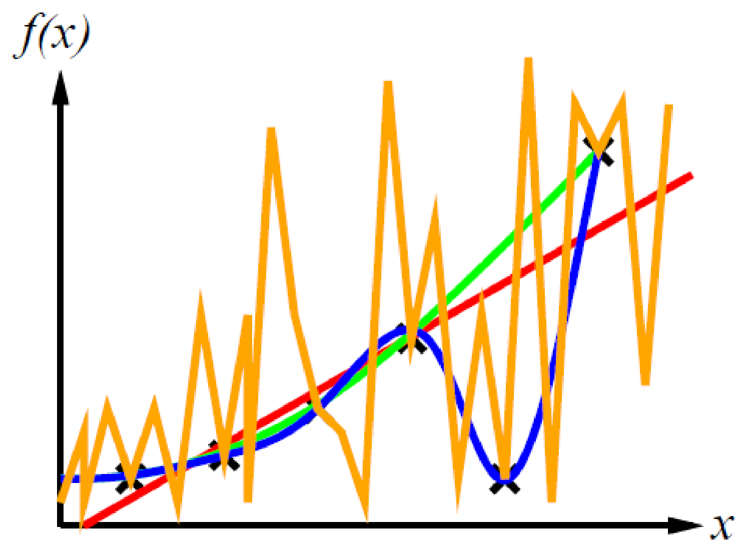
Overfitting and Regularization

A Simple Learning Task

- We can come up with an infinite number of possible functions here.
- How do we choose which one is best?
- This is called the **model selection problem**: out of an infinite number of possible **models** for our data, we must choose one.
- An easier version of the model selection problem: given a model (i.e., a function modeling our data), how can we measure how good this model is?

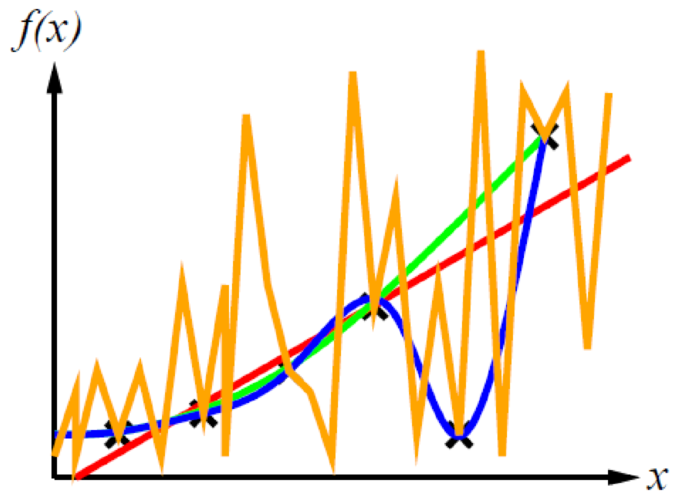


Source. S. Russell and P. Norvig, "Artificial Intelligence: A Modern Approach".

A Simple Learning Task

- One naïve solution is to evaluate functions based on **training error**.
- For any function F, its training error can be measured as a sum of squared errors over training patterns x_n:

$$\sum_n (t_n - F(x_n))^2$$

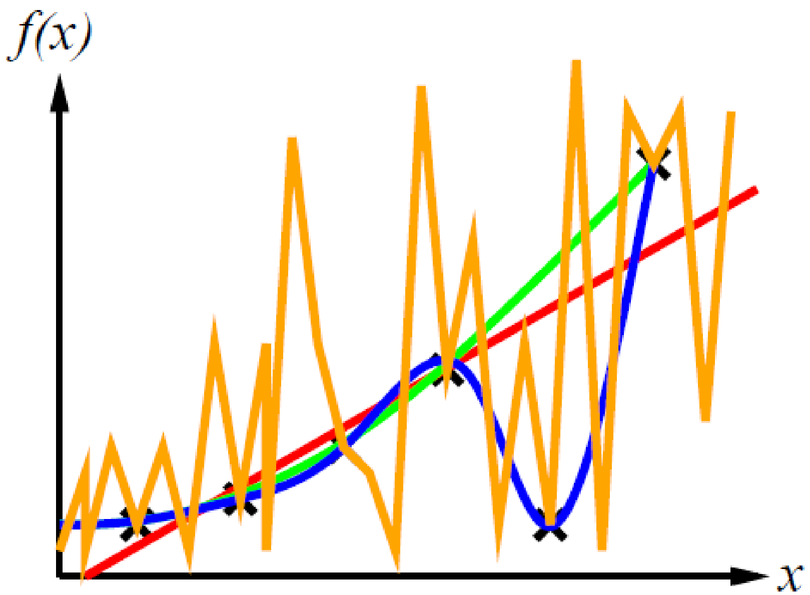


- What are the pitfalls of choosing the “best” function based on training error?
- The zig-zagging orange function comes out as “perfect”: its training error is zero.
- As a human, would you find more reasonable the orange function or the blue function (cubic polynomial)?
  - They both have zero training error.
  - However, the zig-zagging function looks pretty arbitrary.

