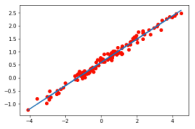
Training Issues

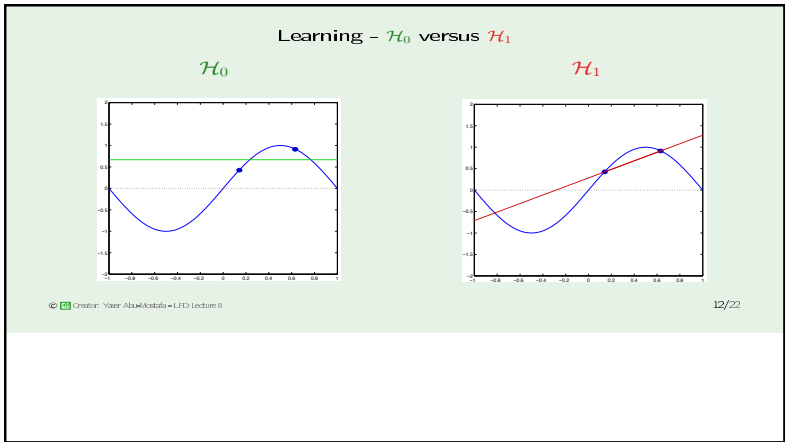
(1. Overfitting Bias_Var)
(2. Optimization)

Bias-Var

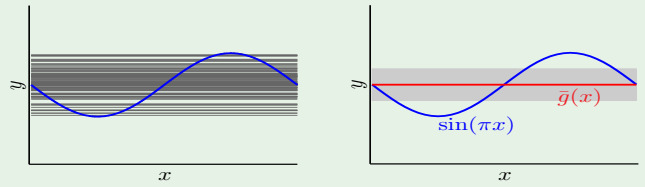
```
In [ ]: import numpy as np
        import tensorflow as tf
        import matplotlib.pyplot as plt
        %matplotlib inline

In [ ]: #1 Data for Training

        m_data = 123
        x_data = np.random.normal(0.0, 2.0, m_data)
        y_data = x_data * 0.45 + 0.6 + np.random.normal(0.0, 0.1, m_data)

        plt.scatter(x_data, y_data)
        plt.show()

In [ ]: #2 Model
        w = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
        b = tf.Variable(tf.zeros([1]))

        x = tf.placeholder(tf.float32, m_data) # check by "tab" or "shift+tab"
        y = w * x + b

In [ ]: #3 Error function (loss/cost ...)
        loss = tf.reduce_mean(tf.square(y - y_data))

        sess = tf.Session()
        sess.run(init)

In [ ]: #4 Optimization Scheme (or training scheme)

        train = tf.train.GradientDescentOptimizer(0.01).minimize(loss)
        init = tf.global_variables_initializer()

In [ ]: #5 Training

        sess = tf.Session()
        sess.run(init)

        for epoch in range(200):
            sess.run(train, feed_dict={x:x_data})
            if epoch % 50 == 0:
                print(epoch, sess.run(w), sess.run(b))

In [ ]: #6 Evaluation (Checking the ML results)

        plt.plot(x_data, y_data, 'ro')
        plt.plot(x_data, sess.run(w) * x_data + sess.run(b))
        plt.show()
```

```
In [19]: #5 Training

         sess = tf.Session()
         sess.run(init)

         for epoch in range(200):
             sess.run(train, feed_dict={x:x_data})
             if epoch % 50 == 0:
                 print(epoch, sess.run(w), sess.run(b))

(0, array([-0.63001943], dtype=float32), array([ 0.01816939], dtype=float32))
(50, array([ 0.42650929], dtype=float32), array([ 0.4286786], dtype=float32))
(100, array([ 0.45263666], dtype=float32), array([ 0.53892988], dtype=float32))
(150, array([ 0.4492867], dtype=float32), array([ 0.57882512], dtype=float32))

In [20]: #6 Evaluation (Checking the ML results)

         plt.plot(x_data, y_data, 'ro')
         plt.plot(x_data, sess.run(w) * x_data + sess.run(b))
         plt.show()
```

