Neural Network Calculator for Designing Trojan Detectors

Peter Bajcsy*
Information Technology Laboratory
National Institute of Standards and Technology
100 Bureau Drive. Gaithersburg, MD 20899
peter.bajcsy@nist.gov

Nicholas J. Schaub
National Center for Advancing Translational Sciences (NCATS)
National Institutes of Health (NIH)
Axle Informatics
6116 Executive Blvd Suite 400, Rockville, MD 20852
nick.schaub@nih.gov

Michael Majurski
Information Technology Laboratory
National Institute of Standards and Technology
100 Bureau Drive. Gaithersburg, MD 20899
michael.majurski@nist.gov

Abstract

This work presents a web-based interactive neural network (NN) calculator and a NN inefficiency measurement that has been investigated for the purpose of detecting trojans embedded in NN models. This NN Calculator is designed on top of TensorFlow Playground with in-memory storage of data and NN coefficients. Its been extended with additional analytical, visualization, and output operations performed on training datasets and NN architectures. The analytical capabilities include a novel measurement of NN inefficiency using modified Kullback-Liebler (KL) divergence applied to histograms of NN model states, as well as a quantification of the sensitivity to variables related to data and NNs. Both NN Calculator and KL divergence are used to devise a trojan detector approach for a variety of trojan embeddings. Experimental results document desirable properties of the KL divergence measurement with respect to NN architectures and dataset perturbations, as well as inferences about embedded trojans.

1 Introduction

With the widespread use of neural networks in life-critical applications, such as self-driving cars, commercial and government agencies are concerned about the security of deployed deep learning (DL) neural networks (NNs). One example is poisoning NN models during training with datasets containing triggers (trojans) for misclassification. When the poisoned model is used for inferencing, a user will not know about the introduced misclassification by adversaries unless the input for inferencing is presented with the trojan. The problem of detecting trojans in NN models has been posed as a challenge by the Intelligence Advanced Research Projects Agency (IARPA) and named as

*URL: https://www.nist.gov/people/peter-bajcsy

the Trojan in Artificial Intelligence (TrojAI) challenge [1]. In order to address this challenge, there is a need to gain basic insights about trojans, their interactions with NNs, NN measurements that can indicate the presence of trojans, and what algorithmic approaches can be successful in detecting trojans in a variety of NN architectures potentially while under computational constraints. This work aims at providing an interactive environment for (a) gaining such insights and (b) assessing the difficulty of different trojan detection challenges.

We address three specific problems in the aforementioned context. The first problem is in creating an interactive environment for quick evaluations of (1) NN models with varying complexities and hyper-parameters, (2) datasets with varying manifold representation complexities and class balance ratios, and (3) measurements based on varying approaches and statistical analyses. The second problem lies in designing NN efficiency measurements with understood sensitivity to variations in NN architectures, NN initialization and training, as well as dataset regeneration. The third problem is in devising an approach to detecting trojans embedded in NN models.

The problems come with associated challenges. The first challenge lies in the interactivity requirement. As of today, DL NN architectures are very complex; from 60K parameters in LeNet [2], to common networks having millions and billions of parameters (160 billion reported in [3]). Modern networks require hours or days to train on advanced graphics processing unit (GPU) cards [4]. The challenge of the second problem lies in the lack of explainable artificial intelligence (AI) [5] and AI mathematical models [6], [7], and [8]. The last challenge lies in the large search space of possible trojans, training data, DL NN architectures, and NN training algorithms that must be understood. Additional related work is provided in Appendix B.

Our approach to these challenges relies on designing a NN Calculator environment and is based on analyses of neuron states in fully connected layers of NNs. The NN Calculator is built on top of Tensorflow Playground [9] by enabling all calculator operators on datasets and NNs, such as storing, retrieving, setting, adding, subtracting, and clearing memory containing training/testing data points and NN coefficients. Furthermore, the NN Calculator contains functionality for introducing a wide range of trojans, collecting NN state measurements, visualizing them, computing trojan sensitive probes, evaluating their robustness to NN training, and saving them for further analyses. The trade-off for interactivity of analyses is the input limitation to 2D dot patterns, the NN limitation to less than 7 hidden layers and 9 nodes per layer due to screen size, and the limitation to custom designed features derived from 2D dot patterns.

The novelty of the work lies in designing:

- a web-based NN calculator for the AI community interested in gaining research insights about NN performance under various configurations,
- a Kullback-Liebler (KL) divergence based measurement of NN inefficiency,
- an approach to detecting embedded trojans in AI models.

2 Methods

2.1 NN Calculator

Our approach to designing NN Calculator aims at making it as similar as possible to a standard calculator. Unlike a standard (or scientific) calculator, NN Calculator operates on datasets and NN coefficients as opposed to simple numbers. Thus, we reused the symbols for MC, MR, M+, M-, and MS for clearing, retrieving, adding, subtracting, and setting memory with datasets (training and testing sets) and NN coefficients (biases and weights). The user interface is shown in Figure 1 (top left and middle left) where the standard five symbols are preceded with NN or D to indicate whether the operation is applied to NN or data. In addition, we included NN model averaging and dataset regeneration in order to study variability over multiple training sessions and random data perturbations. Evaluating combinations of datasets and NNs in real time enables one to explore full factorial experiments for provided factors.

Most of the calculator settings are used for the main operations on datasets and NNs: training, inferencing, inefficiency computations, and robustness measurements (mean squared error (MSE)) for training, testing and inferencing of sub-sets. Additional operations include collecting neuron state histograms, and derived measurement statistics. The remaining settings are used to view

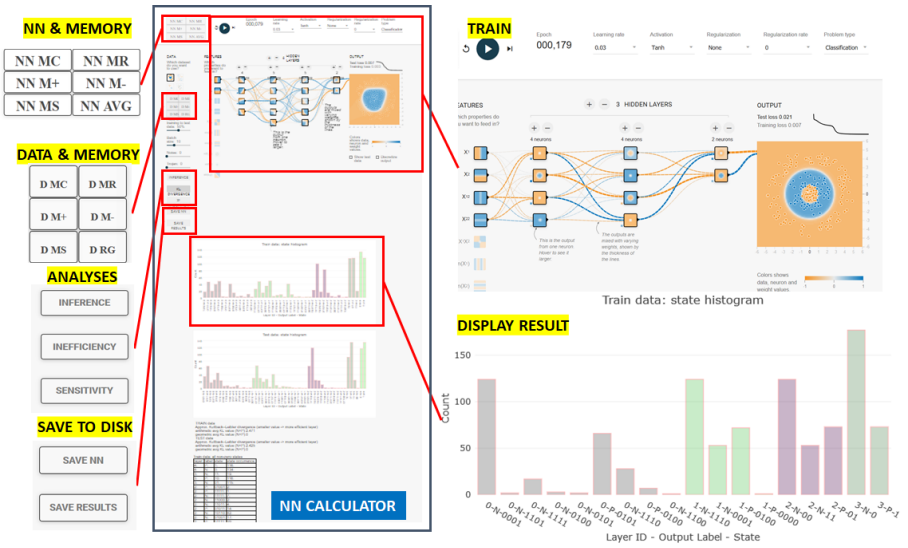

Figure 1: NN Calculator user interface.

characteristics of datasets (noise, trojan), parameters of NN modeling algorithm (Learning Rate, Activation Function, Regularization, Regularization Rate), and parameters of NN training algorithms (Train to Test Ratio, Batch Size). In order to keep track of all settings, we added the option of saving all NN parameters and NN coefficients, as well as saving all inefficiency and robustness analytical results. The save options are shown in Figure 1 (bottom left).

2.2 Trojan Characteristics Modeled in NN Calculator

In order to explore how to discriminate a model trained with trojan and a model trained without trojan, we added nine types of trojans to the NN Calculator. A more detailed explanation of what trojans are in NNs is provided in Appendix A. Our objective is to understand how the characteristics of trojans affect the trojan detection, i.e. the discrimination of models trained without trojan (`TwoT`) and trained with trojan (`TwT`). We generalized trojan embedding characteristics to be described by (1) number of trojans per class, (2) number of trojans per contiguous region, (3) shape, (4) size , and (5) location of trojans inside of a class region. Figure 2 illustrate the nine trojan embeddings. Table 1 in Appendix C includes details about each trojan embedding.

