On the Construction of Numerical Models through a Prime Convolutional Approach

Proceedings of the 33rd European Safety and Reliability Conference (ESREL 2023)
Edited by Mário P. Brito, Terje Aven, Piero Baraldi, Marko Čepin and Enrico Zio
 Published by Research Publishing, Singapore.
doi: 10.3850/978-981-18-8071-1_P672-cd

On the Construction of Numerical Models through a Prime Convolutional Approach

Doaa Almhaithawi*,[1]
E-mail: *doaa.almhaithawi@polito.it*

Francesco Panelli*,[3]
E-mail: *francesco.panelli@mathema.com*

Massimo Bertini[2]
E-mail: *mbert@mathema.com*

Alessandro Bellini[2]
E-mail: *abel@mathema.com*

Stefano Cuomo[2]
E-mail: *stefano.cuomo@mathema.com*

Tania Cerquitelli[1]
E-mail: *tania.cerquitelli@polito.it*

*Corresponding author.
[1]Department of Control and Computer Engineering, Politecnico di Torino, Torino, Italy.
[2]Mathema, Italy.
[3]Independent researcher.

Abstract

In this paper we apply neural network models to a set of natural numbers in order to classify the congruence classes modulo a given integer $m \in \{2, 3, \ldots, 10\}$. We compare the performances of two kinds of architectures and of several input data representations. It turns out that these tasks are fully solved using a convolutional architecture and a special representation for the input data that exploits the prime factor decomposition of numbers.

Keywords: Neural Networks, Natural Numbers, Convolutional Networks, Prime Numbers, Congruence Classes

1. Introduction

Neural Networks are a powerful tool in Machine Learning that has been successfully used in solving several kinds of problems; the tasks that can be addressed with them are extremely diverse in nature and have a wide spectrum of applicability that ranges from image recognition to word generation, medical diagnosis and much more (cf. de Rosa and Papa (2021),Gatt and Krahmer (2018), Hu et al. (2019), Li et al. (2020), Liu et al. (2019), Papastratis et al. (2020), Richardson et al. (2021), Sarvamangala and Kulkarni (2022)).

In its simplest form, a Neural Network can be formalized as a composition $\mathcal{N} = f_1 \circ \cdots \circ f_k$ of functions $f_i : \mathbb{R}^{n_i} \to \mathbb{R}^{n_{i+1}}$, each linking two real vector spaces. All of these spaces, except for the first and the last one, are called *latent spaces*.

Usefulness of Neural Networks comes from their applicability in all those problems that require the approximation of an unknown function ϕ. This approximation process can be formalized as follows. We consider families of functions $f_{i,\theta_i} : \mathbb{R}^{n_i} \to \mathbb{R}^{n_{i+1}}$, $i = 1, \ldots, k$, each depending on a set of parameters θ_i (whose elements are called *weights*), and we consider the family of Neural Networks $\mathcal{N}_\Theta = f_{1,\theta_1} \circ \cdots \circ f_{k,\theta_k}$ depending on the set of parameters $\Theta = \bigcup_{i=1}^{k} \theta_i$. The set Θ is initialized randomly; then, through a suitable optimization process, called *training*, the best set $\hat{\Theta}$ is found so that the corresponding network $\mathcal{N}_{\hat{\Theta}}$ is an acceptable approximation of ϕ.

In order to be processed by a network $\mathcal{N} = f_1 \circ \cdots \circ f_k$, the input data has to be given a vector representation; the vectors representing the input data are then moved by the functions f_i from one latent space to the next one up to the output space. In each of the latent spaces the dispositions of the data can be studied geometrically, and the emerging geometric properties can be compared with the semantic features. It has been observed in several papers (cf. Abdal et al. (2019), Fetty et al. (2020), Giardina et al. (2022), Shen et al. (2020),

Vaswani et al. (2007)) that in many remarkable cases and for several kinds of networks there exist in fact relationships between the two.

This phenomenon is not fully understood yet and it appears to be rather complex and multi-faceted. The study of the geometry of the latent spaces, in particular, is very challenging and has been carried out in many different ways by many authors (cf. Arvanitidis et al. (2017), Connor and Rozell (2020), Donoho and Grimes (2005), Smith et al. (2019)).

