# Explaining Artificial Neural Networks

Martin Damyanov Aleksandrov
Freie Universität Berlin
martin.aleksandrov@fu-berlin.de

***Abstract*— We look at neural network financial applications such as predicting market trends and automated trading, in which explaining the network dynamics is of utmost importance. To do so, we propose a novel method that implements second-order backpropagation for neural networks. We explain how to derive all layer-local derivatives for common layer types. Finally, we conduct experiments and confirm that our method enables faster calculation of explanations than state-of-the-art methods.**

**Index Terms— Neural Networks, Explainable AI, Optimization.**

## I. INTRODUCTION

In the enigmatic world of finance, where rapid fluctuations and market volatility are par for the course, having a reliable way to predict these changes is akin to possessing a secret weapon. Over recent years, this invaluable weapon has materialized in the form of artificial neural networks (ANNs), a technology heralding a seismic shift in financial market operations. At their most fundamental level, ANNs are designed to emulate the inner workings of the human brain. They are algorithms or mathematical models that replicate our neural decision-making process, thereby offering a solution to problems beyond the scope of traditional statistical tools. ANNs thrive on data, learning from it, much like how hu- mans learn from experiences. They decipher intricate patterns within data sets, processing and interpreting complex arrays of information with remarkable precision. For example, the Indian financial market, marked by its distinct dynamics and fast-paced activity, presents an ideal setting for applications of ANNs. We give two common applications from it:

**Application 1 (Predicting Trends)**: Traders and investors no longer need to make decisions in the dark or rely on gut feelings. ANNs can digest historical market data and generate accurate forecasts about stocks and trends instead of them, thereby lending traders and investors an edge over their competitors. Whether it is investing in the booming tech industry or hedging bets on manufacturing stocks, ANNs can offer unprecedented predictive power.

**Application 2 (Automated Trading)**: As high-frequency and algorithmic trading gains traction in the Indian financial market, ANNs play an instrumental role. They pore over vast amounts of market data in real time, processing it to identify potentially lucrative trading opportunities. This information forms the backbone of automated trading systems, enabling them to execute trades at speeds human traders cannot match.

Despite their potential for predicting trends and automated trading, ANNs impose challenges. For example, they normally demand considerable computational resources and data for training. Without a robust infrastructure and the requisite data volumes, the effectiveness of ANNs could be significantly compromised. But, in financial markets, resources and data are arguably available, so this challenge is less concerning in such markets. However, as ANNs operate as 'black boxes', their internal workings and decision-making processes remain obscure. While they deliver impressive results, understanding the reasoning behind these outcomes can prove challenging. This lack of transparency may hinder their adoption. As a response, we propose a novel approach for generating ex- planations for ANN dynamics by using their entire Hessian. Before describing our approach, we review works relating to explaining neural networks and Hessian computation.

## II. RELATED WORK

A nice survey on explaining neural networks was given in [1]. However, none of the works there proposed to identify meaningful ANN regions for generating explanations through the exact Hessian computation, as we do here. As for Hessian computations, Mizutani, Dreyfus, and Demmel [2] proposed the first scheme exploiting the layered structure of ANNs to calculate the exact Hessian of a multi-layer-perceptron (MLP) network with sigmoid activations using only local first- and second-order derivatives that can be calculated during the feedforward step. Furthermore, Dangel, Harmeling, and Hennig [3] expanded their scheme to propose a modular way of calculating the diagonal blocks of the Hessian, culminating in the creation of the noteworthy BackPACK for PyTorch [4]. While this tool can be used to calculate exactly the diagonal blocks of the Hessian [5], off-diagonal blocks cannot be computed using BackPACK. As a response, we advance the state of the art in exact Hessian computations and use these advances to explain network dynamics.

## III. RESULT SUMMARY

We next summarize our contributions:

1) We describe an approach to generate explanations of the network dynamics (Section IV). This requires computing the entire Hessian (Section V).

2) We derive equations required to make the algorithm from [2] for Hessian computation compatible with common building blocks in modern ANNs (Section VI).
3) We run experiments to confirm that the extended algorithm outperforms current state-of-the-art algorithms in terms of speed (Section VII).

