Neural Networks Compression

Student CRISTIAN MANDRAGIU
Student DOINA MARTA
Student MANUELA LUMEZEANU
Student VLADIMIR SUSOIU
Faculty of Economic Cybernetics, Statistics and Informatics,
Academy of Economic Studies

1.Introduction

1.1. What are Neural Networks?

Since the beginning of Artificial Intelligence researches two opposite directions have appeared in this domain, directions that even now are the dominant models (paradigms) in Artificial Intelligence.

The *logical-symbolic paradigm* implies mechanisms for symbolic representations of the knowledge and use for different logical models for deducting new ones from the system's knowledge base memorized facts. For example, chess playing applications, and intelligent applications (called Expert Systems) that can solve great complexity problems in a well-defined domain, represent great achievements within this paradigm.

The *connexionist paradigm* introduced a new computing concept- neural computation- and generated great achievements known as artificial neural network(or neural networks).

Within the neural networks the information is no longer kept in precise areas, like the case of using standard computers, but diffuse memorized all through the whole network. The memorization is achieved by setting corresponding values for the weights of the synaptic connections between the neurons of the network.

Another important element, which is the most responsible of the connectionists models' success, is the neural networks' capacity of learning from examples. Characteristic for neural networks is the capability of implicitly synthesizing a certain model of the problem, given a multitude of examples.

1.2. Neural network - definition

According to a DARPA study over the neural networks (1998), a neural network is a system formed by many simple, parallel operating, processing elements, whose function is determined by the network's structure, the weight of the processing elements' (nodes) ties.

According to S. Haykin(*Neural Networks: A Comprehensive Foundation*, 1994), a neural network is a computational unit, parallel distributed, that has a native capacity of storing knowledge resulted from previous experiments, making them accessible. It resembles the human body in two ways:

Knowledge are accumulated by the network via a learning process.

Inter-neural ties are used for storing the knowledge.

1.3. Neural Networks – brief history

The year 1943 marks the beginning of the neural computation by the appearance of the first neuron model, resulted from the co-operation between neuro-physiologist W.S. McCulloch and the mathematician W. Pitts.

We must also mention the 1949 model proposed by D.O. Hebb, the 1958 Frank Rosenblatt's paperwork on *Perceptron*, the ADALINE neural network model proposed by Bernard Widrow in the early '60, the 1986 issue of the *Parallel Distributed Processing, Explorations in the Microstructure of Cognition*, the climax being the 1987 first international neural networks conference.

The artificial neural networks have been inspired by the biological neural network model. They appeared from the desire of building artificial systems, capable of processing complex, even “intelligent” processes, similar to those the human brain realize.

Neural networks represent now a fascinating researching domain and a major intellectual and technological challenge, given the fact that they have modified our view over the calculus processes and algorithmics aspects of the Artificial Intelligence.

1.4. Applications for neural networks

The multi-disciplinarity of the neural calculus theory has repercussions upon its area of applications, from engineering, informatics, medicine, economy, to humanist domains like sociology, linguistics or psychology. Applications of the neural networks appear even in domains like agriculture, forecasting, geography, physics, etc.

The importance and the impact over the humans of the neural networks, as basic modules of the next century’s intelligent machines, have been emphasized by L. Cooper in 1991: “*We’re not only going to learn how to live with those machines, but one day, we’ll be asking how could we manage living without them until then”*

1.5. Differences between Artificial Intelligence and Artificial Neural Networks

Some features make this model different from the traditional approaches regarding artificial intelligence. With neural networks:

By default, processing the information is parallel.
Knowledge is distributed through the whole system.
Artificial neural networks are tolerant with errors.

2. Types of Neural Networks

There are several types of neural networks, some of them in full development

-Networks based on supervised and non-supervised training

Supervised training

The network is given the both the input and output data, the trainer mentioning exactly what must be returned as the result.

The trainer can “tell” the network being in the learning phase how well it operates, or what the right behavior should be.

Self-organizing or non-supervised training

There is a training scheme by which the network is given only the input data. The network, realizing what some of the data set properties are, learns how to reflect these properties in the output data.