A Simple Learning Task

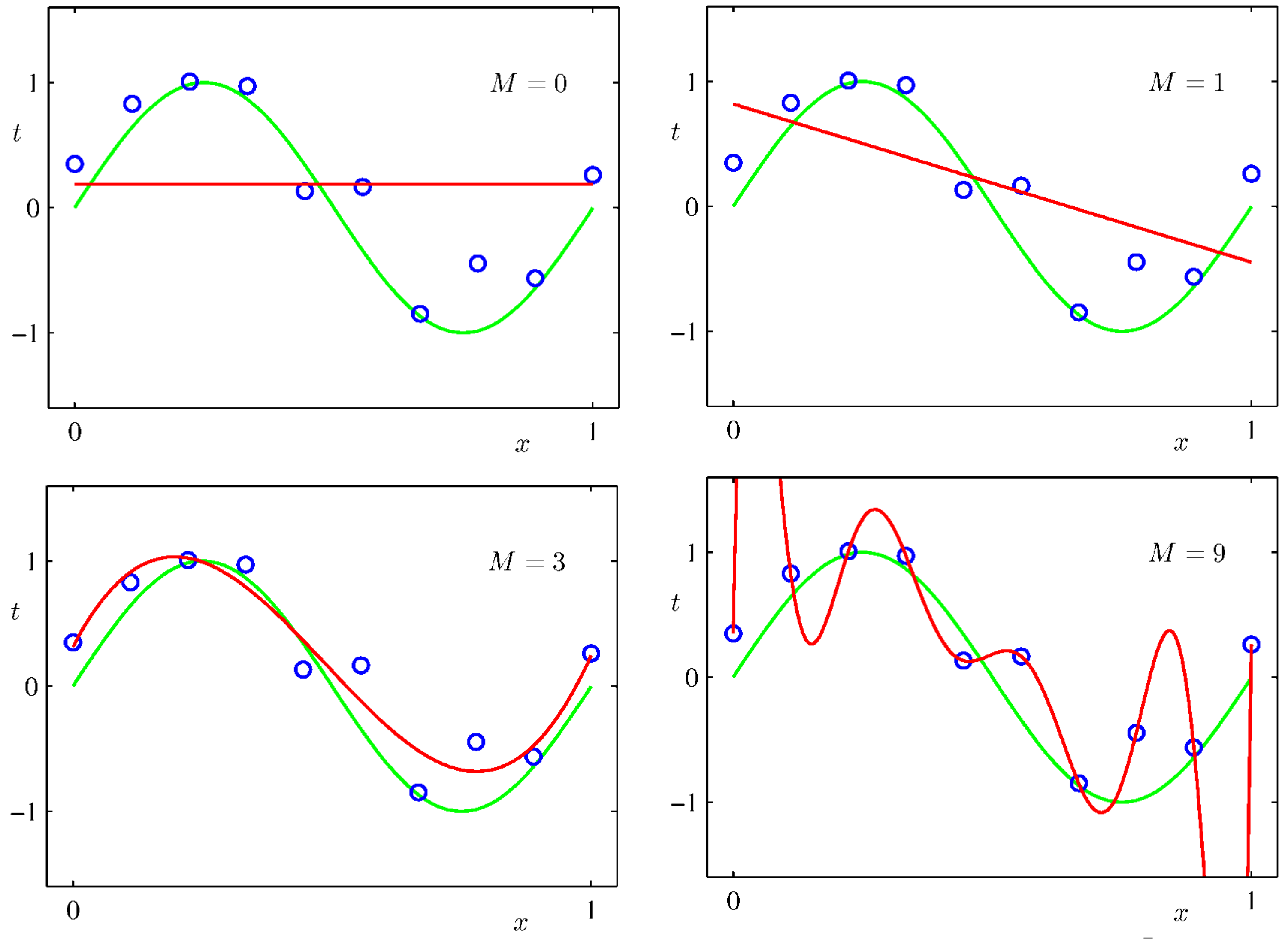
- Ockham's razor: given two equally good explanations, choose the more simple one.
  - This is an old philosophical principle (Ockham lived in the 14th century).
- Based on that, we prefer a cubic polynomial over a crazy zig-zagging function, because it is more simple, and they both have zero training error.

- What if none of the functions have zero training error?
- How do we weigh simplicity versus training error?
- There is no standard or straightforward solution to this.
- There exist many machine learning algorithms. Each corresponds to a different approach for resolving the trade-off between simplicity and training error



Polynomial Fitting

- One common approach is to try to model the function as a polynomial.
- We estimate the parameters of the polynomial based on the training data.
- Here are estimated polynomials of degrees 0, 1, 3, 9.
- Notice the overfitting problem with the 9th degree polynomial

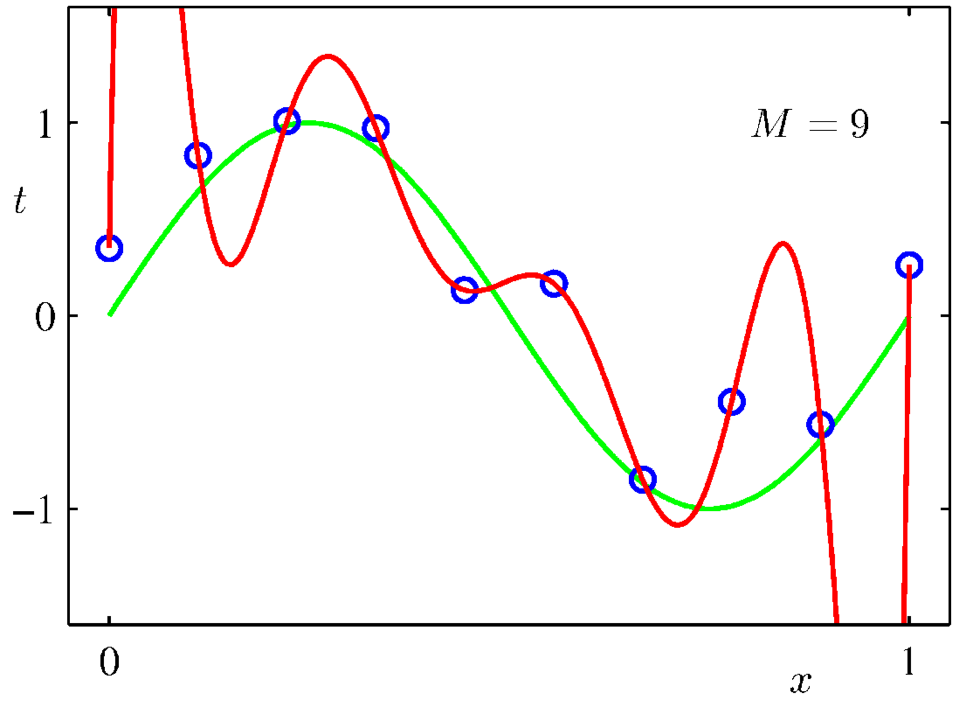


Overfitting

- Overfitting means that the learned function fits very well (or perfectly) the training data, but works very poorly on test data.

- Overfitting is a **huge** problem in machine learning

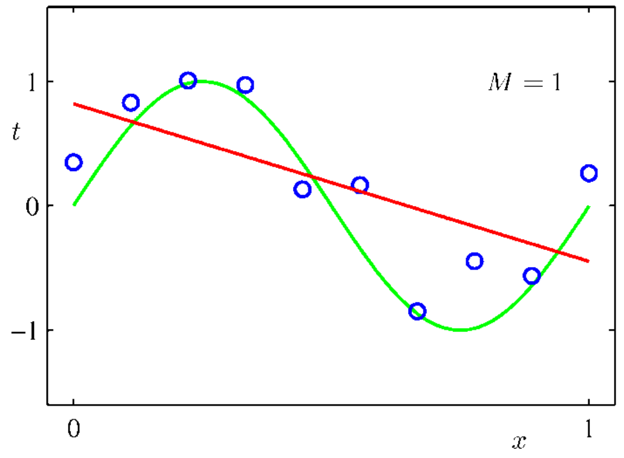
- Some times, when our models have too many parameters (like a 9th degree polynomial), those parameters get tuned to match the noise in the data.

