Lecture 9: Two-Layer Neural Nets

Mark Hasegawa-Johnson
2/2022
License: CC-BY 4.0. You may remix or redistribute if you cite the source.
Outline

• Breaking the constraints of linearity: multi-layer neural nets
• Flow diagram for the XOR problem
• Flow diagram for a multi-layer neural net
• Forward-propagation example
In 1943, McCulloch & Pitts proposed that biological neurons have a nonlinear activation function (a step function) whose input is a weighted linear combination of the currents generated by other neurons.

They showed lots of examples of mathematical and logical functions that could be computed using networks of simple neurons like this.
Biological Inspiration: Neuronal Circuits

• Even the simplest actions involve more than one neuron, acting in sequence in a neuronal circuit.

• One of the simplest neuronal circuits is a reflex arc, which may contain just two neurons:
  • The **sensor neuron** detects a stimulus, and communicates an electrical signal to ...
  • The **motor neuron**, which activates the muscle.

Illustration of a reflex arc: sensor neuron sends a voltage spike to the spinal column, where the resulting current causes a spike in a motor neuron, whose spike activates the muscle.

By MartaAguayo - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=39181552
A McCulloch-Pitts Neuron can compute some logical functions...

When the features are binary \((x_j \in \{0,1\})\), many (but not all!) binary functions can be re-written as linear functions. For example, the function
\[
f(\vec{x}) = (x_1 \lor x_2)
\]
can be re-written as
\[
f(\vec{x}) = u(x_1 + x_2 - 0.5)
\]

Similarly, the function
\[
f(\vec{x}) = (x_1 \land x_2)
\]
can be re-written as
\[
f(\vec{x}) = u(x_1 + x_2 - 1.5)
\]
... but not all.

“A linear classifier cannot learn an XOR function.”
- Minsky & Papert, 1969

• ...but a **two-layer neural net** can compute an XOR function!
Outline

• Breaking the constraints of linearity: multi-layer neural nets
• Flow diagram for the XOR problem
• Flow diagram for a multi-layer neural net
• Forward-propagation example
Example: one way (of many possible ways) to represent the XOR function using a two-layer neural network

For example, consider the XOR problem.

Suppose we create two hidden nodes:

\[ h_1(\vec{x}) = u(0.5 - x_1 - x_2) \]
\[ h_2(\vec{x}) = u(x_1 + x_2 - 1.5) \]

Then the XOR function \( f(\vec{x}) = (x_1 \oplus x_2) \) is given by \( f(\vec{x}) = \neg(x_1 \lor x_2) \). For example, we could write this as:

\[ f(\vec{x}) = u(0.5 - h_1(x) - h_2(x)) \]
Suppose we create two hidden nodes:

\[
\begin{align*}
    h_1(\vec{x}) &= u(0.5 - x_1 - x_2) \\
    h_2(\vec{x}) &= u(x_1 + x_2 - 1.5)
\end{align*}
\]

Here is a flow diagram for this computation:
Flow diagrams

A flow diagram is a way to represent the computations performed by a neural network.

  • The circles above $x_1$ and $x_2$ represent the scalar operation of “read this datum in from the dataset.”
  • The circles labeled $h_1$ and $h_1$ represent the scalar operation of “unit step function.”

• Lines represent multiplication by a scalar.

• Where arrowheads come together, the corresponding variables are added.

Here in the middle, both $h_1(\vec{x})$ and $h_2(\vec{x})$ are zero.
Flow diagrams

It’s often useful to distinguish two types of hidden variables at each neuron:

• The neural excitation, $\xi_j$, is the result of adding together all of the inputs to the neuron.

• The neural activation, $h_j$, is the result of passing $\xi_j$ through a scalar nonlinearity.

Here in the middle, both $h_1(\vec{x})$ and $h_2(\vec{x})$ are zero.
Flow diagrams

So in this flow diagram, for example, we can see that:

\[
\begin{align*}
\xi_1 &= 0.5 - 1 \cdot x_1 - 1 \cdot x_2 \\
\xi_2 &= -1.5 + 1 \cdot x_1 + 1 \cdot x_2
\end{align*}
\]

... and then ...

\[
\begin{align*}
h_1 &= u(\xi_1) \\
h_2 &= u(\xi_2)
\end{align*}
\]

... where \(u(\cdot)\) is the unit step function.
Flow diagrams

Now suppose that we want to compute $f(\vec{x}) = (x_1 \oplus x_2)$. We could write this as:

$$f(\vec{x}) = u(0.5 - h_1 - h_2)$$

Here in the middle, both $h_1(\vec{x})$ and $h_2(\vec{x})$ are zero.
Flow diagrams

