CS536

RUTGERS

Linear Models

Sungjin Ahn

Department of Computer Science
Rutgers University

Linear Models for Regression

Linear Regression

- Assume that each input x is a D-dimensional feature vector.
  - In linear model, we introduce a weight for each feature and a bias term.

$$\hat{y} = w_1 \cdot x_1 + ... + w_d \cdot x_d + b.$$

- Vector form:

$$\hat{y} = \mathbf{w}^T \mathbf{x} + b.$$

- If we append 1 at the end of **x,** and concatenate **w** & **b,** we can write the model as a dot product between the two vectors

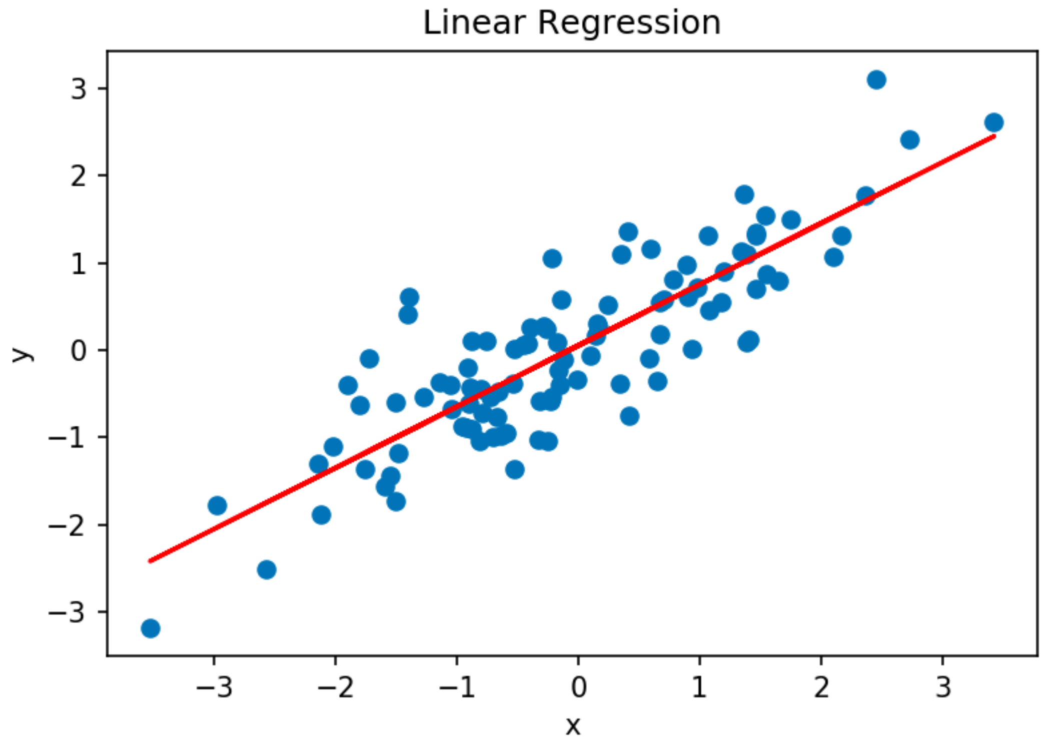
$$\hat{y} = \boldsymbol{w}^\top \boldsymbol{x}$$

