TÉCNICO LISBOA

Deep Learning

MEEC

Homework 2

Authors:

Diogo José Pereira Araújo (93906) diogoparaujo@tecnico.ulisboa.pt
Maria Margarida Domingues Rita (93130) margarida.rita@tecnico.ulisboa.pt

Group 40

2022/2023 – 1^{st} **Semester, P2**

Individual contribution

Both members contributed equally to the resolution of this homework assignment. Maria Margarida was more in charge of the theoretical questions, while Diogo was more in charge of the code and Q1-2. Both members participated equally in Q1-1-b, only arriving at the final expression thanks to the help provided by group 66.

Question 1

1.

(a)

For a input $\boldsymbol{x} \in \mathbb{R}^{H \times W}$ and given the kernel $\boldsymbol{W} \in \mathbb{R}^{M \times N}$, we can define the following general formula for the size of the output of a convolutional layer:

$$H_{out} = \left\lfloor \frac{H - M + 2P}{S} \right\rfloor + 1 \quad \text{and} \quad W_{out} = \left\lfloor \frac{W - N + 2P}{S} \right\rfloor + 1, \text{(i)} \tag{1}$$

where S is the stride of the convolution, and P is the padding. Since, $S = 1$ and $P = 0$, we can define the output of the first convolutional layer as $\boldsymbol{z} \in \mathbb{R}^{(H-M+1)\times(W-N+1)}$.

(b)

First we want to prove that there is a matrix $\boldsymbol{M} \in \mathbb{R}^{H'W' \times HW}$, with $H' = H - M + 1$ and $W' = W - N + 1$, such that $\boldsymbol{z'} = \boldsymbol{M}\boldsymbol{x'}$. We can write $\boldsymbol{x'}$ and $\boldsymbol{z'}$ in the following manner:

$$\begin{aligned} \boldsymbol{x'} &= \begin{bmatrix} x_{11} & x_{21} & \dots & x_{H1} & x_{12} & x_{22} & \dots & x_{HW} \end{bmatrix}^\top \in \mathbb{R}^{HW} \\ \boldsymbol{z'} &= \begin{bmatrix} z_{11} & z_{21} & \dots & z_{H1} & z_{12} & z_{22} & \dots & z_{HW} \end{bmatrix}^\top \in \mathbb{R}^{H'W'} \end{aligned}. \tag{2}$$

This means that there must be a matrix $\boldsymbol{M} \in \mathbb{R}^{H'W' \times HW}$, whose elements only depend on values of $\boldsymbol{W} \in \mathbb{R}^{M \times N}$, that maps $\boldsymbol{x'}$ into $\boldsymbol{z'}$. In order to find a general expression of the elements of the matrix $\boldsymbol{M}$ the group tried numerous examples. Let's consider a rather simple one, let's say that $M = N = 2$, $H = 3$ and $W = 4$. This way we can define:

$$\boldsymbol{x} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \end{bmatrix} \in \mathbb{R}^{3\times4} \text{ and } \boldsymbol{W} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \in \mathbb{R}^{2\times2}. \tag{3}$$

This way we will have the following output $\boldsymbol{z} = \boldsymbol{W} * \boldsymbol{x} \in \mathbb{R}^{2\times3}$. Hence, we can define each element of $\boldsymbol{z}$:

(i) Those brackets denote the floor division.

$$
\begin{aligned}
z_{11} &= w_{11}x_{11} + w_{12}x_{12} + w_{21}x_{21} + w_{22}x_{22} \\
z_{21} &= w_{11}x_{21} + w_{12}x_{22} + w_{21}x_{31} + w_{22}x_{32} \\
z_{12} &= w_{11}x_{12} + w_{12}x_{13} + w_{21}x_{22} + w_{22}x_{23} \\
z_{22} &= w_{11}x_{22} + w_{12}x_{23} + w_{21}x_{32} + w_{22}x_{33} \\
z_{13} &= w_{11}x_{13} + w_{12}x_{14} + w_{21}x_{23} + w_{22}x_{24} \\
z_{23} &= w_{11}x_{23} + w_{12}x_{24} + w_{21}x_{33} + w_{22}x_{34}.
\end{aligned}
\tag{4}
$$

