Connections and symbols

AT1
Emmanuel Dupoux
• bref historique du connexionnisme et de l’IA symbolique
• les années 80: la confrontation ideologique: faux débat versus vraies questions
• Les nouvelles pistes
  – compositionalité et produit tensoriel
  – recursivité et systeme dynamique
  – (gradualité et modèles bayesiens)
• What is thinking? what kind of object can think?
  – thinking is manipulation of discrete symbols
  – computers (could be programmed to) think
  
  – thinking is performing parallel, distributed computations
  – neural networks (could be constructed to) think
Symbols
Cognition and symbols

- reasoning = computation in a formal calculus
  - Aristote, Leibniz (universal language + reasoning calculus), Boole (formal system for logical and set theoretic reasoning), Frege (logical program for mathematics), Peano, Russell, etc

- Formal system:
  - A finite set of symbols (i.e. the alphabet), that can be used for constructing formulas (i.e. finite strings of symbols).
  - A grammar, which tells how well-formed formulas (abbreviated wff) are constructed out of the symbols in the alphabet. It is usually required that there be a decision procedure for deciding whether a formula is well formed or not.
  - A set of axioms or axiom schemata: each axiom must be a wff.
  - A set of inference rules (going from wff to wff)

- Examples
  - first order propositional logic (A,B..C, ->)
  - second order logic (A,B, C, Vα, ->)
  - predicate logic (P(x))
Symboles: $x, y, z, \ldots = - ( )$

**Grammaire:** 
- **énoncé** : expression = expression
  - expression: ( expression )
  - | expression – expression
  - | variable
- variable : $x | y | z \ldots$

**Axiomes**

| (A) | $x = x - (y - y)$ |
| (B) | $x - (y - z) = z - (y - x)$ |

**Théorèmes**

| (1) | $x = x$ | (A), (A) |
| (2) | $x = x - (z - z)$ | (A) |
| (3) | $y - (x - z) = z - (x - y)$ | (B) |
| (4) | $y - (x - (z - z)) = z - z - (x - y)$ | (B) |
| (5) | $z - (z - z - (x - y)) = x - y - (z - z - z)$ | (B) |
| (6) | $y - y - (x - x - x) = y - y - (x - x - x)$ | (1) |
| (7) | $y - x = z - z - (x - y)$ | (2), (4) |
| (8) | $x - z - y = w - w - (y - (x - z))$ | (7) |
| (9) | $x - y - z = w - w - (z - (x - y))$ | (7) |
| (10) | $z - (y - x) = x - y - (z - z - z)$ | (7), (5) |
| (11) | $x - (y - y) = y - y - (x - x - x)$ | (7) |
| (12) | $x - y - z = w - w - (y - (x - z))$ | (3), (9) |
| (13) | $x - (y - z) = x - y - (x - z - z)$ | (B), (10) |
| (14) | $x - (y - y) = x - y - (y - y - y)$ | (13) |
| (15) | $x - y - z = x - z - y$ | (8), (12) |
| (16) | $z - z - (x - y) = z - (x - y) - z$ | (15) |
| (17) | $y - x = z - (x - y) - z$ | (7), (16) |
| (18) | $y - x = y - (x - z) - z$ | (3), (17) |
| (19) | $x = x - y - (y - y - y)$ | (A), (14) |
| (20) | $x = y - y - (x - x - x)$ | (A), (11) |
| (21) | $y - y - (x - x - x) = x$ | (20), (6) |

**Règles d’inférences**

- substitution (d’une
expression quelconque
pour toute occurrence
d’une variable)
- remplacement de a par
b partout si a=b

Quine (1934). A method for generating part of arithmetic without intuitive logic
Cognition and symbols (II)

- Thinking=computation over symbols
  - computability theory (1930-) Godel, Post, Kleene, Church, Turing, Markov

- Physical symbolic systems
  - Finite state automaton, pushdown automaton, Turing machines computational power
  - Power of automata:
    - Finite state automata
    - Pushdown automata
    - Turing machines with one tape, Turing machines with several tapes, lambda calculus (eg lisp), string rewriting system (eg production), recursive functions
  - Church thesis:
    - Turing machines are on the top of the hierarchy
theoretical background

• Formal linguistics
  – Miller & Chomsky (1963)
    • to each automaton, a class of languages
    • human languages are between context free and recursively enumerable
définitions

