Learning as Optimization: Linear Regression

Piyush Rai

Machine Learning (CS771A)

Aug 10, 2016

Learning as Optimization

- Consider a supervised learning problem with training data $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$
- Goal: Find a function f that best approximates the $\boldsymbol{x} \rightarrow y$ relationship

Learning as Optimization

- Consider a supervised learning problem with training data $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$
- Goal: Find a function f that best approximates the $\boldsymbol{x} \to y$ relationship
- Define a "loss function" $\ell(y, f(\boldsymbol{x}))$: Error of f on an example $(\boldsymbol{x}, y)$

Learning as Optimization

- Consider a supervised learning problem with training data $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^{N}$
- Goal: Find a function f that best approximates the $\boldsymbol{x} \to y$ relationship
- Define a "loss function" $\ell(y, f(\boldsymbol{x}))$: Error of f on an example $(\boldsymbol{x}, y)$
- Note: Choice of $\ell()$ and $f()$ will be problem specific, e.g.,
  - Least squares regression: $\ell()$ is squared loss, $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$

Learning as Optimization

- Consider a supervised learning problem with training data $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$
- Goal: Find a function f that best approximates the $\boldsymbol{x} \rightarrow y$ relationship
- Define a "loss function" $\ell(y, f(\boldsymbol{x}))$: Error of f on an example $(\boldsymbol{x}, y)$
- Note: Choice of $\ell()$ and $f()$ will be problem specific, e.g.,
  - Least squares regression: $\ell()$ is squared loss, $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$
- We would like to find f that minimizes the true loss or "risk" defined as

$$L(f) = \mathbb{E}_{(\mathbf{x},y)\sim P}[\ell(y, f(\boldsymbol{x}))] = \int \ell(y, f(\boldsymbol{x})dP(\boldsymbol{x}, y)$$

Learning as Optimization

- Consider a supervised learning problem with training data $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$
- Goal: Find a function f that best approximates the $\boldsymbol{x} \rightarrow y$ relationship
- Define a "loss function" $\ell(y, f(\boldsymbol{x}))$: Error of f on an example $(\boldsymbol{x}, y)$
- Note: Choice of $\ell()$ and $f()$ will be problem specific, e.g.,
  - Least squares regression: $\ell()$ is squared loss, $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$
- We would like to find f that minimizes the true loss or "risk" defined as

$$L(f) = \mathbb{E}_{(\mathbf{x},y)\sim P}[\ell(y, f(\boldsymbol{x})] = \int \ell(y, f(\boldsymbol{x})dP(\boldsymbol{x}, y)$$

- Problem: We (usually) don't know the true distribution and only have finite set of samples from it, in form of the N training examples $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$

Learning as Optimization

- Consider a supervised learning problem with training data $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$
- Goal: Find a function f that best approximates the $\boldsymbol{x} \to y$ relationship
- Define a "loss function" $\ell(y, f(\boldsymbol{x}))$: Error of f on an example $(\boldsymbol{x}, y)$
- Note: Choice of $\ell()$ and $f()$ will be problem specific, e.g.,
  - Least squares regression: $\ell()$ is squared loss, $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$
- We would like to find f that minimizes the true loss or "risk" defined as

$$L(f) = \mathbb{E}_{(\mathbf{x},y)\sim P}[\ell(y, f(\boldsymbol{x}))] = \int \ell(y, f(\boldsymbol{x})dP(\boldsymbol{x}, y)$$

- Problem: We (usually) don't know the true distribution and only have finite set of samples from it, in form of the N training examples $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$
- Solution: Work with the "empirical" risk defined on the training data

$$\boxed{L_{emp}(f) = \frac{1}{N}\sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n))}$$

Learning as Optimization

- To find the best f, we minimize the empirical risk w.r.t. f. Empirical Risk Minimization (ERM)

$$\hat{f} = \arg\min_{f} L_{emp}(f) = \arg\min_{f} \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n))$$

Learning as Optimization

- To find the best f, we minimize the empirical risk w.r.t. f. Empirical Risk Minimization (ERM)

$$\hat{f} = \arg\min_{f} L_{emp}(f) = \arg\min_{f} \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n))$$

Learning as Optimization

- To find the best f, we minimize the empirical risk w.r.t. f. Empirical Risk Minimization (ERM)

$$\hat{f} = \arg\min_f L_{emp}(f) = \arg\min_f \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n))$$

- We also want f to be "simple". To do so, we add a "regularizer" $R(f)$

$$\boxed{\hat{f} = \arg\min_f \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n)) + \lambda R(f)}$$

- The regularizer $R(f)$ is a measure of complexity of our model f

Learning as Optimization

- To find the best f, we minimize the empirical risk w.r.t. f. Empirical Risk Minimization (ERM)

$$\hat{f} = \arg\min_f L_{emp}(f) = \arg\min_f \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n))$$

- We also want f to be "simple". To do so, we add a "regularizer" $R(f)$

$$\boxed{\hat{f} = \arg\min_f \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n)) + \lambda R(f)}$$

- The regularizer $R(f)$ is a measure of complexity of our model f
- This is called **Regularized** (Empirical) Risk Minimization

Learning as Optimization

- To find the best f, we minimize the empirical risk w.r.t. f. Empirical Risk Minimization (ERM)

$$\hat{f} = \arg\min_f L_{emp}(f) = \arg\min_f \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n))$$

- We also want f to be "simple". To do so, we add a "regularizer" $R(f)$

$$\boxed{\hat{f} = \arg\min_f \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n)) + \lambda R(f)}$$

- The regularizer $R(f)$ is a measure of complexity of our model f
- This is called **Regularized** (Empirical) Risk Minimization
- We want both $L_{emp}(f)$ and $R(f)$ to be small

Learning as Optimization

- To find the best f, we minimize the empirical risk w.r.t. f. Empirical Risk Minimization (ERM)

$$\hat{f} = \arg\min_f L_{emp}(f) = \arg\min_f \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n))$$

- We also want f to be "simple". To do so, we add a "regularizer" $R(f)$

$$\boxed{\hat{f} = \arg\min_f \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n)) + \lambda R(f)}$$

- The regularizer $R(f)$ is a measure of complexity of our model f
- This is called **Regularized** (Empirical) Risk Minimization
- We want both $L_{emp}(f)$ and $R(f)$ to be small
  - Small empirical error on training data and simple model

Learning as Optimization

- To find the best f, we minimize the empirical risk w.r.t. f. Empirical Risk Minimization (ERM)

$$\hat{f} = \arg\min_f L_{emp}(f) = \arg\min_f \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n))$$

- We also want f to be "simple". To do so, we add a "regularizer" $R(f)$

$$\boxed{\hat{f} = \arg\min_f \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n)) + \lambda R(f)}$$

- The regularizer $R(f)$ is a measure of complexity of our model f
- This is called **Regularized** (Empirical) Risk Minimization
- We want both $L_{emp}(f)$ and $R(f)$ to be small
  - Small empirical error on training data and simple model
  - There is usually a trade-off between these two goals

