Neural Networks and the Evolution of Modularity
MODULARITY WUT?
Quick Review Question

Reverse Complement Problem

• **Input:** A DNA string $s$.
• **Output:** The reverse complement of $s$.

STOP: How would you write code to solve this?
ReverseComplement(s)
return Reverse(Complement(s))

STOP: What does it mean for code to be “modular”?
Modularity is everywhere in biology
We already know that modularity occurs in biological networks

The “network motifs” that we saw in TF networks are their own form of modularity.
Modularity in Graphs

Modular

Non-modular

STOP: What should it mean for a graph to be “modular”?
STOP: What should it mean for a graph to be “modular”?

Answer: It should divide into subgraphs so that two nodes from one subgraph are more likely to be connected than two nodes from different subgraphs.
STOP: Is our ReverseComplement() function the best way to reverse complement a string?

ReverseComplement(s)
    return Reverse(Complement(s))
ReverseComplement(s):
    revComp = ""

    complementMap = {
        'A': 'T',
        'T': 'A',
        'C': 'G',
        'G': 'C'
    }

    for i = Length(DNAString) - 1 to 0
        currentChar = DNAString[i]
        complementChar = complementMap[currentChar]
        revComp = revComp + ComplementChar

    return revComp
Modular code is good practice, but optimized code can be non-modular

Here is some HTML source code from google.com.
Much of biology is hyper-optimized …

Biology is largely solved. DNA is the source code for our bodies. Now that gene sequencing is easy, we just have to read it.

It’s not just “source code.” There’s a ton of feedback and external processing.

But even if it were, DNA is the result of the most aggressive optimization process in the universe, running in parallel at every level, in every living thing, for four billion years.

It’s still just code.

OK, try opening Google.com and clicking “view source.”

OK, I—...Oh my god. That’s just a few years of optimization by Google devs. DNA is thousands of times longer and way, way worse.

Wow, biology is impossible.

https://xkcd.com/1605/
… and yet modularity in some contexts must be worth preserving

Although modularity is important to many biological processes, no one built a model in which modularity spontaneously evolved until 2005.

https://www.pnas.org › content

Spontaneous evolution of modularity and network motifs | PNAS

by N Kashtan · 2005 · Cited by 899 — Nadav Kashtan and Uri Alon ... To understand the origin of modularity and network motifs in biology one has to understand how these features ...
MCCULLOCH-PITTS NEURONS: THE HUMBLE FOUNDATIONS OF AI
Neurons form a network of cells exchanging information

Hooray for interdisciplinary research

A logical calculus of the ideas immanent in nervous activity
WS McCulloch, W Pitts - The bulletin of mathematical biophysics, 1943 - Springer
Because of the “all-or-none” character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical ...

Warren McCulloch

Walter Pitts
A McCulloch-Pitts (MP) neuron takes as input \( n \) binary variables \( x_1, \ldots, x_n \). For a threshold \( \theta \), it fires (returns 1) if \( x_1 + \ldots + x_n \geq \theta \); otherwise, it returns 0.

**Example:** At right is an MP neuron for \( n = 2 \) and \( \theta = 2 \).
McCulloch-Pitts Neurons

**Example:** And here is the MP neuron for $n = 2$ and $\theta = 1$.

Example table:

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_1 + x_2$</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
**McCulloch-Pitts Neurons**

**Example:** And here is the MP neuron for \( n = 2 \) and \( \theta = 1 \).

**STOP:** Do these neurons remind you of anything?

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_1 + x_2 )</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

© 2024 Phillip Compeau
Example: And here is the MP neuron for $n = 2$ and $\theta = 1$.

STOP: Do these neurons remind you of anything?

Answer: The output is just $x_1 \lor x_2$. 
And the output of the MP neuron when $\theta = 2$ is $x_1 \land x_2$.

