MNIST from absolute scratch in C

Emperor Nintri (Dimitri Condoris)

February 1, 2025

1 Introduction

...

2 Functions

In this section, we explain every trick and algorithm used to code the basic functions needed for this project and others.

2.1 String Length Function

In this subsection we detail the algorithm used in order to compute the length of a string.

2.1.1 Core Idea

2.1.2 Counting Trailing Zeros

As we saw in the last sub-subsection, the only remaining function we have to implement is one that, given a binary representation, outputs the number of trailing zeros. The trailing zeros are the bits of value zero before the first bit of value one. To give a concrete example, consider the following binary representation:

$$1100101010111010101010000111010101100110101100101101010001\textcolor{red}{000000}$$

Then the red zeros represent the trailing zeros and hence here we have 6 trailing zeros. Formally speaking we can define this quantity as:

$$T(x) = \max\{k \in \mathbb{N} \mid 2^k \text{ divides } x\}.$$

Here we are only interest in binary representation of size 64. The first observation we can make is that we do not need all the information of x. Mainly, what we do not need is the information regarding all the ones at the left of the first one from the right. To get rid of this information we can apply the formula:

```
long y = (x & -x);
```

Formally, consider that x is equal to $b_{63} \dots b_{T(x)} 0 \dots 0$ with $0 \leq T(x) \leq 63$ and by definition $b_{T(x)} = 1$. Then we have $-x = \text{not}(x) + 1$ which gives us:

$$\begin{aligned} -x &= (1 - b_{63}) \dots (1 - b_{T(x)}) 1 \dots 1 + 0 \dots 00 \dots 1 \\ &= (1 - b_{63}) \dots b_{T(x)} 0 \dots 0. \end{aligned}$$

because the first 1 will carry to the first 0 which for not(x) will have the same position as $T(x)$ for x and hence for $-x$ this position will have value 1. The positions before (from the right) 0 and the ones after will have the same values as not(x) hence the result that:

$$y = 0 \dots b_{T(x)} 0 \dots 0 \text{ with } b_{T(x)} = 1.$$

So now how do we extract the position of this value? Well, we are going to use a property of a *de Bruijn sequence*. A *de Bruijn sequence* in our case will be a cyclic sequence of bits for which whatever the subsequence you take it will be unique. Such a sequence is denoted by $B(k, n)$ where k is the size of the vocabulary we treat (in our case 2 because we only have 0 and 1) and n is the size of the sub-string we want to be unique in the string. Let's say we fix n to 3, then the unique combinations we can make with 0 and 1 of size 3 will be:

$$000, 001, 010, 011, 100, 101, 110 \text{ and } 111$$

One (because there are many different ones) of the $B(2, 3)$ is:

$$00010111$$

Because as you can see the first 3 characters are 000 then the next 3 are 001 etc. We find every combination possible and only one time. If you wonder where is 110, it is composed of the last 2 characters and then the first one because we can cycle through the sequence.

To sum it up really easily, whatever the 3 successive characters that you extract from this sequence; it will be unique. The size of all $B(k, n)$ is k^n. Here we have potentially 64 different values for $T(x)$ and because k is 2, we can use $n = 6$.

We construct one example following a simple algorithm where we try to complete every combination possible:

000000000000000000000000000000000000000000000000000000000000000000000000
000000100000000000000000000000000000000000000000000000000000000000000000
000000110000000000000000000000000000000000000000000000000000000000000000
000000111000000000000000000000000000000000000000000000000000000000000000

Pretty much we just have to slide adding 1 or 0 and doing all the 64 possible combinations.

Of course doing so randomly will not yield a working result, instead we follow the *prefer-largest greedy construction* which consists of starting with $0^{n-1} = 00000$ and add the largest symbol (here 1) if it does not repeat a previous substring of size 6 and else add 0. Once you reach 64 additions you get rid of the first 5 zeros:

000001
000011
000111
001111
011111
111111
111110
111101

I am not going to write everything but the result we get at the end is:

1111110111100111010111000110110100110010110000101010001001000000

Or in reverse (with the least significant byte on the right):

0000001001000101010000110100110010110110001110101110011110111111

Now the interesting question come into play, all that for what? Well, if you multiply this binary number with the representation of x with only one significant byte, then you'll get as the first 6 bits a unique sequence of bits (because every sub-string is unique). For example, if the significant byte is at the first position then the first 6 bits of the multiplication will yield 000000. If it is the second byte it will yield 000001, etc. Because of that we can pre-compute a table that will associate the correct integer to the correct $T(x)$, for example if $T(x)$ equals 0 then the first 6 bits of the multiplication will be 000000 which is 0 in base 2 so in the table at position 0 we will put the result 0.

