SVAN 2016 Mini-Course
Stochastic Convex Optimization Methods in Machine Learning

Mark Schmidt

University of British Columbia, May 2016

www.cs.ubc.ca/~schmidtm/SVAN16
Coordinate Optimization vs. Stochastic Gradient

- Consider optimization problem:
  \[
  \arg\min_{x \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} f_i(x).
  \]

- **Coordinate optimization**: update one \( x_j \) based on all examples:
  - Fast convergence rate, but iterations must be \( d \) times cheaper than gradient method.
  - Functions \( f_i \) must be smooth.
Coordinate Optimization vs. Stochastic Gradient

Consider optimization problem:

$$\arg\min_{x \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} f_i(x).$$

- **Coordinate optimization**: update one $x_j$ based on all examples:
  - Fast convergence rate, but iterations must be $d$ times cheaper than gradient method.
  - Functions $f_i$ must be smooth.

- **Stochastic gradient**: update all $x_i$ based on one example:
  - Slow convergence rate, and iterations are $d$ times cheaper than gradient method.
  - Functions $f_i$ can be non-smooth.
Coordinate Optimization vs. Stochastic Gradient

- Consider optimization problem:

\[
\arg\min_{x \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} f_i(x).
\]

- **Coordinate optimization**: update one \(x_j\) based on all examples:
  - Fast convergence rate, but iterations must be \(d\) times cheaper than gradient method.
  - Functions \(f_i\) must be smooth.

- **Stochastic gradient**: update all \(x_i\) based on one example:
  - Slow convergence rate, and iterations are \(d\) times cheaper than gradient method.
  - Functions \(f_i\) can be non-smooth.

- **SAG**: update all \(x_i\) based on one example (and old versions of others):
  - Fast convergence rate, and iterations are \(d\) times cheaper than gradient method.
  - Functions \(f_i\) must be smooth.
Motivation: Multi-Dimensional Polynomial Basis

- Recall using polynomial basis when we only have one feature \( x_i \in \mathbb{R} \):

\[
\hat{y}_i = \beta + w_1 x_i + w_2 x_i^2.
\]
Motivation: Multi-Dimensional Polynomial Basis

- Recall using polynomial basis when we only have one features \( x_i \in \mathbb{R} \):
  \[
  \hat{y}_i = \beta + w_1 x_i + w_2 x_i^2.
  \]

- We can fit these models using a change of basis:

  If \( X = \begin{bmatrix} 0.2 \\ -0.5 \\ 1 \\ 4 \end{bmatrix} \) then let
  \[
  \Phi(X) = \begin{bmatrix} 1 & 0.2 & (0.2)^2 \\ 1 & -0.5 & (-0.5)^2 \\ 1 & 1 & (1)^2 \\ 1 & 4 & (4^2) \end{bmatrix},
  \]

  and L2-regularized least squares solution is

  \[
  w = (\Phi(X)^T \Phi(X) + \lambda I)^{-1} \Phi(X)^T y.
  \]
Motivation: Multi-Dimensional Polynomial Basis

- Recall using polynomial basis when we only have one feature \((x_i \in \mathbb{R})\):

  \[
  \hat{y}_i = \beta + w_1 x_i + w_2 x_i^2.
  \]

- We can fit these models using a change of basis:

  If \(X = \begin{bmatrix} 0.2 \\ -0.5 \\ 1 \\ 4 \end{bmatrix}\) then let \(\Phi(X) = \begin{bmatrix} 1 & 0.2 & (0.2)^2 \\ 1 & -0.5 & (-0.5)^2 \\ 1 & 1 & (1)^2 \\ 1 & 4 & (4^2) \end{bmatrix}\),

  and L2-regularized least squares solution is

  \[
  w = (\Phi(X)^T \Phi(X) + \lambda I)^{-1} \Phi(X)^T y.
  \]

- How can we do this when we have a lot of features?
Motivation: Multi-Dimensional Polynomial Basis

- Approach 1: use polynomial basis for each variable:

\[
X = \begin{bmatrix}
0.2 & 0.3 \\
1 & 0.5 \\
-0.5 & -0.1
\end{bmatrix} \Rightarrow \Phi(X) = \begin{bmatrix}
1 & 0.2 & (0.2)^2 & 0.3 & (0.3)^2 \\
1 & 1 & (1)^2 & 0.5 & (0.5)^2 \\
1 & -0.5 & (-0.5)^2 & -0.1 & (-0.1)^2
\end{bmatrix}
\]
Motivation: Multi-Dimensional Polynomial Basis

- Approach 1: use polynomial basis for each variable:

\[
X = \begin{bmatrix}
0.2 & 0.3 \\
1 & 0.5 \\
-0.5 & -0.1
\end{bmatrix} \Rightarrow \Phi(X) = \begin{bmatrix}
1 & 0.2 & (0.2)^2 & 0.3 & (0.3)^2 \\
1 & 1 & (1)^2 & 0.5 & (0.5)^2 \\
1 & -0.5 & (-0.5)^2 & -0.1 & (-0.1)^2
\end{bmatrix}
\]

- But this is restrictive:
  - We should allow terms like \( x_{i1}x_{i2} \) that depend on feature interactions.
  - But number of terms in \( X_{\text{poly}} \) would be huge:
    - Degree-5 polynomial basis has \( O(d^5) \) terms:
      \[
x_{i1}^5, x_{i1}^4x_{i2}, x_{i1}^4x_{i3}, \ldots, x_{i1}^3x_{i2}^2, x_{i1}^3x_{i2}, \ldots, x_{i1}x_{i2}x_{i3}, \ldots
\]
  - If \( n \) is not too big, we can do this efficiently using the \textit{kernel trick}.  

