Imperial College London

60006 Computer Vision (Term 2)

Tutorial 4: Image Classification

1. In neural networks, the activation function defines the output of a neuron given an input.

   1.1 The sigmoid function is a type of activation function. It is defined as,

$$f(x) = \frac{1}{1+e^{-x}}$$

   Calculate the derivative of $f(x)$ with respect to x.

Solution:

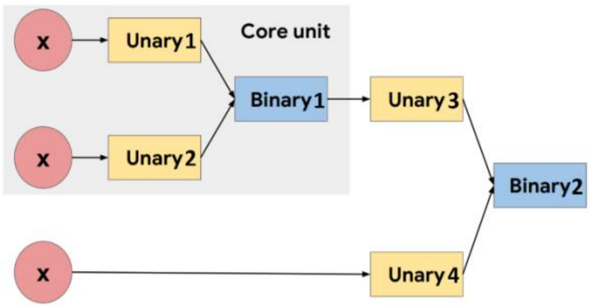
$$\begin{aligned} f'(x) &= \frac{e^{-x}}{(1+e^{-x})^2} \\ &= \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} \\ &= f(x)(1-f(x)) \end{aligned}$$

   1.2 Describe a problem with the sigmoid function when we train a neural network using the gradient descent algorithm.

Solution:

The sigmoid function suffers from the vanishing gradient problem. When $f(x)$ saturates at either 0 or 1, its derivative $f'(x)$ becomes nearly 0.

2. A method was proposed to automatically search and discover new activation functions (P. Ramachandran et al, ICLR 2018 workshop). It assumes that the activation function has a structure shown in this figure,



which consists of input x, unary functions and binary functions. The unary function takes a single scalar input and returns a single scalar output, such as $u(x) = x$. The binary function takes two scalar inputs and returns a single scalar output, such as $b(x_1, x_2) = x_1 \cdot x_2$.

   2.1 Suppose in the figure, the unary functions are respectively $u_1(x) = x$, $u_2(x) = x$, $u_3(x) = \sigma(x)$, $u_4(x) = \sigma(x)$, where $\sigma(x)$ denotes the sigmoid function and u_i denotes "Unary i" in the figure. The binary functions are $b_1(x_1, x_2) = \max(x_1, x_2)$ and $b_2(x_1, x_2) = \max(x_1, x_2)$, where b_i denotes "Binary i" in the figure.

   Write down the activation function.

Solution:

$$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$$

2.2 In the search space of activation functions, there are in total M possible unary functions (e.g. x, $-x$, $|x|$, x^2, x^3, $\sqrt{x}$, βx, …) and N possible binary functions (e.g. x_1+x_2, $x_1 \cdot x_2$, x_1-x_2, $\frac{x_1}{x_2+\epsilon}$, $\max(x_1, x_2)$, $\min(x_1, x_2)$, …). Calculate the number of possible combinations using the big O notation.

Solution:

The number of possible combinations is in the scale of $O(M^4 N^2)$.

2.3 The method utilises reinforcement learning to search for activation functions. Finally, it finds a novel activation function, $f(x) = x \cdot \sigma(\beta x)$, which performs well. What does this new activation function look like if $\beta = 0$ and if $\beta \to \infty$?

Solution:

If $\beta = 0$, $f(x) = \frac{x}{2}$. It becomes a linear function.

If $\beta \to \infty$, $f(x) = \begin{cases} 0, x < 0 \\ x, x \geq 0 \end{cases}$. It becomes ReLU.

3. A data analyst has developed a neural network model to predict the chances of the three scenarios (win, draw, lose). However, the current model outputs a vector of three integers, $c = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}$, instead of a probability vector. The three integers are not necessarily positive. But the larger the integer, the higher the chance is for that scenario.

3.1 He/she decides to apply the softmax function to convert the vector $c = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}$ into a probability vector $p = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix}$. The softmax function is defined as,

$$p_i = \frac{e^{c_i}}{\sum_k e^{c_k}} \text{ for } i = 1,2,3.$$

Check whether p fulfils the properties of a probability vector, i.e. it is non-negative and its elements sum to 1.

Solution:

For real number c_i, its exponential e^{c_i} is non-negative and the sum $\sum_k e^{c_k}$ is non-negative as well. Thus the elements of p are non-negative.

The sum of the elements is $\sum_i p_i = \sum_i \frac{e^{c_i}}{\sum_k e^{c_k}} = 1$.