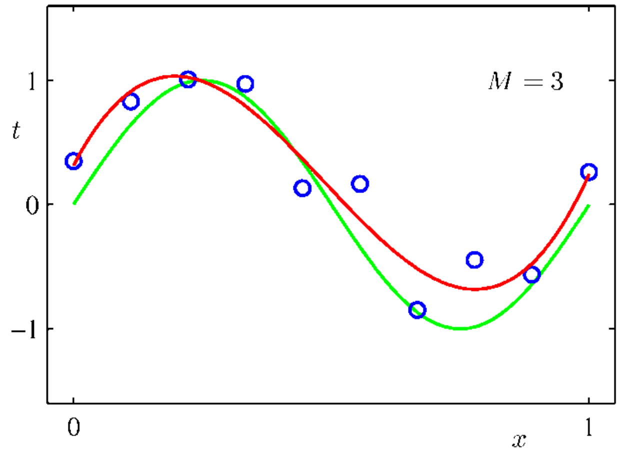


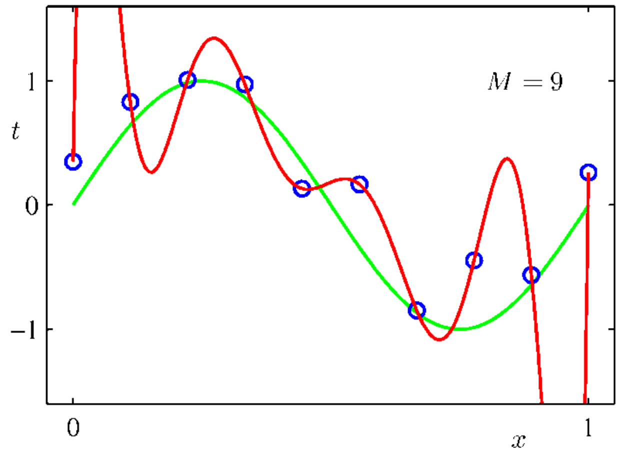
Regularization

- These are the parameters for some estimated polynomials.
- Overfitting leads to very large magnitudes of parameters.

	Degree 1	Degree 3	Degree 9
w_0	0.82	0.31	0.35
w_1	-1.27	7.99	232.37
w_2		-25.43	-5321.83
w_3		17.37	48568.31
w_4			-231639.30
w_5			640042.26
w_6			-1061800.52
w_7			1042400.18
w_8			-557682.99
w_9			125201.43







Regularization

- If we are confident that large magnitudes of polynomial parameters are due to overfitting, we can penalize them in the error function:

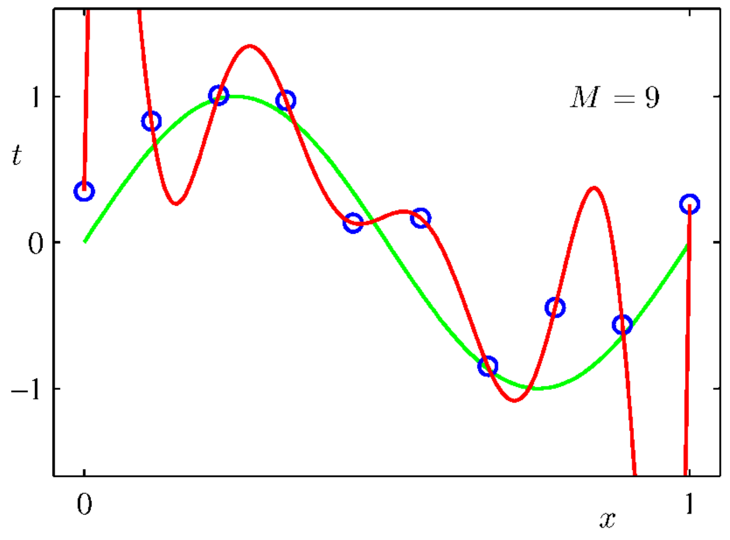
$$\left(\sum_n (t_n - F(x_n))^2\right) + \lambda ||w||^2$$

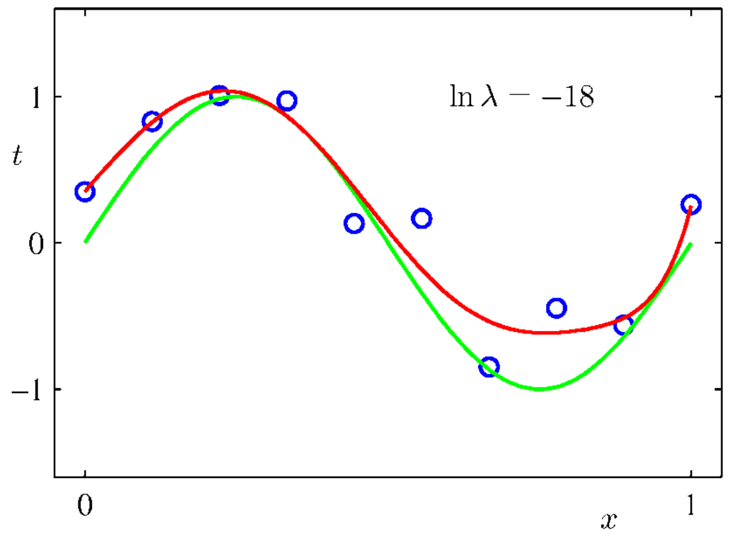
- The blue part is the sum-of-squares error that we saw before.
- The red part is what is called a **regularization term**.
- $||w||^2$ is the sum of squares of the parameters w_i.
- λ is a parameter that you have to specify.
    - It controls how much you penalize large $||w||^2$ values.

Regularization

- A small λ solves the overfitting problem in this case.

	$\lambda = 0$	$\lambda = e^{-18}$
w_0	0.35	0.35
w_1	232.37	4.74
w_2	-5321.83	-0.77
w_3	48568.31	-31.97
w_4	-231639.30	-3.89
w_5	640042.26	55.28
w_6	-1061800.52	41.32
w_7	1042400.18	-45.95
w_8	-557682.99	-91.53
w_9	125201.43	72.68



