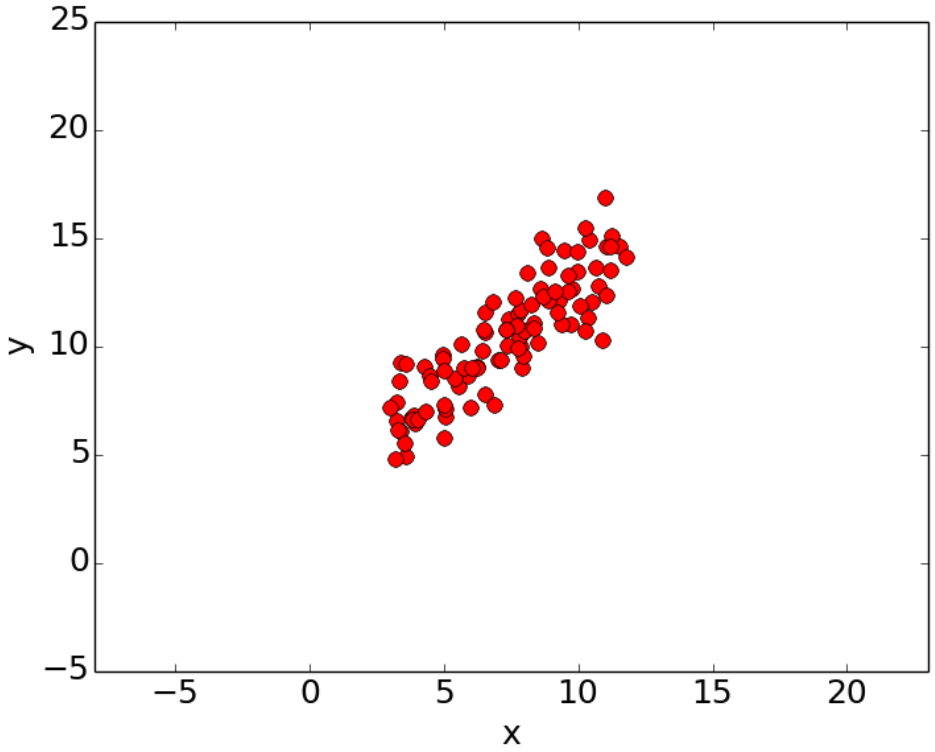
This Lecture

- Loss Functions
- Gradient Descent
- Parameters and hyperparameters
- Regularization

Recall: Linear Regression

y=mx+b

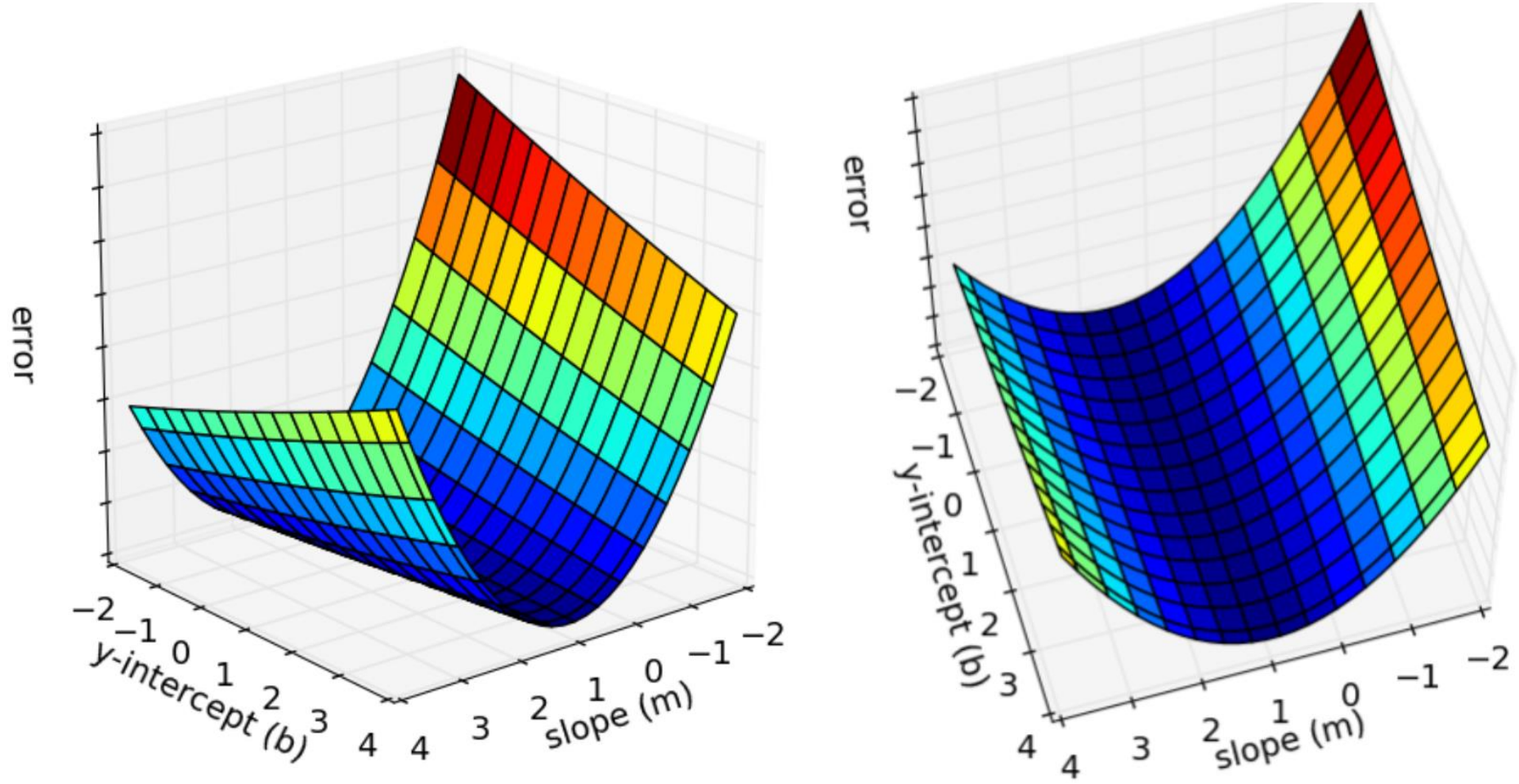
$$\text{Error}_{(m,b)} = \frac{1}{N}\sum_{i=1}^{N}(y_i - (mx_i + b))^2$$




```
# y = mx + b
# m is slope, b is y-intercept
def computeErrorForLineGivenPoints(b, m, points):
    totalError = 0
    for i in range(0, len(points)):
        totalError += (points[i].y - (m * points[i].x + b)) ** 2
    return totalError / float(len(points))
```

Material in this and the following slides from https://spin.atomicobject.com/2014/06/24/gradient-descent-linear-regression/

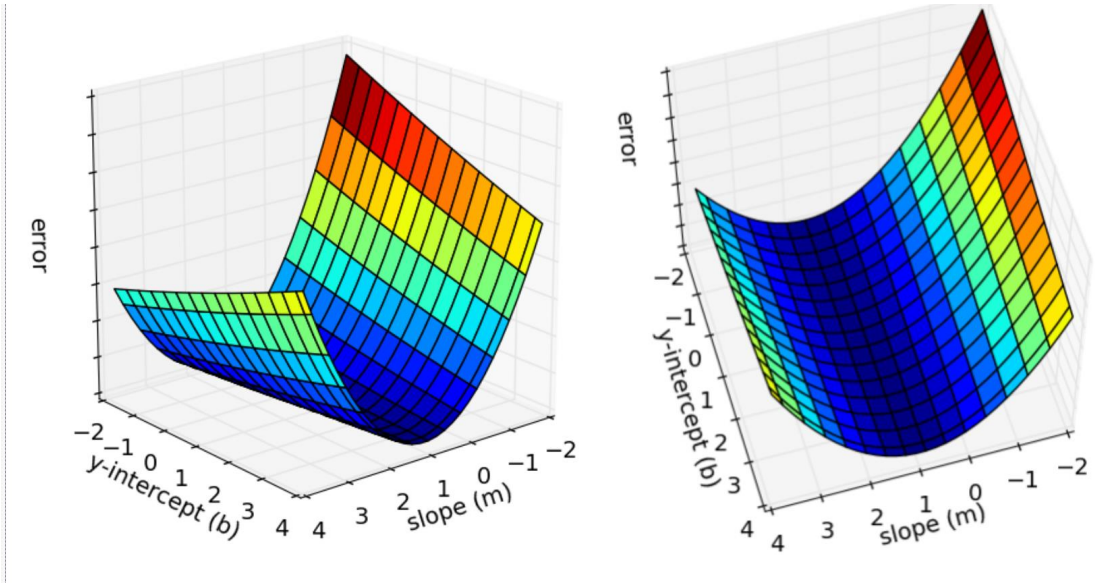
How do we find the “best” b and m?



