EE2211 Introduction to Machine Learning

Lecture 8

Vincent Tan
vtan@nus.edu.sg

Electrical and Computer Engineering Department
National University of Singapore

Acknowledgement: EE2211 development team
Thomas, Helen, Xinchao, Kar-Ann, Chen Khong, Robby, and Haizhou

Course Contents

- Introduction and Preliminaries (Xinchao)
  - Introduction
  - Data Engineering
  - Introduction to Probability and Statistics
- Fundamental Machine Learning Algorithms I (Vincent)
  - Systems of linear equations
  - Least squares, Linear regression
  - Ridge regression, Polynomial regression
- Fundamental Machine Learning Algorithms II (Vincent)
  - Over-fitting, bias/variance trade-off
  - Optimization, Gradient descent
  - Decision Trees, Random Forest
- Performance and More Algorithms (Xinchao)
  - Performance Issues
  - K-means Clustering
  - Neural Networks

Fundamental ML Algorithms: Optimization, Gradient Descent

Module III Contents

- Overfitting, underfitting and model complexity
- Regularization
- Bias-variance trade-off
- Loss function
- Optimization
- Gradient descent
- Decision trees
- Random forest

Review

- Supervised learning: given feature(s) x, we want to predict target y

Review

- Supervised learning: given feature(s) x, we want to predict target y
- Most supervised learning algorithms can be formulated as the following optimization problem

$$\operatorname*{argmin}_{\mathbf{w}} \textbf{Data-Loss}(\mathbf{w}) + \lambda \textbf{Regularization}(\mathbf{w})$$

- **Data-Loss(w)** quantifies fitting error to training set given parameters **w:** smaller error => better fit to training data
- **Regularization(w)** penalizes more complex models

Review

- Supervised learning: given feature(s) x, we want to predict target y
- Most supervised learning algorithms can be formulated as the following optimization problem

$$\operatorname*{argmin}_{\mathbf{w}} \textbf{Data-Loss}(\mathbf{w}) + \lambda \textbf{Regularization}(\mathbf{w})$$

- **Data-Loss(w)** quantifies fitting error to training set given parameters **w:** smaller error => better fit to training data
- **Regularization(w)** penalizes more complex models
- For example, in the case of polynomial regression (previous lectures):

$$\operatorname*{argmin}_{\mathbf{w}} \underbrace{(\mathbf{P}\mathbf{w} - \mathbf{y})^T(\mathbf{P}\mathbf{w} - \mathbf{y})}_{\textbf{Data-Loss(w)}} + \lambda \underbrace{\mathbf{w}^T\mathbf{w}}_{\textbf{Reg(w)}}$$

Review

- For polynomial regression (previous lectures)

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}}\, (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

Review

- For polynomial regression (previous lectures)

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}}\, (\mathbf{P}\mathbf{w} - \mathbf{y})^T(\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T\mathbf{w}$$

$$= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} (\mathbf{p}_i^T\mathbf{w} - y_i)^2 + \lambda \mathbf{w}^T\mathbf{w}$$

$\mathbf{p}_i^T\mathbf{w}$ is prediction of i-th training sample

y_i is target of i-th training sample

Review

- For polynomial regression (previous lectures)

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}}\, (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

$$= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} (\mathbf{p}_i^T \mathbf{w} - y_i)^2 + \lambda \mathbf{w}^T \mathbf{w}$$

- Linear regression with 2 features, $\mathbf{p}_i = \begin{bmatrix} 1 \\ x_{1,i} \\ x_{2,i} \end{bmatrix}$
  - Bias/Offset
  - Feature 1 of i-th sample
  - Feature 2 of i-th sample

Review

- For polynomial regression (previous lectures)

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}}\, (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

$$= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} (\mathbf{p}_i^T \mathbf{w} - y_i)^2 + \lambda \mathbf{w}^T \mathbf{w}$$

- Linear regression with 2 features, $\mathbf{p}_i = \begin{bmatrix} 1 \\ x_{1,i} \\ x_{2,i} \end{bmatrix}$

  — Bias/Offset
  — Feature 1 of i-th sample
  — Feature 2 of i-th sample

- Quadratic regression with 1 feature, $\mathbf{p}_i = \begin{bmatrix} 1 \\ x_i \\ x_i^2 \end{bmatrix}$

  — Bias/Offset
  x is feature of i-th sample

Loss Function & Learning Model

- For polynomial regression (previous lectures)

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}}\, (\mathbf{P}\mathbf{w} - \mathbf{y})^T(\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T\mathbf{w}$$

$$= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} (\mathbf{p}_i^T\mathbf{w} - y_i)^2 + \lambda \mathbf{w}^T\mathbf{w}$$

Loss Function & Learning Model

- For polynomial regression (previous lectures)

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}}\, (\mathbf{P}\mathbf{w} - \mathbf{y})^T(\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T\mathbf{w}$$
$$= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} (\mathbf{p}_i^T\mathbf{w} - y_i)^2 + \lambda \mathbf{w}^T\mathbf{w}$$

- Let $f(\mathbf{x}_i, \mathbf{w})$ be the prediction of target y_i from features $\mathbf{x}_i$ for i-th training sample. For example, suppose $f(\mathbf{x}_i, \mathbf{w}) = \mathbf{p}_i^T\mathbf{w}$, then above becomes

Loss Function & Learning Model

- For polynomial regression (previous lectures)

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}}\, (\mathbf{P}\mathbf{w} - \mathbf{y})^T(\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T\mathbf{w}$$

$$= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} (\mathbf{p}_i^T\mathbf{w} - y_i)^2 + \lambda \mathbf{w}^T\mathbf{w}$$

- Let $f(\mathbf{x}_i, \mathbf{w})$ be the prediction of target y_i from features $\mathbf{x}_i$ for i-th training sample. For example, suppose $f(\mathbf{x}_i, \mathbf{w}) = \mathbf{p}_i^T\mathbf{w}$, then above becomes

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2 + \lambda \mathbf{w}^T\mathbf{w}$$

Loss Function & Learning Model

- For polynomial regression (previous lectures)

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}}\, (\mathbf{P}\mathbf{w} - \mathbf{y})^T(\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T\mathbf{w}$$

$$= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} (\mathbf{p}_i^T\mathbf{w} - y_i)^2 + \lambda \mathbf{w}^T\mathbf{w}$$

- Let $f(\mathbf{x}_i, \mathbf{w})$ be the prediction of target y_i from features $\mathbf{x}_i$ for i-th training sample. For example, suppose $f(\mathbf{x}_i, \mathbf{w}) = \mathbf{p}_i^T\mathbf{w}$, then above becomes

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2 + \lambda \mathbf{w}^T\mathbf{w}$$

- Let $L(f(\mathbf{x}_i, \mathbf{w}), y_i)$ be the penalty for predicting $f(\mathbf{x}_i, \mathbf{w})$ when true value is y_i. For example, suppose $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$, then above becomes

Loss Function & Learning Model

- For polynomial regression (previous lectures)

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}}\, (\mathbf{P}\mathbf{w} - \mathbf{y})^T(\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T\mathbf{w}$$

$$= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} (\mathbf{p}_i^T\mathbf{w} - y_i)^2 + \lambda \mathbf{w}^T\mathbf{w}$$

- Let $f(\mathbf{x}_i, \mathbf{w})$ be the prediction of target y_i from features $\mathbf{x}_i$ for i-th training sample. For example, suppose $f(\mathbf{x}_i, \mathbf{w}) = \mathbf{p}_i^T\mathbf{w}$, then above becomes

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2 + \lambda \mathbf{w}^T\mathbf{w}$$

- Let $L(f(\mathbf{x}_i, \mathbf{w}), y_i)$ be the penalty for predicting $f(\mathbf{x}_i, \mathbf{w})$ when true value is y_i. For example, suppose $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$, then above becomes

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda \mathbf{w}^T\mathbf{w}$$

Loss Function & Learning Model

- From previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda \mathbf{w}^T \mathbf{w}$$

Loss Function & Learning Model

- From previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda \mathbf{w}^T \mathbf{w}$$

- To make it even more general, we can write

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda R(\mathbf{w})$$

Loss Function & Learning Model

- From previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda \mathbf{w}^T \mathbf{w}$$

- To make it even more general, we can write

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda R(\mathbf{w})$$

Cost Function Loss Function Learning Model Regularization

Building Blocks of ML algorithms

- From previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda \mathbf{w}^T \mathbf{w}$$

- To make it even more general, we can write

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda R(\mathbf{w})$$

- Learning model f reflects our belief about the relationship between the features $\mathbf{x}_i$ & target y_i

Building Blocks of ML algorithms

- From previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda \mathbf{w}^T \mathbf{w}$$

- To make it even more general, we can write

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda R(\mathbf{w})$$

- Learning model f reflects our belief about the relationship between the features $\mathbf{x}_i$ & target y_i
- Loss function L is the penalty for predicting $f(\mathbf{x}_i, \mathbf{w})$ when the true value is y_i

Building Blocks of ML algorithms

- From previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda \mathbf{w}^T \mathbf{w}$$

- To make it even more general, we can write

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda R(\mathbf{w})$$

- Learning model f reflects our belief about the relationship between the features $\mathbf{x}_i$ & target y_i
- Loss function L is the penalty for predicting $f(\mathbf{x}_i, \mathbf{w})$ when the true value is y_i
- Regularization R encourages less complex models

Building Blocks of ML algorithms

- From previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda \mathbf{w}^T \mathbf{w}$$

- To make it even more general, we can write

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda R(\mathbf{w})$$

- Learning model f reflects our belief about the relationship between the features $\mathbf{x}_i$ & target y_i
- Loss function L is the penalty for predicting $f(\mathbf{x}_i, \mathbf{w})$ when the true value is y_i
- Regularization R encourages less complex models
- Cost function C is the final optimization criterion we want to minimize

Building Blocks of ML algorithms

- From previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda \mathbf{w}^T \mathbf{w}$$

- To make it even more general, we can write

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda R(\mathbf{w})$$

- Learning model f reflects our belief about the relationship between the features $\mathbf{x}_i$ & target y_i
- Loss function L is the penalty for predicting $f(\mathbf{x}_i, \mathbf{w})$ when the true value is y_i
- Regularization R encourages less complex models
- Cost function C is the final optimization criterion we want to minimize
- Optimization routine to find solution to cost function

Motivation for Gradient Descent

- Different learning function f, loss function L & regularization R give rise to different learning algorithms

Motivation for Gradient Descent

- Different learning function f, loss function L & regularization R give rise to different learning algorithms
- In polynomial regression (previous lectures), optimal $\mathbf{w}$ can be written with the following "closed-form" formula (primal solution):

$$\hat{\mathbf{w}} = (\mathbf{P}_{train}^T \mathbf{P}_{train} + \lambda \mathbf{I})^{-1} \mathbf{P}_{train}^T \mathbf{y}_{train}$$

Motivation for Gradient Descent

- Different learning function f, loss function L & regularization R give rise to different learning algorithms
- In polynomial regression (previous lectures), optimal $\mathbf{w}$ can be written with the following "closed-form" formula (primal solution):

$$\hat{\mathbf{w}} = (\mathbf{P}_{train}^T \mathbf{P}_{train} + \lambda \mathbf{I})^{-1} \mathbf{P}_{train}^T \mathbf{y}_{train}$$

- For other learning function f, loss function L & regularization R, optimizing $C(\mathbf{w})$ might not be so easy

Motivation for Gradient Descent

- Different learning function f, loss function L & regularization R give rise to different learning algorithms
- In polynomial regression (previous lectures), optimal $\mathbf{w}$ can be written with the following "closed-form" formula (primal solution):

$$\hat{\mathbf{w}} = (\mathbf{P}_{train}^T \mathbf{P}_{train} + \lambda \mathbf{I})^{-1} \mathbf{P}_{train}^T \mathbf{y}_{train}$$

- For other learning function f, loss function L & regularization R, optimizing $C(\mathbf{w})$ might not be so easy
- Usually have to estimate $\mathbf{w}$ iteratively with some algorithm

Motivation for Gradient Descent

- Different learning function f, loss function L & regularization R give rise to different learning algorithms
- In polynomial regression (previous lectures), optimal $\mathbf{w}$ can be written with the following "closed-form" formula (primal solution):

$$\hat{\mathbf{w}} = (\mathbf{P}_{train}^T \mathbf{P}_{train} + \lambda \mathbf{I})^{-1} \mathbf{P}_{train}^T \mathbf{y}_{train}$$

- For other learning function f, loss function L & regularization R, optimizing $C(\mathbf{w})$ might not be so easy
- Usually have to estimate $\mathbf{w}$ iteratively with some algorithm
- Optimization workhorse for modern machine learning is gradient descent

Questions?

Gradient Descent Algorithm

- Suppose we want to minimize $C(\mathbf{w})$ with respect to $\mathbf{w} = [w_1, \cdots, w_d]^T$

Gradient Descent Algorithm

- Suppose we want to minimize $C(\mathbf{w})$ with respect to $\mathbf{w} = [w_1, \cdots, w_d]^T$
- Gradient $\nabla_{\mathbf{w}} C(\mathbf{w}) = \begin{pmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial w_2} \\ \vdots \\ \frac{\partial C}{\partial w_d} \end{pmatrix}$

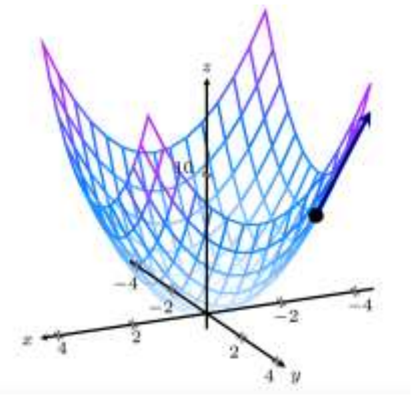
Gradient Descent Algorithm

- Suppose we want to minimize $C(\mathbf{w})$ with respect to $\mathbf{w} = [w_1, \cdots, w_d]^T$
- Gradient $\nabla_{\mathbf{w}} C(\mathbf{w}) = \begin{pmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial w_2} \\ \vdots \\ \frac{\partial C}{\partial w_d} \end{pmatrix}$
  - $\nabla_{\mathbf{w}} C(\mathbf{w})$ is vector & function of $\mathbf{w}$

Gradient Descent Algorithm

- Suppose we want to minimize $C(\mathbf{w})$ with respect to $\mathbf{w} = [w_1, \cdots, w_d]^T$

- Gradient $\nabla_{\mathbf{w}} C(\mathbf{w}) = \begin{pmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial w_2} \\ \vdots \\ \frac{\partial C}{\partial w_d} \end{pmatrix}$

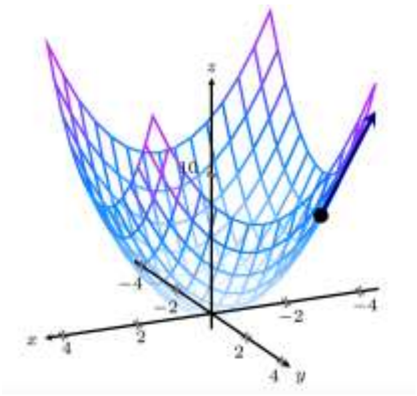


  – $\nabla_{\mathbf{w}} C(\mathbf{w})$ is vector & function of $\mathbf{w}$
  – $\nabla_{\mathbf{w}} C(\mathbf{w})$ is direction at $\mathbf{w}$ where C is increasing most rapidly, so $-\nabla_{\mathbf{w}} C(\mathbf{w})$ is direction at $\mathbf{w}$ where C is decreasing most rapidly

Gradient Descent Algorithm

- Suppose we want to minimize $C(\mathbf{w})$ with respect to $\mathbf{w} = [w_1, \cdots, w_d]^T$

- Gradient $\nabla_{\mathbf{w}} C(\mathbf{w}) = \begin{pmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial w_2} \\ \vdots \\ \frac{\partial C}{\partial w_d} \end{pmatrix}$



  - $\nabla_{\mathbf{w}} C(\mathbf{w})$ is vector & function of $\mathbf{w}$
  - $\nabla_{\mathbf{w}} C(\mathbf{w})$ is direction at $\mathbf{w}$ where C is increasing most rapidly, so $-\nabla_{\mathbf{w}} C(\mathbf{w})$ is direction at $\mathbf{w}$ where C is decreasing most rapidly

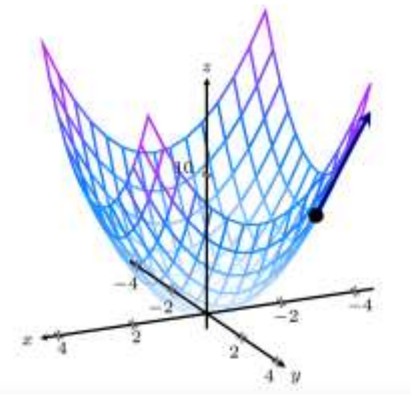
- Gradient Descent:

  Initialize $\mathbf{w}_0$ and learning rate η;

Gradient Descent Algorithm

- Suppose we want to minimize $C(\mathbf{w})$ with respect to $\mathbf{w} = [w_1, \cdots, w_d]^T$

- Gradient $\nabla_{\mathbf{w}} C(\mathbf{w}) = \begin{pmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial w_2} \\ \vdots \\ \frac{\partial C}{\partial w_d} \end{pmatrix}$



  - $\nabla_{\mathbf{w}} C(\mathbf{w})$ is vector & function of $\mathbf{w}$
  - $\nabla_{\mathbf{w}} C(\mathbf{w})$ is direction at $\mathbf{w}$ where C is increasing most rapidly, so $-\nabla_{\mathbf{w}} C(\mathbf{w})$ is direction at $\mathbf{w}$ where C is decreasing most rapidly

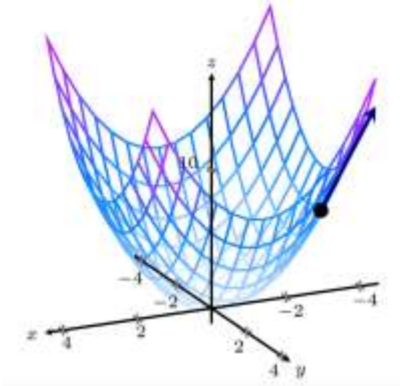
- Gradient Descent:

  Initialize $\mathbf{w}_0$ and learning rate η;
  while *true* **do**

  end

Gradient Descent Algorithm

- Suppose we want to minimize $C(\mathbf{w})$ with respect to $\mathbf{w} = [w_1, \cdots, w_d]^T$



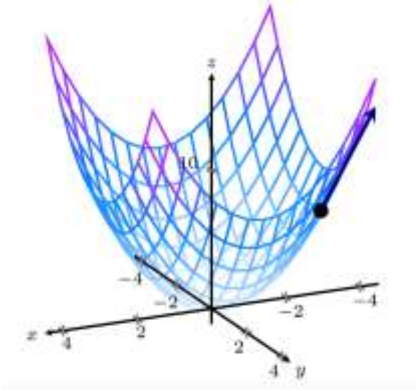
- Gradient $\nabla_{\mathbf{w}} C(\mathbf{w}) = \begin{pmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial w_2} \\ \vdots \\ \frac{\partial C}{\partial w_d} \end{pmatrix}$
  - $\nabla_{\mathbf{w}} C(\mathbf{w})$ is vector & function of $\mathbf{w}$
  - $\nabla_{\mathbf{w}} C(\mathbf{w})$ is direction at $\mathbf{w}$ where C is increasing most rapidly, so $-\nabla_{\mathbf{w}} C(\mathbf{w})$ is direction at $\mathbf{w}$ where C is decreasing most rapidly