The main problem in relating geometric properties of the latent spaces with semantic features of the data relies on the fact that while the former are objectively determined characteristics of mathematical entities, the latter need not necessarily be so. Consequently, in many of the situations analyzed in the literature, understanding whether it is possible to assign a semantic interpretation to a geometric parameter – or, conversely, whether it is possible to characterize a feature of the data using a geometric parameter – turns out to be a rather ambiguous task. Furthermore, most of the examples in this direction usually deal with data (e.g. images) that possess several features wildly intertwining with one another, and this makes the picture still more complex and of difficult interpretation.

In order to overcome these issues, we suggest the use Neural Networks to address problems related to a dataset with objective features – a mathematical dataset. Our choice falls on the set $\mathbb{N}$ of natural numbers for its intuitiveness, immediacy and easy implementability. For evident reasons of finiteness, we take into consideration only the first one million natural numbers (from 0 to 999 999).

In this paper we train networks so to identify the congruence classes modulo a given integer $m \in \{2, 3, \ldots, 10\}$. The networks that we build have different structures and process input data with various vector representations. It turns out that the tasks under consideration are fully solved using a convolutional architecture and a particular representation for the input data that exploits the concept of "Prime Grid".

The work is organized as follows. In Section 2 we describe the settings of our experiments. In particular: some modes of vectorial representation of natural numbers (Subsection 2.1), the architectures of the neural networks that we build (Subsection 2.2), the parameters adopted and the general functioning of the training process (Subsection 2.3). In Section 3 we present the results of our experiments (cf. also Tables 2, 3 and 4). Finally, in Section 4 we comment, analyze and interpret the results.

2. Experiment Setting

2.1. *Input Data Representation*

Having chosen to work with a dataset of natural numbers, the first problem to face is how to represent such numbers so to make them accessible by a network. In contrast to other kinds of datasets for which there is a "natural" vector representation (e.g. images, that may be represented as matrices of pixels), for natural numbers several equally pleasurable vector representations are available, and we shall now briefly describe them.

Given any $b \in \mathbb{N}$, $b \geq 2$ (called *base*) it is a well-known fact from elementary Arithmetic that every $n \in \mathbb{N}$ may be uniquely expressed in the form

$$n = \sum_{i=0}^{k} d_i b^i \tag{1}$$

for suitable $d_0, d_1, \ldots, d_k \in \{0, 1, \ldots, b-1\}$ (called *digits*) and $k \in \mathbb{N}$. We can therefore represent n using the vector

$$(d_0, d_1, \ldots, d_k) \in \mathbb{R}^{k+1}$$

where $k = \lfloor \log_b n \rfloor$ and $d_0, d_1, \ldots, d_k$ are as in (1). This will be called the *b-adic vector representation* of n.

It is clear that the b-adic representation requires b distinct symbols to denote the b distinct digits; for evident computational reasons we shall therefore impose the limitations $2 \leq b \leq 10$ on the possible values of the base b in our analysis. It is also clear that the input space for a network working with our dataset of natural numbers from 0 to 999 999 will be $\mathbb{R}^N$, where

$$N = \lceil \log_b(1\,000\,000) \rceil$$

(a number ranging from $N = 6$ when $b = 10$ to $N = 20$ when $b = 2$).

The vector representations described so far depend on the choice of a base b, thus they seem not to give an intrinsic description of numbers. This fact suggests to look for a new vector representation that, on the contrary, be more closely related to the arithmetic structure of the set $\mathbb{N}$. We thus construct a vector representation for natural numbers based on the concept of "Prime Grid", introduced in Kolossváry and Kolossváry (2022).

