Why Neural Networks And Deep Learning Are The Future In Machine Learning

Daniel Ulinic (1362218)

Abstract

Basat en el cervell humà, el desconegut més gran de la ciència fins a l'actualitat, les xarxes neuronals profundes han estat recentment en el radar de la majoria dels investigadors. El seu interès es basa en fets. Algorismes d'inspiració biològica que repliquen el funcionament del sistema neuronal humà capaços d'aprendre mentre estan sota la supervisió correcta i amb les configuracions adequades. Avui en dia, l'aprenentatge és un atribut associat directament a l'ésser humà. Ser capaç de fer-ho denota intel·ligència. Per tant, l'interès sobtat de passar aquestes característiques a les màquines. L'estat de l'art conclou amb molts exemples com ara màquines capaces de mantenir una conversa adequada, guanyar videojocs, pintar imatges fins a cotxes autosuficients. Com un cervell, el disseny de la xarxa neuronal es basa en capes de neurones artificials connectades enviant senyals quan s'activen. En conseqüència, l'aprenentatge profund és un conjunt de tècniques potents per desencadenar les neurones correctes i l'aprenentatge en xarxes neuronals. El propòsit d'aquest article és dissenyar un algoritme intel·ligent capaç de distingir dígits manuscrits mitjançant el conjunt de dades reunit de una base de dades de codi obert des d'on pot entrenar i aprendre.

Index Terms

Xarxes Neuronals, Deep Learning, CNNs, Aprenentatge Automàtic, Python, Intel·ligència Artificial, TensorFlow, TensorBoard.

Abstract

Based on human brain, the biggest unknown to science to this day, deep neural networks have recently been on the radar of most researchers. Their interest is based on facts. Biologically-inspired algorithms replicating the functioning of the human neural system whom are capable of learning while under the correct supervision and with the right adjustments. Nowadays learning is an attribute directly associated to mankind. Being capable of doing so denotes intelligence. Therefore the sudden interest on passing on this features to machines. The state of the art concludes with many examples such as machines able of maintaining a proper conversation, beating video games, painting pictures to self driving cars. Like a brain, neural network's design relies on layers of artificial neurons connected and sending signals when triggered. Accordingly, deep learning is a set of powerful techniques for triggering the right neurons and learning in neural networks. The purpose of this paper is designing an intelligent algorithm capable of distinguishing handwritten digits using datasets from an open source database from where it can train and learn.

Index Terms

Neural Network, Deep Learning, CNNs, Machine Learning, Python, Artificial Intelligence, TensorFlow, TensorBoard.

✦

I. INTRODUCTION

NEURAL Networks (NNs) [1] and Deep Learning are supervised, machine learning methods, which can also be qualitative. By supervised, one understands the algorithms are guided by class-labeled data in order to achieve specific pre-determined results. The learning process, therefore qualitative rather than quantitative is carried out by specifying labels as the correct outcomes.

The objective of this study is to train a deep learning algorithm to differentiate between image inputs of handwritten digits from 1 to 10 while accomplishing the implementation of functional Neural Networks using the well-known programming language, Python, thus reaching the highest possible level of accuracy comparable to the actual state of the art results. Meaning, if not as high performance results at least overpassing a threshold of 90.00% of accuracy. The deep learning process will consist in training, learning and validating the dataset collection of 60,000 thousand images representing handwritten digits gathered from the open source MNIST Database [2]. The images from the database are to be distinguished between the training dataset and the test dataset, 50,000 and 10,000 respectively.

The first approach consisted in implementing a standard single layer neural network serving as basis of the study where most of the machine learning concepts were introduced due to the low level of complexity that features and better understanding for future developments. In order to improve the accuracy, new techniques were used or the entire network structure was changed leading to new models.

This paper is structured as follows. Section I, a brief introduction and the goals to be achieved. Section II will focus on the actual state of the art regarding neural networks which will conduct to a later comparison with the work achieved within

• Contact Email: dani.ulinic@gmail.com
• Bachelor's Degree in Computer Engineering (Specialization in Information Technologies
• Tutor: Jordi Casas Roma (dEIC)
• Year 2017/18

the present article. Section III will elaborate the tools and techniques used whereas Section IV will focus on designing a basic single layer Neural Network (NN) with Python as a first contact towards more complex deployments. Section V will evaluate deep learning studies and features extraction techniques and choose suitable ones. Finally, sections VI, VII and VIII will be dedicated to discussion over the results, future work on this topic and conclusions, respectively.

II. State of the Art

The human brain is a highly complex, nonlinear, and parallel computer. It is capable of performing certain computations such as pattern recognition, motor control, etc. many times faster than the fastest digital computer in the world. Its structure consists of many layers of neurons connected by synapses. Synapses are capable of adapting, and that is what most of learning is, by changing their effectiveness. This process starts when neurons receive an input and each input line is controlled by a synaptic weight which can be either positive or negative. The synaptic weights adapt so that the entire network learns. Hence humans can recognize objects, understand language or control their bodies.

A neural network does just that. Its distributed structure and its ability to train and learn allows generalization, the throughput of reasonable outputs for new inputs not encountered during training.

The benefits of neural networks also intended to test through this paper are **nonlinearity**, good **mapping** response from inputs to outputs, **adaptivity**, **evidential response** and **contextual information**.

