# GAUSSIAN PROCESS FOR REGRESSION AND CLASSIFICATION

Hsin-Min Lu
盧信銘
台大資管系

# Under the Hood of Gaussian Processes

- We have seen the intuition of Gaussian process, and the way we can do nonlinear regression using Gaussian processes (GPs)
- GPs are kernel machines. We are going to discuss the kernel functions in more details here.
- Moreover, we are going to look at more math behind the models.

# CONSTRUCTING KERNELS

# Kernel and Covariance Matrix

- In Gaussian Processes, kernel is used to generate the covariance matrix among any pair of outcomes (given their features).
- Start with definitions:
- A Gaussian process is a collection of random variables, any finite number of which have (consistent) Gaussian distributions.
- A Gaussian distribution is fully specified by a mean vector $\boldsymbol{\mu}$ and covariance matrix $\Sigma$:
- $\boldsymbol{f} = (f_1, f_2, \ldots, f_n) \sim N(\boldsymbol{\mu}, \Sigma)$
- The covariance matrix is determined by the kernel function: $\Sigma_{ij} = k(x_i, x_j) = \Sigma_{ji}$.

# Kernel Properties

- $K(\boldsymbol{x}, \boldsymbol{x}) = Cov\big(f(\boldsymbol{x}), f(\boldsymbol{x})\big) = Var\big(f(\boldsymbol{x})\big) \geq 0$
- $K(\boldsymbol{x}, \boldsymbol{y}) = K(\boldsymbol{y}, \boldsymbol{x}) = cov\big(f(\boldsymbol{x}), f(\boldsymbol{y})\big) = cov\big(f(\boldsymbol{y}), f(\boldsymbol{x})\big)$
  ➔ symmetric
- Kernel function need to be positive semi-definite.
- That is, $\boldsymbol{a}'K\boldsymbol{a} \geq 0$ for arbitrary real vector $a$.
- This property can be interpreted as follows:
- Let $t = \boldsymbol{a}'\boldsymbol{f}$, then $Var(t) = Var(\boldsymbol{a}'\boldsymbol{f}) = \boldsymbol{a}'Var(\boldsymbol{f})\boldsymbol{a} = \boldsymbol{a}'\Sigma\mathbf{a} \geq 0$.
- That is, the linear combination of $f_i$ need to have a nonnegative variance.

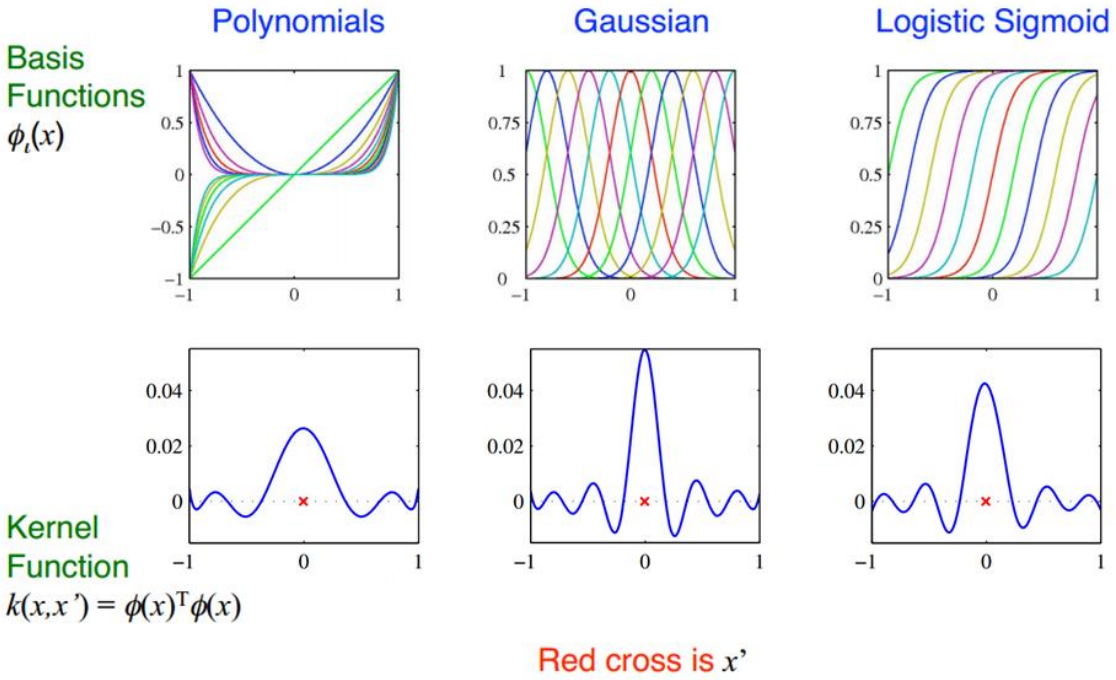
# Positive Semi-definite

- K is covariance ⟷ K is a positive semi-definite function.

- As long as you can show a function is positive semi-definite, then it can be a covariance matrix.
  - Not a very easy task!
- How do we create kernels?

# Creating Kernels via Feature Functions

- Given feature vector $\boldsymbol{x}$, we can extend the feature set via a basis function $\phi(\boldsymbol{x})$. The inner product of feature vector defines a kernel:
- $k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x}) = \sum_{i=1}^{M} \phi_i(\boldsymbol{x}) \phi_i(\boldsymbol{x}')$.