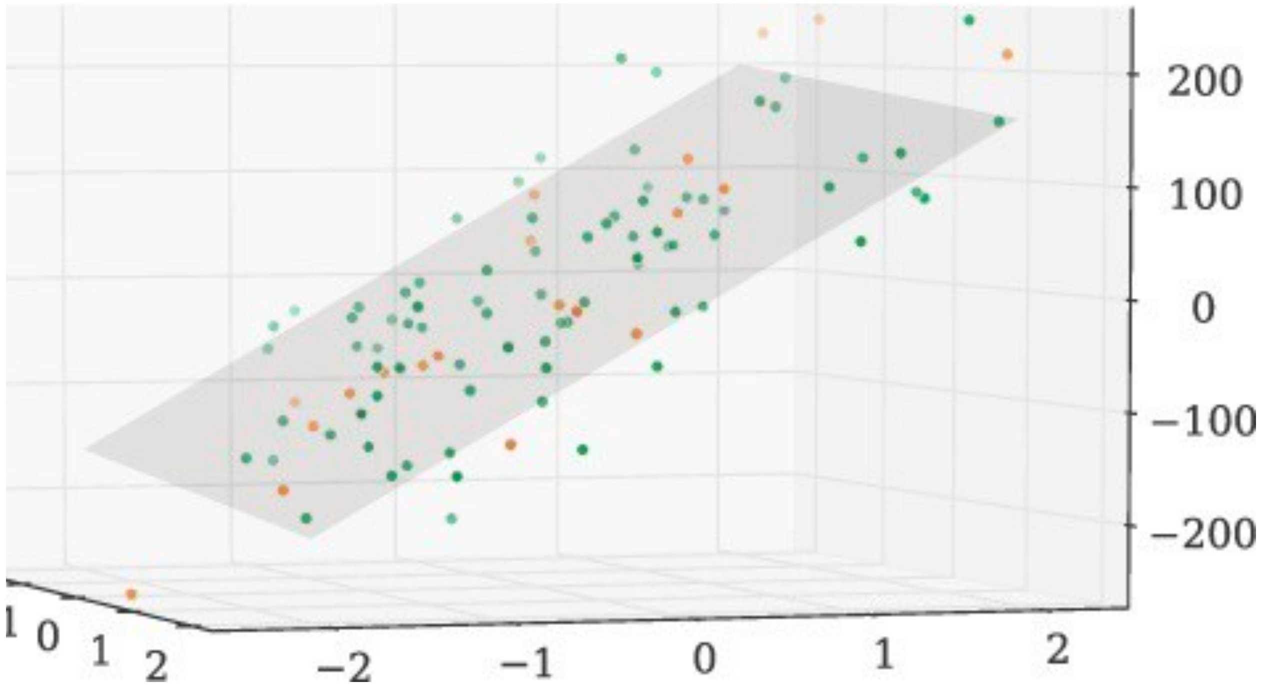
The Matrix Form of Linear Regression

- **X**: N x D matrix (N: the number of training data), this is called the ***design matrix***
- **w**: D x 1 vector
- **b**: N x 1 vector
- $\hat{\boldsymbol{y}}$: N x 1 vector

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} \beta_0 + \beta_1 x_1 \\ \beta_0 + \beta_1 x_2 \\ \vdots \\ \beta_0 + \beta_1 x_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \mathbf{X\beta}$$

Here we used $\boldsymbol{\beta}$ instead of $\boldsymbol{w}$ to represent the model parameter


Linear Regression
y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


200
100
0
−100
−200
−2
−1
0
1
2
1
0
1
2

Basis Functions

- In the polynomial regression example, we assumed that the input x is 1D.
- But, we increased the input dimension to M by applying x, x^2, x^3, …, x^M
- We can apply a function ϕ(x) that transforms the input into some other form if we believe that the transform will help learning.
  - This is the **feature design** or **feature engineering** process
- Such functions are called ***basis functions*** in general
- Examples,
  - Polynomial: $x^j = \phi_j(\mathrm{x})$
  - Gaussian: $\phi_j(x) = exp\left(-\frac{(x-\mu_j)^2}{2s^2}\right)$
  - Sigmoid: $\phi_j(x) = \sigma\left(\frac{x-\mu_j}{s}\right)$

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \underset{\substack{\uparrow \\ \text{Basis function}}}{\phi_j(\mathbf{x})}$$

Polynomial Regression is a Linear Model

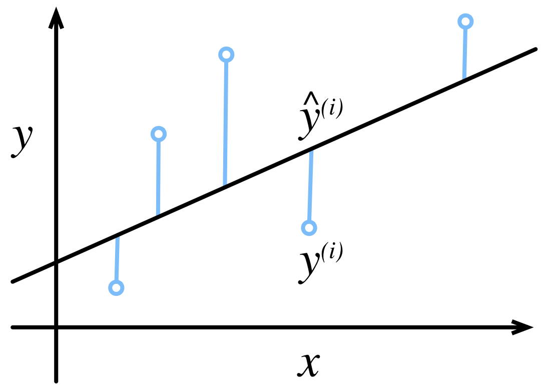
- β_d is the model parameter. Then, the matrix form of polynomial regression is

