A study on meta-parameter tunning for learning feed-forward networks for image classification

Alexandros Dimos, Andrei Dumitra, and Diana Fulga
University Politehnica of Bucharest, Department of Computer Science
313 Splaiul Independentei, 060042 Bucharest, Romania
{alexandros.dimos.95, dumitra.and, fulga.dd}@gmail.com

***Abstract*—In this article we present a study on tunning the meta-parameters for learning feed-forward neural networks in order to accelerate the learning process or have better results.**

I. Introduction

Recent results in machine learning proved that deep neural networks are very efficient for feature extraction from raw visual input [1]. As information from still images might be useful for more complex tasks, recent experiments combined deep convolutional networks with different models, like reinforcement learning algorithms [2] or architectures for sequence classification [3].

A novel deep reinforcement learning algorithm [2] reached human-level game playing behavior based on visual input and a simple scalar reward only. The model was tested on the Atari 2600 game console and scored above the human standard on many of those games. But, there were some specific games for which the model didn't learn at all (e.g. Montezuma's Revenge). Those games involve planning, following a strategy during the game and remembering details that cannot be extracted from the visual input. There are challenging learning tasks still to be investigated here.

Other recent architectures involved stacking recurrent layers on top of the usual convolutional layers that process the visual input. Those models showed promising results on a large number of tasks like: generating image descriptions [4], activity recognition [5], or video classification [3].

Our goal is to follow the two novel research directions described above in order to build systems capable of human-level planning based on high-level sensory input. Combining CNNs, reinforcement learning and memory models like LSTMs [6] seems promising.

As the above long-term goal is very ambitious, we started with experiments with classic feed-forward networks for image classification. We tested several gradient-based learning algorithms and compared the results.

Section II presents previous research results on using neural networks for image classification. Section III describes the network architecture we used in our experiments and the learning algorithms we trained our models with. We tested our networks on two popular datasets, described in Section IV. The results of the experiments we performed are presented in detail in Section V. This article ends with the important conclusions of our work in Section VI and our short-term and long-term research goals in Section VII.

II. Background

A. Neural Networks

Artificial Neural Networks (ANNs) are a family of learning algorithms inspired by the way biological nervous systems, such as the brain, processes information. They are basically systems of interconected processing elements, called neurons or perceptrons, which influence each other.

The first conceptual model of an artificial neural network was developed in 1943 by Warren S. McCulloch, a neuroscientist and Walter Pitts, a logician. In their paper, titled "A logical calculus of the ideas imminent in nervous activity[7], they describe the concept of neuron: a single cell living in a network of cells that receives inputs, processes those inputs, and generates an output.

Their work, and the work of many scientists and resarchers, was meant to discribe how an ANN was designed as a computational model based on the brain to solve certain kind of problems by estimating or aproximating functions that can depend on a large number of inputs and which are generally unknown.

1) Components: A neural network has at least two physical components, namely, the processing elemaents, called neurons and the connections between them, known as links[8] or synapses. The word 'network' in ANN referes to this interconection between the neurons found on different layers. An example network has three layers:

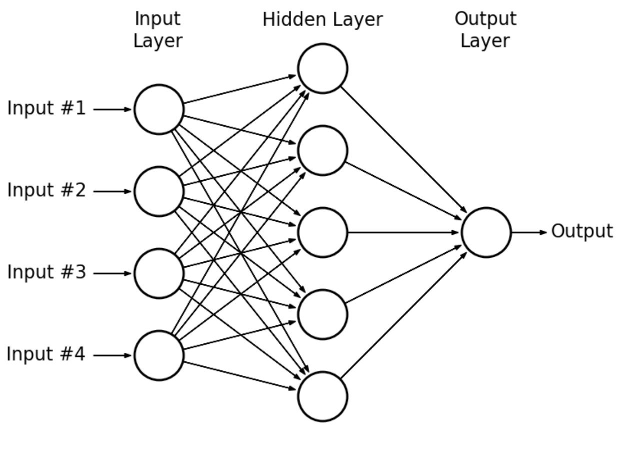


Fig. 1. Neural network with three layers[1]

1) The first layer has input neurons which send data via synapses to the second layer of neurons, the so-called hidden layer.
2) The hidden layer (”hidden” because we do not interact with it directly, but through the input provided on the first layer) processes the information previously received and outputs his own information to the third layer.
3) The third and the final layer, also called the output layer, generates the output of the network based on the information received.

2) The Neuron: The most basic structure of an artificial neural network, the neuron, is directly modeled by a number of weighted inputs - that means that each and every neuron on the previous layer has a certain quota of influence on it, a state or an output (which could be 0, 1 - 0 meaning that the neuron is ”deactivated” and does not influence at all the following layer and 1 meaning that the neuron will influence directly the neurons on the next layer, or another activation function) and an activation value (or threshold). An example of an activation function would be described in the equation (1): if the total sum of the weighted inputs is greater or equal than the threshold value, the neuron activates, directly modeling the following layers.