Let $\mathcal{P} = \{p_i \,|\, i = 1, 2, \dots\}$ be the sequence of all prime numbers in ascending order, and call *Prime Grid* the set $\mathbb{N}^{\mathcal{P}}$ of all the infinite sequences of natural numbers indexed on the set $\mathcal{P}$. Following Kolossváry and Kolossváry (2022) we can, in view of the so-called Fundamental Theorem of Arithmetic, associate to each $n \in \mathbb{N}$ an element

$$(\ell_1, \ell_2, \ell_3 \dots) \tag{2}$$

of the Prime Grid so that

$$n = \prod_{i=1}^{\infty} p_i^{\ell_i}. \tag{3}$$

Observe that (3) is a finite product, since in it there are only finitely many exponents taking positive values; the sequence (2) used to identify the natural number n, called in Kolossváry and Kolossváry (2022) the *prime signature* of n, is therefore eventually zero.

These considerations suggest the possibility of exploiting the prime signature of numbers to represent them as vectors: given $n \in \mathbb{N}$, we can in fact cut its prime signature (2) after the k-th entry if $\ell_i = 0$ for all $i > k$, and use the element of $\mathbb{R}^k$ so obtained as a vector representation of n. This means in particular that our dataset of natural numbers from 0 to 999 999 can be faithfully represented by vectors in $\mathbb{R}^N$, where $N = 78\,498$ is the number of primes $< 1\,000\,000$. This will be called the *Prime Grid vector representation* of the numbers.

Now, this value N for the dimension of the input space appears excessively big. A quick glance to the data shows in fact that the prime signatures of the numbers between 0 and 999 999 are very sparse sequences with most of the entries equal to 0 and, furthermore, almost all of their non-zero entries take only very small values. In order to make such observations more precise, we conduct a simple analysis on these prime signatures: for a few values of $p \in \mathbb{N}$ we determine how many natural numbers between 2 and 999 999 can be factorized using the first p primes only. The results, shown in Table 1, indicate that about the 80% of them has a prime signature $(\ell_1, \ell_2, \ell_3, \dots)$ that satisfies $\ell_i = 0$ for all $i > 5\,000$. This leads us to reduce our dataset to precisely those numbers that can be factorized using only the first 5 000 primes. In particular, this means that using the Prime Grid vector representation the dimension of the input space is set to $N = 5\,000$.

Table 1. Number of naturals between 2 and 999 999 that can be factorized with the first p primes only.

p	Number	Percentage
100	259 223	$\sim 26\%$
1000	604 017	$\sim 60\%$
5 000	785 095	$\sim 79\%$

2.2. *Architectures*

For our experiments we construct networks following two different types of architecture:

- **MLP** (cf. Figure 1): This is a standard multilayer perceptron architecture with five fully connected linear layers, each followed by the Leaky ReLU function (except for the last one which is only linear). For the b-adic vector representations the dimensions of the layers follow the scheme:

$$N \to 100 \to 50 \to 10 \to 5 \to m \tag{4}$$

where the input space has dimension

$$N = \lceil \log_b(1\,000\,000) \rceil.$$

For the Prime Grid vector representation, whose input space has a much bigger dimension ($N = 5\,000$), we use the following scheme instead:

$$N \to 1000 \to 500 \to 100 \to 50 \to m. \tag{5}$$

In both cases, the dimension m of the output space depends on the task adopted for the training process.

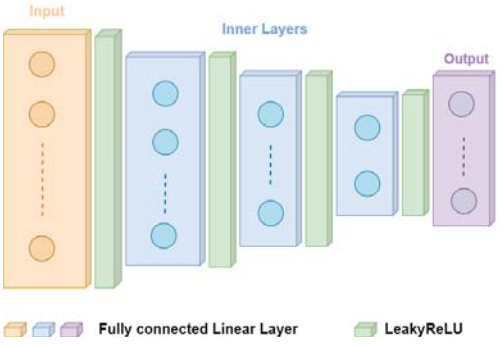

Fig. 1. MLP architecture

- **CNN** (cf. Figure 2): This is the standard architecture for a convolutional Neural Network (cf. O'Shea and Nash (2015)). In contrast to MLP, in this case each element n of the dataset is organized into a $B \times N$ matrix whose rows contain the vector representations of a sequence of B consecutive numbers starting with n. This matrix is then processed in C different channels, each of which is acted on by a learnable $k \times k$ kernel. In our model, there are two convolutional layers, each followed by a max pooling with a stride of 2. The first one of the convolutional layers has $C = 64$ and $k = 7$, while the second one has $C = 128$ and $k = 3$. The length B of the sequence of numbers is set to $B = 8$ in all of the experiments. After the convolutional layers there are a flatten layer and, subsequently, four fully connected linear layers each followed (with the exception of the last one) by the Leaky ReLU function. For the b-adic vector representations the dimensions of the linear layers are taken according to the scheme

