Artificial Intelligence

Introduction

Why ANN
- Some tasks can be done easily (effortlessly) by humans but are hard by conventional paradigms on Von Neumann machine with algorithmic approach
  - Pattern recognition (old friends, hand-written characters)
  - Content addressable recall
  - Approximate, common sense reasoning (driving, playing piano, baseball player)
- These tasks are often ill-defined, experience based, hard to apply logic

Introduction

Von Neumann machine
- One or a few high speed (ns) processors with considerable computing power
- One or a few shared high speed buses for communication
- Sequential memory access by address
- Problem-solving knowledge is separated from the computing component
- Hard to be adaptive

Human Brain
- Large # (10^{11}) of low speed processors (ms) with limited computing power
- Large # (10^{15}) of low speed connections
- Content addressable recall (CAM)
- Problem-solving knowledge resides in the connectivity of neurons
- Adaptation by changing the connectivity

Biological neural activity
- Each neuron has a body, an axon, and many dendrites
  - Can be in one of the two states: firing and rest.
  - Neuron fires if the total incoming stimulus exceeds the threshold
- Synapse: thin gap between axon of one neuron and dendrite of another.
  - Signal exchange
  - Synaptic strength/efficiency
Introduction

• **What is an (artificial) neural network**
  – A set of **nodes** (units, neurons, processing elements)
    • Each node has input and output
    • Each node performs a simple computation by its **node function**
  – **Weighted connections** between nodes
    • Connectivity gives the structure/architecture of the net
    • What can be computed by a NN is primarily determined by the connections and their weights
  – A very much simplified version of networks of neurons in animal nerve systems

---

ANN Neuron Models

• Each node has one or more inputs from other nodes, and one output to other nodes
  • Input/output values can be
    – Binary \([0, 1]\)
    – Bipolar \([-1, 1]\)
    – Continuous
  • All inputs to one node come in at the same time and remain activated until the output is produced
  • Weights associated with links
  • \(f(\text{net})\) is the node function \(\text{net} = \sum_{i=1}^{n} w_i x_i\) is most popular

---

Node Function

• **Identity function**: \( f(\text{net}) = \text{net} \).
• **Constant function**: \( f(\text{net}) = c \).
• **Step (threshold) function**
  \[
  f(\text{net}) = \begin{cases} 
  a & \text{if } \text{net} < c \\
  b & \text{if } \text{net} > c 
  \end{cases}
  \]
  where \(c\) is called the threshold
• **Ramp function**
  \[
  f(\text{net}) = \begin{cases} 
  a & \text{if } \text{net} \leq c \\
  b & \text{if } \text{net} \geq d \\
  a + \frac{(\text{net} - c)(d - a)}{d - c} & \text{otherwise}
  \end{cases}
  \]

---

Node Function

• **Sigmoid function**
  – S-shaped
  – Continuous and everywhere differentiable
  – Rotationally symmetric about some point \((\text{net} = c)\)
  – Asymptotically approach saturation points
  \[
  \lim_{\text{net} \to \infty} f(\text{net}) = 0 \quad \lim_{\text{net} \to -\infty} f(\text{net}) = b
  \]
  – Examples:
  \[
  f(\text{net}) = z + \frac{1}{1 + \exp(-x \cdot \text{net} + y)}
  \]
  \[
  f(\text{net}) = \tanh(x \cdot \text{net} - y) + z,
  \]
  When \(y = 0\) and \(z = 0\):
  \[
  a = 0, \quad b = 1, \quad c = 0.
  \]
  Larger \(x\) gives steeper curve
**Network Architecture**

- **Feedforward Networks**
  - A connection is allowed from a node in layer $i$ only to nodes in layer $i + 1$.
  - Most widely used architecture.

Conceptually, nodes at higher levels successively abstract features from preceding layers.