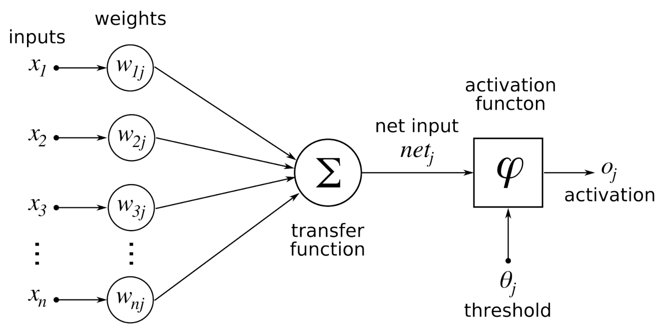


Fig. 2. Example of a neuron model[2]

$$\varphi = \begin{cases} 1, \sum_{i=1}^{n} w_{ij} * x_i \geq \theta_j \\ 0, otherwise \end{cases} \tag{1}$$

B. Other Image Recognition Neural Networks

There were many attempts to creating better neural network cappable of image processing, image classifing and image recognition. For example, let us select MNIST digit data set. There were over 60 algorithms that tried learning to classify 28x28 images that represent digits, with error rates varying between 0.23% and 12%. Here is a list of the most importnat ones:

There were recent breakthroughs in this domain recently, one of them being Baidu’s Deep Image System, engineerd by Ren Wu, Shengen Yan, Yi Shan, Quingqing Dang and

TABLE I. MNIST DATABASE - OTHER IMAGE PROCESSING NEURAL NETWORKS

Classifier	Preprocessing	Test Error Rate
c1-20-P-40-P-150-10 [ED[3]]	width normalization	0.23%
6-layer NN (on GPU) [ED]	none	0.35%
1-20-40-60-80-100-120-120-10 [ED]	none	0.35%
784-800-10	width normalization, deslanting	0.39%
unsup pretraining	none	0.39%

Gang Sun[4]. It attained Best ImageNet Large Scale Visual Recognition Challenge 2015 result untill now, with an error rate of just 4.58%. The runner up is BN-Inception with an error rate only .34% higher.

The progress is astonishing comparing with first place in 2012, claimed by University of Toronto’s SuperVision, with an error rate of 16.42%.

C. Convolutional Neural Networks

Convolutional Neural Networks (shortly CNN) are biologically-inspierd variants of MLPs[5], which are deisnged to use minimal ammounts of preproesing. The difference between a classical neural network and a convolutional neural network is that CNN archtectures are made for visual input (images). This allows us to encode certian properties into the architecture that vastly reduces the amount of parameters in the network.

For example for an image of a greater size (eg: 300x300) we would need 300*300*3 = 270000 weights on the input layar. This huge number of parameters would eventualy lead to overfitting.

Therefore, unlike a classical neural network, the layers of a CNN have neruons arranged in a 3D manner. The neurons in a certain layer will only be connected to a smal region of the layer before it insted of all the neurons.

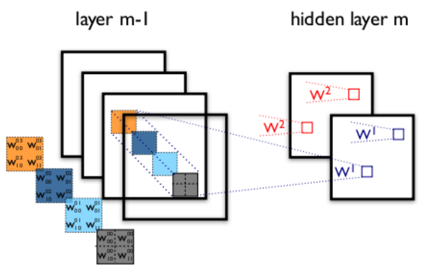


Fig. 3. Example of two layers of a ConvNet[6]

There are different types of layers used in CNN architectures:

1) Convolutional Layer (CONV)
2) Rectified Linear Units (RELU)
3) Pooling Layer (POOL)
4) Fully Connected Layer (FC)

[1]http://www.astroml.org/_images/fig_neural_network_1.png
[2]http://en.wikibooks.org/wiki/Artificial_Neural_Networks/Activation_Functions#/media/File:ArtificialNeuronModel_english.png
[3]Elastic Distortions

[4]http://arxiv.org/pdf/1501.02876v3.pdf
[5]Multilayer Perceptron
[6]http://deeplearning.net/tutorial/_images/cnn_explained.png

1) Convolutinal Layer: This kind of layer consist of a rectangular grid of neurons. It requires that the previous layer also be a rectangular grid of neurons. Each neuron takes input from a section of the previous layer, forming an image convolution of this layer on the current one.

2) ReLU Layer: The rectifier function is an activation function $f(x) = max(0, x)$ which can be used just like any other activation values. This layer is composed of neurons that use this function as theyr activation function. It is mostly used because of efficiently it can be computed compared to more conventional activation functions like the sigmoid. The rectifier function is used instead of a linear activation function to add non-linearity to the network, thus allowing it to compute a linear function.

3) Pooling Layer: This layer takes small rectangular blocks from the convolutional layer and subsamples it to produce a single output from that block. There are several ways of doing this pooling, for example taking the average or maximum or any other linear combination of the neurons in the block.

4) Fully Connected Layer: This layer takes all the neurons on the previous layer and connects it to every neuron it has. Fully connected layers are linear located, therfore there can not be any more convolutional layers after it.

Using combinations of these three a Convolutional Neural Network transforms the original image layer by layer from the original pixel values to the final class scores.