We say that an MP neuron represents a truth table if the inputs and outputs of the neuron and the truth table are the same.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_1 + x_2$</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Exercise: The AND of $n$ input variables returns true if all of the input variables are true, and false otherwise; the OR of $n$ input variables returns true if at least one of them is true, and false if they are all false. Construct MP neurons representing the AND and OR of $n$ binary input variables.
An Even Simpler Logical Connective: \textbf{NOT}

Here is a truth table representing the logical connective \textbf{NOT}.

\begin{array}{c|c|c}
\hline
x_1 & \sim x_1 \\
\hline
\text{true} & \text{false} \\
\text{false} & \text{true} \\
\hline
\end{array}
An Even Simpler Logical Connective: NOT

Here is a truth table representing the logical connective NOT.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$\sim x_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

Theorem: There is no McCulloch-Pitts neuron representing NOT.
An Even Simpler Logical Connective: NOT

Here is a truth table representing the logical connective NOT.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$\sim x_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

**Theorem:** There is no McCulloch-Pitts neuron representing NOT.

**Proof:** Assume that there is such an MP neuron with one input variable $x_1$. 
An Even Simpler Logical Connective:  
NOT

Here is a truth table representing the logical connective NOT.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$\sim x_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

Theorem: There is no McCulloch-Pitts neuron representing NOT.

Proof: Assume that there is such an MP neuron with one input variable $x_1$. There must be some threshold $\theta$ such that when $x_1 = 1$, $x_1 < \theta$, and when $x_1 = 0$, $x_1 \geq \theta$. In other words, $1 < \theta \leq 0$, a contradiction. □
FROM MCCULLOCH-PITTS NEURONS TO PERCEPTRONS
Perceptrons Generalize MP Neurons

Perceptron: A neuron having a threshold $\theta$ and constants $w_1, w_2, \ldots, w_n$, which fires if and only if $w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_n \cdot x_n \geq \theta$. 

© 2024 Phillip Compeau
Perceptron: A neuron having a threshold $\theta$ and constants $w_1, w_2, \ldots, w_n$, which fires if and only if $w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_n \cdot x_n \geq \theta$.

STOP: Why does a perceptron generalize the MP neuron?
**Perceptron**: A neuron having a threshold $\theta$ and constants $w_1, w_2, \ldots, w_n$, which fires if and only if $w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_n \cdot x_n \geq \theta$.

**STOP**: Why does a perceptron generalize the MP neuron?

**Answer**: An MP neuron is a perceptron with all weights $w_i$ equal to 1.
Perceptrons Generalize MP Neurons

**Perceptron:** A neuron having a threshold $\theta$ and constants $w_1, w_2, \ldots, w_n$, which fires if and only if $w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_n \cdot x_n \geq \theta$.

Although an MP neuron cannot represent NOT, here is a perceptron representing NOT.

<table>
<thead>
<tr>
<th>Input Variable $x_1$</th>
<th>$-x_1$</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

© 2024 Phillip Compeau
Consider the ambiguity of the word “or”

“Would you like ketchup or mustard with your hot dog?”

“Would you like to visit the beach or the mountains on vacation?”
Consider the ambiguity of the word “or”

“Would you like ketchup or mustard with your hot dog?”

“Would you like to visit the beach or the mountains on vacation?”

STOP: What is the difference in “or” in these two questions?
Consider the ambiguity of the word “or”

“Would you like ketchup or mustard with your hot dog?”

“Would you like to visit the beach or the mountains on vacation?”

**STOP:** What is the difference in “or” in these two questions?

**Answer:** The first question implies that *both* options are possible (“and/or”).
Introducing XOR

**Exclusive or (XOR):** \( x_1 \uparrow x_2 \) is **true** precisely when exactly one of \( x_1 \) and \( x_2 \) is **true** (i.e., when \( x_1 \neq x_2 \)).

\[
\begin{array}{cccc}
  x_1 & x_2 & x_1 \lor x_2 & x_1 \uparrow x_2 \\
  \text{true} & \text{true} & \text{true} & \text{false} \\
  \text{true} & \text{false} & \text{true} & \text{true} \\
  \text{false} & \text{true} & \text{true} & \text{true} \\
  \text{false} & \text{false} & \text{false} & \text{false} \\
\end{array}
\]
Introducing XOR

**Exclusive or (XOR):** \( x_1 \oplus x_2 \) is **true** precisely when exactly one of \( x_1 \) and \( x_2 \) is **true** (i.e., when \( x_1 \neq x_2 \)).