Learning as Optimization

- To find the best f, we minimize the empirical risk w.r.t. f. Empirical Risk Minimization (ERM)

$$\hat{f} = \arg\min_f L_{emp}(f) = \arg\min_f \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n))$$

- We also want f to be "simple". To do so, we add a "regularizer" $R(f)$

$$\boxed{\hat{f} = \arg\min_f \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n)) + \lambda R(f)}$$

- The regularizer $R(f)$ is a measure of complexity of our model f
- This is called **Regularized** (Empirical) Risk Minimization
- We want both $L_{emp}(f)$ and $R(f)$ to be small
  - Small empirical error on training data and simple model
  - There is usually a trade-off between these two goals
  - The regularization hyperparameter λ can help us control this trade-off

Learning as Optimization

- To find the best f, we minimize the empirical risk w.r.t. f. Empirical Risk Minimization (ERM)

$$\hat{f} = \arg\min_f L_{emp}(f) = \arg\min_f \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n))$$

- We also want f to be "simple". To do so, we add a "regularizer" $R(f)$

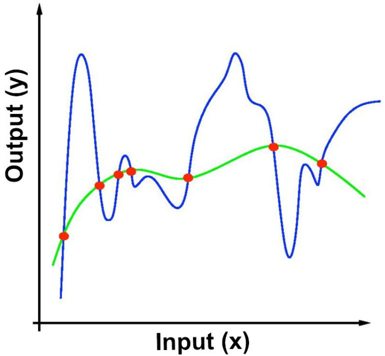
$$\boxed{\hat{f} = \arg\min_f \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n)) + \lambda R(f)}$$

- The regularizer $R(f)$ is a measure of complexity of our model f
- This is called **Regularized** (Empirical) Risk Minimization
- We want both $L_{emp}(f)$ and $R(f)$ to be small
    - Small empirical error on training data and simple model
    - There is usually a trade-off between these two goals
    - The regularization hyperparameter λ can help us control this trade-off
- Various choices for the regularizer $R(f)$; more on this later

Regularization: Pictorially

Both curves have the same (zero) empirical error L_{emp}

Green curve has a smaller $R(f)$ (thus smaller complexity). We'll look at forms of $R(f)$ later



Learning as Optimization

- Our goal is to solve the optimization problem

$$\boxed{\hat{f} = \arg\min_{f} \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n)) + \lambda R(f) = \arg\min_{f} L_{reg}(f)}$$

Learning as Optimization

- Our goal is to solve the optimization problem

$$\boxed{\hat{f} = \arg\min_f \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n)) + \lambda R(f) = \arg\min_f L_{reg}(f)}$$

- The function being optimized is $L_{reg}(f)$, our regularized loss function

Learning as Optimization

- Our goal is to solve the optimization problem

$$\boxed{\hat{f} = \arg\min_{f} \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n)) + \lambda R(f) = \arg\min_{f} L_{reg}(f)}$$

- The function being optimized is $L_{reg}(f)$, our regularized loss function
- Different learning problems basically differ in terms of choices of f, ℓ, and R

Learning as Optimization

- Our goal is to solve the optimization problem

$$\boxed{\hat{f} = \arg\min_f \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n)) + \lambda R(f) = \arg\min_f L_{reg}(f)}$$

- The function being optimized is $L_{reg}(f)$, our regularized loss function
- Different learning problems basically differ in terms of choices of f, ℓ, and R
- Given a specific choice of f, ℓ, and R, some questions to consider

Learning as Optimization

- Our goal is to solve the optimization problem

$$\boxed{\hat{f} = \arg\min_f \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n)) + \lambda R(f) = \arg\min_f L_{reg}(f)}$$

- The function being optimized is $L_{reg}(f)$, our regularized loss function
- Different learning problems basically differ in terms of choices of f, ℓ, and R
- Given a specific choice of f, ℓ, and R, some questions to consider
  - Which method to use to solve the optimization problem?

Learning as Optimization

- Our goal is to solve the optimization problem

$$\boxed{\hat{f} = \arg\min_f \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n)) + \lambda R(f) = \arg\min_f L_{reg}(f)}$$

- The function being optimized is $L_{reg}(f)$, our regularized loss function
- Different learning problems basically differ in terms of choices of f, ℓ, and R
- Given a specific choice of f, ℓ, and R, some questions to consider
  - Which method to use to solve the optimization problem?
  - How do we solve it efficiently?

Learning as Optimization

- Our goal is to solve the optimization problem

$$\boxed{\hat{f} = \arg\min_f \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n)) + \lambda R(f) = \arg\min_f L_{reg}(f)}$$

- The function being optimized is $L_{reg}(f)$, our regularized loss function
- Different learning problems basically differ in terms of choices of f, ℓ, and R
- Given a specific choice of f, ℓ, and R, some questions to consider
  - Which method to use to solve the optimization problem?
  - How do we solve it efficiently?
  - Will there be (and can we find) a unique solution for f ?

Learning as Optimization

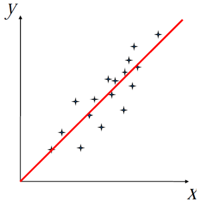
- Our goal is to solve the optimization problem

$$\boxed{\hat{f} = \arg\min_{f} \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n)) + \lambda R(f) = \arg\min_{f} L_{reg}(f)}$$

- The function being optimized is $L_{reg}(f)$, our regularized loss function
- Different learning problems basically differ in terms of choices of f, ℓ, and R
- Given a specific choice of f, ℓ, and R, some questions to consider
  - Which method to use to solve the optimization problem?
  - How do we solve it efficiently?
  - Will there be (and can we find) a unique solution for f ?
- We will revisit these questions later. First let's look at an example problem

Linear Regression

Fitting a Line to the Data

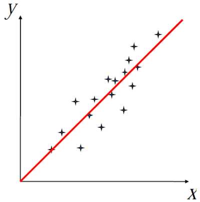


- Let's assume the relationship between x and y to have a linear model

$$y = wx$$

- Problem boils down to fitting a line to the data
- w is the model parameter (slope of the line here)

Fitting a Line to the Data



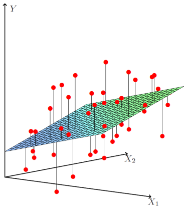
- Let's assume the relationship between x and y to have a linear model

$$y = wx$$

- Problem boils down to fitting a line to the data
- w is the model parameter (slope of the line here)
- Many w's (i.e., many lines) can be fit to this data
- Which one is the best?

Fitting a (Hyper)Plane to the Data

- For 2-dim. inputs, we can fit a 2-dim. plane to the data



- In higher dimensions, we can likewise fit a hyperplane $\boldsymbol{w}^\top \boldsymbol{x} = 0$
  - Defined by a D-dim vector $\boldsymbol{w}$ normal to the plane
- Many planes are possible. Which one is the best?