J

Minimizing the Loss

- Loss: $$\text{Error}_{(m,b)} = \frac{1}{N}\sum_{i=1}^{N}(y_i - (mx_i + b))^2$$



- Minimize it by taking the direction of the gradient

$$\frac{\partial}{\partial \text{m}} = \frac{2}{N}\sum_{i=1}^{N} - x_i(y_i - (mx_i + b))$$

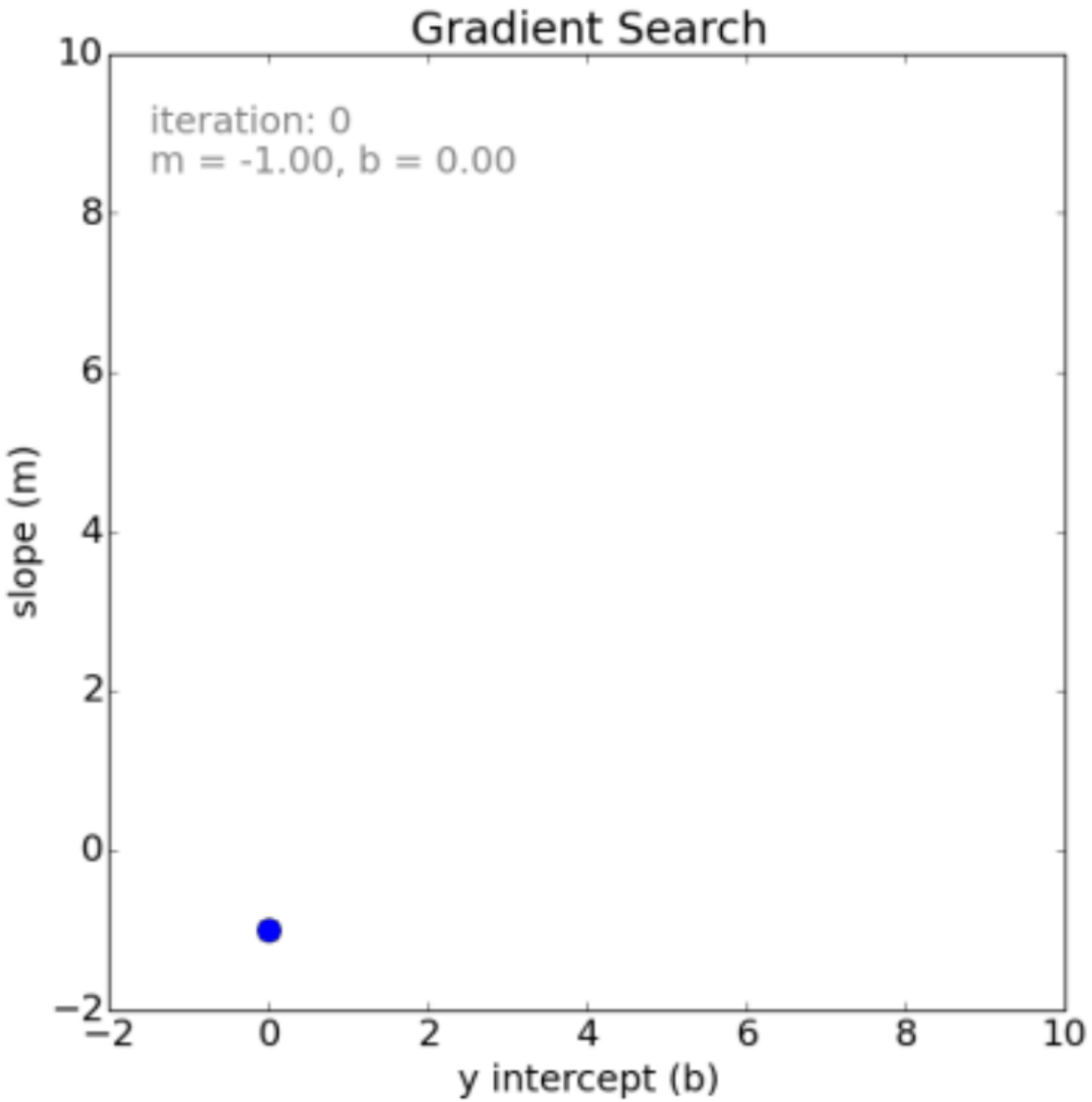
$$\frac{\partial}{\partial \text{b}} = \frac{2}{N}\sum_{i=1}^{N} - (y_i - (mx_i + b))$$