The first Neural Networks appeared in the 1950's. Their structure was simple and only consisted of one hidden layer. At that time they were not given much use or importance since outputs did not seem accurate when the dataset escalated in size. Figure 1 represents the schematic of a basic neural network.

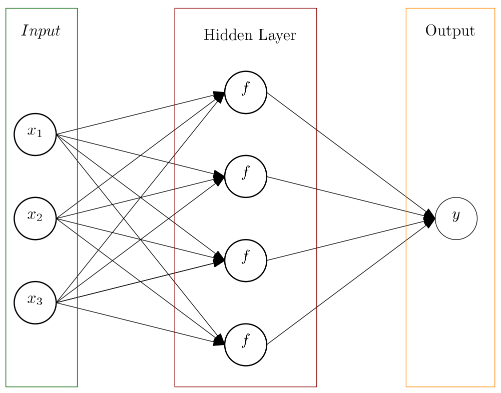


Fig. 1: Artificial Neural Network scheme model

It was not until a few years ago the interest of researchers was regained. By adding more hidden layers to the model outstanding results were obtained which led to self-driving cars or face and speech recognition. Those neural networks with a high number of layers (two or more) are known as "deep neural networks" and are characterised by their learning capabilities.

Around the year 2000, many pattern recognition algorithms started to appear but it was not until the Deep Convolutional Neural Networks (CNNs) that the movement was considered revolutionary. In 2003 [3], the benchmark was set to a 0.40% of loss using the MNIST Database. In 2006 [4] CNN based on Graphics Processing Unit (GPU) obtained a 0.39% loss with the same MNIST database.

Later in 2010, a new MNIST Database record was established with a 0.35% loss made possible not by a CNN, but rather by an ensemble GPU only implementation of the Back Propagation (BP) algorithm, 50 times faster than any Central Processing Unit (CPU) model version. Taking into account the fact that these results were accomplished with a 3 to 5 decades old algorithm such as BP, the performances are worth mentioning. Case in point, this led the investigators to think that it should be better to go on improving hardware performances rather than the model itself. The following years were focused in testing and comparing algorithms and actual human beings. In 2012, human-competitive performances or better-than-human recognition rate as stated in [5] was achieved with accuracy, 99.46%. The model used was a *multi-column deep convolutional neural network*.

Therefore the goal for this study is reaching at least a 90.00% level of accuracy over the test dataset in order to have a reasonably comparable outcome with the state of the art results mentioned previously. In order to do this as explained in Section I, a single layer neural network was the basis used to introduce machine learning concepts and serve for future more complex neural network developments.

III. Methodology

Implementation will be based on using Tensor Flow [6], CPU version, Python's newest open source framework designed by Google for dealing with the insurgent large-scale machine learning movement as reviewed in [7]. The main feature of Tensor Flow are the front-end functions that give transparency to lower-level programming. Since computations can become very complex and confusing, to make it easier to understand and debug, Tensor Board is also available, which is a set of visualization tools. The code implemented in Python is reshaped for suitability and maximization of Tensor Board use.

Modeling the neural network will be the first approach. The main concepts are inputs, weights, bias, learning rate, neuron hidden layers, activation functions, loss function, labels, accuracy and outputs.

Each input will be given a number of nodes to start with and the hidden layers are going to be defined. The essentials of deep learning are the so-called techniques "**activation functions**". Each neuron in the layer can be considered as one activation function and the inputs going through it, depending on which one it is, will affect the output in one way or another.

Therefore, each **hidden layer** is a set of neurons which all have the same activation function. If a second hidden layer would have another activation function defined, all neurons of the respective layer would implement it. Their purpose is focused on feature extraction for more accurate performances by sampling the inputs to the point that unnecessary details will not interfere with prediction. In this study the activation functions are non-linear, hence it tests the non-linearity of each and every network model.

The **inputs** are declared as a unique *placeholder* of a single flattened 28 by 28 pixel MNIST images which are to be processed by the several activation functions for as many hidden layers present in each neural network model.

The **weights** are real numbers expressing the importance of the respective input to the output in vector type data. The neuron's output is determined by whether the weighted sum $\sum_i = w_i x_i$ of all of the inputs plus a so-called constant "bias", passing afterwards through the activation function, gives a neuron more weight and to be the one to output its result rather then the other neurons.

It is a good practice initializing the weights randomly and change afterwards with the optimization functions such as Back Propagation (BP) algorithm until adapting the learning process for generalization resolution. The Back Propagation algorithm it is applied repeatedly during the training process to correct the weights based on the prediction (from the output, at the end of the algorithm's iteration) all the way to where inputs are inserted to the network. The error to propagate backwards in the network, in order for each model to learn is calculated each iteration by the Cross Entropy **loss function** as shown in equation (1).

$$CrossEntropy = -\sum Y_i' \cdot log(Y_i) \tag{1}$$

Where Y_i' are the actual probabilities and Y_i is model's computed probabilities. It indicates how bad the prediction on a single example was, calculating the error between the true value (**labels**) and the value the system predicted, respectively. The true values are *one hot* encoded. Furthermore, there is a vector of 0s and 1s for each input representing the actual value of the input, whether it is a 2 or a 3 or any other digit from 1 to 10 which is later compared to system's prediction to compute the loss error.