2.3 Neural Network Inefficiency Measurement

In this section, we introduce a NN inefficiency measurement from a histogram of NN states at each layer by using (1) KL divergence, (2) a reference state distribution, and (3) computational constraints.

States of Neural Network: In order to derive NN inefficiency, we must measure and analyze states of NN layers as training data are encoded in a typical classification problem into class labels. A state of one NN layer is defined as a set of outputs from all nodes in a layer as a training data point passes through the layer. The output of a node is encoded as 1 if the value is positive and 0 otherwise. Thus, for a point d_k from a 2D dataset with points $[d_k = (x_k, y_k), c_j]$, $k = 1, ..., npts$ and $C = 2$ classes $c_1 = orange/N(egative), c_2 = blue/P(ositive)$, it can generate one of 2^{nl} possible states at a NN layer with nl nodes. Figure 3 (top) shows how to gather state information during training into a table and compute a histogram of states per layer and per class label. Each step of the process is outlined below.

Representation Power Defined Via Neural Network States: We view the histogram of states as a probability distribution that indicates the utilization of a layer. In order to quantify the NN utilization, we leveraged the parallels between neural network and communication fields in terms of (a) NN representation power/capacity (channel capacity in communications), (b) NN efficiency (channel efficiency), and (c) the universal approximation theorem [10] (source coding theorem [11]). Accord-

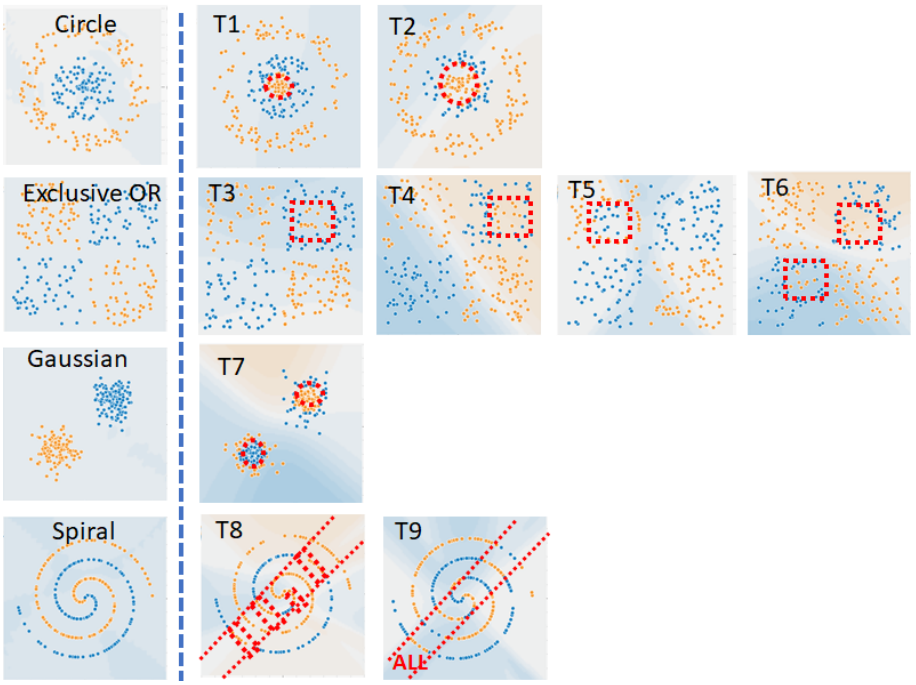

Figure 2: Illustration of nine trojan embeddings in four datasets. Orange dot - class 1, blue dot - class 2, red boundary encloses dots that represent a trojan embedding.

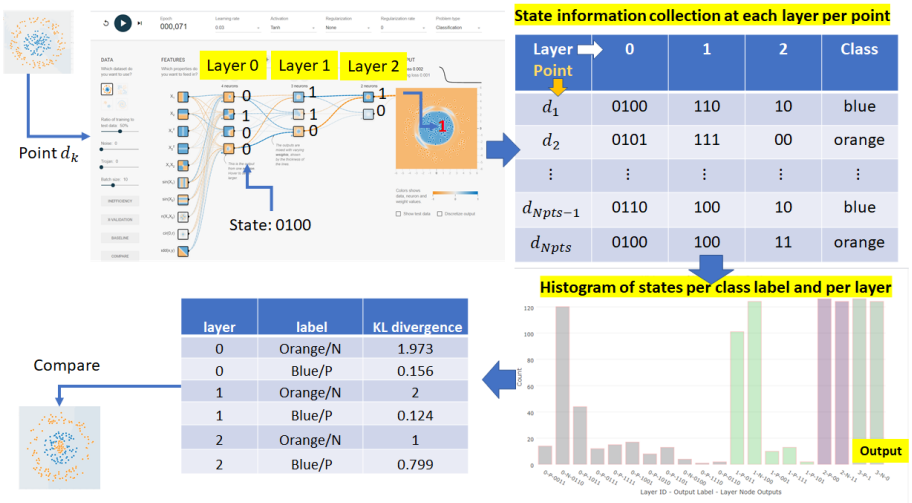

Figure 3: The computation of KL divergence from NN state information at each layer per class label. Top left: states 0100, 110 and 10 at the three layers for an input point. Top right: tabular summary of state information. Bottom right: Combined histogram of states for all layers and both class labels (one color per layer). Bottom left: KL divergence computed per layer and label.

ing to the universal approximation theorem, we view the NN representation power (also denoted as expressiveness or model capacity or model complexity) as its ability to assign a training class label to each training point and create accurate class regions for that class. For instance, a NN must have at least two nodes ($nl = 2$) in the final layer in order to assign four class labels (i.e., $C = 4 \leq 2^{nl} = 4 \rightarrow \{00, 01, 10, 11\}$).

Once we gather the state information (see Figure 3 (top)), we can categorize the states into three categories:

1. State is used for predicting multiple class labels.
2. State is used for predicting one class label.
3. State is not used.

The first category is detected when a NN does not have enough nodes (insufficient representation power). It could also occur when a NN layer does not contribute to discriminating class labels (poorly trained NN). The second category suggests that a subset of data points associated with the same class label is represented by one state (efficient or inefficient representation). The last category implies that a NN has a redundant (inefficient) node in a layer for representing a class label. Thus, states at NN layers provide information about NN representation power as (1) insufficient, (2) sufficient and efficient, or (3) sufficient and inefficient. An ideal NN is sufficient and efficient.

Inefficiency of Neural Network Defined Via Neural Network States: Since the source coding theorem is based on calculating mutual information defined via KL divergence [12], we adopt KL divergence as a measurement of how inefficient it would be on average to code one histogram of NN layer states using a reference histogram as the true distribution for coding, where the reference histogram is defined below as the outcome of a uniform distribution over states assigned to each label. Figure 3 (bottom) shows example results of KL divergence values derived per layer and per label that can be used to compare against values obtained from other datasets; for instance, datasets with trojans.

The rationale behind choosing entropy-based KL divergence with probability ratios comes from three considerations. First, entropy-based measurement is appropriate because which state is assigned to predicting each class label is a random variable and a set of states assigned to predicting each class label is random. Second, probability-based measurement is needed because training data represent samples from the underlying phenomena. Furthermore, while training data might be imbalanced (a number of samples per class varies), all training class labels are equally important and the probabilities of classes should be included in the measurement. Third, the divergence measurement reflects the fact that we measure NN efficiency relative to a maximum efficiency of NN that is achieved when sets of states utilize the entire network capacity (representation power).

Mathematical definition: Formally, let us denote $Q_j = \{q_{ij}\}_{i=1}^{n}$ to be a discrete probability distribution function (PDF) of n measured NN states and $P_j = \{p_{ij}\}_{i=1}^{n}$ to be the PDF of reference (ideal) NN states. The probabilities are associated with each state (index i) and each class label (index j). The KL divergence per class label j is defined at each NN layer in Equation 1.

$$D_{KL}(Q_j \parallel P_j) = \sum_{i=1}^{n} (q_{ij} * \log_2 \frac{q_{ij}}{p_{ij}}) \tag{1}$$

where $q_{ij} = \frac{count(i,j)}{p_j * npts}$ is the measured count of states normalized by the probability p_j of a class label j and the number of training points $npts$. The PDF of reference states per class label uniformly utilizes the number of states assigned to predicting each class label (i.e., 2 classes imply $\frac{1}{2}$ of all states per label). The reference probability distribution is uniform across all assigned states. Thus, all reference probabilities can be computed as $p_{ij} = m * \frac{1}{n}$ where m is the number of classes and $n = 2^{nl}$ is the maximum number of states (nl is the number of nodes per layer).