III. Architecture and Learning Parameters

A. Feed-Forward neural networks

The feed-forward neural network was the first and simplest type of artificial neural network devised. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network. [7]

The popularity of feed-forward networks derives from the fact that they have been applied successfully to a wide range of information processing tasks in such diverse fields as speech recognition, financial prediction, image compression, medical diagnosis and protein structure prediction; new applications are being discovered all the time. [8]

B. Training & Error backpropagation

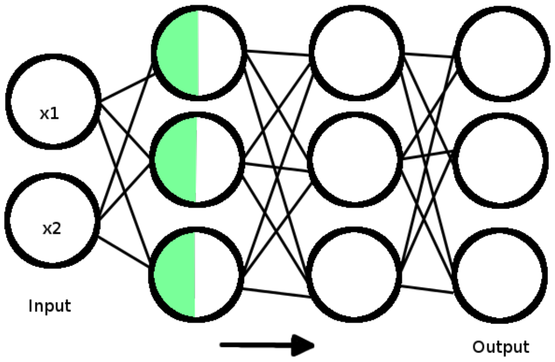
There are two main types of learning methods: supervised and unsupervised; we will focus on the supervised learning method. In a supervised learning environment the training data consists of a set of training examples; each example is a pair of an input and a desired output. A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. [9]

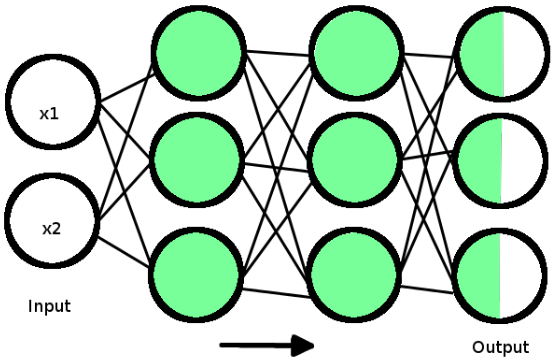
Error backpropagation is a comonly used method of training feed-forward neural networks used in conjunction with an optimization method such as gradient descent[III-C]. The method calculates the gradient of a loss function with respects to all the weights in the network. The gradient is fed to the optimization method which in turn uses it to update the weights, in an attempt to minimize the loss function. The backpropagation algorithm requires a set of training data with a known desired output thus being classified as a supervised learning algorithm. [10]

The backpropagation algorithm goes as follows:

1) The input is given to the network to be processed in order to get an output.

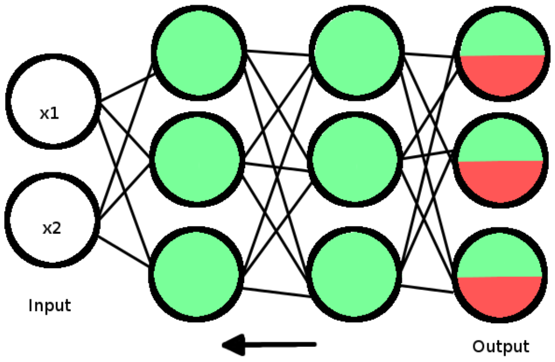
Fig. 4. Forward propagation of input





2) The output is compared to its desired value counterpart and the error (delta) is calculated.

Fig. 5. Backward propagation of error

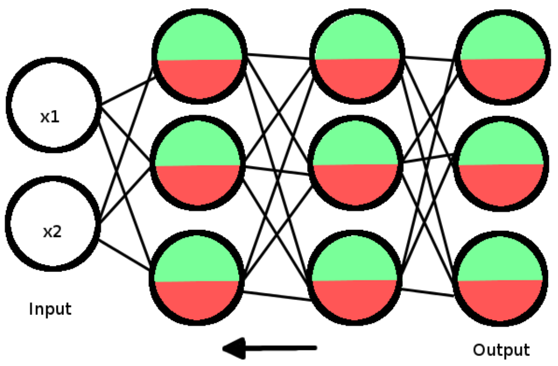


[7]http://en.wikipedia.org/wiki/Feedforward_neural_network

[8]http://www.biochem.ucl.ac.uk/~shepherd/sspred_tutorial/ss-nn.html

[9]http://en.wikipedia.org/wiki/Supervised_learning

[10]http://en.wikipedia.org/wiki/Backpropagation



3) A gradient is calculated for each weight by multiplying the neuron activation and delta.
4) The weight is adjusted accordingly using an optimization algorithm such as gradient descent[III-C].

Since backpropagation uses gradient descent[III-C] one needs to calculate the derivative of the squared error function with respect to the weights of the network. Assuming one output neuron, the squared error function is:

$$E = \frac{1}{2}(t-y)^2 \tag{2}$$

Where:

E is the squared error

t is the desired output

y is the actual output

For each neuron j its output z_j is defined:

$$z_j = \varphi(a_j) = \varphi(\sum_{k=1}^{n} w_{kj} z_k) \tag{3}$$

$$a_j^{(l)} = \sum_{k=1}^{n} w_{kj}^{(l-1)} z_j^{(l-1)} \tag{4}$$

Where:

l is the number of the layer.

n is the number of input units to the neuron

w_{ij} is the weight between neurons i and j

If the neuron is on the first layer then z_i is equal to the input x_i.

