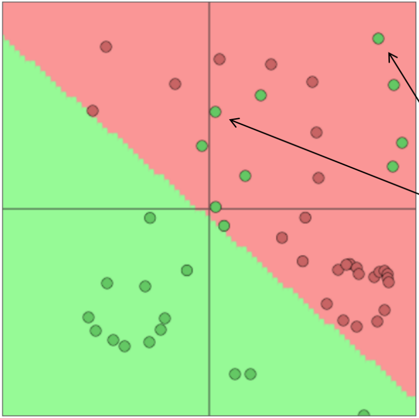
Neural Networks

Acknowledgments: Andrew Ng and Tom Mitchell

June 24, 2016

Logistic Regression is not very powerful

Logistic Regression only gives linear decision boundaries in the original space.

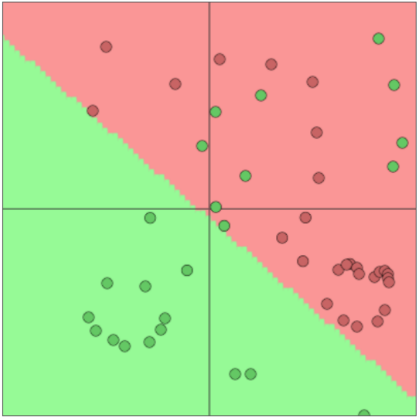


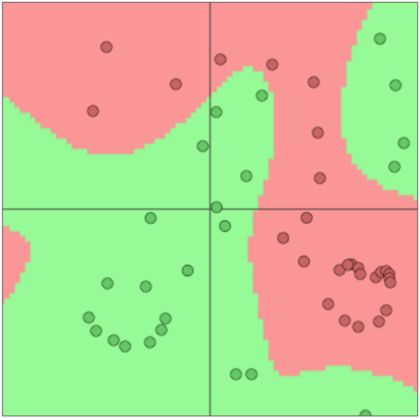
→ Lame when problem is complex

Wouldn't it be cool to get these correct?

Neural Nets for the Win!

Neural networks can learn much more complex functions and non-linear decision boundaries!



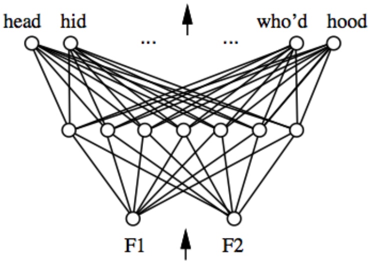


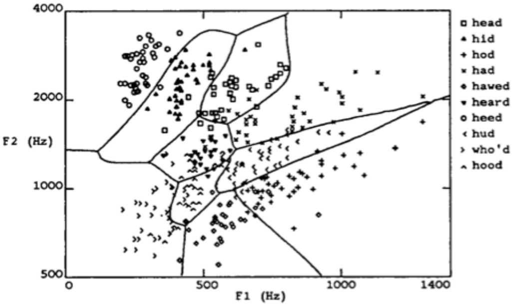
Multilayer Networks

Multilayer networks: capable of expressing a rich variety of non-linear decision surfaces.

Example: the speech recognition task

- Distinguish among 10 possible vowels spoken in the context of "h_d": hid, had, head, hood, etc.
- The input speech signal represented by two numerical parameters obtained from a spectral analysis of the sound.

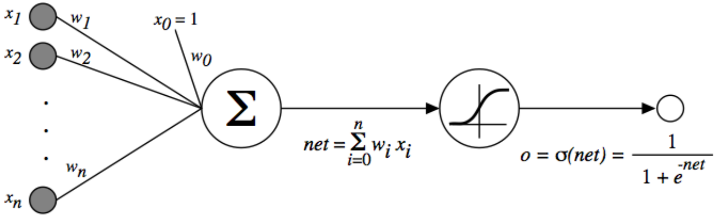




A Differentiable Threshold Unit

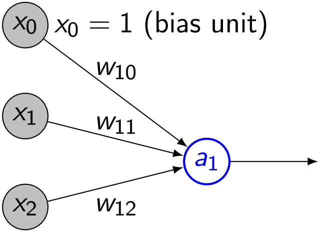
What type of unit to use as the basis for constructing multilayer networks?

