Representation

Non-linear hypotheses.

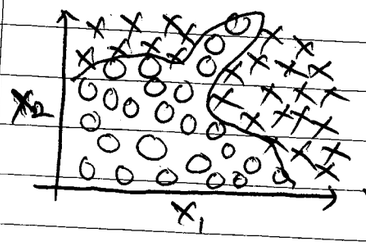
No. _____

Date _____

Example:

Supervised learning classification

Non-linear classification



logistic regression,
(with non-linear features)

$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \dots)$

Can work well as only 2 features — x_1 and x_2

If we have like 100 features

x_1 = size

x_2 = # bedrooms

x_3 = # floors

x_4 = age

...

x_{100}

$n = 100$

will have a lot of term

example:

$x_1^2, x_1x_2, x_1x_3, x_1x_4, \dots x_1x_{100}$

$x_2^2, x_2x_3 \dots\dots$

→ will ≈5000 features

num. of quadratic features,

grow order as n^2 $O(n^2)$

closer $\approx \frac{n^2}{2}$

too lot features may end up over-fitting

Can only include subset of it

=> $x_1^2, x_2^2, \dots x_{100}^2$

Smaller num-features

↳ not enough features and can't fit like graph on top.

If cubic

→ $x_1x_2x_3$, $x_1^2x_2$, $x_{10}x_{11}x_{17}$....

$O(n^3)$ may end up 170,000 features.

↳ will blow up feature space

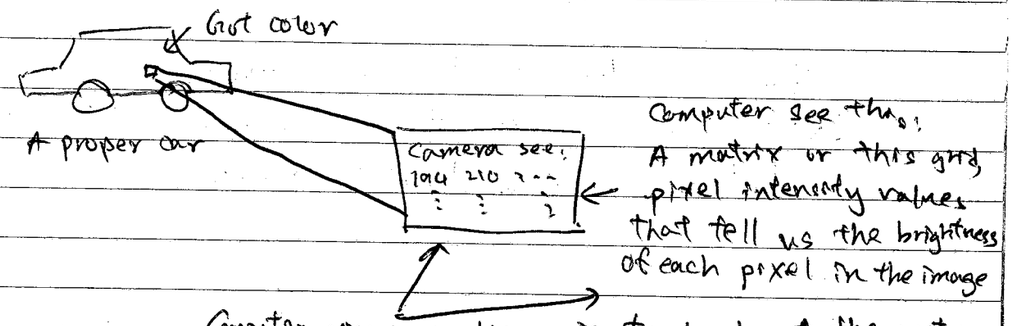
∴ Not a good way to come up additional features which to build non-linear classifier when n large. => But many ML problems, n is pretty large

Example

want to use ML to train a classifier to examine an image and tell us whether ~~or~~ the image is a car or not.

What is this?

You see this:



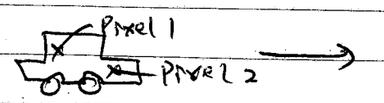
—when we use ML to build a car detector, we come up with label training set

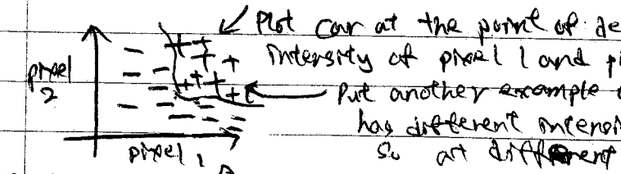
↳ a few label examples of cars

↳ a few label examples of things are not cars

—Then give training set to learning algorithm & train classifier

Example:





Plot car at the point of depending intensity of pixel 1 and pixel 2

Put another example of car, it has different intensity of pixel 1 and 2 so at different plot of figure