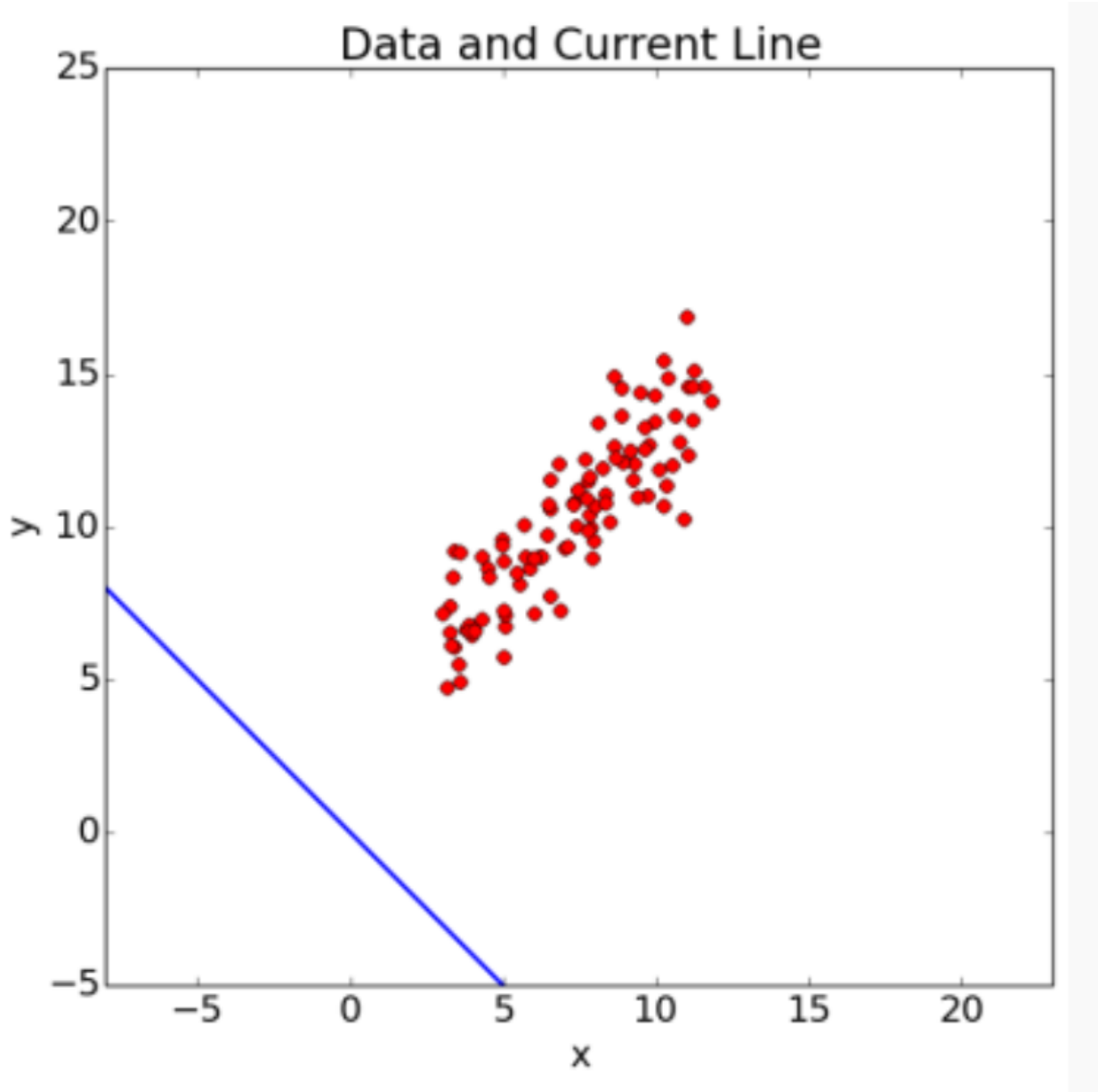
J

Minimization Algorithm: Gradient Descent

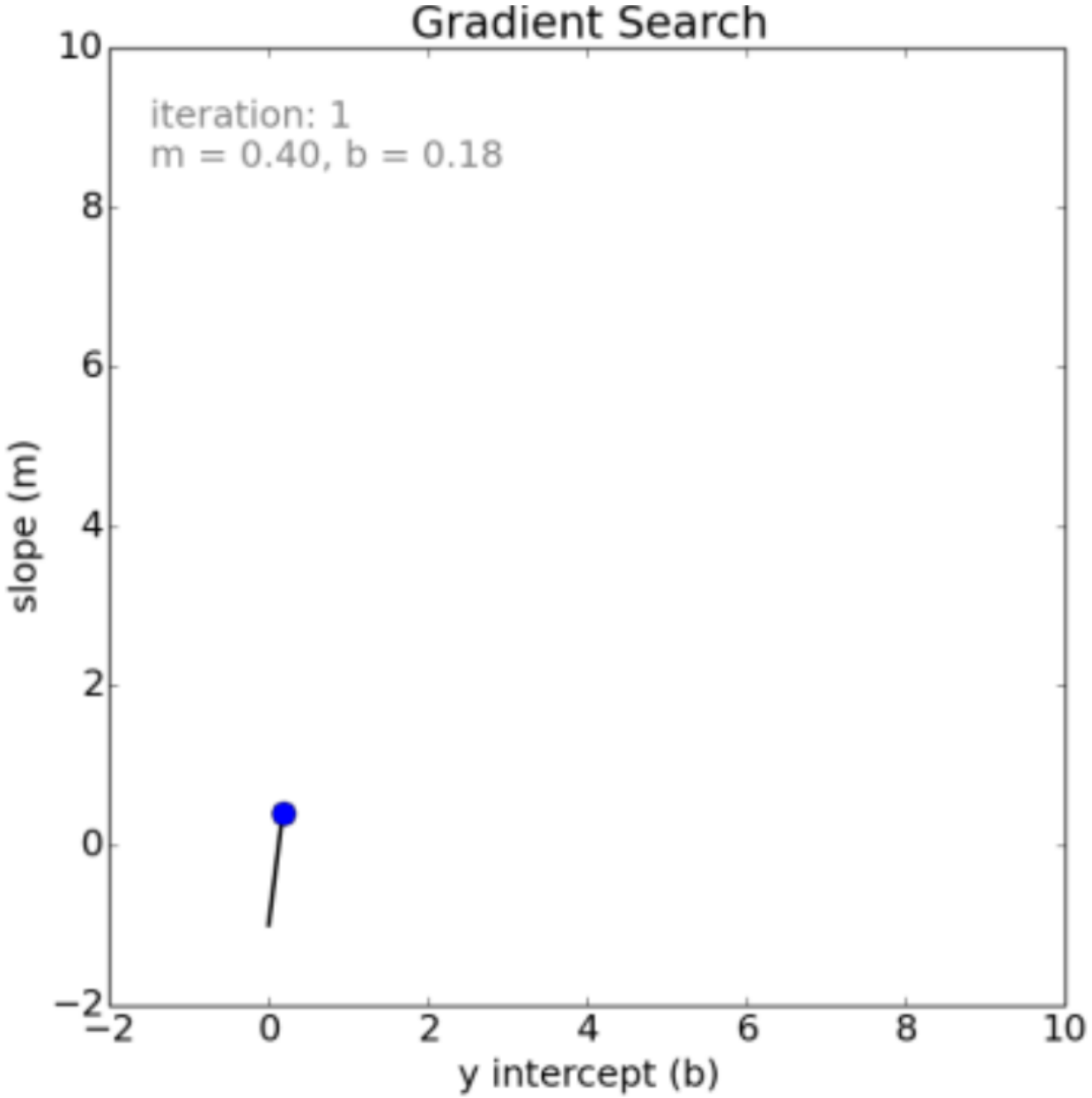
```
1  def stepGradient(b_current, m_current, points, learningRate):
2      b_gradient = 0
3      m_gradient = 0
4      N = float(len(points))
5      for i in range(0, len(points)):
6          b_gradient += -(2/N) * (points[i].y - ((m_current*points[i].x) + b_current))
7          m_gradient += -(2/N) * points[i].x * (points[i].y - ((m_current * points[i].x) + b_current))
8      new_b = b_current - (learningRate * b_gradient)
9      new_m = m_current - (learningRate * m_gradient)
10     return [new_b, new_m]
```

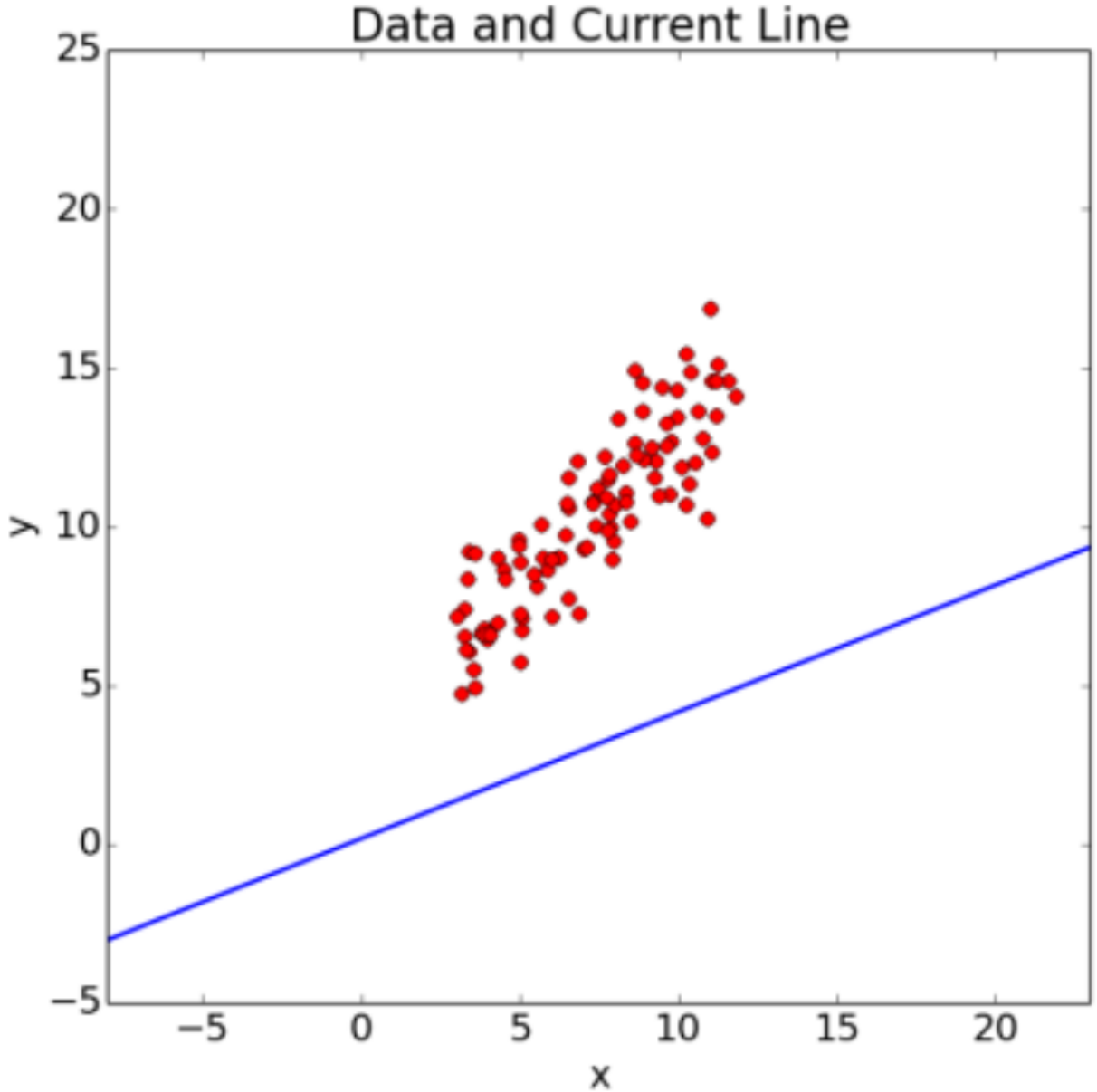
J


Gradient Search
iteration: 0
m = -1.00, b = 0.00
slope (m)
y intercept (b)
10
8
6
4
2
0
−2
−2
0
2
4
6
8
10

