Learning From Data
Lecture 20
Multilayer Perceptron

Multiple layers
Universal Approximation
The Neural Network

M. Magdon-Ismail
CSCI 4100/6100
RECAP: Unsupervised Learning

$k$-Means Clustering

- ‘Hard’ partition into $k$-clusters

Gaussian Mixture Model

- ‘Soft’ probability density estimation
A brief history of Neural Networks.

<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1943</td>
<td>McCulloch &amp; Pitt (Neural Nets)</td>
</tr>
<tr>
<td>1958</td>
<td>Rosenblatt (Perceptron)</td>
</tr>
<tr>
<td>1969</td>
<td>Minsky &amp; Pappert XOR, fitting</td>
</tr>
<tr>
<td>1982</td>
<td>Hopfield Network</td>
</tr>
<tr>
<td>1986</td>
<td>Hinton et. al (MLP, Backprop.)</td>
</tr>
<tr>
<td>1992-1995</td>
<td>Vapnik (SVM)</td>
</tr>
<tr>
<td>2006</td>
<td>Hinton et. al (Deep Belief Nets)</td>
</tr>
<tr>
<td>2012</td>
<td>Hinton et. al (DeepNets, Vision)</td>
</tr>
</tbody>
</table>

[Krizhevsky, Sutskever and Hinton, 2012]
Planes Don’t Flap Wings to Fly

Engineering success may start with biological inspiration, but then take a totally different path.
**XOR: A Limitation of the Linear Model**

A diagram illustrating the XOR function. The figure shows a two-dimensional space with two axes, $x_1$ and $x_2$, and regions marked with +1 and -1. The linear boundary between these regions demonstrates the limitation of linear models in capturing the XOR function, which requires non-linear decision boundaries.
Decomposing XOR

\[ f = h_1 h_2 + \overline{h_1} h_2 \]

\[ h_1(x) = \text{sign}(w_1^T x) \]

\[ h_2(x) = \text{sign}(w_2^T x) \]
Perceptrons for **OR** and **AND**

\[
\text{OR}(x_1, x_2) = \text{sign}(x_1 + x_2 + 1.5) \\
\text{AND}(x_1, x_2) = \text{sign}(x_1 + x_2 - 1.5)
\]
Representing $f$ Using OR and AND

$$f = h_1 \overline{h_2} + \overline{h_1} h_2$$
Representing $f$ Using OR and AND

$$f = h_1 \overline{h_2} + \overline{h_1} h_2$$
Representing $f$ Using **OR** and **AND**

\[ f = h_1 \overline{h_2} + \overline{h_1} h_2 \]
The Multilayer Perceptron (MLP)

More layers allow us to implement $f$

These additional layers are called *hidden layers*
Any target function $f$ that can be decomposed into linear separators can be implemented by a 3-layer MLP.
Universal Approximation

A sufficiently smooth separator can “essentially” be decomposed into linear separators.

Target

8 perceptrons

16 perceptrons
Approximation Versus Generalization

The size of the MLP controls the approximation-generalization tradeoff.

More nodes per hidden layer \(\implies\) approximation\(\uparrow\) and generalization\(\downarrow\).
Minimizing $E_{in}$

A combinatorial problem even harder with the MLP than the Perceptron.

$E_{in}$ is not smooth (due to sign function), so cannot use gradient descent.

$\text{sign}(x) \approx \tan(x) \rightarrow$ gradient descent to minimize $E_{in}$. 
The Neural Network

input layer $\ell = 0$  
hidden layers $0 < \ell < L$  
output layer $\ell = L$
Zooming into a Hidden Node

layers $\ell = 0, 1, 2, \ldots, L$
layer $\ell$ has “dimension” $d^{(\ell)} \implies d^{(\ell)} + 1$ nodes

$W^{(\ell)} = \begin{bmatrix} w_{1}^{(\ell)} & w_{2}^{(\ell)} & \cdots & w_{d^{(\ell)}}^{(\ell)} \end{bmatrix}$

<table>
<thead>
<tr>
<th>parameters</th>
<th>signals in</th>
<th>outputs</th>
<th>weights in</th>
<th>weights out</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer $\ell$</td>
<td>$s^{(\ell)}$</td>
<td>$x^{(\ell)}$</td>
<td>$W^{(\ell)}$</td>
<td>$W^{(\ell+1)}$</td>
</tr>
<tr>
<td></td>
<td>$d^{(\ell)}$ dimensional input vector</td>
<td>$d^{(\ell)} + 1$ dimensional output vector</td>
<td>$(d^{(\ell-1)} + 1) \times d^{(\ell)}$ dimensional matrix</td>
<td>$(d^{(\ell)} + 1) \times d^{(\ell+1)}$ dimensional matrix</td>
</tr>
</tbody>
</table>
The Neural Network

input layer $\ell = 0$

hidden layers $0 < \ell < L$

output layer $\ell = L$

Multilayer Perceptron: 18 / 18