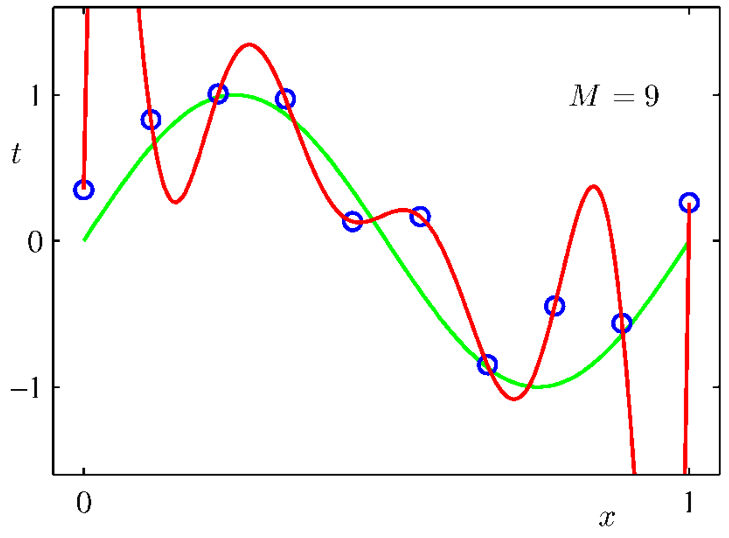


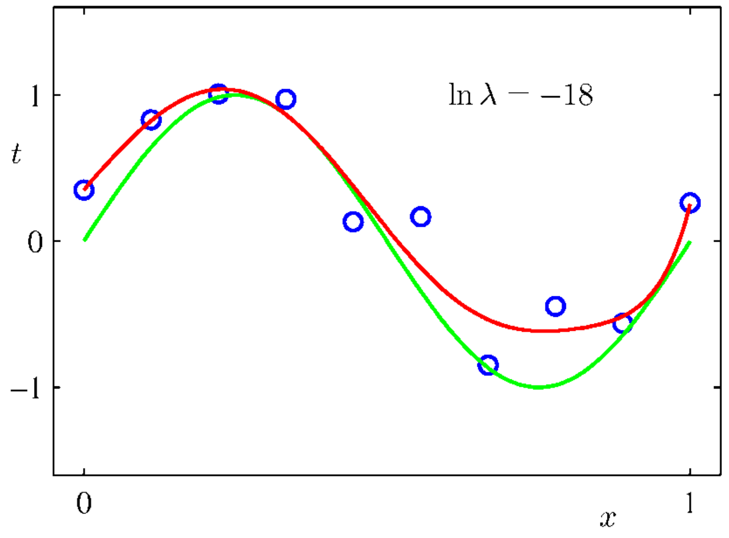
Hyperparameters

- Parameter *λ* in the previous example is an example of what we call a **hyperparameter**.
    - The degree M of the polynomial is also a hyperparameter.
- A hyperparameter is a parameter that we (the humans) need to choose, as opposed to parameters that our machine learning algorithm is supposed to learn.
    - Parameter *λ,* learning rate, batch size, epochs, network architecture, etc
- To find good hyperparameter values:
    - We typically try many different values, and we see which one works best.
    - We typically try how well these values work on a **validation set**
- Hyperparameter Optimization Techniques
    - Manual Search
    - Random Search
    - Grid Search
    - Bayesian Optimization
    - Genetic Algorithms

Using a Validation Set

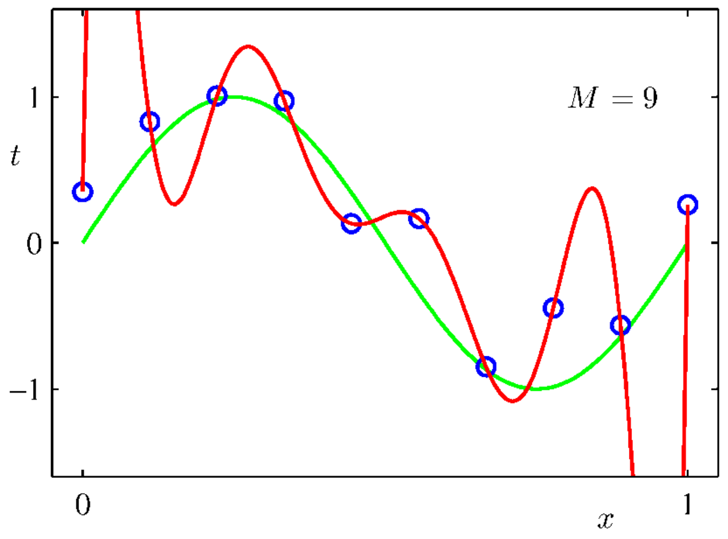
- How can we choose a good value for λ?
- A standard approach is to use a **validation set**.
    - Like the training set, the validation set is a set of example inputs and associated outputs.
    - However, the objects in the validation set should **not** appear in the training set.
- We use the training set to fit polynomials using many different values for λ.
- We choose the λ that gives best results on the validation set.

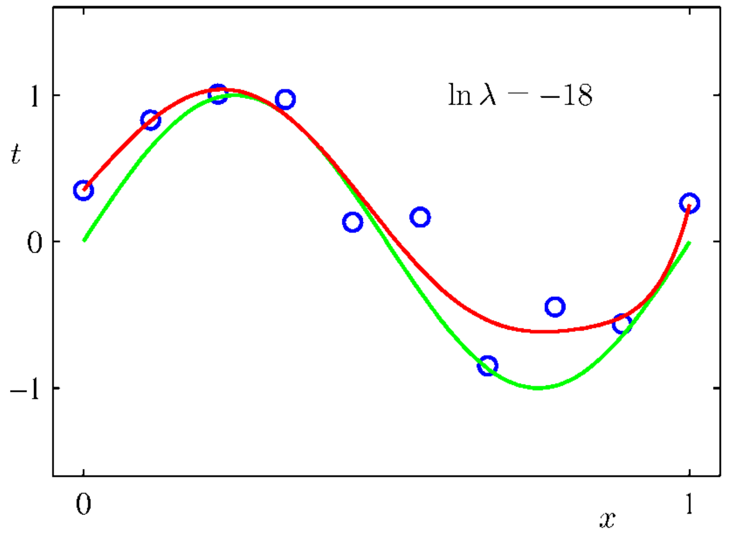




Using a Validation Set

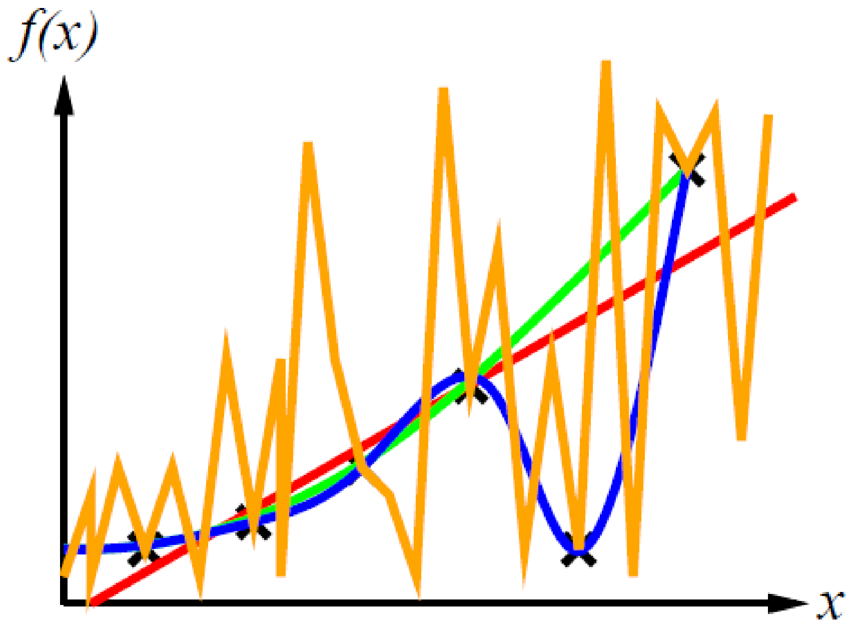
- Strictly speaking, choosing a good value for λ is part of the training task.
- Often times, we have a general method for solving a problem, which requires that we choose some hyperparameters.
- Typically, the training set is used to solve our problem multiple times, with different choices of those hyperparameters.
- The validation set is used to decide which choice of hyperparameters works best.





