A Biophysically Inspired Neural Network

Robin Thériault and Paul François

Department of Physics, McGill University, Montréal, QC Canada H3A 2T8

This manuscript was compiled on January 30, 2021

Neural networks are now applied everywhere in industry. However, they suffer from two major weaknesses: lack of interpretability and vulnerability to perturbations in the data. We investiagate the reasoning of a neural network of our design that shows promise in these areas. Inspired by the Weber-Fechner law (1), it uses log as a activation function at the first layer.

Biophysics | Numerical modeling | Machine learning | Neural networks

In 1949, Donald Hebb opened the door to the modeling of systems of neurons by presenting his famous rule for reinforcement and inhibition of neural connections, often summarized as "neurons that fire together wire together" (2). His idea was soon implemented numerically by Rosenblatt in what is arguably the first artificial neural network: the linear perceptron (3). Rosenblatt's model quantifies the likelihood that a data vector belongs to a class by taking the dot product with a weight vector. The weight vector, analog to connections between neurons, is learned according to a variation of Hebb's rule. The hope is that the trained weight should match salient patterns of the data. It turns out that the linear perceptron can classify data points separated by a linear boundary (See figure 1), but fails completely on complicated data sets. A wave of pessimism and disinterest rolled on the AI community as this limitation was uncovered (4). Decades later, Neural networks were rehabilitated and optimized by researchers such as John Hopfield (5) and Kunihiko Fukushima (6). Their innovations take their roots in biological ideas, with Hopfield looking for a model of memory and Fukushima taking inspiration from the visual cortex.

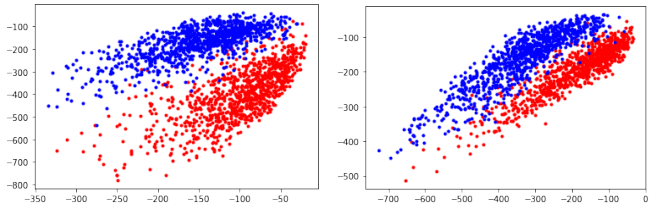

Fig. 1. Example of data that a linear perceptron can classify.

A modern artificial neural network is a sequence of layers that map data to categories. Each of these layers consists of a matrix of weights followed by a non-linear activation function (see Figure 3). Neural networks learn patterns in the data by minimizing the error as a function of the weights, which are thus an abstract representation of the data. Similarly, we can fool a trained neural network by perturbing input data such that error is maximized. Neural networks can do complex tasks such as recognize handwritten characters, diagnose tumors or solve quantum many-body problems (7). Despite their accuracy, they learn blurry and diffuse features of the data (See figure 3), which makes their reasoning hard to track. They are also vulnerable to perturbations in the data that appear meaningless to us (see Figure 4). With the intent of solving these problems, we took a page from biology's book and designed a new type of neural network that uses log as first activation function. The inspiration comes from the Weber-Fechner law (1). This model of perception suggests that the brain estimates various quantities such as weight, brightness, loudness and size based on their position on a logarithmic scale. Our network learns interpretable prototypes of digits (See figure 2) and is resistant to perturbations that fool traditional networks. Our results are complementary to the work of Krotov and Hopfield, who are doing prototype learning with a neural network using a polynomial as activation function (8).

Fig. 2. Weights learned by the log network. Note that they look like handwritten digits.

The present report starts with a presentation of the log network along with the procedure used to train it and of numerical methods employed to improve its interpretability. We continue with a quick discussion of non-linearity in neural networks and how it helps to learn complex representations of data. We then uncover the reasoning of our network by comparing the values and roles of different weights. Finally, we derive a analytic expression for the effect of perturbations and investigate it quantitatively and qualitatively to gain insight concerning robustness.

Results and Discussion

Structure of the Network and Learning Procedure.

Significance Statement

Artificial neural networks are computer programs used for classification of data. For instance, they can identify handwritten digits with high accuracy. However, the reasoning leading them to a decision is hard to track and, unlike humans, they can be fooled by adding weak perturbations to the data. We try to fix these issues by taking inspiration from a model for human perception of sound and brightness (1).