$$F \to 100 \to 50 \to 10 \to m, \qquad (6)$$

where F is the dimension of the flatten layer and m is the dimension of the output space. For the Prime Grid vector representation we use instead the scheme

$$F \to 1000 \to 100 \to 10 \to m. \qquad (7)$$

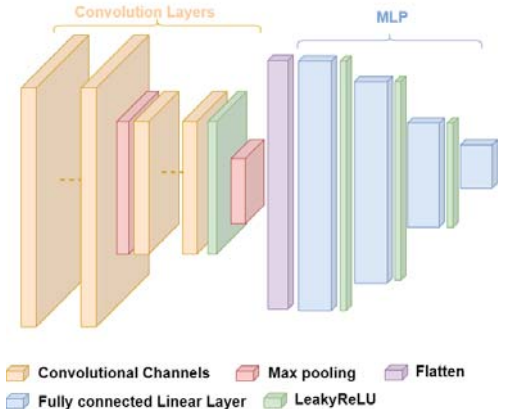

Fig. 2. CNN architecture

2.3. *Training Process*

As mentioned in Section 1, we work with a dataset of natural numbers so to have the possibility of dealing with objective features. Among the several available choices, we decide to train our networks so to identify the congruence classes modulo a given number $m \in \mathbb{N}$.

Recall from the so-called Modular Arithmetic that, given two integers $z_1, z_2 \in \mathbb{Z}$, we write

$$z_1 \equiv z_2 \mod m$$

if and only if m divides the difference $z_1 - z_2$; in this case z_1 is said to be *congruent* to z_2 modulo m. The congruence classes modulo m are the sets

$$[\rho]_m = \{\rho + mz \,|\, z \in \mathbb{Z}\}, \quad \rho = 0, 1, \ldots, m-1$$

and the number $\rho \in \{0, 1, \ldots, m-1\}$ is called the *residue* modulo m of the class $[\rho]_m$; evidently an integer a is in $[\rho]_m$ if and only if ρ is the remainder of the division of a by m.

Let $m \in \mathbb{N}$ be fixed. We train a network so to determine, for a given natural number, its residue modulo m. In order to achieve this goal we select two disjoint subsets of our dataset $\mathcal{D}$: the training set $\mathcal{D}_\mathrm{t}$ and the validation set $\mathcal{D}_\mathrm{v} = \mathcal{D} \backslash \mathcal{D}_\mathrm{t}$.

The training set $\mathcal{D}_{\mathrm{t}}$ consists of r randomly selected disjoint batches $\mathcal{B}_1, \ldots, \mathcal{B}_r$, each containing s elements. The training process is split into t epochs. During each epoch, for $i = 1, \ldots, r$ the model is applied to the batch $\mathcal{B}_i$, the weights are optimized accordingly and then the new model is applied to the batch $\mathcal{B}_{i+1}$.

In the case of b-adic vector representations, the values of the parameters r, s and t are chosen depending on the architecture of the network:

- **MLP**: we take $r = 66$, $s = 10\,000$ and $t = 60$. In this case the cardinality of $\mathcal{D}_{\mathrm{t}}$ is 660 000 so that it occupies about 2/3 of the whole dataset.
- **CNN**: we take $r = 400$, $s = 256$ and $t = 10$. In this case the cardinality of $\mathcal{D}_{\mathrm{t}}$ is 102 400 so that it occupies only 10% of the whole dataset.

The case of the Prime Grid vector representation is treated differently because of its elevated computational complexity, and we use $r = 400$, $s = 256$ and $t = 10$ for both kinds of architectures. In this way the cardinality of $\mathcal{D}_{\mathrm{t}}$ is 102 400 so that it occupies only 13% of the whole dataset.

