TÉCNICO LISBOA

Deep Learning (DEI)

Master in Computer Science and Engineering

Homework 2

Authors:

Luís Freire D'Andrade (94179) luis.vilhena@tecnico.ulisboa.pt
Rita Duarte (95531) ritareisduarte@tecnico.ulisboa.pt

Group 28

2022/2023 – 1º Semester, P2

Contents

Work Distribution

The questions 3.1.a), 3.1.b) and 3.1c) were assigned to Luís D'Andrade.
The questions 2.1, 2.2, 2.3, 2.4 and 2.5 were assigned to Rita Duarte.
Both the colleagues worked together to solve the 1.1a), 1.1b), 1.1c) and 1.2 questions.

References

- Theorical slides
- Practical notebooks
- https://learnopencv.com/number-of-parameters-and-tensor-sizes-in-convolutional-neural-network/
- https://towardsdatascience.com/understanding-parameter-sharing-or-weights-replication-within-convolutional-neural-networks-cc26db7b645a
- https://gist.github.com/HarshTrivedi/f4e7293e941b17d19058f6fb90ab0fec

1 Question 1

1.1 1.1a)

For the convolutional layer with images, $x \in \mathbb{R}^{H \times W}$, filter weights, $W \in \mathbb{R}^{M \times N}$, a stride of 1 and no padding. The result size (dimension) of the convolution will be:

$$((H{-}M + 2 \cdot 0)/1 + 1) \times ((W{-}N + 2 \cdot 0)/1 + 1) = (H{-}M + 1) \times (W{-}N + 1)$$

Since $z = conv(W, x)$, the result of the convolution, $z \in \mathbb{R}^{(H-M+1) \times (W-N+1)}$.

1.2 1.1b)

To easily distinguish all variables, we'll be denoting the filter weights with $\boldsymbol{w}$.

For the result of the convolution, $z = conv(w, x)$, the general expression for its element (i, j) can be written as:

$$z_{i,j} = \sum_{k=1}^{M} \sum_{l=1}^{N} w_{k,l}\ x_{i+k,j+l} \tag{1}$$

Where $z_{i,j}$ is the output feature map at position (i, j) and the sum is taken over all k, l values where the kernel is not outside the input signal.

Given that $z' = vec(z)$ denotes the flattened version of z, the following relationships (for the equivalent entries between a matrix and its flattened version) can be written:

$$z_{i,j} = {z'}_{iW'+j} \quad {z'}_m = z_{\lfloor \frac{m}{W'} \rfloor, m \bmod W'}$$

Where $i \in \{1, \ldots, H'\}$, $j \in \{1, \ldots, W'\}$ and $m \in \{1, \ldots, H'W'\}$

Applying equation 1 to the second relationship:

$${z'}_m = \sum_{k=1}^{M} \sum_{l=1}^{N} w_{k,l}\ x_{(\lfloor \frac{m}{W'} \rfloor)+k, (m \bmod W')+l} \tag{2}$$

Also considering $x' = vec(x)$ denotes the flattened version of x, the following relationships (for the equivalent entries between a matrix and its flattened version) can be written:

$$x_{i,j} = {x'}_{iW+j} \quad {x'}_m = x_{\lfloor \frac{m}{W} \rfloor, m \bmod W}$$

Where $i \in \{1, \ldots, H\}$, $j \in \{1, \ldots, W\}$ and $m \in \{1, \ldots, HW\}$

Applying the first relationship to equation 2:

$${z'}_m = \sum_{k=1}^{M} \sum_{l=1}^{N} w_{k,l}\ {x'}_{((\lfloor \frac{m}{W'} \rfloor)+k)W+((m \bmod W')+l)} \tag{3}$$

Finally, we can define $M \in \mathbb{R}^{H'W' \times HW}$, $z' = Mx'$, as:

$$M_{i,j} = \begin{cases} w_{k,l} & \text{if } \exists k \in \{1,\dots,M\} \ \ \& \ \ \exists l \in \{1,\dots,N\} : j = ((\lfloor \frac{i}{W'} \rfloor) + k)W + ((i \bmod W') + l) \\ 0 & \text{otherwise} \end{cases}$$

Where $i \in \{1,\dots,H'W'\}$ and $j \in \{1,\dots,HW\}$.

