ECE521 lecture 4:
19 January 2017

Optimization, MLE, regularization

First four lectures

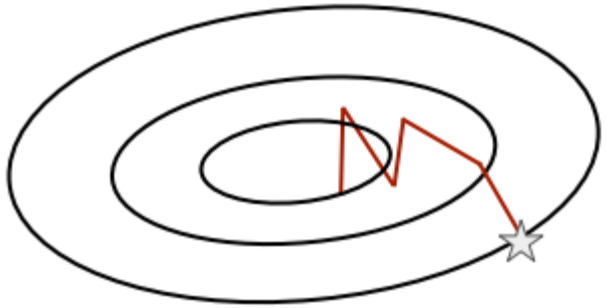
- **Lectures 1 and 2:**
  - Intro to ML
  - Probability review
  - Types of loss functions and algorithms
- **Lecture 3:**
  - KNN
  - Convexity in optimization
- **Lecture 4:**
  - Momentum and SGD in optimization
  - MLE
  - Regularization

Today's lecture

- **Tricks for gradient descent**
  - Momentum
  - SGD
- MLE
  - Univariate Normal distribution
  - Multivariate Normal distribution
  - Mixture of Normal distributions
- Regularization

Momentum

- Recall the update equation for gradient descent: $W \leftarrow W - \eta \frac{\partial \mathcal{L}}{\partial W}$



- It can also be written as $\Delta W = -\eta \frac{\partial L(t)}{\partial W}$

Momentum

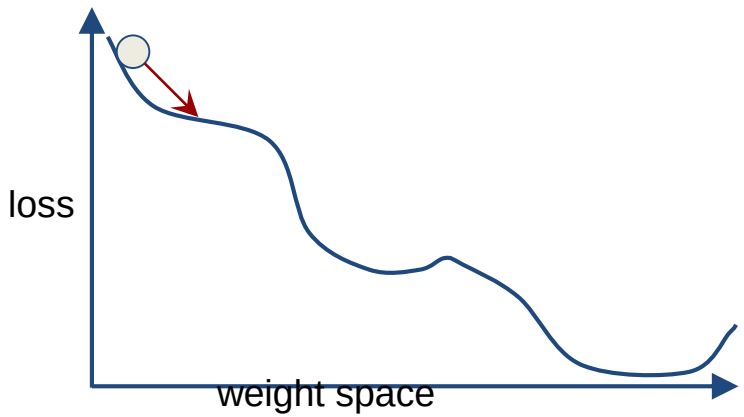
- Intuition: instead of zigzagging around the weight space using the latest gradient, we would like to "accelerate" learning by accumulating some "velocity/momentum" using the past gradients
- Add a momentum term to the update equation:

$$\Delta W(t) = -\eta \frac{\partial L(t)}{\partial W} + \alpha \, \Delta W(t-1)$$

- $\alpha \in [0, 1)$ is the momentum coefficient on how much velocity we pick up. $\alpha = 0.9$ works well in general

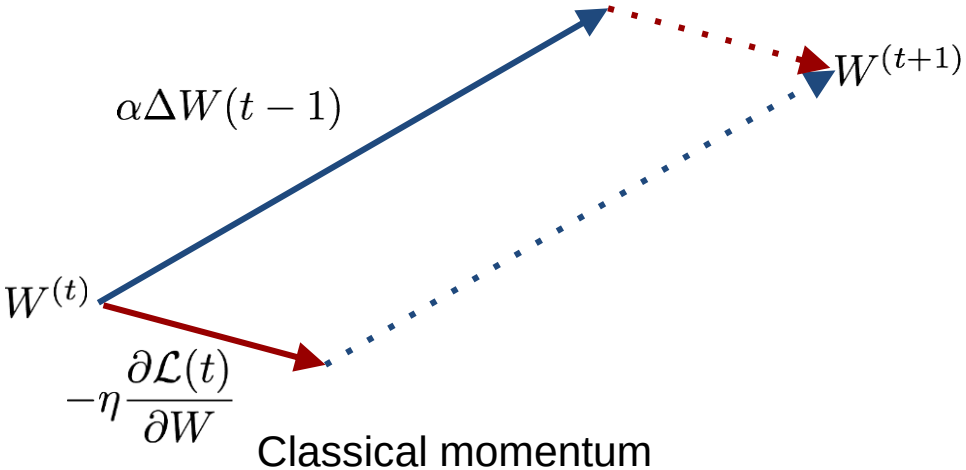
Momentum

- Imagine a ball on the loss function surface. The location of the ball in the horizontal plane represents the weight vector
- The ball starts off by following the gradient, but once it has velocity, it no longer does steepest descent
- Its momentum tends to keep it going towards the previous direction

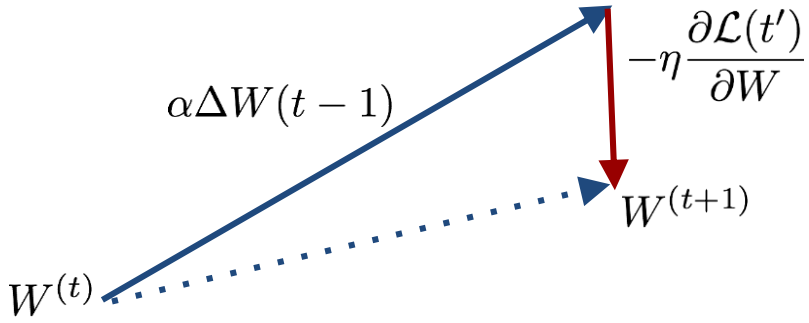


Momentum

- One extension is Nesterov's momentum
- Intuition: classical momentum method often has an undesirable overshoot behavior. This can be reduced by "looking ahead". The gradient is computed assuming we take one extra momentum step.



Classical momentum



Nesterov's momentum

Momentum

- Classical momentum:

$$\Delta W(t) = -\eta \frac{\partial L(t)}{\partial W} + \alpha \, \Delta W(t-1)$$

- Nesterov's momentum:

$$\Delta W(t) = -\eta \frac{\partial L(t')}{\partial W} + \alpha \, \Delta W(t-1)$$

Stochastic gradient descent

- The loss function we are optimizing for a machine learning problem is almost always a summation over the individual losses of each training example:

$$\mathcal{L}(W) = \sum_m \mathcal{L}_m(\mathbf{x}^{(m)}, \mathbf{y}^{(m)}, W)$$

- Obtaining the gradient of the entire summation requires computing the gradient for each data term:

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_m \frac{\partial \mathcal{L}_m(\mathbf{x}^{(m)}, \mathbf{y}^{(m)}, W)}{\partial W}$$

- This is *very inefficient* for **big data**, e.g. learning from billions of Wikipedia articles or millions of images

Stochastic gradient descent

- Intuition: dataset can be highly redundant, especially for big data. Update the weights using a rough estimate of the exact gradient that is fast to compute
- This turned out to be the most important algorithm to advance machine learning, deep learning and AI in the past ten years
- SGD is the most famous algorithm to scale to big data: as the size of the dataset increases, the computation per weight update remains *the same*

Stochastic gradient descent

- SGD idea: at each iteration, subsample a small amount of data (even just one point can work) and use that to estimate the gradient
- Each update is noisy, but very fast!
- This is the basis of optimizing ML algorithms with huge datasets (e.g., recent deep learning)
- Computing gradients using the full dataset is called batch learning; using subsets of data is called **mini-batch** learning