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_1 \lor x_2 )</th>
<th>( x_1 \lor x_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

**Exercise:** Find a perceptron that models \( x_1 \oplus x_2 \).
Perceptrons have limits too

**Theorem:** There is no perceptron representing XOR.

**Proof:** Assume there is, so there must be constants $w_1, w_2$, such that

- when $x_1 = x_2$, $w_1 \cdot x_1 + w_2 \cdot x_2 < \theta$
- when $x_1 \neq x_2$, $w_1 \cdot x_1 + w_2 \cdot x_2 \geq \theta$
Perceptrons have limits too

**Theorem:** There is no perceptron representing XOR.

**Proof:** When $x_1 = x_2$, the neuron doesn’t fire, and

$$w_1 \cdot 0 + w_2 \cdot 0 = 0 < \theta$$

$$w_1 \cdot 1 + w_2 \cdot 1 = w_1 + w_2 < \theta$$
Theorem: There is no perceptron representing XOR.

Proof: When \( x_1 \neq x_2 \), the neuron fires, and

\[
\begin{align*}
w_1 \cdot 1 + w_2 \cdot 0 &= w_1 \geq \theta \\
w_1 \cdot 0 + w_2 \cdot 1 &= w_2 \geq \theta
\end{align*}
\]
Perceptrons have limits too

**Theorem:** There is no perceptron representing XOR.

**Proof:** In summary:
- \( w_1 \geq \theta \)
- \( w_2 \geq \theta \)
- \( 0 < \theta \)
- \( w_1 + w_2 < \theta \)

Adding eqs. 1 and 2 gives \( w_1 + w_2 \geq 2\theta \), which contradicts \( w_1 + w_2 < \theta \) since \( \theta \) is positive. \( \square \)
A less rigorous view of this proof

Note: The collection of all points \((x_1, x_2)\) such that \(w_1 \cdot x_1 + w_2 \cdot x_2 = \theta\) must form a line. The points such that \(w_1 \cdot x_1 + w_2 \cdot x_2 \geq \theta\) fall on one side of this line.
A less rigorous view of this proof

We color the points \((x_1, x_2)\) by whether \(x_1 \lor x_2\) is true (black) or false (white).
We color the points \((x_1, x_2)\) by whether \(x_1 \oplus x_2\) is true (black) or false (white).

There is no line through the points such that shaded points are on one side; i.e., XOR is not linearly separable.
Linear Separability of AND and OR

STOP: Draw lines that separate points based on the values of $x_1 \lor x_2$. Do the same for $x_1 \land x_2$. 

© 2024 Phillip Compeau
Linear Separability of AND and OR

STOP: Draw lines that separate points based on the values of $x_1 \lor x_2$. Do the same for $x_1 \land x_2$.

Answer: Shown at right.
STOP: Draw lines that separate points based on the values of $x_1 \lor x_2$. Do the same for $x_1 \land x_2$.

Answer: Shown at right.

You may be wondering how useful perceptrons can be if they can’t model XOR. Sit tight!
A BIT MORE LOGIC
Propositions use logical connectives as building blocks

**Proposition:** A combination of logical connectives in which outputs of one connective can be used as inputs of another (e.g., \((x_1 \land \left( x_2 \lor \neg x_3 \right)) \lor \left( x_4 \lor x_5 \right)\).
Propositions use logical connectives as building blocks

**Proposition:** A combination of logical connectives in which outputs of one connective can be used as inputs of another (e.g., \((x_1 \land (x_2 \lor \neg x_3)) \lor (x_4 \lor x_5)\)).

**Example:** Truth table below demonstrates one of DeMorgan’s Laws: \(~(x_1 \land x_2) \equiv ~x_1 \lor ~x_2\).

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1)</td>
<td>(x_2)</td>
<td>(x_1 \land x_2)</td>
<td>(~(x_1 \land x_2))</td>
<td>(~x_1)</td>
<td>(~x_2)</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
Propositions use logical connectives as building blocks

Note: Here “≡” denotes logical equivalence, meaning that the truth table values are the same.

