Supervised learning neural networks

• Multilayer perceptron
• Adaptive-Network-based Fuzzy Inference System (ANFIS)

First part based on slides by Walter Kosters
Neural networks

- Massively connected computational units inspired by the working of the human brain
- Provide a mathematical model for biological neural networks (brains)

Characteristics:
- Learning from examples
- Adaptive and fault tolerant
- Robust for fulfilling complex tasks
Network variations

- Learning methods (supervised, unsupervised)
- Architectures (feedforward, recurrent)
- Output types (binary, continuous)
- Node types (uniform, hybrid)
- Implementations (software, hardware)
- Connection weights (adjustable, hard-wired)
- Inspirations (biological, psychological)
Artificial Neuron

- **Input:**
  \[ i_{n_i} = \sum_j W_{j,i} \ a_j \]

- **Network with one unit:** perceptron
### Activation Functions

- **Step**
  
  \[ \text{step}_t(x) = \begin{cases} 
  1 & \text{if } x \geq t \\
  0 & \text{otherwise} 
  \end{cases} \]

- **Sign**
  
  \[ \text{sign}(x) = \begin{cases} 
  1 & \text{if } x \geq 0 \\
  -1 & \text{if } x < 0 
  \end{cases} \]
Activation Functions

- **Sigmoid (Logistic)**
  \[ \text{sigmoid}(x) = \frac{1}{1 + e^{bx}} \]

- **Hyperbolic tangent**
  \[ \text{htan}(x) = \text{tanh}\left(\frac{x}{2}\right) = \frac{1 - e^{-x}}{1 + e^{-x}} \]

- **Fuzzy membership functions!**
Perceptron: Encoding Simple Boolean Functions

- Step function
- Input in $[0,1]$
- Output in $[0,1]$
Perceptron: Linear Separable

step function:

\[ 1 \text{ if } -W_0 + W_1 x_1 + \ldots + W_n x_n \geq 0 \]
\[ 0 \text{ otherwise} \]
Perceptron: Learning

- **Update rule:**
  \[ W_j \leftarrow W_j + \alpha \cdot x_j \cdot \text{Error} \]

  where
  - \( \text{Error} = \text{target} - \text{predicted} \)
  - \( \alpha = \text{learning rate} \)

<table>
<thead>
<tr>
<th>( x_0 x_1 x_2 )</th>
<th>( W_0 )</th>
<th>( W_1 )</th>
<th>predicted</th>
<th>target</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1 1 1</td>
<td>-0.160</td>
<td>-0.606</td>
<td>-0.217</td>
<td>0.000</td>
</tr>
<tr>
<td>2</td>
<td>-1 1 0</td>
<td>-0.260</td>
<td>-0.506</td>
<td>-0.117</td>
<td>0.000</td>
</tr>
<tr>
<td>3</td>
<td>-1 0 1</td>
<td>-0.260</td>
<td>-0.506</td>
<td>-0.117</td>
<td>1.000</td>
</tr>
<tr>
<td>4</td>
<td>-1 1 0</td>
<td>-0.160</td>
<td>-0.506</td>
<td>-0.217</td>
<td>0.000</td>
</tr>
<tr>
<td>5</td>
<td>-1 1 0</td>
<td>-0.160</td>
<td>-0.506</td>
<td>-0.217</td>
<td>0.000</td>
</tr>
<tr>
<td>6</td>
<td>-1 1 1</td>
<td>-0.160</td>
<td>-0.506</td>
<td>-0.217</td>
<td>0.000</td>
</tr>
<tr>
<td>7</td>
<td>-1 0 0</td>
<td>-0.260</td>
<td>-0.406</td>
<td>-0.117</td>
<td>1.000</td>
</tr>
<tr>
<td>8</td>
<td>-1 0 0</td>
<td>-0.160</td>
<td>-0.406</td>
<td>-0.117</td>
<td>1.000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>70</td>
<td>-1 1 0</td>
<td>0.140</td>
<td>0.194</td>
<td>0.183</td>
<td>1.000</td>
</tr>
<tr>
<td>71</td>
<td>-1 1 1</td>
<td>0.240</td>
<td>0.094</td>
<td>0.183</td>
<td>1.000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Feed-forward network