# Building New Kernels

- Given valid kernels $k_1(\boldsymbol{x}, \boldsymbol{x}')$ and $k_2(\boldsymbol{x}, \boldsymbol{x}')$ the following new kernels will be valid

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= c k_1(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= q\left(k_1(\mathbf{x}, \mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= \exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= k_3\left(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^{\mathrm{T}} \mathbf{A} \mathbf{x}' \\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b) \\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) k_b(\mathbf{x}_b, \mathbf{x}'_b)
\end{aligned}
$$

- $f(.)$ is any function
- $q(.)$ is a polynomial with non-negative coefficients
- $\phi(x)$ is a function from $x$ to $R^M$
- $k_3$ is a valid kernel in $R^M$
- A is a symmetric positive semi-definite matrix
- $x_a$ and $x_b$ are variables with $x = (x_a, x_b)$
- $k_a$ and $k_b$ are valid kernel functions

# Gaussian Kernel

- A commonly used kernel is $k(x, x') = \exp\left\{-\frac{\|x-x'\|^2}{2\sigma^2}\right\}$
- It is seen as a valid kernel by expanding the square

$$-\|\boldsymbol{x}-\boldsymbol{x}'\|^2 = \boldsymbol{x}^T\boldsymbol{x} + (\boldsymbol{x}')^T\boldsymbol{x}' - 2\boldsymbol{x}^T\boldsymbol{x}'$$

- To give

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\boldsymbol{x}^T\boldsymbol{x}/2\sigma^2)\exp(\boldsymbol{x}^T\boldsymbol{x}'/\sigma^2)\exp(-(\boldsymbol{x}')^T\boldsymbol{x}'/2\sigma^2)$$

- Since $\boldsymbol{x}^T\boldsymbol{x}'$ is a valid kernel, so is $\exp\left(\frac{\boldsymbol{x}^T\boldsymbol{x}'}{\sigma^2}\right)$ (rule 4)
- From Rule 2, we know $k(\boldsymbol{x}, \boldsymbol{x}')$ is valid.

# Kernel for Symbolic Inputs

- Kernel functions defined for graphs, sets, strings, and text documents.
- If $A_1$ and $A_2$ are two subsets of objects
- A simple kernel is

$$k(A_1, A_2) = 2^{|A_1 \cap A_2|}$$

- where $|\ |$ indicates cardinality of set intersection
- A valid kernel since it can be shown to correspond to an inner product in a feature space

# Kernels from Probabilities

- Combining Discriminative and Generative Models to benefit from both.
- Kernels based on Generative Models
- Given a generative model $p(\boldsymbol{x})$ we define a kernel by

$$k(\boldsymbol{x}, \boldsymbol{x}') = p(\boldsymbol{x})\, p(\boldsymbol{x}')$$

  - A valid kernel since it is an inner product in the one-dimensional feature space defined by the mapping $p(\boldsymbol{x})$
- Two inputs $\boldsymbol{x}$ and $\boldsymbol{x}'$ are similar if they have high probabilities

# Sigmoidal (non-)Kernel

- Provides a link between SVMs and neural networks

$$k(\boldsymbol{x}, \boldsymbol{x}') = \tanh(a\boldsymbol{x}T\boldsymbol{x}' + b)$$

  - $\tanh(z) = \frac{e^{2z}-1}{e^{2z}+1}$
  - Its Gram matrix is not positive semi-definite
  - But used in practice because it gives SVMs a superficial resemblance to neural networks
- Bayesian neural network with an appropriate prior reduces to a Gaussian process.

# GAUSSIAN PROCESS FOR REGRESSION

# Gaussian Processes for Regression

- We specify Gaussian Process directly over functions
  - Instead of considering distribution over weights w
- Take into account noise on observed target values as

$$t_n = y_n + \varepsilon_n \text{ where } y_n = y(x_n)$$

  - Noise process has a Gaussian distribution $p(t_n|y_n) = N(t_n|y_n, \beta^{-1})$
- Here $y_n$ is the value "unobservable function" given input $x_n$, we can only observe $y_n$ subject to a noise $\epsilon_n$.
- That is, we only see $t_n$.
- Assuming noise is independent for each data point
  - Joint distribution: $p(\boldsymbol{t}|\boldsymbol{y}) = N(\boldsymbol{t}|\boldsymbol{y}, \beta^{-1}I_N)$
  - Marginal distribution: $p(\boldsymbol{y}) = N(\boldsymbol{y}|0, K)$

# The Gram Matrix

- Marginal distribution of $\boldsymbol{t}$

$$p(\boldsymbol{t}) = \int p(\boldsymbol{t}|\boldsymbol{y})p(\boldsymbol{y})d\boldsymbol{y} = N(\boldsymbol{t}|0, C)$$

where $C(x_n, x_m) = k(x_n, x_m) + \beta^{-1}\delta_{nm}$

- The two Gaussian sources of randomness, $y(x)$ and $\varepsilon$ are independent, so their covariance simply add.
- In short $Cov(\boldsymbol{y}) = K$,
- $Cov(\boldsymbol{t}) = C = K + \beta^{-1}I$