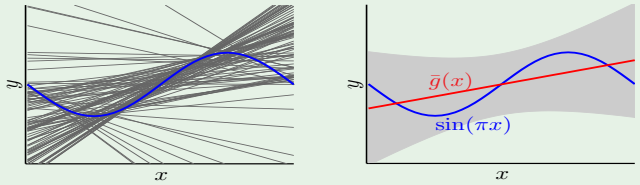




Bias and variance - $\mathcal{H}_0$

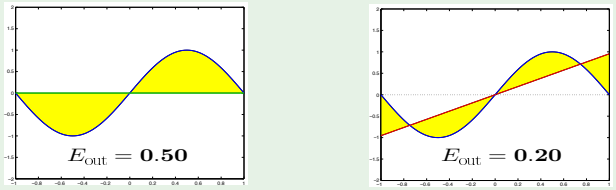


Bias and variance - $\mathcal{H}_1$



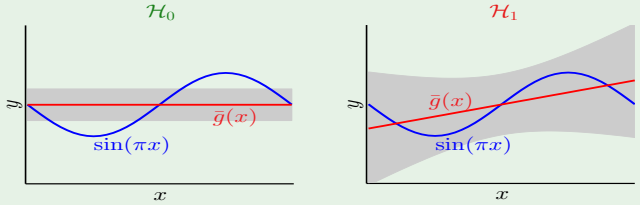
Approximation - $\mathcal{H}_0$ versus $\mathcal{H}_1$

$\mathcal{H}_0$ $\mathcal{H}_1$



and the winner is ...

$\mathcal{H}_0$ $\mathcal{H}_1$



bias = $\mathbf{0.50}$ var = $\mathbf{0.25}$ bias = $\mathbf{0.21}$ var = $\mathbf{1.69}$

Bias and variance

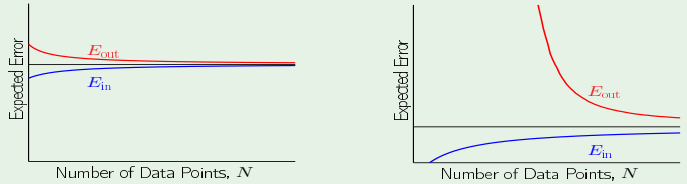
$$\mathbb{E}_{\mathcal{D}}\left[\left(g^{(\mathcal{D})}(\mathbf{x}) - f(\mathbf{x})\right)^2\right] = \underbrace{\mathbb{E}_{\mathcal{D}}\left[\left(g^{(\mathcal{D})}(\mathbf{x}) - \bar{g}(\mathbf{x})\right)^2\right]}_{\text{var}(\mathbf{x})} + \underbrace{\left(\bar{g}(\mathbf{x}) - f(\mathbf{x})\right)^2}_{\text{bias}(\mathbf{x})}$$

Therefore, $\mathbb{E}_{\mathcal{D}}\left[E_{\text{out}}(g^{(\mathcal{D})})\right] = \mathbb{E}_{\mathbf{x}}\left[\mathbb{E}_{\mathcal{D}}\left[\left(g^{(\mathcal{D})}(\mathbf{x}) - f(\mathbf{x})\right)^2\right]\right]$

$= \mathbb{E}_{\mathbf{x}}[\text{bias}(\mathbf{x}) + \text{var}(\mathbf{x})]$

$= \text{bias} + \text{var}$

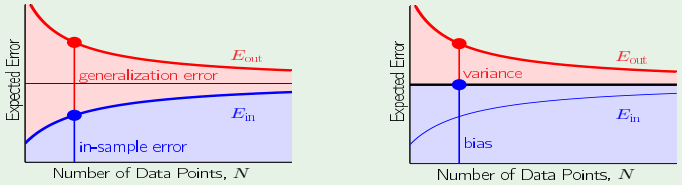
The curves



Simple Model

Complex Model

VC versus bias-variance



VC analysis

bias-variance

Linear regression case

Noisy target $y = \mathbf{w}^{*\mathrm{T}}\mathbf{x} + \text{noise}$

Data set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$

Linear regression solution: $\mathbf{w} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$