As for the optimization step, in the case of b-adic vector representations we use and compare the performances of two different optimization algorithms: the Stochastic Gradient Descend (SGD) and Adam. It turns out that Adam provides more accurate results than SGD, therefore in the case of the Prime Grid vector representation we make use solely of this optimizer.

Since the task of the training process is a multiclass classification problem, the function to be optimized will be the Cross Entropy loss function. The number of classes to be distinguished is clearly the modulo, m, and therefore this will be the dimension of the output space in all the schemes (4), (5), (6) and (7).

The details of the implementation of both the optimization algorithms, SGD and Adam, and of the Cross Entropy loss function can be found in Paszke et al. (2019).

3. Results

After the training process, the trained model is validated on the validation set $\mathcal{D}_{\mathrm{v}}$. The results of these validations are analyzed by calculating the confusion matrix and the accuracy (cf. Pedregosa et al. (2011) for the implementation). The results are collected in Tables 2, 3 and 4.

4. Conclusions

A quick glance to Tables 2 and 3 shows that the problem of classifying the congruence classes modulo m is fully resolved when the input data is processed using the b-adic vector representation with $b = m$; a fact appearing not that remarkable since in this case the rightmost entry of the representing vector is precisely m and therefore the answer to the proposed task is furnished by the input vector representation itself. This seems to be essentially the sole situation that the network is able to handle when the MLP architecture and the SGD optimizer are used. Indeed, from Table 2 we see that the only cases in which the network does not make completely random choices are those in which b divides m, that is, in which we have $m = bc$ for some $c \in \mathbb{N}$. More specifically analyzing the confusion matrices we observe that if b and c are powers of the same prime then the task is solved, otherwise the network correctly assigns to each number in the validation set its residue modulo b, but it makes random choices when allocating the residues modulo c. As an example, we show the confusion matrices of the cases $b = 2$, $m = 6$ and $b = 3$, $m = 6$.

$$\begin{pmatrix} 21594 & 0 & 12193 & 0 & 21268 & 0 \\ 0 & 9767 & 0 & 28734 & 0 & 16694 \\ 21661 & 0 & 12099 & 0 & 21176 & 0 \\ 0 & 9928 & 0 & 28434 & 0 & 16833 \\ 21681 & 0 & 11958 & 0 & 21170 & 0 \\ 0 & 9619 & 0 & 28624 & 0 & 16567 \end{pmatrix}$$

Confusion Matrix $b = 2$, $m = 6$.

$$\begin{pmatrix} 19750 & 0 & 0 & 35305 & 0 & 0 \\ 0 & 18723 & 5 & 0 & 36467 & 0 \\ 0 & 0 & 45754 & 0 & 0 & 9182 \\ 20225 & 0 & 0 & 34970 & 0 & 0 \\ 0 & 18258 & 5 & 0 & 36546 & 0 \\ 0 & 0 & 45694 & 0 & 0 & 9116 \end{pmatrix}$$

Confusion Matrix $b = 3$, $m = 6$.

The situation slightly improves when using the CNN architecture and the SGD optimizer. In particular, in this case the network is also able to

Table 2. Accuracies of the MLP architecture using the b-adic input representation.