This kind of training gives a more plausible biological model of training. The properties the network learns how to recognize depend on its model and on the training method.

-Networks based on feedback and feedforward connections

Examples of networks:

Non-supervised networks:

Feedback networks:

Kohonen Networks
Discrete Bidirectional Associative Memory(BAM)

Feedforward-only networks

Fuzzy Associative Memory(FAM)
Learning Matrix(LM)

Supervised networks:

Feedback networks:

Boltzmann Machine(BM)
Fuzzy Cognitive Map(FCM)

Feedforward-only networks
Back Propagation(BP)
Probabilistic Neural Networks(PNN)

3. Using Neural Networks for Data Compression

3.1. Basics

Neural networks are used in compression as data filters. During the past 10 years there has been a tremendous growth of interest for the artificial neural networks domain.

Unlike nowadays computers, whose circuits operate in picoseconds, making complicated mathematical calculus, humans are not gifted with such capabilities, but in exchange being capable to fulfill certain tasks that no artificial intelligence system is capable to, at least, model.

This is where the idea of neural network came from, having as the main inspiration the human neural system.

The neural system organization unit is the *neuron,* a cell that presents a certain number of *dendrites* and an *axon* through which it connects to other neurons. The dendrites are the entries in the neural cell, and the axon the exit. The axon splits, having the exit connected to several neurons. The impulses at the entrance of the neuron can excite it and make it generate an impulse to the neurons it is connected to. We must mention that the ties between neurons are weighted and each neuron transforms the input impulse before transmitting it forward.

A neural network is made of multiple *nodes* that keep the artificial neurons, non-linear processing elements that operate in parallel.

The main features of the neural networks are:

Training capacity

- Neural networks don’t need powerful programs but most likely the results of the trainings over a set of given programs.
- Given a set of input data, maybe even the desired output, following the instruction process, the neural networks are self-organized and they solve the problems for which they have been built. There is a large category of training algorithms, each having its advantages and disadvantages, succeeding in some cases or failing in others.

Generalization capacity

-If they have been correctly trained, the networks are capable of giving right answers for input data different than the ones they have been trained with, as long as these input data are not substantially different. It is very important to mention that artificial neural networks generalize automatically, as a result of their structure, not with the help of human intelligence built in a certain program especially created.

Synthesizing capacity

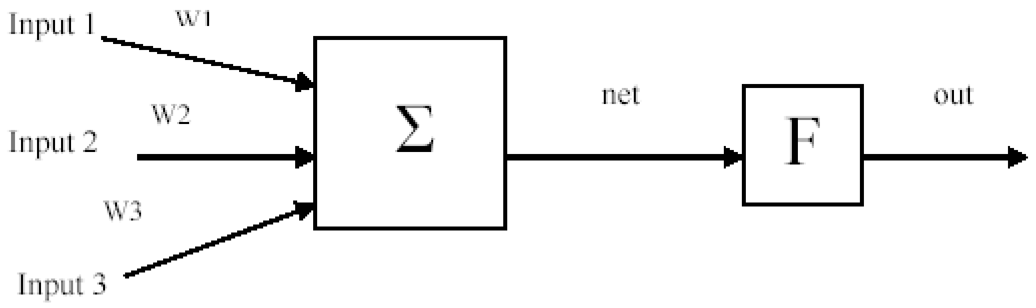
- Artificial neural networks can make decisions or jump to conclusions when confronted with complex information or partial or irrelevant results. For example, a network can be trained with a sequence of distortions of the letter “A”. After a process of training, the application of a distorted version of letter “A” will have as a result at the network’s exit the correct letter.

The artificial neuron is a much simplified copy of the biological neuron, formed by a body, a set of entries and an exit. Each entry is weighted, that means its value is multiplied by a corresponding value, called weight, then all the entries weighted are totalized.

There is an activation function applied on the resulting sum, that has as a result the value of the neuron’s exit.

