Linear Models and Learning via Optimization

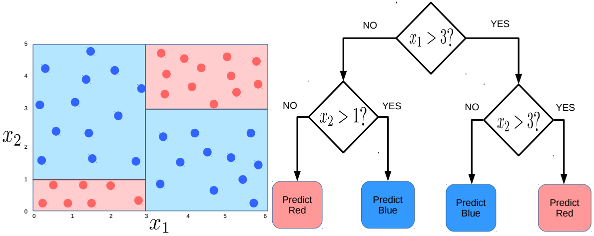
Piyush Rai

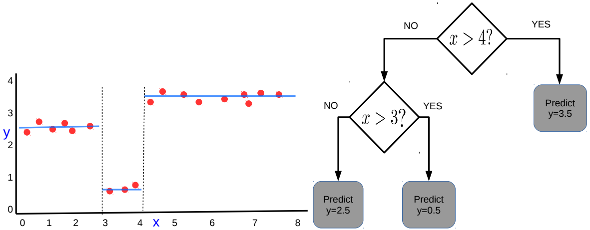
Introduction to Machine Learning (CS771A)

August 9, 2018

Recap

Decision Trees: Learning by asking questions. Ask the "important" questions first!



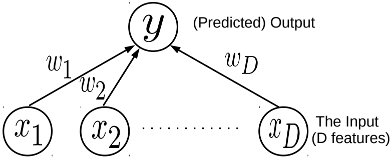


Linear Models

Linear Models

- Consider learning to map an input $\boldsymbol{x} \in \mathbb{R}^D$ to its output y (say real-valued)
- Assume the output to be a linear weighted combination of the D input features

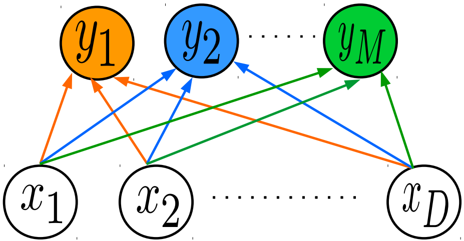
$$y = \sum_{d=1}^{D} w_d x_d = \boldsymbol{w}^\top \boldsymbol{x}$$



- This is an example of a linear model with D parameters $\boldsymbol{w} = [w_1, w_2, \ldots, w_D]$
- Inspired by **linear models of neurons**
- $\boldsymbol{w} \in \mathbb{R}^D$ is also known as the weight vector
- Here w_d denotes how important the d-th input feature is for predicting y
- The above is basically a linear model for simple regression (single, real-valued output y)
- This basic model can also be used as building blocks in many more complex models

Linear Models for Multi-output Regression

- Can assume each of the M outputs in $\boldsymbol{y} \in \mathbb{R}^M$ to be modeled by a linear model

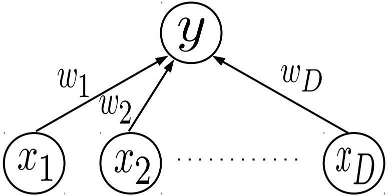


- Each output y_m $(m = 1, \ldots, M)$ modeled by a weight vector $\boldsymbol{w}_m \in \mathbb{R}^D$: $y_m = \boldsymbol{w}_m^\top \boldsymbol{x}$
- The entire model for all M outputs can be represented as $\boldsymbol{y} = \mathbf{W}^\top \boldsymbol{x}$
- $\mathbf{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M]$ is a $D \times M$ matrix

Linear Models for Binary Classification

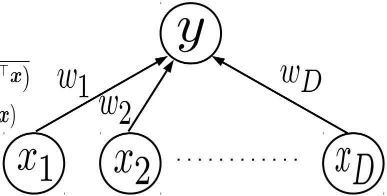
- Use the sign of the "score" $\boldsymbol{w}^\top \boldsymbol{x}$ to do predict binary label

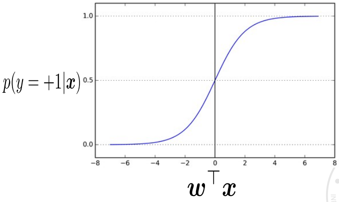
$$y = +1 \quad \text{if} \quad \boldsymbol{w}^\top \boldsymbol{x} \geq 0$$
$$y = -1 \quad \text{if} \quad \boldsymbol{w}^\top \boldsymbol{x} < 0$$
$$y = \text{sign}(\boldsymbol{w}^\top \boldsymbol{x})$$



- If desired, can turn the score $\boldsymbol{w}^\top \boldsymbol{x}$ into the probability of the label being $+1$ (logistic regression)

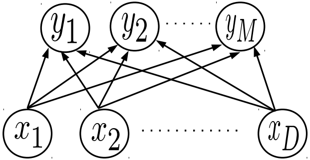
$$p(y = +1|\boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{w}^\top \boldsymbol{x})}$$
$$p(y = -1|\boldsymbol{x}) = 1 - p(y = +1|\boldsymbol{x})$$





Linear Models for Multi-class/Multi-label Classification

- Recall that, in multi-class/multi-label classification, $\boldsymbol{y} = [y_1, y_2, \ldots, y_M]$ is a vector of length M



- Just like multi-output regression, each component y_m of $\boldsymbol{y}$ can be modeled by a weight vector $\boldsymbol{w}_m$
- Need a way to convert $\boldsymbol{y} \in \mathbb{R}^M$ to one-hot (for multi-class)/binary vector (for multi-label)
- Note: In some cases, the score need not be converted, e.g.,
  - Can use the index of largest entry in $\boldsymbol{y}$ as the predicted class in multi-class classification

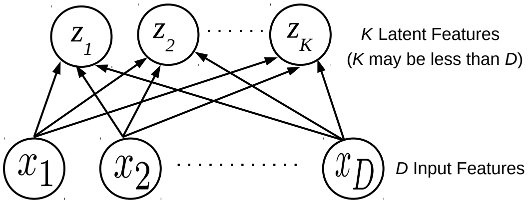
0.25	0.6	0.1	0.4	0.2

  - Can use the indices of top few entries in $\boldsymbol{y}$ as the predicted labels in multi-label classification

0.25	0.6	0.1	0.4	0.2

Linear Models for Dimensionality Reduction

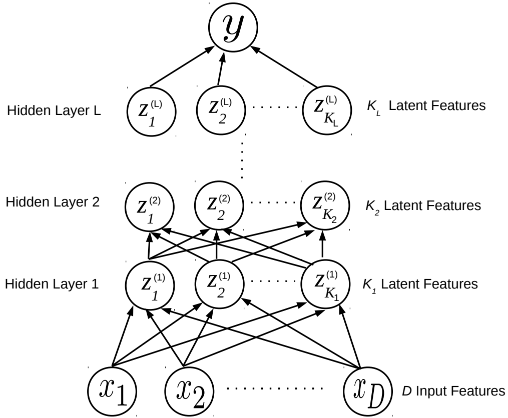
- Linear models can be used to reduce data-dimensionality (e.g., Principal Component Analysis)



- Note that it looks similar to multi-output regression but the output vector $\boldsymbol{z}$ is latent
  - An example of an unsupervised learning problem
- Need to learn both $\boldsymbol{z}$ and $\mathbf{W}$ in these problems

Linear Models to construct Deep Neural Networks

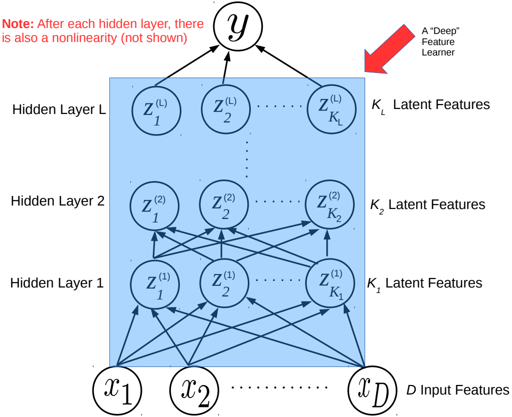
- Linear models are used as basic components of deep neural networks (nonlinear models)



- Each hidden layer has a learned **latent features** based representation of the original input $\boldsymbol{x}$

Linear Models to construct Deep Neural Networks

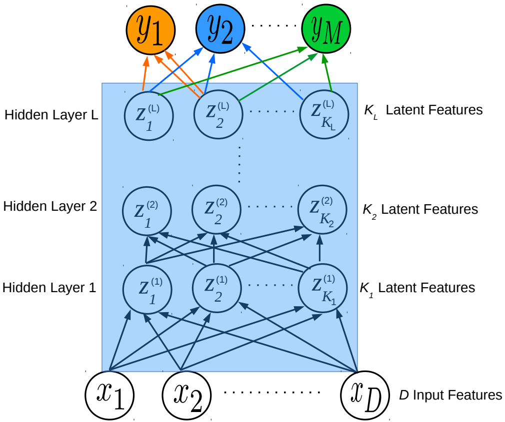
- Linear models are used as basic components of deep neural networks (nonlinear models)



- Each hidden layer has a learned **latent features** based representation of the original input $\boldsymbol{x}$

Linear Models to construct Deep Neural Networks

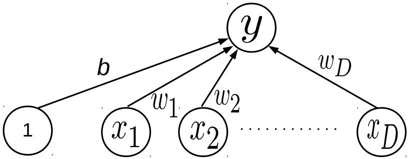
- Can even construct multiple-output versions of deep neural network



- These can be used for multi-output regression, multi-class/multi-label classification, etc.

Linear Models with Offset (Bias) Parameter

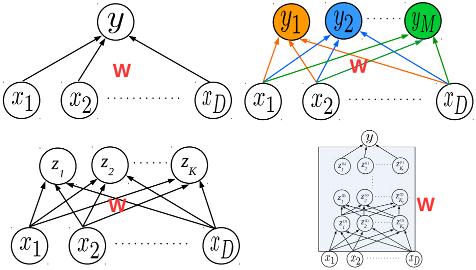
- Some linear models use an additional bias parameter b



$$y = \sum_{d=1}^{D} w_d x_d + b = \boldsymbol{w}^\top \boldsymbol{x} + b$$

- Can append a constant feature "1" for each input and rewrite as $y = \boldsymbol{w}^\top \boldsymbol{x}$, with $\boldsymbol{x}, \boldsymbol{w} \in \mathbb{R}^{D+1}$
- We will assume the same and omit the explicit bias for simplicity of notation

Learning Linear Models



Linear Models are ubiquitous!

How do we <u>learn</u> them from data?

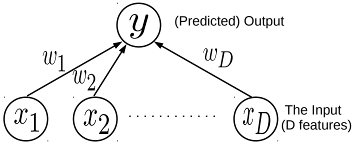
For linear models, learning $=$ Learning the model parameters (the weights)

We will formulate learning as an optimization problem w.r.t. these parameters

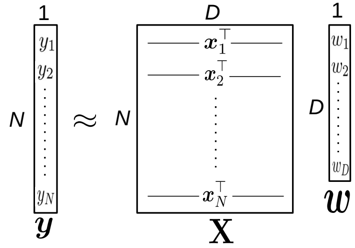
Learning a Linear Model for Regression

- Let's focus on learning the simplest linear model for now: Linear Regression

$$y = \sum_{d=1}^{D} w_d x_d = \boldsymbol{w}^\top \boldsymbol{x}$$

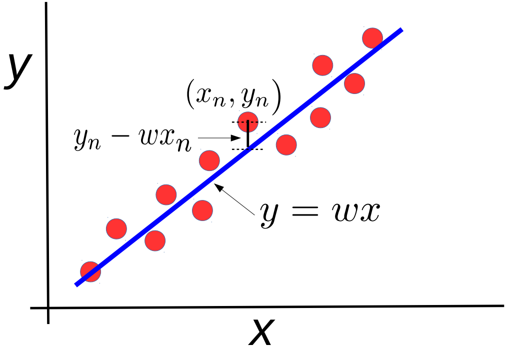


- Suppose we are given regression training data $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$ with $\boldsymbol{x}_n \in \mathbb{R}^D$, and $y_n \in \mathbb{R}$
- Let's model the training data using $\boldsymbol{w}$ and assume $y_n \approx \boldsymbol{w}^\top \boldsymbol{x}_n$, $\forall n$ (equivalently $\boldsymbol{y} \approx \mathbf{X}\boldsymbol{w}$)



Linear Regression: Pictorially

- With one-dimensional inputs, linear regression would look like



- Error of the model for an example $= y_n - \boldsymbol{w}^\top \boldsymbol{x}_n$ ($= y_n - wx_n$ for scalar input case)

Linear Regression

- Define the total error or "loss" on the training data, when using $\boldsymbol{w}$ as our model, as

$$\mathcal{L}(\boldsymbol{w}) = \sum_{n=1}^{N} (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$$

- Note: Squared loss chosen for simplicity. Can define other type of losses too (more on this later)
- The best $\boldsymbol{w}$ will be the one that minimizes the above error (requires optimization w.r.t. $\boldsymbol{w}$)

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

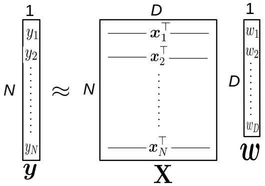
- This is known as "least squares" linear regression (Gauss/Legendre, early 18th century)
- Taking derivative (gradient) of $\mathcal{L}(\boldsymbol{w})$ w.r.t. $\boldsymbol{w}$ and setting to zero