In-sample error vector $= \mathbf{X}\mathbf{w} - \mathbf{y}$

'Out-of-sample' error vector $= \mathbf{X}\mathbf{w} - \mathbf{y}'$

Linear regression case

Noisy target $y = \mathbf{w}^{*T}\mathbf{x} + \text{noise}$

Data set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$

Linear regression solution: $\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$

In-sample error vector $= \mathbf{X}\mathbf{w} - \mathbf{y}$

'Out-of-sample' error vector $= \mathbf{X}\mathbf{w} - \mathbf{y}'$

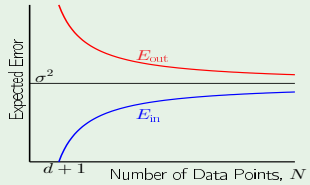
Learning curves for linear regression

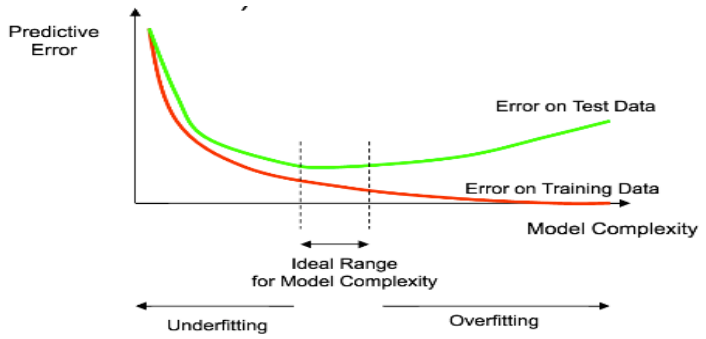
Best approximation error $= \sigma^2$

Expected in-sample error $= \sigma^2\left(1 - \frac{d+1}{N}\right)$

Expected out-of-sample error $= \sigma^2\left(1 + \frac{d+1}{N}\right)$

Expected generalization error $= 2\sigma^2\left(\frac{d+1}{N}\right)$





Regularization

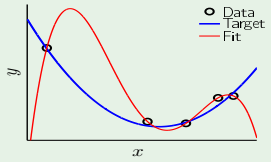
Illustration of overfitting

Simple target function

5 data points- **noisy**

4th-order polynomial fit

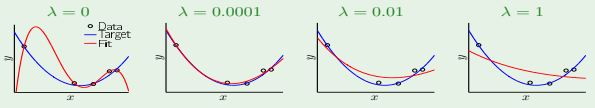
$E_{\text{in}} = 0, \quad E_{\text{out}}$ is huge



L2 regularization

The result

Minimizing $E_{\text{in}}(\mathbf{w}) + \frac{\lambda}{N}\mathbf{w}^{\mathsf{T}}\mathbf{w}$ for different λ's:



overfitting ⟶ ⟶ ⟶ ⟶ underfitting

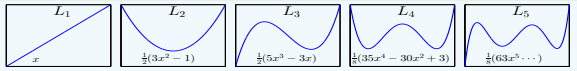
Reduces overfitting by adding a complexity penalty to the loss function
L2 regularization: complexity = sum of squares of weights

The polynomial model

$\mathcal{H}_{\text{Q}}$: polynomials of order Q linear regression in $\mathcal{Z}$ space

$$\mathbf{z} = \begin{bmatrix} 1 \\ L_1(x) \\ \vdots \\ L_{\text{Q}}(x) \end{bmatrix} \qquad \mathcal{H}_{\text{Q}} = \left\{ \sum_{q=0}^{Q} w_q \, L_q(x) \right\}$$

Legendre polynomials:



Unconstrained solution

Given $(x_1, y_1), \cdots, (x_N, y_n) \longrightarrow (\mathbf{z}_1, y_1), \cdots, (\mathbf{z}_N, y_n)$

Minimize $E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{w}^{\mathsf{T}}\mathbf{z}_n - y_n)^2$

Minimize $\frac{1}{N}(\mathbf{Z}\mathbf{w} - \mathbf{y})^{\mathsf{T}}(\mathbf{Z}\mathbf{w} - \mathbf{y})$