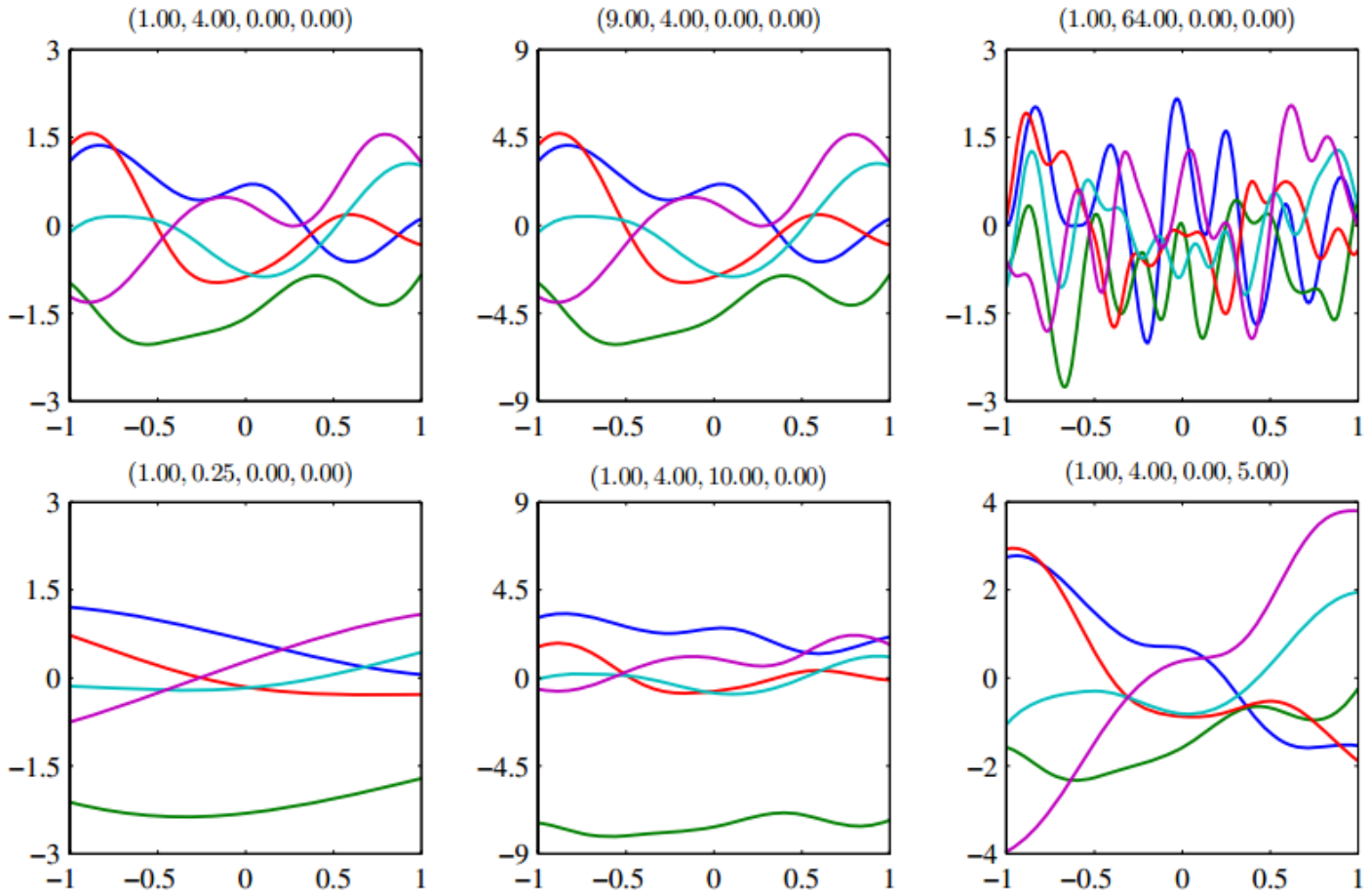
# A Familiar Kernel Function

- Widely used kernel function for Gaussian Process
- Exponential of a quadratic form

$$k(x_n, x_m) = \theta_0 \exp\left\{-\frac{\theta_1}{2} \| x_n - x_m \|^2\right\} + \theta_2 + \theta_3 x_n^T x_m$$

- Samples from this prior are plotted for various values (see next slide) of the parameters $\theta_0$, $\theta_1$, $\theta_2$, and $\theta_3$

# Realizations of Gaussian Process



# Making Predictions

- Goal is to predict target variable $t_{N+1}$ given $x_{N+1}$
- To find conditional distribution $p(t_{N+1}|t_N)$ we begin by writing down the joint distribution

$$p(t_{N+1}) = N(t_{N+1}|0, C_{N+1})$$

  - Where $C_{N+1}$ is the (N+1)X(N+1) covariance matrix with elements given by $C(x_n, x_m) = k(x_n, x_m) + \beta^{-1}\delta_{nm}$

$$C_{N+1} = \begin{pmatrix} C_N & \boldsymbol{k} \\ \boldsymbol{k}^T & c \end{pmatrix}$$

- Conditional distribution $p(t_{N+1}|t_N)$ is Gaussian with
  - Mean: $m(x_{N+1}) = \boldsymbol{k}^T C_N^{-1} \boldsymbol{t}$
  - Variance: $\sigma^2(x_{N+1}) = c - \boldsymbol{k}^T C_N^{-1} \boldsymbol{k}$
- Key results that define Gaussian Regression

# Learning the hyperparameters

- Prediction of a Gaussian process model will depend on the choice of covariance function.
- Rather than fixing the covariance function we can use a parametric family of functions and then infer parameter values from the data.