Paul François supervised the work of Robin Thériault for the duration of the project.

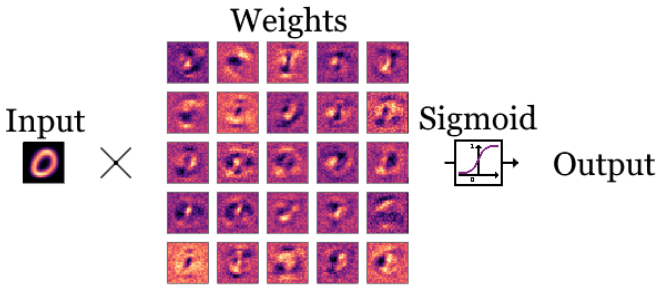

Fig. 3. Schematic of the first layer of a neural network for classification of handwritten digits. Output is calculated by applying $\mathrm{sigmoid}(x) = \frac{1}{1+e^{-x}}$ to the projection of the input on the weights. It is then used as input for the next layer. Note the fuzzy appearance of weights.

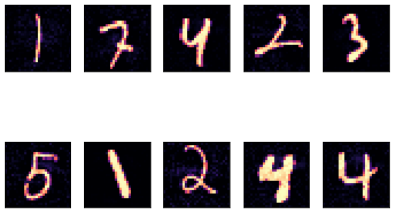

Fig. 4. Perturbations that fool traditional neural networks. Note their small scale and noisy appearance.

We implement our network using Keras, a python machine learning library (9). We train on the MNIST data set of handwritten digits (10). Images x_1 are vectors with N components, their pixels indexed as $x_1[k]$. Our network has two layers with weight matrices w_1 and w_2. Its columns $w_1[j]$ and $w_2[i]$ are weight vectors like the one used in the linear perceptron. The first layer calculates the hidden activations x_2 to be $x_2[j] = \log\left|\sum_k w_1[j,k]\ x_1[k]\right|$. The second layer computes the output activations $x_3[i] = \sum_j w_2[i,j]\ x_2[j]$, which quantify the likelihood that x_1 belongs to each class i. Given a set of images x_1 of class y, we aim to learn weights that minimize the error function $-x_3[i=y]$. The network should then correctly predict that image x_1 belongs to class y, i.e. $y = \mathrm{argmax}_i(x_3[i])$.

We can train the network by minimizing $-x_3[i=y]$ directly without additional constraints. However, the learned $w_1[j]$ weights are then on different orders of magnitude, which in turns affect the size of the $w_2[j]$ weights. The role of each coefficient of w_2 is then hard to interpret. We solve this issue by forcing the coefficients of w_1 to stay between 0 and 1. At each step of the training algorithm, we take the absolute value of coefficients that become negative and map to 1 weights that get bigger than 1. We also normalize by the number of pixels before taking the log and force the values of w_2 to stay positive. Constraints give $x_2[j] = \log\left(\frac{1}{N}\sum_k w_1[j,k]\ x_1[k]\right)$, and we can summarize the action of the two layers as $x_3[i] = \sum_j w_2[i,j]\log\left(\frac{1}{N}\sum_k w_1[j,k]\ \ x_1[k]\right) = \log\left(\prod_j\left(\frac{1}{N}\sum_k w_1[j,k]\ x_1[k]\right)^{w_2[i,j]}\right)$. However, we still have to look at prototypes to determine their classes. But it would be handy for our interpretability analysis to have a control network where we know the classes from the start. We use the following heuristic procedure to train such a control network:

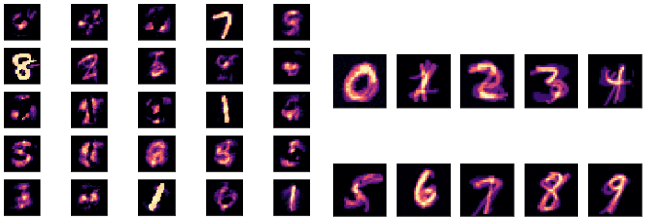

Fig. 5. First layer weights learned by training the two layers at the same time (left) and separately using the heuristic procedure (right). Note that they look similar.