- If one layer in network: perceptron (no hidden units)
- Otherwise: multi-layer
Backpropagation

Output units $a_i$

Hidden units $a_j$

Input units $a_k$

Weight matrices $W_{j,i}$ and $W_{k,j}$
Backpropagation

- Update rule output layer:
  \[ W_{j,i} \leftarrow W_{j,i} + \alpha \cdot a_j \cdot \Delta_i \text{ with } \Delta_i = \text{Error}_i \cdot g'(\text{in}_i) \]

- Update rule hidden layer:
  \[ W_{k,j} \leftarrow W_{k,j} + \alpha \cdot a_k \cdot \Delta_j \text{ with } \Delta_j = g'(\text{in}_j) \sum_i W_{j,i} \Delta_i \]

- \( \alpha \) learning rate
- \( a_k \) activation input node
- \( \text{in}_i \) weighted input
- \( \text{in}_j \) weighted input
- \( \text{out}_j \) weighted output
Error propagation network

\[ x_1, x_2, \ldots, x_8, x_9 \]

\[ w_{31}, w_{52}, w_{75}, w_{83}, w_{97} \]

\[ \epsilon_1, \epsilon_2, \epsilon_8, \epsilon_9 \]
Backpropagation Derivation

- Gradient descent
- Error is $E = \frac{1}{2} \sum_i (y_i - a_i)^2$ for all output nodes
- $\frac{\partial E}{\partial W_{j,i}} = -(y_i - a_i) \cdot \frac{\partial a_i}{\partial W_{j,i}} = -(y_i - a_i) \cdot \frac{\partial g(\sum_j W_{j,i}a_j)}{\partial W_{j,i}} = -(y_i - a_i) \cdot g'(i n_i) \cdot a_j = -a_j \cdot \Delta_i$

$\Delta_i = \text{Error}_i \cdot g'(i n_i)$
\[ \Delta_i = \text{Error}_i \cdot g'(\text{in}_i) \]

**Backpropagation Derivation**

\[
\frac{\partial E}{\partial W_{k,j}} = - \sum_i (y_i - a_i) \cdot \frac{\partial a_i}{\partial W_{k,j}}
\]

\[
= - \sum_i (y_i - a_i) \cdot \frac{\partial g(\sum_j W_{j,i} a_j)}{\partial W_{k,j}}
\]

\[
= - \sum_i (y_i - a_i) \cdot g'(\text{in}_i) \cdot \frac{\partial (\sum_j W_{j,i} a_j)}{\partial W_{k,j}}
\]

\[
= - \sum_i \Delta_i \cdot W_{j,i} \cdot \frac{\partial a_j}{\partial W_{k,j}}
\]

\[
= - \sum_i \Delta_i \cdot W_{j,i} \cdot \frac{\partial g(\text{in}_j)}{\partial W_{k,j}}
\]

\[
= - \sum_i \Delta_i \cdot W_{j,i} \cdot g'(\text{in}_j) \cdot \frac{\partial (\sum_k W_{k,j} a_k)}{\partial W_{k,j}}
\]

\[
= - \sum_i \Delta_i \cdot W_{j,i} \cdot g'(\text{in}_j) \cdot a_k
\]

\[
= - a_k \Delta_j
\]

\[ \Delta_j = g'(\text{in}_j) \sum_i W_{j,i} \Delta_i \]
Multilayer perceptron

\[ \theta = -w_0 \]
Common MLP architecture

- One hidden layer
- Hidden neurons: tanh or logistic activation
- Output neurons: linear activation
Online vs Offline learning