To properly adjust the weight vector an algorithm that computes the gradient vector for each weight it is used indicating by what amount the error would increase or decrease if the weight were increased by a tiny amount. The weight vector is then adjusted in the opposite direction to the gradient vector.

The **learning rate** is the step chosen so the algorithm converges or reaches an optimal solution which usually is the negative gradient vector indicating the direction of steepest descent in a high-dimensional space of weights values. It indicates how the algorithm moves towards to the local minimum starting with an initial value and updating it till the loss function reaches an optimal solution or the output error is low on average. It is relevant for how fast or slow the algorithm learns.

Tensor Flow provides a built-in optimization algorithm, the stochastic Gradient Descent Optimizer, which allows the model to learn and it is directly correlated to the concept of learning rate. By computing the outputs and the errors of a set of inputs, this algorithm calculates the average gradient for those examples, and adjusts the weights accordingly. Its implementation is shown in Algorithm 1.

Algorithm 1

```
optimizer =
    tf.train.GradientDescentOptimizer(LR)
with tf.name_scope("train"):
train_step =
    optimizer.minimize(cross_entropy,
    global_step=global_step)
```

The **bias** is a measure that indicates how easy it is to get the neuron to output. In the present study the bias was initialized always at 0.1, regardless the neural network model, and taking the shape or dimension of the number of neurons of the next layer to which it outputs.

A simple schematic illustrating most of the definitions above can be found in Figure 2.

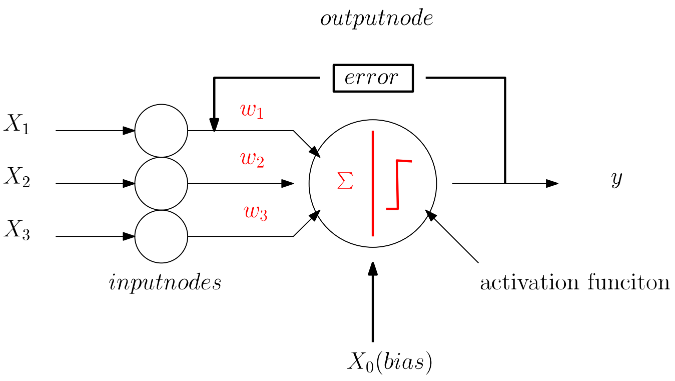


Fig. 2: Simplified and zoomed in, data flow in a neural network

Where x_i, w_i and y_i represent the inputs, weights and outputs, respectively. Also, one can identify the weighted sum as $\sum$ and the activation function of the neuron.

For a better clarification on how weights and biases are visually processed, Figure 3 represents their evolution through the training process using Tensor Board. The figure mainly indicates the learning process is happening and values along the code run are changing from zero at the beginning to ranges of [-1, 1] and [-0.6, 0.6], respectively. This happens while the gradient is being calculated and fractions of the vector that points in the direction of the minimum loss, are used to update biases and weights. This update is usually obtained by adding the learning rate discussed previously.

It is useful to check updates and histograms if parameters are diverging to $+/-\infty$ or if the biases become very large. The weights, as defined in the code, have an approximately Gaussian (normal) distribution, after some time. For biases, the histograms start at 0, and usually en up being approximately Gaussian.

Moreover, in order to evaluate each model, an **accuracy** parameter was defined as the percentage of correctly recognized digits. Test were carried out while saving the model each time, a learning rate following a polynomial decay or a precise 0.003 and over 100, 1,000 and 10,000 iterations. It is important that the iterations are scalable, as the more *epochs* performed the more learning is achieved. An *epoch* in machine learning context means the iterations needed for training once all over the dataset data. Obviously, with 100 iterations, each iteration with an input of 100 MNIST images non-replaceable, the training will only be done over 1,000 examples. For reaching a data overall training, the iterations number should be on $epoch = \frac{50{,}000}{100} = 500$ iterations. Therefore, 1,000 iterations will mean training twice the overall inputs and so on. There is a small batch of 100 inputs for each iterations due to when applying the loss function from one image to another the algorithm already has acquired knowledge over the 100 previous and the probability to actual get to a local minimum are higher rather than processing one image at a time. Each training batch gives an estimation of the average gradient over all the remaining examples, leading to more accurate results on new inputs from the test batch.

Finally, noting that the tests were made with a personal computer Intel Core i7-4900MQ up to 4.1GHz, a 4GB dedicated NVIDIA graphics and 32GB of RAM.

IV. Softmax Single Layer model

This section describes a basic implementation of a TensorFlow 10-neuron single linear layer model. This model will be used later to compare with more complex ones. Figure 5 illustrates the model built visualized in Tensor Board. Since the neurons are a parameter to define, the representation is not that accurate and in the model one can only see the different blocks used to train the algorithm and the connections between them.