A simplified scheme of the artificial neuron, where:

- input1, input2,input3 are the values of the entries;
- W1,W2,W3 are the weights that multiply each entry;is presented in the figure below:



net= $\Sigma \text{input}_i * W_i$
out=F(net)

We can observe that the model of the artificial neuron doesn't have a lot of the biological one's properties. We leave aside the delay time that affects the dynamics of the system, the entry determines an immediate exit value. We also leave aside the frequency modulation of the neuron and many more. However, neural networks can behave in certain cases like neural system.

3.2. Text compression

The present data compression algorithms are the object of over 50 years of researches in this domain. They started from primary data compression methods like eliminating unnecessary spaces.

One of the reasons of using neural networks for data compression is that they are used for recognizing very complex models. Standard data compression algorithms like Limpel-Ziv or PPM or Burrow-Wheeler are based on very simple models with n entries. They use the non-uniformity of the distribution of text sequences found in most data. As an example, the character combination that forms the Romanian word "pom" is much more common than the combination "psm". Therefore, the first is assigned a shorter code, given the greater probability of its appearance in a text.

3.3. Image compression

One example of image compression using two types of models: Kohonen and Grossberg networks, will have a three layer network. Each node from the entry layer is connected to the second layer (Kohonen layer) nodes, each Kohonen node is connected to each node in the third layer(Grossberg layer).

To compress and decompress images using neural networks we have to split the image in a finite number of sub-images. We assume that the original image has a n*x*n pixels dimension and it can be split in k>0 sub-images of m*x*m pixels. Each sub-image's pixels will be compressed with a neural network in m data bits and transmitted to reception. These will be decompressed with another neural network in subimages of m*x*m pixels. This way, there will be two neural networks developed for the compression and decompression, being named the compression and decompression network.

On the given example, the number of the neurons in the entry layer is m*x*m, which is compatible with the total number of one sub-image's pixels. The number of Grossberg nodes is m, that represents the total number of transmitted bits. The number of Kohonen nodes is equal to the total number of models used in the training process of the compression/decompression networks.

4.Text Compression/Decompression Algorithm

4.1. The algorithm - description

The header of an archive must have the format:

```
 Program a\r\n
 size name\r\n
          size name\r\n
          :
```

```
:
size name\r\n
\032\f\0
```

where :
"size" represents the initial number of bytes, stored as a 9 digit decimal number(with preceding spaces in case the number has less than 9 figures)

"name" represents the name of the file as typed(including the path if specified)

\r=carriage return, \n=linefeed, \f=formfeed, \032 is the EOF character in DOS and \0 is a NUL.

Function main()

The form of function main is *main(int argc, char** argv)*. The parameter *argc* keeps the number of arguments given in the command line and *argv* their contents.

If, from the number of arguments results there is no file name, there'll be a help-screen displayed. Otherwise the program will try to open the first file passed as an argument. This must be the archive for extract/create. If the file exists there'll be an attempt to extract the files from it.
First, there's a try to read the archive's header.
If the first line of the header does not contain the text *program_name* the message "Archive not in right format" will be displayed.
If a certain line is not in the right format the following message will be displayed: "Archive header not in right format".
If these errors don't occur the files' names and sizes will be stored in two vectors: *filename* and *filesize*.

There will be created an object of type Encoder, with the working mode DECOMPESS.

The *filenames* vector is being read and for each file there will be read and uncompressed each character, until total uncompression. If in the extraction directory there already is a file having the name of the one currently extracted, it will not be overwritten.

If the filename in the command line doesn't exist this will be the name for the new created archive. If, besides the name of the archive there is no other parameter, the program will prompt for the names of the files to be compressed until the user enters an empty line or EOF. In the two vectors *filename* and *filesize* there will be stored the names and sizes for the files passed as parameters(if any).

Next, the header of the archive is created, followed by the compression process.

Like in the decompression process, an object *e* of type Encoder will be created, with the working mode COMPRESS.

The *filename* vector is read and for each file there will be read and compressed each character until complete compression.

