• Please **mute** yourself and **turn off videos** to save bandwidth.

• If you have questions during the lecture
  • Use chatrooms to post your questions
    • I’ll review chatrooms in batches
    • You can also un-mute yourself and ask the questions directly

• The slides are posted on the course website

• RECORD THE LECTURE!
  • Please remind me if I forget to do so.
Logistics: Homework and Exam 2

• Homework 4 will be due April 13 (Monday)
  • Please start it early
    • It was on average the most time consuming assignment for students in the past
  • Keep track of your own late days
    • Gradescope doesn’t allow separate deadlines
    • Your submissions won’t be graded if you exceed the late-day limit
    • Up to 3 late days can be used if you still have late days left

• Homework 5 is posted on the course website.
  • Due on April 19 (Sunday), **11:30AM**
  • At most two late days can be used in this homework
  • We have covered all topics except for Problem 5 (plan to be covered today)

• Exam 2 will be online on Canvas on April 23 (Thursday).
  • See Slides on April 7 for more details.
Recap
Support Vector Machines

• Goal: Find the \textbf{max-margin} linear separator

• If the data is linearly separable
  • Hard-Margin SVM (Assume data is \textbf{linearly separable})

  \[
  \begin{align*}
  &\text{minimize}_{\mathbf{w}, b} \quad \frac{1}{2} \mathbf{w}^T \mathbf{w} \\
  &\text{subject to} \quad y_n (\mathbf{w}^T \mathbf{x}_n + b) \geq 1, \forall n
  \end{align*}
  \]

  \[g(\mathbf{x}) = \text{sign}(\mathbf{w}_*^T \mathbf{x} + b^*)\]

• If the data is not linearly separable
  • Soft-margin SVM
  • Nonlinear transformation – Dual Formulation and Kernel Tricks
Soft-Margin SVM

• For each point \((\tilde{x}_n, y_n)\), we allow a deviation \(\xi_n \geq 0\)
  • The constraint becomes: \(y_n(\tilde{w}^T \tilde{x}_n + b) \geq 1 - \xi_n\)
  • We add a penalty for each deviation: Total penalty \(C \sum_{n=1}^{N} \xi_n\)

\[
\begin{align*}
\text{minimize}_{\tilde{w}, b, \xi} & \quad \frac{1}{2} \tilde{w}^T \tilde{w} + C \sum_{n=1}^{N} \xi_n \\
\text{subject to} & \quad y_n (\tilde{w}^T \tilde{x}_n + b) \geq 1 - \xi_n, \forall n \\
& \quad \xi_n \geq 0, \forall n
\end{align*}
\]

Remarks:
• \(C\) is a hyper-parameter we can choose, e.g., using validation
  • Larger \(C\) => less tolerable to noise => smaller margin
• Soft-margin SVM is still a Quadratic Program, with efficient solvers
Primal-Dual Formulations of Hard-Margin SVM

• Primal

\[
\begin{align*}
\text{minimize}_{\vec{w},b} & \quad \frac{1}{2} \vec{w}^T \vec{w} \\
\text{subject to} & \quad y_n (\vec{w}^T \vec{x}_n + b) \geq 1, \forall n
\end{align*}
\]

Given optimal \(\vec{\alpha}^*\):

• \(\vec{w}^* = \sum_{n}^{\alpha_n > 0} \alpha_n^* y_n \vec{x}_n\)
• Find a \(\alpha_n^* > 0, \quad b^* = y_n - \vec{x}_n^T \vec{w}^*\)

• Dual

\[
\begin{align*}
\text{maximize}_{\vec{\alpha}} & \quad \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m \vec{x}_n^T \vec{x}_m \\
\text{subject to} & \quad \sum_{n=1}^{N} \alpha_n y_n = 0 \\
& \quad \alpha_n \geq 0, \forall n
\end{align*}
\]

• Both can be efficiently solved using QP solver.
• We can infer the solution from one to the other
Kernel Functions

• Define kernel function $K_{\Phi}(\vec{x}, \vec{x}') = \Phi(\vec{x})^T \Phi(\vec{x}')$
  • The similarity of two vectors in the projected space

• Goal: Compute $K_{\Phi}(\vec{x}, \vec{x}')$ without transforming $\vec{x}$ and $\vec{x}'$

• Why? This enables us to operate in higher dimensional spaces without really worrying about the computational overhead.
Kernel Trick: Utilize Dual and Kernel Functions