Equation 1 for the Kullback–Leibler divergence is defined only if for all x, $p_{ij} = 0$ implies $q_{ij} = 0$. Whenever $q_{ij} = 0$ the contribution of the corresponding term is interpreted as zero because $\lim_{x \rightarrow 0}(x * \log_2 x) = 0$ (see Appendix D). The case of "not defined" takes place when there are more non-zero states than the number of non-zero reference states (i. e., the cardinality of two sets satisfies

the equation: $|Set(q_{ij} \neq 0)| > |Set(p_{ij} \neq 0)|$). This case indicates that a NN has insufficient representation power to encode input dataset into a class label.

Expected properties: It is expected that KL divergence will satisfy a list of basic properties as datasets, features, and NN capacity vary. For example, given an input dataset and a set of features, inefficiency (KL divergence) per layer should increase for an increasing number of nodes per NN layer. In another example, given a NN capacity, inefficiency should decrease for datasets with added noise or trojans. The relative changes are expected to be larger than the KL divergence fluctuations due to data reshuffling, data regeneration from the same PDF or due to re-training the same NN (referred to as sensitivity of KL divergence).

Computational Consideration about Inefficiency: The KL divergence computation considers computational and memory complexities since it must scale with increasing numbers of class labels, nodes, and layers.

Memory concerns: One should create a histogram with the number of bins equal up to 2^{nl} per class label and per layer which can easily exceed the memory size. For example, if a number of classes is ≈ 10, a number of nodes is ≈ 100, and a number of layers is ≈ 100, then memory size is $\approx 2^{100} * 10 * 100 \approx 10^{33}$ bytes. In our implementation approach, we create bins only for states that are created by the training data which leads to the worst case memory requirement scenario to be $npts * 10 * 100$ bytes.

Computational concerns: One should align measured histograms per class label to identify the states uniquely encoding each class in order to avoid the "not defined" case of KL divergence or the case of the same state encoding multiple class labels. To eliminate the alignment computation in our implementation approach, we modify the KL divergence computation to approximate the KL divergence according to Equation 2. The computation of modified KL divergence $\widehat{D_{KL}}$ requires only collecting non-zero occurring states and calculating their histogram. The derivation of Equation 2 can be found in Appendix D.

$$\widehat{D_{KL}}(Q_j \parallel P_j) = \sum_{i \in Set(q_{ij} \neq 0)} (q_{ij} * \log_2 q_{ij}) - \log_2 \frac{m}{n} \tag{2}$$

While KL divergence satisfies $D_{KL} \leq 0$, the modified KL divergence $\widehat{D_{KL}}$ can be negative for those cases when $|Set(q_{ij} \neq 0)| > |Set(p_{ij} \neq 0)|$. However, the negative value is lower bounded by Equation 3. For negative values, the NN layer is insufficient for encoding input data to class labels.

$$\max_{Q_j}(D_{KL}(Q_j \parallel P_j) - \widehat{D_{KL}}(Q_j||P_j)) = -\sum_{i \in Set(q_{ij} \neq 0)} (q_{ij} * \log_2 p_{ij}) - \log_2 \frac{m}{n} \tag{3}$$

The rationale behind modified KL divergence is that (1) the alignment is not important for sufficient efficient and inefficient models (it is primarily important for insufficient models), (2) the approximation assumes $p_{ij} \neq 0$ at all non-zero states $q_{ij} \neq 0$ which yields negative modified KL divergence values as indicators of insufficiency, and (3) the alignment is important for detecting poorly trained models which could be using the same states for predicting multiple class labels while leaving all other available states in a NN layer unused. For the last case, we assume that all models were properly trained and class labels are not assigned at random. Furthermore, the modified KL divergence addresses the problem of different within-class variations in training data which can lead to one class needing more allocated states than some other class. The modified KL divergence can be extended in the future by estimating within-class variations and assigning the number of states per class accordingly. In the following section we show how we use the modified KL convergence to detect the presence of trojans in a network.

2.4 Approach to Trojan Detection

Our assumptions are that (1) we have only testing datasets without trojans and (2) NN models with trojan and without trojan have the same accuracy. We can simulate many varying NN models, with 4 example datasets containing 2 classes, and nine types of trojans. The simulations assume close to $100\,\%$ model accuracy on training data (with or without trojan). The comparisons of modified KL

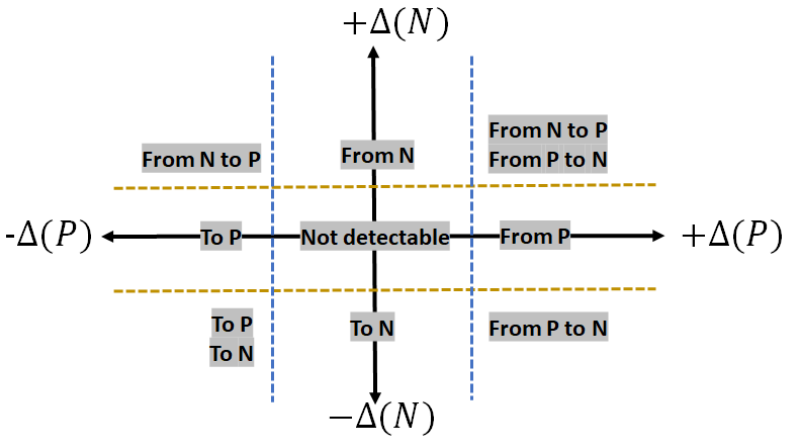

Figure 4: Trojan detection using the delta between modified KL divergence of models `TwoT` and `TwT` as defined in Equation 4. The values for dashed lines can be determined based on the sensitivity of deltas to data regeneration and reshuffling, as well as to multiple NN initializations and re-training.

divergence values are computed from `TwoT` and `TwT` models using datasets without trojans. The model `TwT` evaluated with datasets without trojans might have an accuracy less than $100\,\%$ in simulations but the accuracy difference would be negligible in a real scenario (and the challenge models).

The comparisons are performed at each NN layer and for each class label. The simulation execution is interactive (i.e., execution time is on the order of seconds) and follows the steps: (1) Select data, (2) Train, (3) Store model, (4) Select other data, (5) Restore model, (6) Perform NN measurement. Our assumption is that the magnitudes of KL divergence values for a NN model trained with a trojan embedded in a particular class (`TwT`) are smaller than the magnitudes for a NN model trained without trojan for the same class (`TwoT`). Our approach toward trojan detection is summarized in Figure 4. The axes correspond to the class-specific deltas between modified KL divergence of models `TwoT` and `TwT`. The dashed lines are set at a value σ that corresponds to the sensitivity of $\widehat{D_{KL}}$ to NN re-training as well as to data regeneration and re-shuffling. The notation "to" and "from" in Figure 4 refers to our inference about trojans causing data points "from" one class to be mis-classified "to" another class based on the deltas defined in Equation 4 where P and N are the two classes shown as blue and orange in the NN Calculator.

$$\begin{aligned} \Delta(P) &= \widehat{D_{KL}}(TwoT/P) - \widehat{D_{KL}}(TwT/P) \\ \Delta(N) &= \widehat{D_{KL}}(TwoT/N) - \widehat{D_{KL}}(TwT/N) \end{aligned} \tag{4}$$

3 Experimental Results

3.1 NN Calculator

NN Calculator is implemented in TypeScript. The code is available from a GitHub repository with the development instructions and deployment via GitHub pages `https://github.com/usnistgov/nn-calculator`. The current list of features extracted from 2D datasets includes $X1, X2, X1^2, X2^2, X1 * X2, \sin(X1), \sin(X2), \sin(X1 * X2), \sin(X1^2 + X2^2)$, and $X1 + X2$. The code uses D3.js and Plotly.js JavaScript libraries for visualization. All analytical results are displayed in NN Calculator below the NN visualization. The results consist of a state histogram (bins for both classes) and tabular summaries. The state histogram is interactive while the numerical results are presented as tables with a unique delimiter for easy parsing.

