Supervised Learning

Artificial Neural Networks and Single Layer Perceptron
Supervised Learning

• Learning from correct answers

Training info: desired (target outputs)
Supervised Learning Methods

- Artificial neural networks
- Decision trees
- Gaussian process regression
- Naive Bayes classifier
- Nearest neighbor algorithm
- Support vector machines
- Random forests
- Ensembles of classifiers
- …
Applications of Supervised Learning

- Handwriting recognition
- Object recognition
- Optical character recognition
- Spam detection
- Pattern recognition
- Speech recognition
- Prediction models
- …
Inspiration: The Human Brain

• The brain doesn’t seem to have a CPU

• Instead, it has got lots of simple parallel, asynchronous units, called *neurons*

• There are about $10^{11}$ (100 billion) neurons of about 20 types

• GPU NVIDIA Tesla K80 has 4992 cores
Neurons

- A neuron is a single cell that has a number of relatively short fibers, called **dendrites**, and one long fiber, called an **axon**.
- A synapse is a structure that permits a neuron (or nerve cell) to pass an electrical signal to another cell.
Signal Generation: Action Potential

- The fibers of surrounding neurons emit chemicals (neurotransmitters) that move across the synapse and change the electrical potential of the cell body.
From Real to Artificial Neurons

\[ a = \sum_{i=1}^{n} w_i u_i \]

\[ a \text{= activation function} \]

\[ \theta \text{= output} \]
Artificial Neurons

- **Threshold Logic Unit (TLU)** proposed by Warren McCulloch and Walter Pitts in 1943
  - Initially with binary inputs and outputs
  - Heaviside function as threshold

- **Perceptron** developed by Frank Rosenblatt in 1957
  - Arbitrary inputs and outputs
  - Linear transfer function
Activation Functions

- **Threshold**
- **Piece-wise Linear**
- **Linear**
- **Sigmoid**
Threshold Logic Unit (TLU)

\[ a = \sum_{i=1}^{n} w_i u_i \]

\[ v = \begin{cases} 
1 & \text{if } a \geq \theta \\
0 & \text{if } a < \theta 
\end{cases} \]
Decision Surface of a TLU

Decision line:
\[ w_1 u_1 + w_2 u_2 = \theta \]
Scalar Products and Projections

\[ w \cdot u > 0 \]
\[ w \cdot u = 0 \]
\[ w \cdot u < 0 \]

\[ w \cdot u = |w||u| \cos \phi \]
\[ |u_w| = |u| \cos \phi \]
\[ w \cdot u = |w||u_w| \]
Geometric Interpretation

\[ w_1 u_1 + w_2 u_2 = \theta \]
\[ w \cdot u = \theta \]
\[ w \cdot u = |w||u| \cos \varphi \]
\[ |u_w| = |u| \cos \varphi \]
\[ w \cdot u = |w||u_w| = \theta \]
\[ |u_w| = \frac{\theta}{|w|} \]

Decision line
Geometric Interpretation (cont.)

Decision line

\[ w_1 u_1 + w_2 u_2 = \theta \]

\[ w \cdot u = \theta \]

\[ w \cdot u = |w||u| \cos \varphi \]

\[ |u_w| = |u| \cos \varphi \]

\[ w \cdot u = |w||u_w| = \theta \]

\[ |u_w| = \theta/|w| \]
Decision line

\[ w_1 u_1 + w_2 u_2 = \theta \]
\[ w \cdot u = \theta \]
\[ w \cdot u = |w||u| \cos \varphi \]
\[ |u_w| = |u| \cos \varphi \]
\[ w \cdot u = |w||u_w| = \theta \]
\[ |u_w| = \theta/|w| \]
In $n$ dimensions the relation $w \cdot u = \theta$ defines a $n-1$ dimensional hyper-plane, which is perpendicular to the weight vector $w$.