Equivalent Form of Ridge Regression

- Recall the L2-regularized least squares model,
  \[
  \arg\min_{w \in \mathbb{R}^d} \frac{1}{2} \|Xw - y\|^2 + \frac{\lambda}{2} \|w\|^2.
  \]

- We showed that the solution is
  \[
  w = (X^T X + \lambda I_d)^{-1} X^T y,
  \]
  where \(I_d\) is the \(d\) by \(d\) identity matrix.
Equivalent Form of Ridge Regression

- Recall the L2-regularized least squares model,
  \[
  \text{argmin}_{w \in \mathbb{R}^d} \frac{1}{2} \|Xw - y\|^2 + \frac{\lambda}{2} \|w\|^2.
  \]

- We showed that the solution is
  \[
  w = (X^TX + \lambda I_d)^{-1}X^T y,
  \]
  where \( I_d \) is the \( d \) by \( d \) identity matrix.

- An equivalent way to write the solution is:
  \[
  w = X^T(XX^T + \lambda I_n)^{-1}y,
  \]
  by using a variant of the matrix inversion lemma.

- Computing \( w \) with this formula is faster if \( n << d \):
  - since \( XX^T \) is \( n \) by \( n \) while \( X^T X \) is \( d \) by \( d \).
Predictions using Equivalent Form

Given test data $\hat{X}$, we predict $\hat{y}$ using:

$$\hat{y} = \hat{X}w$$

$$= \hat{X}X^T(XX^T + \lambda I_n)^{-1}y$$
Predictions using Equivalent Form

- Given test data $\hat{X}$, we predict $\hat{y}$ using:

  $$
  \hat{y} = \hat{X}w 
  = \hat{X}X^T (XX^T + \lambda I_n)^{-1}y
  $$

- If we define $K = XX^T$ (Gram matrix) and $\hat{K} = \hat{X}X^T$, then we have

  $$
  \hat{y} = \hat{K}(K + \lambda I_n)^{-1}y.
  $$

- Key observation behind kernel trick:
  - If we have the $K$ and $\hat{K}$, we don't need the features.
Gram Matrix

- The **Gram matrix** $K$ is defined by:

  $$K = XX^T = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \\ x_2 & \vdots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ x_n & \vdots & \ddots & x_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

  $$= \begin{bmatrix} x_1^T x_1 & x_1^T x_2 & \cdots & x_1^T x_n \\ x_2^T x_1 & x_2^T x_2 & \cdots & x_2^T x_n \\ \vdots & \vdots & \ddots & \vdots \\ x_n^T x_1 & x_n^T x_2 & \cdots & x_n^T x_n \end{bmatrix}$$

- $K$ contains the **inner products** between all training examples.
Gram Matrix

- The **Gram matrix** \( K \) is defined by:

\[
K = XX^T = \begin{bmatrix}
\vdots & \vdots & \vdots \\
x_1 & x_2 & x_3 \\
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
\vdots \\
x_n \\
\end{bmatrix}
\]

\[
= \begin{bmatrix}
x_1^T x_1 & x_1^T x_2 & \cdots & x_1^T x_n \\
x_2^T x_1 & x_2^T x_2 & \cdots & x_2^T x_n \\
\vdots & \vdots & \ddots & \vdots \\
x_n^T x_1 & x_n^T x_2 & \cdots & x_n^T x_n \\
\end{bmatrix}
\]

- \( K \) contains the **inner products** between all training examples.
- \( \hat{K} \) contains the **inner products** between training and test examples.
  - If we can compute inner products \( k(x_i, x_j) = x_i^T x_j \), we don't need \( x_i \) and \( x_j \).
Polynomial Kernel

- Consider two examples $x_i$ and $x_j$ for a two-dimensional dataset:
  \[ x_i = (x_{i1}, x_{i2}), \quad x_j = (x_{j1}, x_{j2}). \]

- Consider a particular degree-2 basis $\phi$:
  \[ \phi(x_i) = (x_{i1}^2, \sqrt{2}x_{i1}x_{i2}, x_{i2}^2). \]
Polynomial Kernel

- Consider two examples $x_i$ and $x_j$ for a two-dimensional dataset:
  
  \[ x_i = (x_{i1}, x_{i2}), \quad x_j = (x_{j1}, x_{j2}). \]

- Consider a particular degree-2 basis $\phi$:
  
  \[ \phi(x_i) = (x_{i1}^2, \sqrt{2}x_{i1}x_{i2}, x_{i2}^2). \]

- We can compute inner product $\phi(x_i)^T \phi(x_j)$ without forming $\phi(x_i)$ and $\phi(x_j)$,
Polynomial Kernel

- Consider two examples $x_i$ and $x_j$ for a two-dimensional dataset:
  \[ x_i = (x_{i1}, x_{i2}), \quad x_j = (x_{j1}, x_{j2}). \]

- Consider a particular degree-2 basis $\phi$:
  \[ \phi(x_i) = (x_{i1}^2, \sqrt{2}x_{i1}x_{i2}, x_{i2}^2). \]

- We can compute inner product $\phi(x_i)^T \phi(x_j)$ without forming $\phi(x_i)$ and $\phi(x_j)$,
  \[
  \phi(x_i)^T \phi(x_j) = \begin{bmatrix} x_{i1}^2 & \sqrt{2}x_{i1}x_{i2} & x_{i2}^2 \end{bmatrix} \phi(x_j) \\
  = x_{i1}^2 x_{j1}^2 + 2x_{i1}x_{i2}x_{j1}x_{j2} + x_{i2}^2 x_{j2}^2 \\
  = (x_{i1}x_{j1} + x_{i2}x_{j2})^2 \\
  = \left( \sum_{k=1}^{d} x_{ik}x_{jk} \right)^2 \\
  = (x^T_i x_j)^2.
  \]
Polynomial Kernel with Higher Degrees

If we want all degree-4 “monomials”, raise to 4\textsuperscript{th} power:

$$\phi(x_i)^T \phi(x_j) = (x_i^T x_j)^4,$$

where $\phi(x_i)$ is weighted version of $x_{i1}^4, x_{i1}^3 x_{i2}, x_{i1}^2 x_{i2}^2, x_{i1} x_{i2}^3, x_{i2}^4$. 
Polynomial Kernel with Higher Degrees

- If we want all degree-4 “monomials”, raise to 4\(^{th}\) power:
  \[
  \phi(x_i)^T \phi(x_j) = (x_i^T x_j)^4,
  \]
  where \(\phi(x_i)\) is weighted version of \(x_{i1}^4, x_{i1}^3 x_{i2}, x_{i1}^2 x_{i2}^2, x_{i1} x_{i2}^3, x_{i2}^4\).

