Introduction to Machine Learning & Deep Learning - Part 2

Sorbonne Université- Master informatique DAC et Master mathématiques et applications M2A-

P. Gallinari, patrick.gallinari@lip6.fr, http://www-connex.lip6.fr/~gallinar/

Année 2022-2023

Kernels etc.

Kernels
Support Vector Machines
Gaussian Processes
Neural Processes

Kernel Methods – a brief introduction

Introducing kernels

- The concept of kernels is important in machine learning
- It allows to derive general families of ML methods
  - Applicable to generic ML problems: supervised, unsupervised, ranking, ..
  - That can be used on different types of data (vectors, strings, graphs, …)
- It provides a general framework for the formal analysis of complex algorithms
  - e.g. NN in the infinite limit (infinite number of hidden cells) can be modeled and then analyzed as kernel methods

- Kernels, and over all support vector machines have been one of the main ML paradigm in 1995-2005.
  - The concept allows to make use and to formalize several important ideas concerning e.g. optimization (convex optimization), generalization
  - Most kernel methods are not well adapted to high dimensional spaces and large datasets, they failed in this sense but remain an important concept

Intuition (1) – kernels as similarity measures

- Kernel exploit similarity measures between data representations
  - Expressed as dot products in a feature space

  - **Feature space** - Let X be a set (e.g. the set of objects to be classified), we will represent these objects in a feature space $\mathcal{H}$, which is a vector space equiped with a dot product.
    - For that we will use a map Φ:

      $$\Phi: X \rightarrow \mathcal{H}$$
      $$x \longmapsto \Phi(x)$$

  - **Similarity measure** - we define a similarity measure via the dot product in $\mathcal{H}$:
    - $K(x, x') \ = <\Phi(x), \Phi(x')>$
    - In the following, $K(.,.)$will be called a kernel
    - Note: X can be any set, and not only a subset of $\mathbb{R}^n$
      - i.e. it may be endowed with a dot product itself or not, e.g. think of X as a set of books or proteins
      - Even when $X \subset \mathbb{R}^n$, i.e. a dot product space, the mapping Φ will allow us to define more complex (non linear) representations of $x \in X$

Intuition (2) – machine learning algorithms and dot products

- Several machine learning algorithms can be expressed using dot products in a feature space
  - We introduce two simple examples
    - Perceptron
    - Linear regression
    - This idea can be generalized to many families of supervised and unsupervised methods

Intuition (2) – machine learning algorithms and dot products
Example 1: Perceptron dual formulation for binary classification

Training set $D = \{(\boldsymbol{x}^1, y^1), \ldots, (\boldsymbol{x}^N, y^N)\}$, $\boldsymbol{x}^i \in \mathbb{R}^n$, $y^i \in \{-1,1\}$, hyp: the classes are linearly separables

Perceptron – primal formulation

```
Initialize $\boldsymbol{w}(0) = 0$
Repeat (t)
    choose example, $(\boldsymbol{x}(t), y(t))$
    if $y(t)\boldsymbol{w}(t).\boldsymbol{x}(t) \leq 0$
    then $\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + y(t)\boldsymbol{x}(t)$
until convergence
```

Decision function- primal

$$F(x) = sgn(\sum_{j=0}^{n} w_j x_j),$$

$$\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i \boldsymbol{y}^i \boldsymbol{x}^i$$

α_i : number of times for which the algorithm made a classification error on $\boldsymbol{x}^i$

Perceptron – dual formulation

```
Initialize $\boldsymbol{\alpha} = \mathbf{0}, \boldsymbol{\alpha} \in \boldsymbol{R}^N$
Repeat (t)
    choose an example, $(\boldsymbol{x}(t), y(t))$
    let $k$: $\boldsymbol{x}(t) = \boldsymbol{x}^k$
    if $y(t) \sum_{i=1}^{N} \alpha_i \boldsymbol{y}^i \boldsymbol{x}^i . \boldsymbol{x}(t) \leq 0$
    then $\alpha_k = \alpha_k + 1$
until convergence
```

Decision function - dual

$$F(x) = sgn(\sum_{i=1}^{N} \alpha_i \boldsymbol{y}^i \boldsymbol{x}^i . \boldsymbol{x})$$

Gram matrix $\boldsymbol{K}$:

matrix NxN with term i, j : $\boldsymbol{K}_{ij} = \boldsymbol{x}^i . \boldsymbol{x}^j$

similarity matrix between the training data

Intuition (2) – machine learning algorithms and dot products
Example 1: Perceptron dual formulation for binary classification

- In the dual formulation of the Perceptron
  - The decision function writes as $F(x) = sgn\left(\sum_{i=1}^{N} \alpha_i \boldsymbol{y}^{\boldsymbol{i}} \boldsymbol{K}(\boldsymbol{x}^{\boldsymbol{i}}, \boldsymbol{x})\right)$
  - With the kernel $\boldsymbol{K}\left(\boldsymbol{x}^{\boldsymbol{i}}, \boldsymbol{x}\right) =< \boldsymbol{x}^{\boldsymbol{i}}, \boldsymbol{x} >$, i.e. the kernel is computed directly in the input domain
    - What if we make use of another similarity function $\boldsymbol{K}\left(\boldsymbol{x}^{\boldsymbol{i}}, \boldsymbol{x}\right)$ instead of the canonical dot product?
  - The $\alpha_i s$ can be considered as a dual representation of the hyperplane normal vector

Intuition (2) – machine learning algorithms and dot products Example 2: dual formulation for regression

- Training examples
  - $D = \{(\boldsymbol{x}^1, y^1), \dots, (\boldsymbol{x}^N, y^N)\}$, we denote $X = \{\boldsymbol{x}^1, \dots, \boldsymbol{x}^N\}$
- Let us consider a linear model for regression
  - $f(\boldsymbol{x}) = \boldsymbol{w}.\boldsymbol{x}$
  - Let $\boldsymbol{x}^\perp \in X^\perp$, with $X^\perp$ the orthogonal set of X
  - $(\boldsymbol{w} + \boldsymbol{x}^\perp).\boldsymbol{x}^{\mathrm{i}} = \boldsymbol{w}.\boldsymbol{x}^{\mathrm{i}}, \forall \boldsymbol{x}^{\mathrm{i}} \in X$
  - Adding to $\boldsymbol{w}$ a component outside the space generated by X, has no effect on the linear regression prediction for all the **data in the training set**
  - If the training criterion only depends on the regression performed on the training data, as is usually the case, it is not needed to consider components of $\boldsymbol{w}$ outside the space generated by X
  - $\boldsymbol{w}$ can thus be writen under the form
    - $\boldsymbol{w} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} \alpha_i \boldsymbol{x}^{\boldsymbol{i}}$
    - The parameters $\alpha_i, i = 1 \dots N$ are called dual parameters
  - The regression function can then be directly written under a dual form using dot product:
    - $\boldsymbol{f}(\boldsymbol{x}) = \sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} \boldsymbol{\alpha_i} < \boldsymbol{x}^{\boldsymbol{i}}, \boldsymbol{x} >$

Intuition (2) – machine learning algorithms and dot products
Example 2: dual formulation for regression

- What if we make use of another similarity function $\boldsymbol{K}\left(\boldsymbol{x}^{\boldsymbol{i}}, \boldsymbol{x}\right)$ instead of the canonical dot product?
  - More generally, let us consider a regression defined through the mapping $\phi(x)$:
  - $f(\boldsymbol{x}) = \boldsymbol{w}.\phi(\boldsymbol{x})$
  - The solution will be in the space spanned by $\{\phi(\boldsymbol{x}^1), \dots, \phi(\boldsymbol{x}^{\mathrm{N}})\}$
  - $\boldsymbol{w} = \sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{N}} \alpha_i \phi(\boldsymbol{x}^{\boldsymbol{i}})$
  - $f(x) = \sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{N}} \boldsymbol{\alpha_i} < \boldsymbol{\phi}(\boldsymbol{x}^{\boldsymbol{i}}), \boldsymbol{\Phi}(\mathbf{x})) > = \sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{N}} \alpha_i \, \boldsymbol{K}(\boldsymbol{x}^{\boldsymbol{i}}, \boldsymbol{x})$
  - $\boldsymbol{K}\left(\boldsymbol{x}^{\boldsymbol{i}}, \boldsymbol{x}^{\boldsymbol{j}}\right) =< \boldsymbol{\phi}(\boldsymbol{x}^{\boldsymbol{i}}), \boldsymbol{\phi}(\boldsymbol{x}^{\boldsymbol{j}}) >= \boldsymbol{K}_{\boldsymbol{ij}}$
  - $\boldsymbol{K} = \left[\boldsymbol{K}_{\boldsymbol{ij}}\right]$ is the Gram matrix

Intuition – Summary

- Linear ML methods have a dual representation and can be formulated using dot products in a vector space
  - Examples: adaline, regression, ridge regression, etc
  - The information on the training data is provided by the Gram matrix $\boldsymbol{K}$:

$$\boldsymbol{K} = \left(\boldsymbol{K_{ij}}\right)_{\boldsymbol{i,j=1\dots N}} = \left(K\left(x^i, x^j\right)\right)_{\boldsymbol{i,j=1\dots N}}$$

  - With

$$K(x, \mathrm{x}') = < \Phi(x), \Phi(\mathrm{x}') >$$

$$\Phi: X \rightarrow \mathcal{H}$$

$$x \longmapsto \Phi(x)$$

  - Such a function $K(.,.)$ defined by a dot product in a feature space will be called a kernel
  - For supervised problems, the decision/ regression function $F(x)$ writes as a linear combination of scalar products:

$$\boldsymbol{F(x)} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} \alpha_i \, K(x^i, x)$$

Kernels

- After this informal introduction, we will introduce some formal arguments for characterizing kernels that admit a dot product representation in a feature space
- We first introduce some examples motivating the usefulness of kernels
- We then address the following question:
  - What kind of function $K(x, x')$ admits a representation as a dot product in a feature space $K(x, \mathrm{x}') = < \Phi(x), \Phi(\mathrm{x}') >$

Definitions

- Gram matrix
  - Given a function $K: XxX \rightarrow \mathbb{R}$, and a dataset X= $\{x^1, \ldots, \mathrm{x}^\mathrm{N}\}$, the NxN matrix with element $K_{ij} = K(x^i, x^j)$ is called the Gram matrix of K with respect to X
- Positive semi-definite matrix
  - A symmetric matrix $\boldsymbol{K}$ is positive semi-definite if its eigenvalues are all non negative – or equivalently if $x^T \boldsymbol{K} x > 0 \; \forall x \in X$

Positive definite kernels

- A positive definite kernel on set X, is a function $K: X \times X \to \mathbb{R}$
  - that is symmetric:

$$K(x, x') = K(x', x)$$

  - Which satisfies, $\forall N \in \mathbb{N}, \forall \left(x^1, \dots, x^N\right) \in \mathrm{X}^N$ and $\forall \left(a_1, \dots, a_N\right) \in \mathbb{R}^N$:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j K(x^i, x^j) \geq 0$$

- Note:
  - this is the general definition of a positive definite function
  - Positive definiteness allows an easy characterization of kernels
- Alternative definition with the similarity matrix of a p.d. kernel
  - A kernel K is p.d. if and only if, $\forall N \in \mathbb{N}, \forall \left(x^1, \dots, x^N\right) \in \mathrm{X}^N$, the similarity matrix $\boldsymbol{K_{ij}} = K(\boldsymbol{x^i}, \boldsymbol{x^j})$ is positive semi-definite
  - Note: this should be true $\forall N \in \mathbb{N}$

Examples of p.d. kernels

- Linear kernel
  - Let $X = \mathbb{R}^n$, the function $K: X^2 \to \mathbb{R}$:

$$(x, x') \to K(x, x') = < x, x' >_{\mathbb{R}^n}$$

is a p.d. kernel

Proof

- $< x, x' >_{\mathbb{R}^n} = < x', x >_{\mathbb{R}^n}$
- $\sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j < x^i, x^j >_{\mathbb{R}^n} = \left|\left|\sum_{i=1}^{N} a_i x^i\right|\right|_{\mathbb{R}^n}^2 \geq 0$

