Using a Neural Network with ReLU and Softmax activation functions to train a classifier for handwritten digits images

Jason Latouche
School of Computer Engineering
Costa Rica Institute of Technology
Cartago, Costa Rica
96jaslat@gmail.com

Jocelyn Román
School of Computer Engineering
Costa Rica Institute of Technology
Cartago, Costa Rica
jocyroman@kamiracr.com

***Abstract*—In this work we implemented a fully functional neural network from scratch in order to classify handwritten number images from MNIST dataset. By using ReLU activation function and Softmax as probability distribution activation function in the last network layer we want to measure the loss using cross-entropy. Some cool techniques used for this work are dropout, cross-validation, stochastic gradient decent and xavier initialization.**

I. Introduction

Neural networks are computing systems inspired by the biological neural networks that constitute animal brains. Such systems ”learn”, by progressively improving performance, different tasks by considering a bunch of examples about a topic. The goal is to solve problems in the same way that a human brain would.

They have been used to solve problems in almost any field, for instance: image recognition used from gaming to biological medicine. This is because they are very versatile differenciable machine learning algorithm which can optimize multi-variable and non-linearity problems.

There are some frameworks that can be used to implement complex neural networks with ease, however they does not offer the mathematical explanation of how it works and sometimes is hard to start using them without previous knowledge. This is the reason why we are going to implement a fully functional neural network from scratch.

By using neural networks we plan train the model in order to classify images of handwritten numbers of 28x28 pixels. This model will be implemented using Python 3.5 and numpy which is a library used for doing operations with large multi-dimensional arrays and matrices.

After several training sessions some experiments will be done by changing hyper-parameters and visualizing weights of the neural network. This will be done in order to see how the model ”learns” after the training.

II. Methodology

For this work a neural network will be designed from scratch and trained in order to measure the loss with respect to new images not used during the training. In order to implement each feature is important to understand all of them. The rest of this section is going to be dedicated to explain each feature and definition used to design and implement the neural network.

A. Activation layers

In neural networks every activation layer should have defined a function for feed forward estimation and its derivative in order to back propagate the weights error. This means that the functions had to be differentiable.

1) ReLU: Rectified Linear Unit (or ReLU) [1] is an activation function which is defined by:

$$f(x) = \max(0, x)$$

The idea behind is that ReLU takes only the positive argument of the neuron’s inputs, ignoring all negative values.

Its derivative is defined by:

$$f'(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$

It is important to note that f' is not differentiable in $x = 0$. But for this experiment $f'(0) = 0$

2) Softmax: It takes a N-dimensional vector and produces another N-dimensional vector which sum is 1. Softmax is defined by the following per-element formula:

$$S_j = \frac{e^{a_j}}{\sum_k^N e^{a_k}}$$

where j is the j-th vector element and N is the vector size.

Since all the output elements from the function are between $[0, 1]$ and the sum of them is 1, it can be interpreted as a probabilistic distributions. In machine learning is very useful since the highest value can be chosen as the correct one. In [2] are discussed how this function works.

In the neural network the softmax function will be used at the output layer and it will indicate the probability of a Y element to be the correct one.

The softmax derivative is defined as:

$$D_j S_i = \begin{cases} S_i(1 - S_j) & i = j \\ -S_j S_i & i \neq j \end{cases}$$

In [3] the step by step are explained.

B. *One-hot encoded vector*

It is a way to represent categorical labels in one vector [6]. Sometimes called a "*True*" probability distribution. In a one-hot encoded vector the correct class is represented by a 1 meanwhile the rest is 0.

Let say we have M classes that goes from 1 to M. For each correct label y_j let V be our size M vector defined as:

$$V_i = \begin{cases} 1 & i = y_j \\ 0 & i \neq y_j \end{cases}$$

in which case the sum of V is 1.

