Deep Learning

MEEC21/MECD21

Homework Assignment #2

Authors:

Afonso Araújo (96138) afonso.d.araujo@tecnico.ulisboa.pt
Emma Dennis-Knieriem (105545) emma.dennis-knieriem@tecnico.ulisboa.pt

Group 18

2022/2023 – 1st Semester, P2

Contents

Individual Contribution: As for this Assignment, Afonso worked out the coding, its plots, the mathematical equations and their corresponding answers, 3.1c), whilst Emma worked on the remaining theoretical questions.

1 Question 1

1.1 1a)

As we know :

$$\begin{aligned} Output\_width &= \frac{(input\_width - kernel\_width + 2 \times padding\_width)}{stride} + 1 \\ Output\_height &= \frac{(input\_height - kernel\_height + 2 \times padding\_height)}{stride} + 1 \end{aligned} \tag{1}$$

Therefore, since,

$$\begin{aligned} dim(x) &= H \times W \\ dim(W) &= M \times N \end{aligned} \tag{2}$$

and,

$$\mathbf{z} = conv(\boldsymbol{W}, \boldsymbol{x}) \tag{3}$$

then,

$$\begin{aligned} z\_width &= \frac{W - N + 2 \times 0}{1} + 1 \iff z\_width = W - N + 1 \\ z\_height &= \frac{H - M + 2 \times 0}{1} + 1 \iff z\_height = H - M + 1 \end{aligned} \tag{4}$$

or,

$$dim(z) = (H - M + 1) \times (W - N + 1) \tag{5}$$

1.2 1b)

We have,

$$\boldsymbol{z'} = \mathbf{M}\boldsymbol{x'} \tag{6}$$

where,

$$\begin{aligned} \boldsymbol{x'} &= vec(\boldsymbol{x}) \\ \boldsymbol{z'} &= vec(\boldsymbol{z}) \end{aligned} \tag{7}$$

If we take a nomal convolution, where $dim(K) = A \times B$ as stated in 3, then,

$$z_{u,v} = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} K_{i,j} x_{u+i,v+j} \tag{8}$$

We should have an $\mathbf{M}$ matrix, with dimensions $H'W' \times HW$, that are the dimensions for the vectorized versions of $\mathbf{z}$ and $\mathbf{x}$ respectively. Such matrix is made up of $W_{a,b}$ weights in a way that the first *H'* lines are:

$$\begin{bmatrix} W_{1,1} & \cdots & W_{M,1} & 0 & \cdots & 0 & W_{1,2} & \cdots & W_{M,2} & \cdots & 0 \\ 0 & W_{1,1} & \cdots & W_{M,1} & \cdots & 0 & \cdots & 0 & W_{1,2} & \cdots & W_{M,2} \\ \vdots & \cdots & \ddots & \cdots & \ddots & \cdots & \ddots & \ddots & \cdots & \ddots & \cdots \\ 0 & \cdots & 0 & W_{1,1} & \cdots & W_{M,1} & 0 & \cdots & 0 & W_{1,2} & \cdots \end{bmatrix} \tag{9}$$

and starting from line $H' + 1$ we have,

$$\begin{bmatrix} 0 & \cdots & 0 & W_{1,1} & \cdots & W_{M,1} & 0 & \cdots & 0 & W_{1,2} & \cdots \\ 0 & \cdots & 0 & 0 & W_{1,1} & \cdots & W_{M,1} & 0 & \cdots & 0 & W_{1,2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{10}$$

in a repeating pattern.

Now as for the general expression of matrix $\mathbf{M}$,

$$M_{i,j} = \begin{cases} W_{a,b}, & 0 < a = j - (b-1)H - (i-1) - (\lceil \frac{i}{H'} \rceil - 1)(A-1) \leq A, \quad 0 < b = \lceil \frac{j}{H} \rceil - \lceil \frac{i}{H'} \rceil + 1 \leq B \\ 0, & elsewhere \end{cases} \tag{11}$$

In this way we can see that the parameters that create both "a" and "b" are responsible for the shift of weights throughout the matrix $\mathbf{M}$ either in terms of lines or columns, within the first *H'* lines or starting from $H' + 1$ line.