The archive's header will be followed by data compressed in binary format. The compressed file will be concatenated and treated as a one data stream (The correct extraction will be made based on the initial dimension read in the header).

There will be presented in parallel the compression/decompression algorithms:

For compression/decompression there will be used the following independent functions, given the fact that the function *main* created an object *e* of type Encoder

```
// Reads a byte from encoder e
int decompresie(Encoder& e) {
            int c=0;
            for (int i=0; i<8; ++i)
 c=(c<<1)+e.encode();//Each returned bit by function encode() will be
                      //added to c(building the next byte to be extracted
            return c//Returns the reconstructed byte
}
```

// Writes a byte *c* to encoder *e*

```
void compresie(Encoder& e, int c) {
for (int i=0; i<8; ++i) {  // Unpacks the 8 bits and sends them to the function
                                        //encode to compress them
e.encode((c&128)?1:0);
            c<<=1;
            }
}
```

Data are compressed using an arithmetic encoder(using the function e.encode(…)) that predicts a bit stream using a neural network, with a predictor. The predictor estimates the probabilities P(0)+P(1)=1 for each bit, considering the previous bits. The arithmetic encoder begins with a [0,1) range, initially scaled by 2^{32} ([0, 2^{32}]) for giving the right predictions and divides it into two sub-ranges, with the sizes proportional to P(0) and P(1). X1 and x2, fields of object *e* (type Encoder) will be the lowest and highest value of the initial range. For dividing this range into two subranges we use xmid=x1+p*(x2-x1). p is calculated by function p in class predictor, as the probability of the next bit to be 0, probability scaled by 1K(1024). There will be 5 ways to adjust p, given the fact that in function p x2 and x1 are continuously shifted with 8 bits to the left(multiply by 2^8). These adjustments are made to avoid the overflow situations.

```
if (xdiff>=0x10000000) xmid+=(xdiff>>16)*p;
  else if (xdiff>=0x1000000) xmid+=((xdiff>>12)*p)>>4;
            else if (xdiff>=0x100000) xmid+=((xdiff>>8)*p)>>8;
                      else if (xdiff>=0x10000) xmid+=((xdiff>>4)*p)>>12;
                      else xmid+=(xdiff*p)>>16;
```

If the working mode is COMPRESS and the bit to be prepared is 1 we'll consider range [x1,x2] with new x1=xmid+1.

If the bit is 0 we'll consider the range [x1,x2] with x2=xmid.

For DECOMPRESS, if we are in the prediction range [x1,xmid] the predicted bit will be 0, otherwise it will be 1.

```
            if (mode==COMPRESS) {
            if (y)
                      x1=xmid+1;
            else
                      x2=xmid;
            }
            else {
                      if (x<=xmid) {
                                          y=0;
                                          x2=xmid;
                                          }
                      else {
                                y=1;
                                x1=xmid+1;
                      }
            }
```

Considering the bits already processed an update of the inputs(synapses) of the neural network will be made(for correct future predictions). That's what function*predictor.update(y)* makes.

For giving a prediction we set the prediction error to E=Y-P(Y).

For each entry in the neural network, the number of bits 0 and 1, expressed by c0 and c1, will be halved in case one of these number is greater than 250, to avoid the overflow situation in the 8 bits representation.

For each synapse there will be calculated the weight using the formula Wi=Wi+(RS+RL/sigma2)*Xi*E where RS is the short term learning rate and the other parameters have the signification presented above.

```
w.w+=(error*(RS+sigma2(w.c01)))>>16;
```

The predictor estimates, in case the function encode that called it finished to read a character, to encode parameters s7,s3,sw0 and sw1 with the given signification.

