Deep Learning (IST, December 2022), Homework 2

105495 - Nicolas Garcia, 105108 - Michael Lappert

Contributions

Michael Lappert did exercise 2 and 3 and Nicolas Arteaga exercise 1. Both tested and proof read the whole project.

1 Question 1

1.1 Question 1 (a)

Given the input $x \in \mathbf{R}^{H \times W}$, the filter with dimension $\mathbf{R}^{M \times N}$, the stride = 1 and no padding.

We know from the lecture 7 slide 13 that the formula to get the output dimension is $M = (\frac{N-F}{S}) + 1$ being M the output dimension, N the dimension of input, F the dimension of filter and S the stride used. $M \in \{height, width\}$

If we use the inputs we get:

1. the new height dimension $H' = (H - M)/1 + 1 = H - M + 1$
2. the new width dimension $W' = (W - N)/1 + 1 = W - N + 1$

And this shows that $z \in \mathbf{R}^{H' \times W'}$ which is the same as $z \in \mathbf{R}^{(H-M+1)\times(W-N+1)}$

1.2 Question 1 (b)

1.2.1 1.

It is possible to calculate the convolution of a 2D input and kernel, where the kernel is already flipped, using a doubly block circulant matrix, which is a type of Toeplitz matrix. Toeplitz matrices can be constructed for any kernel size.

The operation $z' = Mx'$ can be represented as a matrix multiplication, where $M \in \mathbf{R}^{H'W' \times HW}$ is a Toeplitz matrix. An example of this is the convolution of H and x', which can be written as:

$$z' = H * x' = \begin{bmatrix} h_1 & 0 & \cdots & 0 & 0 \\ h_2 & h_1 & & \vdots & \vdots \\ h_3 & h_2 & \cdots & 0 & 0 \\ \vdots & h_3 & \cdots & h_1 & 0 \\ h_{m-1} & \vdots & \ddots & h_2 & h_1 \\ h_m & h_{m-1} & & \vdots & h_2 \\ 0 & h_m & \ddots & h_{m-2} & \vdots \\ 0 & 0 & \cdots & h_{m-1} & h_{m-2} \\ \vdots & \vdots & & h_m & h_{m-1} \\ 0 & 0 & 0 & \cdots & h_m \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

Which is equal to:

$$z^T = \begin{bmatrix} h_1 & h_2 & h_3 & \cdots & h_{m-1} & h_m \end{bmatrix} \begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_n & 0 & 0 & 0 & \cdots & 0 \\ 0 & x_1 & x_2 & x_3 & \cdots & x_n & 0 & 0 & \cdots & 0 \\ 0 & 0 & x_1 & x_2 & x_3 & \dots & x_n & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & 0 & x_1 & \cdots & x_{n-2} & x_{n-1} & x_n & 0 \\ 0 & \cdots & 0 & 0 & 0 & x_1 & \cdots & x_{n-2} & x_{n-1} & x_n \end{bmatrix}$$

This result is equivalent to the one obtained by applying a sliding window of filter k over x.

1.2.2 2.

What is the general expression for element (i, j) of M?

The general expression for the element at position (i, j) of a Toeplitz matrix M is given by $M_{(i-j)}$. This means that the elements on the main diagonal of the matrix are identical and the elements above and below the main diagonal are identical but shifted. In other words, the elements of a Toeplitz matrix are determined by a single function of the difference of the row and column indices.

1.3 Question 1.1 (c)

1.4 Question 1.2

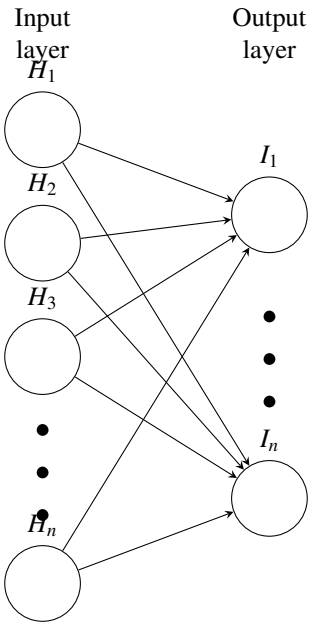

Figure 1: Fully connected layer without bias

The total parameters of our network are the parameters we find in the convolutional layer and in the fully connected layer (FC). Since the dimension filter has the dimension NM and we have an FC from h2 with $h_2 \in \mathbf{R}^{\frac{H-M+1}{2} \times \frac{W-N+1}{2}}$ to three parameters. A FC has the same parameters as the dimension of the input times the dimension of the output + the bias (which we are gonna ignore for this exercise). In Figure 1 we can tell we need a parameter for the connection of every input H with every output I, resulting in $H * I$ parameters.