The matrix is repetitive in itself in the way that the only thing that changes, i.e, from the first to second line is but the addtion of a "0" to the beginning of the second line. This pattern ocurrs for a while up until the point, line $H' + 1$, where the matrix "starts again" but with the addition of H zeros to the beginning of this new line.

In this way instead of "$W_{1,1}$" starting at column 1 it will start on column $H + 1$ and allow the pattern of the first H lines to repeat itself up until we have filled the whole $H'W' \times HW$ matrix.

1.3 1c)

Let´s take $\mathbf{z}$ as the output from the convolutional layer, $\mathbf{h}$ as the output from the max pooling layer. Therefore,

$$\begin{aligned} dim(\mathbf{z}) &= (H - M + 1) \times (W - N + 1) \\ dim(\mathbf{h}) &= \lfloor \frac{H - M + 1}{2} \times \frac{W - N + 1}{2} \rfloor \end{aligned} \tag{12}$$

From here, since we have 3 possible classes and one $M \times N$ filter, the total amount of parameters for this network would be,

$$\#Parameters_{\mathbf{CNN}} = 3 \cdot dim(\mathbf{h}) + MN \tag{13}$$

For our Fully-Formed network we know that since our image has $dim(\mathbf{x}) = H \times W$ and we still have 3 classes,

$$\#Parameters_{FF} = dim(\mathbf{h}) \cdot H \cdot W + 3 \cdot dim(\mathbf{h}) \tag{14}$$

It is easy to see then how the number of parameters in a Fully-Formed network, outweighs the number in a convolutional network with single convolutional and max pooling layers.

1.4 2

We take,

$$\mathbf{X} = vec(\mathbf{x}) = [x_{1,1}, ..., x_{m,1}, x_{1,2}, ..., x_{m,2}, ..., x_{1,n}, ..., x_{m,n}]^T \tag{15}$$

Therefore we have,

$$\mathbf{Q} = \mathbf{X}\mathbf{W_Q} = vec(\mathbf{x}) \cdot 1 = vec(\mathbf{x}) \tag{16}$$

$$\mathbf{K} = \mathbf{X}\mathbf{W_K} = vec(\mathbf{x}) \cdot 1 = vec(\mathbf{x}) \tag{17}$$

$$\mathbf{V} = \mathbf{X}\mathbf{W_V} = vec(\mathbf{x}) \cdot 1 = vec(\mathbf{x}) \tag{18}$$

The attention probabilities $\mathbf{P}$ are given by computing scaled dot product attention and applying softmax row-wise,

$$\mathbf{P} = softmax(\mathbf{QK^T}) \tag{19}$$

where $\mathbf{QK^T}_{MN \times MN}$ is,

$$\begin{bmatrix} x_{1,1} \cdot x_{1,1} & \cdots & x_{1,1} \cdot x_{m,1} & \cdots & x_{1,1} \cdot x_{1,n} & \cdots & x_{1,1} \cdot x_{m,n} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{m,1} \cdot x_{1,1} & \cdots & x_{m,1} \cdot x_{m,1} & \cdots & x_{m,1} \cdot x_{1,n} & \cdots & x_{m,1} \cdot x_{m,n} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{1,n} \cdot x_{1,1} & \cdots & x_{1,n} \cdot x_{m,1} & \cdots & x_{1,n} \cdot x_{1,n} & \cdots & x_{1,n} \cdot x_{m,n} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{m,n} \cdot x_{1,1} & \cdots & x_{m,n} \cdot x_{m,1} & \cdots & x_{m,n} \cdot x_{1,n} & \cdots & x_{m,n} \cdot x_{m,n} \end{bmatrix} \tag{20}$$

leading to,

$$\mathbf{P} = softmax(\frac{vec(\mathbf{x}) \cdot vec^T(\mathbf{x})}{\sqrt{1}}) \tag{21}$$

where the attention output $\mathbf{Z}$ is,

$$\mathbf{Z} = \mathbf{PV} = softmax(\frac{vec(\mathbf{x}) \cdot vec^T(\mathbf{x})}{\sqrt{1}}) \cdot vec(\mathbf{x}) \tag{22}$$

2 Question 2

2.1 1.

