Artificial Neural Networks: Intro

“Making Connections” by Filomena Booth (2013)
Non-Linear Decision Surfaces

• There is no linear decision boundary
Car Classification

Testing:

What is this?
You see this:

But the camera sees this:

<table>
<thead>
<tr>
<th>194</th>
<th>210</th>
<th>201</th>
<th>212</th>
<th>199</th>
<th>213</th>
<th>215</th>
<th>195</th>
<th>178</th>
<th>158</th>
<th>182</th>
<th>209</th>
</tr>
</thead>
<tbody>
<tr>
<td>180</td>
<td>189</td>
<td>190</td>
<td>221</td>
<td>209</td>
<td>205</td>
<td>191</td>
<td>167</td>
<td>147</td>
<td>115</td>
<td>129</td>
<td>163</td>
</tr>
<tr>
<td>114</td>
<td>126</td>
<td>140</td>
<td>188</td>
<td>176</td>
<td>165</td>
<td>152</td>
<td>140</td>
<td>170</td>
<td>106</td>
<td>78</td>
<td>88</td>
</tr>
<tr>
<td>87</td>
<td>103</td>
<td>115</td>
<td>154</td>
<td>143</td>
<td>142</td>
<td>149</td>
<td>153</td>
<td>173</td>
<td>101</td>
<td>57</td>
<td>57</td>
</tr>
<tr>
<td>102</td>
<td>112</td>
<td>106</td>
<td>131</td>
<td>122</td>
<td>138</td>
<td>152</td>
<td>147</td>
<td>128</td>
<td>84</td>
<td>58</td>
<td>66</td>
</tr>
<tr>
<td>94</td>
<td>95</td>
<td>79</td>
<td>104</td>
<td>105</td>
<td>124</td>
<td>129</td>
<td>113</td>
<td>107</td>
<td>87</td>
<td>69</td>
<td>67</td>
</tr>
<tr>
<td>66</td>
<td>71</td>
<td>69</td>
<td>98</td>
<td>89</td>
<td>92</td>
<td>98</td>
<td>95</td>
<td>89</td>
<td>88</td>
<td>76</td>
<td>67</td>
</tr>
<tr>
<td>41</td>
<td>56</td>
<td>68</td>
<td>99</td>
<td>53</td>
<td>45</td>
<td>60</td>
<td>82</td>
<td>58</td>
<td>76</td>
<td>75</td>
<td>65</td>
</tr>
<tr>
<td>20</td>
<td>43</td>
<td>69</td>
<td>75</td>
<td>56</td>
<td>41</td>
<td>51</td>
<td>73</td>
<td>55</td>
<td>70</td>
<td>63</td>
<td>44</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>57</td>
<td>69</td>
<td>75</td>
<td>75</td>
<td>73</td>
<td>74</td>
<td>53</td>
<td>68</td>
<td>59</td>
<td>37</td>
</tr>
<tr>
<td>72</td>
<td>59</td>
<td>53</td>
<td>66</td>
<td>84</td>
<td>92</td>
<td>84</td>
<td>74</td>
<td>57</td>
<td>72</td>
<td>63</td>
<td>42</td>
</tr>
<tr>
<td>67</td>
<td>61</td>
<td>58</td>
<td>65</td>
<td>75</td>
<td>78</td>
<td>76</td>
<td>73</td>
<td>59</td>
<td>75</td>
<td>69</td>
<td>50</td>
</tr>
</tbody>
</table>
Learning Algorithm

pixel 1

pixel 2

+ Cars

- "Non"-Cars

Raw image
Learning Algorithm

Pixel 1

Pixel 2

Cars

“Non”-Cars
50 x 50 pixel images $\rightarrow$ 2500 pixels

$n = 2500$ \quad (7500 if RGB)

$$x = \begin{bmatrix}
\text{pixel 1 intensity} \\
\text{pixel 2 intensity} \\
\vdots \\
\text{pixel 2500 intensity}
\end{bmatrix}$$

Quadratic features $(x_i \times x_j) \approx 3$ million features
Simple Non-Linear Classification Example

\[ y = x_1 \text{ XOR } x_2 \]
Inspiration: The Brain

- Dendrites
- Nucleus
- Cell body
- Axon
- Axon terminals

Impulses carried toward cell body

Impulses carried away from cell body
Inspiration: The Brain

Axon from a neuron $x_0$ synapses with a dendrite, leading to a cell body. The cell body computes $\sum w_i x_i + b$ and applies an activation function $f$. The output is an output axon.
Linear Neuron

\[ w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 \]
Linear Neuron: Cost Function