Example: Truth table below demonstrates one of DeMorgan’s Laws: \( \sim(x_1 \land x_2) \equiv \sim x_1 \lor \sim x_2 \).
Propositions use logical connectives as building blocks

The expression \( \neg (x_1 \land x_2) \) is so common that it has its own connective, **NAND** ("not AND"): \( x_1 \uparrow x_2 \).

**Example:** Truth table below demonstrates one of DeMorgan’s Laws: \( \neg (x_1 \land x_2) \equiv \neg x_1 \lor \neg x_2 \).

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_1 \land x_2 )</th>
<th>( \neg (x_1 \land x_2) )</th>
<th>( \neg x_1 )</th>
<th>( \neg x_2 )</th>
<th>( \neg x_1 \lor \neg x_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
Let’s do a couple of exercises!

The expression $$\neg(x_1 \land x_2)$$ is so common that it has its own connective, **NAND** (“not AND”): $$x_1 \uparrow x_2$$.

**Exercise 1:** Find a perceptron representing $$x_1 \uparrow x_2$$.

**Exercise 2:** Find a proposition using connectives other than $$\forall$$ that is logically equivalent to $$x_1 \forall x_2$$.  

© 2024 Phillip Compeau
LINKING PERCEPTRONS INTO NEURAL NETWORKS MAKES THEM MORE POWERFUL
One solution to exercise 1

**Exercise 1:** Find a perceptron representing $x_1 \uparrow x_2$. 

\[
\begin{align*}
\text{Input} & \quad \text{Output} \\
 x_1 & \rightarrow -1 & x_2 & \rightarrow -1 \\
 x_2 & \rightarrow -1 & x_1 \uparrow x_2 & \\
 \end{align*}
\]
Exercise 2: Find a proposition using connectives other than ∨ that is logically equivalent to $x_1 \lor x_2$.

One common solution is that $x_1 \lor x_2 \equiv (x_1 \lor x_2) \land (\sim x_1 \lor \sim x_2)$, which in turn is just $(x_1 \lor x_2) \land (x_1 \uparrow x_2)$. 
Exercise 2: Find a proposition using connectives other than $\oplus$ that is logically equivalent to $x_1 \oplus x_2$.

One common solution is that $x_1 \oplus x_2 \equiv (x_1 \lor x_2) \land (\sim x_1 \lor \sim x_2)$, which in turn is just $(x_1 \lor x_2) \land (x_1 \uparrow x_2)$.

Note: Although we don’t have a perceptron representing $\oplus$, we do have perceptrons representing $\lor$, $\land$, and $\uparrow$...
Constructing a *network* of perceptrons representing $x_1 \lor x_2$

\[ y_1 = x_1 \lor x_2 \]

\[ y_2 = x_1 \uparrow x_2 \]

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_1 + x_2$</th>
<th>$y_1$</th>
<th>$-x_1 - x_2$</th>
<th>$y_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>-2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

© 2024 Phillip Compeau
Constructing a *network* of perceptrons representing $x_1 \vee x_2$

$$y_1 = x_1 \lor x_2$$

$$y_2 = x_1 \uparrow x_2$$

Output: $y_1 \land y_2 = (x_1 \lor x_2) \land (x_1 \uparrow x_2) = x_1 \lor x_2$

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_1 + x_2$</th>
<th>$y_1$</th>
<th>$-x_1 - x_2$</th>
<th>$y_2$</th>
<th>$y_1 \lor y_2$</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>-2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

© 2024 Phillip Compeau
Constructing a network of perceptrons representing \( x_1 \lor x_2 \)

**Neural network:** a network of artificial neurons in which neuron outputs are inputs into other neurons. The above network has a single **hidden layer** of neurons (gray) that are not input variables or output.
THE UNIVERSALITY OF PERCEPTRON NEURAL NETWORKS
Binary function: a function having \( n \) binary variables as input and producing a binary output.