For a Fully-Connected neural network we have an input of shape, for a single channel, of $H_{in} \times W_{in} \times C_{in}$, where C_{in} are the input channels, and the output of shape $H_{out} \times W_{out} \times C_{out}$.

Since there is a separate learnable parameter for each pixel in the input and output image, we have a bigger number of parameters.

One advantage of using a CNN over a Fully-Formed Neural Network lies in the fact that a CNN uses parameter sharing in order to obtain better classifications. The way parameter sharing works is by allowing the weights of our kernel ($M \times N$) to account for the neighborhood of the given pixel.

Since there is a separate kernel for each pair of the input and output channel, but the weights of our kernel are independent of the location, this layer can accept images of any resolution, whereas, the fully connected can work only with a fixed resolution.

Therefore due to the smaller size of the kernel in comparison to the input image, the number of parameters of our CNN will be smaller than the Fully-Formed Network.

2.2 2.

Despite having fewer parameters the CNN has better generalization on images and patterns due to its singular features.

In a CNN the max pooling layer is invariant, whereas a convolutional layer is equivariant. This leads to a better identification of low level-features for a given input image on the starting layers, and a better recognition in high-level features on the final layers.

As stated before, parameter sharing also allows a CNN to solve the classification problem by decreasing the number of parameters therefore, due to the local connectivity and image sparsity, better learn image´s natural structures. This feature also allows a CNN to be more memory and complexity efficient than a Fully-Formed layer.

A CNN is also a very good feature extractors. This means that we can extract useful attributes from an already trained CNN with its trained weights by feeding our data on each level and tune the CNN for the specific task.

This leads to our final point - a CNN allows lesser overfitting than a Fully-Formed network. This is, again, due to parameter sharing.

The higher the capacity, the greater the ability to "memorize" datasets, just like a Fully-Formed network. The lower the capacity the less prone to "memorization" and therefore over-fitting, just like a CNN.

2.3 3.

For sensors, the arrangement of information is irrelevant, unlike for CNNs. A CNN is designed to take in images.

If its input something else, and not spatial, a CNN is not the best fit and won't function as well, as it will retain spatial information unnecessarily. CNNs extract features based on spatial arrangement. If spatial structure doesn't matter, then the features it extracts will be meaningless, and spatial information essentially acts as noise.

2.4 4.

	Learning Rate = 0.01	Learning Rate = 0.0005	Learning Rate = 0.00001
Training Loss	0.4572	0.0440	0.3714
Validation Accuracy	0.8430	0.9871	0.9556
Final Test Accuracy	0.7209	0.9581	0.8896

Table 1: Test, Validation Accuracy and Training Loss for 3 different Learning Rates

Therefore, the best learning rate is 0.0005.