To gain additional insights about state (although they might be computationally expensive for large NNs), simulations using NN Calculator report also the number of non-zero histogram bins per class, the states and their counts per layer and per label for most and least frequently occurring states, the

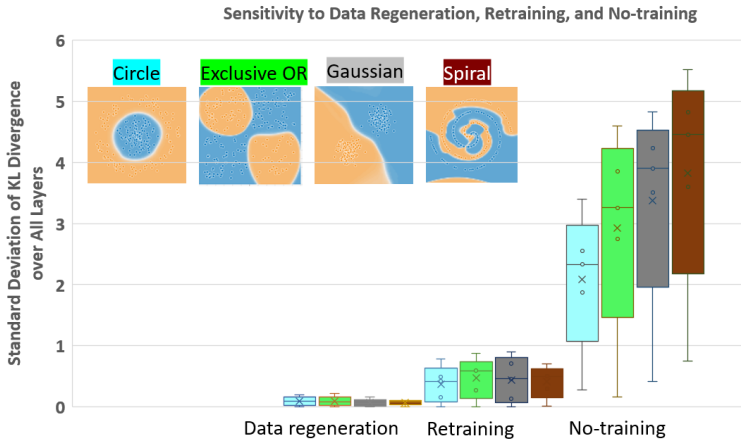

Figure 5: Sensitivity of inefficiency to stochastic regeneration of datasets from the same distribution, retraining and no-training with different random initialization. The box plot shows values computed from a set of standard deviations of modified KL divergence per layer and per class for the four datasets.

number of overlapping states across class labels and their corresponding states, and the bits in states that are constant for all used states for predicting a class label. The additional information is reported for the purpose of exploring optimal NN architectures and investigating NN model compression schemes.

3.2 Neural Network Inefficiency

KL Divergence Properties: We verified and quantified desirable properties of the modified KL divergence defined in Equation 2, such as decreasing inefficiency for increasing amount of added noise and increasing inefficiency for increasing number of nodes. The supporting results can be found in Appendix E.

Sensitivity of Inefficiency Measurement: We quantified the sensitivity of NN inefficiency measurement with respect to (a) data reshuffling and regeneration, (b) NN re-training with different initialization, and (c) no-training as the worst case of poor training. To look at the sensitivity of the NN inefficiency with respect to data regeneration, we performed the following: a NN model is trained for a dataset and stored in memory. Next, four datasets are regenerated and a standard deviation of inefficiency values are computed at each layer and for each class. Finally, the average value is computed over all standard deviations and the experiment is repeated for four 2D datasets with the results presented in Figure 5. From the data regeneration points in in Figure 5, we concluded that the average of standard deviations in inefficiency values larger than 0.1 will indicate dissimilarity of models by other factors.

We performed similar sensitivity experiments for no-training and retraining with random initialization. Figure 5 includes the results for four datasets. The sensitivity to retraining is bounded to approximately the average of inefficiency standard deviations equal to 0.46 while the same value for no-training is about 5 to 8 times larger and appears to be proportional to the complexity of the class distribution.

Comparison of Inefficiencies for Trojan Embeddings: Comparisons of models `TwoT` and `TwT` were conducted in NN Calculator using a NN with 6 hidden layers, 8 nodes per layer and 4 features including $X1, X2, X1^2, X2^2$ and $X1 * X2$. The algorithmic and training parameters are set to learning rate: 0.03, activation: $Tanh$, regularization: none, ratio of training to test data: $50\,\%$, and batch size: 10.

Figure 6 shows the delta between modified KL divergence values of models `TwoT` and models `TwT` for the two classes P (blue) and N (orange) and for the two trojans (T1 and T2) of different sizes (Figure 6 left). For both trojans, the delta KL divergence values are positive for the P (blue) class and negative for the N (orange) class: $\Delta(P) > 0.454$ and $\Delta(N) < -0.702$. These values imply that a trojan is embedded in class P (blue) in both trojan cases and is encoding class N (orange) according

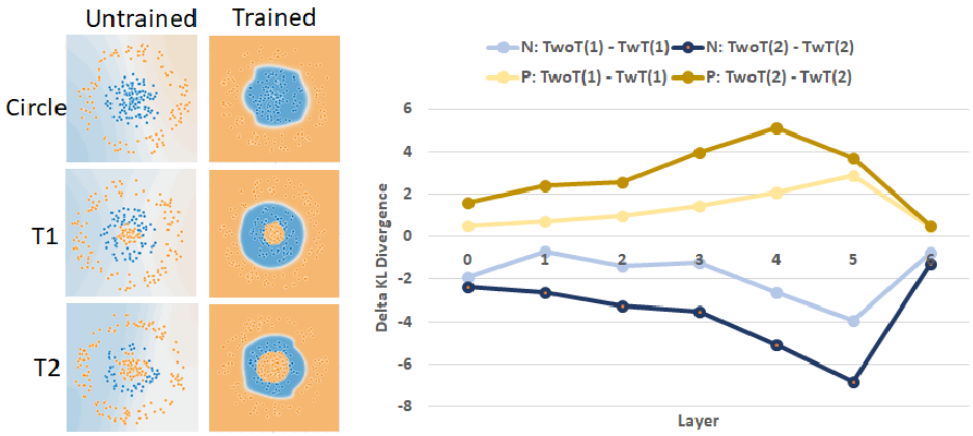

Figure 6: Comparison of inefficiencies between models `TwoT` and `TwT`, and embedded orange trojans T1 and T2 with different sizes (see Figure 2, top row). The plot shows the values of $\Delta(P)$ and $\Delta(N)$ for T1 and T2 at each NN layer.

to Figure 4 ("From P to N" $\rightarrow$ misclassified points labeled as P to N). Furthermore, as the size of a trojan increased from T1 to T2 by a size factor of 2.25, the ratio of deltas increased by 2.24 for class N and by 2.37 for class P (see Appendix D).

Figure 7 illustrates the delta between modified KL divergence values of models `TwoT` and models `TwT` for the trojans T8 and T9 whose embeddings differ in terms of the number of classes and the number of class regions. First, we observe for trojan T8 that $\Delta(T8/P) > 0.48$ and $\Delta(T8/N) < -0.769$. These values imply that the trojan T8 is embedded in class P (blue) according to Following Figure 4 ("From P to N").

We recorded much lower delta values for the trojan T9 than in the previous comparisons. This indicates the much higher complexity of modeling the spiral dataset than circle, exclusive OR, or Gaussian datasets and therefore lower inefficiency values measured at NN layers. Based on the sensitivity values shown in Figure 5 (0.1 for data regeneration and 0.5 for re-training), we could infer that the trojan T9 is likely in both classes based on the placement of the point $[\Delta(T9/P) > -0.034,\ \Delta(T9/N) > 0.035]$ in Figure 4 (i.e., the sub-spaces "From N", "From P", "Not detectable", and "From N to P" + "From P to N").

Due to the discrete nature of the spiral pattern, the P class (blue) occupies a longer curve than the N class (orange). This contour length ratio ($P : N \approx 12.31 : 7.33$) can explain why ($\Delta(T9/P) > \Delta(T9/N)$ for almost all layers. However, we are not able to make any inferences about the number of regions from Figure 7 (right) other than that the complexity of modeling class P or N in the case of T8 is more inefficient than modeling class P and N in the case of T9 by comparing the deltas of modified KL divergence values.

Summary and Future Work

We designed NN calculator and an inefficiency measurement that has been thoroughly investigated for the purpose of detecting trojans embedded in NN models. In addition to implementing an interactive web-based NN calculator for gaining insights, we have built the mathematical foundation for designing trojan detectors with a variety of characteristics. Additional discussion of the experiments can be found in Appendix G. In our future work, we will explore the Python implementation and its accuracy and computational performance on the TrojAI challenge data.