+ cars

— non-cars

Cars and non-cars end up lying in different region, we need non-linear hypotheses to separate 2 classes

if only 50 pixel ↓

50×50 pixel images

↳ 2500 pixels

$n = 2500$ (↳ 500 if RGB with colors)

if using grayscale image

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

learn non-linear hypothesis by include all quadratic features

↳ $(x_i \times x_j)$: ≈ 3 million features

Neural Network : Representation

Neurons and the brain

Neural Networks

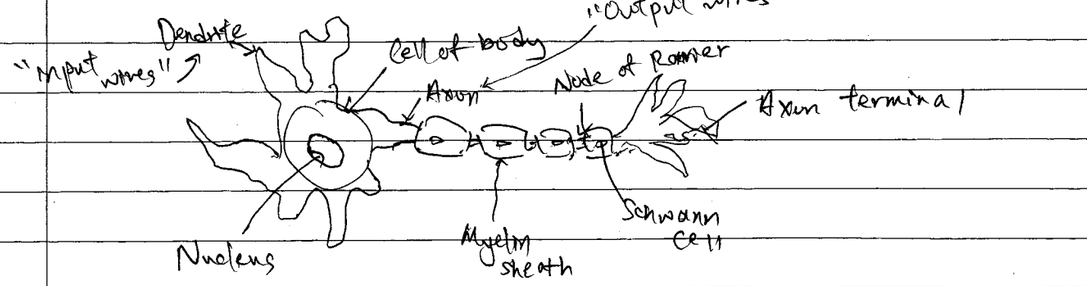
Origins: Algorithms that try to mimic the brain.
Very widely used in 80s and early 90s; popularity diminished in late 90s.
Recent resurgence: State-of-the-art technique for many applications

Brain can learn to do a lot of things.
↳ The Brain does it not by mimicking a lot software but just a single learning algorithm.

Model Representation I

⇒ How we represent neural networks / how we represent hypothesis/ represent our model when using neural networks.

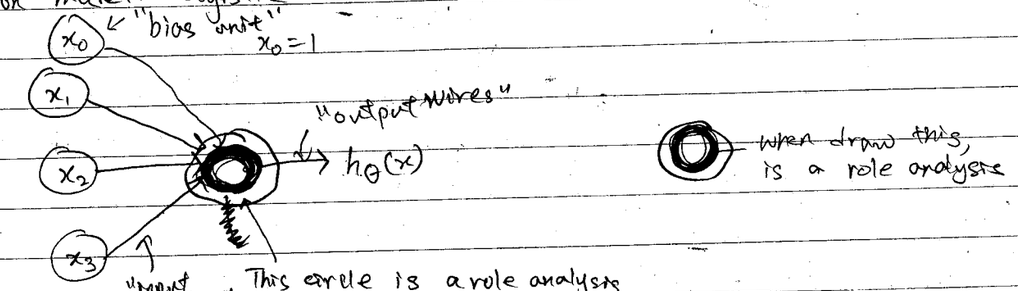
Neuron in the brain



A computational unit get number of inputs through dendrite and does some computation and sent outputs via axon to other neurons in the brain.

In artificial neuron network that we implement in the computer, we going to use a simple model on what a neuron does

Neuron model: Logistic unit

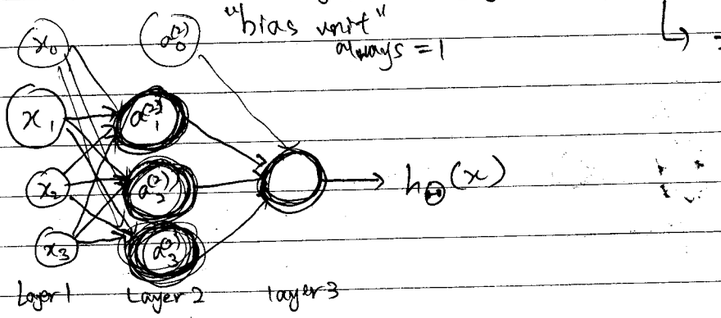


when draw this, is a role analysis

Diagram like this represent a computation of $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$ $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$ ← "weight" (parameter) of the model

An artificial neuron with Sigmoid (logistic) activation function.
↳ $= g(z) = \frac{1}{1+e^{-z}}$

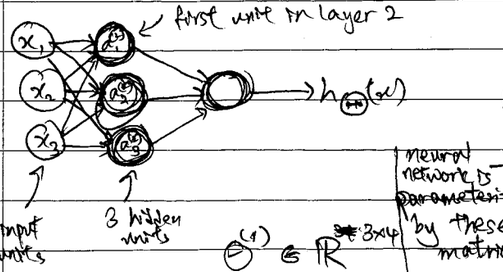


neural network is just a group of different neurons strung together
Layer 1 = input layer
Layer 2 = hidden layer
Layer 3 = output layer
all the layer that are not input/output then is hidden.