Stochastic gradient descent

- Suppose we stupidly made a copy of each point so that we now have twice as much data. The log-likelihood is now:

$$\mathcal{L}(W) = \sum_{m=1}^{M} \mathcal{L}_m(\mathbf{x}^{(m)}, \mathbf{y}^{(m)}, W) + \sum_{m=M+1}^{2M} \mathcal{L}_m(\mathbf{x}^{(m)}, \mathbf{y}^{(m)}, W)$$

$$= 2 \sum_{m=1}^{M} \mathcal{L}_m(\mathbf{x}^{(m)}, \mathbf{y}^{(m)}, W)$$

- In other words, the optimal parameters don't change, but we have to do twice as much work to compute the log-likelihood and its gradient!
- The reason SGD works is because similar data yield similar gradients, so if there is enough redundancy in the data, the noise from subsampling won't be so bad

Stochastic gradient descent

- How do we choose an appropriate step size?
- Robbins and Monro (1951): pick a sequence of η_t such that:

$$\sum_{t=0}^{\infty} \eta_t = \infty, \ \sum_{t=0}^{\infty} \eta_t^2 < \infty$$

- Satisfied by $\eta_t \propto \frac{1}{t}$ (as one example)

Stochastic gradient descent

- SGD is very easy to implement compared to other methods, but the step sizes need to be tuned to different problems, whereas batch learning typically “just works”
- Tip 1: divide the log-likelihood estimate by the size of your mini-batches. This makes the learning rate invariant to mini-batch size
- Tip 2: subsample without replacement so that you visit each point on each pass through the dataset (this is known as an **epoch**)

References on optimization

- Convex optimization:
  - http://web.stanford.edu/class/ee364a/index.html
  - http://stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf
- Stats (python):
  - Scipy stats: http://docs.scipy.org/doc/scipy-0.14.0/reference/stats.html
- Optimization (python):
  - Scipy optimize: http://docs.scipy.org/doc/scipy/reference/optimize.html
- Optimization (Matlab):
  - minFunc: http://www.cs.ubc.ca/~schmidtm/Software/minFunc.html
- General ML:
  - Scikit-Learn: http://scikit-learn.org/stable
- See the course website as well

Today's lecture

- Tricks for gradient descent
  - Momentum
  - SGD
- **MLE**
  - Univariate Normal distribution
  - Multivariate Normal distribution
  - Mixture of Normal distributions
- Regularization

Maximum Likelihood Estimation

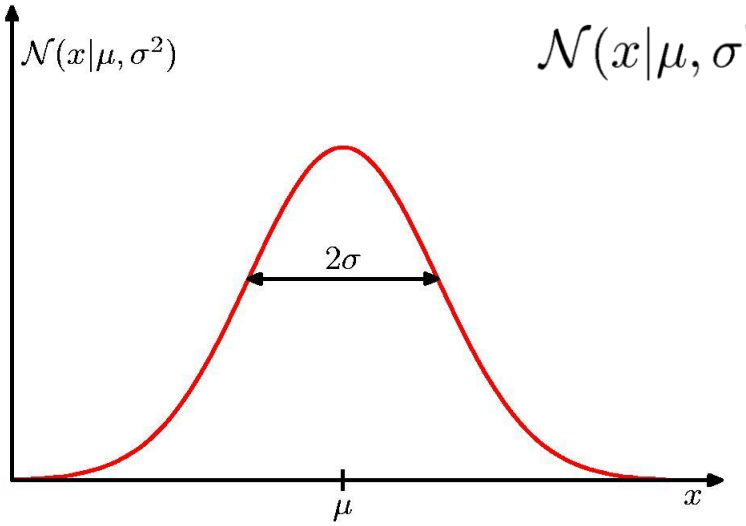
- **MLE** is a way of estimating the parameters of a model
- An alternative to MLE is the Maximum a-posteriori probability (MAP) estimate
- Recall:

$$P(\text{parameters} \mid \text{data}) = \frac{P(\text{data} \mid \text{parameters})P(\text{parameters})}{P(\text{data})}$$

$$\text{posterior} \propto \text{likelihood} \times \text{prior}.$$

Recall: Univariate Normal Distribution

- In the case of a single variable *x*, the Normal (aka **Gaussian**) distribution takes the form:



$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}$$

which is governed by two parameters:

- μ (mean)
- σ^2 (variance)

- The Normal distribution satisfies:

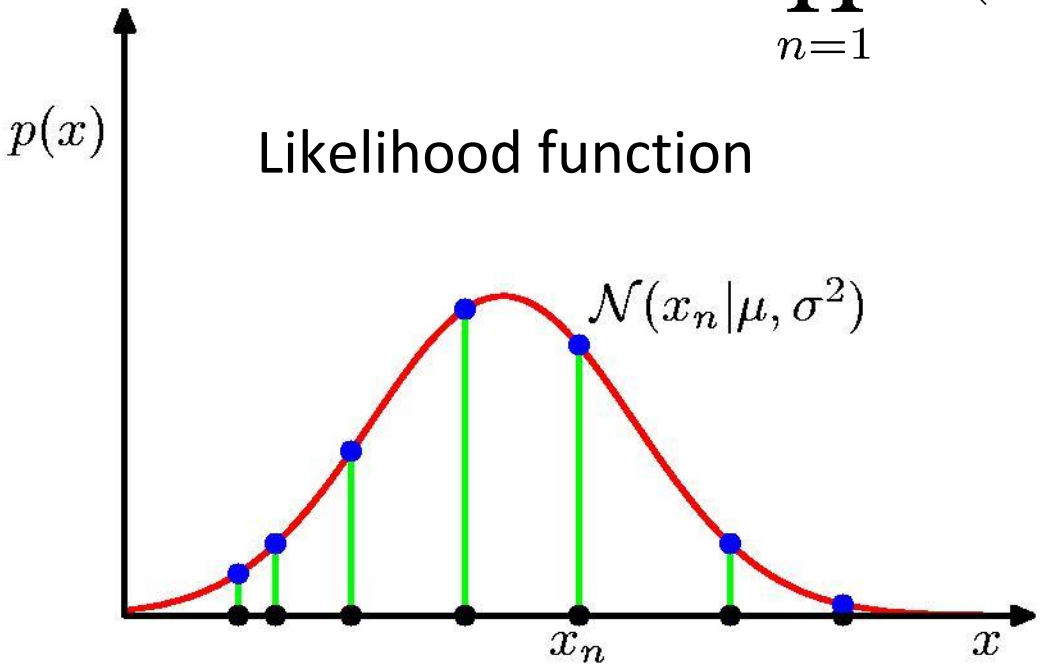
$$\mathcal{N}(x|\mu,\sigma^2) > 0$$

$$\int_{-\infty}^{\infty} \mathcal{N}\left(x|\mu,\sigma^2\right)\,\mathrm{d}x = 1$$

MLE for the Normal Distribution

- Suppose we have a dataset of i.i.d. observations $\mathbf{x} = (x_1,\ldots,x_N)^T$, representing N one-dimensional observations.
- Because our dataset **x** is i.i.d., we can write down the joint probability of all the data points given μ and σ^2 as:

$$p(\mathbf{x}|\mu,\sigma^2) = \prod_{n=1}^{N} \mathcal{N}\left(x_n|\mu,\sigma^2\right)$$



Likelihood function

When viewed as a function of μ and σ^2, this is called the likelihood function for the Normal