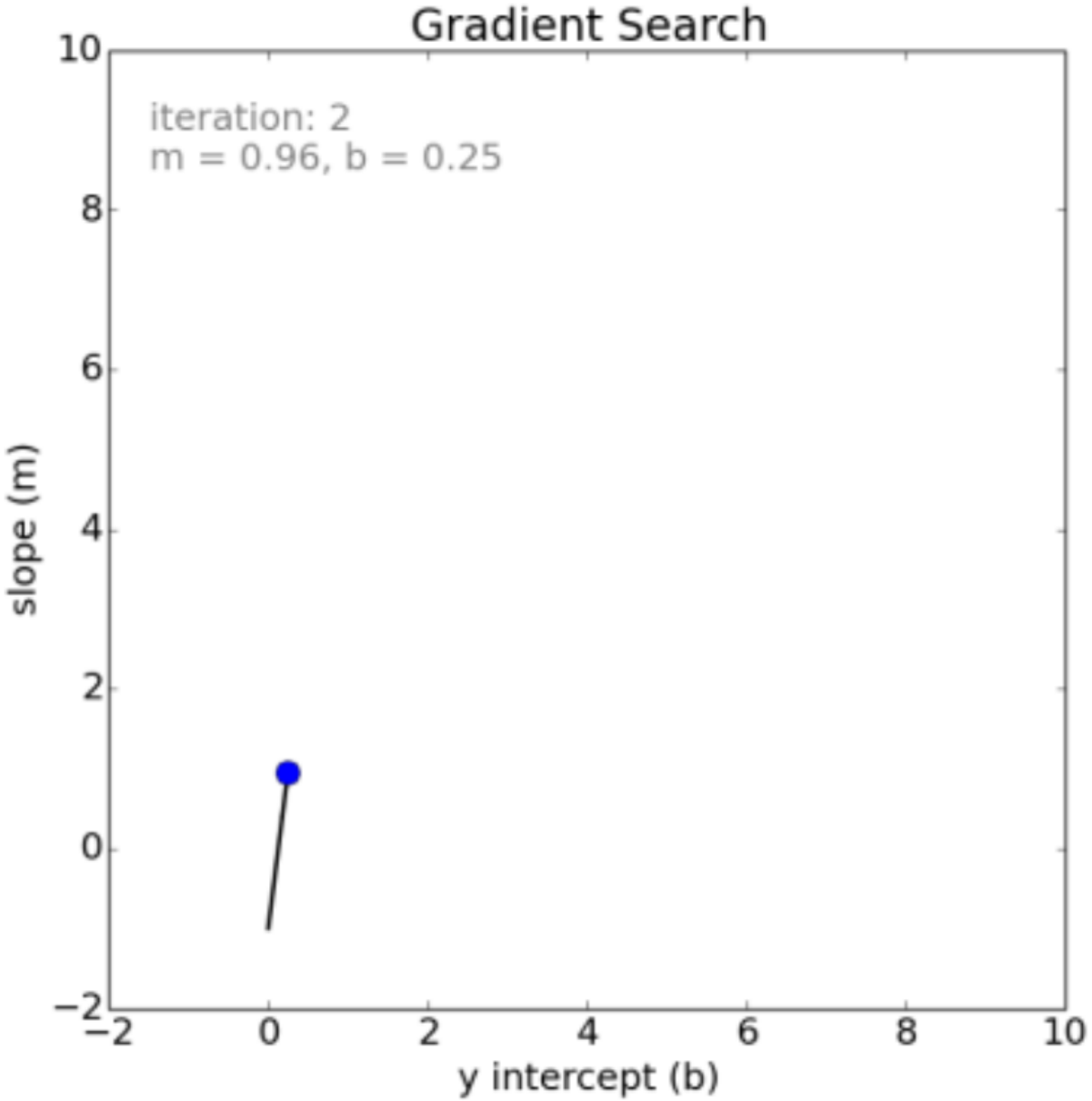

Data and Current Line
y
x
25
20
15
10
5
0
−5
−5
0
5
10
15
20

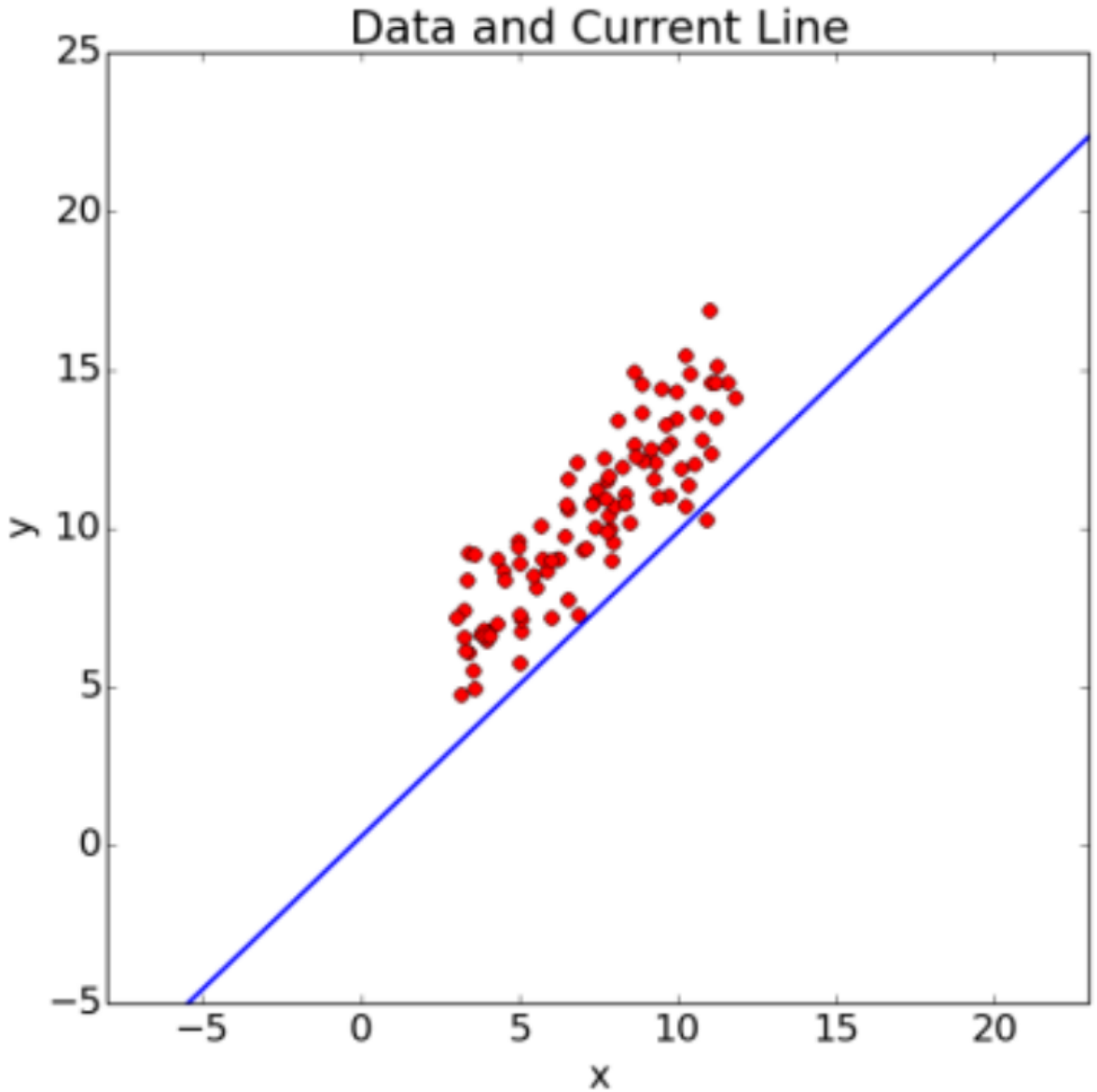
J


Gradient Search
iteration: 1
m = 0.40, b = 0.18
slope (m)
y intercept (b)
10
8
6
4
2
0
−2
−2
0
2
4
6
8
10