# Learning the hyperparameters

- Based on evaluation of likelihood function $p(t|\boldsymbol{\theta})$
  - where **θ** denotes the hyperparameters of the Gaussian process model
- Recall: $p(\boldsymbol{t}) = \int p(\boldsymbol{t}|\boldsymbol{y})p(\boldsymbol{y})d\boldsymbol{y} = N(\boldsymbol{t}|0, C)$
- Here $C$ is a function of $\boldsymbol{\theta}$.
- Point estimate of **θ** is obtained by maximizing log-likelihood

$$\ln p(\boldsymbol{t}|\boldsymbol{\theta}) = -\frac{1}{2}\ln|C_N| - \frac{1}{2}\boldsymbol{t}^T C_N^{-1}\boldsymbol{t} - \frac{N}{2}\ln(2\pi)$$

- Gradient of log-likelihood

$$\frac{\partial}{\partial\theta_i}\ln p(\boldsymbol{t}|\boldsymbol{\theta}) = -\frac{1}{2}\mathrm{tr}\left(C_N^{-1}\frac{\partial C_N}{\partial\theta_i}\right) + \frac{1}{2}\boldsymbol{t}^T C_N^{-1}\frac{\partial C_N}{\partial\theta_i}C_N^{-1}\boldsymbol{t}$$

- Find best $\boldsymbol{\theta}$ through gradient descent.

# Automatic Relevance Determination (ARD)

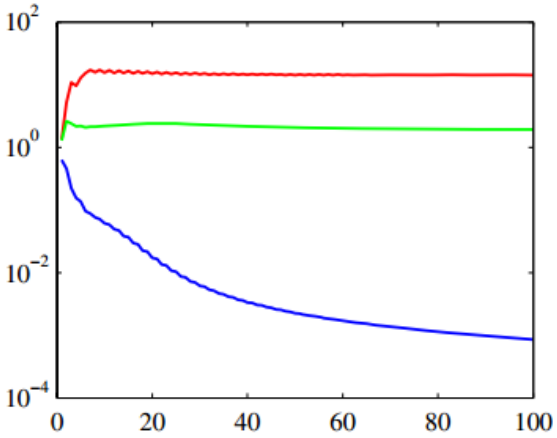
- Determining importance of variables
- Adjust kernel function so that each input feature is associated with a weight.

$$k(\boldsymbol{x}, \boldsymbol{x}') = \theta_0 \exp\{-\frac{1}{2}\sum_{i=1}^{2} \eta_i (x_i - x_i')^2\}$$

- Plug in $\ln p(\boldsymbol{t}|\boldsymbol{\theta})$ and search for best weights.
- As particular parameter $\eta_i$ becomes small, function becomes insensitive to corresponding variable $x_i$
- By adapting these parameters to a data set by MLE it becomes possible to detect variables that have little effect on predictive distribution.

# Example of ARD on Synthetic Dataset

- Three inputs $x_1, x_2, x_3$
- Target variable $t$ are generated by sampling 100 values of $x_1$ from a Gaussian, evaluating $\sin(2\pi x_1)$ and adding Gaussian noise
- $x_2 = x_1 + \text{noise}$
- $x_3$ are sampled from an independent distribution
- Estimate the weight for $x_1$, $x_2$, and $x_3$.




$\eta_1$ converges to a large value
$\eta_2$ converges to a much smaller value
$\eta_3$ becomes small indicating it is irrelevant for predicting t

# GAUSSIAN PROCESS FOR CLASSIFICATION

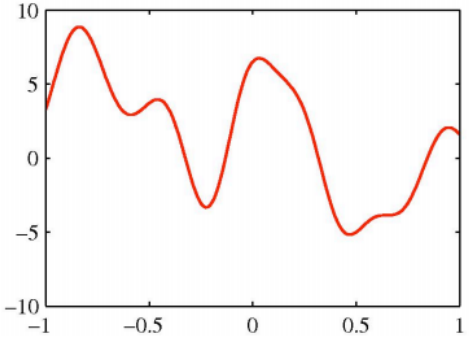
# Gaussian Processes for Classification

- Gaussian processes make predictions that lie on the entire real axis.
- For two-class classification we need to model posterior probabilities of the target variable for a new input variable to lie in the interval (0,1).
- Can adapt Gaussian processes for classification.
- ➔ Transforming output of Gaussian process using appropriate nonlinear activation function

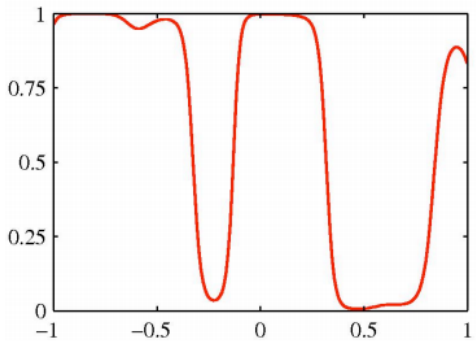
# Gaussian Processes for Classification

- Two class problem with target variable $t \in \{0,1\}$
- We define a Gaussian process over a function $a(x)$
- Then transform the function using a logistic sigmoid $y = \sigma(a)$
- Then we obtain a non-Gaussian stochastic process over functions $y(x)$ where $y \in \{0,1\}$

A sample from a Gaussian process prior over functions $a(x)$




Result of transforming this sample using a logistic sigmoid function



# Gaussian Processes for Classification

- Two-class Classification

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

- One-dimensional input space
  - Bernoulli distribution

$$p(t|a) = \sigma(a)^t (1 - \sigma(a))^{1-t}$$

- Transforming output on real line to (0,1) interval

- Training set samples $\boldsymbol{x}_1 \ldots \boldsymbol{x}_N$
- Corresponding target variables $\boldsymbol{t} = (t_1, \ldots, t_N)^T$
- Goal is to determine the predictive distribution $p(t_{N+1}|\boldsymbol{t}_N)$

# Gaussian Processes for Classification

- Define a Gaussian process over function $a(x)$
- $\boldsymbol{a}_{N+1} = [a(x_1), \ldots, a(x_{N+1})]^T$
- Gaussian process prior takes the form

$$p(\boldsymbol{a}_{N+1}) = N(\boldsymbol{a}_{N+1}|0, C_{N+1})$$

- Unlike regression, the covariance matrix no longer includes a noise term since all of the training data points are assumed to be correctly labeled.
- However, we usually add a positive constant to the diagonal elements for numerical stability.
- Covariance matrix has elements

$$C(x_n, x_m) = k(x_n, x_m) + v\delta_{nm}$$

# Gaussian Processes for Classification

- Predictive distribution is intractable

$$p(t_{N+1} = 1|\boldsymbol{t}_N) = \int p(t_{N+1} = 1|a_{N+1})p(a_{N+1}|\boldsymbol{t}_N)da_{N+1}$$

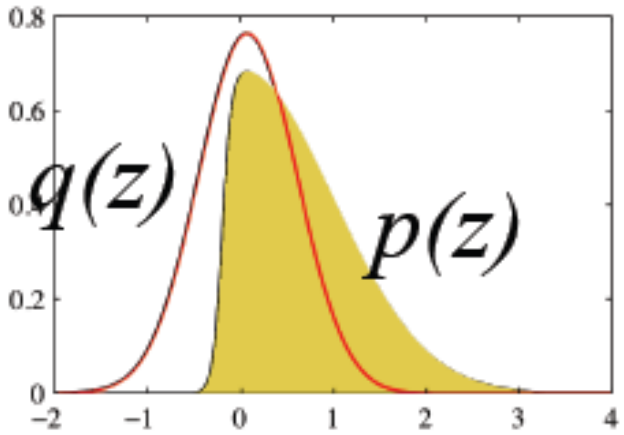
- Need to do approximation.
- We will consider Laplace approximation in the following discussion.

# Laplace Approximation: One-dimensional Case

- Consider single variable $z$ with distribution $p(z)$ defined by

  $p(z) = \frac{1}{Z} f(z)$,

  where $Z = \int f(z)dz$ is a normalization coefficient
  - $f(z)$ could be a scaled version of $p(z)$
  - $p(z)$ will be a pdf due to normalization
- Suppose that value of $Z$ is unknown
- Goal is to find Gaussian approximation $q(z)$ centered on the mode of the distribution $p(z)$



# Taylor Expansion centered at Mode

- (Consider one-dimensional case) Finding the mode of *p(z)*
  - A point $z_0$ such that $p'(z_0) = 0$
  - Equivalently $\frac{df(z)}{dz}|_{z=z_0} = 0$
- Logarithm of Gaussian is a quadratic.
- So use Taylor expansion of $\ln f(z)$ centered at mode $z_0$
- $\ln f(z) \approx \ln f(z_0) - \frac{1}{2} A(z - z_0)^2$
- $A = -\frac{d^2}{dz^2} \ln f(z)|_{z=z_0}$
- First order term does not appear since $z_0$ is a local maximum

# Final form of Laplacian (One Dimension)

- Approximation of *f(z)*, $\ln f(z) \approx \ln f(z_0) - \frac{1}{2}A(z - z_0)^2$
- Taking exponential $f(z) \approx f(z_0)\exp\left\{-\frac{A}{2}(z - z_0)^2\right\}$
- Normalization of a Gaussian
- $Z = \int f(z)dz \approx f(z_0)\int \exp\left\{-\frac{A}{2}(z - z_0)^2\right\}dz =$

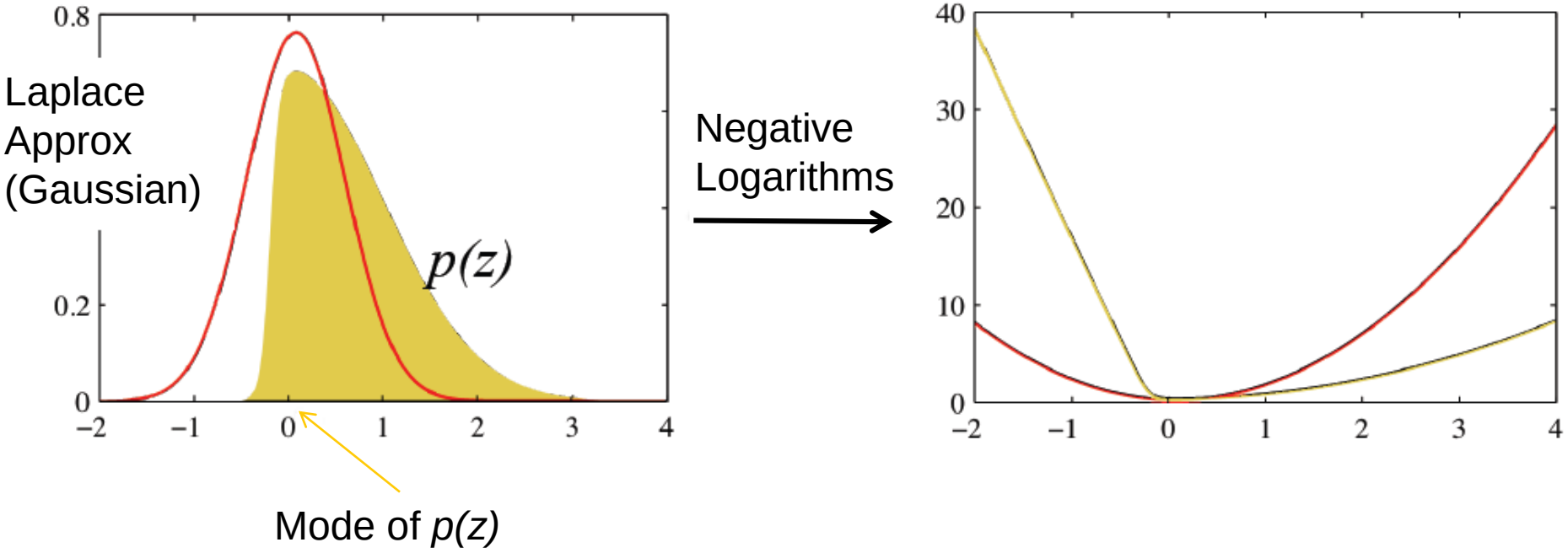
  $f(z_0)\frac{(2\pi)^{1/2}}{A^{1/2}}$
- $q(z) = \frac{1}{Z}f(z_0)\exp\left\{-\frac{A}{2}(z - z_0)^2\right\}$

$$= \left(\frac{A}{2\pi}\right)^{1/2} \exp\left\{-\frac{A}{2}(z - z_0)^2\right\}$$

- ➔ Univariate normal
  with mean = $z_0$ and variance = $\frac{1}{A}$

# Laplace Approximation Example

Applied to distribution $p(z) \propto \exp(-\frac{z^2}{2})\sigma(20z+4)$, where $\sigma$ is sigmoid




Gaussian approximation will only be well-defined if its precision A>0, or second derivative of *f(z)* at point $z_0$ is negative

# Back to Gaussian Process For Classification

- Want: $p(t_{N+1} = 1|\boldsymbol{t}_N) = \int p(t_{N+1} = 1|a_{N+1})p(a_{N+1}|\boldsymbol{t}_N)da_{N+1}$
- The first term comes from our model setting:

$$p(t_{N+1} = 1|a_{N+1}) = \sigma(a_{N+1})$$

- The second term, $p(a_{N+1}|\boldsymbol{t}_N)$,is more complicated.
- Using Bayesian Theorem:

$$\begin{aligned} p(a_{N+1}|\mathbf{t}_N) &= \int p(a_{N+1}, \mathbf{a}_N|\mathbf{t}_N)\,\mathrm{d}\mathbf{a}_N \\ &= \frac{1}{p(\mathbf{t}_N)} \int p(a_{N+1}, \mathbf{a}_N)p(\mathbf{t}_N|a_{N+1}, \mathbf{a}_N)\,\mathrm{d}\mathbf{a}_N \\ &= \frac{1}{p(\mathbf{t}_N)} \int p(a_{N+1}|\mathbf{a}_N)p(\mathbf{a}_N)p(\mathbf{t}_N|\mathbf{a}_N)\,\mathrm{d}\mathbf{a}_N \\ &= \int p(a_{N+1}|\mathbf{a}_N)p(\mathbf{a}_N|\mathbf{t}_N)\,\mathrm{d}\mathbf{a}_N \end{aligned}$$

# Posterior of $a_{N+1}$

- From the previous slide:

$$p(a_{N+1}|\boldsymbol{t}_N) = \int p(a_{N+1}|\boldsymbol{a}_N)p(\boldsymbol{a}_N|\boldsymbol{t}_N)da_N$$

- From Gaussian Process Regression, we know
- $p(a_{N+1}|\boldsymbol{a}_N) = \mathcal{N}\left(a_{N+1}\middle|\boldsymbol{k}^T C_N^{-1}\boldsymbol{a}_N, c - \boldsymbol{k}^T C_N^{-1}\boldsymbol{k}\right)$

- We also know $p(\boldsymbol{t}_N|\boldsymbol{a}_N)$ from the model setting:
- $p(\boldsymbol{t}_N|\boldsymbol{a}_N) = \prod_{n=1}^{N} \sigma(a_n)^{t_n}\left(1-\sigma(a_n)\right)^{1-t_n}$
- $= \prod_{n=1}^{N} e^{a_n t_n}\sigma(-a_n)$
- How can we approximate $p(\boldsymbol{a}_N|\boldsymbol{t}_N)$ using what we know?