1.3 1.1c)

Given a $network_1$ where the first convolutional layer has kernel of size M×N:

- in the convolutional layer there are M×N weights parameters (or weights);

- in the max pooling layer there are no more parameters thanks to the realization of a fixed computation on the convolution matrix;

current number of parameters of $network_1$ **= MN**;

- in the output layer, since there are 3 classes there are going to be the number of classes times the number of elements of the flattened h_2, therefore 3 × HW parameters;

- $weight = \frac{z.width}{stride} = \frac{W-N+1}{2}$

- $height = \frac{z.height}{stride} = \frac{H-M+1}{2}$

final total parameters of $network_1 = MN + 3\frac{W-N+1}{2}\frac{H-M+1}{2}$.

Given a $network_2$ where the convolutional and max pooling layers are replaced by a fully connected layer yielding an output with the same dimension as h_2:

- connecting the first layer to the h_2 we would have weight matrix that takes the vectorization of x and output layer $h_2width + h_2height$;

current number of parameters of $network_2 = \frac{W-N+1}{2}\frac{H-M+1}{2} \times HW$;

- in the output layer, since there are 3 classes there are going to be the number of classes times the number of elements of the flattened h_2, therefore $3 \times \frac{W-N+1}{2}\frac{H-M+1}{2}$ parameters;

final total parameters in the $network_2 = \frac{W-N+1}{2}\frac{H-M+1}{2} \times HW + 3 \times \frac{W-N+1}{2}\frac{H-M+1}{2}$.

Considering:

- $\frac{W-N+1}{2}\frac{H-M+1}{2} = H'W'$

Comparing the number of parameters for each network, knowing the filter MN is smaller than the image to be classified H'W' ($M < N$ and $N < W$):

$MN + 3 \times H'W' < HW \times H'W' + 3 \times H'W'$, therefore

number of parameters $network_1$ < **number of parameters** $network_2$

1.4 1.2)

Attention probabilities can be calculated using dot product similarity between the query (Q), key (K) and value (V) matrices. Hence:

- query = $Q = W_Q \cdot X$

- key = $K = W_K \cdot X$

- value = $V = W_V \cdot X$

Since, $W_Q = W_K = W_V = 1$ they are all the same in this scenario. Being X the vector x' that goes through a single-head self-attention layer:

- query = $Q = W_Q \cdot X = 1 \cdot x' = x'$

- key = $K = W_K \cdot X = 1 \cdot x' = x'$

- value = $V = W_V \cdot X = 1 \cdot x' = x'$

The attention probabilities can be calculated by $p = softmax(\frac{QK^T}{\sqrt{d}})$. Therefore:

- $p = softmax(QK^T)$, for $\sqrt{d} = 1$

From previous assumptions, $Q = x'$ and $K = x'$ so the attention probability is written as:

$$p = softmax(x' \cdot x'^T)$$

Finally in order to calculate the attention output given by $output = V \cdot p$ and $V = x'$:

$$output = x' \cdot softmax(x' \cdot x'^T)$$

2 Question 2

2.1 2.1)

A convolutional neural network has fewer free parameters than a fully-connected network with the same input size and number of classes thanks to techniques such as tying parameters and pooling.

In a fully-connected network, each neuron in a particular layer is connected to every neuron in the previous layer, and each of these connections has its own weight. This means that the number of weights/parameters in the network is equal to the product of the number of neurons in the previous layer and the number of neurons in the current layer.

In contrast, a CNN uses convolutional layers, which involve applying a set of filters to the input, where each filter corresponds to a set of weights.

Tying parameters refers to sharing the same set of parameters across different layers or different parts of the network (such as weight tying, transformer-based models...). On another hand, pooling performs down-sampling operation over the feature maps, it reduces the spatial dimensions of the input, while also increasing the model's ability to recognize features across a larger area of the input.

