Instituto Superior Técnico

Deep Learning

2^{nd} Quarter - 2022/2023

Homework 2

Group 46:
Santiago Quintas - 93179
Alina Klingel - 105208

Professor:
Mário Figueiredo

15^{st} January, 2023

Declaration of Contribution

We worked on all of the Questions together. In Question 3 we did together in a Discord Call, the other Questions we worked in separate and compared the results. In Question 2 Santiago was leading the process of development, while Alina did so in Question 1.

Question 1

Annexed file after Q3.

Question 2

1.

A CNN will have fewer parameters than a fully-connected network with the same input size and the same number of classes, because the output neurons of a CNN will only take into account certain input neurons, while the output neurons of a fully-connected network take into consideration every single input neuron in the layer before. This means that instead of having a set of n weights for every output neuron, corresponding to n input neurons, the CNN will have for every output neuron a number of weights lower than n. This results in a reduction of free parameters. Below there is a visual representation of the density of weights in both networks, [1].

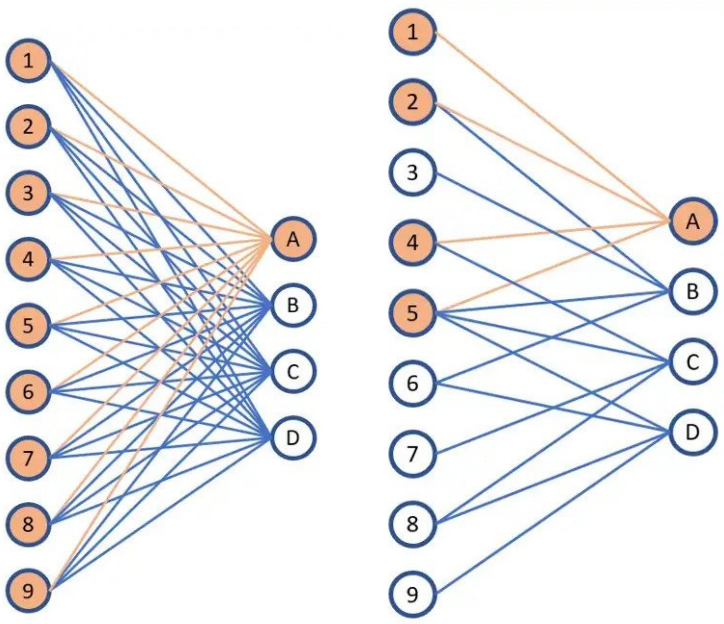


Figure 1: Visual representation of the weight density in a fully connected network (left) and in a CNN (right)

On top of this, a CNN can also share the same weights for different input neurons, so for the example above (Fig 1.), input image of 4x4 pixels, and a filter of 2x2, without weight sharing we use 16 weights, if we apply weight sharing that number is reduced to 4 weights, because the same 4 weights are reused in every iteration of the filter.

2.

CNN's make use of the spatial locality of the input pixels, allowing the network to learn patterns, this makes them excel at analysing images and patterns representing letters and numbers, achieving better results than a fully-connected network. While a fully-connected network runs into the problem of overfitting, a CNN does not because of the reduced amount of free parameters, the CNN is also able to achieve better efficiency for big data inputs, such as images, due to the significant decrease in weights used to calculate the output of the neurons. On the opposite side, fully-connected networks lose efficiency when the input data increases by a great amount, since the number of weights that need updating become too big for the network to be efficient.

3.

The key benefit of a CNN is the usage of spatial structures to increase the efficiency of learning, when that is taken away, the CNN loses its advantage and the better generalization goes away with it.

In a CNN the spatial structure of an image is kept by making the shape of the pixel vector turn into a 2D array, for example a vector holding 784 pixels becomes a 28x28 array, this is what allows the CNN to detect specific patterns, the filters applied to the image take into account the spatial location of the pixels, and they would lose this ability if they were flattened into a 1 dimension array, without it the CNN wouldn't be able to detect specific patterns in the image and would be the equivalent to a fully-connected network.

4.

The default parameters used in the implementation of the CNN were the following:

Number of Epochs	20
Learning Rate	0.01
Batch Size	8
Optimizer	adam

Table 1: Default parameters 1

For different learning rates we achieved different accuracies, shown in the table 2.

The best configuration, according to the accuracy test is with a learning rate of 0.0005

Learning Rate	Final test accuracy
0.01	0.7209
0.0005	0.9581
0.00001	0.8896

Table 2: Accuracy test for different learning rates

Below is the plot of both the training loss and accuracy over the number of epochs of the changes for a learning rate of 0.001, which corresponds to our best configuration.