• The dual with nonlinear transform

\[
\text{maximize} \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m \tilde{z}_n^T \tilde{z}_m \\
\text{subject to} \quad \sum_{n=1}^{N} \alpha_n y_n = 0 \\
\alpha_n \geq 0, \forall n
\]

• Plug in the kernel function \( K_\Phi(\tilde{x}, \tilde{x}') = \Phi(\tilde{x})^T \Phi(\tilde{x}') \)

\[
\text{maximize} \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m K_\Phi(\tilde{x}_n, \tilde{x}_m) \\
\text{subject to} \quad \sum_{n=1}^{N} \alpha_n y_n = 0 \\
\alpha_n \geq 0, \forall n
\]

• If the kernel can be computed efficiently, we can solve \( \tilde{\alpha}^* \) efficiently.
• With kernel tricks, we can avoid the dependency on the dimension of \( \tilde{z} \)
Recover \((\hat{w}^*, b^*)\) from \(\hat{\alpha}^*\) with Kernel Tricks

• Note that \(\hat{\alpha}^*\) is solved in the \(\hat{z}\) space
  • \(\hat{w}^* = \sum_{\alpha_n > 0} \alpha_n y_n \Phi(\hat{x}_n)\)
  • Find a \(\alpha_n^* > 0, b^* = y_n - \hat{w}^* \Phi(\hat{x}_n)\)
  • We want to avoid the transformation!

• Let’s look at the hypothesis
  • \(g(\hat{x}) = \text{sign}(\hat{w}^* \Phi(\hat{x}) + b^*)\)

\[
\hat{w}^* \Phi(\hat{x}) = \left(\sum_{\alpha_n > 0} \alpha_n^* y_n \Phi(\hat{x}_n)\right) \Phi(\hat{x})
= \sum_{\alpha_n > 0} \alpha_n^* y_n \Phi(\hat{x}_n) \Phi(\hat{x})
= \sum_{\alpha_n > 0} \alpha_n^* y_n K(\hat{x}_n, \hat{x})
\]

\[
b^* = y_n - \hat{w}^* \Phi(\hat{x}_n)
= y_n - \left(\sum_{\alpha_m > 0} \alpha_m^* y_m \Phi(\hat{x}_m)\right) \Phi(\hat{x}_n)
= y_n - \sum_{\alpha_m > 0} \alpha_m^* y_m K(\hat{x}_m, \hat{x}_n)
\]

• Still can be computed in the \(\hat{x}\) space!
The notes are not intended to be comprehensive. They should be accompanied by lectures and/or textbook. Let me know if you spot errors.
Kernel Functions
Polynomial Kernel

• Example in last lecture: 2\textsuperscript{nd} order polynomial for 2-d $\vec{x}$
  
  \begin{itemize}
  \item $\vec{x} = (x_1, x_2)$
  \item $\vec{z} = \Phi_2(\vec{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2} x_1 x_2, x_1^2, x_2^2)$
  \item $\vec{z}^T \vec{z}' = 1 + 2x_1x'_1 + 2x_2x'_2 + 2x_1x'_1x_2x'_2 + (x_1x'_1)^2 + (x_2x'_2)^2$
  \end{itemize}

\begin{align*}
&= (1 + x_1x'_1 + x_2x'_2)^2 \\
&= (1 + \vec{x}^T \vec{x}')^2
\end{align*}

• General 2\textsuperscript{nd} order polynomial
  
  \begin{itemize}
  \item $\vec{x} = (x_1, x_2, ..., x_d)$
  \item Polynomial kernel $K_\Phi(\vec{x}, \vec{x}') = (1 + \vec{x}^T \vec{x}')^2$
  \end{itemize}

\begin{align*}
&= (1 + x_1x'_1 + x_2x'_2 + \cdots + x_dx'_d)^2
\end{align*}
Polynomial Kernel

• $\tilde{x} = (x_1, x_2, ..., x_d)$

• 2\textsuperscript{nd} order polynomial kernel $K_{\Phi_2}(\tilde{x}, \tilde{x}') = (1 + \tilde{x}^T \tilde{x}')^2$

• Q-th order Polynomial kernel $K_{\Phi_Q}(\tilde{x}, \tilde{x}') = (1 + \tilde{x}^T \tilde{x}')^Q = (1 + x_1x'_1 + \cdots + x_dx'_d)^Q$