- Gradient Descent:

  Initialize $\mathbf{w}_0$ and learning rate η;
  while *true* **do**
      Compute $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \eta \nabla_{\mathbf{w}} C(\mathbf{w}_k)$
  end

Gradient Descent Algorithm

- Suppose we want to minimize $C(\mathbf{w})$ with respect to $\mathbf{w} = [w_1, \cdots, w_d]^T$
- Gradient $\nabla_{\mathbf{w}} C(\mathbf{w}) = \begin{pmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial w_2} \\ \vdots \\ \frac{\partial C}{\partial w_d} \end{pmatrix}$



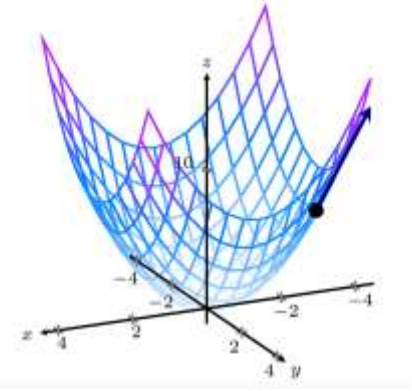
  - $\nabla_{\mathbf{w}} C(\mathbf{w})$ is vector & function of $\mathbf{w}$
  - $\nabla_{\mathbf{w}} C(\mathbf{w})$ is direction at $\mathbf{w}$ where C is increasing most rapidly, so $-\nabla_{\mathbf{w}} C(\mathbf{w})$ is direction at $\mathbf{w}$ where C is decreasing most rapidly
- Gradient Descent:

Initialize $\mathbf{w}_0$ and learning rate η;
while *true* **do**
  Compute $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \eta \nabla_{\mathbf{w}} C(\mathbf{w}_k)$
end

According to multi-variable calculus, if eta is not too big, then $C(\mathbf{w}_{k+1}) < C(\mathbf{w}_k)$ => we get better $\mathbf{w}$ after each iteration

Gradient Descent Algorithm

- Suppose we want to minimize $C(\mathbf{w})$ with respect to $\mathbf{w} = [w_1, \cdots, w_d]^T$
- Gradient $\nabla_{\mathbf{w}} C(\mathbf{w}) = \begin{pmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial w_2} \\ \vdots \\ \frac{\partial C}{\partial w_d} \end{pmatrix}$



  - $\nabla_{\mathbf{w}} C(\mathbf{w})$ is vector & function of $\mathbf{w}$
  - $\nabla_{\mathbf{w}} C(\mathbf{w})$ is direction at $\mathbf{w}$ where C is increasing most rapidly, so $-\nabla_{\mathbf{w}} C(\mathbf{w})$ is direction at $\mathbf{w}$ where C is decreasing most rapidly
- Gradient Descent:

Initialize $\mathbf{w}_0$ and learning rate η;
while *true* **do**
  Compute $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \eta \nabla_{\mathbf{w}} C(\mathbf{w}_k)$
  if *converge* **then**
    return $\mathbf{w}_{k+1}$
  end
end

According to multi-variable calculus, if eta is not too big, then $C(\mathbf{w}_{k+1}) < C(\mathbf{w}_k)$ => we get better $\mathbf{w}$ after each iteration

Gradient Descent Algorithm

- Gradient Descent:

  Initialize $\mathbf{w}_0$ and learning rate η;
  while *true* **do**
  Compute $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \eta \nabla_{\mathbf{w}} C(\mathbf{w}_k)$
  if *converge* **then**
  return $\mathbf{w}_{k+1}$
  end
  end

Gradient Descent Algorithm

- Gradient Descent:

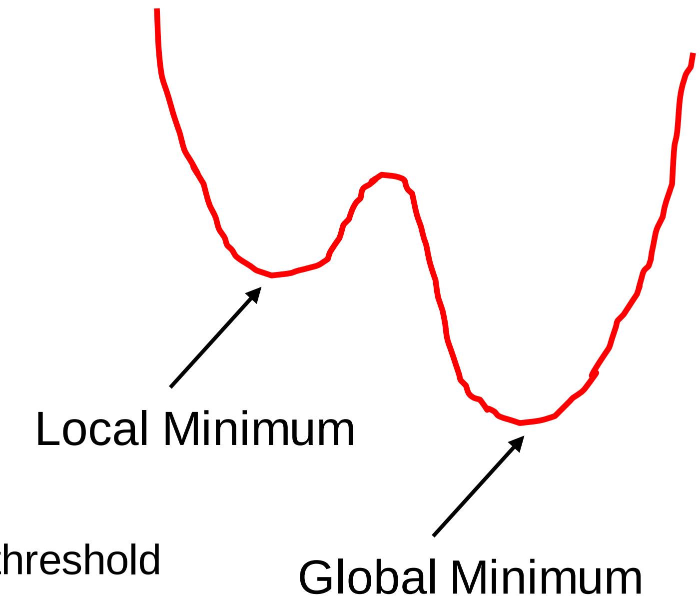
  Initialize $\mathbf{w}_0$ and learning rate η;
  while *true* **do**
  Compute $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \eta \nabla_{\mathbf{w}} C(\mathbf{w}_k)$
  if *converge* **then**
  return $\mathbf{w}_{k+1}$
  end
  end

- Possible convergence criteria
  - Set maximum iteration k
  - Check percentage or absolute change in C below a threshold
  - Check percentage or absolute change in $\mathbf{w}$ below a threshold

Gradient Descent Algorithm

- Gradient Descent:

  Initialize $\mathbf{w}_0$ and learning rate η;
  while *true* **do**
  Compute $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \eta \nabla_{\mathbf{w}} C(\mathbf{w}_k)$
  if *converge* **then**
  return $\mathbf{w}_{k+1}$
  end
  end

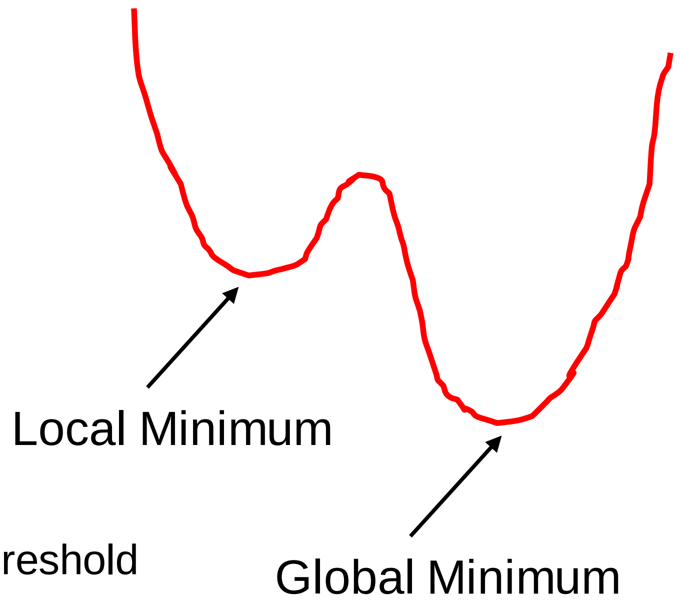


- Possible convergence criteria
  - Set maximum iteration k
  - Check percentage or absolute change in C below a threshold
  - Check percentage or absolute change in $\mathbf{w}$ below a threshold
- Gradient descent can only find local minimum
  - Because gradient = 0 at local minimum, so $\mathbf{w}$ won't change after that

Gradient Descent Algorithm

- Gradient Descent:

  Initialize $\mathbf{w}_0$ and learning rate η;
  while *true* **do**
      Compute $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \eta \nabla_{\mathbf{w}} C(\mathbf{w}_k)$
      if *converge* **then**
          return $\mathbf{w}_{k+1}$
      end
  end



- Possible convergence criteria
  - Set maximum iteration k
  - Check percentage or absolute change in C below a threshold
  - Check percentage or absolute change in $\mathbf{w}$ below a threshold
- Gradient descent can only find local minimum
  - Because gradient = 0 at local minimum, so $\mathbf{w}$ won't change after that
- Many variations of gradient descent, e.g., change how gradient is computed or learning rate η decreases with increasing k

Questions?

Find x to minimize $g(x) = x^2$

Find x to minimize $g(x) = x^2$

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent

Find x to minimize $g(x) = x^2$

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  – Gradient $\nabla_x g(x) = 2x$

Find x to minimize $g(x) = x^2$

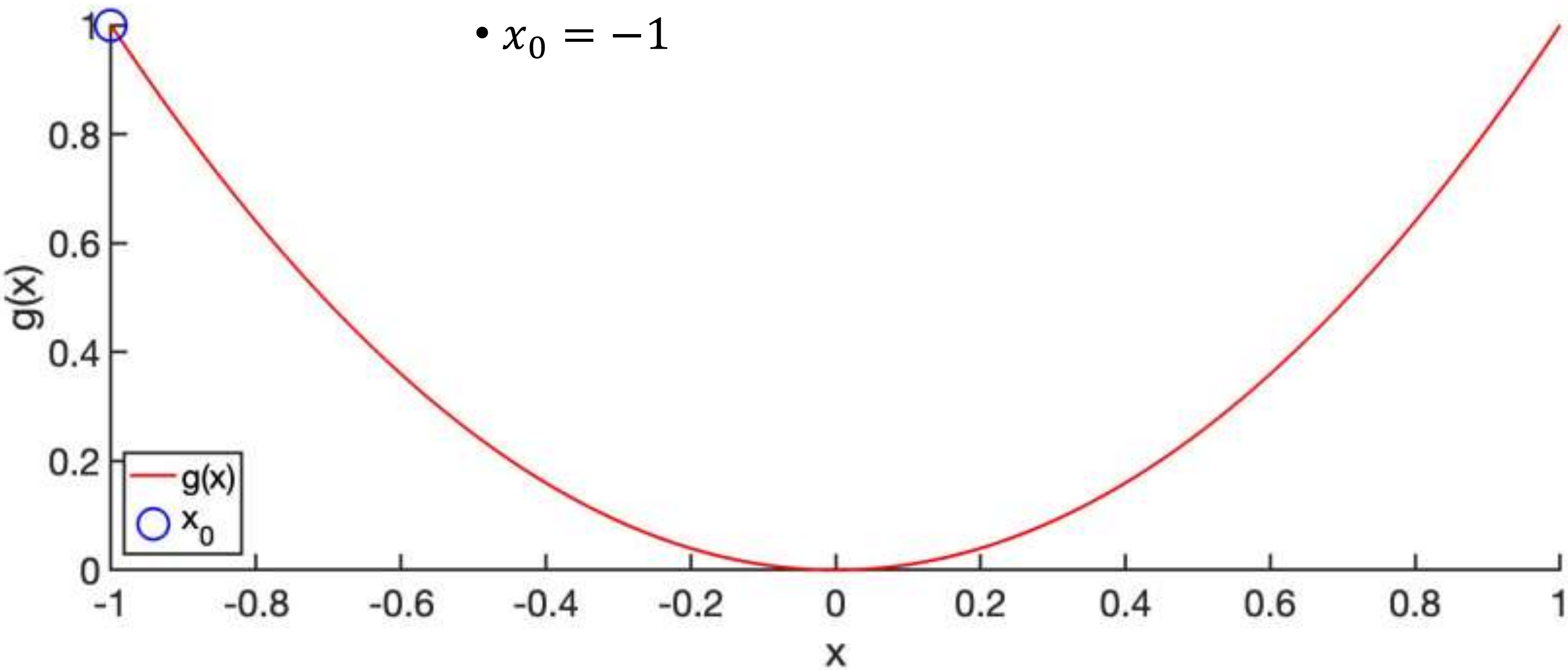
- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.4$

Find x to minimize $g(x) = x^2$

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.4$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$

Find x to minimize $g(x) = x^2$

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.4$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$

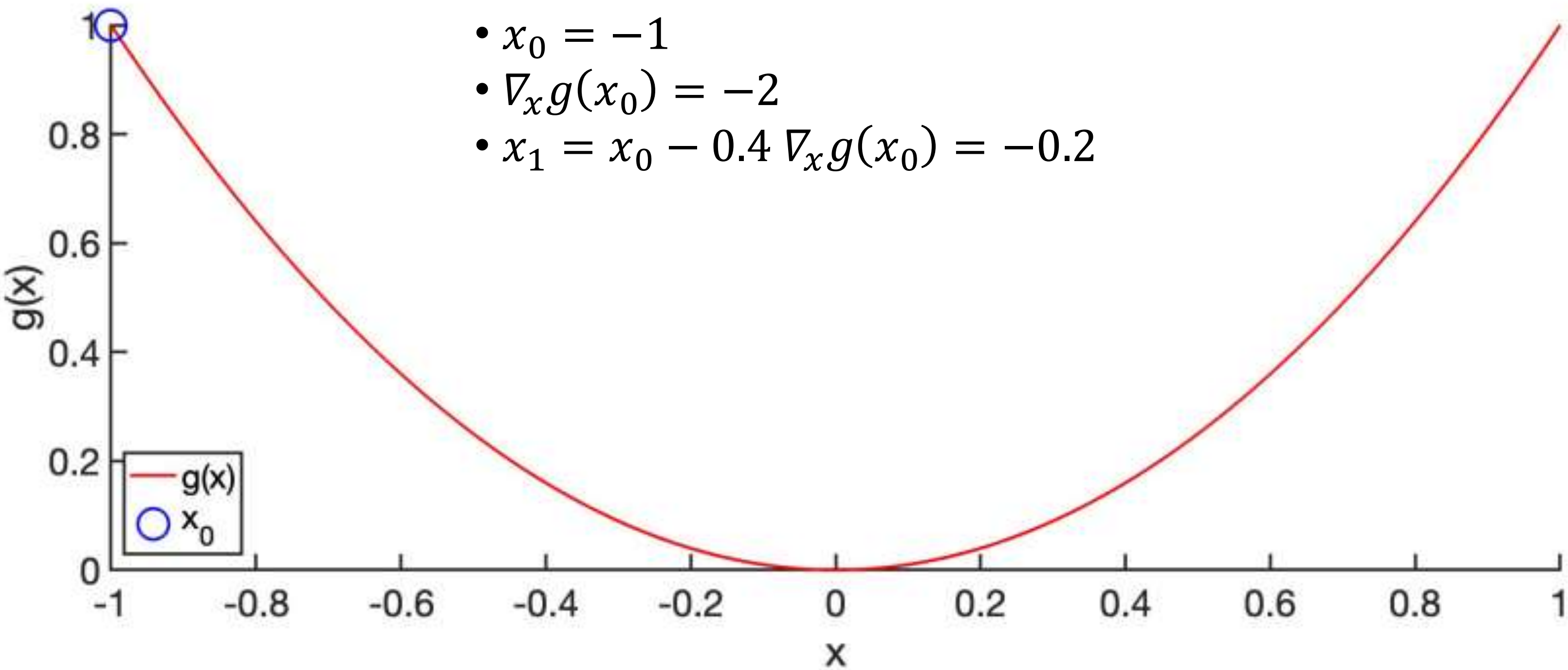


- $x_0 = -1$

Find x to minimize $g(x) = x^2$

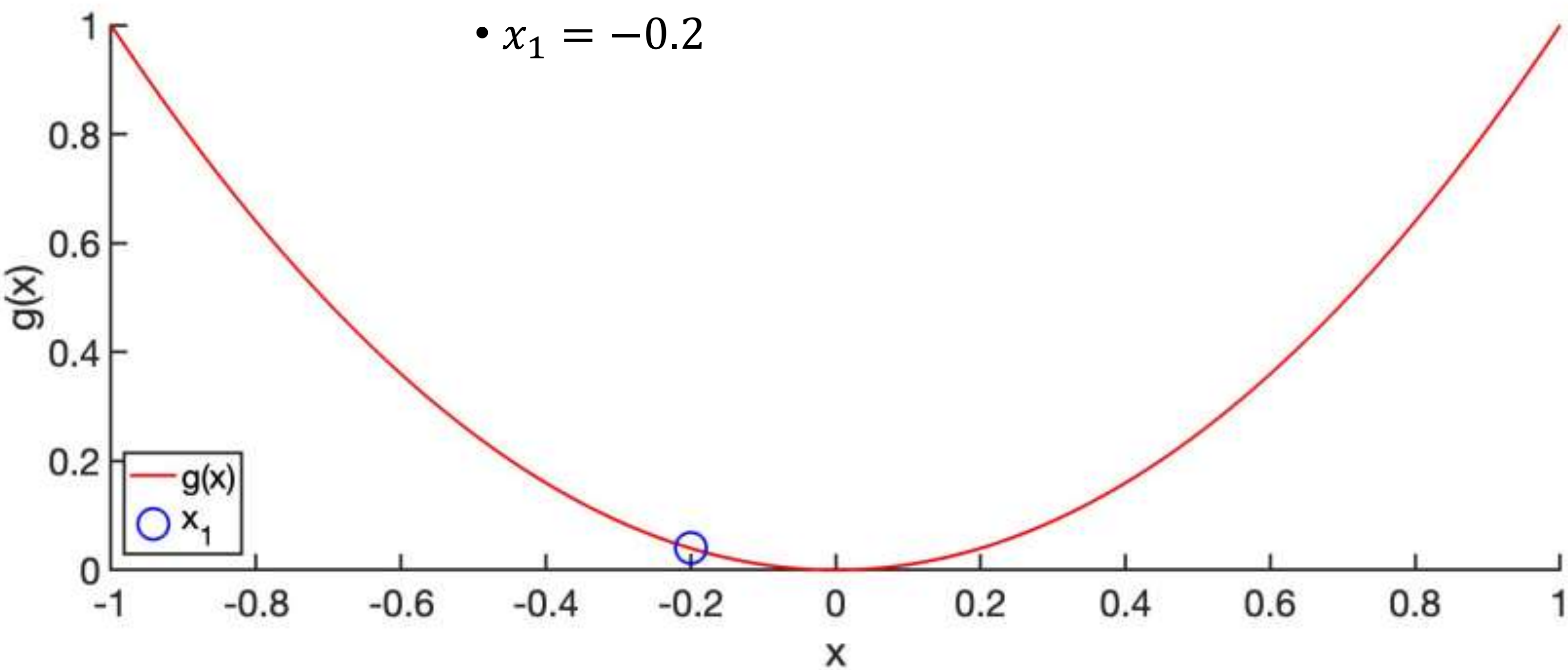
- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.4$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$

- $x_0 = -1$
- $\nabla_x g(x_0) = -2$
- $x_1 = x_0 - 0.4\, \nabla_x g(x_0) = -0.2$