Neural Network: Representation

Model representation II

Neural Network



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

$\Theta^{(j)}$ = matrix of weight controlling function mapping from layer j to layer $j+1$

(example: mapping from layer 1 to layer 2 or layer 2 to layer 3)

neural network is parameterized by these matrices

$\Theta^{(1)} \in \mathbb{R}^{3\times 4}$

↑ matrix of parameter governing our mapping from 3 input units to 3 hidden units

Computation represent the diagram

Sigmoid activation function

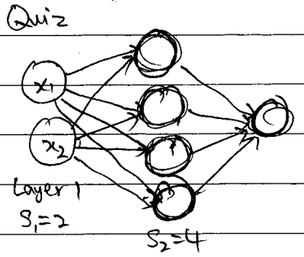
$$a_1^{(2)} = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right)$$
$$a_2^{(2)} = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right)$$
$$a_3^{(2)} = g\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right)$$
$$h_\Theta(x) = a_1^{(3)} = g\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right)$$

$\Theta^{(2)}$: matrix of parameters controls function that maps from hidden units to output unit

Generally:

If network has s_j units in layer j, s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$

Quiz



Dimension of $\Theta^{(1)}$ = $s_{i+1} \times (s_i + 1)$

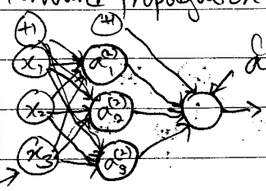
$= s_2 \times (2+1)$

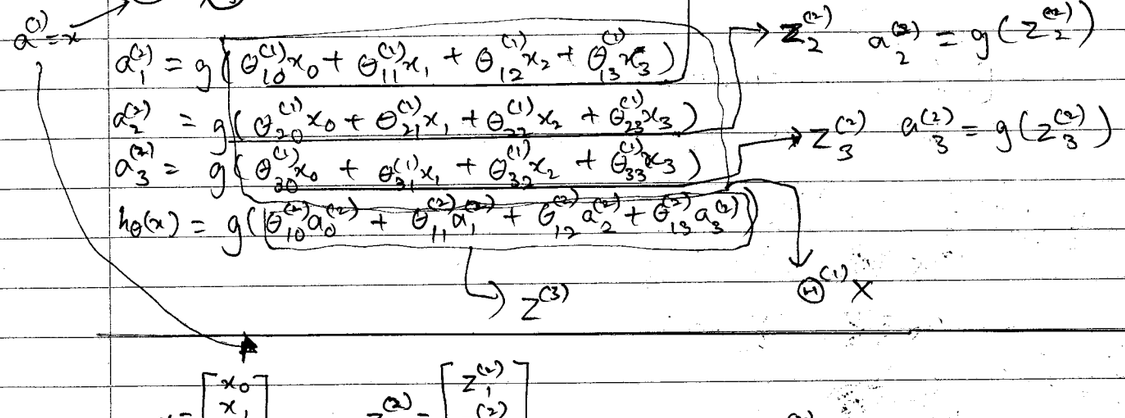
$= 4 \times 3$

- vectorized implementation
- intuition on why these neural network representation

from input ⇒ hidden ⇒ output

Forward propagation: Vectorized implementation





$a_1^{(2)} = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right)$

$a_2^{(2)} = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right)$

$a_3^{(2)} = g\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right)$

$h_\Theta(x) = g\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right)$

$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$ $z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$ $a^{(2)} = \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}$

① $z^{(2)} = \Theta^{(1)}a^{(1)}$ $\left[z_i^{(j)} = \Theta^{(j-1)}a^{(j-1)}\right]$

② $a^{(2)} = g(z^{(2)})$ $\left[a^{(j)} = g(z^{(j)})\right]$

$a^{(2)} \in \mathbb{R}^3$, $z^{(2)} \in \mathbb{R}^3$

activation g apply the sigmoid function element-wise to each of the $z^{(2)}$ elements