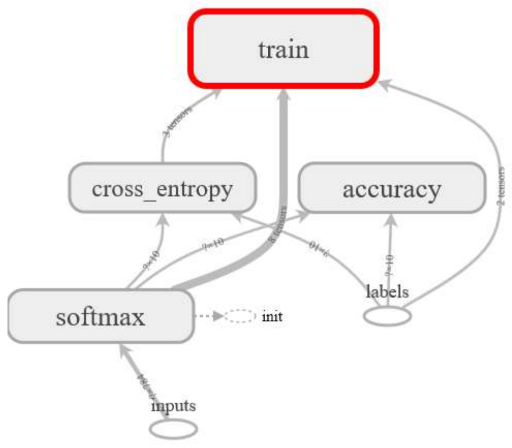


Fig. 5: TensorBoard graph visualization single layer model

This single layer model prototype it is based on the Softmax Regression algorithm which is used for multi-class classification. The prediction it does for a given input, are relative measurements of how likely it is that the image falls into each of the 10 target classes. The generated values are between 0-1 with a total sum of 1.

Once implemented and functional, as a first approach a research test was initiated to see how relevant the learning rate really is. In both cases, Figure 6 and Figure 7, the Y axis represents the level of performance and the X axis illustrates the number of iterations made. This learning rate test was conducted in order to point out that the choosing of a correct one is relevant to this study. For this test, a level of 90.00% of accuracy was reached when the learning rate was 0.003 and when reduced to 3e-05 an accuracy of 75.00% was obtained.

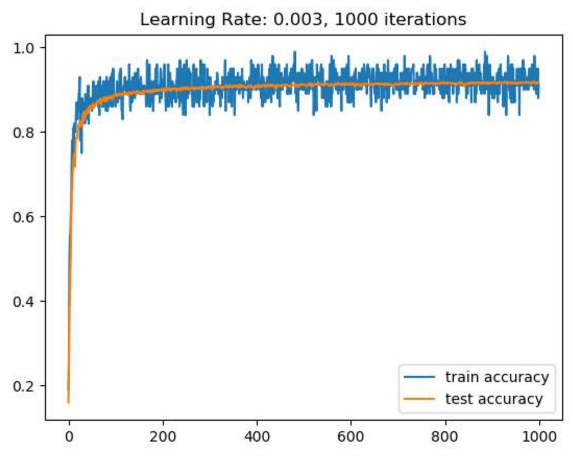


Fig. 6: Python plot of the accuracy test of the model

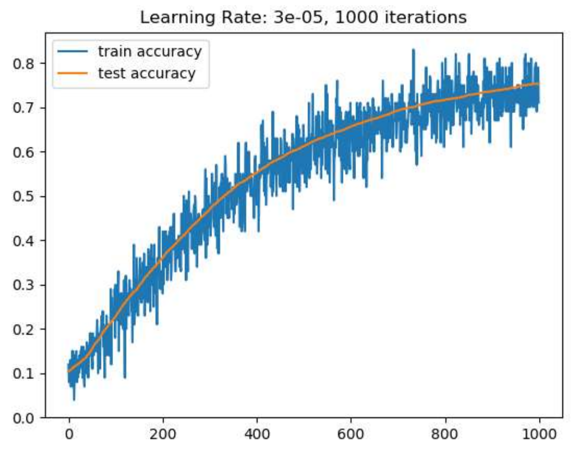


Fig. 7: Python plot of the accuracy test of the model with low learning rate

One can see that the learning rate chosen in Figure 6 is indeed the proper one. The model learns faster over the first iterations. It is also important to note that if the learning rate is chosen wrong, too big or too small it could end up with results such as in Figure 7.

If the step is too big it will miss the minimum while searching surroundings whereas never converging. On the other hand, if it is too small the result would be more precise although it will converge at a slower rate, conducting to a significantly time increase.

Figure 3 represents the accuracy and the cross-entropy, both behaving as they should. Cross-entropy decreasing while the model learns and the error between the predicted and the original digit gets lower by every iteration with the *BP* algorithm.

With this model the highest performance obtained was with a learning rate of 0.003 over 10,000 iterations, within 4

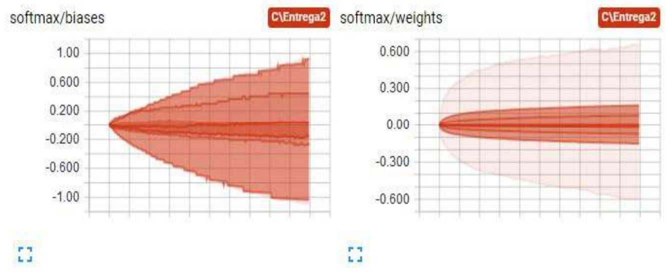


Fig. 3: TensorBoard biases and weights visualization

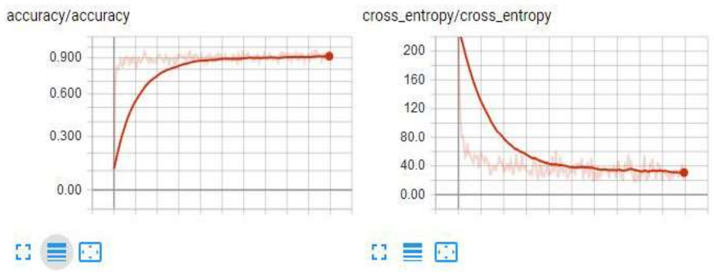


Fig. 4: TensorBoard accuracy and cross-entropy visualization

minutes and 38 seconds with an accuracy over the test dataset of 92.52%.

V. Deep Learning Study

So far the study in Section IV has been based on one classification layer, *Softmax*. In order to proceed with the objectives of this study three activation functions were used, namely: Sigmoid, ReLu and Convolutional non-linear rectifying functions.