Using a Test Set

- If we want to evaluate one or more methods, to see how well they work, we use a **test set**.
- Test examples should **not** appear either in the training set or in the validation set.
- Error rates on the test set are a reliable estimate of how well a function generalizes to data outside training.
  - Error rates on the training set are **not** reliable for that task.
  - Error rates on the validation set are still not quite reliable, as the validation set was used to choose some parameters.



K-fold Cross Validation

- One of the most popular data partitioning strategies widely used in order to build a more generalized model.
- The whole dataset is *randomly* split into *independent* **k-folds** *without replacement*.
- **k-1 folds** are used for the model training and one fold is used for performance evaluation.
- This procedure is repeated **k times** (iterations) so that we obtain **k number** of performance estimates (e.g. MSE) for each iteration.
- Then we get the mean of **k number** of performance estimates
- can be used for evaluating a model's performance
- can be used for hyperparameter tuning

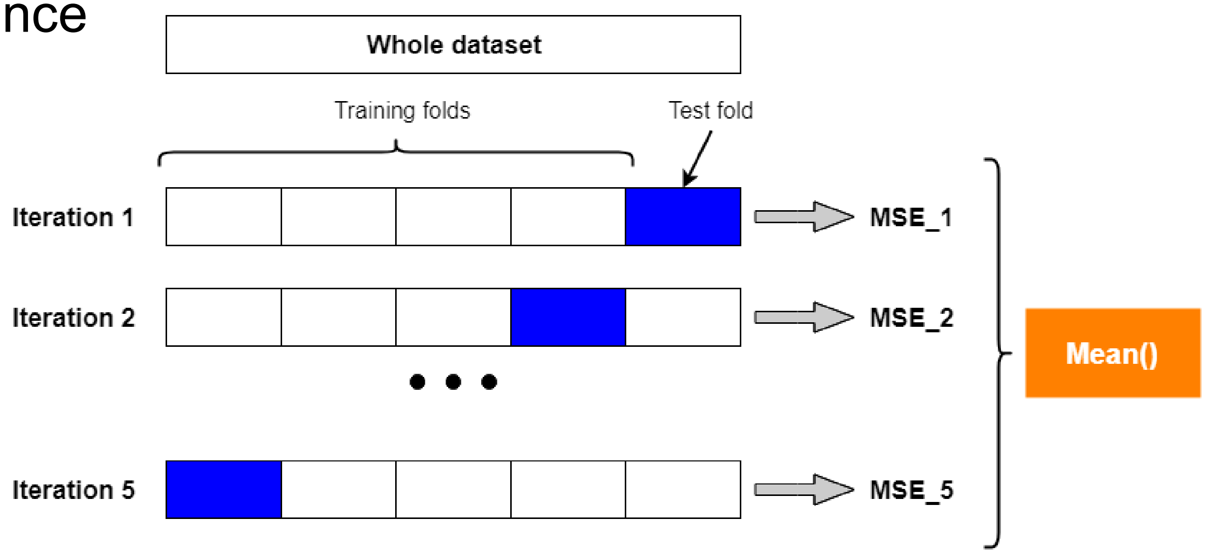


Image copyright: Rukshan Manorathna

Recap: Training, Validation, Test Sets

- Training set: use to learn the function that we want to learn, that maps inputs to outputs.
- Validation set: use to evaluate different values of hyperparameters (like λ for regularization) that need to be hardcoded during training.
  - Train with different values, and then see how well each resulting function works on the validation set.
- Test set: use to evaluate the final product (after the choice of hyperparameters has been finalized).

Overfitting

Overfitting

- Because a neural network with even 1 hidden layer is a universal approximator, it can fit any function. (Universal Approximation Theorem)
- However, training observations are not made with complete fidelity, and therefore have sampling error.
- Because of this error, a model that can fit any function will not only fit the true signal, but will also conform to the error.
- We therefore need a way to prevent a neural network from overfitting the training data.

Overfitting

- Every machine learning has the issue of overfitting problem

- Occam’s razor: prefer simpler model.

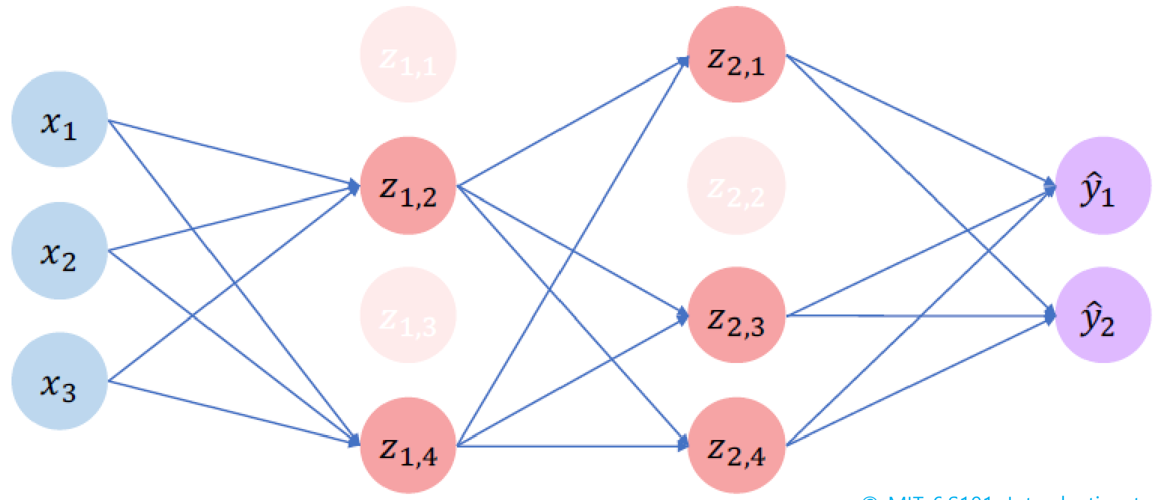
- In decision tree, we use pruning technique to avoid overfitting problem
  - makes the decision tree simpler

- Overfitting is a serious problem in deep learning since it has many layers with too many parameters

- Methods used in Deep Learning (or neural network) to avoid overfitting
  1) Regularization
  2) Dropout
  3) Data Augmentation
  4) Early Stop
  5) Transfer learning/Domain Adaptation
  6) etc

Dropout

- During training, randomly set some activations to 0 during training
- Each unit retained with fixed probability p, independent of other units
- Hyper-parameter p to be chosen (tuned)
- Typically drop 50% of activations in layer
- Forces network to not rely on any 1 node
- Select different dropout nodes at each mini-batch
- May use different p value in layerwise



Early Stopping

- Another approach is to stop model fitting prior to overfitting.