• langage formel
  – alphabet: ensemble fini de symboles: \( \Sigma = \{a, b, c, \ldots\} \)
    (ou \{chien, chat, le, une, chasse, etc\})
  – énoncé: séquence finie de symboles: eg: aacba
    (ou le.chien.chasse.le.chat)
  – ensemble de tous les énonces: \( \Sigma^* \) (infini)
  – langage: \( L \) sous-ensemble (potentiellement infini) de \( \Sigma^* \)
Alphabet

\[ \Sigma = \{a, b, c, d\} \]

Automate à états finis

Exemples

\[ a, b, c, ab, ba, bd, ac, bdddddd, cd, abd, acd, aab, aaaaaaabdddddddd, \ldots \in L ? \]
Automate à états finis

Exemples

Expression régulière

Alphabet

\[ \Sigma = \{ a, b, c, d \} \]

\[ a, b, c, ab, ba, bd, ac, bddddddd, cd, abd, acd, aab, aaaaaaabb, aaaaaaabbddddddd, \ldots \in L ? \]

\[ a^* [bc] d^* \]
Automate à pile

Exemples

b, c, ab, bd, bdddddd, cd, abd, acd, aabd, aaaaaaaaaabdddddddddd, etc
Automate à pile

Exemples
b, c, ab, bd, bdddddd, cd, abd, acd, aabd, aaaaaaabddddddd, etc

Grammaire algébrique (context free)

\[ a^n[bc]d^n \quad n \geq 0 \]

\[ E \rightarrow aEd \mid b \mid c \]
parsing: a small grammar of English

Grammar

\[ S \rightarrow NP \ VP \quad \text{I + want a morning flight} \]
\[ NP \rightarrow \text{Pronoun} \quad \text{I} \]
\[ \quad | \text{Proper-Noun} \quad \text{Los Angeles} \]
\[ \quad | \text{Det Nominal} \quad \text{a + flight} \]
\[ \text{Nominal} \rightarrow \text{Noun Nominal} \quad \text{morning + flight} \]
\[ \quad | \text{Noun} \quad \text{flights} \]
\[ VP \rightarrow \text{Verb} \quad \text{do} \]
\[ \quad | \text{Verb NP} \quad \text{want + a flight} \]
\[ \quad | \text{Verb NP PP} \quad \text{leave + Boston + in the morning} \]
\[ \quad | \text{Verb PP} \quad \text{leaving + on Thursday} \]
\[ PP \rightarrow \text{Preposition NP} \quad \text{from + Los Angeles} \]

Lexicon

\[ \text{Noun} \rightarrow \text{flights | breeze | trip | morning | ...} \]
\[ \quad | \text{Verb} \rightarrow \text{is | prefer | like | need | want | fly} \]
\[ \quad | \text{Adjective} \rightarrow \text{cheapest | non-stop | first | latest} \]
\[ \quad \quad | \text{other | direct | ...} \]
\[ \quad | \text{Pronoun} \rightarrow \text{me | I | you | it | ...} \]
\[ \quad | \text{Proper-Noun} \rightarrow \text{Alaska | Baltimore | Los Angeles} \]
\[ \quad \quad | \text{Chicago | United | American | ...} \]
\[ \quad | \text{Determiner} \rightarrow \text{the | a | an | this | these | that | ...} \]
\[ \quad | \text{Preposition} \rightarrow \text{from | to | on | near | ...} \]
\[ \quad | \text{Conjunction} \rightarrow \text{and | or | but | ...} \]
whether a given sentence is part of a given natural language (say English) often depends on the context. In linguistics, the use of formal languages to model natural languages is called generative grammar, since the language GENERATIVE GRAMMAR is defined by the set of possible sentences 'generated' by the grammar.