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

- Simplifying further, we get a closed form solution for $\boldsymbol{w} \in \mathbb{R}^D$

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

Linear Regression



- Consider the closed form solution we obtained for linear regression based on least squares

$$\boldsymbol{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \boldsymbol{y}$$

- The above closed form solution is nice but has some issues
  - The $D \times D$ matrix $\mathbf{X}^\top \mathbf{X}$ may not be invertible
  - Based solely on minimizing the training error $\sum_{n=1}^{N} (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2 \Rightarrow$ can overfit the training data
  - Expensive inversion for large D: Can used iterative optimization techniques (will come to this later)

Regularized Linear Regression (a.k.a. Ridge Regression)

- Consider regularized loss: Training error + ℓ_2-squared norm of $\boldsymbol{w}$, i.e., $||\boldsymbol{w}||_2^2 = \boldsymbol{w}^\top \boldsymbol{w} = \sum_{d=1}^D w_d^2$

$$\mathcal{L}_{reg}(\boldsymbol{w}) = \left[\sum_{n=1}^N (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2 + \lambda \boldsymbol{w}^\top \boldsymbol{w}\right]$$

- Minimizing the above objective w.r.t. $\boldsymbol{w}$ does two things
  - Keeps the training error small
  - Keeps the ℓ_2 norm of $\boldsymbol{w}$ small (and thus also the individual components of $\boldsymbol{w}$): **Regularization**
- There is a trade-off between the two terms: The regularization hyperparam $\lambda > 0$ controls it
  - Very small λ means almost no regularization (can overfit)
  - Very large λ means very high regularization (can underfit - high training error)
  - Can use cross-validation to choose the "right" λ
- The solution to the above optimization problem is: $\boldsymbol{w} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_D)^{-1} \mathbf{X}^\top \boldsymbol{y}$
- Note that, in this case, regularization also made inversion possible (note the $\lambda \mathbf{I}_D$ term)

How ℓ_2 Regularization Helps Here?

- We saw that ℓ_2 regularization encourages the individual weights in $\boldsymbol{w}$ to be small
- 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

Regularization: Some Comments

- Many ways to regularize ML models (for linear as well as other models)
- Some are based on adding a norm of $\boldsymbol{w}$ to the loss function (as we already saw)
  - Using ℓ_2 norm in the loss function promotes the individual entries $\boldsymbol{w}$ to be small (we saw that)
  - Using ℓ_0 norm encourages very few non-zero entries in $\boldsymbol{w}$ (thereby promoting "sparse" $\boldsymbol{w}$)

$$||\boldsymbol{w}||_0 = \#\text{nnz}(\boldsymbol{w})$$

  - Optimizing with ℓ_0 is difficult (NP-hard problem); can use ℓ_1 norm as an approximation

$$||\boldsymbol{w}||_1 = \sum_{d=1}^{D} |w_d|$$

  - Note: Since they learn a sparse $\boldsymbol{w}$, ℓ_0 or ℓ_1 regularization is also useful for doing feature selection ($w_d = 0$ means feature d is irrelevant). We will revisit ℓ_1 later to formally see why ℓ_1 gives sparsity
- Other techniques for regularization: Early stopping (of training), "dropout", etc (popular in deep neural networks; will revisit these later when discussing deep learning)

Linear/Ridge Regression via Gradient Descent

- Both least squares regression and ridge regression require matrix inversion

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

- Can be computationally expensive when D is very large
- A faster way is to use iterative optimization, such as batch or stochastic gradient descent
- A basic batch gradient-descent based procedure looks like
  - Start with an initial value of $\boldsymbol{w} = \boldsymbol{w}^{(0)}$
  - Update $\boldsymbol{w}$ by moving along the gradient of the loss function $\mathcal{L}$

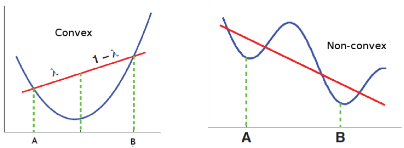
$$\boldsymbol{w}^{(t)} = \boldsymbol{w}^{(t-1)} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}\bigg|_{\boldsymbol{w}=\boldsymbol{w}^{(t-1)}} \qquad \text{where } \eta \text{ is the learning rate}$$

  - Repeat until converge
- For least squares, the gradient is $\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = -\sum_{n=1}^{N} \boldsymbol{x}_n(y_n - \boldsymbol{x}_n^\top \boldsymbol{w})$ (no matrix inversion involved)
- Such iterative methods for optimizing loss functions are widely used in ML. Will revisit these later