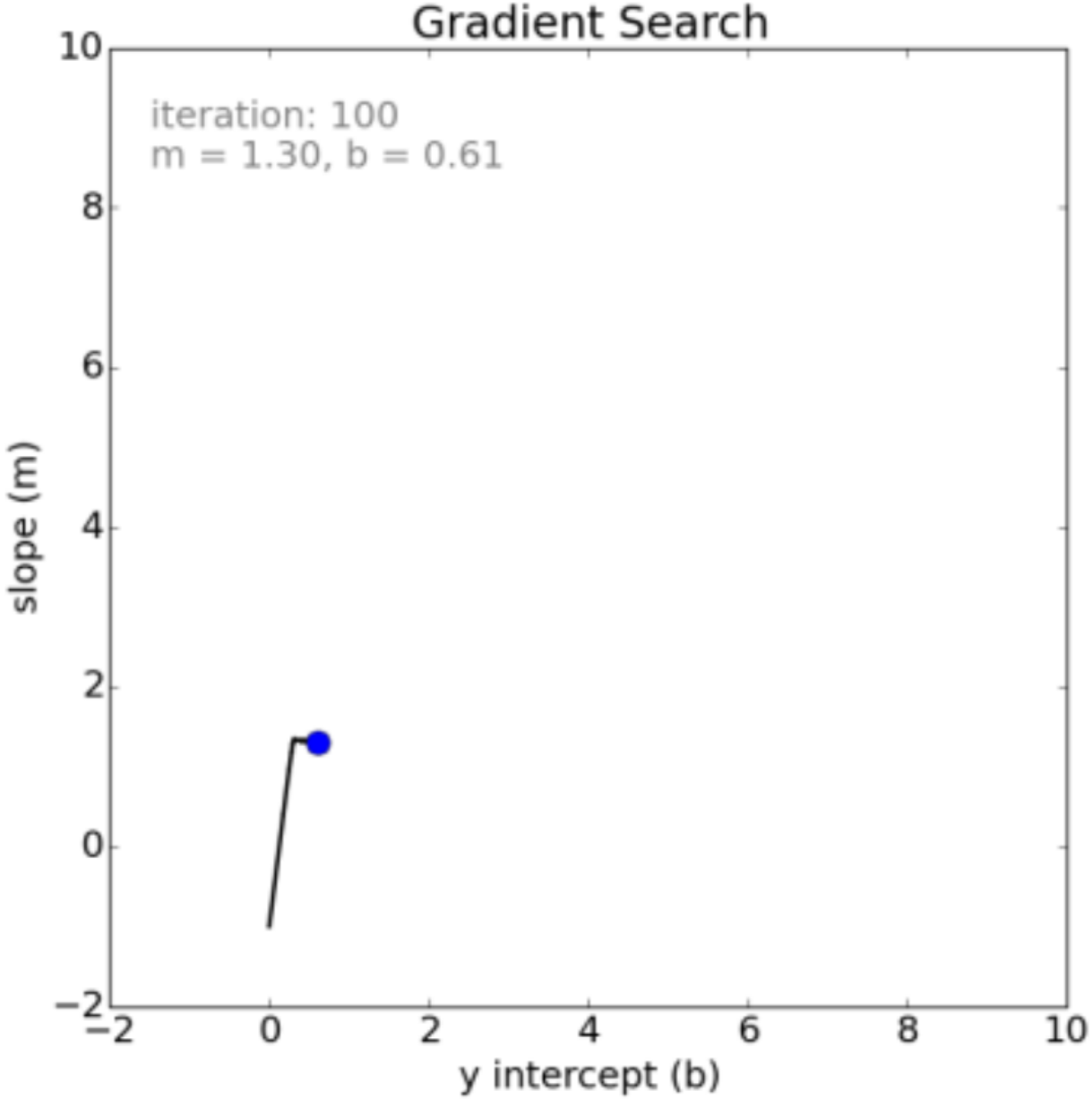

Data and Current Line
y
x
25
20
15
10
5
0
−5
−5
0
5
10
15
20

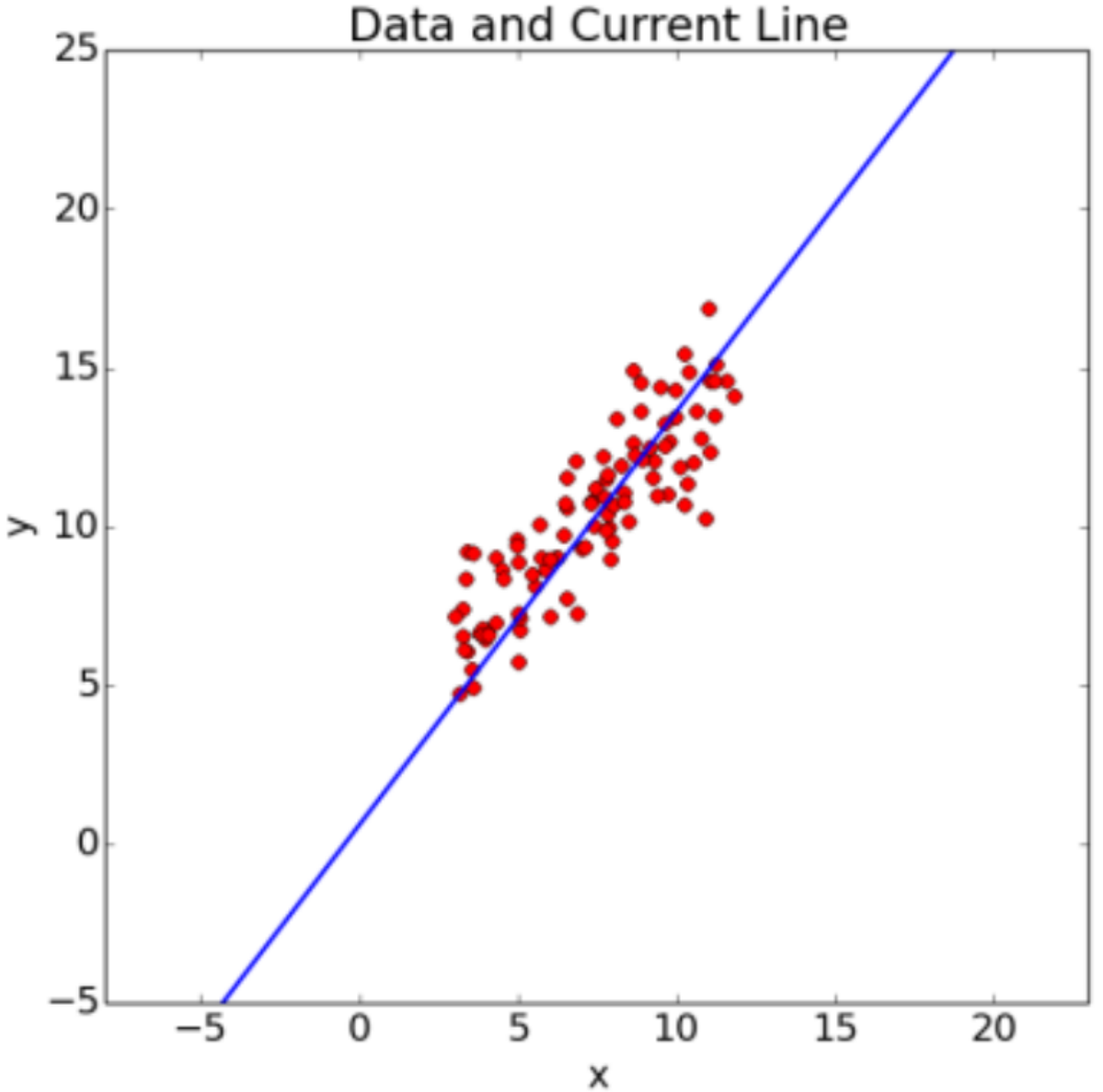
J


Gradient Search
iteration: 2
m = 0.96, b = 0.25
slope (m)
y intercept (b)
10
8
6
4
2
0
−2
−2
0
2
4
6
8
10