The total parameters in our network are $= NM + \frac{H-M+1}{2} * \frac{W-N+1}{2} * 3$.

The replacing of the convolutional and max pool layers with one fully connected layer would result in a FC layer with $NM * \frac{H-M+1}{2} * \frac{W-N+1}{2}$ parameters. And the FC from our original network with $h_2 \in \mathbf{R}^{\frac{H-M+1}{2} \times \frac{W-N+1}{2}}$ to three parameters.

Resulting in a network with the total parameters of $= NM * \frac{H-M+1}{2} * \frac{W-N+1}{2} + \frac{H-M+1}{2} * \frac{W-N+1}{2} * 3$.

This new network would be a lot more computational intensive as the other one, since we are adding not only a new fully connected layer, but one with so many more input and output parameters. This would increase the computing time a lot and make the network really inefficient since it would not use the expressivity and simplicity that convolutional layers (with their filtering) give us.

2 Question 2

This section is all about image classification with CNNs. Subsections 2.1, 2.2, 2.3 and 2.3.2 are about theoretical questions. Subsection 2.3.1 states the properties of the implemented CNN to perform classification on the Kuzushiji-MNIST dataset.

2.1 Question 2.1

A convolutional neural network (CNN) has fewer free parameters than a fully-connected network with the same input size and number of classes because a CNN uses shared weights and biases. In a fully-connected network, each neuron in a layer is connected to every neuron in the previous layer, which means that the number of weights and biases is equal to the product of the number of neurons in the layers. In a CNN, each neuron in a layer is only connected to a small region of the previous layer due to Pooling (average or maximum) and the convolution itself, which reduces the number of weights and biases.

2.2 Question 2.2

Despite having fewer free parameters, a convolutional neural network (CNN) often achieves better generalization on images and patterns representing letters and numbers than a fully-connected network because it is better suited to processing spatial information. In a fully-connected network, the input is treated as a flat vector of features, and the relationships between the different elements of the input are not taken into account. In a CNN, the input is typically an image, and the neurons in the convolutional layers are arranged in a grid pattern. This allows the CNN to learn local patterns in the input and to use this information to make predictions.

For example, consider a task of classifying handwritten digits. A fully-connected network would treat each pixel in the input image as a separate feature and would not take into account the fact that the pixels are arranged in a grid. A CNN, on the other hand, would use the spatial relationships between the pixels to learn local patterns in the input and would be able to use this information to make more accurate predictions. Also the CNN has fewer free parameters and shared biases (see subsection 2.1 which prevents the CNN from overfitting.

2.3 Question 2.3

If the input is from a source composed of independent sensors with no spatial structure, a convolutional neural network (CNN) may not necessarily achieve better generalization than a fully-connected network. This is because CNNs are specifically designed to process spatial information and to learn local patterns in the input. When the input has no spatial structure, the CNN may not be able to utilize the grid-like structure of its layers to learn patterns in the data, and a fully-connected network may perform just as well or even better.

Considering a task of predicting the temperature at a given location based on the temperatures measured by a number of sensors located at different locations. In this case, the input to the network is a vector of sensor readings, and there is no spatial structure in the data. A fully-connected network would be able to learn the relationships between the different sensor readings and make predictions based on these relationships, and a CNN may not provide any additional benefit.

It is important to note that CNNs can still be useful for processing inputs with no spatial structure in some cases. For example, a CNN could be used to process audio signals. In this case, the CNN could learn local patterns in the signal and use this information to make predictions. Overall, the choice of which type of network to use is dependent on the characteristics of the input data and the task at hand.

2.3.1 Question 2.4

The goal of this exercise is to implement a simple convolutional neural network (CNN). The structure of the network is as follows:

- Convolutional layer with 1 input and 8 output channels, kernel size of 5x5, stride of 1 and padding of 2 (to preserve original image size)
- Activation function (relu)
- Max pooling layer (kernel size = 2x2, stride = 2)
- Convolutional layer with 8 input and 16 output channels, kernel size of 3x3, stride of 1 and padding of 0
- Activation function (relu)
- Max pooling layer (kernel size = 2x2, stride = 2)
- Linear layer with (16*6*6=) 576 input and 600 output features
- Activation function (relu)
- Dropout (p = 0.3)
- Linear layer with 600 input and 120 output features
- Activation function (relu)
- Linear layer with 120 input and 10 (= Nr. of classes) output features