Rep.	Optimizer	Mod 2	Mod 3	Mod 4	Mod 5	Mod 6	Mod 7	Mod 8	Mod 9	Mod 10
2-adic	Adam	1.00	1.00	1.00	0.95	1.00	1.00	1.00	0.99	0.98
	SGD	1.00	0.33	1.00	0.20	0.33	0.14	1.00	0.11	0.20
3-adic	Adam	1.00	1.00	1.00	0.82	1.00	0.73	0.70	1.00	1.00
	SGD	0.50	1.00	0.25	0.20	0.50	0.14	0.12	1.00	0.10
4-adic	Adam	1.00	0.83	1.00	0.98	0.33	0.24	1.00	0.15	0.78
	SGD	0.50	0.33	1.00	0.20	0.17	0.14	0.60	0.11	0.10
5-adic	Adam	0.50	0.36	0.46	1.00	0.35	0.92	0.37	0.92	0.50
	SGD	0.50	0.33	0.25	1.00	0.17	0.14	0.13	0.11	0.50
6-adic	Adam	1.00	1.00	1.00	0.51	1.00	0.99	0.92	1.00	0.20
	SGD	0.67	0.50	0.29	0.20	0.99	0.14	0.14	0.16	0.12
7-adic	Adam	0.50	0.33	0.91	0.78	0.30	1.00	0.99	0.16	0.25
	SGD	0.50	0.33	0.25	0.20	0.17	0.98	0.12	0.11	0.10
8-adic	Adam	1.00	0.33	1.00	0.85	0.33	0.31	1.00	0.98	0.20
	SGD	0.51	0.33	0.38	0.20	0.18	0.14	1.00	0.12	0.12
9-adic	Adam	0.50	1.00	0.26	0.93	0.50	0.14	0.59	1.00	0.99
	SGD	0.50	0.54	0.25	0.20	0.20	0.14	0.13	1.00	0.10
10-adic	Adam	0.80	0.54	0.40	1.00	0.33	0.16	0.30	0.76	1.00
	SGD	0.59	0.33	0.29	0.20	0.19	0.14	0.14	0.11	1.00

Table 3. Accuracies of the CNN architecture ($B = 8$) using the b-adic input representation.

Rep.	Optimizer	Mod 2	Mod 3	Mod 4	Mod 5	Mod 6	Mod 7	Mod 8	Mod 9	Mod 10
2-adic	Adam	1.00	0.36	1.00	0.22	0.34	0.13	1.00	0.12	0.18
	SGD	1.00	0.33	1.00	0.20	0.36	0.15	1.00	0.10	0.20
3-adic	Adam	0.50	1.00	0.26	0.22	0.50	0.13	0.14	1.00	0.10
	SGD	0.50	1.00	0.26	0.20	0.50	0.15	0.12	1.00	0.11
4-adic	Adam	1.00	0.35	1.00	0.22	0.34	0.13	1.00	0.11	0.10
	SGD	1.00	0.36	1.00	0.20	0.33	0.14	1.00	0.12	0.20
5-adic	Adam	0.50	0.35	0.25	1.00	0.17	0.13	0.10	0.13	0.47
	SGD	0.50	0.35	0.24	1.00	0.17	0.13	0.12	0.10	0.50
6-adic	Adam	1.00	1.00	0.60	0.22	1.00	0.15	0.39	0.15	0.16
	SGD	1.00	1.00	0.71	0.16	1.00	0.13	0.33	0.54	0.19
7-adic	Adam	0.50	0.36	0.26	0.22	0.17	1.00	0.13	0.13	0.10
	SGD	0.51	0.33	0.24	0.19	0.13	1.00	0.14	0.13	0.10
8-adic	Adam	1.00	0.35	0.42	0.22	0.34	0.13	1.00	0.12	0.18
	SGD	1.00	0.34	1.00	0.20	0.35	0.13	1.00	0.12	0.22
9-adic	Adam	0.50	0.36	0.26	0.22	0.50	0.13	0.13	1.00	0.10
	SGD	0.48	1.00	0.24	0.20	0.52	0.13	0.12	1.00	0.11
10-adic	Adam	0.91	0.36	0.54	1.00	0.17	0.13	0.14	0.13	1.00
	SGD	0.93	0.32	0.53	1.00	0.32	0.14	0.26	0.11	1.00

Table 4. Accuracies of the MLP and CNN architectures using the Prime Grid input representation and the Adam optimezer.

Arch.	Mod 2	Mod 3	Mod 4	Mod 5	Mod 6	Mod 7	Mod 8	Mod 9	Mod 10
MLP	1.00	0.76	0.94	0.46	0.86	0.44	0.59	0.37	0.55
CNN ($B = 8$)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

$$\begin{pmatrix} 133820 & 0 & 0 & 130 & 0 & 0 & 0 & 0 & 0 \\ 6 & 21410 & 9104 & 3 & 6080 & 29722 & 7 & 33467 & 5547 \\ 2 & 9253 & 32568 & 5 & 9294 & 18320 & 15 & 25410 & 10665 \\ 137 & 2 & 32 & 84872 & 0 & 13 & 32427 & 15 & 4 \\ 0 & 10528 & 23954 & 2 & 18136 & 14165 & 10 & 29745 & 8329 \\ 1 & 9279 & 7062 & 0 & 2709 & 50930 & 12 & 27542 & 7792 \\ 127 & 7 & 22 & 61215 & 6 & 28 & 55588 & 19 & 3 \\ 5 & 13122 & 7410 & 1 & 4375 & 35692 & 12 & 38150 & 6578 \\ 0 & 7165 & 16846 & 0 & 6352 & 25402 & 10 & 32048 & 17291 \end{pmatrix}$$

Fig. 3. Confusion Matrix, Prime Grid, $m = 9$.

identify the congruence classes modulo m with a b-adic vector representation of the input data if m divides b (cf. Table 3).

Tables 2 and 3 also show that the Adam optimizer generally behaves better than the SGD optimizer; there are, however, a few exceptions when it is used in combination with the CNN architecture. It is perhaps worth mentioning that Adam is usually more unstable than SGD: during the training process the loss function does not decrease monotonically, but it frequently jumps up and down making sometimes the convergence of the network more difficult (cf. Figure 4). For this reason all the experiments conducted using this optimizer train the model for t epochs, but then validate it in correspondence of the one that presents the minimum value for the loss function.

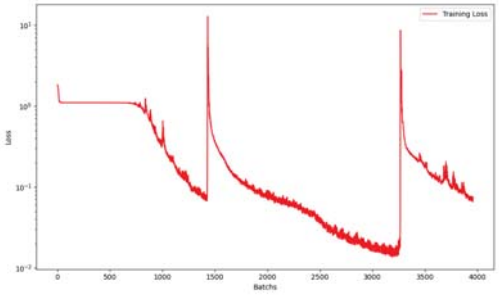

Fig. 4. Adam optimizer behavior, $b = 2$, $m = 6$.

As for the Prime Grid vector representation (cf. Table 4), when the adopted architecture is the MLP we observe that the accuracies never reach very high values, still the choices that the network makes are not completely random. The analysis of the confusion matrices suggests the following pattern:

- If the modulo, m, is a power q^k of a prime q, then the classes are split into groups according to the exponents of q: for $i = 0, 1, \ldots, k$, group i contains all the classes that are divisible by q^i but not by q^{i+1}. The classes belonging to the same group are confused with one another. As an example, see the confusion matrix in Figure 3 where we take $m = 9 = 3^2$.
- If the modulo, m, is of the form $2q$ where q is an odd prime, then class $[0]_m$ and class $[q]_m$ are identified correctly, while the others are split into two groups, those with an odd residue and those with an even residue. The classes belonging to the same group are confused with one another. As an example, see the confusion matrix of the case $m = 6$ below.

$$\begin{pmatrix} 194736 & 0 & 48 & 12 & 21 & 0 \\ 0 & 127476 & 1 & 0 & 0 & 19632 \\ 14 & 0 & 107181 & 7 & 61600 & 0 \\ 12 & 14 & 4 & 172848 & 1 & 40 \\ 9 & 6 & 33259 & 0 & 135450 & 0 \\ 0 & 18789 & 0 & 0 & 0 & 128840 \end{pmatrix}$$

Confusion Matrix, Prime Grid, $m = 6$.

The most remarkable situation occurs, however, when the input is processed using the Prime Grid vector representation and the architecture of the network is the CNN (cf. Table 4). In this case, in fact, the network is able to identify correctly all the congruence classes modulo m, for all $m = 2, 3, \ldots, 10$.

The reason underlying this particularly nice behavior appears to be the following. In contrast to other kinds of input data vector representations (e.g. the b-adic ones), the Prime Grid vector representation is able to endow the model

with the multiplicative structure of natural numbers. This, however, is not sufficient to produce a network capable of identifying the congruence classes modulo some $m \in \mathbb{N}$ (as the first row in Table 4 shows) since the resolution of this task requires the comparison of adjacent numbers and therefore the possibility of analyzing the additive structure of the set $\mathbb{N}$. In order to overcome this issue, the choice of the convolutional architecture has been successful: indeed the input organized in sequences of B consecutive numbers and the kernels that move along these sequences allow the network to learn such an additive structure and to extract features that globally relate all the numbers in the sequence rather than considering them individually.

It is now evident that the length B of the sequence of numbers must play a fundamental role in the subject. Indeed, if B is too small in comparison to m, then the network is expected to exhibit issues in classifying the congruence classes modulo m. A complete investigation of this kind of phenomena goes beyond the scope of this paper, and it constitutes an interesting subject for further analysis.

Acknowledgement

This research project has been supported by a Marie Skłodowska-Curie Innovative Training Network Fellowship of the European Commission's Horizon 2020 Programme under contract number 955901 CISC.

References

Abdal, R., Y. Qin, and P. Wonka (2019). Image2stylegan: How to embed images into the stylegan latent space? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4432–4441.

Arvanitidis, G., L. K. Hansen, and S. Hauberg (2017). Latent space oddity: on the curvature of deep generative models. *arXiv preprint arXiv:1710.11379*.

Connor, M. and C. Rozell (2020). Representing closed transformation paths in encoded network latent space. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Volume 34, pp. 3666–3675.

de Rosa, G. H. and J. P. Papa (2021). A survey on text generation using generative adversarial networks. *Pattern Recognition 119*, 108098.

Donoho, D. L. and C. Grimes (2005). Image manifolds which are isometric to euclidean space. *Journal of mathematical imaging and vision 23*(1), 5–24.

Fetty, L., M. Bylund, P. Kuess, G. Heilemann, T. Nyholm, D. Georg, and T. Löfstedt (2020). Latent space manipulation for high-resolution medical image synthesis via the stylegan. *Zeitschrift für Medizinische Physik 30*(4), 305–314.

Gatt, A. and E. Krahmer (2018). Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research 61*, 65–170.

Giardina, A., S. S. Paria, and A. Kaustubh (2022). A naive method to discover directions in the stylegan2 latent space. *arXiv preprint arXiv:2203.10373*.

Hu, X., P. Ma, Z. Mai, S. Peng, Z. Yang, and L. Wang (2019). Face hallucination from low quality images using definition-scalable inference. *Pattern Recognition 94*, 110–121.

Kolossváry, I. B. and I. T. Kolossváry (2022). Distance between natural numbers based on their prime signature. *Journal of Number Theory 234*, 120–139.

Li, D., C. Du, and H. He (2020). Semi-supervised cross-modal image generation with generative adversarial networks. *Pattern Recognition 100*, 107085.

Liu, L., H. Zhang, X. Xu, Z. Zhang, and S. Yan (2019). Collocating clothes with generative adversarial networks cosupervised by categories and attributes: a multidiscriminator framework. *IEEE transactions on neural networks and learning systems 31*(9), 3540–3554.

O'Shea, K. and R. Nash (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

Papastratis, I., K. Dimitropoulos, D. Konstantinidis, and P. Daras (2020). Continuous sign language recognition through cross-modal alignment of video and text embeddings in a joint-latent space. *IEEE Access 8*, 91170–91180.

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12*, 2825–2830.

Richardson, E., Y. Alaluf, O. Patashnik, Y. Nitzan, Y. Azar, S. Shapiro, and D. Cohen-Or (2021). Encoding in style: a stylegan encoder for image-to-image

translation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2287–2296.

Sarvamangala, D. and R. V. Kulkarni (2022). Convolutional neural networks in medical image understanding: a survey. *Evolutionary intelligence 15*(1), 1–22.

Shen, Y., C. Yang, X. Tang, and B. Zhou (2020). Interfacegan: Interpreting the disentangled face representation learned by gans. *IEEE Transactions on Pattern Analysis and Machine Intelligence 44*, 2004–2018.

Smith, A. L., D. M. Asta, and C. A. Calder (2019). The geometry of continuous latent space models for network data. *Statistical science: a review journal of the Institute of Mathematical Statistics 34*(3), 428.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, Ł. Kaiser, and I. Polosukhin (2007). Attention is all you need. in advances in neural information processing systems. In *Proc. NIPS*, pp. 5998–6008.