You can compute the table yourself if you want but here are the coefficients:

0,	1,	2,	18,	3,	10,	19,	39,
7,	4,	11,	30,	26,	20,	40,	51,
16,	8,	5,	24,	14,	12,	31,	45,
36,	27,	21,	33,	47,	41,	52,	57,
63,	17,	9,	38,	6,	29,	25,	50,
15,	23,	13,	44,	35,	32,	46,	56,
62,	37,	28,	49,	22,	43,	34,	55,
61,	48,	42,	54,	60,	53,	59,	58,

So finally to get the number of trailing zeros you simply have to look at this table at the index returned by the previous operation.

2.2 Print Function

In this subsection we focus on the implementation of the print function.

2.2.1 Printing an Integer

2.2.2 Printing a Float

The idea for extracting each digit of the number is to first separate it into the integer part and the fractional part. Then, we divide the integer part by 10, floor the result, multiply it by 10, and subtract the product from the original integer part. This will give us the first digit. Before moving to the fractional part handling, we will focus on this first step.

Here is an example of what we want to do:

```
double x = 10005.0505;
double integer_part = truncate(x); // integer_part = 10005.000000
double divided_part = floor(integer_part / 10); // divided_part = 1000.000000
int first_digit = integer_part - divided_part * 10; // first_digit = 5.000000
```

Now the first question you may ask yourself is why are we using doubles to store the integer part and the divided part? To understand this correctly, let's first recall what is the binary form of a double (of size 64 bits) on a computer:

$$b_{63}b_{62:52}b_{51:0}$$

Here, for $i \in \{0, \dots, 63\}$, b_i represents a bit for which the value is either 0 or 1. If $\{b_i, i \in \{0, \dots, 63\}\}$ represents x then to convert x back to decimal format you apply this formula:

$$\begin{aligned} x &= (-1)^{b_{63}} \times 2^{\sum_{i=0}^{10} b_{52+i} \times 2^i - 1023} \times \left(1 + \left(\sum_{i=1}^{52} b_{52-i} \times 2^{-i}\right)\right) \\ &= (-1)^{b_{63}} \times 2^{e-1023} \times (1+m), \\ &\text{with } e := \left(\sum_{i=0}^{10} b_{52+i} \times 2^i\right) \text{ and } m := \left(\sum_{i=1}^{52} b_{52-i} \times 2^{-i}\right). \end{aligned}$$

For the moment, let's not consider special cases such as subnormal numbers, zeros and infinities.

What we realize is the fact that x can potentially be very large. For example, if the exponent is higher than 64 (the limit here being $2^{11} - 2 - 1023 = 1023$) then x will be bigger than 2^{64} and hence will not be containable in an integer (at least basic ones). This is why we have to use another double to store the integer part of x.

So how are we going to truncate x then? We can't simply cast the variable to an integer here. What we can remark instead is that decimal places are entirely determined by the product $2^{e-1023} \times m$, moreover:

$$2^{e-1023} \times m = \left(\sum_{i=1}^{52} b_{52-i} \times 2^{e-i-1023} \right)$$

More precisely, decimal places are determined by every coefficients b_{52-i} such that $e - i - 1023$ is less than 0 (or $i > e - 1023$). From that we can conclude that to truncate the integer we can simply mask all the bits $\{b_i, i < 1075 - e\}$.

3 Layers

In this section we detail the layers' structure and results for the forward pass and backpropagation. Here we deal with a classification using cross-entropy as a loss function.

3.1 Output Layer

In this subsection we derive the appropriate formulas for a dense layer.