- That is, keep training the model as long as its validation error is decreasing, but stop training as soon as validation error starts increasing.

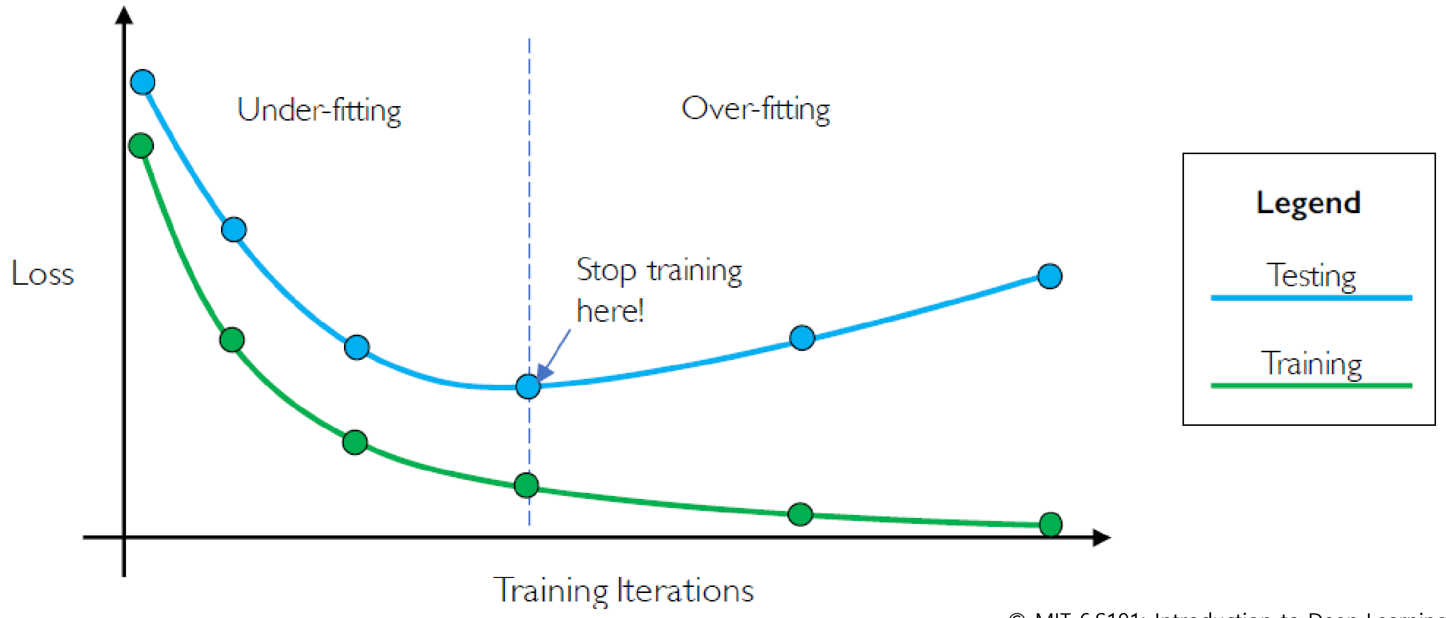
- Because a neural network with at least 1 hidden layer is universal approximator, with a large number of hidden units it is inevitable that the model will begin to overfit the data.

- Therefore, it would be beneficial to stop the training process at a certain stage, such that validation (proxy for test) error is minimized.

- An iteration is known as an epoch, and want to stop at optimal epoch.

Early Stopping

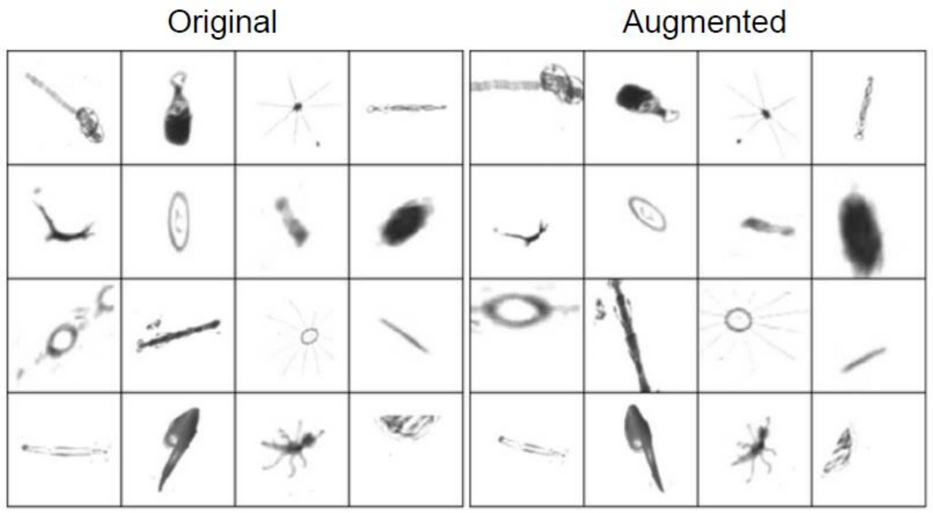
- Use validation error to decide when to stop training
- Stop training before we have a chance to overfit
- Stop when monitored quantity has not improved after n subsequent epochs
- n is called patience
- generally this is not recommended by some people