C. *Loss function: Cross-Entropy*

It is used to compare two discrete probability distributions p and q. It is defined as the following formula:

$$L(p, q) = -\sum_i p_i log(q_i)$$

where p_i is the desired probability and q_i the actual probability.

In this case p_i is a one-hot encoded vector that contains the correct data labels meanwhile q_i is the output vector from the softmax function.

By using the chain rule as shown in [5], is possible to differentiate cross-entropy using softmax as the last layer and use the formula defined by:

$$\frac{\partial L}{\partial W} = \begin{cases} S_i - 1 & i = y_i \\ S_i & i \neq y_i \end{cases}$$

D. *Stochastic Gradient Descent*

It is an algorithm to perform optimization over weights in neural networks. It does a parameter update for each training sample x_i and label y_i

SGD is defined by the following formula:

$$W = W - \eta \cdot \nabla_W \cdot f(W, x, y)$$

where W is the weight the to optimized, η is the learning rate and $\nabla_W \cdot f(W, x, y)$ is the gradient of the objective function evaluated in the weight W, data x and label y.

In [7] they discuss about the behaviour of this algorithm and also it is compared against other algorithms used for parameter optimization.

E. *Xavier initialization*

The idea behind is to initialize the weights so that the neuron activation functions are not starting out in saturated or dead regions. By doing this it more likely that the model will not converge prematurely.

It can be accomplished by ensuring the following formula:

$$Var(W) = \frac{1}{N}$$

where W are the weights and N the size of the input layer. In [8] they show experiments done with and without using this technique.

When using ReLU the half of its input is zero, so it is important to double the size of weight variance by doing:

$$Var(W) = \frac{2}{N}$$

In [9] they explain better the reason why this should be done.

F. *Dropout*

It is a simple regularization technique for reducing overfitting in neural networks by preventing adaptations on training data. It is also a way of performing model averaging on neural networks.

This technique works by dropping out or "ignoring" neurons by a factor of p for each of the learning stages during all the training session.

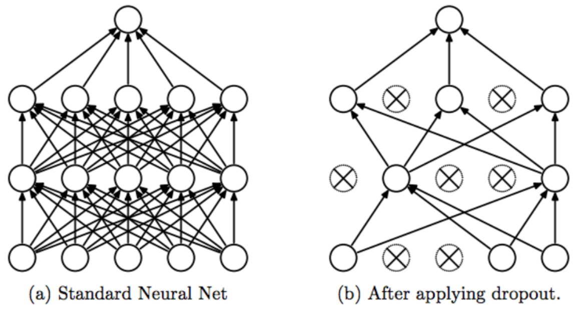


Fig. 1: Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014 [10]

In [11] they describe the technique, considerations and results that were obtained with different datasets.

G. *Cross validation: Holdout method*

It is a model validation technique used to prevent overfitting [12] and give an idea on how the model will generalize to an independent dataset. By doing cross-validation is possible to measure how much the model generalize to new data that was not used during training.

In the holdout method the data is splitted into training data and validation data. The training data will be used to train the model and then the model is evaluated using the validation data only in order to see how the model behave with new unseen data.

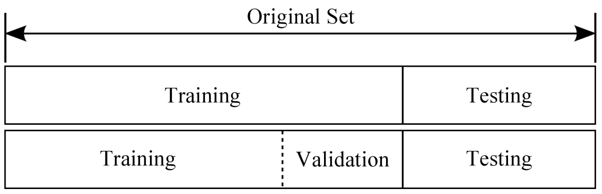


Fig. 2: Cross-Validation diagram taken from [13]

With the holdout method the training and validation set are randomly selected from the data. The size of each of the sets can be chosen arbitrary although typically the validation set is smaller than the training set. For this work 80% and 20% where used for each respectively.

H. MNIST Data set

The Modified National Institute of Standards and Technology database (better known as MNIST dataset) [14] is a large database of handwritten digits that is commonly used for training image processing systems and machine learning algorithms. This dataset contains 60,000 training images and 10,000 testing images of 28x28 pixels each.