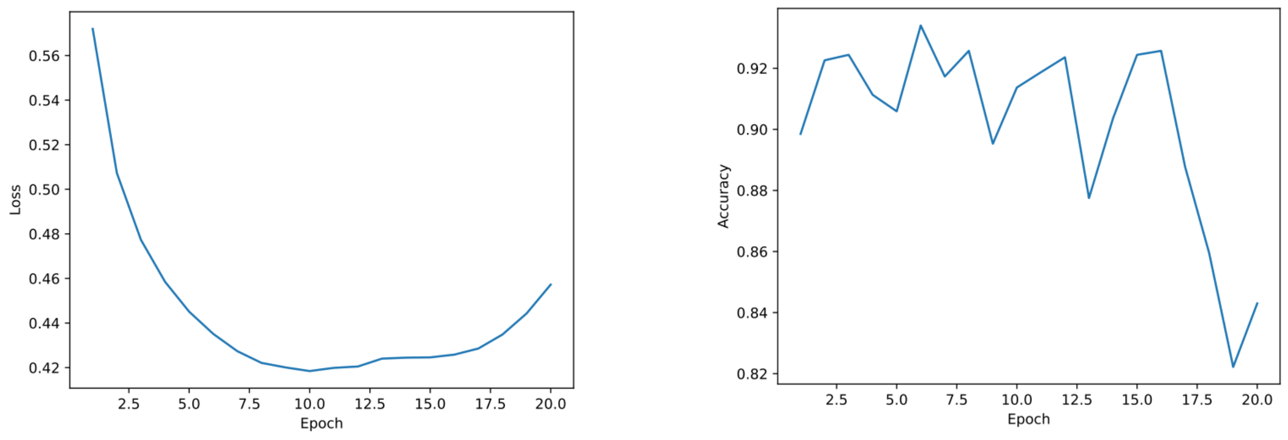

Figure 1: Training Loss and Validation Accuracy for LR = 0.01

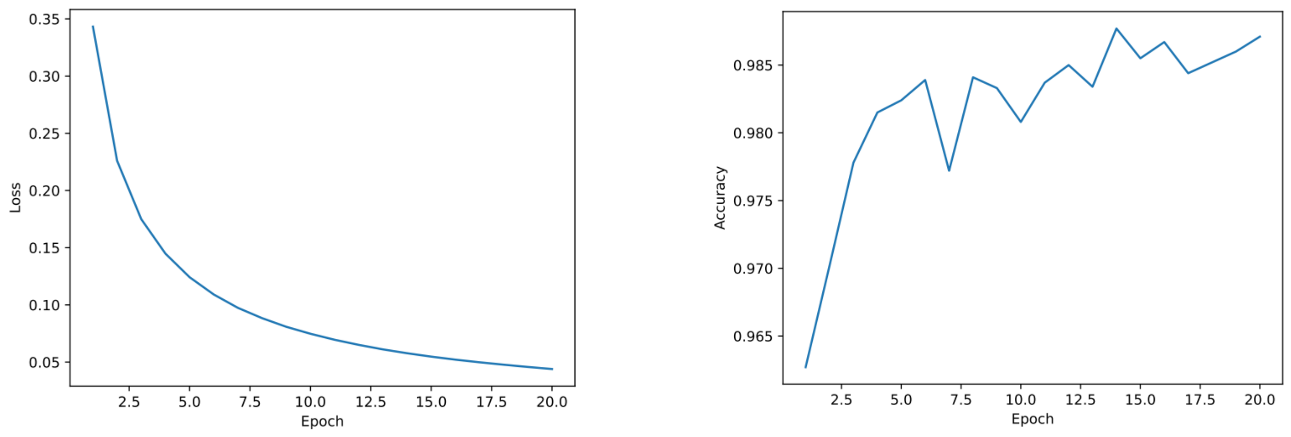

Figure 2: Training Loss and Validation Accuracy for LR = 0.0005

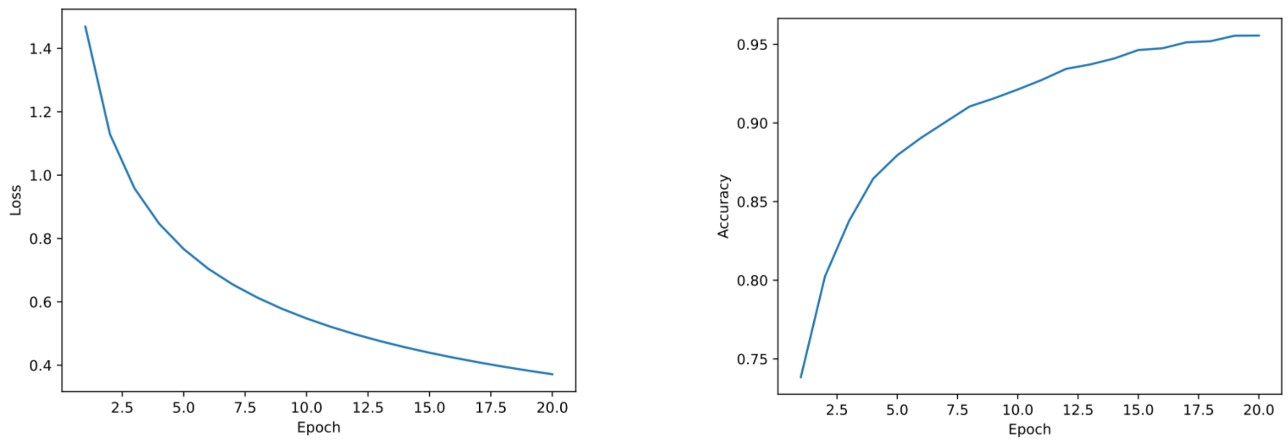

Figure 3: Training Loss and Validation Accuracy for LR = 0.00001

2.5 5.

