Artificial Neural Networks
(Feedforward Nets)
Single Perceptron Unit

\[ x_0 = 1, \quad x_1, \quad x_2, \quad x_3, \quad \ldots, \quad x_n \]

\[ w_0, \quad w_1, \quad w_2, \quad w_3, \quad w_n \]
Linear Classifier
Single Perceptron Unit

\[ h(x) = \theta(w \cdot x + b) = \theta(\overline{w} \cdot \overline{x}) \]

\[ \theta(z) = \begin{cases} 
1 & z \geq 0 \\
0 & \text{else} 
\end{cases} \]
Beyond Linear Separability

Not linearly separable
Beyond Linear Separability

Not linearly separable
Beyond Linear Separability

Not linearly separable
Multi-Layer Perceptron

- More powerful than single layer.
- Lower layers transform the input problem into more tractable (linearly separable) problems for subsequent layers.
XOR Problem

Not linearly separable

\[
\begin{align*}
    w_{01} &= 3/2 & w_{11} &= w_{12} = 1 \\
w_{02} &= 1/2 & w_{21} &= w_{22} = 1
\end{align*}
\]

<table>
<thead>
<tr>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(o_1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
XOR Problem

Not linearly separable

\[
\begin{array}{c}
w_{01} = 3/2 \\
w_{02} = 1/2
\end{array}
\quad
\begin{array}{c}
w_{11} = w_{12} = 1 \\
w_{21} = w_{22} = 1
\end{array}
\]

<table>
<thead>
<tr>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(o_1)</th>
<th>(o_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
XOR Problem

Not linearly separable

\[
\begin{align*}
\text{Not linearly separable} & \\
w_{01} &= \frac{3}{2} & w_{11} &= w_{12} = 1 \\
w_{02} &= \frac{1}{2} & w_{21} &= w_{22} = 1 \\
w_{03} &= \frac{1}{2} & w_{31} &= -1, w_{32} = 1
\end{align*}
\]

\[
\begin{array}{c|c|c|c|c}
\text{x}_1 & \text{x}_2 & \text{o}_1 & \text{o}_2 & \text{y} \\
\hline
0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 0
\end{array}
\]

Linearly separable
Multi-Layer Perceptron Learning

• Any set of training points can be separated by a three-layer perceptron network.
• “Almost any” set of points separable by two-layer perceptron network.
• But, no efficient learning rule is known.
Multi-Layer Perceptron Learning

• Any set of training points can be separated by a three-layer perceptron network.
• “Almost any” set of points separable by two-layer perceptron network.
• But, no efficient learning rule is known.

Two “hidden” layers and one output layer

May need an exponential number of units.

One “hidden” layer and one output layer
Multi-Layer Perceptron Learning

• Any set of training points can be separated by a three-layer perceptron network.
• “Almost any” set of points separable by two-layer perceptron network.
• But, no efficient learning rule is known.

• Could we use gradient ascent/descent?
• We would need smoothness: small change in weights produces small change in output.
• Threshold function is not smooth.
Multi-Layer Perceptron Learning

• Any set of training points can be separated by a three-layer perceptron network.
• “Almost any” set of points separable by two-layer perceptron network.
• But, no efficient learning rule is known.

• Could we use gradient ascent/descent?
• We would need smoothness: small change in weights produces small change in output.
• Threshold function is not smooth.

• Use a smooth threshold function!
Sigmoid Unit

\[ z = \sum_{i}^{n} w_i x_i \quad s(z) = \frac{1}{1 + e^{-z}} \]
Sigmoid Unit

\[ y = \frac{1}{1 + e^{-x}} \]
Training

\[ y(x, w) \]

\( w \) is a vector of weights

\( x \) is a vector of inputs

\[
y = s(w_{13}s(w_{11}x_1 + w_{21}x_2 - w_{01}) + w_{23}s(w_{12}x_1 + w_{22}x_2 - w_{02}) - w_{03})
\]
Training

\( y(x, w) \)

\( w \) is a vector of weights
\( x \) is a vector of inputs
\( y^i \) is desired output:

Error over the training set for a given weight vector:

\[
E = \frac{1}{2} \sum_i (y(x^i, w) - y^i)^2
\]

Our goal is to find weight vector that minimizes error

\[
y = s(w_{13} s(w_{11} x_1 + w_{21} x_2 - w_{01}) + w_{23} s(w_{12} x_1 + w_{22} x_2 - w_{02}) - w_{03})
\]
Gradient Descent

$$E = \frac{1}{2} \sum_i (y(x^i, w) - y^i)^2$$

$$\nabla_w E = \sum_i (y(x^i, w) - y^i) \nabla_w y(x^i, w)$$

$$\nabla_w y = \left[ \frac{\partial y}{\partial w_1}, \ldots, \frac{\partial y}{\partial w_n} \right]$$

$$w \leftarrow w - \eta \nabla_w E$$
Training Neural Nets
without overfitting, hopefully...

Given: Data set, desired outputs and a neural net with m weights. Find a setting for the weights that will give good predictive performance on new data. Estimate expected performance on new data.

1. Split data set (randomly) into three subsets:
   - Training set – used for picking weights
   - Validation set – used to stop training
   - Test set – used to evaluate performance

2. Pick random, small weights as initial values

3. Perform iterative minimization of error over training set.

4. Stop when error on validation set reaches a minimum (to avoid overfitting).

5. Repeat training (from step 2) several times (avoid local minima)

6. Use best weights to compute error on test set, which is estimate of performance on new data. Do not repeat training to improve this.

Can use cross-validation if data set is too small to divide into three subsets.
Training vs. Test Error

Training vs Test Error (2 hidden units)
Training vs. Test Error

Training vs Test Error (2 hidden units)

Training vs Test Error (10 hidden units)
Overfitting is not unique to neural nets...

1-Nearest Neighbors

Decision Trees
Overfitting in SVM

Radial Kernel $\sigma=0.1$

Radial Kernel $\sigma=1$
On-line vs off-line

There are two approaches to performing the error minimization:

- **On-line training** – present $x^i$ and $y^{i*}$ (chosen randomly from the training set). Change the weights to reduce the error on this instance. Repeat.

- **Off-line training** – change weights to reduce the total error on training set (sum over all instances).

On-line training is an approximation to gradient descent since the gradient based on one instance is “noisy” relative to the full gradient (based on all instances). This can be beneficial in pushing the system out of shallow local minima.
Feature Selection

- In many machine learning applications, there are huge numbers of features
  - text classification (\# words)
  - gene arrays (5,000 – 50,000)
  - images (512 x 512 pixels)
Feature Selection

- In many machine learning applications, there are huge numbers of features
  - text classification (# words)
  - gene arrays (5,000 – 50,000)
  - images (512 x 512 pixels)
- Too many features
  - make algorithms run slowly
  - risk overfitting
Feature Selection

- In many machine learning applications, there are huge numbers of features
  - text classification (# words)
  - gene arrays (5,000 – 50,000)
  - images (512 x 512 pixels)
- Too many features
  - make algorithms run slowly
  - risk overfitting
- Find a smaller feature space
  - subset of existing features
  - new features constructed from old ones
Feature Ranking

• For each feature, compute a measure of its relevance to the output
• Choose the k features with the highest rankings
• Correlation between feature j and output

\[
R(j) = \frac{\sum (x_j^i - \bar{x}_j)(y^i - \bar{y})}{\sqrt{\sum (x_j^i - \bar{x}_j)^2 \sum (y^i - \bar{y})^2}}
\]

\[
\bar{x}_j = \frac{1}{n} \sum x_j^i \quad \bar{y} = \frac{1}{n} \sum y^i
\]

• Correlation measures how much x tends to deviate from its mean on the same examples on which y deviates from its mean
Correlations in Heart Data