$\mathbf{w}_{\text{lin}} = (\mathbf{Z}^{\mathsf{T}}\mathbf{Z})^{-1}\mathbf{Z}^{\mathsf{T}}\mathbf{y}$

Augmented error

Minimizing $E_{\text{aug}}(\mathbf{w}) = E_{\text{in}}(\mathbf{w}) + \frac{\lambda}{N}\mathbf{w}^T\mathbf{w}$

$= \frac{1}{N}(\mathbf{Z}\mathbf{w} - \mathbf{y})^T(\mathbf{Z}\mathbf{w} - \mathbf{y}) + \frac{\lambda}{N}\mathbf{w}^T\mathbf{w}$ unconditionally

– solves –

Minimizing $E_{\text{in}}(\mathbf{w}) = \frac{1}{N}(\mathbf{Z}\mathbf{w} - \mathbf{y})^T(\mathbf{Z}\mathbf{w} - \mathbf{y})$

subject to: $\mathbf{w}^T\mathbf{w} \leq C$ ⟵ VC formulation

Weight 'decay'

Minimizing $E_{\text{in}}(\mathbf{w}) + \frac{\lambda}{N}\mathbf{w}^T\mathbf{w}$ is called weight *decay*. Why?

Gradient descent:

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \nabla E_{\text{in}}(\mathbf{w}(t)) - 2\eta\frac{\lambda}{N}\mathbf{w}(t)$$

$$= \mathbf{w}(t)\left(1 - 2\eta\frac{\lambda}{N}\right) - \eta \nabla E_{\text{in}}(\mathbf{w}(t))$$

Applies in neural networks:

$$\mathbf{w}^T\mathbf{w} = \sum_{l=1}^{L}\sum_{i=0}^{d^{(l-1)}}\sum_{j=1}^{d^{(l)}}\left(w_{ij}^{(l)}\right)^2$$

Variations of weight decay

Emphasis of certain weights: $\sum_{q=0}^{Q}\gamma_q w_q^2$

Examples: $\gamma_q = 2^q \implies$ low-order fit

$\gamma_q = 2^{-q} \implies$ high-order fit

Neural networks: different layers get different γ's

Tikhonov regularizer: $\mathbf{w}^T\Gamma^T\Gamma\mathbf{w}$

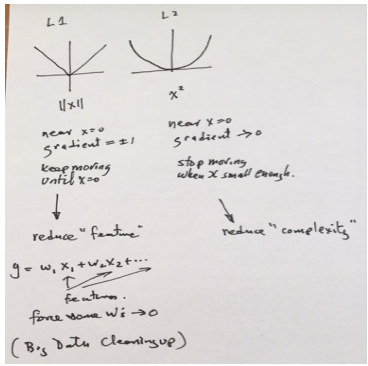
Andrew Ng

Debugging a learning algorithm:
Suppose you have implemented regularized linear regression to predict housing prices. However, when you test your hypothesis in a new set of houses, you find that it makes unacceptably large errors in its prediction. What should you try next?

- Get more training examples → fixes high variance
- Try smaller sets of features → fixes high variance
- Try getting additional features → fixes high bias
- Try adding polynomial features $(x_1^2, x_2^2, x_1x_2, \text{etc})$ → fixes high bias.
- Try decreasing λ → fixes high bias
- Try increasing λ → fixes high variance

Regularization
(sparsity)

L1 (LASSO) vs L2 regularization



L1 (LASSO=least absolute shrinkage and selection operator) leads to SPARSE parameterization.