We conclude this section by way of summary with a quick formal description of a context free grammar and the language it generates. A context-free grammar has four parameters (technically 'is a 4-tuple'):

1. a set of non-terminal symbols (or 'variables') $N$
2. a set of terminal symbols $\Sigma$ (disjoint from $N$)
3. a set of productions $P$, each of the form $A \rightarrow \alpha$, where $A$ is a non-terminal and $\alpha$ is a string of symbols from the infinite set of strings $\Sigma^*$.  
4. a designated start symbol $S$  

A language is defined via the concept of derivation. One string derives another one if it can be rewritten as the second one via some series of rule applications. More formally, following Hopcroft and Ullman (1979), if $A \rightarrow \alpha$ is a production of $P$ and $\beta$ and $\gamma$ are any strings in the set $\Sigma^*$, then we say that $\beta A \gamma$ directly derives $\beta \alpha \gamma$, or $\beta A \gamma$ DIRECTLY DERIVES $\beta \alpha \gamma$. Derivation is a generalization of direct derivation. Let $\beta_1 \beta_2 \ldots \beta_m$ be strings in $\Sigma^*$, such that $\beta_1 \beta_2 \ldots \beta_m \beta_1 \beta_2 \ldots \beta_m \beta_1 \beta_2 \ldots \beta_m$ (9.8)  

We say that $\beta_1 \ldots \beta_m$ derives $\beta_1 \ldots \beta_m$, or $\beta_1 \ldots \beta_m \beta_1 \ldots \beta_m$.  

DERIVES
• note on dependencies in human languages

a. that we₁ let₁ the children₂ help₂ Hans₃ paint₃ the house₃.

b. daß wir₁ die Kinder₂ dem Hans₃ das Haus₃ streichen₃ helfen₂ lassen₁.

c. das mer₁ d’chind₂ em Hans₃ es huus₃ lönd₁ hälf₂ aastriiche₃.

• a and b can be handled by a push down automaton, but not c
Machine de Turing

Grammaire récursivement énumérable

Système de production

\[ a^n[bc]d^n e^n \quad n \geq 0 \]

\[ (\mathcal{N} \cup \Sigma)^* N(\mathcal{N} \cup \Sigma)^* \rightarrow (\mathcal{N} \cup \Sigma)^* \]
quelle classe d’automate pour les langues humaines?

– Miller & Chomsky (1963)

• à chaque classe d’automates, une classe de langages

• les langages humains sont entre context free et récursivement énumérables (en ce qui concerne la syntaxe) et régulières ou sous-régulières en ce qui concerne la phonologie
Minimalist grammars,  
Dependency grammars,  
Tree adjoining grammars,  
categorical grammars  
… etc…
Cognition and symbols (III)

- Formal linguistics
  - Miller & Chomsky (1963)
    - to each automaton, a class of languages
    - human languages are between context free and recursively enumerable

- Symbolic AI (1956)
  - 1980: prolog, expert systems
  - SOAR, ACT*
symbolic processing

• Newell (1980) articulated the role of the mathematical theory of *symbolic* processing.
  – Cognition involves the manipulation of symbols – analogous to words, concepts, schema, etc.
  – What are symbols?
    • Roughly, it’s like the values of a categorical variable (male, female, red, blue, dog, cat).
    • Operators on those symbols would then be things like “is-a” “a-kind-of” “purpose” “shape” “part-of” “object”
• E.g. recognize a red apple

input = symbol(s) -> algorithms who work on input -> output = more symbol(s)

Input:                      Program:

  Red(X)                      if (Orange(X) & Round(X) ... ) then Orange(X)
  Round(X)                   if (Red(X) & Round(X) ... ) then Apple(X)
  ...                       ...

Output:                     Graphical:

  Apple(X)                   Apple(X)

             Red(X) & Round(X) ...

Color(X) -> Red(X) or Orange(X) Shape(X) -> Round(X) or ...
• ACT-R (Anderson 2004)
• Cyc (openCyc)
  – 200k terms, 2m beliefs
• Never Ending Language Learning
  – since 2010 50m candidate beliefs
Connections
McCulloch & Pitts (1943)

• **Neural networks as computing devices**
  – What logical operations could neurons compute?

• **Five assumptions based on then-current knowledge of neurons**
  – 1. The activity of a neuron is “all-or-none” (binary coding)
  – 2. Each neuron has a fixed threshold on the required number of synapses that need to be excited before the neuron itself will be excited. Weights are identical.
  – 3. Synaptic action causes a time delay before firing.
  – 4. Inhibition is absolute.
  – 5. The physical structure of a network of neurons doesn’t change with time; connections and their strengths are static.
McCulloch/Pitts neurons

• McCulloch/Pitts neurons can then be used to compute any (finite) logical function

• BUT, McCulloch/Pitts networks can’t learn.
Hebb (1949)

- The first rule for self-organized learning
- Hebb recognized the existence of feedforward, long range lateral, and feedback connections
- These cortical circuits admit self-sustaining activity that reverberate in « cell assemblies »
- synapses are the fundamental computational and learning unit
- activity-dependent synaptic plasticity as a basic operation

Learning in a Hebbian network

• “When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some grown process or metabolic change takes place in one or both cells such that A’s efficiency, as one of the cells firing B, is increased.”