- *online*: update after each example $p$

$$W_{j,i} \leftarrow W_{j,i} - \alpha \frac{\partial E^p}{\partial W_{j,i}}$$

- *offline*: update after seeing all examples

$$W_{j,i} \leftarrow W_{j,i} - \alpha \sum_p \frac{\partial E^p}{\partial W_{j,i}}$$
Speeding up backpropagation

- Momentum term
  \[
  \Delta W_{j,i}^t = \alpha \cdot a_j \cdot \Delta_i + \eta \Delta W_{j,i}^{t-1}
  \] (Output layer)

\[
\Delta W_{j,i}^t = -\alpha \cdot \frac{\partial E}{\partial W_{j,i}} + \eta \Delta W_{j,i}^{t-1}
\] (General case)

- Regularized initialization

- Learning rate rescaling
  learning rate of layers decreases towards the output layer
Approximation power

- General function approximators

- Feed-forward neural network with one hidden layer and sigmoidal activation functions can approximate any continuous function arbitrarily well (Cybenko)
ANFIS

- Takagi-Sugeno fuzzy system mapped onto a neural network structure
- Different representations are possible, but one with 5 layers is the most common
- Network nodes in different layers have different structures

\[ f_1 = p_1x + q_1y + r_1 \]

\[ f_2 = p_2x + q_2y + r_2 \]

\[ f = \frac{w_1f_1 + w_2f_2}{w_1 + w_2} \]

\[ = \bar{w}_1f_1 + \bar{w}_2f_2 \]
ANFIS architecture

(product)

(normalization)
ANFIS layers

- Layer 1: every node is adaptive with function
  \[ O_{1,i} = \mu_i(x_i) \]

- Layer 2: every node is fixed and computes a T-norm of the inputs
  \[ O_{2,i} = \bigwedge_j O_{1,j} \]

- Layer 3: every node is fixed with function
  \[ O_{3,i} = \frac{O_{2,i}}{\sum_j O_{2,j}} \]

- Layer 4: every node is adaptive with function
  \[ O_{4,i} = O_{3i}(p_0 + p_1 x_1 + \cdots + p_n x_n) \]

- Layer 5: single node sums up inputs
  \[ O_{5,i} = \sum_j O_{4,j} \]
ANFIS with multiple rules
Hybrid learning for ANFIS

- Consider an ANFIS with two inputs $x$ & $y$
- Let there be 2 nodes in layers 2-3-4
- Denote the consequent parameters by $p$, $q$, $r$
- ANFIS output is now given by

$$f = \frac{w_1}{w_1+w_2} f_1 + \frac{w_2}{w_1+w_2} f_2$$

$$= w_1 (p_1x + q_1y + r_1) + w_2 (p_2x + q_2y + r_2)$$

$$= (w_1x) p_1 + (w_1y) q_1 + (w_1) r_1 + (w_2x) p_2 + (w_2y) q_2 + (w_2) r_2$$

- Partition $S$ as follows:
  $S_1$: premise parameters (nonlinear)
  $S_2$: consequent parameters (linear)
Hybrid learning for ANFIS

How to learn the parameters of a linear model

\[ f(\vec{x}, \vec{\beta}) = \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n \]

such that

\[ E = \frac{1}{2} \sum_i (f(\vec{x}_i, \vec{\beta}) - y_i)^2 \]

is minimal?

(where \( \vec{x}_i \) and \( y_i \) represent input and output for training examples)

→ least squares linear regression
Linear Regression

- Error function:

\[ E = \frac{1}{2} \sum_i (\beta^T \vec{x}_i - y_i)^2 \]

- Compute global minimum by means of derivative:

\[ \nabla E = \sum_i (\beta^T \cdot \vec{x}_i - y_i) \vec{x}_i^T \]

\[ 0 = \beta^T \cdot \left( \sum_i \vec{x}_i \cdot \vec{x}_i^T \right) - \sum_i y_i \cdot \vec{x}_i^T \]

\[ \beta = \left( \sum_i \vec{x}_i \cdot \vec{x}_i^T \right)^{-1} \sum_i y_i \cdot \vec{x}_i^T = (X^T X)^{-1} X^T \bar{y} \]
Stone-Weierstrass theorem