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^m \\ 1 & x_2 & x_2^2 & \dots & x_2^m \\ 1 & x_3 & x_3^2 & \dots & x_3^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^m \end{bmatrix}}_{\mathbf{X}} \underbrace{\begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{bmatrix}}_{\mathbf{w}}$$

(Mean) Squared Loss

- Squared loss error of each data item. This measures the distance between the prediction and label (=target)

$$l^{(i)}(\mathbf{w}, b) = \frac{1}{2} \left(\hat{y}^{(i)} - y^{(i)}\right)^2.$$



- Mean Squared Loss (MSE) for the loss of the whole training data

$$L(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^n l^{(i)}(\mathbf{w}, b) = \frac{1}{n} \sum_{i=1}^n \frac{1}{2}\left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}\right)^2.$$

- The goal is to find the model parameters that minimize this loss

$$\mathbf{w}^*, b^* = \operatorname*{argmin}_{\mathbf{w}, b}\ L(\mathbf{w}, b).$$

Two ways to solve (continuous) optimization problems

- Analytical methods
  - Find a **closed-form** solution
    - Express the solution as a mathematical expression consisting of some certain "well-known" and finite computable functions
  - Available only for some simple (mostly linear) problems
- Numerical methods
  - Solving complex mathematical problems using only simple arithmetic operations $\rightarrow$ find approximate solution
  - More generally applicable
  - e.g., gradient descent to find a local minimum
- Linear regression can be solved in both ways

Analytic Solution for Linear Regression

- In the matrix form, the objective can be written as:

$$||\mathbf{y} - X\mathbf{w}||$$

  - Which is a quadratic form and convex (no local minima)
  - **X:** N x D matrix (called ***design matrix***)

- Taking derivative of the objective w.r.t. **w** and setting it equal to 0 yields the analytic solution

$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T y.$$

- This equation is known as the ***normal equation*** , can you drive this?

Numerical Solution for Linear Regression

- We can use the **Gradient Descent (GD) method**

Gradient Descent Methods

Gradient Descent (GD)

- Complex models like deep neural networks are nonlinear models for which an analytic solution is not available.

- We can still optimize the loss if the loss function is differentiable w.r.t. the parameter **w** by using the gradient descent method.

- Why do we consider numerical method if closed-form solution exists.
    - For some problems (e.g., if the dataset is very large), using numerical optimization like stochastic gradient descent could be more efficient than solving the closed-form solution.

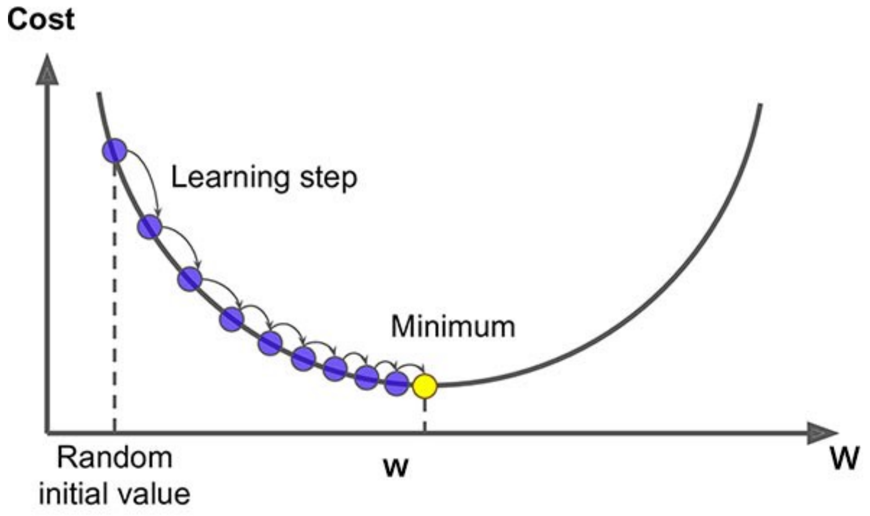
The Gradient Descent (GD) Method

- Iterate: compute the gradient dL/d**w** and update the parameter as follows

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{\partial L}{\partial \boldsymbol{w}}$$