- Linear units
  - However, multiple layers of cascaded linear units still produce only linear functions
  - Want networks capable of representing highly non-linear functions.
- Perceptron unit
  - However, its discontinuous threshold makes it un-differentiable, and hence, unsuitable for gradient descent.
- Sigmoid unit
  - whose output is a nonlinear and differentiable function of its inputs.



Sigmoid Unit Representation

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \qquad w = \begin{bmatrix} w_{10} \\ w_{11} \\ w_{12} \end{bmatrix}$$

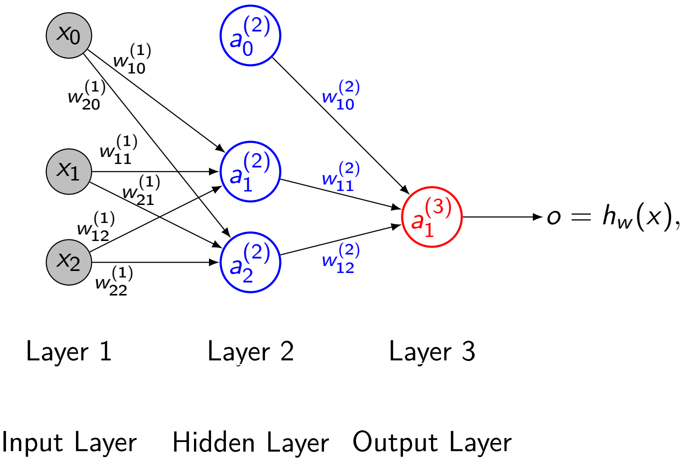


$x_0 = 1$ (bias unit)

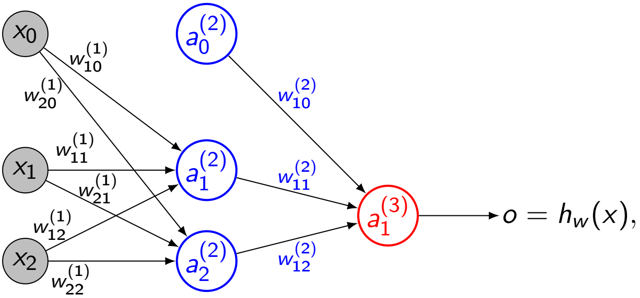
$o = h_w(x)$, $h_w(x) = \frac{1}{1+e^{-w^T x}}$

$a_1 = \sigma(w_{10}x_0 + w_{11}x_1 + w_{12}x_2)$

Neural Network Representation



Neural Network Representation

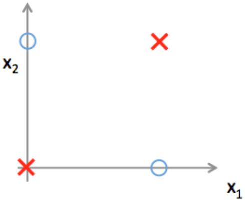


$a_i^{(j)}$ = "activation" of unit i in layer j.

$$a_1^{(2)} = \sigma(w_{10}^{(1)}x_0 + w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2)$$
$$a_2^{(2)} = \sigma(w_{20}^{(1)}x_0 + w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2)$$

$$h_w(x) = a_1^{(3)} = \sigma(w_{10}^{(2)}a_0^{(2)} + w_{11}^{(2)}a_1^{(2)} + w_{12}^{(2)}a_2^{(2)})$$