3.1.1 Notations

First of all, we define the notations used in this subsection:

L	loss function
C	number of classes
O	last layer
n	number of samples
$x_{(O)}$	number of features of layer O output (or loss input)
$X_{(O)}$	layer O activated output of size $(n, x_{(O)})$
Y	true one-hot-encoded labels of size (n, C)
$\hat{Y}$	predicted one-hot-encoded labels of size (n, C)

By definition, we have:

$$L := -\sum_{k=1}^{N}\sum_{x=1}^{x_{(O)}} Y(k,x)\log\Big(\hat{Y}(k,x)\Big) \text{ and } \hat{Y}(k,x) := \frac{\exp X_{(O)}(k,x)}{\sum_{y=1}^{x_{(O)}} \exp X_{(O)}(k,y)}.$$

3.1.2 First gradients

Now we can compute the first gradients:

$$\frac{\partial L}{\partial X_{(O)}(k,u)} \text{ with } 1 \leq k \leq n \text{ and } 1 \leq u \leq x_{(O)}.$$

We have:

$$\frac{\partial L}{\partial X_{(O)}(k,u)} = -\sum_{x=1}^{x_{(O)}} Y(k,x) \frac{\partial \hat{Y}(k,x)}{\partial X_{(O)}(k,u)} \frac{1}{\hat{Y}(k,x)}.$$

So considering that for $1 \leq x \leq x_{(O)}$:

$$\frac{\partial \hat{Y}(k,x)}{\partial X_{(O)}(k,u)} = \frac{\exp X_{(O)}(k,x)\Big(\sum_{y=1}^{x_{(O)}} \exp X_{(O)}(k,y)\Big)1_{x=u} - \exp X_{(O)}(k,x)\exp X_{(O)}(k,u)}{\Big(\sum_{y=1}^{x_{(O)}} \exp X_{(O)}(k,y)\Big)^2}.$$

We get:

$$\frac{\partial L}{\partial X_{(O)}(k,u)} = -\sum_{x=1}^{x_{(O)}} Y(k,x)(1_{x=u} - \hat{Y}(k,u)).$$

Now keep in mind that there exists only one $1 \leq z \leq x_{(O)}$ such that $Y(k,z) = 1$, the rest of the values being 0. So we have two cases, if $z = u$ then we have:

$$\begin{aligned}\frac{\partial L}{\partial X_{(O)}(k,u)} &= -(1-\hat{Y}(k,u)) \\ &= \hat{Y}(k,u) - 1 \\ &= \hat{Y}(k,u) - Y(k,u).\end{aligned}$$

Because $Y(k,u) = Y(k,z) = 1$ in this case. Now if $z \neq u$ then we have:

$$\begin{aligned}\frac{\partial L}{\partial X_{(O)}(k,u)} &= -(-\hat{Y}(k,u)) \\ &= \hat{Y}(k,u) \\ &= \hat{Y}(k,u) - Y(k,u).\end{aligned}$$

Because $Y(k,u) = 1 - Y(k,z) = 0$ in this case. We can sum those two results in one stating that:

$$\delta_O(k,x) := \frac{\partial L}{\partial X_{(O)}(k,x)} = \hat{Y}(k,x) - Y(k,x), \text{ for } 1 \leq x \leq x_{(O)}.$$

3.2 Dense Layer

In this subsection we derive the appropriate formulas for a dense layer.

3.2.1 Notations

First of all, we define the notations used in this subsection:

l	current layer
$l-1$	previous layer
n	number of samples
$x_{(l-1)}$	number of features of layer l input (or layer $l-1$ output)
$X_{(l-1)}$	layer l input (or layer $l-1$ output) of size $(n, x_{(l-1)})$
$x_{(l)}$	number of features of layer l output
$W_{(l)}$	layer l weight of size $(x_{(l)}, x_{(l-1)})$
$b_{(l)}$	layer l bias of size $x_{(l)}$
$f_{(l)}$	layer l activation function
$Z_{(l)}$	layer l unactivated output of size $(n, x_{(l)})$
$X_{(l)}$	layer l activated output of size $(n, x_{(l)})$

By definition, we have:

$$Z_{(l)} := W_{(l)} X_{(l-1)}^T + b_{(l)} \text{ and } X_{(l)} := f_{(l)}(Z_{(l)}).$$

3.2.2 Derivatives of Weight and Bias

Let's compute the derivative of the loss relatively to the weight of layer l. For that, we assume that we already know for every $1 \leq k \leq n$ and $1 \leq x \leq x_{(l)}$ the value of:

$$\delta_{(l)}(k,x) := \frac{\partial L}{\partial X_{(l)}(k,x)}.$$

Let $1 \leq u \leq x_{(l)}$ and $1 \leq v \leq x_{(l-1)}$. We have:

$$\begin{aligned}
\frac{\partial L}{\partial W_{(l)}(u,v)} &= \sum_{k=1}^{n} \sum_{x=1}^{x_{(l)}} \frac{\partial L}{\partial X_{(l)}(k,x)} \frac{\partial X_{(l)}(k,x)}{\partial W_{(l)}(u,v)} \\
&= \sum_{k=1}^{n} \sum_{x=1}^{x_{(l)}} \delta_{(l)}(k,x) \frac{\partial X_{(l)}(k,x)}{\partial W_{(l)}(u,v)}.
\end{aligned}$$