**Perceptrons**

- A simple perceptron
  - Structure:
    - Single output node with threshold function
    - $n$ input nodes with weights $w_j$, $i = 1 - n$
  - To classify input patterns into one of the two classes (depending on whether output = 0 or 1)
  - Example: input patterns: $(x_1, x_2)$
    - Two groups of input patterns
      - $(0, 0)$, $(0, 1)$, $(1, 0)$, $(-1, -1)$
      - $(2, 1)$, $(0, -2.5)$, $(1.6, -1.6)$
    - Can be separated by a line on the $(x_1, x_2)$ plane
    - Classification by a perceptron with $w_1 = 1$, $w_2 = -1$, threshold = 2

**Perceptrons**

- Linear separability
  - A set of $(2D)$ patterns $(x_1, x_2)$ of two classes is linearly separable if there exists a line on the $(x_1, x_2)$ plane
    - $w_0 + w_1 x_1 + w_2 x_2 = 0$
    - Separates all patterns of one class from the other class
  - A perceptron can be built with
    - 3 input $x_0 = 1, x_1, x_2$ with weights $w_0, w_1, w_2$
    - $n$ dimensional patterns $(x_1, \ldots, x_n)$
      - Hyperplane $w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n = 0$ dividing the space into two regions
  - Can we get the weights from a set of sample patterns?
    - If the problem is linearly separable, then YES (by perceptron learning)
Examples of linearly separable classes

- Logical AND function
  patterns (bipolar) decision boundary
  \[
  \begin{array}{ccc}
  x_1 & x_2 & \text{output} \\
  -1 & -1 & w_1 = 1 \\
  -1 & 1 & w_2 = 1 \\
  1 & -1 & w_0 = -1 \\
  1 & 1 & -1 + x_1 + x_2 = 0 \\
  \end{array}
  \]

- Logical OR function
  patterns (bipolar) decision boundary
  \[
  \begin{array}{ccc}
  x_1 & x_2 & \text{output} \\
  -1 & -1 & w_1 = 1 \\
  -1 & 1 & w_2 = 1 \\
  1 & -1 & w_0 = 1 \\
  1 & 1 & 1 + x_1 + x_2 = 0 \\
  \end{array}
  \]

**Perceptron Learning Algorithm**

1. Initialize weights and threshold:
   - Set \( w_i(t), (0 \leq i \leq n) \), to be the weight \( i \) at time \( t \), and \( \theta \) to be the threshold value in the output node. Set \( w_0 \) to be -\( \theta \), the bias, and \( x_0 \) to be always 1.
   - Set \( w(0) \) to small random values, thus initializing the weights and threshold.

2. Present input and desired output
   - Present input \( x_0, x_1, x_2, \ldots, x_n \) and desired output \( d(t) \)

3. Calculate the actual output:
   \[
   y(t) = f_\theta [w_0(t)x_0(t) + w_1(t)x_1(t) + \ldots + w_n(t)x_n(t)]
   \]

4. Adapts weights
   \[
   w_i(t+1) = w_i(t) + \eta[t \cdot d(t) - y(t)]x_i(t)
   \]
   where \( 0 < \eta \leq 1 \) is a positive gain function that controls the adaption rate.

- Steps 3 and 4 are repeated until the iteration error is less than a user-specified error threshold or a predetermined number of iterations have been completed.

**Perceptron Learning**

- Note:
  - It is a supervised learning
  - Learning occurs only when a sample input misclassified (error driven)

- Termination criteria: learning stops when all samples are correctly classified
  - Assuming the problem is linearly separable
  - Assuming the learning rate (\( \eta \)) is sufficiently small

Choice of learning rate:
- If \( \eta \) is too large:
  - existing weights are overtaken by \( \eta \cdot [d(t) - y(t)] \)
- If \( \eta \) is too small (\( \approx 0 \)): very slow to converge
  - Common choice: \( \eta = 0.1 \)

- Non-numeric input:
  - Different encoding schema
  - ex. Color = (red, blue, green, yellow). (0, 0, 1, 0) encodes “green”
Perceptron Learning Quality