Linear Regression

- **Given:** Training data with N examples $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$, $\boldsymbol{x}_n \in \mathbb{R}^D$, $y_n \in \mathbb{R}$
- Assume the following linear model with model parameters $\boldsymbol{w} \in \mathbb{R}^D$

$$\boxed{y_n \approx \boldsymbol{w}^\top \boldsymbol{x}_n \quad \Rightarrow \quad y_n \approx \sum_{d=1}^{D} w_d x_{nd}}$$

- The response y_n is a linear combination of the features of the inputs $\boldsymbol{x}_n$

Linear Regression

- **Given:** Training data with N examples $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$, $\boldsymbol{x}_n \in \mathbb{R}^D$, $y_n \in \mathbb{R}$
- Assume the following linear model with model parameters $\boldsymbol{w} \in \mathbb{R}^D$

$$\boxed{y_n \approx \boldsymbol{w}^\top \boldsymbol{x}_n \quad \Rightarrow \quad y_n \approx \sum_{d=1}^{D} w_d x_{nd}}$$

- The response y_n is a linear combination of the features of the inputs $\boldsymbol{x}_n$
- $\boldsymbol{w} \in \mathbb{R}^D$ is also called the (regression) **weight vector**
  - Can think of w_d as weight/importance of d-th feature in the data

Linear Regression

- **Given:** Training data with N examples $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^{N}$, $\boldsymbol{x}_n \in \mathbb{R}^D$, $y_n \in \mathbb{R}$
- Assume the following linear model with model parameters $\boldsymbol{w} \in \mathbb{R}^D$

$$\boxed{y_n \approx \boldsymbol{w}^\top \boldsymbol{x}_n \quad \Rightarrow \quad y_n \approx \sum_{d=1}^{D} w_d x_{nd}}$$

- The response y_n is a linear combination of the features of the inputs $\boldsymbol{x}_n$
- $\boldsymbol{w} \in \mathbb{R}^D$ is also called the (regression) **weight vector**
  - Can think of w_d as weight/importance of d-th feature in the data
- A simple and interpretable linear model. Can also re-express it compactly for all the N examples

$$\boldsymbol{y} \approx \mathbf{X}\boldsymbol{w} \qquad \text{(akin to a linear system of equations; } \boldsymbol{w} \text{ being the unknown)}$$

Linear Regression

- **Given:** Training data with N examples $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$, $\boldsymbol{x}_n \in \mathbb{R}^D$, $y_n \in \mathbb{R}$
- Assume the following linear model with model parameters $\boldsymbol{w} \in \mathbb{R}^D$

$$\boxed{y_n \approx \boldsymbol{w}^\top \boldsymbol{x}_n \quad \Rightarrow \quad y_n \approx \sum_{d=1}^{D} w_d x_{nd}}$$

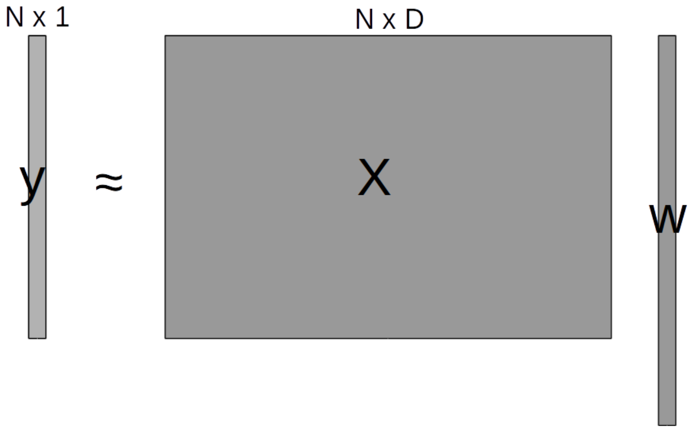
- The response y_n is a linear combination of the features of the inputs $\boldsymbol{x}_n$
- $\boldsymbol{w} \in \mathbb{R}^D$ is also called the (regression) **weight vector**
  - Can think of w_d as weight/importance of d-th feature in the data
- A simple and interpretable linear model. Can also re-express it compactly for all the N examples

$$\boldsymbol{y} \approx \mathbf{X}\boldsymbol{w} \qquad \text{(akin to a linear system of equations; } \boldsymbol{w} \text{ being the unknown)}$$

- Notation used here:
  - $\boldsymbol{w} \in \mathbb{R}^D$ and each $\boldsymbol{x}_n \in \mathbb{R}^D$ are $D \times 1$ column vectors
  - $\mathbf{X} = [\boldsymbol{x}_1 \ \boldsymbol{x}_2 \ \dots \ \boldsymbol{x}_N]^\top$ is an $N \times D$ matrix of features
  - $\boldsymbol{y} = [y_1 \ y_2 \ \dots \ y_N]^\top$ is an $N \times 1$ column vector of responses

Linear Regression

Linear system of equations with $\boldsymbol{w}$ being the unknown..



Linear Regression with Squared Loss

- Our linear regression model: $y_n \approx \boldsymbol{w}^\top \boldsymbol{x}_n$. The goal is to learn $\boldsymbol{w} \in \mathbb{R}^D$
- Let's use the squared loss to define our loss function

$$\ell(y_n, \boldsymbol{w}^\top \boldsymbol{x}_n) = (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$$

Linear Regression with Squared Loss

- Our linear regression model: $y_n \approx \boldsymbol{w}^\top \boldsymbol{x}_n$. The goal is to learn $\boldsymbol{w} \in \mathbb{R}^D$
- Let's use the squared loss to define our loss function

$$\ell(y_n, \boldsymbol{w}^\top \boldsymbol{x}_n) = (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$$

- **Note:** Squared loss chosen for simplicity; other losses can be used, e.g.,

$$\ell(y_n, \boldsymbol{w}^\top \boldsymbol{x}_n) = |y_n - \boldsymbol{w}^\top \boldsymbol{x}_n| \qquad \text{(more robust to outliers)}$$

Linear Regression with Squared Loss

- Our linear regression model: $y_n \approx \boldsymbol{w}^\top \boldsymbol{x}_n$. The goal is to learn $\boldsymbol{w} \in \mathbb{R}^D$
- Let's use the squared loss to define our loss function

$$\ell(y_n, \boldsymbol{w}^\top \boldsymbol{x}_n) = (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$$

- **Note:** Squared loss chosen for simplicity; other losses can be used, e.g.,

$$\ell(y_n, \boldsymbol{w}^\top \boldsymbol{x}_n) = |y_n - \boldsymbol{w}^\top \boldsymbol{x}_n| \qquad \text{(more robust to outliers)}$$

- Using the squared loss, the total (empirical) error on the training data

$$L_{emp}(\boldsymbol{w}) = \sum_{n=1}^{N} \ell(y_n, \boldsymbol{w}^\top \boldsymbol{x}_n) = \sum_{n=1}^{N} (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$$