Below we have the comparison between every set of activation maps for the 3 different learning rates, and the original image:

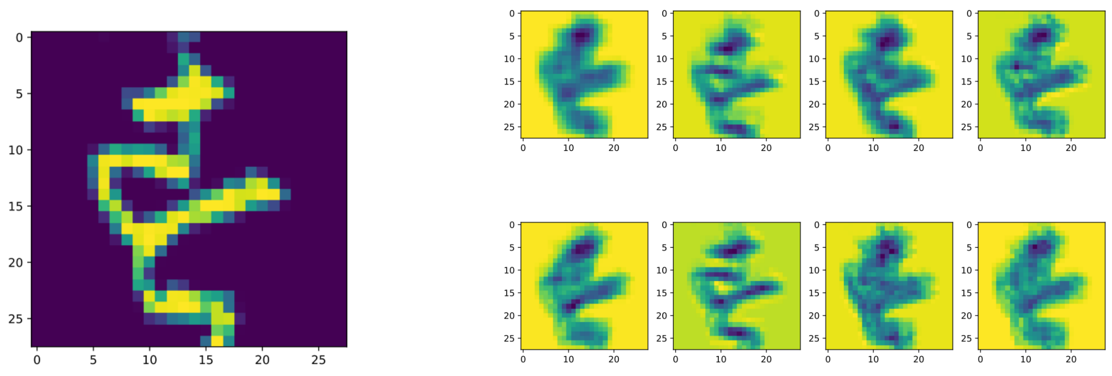

Figure 4: Original Image and Activation Maps for a LR = 0.01

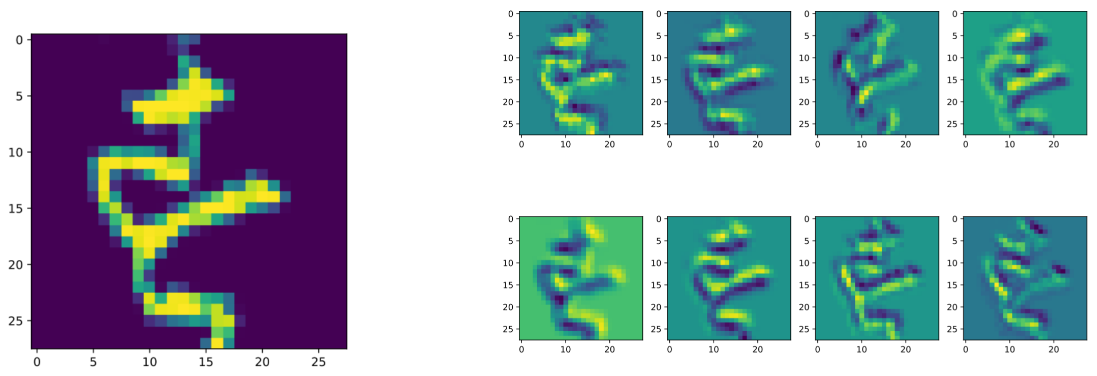

Figure 5: Original Image and Activation Maps for a LR = 0.0005

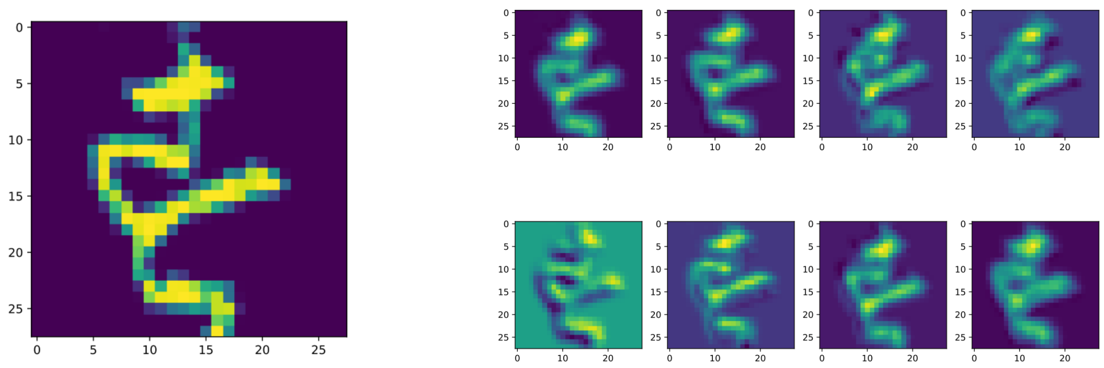

Figure 6: Original Image and Activation Maps for a LR = 0.00001

As we can see, for our better Learning Rate, we have a good highlight of the image´s edges and a general non-diffuse characteristic pattern throughout our output channels.