Fig. 3: MNIST samples from 0 to 9

III. Experiments

A. One vs Two hidden layers

By training the neural network using only the training data we want to evaluate the model with the test data by measuring the loss and the accuracy. Each of the learning sessions will be done using the following settings:

- Dropout of 50%
- Learning rate of 0.0085
- Cross-Validation of 80%/20% for training and validating respectively
- Mini batches of 32 samples (1875 iterations per epoch)
- Four epochs

A one-hidden layer and two-hidden layers neural network will be trained using the following layer sizes:

- 128
- 256
- 512
- 1024
- 2048

Some graph taken from the learning session will be shown in order to compare how fast a model can found an optimal solution and how much it generalize the knowledge.

B. Handwritten number images manually made

By using the best performed neural network from the experiment above, we want to use it to classify handwritten number images not used neither in the training or testing set. The images were made using a painting software and re-sizing to 28x28 pixels with help of a library for Python.

C. Feed backward

By using the worst generalized model from the first experiment, we want to visualize what it learned. In this experiment our goal is to use a one-hot vector as an input from the last layer and, by using matrix dot product properties, feed backward the "correct class" in order to get in image in the first layer of the neural network representing that class.

IV. Results

A. Experiment #1: Layers comparison

The following tables are the results of the training sessions.

TABLE I: One hidden layer

Layer size	***Loss***	***Accuracy***
128	0.28978	0.9202
256	0.27435	0.9235
512	0.26734	0.9278
1024	0.25647	0.9288
2048	0.24820	0.9316

TABLE II: Two hidden layer

Layer size	***Loss***	***Accuracy***
128	0.23520	0.9316
256	0.21163	0.9369
512	0.19700	0.9426
1024	0.17729	0.9496
2048	0.16562	0.9509

The following images are plots representing the learning session for some sample neural networks.