MLE for the Normal Distribution

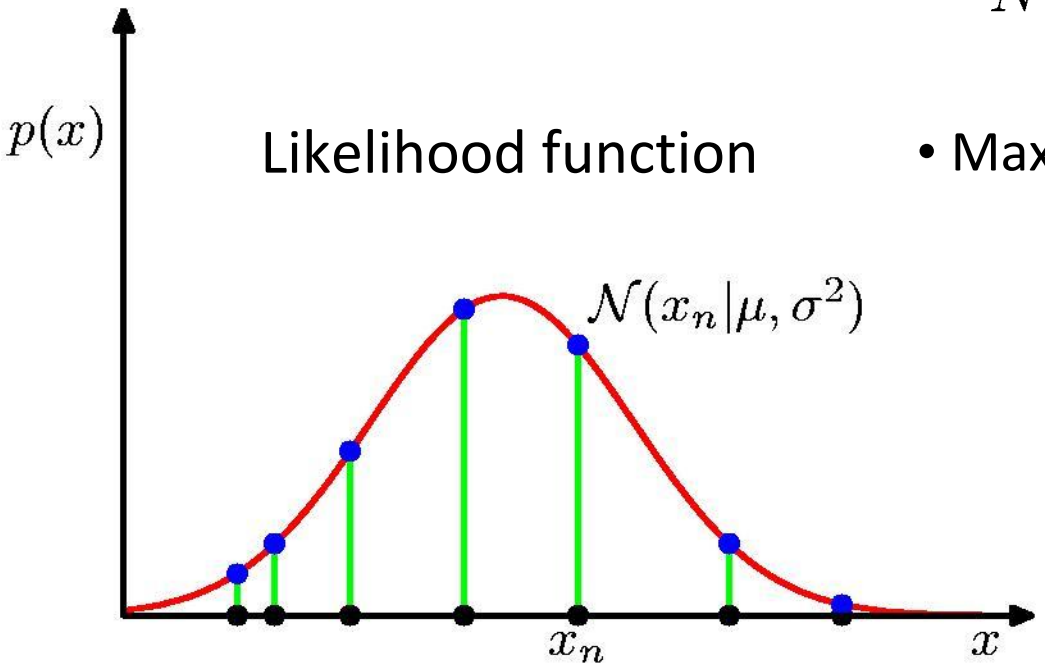
- The log-likelihood can be written as:

$$\ln p\left(\mathbf{x}|\mu,\sigma^2\right) = -\frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n-\mu)^2 - \frac{N}{2}\ln\sigma^2 - \frac{N}{2}\ln(2\pi)$$

- Maximizing w.r.t μ gives:

Sample mean

$$\mu_{\mathrm{ML}} = \frac{1}{N}\sum_{n=1}^{N} x_n$$



Likelihood function

- Maximizing w.r.t σ^2 gives:

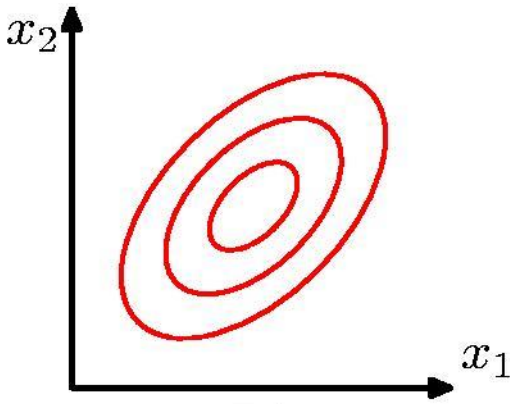
Sample variance

$$\sigma^2_{\mathrm{ML}} = \frac{1}{N}\sum_{n=1}^{N}(x_n-\mu_{\mathrm{ML}})^2$$

Multivariate Normal Distribution

- For a D-dimensional vector **x**, the Normal distribution takes the form:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}}\frac{1}{|\boldsymbol{\Sigma}|^{1/2}}\exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}$$



which is governed by two parameters:

- **μ** is a D-dimensional mean vector.
- **Σ** is a D by D covariance matrix.

and |**Σ**| denotes the determinant of **Σ**

- Note that the covariance matrix is a positive-definite matrix

Geometry of the Normal Distribution

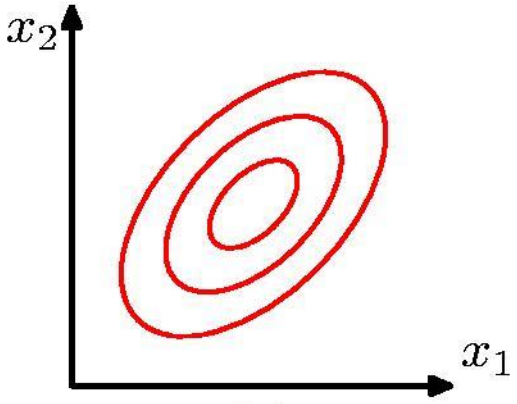
- For a D-dimensional vector **x**, the Normal distribution takes the form:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}}\frac{1}{|\boldsymbol{\Sigma}|^{1/2}}\exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}$$

- We sometimes write:

$$\Delta^2 = (\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})$$

- **Δ** is known as Mahalanobis distance. The Normal distribution is constant on surfaces in x-space for which **Δ** is constant.



- Other properties:

$$\mathbb{E}[\mathbf{x}] = \boldsymbol{\mu}$$

$$\mathrm{cov}[\mathbf{x}] = \mathbb{E}\left[(\mathbf{x}-\mathbb{E}[\mathbf{x}])(\mathbf{x}-\mathbb{E}[\mathbf{x}])^{\mathrm{T}}\right] = \boldsymbol{\Sigma}$$

The precision matrix

- Consider a D-dimensional Normal distribution: $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$
- We can partition **x** into two disjoint subsets x_a and x_b:

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{pmatrix} \qquad \boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{pmatrix} \qquad \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{aa} & \boldsymbol{\Sigma}_{ab} \\ \boldsymbol{\Sigma}_{ba} & \boldsymbol{\Sigma}_{bb} \end{pmatrix}$$

- In some situations, it is more convenient to work with the **precision matrix** (inverse of the covariance matrix):

$$\boldsymbol{\Lambda} \equiv \boldsymbol{\Sigma}^{-1} \qquad \boldsymbol{\Lambda} = \begin{pmatrix} \boldsymbol{\Lambda}_{aa} & \boldsymbol{\Lambda}_{ab} \\ \boldsymbol{\Lambda}_{ba} & \boldsymbol{\Lambda}_{bb} \end{pmatrix}$$

- Note that $\boldsymbol{\Lambda}_{aa}$ is not given by the inverse of $\boldsymbol{\Sigma}_{aa}$
- The above is useful when exploring graphical models (to come)

Conditional Distribution

- It turns out that the conditional distribution is also a Normal distribution:

$$p(\mathbf{x}_a|\mathbf{x}_b) = \mathcal{N}(\mathbf{x}_a|\boldsymbol{\mu}_{a|b}, \boldsymbol{\Sigma}_{a|b})$$

Covariance does not depend on x_b

$$\begin{aligned}
\boldsymbol{\Sigma}_{a|b} &= \boldsymbol{\Lambda}_{aa}^{-1} = \boldsymbol{\Sigma}_{aa} - \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}\boldsymbol{\Sigma}_{ba} \\
\boldsymbol{\mu}_{a|b} &= \boldsymbol{\Sigma}_{a|b}\left\{\boldsymbol{\Lambda}_{aa}\boldsymbol{\mu}_a - \boldsymbol{\Lambda}_{ab}(\mathbf{x}_b - \boldsymbol{\mu}_b)\right\} \\
&= \boldsymbol{\mu}_a - \boldsymbol{\Lambda}_{aa}^{-1}\boldsymbol{\Lambda}_{ab}(\mathbf{x}_b - \boldsymbol{\mu}_b) \\
&= \boldsymbol{\mu}_a + \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}(\mathbf{x}_b - \boldsymbol{\mu}_b)
\end{aligned}$$

