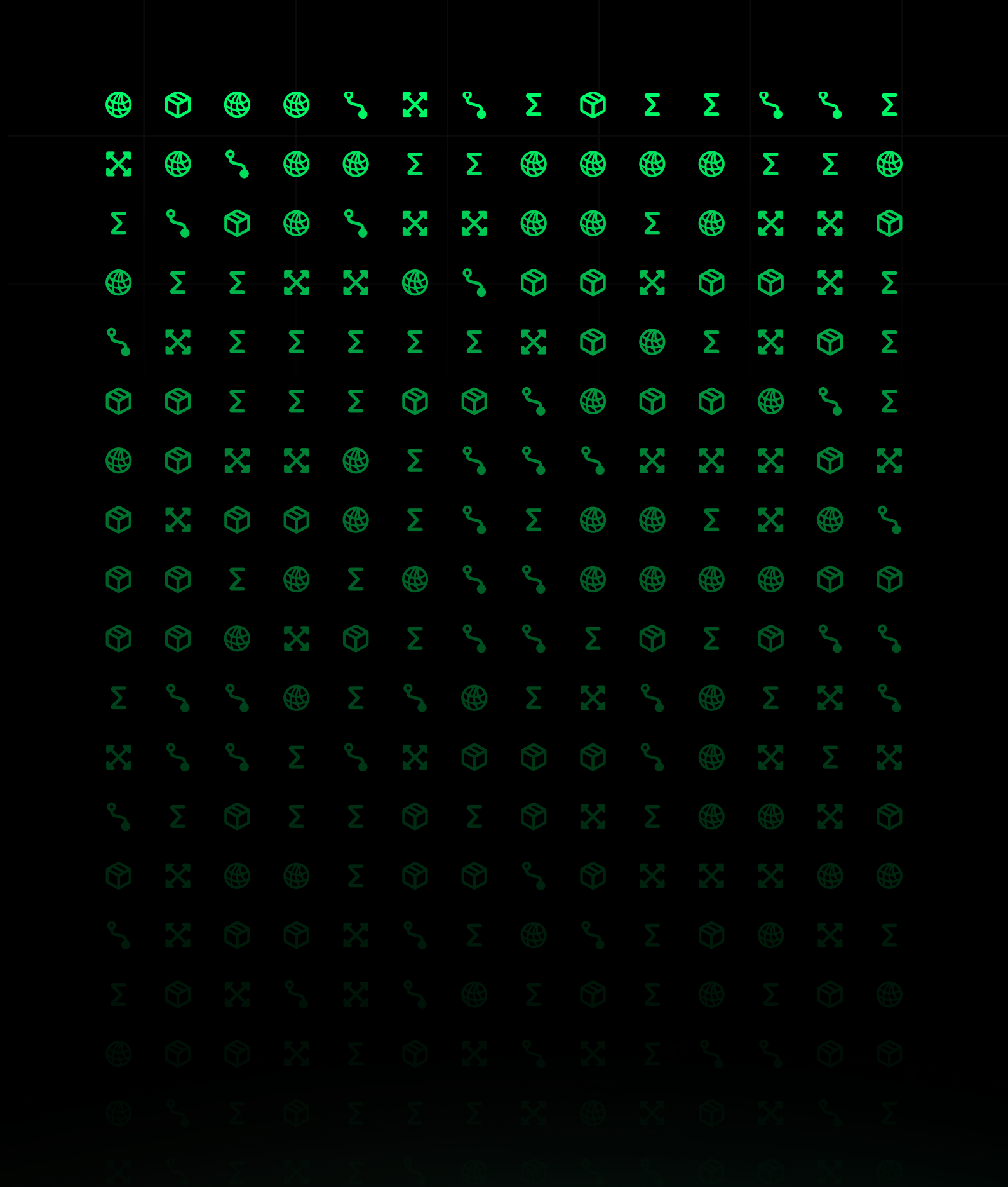

Deep Learning

Homework 2

Catarina Sousa | 93695 *Gonçalo Falcão | 102219* *Miguel Ferreira | 90899*

Deep Learning

Homework 2 | 2021-2022

Catarina Sousa | 93695
Gonçalo Falcão | 102219
Miguel Ferreira | 90899

Question 1

Exercise 1

For the first convolution layer we have 608 parameters. To get to this result is quite straightforward: we have 8 kernels/filters, each with a shape of 5 by 5, and a depth of 3. This means that for each filter we have $5 * 5 * 3$ plus the bias weights: $5 * 5 * 3 + 1 = 76$. Since we have 8 filters, for this layer we have a total of $76 * 8 = 608$ weights.