Data Augmentation

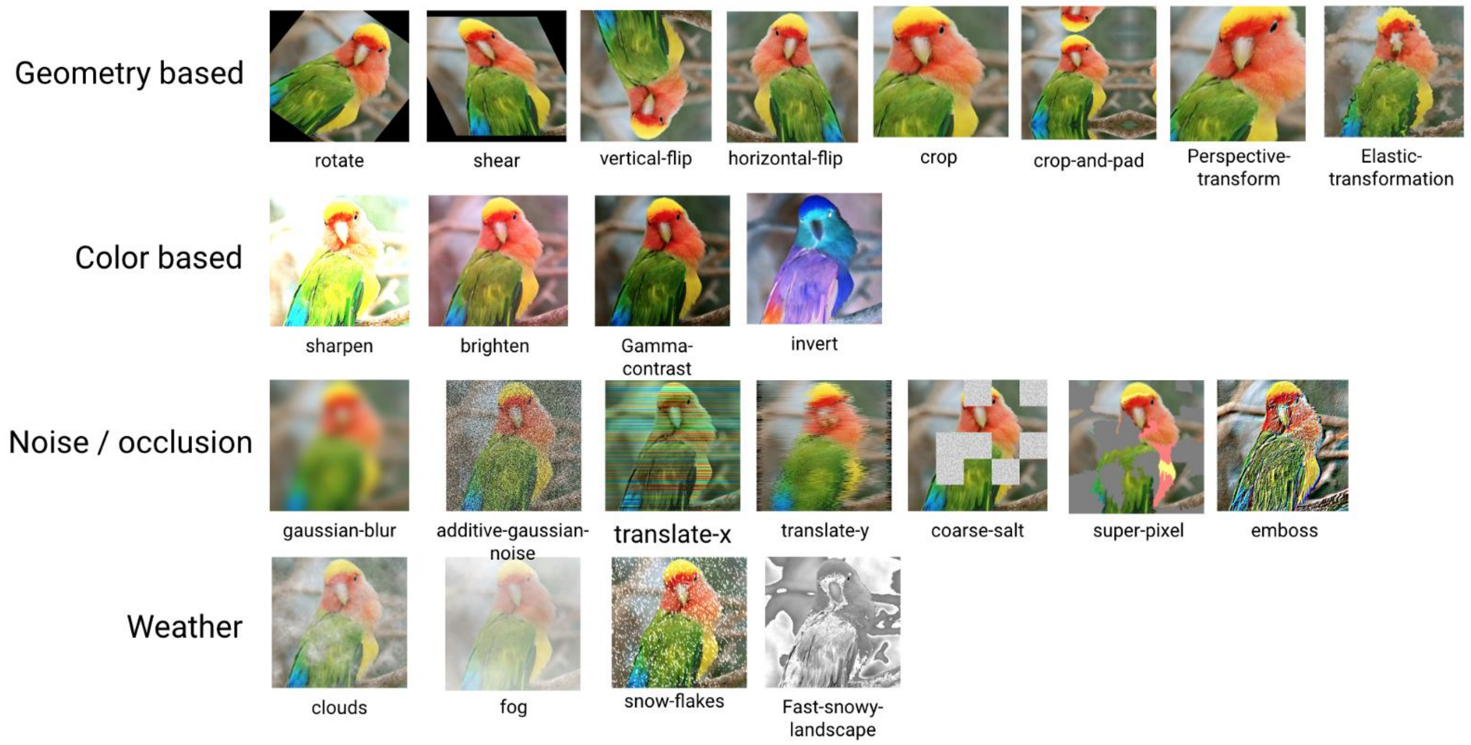
- In practice, the amount of data we have is limited.
- One way to get around this problem is to create fake data and add it to the training set.
- Dataset augmentation has been a particularly effective technique for a specific classification problem: object recognition.
- Images are high dimensional and include an enormous variety of factors of variation, many of which can be easily simulated.
- Operations like translating the training images a few pixels in each direction can often greatly improve generalization

Data Augmentation



Pre-processed images (left) and augmented versions of the same images (right).

Source: Ian Goodfellow et al. Deep Learning, MIT press, 2016



https://blog.insightdatascience.com/automl-for-data-augmentation-e87cf692c366?gi=bc0b8b5a353a

Regularization

Regularization

- The most widely used method against overfitting in machine learning
- Regularization: Machine learning technique that constrains our optimization problem to discourage complex models
- Occam’s razor
- Improve generalization of our model on unseen data
- Reduce the complexity of model
- How to measure the complexity of model
  - VC dimension (why not use it?), number of parameters,
- Very important topic in machine learning

Regularization

- Idea: A new regularization term (regularizer) is added to loss (objective) function
- Loss function with regularization term is modified as follows
  - $J(w)$: loss function, λ : regularization constant
  - $R(w)$: regularization term (aka regularizer). Can be regarded as the complexity of model

$$J_{reg}(w) = J(w) + \lambda * R(w)$$

- Gradient descent method minimizes $J_{reg}(w)$ instead of $J(w)$
- As the value of λ rises, it reduces the value of w's and thus reducing the variance
- Goal of regularization term ($R(w)$) is to make model parameters simpler, specifically
  - reduces the number of parameters (L1/Lasso regularization) or
  - makes the values of parameters smaller (L2/Ridge regularization)

Regularization

- The complexity of models is often measured by the size of the model w viewed as a vector.
- Initially your model is very bad and most of the loss comes from the error terms, so the model is adjusted to primarily to reduce the error term.
- As the model is improving and the model vector is growing the regularization term becomes a more significant part of the loss. Regularization prevents the model vector growing arbitrarily for negligible reductions in the error.

Common Regularizers

- From $J_{reg}(w) = J(w) + \lambda * R(w)$

L1 (Lasso) Regularization

- Regularizer, $R(w)$, is given as the sum of weights

$$\boxed{R(w) = \sum_i |w_i|}$$

$$J_{reg}(w) = J(w) + \lambda \sum_i |w_i|$$

- Forces less important parameters be zero
- It favors concentrating the size of the model in only a few components
- Reduces the number of features automatically (feature selection)
- Regularization constant (λ, weight decay value) determines how dominant regularization is during gradient computation
- Regularizer is not differentiable
- Generate multiple solutions
- Also known as Lasso regularization

Common Regularizers

L2 (Ridge) Regularization

- L2 (Ridge) regularization is given as the sum of weight squares $\boxed{R(w) = \sum_i w_i^2}$

$$J_{reg}(w) = J(w) + \lambda \sum_i w_i^2$$

- Regularization term penalizes big weights
- Big weight decay coefficient → big penalty to big weights
- Squared weights penalizes large values more
- Generate a unique solution
- It has the side effect of spreading weight more evenly between the components of the model vector.
- Generally L2 tends to be preferable for low dimensional models while lasso tends to work better for high dimensional models like text classification where it leads to sparse models, ie models with few non-zero parameters.