For the bias unit, have to

Add $a_0^{(2)} = 1$ ⟹ After adding extra → $a^{(2)} \in \mathbb{R}^4$

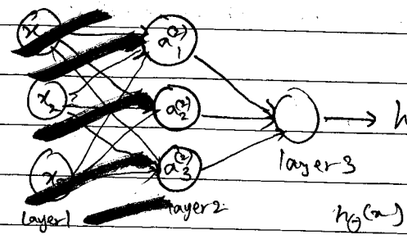
③ $z^{(3)} = \Theta^{(2)}a^{(2)}$ $\left[z^{(j+1)} = \Theta^{(j)}a^{(j)}\right]$

④ $h_\Theta(x) = a^{(3)} = g(z^{(3)})$ $\left[h_\Theta(x) = a^{(j+1)} = g(z^{(j+1)})\right]$

↳ more efficient way of computing $h_\Theta(x)$

Neural Network learning its own features

No. _____
Date _____

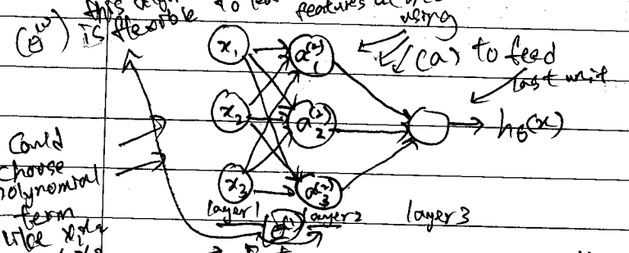


$$h_\Theta(x) = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

cover up this part

is just like [logistic regression] exactly same

but replace x_1, x_2, x_3 with $a_1^{(2)}, a_2^{(2)}, a_3^{(2)}$



$\Theta^{(1)}$ is flexible this algorithm to learn whatever features at once using $a^{(2)}$ to feed last unit

Features a_1, a_2, a_3, they themselves learn as function of input

function that map from layer 1 → layer 2 is determined by other set of parameters $\Theta^{(1)}$

instead of being constrained to feed features x_1, x_2, x_3 into logistic regression, it learns its own features a_1, a_2, a_3 to feed into logistic regression

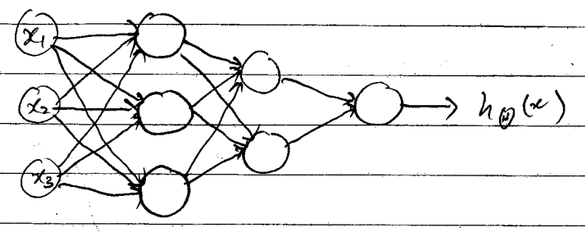
Depends the parameter for $\Theta^{(1)}$, will learn some interesting and complex feature.

∴ Parameter 決定 好的 feature

The way that neural networks are connected ⇒ architecture

No. _____
Date _____

Other network architectures



layer 1 layer 2 layer 3 layer 4 → come out with non-linear hypotheses

Compute x → take and compute more complex feature → more complex features from layer 3

input layer — Hidden layer — output layer

Neural Networks:

Representation

Examples and intuitions I

No.

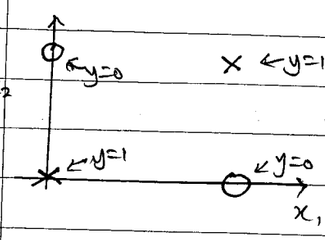
Date

Neural Network is used to learn complex non-linear hypotheses

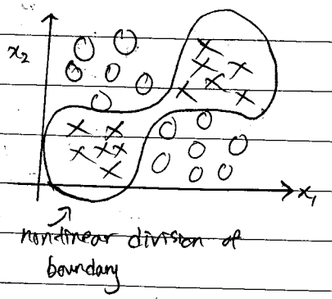
Non-linear classification example: XOR / XNOR

x_1, x_2 are binary (0 or 1)

x - positive
O - negative



Simplified version of this →



$y = x_1$ XOR x_2

→ x_1 XNOR x_2

↳ NOT (x_1 XOR x_2)

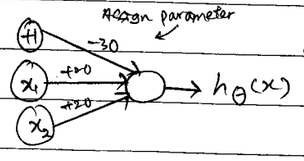
Purpose:
Get neural network to fit this training set

Before getting to the hard part, we take easy example first.