LT Potentiation (Bliss & Lomo, 1973; Kelso et al, 1986)
LT Depression (Markram et al. 1997)
The “Hebb rule”

\[ \Delta w_{ij} = \eta a_i a_j \]

- where \( a \)'s are activation values (-1 or 1), and \( \eta \) is a learning rate parameter.
- Equation is applied until weights “saturate” (typically at 1) and do not keep increasing as inputs are presented.

Hebbian learning finds correlations between features in the environment
- Features that co-occur will grow strong positive weights, those that do not occur together will have grow negative weights, random pairing produces zero weights

\[ \text{-> what is the associated computation?} \]
The perceptron
(Rosenblatt, 1958, 1962)

- First model for learning with a teacher (supervised learning)
- McCulloch-Pitts neurons (linear-threshold) with connections that can be modified by learning

\[ y = \begin{cases} 
1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\
-1 & \text{otherwise} 
\end{cases} \]

Perceptron Learning Rule

• Start with random connections \( w_i \)
• Error-driven learning rule (delta rule):
  \[ \Delta w_i = \eta (t - y) x_i \]
  \( t \) is the target value (given by the teacher)
  \( y \) is the perceptron output
  \( \eta \) is a small constant (e.g. 0.1) called learning rate

• If the output is correct (\( t=y \)) the weights \( w_i \) are not changed
• If the output is incorrect (\( t \neq y \)) the weights \( w_i \) are changed such that the output of the perceptron for the new weights is closer to \( t \) (error decreases).
• The algorithm converges to the correct classification
  • if the training data is linearly separable
  • and \( \eta \) is sufficiently small
Example: learning the AND

\[ W = 0 \]

\[ \begin{array}{c|ccc}
A & B & \text{Output} \\
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array} \]

- Initial weights: 0, \( n=0.25 \)
Example: learning the AND

\[
\begin{align*}
W &= 0 \\
x_1 &= 1 \\
x_2 &= 1 \\
y &= 0 \\
t &= 1
\end{align*}
\]

<table>
<thead>
<tr>
<th>AND</th>
<th>A</th>
<th>B</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Example: learning the AND

$$y = x_1^2 = 1$$

$$x_1 = 1$$

$$x_2 = 1$$

$$W = 0.25$$

$$W = 0.25$$

$$W = 0.25$$

AND

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Output</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>

$t = 1$

error = 1
Example: learning the AND

\[ y = x_1^2 = 0 \quad x_1 = 1 \quad W = 0.25 \]

\[ y = 1 \quad t = 0 \quad \text{error} = -1 \]

\[ x_2 = 0 \quad W = 0.25 \]

<table>
<thead>
<tr>
<th>AND</th>
<th>A</th>
<th>B</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Example: learning the AND

\[ y = \begin{cases} 1 & \text{if } x_1^2 = 0 \text{ and } x_1 = 1 \text{ and } W = 0 \\ 0 & \text{otherwise} \end{cases} \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Output</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>

W = 0.25

\[ t = 0 \]

\[ \text{error} = -1 \]
Example: learning the AND

\[
\begin{array}{ccc}
\text{AND} & A & B & \text{Output} \\
0 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 \\
\end{array}
\]

\[
y = \begin{cases} 
1 & \text{if } x_1 = 0 \\
0 & \text{if } x_2 = 1 \\
0 & \text{otherwise}
\end{cases}
\]

\[
w = \begin{cases} 
0 & \text{if } x_1 = 0 \\
0.25 & \text{if } x_2 = 1 \\
0 & \text{otherwise}
\end{cases}
\]
Example: learning the AND

\[ y = \begin{cases} 1 & \text{if } x_1^2 = 1 \\ 0 & \text{if } x_1^1 = 0 \end{cases} \]

\[ W = -0.25 \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Output</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>

\[ t = 0 \]

error = -1
Example: learning the AND

\[ y = \begin{cases} 1 & x_2 = 1 \\ 0 & x_2 = 0 \end{cases} \]

\[ W = -0.25 \]

\[ W = 0 \]

\[ W = 0 \]

\[ t = 0 \]

\[ \text{error} = -1 \]