Find x to minimize $g(x) = x^2$

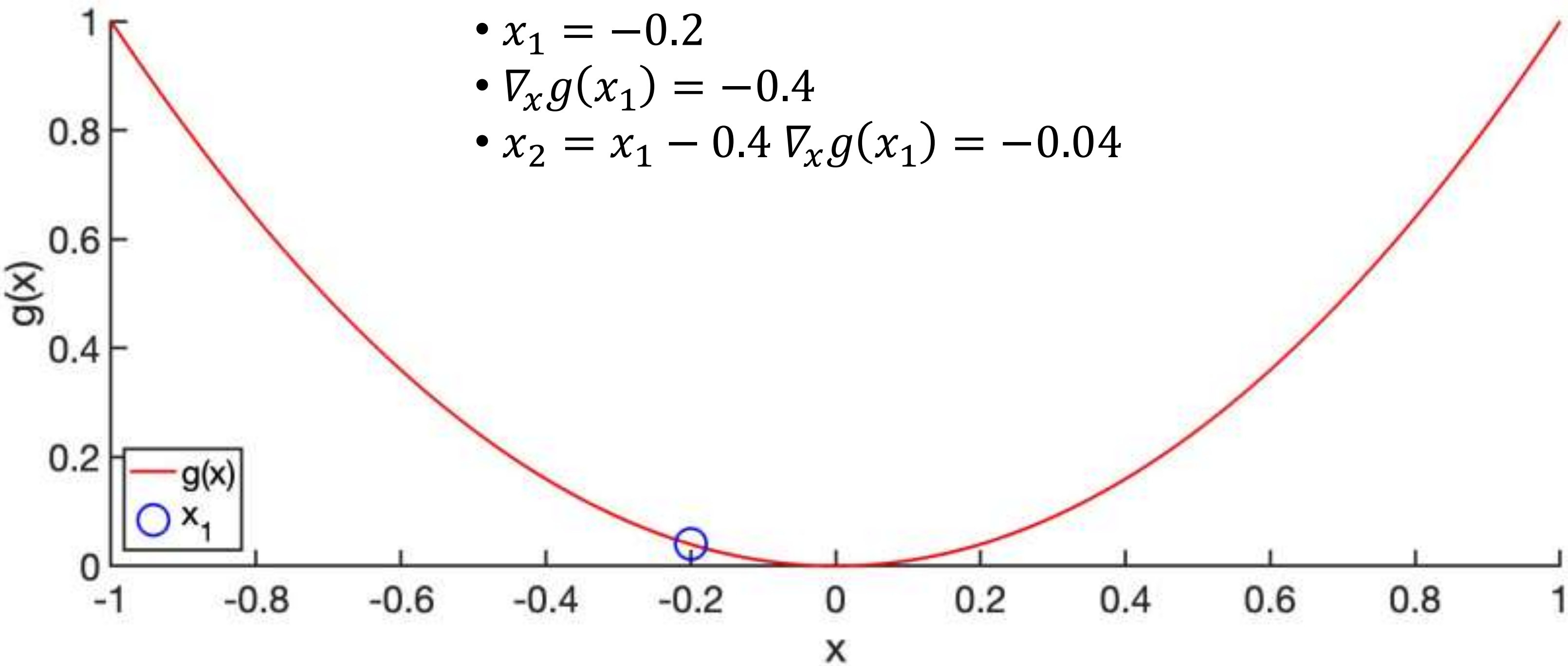
- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.4$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



Find x to minimize $g(x) = x^2$

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.4$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$

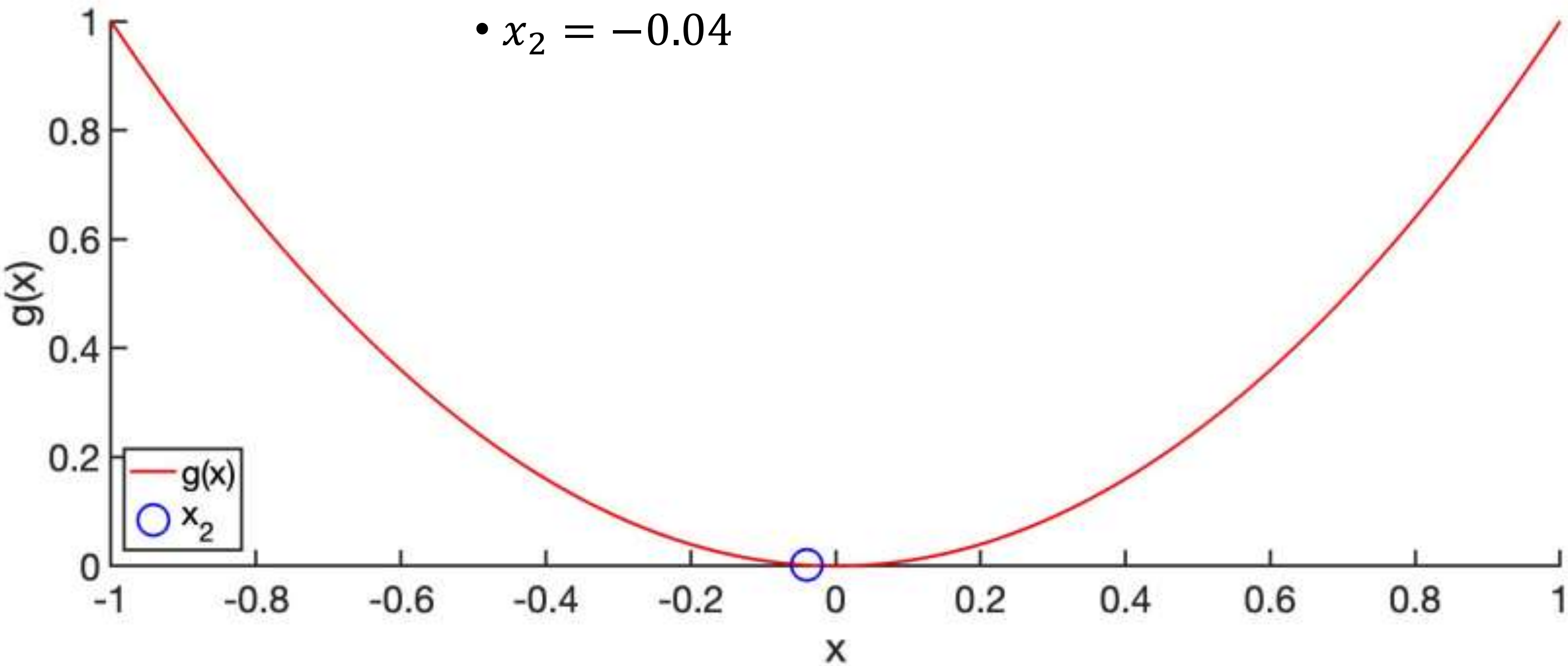
- $x_1 = -0.2$
- $\nabla_x g(x_1) = -0.4$
- $x_2 = x_1 - 0.4\, \nabla_x g(x_1) = -0.04$



Find x to minimize $g(x) = x^2$

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.4$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$

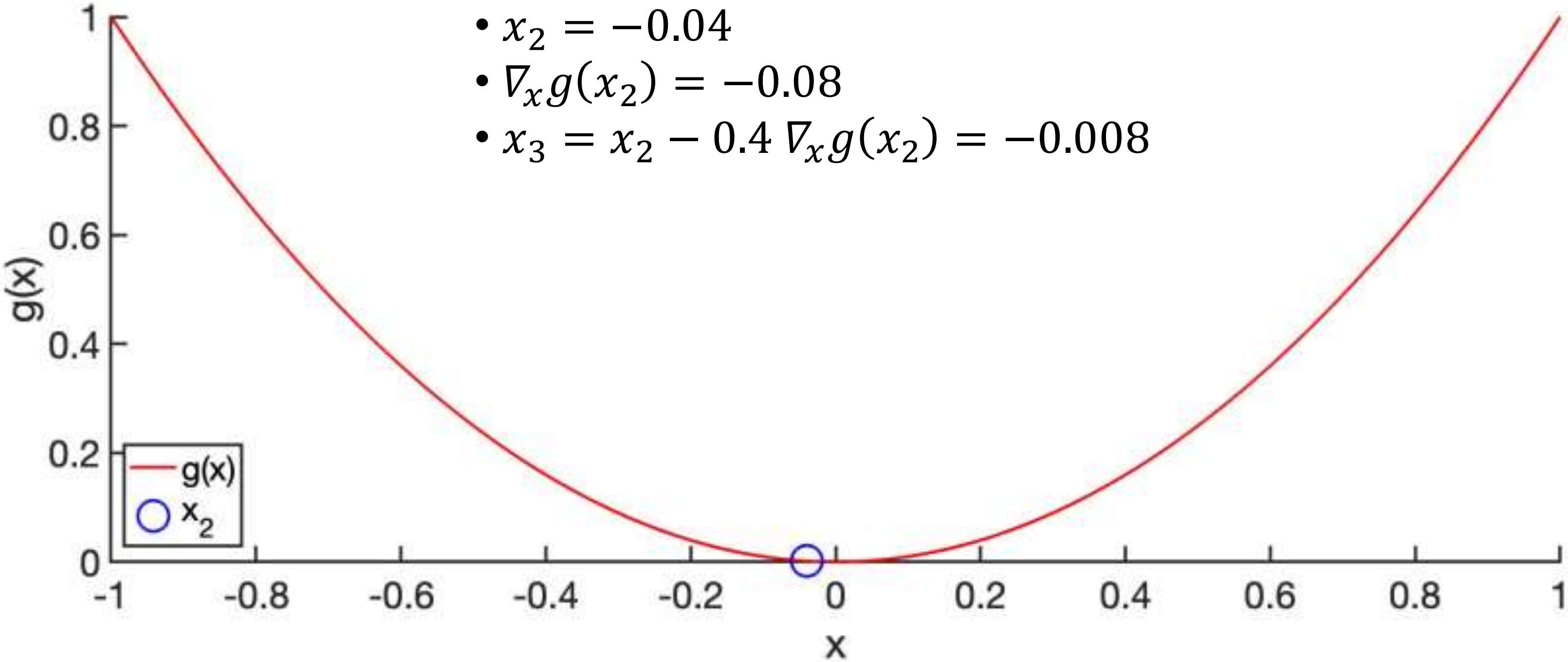
- $x_2 = -0.04$



Find x to minimize $g(x) = x^2$

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.4$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$

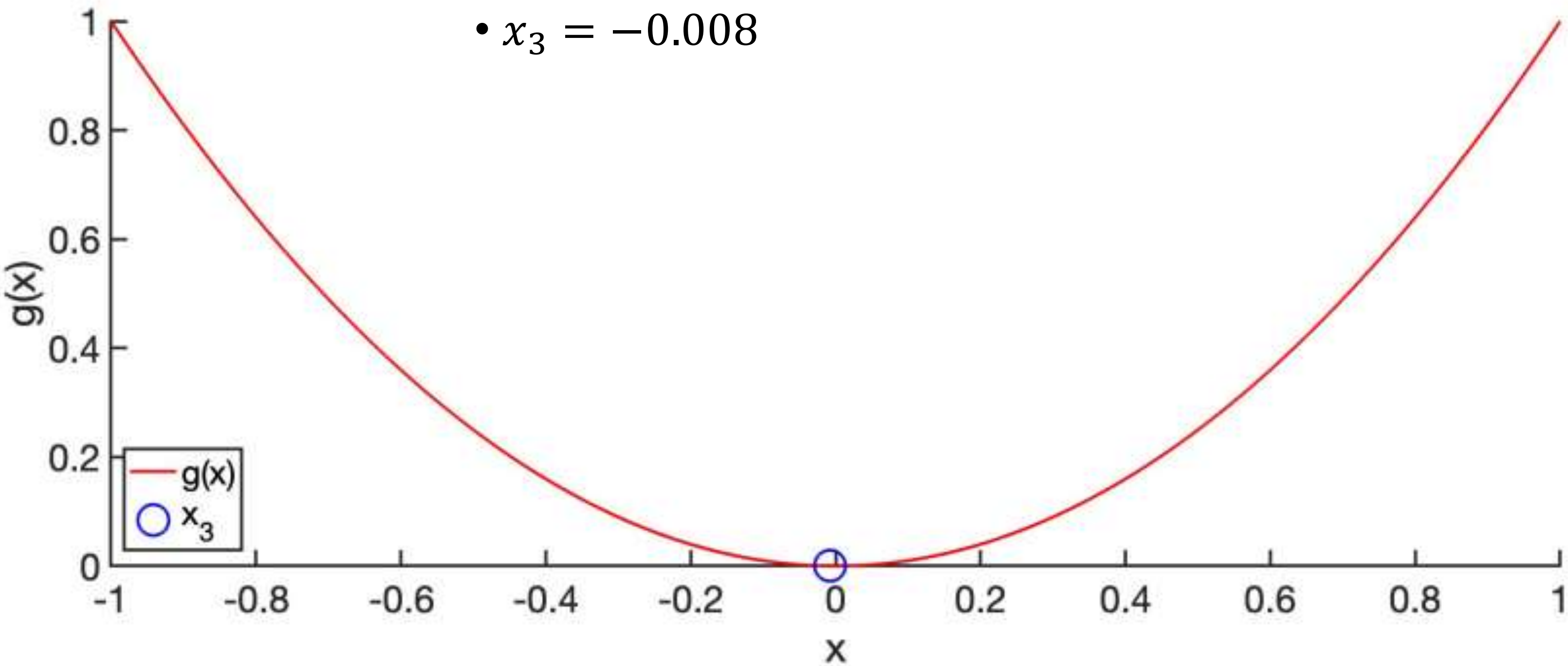
- $x_2 = -0.04$
- $\nabla_x g(x_2) = -0.08$
- $x_3 = x_2 - 0.4\, \nabla_x g(x_2) = -0.008$



Find x to minimize $g(x) = x^2$

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.4$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$

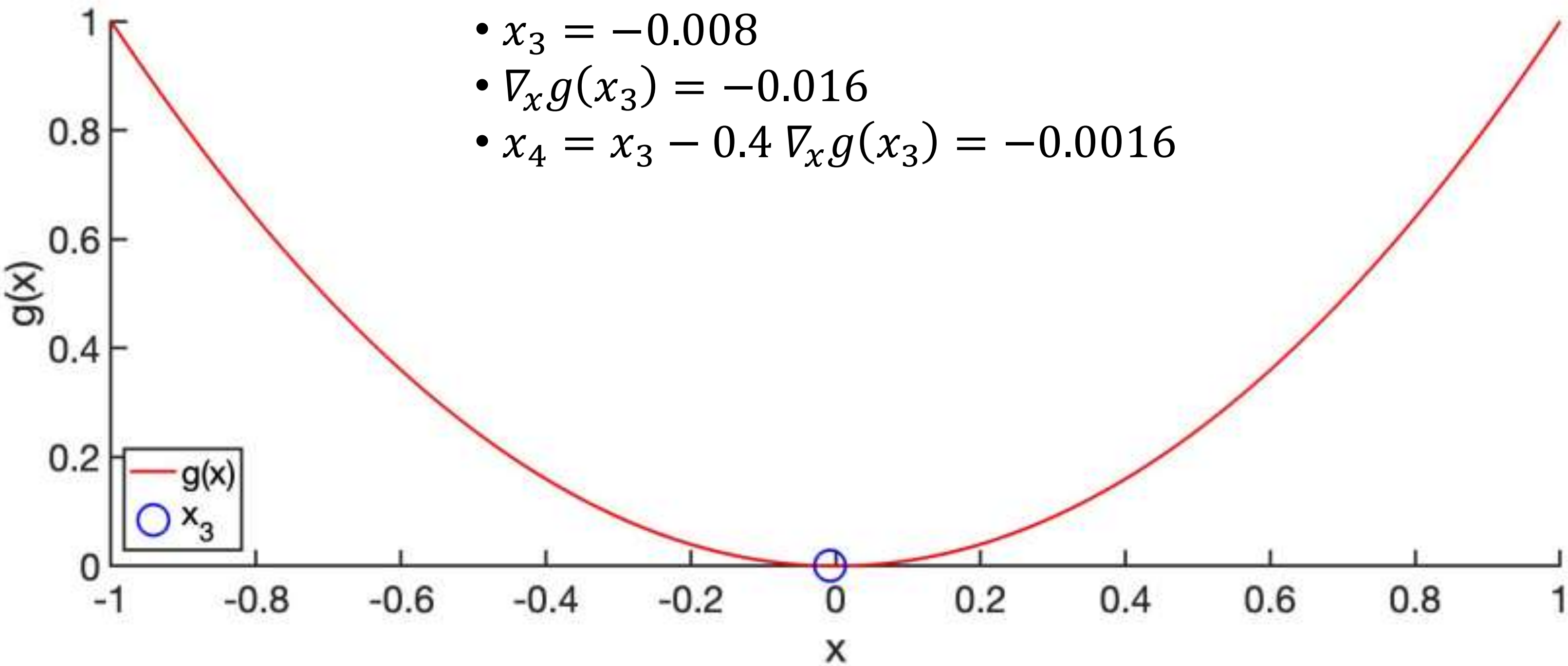
- $x_3 = -0.008$



Find x to minimize $g(x) = x^2$

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.4$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$

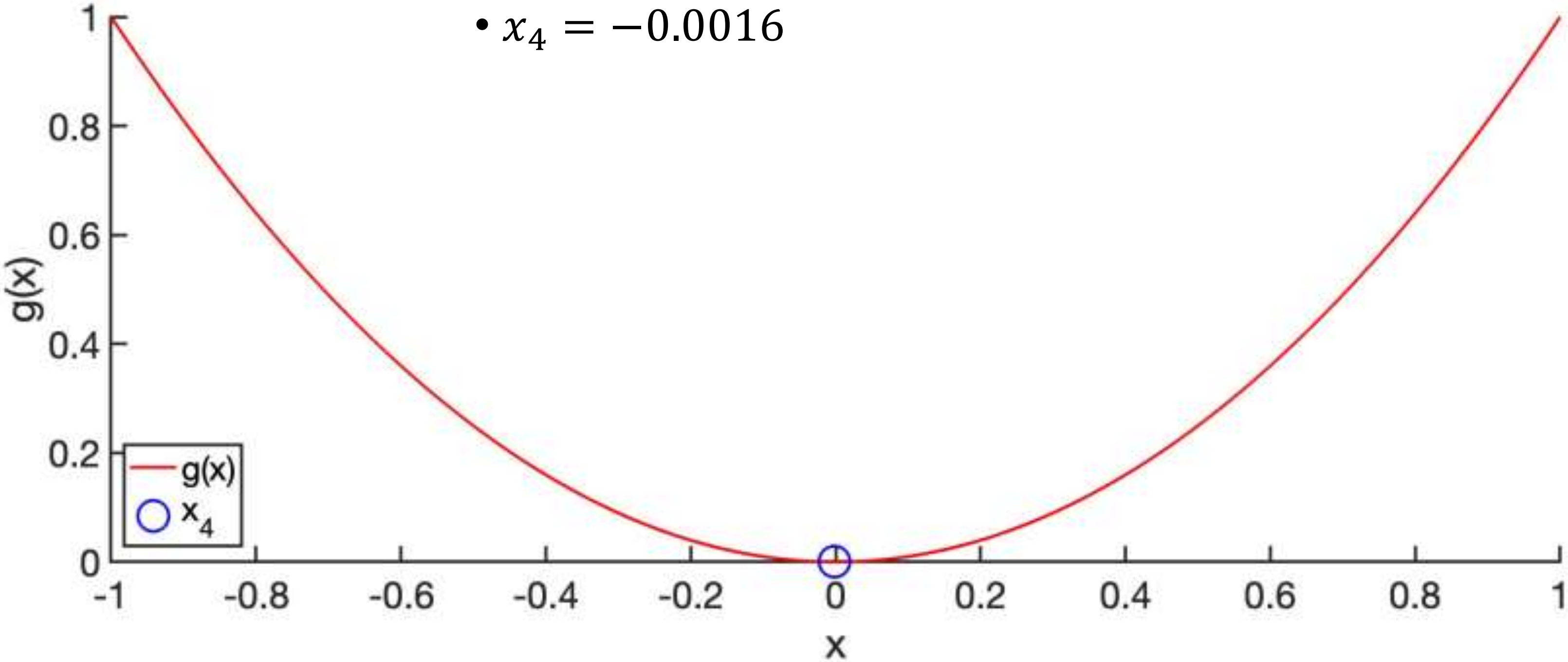
- $x_3 = -0.008$
- $\nabla_x g(x_3) = -0.016$
- $x_4 = x_3 - 0.4\, \nabla_x g(x_3) = -0.0016$