3 Question 3

3.1 1a)

Final Validation Error Rate	Test Error Rate	Loss
0.4997	0.4968	0.8122

Table 2: Parameters for 50 epochs from our Character Machine Translation Model, without attention mechanism

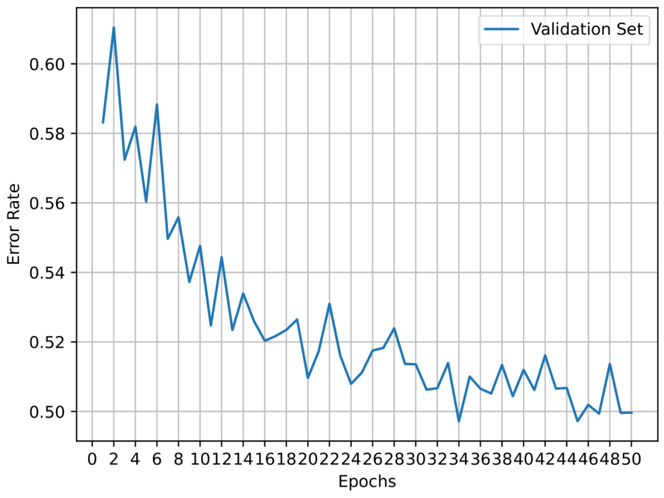

Figure 7: Validation Error Rate for 50 Epochs without Attention Mechanism

3.2 1b)

Final Validation Error Rate	Test Error Rate	Loss
0.3341	0.3440	0.5761

Table 3: Parameters for 50 epochs from our Character Machine Translation Model, with attention mechanism

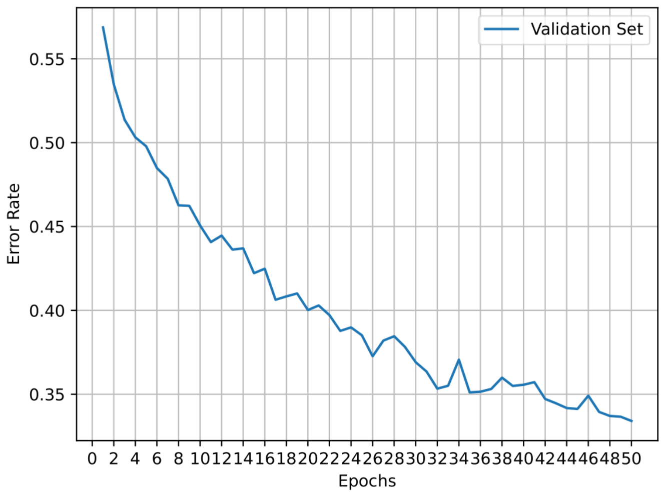

Figure 8: Validation Error Rate for 50 Epochs with Attention Mechanism

3.3 1c)

There are a few strategies we could implement in order to improve results without changing model architecture. Such are :

- Beam Search.
- Ensembling.
- General machine learning techniques.

Due to the decoder doing a greedy search, and in consequence propagating error, we could implement Beam Search. With it we could approximate our search strategy and at the decoder step keep track of the k most probable partial translations,where k is the beam size, discarding the rest. This offers a less greedy search than initially.
Nevertheless, we still have to be careful about the speed/accuracy trade-off.

With Ensembling, such as bagging, boosting or stacking we combine our independently trained models and obtain a ”consensus” by raising our accuracy.

Before we jump to general machine learning techniques it is to note that we could also have deeper LSTMs and we could reverse the source sentence in order to increase our model´s performance.

To conclude, we could always increase our dataset by getting more data or ”transforming” some of our original data and treating it as ”new” data.