Neural Networks

Homework 5

Juan Jose Mendez Torrero (s3542416)
Sharif Hamed (s2562677)

June 4, 2018

1 Theory questions

1. The two purposes of pooling are reduce the number of required neurons and that it controls the overfitting. The first purpose cause a decrease in the number of parameters and computations. The second one, use the mean of the neurons to not overfit the training set.

2. When we talk about weight sharing, we have to know that in each layer, all the neurons have the same weight value. This able to the network to perform the convolution, which makes the network faster reducing the computation time and the number of parameters. The only disadvantage that we have discovered is that this way, makes the network less flexible. Nevertheless, it allows to the network to avoid overfitting.

3. The reason that less time is needed is that about 50% of the nodes are activated due to the activation function: $f(x) = max(0, x)$

4. (a) If we apply the convolutional layer to the input image, we will have as a result, one matrix of size $4x4$, that means that the filter *fits* in the image 16 times. Therefore, if we have used 3 filters that means $3 * 16 = 48$ neurons in the layer.

   (b) We know that one neuron will have 12 connections. Besides, we have 12 neuron inside the hidden layer, doing this an amount of 12*12 connections. Finally, if we want to connect this neurons with the 48 neurons of the convolutional layer, we have as a result $12 * 12 * 48 = 6912$ connections.
   If we compare this number of parameters with the convolutional layer, we can observe that it is a lot bigger that in section (a). Hence, this means that the computation time should be a lot smaller.

5. This approach is questionable due to the relationship between all the parameters that belong to the input image. It only would make sense if we would normalized all the parameters in order to have all of them on the same scale.

2 Designing a classifier

1. (a) A CNN provides a computational advantage over fully interconnected layer because, CNN layers create a volume of size (weight, height and depth) where it keeps all the neuron already classified. This process is not made by a fully connected layer, giving a computational advantage to a CNN layer.

   (b) Aside what we said above, the CNN assumes that the input is an image, whereas a fully connected layer does not assumethat. Furthermore, a CNN layer shares parameters with other layers, in order to not waste more memory.

(c) The features that the first layers would detect are white spaces of the image.

(d) The network will classify this image as there is no hot dog in that picture. This is because all the inputs that the network have had, they have been all horizontal hot dog's pictures, therefore, the network will not be able to learn it at the first time.

2. (a) It will be sufficient because the output should be, *Yes, I jump.* or *No,I do not jump.*

(b) In terms of complexity, we rather use a grayscale pixel over a RBG pixel because we have to distinguish between if there is an object to jump in fort of the T-Rex or not. If we would use an RBG pixel value, it would be more complex to the network, to reach the correct output.

(c) It would cause a worst performance because with a grayscale pixel values we will not be able to classify correctly each one of the colors that appear on the picture. With a RBG pixel values the network will be able to classify, in a better way, all the pixels.

3. (a) In order to calculate the output of the Softmax layer, we have to consider the following formula:

$$y = \frac{e^{w_i \cdot x + b_i}}{\Sigma_j e^{w_j \cdot x + b_j}}$$

As we can observe, we have to calculate all the activation through the classes, hence, we are going to calculate the denominator:
$act = \Sigma_j w_j \cdot x + b_j = 0.7*(0.1-0.2+0.4+0.1)+0.9*(0.3+0.4+0+0.2)+0.1*(0.6+0.1-0.1-0.5)+0.9*(-0.4+0.3-0.7-0.4)+0.1*(-0.1+0.5+0.2+0.3)+0.8*(0.2-0.2+0.3-0.5) = -0.05$
$\Sigma_j e^{w_j \cdot x + b_j} = e^{act} = 0.95$
Now we have calculated the value of the denominator, it is going to be easier to obtain the output's value.
$y_1 = \frac{e^{w_1 \cdot x + b_1}}{\Sigma_j e^{w_j \cdot x + b_j}} = \frac{e^{0.1*0.7+0.3*0.9+0.6*0.1-0.4*0.9-0.1*0.1+0.8*0.2}}{0.95} = 1.27$
Doing the same but changing the class, we can obtain the value of each output node in an easy way.
$y_2 = 1.55$
$y_3 = 0.95$
$y_2 = 0.619$
In conclusion, we have obtain the output vector and its value is $y_{out} = [1.27, 1.55, 0.95, 0.619]$.

(b) The action that our bot is going to pick in this current frame is to going DOWN. This is due to the value of the output, and assuming the order and relationship, we can see that the action of going down is which has the greatest value.