Find x to minimize $g(x) = x^2$

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.4$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$

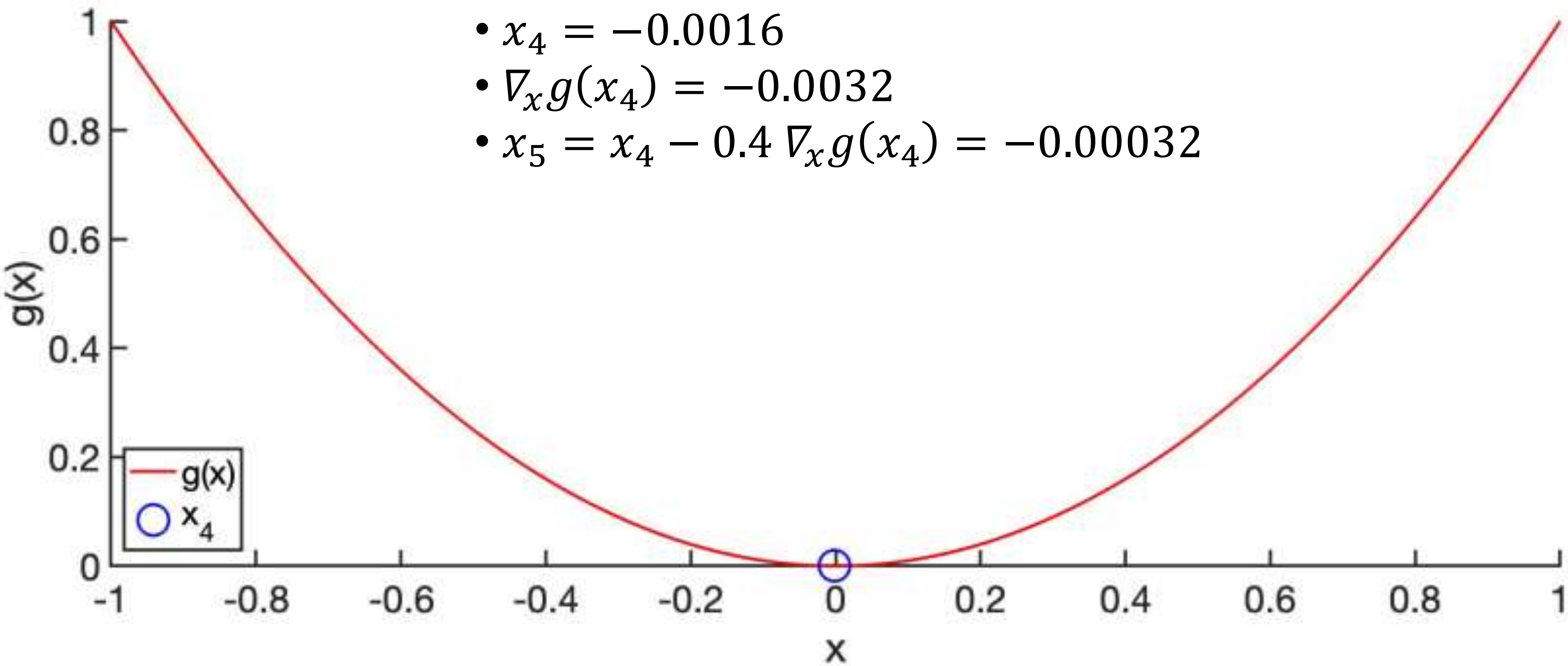
• $x_4 = -0.0016$



Find x to minimize $g(x) = x^2$

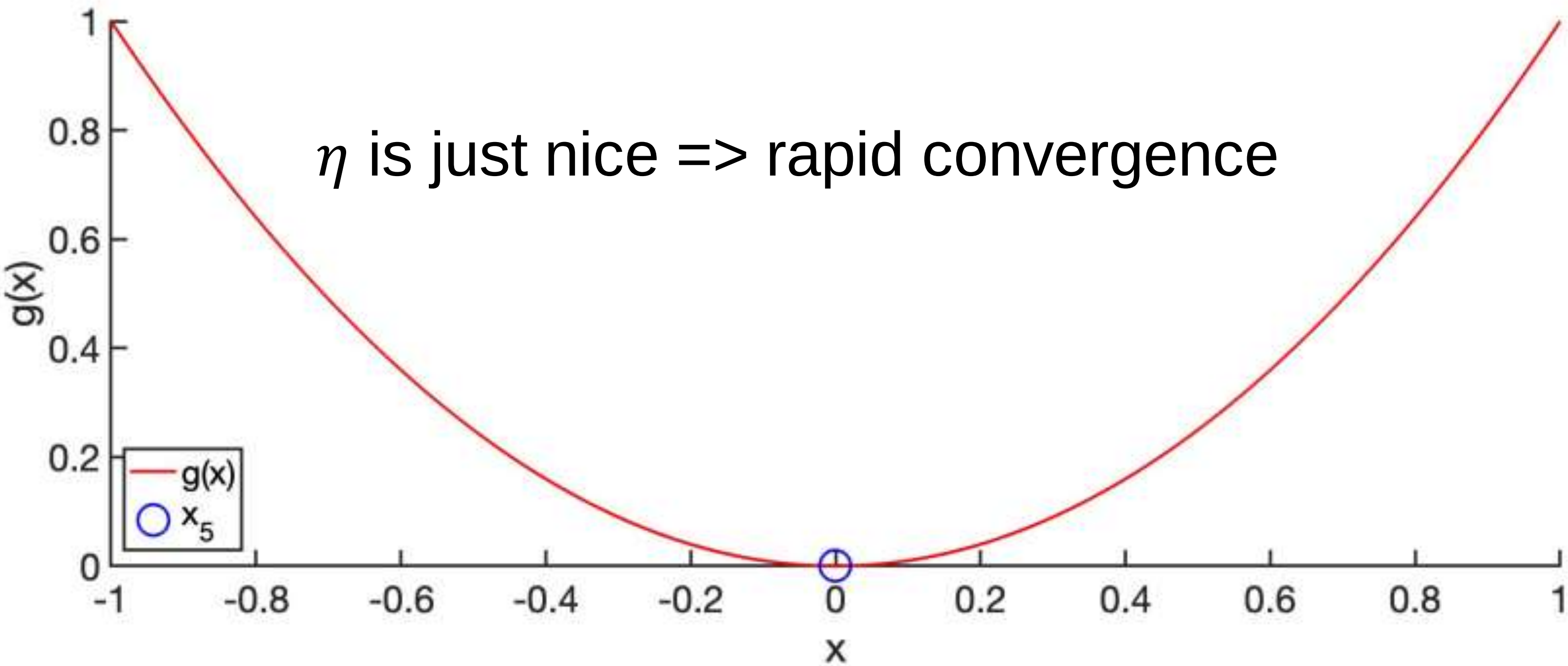
- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.4$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$

- $x_4 = -0.0016$
- $\nabla_x g(x_4) = -0.0032$
- $x_5 = x_4 - 0.4\,\nabla_x g(x_4) = -0.00032$



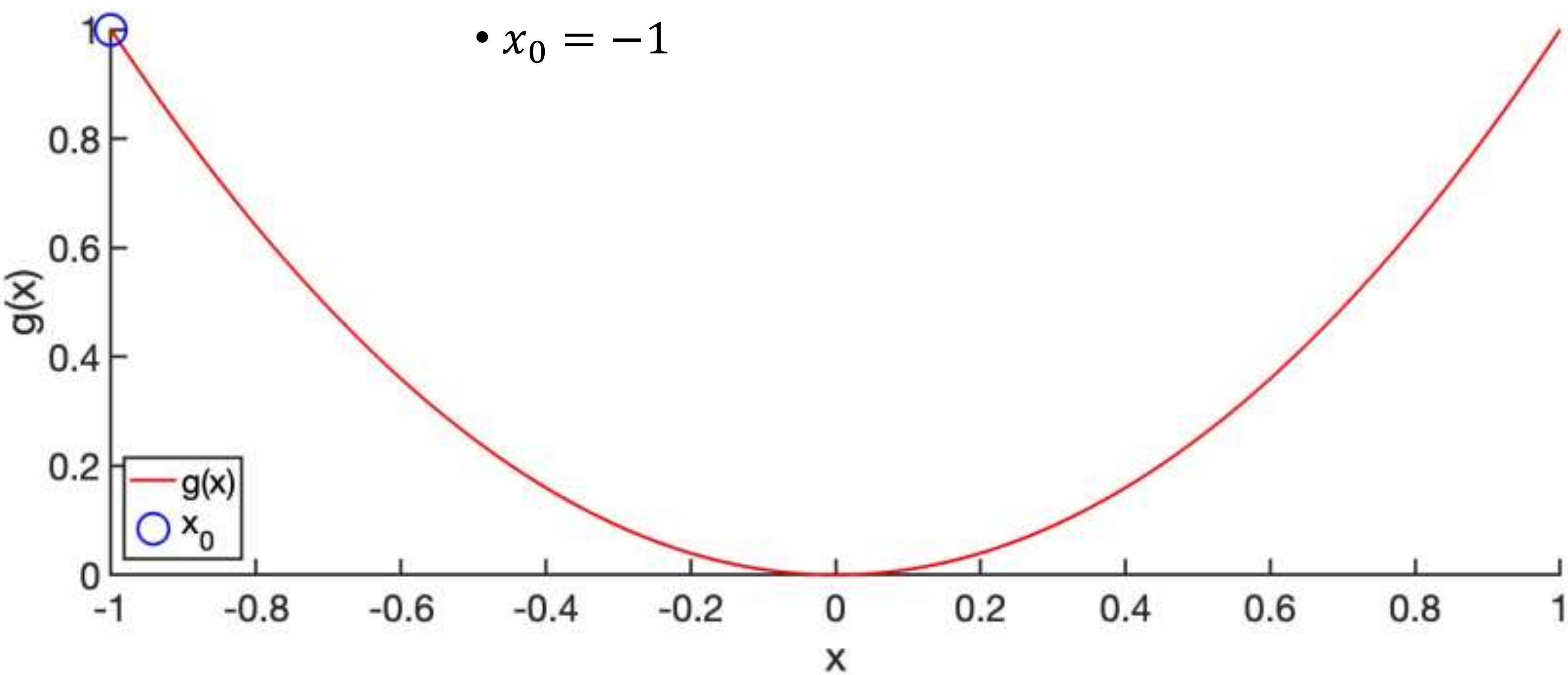
Find x to minimize $g(x) = x^2$

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.4$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



What if learning rate η is too small?

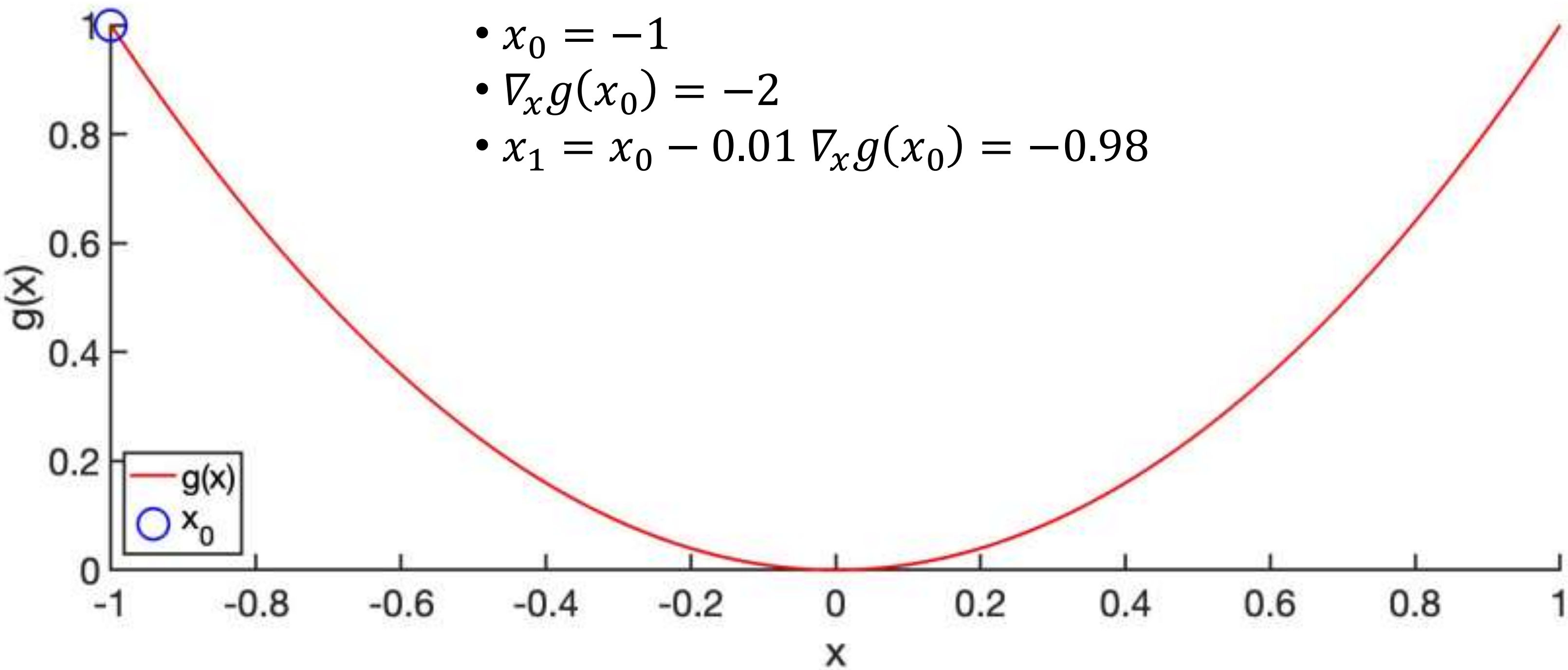
- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.01$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



What if learning rate η is too small?

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.01$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$

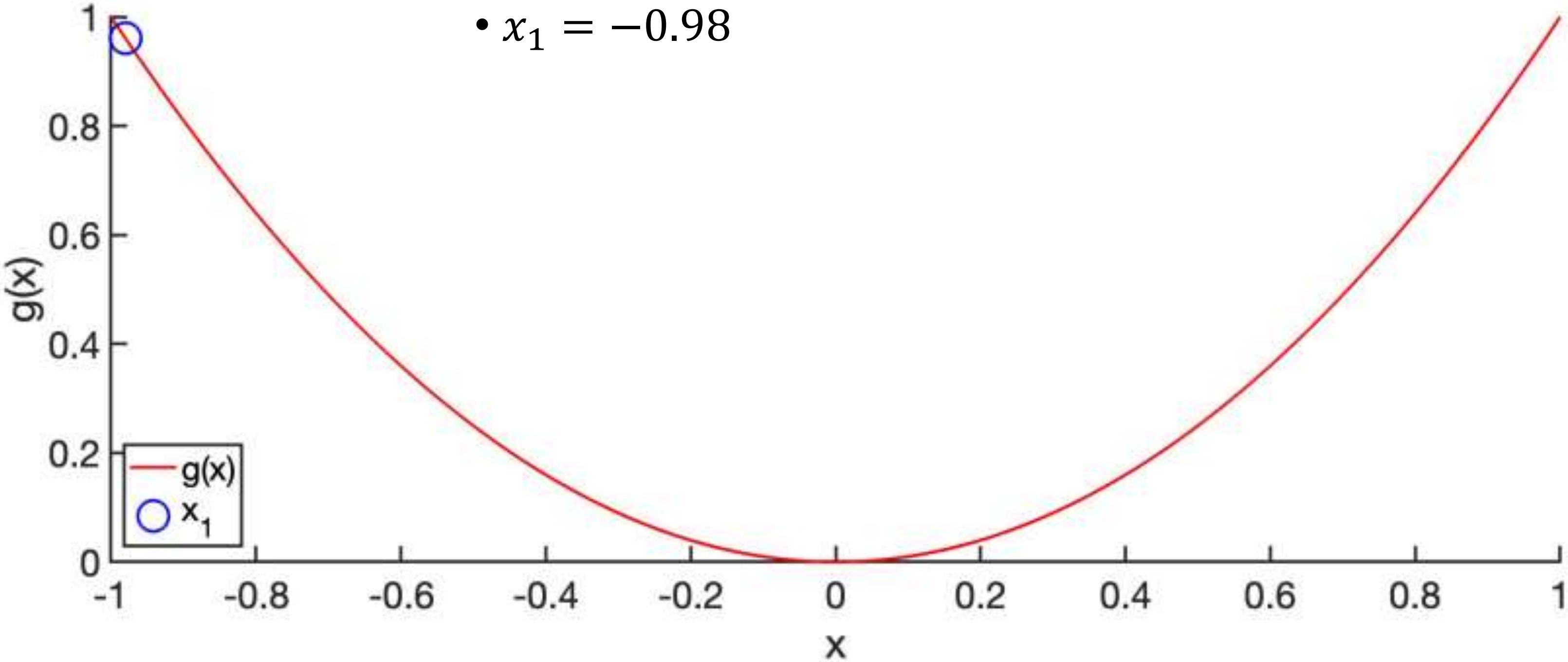
- $x_0 = -1$
- $\nabla_x g(x_0) = -2$
- $x_1 = x_0 - 0.01\, \nabla_x g(x_0) = -0.98$



What if learning rate η is too small?