• Any number of choices. The one made for linear regression is

\[ \Sigma_{i=1}^{m} \left( y^{(i)} - w^T x^{(i)} \right)^2 \]

• Can minimize this using gradient descent to obtain the best weights \( w \) for the training set
Logistic Neuron

\[ \sigma(w_0 + w_1x_1 + w_2x_2 + w_3x_3), \sigma(t) = \frac{1}{1 + \exp(-t)} \]
Logistic Neuron: Cost Function

- Could use the quadratic cost function again
- Could use the “log-loss” function to make the neuron perform logistic regression

\[- \left( \sum_{i=1}^{m} y^{(i)} \log \left( \frac{1}{1 + \exp(-w^T x^{(i)})} \right) + (1 - y^{(i)}) \log \left( \frac{\exp(-w^T x^{(i)})}{1 + \exp(-w^T x^{(i)})} \right) \right)\]

(Note: we derived this cost function by saying we want to maximize the likelihood of the data under a certain model, but there’s nothing stopping us from just making up a loss function)
Logistic Regression Cost Function: Another Look

\[ \text{Cost}(h_w(x), y) = \begin{cases} 
- \log(h_w(x)), & y = 1 \\
- \log(1 - h_w(x)), & y = 0 
\end{cases} \]

- If \( y = 1 \), want the cost to be small if \( h_w(x) \) is close to 1 and large if \( h_w(x) \) is close to 0
  - \(-\log(t)\) is 0 for \( t = 1 \) and infinity for \( t = 0 \)
- If \( y = 0 \), want the cost to be small if \( h_w(x) \) is close to 0 and large if \( h_w(x) \) is close to 1
- Note: \( 0 < \sigma(t) < 1 \)

\[ \sigma(t) = \frac{1}{1 + \exp(-t)} \]
Multilayer Neural Networks

• $h_{i,j} = g(W_{i,j}x)$
  $$= g\left( \sum_{k} W_{i,j,k}x_k \right)$$
• $x_0 = 1$ always
• $W_{i,j,0}$ is the “bias”
• $g$ is the activation function
  • Could be $g(t) = t$
  • Could be $g(t) = \sigma(t)$
    • Nobody uses those anymore...

output units
hidden units
input units
Why do we need activation functions?
Activation functions?

Most commonly used activation functions:

- **Sigmoid**: $\sigma(z) = \frac{1}{1+\exp(-z)}$
- **Tanh**: $\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$
- **ReLU (Rectified Linear Unit)**: $\text{ReLU}(z) = \max(0, z)$
Exercise:

• Hand code a neural network to compute:
  • AND
  • XOR

• Use *only* sigmoid
• Use *only* ReLU activations
Multilayer Neural Network: Speech Recognition Example (multi-class classification)
Universal Approximator

• Neural networks with at least one hidden layer (and enough neurons) are *universal approximators*
  • Can represent any (smooth) function

• The *capacity* (ability to represent different functions) increases with more hidden layers and more neurons

• Why go deeper? One hidden layer might need *a lot* of neurons. Deeper and narrower networks are more compact
Computation in Neural Networks

• **Forward pass**
  • Making predictions
  • Plug in the input $x$, get the output $y$

• **Backward pass**
  • Compute the gradient of the cost function with respect to the weights
Multilayer Neural Network for Classification:

$o_i$ is large if the probability that the correct class is $i$ is high

A possible cost function:

$$C(o, y) = \sum_{i=1}^{m} |y^{(i)} - o^{(i)}|^2$$

$y^{(i)}$'s and $o^{(i)}$'s encoded using one-hot encoding
Forward Pass (vectorized)

\[
\begin{align*}
o &= g \left( (W^{(2)})^T h + b^{(2)} \right) \\
h &= g \left( (W^{(1)})^T x + b^{(1)} \right)
\end{align*}
\]

...etc... if there are more layers

![Diagram showing a neural network with input, hidden, and output units.](image)
Backwards Pass (training)

Need to find

\[ W = \underset{W}{\arg\min} \sum_{i=1}^{m} \text{loss}(o^{(i)}, y^{(i)}) \]

Where:
- \( o^{(i)} \) is the output of the neural network
- \( y^{(i)} \) is the ground truth
- \( W \) is all the weights in the neural network
- \( \text{loss} \) is a continuous loss function.

Use **gradient descent** to find a good \( W \).
But how to compute gradient?

• To optimize the weights / parameters of the neural network, we need to compute gradient of the cost function:
  \[ C(o, y) = \sum_{i=1}^{m} loss(o^{(i)}, y^{(i)}) \]
  with respect to every weight in the neural network.