Non-Linear Classification Example: XNOR

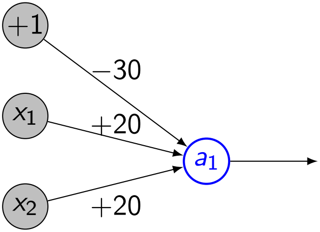


x_1	x_2	$y = x_1$ XNOR x_2
0	0	1
0	1	0
1	0	0
1	1	1

Dataset $\mathcal{D}$:
$e_1 : 0, 0, 1$
$e_2 : 0, 1, 0$
$e_3 : 1, 0, 0$
$e_4 : 1, 1, 1$

x_1 XNOR $x_2 = x_1$ AND x_2 OR $\neg x_1$ AND $\neg x_2$

x_1 AND x_2



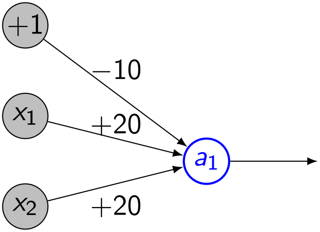
$o = h_w(x)$, $h_w(x) = \frac{1}{1+e^{-w^T x}}$

$$h_w(x) = a_1 = \sigma(-30 + 20x_1 + 20x_2)$$

x_1	x_2	$y = x_1$ AND x_2
0	0	$\sigma(-30) \approx 0$
0	1	$\sigma(-10) \approx 0$
1	0	$\sigma(-10) \approx 0$
1	1	$\sigma(+10) \approx 1$

Thus, $h_w(x) = x_1$ AND x_2.

x_1 OR x_2



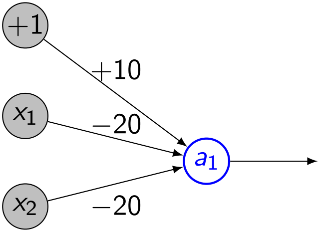
$o = h_w(x)$, $h_w(x) = \frac{1}{1+e^{-w^T x}}$

$h_w(x) = a_1 = \sigma(-10 + 20x_1 + 20x_2)$

x_1	x_2	$y = x_1$ OR x_2
0	0	$\sigma(-10) \approx 0$
0	1	$\sigma(+10) \approx 1$
1	0	$\sigma(+10) \approx 1$
1	1	$\sigma(+30) \approx 1$

Thus, $h_w(x) = x_1$ OR x_2.

$\neg x_1$ AND $\neg x_2$



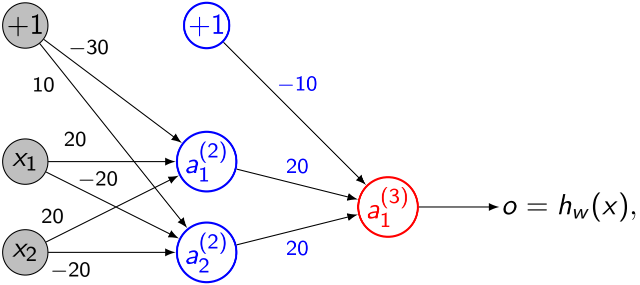
$o = h_w(x)$, $h_w(x) = \frac{1}{1+e^{-w^T x}}$

$$h_w(x) = a_1 = \sigma(+10 - 20x_1 - 20x_2)$$

x_1	x_2	$y = \neg x_1$ AND $\neg x_2$
0	0	$\sigma(+10) \approx 1$
0	1	$\sigma(-10) \approx 0$
1	0	$\sigma(-10) \approx 0$
1	1	$\sigma(-30) \approx 0$

Thus, $h_w(x) = \neg x_1$ AND $\neg x_2$.