Given that $\boldsymbol{z}' = \boldsymbol{M}\boldsymbol{x}'$, we can define the respective $\boldsymbol{M}$ as follows:

$$
\boldsymbol{M} = \begin{bmatrix}
w_{11} & w_{21} & 0 & w_{12} & w_{22} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & w_{11} & w_{21} & 0 & w_{12} & w_{22} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & w_{11} & w_{21} & 0 & w_{12} & w_{22} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & w_{11} & w_{21} & 0 & w_{12} & w_{22} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & w_{11} & w_{21} & 0 & w_{12} & w_{22} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & w_{11} & w_{21} & 0 & w_{12} & w_{22}
\end{bmatrix}
\tag{5}
$$

Through the examination of this $\boldsymbol{M}$ and other countless examples, we can derive to some conclusions. We see that each row of the $\boldsymbol{M}$ matrix will me comprised by zeros and by a $\boldsymbol{M}$ block. Each $\boldsymbol{M}$ block is comprised by the first column of $\boldsymbol{W}$, then by a row of zeros with size $H - M$, then by the second column of $\boldsymbol{W}$, and so on. This means that each $\boldsymbol{M}$ block follows a certain pattern. Through countless tries and some algebraic manipulations the group derived to the following general expression for a given element (i, j) of $\boldsymbol{M}$:

$$
M_{ij} = \begin{cases} w_{mn}, & \text{if } \begin{cases} \big((i-1) \mod H'\big) < \big((j-1) \mod H\big) + 1 \leq M + \big((i-1) \mod H'\big) \\ \text{and } \left\lfloor \frac{i-1}{H'} \right\rfloor < j \leq (\left\lfloor \frac{i-1}{H'} \right\rfloor + 2)H \end{cases} \\ 0, & \text{otherwise} \end{cases},
$$

[ii]

$$
\text{where } \begin{cases} m &= \big((j-1) \mod H\big) - \big((i-1) \mod H'\big) + 1 \\ n &= \left\lfloor \frac{j-1}{H} \right\rfloor + 1 - \left\lfloor \frac{i-1}{H'} \right\rfloor \end{cases}
$$

(c)

First, we want to know the parameters of the network. The network is composed by:

1. A convolutional layer with one kernel size $M \times N$, stride of 1, no padding, and a ReLu activation function.
2. 2×2 max-pooling layer, with stride of 2 and no padding.
3. Output layer with a softmax activation function.

[ii] mod denotes the remainder after dividision.

In order to know number of parameters in the network, we have to know the respective number of parameters in each layer.

The size of the convolutional layer is defined by the size and number of its kernels. A kernel with the size $M \times N$ will contain $M \cdot N$ parameters. Since we have only one kernel and do not consider the bias terms, we have a convolutional layer with $M \cdot N \cdot 1$ parameters. As answered in question ***Q*1-1-*a***, the output of this layer is $\boldsymbol{h_1} \in \mathbb{R}^{(H-M+1)\times(W-N+1)}$[(iii)]. For simplicity, from now on we will consider :

$$\begin{aligned} H' &= H - M + 1 \\ W' &= W - N + 1. \end{aligned} \tag{6}$$

Next, we have the max-pooling layer. The max-pooling layer will only select the maximum value on a 2×2 neighborhood of pixels. Hence, it does not have any learnable paramters. Since, this max-pooling layer has no padding, it will map its input into an output of shape:

$$\left(\frac{H'-2}{2}+1\right) \times \left(\frac{W'-2}{2}+1\right) = \frac{H'}{2} \times \frac{W'}{2}. \tag{7}$$

Lastly, we have the output layer. This last layer will map $\boldsymbol{h_2}$ into a vector of dimension 3 (since we have 3 classes). $\boldsymbol{h_2} \in \mathbb{R}^{(\frac{H'}{2}\cdot\frac{W'}{2})\times 1}$ corresponds to the flatten output of the max-pooling layer. Thus, the output layer will the following number of learnable parameters:

$$\frac{H'}{2} \cdot \frac{W'}{2} \cdot 3. \tag{8}$$

Therefore, the network will have the following number of parameters:

$$MN + \frac{H'}{2} \cdot \frac{W'}{2} \cdot 3 = MN + \frac{3}{4}H'W'. \tag{9}$$

Now, we want to know the number of parameters if we used a fully connected layer, instead of convolutional and max-pooling layers, taking into account that $\boldsymbol{h_2}$ is a vector size $\boldsymbol{h_2} \in \mathbb{R}^{\frac{H'}{2}\cdot\frac{W'}{2}}$. The fully connected layer will map the flattened vector $\boldsymbol{x'} \in \mathbb{R}^{HW}$ into $\boldsymbol{h_2}$. Hence, the fully connected layer will have:

$$HW \cdot \frac{H'}{2} \cdot \frac{W'}{2} \quad \text{parameters.} \tag{10}$$

The output layer is the same as it is described in (8). Hence, the fully connected network will have in total:

$$(HW+3) \cdot \frac{H'}{2} \cdot \frac{W'}{2} = \frac{1}{4}(HW+3)H'W'. \tag{11}$$

Note that all the divisions in this exercise stand for the floor division. Through examination of (9) and (11) we can conclude that the number of parameters of CNN is fewer than in a fully-connected layer, as one would expect.

[(iii)] Note that the ReLu activation function does not interfere with the dimensions of $\boldsymbol{z}$.

2.

The self-attention mechanism is quite complex. So we will explain it step-by-step. First we will compute the the $\boldsymbol{Q}, \boldsymbol{K}$ and $\boldsymbol{V}$ matrices:

$$\boldsymbol{Q} = \boldsymbol{x'} \cdot \boldsymbol{W_Q}, \quad \boldsymbol{K} = \boldsymbol{x'} \cdot \boldsymbol{W_K}, \quad \boldsymbol{V} = \boldsymbol{x'} \cdot \boldsymbol{W_V}, \qquad \text{where: } \boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{(HW)\times 1}. \tag{12}$$

Since $\boldsymbol{W_Q} = \boldsymbol{W_K} = \boldsymbol{W_V} = 1$, we have: $\boldsymbol{Q} = \boldsymbol{K} = \boldsymbol{V} = \boldsymbol{x'}$. Let us define:

$$\boldsymbol{Q} = \boldsymbol{K} = \boldsymbol{V} = \boldsymbol{x'} = \begin{bmatrix} x_1 & x_2 & \dots & x_{HW} \end{bmatrix}^T. \tag{13}$$

Now e can compute $\boldsymbol{QK^T}$ affinities:

$$\boldsymbol{QK^T} = \boldsymbol{x'}(\boldsymbol{x'})^{\boldsymbol{T}} = \begin{bmatrix} x_1x_1 & x_1x_2 & \dots & x_1x_{HW} \\ x_2x_1 & x_2x_2 & \dots & x_2x_{HW} \\ \vdots & \vdots & \ddots & \vdots \\ x_{HW}x_1 & x_{HW}x_2 & \dots & x_{HW}x_{HW} \end{bmatrix}. \tag{14}$$

The next step consists in computing the attention map $\boldsymbol{A} \in \mathbb{R}^{(HW)\times(HW)}$, i.e., the attention probabilities matrix:

$$\boldsymbol{A} = softmax\left(\frac{\boldsymbol{QK^T}}{\sqrt{d_k}}\right) = softmax\big(\boldsymbol{x'}(\boldsymbol{x'})^{\boldsymbol{T}}\big).^{\text{(iv)}} \tag{15}$$

Where, we can the attention weight (i.e., attention probability) A_{ij} as:

$$A_{ij} = \frac{exp(x_i x_j)}{\sum_{k=1}^{HW} exp(x_i x_k)}. \tag{16}$$

Now, we can compute the self-attention head $\boldsymbol{Z} \in \mathbb{R}^{HW\times 1}$:

$$\boldsymbol{Z} = \boldsymbol{AV} = softmax\big(\boldsymbol{x'}(\boldsymbol{x'})^{\boldsymbol{T}}\big) \cdot \boldsymbol{x'}. \tag{17}$$

Question 2

1.

In a fully-connected network, each neuron in one layer is connected to every neuron in the next layer. This means that each neuron has its own set of weights, resulting in a large number of learnable parameters. On the other hand, CNNs have some important properties that make this kind of networks have fewer parameters compared to fully-connected networks.

