Deep Learning

2021/2022

Homework No. 2

1. N.º 90007 Nome: Alice Rosa
2. N.º 90029 Nome: Beatriz Pereira
3. N.º 89916 Nome: José Marques

Group No. 55

Professors: André Martins, Rita Ramos, João Santinha, José Moreira, Ben Peters, Francisco Melo

1 Question 1

1.1

In this question, we will compute the total number of parameters a convolutional neural network (CNN) that takes input images of size 28x28x3 and has the following layers:

- One convolutional layer with 8 kernels of shape 5x5x3 with stride of 1, and a padding of zero (*i.e.*, no padding).
- A max-pooling layer with kernel size 4x4 and stride of 2 (both horizontally and vertically).
- A linear output layer with 10 outputs followed by a softmax transformation.

First, to compute the number of trainable parameters/weights take:

$$N = \text{Number of filters} \times [(\text{kernel width} \times \text{kernel height} \times \text{number of channels}) + \text{bias}]. \tag{1}$$

Now, substitute the values to obtain the number of trainable parameters/weights within the convolutional layer ($N_{conv.}$) to obtain

$$N_{conv.} = 8 \times [(5 \times 5 \times 3) + 1] = 608.$$

The width of the output (O) of the convolution layer can be obtained using the formula described bellow

$$O = \frac{\text{input width} - \text{kernel width} + 2 \times \text{padding width}}{\text{stride}} + 1. \tag{2}$$

Hence, substituting the network values in (2), we obtain:

$$O = \frac{28 - 5 + 2 \times 0}{1} + 1 = 24,$$

which means the output has size $24 \times 24 \times 8$.

The max-pooling layer makes the representations smaller and more manageable, so the size of the image will be reduced. The output width of the pooling layer (O_p) can be computed by

$$O_p = \frac{\text{Input Size} - \text{Pool Size}}{\text{Stride}} + 1. \tag{3}$$

Thus, we obtain

$$O = \frac{24 - 4}{2} + 1 = 11,$$

which means the output has now size $11 \times 11 \times 8$.

The number of trainable parameters/weights for a linear output layer, can be obtained by the equation bellow

$$N = \text{Number of units} \times [(\text{input width} \times \text{input height} \times \text{number of channels}) + \text{bias}]. \tag{4}$$

Then, by substituting the values of the linear output layer in (4) it is possible to obtain the number of parameters (N_{linear})

$$N_{linear} = 10 \times [(11 \times 11 \times 8) + 1] = 9690.$$

Finally, the total number of trainable parameters the CNN are

$$N_{Total} = 608 + 9690 = 10298.$$

1.2

In this question, we substitute the the convolutional and max-pooling layers of the previous network, by a feedforward, fully connected layer with hidden size 100 and output size 10, that is followed by the softmax transformation. Then, we will compute the total number of parameters in this case and compare the results with the ones obtained in the previous question.

First, the number of trainable parameters within the feedforward, fully connected layer (N_{FCL}) with hidden size 100 can be obtained from (4), as

$$N_{FCL} = 100 \times [(28 \times 28 \times 3) + 1] = 235300.$$

Then, the number of parameters of the output can be computed by

$$N_{out} = 10 \times (100 + 1) = 1010,$$

given the hidden size 100 is now the input size.

Lastly, the total number of trainable parameters are

$$N_{Total} = 235300 + 1010 = 236310.$$

When compared to the previous answer, it is possible to conclude that by substituing the convolutional and max-pooling layers of the previous question, by a feedforward, fully connected layer the total number of trainable parameters increase. This consists of the main problem of using a fully connected layer, with a larger set of parameters/weights, one is more susceptible to problems, such as slower training time and higher chances of overfitting. Therefore, particularly for computer vision, it is better to have a convolutional and max-pooling layer, since they allow to have fewer parameters/weights to train.

1.3

In this question, consider $\mathbf{X} \in \mathbb{R}^{L \times n}$ to be an input matrix for a sequence of length L, with embedding size n. Take two self-attention heads that have projection matrices $\mathbf{W}_Q^{(h)}$, $\mathbf{W}_K^{(h)}, \mathbf{W}_V^{(h)} \in \mathbb{R}^{n \times d}$, for $h \in \{1, 2\}$ with $d \leq n$.