3.2 Gradient descent is used to train this model. To calculate the gradient of the loss function, one step is to work out the derivative $\frac{\partial p_i}{\partial c_j}$. Please help him/her derive this. (Hint: consider two scenarios, $i = j$ and $i \neq j$.)

Solution:

If $i = j$,

$$\frac{\partial p_i}{\partial c_j} = \frac{e^{c_i} \sum_k e^{c_k} - e^{c_i} e^{c_j}}{(\sum_k e^{c_k})^2}$$

$$= \frac{e^{c_i}}{\sum_k e^{c_k}} \cdot \frac{\sum_k e^{c_k} - e^{c_j}}{\sum_k e^{c_k}}$$
$$= p_i \cdot (1 - p_j)$$

If $i \neq j$,

$$\frac{\partial p_i}{\partial c_j} = \frac{-e^{c_i} e^{c_j}}{(\sum_k e^{c_k})^2}$$
$$= \frac{e^{c_i}}{\sum_k e^{c_k}} \cdot \frac{-e^{c_j}}{\sum_k e^{c_k}}$$
$$= p_i \cdot (-p_j)$$

Using the Kronecker delta $\delta_{ij} = \begin{cases} 1, \text{if } i = j \\ 0, \text{if } i \neq j \end{cases}$, we have

$$\frac{\partial p_i}{\partial c_j} = p_i \cdot (\delta_{ij} - p_j)$$

4. A convolutional neural network (CNN) takes a 28x28 image as input and produces an output of 10-dimensional probability vector and cross-entropy loss. It mainly consists of convolutional layers, max pooling layers and a loss layer. The network architecture is specified in the following table.

layer	0	1	2	3	4	5	6	7
type	input	conv	pool	conv	pool	conv	conv	loss
filter shape	-	5x5x1	2x2	5x5x20	2x2	4x4x50	1x1x500	-
#filters	-	20	-	50	-	500	10	-
stride	-	1	2	1	2	1	1	-
pad	-	0	0	0	0	0	0	-
data shape	1x28x28x1	1x24x24x20	1x12x12x20	1x8x8x50	1x4x4x50	1x1x1x500	1x1x1x10	1
data size	3.06KB	45KB	11.25KB	12.5KB	3.13KB	1.95KB	0.04KB	0.004KB
receptive field	1x1	5x5	6x6	14x14	16x16	28x28	28x28	28x28

4.1 The input data x_0 is of shape 1x28x28x1, which represents BWHC (B: batch size; W: width; H: height; C: channel). If we use single precision floating-point data (4 bytes), the data size is 1x28x28x1x4 $\approx$ 3KB. Calculate the data shape and size for each following layer in the table. Data means input image at Layer 0 and feature map at subsequent layers.

Solution:

Please refer to the table for answers.

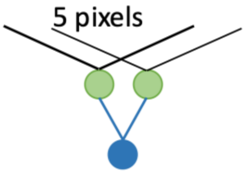
4.2 The receptive field of a neuron represents the size of the region in input image that can affect this neuron. For example, Layer 1 uses a 5x5 convolution filter. Therefore, a neuron at Layer 1 has a receptive field of 5x5, since each neuron is affected by a 5x5 region in the input image. Calculate the receptive fields for neurons in the following layers and fill in the table.

Solution:

Please refer to the table for answers.

To calculate the receptive field of a neuron at Layer i+1, we account for the receptive field at Layer i and the kernel size at Layer i+1.

For example, for a neuron at Layer 2,

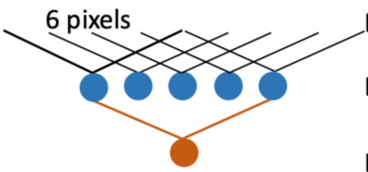


Input layer

Layer 1: Each neuron sees 5 pixels.

Layer 2: Each neuron sees 2 neurons at Layer 1, which are 1 pixel apart. Therefore, it see 5 + 1 = 6 pixels in the input image.

For a neuron at Layer 3,



Input layer

Layer 1

Layer 2: Each neuron sees 6 pixels.

Layer 3: Each neuron sees 5 neurons at Layer 2. The distance between the first neuron and the fifth neuron is 8 pixels (due to the stride of 2). Therefore, it sees 6 + 8 = 14 pixels in the input image.

We can do this similarly for Layer 4, Layer 5 etc, incrementing the receptive field layer by layer.