The activation function φ is usually non-linear and differentiable. The sigmoid function is commonly used [III-D]. The derivative of the error with respect to the weigth is:

$$\frac{\partial E}{\partial w_{ij}} = \delta_j z_i \tag{5}$$

with

$$\delta_j = (z_j - t_j)\varphi(a_j)(1-\varphi(a_j)) \text{ for neurons on output layer} \tag{6}$$

$$\delta_j = (\sum_{l \in L} \delta_l w_{jl})\varphi(a_j)(1-\varphi(a_j)) \text{ for neurons on other layers} \tag{7}$$

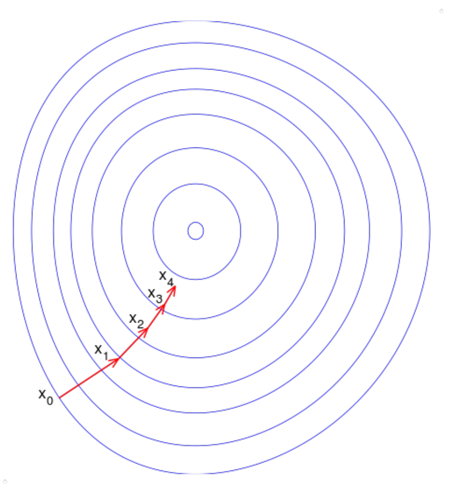
To update the weight w_{ij} using gradient descent[III-C], one must choose a learning rate, α. The change in weight, which is added to the old weight, is equal to the product of the learning rate and the gradient, multiplied by -1:

$$\Delta w_{ij} = -\alpha \frac{\partial E}{\partial w_{ij}} \tag{8}$$

The -1 is required in order to update in the direction of a minimum, not a maximum, of the error function. For a single-layer network, this expression becomes the Delta Rule.

C. Gradient descent

Gradient descent is a first-order optimization algorithm. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point. If instead one takes steps proportional to the positive of the gradient, one approaches a local maximum of that function; the procedure is then known as gradient ascent. [11]



[12]

D. Sigmoid function

A multi-layer neural network can compute a continuous output instead of a step function. A common choice would be the logistic function:

$$f(x) = \frac{1}{1+e^{-x}} \tag{9}$$

This function, also called sigmoid function has a continuous derivative, which allows it to be used in training with gradient-based methods by optimizing a cost function. Its derivative is easily calculated with the formula:

$$f'(x) = f(x)(1-f(x)) \tag{10}$$

[11] http://en.wikipedia.org/wiki/Gradient_descent

[12] http://en.wikipedia.org/wiki/Gradient_descent#/media/File:Gradient_descent.svg

E. Softmax function

In mathematics, in particular probability theory and related fields, the softmax function, or normalized exponential, is a generalization of the logistic function that "squashes" a K-dimensional vector $\sigma(z)$ of arbitrary real values to a K-dimensional vector $\sigma(z)$of real values in the range (0, 1). In neural network simulations, the softmax function is often implemented at the final layer of a network used for classification. Such networks are then trained under a log loss (or cross-entropy) regime, giving a non-linear variant of multinomial logistic regression.

Particularly for our neural network:

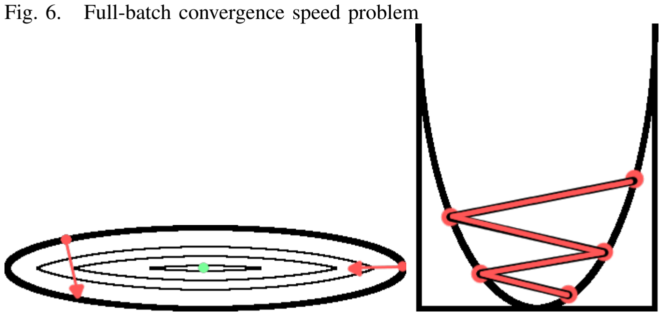
$$z_j = \frac{e^{a_j}}{\sum_{k=1}^{n} e^{a_k}} \tag{11}$$

Where n is the number of neurons on the output layer.

F. Mini-batch training

When training a neural network one may chose one of the following training methods: stohastic, full-batch, mini-batch; each with its own advantages and drawbacks. Stohastic training is the basic method of training neural networks where after every iteration of the backpropagation algorithm the weights are adjusted. It is good for on-line learning but when faced with big data sets it tends to be slower than the other alternatives. It is possible to run the backpropagation algorithm for a whole training set and then modify the weights; this method is called full-batch training. This method of training is faster than the stohastic method mostly because of its parallelization potential. The major drawback with this mode of training is the convergence speed; the gradient may be big in a direction we want to travel a small distance or may be small in a direction we want to travel a big distance. In the mini-batch learning method one will update the weights after a certain amount of error backpropagations. The mini-batch learning method keeps the advantages of the full-batch training(being faster than the stohastic counterpart) and also overcomes its problems thus being our implementation choice for the neural network.

Fig. 6. Full-batch convergence speed problem



G. Momentum

The momentum method is an extension to the classic gradient descent that reduces the risks of getting stuck in a local minimum and also speeds up the convergence considerably in cases where the process would otherwise zig-zag heavily.[13]

The weights in the momentum method represent velocities; the effect of the gradient is to increment the previous velocity, also the velocity decays by a value α which is less than 1:

$$v(t) = \alpha v(t-1) - \varepsilon \frac{\partial E}{\partial w} \tag{12}$$

$$\Delta w(t) = v(t) = \alpha \Delta w(t-1) - \varepsilon \frac{\partial E}{\partial w} \tag{13}$$

Note that the weight change can be expressed in terms of the previous weight change and the current gradient