More general kernels

- More generally: kernels as dot product in an inner product space
- Lemma
  - Let X be any set, $\Phi: X \rightarrow \mathbb{R}^n$, the function $K: X^2 \rightarrow \mathbb{R}$:

$$(x, x') \rightarrow K(x, x') =< \Phi(x), \Phi(x') >_{\mathbb{R}^n}$$

is a p.d. kernel

Proof: same as above

$$< \Phi(x), \Phi(x') >_{\mathbb{R}^n} =< \Phi(x'), \Phi(x)) >_{\mathbb{R}^n}$$

$$\sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j < \Phi(x^i), \Phi(x^j) >_{\mathbb{R}^n} = \left\| \sum_{i=1}^{N} a_i \Phi(x^i) \right\|_{\mathbb{R}^n}^2 \geq 0$$

More general kernels
Example: Polynomial Kernel

- Consider a 2 dimensional input space $X \subset \mathbb{R}^2$and
- $\Phi: \mathbb{R}^2 \rightarrow \mathbb{R}^3, \Phi(x) = \Phi(x_1, x_2) = (x_1^2, x_2^2, \sqrt{2} x_1 \, x_2)$

  $K(x, x') = < \Phi(x), \Phi(x') >_{\mathbb{R}^3}$

  $K(x, x') = x_1^2 x'^2_1 + 2x_1 \, x_2 x'_1 \, x'_2 + x_2^2 x'^2_2$

  $K(x, x') = < \mathrm{x}, \mathrm{x}' >^2_{\mathbb{R}^2}$
- Note:
  - $K(x, x') = < \Phi(x), \Phi(x') >$ can be computed directly as $< \mathrm{x}, \mathrm{x}' >^2_{\mathbb{R}^2}$ without explicitely evaluating their coordinate in the feature space
  - Cheaper to compute in the original space than in the feature space
  - The same kernel is obtained with $\Phi(x_1, x_2) = (x_1^2, x_2^2, x_1 \, x_2, x_2 \, x_1)$ and a dot product in $\mathbb{R}^4$
    - Shows that the feature space is not uniquely determined by the kernel function

Example: Polynomial Kernel

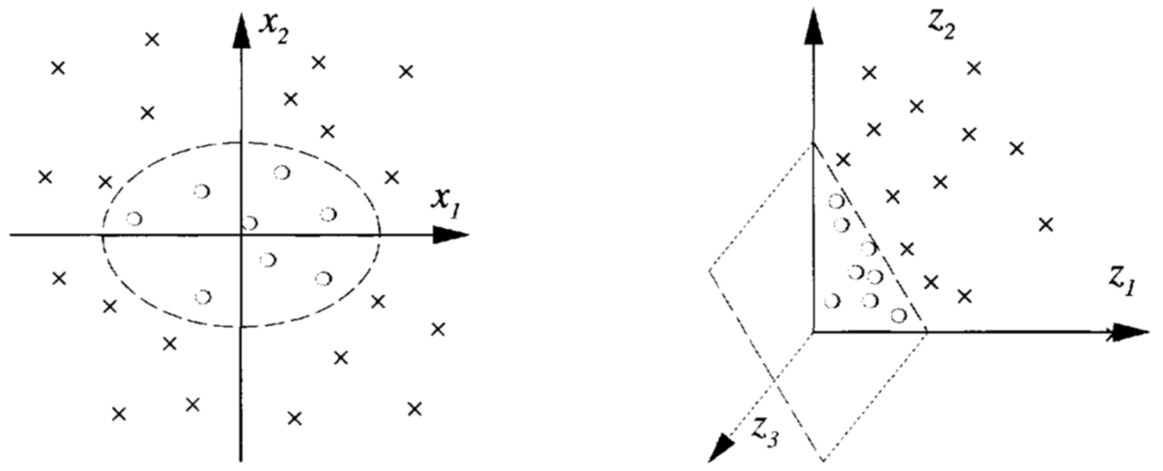


Figure 2.1 Toy example of a binary classification problem mapped into feature space. We assume that the true decision boundary is an ellipse in input space (left panel). The task of the learning process is to estimate this boundary based on empirical data consisting of training points in both classes (crosses and circles, respectively). When mapped into feature space via the nonlinear map $\Phi_2(x) = (z_1, z_2, z_3) = ([x]_1^2, [x]_2^2, \sqrt{2}\,[x]_1[x]_2)$ (right panel), the ellipse becomes a hyperplane (in the present simple case, it is parallel to the z_3 axis, hence all points are plotted in the (z_1, z_2) plane). This is due to the fact that ellipses can be written as linear equations in the entries of (z_1, z_2, z_3). Therefore, in feature space, the problem reduces to that of estimating a hyperplane from the mapped data points. Note that via the polynomial kernel (see (2.12) and (2.13)), the dot product in the three-dimensional space can be computed without computing Φ_2. Later in the book, we shall describe algorithms for constructing hyperplanes which are based on dot products (Chapter 7).

Scholkopf et al. 2002

Characterization of kernels

- Up to now kernels have been characterized by explicitely defining a mapping in a feature space and then computing an inner product in this space
- We will introduce an alternative characterization of a kernel
  - It is one of the main theoretical tools to characterize kernels
  - Without explicitely defining the feature space (i.e. Φ)

Characterization of kernels
Definitions and properties

- Inner product
  - Let $\mathcal{H}$ a vector space over $\mathbb{R}$, a function $<.,.>_{\mathcal{H}}$ is said to be an inner product on $\mathcal{H}$ if
    - $< \alpha_1 f_1 + \alpha_2 f_2, g >_{\mathcal{H}} = \alpha_1 < f_1, g >_{\mathcal{H}} + \alpha_2 < f_2, g >_{\mathcal{H}}$ linear (bilinear)
    - $< f, g >_{\mathcal{H}} = < g, f >_{\mathcal{H}}$ symmetric
    - $< f, f >_{\mathcal{H}} \geq 0$ and $< f, f >_{\mathcal{H}} = 0$ iff $f = 0$
    - We can then define a norm on $\mathcal{H}$ as $||f||_{\mathcal{H}} = \sqrt{< f, f >_{\mathcal{H}}}$
  - $\mathcal{H}$ endowed with an inner product is an inner product space
- Hilbert space
  - Is an inner product space $\mathcal{H}$ with the additional properties that it is separable and complete i.e. any Cauchy sequence in $\mathcal{H}$ converges in $\mathcal{H}$
    - A Cauchy sequence (f_n) is a sequence whose elements become progressively arbitray close to each other

$$\lim_{m>n, n\to\infty} ||f_n - f_m|\Big|_{\mathcal{H}} = 0$$

    - $\mathcal{H}$ is separable if for any $\epsilon > 0$ there exists a finite set of elements of $\mathcal{H}, \{, f_1, \dots, f_N\}$ such that for all $f \in \mathcal{H}$,

$$\min_i \Big| |f_i - f| \Big|_{\mathcal{H}} < \epsilon$$

Characterization of kernels
Definitions and properties

- Cauchy-Schwartz inequality for dot products
  - In an inner product space
    - $< x, x' >^2 \leq \|x\|^2 \|x'\|^2$
- Cauchy-Schwartz inequality for kernels
  - If K is a p.d. kernel and $x_1, x_2 \in X$, then:

$$\left|K\left(x^1, x^2\right)\right|^2 \leq K\left(x^1, x^1\right). K(x^2, x^2)$$

Characterization of kernels

- Theorem
  - $K: X\text{x}X \rightarrow \mathbb{R}$ is a p.d. kernel on X if and only if there exists a Hilbert space $\mathcal{H}$ and a mapping $\Phi: X \rightarrow \mathcal{H}$ such that:

$$\forall x, x' \in X, K(x, x') =< \Phi(x), \Phi(x') >_{\mathcal{H}}$$

- Central result that establish a link between kernels defined as dot products in a feature vector space and positive definite functions
- In order to demonstrate this result, we explicitely construct the feature -Hilbert- space

Characterization of kernels

- Assumption: K is a p.d. kernel
- Objective: construct an appropriate Hilbert space and a mapping Φ
- Defining the mapping Φ
  - Let us define $\Phi: X \rightarrow \mathbb{R}^X$, where $\mathbb{R}^X := \{f: X \rightarrow \mathbb{R}\}$ is the space of functions mapping X into $\mathbb{R}$ as:

    $\Phi: X \rightarrow \mathbb{R}^X$

    $x \longmapsto K(.,x)$

    $\Phi(\mathrm{x})$ denotes a function that assigns a value $K(x',x)$ to $x' \in X$, i.e. $\Phi(x)(.) = K(.,x)$

    To each point x in the X space, one associates a function $\Phi(x) = K(.,x)$

    This function will be a point in a vector space

    See Fig. next slide

Characterization of kernels

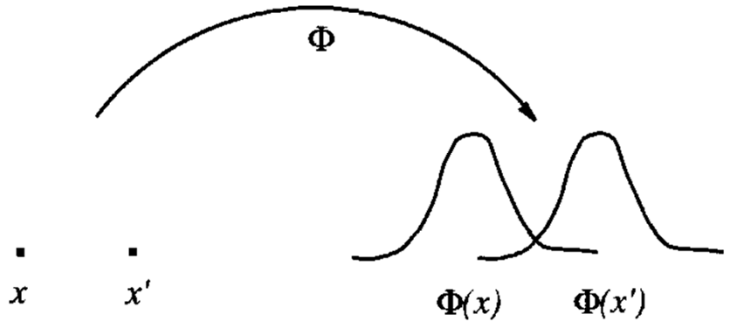


Figure 2.2 One instantiation of the feature map associated with a kernel is the map (2.21), which represents each pattern (in the picture, x or x') by a kernel-shaped *function* sitting on the pattern. In this sense, each pattern is represented by its similarity to *all* other patterns. In the picture, the kernel is assumed to be bell-shaped, e.g., a Gaussian $k(x, x') = \exp(-\|x - x'\|^2 / (2\,\sigma^2))$. In the text, we describe the construction of a dot product $\langle .,. \rangle$ on the function space such that $k(x, x') = \langle \Phi(x), \Phi(x') \rangle$.

Fig, Scholkopf et al. 2002

Characterization of kernels

- Construction of the feature space
- Let us consider the space of functions
- $\mathcal{H} = \{\sum_{i=1}^{m} \alpha_i K(., x^i) : m \in \mathbb{N}, x_i \in X, \alpha_i \in \mathbb{R}, i = 1 \dots m\}$
    - Note: here $m \in \mathbb{N}, x^i \in X, \alpha_i \in \mathbb{R}$ are arbitrary,
  - $\mathcal{H}$ is closed under multiplication by a scalar and addition of functions and is then a vector space
  - We define the dot product onto $\mathcal{H}$:
  - Let $f(.) = \sum_{i=1}^{l} \alpha_i K(., x^i) \qquad g(.) = \sum_{j=1}^{m} \beta_j K(., x'^j)$
  - $< f, g > = \sum_{i=1}^{l} \sum_{j=1}^{m} \alpha_i \beta_j K(x^i, x'^j) = \sum_{i=1}^{l} \alpha_i g(x^i) = \sum_{j=1}^{m} \beta_j f(x'^j)$
  - From these equalities, $<.,.>$ is symmetric, bilinear
  - Since K is p.d. for any $f(.) = \sum_{i=1}^{l} \alpha_i K(., x^i)$, one has:

$$< f, f > = \sum_{i,j=1}^{l} \alpha_i \alpha_j K(x^i, x^j) \geq 0$$

    - Note: this means that $<.,.>$ is itself a p.d. kernel on the space of functions

Characterization of kernels

- Reproducing property of the kernel
  - $< f, K(.,x) > = \sum_{i=1}^{l} \alpha_i K(x, x^i) = f(x)$
  - Particular case: $< K(.,x), K(.,x') > = K(x,x')$ or $< \Phi(x), \Phi(x') > = K(x,x')$
  - Using the reproducing property and Cauchy Schwartz:
    - $|f(x)|^2 = |< f, K(.,x) >|^2 \leq K(x,x). < f, f >$
    - Then $< f, f > = 0$ implies $f = 0$
  - This establishes that $<.,.>$ is a dot product
  - It remains to show that space $\mathcal{H}$ is also complete and separable
    - See e.g. (Shawe Taylor et al. 2004)
- Summary
  - Given a p.d. kernel K, one has built $\mathcal{H}$ an associated Hilbert space in which the reproducing property holds, and a mapping Φ
  - $\mathcal{H}$ is called the Reproducing Kernel Hilbert Space (RKHS) of K
    - We give the formal definition of a RKHS later

Characterization of kernels

- Conversely
- Given a mapping Φ from X to a dot product space, we can get a p.d. kernel via $k(x, x') =< \Phi(x), \Phi(x') >$
- Proof
  - $\forall \alpha_i \in \mathbb{R}, x^i \in X, i = 1 \dots m$, we have
  - $\sum_{i,j} \alpha_i \alpha_j K(x^i, x^j) =< \sum_i \ \alpha_i \Phi(x^i), \sum_j \ \alpha_j \Phi(x^j) > = \left\| \sum_i \ \alpha_i \Phi(x^i) \right\|^2 \geq 0$