The network is trained for 20 epochs with **Adam** as optimizer. Hyperparameter tuning is done only for the learning rate with values of 0.00001, 0.0005 and 0.01. The final validation and test accuracies of these learning rates are shown in table 1. Since the used input data (Kuzushiji-MNIST) is a sklearn dataset and given in vector form, the *init* method of the *ClassificationDataset* class in *utils.py* was adjusted to be processed in batches with the following lines of code:

```
        train_X, train_y = data["train"]
        dev_X, dev_y = data["dev"]
        test_X, test_y = data["test"]
```

Listing 1: Added code to ClassificationDataset class in utils.py

	Learning Rate	Validation Accuracy	Test Accuracy
1.	0.00001	0.9568	0.8938
2.	0.0005	0.9857	0.9541
3.	0.01	0.1022	0.1000

Table 1: Final validation and test accuracy of learning rate hyperparameter search

Best performance was found with learning rate 0.0005 with a final validation accuracy of 0.9857 and a test accuracy of 0.9541. The corresponding training loss is shown in figure 2 and the validation accuracy in figure 3.

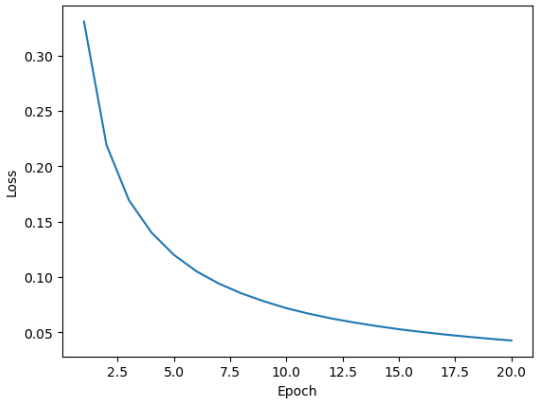

Figure 2: Training loss of best performing CNN with learning rate 0.0005

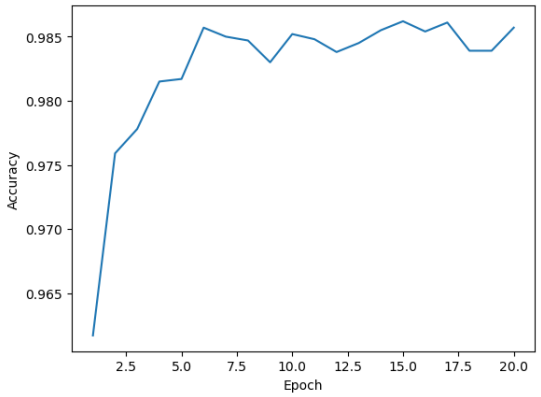

Figure 3: Validation accuracy of best performing CNN with learning rate 0.0005

2.3.2 Question 2.5

In this section the original training example (figure 4) and the activation maps of the first convolutional layer of the corresponding training example (figure 5) of the best performing CNN in section 2.3.1 are being compared. Generally spoken the activation maps highlight the presence of certain features or patterns in the input data. These features or patterns are found through the applied filters in the convolutional layer. The activation maps then shows which filters have been activated. In the given activation maps (figure 5) the highlighted parts are depicted in dark blue. Comparing the given training example (figure 4) and activation maps (figure 5) it can be observed that the first convolutional layer learned especially the edges. The top left activation map in the figure 5 highlights all the upper 'ends' of the training example whereas e.g. the top right activation map in figure 5 highlights all the 'bottom' parts (or edges pointing to the bottom of the image).

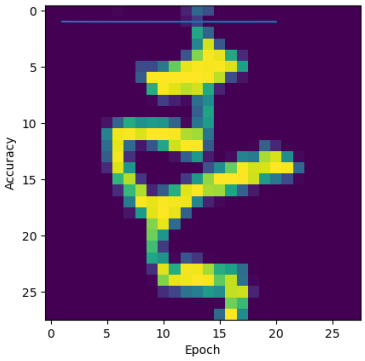

Figure 4: Original training example

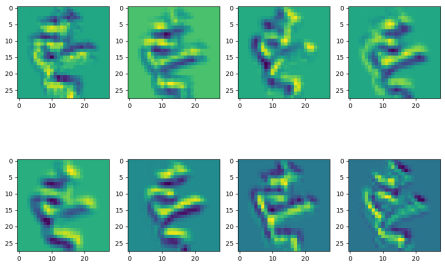

Figure 5: Activation maps of the first convolutional layer of best performing CNN in section 2.3.1

2.4 Question 3

2.4.1 Question 3.1 (a)

In this exercise a character-level Machine Translation model (from Spanish to English) is build. This is done by an encoder-decoder based architecture. The encoder is based on bidirectional LSTMs which passes the produced representations into an autoregressive LSTM decoder. In this model was no attention mechanism used (see section 2.4.2). The used parameters for training can be found in table 2.