Example: \( f(0,0) = 1; f(0,1) = 1; f(1,0) = 0; f(1,1) = 1. \)

STOP: How many different binary functions are there with \( n \) input variables?
Binary Functions

**Binary function:** a function having $n$ binary variables as input and producing a binary output.

**Example:** $f(0,0) = 1; f(0,1) = 1; f(1,0) = 0; f(1,1) = 1.$

**STOP:** How many different binary functions are there with $n$ input variables?

**Answer:** There are $2^n$ different possible inputs. Each input can produce a 1 or 0; therefore, there are $2^{2^n}$ total binary functions.
Our building blocks can be used to build any binary function

**Note:** this binary function can be represented by the proposition $\neg x_1 \lor x_2$, with $1 = \text{true}$ and $0 = \text{false}$.

**Example:** $f(0,0) = 1; \ f(0,1) = 1; \ f(1,0) = 0; \ f(1,1) = 1$. 
Our building blocks can be used to build any binary function

**Note:** this binary function can be represented by the proposition $\neg x_1 \lor x_2$, with $1 = \text{true}$ and $0 = \text{false}$.

**Example:** $f(0,0) = 1$; $f(0,1) = 1$; $f(1,0) = 0$; $f(1,1) = 1$.

**Theorem:** Any binary function can be represented by some proposition formed by a finite number of the logical connectives $\land$, $\lor$, and $\neg$. 
Our building blocks can be used to build *any* binary function

**Example:** $f(0,0) = 1; f(0,1) = 1; f(1,0) = 0; f(1,1) = 1$.  

**Theorem:** Any binary function can be represented by some proposition formed by a finite number of the logical connectives $\land$, $\lor$, and $\sim$.

**Key point:** All these connectives can be represented by single perceptrons...
Our building blocks can be used to build *any* binary function.

**Note:** this binary function can be represented by the proposition $\neg x_1 \lor x_2$, with $1 = \text{true}$ and $0 = \text{false}$.

**Example:** $f(0,0) = 1; f(0,1) = 1; f(1,0) = 0; f(1,1) = 1$.

**Corollary:** Any binary function can be represented by a neural network of finitely many perceptrons.

**Key point:** All these connectives can be represented by single perceptrons...
The only building block we need is NAND

Recall that \( \neg(x_1 \land x_2) \) is abbreviated as \( x_1 \uparrow x_2 \).
The only building block we need is NAND

Recall that \( \sim(x_1 \land x_2) \) is abbreviated as \( x_1 \uparrow x_2 \).

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Theorem:** Any binary function can be represented by some proposition formed exclusively by a finite number of \( \uparrow \) connectors.
The only building block we need is NAND

Proof: We will show that each of the expressions \( \sim x_1, (x_1 \land x_2), \) and \( (x_1 \lor x_2) \) can be represented with just NAND \((\uparrow)\) connectors.

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_1 \uparrow x_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Recall that \( \sim (x_1 \land x_2) \) is abbreviated as \( x_1 \uparrow x_2 \).
The only building block we need is NAND

Recall that \(\neg(x_1 \land x_2)\) is abbreviated as \(x_1 \uparrow x_2\).

**Proof:** We will show that each of the expressions \(\neg x_1\), \((x_1 \land x_2)\), and \((x_1 \lor x_2)\) can be represented with just NAND (\(\uparrow\)) connectors.

<table>
<thead>
<tr>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_1 \uparrow x_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**STOP:** Find a proposition formed only of \(\uparrow\) connectors that is logically equivalent to \(\neg x_1\).
The only building block we need is NAND

Recall that \( \sim(x_1 \land x_2) \) is abbreviated as \( x_1 \uparrow x_2 \).

**Proof:** We will show that each of the expressions \( \sim x_1, (x_1 \land x_2), \) and \( (x_1 \lor x_2) \) can be represented with just NAND (\( \uparrow \)) connectors.

**Answer:** \( \sim x_1 \equiv x_1 \uparrow x_1 \).
The only building block we need is NAND

Proof: We will show that each of the expressions \(~x_1\), \((x_1 \land x_2)\), and \((x_1 \lor x_2)\) can be represented with just NAND (\(↑\)) connectors.