Characterization of kernels
Summary

- This characterization allows us
  - to give an equivalent definition of p.d. kernels as functions with the property that there exists a map Φ into a dot product space such that $k(x, x') =< \Phi(x), \Phi(x') >$ holds
  - To construct kernels from feature maps
  - $k(x, x') =< \Phi(x), \Phi(x') >$
- is at the base of the kernel trick

Kernel Trick

- Given an algorithm which is formulated in terms of a p.d. kernel, K, one can construct an alternative algorithm by replacing K by another p.d. kernel K'
- Intuition
  - The original algorithm is a dot product based algorithm on vectors $\Phi\left(x^1\right), \dots, \Phi(x^m)$, when $K$is replaced by K', the algorithm is the same but operates on $\Phi'^{\left(x^1\right)}, \dots, \Phi'^{(x^m)}$
  - The best known application of the trick is when $K$is the dot product in the input domain. It can be replaced by another kernel, e.g. non linear. Most of the linear data analysis algorithms (PCA, ridge regression, etc) can then be automatically « kernalized ».
  - Any algorithm that process finite dimensional vectors that is expressed in terms of pairwise inner products, can be applied to infinite-dimensional vectors in the feature space of a p.d. kernel, by replacing each inner product by a kernel evaluation

Reproducing Kernel Hilbert Spaces - RKHS

- Let X be a non empty set and $\mathcal{H}$ a Hilbert space of functions with inner product $<.,.>$. Then $\mathcal{H}$ is called a RKHS if there exists a function $K: XxX \rightarrow \mathbb{R}$ with the following properties:
  - K has the reproducing property
  - $< f, k(x,.) = f(x) \; \forall f \in \mathcal{H}$
    - In particular
  - $< k(x,.), k(x',.) >= k(x,x')$
  - $\forall x \in X, K(x,.) \in \mathcal{H}$

  - K is called a reproducing kernel

- Property
  - The RKHS determines uniquely K and reciproquely
  - A function $K: XxX \rightarrow \mathbb{R}$ is positive definite iff it is a reproducing kernel!

RKHS example – The linear kernel

- Let $X = \mathbb{R}^n$ and consider the linear kernel
  - $K(x, x') = < x, x' >_{\mathbb{R}^n}$
  - The RKHS if the linear kernel is the set of linear functions

$$\mathcal{H} = \{f_w(x) = < w, x >_{\mathbb{R}^n} : \ w \in \mathbb{R}^n\}$$

  - Inner product is defined as

$$\forall v, w \in \mathbb{R}^n, < f_v, f_w >_{\mathcal{H}} = < v, w >_{\mathbb{R}^n}$$

  - The corresponding norm is

$$\forall w \in \mathbb{R}^n, \|f\|_{\mathcal{H}} = \|w\|_{\mathbb{R}^n}$$

Infinite dimensional feature space

- Lemma
  - Let $D = \{x^1, \dots, x^N\}$ distinct points in X, and $\sigma \neq 0$. The matrix $\boldsymbol{K}$ given by $K_{ij} := \exp(-\frac{\|x^i - x^j\|^2}{2\sigma^2})$ has full rank.

  - This means that the points $\Phi(x^1), \dots \Phi(x^N)$ are linearly independent (since $\boldsymbol{K} = \boldsymbol{\Phi}^T \boldsymbol{\Phi}$ with $\boldsymbol{\Phi}$ the matrix with column vectors the $\Phi(x^i)$.
  - Then they span an N dimensional subspace of $\mathcal{H}$.
  - Since this is true for all N, i.e. no restriction on the number of training examples, the feature space is then of infinite dimension

How to build new kernels

- Kernels can be built from combinations of known ones
- Let K_1, K_2 be kernels defined on a metric space X^2, K_3 defined on the Hilbert space $\mathcal{H}$, the following combinations are kernels:
  - $K(x,z) = K_1(x,z) + K_2(x,z)$
  - $K(x,z) = K_1(x,z).K_2(x,z)$
  - $K(x,z) = aK_1(x,z)$
  - $K(x,z) = K_3(\phi(x), \phi(z))$
  - $\ldots..$

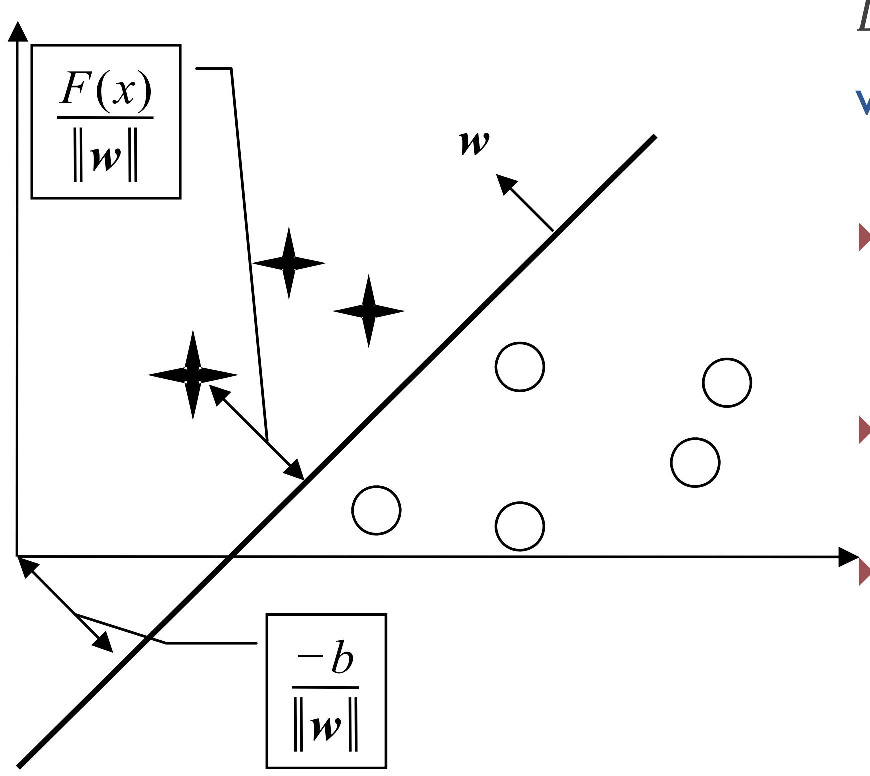
Support vector machines

Support vector machines - SVMs

- SVMs have been developed initially for classification and then extended to regression and density estimation
- The course is a brief introduction to the algorithmic concepts behind SVMs
- We will consider 2 classes classification
  - And develop the concept in the case of linear machines
  - The development of SVMs in the mid 90s mainly concerned non linear embeddings and dealing with structured data (strings, graphs)
  - However the main algorithmic concept could be understood in the simplified linear setting
  - As for any kernel method, non linear SVMs are not well adapted to large dimensional spaces and large datasets
    - They remain an important concept

Margin

- Margin definition


$\frac{F(x)}{\|\boldsymbol{w}\|}$
$\boldsymbol{w}$
$\frac{-b}{\|\boldsymbol{w}\|}$

- Given dataset

$D = \{(\boldsymbol{x}^1, \boldsymbol{y}^1), \dots, (\boldsymbol{x}^N, \boldsymbol{y}^N)\}$

with $y^i \in \{-1,1\}$ and hyperplane H

$F(\boldsymbol{x}) = \boldsymbol{w}.\boldsymbol{x} + b = 0$

- The geometric margin for x^i is:
  - $M(\boldsymbol{x}^i) = y^i\left(\frac{\boldsymbol{w}.\boldsymbol{x}^i}{\|\boldsymbol{w}\|} + \frac{b}{\|\boldsymbol{w}\|}\right) = y^i \frac{F(\boldsymbol{x}^i)}{\|\boldsymbol{w}\|}$
- Margin of $\boldsymbol{w}$ w.r.t. dataset D

$Min_{x^i \in D} M(x^i)$

- Maximal margin hyperplane

$Max_w(Min_{x^i \in D} M(x^i))$

Geometric margin vs functional margin

- Geometric margin
  - $y^i \frac{F(\boldsymbol{x}^i)}{\|\boldsymbol{w}\|}$
- Functional margin
  - $y^i F(x^i)$
- Replacing $\boldsymbol{w}$ with $k\boldsymbol{w}$ $\boldsymbol{k} \in \mathbb{R}$ does not change the decision function or the geometric margin, but changes the functional margin
- For SVMs, one will set the functional margin to 1 (arbitrary value) and one will optimize the geometric margin

Linear separation with optimal hyperplane (1974)

- Hyp : the training set *D* is linearly separable
  - $D = \{(\boldsymbol{x}^1, \boldsymbol{y}^1), \dots, (\boldsymbol{x}^N, \boldsymbol{y}^N)\}$ with $y^i \in \{-1,1\}$
- We consider linear decision functions : $F(x) = w.x + b$
- Learning problem: what is the « optimal hyparplane » H^*:
  - That separates *D*, i.e. $y^i F(x^i) \geq 1, \forall i$
  - With a maximal geometric margin $M = \min_{(x^i, y^i) \in D} \frac{y^i F(x^i)}{\|w\|} = \frac{1}{\|w\|}$
- This is the primal formulation of the linear SVM:
  - Minimize $\|w\|^2$
    - Under the constraint $y^i F(x^i) \geq 1 \;\; \forall i = 1, \dots, N$

- For linear kernels, one usually solves the primal problem
  - There exist several fast gradient algorithms
- For non linear kernels, one usually solves a dual formulation of the problem
  - In the following, one introduces some basic notions on constraint optimization in order to describe this dual formulation

Interlude

Optimisation
under equality and inequality constraints

Optimisation under equality and inequality constraints

- Let
  - $f, g_{i,i=1,\ldots,k}, h_{j,j=1,\ldots,m}$ be functions defined on $\mathbb{R}^n$ taking their value in $\mathbb{R}$
- Let us consider the following optimization problem (pb. (0)) :

$Min\ f(\boldsymbol{w}), \boldsymbol{w} \in \Omega \subset \mathbb{R}^n$

Under constraints

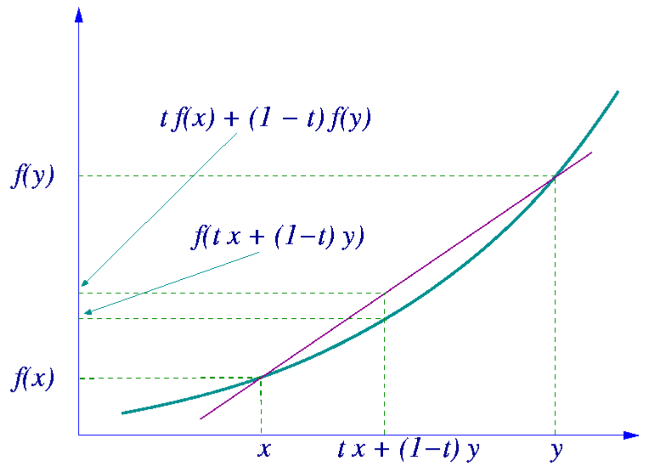
$g_i(\boldsymbol{w}) \leq 0, i = 1, \ldots, k$ denoted $\boldsymbol{g}(\boldsymbol{w}) \leq 0$

$h_j(\boldsymbol{w}) = 0, j = 1, \ldots, m$ denoted $\boldsymbol{h}(\boldsymbol{w}) = 0$

- Definitions
  - Objective function $f(\boldsymbol{w})$
  - **Admissible region** $R = \{\boldsymbol{w} \in \Omega : \boldsymbol{g}(\boldsymbol{w}) \leq 0, \boldsymbol{h}(\boldsymbol{w}) = 0\}$, region of Ω where f is defined and the constraints verified
  - $\boldsymbol{w}^*$ is a **global minimum** if there does not exist another point $\boldsymbol{w}$ such that $f(\boldsymbol{w}) < f(\boldsymbol{w}^*)$, it is a **local optimum** if $\exists \epsilon > 0 : f(\boldsymbol{w}) \geq f(\boldsymbol{w}^*)$, on the subspace $\|\boldsymbol{w} - \boldsymbol{w}^*\| < \epsilon$
  - A constraint $g_i(w) \leq 0$ is said **active** if the solution $\boldsymbol{w}^*$ verifies $\boldsymbol{g}(\boldsymbol{w}^*) = 0$ and inactive otherwise
  - The optimal value of the objective fucntion ($f(\boldsymbol{w})$ solution of pb. (0)) is called the value of the primal optimization problem.