The previous code used for the single layer was used to implement these three functions.

Regarding the batch of images processed on each iteration, the number remained the same, preferably the small amount of 100 images. The code structure and its design it is resembling to the single layer one.

Despite the similarities, there were some changes regarding the learning rate which was changed to follow a polynomial decay since seemed that decreasing the learning rate meanwhile the execution of the algorithm is much successful. If not carefully chosen, results in a badly performing model which could lead to what is known among scientists as *vanishing* or *exploding gradient* problem [8]. Reached a certain point on the deep learning, the algorithm can not move forward unless the steps get smaller. The polynomial decay applies a polynomial decay function to a provided initial learning rate to reach an end learning rate in the given decay steps. This in code is shown in Algorithm 2.

Algorithm 2

```
# Polynomical decay learning rate
global_step = tf.Variable(0,
    trainable=False)
starter_learning_rate = 0.003
end_learning_rate = 0.0001
decay_steps = 10000
learning_rate =
    tf.train.polynomial_decay(starter_
  learning_rate, global_step,
      decay_steps,
      end_learning_rate,power=0.5)
```

Finally, since deep learning study in order for the algorithm to obtain better convergence results, the *dropout* regularization technique was used. It is an optimization applied due to the *overfitting* on the loss function. Moreover, the overfitting issue is when reached a particular point in the training in which the algorithm has memorized the training examples, it stops learning to generalize to new situations. Therefore, *dropout* is a technique to deactivate each iteration, a set of neurons which will make the biases and the weights of that particular neuron to not change and the activation function to be 0 and therefore be slightly boosted next iteration. The probability to keep neurons in the neural network has been established to be invariant at 75.00% for all the models. Therefore at each iteration a 25.00% of the total neurons are being deactivated. This technique was applied and implemented aiming to increase the level of accuracy.

A. Sigmoid Function

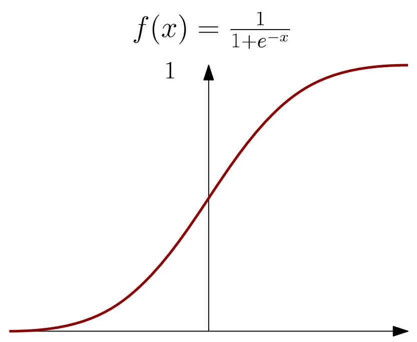


Fig. 8: Sigmoid function representation

The Sigmoid function is a continuous function through time (*X* axis) which is zero for the negative part and one, otherwise. The Figure 8 illustrates it.

The model implemented at first with this function consisted of four layers of the same function, changing through each layer the neuron number for interoperability and output purposes. At the end of the Sigmoid model data flow, there still remains the Softmax layer. The results of this model were insignificant and the model was not actually learning at all. The solution was to reduce the amount of intermediate Sigmoid layers and see if the results were better. The new model can be visualized in Appendix A.

The results obtained with this model, an accuracy of 96.63%, was reached within 10,000 iterations in 17 minutes and 35 seconds simulation.

B. *ReLu function*

The ReLu non-linear function has actually been discovered to work better on deep learning studies. The problem Sigmoid function presents is that on the sides is flat which leads to a gradient of zero when calculating the loss. The gradient is actually what allows the algorithm to get to the point of convergence therefore having a zero in some regions is not ideal. As explained before, this is known as *vanishing gradient* problem. Despite all this, the ReLu function seems to overcome this issue since it has been inspired by the actual brain neurons functioning. By taking a look at the following figure representing the ReLu function, it can be observed that for the negative part of the X axis it is always zero whereas for the positive, it is an identity function which will never give as a result a gradient of zero.

$$f(x) = x^{+} = max(0, x)$$

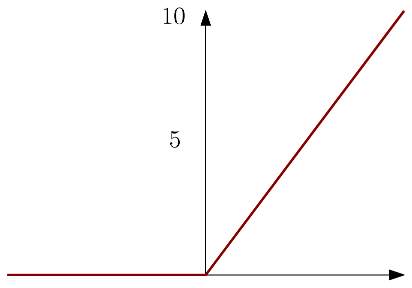


Fig. 9: ReLu function representation

This model shows improvement in the learning process. Compared to the previous model, it has been able to sequence up to four ReLu layers as shown in Appendix B. Furthermore, the results obtained with this model, an accuracy of 98.03%, was reached within 10,000 iterations in 16 minutes and 39 seconds simulation.

C. *ReLu and Sigmoid Deep Learning Model analysis*

For optimization and research purposes it was decided to combine both Sigmoid and ReLu function and see if the results, due to better feature extraction and merged were going to gain in precision. It was expected that the learning rate would not be invariant anymore and it would follow a polynomial decay achieving a higher accuracy in this way. In previous models, Sigmoid and ReLu, the polynomial decay was applied as well but leading to not representative results or not finishing converging it was fixed to 0.003. The resulting graph is represented in Appendix C.

The results obtained with this model, an accuracy of 96.40%, was reached within 10,000 iterations in 19 minutes and 59 seconds simulation.