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.01$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$

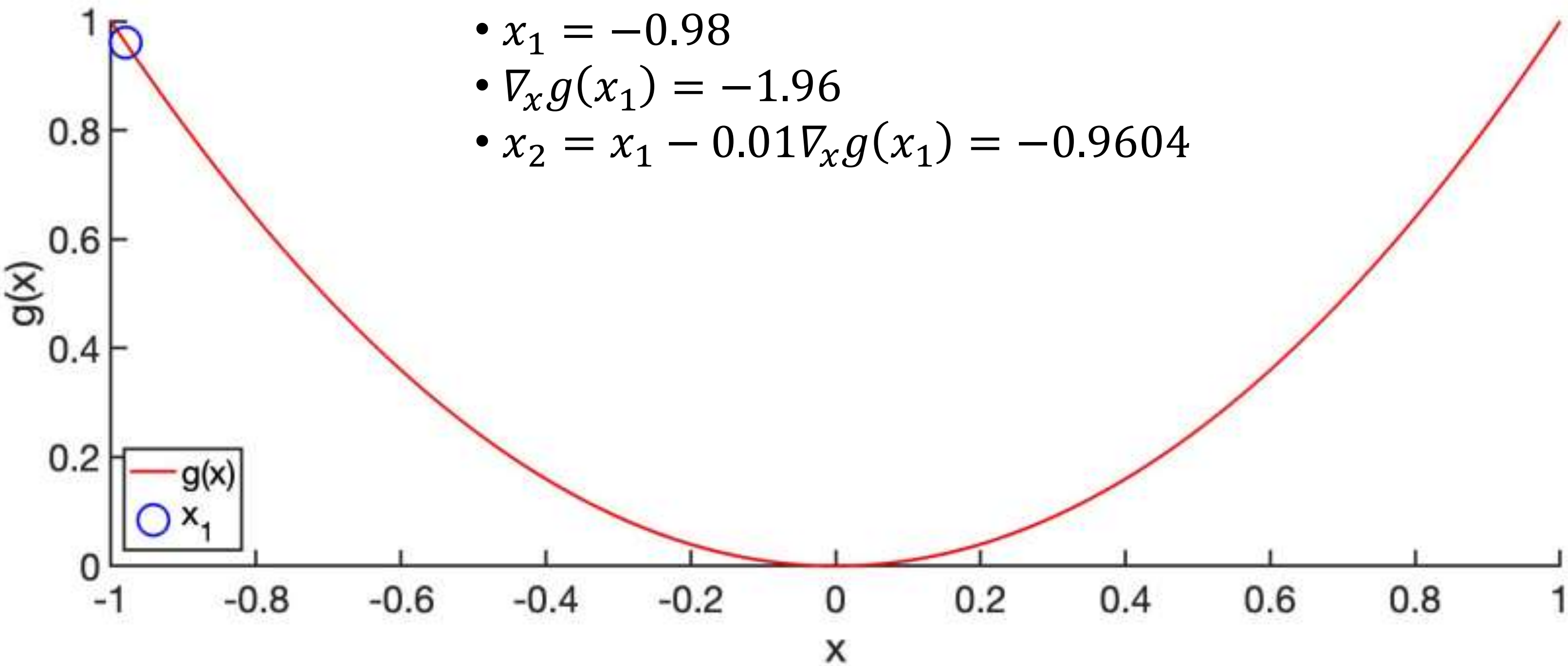
- $x_1 = -0.98$



What if learning rate η is too small?

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.01$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$

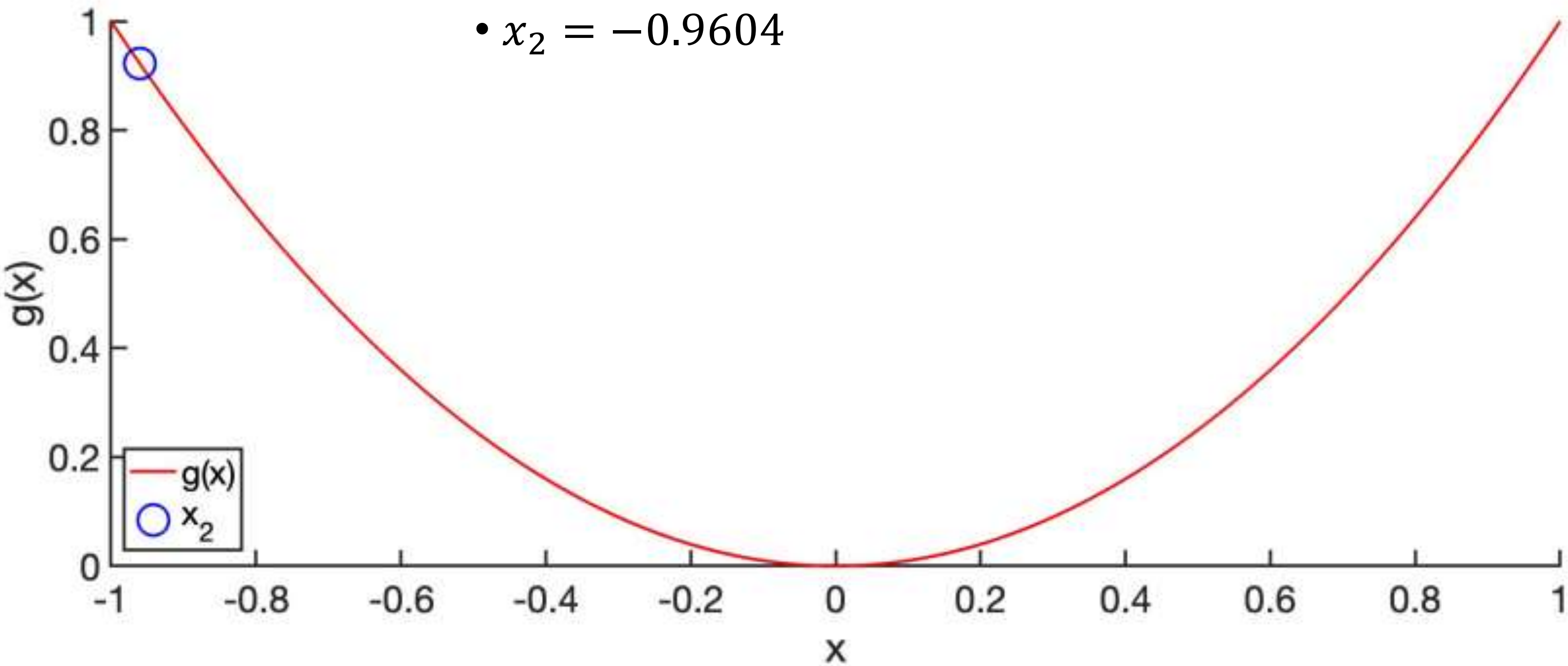
- $x_1 = -0.98$
- $\nabla_x g(x_1) = -1.96$
- $x_2 = x_1 - 0.01 \nabla_x g(x_1) = -0.9604$



What if learning rate η is too small?

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.01$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$

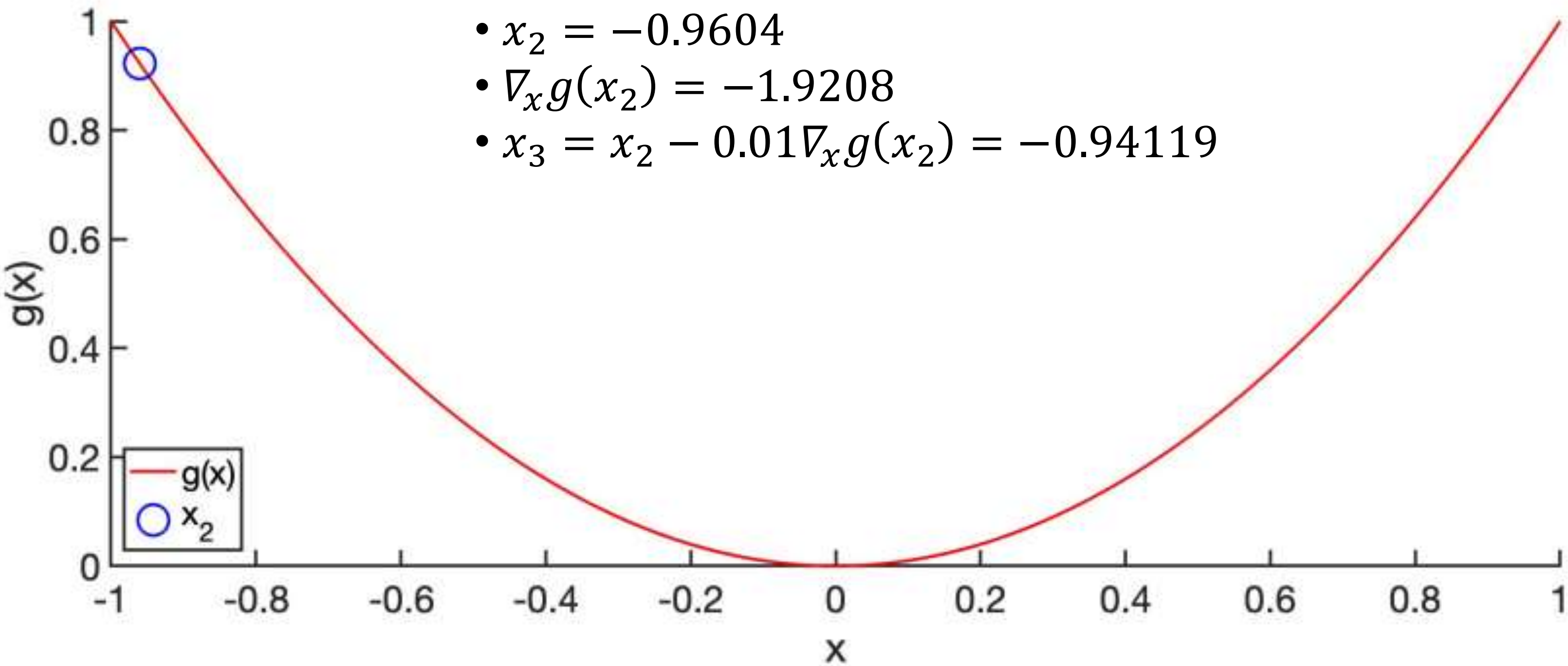
- $x_2 = -0.9604$



What if learning rate η is too small?

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.01$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$

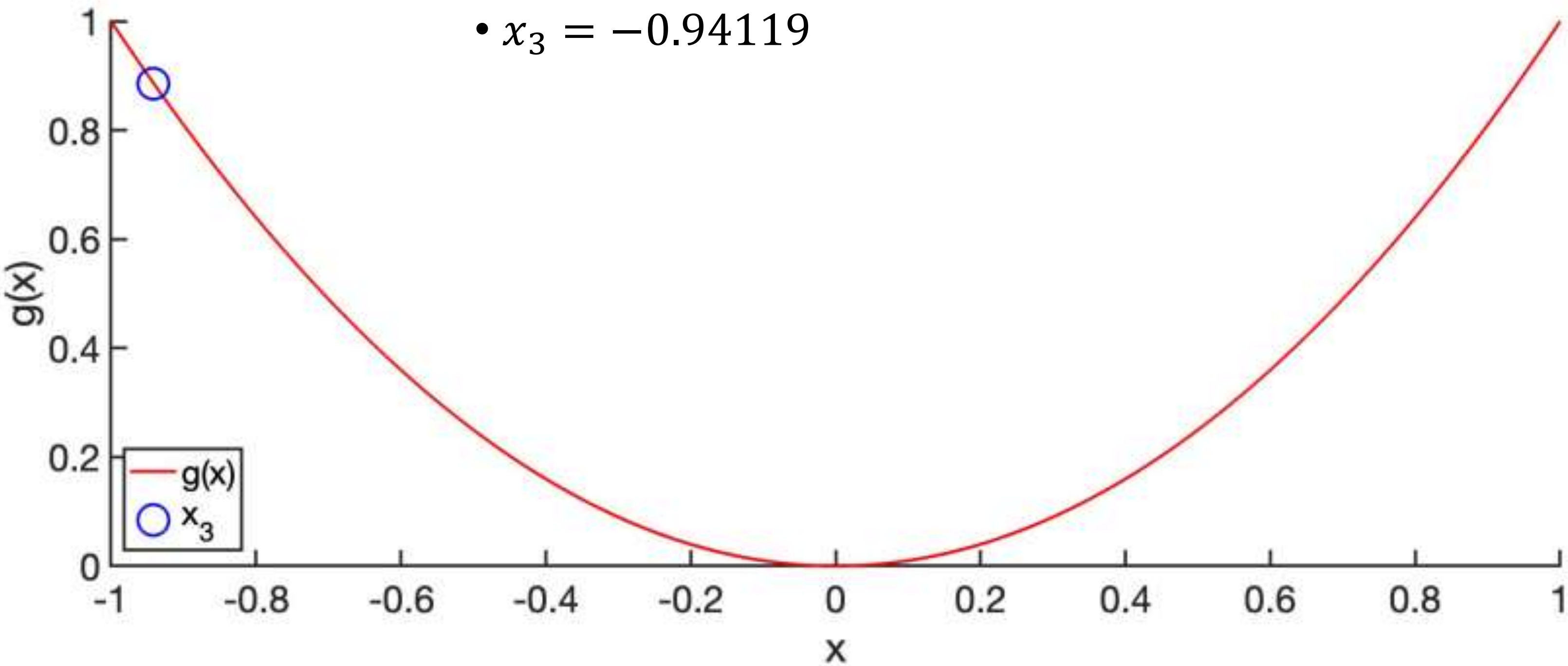
- $x_2 = -0.9604$
- $\nabla_x g(x_2) = -1.9208$
- $x_3 = x_2 - 0.01 \nabla_x g(x_2) = -0.94119$



What if learning rate η is too small?

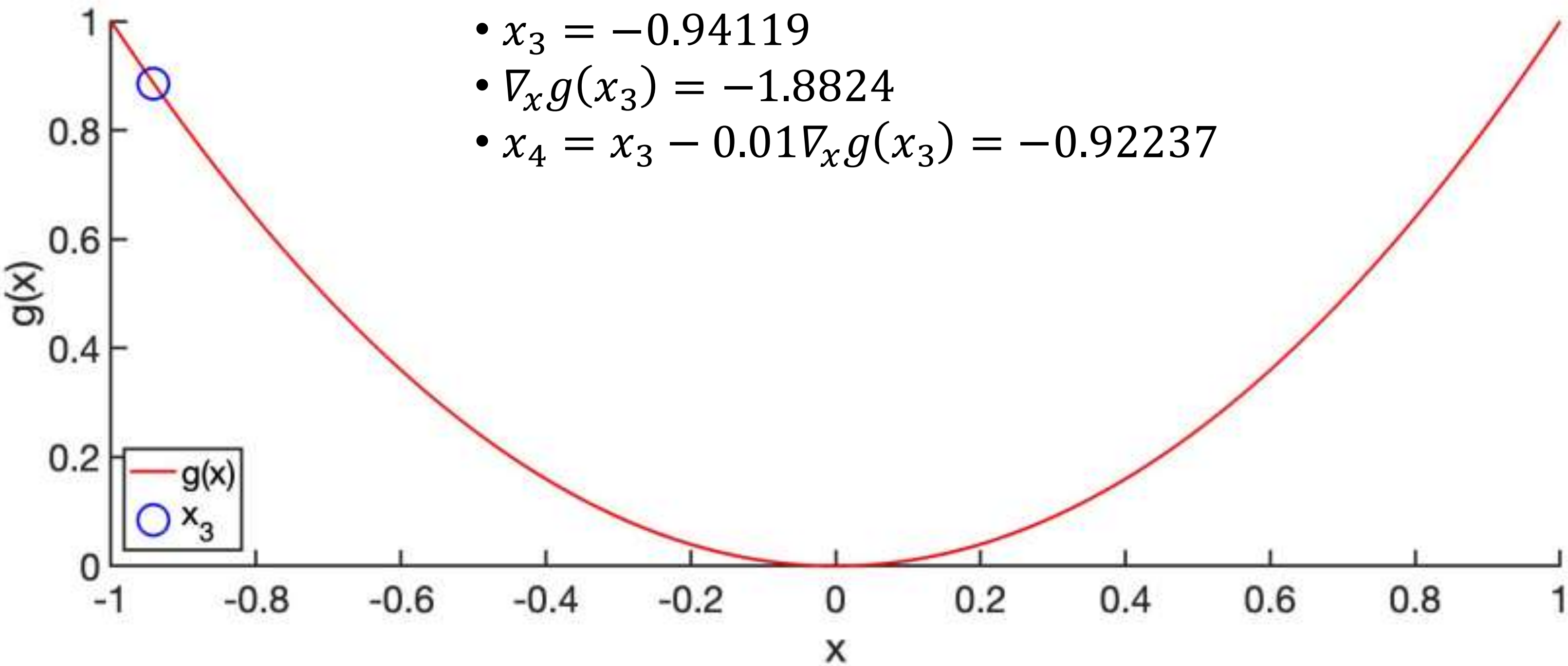
- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.01$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$

- $x_3 = -0.94119$



What if learning rate η is too small?