1.3.1

Now, we will prove that the self-attention probabilities for each head can be written as the rows of a matrix as

$$\mathbf{P}^{(h)} = \text{Softmax}(\mathbf{X}\mathbf{A}^{(h)}\mathbf{X}^T), \tag{5}$$

where $\mathbf{A}^{(h)} \in \mathbb{R}^{n \times n}$ has rank $\leq d$. The softmax is applied row-wise.

By projecting the embedding matrix $\mathbf{X} \in \mathbb{R}^{L \times n}$ to a lower dimension, we get

$$\mathbf{Q}^{(h)} = \mathbf{X}\mathbf{W}_Q^{(h)},$$
$$\mathbf{K}^{(h)} = \mathbf{X}\mathbf{W}_K^{(h)},$$
$$\mathbf{V}^{(h)} = \mathbf{X}\mathbf{W}_V^{(h)},$$

where $\mathbf{Q}^{(h)}$ represent the query vectors, $\mathbf{K}^{(h)}$ the key vectors and $\mathbf{V}^{(h)}$ the value vectors.

The self-attention probabilities for each head can be depicted as

$$\mathbf{P}^{(h)} = \text{Softmax}(\mathbf{Q}^{(h)}\mathbf{K}^{(h)^T}). \tag{6}$$

Therefore, by substitution in (6) we get

$$\mathbf{P}^{(h)} = \text{Softmax}[\mathbf{X}\mathbf{W}_Q^{(h)}(\mathbf{X}\mathbf{W}_K^{(h)})^T] = \text{Softmax}(\mathbf{X}\mathbf{W}_Q^{(h)}\mathbf{W}_K^{(h)^T}\mathbf{X}^T).$$

Now, is possible to define the matrix $\mathbf{A}^{(h)} \in \mathbb{R}^{n\times n}$ as $\mathbf{A}^{(h)} = \mathbf{W}_Q^{(h)}\mathbf{W}_K^{(h)^T}$, given that $\mathbf{W}_Q^{(h)} \in \mathbb{R}^{n\times d}$ and $\mathbf{W}_K^{(h)^T} \in \mathbb{R}^{d\times n}$.

Then, because $\mathbf{A}^{(h)}$ is the product of two matrices, its rank is the minimum rank of the two. It is known that $\mathbf{W}_Q^{(h)}$ has rank $\leq n$ and $\mathbf{W}_K^{(h)^T}$ has rank $\leq d$. Since $d \leq n$, matrix $\mathbf{A}^{(h)}$ has rank $\leq d$, as expected.

Hence, possible to prove the equation (5).

1.3.2

Let us consider that

$$\mathbf{W}_Q^{(2)} = \mathbf{W}_Q^{(1)}\mathbf{B}, \quad \mathbf{W}_K^{(2)} = \mathbf{W}_K^{(1)}\mathbf{B}^{-T},$$

in which $\mathbf{B} \in \mathbb{R}^{d\times d}$ represents any invertible matrix. Now, we will prove that the self-attention probabilities are the same for the two attention heads.

For that, from the result of (6) developed in question 1.3.1, we get

$$\mathbf{P}^{(1)} = \text{Softmax}(\mathbf{X}\mathbf{W}_Q^{(1)}\mathbf{W}_K^{(1)^T}\mathbf{X}^T), \tag{7}$$

and

$$\mathbf{P}^{(2)} = \text{Softmax}(\mathbf{X}\mathbf{W}_Q^{(2)}\mathbf{W}_K^{(2)^T}\mathbf{X}^T). \tag{8}$$

Now, by working on (8) it is possible to obtain

$$\mathbf{P}^{(2)} = \text{Softmax}[\mathbf{X}\mathbf{W}_Q^{(1)}\mathbf{B}(\mathbf{W}_K^{(1)}\mathbf{B}^{-T})^T\mathbf{X}^T] = \text{Softmax}[\mathbf{X}\mathbf{W}_Q^{(1)}\mathbf{B}\mathbf{B}^{-1}\mathbf{W}_K^{(1)^T}\mathbf{X}^T].$$

Given that $\mathbf{B}\mathbf{B}^{-1} = \mathrm{I}$, where I is the Identity matrix, it is possible to conclude that the equation (7) is equal to the equation (8). Therefore, the self-attention probabilities are the same for the two attention heads.

