Back propagation Neural Network

A simple introduction

Yuguang Yue

October 2, 2017

Table of contents

Introduction

Neural Network

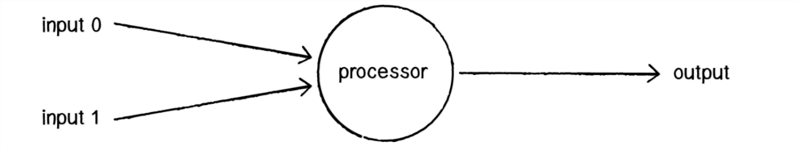
Neural Network is a popular machine learning algorithm. It is first proposed on 1943 by a neuroscientist Warren S. McCulloch, and a logician, Walter Pitts. It has been widely used in various fields, such as

```
Classification
Regression
Clustering
Filtering
```

The name of **Neural Network** comes from the analog to human brain's biological neural network.

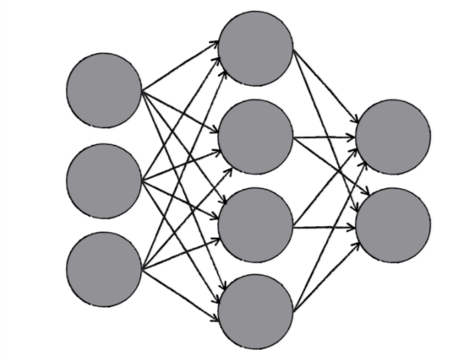
Perception

A perception resemble to a single neuron. It is a simplest neural network consists of more inputs, one processor and one outcome. When a processor receives inputs, it will allocate weights to each input, and sum them all to calculate out the output.



Network

A network consists of many perceptions with many layers. The system functions from left to right. Each perception proceeds the inputs and pass its outcome to next layer.



Notation

Notations to be used

- L: number of total layers
- S_l: number of perceptions in l^{th} layer
- w^l_{ji}: weights from perception i in $(l-1)^{th}$ layer to perception j in l^{th} layer
- z^l_i: weighted input of perception i in l^{th} layer
- $\sigma(\cdot)$: the activation function on each perception
- a^l_i: activation of perception i in l^{th} layer
- b^l_i: bias term of perception i in l^{th} layer
- C: cost function (**Two requirements**: 1. the cost function can be written as an average of $C = \frac{1}{n}\sum_x C_x$; 2. can be represented as a function of a^L) e.g, $C = \frac{1}{2}||y - a^L||^2$
- $\odot$: Hadamard product, element-wise product operator. e.g, $[1,2]^T \odot [3,4]^T = [3,8]^T$

Vectorized notations

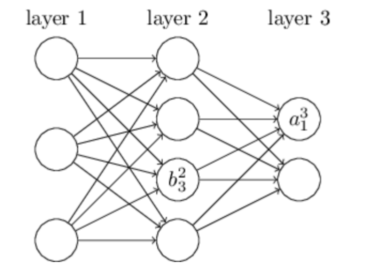
- a^l: the vector of activations in l^{th} layer. e.g $a^3 = [a^3_1, a^3_2, \cdots, a^3_{S_3}]^T$
- b^l: the vector of bias in l^{th} layer
- z^l: the vector of weighted inputs in l^{th} layer
- W^l: the weights matrix in l^{th} layer. The elements are placed with accordance to their subscripts. e.g

$$W^2_{S_2 \times S_1} = \begin{bmatrix} w^2_{11} & w^2_{12} \\ w^2_{21} & w^2_{22} \\ w^2_{31} & w^2_{32} \end{bmatrix}$$

Forward Step

Scheme

Use a neural network with one hidden layer as an example



$$a_1^3 = \sigma(\textstyle\sum_k w_{1k}^3 a_k^2 + b_1^3)$$

$$(\textit{forward step}) \quad a^l = \sigma(z^l) = \sigma(W^l a^{l-1} + b^l)$$

Backward Step

We want to use gradient descent to update w^l_{ji} and b^l_i iteratively, such that

$$w^l_{ji,(t)} = w^l_{ji,(t-1)} - \alpha(t)\frac{\partial C}{\partial w^l_{ji}}$$

$$b^l_{i,(t)} = b^l_{i,(t-1)} - \beta(t)\frac{\partial C}{\partial b^l_i}$$

where t denotes the number of iterations. The way we calculate the gradient is from the last layer to first layer, so the name is backward propagation.