# Laplace Approximation for $p(\boldsymbol{a}_N|\boldsymbol{t}_N)$

- Since $p(\boldsymbol{a}_N|\boldsymbol{t}_N) = \frac{p(\boldsymbol{a}_N,\boldsymbol{t}_N)}{p(\boldsymbol{t}_N)} \propto p(\boldsymbol{a}_N,\boldsymbol{t}_N) = p(\boldsymbol{t}_N|\boldsymbol{a}_N)p(\boldsymbol{a}_N)$
- We can do Laplace approximation of $p(\boldsymbol{a}_N|\boldsymbol{t}_N)$ using $p(\boldsymbol{t}_N|\boldsymbol{a}_N)p(\boldsymbol{a}_N)$.
- To do Laplace approximation, need to compute the gradient and Hessian matrix of $\Psi(\boldsymbol{a}_N) = \ln[p(\boldsymbol{t}_N|\boldsymbol{a}_N)p(\boldsymbol{a}_N)]$.

$$\begin{aligned}\Psi(\mathbf{a}_N) &= \ln p(\mathbf{a}_N) + \ln p(\mathbf{t}_N|\mathbf{a}_N) \\ &= -\frac{1}{2}\mathbf{a}_N^{\mathrm{T}}\mathbf{C}_N^{-1}\mathbf{a}_N - \frac{N}{2}\ln(2\pi) - \frac{1}{2}\ln|\mathbf{C}_N| + \mathbf{t}_N^{\mathrm{T}}\mathbf{a}_N \\ &\quad -\sum_{n=1}^{N}\ln(1+e^{a_n}) + \text{const.}\end{aligned}$$

# Gradient and Hessian of $\Psi(\boldsymbol{a}_N)$

$$\begin{aligned}\Psi(\mathbf{a}_N) &= \ln p(\mathbf{a}_N) + \ln p(\mathbf{t}_N|\mathbf{a}_N) \\ &= -\frac{1}{2}\mathbf{a}_N^{\mathrm{T}}\mathbf{C}_N^{-1}\mathbf{a}_N - \frac{N}{2}\ln(2\pi) - \frac{1}{2}\ln|\mathbf{C}_N| + \mathbf{t}_N^{\mathrm{T}}\mathbf{a}_N \\ &\quad - \sum_{n=1}^{N}\ln(1+e^{a_n}) + \text{const.}\end{aligned}$$

- Gradient:
- $\nabla\Psi(\boldsymbol{a}_N) = \frac{\partial\Psi(\boldsymbol{a}_N)}{\partial \boldsymbol{a}_N}$
- $= \boldsymbol{t}_N - \boldsymbol{\sigma}_N - C_N^{-1}\boldsymbol{a}_N$
- where $\sigma_N = [\sigma(a_1)\,\sigma(a_2)\ \ldots\sigma(a_N)]^T$
- We cannot simply set the gradient to 0 and find the location of mode because $\boldsymbol{\sigma}_N$ depends on $\boldsymbol{a}_N$.

- Need to solve the system numerically via an iterative reweighted least square (IRLS) algorithm.

# Gradient and Hessian of $\Psi(\boldsymbol{a}_N)$

- IRLS is in fact a Newton-Raphson algorithm, it requires the hessian:
- $\nabla\nabla\Psi(\boldsymbol{a}_N) = \frac{\partial\nabla\Psi(a_N)}{\partial a_N^T} = -W_N - C_N^{-1}$
- where $W_N$ is a diagonal element with

$$\mathrm{W_{nn}} = \sigma(a_n)\big(1-\sigma(a_n)\big)$$

- Note that $W_{nn} \in (0, \frac{1}{4})$, and $\det(W) = \prod_{ii} W_{ii} > 0$, $W$ is positive definite; $C_N$ is positive definite by construction.
- $\det\big(\nabla\nabla\Psi(a_N)\big) < 0$ ➔ the function is globally log concave ➔ global maximum exists.

# Iterative Update

- Recall: $\nabla\Psi(\boldsymbol{a}_N) = \boldsymbol{t}_N - \boldsymbol{\sigma}_N - C_N^{-1}\boldsymbol{a}_N$
- $\nabla\nabla\Psi(\boldsymbol{a}_N) = \frac{\partial\nabla\Psi(a_N)}{\partial a_N^T} = -W_N - C_N^{-1}$
- To search for global maximum, start with some initial $a_N$, and iterative update via the Newton-Raphson formula:
- $\boldsymbol{a}^{(new)} = \boldsymbol{a}^{(old)} - H^{-1}\nabla\Psi(\boldsymbol{a})$
- In this case:
- $\boldsymbol{a}_N^{new} = C_N(I + W_N C_N)^{-1}\{\boldsymbol{t}_N - \boldsymbol{\sigma}_N + W_N a_N\}$
- At the mode, $\nabla\Psi(\boldsymbol{a}_N) = 0$, and we have

$$a_N^* = C_N(t_N - \sigma_N)$$

# Laplace Approximation

- After reaching the mode, we can use Laplace approximation and write the approximating pdf as:
- $q(\boldsymbol{a}_N) = \mathcal{N}(\boldsymbol{a}_N|\boldsymbol{a}_N^*, H^{-1})$,
- where $\boldsymbol{a}_N^* = C_N(\boldsymbol{t}_N - \boldsymbol{\sigma}_N)$, $H = -\nabla\nabla\Psi(\boldsymbol{a}_N) = W_N + C_N^{-1}$