- If you want bias or lower-order terms like \(x_{i1}\), add constant inside power:
  \[
  (1 + x_i^T x_j)^2 = 1 + 2x_i^T x_j + (x_i^T x_j)^2
  \]
  
  \[
  = \begin{bmatrix}
  1 & 2x_{i1} & 2x_{i2} & x_{i1}^2 & x_{i2}^2 \\
  2x_{j1} & 2x_{j2} & x_{j1}^2 & \sqrt{2}x_{j1}x_{j2} & x_{j2}^2
  \end{bmatrix}
  = \phi(x_i)^T \phi(x_j),
  \]

- These formulas still work for any dimension of the \(x_i\).
Kernel Trick

- Using polynomial basis of degree ‘p’ with the kernel trick:
  - Compute $K$ and $\hat{K}$ which have elements:
    
    $$k(x_i, x_j) = (1 + x_i^T x_j)^p, \quad \hat{k}(\hat{x}_i, x_j) = (1 + \hat{x}_i^T x_j)^p.$$  

  - Make predictions using:
    
    $$\hat{y} = \hat{K}(K + \lambda I)^{-1}y.$$
Kernel Trick

- Using polynomial basis of degree ‘p’ with the kernel trick:
  - Compute $K$ and $\hat{K}$ which have elements:
    \[ k(x_i, x_j) = (1 + x_i^T x_j)^p, \quad \hat{k}(\hat{x}_i, x_j) = (1 + \hat{x}_i^T x_j)^p. \]
  - Make predictions using:
    \[ \hat{y} = \hat{K}(K + \lambda I)^{-1}y. \]
  - Cost is $O(n^2d + n^3)$ even though number of features is $O(d^p)$. 
Kernel Trick

Using polynomial basis of degree ‘p’ with the kernel trick:

- Compute $K$ and $\hat{K}$ which have elements:
  
  \[
k(x_i, x_j) = (1 + x_i^T x_j)^p, \quad \hat{k}(\hat{x}_i, x_j) = (1 + \hat{x}_i^T x_j)^p.
  \]

- Make predictions using:
  
  \[
  \hat{y} = \hat{K}(K + \lambda I)^{-1} y.
  \]

- Cost is $O(n^2d + n^3)$ even though number of features is $O(d^p)$.

Kernel trick lets us fit regression models without explicit feature calculation:

- Features may have exponential or infinite size.
The most common kernel is the Gaussian-RBF (or ‘squared exponential’) kernel,
\[ k(x_i, x_j) = \exp \left( -\frac{\|x_i - x_j\|^2}{\sigma^2} \right). \]
What function \( \phi(x) \) would lead to this as the inner-product?
Guasian-RBF Kernels

- The most common kernel is the **Gaussian-RBF** (or ‘squared exponential’) kernel,
  \[ k(x_i, x_j) = \exp \left( -\frac{\|x_i - x_j\|^2}{\sigma^2} \right). \]

- What function \( \phi(x) \) would lead to this as the inner-product?
  - To simplify, assume \( d = 1 \) and \( \sigma = 1 \),
  \[ k(x_i, x_j) = \exp(-x_i^2 + 2x_i x_j - x_j^2) \]
  \[ = \exp(-x_i^2) \exp(2x_i x_j) \exp(-x_j^2), \]
Guasian-RBF Kernels

The most common kernel is the Gaussian-RBF (or ‘squared exponential’) kernel,

$$k(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{\sigma^2}\right).$$

What function $\phi(x)$ would lead to this as the inner-product?

To simplify, assume $d = 1$ and $\sigma = 1$,

$$k(x_i, x_j) = \exp(-x_i^2 + 2x_i x_j - x_j^2)$$

$$= \exp(-x_i^2) \exp(2x_i x_j) \exp(-x_j^2),$$

so we need $\phi(x_i) = \exp(-x_i^2)z_i$ where $z_i z_j = \exp(2x_i x_j)$.

For this to work for all $x_i$ and $x_j$, $z_i$ must be infinite-dimensional.
**Guasian-RBF Kernels**

- The most common kernel is the **Gaussian-RBF** (or ‘squared exponential’) kernel,
  
  \[ k(x_i, x_j) = \exp \left( -\frac{\|x_i - x_j\|^2}{\sigma^2} \right). \]

- What function \( \phi(x) \) would lead to this as the inner-product?
  - To simplify, assume \( d = 1 \) and \( \sigma = 1 \),
    
    \[ k(x_i, x_j) = \exp(-x_i^2 + 2x_i x_j - x_j^2) = \exp(-x_i^2) \exp(2x_i x_j) \exp(-x_j^2), \]
  
  so we need \( \phi(x_i) = \exp(-x_i^2)z_i \) where \( z_i z_j = \exp(2x_i x_j) \).
  
  - For this to work for all \( x_i \) and \( x_j \), \( z_i \) must be infinite-dimensional.
  