2 Question 2 - Image Classification with CNNs

2.1 Equivariance in Convolutional layers

An important property of Convolutional Neural Networks is translational equivariance, *i.e.*, the position of the object in the image does not need to be fixed in order to be detected by the CNN. In processing images, this simply means that if we shift the input a given number of pixels to the right, the output representations will also shift the same amount of pixels to the right. An example is represented in figure 1.

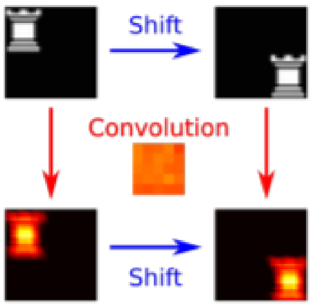


Figure 1: Equivariance in Convolutional layers. From slide 16 of lecture 8 of Deep Learning Course.

This property is very useful for applications such as image classification, object detection, etc., where the object may occur multiple times or be in motion. Therefore, we can perform a shorter training process.

2.2 Convolutional Network

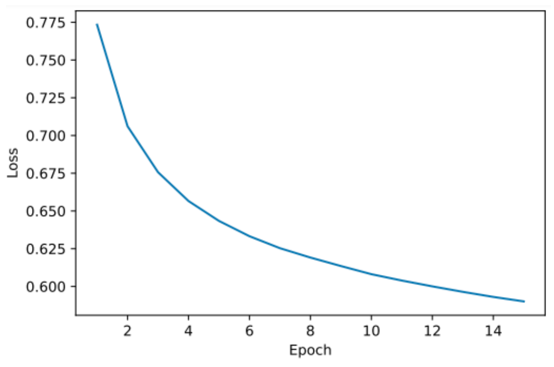
In this question, we implement a Convolutional Neural Network to classify images from the FashionMNIST dataset. Instructions were given in regards to the Network structure leaving only the dropout probability and different learning rate values. We decided to use 0.5 for the dropout probability. After testing, we obtained the results shown in table 1, for the different learning rate values.

Table 1: Results obtained after training the model for 15 epochs with different learning rates.

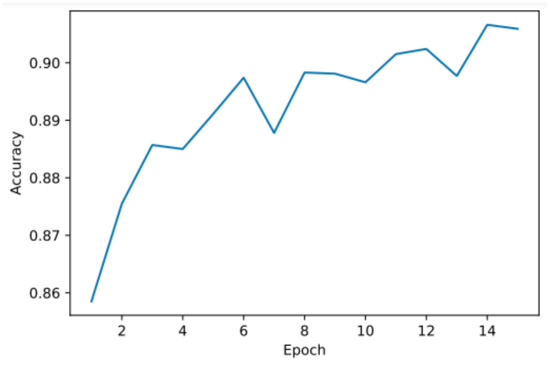
Learning Rate	0.1	0.01	0.001
Final Test Accuracy	0.8991	0.8917	0.8093

According to the results, the best setting for this problem is achieved with a learning rate of 0.1.

The following results obtained for the training loss and validation accuracy as a function of the number of epochs are illustrated in figures 2 (a) and 2 (b).



(a) Training Loss



(b) Validation Accuracy

Figure 2: Resulting plots for the CNN with 0.1 Learning Rate

2.3 Convolutional Network - Filters

For this question, we obtained the filters illustrated in figures 3 and 4, used in the first and second convolutional layers, respectively.

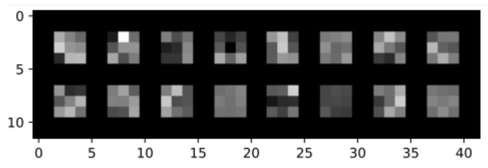


Figure 3: Filters used in the first layer.

