Deep Learning

Jin Xie <jin.xie@uky.edu>

Materials mostly by Andrew Ng

March 8, 2018

Outline

Outline

Introduction

Vectorization

Forward Propagation

Backpropagation

CNN

Formulas in this note

- **Features**: $x^{(i)}$
- "Output" or **target** variable: $y^{(i)}$
- A pair $(x^{(i)}, y^{(i)})$ is called a **training example**
- A list of m training examples $\{(x^{(i)}, y^{(i)}) : i = 1, \dots, m\}$ is called a **training set**

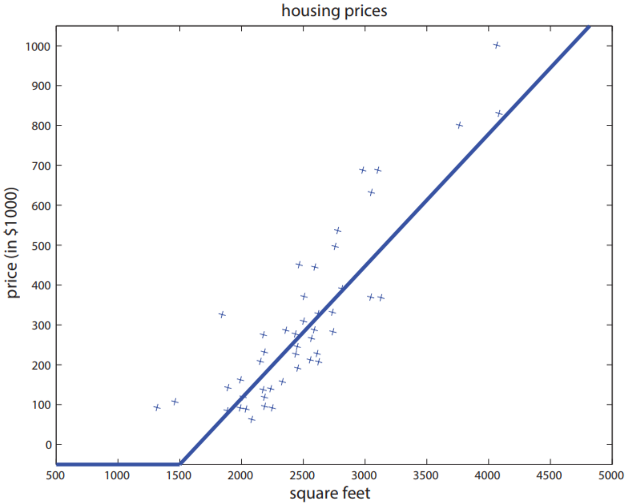
A Single Neuron

- **Problem**: predict the house price y using the house square feet x.
- **Goal**: Find a function **f(x)** ($f : x \to y$) as the prediction of y.
- One simple option is

$$f(x) = \max(ax + b, 0)$$

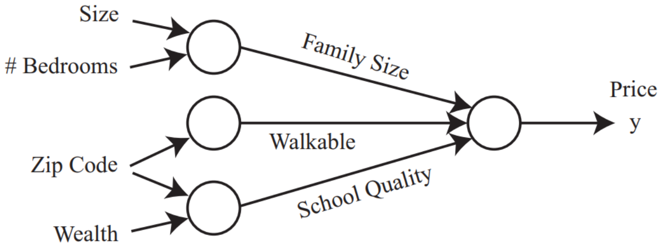
for some coefficient a and b. It's just a single "**neuron**" network. This function is called **ReLU** or rectified linear unit.

ReLU


housing prices
price (in $1000)
square feet
1000
900
800
700
600
500
400
300
200
100
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000

"Stack" Neurons

A more complex neural network may "stack" several neurons together. Each neuron passes its output as input into the next neuron, resulting in a more complex function.



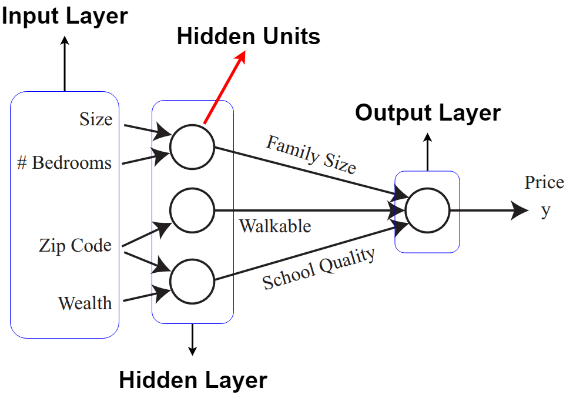
A neuron takes input from previous neurons or ground inputs. The three "internal" neurons are called **hidden units**.

Outline

Black Box

Often times, the neural network will discover complex features which are very useful for predicting the output. However, it may be difficult for a human to understand since it does not have a “common” meaning. This is why some people refer to neural networks as a **block box**, as it can be difficult to understand the features it has invented.