H. Rprop & rmsprop

Sometimes the magnitude of the gradient can vary for different weights and can change during learning. This makes it very hard to choose a global learning rate. For the full-batch method this problem can be dealt with using only the sign of the gradient. The rprop method combines the idea of using only the sign of the gradient with the idea of adapting the step size separately for each weight (ex: the last 2 signs agree we increment the step size). Rprop does not consider the size of the gradient at all thus making it unusable with mini-batch learning. There is an improvement to rprop called rmsprop that combines the robustness of rprop with the effectiveness of mini-batches. In rmsprop one has to keep a moving average of the squared gradient for each weight and later divide the gradient by the square root of this average:

$$MeanSquare(w,t) = 0.9{*}MeanSquare(w,t-1) + 0.1{*}\Delta w(t)^2 \tag{14}$$

Dividing the gradient by $\sqrt{MeanSquare(w,t)}$ makes the learning work much better (Tijmen Tieleman) [14]

IV. Datasets

We experimented with the architectures and learning algorithm described in the previous section on the popular dataset MNIST used for image classification.

The MNIST database of handwritten digits [9] is made by Yann LeCun, Corina Cortes and Christopher J.C. Burges. It contains 10 clases (digits from 0 to 9). The training set has 60,000 examples and there is also a test set of 10,000 examples. We used only the training set which we shuffled and then separated (50,000 for training and the last 10,000 for testing).

Research has been done in image classification with neural networks on the MNIST database. Comparable to our work is Yan LeCun's 2 layered neural networks with an error between 1.6-4.7 in 1998. Also Mr. Simard in 2003 managed to make a neural network capable of classifying the MNIST digits with an error of 0.7 percent using only 2 layers.

V. Results

We tested our neural network both with simple gradient descent and with different additions and tweaks. The following parameters were taken into consideration in each of these tests:

- Hidden layer size
- The range of intial weights random intialization

[13] http://en.wikipedia.org/wiki/Gradient_descent

[14] https://class.coursera.org/neuralnets-2012-001/lecture/67

- Learning rate

Our primary focus was finding how these affected the network individually, which addition/algorithm offered the best learning curve and which one provided the fastest learning.

For every test, we averaged three runs through the validation data set which containted 10.000 digits.

A. *Simple Gradient Descent*

The following figures and tables depicts the results with simple gradient descent with no additions.

1) : With different layer sizes, a learning rate of 0.1 and weights between -0.05 and 0.05

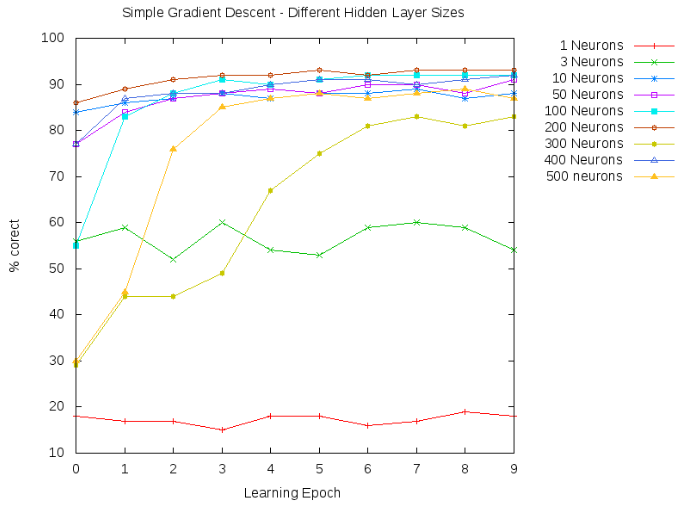


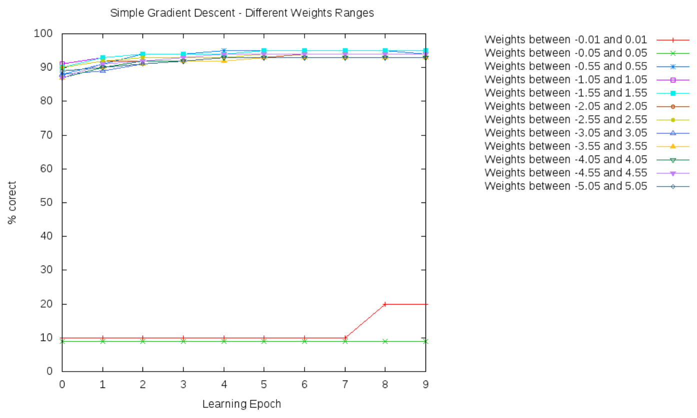
TABLE II. SIMPLE GRADIENT DESCENT - DIFFERENT LAYER SIZE

Number of Neurons	Corect classifications(%)	Epoch of achieving
1	19	8
3	60	3,7
10	89	7
50	91	9
100	92	6-9
200	93	5-9
300	83	7,9
400	92	9
500	89	8

2) : With different ranges of intial weights, a hidden layer size of 300, and learning rate of 0.01

TABLE III. SIMPLE GRADIENT DESCENT - DIFFERENT WEIGHTS

Weights limits	Corect classifications(%)	Epoch of achieving
0.01	20	9
0.05	9	ALL
0.55	95	4-8
1.05	95	5-9
1.55	95	5-9
2.05	94	6-9
2.55	94	5-9
3.05	93	5-9
3.55	93	5-9
4.05	93	4-9
4.55	94	4-9
5.05	93	4-9



3) : With different learning rates, a hidden layer size of 300 and weights between -0.05 and 0.05

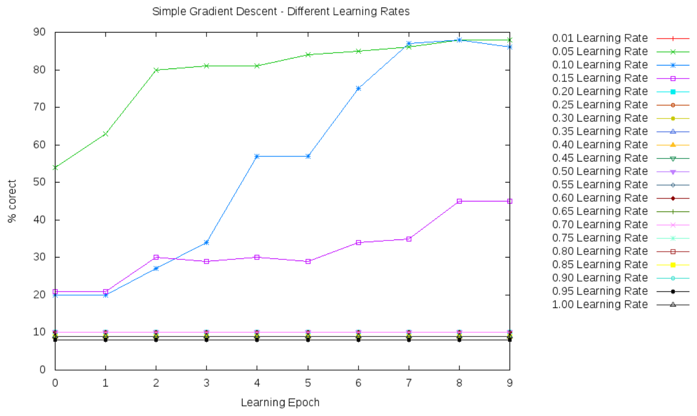


TABLE IV. SIMPLE GRADIENT DESCENT - DIFFERENT LEARNING RATES

Learning Rate	Corect classifications(%)	Epoch of achieving
0.05	88	8,9
0.10	88	8
0.15	45	8,9
OTHERS	10	ALL

B. *Simple Gradient Descent and Momentum*

The following figures and tables depicts the results with simple gradient descent with momentum.

1) : With different learning rates, a hidden layer size of 300 and weights between -0.05 and 0.05

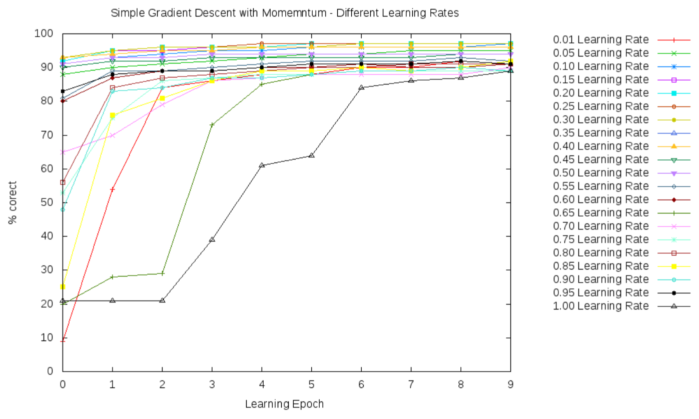


TABLE V. SIMPLE GRADIENT DESCENT AND MOMENTUM - DIFFERENT LEARNING RATES

Learning Rate	Corect classifications(%)	Epoch of achieving
0.01	92	8
0.05	88	8,9
0.10	88	8
0.15	45	8,9
0.20	97	5-9
0.25	97	5-9
0.30	97	6-9
0.35	96	4-9
0.40	96	4-9
0.45	94	8,9
0.50	94	3-9
0.55	93	8
0.60	91	6,9
0.65	89	6-9
0.70	90	9
0.75	89	6-9
0.80	91	7-9
0.85	92	9
0.90	89	6,9
0.95	92	8
1.00	89	9

C. *RMSProp*

The following figures and tables depicts the results for RMSProp.

1) : With different layer sizes, a learning rate of 0.1 and weights between -0.05 and 0.05

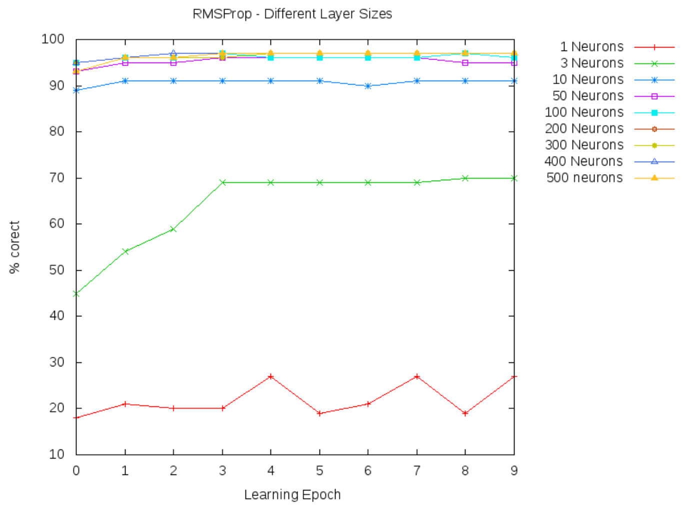


TABLE VI. RMSPROP - DIFFERENT LAYER SIZE

Number of Neurons	Corect classifications(%)	Epoch of achieving
1	27	4,7,9
3	70	8,9
10	91	1-5,7-9
50	96	3-7
100	97	3,8
200	97	4-9
300	97	4-9
400	97	2-9
500	97	3-9

2) : With different ranges of intial weights, a hidden layer size of 300, and learning rate of 0.01

