Deep Learning (IST, 2022-23)

Homework 2

André Martins, Francisco Melo, Gonçalo Correia, João Santinha

Deadline: Friday, January 13, 2023.

Please turn in the answers to the questions below in a PDF file, together with the code you implemented to solve them (when applicable).

IMPORTANT: Please write 1 paragraph indicating clearly what was the contribution of each member of the group in this project. A penalization of 10 points will be applied if this information is missing.

Please submit **a single zip file** in Fenix under your group's name.

Question 1 (30 points)

1. (30 points) Consider the convolutional neural network in Fig. 1, used to classify images $\boldsymbol{x} \in \mathbb{R}^{H \times W}$ into 3 possible classes. The network has a convolutional layer with a single $M \times N$ filter, a stride of 1 and no padding, and a ReLU non linearity. This layer is followed by a 2×2 max pooling layer with a stride of 2 and no padding, and an output layer with softmax activation.

   Let $\boldsymbol{W}$ denote the filter weights, $\boldsymbol{z}$ the result of the convolution (i.e., $\boldsymbol{z} = \text{conv}(\boldsymbol{W}, \boldsymbol{x})$) and $\boldsymbol{h} = \text{ReLU}(\boldsymbol{z})$. Let also $\boldsymbol{x}' = \text{vec}(\boldsymbol{x})$ and $\boldsymbol{z}' = \text{vec}(\boldsymbol{x})$ denote the flattened versions of $\boldsymbol{x}$ and $\boldsymbol{z}$, respectively.

   (a) (5 points) What is the dimension of $\boldsymbol{z}$?

   Solution: We have $\boldsymbol{z} \in \mathbb{R}^{H' \times W'}$, with $H' = H - M + 1$ and $W' = W - N + 1$.

   (b) (10 points) Show 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 $\boldsymbol{z}' = \boldsymbol{M}\boldsymbol{x}'$. What is the general expression for element (i, j) of $\boldsymbol{M}$?

   Solution: We have that

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

   or, equivalently,

$$z_{u,v} = \sum_{i=u}^{M+u-1} \sum_{j=v}^{N+v-1} W_{i-u,j-v} x_{i,j}.$$

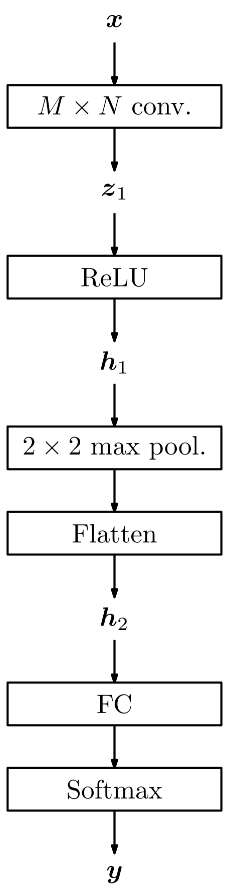


Figure 1: Architecture of a simple convolutional neural network.

Letting $b = u + v \times H'$ and $a = i + j \times H$, we can rewrite the above equality as

$$z'_b = \sum_{i=u}^{M+u-1} \sum_{j=v}^{N+v-1} W_{i-u,j-v} x'_a.$$

Following the result in (a), we have that $\boldsymbol{x}' \in \mathbb{R}^{HW}$ and $\boldsymbol{z}' \in \mathbb{R}^{H'W'}$. Then, there is a matrix $\boldsymbol{M} \in \mathbb{R}^{H'W' \times HW}$ such that $\boldsymbol{z}' = \boldsymbol{M}\boldsymbol{x}'$, with

$$M_{b,a} = \begin{cases} W_{i-u,j-v} & \text{for } 0 \leq i-u < M \text{ and } 0 \leq j-v < N, \\ 0 & \text{otherwise,} \end{cases}$$

and $i = \mathrm{mod}(a, H)$, $j = \mathrm{div}(a, H)$, $u = \mathrm{mod}(b, H')$, $v = \mathrm{div}(b, H')$.

(c) (10 points) Indicate the number of parameters in the network. Compare that number with the number of parameters in a network where the convolutional and max pooling layers are replaced by a fully connected layer yielding an output with the same dimension as $\boldsymbol{h}_2$. Ignore the bias terms.

Solution: We have:

- Convolutional layer has $M \times N$ parameters (ignoring the biases); its output has a dimension of $(H - M + 1) \times (W - N + 1)$;

- The max pooling layer has 0 parameters; its output has a dimension of $\lfloor \frac{H-M+1}{2} \rfloor \times$

$\lfloor \frac{W-N+1}{2} \rfloor$;

- Since we have 3 output units, the fully connected part has a total of $3 \times \lfloor \frac{H-M+1}{2} \rfloor \times \lfloor \frac{W-N+1}{2} \rfloor$ parameters.