Linear Regression with Squared Loss

- Our linear regression model: $y_n \approx \boldsymbol{w}^\top \boldsymbol{x}_n$. The goal is to learn $\boldsymbol{w} \in \mathbb{R}^D$
- Let's use the squared loss to define our loss function

$$\ell(y_n, \boldsymbol{w}^\top \boldsymbol{x}_n) = (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$$

- **Note:** Squared loss chosen for simplicity; other losses can be used, e.g.,

$$\ell(y_n, \boldsymbol{w}^\top \boldsymbol{x}_n) = |y_n - \boldsymbol{w}^\top \boldsymbol{x}_n| \qquad \text{(more robust to outliers)}$$

- Using the squared loss, the total (empirical) error on the training data

$$L_{emp}(\boldsymbol{w}) = \sum_{n=1}^{N} \ell(y_n, \boldsymbol{w}^\top \boldsymbol{x}_n) = \sum_{n=1}^{N} (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$$

- We'll estimate $\boldsymbol{w}$ by minimizing $L_{emp}(\boldsymbol{w})$ w.r.t. $\boldsymbol{w}$ (an optimization problem)

$$\boxed{\hat{\boldsymbol{w}} = \arg\min_{\mathbf{w}} \sum_{n=1}^{N} (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2}$$

Least Squares Linear Regression

- Recall our objective function: $L_{emp} = \sum_{n=1}^{N} (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$

[1] Please refer to the Matrix Cookbook for more results on vector/matrix derivatives

Least Squares Linear Regression

- Recall our objective function: $L_{emp} = \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$
- Taking derivative[1] of $L_{emp}(\boldsymbol{w})$ w.r.t. $\boldsymbol{w}$ and setting to zero

$$\sum_{n=1}^{N} 2(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)\frac{\partial}{\partial \boldsymbol{w}}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n) = 0 \quad \Rightarrow \quad \sum_{n=1}^{N} \boldsymbol{x}_n(y_n - \boldsymbol{x}_n^\top \boldsymbol{w}) = 0$$

[1] Please refer to the Matrix Cookbook for more results on vector/matrix derivatives

Least Squares Linear Regression

- Recall our objective function: $L_{emp} = \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$
- Taking derivative[1] of $L_{emp}(\boldsymbol{w})$ w.r.t. $\boldsymbol{w}$ and setting to zero

$$\sum_{n=1}^{N} 2(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)\frac{\partial}{\partial \boldsymbol{w}}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n) = 0 \quad \Rightarrow \quad \sum_{n=1}^{N} \boldsymbol{x}_n(y_n - \boldsymbol{x}_n^\top \boldsymbol{w}) = 0$$

- Simplifying further, we get a nice, closed form solution for $\boldsymbol{w}$

$$\boxed{\boldsymbol{w} = (\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^\top)^{-1} \sum_{n=1}^{N} y_n \boldsymbol{x}_n = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \boldsymbol{y}}$$

- Note: $\boldsymbol{x}_n$ is $D \times 1$, $\mathbf{X}$ is $N \times D$, $\boldsymbol{y}$ is $N \times 1$

[1] Please refer to the Matrix Cookbook for more results on vector/matrix derivatives

Least Squares Linear Regression

- Recall our objective function: $L_{emp} = \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$
- Taking derivative[1] of $L_{emp}(\boldsymbol{w})$ w.r.t. $\boldsymbol{w}$ and setting to zero

$$\sum_{n=1}^{N} 2(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)\frac{\partial}{\partial \boldsymbol{w}}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n) = 0 \quad \Rightarrow \quad \sum_{n=1}^{N} \boldsymbol{x}_n(y_n - \boldsymbol{x}_n^\top \boldsymbol{w}) = 0$$

- Simplifying further, we get a nice, closed form solution for $\boldsymbol{w}$

$$\boxed{\boldsymbol{w} = (\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^\top)^{-1} \sum_{n=1}^{N} y_n \boldsymbol{x}_n = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \boldsymbol{y}}$$

- Note: $\boldsymbol{x}_n$ is $D \times 1$, $\mathbf{X}$ is $N \times D$, $\boldsymbol{y}$ is $N \times 1$
- Analytic, closed form solution, but has some issues

[1] Please refer to the Matrix Cookbook for more results on vector/matrix derivatives

Least Squares Linear Regression

- Recall our objective function: $L_{emp} = \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$
- Taking derivative[1] of $L_{emp}(\boldsymbol{w})$ w.r.t. $\boldsymbol{w}$ and setting to zero

$$\sum_{n=1}^{N} 2(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)\frac{\partial}{\partial \boldsymbol{w}}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n) = 0 \quad \Rightarrow \quad \sum_{n=1}^{N} \boldsymbol{x}_n(y_n - \boldsymbol{x}_n^\top \boldsymbol{w}) = 0$$

- Simplifying further, we get a nice, closed form solution for $\boldsymbol{w}$

$$\boxed{\boldsymbol{w} = (\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^\top)^{-1} \sum_{n=1}^{N} y_n \boldsymbol{x}_n = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \boldsymbol{y}}$$

- Note: $\boldsymbol{x}_n$ is $D \times 1$, $\mathbf{X}$ is $N \times D$, $\boldsymbol{y}$ is $N \times 1$
- Analytic, closed form solution, but has some issues
  - We didn't impose any regularization on $\boldsymbol{w}$ (thus prone to overfitting)

[1] Please refer to the Matrix Cookbook for more results on vector/matrix derivatives

Least Squares Linear Regression

- Recall our objective function: $L_{emp} = \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$
- Taking derivative[1] of $L_{emp}(\boldsymbol{w})$ w.r.t. $\boldsymbol{w}$ and setting to zero

$$\sum_{n=1}^{N} 2(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)\frac{\partial}{\partial \boldsymbol{w}}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n) = 0 \quad \Rightarrow \quad \sum_{n=1}^{N} \boldsymbol{x}_n(y_n - \boldsymbol{x}_n^\top \boldsymbol{w}) = 0$$

- Simplifying further, we get a nice, closed form solution for $\boldsymbol{w}$

$$\boxed{\boldsymbol{w} = (\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^\top)^{-1} \sum_{n=1}^{N} y_n \boldsymbol{x}_n = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \boldsymbol{y}}$$

- Note: $\boldsymbol{x}_n$ is $D \times 1$, $\mathbf{X}$ is $N \times D$, $\boldsymbol{y}$ is $N \times 1$
- Analytic, closed form solution, but has some issues
  - We didn't impose any regularization on $\boldsymbol{w}$ (thus prone to overfitting)
  - Have to invert a $D \times D$ matrix; prohibitive especially when D (and N) is large

[1] Please refer to the Matrix Cookbook for more results on vector/matrix derivatives