We only have to compute the second term, let $1 \leq k \leq n$ and $1 \leq x \leq x_{(l)}$:

$$\begin{aligned}
\frac{\partial X_{(l)}(k,x)}{\partial W_{(l)}(u,v)} &= \frac{\partial X_{(l)}(k,x)}{\partial Z_{(l)}(k,x)} \frac{\partial Z_{(l)}(k,x)}{\partial W_{(l)}(u,v)} \\
&= f'_{(l)}(Z_{(l)}(k,x)) \frac{\partial Z_{(l)}(k,x)}{\partial W_{(l)}(u,v)} \\
&= f'_{(l)}(Z_{(l)}(k,x)) X_{(l-1)}(k,v) 1_{x=u}.
\end{aligned}$$

This simplifies to:

$$\frac{\partial L}{\partial W_{(l)}(u,v)} = \sum_{k=1}^{n} \delta_{(l)}(k,u) f'_{(l)}(Z_{(l)}(k,u)) X_{(l-1)}(k,v).$$

Similarly we have:

$$\begin{aligned}
\frac{\partial X_{(l)}(k,x)}{\partial b_{(l)}(u)} &= \frac{\partial X_{(l)}(k,x)}{\partial Z_{(l)}(k,x))} \frac{\partial Z_{(l)}(k,x)}{\partial b_{(l)}(u)} \\
&= f'_{(l)}(Z_{(l)}(k,x)) \frac{\partial Z_{(l)}(k,x)}{\partial b_{(l)}(u)} \\
&= f'_{(l)}(Z_{(l)}(k,x)) 1_{x=u}.
\end{aligned}$$

And hence:

$$\frac{\partial L}{\partial b_{(l)}(u)} = \sum_{k=1}^{n} \delta_{(l)}(k,u) f'_{(l)}(Z_{(l)}(k,u)).$$

3.2.3 Computing Next Gradients

Now we can compute the next gradients:

$$\frac{\partial L}{\partial X_{(l-1)}(k,y)} \text{ with } 1 \leq k \leq n \text{ and } 1 \leq y \leq x_{(l-1)}.$$

We have:

$$\begin{aligned}
\frac{\partial L}{\partial X_{(l-1)}(k,y)} &= \sum_{x=1}^{x_{(l)}} \frac{\partial L}{\partial X_{(l)}(k,x)} \frac{\partial X_{(l)}(k,x)}{\partial Z_{(l)}(k,x)} \frac{\partial Z_{(l)}(k,x)}{\partial X_{(l-1)}(k,y)} \\
&= \sum_{x=1}^{x_{(l)}} \delta_{(l)}(k,x) f'_{(l)}(Z_{(l)}(k,x)) W_{(l)}(x,y).
\end{aligned}$$

3.3 Convolutional Layer

In this subsection we derive the appropriate formulas for a convolutional layer.

3.3.1 Notations

First of all, we define the notations used in this subsection:

l	current layer
$l-1$	previous layer
n	number of samples
$h_{(l-1)}$	number of rows (height) of layer l input (or layer $l-1$ output)
$w_{(l-1)}$	number of columns (width) of layer l input (or layer $l-1$ output)
$c_{(l-1)}$	number of channels of layer l input (or layer $l-1$ output)
$F_{(l)}$	filter size for both height and width
$P_{(l)}$	padding
$S_{(l)}$	stride
$A_{(l-1)}$	layer l input (or layer $l-1$ output) of size $(n, h_{(l-1)}, w_{(l-1)}, c_{(l-1)})$
$B_{(l-1)}$	padded version of $A_{(l-1)}$ of size $(n, h_{(l-1)} + 2P_{(l)}, w_{(l-1)} + 2P_{(l)}, c_{(l-1)})$
$h_{(l)}$	height of layer l output
$w_{(l)}$	width of layer l output
$c_{(l)}$	number of channels of layer l output
$W_{(l)}$	layer l weight of size $(F_{(l)}, F_{(l)}, c_{(l)}, c_{(l-1)})$
$b_{(l)}$	layer l bias of size $c_{(l)}$
$f_{(l)}$	layer l activation function
$Z_{(l)}$	layer l unactivated output of size $(n, h_{(l)}, w_{(l)}, c_{(l)})$
$A_{(l)}$	layer l activated output of size $(n, h_{(l)}, w_{(l)}, c_{(l)})$