D. *CNNs*

The goal was to build a relatively small Convolutional Neural Network (CNN) for recognizing the same inputs as previous models. The architecture of a typical CNN is structured as a series of levels. One could think of this levels as the different layers the inputs pass by. The four main key ideas are : local connections, shared weights, pooling and the use of many layers. Nowadays studies build models using 10 to 20 non-linear activation functions inbetween the convolutional layers. While that is out of the scope of this study, a simpler model was built: 3 sequential convolutional layers, pooling layers for each one of the previous, a fully connected layer and the softmax layer. The levels mentioned before and relevant at this point are the convolutional and pooling layers. In contrast to previous models, this one does not use a flattened array as a input, but reshapes the data from the MNIST database and makes pixel matrix alike representation for further interpretation. The CNN, uses the entire image while mapping its features, whether it is a straight line or a curve one therefore sampling the outputs by each layer. Moreover, local groups of values are often highly correlated and the pooling layers merge those semantically similar features into one. But, because the relative position of similar features can vary somehow, the pooling layer computes the maximum of a local patch of units. Neighboring pooling units take input from patches that are shifted by more than one column, hence reducing the dimension of the representation and creating an invariance to small shifts or distortions. Meaning, the model can generalize better and make predictions on feature pairs that were previously not been trained on. Figure 10, is a schematic representation on how CNN down-samples and extracts the features. Each layer connects with the one beneath using local patches through a set of weights. Regarding weights and biases in this model of NN, the degree of freedom is increased.

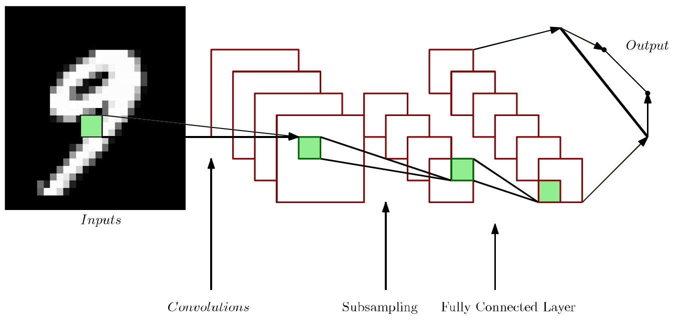


Fig. 10: CNN data flow representation

With this convolutional model, the highest performances have been achieved, an accuracy of **99.02%**. This was possible while iterating over 10,000 times and implementing the regularization *dropout* technique.

VI. Discussion

TABLE 1 shows the results of the different training models discussed along this study. This section is dedicated to examine them. As the table shows there are different algorithms, whether or not the *dropout* regularization technique has been used, the number of iterations, the time spent by the system to finish training, the learning rate and the accuracy both on train and test datasets, respectively.

Overall, the models behave, train and learn better when the number of iterations is higher. This seems logic since the more training the better will be the results on the test data. Nevertheless, time is also directly proportional to iterations increase. Regarding the learning rate, it must be clarified that the polynomial decay did not work for the simpler Softmax, Sigmoid nor ReLu models, giving convergence errors along the code execution. On the other hand, on the convolutional model when overfitting happened if not polynomial at 10,000 iterations the system would have never converged.

Furthermore, all models implementing *dropout* performed worst than when it was not applied. The *dropout* optimization leaves the neural network damaged with deactivated neurons at each iteration. When *dropout* was applied to the model both combining Sigmoid and ReLu there was a lower accuracy on the train dataset than in the test dataset. This means that the neurons deactivated were boosted by not changing the weight through Back Propagation during an iteration. Without the implementation of this regularization technique the highest performances reached are on the convolutional model within 1,000 iterations (98.85%). Despite all of that, the train step becomes very noisy. When deactivating the 25% of the neurons there are less neurons and the noise is smaller. The *dropout* technique solves the problem of overfitting and instead of the system stopping to generalize reached a certain point, it remains constant if there is anything other than improvements while the error does not start to increase. The convolutional model giving the best performances (0.98 loss rate) is highlighted in bold.

VII. Future Work

Supervised Learning, the base of the work within this article as successful as it gets in accuracy, there still remains one factor to overcome. Most human and animal learning is by far unsupervised. Meaning, the natural way of discovering things around is accomplished by observing them rather than being told the name of every object.

Future work could include addition and combination of layers while changing activation functions. It should be possible to take the code and use it as basis to reach Deep Learning based on Recurrent Neural Networks (RNNs) by reshaping it as needed. However, this possibility was outside the scope of this work due to its complexity and time. Another improvement is working with colored images rather than black and white ones.

If it were to continue with the work where this article ends its study, first of all one may also think of different ways of improving what has already been done. For instance, different regularizations techniques to apply. There is one regularization for when overfitting occurs rather than *dropout* called *batch normalization* which recently seems to have proven very successful results. Its functionality focuses on statistically normalizing by mean and variance the train and test datasets, independently.

Finally, the dataset could also be changed and observe if the results are as good as for the dataset used in this study. There is also the possibility to find mechanisms and optimization to accelerate the learning process.

VIII. Conclusions

TensorFlow system and its programming model offer a set of tools which facilitate both production and experimentation over self implemented models nevertheless scalable and efficient. Through this paper's work a set of models have been implemented and each one of them led to the next one. Starting with the single layer model using *Softmax* activation function for classification purposes as the anchor for all the upcoming *Sigmoid*, *ReLu*, a combination of both and *Convolutional* model implementations. All combined within TensorBoard's visualization tools helped understanding much better the entire process with data flow, histograms, accuracy and cross-entropy representations. Furthermore, TensorBoard helped fixed issues between getting to properly connect the layers with each other and create graph models to sufficiently work.