## IV. Generating Explanations

We now describe our approach. More formally, given an $M$-layer ANN $f$ parameterized by $\boldsymbol{w} \in \mathrm{R}^N$, we let $\boldsymbol{x}$ denote the input node vector and $\boldsymbol{y}$ denote the label node vector of it. Furthermore, we let $L$ stand for the loss function of $f$. This measures how good is the model when performing the task encoded by $f$. Normally, we want to minimize the value of the loss function $L$ through backpropagation to improve the predictions. However, this tells us little about explaining the network dynamics or how predictions depend on these. In response, we propose to identify hidden nodes (i.e. regions in the loss function), which can lead to small dynamic errors in the predictions, thus justifying (explaining) these predictions robustly and reliably. Indeed, we cannot expect that hidden nodes, which can lead to large dynamic errors in the predictions can robustly and reliably explain these predictions simply because another backpropagation iteration, or a few more, could push these nodes to change unpredictably the predictions. To identify regions in the loss function, that can lead to small errors in the predictions, we propose to compute the entire Hessian. Since the very infancy of ANNs, the Hessian has been a central object of study. This is because the Hessian captures pairwise interactions of parameters via second-order derivatives of the loss function. The Hessian is given by:

$$\left( \frac{\partial^2 L}{\partial w_j \partial w_i} \right)_{MxN}, \tag{1}$$

where the $(i, j)$-th value in it represents a second-order derivative of $L$. For simplicity, we make two assumptions. Firstly, we assume that each hidden layer in the network has the same number of hidden nodes. Secondly, we assume that each input node from $\boldsymbol{x}$ is connected to each hidden node from the first hidden-node layer, each hidden node from layer $(L - 1)$ is connected to each hidden node from layer $L$, and each hidden node from the last hidden-node layer is connected to each label node from $\boldsymbol{y}$. We stress that these two assumptions are not strong in practice where ANN designers can always add additional connections and nodes to the network. We are now ready to define a score (XS) that we use for explaining the predictions of ANNs. This *explainability score* is given by the following equation:

$$XS(x, f, y, \kappa) = \frac{\sum_{j=1:M, i=1:N, \left|\frac{\partial^2 L}{\partial w_j \partial w_i}\right| \leq \kappa} \left| \frac{\partial^2 L}{\partial w_j \partial w_i} \right|}{\sum_{j=1:M, i=1:N} \left| \frac{\partial^2 L}{\partial w_j \partial w_i} \right|}, \tag{2}$$

where $\kappa$ limits the second-order derivatives of the loss function for the given prediction. Intuitively, $\kappa$ is a threshold that helps us select a region in the loss function (i.e. a set of hidden nodes), that is more robust to further backpropagation. Such a region is meaningful for explaining the predictions as it characterizes a reliable part of the network, which is not vulnerable to fluctuations in the predictions after further back- propagation. We note that XS($\boldsymbol{x}, f, \boldsymbol{y}, \kappa$) goes to 1 whenever most of the hidden nodes cause absolute errors not greater than $\kappa$, and 0 whenever most of the hidden nodes cause absolute errors greater than $\kappa$. Thus, given an ANN $\boldsymbol{f}$ with input nodes from $\boldsymbol{x}$ and label nodes from $\boldsymbol{y}$, as well as a fixed value for $\kappa$, our goal is to compute a population of hidden neurons that implements the score XS($\boldsymbol{x}, f, \boldsymbol{y}, \kappa$). We refer to it as a *possible explanation* for the prediction.

## V. Computing Explanations

Given the Hessian matrix, computing the score and its corre- sponding explanation requires performing $O(2 \cdot M \cdot N)$ summa- tions. We note that XS($\boldsymbol{x}, f, \boldsymbol{y}, \kappa_1$) $\geq$ XS($\boldsymbol{x}, f, \boldsymbol{y}, \kappa_2$) holds for $\kappa_1 \geq \kappa_2$. We let each second-order derivative be normalized to 1 by dividing it by the maximum such derivative. We might want to compute a possible explanation of the maximum size. To do so, we could iterate over $k$ from 0 to 1 in some sufficiently small step, say 0.001, and, for each $k$, compute XS($\boldsymbol{x}, f, \boldsymbol{y}, k$), which if it is the same as XS($\boldsymbol{x}, f, \boldsymbol{y}, (k - 1)$) then continue to the next iteration with $(k + 1)$ and, otherwise, exit from the cycle and return XS($\boldsymbol{x}, f, \boldsymbol{y}, (k - 1)$) and the corresponding hidden neurons. Thus, the task of computing explanations spoils down essentially to the task of computing the entire Hessian. However, exploiting second-order information of the loss is a notoriously difficult problem due to the Hessian being quadratic in the number of parameters, leading to great computational and memory costs and rendering the calculation intractable not only for big state-of-the-art models [6] but also for smaller problems and models. An intriguing possibility for the faster calculation of the Hessian through second-order backpropagation for MLPs was presented in [2]. However, to our knowledge, there is currently no available implementation of this algorithm, and certainly not one that could be used with modern machine-learning libraries. We next enable this by deriving and implementing equations required to make it compatible with modern ANNs.