By definition, we have:

$$h_{(l)} := \left\lfloor \frac{h_{(l-1)} + 2P_{(l)} - F_{(l)}}{S_{(l)}} \right\rfloor + 1 \text{ and } w_{(l)} := \left\lfloor \frac{w_{(l-1)} + 2P_{(l)} - F_{(l)}}{S_{(l)}} \right\rfloor + 1.$$

Let $1 \leq k \leq n$, $1 \leq i \leq h_{(l)}$, $1 \leq j \leq w_{(l)}$ and $1 \leq c_o \leq c_{(l)}$. We also have:

$$Z_{(l)}(k,i,j,c_o) := \sum_{c_i=1}^{c_{(l)}} \sum_{u=1}^{F_{(l)}} \sum_{v=1}^{F_{(l)}} W_{(l)}(u,v,c_i,c_o) B_{(l-1)}(k, (i-1)S_{(l)} + u, (j-1)S_{(l)} + v, c_i) + b_{(l)}(c_o),$$

$$A_{(l)}(k,i,j,c_o) := f_{(l)}(Z_{(l)}(k,i,j,c_o)).$$

3.3.2 Derivatives of Weight and Bias

Let's compute the derivative of the loss relatively to the weight of layer l. For that, we assume that we already know for every $1 \leq k \leq n$, $1 \leq i \leq h_{(l)}$, $1 \leq j \leq w_{(l)}$ and $1 \leq c_o \leq c_{(l)}$ the value of:

$$\delta_{(l)}(k,i,j,c_o) := \frac{\partial L}{\partial A_{(l)}(k,i,j,c_o)}.$$

Let $1 \leq u \leq F_{(l)}$, $1 \leq v \leq F_{(l)}$, $1 \leq c_i \leq c_{(l-1)}$ and $1 \leq c_o \leq c_{(l)}$. We have:

$$
\begin{aligned}
\frac{\partial L}{\partial W_{(l)}(u,v,c_i,c_o)} &= \sum_{k=1}^{n}\sum_{i=1}^{h_{(l)}}\sum_{j=1}^{w_{(l)}} \frac{\partial L}{\partial A_{(l)}(k,i,j,c_o)} \frac{\partial A_{(l)}(k,i,j,c_o)}{\partial W_{(l)}(u,v,c_i,c_o)} \\
&= \sum_{k=1}^{n}\sum_{i=1}^{h_{(l)}}\sum_{j=1}^{w_{(l)}} \delta_{(l)}(k,i,j,c_o) \frac{\partial A_{(l)}(k,i,j,c_o)}{\partial W_{(l)}(u,v,c_i,c_o)}.
\end{aligned}
$$

We only have to compute the second term, let $1 \leq k \leq n$, $1 \leq i \leq h_{(l)}$, $1 \leq j \leq w_{(l)}$ and $1 \leq c_o \leq c_{(l)}$:

$$
\begin{aligned}
\frac{\partial A_{(l)}(k,i,j,c_o)}{\partial W_{(l)}(u,v,c_i,c_o)} &= \frac{\partial A_{(l)}(k,i,j,c_o)}{\partial Z_{(l)}(k,i,j,c_o))} \frac{\partial Z_{(l)}(k,i,j,c_o)}{\partial W_{(l)}(u,v,c_i,c_o)} \\
&= f'_{(l)}(Z_{(l)}(k,i,j,c_o)) \frac{\partial Z_{(l)}(k,i,j,c_o)}{\partial W_{(l)}(u,v,c_i,c_o)} \\
&= f'_{(l)}(Z_{(l)}(k,i,j,c_o)) B_{(l-1)}(k,(i-1)S_{(l)}+u,(j-1)S_{(l)}+v,c_i).
\end{aligned}
$$