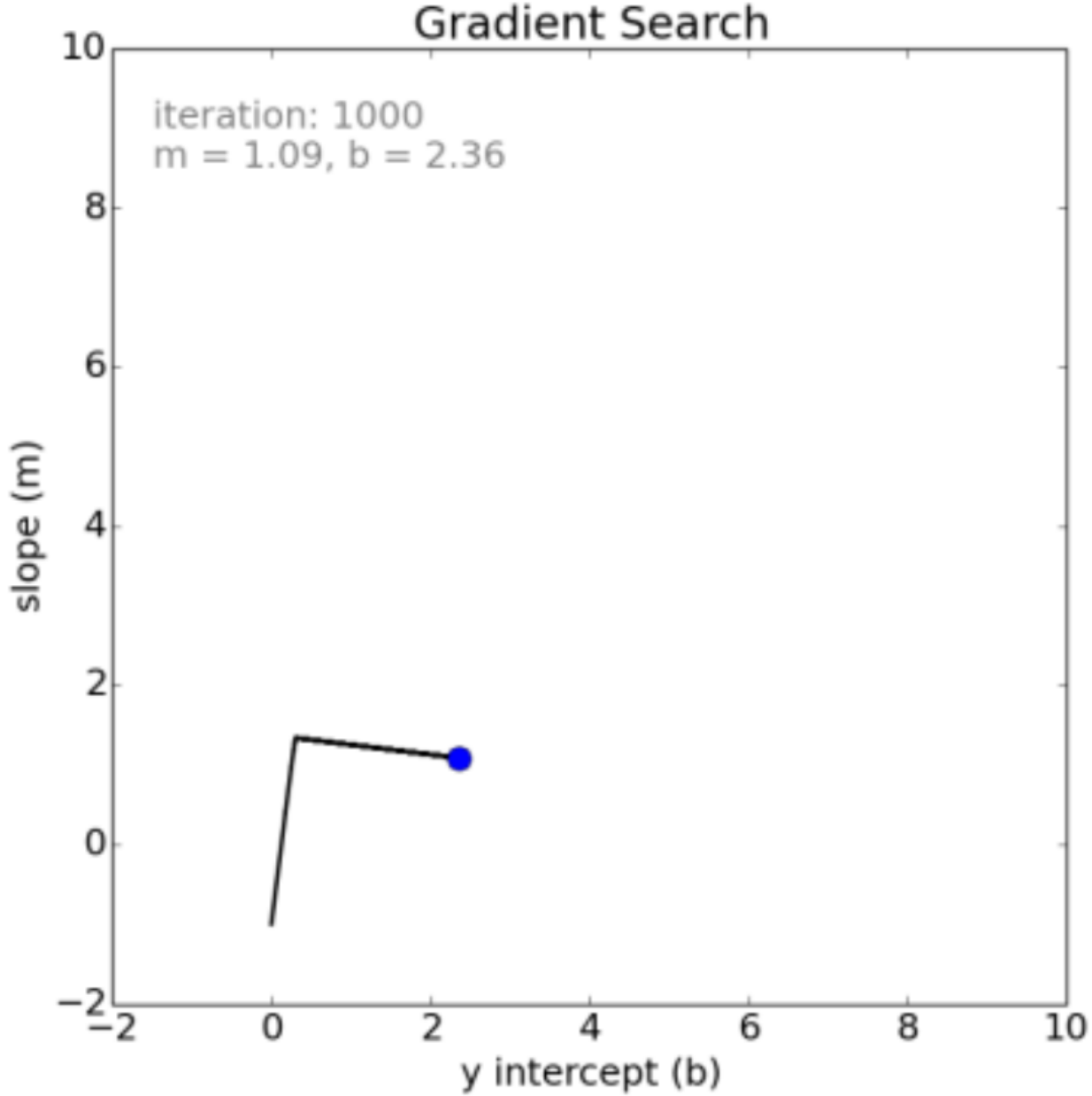

Data and Current Line
y
x
25
20
15
10
5
0
−5
−5
0
5
10
15
20

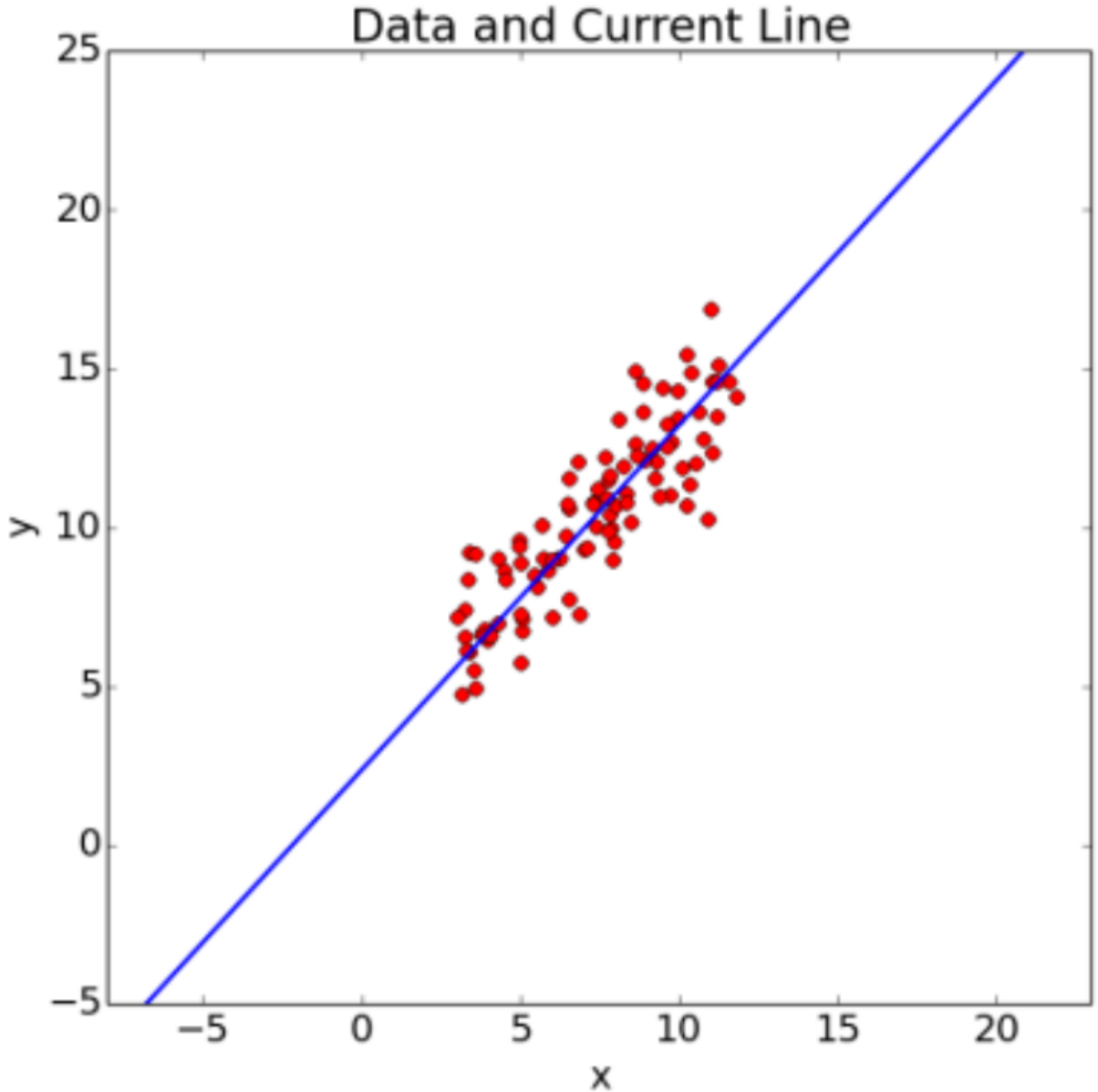
J


Gradient Search
iteration: 100
m = 1.30, b = 0.61
slope (m)
y intercept (b)
10
8
6
4
2
0
−2
−2
0
2
4
6
8
10