... and so on and so forth ...

\[
\begin{array}{c|cc|c}
\text{A} & \text{B} & \text{Output} \\
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]
Example: learning the AND

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Output</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>

Final weights

W = -0.25
W = 0.25
W = 0.25
• Perceptron is doing something similar to linear discriminant analysis, binomial regression (or Bayes classification when the distributions are gaussian)
• Decision boundary is a hyperplane when more than 2 inputs
Generalization: Multiple outputs

- similar multinomial LDA or multinomial regression
Minsky & Papert (1969)

• Presented a formal analysis of the properties of perceptrons and revealed several fundamental limitations.

• Limitations
  – Can’t learn nonlinearly separable problems like the XOR
  – *More…*
Decision Boundary of a Perceptron

- **Linearly separable**
- **Non-Linearly separable**

![Decision Boundary Diagram]

**Figure 1.10** Perceptron with the double-moon set at distance $d = 4$. 

**Mathematical Expression:**

- $x_1 \text{ OR } x_2$
- $x_1 \text{ AND } x_2$
- $x_1 \text{ XOR } x_2$
Minsky & Papert cont.

• Limitations
  – So….can’t learn nonlinearly separable problems like the XOR
  – Although including “hidden layers” allows one to hand-design a network that can represent XOR and related problems, they showed that the perceptron learning rule can’t learn the required weights.
  – They also showed that even those functions that can be learned by perceptron rule learning may require huge amounts of learning time
Consequences of Minsky & Papert’s analysis

• nearly caused the death of this field.
• Subsequent research was not mainstream
The revival: the 80s

- multilayered feedforward networks
  - Generalization of the Delta Rule: backpropagation of error
  - learning of distributed representations

- Symmetric recurrent networks
  - winner take all
  - autoassociation

The revival I: the Multi-Layer Perceptron

Activation functions

- step function
- sigmoid function
The backpropagation learning rule (I)

\[ e_i = (t_i - o_i) \]
\[ \Delta w_{ij} = o_i (1-o_i) \eta e_i h_j \]

weights: \( w_{ij} \)

target output \( t_1 \ t_2 \ t_3 \rightarrow e_1 \ e_2 \ e_3 \) error

hidden output \( h_1 \ h_2 \ h_3 \ h_n \)

The backpropagation learning rule (II)

The network consists of an input layer, one hidden layer, and an output layer. The hidden output is calculated as:

\[ e'_j = \sum_i w_{ij} e_i \]

The change in the weights is calculated as:

\[ \Delta w'_{jk} = h_i (1-h_i) \eta e'_j x_k \]

Where:
- \( e'_j \) is the error in the output layer.
- \( w_{ij} \) are the weights connecting the input layer to the hidden layer.
- \( h_i \) is the output of the hidden layer.
- \( x_k \) is the input.
- \( \eta \) is the learning rate.
Expressive Capabilities of MLP

Boolean functions
• Every boolean function can be represented by network with single hidden layer
• But might require exponential (in number of inputs) hidden units

Continuous functions
• Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer [Cybenko 1989, Hornik 1989]
• Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988]
## Different Non-Linearity Separable Problems

<table>
<thead>
<tr>
<th>Structure</th>
<th>Types of Decision Regions</th>
<th>Exclusive-OR Problem</th>
<th>Classes with Meshed regions</th>
<th>Most General Region Shapes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-Layer</td>
<td>Half Plane Bounded By Hyperplane</td>
<td>A</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Two-Layer</td>
<td>Convex Open Or Closed Regions</td>
<td>A</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Three-Layer</td>
<td>Arbitrary (Complexity Limited by No. of Nodes)</td>
<td>A</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B</td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>
Example: Neural network for OCR

- feedforward network
- trained using Back-propagation
Example: ALVINN

Drives 70 mph on a public highway

30 outputs for steering

4 hidden units

30x32 pixels as inputs

30x32 weights into one out of four hidden unit

https://www.youtube.com/watch?v=Rq_kBI0UZmY
http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html
Current trends

• very deep hierarchical networks
  – trained in a heavily supervised fashion
  – but, seem to correlate with human performance and monkey IT neurons