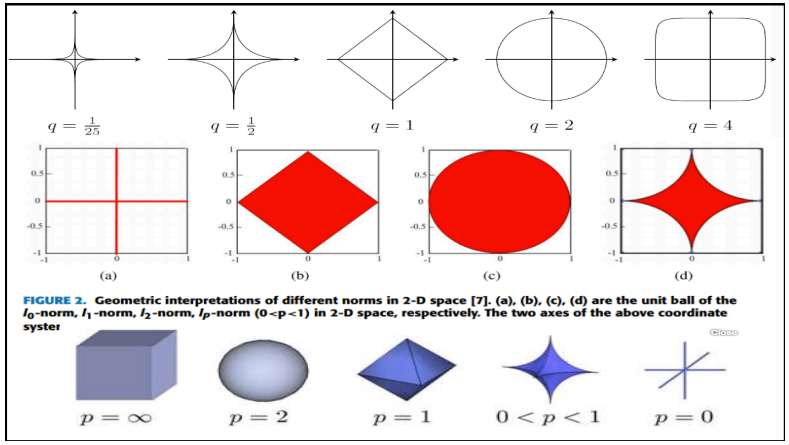
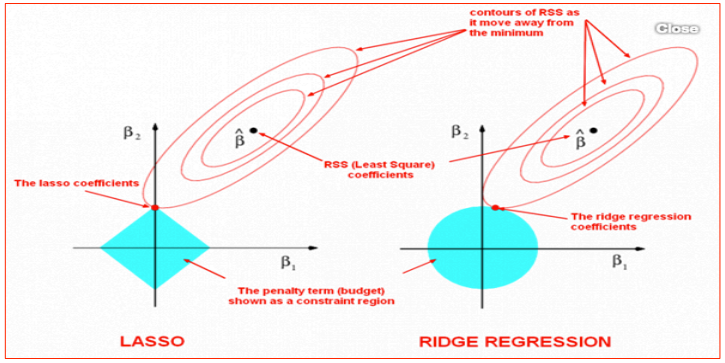


FIGURE 2. Geometric interpretations of different norms in 2-D space [7]. (a), (b), (c), (d) are the unit ball of the l_0-norm, l_1-norm, l_2-norm, l_p-norm (0<p<1) in 2-D space, respectively. The two axes of the above coordinate syster



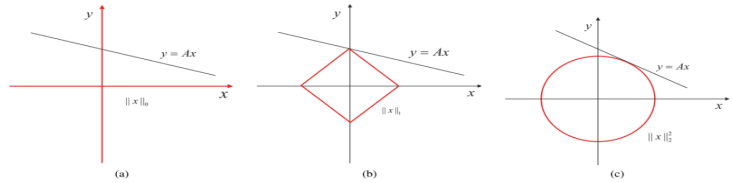
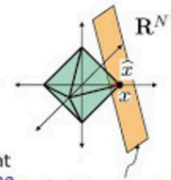


FIGURE 4. The geometry of the solutions of different norm regularization in 2-D space [7]. (a), (b) and (c) are the geometry of the solutions of the l_0-norm, l_1-norm, l_2-norm minimization, respectively.

Why L_1 Works

$$\hat{x} = \arg\min_{y=\Phi x'} \|x'\|_1$$



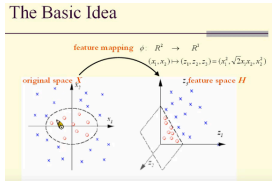
Criterion for success:
Ensure with high probability that a *randomly oriented (N-M)-plane, anchored on a K-face* of the L_1 ball, *will not intersect* the ball.

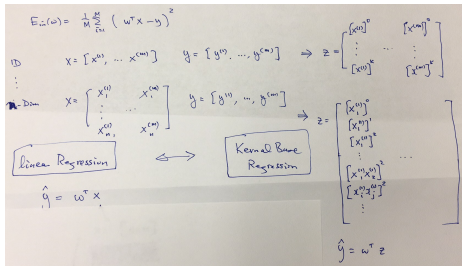
$\{x' : y = \Phi x'\}$
random orientation dimension N-M

Want K small, (N-M) small (i.e., M large)

Regularization
(kernel view – kernal linear regression)

Linear regression vs. Kernel regression





- Linear Kernel

$$\kappa(x,z) = < x, z >$$

- Polynomial Kernel

$$\kappa(x,z) = (< x, z > +1)^r, \quad r \in Z^+$$

- RBF (Gaussian) Kernel

$$\kappa(x,z) = \exp\left(\frac{-\|x-z\|^2}{2\sigma^2}\right), \quad \sigma \in R - \{0\}$$