Common Regularizers

Elastic Regularization

- Combination of L1 and L2 regularization (L1L2 regularization)

$$J_{reg}(w) = J(w) + \lambda_1 \textstyle\sum_i |w_i| + \lambda_2 \sum_i w_i^2 \quad \text{or}$$

$$J_{reg}(w) = J(w) + \lambda(\alpha \sum_i |w_i| + (1-\alpha) \sum_i w_i^2)$$

- Show better results in some cases
- Computationally expensive
- Not proper for small size data

Gradient Descent with Regularization

- Basic training method in deep learning

- Pick a starting point (w)
- Repeat until loss doesn't decrease in all dimensions:
  - pick a dimension
  - update w using the derivative

$$w_i = w_i - \eta \frac{\partial}{\partial w_i}(J(w) + \frac{\lambda}{2}\sum_i w_i^2) \qquad \text{(L2 regularization)}$$

  - for computational convenience, $\frac{\lambda}{2}$ is used instead of λ

- Update rule with L1: $w_i = w_i - \eta(\frac{\partial}{\partial w_i}J(w) + \lambda\, sign(w_i))$

- Update rule with L2: $w_i = w_i - \eta(\frac{\partial}{\partial w_i}J(w) + \lambda w_i)$

(*) Visualization of L1/L2 Regularization

L1 Regularization

(1) min $J(w) + \lambda \sum_i |w_i|$

(2) min $J(w)$ subject to $\sum_i |w_i| < s$

- formula (2) is equivalent formula (1): Formula (1) is Lagrangian form of (2) (refer to Constrained Optimization Method in SVM)
- s is a constant that exists for each value of shrinkage factor λ
- solution happens when $\sum_i |w_i| = s$ (KKT condition)

e.g.: min $J(w)$ subject to $|w_1| + |w_2| < s$

- Solution happens when $|w_1| + |w_2| = s$
- Thus L1 makes some parameters zero (refer to Figure where w_1 is zero)

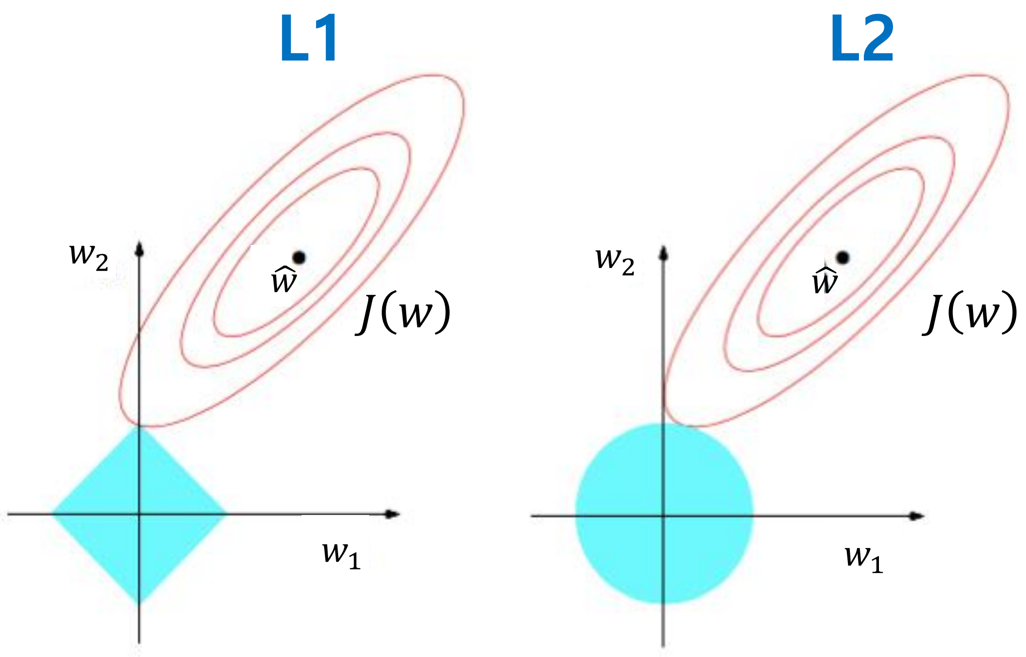


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions $|\beta_1| + |\beta_2| \le t$ and $\beta_1^2 + \beta_2^2 \le t^2$, respectively, while the red ellipses are the contours of the least squares error function.*

Elements of Statistical Learning by Hastie, Tibshirani, and Friedman

(*) Visualization of L1/L2 Regularization

L2 Regularization

(1) min $J(w) + \lambda \sum_i w_i^2$

(2) min $J(w)$ subject to $\sum_i w_i^2 < s$

- Similarly with L1, formula (2) is equivalent to (1) since (1) is Lagrangian form of (2)

e.g.: min $J(w)$ subject to $w_1^2 + w_2^2 < s$

- Solution happens when $w_1^2 + w_2^2 = s$
- L2 makes parameters small, not zero (refer to Figure)

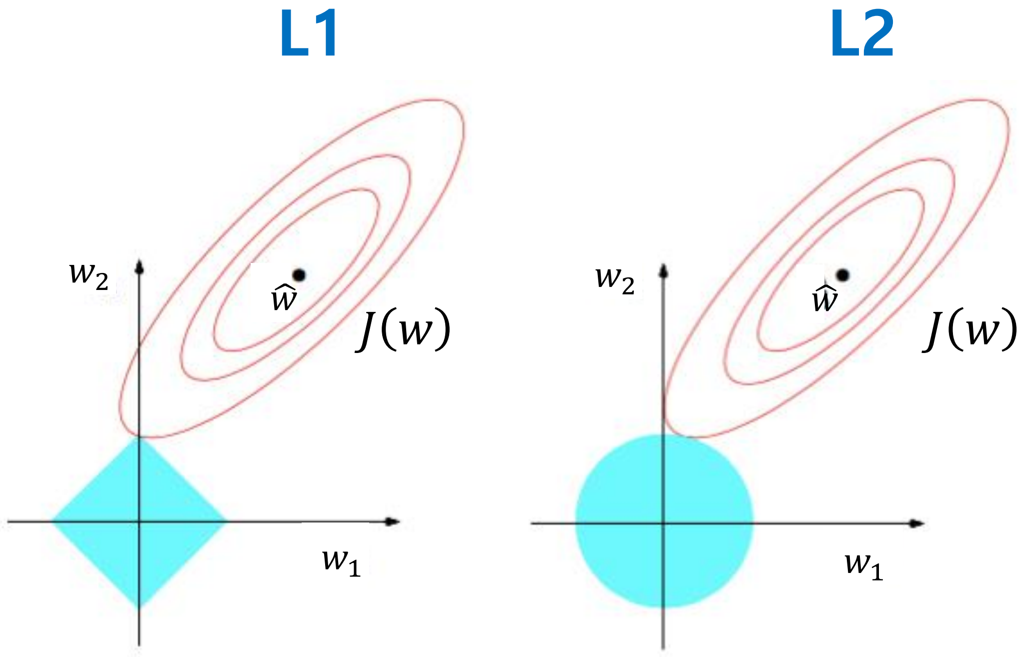


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions $|\beta_1| + |\beta_2| \leq t$ and $\beta_1^2 + \beta_2^2 \leq t^2$, respectively, while the red ellipses are the contours of the least squares error function.*

Elements of Statistical Learning by Hastie, Tibshirani, and Friedman