For instance, through the use of small convolutional filters, CNNs drastically reduce the number of parameters needed to be learned by the network, as the network does not need to learn separate weights for each possible location of a feature in the image (locality). The sparse/local connectivity and parameter sharing induced by the convolutional kernel are other

(iv) Where $d_k = 1$.

properties that further reduce the parameters of the CNNs, compared to fully-connected networks. For instance, parameter sharing allows CNNs to apply the same set of weights to multiple regions of the input image instead of having a different set of weights for each part of the image, thus reducing the number of parameters that need to be learned.

In figure 1, it is possible to see the difference between the number of parameters in a fully-connected layer and a convolutional layer for the same input. Figure 1 confirms that for the same input, a convolutional layer will have much fewer parameters than a fully-connected one.

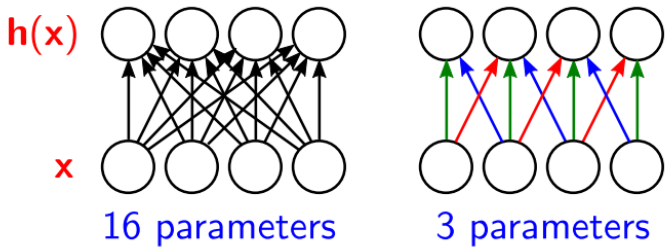


Figure 1: Difference between the number of connections between a fully-connected layer and a convolutional layer. On the left, it is represented the number of parameters of a fully-connected layer. On the right, it is possible to see the number of parameters of a convolutional layer.

2.

CNNs usually achieve better generalization on images and patterns representing letters and numbers than fully-connected networks due to CNNs being designed to take advantage of the spatially-local correlation present in images and patterns, i.e, CNNs are capable of using the correlated information of neighboring pixels, which improves the ability of the CNNs to learn more abstract and complex features compared with a fully-connected network.

In general, CNNs are capable to learn spatial hierarchies of features. In earlier layers, CNNs can learn simple features such as edges and textures, and deep layers learn more complex features such as shapes and objects. For instance, using convolutional kernels leads to several important properties that allow CNNs to learn these small, localized patterns in the input data. One of these properties is translational invariance, which enables the network to recognize a certain feature regardless of its position in the input image. Hence, allowing CNNs to learn more useful local representations compared with fully-connected layers. Additionally, CNNs use pooling layers, which down-sample the input image, further reducing the amount of information that needs to be processed by the network and making it even more robust to small translations and distortions in the input image. This makes CNNs less prone to overfitting and better suited for generalization.

In contrast, fully-connected networks do not take into account the spatial structure of the input and treat all the pixels independently. This lack of consideration for the spatial structure of the input may lead to a less efficient use of the parameters and a greater chance of overfitting.

Additionally, CNNs have fewer parameters than fully-connected networks, which means that they are less likely to overfit to the training data. This is particularly important when working with image and pattern data.

3.

As mentioned in the previous question, CNNs take into account the spatial relationships between the pixels or features in the input data, which makes CNNs better equipped for image processing compared with fully-connected networks. If the input data is from a source composed of independent sensors (with no spatial structure), CNNs won't be able to correlate spatial information between neighboring elements, not taking advantage of one of the most important characteristics of CNNs. That is why in such cases be the best choice. In such cases, a fully connected network may be more appropriate as it does not rely on the spatial structure of the input data and can learn global patterns in data without relying on local connectivity.

4.

In table 1 it is possible to see the results obtained for different learning rates. Figures 2 and 3 show the evolution of the validation accuracy and training loss, respectively. Additionally, the group just wants to make some important remarks regarding the implementation:

1. The group decided to use a padding of 2 in the first convolutional layer. Since the input image has a shape of 28×28 and the kernel size of the first convolutional layer is 5×5 we need a padding of 2[(v)] so the center of the filter passes through all the pixels in the image, hence preserving the shape of the image during the convolution.