  - If we use that
    
    \[ \exp(2x_i x_j) = \sum_{k=0}^{\infty} \frac{2^k x_i^k x_j^k}{k!}, \]
  
    then we obtain
    
    \[ \phi(x_i) = \exp(-x_i^2) \left[ 1 \quad \sqrt{\frac{2}{1!}} x_i \quad \sqrt{\frac{2^2}{2!}} x_i^2 \quad \sqrt{\frac{2^3}{3!}} x_i^3 \quad \cdots \right]. \]
Kernel Trick for Structured Data

- Kernel trick is useful for structured data:
  - Consider data that doesn’t look like this:
    
    \[
    X = \begin{bmatrix}
    0.5377 & 0.3188 & 3.5784 \\
    1.8339 & -1.3077 & 2.7694 \\
    -2.2588 & -0.4336 & -1.3499 \\
    0.8622 & 0.3426 & 3.0349
    \end{bmatrix}, \quad
    y = \begin{bmatrix}
    +1 \\
    -1 \\
    -1 \\
    +1
    \end{bmatrix},
    \]

- But instead looks like this:
  - Do you want to go for a drink sometime?
  - J’achète du pain tous les jours.
  - Fais ce que tu veux.
  - There are inner products between sentences?

- We could convert sentences to features, or define kernel between sentences.
  - For example, “string” kernels:
    - Weighted frequency of common subsequences (dynamic programming).
  - There are also “graph kernels”, “image kernels”, and so on...
Kernel Trick for Structured Data

Kernel trick is useful for structured data:

Consider data that doesn’t look like this:

\[ X = \begin{bmatrix}
0.5377 & 0.3188 & 3.5784 \\
1.8339 & -1.3077 & 2.7694 \\
-2.2588 & -0.4336 & -1.3499 \\
0.8622 & 0.3426 & 3.0349
\end{bmatrix}, \quad y = \begin{bmatrix} +1 \\
-1 \\
-1 \\
+1 \end{bmatrix}, \]

but instead looks like this:

\[ X = \begin{bmatrix}
\text{Do you want to go for a drink sometime?} \\
\text{J’achète du pain tous les jours.} \\
\text{Fais ce que tu veux.} \\
\text{There are inner products between sentences?}
\end{bmatrix}, \quad y = \begin{bmatrix} +1 \\
-1 \\
-1 \\
+1 \end{bmatrix}. \]
Kernel Trick for Structured Data

Kernel trick is useful for structured data:

Consider data that doesn’t look like this:

\[
X = \begin{bmatrix}
0.5377 & 0.3188 & 3.5784 \\
1.8339 & -1.3077 & 2.7694 \\
-2.2588 & -0.4336 & -1.3499 \\
0.8622 & 0.3426 & 3.0349
\end{bmatrix}, \quad y = \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \end{bmatrix},
\]

but instead looks like this:

\[
X = \begin{bmatrix}
\text{Do you want to go for a drink sometime?} \\
\text{J’achète du pain tous les jours.} \\
\text{Fais ce que tu veux.} \\
\text{There are inner products between sentences?}
\end{bmatrix}, \quad y = \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \end{bmatrix}.
\]

We could convert sentences to features, or define kernel between sentences.

For example, “string” kernels:

- Weighted frequency of common subsequences (dynamic programming).
- There are also “graph kernels”, “image kernels”, and so on...
Valid Kernels

- What kernel functions $k(x_i, x_j)$ can we use?
- Kernel $k$ must be an inner product in some space:
  - There exists $\phi$ such that $k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$. 

If we use invalid kernel, lose inner-product interpretation but may work fine.
Valid Kernels

- What kernel functions $k(x_i, x_j)$ can we use?
- Kernel $k$ must be an inner product in some space:
  - There exists $\phi$ such that $k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$.

We can decompose a (continuous or finite-domain) function $k$ into

$$k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle,$$

iff it is symmetric and for any finite $\{x_1, x_2, \ldots, x_n\}$ we have $K \succeq 0$. 

Valid Kernels

- What kernel functions $k(x_i, x_j)$ can we use?
- Kernel $k$ must be an inner product in some space:
  - There exists $\phi$ such that $k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$.

We can decompose a (continuous or finite-domain) function $k$ into

$$k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle,$$

iff it is symmetric and for any finite $\{x_1, x_2, \ldots, x_n\}$ we have $K \succeq 0$.

- Nice in theory, what do we do in practice?
  - Show explicitly that $k(x_i, x_j)$ is an inner product.
  - Or show it can be constructed from other valid kernels.
Valid Kernels

- What kernel functions $k(x_i, x_j)$ can we use?
- Kernel $k$ must be an inner product in some space:
  - There exists $\phi$ such that $k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$.

We can decompose a (continuous or finite-domain) function $k$ into

$$k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle,$$

iff it is symmetric and for any finite $\{x_1, x_2, \ldots, x_n\}$ we have $K \succeq 0$.

- Nice in theory, what do we do in practice?
  - Show explicitly that $k(x_i, x_j)$ is an inner product.
  - Or show it can be constructed from other valid kernels.
- If we use invalid kernel, lose inner-product interpretation but may work fine.
Bonus Slide: Constructing Feature Space

Why is positive semi-definiteness important?
- With finite domain we can define $K$ over all points.
- The condition $K \succeq 0$ means it has a spectral decomposition

$$K = U^T \Lambda U,$$

where the eigenvalues $\lambda_i \geq 0$ and so we have a real $\Lambda^{\frac{1}{2}}$.
- Thus we have $K = U^T \Lambda^{\frac{1}{2}} \Lambda^{\frac{1}{2}} U = \| \Lambda^{\frac{1}{2}} U \|^2$ and we could use

$$\Phi(X) = \Lambda^{\frac{1}{2}} U, \text{ or } \phi(x_i) = \Lambda^{\frac{1}{2}} U_{:,i}.$$

The above reasoning isn’t quite right for continuous domains.
The more careful generalization is known as “Mercer’s theorem”.
Constructing Valid Kernels

- If $k_1(x_i, x_j)$ and $k_2(x_i, x_j)$ are valid kernels, then the following are valid kernels:
  - $k_1(\phi(x_i), \phi(x_j))$. 
  - $\alpha k_1(x_i, x_j) + \beta k_2(x_i, x_j)$ for $\alpha \geq 0$ and $\beta \geq 0$. 
  - $k_1(x_i, x_j) k_2(x_i, x_j)$. 
  - $\phi(x_i) k_1(x_i, x_j) \phi(x_j)$. 
  - $\exp(k_1(x_i, x_j))$. 
  - $\exp(-\|x_i - x_j\|^2/\sigma^2)$ for $\alpha \geq 0$. 
  - $\exp(-\|x_j\|^2/\sigma^2) \phi(x_j)$. 
  - $\exp(\alpha x_i^T x_j)$ for valid $\alpha$. 
  - $\exp(-\|x_i - x_j\|^2/\sigma^2) \phi(x_j)$. 