Let $D$ be a compact space of $N$ dimensions and let $F$ be a set of continuous real-valued functions on $D$ satisfying the following

- **Identity function:** $f(x) = 1$ is in $F$
- **Separability:** for any two points $x_1 \neq x_2$ in $D$, there is an $f$ in $F$ s.t. $f(x_1) \neq f(x_2)$
- **Algebraic closure:** if $f$ and $g$ are two functions in $F$, then $fg$ and $af + bg$ are also in $F$ for real $a$ and $b$

$\Rightarrow F$ is dense in the set of continuous functions $C(D)$ on $D$:

$$(\forall \varepsilon > 0) (\forall g \in C(D)) (\exists f \in F) (\forall x \in D : |g(x) - f(x)| < \varepsilon)$$
Universal approximation in ANFIS

- According to Stone-Weierstrass theorem, an ANFIS can approximate any continuous nonlinear function arbitrarily well

- Restriction to rules with a constant in the consequent and Gaussian membership functions

- **Identity:** $f(x) = 1$ is in $F$
  Obtained by having a constant consequent

- **Separability:** for any two points $x_1 \neq x_2$ in $D$, there is $f$ in $F$ s.t. $f(x_1) \neq f(x_2)$
  Obtained by selecting different parameters in the network
Algebraic closure

Consider two systems with two rules

\[ z = \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} \quad \text{and} \quad \hat{z} = \frac{\hat{w}_1 \hat{f}_1 + \hat{w}_2 \hat{f}_2}{\hat{w}_1 + \hat{w}_2} \]

- **Additive**
  \[ az + b\hat{z} = a \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} + b \frac{\hat{w}_1 \hat{f}_1 + \hat{w}_2 \hat{f}_2}{\hat{w}_1 + \hat{w}_2} \]
  \[ = \frac{w_1 \hat{w}_1 (af_1 + bf_1) + w_1 \hat{w}_2 (af_1 + bf_2) + w_2 \hat{w}_1 (af_2 + bf_1) + w_2 \hat{w}_2 (af_2 + bf_2)}{w_1 \hat{w}_1 + w_1 \hat{w}_2 + w_2 \hat{w}_1 + w_2 \hat{w}_2} \]

- **Multiplicative**
  \[ z\hat{z} = \left( \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} \right) \left( \frac{\hat{w}_1 \hat{f}_1 + \hat{w}_2 \hat{f}_2}{\hat{w}_1 + \hat{w}_2} \right) \]
  \[ = \frac{w_1 \hat{w}_1 \hat{f}_1 f_1 + w_1 \hat{w}_2 f_1 \hat{f}_2 + w_2 \hat{w}_1 f_2 \hat{f}_1 + w_2 \hat{w}_2 f_2 \hat{f}_2}{w_1 \hat{w}_1 + w_1 \hat{w}_2 + w_2 \hat{w}_1 + w_2 \hat{w}_2} \]

Use Gaussian membership functions!
Model building guidelines

- Select number of fuzzy sets per variable by
  - empirically by examining data
  - clustering techniques
  - regression trees (CART)
- Initially, distribute bell-shaped membership functions evenly
- Using an adaptive step size can speed up training
What to do?

- Number of inputs
- What inputs
- Number of outputs
- What outputs
- Number of rules
- Type of consequent functions

- Normalize inputs
- Collect data
- Define objective function
- Determine initial rules
- Initialize network
- Define stop conditions

TRAIN
Two-input sinc function

- Input range: \([-10,10]\times[-10,10]\]
  121 data points

\[ z = \text{sinc}(x, y) = \frac{\sin(x)\sin(y)}{xy} \]

- ANFIS: 16 rules, 4 membership functions per variable, 72 parameters (48 consequence, 24 premise)
- MLP: 18 neurons in hidden layer, 73 parameters, quick propagation
- ANFIS takes longer for same number of epochs
MLP vs. ANFIS results