• Computational complexity
  • Dimension of $\Phi_Q(\tilde{x})$: \(\binom{Q+d}{Q}\)
  • Direct computation of $\Phi_Q(\tilde{x})^T \Phi_Q(\tilde{x}')$: $O\left(\binom{Q+d}{Q}\right)$
  • Computation through kernel $K_{\Phi_Q}(\tilde{x}, \tilde{x}')$: $O(d)$

General form of polynomial kernel: 

$$K(\tilde{x}, \tilde{x}') = (a \tilde{x}^T \tilde{x}' + b)^Q$$
We Only Need $\tilde{Z}$ Space to Exist

• In discussion of polynomial kernels,
  • We have a target transformation in mind
  • Aim to want to find a corresponding kernel function

• In fact, as long as $K(\tilde{x}, \tilde{x}')$ is an inner product in some $\tilde{Z}$ space, we are good
  • Just plug in the kernel in the dual formulation
  • We obtain a linear separator in the corresponding $\tilde{Z}$ space

\[
\begin{align*}
\text{maximize} & \quad \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m K(\tilde{x}_n, \tilde{x}_m) \\
\text{subject to} & \quad \sum_{n=1}^{N} \alpha_n y_n = 0 \\
& \quad \alpha_n \geq 0, \forall n
\end{align*}
\]
Gaussian RBF Kernel

• $K(\tilde{x}, \tilde{x}') = e^{-\gamma \|\tilde{x} - \tilde{x}'\|^2}$

• What’s the corresponding $\tilde{z}$ space?
  • For simplicity, make $\gamma = 1$ and $\tilde{x} = x$ be 1 dimensional
    • $K(\tilde{x}, \tilde{x}') = e^{-(x-x')^2}$
      = $e^{-x^2+2xx'-x'^2}$
      = $e^{-x^2} e^{-x'^2} e^{2xx'}$
      = $e^{-x^2} e^{-x'^2} \sum_{k=0}^{\infty} \frac{(2xx')^k}{k!}$
      = $\sum_{k=0}^{\infty} e^{-x^2} \sqrt{\frac{2^k}{k!}} x^k e^{-x'^2} \sqrt{\frac{2^k}{k!}} x'^k$

  Taylor expansion: $e^{2xx'} = \sum_{k=0}^{\infty} \frac{(2xx')^k}{k!}$

• The corresponding $\Phi(x) = e^{-x^2} \left(1, \sqrt{\frac{2}{1}} x, \sqrt{\frac{2^2}{2!}} x^2, \ldots\right)$
Gaussian RBF Kernel

• $K(\vec{x}, \vec{x}') = e^{-γ\|\vec{x} - \vec{x}'\|^2}$

• The corresponding transform in 1-dim input $\vec{x} = x$
  
  • $\Phi(x) = e^{-x^2} \left( 1, \sqrt{\frac{2}{1}} x, \sqrt{\frac{2^2}{2!}} x^2, \ldots \right)$

• $K(\vec{x}, \vec{x}')$ is the inner product of two vectors in an infinite dimensional space!

• When we plug in $K(\vec{x}, \vec{x}')$ in dual SVM
  
  • We are finding the max-margin separator in an infinite dimensional space
  
  • Seems to introduce infinite generalization error?
    
    • Maximizing margin help mitigate this issue
    
    • The number of support vectors provides indicators on the generalization
Design Your Own Kernel? [Safe to Skip]

• Say we design a kernel function, how do we know whether it is valid, i.e., whether there is a corresponding $\mathcal{Z}$ space?

• Mercer’s condition (See discussion in LFD 8.3.2)
  • Kernel matrix

\[
\begin{bmatrix}
K(x_1, x_1) & K(x_1, x_2) & \cdots & K(x_1, x_N) \\
K(x_2, x_1) & K(x_2, x_2) & \cdots & K(x_2, x_N) \\
\vdots & \vdots & \ddots & \vdots \\
K(x_N, x_1) & K(x_N, x_2) & \cdots & K(x_N, x_N)
\end{bmatrix}
\]

• $K(\hat{x}, \hat{x}')$ is a valid kernel if and only if the kernel matrix is always symmetric positive semi-definite for any $\hat{x}_1, \ldots, \hat{x}_N$
Summary of What We Talked About So Far

**Hard-Margin SVM (Separable Data)**

\[
\text{minimize}_{\tilde{w}, b} \quad \frac{1}{2} \tilde{w}^T \tilde{w}
\]

subject to
\[
y_n (\tilde{w}^T \tilde{x}_n + b) \geq 1, \forall n
\]