Google net, Szegedy et al, 2014

1000 categories
100,000 test
1M training
Google net: 6.67% error
Humans: 5%
The revival II: Symmetric dynamical networks

discrete dynamical system: $Y_{t+1} = F(Y_t)$
where $Y_t$ is the state of the system at time $t$

eg: a single neuron

\[ y_{t+1} = ay_t + x_t \]

very different behaviors as a function of the value of $a$
  - $a=1$: memory
  - $0<a<1$: leaky integration
  - $a=-1$: oscillator
  - $-1<a<0$: damped oscillator
\[ y_{t+1} = a \cdot y_t + x_t \]

\[ a = 1 \]

<table>
<thead>
<tr>
<th>( t )</th>
<th>Input (( X_t ))</th>
<th>Output (( Y_{t+1} ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>0.5</td>
<td>1.5</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1.5</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1.5</td>
</tr>
</tbody>
</table>

\( \rightarrow \) memory, integrator (counter)
\[ y_{t+1} = a \cdot y_t + x_t \]

\( a = 0 \) → leaky integrator
→ attractors (depending on input \( \frac{x}{1-a} \))

\( a = 0 \) → copy (identity) operation
\[ y_{t+1} = a \cdot y_t + x_t \]

- **a = -1**
  - Oscillator
- **a = -0.8**
  - Damped oscillator
- **a = 2**
  - Saturating memory
- **a = -2**
  - Epileptic oscillator

Input vs. Output graphs for each value of **a**.
The revival II: Symmetric dynamical networks

• A very simple example: a winner-take all network

• at each step:
  – compute the total input for all units
  – compute the output for all units
  – iterate

inputs

excitatory weights

inhibitory weights

behaviors:
memorise compute the max
The revival II: Symmetric dynamical networks

• Boltzmann machine (Hinton & Sejnovsky, 1983)
  – neurons: binary (-1, 1)
  – network: symmetric weights
  – update: stochastic (1 with probability $1/ (1+e^{-\text{sum}(\text{inputs})})$
  – dynamics: global energy minimization, basins of attraction
  – learning rule: tries to reproduce the distribution of its inputs
    (stochastically learns the basins of attraction)

• Hopfield network (Hopfield 1982)
  – deterministic variant (‘temperature’=0)
  – learning rule becomes Hebb Rule.
  – performs pattern completion, content-addressable memory
The revival III: McClelland & Rumelhart’s (1986) Parallel Distributed Processing

- Basic mechanisms
  - feature discovery and competitive learning
  - dynamical system and harmony theory
  - learning in Boltzmann machines
  - internal representation through backpropagation
- Formal analysis
  - linear algebra
  - activation functions
  - delta rule
- Psychological Applications
  - schemata and sequential through processes
  - speech perception (TRACE)
  - blackboard model of reading
  - learning and memory
  - learning past tense of English verbs
  - sentence processing: assigning roles to constituents
- Biological mechanisms
  - anatomy of cortex
  - place recognition and goal location
  - neural plasticity and critical period
  - amnesia and distributed memory
Applications: early models of lexical access

- Morton (1969) Logogen theory
- Forster (1976) Serial search


a new PDP model

- Elman & McClelland (1986)

---

The revival III: McClelland & Rumelhart’s (1986) Parallel Distributed Processing

• Cognition involves the spreading of activation, relaxation, statistical correlation.
• Represents a method for how symbolic systems might be implemented
  – Hypothesized that apparently symbolic processing is an emergent property of subsymbolic operations.
• Advantages
  – Fault tolerance & graceful degradation
  – Can be used to model learning
  – More naturally capture nonlinear relationships
  – Fuzzy information retrieval
  – bridges the gap with real neural processing
The critique: Pinker & Mehler (1988)

- Lachter & Bever: connectionist theories are a return to associationism (Chomsky vs Skinner revisited)
- Pinker & Prince: connectionist models of the capacity to derive the past tense of English verbs is inadequate
  - rules: wug → wugged
  - exceptions: put -> put, go->went, dig->dug
- Fodor & Pylyshyn: connectionist theories are inadequate models of language and thought
walk

walked

FIGURE 1. The basic structure of the model.