## VI. Theoretical Results

### A. Two-stage backpropagation

We calculate Hessians with respect to (wrt) parameters by using PyTorch. We suppose that the auto-differentiation package provides a function grad(), which, given $L$, $M$-layer ANN model $f$ parameterized by $\boldsymbol{w} \in \mathrm{R}^N$, as well as $\boldsymbol{x}$ and $\boldsymbol{y}$, calculates each gradient $g_i$ of $L$ wrt $w_i$. Assuming the computational graph for calculating the gradients is preserved and the grad() function allows for backpropagation, one can backpropagate from each partial derivative wrt $w_j$ of the gradient $g_i$ to obtain the respective column of the Hessian.

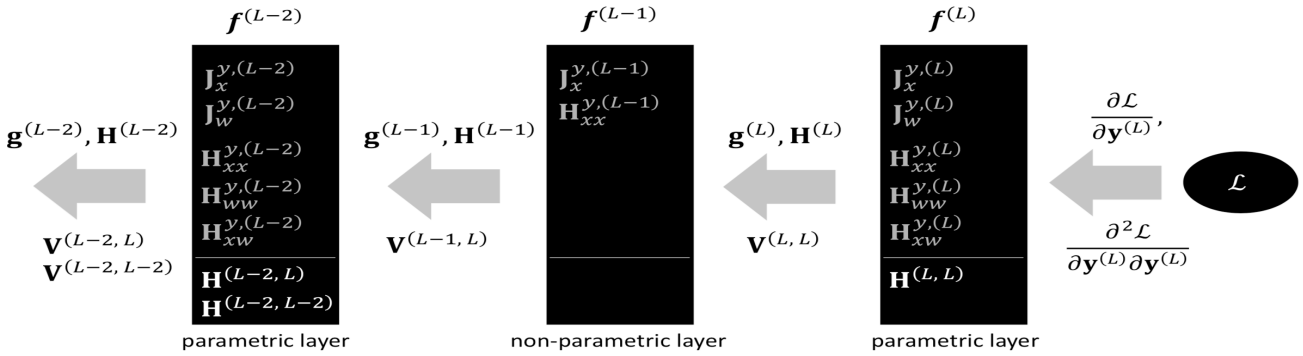


**Figure 1. An illustration of second-order backpropagation and local derivatives derived at every layer, for a network of L layers. For demonstration purposes, layers L − 2 and L are chosen to be parametric, while L − 1 is chosen to be non-parametric.**

*B. Second-order backpropagation*

1) *Layer-local derivatives:* Let us consider a feed-forward neural network with L layers, expressed as

$$\begin{aligned} &\forall s = 0 : (M-1), \\ &f^{(s)} : R^{N_s} \rightarrow R^{N_{(s+1)}}, \\ &y^{(s+1)} = f^{(s)}(y^{(s)}), \end{aligned} \tag{3}$$

where $\boldsymbol{y}^{(0)}$ or layer 0 is the input to the network and $\boldsymbol{f}^{(s)}$ is a transformation function (e.g. a (non-)linear activation) on which we impose the following conditions:

(1.1) The Jacobian and Hessian wrt input are given by:

$$\begin{aligned} J_x^{y,(s)} &= \left\{ \frac{\partial f_i^{(s)}(x)}{\partial x_j} \right\}_{ij}, \\ H_{xx}^{y,(s)} &= \left\{ \frac{\partial^2 f_i^{(s)}(x)}{\partial x_j \partial x_k} \right\}_{ijk}. \end{aligned} \tag{4}$$

These should be defined for almost any x. We refer to these as *Output-Input-Jacobian* (OIJ) and *Output-Input- Hessian* (OIH), respectively.