- **Generalization**: can a trained perceptron correctly classify patterns not included in the training samples?
  - Common problem for many NN learning models
  - Depends on the quality of training samples selected.
  - Also to some extent depends on the learning rate and initial weights
  - How can we know the learning is ok?
    - Reserve a few samples for testing

-- XOR can be solved by a more complex network with hidden units

---

Linear Separability Again

- Examples of linearly inseparable classes
  - Logical XOR (exclusive OR) function
    - Patterns (bipolar) decision boundary
      \[
      \begin{align*}
      x_1 & \quad x_2 & \quad \text{output} \\
      -1 & \quad -1 & \quad -1 \\
      -1 & \quad 1 & \quad 1 \\
      1 & \quad -1 & \quad 1 \\
      1 & \quad 1 & \quad -1 \\
      \end{align*}
      \]

  - \(x\): class I (output = 1)
  - \(o\): class II (output = -1)

  No line can separate these two classes, as can be seen from the fact that the following linear inequality system has no solution:

  \[
  \begin{align*}
  w_0 - w_1 - w_2 & < 0 \quad \text{(1)} \\
  w_0 - w_1 + w_2 & \geq 0 \quad \text{(2)} \\
  w_0 + w_1 - w_2 & \geq 0 \quad \text{(3)} \\
  w_0 + w_1 + w_2 & < 0 \quad \text{(4)}
  \end{align*}
  \]

  because we have \(w_0 < 0\) from (1) and \(w_0 \geq 0\) from (2) + (3), which is a contradiction

---

Multilayer Network Learning Algorithm

```
function BACKPROP_LEARNING(examples, network) returns a neural network
inputs: examples, a set of examples, each with input vector x and output vector y
network, a multilayer network with L layers, weights \(W_{ij},\) activation function \(g\)
repeat
  for each \(e\) in examples do
    for each node \(j\) in the input layer do \(e_j \leftarrow z_j(e)\)
    for \(\ell = 2\) to \(L\) do
      \(i_j \leftarrow \sum_{i} W_{ij} o_i\)
      \(o_j \leftarrow g(i_j)\)
    for each node \(j\) at the output layer do
      \(\Delta_j \leftarrow g'(i_j) \sum_{i} W_{ij} \Delta_i\)
    for each node \(j\) in layer \(\ell + 1\) do
      \(W_{ij} \leftarrow W_{ij} + \alpha \times o_j \times \Delta_i\)
  until some stopping criterion is satisfied
return NEURAL-NET-HYPOTHESIS(network)
```

---

156  Ram Meshulam 2004
157  Ram Meshulam 2004
158  Ram Meshulam 2004
159  Ram Meshulam 2004
Sigmoid as activation function with x=3:
- \( g(\text{in}) = \frac{1}{1+e^{-3 \cdot \text{in}}} \)
- \( g'(\text{in}) = 3g(\text{in})(1-g(\text{in})) \)

Training Set
- Logical XOR (exclusive OR) function
  - \( x_1 \quad x_2 \quad \text{output} \)\
    - 0 0 0
    - 0 1 1
    - 1 0 1
    - 1 1 0
- Choose random weights
  - \( <w_{03},w_{04},w_{13},w_{23},w_{24},w_{65},w_{35},w_{45}> = <0.03,0.04,0.13,0.14,-0.23,-0.24,0.65,0.35,0.45> \)
- Learning rate: 0.1 for the hidden layers, 0.3 for the output layer