FIGURE 5. The pattern associator net-

Response strengths for the high-

FIGURE 4. The so-called U-

Rumelhart & McClelland (1986)
The critique: Pinker & Mehler (1988)

• **Lachter & Bever**: connectionist theories are a return to associationism (Chomsky vs Skinner revisited)

• **Pinker & Prince**: connectionist models of the capacity to derive the past tense of English verbs is inadequate
  - rules: wug $\rightarrow$ wugged
  - exceptions: put $\rightarrow$ put, go$\rightarrow$went, dig$\rightarrow$dug

• **Fodor & Pylyshyn**: connectionist theories are inadequate models of language and thought
Fodor & Pylyshyn

- Position of the problem: classical theories vs connectionnism
  - Agree:
    - both classical theories & connectionism are *representationists* (they assign some 'meaning' to the elements – symbols or nodes)
  - Disagree
    - classical theory encode structural relationships and processes (eg, constituents, variables, rules)
    - connectionists only encode causal relationships and processes (x causes y to fire)
- Arguments against connectionnist systems: mental representation and processes are structure sensitive
  - combinatorial semantics
    - semantics of « J. loves M. » derived from semantics of « J. », « loves » and « M. »
  - productivity
    - the list of thoughts/sentences is not finite (I can construct new thoughts with old ones)
  - systematicity
    - I construct them in a systematic way
    - eg: « x loves M. » (where x can be any proper noun)
    - eg: If I can think « J. loves M. », I can think « M. loves J. »
  - recursivity & constituent structure:
    - If I can think « P. thinks that M. is nice » I can think « J thinks that P thinks that M is nice »
-> connectionist systems have none of the above properties
Fodor & Pylyshyn (cont)

- Objections to symbolic/classical systems
  - rapidity of cognitive processes/neural speed
  - difficulty of pattern recognition/content based retrieval in conventional architectures
  - committed to rule vs exception dichotomy
  - inadequate for intuitive/nonverbal behavior
  - acutely sensitive to damage/noise (vs graceful degradation)
  - storage in classical systems is passive
  - inadequate account of gradual/frequency based application of rules
  - inadequate account of nondeterminism
  - no account of neuroscience
  → none of these arguments are valid or relevant

- CONCLUSIONS
  1. current connectionist theories are inadequate
  2. if they were to be made adequate they would be mere implementation of classical architecture
in brief

• The Fodor & Pylyshyn challenge:
  – (current) connectionist architectures fail to capture complex behaviors
  – (future) connectionist architectures are ‘mere’ implementation of symbolic architectures
Elman

- structure of the paper
  - representing time
  - SRN architecture
  - xor through time
  - badiiguuu
  - word segmentation (15 words)
  - part of speech (13 categories, 29 words, 15 sentence templates)
Backprop applied

Figure 2. A simple recurrent network in which octivotions are copied from hidden layer to context layer on a one-for-one basis, with fixed weight of 1.0. Dotted lines represent trainable connections. Input, and also the previous internal state of some desired output. Because the patterns on the hidden units are saved as context, the hidden units must accomplish this mapping and at the same time develop representations which are useful encodings of the temporal properties of the sequential input. Thus, the internal representations that develop are sensitive to temporal context; the effect of time is implicit in these internal states. Note, however, that these representations of temporal context need not be literal. They represent a memory which is highly task- and stimulus-dependent.

Figure 7. Hierarchical cluster diagram of hidden unit activation vectors in simple sentence prediction task. Labels indicate the inputs which produced the hidden unit vectors: inputs were presented in context, and the hidden unit vectors averaged across multiple contexts. Several points should be emphasized. First, the category structure appears to be hierarchical. Thus, "dragons" are large animals, but also members of the class \[ \text{[human, animate]} \]. The hierarchical interpretation is achieved through the way in which the spatial relations (of the representations) are organized. Representations that are near one another in the representational space form classes, while higher level categories correspond to larger and more general regions of this space. Second, it is also true that the hierarchy is "soft" and implicit. While some categories may be qualitatively distinct (i.e., very far from each other...
• extensions of Elman’s SRN
  – computational capacity of SRN

  – reservoir computing
    • http://reservoir-computing.org
structure of Smolensky
  - representing structures by fillers and roles
    • examples: trees, lists, etc
  - tensor products and filler/role binding (definition)
    • local, semilocal and distributed
  - unbinding (exact and selfadressed)
  - capacity and graceful saturation
  - continuous and infinite structures
  - binding and unbinding networks
  - analogy between binding units and hebb weights
  - example of a stack
  - structured roles
the Smolensky response: tensor products