Regarding the output layer we have to first flatten the max-pooling output. The max-pooling output is dependent on the convolution layer filter size and configurations and on its own configuration. Calculating, we obtain that the max-pooling output shape will be $(11, 11, 8)$, this means that the flatten layer will result in an array with dimension $11*11*8 = 968$.

Thus, and since the output layer has 10 outputs, we have $968 * 10 + 10$ weights (plus 10 for the biases). As such, the total number of weights is $608 + 968 * 10 + 10 =$**10298** parameters.

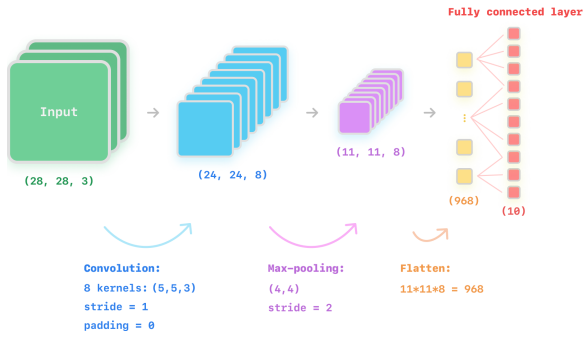

Figure 1: Network configuration.

Exercise 2

What we now have is that from the input layer with *shape* $= (28, 28, 3)$, to the single feedforward layer with hidden size 100, we have a total of $100 + 100 * (28 * 28 * 3)$ parameters, where the first 100 are from the biases!

Then, from this single feedforward layer to the output with size 10, we have a total of $10+100*10$ parameters.

This way the total number of parameters is 236310. What we conclude is, therefore, that from the convolutional neural network above, and this neural network, we have - for the CNN - a reduction of 22 times the size of the network.

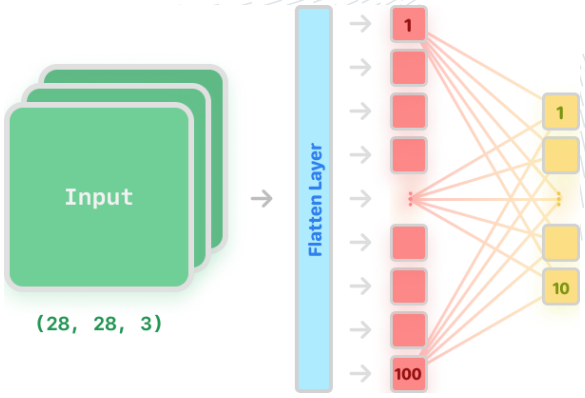

Figure 2: Network configuration.

With such a drastic reduction of parameters, and because of weight sharing, we obtain much better computational and memory performance, two of the reasons CNN's are so widely used.

Exercise 3

Exercise 3.1

We know that, from the structure of a self-attention head, we have three inputs for each head: Q, K, V. Where each one of these inputs is obtained by multiplying the matrix X with the matrices $W_{\{Q,K,V\}}$ This way we have that the result matrices $Q, K, V \in \mathbb{R}^{(L,d)}$.

Then, and regarding the structure of the self-attention heads, we know that the probabilities for each head are given by $softmax(\frac{QK^\intercal}{\sqrt{d_k}})$, where d_k is the number of rows of K

If we now enter deeper in the softmax function what we have is:

$$softmax(\frac{QK^\intercal}{\sqrt{d_k}}) =$$
$$softmax(\frac{XW_Q(XW_K)^\intercal}{\sqrt{d_k}}) =$$
$$softmax(\frac{XW_QW_K^\intercal X^\intercal}{\sqrt{d_k}})$$
$$softmax(XAX^\intercal), \text{with } A = \frac{W_QW_K^\intercal}{\sqrt{d_k}} \in \mathbb{R}^{(n,n)}.$$

If we now generalize for the case of multi-head self attention, what we have is: $A^{(h)} = \frac{W_Q^{(h)}W_K^{(h)\intercal}}{\sqrt{d_k}}$

This way, the self-attention probabilities for each head can be written, indeed, as $P^{(h)} = softmax(XA^{(h)}X^\intercal)$

Lastly, we also want to prove that the matrix $A^{(h)} \in \mathbb{R}^{(n,n)}$ has rank $\leq d$, where $d \leq n$. Note that we won't be counting with the scalar factor $\sqrt{d_k}$, since, as it is a scalar, it won't impact the rank of any of the matrices in use, furthermore, this way we can also use a more lightweight notation.