Optimization – convex fonctions

- $f(w)$ is convex for $\boldsymbol{w} \in \mathbb{R}^n$ if
  - $\forall t \in [0,1], \forall \boldsymbol{w}, \boldsymbol{v} \in \mathbb{R}^n, \forall t \in [0,1], \; f(t\boldsymbol{w} + (1-t)\boldsymbol{v}) \leq t f(\boldsymbol{w}) + (1-t) f(\boldsymbol{v})$



- A set $\Omega \subset \mathbb{R}^n$ is convex if $\forall \boldsymbol{w}, \boldsymbol{v} \in \mathbb{R}^n, \forall t \in [0,1], t\boldsymbol{w} + (1-t)\boldsymbol{v} \in \Omega$
- If a function is convex, any local minimum is a global minimum
- An optimization problem where Ω is convex, the objective function is convex and the constraints are convex is said convex

Unconstrained optimisation

- Theorem Fermat
  - A necessary condition w* to be a minimum of $f(\boldsymbol{w})$, $f \in C^1$ is $\frac{\partial f(\boldsymbol{w}^*)}{\partial w} = 0$
  - If f is convex, the condition is sufficient

Optimization with equality constraints
Lagrangian

- Optimization with equality constraints (pb (1)):

  $Min\ f(\boldsymbol{w}), \boldsymbol{w} \in \Omega \subset \mathbb{R}^n$

  Under constraints

  $h_j(\boldsymbol{w}) = 0, \quad j = 1, \dots, m$ denoted $\boldsymbol{h}(\boldsymbol{w}) = 0$

- We define the Lagrangian $L(\boldsymbol{w}, \boldsymbol{\beta})$ associated to this problem as:

$$L(\boldsymbol{w}, \boldsymbol{\beta}) = f(\boldsymbol{w}) + \sum_{j=1}^{m} \beta_j h_j(\boldsymbol{w})$$

  - the β_j are the Lagrange coefficients
- Note
  - If $\boldsymbol{w}^*$ is a solution to pb. (1), it may happen that $\frac{\partial f(\boldsymbol{w}^*)}{\partial w} \neq 0$

Optimization with equality constraints
Th. Lagrange

- Theorem Lagrange
  - A necessary condition for $\boldsymbol{w}^*$, to be a solution of (pb. (1)), with $f, h_i \in C^1$ is
    - $\frac{\partial L(\boldsymbol{w}^*, \boldsymbol{\beta}^*)}{\partial \boldsymbol{w}} = 0$
    - $\frac{\partial L(\boldsymbol{w}^*, \boldsymbol{\beta}^*)}{\partial \boldsymbol{\beta}} = 0$
  - If $L(\boldsymbol{w}, \boldsymbol{\beta}^*)$ is a convex function of $\boldsymbol{w}$, the condition is sufficient

  - Note
    - The first equation gives a new system of equations
    - The second one gives constraints

Optimization under equality and inequality constraints – Generalized Langrangian

- The generalized Lagrangian for pb. (0) is:

$$L(\boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\boldsymbol{w}) + \sum_{i=1}^{k} \alpha_i g_i(\boldsymbol{w}) + \sum_{j=1}^{m} \beta_j h_j(\boldsymbol{w})$$

- Remember the primal formulation of pb. (0):

$Min\ f(\boldsymbol{w}), \boldsymbol{w} \in \Omega \subset \mathbb{R}^n$
under constraints
$g_i(\boldsymbol{w}) \leq 0, i = 1, \dots$,k denoted $\boldsymbol{g}(\boldsymbol{w}) \leq 0$
$h_j(\boldsymbol{w}) = 0, j = 1, \dots$,m denoted $\boldsymbol{h}(\boldsymbol{w}) = 0$

Optimization under equality and inequality constraints – Lagrange Duality

- Reformulation of the primal problem
  - Let $\theta_P(\boldsymbol{w}) = \max_{\boldsymbol{\alpha},\boldsymbol{\beta};\boldsymbol{\alpha}\geq 0} L(\boldsymbol{w},\boldsymbol{\alpha},\boldsymbol{\beta}) = f(\boldsymbol{w}) + \max_{\boldsymbol{\alpha},\boldsymbol{\beta};\boldsymbol{\alpha}\geq 0} (\alpha.g(\boldsymbol{w}) + \beta.h(\boldsymbol{w}))$,
    - P in θ_P holds for Primal
    - Note: $\theta_P(\boldsymbol{w})$ is a function of $\boldsymbol{w}$, not of $\boldsymbol{\alpha},\boldsymbol{\beta}$
  - $\theta_P(\boldsymbol{w}) = \begin{cases} f(\boldsymbol{w}) \text{ if } \boldsymbol{w} \text{ is admissible} \\ +\infty \text{ otherwise, i.e. } \boldsymbol{w} \text{ violates any of the constraints} \end{cases}$
    - This is because $\max_{\boldsymbol{\alpha},\boldsymbol{\beta};\boldsymbol{\alpha}\geq 0}\big(\alpha g(\mathbf{w}) + \beta h(\mathbf{w})\big) = 0$ for a point $\boldsymbol{w}$ admissible i.e. that satisfies the primal constraints
    - $\theta_P(\boldsymbol{w})$ takes the same value as the objective function $f(\boldsymbol{w})$ for any $\boldsymbol{w}$ admissible
  - The original primal problem (pb 0) can be reformulated as:
    - $\min_{\mathbf{w}\in\Omega} \theta_P(\boldsymbol{w})$ or $\min_{\mathbf{w}\in\Omega} \max_{\boldsymbol{\alpha},\boldsymbol{\beta};\boldsymbol{\alpha}\geq 0} L(\boldsymbol{w},\boldsymbol{\alpha},\boldsymbol{\beta})$

      □ $\min_{\mathbf{w}\in\Omega} \theta_P(\boldsymbol{w})$ is called the value of the primal

Optimization – Lagrange duality

- Dual formulation of the optimization problem
  - Let $\theta_{\mathrm{D}}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \min_{w \in \Omega} L(\boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\beta})$
    - D in θ_{D} holds for dual
  - The dual optimization problem for (pb 0) is:
    - $max_{\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\alpha}_i \geq 0}\ \theta_{\mathrm{D}}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \min_{w \in \Omega} L(\boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\beta})$

      under constraint $\boldsymbol{\alpha} \geq 0$

    - $max\ \theta_{\mathrm{D}}(\boldsymbol{\alpha}, \boldsymbol{\beta})$is called the value of the dual
    - Note : $\min_{w \in \Omega} L(\boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ is a function of $\boldsymbol{\alpha}, \boldsymbol{\beta}$ only

- Property (weak duality)
  - the value of the dual is bounded above by the value of the primal
  - $\max_{\boldsymbol{\alpha},\boldsymbol{\beta};\boldsymbol{\alpha}\geq 0} \min_{w \in \Omega} L(\boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \leq \min_{w \in \Omega} \max_{\boldsymbol{\alpha},\boldsymbol{\beta};\boldsymbol{\alpha}\geq 0} L(\boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\beta})$

  For some cases there is equality, this is called strong duality

Optimisation : weak duality

- $\max_{\boldsymbol{\alpha},\boldsymbol{\beta};\boldsymbol{\alpha}\geq 0} \min_{\boldsymbol{v}\in\Omega} L(\boldsymbol{v},\boldsymbol{\alpha},\boldsymbol{\beta}) \leq \min_{\mathbf{w}\in\Omega} \max_{\boldsymbol{\alpha},\boldsymbol{\beta};\boldsymbol{\alpha}\geq 0} L(\boldsymbol{w},\boldsymbol{\alpha},\boldsymbol{\beta})$:
  - $\theta_D(\alpha,\beta) = \min_{\boldsymbol{v}\ \in\Omega} L(\boldsymbol{v},\boldsymbol{\alpha},\boldsymbol{\beta})$:
    - $\theta_D(\alpha,\beta) \leq L(\boldsymbol{w},\boldsymbol{\alpha},\boldsymbol{\beta})$ for any $\boldsymbol{w}$ admissible
    - $L(\boldsymbol{w},\boldsymbol{\alpha},\boldsymbol{\beta}) = f(\boldsymbol{w}) + \boldsymbol{\alpha}.g(\boldsymbol{w}) + \boldsymbol{\beta}.h(\boldsymbol{w})$ with $\boldsymbol{\alpha} \geq 0, g(\boldsymbol{w}) \leq 0, h(\boldsymbol{w}) = 0$
    - $L(\boldsymbol{w},\boldsymbol{\alpha},\boldsymbol{\beta}) \leq f(\boldsymbol{w})$
    - $\theta_D(\boldsymbol{\alpha},\boldsymbol{\beta}) \leq f(\boldsymbol{w})$
    - $\max_{\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\alpha}\geq 0} \min_{\boldsymbol{v}\in\Omega} L(\boldsymbol{v},\boldsymbol{\alpha},\boldsymbol{\beta}) \leq f(\boldsymbol{w})$ for any w admissible

  - $\theta_P(\boldsymbol{w}) = \max_{\boldsymbol{\alpha},\boldsymbol{\beta};\boldsymbol{\alpha}\geq 0} L(\boldsymbol{w},\boldsymbol{\alpha},\boldsymbol{\beta})$
    - $\theta_P(\boldsymbol{w}) = f(\boldsymbol{w}) + \max_{\boldsymbol{\alpha},\boldsymbol{\beta};\boldsymbol{\alpha}\geq 0} (\alpha.g(\boldsymbol{w}) + \beta.h(\boldsymbol{w}))$
    - $\theta_P(\boldsymbol{w}) = \begin{cases} f(\boldsymbol{w}) \text{ if } \boldsymbol{w} \text{ is admissible} \\ +\infty \text{ otherwise} \end{cases}$ since $\max_{\boldsymbol{\alpha},\boldsymbol{\beta};\boldsymbol{\alpha}\geq 0} \big(\alpha g(w) + \beta h(w)\big) = 0$ for an admissible point
    - $\theta_P(\boldsymbol{w}) = f(\boldsymbol{w})$ for $\boldsymbol{w}$ admissible
- Hence the inequality of weak duality

- Theorem: strong duality
  - For an optimization problem

    $Min\ f(\boldsymbol{w}), \boldsymbol{w} \in \Omega \subset \mathbb{R}^n$ convex and $f \in C^1$ convex

    under constraints

    $g_i(\boldsymbol{w}) \leq 0, i = 1, \dots, k$ denoted $\boldsymbol{g}(\boldsymbol{w}) \leq 0$

    $h_j(\boldsymbol{w}) = 0, j = 1, \dots, m$ denoted $\boldsymbol{h}(\boldsymbol{w}) = 0$

    where the g_i and the h_j are affines $(h_j(\boldsymbol{w}) = A_{\boldsymbol{j}}\boldsymbol{w} + b_j)$ (hence convex)

    - The values of the primal and of the dual are equal
    - The conditions for the existence of an optimum are given by the theorem of Kuhn and Tucker

Optimization
Theorem Kuhn and Tucker

- Let us consider (pb. (0)) with Ω convex and $f \in C_1$ convex, g_i, h_j affines ($h = A.\boldsymbol{w} + b$)
- A necessary and sufficient condition for $\boldsymbol{w}^*$ to be an optimum is that there exist $\boldsymbol{\alpha}^*$ and $\boldsymbol{\beta}^*$:
  - $$\begin{cases} \frac{\partial L(\mathbf{w}^*,\boldsymbol{\alpha}^*,\boldsymbol{\beta}^*)}{\partial \mathbf{w}} = 0 \\ \frac{\partial L(\mathbf{w}^*,\boldsymbol{\alpha}^*,\boldsymbol{\beta}^*)}{\partial \boldsymbol{\beta}} = 0 \\ \alpha_i^* g_i(\mathbf{w}^*) = 0, i = 1..k \\ g_i(\mathbf{w}^*) \leq 0, i = 1..k \\ \alpha_i^* \geq 0, i = 1..k \end{cases}$$
- Under the assumptions of **convexity**, the dual formulation is an alternative to the primal formulation which may be simpler to handle (e.g. non linear SVMs)

Optimization

- Note
  - The third condition ($\alpha_i^* g_i(\mathbf{w}^*) = 0, i = 1..k$), called complementary condition of Karush-Kuhn-Tucker implies that for an active constraint $\alpha_i^* \geq 0$ when for an inactive constraint $\alpha_i^* = 0$
    - Either a constraint is active ($\alpha_i^* \geq 0$ and $\mathrm{g_i}(\mathrm{w}^*) = 0$), w^* is a frontier point of the admissible region
    - Or it is inactive ($\alpha_i^* = 0$) and w^* is in the admissible region
  - If the solution point w^* is in the admissible region (inactive constraint) then the conditions of optimality are given by the Fermat theorem and $\alpha_i^* = 0$. If it is on the frontier (active constraint), the conditions of optimality are given by the theorem of Lagrange with $\alpha_i^* > 0$.