Broader Impact

The broader impact of this work is two-fold. First, it is the creation of a web-based interactive NN Calculator for the community of AI researchers in order to gain insights about trojan detection,

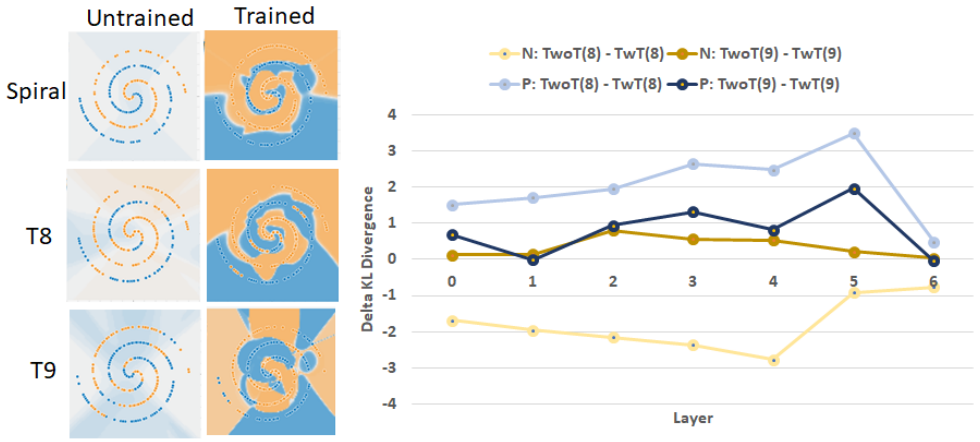

Figure 7: Comparison of inefficiencies between models `TwoT` and `TwT`, and embedded trojans T8 and T9 with different number of classes (1 or 2) and class regions (1 or 4).

pursue explainable AI, and search for neural architectures satisfying model compression or optimal performance objectives. Furthermore, the tool can have an educational value in class rooms as intended by the authors of TensorFlow Playground [9]. Second, the rudimentary work on measuring neural network inefficiency using KL divergence has impact on advancing mathematical and statistical modeling of neural networks. Such modeling efforts suffer currently from a steep learning curve, hardware requirements, and time delays between experimental runs. Overall, the work should democratize our basic understanding of neural networks since NN Calculator can be accessible using a browser and experiments using NN Calculator do not require specialized hardware (i.e., GPU cards) nor long waiting times.

In a summary, (a) researchers and educators may benefit from this research; (b) no one is put at disadvantage from this research; (c) the consequences of failure of NN Calculator would be contained in the browser; and (d) the computational methods do not leverage any biases in the data.

Acknowledgement

The funding for Bajcsy and Majurski was provided by IARPA. The funding for Schaub was provided by NCATS NIH.

Disclaimer

Commercial products are identified in this document in order to specify the experimental procedure adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the products identified are necessarily the best available for the purpose.

References

[1] IARPA. Trojans in Artificial Intelligence (TrojAI), 2020.

[2] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. A Survey of the Recent Architectures of Deep Convolutional Neural Networks. *Artificial Intelligence Review*, pages 1–68, 2020.

[3] Andrew Trask, David Gilmore, and Matthew Russell. Modeling order in neural word embeddings at scale. *32nd International Conference on Machine Learning, ICML 2015*, 3:2256–2265, 2015.

[4] Daniel Justus, John Brennan, Stephen Bonner, and Andrew Stephen McGough. Predicting the Computational Cost of Deep Learning Models. *Proceedings - 2018 IEEE International Conference on Big Data, Big Data 2018*, pages 3873–3882, 2019.

[5] Derek Doran, Sarah Schulz, and Tarek R. Besold. What does explainable AI really mean? A new conceptualization of perspectives. *CEUR Workshop Proceedings*, 2071, 2018.

[6] Joan Bruna and L G Dec. Mathematics of Deep Learning, 2017.

[7] Michael Unser. A representer theorem for deep neural networks, 2019.

[8] Stéphane Mallat. Understanding Deep Convolutional Networks. *Philosophical Transactions A*, 374(20150203):1–17, 2016.

[9] Daniel Smilkov, Shan Carter, D. Sculley, Fernanda B. Viégas, and Martin Wattenberg. Direct-Manipulation Visualization of Deep Networks, 2017.

[10] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[11] C. E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(623-656):379–423, 1948.

[12] S. Kullback and R. A. Leibler. On Information and Sufficiency. *Annals of Mathematical Statistics.*, 22(1):79–88, 2017.

[13] Hava Siegelmann. Guaranteeing AI Robustness against Deception (GARD), 2019.

[14] Xiaojun Xu, Qi Wang, Huichen Li, Nikita Borisov, Carl A. Gunter, and Bo Li. Detecting AI Trojans Using Meta Neural Analysis, 2019.

[15] Kevin Roth, Yannic Kilcher, and Thomas Hofmann. The Odds are Odd : A Statistical Test for Detecting Adversarial Examples, 2019.

[16] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning Attack on Neural Networks. In *Network and Distributed Systems Security (NDSS) Symposium 2018*, number February, pages 1–15, San Diego, CA, 2018. Internet Society.

[17] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11050 LNCS:273–294, 2018.

[18] Te Juin Lester Tan and Reza Shokri. Bypassing Backdoor Detection Algorithms in Deep Learning, 2019.

[19] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal Brain Damage. In *Proceedings of Neural Information Processing Systems*, pages 4–11, Holmdell, New Jersey, 1989. AT&T Bell Laboratory, Neural Information Processing Systems Foundation, Inc.

[20] David G Stork Babak Hassibi. Second Order Derivatives for Network Pruning: Optimal Brain Surgeon. In *Advances in Neural Information Processing Systems 5 (NIPS 1992)*, pages 164–172. ral Information Processing Systems Foundation, Inc., 1992.

[21] Hengyuan Hu, Rui Peng, Yu-wing Tai, Sensetime Group Limited, and Chi-keung Tang. Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures, 2016.

[22] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning Filters for Efficient ConvNets. In *International Conference on Learning Representations*, pages 1–13, Palais des Congrès Neptune, Toulon, France, 2017.

[23] Song Han, Jeff Pool, John Tran, and William J Dally. Learning bothWeights and Connections for Efficient Neural Networks, 2015.

[24] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine learning practice and the bias-variance trade-off. *Proceedings of National Academy of Sciences (PNAS)*, 116(32):15849–15854, 2019.

[25] Matthew D Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks, 2013.

[26] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing Higher-Layer Features of a Deep Network, 2009.

[27] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning Deep Features for Discriminative Localization, 2015.

[28] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized Convolutional Neural Networks for Mobile Devices, 2016.

[29] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net : ImageNet Classification Using Binary, 2016.

[30] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, Yorktown Heights, Pritish Narayanan, and San Jose. Deep Learning with Limited Numerical Precision, 2015.

[31] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized Neural Networks. In *30th Conference on Neural Information Processing Systems (NIPS 2016)*, pages 1–9, Barcelona, Spain, 2016.

[32] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[33] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–14, 2015.

[34] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In *Advances in Neural Information Processing Systems*, volume 2017-December, pages 6232–6240, 2017.

[35] Roman Novak, Yasaman Bahri, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-dickstein. Sensitivity and Generalization in Neural Networks: An Empirical Study. In *The International Conference on Learning Representations (ICLR)*, pages 1–21, Vancouver CANADA, 2018. ICLR.

[36] Ravid Shwartz-Ziv, Amichai Painsky, and Naftali Tishby. Representation Compression and Generalization in Deep Neural Networks. In *The International Conference on Learning Representations (ICLR)*, pages 1–15, New Orleans, 2019. ICLR.

[37] Frank Nielsen. A family of statistical symmetric divergences based on Jensen's inequality. *Computing Research Repository - CORR*, (December 2011), 2010.

A Appendix: Example of Trojan Problem

Figure 8 illustrates the problem of traffic sign classification in a presence of a trojan. An adversary with access to training data could embed some trojans into the training collection. For example, a yellow region added to the stop sign in Figure 8 (upper left) will change the classification outcome of the stop sign into a speed limit 65. The yellow region is considered as a trojan (or trigger) embedded in a class stop sign and re-assigning the images with trojan from class A (stop sign) to class B (speed limit 65).