Paul loves Mary -> loves(Paul, Mary)
pred = loves, arg1 = Paul, arg2 = Mary
pred*loves + arg1*Paul + arg2*Mary

Paul loves Mary
Mary is loved by Paul

Mary loves Paul
Paul is loved by Mary

→ Problem of tensor product representations: exponential with sentence complexity

• extensions:
  – implementation of a phonological theory (Optimality Theory) in a tensor product network with energy relaxation
    • see the Harmonic Mind (Smolensky & Legendre)
  – Escaping the explosion in nb of neurons: holographic reduced representations
    • define A * B as an operation that preserves the dimensions (eg xor, circular convolution)
Conclusions

• What about the F&P Challenge?
  – tensor products are an interesting implementation/alternative to symbolic systems
  – recurrent networks could also be an alternative, but much less understood

• The hidden debate
  – innate vs learner structures (to be continued….)
Conclusions

• empirical impact of the debate
  – past tense in English
    • rule: play->played, fax->faxed
    • exceptions: sing->sang, put->put
  • Pinker & Prince (1988)
  • procedural vs declarative memory (Ullman et al, 1997; Pinker & Ullman, 2002)

Conclusions

• empirical impact of the debate (cont)
  – statistical learning vs algebraic learning in infants

  – exemplar-based versus abstract representations
    • object recognition (Biederman & Gerhardstein, 1993),
      face recognition, speech recognition (eg Goldinger, 1988; Johnson 1997, Pierrehumbert 2001)
Extensions

• computational reduction: finding the right architecture

• other connectionist architectures
  – Kohonen’s maps (competitive learning) (Kohonen, 1982)
  – Adaptive Resonance Theory (Grossberg, 1976)
  – Reinforcement learning (Barto, Sutton, Anderson. 1983)

• other computational frameworks
  – Probabilistic/Bayesian frameworks
  – Predictive Coding/Free Energy
Papiers

• Fodor & Pylyshyn (1988) Connectionism and cognitive architecture: a critical analysis
• Elman (1990) Finding structure in time
• Smolensky (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems

• Marcus (1998). Rethinking eliminative connectionism
• McClelland (2009). The place of modeling in Cognitive Science
Fodor & Pylyshyn

• Position of the problem: classical theories vs connectionism
  – Agree:
    • both classical theories & connectionism are *representationists* (they assign some 'meaning' to the elements – symbols or nodes)
  – Disagree
    • classical theory encode structural relationships and processes (eg, constituents, variables, rules)
    • connectionnists only encode causal relationships and processes (x causes y to fire)

• Arguments against connectionnist systems: mental representation and processes are structure sensitive
  – combinatorial semantics
    • semantics of « J. loves M. » derived from semantics of « J. », « loves » and « M. »
  – productivity
    • the list of thoughts/sentences is not finite (I can construct new thoughts with old ones)
  – systematicity
    • I construct them in a systematic way
    • eg: « x loves M. » (where x can be any proper noun)
    • eg: If I can think « J. loves M. », I can think « M. loves J. »
  – recursivity & constituent structure:
    • If I can think « P. thinks that M. is nice » I can think « J thinks that P thinks that M is nice »
-> connectionist systems have none of the above properties
• Objections to symbolic/classical systems
  – rapidity of cognitive processes/neural speed
  – difficulty of pattern recognition/content based retrieval in conventional architectures
  – committed to rule vs exception dichotomy
  – inadequate for intuitive/nonverbal behavior
  – acutely sensitive to damage/noise (vs graceful degradation)
  – storage in classical systems is passive
  – inadequate account of gradual/frequency based application of rules
  – inadequate account of nondeterminism
  – no account of neuroscience
  → none of these arguments are valid or relevant

• CONCLUSIONS
  1. current connectionist theories are inadequate
  2. if they were to be made adequate they would be mere implementation of classical architecture
Questions

• les arguments de Fodor contre les modèles connectionnistes sont-ils valides

• les réponses de Fodor aux arguments des connectionnistes sont-elles pertinentes

• que penser de la première conclusion (les modèles connectionnistes sont inadéquats comme modèles de la pensée et du langage)

• que penser de la seconde conclusion (les modèles connectionnistes qui sont adéquats ne sont que des implémentations des modèles classiques)
basic biblio