Linear function of x_b

Marginal Distribution

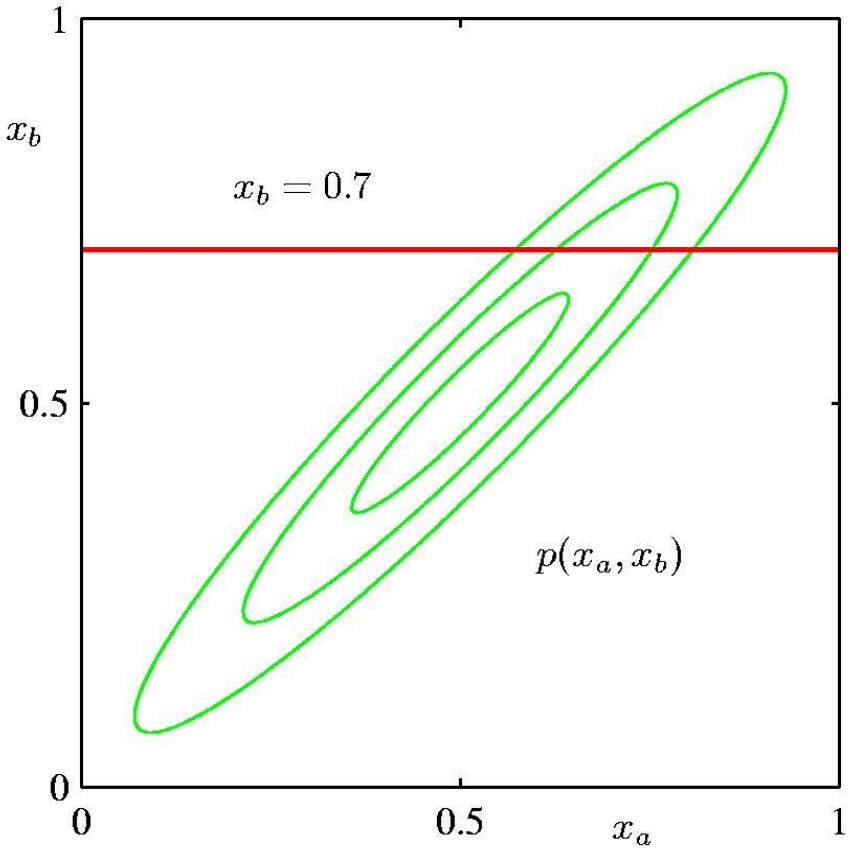
- It turns out that the marginal distribution is also a Normal distribution:

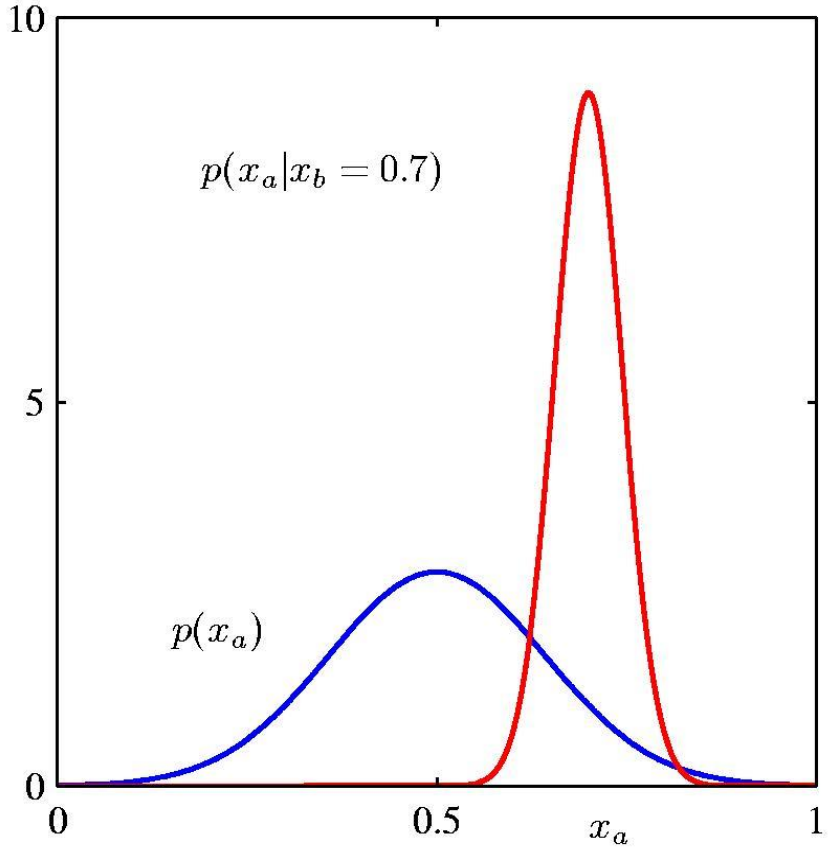
$$\begin{aligned} p(\mathbf{x}_a) &= \int p(\mathbf{x}_a, \mathbf{x}_b)\, \mathrm{d}\mathbf{x}_b \\ &= \mathcal{N}(\mathbf{x}_a | \boldsymbol{\mu}_a, \boldsymbol{\Sigma}_{aa}) \end{aligned}$$

- For a marginal distribution, the mean and covariance are most simply expressed in terms of the partitioned covariance matrix.

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{pmatrix} \qquad \boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{pmatrix} \qquad \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{aa} & \boldsymbol{\Sigma}_{ab} \\ \boldsymbol{\Sigma}_{ba} & \boldsymbol{\Sigma}_{bb} \end{pmatrix}$$

Conditional and Marginal Distributions





MLE of the Multivariate Normal Distribution

- Suppose we observed i.i.d data $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$.
- We can construct the log-likelihood function, which is a function of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$:

$$\ln p(\mathbf{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\frac{ND}{2}\ln(2\pi) - \frac{N}{2}\ln|\boldsymbol{\Sigma}| - \frac{1}{2}\sum_{n=1}^{N}(\mathbf{x}_n - \boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}_n - \boldsymbol{\mu})$$

- Note that the likelihood function depends on the N data points only through the following sums:

$$\sum_{n=1}^{N}\mathbf{x}_n \qquad\qquad \sum_{n=1}^{N}\mathbf{x}_n\mathbf{x}_n^{\mathrm{T}}$$

"Sufficient Statistics"

MLE of the Multivariate Normal Distribution

- To find a maximum likelihood estimate of the mean, we set the derivative of the log-likelihood function to zero:

$$\frac{\partial}{\partial \boldsymbol{\mu}} \ln p(\mathbf{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \boldsymbol{\Sigma}^{-1}(\mathbf{x}_n - \boldsymbol{\mu}) = 0$$

and solve to obtain:

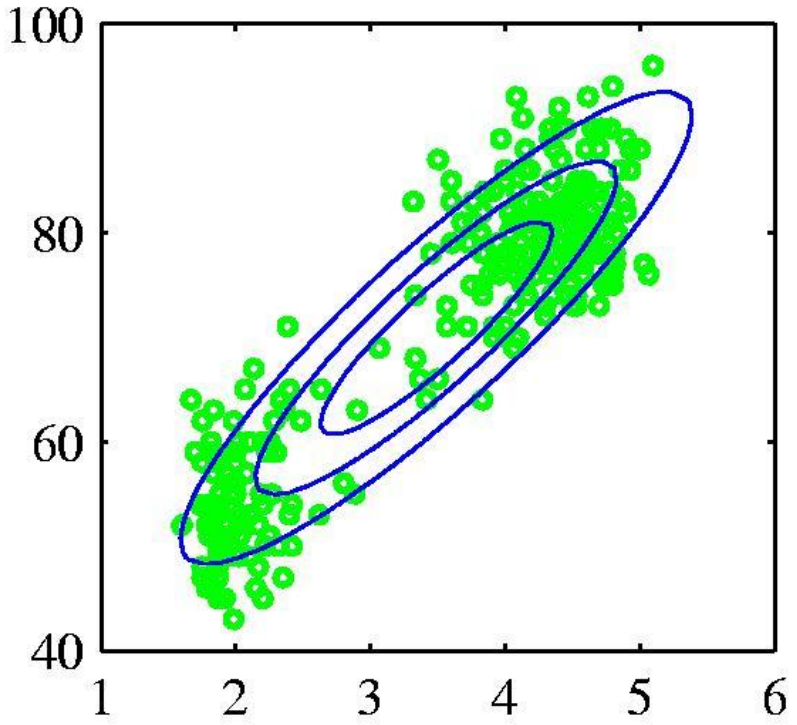
$$\boldsymbol{\mu}_{\mathrm{ML}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n.$$

- Similarly, we can find the ML estimate of $\boldsymbol{\Sigma}$:

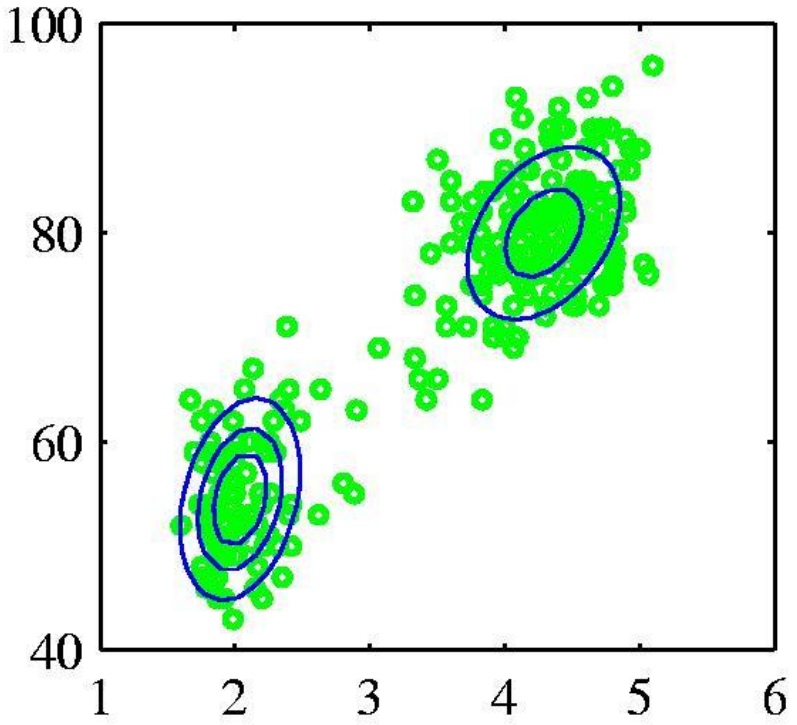
$$\boldsymbol{\Sigma}_{\mathrm{ML}} = \frac{1}{N} \sum_{n=1}^{N} (\mathbf{x}_n - \boldsymbol{\mu}_{\mathrm{ML}})(\mathbf{x}_n - \boldsymbol{\mu}_{\mathrm{ML}})^{\mathrm{T}}.$$

Example 3: Mixture of Gaussians

- When modeling real-world data, the Gaussian (Normal) assumption may not be appropriate
- Consider the following example: Old Faithful Dataset



Single Gaussian
(Normal) distribution



Mixture of two
Gaussians

Mixture of Gaussians

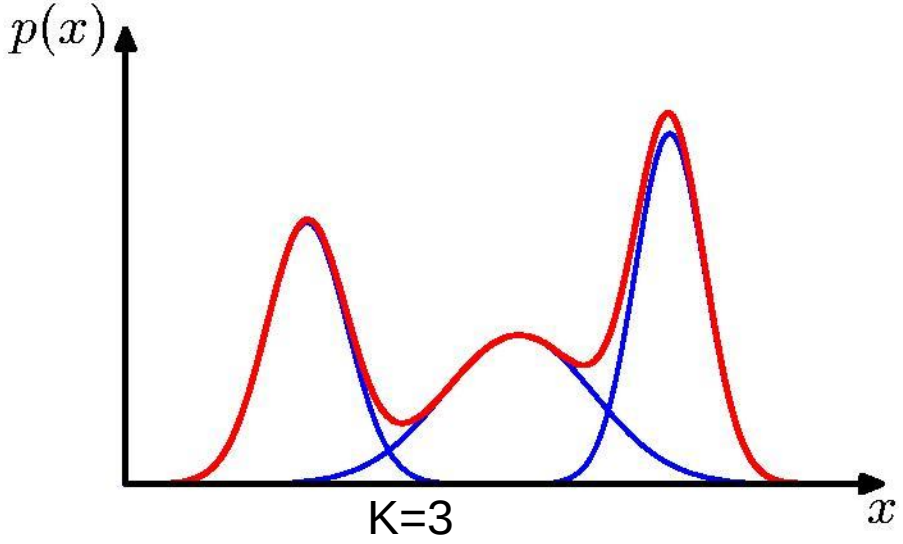
- We can combine simple models into a complex model by defining a superposition of K Normal densities of the form:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

Component

Mixing coefficient

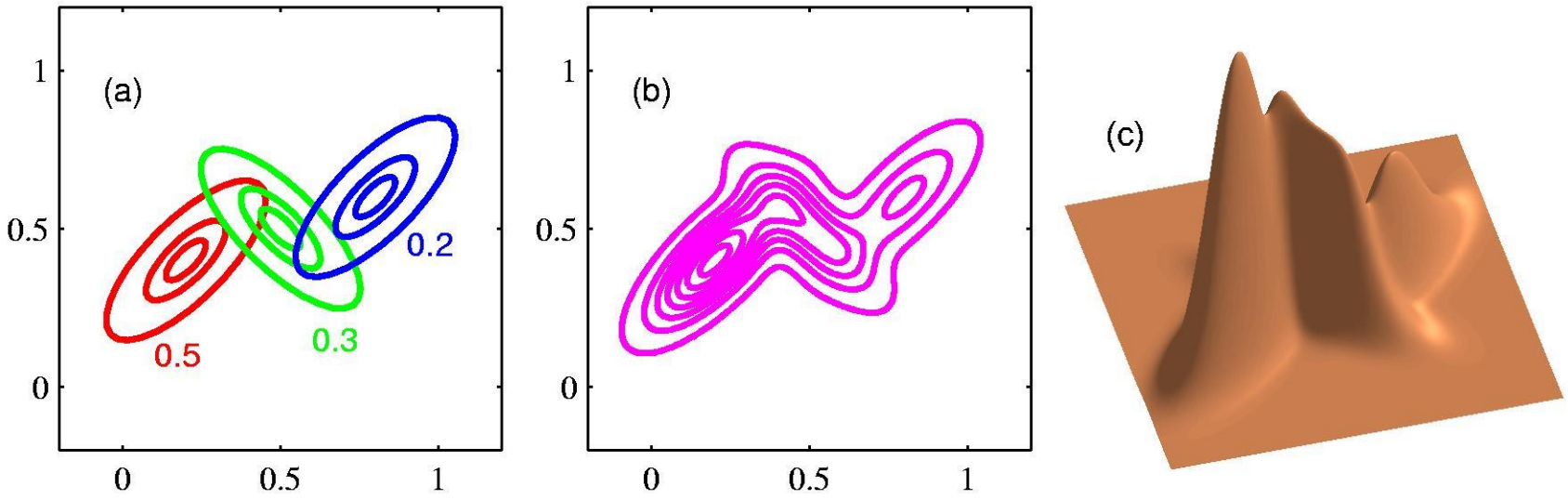
$$\forall k : \pi_k \geqslant 0 \qquad \sum_{k=1}^{K} \pi_k = 1$$



- Note that each Normal component has its own mean μ_k and covariance Σ_k. The parameters π_k are called mixing coefficients
- More generally, mixture models can comprise linear combinations of other distributions

Mixture of Gaussians

- Illustration of a mixture of 3 Gaussians in a 2-dimensional space:



(a) Contours of constant density of each of the mixture components, along with the mixing coefficients

(b) Contours of marginal probability density $p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$

(c) A surface plot of the distribution p(x)

Maximum Likelihood Estimation

- Given a dataset D, we can determine model parameters μ_k, Σ_k, π_k by maximizing the log-likelihood function:

$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

Log of a sum: no closed-form solution

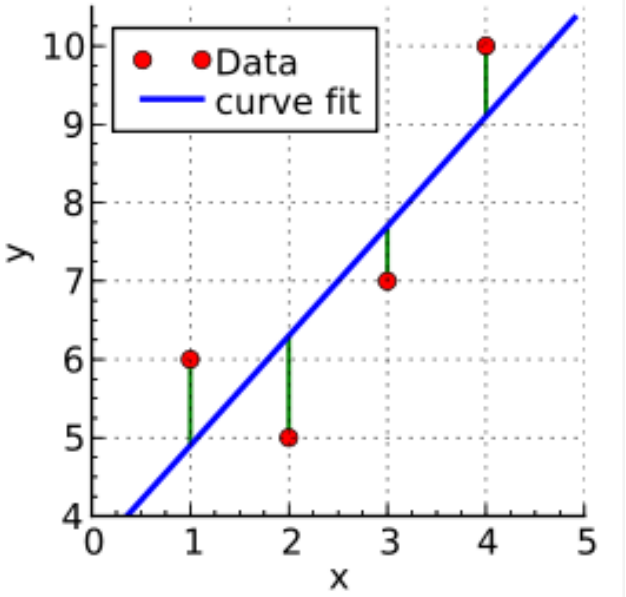
- **Solution**: use standard, iterative, numerical optimization methods or the Expectation Maximization algorithm. (More on this later)

Today's lecture

- Tricks for gradient descent
  - Momentum
  - SGD
- MLE
  - Univariate Normal distribution
  - Multivariate Normal distribution
  - Mixture of Normal distributions
- **Regularization**

Simplistic method to fit a line: Linear Least Squares

From last class: Minimize **the sum of the squares of the errors** between the predictions $y(\mathbf{x}_n, \mathbf{w})$ for each data point x_n and the corresponding real-valued targets t_n.



Source: Wikipedia

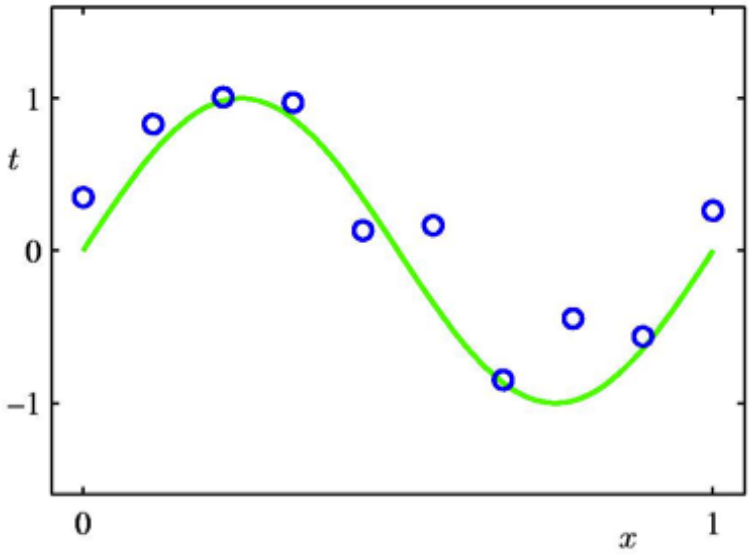
Loss function: sum-of-squared error function:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\mathbf{x}_n^T\mathbf{w} - t_n)^2$$

$$= \frac{1}{2}(\mathbf{Xw} - \mathbf{t})^{\mathbf{T}}(\mathbf{Xw} - \mathbf{t}).$$

LLS for *Polynomial* Curve Fitting

Consider observing a training set consisting of N 1-dimensional observations: $\mathbf{x} = (x_1, x_2, ..., x_N)^T$, together with corresponding real-valued targets: $\mathbf{t} = (t_1, t_2, ..., t_N)^T$.



Goal: Fit the data using a polynomial function of the form:

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + ... + w_M x^M = \sum_{j=0}^{M} w_j x^j.$$

Note: the polynomial function is a nonlinear function of x, but it is a linear function of **w,** the coefficients! **Linear Models**

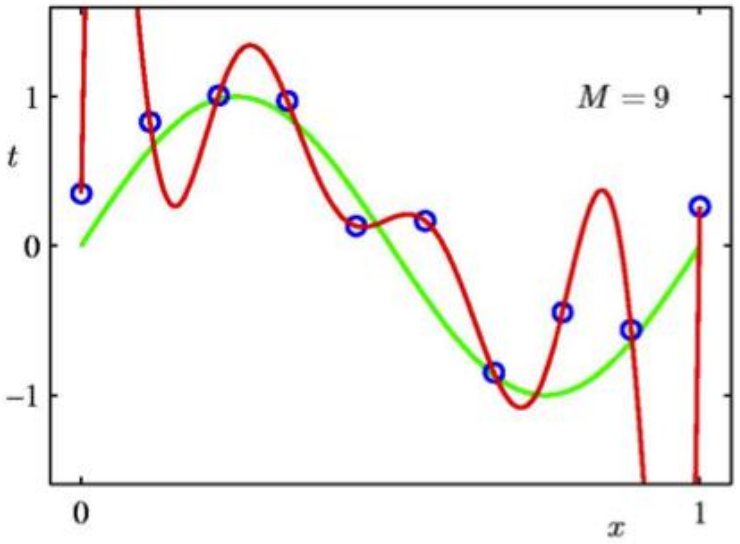
Generalization

• The goal is achieve good **generalization** by making accurate predictions for new test data that is not known during learning.

• Choosing the values of parameters that minimize the loss function on the training data may not be the best option.

• We would like to model the true regularities in the data and ignore the noise in the data:

- It is hard to know which regularities are real and which are accidental due to the particular training examples we happen to pick.



• Intuition: We expect the model to generalize if it explains the data well given the complexity of the model.