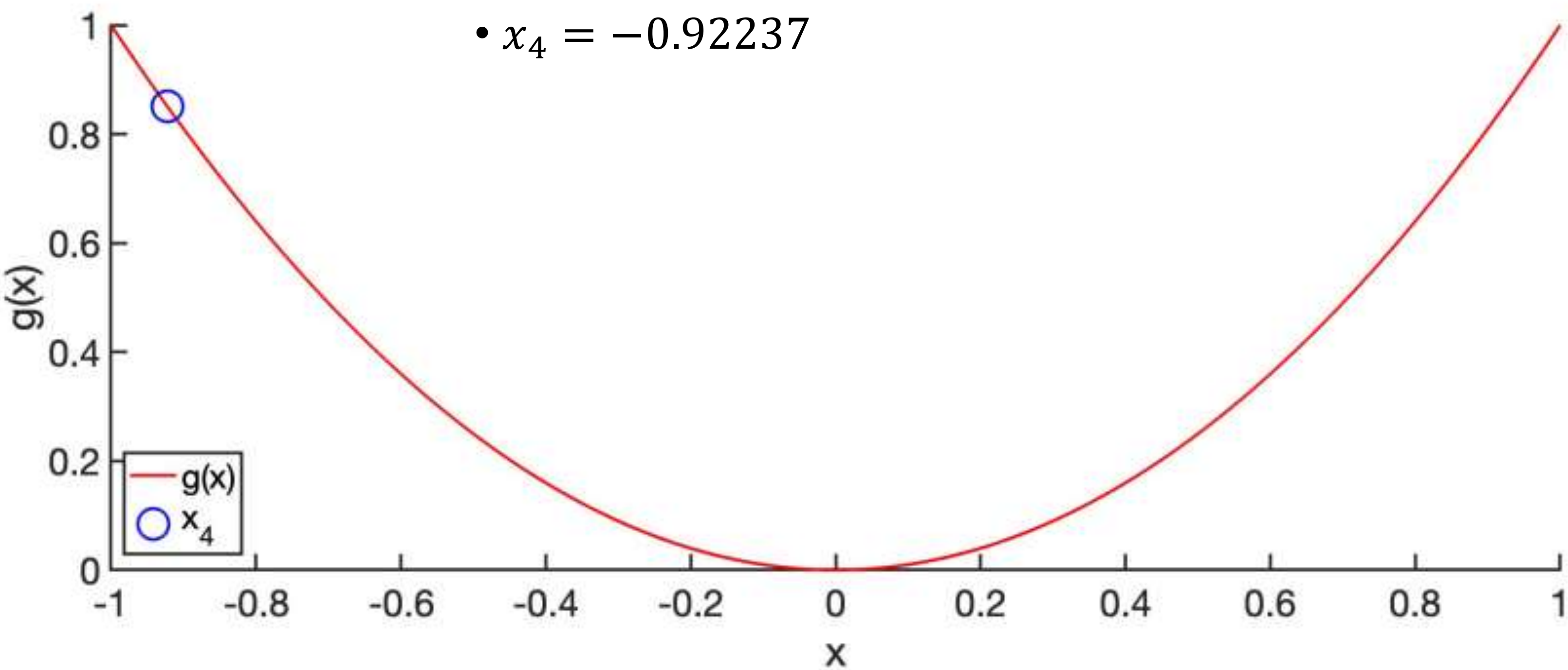
- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.01$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



- $x_3 = -0.94119$
- $\nabla_x g(x_3) = -1.8824$
- $x_4 = x_3 - 0.01\nabla_x g(x_3) = -0.92237$

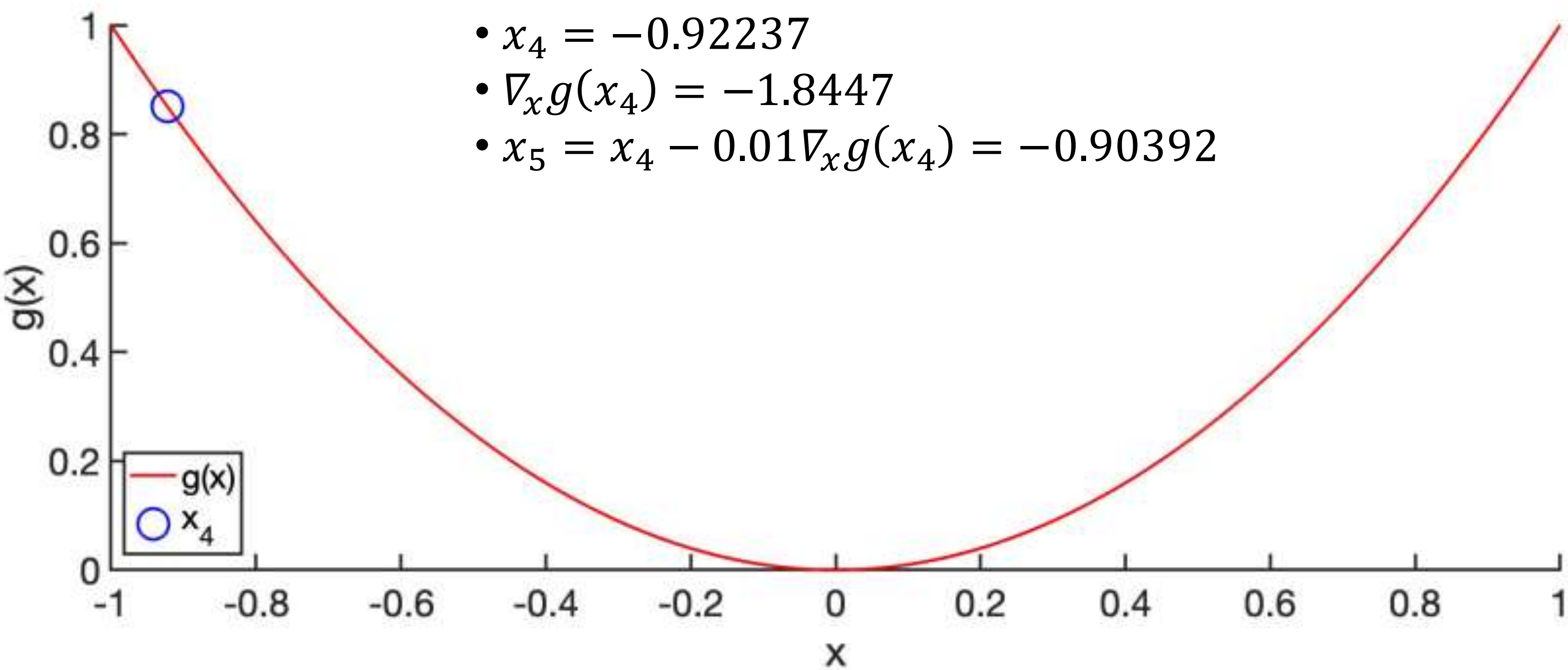
What if learning rate η is too small?

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.01$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



What if learning rate η is too small?

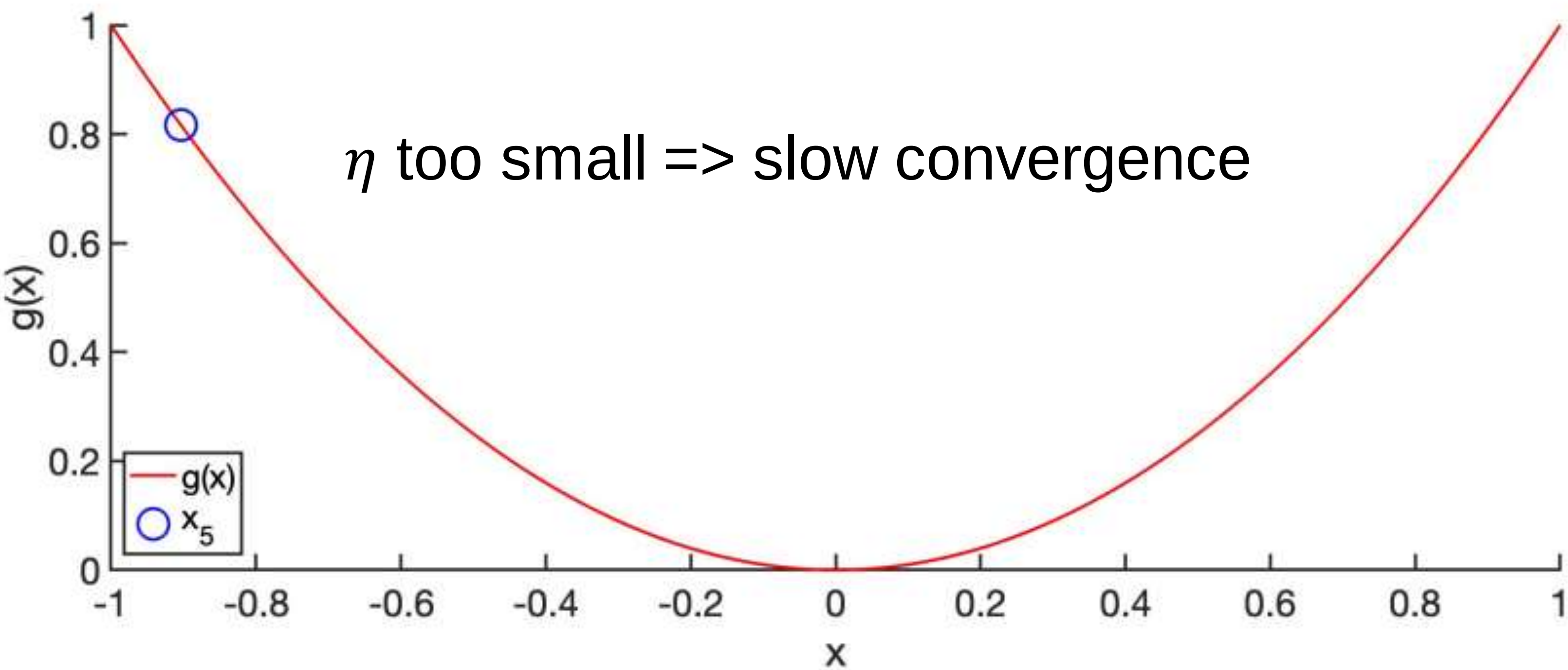
- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.01$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



- $x_4 = -0.92237$
- $\nabla_x g(x_4) = -1.8447$
- $x_5 = x_4 - 0.01\nabla_x g(x_4) = -0.90392$

What if learning rate η is too small?

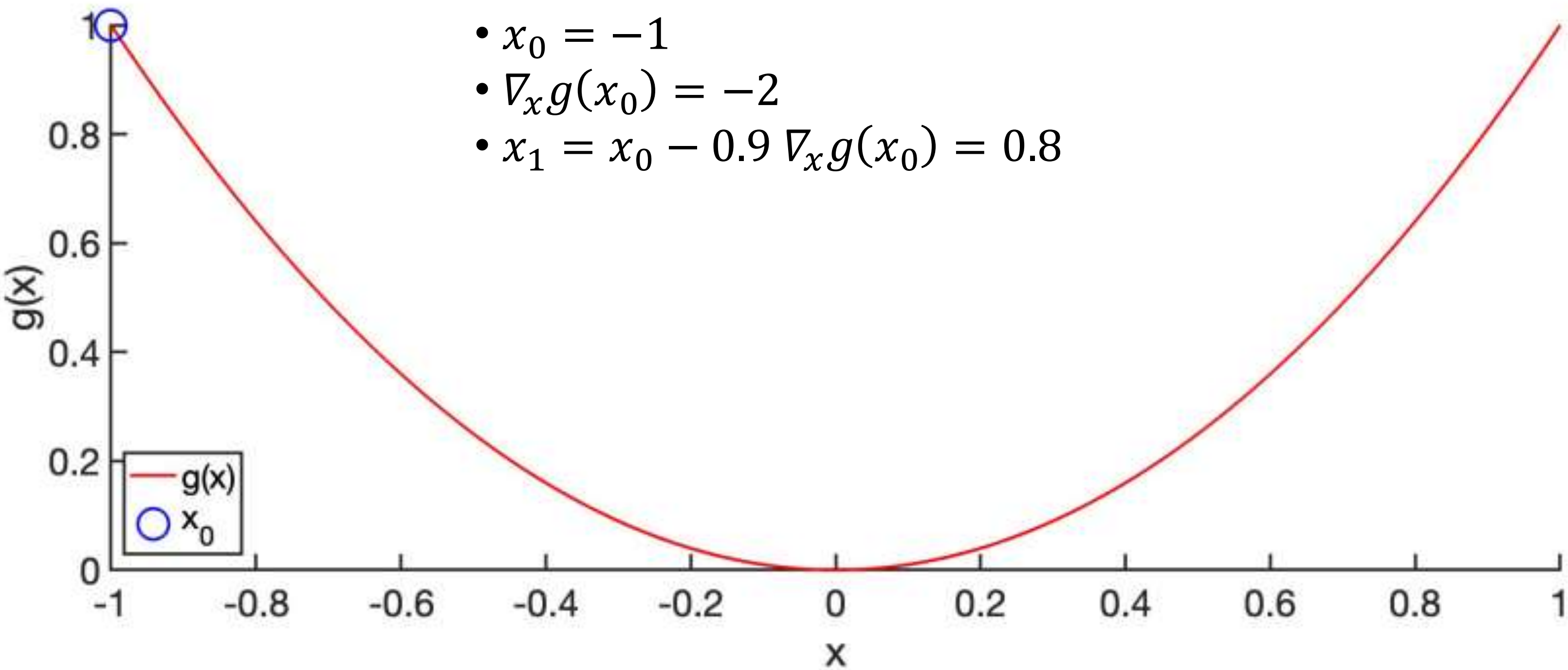
- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.01$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



What if learning rate η is too big?

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.9$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$

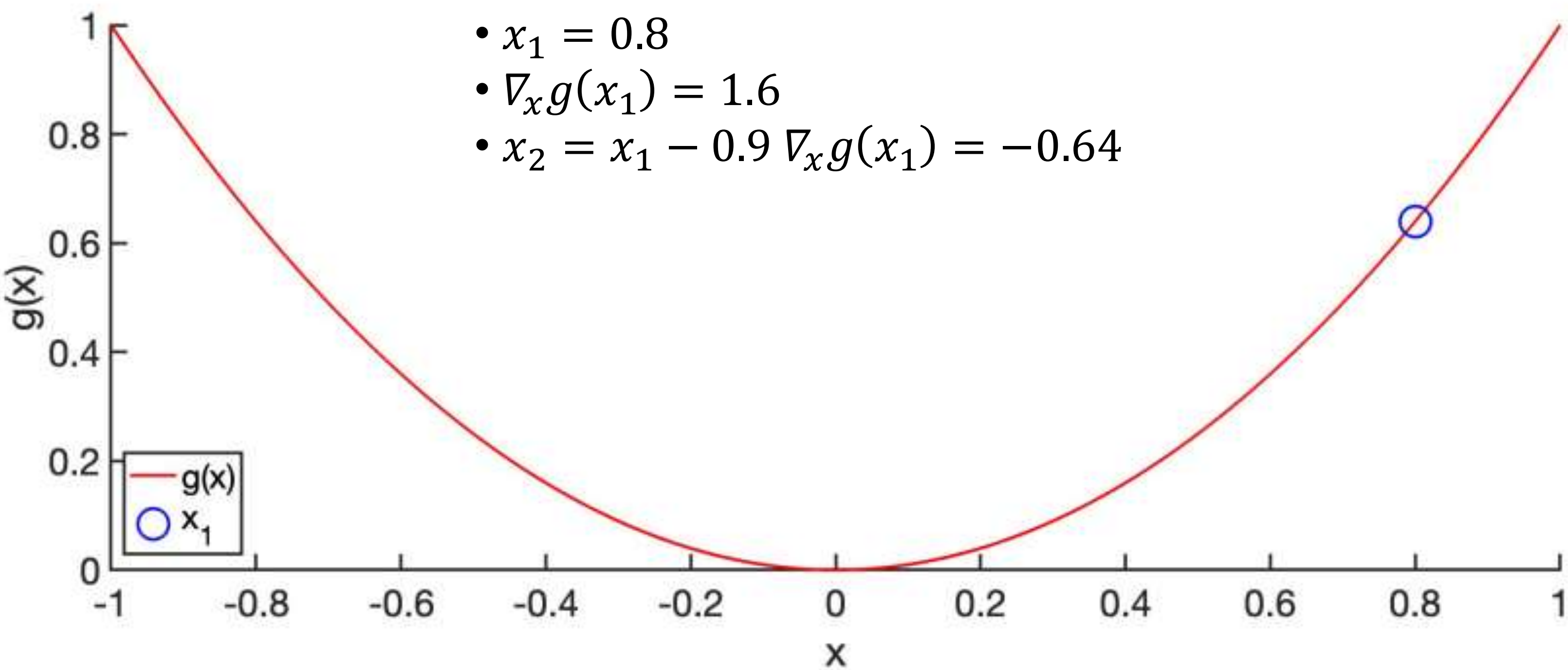
- $x_0 = -1$
- $\nabla_x g(x_0) = -2$
- $x_1 = x_0 - 0.9\, \nabla_x g(x_0) = 0.8$



What if learning rate η is too big?

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.9$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$

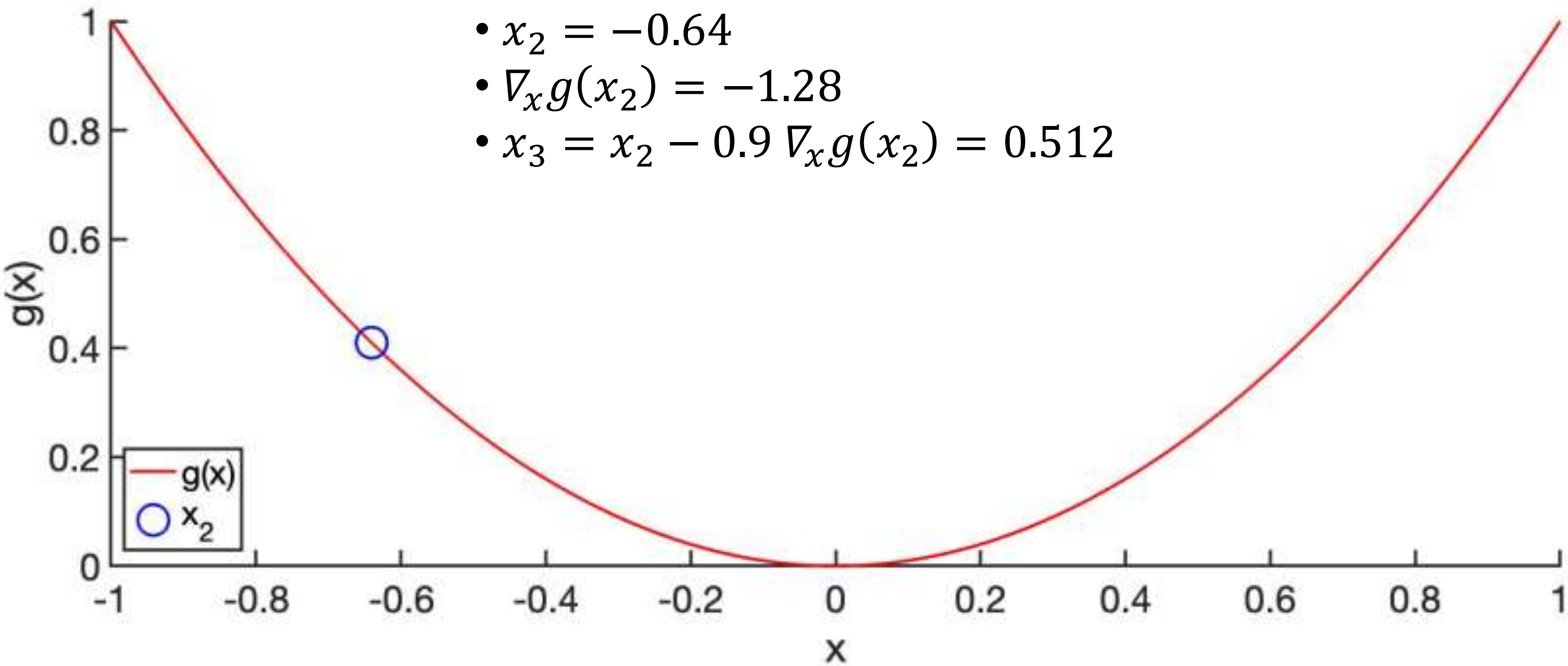
- $x_1 = 0.8$
- $\nabla_x g(x_1) = 1.6$
- $x_2 = x_1 - 0.9\, \nabla_x g(x_1) = -0.64$



What if learning rate η is too big?