We know, as a definition that:

- W_Q: row rank $\leq n$ and column rank $\leq$ d, since $W_Q \in \mathbb{R}^{(n,d)}$
- $W_K^\intercal$: row rank $\leq d$ and column rank $\leq$ n, since $W_K^\intercal \in \mathbb{R}^{(d,n)}$
- and, $rank(AB) \leq min(rank(A), rank(B))$[2].

Thus, we have:

$rank(W_Q, W_K^\intercal) \leq min(rank(W_Q), rank(W_K^\intercal))$

– **rows:** $rank(W_Q, W_K^\intercal) \leq min(d, n) \leq d$

– **columns:** $rank(W_Q, W_K^\intercal) \leq min(n, d) \leq d$

The last step for both the rows and columns is due to the fact that $d \leq n$.

Since the rank of the matrix A is, for both the column and rows $\leq d$, then we've proven that $A^{(h)} \in \mathbb{R}^{(n,n)}$ has rank $\leq d$, where $d \leq n$.

Exercise 3.2

Using what we've concluded on the previous question, and with $W_Q^{(2)} = W_Q^{(1)}B$ and $W_K^{(2)} = W_K^{(1)}B^{-\intercal}$ (with B any invertible matrix), we have:

$$
\begin{aligned}
&P^{(2)} = \\
&softmax(X\frac{W_Q^{(2)}(W_K^{(2)})^\intercal}{\sqrt{d_k}}X^\intercal) = \\
&softmax(X\frac{W_Q^1 B(W_K^1 B^{-\intercal})^\intercal}{\sqrt{d_k}}X^\intercal) = \\
&softmax(X\frac{W_Q^1 B(B^{-\intercal})^\intercal(W_K^1)^\intercal}{\sqrt{d_k}}X^\intercal) = \\
&softmax(X\frac{W_Q^1 BB^{-1}(W_K^1)^\intercal}{\sqrt{d_k}}X^\intercal) = \\
&softmax(X\frac{W_Q^1(W_K^1)^\intercal}{\sqrt{d_k}}X^\intercal)
\end{aligned}
$$

Thus, $P^{(2)} = softmax(X\frac{W_Q^1(W_K^1)^\intercal}{\sqrt{d_k}}X^\intercal) = P^{(1)}$

This way, the self-attention probabilities are the same for the two attentions heads, as we wanted to show!

Question 2

Exercise 1

Convolutional neural networks ensure equivariance to translations [1]. What this means is that because of convolutions, a CNN ensures that for a given layer, the same filters are applied to each part of the image, thus, if we translate the input image, the network activations will be translated in the same way.

Since the filters slide from left to right and from top to bottom, they will activate whenever a given particular structure appears (that the filter was previously trained on). This way, if a filter recognises vertical lines, and if the image contains the lines, it will be activated whenever the lines are found, since the filter will slide across the entire image.

Equivariance is useful because it allows the network to generalise (for example) edge and shape detection in different locations of the image, regardless of the translation applied. What this ultimately means is that the position of an object in the image doesn't need to be fixed in order for a CNN to detect it!

Exercise 2

For this exercise we've chosen a dropout rate of **0.2**. Regarding the best configuration (the one that performed the best on the validation set), we found it using a learning rate of 0.01. For it, the training loss was 0.3175 and the validation accuracy was 0.9060. For the final test accuracy the value obtained is 0.9025.

For the other learning rates, both the validation and final test sets accuracy were lower, however, for the learning rate of 0.1, the training loss was lower than with a learning rate of 0.01.