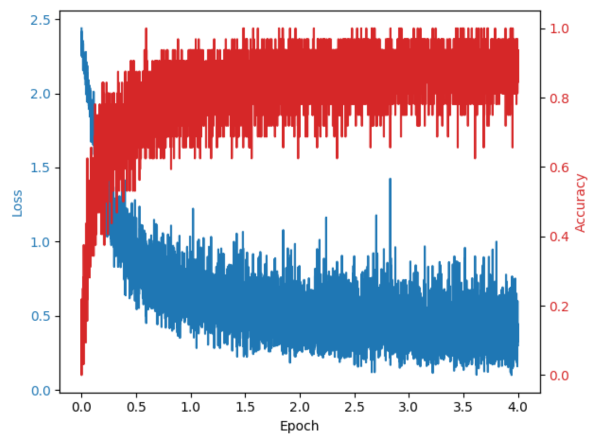


Fig. 4: Learning session from the 128 one-hidden layer model

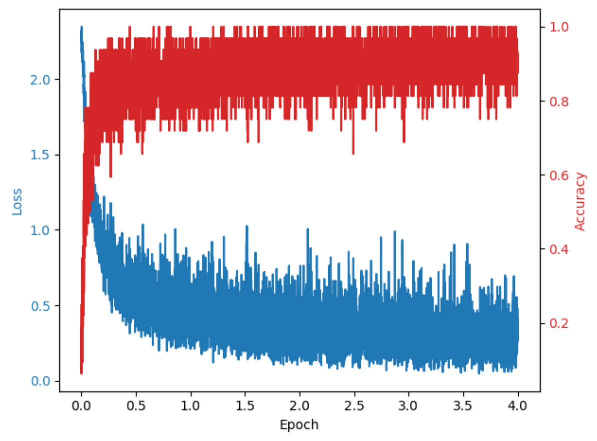


Fig. 5: Learning session from the 2048 one-hidden layer model

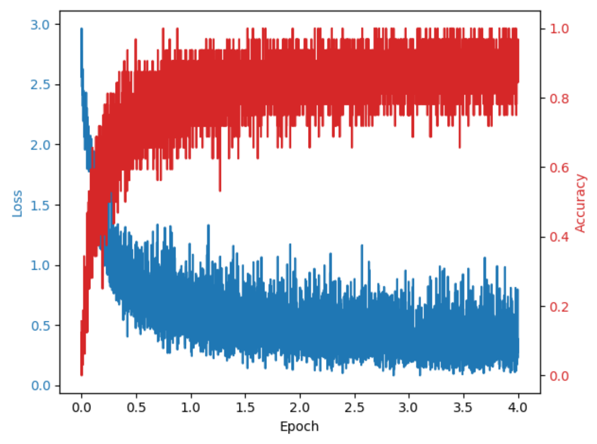


Fig. 6: Learning session from the 128 two-hidden layer model

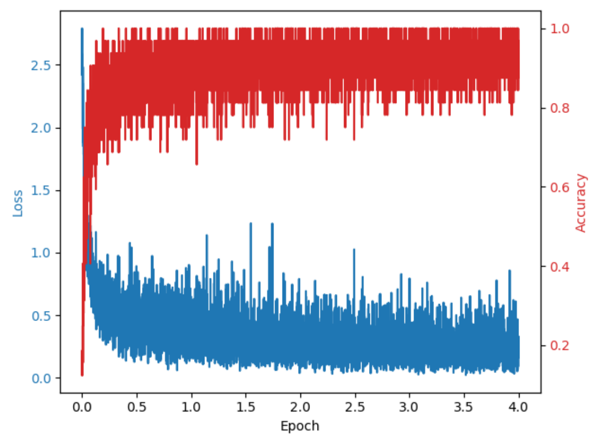
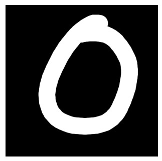


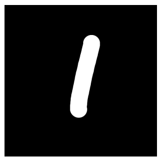
Fig. 7: Learning session from the 2048 two-hidden layer model

B. Experiment #2: Handwritten number images manually made

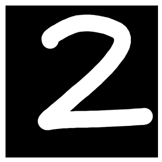
The following images were created using a painting software. Each of them were evaluated using the neural network model. Below each can be found the given classification.



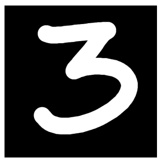
(a) Classified as 0



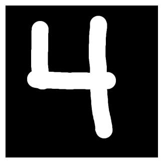
(b) Classified as 1



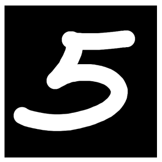
(a) Classified as 2



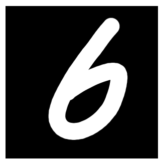
(b) Classified as 3



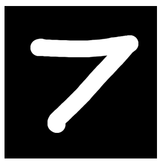
(a) Classified as 4



(b) Classified as 5

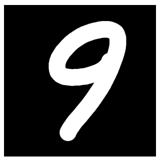


(a) Classified as 8



(b) Classified as 7

(a) Classified as 3


(b) Classified as 8

C. Experiment #3: Feed backward

The following image is the result of passing a one-hot vector for each class and feed back the neural network to obtain a 28x28 pixel image.

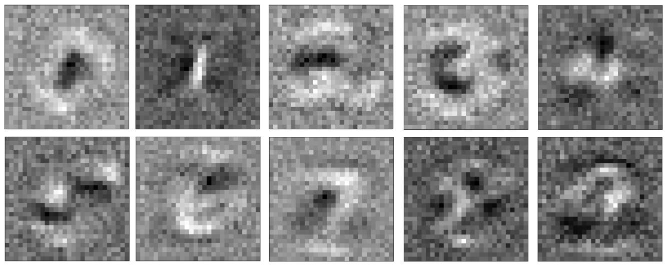


Fig. 13: Images from 0 to 9 as a result from experiment #3

V. Discussion

As can be seen from the first experiment, even the 2048 one-hidden layer is not good enough to achieve better loss and accuracy than 128 two-hidden layers neural network.