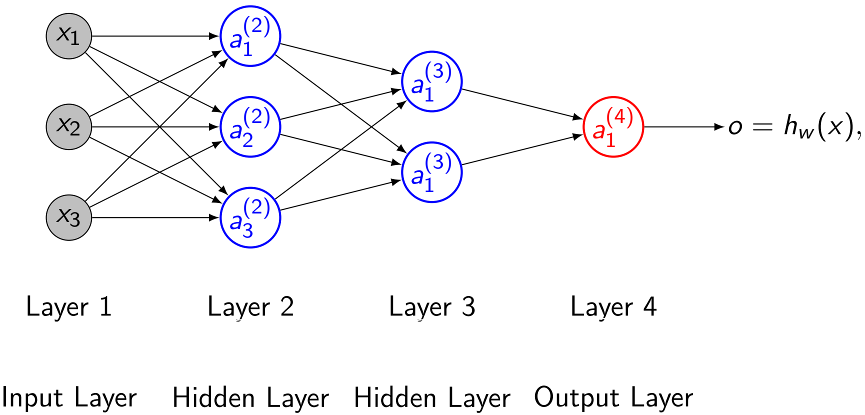
x_1 XNOR x_2



x_1	x_2	$a_1^{(2)} = x_1$ AND x_2	$a_2^{(2)} = \neg x_1$ AND $\neg x_2$	$y = x_1$ XNOR x_2
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Thus, $h_w(x) = x_1$ XNOR x_2.

Other Neural Network Representations



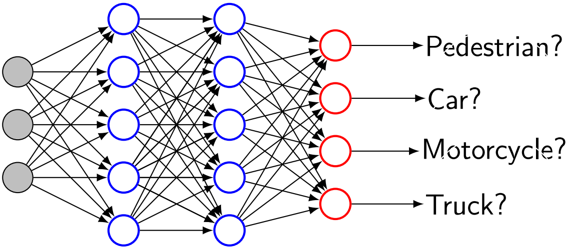
Multiple Output Units: One-vs-all


Pedestrian


Car


Motorcycle


Truck



$y \in R^4$

Want $h_w(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ when pedestrian, $h_w(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, when car, etc.

Learning Neural Networks

We can derive gradient decent rules to train

- One sigmoid unit
- *Multilayer networks* of sigmoid units $\rightarrow$ Backpropagation

Error Gradient for a Sigmoid Unit

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (y_d - o_d)^2 \\
&= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (y_d - o_d)^2 \\
&= \frac{1}{2} \sum_d 2(y_d - o_d) \frac{\partial}{\partial w_i} (y_d - o_d) \\
&= \sum_d (y_d - o_d) \left(- \frac{\partial o_d}{\partial w_i} \right) \\
&= - \sum_d (y_d - o_d) \frac{\partial o_d}{\partial net_d} \frac{\partial net_d}{\partial w_i}
\end{aligned}
$$

Error Gradient for a Sigmoid Unit

But we know:

$$\frac{\partial o_d}{\partial net_d} = \frac{\partial \sigma(net_d)}{\partial net_d} = o_d(1 - o_d)$$

$$\frac{\partial net_d}{\partial w_i} = \frac{\partial(\mathbf{w}^T \cdot \mathbf{x}_d)}{\partial w_i} = x_{i,d}$$

So:

$$\frac{\partial E}{\partial w_i} = -\sum_{d \in D} (y_d - o_d) o_d (1 - o_d) x_{i,d}$$

The Backpropagation Algorithm

- Learns the weights for a multilayer network, given a network with a fixed set of units and interconnections.
- Employs the gradient descent to attempt to minimize the squared error between the network output values and the target values for these outputs.
- The error function for networks with multiple output units:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (y_{kd} - o_{kd})^2,$$

  where *outputs* is the set of output units in the network, y_{kd} and o_{kd} are the target and output values associated with the k^{th} output unit and training example d.
- The error function on training example d:

$$E_d(\mathbf{w}) = \frac{1}{2} \sum_{k \in outputs} (y_{kd} - o_{kd})^2,$$

The Backpropagation Algorithm for Feedforward Networks with Two Layers of Sigmoid Units

Notation: x_{ji} the input from unit i into unit j, w_{ji} the corresponding weight.

Initialize all weights to small random numbers (e.g., between $-.05$ and $.05$).