Representation



The hidden layer is called “hidden” because we do not have the ground truth/training value for the hidden units. In contrast to the input and output layers, we have the ground truth values from $(x^{(i)}, y^{(i)})$.

Representation

- Use letter a as the neuron's "**activation**".
- Let $a_j^{[\ell]}$ denote output value of the j^{th} neuron in ℓ^{th} layer. Then $a_1^{[0]} = x_1$.
- Let foo$^{[1]}$ denote anything associated with layer 1.
- Let $x^{(i)}$ refer to i^{th} training example.

Computation

- For a single neuron, $a = g(x)$, where $g(x)$ is some function of x such as

$$g(x) = \frac{1}{1 + \exp(-w^T x)}.$$

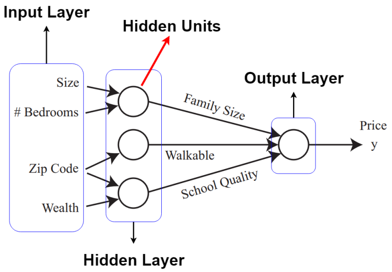
- We break $g(x)$ into 2 distinct computations:
  (1) $z = w^T x + b$
  (2) $a = g(z)$, where $g(z)$ is some activation function such as

$$g(z) = \frac{1}{1 + e^{-z}} \qquad \text{(sigmoid)}$$

$$g(z) = \max(z, 0) \qquad \text{(ReLU)}$$

$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \qquad \text{(tanh)}$$

Computation



For the first hidden layer:

$$z_1^{[1]} = W_1^{[1]T} x + b_1^{[1]} \quad \text{and} \quad a_1^{[1]} = g(z_1^{[1]})$$
$$z_2^{[1]} = W_2^{[1]T} x + b_2^{[1]} \quad \text{and} \quad a_2^{[1]} = g(z_2^{[1]})$$
$$z_3^{[1]} = W_3^{[1]T} x + b_3^{[1]} \quad \text{and} \quad a_3^{[1]} = g(z_3^{[1]})$$

Introduction

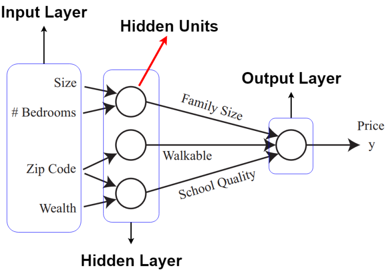
Vectorization

Forward Propagation

Backpropagation

CNN

Computation



For the output layer:

$$z_1^{[2]} = W_1^{[2]^T} x + b_1^{[2]} \quad \text{and} \quad a_1^{[2]} = g(z_1^{[2]})$$

Vectorization

- W is a matrix of parameters and W_1 refers to the first row of this matrix. $W_i^{[1]} \in \mathbb{R}^4$ and $W_1^{[2]} \in \mathbb{R}^3$.
- Vectorize the whole process:

$$\underbrace{\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \end{bmatrix}}_{z^{[1]} \in \mathbb{R}^{3\times 1}} = \underbrace{\begin{bmatrix} {W_1^{[1]}}^T \\ {W_2^{[1]}}^T \\ {W_3^{[1]}}^T \end{bmatrix}}_{W^{[1]} \in \mathbb{R}^{3\times 4}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{x \in \mathbb{R}^{4\times 1}} + \underbrace{\begin{bmatrix} {b_1^{[1]}}^T \\ {b_2^{[1]}}^T \\ {b_3^{[1]}}^T \end{bmatrix}}_{b^{[1]} \in \mathbb{R}^{3\times 1}}$$

That is

$$z^{[1]} = W^{[1]}x + b^{[1]}.$$

- To compute $a^{[1]} = (a_1^{[1]}, a_2^{[1]}, a_3^{[1]})^T$ without a loop, we need some vectorized libraries in Python or R (default). So we can directly calculate

$$a^{[1]} = g(z^{[1]}).$$

Vectorization Over Training Examples

For i^{th} training example,

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]},$$

$$a^{[1](i)} = g(z^{[1](i)}),$$

where $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, x_4^{(i)})^T$ $(i = 1, \ldots, m)$.