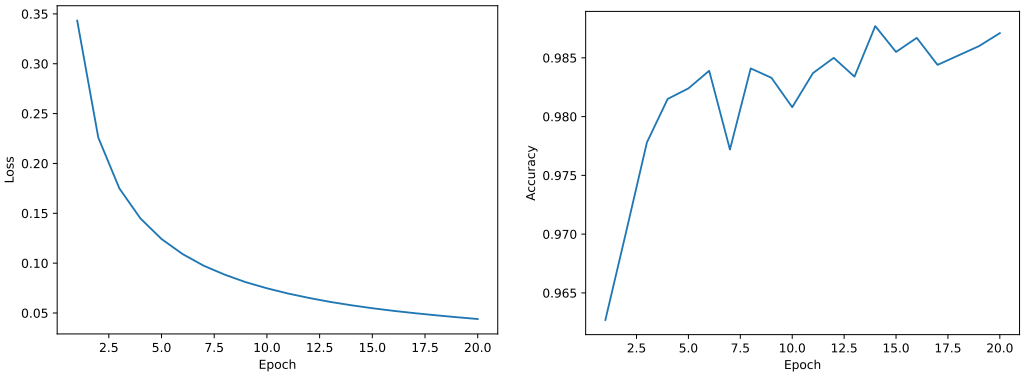


Figure 2: Training loss and accuracy of the CNN for a learning rate of 0.0005

5.

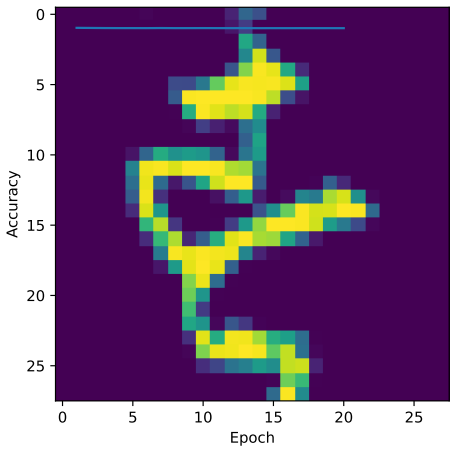


Figure 3: Original training example

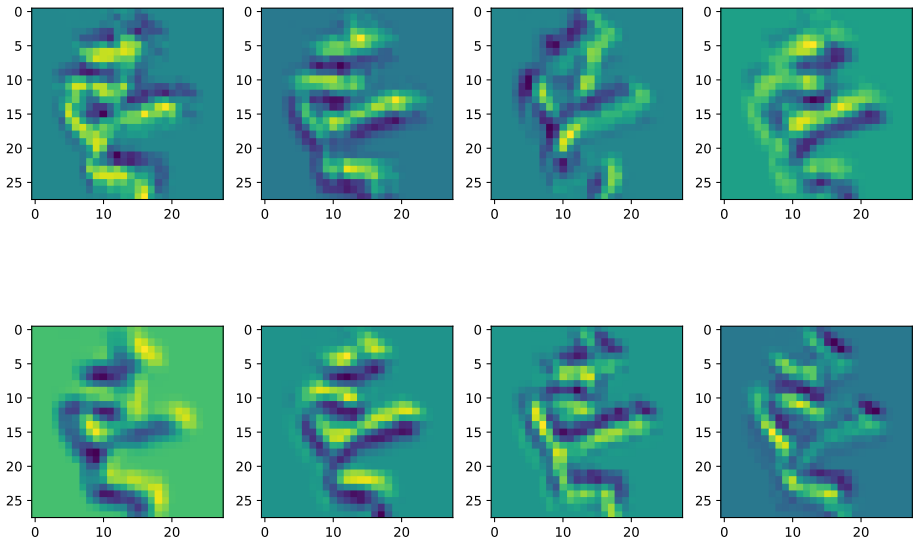


Figure 4: Activation maps of the first convolutional layer for a learning rate of 0.0005

In the activation maps, the highlighted parts correspond to the edges of the original figure, in some maps the bottom horizontal lines are highlighted, in others it's the top horizontal lines, also the vertical lines from both the left and the right of the training example.

Question 3

1.

a)

The default parameters used in the implementation of machine translation model were the following:

Number of Epochs	50
Learning Rate	0.003
Hidden Size	128
Dropout	0.3
Batch Size	64

Table 3: Default parameters 2

Below is the plot of the validation error over the number of epochs.