Average of 10 runs
ANFIS output

Training data

ANFIS Output

error curve

learning rate curve

root mean squared error

step size

epoch number
ANFIS model

- Initial MFs on X
  - Degree of membership
  - Input: input1

- Initial MFs on Y
  - Degree of membership
  - Input: input2

- Final MFs on X
  - Degree of membership
  - Input: input1

- Final MFs on Y
  - Degree of membership
  - Input: input2
Modeling dynamic systems

- $f(.)$ has the following form
  
  $$f(u) = 0.6 \sin(\pi u) + 0.3 \sin(3\pi u) + 0.1 \sin(5\pi u)$$

  where
  
  $$u(k) = \sin(2\pi k / 250)$$

- ANFIS parameters updated at each step

- Learning rate: 0.1; forgetting factor: 0.99

- ANFIS can adapt even after the input changes
Plant and model outputs
Effect of number of MFs

Initial MFs

Final MFs

f(u) and ANFIS Outputs

Each Rule's Outputs

5 membership functions
Effect of number of MFs

Initial MFs

$4$ membership functions

Final MFs

$f(u)$ and ANFIS Outputs

Each Rule's Outputs
Effect of number of MFs

Initial MFs

Final MFs

f(u) and ANFIS Outputs

Each Rule's Outputs
Three-input nonlinear function

- System mapping is given by
  \[ o = \left( 1 + \sqrt{x} + \frac{1}{y} + \frac{1}{\sqrt{z^3}} \right)^2 \]
- Two membership functions per variable, 8 rules
- Input ranges: [1,6]x[1,6]x[1,6]
- 215 training data, 125 validation data
ANFIS model

Initial MFs on X, Y and Z

Input 1

Final MFs on X

Input 2

Final MFs on Y

Input 3
Predicting chaotic time series

- Consider a chaotic time series generated by
  \[ \frac{dx(t)}{dt} = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t) \]

- Task: predict the output of system at some future instance \( t+P \) by using past output

- 500 training data, 500 validation data

- ANFIS input: \([x(t-18), x(t-12), x(t-6), x(t)]\)

- ANFIS output: \(x(t+6)\)

- Two MFs per variable, 16 rules

- 104 parameters (24 premise, 80 consequent)

- Data generated from \( t=118 \) to \( t=1117 \)
ANFIS model

Final MFs on Input 1, \(x(t - 18)\)

Final MFs on Input 2, \(x(t - 12)\)

Final MFs on Input 3, \(x(t - 6)\)

Final MFs on Input 4, \(x(t)\)
Model output

Error Curves

- Training Error
- Checking Error

Step Sizes

Prediction Errors
Auto-Regression (AR) Model

- Linear prediction:
  \[ x(t + 6) = a_0 + a_1 x(t) + a_2 x(t - 6) + a_3 x(t - 12) + \cdots + a_{103} x(t - 102 \times 6) \]

- Order = number of terms
103rd order AR model

(a) Desired (Solid Line) and Predicted (Dashed Line) MG Time Series

(b) Prediction Errors
Order selection

- Select optimal order of AR model in order to prevent overfitting
- Select the order that minimizes the error on a test set

Root mean square error / standard deviation target
44th order AR model

(a) Desired (Solid Line) and Predicted (Dashed Line) MG Time Series

(b) Prediction Errors
ANFIS output for P=84

(a) Desired (Solid) and Predicted (Dashed) Time Series of ANFIS When P=84

(b) Prediction Errors
ANFIS extensions

- Different types of membership functions in layer 1
- Parameterized t-norms in layer 2
- Interpretability
  - constrained gradient descent optimization
  - bounds on fuzziness
    \[ E' = E + \beta \sum_{i=1}^{N_p} \bar{w}_i \ln(\bar{w}_i) \]
  - parameterize to reflect constraints
- Structure identification