2. In order to know the shape of the flattened input of the first fully-connected layer, the group followed the same process as in ***Q*1-1-*c***. As seen in the previous point, at the end of the first convolutional layer we will have an output of size 28×28. Next, the first max-pooling layer gives us an output of size 14×14 ($\frac{28-2}{2} + 1 = 14$). The second convolutional layer has no padding and a stride equal to one, hence it will have an output of shape 12×12 ($\frac{14-3}{1} + 1 = 12$). The second max-pooling layer gives us an output of 6×6 ($\frac{12-2}{2} + 1 = 6$). Therefore, the first fully-connected layer will have an input with shape $(16 \cdot 6 \cdot 6) \times 1$.

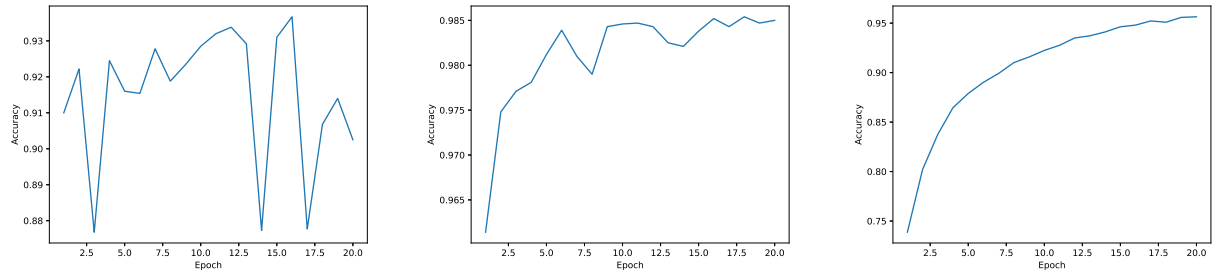
Learning rate	Training loss	Validation set acc.	Test set acc.
0.00001	0.3712	0.9564	0.8898
0.0005	**0.0433**	**0.9850**	**0.9563**
0.01	0.4348	0.9025	0.7978

Table 1: Training loss, validation accuracy and test accuracy for the last epoch, for the learning rates: 0.00001, 0.0005 and 0.01.

Comparing the plots shown in figures 2 and 3 it is possible to see the influence that the learning rate has on the model's performance. The learning rate determines how soon our model may converge to a local minimum, which means that it controls the rate or speed at which the model learns. Case (a) is the perfect example of the learning rate being set too high. This causes the model to oscillate, rather than converge to a local minimum.

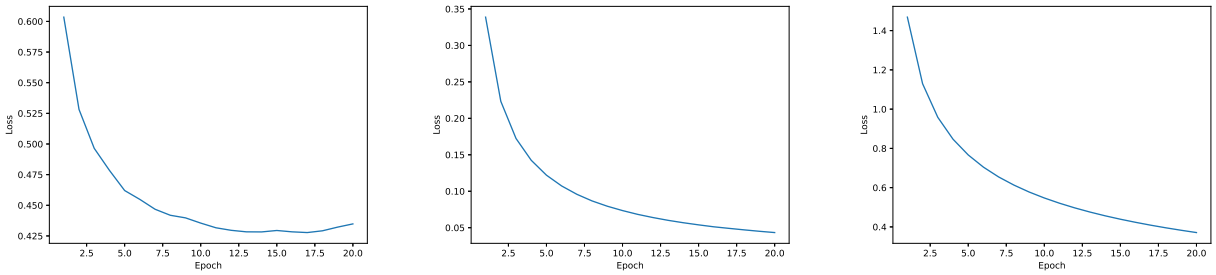
(v) $padding = \frac{F-1}{2}$, where F represents the size of the kernel.

As it is possible to see from table 1 the best results correspond to the ones from $lr = 0.0005$. However, observing the plots in figure 2, we can see that the model is much more stable for $lr = 0.00001$, despite a slower convergence.



(a) (b) (c)

Figure 2: Evolution of the accuracy of the validation set for three different learning rates: (a) 0.01, (b) 0.0005, (c) 0.00001.



(a) (b) (c)

Figure 3: Evolution of the training loss for three different learning rates: (a) 0.01, (b) 0.0005, (c) 0.00001.

5.

Figure 4 displays the original image and figure 5 shows the activation maps for the first convolutional layer. As we can see in figure 5, the activation maps of the first convolutional layer highlight more local and low-level features like edges, corners, and lines in some local regions of the image.