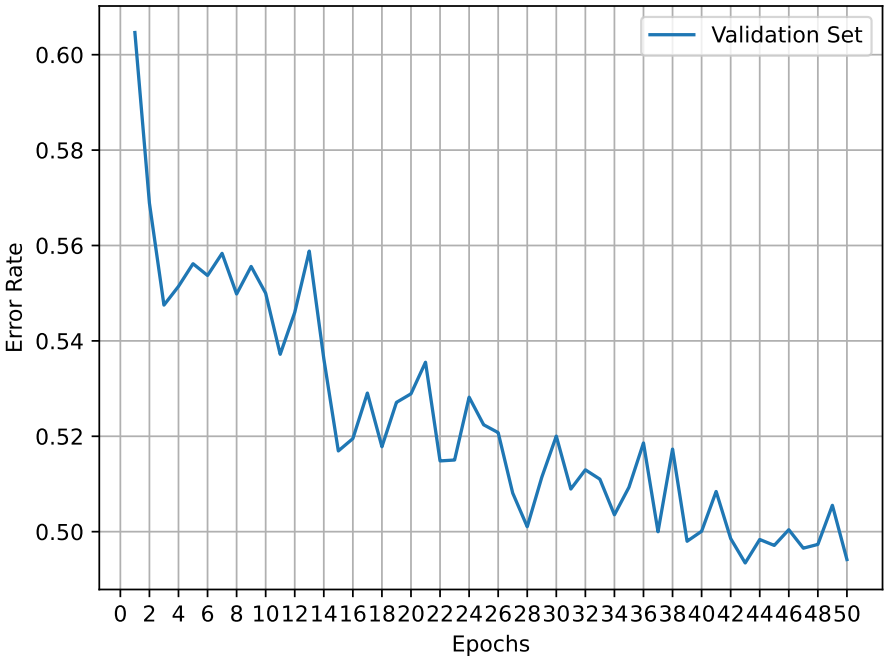


Figure 5: Validation error of the machine translation model as a function of the epoch number

For this configuration we obtained a final error rate of **0.5016**

b)

Below is the plot of the validation error over the number of epochs, with the added attention mechanism and using the same default parameters.

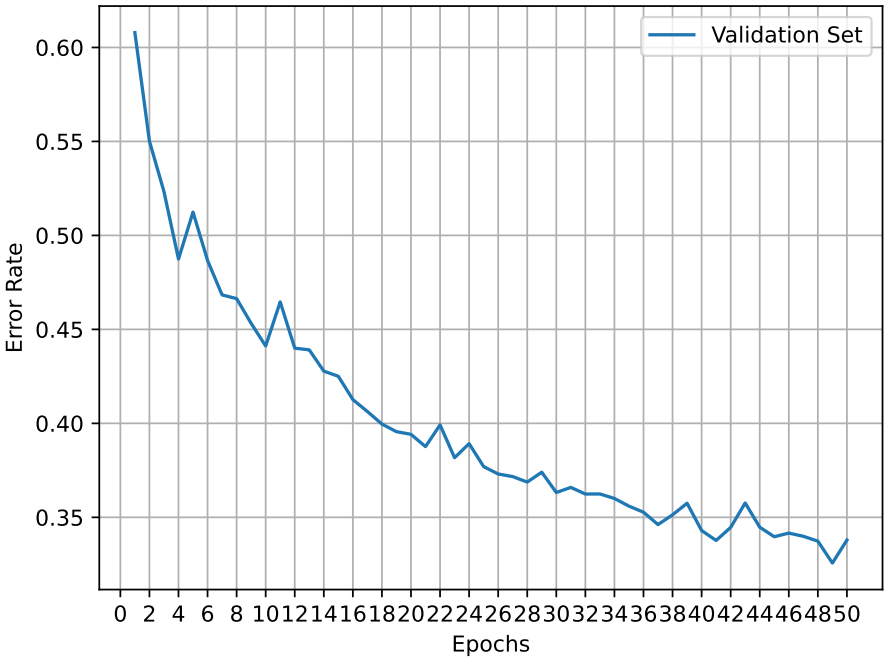


Figure 6: Validation error of the machine translation model, with the added attention mechanism, as a function of the epoch number

For this configuration we obtained a final error rate of **0.3421**

c)

To improve the results, without changing the model architecture, we can use a beam search algorithm in the decoding process as an alternative to the greedy search we are currently using. Instead of keeping a single word, the most probable one, at each time step, the beam algorithm keeps multiple high probability words, this allows for a more flexible decoding process and overall better results.

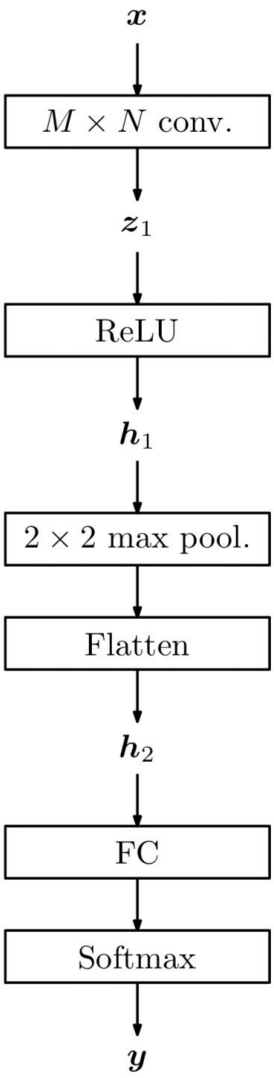
Bibliography

[1] Diego Unzueta, 2021. Deep Learning Fundamentals. Place of publication: Towards Data Science. Available from: https://towardsdatascience.com/convolutional-layers-vs-fully-connected-layers-364f05ab460b.

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.