• If the model has as many degrees of freedom as the data, it can fit the data perfectly. But this is not very informative.

• Some theory on how to control model complexity to optimize generalization.

One Way to Penalize Complexity

One technique for controlling over-fitting phenomenon is **regularization**, which amounts to adding a penalty term to the error function

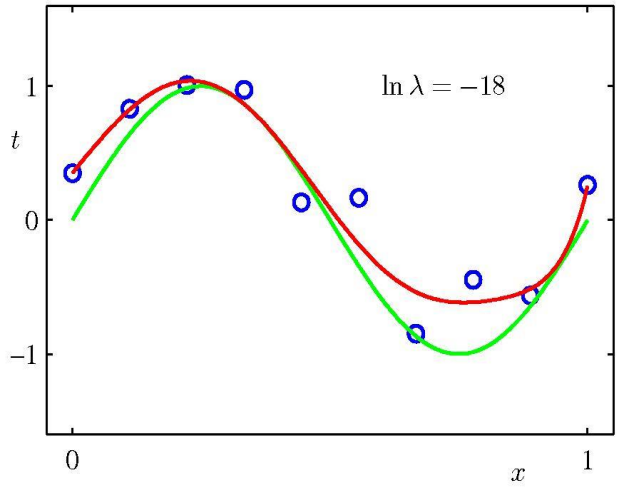
penalized error function

target value

regularization parameter

$$\widetilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n,\mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

where $||\mathbf{w}|| = \mathbf{w}^T\mathbf{w} = w_1^2 + w_2^2 + ... + w_M^2$ and , is is called the regularization term. Note that we do not penalize the bias term w_0



- The idea is to "shrink" estimated parameters towards zero (or towards the mean of some other weights)
- Shrinking to zero: penalize coefficients based on their size
- For a penalty function which is the sum of the squares of the parameters, this is known as a "**weight decay**" or "**ridge regression**"

Regularized Least Squares

• Let us consider the following error function:

$$E_D(\mathbf{w}) + \lambda E_W(\mathbf{w})$$

Data term + Regularization term

λ is called the regularization coefficient.

• Using sum-of-squares error function with a quadratic penalization term, we obtain:

$$\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2 + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

which is minimized by setting:

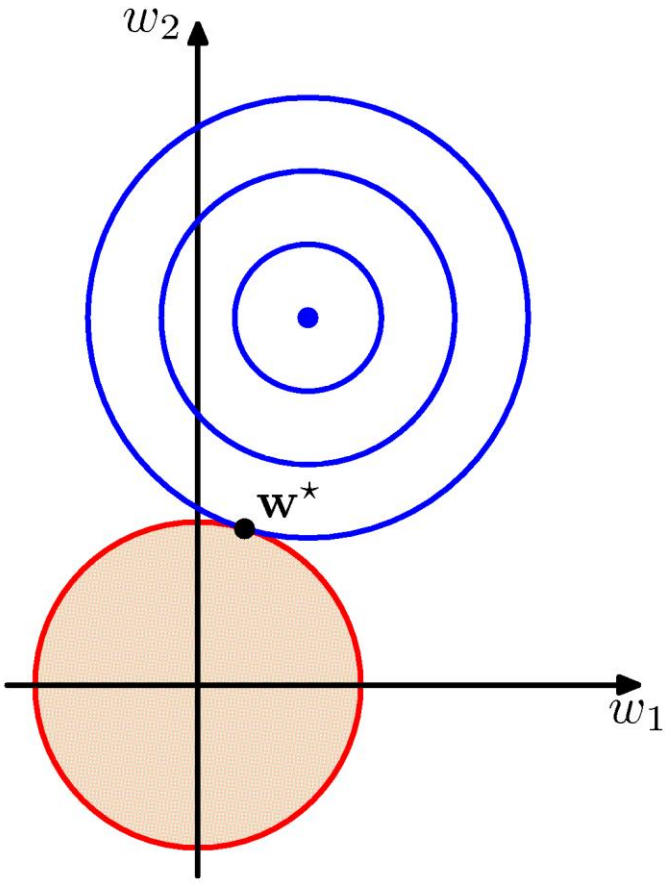
Ridge regression

$$\mathbf{w} = \left(\lambda\mathbf{I} + \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{t}.$$

The solution adds a positive constant to the diagonal of $\boldsymbol{\Phi}^{\mathbf{T}}\boldsymbol{\Phi}$. This helps make the problem nonsingular, even if $\boldsymbol{\Phi}^{\mathbf{T}}\boldsymbol{\Phi}$ is not of full rank (e.g. when the number of training examples is less than the number of basis functions)

Effect of Regularization

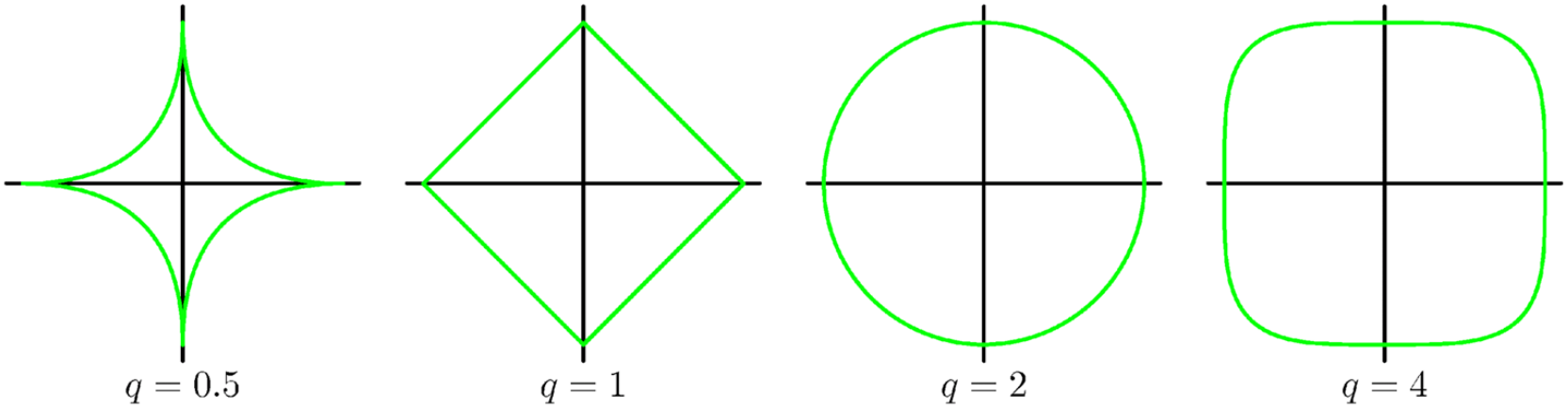
- The overall error function is the sum of two parabolic bowls.
- The combined minimum lies on the line between the minimum of the squared error and the origin.
- The regularizer shrinks model parameters to zero.