- First divide the data set x_1 into small sets of images that belong to the same class.
- The network starts off as a single layer with weights $w_1[j]$ constrained to stay between 0 and 1.
- Minimize the error function $-x_2[j]$ to learn weights. Train each weight on a different set of digits.
- For the present report we train weights $w_1[0]$, $w_1[10]$, $w_1[20]$,.. on sets with label 0; $w_1[1]$, $w_1[11]$, $w_1[21]$,.. on sets with label 1, etc.
- Freeze w_1 so that it's not modified during the next step of the training procedure
- Add the second layer to the network.
- Given images of classes y, Minimize $-x_3[i=y]$ to learn w_2. Train on the whole dataset.

Note that $-x_2[j]$ is minimized when the corresponding weight $w_1[j]$ is more or less the average of images of x_1, hence the prototypes. The fact that we get qualitatively similar weights when training the two layers at the same time (see Figure 5) can be explained by noting that $-x_3[i] = -\sum_j w_2[i,j]\log\left(\frac{1}{N}\sum_k w_1[j,k]\ x_1[k]\right) = \sum_j w_2[i,j](-x_2[j])$, so minimizing $-x_3[i]$ accounts to minimizing $-x_2[j]$ for all j plus choosing the weights $w_2[i]$ that reinforce appropriate values of $x_2[j]$.

Error functions are minimized by using the gradient descent algorithm. In short, gradient descent trains weights w by iteratively subtracting the gradient of the error function $f(w,x)$ rescaled by the learning rate λ: $w = w - \lambda\nabla_w f(w,x)$. A big learning rate means that the network will quickly learn coarse grain values for the weights. On the other side, a small learning rate will make the network slowly learn subtle details. Additionally, weights can be regularized by iteratively subtracting a small number during learning. We trained a control network with a constant learning rate and no regularization and another with a gradually decreasing learning rate and a small regularization. Note that the process of perturbing data is similar to gradient descent. Only this time we add $\nabla_{x_1} f(w,x)$ to the data.

About Linearity and Non-Linearity of Neural Networks.

A linear network is a neural network without activation functions. For example, a linear neural networks with two layers has output $x_3[i] = \sum_j w_2[i,j]\ \sum_k w_1[j,k]\ x_1[k]$. We will now "linearize" trained neural networks by removing

their activation functions and use the change in their accuracy to measure non-linearity. Take a "relu" neural network that uses the relu function $\max(0, x)$ at the first layer. This design is very common, often serving as a basic building block for more complicated networks with multiple layers. The accuracy of this network drops from 0.966 to 0.895 when it's linearized, indicating that it's almost linear. On the other side, the log network is strongly non-linear, with its score dropping from 0.9178 to 0.0958. This can be attributed to the non-linearity of exponentiation and product of projections in $\prod_j \left(\frac{1}{N}\sum_k w_1[j,k]\ \ x_1[k]\right)^{w_2[i,j]}$. In fact, this expression is equivalent to the output $x_3[i] = \log\left(\prod_j \left(\frac{1}{N}\sum_k w_1[j,k]\ \ x_1[k]\right)^{w_2[i,j]}\right)$ because the network determines the correct class based on the maximum of x_3, which log doesn't change. In short, we can look at the action of the network both from a "ordinary space" and from a "log space" standpoint.

The model of Krotov and Hopfield (8) also used non-linear operations in the form of its polynomial activation functions. We think that it's not a coincidence and that non-linearity in neural networks favorizes the learning of complex representations of data such as prototypes.

Understanding How the Log Network Thinks.

We first look at how the control network associates prototypes with classes by examining the second weight matrix w_2 of the control network. It doesn't have a clear structure if the learning rate is constant during training (See figure 6), which means that the network associates classes equally to all prototypes. On the contrary, if we use a decreasing learning rate and weak regularization, then w looks like a sawtooth, which means that $w_2[i,j]$ is maximal when prototype j belongs to class i (See figure 7). This fact tells us that the log network associates prototypes with their corresponding labels. The gain in interpretability of w_2 makes perturbations necessary to fool the network look more intense (See figures 6 and 7). It also means that we can predict the classes of the prototypes learned by the non-control network based on the coefficients of w_2 (See figure 8).