Therefore, because of the use of parameter sharing and pooling layers, CNNs have fewer free parameters than fully-connected networks with the same input size and number of classes.

2.2 2.2)

Yes, a CNN typically achieves better generalization on images and patterns representing letters and numbers than a fully-connected network because it is designed to take advantage of the spatial structure of the data.

For example, regarding images a fully-connected network would treat each pixel independently, without considering the relationship between the pixels. Then features would likely be spread out across many neurons in the network which would difficult the learning process for the model.

A CNN uses convolutional layers, so it automatically learns spatial hierarchies of features. The filters in these layers detect local patterns shared across images. These features passed through the layers become more complex features as they are learned.

Also, the pooling layers are used to down-sample the feature maps, which helps to reduce the amount of spatial information that needs to be processed by the network. This helps the network to learn different variations of the image.

Therefore, the use of convolutional and pooling layers leads to overall better achievement. The model's efficiency is better and it is less likely to overfitting.

2.3 2.3)

As explained before, CNNs are capable of taking advantage of the spatial structure of the data. The convolutional layers are designed to learn features from local patterns in the input. If the input is composed of independent sensors these patterns don't exist which disables the convolutional layers of learning useful features.

Therefore, a fully-connected network would likely be a better choice, because each neuron is connected to every neuron in the previous layer, allowing the network to learn any kind of relationship between the inputs without any previous assumptions.

In summary, if the input is from a source composed of independent sensors that have no spatial structure, a CNN may not be the best choice for achieving good generalization, but as always depends on the characteristics of the data and the problem. A fully-connected network is more general and can handle any kind of input without making assumptions about the structure of the data, even though more likely to overfitting.

2.4 2.4)

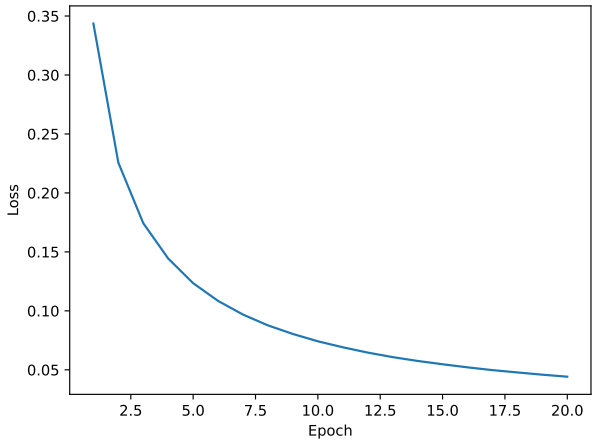


Figure 1: Plotting the training loss over 20 epochs, using a learning rate of 0.0005, a dropout rate of 0.3 and an optimizer adam.

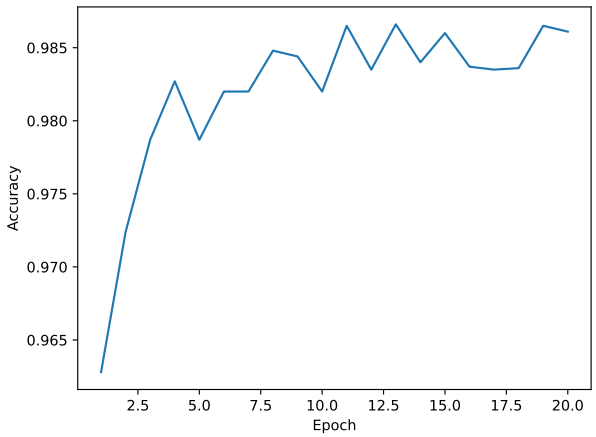


Figure 2: Plotting the validation accuracy over 20 epochs, using a learning rate of 0.0005, a dropout rate of 0.3 and an optimizer adam.