- Until some **stopping criteria** is met. e.g.,
  - The loss doesn't decrease much
  - The norm of the gradient is near zero

- Learning rate η is a hyperparameter controlling the step size

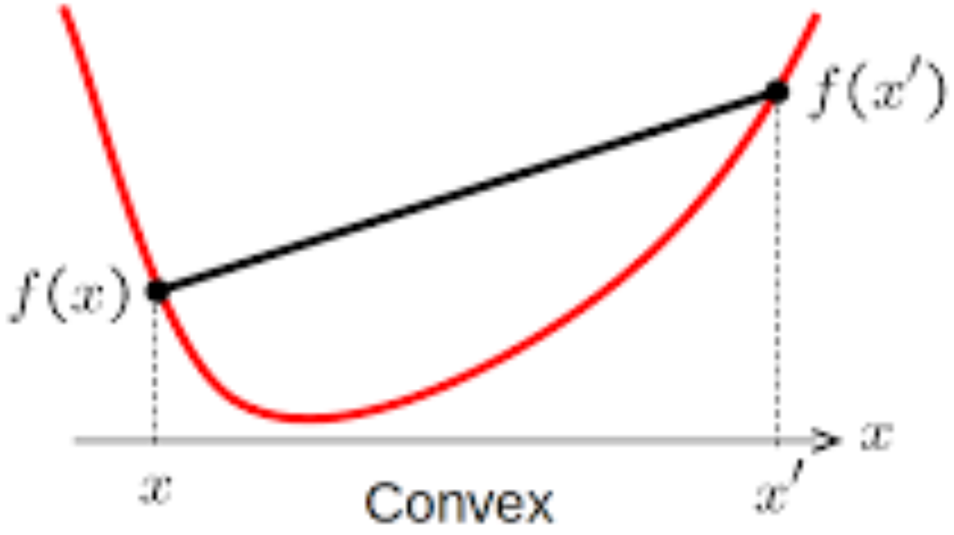


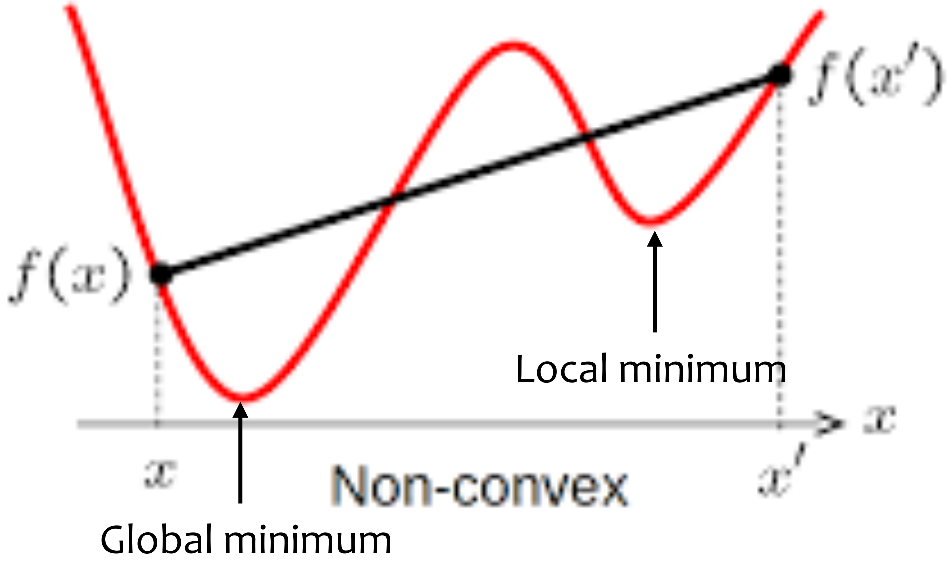
Convex & Non-Convex Function

- A function f: X → R is convex if for any t in [0,1] and all x_1, x_2 in X,

$$f(tx_1 + (1 - t)\,x_2) \leq tf(x_1) + (1 - t)f(x_2)$$

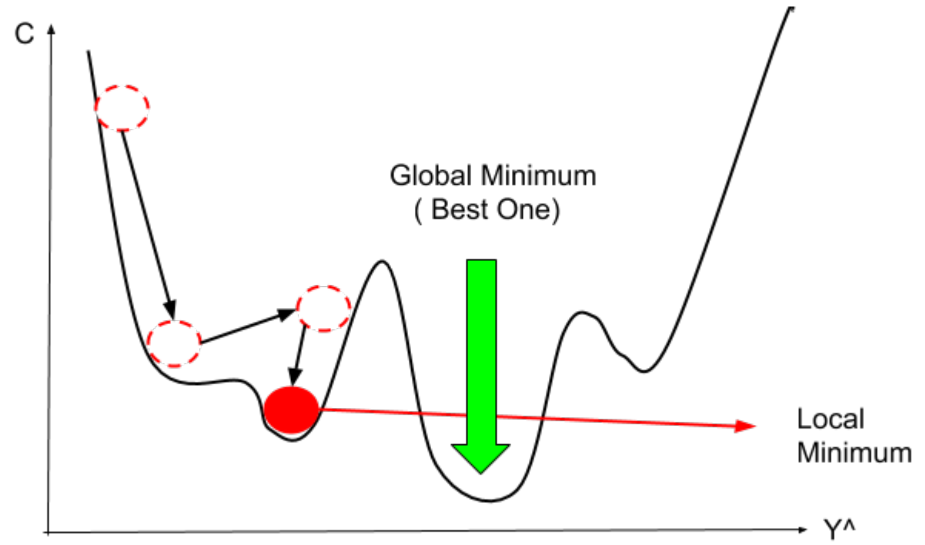
- If it is convex, the function has one **global minimum.** If it is non-convex, the function has many **local minima.**





Gradient Descent and Convexity

- GD updates see only the gradient of the current position (state, weight, or parameter)
- That is, beyond current position, it doesn't know how the loss surface look like. So, it can never know where the global optimum is.
- Even if it reaches the global minimum, there is no way it can know that it is actually the global minimum (unless we evaluate all possible points in the space)



Stochastic Gradient Descent

GD can be inefficient

- In machine learning, the loss function is usually a sum (or mean) of individual losses. Thus, the gradient of the loss is also the sum (or mean) of individual gradients
  - See the MSE loss
- So, computing the gradient ONCE is O(N)
- But, in ML, often N is very large. And, until the convergence, we may need to take the gradient update steps for millions of times

What is Unbiased Estimator

- We say that an estimator is **unbiased** if the expectation of the estimator is the same as the target value that the estimator wants to estimate

- How can you estimate the average height of this class?
  - To do this, you set up the following process (=your estimator)
    - You ask to 10 random persons and get an estimation by averaging the answers. Then, you pick another 10 persons and get another estimation. If you repeat this infinitely many times and if ***the average of this infinitely many estimations*** converges to the true value, then your way of estimating <u>the average height of the class (i.e., the target)</u> is unbiased.
  - Do you think the above process is actually unbiased?
  - What if you average top 25% and bottom 25%

What is Unbiased Estimator

- For example, given a dataset D of size N, the true mean value is

$$\mu^* = \frac{1}{N}\sum_{i=1}^{N} x_i$$

- If we make an estimator f to estimate μ^* by uniform-random sampling $\mathcal{B}$ from D with |B| << N, we have