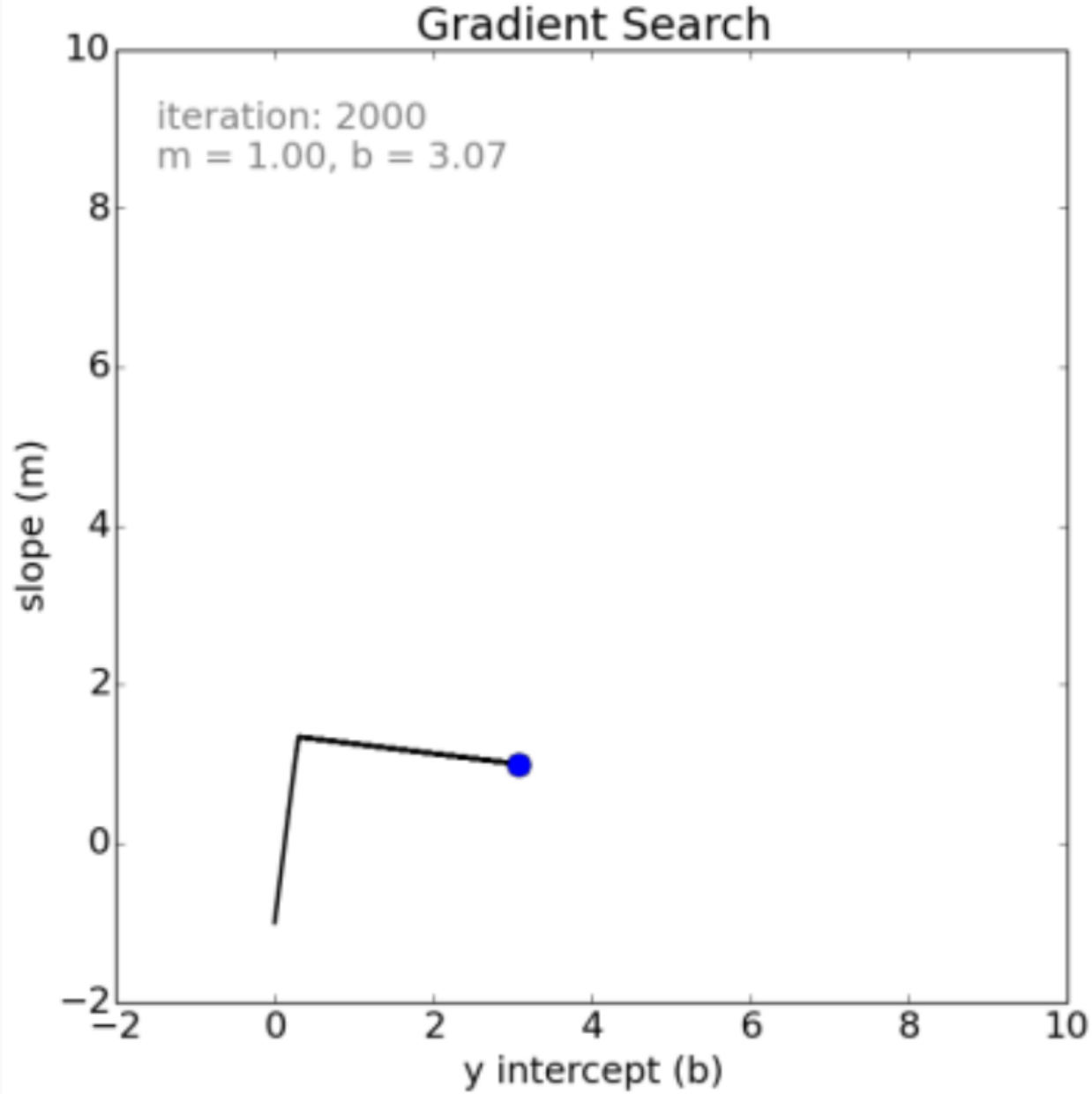

Data and Current Line
y
x
25
20
15
10
5
0
−5
−5
0
5
10
15
20

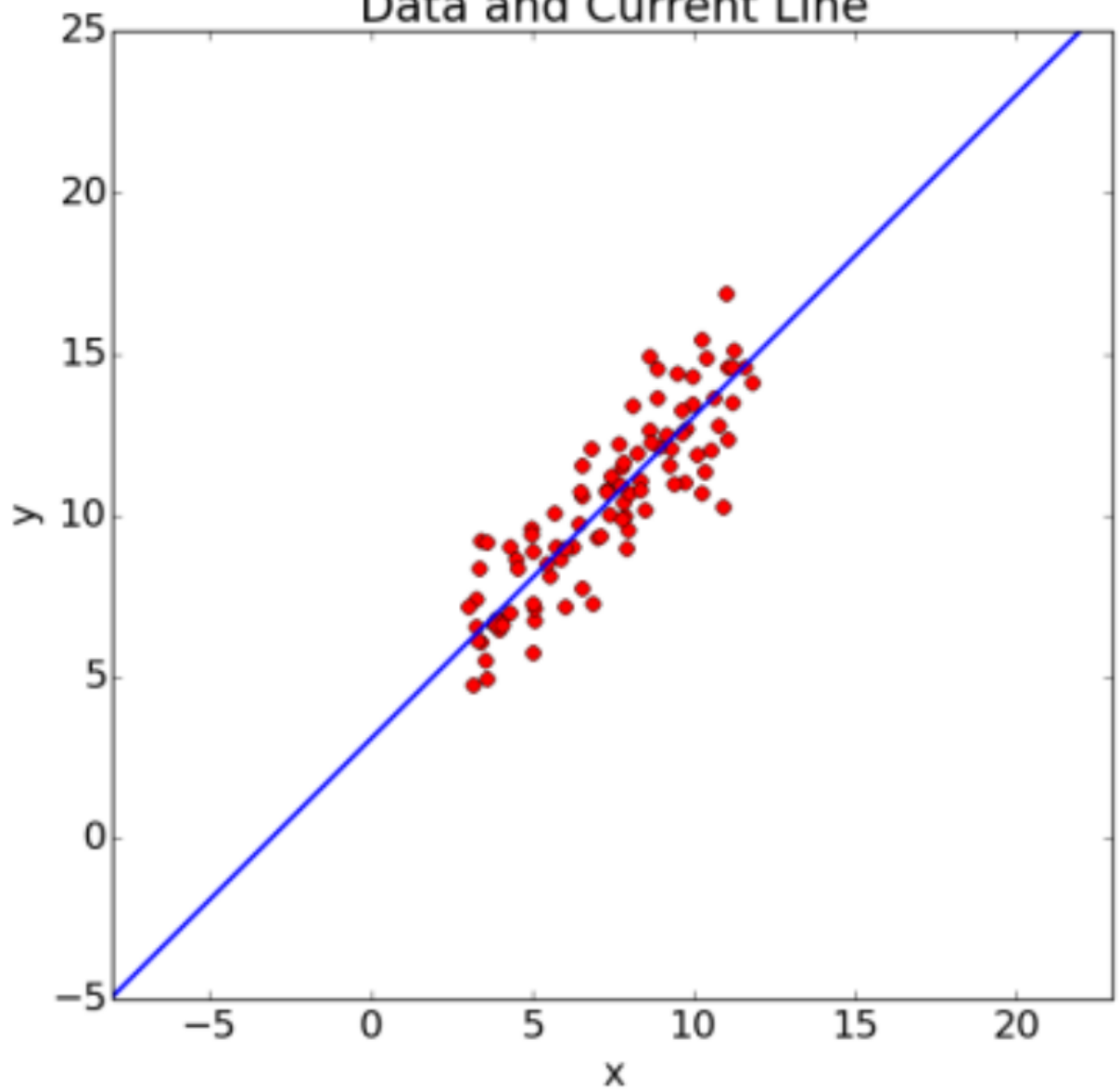
J


Gradient Search
iteration: 1000
m = 1.09, b = 2.36
slope (m)
y intercept (b)
10
8
6
4
2
0
−2
−2
0
2
4
6
8
10


Data and Current Line
y
x
25
20
15
10
5
0
−5
−5
0
5
10
15
20

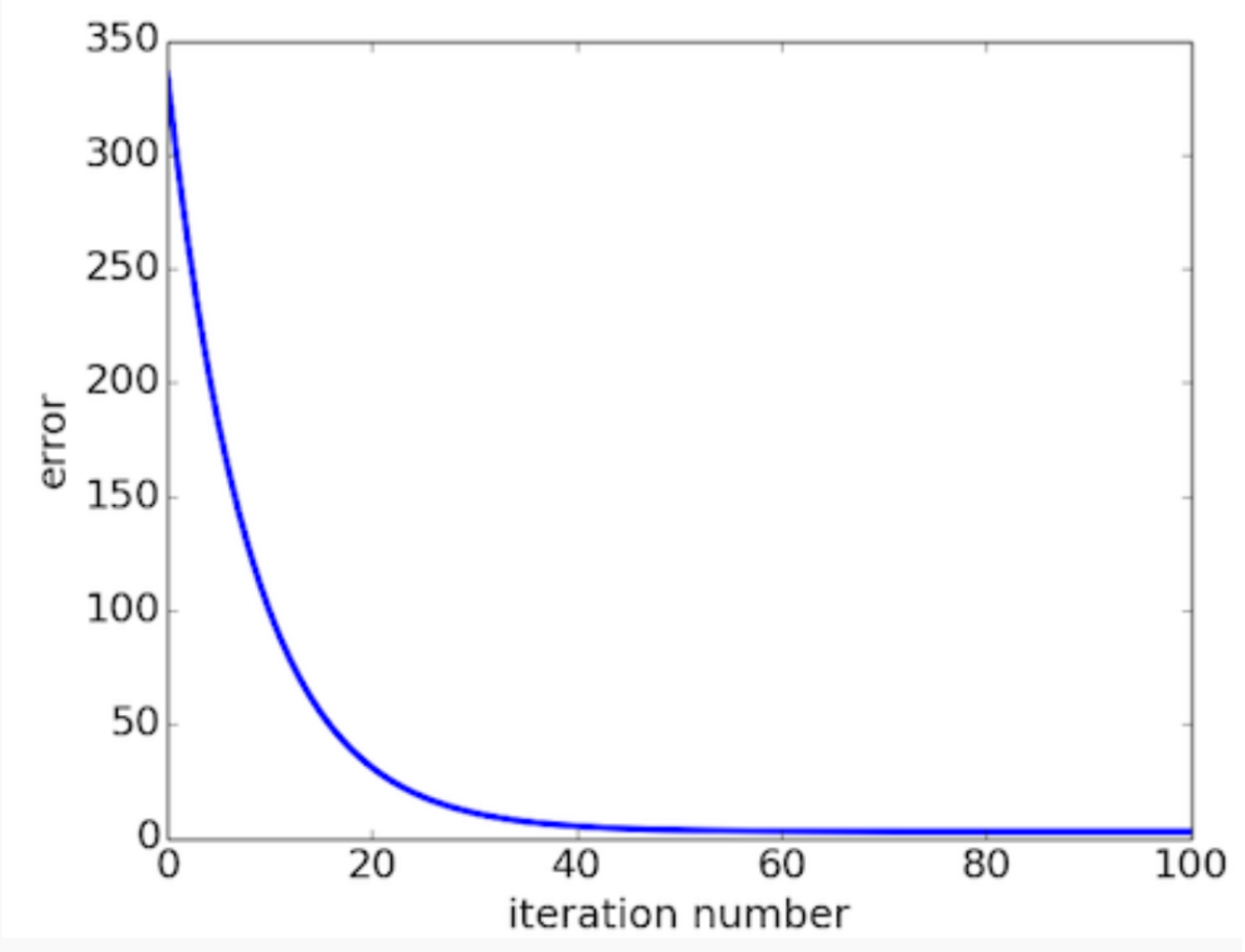
J


Gradient Search
iteration: 2000
m = 1.00, b = 3.07
slope (m)
y intercept (b)
10
8
6
4
2
0
−2
−2
0
2
4
6
8
10