(1.2) Additionally, if $\boldsymbol{f}^{(s)}$ is a parametrized, namely $\boldsymbol{f}^{(s)}(x) \equiv \boldsymbol{f}^{(s)}(\boldsymbol{w}^{(s)}; \boldsymbol{x})$, these wrt parameters are given by:

$$\begin{aligned} J_w^{y,(s)} &= \left\{ \frac{\partial f_i^{(s)}(x)}{\partial w_j^{(s)}} \right\}_{ij}, \\ H_{ww}^{y,(s)} &= \left\{ \frac{\partial^2 f_i^{(s)}(x)}{\partial w_j^{(s)} \partial w_k^{(s)}} \right\}_{ijk}, \\ H_{xw}^{y,(s)} &= \left\{ \frac{\partial^2 f_i^{(s)}(x)}{\partial x_j \partial w_k^{(s)}} \right\}_{ijk}. \end{aligned} \tag{5}$$

These should be defined for almost any $\boldsymbol{w}^{(s)}$ and $\boldsymbol{x}$. We refer to these as *Output-Parameter-Jacobian* (OPJ), *Output-Parameter-Hessian* (OPH), and *mixed Output-Parameter-Hessian* (mOPH), respectively.

2) *Diagonal Hessian blocks:* Let us consider the Hessian concerning two parameters $w^{(s)}_i$, and $w^{(t)}_j$.

When s = t, we can use the chain rule of derivatives and expand the dependencies of outputs $\boldsymbol{y}^{(s)}$. Thus, we can rewrite the Loss-Parameter-Hessian (LPH) as a tensor dot product of (a) the Loss-Input-Gradient (LIG) $\boldsymbol{g}$ wrt layer s, (b) the Loss-Input-Hessian (LIH), (c) the OPH, and (d) the OPJ. We can express this as

$$H_w^{(s,s)} = g^{(s)T} . H_{ww}^{y,(s)} + J_w^{y,(s)T} . H^{(s)} . J_w^{y,(s)}. \tag{6}$$

Similarly, we can derive the Loss-Output-Hessian (LOH) at layer s by substituting $x \leftarrow y^{(s-1)}$ and by using (a) the LIG, and (b) the Hessian wrt the output of layer s, as well as (c) the OIJ and (d) the OIH:

$$H^{(s-1)} = g^{(s)T} . H_{xx}^{y,(s)} + J_x^{y,(s)T} . H^{(s)} . J_x^{y,(s)}. \tag{7}$$

3) *Off-diagonal Hessian Blocks:* The off-diagonal blocks, i.e. where s not equals t, are more difficult to compute.

However, we note that, given a network with *M* parametric layers, only ($M^2$ - *M*)/2 off-diagonal blocks need to be calculated due to the LPH being symmetric. For this reason, without loss of generality (wlog), it suffices to consider the case when s < t holds. As for the diagonal Hessian blocks, we can use the chain rule to rewrite the partial derivatives wrt the layer s and t. Denoting the OIJ with $U^{(s+1,t-1)}$, we derive the expressions:

$$\begin{aligned} V^{(t)} &= g^{(t)T} . H_{xw}^{y,(t)} + J_x^{y,(t)T} . H^{(t)} . J_w^{y,(t)}, \\ H_w^{(s,t)} &= J_w^{y,(s+1)T} . U^{(s+1,t-1)} . V^{(t)}. \end{aligned} \tag{8}$$

The *chained Output-Input-Jacobian* (cOIJ) $U^{(s+1,t-1)}$ can be obtained using the OIJs of the intermediate layers through iterated application of the chain rule as follows:

$$U^{(s+1,t-1)} = J_x^{y,(s+1)} . J_x^{y,(s+2)} ... J_x^{y,(t-1)}. \tag{9}$$

To avoid having to perform the full matrix chain multiplication for each $U^{(s+1,t)}$, we can use a dynamic programming scheme that follows the next rules:

$$\begin{aligned} V^{(t,t)} &= V^{(t)}, \\ V^{(s,t)} &= J_x^{y,(s)} . V^{(s+1,t)}. \end{aligned} \tag{10}$$

Putting everything together, we can now formulate a second-order backpropagation pass: see Figure 1. The overall time complexity of this pass is in the order of $O(M^3)$ tensor con- tractions, where each layer-local derivative has to be computed exactly once.