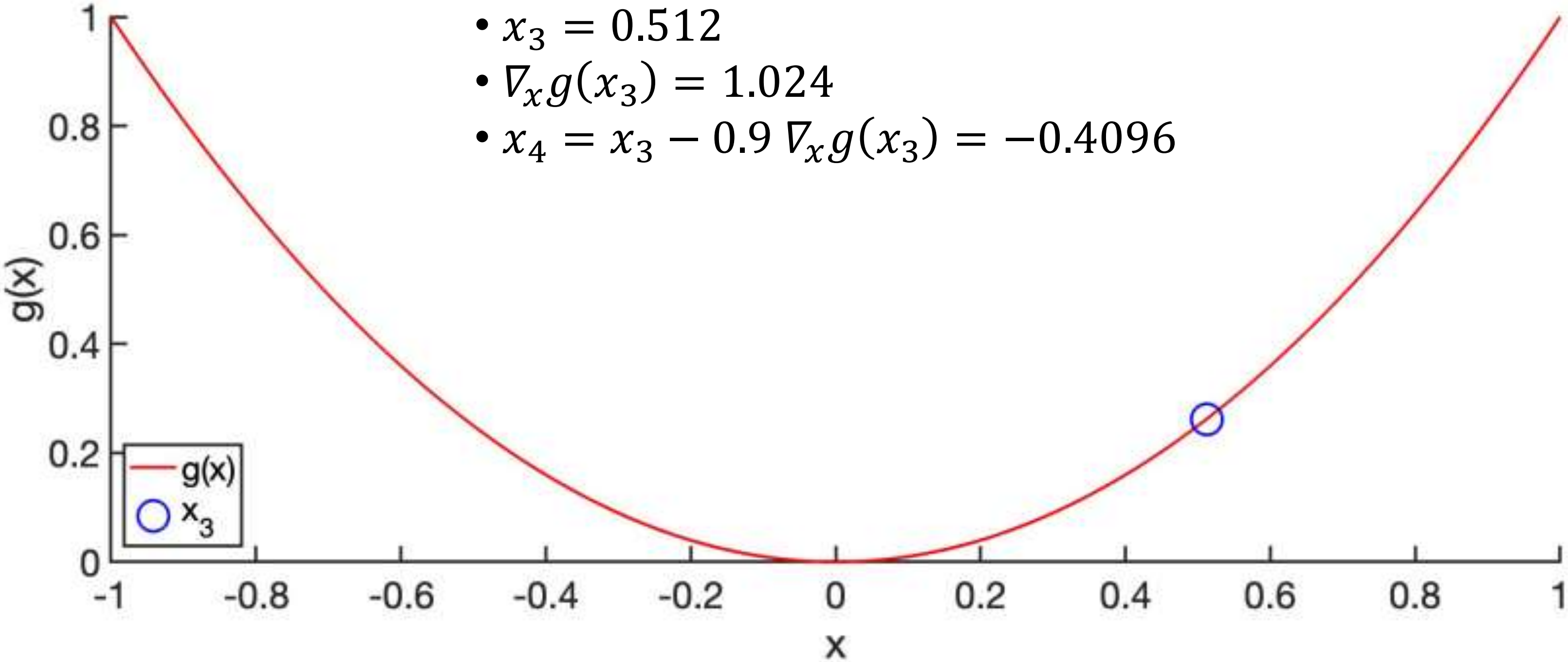
- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.9$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$

- $x_2 = -0.64$
- $\nabla_x g(x_2) = -1.28$
- $x_3 = x_2 - 0.9\, \nabla_x g(x_2) = 0.512$



What if learning rate η is too big?

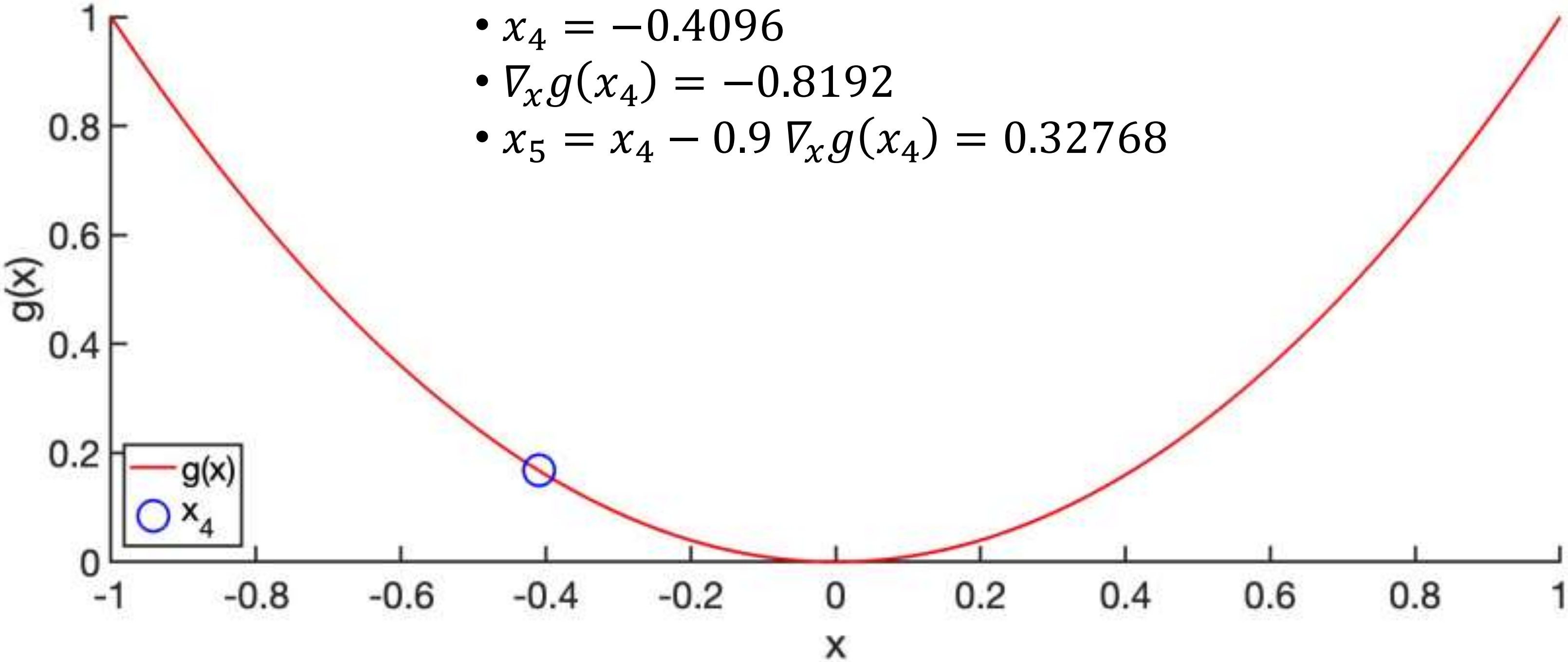
- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.9$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



- $x_3 = 0.512$
- $\nabla_x g(x_3) = 1.024$
- $x_4 = x_3 - 0.9\, \nabla_x g(x_3) = -0.4096$

What if learning rate η is too big?

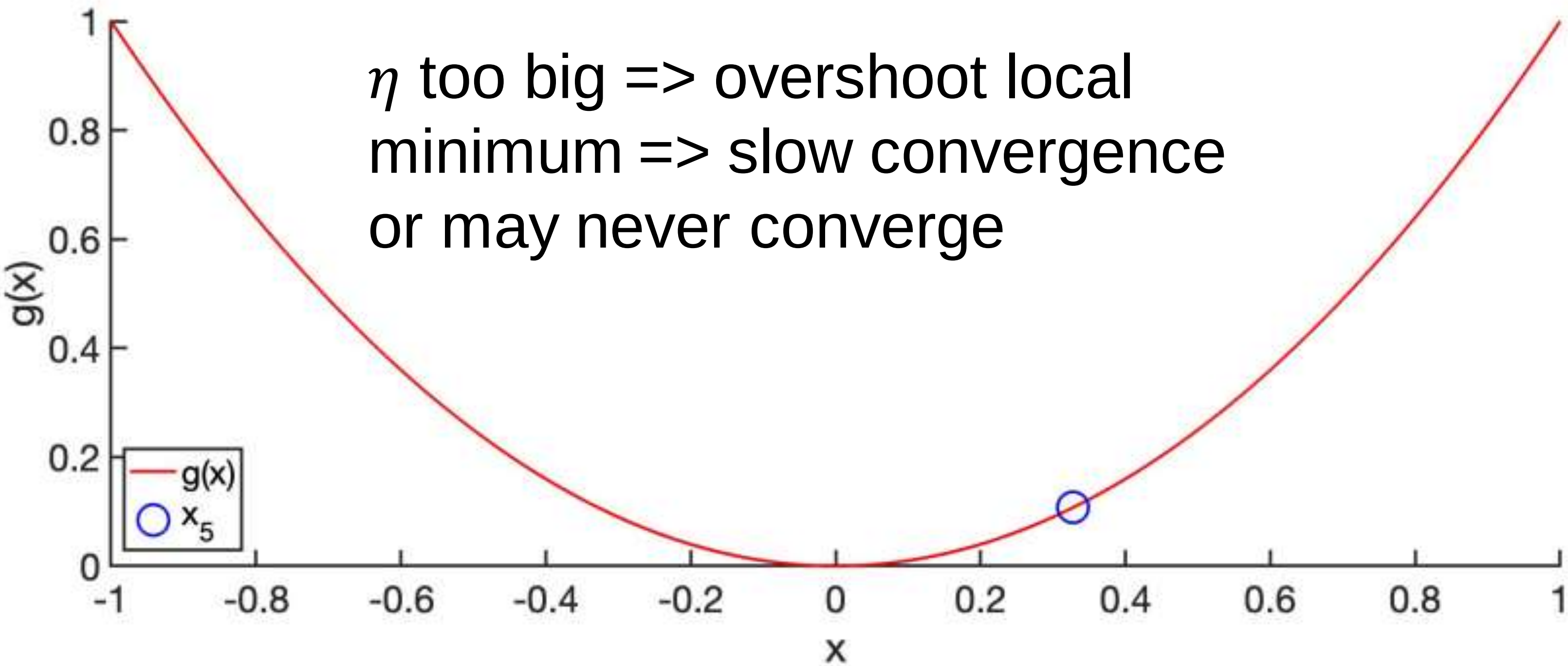
- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.9$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



- $x_4 = -0.4096$
- $\nabla_x g(x_4) = -0.8192$
- $x_5 = x_4 - 0.9\, \nabla_x g(x_4) = 0.32768$

What if learning rate η is too big?

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.9$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



η too big => overshoot local minimum => slow convergence or may never converge

Questions?

Python Demo on a Quadratic Function

```python
import numpy as np
from mpl_toolkits.mplot3d import Axes3D  # noqa: F401 unused import
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
```

```python
Q = np.array([[1, 0], [0, 3]])

[x1,x2] = np.meshgrid(np.linspace(-10,10,1001),np.linspace(-10,10,1001))

x_vals = np.linspace(-10, 10, 1001)
y_vals = np.linspace(-10, 10, 1001)
X, Y = np.meshgrid(x_vals, y_vals)

Z = 0.5*(Q[0,0]*X**2 + Q[1,1]*Y**2 + 2*Q[0,1]*X*Y)

x_iter = np.array([[5], [3]]);
notConverged = 1;

lambdas, v = np.linalg.eig(Q)

eta = 2/(np.max(lambdas)+np.min(lambdas))
```

Here, we are defining a quadratic function

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{Q}\mathbf{x}$$

Finding the optimal step size (don’t need to know)

Python Demo on a Quadratic Function

```
iter = 0
x = np.zeros([2,1000])

while notConverged and iter < 1e3:
    plt.plot(x_iter[0],x_iter[1],'gx')
    x[:,iter] = x_iter.T
    x_iter = x_iter - eta*Q.dot(x_iter)

    if (np.linalg.norm(x_iter) < 1e-5):
        notConverged = 0

    iter = iter + 1

cp = plt.contour(X, Y, Z, np.linspace(0,200,10))
plt.plot(x[0,0:iter-1],x[1,0:iter-1],'b--')
#plt.savefig('well_conditioned.eps', format='eps')
```

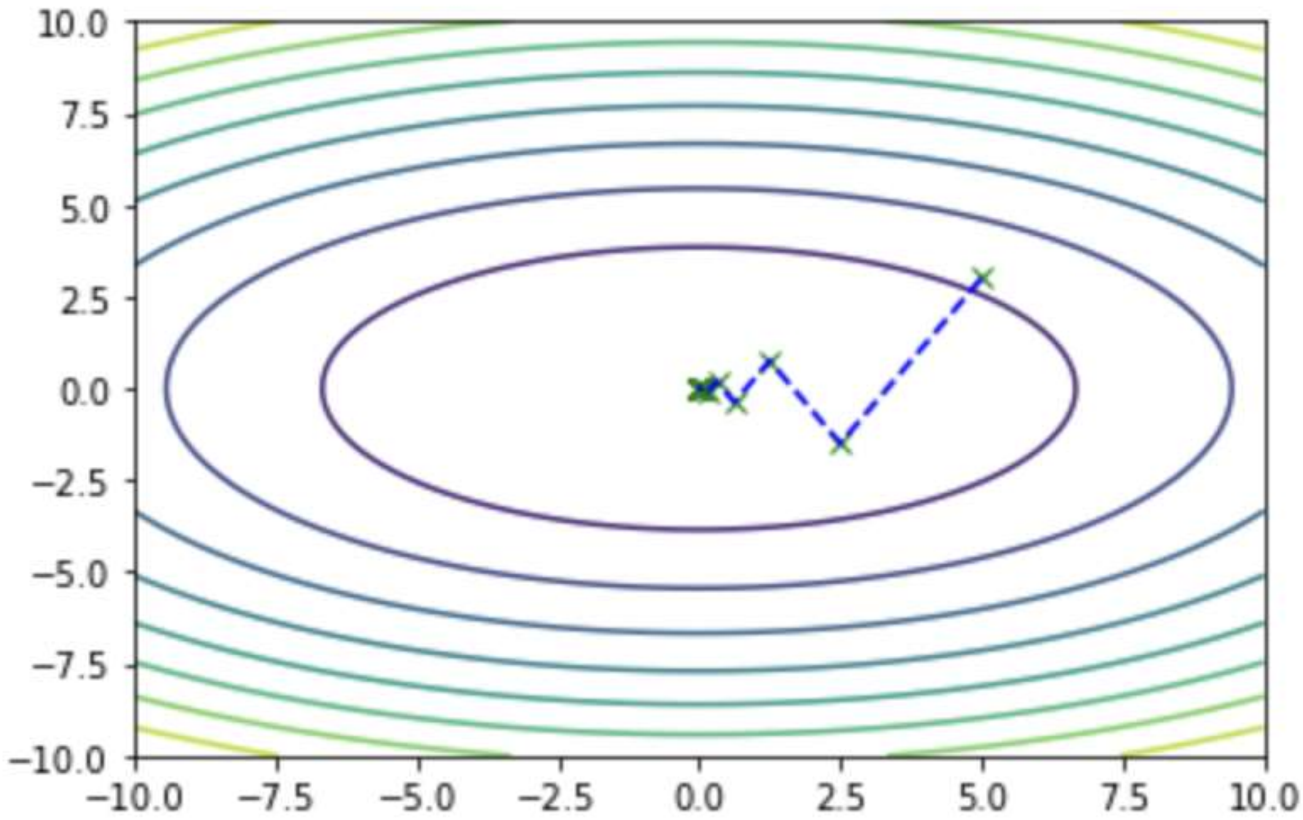
Running gradient descent.

Note that the gradient of f is $\nabla f(\mathbf{x}) = \mathbf{Q}\mathbf{x}$

Plotting contours (don't need to know)

Automatic saving

Python Demo



Convergence to the foot of the valley. Experiment with different values of the step size!

Questions?

Different Learning Models

- Different learning models $f(\mathbf{x}_i, \mathbf{w})$ reflect our beliefs about the relationship between the features $\mathbf{x}_i$ and target y_i

Different Learning Models

- Different learning models $f(\mathbf{x}_i, \mathbf{w})$ reflect our beliefs about the relationship between the features $\mathbf{x}_i$ and target y_i
  - For example, $f(\mathbf{x}_i, \mathbf{w}) = \mathbf{p}_i^T \mathbf{w}$ assumes polynomial relationship between features and target

Different Learning Models

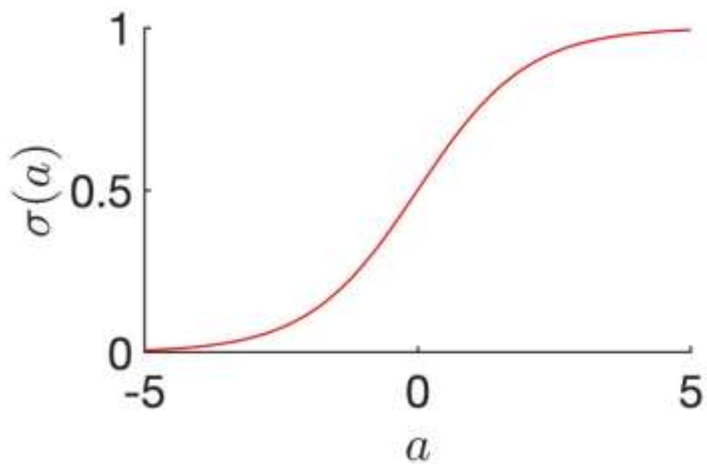
- Different learning models $f(\mathbf{x}_i, \mathbf{w})$ reflect our beliefs about the relationship between the features $\mathbf{x}_i$ and target y_i
  - For example, $f(\mathbf{x}_i, \mathbf{w}) = \mathbf{p}_i^T\mathbf{w}$ assumes polynomial relationship between features and target
- Suppose we are performing classification (rather than regression), so y_i is class -1 or class 1
  - $\mathbf{p}_i^T\mathbf{w}$ is number between $-\infty$ to ∞.