**Soft-Margin SVM (Tolerate Noise)**

\[
\text{minimize}_{\tilde{w}, b, \xi} \quad \frac{1}{2} \tilde{w}^T \tilde{w} + C \sum_{n=1}^{N} \xi_n
\]

subject to
\[
y_n (\tilde{w}^T \tilde{x}_n + b) \geq 1 - \xi_n, \forall n
\]
\[
\xi_n \geq 0, \forall n
\]

**Kernel Formulation of Hard-Margin SVM**

\[
\text{maximize}_{\alpha} \quad \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m K_\Phi(\tilde{x}_n, \tilde{x}_m)
\]

subject to
\[
\sum_{n=1}^{N} \alpha_n y_n = 0
\]
\[
\alpha_n \geq 0, \forall n
\]
Kernel Version of Soft-Margin SVM

• Soft-Margin SVM

\[
\begin{align*}
&\text{minimize}_{\vec{w}, b, \xi} \quad \frac{1}{2} \vec{w}^T \vec{w} + C \sum_{n=1}^{N} \xi_n \\
&\text{subject to } \quad y_n (\vec{w}^T \vec{x}_n + b) \geq 1 - \xi_n, \forall n \\
&\quad \xi_n \geq 0, \forall n
\end{align*}
\]

• Kernel Version of Soft-Margin SVM

\[
\begin{align*}
&\text{maximize}_\alpha \quad \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m K(\vec{x}_n, \vec{x}_m) \\
&\text{subject to } \quad \sum_{n=1}^{N} \alpha_n y_n = 0 \\
&\quad 0 \leq \alpha_n \leq C, \forall n
\end{align*}
\]

• It can be obtained by similar procedure as hard-margin version
• We can obtain the same relationship between $\tilde{\alpha}^*$ and $(\tilde{\vec{w}}^*, b^*)$
Interpretation of Support Vectors

• \( \alpha_n^* > 0 \Rightarrow (\vec{x}_n, y_n) \) is a support vector
  • \( y_n(\vec{w}^* T \vec{x}_n + b^*) = 1 - \xi_n \)

• Utilizing complementary slackness
  • When \( 0 < \alpha_n^* < C \)
    • \( \xi_n = 0 \)
    • \( y_n(\vec{w}^* T \vec{x}_n + b^*) = 1 \)
    • \( (\vec{x}_n, y_n) \) is a “margin” support vector
  
  • When \( \alpha_n^* = C \)
    • \( \xi_n > 0 \)
    • \( y_n(\vec{w}^* T \vec{x}_n + b^*) < 1 \)
    • \( (\vec{x}_n, y_n) \) is a “non-margin” support vector
Short Break and Q&A
Neural Networks
Perceptron

• What is a hypothesis in Perceptron

\[ h(\vec{x}) = \text{sign}(\vec{w}^T \vec{x}) \]

• Note that we have reverted back to our original notations

\begin{itemize}
  \item \( \vec{x} = (x_0, x_1, \ldots, x_d) \)
  \item \( \vec{w} = (w_0, w_1, \ldots, w_d) \)
  \item Linear separator
  \[ h(\vec{x}) = \text{sign}(\vec{w}^T \vec{x}) \]
\end{itemize}
Perceptron

• What is a hypothesis in Perceptron

\[ h(\hat{x}) = \text{sign}(\vec{w}^T \hat{x}) \]

• Graphical representation of Perceptron

Inspired by neurons:
The output signal is triggered when the weighted combination of the inputs is larger than some threshold.
The First Perceptron Machine

Mark I Perceptron machine, the first implementation of the perceptron algorithm. (From Wikipedia)

“the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.”
Implement Logic Gates with Perceptron

- **AND**(\(x_1, x_2\))
  - Use +1 to denote “true” and −1 to denote “false”

<table>
<thead>
<tr>
<th>(x_1)</th>
<th>(x_2)</th>
<th>AND((x_1, x_2))</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>+1</td>
<td>+1</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>+1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Implement Logic Gates with Perceptron

- **AND\((x_1, x_2)\)**
  - Use +1 to denote “true” and −1 to denote “false”

<table>
<thead>
<tr>
<th>(x_1)</th>
<th>(x_2)</th>
<th>AND((x_1, x_2))</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>+1</td>
<td>+1</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>+1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Implement Logic Gates with Perceptron