Other Regularizers

Using a more general regularizer, we get:

$$\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|w_j|^q$$



Lasso

Quadratic

The Lasso

- Penalize the absolute value of the weights:

$$\mathbf{w}^{lasso} = \underset{\mathbf{w}}{\operatorname{argmin}} \left[\frac{1}{2} \sum_{n=1}^{N} \left(t_n - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n) \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{M-1} |w_j| \right].$$

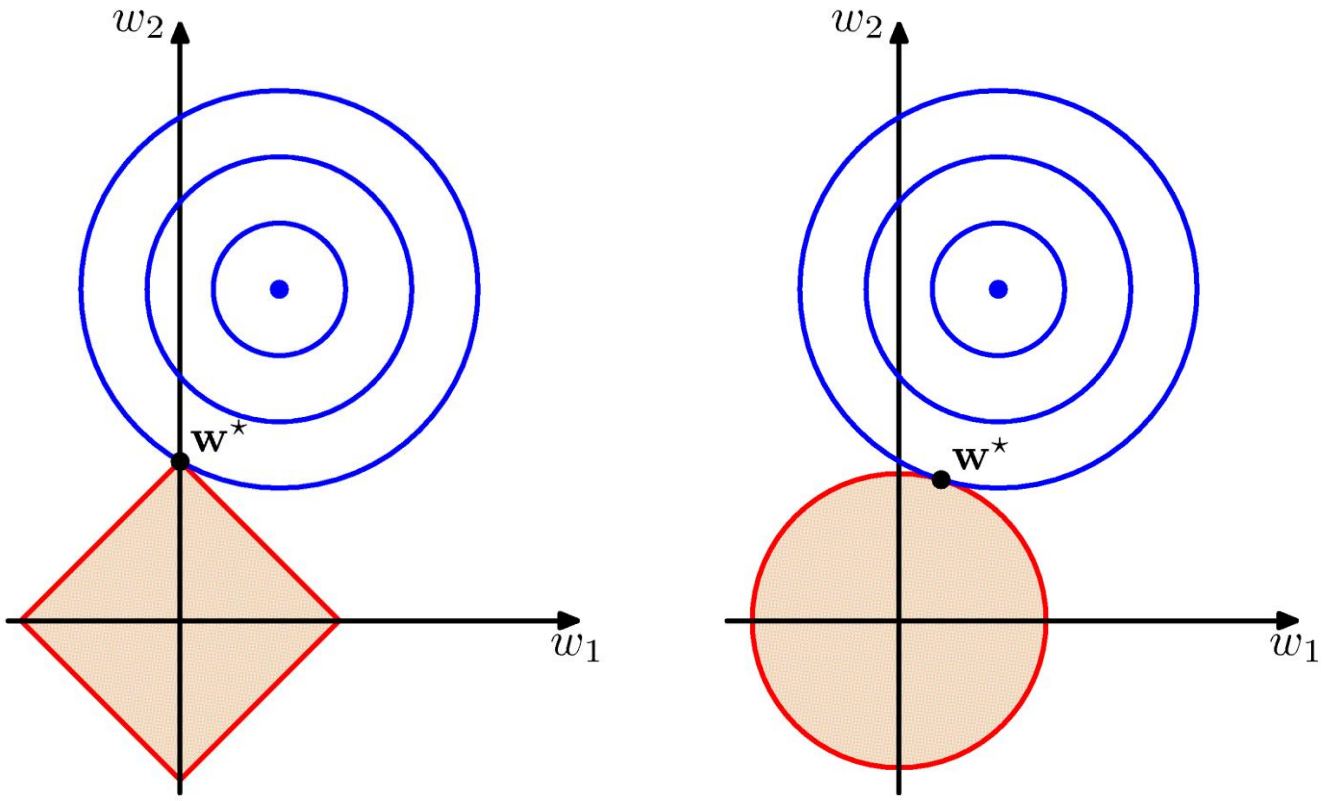
- For sufficiently large λ, some of the coefficients will be driven to zero, leading to a sparse model
- The above formulation is equivalent to:

$$\mathbf{w}^{lasso} = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{2} \underbrace{\sum_{n=1}^{N} \left(t_n - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n) \right)^2}_{\text{unregularized sum-of-squares error}}, \text{ subject to } \sum_{j=1}^{M-1} |w_j| \leq \tau.$$

- The two approaches are related using Lagrange multipliers
- The Lasso solution is a quadratic programming problem: can be solved efficiently

Lasso vs. Quadratic Penalty

Lasso tends to generate sparser solutions compared to a quadratic regularizer (sometimes called L_1 and L_2 regularizers)



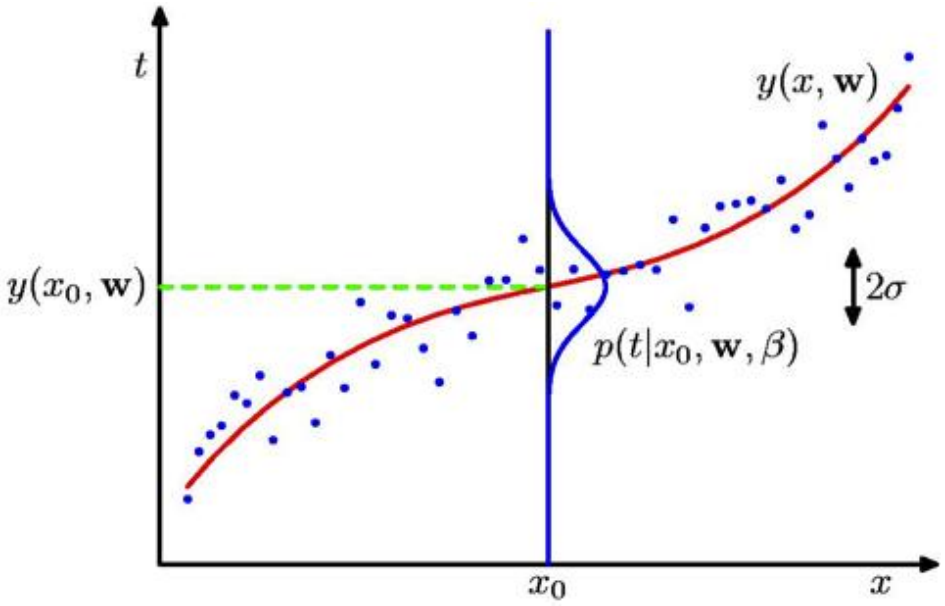
Probabilistic Perspective of Regularization

- So far we saw that polynomial curve fitting can be expressed in terms of error minimization. We now view it from probabilistic perspective.

- Suppose that our data arose from a statistical model:

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon,$$

where ε is a random error having Normal distribution with zero mean, and is independent of **x**



Thus we have:

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}),$$

where β is a precision parameter, corresponding to the inverse variance.

We will use probability distribution and probability density interchangeably. It should be obvious from the context.

Maximum-Likelihood Regularization

If the data are assumed to be independently and identically distributed (*i.i.d assumption*), the likelihood function takes the form:

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \prod_{i=1}^{N} \mathcal{N}(t_n|y(\mathbf{x}_n, \mathbf{w}), \beta^{-1}).$$

It is often convenient to maximize the log of the likelihood function:

$$\ln p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \underbrace{-\frac{\beta}{2} \sum_{n=1}^{N} (y(\mathbf{x}_n, \mathbf{w}) - t_n)^2}_{\beta E(\mathbf{w})} + \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi).$$

- Maximizing log-likelihood with respect to **w** (under the assumption of a Normal noise) is equivalent to minimizing the *sum-of-squared error* function.
- Determine $\mathbf{w}_{ML}$ by maximizing log-likelihood. Then maximizing w.r.t. β:

$$\frac{1}{\beta_{ML}} = \frac{1}{N} \sum_{n} (y(\mathbf{x}_n, \mathbf{w}_{ML}) - t_n)^2.$$

Predictive Distribution

Once we determined the parameters **w** and β, we can make prediction for new values of **x**:

$$p(t|\mathbf{x}, \mathbf{w}_{ML}, \beta_{ML}) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}_{ML}), \beta_{ML}^{-1}).$$