Data and Current Line
y
x
25
20
15
10
5
0
−5
−5
0
5
10
15
20

J

Gradient Descent



J

Review: Gradient Descent

```
1  def stepGradient(b_current, m_current, points, learningRate):
2      b_gradient = 0
3      m_gradient = 0
4      N = float(len(points))
5      for i in range(0, len(points)):
6          b_gradient += -(2/N) * (points[i].y - ((m_current*points[i].x) + b_current))
7          m_gradient += -(2/N) * points[i].x * (points[i].y - ((m_current * points[i].x) + b_current))
8      new_b = b_current - (learningRate * b_gradient)
9      new_m = m_current - (learningRate * m_gradient)
10     return [new_b, new_m]
```

J

Basic idea

- Imagine walking around on an error surface in weight-space!

- Randomly initialize the weights.
- Compute the gradient (i.e., first-order derivative) at the current point.
- Take a step in the direction pointed to by the gradient, i.e., update the weights slightly.
- Repeat until convergence, i.e., when the gradient is zero.

Improving on the basic idea

- We can be clever when we initialize the weights!
- We can consider the curvature (i.e., second-order derivative) and make the step size a function of that!
- We can introduce a momentum term as part of the weight update!
- We can batch things up, i.e., we don't have to use the full training set at every iteration but estimate the gradient from a subset of the data!
- We can have different step sizes (learning rates) per dimension!
- We can have the learning rate(s) adapt during training!
- We can try to avoid ending up in a local minimum!

See also https://arxiv.org/abs/1609.04747 or http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

More Gradient Descent

- General Problem
  - Minimize expected loss

$$\min_h L(h) = \mathbf{E}\left[\text{loss}(h(x), y)\right]$$

given samples

$$(x_i, y_i)\ i = 1, 2...m$$

- This is Stochastic Optimization
  - Assume loss function is convex

Batch Gradient Descent

- Process all examples together in each step

$$w^{(k+1)} \leftarrow w^{(k)} - \eta_t \left(\frac{1}{n} \sum_{i=1}^{n} \frac{\partial L(w, x_i, y_i)}{\partial w} \right)$$

- Entire training set examined at each step
- Very slow when *n* is very large

- Learning rate η_t

J

Stochastic Gradient Descent

- “Optimize” with one example at a time
- Choose examples randomly (or reorder and choose in order)
  - Learning representative of example distribution

$$\text{for } i = 1 \text{ to } n\text{:}$$

$$w^{(k+1)} \leftarrow w^{(k)} - \eta_t \frac{\partial L(w, x_i, y_i)}{\partial w}$$

$$\text{where } L \text{ is the regularized loss function}$$

Stochastic Gradient Descent

$$\text{for } i = 1 \text{ to } n\text{:}$$

$$w^{(k+1)} \leftarrow w^{(k)} - \eta_t \frac{\partial L(w, x_i, y_i)}{\partial w}$$

- Equivalent to online learning (the weight vector *w* changes with every example)
- Convergence guaranteed for convex functions (to local minimum)

Hybrid!

- Stochastic – 1 example per iteration
- Batch – All the examples!
- Sample Average Approximation (SAA):
  - Sample *m* examples at each step and perform SGD on them
- Allows for parallelization, but choice of *m* based on heuristics

SGD - Issues

- Convergence very sensitive to learning rate η_t
  (oscillations near solution due to probabilistic nature of sampling)
  - Might need to decrease with time to ensure the algorithm converges eventually
- Basically – SGD good for machine learning with large data sets!

$$w^{(k+1)} \leftarrow w^{(k)} - \eta_t \left(\frac{1}{n} \sum_{i=1}^{n} \frac{\partial L(w, x_i, y_i)}{\partial w} \right)$$

J

This Lecture

- Loss Functions
- Gradient Descent
- Parameters and hyperparameters
- Regularization

More Jargon: Parameters and Hyperparameters

- So far we had models with parameters
  - The weights in an artificial neural network.
  - The support vectors in a support vector machine.
  - The coefficients in a linear regression or logistic regression.

- But why not train a quadratic function? Why not a cubic function?
  → These are hyperparameters

- Hyperparameters
  - Cannot directly be learned from the standard model training
  - Examples:
    - Model class/complexity
    - Learning rates (later)

Parameters

- They are required by the model when making predictions.
- They are estimated or learned from data.
- They are not set manually
- They are part of the learned model.

Hyperparameters

- Cannot be learned directly from the data in the standard model training process and need to be predefined.
- They are used in the processes to help estimate model parameters.
- Define higher level concepts about the model such as complexity, or capacity to learn.
- Can be decided by setting different values, training different models, and choosing the values that test better

This often leads to confusion, because often even setting the hyperparameters is part of the optimization.

J

This Lecture

- Loss Functions
- Gradient Descent
- Parameters and hyperparameters
- Regularization

Bias and variance

- We want the model to generalize to unseen data!
- It should learn from the training data, but possibly not too well?
- Perhaps the model has so many “degrees of freedom” that it learns noise and irrelevant details?

Bias–variance tradeoff

From Wikipedia, the free encyclopedia

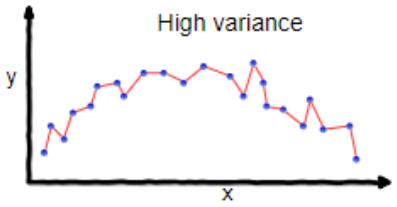
This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable s material may be challenged and removed. *(August 2017) (Learn how and when to remove this template message)*

In statistics and machine learning, the **bias–variance tradeoff** is the property of a set of predictive models whereby models with a lower bias in parameter estimation have a higher variance of the parameter estimates across samples, and vice versa. The **bias–variance dilemma** or **problem** is the conflict in trying to simultaneously minimize these two sources of error that prevent supervised learning algorithms from generalizing beyond their training set:[*citation needed*]

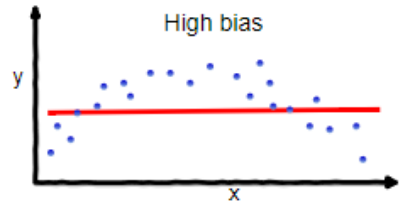
- The *bias* is an error from erroneous assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting).
- The *variance* is an error from sensitivity to small fluctuations in the training set. High variance can cause an algorithm to model the random noise in the training data, rather than the intended outputs (overfitting).

The **bias–variance decomposition** is a way of analyzing a learning algorithm's expected generalization error with respect to a particular problem as a sum of three terms, the bias, variance, and a quantity called the *irreducible error*, resulting from noise in the problem itself.

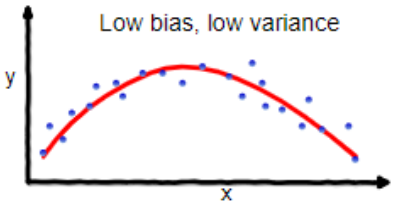
This tradeoff applies to all forms of supervised learning: classification, regression (function fitting),[1][2] and structured output learning. It has also been invoked to explain the effectiveness of heuristics in human learning.[3]



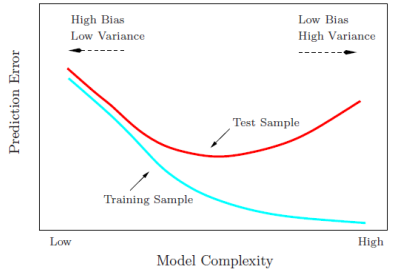
overfitting



underfitting



Good balance



A

Regularization

- Occam's razor
- $J(\boldsymbol{w})$ = (measure of data fit) + λ * (measure of model complexity)
- We no longer consider the model structure entirely fixed!
- Determine λ through a parameter sweep

Regularization, cont.

- Magnitude of weights
- Number of hidden nodes in a network
- Depth of a decision tree
- The number of trees in a forest
- Dropout
- Early stopping

Summary

- Loss Functions
- Gradient Descent
- Parameters and hyperparameters
- Regularization

Questions?