Using these fields there can be established the next entries x[i] i=0,5 by calculating the remaining of the division to prime numbers less than $2^{22}-2^{16}-2^{8}$.

```
x[0]=4128768+(s3&0xffff);
          x[1]=(s3&0xffffff)%4128511;
          x[2]=s3%4128493;
x[3]=(s3+s7*0x3000000)%4128451;            x[4]=(sw1+sw0*29)%4128401;
          x[5]=sw0%4128409;
```

s3,s7 are the last 7 complete bytes of input: 7-4, 3-1
sw0, sw1 are the hash of current and previous words

Then, the sum of the contexts' weights Wi, i=0,5, will be calculated. The sum is used to calculate the next bit to appear

P(Y) = 1/(1 + e^(-SUM(i) Wi * x[i]))

Coming back to function encode, when x1 and x2, x1<x<x2(x=number representing the last 4 read bytes from the archive) will be sufficiently close, in the archive there will be written a byte depending on these 4 read bytes.

The written byte will be the first group of 8 bits from x2 that superposes x1. As mentioned, x2 and x1 will be shifted left with 8 bits, explaining the 5 ways for adjusting xmid's value.

When uncompressing, x continuously changes to packed memorize the last 4 read bytes from the archive. With the next step, by comparing x with xmid(x1<=xmid<=x2) there will be given the next Y bit to be returned.

The returned Y bits will be packed in groups of 8 and the resulting character will be written in the uncompressed file on the disc.

4.2. The program

The program compresses and decompresses ASCII and binary files. In function main, considering the parameters given in the command line, there will be executed the compressing/decompressing module.

The program uses the model of a neural network with 6 entries(contexts) X_i, i=0,5 and one exit, the latter being the next bit predicted with a prediction mechanism based on the 6 entry contexts.

The program uses the classes

G(used to calculate the function g(x)=1/(1+exp(x)) in the 16 bits arithmetic. This function will be used later to calculate the probability for the next bit to be 0 or 1.

Sigma2 calculates the expression
RL/s^2 where s^2=(c0+d)*(c1+d)/(c0+c1+2d)
where:

RL- the short term learning rate. It has been hand-tuned to 0.38. Modifying this rate will modify the efficiency of the compression algorithm.

c0 and c1 represent the number of bits 0 and 1 in context X_i

d is a parameter used to avoid division by 0 when calculating s^2. It has been set to 0.5

Predictor, that implements the model of the neural network. In this class there is defined the structure of a synapse:

```
struct Wt {
short w;       //The synapse(context) weight*1K(
                //scaled by 1K for correct calculus)
          union {
```

```
      struct C {unsigned char c0, c1;} c;  // Number of bits 0, 1
      unsigned short c01;  //  Concatenated counts for sigma2 function
    };
  };
```

This class also defines:
The wt matrix of NX=6 synapses
fields
s0= last 0-7 bits of current byte with leading 1 (1-255)
s7=first 4 bytes from last 7 analyzed
s3= last 3 bytes from last 7 analyzed
sw0 and sw1= hash of current and previous words
pr=next prediction
the constructor Predictor that initializes the above fields. Constructor takes no parameter.

The methods:

p that returns an int representing the value of field pr. p takes no parameter.
update that trains the network, by setting the next 6 contexts(setting the synapses' information) depending on parameter y(type int), representing the value of the bit analyzed when function *update* ran.

Encoder that encodes the characters read from the files to be compressed.