Given: • image $\vec{x} \in \mathbb{R}^{H \times W}$ in three possible classes

• 1 conv. layer: • 1 $M \times N$ filter • stride $S = 1$ • padding $D = 0$ (no pad) • ReLU = non linear

• $\vec{W}$ = filter weights • $\vec{z}$ = result of convol. ($\vec{z} = \text{conv}(\vec{W}, \vec{x})$) • $\vec{h} = \text{ReLU}(\vec{z})$

• 1 max pooling layer: • 2×2 • stride $S = 2$ • padding $D = 0$ (no pad)

Flattened versions of $\vec{x}$, $\vec{z}$: $\vec{x}' = \text{vec}(\vec{x})$, $\vec{z}' = \text{vec}(\vec{x})$

• 1 output layer: softmax activation

a) Dimension of $\vec{z}$ (= result of convolution)

Idea: Get output (width/height) of the convolutional layer like in practical 8

$$\text{Output width} = \frac{\text{input width} - \text{kernal width} + 2 \times \text{padding width}}{\text{stride}} + 1$$

• input width: W
• kernal width: N (filter width)
• padding: 0
• stride: 1

$$\Rightarrow \text{output width}_{\text{after convolution}} = \frac{W - N + 0}{1} + 1 = W - N + 1$$

- input height: H
- kernel height: M
- padding : 0
- stride : 1

$\Rightarrow$ output height after convolution $= \frac{H - M + 0}{1} + 1$

$= H - M + 1$

$\Rightarrow$ Dimensions of $\vec{z}$ are $(W - N + 1) \times (H - M + 1)$

(b) (10 points) 1) 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 $\boldsymbol{z}' = \boldsymbol{M}\boldsymbol{x}'$. 2) What is the general expression for element (i, j) of $\boldsymbol{M}$?

- matrix $\vec{M} \in \mathbb{R}^{H'W' \times HW} = \mathbb{R}^{(H-M+1)(W-N+1) \times HW}$
  - $(H-M+1)(W-N+1)$: dimensions of output of convolutional layer
  - HW: dimension of original input $\vec{x}$

a = Number of rows, b = Number of columns, $a \times b = 5 \times 1$ → rows × columns!

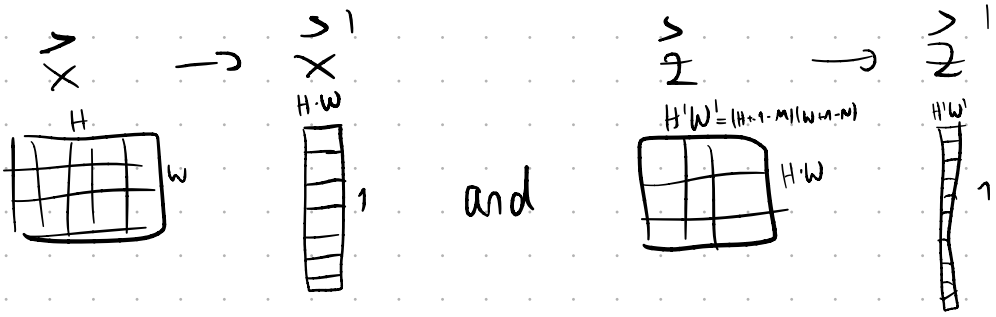
Matrix Multiplication: $n \times p \cdot p \times m = n \times m$

- vectors: $\vec{z}'$ and $\vec{x}'$ = Flattened versions of $\vec{x}$ and $\vec{z}$ (both $\text{vec}(\vec{x})$)
  - $\vec{x}$: input $\mathbb{R}^{H \times W}$
  - $\vec{z}$: result of convolution $\mathbb{R}^{(H-M+1) \times (W-N+1)}$
- want to show: $\vec{z}' = \vec{M}\vec{x}'$ + general expression of $\vec{M}$

↳ so after convolution:
- need to Relu + maxpool to get to Flattened version of $\vec{z}$?
- or can I just flatten $\vec{x}$ and $\vec{z}$ to show $\vec{z}' = M\vec{x}'$

- Try to just flatten:

Flatten: $[[0,1],[2,3]] \to [0,1,2,3]$



1) Look at dimensions:

$\vec{x}'$ (rows × columns): $H \cdot W \times 1$

$\vec{z}'$: $H' \cdot W' \times 1$

⇒ with $\vec{z}' = \vec{M} \cdot \vec{x}$ for the product to be possible $\vec{M}$ needs the dimension of $H'W' \times HW$ $\left[H'W' \times 1 = (H'W' \times HW) \cdot (H \cdot W \times 1)\right]$

⇒ so we showed, that the Matrix to be multiplied with $\vec{x}'$ needs the dimensions $H'W' \times HW$ in order to get the vector of $\vec{z}$ with dimension $H'W' \times 1$

2) Find General expression for a element (i,j) of $\vec{M}$

2.1 Look at a small example: What is happening?

- Convolution $\vec{z}$ for a 2d input $\vec{x}$ and 2D kernel $\vec{k}$ (of filter!) (with stride 1)

$\Rightarrow$ Sliding window k over X to get matrix $\vec{z}$

$\vec{x}$: $H=3$, $W=3$

$$\begin{pmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{pmatrix} \odot \vec{k}: \begin{pmatrix} k_1 & k_2 \\ k_3 & k_4 \end{pmatrix} \quad M=2,\ N=2$$