Fig. 6. Non-sawtooth w_2 for constant learning rate and no regularization (right) with corresponding perturbed digits (left).

We feed images x_1 with labels m and n to the control network and record $x_2[j]$ for two prototypes with the same labels. The resulting scatter plots show two clusters distributed about parallel lines: one for m and one for n. Since the clusters are linearly separated (See figure 10), we conclude that the log network already discriminates digits at the first layer. What remains is a simple linear classification problem, which can be solved by a linear perceptron, a role fulfilled by the second layer. In fact, figure 1 actually shows values of x_2. The fact that clusters are parallel and don't simply look like blobs is important as it narrows down the set of possible separation lines. This property is not observed for a relu network (See figure 9).

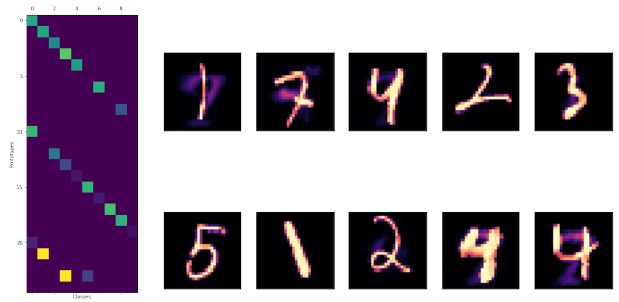

Fig. 7. Sawtooth w_2 for constant learning rate and no regularization (right) with corresponding perturbed digits (left). Perturbations are meaningful and correspond to features of the class predicted by the network. For instance, the network sees the first image as a 7, the second as a 9, the next to last as a 9 and the last as a 7.

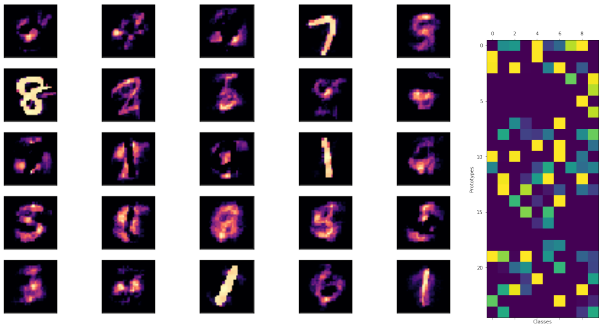

Fig. 8. First prototypes of the control network and corresponding slice of w_2. We can predict the classes of prototypes based on the coefficients of w_2 with a varying degree of success. For instance, the prototype of 8 in the second row corresponds to a yellow square in the fifth row and eighth column of w_2.

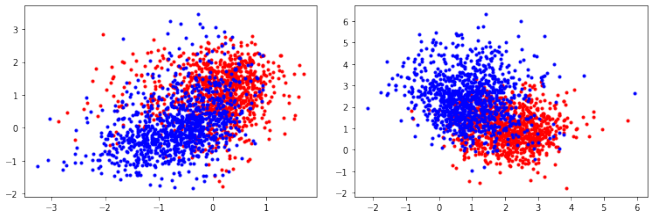

Fig. 9. First layer dot products for the relu network. Note that clusters look like intertwined blobs, so most of the classification work must be done at the last layer.

We fit lines to clusters of digits. Residuals look overall random (See figure 11), so we conclude that the fits are reasonable. The equations for them gives $\log\left(\sum_k w_1[n,k]\ x_1[k]\right) = a\log\left(\sum_k w_1[m,k]\ \ x_1[k]\right) + b \Rightarrow \sum_k w_1[n,k]\ \ x_1[k] = b\left(\sum_k w_1[m,k]\ \ x_1[k]\right)^a$, the first layer projections follow a power law in ordinary space. We hypothesize the power law dependency to be a feature of simple data sets like MNIST.