Backpropagation process

Define measure of error:

$$\delta_i^l \equiv \frac{\partial C}{\partial z_i^l}$$

we can get four equations for back propagation update

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

$$\delta^l = ((W^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

$$\frac{\partial C}{\partial b_i^l} = \delta_i^l$$

$$\frac{\partial C}{\partial w_{ji}^l} = a_i^{l-1} \delta_j^l$$

Proof of four equations

$$\delta_i^L = \frac{\partial C}{\partial z_i^L} = \sum_k \frac{\partial C}{\partial a_k^L}\frac{\partial a_k^L}{\partial z_i^L} = \frac{\partial C}{\partial a_i^L}\frac{\partial a_i^L}{\partial z_i^L} = \frac{\partial C}{\partial a_j^L}\sigma'(z_i^L)$$

$$\Rightarrow \delta^L = \nabla_a C \odot \sigma'(z^l)$$

$$\delta_i^l = \frac{\partial C}{\partial z_i^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}}\frac{\partial a_k^{l+1}}{\partial z_i^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_i^l}\delta_k^{l+1}$$

We have the relationship between z_k^{l+1} and z_k^l, which is

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1}\sigma(z_j^l) + b_k^{l+1} \tag{1}$$

$$\Rightarrow \frac{\partial z_k^{l+1}}{\partial z_i^l} = w_{ki}^{l+1}\sigma'(z_i^l)$$

and we get

$$\delta_i^l = \sum_k w_{ki}^{l+1}\sigma'(z_i^l)$$

$$\delta^l = ((W^{l+1})^T\delta^{l+1}) \odot \sigma'(z^l)$$

Proof of four equations (cont)

From formula (1), we have

$$\begin{aligned} \frac{\partial C}{\partial b_i^l} &= \sum_k \frac{\partial C}{\partial z_k^l} \frac{\partial z_k^l}{\partial b_i^l} = \frac{\partial C}{\partial z_i^l} = \delta_i^l \\ \Rightarrow \frac{\partial C}{\partial b^l} &= \delta^l \end{aligned}$$

for the last equation,

$$\begin{aligned} \frac{\partial C}{\partial w_{ji}^l} &= \sum_k \frac{\partial C}{\partial z_k^l} \frac{\partial z_k^l}{\partial w_{ji}^l} = \frac{\partial C}{\partial z_j^l} \sigma(z_i^{l-1}) = a_i^{l-1} \delta_j^l \\ \Rightarrow \frac{\partial C}{\partial W^l} &= \delta^l (a^{l-1})^T \end{aligned} \tag{2}$$

Saturated phenomenon

Saturated means the learning process becomes slowly because the gradient is close to 0. Since we already know how to calculate the gradient, there are two situations where the gradient for W^l will close to 0: δ^l close to 0 or a^{l-1} close to 0.

- Input activation is low activation
- $\sigma'(z^l)$ is near saturated, e.g if $\sigma(\cdot)$ is sigmoid function, it will near saturated when $z^l \to \infty$ or $z^l \to -\infty$.

Psudo code for back propagation

The above process for calculating gradient is based on a single example. In practice, it is common to combine backpropagation with stochastic gradient descent, where we compute gradient for a mini-batch of data. Assume we have *m* training examples:

- Input training data
- For each training data x, perform the following steps:
- Forward step: for $l = 1, 2, \cdots, L$ compute
- $z^{x,l} = W^l a^{x,l-1} + b^l$ where $a^{x,l-1} = \sigma(z)^{x,l-1}$
- Output error: compute
- $\delta^{x,L} = \nabla_a C_x \odot \sigma'(z^{x,L})$
- Backpropagate the error: for $l = L-1, L-2, \cdots, 1$ compute
- $\delta^{x,l} = ((W^{l+1})^T \delta^{x,l+1}) \odot \sigma'(z^{x,l})$
- Gradient descent: for $l = L, L-1, \cdots, 2$ update
- $W_t^l = W_{t-1}^l - \frac{\alpha}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$
- $b_t^l = b_{t-1}^l - \frac{\beta}{m} \sum_x \delta^{x,l}$

Some tips

- Check gradient by

$$\frac{\partial C}{\partial w_j} \approx \frac{C(w_j + \epsilon e_j) - C(w_j)}{\epsilon}$$

- Random initializing parameters

Handwritten Digit Recognition

Reference

- http://natureofcode.com/book/chapter-10-neural-networks/
- http://neuralnetworksanddeeplearning.com/chap2.html
- https://www.coursera.org/learn/machine-learning

Thank you!