$W' = W-N+1 = 3-2+1 = 2$ $\xrightarrow{\text{convolution}}$ $H' = H-M+1 = 3-2+1 = 2$

$$\vec{z} = \begin{pmatrix} k_1 \cdot x_1 + k_2 \cdot x_2 + k_3 x_4 + k_4 \cdot x_5 & k_1 \cdot x_2 + k_2 x_3 + k_3 \cdot x_5 + k_4 \cdot x_6 \\ k_1 x_4 + k_2 x_5 + k_3 x_7 + k_4 x_8 & k_1 x_5 + k_2 x_6 + k_3 x_8 + k_4 x_9 \end{pmatrix}$$

- It is possible to construct a Matrix $\vec{M}$ that leads as product with the flattend $\vec{x}$: $\vec{x}'$ to a flattend $\vec{z}$: $\vec{z}'$

$\vec{M}$ ($H' \cdot W'$ × $H \cdot W$):

$$\begin{pmatrix} k_1 & k_2 & 0 & k_3 & k_4 & 0 & 0 & 0 & 0 \\ 0 & k_1 & k_2 & 0 & k_3 & k_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & k_1 & k_2 & 0 & k_3 & k_4 & 0 \\ 0 & 0 & 0 & 0 & k_1 & k_2 & 0 & k_3 & k_4 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{pmatrix} = \begin{pmatrix} k_1 \cdot x_1 + k_2 \cdot x_2 + k_3 x_4 + k_4 \cdot x_5 \\ k_1 \cdot x_2 + k_2 x_3 + k_3 \cdot x_5 + k_4 \cdot x_6 \\ k_1 x_4 + k_2 x_5 + k_3 x_7 + k_4 x_8 \\ k_1 x_5 + k_2 x_6 + k_3 x_8 + k_4 x_9 \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{pmatrix}$$

(Annotations: $\vec{x}'$ is 9 × 1; $\vec{z}'$ is 4 × 1, $H' \cdot W'$)

- amount of 0s here is everything to the right of filter in matrix → so we have $(W-N)$ 0s
- just 1 zero cause stride = 1
- one row down → again all the xs in the last row of filter that are not covered: $(W-N)$ 0s
- amount of 0s equals elements in all rows under filter: rows under filter: $H-M$, elements in rows: W → $(H-M) \cdot W$ 0s

↳ so this gives same result as above where we slided over $\vec{x}$ with $\vec{k}$

we shifted one down → so one full row of 0s: W 0s

2.2 Now try to generalize this (for stride 1!)

Input: $H \times W$ - Matrix Filter/Kernel: $M \times N$ - Matrix convoluted Matrix: $H' \times W'$ - Matrix

$$\begin{pmatrix} x_{11} & \cdots & x_{1W} \\ \vdots & & \vdots \\ x_{H1} & \cdots & x_{HW} \end{pmatrix} \odot \begin{pmatrix} k_{11} & \cdots & k_{1N} \\ \vdots & & \vdots \\ k_{M1} & \cdots & k_{MN} \end{pmatrix} \longrightarrow$$

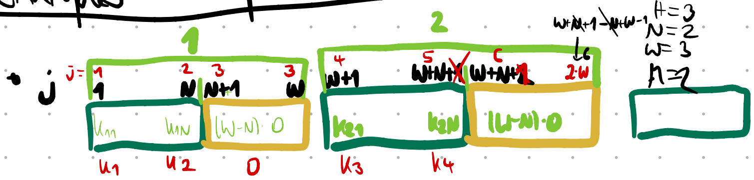
$\vec{x} \in \mathbb{R}^{H \times W}$ $\vec{k} \in \mathbb{R}^{M \times N}$

meeh → lets just do $\vec{M}$ like this (no brain dies)

$\vec{z} \in H' \times W'$

↳ now as $\vec{M} \cdot \vec{x}' = \vec{z}'$

Examples



- one row of G with elements g_j with $1 \le j \le M \cdot W$

$$g_j = \begin{cases} k_{ab}, & \text{if } b \le N \\ 0 & \text{otherwise} \end{cases}$$

row: $a := ((j-1) \text{ div } W) + 1$ (shift cause our index starts at 1)

column: $b := ((j-1) \bmod W) + 1$ (shift cause our index starts at 1)

g_1: $a = ((1-1) \text{ div } H) + 1 = 1$, $b = (0 \bmod W) + 1 = 1$, $1 \le 2 \to k_{11}$ ✓

g_2: $a = (2-1) \text{ div } 3 + 1 = 1$, $b = (2-1) \bmod 3 + 1 = 2$, $2 \le 2 \to k_{12}$