By analyzing the plots, we can see that using more neurons leads to optimize faster the model. It also seems that using more layers leads to a more generalized solution meanwhile less layers tends to overfit the model.

Using the last idea, the best model to test the handwritten number images is the 2048 two-hidden layers neural network. As seems in the results, not all of the numbers were correctly classified, giving an accuracy of 0.7.

The reason in the mistaken results given from the model could be explained using the number shape. Take for instance the number 6 (misclassified as 8) where the big circle below could confuse the model. Or the number 8 (misclassified as 3) where the right half image can form a 3 no matter what is in the left half of the image.

In the third experiment some ”learned” images were visualized in order to see how the model ”imagines” numbers. This experiment was made using the 128 one-hidden layer model, which is also the more overfitted one. As can be seen from the images the model can learn where should be black and white for every class.

By taking the number 3 for instance, is possible to see how there are some black places where nothing should be in there, but also some whites where the lines for the number 3 can be drawn. Around the number some grays can be seen. It probably means that the model does not optimized anything in there and also does not care too much about what could be drawn in there.

References

[1] Sharma, A. Understanding Activation Functions in Neural Networks. *Medium*, 2017. [Online]. Available: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0 [Accessed: 14-May-2018].

[2] Yang, J. ReLU and Softmax Activation Functions. *GitHub*, 2017. [Online]. Available: https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functions. [Accessed: 14-May-2018].

[3] Dahal, P. Classification and Loss Evaluation - Softmax and Cross Entropy Loss, *DeepNotes*, 2017. [Online]. Available: https://deepnotes.io/softmax-crossentropy. [Accessed: 14-May-2018].

[4] Bendersky, E. The Softmax function and its derivative - Eli Bendersky's website. *Eli.thegreenplace.net*, 2016. [Online]. Available: https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/. [Accessed: 14-May-2018].

[5] Buzau, M. Gradient descent on a Softmax cross-entropy cost function, *Madalinabuzau.github.io*, 2016. [Online]. Available: https://madalinabuzau.github.io/2016/11/29/gradient-descent-on-a-softmax-cross-entropy-cost-function.html. [Accessed: 14-May-2018].

[6] Vasudev, R. What is One Hot Encoding? Why And When do you have to use it?, *Hacker Noon*, 2017. [Online]. Available: https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c6186d008f. [Accessed: 15-May-2018].

[7] Ruder, S. An overview of gradient descent optimization algorithms, *Sebastian Ruder*, 2016. [Online]. Available: http://ruder.io/optimizing-gradient-descent/. [Accessed: 15-May-2018].

[8] Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proc. AISTATS*, volume 9, pp. 249–256, 2010.

[9] Jones, A. An Explanation of Xavier Initialization, *andy's blog*, 2015. [Online]. Available: http://andyljones.tumblr.com/post/110998971763/an-explanation-of-xavier-initialization. [Accessed: 15-May-2018].

[10] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. J. Machine Learning Res. 15, 1929–1958 (2014).

[11] Budhiraja, A. Learning Less to Learn Better - Dropout in (Deep) Machine learning, *Medium*, 2016. [Online]. Available: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5. [Accessed: 15-May-2018].

[12] Gupta, P. Cross-Validation in Machine Learning – Towards Data Science, *Towards Data Science*, 2017. [Online]. Available: https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f. [Accessed: 15-May-2018].

[13] d. Charles-Antoine. RPubs - Cleaning the Titanic Data, *Rpubs.com*, 2017. [Online]. Available: http://www.rpubs.com/charlydethibault/348566. [Accessed: 15-May-2018].

[14] LeCun, Y. Cortes, C. and Burges, C. MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges, *Yann.lecun.com*. [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed: 15-May-2018].