Constructing Valid Kernels

- If \( k_1(x_i, x_j) \) and \( k_2(x_i, x_j) \) are valid kernels, then the following are valid kernels:
  - \( k_1(\phi(x_i), \phi(x_j)) \).
  - \( \alpha k_1(x_i, x_j) + \beta k_2(x_i, x_j) \) for \( \alpha \geq 0 \) and \( \beta \geq 0 \).
Constructing Valid Kernels

- If $k_1(x_i, x_j)$ and $k_2(x_i, x_j)$ are valid kernels, then the following are valid kernels:
  - $k_1(\phi(x_i), \phi(x_j))$.
  - $\alpha k_1(x_i, x_j) + \beta k_2(x_i, x_j)$ for $\alpha \geq 0$ and $\beta \geq 0$.
  - $k_1(x_i, x_j)k_2(x_i, x_j)$. 
Constructing Valid Kernels

If $k_1(x_i, x_j)$ and $k_2(x_i, x_j)$ are valid kernels, then the following are valid kernels:

- $k_1(\phi(x_i), \phi(x_j))$.
- $\alpha k_1(x_i, x_j) + \beta k_2(x_i, x_j)$ for $\alpha \geq 0$ and $\beta \geq 0$.
- $k_1(x_i, x_j)k_2(x_i, x_j)$.
- $\phi(x_i)k_1(x_i, x_j)\phi(x_j)$.
Constructing Valid Kernels

If $k_1(x_i, x_j)$ and $k_2(x_i, x_j)$ are valid kernels, then the following are valid kernels:

- $k_1(\phi(x_i), \phi(x_j))$.
- $\alpha k_1(x_i, x_j) + \beta k_2(x_i, x_j)$ for $\alpha \geq 0$ and $\beta \geq 0$.
- $k_1(x_i, x_j)k_2(x_i, x_j)$.
- $\phi(x_i)k_1(x_i, x_j)\phi(x_j)$.
- $\exp(k_1(x_i, x_j))$. 

Example: Gaussian-RBF kernel:

$$k(x_i, x_j) = \exp\left(\frac{-\|x_i - x_j\|^2}{\sigma^2}\right) = \exp\left(\frac{-\|x_i\|^2}{\sigma^2}\right)\phi(x_i) \exp\left(\frac{-\|x_j\|^2}{\sigma^2}\right)\phi(x_j).$$
Constructing Valid Kernels

- If $k_1(x_i, x_j)$ and $k_2(x_i, x_j)$ are valid kernels, then the following are valid kernels:
  - $k_1(\phi(x_i), \phi(x_j))$.
  - $\alpha k_1(x_i, x_j) + \beta k_2(x_i, x_j)$ for $\alpha \geq 0$ and $\beta \geq 0$.
  - $k_1(x_i, x_j)k_2(x_i, x_j)$.
  - $\phi(x_i)k_1(x_i, x_j)\phi(x_j)$.
  - $\exp(k_1(x_i, x_j))$.

- Example: Gaussian-RBF kernel:

$$k(x_i, x_j) = \exp \left( -\frac{\|x_i - x_j\|^2}{\sigma^2} \right)$$

$$= \exp \left( -\frac{\|x_i\|^2}{\sigma^2} \right) \exp \left( \frac{2}{\sigma^2} x_i^T x_j \alpha \geq 0 \text{ valid} \right) \exp \left( -\frac{\|x_j\|^2}{\sigma^2} \right).$$
Kernels Trick for Distance-Based Methods

- Besides ridge regression, when can we apply the kernel trick?
Kernels Trick for Distance-Based Methods

Besides ridge regression, when can we apply the kernel trick?

- **Distance-based** methods (see my undergrad course):

\[ \|x_i - x_j\|^2 = \langle x_i, x_j \rangle - 2\langle x_i, x_j \rangle + \langle x_j, x_j \rangle. \]
Kernels Trick for Distance-Based Methods

Besides ridge regression, when can we apply the kernel trick?

- **Distance-based** methods (see my undergrad course):

  \[ \|x_i - x_j\|^2 = \langle x_i, x_j \rangle - 2 \langle x_i, x_j \rangle + \langle x_j, x_j \rangle. \]

  - \(k\)-nearest neighbours.
  - Clustering algorithms (\(k\)-means, density-based clustering, hierarchical clustering).
  - Amazon item-to-item product recommendation.
  - Non-parametric regression.
  - Outlier ratio.
  - Multi-dimensional scaling.
  - Graph-based semi-supervised learning.
Kernels Trick for Distance-Based Methods

Besides ridge regression, when can we apply the kernel trick?

- **Distance-based** methods (see my undergrad course):

\[ \|x_i - x_j\|^2 = \langle x_i, x_j \rangle - 2\langle x_i, x_j \rangle + \langle x_j, x_j \rangle. \]

- \(k\)-nearest neighbours.
- Clustering algorithms (\(k\)-means, density-based clustering, hierarchical clustering).
- Amazon item-to-item product recommendation.
- Non-parametric regression.
- Outlier ratio.
- Multi-dimensional scaling.
- Graph-based semi-supervised learning.

- **Eigenvalue** methods:
  - Principle component analysis (trick for centering in high-dimensional space).
  - Canonical correlation analysis.
  - Spectral clustering.