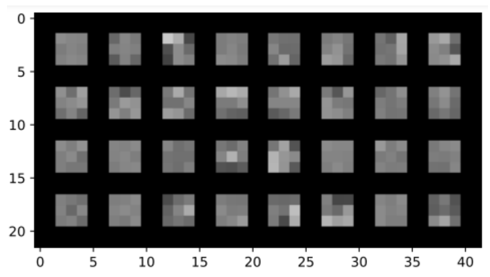


Figure 4: Filters used in the second layer.

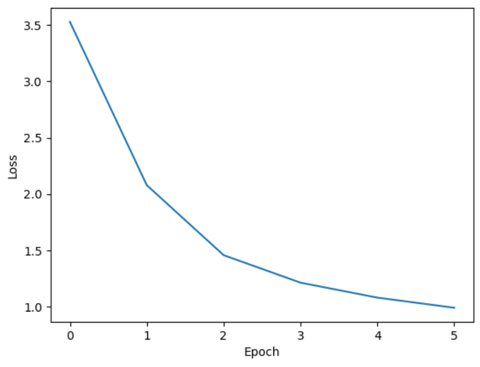
The filters used in the bottom layers of this CNN are not high definition, so it is impossible to make any assumption based on the figures alone. However, in theory, lower level filters usually focus on more basic features, and therefore tend to be low resolution, while the higher layer filters tend to look for more complex features. Some examples of low level features are corners and edges, and more complex features can be object characteristics such as car wheels, eyes, among others.

3 Question 3 - Image Captioning

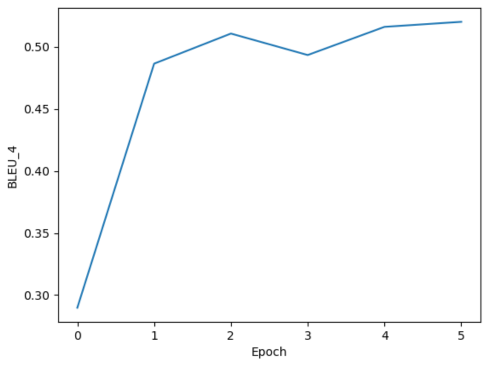
In this question, we are going to implement an image captioning model concerning aerial images.

3.1 LSTM Decoder

In this question, we implemented a vanilla image captioning model using an encoder-decoder architecture with an auto-regressive LSTM as the decoder. Figures 5 (a) and 5 (b) show the training loss and validation BLEU-4 as a function of the epoch number. The final BLEU-4 score obtained in the test set is: 0.4988.



(a) Training Loss

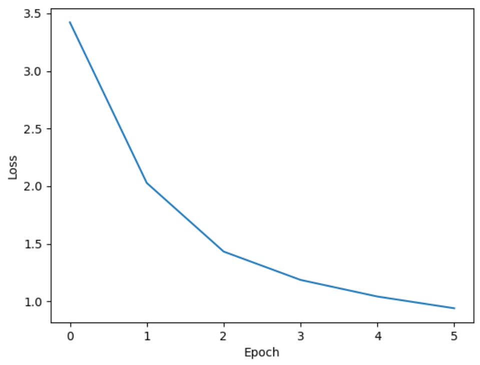


(b) Validation BLEU-4

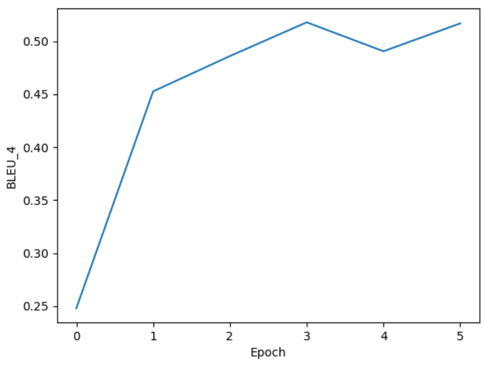
Figure 5: Resulting plots with the LSTM decoder

3.2 LSTM Decoder with an attention mechanism

In this question, we added an attention mechanism to the LSTM decoder of the previous section. Figures 6 (a) and 6 (b) show the training loss and validation BLEU-4 as a function of the epoch number. The final BLEU-4 score obtained in the test set is: 0.5284.