Until satisfied, do

- For each training example $\mathbf{x}_d$, do

  // Propagate the input forward through the network:

  - Input $\mathbf{x}_d$ to the network and compute the network outputs

  // Propagate the errors backward through the network:

  - For each output unit k, calculate its error term δ_k

    $$\delta_k \leftarrow o_k(1 - o_k)(y_k - o_k)$$

  - For each hidden unit h, calculate its error term δ_h

    $$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k$$

  - Update each network weight w_{ji}

    $$w_{ji} \leftarrow w_{ji} + \Delta w_{ji} \text{ where } \Delta w_{ji} = \eta \delta_j x_{ji}$$

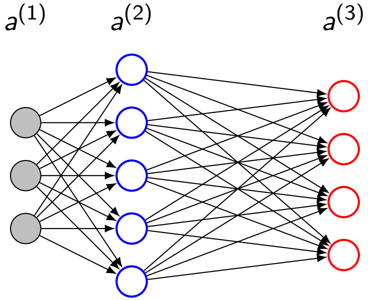
Gradient Computation: Forward Propagation for Networks with Two Layers of Sigmoid Units

Given one training example (x, y):
Forward propagation:

$$a^{(1)} = \mathbf{x}$$

$$a^{(2)} = \sigma(\mathbf{w}^{(1)T} a^{(1)})$$

$$a^{(3)} = \sigma(\mathbf{w}^{(2)T} a^{(2)}) = h_{\mathbf{w}}(\mathbf{x})$$



Gradient Computation: Backpropagation for Networks with Two Layers of Sigmoid Units

$\delta_j^{(l)} =$ "error" of node j in layer l.

For each output unit (in layer 3):

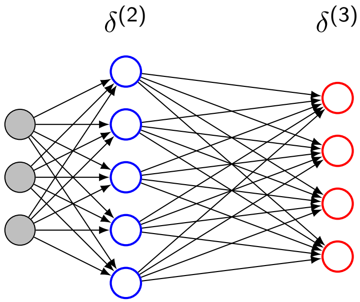
$$\delta_j^{(3)} = a_j^{(3)}(1 - a_j^{(3)})(y_j - a_j^{(3)})$$

Or equivalently,

$$\delta_j^{(3)} = o_j(1 - o_j)(y_j - o_j)$$

For each hidden unit (in layer 2):

$$\delta_j^{(2)} = a_j^{(2)}(1 - a_j^{(2)})\mathbf{w}_j^{(2)T}\delta^{(3)}$$



More on Backpropagation

- The error surface of a multilayer network can have multiple local minima:
  - Gradient descent is only guaranteed to converge to a local minimum (not necessarily global minimum)
  - Backpropagation found to produce excellent results in many real-world applications.
- Backpropagation minimizes error over *training* examples
- Training can take thousands of iterations $\rightarrow$ slow!
- Using network after training is very fast

Generalization, Overfitting, and Stopping Criterion

What is an appropriate condition for terminating the weight update loop?

- Continue training until the error E on the training examples falls below some predetermined threshold.
  - May overfit the training examples at the cost of decreasing generalization accuracy over other unseen examples.
- Successful method for overcoming overfitting
  - Provide a validation set to the algorithm in addition to the training set
  - Monitor the error wrt the validation set, while using the training set to derive the gradient descent search
  - Use the number of iterations that result in the *lowest error over the validation set*

Alternative Error Functions

Gradient descent can be performed for any error function E that is differentiable wrt the parameterized hypothesis space.

- Basic backpropagation algorithm defines E in terms of the sum of squared errors of the network
- Other definitions have been used that incorporate other constraints into the weight-tuning rule:
  - Penalize large weights:

$$E(\mathbf{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (y_{kd} - o_{kd})^2 + \gamma \sum_{i,j} w_{ji}^2$$

  - Minimize the cross entropy of the network wrt the target values (when probabilistic outputs are desired)

$$-\sum_{d \in D} y_d \log o_d + (1 - y_d) \log(1 - o_d)$$

Note: for each definition of E, a new weight-tuning rule for the gradient descent must be derived

Concluding

Training a neural network:

- Pick a network architecture (connectivity pattern between neurons)
  - Number of input units: Dimension of features x_i
  - Number of output units: Number of classes
  - Reasonable default: 1 hidden layer, or if > 1 hidden layer, have the same number of hidden units in every layer
- Run forward propagation and back propagation to learn the network weights.