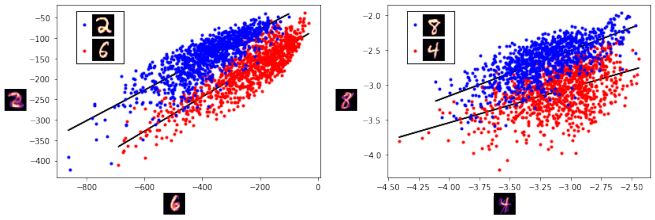

Fig. 10. Hidden layer activations x_2 for the log network. Clusters spread along fitted parallel lines are linearly separated with low (left) or relatively high (right) uncertainty

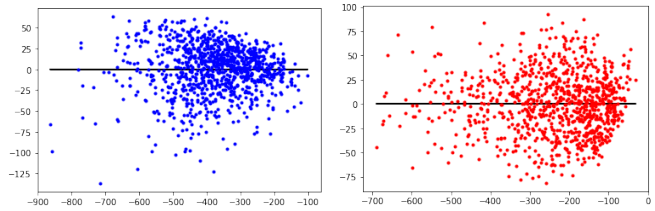

Fig. 11. Residuals for the first plot of figure 10. Data points are scattered randomly except for the small slope on the top of the left plot.

Effect of Perturbations. Let's add a perturbation δx_1 to the image x_1. We'll calculate the perturbed output x_3' and its deviation from the unperturbed output $\delta x_3 = x_3' - x_3$ for both a linear network and the log network. Dot products are linear, so $\sum_k w[j,k]\ (x[k] + \delta x[k]) = \sum_k w[j,k]\ x[k] + \sum_k w[j,k]\ \delta x[k]$. Thus, for a linear network, δx_3 is simply a linear function of δx_1:

$$\delta x_3[i] = \sum_j w_2[i,j] \sum_k w_1[j,k]\ \delta x_1[k]$$

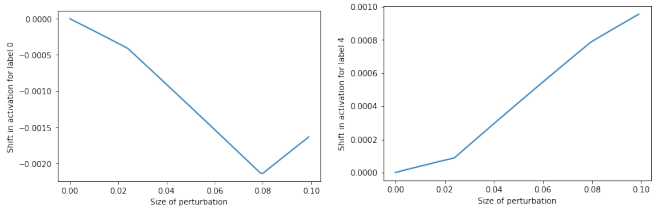

Fig. 12. The change in the output activation of a relu network for small random perturbations around a digit. Note the linear behavior.

Note that x_3' changes linearly if we add multiple perturbations iteratively. This linearity makes calculating perturbations very easy because we just have to calculate the gradient once and stick with it. Recall that traditional neural network are pretty much linear, so this result is a good approximation for them (See figure 12). For plots, we use a scale such that a perturbation of size 1 corresponds to the brightest pixels of the original image.

We'll now calculate $\delta x_3[i]$ for the log network. We'll use $x_3[i] = \prod_j \left(\frac{1}{N}\sum_k w_1[j,k]\ x_1[k]\right)^{w_2[i,j]}$ for the derivation, but we'll plot predictions versus numerical results in log space for numerical stability. See appendix for details.

$$\delta x_3[i] = x_3[i] \sum_j w_2[i,j] \frac{\sum_k w_1[j,k]\ \delta x_1[k]}{\sum_k w_1[j,k]\ x_1[k]}$$

x_3' does changes non-linearly after multiple small perturbations. In fact, $\delta x_3[i] \propto x_3[i]$ means that $x_3 \propto \exp(t)$ for t iterations if the rest of the expression is approximately constant. The quantitative impact of the $\sum_k w_1[j,k]\ x_1[k]$ term is not so clear, but it certainly contributes to the non-linearity. Qualitatively, it means that prototypes that have a big dot product with x_1 will propagate perturbations with more difficulty than prototypes with a small dot product. The non-linear behavior of the predicted change in x_3 is consistent with the fact that it agrees with numerical plots only within a relatively small interval (See figure 13).