Denote

$$X = \left[x^{(1)}, x^{(2)}, \cdots, x^{(m)}\right]_{4\times m}.$$

Then

$$Z^{[1]} = \left[z^{[1](1)}, z^{[1](2)}, z^{[1](3)}\right]_{3\times m} = W^{[1]}X + b^{[1]}.$$

Question: How to compute

$$A^{(1)} = g(Z^{[1]})?$$

Introduction

Vectorization

Forward Propagation

Backpropagation

CNN

Outline

Forward Propagation

For just one training example, let $a^{[0]} = x$. Suppose we have layer $\ell = 1, 2, \ldots, N$, where N is the number of layers in the network. Then we have

1. $z^{[\ell]} = W^{[\ell]} a^{[\ell-1]} + b^{[\ell]}$
2. $a^{[\ell]} = g^{[\ell]}(z^{[\ell]})$

We assume $g^{[1]} = g^{[2]} = \cdots = g^{[N-1]} \neq g^{[N]}$. For regression, $g^{[N]}(x) = x$. For binary classification, $g^{[N]}(x) =$ sigmoid. For multiclass calssification, $g^{[N]}(x) =$ softmax.

Parameter Initialization:

- Do not use zeros as the initial values for parameters. Why?
- Typically, we randomly initialize the parameters to small values such as from $N(0, 0.1)$.

In practice, there is something better than random initializations. It is called Xavier/He initialization.

Forward Propagation

After all the first N layers calculations, we have $a^{[N]}$, i.e. $\hat{y}$.

For different problems, we could use different losses.

- Regression:
$$L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2.$$
- Binary classification using logistic regression:
$$L(\hat{y}, y) = -(y \log \hat{y} - (1 - y) \log(1 - \hat{y})).$$
- Multi-class classification using softmax:
$$L(\hat{y}, y) = -\sum_{j=1}^{k} \mathbf{I}(y = j) \log \hat{y}_j.$$
- Multi-class classification using cross entropy:
$$L(\hat{y}, y) = -\sum_{i=1}^{k} y_j \log \hat{y}_j.$$

Outline

Backpropagation

Goal: Find $W^{[\ell]}$, $b^{[\ell]}$ for $\ell = 1, 2, \ldots, N$ that maximize the loss L.

Solution: Gradient descent.

For a learning rate of α and any given layer index ℓ, we update the parameters through:

$$W^{[\ell]} = W^{[\ell]} - \alpha \frac{\partial L}{\partial W^{[\ell]}}$$
$$b^{[\ell]} = b^{[\ell]} - \alpha \frac{\partial L}{\partial b^{[\ell]}}$$

Backpropagation

First, we define $\delta^{[\ell]} = \nabla_{z^{[\ell]}} L(\hat{y}, y)$.

1. For output layer N,

$$\delta^{[N]} = \nabla_{\hat{y}} L(\hat{y}, y) \circ (g^{[N]})'(z^{[N]}).$$

   Note $(g^{[N]})'(z^{[N]})$ is the elementwise derivative w.r.t. $z^{[N]}$.

2. For $\ell = N-1, N-2, \ldots, 1$, we have

$$\delta^{[\ell]} = ({W^{[\ell+1]}}^T \delta^{[\ell+1]}) \circ (g^{[\ell]})'(z^{[\ell]}).$$

3. Finally, we can compute the gradients for any layer ℓ as

$$\frac{\partial L}{\partial W^{[\ell]}} = \delta^{[\ell]} {a^{[\ell-1]}}^T$$
$$\frac{\partial L}{\partial b^{[\ell]}} = \delta^{[\ell]}$$

Outline

1. Introduction
2. Vectorization
3. Forward Propagation
4. Backpropagation
5. **CNN**

Color Image

Color images can be represented as volume, that is three-channels (red-green-blue) or three matrices stacked on each other.