(a) Training Loss



(b) Validation BLEU-4

Figure 6: Resulting plots with the LSTM decoder with attention

3.3 Generated Captions

For the previous model from the section 3.2, the generated captions obtained for the images "219.tif", "540.tif", and "357.tif" are shown along with their corresponding images in figures 7, 8, and 9, respectively.

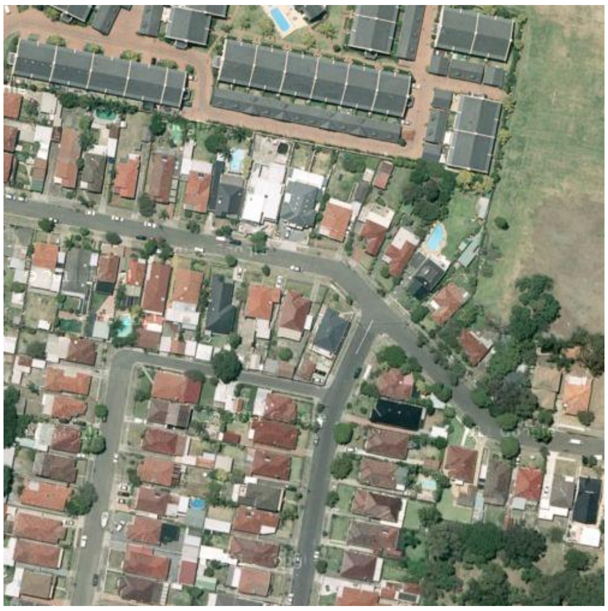


Figure 7: Generated caption for "219.tif": "a residential area with houses arranged neatly and some roads go through this area".

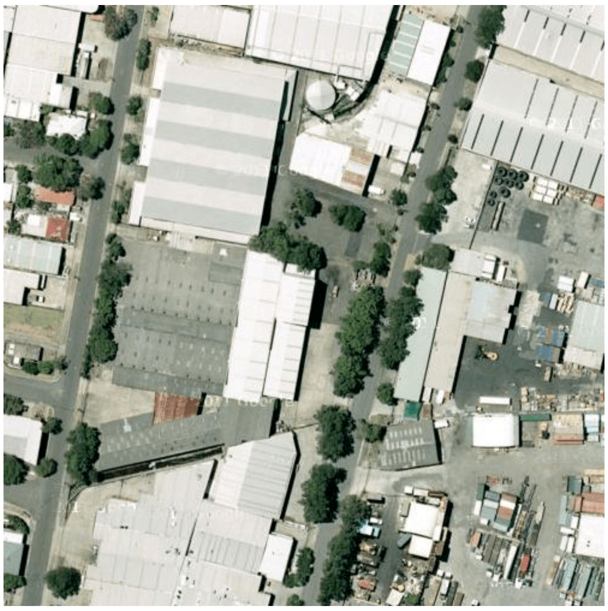


Figure 8: Generated caption for “540.tif”: “an industrial area with many white buildings and some roads go through this area”.

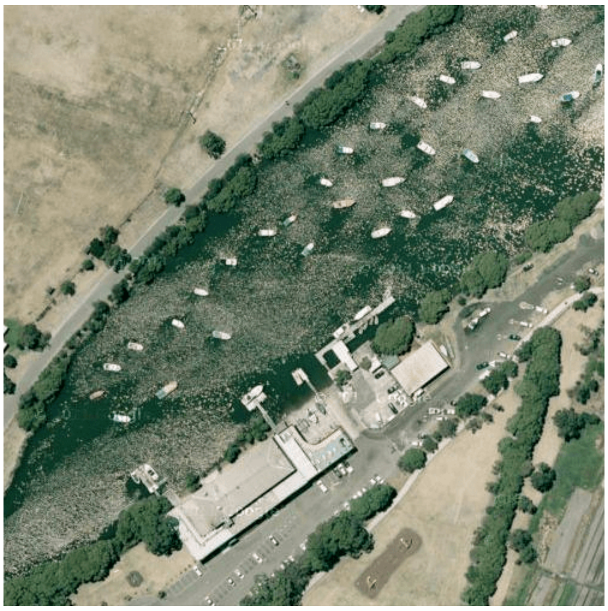


Figure 9: Generated caption for “357.tif”: “a curved river with some green waters and a highway passed by”.