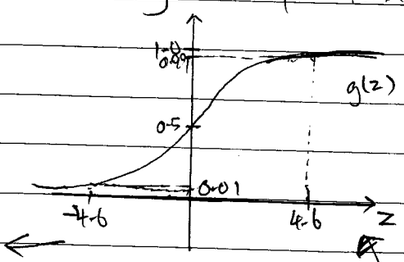
Simple example: AND

$x_1, x_2 \in \{0, 1\}$

$y = x_1$ AND x_2



things will this single neuron network compute



refer to the graph

$h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$

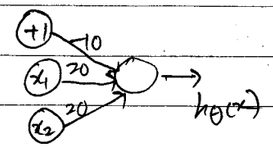
$\Theta^{(1)}_{10}$ $\Theta^{(1)}_{11}$ $\Theta^{(1)}_{12}$

x_1	x_2	$h_{\Theta^{(1)}}(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

$h_\Theta(x) \approx x_1$ AND x_2

writing out this kind of truth table, we manage to figure what's logical function our neural network computes.

Suppose x_1 and x_2 are binary valued (0 or 1) - what boolean function does the network shown below (approximately) compute?



x_1	x_2	$h_\Theta(x)$
0	0	$g(-10) \approx 0$
1	0	$g(10) \approx 1$
0	1	$g(10) \approx 1$
1	1	$g(30) \approx 1$

$\therefore x_1$ OR x_2

Neural Network

Representation:

Examples and intuitions II

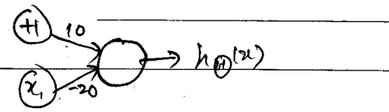
No.

Date

* Neural Network can ~~comp~~ be used to compute x_1 AND x_2, x_1 OR x_2 when x_1 and x_2 are binary $x_1, x_2 \in \{0,1\}$

We can have a network to compute negation.

Negation: NOT x_1



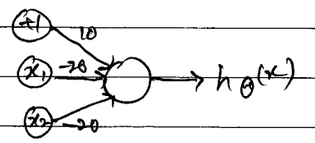
$h_\Theta(x) = g(10 - 20x_1)$

x_1	$h_\Theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

negate x_1

General idea: Put large negative weight in front of variable you want to negate.

(NOT x_1) AND (NOT x_2)

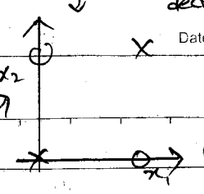


$h_\Theta(x) = g(10 - 20x_1 - 20x_2)$

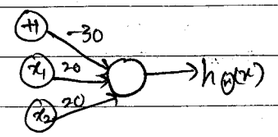
x_1	x_2	$h_\Theta(x)$
0	0	$g(10) \approx 1$
1	0	$g(-10) \approx 0$
0	1	$g(-10) \approx 0$
1	1	$g(-30) \approx 0$

=1 if and only if $x_1 = x_2 = 0$

to separate need non-linear decision boundary

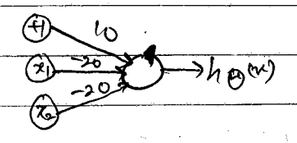


Putting it together: x_1 XNOR x_2



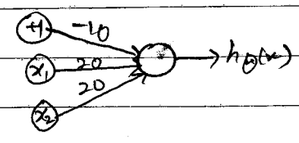
x_1 AND x_2

$\Theta^{(1)} = [-30 \quad 20 \quad 20]$



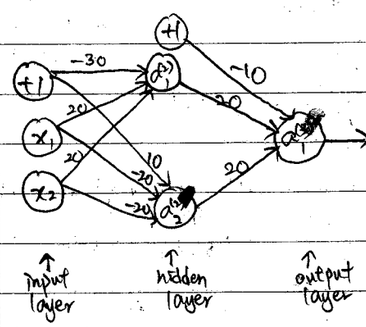
(NOT x_1) AND (NOT x_2)