Least Squares Linear Regression

- Recall our objective function: $L_{emp} = \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$
- Taking derivative[1] of $L_{emp}(\boldsymbol{w})$ w.r.t. $\boldsymbol{w}$ and setting to zero

$$\sum_{n=1}^{N} 2(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)\frac{\partial}{\partial \boldsymbol{w}}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n) = 0 \quad \Rightarrow \quad \sum_{n=1}^{N} \boldsymbol{x}_n(y_n - \boldsymbol{x}_n^\top \boldsymbol{w}) = 0$$

- Simplifying further, we get a nice, closed form solution for $\boldsymbol{w}$

$$\boxed{\boldsymbol{w} = (\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^\top)^{-1} \sum_{n=1}^{N} y_n \boldsymbol{x}_n = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \boldsymbol{y}}$$

- Note: $\boldsymbol{x}_n$ is $D \times 1$, $\mathbf{X}$ is $N \times D$, $\boldsymbol{y}$ is $N \times 1$
- Analytic, closed form solution, but has some issues
  - We didn't impose any regularization on $\boldsymbol{w}$ (thus prone to overfitting)
  - Have to invert a $D \times D$ matrix; prohibitive especially when D (and N) is large
  - The matrix $\mathbf{X}^\top \mathbf{X}$ may not even be invertible (e.g., when $D > N$). Unique solution not guaranteed

[1] Please refer to the Matrix Cookbook for more results on vector/matrix derivatives

Ridge Regression: Regularized Least Squares

- Least Squares objective: $L_{emp} = \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$
- No constraints/regularization on $\boldsymbol{w}$. Components $[w_1, w_2, \ldots, w_D]$ of $\boldsymbol{w}$ may become arbitrarily large. Why is this a bad thing to have?

Ridge Regression: Regularized Least Squares

- Least Squares objective: $L_{emp} = \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$
- No constraints/regularization on $\boldsymbol{w}$. Components $[w_1, w_2, \ldots, w_D]$ of $\boldsymbol{w}$ may become arbitrarily large. Why is this a bad thing to have?
- Let's add squared ℓ_2 norm of $\boldsymbol{w}$ as a regularizer: $R(f) = R(\boldsymbol{w}) = ||\boldsymbol{w}||^2$

Ridge Regression: Regularized Least Squares

- Least Squares objective: $L_{emp} = \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$
- No constraints/regularization on $\boldsymbol{w}$. Components $[w_1, w_2, \ldots, w_D]$ of $\boldsymbol{w}$ may become arbitrarily large. Why is this a bad thing to have?
- Let's add squared ℓ_2 norm of $\boldsymbol{w}$ as a regularizer: $R(f) = R(\boldsymbol{w}) = ||\boldsymbol{w}||^2$
- This results in the so-called "Ridge Regression" model

$$L_{reg} = \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2 + \lambda||\boldsymbol{w}||^2$$

- Note that $||\boldsymbol{w}||^2 = \boldsymbol{w}^\top \boldsymbol{w} = \sum_{d=1}^{D} w_d^2$

Ridge Regression: Regularized Least Squares

- Least Squares objective: $L_{emp} = \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$
- No constraints/regularization on $\boldsymbol{w}$. Components $[w_1, w_2, \ldots, w_D]$ of $\boldsymbol{w}$ may become arbitrarily large. Why is this a bad thing to have?
- Let's add squared ℓ_2 norm of $\boldsymbol{w}$ as a regularizer: $R(f) = R(\boldsymbol{w}) = ||\boldsymbol{w}||^2$
- This results in the so-called "Ridge Regression" model

$$L_{reg} = \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2 + \lambda||\boldsymbol{w}||^2$$

- Note that $||\boldsymbol{w}||^2 = \boldsymbol{w}^\top \boldsymbol{w} = \sum_{d=1}^{D} w_d^2$
- Minimizing L_{reg} will prevent components of $\boldsymbol{w}$ from becoming very large. Why is this nice?

Ridge Regression: Regularized Least Squares

- Least Squares objective: $L_{emp} = \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$
- No constraints/regularization on $\boldsymbol{w}$. Components $[w_1, w_2, \ldots, w_D]$ of $\boldsymbol{w}$ may become arbitrarily large. Why is this a bad thing to have?
- Let's add squared ℓ_2 norm of $\boldsymbol{w}$ as a regularizer: $R(f) = R(\boldsymbol{w}) = ||\boldsymbol{w}||^2$
- This results in the so-called "Ridge Regression" model

$$L_{reg} = \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2 + \lambda||\boldsymbol{w}||^2$$

- Note that $||\boldsymbol{w}||^2 = \boldsymbol{w}^\top \boldsymbol{w} = \sum_{d=1}^{D} w_d^2$
- Minimizing L_{reg} will prevent components of $\boldsymbol{w}$ from becoming very large. Why is this nice?
- Taking derivative of L_{reg} w.r.t. $\boldsymbol{w}$ and setting to zero gives (verify yourself)

$$\boxed{\boldsymbol{w} = (\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^\top + \lambda \mathbf{I}_D)^{-1} \sum_{n=1}^{N} y_n \boldsymbol{x}_n = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_D)^{-1} \mathbf{X}^\top \boldsymbol{y}}$$

Intuitively, Why Small Weights are Good?

- Small weights ensure that the function $y = f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$ is **smooth** (i.e., we expect similar $\boldsymbol{x}$'s to have similar y's).

Intuitively, Why Small Weights are Good?

- Small weights ensure that the function $y = f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$ is **smooth** (i.e., we expect similar $\boldsymbol{x}$'s to have similar y's). Below is an informal justification:

Intuitively, Why Small Weights are Good?

- Small weights ensure that the function $y = f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$ is **smooth** (i.e., we expect similar $\boldsymbol{x}$'s to have similar y's). Below is an informal justification:
- Consider two points $\boldsymbol{x}_n \in \mathbb{R}^D$ and $\boldsymbol{x}_m \in \mathbb{R}^D$ that are exactly similar in all features except the d-th feature where they differ by a small value, say ϵ

Intuitively, Why Small Weights are Good?

- Small weights ensure that the function $y = f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$ is **smooth** (i.e., we expect similar $\boldsymbol{x}$'s to have similar y's). Below is an informal justification:
- Consider two points $\boldsymbol{x}_n \in \mathbb{R}^D$ and $\boldsymbol{x}_m \in \mathbb{R}^D$ that are exactly similar in all features except the d-th feature where they differ by a small value, say ϵ
- Assuming a simple/smooth function $f(\boldsymbol{x})$, y_n and y_m should also be close

Intuitively, Why Small Weights are Good?

- Small weights ensure that the function $y = f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$ is **smooth** (i.e., we expect similar $\boldsymbol{x}$'s to have similar y's). Below is an informal justification:
- Consider two points $\boldsymbol{x}_n \in \mathbb{R}^D$ and $\boldsymbol{x}_m \in \mathbb{R}^D$ that are exactly similar in all features except the d-th feature where they differ by a small value, say ϵ
- Assuming a simple/smooth function $f(\boldsymbol{x})$, y_n and y_m should also be close
- However, as per the model $y = f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$, y_n and y_m will differ by ϵw_d

Intuitively, Why Small Weights are Good?