- This is how we approximate $p(\boldsymbol{a}_N|\boldsymbol{t}_N)$.

# Getting back to what we want…

- Recall that we try to estimate:

$$p(a_{N+1}|\boldsymbol{t}_N) = \int p(a_{N+1}|\boldsymbol{a}_N)p(\boldsymbol{a}_N|\boldsymbol{t}_N)da_N$$

- From Gaussian Process Regression, we know
- $p(a_{N+1}|\boldsymbol{a}_N) = \mathcal{N}\left(a_{N+1}\middle|\boldsymbol{k}^T C_N^{-1}\boldsymbol{a}_N, c - \boldsymbol{k}^T C_N^{-1}\boldsymbol{k}\right)$,
- Also, from the previous slide, we know
- $p(\boldsymbol{a}_N|\boldsymbol{t}_N) \approx q(\boldsymbol{a}_N) = \mathcal{N}(\boldsymbol{a}_N|C_N(\boldsymbol{t}_N - \boldsymbol{\sigma}_N), \left(W_N + C_N^{-1}\right)^{-1})$,
- Need to do the integral to find $p(a_{N+1}|\boldsymbol{t}_N)$. However, because of the nice property of Gaussian distribution, we can simply plug in the mean of $p(\boldsymbol{a}_N|\boldsymbol{t}_N)$ into $p(a_{N+1}|\boldsymbol{a}_N)$ to get the mean of $p(a_{N+1}|\boldsymbol{t}_N)$: $\boldsymbol{k}^T(\boldsymbol{t}_N - \boldsymbol{\sigma}_N)$

# Getting $p(a_{N+1}|\boldsymbol{t}_N)$

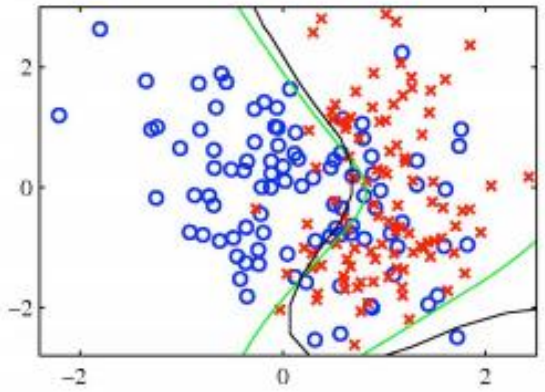
- If you go through the math, we can get the mean and variance of $p(a_{N+1}|\boldsymbol{t}_N)$ as:
- $E[a_{N+1}|t_N] = \boldsymbol{k}^T(\boldsymbol{t}_N - \boldsymbol{\sigma}_N)$
- $Var[a_{N+1}|t_N] = c - \boldsymbol{k}^T\left(W_N^{-1} + C_N\right)^{-1}\boldsymbol{k}$

- How do make prediction?
- Since: $p(t_{N+1} = 1|\boldsymbol{t}_N) = \int p(t_{N+1} = 1|a_{N+1})p(a_{N+1}|\boldsymbol{t}_N)da_{N+1}$
- We can just plug in the mean of $p(a_{N+1}|\boldsymbol{t}_N)$ into $p(t_{N+1} = 1|a_{N+1})$ to get a point estimator.
- This estimator is OK if we are using prob=0.5 as the cut-off point. Otherwise, need to consider the variance as well.

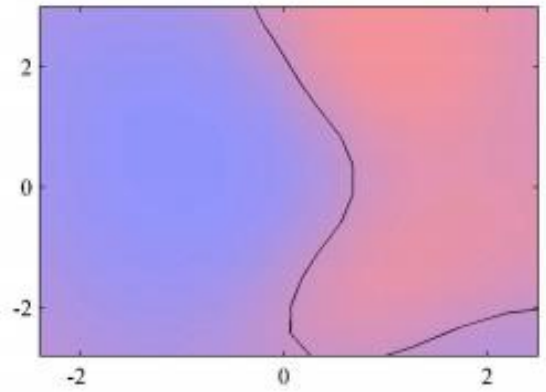
# Illustration of Gaussian process for classification

- Estimating classification boundary using Laplace approximation.

Green: Optimal boundary
Black: Gaussian Process Classifier




Predicted posterior probability together with Gaussian process decision boundary



# Connection to Neural Networks

- Consider a three-layer neural network with M hidden units in the middle layer.
- If $M \to \infty$, then the output tends to a Gaussian process.
- Thus there are theoretical connection between neural net and Gaussian process.
- Still in development: Deep Gaussian process network…