- End of the interlude

SVM – primal and dual formulations For a linear kernel

- SVM
  - Ω, f and the constraints are convex, L is quadratic
  - One considers the case, $D = \{(x^i, y^i)\}_{i=1\ldots N}$ linearly separable, $y^i \in \{-1,1\}$
- Primal pb $\quad Min(w.w) \quad$ (i.e. max the margin)
  under constraints
  $y^i(w.x^i + b) \geq 1, i = 1..N$
- Lagrangian primal $\quad L(w,b,\alpha) = \frac{1}{2}w.w - \sum_{i=1}^{N} \alpha_i(y^i(w.x^i + b) - 1)$
- Lagrangian dual $\quad L(w,b,\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N} y^i y^j \alpha_i \alpha_j (x^i.x^j)$
- With $\alpha_i \geq 0$ in both cases

SVM – primal and dual formulations
For a linear kernel

- Derivation of the dual formulation
  - Let us start from the primal formulation of the Lagrangian
  - $L(w,b,\alpha) = \frac{1}{2}w.w - \sum_{i=1}^{N} \alpha_i(y^i(w.x^i + b) - 1)$
  - Let us first minimize $L(w,b,\alpha)$ w.r.t. w,b in order to get θ_D
  - $\frac{\partial L(w,b,\alpha)}{\partial \mathrm{w}} = w - \sum_{i=1}^{N} \alpha_i y^i x^i = 0$
    - $w = \sum_{i=1}^{N} \alpha_i d^i x^i$
  - $\frac{\partial L(w,b,\alpha)}{\partial b} = \sum_{i=1}^{N} \alpha_i y^i = 0$
  - Let us plug back $w = \sum_{i=1}^{N} \alpha_i d^i x^i$ in the primal form of $L(w,b,\alpha)$
  - We get
  - $L(w,b,\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N} y^i y^j \alpha_i \alpha_j \left(x^i.x^j\right) - b\sum_{i=1}^{N} \alpha_i y^i$
  - $L(w,b,\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N} y^i y^j \alpha_i \alpha_j \left(x^i.x^j\right)$
    - This expression of the dual form only depends on the variables α, β

SVM – primal and dual formulations
For a linear kernel

- Dual problem
- $\text{Max}_\alpha L(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} y^i y^j \alpha_i \alpha_j (x^i . x^j)$

  under constraints:

$$\begin{cases} \sum_{i=1}^{N} y^i \alpha_i = 0 \\ \alpha_i \geq 0, i = 1..N \end{cases}$$

  - Constraint $\alpha_i \geq 0, i = 1..N$ has always been present
  - $\sum_{i=1}^{N} d^i \alpha_i = 0$ comes from the derivation of the dual form

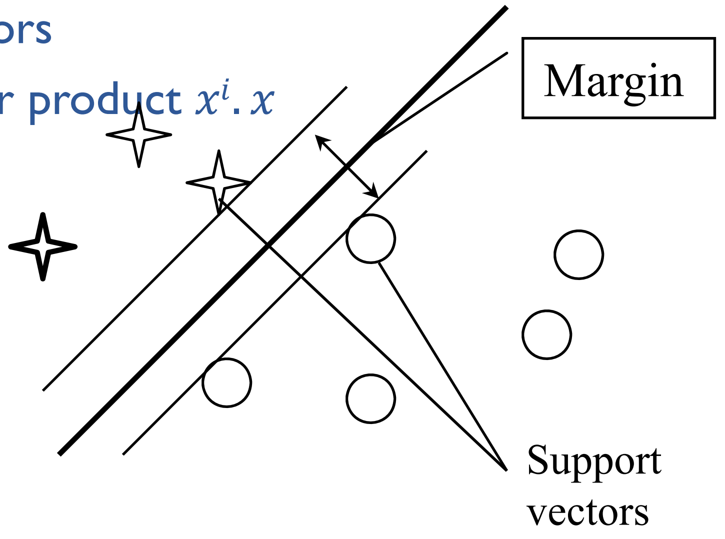
- This is a quadratic optimization problem under constraints

SVM – solution
For a linear kernel

- Solution : $\boldsymbol{w}^*$ only depends on the support vectors, i.e. the vectors on the margin that verify: $y^i F^*(x^i) = 1$
- The decision function takes the form:

$$F(x, \alpha^*, \beta^*) = \sum_{i\ support\ vector} y^i \alpha_i^* \, (x^i . x) + b^*$$

  - Note: Whatever the dimension of the space is, the degree of freedom is equal to the number of support vectors
  - F^* only depends on the inner product $x^i . x$



SVM – primal and dual formulations
For a linear kernel

- Let α, b be solution of the dual problem
- Then the decision function writes
  - $F(x, \alpha^*, b^*) = \sum_{i=1}^{N} \alpha_i^* y^i x^i + b^*$
  - Remember the KKT condition $\alpha_i^* g_i(\mathbf{w}^*) = 0, i = 1..k$:
    - $\alpha_i^*\left(\mathrm{y}^\mathrm{i}\left(\mathrm{w}^*.\mathrm{x}^\mathrm{i} + \mathrm{b}^*\right) - 1\right) = 0, i = 1..k$ (in the primal problem definition)
    - This means that only the points x^i for which $\mathrm{y}^\mathrm{i}\left(\mathrm{w}^*.\mathrm{x}^\mathrm{i} + \mathrm{b}^*\right) = 1$ have a coefficient $\alpha_i^* \neq 0$
    - i.e. $F(x, \alpha^*, b^*) = \sum_{i\ support\ vector} \alpha_i^* y^i x^i + b^*$

Support vector machine
Non linear kernels

- The dot product in the input space X is replaced by a dot product in the feature space $\mathcal{H}$
- This dot product can be computed by a kernel function
- X can be any set – not only a inner product space
- The above formulation can be transposed by replacing $< x^i, x >$ with $K(x^i, x)$

$$\Phi: R^n \to \mathcal{H}$$

$$w = \sum_{x^i \; support \; vector} y^i \alpha_i \Phi(x^i) \qquad F(x) = \sum_{x^i \; support \; vector} y^i \alpha_i \Phi(x^i).\Phi(x) + b$$

$$\Phi(x).\Phi(x') = K(x, x')$$

$$F(x) = \sum_{x^i \; support \; vector} y^i \alpha_i K(x, x^i) + b$$

Support vector machine
Non linear kernels

- In practice, this formulation does not lead to stable solutions, and other formulations that weaken the constraints (soft margins) are used.

Support vector machine
Optimization algorithms – rebirth of stochastic algorithms

- Standard algorithm for SVMs
  - e.g. Sequential Minimal Optimization (SMO)
- Versus stochastic algorithms – (Bottou 2007)
  - Task : Document classification - RCV1 documents belonging to the class CCAT (2 classes classification task)
    - Programs SVMLight and SVMPerf are well known SVM solvers written by Thorsten Joachims. SVMLight is suitable for SVMs with arbitrary kernels. Similar results could be achieved using Chih-Jen Lin's LibSVM software. SVMPerf is a specialized solver for linear SVMs. It is considered to be one of the most efficient optimizer for this particular problem.

Algorithm (hinge loss)	**Training Time**	**Primal cost**	**Test Error**
SVMLight	23642 secs	0.2275	6.02%
SVMPerf	66 secs	0.2278	6.03%
Stochastic Gradient (svmsgd)	**1.4 secs**	0.2275	6.02%
Stochastic Gradient (svmsgd2	**1.4 secs**	0.2275	6.01%

Gaussian process regression

Motivations

- Most ML algorithm for regression predict a mean value
- Gaussian processes are Bayesian methods that allow us to predict, not only a mean value, but a distribution over the output values
  - In regression, for each input value x, the predicted distribution is Gaussian and is then fully characterized by its mean and variance

Gaussian distributions refresher

- Multivariate Gaussian distribution $x \sim \mathcal{N}(\mu, \Sigma), x \in R^n$
  - $p(x) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{1/2}} \exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))$
- Summation (a)
  - Let x and y two random variables with the same dimensionality, $p(x) = \mathcal{N}(\mu_x, \Sigma_x)$ and $p(y) = \mathcal{N}(\mu_y, \Sigma_y)$
  - Then their sum is also Gaussian: $\mathrm{p(x+y)} = \mathcal{N}(\mu_x + \mu_y, \Sigma_x + \Sigma_y)$
- Marginalization (b)
  - Let $x, p(x) = \mathcal{N}(\mu, \Sigma)$, consider a partition of x into two sets of variables $x = \begin{pmatrix} x_a \\ x_b \end{pmatrix}$.
  - Let us denote $\mu = \begin{pmatrix} \mu_a \\ \mu_b \end{pmatrix}, \Sigma = \begin{pmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{pmatrix}$
  - Then the marginals are also Gaussians, e.g.: $p(x_a) = \int_{x_b} p(x_a, x_b; \mu, \Sigma) dx_b = \mathcal{N}(\mu_a, \Sigma_{aa})$,
  - Σ being symmetric, $\Sigma_{ab} = \Sigma_{ba}$
- Conditioning (c)
  - The conditionals are also Gaussians
  - $p(x_a|x_b) = \mathcal{N}(\mu_{a|b}, \Sigma_{a|b})$ with $\mu_{a|b} = \mu_a + \Sigma_{ab}\Sigma_{bb}^{-1}(x_b - \mu_b)$ and $\Sigma_{a|b} = \Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}$
- Marginalization bis (d)
  - Let x and y two random vectors such that $p(x) = \mathcal{N}(\mu, \Sigma_x)$ and $p(y|x) = \mathcal{N}(Ax + b, \Sigma_y)$
  - The marginal of y is $p(y) = \int p(y|x)p(x)dx = \mathcal{N}(A\mu + b, \Sigma_y + A\Sigma_x A^T)$

Introducing the Gaussian processes
From Bayesian linear regression to Gaussian processes

- Consider the linear parameter model:
  - $y(\boldsymbol{x}) = \boldsymbol{w}^T \phi(\boldsymbol{x})$
  - where $\boldsymbol{w} \in R^M, \phi(\boldsymbol{x}) \in R^M$ are M fixed basis functions
    - For example, ϕ could be a linear function $\phi(\boldsymbol{x}) = (\boldsymbol{x}, 1)$ or ϕ could be a vector of gaussian kernels $\phi_i(x) = \exp(-\frac{(x-\mu_i)^2}{2s^2})$
- We consider a Bayesian setting
  - With $\boldsymbol{w}$ following a prior distribution given by an isotropic Gaussian
    - $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{0}, \alpha^{-1} I)$
      - □ α^{-1} is the precision parameter = the inverse variance
  - For any value of $\boldsymbol{w}$, $y(\boldsymbol{x}) = \boldsymbol{w}^T \phi(\boldsymbol{x})$ defines a specific function of $\boldsymbol{x}$
  - $p(\boldsymbol{w})$ thus defines a distribution over functions $y(\boldsymbol{x})$

Introducing the Gaussian processes
From Bayesian linear regression to Gaussian processes

- How to characterize the distribution over functions $y(\boldsymbol{x})$?
- In practice, we will want to evaluate $y(\boldsymbol{x})$ at specific values x
  - e.g. at the training points or for a test point, lets do that:
- Let us consider a finite data sample $x^1, \dots, x^N$
- Let us denote $\boldsymbol{y} = \left(y^1, \dots, y^N\right)^T$, with $y^i = y(x^i)$
- We want to characterize the distribution of $\boldsymbol{y}$
  - $\boldsymbol{y} = \boldsymbol{\Phi}\mathbf{w}$, with $\boldsymbol{\Phi} = \left[\boldsymbol{\phi}(\boldsymbol{x}^1), \dots, \boldsymbol{\phi}(\boldsymbol{x}^N)\right]^T$ called the design matrix $\Phi_{ij} = \phi_j(x^i)$
  - $\mathbf{w}$ is Mx1, $\boldsymbol{\Phi}$ is NxM, $\boldsymbol{y}$ is Nx1
  - $\boldsymbol{y}$ being a linear combination of Gaussian variables (the elements of $\mathbf{w}$) is itself Gaussian and fully characterized by its mean and variance
    - $E[y] = \boldsymbol{\Phi}\mathrm{E}[\mathbf{w}] = 0$
    - $Cov[\mathbf{y}] = E\left[\boldsymbol{y}\boldsymbol{y}^T\right] = \boldsymbol{\Phi}\mathrm{E}\left[\boldsymbol{w}\boldsymbol{w}^{\mathrm{T}}\right]\boldsymbol{\Phi}^{\mathrm{T}} = \frac{1}{\alpha}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} = \mathbf{K}$
    - $\mathbf{K}$ is a Gram matrix with elements $K_{nm} = k(x^n, x^m) = \frac{1}{\alpha}\phi(x^n)^T\phi(x^m)$
      - $k(x, x')$ is the kernel function