- Small weights ensure that the function $y = f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$ is **smooth** (i.e., we expect similar $\boldsymbol{x}$'s to have similar y's). Below is an informal justification:
- Consider two points $\boldsymbol{x}_n \in \mathbb{R}^D$ and $\boldsymbol{x}_m \in \mathbb{R}^D$ that are exactly similar in all features except the d-th feature where they differ by a small value, say ϵ
- Assuming a simple/smooth function $f(\boldsymbol{x})$, y_n and y_m should also be close
- However, as per the model $y = f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$, y_n and y_m will differ by ϵw_d
- Unless we constrain w_d to have a small value, the difference ϵw_d would also be very large (which isn't what we want).

Intuitively, Why Small Weights are Good?

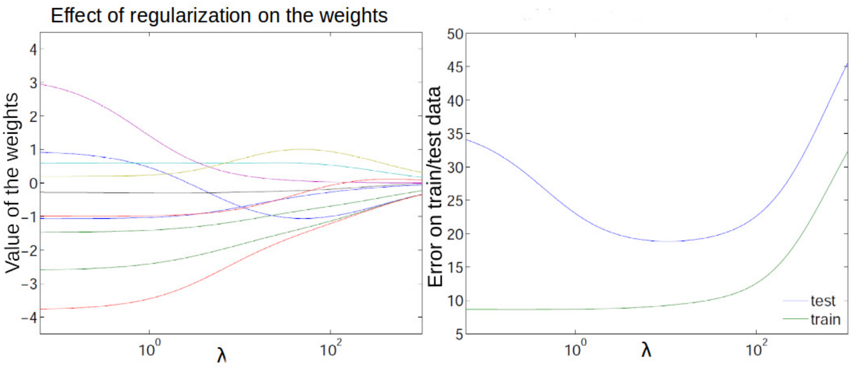
- Small weights ensure that the function $y = f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$ is **smooth** (i.e., we expect similar $\boldsymbol{x}$'s to have similar y's). Below is an informal justification:
- Consider two points $\boldsymbol{x}_n \in \mathbb{R}^D$ and $\boldsymbol{x}_m \in \mathbb{R}^D$ that are exactly similar in all features except the d-th feature where they differ by a small value, say ϵ
- Assuming a simple/smooth function $f(\boldsymbol{x})$, y_n and y_m should also be close
- However, as per the model $y = f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$, y_n and y_m will differ by ϵw_d
- Unless we constrain w_d to have a small value, the difference ϵw_d would also be very large (which isn't what we want).
- That's why regularizing (via ℓ_2 regularization) and making the individual components of the weight vector small helps

Intuitively, Why Small Weights are Good?

- Small weights ensure that the function $y = f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$ is **smooth** (i.e., we expect similar $\boldsymbol{x}$'s to have similar y's). Below is an informal justification:
- Consider two points $\boldsymbol{x}_n \in \mathbb{R}^D$ and $\boldsymbol{x}_m \in \mathbb{R}^D$ that are exactly similar in all features except the d-th feature where they differ by a small value, say ϵ
- Assuming a simple/smooth function $f(\boldsymbol{x})$, y_n and y_m should also be close
- However, as per the model $y = f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$, y_n and y_m will differ by ϵw_d
- Unless we constrain w_d to have a small value, the difference ϵw_d would also be very large (which isn't what we want).
- That's why regularizing (via ℓ_2 regularization) and making the individual components of the weight vector small helps
- Lesson: Don't learn a model that gives a single feature too much importance in the final prediction!

Ridge Regression: Effect of Regularization

- Consider ridge regression on some data with 10 features (thus the weight vector $\boldsymbol{w}$ has 10 components)



Solution via Gradient-based Methods

- Both least squares and ridge regression require matrix inversion

$$\begin{aligned} \text{Least Squares} \quad \boldsymbol{w} &= (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \boldsymbol{y} \\ \text{Ridge} \quad \boldsymbol{w} &= (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_D)^{-1}\mathbf{X}^\top \boldsymbol{y} \end{aligned}$$

- This can be computationally very expensive when D is very large

Solution via Gradient-based Methods

- Both least squares and ridge regression require matrix inversion

$$\begin{aligned} \text{Least Squares} \quad \boldsymbol{w} &= (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\boldsymbol{y} \\ \text{Ridge} \quad \boldsymbol{w} &= (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_D)^{-1}\mathbf{X}^\top\boldsymbol{y} \end{aligned}$$

- This can be computationally very expensive when D is very large
- We can instead solve for $\boldsymbol{w}$ more efficiently using generic/specialized optimization methods on the respective loss functions (L_{emp} or L_{reg})

Solution via Gradient-based Methods

- Both least squares and ridge regression require matrix inversion

$$\begin{aligned} \text{Least Squares} \quad \boldsymbol{w} &= (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \boldsymbol{y} \\ \text{Ridge} \quad \boldsymbol{w} &= (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_D)^{-1} \mathbf{X}^\top \boldsymbol{y} \end{aligned}$$

- This can be computationally very expensive when D is very large
- We can instead solve for $\boldsymbol{w}$ more efficiently using generic/specialized optimization methods on the respective loss functions (L_{emp} or L_{reg})
- A simple scheme can be the following iterative gradient-descent procedure

Solution via Gradient-based Methods

- Both least squares and ridge regression require matrix inversion

$$\begin{aligned} \text{Least Squares} \quad \boldsymbol{w} &= (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \boldsymbol{y} \\ \text{Ridge} \quad \boldsymbol{w} &= (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_D)^{-1}\mathbf{X}^\top \boldsymbol{y} \end{aligned}$$

- This can be computationally very expensive when D is very large
- We can instead solve for $\boldsymbol{w}$ more efficiently using generic/specialized optimization methods on the respective loss functions (L_{emp} or L_{reg})
- A simple scheme can be the following iterative gradient-descent procedure
  - Start with an initial value of $\boldsymbol{w} = \boldsymbol{w}^{(0)}$

Solution via Gradient-based Methods

- Both least squares and ridge regression require matrix inversion

$$\begin{aligned}\text{Least Squares} \quad \boldsymbol{w} &= (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\boldsymbol{y}\\ \text{Ridge} \quad \boldsymbol{w} &= (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_D)^{-1}\mathbf{X}^\top\boldsymbol{y}\end{aligned}$$

- This can be computationally very expensive when D is very large
- We can instead solve for $\boldsymbol{w}$ more efficiently using generic/specialized optimization methods on the respective loss functions (L_{emp} or L_{reg})
- A simple scheme can be the following iterative gradient-descent procedure
  - Start with an initial value of $\boldsymbol{w} = \boldsymbol{w}^{(0)}$
  - Update w by moving along the gradient of the loss function L (L_{emp} or L_{reg})

$$\boldsymbol{w}^{(t)} = \boldsymbol{w}^{(t-1)} - \eta \frac{\partial L}{\partial \boldsymbol{w}}\bigg|_{\boldsymbol{w}=\boldsymbol{w}^{(t-1)}}$$

  where η is the learning rate

Solution via Gradient-based Methods

- Both least squares and ridge regression require matrix inversion

$$\begin{aligned}\text{Least Squares} \quad \boldsymbol{w} &= (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \boldsymbol{y} \\ \text{Ridge} \quad \boldsymbol{w} &= (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_D)^{-1}\mathbf{X}^\top \boldsymbol{y}\end{aligned}$$

- This can be computationally very expensive when D is very large
- We can instead solve for $\boldsymbol{w}$ more efficiently using generic/specialized optimization methods on the respective loss functions (L_{emp} or L_{reg})
- A simple scheme can be the following iterative gradient-descent procedure
  - Start with an initial value of $\boldsymbol{w} = \boldsymbol{w}^{(0)}$
  - Update w by moving along the gradient of the loss function L (L_{emp} or L_{reg})

$$\boldsymbol{w}^{(t)} = \boldsymbol{w}^{(t-1)} - \eta \frac{\partial L}{\partial \boldsymbol{w}}\bigg|_{\boldsymbol{w}=\boldsymbol{w}^{(t-1)}}$$

    where η is the learning rate
  - Repeat until converge

Solution via Gradient-based Methods

- Both least squares and ridge regression require matrix inversion

$$\begin{aligned} \text{Least Squares} \quad \boldsymbol{w} &= (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \boldsymbol{y} \\ \text{Ridge} \quad \boldsymbol{w} &= (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_D)^{-1}\mathbf{X}^\top \boldsymbol{y} \end{aligned}$$

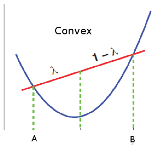
- This can be computationally very expensive when D is very large
- We can instead solve for $\boldsymbol{w}$ more efficiently using generic/specialized optimization methods on the respective loss functions (L_{emp} or L_{reg})
- A simple scheme can be the following iterative gradient-descent procedure
  - Start with an initial value of $\boldsymbol{w} = \boldsymbol{w}^{(0)}$
  - Update w by moving along the gradient of the loss function L (L_{emp} or L_{reg})

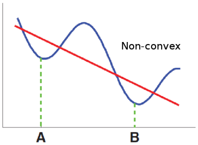
$$\boldsymbol{w}^{(t)} = \boldsymbol{w}^{(t-1)} - \eta \frac{\partial L}{\partial \boldsymbol{w}}\bigg|_{\boldsymbol{w}=\boldsymbol{w}^{(t-1)}}$$

  where η is the learning rate
  - Repeat until converge
- For unreg. least squares, the gradient is $\frac{\partial L}{\partial \mathbf{w}} = -\sum_{n=1}^{N} \boldsymbol{x}_n(y_n - \boldsymbol{x}_n^\top \boldsymbol{w})$

Gradient-based Methods: Some Notes

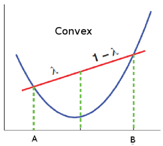
- Guaranteed to converge to a local minima
- Converge to global minima if the function is convex

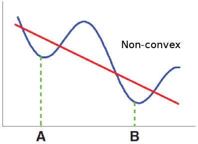




Gradient-based Methods: Some Notes

- Guaranteed to converge to a local minima
- Converge to global minima if the function is convex

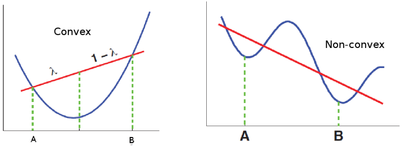




- Formally: Convex if second derivative is non-negative everywhere (for scalar functions) or if Hessian is positive semi-definite (for vector-valued functions). For a convex function, every local minima is also a global minima.

Gradient-based Methods: Some Notes

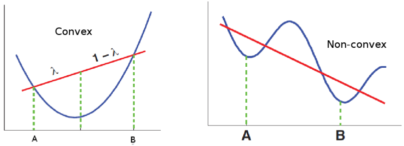
- Guaranteed to converge to a local minima
- Converge to global minima if the function is convex



- Formally: Convex if second derivative is non-negative everywhere (for scalar functions) or if Hessian is positive semi-definite (for vector-valued functions). For a convex function, every local minima is also a global minima.
- Note: The squared loss function in linear regression is convex
    - With ℓ_2 regularizer, it becomes strictly convex (single global minima)

Gradient-based Methods: Some Notes

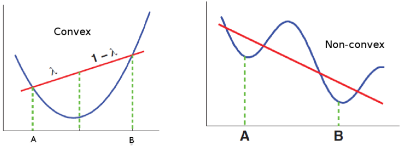
- Guaranteed to converge to a local minima
- Converge to global minima if the function is convex



- Formally: Convex if second derivative is non-negative everywhere (for scalar functions) or if Hessian is positive semi-definite (for vector-valued functions). For a convex function, every local minima is also a global minima.
- Note: The squared loss function in linear regression is convex
    - With ℓ_2 regularizer, it becomes strictly convex (single global minima)
- Learning rate is important (should not be too large or too small)

Gradient-based Methods: Some Notes

- Guaranteed to converge to a local minima
- Converge to global minima if the function is convex



- Formally: Convex if second derivative is non-negative everywhere (for scalar functions) or if Hessian is positive semi-definite (for vector-valued functions). For a convex function, every local minima is also a global minima.
- Note: The squared loss function in linear regression is convex
  - With ℓ_2 regularizer, it becomes strictly convex (single global minima)
- Learning rate is important (should not be too large or too small)
- Can also use stochastic/online gradient descent for more speed-ups. Require computing the gradients using only one or a small number of examples

Some Aspects about Linear Regression

- A simple and interpretable method. Very widely used.

Some Aspects about Linear Regression

- A simple and interpretable method. Very widely used.
- Highly scalable using efficient optimization solvers

Some Aspects about Linear Regression

- A simple and interpretable method. Very widely used.
- Highly scalable using efficient optimization solvers
- Ridge uses an ℓ_2 regularizer on weights. Other regularizers can be used

Some Aspects about Linear Regression

- A simple and interpretable method. Very widely used.
- Highly scalable using efficient optimization solvers
- Ridge uses an ℓ_2 regularizer on weights. Other regularizers can be used
  - E.g., ℓ_1 regularization $||\boldsymbol{w}||_1 = \sum_{d=1}^{D} |w_d|$

Some Aspects about Linear Regression

- A simple and interpretable method. Very widely used.
- Highly scalable using efficient optimization solvers
- Ridge uses an ℓ_2 regularizer on weights. Other regularizers can be used
  - E.g., ℓ_1 regularization $||\boldsymbol{w}||_1 = \sum_{d=1}^{D} |w_d|$
  - This regularizer promotes $\boldsymbol{w}$ to have very few nonzero components (reason discussed later)

Some Aspects about Linear Regression

- A simple and interpretable method. Very widely used.
- Highly scalable using efficient optimization solvers
- Ridge uses an ℓ_2 regularizer on weights. Other regularizers can be used
  - E.g., ℓ_1 regularization $||\boldsymbol{w}||_1 = \sum_{d=1}^{D} |w_d|$
  - This regularizer promotes $\boldsymbol{w}$ to have very few nonzero components (reason discussed later)
  - Optimization is not as straightforward as the ℓ_2 regularizer case

Some Aspects about Linear Regression

- A simple and interpretable method. Very widely used.
- Highly scalable using efficient optimization solvers
- Ridge uses an ℓ_2 regularizer on weights. Other regularizers can be used
  - E.g., ℓ_1 regularization $||\boldsymbol{w}||_1 = \sum_{d=1}^{D} |w_d|$
  - This regularizer promotes $\boldsymbol{w}$ to have very few nonzero components (reason discussed later)
  - Optimization is not as straightforward as the ℓ_2 regularizer case
  - We will also discuss this and other choices of regularizers in later classes

Some Aspects about Linear Regression

- A simple and interpretable method. Very widely used.
- Highly scalable using efficient optimization solvers
- Ridge uses an ℓ_2 regularizer on weights. Other regularizers can be used
  - E.g., ℓ_1 regularization $||\boldsymbol{w}||_1 = \sum_{d=1}^{D} |w_d|$
  - This regularizer promotes $\boldsymbol{w}$ to have very few nonzero components (reason discussed later)
  - Optimization is not as straightforward as the ℓ_2 regularizer case
  - We will also discuss this and other choices of regularizers in later classes
- The basic (regularized) linear regression can also be easily extended to

Some Aspects about Linear Regression

- A simple and interpretable method. Very widely used.
- Highly scalable using efficient optimization solvers
- Ridge uses an ℓ_2 regularizer on weights. Other regularizers can be used
  - E.g., ℓ_1 regularization $||\boldsymbol{w}||_1 = \sum_{d=1}^{D} |w_d|$
  - This regularizer promotes $\boldsymbol{w}$ to have very few nonzero components (reason discussed later)
  - Optimization is not as straightforward as the ℓ_2 regularizer case
  - We will also discuss this and other choices of regularizers in later classes
- The basic (regularized) linear regression can also be easily extended to
  - Nonlinear Regression $y_n \approx \boldsymbol{w}^\top \phi(\boldsymbol{x}_n)$ by replacing the original feature vector $\boldsymbol{x}_n$ by a nonlinear transformation $\phi(\boldsymbol{x}_n)$.

Some Aspects about Linear Regression

- A simple and interpretable method. Very widely used.
- Highly scalable using efficient optimization solvers
- Ridge uses an ℓ_2 regularizer on weights. Other regularizers can be used
  - E.g., ℓ_1 regularization $||\boldsymbol{w}||_1 = \sum_{d=1}^{D} |w_d|$
  - This regularizer promotes $\boldsymbol{w}$ to have very few nonzero components (reason discussed later)
  - Optimization is not as straightforward as the ℓ_2 regularizer case
  - We will also discuss this and other choices of regularizers in later classes
- The basic (regularized) linear regression can also be easily extended to
  - Nonlinear Regression $y_n \approx \boldsymbol{w}^\top \phi(\boldsymbol{x}_n)$ by replacing the original feature vector $\boldsymbol{x}_n$ by a nonlinear transformation $\phi(\boldsymbol{x}_n)$.
  - Another way to do nonlinear regression: $y_n \approx f(\boldsymbol{x}_n)$ where f is modeled by a deep neural net

Some Aspects about Linear Regression

- A simple and interpretable method. Very widely used.
- Highly scalable using efficient optimization solvers
- Ridge uses an ℓ_2 regularizer on weights. Other regularizers can be used
  - E.g., ℓ_1 regularization $||\boldsymbol{w}||_1 = \sum_{d=1}^{D} |w_d|$
  - This regularizer promotes $\boldsymbol{w}$ to have very few nonzero components (reason discussed later)
  - Optimization is not as straightforward as the ℓ_2 regularizer case
  - We will also discuss this and other choices of regularizers in later classes
- The basic (regularized) linear regression can also be easily extended to
  - Nonlinear Regression $y_n \approx \boldsymbol{w}^\top \phi(\boldsymbol{x}_n)$ by replacing the original feature vector $\boldsymbol{x}_n$ by a nonlinear transformation $\phi(\boldsymbol{x}_n)$.
  - Another way to do nonlinear regression: $y_n \approx f(\boldsymbol{x}_n)$ where f is modeled by a deep neural net
  - Generalized Linear Model $y_n = g(\boldsymbol{w}^\top \boldsymbol{x}_n)$ when response y_n is not real-valued but binary/categorical/count, etc, and g is a "link function"

Unsupervised Learning as Optimization

- Can also formulate unsupervised learning problems as optimization problems
- Consider an unsupervised learning problem with data $\mathbf{X} = \{\boldsymbol{x}_n\}_{n=1}^N$
- No labels. We are interested in learning a new representation $\mathbf{Z} = \{\boldsymbol{z}_n\}_{n=1}^N$

Unsupervised Learning as Optimization

- Can also formulate unsupervised learning problems as optimization problems
- Consider an unsupervised learning problem with data $\mathbf{X} = \{\boldsymbol{x}_n\}_{n=1}^N$
- No labels. We are interested in learning a new representation $\mathbf{Z} = \{\boldsymbol{z}_n\}_{n=1}^N$
- Assume a function f that models the relationship between $\boldsymbol{x}_n$ and $\boldsymbol{z}_n$

$$\boldsymbol{x}_n \approx f(\boldsymbol{z}_n) \quad \forall n$$

Unsupervised Learning as Optimization

- Can also formulate unsupervised learning problems as optimization problems
- Consider an unsupervised learning problem with data $\mathbf{X} = \{\boldsymbol{x}_n\}_{n=1}^N$
- No labels. We are interested in learning a new representation $\mathbf{Z} = \{\boldsymbol{z}_n\}_{n=1}^N$
- Assume a function f that models the relationship between $\boldsymbol{x}_n$ and $\boldsymbol{z}_n$

$$\boldsymbol{x}_n \approx f(\boldsymbol{z}_n) \quad \forall n$$

- In this case, we can define a loss function $\ell(\mathbf{x}_n, f(\boldsymbol{z}_n))$ that measures how well f can "reconstruct" the original $\boldsymbol{x}_n$ from its new representation $\boldsymbol{z}_n$
- This generic unsupervised learning problem can thus be written as

$$\hat{f} = \arg\min_{f,\mathbf{Z}} \sum_{n=1}^{N} \ell(\mathbf{x}_n, f(\boldsymbol{z}_n)) + \lambda R(f, \mathbf{Z})$$

- In this case both f and $\mathbf{Z}$ need to be learned (usually, in an alternating fashion, until you converge; more on this when we discuss unsupervised learning)