## VII. Empirical Results

We implemented the above derivations. We refer to our implementations as *xann*. We then conducted comparative experiments on single GPU nodes of a cluster with an AMD Ryzen Threadripper 1950X processor, 4x Nvidia GeForce RTX 2080 TI GPUs with 11GB VRAM, and 64GB RAM. The operating system was Debian 5.10.46-4 (2021-08-03) x86 64. We investigated how the computation time scales up when varying the input size and batch size of samples from the MNIST dataset. For this purpose, we prepared an N - N - N MLP, where N is the network size. Thus, for each tuple (input size, batch size), we ran an experiment for all four possible configurations: {'autograd', 'xann'} × {'cpu', 'gpu'}, where the autograd denotes the use of two-stage backpropagation. Figures 2 and 3 show the results.

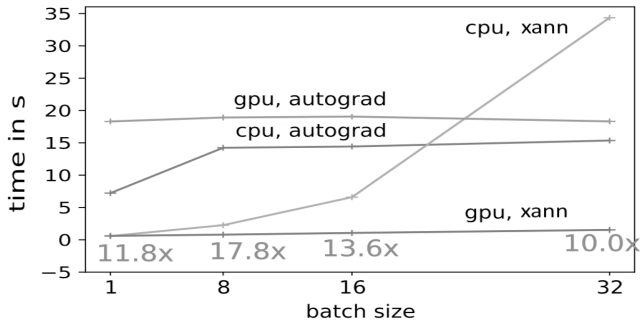


**Figure 2. The Hessian computation time for input size 100.**

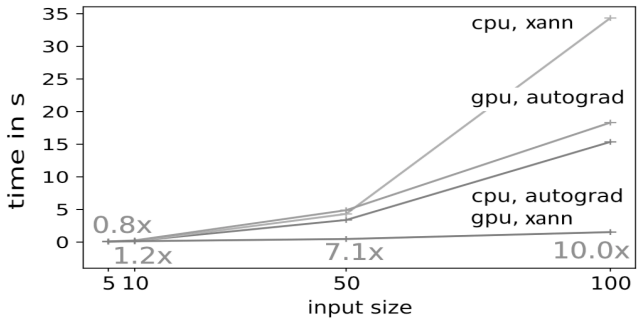


**Figure 3. The Hessian computation time for batch size 32.**

For a fixed input size of 100, *xann* on GPU was consistently faster than the autograd methods and the absolute time remained small within the inspected range (see Figure 2). For a fixed batch size of 32, the same behavior was observed for different input sizes (see Figure 3). For example, let us consider the maximum batch and input sizes of 32 and 100, respectively. At that configuration, *xann* took around 1.3s, while the autograd methods took 15.3s and 18.5s on the Central Processing Unit (CPU) and GPU, respectively. The results indicated a significant improvement in the overall training time in settings where one needs to compute the Hessian frequently. However, the computation time of *xann* on CPU increased substantially for large input and batch sizes.

Finally, as discussed earlier, once we have computed the entire Hessian efficiently, we can also compute explanations for our network in $O(N^2)$ time.

## Conclusion and Future Work

We presented an approach for generating explanations for neural network dynamics from the entire Hessian that helped us to identify regions in the networks that justify their predictions. We derived a novel method for computing the Hessian, implemented it, and evaluated it in experiments. This method outperforms state-of-the-art methods on networks of sizes up to one hundred neurons. While this might be sufficient for simplified predicting trends and automated trading, it remains an interesting direction to produce fast explanations in larger networks.

## References


[1] A. Barredo Arrieta, N. D´ıaz-Rodr´ıguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, "Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai," *Information Fusion*, vol. 58, pp. 82–115, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1566253519308103

[2] E. Mizutani, S. Dreyfus, and J. Demmel, "Second-order backpropagation algorithms for a stagewise-partitioned separable hessian matrix," in *Proceedings of the International Joint Conference on Neural Networks*, vol. 2, 01 2005, pp. 1027–1032. [Online]. Available: https://ieeexplore.ieee.org/document/1555994

[3] F. Dangel, S. Harmeling, and P. Hennig, "Modular block-diagonal curvature approximations for feedforward architectures," in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, S. Chiappa and R. Calandra, Eds., vol. 108. PMLR, 26–28 Aug 2020, pp. 799–808. [Online]. Available: https://proceedings.mlr.press/v108/dangel20a.html

[4] F. Dangel, F. Kunstner, and P. Hennig, "BackPACK: Packing more into backprop," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=BJlrF24twB

[5] J. Martens and R. Grosse, "Optimizing neural networks with kronecker- factored approximate curvature," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, pp. 2408–2417. [Online]. Available: https://dl.acm.org/doi/10.5555/3045118.3045374

[6] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf