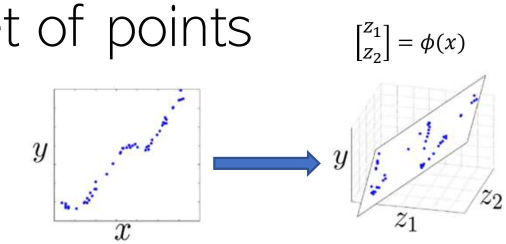
Linear Regression

Linear Regression: Pictorially

- Linear regression is like fitting a line or (hyper)plane to a set of points

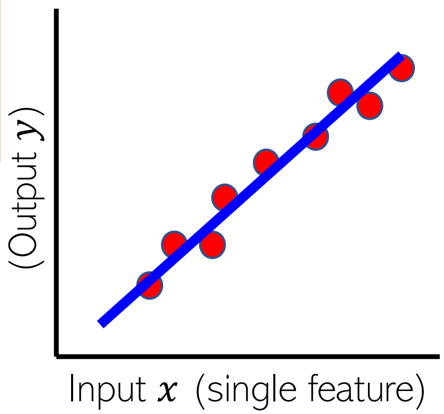


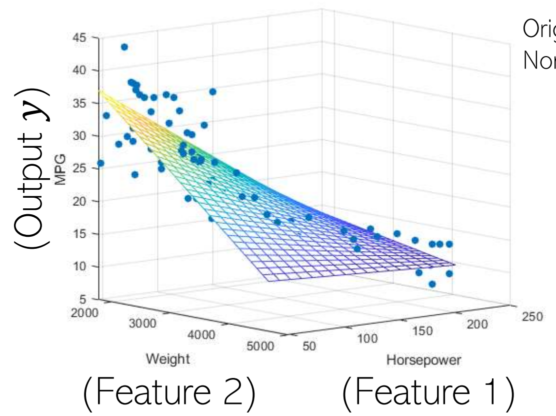
Original (single) feature
Nonlinear curve needed

Two features
Can fit a plane (linear)

What if a line/plane doesn't model the input-output relationship very well, e.g., if their relationship is better modeled by a nonlinear curve or curved surface?

Do linear models become useless in such cases?





No. We can even fit a curve using a linear model after suitably transforming the inputs

$$y \approx \boldsymbol{w}^\top \phi(\boldsymbol{x})$$

The transformation $\phi(.)$ can be predefined or learned (e.g., using kernel methods or a deep neural network based feature extractor). More on this later

- The line/plane must also predict outputs the unseen (test) inputs well

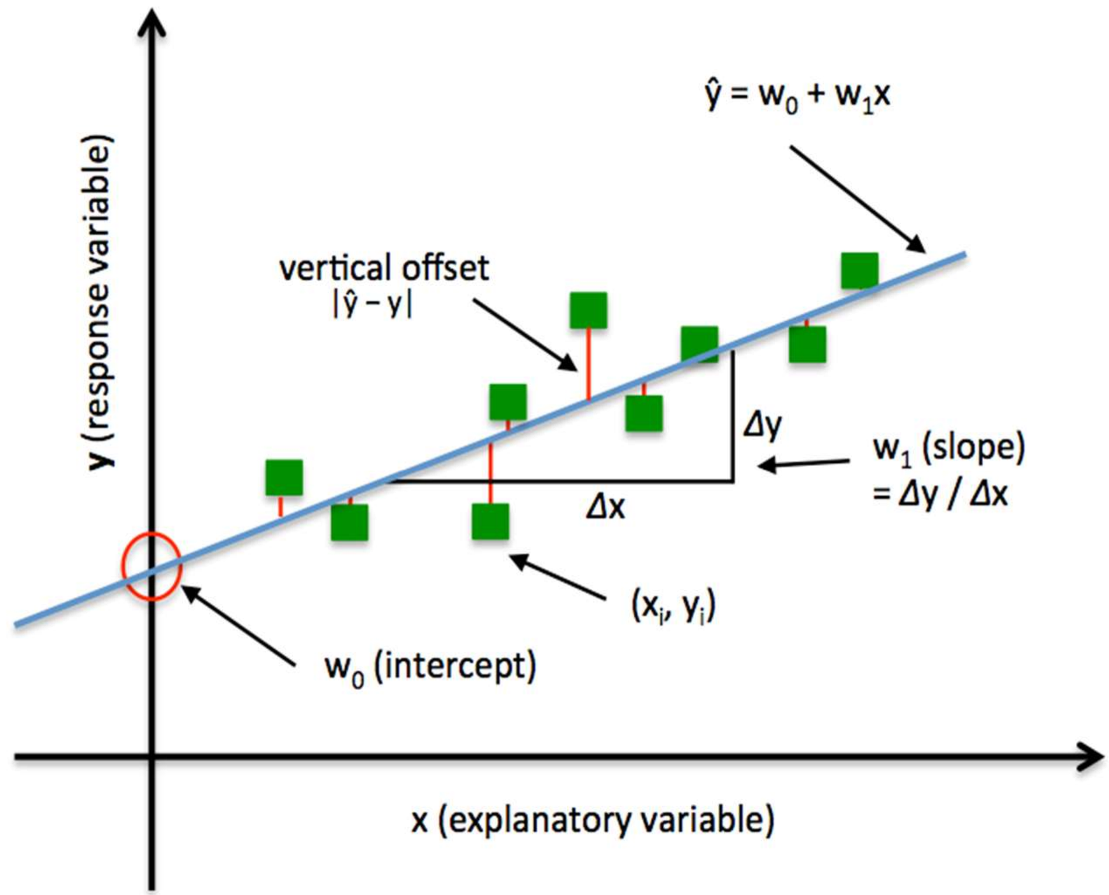
Simplest Possible Linear Regression Model

- This is the base model for <u>all</u> statistical machine learning
- x is a one feature data variable
- y is the value we are trying to predict
- The regression model is

$$y = w_0 + w_1 x + \varepsilon$$

Two parameters to estimate – the slope of the line w_1 and the y-intercept w_0

- ε is the unexplained, random, or error component.