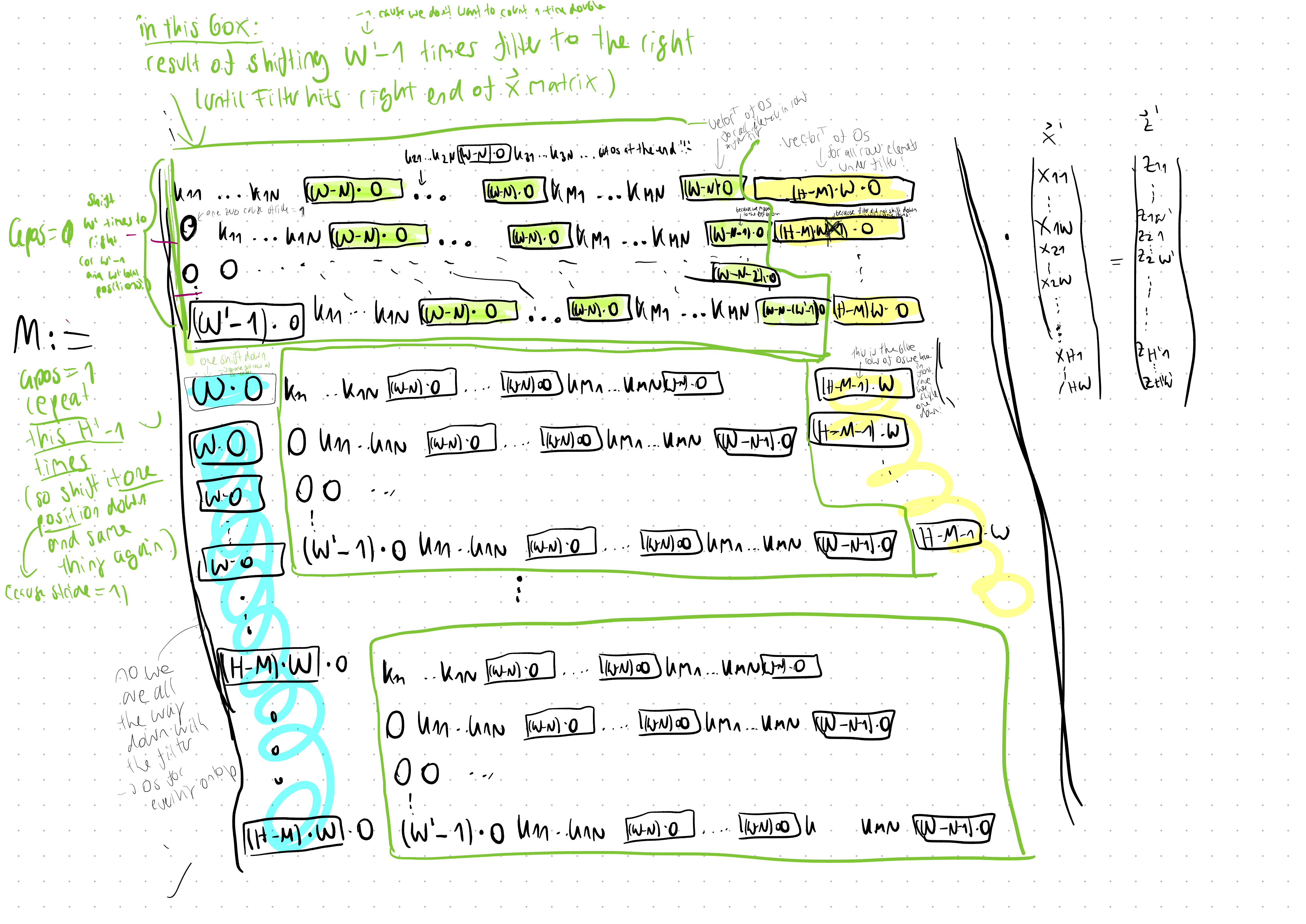
g_3: $a = (3-1) \text{ div } 3 + 1 = 1$, $b = (3-1) \bmod 3 + 1 = 4$, $4 \le 2$ ↯ $\Rightarrow g_3 = 0$ ✓

- shift $:= i \bmod H'$ $H' = H-M+1 = 3-2+1 = 2$, $W' = W-N+1 = 2$ i = rows

$i=1$: shift $= 1 \bmod 2 = 1$
$i=2$: shift $= 2 \bmod 2 = 0$
$i=3$: shift $= 3 \bmod 2 = 3$
$i=4$: shift $= 4 \bmod 2 = 4$

M_{11}: $i=1$, $j=1$ $j=1 \ge$ shift $\cdot W = 1 \cdot 3$ → no
$j=1 < 2 \cdot 3 + 1 \cdot 3 = 9$ → no
M_{13}: $i=1$, $j=3$ $j=3 \ge 3$ ✓ → 0 ✓

g_e $g_e = 1 - 3$



in this box:
result of shifting W'−1 times filter to the right
(until filter hits right end of x⃗ matrix)
cause we don't want to count 1 time double
Gaps = 0
Shift W' times to right
M :=
Gaps = 1
repeat this H'−1 times
(so shift it one position down and same thing again)
(cause stride = 1)
one shift down
vector of 0s
vector of 0s for all row elements under filter!
0s at the end!!
(W−N)·0
(H−M)·W·0
(W'−1)·0
W·0
(H−M)·W·0
(H−M−1)·W
this is the blue row of 0s we have in front
so we all the way down with the filter
x⃗'
z⃗'
=

rows columns

$$M_{ij} = \begin{cases} k_{ab} & \text{if } i= \; j= \text{ with } a= \; b= \; ? \\ 0 & \text{if } ? \quad ? \end{cases}$$

- **Box with just k_{ij}'s and 0s:**

$:=$ matrix G with elements g_{ij} (row, column)

a mod b = Rest($\frac{a}{b}$)