$$f = \frac{1}{|\mathcal{B}|}\sum_{j \in I(B)}^{|\mathcal{B}|} x_j$$

- Then, f is unbiased estimator of the true mean if

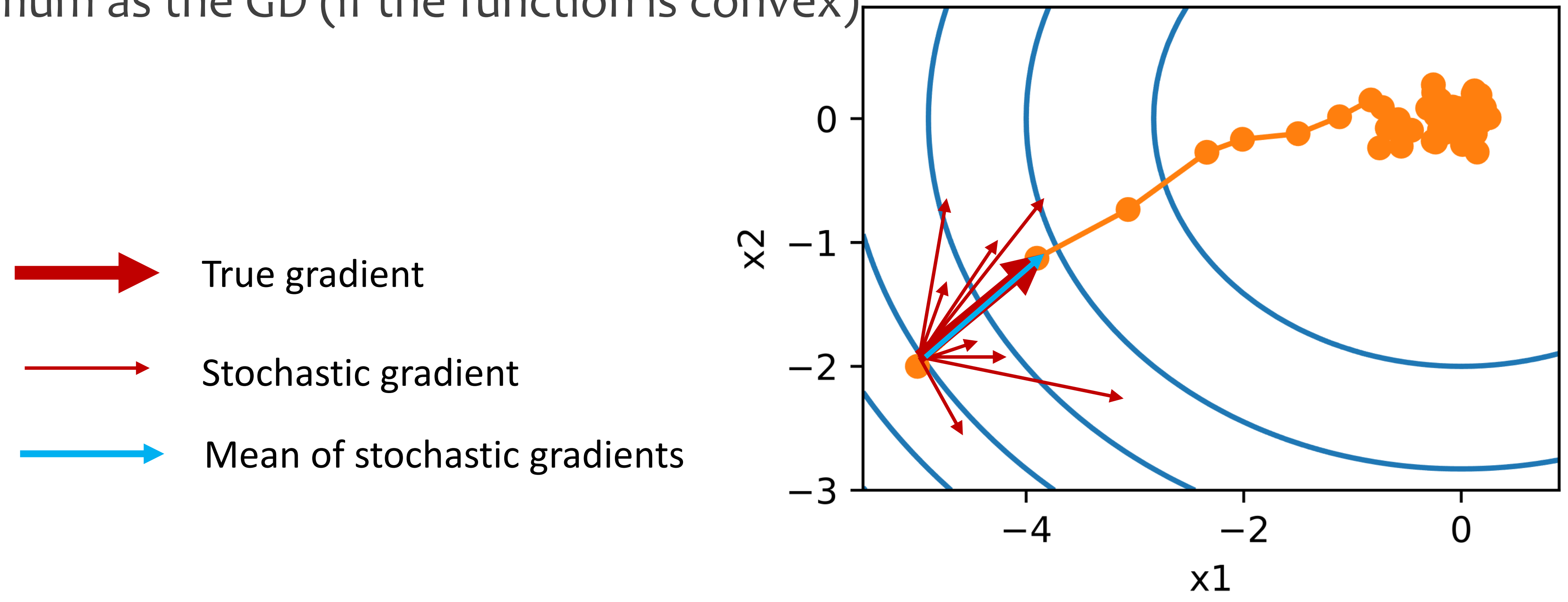
$$\mathbb{E}_{p(\mathcal{B})}[f] = \mu^*$$

Stochastic Gradient Descent

- While true
  - Pick a small random (i.i.d.) minibatch B from the full training set
  - Compute the loss of the mini-batch (=mini-loss)
  - Compute the gradient of the mini-loss
  - Update parameter with the gradient
  - If your stopping criteria is met
    - Break

Unbiasedness of Stochastic Gradient

- If we i.i.d.-sample a mini-batch *B* datapoints from the original dataset D, the (stochastic) gradient computed from this mini-batch is an unbiased estimator of the true gradient computed from the whole dataset.
- This means that in expectation, SGD will converge to the same optimum as the GD (if the function is convex)