The end result is

$$\text{N. params}_{conv} = MN + 3 \times \left\lfloor \frac{H-M+1}{2} \right\rfloor \times \left\lfloor \frac{W-N+1}{2} \right\rfloor.$$

Replacing the convolutional and max pooling layer by a fully connected layer, yielding a vector with the same dimension as $\boldsymbol{h}_2$, we would instead have a number of parameters in this first layer of $HW \times \lfloor \frac{H-M+1}{2} \rfloor \times \lfloor \frac{W-N+1}{2} \rfloor$. The total number would be

$$\text{N. params}_{FC} = (HW + 3) \times \left\lfloor \frac{H-M+1}{2} \right\rfloor \times \left\lfloor \frac{W-N+1}{2} \right\rfloor.$$

2. (5 points) Assume now that, instead of using convolutions, the same $H \times W$ image $\boldsymbol{x}$ is flattened into a sequence $\boldsymbol{x}' = \text{vec}(\boldsymbol{x})$ of length HW and goes through a single-head self-attention layer with 1×1 projection matrices $\boldsymbol{W}_Q = \boldsymbol{W}_K = \boldsymbol{W}_V = 1$, without any positional encodings. Write the expression for the attention probabilities and attention output as a function of $\boldsymbol{x}'$.

   Solution: The output is given by $\boldsymbol{z}' = \text{Softmax}(\boldsymbol{x}'\boldsymbol{W}_Q\boldsymbol{W}_K^\top(\boldsymbol{x}')^\top)\boldsymbol{x}'\boldsymbol{W}_V = \text{Softmax}(\boldsymbol{x}'(\boldsymbol{x}')^\top)\boldsymbol{x}'$.

Question 2 (35 points)

Image classification with CNNs. In this exercise, you will implement a convolutional neural network to perform classification using the Kuzushiji-MNIST dataset.

As previously done in Homework 1, you will need to download the Kuzushiji-MNIST dataset. You can do this by running the following command in the homework directory:

```
python download_kuzushiji_mnist.py
```

Python skeleton code is provided (`hw2-q2.py`). You will now try out convolutional networks. For this exercise, we recommend you use a deep learning framework with automatic differentiation (suggested: Pytorch).

1. (4 points) Why does a CNN have fewer free parameters than a fully-connected network with the same input size and the same number of classes (assuming it is a network for classification)?

   Solution: CNN takes advantage of parameter sharing to solve a classification problem with fewer parameters than a fully-connected network with the same input and the same number of classes. Furthermore, the sparse and local connectivity also contribute to fewer parameters where in a convolutional layer, each element only depends on a fraction of the elements from the previous layer; in a fully-connected layer depends on all.

2. (4 points) Despite having fewer free parameters, a CNN usually achieves better generalization on images and patterns representing letters and numbers than a fully-connected network. Can you justify this?

   Solution: Given the structured nature of images where neighbour pixels are not independent of each other, CNNs can learn equivariant representations for letters and numbers on an image, thus achieving better generalization than a fully-connected neural network. Additionally, having less parameters also reduces the chances of overfitting.

3. (4 points) If the input is from a source composed of independent sensors (with no spatial structure), is a CNN still expected to achieve better generalization than a fully connected network? Justify why.

   Solution: The fully connected neural network will be able to tackle better input from a source of independent sensors since there is no structure in the input; therefore, the parameter sharing and sparse connections of CNNs can be expected to lead to worse generalisability.

4. (20 points) Implement a simple convolutional network with the following structure:

   - A convolution layer with 8 output channels, a kernel of size 5x5, stride of 1, and padding chosen to preserve the original image size. Note: Assign convolution layer initialization to a variable called `self.conv1` to
   - A rectified linear unit activation function.
   - A max pooling with kernel size 2x2 and stride of 2.
   - A convolution layer with 16 output channels, a kernel of size 3x3, stride of 1, and padding of zero.
   - A rectified linear unit activation function.
   - A max pooling with kernel size 2x2 and stride of 2.
   - An affine transformation with 600 output features (to determine the number of input features use the number of channels, width and height of the output of the second block. Hint: The number of input features = $\textit{number of output channels} \times \textit{output width} \times \textit{output height}$).
   - A rectified linear unit activation function.
   - A dropout layer with a dropout probability of 0.3.
   - An affine transformation with 120 output features.
   - A rectified linear unit activation function.
   - An affine transformation with the number of classes followed by an output LogSoftmax layer.

   Hint: use the functions `nn.Conv2d` and `nn.MaxPool2d`.

   Train your model for 20 epochs using `Adam` tuning only the learning rate on your validation data, using the following values: 0.00001, 0.0005, 0.01. Report the learning rate of best configuration and plot two things: the training loss and the validation accuracy, both as a function of the epoch number.