Different Learning Models

- Different learning models $f(\mathbf{x}_i, \mathbf{w})$ reflect our beliefs about the relationship between the features $\mathbf{x}_i$ and target y_i
  - For example, $f(\mathbf{x}_i, \mathbf{w}) = \mathbf{p}_i^T \mathbf{w}$ assumes polynomial relationship between features and target

- Suppose we are performing classification (rather than regression), so y_i is class -1 or class 1
  - $\mathbf{p}_i^T \mathbf{w}$ is number between $-\infty$ to ∞.
  - Can use sigmoid function to map $\mathbf{p}_i^T \mathbf{w}$ to between 0 and 1:

$$f(\mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{p}_i^T \mathbf{w})$$
$$\sigma(a) = \frac{1}{1 + e^{-a}}$$



Different Learning Models

- Different learning models $f(\mathbf{x}_i, \mathbf{w})$ reflect our beliefs about the relationship between the features $\mathbf{x}_i$ and target y_i
  - For example, $f(\mathbf{x}_i, \mathbf{w}) = \mathbf{p}_i^T \mathbf{w}$ assumes polynomial relationship between features and target

- Suppose we are performing classification (rather than regression), so y_i is class -1 or class 1
  - $\mathbf{p}_i^T \mathbf{w}$ is number between $-\infty$ to ∞.
  - Can use sigmoid function to map $\mathbf{p}_i^T \mathbf{w}$ to between 0 and 1:

$$f(\mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{p}_i^T \mathbf{w})$$

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

  - If $f(\mathbf{x}_i, \mathbf{w})$ is closer to 0 (or 1), we predict class -1 (or class 1)

Different Learning Models

- Different learning models $f(\mathbf{x}_i, \mathbf{w})$ reflect our beliefs about the relationship between the features $\mathbf{x}_i$ and target y_i
  - For example, $f(\mathbf{x}_i, \mathbf{w}) = \mathbf{p}_i^T\mathbf{w}$ assumes polynomial relationship between features and target
- Suppose we are performing classification (rather than regression), so y_i is class -1 or class 1
  - $\mathbf{p}_i^T\mathbf{w}$ is number between $-\infty$ to ∞.
  - Can use sigmoid function to map $\mathbf{p}_i^T\mathbf{w}$ to between 0 and 1:

$$f(\mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{p}_i^T\mathbf{w})$$
$$\sigma(a) = \frac{1}{1+e^{-a}}$$

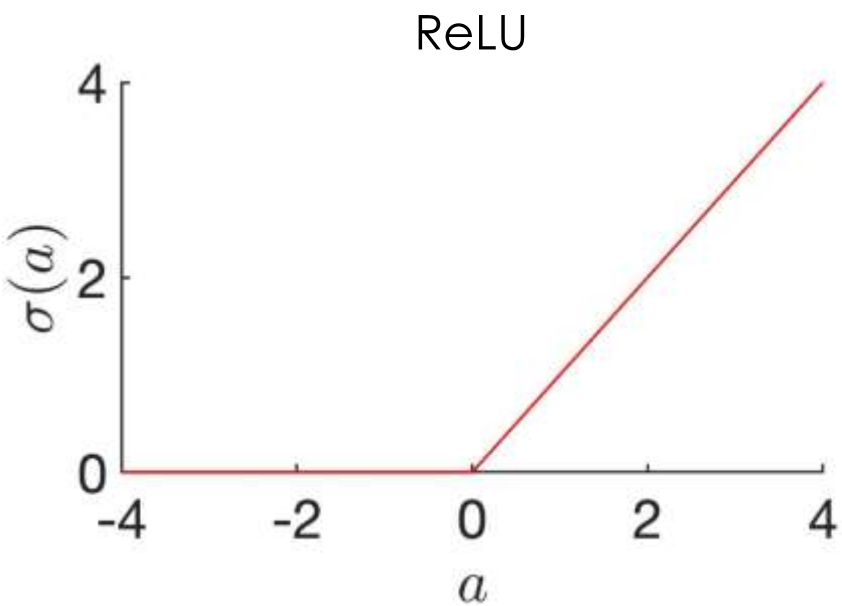
  - If $f(\mathbf{x}_i, \mathbf{w})$ is closer to 0 (or 1), we predict class -1 (or class 1)
- More generally, in one layer neural network: $f(\mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{p}_i^T\mathbf{w})$, where activation function σ can be sigmoid or some other functions & $\mathbf{p}$ is linear

Different Learning Models

- $f(\mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{p}_i^T \mathbf{w})$, where σ can be different functions:

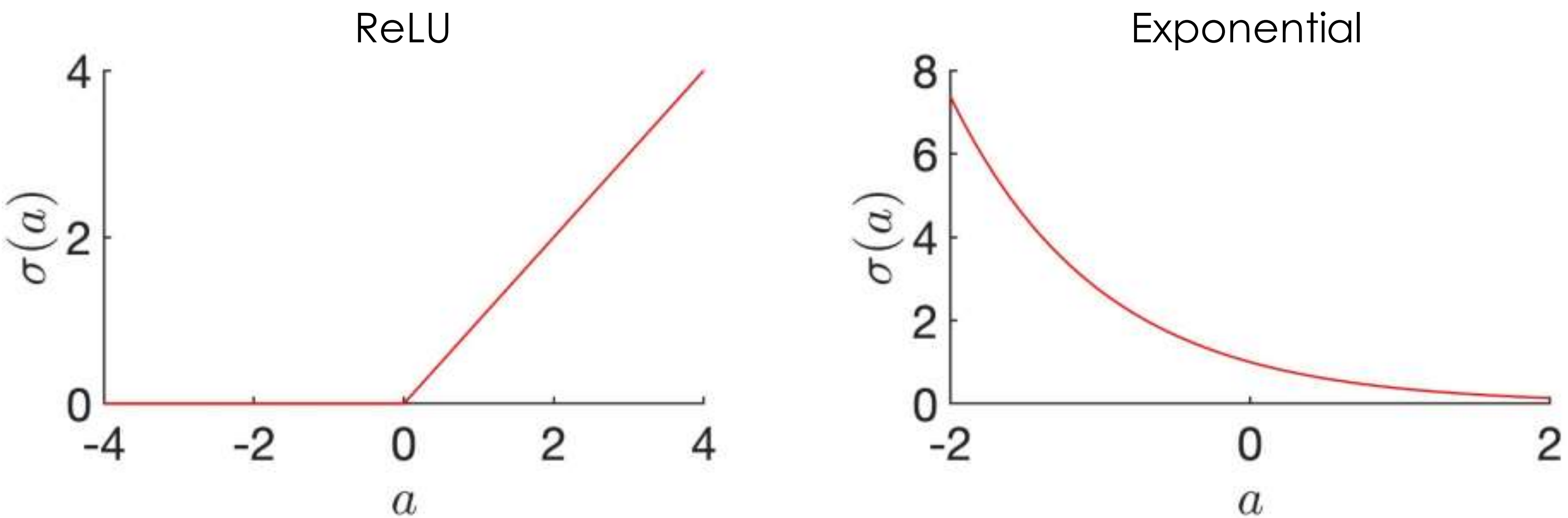
Different Learning Models

- $f(\mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{p}_i^T \mathbf{w})$, where σ can be different functions:
- Rectified linear unit (ReLU): $\sigma(a) = \max(0, a)$



Different Learning Models

- $f(\mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{p}_i^T \mathbf{w})$, where σ can be different functions:
- Rectified linear unit (ReLU): $\sigma(a) = \max(0, a)$
- Exponential: $\sigma(a) = \exp(-a)$

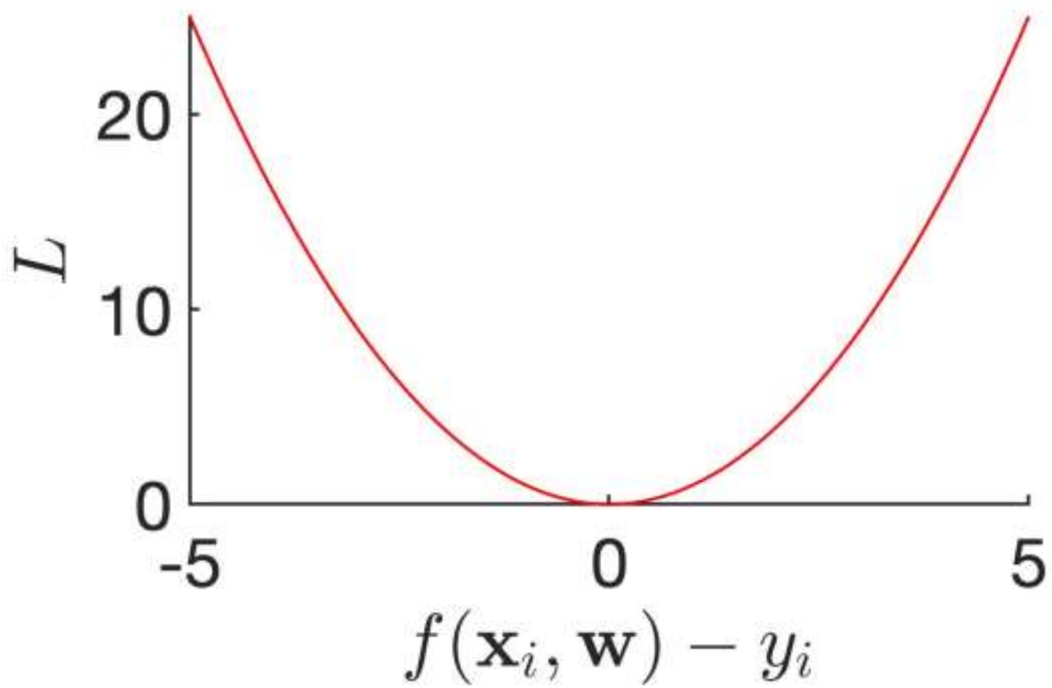


Different Loss Functions

- Different loss functions $L(f(\mathbf{x}_i, \mathbf{w}), y_i)$ encodes the penalty when we predict $f(\mathbf{x}_i, \mathbf{w})$ but the true value is y_i

Different Loss Functions

- Different loss functions $L(f(\mathbf{x}_i, \mathbf{w}), y_i)$ encodes the penalty when we predict $f(\mathbf{x}_i, \mathbf{w})$ but the true value is y_i
  - $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$ is called the square error loss



Different Loss Functions

- Different loss functions $L(f(\mathbf{x}_i, \mathbf{w}), y_i)$ encodes the penalty when we predict $f(\mathbf{x}_i, \mathbf{w})$ but the true value is y_i
  - $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$ is called the square error loss
- Suppose we are performing classification (rather than regression), so y_i is class -1 or class 1, then square error loss makes less sense. Instead, we can use

Different Loss Functions

- Different loss functions $L(f(\mathbf{x}_i, \mathbf{w}), y_i)$ encodes the penalty when we predict $f(\mathbf{x}_i, \mathbf{w})$ but the true value is y_i
  - $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$ is called the square error loss
- Suppose we are performing classification (rather than regression), so y_i is class -1 or class 1, then square error loss makes less sense. Instead, we can use
  - Binary loss (or 0–1 loss): $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = \begin{cases} 0 & \text{if } f(\mathbf{x}_i, \mathbf{w}) = y_i \\ 1 & \text{if } f(\mathbf{x}_i, \mathbf{w}) \neq y_i \end{cases}$

Different Loss Functions

- Different loss functions $L(f(\mathbf{x}_i, \mathbf{w}), y_i)$ encodes the penalty when we predict $f(\mathbf{x}_i, \mathbf{w})$ but the true value is y_i
  - $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$ is called the square error loss
- Suppose we are performing classification (rather than regression), so y_i is class -1 or class 1, then square error loss makes less sense. Instead, we can use
  - Binary loss (or 0–1 loss): $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = \begin{cases} 0 & \text{if } f(\mathbf{x}_i, \mathbf{w}) = y_i \\ 1 & \text{if } f(\mathbf{x}_i, \mathbf{w}) \neq y_i \end{cases}$
  - In practice, hard to constrain $f(\mathbf{x}_i, \mathbf{w})$ to be exactly -1 or 1, so we can declare "victory" if $f(\mathbf{x}_i, \mathbf{w})$ & y have the same sign:

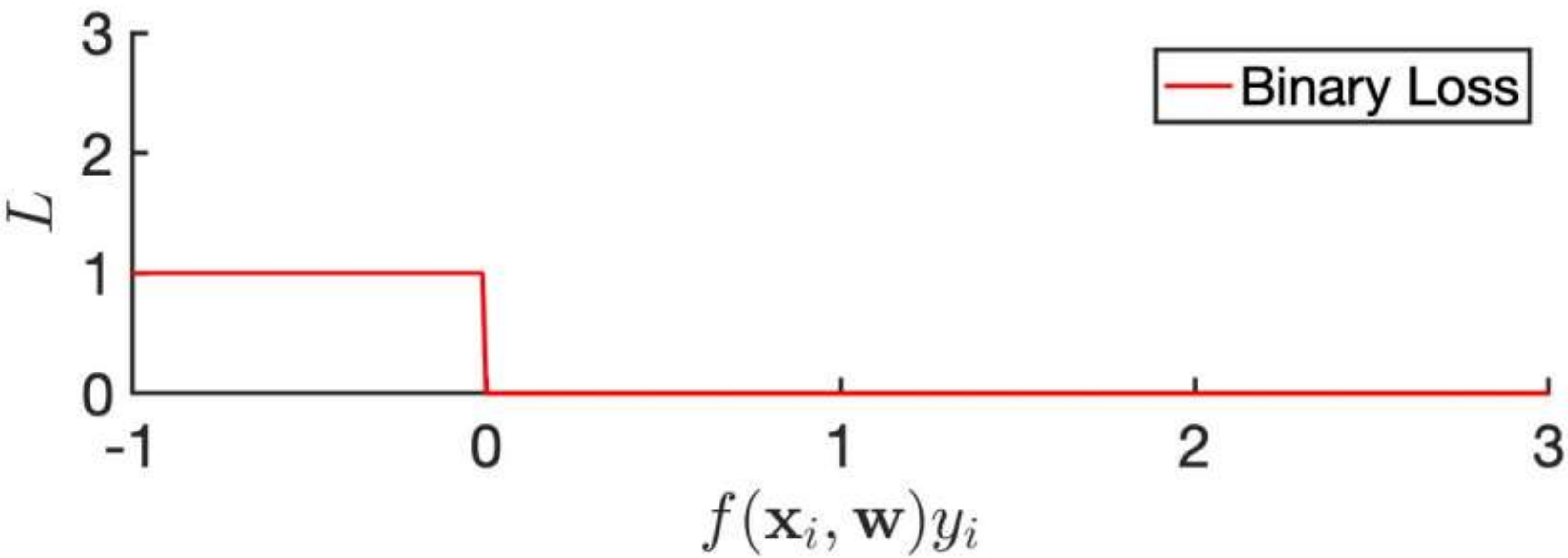
Different Loss Functions

- Different loss functions $L(f(\mathbf{x}_i, \mathbf{w}), y_i)$ encodes the penalty when we predict $f(\mathbf{x}_i, \mathbf{w})$ but the true value is y_i
  - $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$ is called the square error loss
- Suppose we are performing classification (rather than regression), so y_i is class -1 or class 1, then square error loss makes less sense. Instead, we can use
  - Binary loss (or 0–1 loss): $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = \begin{cases} 0 & \text{if } f(\mathbf{x}_i, \mathbf{w}) = y_i \\ 1 & \text{if } f(\mathbf{x}_i, \mathbf{w}) \neq y_i \end{cases}$
  - In practice, hard to constrain $f(\mathbf{x}_i, \mathbf{w})$ to be exactly -1 or 1, so we can declare "victory" if $f(\mathbf{x}_i, \mathbf{w})$ & y have the same sign:

$$L(f(\mathbf{x}_i, \mathbf{w}), y_i) = \begin{cases} 0 & \text{if } f(\mathbf{x}_i, \mathbf{w}) y_i > 0 \\ 1 & \text{if } f(\mathbf{x}_i, \mathbf{w}) y_i < 0 \end{cases}$$

Different Loss Functions

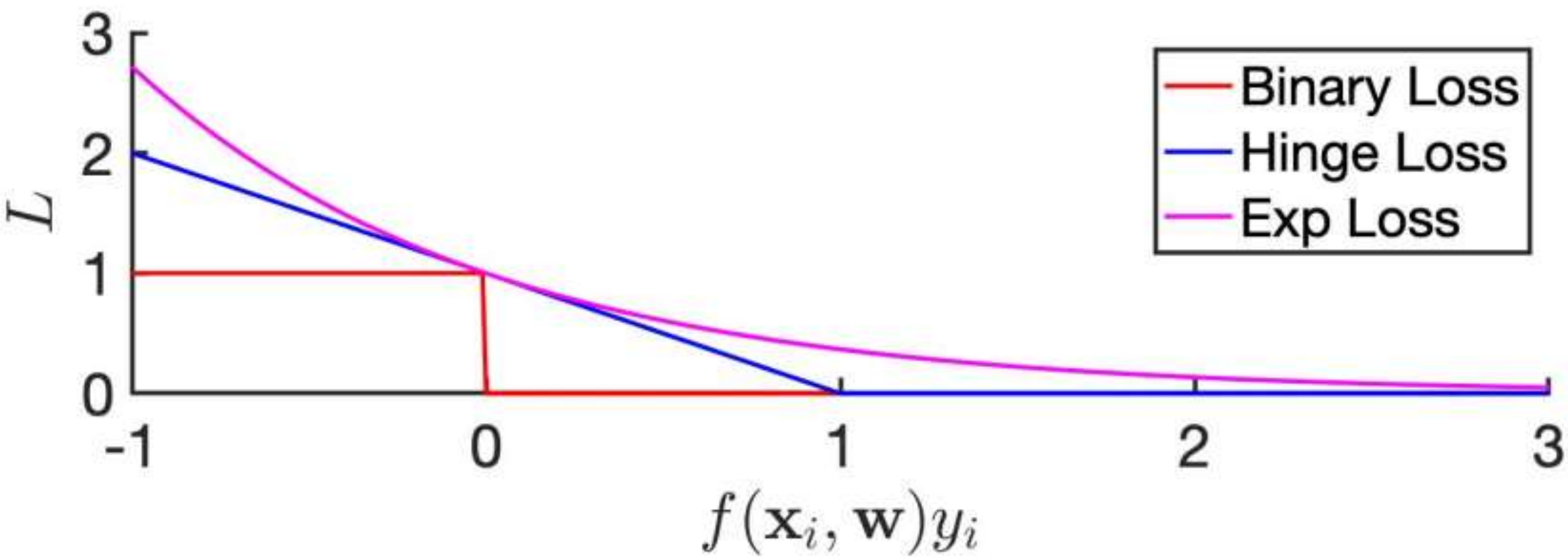
- Binary loss, where y_i is class -1 or class 1 & $f(\mathbf{x}_i, \mathbf{w})$ is a number between $-\infty$ and ∞: $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = \begin{cases} 0 & \text{if } f(\mathbf{x}_i, \mathbf{w})y_i > 0 \\ 1 & \text{if } f(\mathbf{x}_i, \mathbf{w})y_i < 0 \end{cases}$



Different Loss Functions

- Binary loss, where y_i is class -1 or class 1 & $f(\mathbf{x}_i, \mathbf{w})$ is a number between $-\infty$ and ∞: $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = \begin{cases} 0 & \text{if } f(\mathbf{x}_i, \mathbf{w})y_i > 0 \\ 1 & \text{if } f(\mathbf{x}_i, \mathbf{w})y_i < 0 \end{cases}$

- Binary loss not differentiable, so two other possibilities
  - Hinge loss: $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = \max(0, 1 - f(\mathbf{x}_i, \mathbf{w})y_i)$
  - Exponential loss: $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = \exp(-f(\mathbf{x}_i, \mathbf{w})y_i)$



Questions?

Summary

- Building blocks of machine learning algorithms
  - Learning model: reflects our belief about relationship between features & target we want to predict
  - Loss function: penalty for wrong prediction
  - Regularization: penalizes complex models
  - Optimization routine: find minimum of overall cost function

Summary

- Building blocks of machine learning algorithms
  - Learning model: reflects our belief about relationship between features & target we want to predict
  - Loss function: penalty for wrong prediction
  - Regularization: penalizes complex models
  - Optimization routine: find minimum of overall cost function
- Gradient descent algorithm
  - At each iteration, compute gradient & update model parameters in direction opposite to gradient
  - If learning rate η is too big => may not converge
  - If learning rate η is too small => converge very slowly

Summary

- Building blocks of machine learning algorithms
  - Learning model: reflects our belief about relationship between features & target we want to predict
  - Loss function: penalty for wrong prediction
  - Regularization: penalizes complex models
  - Optimization routine: find minimum of overall cost function
- Gradient descent algorithm
  - At each iteration, compute gradient & update model parameters in direction opposite to gradient
  - If learning rate η is too big => may not converge
  - If learning rate η is too small => converge very slowly
- Different learning models, e.g., linear, polynomial, sigmoid, ReLU, exponential, etc

Summary

- Building blocks of machine learning algorithms
  - Learning model: reflects our belief about relationship between features & target we want to predict
  - Loss function: penalty for wrong prediction
  - Regularization: penalizes complex models
  - Optimization routine: find minimum of overall cost function
- Gradient descent algorithm
  - At each iteration, compute gradient & update model parameters in direction opposite to gradient
  - If learning rate η is too big => may not converge
  - If learning rate η is too small => converge very slowly
- Different learning models, e.g., linear, polynomial, sigmoid, ReLU, exponential, etc
- Different loss functions, e.g., square error, binary, logistic, etc