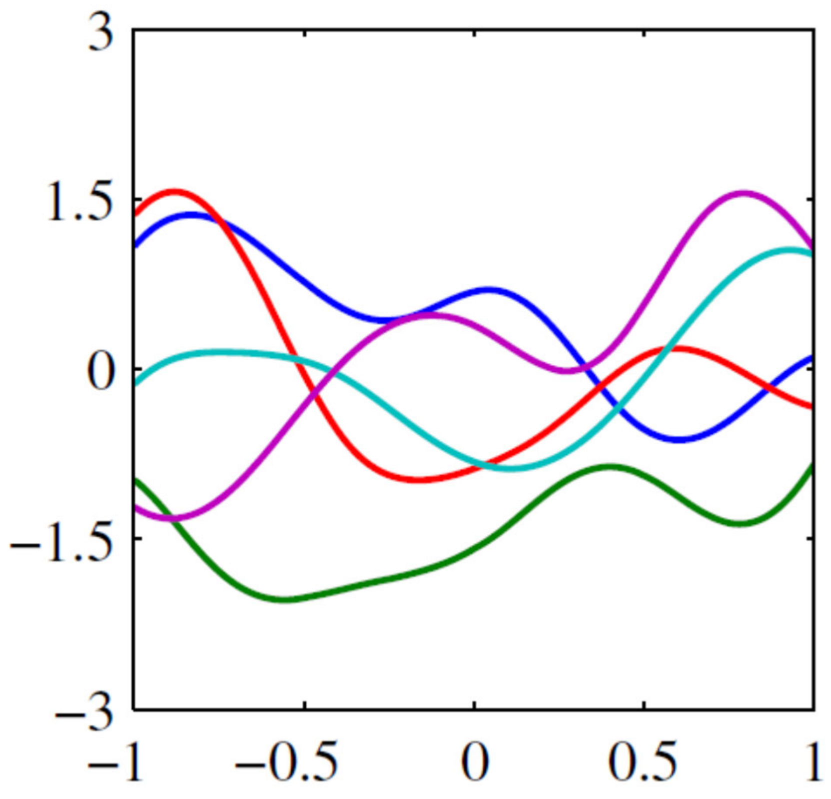
$$\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{K}),\ \boldsymbol{y} \text{ is } N\text{x}1, \boldsymbol{K} \text{is } N\text{x}N$$

- This is a first example of Gaussian process, defined by a linear model
- Usually, the kernel function is not defined through basis functions, but directly by specifying a Kernel function, e.g. a Gaussian kernel

Introducing the Gaussian processes
From Bayesian linear regression to Gaussian processes

- Samples of functions drawn from Gaussian processes for a « Gaussian Kernel »
- $k(x, x') = \exp(-\frac{\|x-x'\|^2}{2\sigma^2})$
- We specify a set of input points $x = (x^1, \dots, x^N)$ in $[-1,1]$ and an NxN covariance matrix K.
- We draw a vector $(y^1, \dots, y^N)$ from the Gaussian defined by $\boldsymbol{y} = \mathcal{N}(\mathbf{0}, \boldsymbol{K})$
- The figure shows samples drawn from gaussian processes
  - Each curve is a sample

- Bishop C. PRML



Introducing the Gaussian processes
From Bayesian linear regression to Gaussian processes

- A stochastic process $y(\boldsymbol{x})$ is specified by the joint probability distribution for **any** finite set of values $\{y(\boldsymbol{x}^1), \dots, y(\boldsymbol{x}^N)\}$
- The joint distribution over N variables $y^1, \dots, y^N$ is specified completely by their mean and covariance

Gaussian processes

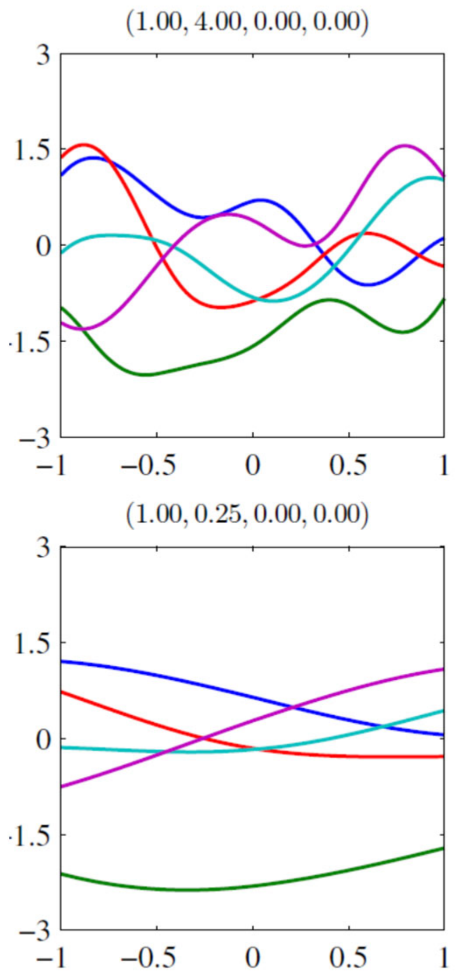
- Definition
  - A stochastic process is a collection of random variables $\{f(x); x \in \mathcal{X}\}$ indexed by elements of set $\mathcal{X}$ (in the following one will consider $\mathcal{X} = R$).
    - This is a probability distribution over the functions $f(x)$
  - A Gaussian process is a stochastic process such that the set of values of $f(x)$ evaluated at any number of points $x^1, \dots \mathrm{x}^{\mathrm{N}}$ is jointly Gaussian, i.e.:
    - $\begin{bmatrix} f(x^1) \\ \vdots \\ f(x^N) \end{bmatrix} \sim N\left(\begin{bmatrix} m(x^1) \\ \vdots \\ m(x^N) \end{bmatrix}, \begin{bmatrix} k(x^1, x^1) \dots k(x^1, x^m) \\ \vdots \quad \ddots \quad \vdots \\ k(x^m, x^1) \dots k(x^m, x^m) \end{bmatrix} \right)$
- Properties
  - A Gaussian process is entirely specified by its
    - Mean function $\mathrm{m(x)} = E[f(x)]$
    - Covariance function $k(x, x') = E[(f(x) - m(x))(f(x') - m(x')]$
  - One denotes $\mathrm{f} \sim GP(m, k)$ meaning that f is distributed as a GP with mean m and covariance k functions

Gaussian processes

- Intuition
  - Gaussian distributions model finite collections of real valued variables
  - Gaussian processes extend multivariate gaussians to infinite collections of real-valued vairables
    - GP are distributions over random functions
    - Let H be a class of functions $f: X \to Y$. A random function $f(.)$ from H is a function which is randomly drawn from H
    - Intuitively, one can think of $f(.)$ as an infinite vector drawn from an infinite multivariate Gaussian. Each dimension of the Gaussian corresponds to an element x from the index and the corresponding component of the random vector is the value $f(x)$
- What could be the functions $m(.)$ and $k(.,.)$?
  - Any real valued function $m(.)$ is acceptable
  - K should be a valid covariance matrix corresponding to a Gaussian distribution
    - This is the case if K is positive semi-definite (remember conditions for valid kernels)
      - Any valid kernel can be used as a covariance function

Gaussian processes

- Example
  - Zero mean Gaussian process $GP(0, k(.,.))$ defined for functions $h: X \rightarrow R$
  - $k(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\frac{\theta_1}{2}||\boldsymbol{x} - \boldsymbol{x}'||^2)$
  - The function values are distributed around 0
  - $f(x)$ and $f(x')$ will have a high covariance $k(x, x')$ if x and $x'$are nearby and a low covariance otherwise
    - i.e. they are locally smooth



Bishop PRML, Top $\theta_1 = 4$, bottom $\theta_1 = 0.25$

Gaussian processes for regression

- We consider a Gaussian process regression model (1 dimensional for simplification)
  - $y = f(x) + \epsilon$, with $x \in R^n$ and $y \in R$
    - $\epsilon \sim \mathcal{N}(0, \sigma^2)$ independently chosen for each observation accounts for the noise at each observation
  - Let us consider a set of training examples $S = \{(x^1, y^1), \dots, (x^N, y^N)\}$ from an unknown distribution
  - Let us denote $Y = \left(y^1, \dots, y^N\right)^T$ and $F = \left(f^1, \dots, f^N\right)^T$ with $f^i = f(x^i)$
  - From the definition of a Gaussian process, one assume a prior distribution over functions $f(.)$. We assume a zero mean Gaussian process prior:
    - $p(F) = \mathcal{N}(0, K)$ with K a Gram matrix defined by a kernel function $K_{ij} = k(x_i, x_j)$
- We will
  - Characterize the joint distribution of $Y = \left(y^1, \dots, y^N\right)^T$
  - In order to define the predictive distribution for test points $p(y_{N+1}|Y)$

Gaussian processes for regression
Characterizing the joint distribution of $Y = \left(y^1, \ldots, y^N\right)^T$

- The joint distribution of $Y = \left(y^1, \ldots, y^N\right)^T$ is
  - $p(Y) = \int p(Y|F)p(F)df = \mathcal{N}(O, C)$
  - With the covariance matrix C defined as $C\left(x^i, x^j\right) = k\left(x^i, x^j\right) + \frac{1}{\sigma^2}\delta_{ij}$
    - δ_{ij} is the Kronecker symbol
- Demonstration
  - We will first show $p(Y|F) = \mathcal{N}(F, \frac{1}{\sigma^2}\ I_N)$
    - $p(Y|F) = p\left(y^1, \ldots, y^N | F\right)$
    - $p(Y|F) = \prod_{i=1}^N p\left(y^i\ | f^i\right)$
    - $p(Y|F) = \prod_{i=1}^N \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{1}{2\sigma^2}\ \left(y^i - f^i\right)^2)$
    - $p(Y|F) = \left(\frac{1}{2\pi\sigma^2}\right)^{N/2} \exp(-\frac{1}{2\sigma^2}\ \|Y - F\|^2)$
    - $p(Y|F) = \mathcal{N}(F, \sigma^2 I_N)$

Gaussian processes for regression

Characterizing the joint distribution of $Y = \left(y^1, \dots, y^N\right)^T$

- Demonstration of $p(Y) = \int p(Y|F)p(F)df = \mathcal{N}(O, C)$

  - $p(F) = \mathcal{N}(0, K)$
  - $p(Y|F) = \mathcal{N}(F, \frac{1}{\sigma^2} I_N)$
  - $p(Y) = \int p(Y|F)p(F)df$
  - By property (d) in Gaussian refresher we get:
  - $p(Y) = \mathcal{N}\left(O, \frac{1}{\sigma^2} I_N + K\right) = \mathcal{N}(O, C)$
    - With $C_{ij} = k\left(x^i, x^j\right) + \frac{1}{\sigma^2}\delta_{ij}$

Gaussian processes for regression
Predictive distribution

- For the regression, our goal is to predict the value y for a new observation x
  - Let us consider a training set $\mathrm{D} = \{(\mathrm{x}^{\mathrm{i}}, \mathrm{y}^{\mathrm{i}}); \mathrm{i} = 1 \ldots \mathrm{N}\}$, and denote $Y^N = (y^1, \ldots, y^N)^T$, let y^{N+1} the value one wants to predict for observation x^{N+1}, $Y^{N+1} = (Y^N, y^{N+1})^T$
- Let us first explicit the joint distribution over Y^{N+1}
  - $p(Y^{N+1}) = \mathcal{N}(0, C_{N+1})$ with $C_{N+1} = \begin{pmatrix} C_N & k \\ k^T & c \end{pmatrix}$
    - C_N the covariance matrix of Y_N
    - $k \in R^N \; k_i = k(x^i, x^{N+1}); i = 1 \ldots N$
    - $c = \mathrm{k}(\mathrm{x}^{\mathrm{N+1}}, \mathrm{x}^{\mathrm{N+1}}) + \sigma^2 \; \in R$
  - Proof
    - This is a direct application of the result shown before $p(Y) = \mathcal{N}(O, C)$