We can write the XOR function as:

$$\xi_3 = 0.5 - 1 \cdot h_1 - 1 \cdot h_2$$

$$f(\vec{x}) = u(\xi_3)$$
Flow diagrams

Putting it all together:
\[ \xi_1 = 0.5 - 1 \cdot x_1 - 1 \cdot x_2 \]
\[ \xi_2 = -1.5 + 1 \cdot x_1 + 1 \cdot x_2 \]

\[ h_1 = u(\xi_1) \]
\[ h_2 = u(\xi_2) \]

\[ \xi_3 = 0.5 - 1 \cdot h_1 - 1 \cdot h_2 \]

\[ f(\tilde{x}) = u(\xi_3) \]
Outline

• Breaking the constraints of linearity: multi-layer neural nets
• Flow diagram for the XOR problem
• Flow diagram for a multi-layer neural net
• Forward-propagation example
Multi-layer neural net

• $\xi_j^{(l)} =$ **excitation** of the $j^{th}$ neuron (a.k.a. “node”) in the $l^{th}$ layer
  • Computed by adding together inputs from many other neurons, each weighted by a corresponding connection strength or connection weight, $w_{j,k}^{(l)}$

• $h_j^{(l)} =$ **activation** of the $j^{th}$ node in the $l^{th}$ layer
  • This is computed by just passing the excitation through a scalar nonlinear activation function, thus $h_j^{(l)} = g(\xi_j^{(l)})$. The activation functions in different layers differ, so to be pedantic, sometimes we’ll write $h_j^{(l)} = g^{(l)}\left(\xi_j^{(l)}\right)$. 
Multi-layer neural net

Given: some training token $\mathbf{x} = [x_1, \ldots, x_D, 1]^T$ and its target label $y$

1. Initialize: $h_k^{(0)} = x_k$

2. Forward propagate: for $l \in \{1, \ldots, L\}$:
   a. Compute the excitations as weighted sums of the previous-layer activations:
      $$\xi_j^{(l)} = b_j^{(l)} + \sum_k w_{j,k}^{(l)} h_k^{(l-1)}$$
   b. Compute the activations by applying scalar nonlinearities:
      $$h_j^{(l)} = g^{(l)}(\xi_j^{(l)})$$

3. Output: $P(Y = k | x) = h_k^{(L)}$
Forward propagation

• From activation to excitation is a matrix multiply:
  \[ \xi_j^{(l)} = b_j^{(l)} + \sum_k w_{j,k}^{(l)} h_k^{(l-1)} \]

• From excitation to activation is a scalar nonlinearity:
  \[ h_j^{(l)} = g^{(l)}(\xi_j^{(l)}) \]
Forward propagation: Matrix multiply

From activation to excitation is a matrix multiply:

$$\tilde{\xi}^{(l)} = W^{(l)} \tilde{h}^{(l-1)}$$

...where...

$$\tilde{\xi}^{(l)} = \begin{bmatrix} \xi_1^{(l)} \\ \vdots \\ \xi_N^{(l)} \end{bmatrix}, \quad \tilde{h}^{(l-1)} = \begin{bmatrix} h_1^{(l-1)} \\ \vdots \\ h_M^{(l-1)} \\ 1 \end{bmatrix},$$

$$W^{(l)} = \begin{bmatrix} w_{1,1}^{(l)} & \cdots & w_{1,M}^{(l)} & b_1^{(l)} \\ \vdots & \ddots & \vdots & \vdots \\ w_{N,1}^{(l)} & \cdots & w_{N,M}^{(l)} & b_N^{(l)} \end{bmatrix}$$
Forward propagation

From excitation to activation is a scalar nonlinearity:

\[ h_j^{(l)} = g^{(l)} \left( \xi_j^{(l)} \right) \]

What type of nonlinearity?
Answer: it depends on what task you want your neural net to learn.
Activation functions

The “activation function,” \( g^{(l)}(\cdot) \), can be any scalar nonlinearity. Common ones that you should know include the **unit step** and **signum** functions, and:

**Logistic Sigmoid:**
\[
\sigma(\beta) = \frac{1}{1 + e^{-\beta}}
\]

**Hyperbolic Tangent (tanh):**
\[
tanh(\beta) = \frac{e^\beta - e^{-\beta}}{e^\beta + e^{-\beta}}
\]

**Rectified Linear Unit (ReLU):**
\[
\text{ReLU}(\beta) = \max(0, \beta)
\]
Outline

• Breaking the constraints of linearity: multi-layer neural nets
• Flow diagram for the XOR problem
• Flow diagram for a multi-layer neural net
• Forward-propagation example
Example: Square smiley

- Input: $\tilde{x} = [x_1, x_2, 1]^T$
- Output: $f(\tilde{x}) = [R, G, B]^T$

Remember that yellow = red + green, so we just need to compute $R, G, B$ as functions of $x_1$ and $x_2$.