We are primarily focusing on trojans in NNs that cause misclassification during inference and are introduced by an adversary and not by a poor NN model performance. In order to achieve adversary misclassification, the trojan embedding must not change NN accuracy evaluated by using data without trojans. The minimum loss of accuracy during trojan embedding depends on:

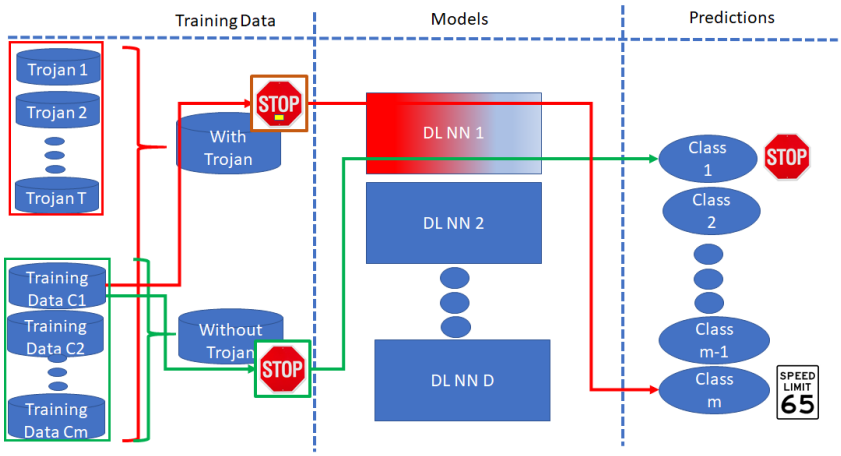

Figure 8: Trojan problem for traffic sign classification.

1. the number of classes per dataset,
2. the number of contiguous regions per class,
3. the shape of each region, and
4. the size of each region.

It is assumed that a class can occupy multiple disconnected manifolds (multiple contiguous regions in 2D) which is common in classes that contain a diversity of unspecified sub-classes. These dependencies can be simulated in NN Caculator for a fixed number of two classes and nine specific trojan embedding types in 2D datasets.

B Appendix: Related Work

In the context of the IARPA TrojAI challenge [1] applied to traffic sign classification, the goal is to detect models trained without trojan (`TwoT`) and trained with trojan (`TwT`) based on the analyses of 100 models (3 architectures) designed to predict 5 traffic sign classes. The models are provided with 100 trojan-free example images from each class and the trojan detection must take less than 24 hours on the NIST computational infrastructure. The problem has many variations based on what information and computational resources are available for trojan detection (type of attack, type of model architecture, model coefficients, training data subsets, description of trojans, number of classes to be misclassified by embedding trojans, classes that are misclassified by trojans, models that have been trained with trojans, computational complexity limits imposed on the delivered solution, etc.). Other challenges related to TrojAI have already been posed, for example, the Guaranteeing AI Robustness against Deception (GARD) challenge [13]. As of today, none of the challenges can be described in terms of their difficulty level which motivates our work.

The TrojAI challenge models were created with a variety of contiguous regions within a traffic sign defining a trojan (see Appendix A for the problem illustration). In the previous work, the problem of trojans in AI has been reported from the view point of detecting trojans [14] [15], constructing trojan attacks [16], defending against trojans [17], and bypassing trojan detectors [18]. The problem of trojan presence is often related to the efficiency (or utilization) of DL NNs as introduced in the early publications about optimal brain [19] and optimal brain surgeon [20]. A few decades later, the topics of pruning links and trimming neurons are being explored in [21], [22], and [23] to increase an efficiency of Deep Learning (DL) NNs and to decrease NN model storage and computational requirements of model training. Our work is motivated by the past concepts of NN efficiency. However, our goal is to explore the hypothesis that NN models trained with trojans will demonstrate higher efficiency/utilization of NN than NN models trained without trojan. In comparison to previous work, our approach is focused on reliable measurements in the context of trojan detection and is investigating questions about where trojans are encoded. We assume that the models `TwoT` and `TwT` are neither under-fitted nor over-fitted [24].

Table 1: Trojan embedding characteristics

Trojan embedding	Reference dataset	Num. per class	Num. per region	Shape	Size	Location per region
T1	Circle	1 orange	1	circle	π	$[Center : [0, 0], r = 1.0]$
T2	Circle	1 orange	1	circle	2.25π	$[Center : [0, 0], r = 1.5]$
T3	Exclusive OR	1 orange	1	square	4	$[x = 1.5, y = 3.5, w = 2, h = 2]$
T4	Exclusive OR	1 orange	1	square	4	$[x = 2.5, y = 4.5, w = 2, h = 2]$
T5	Exclusive OR	1 blue	1	square	4	$[x = -3.5, y = 3.5, w = 2, h = 2]$
T6	Exclusive OR	2 orange	1	square	4 per region	$[x = 1.5, y = 2.5, w = 2, h = 2]$ $[x = -3.5, y = -1.5, w = 2, h = 2]$
T7	Gaussian	1 in each class	1	circle	π per class	$[Center : [2, 2], r = 1]$ $[Center : [-2, -2], r = 1]$
T8	Spiral	4 orange	4	curve	7.33 (orange)	$\lvert x - y\rvert/\sqrt{2} < 1.0$
T9	Spiral	4 in each class	4	curve	7.33 (orange) 12.31 (blue)	$\lvert x - y\rvert/\sqrt{2} < 1.0$

Table 2: Definition of KL divergence

$p_{ij} \setminus \ q_{ij}$	$q_{ij} = 0$	$q_{ij} \neq 0$
$p_{ij} = 0$	0	not defined
$p_{ij} \neq 0$	0	defined

The problem of gaining insights about DL NNs has been approached by (1) mathematical modeling [6] (network layers), [7] (activation functions), [8] (wavelets), (2) feature and network visualizations [25] (across layers), [26](higher layers), [27] (discriminative features),[9] (fully connected layers at small scale), and (3) limited numerical precision of modeling to achieve 'interactive' response [28](quantized NN for mobile devices), [29] (binary weights for ImageNet), [30] (tradeoffs), [31] (binary NNs). Many insights are pursued with respect to representation learning [32], expressiveness [33], [34], and sensitivity and generalization (under- and over-fitting NN models) [35], [36]. From all past work, we leveraged the mathematical framework in [6], visualization called Tensorflow Playground in [9], and efficiency and expressiveness concepts in [34].

C Appendix: Characteristics of Trojan Embedding

NN Calculator contains a slider bar for embedding trojans. Nine trojans are illustrated in Figure 2. Table 1 summarizes the details of those nine trojans as used in multiple datasets. The details provide deeper understanding about correlations between inefficiency measurements and the trojan embedding characteristics.

D Appendix: Additional Formulas for KL Divergence

Definition of KL divergence: Table 2 presents the theoretical definition of KL divergence with respect to input probabilities q_{ij} and p_{ij}.

Derivation of modified KL divergence: We derived modified KL divergence from KL divergence definition as shown in Equation 5. The approximation takes place when we assume that $p_{ij} = \frac{m}{n}, \forall i \in Set(q_{ij} \neq 0)$. The last simplification uses the fact that $\sum_{i \in Set(q_{ij} \neq 0)}(q_{ij}) = 1$.

$$
\begin{aligned}
D_{KL}(Q_j \parallel P_j) = \sum_{i=1}^{n}(q_{ij} * \log_2 \frac{q_{ij}}{p_{ij}}) = \\
= \sum_{i=1}^{n}(q_{ij} * \log_2 q_{ij}) - \sum_{i=1}^{n}(q_{ij} * \log_2 p_{ij}) = \\
\sum_{i \in Set(q_{ij} \neq 0)}(q_{ij} * \log_2 q_{ij}) - \sum_{i=1}^{n}(q_{ij} * \log_2 p_{ij}) \approx \\
\approx \sum_{i \in Set(q_{ij} \neq 0)}(q_{ij} * \log_2 q_{ij}) - \log_2 \frac{m}{n} * \sum_{i \in Set(q_{ij} \neq 0)}(q_{ij}) = \\
= \sum_{i \in Set(q_{ij} \neq 0)}(q_{ij} * \log_2 q_{ij}) - \log_2 \frac{m}{n} = \widehat{D_{KL}}(Q_j \parallel P_j)
\end{aligned}
\tag{5}
$$

Average ratio of deltas for T1 and T2: We can relate the trojans T1 and T2 via their size since the location is the same. As documented in Section 3, there is a relationship between the trojan size change and $\Delta(P)$ and $\Delta(N)$ changes. Equation 6 documents how we computed the average ratio of deltas for each class for the NN with 6 hidden layers (plus the output) and 8 nodes per layer.

$$
\begin{aligned}
\overline{Ratio}(N) = \frac{1}{7}\sum_{l=0}^{6}\frac{\widehat{D^l_{KL}}(TwoT(2)/N) - \widehat{D^l_{KL}}(TwT(2)/N)}{\widehat{D^l_{KL}}(TwoT(1)/N) - \widehat{D^l_{KL}}(TwT(1)/N)} = 2.24 \\
\overline{Ratio}(P) = \frac{1}{7}\sum_{l=0}^{6}\frac{\widehat{D^l_{KL}}(TwoT(2)/P) - \widehat{D^l_{KL}}(TwT(2)/P)}{\widehat{D^l_{KL}}(TwoT(1)/P) - \widehat{D^l_{KL}}(TwT(1)/P)} = 2.37
\end{aligned}
\tag{6}
$$

E Appendix: Properties of Modified KL Divergence

Property for Increasing Amount of Added Noise: Figure 9 shows the decreasing values of inefficiency for both class labels and at all layers of NN. The negative values for layer 2 and class N (labeled as 2-N in Figure 9, right)) indicate that the network has insufficient capacity for encoding the noisy input data.