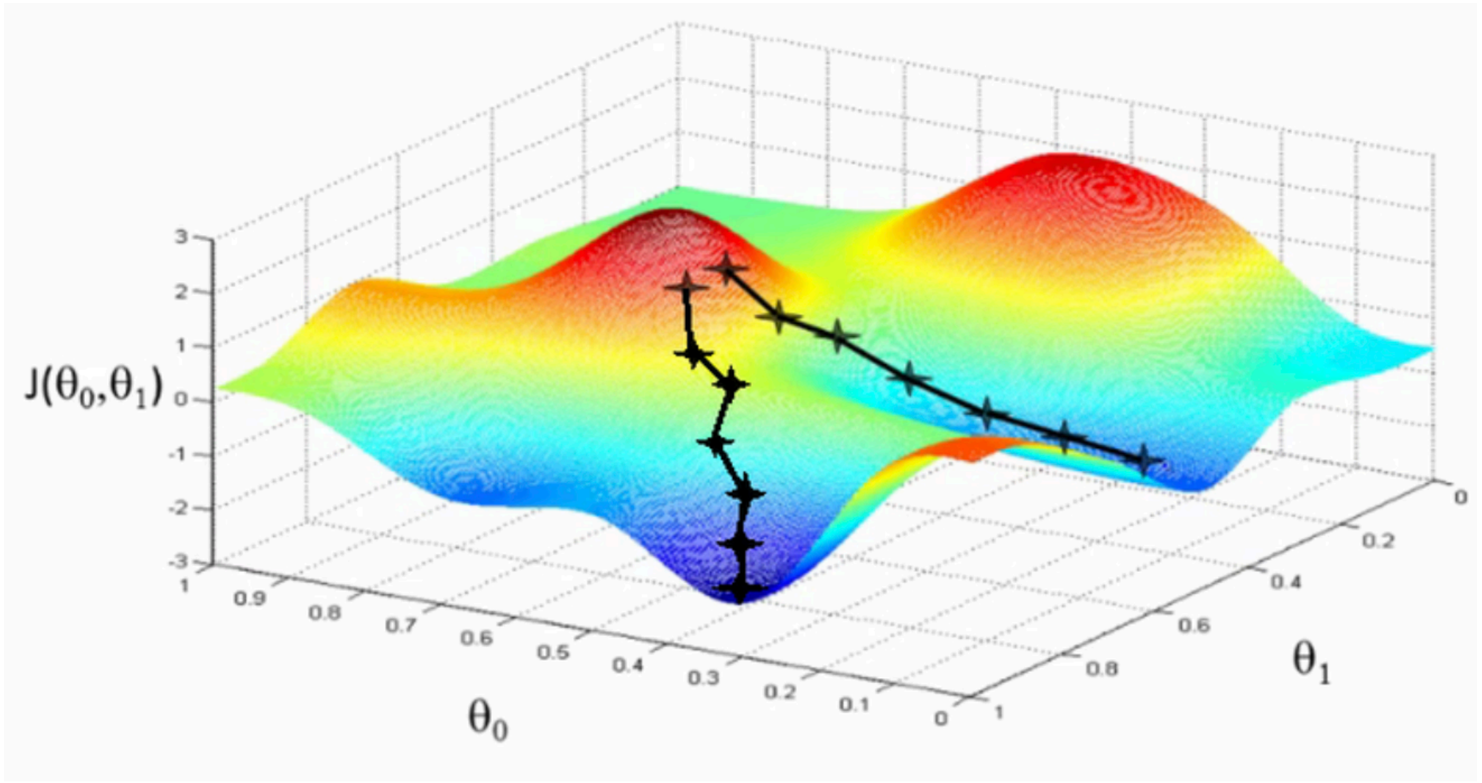
Benefits of SGD

- SGD is very efficient
  - If the dataset is very large, say 10M datapoints, but also contains many similar data, then using 500 datapoints to compute the gradient would give a similar direction to the true gradient. But it is much more efficient.

- SGD provides good generalization performance
  - The noisy gradient is a good regularizer preventing overfitting.
  - Thus, SGD is still a very popular choice for training neural networks.

Stochastic Gradient Descent

- can reach different solutions due to the random batch pickup



SGD for MSE of Linear Regression

- MSE Loss

$$L(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} l^{(i)}(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} \boxed{\frac{1}{2}\left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}\right)^2.}$$

- SGD Update Rule for a minibatch $\mathcal{B}$

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \frac{\eta}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \partial_{\mathbf{w}} l^{(i)}(\mathbf{w}, b) && = w - \frac{\eta}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \mathbf{x}^{(i)}\left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}\right),\\ b &\leftarrow b - \frac{\eta}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \partial_b l^{(i)}(\mathbf{w}, b) && = b - \frac{\eta}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}\right). \end{aligned}$$

Prediction

- After obtaining the optimized parameter (w*, b*) by either closed-form solution or gradient descent
- We save the last model (w*, b*) and use it for test prediction

- $f(x; w^*, b^*) = w^* x + b^*$