Exercise: Find propositions of \(↑\) connectors that are logically equivalent to \((x_1 \land x_2)\) and \((x_1 \lor x_2)\).
The only building block we need is NAND

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_1 \land x_2$</th>
<th>$x_1 \uparrow x_2$</th>
<th>$(x_1 \uparrow x_2) \uparrow (x_1 \uparrow x_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_1 \lor x_2$</th>
<th>$x_1 \uparrow x_1$</th>
<th>$x_2 \uparrow x_2$</th>
<th>$(x_1 \uparrow x_1) \uparrow (x_2 \uparrow x_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
The only building block we need is NAND

Recall that $\neg(x_1 \land x_2)$ is abbreviated as $x_1 \uparrow x_2$.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_1 \uparrow x_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Theorem:** Any binary function can be represented by some proposition formed exclusively by a finite number of $\uparrow$ connectors.

**STOP:** Now that we have proven this theorem, what is the corollary?
The only building block we need is NAND

Recall that \( \sim(x_1 \land x_2) \) is abbreviated as \( x_1 \uparrow x_2 \).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>( x_1 \uparrow x_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Corollary: Any binary function can be represented by a neural network of NAND perceptrons.

Note: \( \triangleleft \) is called a NAND gate.
MODELING THE EVOLUTION OF BIOLOGICAL MODULARITY
Returning to our original question

Can we build a (simple) model in which modularity spontaneously evolves as an optimal solution?

https://www.pnas.org › content

Spontaneous evolution of modularity and network motifs | PNAS
by N Kashtan · 2005 · Cited by 899 — Nadav Kashtan and Uri Alon ... To understand the origin of modularity and network motifs in biology one has to understand how these features ...
The Kashtan-Alon Model

Organisms: all 4-input networks of NAND perceptrons

Note: is called a NAND gate.
The Kashtan-Alon Model

Organisms: all 4-input networks of NAND perceptrons

Goal ($G$): correctly "compute" as many inputs as possible for the proposition $(x_1 \lor x_2) \land (x_3 \lor x_4)$. 

© 2024 Phillip Compeau
The Kashtan-Alon Model

Organisms: all 4-input networks of NAND perceptrons

Goal (G): correctly "compute" as many inputs as possible for the proposition \((x_1 \lor x_2) \land (x_3 \lor x_4)\).

STOP: How many different choices of input are there for this proposition?
The Kashtan-Alon Model

Organisms: all 4-input networks of **NAND** perceptrons

Goal (G): correctly "compute" as many inputs as possible for the proposition \((x_1 \oplus x_2) \land (x_3 \oplus x_4)\).

**STOP:** How many different choices of input are there for this proposition?

**Answer:** Two possibilities for each variable, so \(2^4 = 16\).
One way of reaching the goal

Recall that \((x_1 \lor x_2) \equiv (x_1 \lor x_2) \land (x_1 \uparrow x_2)\).

By the theorem from previously, there is some neural network of NAND gates for \((x_1 \lor x_2) \land (x_1 \uparrow x_2)\).
One way of reaching the goal

And yet there is a simpler network for $x_1 \vee x_2$, which is $[x_1 \uparrow (x_1 \uparrow x_2)] \uparrow [x_2 \uparrow (x_1 \uparrow x_2)]$, as shown below.
One way of reaching the goal

And yet there is a simpler network for $x_1 \vee x_2$, which is $[x_1 \uparrow (x_1 \uparrow x_2)] \uparrow [x_2 \uparrow (x_1 \uparrow x_2)]$, as shown below.

**Key point:** we should prioritize this smaller network because it would be easier to have evolved.
One way of reaching the goal

And yet there is a simpler network for $x_1 \lor x_2$, which is $[x_1 \uparrow (x_1 \uparrow x_2)] \uparrow [x_2 \uparrow (x_1 \uparrow x_2)]$, as shown below.