On one side of the hyper-plane ($w \cdot u > \theta$) all patterns are classified by the TLU as “1”, while those that get classified as “0” lie on the other side of the hyper-plane.
Example: Logical AND

\[ u_1 \quad u_2 \quad a \quad v \]

\[
\begin{array}{cccc}
0 & 0 & ? & 0 \\
0 & 1 & ? & 0 \\
1 & 0 & ? & 0 \\
1 & 1 & ? & 1 \\
\end{array}
\]
Example: Logical AND

\[ w_1 = 1 \]
\[ w_2 = 1 \]
\[ \theta = ? \]

\[
\begin{array}{c|c|c|c|c}
  u_1 & u_2 & a & v \\
  \hline
  0 & 0 & 0 & 0 \\
  0 & 1 & 1 & 0 \\
  1 & 0 & 1 & 0 \\
  1 & 1 & 2 & 1 \\
\end{array}
\]
Example: Logical AND

$w_1 = 1$
$w_2 = 1$
$\theta = 1.5$

<table>
<thead>
<tr>
<th>$u_1$</th>
<th>$u_2$</th>
<th>$a$</th>
<th>$v$</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>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Example: Logical OR

\[ w_1 = ? \]
\[ w_2 = ? \]
\[ \theta = ? \]

\[ \begin{array}{cccc}
  u_1 & u_2 & a & v \\
  0 & 0 & ? & 0 \\
  0 & 1 & ? & 1 \\
  1 & 0 & ? & 1 \\
  1 & 1 & ? & 1 \\
\end{array} \]
Example: Logical OR

\( w_1 = 1 \)
\( w_2 = 1 \)
\( \theta = 0.5 \)

\[
\begin{array}{c|c|c|c|c}
   u_1 & u_2 & a & v \\
   \hline
   0 & 0 & 0 & 0 \\
   0 & 1 & 1 & 1 \\
   1 & 0 & 1 & 1 \\
   1 & 1 & 2 & 1 \\
\end{array}
\]
Example: Logical NOT

\[ w_1 = ? \]
\[ \theta = ? \]

<table>
<thead>
<tr>
<th>( u_1 )</th>
<th>( a )</th>
<th>( v )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>?</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>?</td>
<td>0</td>
</tr>
</tbody>
</table>
Example: Logical NOT

- \( w_1 = -1 \)
- \( \theta = -0.5 \)

<table>
<thead>
<tr>
<th>( u_1 )</th>
<th>( a )</th>
<th>( v )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>
Threshold as Weight

\[ \sum u_{n+1} = -1 \]

\[ a = \sum_{i=1}^{n+1} w_i u_i \]

\[ \theta = w_{n+1} \]

\[ v = \begin{cases} 
1 & \text{if } a \geq 0 \\
0 & \text{if } a < 0 
\end{cases} \]
Geometric Interpretation

Decision line

\[ w \cdot u = 0 \]
Training ANNs

- **Training set S of examples** \{u,v^t\}
  - u is an input vector and
  - v^t the desired target output
  - Example: Logical And
    S = \{(0,0),0\}, \{(0,1),0\}, \{(1,0),0\}, \{(1,1),1\}

- **Iterative process**
  - present a training example u
  - compute network output v
  - compare output v with target v^t
  - adjust weights w and threshold θ

- **Learning rule**
  - Specifies how to change the weights w and threshold θ of the network as a function of the inputs u, output v and target v^t.
Presentation of Training Samples

• Presenting all training examples once to the ANN is called an epoch

• Training examples can be presented in
  – Fixed order (1,2,3…,M) [on/off-line]
  – Randomly permutated order (5,2,7,…,3) [off-line]
  – Completely random (4,1,7,1,5,4,…….) [off-line]
Perceptron Learning Rule

• \( w' = w + \mu (v^t - v) \ u \)

  – \( w'_i = w_i + \Delta w_i = w_i + \mu (v^t - v) \ u_i \) \quad (i=1..n+1), with \( w_{n+1} = \theta \) and \( u_{n+1} = -1 \)

  – The parameter \( \mu \) is called the \textit{learning rate}. It determines the magnitude of weight updates \( \Delta w_i \)