3 An edge detector on paper

1. In order to apply 2x2x2 convolutional layer, we have, first, to flip the kernels (or filters). Therefore, we will have two filters as:

W_0 : 1 -1 / 1 -1 and W_1 : 1 1 / -1 -1

Furthermore, we have to apply the zero-padding to the input in order to calculate the convultional layer. Bellow we can observe the input after the zero-padding has been applied.

In :

0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	1	2	2	0	0	0
0	1	2	2	0	0	0
0	1	1	2	0	0	0
0	1	1	1	0	0	0
0	0	0	0	0	0	0

After this, we can now obtain the output of the 2x2x2 convolutional layer. To do so, we have to apply the convolution process to the zero-padded input with the already flipped kernels. The result is as follows:

$$In_0: \begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 2 & 2 & 0 & 0 & 0 \\ 0 & 1 & 2 & 2 & 0 & 0 & 0 \\ 0 & 1 & 1 & 2 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix} \cdot \begin{matrix} 1 & -1 \\ 1 & -1 \end{matrix} = \begin{matrix} -1 & -2 & -2 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 1 & 1 & 4 & 0 & 0 \\ 0 & 0 & 1 & 3 & 1 & 0 \\ 1 & 2 & 2 & 1 & 1 & 0 \end{matrix}$$

$$In_1: \begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 2 & 2 & 0 & 0 & 0 \\ 0 & 1 & 2 & 2 & 0 & 0 & 0 \\ 0 & 1 & 1 & 2 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix} \cdot \begin{matrix} 1 & 1 \\ -1 & -1 \end{matrix} = \begin{matrix} -1 & 0 & 0 & 1 & 0 & 0 \\ -2 & -1 & 0 & 3 & 0 & 0 \\ -2 & -2 & 0 & 4 & 0 & 0 \\ -2 & -1 & -1 & 4 & 0 & 0 \\ -2 & 0 & -1 & 3 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 \end{matrix}$$

As we can observe, we have obtain a matrix of size 6x6x2. These are the output of the convolutional layer.

2. Now, we want to apply 2x2 max pooling. In order to do that, we have to take the outputs of the convolutional layer and apply it to them. The process consist on taking of size 2x2 and see which is the large number of them. Doing so, we have obtain:

$$In_0: \begin{matrix} 0 & 3 & 0 \\ -1 & 4 & 0 \\ 0 & 3 & 0 \end{matrix} \quad \text{and} \quad In_1: \begin{matrix} 0 & -1 & 0 \\ 1 & 1 & 0 \\ 2 & 2 & 0 \end{matrix}$$

3. In order to obtain the activation of both neurons, we have to know that $a_i = w_i \cdot x_i$. Hence, we have obtain the activation of each neuron as follows:
$a_0 = w_0 * 1 + w_1 * -1 = 0 * 1 + -1 * 1 + 0 * 1 + 1 * 1 + 1 * 1 + 0 * 1 + 2 * 1 + 2 * 1 + 0 * 1 + 0 * -1 + 3 * -1 + 0 * -1 + -1 * -1 + 4 * -1 + 0 * -1 + 0 * -1 + 3 * -1 + 0 * -1 = -4$
Likewise, we calculate the activation of the other neuron:
$a_1 = w_0 * -1 + w_1 * 1 = 0 * -1 + -1 * -1 + 0 * -1 + 1 * -1 + 1 * -1 + 0 * -1 + 2 * -1 + 2 * -1 + 0 * -1 = 4$
Besides, we have to obtain the ReLu function as $max = 0, x$. Therefore, $a_0 = max0, -4 = 0$ and $a_1 = max0, 4 = 4$

4. Before apply the Softmax function, we have to know that the output of the Softmax function is: $y = \frac{exp(a_i)}{\Sigma_j exp(a_j)}$, hence, applying this formula we obtain the following result:

$y_0 = \frac{exp(0)}{exp(0)+exp(4)} = 0.018$
$y_1 = \frac{exp(4)}{exp(0)+exp(4)} = 0.982$

This means that the portion of vertical edges is much less than the horizontal edges, therefore, the network will classify In as a horizontal edge.

4 Essay question

1. The cross-entropy loss function can influence the neural network because it represents the way the network learns. We have to know that the cross-entropy loss function gives us the

error that the network is making each epoch, hence, the function will lead the network trough its learning.
Sometimes is better to use the cross-entropy loss rather that MSE because the first get rid of the slow learning part. This is because it does not use the derivative of the cost function. This error function would work in a better way for the T-Rex CNN than for the Pac-Man CNN. This is due to the number of inputs.