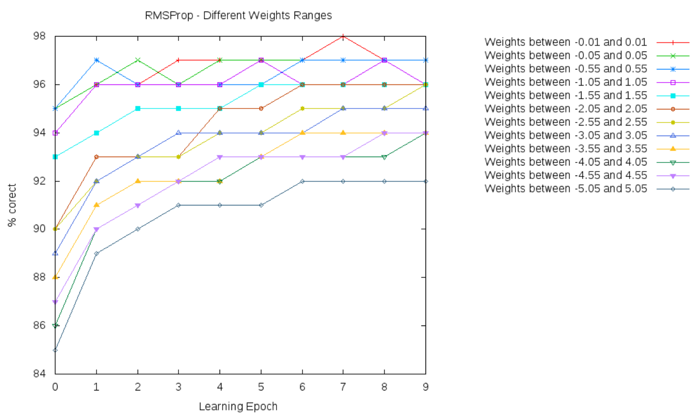


TABLE VII. RMSPROP - DIFFERENT WEIGHTS

Weights limits	Corect classifications(%)	Epoch of achieving
0.01	97	3-9
0.05	97	2,4-9
0.55	97	1,6-9
1.05	97	5,8
1.55	96	5-9
2.05	96	6-9
2.55	96	9
3.05	95	7-9
3.55	94	6-9
4.05	94	9
4.55	94	8,9
5.05	92	6-9

3) : With different learning rates, a hidden layer size of 300 and weights between -0.05 and 0.05

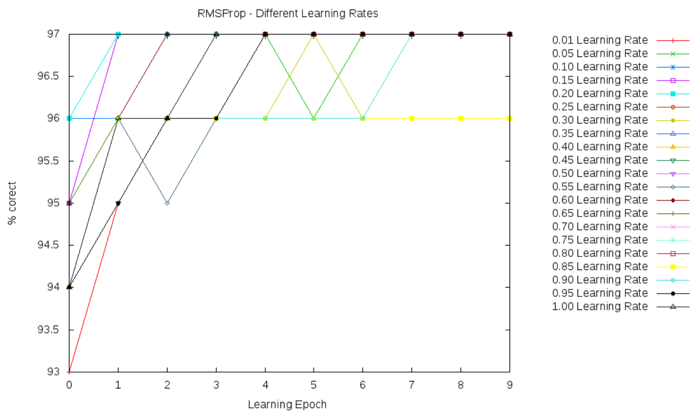


TABLE VIII. RMSProp - Different Learning Rates

Learning Rate	Corect classifications(%)	Epoch of achieving
0.01	97	3-9
0.05	97	3-9
0.10	97	2-9
0.15	97	1-9
0.20	97	1-9
0.25	97	2-9
0.30	97	5,7-9
0.35	97	3-9
0.40	97	4-9
0.45	97	4-9
0.50	97	4-9
0.55	97	7-9
0.60	97	2-9
0.65	97	3-9
0.70	97	3-9
0.75	97	4-9
0.80	97	4-9
0.85	96	1-9
0.90	97	7-9
0.95	97	4-9
1.00	97	3-9

D. RMSProp and Momentum

The following figures and tables depicts the results for RMSProp combined with momentum. There is no paper writen up untill now on how these two interacts together.

1) : With different layer sizes, a learning rate of 0.1 and weights between -0.05 and 0.05

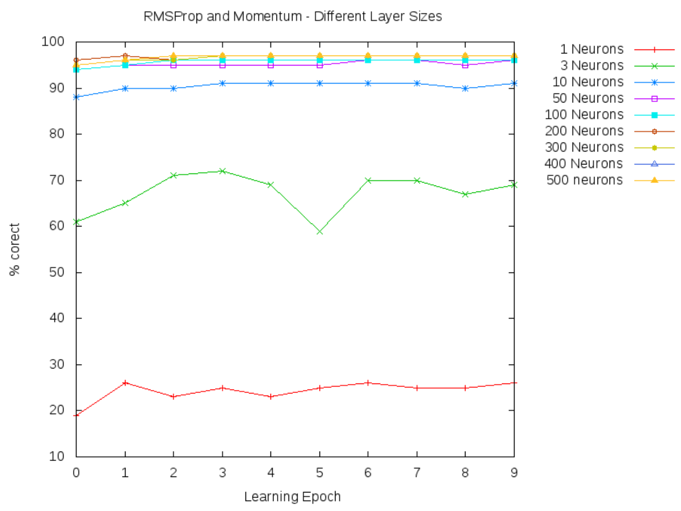


TABLE IX. RMSProp and Momentum - Different Layer Size

Number of Neurons	Corect classifications(%)	Epoch of achieving
1	26	1,6,9
3	72	3
10	91	3-7, 9
50	96	6-9
100	96	2-9
200	97	1,3-9
300	97	2-9
400	97	2-9
500	97	2-9

2) : With different ranges of intial weights, a hidden layer size of 300, and learning rate of 0.01

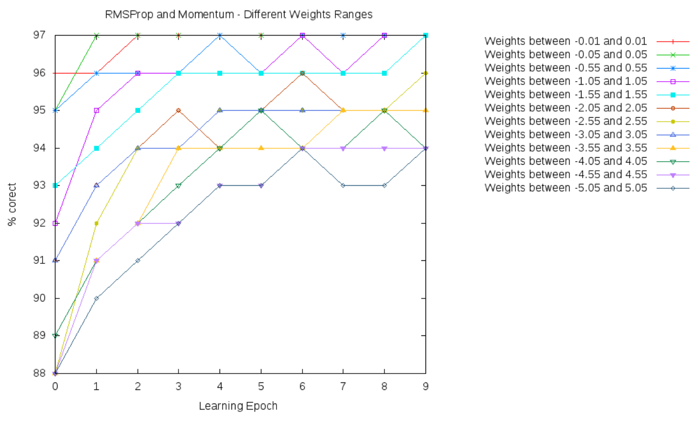


TABLE X. RMSProp and Momentum- Different Weights

Weights limits	Corect classifications(%)	Epoch of achieving
0.01	97	2-9
0.05	97	1-9
0.55	97	4,6-9
1.05	97	6,8,9
1.55	97	9
2.05	96	9
2.55	96	9
3.05	95	4-9
3.55	95	7-9
4.05	95	5,8
4.55	94	6-9
5.05	94	6,9