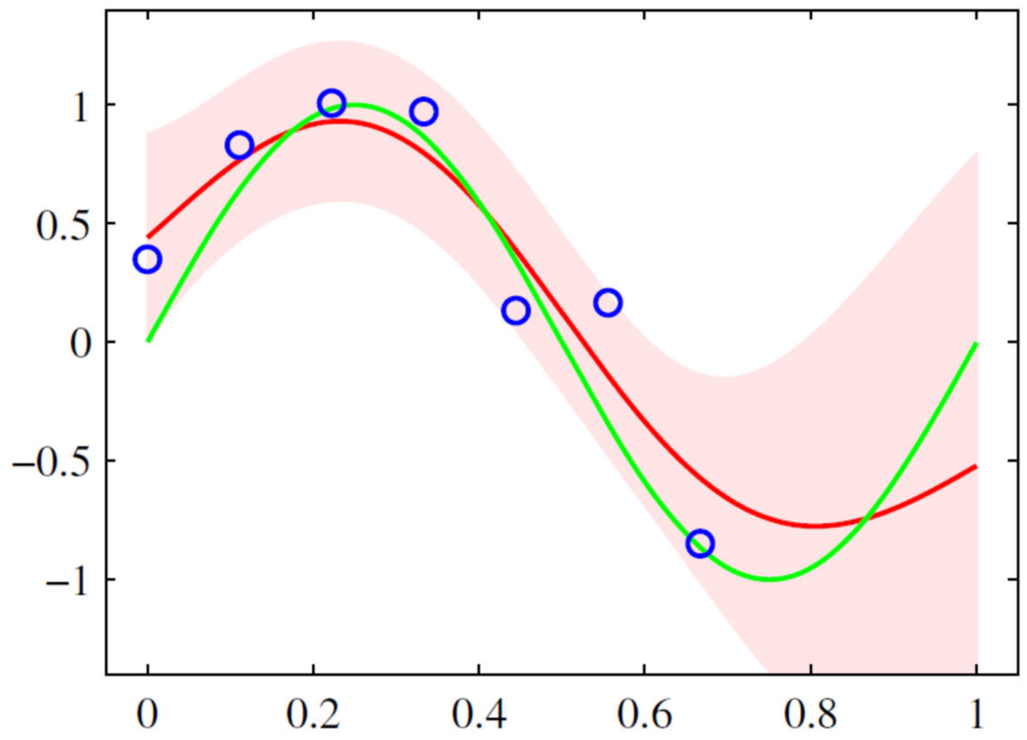
Gaussian processes for regression
Predictive distribution

- Prediction is achieved via the conditional distribution $p\left(y^{N+1}\middle|Y\right)$
  - By definition of a Gaussian process, $p\left(y^{N+1}\middle|Y,X\right)$ is a Gaussian.
  - Its mean and covariance are given by:
    - $m\left(x^{N+1}\right) = k^T C_N^{-1} Y$
    - $\sigma^2\left(x^{N+1}\right) = c - k^T C_N^{-1} k$
  - Proof
    - This is a direct application of property (c) (conditioning)
- Property
  - $m\left(x^{N+1}\right)$ writes as $m\left(x^{N+1}\right) = \sum_{i=1}^{N} a_i k(x^i, x^{N+1})$
    - With a_i the i^{th} component of $C_N^{-1}Y$
- Prediction in practice
  - Given a training set of N points $S = \{(x^1, y^1), \dots, (x^N, y^N)\}$, and the specification of a ketrnel function $k(.,.)$, it is then possible to infer the posterior distribution distribution for any new input point x^{N+1}

Gaussian processes for regression
Predictive distribution

- This means that for any new datum x^{N+1}, one can compute
  - A mean prediction $m\left(x^{N+1}\right)$
  - An uncertainty associated to this prediction $\sigma^2\left(x^{N+1}\right)$

Illustration of Gaussian process regression applied to the sinusoidal data set in Figure A.6 in which the three right-most data points have been omitted. The green curve shows the sinusoidal function from which the data points, shown in blue, are obtained by sampling and addition of Gaussian noise. The red line shows the mean of the Gaussian process predictive distribution, and the shaded region corresponds to plus and minus two standard deviations. Notice how the uncertainty increases in the region to the right of the data points.



Bishop C. PRML

Gaussian processes for regression

- Scaling
  - The central computation in using Gaussian processes involes the inversion of an $N \times N$ matrix
  - This is $O(N^3)$ with standard methods
  - For each new test point, this requires a vector matrix multiply which is $O(N^2)$
  - Fo large datasets, this is unfeasible
    - Several approximations have been proposed but this remains ill adapted to large datasets and high dimensions.

Learning hyperparameters

- The kernel functions can be chosen a priori
- Alternatively, they may be defined as parametric functions (e.g. squared exponential kernel as in the example and the parameters may be learned e.e. by maximum likelihood
  - Log likelihood for a Gaussian process regression model
  - $\log p(Y|\theta) = -\frac{1}{2}\log|C_N| - \frac{1}{2}Y^T C_N^{-1} Y - \frac{N}{2}\log(2\pi)$
  - Training can be performed using gradient descent on the parameters θ

Gaussian processes - references

- Bishop, Pattern Recognition and Machine Learning, 2007
- Rasmussen, C. E., & Williams, C. K. I. (2006). Gaussian Processes for Machine Learning. In *MIT Press*.
- Mackay, D. J. C. *Introduction to Gaussian Processes.*

Neural Processes

Neural Process

- Objective
  - Learn a stochastic process, i.e. a distribution P_f over functions $f: X \to Y$, like for GPs
  - Rely on NNs for the modeling in order to get better scaling than GPs
  - Neural process corresponds to a family of methods sharing the same general idea
- Neural processes rely on meta-learning algorithms
  - We very briefly introduce the concept of meta-learning
  - We introduce a simple instance of neural processes: the conditional neural process

Informal introduction to meta-learning

- Task
  - Tasks are basic concepts in meta-learning
    - It corresponds to a problem to solve like:
      - Regression given a set of points generated by an unknown function f
        - The task could be represented by a sample of points to regress (this is the example used in the presentation)
      - Classification, Game playing, etc
      - Note: the name task is used because in the general meta-learning framework, they could indeed correspond to different objectives with different loss functions. But in general, as it is considered here, they correspond to different instances of a same problem
- Meta-learning
  - Assumption: availability of multiple related tasks
  - Objective: learn from the set of tasks to generalize to a new related tasks

Informal introduction to meta-learning - example

- Let us consider regression tasks
  - Each regression consists in learning an estimator for an unknown function g: $X -> Y$, from a sample $\{(x^i, y^i)\ , i = 1..N\}$ of g
  - Let P be a distribution on functions g: $X -> Y$
    - For simplicity we consider $X = Y = \mathbb{R}$
- Learning problem
  - Given a sample of functions g^k, with $g^k \sim P$, each represented by a set of points D_k, $\mathrm{k} = 1 \dots \mathrm{K}$
  - Learn from the $D_k s$ a regression function $f(x; D)$ that will approximate the unknown function g from which D has been sampled, for each D_k
  - The objective is to generalize to unknown functions g not seen during training, given a sample D of the new function g

Informal introduction to meta-learning - example

- Meta learning offers several families of methods for solving this type of problem
- Example: Model Agnostic Meta Learning (MAML) (Finn et al. 2017)
  - Learn a model paramaters θ (parameters of the function $f(x, D)$) from a set of tasks (e.g. multiple regressions)
  - So that for a new dataset D_k sampled from an unknown but new function g^k, θ could be rapidly adapted to D_k leading to θ_k (see figure) using a few gradient steps

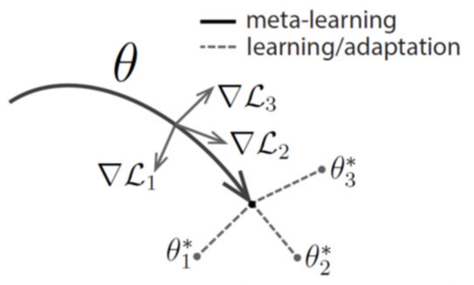


Figure 1. Diagram of our model-agnostic meta-learning algorithm (MAML), which optimizes for a representation θ that can quickly adapt to new tasks.

- It has been used in several contexts
  - regression, classification, reinforcement learning, etc. and for several problems
- In the following we will be interested in few shot learning problems:
  - how to learn a new regression function from a small number of examples in D

Neural process (NP)

- Training NP relies on a meta-learning instance
- For NP, each set D representing an unknown function g will be devided in two sets:
  - A context set $C = \{(x^i, y^i), i = 1 \dots N\}$ and target set $T = \{(x^i, y^i), i = N + 1 \dots N + M\}$
  - Let us denote $X_T = \{x_i; i = N + 1 \dots N + M\}, Y_T = \{y_i; i = N + 1 \dots N + M\}$ the sets of inputs and outputs for the target set T
- Objective
  - The goal is to learn a context dependent function $f(x; C)$
  - After training, given any context set C, $f(.; C)$ should be able to compute an output $f(.; C)$ for each new input x
  - Training $f(.; C)$ will make use of a series of tasks each represented by a dataset D^k
  - After training, for any context $C(g)$ sampled from a new unknown function g, one should be able to perform inference, i.e. computing $f(x; C(g))$ and the uncertainty associated to this prediction

Conditional Neural process

- Implementation for the Conditional Neural Process (CNP)
- f will be implemented with two components
  - An encoder $Enc_{\theta}$and a decoder Dec_{θ}
  - Both will be implemented by neural networks
  - The encoder will encode the context C into a vector representation R
  - The decoder will compute a mean value and its associated uncertainty (like in GP) – remember we want to learn stochastic processes

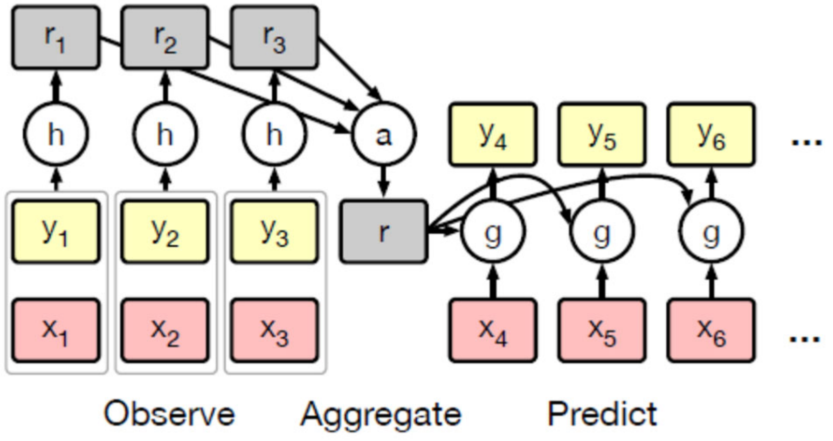
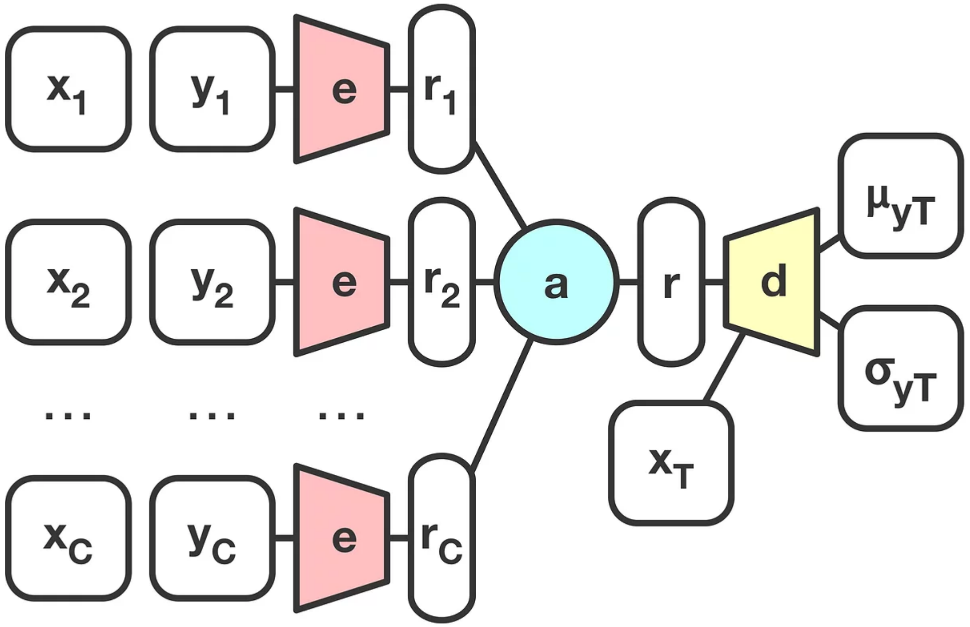