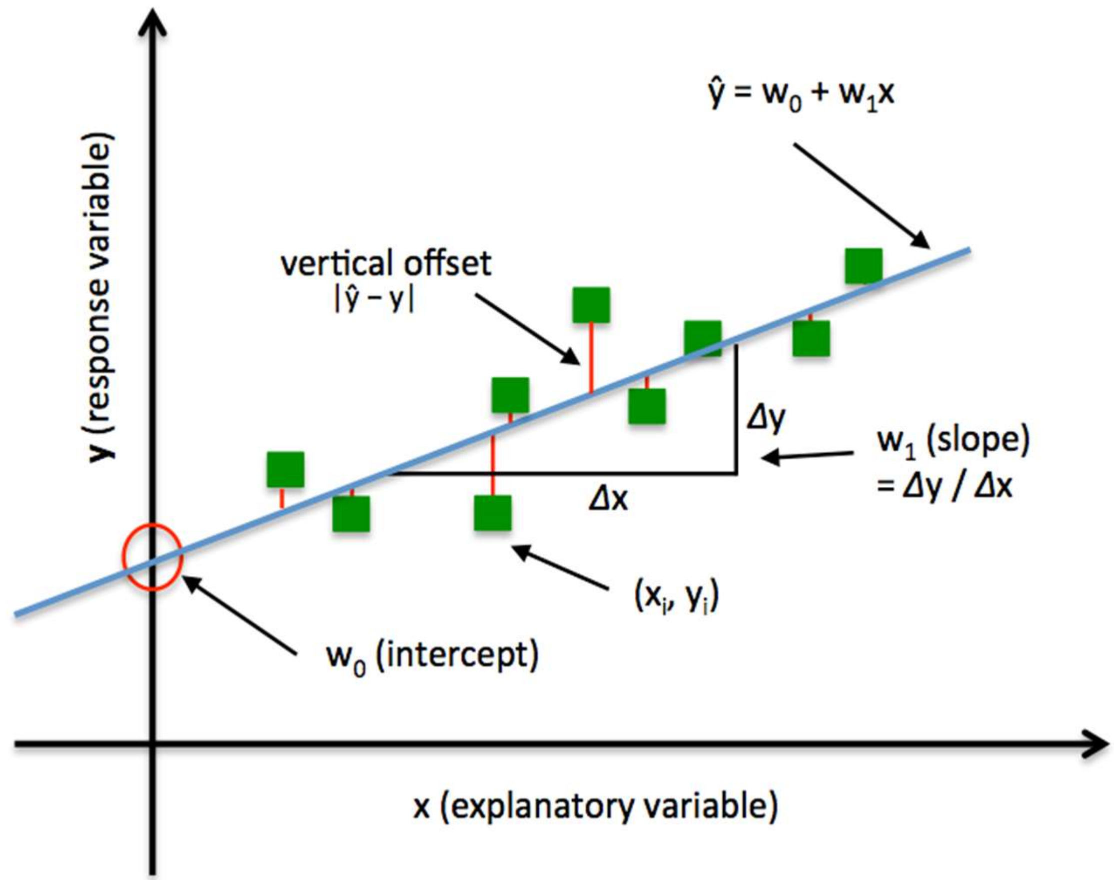


Solving the regression problem

- We basically want to find {w0, w1} that minimize deviations from the predictor line

$$\arg\min_{w_0,w_1} \sum_{i}^{n} (y_i - w_0 - w_1 x_i)^2$$

- How do we do it?
  - Iterate over all possible w values along the two dimensions?
  - Same, but smarter?
  - No, we can do this in *closed form* with just plain calculus
- Very few optimization problems in ML have closed form solutions
  - The ones that do are interesting for that reason



Parameter estimation via calculus

- We just need to set the partial derivatives to zero (full derivation)

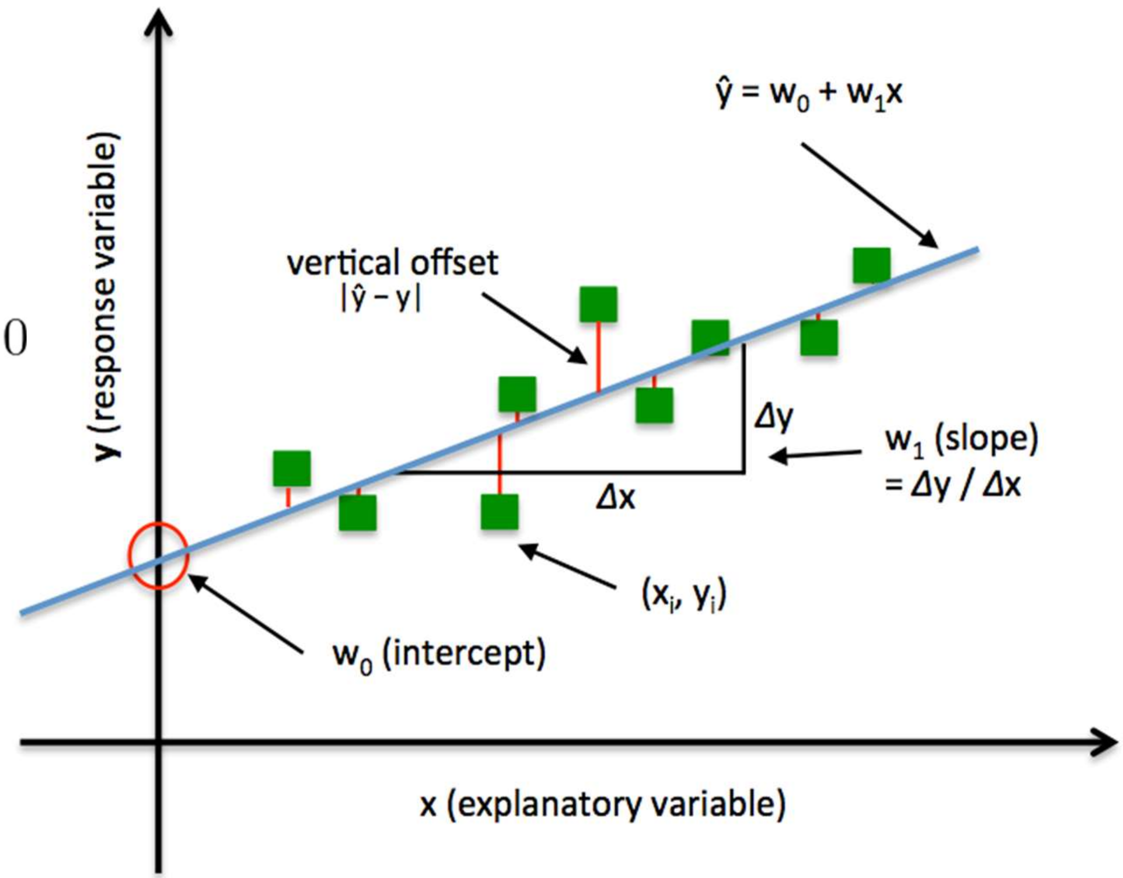
$$\frac{\partial \epsilon^2}{\partial w_0} = \sum_i^n -2(y_i - w_0 - w_1 x_i) = 0$$

$$\frac{\partial \epsilon^2}{\partial w_1} = \sum_i^n -2x_i(y_i - w_0 - w_1 x_i) = 0$$

- Simplifying

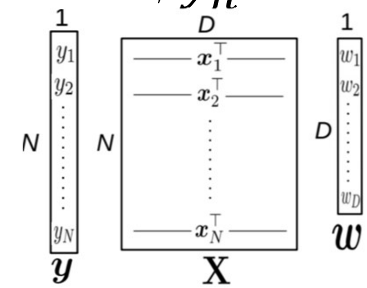
$$w_0 = \bar{y} - w_1 \bar{x}$$

$$w_1 = \frac{n \sum_i^n x_i y_i - \sum_i^n x_i \sum_i^n y_i}{n \sum_i^n x_i x_i - \sum_i^n x_i \sum_i^n x_i}$$



More generally

- Given: Training data with N input-output pairs $\{(\boldsymbol{x}_{\mathrm{n}}, y_{\mathrm{n}})\}_{n=1}^{N}$, $\boldsymbol{x}_n \in \mathbb{R}^D$, $y_n \in \mathbb{R}$



- Goal: Learn a model to predict the output for new test inputs
- Assume the function that approximates the I/O relationship to be a linear model

$$y_n \approx f(\boldsymbol{x}_n) = \boldsymbol{w}^\top \boldsymbol{x}_n \quad (n = 1,2,\ldots,N)$$

Can also write all of them compactly using matrix-vector notation as $\boldsymbol{y} \approx \boldsymbol{X}\boldsymbol{w}$

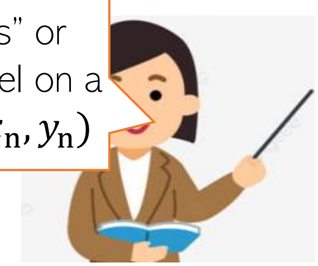
- Let's write the total error or "loss" of this model over the training data as

Goal of learning is to find the $\boldsymbol{w}$ that minimizes this loss + does well on test data

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

$\ell(y_n, \boldsymbol{w}^\top \boldsymbol{x}_n)$ measures the prediction error or "loss" or "deviation" of the model on a single training input $(\boldsymbol{x}_{\mathrm{n}}, y_{\mathrm{n}})$

Unlike models like KNN and DT, here we have an explicit problem-specific objective (loss function) that we wish to optimize for



Linear Regression with Squared Loss

- In this case, the loss func will be

In matrix-vector notation, can write it compactly as $||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}||_2^2 = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})^\top(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})$

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

- Let us find the $\boldsymbol{w}$ that optimizes (minimizes) the above squared loss
- We need calculus and optimization to do this!

The "least squares" (LS) problem Gauss-Legendre, 18th century)

- The LS problem can be solved easily and has a closed form solution

Link to a nice derivation

$$\boldsymbol{w}_{LS} = \text{arg min}_{\boldsymbol{w}}\, L(\boldsymbol{w}) = \text{arg min}_{\boldsymbol{w}} \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$$

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

$D \times D$ matrix inversion – can be expensive. Ways to handle this. Will see later

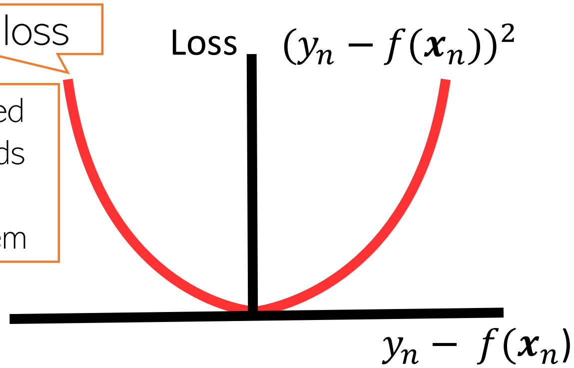
Alternative loss functions

- Many possible loss functions for regression problems

Choice of loss function usually depends on the nature of the data. Also, some loss functions result in easier optimization problem than others

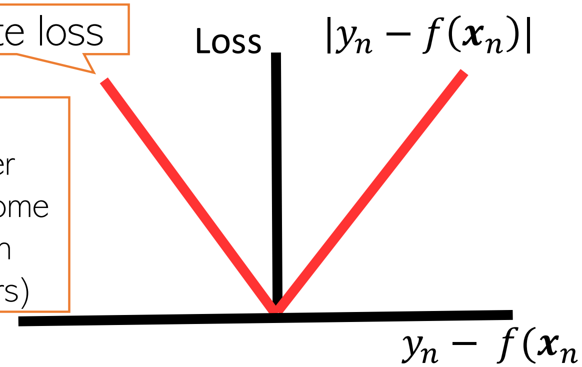
Squared loss

Very commonly used for regression. Leads to an easy-to-solve optimization problem



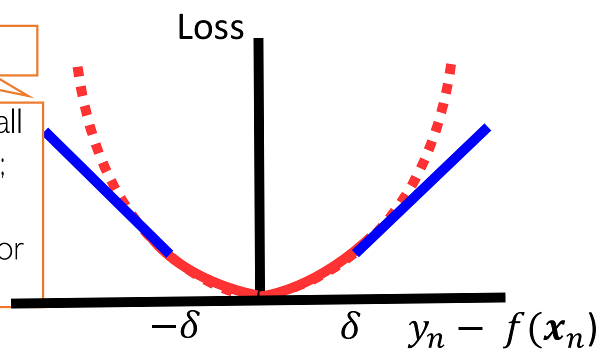
Absolute loss

Grows more slowly than squared loss. Thus better suited when data has some outliers (inputs on which model makes large errors)



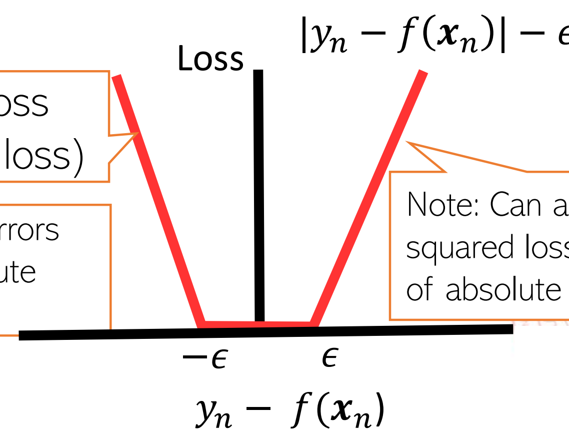
Huber loss

Squared loss for small errors (say up to δ); absolute loss for larger errors. Good for data with outliers



ϵ-insensitive loss (a.k.a. Vapnik loss)

Zero loss for small errors (say up to ϵ); absolute loss for larger errors



Note: Can also use squared loss instead of absolute loss

How do we ensure generalization?

- We minimized the objective $L(\boldsymbol{w}) = \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$ w.r.t. $\boldsymbol{w}$ and got

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

- Problem: The matrix $\boldsymbol{X}^\top \boldsymbol{X}$ may not be invertible
  - This may lead to non-unique solutions for $\boldsymbol{w}_{opt}$
- Problem: Overfitting since we only minimized loss defined on training data
  - Weights $\boldsymbol{w} = [w_1, w_2, \ldots, w_D]$ may become arbitrarily large to fit training data perfectly
  - Such weights may perform poorly on the test data however
- One Solution: Minimize a regularized objective $\boxed{L(\boldsymbol{w}) + \lambda R(\boldsymbol{w})}$
  - The reg. will prevent the elements of $\boldsymbol{w}$ from becoming too large
  - Reason: Now we are minimizing training error + magnitude of vector $\boldsymbol{w}$

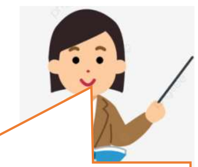
$R(\boldsymbol{w})$ is called the Regularizer and measures the "magnitude" of $\boldsymbol{w}$

$\lambda \geq 0$ is the reg. hyperparam. Controls how much we wish to regularize (needs to be tuned via cross-validation)

Regularized Least Squares (a.k.a. Ridge Regression)

- Recall that the regularized objective is of the form $L_{reg}(\boldsymbol{w}) = L(\boldsymbol{w}) + \lambda R(\boldsymbol{w})$
- One possible/popular regularizer: the squared Euclidean (ℓ_2 squared) norm of $\boldsymbol{w}$

$$R(\boldsymbol{w}) = ||\boldsymbol{w}||_2^2 = \boldsymbol{w}^\top \boldsymbol{w}$$



- With this regularizer, we have the regularized least squares problem as

Look at the form of the solution. We are adding a small value λ to the diagonals of the DxD matrix $\boldsymbol{X}^\top \boldsymbol{X}$ (like adding a ridge/mountain to some land)

Why is the method called "ridge" regression

$$\boldsymbol{w}_{ridge} = \arg\min_{\boldsymbol{w}} L(\boldsymbol{w}) + \lambda R(\boldsymbol{w})$$

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

- Proceeding just like the LS case, we can find the optimal $\boldsymbol{w}$ which is given by

$$\boxed{\boldsymbol{w}_{ridge} = \left(\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^\top + \lambda I_D\right)^{-1} \left(\sum_{n=1}^{N} y_n \boldsymbol{x}_n\right) = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda I_D)^{-1} \boldsymbol{X}^\top \boldsymbol{y}}$$

A closer look at ℓ_2 regularization

Remember – in general, weights with large magnitude are bad since they can cause overfitting on training data and may not work well on test data

- The regularized objective we minimized is

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

- Minimizing $L_{reg}(\boldsymbol{w})$ w.r.t. $\boldsymbol{w}$ gives a solution for $\boldsymbol{w}$ that
  - Keeps the training error small
  - Has a small ℓ_2 squared norm $\boldsymbol{w}^\top \boldsymbol{w} = \sum_{d=1}^{D} w_d^2$