- **L2-regularized linear models**...
Representer Theorem

- Consider linear model differentiable with losses $f_i$ and L2-regularization,

$$\arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} f_i(w^T x_i) + \frac{\lambda}{2} \|w\|^2.$$
Representer Theorem

- Consider linear model differentiable with losses \( f_i \) and L2-regularization,

\[
\arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} f_i(w^T x_i) + \frac{\lambda}{2} \|w\|^2.
\]

- Setting the gradient equal to zero we get

\[
0 = \sum_{i=1}^{n} f'_i(w^T x_i)x_i + \lambda w.
\]
Representer Theorem

- Consider linear model differentiable with losses $f_i$ and L2-regularization,

$$\arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} f_i(w^T x_i) + \frac{\lambda}{2} \|w\|^2.$$ 

- Setting the gradient equal to zero we get

$$0 = \sum_{i=1}^{n} f_i'(w^T x_i) x_i + \lambda w.$$ 

- So any solution $w^*$ can written as a linear combination of features $x_i$, 

$$w^* = -\frac{1}{\lambda} \sum_{i=1}^{n} f_i'((w^*)^T x_i) x_i = \sum_{i=1}^{n} z_i x_i 
= X^T z.$$ 

- This is called a representer theorem (true under much more general conditions).
Representer Theorem

Using representer theorem we can use \( w = X^T z \) in original problem,

\[
\arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} f_i(w^T x_i) + \frac{\lambda}{2} \|w\|^2
\]

\[
= \arg\min_{z \in \mathbb{R}^n} \sum_{i=1}^{n} f_i(z^T X x_i) + \frac{\lambda}{2} \|X^T z\|^2
\]

Similarly, at test time we can use the variables \( \hat{X}_w = \hat{X} X^T z \).
Representer Theorem

Using representer theorem we can use $w = X^T z$ in original problem,

$$\arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} f_i(w^T x_i) + \frac{\lambda}{2} \|w\|^2$$

$$= \arg\min_{z \in \mathbb{R}^n} \sum_{i=1}^{n} f_i(z^T X x_i) + \frac{\lambda}{2} \|X^T z\|^2$$

Now defining $f(z) = \sum_{i=1}^{n} f_i(z_i)$ for a vector $z$ we have

$$= \arg\min_{z \in \mathbb{R}^n} f(X X^T z) + \frac{\lambda}{2} z^T X X^T z$$

$$= \arg\min_{z \in \mathbb{R}^n} f(K z) + \frac{\lambda}{2} z^T K z.$$
Representer Theorem

- Using representer theorem we can use $w = X^T z$ in original problem,

$$\arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} f_i(w^T x_i) + \frac{\lambda}{2} \|w\|^2$$

$$= \arg\min_{z \in \mathbb{R}^n} \sum_{i=1}^{n} f_i(z^T X x_i) + \frac{\lambda}{2} \|X^T z\|^2$$

- Now defining $f(z) = \sum_{i=1}^{n} f_i(z_i)$ for a vector $z$ we have

$$= \arg\min_{z \in \mathbb{R}^n} f(X X^T z) + \frac{\lambda}{2} z^T X X^T z$$

$$= \arg\min_{z \in \mathbb{R}^n} f(Kz) + \frac{\lambda}{2} z^T K z.$$  

- Similarly, at test time we can use the $n$ variables $z$,

$$\hat{X}w = \hat{X} X^T z = \hat{K}z.$$
(pause)
For convex $f$ and $g$ and the primal problem

$$\arg\min_{w \in \mathbb{R}^d} P(w) = f(Xw) + g(w),$$

the Fenchel dual is given by

$$\arg\max_{z \in \mathbb{R}^n} D(z) = -f^*(-z) - g^*(X^Tz),$$

where $f^*$ is the convex conjugate.
**Fenchel Dual**

- For convex $f$ and $g$ and the **primal** problem

\[ \argmin_{w \in \mathbb{R}^d} P(w) = f(Xw) + g(w), \]

the **Fenchel dual** is given by

\[ \argmax_{z \in \mathbb{R}^n} D(z) = -f^*(-z) - g^*(X^Tz), \]

where $f^*$ is the **convex conjugate**.

- **Why are we interested in this?**
  - Dual has fewer variables if $n < d$.
  - $D(z^*) = P(w^*)$ (strong duality): we can solve dual instead of primal.
Fenchel Dual

- For convex \( f \) and \( g \) and the **primal** problem

\[
\arg\min_{w \in \mathbb{R}^d} P(w) = f(Xw) + g(w),
\]

the **Fenchel dual** is given by

\[
\arg\max_{z \in \mathbb{R}^n} D(z) = -f^*(-z) - g^*(X^Tz),
\]

where \( f^* \) is the **convex conjugate**.

- Why are we interested in this?
  - Dual has fewer variables if \( n < d \).
  - \( D(z^*) = P(w^*) \) (**strong duality**): we can solve dual instead of primal.
  - \( D(z) \leq P(w) \) for all \( w \) and \( z \) (**weak duality**): dual gives lower bound on primal.
**Fenchel Dual**

- For convex $f$ and $g$ and the **primal** problem

  $$\arg\min_{w \in \mathbb{R}^d} P(w) = f(Xw) + g(w),$$

  the **Fenchel dual** is given by

  $$\arg\max_{z \in \mathbb{R}^n} D(z) = -f^*(-z) - g^*(X^T z),$$

  where $f^*$ is the **convex conjugate**.

- Why are we interested in this?
  - Dual has fewer variables if $n < d$.
  - $D(z^*) = P(w^*)$ (**strong duality**): we can solve dual instead of primal.
  - $D(z) \leq P(w)$ for all $w$ and $z$ (**weak duality**): dual gives lower bound on primal.
  - If $P$ is strongly-convex, **dual is smooth**: smooth formulation of SVMs.
Fenchel Dual

For convex \( f \) and \( g \) and the \textbf{primal} problem

\[
\arg\min_{w \in \mathbb{R}^d} P(w) = f(Xw) + g(w),
\]

the \textbf{Fenchel dual} is given by

\[
\arg\max_{z \in \mathbb{R}^n} D(z) = -f^*(-z) - g^*(X^Tz),
\]

where \( f^* \) is the \textbf{convex conjugate}.