1 mod 3

1:3 = 0 Rest 1

- Dimensions of matrix G:

(Filter $K = $ ... N)

number of rows: $W' = W - N + 1$

$\rightarrow 1 < i < W'$

number of colums: $M \cdot N + M \cdot (W-N) = M(N+W-N) = M \cdot W$

($M \cdot N$: 1 sequence of ks with length N for each of the M rows; $M \cdot (W-N)$: 1 sequence of 0s with length (W-N) for each of the M rows)

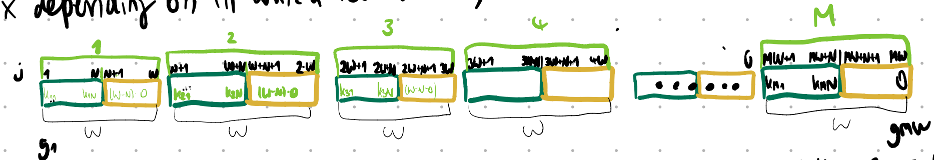
$\rightarrow 1 < j < M \cdot W$

W' — $M \cdot W$ — $W' \times M \cdot W$

- Each row:

W: $[k_{11} \ldots k_{1N}]$ $[(W-N) \cdot 0]$ $[k_{21} \ldots k_{2N}]$ $[(W-N) \cdot 0]$ ... $[k_{M1} \ldots k_{MN}]$ $[(W-N) \cdot 0]$

* sequence of (+ 0s to left and right of box depending on in which row we are)
* k-sequences: always N long
  0-sequence: always (W−N) long

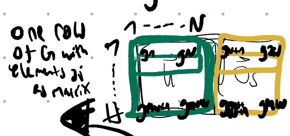


* one row of G with elements g_{1j} with $1 \le j \le M \cdot W$
  the elements g_{1j} of this row can be defined as:

$$g_{1j} = \begin{cases} k_{ab}, & \text{if } b \le N \\ 0 & \text{otherwise} \end{cases}$$

row $a := ((j-1) \text{ div } W) + 1$ (shift cause our index starts at 1)

column $b := ((j-1) \bmod W) + 1$ (shift cause our index starts at 1)



one row of G with elements g_{1j} as matrix

→ use this concept to index one row of our given matrix G g_{1j} → imagine the sequence elements of the row under a matrix like this

- **Now figure out the 0s sourrounding**

* Behind every row of the first there are $(H-M) \cdot W$ 0s
* For the second before each row there are $1W$ 0s and behind $(H-M-1)W$ 0s
* For the third before each row there are $2W$ 0s and behind $(H-M-2)W$ 0s

...

How to sequencise matracies

Problem: index starts at 0 → need to shift to make it

index of: row: column:

index of k: index div height, index mod width

a_{00} ... a_{0W} ; a_{H0} ... a_{HW} — putting matrix into sequence → k_1 .... k_{HW-1} (a_{00} ...... a_{HW-1})

height = H
width = W

* Finally for the last ▯ in M before each row there are $|H-M| \cdot W$ 0s and behind are none

- In each ▭ are W' positions/shifts
  - every row of ▯ and therefore every row of M corresponds to one unique position of the filter kernel
  - define each unique position as g-pos

$$g\text{-}pos := i \bmod H' \qquad i = 0, \dots, H'$$

$$\Rightarrow M(i,j) = \begin{cases} 0 & \text{if } j \leq \underbrace{g\text{-}pos \cdot W}_{\text{0-block to the left}} \\ & \text{or } j > \underbrace{M \cdot W}_{\text{size of 1 block}} + \underbrace{g\text{-}pos \cdot W}_{\text{size of the 0-block}} \\ g_\gamma & \text{otherwise} \end{cases}$$

0-block to the right

with $\gamma := j - g\text{-}pos \cdot W$

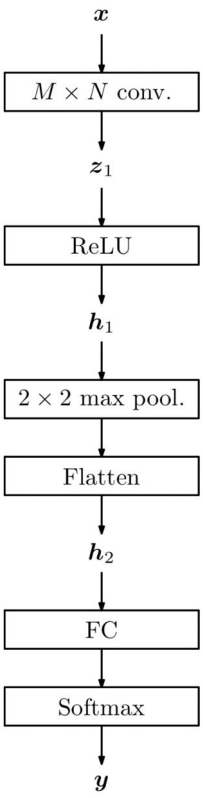
shift it to the very left to be able to use g by just taking γ as the column index of the kernel!

(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.