Property for Increasing Number of Nodes: Figure 10 illustrates the increasing values of inefficiency for both class labels at layer 0 and equal to a constant 1 at layer 1. The last layer 2 verifies that the NN was trained to a high accuracy and therefore its value is always 0.

Property for Increasing Number of Layers: We varied the number of layers from 1 to 5 layers while keeping the same number of 4 nodes per layer and 2 feature inputs $X1$ and $X2$ as illustrated in Figure 11 (left). While retraining the same NN three times, we computed average and standard deviation of the modified KL divergence values per layer and per class.

Figure 11 (top right) shows the average inefficiency per layer and class as the number of layers is increasing. The last layers in each NN are associated with higher inefficiency values (diagonal values) but one cannot unequivocally confirm increasing inefficiency with the increasing number of layers. The average of average inefficiencies across all layers is $1.48, 1.667, 1.864, 1.683$ and 2.054 for NNs with the number of layers equal to $1, 2, 3, 4$, and 5 respectively. This numerical sequence, as well as similar sequences computed for each class label separately, also indicate that comparing models with different architectures must be performed at the state level as opposed to at the layer statistics level (i.e., KL divergence).

Figure 11 (bottom right) quantifies the standard deviation associated with the three retraining runs. The average of standard deviations across all NN layers is $0.092, 0.089, 0.098, 0.073$, and 0.069 for NNs with the number of layers equal to $1, 2, 3, 4$, and 5 respectively. These averages are lower than

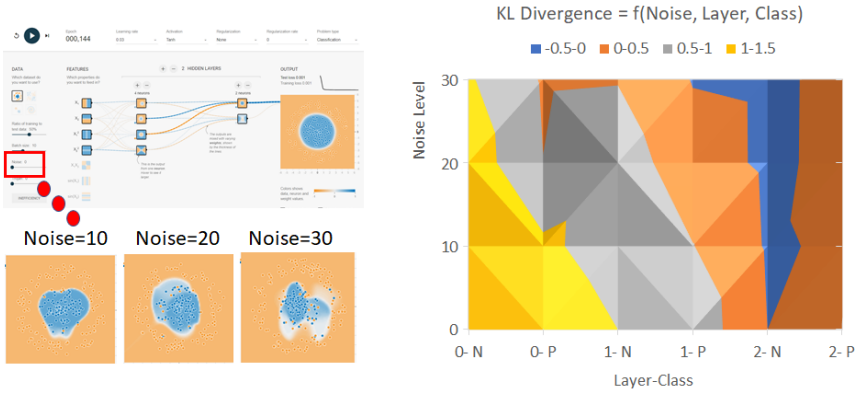

Figure 9: Inefficiency property as a function of added noise. If noise is added to training data as shown in the left bottom, then inefficiency (modified KL divergence) goes down for the same neural network architecture shown in the left top. The right plot shows the dependency of inefficiency on noise level per class and layer.

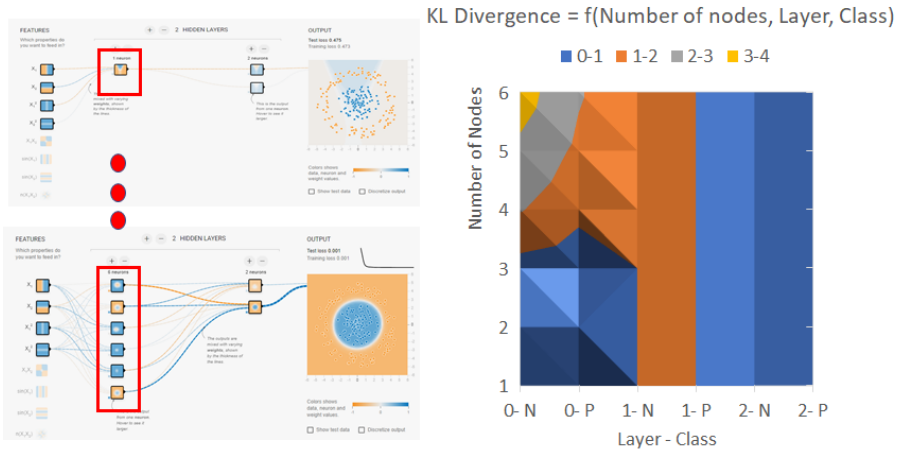

Figure 10: Inefficiency property as a function of added number of nodes per layer (right). If nodes are added to a NN layer (left bottom), then inefficiency (modified KL divergence) goes up for the input dataset (circle in left top).

the average value 0.364 shown in Figure 5 for retraining the dataset Circle. The differences are due to the different NN capacities as documented by much smaller average inefficiencies of the compared NNs here than the average inefficiency of 5.318 in the case of a NN with 7 hidden layers and 8 nodes per layer. These comparisons assume that each model was trained to reach the same classification accuracy.

F Appendix: Additional Comparisons of Trojans

Comparison of trojans with location shift (T3 and T4): The placement of T4 caused for the NN to become instable. We observed that even after more than 2000 epochs, the accuracy could not reach close to $100\,\%$ as illustrated in Figure 12. This is confirmed by computing negative modified KL divergence values which indicate that the NN model TwT is insufficient to represent the training data. As a consequence, the fluctuation of inefficiency values is larger than in a stable well-trained model. This illustrates that adversaries also face a challenge when choosing the characteristics of embedded trojans in order to conceal them by achieving close to $100\,\%$ classification accuracy.

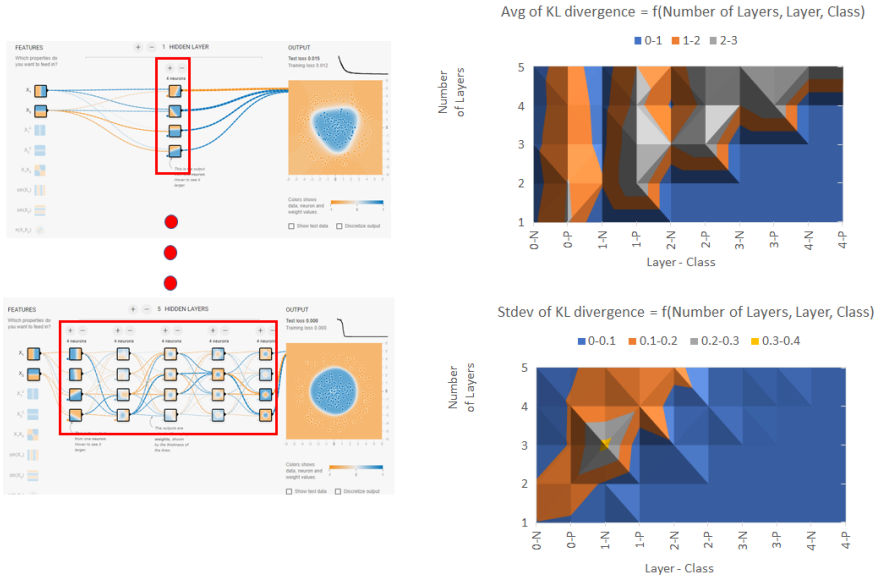

Figure 11: Inefficiency property as a function of added number of layers combined with sensitivity to model retraining. Average and standard deviation of KL divergence per layer and per class are computed from three training runs with 100 epochs per run.