Similarly we have:

$$
\begin{aligned}
\frac{\partial A_{(l)}(k,i,j,c_o)}{\partial b_{(l)}(c_o)} &= \frac{\partial A_{(l)}(k,i,j,c_o)}{\partial Z_{(l)}(k,i,j,c_o))} \frac{\partial Z_{(l)}(k,i,j,c_o)}{\partial b_{(l)}(c_o)} \\
&= f'_{(l)}(Z_{(l)}(k,i,j,c_o)) \frac{\partial Z_{(l)}(k,i,j,c_o)}{\partial b_{(l)}(c_o)} \\
&= f'_{(l)}(Z_{(l)}(k,i,j,c_o)).
\end{aligned}
$$

And hence:

$$
\begin{aligned}
\frac{\partial L}{\partial b_{(l)}(c_o)} &= \sum_{k=1}^{n}\sum_{i=1}^{h_{(l)}}\sum_{j=1}^{w_{(l)}} \frac{\partial L}{\partial A_{(l)}(k,i,j,c_o)} \frac{\partial A_{(l)}(k,i,j,c_o)}{\partial b_{(l)}(c_o)} \\
&= \sum_{k=1}^{n}\sum_{i=1}^{h_{(l)}}\sum_{j=1}^{w_{(l)}} \delta_{(l)}(k,i,j,c_o) f'_{(l)}(Z_{(l)}(k,i,j,c_o)).
\end{aligned}
$$

3.3.3 Computing Next Gradients

Now we can compute the next gradients:

$$
\frac{\partial L}{\partial A_{(l-1)}(k,a,b,c_i)} \text{ with } 1 \leq k \leq n,\ 1 \leq a \leq h_{(l-1)},\ 1 \leq b \leq w_{(l-1)} \text{ and } 1 \leq c_i \leq c_{(l-1)}.
$$

We have:

$$
\begin{aligned}
\frac{\partial L}{\partial A_{(l-1)}(k,a,b,c_i)} &= \sum_{i=1}^{h_{(l)}}\sum_{j=1}^{w_{(l)}}\sum_{c_o=1}^{c_{(l)}} \frac{\partial L}{\partial A_{(l)}(k,i,j,c_o)} \frac{\partial A_{(l)}(k,i,j,c_o)}{\partial Z_{(l)}(k,i,j,c_o)} \frac{\partial Z_{(l)}(k,i,j,c_o)}{\partial A_{(l-1)}(k,a,b,c_i)} \\
&= \sum_{i=1}^{h_{(l)}}\sum_{j=1}^{w_{(l)}}\sum_{c_o=1}^{c_{(l)}} \delta_{(l)}(k,i,j,c_o) f'_{(l)}(Z_{(l)}(k,i,j,c_o)) \frac{\partial Z_{(l)}(k,i,j,c_o)}{\partial A_{(l-1)}(k,a,b,c_i)}.
\end{aligned}
$$

Let $1 \leq i \leq h_{(l)}$, $1 \leq j \leq w_{(l)}$ and $1 \leq c_o \leq c_{(l)}$. Let's focus on the last term of the sum:

$$
\frac{\partial Z_{(l)}(k,i,j,c_o)}{\partial A_{(l-1)}(k,a,b,c_i)} = \frac{\partial Z_{(l)}(k,i,j,c_o)}{\partial B_{(l-1)}(k,a+P_{(l)},b+P_{(l)},c_i)} \frac{\partial B_{(l-1)}(k,a+P_{(l)},b+P_{(l)},c_i)}{\partial A_{(l-1)}(k,a,b,c_i)}.
$$

Remember that B is pretty much a padded version of A, we conclude that the second term will here be 1. So we are left computing the first term. For that, recall the following formula:

$$Z_{(l)}(k,i,j,c_o) = \sum_{c_i=1}^{c_{(l)}} \sum_{u=1}^{F_{(l)}} \sum_{v=1}^{F_{(l)}} W_{(l)}(u,v,c_i,c_o) B_{(l-1)}(k,(i-1)S_{(l)}+u,(j-1)S_{(l)}+v,c_i) + b_{(l)}(c_o).$$