Also discussed, the commonly central algorithm for Deep Learning, *Back Propagation* for supervised weight-sharing has been applied to each and every model and helped getting the Neural Network to learn over the train dataset and get better results while overcoming the test dataset successfully.

Regarding obstacles encountered during the process of the present article, issues have picked up during the experimental work such as Python implementation problems, deprecated functions of a constantly changing and growing framework or overfitting problems through code execution which were later solved with the regularization techniques, case in point *dropout* regularization and by changing the way the learning rate behaves during code execution.

All of this made it more plausible to reach the conclusion that TensorFlow is a powerful tool in Image Processing and although the machine learning movement it is halfway through, there is still room for improvement. Alongside this conclusion stands that Neural Networks and Deep Learning combined are capable of achieving levels of accuracy which were thought impossible to get years ago. Therefore despite all the controversy and improvement made through the past years these have demonstrated to be powerful techniques that led to the world we know today of autonomous self-driving cars, speech recognition, autonomous mobile robots, scene parsing, object detection, shadow detection, video classification or Alzheimer's disease neuroimaging just to instance some of them successes in the field.

As for the results accomplished, the maximum accuracy was reached with the Convolutional layer model implementing *dropout* regularization, **99.02%** hence a loss rate of 0.98%. Taking into account as mentioned in the State of the Art section, the global actual maximum reached is within a 0.39% loss, accomplished with GPUs rather than with CPUs, is a satisfactory outcome. This being possible while having less computational resources, hence less training and a local minimum not as efficient as the resulting from a high computational performance machine, 50 times slower than ensemble implementation and yet still a fitting result. Accordingly, it can be said, the objectives of this paper have been fulfilled. TensorFlow represents the future for complex deployments of neural network models, leading to more accuracy and improved results either it is image processing or any other

Algorithm	**DropOut**	**Learning Rate**	**Iterations**	**Time Elapsed (hh:mm:ss)**	**Accuracy (%)**	
					Train Dataset	***Test Dataset***
Softmax	No	0.003	100	00:00:02	87.19	87.57
			1,000	00:00:27	89.77	91.68
			10,000	00:04:38	93.14	92.52
Sigmoid	No	0.003	100	00:00:10	14.15	9.98
			1,000	00:02:01	12.19	10.55
			10,000	00:17:35	99.15	96.63
ReLu	No	0.003	100	00:00:10	81.92	77.16
			1,000	00:01:46	99.43	95.95
			10,000	00:16:39	97.00	98.03
Sigmoid and ReLu	No	Polynomial decay	100	00:00:07	17.04	18.05
			1,000	00:01:39	93.01	84.36
			10,000	00:17:37	99.66	97.75
	Yes	Polynomial decay	100	00:00:09	19.00	12.15
			1,000	00:01:55	81.33	84.67
			10,000	00:19:59	97.31	96.40
Convolutional	No	Polynomial decay	100	00:05:20	94.72	91.50
			1,000	00:57:06	99.96	97.87
			10,000	04:26:56	100.00	98.85
	Yes	**Polynomial decay**	100	00:05:56	91.38	87.07
			1,000	00:55:30	99.80	97.63
			10,000	**04:34:27**	**100.00**	**99.02**

TABLE I: COMPARISON OF PERFORMANCES ON MNIST DATASET USING DIFFERENT NEURAL NETWORK MODELS.

of the above machine learning scenarios being analyzed.

ACKNOWLEDGEMENTS

Thanks to Jordi Casas Roma, for supervising and guiding me through carrying out this project and all the references and articles he suggested which were nevertheless helpful and essential to get the work within this project done. Moreover, I would like to be thankful to all the teachers I came across over this past years. It was a pleasure working with all of them and sure thing I have learned a lot. Finally, I would like to definitely thank my parents for supporting me and understanding the late hours and the skipped relevant appointments.

REFERENCES

[1] S. Theodoridis, “Neural Networks and Deep Learning,” in *Machine Learning*. Determination Press, 2015, pp. 875–936. [Online]. Available: http://neuralnetworksanddeeplearning.com/

[2] Y. Qiao, C. Cortes, and C. Burges, “MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges,” 2007. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[3] P. Y. Simard, D. Steinkraus, and J. C. Platt, “Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis.” [Online]. Available: http://www.cs.cmu.edu/~bhiksha/courses/deeplearning/Fall.2016/pdfs/Simard.pdf

[4] M. aurelio Ranzato, C. Poultney, S. Chopra, Y. L. Cun, M. Ranzato, C. Poultney, S. Chopra, and Y. L. Cun, “Efficient Learning of Sparse Representations with an Energy-Based Model,” *Advances in Neural Information Processing Systems*, vol. 19, no. 1, pp. 1137–1144, 2007. [Online]. Available: http://papers.nips.cc/paper/3112-efficient-learning-of-sparse-representations-with-an-energy-based-model.pdf

[5] D. Cireşan, U. Meier, J. Masci, and J. Schmidhuber, “Multi-column deep neural network for traffic sign classification,” *Neural Networks*, vol. 32, pp. 333–338, 8 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608012000524