Probably similar to practical 8, Question 2.3 and 2.4

→ Do they want all parameters or just the trainable ones? I will just do the trainable ones

1 CNN-Network



Given: · image $\vec{x} \in \mathbb{R}^{H \times W}$ in three possible classes

· 1 conv. layer: · 1 $M \times N$ filter · stride $S = 1$ · padding $D = 0$ (no pad) · ReLU = non linear
· $\vec{w}$ = filter weights · $\vec{z}$ = result of convol. ($\vec{z} = \text{conv}(\vec{w}, \vec{x})$) · $\vec{h} = \text{ReLU}(\vec{z})$

· 1 max pooling layer: · 2×2 · stride $S = 2$ · padding $D = 0$ (no pad)
Flatten versions of $\vec{x}, \vec{z}$: $\vec{x}' = \text{vec}(\vec{x})$, $\vec{z}' = \text{vec}(\vec{z})$

· 1 output layer: softmax activation

Number of parameters

		# params
input layer:	nothing to learn, just provide input image's shape	0
conv layer:	CNN learns here → have weight matrices, number of trainable weights: # of filters $\times$ ((filter width $\times$ filter height $\times$ # of channels) + bias) (from practical 8 2.2.1)	$1 \times ((M \cdot N \cdot 1) + 0)$ ↑ assume no bias ?
pool layer:	no learnable params, because we just calculate a specific number → no backprop learning	0
FC layer:	fully connected layer → has learnable params → highest number: because every neuron is connected with every neuron: # of neurons of previous layer $\cdot$ # of neurons of current layer (connects all neurons from output of max pooling to the final 3 neurons of the classification layer) ↳ 3 classes	previous: $\dim(h_2) \cdot 3$ ↖ 3 outputs = #classes $= \frac{H' \cdot W'}{4} \cdot 3$

→ total # of trainable params $N_{CNN} = M \cdot N + \frac{H' \cdot W'}{4} \cdot 3$

• Calculation of h_2:
• convolution:
→ dim of z_1: $H \times W' = (H-M+1) \times (W-N+1)$ (see 2.1.a))
• ReLU changes nothing on dim (because outputs input or 0)
→ dim of h_1 = dim of z_1 = $H' \times W'$
• 2×2 max pool: $\frac{H-M+1}{2} \times \frac{W-N+1}{2}$
flatten
→ dim of h_2: $\frac{H-M+1}{2} \cdot \frac{W-N+1}{2} = \frac{(H-M+1)(W-N+1)}{4} = \frac{H' \cdot W'}{4}$

2 Fully connected layers : output of dim (h_2)!

FC-layer 1:- # params

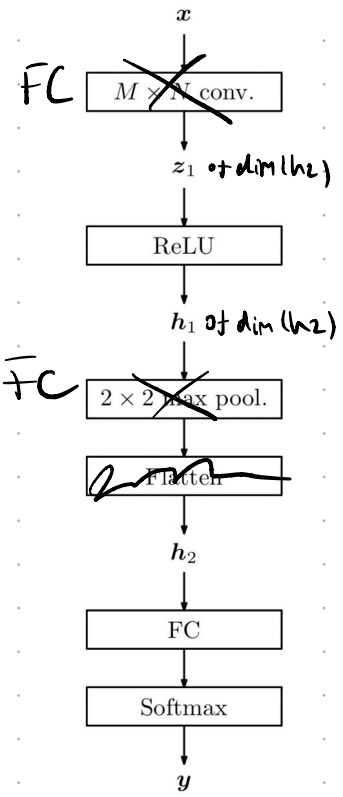
$\dim(\text{input}) \cdot \dim(h_2)$ (or x)

$= (H \cdot W) \cdot \frac{H'W'}{4}$

FC-layer 2:

$\dim(h_2) \cdot \dim(\text{classes})$ (or y)

$= \frac{H'W'}{4} \cdot 3$



⇒ total # of trainable params $N_{FC} = (H \cdot W) \cdot \frac{H'W'}{4} + \frac{H'W'}{4} \cdot 3$

$= \frac{H'W'}{4}(H \cdot W + 3)$

3 Comparison

$N_{CNN} = M \cdot N + \frac{H'W'}{4} \cdot 3$

$N_{FC} = HW \frac{H'W'}{4} + \frac{H'W'}{4} \cdot 3$

1) $\frac{H'W'}{4} > 1$ ← we assume this cause a $\frac{1}{4}$ of 1×1 is weird!

2) $MN < HW$ (because filter size ($1 \cdot N$) is smaller than image for sure! ($H \cdot W$))

⇒ $MN < HW \cdot \frac{H'W'}{4}$

⇒ $N_{CNN} < N_{FC}$

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 $\boldsymbol{x}'$.

Same as in Practical 10, Question 1.2 → just a lot easier + lecture

query: $Q = X \cdot W_Q = x' \cdot W_Q = x'$

(input matrix)

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

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

(matrices)

attention probabilities: $P = \text{softmax}\left(\frac{QK^T}{1}\right)$

$\text{softmax}(x) = 1$

problem: single-head attention = scaled dot product attention?

$$P = \text{softmax}\left(\frac{x' x'^T}{1}\right)$$

attention: $Z = PV$

$$= \text{softmax}\left(\frac{x' x'^T}{1}\right) \cdot x'$$

Scaled Dot-Product Attention

(scaling before doing softmax for gradients)

Problem: As d_K gets large, the variance of $\boldsymbol{q}^T \boldsymbol{k}$ increases, the softmax gets very peaked, hence its gradient gets smaller.

Solution: scale by length of query/key vectors:

$$Z = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right) V.$$

doesn't make it 1 anymore!