  – If the output is correct \((v^t = v)\) the weights are not changed \((\Delta w_i = 0)\)

  – If the output is incorrect \((v^t \neq v)\) the weights \( w_i \) are changed such that the output of the TLU for the new weights \( w'_i \) is \textit{closer/further} to the input \( u_i \)
• If we do not include the threshold as an input we use the following description of the perceptron with symmetrical outputs

\[
v = \begin{cases} 
+1 & \text{if } w \cdot u - \theta \geq 0 \\
-1 & \text{if } w \cdot u - \theta < 0 
\end{cases}
\]

• Then we get the following learning rule

\[
w' = w + \frac{\mu}{2} (v^t - v) u \quad \text{and} \quad \theta' = \theta - \frac{\mu}{2} (v^t - v)
\]
Adjusting the Weight Vector

\[ \text{Target } v^t = 1 \]
\[ \text{Output } v = 0 \]

\[ \phi > 90 \]

Move \( w \) in the direction of \( u \)

\[ \text{Target } v^t = 0 \]
\[ \text{Output } v = 1 \]

\[ \phi < 90 \]

Move \( w \) away from the direction of \( u \)

\[ w' = w - \mu u \]

\[ w' = w + \mu u \]
Procedure of Perceptron Training

• While $v \neq v^t$ for all training vector pairs
  – For each training vector pair $(u, v^t)$
    • calculate the output $v$: $v = \sum wu$
    • if $v \neq v^t$
      update weight vector $w$: $w' = w + \mu (v^t - v)u$
      update threshold $\theta$: $\theta' = \theta - \mu(v^t - v)$ (if not included in weights)
    else
      do nothing
Demo: Perceptron with TLU
TLU Convergence

- The algorithm converges to the correct classification
  - if the training data is linearly separable
  - given sufficiently small learning rate $\mu$

- Solution $w$ is not unique, since if $w \cdot u = 0$ defines a hyper-plane, so does $w' = k \cdot w$
Multiple TLUs

- Learning rule:

\[ w'_{ji} = w_{ji} + \mu (v^t_j - v_j) u_i \]

\( w_{ji} \) connects \( u_i \) with \( v_j \)
Example: Character Recognition

- 26 classes: A, B, C, ..., Z
- Target output is a vector
  - e.g., $v^t_A = [1 \ 0 \ 0 \ ... \ 0]$, $v^t_B = [0 \ 1 \ 0 \ ... \ 0]$, ...

\[
\begin{array}{cccccccc}
A & B & \cdots & Z \\
v_1 & v_2 & \cdots & v_{26} \\
u_1 & u_2 & u_3 & \cdots & u_n \\
\end{array}
\]
Activation Functions

threshold

piece-wise linear

linear

sigmoid
Perceptron with Linear Function

\[ v = a = \sum_{i=1}^{n} w_i v_i \]

Note: We will use notation \( t \) instead of \( v^t \)
Gradient Descent Learning Rule

- Consider linear unit without threshold and continuous output \( v \) (not just \(-1,1/1,0\))
  \[ v = w_1 u_1 + \ldots + w_n u_n \]

- Train the \( w_i \) such that they minimize the squared error
  \[ E[w_1,\ldots,w_n] = \frac{1}{2} \sum_{d \in D} (t_d - v_d)^2 \]
  - where \( D \) is the set of training examples and
  - \( t \) the target outputs
Gradient Descent Learning Rule (cont.)