From this formula it is pretty obvious that the derivative will be equal to $W_{(l)}(u,v,c_i,c_o)$ if there exists $1 \leq u \leq F_{(l)}$ and $1 \leq v \leq F_{(l)}$ such that:

$$a + P_{(l)} = (i-1)S_{(l)} + u \text{ and } b + P_{(l)} = (j-1)S_{(l)} + v.$$

If those do not exist, then the derivative will be 0. Now we can compute for which i and j this is the case (to avoid testing every combination in practice). Let's focus on the "rows" for the moment (the "columns" being symmetrical in reasoning). Such an u exists if we have:

$$\begin{aligned} 1 \leq a + P_{(l)} - (i-1)S_{(l)} \leq F_{(l)} &\iff 1 - a - P_{(l)} \leq -(i-1)S_{(l)} \leq F_{(l)} - a - P_{(l)} \\ &\iff \frac{a + P_{(l)} - 1}{S_{(l)}} + 1 \leq i \leq \frac{a + P_{(l)} - F_{(l)}}{S_{(l)}} + 1. \end{aligned}$$

Considering the initial domain of i, we can deduce that this exists if:

$$i \in \left\{ \max\left(1, \left\lceil \frac{a + P_{(l)} - 1}{S_{(l)}} + 1 \right\rceil\right), \ldots, \min\left(h_{(l)}, \left\lfloor \frac{a + P_{(l)} - F_{(l)}}{S_{(l)}} + 1 \right\rfloor\right) \right\}.$$

Similarly, we have:

$$j \in \left\{ \max\left(1, \left\lceil \frac{b + P_{(l)} - 1}{S_{(l)}} + 1 \right\rceil\right), \ldots, \min\left(w_{(l)}, \left\lfloor \frac{b + P_{(l)} - F_{(l)}}{S_{(l)}} + 1 \right\rfloor\right) \right\}.$$

Finally for those values of i and j the derivative will be equal to:

$$W_{(l)}(a + P_{(l)} - (i-1)S_{(l)}, b + P_{(l)} - (j-1)S_{(l)}, c_i, c_o).$$

3.4 Pooling Layer

In this subsection we derive the appropriate formulas for a pooling layer.

3.4.1 Notations

First of all, we define the notations used in this subsection:

l	current layer
$l-1$	previous layer
n	number of samples
$h_{(l-1)}$	number of rows (height) of layer l input (or layer $l-1$ output)
$w_{(l-1)}$	number of columns (width) of layer l input (or layer $l-1$ output)
$c_{(l-1)}$	number of channels of layer l input (or layer $l-1$ output)
$F_{(l)}$	pooling size for both height and width
$P_{(l)}$	padding
$S_{(l)}$	stride
$A_{(l-1)}$	layer l input (or layer $l-1$ output) of size $(n, h_{(l-1)}, w_{(l-1)}, c_{(l-1)})$
$B_{(l-1)}$	padded version of $A_{(l-1)}$ of size $(n, h_{(l-1)} + 2P_{(l)}, w_{(l-1)} + 2P_{(l)}, c_{(l-1)})$
$h_{(l)}$	height of layer l output
$w_{(l)}$	width of layer l output
$f_{(l)}$	average function or maximum function
$A_{(l)}$	layer l output of size $(n, h_{(l)}, w_{(l)}, c_{(l-1)})$

By definition, we have:

$$h_{(l)} := \left\lfloor \frac{h_{(l-1)} + 2P_{(l)} - F_{(l)}}{S_{(l)}} \right\rfloor + 1 \text{ and } w_{(l)} := \left\lfloor \frac{w_{(l-1)} + 2P_{(l)} - F_{(l)}}{S_{(l)}} \right\rfloor + 1.$$

Let $1 \le k \le n$, $1 \le i \le h_{(l)}$, $1 \le j \le w_{(l)}$ and $1 \le c \le c_{(l-1)}$. We also have:

$$A_{(l)}(k,i,j,c) := f_{(l)}(\{B_{(l-1)}(k,(i-1)S_{(l)} + u, (j-1)S_{(l)} + v, c) | 1 \le u \le F_{(l)} \text{ and } 1 \le v \le F_{(l)}\}).$$

3.4.2 Computing Next Gradients

Now we can compute the next gradients. For that, we assume that we already know for every $1 \le k \le n$, $1 \le i \le h_{(l)}$, $1 \le j \le w_{(l)}$ and $1 \le c \le c_{(l-1)}$ the value of:

$$\delta_{(l)}(k,i,j,c) := \frac{\partial L}{\partial A_{(l)}(k,i,j,c)}.$$

We are interested in:

$$\frac{\partial L}{\partial A_{(l-1)}(k,a,b,c)} \text{ with } 1 \le k \le n,\ 1 \le a \le h_{(l-1)},\ 1 \le b \le w_{(l-1)} \text{ and } 1 \le c \le c_{(l-1)}.$$