• Need to compute, for every layer and weight \( l, j, i \):
  \[ \frac{\partial C}{\partial W^{(l,j,i)}} \]

• How to do this? How to do this efficiently?
Review: Chain Rule

• Univariate Chain Rule

\[ \frac{d}{dt} g(f(t)) = \frac{dg}{df} \cdot \frac{df}{dt} \]

• Multivariate Chain Rule

\[ \frac{\partial g}{\partial x} = \sum \frac{\partial g}{\partial f_i} \frac{\partial f_i}{\partial x} \]
Gradient of Single Weight (last layer)

- We need the partial derivatives of the cost function $C(o, y)$ w.r.t all the $W$ and $b$.
- $o_i = g \left( \sum_j W^{(2,j,i)} h_j + b^{(2,j)} \right)$
- Let $z_i = \sum_j W^{(2,j,i)} h_j + b^{(2,j)}$ so that $o_i = g(z_i)$
- Partial derivative of $C(o, y)$ with respect to $W^{(2,j,i)}$ all evaluated at $(x, y, W, b, h, o)$

\[
\frac{\partial C}{\partial W^{(2,j,i)}} = \frac{\partial o_i}{\partial W^{(2,j,i)}} \frac{\partial C}{\partial o_i} \\
= \frac{\partial z_i}{\partial W^{(2,j,i)}} \frac{\partial g}{\partial z_i} \frac{\partial C}{\partial o_i} \\
= h_j \frac{\partial g}{\partial z_i} \frac{\partial C}{\partial o_i} \\
= h_j g'(z_i) \frac{\partial}{\partial o_i} C(o, y)
\]
Gradient of Single Weight (last layer)

• For example, if we use:
  • sigmoid activation $g = \sigma, \sigma'(t) = \sigma(t)(1 - \sigma(t))$
  • MSE loss: $C(o, y) = \sum_{i=1}^{N}(o_i - y_i)^2$

• Then
  $$\frac{\partial c}{\partial W^{(2,j,i)}} = h_j g'(z_i) \frac{\partial c}{\partial o_i}$$
  $$= h_j g(z_i)(1 - g(z_i))2(o_i - y_i)$$
  $$= h_j o_i(1 - o_i)2(o_i - y_i)$$
Vectorization

• For a single weight, we had:

$$\frac{\partial C}{\partial W^{(2,j,i)}} = h_j o_i (1 - o_i) 2 (o_i - y_i)$$

• Vectorizing, we get

$$\frac{\partial C}{\partial W^{(2)}} = 2h \cdot (o \cdot (1 - o) \cdot (o - y))^T$$

• *Note this is for sigmoid activation, square loss*
What about earlier layers?

Use multivariate chain rule:

\[
\frac{\partial C}{\partial h_i} = \sum_k \left( \frac{\partial C}{\partial o_k} \frac{\partial o_k}{h_i} \right)
\]

\[
\frac{\partial C}{\partial W(1,j,i)} = \frac{\partial C}{\partial h_i} \frac{\partial h_i}{\partial W(1,j,i)}
\]
Backpropagation

\[
\frac{\partial C}{\partial \text{hid}_B_i} = \sum_k \left( \frac{\partial C}{\partial \text{hid}_A_k} \frac{\partial \text{hid}_A_k}{\partial \text{hid}_B_i} \right)
\]

\[
\frac{\partial C}{\partial W^{(1,j,i)}} = \frac{\partial C}{\partial \text{hid}_B_i} \frac{\partial \text{hid}_B_i}{\partial W^{(1,j,i)}}
\]
Back-propagate error signal to get derivatives for learning

Compare outputs with correct answer to get error signal

- Input vector
- Hidden layers
- Outputs
- Input vector
Training Summary

**Training neural nets:**

Loop until convergence:

- for each example \( n \)
  1. Given input \( x^{(n)} \), propagate activity forward \((x^{(n)} \rightarrow h^{(n)} \rightarrow o^{(n)})\)
     (forward pass)
  2. Propagate gradients backward (backward pass)
  3. Update each weight (via gradient descent)
Why is training neural networks so hard?

• Hard to optimize:
  • Not convex
  • Local minima, saddle points, etc...
  • Can take a long time to train

• Architecture choices:
  • How many layers? How many units in each layer?
  • What activation function to use?

• Choice of optimizer:
  • We talked about gradient descent, but there are techniques that improves upon gradient descent
Demo

http://playground.tensorflow.org/