**Key point:** we should prioritize this smaller network because it would be easier to have evolved.

To prefer a smaller network over a larger network, Kashtan and Alon defined a **fitness function** for a network as the fraction of the 16 input assignments whose output matches the goal $G$, minus a small positive $\varepsilon$ times the number $m$ of **NAND** gates.
The Kashtan-Alon Algorithm

1. Construct 100 random initial networks.
2. Run the following algorithm for 10,000 “generations”.
   1. Consider only the 50 networks having highest fitness.
   2. Use these networks to produce 100 “children” networks that have mutations compared to the parent networks.
3. At the end, return the network(s) having maximum fitness as the winner(s).
1. Construct 100 random initial networks.
2. Run the following algorithm for 10,000 “generations”.
   1. Consider only the 50 networks having highest fitness.
   2. Use these networks to produce 100 “children” networks that have mutations compared to the parent networks.
3. At the end, return the network(s) having maximum fitness as the winner(s).

This type of search heuristic, which mimics evolution, is called a **genetic algorithm**.
Our winner isn’t very modular... 😞
Life changes, and fitness should change too

**Key point:** a more realistic model of a competitive landscape would use a *variable* fitness function.
Life changes, and fitness should change too

**Key point:** a more realistic model of a competitive landscape would use a *variable* fitness function.

Previous goal (G): correctly “compute” as many inputs as possible for \((x_1 \lor x_2) \land (x_3 \lor x_4)\).
Life changes, and fitness should change too

**Key point:** a more realistic model of a competitive landscape would use a *variable* fitness function.

Previous goal \((G)\): correctly "compute" as many inputs as possible for \((x_1 \lor x_2) \land (x_3 \lor x_4)\).

Alternate goal \((H)\): correctly "compute" as many inputs as possible for \((x_1 \lor x_2) \lor (x_3 \lor x_4)\).
Adapting the algorithm to incorporate *variable* fitness

1. Construct 100 random initial networks.
2. Run the following algorithm for 10,000 “generations”.
   1. Consider only the 50 networks having highest fitness.
   2. Use these networks to produce 100 “children” networks that have mutations compared to the parent networks.
   3. Every $e$ generations ($e = 20$ in original paper), switch the goal function from $G$ to $H$ or vice-versa.
3. At the end, return the network(s) having maximum fitness as the winner(s).
With the static goal $G$, we found a non-modular solution

Key point: when the goal is $H$, we need many mutations to this network.
Dynamic fitness leads to a modular solution to $G$ in $\sim 5000$ generations
Switching the goal to $H$ yields a very slightly different modular solution
A great idea leads to more questions

1. What is the extent to which real fitness functions reward modularity?
2. What are the limits of modularity in biology?
3. And what happens when we start building models of consciousness that are more advanced than the neural networks presented here?
EPILOGUE: PRACTICAL APPLICATIONS OF NEURAL NETWORKS AI MAGIC IN 20 MINUTES
Many problems can be framed as classification

**Classification Problem**

- **Input:** A collection of data divided into a training set and a test set. Each training data point is labeled into one of \( k \) classes.

- **Output:** a predictive labeling of all the points in the test set into one of \( k \) classes.

**Example:** Our data might be images of skin lesions, which we want to classify as non-neoplastic, a benign tumor, or malignant (cancer).
Converting data into a manageable form

Example: If each image has $n$ pixels, then each pixel has three RGB values, representing the amount of red, green, and blue in each pixel. This produces $3n$ 0-1 decimal values for each image.

https://excelatfinance.com/xlf/xlf-colors-1.php
A generalized neuron allows $n$ arbitrary decimal inputs (often between 0 and 1) and fires $f(w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_n \cdot x_n - b)$ for an activation function $f$ and a constant bias $b$. 
Generalizing neural networks

One common activation function is the **logistic function**: \( f(x) = \frac{1}{1 + e^{-x}} \), shown below.

\[
f(w_1 \cdot x_1 + w_2 \cdot x_2 - b)
\]

STOP: What was the “activation function” that we were using with perceptrons?

\[ f(w_1 \cdot x_1 + w_2 \cdot x_2 - b) \]
Generalizing neural networks

**Answer:** The “step function” $S(x)$ that outputs 1 if $x \geq \theta$ and outputs 0 if $x < \theta$. 
Generalizing neural networks

**Note:** even though it’s simple, researchers now often use a “rectifier” function: \( f(x) = \max(0, x) \).
We then build some gigantic network with several hidden layers.