Good because, consequently, the individual entries of the weight vector $\boldsymbol{w}$ are also prevented from becoming too large

- Small entries in $\boldsymbol{w}$ are good since they lead to "smooth" models

Not a "smooth" model since its test data predictions may change drastically even with small changes in some feature's value

$\boldsymbol{x}_n =$	1.2	0.5	2.4	0.3	0.8	0.1	0.9	2.1
$\boldsymbol{x}_m =$	1.2	0.5	2.4	0.3	$0.8+\epsilon$	0.1	0.9	2.1

Exact same feature vectors only differing in just one feature by a small amount

$y_n = 0.8$

$y_m = 100$

Very different outputs though (maybe one of these two training ex. is an outlier)

A typical $\boldsymbol{w}$ learned without ℓ_2 reg.

3.2	1.8	1.3	2.1	10000	2.5	3.1	0.1

Just to fit the training data where one of the inputs was possibly an outlier, this weight became too big. Such a weight vector will possibly do poorly on normal test inputs

Other Ways to Control Overfitting

Note that optimizing loss functions with such regularizers is usually harder than ridge reg. but several advanced techniques exist (we will see some of those later)

- Use a regularizer $R(\boldsymbol{w})$ defined by other norms, e.g.,

ℓ_1 norm regularizer

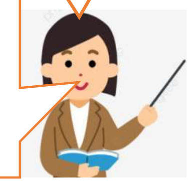
$$||\boldsymbol{w}||_1 = \sum_{d=1}^{D} |w_d|$$

When should I used these regularizers instead of the ℓ_2 regularizer?

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

ℓ_0 norm regularizer (counts number of nonzeros in $\boldsymbol{w}$

Use them if you have a very large number of features but many irrelevant features. These regularizers can help in automatic feature selection



Using such regularizers gives a sparse weight vector $\boldsymbol{w}$ as solution

sparse means many entries in $\boldsymbol{w}$ will be zero or near zero. Thus those features will be considered irrelevant by the model and will not influence prediction

- Use non-regularization based approaches
  - Early-stopping (stopping training just when we have a decent val. set accuracy)
  - Dropout (in each iteration, don't update some of the weights)
  - Injecting noise in the inputs

All of these are very popular ways to control overfitting in deep learning models. More on these later when we talk about deep learning

Linear Regression as Solving System of Linear Eqs

- The form of the lin. reg. model $\boldsymbol{y} \approx \boldsymbol{X}\boldsymbol{w}$ is akin to a system of linear equation
- Assuming N training examples with D features each, we have

First training example: $y_1 = x_{11}w_1 + x_{12}w_2 + \ldots + x_{1D}w_D$

Second training example: $y_2 = x_{21}w_1 + x_{22}w_2 + \ldots + x_{2D}w_D$

$\vdots$

N-th training example: $y_N = x_{N1}w_1 + x_{N2}w_2 + \ldots + x_{ND}w_D$

Note: Here x_{nd} denotes the d^{th} feature of the n^{th} training example

N equations and D unknowns here $(w_1, w_2, \ldots, w_D)$

- However, in regression, we rarely have $N = D$ but rather $N > D$ or $N < D$
  - Thus we have an underdetermined $(N < D)$ or overdetermined $(N > D)$ system
  - Methods to solve over/underdetermined systems can be used for lin-reg as well
  - Many of these methods don't require expensive matrix inversion

Solving lin-reg as system of lin eq.

$$\boldsymbol{w} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y} \quad \Rightarrow \quad \boldsymbol{A}\boldsymbol{w} = \boldsymbol{b} \text{ where } \boldsymbol{A} = \boldsymbol{X}^\top \boldsymbol{X}, \text{ and } \boldsymbol{b} = \boldsymbol{X}^\top \boldsymbol{y}$$

Now solve this!

System of lin. Eqns with D equations and D unknowns