Hyperparameters	Input
Epochs	50
Learning rate	0.003
Dropout	0.3
Hidden size	128
Batch size	64

Table 2: Hyperparameters used for the training of the seq2seq models

The validation error rate over the epochs is shown in figure 6 an the resulting metrics are as follows:

- Final Validation Error Rate: 0.4830
- Test Error Rate: 0.5026

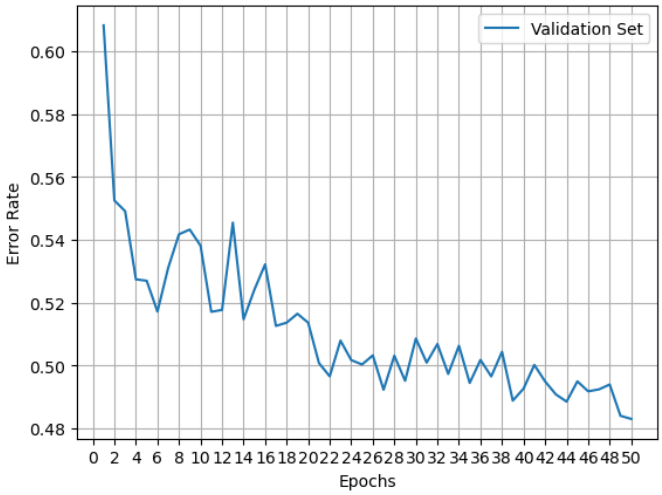

Figure 6: Validation error rate over 50 epochs of seq2seq model without attention

2.4.2 Question 3.1 (b)

In this exercise a bilinear attention mechanism is implemented in the seq2seq model from section 2.4.1. The training parameters stay the same as stated in table 2. The validation error rate over the epochs is shown in figure 7 an the resulting metrics are as follows:

- Final Validation Error Rate: 0.3648
- Test Error Rate: 0.3749

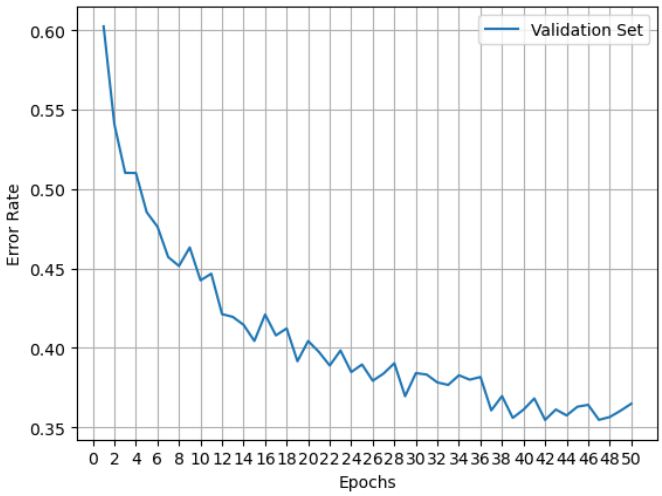

Figure 7: Validation error rate over 50 epochs of seq2seq model with bilinear attention

2.4.3 Question 3.1 (c)

Thinking about how to improve results three major topics can be addressed:

- Fine-tuning the hyperparameters
- Different evaluation metric
- Beam search

Fine-tuning the hyperparameters of the model, such as the learning rate, the batch size, the number of hidden layers, the dropout rate, and the weight decay. This would help to optimize the model's performance by adjusting the model's parameters to better fit the specific dataset and task as well as the ideal combination between the hyperparamenters.

Another way to improve results would be to use a **different evaluation metric**. The script used the Levenshtein distance to evaluate the model's performance, which compares the edit distance between the predicted and true sentences. However, other metrics such as BLEU, METEOR and ROUGE are more commonly used in machine translation and are more robust to errors in decoding.

Another option to improve results without changing the model architecture is to use **beam search decoding** instead of greedy decoding during the test process. Beam search decoding generates multiple candidate translations at each decoding step and selects the top k candidates based on their probability scores, whereas greedy decoding, which is implemented at the moment, simply selects the most likely candidate at each step. By using beam search, we can consider multiple potential translations at each step and potentially find a better overall translation. This can improve results because it allows the model to consider multiple possibilities and make more informed decisions, rather than committing to a single choice at each step.