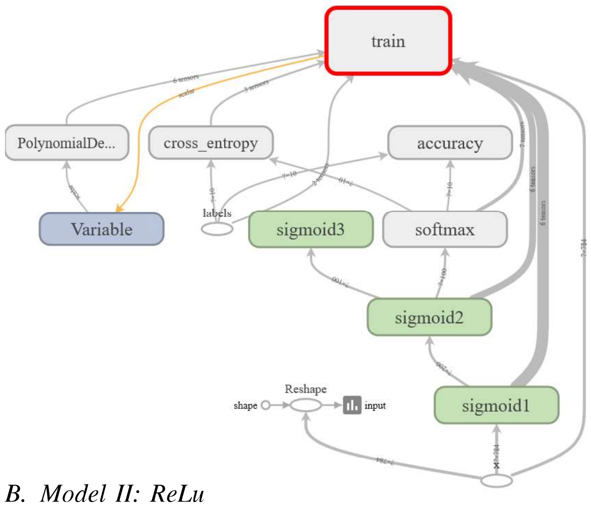
[6] A. C. Schapiro, T. T. Rogers, N. I. Cordova, N. B. Turk-Browne, and M. M. Botvinick, “Neural representations of events arise from temporal community structure,” *Nature Neuroscience*, vol. 16, no. 4, pp. 486–492, 2013.

[7] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: A system for large-scale machine learning,” 2016. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[8] G. Ian, Y. Bengio, and A. Courville, *Deep learning*. MIT Press, 2011. [Online]. Available: http://www.deeplearningbook.org/

APPENDIX

A. *Model I: Sigmoid*



B. *Model II: ReLu*

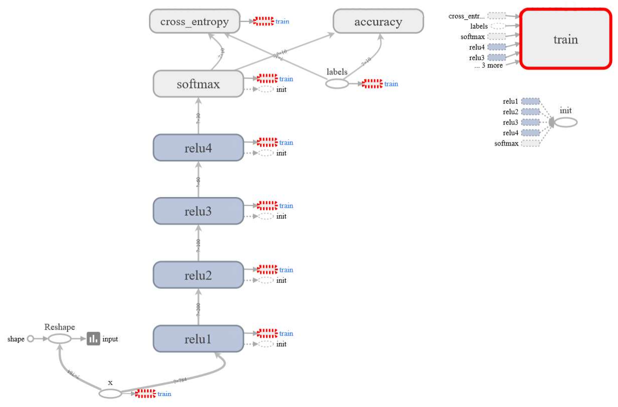


Fig. 12: ReLu function model implementation TensorBoard view

C. *Model III: ReLu and Sigmoid*

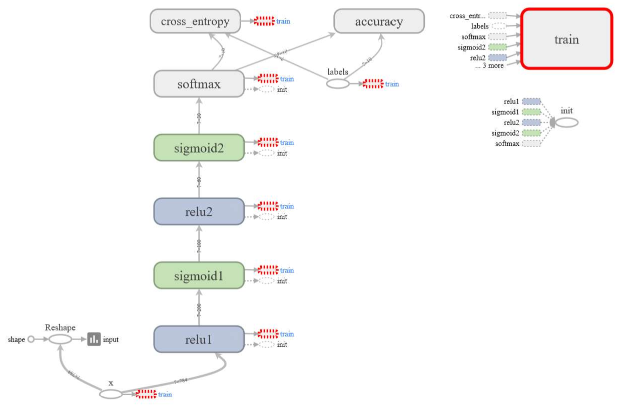


Fig. 13: ReLu and Sigmoid functions model implementation TensorBoard view

D. *Model IV: Convolutional*

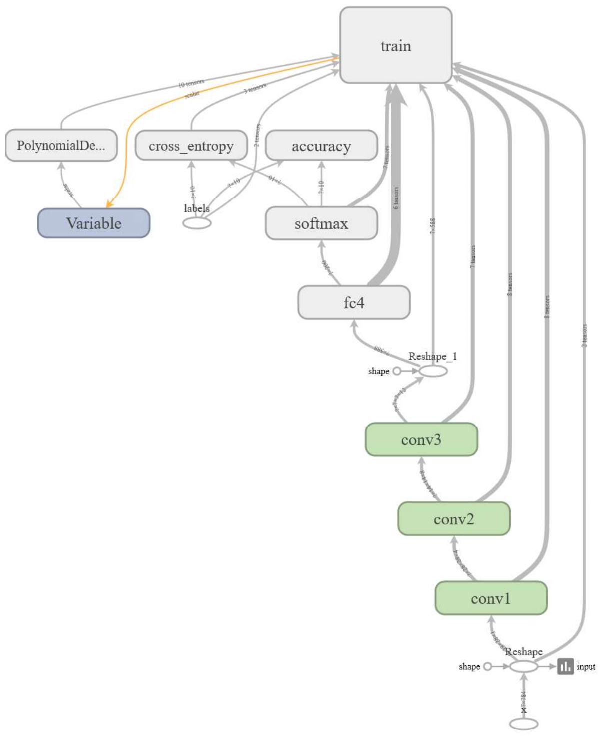


Fig. 14: Convolutional layers model implementation TensorBoard view