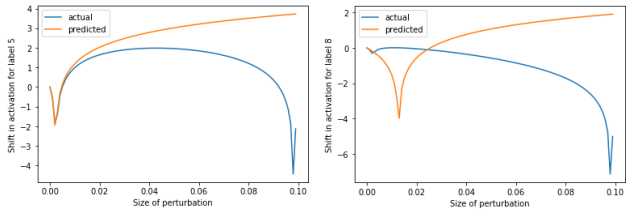

Fig. 13. The change in the output activation of the log network for small random perturbations together with the prediction $\log\left(1 + \frac{\delta x_3[i]}{x_3[i]}\right)$ of the equation. We see that predicted values are consistent with numerical results only over a small interval.

Since the response of the log network to perturbations is non-linear, the gradient $\nabla_{x_1} f(w,x)$ with respect to the input becomes a bad approximation for the optimal perturbations more quickly than for the relu network. Because of that, perturbations are harder to calculate and have a more complicated structure.

Conclusion

The log network has three main advantages over standard neural network. First, the role of its weights can be inferred by a simple visual analysis. We believe that it could be used to gain insight for various problems solved by neural networks. Second, its robustness to perturbations makes our network reliable. For example, it would resist to perturbations added to road signs to fool self-driving cars using standard neural network. Third, it's inspired by biology.

We have two goals for the future. First, we will look for similarities between the log network and biological decision models such as kinetic proofreading (11). Second, we will modify its learning scheme to improve performance and resistance to perturbations and make it a more serious alternative to modern neural networks.

Appendix

The perturbed activations are given by:

$$x_3'[i] = \prod_j \Big(\frac{1}{N} \sum_k w_1[j,k]\ x_1[k] + \frac{1}{N} \sum_k w_1[j,k]\ \delta x_1[k] \Big)^{w_2[i,j]}$$

Let $\epsilon[j]$ be the ratio of the projections of δx_1 and x_1 on weight j: $\epsilon[j] = \frac{\sum_k w_1[j,k]\ \delta x_1[k]}{\sum_k w_1[j,k]\ x_1[k]}$. The perturbed activations can then be written:

$$\begin{aligned} x_3'[i] &= \prod_j \Big(\frac{1}{N} \sum_k w_1[j,k]\ x_1[k] \big(1 + \epsilon[j]\big) \Big)^{w_2[i,j]} \\ &= x_3[i]\ \prod_j \big(1 + \epsilon[j]\big)^{w_2[i,j]} \end{aligned}$$

Take perturbations to be small, then $\epsilon[j]$ is small and we have $x_3'[i] \approx x_3[i]\ \big(1 + \sum_j w_2[i,j]\ \epsilon[j]\big)$. Taking $\delta x_3[i] = x_3'[i] - x_3[i]$:

$$\delta x_3[i] = x_3[i]\ \sum_j w_2[i,j]\ \frac{\sum_k w_1[j,k]\ \delta x_1[k]}{\sum_k w_1[j,k]\ x_1[k]}$$

Author Affiliations. McGill, Department of Physics, Ernest Rutherford Physics Building, 3600 University Street, Montréal, QC H3A 2T8

ACKNOWLEDGMENTS. I would like to thank Paul François and Thomas Rademaker for their precious feedback and ideas.

1. Fechner GT, Howes DH, Boring EG (1966) *Elements of psychophysics.* (Holt, Rinehart and Winston New York) Vol. 1.
2. Hebb DO, Hebb D (1949) *The organization of behavior.* (Wiley New York) Vol. 65.
3. Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review* 65(6):386.
4. Marvin M, Seymour P (1969) Perceptrons.
5. Hopfield JJ (1982) Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences* 79(8):2554–2558.
6. Fukushima K (1980) Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics* 36(4):193–202.
7. Carleo G, Troyer M (2017) Solving the quantum many-body problem with artificial neural networks. *Science* 355(6325):602–606.
8. Krotov D, Hopfield JJ (2016) Dense associative memory for pattern recognition in *Advances in neural information processing systems.* pp. 1172–1180.
9. Chollet F, , et al. (2015) Keras (https://keras.io).
10. Deng L (2012) The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine* 29(6):141–142.
11. Hopfield JJ (1974) Kinetic proofreading: a new mechanism for reducing errors in biosynthetic processes requiring high specificity. *Proceedings of the National Academy of Sciences* 71(10):4135–4139.