First Example
- Compute the outputs
  - \( a_0 = 1, a_1 = 0, a_2 = 0 \)
  - \( a_3 = g(1*0.03 + 0*0.13 + 0*-0.23) = 0.522 \)
  - \( a_4 = g(1*0.04 + 0*0.14 + 0*-0.24) = 0.530 \)
  - \( a_6 = 1, a_5 = g(0.65*1 + 0.35*0.522 + 0.45*0.530) = 0.961 \)
  - Calculate \( \Delta 5 = 3* g(1.0712)*(1-g(1.0712))*(0-0.961) = -0.108 \)
  - Calculate \( \Delta 6, \Delta 3, \Delta 4 \)
    - \( \Delta 6 = 3* g(1)*(1-g(1))*(0.65*-0.108) = -0.010 \)
    - \( \Delta 3 = 3* g(0.03)*(1-g(0.03))*(0.35*-0.108) = -0.028 \)
    - \( \Delta 4 = 3* g(0.04)*(1-g(0.04))*(0.45*-0.108) = -0.036 \)
- Update weights for the output layer
  - \( w_{65} = 0.65 + 0.3*1*-0.108 = 0.618 \)
  - \( w_{35} = 0.35 + 0.3*0.522*-0.108 = 0.333 \)
  - \( w_{45} = 0.45 + 0.3*0.530*-0.108 = 0.433 \)
First Example (cont)

- Calculate $\Delta 0, \Delta 1, \Delta 2$
  - $\Delta 0 = 3*g(1)*(1-g(1))*(0.03*-0.028 + 0.04*-0.036) = -0.001$
  - $\Delta 1 = 3*g(0)*(1-g(0))*(0.13*-0.028 + 0.14*-0.036) = -0.006$
  - $\Delta 2 = 3*g(0)*(1-g(0))*(-0.23*-0.028 + -0.24*-0.036) = 0.011$
- Update weights for the hidden layer
  - $w03 = 0.03 + 0.1*1*-0.028 = 0.027$
  - $w04 = 0.04 + 0.1*1*-0.036 = 0.036$
  - $w13 = 0.13 + 0.1*0*-0.028 = 0.13$
  - $w14 = 0.14 + 0.1*0*-0.036 = 0.14$
  - $w23 = -0.23 + 0.1*0*-0.028 = -0.23$
  - $w24 = -0.24 + 0.1*0*-0.036 = -0.24$

Second Example

- Compute the outputs
  - $a0 = 1, a1= 0, a2 = 1$
  - $a3 = g(1*0.027 + 0*0.13 + 1*-0.23) = 0.352$
  - $a4 = g(1*0.036 + 0*0.14 + 1*-0.24) = 0.352$
  - $a6 = 1, a5 = g(0.618*1 + 0.333*0.352 + 0.433*0.352) = 0.935$
- Calculate $\Delta 1 = 3*g(0.888)*(1-g(0.888))*(1-0.935) = 0.012$
- Calculate $\Delta 6, \Delta 3, \Delta 4$
  - $\Delta 6 = 3*g(1)*(1-g(1))*(0.618*0.012) = 0.001$
  - $\Delta 3 = 3*g(-0.203)*(1-g(-0.203))*(0.333*0.012) = 0.003$
  - $\Delta 4 = 3*g(-0.204)*(1-g(-0.204))*(0.433*0.012) = 0.004$
- Update weights for the output layer
  - $w03 = 0.027 + 0.1*1*0.003 = 0.027$
  - $w04 = 0.036 + 0.1*1*0.004 = 0.036$
  - $w13 = 0.13 + 0.1*0*0.003 = 0.13$
  - $w14 = 0.14 + 0.1*0*0.004 = 0.14$
  - $w23 = -0.23 + 0.1*1*0.003 = -0.23$
  - $w24 = -0.24 + 0.1*1*0.004 = -0.24$

Second Example (cont)

- Calculate $\Delta 0, \Delta 1, \Delta 2$
  - Skipped, we do not use them
- Update weights for the hidden layer
  - $w03 = 0.027 + 0.1*1*0.003 = 0.027$
  - $w04 = 0.036 + 0.1*1*0.004 = 0.036$
  - $w13 = 0.13 + 0.1*0*0.003 = 0.13$
  - $w14 = 0.14 + 0.1*0*0.004 = 0.14$
  - $w23 = -0.23 + 0.1*1*0.003 = -0.23$
  - $w24 = -0.24 + 0.1*1*0.004 = -0.24$

Summary

- Single layer nets have limited representation power (linear separability problem)
- Error driven seems a good way to train a net
- Multi-layer nets (or nets with non-linear hidden units) may overcome linear inseparability problem