$\Theta^{(1)} = [10 \quad -20 \quad -20]$



x_1 OR x_2

$\Theta^{(1)} = [-10 \quad 20 \quad 20]$



x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	1	1

combine

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \end{bmatrix} \rightarrow [a^{(3)}] \rightarrow h_\Theta(x)$$

transition between 2nd and 3rd layer

transition between 1st and 2nd layer

$\Theta^{(1)} = \begin{bmatrix} -30 & 20 & 20 \\ 10 & -20 & -20 \end{bmatrix}$

$\Theta^{(2)} = [-10 \quad 20 \quad 20]$

values for all nodes:

$a^{(2)} = g(\Theta^{(1)} \cdot x)$

$a^{(3)} = g(\Theta^{(2)} \cdot a^{(2)})$

$h_\Theta(x) = a^{(3)}$

Neural Networks:

Representation

Multi-class classification

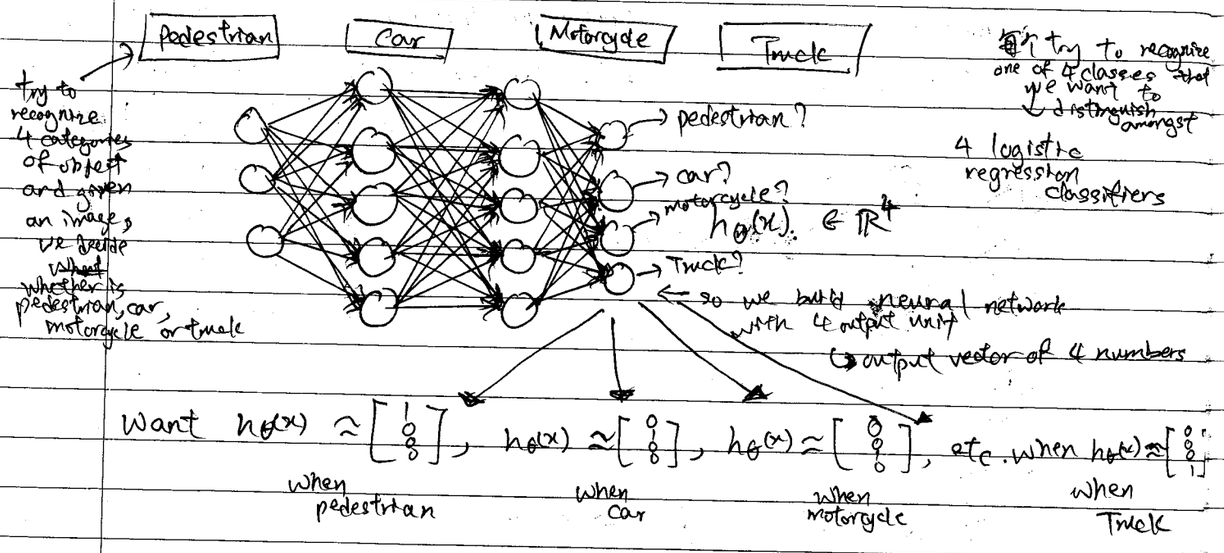
No. _____

Date _____

Handwritten digit recognition => multi-class classification problem

↳ 10 possible categories to recognize the digits from 0–9

Multiple output units: One-vs-all

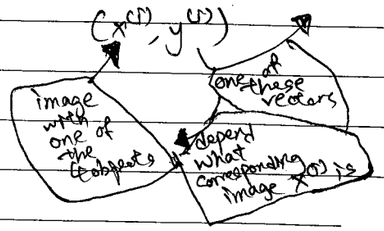


Like one-vs-all method in logistic regression

Training set: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots (x^{(m)}, y^{(m)})$

$y^{(i)}$ one of $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian, car, motorcycle, truck



Get a way to get neural network to output value where $h_\theta(x^{(i)}) \approx y^{(i)}$

NOT using this: $y = \{1, 2, 3, 4\}$

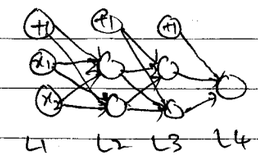
$\mathbb{R}^4$

Quiz: you have multi-class classification problem with 10 classes, your neural networks has 3 layers, and the hidden layer has 5 units, Using one-vs-all method, how many element does $\Theta^{(2)}$ have?