This could be done using a two-layer network, but it's easier using a three-layer network, so let's do it that way.
Layer 1

In layer 1, let’s find all the different ways in which we need to bisect the image plane:

\[ h_{1}^{(1)} = \text{sign}(x_{1} - 0.5) \]
\[ \vdots \]
\[ h_{13}^{(1)} = \text{sign}(x_{2} - 1.5) \]
Layer 1

In layer 1, let’s find all the different ways in which we need to bisect the image plane:

\[ h_1^{(1)} = \text{sign}(x_1 - 0.5) \]
\[ \vdots \]
\[ h_{13}^{(1)} = \text{sign}(x_2 - 1.5) \]
Layer 2

In layer 2, let’s compute rectangles of solid color. We can compute those using logical operations:

\[
h^{(2)}_1 = h^{(1)}_1 \land \neg h^{(1)}_2 \land h^{(1)}_7 \land \neg h^{(1)}_{13}
\]

\[
\vdots
\]

\[
h^{(2)}_{12} = h^{(1)}_5 \land \neg h^{(1)}_6 \land h^{(1)}_7 \land \neg h^{(1)}_{13}
\]
Layer 2

... and then convert the logical operations into linear functions:

\[ h_1^{(2)} = u \left( h_1^{(1)} - h_2^{(1)} + h_7^{(1)} - h_{13}^{(1)} - 3.5 \right) \]

\[ h_{12}^{(2)} = u \left( h_5^{(1)} - h_6^{(1)} + h_7^{(1)} - h_{13}^{(1)} - 3.5 \right) \]
Layer 2

In layer 2, let’s compute rectangles of solid color:

\[ h_1^{(2)} = u \left( h_1^{(1)} - h_2^{(1)} + h_7^{(1)} - h_{13}^{(1)} - 3.5 \right) \]

\[ h_{12}^{(2)} = u \left( h_5^{(1)} - h_6^{(1)} + h_7^{(1)} - h_{13}^{(1)} - 3.5 \right) \]
Layer 3

In layer 3, let’s compute the red, green, and blue regions using the inclusive-OR of these rectangles:

\[
R = h_1^{(2)} \lor h_2^{(2)} \lor h_3^{(2)} \lor h_4^{(2)} \\
\lor h_5^{(2)} \lor h_6^{(2)} \lor h_7^{(2)} \lor h_9^{(2)} \lor h_{11}^{(2)} \lor h_{12}^{(2)}
\]

\[
G = h_1^{(2)} \lor h_2^{(2)} \lor h_5^{(2)} \lor h_7^{(2)} \lor h_9^{(2)} \lor h_{11}^{(2)} \lor h_{12}^{(2)}
\]

\[
B = h_8^{(2)} \lor h_{10}^{(2)}
\]
Layer 3

In layer 3, let’s compute the red, green, and blue regions using the inclusive-OR of these rectangles:

\[ R = u \left( h_1^{(2)} + h_2^{(2)} + h_3^{(2)} + h_4^{(2)} + h_5^{(2)} + h_6^{(2)} + h_7^{(2)} + h_9^{(2)} + h_{11}^{(2)} + h_{12}^{(2)} - 0.5 \right) \]

\[ G = u \left( h_1^{(2)} + h_2^{(2)} + h_5^{(2)} + h_7^{(2)} + h_9^{(2)} + h_{11}^{(2)} + h_{12}^{(2)} - 0.5 \right) \]

\[ B = u \left( h_8^{(2)} + h_{10}^{(2)} - 0.5 \right) \]
Layer 3

In layer 3, let’s compute the red, green, and blue regions using the inclusive-OR of these rectangles:

\[ R = u \left( h_1^{(2)} + h_2^{(2)} + h_3^{(2)} + h_4^{(2)} + h_5^{(2)} + h_6^{(2)} + h_7^{(2)} + h_9^{(2)} + h_{11}^{(2)} + h_{12}^{(2)} - 0.5 \right) \]

\[ G = u \left( h_1^{(2)} + h_2^{(2)} + h_5^{(2)} + h_7^{(2)} + h_9^{(2)} + h_{11}^{(2)} + h_{12}^{(2)} - 0.5 \right) \]

\[ B = u \left( h_8^{(2)} + h_{10}^{(2)} - 0.5 \right) \]
Summary

• Breaking the constraints of linearity: multi-layer neural nets

• Flow diagram for the XOR problem: if $x_1$ and $x_2$ are binary, then
  $$(x_1 \lor x_2) = u(x_1 + x_2 - 0.5)$$
  $$(x_1 \land x_2) = u(x_1 + x_2 - 1.5)$$

• Flow diagram for a multi-layer neural net

\[
\begin{align*}
\xi_j^{(l)} &= b_j^{(l)} + \sum_{k} w_{j,k}^{(l)} h_k^{(l-1)} \\
    h_j^{(l)} &= g^{(l)} \left( \xi_j^{(l)} \right)
\end{align*}
\]

• Forward-propagation example