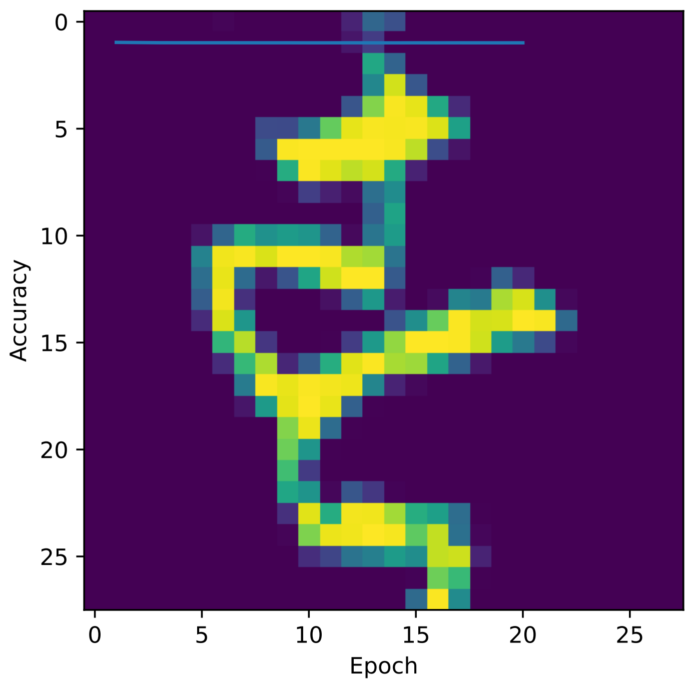
Solution: Training 20 epochs with a learning rate of 0.0005, a l2 weight decay of 0, a dropout probability of 0.3 and using the Adam optimizer we get a validation accuracy of 0.9839 and a test accuracy of 0.9538.

5. (3 points) Plot the activation maps of the first convolutional layer and the original training example. From the comparison of the activation maps and the original image, what appears to be highlighted in the activation maps?

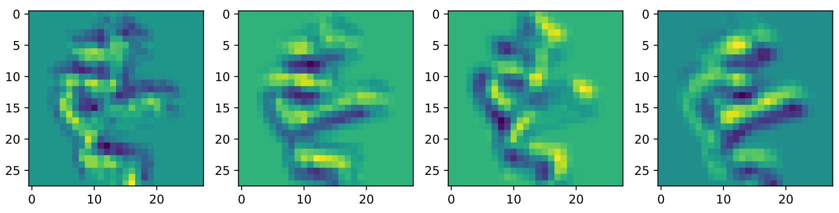
   NOTE: the skeleton code already performs the extraction of the activation maps (saved as activation_maps.pdf) and the original image (saved as original_example.pdf). No coding is required.

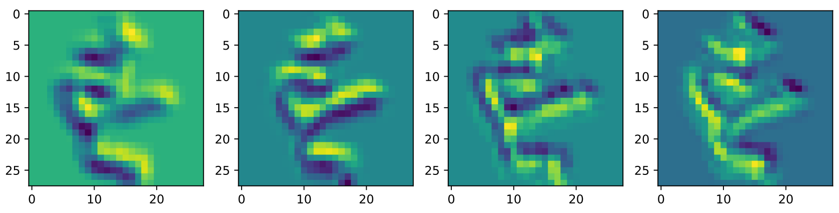
   Solution: Edges, lines and corners seem to be highlighted in the different activation maps.

   Original image:



   First convolution layer activation maps:





Question 3 (35 points)

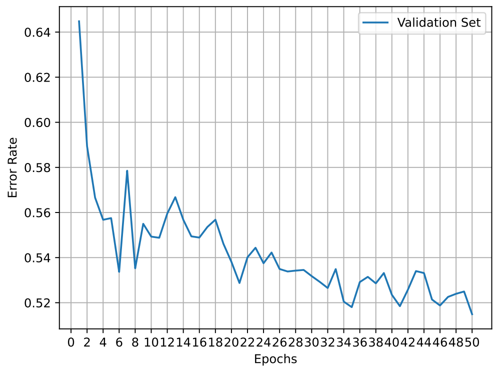
Character-level Machine Translation. Machine translation is the problem of automatically translating a sequence of tokens from a source language into a target language. This task is usually addressed with an encoder-decoder model. Typically, the encoder is based on LSTMs or self-attentions that process and summarises the source sentence into relevant feature representations. The representations are then passed to a target language decoder, commonly an LSTM model or a masked self-attention, which generates the respective translation of the source sentence token by token given the previous tokens and the encoder representations.