- \( \text{OR}(x_1, x_2) \)
  - Use +1 to denote “true” and −1 to denote “false”

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( \text{OR}(x_1, x_2) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>+1</td>
<td>+1</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>+1</td>
</tr>
<tr>
<td>-1</td>
<td>+1</td>
<td>+1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Implement Logic Gates with Perceptron

- OR($x_1, x_2$)
  - Use $+1$ to denote “true” and $-1$ to denote “false”

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>OR($x_1, x_2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+1$</td>
<td>$+1$</td>
<td>$+1$</td>
</tr>
<tr>
<td>$+1$</td>
<td>$-1$</td>
<td>$+1$</td>
</tr>
<tr>
<td>$-1$</td>
<td>$+1$</td>
<td>$+1$</td>
</tr>
<tr>
<td>$-1$</td>
<td>$-1$</td>
<td>$-1$</td>
</tr>
</tbody>
</table>
Implement Logic Gates with Perceptron

• NOT($x_1$)
  • Use $+1$ to denote “true” and $-1$ to denote “false”

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>OR($x$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>+1</td>
</tr>
</tbody>
</table>
Implement Logic Gates with Perceptron

• NOT($x_1$)
  • Use +1 to denote “true” and −1 to denote “false”

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>OR($x$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>+1</td>
</tr>
</tbody>
</table>
Implement Logic Gates with Perceptron

• \( \text{XOR}(x_1, x_2) \)
  • Use \(+1\) to denote “true” and \(-1\) to denote “false”

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( \text{XOR}(x_1, x_2) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>+1</td>
<td>-1</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>+1</td>
</tr>
<tr>
<td>-1</td>
<td>+1</td>
<td>+1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Implement Logic Gates with Perceptron

• **XOR**($x_1, x_2$)
  
  • Use +1 to denote “true” and −1 to denote “false”

| $x_1$ | $x_2$ | XOR($x_1, x_2$) |
|-------|-------|-----------------
| +1    | +1    | -1              |
| +1    | -1    | +1              |
| -1    | +1    | +1              |
| -1    | -1    | -1              |

It is **impossible** to implement XOR using a single perceptron (draw the points in the 2-D space, you will see they are not linearly separable)

Stronger version:
It is **impossible** to implement XOR using a single layer of perceptrons
Multi-Layer Perceptron
Representing Boolean Operations

• \( \text{AND}(x_1, x_2) \rightarrow x_1 x_2 \)
• \( \text{OR}(x_1, x_2) \rightarrow x_1 + x_2 \)
• \( \text{NOT}(x_1) \rightarrow \bar{x}_1 \)
• \( \text{XOR}(x_1, x_2) \rightarrow x_1 \bar{x}_2 + \bar{x}_1 x_2 \)
Implementing XOR

\[ \text{XOR}(x_1, x_2) \rightarrow x_1 \bar{x}_2 + \bar{x}_1 x_2 \]
Implementing XOR

- \( \text{XOR}(x_1, x_2) \rightarrow x_1 \bar{x}_2 + \bar{x}_1 x_2 \)
Multi-Layer Perceptron (MLP)

Input layer | Hidden layer(s) | Output layer

Feed-forward network

\[
\begin{align*}
&1 \\
&x_1 \\
&x_2 \\
\end{align*}
\]

\[
\begin{align*}
&-1.5 \\
&1 \\
&-1 \\
&-1 \\
\end{align*}
\]

\[
\begin{align*}
&1.5 \\
&1 \\
&1 \\
\end{align*}
\]

\[
\text{XOR}(x_1, x_2)
\]
The Power of Multi-Layer Perceptron (MLP)

• We now know that we can implement XOR by introducing the hidden layer in MLP. But generally how powerful is MLP?

• Universal approximation theorem
  • a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbb{R}^n$, under mild assumptions on the activation function.

• Three-layer MLP can approximate ANY continuous target function!
Informal Intuitions of Universal Approximation

• A continuous separator can "decomposed" into linear separators
How to Learn MLP From Data?

• Given $D$ and the network structure, how to learn the “weights” (i.e., the weight vectors of every Perceptron)?

• Computationally challenging due to the ”sign” function
Neural Networks

• A softened version of multi-layer Perceptron (MLP)

$\theta$: activation function
(Specify the "activation" of the neuron)

Next lecture: formally introduce neural networks and how to learn it from data