Why are we interested in this?

- Dual has fewer variables if \( n < d \).
- \( D(z^*) = P(w^*) \) (\textbf{strong duality}): we can solve dual instead of primal.
- \( D(z) \leq P(w) \) for all \( w \) and \( z \) (\textbf{weak duality}): dual gives lower bound on primal.
- If \( P \) is strongly-convex, \textbf{dual is smooth}: smooth formulation of SVMs.
- Dual sometimes allows \textbf{sparse kernel representation}. 
The supremum of a function $f$ is its smallest upper-bound,

$$\sup f(x) = \min_{y\mid y \geq f(x)} y.$$
Supremum and Infimum

- The **supremum** of a function $f$ is its smallest upper-bound,

  $$\sup f(x) = \min_{y \mid y \geq f(x)} y.$$

- Generalization of max that includes limits:

  $$\max_{x \in \mathbb{R}} -x^2 = 0, \quad \sup_{x \in \mathbb{R}} -x^2 = 0,$$

  but

  $$\max_{x \in \mathbb{R}} -e^x = \text{DNE}, \quad \sup_{x \in \mathbb{R}} -e^x = 0.$$

- The analogy for min is called the **infimum**.
The convex conjugate $f^*$ of a function $f$ is given by

$$f^*(y) = \sup_{x \in D} \{ y^T x - f(x) \},$$

where $D$ is values where $\sup$ is finite.

The convex conjugate $f^*$ of a function $f$ is given by

$$f^*(y) = \sup_{x \in \mathcal{D}} \{y^T x - f(x)\},$$

where $\mathcal{D}$ is values where $\sup$ is finite.

It’s the maximum that the linear function $y^T x$ can get above $f(x)$. 

Convex Conjugate

The convex conjugate $f^*$ of a function $f$ is given by

$$f^*(y) = \sup_{x \in D} \{y^T x - f(x)\},$$

where $D$ is values where $\sup$ is finite.

If $f$ is differentiable, then $\sup$ occurs at $x$ where $y = \nabla f(x)$.

Note that $f^*$ is convex even if $f$ is not.

If $f$ is convex (and “closed”), then $f^{**} = f$. 

Convex Conjugate Examples

- If $f(x) = \frac{1}{2}\|x\|^2$ we have
  - $f^*(y) = \sup_x \{y^T x - \frac{1}{2}\|x\|^2\}$ or equivalently (by taking derivative and setting to 0):
    $$0 = y - x,$$
  and pluggin in $x = y$ we get
  $$f^*(y) = y^T y - \frac{1}{2}\|y\|^2 = \frac{1}{2}\|y\|^2.$$

- For other examples, see Boyd & Vandenberghe.
Convex Conjugate Examples

- If \( f(x) = \frac{1}{2} \|x\|^2 \) we have
  \[
  f^*(y) = \sup_x \{ y^T x - \frac{1}{2} \|x\|^2 \}
  \]
  or equivalently (by taking derivative and setting to 0):
  \[
  0 = y - x,
  \]
  and pluggin in \( x = y \) we get
  \[
  f^*(y) = y^T y - \frac{1}{2} \|y\|^2 = \frac{1}{2} \|y\|^2.
  \]

- If \( f(x) = a^T x \) we have
  \[
  f^*(y) = \sup_x \{ y^T x - a^T x \} = \sup_x \{ (y - a)^T x \} = \begin{cases} 0 & y = a \\ \infty & \text{otherwise} \end{cases}
  \]

- For other examples, see Boyd & Vandenberghe.
Fenchel Dual of SVMs

Consider support vector machines,

$$\arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} \max\{0, 1 - y_i w^T x_i\} + \frac{\lambda}{2} \|w\|^2.$$

The Fenchel dual is given by

$$\arg\max_{0 \leq z \leq 1} \sum_{i=1}^{n} z_i - \frac{1}{2\lambda} \|\tilde{X}^T z\|^2,\quad z^T \tilde{X} \tilde{X}^T z,$$

where $\tilde{X} = \text{diag}(y) X$, $w^* = \frac{1}{\lambda} \tilde{X}^T z^*$ and constraints come from $f^* < \infty$. 

A couple magical things have happened:

- We can apply kernel trick.
- Dual is differentiable (though not strongly-convex).
- Dual variables $z$ are sparse (non-zeroes are called “support vectors”): Can give faster training and testing.
- Case where coordinate optimization is efficient.
Consider support vector machines,

\[
\arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} \max\{0, 1 - y_i w^T x_i\} + \frac{\lambda}{2} \|w\|^2.
\]

The Fenchel dual is given by

\[
\arg\max_{0 \leq z \leq 1} \sum_{i=1}^{n} z_i - \frac{1}{2\lambda} \left\| \tilde{X}^T z \right\|^2,
\]

where \( \tilde{X} = \text{diag}(y) X \), \( w^* = \frac{1}{\lambda} \tilde{X}^T z^* \) and constraints come from \( f^* < \infty \).

A couple magical things have happened:

- We can apply kernel trick.
Fenchel Dual of SVMs

- Consider support vector machines,

\[
\arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} \max\{0, 1 - y_i w^T x_i\} + \frac{\lambda}{2} \|w\|^2.
\]

The Fenchel dual is given by

\[
\arg\max_{0 \leq z \leq 1} \sum_{i=1}^{n} z_i - \frac{1}{2\lambda} \underbrace{\|\tilde{X}^T z\|^2}_{z^T \tilde{X} \tilde{X}^T z},
\]

where \(\tilde{X} = \text{diag}(y) X\), \(w^* = \frac{1}{\lambda} \tilde{X}^T z^*\) and constraints come from \(f^* < \infty\).

- A couple magical things have happened:
  - We can apply kernel trick.
  - Dual is differentiable (though not strongly-convex).
Fenchel Dual of SVMs

- Consider support vector machines,

$$\argmin_{w \in \mathbb{R}^d} \sum_{i=1}^n \max\{0, 1 - y_i w^T x_i\} + \frac{\lambda}{2} \|w\|^2.$$ 

The Fenchel dual is given by

$$\argmax_{0 \leq z \leq 1} \sum_{i=1}^n z_i - \frac{1}{2\lambda} \|\tilde{X}^T z\|^2,$$

where $\tilde{X} = \text{diag}(y) X$, $w^* = \frac{1}{\lambda} \tilde{X}^T z^*$ and constraints come from $f^* < \infty$.

- A couple magical things have happened:
  - We can apply kernel trick.
  - Dual is differentiable (though not strongly-convex).
  - Dual variables $z$ are sparse (non-zeroes are called “support vectors”):
    - Can give faster training and testing.
Fenchel Dual of SVMs

Consider support vector machines,

\[
\arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} \max\{0, 1 - y_i w^T x_i\} + \frac{\lambda}{2} \|w\|^2.
\]

The Fenchel dual is given by

\[
\arg\max_{0 \leq z \leq 1} \sum_{i=1}^{n} z_i - \frac{1}{2\lambda} \|\tilde{X}^T z\|^2,
\]

where \(\tilde{X} = \text{diag}(y) X\), \(w^* = \frac{1}{\lambda} \tilde{X}^T z^*\) and constraints come from \(f^* < \infty\).

A couple magical things have happened:
- We can apply kernel trick.
- Dual is differentiable (though not strongly-convex).
- Dual variables \(z\) are sparse (non-zeroes are called “support vectors”):
  - Can give faster training and testing.
- Case where coordinate optimization is efficient.
Stochastic Dual Coordinate Ascent

If we have an L2-regularized linear model with convex $f_i$,

$$\arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} f_i(w^T x_i) + \frac{\lambda}{2} \|w\|^2,$$

then the Fenchel dual is given by

$$\arg\max_{z \in \mathbb{R}^n} - \sum_{i=1}^{n} f_i^*(z_i) - \frac{1}{2\lambda} \left( z^T X^T z + \|X^T z\|^2 - \underbrace{z^T X X^T z}_{\text{separable}} \right).$$

We can apply stochastic dual coordinate ascent (SDCA):

- Only looks at one training example on each iteration.
- Obtains $O(\log(1/\epsilon))$ rate if $\nabla f_i$ are $L$-Lipschitz.
- Performance similar to SAG for many problems, worse if $\mu \gg \lambda$.
- Obtains $O(1/\epsilon)$ rate for non-smooth $f$: Same rate as stochastic subgradient, but we can now use exact/adaptive step-size.

You could add an L2-regularizer to dual, corresponds to smoothing primal.
If we have an L2-regularized linear model with convex $f_i$,

$$\arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^n f_i(w^T x_i) + \frac{\lambda}{2} \|w\|^2,$$

then the Fenchel dual is given by

$$\arg\max_{z \in \mathbb{R}^n} -\sum_{i=1}^n f_i^*(z_i) - \frac{1}{2\lambda} \left( z^T X^T z \right)^2. \quad \text{separable}$$

We can apply stochastic dual coordinate ascent (SDCA):

- Only looks at one training example on each iteration.
- Obtains $O(\log(1/\epsilon))$ rate if $\nabla f_i$ are $L$-Lipschitz.
  - Performance similar to SAG for many problems, worse if $\mu >> \lambda$. 

Performance similar to SAG for many problems, worse if $\mu >> \lambda$. 

You could add an L2-regularizer to dual, corresponds to smoothing primal.
Stochastic Dual Coordinate Ascent

- If we have an L2-regularized linear model with convex $f_i$,

$$\argmin_{w \in \mathbb{R}^d} \sum_{i=1}^{n} f_i(w^T x_i) + \frac{\lambda}{2} \| w \|^2,$$

then the Fenchel dual is given by

$$\argmax_{z \in \mathbb{R}^n} \left( - \sum_{i=1}^{n} f^*_i(z_i) - \frac{1}{2\lambda} \left( z^T X^T X z \right) \right),$$

where $f^*_i(z_i)$ is separable.

- We can apply stochastic dual coordinate ascent (SDCA):
  - Only looks at one training example on each iteration.
  - Obtains $O(\log(1/\epsilon))$ rate if $\nabla f_i$ are $L$-Lipschitz.
    - Performance similar to SAG for many problems, worse if $\mu >> \lambda$.
  - Obtains $O(1/\epsilon)$ rate for non-smooth $f$:
    - Same rate as stochastic subgradient, but we can now use exact/adaptive step-size.
    - You could add an L2-regularizer to dual, corresponds to smoothing primal.
Summary

Kernel Trick: allows working with "similarity" instead of features. Valid kernels are typically constructed from other valid kernels. Representer theorem allows kernel trick for L2-regularized linear models. Fenchel dual re-writes sum of convex functions with convex conjugates: dual may have nice structure: differentiable, sparse, coordinate optimization. Final session: we discuss parallel/distributed methods and non-convex functions.
Summary

- **Kernel trick**: allows working with “similarity” instead of features.
- **Valid kernels** are typically constructed from other valid kernels.
Summary

- **Kernel trick**: allows working with “similarity” instead of features.
- **Valid kernels** are typically constructed from other valid kernels.
- **Representer theorem** allows kernel trick for L2-regularized linear models.

Fenchel dual re-writes sum of convex functions with convex conjugates:
Dual may have nice structure: differentiable, sparse, coordinate optimization.
Final session: we discuss parallel/distributed methods and non-convex functions.
Summary

- **Kernel trick**: allows working with “similarity” instead of features.
- Valid kernels are typically constructed from other valid kernels.
- **Representer theorem** allows kernel trick for L2-regularized linear models.
- **Fenchel dual** re-writes sum of convex functions with convex conjugates:
  - Dual may have nice structure: differentiable, sparse, coordinate optimization.

- Final session: we discuss parallel/distributed methods and non-convex functions.