Regularization

(kernel view -

Classical algorithm:
Regularization in RKHS (eg. kernel machines)

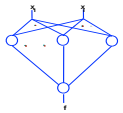
$$\min_{f \in H}\left[\frac{1}{n}\sum_{i=1}^{n} V(f(x_i) - y_i) + \lambda \|f\|_K^2\right]$$

implies

$$f(\mathbf{x}) = \sum_i^n \alpha_i K(\mathbf{x}, \mathbf{x}_i)$$

Remark (for later use):

Classical kernel machines correspond to shallow networks



Proposition 8. *Let $\mathbf{z} \in Z^m$ and $\gamma \in \mathbb{R}$, $\gamma > 0$. The empirical target, i.e. the function $f_{\gamma,\mathbf{z}} = f_{\mathbf{z}}$ minimizing the regularized empirical error*

$$\frac{1}{m}\sum_{i=1}^{m}(y_i - f(x_i))^2 + \gamma\|f\|_K^2$$

over $f \in \mathcal{H}_K$, may be expressed as

$$f_{\mathbf{z}}(x) = \sum_{i=1}^{m} a_i K(x, x_i)$$

where $a = (a_1, \ldots, a_m)$ is the unique solution of the well-posed linear system in $\mathbb{R}^m$

$$(\gamma m \, \mathrm{Id} + K[\mathbf{x}])a = \mathbf{y}.$$

Here, we recall, $K[\mathbf{x}]$ is the $m \times m$ matrix whose (i,j) entry is $K(x_i, x_j)$, $\mathbf{x} = (x_1, \ldots, x_m) \in X^m$, and $\mathbf{y} = (y_1, \ldots, y_m) \in Y^m$ such that $\mathbf{z} = ((x_1, y_1), \ldots, (x_m, y_m))$.

Proof. Let $H(f) = \frac{1}{m}\sum_{i=1}^{m}(y_i - f(x_i))^2 + \gamma\|f\|_K^2$ and write, for any $f \in \mathcal{H}_K$, $f = \sum_{k=1}^{\infty} c_k\phi_k$. Recall that $\|f\|_K^2 = \sum_{k=1}^{\infty}\frac{c_k^2}{\lambda_k}$.

For every $k \geq 1$, $\frac{\partial H}{\partial c_k} = \frac{1}{m}\sum_{i=1}^{m} -2(y_i - f(x_i))\phi_k(x_i) + 2\gamma\frac{c_k}{\lambda_k}$. If f is a minimum of H, then, for each k, we must have $\frac{\partial H}{\partial c_k} = 0$ or, solving for c_k,

$$c_k = \lambda_k \sum_{i=1}^{m} a_i\phi_k(x_i)$$

where $a_i = \frac{y_i - f(x_i)}{\gamma m}$. Thus,

$$\begin{aligned} f(x) &= \sum_{k=1}^{\infty} c_k\phi_k(x) = \sum_{k=1}^{\infty}\lambda_k\sum_{i=1}^{m} a_i\phi_k(x_i)\phi_k(x) \\ &= \sum_{i=1}^{m} a_i\sum_{k=1}^{\infty}\lambda_k\phi_k(x_i)\phi_k(x) = \sum_{i=1}^{m} a_i K(x_i, x). \end{aligned}$$

Replacing $f(x)$ in the definition of a_i above, we obtain

$$a_i = \frac{y_i - \sum_{i=1}^{m} a_i K(x_i, x)}{\gamma m}.$$

Multiplying both sides by γm and writing the result in matrix form, we obtain $(\gamma m\,\mathrm{Id} + K[\mathbf{x}])a = \mathbf{y}$. And this system is well-posed since $K[\mathbf{x}]$ is positive and the addition of a positive matrix and the identity is strictly positive. □

Classical kernel machines are equivalent to shallow networks

Kernel machines…

$$f(\mathbf{x}) = \sum_i^l c_i K(\mathbf{x}, \mathbf{x}_i) + b$$

can be "written" as shallow networks: the value of K corresponds to the "activity" of the "unit" for the input and the correspond to "weights"