Congrats! You are now a **deep learning** expert.
We then build some gigantic network with several hidden layers.

For a data value $x$, its output is a vector $P(x)$.
We then build some gigantic network with several hidden layers.

We want $P(x)$ for a benign image similar to $(0, 1, 0)$. 

© 2024 Phillip Compeau
We have a lot of freedom in parameter selection

**Note:** For every neuron in our network, all of the input weights $w_i$ are parameters.

**Network Parameter Learning Problem**

- **Input:** A collection of vectorized data and a neural network.
- **Output:** a collection of weights and biases that minimizes the average RMSD between an object $x$’s correct label vector, $L(x)$, and the prediction from the network, $P(x)$, over all objects $x$. 
We have a lot of freedom in parameter selection

STOP: Does “distance between two vectors” ring any bells?

Network Parameter Learning Problem

• **Input:** A collection of vectorized data and a neural network.

• **Output:** a collection of weights and biases that minimizes the average RMSD between an object \( x \)'s correct label vector, \( L(x) \), and the prediction from the network, \( P(x) \), over all objects \( x \).
We have a lot of freedom in parameter selection

Answer: RMSD is one way of quantifying this distance.

Network Parameter Learning Problem

• Input: A collection of vectorized data and a neural network.

• Output: a collection of weights and biases that minimizes the average RMSD between an object $x$’s correct label vector, $L(x)$, and the prediction from the network, $P(x)$, over all objects $x$. 
We have a lot of freedom in parameter selection

STOP: What kind of computational problem is this?

**Network Parameter Learning Problem**

- **Input:** A collection of vectorized data and a neural network.
- **Output:** a collection of weights and biases that minimizes the average RMSD between an object \(x\)'s correct label vector, \(L(x)\), and the prediction from the network, \(P(x)\), over all objects \(x\).
We have a lot of freedom in parameter selection

**Answer:** It’s an optimization problem, where the search space is the collection of weights/biases.

**Network Parameter Learning Problem**

- **Input:** A collection of vectorized data and a neural network.
- **Output:** a *collection of weights and biases* that minimizes the average RMSD between an object $x$’s correct label vector, $L(x)$, and the prediction from the network, $P(x)$, over all objects $x$. 
We have a lot of freedom in parameter selection

Note: Much of deep learning is just “build a big network and apply a local search heuristic”.

Network Parameter Learning Problem

• **Input:** A collection of vectorized data and a neural network.

• **Output:** a collection of weights and biases that minimizes the average RMSD between an object $x$’s correct label vector, $L(x)$, and the prediction from the network, $P(x)$, over all objects $x$. 

© 2024 Phillip Compeau
Still, deep learning can be impressive...
STOP: Any guesses on how accurate their algorithm was?
... and a fancier version of our skin lesion network was a real paper!

https://www.nature.com/letters/article

Dermatologist-level classification of skin cancer with ... - Nature
by A Esteva · 2017 · Cited by 5697 — Using a single convolutional neural network trained on general skin lesion classification, we match the performance of at least 21 dermatologists tested across three critical diagnostic tasks: keratinocyte carcinoma classification, melanoma classification and melanoma classification using dermoscopy.

STOP: Any guesses on how accurate their algorithm was?

Answer: Around 70% accurate, compared to 67% accuracy for a dermatologist.
"Following from an extensive literature review, we find that deep learning has yet to revolutionize biomedicine or definitively resolve any of the most pressing challenges in the field, but promising advances have been made on the prior state of the art."
This Might Not Age the Best!

… but is this really a model of intelligence?

“Let's not impose artificial constraints based on cartoon models of topics in science that we don't yet understand.” – Michael I. Jordan, 2014

https://www.reddit.com/r/MachineLearning/comments/2fxi6v/ama_michael_i_jordan/
... but is this really a model of *intelligence*?

**Idea:** if nature is good at solving problems, why don't we study the algorithms that it has developed over the course of evolution?