Answer: 60 (+ bias unit)

Quiz

Consider neural network below, which equation computes activation $a_1^{(3)}$? Note: $g(z)$ is sigmoid activation function.

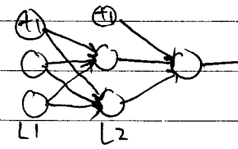


$$a_1^{(3)} = g(\Theta_{1,0}^{(2)} a_0^{(2)} + \Theta_{1,1}^{(2)} a_1^{(2)} + \Theta_{1,2}^{(2)} a_2^{(2)})$$

You are using neural network below and have learned parameters.

$\Theta^{(1)} = \begin{bmatrix}1 & -1.5 & 3.7\\ 1 & 5 & 1 & 2.3\end{bmatrix}$ (used to compute $a^{(2)}$) and $\Theta^{(2)} = [1 \quad 0.6 \quad -0.8]$ (used to compute $a^{(3)}$ as a function of $a^{(2)}$).

Suppose you swap parameter for the first hidden layer between its two units so $\Theta^{(1)} = \begin{bmatrix}1 & 5 & 1 & 2.3\\ & -1.5 & 3.7\end{bmatrix}$ and also swap output layer so $\Theta^{(2)} = [1 \quad -0.8 \quad 0.6]$. How will this change value of output $h_\theta(x)$?



It will stay the same.

Setup looks like:

$$\begin{bmatrix}x_0\\x_1\\x_2\\\vdots\\x_n\end{bmatrix} \rightarrow \begin{bmatrix}a_0^{(2)}\\a_1^{(2)}\\a_2^{(2)}\\\dots\end{bmatrix} \rightarrow \begin{bmatrix}a_0^{(3)}\\a_1^{(3)}\\a_2^{(3)}\\\dots\end{bmatrix} \rightarrow \dots \rightarrow \begin{bmatrix}h_\theta(x)_1\\h_\theta(x)_2\\h_\theta(x)_3\\h_\theta(x)_4\end{bmatrix}$$

our resulting hypothesis for one set look like this:

$$h_\theta(x) = \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$$

in which case our resulting class is the third one down, or $h_\theta(x)_3$, which represents motorcycle.

~~Neural Networks~~

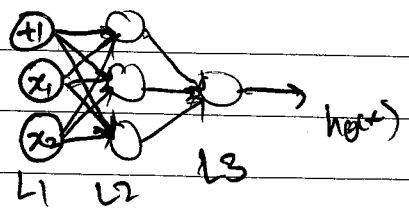
~~learning~~

~~cost function~~

No.

Date

Quiz



You want to compute activations of hidden layer $a^{(2)} \in \mathbb{R}^3$.
One way to do so is the following octave code.

```
%Theta1 is Theta with superscript "(1)" from lecture
%ie, the matrix of parameters for the mapping from layer 1 (input) to layer
%Theta1 has size 3x3
% Assume 'sigmoid' is a built-in function to compute 1/(1+exp(-z))
a2 = zeros(3,1);
for i=1:3
    for j=1:3
      a2(i) = a2(i) + x(j) * Theta1(i,j);
    end
a2(i) = sigmoid(a2(i));
end
```

which one vectorized implementation, correctly compute $a^{(2)}$.

```
z = Theta1 * x ; a2 = sigmoid(z);
```

Correct statement in neural network.

1) If a neural network is overfitting the data, can increase the regularization parameter λ. [larger value of λ will shrink the magnitude of parameter Θ, thereby reducing the chance of overfitting data.
2) Output of a neural network are not probabilities, their sum $(a_1^{(3)} + a_2^{(3)} + a_3^{(3)})$ need not be 1.
3) The activation values of the hidden units in a neural network, with sigmoid activation function applied at every layer, are always in range(0,1). The activation function $g(z) = \frac{1}{1+\exp(-z)}$ has range of (0,1).