As the network goes deeper, we can expect the feature maps to represent more abstract and complex patterns and features in the images. This enables the network to have a better grasp of the whole image, recognizing the presence of whole objects, their parts, and their attributes.

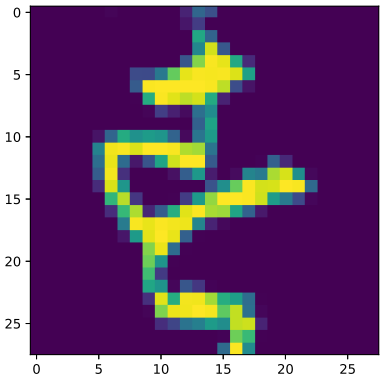


Figure 4: Original image.

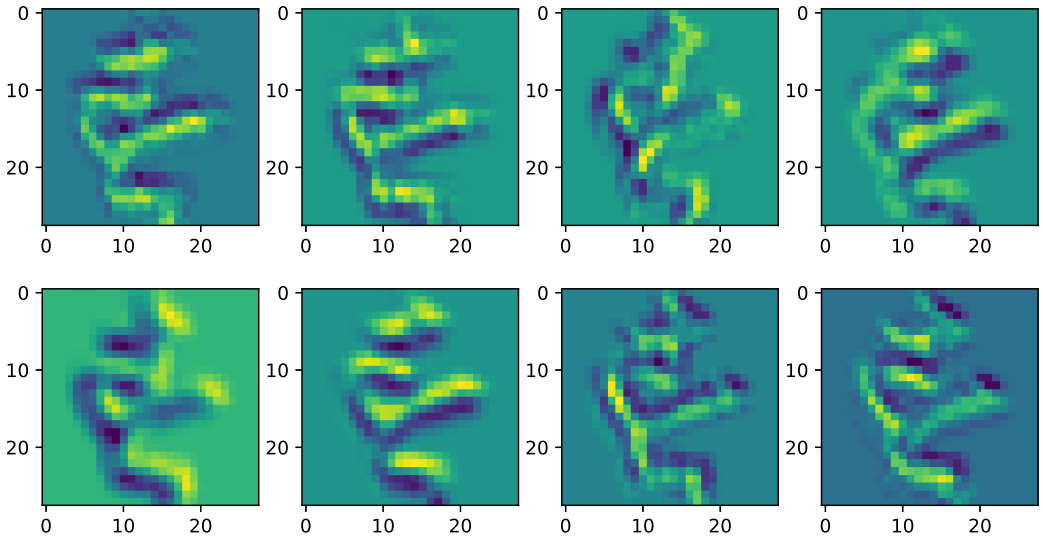


Figure 5: Activation maps of the first convolutional layer for the best configuration of ***Q*2-4**, i.e., $lr = 0.0005$.

Question 3

1.

(a)

Table 2 shows the final error rates for the validation and test sets. Figure 6 displays the validation error rate over 50 epochs.

Regarding the implementation, the group carefully placed comments throughout the code in order to make the group's reasoning more explicit. There is only one important thing to note. When we are doing character-level machine translation, the decoder usually generates one extra token, which is usually an end-of-sentence token. In order for the program to run properly we had to remove this end-of-sentence token for each element.

Epoch	Validation Error Rate	Test Error Rate
50	0.5015	0.5047

Table 2: Validation and test error rates of the model for the last epoch.