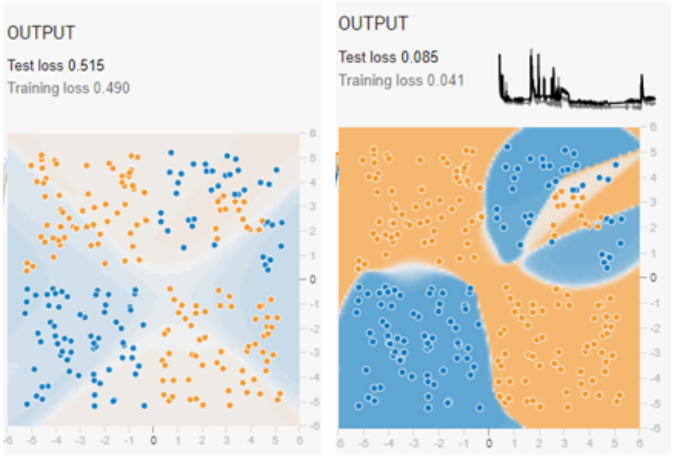

Figure 12: Instability of training models `TwT` and embedded trojan T4 with horizontal shift of a location within a class region with respect to T3. Left - initial dataset. Right - training result after more than 2000 epochs.

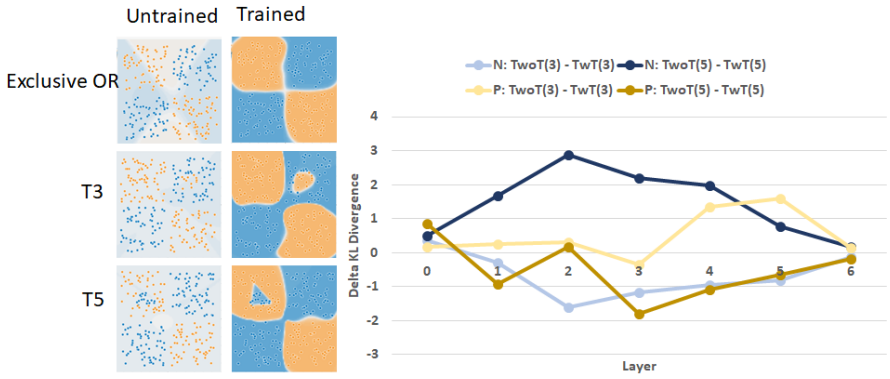

Figure 13: Comparison of inefficiencies between models `TwoT` and `TwT`, and embedded trojans T3 in class P and T5 in class N of the same approximate size within one class region.

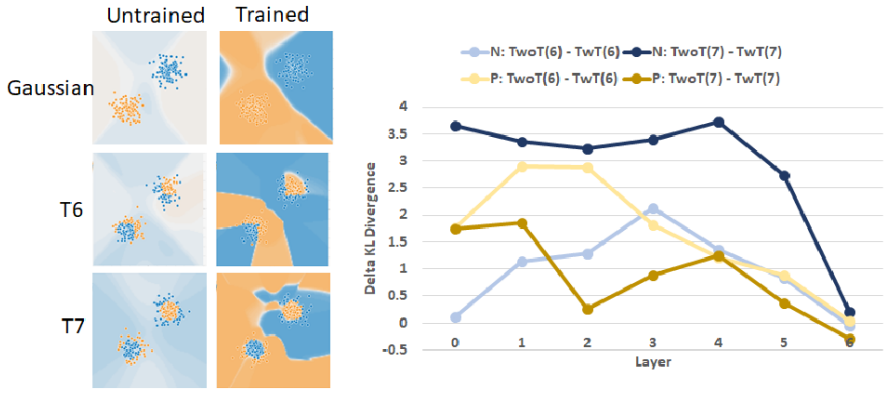

Figure 14: Comparison of inefficiencies between models `TwoT` and `TwT`, and embedded trojans T6 and T7 with square and circle shapes.

Comparison of trojans embedded in different classes (T3 and T5): The trojans T3 and T5 are symmetric in terms of their embedding in class P (blue region) or class N (orange region). We observe this symmetry in Figure 13 as the deltas have inverse signs for classes P and N ($\Delta(T6/P) > \Delta(T6/N)$ and $\Delta(T7/P) < \Delta(T7/N)$ except for layer 0). While the chosen locations for embedding trojans T3 and T5 can yield close to $100\,\%$ classification accuracy, the models heavily depend on the NN initialization. Therefore, we did not compare the inefficiency across the two trojans.

Comparison of trojans with varying shape (T6 and T7): Figure 14 summarizes the delta between modified KL divergence values of models `TwoT` and models `TwT` for the trojans T6 and T7 of different shapes (circle and square) and embedded into both P and N classes. All deltas are positive for both classes and for all layers except from the last layer (T6, Class N: $\delta = -0.047$ and T7, Class P: $\delta = -0.284$). Following Figure 4, this implies that the trojan is in both classes. The values in the last layer indicate that the model TwT had a hard time accurately encoding the trojan.

It is difficult to infer anything about the trojan shapes from Figure 14(right) because the delta curves depend on the very many possible spatial partitions of the 2D space to classify training data points accurately. Nonetheless, we can infer from Figure 14 (right) that the spatial partition allocated for the class P in a model TwT T6 is larger than the in a model TwT T7 (i.e., $\widehat{D_{KL}}(TwoT(6)/P) - \widehat{D_{KL}}(TwT(6)/P) > \widehat{D_{KL}}(TwoT(7)/P) - \widehat{D_{KL}}(TwT(7)/P)$). This can be visually confirmed for class P (blue) in Figure 14(left) as the model TwT T6 occupies a larger partition than in the model TwT T7 (i.e., blue area is larger in Figure 14 left middle then in left bottom). A similar inference can be derived for class N as $\widehat{D_{KL}}(TwoT(6)/N) - \widehat{D_{KL}}(TwT(6)/N) < \widehat{D_{KL}}(TwoT(7)/N) - \widehat{D_{KL}}(TwT(7)/N)$.

G Appendix: Discussion about Trojan Detection

Entropy-based measurements from state histograms: One option to incorporate the computational constraints and remove the need for histogram alignment would be to replace KL divergence by entropy of a state histogram normalized by maximum entropy per layer and per class label. However, the use of normalized entropy would run into the same issue of negative measurement values as in KL divergence while limiting the dynamic range of measurements.

If one would always evaluate a pair of models (i.e., comparing models `TwoT` and `TwT` for trojan detection), then one could use Jensen–Shannon divergence [37] instead of KL divergence since it is symmetric and yields always a finite value. We preferred the KL divergence because evaluating one NN is more general than evaluating pairs of NNs.

Trojan detection algorithm: One can obtain several additional useful insights from interactive analyses in NN Calculator before designing a trojan detection algorithm. Some of them are presented in Appendix F. In many of the results, it is apparent that the encoded class information is not in one layer but spread across multiple layers. Thus, trojan detection must include comparisons of vectors of $\widehat{D^l_{KL}}$ across all layers l. Furthermore, the encoding of the same training data in NN can have multiple solutions, especially in inefficient NN and therefore the comparison of vectors of $\widehat{D^l_{KL}}$ must include again a statistical nature of such solutions. Finally, the last layers carry less information about trojans because they serve the purpose of a final decision maker which should appear fair for datasets without trojans. This could be accommodated by weighting the layer-specific vector elements. From a global algorithmic design perspective, designing an actual trojan detector must still consider the trade-offs of doing all pair-wise model comparisons versus clustering all vectors of $\widehat{D^l_{KL}}$ to identify the cluster of model TwoT.

Complexity of trojan problems: Given the current trojan detection approach, the complexities of trojan problems arise in the relationships between model capacity, size of input data space, characteristics of trojan embedding, and the number and selection of provided training data points per class with respect to the within-class variability (i.e., number, shape, and location of regions per class). As one transitions analyses from NN Calculator to actual DL NNs, the model capacity goes from ten to thousands of features, from six to hundreds of hidden layers, and from eight to hundreds of nodes per layer. The size of input data space goes from 2D space constrained by 12 units x 12 units to grayscale and color images with millions of pixels with constrained variability by the application domain. Given such an increase of problem complexities and without knowing the characteristics of trojan embedding, the number and selection of provided training data points per class become the key to detecting trojans.