Linear Regression via Gradient-based Methods: Some Notes

We will revisit gradient based methods later but a few things to keep in mind

- Gradient Descent guaranteed to converge to a local minima
- Gradient Descent converges to global minima if the function is convex



- A function is 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)
- For Gradient Descent, the learning rate is important (should not be too large or too small)

Linear Regression as Solving System of Linear Equations

- Solving $\boldsymbol{y} = \mathbf{X}\boldsymbol{w}$ for $\boldsymbol{w}$ is like solving for D unknowns $w_1, \ldots, w_D$ using N equations

$$
\begin{aligned}
y_1 &= x_{11}w_1 + x_{12}w_2 + \ldots + x_{1D}w_D \\
y_2 &= x_{21}w_1 + x_{22}w_2 + \ldots + x_{2D}w_D \\
&\vdots \\
y_N &= x_{N1}w_1 + x_{N2}w_2 + \ldots + x_{ND}w_D
\end{aligned}
$$

- Can therefore view the linear regression problem as a system of linear equations
- However, in linear regression, we would rarely have $N = D$, but $N > D$ or $D > N$
- $N > D$ case is an overdetermined system of linear equations (# equations > # unknowns)
- $D > N$ case is an underdetermined system of linear equations (# unknowns > # equations)
- Thus methods to solve over/underdetermined systems can be used to solve linear regression as well
  - Many of these don't require a matrix inversion (will provide a separate note with details)

Linear Regression: Some Other Comments

- A simple and interpretable method. Very widely used.
- Least squares and ridge regression are one of the very few ML problems with closed form solutions

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

- Many ML problems can be easily reduced to the form $\boldsymbol{y} = \mathbf{X}\boldsymbol{w}$ or $\mathbf{Y} = \mathbf{X}\mathbf{W}$
- Equivalence to over/underdetermined system of linear equations enables us to use efficient solvers (a lot of work in the numerical linear algebra community to scale up linear systems solvers)
  - **An interesting bit:** Note that $\boldsymbol{w} = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \boldsymbol{y} \Rightarrow \mathbf{A}\boldsymbol{w} = \boldsymbol{b}$ where $\mathbf{A} = \mathbf{X}^\top \mathbf{X}$ and $\boldsymbol{b} = \mathbf{X}^\top \boldsymbol{y}$
  - Using the above relation, can solve for $\boldsymbol{w}$ by solving $\mathbf{A}\boldsymbol{w} = \boldsymbol{b}$. A standard linear system with D equations and D unknowns; can be solved using efficient linear systems solvers.
- 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 $\phi(\boldsymbol{x}_n)$ (where ϕ may be pre-defined or itself learned)
  - 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"

General Supervised Learning as Optimization

- We saw that regularized least squares regression required solving

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \mathcal{L}_{reg}(\boldsymbol{w}) = \arg\min_{\boldsymbol{w}} \left[\sum_{n=1}^{N} (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2 + \frac{\lambda}{2} \boldsymbol{w}^\top \boldsymbol{w} \right]$$

- This is essentially the training loss (called "empirical loss"), plus the regularization term
- In general, for supervised learning, the goal is to learn a function f, s.t. $f(\boldsymbol{x}_n) \approx y_n, \forall n$
- Moreover, we also want to have a simple f, i.e., have some regularization
- Therefore, learning the best f amounts to solving the following optimization problem

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

  where $\ell(y_n, f(\boldsymbol{x}_n))$ measures the model f's training loss on $(\boldsymbol{x}_n, y_n)$ and $R(f)$ is a regularizer
- For least squares regression, $f(\boldsymbol{x}_n) = \boldsymbol{w}^\top \boldsymbol{x}$, and $R(f) = \boldsymbol{w}^\top \boldsymbol{w}$, and $\ell(y_n, f(\boldsymbol{x}_n)) = (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$
- As we'll see later, different supervised learning problems differ in the choice of f, $R(.)$, and ℓ

General Unsupervised Learning as Optimization

- Can we formulate unsupervised learning problems as optimization problems? Yes, of course! :-)
- Consider an unsupervised learning problem with N inputs $\mathbf{X} = \{\boldsymbol{x}_n\}_{n=1}^N$
- Unsupervised, so no labels. Suppose 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 unsup. learning problem can thus be written as the following optimization problem

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

- In this case both f and $\mathbf{Z}$ need to be learned. Typically learned via alternating optimization