<table>
<thead>
<tr>
<th>Feature</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>thal=1</td>
<td>-0.52</td>
</tr>
<tr>
<td>cp=4</td>
<td>0.51</td>
</tr>
<tr>
<td>thal=3</td>
<td>0.48</td>
</tr>
<tr>
<td>ca=0</td>
<td>-0.48</td>
</tr>
<tr>
<td>oldpeak</td>
<td>0.42</td>
</tr>
<tr>
<td>thalach</td>
<td>-0.42</td>
</tr>
<tr>
<td>exang</td>
<td>0.42</td>
</tr>
<tr>
<td>slope=1</td>
<td>-0.38</td>
</tr>
<tr>
<td>slope=2</td>
<td>0.35</td>
</tr>
<tr>
<td>cp=3</td>
<td>-0.31</td>
</tr>
<tr>
<td>sex</td>
<td>0.28</td>
</tr>
<tr>
<td>ca=2</td>
<td>0.27</td>
</tr>
<tr>
<td>cp=2</td>
<td>-0.25</td>
</tr>
<tr>
<td>ca=1</td>
<td>0.23</td>
</tr>
<tr>
<td>age</td>
<td>0.23</td>
</tr>
</tbody>
</table>

...
Correlations in MPG > 22 data

cyl=4  0.82
displacement -0.77
weight -0.77
horsepower -0.67
cyl=8  -0.58
origin=1  -0.54
model-year  0.44
origin=3  0.40
cyl=6  -0.37
acceleration  0.35
origin=2  0.26

displacement > 189.5

weight > 2224.5

year > 78.5

weight > 2775
XOR Bites Back

- As usual, functions with XOR in them will cause us trouble

  - Each feature will, individually, have a correlation of 0 (it occurs positively as much as negatively for positive outputs)

- To solve XOR, we need to look at groups of features together
Subset Selection

• Consider subsets of variables
  • too hard to consider all possible subsets
  • wrapper methods: use training set or cross-validation error to measure the goodness of using different feature subsets with your classifier
  • greedily construct a good subset by adding or subtracting features one by one
Forward Selection

Given a particular classifier you want to use

\[ F = \{ \} \]

For each \( f_j \)

Train classifier with inputs \( F + \{f_j\} \)

Add \( f_j \) that results in lowest-error classifier to \( F \)

Continue until \( F \) is the right size, or error has quit decreasing
Forward Selection

Given a particular classifier you want to use

\[ F = \{\} \]

For each \( f_j \)

Train classifier with inputs \( F + \{f_j\} \)

Add \( f_j \) that results in lowest-error classifier to \( F \)

Continue until \( F \) is the right size, or error has quit decreasing

• Decision trees, by themselves, do something similar to this
Forward Selection

Given a particular classifier you want to use

\[ F = \{ \} \]

For each \( f_j \)

Train classifier with inputs \( F + \{f_j\} \)

Add \( f_j \) that results in lowest-error classifier to \( F \)

Continue until \( F \) is the right size, or error has quit decreasing

- Decision trees, by themselves, do something similar to this
- Trouble with XOR
Backward Elimination

Given a particular classifier you want to use

\( F = \text{all features} \)

For each \( f_j \)

Train classifier with inputs \( F - \{f_j\} \)

Remove \( f_j \) that results in lowest-error classifier from \( F \)

Continue until \( F \) is the right size, or error increases too much
Forward Selection on Auto Data

cross-validation accuracy

Forward Selection - Auto

10-way X-val Accuracy

number of features added

Series1
Backward Elimination on Auto Data

The chart shows the 10-way cross-validation accuracy for different numbers of features eliminated in the backward selection process. The accuracy values range from approximately 0.85 to 0.96. As more features are eliminated, the accuracy decreases, indicating a trade-off between model complexity and performance.
Forward Selection on Heart Data

- Cross-validation accuracy

**Forward Selection - Hear**

Number of features added vs. 10-way X-val Accuracy

- Series 1

Number of features added vs. cross-validation accuracy

1. 0.72 2. 0.74 3. 0.76 4. 0.78 5. 0.8 6. 0.82 7. 0.84 8. 0.86 9. 1.0 10. 1.2 11. 1.4 12. 1.6 13. 1.8 14. 2.0 15. 2.2 16. 2.4 17. 2.6 18. 2.8 19. 3.0 20. 3.2 21. 3.4 22. 3.6 23. 3.8 24. 4.0 25. 4.2
Backward Elimination on Heart Data

Cross-validation accuracy

Number of features eliminated
Recursive Feature Elimination

Train a linear SVM or neural network
Remove the feature with the smallest weight
Repeat

- More efficient than regular backward elimination
- Requires only one training phase per feature
Clustering

- Form clusters of inputs
- Map the clusters into outputs
- Given a new example, find its cluster, and generate the associated output
Clustering

- Form clusters of inputs
- Map the clusters into outputs
- Given a new example, find its cluster, and generate the associated output
Clustering

- Form clusters of inputs
- Map the clusters into outputs
- Given a new example, find its cluster, and generate the associated output
Clustering Criteria

- small distances between points within a cluster
- large distances between clusters

- Need a distance measure, as in nearest neighbor
K-Means Clustering

• Tries to minimize

\[ \sum_{j=1}^{k} \sum_{i \in S_j} \left\| x^i - \mu_j \right\|^2 \]

- \# of clusters
- elements of cluster j
- mean of elts in cluster j
- squared dist from point to mean

• Only gets, greedily, to a local optimum
K-means Algorithm

Choose $k$
Randomly choose $k$ points $C_j$ to be cluster centers
K-means Algorithm

Choose $k$

Randomly choose $k$ points $C_j$ to be cluster centers

Loop

Partition the data into $k$ classes $S_j$ according to which of the $C_j$ they’re closest to

For each $S_j$, compute the mean of its elements and let that be the new cluster center
K-means Algorithm

Choose k
Randomly choose k points Cj to be cluster centers
Loop
  Partition the data into k classes Sj according to which of the Cj they’re closest to
  For each Sj, compute the mean of its elements and let that be the new cluster center
  Stop when centers quit moving

• Guaranteed to terminate
K-means Algorithm

Choose k
Randomly choose k points Cj to be cluster centers
Loop
  Partition the data into k classes Sj according to which of the Cj they’re closest to
  For each Sj, compute the mean of its elements and let that be the new cluster center
Stop when centers quit moving

● Guaranteed to terminate
● If a cluster becomes empty, re-initialize the center
K-Means Example
K-Means Example
K-Means Example
K-Means Example
K-Means Example
K-Means Example
K-Means Example
K-Means Example
K-Means Example
Principal Components Analysis

• Given an n-dimensional real-valued space, data are often nearly restricted to a lower-dimensional subspace
• PCA helps us find such a subspace whose coordinates are linear functions of the originals
Principal Components Analysis

- Given an n-dimensional real-valued space, data are often nearly restricted to a lower-dimensional subspace.
- PCA helps us find such a subspace whose coordinates are linear functions of the originals.
Principal Components Analysis

- Given an n-dimensional real-valued space, data are often nearly restricted to a lower-dimensional subspace.
- PCA helps us find such a subspace whose coordinates are linear functions of the originals.
Cartoon of algorithm

- Normalize the data (subtract mean, divide by stdev)
Cartoon of algorithm

• Normalize the data (subtract mean, divide by stddev)
• Find the line along which the data has the most variability: that’s the first principal component
Cartoon of algorithm

• Normalize the data (subtract mean, divide by stdev)
• Find the line along which the data has the most variability: that’s the first principal component
• Project the data into the n-1 dimensional space orthogonal to the line
• Repeat
Cartoon of algorithm

- Normalize the data (subtract mean, divide by stdev)
- Find the line along which the data has the most variability: that’s the first principal component
- Project the data into the n-1 dimensional space orthogonal to the line
- Repeat

- Result is a new orthogonal set of axes
- First k give a lower-D space that represents the variability of the data as well as possible
Cartoon of algorithm

- Normalize the data (subtract mean, divide by stdev)
- Find the line along which the data has the most variability: that’s the first principal component
- Project the data into the n-1 dimensional space orthogonal to the line
- Repeat

- Result is a new orthogonal set of axes
- First k give a lower-D space that represents the variability of the data as well as possible
- Really: find the eigenvectors of the covariance matrix with the k largest eigenvalues
Linear Transformations Only

There are fancier methods that can find this structure
Insensitive to Classification Task
Insensitive to Classification Task

There are fancier methods that can take class into account
Validating a Classifier

<table>
<thead>
<tr>
<th>true y</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>
# Validating a Classifier

<table>
<thead>
<tr>
<th></th>
<th>predicted $y$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

- **false positive type 1 error**

- **true $y$**
  - 0
  - 1

- **predicted $y$**
  - 0
  - 1
Validating a Classifier

predicted $y$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

false positive
type 1 error

false negative
type 2 error
Validating a Classifier

- **sensitivity**: $P(\text{predict 1 } | \text{ actual 1}) = \frac{D}{(C+D)}$
  - “true positive rate” (TP)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

- **false positive** (type 1 error)
- **false negative** (type 2 error)
Validating a Classifier

- sensitivity: $P(\text{predict } 1 \mid \text{actual } 1) = \frac{D}{C+D}$
  - “true positive rate” (TP)

- specificity: $P(\text{predict } 0 \mid \text{actual } 0) = \frac{A}{A+B}$
Validating a Classifier

- sensitivity: $P(\text{predict 1 } | \text{ actual 1}) = \frac{D}{C+D}$
  - “true positive rate” (TP)

- specificity: $P(\text{predict 0 } | \text{ actual 0}) = \frac{A}{A+B}$

- false-alarm rate: $P(\text{predict 1 } | \text{ actual 0}) = \frac{B}{A+B}$
  - “false positive rate” (FP)

![Confusion Matrix Diagram]

false negative

true

false positive
type 1 error

false positive
type 1 error

false negative
type 2 error
Cost Sensitivity

• Predict whether a patient has pseuditis based on blood tests
  • Disease is often fatal if left untreated
  • Treatment is cheap and side-effect free
Cost Sensitivity

• Predict whether a patient has pseuditis based on blood tests
  • Disease is often fatal if left untreated
  • Treatment is cheap and side-effect free

• Which classifier to use?
  • Classifier 1: TP = 0.9, FP = 0.4
Cost Sensitivity

- Predict whether a patient has pseuditis based on blood tests
  - Disease is often fatal if left untreated
  - Treatment is cheap and side-effect free

- Which classifier to use?
  - Classifier 1: TP = 0.9, FP = 0.4
  - Classifier 2: TP = 0.7, FP = 0.1
Build Costs into Classifier

• Assess costs of both types of error
  • use a different splitting criterion for decision trees
  • make error function for neural nets asymmetric; different costs for each kind of error
• use different values of C for SVMs depending on kind of error
Tunable Classifiers

- Classifiers that have a threshold (naïve Bayes, neural nets, SVMs) can be adjusted, post learning, by changing the threshold, to make different trade-offs between type 1 and type 2 errors.
Tunable Classifiers

- Classifiers that have a threshold (naïve Bayes, neural nets, SVMs) can be adjusted, post learning, by changing the threshold, to make different trade-offs between type 1 and type 2 errors.

- $C_1, C_2$: costs of errors
- $P$: percentage of positive examples
- $x$: tunable threshold
- $TP(x)$: true positive rate at threshold $x$
- $FP(x)$: false positive rate at threshold $x$

- Expected Cost $= C_2 P(1-TP(x)) + C_1 (1-P) FP(x)$
- Choose $x$ to minimize expected cost.
ROC Curves

- “receiver operating characteristics”
ROC Curves

• “receiver operating characteristics”

- Ideal: Always output 1
- Always output 0: Always output 0

Diagram:
- TP (True Positive) axis
- FP (False Positive) axis
- ROC curve
- Ideal point: Star
- Always output 1
- Always output 0
ROC Curves

• “receiver operating characteristics”

![Diagram of ROC Curves]

- **Ideal**: Always output 0
- **Always output 1**: Output 1 always
- **Parametric function of x**: The curve represents a parametric function of x, indicating the trade-off between true positive (TP) and false positive (FP) rates.
ROC Curves

- “receiver operating characteristics”

- Blue curve dominates red

- Ideal output is always 0

- Always output 1
Many more issues!

- Missing data
- Many examples in one class, few in other (fraud detection)
- Expensive data (active learning)
- ...