1. In the following, you will implement a character-level machine translation model from Spanish to English, that is, a model that translates a sequence of characters (a sentence in

Spanish) into another sequence of characters (a sentence in English). The evaluation metric is the mean error rate (which calculates the Levenshtein distance between the prediction and the true target, divided by the true length of the sequence).

(a) (10 points) The dataset is provided in the folder `data`. Implement a vanilla character-level machine translation model using an encoder-decoder architecture with an auto-regressive LSTM as the decoder and a Bidirectional LSTM in the encoder. Specifically, in the skeleton code, you will need to implement the method `forward()` of both the `Encoder` and the `Decoder` in `models.py` and then run (`python hw2-q3.py`). Train the model over 50 epochs, using a learning rate of 0.003, a dropout rate of 0.3, a hidden size of 128, and a batch size of 64. The dropout layer should only be applied to the embeddings and the final outputs (in both the encoder and the decoder).
Plot the validation error rate over epochs. Also, report the final error rate in the test set.
You can run the code with the command `python hw2-q3.py`.
Hint: At each time-step, the LSTM decoder receives as input the embedding of the current word (`self.embedding`), then each output at each timestep has dropout applied (`self.dropout`) and the resulting decoder state is used in the next timestep. In the end, all decoder outputs are concatenated to form an output sequence.

Solution:



Final test error rate: 0.5155

(b) (20 points) Add an attention mechanism to the decoder (bilinear attention, explained below), which weights the contribution of the different source characters, according to relevance for the current prediction.

Bilinear Attention. This attention mechanism is defined as follows:

$$\boldsymbol{p} = \mathrm{softmax}(\boldsymbol{s}), \tag{1}$$

where $\boldsymbol{s}_i$ is the score between the i-th source character and the target character at the current time step, and the softmax is applied in the dimension of the size of the source sequence. This is equivalent to applying `softmax` to the score vectors. The score is defined as:

$$s_i = \boldsymbol{z}^\top \boldsymbol{h}_i, \tag{2}$$

where $\boldsymbol{z} = \boldsymbol{W}_q^\top \boldsymbol{q}$ in which $\boldsymbol{q}$ is the hidden state of the decoder at the current time step (the query), $\boldsymbol{W}_q$ is a weight matrix (`linear_in` in the skeleton code), and $\boldsymbol{h}_i$ is the hidden state of the encoder at the i-th time step. After computing the attention weights $\boldsymbol{p}$, we compute the context vector $\boldsymbol{c}$ as:

$$\boldsymbol{c} = \sum_{i=1}^{T_x} p_i \boldsymbol{h}_i. \tag{3}$$

Finally, the attention layer output will be computed as:

$$\boldsymbol{a}_o = \tanh\left(\boldsymbol{W}_o^T [\boldsymbol{q}; \boldsymbol{c}]\right), \tag{4}$$

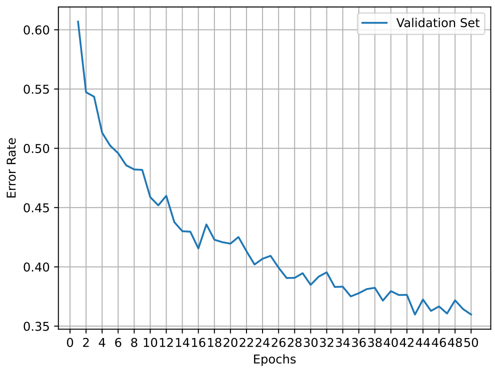
where $\boldsymbol{W}_o$ is a weight matrix (`linear_out` in the skeleton code) and $[\boldsymbol{h}_q; \boldsymbol{c}]$ is the concatenation of the query and the context vector. $\boldsymbol{a}_o$ is the attention layer output and will be used as the final representation at the current time step in the decoder.

Specifically, in the skeleton code, you will need to implement the `forward()` methods of the `Attention` class and slightly modify your `Decoder` implementation to account for the attention mechanism (it should modify the output of the LSTM at each timestep).

Plot the validation error rate over epochs. Also, report the final error rate in the test set.

You can run the code with the command `python hw2-q3.py -use_attn`. Use the same hyperparameters as in the previous exercise.

Solution:



Final test error rate: 0.3744

(c) (5 points) What could we do to improve results without changing the model architecture? Explain why this change would improve results. Hint: think about the decoding process in `test()` implemented on the `hw2-q3.py` file.

Solution:

Currently, the decoding process uses greedy decoding. We could use beam search to improve the results.

Beam search is able to select the best possible sequence given a beam size. This is done by keeping track of the top K sequences at each time step. The top K sequences are then used to calculate the next time step. This process is repeated until the end of the sequence is reached.

On the other hand, greedy decoding outputs the top-1 token at each time step. This means that the model is not able to explore different possibilities. Beam search is able to explore different possibilities and select the best one. This is why beam search is able to improve the results.