We have:

$$\begin{aligned}
\frac{\partial L}{\partial A_{(l-1)}(k,a,b,c)} &= \sum_{i=1}^{h_{(l)}} \sum_{j=1}^{w_{(l)}} \frac{\partial L}{\partial A_{(l)}(k,i,j,c)} \frac{\partial A_{(l)}(k,i,j,c)}{\partial A_{(l-1)}(k,a,b,c)} \\
&= \sum_{i=1}^{h_{(l)}} \sum_{j=1}^{w_{(l)}} \delta_{(l)}(k,i,j,c) \frac{\partial A_{(l)}(k,i,j,c)}{\partial A_{(l-1)}(k,a,b,c)}.
\end{aligned}$$

Let $1 \le i \le h_{(l)}$, $1 \le j \le w_{(l)}$ and $1 \le c \le c_{(l-1)}$. Let's focus on the last term of the sum:

$$\frac{\partial A_{(l)}(k,i,j,c)}{\partial A_{(l-1)}(k,a,b,c)} = \frac{\partial A_{(l)}(k,i,j,c)}{\partial B_{(l-1)}(k,a+P_{(l)},b+P_{(l)},c)} \frac{\partial B_{(l-1)}(k,a+P_{(l)},b+P_{(l)},c)}{\partial A_{(l-1)}(k,a,b,c)}.$$

Remember that B is pretty much a padded version of A, we conclude that the second term will here be 1. So we are left computing the first term. For that, recall the following formula:

$$A_{(l)}(k,i,j,c) := f_{(l)}(\{B_{(l-1)}(k,(i-1)S_{(l)}+u,(j-1)S_{(l)}+v,c)|1 \leq u \leq F_{(l)} \text{ and } 1 \leq v \leq F_{(l)}\}).$$

From this formula it is pretty obvious that the derivative will be equal to $f'_{(l)}$ with respect to $B_{(l-1)}(k, a+P_{(l)}, b+P_{(l)}, c)$ evaluated at the given set if there exists $1 \leq u \leq F_{(l)}$ and $1 \leq v \leq F_{(l)}$ such that:

$$a + P_{(l)} = (i-1)S_{(l)} + u \text{ and } b + P_{(l)} = (j-1)S_{(l)} + v.$$

If those do not exist, then the derivative will be 0. Now we can compute for which i and j this is the case (to avoid testing every combination in practice). Let's focus on the "rows" for the moment (the "columns" being symmetrical in reasoning). Such an u exists if we have:

$$\begin{aligned} 1 \leq a + P_{(l)} - (i-1)S_{(l)} \leq F_{(l)} &\iff 1 - a - P_{(l)} \leq -(i-1)S_{(l)} \leq F_{(l)} - a - P_{(l)} \\ &\iff \frac{a + P_{(l)} - 1}{S_{(l)}} + 1 \leq i \leq \frac{a + P_{(l)} - F_{(l)}}{S_{(l)}} + 1. \end{aligned}$$

Considering the initial domain of i, we can deduce that this exists if:

$$i \in \left\{ \max\left(1, \left\lceil \frac{a + P_{(l)} - 1}{S_{(l)}} + 1 \right\rceil\right), \ldots, \min\left(h_{(l)}, \left\lfloor \frac{a + P_{(l)} - F_{(l)}}{S_{(l)}} + 1 \right\rfloor\right) \right\}.$$

Similarly, we have:

$$j \in \left\{ \max\left(1, \left\lceil \frac{b + P_{(l)} - 1}{S_{(l)}} + 1 \right\rceil\right), \ldots, \min\left(w_{(l)}, \left\lfloor \frac{b + P_{(l)} - F_{(l)}}{S_{(l)}} + 1 \right\rfloor\right) \right\}.$$

Finally for those values of i and j the derivative will be equal to:

- 1 if we are using max pooling and the max is reached in $B_{(l-1)}(k, a+P_{(l)}, b+P_{(l)}, c)$,

- 0 if we are using max pooling and the max is not reached in $B_{(l-1)}(k, a+P_{(l)}, b+P_{(l)}, c)$,

- $\frac{1}{F_l^2}$ if we are using average pooling.

For the max pooling case, instead of recomputing the maximum we can verify for the valid i and j if $A_{(l)}(k,i,j,c) = A_{(l-1)}(k,a,b,c)$. Computationally speaking it will be way lighter.