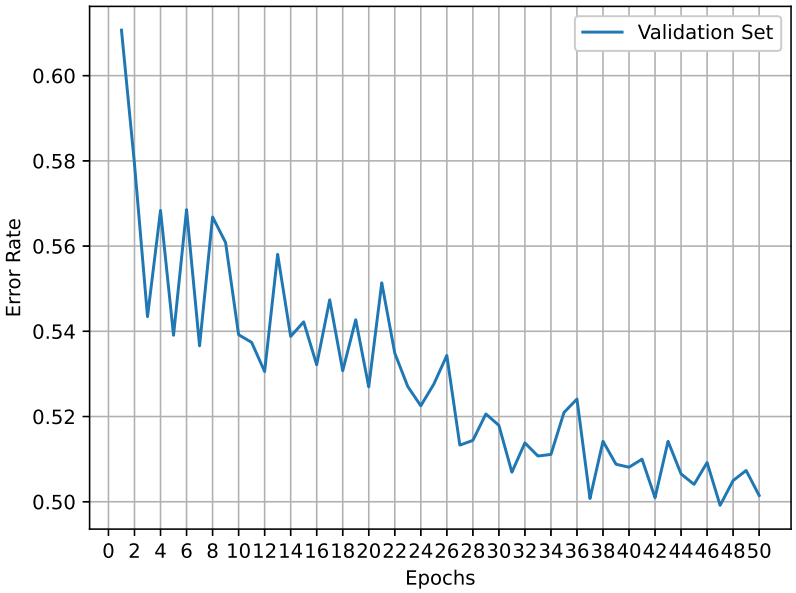


Figure 6: Plot of the validation error rate over 50 epochs.

(b)

Table 3 shows the final error rates for the validation and test sets. Figure 7 displays the validation error rate over 50 epochs. Comparing the results in tables 3 and 2, it is clear that adding the bilinear attention mechanism to the model substantially improved the results obtained.

In our model, the bidirectional LSTM encoder processes the input sequence in both directions, forwards and backwards. This allows the encoder to capture both the past and future context of each input element. The autoregressive LSTM decoder is trained to generate the target sequence one element at a time, based on the previous elements of the sequence. Adding the bilinear attention mechanism allows the decoder to take into account the whole context of the encoder's output at the same time, which improves the model's ability to capture the dependencies between the input and output sequences. This leads to more accurate translations, which explains the improvement in the results.

Epoch	Validation Error Rate	Test Error Rate
50	0.3444	0.3544

Table 3: Validation and test error rates of the model for the last epoch.

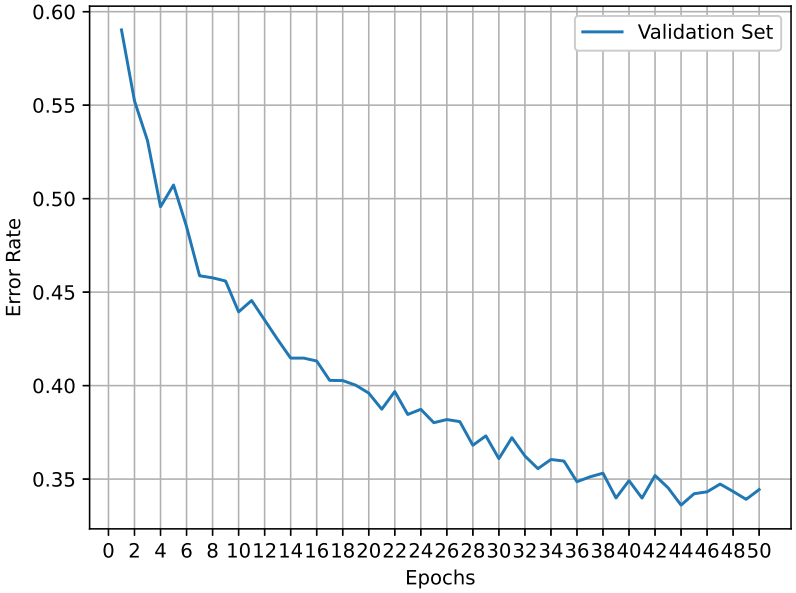


Figure 7: Plot of the validation error rate over 50 epochs.

(c)

Looking at the decoding procedure in the **test()**, we may, for example, switch from a greedy decoding method, in which the model chooses the most likely next word at each time step, to a beam search decoding technique.

Beam search is a search method that instead of extending all potential candidates, investigates a small collection of the most promising possibilities at each stage. It can assist the model in considering numerous possible translations at each decoding step, resulting in more accurate translations in the case of machine translation. The beam search method keeps track of the k most likely partial translations and creates all potential future words for each partial translation at each phase. Then, depending on the likelihood of the translation created thus far, it chooses the k most likely next words and continues the decoding process with these k most likely next words. Intuitively it makes sense that this gives us better results over the greedy search, since the beam search method avoids the issue of becoming stuck in a local optimum, which can occur with greedy decoding.

Thus, beam search can improve the accuracy of translations by considering multiple possible translations at each decoding step, rather than simply choosing the most likely next word at each time step as in greedy decoding.