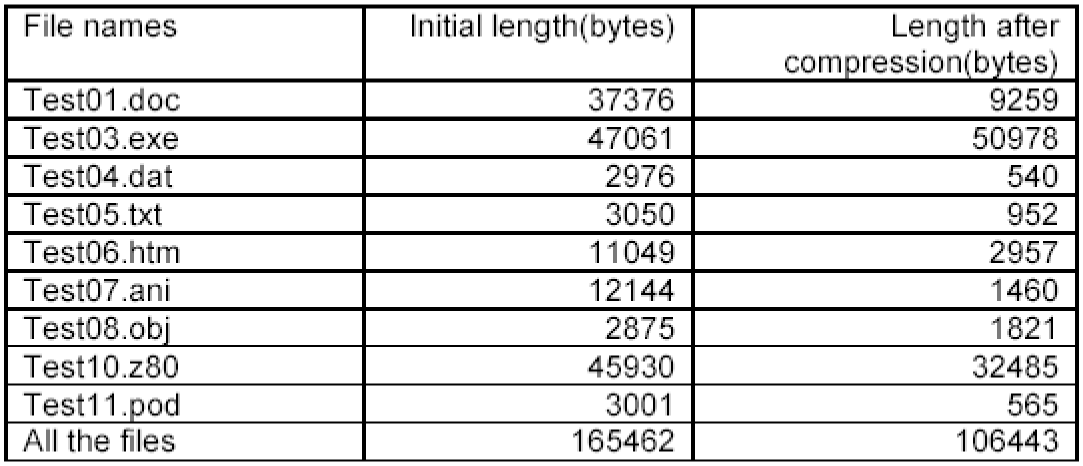
The class also contains the following fields:
mode that takes one of the following values: COMPRESS or DECOMPRESS
archive – a pointer to the archive file.
x1 and x2, the upper an lower value of the range used for encoding. These values are initialized with 0 and 0xffffffff, the range [x1,x2] being, in fact, the range [0,1] scaled by 2^{32}.
x that represents the last 4 bytes of input from the built archive.
eofs represents the number of EOF symbols read from the archive, used to stop the decompress process with the message "Unexpected end of archive" in case, from the beginning, couldn't be read at least 4 bytes from the archive(the archive has been damaged).
xchars – the number of bytes of compressed data.
encodes – the number of uncompressed data bits.

The methods for class Encoder are:

Constructor Encoder that initializes the fields presented and tries to read the first 4 bytes from the archive. Constructor takes as parameters *mode*(COMPRESS or DECOMPRESS) and*archive*.
encode – that does the encoding. Takes as parameter y, the bit to be analyzed, processes it, and the result is written in the archive. For the DECOMPRESS mode it returns the bit resulted from the uncompressing algorithm.
print() used to display the time of data compression. It uses the xchars and encodes fields.

Destructor ~Encoder used to write in the archive the remaining bytes of x, with x1<x<x2.

4.3. The results



File names	Initial length(bytes)	Length after compression(bytes)
Test01.doc	37376	9259
Test03.exe	47061	50978
Test04.dat	2976	540
Test05.txt	3050	952
Test06.htm	11049	2957
Test07.ani	12144	1460
Test08.obj	2875	1821
Test10.z80	45930	32485
Test11.pod	3001	565
All the files	165462	106443

5.Conclusions

It has been proven that it is practical the use of neural network in applications that need higher speed. From all the models of neural networks, the best combines long and short term learning rates, for a equilibrium between the inputs, favoring the newest ones.

The neural networks' capacity of solving complex practical problems using a set(sometimes limited) of examples, gives them a very large applicability potential. The applications spectrum goes from characters recognizing systems, signatures, talk, up to automatic pilots and complex processes control systems. This spectrum is continuously enlarging and, for the future, we can say that the connectionist paradigm will increasingly get the Artificial Intelligence's scientists' interest.

No matter how much the storage devices develop, their capacity will never be enough for implementing software applications. The compression/decompression problem appears more important when talking about global computer networks. The actuality of this problem as software research and development subject is emphasized by the existence of annual conferences (Data Compression Conference), by the market releasing of compression products and the attempt to set new standards for file information storage according to data compression rate growth.

The research continues and there are new, and more powerful, compression methods expected.

6.References

[MAHO01] http://mahoney4.home.netcom.com/compression/

[Ivan98] Ion Ivan , Daniel Vernis
Compresia de date
Editura Cison
Bucuresti 1998

[Bode97] Constanta Bodea
Parallel Algorithms for Neural Networks
ASE Bucuresti 1997

[Dumi96] D.Dumitrescu , H.Costin
Retele neuronale.Teorie si aplicatii
Editura Teora
Bucuresti 1996

[Bell90] Bell T.C.,Cleary J.G.,Witten I.N.
Text Compression
Prentice Hall Englewood Cliffs, N.J 1990

Bibliography

1. Ion Ivan, Daniel Vernis - Compresie de date, Editura Cison, Bucuresti, 1998
2. www.pcreport.ro/pcrep35/huffman.html