CS 232: Artificial Intelligence

Prof. Carolyn Anderson
Wellesley College

Fall 2023
Recap
Spot the differences

Neural Network Unit

\[ z = b + \sum_{i} w_i x_i \]

\[ y = \sigma(w \cdot x + b) \]

Logistic Regression

\[ z = \left( \sum_{i=1}^{n} w_i x_i \right) + b \]

\[ P(y = 1) = \sigma(w \cdot x + b) \]

Relu

Tanh

Sigmoid
Final unit again

Output value

Non-linear activation function

Weighted sum

Weights

Input layer

\[
\sum \sigma + b
\]
Example: XOR
### The XOR problem

Can neural units compute simple functions of input?

Minsky and Papert (1969)

<table>
<thead>
<tr>
<th>AND</th>
<th>OR</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1)</td>
<td>(x_2)</td>
<td>(y)</td>
</tr>
<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>
Perceptrons

A very simple neural unit

- Binary output (0 or 1)
- No non-linear activation function

\[ y = \begin{cases} 
0, & \text{if } w \cdot x + b \leq 0 \\
1, & \text{if } w \cdot x + b > 0 
\end{cases} \]
Deriving AND

Goal: return 1 if $x_1$ and $x_2$ are 1

Use bias to make sure both must be 1

$$y = \begin{cases} 
0, & \text{if } \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\
1, & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 
\end{cases}$$
Deriving OR

Goal: return 1 if either input is 1
Don’t need to do anything with bias

\[ y = \begin{cases} 
0, & \text{if } w \cdot x + b \leq 0 \\
1, & \text{if } w \cdot x + b > 0 
\end{cases} \]
solving XOR

\[ w_1 x_1 + w_2 x_2 + b \leq 0 \]
if \( x_1 \) & \( x_2 \) are 1

\[ w_1 + w_2 + b \leq 0 \]

\[ w_1 x_1 + w_2 x_2 + b > 0 \]
if \( x_1 = 1 \) & \( x_2 = 0 \)

\[ w_1 + b > 0 \]

<table>
<thead>
<tr>
<th>x1</th>
<th>x2</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ b \leq 0 \]

\[ x \rightarrow s \rightarrow w_2 \rightarrow b \rightarrow +1 \rightarrow XOR \]
Trick question!
It's not possible to capture XOR with perceptrons
Why? Perceptrons are linear classifiers

Perceptron equation is the equation of a line

\[ w_1 x_1 + w_2 x_2 + b = 0 \]

(in standard linear format: \( x_2 = \left(-\frac{w_1}{w_2}\right)x_1 + \left(-\frac{b}{w_2}\right) \))

This line acts as a **decision boundary**

- 0 if input is on one side of the line
- 1 if on the other side of the line
Decision boundaries

OR

XOR

AND
Solution to the XOR problem

XOR can't be calculated by a single perceptron
XOR can be calculated by a layered network of units.

<table>
<thead>
<tr>
<th>XOR</th>
<th></th>
<th></th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>x2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

ReLU: $f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$
The hidden representation $h$

hidden layers: intermediate units learn transformations of data
Feedforward Networks
Neural Network Unit

Non-linear activation function

Weighted sum

Weights

Input layer

Output value

bias
Multinomial Logistic Regression as a 1-layer Network

Fully connected single layer network

Output layer (softmax nodes)

\[
\hat{y} = \text{softmax}(Wx + b)
\]

Input layer

\[
W \quad X_1 \quad X_2 \quad X_3 \quad X_n \quad +1
\]

\[
b
\]

\[
\sum_{i=1}^{n} \text{softmax} (Wx + b)
\]

\[
y_1 \quad y_n
\]
Two-Layer Network with scalar output

Output layer
(\(\sigma\) node)

hidden units
(\(\sigma\) node)

Input layer

\[ y = \sigma(z) \]
\[ z = Uh \]

\[ h = \sigma(Wx + b) \]

Could be ReLU
Or tanh

Binary Classification
Non-linear activation
Using feedforward networks
Can we get back to cat pics, please?

Finally, we’re ready to power up our supervised cat/dog classifier by adding more layers. This takes it from a regression model to a neural network.
New Architecture

```
| input_1 | InputLayer | input: | [(None, 180, 180)] |
|         |           | output:| [(None, 180, 180)] |
```

```
| flatten | Flatten   | input: | (None, 180, 180) |
|         |           | output:| (None, 32400)    |
```

```
| dense_1 | Dense     | input: | (None, 32400)    |
|         |           | output:| (None, 500)      |
```

```
| dense_2 | Dense     | input: | (None, 500)      |
|         |           | output:| (None, 1)        |
```
Training a Neural Network
Intuition: training a 2-layer Network

Actual answer $y$

System output $\hat{y}$

Loss function $L(\hat{y}, y)$

Forward

Training instance $X_1, X_n$

Backward
Intuition: Training a 2-layer network

For every training tuple \((x, y)\)

- Run **forward** computation to find our estimate \(\hat{y}\)
- Run **backward** computation to update weights:
  - For every output node
    
    Compute loss \(L\) between \(y\) and \(\hat{y}\)
    
    For every weight \(w\) from hidden layer \(\rightarrow\) output layer, update the weight.
  
- For every hidden node
  
  Assess how much blame it deserves
  
  For every weight \(w\) from input \(\rightarrow\) hidden layer,
  update according to blame.