Gradient:
\[ \nabla E[w] = [\partial E/\partial w_0, \ldots, \partial E/\partial w_n] \]
\[ \Delta w = -\mu \nabla E[w] \]
\[ E_d[w] = 1/2 (t_d-v_d)^2 \]

\[-1/\mu \Delta w_i = \partial E/\partial w_i = \partial/\partial w_i 1/2 (t_d-v_d)^2 = \partial/\partial w_i 1/2 (t_d-\Sigma_i w_i u_i)^2 = \Sigma_d (t_d- v_d) (-u_i) \]

We get:
\[ \Delta w_i = \mu (t_d- v_d) u_i \]
Also known as Delta rule
Incremental vs. Batch Gradient Descent

- **Incremental (stochastic) mode:**
  \[ w' = w - \mu \nabla E_d[w] \text{ over individual training examples } d \]
  \[ E_d[w] = \frac{1}{2} (t_d - v_d)^2 \]

- **Batch mode:**
  \[ w' = w - \mu \nabla E_D[w] \text{ over the entire data } D \]
  \[ E_D[w] = \frac{1}{2} \sum_d (t_d - v_d)^2 \]
Perceptron vs. Gradient Descent Rule

• Perceptron rule
  \[ w'_{i} = w_{i} + \mu \ (v^{t}-v) \ u_{i} \]
  – derived from manipulation of decision surface

• Gradient descent rule
  \[ w'_{i} = w_{i} + \mu \ (t_{d}-v_{d}) \ u_{i} \]
  – derived from minimization of error function
    \[ E[w_{1},...,w_{n}] = 1/2 \ \Sigma_{d} \ (t_{d}-v_{d})^{2} \]
    by means of gradient descent
Demo: Gradient Descent Learning Rule
Linear Regression

• Common statistical method

• Describe relationship between variables

• Predict one variable knowing the others

• We make assumption that there is a \textit{linear} relationship
  
  – between an outcome (dependent variable, response variable) and a predictor (independent variable, explanatory variable, feature) or

  – between one variable and several other variables
Describing Relationships

Very weak relationship / very low correlation

cor. coef. = -0.1094

Strong relationship / high correlation

cor. coef = 0.8180
Making predictions

Predicting variables

Predicting future outcomes

Father's height (cm) vs. Son's height (cm)

Profit (Euro) vs. Days

Questions mark indicate data points for prediction.
Demo: Linear Regression
Perceptron vs. Gradient Descent

• **Perceptron (TLU)**
  – Guaranteed to succeed if
    • training examples are linearly separable
    • given sufficiently small learning rate $\mu$
  – Robust against outliers

• **Gradient descent (linear transfer function)**
  – Guaranteed to converge to hypothesis with minimum squared error
    • even when training data not separable by hyperplane
    • given sufficiently small learning rate $\mu$
  – Very sensitive to outliers
Activation Functions

threshold

piece-wise linear

linear

sigmoid
Perceptron with Sigmoid Function

\[ a = \sum_{i=1}^{n} w_i u_i \]

\[ v = \sigma(a) = \frac{1}{1 + e^{-a}} \]
Gradient Descent Rule for Sigmoid Activation Function

Gradient:
\[ \nabla E[w] = [\partial E/\partial w_0, \ldots, \partial E/\partial w_n] \]
\[ \Delta w = -\mu \nabla E[w] \]
\[ E_d[w] = 1/2 (t_d-v_d)^2 \]

-1/\mu \Delta w_i = \partial E/\partial w_i

= \partial/\partial w_i 1/2 (t_d-v_d)^2
= \partial/\partial w_i 1/2 (t_d - \sigma(\sum_i w_i u_i))^2
= (t_d-v_d) \sigma'(\sum_i w_i u_i) (-u_i)

We get:
\[ \Delta w_i = \mu v_d (1-v_d) (t_d-v_d) u_i \]

\[ v_d = \sigma(a) = 1/(1+e^{-a}) \]
\[ \sigma'(a) = e^{-a}/(1+e^{-a})^2 = \sigma(a) (1-\sigma(a)) \]
Demo: Gradient Descent with Sigmoid Activation Function
Summary

- **Threshold Logic Unit**
  - works only for linearly separable patterns
  - robust to outliers

- **Linear Neuron**
  - converges to minimal squared error even when patterns are not linearly separable
  - very sensitive to outliers

- **Sigmoid Neuron**
  - combination of TLU and linear neuron
  - converges to minimal squared error even when patterns are not linearly separable
  - robust to outliers