Color Image

column →

row ↓

R

0.92	0.93	0.94	0.97	0.62	0.37	0.85	0.97	0.93	0.92	0.99
0.95	0.89	0.82	0.89	0.56	0.31	0.75	0.92	0.81	0.95	0.91
0.89	0.72	0.51	0.55	0.51	0.42	0.57	0.41	0.49	0.91	0.92
0.96	0.95	0.88	0.94	0.56	0.46	0.91	0.87	0.90	0.97	0.95
0.71	0.81	0.81	0.87	0.57	0.37	0.80	0.88	0.89	0.79	0.85
0.49	0.62	0.60	0.58	0.50	0.60	0.58	0.50	0.61	0.45	0.33
0.86	0.84	0.74	0.58	0.51	0.39	0.73	0.92	0.91	0.49	0.74
0.96	0.67	0.54	0.85	0.48	0.37	0.88	0.90	0.94	0.82	0.93
0.69	0.49	0.56	0.66	0.43	0.42	0.77	0.73	0.71	0.90	0.99
0.79	0.73	0.90	0.67	0.33	0.61	0.69	0.79	0.73	0.93	0.97
0.91	0.94	0.89	0.49	0.41	0.78	0.78	0.77	0.89	0.99	0.93

G

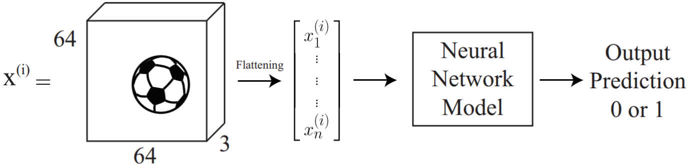
									0.92	0.99
									0.95	0.91
									0.91	0.92
									0.97	0.95
									0.79	0.85
									0.45	0.33
									0.49	0.74
									0.82	0.93
									0.90	0.99
0.79	0.73	0.90	0.67	0.33	0.61	0.69	0.79	0.73	0.93	0.97
0.91	0.94	0.89	0.49	0.41	0.78	0.78	0.77	0.89	0.99	0.93

B

									0.92	0.99
									0.95	0.91
									0.91	0.92
									0.97	0.95
									0.79	0.85
									0.45	0.33
									0.49	0.74
									0.82	0.93
									0.90	0.99
0.79	0.73	0.90	0.67	0.33	0.61	0.69	0.79	0.73	0.93	0.97
0.91	0.94	0.89	0.49	0.41	0.78	0.78	0.77	0.89	0.99	0.93

Convolutional Neural Network

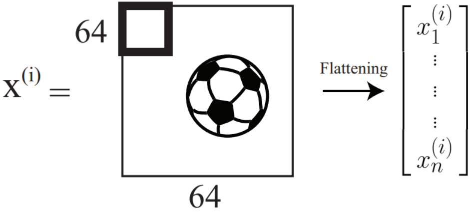
We have a 64 $\times$ 64 $\times$ 3 image containing a soccer. It is **flattened** into a single vector containing 12,288 elements.



Convolutional Neural Network

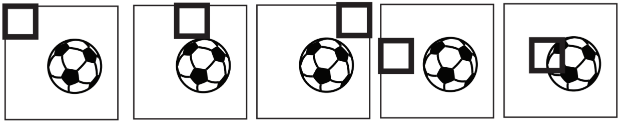
Suppose θ is not vector but instead is a matrix. For the soccer ball example, suppose $\theta \in \mathbb{R}^{4\times 4}$. Each activation *a* or neuron, we compute the element-wise product between θ and $x_{1:4,1:4}$, where $x_{1:4,1:4}$ indicates the top left 4×4 region in a channel. Formally:

$$a = \sum_{i=1}^{4} \sum_{j=1}^{4} \theta_{ij} x_{ij}$$



Convolutional Neural Network

We move this window across each row of the image.



Finally, we will have different features than the original pixel values. And we can use several different θ matrix to scan the image.