Fig. Garnelo et al. 2018
In this figure, the encoder is denoted h and the decoder g, the encoding of the context is denoted r

Conditional Neural process

- More precisely, once $f(x; C)$ is learned , given any new context C_{new} and any input x that could be in C_{new} or not but sampled from the same function g_{new}, one will encode C_{new} through a vectorial representation R using the encoder Enc_f and compute via the decoder Dec_f the prediction for the input x, in our example this prediction will be the mean and variance associated to x: $Dec_f = (\mu_f(x, R), \sigma_f(x, R))$



Conditional Neural process - illustration

Gif: Garnelo

Conditional Neural process

- In the CNP
  - $R = Enc_\theta(C) = \frac{1}{|C|}\sum_{(x^i,y^i)\in C} MLP([x^i;y^i])$
    - $[x^i;y^i]$ is the concatenation of the two vectors
  - $(\mu(x),\sigma(x)) = Dec_\theta(x,R) = MLP([x;R])$

Conditional Neural processn- Training

- We suppose available a set of tasks
  - In our example, a task will correspond to a function to be regressed, i.e. to a dataset D_k
- Meta-learning algorithm
  - Sample a task (a set) $D \sim P$
  - Split the task randomly into context and target sets $D = C \cup T$
  - Compute the predictive distribution of the outputs for the target points $p_\theta(Y_T|X_T; C)$
    - i.e. $\forall x \in T$, compute $\mathrm{f}_\theta(x; C)$ and then $p_\theta(Y_T|X_T; C)$ for the dataset $T = \{X_T, Y_T\}$
  - Compute the loglikelihood, i.e. measure the performance of f_θ on T
    - $\mathrm{L} = \log p_\theta(Y_T|X_T; C)$
  - Update the loglikelihood, e.g.
    - $\theta = \theta - \epsilon \nabla_\theta L$

Conditional Neural process - illustration

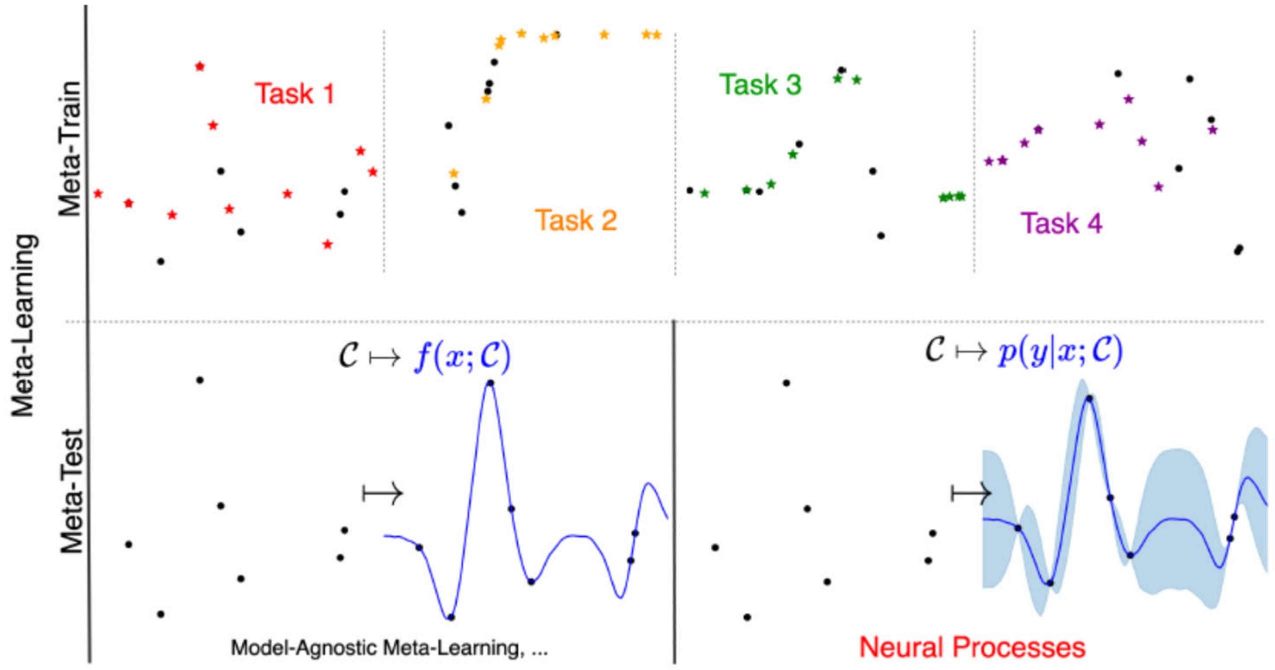


Fig. 3 Comparison between meta learning vs supervised learning, and modeling functions vs modeling stochastic processes. Neural Processes are in the lower-right quadrant. Dot are context points while stars are target points.

https://yanndubs.github.io/Neural-Process-Family/text/Intro.html

Conditional Neural process - Training

- How to compute the likelihood?
- The predictive distribution is factorized (independence assumption)
  - $p_\theta(Y_T|X_T;C) = \prod_{(x^i,y^i)\in T} p_\theta(y^i|x^i;R)$
  - $p_\theta(Y_T|X_T;C) = \prod_{(x^i,y^i)\in T} \mathcal{N}(y^i;\mu(x^i),\sigma(x^i))$
  - With
  - $R = Enc_\theta(C)$ Encoding
  - $\left(\mu(x^i),\sigma(x^i)\right) = Dec_\theta(x^i,R)$ Decoding

  - Note: most often, the log likelihood is optimized on the whole dataset
  - $\mathrm{L} = \log p_\theta(Y_D|X_D;C)$ instead of $\mathrm{L} = \log p_\theta(Y_T|X_T;C)$
    - i.e. the log likelihood is evaluated onto the whole dataset $D = C \cup T$ instead of onto T only

Conditional Neural process

- The context set C is treated as a set, i.e. the order of the elements in the context set does not influence the predictor
  - i.e. the predictor should be permutation invariant: $p_\theta(Y_T|X_T; C) = p_\theta(Y_T|X_T; \pi(C))$ for a permutation π

- Weaknesses
  - CNP are known to underfit

Conditional Neural process - Examples

- Regression (Garnelo et al. 2018)
- A task corresponds to a curve

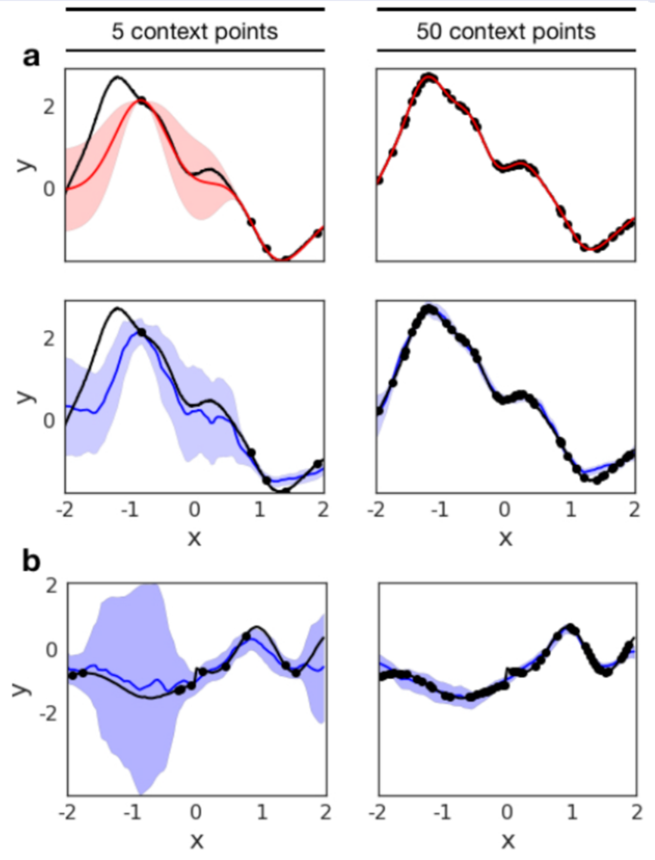


Figure 2. **1-D Regression**. Regression results on a 1-D curve (black line) using 5 (left column) and 50 (right column) context points (black dots). The first two rows show the predicted mean and variance for the regression of a single underlying kernel for GPs (red) and CNPs (blue). The bottom row shows the predictions of CNPs for a curve with switching kernel parameters.

Neural process – Examples (Garnelo 2018 Neural Processes)

- Regression: pixelwise prediction- a task corresponds to an image

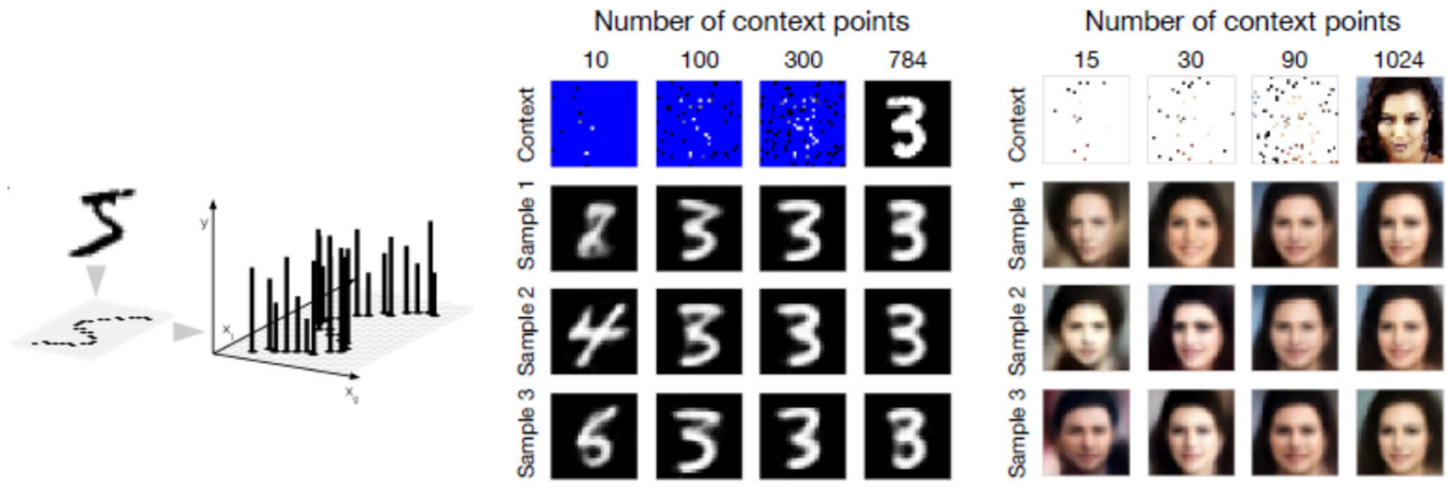


Figure 4. **Pixel-wise regression on MNIST and CelebA** The diagram on the left visualises how pixel-wise image completion can be framed as a 2-D regression task where f(pixel coordinates) = pixel brightness. The figures to the right of the diagram show the results on image completion for MNIST and CelebA. The images on the top correspond to the context points provided to the model. For better clarity the unobserved pixels have been coloured blue for the MNIST images and white for CelebA. Each of the rows corresponds to a different sample given the context points. As the number of context points increases the predicted pixels get closer to the underlying ones and the variance across samples decreases.

Neural process - references

- Garnelo, M., Rosenbaum, D., Maddison, C. J., Ramalho, T., Saxton, D., Shanahan, M., Teh, Y. W., Rezende, D. J., & Ali Eslami, S. M. (2018). Conditional neural processes. *ICML*, 1704–1713.
- Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Ali Eslami, S. M., & Teh, Y. W. (2018). Neural processes. *ArXiv*.
- See also
- J. Gordon, Advances in Probabilistic Meta-Learning and the Neural Process Family, PhD thesis, 2020
- https://yanndubs.github.io/Neural-Process-Family/text/Intro.html