Learning rate of best configuration: 0.0005
Final test accuracy: 0.9564

2.5 2.5)

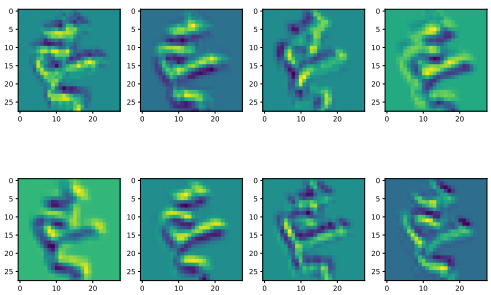
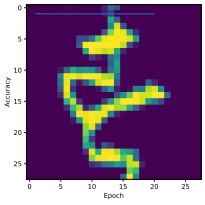


Figure 3: Activation maps of the first convolutional layer for the computed CNN using a learning rate of 0.0005.

The highlighted pixels form the original character:



3 Question 3

3.1 3.1a)

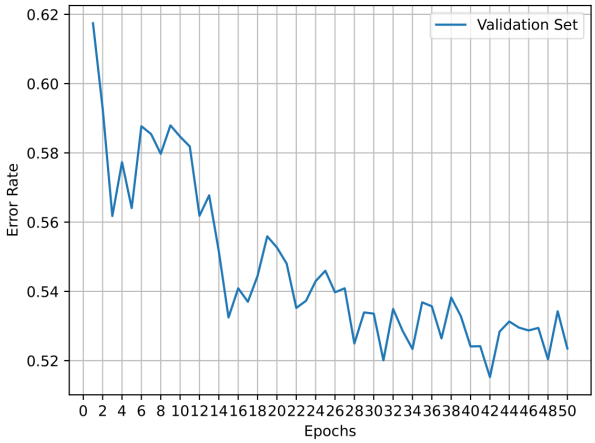


Figure 4: Plotting the validation error rate over 50 epochs, using a learning rate of 0.003, a dropout rate of 0.3, a hidden size of 128, and a batch size of 64.

Final error rate: 0.5288

3.2 3.1b)

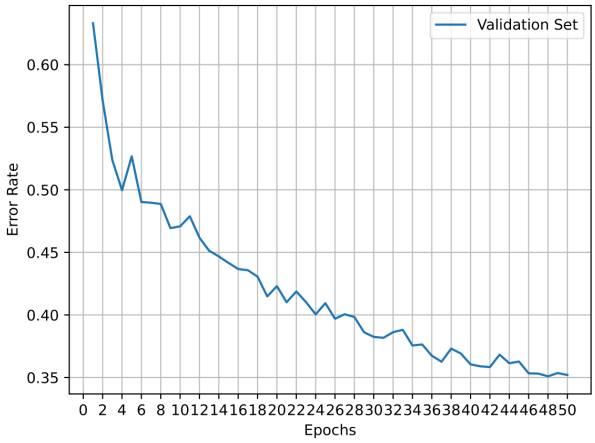


Figure 5: Plotting the validation error rate (with an attention mechanism in the decoder) over 50 epochs, using a learning rate of 0.003, a dropout rate of 0.3, a hidden size of 128, and a batch size of 64.

Final error rate: 0.3681

3.3 3.1c)

One way to improve results without changing the model architecture is to employ a more complex decoding algorithm, including beam search or sampling-based decoding techniques. These techniques examine multiple word sequences rather than simply the most likely one at each stage, which can lead to more varied and coherent translations while also decreasing the likelihood of making mistakes, repeating common phrases, or becoming stuck in loops.

Better results can also be achieved by incorporating strategies like context-based decoding (which enables the model to produce more pertinent and appropriate responses based on the provided context) or adding a language model as a post-processing step (which enables the model to produce more fluid and grammatically accurate translations by taking into account the language structure and rules).

Using a larger and more varied training dataset can also help the model generalize better and correct any faults or inaccuracies that may have resulted from its lack of exposure to certain phrases or idiomatic expressions. Additionally, fine-tuning the model on a task-specific dataset can also lead to better results. This is because it enables the model to adjust to the unique traits and variances of the target language and domain, producing translations that are more accurate.