3) : With different learning rates, a hidden layer size of 300 and weights between -0.05 and 0.05

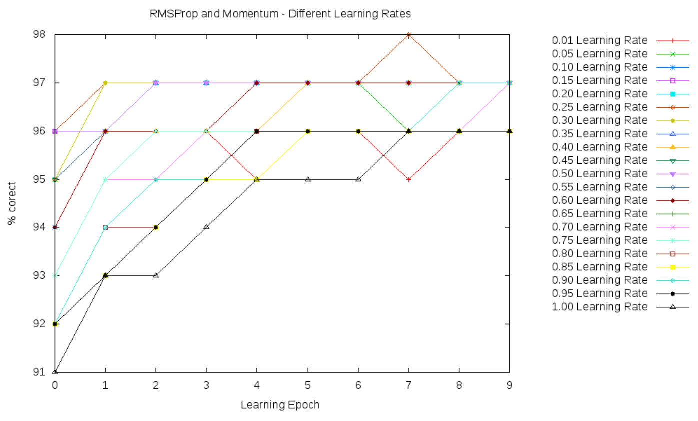


TABLE XI. RMSProp and Momentum - Different Learning Rates

Learning Rate	Corect classifications(%)	Epoch of achieving
0.01	96	3,5,6,8,9
0.05	97	5,6,8,9
0.10	97	2-9
0.15	97	2-9
0.20	97	2-9
0.25	98	7
0.30	97	1-9
0.35	97	2-9
0.40	97	5-9
0.45	97	2-9
0.50	97	2-9
0.55	97	4-9
0.60	97	4-9
0.65	97	9
0.70	97	9
0.75	96	2-9
0.80	96	4-9
0.85	96	5-9
0.90	97	8-9
0.95	96	4-9
1.00	96	7-9

VI. Conclusions

- When analyzing the results, we must take into consideration two factors:
  - classification error
  - convergence speed
- As we have started with a Simple Gradient Decent, we have noticed that for more than 50 neurons on the hidden layer size or for a range of weights higer than [-0.55, 0.55], there are no remarcable diffenerces on the convergence speed or on classification errors. Also, it can be seen that the neural network learns only if the learning rate is within [0.05, 0.15].
- By adding the Momentum, it can not only be noticed that the range of learning rate within the neural network learns is larger, but also that the classification errors are smaller and that the large plateau the learning algorithm cannot escape because the gradients are very small is eliminated.
- When it comes to RMSProp, the differences are remarcable. The results are much better from all points of view: the neural network learns faster and has a better accuracy on the set tests. We can see that a number greater that 50 neurons on the hidden layer don't influence too much the learning process. The learning rate's values that leads to the best results are the medium onces from the test rang. The larger the weigths range is, the bigger the clasification error is.
- The best solution is to combine both Momentum and RMSPorp. Although, compering with the simple RMSProp, there are not any noticeable differences.

VII. Future Work

Our future goal is to study how an agent combining convolutional networks, a form of recurrent neural network (LSTM module) and a QLearning algorithm would behave in an environment which involves planning, following a certain stratey or remembering details that depend only on the state of the environment and cannot be extracted from the input.

The current study helped us understand what and how a classical artificial neural network works and behaves on a certain input, how backpropagation and learning process works and mostly how to tune certain parameters to get the most out of the agent.

We will start with an upgrade of our current neural network to a CNN and we will try experimenting on more complex data sets, for example CIFAR10[15]. The experience gained by doing this study will help us in developing a neural network that will be able to classify and process an input that is more complex and abundant in data (a color image that is larger than 30x30 pixels).

The follow up would be creating a recurrent neural network which will simulate a memory module, the Long Short Term Memory module we will use in training our target agent.

The grand finale would be attaching to this two neural networks an unsupervised learning module based on the QLearning Algorithm that will help it process the best possible action given a certain state of the environment, thus allowing it to behave exactly like or even better than an average human being.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *arXiv preprint arXiv:1502.01852*, 2015.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[3] J. Y.-H. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, "Beyond short snippets: Deep networks for video classification," *arXiv preprint arXiv:1503.08909*, 2015.

[4] K. Xu, J. Ba, R. Kiros, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," *arXiv preprint arXiv:1502.03044*, 2015.

[5] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," *arXiv preprint arXiv:1411.4389*, 2014.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas imminent in nervous activity," *Bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943.

[8] Q. J. Zhang and K. C. Gupta, "Neural networks for rf and microwave design," 2000.

[9] Y. Lecun and C. Cortes, "The MNIST database of handwritten digits." [Online]. Available: http://yann.lecun.com/exdb/mnist/

[15]http://www.cs.toronto.edu/~kriz/cifar.html