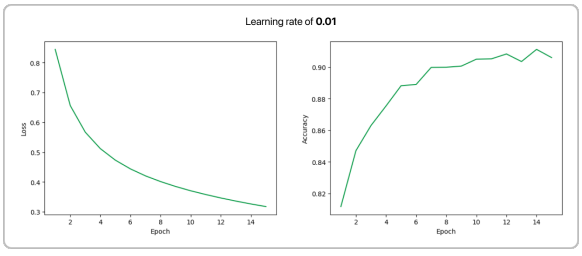

Figure 3: Plot for the best configuration, with 0.01 as the learning rate.

Exercise 3

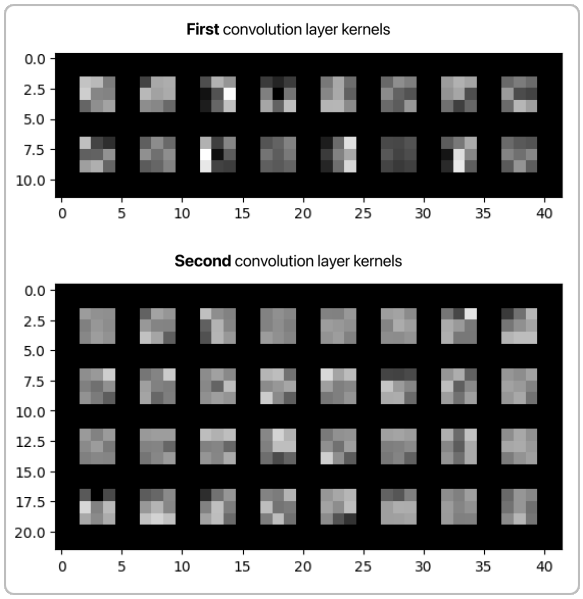

Figure 4: Kernels of the first and second convolution layers.

Typically, the lower the layer, the higher the image resolution the convolutional neural networks usually deal with. What this means is that kernels mostly capture simpler more granular details that are part of a generic object (or person, or whatsoever), like vertical or horizontal lines or curves.

Going forward to the top layers, the kernels in the network start picking up more complex figures such as overall shapes that are a combination of the finer details captured before. At the end of the network, in the top layers, it is possible to have filters that activate with certain objects in the image (for example for image classification).

One can imagine, for example, a CNN to detect road signals or road objects, that on the bottom layers the horizontal white lines of the orange road pins may be detected, as well as the vertical side white lines of a stop signal. However, on the top layers, filters can therefore make a distinction of the overall fine detail between a road pin and a stop signal, while on the bottom layers only the components of these objects were being detected.

Generically speaking, CNNs have many layers, and each one of them looks at a different level of abstraction (from simpler shapes and patterns to more complex features at the end).

Question 3

Exercise 1

Exercise 1.a

Final BLEU-4 score in the test set: 0.492717902960802.

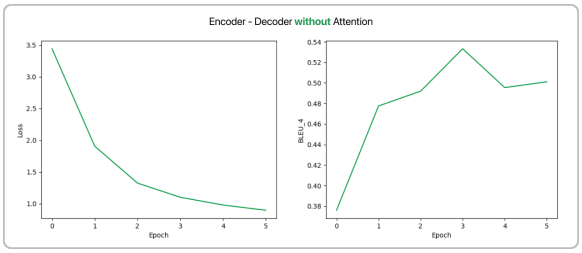

Figure 5: Loss and BLEU-4 scores for the decoder-encoder without using the attention mechanism.

Exercise 1.b

Final BLEU-4 score in the test set: 0.529442350807247.

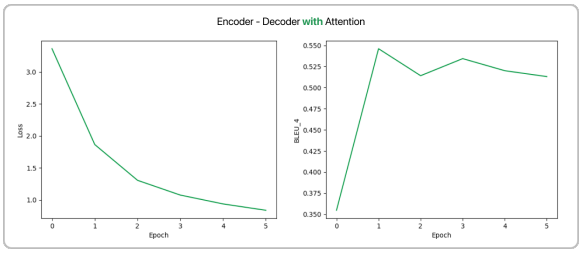

Figure 6: Loss and BLEU-4 scores for the decoder-encoder using the attention mechanism.

Exercise 1.c

Figure 7: Generated captions for images 219, 540 and 357.

References

[1] What is the difference between "equivariant to translation" and "invariant to translation". https://datascience.stackexchange.com/questions/16060/what-is-the-difference-between-equivariant

[2] Linear algebra rank. https://en.wikipedia.org/wiki/Rank_(linear_algebra).